Outline of the course

- Introduction
- EOP
  - About Environment Oriented Programming
  - A&A and CArtAgO
- OOP
  - About Organisation Oriented Programming
  - Moise
- Conclusions
- Practical Exercise: a hands-on lab session!
Introduction
Abstractions in Multi-Agent Systems
Abstractions in Multi-Agent Systems

- **Individual Agent Level**: autonomy, situatedness
  - **Cognitive Concepts**: beliefs, desires, goals, intentions, plans
  - **Reasoning Cycle**: sense/reason/act, reactive/pro-active behaviour

- **Environment Level**: resources and services that agents can access and control; sensing and acting in an environment

- **Social and Organisation Level**: cooperation, coordination, regulation patterns
  - **Roles**: rights, responsibilities, ...
  - **Organisational Rules**: constraints on roles and their interactions, norms, deadlines, ...
  - **Organisational Structures**: topology of interaction patterns and relations over activity control
Agent Oriented Programming

- Proposed by Shoham [Shoham, 1993]
- Use of mentalistic notions and a societal view of computation (anthropomorphism)
- Levels of abstraction: Agents – Organisations – Environment
- Programming languages for agents have developed a lot since then, but still not a mature paradigm
- Programming languages/platforms for organisation and environment are also being developed
- Some agent development platforms have a formal basis
- Many influenced by the BDI agent architecture
BDI Architecture

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

Programming Languages for Multi-Agent Systems
E.g., Jason, Jadex, JACK, 2APL, GOAL, Brahms, JIAC, Agent Factory, MetateM, Golog variants, ...

- **Architecture** to represent an agent mental state:
  - **Beliefs**: information available to agent (e.g., about the environment or other agents)
  - **Goals**: states of affairs that the agent wants to achieve
  - **Events**: changes in agents beliefs or goals
  - **Capabilities**: reusable modules of activities that the agent can perform
  - **Plans**: reasoning about courses of action to achieve goals
  - **Rules**: reasoning about beliefs
Programming Languages for Cognitive Agents

Some steps of a **Reasoning Cycle**:

- Determining Relevant Plans for Handling Events
- Select a Plan for Execution
- Execute Part of an Intended Plans
- Handle Plan Failures

**Agent Interpreter** is an infinite loop of such reasoning cycles. The architecture and reasoning cycle together with the agent program (specially plans) determine the behaviour of the agent.
Concepts used to specify the state of an organisation:
- Agents, Roles, Groups
- Norms, Obligations, Prohibitions, Permissions, Violations
- Dependency, Power, Delegation, Information flow relations
- Deadlines, Sanctions, Rewards

Management Infrastructure to control and coordinate agent behaviour at run-time:
- Endogenous: The control is a part of the agent program
- Exogenous: The control is performed by an external system
  - Monitoring Agent Behaviour
  - Enforcing Organisational Rules
  - Regimenting Organisational Rules
Programming Languages/Platforms for Environments

- **Artifacts** to represent the state of the environment
  - Access to Databases/Services/etc., Coordination, Interaction
  - Environment “objects”, i.e., non-proactive entities

- **Processing Operations** on Artifacts
  - Realising the effects of environments actions
  - Providing events related to sensing the environment
  - Synchronising agent actions

- At the right level of abstraction for a multi-agent system
We will use **JaCaMo**

First fully operational, unified platform covering the 3 main levels of abstractions for multi-agent oriented programming

JaCaMo = Jason + CArtAgO + Moise

http://jacamo.sourceforge.net

More than the sum of 3 successful platforms

Revealing the full potential of **Multi-Agent Oriented Programming**
Environment Oriented Programming — EOP —
Outline

2 Environment Programming
- Why Environment Programming in MAS
- Basic Level
- Advanced Level
- A&A and CArtAgO
- Conclusions and Wrap-up
The notion of environment is intrinsically related to the notion of agent and multi-agent system

- “An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective” [Wooldridge, 2002]
- “An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors.” [Russell and Norvig, 2003]

Including both physical and software environments
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
In evidence

- overlapping spheres of visibility and influence
- ..which means: interaction
Why Environment Programming

- **Basic level**
  - to create testbeds for real/external environments
  - to ease the interface/interaction with existing software environments

- **Advanced level**
  - to uniformly **encapsulate** and **modularise** functionalities of the MAS out of the agents
    - typically related to interaction, coordination, organisation, security
  - **externalisation**
  - this implies changing the perspective on the environment
    - environment as a **first-class abstraction** of the MAS
    - **endogenous** environments (vs. exogenous ones)
    - **programmable** environments
Environment Programming: General Issues

