

Integrating Heterogeneous Agent Programming Platforms within Artifact-Based Environments

A. Ricci, M. Piunti, L.D. Acay, R. H. Bordini, J. F. Hübner, M. Dastani

speaker: Alessandro Ricci (a.ricci@unibo.it)

AAMAS 2008 Estoril, Portugal, 20080514

CONTEXT: ENVIRONMENT CONCEPT IN MAS PROGRAMMING

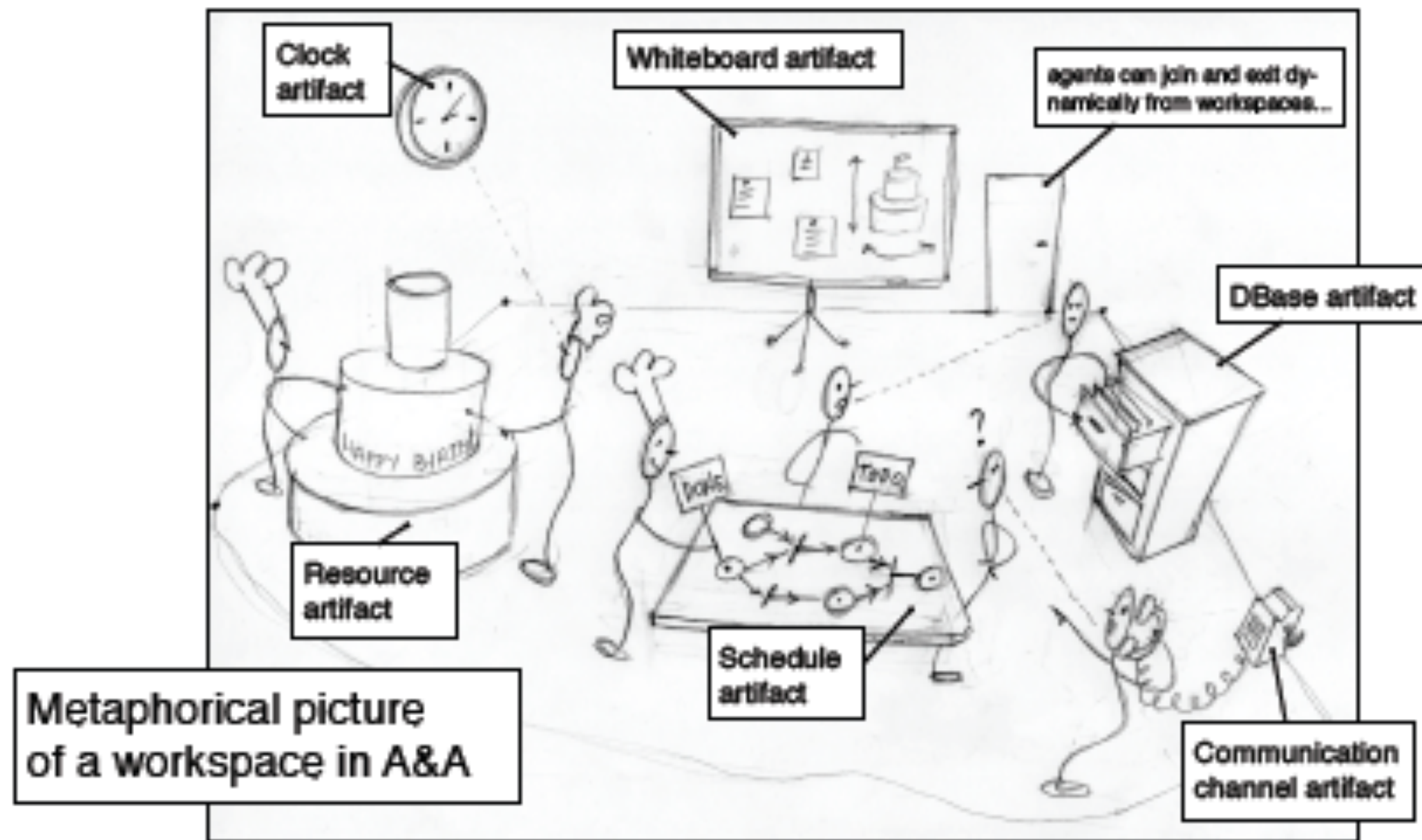
- *Environment* as first-class aspect in engineering MAS
[Weyns et al, JAAMAS special issue 07/2007]
 - mediating interaction among agents
 - encapsulating functionalities
 - coordination, organisation, security,...
- Lack of well-defined environment programming model in current cognitive agent-models and platforms
 - ~monolithic action provider / generator of percepts
 - something not really part of MAS design

