ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

WOA 2009 MINI-SCUOLA

ENVIRONMENT PROGRAMMING IN MULTI-AGENT SYSTEMS

Alessandro Ricci

aliCE group at DEIS, Università di Bologna, Cesena

a.ricci@unibo.it

OUTLINE

- **PART ONE** / Background
  - the basic concept of environment in MAS
- **PART TWO** / Environment as First-Class Abstraction
  - AOSE and MAS engineering perspective
- **PART THREE** / Environment Programming
  - MAS programming perspective
  - Practice with CArtAgO + Jason

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

WOA 2009 MINI-SCUOLA
ENVIRONMENT PROGRAMMING IN
MULTI-AGENT SYSTEMS

PART ONE

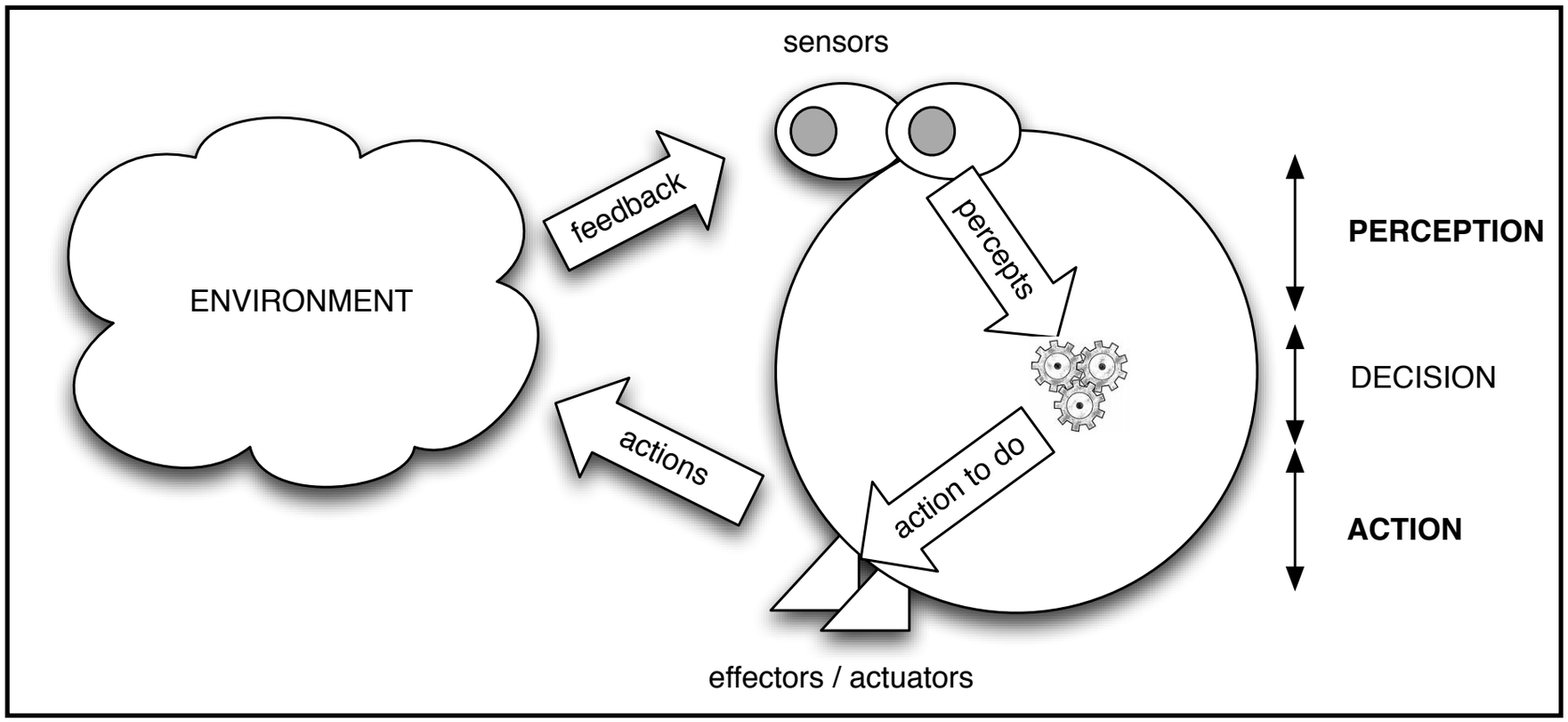
THE CONCEPT OF ENVIRONMENT IN MULTI-AGENT SYSTEMS

- BACKGROUND -

THE NOTION OF ENVIRONMENT

- The notion of environment is intrinsically related to the notion of 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 & Jennings, 1995]
  - *“An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors."* [Russell & Norvig, 1995]
- Including both physical and software environment
  - “computational”, ”virtual” environments

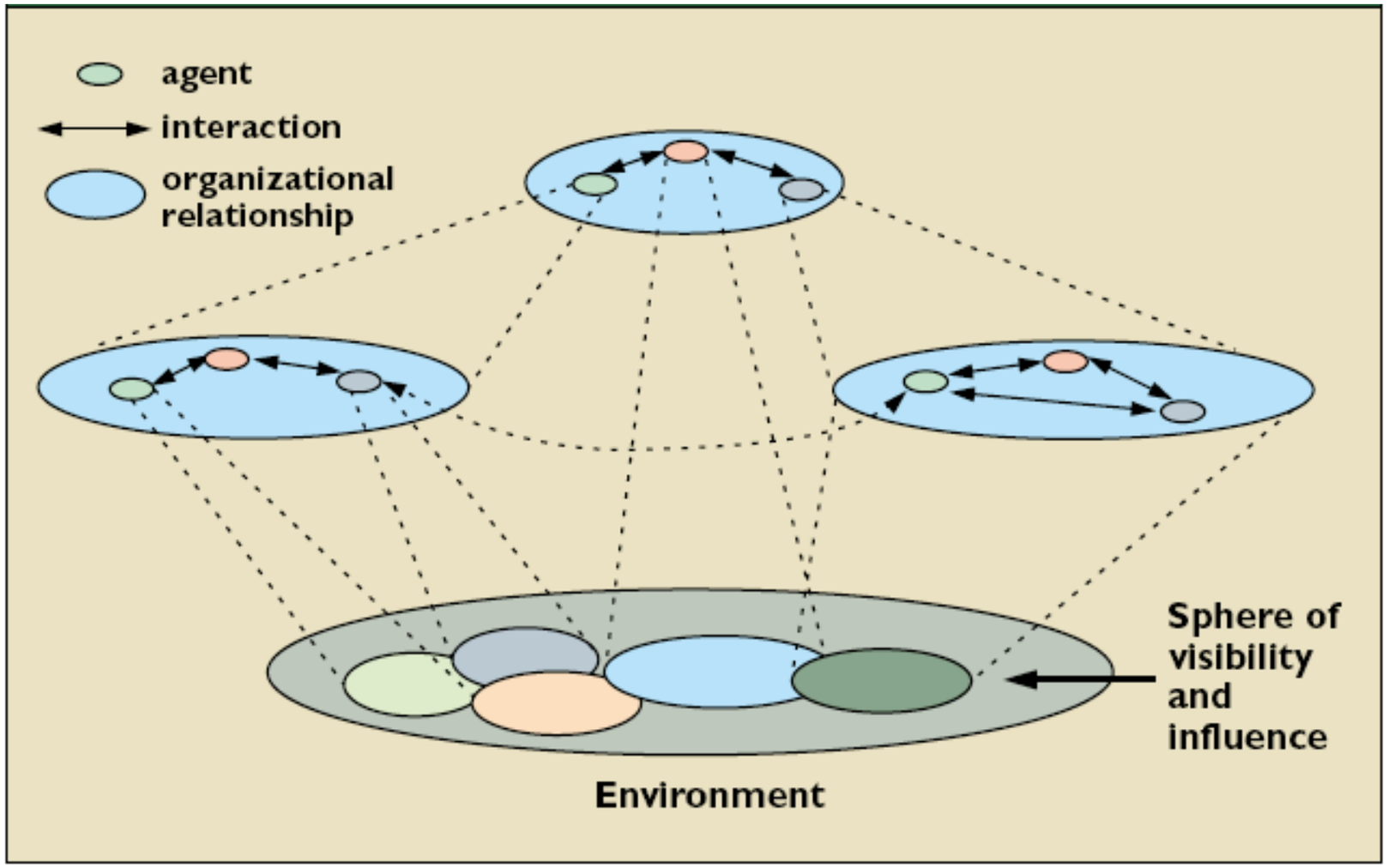
SINGLE-AGENT PERSPECTIVE



[Wooldridge, 2002]

- 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



[Jennings, 2001]

- In evidence
  - spheres of visibility and influence
  - overlapping => interaction

ABSTRACT “ENVIRONMENT PROGRAM”

```
procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
  inputs: state, the initial state of the environment
          UPDATE-FN, function to modify the environment
          agents, a set of agents
          termination, a predicate to test when we are done

  repeat
    for each agent in agents do
      PERCEPT[agent] ← GET-PERCEPT(agent, state)
    end
    for each agent in agents do
      ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
    end
    state ← UPDATE-FN(actions, agents, state)
  until termination(state)
```

[Russell and Norvig,1993]

A BASIC CLASSIFICATION

- Accessible versus inaccessible
  - indicates whether the agents have access to the complete state of the environment or not
- Deterministic versus nondeterministic
  - indicates whether a state change of the environment is uniquely determined by its current state and the actions selected by the agents or not
- Static versus dynamic
  - indicates whether the environment can change while an agent deliberates or not
- Discrete versus continuous
  - indicates whether the number of percepts and actions are limited or not

FURTHER CLASSIFICATION

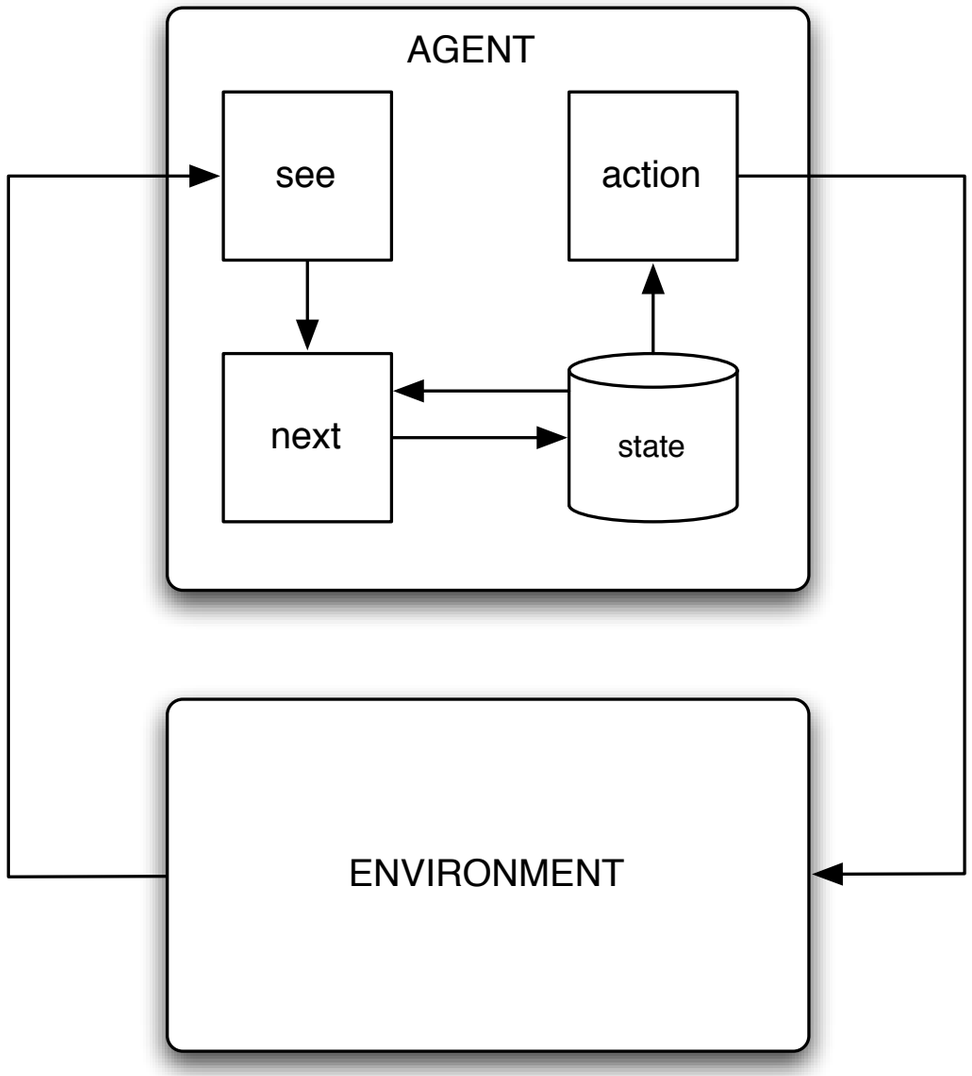
[Ferber, 1999]

- Centralized vs. distributed
  - centralized environment: a single monolithic system
    - all agents have access to the same structure
  - distributed environment: as a set of *places* assembled in a network
    - the state of a place depends on the surrounding places
    - the perception of agents is related to one or multiple places
    - support for agent mobility, moving from place to place
- Generalized vs specialized
  - a generalized model is independent of the kind of actions that can be performed by agents
  - a specialized model is characterized by a well-defined set of actions

COMMUNICATIVE & SITUATED MAS

- Purely communicative MAS
  - agents can only communicate by message transfer
- Purely *situated* MAS
  - agents can only act in the environment
- Combination of communicating and situated MAS

ACTION AND PERCEPTION IN AGENT ABSTRACT ARCHITECTURE



- Perception function ***see***
  - representing the ability to obtain information from its environment

    $see : E \rightarrow Per$
- Action-selection function ***action***
  - mapping from internal states to actions

    $action : I \rightarrow Ac$
- Update state ***next*** function
  - mapping an internal state and a percept to an internal state

    $next : I \times Per \rightarrow I$

ENVIRONMENT & REACTIVE AGENTS

- "Reactive" agents
  - no symbolic representations, no reasoning or symbolic manipulations in decision-making
- Strongly-based on the notion of environment
  - "reactive" ~ "***situated***"
    - reactive agents are typically situated in some environment rather being disembodied
  - agent "intelligent" behaviour is seen as innately linked to the environment agents occupy
  - the behaviour is a product of the ***interaction*** the agents maintain with their environment

ENVIRONMENT & INTELLIGENT AGENTS

- Intelligent agents [Wooldridge and Jennings, 1995]
  - reactivity
    - intelligent agents are able to *perceive the environment* and respond in a *timely fashion* to changes that occur in its order to satisfy their design objectives
  - pro-activeness
    - intelligent agents are able to exhibit *goal-directed behaviour* by taking the initiative in order to satisfy their design objectives
    - goals are defined as ***state of affairs of the environment*** to bring about
  - social ability
    - intelligent agents are capable of interacting with other agents in order to satisfy their design objectives
    - role of ***mediated interaction***

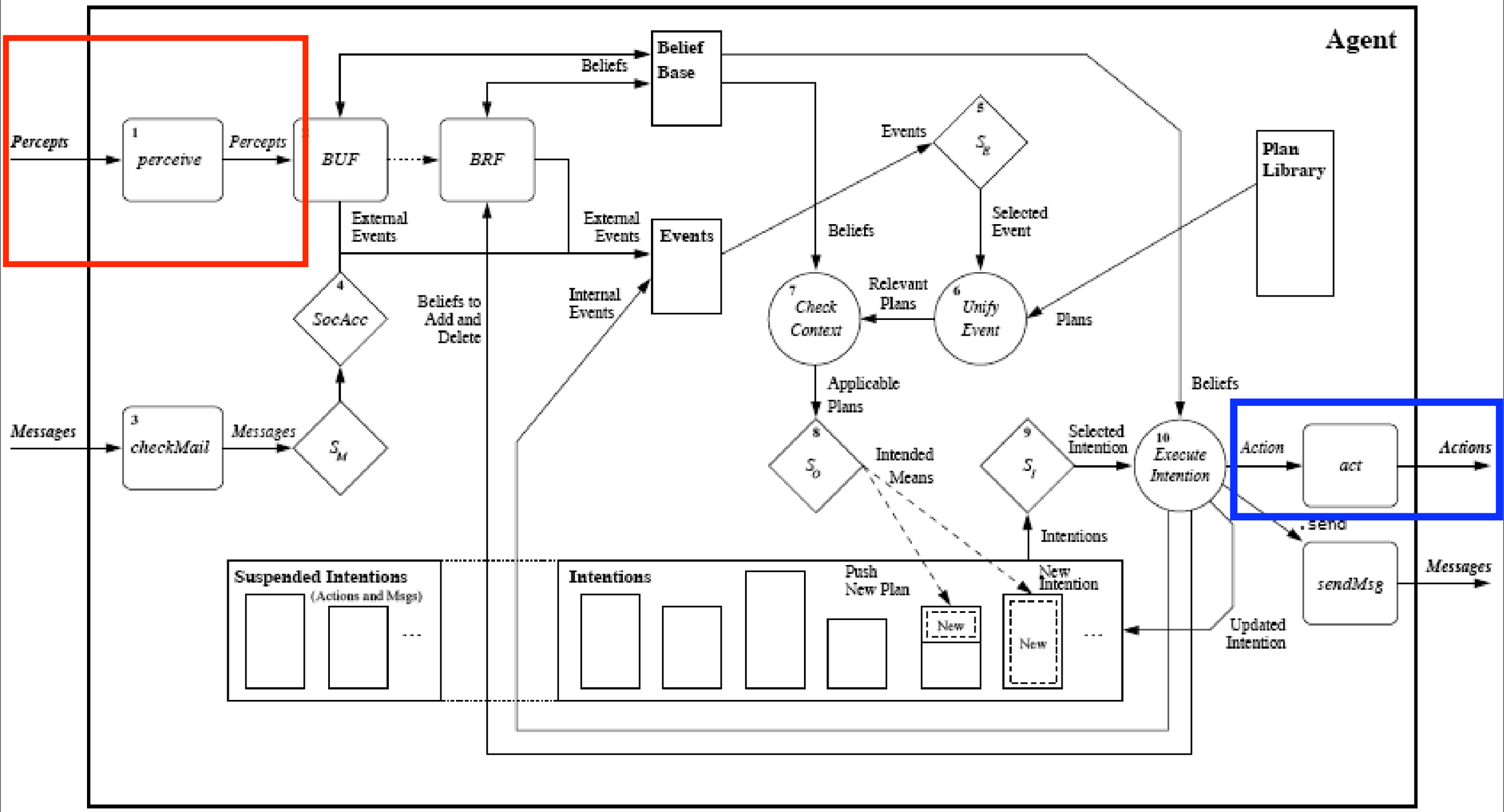
ACTIONS AND PERCEPTIONS IN BDI AGENTS

- BDI agent model/architecture
  - agent programs specified in terms of mental attitudes (beliefs, desires, intentions)
  - based on *practical reasoning*
    - deliberation
      - deciding *what* states of affairs to achieve / mantain
    - means-ends reasoning
      - *how* to achieve/mantain the states of affairs
- Action and perception stage as part of *the execution cycle* (or control loop)

AGENT EXECUTION CYCLE PSEUDO-CODE

```
B ← B0;          /* B0 are initial beliefs */
I ← I0;          /* I0 are initial intentions */
while true do
    get next percept ρ through see(...) function;
    B ← brf(B, ρ);
    D ← options(B, I);
    I ← filter(B, D, I);
    π ← plan(B, I, Ac);
    while not (empty(π) or succeeded(I, B) or impossible(I, B)) do
        α ← hd(π);
        execute(α);
        π ← tail(π);
        get next percept ρ through see(...) function;
        B ← brf(B, ρ);
        if reconsider(I, B) then
            D ← options(B, I);
            I ← filter(B, D, I);
        end-if
        if not sound(π, I, B) then
            π ← plan(B, I, Ac);
        end-if
    end-while
end-while
```

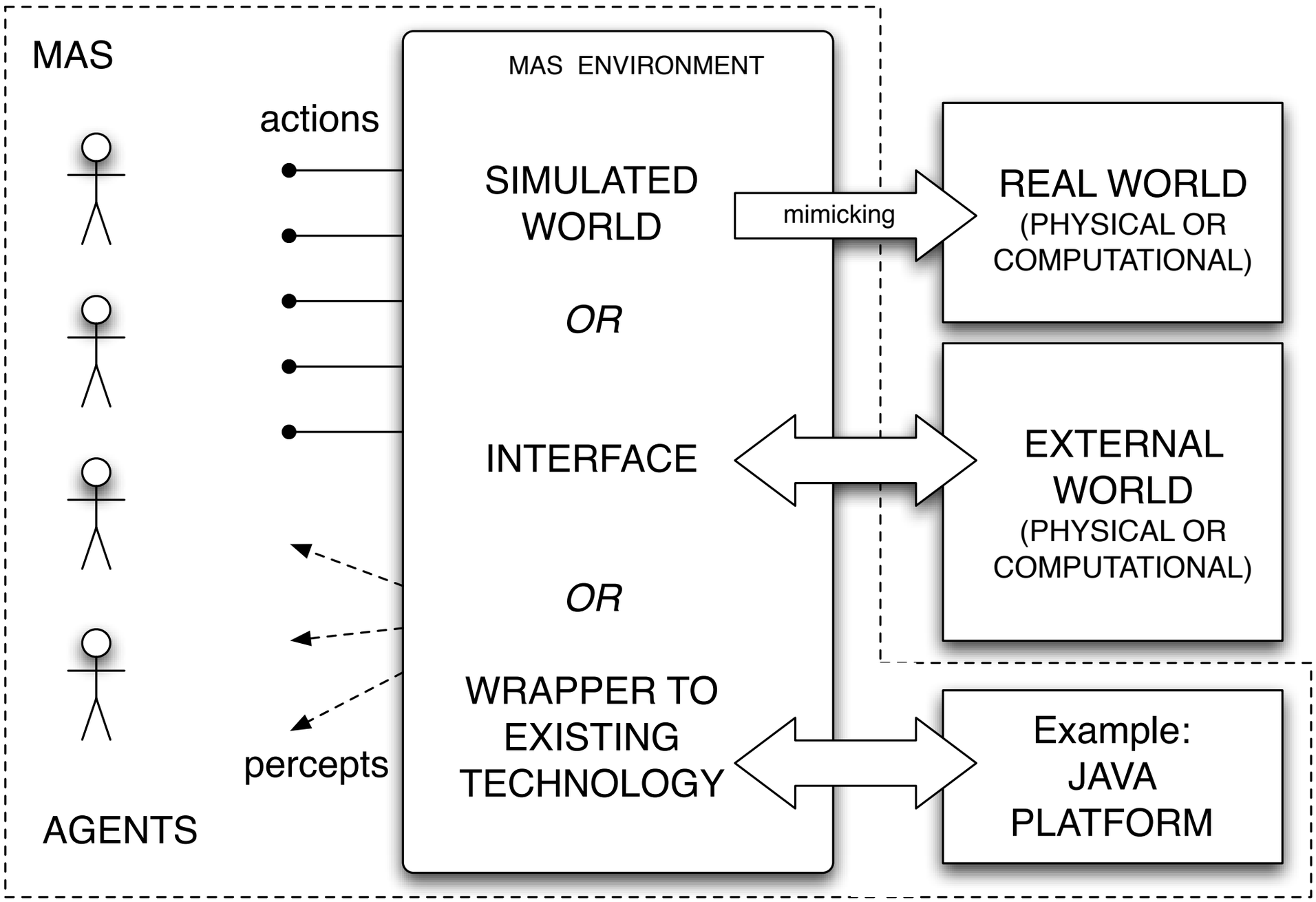
EXECUTION CYCLE IN A CONCRETE ARCHITECTURE: JASON CASE



