Multi-Agent Oriented Programming
(with JaCaMo)

O. Boissier\textsuperscript{1} R.H. Bordini\textsuperscript{2} J.F. Hübner\textsuperscript{3}
A. Ricci\textsuperscript{4} J.S. Sichman\textsuperscript{5}

1. Ecole Nationale Supérieure de Mines (ENSMSE), Saint Ettiene, France
2. Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil
3. Universidade Federal de Santa Catarina (UFSC), Florianópolis, Brazil
4. Università di Bologna (UNIBO), Bologna, Italy
5. Universidade de São Paulo (USP), São Paulo, Brazil

IJCAI 2015 Tutorial
Buenos Aires, Argentina, July 2015
Tutorial Organisation

- Introduction
- AOP – Agent Oriented Programming: Jason
- EOP – Environment Oriented Programming: CArtAgO
- OOP – Organisation Oriented Programming: Moise
- Conclusions
In collaboration with

🌍 Brazil
  ➤ L. Coutinho © Universidade de São Paulo & Universidade Federal do Maranhão, Brazil

✈️ France
  ➤ G. Picard, ENS Mines St-Etienne (gauthier.picard@emse.fr)
  ➤ M. Hannoun, B. Gâteau, G. Danoy, R. Kitio, C. Persson, R. Yaich © ENS Mines St-Etienne, France

✈️ Italy
  ➤ M. Piunti, A. Santi, Università degli studi di Bologna - DEIS, Bologna (a.ricci@unibo.it)

✈️ Romania
  ➤ A. Ciortea, A. Sorici, Politehnica University of Bucharest
Acknowledgements

- Dagstuhl Seminars:
  - #12342 (2012), #08361 (2008), #06261 (2006)

- Bilateral Projects:
  - USP-COFECUB 98-04
  - CMIRA Rhône-Alpes Region 2010

- French National Project:
  - FORTRUST Project ANR 06-10
  - ETHICAA Project ANR 14-18
Introduction
Outline

Introduction

Context & Requirements
Multi-Agent Systems (Our view)
  Definitions
  Conceptual Framework
Multi-Agent Oriented Programming (MAOP)
  MAOP Meta-Model
  Focus on Agent meta-model
  Focus on Environment meta-model
  Focus on Organisation meta-model
MAOP Perspective: the JaCaMo Platform

AOP: Agent Oriented Programming
  Reasoning Cycle
  Tools
  Shortfalls
  Trends
  Conclusions

EOP: Environment Oriented Programming
Context

Current Applications are:

- Open, non centralized & distributed socio-technical systems,
- Operating into Dynamic, Knowledge Intensive, Complex Environments
- Requiring:
  - Local/global computation
  - Flexibility (micro-macro or local-global loops)
  - Socio-technical integration (Trust, Policy/Norms, Legal knowledge, ...)


Multiple abstraction levels / Multiple decision mechanisms

Connection to the Physical World: Sensing/Acting, Reactive/Pro-active M2M Infrastructure

Combination of dynamics from Applications and M2M Domains (Applications/SLAs, M2M Infrastructure, Environment/Sensors)
Context (e.g. Smart Building)

- Smart co-working space (e.g. school, office building, ...) where people can book and use rooms according to their needs, location, current occupancy schedule
- Connection to the physical world: rooms are (i) equipped with projectors, white-boards, TV sets, ..., (ii) tagged by several usage categories (meeting, teaching, ...), (iii) augmented with sensors (temperature, light, presence, ...) and actuators
- Adaptive Coordination for managing allocation and functioning of rooms
Context (e.g. Ambient Assisted Living)

AAL collaboration with DOMUS Lab. [Castebrunet et al., 2010]

- Support of Human activities (representation, monitoring, adapting/reacting/anticipating) in several places (i.e. users should be assisted even if visiting other AAL persons in other apartment)
- Connection to the physical and human worlds (global configuration of the provided services, local smart place configuration & the user personal configuration that moves along with the inhabitant)
Requirements

- Open, Non centralized & Distributed Socio-Technical Systems
- Operating into Dynamic, Knowledge Intensive, Complex Environments
- Requiring Local/global computation, Flexibility (micro-macro loops) Socio-technical integration (Trust, Policy/Norms, Legal knowledge), ...

How to engineer such applications?
Outline

Introduction
  Context & Requirements
  Multi-Agent Systems (Our view)
    Definitions
    Conceptual Framework
  Multi-Agent Oriented Programming (MAOP)
    MAOP Meta-Model
    Focus on Agent meta-model
    Focus on Environment meta-model
    Focus on Organisation meta-model
  MAOP Perspective: the JaCaMo Platform

AOP: Agent Oriented Programming
  Reasoning Cycle
  Tools
  Shortfalls
  Trends
  Conclusions

EOP: Environment Oriented Programming
Multi-Agent Systems (MAS)

An organisation of autonomous agents interacting with each other within a shared environment

- **agents** can be: software/hardware, coarse-grain/small-grain, heterogeneous/homogeneous, reactive/pro-active entities
- **environment** can be virtual/physical, passive/active, deterministic/non deterministic, ...
- **interaction** is the motor of dynamic in MAS. Interaction can be: direct/indirect between agents, interaction between agent and environment
- **organisation** can be pre-defined/emergent, static/adaptive, open/closed, ...
Multi-Agent Systems (MAS)

An organisation of autonomous agents interacting with each other within a shared environment

- **agents** can be: software/hardware, coarse-grain/small-grain, heterogeneous/homogeneous, reactive/pro-active entities
- **environment** can be virtual/physical, passive/active, deterministic/non deterministic, ...
- **interaction** is the motor of dynamic in MAS. Interaction can be: direct/indirect between agents, interaction between agent and environment
- **organisation** can be pre-defined/emergent, static/adaptive, open/closed, ...
Multi-Agent Systems (MAS)

An organisation of autonomous agents interacting with each other within a shared environment

- **agents** can be: software/hardware, coarse-grain/small-grain, heterogeneous/homogeneous, reactive/pro-active entities
- **environment** can be virtual/physical, passive/active, deterministic/non deterministic, ...
- **interaction** is the motor of dynamic in MAS. Interaction can be: direct/indirect between agents, interaction between agent and environment
- **organisation** can be pre-defined/emergent, static/adaptive, open/closed, ...
Multi-Agent Systems (MAS)

An organisation of autonomous agents interacting with each other within a shared environment

- **agents** can be: software/hardware, coarse-grain/small-grain, heterogeneous/homogeneous, reactive/pro-active entities
- **environment** can be virtual/physical, passive/active, deterministic/non deterministic, ...
- **interaction** is the motor of dynamic in MAS. Interaction can be: direct/indirect between agents, interaction between agent and environment
- **organisation** can be pre-defined/emergent, static/adaptive, open/closed, ...
Multi-Agent Systems (MAS)

An organisation of autonomous agents interacting with each other within a shared environment

- **agents** can be: software/hardware, coarse-grain/small-grain, heterogeneous/homogeneous, reactive/pro-active entities
- **environment** can be virtual/physical, passive/active, deterministic/non deterministic, ...
- **interaction** is the motor of dynamic in MAS. Interaction can be: direct/indirect between agents, interaction between agent and environment
- **organisation** can be pre-defined/emergent, static/adaptive, open/closed, ...
Multi-Agent Systems (MAS)

An organisation of autonomous agents interacting with each other within a shared environment

MAS is not a simple set of agents

- **agents** can be: software/hardware, coarse-grain/small-grain, heterogeneous/homogeneous, reactive/pro-active entities
- **environment** can be virtual/physical, passive/active, deterministic/non deterministic, ...
- **interaction** is the motor of dynamic in MAS. Interaction can be: direct/indirect between agents, interaction between agent and environment
- **organisation** can be pre-defined/emergent, static/adaptive, open/closed, ...
## MAS Principles

### Agent Principles (Micro perspective)

- **Reactive, Pro-Active & Social** entities
- **Autonomy**: agents may exhibit activities that are not the one expected by the other agents in the system
- **Delegation**: agents may receive some control over their activities (loosely coupled entities)

### Multi-Agent System Principles (Macro perspective)

- **Distribution** of knowledge, resources, reasoning/decision capabilities
- **Decentralisation** of control, authority
- Agreement technologies, **Coordination** models and mechanisms to install coordination among the autonomous agents
- Interlacement of emergent, social order, normative functioning
Agents: abstractions for the definition of the decision/reasoning entities architectures

Environment: abstractions for structuring resources, processing entities shared among the agents

Interaction: abstractions for structuring interactions among entities

Organisation: abstractions for structuring and ruling the sets of entities within the MAS

A rich set of abstractions for capturing applications complexity!

Each dimension has its own dynamics

Dynamics may be interlaced into bottom-up / top-down global cycles

Coordination of these dynamics may be programmed into one or several dimensions [Boissier, 2003]

A rich palette of possible dynamics & coordination!!
In these approaches, some dimensions lose their control & visibility!

- Agent Oriented Programming [Shoham, 1993]
- Environment Oriented Programming [Ricci et al., 2011]
- Interaction Oriented Programming [Huhns, 2001]
- Organisation Oriented Programming [Pynadath et al., 1999]

Integrating the dimensions into one programming platform is not so easy!

- Volcano platform [Ricordel and Demazeau, 2002], MASK platform [Occello et al., 2004], MASQ [Stratulat et al., 2009], Situated E-Institutions [Campos et al., 2009], ...
Challenge

Shifting from an A/E/I/O oriented approaches to a Multi-Agent Oriented approach

- keeping alive the concepts, dynamics and coordinations of the A, E, I and O dimensions

in order to address the Intelligent Environment requirements.
Outline

Introduction
- Context & Requirements
- Multi-Agent Systems (Our view)
  - Definitions
  - Conceptual Framework

Multi-Agent Oriented Programming (MAOP)
- MAOP Meta-Model
- Focus on Agent meta-model
- Focus on Environment meta-model
- Focus on Organisation meta-model

MAOP Perspective: the JaCaMo Platform

AOP: Agent Oriented Programming
- Reasoning Cycle
- Tools
- Shortfalls
- Trends
- Conclusions

EOP: Environment Oriented Programming
Seamless Integration of A & E & I & O

JaCaMo Meta-model [Boissier et al., 2011], based on Cartago [Ricci et al., 2009b], Jason [Bordini et al., 2007c], Moise [Hübner et al., 2009] meta-models
Agent meta-model

Based on Jason meta-models [Bordini et al., 2007c]
Agent example 1

Example (Giacomo Agent Code)

!have_a_house. // Initial Goal
/* Plan */
+!have_a_house <- !contract;
   !execute.