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

MAS

AGENTS

actions

percepts

SIMULATED WORLD

OR

INTERFACE

OR

WRAPPER TO EXISTING TECHNOLOGY

REAL WORLD (PHYSICAL OR COMPUTATIONAL)

EXTERNAL WORLD (PHYSICAL OR COMPUTATIONAL)

Example: JAVA PLATFORM

mimicking
Basic Level: Features

- **Environment** conceptually conceived as a single monolithic block
  - providing actions, generating percepts

- **Environment API**
  - to define the set of actions and program actions computational behaviour
    - which include the generation of percepts
  - typically implemented using as single object/class in OO such as Java
    - method to execute actions
    - fields to store the environment state
  - available in many agent programming languages/frameworks
    - e.g., Jason, 2APL, GOAL, JADEX
An Example: *Jason* [Bordini et al., 2007]

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

![Diagram of User Environment and Agent Architecture](image-url)

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

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

User
Environment
Agent
Architecture
getPercepts
executeAction
change
percepts
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;
  }

  ...
// 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, 2008]

- **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 \texttt{2APL}

```java
package blockworld;

public class Env extends apapl.Environment {

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

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

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

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

    ...

}
```
### Belief Updates:

- `{ 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( )`
  - `goto( X, Y )`

### 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 Standard – EIS Initiative

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

- actually actions/percepts are issued to controllable entities in environment model
- represent the agent bodies, with effectors and sensors
Environment Interface Features

- **Interface functions**
  - attaching, detaching, and notifying observers (software design pattern);
  - registering and unregistering agents;
  - adding and removing entities;
  - managing the agents-entities-relation;
  - performing actions and retrieving percepts;
  - managing the environment

- **Interface Intermediate language**
  - to facilitate data-exchange
  - encoding percepts, actions, events
Vision: environment as a **first-class abstraction** in MAS [Weyns et al., 2007, Ricci et al., 2010b]

- **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
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
- **Regulate** the access to shared resources
- **Mediate** interaction between agents
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.
Environments for Multi-Agent Systems research field / **E4MAS** workshop series [Weyns et al., 2005]
- different themes and issues (see JAAMAS Special Issue [Weyns and Parunak, 2007] for a good survey)
  - mechanisms, architectures, infrastructures, applications [Platon et al., 2007, Weyns and Holvoet, 2007, Weyns and Holvoet, 2004, Viroli et al., 2007]
- the main perspective is (agent-oriented) software engineering

**Focus of this tutorial: the role of the environment abstraction in** **MAS programming**
- **environment programming**
Environment Programming

- Environment as **first-class programming abstraction** [Ricci et al., 2010b]
  - software designers and engineers perspective
  - *endogenous* environments (vs. exogenous one)
  - programming MAS = programming Agents + programming Environment
    - ..but this will be extended to include OOP in next part

- Environment as **first-class runtime abstraction** for agents
  - agent perspective
  - to be observed, used, adapted, constructed, ...

- Defining computational and programming frameworks/models also for the environment part
Computational Frameworks for Environment Programming: Issues

- Defining the environment interface
  - actions, percepts, data model
  - **contract** concept, 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., 2010b]**
  - Introducing a computational notion of artifact to design and implement agent environments
A&A and CArtAgO
Agents and Artifacts (A&A) Conceptual Model: Background Human Metaphor
A&A Basic Concepts [Omicini et al., 2008]

- **Agents**
  - autonomous, goal-oriented pro-active entities
  - create and co-use artifacts for supporting their activities
    - besides direct communication

- **Artifacts**
  - non-autonomous, function-oriented, stateful entities
    - controllable and observable
  - modelling the tools and resources used by agents
    - designed by MAS programmers

- **Workspaces**
  - grouping agents & artifacts
  - defining the topology of the computational environment
A&A Programming Model Features [Ricci et al., 2007b]

- **Abstraction**
  - artifacts as first-class resources and tools for agents
- **Modularisation**
  - artifacts as modules encapsulating functionalities, organized in workspaces
- **Extensibility and openness**
  - artifacts can be created and destroyed at runtime by agents
- **Reusability**
  - artifacts (types) as reusable entities, for setting up different kinds of environments
A&A Meta-Model in More Detail [Ricci et al., 2010b]
Artifact Abstract Representation

Signal: ObsPropName(Args)

Usage Interface:
- OperationX(Params)
- ...