ENVIRONMENT IN AGENT-ORIENTED PROGRAMMING

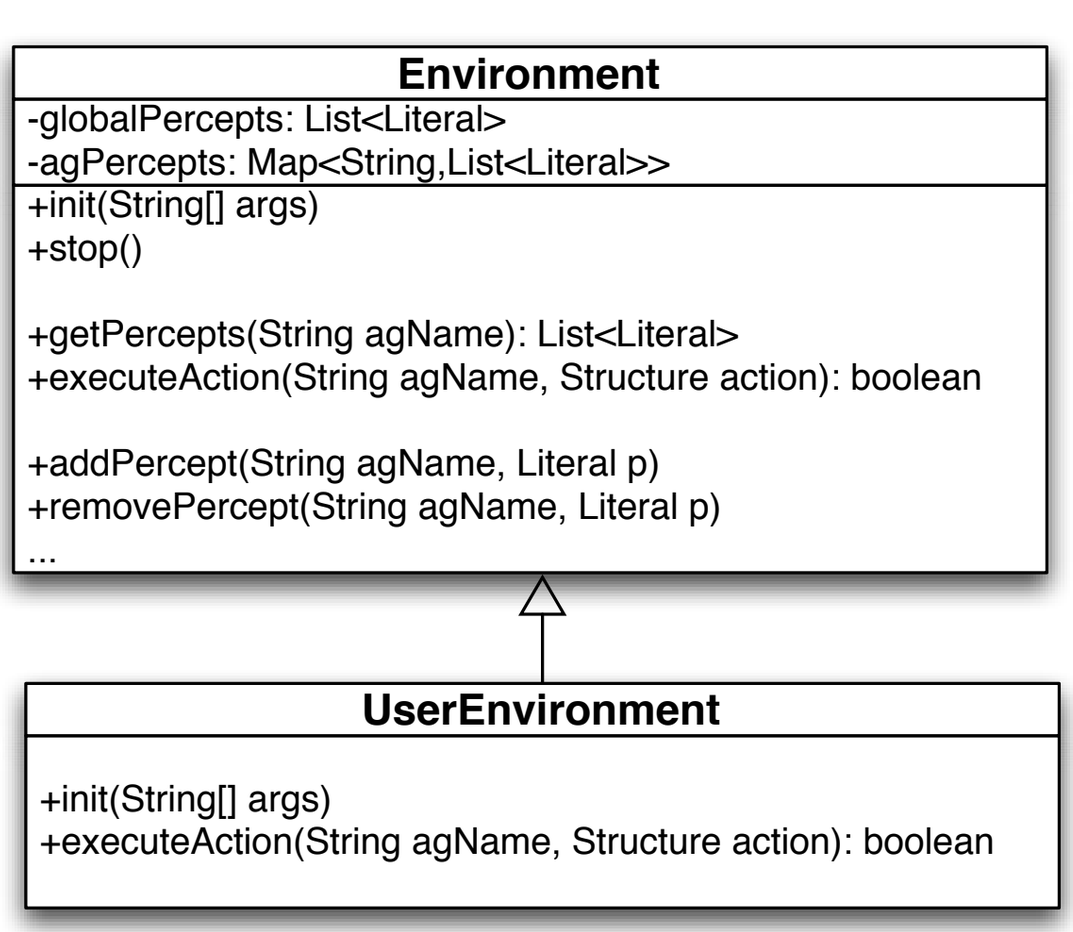
- Agent-Oriented Programming perspective
  - agents & MAS as a paradigm for programming
    - strongly related to Agent-Oriented Software Engineering
  - many languages / platforms, many different perspectives
    - Agent-0, Placa, April, Concurrent Metatem, ConGolog / IndiGolog, AgentSpeak, AgentSpeak(L) / Jason, 3APL, IMPACT, Claim/Sympa, 2APL, GOAL, Dribble, etc..
    - Jack, JADE, JADEX, AgentFactory, Bhrams, JIAC, etc
- Environment support
  - typically minimal
    - most of the focus is on agent architecture & agent communication
  - in some cases: basic environment API
    - for “customising” the MAS with a specific kind of environment
    - Jason, 2APL, Jadex

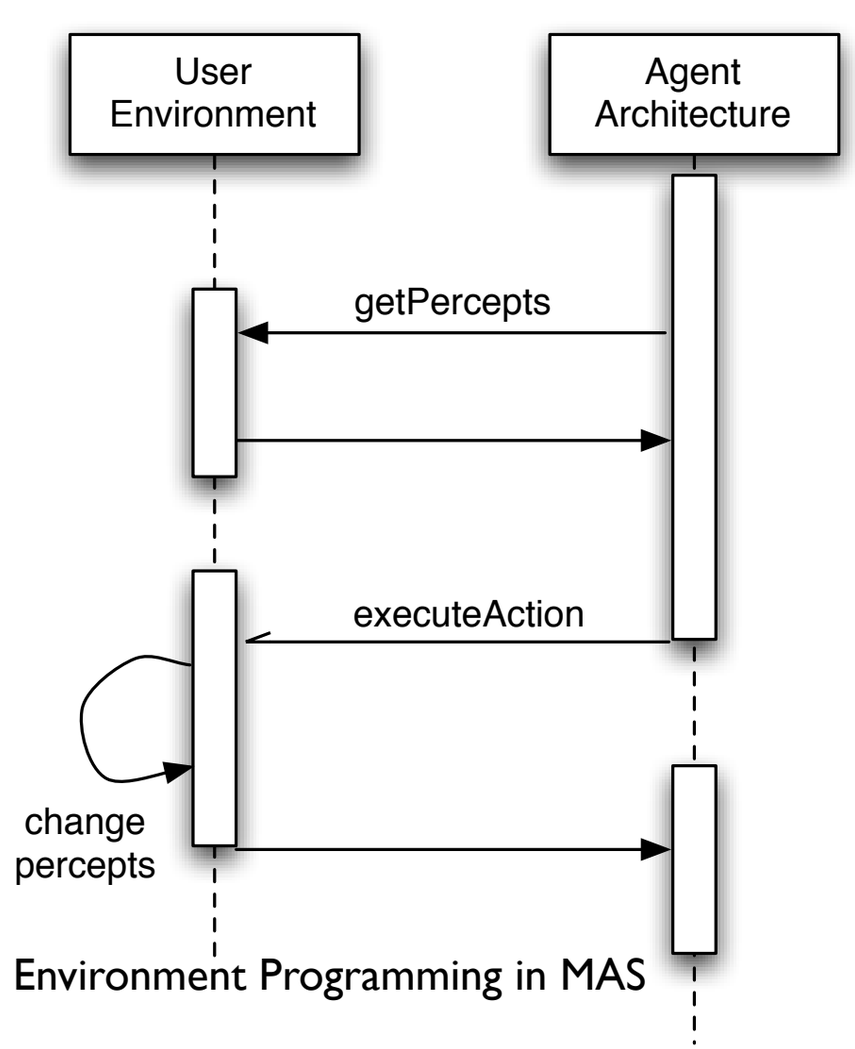
ENVIRONMENT API IN AOP (GENERAL PERSPECTIVE)



ENVIRONMENT API IN JASON

- Jason [Bordini, Hübner, Woodridge - 2007]
  - Java-based open-source interpreter of an extended version of AgentSpeak(L) for programming MAS
    - BDI architecture
  - fully customisable agent architecture
    - basic environment API to define customised environments





AN EXAMPLE IN JASON: ENV. SIDE

```
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 { ... }
}
```

AN EXAMPLE IN JASON: AGENT SIDE

```
// 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).
```

ENVIRONMENT API: REMARKS

- In most cases: no direct support
  - indirectly support by lower-level implementing technology
    - e.g. Java
- In some cases: first environment API
  - useful to create simulated environment or to interface with external resources
  - simple model: a single / centralised object
    - defining agent (external) actions
      - typically a static list of actions, shared by all the agents
    - generator of percepts
      - establishing which percepts for which agents

WRAP-UP

- Classic view about the environment in MAS
  - something outside agents and the MAS
    - outside MAS design and development
  - the context where agents' activities take place
    - the context upon which agent goals are defined
  - *single agent-perspective* as the main perspective
- Mostly inherited from (D)AI

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

WOA 2009 MINI-SCUOLA

ENVIRONMENT PROGRAMMING IN MULTI-AGENT SYSTEMS

PART TWO

ENVIRONMENT AS FIRST-CLASS ABSTRACTION

CHANGING THE PERSPECTIVE

- From the (D)AI-oriented perspective...
  - environment as a part outside the MAS
    - out of MAS design and development
- ...to AOSE and MAS programming perspective
  - all *non-agent* elements of the MAS are typically considered to be part of the MAS environment
    - resources, services, infrastructure,...
- Variety of responsibilities in MAS
  - communication, coordination, organisation
  - specific examples
    - coordination infrastructures
    - e-Institutions

COORDINATION INFRASTRUCTURES

- Providing functionalities for MAS coordination
  - mechanisms to *enable* agent interaction
    - including agent indirect communication
  - mechanisms to *manage* the interaction (~coordination)
    - shaping the interaction space to achieve some global coordinated behaviour
- Specific examples
  - stigmergic coordination
    - computational fields, digital pheromone infrastructure
  - tag-based interactions
  - space-based coordination middleware

STIGMERGIC COORDINATION

- Agents coordinate their behaviour through the manipulation of *marks* in the environment in a similar way as social ants coordinate
  - abstract notion of mark as information signs
- Environment is the coordination medium mantaining processes independent from agents' activities
  - providing actions to create and perceive marks
  - embedding processes manipulating mark
    - analogous to aggregation, diffusion, evaporation

STIGMERGIC APPROACHES: DIGITAL PHEROMONES

- Digital pheromones [Parunak, Brueckner, Sauter - 2005]
  - distributed environment implementing dynamic structure aggregating, diffusing, evaporating *pheromones* (which are software objects)
  - software agents can use pheromones to dynamically form paths to locations of interest

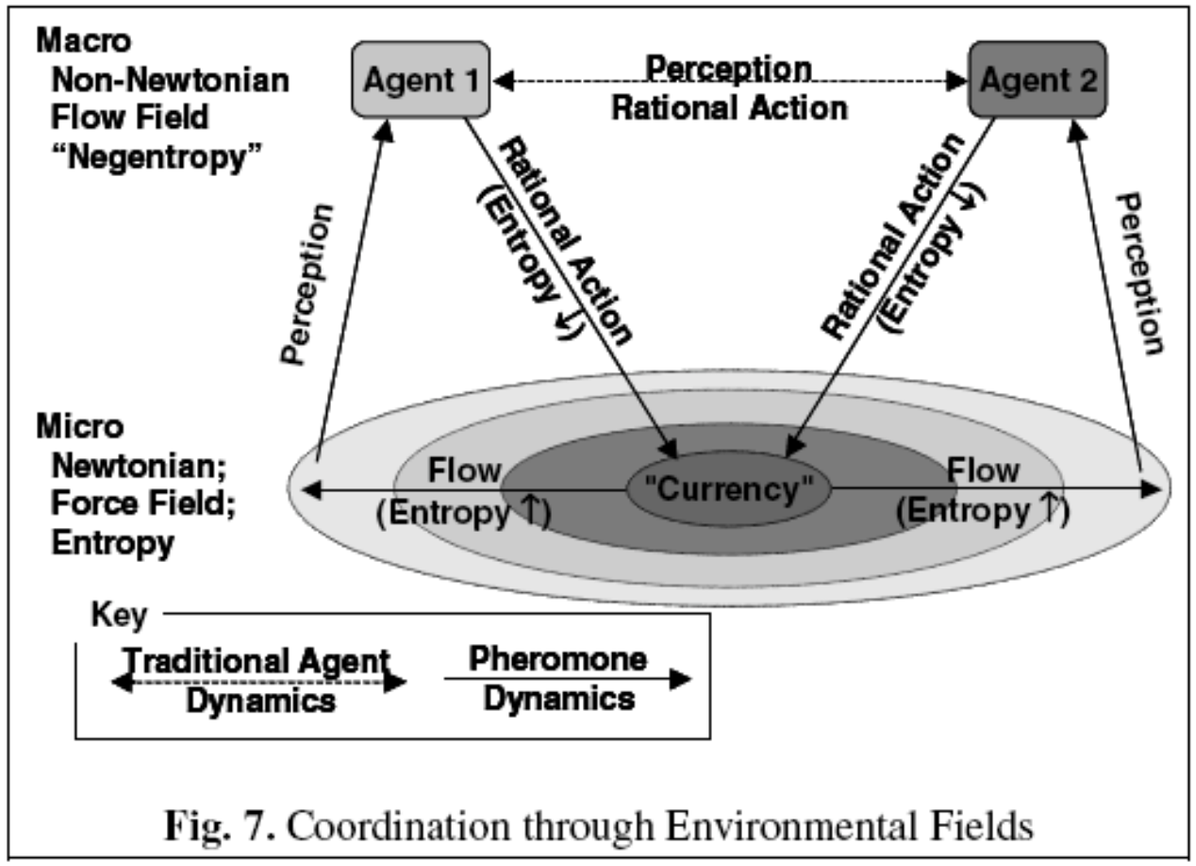
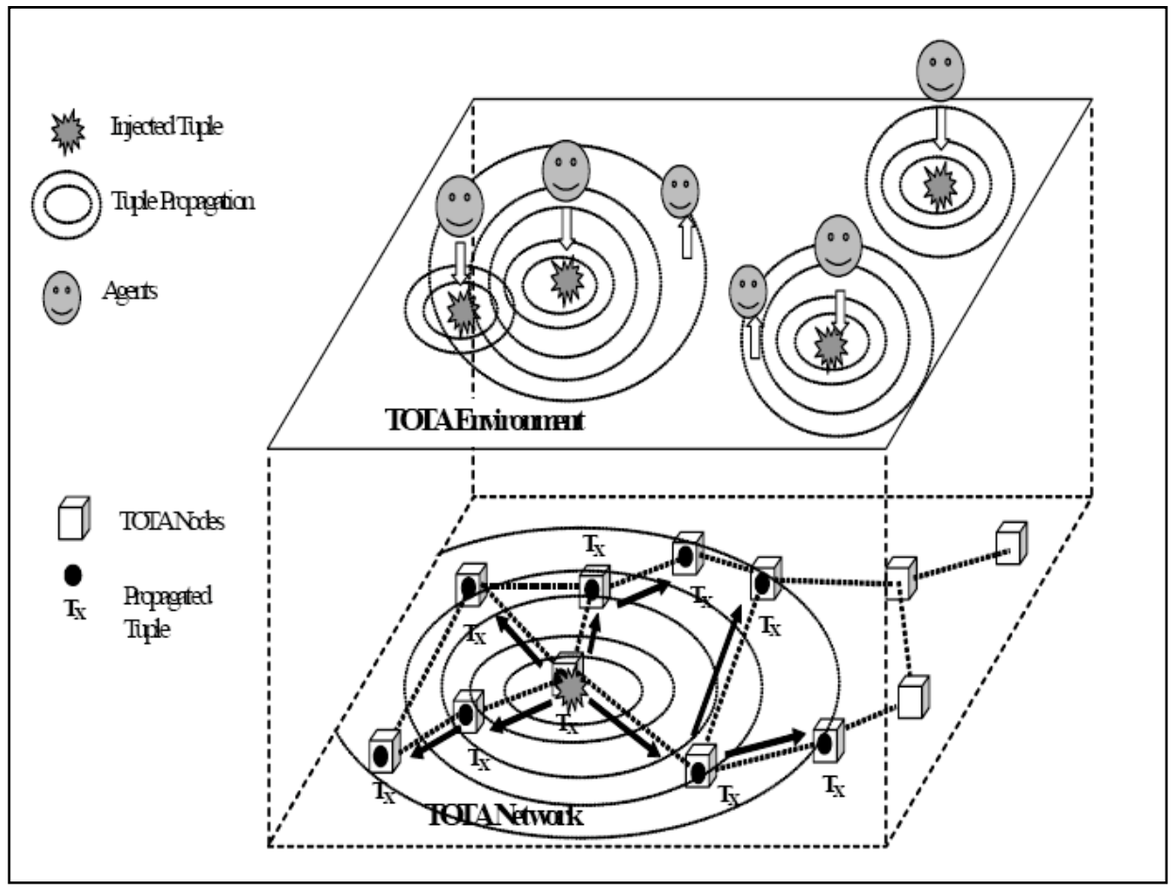


Fig. 7. Coordination through Environmental Fields

(from [Parunak, Brueckner, Sauter - 2005])

STIGMERGIC APPROACHES: TOTA

- TOTA (Tuple-On-The-Air) [Mamei, Zambonelli - 2005]
  - tuple-based middleware for implementing different kind of stigmergic-based coordination activities
  - distributed environment composed by a network of nodes managing the insertion, retrieval and diffusion/evaporation/propagation
    - tuple contains rules “programming” their propagation/evaporation/etc.



FIELD-BASED ENVIRONMENTS

- Computational fields [Mamei, Zambonelli - 2006]
  - agents move and coordinate their behaviour by following *abstract force fields* spread in the environment (by agents or the environment itself)
  - environment dynamics and agents movements induce changes in the fields, realizing the feedback cycle
  - implemented upon TOTA middleware

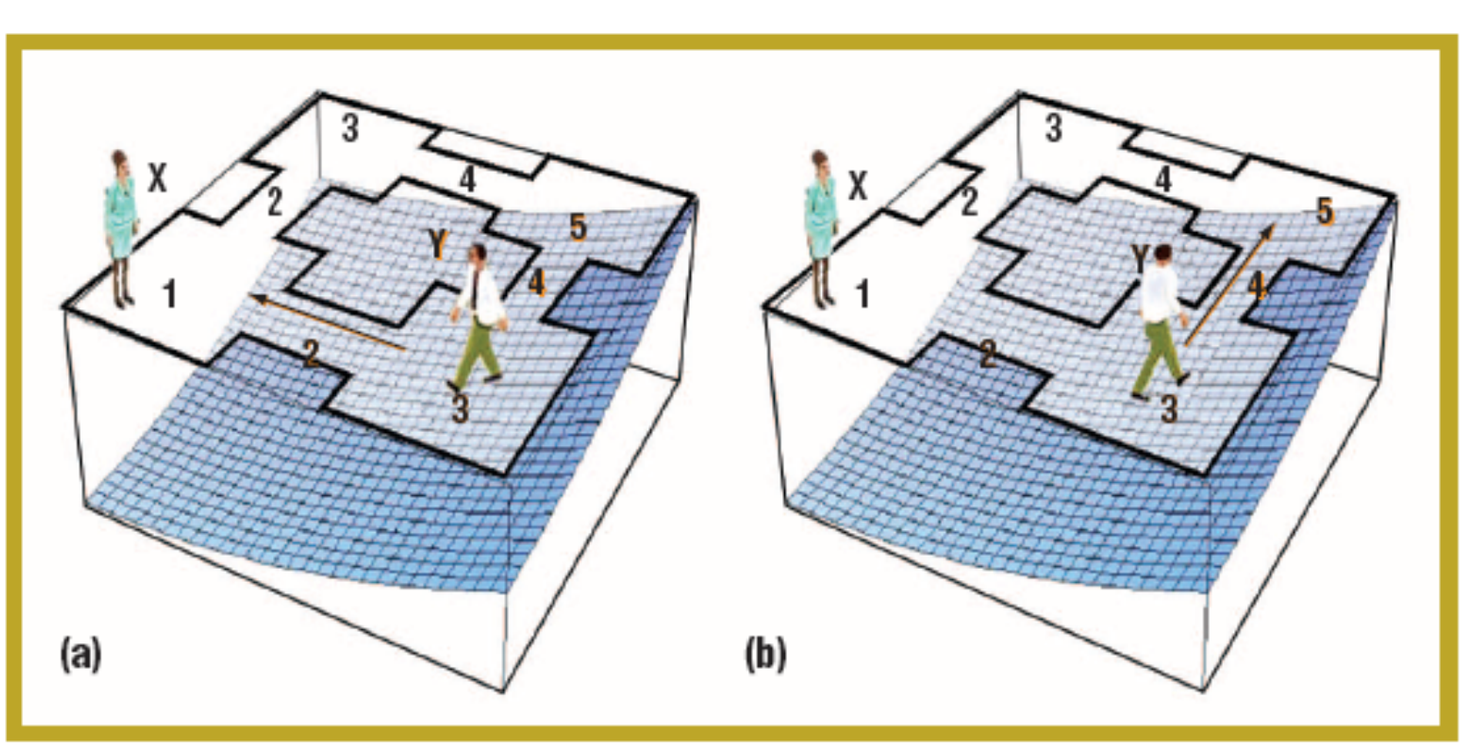


Figure 1. Agent X propagates a presence field whose value has a minimum where Agent X is located. Agent Y senses that presence field to (a) approach Agent X by following the gradient downhill and (b) move farther from Agent X by following the presence field's gradient uphill.

