Multi-Agent Oriented Programming

O. Boissier

Univ. Lyon, IMT Mines Saint-Etienne, LaHC UMR CNRS 5516, France

in tight collaboration with R.H. Bordini[2], J.F. Hübner[3], A. Ricci[4]

[2] Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil
[3] Federal University of Santa Catarina (UFSC), Florianópolis, Brazil
[4] University of Bologna (UNIBO), Bologna, Italy

Winter 2019

Multi-Agent Oriented Programming

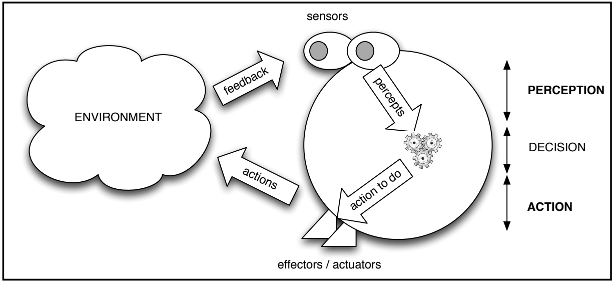
Programming Agents' Environment

Outline

Back to the Notion of Environment in MAS

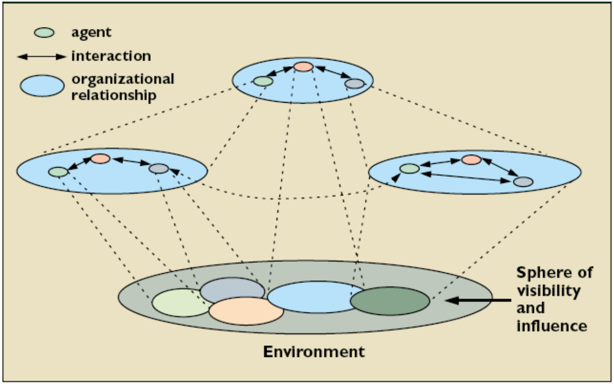
- The notion of environment is intrinsically related to the notion of agent and multi-agent system
  - "An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective" [Wooldridge, 2002]
  - "An agent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

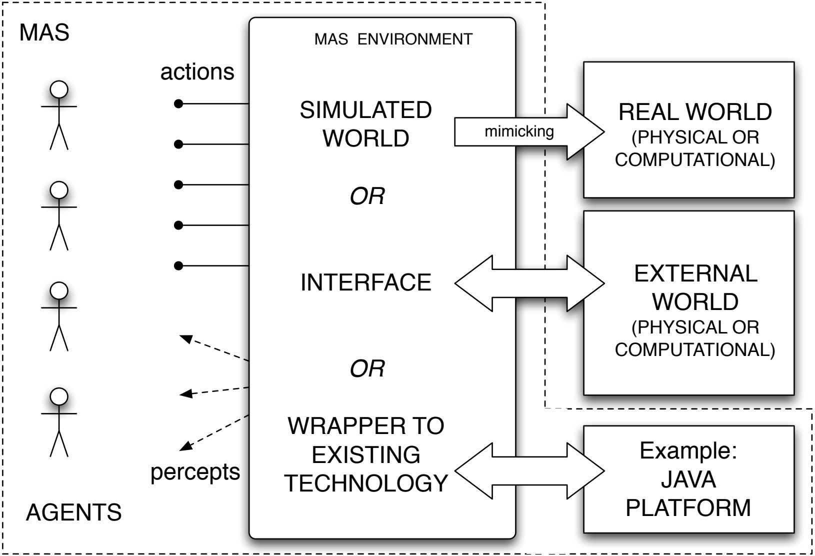
- Basic level
  - to create testbeds for real/external environments
  - to ease the interface/interaction with existing software environments
- Advanced level
  - to uniformly encapsulate and modularise functionalities of the MAS out of the agents
    - typically related to interaction, coordination,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

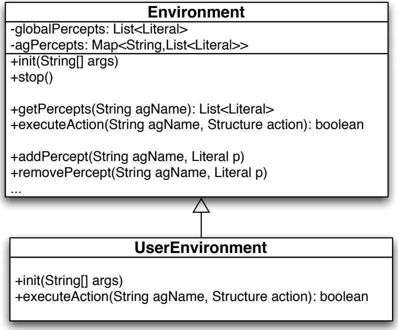


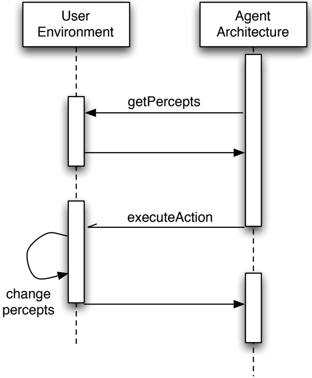
Basic Level: Features

- 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., 2007] (without JaCaMo)

