

Action and Perception in Multi-Agent Programming Languages: From Exogenous to Endogenous Environments

Alessandro Ricci, Andrea Santi and Michele Piunti
a.ricci@unibo.it, a.santi@unibo.it,
michele.piunti@unibo.it

DEIS
Alma Mater Studiorum—Università di Bologna a Cesena

ProMAS Workshop

Outline

- 1 Endogenous Environments in Multi-Agent Programming
- 2 An Action/Perception model for Endogenous Environments
- 3 Conclusions

Outline

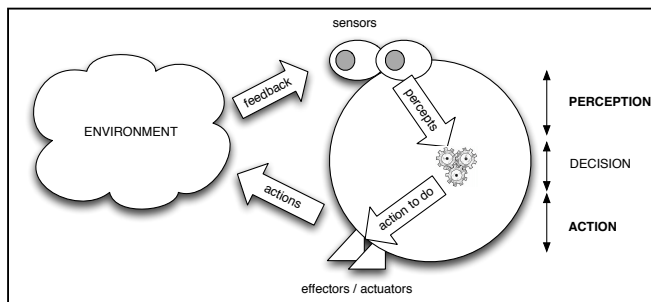
- 1 Endogenous Environments in Multi-Agent Programming
- 2 An Action/Perception model for Endogenous Environments
 - The action model
 - The perception model
- 3 Conclusions

The Notion of Environment in MAS

- First-class concept in Agent and Multi-Agent Systems
 - *“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 and Jennings, 1995]*
 - *“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]*

The Notion of Environment in MAS

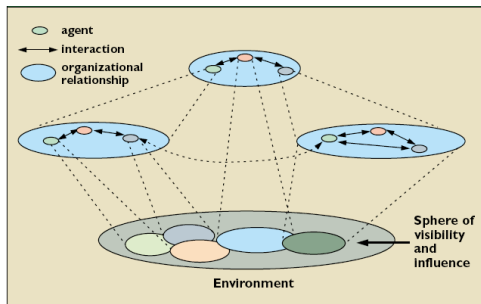
- Single-agent perspective ([Wooldridge, 2002])



- In evidence: actions and percepts

The Notion of Environment in MAS

- Multi-agent perspective ([Jennings, 2001])



- In evidence: interaction

Environment as First-Class Abstraction

AOSE Perspective

Environment as a first-class abstraction in MAS design [Weyns et al., 2007]

- providing the surrounding conditions for agents to exist
- mediates the interaction among agents
- mediates the access to resources

Environment as First-Class Abstraction

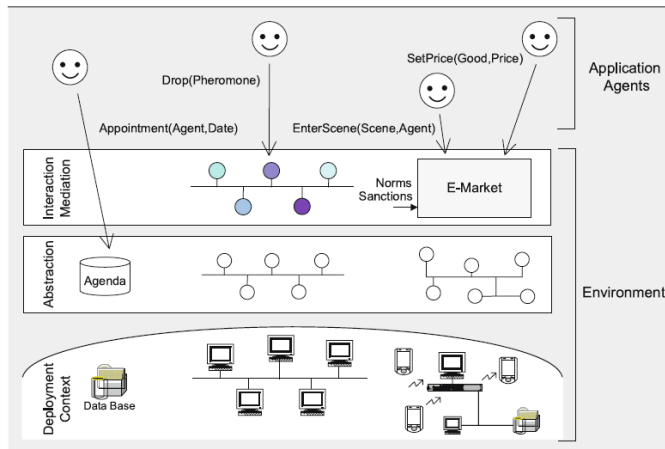


Figure: [Weyns et al., 2007]

Endogenous Environments

- Software/computational layer encapsulating functionalities
 - embedding resources and services
 - encapsulating a state and processes
 - ruling and mediating function
- Notable examples
 - Coordination infrastructures
 - e-Institutions
 - environment for stigmergic coordination

Endogenous (vs. Exogenous)

- part of the MAS
- designed by MAS designers
- used/adapted/extended by agents at runtime