Example (companyX Agent Code)

my_price(300). // initial belief
/* plans for contracting phase */
// there is a new value for current bid
+currentBid(V)
  : not i_am_winning(Art) & // I am not the current winner
      my_price(P) & P < V // I can offer a better bid
  <- .bid( P ). // place my bid offering a cheaper service
Agent & Agent Interaction meta-model

Agent
Belief
Goal
Plan
External Action
Internal Action
Trigger event
Action
Agent Dimension

Interaction Dimension
Message
Content
SpeechAct

Agent
Dimension
Environment meta-model

Based on A&A meta-model [Omicini et al., 2008]
public class AuctionArt extends Artifact {

    @OPERATION void init(String taskDs, int maxValue) {
        defineObsProperty("task", taskDs); // task description
        defineObsProperty("maxValue", maxValue); // max. value
        // current best bid (lower service price)
        defineObsProperty("currentBid", maxValue);
        // current winning agent ID
        defineObsProperty("currentWinner", "no_winner");
    }

    // places a new bid for doing the service for price p
    // (used by company agents to bid in a given auction)
    @OPERATION void bid(double bidValue) {
        ObsProperty opCurrentValue = getObsProperty("currentBid");
        ObsProperty opCurrentWinner = getObsProperty("currentWinner");
        if (bidValue <= opCurrentValue.intValue()) {
            opCurrentValue.updateValue(bidValue);
            opCurrentWinner.updateValue(getOpUserName());
        }
    }
}
A & E Interaction meta-model
Example

!have_a_house.  // Initial Goal
/* Plans */
+!have_a_house <- !contract; !execute.
+!contract <- !create_auction_artifacts; !wait_for_bids.
+!create_auction_artifacts
  <- !create_auction_artifact("SitePreparation", 2000);
  !create_auction_artifact("Floors", 1000);
  !create_auction_artifact("Walls", 1000);
  !create_auction_artifact("Roof", 2000);
  !create_auction_artifact("WindowsDoors", 2500);
  !create_auction_artifact("Plumbing", 500);
  !create_auction_artifact("ElectricalSystem", 500);
  !create_auction_artifact("Painting", 1200).
+create_auction_artifact(Task,MaxPrice)
  <- .concat("auction_for_", Task, ArtName);
  makeArtifact(ArtName, "tools.AuctionArt", [Task, MaxPrice],
              ArtId);
  focus(ArtId).
!create_auction_artifact(Task,MaxPrice)[error_code(Code)]
  <- .print("Error creating artifact ", Code).
+wait_for_bids
  <- println("Waiting the bids for 5 seconds...");
  .wait(5000); // use intern deadline of 5 sec to close auctions
+show_winners
  <- for ( currentWinner(Ag)[artifact_id(ArtId)] ) {
    ?currentBid(Price)[artifact_id(ArtId)]; // check current bid
    ?task(Task)[artifact_id(ArtId)]; // and task it is for
    println("Winner of task ", Task," is ", Ag, " for ", Price)
  }.
Example

my_price(1500). // initial belief
!discover_art("auction_for_Plumbing"). // initial goal
i_am_winning(Art) :- .my_name(Me) &
                   currentWinner(Me)[artifact_id(Art)].

/* plans for contracting phase */
+!discover_art(ToolName)
  <- joinWorkspace("HouseBuildingWsp");
  lookupArtifact(ToolName, ToolId);
  focus(ToolId).
// there is a new value for current bid
+currentBid(V)[artifact_id(Art)]
  : not i_am_winning(Art) & // I am not the current winner
    my_price(P) & P < V // I can offer a better bid
  <- bid(math.max(V-150, P))[artifact_id(Art)].
/* plans for execution phase */
...
Environment’s dynamics

<table>
<thead>
<tr>
<th>Artifact life-cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Creation/Deletion</td>
</tr>
<tr>
<td>▶ Activation/Execution/Fail or Success/Deactivation of an Operation</td>
</tr>
<tr>
<td>▶ Linking / Unlinking</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workspace life-cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Creation/Deletion of a workspace</td>
</tr>
<tr>
<td>▶ Creation/Deletion of Artifacts</td>
</tr>
<tr>
<td>▶ Creation/Deletion &amp; Entry/Exit of Agents</td>
</tr>
</tbody>
</table>
Outcomes of A & E Integration

- Agents with dynamic action repertoire, extended/reshaped by agents themselves
- Uniform implementation of 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]
Organisation meta-model

Simplified Moise meta-model [Hübner et al., 2009]
Example: Organisation Structural Specification

Graphical representation of Moise Struct. Spec.
Example: Organisation Functional Specification

Graphical representation of Moise Func. Spec.
**Example: Organisation Normative Specification**

<table>
<thead>
<tr>
<th>norm</th>
<th>modality</th>
<th>role</th>
<th>mission / goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1</td>
<td>Obl</td>
<td>house_owner</td>
<td>house built</td>
</tr>
<tr>
<td>n2</td>
<td>Obl</td>
<td>site_prep_contractor</td>
<td>site prepared</td>
</tr>
<tr>
<td>n3</td>
<td>Obl</td>
<td>bricklayer</td>
<td>floors laid, walls built</td>
</tr>
<tr>
<td>n4</td>
<td>Obl</td>
<td>roofer</td>
<td>roof built</td>
</tr>
<tr>
<td>n5</td>
<td>Obl</td>
<td>window_fitter</td>
<td>windows fitted</td>
</tr>
<tr>
<td>n6</td>
<td>Obl</td>
<td>door_fitter</td>
<td>doors fitted</td>
</tr>
<tr>
<td>n7</td>
<td>Obl</td>
<td>plumber</td>
<td>plumbing installed</td>
</tr>
<tr>
<td>n8</td>
<td>Obl</td>
<td>electrician</td>
<td>electrical system installed</td>
</tr>
<tr>
<td>n9</td>
<td>Obl</td>
<td>painter</td>
<td>interior painted, exterior painted</td>
</tr>
</tbody>
</table>

Simplified representation of \( M \)oise Norm. Spec.
A & E & O Interaction meta-model

Based on Cartago [Ricci et al., 2009b], Jason [Bordini et al., 2007c], Moise [Hübner et al., 2009] meta-models
A & O Integration

- Instrumenting Organisation Management by dedicated Organisational Artifacts
  - Mapping of the organisational state onto artifacts computational state
  - Encapsulation of organisational functionalities by suitably designed artifacts providing organisational operations

- Reification of organisation management actions/perceptions by actions/percepts on the artifacts

- Extensible set of organisational artifacts:
  - Openness Management Artifact [Kitio, 2011]
  - Reorganisation Artifact [Sorici, 2011]
  - Evaluation Artifact (kind-of reputation artifact) [Hübner et al., 2009]
  - Communication management Artifact [Ciortea, 2011]
A & O Integration (2)

▶ Exploit the uniform access to artifacts

⇒ Agents may be aware of the Organisation by the way of:
   ▶ organisational events
   ▶ organisational actions

⇒ Agents can reason on the organisation:
   ▶ to achieve organisational goals
   ▶ by developing organisational plans
Example (Adoption of Role)

...  
+!discover_art(ToolName)  
    <- joinWorkspace("HouseBuildingWsp");  
    lookupArtifact(ToolName, ToolId);  
    focus(ToolId).

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

+!site_prepared  
    <- ... // actions to prepare the site..
Env. Artifacts provide operations on shared resources

Org. Artifacts provide organisational operations

Both artifacts bound by count-as, enact constitutive rules [Piunti et al., 2009a, de Brito et al., 2012]

Org-agnostic agents may indirectly act on the organisation

Environment can act on the organisation

Organisation is embodied, situated in the environment
Count-as rules [de Brito et al., 2012]

Example

/* If an auction "Art" is finished, its winner ("Winner") plays a role "Role", if it doesn’t adopted it yet */

*auctionStatus(closed)[source(Art)]
count-as
play(Winner,Role,hsh_group)[source(hsh_group)]
in
  currentWinner(Winner)[source(Art)] &
  not(Winner==no_winner) &
  auction_role(Art,Role).

/* The occurrence of the event "prepareSite" means the achievement of organisational goal "site_prepared" */

+ prepareSite[agent_name(Ag),artifact_name(housegui)]
count-as
  goalState(bhsch,site_prepared,Ag,Ag,satisfied)[source(bhsch)].
Organisation’s dynamics (triggered by Agents, Environment)

- Organisation life-cycle
  - Entrance/Exit of an agent
  - Creation/Deletion of an Organisation entity
  - Change of Organisation specification

- Structural Organisation life-cycle
  - Creation/Deletion of a group
  - Adoption/Release of a role

- Functional Organisation life-cycle
  - Creation/End of a schema
  - Commitment/Release of a mission
  - Change of a global goal state

- Normative Organisation life-cycle
  - Activation/De-activation of obligation
  - Fulfilment/Violation/Sanction
Outcomes of A & E & 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/adoptions/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)
- “Embodied” Organisation / Organisation Aware Environment
  - possibility to connect organisation to environment
A MAOP meta-model

JaCaMo Meta-model [Boissier et al., 2011], based on Cartago [Ricci et al., 2009b], Jason [Bordini et al., 2007c], Moise [Hübner et al., 2009] meta-models
Outline

Introduction

Context & Requirements
Multi-Agent Systems (Our view)
  Definitions
  Conceptual Framework
Multi-Agent Oriented Programming (MAOP)
  MAOP Meta-Model
  Focus on Agent meta-model
  Focus on Environment meta-model
  Focus on Organisation meta-model

MAOP Perspective: the JaCaMo Platform

AOP: Agent Oriented Programming
  Reasoning Cycle
  Tools
  Shortfalls
  Trends
  Conclusions

EOP: Environment Oriented Programming
Integration of Multi-Agent technologies

- **Agent:** *Jason* agents [Bordini et al., 2007c]
- **Environment:** CArtAgO platform [Ricci et al., 2009b]
- **Organisation:** *Moise* framework with the extended/refactored version of the *Moise* OMI: ORA4MAS [Hübner et al., 2009]
- **Interaction:** based on tight integration between *Jason* and KQML or ACL/FIPA

Dimensions are integrated with dedicated bridges:

- **A–E** (c4Jason, c4Jadex [Ricci et al., 2009b])
- **E–O** (count-as/enact rules [Piunti et al., 2009a])
- **A–O** is for free (thanks to ORA4MAS). Strategies and reasoning capabilities from *J-Moise* [Hübner et al., 2007] can be reused.