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





Example (continued): 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 { ... }
}
```

Example (continued): *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, 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 **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

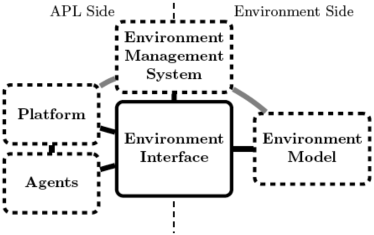
```
BeliefUpdates:
  { bomb(X,Y) }          RemoveBomb(X,Y){ not bomb(X,Y) }
  { true }               AddBomb(X,Y)   { bomb(X,Y) }
  { carry(bomb) }        Drop( )        { not carry(bomb)}
  { not carry(bomb) }    PickUp( )      { carry(bomb) }

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( )
  }
...
```

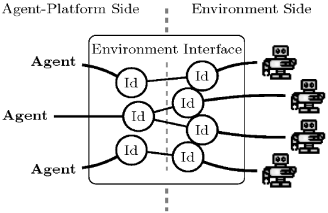
```
...

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 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

EIS Meta-Model





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

Environment Interface Features

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

Advanced Level Overview

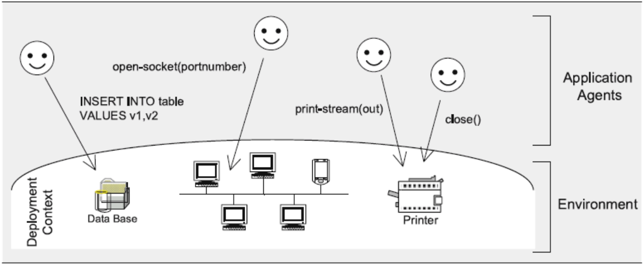
- Vision: environment as a first-class abstraction in MAS [Weyns et al., 2007, Ricci et al., 2010a]
  - application or endogenous environments, i.e. that environment which is an explicit part of the MAS
  - providing an exploitable design & programming abstraction to build MAS applications
- Outcome
  - 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

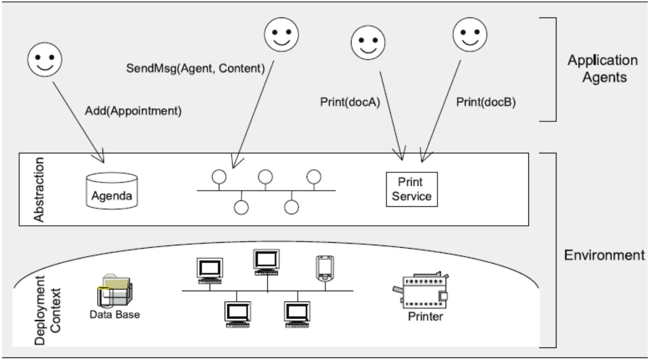
Basic Interface Support

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



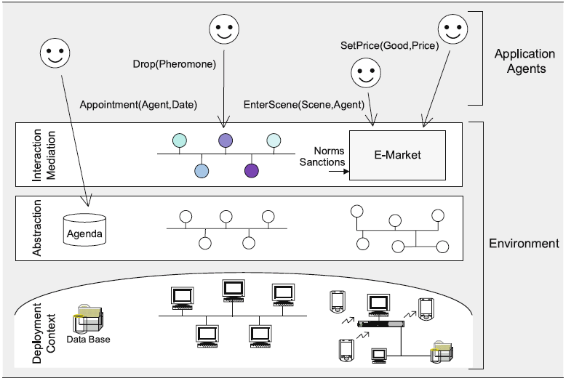
Abstraction Support

- 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., 2010a]
  - 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

