Multi-Agent Oriented Programming (with JaCaMo)

O. Boissier[1] R.H. Bordini[2] J.F. Hübner[3]
A. Ricci[4] J.S. Sichman[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: $\mathcal{M}$oise
- 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

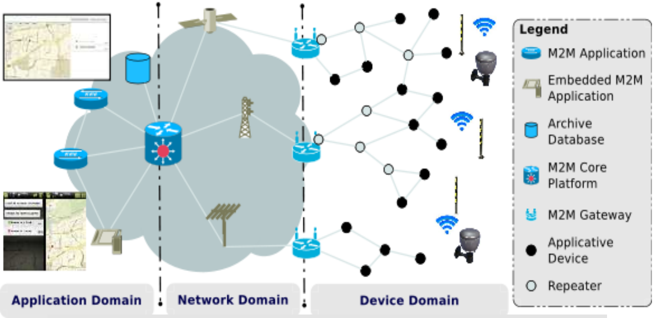
Outline

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, ...)

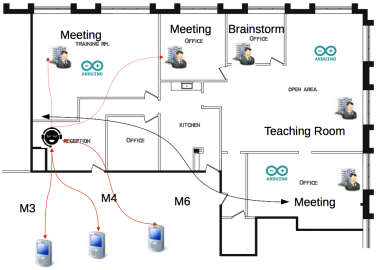
Context (e.g. Smart City M2M Infrastructure)



European Telecommunications Standards Institute (ETSI) view on M2M infrastructure

- 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)



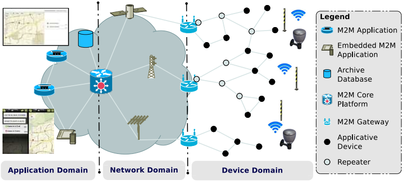
- Smart co-working space (e.g. school, office building, ...) where people can book and use rooms according to their needs, location, current occupancy schedule
- Connection to the physical world: rooms are (i) equipped with projectors, white-boards, TV sets, ..., (ii) tagged by several usage categories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

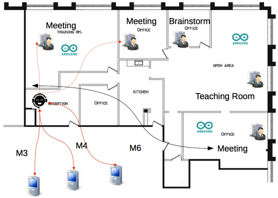
Context (e.g. Ambient Assisted Living)

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)

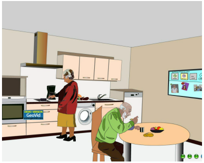
Requirements

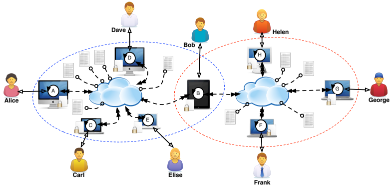




- Open, Non centralized & Distributed Socio-Technical Systems
- Operating into Dynamic, Knowledge Intensive, Complex Environments
- Requiring Local/global computation, Flexibility (micro-macro loops) Socio-technical integration (Trust, Policy/Norms, Legal knowledge), ...

How to engineer such applications?





Outline

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

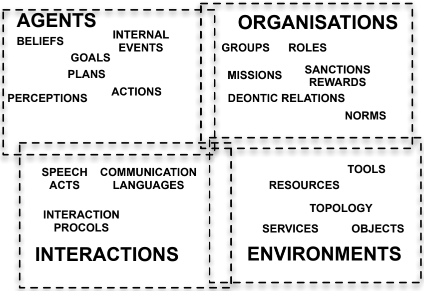
Agent Principles (Micro perspective)

- 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

MAS Conceptual framework / Dimensions

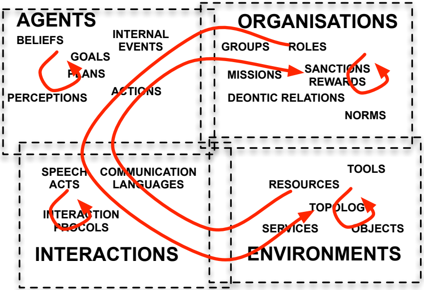


cf. VOWELS [Demazeau, 1995, Demazeau, 1997]

- 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!

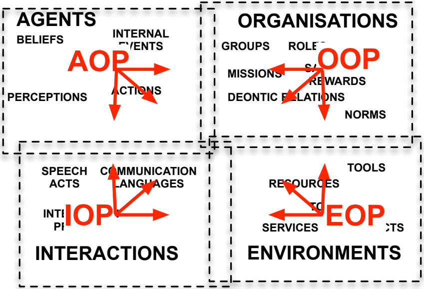
MAS Conceptual framework / Dynamics



- 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!!

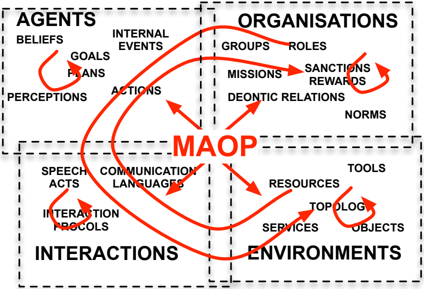
MAS Programming



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

- In these approaches, some dimensions lose their control & visibility!
- 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], ...)

MAS Programming



Challenge

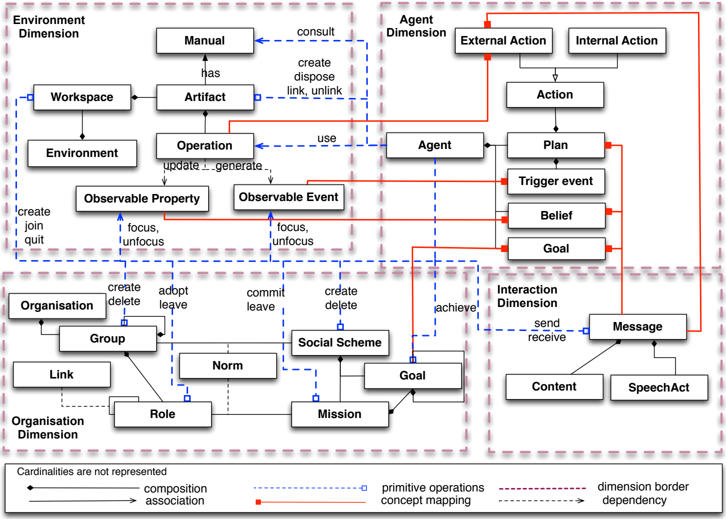
Shifting from an A/E/I/O oriented approaches to a Multi-Agent Oriented approach

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

in order to address the Intelligent Environment requirements.

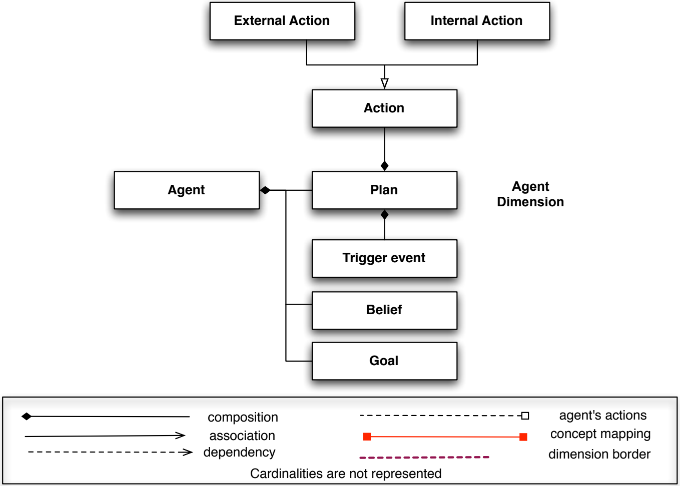
Outline

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], $\mathcal{M}$oise [Hübner et al., 2009] meta-models

Agent meta-model



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

Agent example I

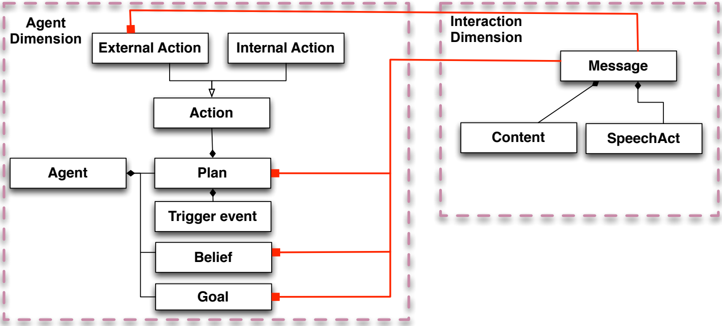
Example (Giacomo Agent Code)

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

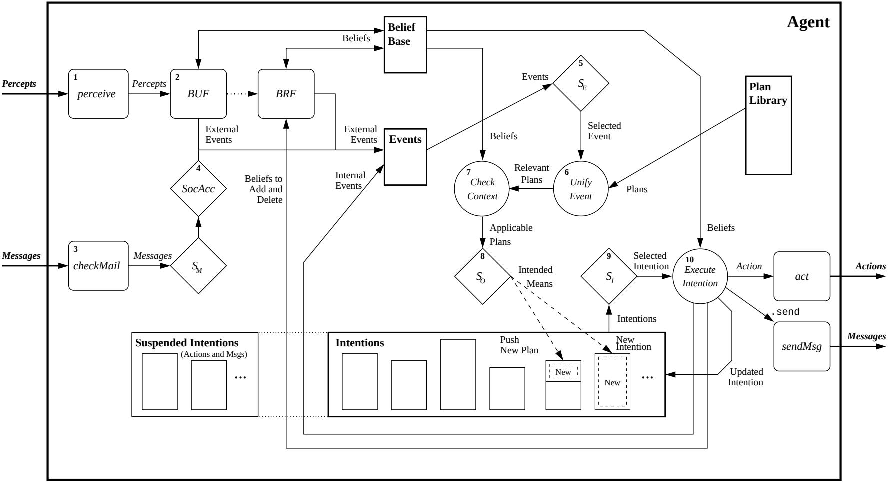
Example (companyX Agent Code)

```
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
```

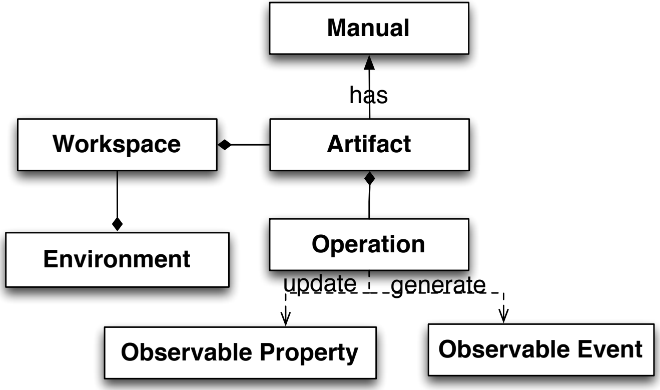
Agent & Agent Interaction meta-model



Agent's dynamics



Environment meta-model



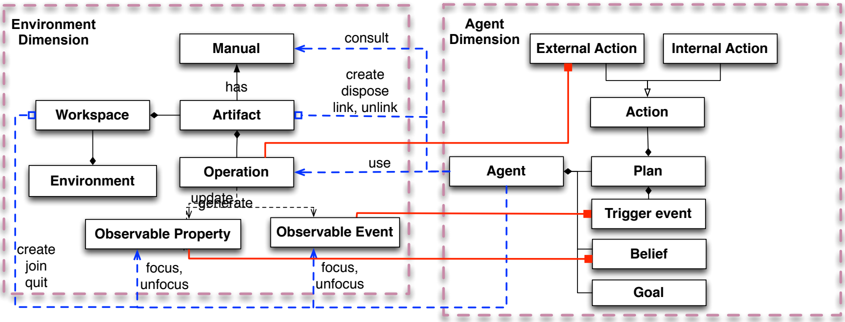
Based on A&A meta-model [Omicini et al., 2008]

Auction Artifact

Example

```
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



Giacomo Agent Code I

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).
```

Example

Giacomo Agent Code II

```
+!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.
+!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)
      }.
```

companyA Agent Code I

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

Artifact life-cycle

- Creation/Deletion
- Activation/Execution/Fail or Success/Deactivation of an Operation
- Linking / Unlinking

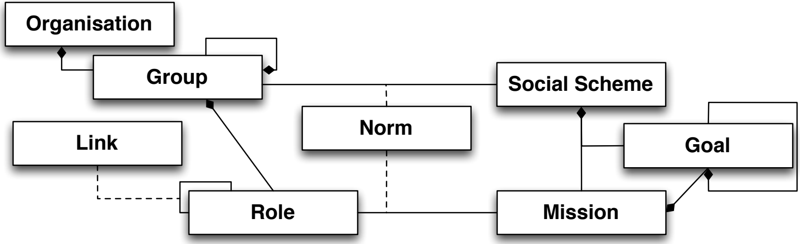
Workspace life-cycle

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

Outcomes of A & E Integration

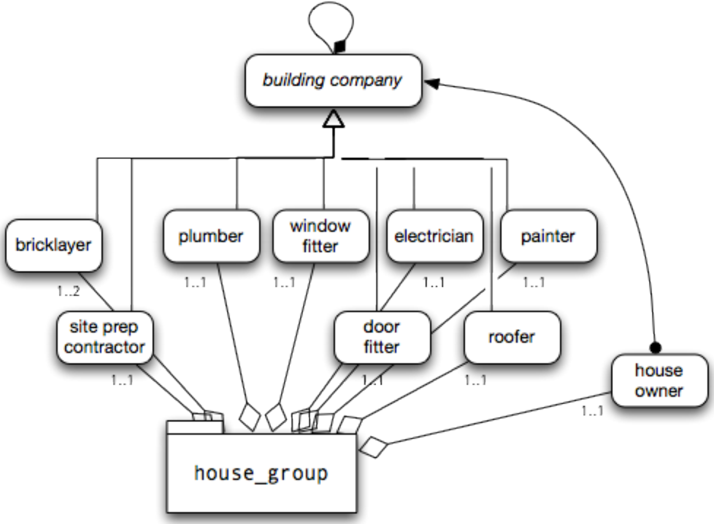
- Agents with dynamic action repertoire, extended/reshaped by agents themselves
- Uniform implementation of any mechanisms (e.g. coordination mechanism) in terms of actions/percepts
  - No need to extend agents with special purpose primitives
- Exploiting a new type of agent modularity, based on externalization [Ricci et al., 2009a]

Organisation meta-model



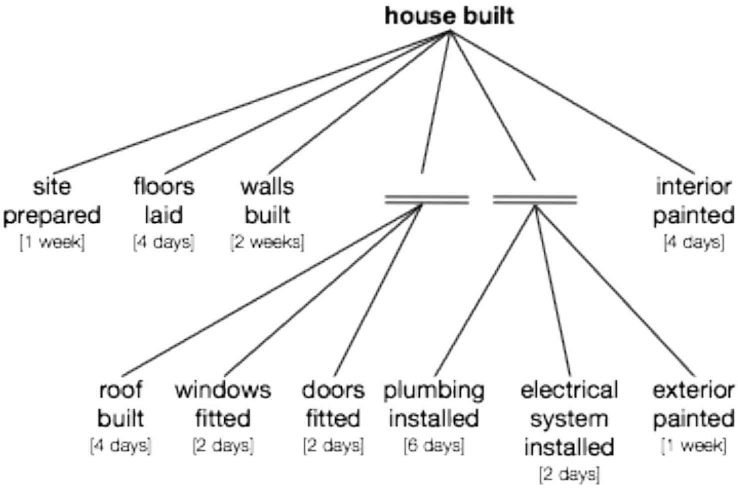
Simplified $\mathcal{M}$oise meta-model [Hübner et al., 2009]

Example: Organisation Structural Specification



Graphical representation of $\mathcal{M}$oise Struct. Spec.

