SISMA 2009/2010 - Seminar

ENVIRONMENT PROGRAMMING IN MAS WITH CArtAgO

Alessandro Ricci

aliCE group at DEIS, Università di Bologna, Cesena

a.ricci@unibo.it

OUTLINE

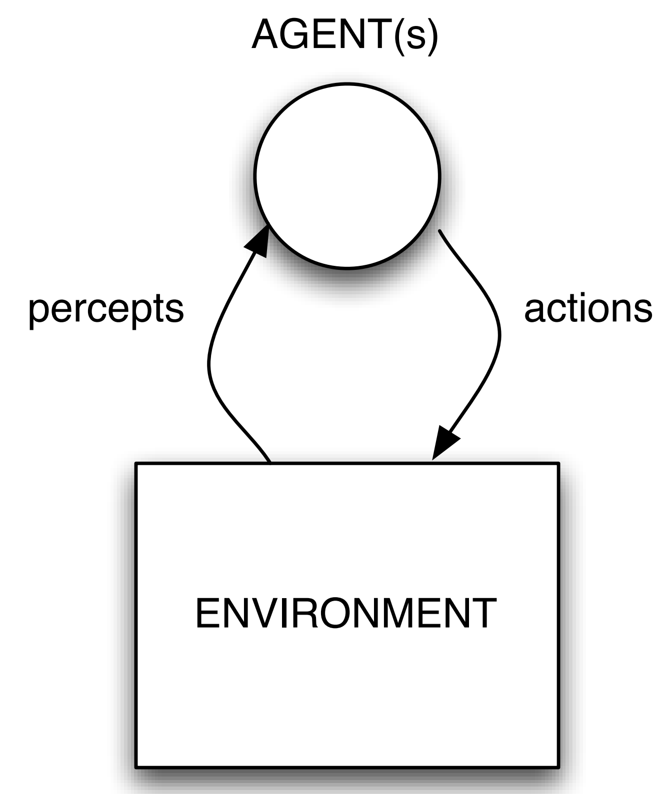
- Environment Programming in (Programming) MAS
  - the road to artifacts and CArtAgO
- A&A model and CArtAgO platform
  - programming model and technology
  - integration with existing agent languages / platforms
- Ongoing work & available projects/theses

PART I

ENVIRONMENT PROGRAMMING IN (PROGRAMMING) MAS

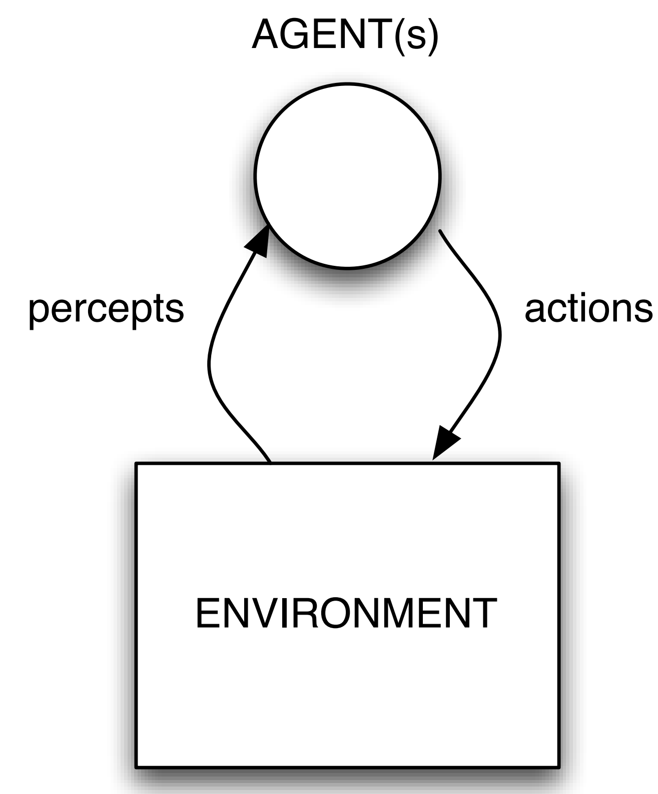
- The ROAD to CArtAgO -

THE ROLE OF ENVIRONMENT IN MAS



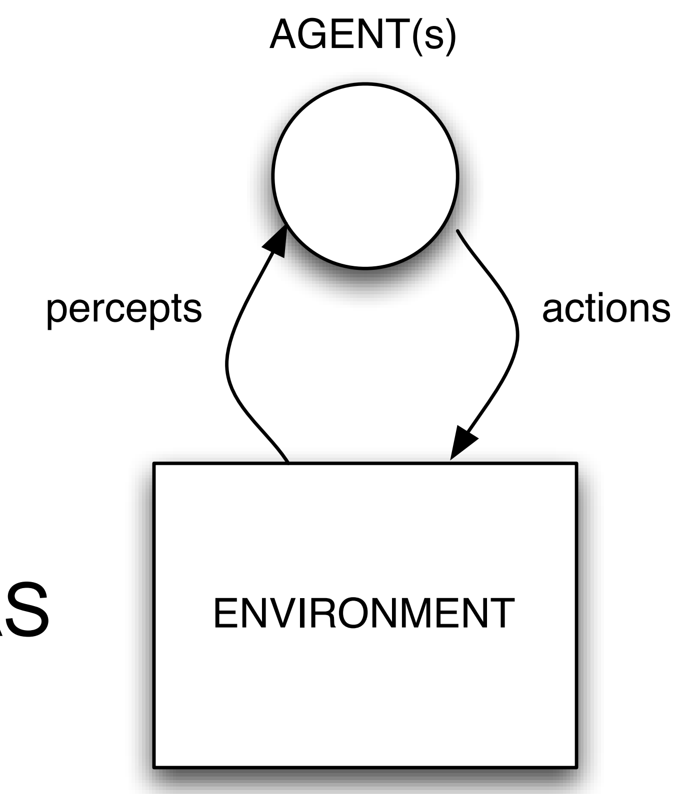
THE ROLE OF ENVIRONMENT IN MAS

- "Traditional" (D)AI / agent / MAS view
  - the target of agent actions and source of agents perception
  - something out of MAS design / engineering



THE ROLE OF ENVIRONMENT IN MAS

- "Traditional" (D)AI / agent / MAS view
  - the target of agent actions and source of agents perception
  - something out of MAS design / engineering
- New perspective in recent works
  - environment as first-class aspect in engineering MAS
    - mediating interaction among agents
    - encapsulating functionalities for managing such interactions
      - coordination, organisation, security,...