(figure from [Mamei, Zambonelli, Leonardi - 2004])

TAG-BASED INTERACTIONS

- Environment supporting *tag interactions* to enable forms of *observation-based* coordination in software MAS [Platon, Sabouret, Honiden - 2006]
  - interactions based on a (soft) body instead of speech acts, where agents can attach a set of *tags* that can be observed by other agents
  - opportunistic tag interactions occur when one agent receives information about others without requesting it

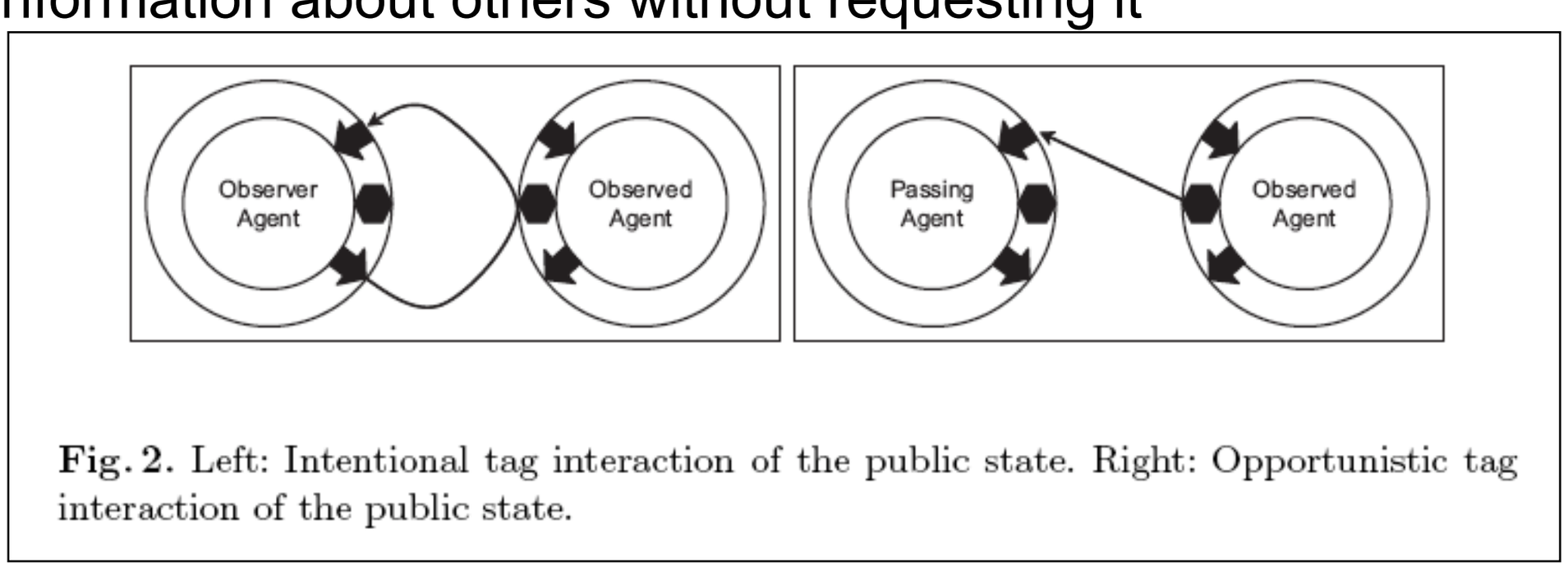
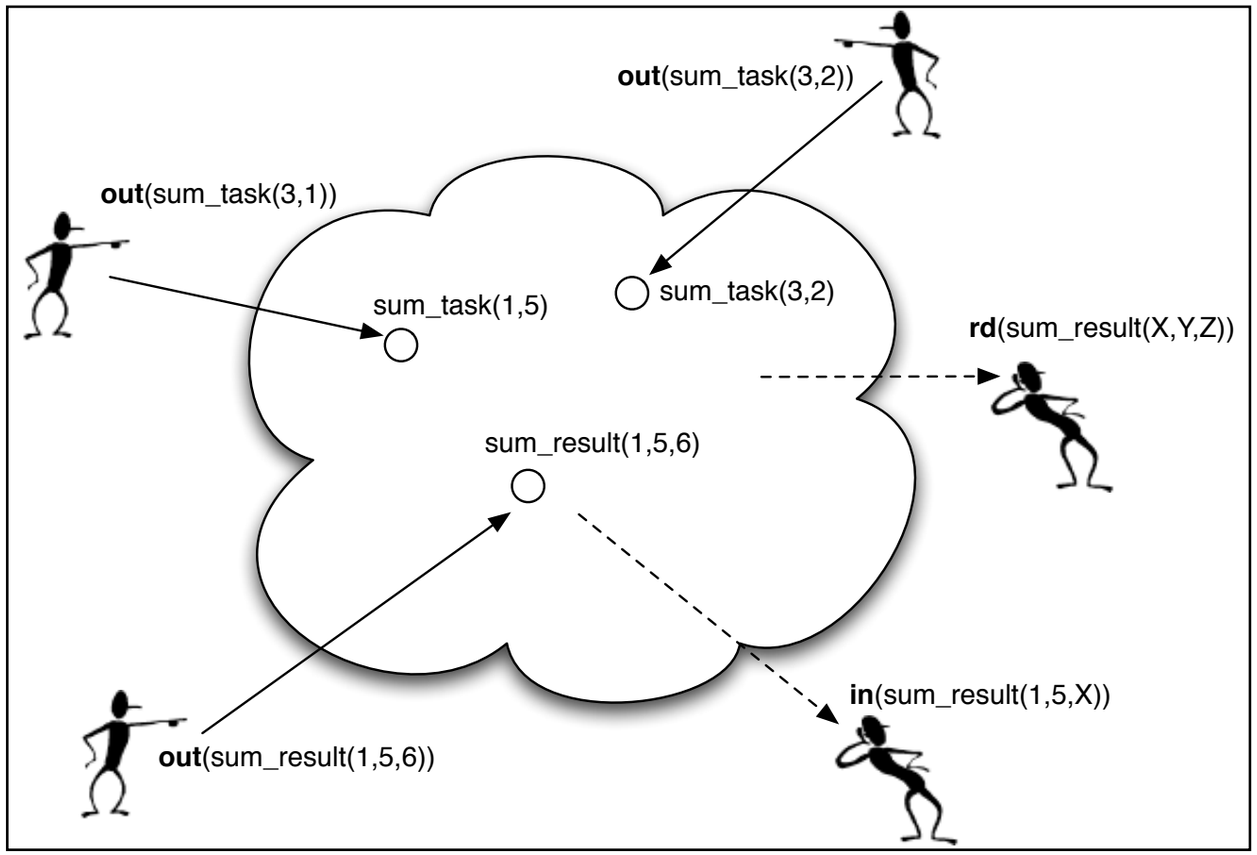


Fig. 2. Left: Intentional tag interaction of the public state. Right: Opportunistic tag interaction of the public state.

(Figure from [Platon, Sabouret, Honiden - 2006])

SPACE-BASED COORDINATION

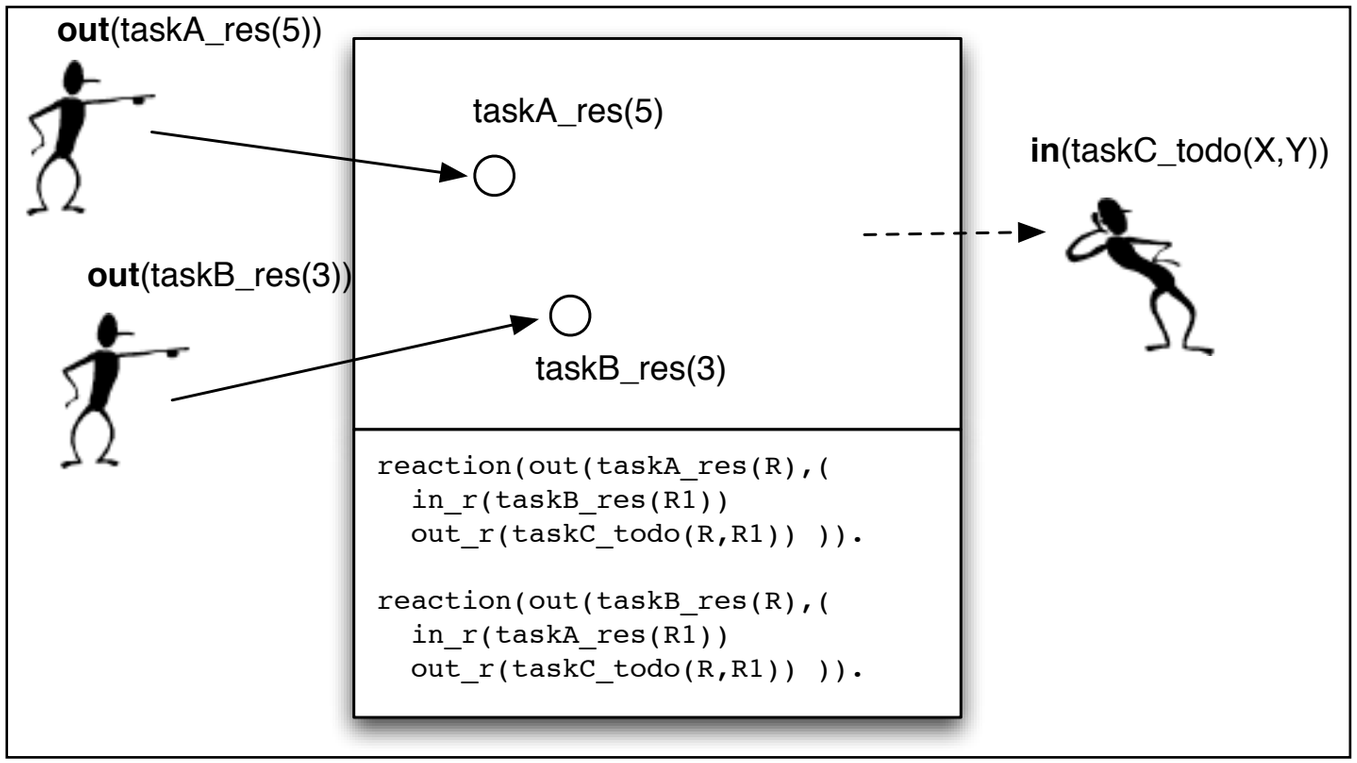
- Coordination media are *tuple spaces* where agent associatively retrieve, read and insert *tuples* [Gelernter, 1985]
  - Linda coordination language
    - coordination primitives providing basic synch. behaviour: in, out, rd



- Many extensions, several oriented to MAS and mobile agents
  - e.g. TuCSoN [Omicini, Zambonelli - 1999], LIME [Murphy, Picco, Roman - 2001]

PROGRAMMABLE SPACES

- Coordination models allowing to shape the interaction space by means of rules programming the coordination medium behaviour
  - an example: Tuple Centres and ReSpecT language [Omicini, Denti - 2001]
    - programmable tuple spaces
      - by means of reactions (specified in ReSpecT) encapsulating the coordination laws (functionalities) to be enforced by the tuple centre

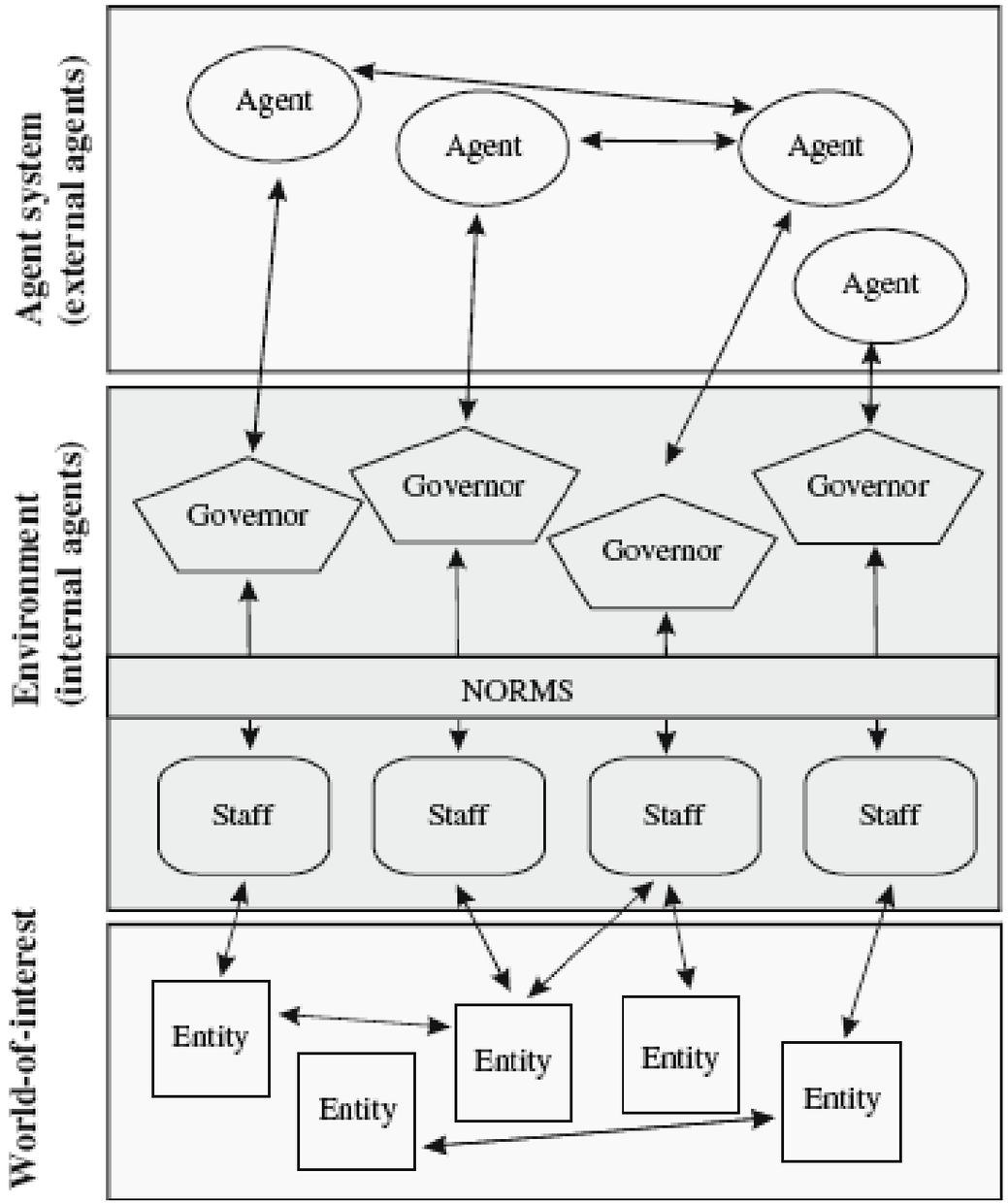


- adopted in TuCSoN's infrastructure
  - tuple centres collected/distributed in TuCSoN nodes

E-INSTITUTIONS

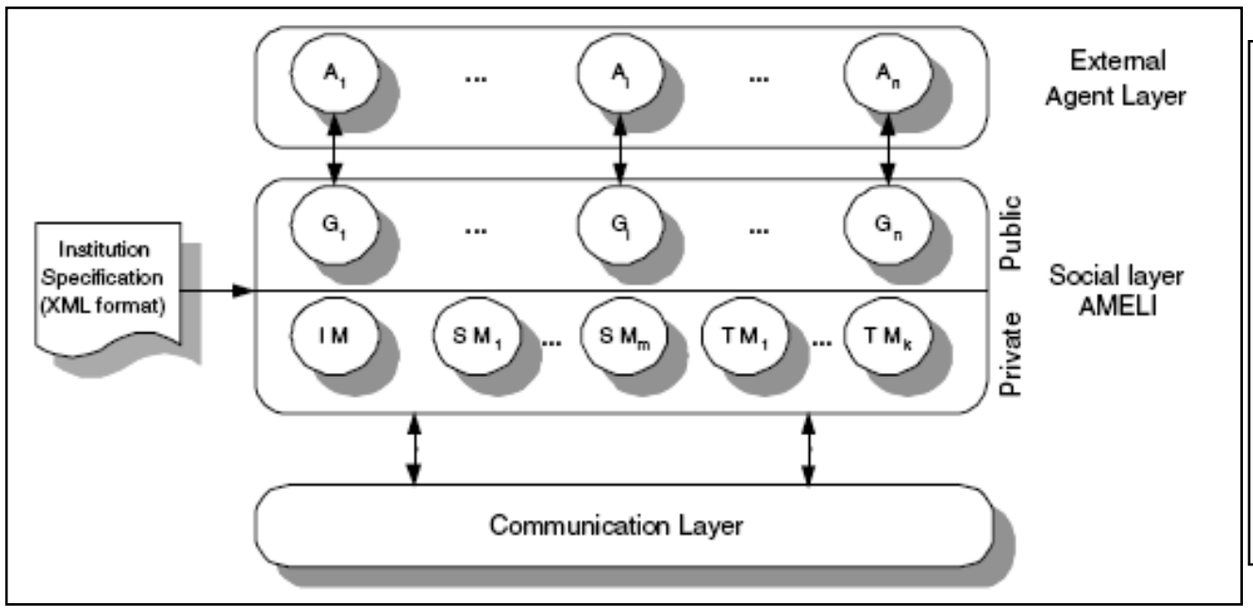
- Environments for articulating and regulating agent interaction in open MAS [Noriega, 1997]

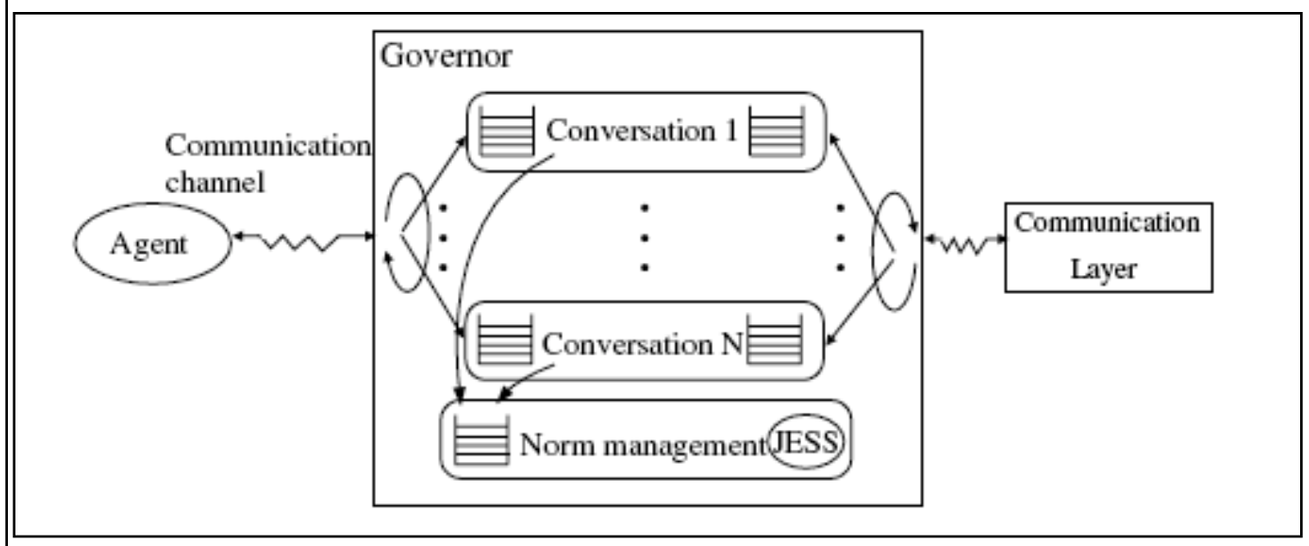
- The E-Institution is part of the environment
  - realized by a collection of so-called **staff agents**
- The application's agent system consists of so-called *external agents*
  - external to the institution
- External agents interact through the environment via a set of specialized agents called **governor**
  - one governor for each agent
- **Norms** are used to structures and regulate interactions in the environment
  - *social laws* enacted by the coordinated actions of governors and staff agents



E-INSTITUTIONS MIDDLEWARE

- AMELI middleware [Esteva et al., 2004]
  - concrete architecture for e-Institutions
  - social layer on top of the communication layer
    - staff agents implemented by scene managers, institution managers, transition managers





(Figures taken from [Esteva et al., 2004])

A NEW PERSPECTIVE ON THE ROLE OF THE ENVIRONMENT

- Environment as *a first-class abstraction* in MAS
  - considering environment as *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 both supports separation of concerns in MAS
  - improving the engineering practice

