Multi-Agent Programming

Olivier Boissier[1], Rafael H. Bordini[2], Mehdi Dastani[3], Jomi F. Hübner[4], Alessandro Ricci[5]

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

[2]INF-UFRGS, Brazil
R.Bordini@inf.ufrgs.br

[3]Utrecht University, Netherlands
mehdi@cs.uu.nl

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

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

European Agent Systems Summer School
23–27 August, 2010 — Saint Etienne, France

Outline of the MAP Course

Introduction

Agent-Oriented Software Development Methodology

Multi-agent systems are a development in software engineering resulting in a new paradigm

Requirement ⇒ Analysis ⇒ Design ⇒ Implementation ⇒ Test

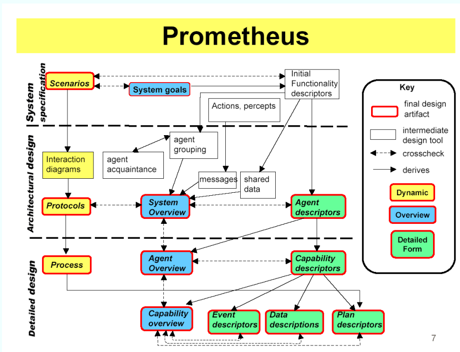
The aim is to provide high-level abstraction to model and develop complex systems

- Structural analysis methodology
- Object-oriented methodology
- Agent-oriented methodology (e.g. Gaia, Prometheus)

Abstraction in Multi-Agent Systems

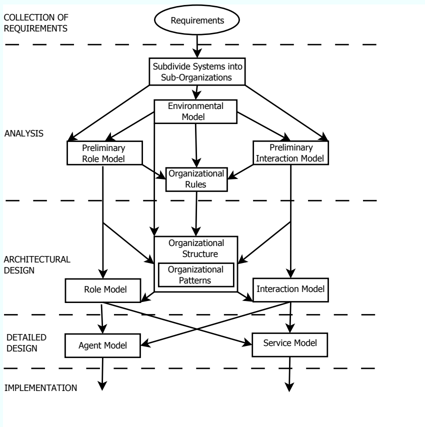
- **Individual Agent Level**: Autonomy, Situatedness
  - **Cognitive concepts**: beliefs, desires/goals, intention/plans
  - **Deliberation and decision**: sense/reason/act, reactive/pro-active
- **Multi-Agent Level**: Social and Organizational Structures
  - **Roles**: functionalities, activities, and responsibilities
  - **Organizational Rules**: constraints on roles and their interactions, norms, deadlines, obligations
  - **Organizational Structures**: topology of interaction patterns and the control of activities
- **Environment**: Resources and Services that MAS can access and control; sensing and acting in an environment

Agent-Oriented Software Engineering: Prometheus



Slide taken from Michael Winikoff

Agent-Oriented Software Engineering: Gaia



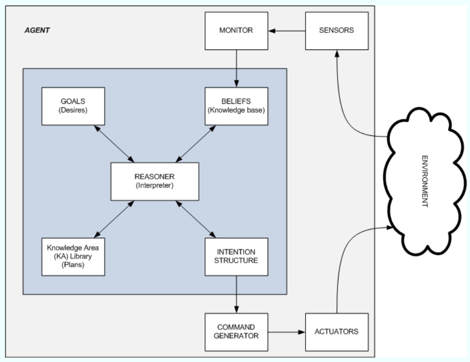
Agent Oriented Programming

- Proposed by Shoham [Shoham, 1993]
- Use of mentalistic notions and a societal view of computation (anthropomorphism)
- Separation of Concerns: Agents – Organisations – Environment
- Programming languages for agents have developed a lot since then, but still not a mature paradigm
- Programming languages/frameworks for organisation and environment are also being developed
- Some agent development platforms have formal basis, others don't

BDI Architecture

- Intentional Stance (Dennett)
- Practical Reasoning (Bratman)
- IRMA (Bratman, Isreal, Pollack)
- PRS (Georgeff, Lansky)
- dMARS (Kinny)
- BDI Logics and Agent Architecture (Rao, Georgeff)
- Wooldridge, Singh, ...

PRS Architecture



Programming Languages for Cognitive Agents

Programming Languages for Multi-Agent Systems
Data Structures + **Programming Instructions**
E.g., 2APL, Jason, Jadex, JACK, GOAL

- **Data Structures** to represent agent mental state
  - **Beliefs**: General and specific information available to agent
  - **Goals**: Objectives that agent want to reach
  - **Events**: Observations of (environmental) changes
  - **Capabilities**: Actions that agent can perform
  - **Plans**: Procedures to achieve objectives
  - **Reasoning rules**: Reason about goals and plans
    - goal → plan
    - events → plan
    - plan → plan

Programming Languages for Cognitive Agents

- **Programming Instructions** to process mental states
  - Generate Plans for Received Events
  - Generate Plans for Goals
  - Process Exceptions and Handle Failures
  - Repair Plans
  - Select Plans for Execution
  - Execute Plans

Agent Interpreter or **Agent Deliberation** is a loop consisting of such instructions. The loop determines the behaviour of the agent.

Programming Languages/frameworks for Organisations

- **Data Structures** to represent the state of organisation
  - Agents, Roles, Groups
  - Norms, Obligations, Prohibitions, Permissions, Violations
  - Dependency, Power, Delegation, Information relations
  - Deadlines, Sanctions, Rewards
  - Agent Management System, Directory Facilitator
- **Programming Instructions** to control and coordinate agents' behaviours
  - Endogenous: The control is a part of the agent program
  - Exogenous: The control is performed by an external program
    - Monitoring Agents' Behaviors
    - Enforcing Organisational Laws and Rules
    - Regimenting Organisational Laws and Rules

Programming Languages/frameworks for Environments

- **Data Structures** to represent the state of the environment
  - Data bases, Services, Data types, Artifacts
- **Programming Instructions** to process sense and act operations
  - Realising action effects
  - Providing events and sense information
  - Synchronising actions
  - Processing Artifact Operations

AOP

Outline

About 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., 2007b, 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., 2007c], Jadex [Pokahr et al., 2005], 2APL [Dastani, 2008a], 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

Some Shortfalls

- **IDEs** and programming tools are still not anywhere near the level of OO languages
- **Debugging** is a serious issue — much more than "mind tracing" is needed
- Combination with **organisational** models is very recent — much work still needed
- Principles for using **declarative goals** in practical programming problems still not "textbook"
- Large applications and **real-world** experience much needed!

Some Trends I

- **Modularity** and encapsulation
- **Debugging** MAS is hard: problems of concurrency, simulated environments, emergent behaviour, mental attitudes
- Logics for Agent Programming languages
- Further work on combining with interaction, environments, and organisations
- We need to put everything together: rational agents, environments, organisations, normative systems, reputation systems, economically inspired techniques, etc.

$\rightsquigarrow$ **Multi-Agent Programming**

Research on Multi-Agent Systems...

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

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. Hbner** 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

Main Architectural Components

Belief base: where beliefs are stored

Set of events: to keep track of events the agent will have to handle

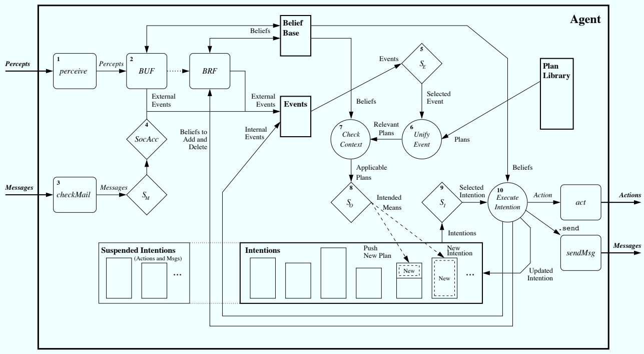
Plan library: stores all the plans currently known by the agent

Set of Intentions: each intention keeps track of the goals the agent is committed to and the courses of action it chose in order to achieve the goals for one of various foci of attention the agent might have

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		Arithmetic operators	
&	(and)	+	(sum)
|	(or)	-	(subtraction)
not	(not)	*	(multiply)
=	(unification)	/	(divide)
>, >=	(relational)	**div**	(divide – integer)
<, <=	(relational)	**mod**	(remainder)
==	(equals)	**	(power)
\==	(different)		

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($term_1$, $term_2$, ...)
  - .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) {
         while (! seeGold) {
         }
         while (seeGold) {
            a = selectDirection();

            doAction(go(a));

} } }  }
```

(how to code the charge battery behaviour?)

Jason × Java III

Example (Java code – charge battery)

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

```
+see(gold)
   <- !goto(gold).
+!goto(gold) :see(gold)       // long term goal
   <- !select_direction(A);
      go(A);
      !goto(gold).
+battery(low)                  // reactivity
   <- !charge.

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

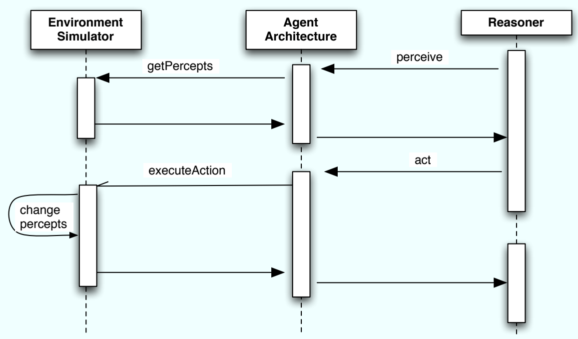
Various communication and execution management infrastructures can be used with *Jason*:

- 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
- Saci: distributed agents, KQML
- ... others defined by the user (e.g. AgentScape)

Definition of a Simulated Environment

- There will normally be an environment where the agents are situated
- The agent architecture needs to be customised to get perceptions and act on such environment
- We often want a simulated environment (e.g. to test an MAS application)
- This is done in Java by extending *Jason*'s Environment class

Interaction with the Environment Simulator



Example of an Environment Class

```
import jason.*;
import ...;
public class robotEnv extends Environment {
  ...
  public robotEnv() {
    Literal gp =
           Literal.parseLiteral("green_patch(souffle)");
    addPercept(gp);
  }

  public boolean executeAction(String ag, Structure action) {
    if (action.equals(...)) {
      addPercept(ag,
           Literal.parseLiteral("location(souffle,c(3,4))");
    }
    ...
    return true;
} }
```


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",
                ...
}
```


MAS Configuration Language III

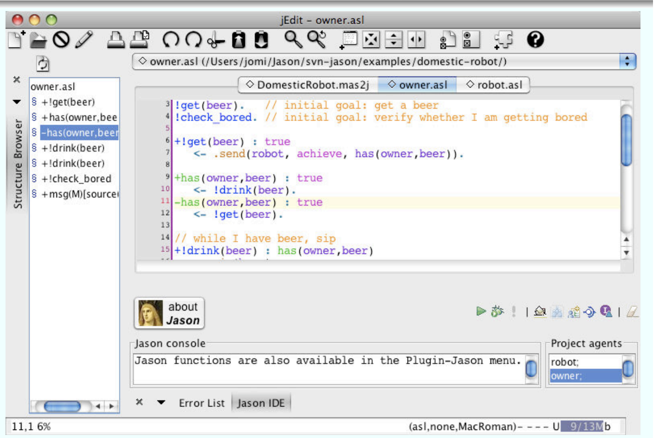
Example (CArtAgO environment)