CONTRIBUTION: A&A FOR COGNITIVE AGENT PLATFORMS

- Integrating existing cognitive agent platforms with a general-purpose and well-founded programming model for MAS environments
 - Agents & Artifacts (A&A) conceptual model
 - CARTAGO middleware / infrastructure
 - Agent platforms
 - *Jason*, 2APL, Jadex, simpA, JADE (ongoing)
- developing open MAS composed by heterogeneous cognitive (and reactive) agents working together inside CARTAGO environments

A&A: BASIC IDEA IN A PICTURE

- Agents, Artifacts, Workspaces
 - background in Activity Theory and Distributed Cognition



A&A MAIN ABSTRACTIONS

- **Agents**

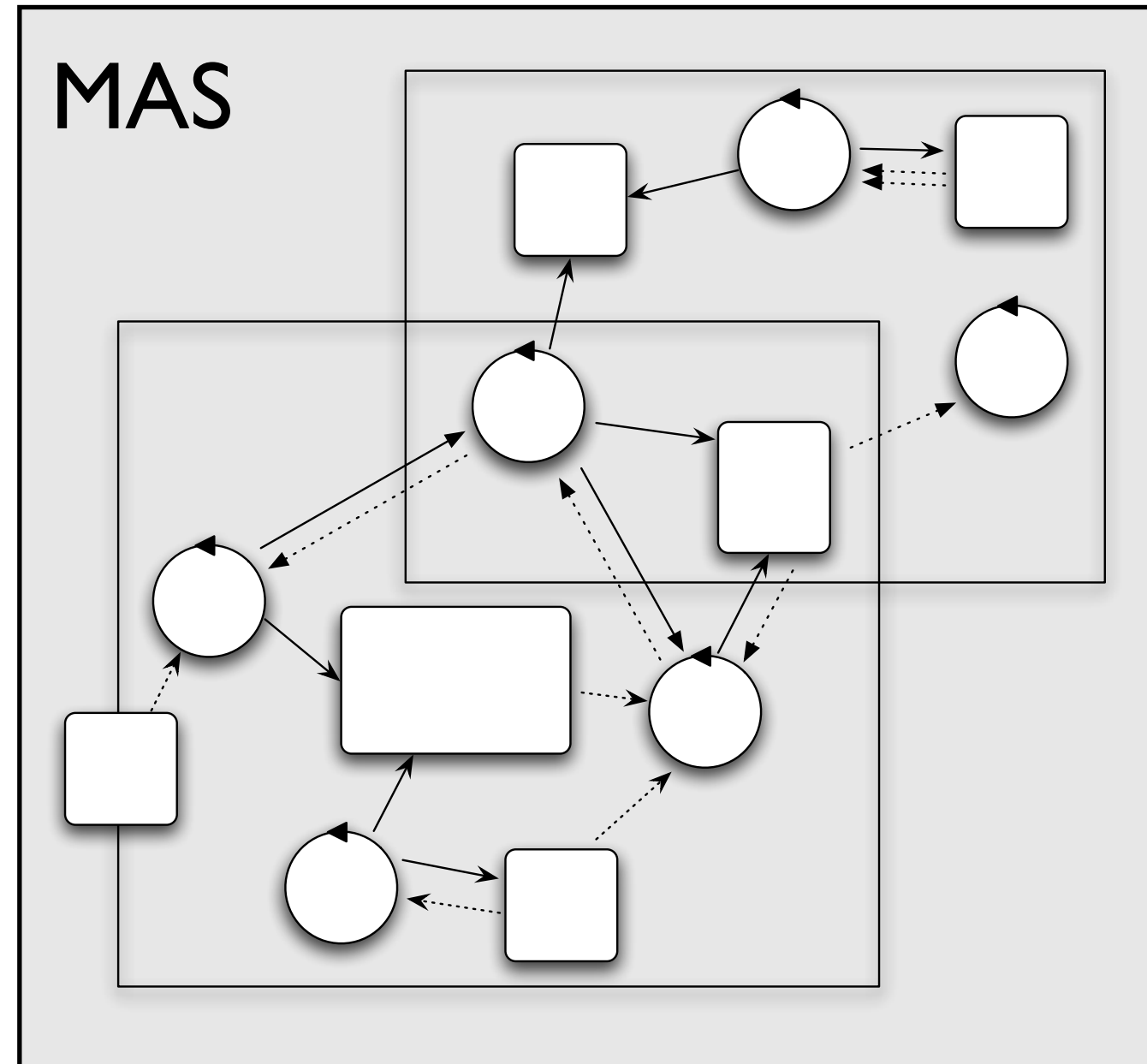
- autonomous, goal / task proactive entities
- create and co-use artifacts for supporting their activities