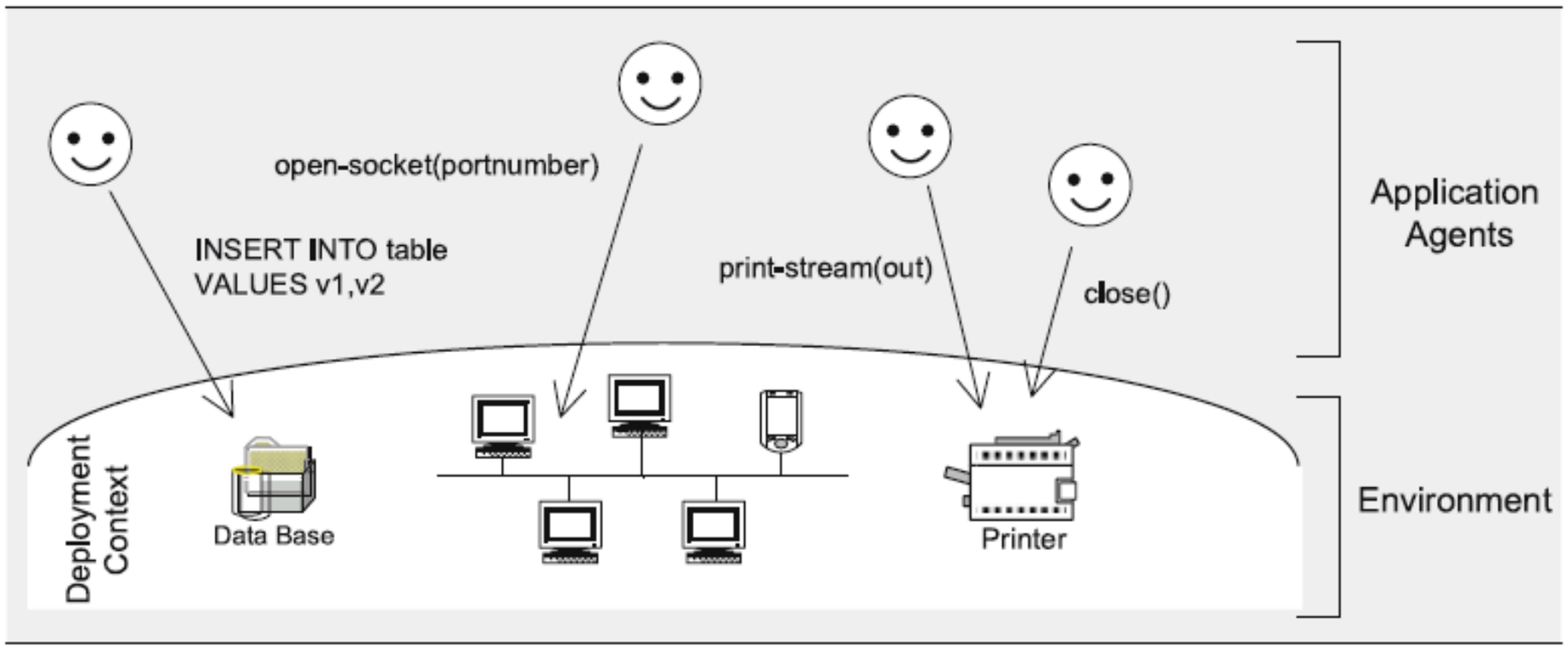
BASIC SUPPORT LEVELS

[Weyns, Omicini, Odell - 2007]

- Three levels
  - basic level
  - abstraction level
  - interaction-mediation level

BASIC LEVEL

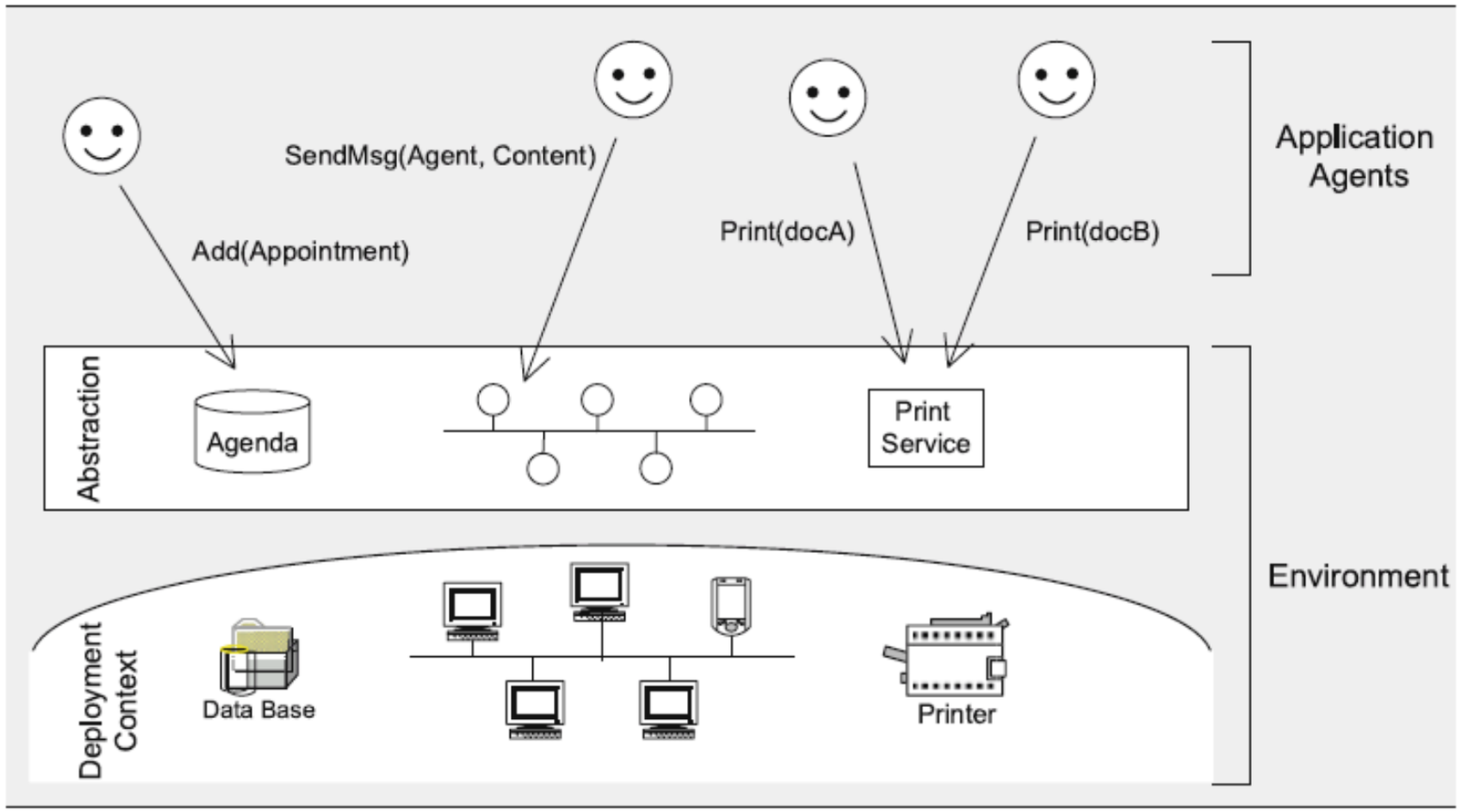
- The environment enables agents to access the *deployment context*
  - i.e. the hardware and software and external resources with which the MAS interacts
    - sensors and actuators, a printer, a network,a database, a Web service, etc.



(Figure from [Weyns, Omicini, Odell - 2007])

ABSTRACTION LEVEL

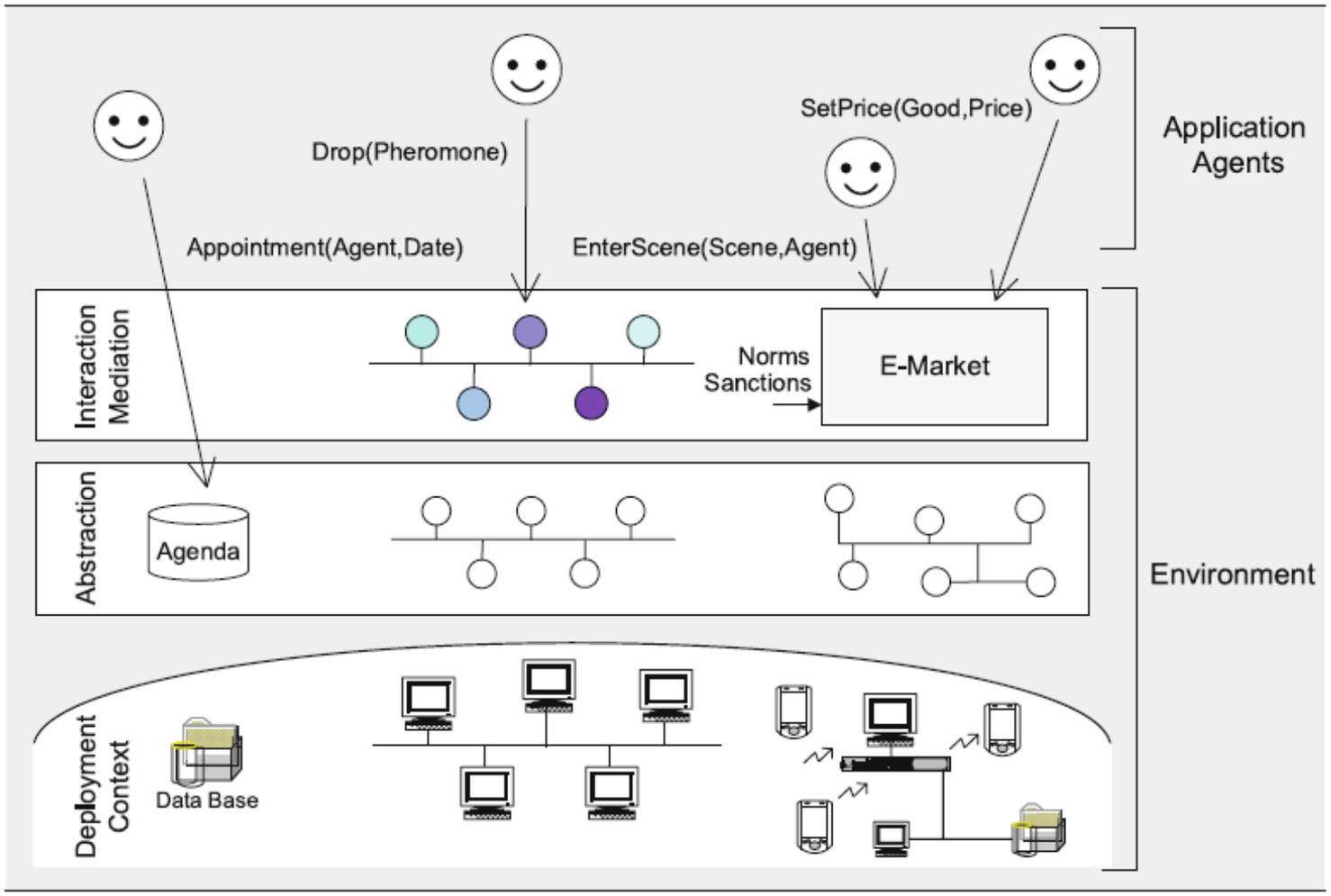
- Bridges the conceptual gap between the agent abstraction and low-level details of the deployment context
  - shields low-level details of the deployment context



(Figure from [Weyns, Omicini, Odell - 2007])

INTERACTION-MEDIATION LEVEL

- Regulate the access to shared resources
- Mediate interaction between agents



(Figure from [Weyns, Omicini, Odell - 2007])

ENVIRONMENT DEFINITION REVISED

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

[Weyns, Omicini, Odell - 2007]

HIGHLIGHTS

- *First-class abstraction*
  - environment as an independent building block in the MAS
  - encapsulating its own clear-cut responsibilities, irrespective of the agents
- *The environment provides the surrounding conditions for agents to exist*
  - environment as an essential part of every MAS
  - the part of the world with which the agents interact, in which the effects of the agents will be observed and evaluated

HIGHLIGHTS

- *Environment as a glue*
  - on their own, agents are just individual loci of control.
  - to build a useful system out of individual agents, agents must be able to interact
  - the environment provides the glue that connects agents into a working system
- *the environment mediates both the interaction among agents and the access to resources*
  - the environment can be an active entity with specific responsibilities in the MAS
  - it provides a medium for sharing information and mediating coordination among agents
    - as a mediator, the environment not only enables interaction, it also constrains it
    - as such, the environment provides a design space that can be exploited by the designer

RESPONSIBILITIES

- Structuring the MAS
  - the environment is first of all a shared “space” for the agents, resources, and services which structures the whole system
  - different forms of structuring can be distinguished
    - physical structure
      - refers to spatial structure, topology, and possibly distribution, see e.g.,
    - communication structure
      - refers to infrastructure for message transfer, infrastructure for stigmergy, or support for implicit communication
    - social structure
      - refers to the organizational structure of the environment in terms of roles, groups, societies

RESPONSIBILITIES

- Embedding resources and services
  - resources and services can be situated either in the physical structure or in the abstraction layer introduced by the environment
  - the environment should provide support at the abstraction level shielding low-level details of resources and services to the agents
- Encapsulating a state and processes
  - besides the activity of the agents, the environment can have processes of its own, independent of agents
    - example: evaporation, aggregation, and diffusion of digital pheromones
  - It may also provide support for maintaining agent-related state
    - for example, the normative state of an electronic institution or tags for reputation mechanisms

RESPONSIBILITIES

- *Ruling* and *mediating* function
  - the environment can define different types of rules on all entities in the MAS.
    - constraints imposed by the domain at hand or laws imposed by the designer
    - may restrict the access of specific resources or services to particular types of agents, or determine the outcome of agent interactions.
  - preserving the agent system in a consistent state according to the properties and requirements of the application domain
- Examples
  - coordination infrastructures
  - e-Institutions

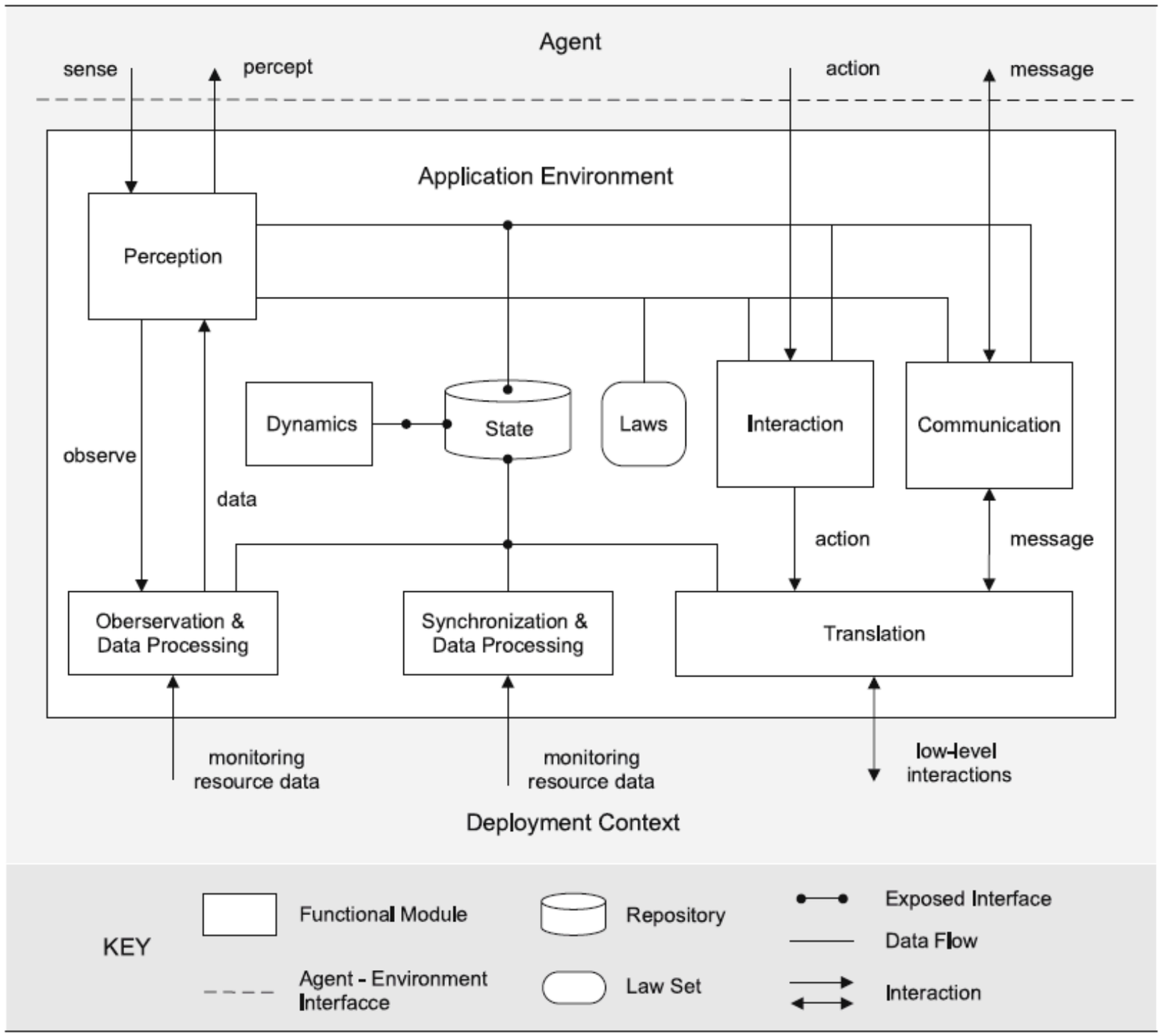
BASIC PROPERTIES: OBSERVABILITY

- Observability
  - providing mechanims to agents for inspecting the different structures of the environment
    - as well as the resources, services, and possibly external state of other agents
  - observation of a structure is typically limited to the current context in which the agent finds itself
    - spatial context, communication context, and social context
  - related to observability is the semantic description of the domain
    - which can be defined by an environment *ontology*
    - covering the different structures of the environment
      - as well as the observable characteristics of resources, services and agents, and possibly the regulating laws

BASIC PROPERTIES: ACCESSIBILITY

- Accessibility
  - agents must be able to access the different structures of the environment
    - as well as its resources, services, and possibly external state of other agents
  - as for observability, accessing a structure is limited to the *current context* in which the agent finds itself
  - in general, resources can be perceived, modified, generated, or consumed by agents
    - services on the other hand provide functionality to the agents on their request
    - the extent to which agents are able to access a particular resource or service may depend on several factors such as the nature of the resource or service, the capabilities of the agent, and the (current) interrelationships with other resources, services, or agents

A REFERENCE ABSTRACT ARCHITECTURE



(Figure from [Weyns, Omicini, Odell - 2007])

ENVIRONMENT AS AN ABSTRACTION LAYER: EXTENDING THE VIEW

- Back to the *abstraction* level support
  - providing abstractions to bridge the gap to the deployment context
- Generalisation
  - proving abstractions to model as first-order entities of the agents world any kind of *resource* and *tools* specifically designed to be exploited by agents to support their individual and collective work
  - first-class design abstraction for sw engineers + first-order runtime abstractions for agents

GIVING AGENTS
THEIR FIRST-CLASS WORLD

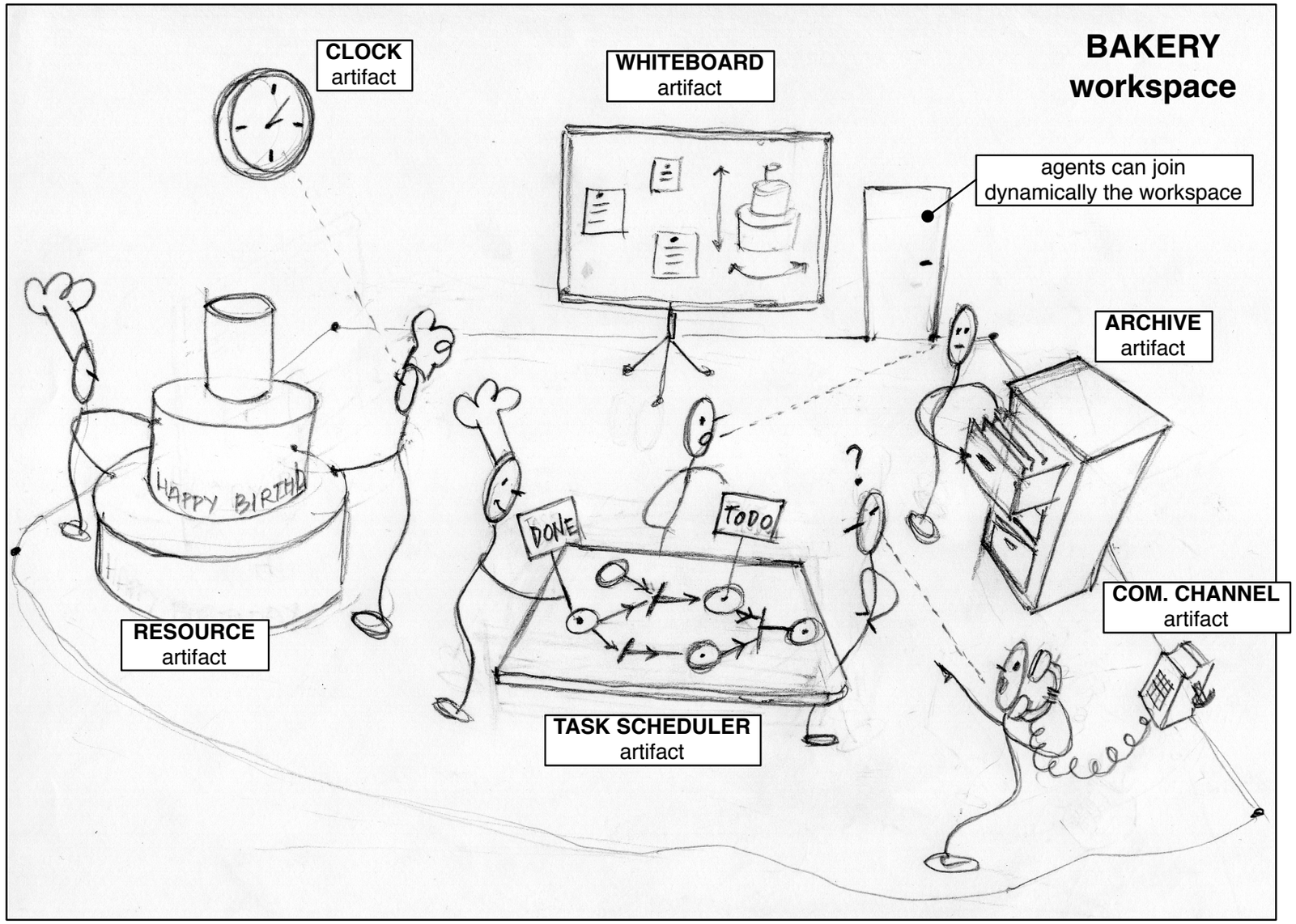
- Environment as a shared computational world that agents can *use* but also *construct / adapt / extend* for their individual and collective purposes
  - beyond pure observability and accessibility
  - runtime perspective (vs. design time)
- Opening new perspectives
  - in particular for MAS composed by cognitive agents