Example: Organisation Functional Specification



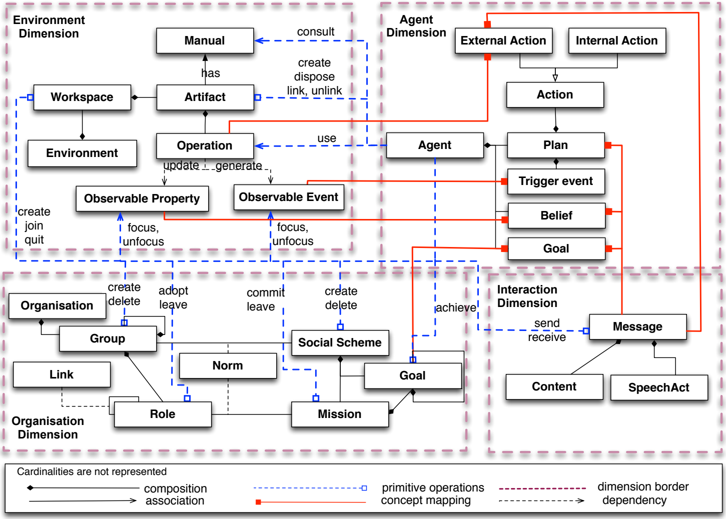
Graphical representation of $\mathcal{M}$oise Func. Spec.

Example: Organisation Normative Specification

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

Simplified representation of $\mathcal{M}$oise Norm. Spec.

A & E & O Interaction meta-model



Based on Cartago [Ricci et al., 2009b], Jason [Bordini et al., 2007c], $\mathcal{M}$oise [Hübner et al., 2009] meta-models

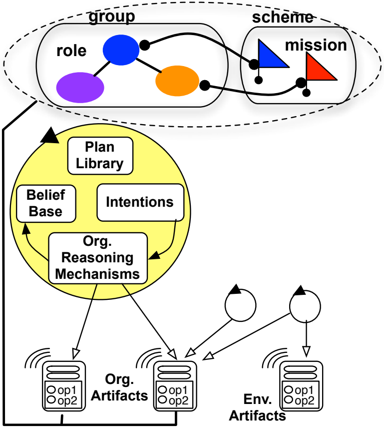
A & O Integration

- 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

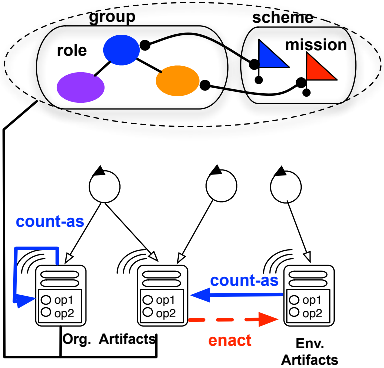
⇝ Agents can reason on the organisation:
  - to achieve organisational goals
  - by developing organisational plans

Example

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..
```

E & O Integration



- 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)].
```

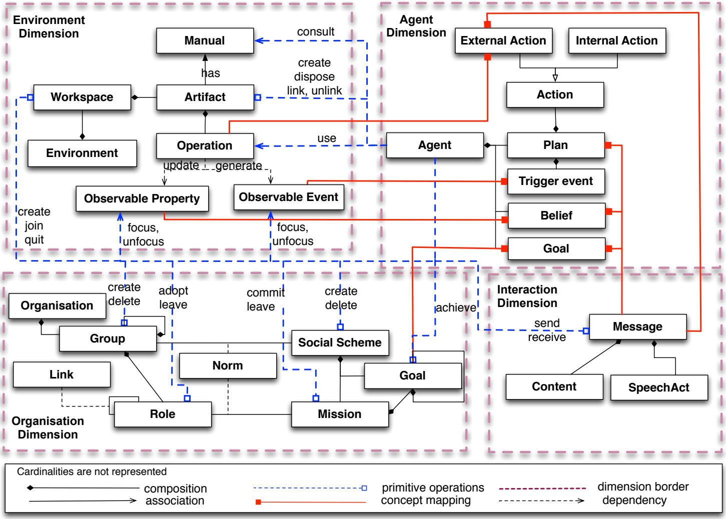
Organisation's dynamics (triggered by Agents, Environment)

- 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/adoption/violation of norms
- Reorganisation, adaptation of the organisation
  - possibility to define proper mechanisms for diagnosing/evaluating/refining/defining organisations
- "Deliberative" Organisations
  - possibility to define dedicated organisational strategies for the regulation/adaptation of the organisation behaviour (organisational agents)
- "Embodied" Organisation / Organisation Aware Environment
  - possibility to connect organisation to environment

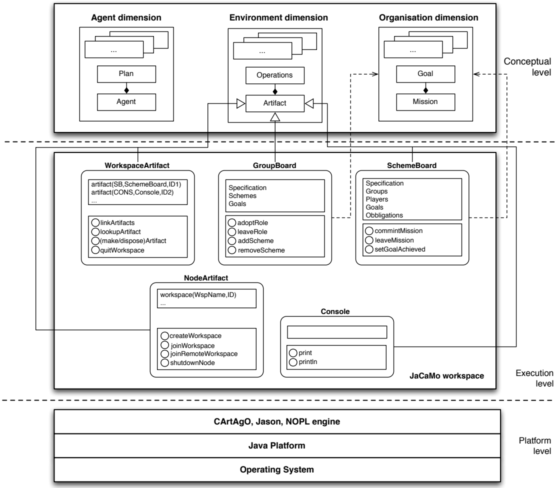
A MAOP meta-model



JaCaMo Meta-model [Boissier et al., 2011], based on Cartago [Ricci et al., 2009b], Jason [Bordini et al., 2007c], $\mathcal{M}$oise [Hübner et al., 2009] meta-models

Outline

JaCaMo Platform http://jacamo.sourceforge.net



Integration of Multi-Agent technologies

- Agent: *Jason* agents [Bordini et al., 2007c]
- Environment: CArtAgO platform [Ricci et al., 2009b]
- Organisation: $\mathcal{M}$oise framework with the extended/refactored version of the $\mathcal{M}$oise 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 $\mathcal{J}$-$\mathcal{M}$oise^{+} [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
  - `http://jaca-web.sourceforge.net`
- Android Platforms
  - implementing mobile computing applications on top of the Android platform
  - `http://jaca-android.sourceforge.net`
- Web Services
  - building SOA/Web Services applications
  - `http://cartagows.sourceforge.net`
- Arduino Platforms
  - building "Web of Things" Applications
  - `http://jacamo.sourceforge.net`
- Semantic Technologies
  - JaSA: Semantically Aware Agents
  - `http://cartago.sourceforge.net`

Agent Oriented Programming

— AOP —

Literature I

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 $\times$ 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 ← perception()
        B ← brf(B, p) ;            // belief revision
        D ← options(B, I) ;        // desire revision
        I ← filter(B, D, I) ;      // deliberation
        execute(I) ;               // means-end
```

BDI architecture [Wooldridge, 2009]

while true **do**

$\quad B \leftarrow \mathit{brf}(B, \mathit{perception}())$

$\quad D \leftarrow \mathit{options}(B, I)$

$\quad I \leftarrow \mathit{filter}(B, D, I)$

$\quad \pi \leftarrow \mathit{plan}(B, I, A)$

$\quad$ **while** $\pi \neq \emptyset$ **do**

$\qquad$ execute($\mathit{head}(\pi)$)

$\qquad \pi \leftarrow \mathit{tail}(\pi)$

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception())
    D ← options(B, I)
    I ← filter(B, D, I)
    π ← plan(B, I, A)
    while π ≠ ∅ do
        execute( head(π) )
        π ← tail(π)
```

BDI architecture [Wooldridge, 2009]

while true **do**
    $B \leftarrow brf(B, perception())$
    $D \leftarrow options(B, I)$
    $I \leftarrow filter(B, D, I)$
    $\pi \leftarrow plan(B, I, A)$
    while $\pi \neq \emptyset$ **do**
        execute($head(\pi)$)
        $\pi \leftarrow tail(\pi)$
        $B \leftarrow brf(B, perception())$
        if $\neg sound(\pi, I, B)$ **then**
            $\pi \leftarrow plan(B, I, A)$;

revise commitment to plan – re-planning for context adaptation

BDI architecture [Wooldridge, 2009]

while true **do**
    $B \leftarrow brf(B, perception())$
    $D \leftarrow options(B, I)$
    $I \leftarrow filter(B, D, I)$
    $\pi \leftarrow plan(B, I, A)$
    while $\pi \neq \emptyset$ and $\neg succeeded(I, B)$ and $\neg impossible(I, B)$ **do**
        execute($head(\pi)$)
        $\pi \leftarrow tail(\pi)$
        $B \leftarrow brf(B, perception())$
        if $\neg sound(\pi, I, B)$ **then**
            $\pi \leftarrow plan(B, I, A)$;

revise commitment to intentions – Single-Minded Commitment

BDI architecture [Wooldridge, 2009]

```
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

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

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

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

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

Hello World

intention revision

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

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

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

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

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)).
```

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

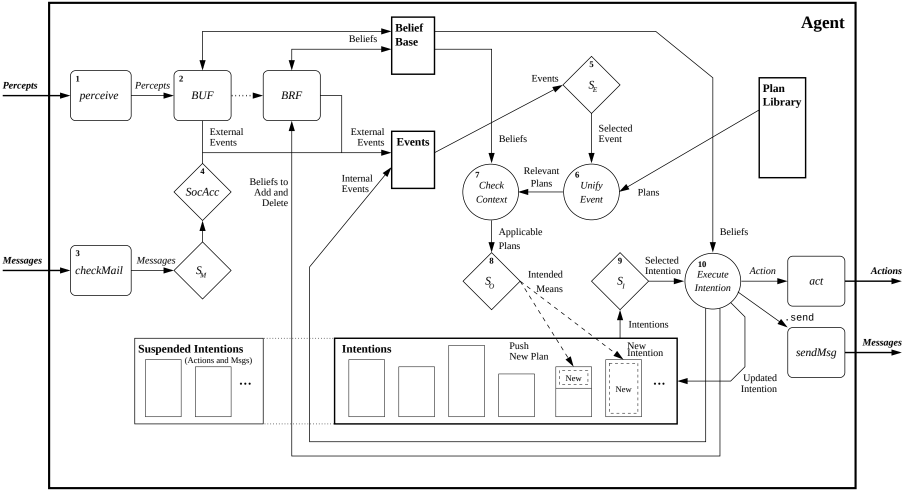
Intentions: plans instantiated to achieve some goal

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



Outline

Beliefs — Representation

Syntax

Beliefs are represented by annotated literals of first order logic

functor(*term*$_1$, ..., *term*$_n$)[*annot*$_1$, ..., *annot*$_m$]

Example (belief base of agent Tom)

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

Beliefs — Dynamics I

by perception

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

by intention

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

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

Beliefs — Dynamics II

by communication

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

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

Goals — Representation

Types of goals

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

Syntax

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

Example (Initial goal of agent Tom)

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

Goals — Dynamics I

by intention

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

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

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

Goals — Dynamics II

by communication – achievement goal

when an agent receives an achieve message, the content is a new achievement goal annotated with the sender of the message

```
.send(tom,achieve,write(book)); // sent by Bob
// adds new goal write(book)[source(bob)] for Tom
...
.send(tom,unachieve,write(book)); // sent by Bob
// removes goal write(book)[source(bob)] for Tom
```

Goals — Dynamics III

by communication – test goal

when an agent receives an askOne or askAll message, the content is a new test goal annotated with the sender of the message

```
.send(tom,askOne,published(P),Answer); // sent by Bob
// adds new goal ?publisher(P)[source(bob)] for Tom
// the response of Tom will unify with Answer
```

Triggering Events — Representation

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

+b	(belief addition)
-b	(belief deletion)
+!g	(achievement-goal addition)
-!g	(achievement-goal deletion)
+?g	(test-goal addition)
-?g	(test-goal deletion)

Plans — Representation

An AgentSpeak plan has the following general structure:

```
triggering_event : context <- body.
```

where:

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

Plans — Operators for Plan **Context**

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

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

Plans — Operators for Plan **Body**

```
+rain :  time_to_leave(T) & clock.now(H) & H >= T
   <- !g1;         // new sub-goal
      !!g2;        // new goal
      ?b(X);       // new test goal
      +b1(T-H);    // add mental note
      -b2(T-H);    // remove mental note
      -+b3(T*H);   // update mental note
      jia.get(X);  // internal action
      X > 10;      // constraint to carry on
      close(door);// external action
      !g3[hard_deadline(3000)].   // goal with deadline
```

Plans — Example

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

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

Plans — Dynamics

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

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

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

Other Language Features

Strong Negation

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

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

Plan Annotations

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

Example (an annotated plan)

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

Failure Handling: Contingency Plans

Example (an agent blindly committed to g)

```
+!g : g.

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

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

Meta Programming

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

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

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

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

Internal Actions

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

- Internal actions can be defined by the user in Java

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

Standard Internal Actions

- Standard (pre-defined) internal actions have an empty library name
  - `.print(`*term*$_1$`,`*term*$_2$`,...)`
  - `.union(`*list*$_1$`,` *list*$_2$`,` *list*$_3$`)`
  - `.my_name(`*var*`)`
  - `.send(`*ag*`,`*perf*`,`*literal*`)`
  - `.intend(`*literal*`)`
  - `.drop_intention(`*literal*`)`
- Many others available for: printing, sorting, list/string operations, manipulating the beliefs/annotations/plan library, creating agents, waiting/generating events, etc.

Outline

Jason × Java

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

Java code – go to gold

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

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

            doAction(go(a));

} } }   }
```

Java code – charge battery

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

Jason code

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

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

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

Jason × Prolog

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

Outline

Communication **Infrastructure**

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

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

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

Jade: distributed agents, FIPA-ACL

...: others defined by the user (e.g. AgentScape)

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
  - $\mathcal{M}$oise
  - MADEM
  - Ontologies
  - ...
- More on `http://jason.sourceforge.net/wp/projects/`

Outline

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.