Observable Properties:
- OperationY(Params)
- ...

Operations:
- Link Interface

Diagram showing Signal, Usage Interface, Observable Properties, and Operations connected through interfaces.
A World of Artifacts

- a counter
  - count: 5
  - inc
  - reset

- a flag
  - state: true
  - switch

- a Stock Quote Web Service
  - state available
  -.wsdl...
  - GetLastTradePrice...

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

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

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

- an event service
  - clearEvents
  - postEvent
  - registerForEvs

- a tuple space
  - out
  - in
  - rd
A Simple Taxonomy

- **Individual or personal artifacts**
  - designed to provide functionalities for a single agent use
    - e.g. an agenda for managing deadlines, a library...

- **Social artifacts**
  - designed to provide functionalities for structuring and managing the interaction in a MAS
  - coordination artifacts [Omicini et al., 2004], organisation artifacts, ...
    - e.g. a blackboard, a game-board,...

- **Boundary artifacts**
  - to represent external resources/services
    - e.g. a printer, a Web Service
  - to represent devices enabling I/O with users
    - e.g. GUI, console, etc.
Actions and Percepts in Artifact-Based Environments

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

**actions ←→ 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
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
Agents can dynamically select which artifacts to observe

- predefined focus/stopFocus actions
By focusing 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
Basic mechanism to enable inter-artifact interaction

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

- Agent-readable description of artifact’s...
  - ...functionality
    - what functions/services artifacts of that type provide
  - ...operating instructions
    - how to use artifacts of that type

- Towards advanced use of artifacts by intelligent agents [Piunti et al., 2008]
  - dynamically choosing which artifacts to use to accomplish their tasks and how to use them
  - strong link with Semantic Web research issues

- Work in progress
  - defining ontologies and languages for describing the manuals
CArtAgO

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

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

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

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

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

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

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

- **Basic operation features**
  - output parameters to represent action feedbacks
  - long-term operations, with a high-level support for synchronization (await primitive, guards)
Example 2: Producers and Consumers

PRODUCERS

item_to_produce(0).
!produce.

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

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

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

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

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

CONSUMERS

!consume.

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

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

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

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

- Process-based operation execution semantics
  - action/operation execution can be long-term
  - action/operation execution can overlap
  - key feature for implementing coordination functionalities
- Operation with output parameters as action feedbacks
Action Execution & Blocking Behaviour

- Given the action/operation map, by executing an action the intention/activity is suspended until the corresponding operation has completed or failed
  - action completion events generated by the environment and automatically processed by the agent/environment platform bridge
  - no need of explicit observation and reasoning by agents to know if an action succeeded

- However **the agent execution cycle is not blocked!**
  - the agent can continue to process percepts and possibly execute actions of other intentions
Example 3: Internal Processes – A Clock

**CLOCK**

```java
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**

```java
!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**
Exploiting artifacts to enable interaction between human users and agents
Example 4: Agent and User Interaction

GUI ARTIFACT

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

  public void setup() {
    frame = new MyFrame();
    // Link actions and key strokes to operations
    linkActionEventToOp(frame.okButton, "ok");
    linkKeyStrokeToOp(frame.text, "ENTER", "updateText");
    linkWindowClosingEventToOp(frame, "closed");
    defineObsProperty("value", getValue());
    frame.setVisible(true);
  }

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

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

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

USER ASSISTANT AGENT

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

+value(V)

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

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

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

- See CArtAgO papers and manuals for more information
A&A and CArtAgO: Some Research Explorations

- Designing and implementing artifact-based organisation Infrastructures
  - JaCaMo model and platform (which is the evolution of the ORA4MAS infrastructure [Hübner et al., 2009c])
- Cognitive stigmergy based on artifact environments [Ricci et al., 2007a]
  - cognitive artifacts for knowledge representation and coordination [Piunti and Ricci, 2009]
- Artifact-based environments for argumentation [Oliva et al., 2010]
- Including A&A in AOSE methodology [Molesini et al., 2005]
- Defining a Semantic (OWL-based) description of artifact environments (CArtAgO-DL)
  - JaSa project = JASDL + CArtAgO-DL
- ...