```
MAS grid_world {

  environment: alice.c4jason.CEnv

  agents:
      cleanerAg
          agentArchClass alice.c4jason.CogAgentArch
          #3;
}
```

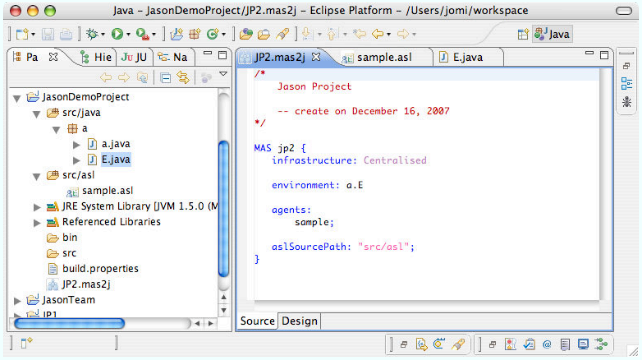

Jason Customisations

- **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, ...)

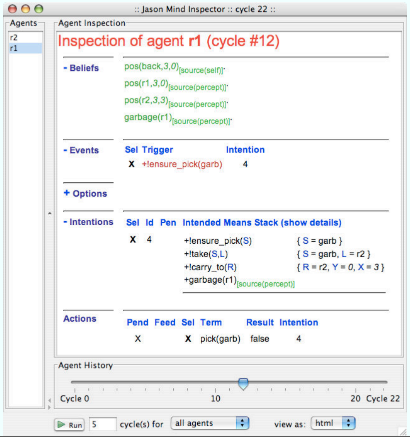
jEdit Plugin



Eclipse Plugin



Mind Inspector



Some Related Projects I

- **Speech-act** based communication
  Joint work with Renata Vieira, Álvaro Moreira, and Mike Wooldridge
- **Cooperative** plan exchange
  Joint work with Viviana Mascardi, Davide Ancona
- **Plan Patterns** for Declarative Goals
  Joint work with M.Wooldridge
- **Planning** (Felipe Meneguzzi and Colleagues)
- **Web and Mobile Applications** (Alessandro Ricci and Colleagues)
- **Belief Revision**
  Joint work with Natasha Alechina, Brian Logan, Mark Jago

Some Related Projects II

- **Ontological** Reasoning
  - Joint work with Renata Vieira, Álvaro Moreira
  - **JASDL**: joint work with Tom Klapiscak
- Goal-Plan Tree Problem (Thangarajah et al.)
  Joint work with Tricia Shaw
- Trust reasoning (ForTrust project)
- Agent verification and model checking
  Joint project with M.Fisher, M.Wooldridge, W.Visser, L.Dennis, B.Farwer

Some Related Projects III

- Environments, Organisation and Norms
  - Normative environments
    Join work with A.C.Rocha Costa and F.Okuyama
  - MADeM integration (Francisco Grimaldo Moreno)
  - Normative integration (Felipe Meneguzzi)
  - **CArtAgO** integration
  - **$\mathcal{M}$oise^{+}** integration
- More on `jason.sourceforge.net`, related projects

Some Trends for *Jason* I

- Modularity and encapsulation
  - Capabilities (JACK, Jadex, ...)
  - Roles (Dastani et al.)
  - Mini-agents (?)
- Recently done: **meta-events**
- To appear soon: annotations for **declarative goals**, improvement in plan failure handling, etc.
- **Debugging** is hard, despite mind inspector, etc.
- Further work on combining with environments and organisations

Summary

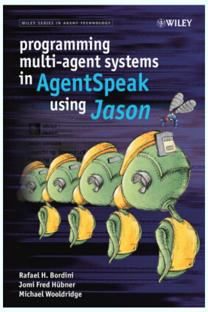
- AgentSpeak
  - Logic + BDI
  - Agent programming language
- *Jason*
  - AgentSpeak interpreter
  - Implements the operational semantics of AgentSpeak
  - Speech-act based communicaiton
  - Highly customisable
  - Useful tools
  - Open source
  - Open issues

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.



2APL: A Practical Agent Programming Language

Motivation

- **Effective and efficient implementation of MAS architectures**
  Individual Cognitive Agents, Shared Environment, Multi-Agent Organisation
- **Support Programming Principles and Techniques**
  Recursion, Compositionality, Abstraction, Exception Handling, Encapsulation, Autonomy, Reusability, Heterogeneity, Legacy Systems
- **Integrated Development Environment (IDE) for Multi-Agent Systems**
  Editor, Debugging and Monitoring Facility, Support the Development of Individual Agents, Multi-Agent Organisation, and Shared Environment

2APL Features (1)

- **Programming Constructs**
  - **Multi-Agent System** Which and how many agents to create? Which environments?
  - **Individual Agent** Beliefs, Goals, Plans, Events, Messages
- **Programming Principles and Techniques**
  - **Abstraction** Procedures and Recursion in Plans
  - **Error Handling** Plan Failure and their revision by Internal Events, Execution of Critical Region of Plans
  - **Legacy Systems** Environment and External Actions
  - **Encapsulation** 2APL Modules, Including 2APL files in other 2APL files
  - **Autonomy** Adjustable Deliberation Process

2APL Features (2)

- **Integrated Development Environment**
  - 2APL platform is Built on JADE and uses related tools
  - 2APL has an Eclipse Plug-in Editor and other Eclipse support tools.
  - 2APL has tools for monitoring mental attitudes of individual agents, their reasoning and communications
  - Executing in one step or continuous mode
  - Programming of the Deliberation Process

Syntax

2APL Syntax: Programming Multi-Agent System

Example (Agents, Environments, and their Initialisations)

```
<apaplmas>
    <agent name="w1" file="worker.2apl"/>
    <agent name="w2" file="worker.2apl"/>
    <agent name="m" file="manager.2apl">
        <beliefs file="nameWorkers.pl" />
    </agent>

    <environment name="blockworld" file="blockworld.jar">
        <parameter key="gridWidth" value="18"/>
        <parameter key="gridHeight" value="18"/>
    </environment>
</apaplmas>
```


2APL Syntax: Programming Individual Agents

General Scheme

⟨*Program*⟩ ::= { "`Include:`" ⟨*ident*⟩
| "`Beliefupdates:`" ⟨*BelUpSpec*⟩
| "`Beliefs:`" ⟨*beliefs*⟩
| "`Goals:`" ⟨*goals*⟩
| "`Plans:`" ⟨*plans*⟩
| "`PG-rules:`" ⟨*pgrules*⟩
| "`PC-rules:`" ⟨*pcrules*⟩
| "`PR-rules:`" ⟨*prrules*⟩ }

2APL Syntax: Programming Individual Agents

Example (Cleaning Environment and Collecting Gold)

```
Beliefupdates:
    { dirt(X,Y) } PickUpDirt() { not dirt(X,Y) }
    { pos(X,Y)  } goto(V,W)    { not pos(X,Y), pos(V,W) }
Beliefs:
            post(5,5).
            dirt(3,6).
            clean(world) :- not dirt(X,Y).
Goals:
            hasGold(2) and clean(world) ,
            hasGold(5)
PG-rules:
            clean(world) <- dirt(X,Y)  |
                        { goto(X,Y) ; PickUpDirt() }
PC-rules:
            event(goldAt(X,Y)) <- true |
                        { goto(X,Y) ; PickUpGold() }
```


2APL Interpreter: Deliberation Cycle

Repeat

- Apply PG-rules
- For each internal event, find and apply a PR-rule
- For each message, find and apply a PC-rule
- For each external event, find and apply a PC-rule
- Execute one step for each plan

2APL: Modularity

Modularity in BDI-based Agent Programming

- Modularity is an essential principle in structured programming. It structures a computer program in separate modules.
- Modularization can be used for information hiding and reusability.
- Modularization in existing BDI-based Agent programming languages is to structure an individual agent's program in separate modules, each encapsulating cognitive components.

Modularity: Our Vision

- **Roles** are functionalities to handle specific situations. They can be specified in terms of BDI concepts.
- An **agent** profile can be specified in terms of BDI concepts.
- A module represents a BDI state on which it can deliberated. A BDI agent is a deliberation process starting with a BDI state.
- 2APL provides a set of programming constructs to **instantiate modules** and to change the **focus of deliberation** at run time.

Modular 2APL

General Scheme

```
⟨2APL_Module⟩ ::= "Beliefupdates: ⟨BelUpSpec⟩"
                |  "Beliefs:" ⟨belief⟩
                |  "Goals:" ⟨goals⟩
                |  "Plans:" ⟨plans⟩
                |  "PG-rules:" ⟨pgrules⟩
                |  ...
...
⟨plan⟩          ::= ... | ⟨createaction⟩ | ⟨releaseaction⟩ | ⟨moduleaction⟩
...
⟨createaction⟩  ::= "create(" ⟨ident⟩ "," ⟨ident⟩ ")"
⟨releaseaction⟩ ::= "release(" ⟨ident⟩ ")"
⟨moduleaction⟩  ::= ⟨ident⟩ "." ⟨maction⟩
...
⟨maction⟩       ::= "execute(" ⟨condition⟩ ")"
                |  "updateBB(" ⟨belief⟩ ")"
                |  "adopt("⟨goal⟩")"
```


Modular 2APL

Example (Exploring Environment and Collecting Gold)

```
Beliefs:
  manager(m).

PC-rules:
  message(A, request, play(explorer)) <- manager(A) |
  {
    create(explorer, myexp);
    myexp.execute( B(gold(POS)) );
    send(A, inform, gold(POS));
    release(myexp)
  }

  message(A, request, play(carrier, POS)) <- manager(A) |
  {
    create(carrier, mycar);
    mycar.updateBB( gold(POS) );
    mycar.execute( B(done) );
    send(A, inform, done(POS))
    release(mycar)
  }
```


Conclusion and Future works

- 2APL provides a variety of distinguished concepts: Beliefs, Goals, Events, Plans, plan repairs, etc.
- 2APL has an complete operational semantics.
- Logics are developed to verify 2APL programs.
- 2APL comes with an implemented framework that facilitates the execution of multi-agent programs.
- 2APL has an Eclipse Plug-in with colored editors and other IDE facilities such as debugging.
- 2APL supports a strong notion of BDI modularity.
- Integrating different Goal Types in the interpreter.

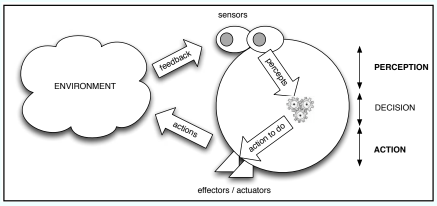
EOP

Outline

Back to the Notion of Environment in MAS

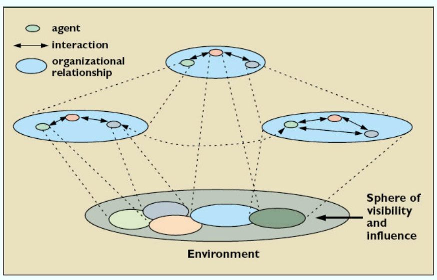
- The notion of environment is intrinsically related to the notion of agent and multi-agent system
  - **"An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective"** [Wooldridge, 2002]
  - **"An 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**

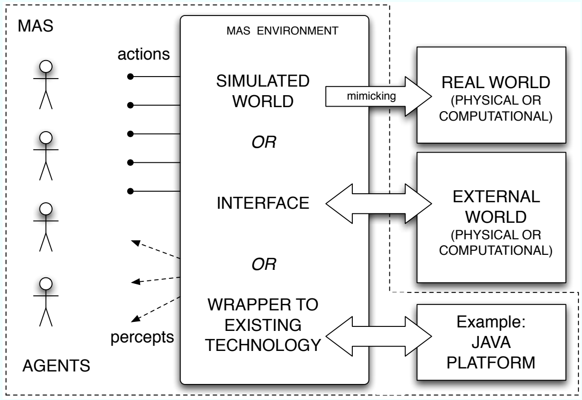
Why Environment Programming

- 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

Environment Programming: General Issues

- Defining the interface
  - actions, perceptions
  - data-model
- Defining the environment computational model & architecture
  - how the environment works
  - structure, behaviour, topology
  - core aspects to face: concurrency, distribution
- Defining the environment programming model
  - how to program the environment

Basic Support Overview

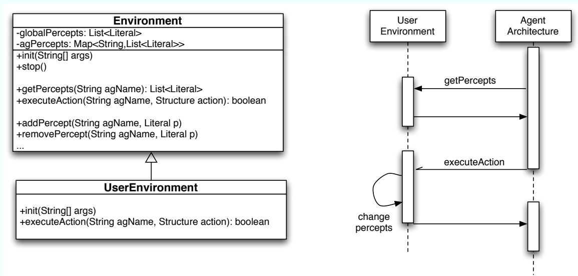


Basic Support: Features

- Environment conceptually conceived as a single monolitic block
  - providing actions, generating percepts
- Environment API
  - to define the set of actions and program actions computational behaviour
    - which include the generation of percepts
  - typically implemented using as single object/class in OO such as Java
    - method to execute actions
    - fields to store the environment state
  - available in many agent programming languages/frameworks
    - e.g., Jason, 2APL, GOAL, JADEX

An Example: *Jason* [Bordini et al., 2007a]

- Flexible Java-based Environment API
  - Environment base class to be specialised
    - executeAction method to specify action semantics
    - addPercept to generate percepts



MARS Environment in *Jason*

```
public class MarsEnv extends Environment {
  private MarsModel model;
  private MarsView  view;

  public void init(String[] args) {
        model = new MarsModel();
        view  = new MarsView(model);
        model.setView(view);
        updatePercepts();
  }

  public boolean executeAction(String ag, Structure action) {
    String func = action.getFunctor();
    if (func.equals("next")) {
      model.nextSlot();
    } else if (func.equals("move_towards")) {
      int x = (int)((NumberTerm)action.getTerm(0)).solve();
      int y = (int)((NumberTerm)action.getTerm(1)).solve();
      model.moveTowards(x,y);
    } else if (func.equals("pick")) {
      model.pickGarb();
    } else if (func.equals("drop")) {
      model.dropGarb();
    } else if (func.equals("burn")) {
      model.burnGarb();
    } else {
      return false;
    }

    updatePercepts();
    return true;
  }
  ...
```

```
  ...