$\leadsto$ 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

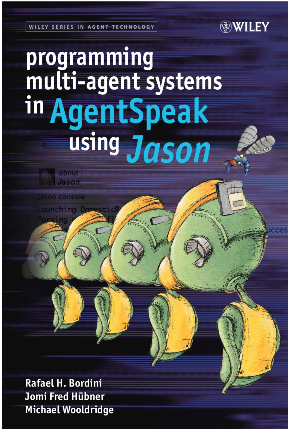
- AgentSpeak
  - Logic + BDI
  - Agent programming language
- *Jason*
  - AgentSpeak interpreter
  - Implements the operational semantics of AgentSpeak
  - Speech-act based communicaiton
  - Highly customisable
  - Useful tools
  - Open source
  - Open issues

Acknowledgements

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

Further Resources

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



Environment Oriented Programming

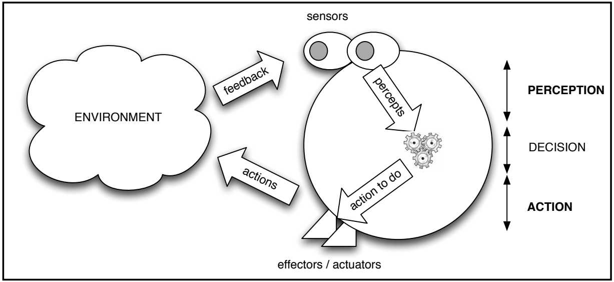
— **EOP** —

Outline

Back to the Notion of Environment in MAS

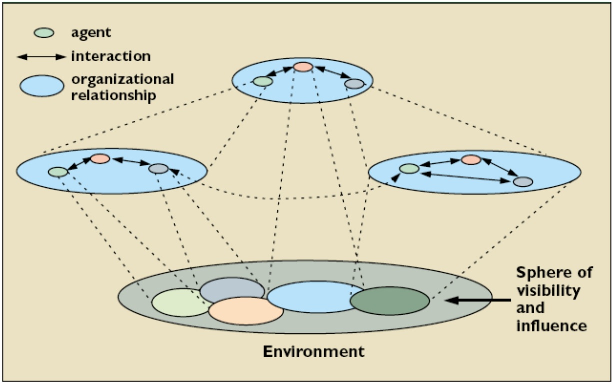
- The notion of environment is intrinsically related to the notion of agent and multi-agent system
  - "An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective" [Wooldridge, 2002]
  - "An agent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

Single Agent Perspective



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

Multi-Agent Perspective



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

Why Environment Programming

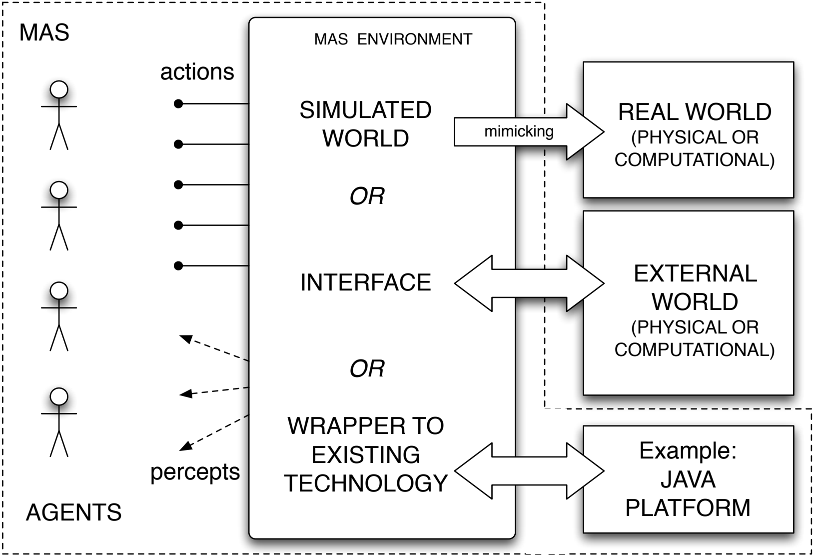
- Basic level
  - to create testbeds for real/external environments
  - to ease the interface/interaction with existing software environments
- Advanced level
  - to uniformly encapsulate and modularise functionalities of the MAS out of the agents
    - typically related to interaction, coordination,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

Environment Programming: General Issues

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

Outline

Basic Level Overview

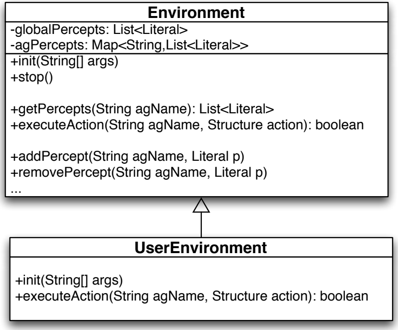


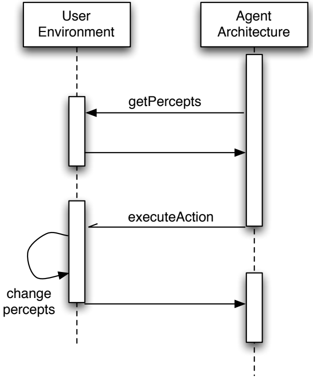
Basic Level: Features

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

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

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





MARS Environment in *Jason*

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

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

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

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

```
  ...

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

    clearPercepts();

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

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

    addPercept(pos1);
    addPercept(pos2);

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

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

  class MarsModel extends GridWorldModel { ... }

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

Jason Agents Playing on Mars

```
// mars robot 1

/* Initial beliefs */

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

/* Initial goal */

!check(slots).

/* Plans */

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

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

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

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

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

```
...

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

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

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

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

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

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

```
package blockworld;

public class Env extends apapl.Environment {


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

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

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

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

  ...

}
```

2APL Agents in the block-world

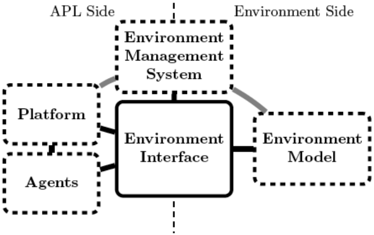
```
BeliefUpdates:
  { bomb(X,Y) }          RemoveBomb(X,Y){ not bomb(X,Y) }
  { true }               AddBomb(X,Y)   { bomb(X,Y) }
  { carry(bomb) }        Drop( )        { not carry(bomb)}
  { not carry(bomb) }    PickUp( )      { carry(bomb) }

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

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

Goals:
  clean( blockWorld )

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

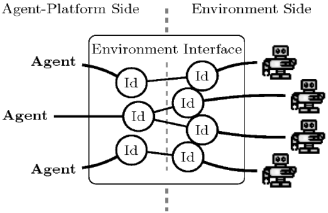
```
...

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

  ...
```

Environment Interface Stardard – EIS Initiative

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

EIS Meta-Model





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

Environment Interface Features

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

Outline

Advanced Level Overview

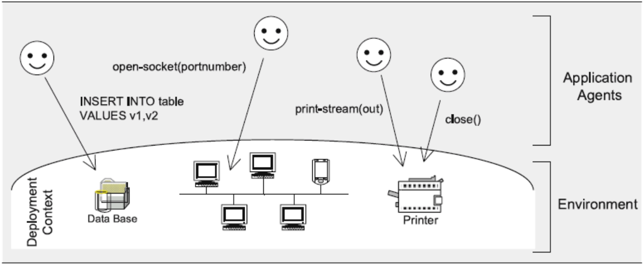
- Vision: environment as a first-class abstraction in MAS [Weyns et al., 2007, Ricci et al., 2011]
  - application or endogenous environments, i.e. that environment which is an explicit part of the MAS
  - providing an exploitable design & programming abstraction to build MAS applications
- Outcome
  - 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

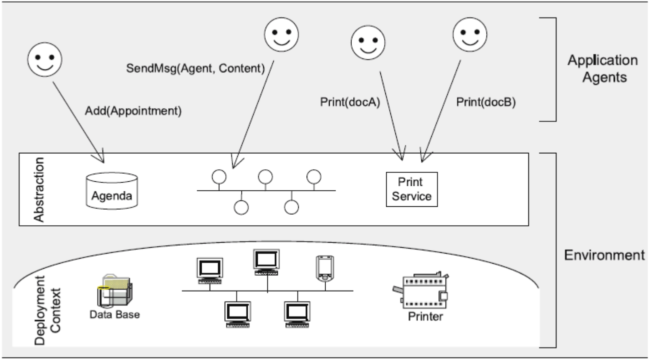
Basic Interface Support

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



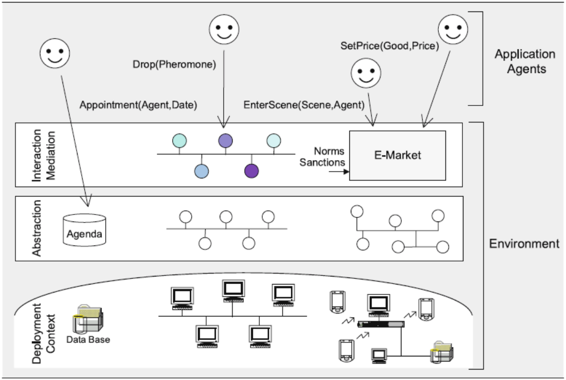
Abstraction Support

- 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 / 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

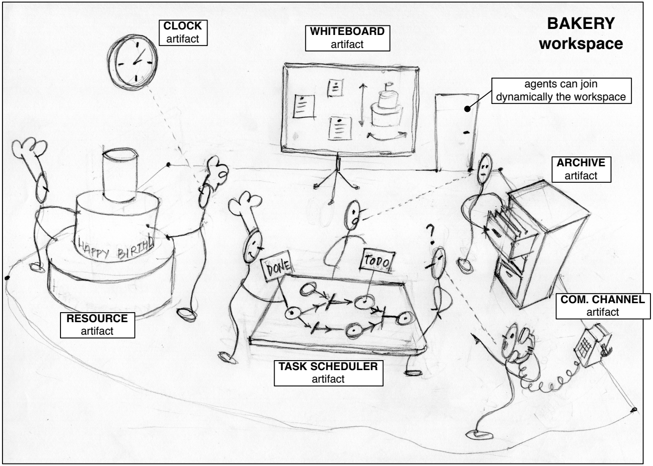
- AGRE / AGREEN / MASQ [Stratulat et al., 2009]
  - AGRE – integrating the AGR (Agent-Group-Role) organisation model with a notion of environment
    - Environment used to represent both the physical and social part of interaction
  - AGREEN / MASQ – extending AGRE towards a unified representation for 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

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



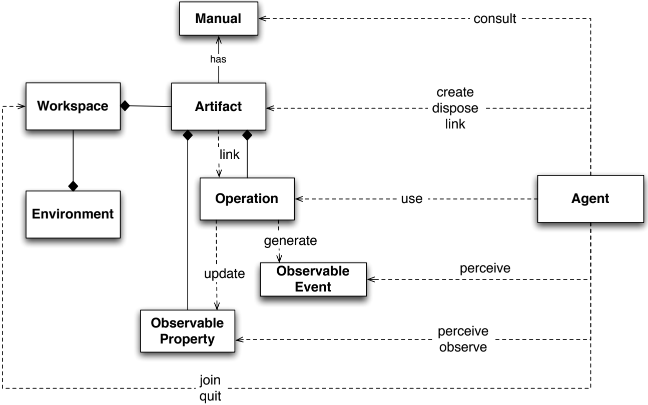
A&A Basic Concepts [Omicini et al., 2008]

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

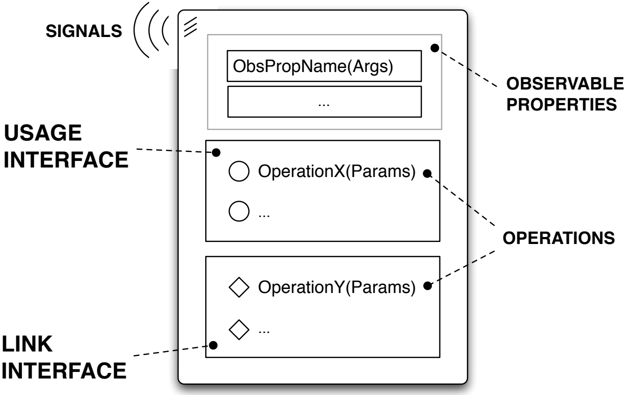
A&A Programming Model Features [Ricci et al., 2007b]

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

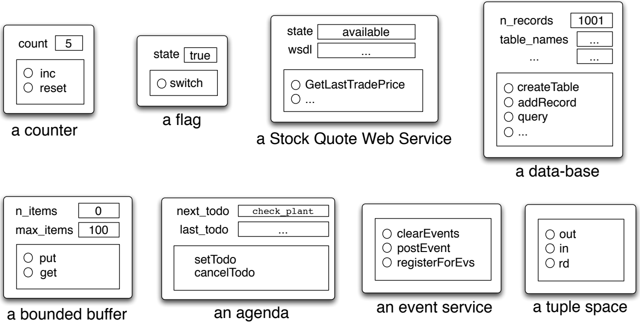
A&A Meta-Model in More Detail [Ricci et al., 2011]



Artifact Abstract Representation



A World of Artifacts



A Simple Taxonomy

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

Actions and Percepts in Artifact-Based Environments

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

actions ⟷ artifacts' operation

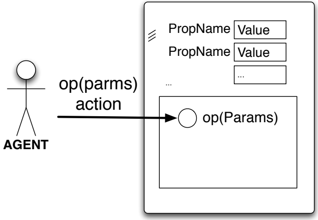
the action repertoire is given by the dynamic set of operations provided by the overall set of artifacts available in the workspace can be changed by creating/disposing artifacts

- action success/failure semantics is defined by operation semantics

percepts ⟷ artifacts' observable properties + signals

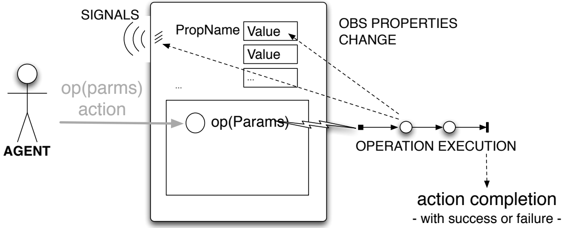
properties represent percepts about the state of the environment signals represent percepts concerning events signalled by the environment

Interaction Model: Use



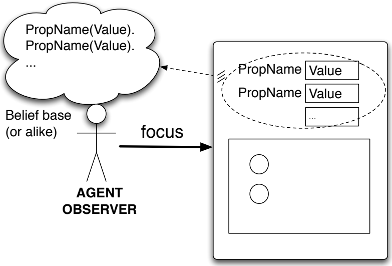
- Performing an action corresponds to triggering the execution of an operation
  - acting on artifact's usage interface

Interaction Model: Operation execution



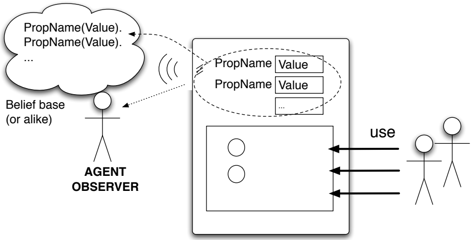
- a process structured in one or multiple transactional steps
- asynchronous with respect to agent
  - ...which can proceed possibly reacting to percepts and executing actions of other plans/activities
- operation completion causes action completion
  - action completion events with success or failure,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

Interaction Model: Observation



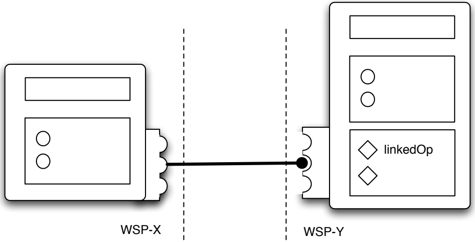
- Agents can dynamically select which artifacts to observe
  - predefined `focus`/`stopFocus` actions

Interaction Model: Observation



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

Artifact Linkability



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

Artifact Manual

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

CArtAgO

- Common ARtifact infrastructure for AGent Open environment (CArtAgO) [Ricci et al., 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
  - available at `http://cartago.sourceforge.net`

Integration with Agent Languages and Platforms

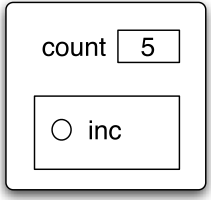
- Integration with existing agent platforms [Ricci et al., 2008]
  - by means of bridges creating an action/perception interface and doing data binding
- Outcome
  - developing open and heterogenous MAS
  - introducing a further perspective on interoperability besides the ACL's one
    - sharing and working in a common work environment
    - common object-oriented data-model

JaCa Platform

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

Example 1: A Simple Counter Artifact

```
class Counter extends Artifact {

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

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



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

Example 1: User and 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).
```

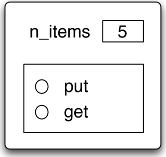
- Working with the shared counter

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

```
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

CLOCK

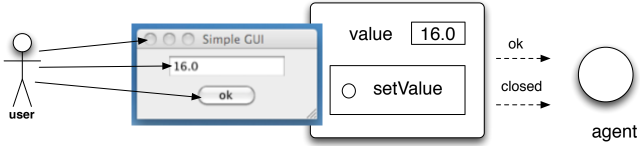
```
public class Clock extends Artifact {

  boolean working;
  final static long TICK_TIME = 100;

  void init(){ working = false; }

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

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

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

CLOCK USER AGENT

```
!test_clock.

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

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

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

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

Example 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

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

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

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

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

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

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

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

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

USER ASSISTANT AGENT

```
!test_gui.