Endogenous Environments in Multi-Agent Programming

- Environment Programming [Ricci et al., 2009]
 - programming MAS = programming Agents + Programming Environment
 - A&A Meta-Model [Omicini et al., 2008] and CArtAgO framework [Ricci et al., 2009]
 - artifact-based environments
- Main aspects
 - Programming/computational model for defining structure and behaviour
 - interface with the agent layer: *actions* and *perceptions*

Actions and Perceptions in Current APL: Limits

- Action/Perception Model in Current Agent Programming Languages
 - actions as events
 - percepts as a snapshot of environment state
- Outcome
 - good for exogenous environments (AI perspective)
 - not fully adequate for effectively exploiting endogenous environment features

Limits: An Example about Actions

- Using an environment with coordination functionalities
 - modeling a tuple space as an environment resource
 - the `in(TT)` action succeeds when a tuple `T` matching the template `TT` is found, getting the tuple as feedback
 - basic synchronizing behaviour
- Outcome
 - no way to model the `in` action using the actions-as-events semantics
 - need to change the semantics of the action, or
 - need to extend the language with an ad hoc action semantics

Limits: An Example about Perceptions

- Concurrent environments
 - one agent observing an environment concurrently modified by another agent
 - agents are asynchronous
- Outcome
 - observer agent can *lose* environment observable states
 - ..because of the asynchrony between environment updates and observer execution cycle

Outline

- 1 Endogenous Environments in Multi-Agent Programming
- 2 An Action/Perception model for Endogenous Environments
 - The action model
 - The perception model
- 3 Conclusions

Overview

- The model is rooted on a rigorous *Environment Contract*
 - Defining actions and action semantics
 - Defining observable state and events percepts
- Implemented in CArtAgO
 - Artifact-based environments
 - Actions $\leftrightarrow$ operations provided by artifacts
 - Perceptions $\leftrightarrow$ artifact observable properties and events

Outline

- 1 Endogenous Environments in Multi-Agent Programming
- 2 An Action/Perception model for Endogenous Environments
 - The action model
 - The perception model
- 3 Conclusions

Actions expose a rigorous and explicit contract

- A stronger action semantics (in charge of the environment)
 - Action success (failure) directly related to environment's successful (failed) operation completion
 - Action effects occurs only in case of successful action execution
- The contract defines
 - The set of actions available into the environment
 - Actions input/output parameters (action feedbacks)
- The contract guarantee
 - The *proper* performing of action's effects on the environment
 - The retrieval of the *proper* action's feedback

Actions as a process semantics

- Action modeled as a (long-term) process
 - Not a simple dispatch to agent's effectors
 - No more action as event semantics
- Actions execution can be overlapped in time
- Action completion is asynchronous
 - Completion notified by an action completion event
- During action execution the agent is *not* blocked, its control cycle goes on so:
 - It can carry on other pro-activities
 - It is reactive to percepts

Implementation in CArtAgO

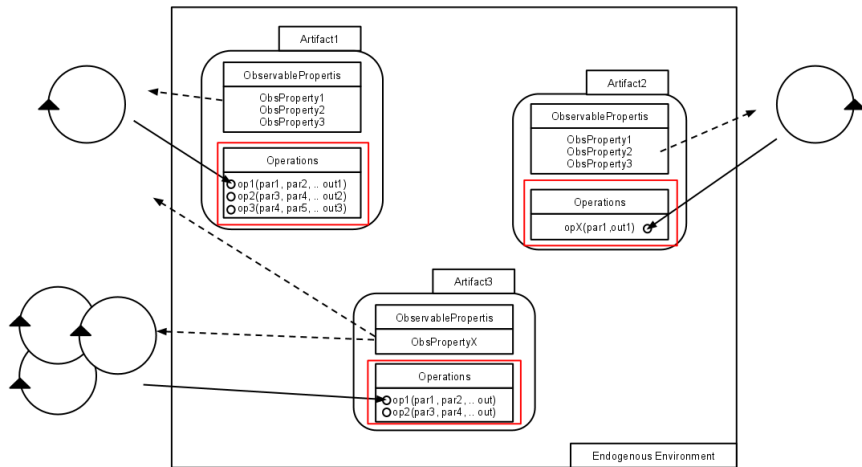


Figure: CArtAgO environment abstract representation: in red the environment actions

Endogenous Environment coordination

```
// agent using the Rendez-Vous

@meeting_plan
+!meet_altogether
  <- println("before meeting altogether.");
    ready;
    println("after meeting altogether.");
```

```
class RendezVous extends Artifact {

    int num;

    void init(int numPart){
        num = numPart;
    }

    @OPERATION void ready() {
        num--;
        await("allReady");
    }

    @GUARD boolean allReady(){
        return num == 0;
    }
}
```

Figure: (left) Snipped of a plan of Jason agent to achieve a meeting point with other agents, exploiting a RendezVous artifact; (right) source code of the RendezVous artifact.