Open to integrate other Multi-Agent Technologies
Integration with other technologies

- **Web 2.0**
  - implementing Web 2.0 applications

- **Android Platforms**
  - implementing mobile computing applications on top of the Android platform

- **Web Services**
  - building SOA/Web Services applications
  - [http://cartagows.sourceforge.net](http://cartagows.sourceforge.net)

- **Arduino Platforms**
  - building “Web of Things” Applications
  - [http://jacamo.sourceforge.net](http://jacamo.sourceforge.net)

- **Semantic Technologies**
  - JaSA: Semantically Aware Agents
  - [http://cartago.sourceforge.net](http://cartago.sourceforge.net)
Agent Oriented Programming
— AOP —
Books: [Bordini et al., 2005], [Bordini et al., 2009]

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

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

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

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

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
Agent Oriented Programming

Features

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

Fundamentals

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

- Heavily influence by the BDI architecture and reactive planning systems [Bratman et al., 1988]
BDI architecture [Wooldridge, 2009]

begin
  while true do
    \[ p \leftarrow \text{perception}() \] // belief revision
    \[ B \leftarrow \text{brf}(B, p) ; \] // belief revision
    \[ D \leftarrow \text{options}(B, I) ; \] // desire revision
    \[ I \leftarrow \text{filter}(B, D, I) ; \] // deliberation
    execute(I) ; // means-end
BDI architecture [Wooldridge, 2009]

```plaintext
1 while true do
2     \( B \leftarrow \text{brf}(B, \text{perception}()) \)
3     \( D \leftarrow \text{options}(B, I) \)
4     \( I \leftarrow \text{filter}(B, D, I) \)
5     \( \pi \leftarrow \text{plan}(B, I, A) \)
6     while \( \pi \neq \emptyset \) do
7         execute( head(\( \pi \)) )
8         \( \pi \leftarrow \text{tail}(\pi) \)
```
BDI architecture [Wooldridge, 2009]

1 while true do
2 \[ B \leftarrow \text{brf}(B, \text{perception}()) \]
3 \[ D \leftarrow \text{options}(B, I) \]
4 \[ I \leftarrow \text{filter}(B, D, I) \]
5 \[ \pi \leftarrow \text{plan}(B, I, A) \]
6 while \( \pi \neq \emptyset \) do
7 \[ \text{execute} \left( \text{head} \left( \pi \right) \right) \]
8 \[ \pi \leftarrow \text{tail}(\pi) \]
while true do
    B ← brf(B, perception())
    D ← options(B, l)
    l ← filter(B, D, l)
    π ← plan(B, l, A)
    while π ≠ ∅ do
        execute(head(π))
        π ← tail(π)
        B ← brf(B, perception())
        if ¬sound(π, l, B) then
            π ← plan(B, l, A);
    revise commitment to plan – re-planning for context adaptation
**BDI architecture** [Wooldridge, 2009]

1. `while true do`
2. \( B \leftarrow \text{brf}(B, \text{perception}()) \)
3. \( D \leftarrow \text{options}(B, I) \)
4. \( I \leftarrow \text{filter}(B, D, I) \)
5. \( \pi \leftarrow \text{plan}(B, I, A) \)
6. `while \( \pi \neq \emptyset \) and \( \neg \text{succeeded}(I, B) \) and \( \neg \text{impossible}(I, B) \) do`
7. \( \text{execute}(\text{head}(\pi)) \)
8. \( \pi \leftarrow \text{tail}(\pi) \)
9. \( B \leftarrow \text{brf}(B, \text{perception}()) \)
10. `if \( \neg \text{sound}(\pi, I, B) \) then`
11. \( \pi \leftarrow \text{plan}(B, I, A) ; \)

revise commitment to intentions – Single-Minded Commitment
while true do

B ← brf(B, perception())
D ← options(B, I)
I ← filter(B, D, I)
π ← plan(B, I, A)

while π ≠ ∅ and ¬succeeded(I, B) and ¬impossible(I, B) do

execute(head(π))
π ← tail(π)
B ← brf(B, perception())

if reconsider(I, B) then

D ← options(B, I);
I ← filter(B, D, I);

if ¬sound(π, I, B) then

π ← plan(B, I, A);

reconsider the intentions (not always!)
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming

Jason
Reasoning Cycle
Main Language Constructs: Beliefs, Goals, and Plans
Other Language Features
Comparison With Other Paradigms
The Jason Platform
Tools
Perspectives: Some Past and Future Projects
Shortfalls
Trends
Conclusions
Jason

(let’s go programming those nice concepts)
(BDI) Hello World – agent bob

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

+!say(X) : happy(bob) <- .print(X).  // I
Desires in Hello World

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

+!say(X) : not today(monday) <- .print(X).
Hello World
source of beliefs

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

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

plan selection

\[+\text{happy}(H)[\text{source}(A)]\]
\[\Rightarrow \text{sincere}(A) \& \text{my\_name}(H)\]
\[\leftarrow \!\text{say}(\text{hello})\].

\[+\text{happy}(H)\]
\[\Rightarrow \text{not} \ \text{my\_name}(H)\]
\[\leftarrow \!\text{say}(\text{i\_envy}(H))\].

\[+!\text{say}(X) : \text{not} \ \text{today}(\text{monday}) \leftarrow \text{print}(X)\].
Hello World

intention revision

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

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

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

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

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

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

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

-happy(H)  
  : .my_name(H)  
  <- .drop_intention(say(hello)).
AgentSpeak
The foundational language for Jason

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

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

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

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

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

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

**Intentions:** plans instantiated to achieve some goal
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
Basic Reasoning cycle

runtime interpreter

- perceive the environment and update belief base
- process new messages
- select event
- select relevant plans
- select applicable plans
- create/update intention
- select intention to execute
- execute one step of the selected intention
Jason Reasoning Cycle

1. **Percepts**
   - perceive

2. **BUF**
   - External Event
   - Selected Beliefs to Add and Delete

3. **checkMail**
   -perceives

4. **SocAcc**
   - Beliefs to Add and Delete

5. **Events**
   - Relevant Plans
   - Selected Event

6. **Unify Event**
   - Plans

7. **Check Context**
   - Consistent Plans

8. **Applicable Plans**
   - Intended Means

9. **Selected Intention**
   - New Intention

10. **Execute Intention**
    - Action
    - New Message

**Belief Base**

**Intentions**
- Push New Plan
- New Intention

**Suspended Intentions**
- (Actions and Msgs)

**Messages**
- perceive
- checkMail
- SocAcc

**Actions**
- act
- send
- sendMsg

**Plan Library**

**Percepts**
- BUF
- BRF

**Events**
- External Events
- Internal Events
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Jason
Reasoning Cycle
Main Language Constructs: Beliefs, Goals, and Plans
Other Language Features
Comparison With Other Paradigms
The Jason Platform
Tools
Perspectives: Some Past and Future Projects
Shortfalls
Trends
Conclusions
Beliefs — Representation

Syntax

Beliefs are represented by annotated literals of first order logic

\[ \text{functor}(term_1, \ldots, term_n)[annot_1, \ldots, annot_m] \]

Example (belief base of agent Tom)

\[
\begin{align*}
\text{red(box1)[source(percept)].} \\
\text{friend(bob,alice)[source(bob)].} \\
\text{lier(alice)[source(self),source(bob)].} \\
\text{\sim lier(bob)[source(self)].}
\end{align*}
\]
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)]
by communication

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

```plaintext
.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
```
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);

...
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

```prolog
.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)
An AgentSpeak plan has the following general structure:

\[
\text{triggering\_event} : \text{context} \leftarrow \text{body}.
\]

where:

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

**Boolean operators**
- `&` (and)
- `|` (or)
- `not` (not)
- `=` (unification)
- `>`, `>=` (relational)
- `<`, `<=` (relational)
- `==` (equals)
- `\==` (different)

**Arithmetic operators**
- `+` (sum)
- `-` (subtraction)
- `*` (multiply)
- `/` (divide)
- `div` (divide – integer)
- `mod` (remainder)
- `**` (power)
Plans — Operators for Plan **Body**

\[+\text{rain} : \ time\_to\_leave(T) \ & \ clock\_now(H) \ & \ H \geq T\]

\[- \text{?!g1}; \quad \text{// new sub-goal}\]

\[?\text{!!g2}; \quad \text{// new goal}\]

\[?\text{?b(X)}; \quad \text{// new test goal}\]

\[+\text{b1}(T-H); \quad \text{// add mental note}\]

\[-\text{b2}(T-H); \quad \text{// remove mental note}\]

\[-+\text{b3}(T*H); \quad \text{// update mental note}\]

\[\text{jia.get}(X); \quad \text{// internal action}\]

\[X > 10; \quad \text{// constraint to carry on}\]

\[\text{close(door); // external action}\]

\[?!\text{g3}[\text{hard\_deadline(3000)}]. \quad \text{// goal with deadline}\]
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).
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`
A note about “Control”

Agents can control (manipulate) their own (and influence the others)
- beliefs
- goals
- plan

By doing so they control their behaviour

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

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming

Jason
Reasoning Cycle
Main Language Constructs: Beliefs, Goals, and Plans

Other Language Features
Comparison With Other Paradigms
The Jason Platform
Tools
Perspectives: Some Past and Future Projects
Shortfalls
Trends
Conclusions
Other Language Features

Strong Negation

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

+!leave(home)
  :  not raining & not ~raining
  <=  .send(mum,askOne,raining,Answer,3000); ...
```
Prolog-like Rules in the Belief Base

\[
tall(X) :- \\
\qquad \text{woman}(X) \& \text{height}(X, H) \& H > 1.70 \\
\qquad \mid \\
\qquad \text{man}(X) \& \text{height}(X, H) \& H > 1.80.
\]

\[
\text{likely_color}(\text{Obj}, C) :- \\
\qquad \text{colour}(\text{Obj}, C)[\text{degOfCert}(D1)] \& \\
\qquad \text{not} \ (\text{colour}(\text{Obj}, \_)[\text{degOfCert}(D2)] \& D2 > D1) \& \\
\qquad \text{not} \ \sim\text{colour}(C, B).
\]
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).
Example (an agent blindly committed to $g$)

$+!g : g$. 

$+!g : \ldots \leftarrow \ldots \ ?g$. 

$-!g : true \leftarrow !g$. 
Example (an agent that asks for plans on demand)

```plaintext
-!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
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.
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Jason
Reasoning Cycle
Main Language Constructs: Beliefs, Goals, and Plans
Other Language Features
Comparison With Other Paradigms
The Jason Platform
Tools
Perspectives: Some Past and Future Projects
Shortfalls
Trends
Conclusions
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
public class Robot extends Thread {
    boolean seeGold, lowBattery;
    public void run() {
        while (true) {
            while (! seeGold) {
                a = randomDirection();
                doAction(go(a));
            }
            while (seeGold) {
                a = selectDirection();
                doAction(go(a));
            }
        }
    }
}
public class Robot extends Thread {
    boolean seeGold, lowBattery;
    public void run() {
        while (true) {
            while (! seeGold) {
                a = randomDirection();
                doAction(go(a));
                if (lowBattery) charge();
            }
            while (seeGold) {
                a = selectDirection();
                if (lowBattery) charge();
                doAction(go(a));
                if (lowBattery) charge();
            }
        }
    }
}
Jason code

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

%!find(gold) // long term goal
    <- ?direction(A);
go(A);
%!find(gold).

+battery(low) // reactivity
    <- !charge.

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

Introduction
   Definitions
   Conceptual Framework
   MAOP Meta-Model
   Focus on Agent meta-model
   Focus on Environment meta-model
   Focus on Organisation meta-model

AOP: Agent Oriented Programming
   Jason
      Reasoning Cycle
   Main Language Constructs: Beliefs, Goals, and Plans
   Other Language Features
   Comparison With Other Paradigms

The Jason Platform
   Tools

Perspectives: Some Past and Future Projects
   Shortfalls
   Trends
   Conclusions
Various communication and execution management infrastructures can be used with *Jason*:

**Centralised**: all agents in the same machine, one thread by agent, very fast

**Centralised (pool)**: all agents in the same machine, fixed number of thread, allows thousands of agents

**Jade**: distributed agents, FIPA-ACL

... others defined by the user (e.g. AgentScape)
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, ...
Tools

- Eclipse Plugin
- Mind Inspector
- Integration with
  - CArtAgO
  - .Moise
  - MADEM
  - Ontologies
  - ...
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Jason
Reasoning Cycle
Main Language Constructs: Beliefs, Goals, and Plans
Other Language Features
Comparison With Other Paradigms
The Jason Platform
Tools
Perspectives: Some Past and Future Projects
Shortfalls
Trends
Conclusions
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

- **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.

⇒ Multi-Agent Programming
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)

- More on jason.sourceforge.net, related projects
Summary

- **AgentSpeak**
  - Logic + BDI
  - Agent programming language

- **Jason**
  - AgentSpeak interpreter
  - Implements the operational semantics of AgentSpeak
  - Speech-act based communication
  - 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 research
Further Resources


- R.H. Bordini, J.F. Hübner, and M. Wooldrige
  Programming Multi-Agent Systems in AgentSpeak using *Jason*
Environment Oriented Programming
— EOP —
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming
Why Environment Programming in MAS
Basic Level
Advanced Level
A&A and CArtAgO
Conclusions and Wrap-up
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
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming
Why Environment Programming in MAS
Basic Level
Advanced Level
A&A and CArtAgO
Conclusions and Wrap-up
Basic Level Overview

MAS

actions

SIMULATED WORLD

OR

REAL WORLD
(PHYSICAL OR COMPUTATIONAL)

EXAMPLE:
JAVA PLATFORM

AGENTS

percepts

INTERFACE

OR

EXTERNAL WORLD
(PHYSICAL OR COMPUTATIONAL)

WRAPPER TO EXISTING TECHNOLOGY

Example:
JAVA PLATFORM
Basic Level: Features

- Environment conceptually conceived as a single monolithic 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

```
Environment
- globalPercepts: List<Literal>
- agPercepts: Map<String, List<Literal>>
+ init(String[] args)
+ stop()
+ getPercepts(String agName): List<Literal>
+ executeAction(String agName, Structure action): boolean
+ addPercept(String agName, Literal p)
+ removePercept(String agName, Literal p)
...