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

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

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

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

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]
    - `http://jaca-android.sourceforge.net`
  - JaCa-WS / CArtAgO-WS
    - building SOA/Web Services applications using JaCa [Ricci et al., 2010a]
    - `http://cartagows.sourceforge.net`
  - JaCa-Web
    - implementing Web 2.0 applications using JaCa
    - `http://jaca-web.sourceforge.net`

Outline

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]
  $\leadsto$ Organisations are supra-individual phenomena
- A decision and communication schema which is applied to a set of actors that together fulfill a set of tasks in order to satisfy goals while guarantying a global coherent state [Malone, 1999]
  $\leadsto$ 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]
  $\leadsto$ 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]
  $\leadsto$ pattern of emergent cooperation

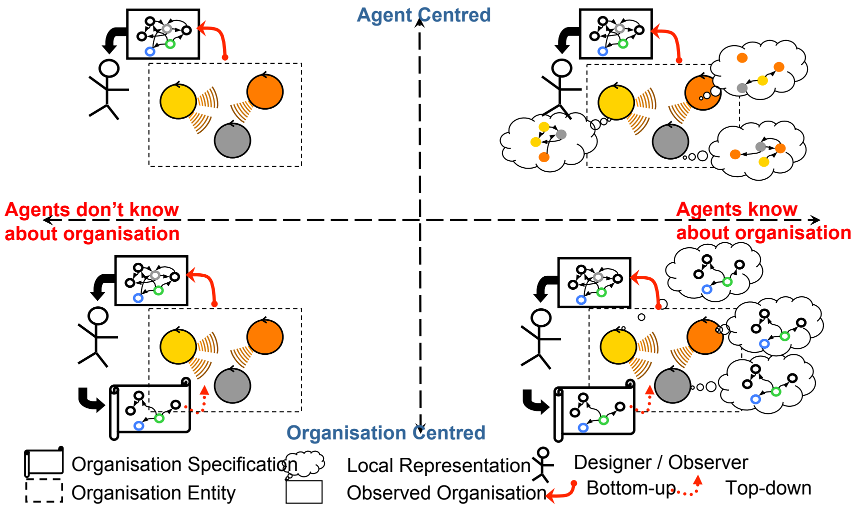
Organisation in MAS

Definition

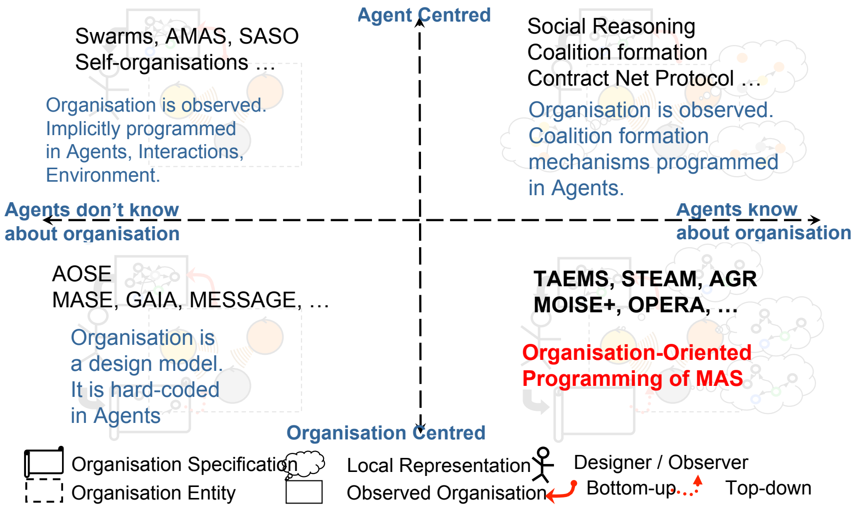
Purposive supra-agent pattern of emergent or (pre)defined agents cooperation, that could be defined by the designer or by the agents themselves.

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

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



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



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

$\leadsto$ Normative Organisations

Norms

Norm

Norms are rules that a society has in order to influence the behaviour of agents.

Norm mechanisms

- 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

  $\leadsto$ Detection of violations, decision about ways of enforcing the norms (e.g. sanctions)

Normative Multi-Agent Organisation

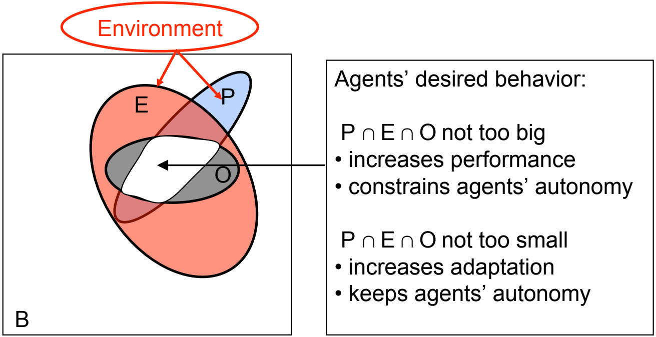
Normative Multi-Agent System [Boella et al., 2008]

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



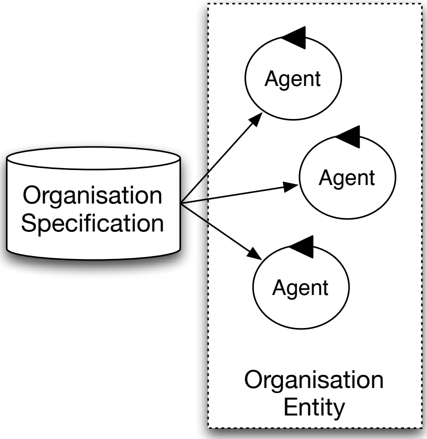
- B: agents' possible behaviors
- P: agents' behaviors that lead to global purpose
- E: agents' possible behaviors constrained by the environment
- O: agents' possible/permitted/obliged behaviors constrained by the normative organisation

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
      $\leadsto$ possibly inconsistant with the other agents' representations
    - Global/local representation in the MAS
      $\leadsto$ 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

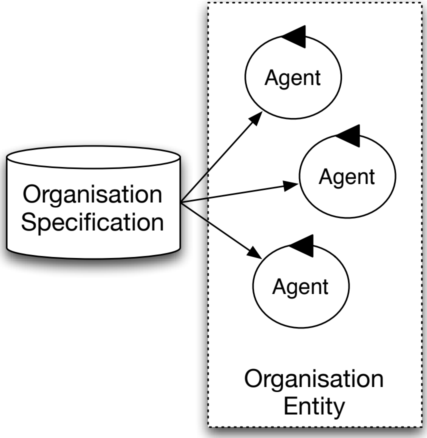
Organisation Oriented Programming (OOP)



- 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

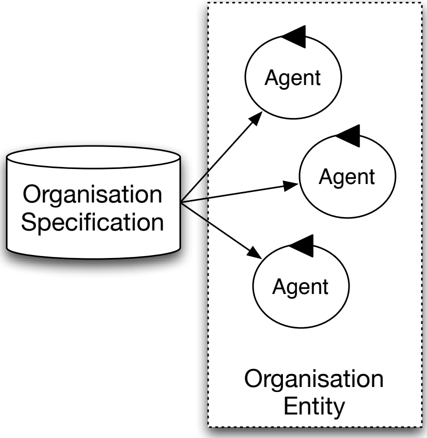
Organisation Oriented Programming (OOP)



First approach

- Agents read the program and follow it

Organisation Oriented Programming (OOP)



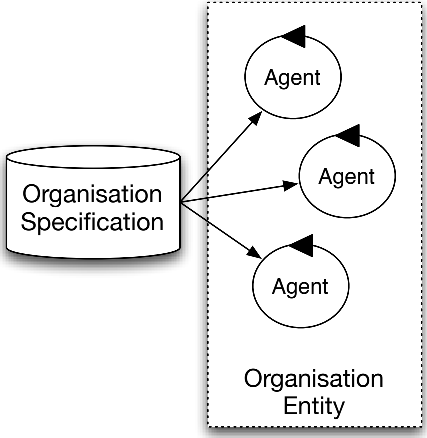
First approach

- 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)



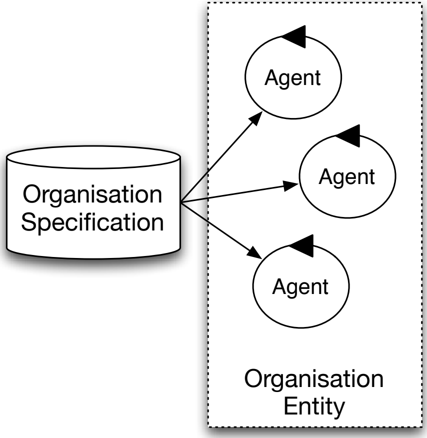
First approach

- 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],
  $\mathcal{M}$oise$^+$ [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],
  $\mathcal{S}$-$\mathcal{M}$oise^{+} [Hübner et al., 2006],
  ORA4MAS [Hübner et al., 2009],
  ...

- Adaptation mechanisms, i.e. reorganisation infrastructure

Components of OOP: Integration mechanisms

- Agent integration mechanisms allow agents to be aware of and to deliberate on:
  - entering/exiting the organisation
  - modification of the organisation
  - obedience/violation of norms
  - sanctioning/rewarding other agents

  e.g. $\mathcal{J}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2007], Autonomy based reasoning [Carabelea, 2007], *ProsA*$_2$ Agent-based reasoning on norms [Ossowski, 1999], ...

- Environment integration mechanisms transform organisation into embodied organisation so that:
  - organisation may act on the environment (e.g. enact rules, regimentation)
  - environment may act on the organisation (e.g. count-as rules)

  e.g [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
  - $\leadsto$ Organisation is no more closed
- to change/adapt the current organisation
  - $\leadsto$ Organisation is no more static
- to obey/disobey the organisation
  - $\leadsto$ 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:

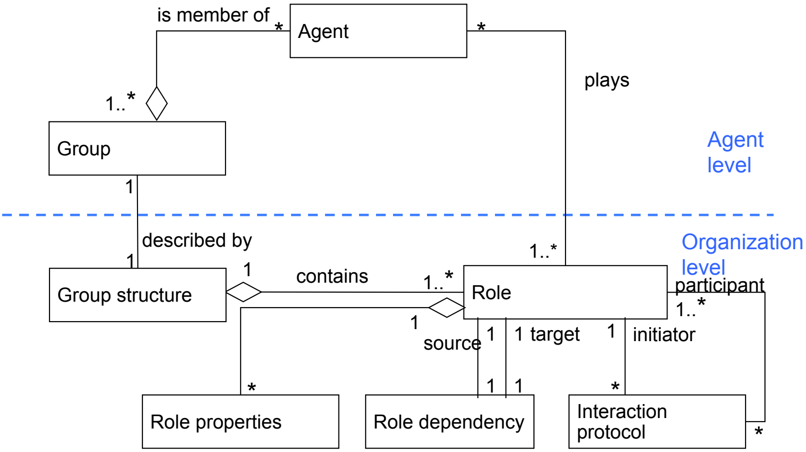
- to decide to let agents enter into/leave from the organisation during execution
  - ⇝ Organisation is no more closed
- to decide to let agents change/adapt the current organisation
  - ⇝ Organisation is no more static and blind
- to govern agents behaviour in the organisation (i.e. monitor, enforce,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

Outline

AGR [Ferber and Gutknecht, 1998]

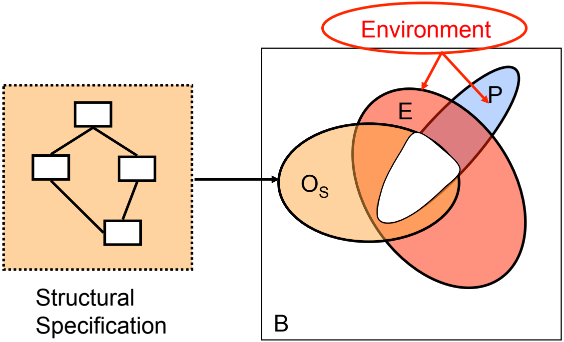
- Agent Group Role, previously known as AALAADIN
  - Agent: Active entity that plays roles within groups. An agent may have several roles and may belong to several groups.
  - Group: set of agents sharing common characteristics, i.e. context for a set of activities. Two agents can't communicate with 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



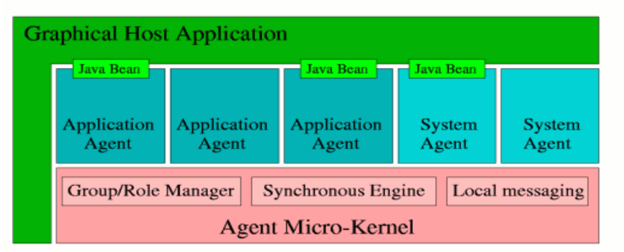
is member of
*
Agent
*
plays
1..*
Group
1
Agent
level
described by
1
Organization
level
1..*
1
contains
1..*
Group structure
Role
participant
1..*
1
source
1
1 target
1
initiator
*
1
1
*
Role properties
Role dependency
Interaction
protocol
*

AGR OML Modelling Dimensions



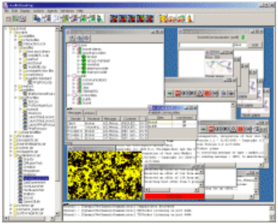
B: agents' possible behaviors
P: agents' behaviors that lead to global purpose
E: agents' possible behaviors constrained by the environment
O_S: agents' possible behaviors structurally constrained by the organization

AGR OMI: Madkit



Multi-Agent Development Kit

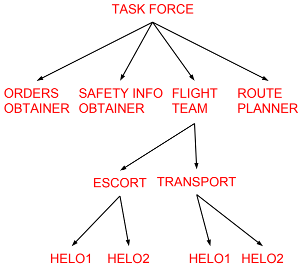
www.madkit.org



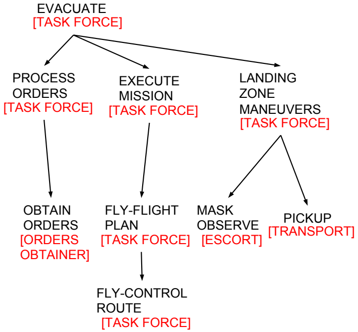
STEAM [Tambe, 1997]

- 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.

STEAM OML [Tambe, 1997]



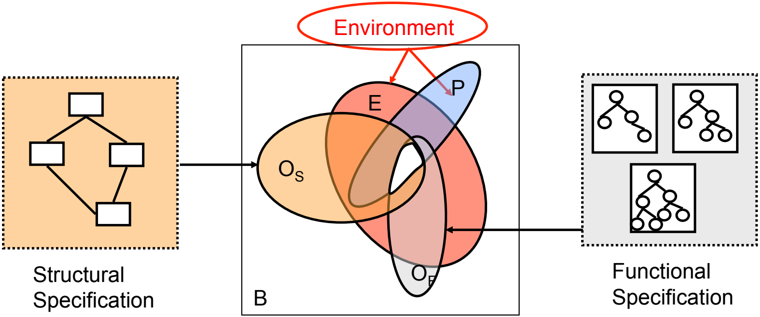
Organization: hierarchy of roles that may be filled by agents or groups of agents.