DESIGNING AND PROGRAMMING AGENTS' WORLD

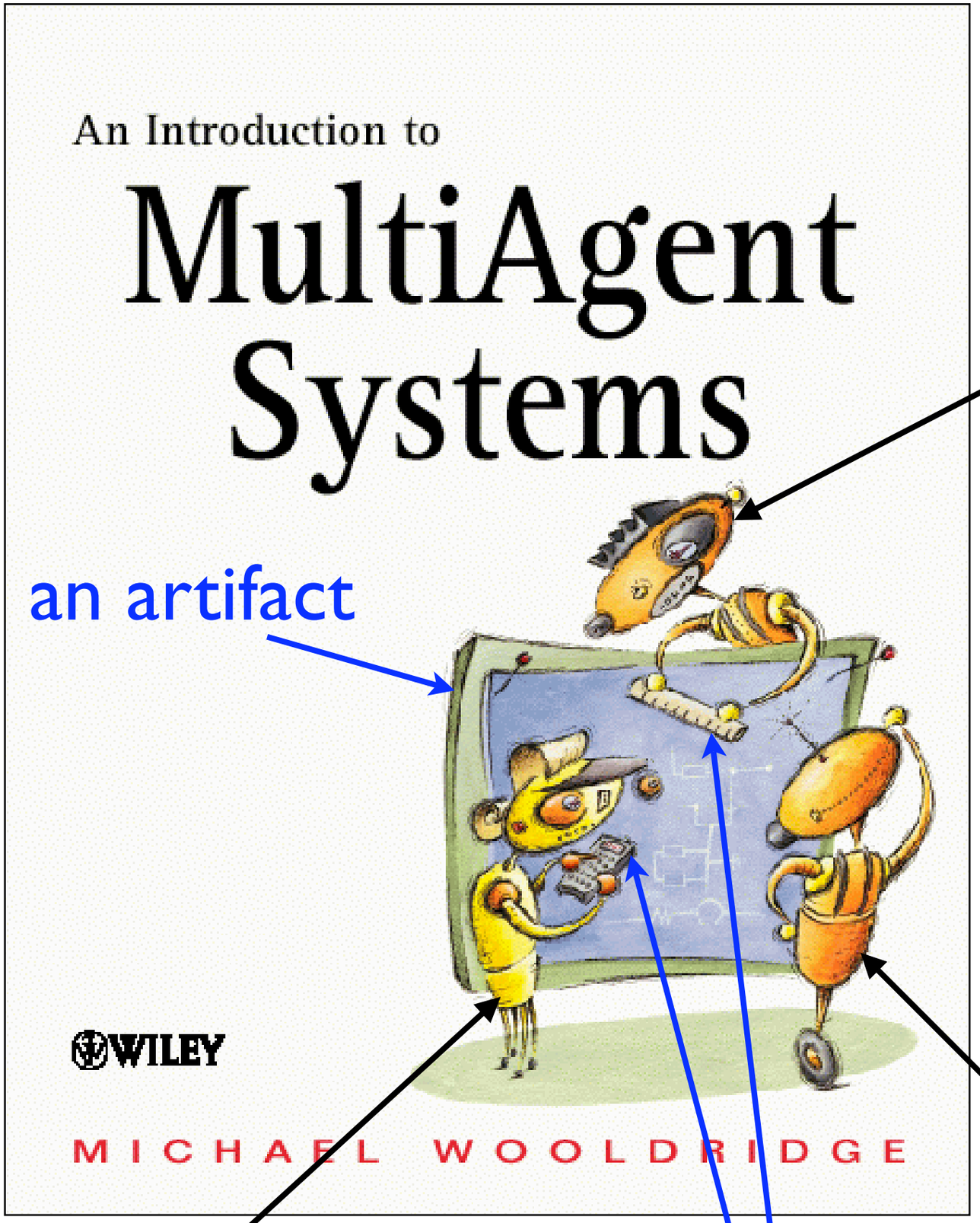
- Looking for general-purpose approaches for conceiving, designing, programming, executing the environment as agents' world
  - orthogonality
  - generality
  - expressiveness
- Uniformly integrating different MAS aspects
  - coordination, organisation, institutions
- Concrete models and technologies: examples
  - AGRE / AGREEN / MASQ [Ferber, Michel, Baez - 2005] [Báez-Barranco,Stratulat, Ferber - 2007]
  - A&A and CArtAgO [Ricci, Viroli, Omicini - 2007][Omicini, Ricci, Viroli - 2008]
  - GOLEM [Bromuri, Stathis - 2008]

AGENTS & ARTIFACTS (A&A)

- Agents' worlds as human cooperative environments
  - workspaces containing *artifacts* that agents create, share, use, adapt for their work [Ricci, Viroli, Omicini - 2007][Omicini, Ricci, Viroli - 2008]



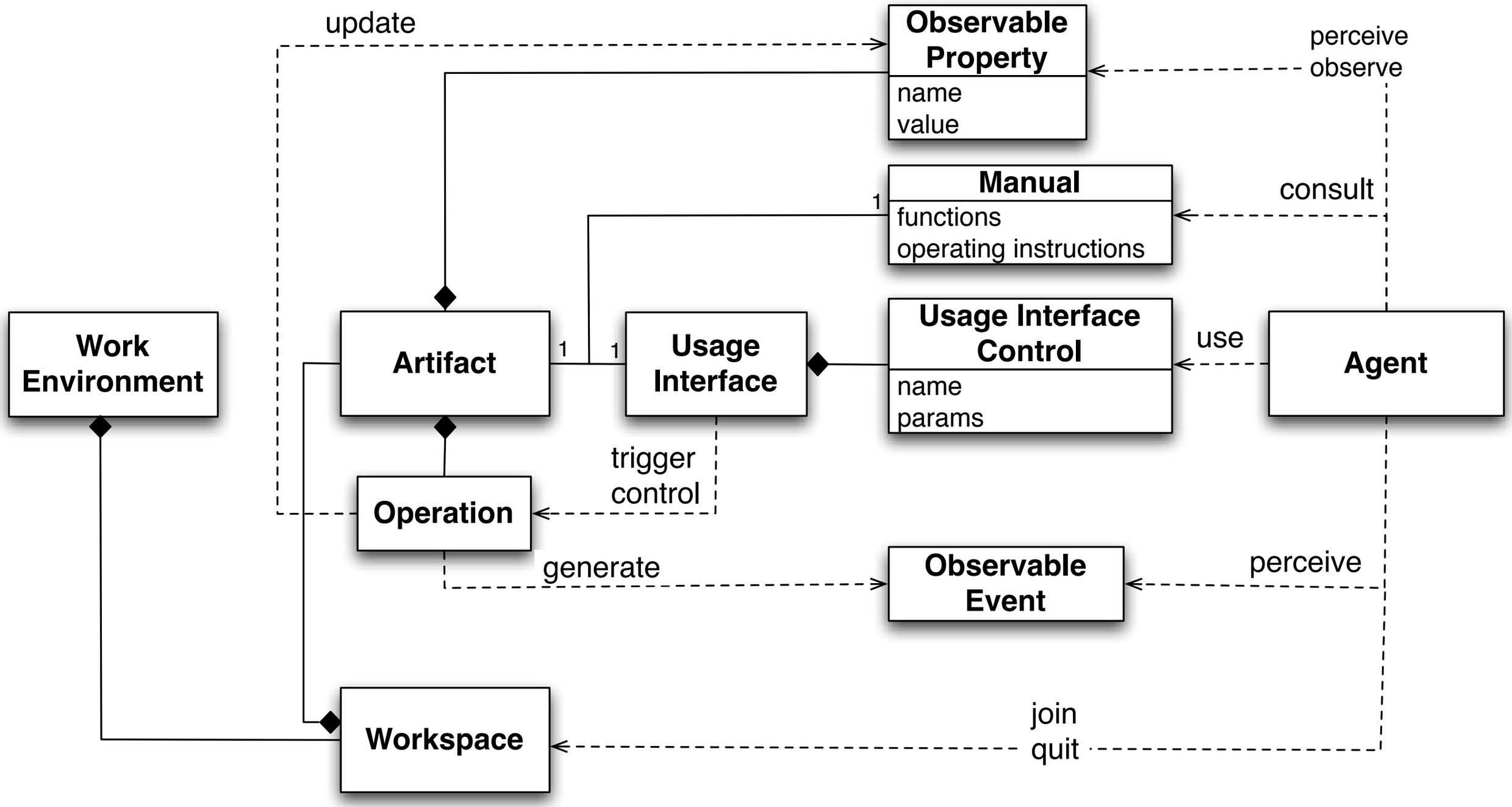
- Main inspiration from Activity Theory [Ricci, Omicini, Denti - 2003]


An Introduction to
MultiAgent Systems
MICHAEL WOOLDRIDGE
WILEY
an artifact
agent
agent
artifacts
agent

A&A BASIC CONCEPTS

- **Agents**
  - autonomous, goal/task-oriented entities
    - pro-activity, reactivity, social abilities, etc.
  - create and co-use artifacts for supporting their activities
    - besides direct communication
- **Artifacts**
  - *non-autonomous*, *function*-oriented entities
    - controllable and observable
  - modelling the *resources* and *tools* used by agents
    - designed by MAS programmers
    - first-class entities for agents
- **Workspaces**
  - grouping agents & artifacts
  - defining the topology of the computational environment

A&A META-MODEL



COORDINATION ARTIFACTS

[Omicini, Ricci, Viroli, Castelfranchi, Tummolini - 2004]

- Artifacts designed and programmed to provide coordination functionalities for agents
  - enabling and mediating agent interaction
    - mediated interaction
  - encapsulating coordination laws
    - constructing and constraining the interaction space
- Enabling the design and programming of MAS interaction
  - by shaping the tools that agents use to interact and coordinate
- Environment-based coordination
  - complimentary approach to coordination based on direct interaction
  - generality
    - implementing *any* kind of coordination medium or mechanism for MAS

WRAP-UP

- Environment as first-class abstraction
  - suitable locus where sw engineers can encapsulate functionalities of the MAS, besides agents
    - related to communication, coordination, organisation, security
- Environment as an abstraction layer for modelling and engineering agents' world
  - composed by environmental abstractions (e.g. artifacts) that agents perceive as first-order entities of their world
    - resources and tools that agents observe, use, adapt, manipulate

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

WOA 2009 MINI-SCUOLA

ENVIRONMENT PROGRAMMING IN MULTI-AGENT SYSTEMS

PART THREE

ENVIRONMENT PROGRAMMING

MULTI-AGENT SYSTEM PROGRAMMING PERSPECTIVE

- Agent-Oriented Programing and MAS programming in a software engineering perspective
  - agents (and MAS) as a paradigm to design and program software systems
    - computer programming perspective
      - computational models, languages,...
    - software engineering perspective
      - architectures, methodologies, specification, verification,...
- Focus on the (programming) language level
  - integrating environment programming with agent programming (languages)

ENVIRONMENT PROGRAMMING

- Environment in the loop of MAS desing and programming
  - environment as first-class design & programming abstraction
    - software designers and engineers perspective
    - programming MAS = programming Agents + programming Environment
  - environment as first-class runtime abstraction for agents
    - agent perspective
    - to be observed, used, adapted, constructed, ...
- Defining computational and programming models also for the environment part

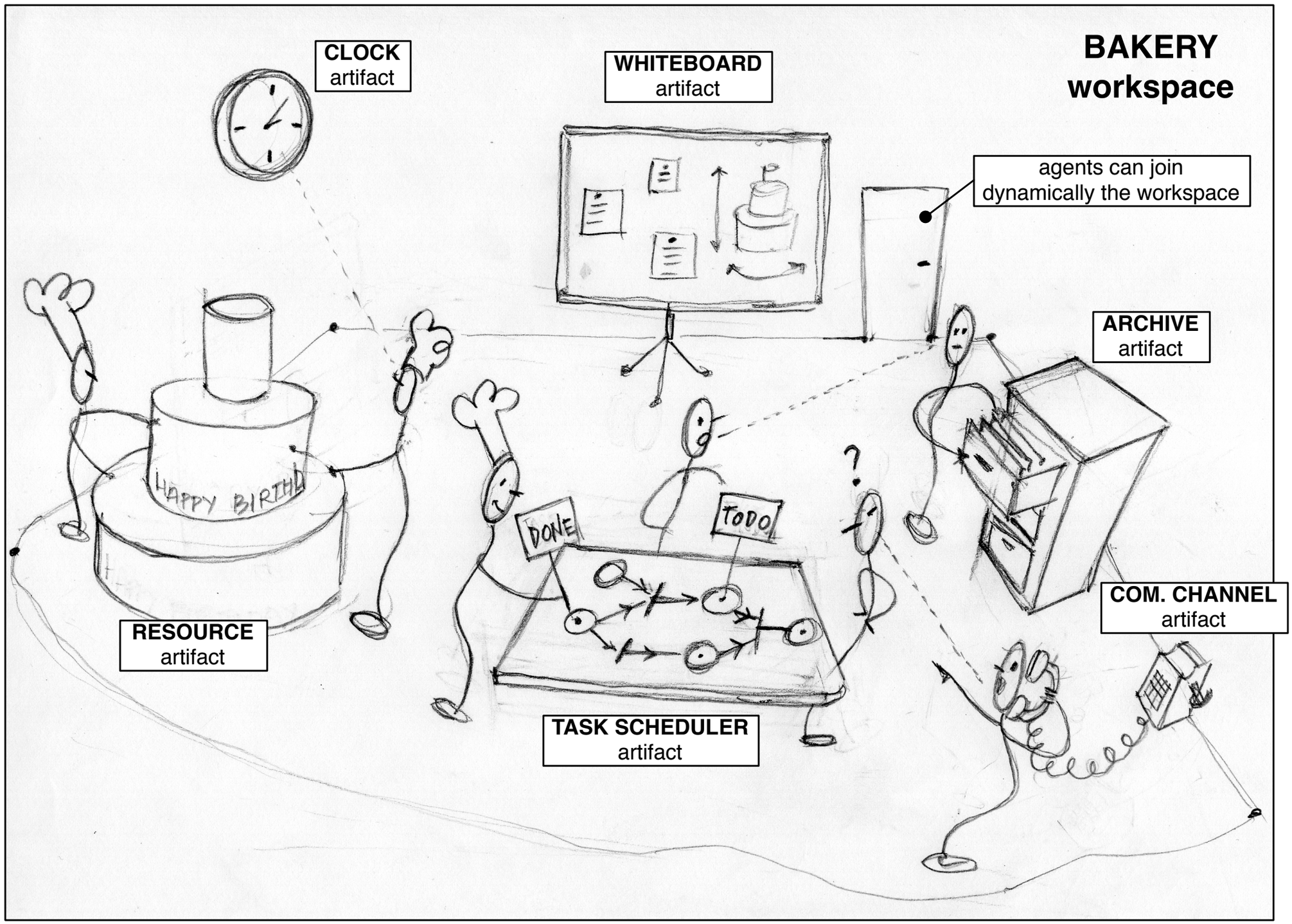
PROGRAMMING MODEL: DESIDERATA (1/2)

- **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

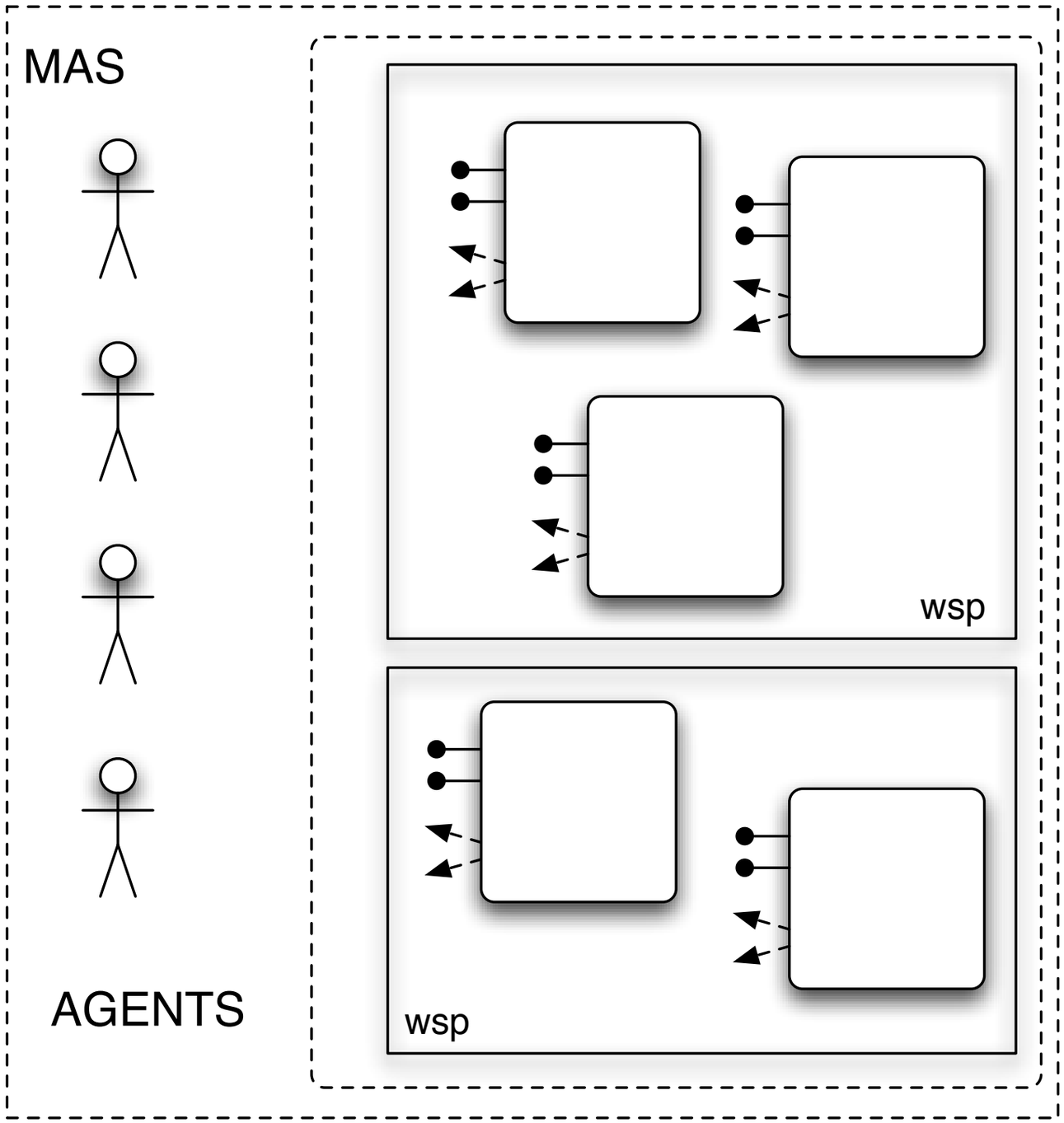
PROGRAMMING MODEL: DESIDERATA (2/2)

- **(Dynamic) extendibility**
  - dynamic construction, replacement, extension of environment parts
  - support for *open* systems
- **Reusability**
  - reuse of environment parts in different application contexts / domains

ARTIFACT-BASED PROGRAMMING MODEL (RECALL)