Applying CArtAgO and JaCa

- Using CArtAgO/JaCa for building real-world applications and infrastructures

- Some examples
  - JaCa-Android
    - implementing mobile computing applications on top of the Android platform using JaCa [Santi et al., 2011]
  - JaCa-WS / CArtAgO-WS
    - building SOA/Web Services applications using JaCa [Ricci et al., 2010a]
  - JaCa-Web
    - implementing Web 2.0 applications using JaCa
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 —
Abstractions in Multi-Agent Systems

[Diagram showing levels of abstraction in multi-agent systems, including organizational, agent, and environment levels.]
Outline

3 Organisation Oriented Programming (OOP)
- Fundamentals
- Motivations
- Some OOP approaches
- Focus on the Moise framework
  - Moise Organisation Modelling Language (OML)
  - Moise Organisation Management Infrastructure (OMI)
  - Moise integration with agents & environment
Organisation in MAS – a definition

- What is an organisation?
Organisation in MAS – a definition

- What is an organisation?

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

- Organisations are structured, patterned systems of activity, knowledge, culture, memory, history, and capabilities that are distinct from any single agent [Gasser, 2001]
  - Organisations are *supra-individual* phenomena

- A decision and communication schema which is applied to a set of actors that together fulfill a set of tasks in order to satisfy goals while guarantying a global coherent state [Malone, 1999]
  - definition by the designer, or by actors, to achieve a *purpose*

- An organisation is characterised by: a division of tasks, a distribution of roles, authority systems, communication systems, contribution-retribution systems [Bernoux, 1985]
  - *pattern of predefined cooperation*

- An arrangement of relationships between components, which results into an entity, a system, that has unknown skills at the level of the individuals [Morin, 1977]
  - *pattern of emergent cooperation*
Perspective on organisations
from EASSS’05 Tutorial (Sichman, Boissier)
Introduction

**Perspective on organisations**

from EASSS’05 Tutorial (Sichman, Boissier)

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

- **Organisation Centred**
  - Organisation is observed.
  - Coalition formation mechanisms programmed in Agents.

- **Agents don’t know about organisation**
  - AOSE
  - MASE, GAIA, MESSAGE, …
  - Organisation is a design model.
  - It is hard-coded in Agents.

- **Agents know about organisation**
  - TAEMS, STEAM, AGR
  - MOISE+, OPERA, …
  - Organisation-Oriented Programming of MAS

- **Designer / Observer**
  - Bottom-up... → Top-down

---

**Local Representation**

**Observed Organisation**

**Organisation Specification**

**Organisation Entity**
Organisation Oriented Programming (OOP)

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

First approach
- Agents read the program and follow it
Second approach

- Agents **are forced** to follow the program
Organisation Oriented Programming (OOP)

Second approach

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

Agent
Agent
Agent
Organisation Entity
Organisation Specification
Organisation Oriented Programming (OOP)

Components
- Programming language (OML)
- Platform (OMI)
- Integration to agent architectures and environment
Components of OOP:
Organisation Modelling Language (OML)

- Declarative specification of the organisation(s)
- Specific constraints, norms and cooperation patterns imposed on the agents
- Based on an organisational **model**
  
  e.g. AGR [Ferber and Gutknecht, 1998],
  TeamCore [Tambe, 1997],
  Islander [Esteva et al., 2001],
  Moise$^+$ [Hübner et al., 2002],
  Opera [Dignum and Aldewereld, 2010],
  2OPL [Dastani et al., 2009a],
  ...
Components of OOP: Organisation Management Infrastructure (OMI)

- **Coordination mechanisms**, i.e. support infrastructure
  
  e.g. MadKit [Gutknecht and Ferber, 2000b],
  Karma [Pynadath and Tambe, 2003],
  ...

- **Regulation mechanisms**, i.e. governance infrastructure
  
  e.g. Ameli [Esteva et al., 2004],
  S-Moise+ [Hübner et al., 2006],
  ORA4MAS [Hübner et al., 2009b],
  ...

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

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

  e.g. $J$-Moise$^+$ [Hübner et al., 2007], Autonomy based reasoning [Carabelea, 2007], ProsA$_2$ Agent-based reasoning on norms [Ossowski, 1999], ...