FROM MAS TO *MAS PROGRAMMING*

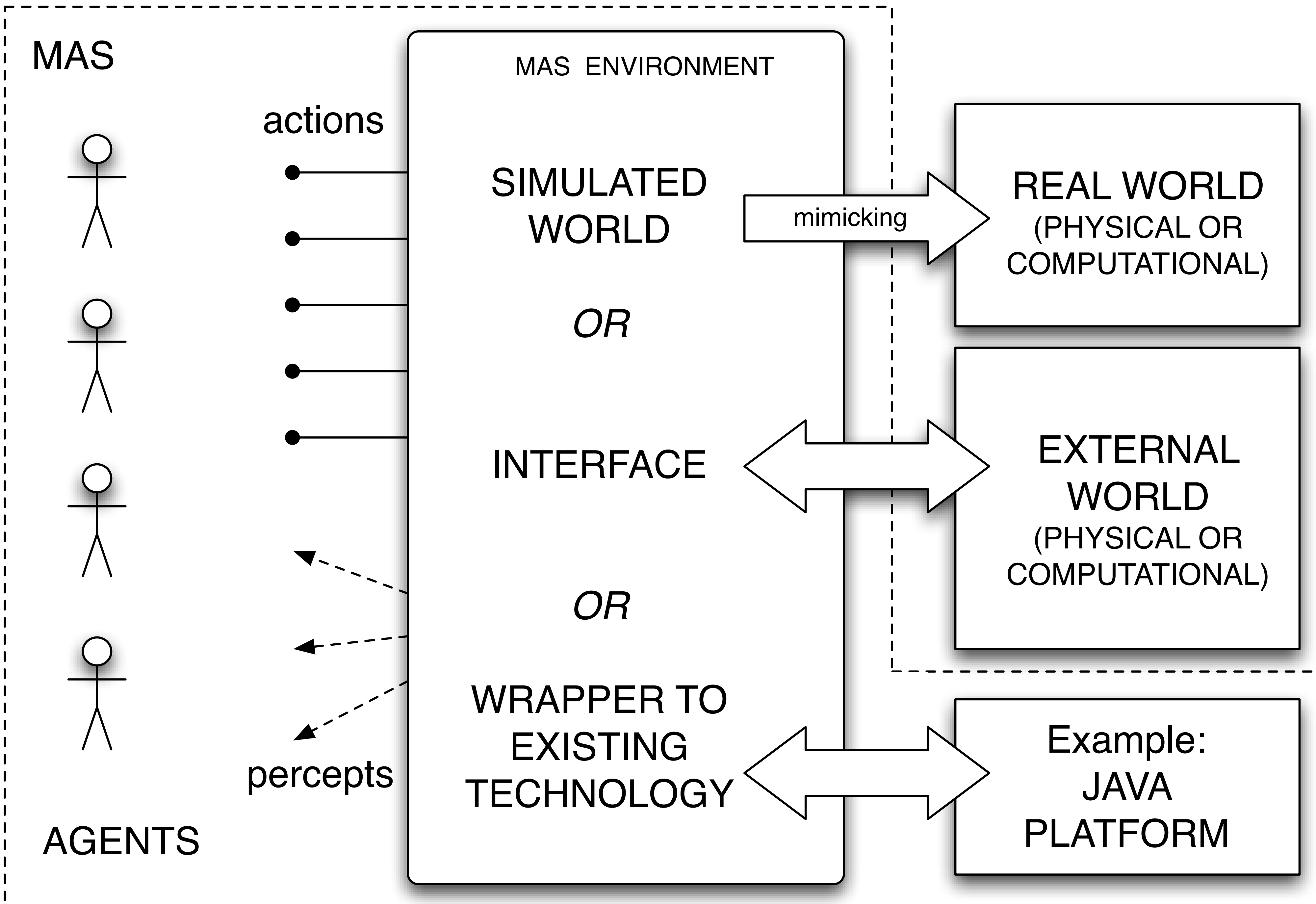
FROM MAS TO *MAS PROGRAMMING*

- Specific perspective on “MAS programming” adopted here
  - 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,...

FROM MAS TO *MAS PROGRAMMING*

- Specific perspective on “MAS programming” adopted here
    - 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,...
- Underlying objective in the long term
    - using agent-orientation as *general-purpose* post-OO paradigm for computer programming
        - concurrent / multi-core / distributed programming in particular

THE ROLE OF SW ENVIRONMENT IN MAS PROGRAMMING (SO FAR)



ENVIRONMENT MODEL IN MAS PROGRAMMING

ENVIRONMENT MODEL IN MAS PROGRAMMING

- Environment as monolithic / centralised block
  - defining agent (external) actions
    - typically a static list of actions, shared by all the agents
  - generator of percepts
    - establishing which percepts for which agents

ENVIRONMENT MODEL IN MAS PROGRAMMING

- Environment as monolithic / centralised block
  - defining agent (external) actions
    - typically a static list of actions, shared by all the agents
  - generator of percepts
    - establishing which percepts for which agents
- No specific programming model for defining structure and behaviour
  - including concurrency management
  - relying on lower-level language feature
    - e.g. Java

ENVIRONMENT MODEL IN MAS PROGRAMMING

- Environment as monolithic / centralised block
  - defining agent (external) actions
    - typically a static list of actions, shared by all the agents
  - generator of percepts
    - establishing which percepts for which agents
- No specific programming model for defining structure and behaviour
  - including concurrency management
  - relying on lower-level language feature
    - e.g. Java
- Typically enough for building simulated world