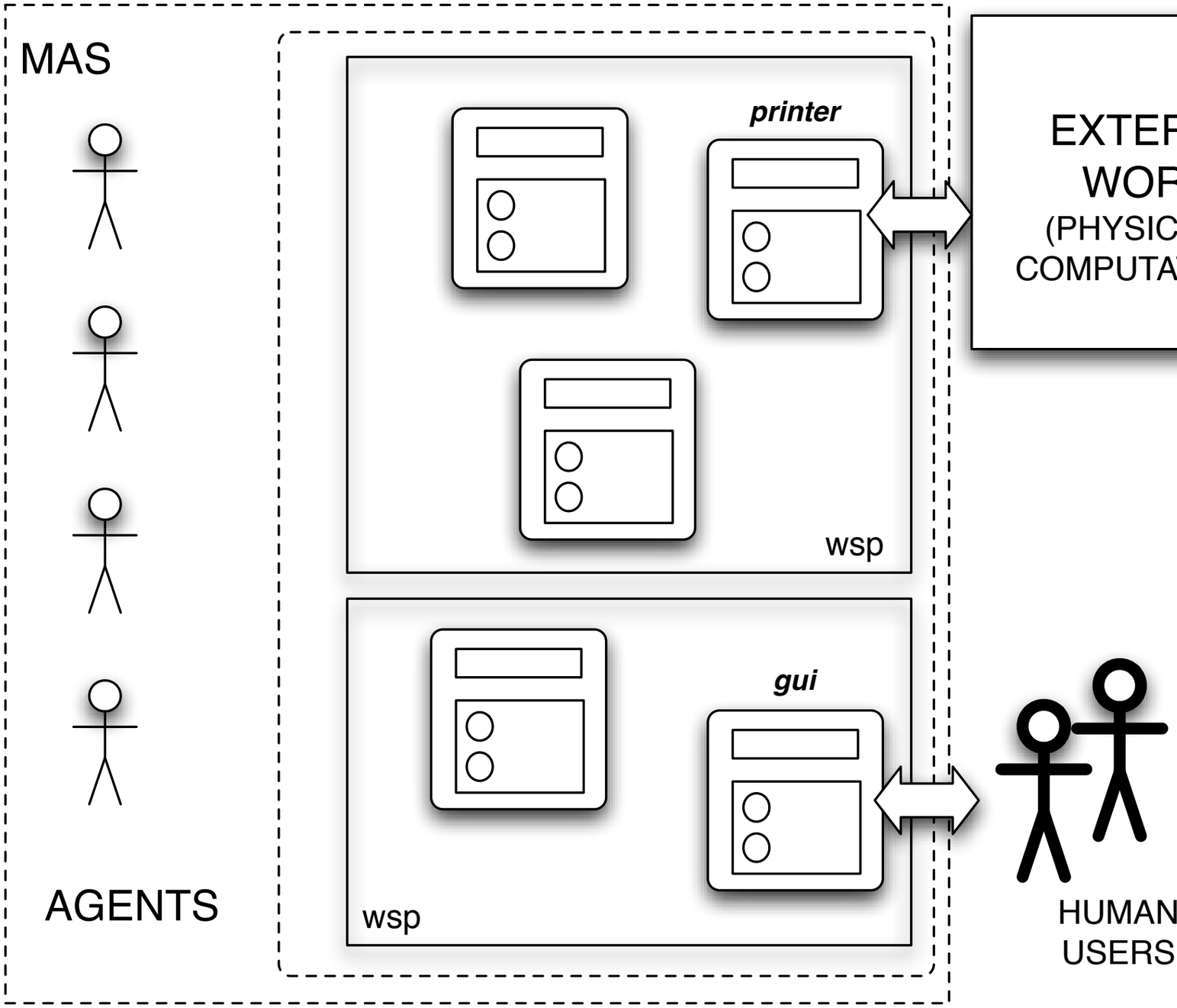


ABSTRACTION & MODULARIZATION



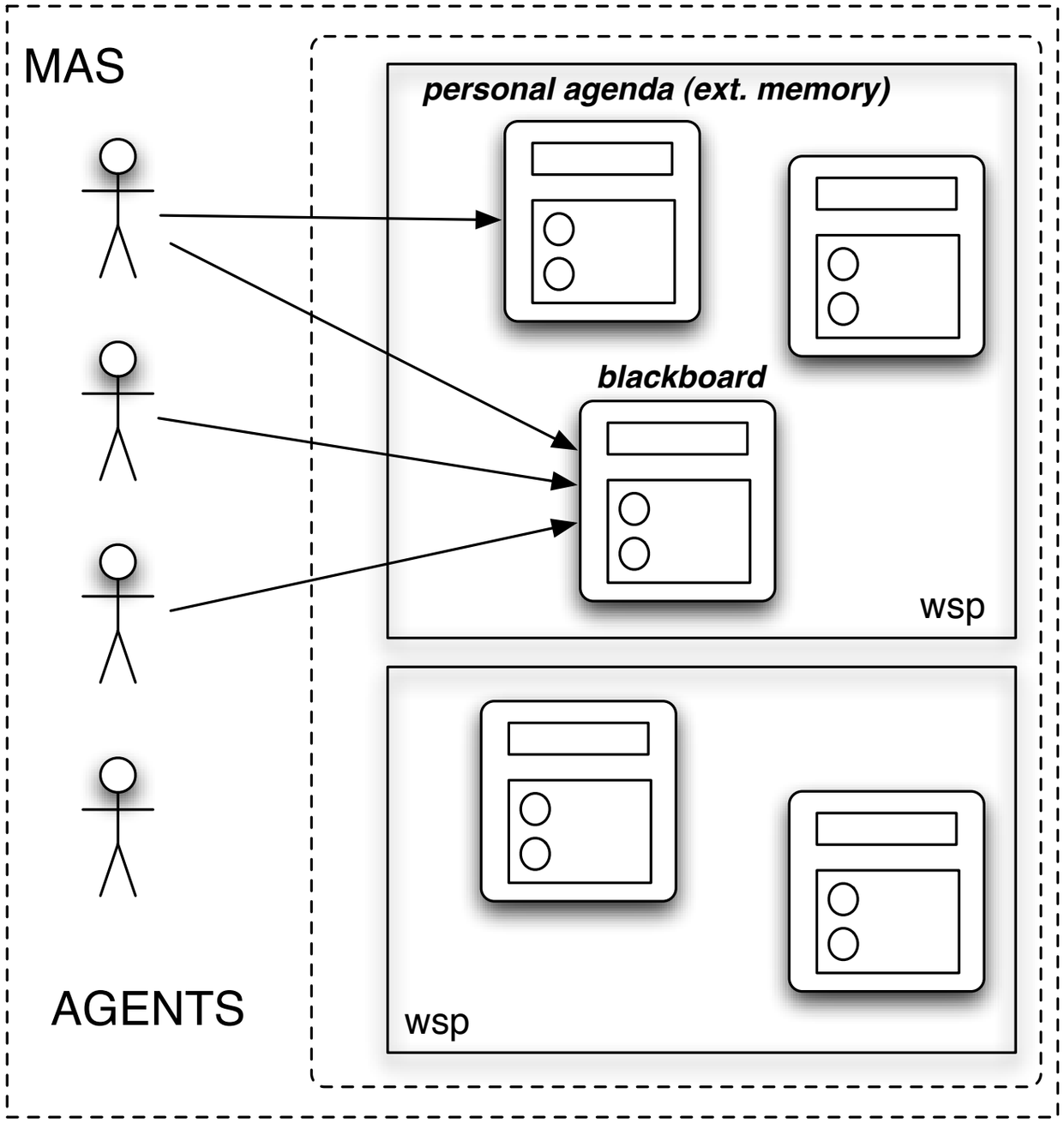
- Abstraction
  - artifacts as first-class resources and tools for agents
- Modularization
  - artifacts as dynamic reusable modules encapsulating functionalities, organized in workspaces
    - distribution

ABSTRACTION LAYER (1/2)



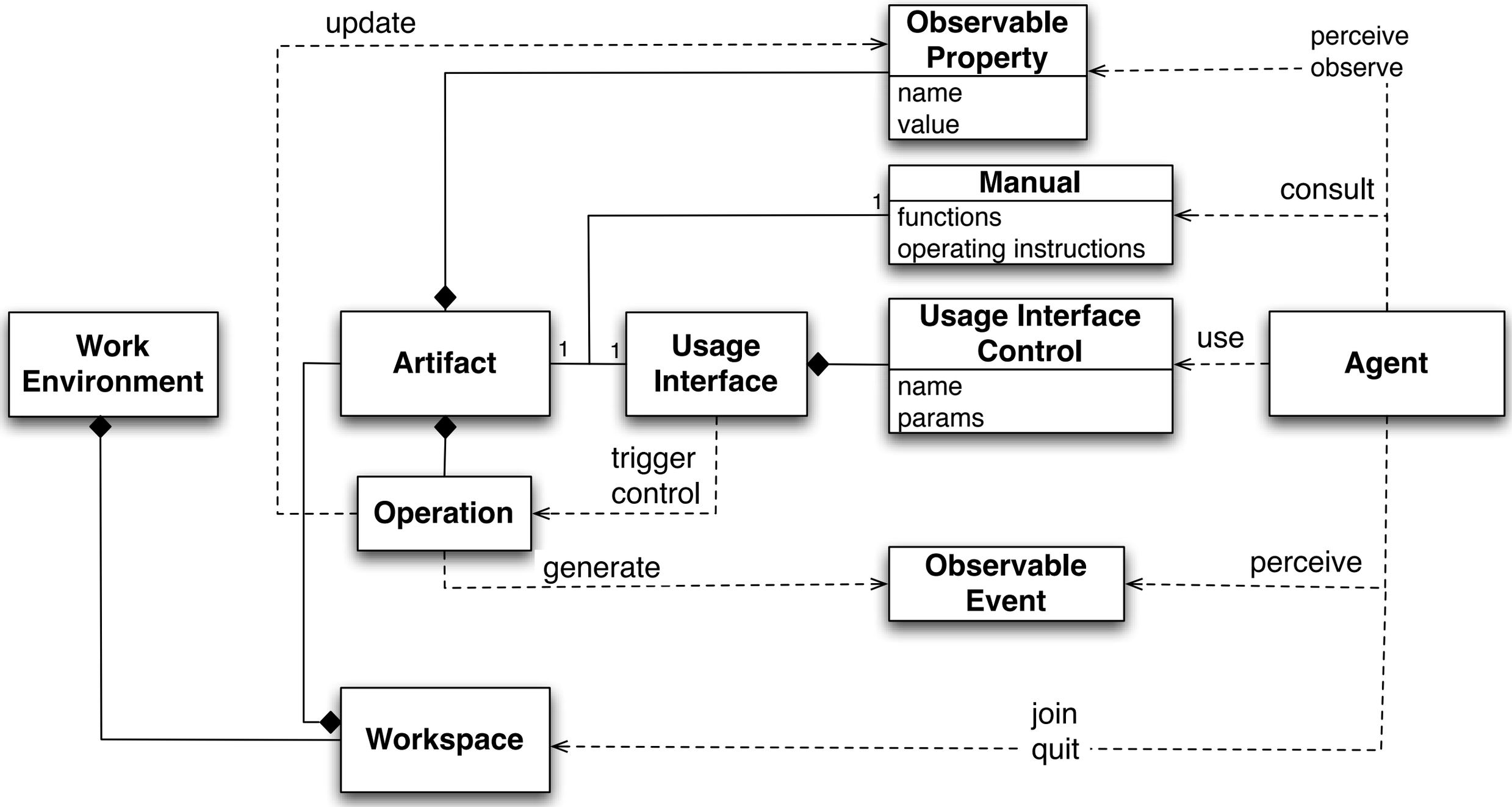
- Using artifacts to represent or interact with *external* resources / users

ABSTRACTION LAYER (2/2)

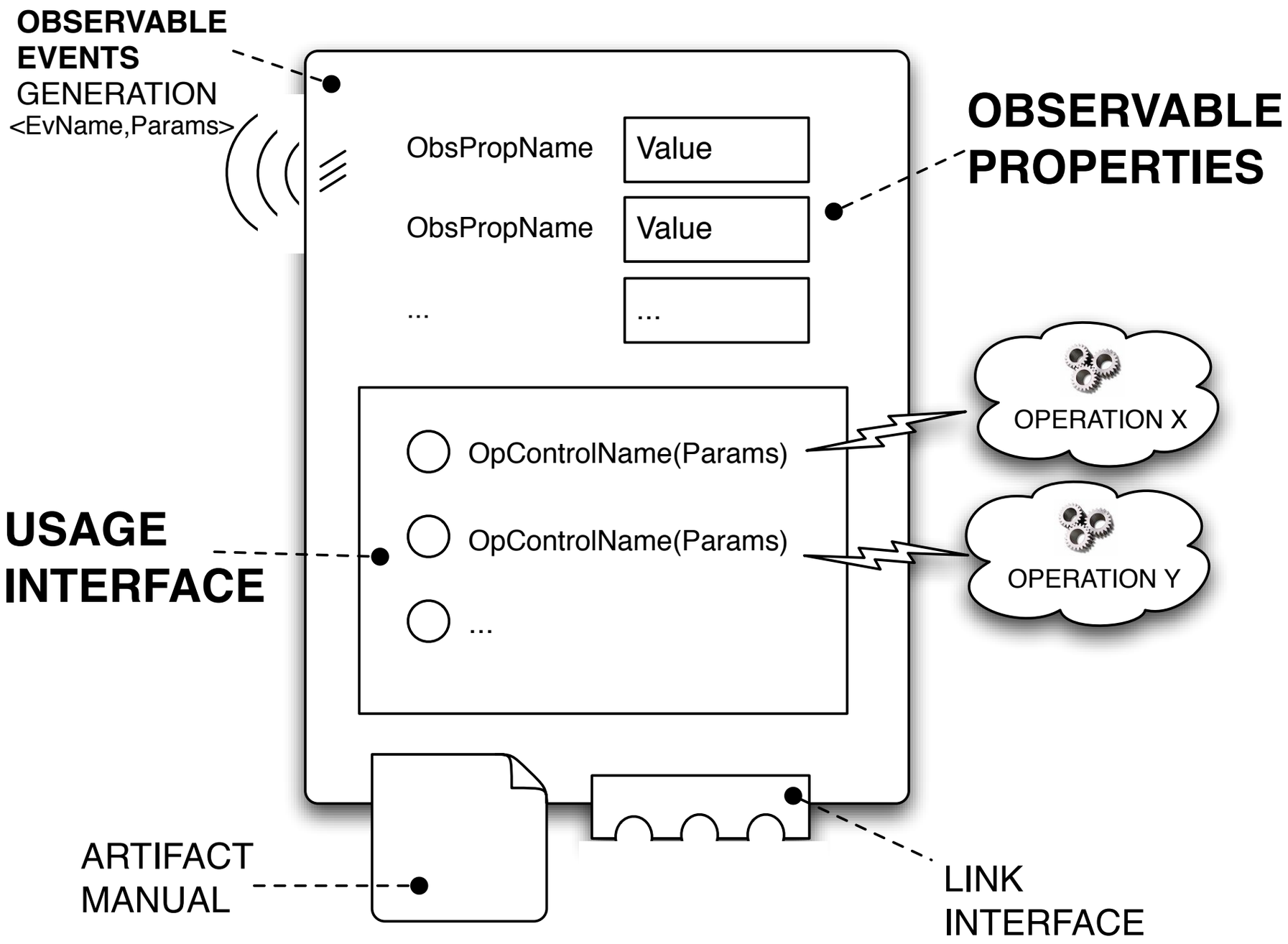


- Using artifacts to represent *internal* resources providing some kind of functionalities to agents
  - e.g. coordinating functionalities, such as blackboard
  - e.g. a personal agenda, enhancing agent capabilities

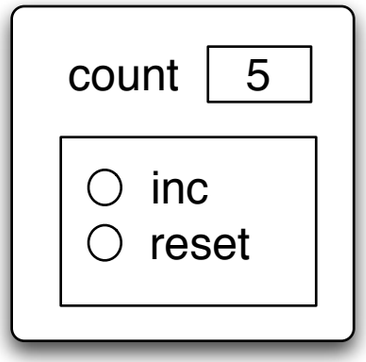
A&A META-MODEL OVERVIEW



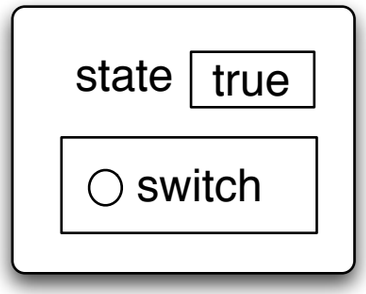
ARTIFACT MODEL
- “COFFEE MACHINE METAPHOR” -



EXAMPLES

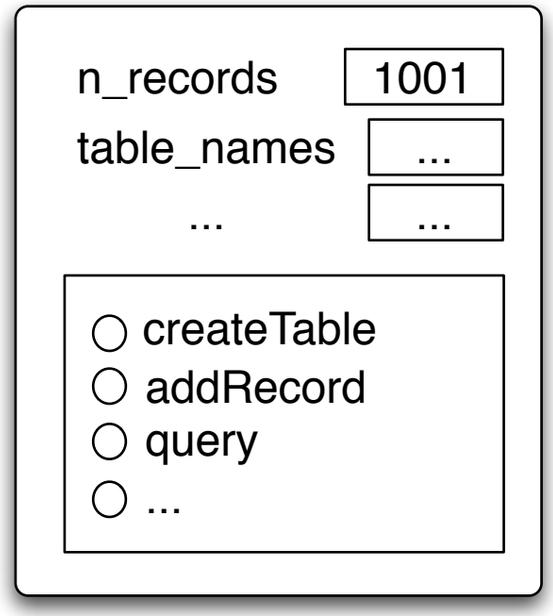


a counter

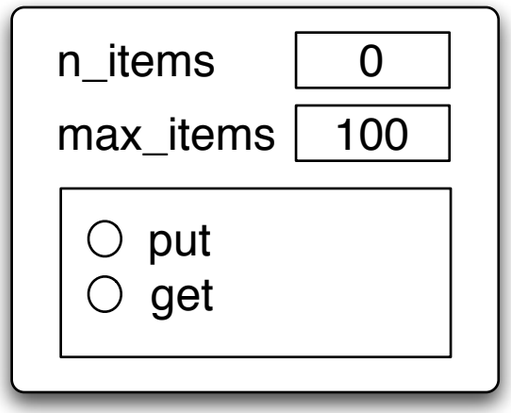


a flag

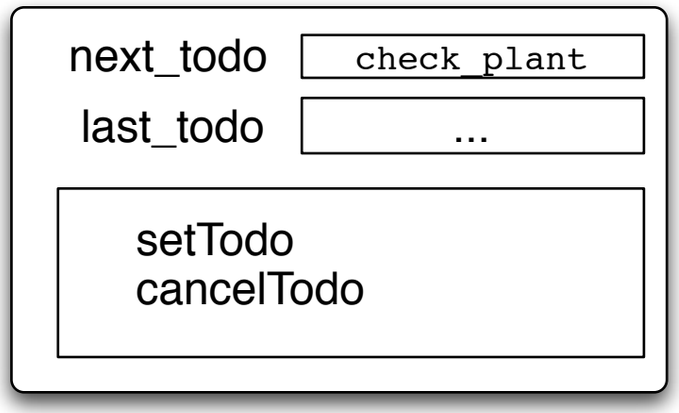
a Stock Quote Web Service



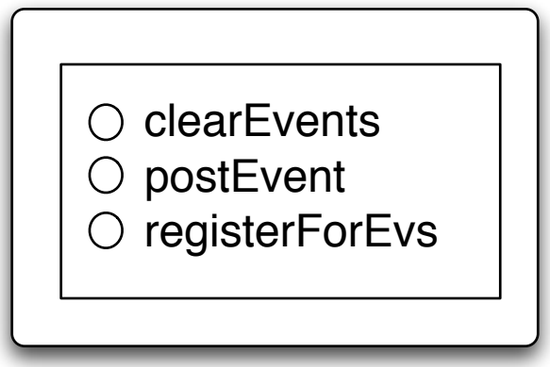
a data-base



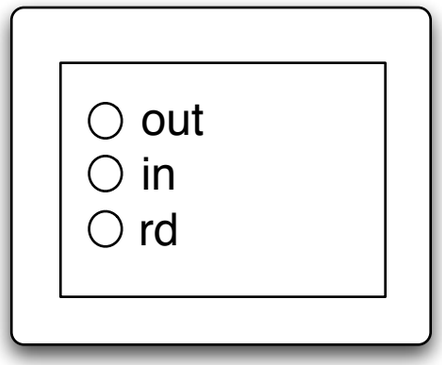
a bounded buffer



an agenda



an event service

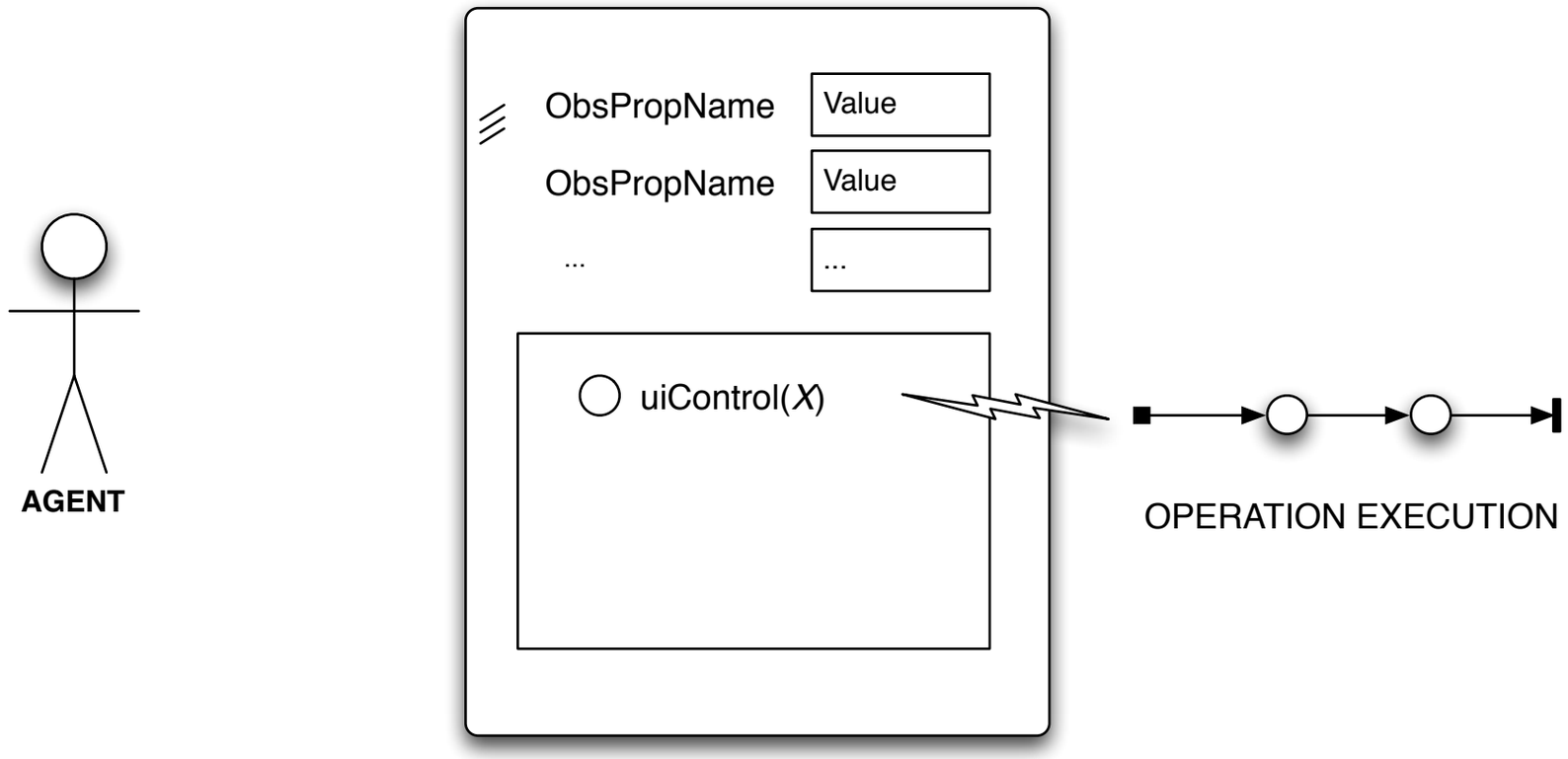


a tuple space

AGENT-ARTIFACT INTERACTION

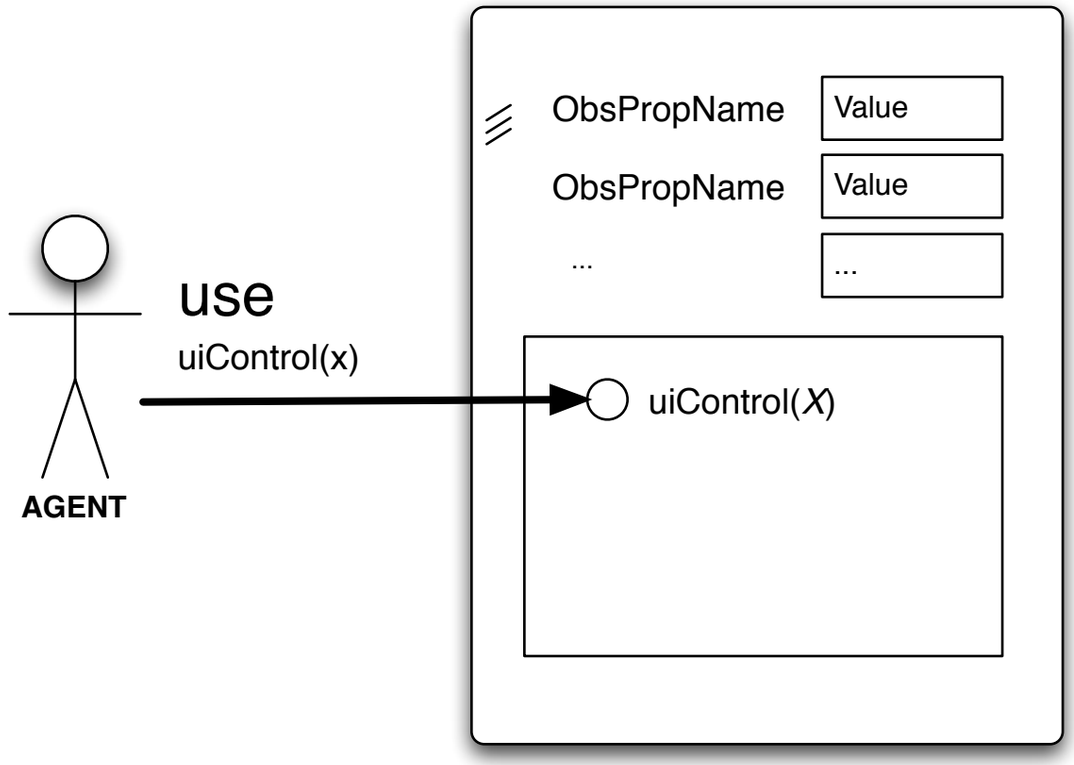
- Based on the concept of *use* and *observation*
  - triggering and controlling the execution of operations by acting on artifact usage interface
  - perceiving events generated by operation execution
    - as percepts concerning events happening in the artifact
  - perceiving artifact observable properties
    - as percepts concerning the state of the artifact