- **Artifacts**

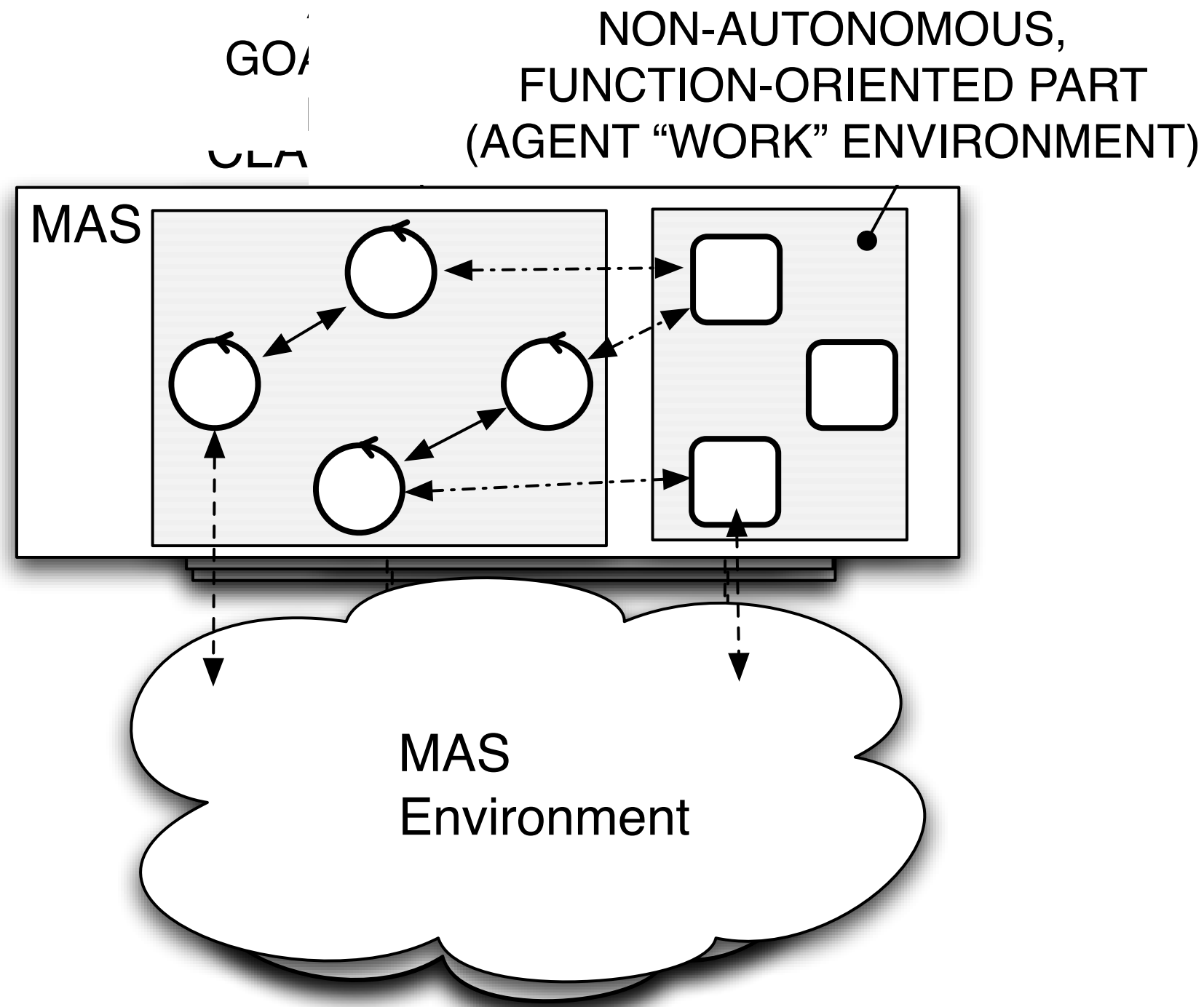
- non-autonomous, function-oriented, reactive entities
- tools, resources *used by* agents and designed by MAS programmers