    /* creates the agents perception
     * based on the MarsModel */
  void updatePercepts() {

    clearPercepts();

    Location r1Loc = model.getAgPos(0);
    Location r2Loc = model.getAgPos(1);

    Literal pos1 =  Literal.parseLiteral
        ("pos(r1," + r1Loc.x + "," + r1Loc.y + ")");
    Literal pos2 = Literal.parseLiteral
        ("pos(r2," + r2Loc.x + "," + r2Loc.y + ")");

    addPercept(pos1);
    addPercept(pos2);

    if (model.hasGarbage(r1Loc)) {
      addPercept(Literal.parseLiteral("garbage(r1)"));
    }

    if (model.hasGarbage(r2Loc)) {
     addPercept(Literal.parseLiteral("garbage(r2)"));
    }
  }

  class MarsModel extends GridWorldModel { ... }

  class MarsView extends GridWorldView { ... }
}
```


Jason Agents Playing on Mars

```
// mars robot 1

/* Initial beliefs */

at(P) :- pos(P,X,Y) & pos(r1,X,Y).

/* Initial goal */

!check(slots).

/* Plans */

+!check(slots) : not garbage(r1)
   <- next(slot);
      !!check(slots).
+!check(slots).

+garbage(r1) : not .desire(carry_to(r2))
   <- !carry_to(r2).

+!carry_to(R)
   <- // remember where to go back
      ?pos(r1,X,Y);
      -+pos(last,X,Y);

      // carry garbage to r2
      !take(garb,R);

      // goes back and continue to check
      !at(last);
      !!check(slots).
...
```

```
...

+!take(S,L) : true
   <- !ensure_pick(S);
      !at(L);
      drop(S).

+!ensure_pick(S) : garbage(r1)
   <- pick(garb);
      !ensure_pick(S).
+!ensure_pick(_).

+!at(L) : at(L).
+!at(L) <- ?pos(L,X,Y);
           move_towards(X,Y);
           !at(L).
```

Another Example: **2APL** [Dastani, 2008b]

- 2APL
  - BDI-based agent-oriented programming language integrating declarative programming constructs (beliefs, goals) and imperative style programming constructs (events, plans)
- Java-based Environment API
  - `Environment` base class
  - implementing actions as methods
    - inside action methods external events can be generated to be perceived by agents as percepts

Example: Block-world Environment in **2APL**

```
package blockworld;

public class Env extends apapl.Environment {


  public void enter(String agent, Term x, Term y, Term c){...}

  public Term sensePosition(String agent){...}

  public Term pickup(String agent){...}

  public void north(String agent){...}

  ...

}
```


2APL Agents in the block-world

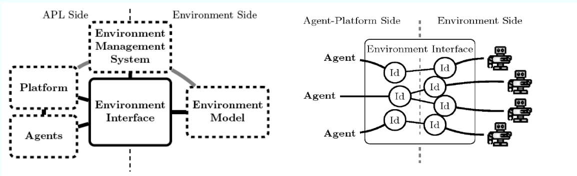
```
BeliefUpdates:
  { bomb(X,Y) }          RemoveBomb(X,Y){ not bomb(X,Y) }
  { true }               AddBomb(X,Y)   { bomb(X,Y) }
  { carry(bomb) }        Drop( )        { not carry(bomb)}
  { not carry(bomb) }    PickUp( )      { carry(bomb) }

Beliefs:
  start(0,1).
  bomb(3,3).
  clean( blockWorld ) :-
     not bomb(X,Y) , not carry(bomb).

Plans:
  B(start(X,Y)) ;
  @blockworld( enter( X, Y, blue ), L )

Goals:
  clean( blockWorld )

PG-rules:
  clean( blockWorld ) <- bomb( X, Y ) |
  {
    goto( X, Y );
    @blockworld( pickup( ), L1 );
    PickUp( );
    RemoveBomb( X, Y );
    goto( 0, 0 );
    @blockworld( drop( ), L2 );
    Drop( )
  }
...
```

```
...

PC-rules:
  goto( X, Y ) <- true |
  {
    @blockworld( sensePosition(), POS );
    B(POS = [A,B]);
    if B(A > X) then
    { @blockworld( west(), L );
      goto( X, Y )
    }
    else if B(A < X) then
    { @blockworld( east(), L );
      goto( X, Y )
    }
    else if B(B > Y) then
    { @blockworld( north(), L );
      goto( X, Y )
    }
    else if B(B < Y) then
    { @blockworld( south(), L );
      goto( X, Y )
    }
  }

  ...
```


Environment Interface Stardard – EIS Initiative

- Recent initiative supported by main APL research groups [Behrens et al., 2010]
  - GOAL, 2APL, GOAL, JADEX, JASON
- Goal of the initiative
  - design and develop a generic environment interface standard
    - a standard to connect agents to environments
    - ... environments such as agent testbeds, commercial applications, video games..
- Principles
  - wrapping already existing environments
  - creating new environments by connecting already existing apps
  - creating new environments from scratch
- Requirements
  - generic
  - reuse

EIS Meta-Model



- By means of the Env. Interface agents perform actions and collect percepts
  - actually actions/percepts are issued to controllable entities in environment model
  - represent the agent bodies, with effectors and sensors

Environment Interface Features

- Interface functions
  - attaching, detaching, and notifying observers (software design pattern);
  - registering and unregistering agents;
  - adding and removing entities;
  - managing the agents-entities-relation;
  - performing actions and retrieving percepts;
  - managing the environment
- Interface Intermediate language
  - to facilitate data-exchange
  - encoding percepts, actions, events

Advanced Support Overview

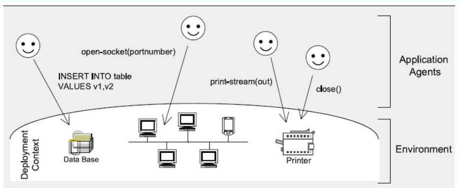
- Vision: environment as a **first-class abstraction** in MAS [Weyns et al., 2007, Ricci et al., 2010b]
  - **endogenous** environments, i.e. that environment which is an explicit part of the MAS
  - providing an exploitable **design** & **programming** abstraction to build MAS applications
- Outcome
  - distinguishing clearly between the responsibilities of agent and environment (...and organisation, see OOP part)
    - separation of concerns
  - improving the engineering practice

Three Support Levels [Weyns et al., 2007]

- Basic **interface** support
- **Abstraction** support level
- **Interaction-mediation** support level

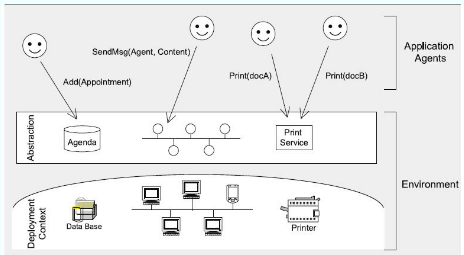
Basic Interface Support

- The environment enables agents to access the deployment context
  - i.e. the hardware and software and external resources with which the MAS interacts



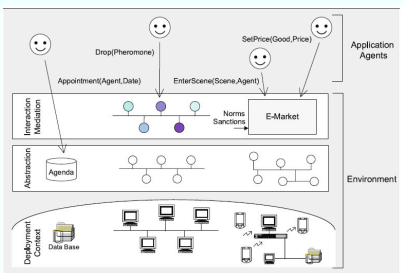
Abstraction Support

- Bridges the conceptual gap between the agent abstraction and low-level details of the deployment context
  - shields low-level details of the deployment context



Interaction-Mediation Support

- **Regulate** the access to shared resources
- **Mediate** interaction between agents



Environment Definition Revised

Environment definition revised [Weyns et al., 2007]

The environment is a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources

Highlights 1/2

- **First-class abstraction**
  - environment as an independent building block in the MAS
  - encapsulating its own clear-cut responsibilities, irrespective of the agents
- The environment provides the **surrounding conditions** for agents to exist
  - environment as an essential part of every MAS
  - the part of the world with which the agents interact, in which the effects of the agents will be observed and evaluated

Highlights 2/2

- Environment as a **glue**
  - on their own, agents are just individual loci of control.
  - to build a useful system out of individual agents, agents must be able to interact
  - the environment provides the glue that connects agents into a working system
- The environment **mediates** both the interaction among agents and the access to resources
  - it provides a medium for sharing information and mediating coordination among agents
    - as a mediator, the environment not only **enables interaction**, it also **constrains it**
    - as such, the environment provides a design space that can be exploited by the designer

Responsibilities 1/3

- **Structuring** the MAS
  - the environment is first of all a shared ?space? for the agents, resources, and services which structures the whole system
- Kind of structuring
  - **physical** structure
    - refers to spatial structure, topology, and possibly distribution, see e.g.,
  - **communication** structure
    - refers to infrastructure for message transfer, infrastructure for stigmergy, or support for implicit communication
  - **social** structure
    - refers to the organizational structure of the environment in terms of roles, groups, societies

Responsibilities 2/3

- Embedding **resources** and **services**
  - resources and services can be situated either in the physical structure or in the abstraction layer introduced by the environment
  - the environment should provide support at the abstraction level shielding low-level details of resources and services to the agents
- Encapsulating a **state** and **processes**
  - besides the activity of the agents, the environment can have processes of its own, independent of agents
    - example: evaporation, aggregation, and diffusion of digital pheromones
  - It may also provide support for maintaining agent-related state
    - for example, the normative state of an electronic institution or tags for reputation mechanisms

Responsibilities 3/3

- **Ruling** and **governing** function
  - the environment can define different types of rules on all entities in the MAS.
    - constraints imposed by the domain at hand or laws imposed by the designer
    - may restrict the access of specific resources or services to particular types of agents, or determine the outcome of agent interactions
    - preserving the agent system in a consistent state according to the properties and requirements of the application domain
- Examples
  - coordination infrastructures
  - e-Institutions

Research on Environments for MAS

- Environments for Multi-Agent Systems research field / **E4MAS** workshop series [Weyns et al., 2005]
  - different themes and issues (see JAAMAS Special Issue [Weyns and Parunak, 2007] for a good survey)
    - mechanisms, architectures, infrastructures, applications [Platon et al., 2007, Weyns and Holvoet, 2007, Weyns and Holvoet, 2004, Viroli et al., 2007]
  - the main perspective is (agent-oriented) software engineering
- Focus of this tutorial: the role of the environment abstraction in **MAS programming**
  - **environment programming**

Environment Programming

- Environment as **first-class programming abstraction** [Ricci et al., 2010b]
  - software designers and engineers perspective
  - **endogenous** environments (vs. exogenous one)
  - programming MAS = programming Agents + programming Environment
    - ..but this will be extended to include OOP in next part
- Environment as **first-class runtime abstraction** for agents
  - agent perspective
  - to be observed, used, adapted, constructed, ...
- Defining computational and programming frameworks/models also for the environment part

Computational Frameworks for Environment Programming: Issues

- Defining the environment interface
  - actions, percepts, data model
  - **contract** concept
- Defining the environment computational model
  - environment structure, behaviour
- Defining the environment distribution model
  - topology

Programming Models for the Environment: Desiderata

- **Abstraction**
  - keeping the agent abstraction level e.g. no agents sharing and calling OO objects
  - effective programming models for controllable and observable computational entities
- **Modularity**
  - away from the monolithic and centralised view
- **Orthogonality**
  - wrt agent models, architectures, platforms
  - support for heterogeneous systems

Programming Models for the Environment: Desiderata

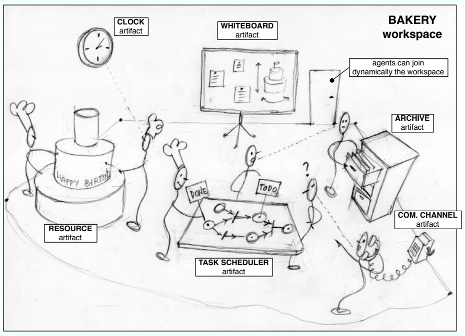
- **Dynamic extensibility**
  - dynamic construction, replacement, extension of environment parts
  - support for open systems
- **Reusability**
  - reuse of environment parts for different kinds of applications

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, 2000a]
- 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., 2010b]
  - introducing a computational notion of artifact to design and implement agent environments

A&A and CArtAgO

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



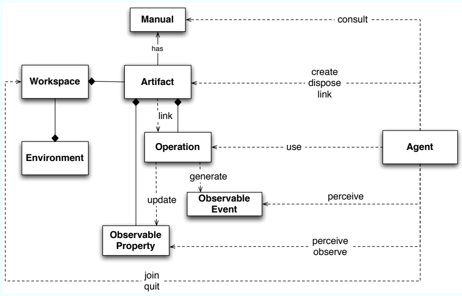
A&A Basic Concepts [Omicini et al., 2008]

- 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

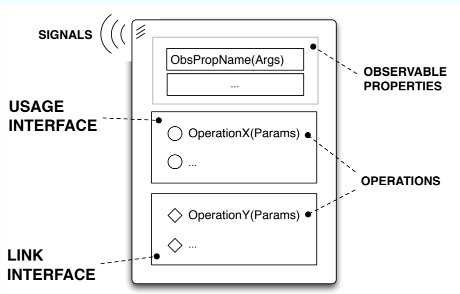
A&A Programming Model Features [Ricci et al., 2007b]

- Abstraction
  - artifacts as first-class resources and tools for agents
- Modularisation
  - artifacts as modules encapsulating functionalities, organized in workspaces
- Extensibility and openness
  - artifacts can be created and destroyed at runtime by agents
- Reusability
  - artifacts (types) as reusable entities, for setting up different kinds of environments

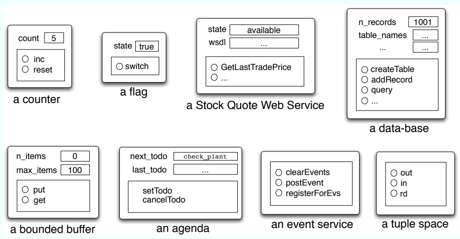
A&A Meta-Model in More Detail [Ricci et al., 2010b]



Artifact Abstract Representation



A World of Artifacts



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** provided by the environment

actions ⟷ artifacts' operation

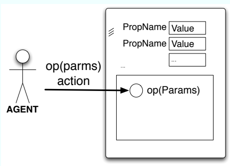
the action repertoire is given by the dynamic set of operations provided by the overall set of artifacts available in the workspace can be changed by creating/disposing artifacts

- action success/failure semantics is defined by operation semantics

percepts ⟷ artifacts' observable properties + signals

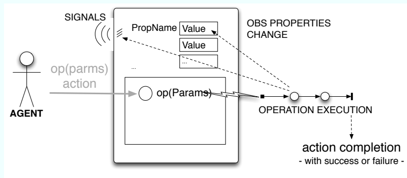
properties represent percepts about the state of the environment signals represent percepts concerning events signalled by the environment

Interaction Model: Use



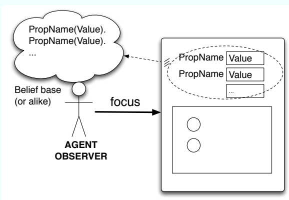
- Performing an action corresponds to triggering the execution of an operation
  - acting on artifact?s usage interface

Interaction Model: Operation execution



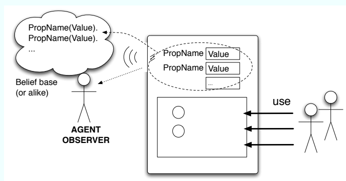
- a process structured in one or multiple transactional steps
- asynchronous with respect to agent
  - **...which can proceed possibly reacting to percepts and executing actions of other plans/activities**
- operation completion causes action completion
  - action completion events with success or failure,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

Interaction Model: Observation



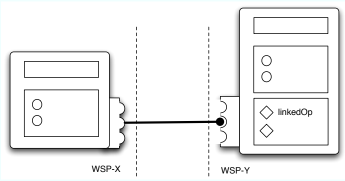
- Agents can dynamically select which artifacts to observe
  - predefined `focus/stopFocus` actions

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

Artifact Linkability



- Basic mechanism to enable inter-artifact interaction
  - **linking** artifacts through interfaces (link interfaces)
    - operations triggered by an artifact over an other artifact
  - Useful to design & program distributed environments
    - realised by set of artifacts linked together
    - possibly hosted in different workspaces

Artifact Manual

- Agent-readable description of artifact's...
  - **...functionality**
    - **what** functions/services artifacts of that type provide
  - **...operating instructions**
    - **how** to use artifacts of that type
- Towards advanced use of artifacts by intelligent agents [Piunti et al., 2008]
  - dynamically choosing which artifacts to use to accomplish their tasks and how to use them
  - strong link with Semantic Web research issues
- Work in progress
  - defining ontologies and languages for describing the manuals

CArtAgO

- Common ARtifact infrastructure for AGent Open environment (CArtAgO) [Ricci et al., 2009b]
- Computational framework / infrastructure to implement and run artifact-based environment [Ricci et al., 2007c]
  - 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`

Integration with Agent Languages and Platforms

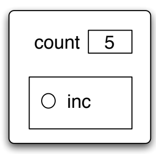
- Integration with existing agent platforms [Ricci et al., 2008]
  - available bridges: *Jason*, Jadex, AgentFactory, simpA, ...
    - ongoing: **2APL**
  - including organisation platforms: *Moise* framework [Hübner et al., 2002b, Hübner et al., 2006]
- Outcome
  - developing open and heterogenous MAS
  - introducing a further perspective on interoperability besides the ACL's one
    - sharing and working in a common work environment
    - common object-oriented data-model

JaCa Platform

- Integration of CArtAgO with *Jason* language/platform
  - a JaCa program is a dynamic set of *Jason* agents working together in one or multiple CArtAgO workspaces
- Mapping
  - actions
    - *Jason* agent external actions are mapped onto artifacts' operations
  - percepts
    - artifacts' observable properties are mapped onto agent beliefs
    - artifacts' signals are mapped as percepts related to observable events
  - data-model
    - *Jason* data-model is extended to manage also (Java) objects

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

Example 1: User and Oberver 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).
```