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

  e.g. [Piunti et al., 2009b], [Okuyama et al., 2008]
Motivations for OOP: Applications point of view

- Current applications show an increase in
  - Number of agents
  - Duration and repetitiveness of agent activities
  - Heterogeneity of the agents
  - Number of designers of agents
  - Agent ability to act and decide
  - Openness, scalability, dynamism

- More and more applications require the integration of human communities and technological communities (ubiquitous and pervasive computing), building connected communities (ICities) in which agents act on behalf of users
  - Trust, security, ..., flexibility, adaptation
Motivations for OOP: Constitutive point of view

- Organisation **helps** the agents to cooperate with other agents by defining **common** cooperation schemes
  - global tasks
  - protocols
  - groups, responsibilities

  **e.g.** ‘to bid’ for a product on eBay is an **institutional action** only possible because eBay defines rules for that very action
    - the bid protocol is a constraint but it also **creates** the action

  **e.g.** when a soccer team wants to play match, the organisation helps the members of the team to synchronise actions, to share information, etc
Motivations for OOP: Normative point of view

- MAS have two properties which seem contradictory:
  - a **global** purpose
  - **autonomous** agents
  - While the autonomy of the agents is essential, it may cause loss in the global coherence of the system and achievement of the global purpose
- Embedding **norms** within the **organisation** of an MAS is a way to constrain the agents’ behaviour towards the global purposes of the organisation, while explicitly addressing the autonomy of the agents within the organisation
  - Normative organisation
  - e.g. when an agent adopts a role, it adopts a set of behavioural constraints that support the global purpose of the organisation.
    It may decide to obey or disobey these constraints
Motivations for OOP: 

**Agents** point of view

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

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

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

- to decide to let agents enter into/leave from the organisation during execution
  - → 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
Some OOP approaches

- AGR/Madkit [Ferber and Gutknecht, 1998]
- STEAM/Teamcore [Tambe, 1997]
- ISLANDER/AMELI [Esteva et al., 2004]
- Opera/Operetta [Dignum and Aldewereld, 2010]
- PopOrg [Rocha Costa and Dimuro, 2009]
- 2OPL [Dastani et al., 2009a]
- ...
**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 cannot communicate with each other if they do not 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

- Agent
  - is member of Group
  - contains Group structure
  - plays Role
  - described by Group structure

- Group structure
  - contains Role properties
  - contains Role dependency

- Role
  - 1..* participant
  - 1..* role dependency
  - 1 source
  - 1 target
  - 1 initiator

- Interaction protocol

Role dependency is described by Group structure, which contains Group and contains Participant. A Participant is a member of a Group. A Participant can play a Role. A Role has properties and dependencies.
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
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 synchronisation: Establish joint intentions, Monitor team progress and repair, Individual may fail or succeed in own role
  - Reorganise if there is a critical role failure
  - Reassign critical roles based on joint intentions
  - Decision theoretic communication
- Supported by the TEAMCORE OMI
**Organization:** hierarchy of roles that may be filled by agents or groups of agents.

**Team Plan:**
- initial conditions,
- term. cond.: achievability, irrelevance, unachievability
- team-level actions.
STEAM OML Modelling Dimensions

B: agents’ possible behaviors
P: agents’ behaviors that lead to global purpose
E: agents’ possible behaviors constrained by the environment
O_S: agents’ possible behaviors structurally constrained by the organization
O_F: agents’ possible behaviors functionally constrained by the organization
Based on different influences: economics, norms, dialogues, coordination

\[\Rightarrow\] electronic institutions

Combining different alternative views: dialogical, normative, coordination

Institution Description Language:
- Performative structure (Network of protocols)
- Scene (multi-agent protocol)
- Roles
- Norms

Ameli as OMI
(define-institution

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

)
**ISLANDER OML Modelling Dimensions**

- **B**: agents’ possible behaviors
- **P**: agents’ behaviors that lead to global purpose
- **E**: agents’ possible behaviors constrained by the environment
- **O_S**: agents’ possible/permitted/obliged behaviors structurally constrained by the organisation
- **O_I**: agents’ possible/permitted/obliged behaviors interactionally constrained by the organisation
The aim is to design and develop a programming language to support the implementation of coordination mechanisms in terms of **normative** concepts.

An organisation

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

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

Effects:
{ -at_platform } enter { at_platform },
{ -ticket } buy_ticket { ticket },
{ at_platform, -in_train } embark
{ -at_platform, in_train }

Counts_as rules:
{ at_platform, -ticket } \Rightarrow \{ \text{viol_ticket} \},
{ in_train, -ticket } \Rightarrow \{ \text{viol_|_} \}

Sanction_rules:
{ viol_ticket } \Rightarrow \{ \text{fined}_10 \}
2OPL Modelling Dimension

Example (Train Station)

Facts:
\{ -at_platform , -is_train , -ticket \}

Effects:
\{ -at_platform \} enter \{ an_platform \},
\{ -ticket \} buy_ticket \{ ticket \},
\{ at_platform , -is_train \}
    embark
    \{ -at_platform, is_train \}