Blocking long-term action in 2APL

```
// agent code (Tom)

Goals:
  do_job

PG-rules:
  do_job <- true |
  { @TestEnv(compute(100000), Res );
    @TestEnv(log(Res));
    dropgoal(do_job)
  }

PC-rules:
  event( my_ev(Num), simple2APLEnv)
                                     <- true |
  {
    @TestEnv(log(Num))
  }
```

```
public class TestEnv extends Environment {

  public APLNum burst(String agent, APLNum howLong)
  {
    long sum = 0;
    int burn= howLong.toInt();
    for (int i = 0; i < burn; i++){
      for (int j = 0; j < 100000; j++){
        sum+=i*j;
      }
    }
    return ( new APLNum( sum ) );
  }

  public void update(String user, APLNum n)
  {
    int nEvents = n.toInt();
    for(int i=0; i < nEvents; i++){
      APLNum evNum = new APLNum(i);
      APLFunction event = new APLFunction( "my_ev", evNum );
      throwEvent(event);
    }
  }

  public void log(String user, APLString s){
    System.out.println(s);
  }
}
```

Figure: (left) 2APL example for an agent blocked on a long term action. (right) A simple 2APL environment in Java.

Non-blocking long-term action in JaCa 1/2

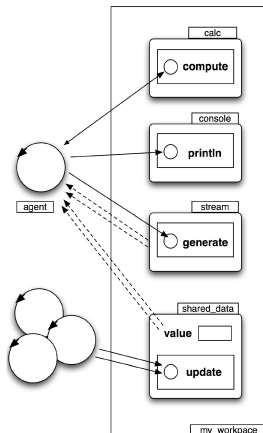


Figure: A Jason agent executing some actions to exploit the functionalities of the artifacts of a workspace, reacting to percepts related to the environment observable state and events.

Non-blocking long-term action in JaCa 2/2

```
// agent code

@main_plan
+!work_todo(X,Y) : true
  <- focus("shared_data");
    compute(X,Result);
    println(Result);
    +sum(0);
    focus("stream");
    generate(Y);
    ?sum(S);
    println(S).

+value(X)
  <- println(C).

+new_number(V) : sum(S)
  <- -+sum(S+V).
```

```
// artifacts code

class Calc extends Artifact {
  ...
  @OPERATION void compute(double x,
    OpFeedbackParam<Double> res){
    double result = longTermCalc(x);
    res.set(result);
  }
  private long longTermCalc(double x){...}
}

class SharedData extends Artifact {
  void init(){
    defineObsProperty("value",0);
  }
  @OPERATION void update(int v){
    updateObsProperty("value",v);
  }
}

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

Figure: Relevant source code fragment for the presented scenario.

Weak environment action semantics: a GOAL example

```

main: test
{
  knowledge(block(a),block(b).)

  beliefs(on(a,b).)

  goals{}

  program(if bel(on(a,b)) then act(1).)

  actionspec{
    act(X) {
      pre { true }
      post{ action_ok }
    }
  }

  perceptrules{}
}

```

```

public class TestEnv implements Environment
{
    public TestEnv() {}

    public boolean executeAction(String agt, String act)
        throws Exception {
        return false;
    }

    public ArrayList<Percept> sendPercepts(String pAgentName)
        throws Exception {
        return new ArrayList<Percept>();
    }