- **Workspaces**

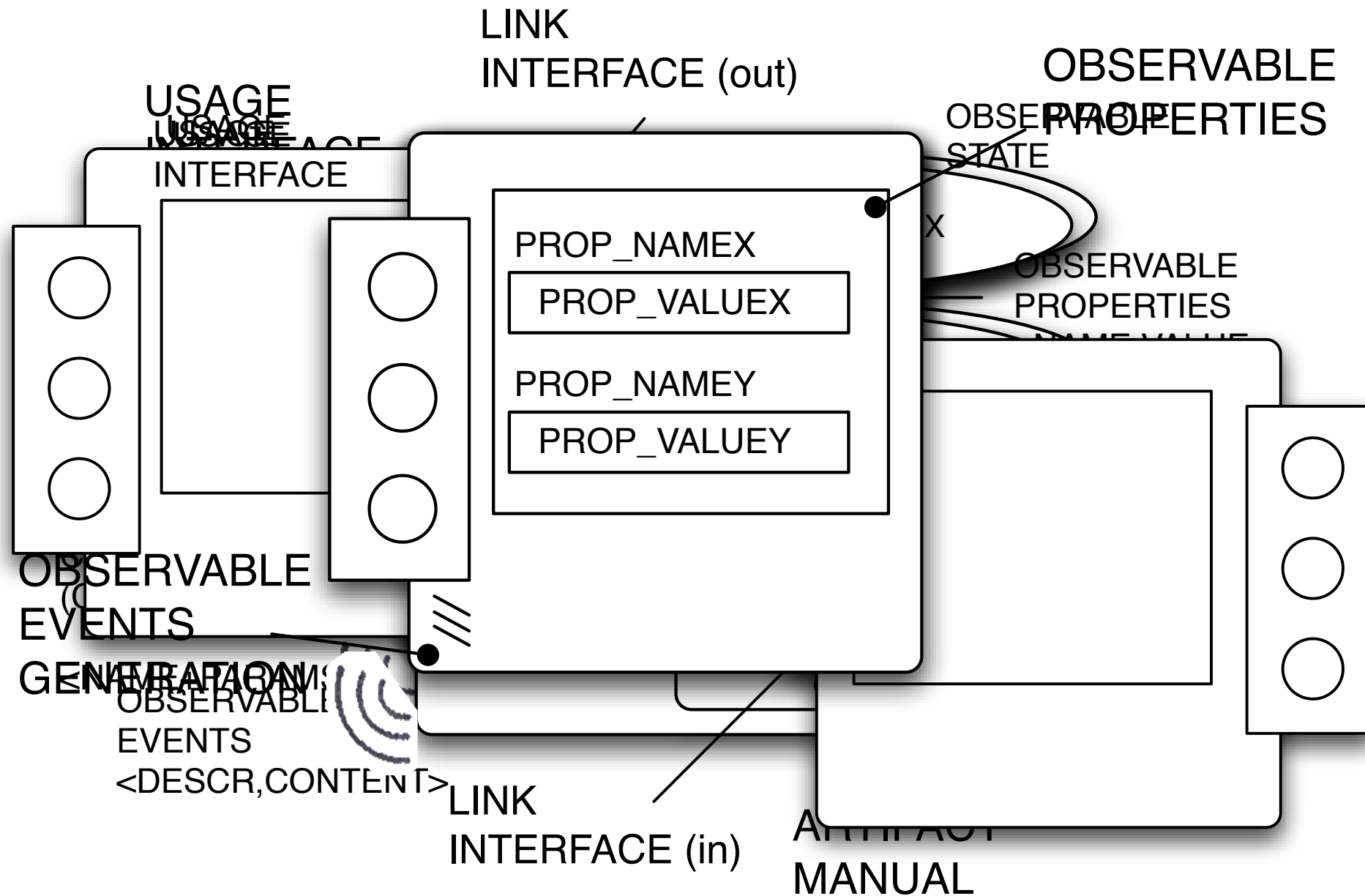
- grouping agents & artifacts
- defining the topology of the computational environment



THE NOTION OF ENVIRONMENT REVISITED



AN ABSTRACT MODEL FOR ARTIFACTS



- function description (why to use it)
- operating instructions (how-to use it)

SOME FURTHER FEATURES

- Composed operations
 - operations composed by multiple atomic computational steps, triggered by guards on artifact internal state
 - essential features for designing and programming *coordination* artifacts
- Observable states
 - usage interface can change according to artifact state
 - analogy with human artifacts: ATM
- Linkability among artifacts in distinct workspaces
 - essential feature for distributed MAS