Team Plan:

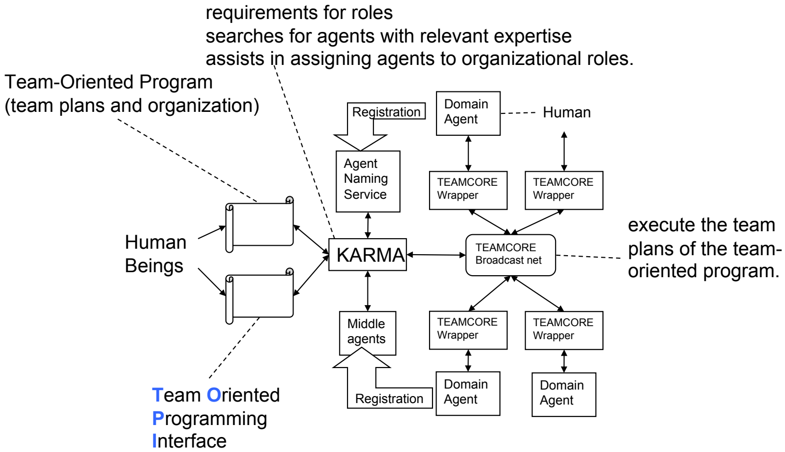
- initial conditions,
- term. cond. : achievability, irrelevance, unachievability
- team-level actions.

STEAM OML Modelling Dimensions



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

STEAM OMI: TEAMCORE [Pynadath and Tambe, 2003]



ISLANDER

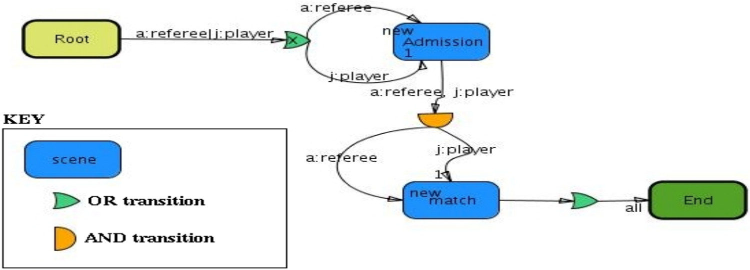
- Based on different influences: economics, norms, dialogues, coordination

⇝ electronic institutions

- Combining different alternative views: dialogical, normative, coordination
- Institution Description Language:
  - Performative structure (Network of protocols),
  - Scene (multi-agent protocol),
  - Roles,
  - Norms
- Ameli as OMI

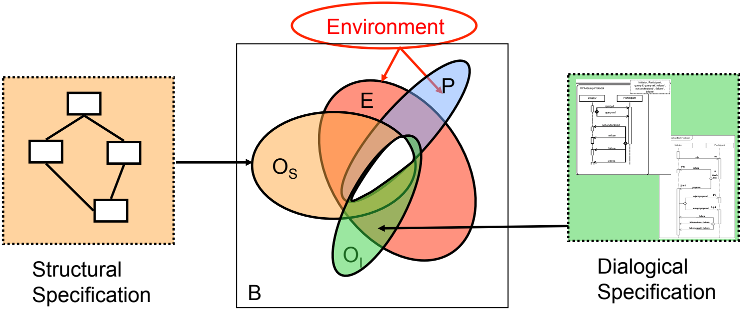
ISLANDER OML: IDL [Esteva et al., 2001]

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



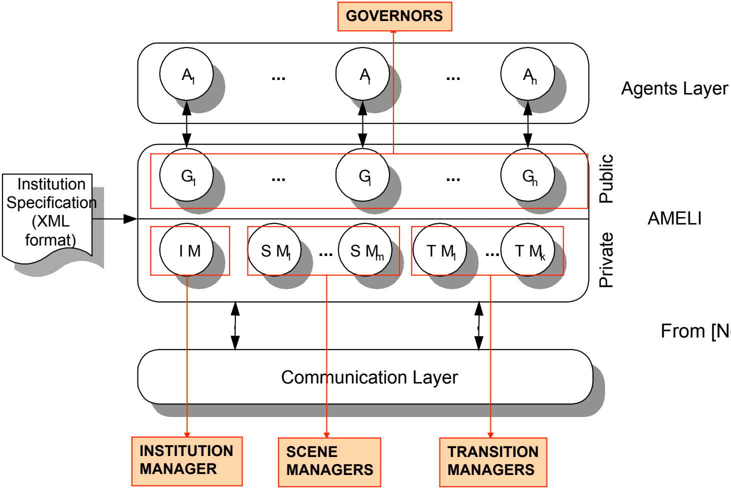
Performative Structure

ISLANDER 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]



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

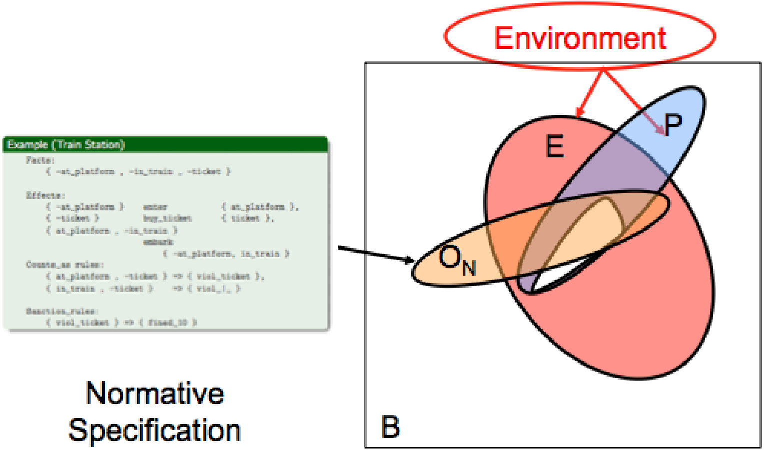
- determines effect of external actions
- normatively assesses effect of agents' actions (monitoring)
- sanctions agents' wrongdoings (enforcement)
- prevents ending up in really bad states (regimentation)

Programming Language for Organisations

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



Summary

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

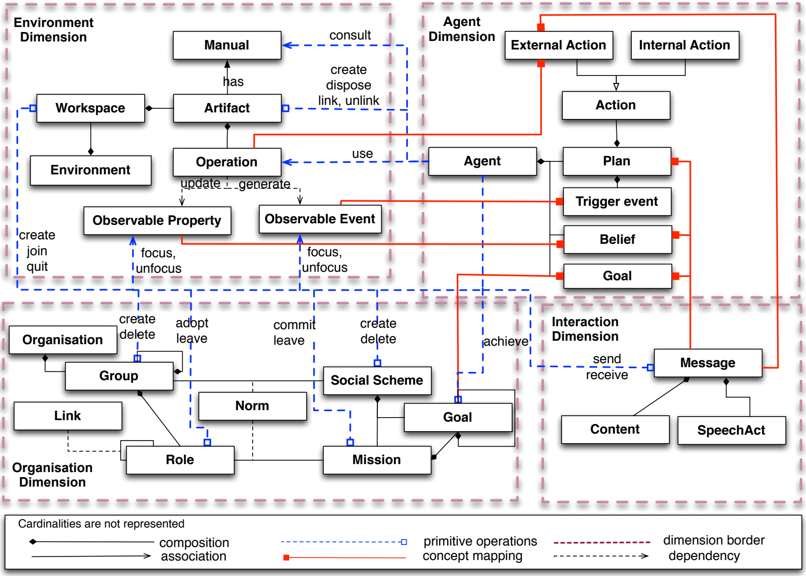
$\mathcal{M}$oise

(let's go programming those nice concepts)

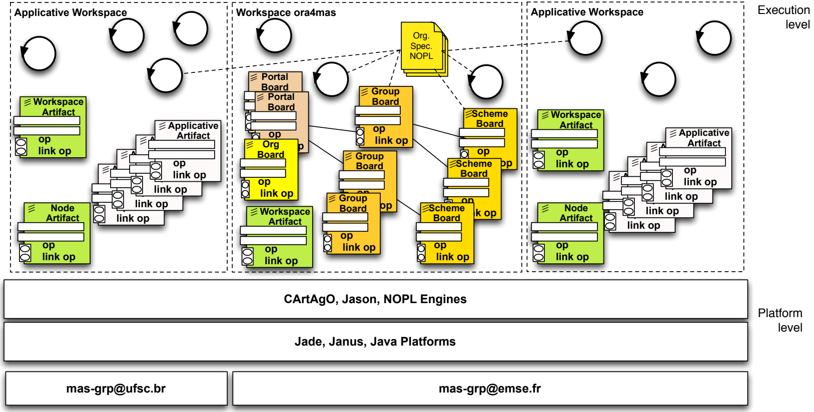
$\mathcal{M}$oise Framework

- OML (language)
  - Tag-based language
    (issued from $\mathcal{M}$oise [Hannoun et al., 2000],
    $\mathcal{M}$oise$^+$ [Hübner et al., 2002], $\mathcal{M}$oiseInst [Gâteau et al., 2005])
- OMI (infrastructure)
  - developed as an artifact-based working environment
    (ORA4MAS [Hübner et al., 2009] based on CArtAgO nodes,
    refactoring of $\mathcal{S}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2006] and
    $\mathcal{S}$ynai [Gâteau et al., 2005])
- Integrations
  - Agents and Environment (c4Jason, c4Jadex [Ricci et al., 2009b])
  - Environment and Organisation ([Piunti et al., 2009a])
  - Agents and Organisation ($\mathcal{J}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2007])

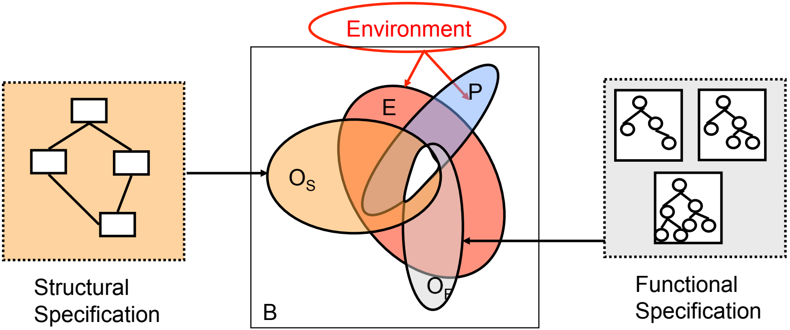
Moise in JaCaMo Metamodel



Moise Framework in JaCaMo



$\mathcal{M}$oise Modelling Dimensions



Groups, links, roles
Compatibilities, multiplicities
inheritance

Normative Specification
Permissions, Obligations
Allows agents autonomy!

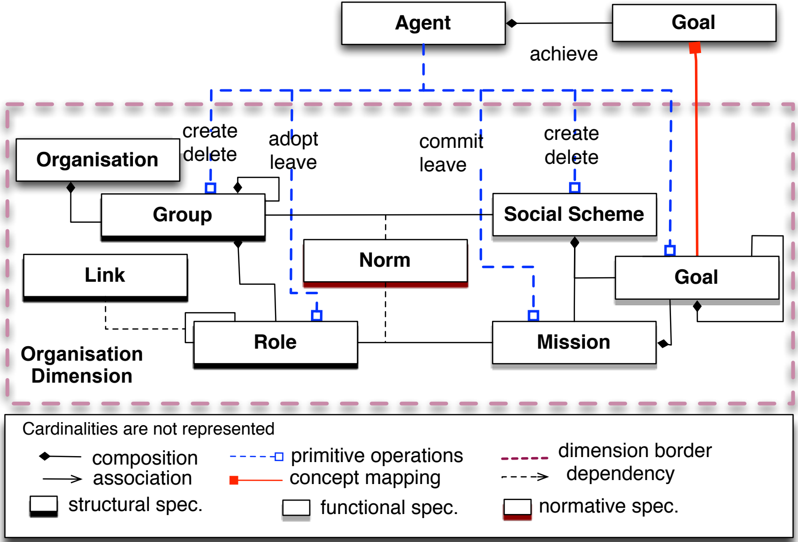
Global goals, plans,
Missions, schemas,
preferences

Outline

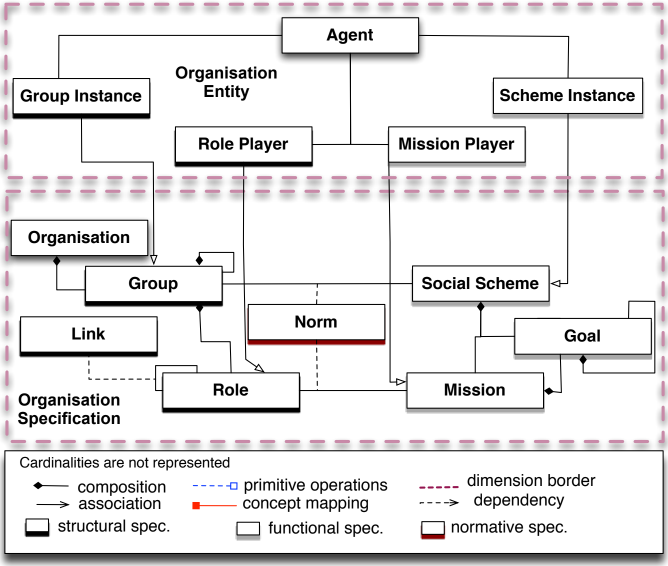
$\mathcal{M}$oise OML

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

$\mathcal{M}$oise OML meta-model (partial & simplified view)



Moise OML global picture



Structural Specification

- Specifies the structure of an MAS along three levels:
  - Individual with Role
  - Social with Link
  - Collective with Group
- Components:
  - Role: label used to assign constraints on the behavior of agents playing it
  - Link: relation between roles that directly constrains the agents in their interaction with the other agents playing the corresponding roles
  - Group: set of links, roles, compatibility relations used to define a shared context for agents playing roles in it

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

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

Role specification

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

Example

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

Group specification

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

Example

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

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

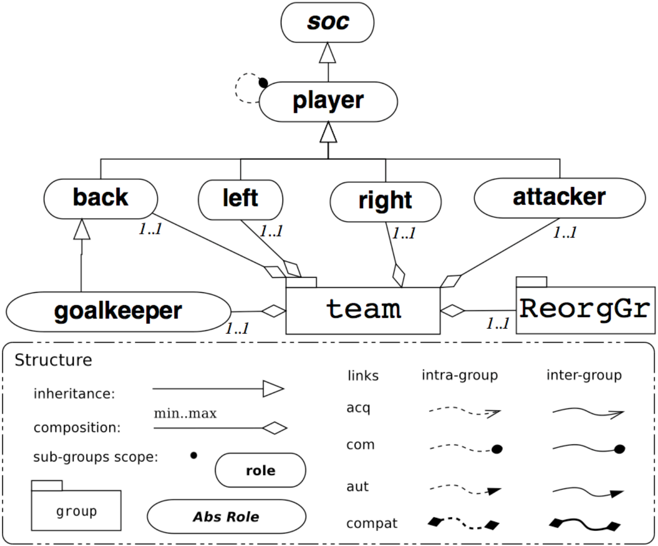
Formation constraint specification

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

Example

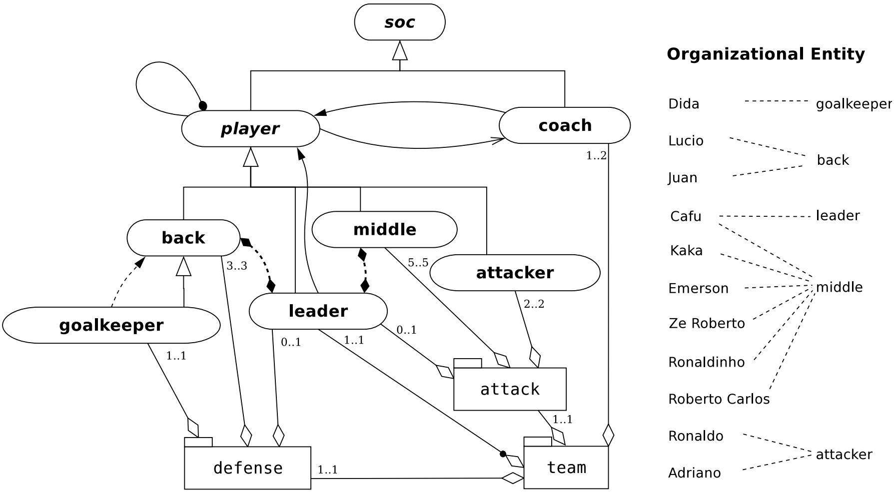
```
<formation-constraints>
  <compatibility from="middle"
                 to="leader"
                 scope="intra-group"
                 extends-subgroups="false"
                 bi-dir="true"/>
  ...
</formation-constraints>
```