Counts as rules:
\{ at_platform , -ticket \} => \{ viol_ticket \},
\{ is_train , -ticket \} => \{ viol_|_ \}

Reaction rules:
\{ viol_ticket \} => \{ fined,10 \}

Normative Specification

Environment
Summary

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

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

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

- **Integrations**
  - Agents and Environment (c4Jason, c4Jadex [Ricci et al., 2009c])
  - Environment and Organisation ([Piunti et al., 2009a])
  - Agents and Organisation (J-Moise+ [Hübner et al., 2007])
Moise+ Modelling Dimensions

**Introduction**

Motivations

**OOP**

**Conclusion**

---

**Moise**

**Modelling Dimensions**

- **Environment**
  - Global goals, plans, Missions, schemas, preferences
  - Structural Specification
    - Groups, links, roles
    - **Compatibilities, multiplicities**
    - **inheritance**
  - Functional Specification
    - **Normative Specification**
      - Permissions, Obligations
      - Allows agents autonomy!
    - **Groups, links, roles**
    - **Compatibilities, multiplicities**
    - **inheritance**

---

**Environment**

**P**

**Environment**

**P**

**B**

**Functional Specification**

**Global goals, plans, Missions, schemas, preferences**

---

**Moise**

**Modelling Dimensions**
OML for defining organisation specification and organisation entity

Three independent dimensions [Hübner et al., 2007] (well adapted for the reorganisation concerns):

- **Structural**: Roles, Groups
- **Functional**: Goals, Missions, Schemes
- **Normative**: Norms (obligations, permissions, interdictions)

Abstract description of the organisation for

- the designers
- the agents

\[ \mathcal{J} \]–Moise\(^+\) [Hübner et al., 2007]

- the Organisation Management Infrastructure

\[ \sim \] ORA4MAS [Hübner et al., 2009b]
Moise OML Structural Specification

- Specifies the structure of an MAS along three levels:
  - **Individual** with **Role**
  - **Social** with **Link**
  - **Collective** with **Group**

- Components:
  - **Role**: label used to assign rights and constraints on the behavior of agents playing it
  - **Link**: relation between roles that directly constrains the agents in their interaction with the other agents playing the corresponding roles
  - **Group**: set of links, roles, compatibility relations used to define a shared context for agents playing roles in it
Structural Specification Example

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

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

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

Motivations

OOP

Conclusion

Functional Specification Example

- Score a goal
  - Get the ball
    - Go towards the opponent field
      - Be placed in the middle field
    - Be placed in the opponent goal area
      - Kick the ball to (agent committed to m2)
  - Shot at the opponent’s goal
    - Kick the ball to the goal area
      - Go to the opponent back line
    - Kick the ball to the goal area
      - (agent committed to m2)

Key

Organizational Entity

- Lucio
- Cafu
- Rivaldo

Graphical representation of social scheme “side_attack” for joj team
Goal States

- **Waiting**: initial state
- **Enabled**: goal pre-conditions are satisfied & scheme is well-formed
- **Satisfied**: agents committed to the goal have achieved it
- **Impossible**: the goal is impossible to be satisfied
Moise OML Normative Specification

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

<table>
<thead>
<tr>
<th>role</th>
<th>deontic</th>
<th>mission</th>
<th>TTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>back</td>
<td>obliged</td>
<td>m1 get the ball, go ...</td>
<td>1 minute</td>
</tr>
<tr>
<td>left</td>
<td>obliged</td>
<td>m2 be placed at ..., kick ...</td>
<td>3 minute</td>
</tr>
<tr>
<td>right</td>
<td>obliged</td>
<td>m2</td>
<td>1 day</td>
</tr>
<tr>
<td>attacker</td>
<td>obliged</td>
<td>m3 kick to the goal, ...</td>
<td>30 seconds</td>
</tr>
</tbody>
</table>
Organisational Entity