UserEnvironment
+ init(String[] args)
+ executeAction(String agName, Structure action): boolean
```

User Environment

Agent Architecture

- getPercepts
- executeAction
- change percepts
### MARS Environment in Jason

```java
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;
    }
}
```

```java
class MarsModel extends GridWorldModel { ... }

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

```java
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){...}

    ...

}
```
Belief Updates:

- \{ \text{bomb}(X,Y) \} \rightarrow \text{RemoveBomb}(X,Y) \{ \text{not bomb}(X,Y) \}
- \{ \text{true} \} \rightarrow \text{AddBomb}(X,Y) \{ \text{bomb}(X,Y) \}
- \{ \text{carry(bomb)} \} \rightarrow \text{Drop}(\) \{ \text{not carry(bomb)} \}
- \{ \text{not carry(bomb)} \} \rightarrow \text{PickUp}(\) \{ \text{carry(bomb)} \}

Beliefs:

- \text{start}(0,1).
- \text{bomb}(3,3).
- \text{clean(blockWorld)} :-
  \text{not bomb(X,Y), not carry(bomb)}.

Plans:

- \text{B(start(X,Y))} ;
- \text{@blockworld( enter( X, Y, blue ), L )}

Goals:

- \text{clean(blockWorld)}

PG-rules:

- \text{clean(blockWorld) \leftarrow \text{bomb}(X,Y) \mid}
  \{ \text{goto( X, Y )};
  \text{@blockworld( pickup( ), L1 );}
  \text{PickUp(\);}
  \text{RemoveBomb( X, Y );}
  \text{goto( 0, 0 );}
  \text{@blockworld( drop( ), L2 );}
  \text{Drop(\)}
  \}

PC-rules:

- \text{goto( X, Y ) \leftarrow \text{true} \mid}
  \{ \text{@blockworld( sensePosition(\), POS \};
  \text{B(POS = [A,B]);}
  \text{if B(A > X) then}
  \{ \text{@blockworld( west(\), L );}
    \text{goto( X, Y )}
  \}
  \text{else if B(A < X) then}
  \{ \text{@blockworld( east(\), L );}
    \text{goto( X, Y )}
  \}
  \text{else if B(B > Y) then}
  \{ \text{@blockworld( north(\), L );}
    \text{goto( X, Y )}
  \}
  \text{else if B(B < Y) then}
  \{ \text{@blockworld( south(\), L );}
    \text{goto( X, Y )}
  \}

...
Environment Interface Standard – 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
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
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming
Why Environment Programming in MAS
Basic Level
Advanced Level
A&A and CArtAgO
Conclusions and Wrap-up
Advanced Level Overview

- Vision: environment as a first-class abstraction in MAS [Weyns et al., 2007, Ricci et al., 2011]
  - application or 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
    - 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.
Research on Environments for MAS

- Environments for Multi-Agent Systems research field / \textit{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., 2011]
  - 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, as defined in software engineering contexts (Design by Contract)
- 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., 2011]
  - introducing a computational notion of artifact to design and implement agent environments
A&A and CArtAgO

(let’s go programming those nice concepts)
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming
Why Environment Programming in MAS
Basic Level
Advanced Level
A&A and CArtAgO
Conclusions and Wrap-up
Agents and Artifacts (A&A) Conceptual Model: Background Human Metaphor
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., 2011]
Artifact Abstract Representation
A World of Artifacts

- A counter
  - count: 5
  - inc
  - reset

- A flag
  - state: true
  - switch

- A bounded buffer
  - n_items: 0
  - max_items: 100
  - put
  - get

- An agenda
  - next_todo: check_plant
  - last_todo: ...
  - setTodo
cancelTodo

- A Stock Quote Web Service
  - state: available
  - wsdl: ...
  - GetLastTradePrice
  - ...

- An event service
  - clearEvents
  - postEvent
  - registerForEvs

- A data-base
  - n_records: 1001
  - table_names: ...
  - ...

- A tuple space
  - out
  - in
  - rd

151
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., 2010b]
  - success/failure semantics, execution semantics
  - defining the **contract** (in the SE acceptation) provided by the environment

<table>
<thead>
<tr>
<th>actions $\leftarrow\rightarrow$ artifacts’ operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
</tr>
<tr>
<td>- action success/failure semantics is defined by operation semantics</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>percepts $\leftarrow\rightarrow$ artifacts’ observable properties $+$ signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties represent percepts about the state of the environment signals</td>
</tr>
<tr>
<td>signals represent percepts concerning events signalled by the environment</td>
</tr>
</tbody>
</table>
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 another 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., 2009c]
- 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
Integration with Agent Languages and Platforms

- Integration with existing agent platforms [Ricci et al., 2008]
  - by means of bridges creating an action/perception interface and doing data binding

- Outcome
  - developing open and heterogeneous 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 Observer Agents

USER(S)

!create_and_use.

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

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

OBSERVER(S)

!observe.

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

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

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

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

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

▶ Working with the shared counter
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: Coordination Artifacts – A Bounded Buffer

```java
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 void put(Item obj) {
        await("bufferNotFull");
        items.add(obj);
        getObsProperty("n_items").updateValue(items.size());
    }

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

    @GUARD boolean itemAvailable() { return items.size() > 0; }
    @GUARD boolean bufferNotFull(Item obj) { return items.size() < nmax; }
}
```

- Basic operation features
  - output parameters to represent action feedbacks
  - long-term operations, with a high-level support for synchronization (await primitive, 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.`
Remarks

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

- Operation with output parameters as action feedbacks
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 3: Internal Processes – A Clock

Internal operations

- execution of operations triggered by other operations
- implementing controllable processes
Example 4: Artifacts for User I/O – GUI Artifacts

- Exploiting artifacts to enable interaction between human users and agents
Example 4: Agent and User Interaction

GUI ARTIFACT

```java
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);
  }

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

USER ASSISTANT AGENT

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

+value(V)
+ok : value(V)
  <- setValue(V+1).
+closed
  <- .my_name(Me);
  .kill_agent(Me).
```
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
  - JaCaMo model and platform (which is the evolution of the ORA4MAS infrastructure [Hübner et al., 2009])
- 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]
- Defining a Semantic (OWL-based) description of artifact environments (CArtAgO-DL)
  - JaSa project = JASDL + CArtAgO-DL
- ...

...
Applying CArtAgO and JaCa

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

Introduction
  - Definitions
  - Conceptual Framework
  - MAOP Meta-Model
  - Focus on Agent meta-model
  - Focus on Environment meta-model
  - Focus on Organisation meta-model

AOP: Agent Oriented Programming
  - Reasoning Cycle
  - Tools
  - Shortfalls
  - Trends
  - Conclusions

EOP: Environment Oriented Programming
  - Why Environment Programming in MAS
  - Basic Level
  - Advanced Level
  - A&A and CArtAgO
  - Conclusions and Wrap-up
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
Organisation Oriented Programming — OOP —
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
Origins and Fundamentals
Some OOP approaches
Moise Organisation Modeling Language (OML)
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 guaranteeing 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)

Agents know about organisation

Agents don’t know about organisation

Organisation Specification

Organisation Entity

Local Representation

Observed Organisation

Designer / Observer

Bottom-up

Top-down
Perspective on organisations from EASSS’05 Tutorial (Sichman, Boissier)

1. **Agent Centred**
   - Swarms, AMAS, SASO
   - Self-organisations ...
   - Organisation is observed.
   - Implicitly programmed in Agents, Interactions, Environment.

2. **Agents don’t know about organisation**

3. **Agents know about organisation**

4. **Organisation Centred**
   - AOSE
   - MASE, GAIA, MESSAGE, ...
   - Organisation is a design model.
   - It is hard-coded in Agents

5. **Organisation-Oriented Programming of MAS**
   - TAEMS, STEAM, AGR
   - MOISE+, OPERA, ...
   - Organisation is observed.
   - Coalition formation mechanisms programmed in Agents.

**Designers / Observers**

**Bottom-up**

**Top-down**
Perspective on Org.-Oriented Programming of MAS

- From organisations as an explicit description of the structure of the agents in the MAS in order to help them

- To organisations as the declarative and explicit definition of the coordination scheme aiming at “controlling/coordinating” the global reasoning of the MAS

〜 Normative Organisations
## Norms

<table>
<thead>
<tr>
<th>Norm</th>
<th>Norm mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Norms are <strong>rules</strong> that a society has in order to influence the behaviour of agents.</td>
<td></td>
</tr>
</tbody>
</table>

**Regimentation**: norm violation by the agents is prevented
- e.g., the access to computers requires an user name
- e.g., messages that do not follow the protocol are discarded

**Enforcement**: norm violation by the agents is made possible but it is monitored and subject to incentives
- e.g., a master thesis should be written in two years

→ Detection of violations, decision about ways of enforcing the norms (e.g., sanctions)
Normative Multi-Agent Organisation

Normative Multi-Agent System [Boella et al., 2008]

A MAS composed of mechanisms to represent, communicate, distribute, detect, create, modify, and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfilment.

Normative Multi-Agent Organisation

- Norms are expressed in the organisation specification to clearly define the coordination of the MAS:
  - anchored/situated in the organisation
  - i.e. norms refer to organisational concepts (roles, groups, etc.)
- Norms are interpreted and considered in the context of the organisation entity
- Organisation management mechanisms are complemented with norms management mechanisms (enforcement, regimentation, ...
Challenges: Normative Organisation vs Autonomy

Agents' desired behavior:

\[ P \cap E \cap O \text{ not too big} \]
- increases performance
- constrains agents' autonomy

\[ P \cap E \cap O \text{ not too small} \]
- increases adaptation
- keeps agents' autonomy

- **B**: agents' possible behaviors
- **P**: agents' behaviors that lead to global purpose
- **E**: agents' possible behaviors constrained by the environment
- **O**: agents' possible/permitted/obliged behaviors constrained by the normative organisation
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 inconsistent 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
Organisation Oriented Programming (OOP)

- Using organisational concepts
- To define a cooperative pattern
- Programmed outside of the agents and outside of the environment
- Program = Specification
- By changing the organisation, we can change the MAS overall behaviour
Organisation Oriented Programming (OOP)

First approach
- Agents read the program and follow it
Organisation Oriented Programming (OOP)

First approach
- Agents read the program and follow it

Second approach
- Agents are forced to follow the program
  - Agents are rewarded if they follow the program
  - Agents are sanctioned in the other case
Organisation Oriented Programming (OOP)

First approach

- Agents read the program and follow it

Second approach

- Agents are forced to follow the program
- Agents are rewarded if they follow the program
- Agents are sanctioned in the other case
Organisation Oriented Programming (OOP)

Components
- Programming Language (Org. Modeling Lang. – OML)
- Management Infrastructure (Org. Mngt Inf. – OMI)
- Integration to Agent architectures and to Environment
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],
  Moise\(^+\) [Hübner et al., 2002], ...