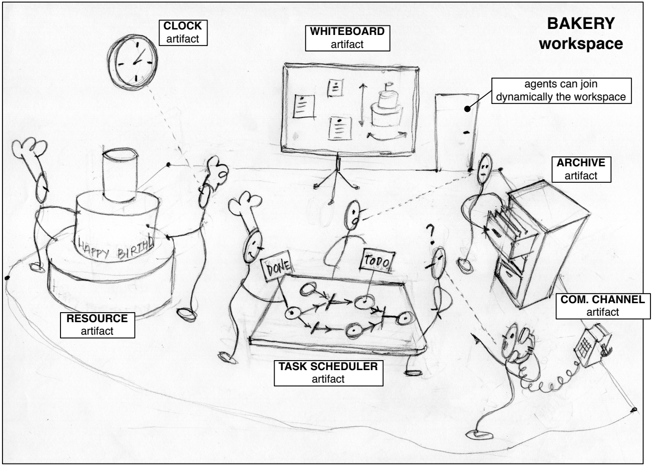
Existing Computational Frameworks

- 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, 2000]
- GOLEM [Bromuri and Stathis, 2008]
  - Logic-based framework to represent environments for situated cognitive agents
  - composite structure containing the interaction between cognitive agents and objects
- A&A and CArtAgO [Ricci et al., 2010a]
  - introducing a computational notion of artifact to design and implement agent environments

A&A and CArtAgO

Outline

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



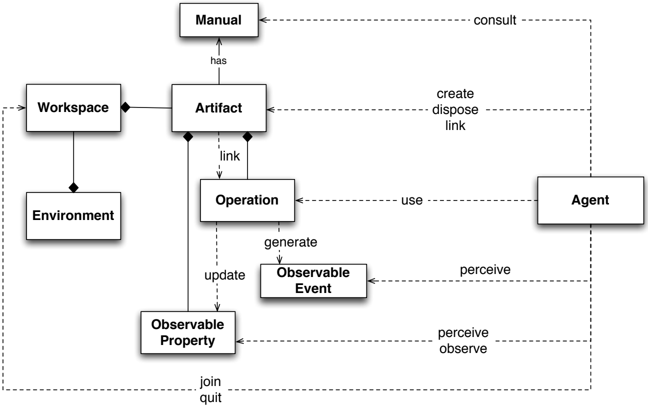
A&A Basic Concepts [Omicini et al., 2008]

- 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

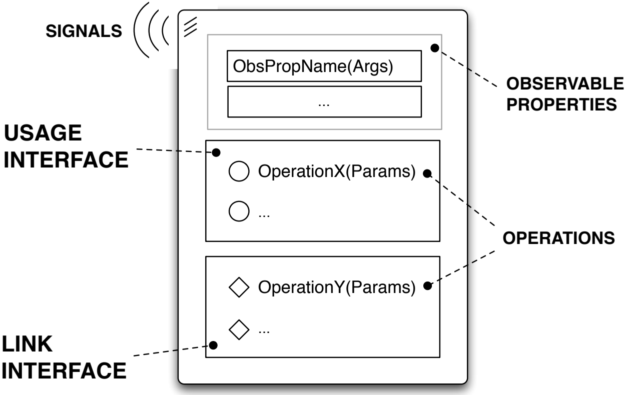
A&A Programming Model Features [Ricci et al., 2007b]

- 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

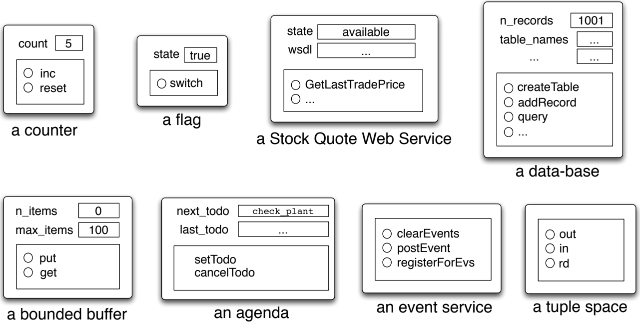
A&A Meta-Model in More Detail [Ricci et al., 2010a]



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 $\longleftrightarrow$ artifacts' operation

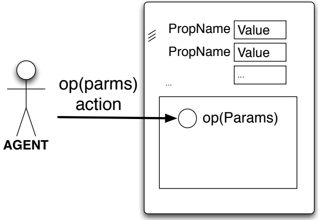
the action repertoire is given by the dynamic set of operations provided by the overall set of artifacts available in the workspace can be changed by creating/disposing artifacts

- action success/failure semantics is defined by operation semantics

percepts $\longleftrightarrow$ artifacts' observable properties + signals

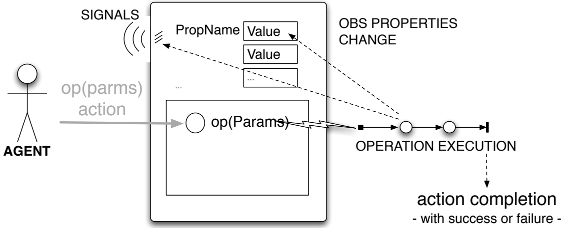
properties represent percepts about the state of the environment signals represent percepts concerning events signalled by the environment