INTERACTION MODEL: USE



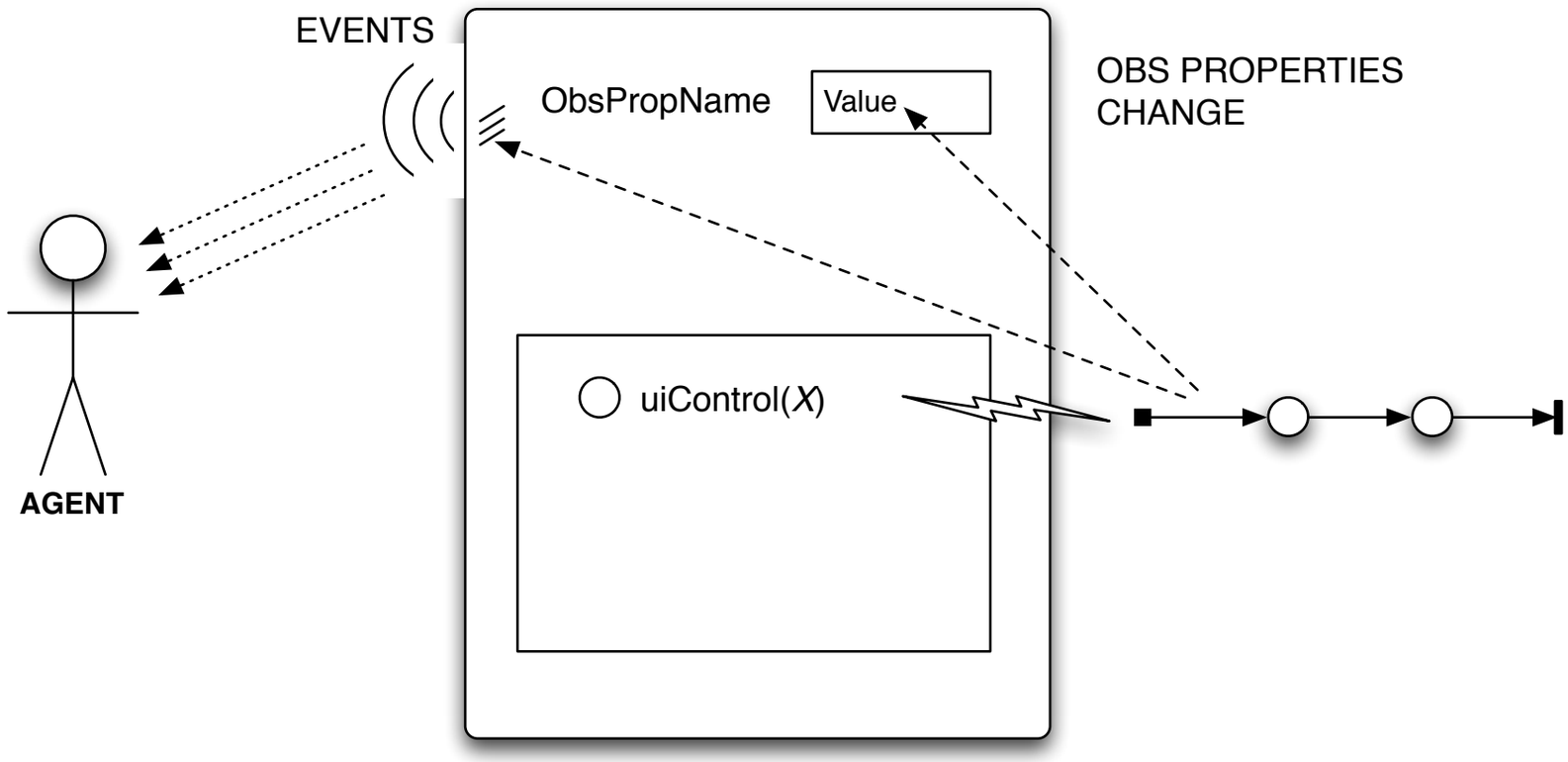
- artifact operation execution
  - asynchronous wrt agent
  - possibly a process structured in multiple atomic steps

INTERACTION MODEL: USE



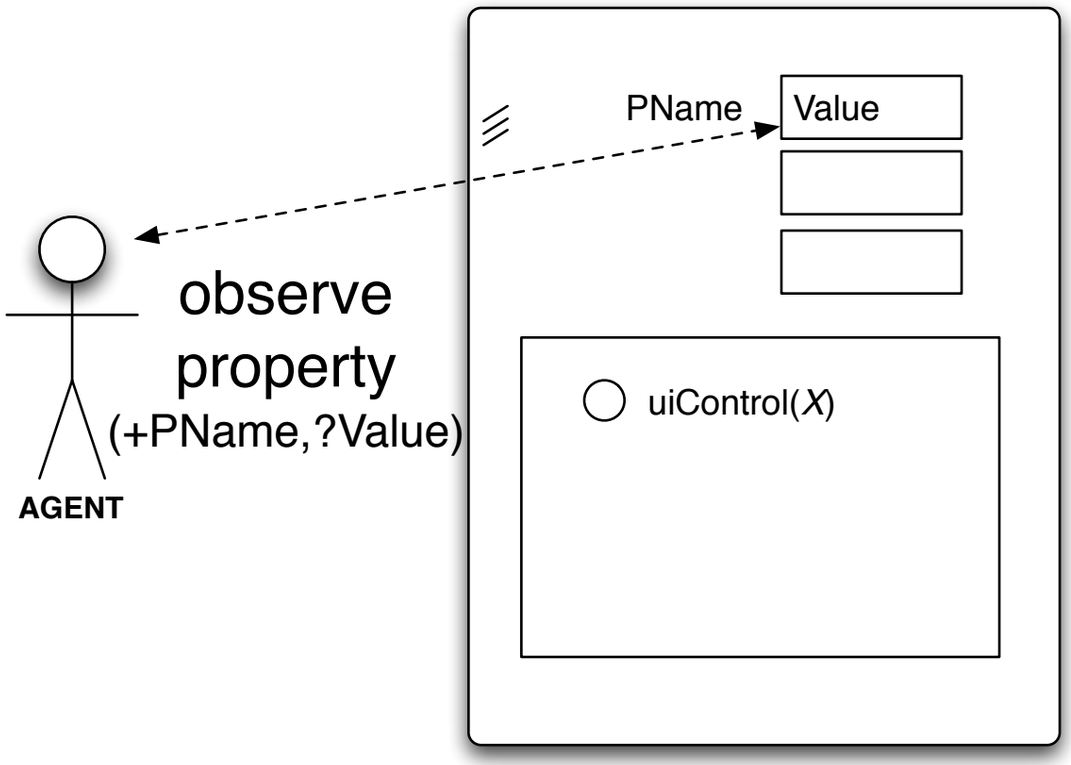
- use action
  - acting on op. controls to trigger op execution
  - **synchronisation point** with artifact time/state

INTERACTION MODEL: USE



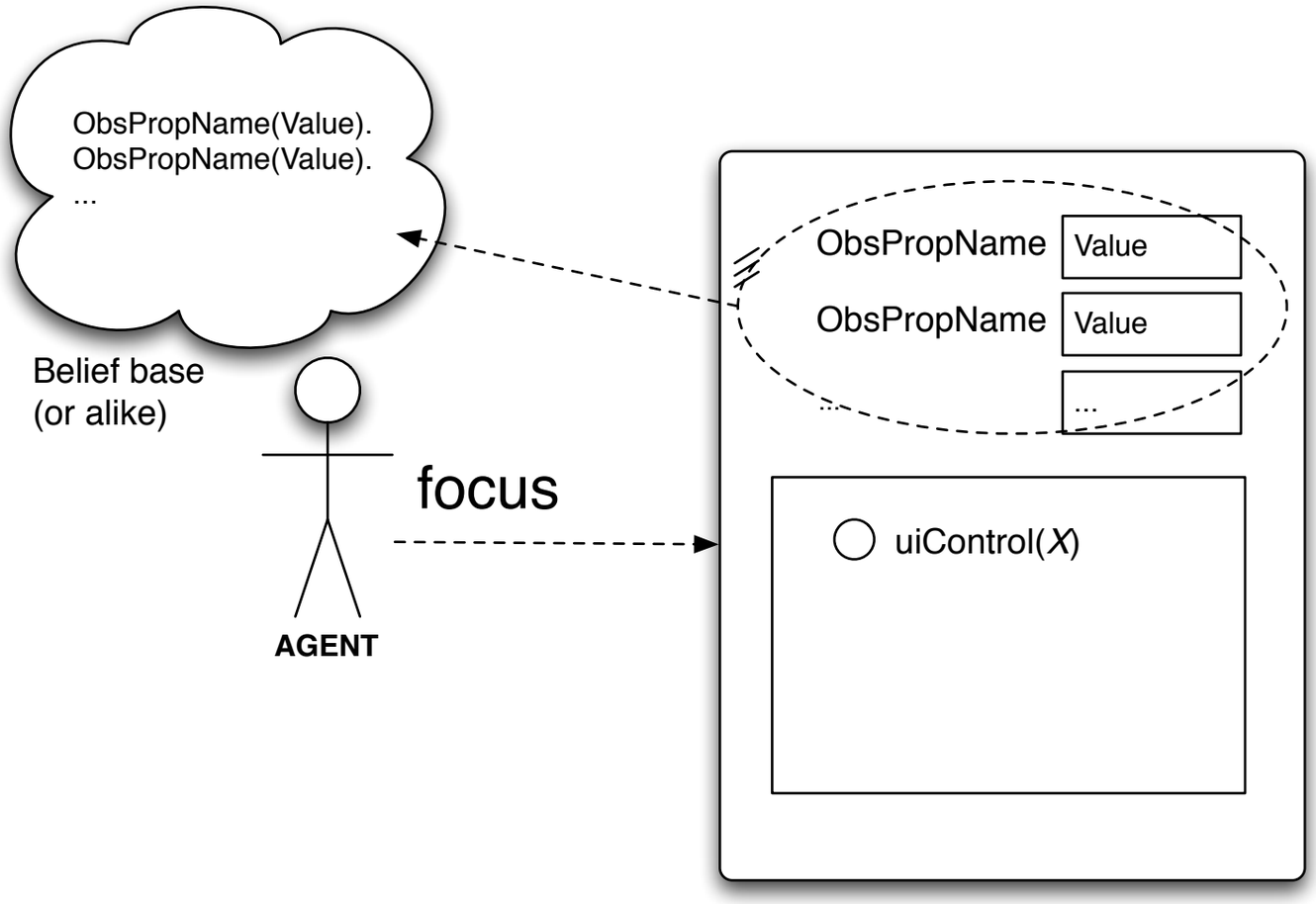
- observable effects
  - observable events & changes in obs property
  - perceived by agents either as (external) events

INTERACTION MODEL: OBSERVATION



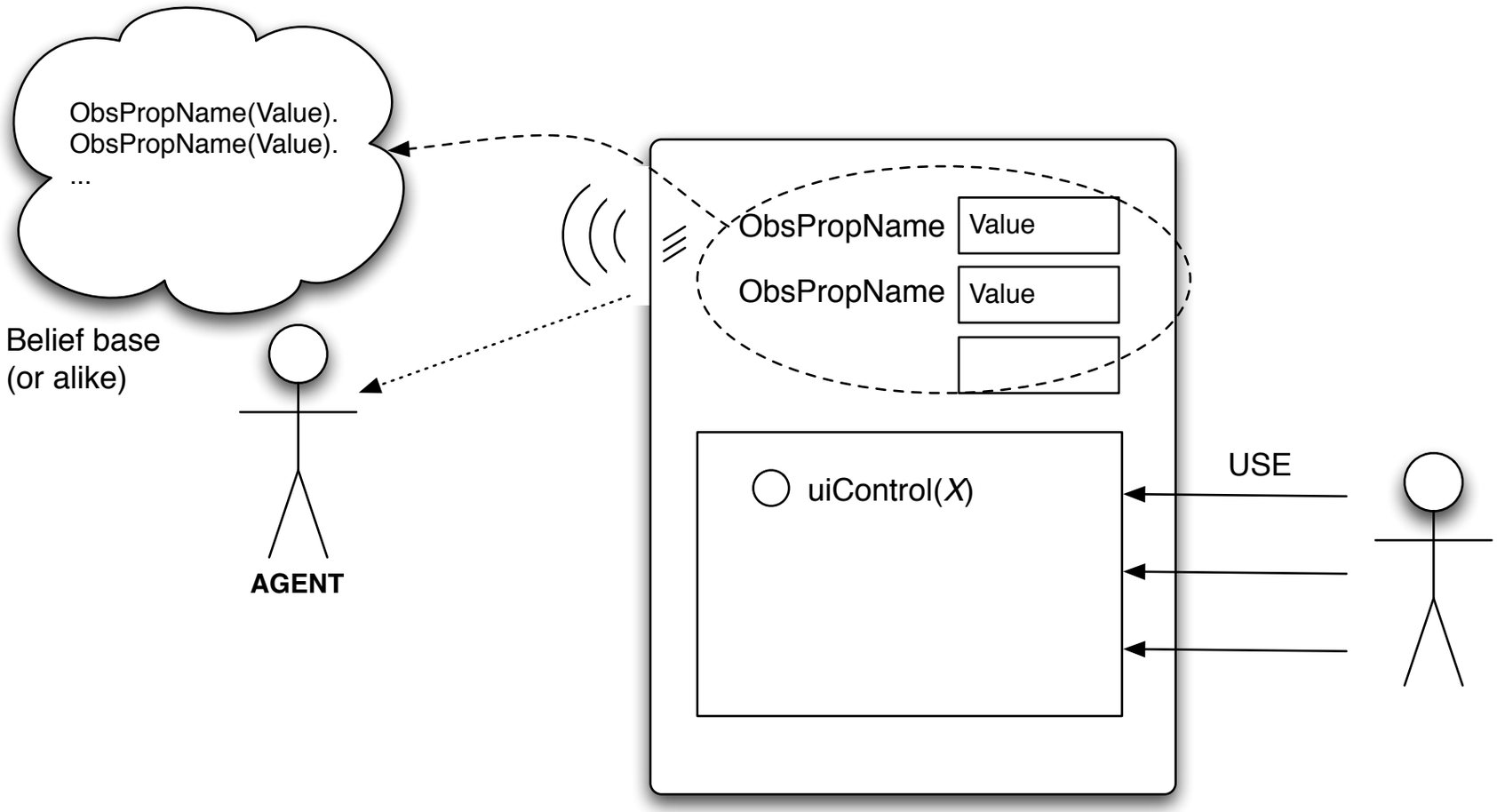
- observeProperty action
  - value of an obs. property as action feedback
  - *no interaction*

INTERACTION MODEL: OBSERVATION



- focus / stopFocus action
  - start / stop a continuous observation of an artifact
    - possibly specifying filters
  - observable properties mapped into percepts

INTERACTION MODEL: OBSERVATION



- continuous observation
  - observable events (> agent events)
  - observable properties (> belief base update)

ARTIFACT MODEL HIGHLIGHTS

- Artifacts as **controllable** and **observable** devices
  - operation execution as a controllable process
    - possibly long-term, articulated
  - two observable levels
    - properties, events
  - transparent management of concurrency issues
    - synchronisation, mutual-exclusion, etc
- Composability through **linking**
  - also across workspaces
- Cognitive use of artifacts through the **manual**
  - function description, operating instructions

CATEGORIES OF ARTIFACTS

- *Personal* artifacts
  - designed to provide functionalities for a single agent use
    - e.g. agenda
- *Social* artifacts
  - designed to provide some kind of global functionalities
    - communication, coordination, organisation...
    - e.g. blackboards, tuple spaces, bounded buffers, etc.
- *Boundary* artifacts
  - designed to wrap the interaction with external systems or to represent inside the MAS an external system
    - e.g. data-base, GUI, Web Services, etc.

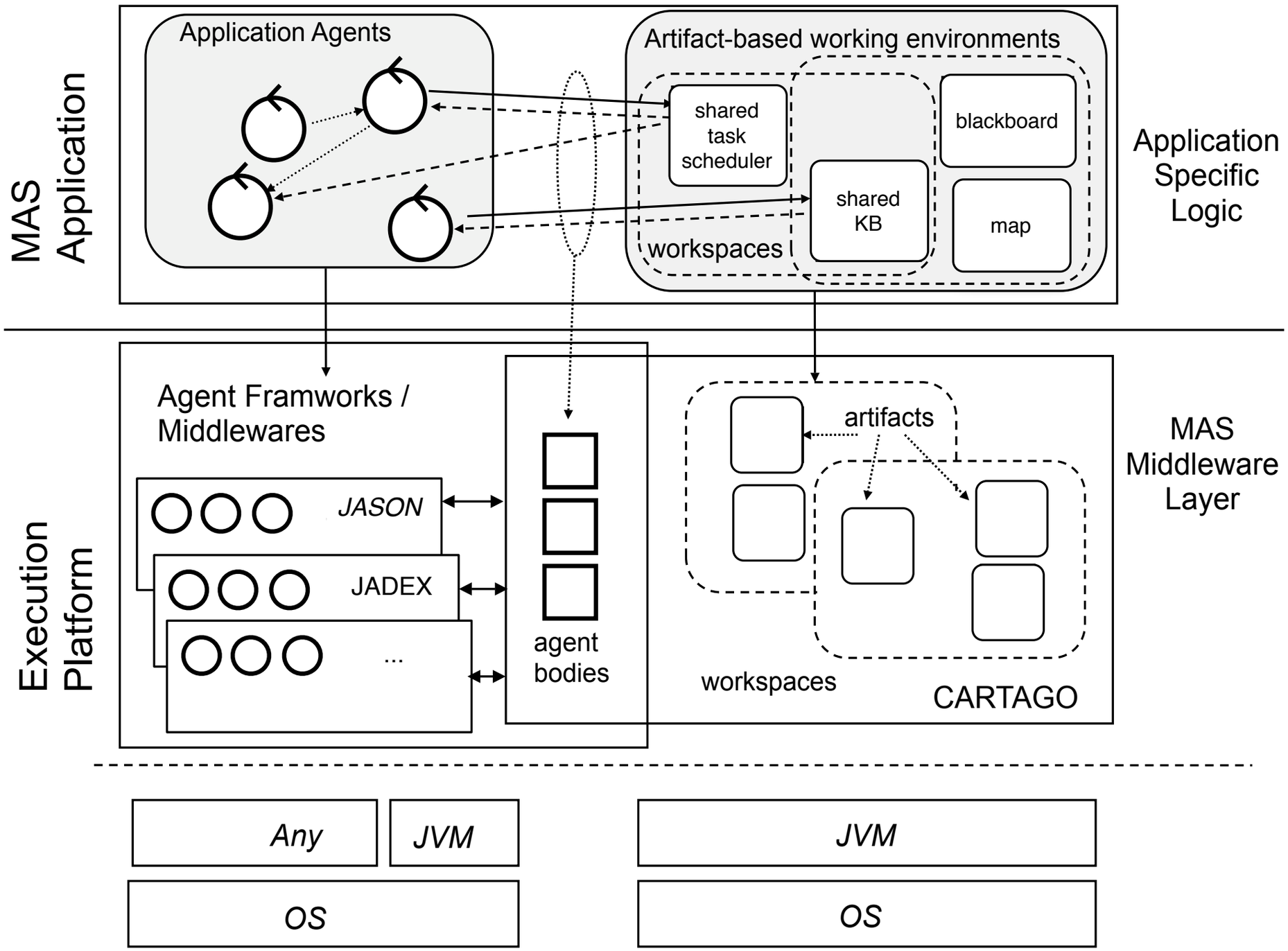
CArtAgO

- Platform / infrastructure implementing the A&A model [Ricci, Viroli, Omicini - 2006] [Ricci, Piunti, Viroli, Omicini - 2009]
  - runtime environment for executing (possibly distributed) artifact-based environments
  - Java-based programming model for defining artifacts
  - set of basic API for agent plaforms to work wihin artifact-based environment
    - integration with heterogeneous agent programming platforms [Ricci, Piunti, Lacay, Bordini, Hubner, Dastani - 2008]
- Distributed and open MAS
  - workspaces distributed on Internet nodes
    - agents can join and work in multiple workspace at a time
  - Role-Based Access Control (RBAC) security model
- Open-source technology
  - available at http://cartago.sourceforge.net