- 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],
  - $S$-Moise$^+$ [Hübner et al., 2006],
  - ORA4MAS [Hübner et al., 2009],
  - ...

- **Adaptation mechanisms**, i.e. reorganisation infrastructure
Components of OOP: Integration mechanisms

- **Agent** integration mechanisms allow agents to be aware of and to deliberate on:
  - entering/exiting the organisation
  - modification of the organisation
  - obedience/violation of norms
  - sanctioning/rewarding other agents
  
  e.g. $J\text{-Moise}^+$ [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 [de Brito et al., 2012], [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 plays a 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

An organisational specification is required to enable agents to “reason” about the organisation:

- to decide to enter into/leave from the organisation during execution
  - Organisation is no more closed
- to change/adapt the current organisation
  - Organisation is no more static
- to obey/disobey the organisation
  - Organisation is no more a regimentation
Motivations for OOP: 

Organisation point of view

An organisational specification is required to enable the organisation to “reason” about itself and about the agents in order to ensure the achievement of its global purpose:

▶ to decide to let agents enter into/leave from the organisation during execution
  → 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
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
Origins and Fundamentals
Some OOP approaches
Moise Organisation Modeling Language (OML)
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 can’t communicate with each other if they don’t 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 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.
**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 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

Diagram:

- Structural Specification
- Functional Specification

Environment
Team-Oriented Program (team plans and organization) requires roles for tasks. It searches for agents with relevant expertise and assists in assigning agents to organizational roles.

Execute the team plans of the team-oriented program.

Human Beings

Team-Oriented Programming Interface
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
(define-institution
soccer-server as
dialogic-framework = soccer-df
performative-structure = soccer-pf
norms = ( free-kick coach-messages ... )
)
**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 OMI: AMELI [Esteva et al., 2004]

- Communication Layer
- Institution Specification (XML format)
- GOVERNORS
- Agents Layer
- Public
- Private
- AMELI
- From [Noriega 04]
The aim is to design and develop a programming language to support the implementation of coordination mechanisms in terms of **normative concepts**.

An organisation

- determines effect of external actions
- normatively assesses effect of agents’ actions (monitoring)
- sanctions agents’ wrongdoings (enforcement)
- prevents ending up in really bad states (regimentation)
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 }
2OPL Modelling Dimension

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 }

Count-as rules:
{ at_platform, ~ticket } ⇒ { viol_ticket },
{ in_train, ~ticket } ⇒ { viol_/ }.

Reaction rules:
{ viol_ticket } ⇒ { fined_10 }.
Summary

- Several models
- Several dimensions on modelling organisation
  - Structural (roles, groups, ...)
  - Functional (global plans, ....)
  - Dialogical (scenes, protocols, ...)
  - Normative (norms)
Moise

(let’s go programming those nice concepts)
Moise Framework

- OML (language)
  - Tag-based language
    (issued from Moise [Hannoun et al., 2000], Moise$^+$ [Hübner et al., 2002], Moiselnst [Gâteau et al., 2005])

- OMI (infrastructure)
  - developed as an artifact-based working environment
    (ORA4MAS [Hübner et al., 2009] based on CArtAgO nodes, refactoring of S-Moise$^+$ [Hübner et al., 2006] and Synai [Gâteau et al., 2005])

- Integrations
  - Agents and Environment (c4Jason, c4Jadex [Ricci et al., 2009b])
  - Environment and Organisation ([Piunti et al., 2009a])
  - Agents and Organisation ($J$-Moise$^+$ [Hübner et al., 2007])
Moise in JaCaMo Metamodel

Cardinalities are not represented

- composition
- association
- primitive operations
- concept mapping
- dependency
- dimension border
Moise Framework in JaCaMo

CArtAgO, Jason, NOPL Engines

Jade, Janus, Java Platforms

mas-grp@ufsc.br  mas-grp@emse.fr
Moise Modelling Dimensions

Environment

Global goals, plans, Missions, schemas, preferences

Normative Specification
Permissions, Obligations
 Allows agents autonomy!

Structural Specification
Groups, links, roles
Compatibilities, multiplicities
inheritance

Functional Specification
Outline

Introduction
  Definitions
  Conceptual Framework
  MAOP Meta-Model
  Focus on Agent meta-model
  Focus on Environment meta-model
  Focus on Organisation meta-model

AOP: Agent Oriented Programming
  Reasoning Cycle
  Tools
  Shortfalls
  Trends
  Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
  Origins and Fundamentals
  Some OOP approaches
  Moise Organisation Modeling Language (OML)
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., 2009]
Moise OML meta-model (partial & simplified view)

Cardinalities are not represented

- composition
- association
- primitive operations
- concept mapping
- dimension border
- dependency
- structural spec.
- functional spec.
- normative spec.
Moise OML global picture
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
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 subgroup specifications (`group-specification` tag)

Example

```xml
<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

```xml
<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 (subgroups tag)
  - formation constraints on the components of the group (formation-constraints)

Example

```xml
<group-specification id="team">
  <roles>
    <role id="coach" min="1" max="2"/> ...
  </roles>
  <links> ... </links>
  <subgroups> ... </subgroups>
  <formation-constraints> ... </formation-constraints>
</group-specification>
```
extends-subgroups, scope

**extends-subgroups**

- Used for links or formation constraints
- if `extends-subgroups==true`, the link/constraint is also valid in all subgroups
- else it is valid only in the group where it is defined
- Default is `false`

**scope**

- Used for links or formation constraints
- if `scope==inter-group`: link or constraint exists for source or target belonging to different instances of the group
- if `scope==intra-group`: link or constraint exists for source or target belonging to the same instance of the group
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
  ▶ a scope (scope)
  ▶ and validity to subgroups (extends-subgroups)

Example

```xml
<link from="coach"
  to="player"
  type="authority"
  scope="inter-group"
  extends-subgroups="true" />
```
Formation constraint specification

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

Example

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

  ...
</formation-constraints>
```
Structural specification example (1)

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

- Specifies the expected behaviour of an MAS in terms of goals along two levels:
  - Collective with Scheme
  - Individual with Mission

- Components:
  - Goals:
    - Achievement goal (default type). Goals of this type should be declared as satisfied by the agents committed to them, when achieved
    - Maintenance goal. Goals of this type are not satisfied at a precise moment but are pursued while the scheme is running. The agents committed to them do not need to declare that they are satisfied
  - Scheme: global goal decomposition tree assigned to a group
    - Any scheme has a root goal that is decomposed into subgoals
  - Missions: set of coherent goals assigned to roles within norms
Functional specification

- 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

```xml
<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 identifier (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
Goal States from the Organization Point of View

- **waiting**: initial state
- **enabled**: goal pre-conditions are satisfied & scheme is well-formed
- **satisfied**: agents committed to the goal have achieved it
- **impossible**: the goal is impossible to be satisfied

Note: goal state from the Organization point of view may be different of the goal state from the Agent point of view
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>
    </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" />
</scheme>
...
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
Normative Specification

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

- 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 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