Structural specification example (1)



Graphical representation of structural specification of Joj Team

Structural specification example (2)



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

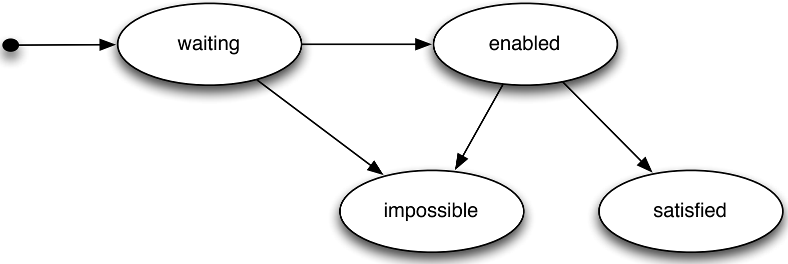
Example

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

Scheme specification

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

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>
    <goal id="g3" min="1" ds="kick the ball to the m2Ag" >
       <argument id="M2Ag" />
    </goal>
    <goal id="g4"       min="1" ds="go to the opponent's back line" />
    <goal id="g5"       min="1" ds="kick the ball to the goal area" />
    <goal id="g6"       min="1" ds="shot at the opponent's goal" />
  </plan>
 </goal>
 ...
```

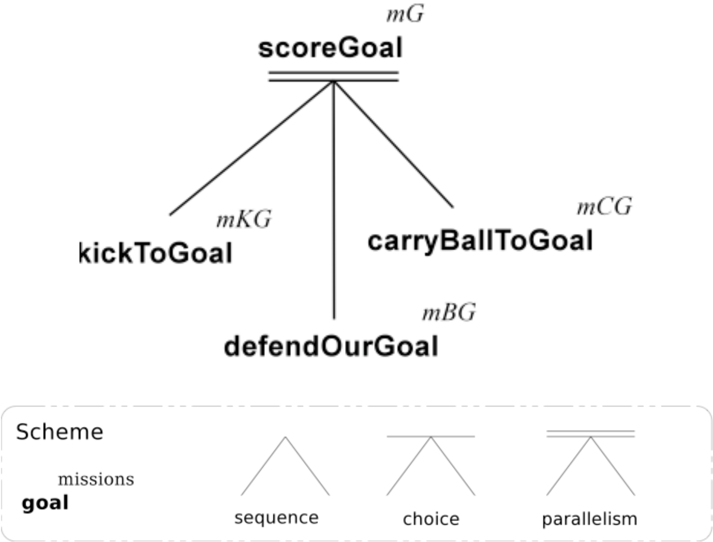
Mission specification

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

Example

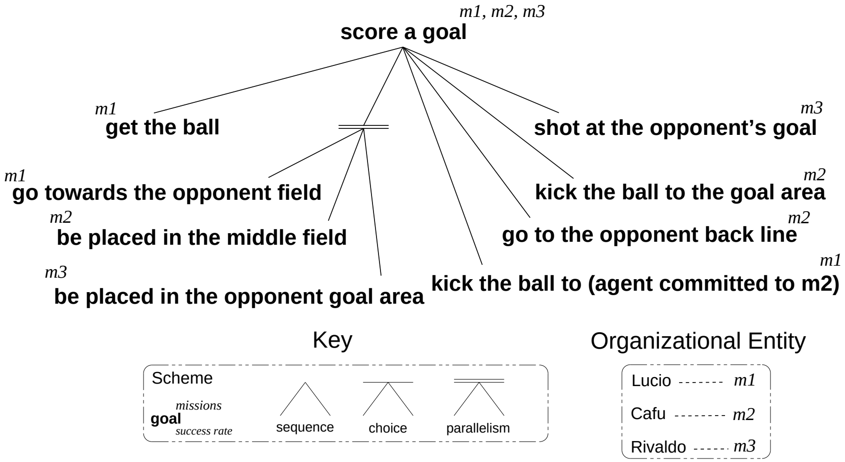
```
<scheme id="sideAttack">
  ... the goals ...
 <mission id="m1" min="1" max="1">
    <goal id="scoreGoal" /> <goal id="g1" />
    <goal id="g3" /> ...
  </mission>
  ...
</scheme>
```

Functional specification example (1)



Graphical representation of social scheme for joj team

Functional specification example (2)



Graphical representation of social scheme "side_attack" for joj team

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 specification

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

Norm Specification – example

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

```
<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

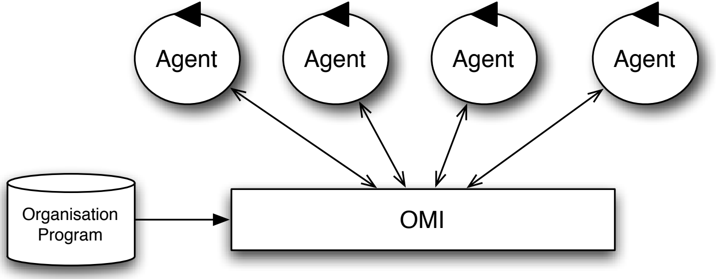
1. Organisation is created (by the agents)
    - instances of groups
    - instances of schemes
2. Agents enter into groups adopting roles
3. When a group is well formed, it may become responsible for schemes
    - Agents from the group are then obliged to commit to missions in the scheme
4. Agents commit to missions
5. Agents fulfil mission's goals
6. Agents leave schemes and groups
7. Schemes and groups instances are destroyed

Outline

Organisation management infrastructure (OMI)

Responsibility

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



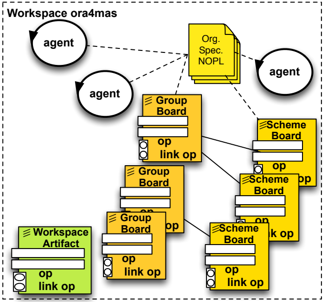
(e.g. MadKit, AMELI, $\mathcal{S}$-$\mathcal{M}$oise^{+}, ...)

ORA4MAS

Based on A&A and $\mathcal{M}$oise, Agents' working environment is instrumented with Organizational Artifacts (OA) offering "organizational" actions

$\rightsquigarrow$ Distributed management of the organization with a clear separation of concerns:

- Agents:
  - create, handle OAs and act on them
    $\rightsquigarrow$ 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

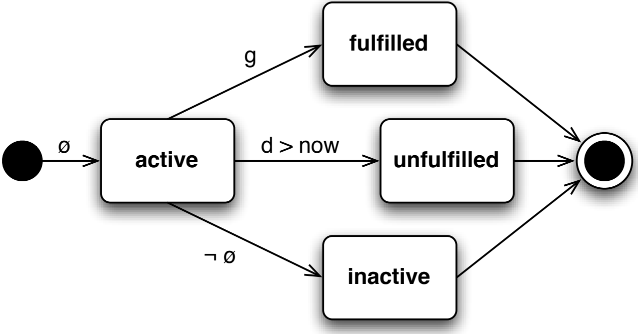
Example (Norm)

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

or

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

Obligations life cycle



$$norm\ n : \ \phi -> \ obligation(a, r, g, d)$$

- ϕ: 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

Structural Operational Semantics

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

Rules for Norm Management

- Failure detection:

$$\frac{n \in N \qquad F \models n_\varphi \qquad n_\psi = \texttt{fail}(\_)}{\langle F, N, \top, OS, t\rangle \longrightarrow \langle F, N, \bot, OS, t\rangle} \qquad (\textbf{Regim})$$

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

- Roll back from failure:

$$\frac{\forall n \in N.(F \models n_\varphi \implies n_\psi \neq \texttt{fail}(\_))}{\langle F, N, \bot, OS, t\rangle \longrightarrow \langle F, N, \top, OS, t\rangle} \qquad (\textbf{Consist})$$

Rules for Norm Management (continued)

- Creation of obligation:

$$\frac{n \in N \qquad F \models n_\varphi \qquad n_\psi = o \qquad o\theta_d > t \qquad \neg\exists\langle o', ost\rangle \in OS\ .\ (o' \stackrel{\text{obl}}{=} o\theta \land ost \neq \textbf{inactive})}{\langle F, N, \top, OS, t\rangle \longrightarrow \langle F, N, \top, OS \cup \langle o\theta, \textbf{active}\rangle, t\rangle} \quad (\textbf{Oblig})$$

where θ is the m.g.u. such that $F \models o\theta$

Rules for Obligation Management

$$\frac{os \in OS \qquad os = \langle o, \textbf{active} \rangle \qquad F \models o_g \qquad o_d \geq t}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \textbf{fulfilled} \rangle\}, t \rangle} \quad (\textbf{Fulfil})$$

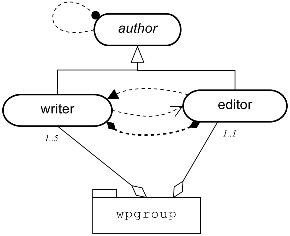
$$\frac{os \in OS \qquad os = \langle o, \textbf{active} \rangle \qquad o_d < t}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \textbf{unfulfilled} \rangle\}, t \rangle} \quad (\textbf{Unfulfil})$$

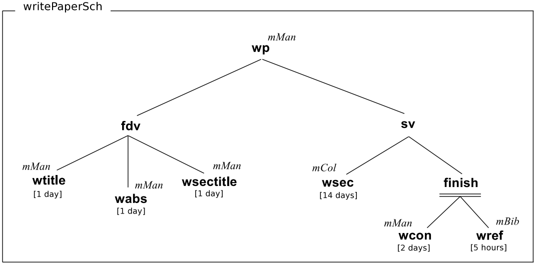
$$\frac{os \in OS \qquad os = \langle o, \textbf{active} \rangle \qquad F \not\models o_r}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \textbf{inactive} \rangle\}, t \rangle} \quad (\textbf{Inactive})$$

NOPL

Normative Organisation Programming Language

- NOPL is a particular class of NPL: facts, rules and norms are specific to a OML (eg. $\mathcal{M}$oise NOML):





id	condition	role	type	mission	TTF
n2		writer	obl	*mCol*	1 day
n3		writer	obl	*mBib*	1 day
n4	unfulfilled(n2)	editor	obl	*ms*	3 hours
n5	fulfilled(n3)	editor	obl	*mr*	3 hours
n6	#gnc	editor	obl	*ms*	3 hours
n7	#rc	editor	obl	*ms*	30 minutes
n6	#mc	editor	obl	*ms*	1 hour
…	…	…	…	…	…

#gnc = goal_non_compliance
#rc = role_compatibility
#mc = mission_cardinality

OS in $\mathcal{M}$oise OML to NOPL translation

Example (role cardinality norm – regimentation)

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

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

Example (role cardinality norm – agent decision)

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

$\mathcal{M}$oise 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*.

$\mathcal{M}$oise Social scheme — NOPL — Rules

- Example of rules used to infer the state of the scheme:
  - Number of players of mission M in scheme S:

```
mplayers(M,S,V) :-
   .count(committed(_,M,S),V).
```

  - Wellformedness property of scheme S:

```
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):

```
ready(S,G) :-
   goal(_, G, PCG, _) & all_achieved(S,PCG).
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‘).
```

$\mathcal{M}$oise — NOPL — Norms

⇝ 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.

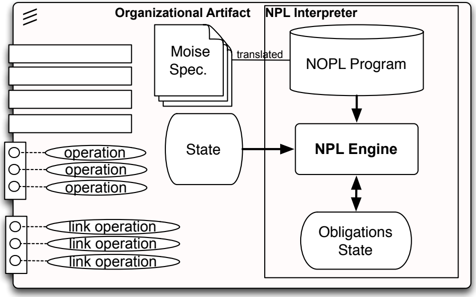
```
norm n2:
  plays(A,writer,Gr) & responsible(Gr,S) &
  mplayers(mCol,S,V) & V < 5
-> obligation(A,n2,committed(A,mCol,S),‘now‘+‘1 day‘).
```

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 $\mathcal{M}$oise programs
- generate signals
  - oblCreated(o), oblFulfilled(o), oblUnfulfilled(o)
  - oblInactive(o), normFailure(f)

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



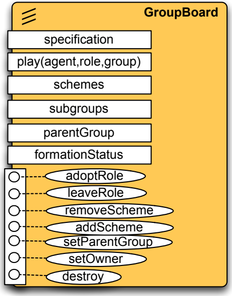
Generic control cycle of an Organisational Artifact

```
// 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.

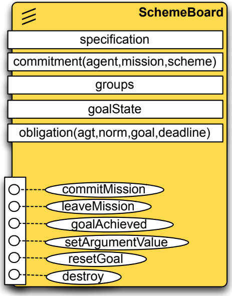
- Operations:
  - adoptRole(role) (resp. leaveRole(role)): attempts to adopt (resp. leave) role in the group
  - addScheme(schid) (resp. removeScheme(schid)): attempts to set (resp. unset) the group responsible for the scheme managed by the SchemeBoard schId
- Observable Properties:
  - specification: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



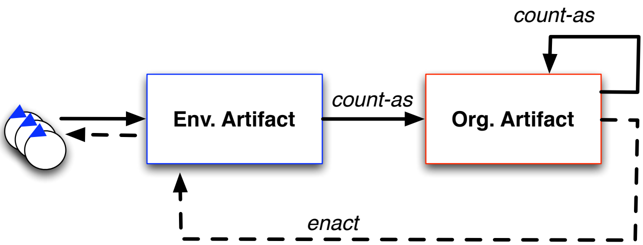
Partial Synthesis

- NPL, based on obligation and regimentation, formalised using operational semantics, specialised into NOPL
- Automatic translation of OS written in $\mathcal{M}$oise 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 $\mathcal{M}$oise OS have been translated to norms)
  - CArtAgO (10%): interface for agents
  - Java (10%): dynamic of organisational state

Outline

Environment integration

- 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

$\rightsquigarrow$ 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

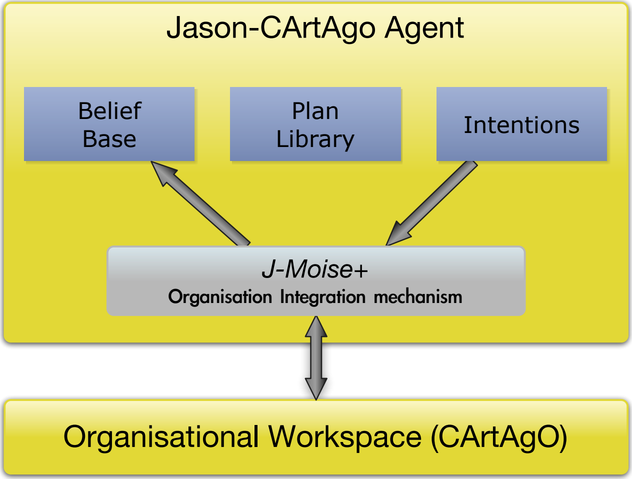
$\mathcal{J}$-$\mathcal{M}$oise: *Jason* + $\mathcal{M}$oise

- Agents are programmed with *Jason*

$\rightsquigarrow$ 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

$\mathcal{J}$-$\mathcal{M}$oise: *Jason* + $\mathcal{M}$oise– General view



Organisational **actions** in *Jason* I

Example (GroupBoard)