    public boolean availableForInput() {
        return true;
    }

    public void close() {}
    public void reset() throws Exception {}
}

```

$$\mathcal{M}(a, m) = \begin{cases} \langle \mathcal{D}, \Sigma', \Gamma \setminus Th(\mathcal{D} \cup \Sigma') \rangle & \text{if } \mathcal{D} \cup \Sigma \models pre(a, m) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $\Sigma' = \Sigma \oplus post(a, m)$ and $Th(T) = \{\varphi \in \mathcal{L} \mid T \models \varphi\}$.

$$\frac{m \models_c \psi \quad \mathcal{M}(a, m) \text{ is defined}}{m \xrightarrow{a} \mathcal{M}(a, m)}$$

Figure: (Top) Source code of the example, (bottom) a significant fragment of GOAL's semantics [Hindriks, 2009]

Outline

- 1 Endogenous Environments in Multi-Agent Programming
- 2 An Action/Perception model for Endogenous Environments
 - The action model
 - The perception model
- 3 Conclusions

Percepts as events

- Percepts now contain information about *changes* in the environment
 - No more as global snapshot of the current state of the environment
- Reconstruction of the environment's state by analyzing the list of perceived events
 - part of the agent control-loop, *sense* stage
- The event-based model guarantee the reconstruction of all the intermediate state
 - No more loss of states

From percepts to beliefs

- Endogenous environments define an explicit model to represent environment's observable parts
- These part can be directly mapped into agents' belief..
 - .. and automatically updated processing environment's events
- No more needs to use/manage belief-update function

Set of possible kinds of events

- Add or removal of an observable part
- Update of an observable part
- Signal generation

Implementation in CArtAgO

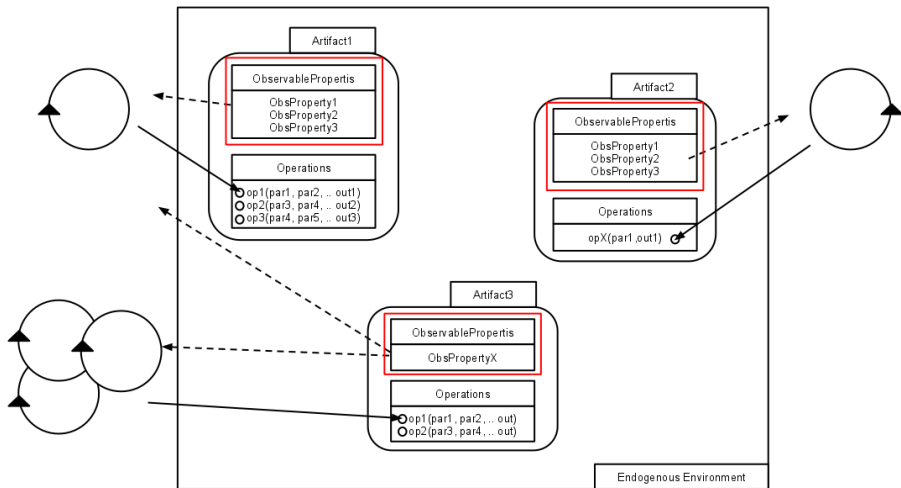


Figure: CArtAgO environment abstract representation: in red the observable part

Environment state loose in Jason

```
// worker source code

times(100).

!run_worker.
+!run_worker : times(X) & X \== 0
  <- acquire_resource;
  !use_resource;
  release_resource;
  -+times(X-1);
  !!run_worker.

-!run_worker : true
  <- .wait(10);
  !!run_worker.
```

```
// Observer agent source code

last_occurrence(0).

@my_plan [atomic]
+max_use(Occurrence) : last_occurrence(N)
  <- .print("New max usage perceived:",
            Occurrence);
  !check_state_lost(Occurrence, N);
  -+last_occurrence(Occurrence).

+!check_state_lost(Current, Local): true
  <- X = Local + 1;
  if (Current > X){
    .print("State(s) lost! from ",
           Local, " to ", Current)
  }.
}
```

```
// Jason Environment

public class ResEnv extends jason.environment.Environment {
  int occurrenceNum = 0;
  int resourceUse = 0;
  int maxResourceUse = 3;
  ReentrantLock lock;

  public void init(String[] args) {
    ReentrantLock lock = new ReentrantLock();
  }

  public boolean synchronized executeAction(String ag,
                                             Structure act) {
    if (act.getFunctor().equals("acquire_resource")){
      if (resourceUse == maxResourceUse){
        return false;
      } else{
        resourceUse++;
        if (resourceUse == maxResourceUse){
          clearPercepts();
          occurrenceNum++;
          String perc = "max_use("+occurrenceNum+")";
          addPercept(Literal.parseLiteral(perc));
        }
      }
    } else if (act.getFunctor().equals("release_resource")){
      if (resourceUse == 0) {
        return false;
      } {
        else resourceUse--;
      }
    }
    informAgsEnvironmentChanged();
    return true;
  }
}
```

No loose of environment state in JaCa

```
// worker source code

times(100).