JASON EXAMPLE
- GOLD-MINER DEMO -

```
public class MiningPlanet extends jason.environment.Environment {
  ...
  public void init(String[] args) {...}

  public boolean executeAction(String ag, Structure action) {
      boolean result = false;
      int agId = getAgIdBasedOnName(ag);
      if (action.equals(up)) {
        result = model.move(Move.UP, agId);
      } else if (action.equals(down)) {
        result = model.move(Move.DOWN, agId);
      } else if (action.equals(right)) {
        ...
      }
      return result;
  }

  private void updateAgPercept(String agName, int ag) {clearPercepts(agName);
    // its location
    Location l = model.getAgPos(ag);
    addPercept(agName, Literal.parseLiteral("pos(" + l.x + "," + l.y + ")"));
    if (model.isCarryingGold(ag)) {
      addPercept(agName, Literal.parseLiteral("carrying_gold"));
    }
    // what's around
    updateAgPercept(agName, l.x - 1, l.y - 1);
    updateAgPercept(agName, l.x - 1, l.y);
    ...
  }
}
```

ENRICHING THE VIEW: WORK ENVIRONMENTS

ENRICHING THE VIEW: WORK ENVIRONMENTS

- Perspective: *designing worlds* for *agents' use & work*
  - designing good and effective place for agents to live and work in
    - environment as the context of agent activities *inside the MAS*
  - beyond simulated worlds

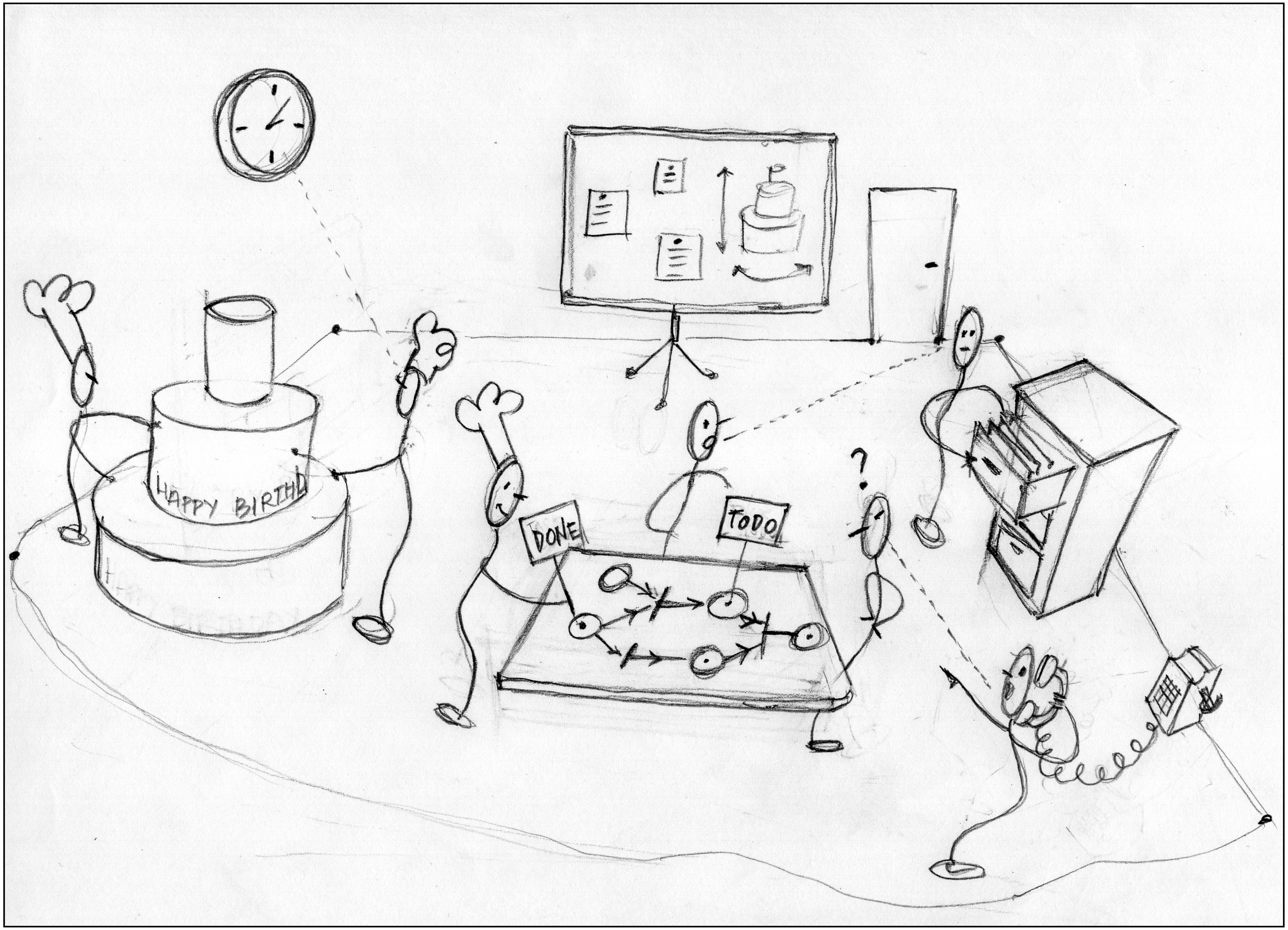
ENRICHING THE VIEW: WORK ENVIRONMENTS

- Perspective: *designing worlds* for *agents' use & work*
  - designing good and effective place for agents to live and work in
    - environment as the context of agent activities *inside the MAS*
  - beyond simulated worlds
- "**Work environment**" notion
  - that *part of the MAS* that is *designed* and *programmed* so as to ease agent activities and work
    - first-class entity of the agent world
    - cooperation, coordination, organisation, security... functionalities

ENRICHING THE VIEW: WORK ENVIRONMENTS

- Perspective: *designing worlds* for *agents' use & work*
  - designing good and effective place for agents to live and work in
    - environment as the context of agent activities *inside the MAS*
  - beyond simulated worlds
- "**Work environment**" notion
  - that *part of the MAS* that is *designed* and *programmed* so as to ease agent activities and work
    - first-class entity of the agent world
    - cooperation, coordination, organisation, security... functionalities
- Work environment as part of MAS design and programming
  - abstractions? computational models? languages? platforms? methodologies?