- **Structural**
  - groups
  - links
  - roles

- **Normative**
  - norms

- **Functional**
  - schemas
  - missions

**Purpose**

**Organisation specification**

- **Group instances**
- **Role player**
- **Agents**

- **Schema instances**
- **Mission player**
Organisation Entity Dynamics

1. Organisation is created (by the agents)
   - instances of groups
   - instances of schemes

2. Agents enter into groups adopting roles

3. Groups become responsible for schemes
   - Agents from the group are then obliged to commit to missions in the scheme

4. Agents commit to missions

5. Agents fulfil mission’s goals

6. Agents leave schemes and groups

7. Schemes and groups instances are destroyed
**Organisation management infrastructure (OMI)**

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

(e.g. MadKit, AMELI, \(S\)-Moise\(^+\), ...)
Organisational artifacts in ORA4MAS

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

- **specification**: the specification of the group in the OS (an object of class moise.os.ss.Group)
- **players**: a list of agents playing roles in the group. Each element of the list is a pair (agent x role)
- **schemes**: a list of scheme identifiers that the group is responsible for
ORA4MAS – GroupBoard artifact

### Operations:
- **adoptRole(role):** the agent executing this operation tries to adopt a role in the group
- **leaveRole(role):**
- **addScheme(schid):** the group starts to be responsible for the scheme managed by the SchemeBoard schId
- **removeScheme(schid):**
**ORA4MAS – SchemeBoard artifact**

**Observable Properties:**

- **Specification**: the specification of the scheme in the OS
- **Groups**: a list of groups responsible for the scheme
- **Players**: a list of agents committed to the scheme. Each element of the list is a pair (agent, mission)
- **Goals**: a list with the current state of the goals
- **Obligations**: list of obligations currently active in the scheme
ORA4MAS – SchemeBoard artifact

Operations:

- **commitMission(mission)** and **leaveMission**: operations to “enter” and “leave” the scheme
- **goalAchieved(goal)**: defines that some goal is achieved by the agent performing the operation
- **setGoalArgument(goal, argument, value)**: defines the value of some goal’s argument
Signals \( o = \text{obligation(to whom, reason, what, deadline)} \):
- \( \text{obl} \_\text{created}(o) \): the obligation \( o \) is created
- \( \text{obl} \_\text{fulfilled}(o) \): the obligation \( o \) is fulfilled
- \( \text{obl} \_\text{unfulfilled}(o) \): the obligation \( o \) is unfulfilled
- \( \text{obl} \_\text{inactive}(o) \): the obligation \( o \) is inactive
- \( \text{norm} \_\text{failure}(f) \): the failure \( f \) has happened
Organisational Artifacts enable organisation and environment integration

Embodied organisation [Piunti et al., 2009a]

status: ongoing work
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

\[ \rightarrow \text{indirect automatic updating of the organisation} \]

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, ...)

Agents can interact with organisational artifacts as with ordinary artifacts by perception and action.

Any Agent Programming Language integrated with CArtAgO can use organisational artifacts.

Agent integration provides some “internal” tools for the agents to simplify their interaction with the organisation:

- maintenance of a local copy of the organisational state
- production of organisational events
- provision of organisational actions
J-Moise: Jason + Moise

- Agents are programmed with Jason
- BDI agents (reactive planning) – suitable abstraction level
- The programmer has the possibility to express sophisticated recipes for adopting roles, committing to missions, fulfilling/violating norms, ...
- Organisational information is made accessible in the mental state of the agent as beliefs
- Integration is totally independent of the distribution/communication layer
**J-Moise: Jason + Moise**

**General view**

Jason-CArtAgo Agent

- Belief Base
- Plan Library
- Intentions

J-Moise+
Organisation Integration mechanism

Organisational Workspace (CArtAgO)
Example (GroupBoard)

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

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

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

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

Inspection of agent bob (cycle #0)

- Beliefs

  commitment(bob,mManager,"sch2")
  [artifact_id(cobj_4),commitment(bob,mManager,"sch1")
  [artifact_id(cobj_3),
  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
galState("sch2",wp,[bob],[bob],satisfied)
  [artifact_id(cot


Handling organisational events in \textit{Jason}

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

\textbf{Example (new agent entered the group)}
\begin{verbatim}
+play(Ag,boss,GId) <- .send(Ag,tell,hello).
\end{verbatim}

\textbf{Example (change in goal state)}
\begin{verbatim}
+goalState(Scheme,wsecs,_,_,satisfied) :
    .my_name(Me) & commitment(Me,mCol,Scheme)
<- leaveMission(mColaborator,Scheme).
\end{verbatim}

\textbf{Example (signals)}
\begin{verbatim}
+normFailure(N) <- .print("norm failure event: ", N).
\end{verbatim}
Typical plans for obligations

**Example**

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

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

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

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

- All available as open source at
  
  http://moise.souceforge.net
Conclusions
Exploiting Orthogonality

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

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

- Mapping
  - agent actions into environment operations (e.g. CArtAgO)
  - environment observable state/events into agent beliefs