A CONCRETE PROGRAMMING MODEL: CARTAGO

- **CARTAGO** Middleware / Infrastructure
 - Runtime distributed environment for executing artifact based environments
 - orthogonal to agent platforms
 - Java-based programming model for defining artifacts
- Open-source technology
 - available at <http://www.alice.unibo.it/cartago>

PROGRAMMING ARTIFACTS IN CARTAGO: A SIMPLE EXAMPLE

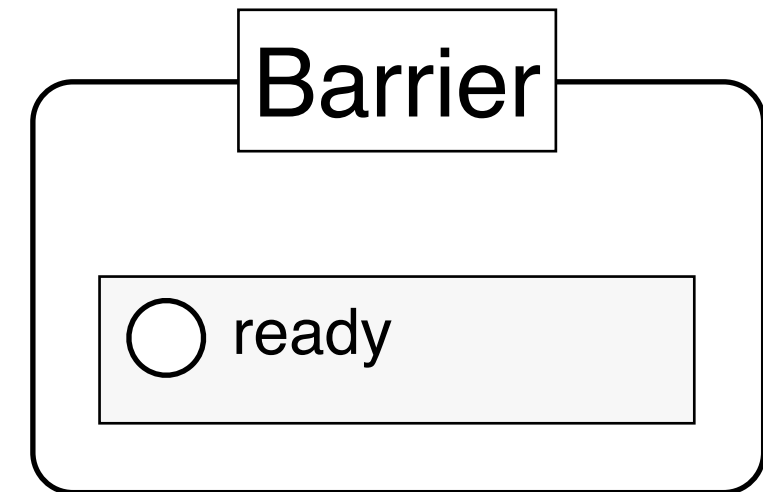
```
public class Barrier extends Artifact {
    int nParticipants;
    int nReady;

    @OPERATION void init(int nParticipants){
        nReady = 0;
        this.nParticipants = nParticipants;
    }

    @OPERATION void ready() {
        signal("ready_accepted");
        nReady++;
        nextStep("notifyAllReady");
    }

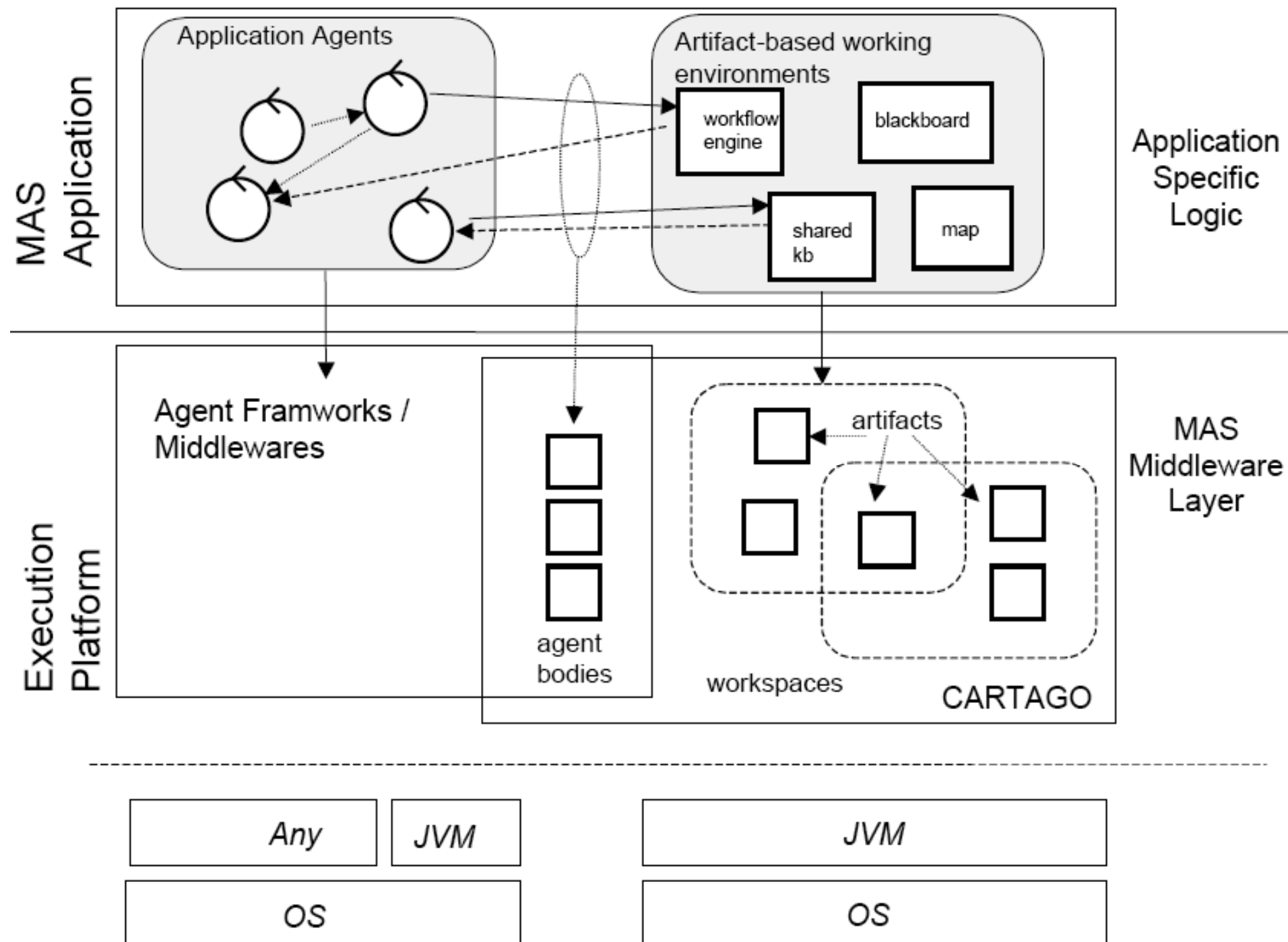
    @OPSTEP(guard="allReady") void notifyAllReady(){
        signal("all_ready");
        nReady = 0;
    }

    @GUARD boolean allReady(){
        return nReady==nParticipants;
    }
}
```