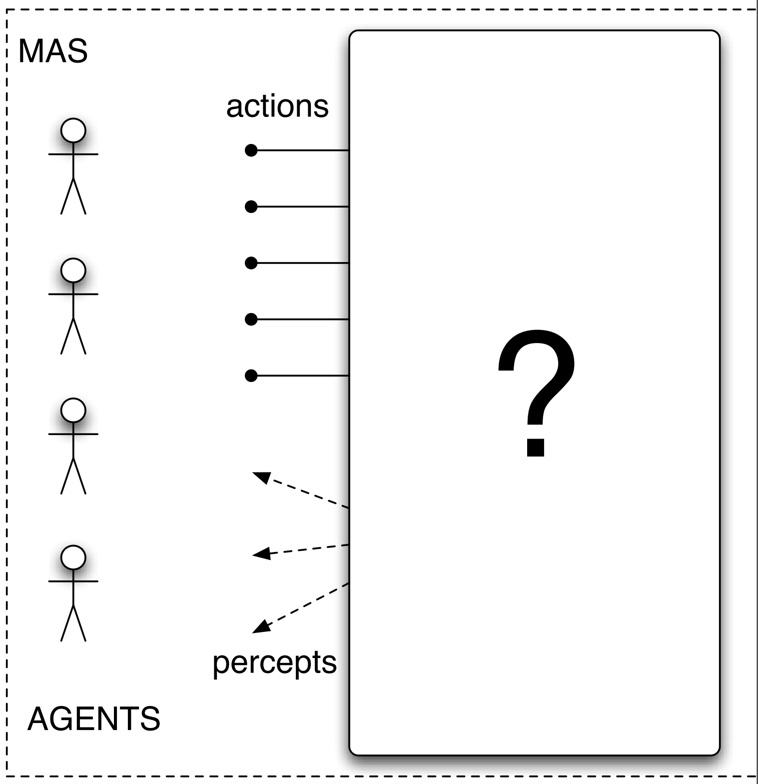
A HUMAN WORK ENVIRONMENT
(~BAKERY)



BACKGROUND LITERATURE

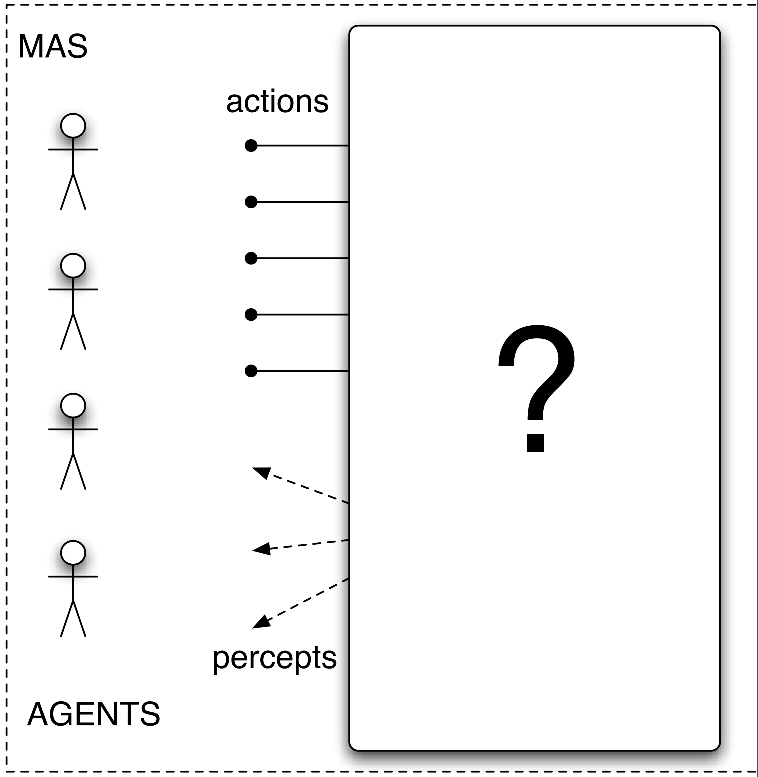
- In human science
  - Activity Theory, Distributed Cognition
    - importance of the environment, *mediation*, interaction for human activity development
  - CSCW and HCI
    - importance of artifacts and tools for coordination and collaboration in human work
  - Active Externalism / extended mind (Clark, Chalmer)
    - environment's obejcts role in aiding cognitive processes
- Distributed Artificial Intelligence
  - Agre & Horswil work ("Lifeworld"...)
  - Kirsch ("The Intelligent Use of Space"...)
  - ...

DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (1/2)



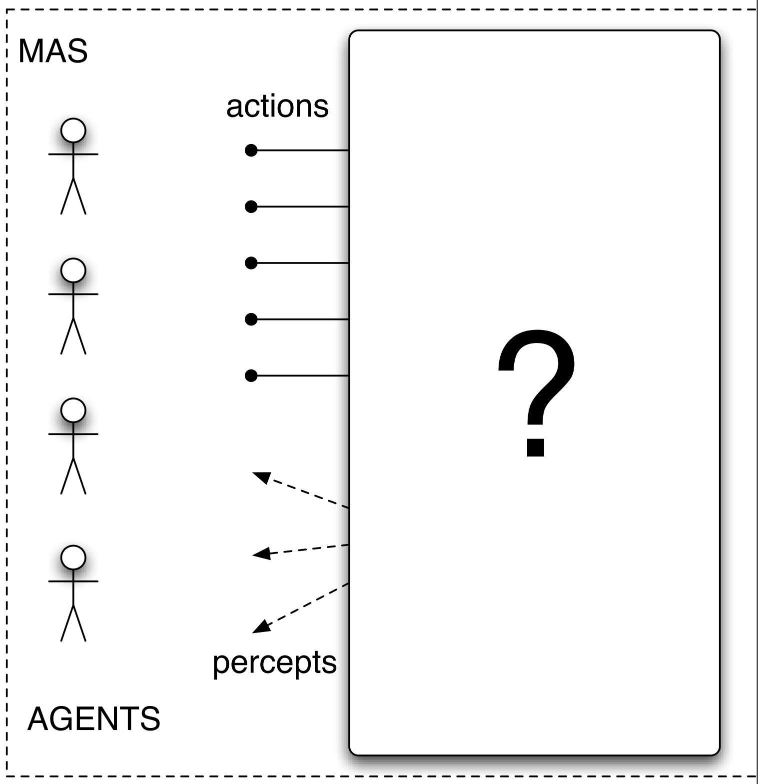
DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (1/2)