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

Organisational **actions** in *Jason* II

Example (SchemeBoard)

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

Organisational **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 focus 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 agent that achieved the goal, state of the goal)
- obligation(agent,norm,goal,dead line)
- normFailure(norm)

Organisational perception – example

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

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

Handling organisational **events** in *Jason*

Whenever something changes in the organisation, the agent architecture updates the agent belief base accordingly producing events (belief update from perception)

Example (new agent entered the group)

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

Example (change in goal state)

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

Example (signals)

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

Typical plans for obligations

Example

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

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

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

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>

     <group-specification id="wpgroup">
        <roles>
           <role id="writer" min="1" max="5" />
           <role id="editor" min="1" max="1" />
        </roles>
        ...
```

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)

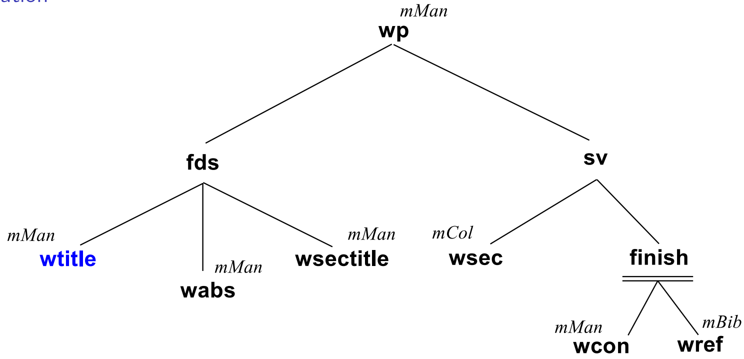
Writing paper sample III

Execution

all perception:
commitment(jaime,mManager,s1)
commitment(olivier,mColaborator,s1)
commitment(olivier,mBib,s1)
commitment(jomi,mColaborator,s1)

Writing paper sample IV

Execution

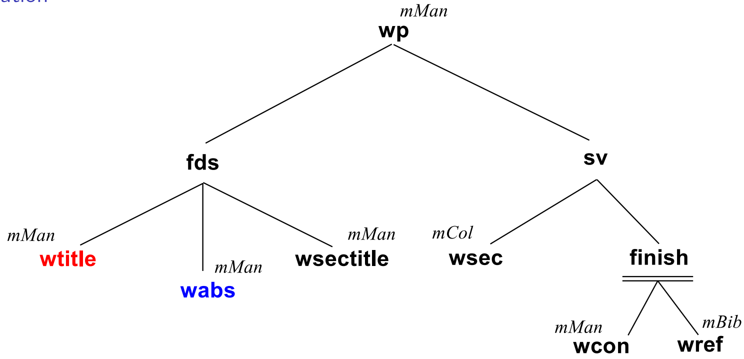


all perception: goal_state(s1,*,unsatisfied)

jaime (only wtitle is possible, Jaime should work)
event: +!wtitle
action: jmoise.set_goal_state(s1,wtitle,satisfied)

Writing paper sample V

Execution

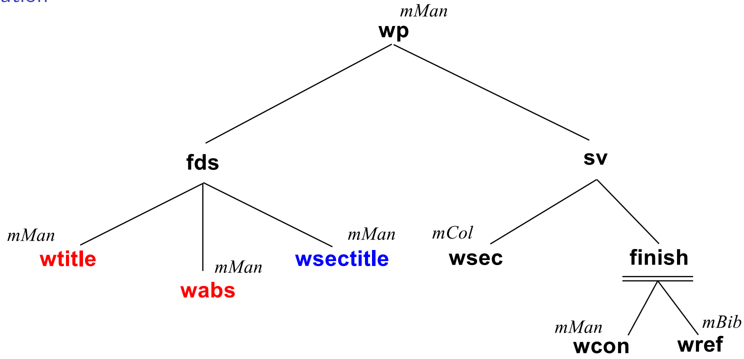


jaime event: +!wabs

action: jmoise.set_goal_state(s1,wabs,satisfied)

Writing paper sample VI

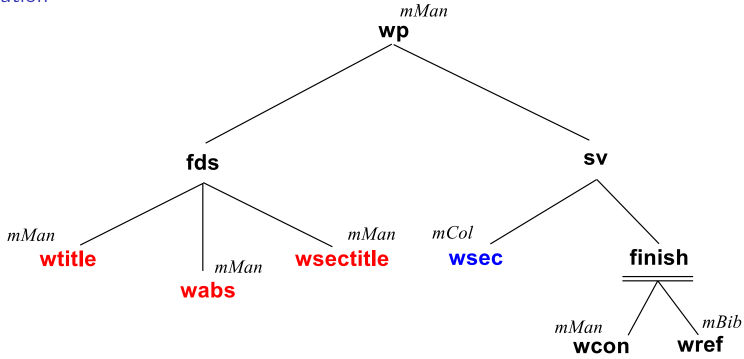
Execution



jaime event: +!wsectitles
action: jmoise.set_goal_state(s1,wsectitles,satisfied)

Writing paper sample VII

Execution

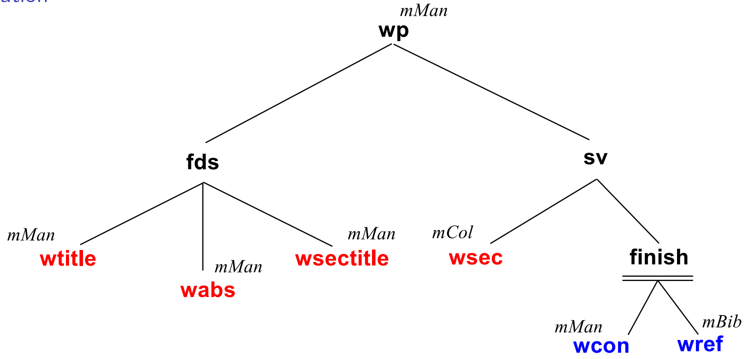


olivier, jomi event: +!wsecs

action: jmoise.set_goal_state(s1,wsecs,satisfied)

Writing paper sample VIII

Execution



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

play(gaucho1,herder,gr_herding_grp_13)[source(orgManager)].
play(gaucho4,herdboy,gr_herding_grp_13)[source(orgManager)].
play(gaucho5,herdboy,gr_herding_grp_13)[source(orgManager)].
pos(45,44,128)[source(percept)].
scheme(herd_sch,sch_herd_sch_18)[owner(gaucho3),source(orgManager)].
scheme(herd_sch,sch_herd_sch_12)[owner(gaucho1),source(orgManager)].
scheme_group(sch_herd_sch_12,gr_herding_grp_13)[source(orgManager)].
steps(700)[source(self)].
target(6,44)[source(gaucho1)].

- Rules

random_pos(X,Y) :-
(pos(AgX,AgY,_418) & (jia.random(RX,40) & ((RX > 5) & ((X = ((RX-20)+AgX)) & ((X >

- Intentions

Sel	Id	Pen	Intended Means Stack (hide details)
	16927	**suspended-self**	+!be_in_formation[scheme(sch_herd_sch_12),mission(hel **+!be_in_formation[scheme(Sch),mission(Mission)]**

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

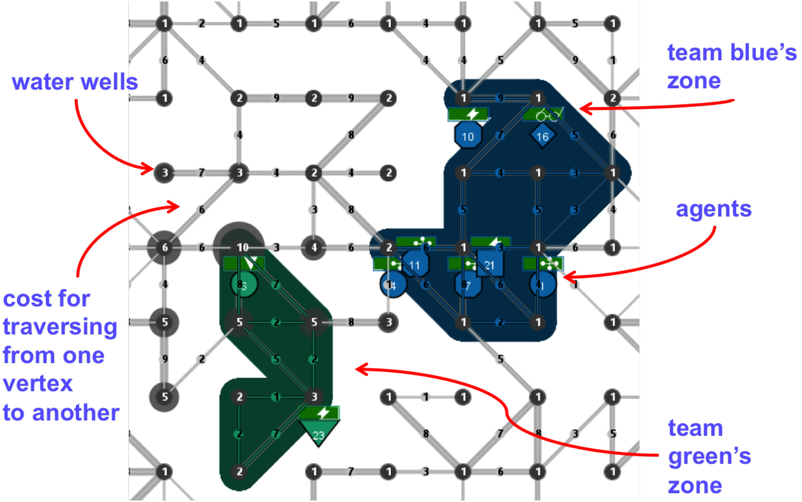
- Reasoning Cycle
- Tools
- Shortfalls
- Trends
- Conclusions

EOP: Environment Oriented Programming

OOP: Organisation Oriented Programming

- Origins and Fundamentals
- Some OOP approaches
- Moise Organisation Modeling Language (OML)

MAPC - Agent on Mars Scenario



MAPC - Agent on Mars Scenario

- 2 teams, 28 agents each.
- Roles and actions:

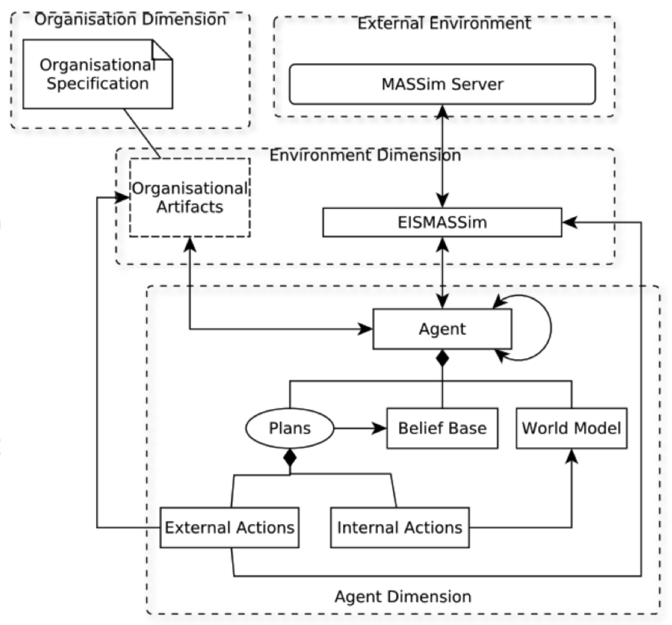
	explorer	repairer	saboteur	sentinel	inspector
recharge	X	X	X	X	X
attack			X		
parry		X	X	X	
goto	X	X	X	X	X
probe	X				
survey	X	X	X	X	X
inspect					X
buy	X	X	X	X	X
repair		X			
skip	X	X	X	X	X

LTI Team - A Jacamo Solution

Moise

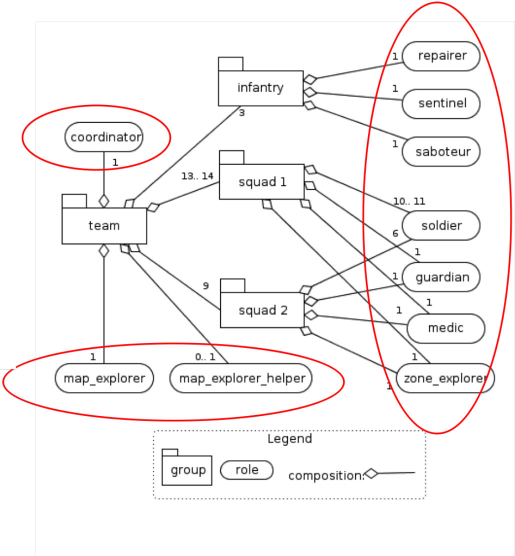
CArt

Jasc



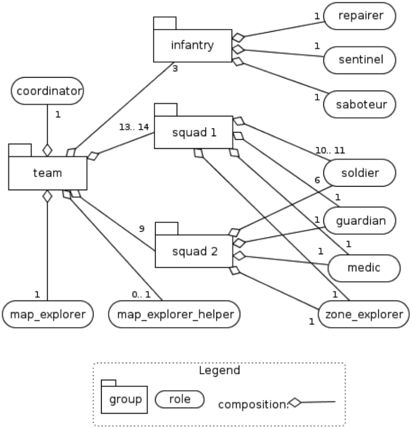
LTI Team - Structural Specification

```
<structural-specification>
    <role-definitions>
        <role id="map_explorer"/>
        <role id="map_explorer_helper"/>
        <role id="zone_explorer"/>
        <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>
```



LTI Team - 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>
</structural-specification>
```



LTI Team Code 1 - Coordinator Creates Groups

```
+!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)];
```

LTI Team - Functional Specification

```
<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>

        <mission id="m_coordinate" min="1" max="1">
            <goal id="coordinate_goal"/>
        </mission>

        <mission id="m_explore_map" min="1" max="1">
            <goal id="explore_map_goal"/>
        </mission>

        <mission id="m_help_explore_map" min="0" max="1">
            <goal id="help_explore_map_goal"/>
        </mission>
    </scheme>
```

LTI Team - 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>
```

LTI Team - Functional Specification

```
<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>

    <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"/>
```

LTI Team Code 2 - Coordinator Creates Schemes and Links to Teams

```
// 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")];
```

LTI Team - Normative Specification

```
<normative-specification>
    <norm id = "n1"  type="obligation" role="coordinator"         mission="m_coordinate"/>
    <norm id = "n2"  type="obligation" role="map_explorer"        mission="m_explore_map"/>
    <norm id = "n3"  type="permission" role="map_explorer_helper" mission="m_help_explore_map"/>
    <norm id = "n4"  type="permission" role="saboteur"            mission="m_attack"/>
    <norm id = "n5"  type="permission" role="sentinel"            mission="m_sabotage"/>
    <norm id = "n6"  type="permission" role="inspector"           mission="m_inspect"/>
    <norm id = "n7"  type="permission" role="inspector"           mission="m_create_zone"/>
    <norm id = "n8"  type="permission" role="repairer"            mission="m_repair"/>
    <norm id = "n9"  type="permission" role="soldier"             mission="m_create_zone"/>
    <norm id = "n10" type="permission" role="guardian"            mission="m_defend_zone"/>
    <norm id = "n11" type="obligation" role="zone_explorer"       mission="m_explore_zone"/>
    <norm id = "n12" type="obligation" role="medic"               mission="m_occupy_center"/>
</normative-specification>
```

LTI Team Code 3 - Adopting a Role

```
// 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)].
```

LTI Team Code 4 - Commiting to a Mission

```
// 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

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

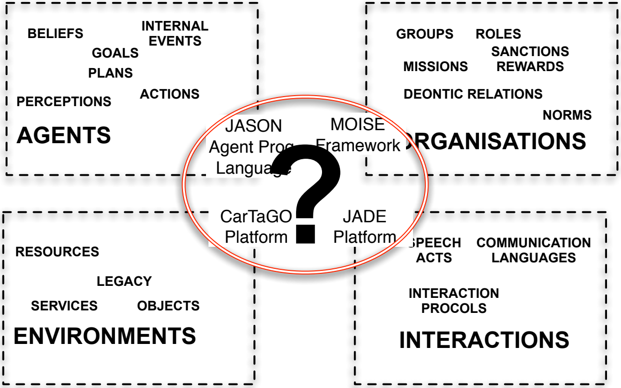
  `http://moise.souceforge.net`

Summary

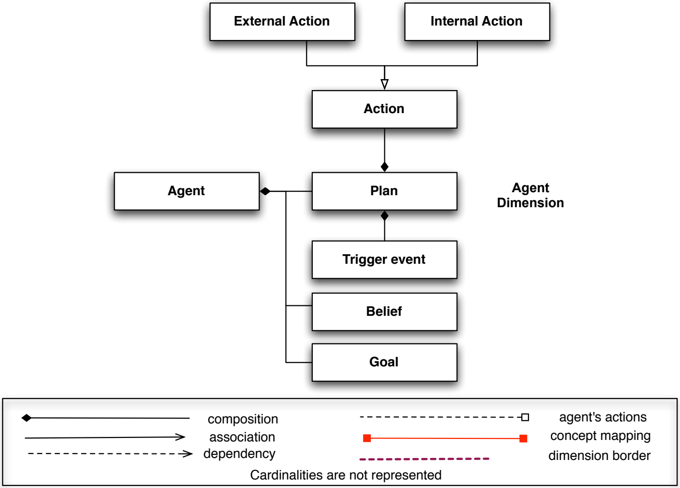
- *Jason*
  - declarative and goal oriented programming
  - goal patterns (maintenance goal)
  - meta-programming (.drop intention([group(g1)])
  - customisations (integration with the simulator and the organisation)
  - internal actions (code in Java)
  - $\rightsquigarrow$ good programming style
- $\mathcal{M}$oise 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
  - $\rightsquigarrow$ team strategy defined at a high level