<table>
<thead>
<tr>
<th>role</th>
<th>deontic</th>
<th>mission</th>
<th>TTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>obliged</td>
<td>m1</td>
<td>1 minute</td>
</tr>
<tr>
<td>left</td>
<td>obliged</td>
<td>m2</td>
<td>3 minute</td>
</tr>
<tr>
<td>right</td>
<td>obliged</td>
<td>m2</td>
<td>1 day</td>
</tr>
<tr>
<td>attacker</td>
<td>obliged</td>
<td>m3</td>
<td>30 seconds</td>
</tr>
</tbody>
</table>

```xml
<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. When a group is well formed, it may 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
Outline

Introduction
  Definitions
  Conceptual Framework
  MAOP Meta-Model
  Focus on Agent meta-model
  Focus on Environment meta-model
  Focus on Organisation meta-model

AOP: Agent Oriented Programming
  Reasoning Cycle
  Tools
  Shortfalls
  Trends
  Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
  Origins and Fundamentals
  Some OOP approaches
  Moise Organisation Modeling Language (OML)
Organisation management infrastructure (OMI)

Responsibility

- Managing – coordination, regulation – the agents’ execution within organisation defined by an organisational specification

(e.g. MadKit, AMELI, S-Moise+, ...)
Based on A&A and Moise, Agents’ working environment is instrumented with Organizational Artifacts (OA) offering "organizational" actions

\( \leadsto \text{Distributed} \) management of the organization with a clear separation of concerns:

- **Agents:**
  - create, handle OAs and act on them
  - deploy and manage their OMI
  - perceive the organization state and violations of norms from the OAs
  - decide about sanctions

- **OAs** are in charge of interpreting Normative Programs
  - to detect and evaluate norms compliance
  - or to regiment norms
Normative Programming Language

The NPL norms have

- an activation condition
- a consequence

Two kinds of consequences are considered

- regimentations
- obligations

Example (Norm)

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

or

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

$\phi \rightarrow$ obligation$(a, r, g, d)$

- $\phi$: activation condition of the norm (e.g. play a role)
- $g$: the goal of the obligation (e.g. commit to a mission)
- $d$: the deadline of the obligation
A normative system configuration is a tuple: \( \langle F, N, ns, OS, t \rangle \) with

- \( F \) is a set of facts
- \( N \) is a set of norms
- \( ns \) is the state of the normative system (sound state \( \top \) or a failure state \( \bot \))
- \( OS \) is a set of obligations
  - each element \( os \in OS \) is \( \langle o, ost \rangle \)
    - where \( o \) obligation and \( ost \) its state
- \( t \) is the current time

The initial configuration of a NP \( P \) is \( \langle P_F, P_N, \top, \emptyset, 0 \rangle \)

- \( P_F \) and \( P_N \) are the initial facts and norms defined in the normative program \( P \)
Failure detection:

\[ n \in N \quad F \models n_\varphi \quad n_\psi = \text{fail}(\_ \_ \_) \]
\[ \langle F, N, \top, OS, t \rangle \rightarrow \langle F, N, \bot, OS, t \rangle \]

when any norm \( n \) becomes active (i.e., its condition component holds in the current state) and its consequence is \( \text{fail}(\_ \_ \_) \), the normative state is no longer sound but in failure (\( \bot \)).

Roll back from failure:

\[ \forall n \in N.(F \models n_\varphi \implies n_\psi \neq \text{fail}(\_ \_ \_)) \]
\[ \langle F, N, \bot, OS, t \rangle \rightarrow \langle F, N, \top, OS, t \rangle \]
Creation of obligation:

\[ n \in N \quad F \models n_\varphi \quad n_\psi = o \quad o_{\theta_d} > t \]

\[ \neg \exists \langle o', ost \rangle \in OS \ . \ (o' \overset{\text{obl}}{=} o_\theta \land ost \neq \text{inactive}) \]

\[ \langle F, N, \top, OS, t \rangle \rightarrow \langle F, N, \top, OS \cup \langle o_\theta, \text{active} \rangle, t \rangle \]

where \( \theta \) is the m.g.u. such that \( F \models o_\theta \)
Rules for Obligation Management

\[ \text{os} \in OS \quad \text{os} = \langle o, \text{active} \rangle \]
\[ F \models o_g \quad o_d \geq t \]
\[ \langle F, N, T, OS, t \rangle \rightarrow \]
\[ \langle F, N, T, (OS \setminus \{\text{os}\}) \cup \{\langle o, \text{fulfilled} \rangle \}, t \rangle \]  

(Fulfill)

\[ \text{os} \in OS \quad \text{os} = \langle o, \text{active} \rangle \quad o_d < t \]
\[ \langle F, N, T, OS, t \rangle \rightarrow \]
\[ \langle F, N, T, (OS \setminus \{\text{os}\}) \cup \{\langle o, \text{unfulfilled} \rangle \}, t \rangle \]

(Unfulfill)

\[ \text{os} \in OS \quad \text{os} = \langle o, \text{active} \rangle \quad F \not\models o_r \]
\[ \langle F, N, T, OS, t \rangle \rightarrow \]
\[ \langle F, N, T, (OS \setminus \{\text{os}\}) \cup \{\langle o, \text{inactive} \rangle \}, t \rangle \]

(Inactive)
NOPL

Normative Organisation Programming Language

- NOPL is a particular class of NPL: facts, rules and norms are specific to a OML (e.g. Moise NOML):

<table>
<thead>
<tr>
<th>id</th>
<th>condition</th>
<th>role</th>
<th>type</th>
<th>mission</th>
<th>TTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>n2</td>
<td>writer</td>
<td>obl</td>
<td>mCol</td>
<td>1 day</td>
<td></td>
</tr>
<tr>
<td>n3</td>
<td>writer</td>
<td>obl</td>
<td>mBib</td>
<td>1 day</td>
<td></td>
</tr>
<tr>
<td>n4</td>
<td>unfulfilled(n2)</td>
<td>editor</td>
<td>obl</td>
<td>ms</td>
<td>3 hours</td>
</tr>
<tr>
<td>n5</td>
<td>fulfilled(n3)</td>
<td>editor</td>
<td>obl</td>
<td>mr</td>
<td>3 hours</td>
</tr>
<tr>
<td>n6</td>
<td>#gnc</td>
<td>editor</td>
<td>obl</td>
<td>ms</td>
<td>30 minutes</td>
</tr>
<tr>
<td>n7</td>
<td>#rc</td>
<td>editor</td>
<td>obl</td>
<td>ms</td>
<td>1 hour</td>
</tr>
<tr>
<td>n6</td>
<td>#mc</td>
<td>editor</td>
<td>obl</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#gnc = goal_non_compliance
#rc = role_compatibility
#mc = mission_cardinality
Example (role cardinality norm – regimentation)

\[
group\_role(\text{writer},1,5).
\]

\[
\text{norm ncar}: \ group\_role(\text{R,}_*,\text{M}) \land \ rplayers(\text{R,G,V}) \land V > M \\
\rightarrow \ \text{fail}(\text{role\_cardinality}(\text{R,G,V,M})).
\]

Example (role cardinality norm – agent decision)

\[
\text{norm ncar}: \ group\_role(\text{R,}_*,\text{M}) \land \ rplayers(\text{R,G,V}) \land V > M \land \ plays(\text{E,editor,G}) \\
\rightarrow \ \text{obligation}(\text{E,ncar,committed}(\text{E,ms,}_*), \\
\text{‘now + 1 hour‘}).
\]
Moise Social scheme — NOPL — Facts

- **Static facts:**
  - `scheme_mission(m,max,min)`: cardinality of mission `m`;
  - `goal(m,g,pre-cond,‘ttf‘)`: mission, preconditions and TTF for goal `g`.

- **Dynamic facts (provided at run-time by the organisational artifact in charge of the management of the social scheme instance):**
  - `plays(a,ρ,gr)`: agent `a` plays the role `ρ` in the group instance identified by `gr`.
  - `responsible(gr,s)`: the group instance `gr` is responsible for the missions of the scheme instance `s`.
  - `committed(a,m,s)`: the agent `a` is committed to mission `m` in scheme `s`.
  - `achieved(s,g,a)`: the goal `g` has been achieved in the scheme `s` by the agent `a`. 
Moise Social scheme — NOPL — Rules

- Example of rules used to infer the state of the scheme:
  - Number of players of mission $M$ in scheme $S$:
    
    ```prolog
    mplayers(M,S,V) :-
      .count(committed(_,M,S),V).
    ```

  - Wellformedness property of scheme $S$:
    
    ```prolog
    well_formed(S) :-
      mplayers(mBib,S,V1) & V1 >= 1 & V1 <= 1 &
      mplayers(mCol,S,V2) & V2 >= 1 & V2 <= 5 &
      mplayers(mMan,S,V3) & V3 >= 1 & V3 <= 1.
    ```

  - Readyness of goal $G$ in scheme $S$ (i.e. goal is ready to be achieved):
    
    ```prolog
    ready(S,G) :-
      goal(_, G, PCG, _) & all_achieved(S,PCG).
    ```

    ```prolog
    all_achieved(_,[]).
    all_achieved(S,[G|T]) :-
      achieved(S,G,_ & all_achieved(S,T).
    ```
Moise Social scheme — NOPL — Norms

Norms for goals

- Agents are obliged to achieve their ready goals

norm ngoa:
  committed(A,M,S) & goal(M,G,_,D) &
  well_formed(S) & ready(S,G)
  -> obligation(A,ngoa,achieved(S,G,A),‘now‘ + D).

Norms for properties

- Mission cardinality as regimentation

norm mission_cardinality:
  scheme_mission(M,_,MMax) & mplayers(M,S,MP) & MP > MMax
  -> fail(mission_cardinality).

- Mission cardinality as obligation

norm mission_cardinality:
  scheme_mission(M,_,MMax) & mplayers(M,S,MP) & MP > MMax
  responsible(Gr,S) & plays(A,editor,Gr)
  -> obligation(A,mission_cardinality,
      committed(A,ms,_), ‘now‘+‘1 hour‘).
Definition of similar kinds of facts, rules and norms for the groups, roles in the structural specification

Domain norms:

- Each norm in the normative specification of the OS has a corresponding norm in the NOP
- Since in the OS, obligations refer to roles and missions, norms in corresponding NOP identify the agents playing the role in groups responsible for the scheme and take into account the property conditions.

norm n2:

\[
\begin{align*}
\text{plays}(A, \text{writer}, Gr) & \& \text{responsible}(Gr, S) \& \\
\text{mplayers}(mCol, S, V) & \& V < 5 \\
\rightarrow \text{obligation}(A, n2, \text{committed}(A, mCol, S), \text{now} + \text{1 day}).
\end{align*}
\]
Organisational Artifact Architecture

Org. Artifacts managing groups and social schemes execution:

▶ interpret programs written in Normative Programming Language (NPL) [Hübner et al., 2010] coming from the automatic translation of Moise programs

▶ generate signals

  ▶ oblCreated(o), oblFulfilled(o), oblUnfulfilled(o)
  ▶ oblInactive(o), normFailure(f)
  \[(o = \text{obligation(to whom, reason, what, deadline)})\]
Generic control cycle of an Organisational Artifact

// oe: current state of the org. managed by the artifact
// p: current NOPL program
// npi: NPL interpreter

When operation o is triggered by agent a do

- oe' <- oe \ creates a ‘‘backup’’ of current oe
- oe <- executes(o,oe)
- f <- a list of predicates representing oe
- r <- npi(p,f) \ runs the interpreter for the new state

If r == fail then

- oe <- oe’ \ restore the state backup
- fail operation o

else

- update observable properties from obligations state
- success operation o
ORA4MAS – GroupBoard artifact

Manages the functioning of an instance of group in the organization.

- **Operations:**
  - `adoptRole(role)` (resp. `leaveRole(role)`) attempts to adopt (resp. leave) role in the group.
  - `addScheme(schid)` (resp. `removeScheme(schid)`) attempts to set (resp. unset) the group responsible for the scheme managed by the SchemeBoard `schId`.

- **Observable Properties:**
  - `specification`: group spec. in the OS
  - `player`: list of players of role in the group
  - `schemes`: list of scheme identifiers that the group is responsible for
ORA4MAS– SchemeBoard artifact

Manages the functioning of an instance of social scheme in the organization.

▶ Operations:
  ▶ commitMission(mission) (resp. leaveMission): attempts to “commit” (resp “leave”) a mission in the scheme
  ▶ goalAchieved(goal): declares that goal is achieved
  ▶ setArgumentValue(goal, argument, value): defines the value of goal’s argument

▶ Observable Properties:
  ▶ specification: scheme spec. in the OS
  ▶ commitments: list of commitments to missions in the scheme
  ▶ groups: list of groups resp. for the scheme
  ▶ goalState: list of goals’ current state
  ▶ obligation: list of active obligations in the scheme
Partial Synthesis

- NPL, based on obligation and regimentation, formalised using operational semantics, specialised into NOPL
- Automatic translation of OS written in Moise OML into several NOPs
- Implementation in ORA4MAS, artifact-based OMI: Organisational Artifacts act as interpreters of NOPs.
  - NOPL (80%): dynamic of obligations (several aspects of the Moise OS have been translated to norms)
  - CArtAgO (10%): interface for agents
  - Java (10%): dynamic of organisational state
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
Origins and Fundamentals
Some OOP approaches
Moise Organisation Modeling Language (OML)
Organisational Artifacts enable organisation and environment integration

Embodied organisation [Piunti et al., 2009a]

status: ongoing work
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, ...)
Outline

Introduction
  Definitions
  Conceptual Framework
  MAOP Meta-Model
  Focus on Agent meta-model
  Focus on Environment meta-model
  Focus on Organisation meta-model

AOP: Agent Oriented Programming
  Reasoning Cycle
  Tools
  Shortfalls
  Trends
  Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
  Origins and Fundamentals
  Some OOP approaches
  Moise Organisation Modeling Language (OML)
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**

- **Jason-CArtAgo Agent**
  - Belief Base
  - Plan Library
  - Intentions

**J-Moise+**

- Organisation Integration mechanism

**Organisational Workspace (CArtAgO)**
Organisational **actions** in *Jason* 1

Example (GroupBoard)