- **Abstraction**
  - keeping the agent abstraction level
    - e.g. no agents sharing and calling *OO objects*
  - effective programming models
    - for controllable and observable computational entities



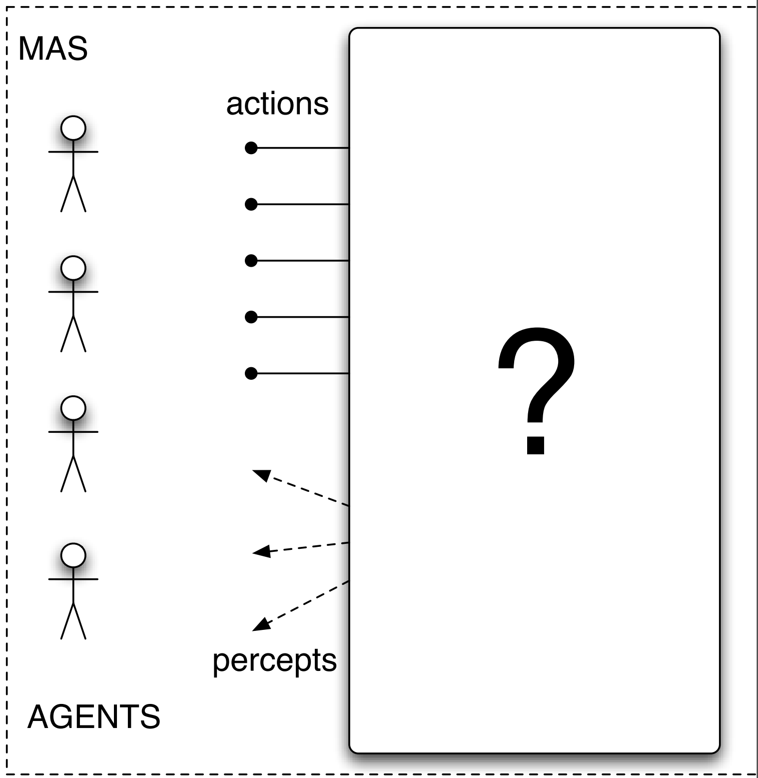
DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (1/2)

- **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

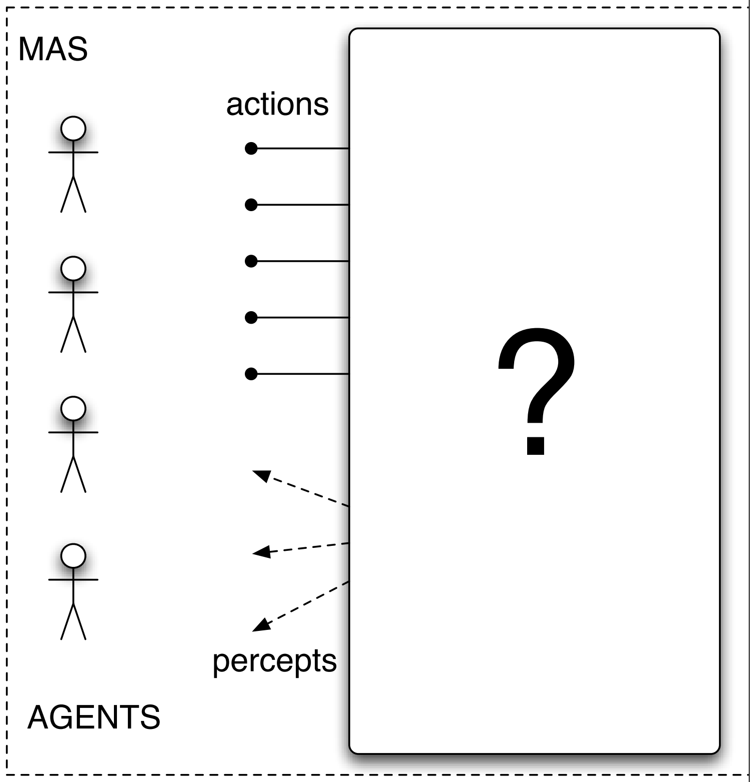


DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (1/2)

- **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

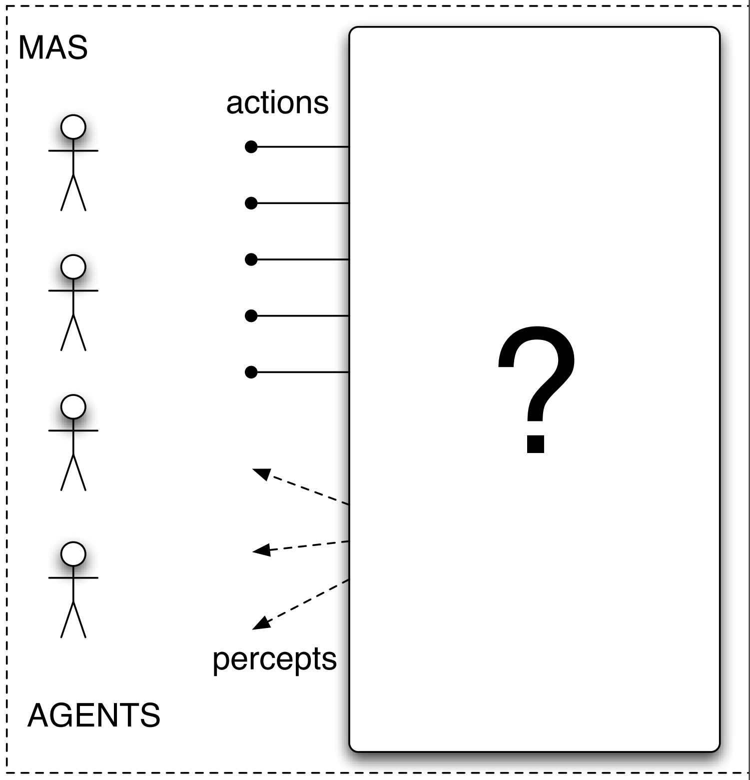


DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (2/2)