INTEGRATION

- Integration with existing agent platforms
  - available bridges: Jason, Jadex, AgentFactory, simpA
    - ongoing: 2APL, Jade
- Outcome
  - developing open and heterogenous MAS
  - different perspective on interoperability
    - sharing and working in a common work environment
    - common data-model based on OOP

CArtAgO ARCHITECTURE



CArtAgO ABSTRACT API (CORE)

- Extending agent actions with a basic set to work within artifact-based environments

workspace management	`joinWsp(Name,?WspId,+Node,+Role,+Cred)` `quitWsp(Wid)`
artifact use	`use(Aid,UICntrName(Params),+Sensor,+Timeout,+Filter)` `sense(Sensor,?Perception,+Filter,+Timeout)`

artifact observation	`observeProperty(Aid,PName,?PValue)`
	`focus(Aid,+Sensor,+Filter)`
	`stopFocus(Aid)`

artifact instantiation, discovery, management	`makeArtifact(Name,Template,+ArtifactConfig,?Aid)`
	`lookupArtifact(Name,?Aid)`
	`disposeArtifact(Aid)`

RAW AGENT API

```
joinWsp
quitWsp

use
sense
observeProp
focus
stopFocus
```

+

basic set of artifacts available in each workspace

- `factory`
- `registry`
- `security-registry`
- `console`
- …

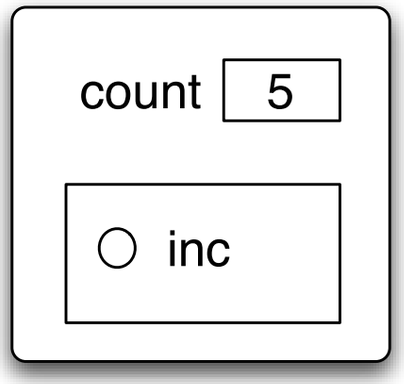
implementing non primitive actions:

`makeArtifact` => use `factory`

`lookupArtifact` => use `registry`

CArtAgO API TASTE
EX1: THE ~SIMPLEST ARTIFACT

- A shared counter



OBSERVABLE PROPERTIES:
count: int

USAGE INTERFACE:
inc: { op_exec_started(inc),
count(X),
op_exec_completed(inc) }

```
public class Counter extends Artifact {
  @OPERATION void init(){
    defineObsProperty("count",0);
  }

  @OPERATION void inc(){
    int count = getObsProperty("count").intValue();
    updateObsProperty("count",count+1);
  }
}
```

AGENTS IN JASON

USER(S)

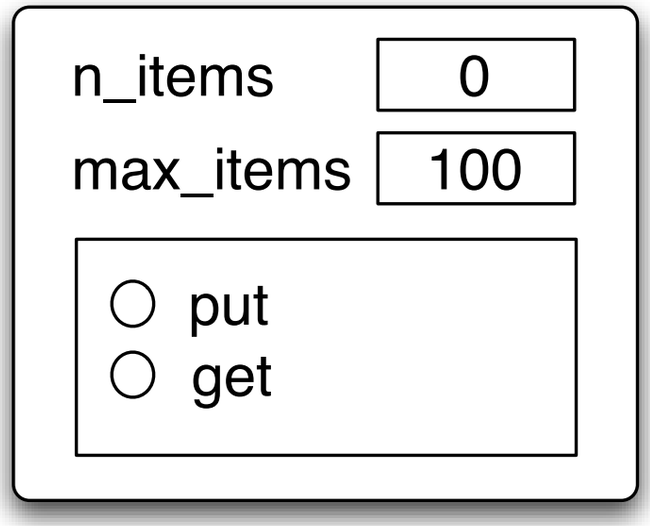
```
!create_and_use.

+!create_and_use : true <-
  !setupTool(C);
  // first use
  cartago.use(C,inc);
  // second use
  cartago.use(C,inc,s0);
  cartago.sense(s0,op_exec_completed).

+!setupTool(C): true <-
  cartago.makeArtifact("ourCount",
                       "Counter",C);
```

OBSERVER(S)

```
!observe.

+!observe : true <-
  ?myTool(C);
  cartago.focus(C).

+count(V) : V < 10 <-
  cartago.use(console,println(“count percept: “,V).
+count(V) [artifact(A)] : V >= 10 <-
  cartago.use(console,println(“stop observing.”);
  cartago.stopFocus(A).

+?myTool(CounterId): true <-
  cartago.lookupArtifact(“ourCount”,CounterId).
-?myTool(CounterId): true <-.wait(10); ?ourCount(C).
```

CArtAgO API TASTE
EX2: USING UI CONTROLS WITH GUARDS

- *bounded-buffer* artifact for producers-consumers scenarios



OBSERVABLE PROPERTIES:

n_items: int+
max_items: int

USAGE INTERFACE:

put(*item*:Item) / (n_items < max_items): {...}

get / (n_items >= 0) :
{ new_item(item:Item),...}

```
public class BBuffer extends Artifact {
  private LinkedList<Item> items;

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

  @OPERATION(guard="bufferNotFull") void put(Item obj){
    items.add(obj);
    updateObsProperty("n_items",items.size()+1);
  }
  @GUARD boolean bufferNotFull(Item obj){
    int maxItems = getObsProperty("max_items").intValue();
    return items.size() < maxItems;
  }

  @OPERATION(guard="itemAvailable") void get(){
    Item item = items.removeFirst();
    updateObsProperty("n_items",items.size()-1);
    signal("new_item",item);
  }
  @GUARD boolean itemAvailable(){ return items.size() > 0; }
}
```

CArtAgO API TASTE

PRODUCERS & CONSUMERS IN JASON

PRODUCERS

```
!produce.

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

+!produceItems : true <-
  ?nextItemToProduce(Item);
  cartago.use(myBuffer,put(Item));
  !!produceItems.

+?nextItemToProduce(Item) : true <- ...

+!setupTools(Buffer) : true <-
  cartago.makeArtifact("myBuffer",
                "BBuffer",[10],Buffer).
-!setupTools(Buffer) : true <-
  cartago.lookupArtifact("myBuffer",Buffer).
```

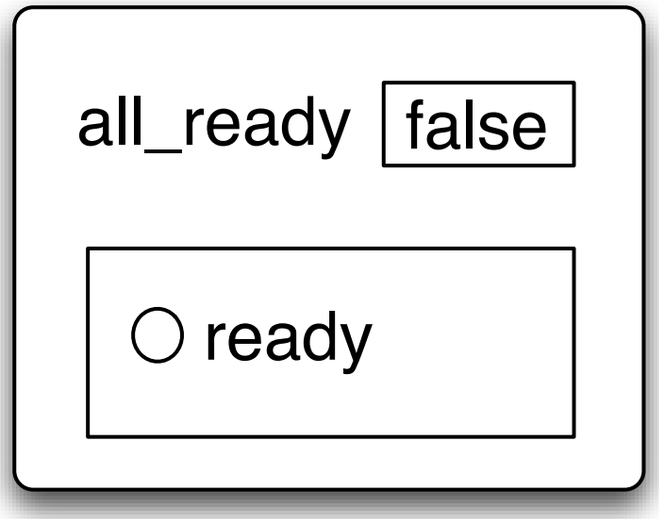
CONSUMERS

```
!consume.

+!consume: true <-
  ?bufferToUse(Buffer);
  !consumeItems.

+!consumeItems : true <-
  cartago.use(myBuffer,get,s0);
  cartago.sense(s0,new_item(Item));
  !consumeItem(Item);
  !!consumeItems.

+!consumeItem(Item) : true <- ...

+?bufferToUse(BufferId) : true <-
  cartago.lookupArtifact("myBuffer",BufferId).
-?bufferToUse(BufferId) : true <-
  .wait(50);
  ?bufferToUse(BufferId).
```

CArtAgO API TASTE

EX3: ARTIFACT WITH MULTI-STEP OPERATIONS

- simple synchronisation artifact (~barrier)



OBSERVABLE PROPERTIES:

all_ready: {true,false}

USAGE INTERFACE:

ready / true: { op_exec_completed }

```
class SimpleSynchroniser extends Artifact {

  int nReady, nParticipants;

  @OPERATION void init(int nParticipants){
    defineObsProperty("all_ready",false);
    nReady = 0;
    this.nParticipants = nParticipants;
  }

  @OPERATION void ready(int nParticipants){
    nReady++;
    nextStep("setAllReady");
  }

  @OPSTEP(guard="allReady") void setAllReady(){
    updateObsProperty("all_ready",true);
  }

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

CArtAgO API TASTE
SYNCH USER (ACTIVE/REACTIVE)

SYNCH USER - WITH SENSOR

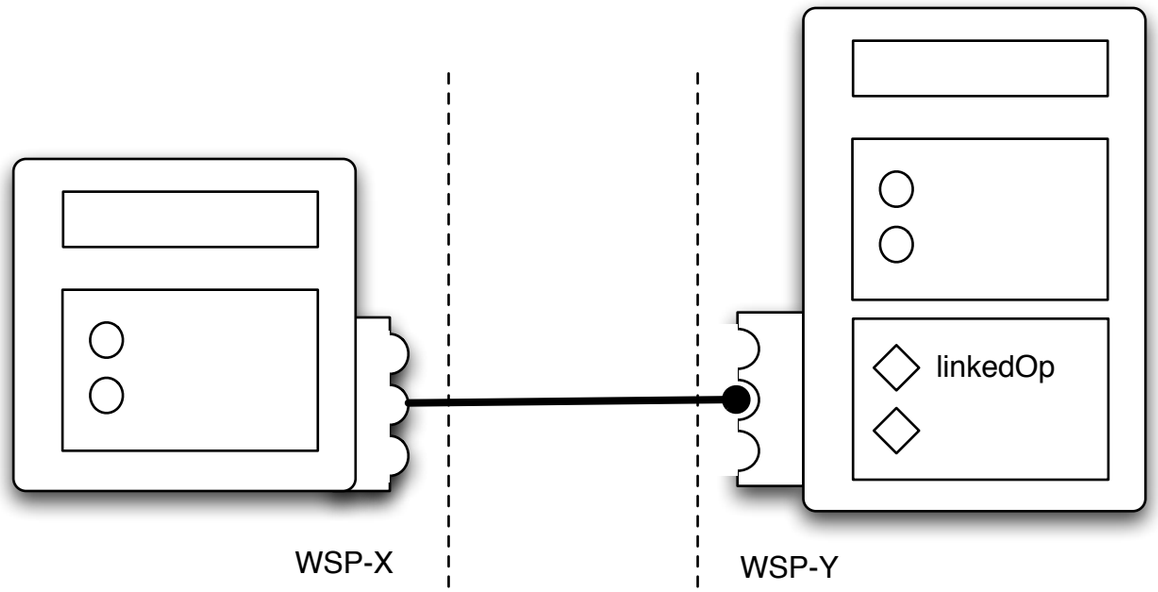
```
!work.

+!work: true <-
  ...
  // locate the synch tool
  cartago.lookupArtifact(“mySynch”,Synch);
  // ready for synch
  cartago.use(Synch,ready,sid);
  // waiting all synchs
  cartago.sense(sid,op_exec_completed(“ready”);
  // all ready, go on.
  ...
```

SYNCH USER - REACTIVE

```
!work.

+!work: true <-
  ...
  // locate the synch tool
  cartago.lookupArtifact(“mySynch”,Synch);
 // observe it.
  cartago.focus(Synch);
  // ready for synch
  cartago.use(Synch,ready).

// react to all_ready(true) percept
+all_ready(true)[artifact(mySynch)] : true
  <-
  // all ready, go on.
  ...
```

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

SOME EXISTING APPLICATIONS

- Organisational infrastructures
  - ORA4MAS [Hubner, Boissier, Kitio, Ricci - 2009]
    - exploiting artifacts to build an organisational infrastructure
- SOA & Web Services
  - CArtAgO-WS [Ricci, Denti, Piunti - 2009]
    - basic set of artifacts for building SOA/WS applications
      - interacting with web services
      - implementing web services
    - programming SOA/WS applications as ensemble of (intelligent) agents working inside workspaces
- Autonomic Computing & Virtualisation
  - implementing autonomic distributed systems as set of (intelligent) agents exploiting artifacts representing virtual machines

WRAP-UP

- Environment programming
  - environment as a programmable part of the MAS
  - programming agents' world for agents' use
- Artifact-based computational & programming model
  - 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

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

WOA 2009 MINI-SCUOLA
ENVIRONMENT PROGRAMMING IN MULTI-AGENT SYSTEMS

WRAP-UP

SUM-UP

- Environment as a first-class design and programming abstraction in AOSE and AOP
  - encapsulating core functionalities for agent(s) work
  - improving separation of concerns in MAS engineering
- Environment as an abstraction layer perspective
  - defining such computational world that agents dynamically exploit, adapt, change for their purposes
- General-purpose computational / programming models / platforms
  - orthogonal to agent models / architecture
  - an example: artifacts and CArtAgO

RESEARCH DIRECTIONS
- TWO SELECTED TOPICS -

- Cognitive use of the environment
  - e.g. in the artifact-based environments: agents dynamically selecting which artifacts to use and how to use them
    - link with semantic web & Service-Oriented research
    - fundamental role of ontologies
  - open MAS perspective
- Integrated platforms & standardisation
  - towards a common model, ontology and platform for environments in AOP and AOSE
  - analogously to the FIPA initiative for ACL

BIBLIOGRAPHY

- [Báez-Barranco, Stratulat, Ferber - 2007] A Unified Model for Physical and Social Environments. A Unified Model for Physical and Social Environments. In [Weyns, Parunak, Michel - 2007], Springer
- [Bordini, Hübner, Woodridge - 2007] Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons, Ltd, 2007.
- [Bromuri, Stathis - 2008] Situating Cognitive Agents in GOLEM. In D. Weyns, S.A. Brueckner, and Y. Demazeau (Eds.): EEMMAS 2007, LNAI 5049, pp. 115–134, 2008.
- [Campos et al., 2009] Formalising Situatedness and Adaptation in Electronic Institutions. Jordi Campos, Maite López-Sánchez, Juan Antonio Rodríguez-Aguilar, and Marc Esteva. In "Coordination, Organizations, Institutions and Norms in Agent Systems IV", LNAI 5428/2009, Springer
- [Dastani, Bobo, Meyer - 2007] Practical Extensions in Agent Programming Languages. AAMAS’07 2007 Honolulu, Hawaii USA
- [Esteva et al - 2004] AMELI: An Agent-based Middleware for Electronic Institutions. Marc Esteva, Bruno Rosell, Juan A. Rodrıguez-Aguilar, Josep Ll. Arcos. AAMAS'04, July 19-23, 2004, New York, New York, USA.
- [Ferber, Michel, Baez - 2005] AGRE: Integrating environments with organizations. In [Weyns, Parunak, Michel - 2005a], Springer-Verlag.

BIBLIOGRAPHY

- [Ferber, 1999] Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence. Addison-Wesley, Great Britain
- [Ferber and Müller, 1996] Influences and reaction: A model of situated multiagent systems. In M. Tokoro (Ed.), Second international conference on multi-agent systems (ICMAS 1996). Kyoto, Japan, AAAI Press, Menlo Park, California, USA.
- [Gelernter, 1985] Generative communication in Linda. ACM Transactions on Programming Languages and Systems, 7(1):80–112, January 1985.
- [Hubner, Boissier, Kitio, Ricci - 2009] Instrumenting multi-agent organisations with organisational artifacts and agents: “Giving the organisational power back to the agents”. Autonomous Agents and Multi-Agent Systems, April 2009. Springer-Verlag
- [Jennings 2001] An agent-based approach for building complex software systems. Commun. ACM, 44(4):35-41.
- [Mamei, Zambonelli - 2005] Programming Stigmergic Coordination with the TOTA Middleware. AAMAS'05, July 2529, 2005, Utrecht, Netherlands.
- [Mamei, Zambonelli - 2006] Field-based coordination for pervasive multiagent systems, Springer. Series on Agent Technology. Springer-Verlag.
- [Murphy, Picco, Roman - 2001] LIME: A middleware for physical and logical mobility. In Twenty-First international conference on distributed computing systems (ICDCS 2001) (p. 254). Washington, DC, USA, IEEE Computer Society.

BIBLIOGRAPHY

- [Noriega, 1997] Agent-Mediated Auctions: The Fishmarket Metaphor. Number 8 in IIIA Phd Monograph. 1997.
- [Omicini and Zambonelli - 1999] Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems, 2(3):251–269, Sept. 1999. Special Issue: Coordination Mechanisms for Web Agents.
- [Omicini and Denti - 2001] From tuple spaces to tuple centres. Science of Computer Programming, 41(3):277–294, Nov. 2001.
- [Omicini, Ricci, Viroli - 2008] Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems, 17 (3), Dec. 2008
- [Omicini, Ricci, Viroli, Castelfranchi, Tummolini - 2004]. Coordination artifacts:
- Environment-based coordination for intelligent agents. In AAMAS'04, volume 1, pages 286–293, New York, USA, 19–23July 2004. ACM.
- [Omicini, Ossowski - 2003] Objective versus subjective coordination in the engineering of agent systems, in M. Klusch, S. Bergamaschi, P. Edwards & P. Petta, eds, 'Intelligent Information Agents: An AgentLink Perspective', Vol. 2586 of LNAI: State-of-the-Art Survey, Springer-Verlag, pp. 179–202.
- [Parunak, Brueckner, Sauter - 2005] Parunak,V.,Brueckner, S., Sauter, J. (2005). Digital pheromones for coordination of unmanned vehicles. In [Weyns, Parunak, Michel - 2005a].

BIBLIOGRAPHY

- [Platon, Sabouret, Honiden - 2006] Tag Interactions in Multi-Agent Systems: Environment Support. In [Weyns, Parunak, Michel - 2007], Springer
- [Pokahr, Braubach, Lamersdorf - 2005] Jadex: A BDI Reasoning Engine, Chapter of Multi-Agent Programming, Kluwer Book, Editors: R. Bordini, M. Dastani, J. Dix and A. Seghrouchni.
- [Ricci, Viroli, Omicini - 2006] Construenda est CArtAgO: Toward an Infrastructure for Artifacts in MAS. Proceedings of the International Symposium: "From Agent Theory to Agent Implementation." (AT2AI-6). Cybernetics and Systems 2006, 18-21 April 2006
- [Ricci, Viroli, Omicini - 2007] CArtAgO : A Framework for Prototyping Artifact-Based Environments in MAS. In [Weyns, Parunak, Michel - 2007], Springer
- [Ricci, Omicini, Denti - 2003]. Activity theory as a framework for MAS coordination. In P. Petta, R. Tolksdorf & F. Zambonelli (Eds.), Engineering societies in the agents world III (ESAW 2002),Vol. 2577 of lecture notes in computer science. Springer-Verlag.
- [Ricci, Piunti, Lacay, Bordini, Hubner, Dastani - 2008] Integrating artifact-based environments with heterogeneous agent-programming platforms. In Proceedings of 7th International Conference on Agents and Multi Agents Systems (AAMAS08), 2008.
- [Ricci, Denti, Piunti 2009] A Platform for Developing SOA/WS Applications as Open and Heterogeneous Multi-Agent Systems. Multi-Agent and Grid Systems. To appear

BIBLIOGRAPHY

- [Ricci, Piunti, Viroli, Omicini - 2009] Environment programming in CArtAgO. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2. Springer Verlag, 2009. To appear.
- [Ricci, Viroli, Omicini - 2007] The A&A programming model & technology for developing agent environments in MAS. In M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, editors,Post-proceedings of the 5th International Workshop “Programming Multi-Agent Systems” (PROMAS 2007), volume 4908 of LNAI, pages 91–109. Springer, 2007.
- [Russel and Norvig, 1995] Artificial Intelligence: A Modern Approach. Prentice Hall.
- [Stratulat, Ferber, Tranier - 2009] MASQ: towards an integral approach to interaction. Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary.
- [Weyns et al.] Weyns, D., Parunak,V.,Michel, F.,Holvoet, T.,Ferber, J. Environments formultiagent systems, state-of-the-art and research challenges. In [Weyns, Parunak, Michel - 2005a], Springer-Verlag.
- [Weyns, Omicini, and Odell - 2007] Environment as a first-class abstraction in multi-agent systems. Autonomous Agents and Multi-Agent Systems, 14(1):5–30, Feb. 2007. Special Issue on Environments for Multi-agent Systems.
- [Wooldridge and Jennings, 1995] Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, 10(2):115-152
- [Wooldridge, 2002] An Introduction to MultiAgent Systems. John Wiley and Sons, Ltd. England

BIBLIOGRAPHY
BOOKS / JOURNAL SPECIAL ISSUES

- [Weyns, Parunak - 2007] Journal of Autonomous Agents and Multi-Agent Systems. Special Issue: Environment for Multi-Agent Systems, volume 14(1). D. Weyns and H. V. D. Parunak, editors. Springer Netherlands, 2007.
- [Weyns, Parunak, Michel - 2005a] Environments for multiagent systems, first international workshop (E4MAS 2004), New York, USA, 2005. Revised Selected Papers, Vol. 3374 of lecture notes in computer science. Springer-Verlag
- [Weyns, Parunak, Michel - 2005b] Environments for multiagent systems II, second international workshop (E4MAS 2005), Utrecht, The Netherlands, 2005. revised papers and invited contributions, Vol. 3830 of lecture notes in computer science. Springer-Verlag
- [Weyns, Parunak, Michel - 2007] Environments for Multi-Agent Systems III, Third International Workshop, E4MAS 2006, Hakodate, Japan, May 8, 2006, Selected Revised and Invited Papers. Lecture Notes in Computer Science 4389 Springer 2007
- [Weyns, Brueckner, Demazeau] Engineering Environment-Mediated Multiagent Systems I (EEMMAS 07).