Usage interface:
ready(): {ready_accepted,
all_ready}

CARTAGO ARCHITECTURE



INTEGRATING A&A WITHIN COGNITIVE AGENT MODELS

- Basic set of actions to work within workspaces:

(1) `lookupWsp(WName,?Wid,+Node)`

(2) `joinWsp(Wid)`

(3) `quitWsp(Wid)`

(4) `createWsp(WName,+Node)`

(5) `removeWsp(WName,+Node)`

**workspace
managem.**

(6) `lookupArtifact(AName,?Aid)`

(7) `use(Aid,OpCntrlName(Params),+Timeout,+Filter)`

(8) `use(Aid,OpCntrlName(Params),Sid,+Timeout,+Filter)`

(9) `sense(Sid,?Perception,+Filter,+Timeout)`

(10) `makeArtifact(Aid,+ATemplate,+AConfig)`

(11) `disposeArtifact(Aid)`

**artifact
use**

(12) `focus(Aid,+Filter)`

(13) `focus(Aid,Sid,+Filter)`

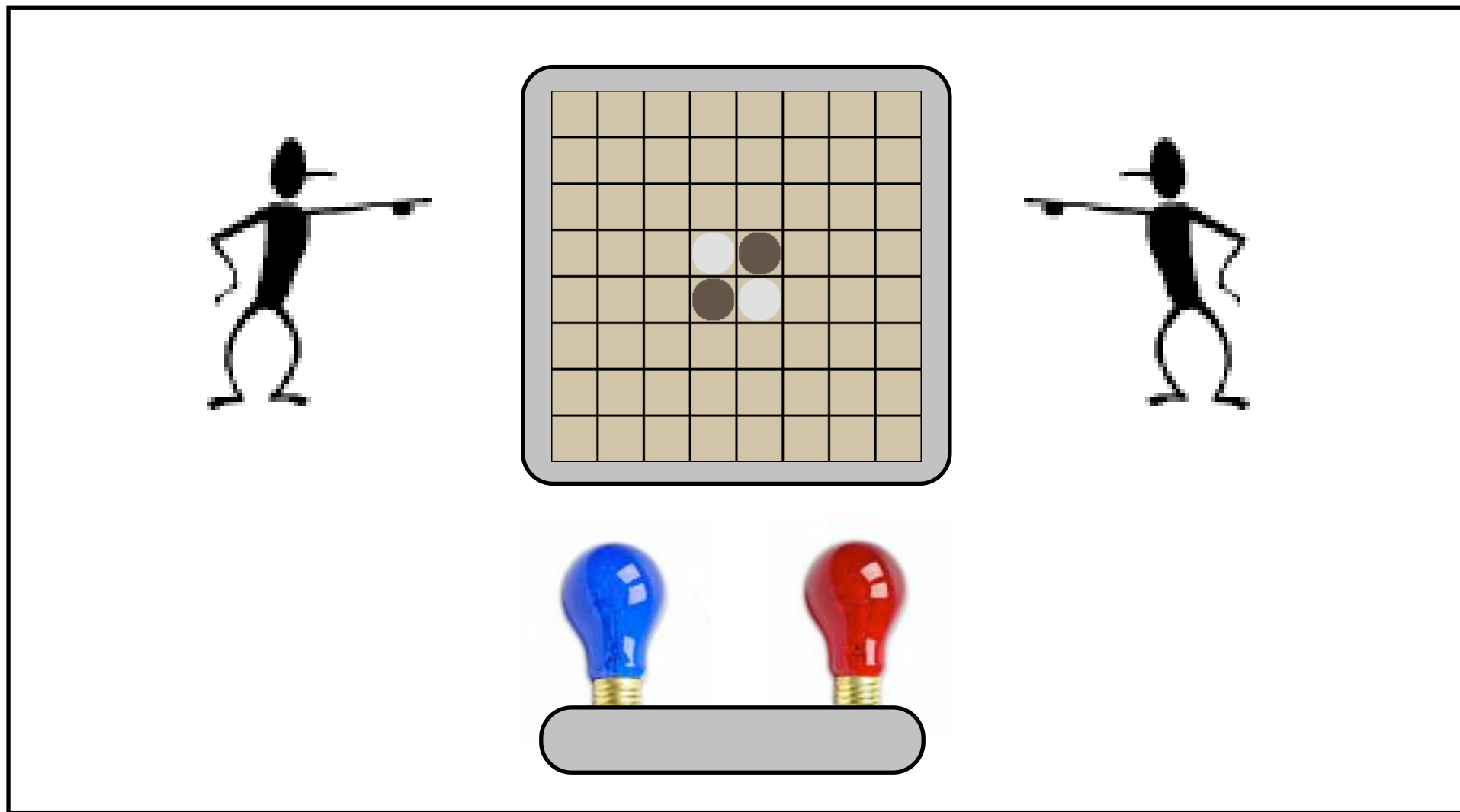
(14) `stopFocussing(Aid)`

(15) `observeProperty(Aid,PFilter,?AProperty)`

**artifact
observation**

AN EXAMPLE

- Agents playing a game (e.g. “reversi”)
 - **gb** artifact to represent the game board
 - **lamp** artifact to signal the turn



GAMEBOARD AND LAMP ARTIFACTS

gb

- ☐ move
- ☐ getContent
- ☐ isMoveAllowed

Gameboard Usage Interface

Operations and Observable Events possibly generated:

```
move(X,Y,X1,Y1):  
    {move_ok(X,Y,X1,Y1),wrong_move(X,Y,X1,Y1),  
     winner(Who)}  
getContent(X,Y):{cell_content(X,Y,{white|black|empty})}  