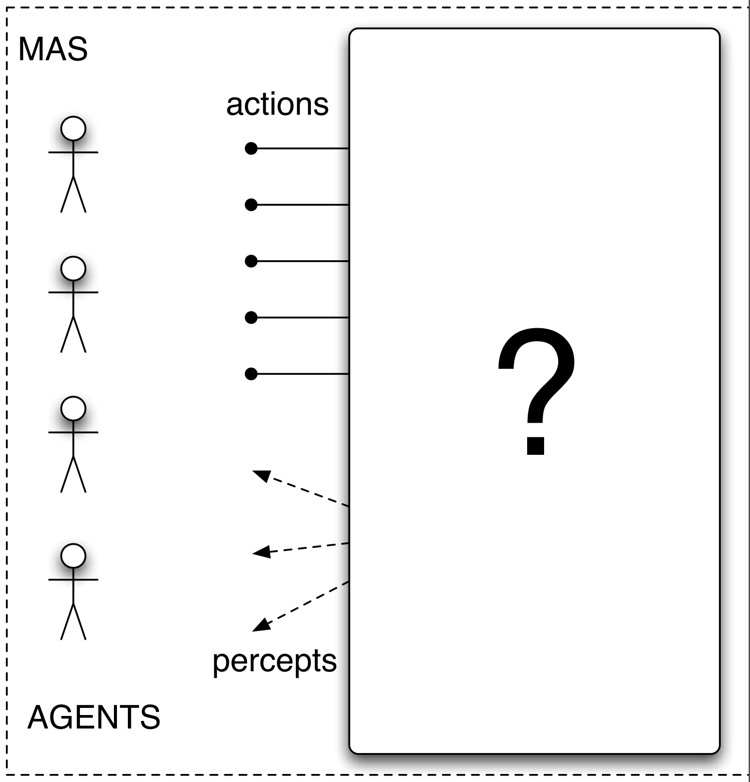
DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (2/2)

- **(Dynamic) extendibility**
  - dynamic construction, replacement, extension of environment parts
  - support for *open* systems



DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (2/2)

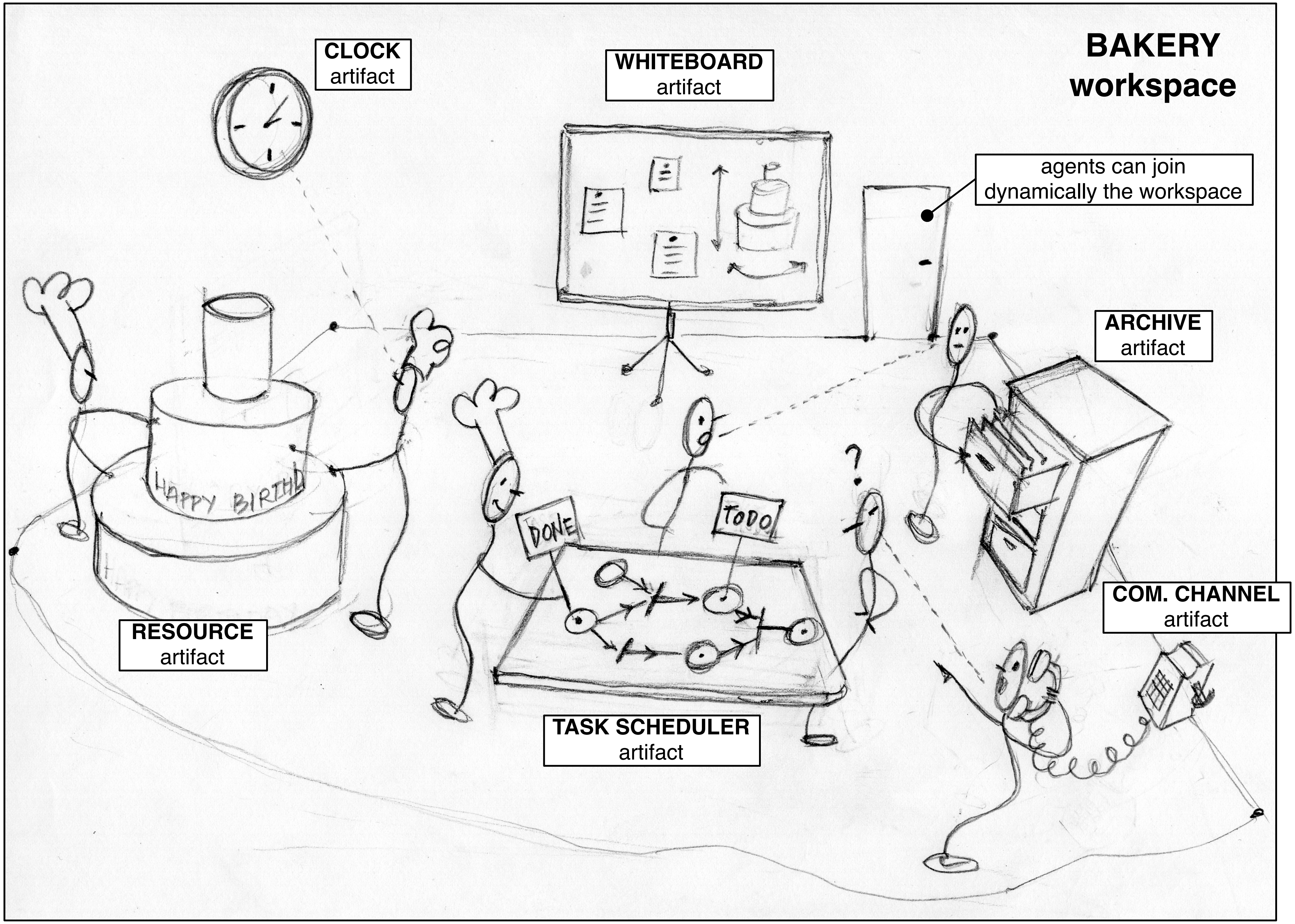
- **(Dynamic) extendibility**
  - dynamic construction, replacement, extension of environment parts
  - support for *open* systems
- **Reusability**
  - reuse of environment parts in different application contexts / domains