!run_worker.
+!run_worker : times(X) & X \== 0
  <- acquire_resource;
  !use_resource;
  release_resource;
  -+times(X-1);
  !!run_worker.

-!run_worker : true
  <- .wait(10); !!run_worker.
```

```
// Observer agent source code

last_occurrence(0).
!setup.

+!setup : true
  <- make_artifact("res", "ResArtifact", Id);
  focus(Id).

@my_plan [atomic]
+max_use(Occurrence) : last_occurrence(N)
  <- .print("New max usage perceived: ",
    Occurrence);
  !check_state_lost(Occurrence, N);
  -!last_occurrence(Occurrence).

+!check_state_lost(Current, Local): true
  <- X = Local + 1;
  if (Current > X){
    .print("State(s) lost! from ",
      Local, " to ", Current)
  }.
}
```

```
// Resource Artifact implemented in CArtAgO

public class ResArtifact extends Artifact {

  int resourceUse = 0;
  int maxResourceUse;

  void init(){
    defineObsProperty("max_use", 0);
    maxResourceUse = 3;
  }

  @OPERATION void acquire_resource(){
    if(resourceUse == maxResourceUse){
      failed("error");
    } else{
      resourceUse++;
      if (resourceUse == maxResourceUse){
        int val = getObsProperty("max_use").intValue();
        updateObsProperty("max_use", val + 1);
      }
    }
  }

  @OPERATION void release_resource(){
    if(resourceUse == 0) {
      failed("error");
    } else {
      resourceUse--;
    }
  }
}
```

Outline

- 1 Endogenous Environments in Multi-Agent Programming
- 2 An Action/Perception model for Endogenous Environments
 - The action model
 - The perception model
- 3 Conclusions

Discussion

- We devised an effective action/perception model for Endogenous environment
- Exploiting all the endogenous environments functionalities
 - Embedding of resources and services
 - Environment-based coordination
- Benefits introduced
 - No more loose of environments observable states
 - No need to change the semantics of actions
 - Seamless integration of agents' pro-active and reactive behavior

Impact on the APLs side?

- Fully exploitation of the proposed model requires a proper integration between the environment and APLs
- APLs must be aware of the environment's action/perception semantics

Integration with EIS

- Realizing MASs using different APLs is for sure a relevant topic
 - E.g. older CArtaGO version provide bridge for integrating 2APL, Jason, Jadex
- Environment Interface Standard [Behrens et al., 2010] tries now to define a de-facto standard for this purpose
 - So far only integration with *exogenous* environments has been considered

EIS and CArtaGO integration

- The objective: try to define a common way for exploiting endogenous environments
- Such an integration must be respectful of the endogenous environment action/perception model

Bibliography I



Behrens, T., Jürgen, D., and Hindriks, K. V. (2010).

Towards an environment interface standard for agent-oriented programming.

Tech. rep. IfI-09-09. Dept. of Informatics, Clausthal University of Technology.



Hindriks, K. V. (2009).

Programming rational agents in goal.

In Bordini, R. H., Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., editors, *Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2*, pages 119–156. Springer.



Jennings, N. R. (2001).

An agent-based approach for building complex software systems.

Commun. ACM, 44(4):35–41.

Bibliography II



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).



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.



Russell, S. and Norvig, P. (2003).
Artificial Intelligence, A Modern Approach (second edition).
Prentice Hall.

Bibliography III



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.
Special Issue on Environments for Multi-agent Systems.



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



Wooldridge, M. and Jennings, N. R. (1995).
Intelligent agents: Theory and practice.
Knowledge Engineering Review, 10:115–152.