isMoveAllowed(X,Y,X1,Y1):{yes,no}
```

No Observable Properties

lamp

turn

Lamp Usage Interface

No Operations

Observable Properties:

```
turn({black|white|none})
```

JASON AGENT PLAYING THE GAME

Actions syntax:

```
cartago.use(+Aid,+Op(Params){,+Sid})  
cartago.sense(+Sid,?Percept,+Timeout)  
cartago.focus(+Aid,Sid)
```

....

observed events:
artifact_event(Aid,Event)

```
myColour(white). // an initial belief  
  
+!play  
  <- ...  
    cartago.focus(lamp);  
    ...  
  
+artifact_event(lamp,property_changed(turn(Who)))  
  : myColour(Who)  
  <- ...  
    // inspect the gameboard at X,Y  
    cartago.use(gb,getContent(X,Y),s0) ;  
    cartago.sense(s0,cell_content(X,Y,Content),1000)) ;  
    ...  
    <decide to place a piece in some position MX,MY>  
    ...  
    // note that MX and MY are now already bound  
    !perform_move(Who,MX,MY).
```

```
+!perform_move(C,X,Y)  
  <- ...  
    cartago.use(gb,move(C,X,Y)).  
  
+artifact_event(gb,wrong_move(C,X,Y)  
  <- ...  
    <select another move>  
    ...  
+artifact_event(gb,winner(Who))  
  : myColour(Who)  
  <- cartago.use(console,print("I won!")).  
  
+artifact_event(gb,winner(Who))  
  : not myColour(Who)  
  <- cartago.use(console,print("Damm it.")).
```