- Outcome
  - agents with dynamic action repertoire
  - uniformly implementing any mechanisms (e.g. coordination mechanism) in terms of actions/percepts
    - no need to extend agents with special purpose primitives
  - exploiting a new type of agent modularity, based on externalization [Ricci et al., 2009a]
Exploiting Synergy on A/O Integration

- Normative deliberative agents
  - possibility to define mechanisms for agents to evolve within an organisation/several organisations
  - possibility to define proper mechanisms for deliberating on the internalisation/adopter/violation of norms
- Reorganisation, adaptation of the organisation
  - possibility to define proper mechanisms for diagnosing/evaluating/refining/defining organisations
- “Deliberative” Organisations
  - possibility to define dedicated organisational strategies for the regulation/adaptation of the organisation behaviour (organisational agents)
Exploiting Synergy between the E/O Dimensions

- Grounding the organisation infrastructure
  - implemented using environment abstractions
  - ... that agents perceive then as first-class entities of their world

- Mapping
  - organisational state reified by the environment computational state
  - organisational actions/perceptions reified by actions/percepts on the environment state
  - organisational functionalities encapsulated by suitably designed environment abstractions

- Outcome
  - “the power is back to agents” [Hübner et al., 2009c]
  - by perceiving and acting upon that environment, agents can reason and dynamically adapt the organisation infrastructure
Exploiting the environment role of **enabler** and **mediator** of agent interaction
- by providing actions and generating percepts

→ natural place where to embed and enforce **organisational rules and norms**
- affecting action execution behaviour and percepts generation

**Examples**
- simple: a game-board artifact in an artifact-based environment
  - providing agents actions to make moves
  - encapsulating and enforcing the rules of the game
- complex: fully-fledged institutions
  - reified into properly programmed environments
Exploiting the environment to create, represent, and manage dependencies and rules that are meaningful at the organisational level

A main example: implementing constitutive rules [Searle, 1997]

- events occurring in concrete environments conveyed as social and institutional events
- typically represented in the form X counts as Y in C
- an example: reaching the environment state S counts as achieving the organisational goal G

The integration E/O allows for naturally design and implementation of these kinds of rules

- without adding any further concepts wrt the ones belonging to the E/O dimensions
A proposal: JaCaMo
Ongoing and Related Research

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

- Normative programming and infrastructures [Hübner et al., 2009a, Tinnemeier et al., 2009, Dastani et al., 2009b]
An interface for agent-environment interaction.
In *In Proceedings of International Workshop on Programming Multi-Agent Systems (ProMAS-8)*.

*La sociologie des organisations*.
Seuil, 3ème edition.

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

Situating Cognitive Agents in GOLEM.
Formalising situatedness and adaptation in electronic institutions.
In Coordination, Organizations, Institutions and Norms in Agent Systems IV,

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

2APL: a practical agent programming language.

Normative multi-agent programs and their logics.
In Meyer, J.-J. and Broersen, J., editors, Knowledge Representation for Agents
and Multi-Agent Systems, volume 5605 of Lecture Notes in Computer Science,
Bibliography III

A programming language for normative multi-agent systems.
In *Multi-Agent Systems: Semantics and Dynamics of Organizational Models.*
IGI-Global.

Operetta: Organization-oriented development environment.

On the formal specification of electronic institutions.

AMELI: An agent-based middleware for electronic institutions.
Bibliography IV

A meta-model for the analysis and design of organizations in multi-agents systems.

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

Moiseinst: An organizational model for specifying rights and duties of autonomous agents.

The MADKIT agent platform architecture.
The MadKit agent platform architecture.

Moise: An organizational model for multi-agent systems.

Normative programming for organisation management infrastructures.
In MALLOW Workshop on Coordination, Organization, Institutions and Norms in Agent Systems in Online Communities (COIN-MALLOW 2009).

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


An automated teamwork infrastructure for heterogeneous software agents and humans.

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

Cognitive stigmergy: Towards a framework based on agents and artifacts.
Integrating artifact-based environments with heterogeneous agent-programming platforms.
In Proceedings of 7th International Conference on Agents and Multi Agents Systems (AAMAS08).


Environment programming in multi-agent systems – an artifact-based perspective.
Autonomous Agents and Multi-Agent Systems.
Published Online with ISSN 1573-7454 (will appear with ISSN 1387-2532).


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,  
Springer Berlin / Heidelberg.

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.

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.