```java
... joinWorkspace("ora4mas", O4MWsp);
makeArtifact(
    "auction",
    "ora4mas.nopl.GroupBoard",
    ["auction-os.xml", auctionGroup, false, true ],
    GrArtId);
adoptRole(auctioneer);
focus(GrArtId);
...
```
Organisational actions in Jason II

Example (SchemeBoard)

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


Organisational **actions** in *Jason III*

- For roles:
  - `adoptRole`
  - `leaveRole`

- For missions:
  - `commitMission`
  - `leaveMission`

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

- specification
- schemeSpecification
- play(agent, role, group)
- commitment(agent, mission, scheme)
- goalState(scheme, goal, list of committed agents, list of agents that achieved the goal, state of the goal)
- obligation(agent, norm, goal, deadline)
- normFailure(norm)
Organisational perception – example

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

- **Beliefs**

  commitment(bob,mManager,"sch2") [artifact_id(cobj_4),concept], artifact_name(cobj_4,"sch2"), artifact_type(cobj_4,"ora4m")
  commitment(bob,mManager,"sch1") [artifact_id(cobj_3),concept], artifact_name(cobj_3,"sch1"), artifact_type(cobj_3,"ora4m")
  current_wsp(cobj_1,"ora4mas","308b05b0-2994-4fe8-formationStatus(ok) [artifact_id(cobj_2),obs_prop_id("obs_"
  obj_2,"mypaper"),artifact_type(cobj_2,"ora4mas.nopl.GroupBo
  goalState("sch2",wp,[bob],[bob],satisfied) [artifact_id(cot

277
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)**

```prolog
+play(Ag,boss,GId) <- .send(Ag,tell,hello).
```

**Example (change in goal state)**

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

**Example (signals)**

```prolog
+normFailure(N) <- .print("norm failure event: ", N).
```
Typical plans for obligations

Example

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

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

+obligation(Ag,Norm,What,DeadLine)
  : .my_name(Ag)
  <- .print("I am obliged to ",What,
         ", but I don’t know what to do!").
Writing paper example
Organisation Specification

<organisational-specification>
  <structural-specification>
    <role-definitions>
      <role id="author" />
      <role id="writer"> <extends role="author"/> </role>
      <role id="editor"> <extends role="author"/> </role>
    </role-definitions>
  </structural-specification>
  <group-specification id="wpgroup">
    <roles>
      <role id="writer" min="1" max="5" />
      <role id="editor" min="1" max="1" />
    </roles>
    ...
  </group-specification>
</organisational-specification>
Writing paper sample I

Execution

jaime  action: jmoise.create_group(wpgroup)
    all  perception: group(wpgroup,g1)[owner(jaime)]

jaime  action: jmoise.adopt_role(editor,g1)

olivier action: jmoise.adopt_role(writer,g1)

jomi   action: jmoise.adopt_role(writer,g1)
    all  perception:
        play(jaime,editor,g1)
        play(olivier,writer,g1)
        play(jomi,writer,g1)
Writing paper sample II

Execution

jaime  action: jmoise.create_scheme(writePaperSch, [g1])
        all  perception: scheme(writePaperSch,s1)[owner(jaime)]
        all  perception: scheme_group(s1,g1)

jaime  perception:
        permission(s1,mManager)[role(editor),group(wpgroup)]

jaime  action: jmoise.commit_mission(mManager,s1)

olivier  perception:
        obligation(s1,mColaborator)[role(writer),group(wpgroup),
        obligation(s1,mBib)[role(writer),group(wpgroup)]

olivier  action: jmoise.commit_mission(mColaborator,s1)

olivier  action: jmoise.commit_mission(mBib,s1)

jomi  perception:
        obligation(s1,mColaborator)[role(writer),group(wpgroup),
        obligation(s1,mBib)[role(writer),group(wpgroup)]

jomi  action: jmoise.commit_mission(mColaborator,s1)
Writing paper sample III

Execution

    all  perception:
    commitment(jaime, mManager, s1)
    commitment(olivier, mColaborator, s1)
    commitment(olivier, mBib, s1)
    commitment(jomi, mColaborator, s1)
Writing paper sample IV

Execution

all perception: goal_state(s1,*,unsatisfied)

jaime (only wtitle is possible, Jaime should work)

event: +!wtitle

action: jmoise.set_goal_state(s1,wtitle,satisfied)
jaime event: +!wabs
action: jmoise.set_goal_state(s1,wabs,satisfied)
jaime  event:  +!wsectitles
action:  jmoise.set_goal_state(s1,wsectitles,satisfied)
Writing paper sample VII

Execution

olivier, jomi event: +!wsecs
action: jmoise.set_goal_state(s1,wsecs,satisfied)
jaime  event:  +!wcon; ...

olivier  event:  +!wref; ...
Writing paper sample IX

Execution

all  action: jmoise.remove_mission(s1)

jaime  action: jmoise.jmoise.remove_scheme(s1)
Useful tools — Mind inspector

\[
\begin{align*}
\text{play}(gaucho1, &\text{herder}, \text{gr\_herding\_grp\_13})[\text{source(\text{orgManager})}] \\
\text{play}(gaucho4, &\text{herdboy}, \text{gr\_herding\_grp\_13})[\text{source(\text{orgManager})}] \\
\text{play}(gaucho5, &\text{herdboy}, \text{gr\_herding\_grp\_13})[\text{source(\text{orgManager})}] \\
\text{pos}(45, &44, 128)[\text{source(\text{percept})}] \\
\text{scheme}(\text{herd\_sch}, &\text{sch\_herd\_sch\_18})[\text{owner(\text{gaucho3})}, \text{source(\text{orgManager})}] \\
\text{scheme}(\text{herd\_sch}, &\text{sch\_herd\_sch\_12})[\text{owner(\text{gaucho1})}, \text{source(\text{orgManager})}] \\
\text{scheme\_group}(\text{sch\_herd\_sch\_12}, &\text{gr\_herding\_grp\_13})[\text{source(\text{orgManager})}] \\
\text{steps}(700)[\text{source(\text{self})}] \\
\text{target}(6, &44)[\text{source(\text{gaucho1})}] \\
\end{align*}
\]

- **Rules**
  
  \[\text{random\_pos}(X, Y) :-
  (\text{pos}(\text{AgX}, \text{AgY}, _418) \& (\text{jia}\_\text{random}(RX, 40) \& ((RX > 5) \& ((X = ((RX-20)+\text{AgX}) \& ((X >
  \]

- **Intentions**
  
<table>
<thead>
<tr>
<th>Sel Id</th>
<th>Pen</th>
<th>Intended Means Stack (hide details)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16927</td>
<td>suspended-</td>
<td>+\text{be_in_formation}[\text{scheme}(\text{sch_herd_sch_12}), \text{mission}(\text{help})]</td>
</tr>
<tr>
<td></td>
<td>self</td>
<td>+\text{be_in_formation}[\text{scheme}(\text{Sch}), \text{mission}(\text{Mission})]</td>
</tr>
</tbody>
</table>
Outline

Introduction
  Definitions
  Conceptual Framework
  MAOP Meta-Model
  Focus on Agent meta-model
  Focus on Environment meta-model
  Focus on Organisation meta-model

AOP: Agent Oriented Programming
  Reasoning Cycle
  Tools
  Shortfalls
  Trends
  Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
MAPC - Agent on Mars Scenario

- Water wells
- Cost for traversing from one vertex to another
- Team blue’s zone
- Agents
- Team green’s zone
MAPC - Agent on Mars Scenario

- 2 teams, 28 agents each.

- Roles and actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Explorer</th>
<th>Repairer</th>
<th>Saboteur</th>
<th>Sentinel</th>
<th>Inspector</th>
</tr>
</thead>
<tbody>
<tr>
<td>recharge</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>attack</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>parry</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>goto</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>probe</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>survey</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>inspect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>buy</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>repair</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>skip</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
<structural-specification>
  <role-definitions>
    <role id="map_explorer"/>
    <role id="map_explorer_helper"/>
    <role id="soldier"/>
    <role id="medic"/>
    <role id="guardian"/>
    <role id="inspector"/>
    <role id="repairer"/>
    <role id="sentinel"/>
    <role id="saboteur"/>
    <role id="coordinator"/>
  </role-definitions>
</structural-specification>
<group-specification id="lti_usp_team">
  <roles>
    <role id="map_explorer" min="0" max="1" />
    <role id="map_explorer_helper" min="0" max="1" />
    <role id="coordinator" min="1" max="1" />
  </roles>
  <subgroups>
    <group-specification id="infantry" min="1" max="1">
      <roles>
        <role id="saboteur" min="0" max="1" />
        <role id="sentinel" min="0" max="1" />
        <role id="repairer" min="0" max="1" />
      </roles>
    </group-specification>
    <group-specification id="squad1" min="1" max="1">
      <roles>
        <role id="soldier" min="0" max="11" />
        <role id="guardian" min="0" max="1" />
        <role id="medic" min="0" max="1" />
        <role id="zone_explorer" min="0" max="1" />
      </roles>
    </group-specification>
    <group-specification id="squad2" min="1" max="1">
      <roles>
        <role id="soldier" min="0" max="6" />
        <role id="guardian" min="0" max="1" />
        <role id="medic" min="0" max="1" />
        <role id="zone_explorer" min="0" max="1" />
      </roles>
    </group-specification>
  </subgroups>
</group-specification>
LTI Team Code 1 - Coordinator Creates Groups

```plaintext
+!start
  <-  createWorkspace("marsWS");
       joinWorkspace("marsWS",MarsMWsp);

 // lti usp team group
 makeArtifact("teamGroupBoard","ora4mas.nopl.GroupBoard","["lti-usp-os.xml",lti_usp_team,false,false],GrArtId);
 setOwner(coordinator);
 focus(GrArtId);

 // squad1 subgroup
 makeArtifact("squad1GroupBoard","ora4mas.nopl.GroupBoard","["lti-usp-os.xml",squad1,false,false],Squad1GrArtId);
 setParentGroup(marsGroupBoard)[artifact_id(Squad1GrArtId)];
 focus(Squad1GrArtId);

 // squad2 subgroup
 makeArtifact("squad2GroupBoard","ora4mas.nopl.GroupBoard","["lti-usp-os.xml",squad2,false,false],Squad2GrArtId);
 setParentGroup(marsGroupBoard)[artifact_id(Squad2GrArtId)];
 focus(Squad2GrArtId);

 // infantry subgroup
 makeArtifact("infantryGroupBoard","ora4mas.nopl.GroupBoard","["lti-usp-os.xml",infantry,false,false],InfantryGrArtId);
 setParentGroup(marsGroupBoard)[artifact_id(InfantryGrArtId)];
 focus(InfantryGrArtId);

 // adopting the coordinator role in the lti_usp_team group
 adoptRole(coordinator)[artifact_id(GrArtId)];
```
<functional-specification>
  <scheme id="team_sch">
    <goal id="support_team_goal" ds="Support team">
      <plan operator="parallel">
        <goal id="coordinate_goal" 
          ds="Coordinate the agents to occupy the best zones"/>
        <goal id="explore_map_goal" 
          ds="Explore the graph"/>
        <goal id="help_explore_map_goal" 
          ds="Help explore the graph"/>
      </plan>
    </goal>
  </scheme>
</functional-specification>
<scheme id="attack_sch">
   <goal id="attack_opponent_goal" ds="Sabotage the opponents">
      <plan operator="parallel">
         <goal id="attack_goal" ds="Attack the opponents"/>
         <goal id="sabotage_goal" ds="Sabotage the opponents"/>
         <goal id="repair_goal" ds="Repair the other teammates"/>
      </plan>
   </goal>

   <mission id="m_attack" min="0" max="1">
      <goal id="attack_goal"/>
   </mission>

   <mission id="m_sabotage" min="0" max="1">
      <goal id="sabotage_goal"/>
   </mission>

   <mission id="m_repair" min="0" max="1">
      <goal id="repair_goal"/>
   </mission>
</scheme>
<scheme id="occupy_zone_sch">
  <goal id="occupy_zone" ds="Occupy the best zones in the map">
    <plan operator="parallel">
      <goal id="create_zone_goal" ds="Occupy the zone of Mars"/>
      <goal id="defend_zone_goal" ds="Defend the zone from opponents"/>
      <goal id="explore_zone_goal" ds="Explore the zone"/>
      <goal id="occupy_center_goal" ds="Occupy the best vertex of the zone"/>
    </plan>
  </goal>
</scheme>

<mission id="m_create_zone" min="6" max="11">
  <goal id="create_zone_goal"/>
</mission>

<mission id="m_defend_zone" min="1" max="2">
  <goal id="defend_zone_goal"/>
</mission>

<mission id="m_explore_zone" min="1" max="1">
  <goal id="explore_zone_goal"/>
</mission>

<mission id="m_occupy_center" min="1" max="1">
  <goal id="occupy_center_goal"/>
</mission>
LTI Team Code 2 - Coordinator Creates Schemes and Links to Teams