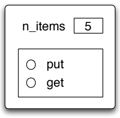
- Working with the shared counter

Pre-defined Artifacts

- Each workspace contains by default a predefined set of artifacts
  - providing core and auxiliary functionalities
  - i.e. a pre-defined repertoire of actions available to agents...
- Among the others
  - `workspace`, type: `cartago.WorkspaceArtifact`
    - functionalities to manage the workspace, including security
    - operations: `makeArtifact`, `lookupArtifact`, `focus`,...
  - `node`, type: `cartago.NodeArtifact`
    - core functionalities related to a node
    - operations: `createWorkspace`, `joinWorkspace`, ...
  - `console`, type `cartago.tools.Console`
    - operations: `println`,...
  - `blackboard`, type `cartago.tools.TupleSpace`
    - operations: `out`, `in`, `rd`, ...
  - ....

Example 2: A Bounded-Buffer Artifact

```
public class BoundedBuffer extends Artifact {
  private LinkedList<Item> items;
  private int nmax;

  void init(int nmax){
    items = new LinkedList<Item>();
    defineObsProperty("n_items",0);
    this.nmax = nmax;
  }

  @OPERATION(guard="bufferNotFull") void put(Item obj){
    items.add(obj);
    getObsProperty("n_items").updateValue(items.size());
  }

  @GUARD boolean bufferNotFull(Item obj){ return items.size() < nmax; }

  @OPERATION(guard="itemAvailable") void get(OpFeedbackParam<Item> res) {
    Item item = items.removeFirst();
    res.set(item);
    getObsProperty("n_items").updateValue(items.size());
  }

  @GUARD boolean itemAvailable(){ return items.size() > 0; }
}
```



- Basic operation features
  - output parameters to represent action feedbacks
  - guards

Example 2: Producers and Consumers

PRODUCERS

```
item_to_produce(0).
!produce.

+!produce: true
  <- !setupTools(Buffer);
     !produceItems.

+!produceItems : true
  <- ?nextItemToProduce(Item);
     put(Item);
     !!produceItems.

+?nextItemToProduce(N) : true
  <- -item_to_produce(N);
     +item_to_produce(N+1).

+!setupTools(Buffer) : true
  <- makeArtifact("myBuffer","BoundedBuffer",
                  [10],Buffer).

-!setupTools(Buffer) : true
  <- lookupArtifact("myBuffer",Buffer).
```

CONSUMERS

```
!consume.

+!consume: true
  <- ?bufferReady;
     !consumeItems.

+!consumeItems: true
  <- get(Item);
     !consumeItem(Item);
     !!consumeItems.

+!consumeItem(Item) : true
  <- .my_name(Me);
     println(Me,": ",Item).

+?bufferReady : true
  <- lookupArtifact("myBuffer",_).
-?bufferReady : true
  <-.wait(50);
     ?bufferReady.
```


Example 3: A Tuple-Space Artifact

```
public class SimpleTupleSpace extends Artifact {
   TupleSet tset;

  void init(){
    tset = new TupleSet();
  }

  @OPERATION void out(String name, Object... args){
    tset.add(new Tuple(name,args));
  }

  @OPERATION void in(String name, Object... params){
    TupleTemplate tt = new TupleTemplate(name,params);
    await("foundMatch",tt);
    Tuple t = tset.removeMatching(tt);
    bind(tt,t);
  }

  @OPERATION void rd(String name, Object... params){
    TupleTemplate tt = new TupleTemplate(name,params);
    await("foundMatch",tt);
    Tuple t = tset.readMatching(tt);
    bind(tt,t);
  }

  @GUARD boolean foundMatch(TupleTemplate tt){
    return tset.hasTupleMatching(tt);
  }

  private void bind(TupleTemplate tt, Tuple t){...}
```

- Multi-step operations
  - operations composed by multiple **transactional** steps, possibly with guards
  - `await` primitive to define the steps

Remarks

- Process-based action execution semantics
  - action/operation execution can be long-term
  - action/operation execution can overlap
- Key feature for implementing coordination functionalities

Example 3: Dining Philosopher Agents

WAITER

```
philo(0,"philo1",0,1).
philo(1,"philo2",1,2).
philo(2,"philo3",2,3).
philo(3,"philo4",3,4).
philo(4,"philo5",4,0).

!prepare_table.

+!prepare_table
  <- for ( .range(I,0,4) ) {
       out("fork",I);
       ?philo(I,Name,Left,Right);
       out("philo_init",Name,Left,Right);
     };
     for ( .range(I,1,4) ) {
       out("ticket");
     };
     println("done.").
```

PHILOSOPHER AGENT

```
!boot.

+!boot
  <- .my_name(Me);
     in("philo_init",Me,Left,Right);
     +my_left_fork(Left); +my_right_fork(Right);
     println(Me," ready.");
     !!enjoy_life.

+!enjoy_life
 <- !thinking; !eating; !!enjoy_life.

+!eating
 <- !acquireRes; !eat; !releaseRes.

+!acquireRes : my_left_fork(F1) & my_right_fork(F2)
  <- in("ticket"); in("fork",F1); in("fork",F2).

+!releaseRes: my_left_fork(F1) & my_right_fork(F2)
 <-  out("fork",F1); out("fork",F2); out("ticket").

+!thinking <- .my_name(Me); println(Me," thinking").
+!eat <- .my_name(Me); println(Me," eating").
```


Example 4: A Clock

CLOCK

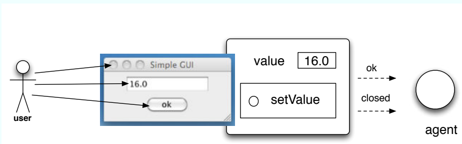
```
public class Clock extends Artifact {

  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**

Example 5: GUI Artifacts



- Exploiting artifacts to enable interaction between human users and agents

Example 5: Agent and User Interaction

GUI ARTIFACT

```
public class MySimpleGUI extends GUIArtifact {
  private MyFrame frame;

  public void setup() {
    frame = new MyFrame();

    linkActionEventToOp(frame.okButton,"ok");
    linkKeyStrokeToOp(frame.text,"ENTER","updateText");
    linkWindowClosingEventToOp(frame, "closed");
    defineObsProperty("value",getValue());
    frame.setVisible(true);
  }

  @INTERNAL_OPERATION void ok(ActionEvent ev){
    signal("ok");
  }

  @OPERATION void setValue(double value){
    frame.setText(""+value);
    updateObsProperty("value",value);
  }
  ...