PART II

A&A MODEL and CArtAgO
PROGRAMMING MODEL & PLATFORM

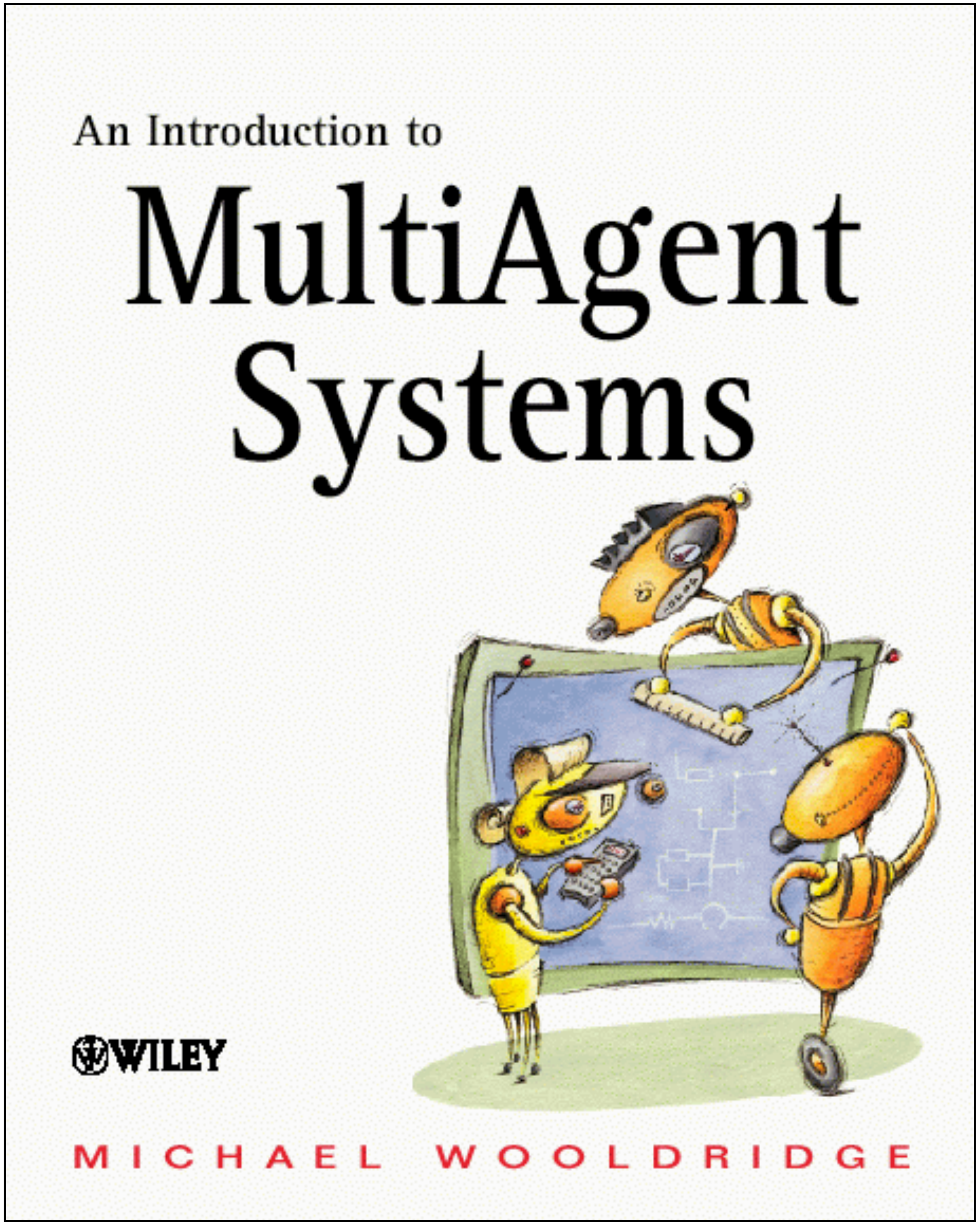
AGENTS & ARTIFACTS (A&A) MODEL: BASIC IDEA IN A PICTURE



A&A BASIC CONCEPTS

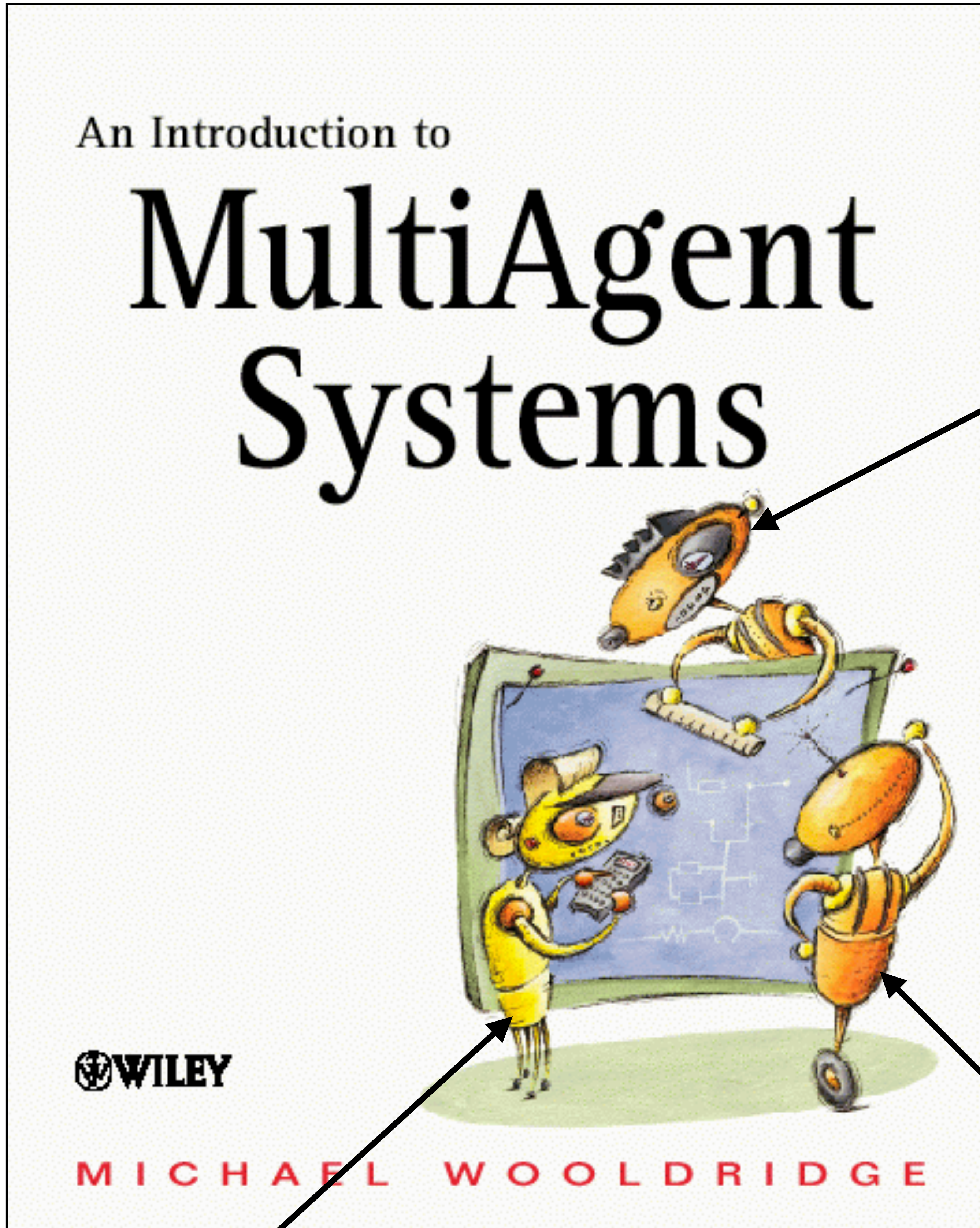
- **Agents**
  - autonomous, goal-oriented pro-active entities
  - create and co-use artifacts for supporting their activities
    - besides direct communication
- **Artifacts**
  - *non-autonomous*, *function*-oriented entities
    - controllable and observable (from the agent viewpoint)
  - modelling the tools and resources used by agents
    - designed by MAS programmers
- **Workspaces**
  - grouping agents & artifacts
  - defining the topology of the computational environment