2APL AGENT PLAYING THE GAME

Actions syntax:

```
use(+Aid,+Op(Params){,+Sid})  
sense(+Sid,?Percept,+Timeout)  
focus(+Aid,Sid)
```

....

observed events:
event(Aid,Event)

```
Beliefs:  
  myColour(white).  
  
PC-rules:  
play <- true |  
{ ...  
  focus(lamp);  
  ...  
}  
event(lamp,property_changed(turn(Who))) <- myColour(Who) |  
{ ...  
  // inspect the gameboard at X,Y  
  use(gb, getContent(X,Y), s0);  
  sense(s0, cell_content(X,Y,Content), 1000);  
  ...  
  <decide to place a piece in some position MX,MY>  
  ...  
  // note that MX and MY are now already bound  
  perform_move(Who,MX,MY)  
}
```

```
PC-rules:  
perform_move(C,X,Y) <- true |  
{ ...  
  use(gb,move(C,X,Y))  
}  
  
event(gb,wrong_move(C,X,Y)) <- true |  
{ <select another move> }  
  
event(gb,winner(Who)) <- myColour(Who) |  
{ use(console,print("I won!")) }  
  
event(gb,winner(Who)) <- not myColour(Who) |  
{ use(console,print("Damn it.!!")) }
```


BENEFITS FOR MAS DEVELOPMENT

- Providing cognitive agent programmers a uniform general purpose programming model for programming agent environments
 - keeping the agent abstraction level
 - artifacts are not objects in OO
 - encapsulation, modularity
 - enhancing reuse
 - artifact libraries (coordination, organisation, GUI, etc.)
 - reusing / wrapping legacy and non-agent technologies
 - e.g. Artifacts wrapping Web Services access and management
- Enabling a new level of *interoperability* in MAS
 - heterogeneous agents interacting by means of working in the same workspace, sharing and using artifacts

A MORE COMPLEX EXAMPLE: ROOMS WORLD

- Distributed scenario
 - multiple rooms to be cleaned in a building
 - room cleaner agents looking for rooms to clean
 - basic strategy: periodically visit rooms and check for trash
- Introducing artifacts to make more effective room cleaners coordination
 - checklist
 - to store the date for last (& previous) clean-up
 - *clock*
- Multiple workspaces, one per room
 - a checklist and a clock per workspace

ROOM-CLEANER STRATEGY

- Use the checklist and the clock to avoid cleaning rooms recently cleaned

```
+!useChecklist
  <- ?target_check_list(N);
  goToChecklist(N);
  -target_check_list(N);
  cartago.use(checklist(N), readLastNote,s0);
  cartago.observeProperty(watch,current_time(_),s0);
  cartago.sense(s0,last_note(LastTime)),
  cartago.sense(s0,current_time(CurrentTime)),
  !decide(N,LastTime,CurrentTime).

+!decide(N,LastTime,CurrentTime)
  : threshold(D) & (D < CurrentTime - LastTime)
  <- cartago.use(console,"I'VE DECIDED TO ENTER!");
  !clean(N).

+!decide(N,LastTime,CurrentTime)
  <- cartago.use(console,"INTENTIONS RECONSIDERED: EXPLORING!");
  !explore.
```

REMARKS

- Simplifying agent choices and ease computational burden
 - artifacts allow for externalise and distribute part of agent activities
 - *checklists* example
 - agents may rely on checklist information to improve their practical behaviour, saving exploration and epistemic activities
- Benefits of mediated interaction and coordination
 - *checklists* example
 - organise and make available relevant information as permanent modification of their state, persistent over agent presence
 - no need for agents of mutual presence within a location or heavy message exchange protocols

ONGOING WORK AND NEXT STEPS

- Autonomous selection / use / creation of artifacts with cognitive agents, towards truly *open* systems
 - exploiting artifact manuals and artifact observability
 - Extrospective agents [L. D. Acay et al.], ProMAS 2008
- Definition of a formal model for artifacts and A&A
 - based on I/O automata-like approaches
 - exploring impact on MAS validation
- Using the integration for tackling MAS main issues
 - e.g. Open MAS Organisation
 - ORA4MAS infrastructure [Boissier, Hübner, Kitio], EUMAS 2007

<END/>