  @INTERNAL_OPERATION
  void updateText(ActionEvent ev){
    updateObsProperty("value",getValue());
  }

  private int getValue(){
    return Integer.parseInt(frame.getText());
  }

  class MyFrame extends JFrame {...}
}
```

USER ASSISTANT AGENT

```
!test_gui.

+!test_gui
  <-  makeArtifact("gui","MySimpleGUI",Id);
      focus(Id).

+value(V)
  <- println("Value updated: ",V).

+ok : value(V)
  <-  setValue(V+1).

+closed
  <-  .my_name(Me);
      .kill_agent(Me).
```


Remark: 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

Example 6: Action Execution & Blocking Behaviour

```
// agent code

@processing_stream
+!processing_stream : true
  <- makeArtifact(“myStream”,”Stream”,Id);
     focus(Id);
     +sum(0);
     generate(1000);
     ?sum(S);
     println(S).

@update [atomic]
+new_number(V) : sum(S)
  <- -+sum(S+V).
```

```
// artifact code

class Stream extends Artifact {
  ...
  @OPERATION void generate(int n){
    for (int i = 0; i < n; i++){
      signal("new_number",i);
    }
  }
}
```

- The agent perceives and processes `new_number` percepts as soon as they are generate by the `Stream`
  - even if the `processing_stream` plan execution is suspended, waiting for generate action completion
- The test goal `?sum(S)` is executed after `generate` action completion
  - so we are sure that all numbers have been generated and processed

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

A&A and CArtAgO: Some Research Explorations

- Designing and implementing artifact-based organisation Infrastructures
  - ORA4MAS infrastructure [Hübner et al., 2009c]
- Cognitive stigmergy based on artifact environments [Ricci et al., 2007a]
  - Cognitive artifacts for knowledge representation and coordination [Piunti and Ricci, 2009]
- Artifact-based environments for argumentation [Oliva et al., 2010]
- Including A&A in AOSE methodology [Molesini et al., 2005]
- ...

Applying CArtAgO and JaCa

- Using CArtAgO/JaCa for building real-world applications and infrastructures
- Some examples
  - 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`
  - JaCa-Android
    - implementing mobile computing applications on top of the Android platform using JaCa
    - `http://jaca-android.sourceforge.net`

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
  - JaCa case

OOP

Outline

Introduction: Intuitive notions of organisation

- 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 characterized 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**

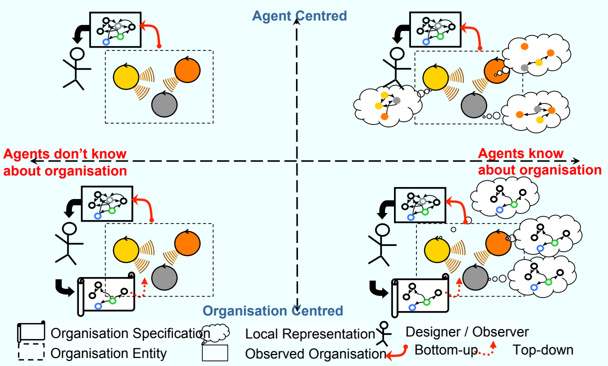
Organisation in MAS

Definition

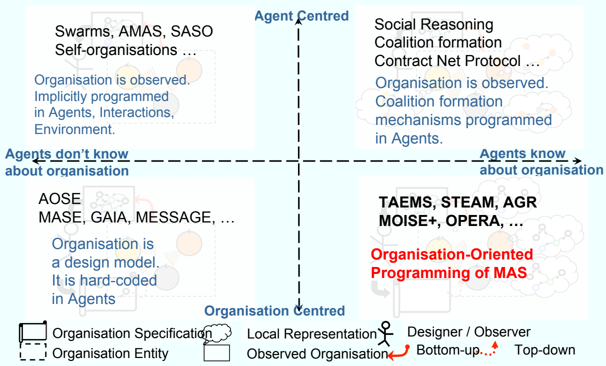
Purposive **supra-agent** pattern of emergent or (pre)defined agents cooperation, that could be defined by the designer or by the agents themselves.

- Pattern of emergent/potential cooperation
  - called **organisation entity**, institution, social relations, commitments
- Pattern of (pre)defined cooperation
  - called **organisation specification**, structure, norms, ...

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



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



Organisation Oriented Programming (OOP)

Organisation as a **first class entity** in the multi-agent eco-system

- Clear distinction between description of the organisation **wrt** agents, **wrt** environment
- Different representations of the organisation:
  - **Organisation specification**
    - partially/totally accessible to the agents, to the environment, to the organisation
  - **Organisation entity**
    - Local representation in the mental state of the agents
      ⇝ possibly inconsistant with the other agents' representations
    - Global/local representation in the MAS
      ⇝ difficulty to manage and build such a representation in a distributed and decentralized setting
- Different sources of actions on (resp. of) the organisation by (resp. on) agents / environment / organisation

Components of OOP: Organisation Modelling Language (OML)

- Declarative specification of the organisation(s)
- Specific constraints, norms and cooperation patterns imposed on the agents
  e.g. AGR [Ferber and Gutknecht, 1998], TeamCore [Tambe, 1997], Islander [Esteva et al., 2001], *M*oise^{+} [Hübner et al., 2002a], ...
- Specific anchors for situating organisations within the environment
  e.g. embodied organisations [Piunti et al., 2009a]

Components of OOP: Organisation Management Infrastructure (OMI)

- **Coordination mechanisms**, i.e. support infrastructure
  e.g. MadKit [Gutknecht and Ferber, 2000b], 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., 2009b], ...

Components of OOP: Integration mechanisms

- **Agent** integration mechanisms
  allow agents to be aware of ant 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, to decide,
  - Action domains of agents, ...
  - Openness, scalability, dynamicity, ...
- 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 the 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 the 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 a 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

Explicit representations, working environments are required so that the agents are able 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

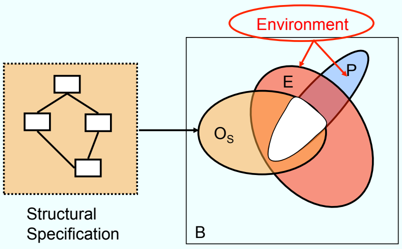
Explicit representations, working environments are required so that the organisation is able to "reason" about itself and about the agents in order to ensure the achievement of its global purpose:

- to decide to let agents enter into/leave from the organisation during execution
  - ⇝ Organisation is no more closed
- to decide to let agents change/adapt the current organisation
  - ⇝ Organisation is no more static and blind
- to govern agents behaviour in the organisation (i.e. monitor, enforce, regiment)
  - ⇝ Organisation is no more a regimentation

AGR [Ferber and Gutknecht, 1998]

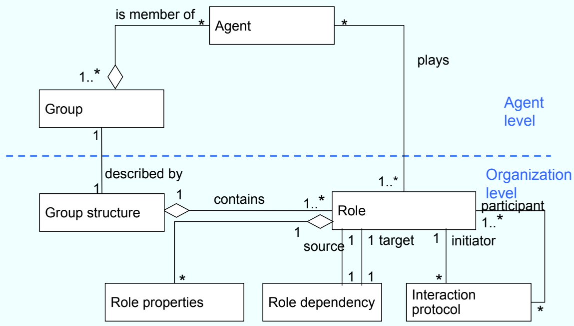
- Agent Group Role, previously known as AALAADIN
  - Agent: Active entity that plays roles within groups. An agent may have several roles and may belong to several groups.
  - Group: set of agents sharing common characteristics, i.e. context for a set of activities. Two agents cant communicate with each other if they dont belong to the same group.
  - Role: Abstract representation of the status, position, function of an agent within a group.
- OMI: the Madkit platform

AGR OML Modelling Dimensions

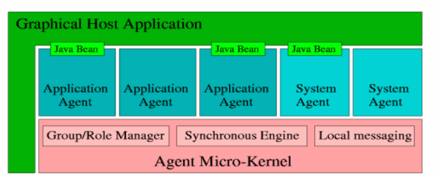


B: agents' possible behaviors
P: agents' behaviors that lead to global purpose
E: agents' possible behaviors constrained by the environment
O_S: agents' possible behaviors structurally constrained by the organization

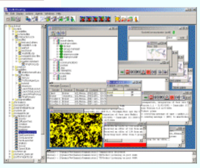
AGR OML



AGR OMI: Madkit



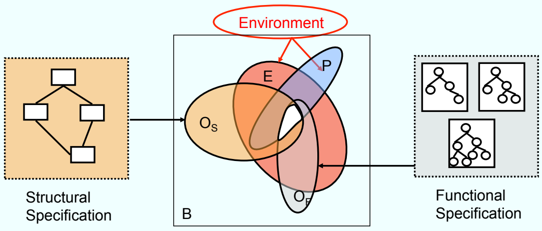
Multi-Agent Development Kit
www.madkit.org



STEAM [Tambe, 1997]

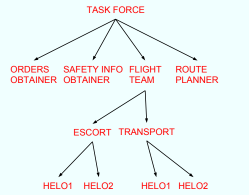
- Shell for TEAMwork is a general framework to enable agents to participate in teamwork.
  - Different applications: Attack, Transport, Robocup soccer
  - Based on an enhanced SOAR architecture and 300 domain independent SOAR rules
- Principles:
  - Team synchronization: Establish joint intentions, Monitor team progress and repair, Individual may fail or succeed in own role
  - Reorganise if there is a critical role failure
  - Reassign critical roles based on joint intentions
  - Decision theoretic communication
- Supported by the TEAMCORE OMI.

STEAM OML Modelling Dimensions

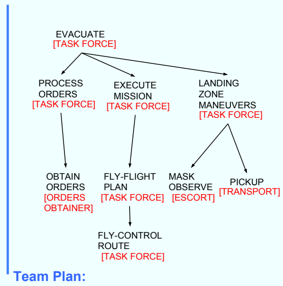


B: agents' possible behaviors
P: agents' behaviors that lead to global purpose
E: agents' possible behaviors constrained by the environment
O_S: agents' possible behaviors structurally constrained by the organization
O_F: agents' possible behaviors functionally constrained by the organization

STEAM OML [Tambe, 1997]

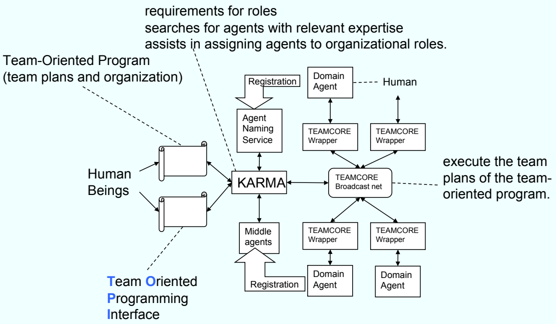


Organization: hierarchy of roles that may be filled by agents or groups of agents.



Team Plan:
- initial conditions,
- term. cond. : achievability, irrelevance, unachievability
- team-level actions.

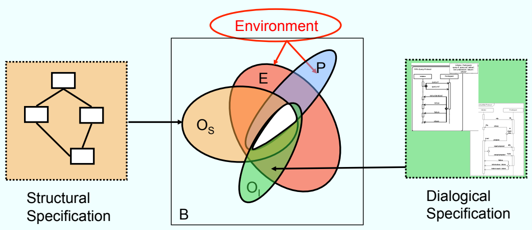
STEAM OMI: TEAMCORE [Pynadath and Tambe, 2003]



ISLANDER

- based on different influences: economics, norms, dialogues, coordination
- ⇝ electronic institutions
- combining different alternative views: dialogical, normative, coordination
- Institution Description Language: Performative structure (Network of protocols), Scene (multi-agent protocol), Roles, Norms
- Ameli as OMI

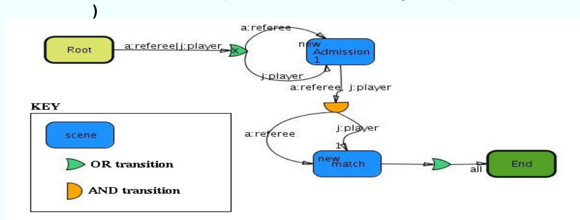
ISLANDER OML Modelling Dimensions



B: agents' possible behaviors
P: agents' behaviors that lead to global purpose
E: agents' possible behaviors constrained by the environment
O_S: agents' possible/permitted/obliged behaviors structurally constrained by the organisation
O_I: agents' possible/permitted/obliged behaviors interactionally constrained by the organisation

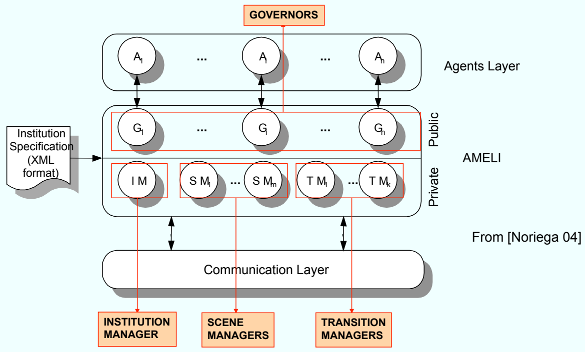
ISLANDER OML: IDL [Esteva et al., 2001]