```python
// scheme creation
+!run_scheme

makeArtifact(teamSch, "ora4mas.nopl.SchemeBoard", ["lti-usp-os.xml", team_sch, false, false ], SchArtId);
    focus(SchArtId);
    .print("scheme teamSch created");
    addScheme(teamSch)[artifact_name("teamGroupBoard")];

makeArtifact(bestZoneSch, "ora4mas.nopl.SchemeBoard", ["lti-usp-os.xml", occupy_zone_sch, false, false ], SchArtId1);
    focus(SchArtId1);
    .print("scheme bestZoneSch created");
    addScheme(bestZoneSch)[artifact_name("squad1GroupBoard")];

makeArtifact(secondBestZoneSch, "ora4mas.nopl.SchemeBoard", ["lti-usp-os.xml", occupy_zone_sch, false, false ], SchArtId2);
    focus(SchArtId2);
    .print("scheme secondBestZoneSch created");
    addScheme(secondBestZoneSch)[artifact_name("squad2GroupBoard")];

makeArtifact(attackSch, "ora4mas.nopl.SchemeBoard", ["lti-usp-os.xml", attack_sch, false, false ], SchArtId3);
    focus(SchArtId3);
    .print("scheme attackSch created");
    addScheme(attackSch)[artifact_name("infantryGroupBoard")];
```
<normative-specification>
  
  <norm id="n1" type="obligation" role="coordinator"/>
  <norm id="n2" type="obligation" role="map_explorer"/>
  <norm id="n3" type="permission" role="map_explorer_helper"/>
  <norm id="n4" type="permission" role="saboteur"/>
  <norm id="n5" type="permission" role="sentinel"/>
  <norm id="n6" type="permission" role="inspector"/>
  <norm id="n7" type="permission" role="inspector"/>
  <norm id="n8" type="permission" role="repairer"/>
  <norm id="n9" type="permission" role="soldier"/>
  <norm id="n10" type="permission" role="guardian"/>
  <norm id="n11" type="obligation" role="zone_explorer"/>
  <norm id="n12" type="obligation" role="medic"/>

  mission="m_coordinate"/>
  mission="m_explorer_map"/>
  mission="m_help_explorer_map"/>
  mission="m_attack"/>
  mission="m_sabotage"/>
  mission="m_inspect"/>
  mission="m_create_zone"/>
  mission="m_repair"/>
  mission="m_create_zone"/>
  mission="m_defend_zone"/>
  mission="m_explore_zone"/>
  mission="m_occupy_center"/>

</normative-specification>
// plan to start to play a role R
+!playRole
  : role(R) & simStart // the role is sent by the MAPC simulator
  <- .print("I want to play role ",R);
  .send(coordinator,tell,want_to_play(R));
  !check_available_role.

// waiting coordinator response
// example: availableRole(explorer,map_explorer,m_explore_map,"teamSch","teamGroupBoard")
+!check_available_role : availableRole(R,F,M,S,G).

+!check_available_role : role(R)
  <- .wait({+availableRole(_,_,_,_,_)},400,_);
  .send(coordinator,tell,want_to_play(R));
  .print("Waiting role ",R);
  !check_available_role.

// adopt the role
+availableRole(R,F,M,S,G) : .my_name(Ag)
  <- !adoptRole(F,G);
  .print("I'm playing ",R, " on ",G);
  !commit_to_mission.

+!adoptRole(F,G)
  <- lookupArtifact(G,GrId);
  adoptRole(F)[artifact_id(GrId)].
// plan to commit to missions which the agent has permission/obligation
+!commit_to_mission
  : availableRole(R,F,M,S,G)
  <= .print("I will try to commit to ", M);
  commitMission(M)[artifact_name(S)].
Outline

Introduction
Definitions
Conceptual Framework
MAOP Meta-Model
Focus on Agent meta-model
Focus on Environment meta-model
Focus on Organisation meta-model

AOP: Agent Oriented Programming
Reasoning Cycle
Tools
Shortfalls
Trends
Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming
Origins and Fundamentals
Some OOP approaches
Meise Organisation Modeling Language (OML)
Summary

- 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.sourceforge.net
Summary

▶ **Jason**
  ▶ declarative and goal oriented programming
  ▶ goal patterns (maintenance goal)
  ▶ meta-programming (.drop intention( [group(g1)]))
  ▶ customisations (integration with the simulator and the organisation)
  ▶ internal actions (code in Java)
    ⇝ good programming style

▶ **Moise Framework**
  ▶ definition of groups and roles
  ▶ allocation of goals to agents based on their roles
  ▶ to change the team, we (developers) “simply” change the organisation
    ⇝ global orchestration
  ▶ team strategy defined at a high level
Putting the Pieces Together

BELIEFS
GOALS
PLANS
INTERNAL
EVENTS
PERCEPTIONS
AGENTS
MISSIONS
ROLES
DEONTIC RELATIONS
GROUPS
ROLES
SANCTIONS
MISSIONS
REWARDS
DEONTIC RELATIONS
NORMS
ORGANISATIONS
RESOURCES
LEGACY
SERVICES
OBJECTS
ENVIRONMENTS
SPEECH
ACTS
COMMUNICATION
LANGUAGES
INTERACTION
PROTOCOLS
INTERACTIONS

JASON
Agent Proc.
Language
MOISE
Framework
CarTaGO
Platform
JADE
Platform

309
Agent meta-model

- **Agent**
- **Belief**
- **Goal**
- **Plan**
- **Trigger event**
- **Action**
  - **External Action**
  - **Internal Action**

Cardinalities are not represented

- composition
- association
- dependency
- agent's actions
- concept mapping
- dimension border

- Cardinalities are not represented
Environment meta-model

- Manual
  - has
    - Workspace
    - Artifact
      - Operation
        - update
        - generate
          - Observable Property
          - Observable Event

Cardinalities are not represented

Composition
Association
Dependency
A & E Interaction meta-model
Organisation meta-model
JaCaMo binding concepts

- Env
  - act
  - perceive
  - count-as
- Org
  - role adoption obligations
- Ag
  - speech
  - acts
Multi Agent Oriented Programming!

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

MAS has many dimensions all as first class entities
Research on Multi-Agent Systems...

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

An interface for agent-environment interaction.

La sociologie des organisations.
Seuil, 3ème edition.

Introduction to the special issue on normative multiagent systems.

Contrôle et coordination orientés multi-agents.
Habilitation à diriger des recherches, ENS Mines Saint-Etienne et Université Jean Monnet.

Multi-agent oriented programming with jacamo.
Science of Computer Programming, pages –.

Programming Multi-Agent Systems in AgentSpeak Using Jason.
Wiley-Interscience.
Bibliography II


Bibliography III

Situating Cognitive Agents in GOLEM.
In Weyns, D., Brueckner, S., and Demazeau, Y., editors, Engineering
Springer Berlin / Heidelberg.

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.

Reasoning about autonomy in open multi-agent systems - an approach based on the social
power theory.
in french, ENS Mines Saint-Etienne.

Castebrunet, M., Boissier, O., Giroux, S., and Rialle, V. (2010).
Organization nesting in a multi-agent application for ambient intelligence.
In Demazeau, Y. and Dignum, F., editors, Proceedings of the 8th International Conference
on Practical Applications of Agents and Multi-Agent Systems (PAAMS’2010), Advances in

Modeling relationships for privacy preservation in virtual communities.
2apl: a practical agent programming language. 

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

Programming institutional facts in multi-agent systems. 
In *COIN-12, Proceedings*.

From interactions to collective behaviour in agent-based systems. 

Steps towards multi-agent oriented programming. 
(slides Workshop) 1st International Workshop on Multi-Agent Systems, IWMAS'97, Boston.

On the formal specification of electronic institutions. 


Bibliography VII

Formal semantics for an abstract agent programming language.
In Singh, M. P., Rao, A. S., and Wooldridge, M., editors, ATAL, volume 1365 of Lecture

A normative organisation programming language for organisation management
infrastructures.
In et al., J. P., editor, Coordination, Organizations, Institutions and Norms in Agent

Instrumenting Multi-Agent Organisations with Organisational Artifacts and Agents.
Journal of Autonomous Agents and Multi-Agent Systems.

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.

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.
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.

Developing Organised Multi-Agent Systems Using the MOISE+ Model: Programming
Issues at the System and Agent Levels.

Instrumenting Multi-Agent Organisations with reputation artifacts.
In Hubner, J. F., Matson, E., Boissier, O., and Dignum, V., editors, Coordination,
Organizations, Institutions, and Norms in Agent Systems IV, volume 5428 of LNAI,
pages 96–110. Springer.

Interaction-oriented programming.
In First international workshop, AOSE 2000 on Agent-oriented software engineering, pages

Gestion de l’ouverture au sein d’organisations multi-agents. Une approche basée sur des
artefacts organisationnels.
Tools for inventing organizations: Toward a handbook of organizational process. 

SODA: A roadmap to artefacts.

*La méthode (1) : la nature de la nature*.
Points Seuil.

MASK: An AEIO toolbox to design and build multi-agent systems.

A distributed normative infrastructure for situated multi-agent organisations.


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.

Embodied organisations in mas environments.

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.

Mechanisms for environments in multi-agent systems: Survey and opportunities.

Jadex: A bdi reasoning engine.
In [Bordini et al., 2005], pages 149–174.
Bibliography XII

An automated teamwork infrastructure for heterogeneous software agents and humans.  

Toward team-oriented programming.  

Agentspeak(l): Bdi agents speak out in a logical computable language.  

A platform for developing SOA/WS applications as open and heterogeneous multi-agent systems.  
To Appear.

Cognitive stigmergy: Towards a framework based on agents and artifacts.  


Bibliography XIV


Artificial Intelligence, A Modern Approach (2nd ed.).
Prentice Hall.

Jaca-android: An agent-based platform for building smart mobile applications.
In Dastani, M., Fallah-Sehrouchni, A. E., Hübner, J., and Leite, J., editors, Languages,
Methodologies and Development Tools for Multi-agent systems, volume 6822 of LNAI.
Springer Verlag.

Agent-oriented programming.

Agile governance in an ambient intelligence environment.

MASQ: towards an integral approach to interaction.
In AAMAS (2), pages 813–820.

Towards flexible teamwork.