Interaction Model: Use



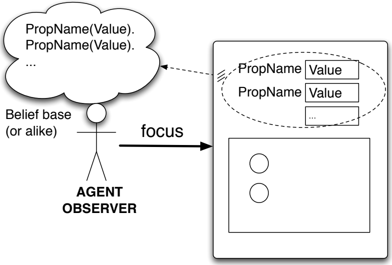
- Performing an action corresponds to triggering the execution of an operation
  - acting on artifact's usage interface

Interaction Model: Operation execution



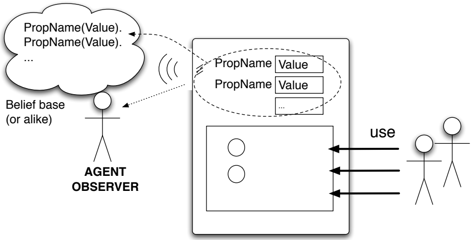
- a process structured in one or multiple transactional steps
- asynchronous with respect to agent
  - ...which can proceed possibly reacting to percepts and executing actions of other plans/activities
- operation completion causes action completion
  - action completion events with success or failure,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

Interaction Model: Observation



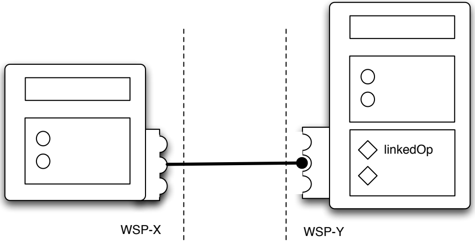
- Agents can dynamically select which artifacts to observe
  - predefined `focus`/`stopFocus` actions

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., 2009]
- 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`

Outline

Integration with Agent Languages and Platforms

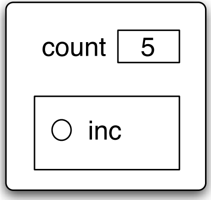
- Integration with existing agent platforms [Ricci et al., 2008]
  - by means of bridges creating an action/perception interface and doing data binding
- Outcome
  - developing open and heterogenous MAS
  - introducing a further perspective on interoperability besides the ACL's one
    - sharing and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

A&A in JaCaMo Platform

- Integration of CArtAgO with *Jason* language/platform
- 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
    - `init`, operation which is executed when artifact is created
  - 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).
```

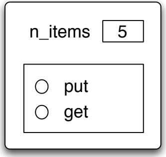
- Working with the shared counter

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

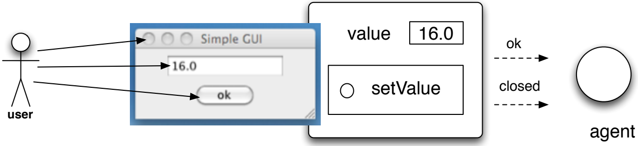
```
public class Clock extends Artifact {

  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

- 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
- ...

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

Multi-Agent Oriented Programming

O. Boissier

Univ. Lyon, IMT Mines Saint-Etienne, LaHC UMR CNRS 5516, France

in tight collaboration with R.H. Bordini[2], J.F. Hübner[3], A. Ricci[4]

[2] Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil
[3] Federal University of Santa Catarina (UFSC), Florianópolis, Brazil
[4] University of Bologna (UNIBO), Bologna, Italy

Winter 2019

Bibliography I

Behrens, T., Bordini, R., Braubach, L., Dastani, M., Dix, J., Hindriks, K., Hübner, J., and Pokahr, A. (2010).
An interface for agent-environment interaction.
In *In Proceedings of International Workshop on Programming Multi-Agent Systems (ProMAS-8)*.

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

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.

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

Bibliography II

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

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.

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.

Bibliography III

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.

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

Piunti, M., Ricci, A., 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.

Bibliography IV

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.

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. (2010a).
Environment programming in multi-agent systems – an artifact-based perspective.
Autonomous Agents and Multi-Agent Systems.
Published Online with ISSN 1573-7454 (will appear with ISSN 1387-2532).

Bibliography V

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

Ricci, A., 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.

Bibliography VI

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.

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

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

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.

Bibliography VII

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.

Bibliography VIII

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.

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