```
(define-institution
        soccer-server as
        dialogic-framework = soccer-df
        performative-structure = soccer-pf
        norms = ( free-kick coach-messages ... )
)
```



Performative Structure

ISLANDER OMI: AMELI [Esteva et al., 2004]

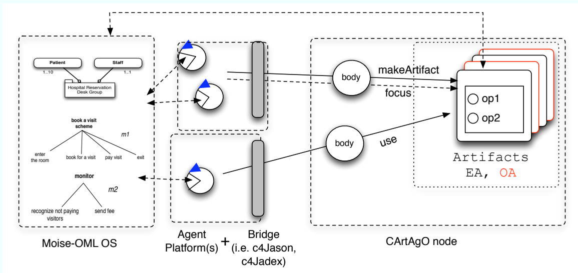


The Moise Framework

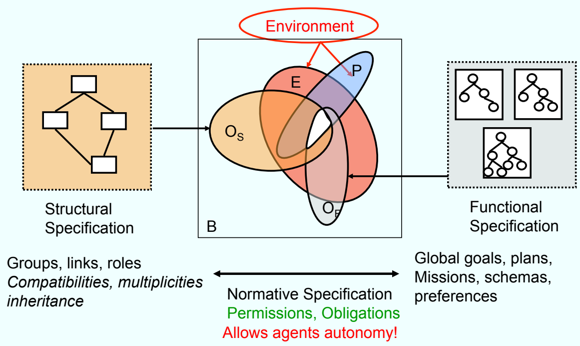
Moise Framework

- Moise Organisation Modelling Language as Tag-based language (issued from Moise [Hannoun et al., 2000], Moise+ [Hübner et al., 2002a], MoiseInst [Gâteau et al., 2005])
- OMI developped as an artefact-based working environment (ORA4MAS [Hübner et al., 2009b] based on CArtAgO nodes) (refactoring of S-Moise+ [Hübner et al., 2006] and Synai [Gâteau et al., 2005])
  - dedicated **organisational artefacts** that provide general services for the agents to work within an organisation
  - **organisational agents** that monitor and manage the functioning of the organisation
- Dedicated integration bridges for
  - Agents and Environment (c4Jason, c4Jadex [Ricci et al., 2009c])
  - Environment and Organisation ([Piunti et al., 2009a])
  - Agents and Organisation (J-Moise+ [Hübner et al., 2007] Moise+ organisational events and actions integration within *Jason*)

Moise Framework as a Concrete Picture of OOP



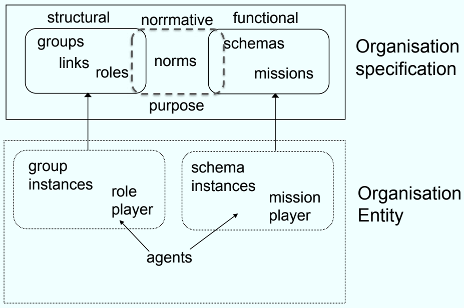
Moise+ Modelling Dimensions



Moise+ 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
    ⇝ J-Moise+ [Hübner et al., 2007]
  - the Organisation Management Infrastructure
    ⇝ ORA4MAS [Hübner et al., 2009b]

Moise OML global picture



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 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

Moise OML Structural Specification

- Defined with the tag structural-specification in the context of an organisational-specification
- One section for definition of all the roles participating to the structure of the organisation (role-definitions tag)
- Specification of the group including all sub-group specifications (groupe-specification tag)

Example

```
 <organisational-specification
   <structural-specification>
      <role-definitions>  ... </role-definitions>
      <group-specification id="xxx">
        ...
      </group-specification>
   </structural-specification>
   ...
</organisational-specification>
```


Role Specification

- Role definition(role tag) in role-definitions section, is composed of:
  - identifier of the role (id attribute of role tag)
  - inherited roles (extends tag) - by default, all roles inherit of the soc role -

Example

```
<role-definitions>
  <role id="player" />
  <role id="coach" />
  <role id="middle"> <extends role="player"/> </role>
  <role id="leader"> <extends role="player"/> </role>
  <role id="r1>
    <extends role="r2" />
    <extends role="r3" />
  </role>
  ...
</role-definitions>
```


Group Specification

- Group definition (group-specification tag) is composed of:
  - group identifier (id attribute of group-specification tag)
  - roles participating to this group and their cardinality (roles tag and id, min, max), i.e. min. and max. number of agents that should adopt the role in the group (default is 0 and unlimited)
  - links between roles of the group (link tag)
  - subgroups and their cardinality (sub-groups tag)
  - formation constraints on the components of the group (formation-constraints)

Example

```
<group-specification id="team">
   <roles>
        <role id="coach" min="1" max="2"/> ...
   </roles>
   <links> ... </links>
   <sub-groups> ... </sub-groups>
   <formation-constraints> ... </formation-constraints>
</group-specification>
```


Link Specification

- Link definition (link tag) included in the group definition is composed of:
  - role identifiers (from, to)
  - type (type) with one of the following values: authority, communication, acquaintance
  - scope of the link (scope): inter-group, intra-group
  - validity in sub-groups: if extends-sub-group set to true, the link is also valid in all sub-groups (default false)

Example

```
    <link from="coach"
          to="player"
          type="authority"
          scope="inter-group"
          extends-sub-groups="true" />
```

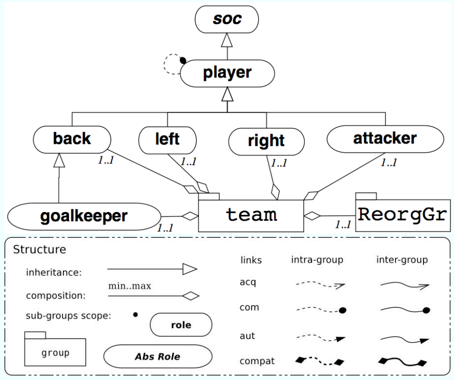
Formation Constraint Specification

- Formation constraints definition (formation-constraints tag) in a group definition is composed of:
  - compatiblity constraints (compatibility tag) between roles (from, to), with a scope, extends-sub-groups and directions (bi-dir)

Example

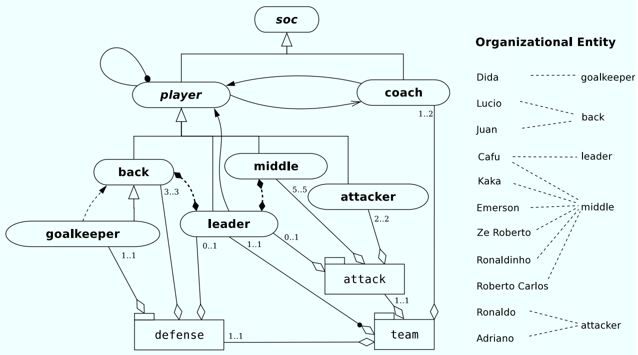
```
<formation-constraints>
  <compatibility from="middle"
                 to="leader"
                 scope="intra-group"
                 extends-sub-groups="false"
                 bi-dir="true"/>
  ...
</formation-constraints>
```


Structural Specification Example (1)



Graphical representation of structural specification of Joj Team

Structural Specification Example (2)

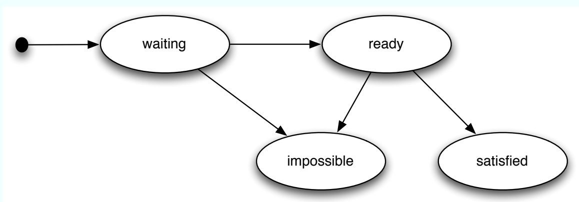


Graphical representation of structural specification of 3-5-2 Joj Team

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

Goal States



- waiting: initial state
- ready: 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 Functional Specification

- Defined with the tag functional-specification in the context of an organisational-specification
- Specification in sequence of the different schemes participating to the expected behaviour of the organisation

Example

```
<functional-specification>
    <scheme id="sideAttack" >
        <goal id="dogoal" > ... </goal>
        <mission id="m1" min="1" max="5">
           ...
        </mission>
        ...
    </scheme>
    ...
</functional-specification>
```

Scheme Specification

- Scheme definition (scheme tag) is composed of:
  - identifier of the scheme (id attribute of scheme tag)
  - the root goal of the scheme with the plan aiming at achieving it (goal tag)
  - the set of missions structuring the scheme (mission tag)
- Goal definition within a scheme (goal tag) is composed of:
  - an idenfier (id attribute of goal tag)
  - a type (achievement default or maintenance)
  - min. number of agents that must satisfy it (min) (default is "all")
  - optionally, an argument (argument tag) that must be assigned to a value when the scheme is created
  - optionally a plan
- Plan definition attached to a goal (plan tag) is composed of
  - one and only one operator (operator attribute of plan tag) with sequence, choice, parallel as possible values
  - set of goal definitions (goal tag) concerned by the operator

Scheme Specification Example

```
<scheme id="sideAttack">
 <goal id="scoreGoal" min="1" >
  <plan operator="sequence">
   <goal id="g1" min="1" ds="get the ball" />
   <goal id="g2" min="3" ds="to be well placed">
     <plan operator="parallel">
       <goal id="g7" min="1" ds="go toward the opponent's field" />
       <goal id="g8" min="1" ds="be placed in the middle field" />
       <goal id="g9" min="1" ds="be placed in the opponent's goal area
     </plan>
   </goal>
   <goal id="g3" min="1" ds="kick the ball to the m2Ag" >
      <argument id="M2Ag" />
   </goal>
   <goal id="g4" min="1" ds="go to the opponent's back line" />
   <goal id="g5" min="1" ds="kick the ball to the goal area" />
   <goal id="g6" min="1" ds="shot at the opponent's goal" />
  </plan>
 </goal>
 ...
```

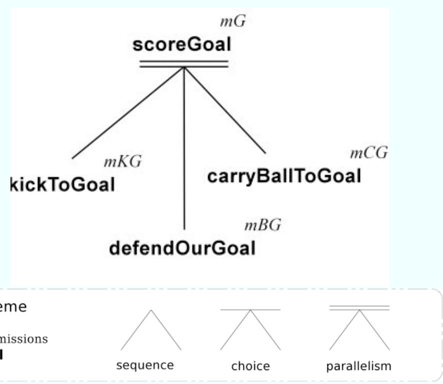

Mission Specification

- Mission definition (mission tag) in the context of a scheme definition, is composed of:
  - identifier of the mission (id attribute of mission tag)
  - cardinality of the mission min (0 is default), max (unlimited is default) specifying the number of agents that can be committed to the mission
  - the set of goal identifiers (goal tag) that belong to the mission

Example

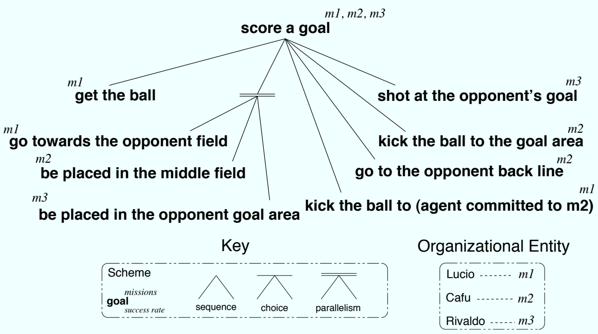
```
<scheme id="sideAttack">
  ... the goals ...
 <mission id="m1" min="1" max="1">
    <goal id="scoreGoal" /> <goal id="g1" />
    <goal id="g3" /> ...
  </mission>
  ...
</scheme>
```


Functional Specification Example (1)



Graphical representation of social scheme for joj team

Functional Specification Example (2)



Graphical representation of social scheme "side_attack" for joj team

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

Moise OML Normative Specification

- Defined with the tag normative-specification in the context of an organisational-specification
- Specification in sequence of the different norms participating to the governance of the organisation

Example