Conclusions

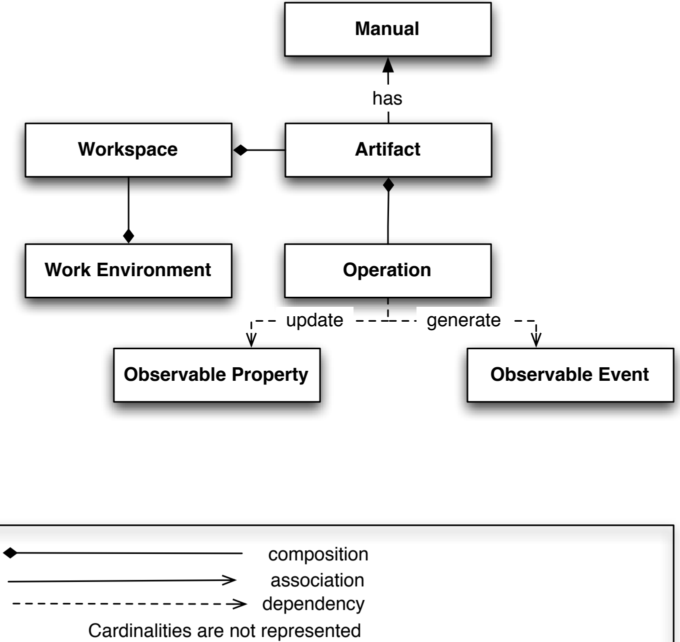
Putting the Pieces Together



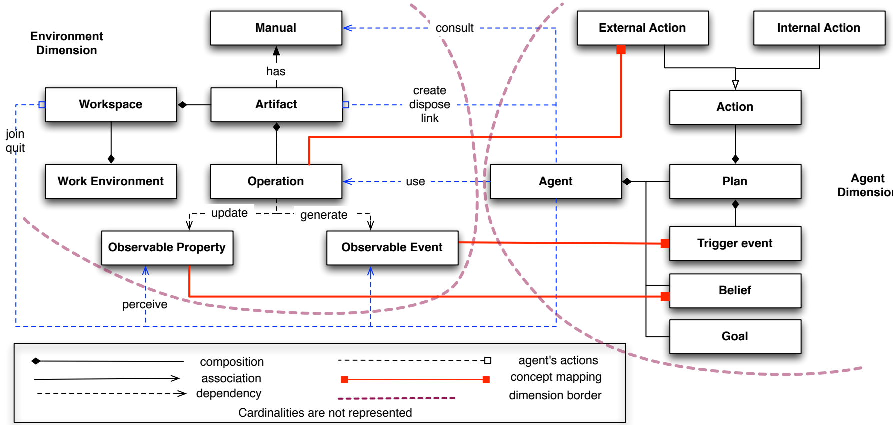
Agent meta-model



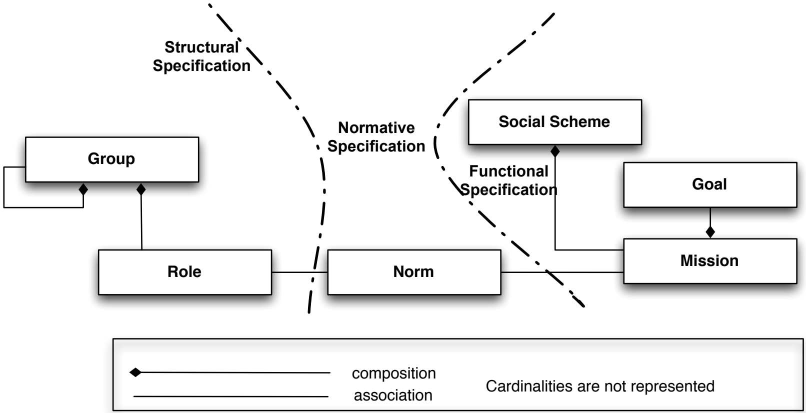
Environment meta-model



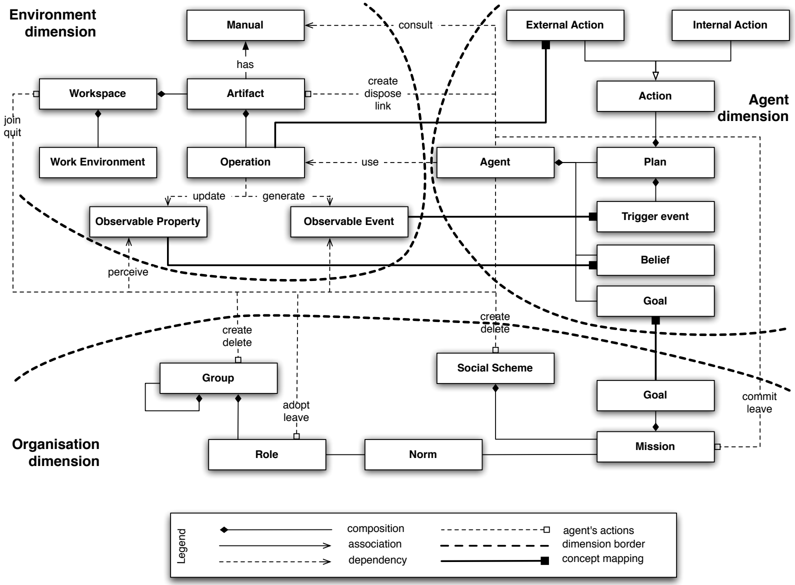
A & E Interaction meta-model



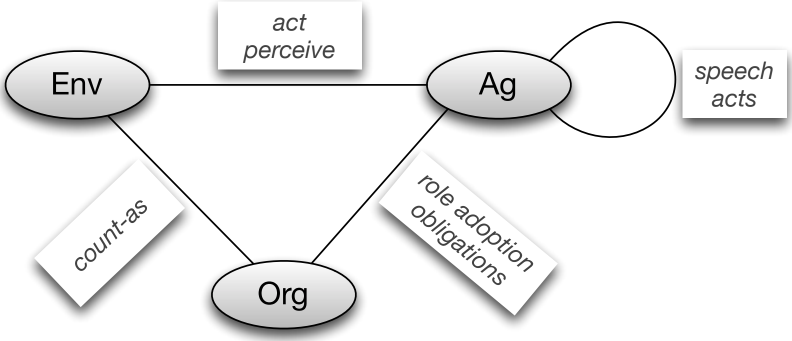
Organisation meta-model



JaCaMo Meta-Model



JaCaMo binding concepts



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

Behrens, T. M., Hindriks, K. V., Bordini, R. H., Braubach, L., Dastani, M., Dix, J., Hübner, J. F., and Pokahr, A. (2010).
An interface for agent-environment interaction.
In Collier, R. W., Dix, J., and Novák, P., editors, *Programming Multi-Agent Systems - 8th International Workshop, ProMAS 2010, Toronto, ON, Canada, May 11, 2010. Revised Selected Papers*, volume 6599 of *Lecture Notes in Computer Science*, pages 139–158. Springer.

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

Boella, G., Torre, L., and Verhagen, H. (2008).
Introduction to the special issue on normative multiagent systems.
Autonomous Agents and Multi-Agent Systems, 17(1):1–10.

Boissier, O. (2003).
Contrôle et coordination orientés multi-agents.
Habilitation à diriger des recherches, ENS Mines Saint-Etienne et Université Jean Monnet.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2011).
Multi-agent oriented programming with jacamo.
Science of Computer Programming, pages –.

Bordini, R., Hübner, J., and Wooldridge, M. (2007a).
Programming Multi-Agent Systems in AgentSpeak Using Jason.
Wiley-Interscience.

Bibliography II

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

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

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

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

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

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

Bibliography III

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

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

Carabelea, C. (2007).
Reasoning about autonomy in open multi-agent systems - an approach based on the social power theory.
in french, ENS Mines Saint-Etienne.

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 Intelligent and Soft-Computing. Springer.

Ciortea, A. (2011).
Modeling relationships for privacy preservation in virtual communities.
Master's thesis, University Politehnica of Bucharest.

Bibliography IV

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

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

de Brito, M., Hübner, J. F., and Bordini, R. H. (2012).
Programming institutional facts in multi-agent systems.
In *COIN-12, Proceedings*.

Demazeau, Y. (1995).
From interactions to collective behaviour in agent-based systems.
In *Proc. of the 1st European Conf. on Cognitive Science. Saint-Malo*, pages 117–132.

Demazeau, Y. (1997).
Steps towards multi-agent oriented programming.
(slides Workshop) 1st International Workshop on Multi-Agent Systems, IWMAS'97, Boston.

Esteva, M., Rodriguez-Aguiar, J. A., Sierra, C., Garcia, P., and Arcos, J. L. (2001).
On the formal specification of electronic institutions.
In Dignum, F. and Sierra, C., editors, *Proceedings of the Agent-mediated Electronic Commerce*, LNAI 1191, pages 126–147, Berlin. Springer.

Bibliography V

Esteva, M., Rodríguez-Aguilar, J. A., Rosell, B., and Arcos, J. L. (2004).
AMELI: An agent-based middleware for electronic institutions.
In Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2004)*, pages 236–243, New York. ACM.

Ferber, J. and Gutknecht, O. (1998).
A meta-model for the analysis and design of organizations in multi-agents systems.
In Demazeau, Y., editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Press.

Fisher, M. (2005).
Metatem: The story so far.
In Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors, *PROMAS*, volume 3862 of *Lecture Notes in Computer Science*, pages 3–22. Springer.

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

Gasser, L. (2001).
Organizations in multi-agent systems.
In *Pre-Proceeding of the 10th European Worshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'2001)*, Annecy.

Bibliography VI

Gâteau, B., Boissier, O., Khadraoui, D., and Dubois, E. (2005).
Moiseinst: An organizational model for specifying rights and duties of autonomous agents.
In *Third European Workshop on Multi-Agent Systems (EUMAS 2005)*, pages 484–485, Brussels Belgium.

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

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

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

Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000).
Moise: An organizational model for multi-agent systems.
In Monard, M. C. and Sichman, J. S., editors, *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI, 15th Brazilian Symposium on AI (IBERAMIA/SBIA'2000), Atibaia, SP, Brazil, November 2000*, LNAI 1952, pages 152–161, Berlin. Springer.

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

Bibliography VII

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

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

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

Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (2009).
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.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2002).
A model for the structural, functional, and deontic specification of organizations in multiagent systems.
In Bittencourt, G. and Ramalho, G. L., editors, *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02)*, volume 2507 of *LNAI*, pages 118–128, Berlin. Springer.

Bibliography VIII

Hübner, J. F., Sichman, J. S., and Boissier, O. (2006).
S-MOISE+: A middleware for developing organised multi-agent systems.
In Boissier, O., Dignum, V., Matson, E., and Sichman, J. S., editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 64–78. Springer.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007).
Developing Organised Multi-Agent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels.
Agent-Oriented Software Engineering, 1(3/4):370–395.

Hübner, J. F., Vercouter, L., and Boissier, O. (2009).
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.

Huhns, M. N. (2001).
Interaction-oriented programming.
In *First international workshop, AOSE 2000 on Agent-oriented software engineering*, pages 29–44, Secaucus, NJ, USA. Springer-Verlag New York, Inc.

Kitio, R. (2011).
Gestion de l'ouverture au sein d'organisations multi-agents. Une approche basée sur des artefacts organisationnels.
PhD thesis, ENS Mines Saint-Etienne.

Bibliography IX

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

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

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

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

Okuyama, F. Y., Bordini, R. H., and da Rocha Costa, A. C. (2008).
A distributed normative infrastructure for situated multi-agent organisations.
In Baldoni, M., Son, T. C., van Riemsdijk, M. B., and Winikoff, M., editors, *DALT*, volume 5397 of *Lecture Notes in Computer Science*, pages 29–46. Springer.

Bibliography X

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

Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17(3):432–456.

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

Ossowski, S. (1999).
Co-ordination in Artificial Agent Societies: Social Structures and Its Implications for Autonomous Problem-Solving Agents, volume 1535 of *LNAI*.
Springer.

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

Bibliography XI

Piunti, M., Ricci, A., Boissier, O., and Hubner, J. (2009a).
Embodying organisations in multi-agent work environments.
In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2009)*, Milan, Italy.

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

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

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

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

Bibliography XII

Pynadath, D. V. and Tambe, M. (2003).
An automated teamwork infrastructure for heterogeneous software agents and humans.
Autonomous Agents and Multi-Agent Systems, 7(1-2):71–100.

Pynadath, D. V., Tambe, M., Chauvat, N., and Cavedon, L. (1999).
Toward team-oriented programming.
In Jennings, N. R. and Lespérance, Y., editors, *ATAL*, volume 1757 of *LNCS*, pages 233–247. Springer.

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

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

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

Bibliography XIII

Ricci, A., Piunti, M., Acay, L. D., Bordini, R., Hubner, J., and Dastani, M. (2008).
Integrating Artifact-Based Environments with Heterogeneous Agent-Programming Platforms.
In *Proceedings of AAMAS-08*.

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

Ricci, A., Piunti, M., and Viroli, M. (2011).
Environment programming in multi-agent systems: an artifact-based perspective.
Autonomous Agents and Multi-Agent Systems, 23:158–192.

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

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

Bibliography XIV

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

Ricci, A., Viroli, M., and Omicini, A. (2007b).
The A&A programming model & technology for developing agent environments in MAS.
In Dastani, M., El Fallah Seghrouchni, A., Ricci, A., and Winikoff, M., editors, *Programming Multi-Agent Systems*, volume 4908 of *LNAI*, pages 91–109. Springer Berlin / Heidelberg.

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

Ricordel, P. and Demazeau, Y. (2002).
VOLCANO: a vowels-oriented multi-agent platform.
In Dunin-Keplicz and Nawarecki, editors, *Proceedings of the International Conference of Central Eastern Europe on Multi-Agent Systems (CEEMAS'01)*, volume 2296 of *LNAI*, pages 252–262. Springer Verlag.

Bibliography XV

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

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

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

Sorici, A. (2011).
Agile governance in an ambient intelligence environment.
Master's thesis, University Politehnica of Bucharest.

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

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

Bibliography XVI

Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K., and Zambonelli, F. (2007).
Infrastructures for the environment of multiagent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):49–60.

Weyns, D. and Holvoet, T. (2004).
A formal model for situated multi-agent systems.
Fundamenta Informaticae, 63(2-3):125–158.

Weyns, D. and Holvoet, T. (2007).
A reference architecture for situated multiagent systems.
In *Environments for Multiagent Systems III*, volume 4389 of *LNCS*, pages 1–40. Springer Berlin / Heidelberg.

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

Weyns, D. and Parunak, H. V. D., editors (2007).
Special Issue on Environments for Multi-Agent Systems, volume 14 (1) of *Autonomous Agents and Multi-Agent Systems*. Springer Netherlands.

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

Bibliography XVII

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

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

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