```
<normative-specification>
    <norm id="n1" ... />
    ...
    <norm id="..." ... />
</normative-specification>
```


Norm Specification

- Norm definition (norm tag) in the context of a normative-specification definition, is composed of:
  - the identifier of the norm (id)
  - the type of the norm (type) with obligation, permission as possible values
  - optionally a condition of activation (condition) with the following possible expressions:
    - checking of properties of the organisation (e.g. #role_compatibility, #mission_cardinality, #role_cardinality, #goal_non_compliance)
    - ⇝ unregimentation of organisation properties!!!
    - (un)fulfillment of an obligation stated in a particular norm (unfulfilled, fulfilled)
  - the identifier of the role (role) on which the role is applied
  - the identifier of the mission (mission) concerned by the norm
  - optionally a time constraint (time-constraint)

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

```
<norm id = "n1" type="obligation"
      role="back" mission="m1" time-constraint="1 minute"/>
...
<norm id = "n4" type="obligation"
      condition="unfulfilled(obligation(_,n2,_,_))"
      role="coach" mission="ms" time-constraint="3 hour"/>
...
```

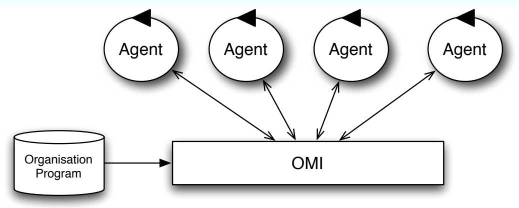

Organisation Entity Dynamics

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)

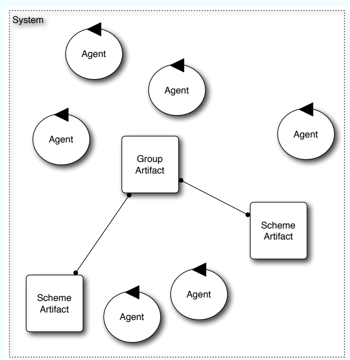
Responsibility

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



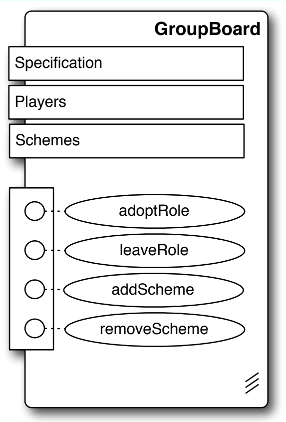
(e.g. MadKit, AMELI, $\mathcal{S}$-Moise$^+$, ...)

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

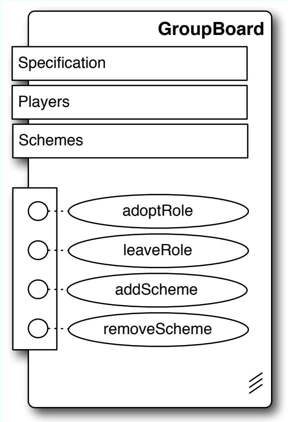
ORA4MAS – GroupBoard artifact



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 – 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 – 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

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

Organisational Artifact Implementation

- Organisational artifacts are programmed with a Normative Programming Language (NPL) [Hübner et al., 2010]
- The NPL **norms** have
  - an activation condition
  - a consequence
- two kinds of consequences are considered
  - regimentations
  - obligations

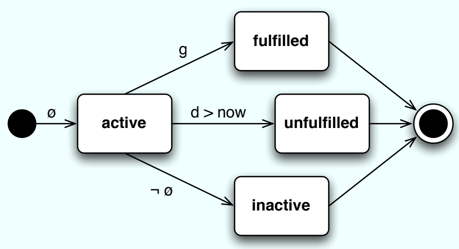
Example (norm)

```
norm n1: plays(A,writer,G) -> fail.
```

or

```
norm n1: plays(A,writer,G)
      -> obligation(A,n1,plays(A,editor,G),
         `now + 3 min`).
```


Obligations life cycle



- ϕ: activation condition (e.g. play a role)
- g: the obligation (e.g. commit to a mission)

OS in Moise OML to NOPL translation

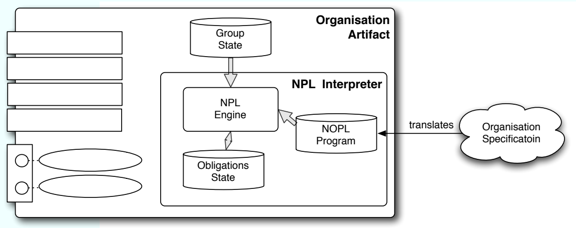
Example (role cardinality norm – regimentation)

```
group_role(writer,1,5).

norm ncar: group_role(R,_,M) &
           rplayers(R,G,V) & V > M
  -> fail(role_cardinality(R,G,V,M)).
```

Example (role cardinality norm – agent decision)

```
norm ncar: group_role(R,_,M) &
           rplayers(R,G,V) & V > M &
           plays(E,editor,G)
  -> obligation(E,ncar,committed(E,ms,_),
              `now + 1 hour`).
```


Organisational Artifact Architecture

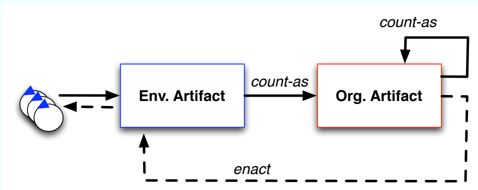


Signals (o = obligation(to whom, reason, what, deadline)):

- obl_created(o): the obligation o is created
- obl_fulfilled(o): the obligation o is fulfilled
- obl_unfulfilled(o): the obligation o is unfulfilled
- obl_inactive(o): the obligation o is inactive
- norm_failure(f): the failure f has happened

Environment integration

- Organisational Artifacts enable organisation and environment integration
- Embodied organisation [Piunti et al., 2009a]



status: ongoing work

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

⇝ 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**

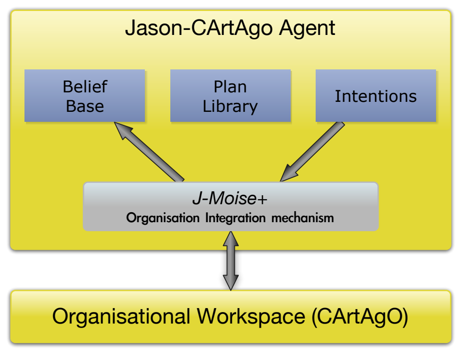
J-Moise: *Jason* + Moise

- Agents are programmed with *Jason*

⇝ BDI agents (reactive planning) – suitable abstraction level

- The programmer has the possibility to express sophisticated recipes for adopting roles, committing to missions, fulfilling/violating norms, ...
- Organisational information is made accessible in the mental state of the agent as beliefs
- Integration is totally independent of the distribution/communication layer

J-Moise: *Jason* + *Moise*– General view



Organisational **actions** in *Jason* I

- For groups:
  - `create_group`
  - `remove_group`

Example

```
...
.my_name(Me);
join_workspace(ora4mas,"",user_id(Me));
create_group(
       mypaper,     // group identification
       "wp-os.xml", // specification file
       wpgroup,     // group type
       false,       // monitoring scheme
       true);       // GUI
adopt_role(editor,mypaper);
```


Organisational **actions** in *Jason* II

- For schemes:
  - `create_scheme`
  - `add_responsible_group`
  - `remove_scheme`
  - `goal_achieved`

Example

```
create_scheme(
    s45,
    "wp-os.xml",
    writePaperSch,
    false,
    true);
add_responsible_group(s45,mypaper);
commit_mission(mManager, S).
```


Organisational **actions** in *Jason* III

- For roles:
  - `adopt_role`
  - `remove_role`
- For missions:
  - `commit_mission`
  - `remove_mission`

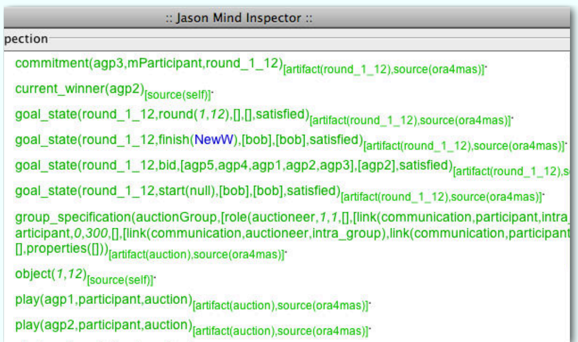
- Those actions usually are executed under **regimentation** (to avoid an inconsistent organisational state)
  e.g. the adoption of role is constrained by
  - the cardinality of the role in the group
  - the compatibilities of the roles played by the agent

Organisational perception

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

- group_specification
- scheme_specification
- plays(agent, role, group)
- responsible_group(group, scheme)
- commitment(agent, mission, scheme)
- goal(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



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)

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

Example (signals)

```
+norm_failure(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`

2OPL: Organisation Oriented Programming Language

2APL Syntax: Programming Multi-Agent System

Example (Exploring World)

```
<apaplmas>
    <agent name="w1" file="worker.2apl"/>
    <agent name="w2" file="worker.2apl"/>
    <agent name="m" file="manager.2apl"/>

    <environment name="blockworld" file="blockworld.jar"/>
</apaplmas>
```

- **Coordination** is needed to achieve the overall objectives of multi-agent systems.
- **Existing coordination mechanisms**
  - Coordination artifacts and languages defined in terms of low-level coordination concepts such as synchronization, shared-space, and channels, e.g., REO and Linda.
  - Organizational models, normative systems, and electronic institutions defined in terms of social and organizational concepts, e.g. Moise$^+$/ORG4MAS and ISLANDER/AMELI.

Programming Normative Organisations

The aim is to design and develop a programming language to support the implementation of coordination mechanisms in terms of normative concepts.

- **Agents**
  - specified in a programming language, for example, 2APL.
  - perform external actions.
  - internal architecture unknown to organization.
- **Organization**
  - determines effect of external actions.
  - normatively assesses effect of agents' actions (monitoring).
  - sanctions agents' wrongdoings (enforcement).
  - prevents ending up in really bad states (regimentation).

Programming Normative Organisations

Example

Example: Simple railway simulation

- simple railway station
- passengers travelling by train
- rules of conduct (e.g. having a ticket while on the train)

Passengers are agents that can:

- embark the train
- enter the platform
- buy a ticket

Programming Language for Organisations (1)

Programming a normative multi-agent organization is to specify:

- the initial state of organization by **brute facts**, e.g.,
  `{-at_platform, -in_train, -ticket}`
- the **effects of actions**, e.g.,
  `{-ticket}` *buy_ticket* `{ticket}`
  `{at_platform, -in_train}` *embark* `{-at_platform, in_train}`
- the norms through **counts-as rules**, e.g.,
  `{at_platform , -ticket}` ⇒ $\{viol_{ticket}\}$
  `{in_train , -ticket}` ⇒ $\{viol_{\perp}\}$
- possible sanctions for agent's through **sanction rules**, e.g.,
  $\{viol_{ticket}\} \Rightarrow \{fined_{10}\}$

Programming Language for Organisations (2)

Example (Train Station)

```
Facts:
    { -at_platform , -in_train , -ticket }

Effects:
    { -at_platform }    enter          { at_platform },
    { -ticket }         buy_ticket     { ticket },
    { at_platform , -in_train }
                        embark
                            { -at_platform, in_train }
Counts_as rules:
    { at_platform , -ticket } => { viol_ticket },
    { in_train , -ticket }    => { viol_|_ }

Sanction_rules:
    { viol_ticket } => { fined_10 }
```


Control Cycle = Monitoring + Enforcement

Repeat

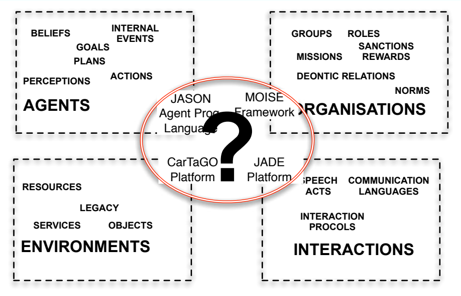
- Take the performed action.
- Realize its effect by means of its specification.
- Evaluate the state by applying Counts-as rules.
- Evaluate the state by applying Sanction-as rules.

Conclusion and Future Work

- Norms enforce/regiment states.
- Choreography language for action orchestration.
- Programming roles, their dynamics, and organisational structures.
- Adding explicit norms: Obligations, Prohibition, and Permissions.
- Adding deadlines to norms.
- Programming constructs for norm change.
- Logics for reasoning about organisation programs.

Conclusions

Putting the Pieces Together



Exploiting Orthogonality

- Treating AOP & EP & OOP as **orthogonal** dimensions
  - improving **separation of concerns**
    - using the best abstraction level and tools to tackle the specific dimensions, avoiding design pitfalls, such as using agents to implement either non-autonomous entities (e.g., a blackboard agent) or a collection of autonomous entities (group agent)
  - promoting openness and heterogeneity
    - E.g., heterogeneous agents working in the same organisation, heterogeneous agents working in the same environment, the same agent working in different and heterogeneous organisations, the same agent working in different heterogeneous environments
- Outcome from a programming point of view
  - code more clean and understandable
  - improving modularity, extensibility, reusability

Beyond Orthogonality: Synergetic Integration

- Exploiting one dimension to effectively design and program also aspects related to the other dimensions
  - for instance, using the environment to design, implement and represent at runtime the organisation infrastructure
- Designing and implementing MAS behaviours that are based on explicit bindings between the different dimensions
  - for instance, exploiting events occurring in the environment to represent events that have an effect at the institutional or social level

Exploiting Synergy between the A/E Dimensions

- Mapping
  - agent actions into environment operations (e.g. CArtAgO)
  - environment observable state/events into agent beliefs
- Outcome
  - agents with dynamic action repertoire
  - uniformly implementing any mechanisms (e.g. coordination mechanism) in terms of actions/percepts
    - no need to extend agents with special purpose primitives
  - exploiting a new type of agent modularity, based on **externalization** [Ricci et al., 2009a]

Exploiting Synergy on A/O Integration

- Normative deliberative agents
  - possibility to define mechanisms for agents to evolve within an organisation/several organisations
  - possibility to define proper mechanisms for deliberating on the internalisation/adoption/violation of norms
- Reorganisation, adaptation of the organisation
  - possibility to define proper mechanisms for diagnosing/evaluating/refining/defining organisations
- "Deliberative" Organisations
  - possibility to define dedicated organisational strategies for the regulation/adaptation of the organisation behaviour (organisational agents)

Exploiting Synergy between the E/O Dimensions

- Grounding the organisation infrastructure
  - implemented using environment abstractions
  - ... that agents perceive then as first-class entities of their world
- Mapping
  - organisational state reified by the environment computational state
  - organisational actions/perceptions reified by actions/percepts on the environment state
  - organisational functionalities encapsulated by suitably designed environment abstractions
- Outcome
  - "the power is back to agents" [Hübner et al., 2009c]
  - by perceiving and acting upon that environment, agents can reason and dynamically adapt the organisation infrastructure itself

An Example: ORA4MAS [Hübner et al., 2009c]

- Implementing organisational infrastructures based on $\mathcal{M}$oise organisational model by exploiting CArtAgO artifact-based environments
- Outcome on the agent side (e.g. *Jason*)
  - no need to introduce specific $\mathcal{M}$oise primitives
    - mapped directly onto artifact operations and so automatically part of agents' action repertoire
  - modularising and encapsulating $\mathcal{M}$oise functionalities into a set of properly designed artifacts
    - that constitute – both for the MAS engineers but also for agents – the organisation infrastructure

E/O Synergy Example: Implementing Regimentation

- Exploiting the environment role of **enabler** and **mediator** of agent interaction
  - by providing actions and generating percepts
- $\longrightarrow$ natural place where to embed and enforce **organisational rules and norms**
  - affecting action execution behaviour and percepts generation
- Examples
  - simple: a game-board artifact in an artifact-based environment
    - providing agents actions to make moves
    - encapsulating and enforcing the rules of the game
  - complex: fully-fledged institutions
    - reified into properly programmed environments

E/O Synergy Example: Implementing Constitutive Rules

- Exploiting the environment to create, represent, and manage dependencies and rules that are meaningful at the organisational level
- A main example: implementing constitutive rules [Searle, 1997]
  - events occurring in concrete environments conveyed as social and institutional events
  - typically represented in the form **X counts as Y in C**
  - an example: reaching the environment state S counts as achieving the organisational goal G
- The integration E/O allows for naturally design and implementation of these kinds of rules
  - without adding any further concepts wrt the ones belonging to the E/O dimensions

Ongoing and Related Research

- Unifying agents, environments and organisation perspectives
  - Volcano platform [Ricordel and Demazeau, 2002]
  - MASK platform [Occello et al., 2004]
  - MASQ [Stratulat et al., 2009], extending AGRE and AGREEN
  - Embodied organisations [Piunti, 2010]
  - Situated E-Institutions [Campos et al., 2009]
- Normative programming and infrastructures [Hübner et al., 2009a, Tinnemeier et al., 2009, Dastani et al., 2009]

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.
- Behrens, T., Bordini, R., Braubach, L., Dastani, M., Dix, J., Hindriks, K., Hbner, J., and Pokahr, A. (2010). An interface for agent-environment interaction. In *In Proceedings of International Workshop on Programming Multi-Agent Systems (ProMAS-8)*.
- Bernoux, P. (1985). *La sociologie des organisations*. Seuil, 3ème edition.
- Bordini, R., Hübner, J., and Wooldridge, M. (2007a). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Wiley-Interscience.

Bibliography III

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.

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.

Bibliography IV

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 (2007b).
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.

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

Bibliography V

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.

Campos, J., López-Sánchez, M., Rodriguez-Aguilar, J. A., and Esteva, M. (2009).
Formalising situatedness and adaptation in electronic institutions.
In *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, volume 5428/2009 of *LNCS*. Springer Berlin / Heidelberg.

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

Bibliography VI

Dastani, M. (2008b).
2APL: a practical agent programming language.
Autonomous Agent 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.

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.

Bibliography VII

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.

Dastani, M., Tinnemeier, N., and Meyer, J.-J. C. (2009).
A programming language for normative multi-agent systems.
In *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI-Global.

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 VIII

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 IX

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. (2000a).
The MADKIT agent platform architecture.
In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55.

Gutknecht, O. and Ferber, J. (2000b).
The MadKit agent platform architecture.
In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55.

Bibliography X

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.

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.

Bibliography XI

Hübner, J. F., Boissier, O., and Bordini, R. H. (2009a).
Normative programming for organisation management infrastructures.
In *MALLOW Workshop on Coordination, Organization, Institutions and Norms in Agent Systems in Online Communities (COIN-MALLOW 2009)*.

Hübner, J. F., Boissier, O., and Bordini, R. H. (2010).
A normative organisation programming language for organisation management infrastructures.
In et al., J. P., editor, *Coordination, Organizations, Institutions and Norms in Agent Systems V*, volume 6069 of *LNAI*, pages 114–129. Springer.

Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (2009b).
Instrumenting Multi-Agent Organisations with Organisational Artifacts and Agents.
Journal of Autonomous Agents and Multi-Agent Systems.

Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (2009c).
Instrumenting multi-agent organisations with organisational artifacts and agents: "Giving the organisational power back to the agents".
Autonomous Agents and Multi-Agent Systems.
DOI-URL: http://dx.doi.org/10.1007/s10458-009-9084-y.

Bibliography XII

Hübner, J. F., Sichman, J. S., and Boissier, O. (2002a).
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. (2002b).
MOISE+: Towards a Structural, Functional, and Deontic Model for MAS Organization.
In Castelfranchi, C. and Johnson, W. L., editors, *Proc. of International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 501–502. ACM Press.

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.

Bibliography XIII

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.

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.

Bibliography XIV

Molesini, A., Omicini, A., Denti, E., and Ricci, A. (2005).
SODA: A roadmap to artefacts.
In Dikenelli, O., Gleizes, M.-P., and Ricci, A., editors, *6th International Workshop "Engineering Societies in the Agents World" (ESAW'05)*, pages 239–252, Kuşadası, Aydın, Turkey. Ege University.

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

Occello, M., Baeijs, C., Demazeau, Y., and Koning, J.-L. (2004).
MASK: An AEIO toolbox to design and build multi-agent systems.
In et al., C., editor, *Knowledge Engineering and Agent Technology*, IOS Series on Frontiers in AI and Applications. IOS press, Amsterdam.

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.

Bibliography XV

Oliva, E., McBurney, P., Omicini, A., and Viroli, M. (2010).
Argumentation and artifacts for negotiation support.
International Journal of Artificial Intelligence, 4(S10):90–117.
Special Issue on Negotiation and Argumentation in Artificial Intelligence.

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.

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.

Bibliography XVI

Piunti, M. (2010).
Embodied organizations: a unifying perspective in programming agents, organizations and environments.
PhD thesis.

Piunti, M. and Ricci, A. (2009).
Cognitive artifacts for intelligent agents in mas: Exploiting relevant information residing in environments.
In *Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS 2008)*, volume 5605 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 XVII

Piunti, M., Ricci, A., Braubach, L., and Pokahr, A. (2008).
Goal-directed interactions in artifact-based mas: Jadex agents playing in CARTAGO environments.
In *Proc. of the 2008 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology (IAT'08)*, volume 2. IEEE Computer Society.

Platon, E., Mamei, M., Sabouret, N., Honiden, S., and Parunak, H. V. (2007).
Mechanisms for environments in multi-agent systems: Survey and opportunities.
Autonomous Agents and Multi-Agent Systems, 14(1):31–47.

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.

Bibliography XVIII

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.

Ricci, A., Omicini, A., Viroli, M., Gardelli, L., and Oliva, E. (2007a).
Cognitive stigmergy: Towards a framework based on agents and artifacts.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 124–140. Springer.

Bibliography XIX

Ricci, A., Piunti, M., Acay, L. D., Bordini, R., Hübner, J., and Dastani, M. (2008).
Integrating artifact-based environments with heterogeneous agent-programming platforms.
In *Proceedings of 7th International Conference on Agents and Multi Agents Systems (AAMAS08)*.

Ricci, A., Piunti, M., and Viroli, M. (2009a).
Externalisation and internalization: A new perspective on agent modularisation in multi-agent system programming.
In Dastani, M., Fallah-Seghrouchni, A. E., Leite, J., and Torroni, P., editors, *LADS*, volume 6039 of *Lecture Notes in Computer Science*, pages 35–54. Springer.

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).

Bibliography XX

Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009b).
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. (2009c).
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 XXI

- Ricci, A., Viroli, M., and Omicini, A. (2007b).
The A&A programming model & technology for developing agent environments in MAS.
In Dastani, M., El Fallah Seghrouchni, A., Ricci, A., and Winikoff, M., editors, *Programming Multi-Agent Systems*, volume 4908 of *LNAI*, pages 91–109. Springer Berlin / Heidelberg.
- Ricci, A., Viroli, M., and Omicini, A. (2007c).
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.
- Ricordel, P. and Demazeau, Y. (2002).
VOLCANO: a vowels-oriented multi-agent platform.
In Dunin-Keplicz and Nawarecki, editors, *Proceedings of the International Conference of Central Eastern Europe on Multi-Agent Systems (CEEMAS'01)*, volume 2296 of *LNAI*, pages 252–262. Springer Verlag.

Bibliography XXII

- Russell, S. and Norvig, P. (2003).
Artificial Intelligence, A Modern Approach (2nd ed.).
Prentice Hall.
- Searle, J. R. (1997).
The Construction of Social Reality.
Free Press.
- 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.

Bibliography XXIII

- Tinnemeier, N., Dastani, M., Meyer, J.-J., and van der Torre, L. (2009).
Programming normative artifacts with declarative obligations and prohibitions.
In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2009)*.
- Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K., and Zambonelli, F. (2007).
Infrastructures for the environment of multiagent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):49–60.
- Weyns, D. and Holvoet, T. (2004).
A formal model for situated multi-agent systems.
Fundamenta Informaticae, 63(2-3):125–158.
- Weyns, D. and Holvoet, T. (2007).
A reference architecture for situated multiagent systems.
In *Environments for Multiagent Systems III*, volume 4389 of *LNCS*, pages 1–40. Springer Berlin / Heidelberg.
- 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.

Bibliography XXIV

- Weyns, D. and Parunak, H. V. D., editors (2007).
Special Issue on Environments for Multi-Agent Systems, volume 14 (1) of *Autonomous Agents and Multi-Agent Systems*. Springer Netherlands.
- Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. (2005).
Environments for multiagent systems: State-of-the-art and research challenges.
In Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J., editors, *Environment for Multi-Agent Systems*, volume 3374, pages 1–47. Springer Berlin / Heidelberg.
- 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.