ARTIFACTS
ARE IN THE
MAINSTREAM
...not really, actually...



ARTIFACTS
ARE IN THE
MAINSTREAM

...not really, actually...



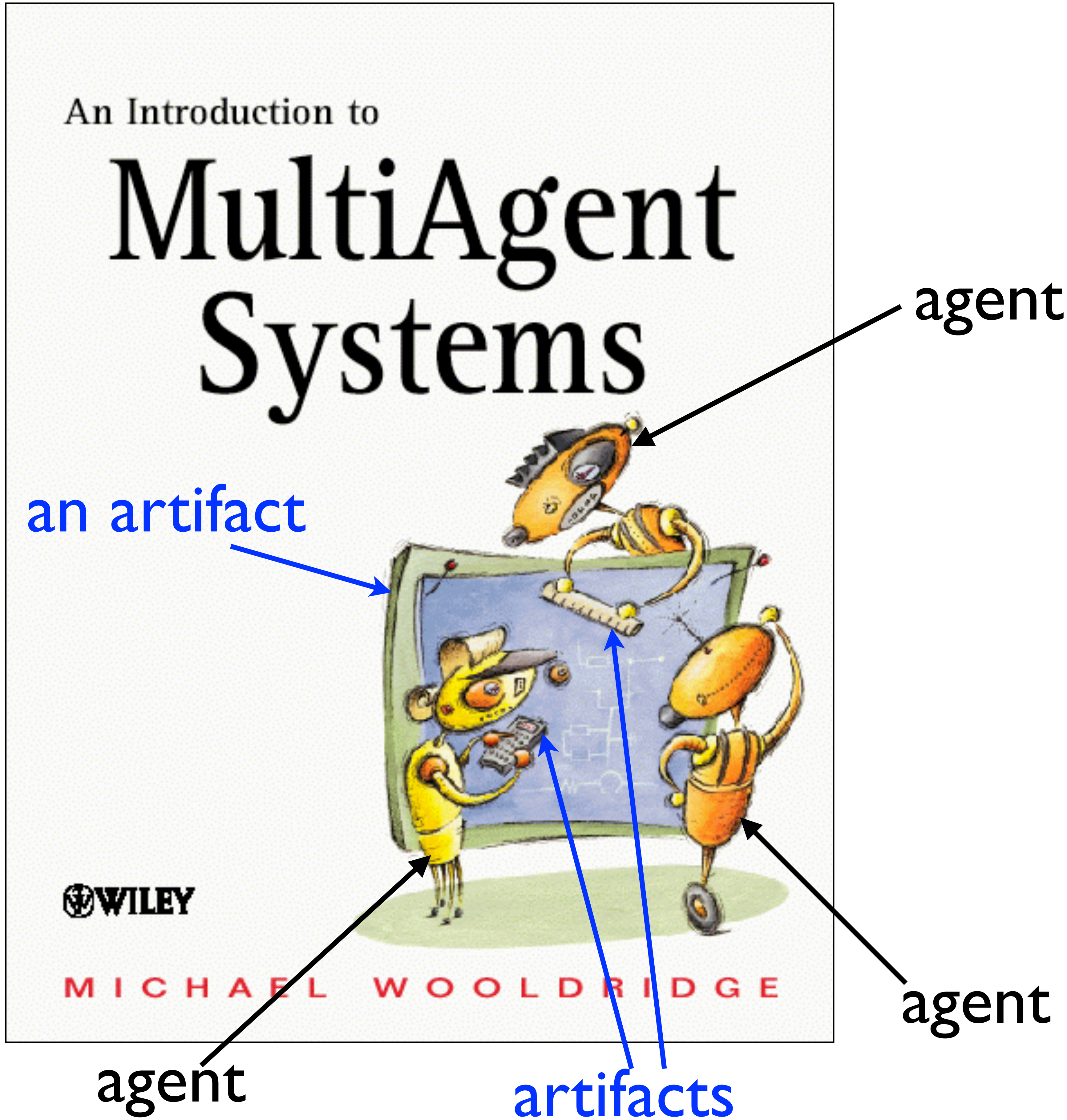
agent

agent

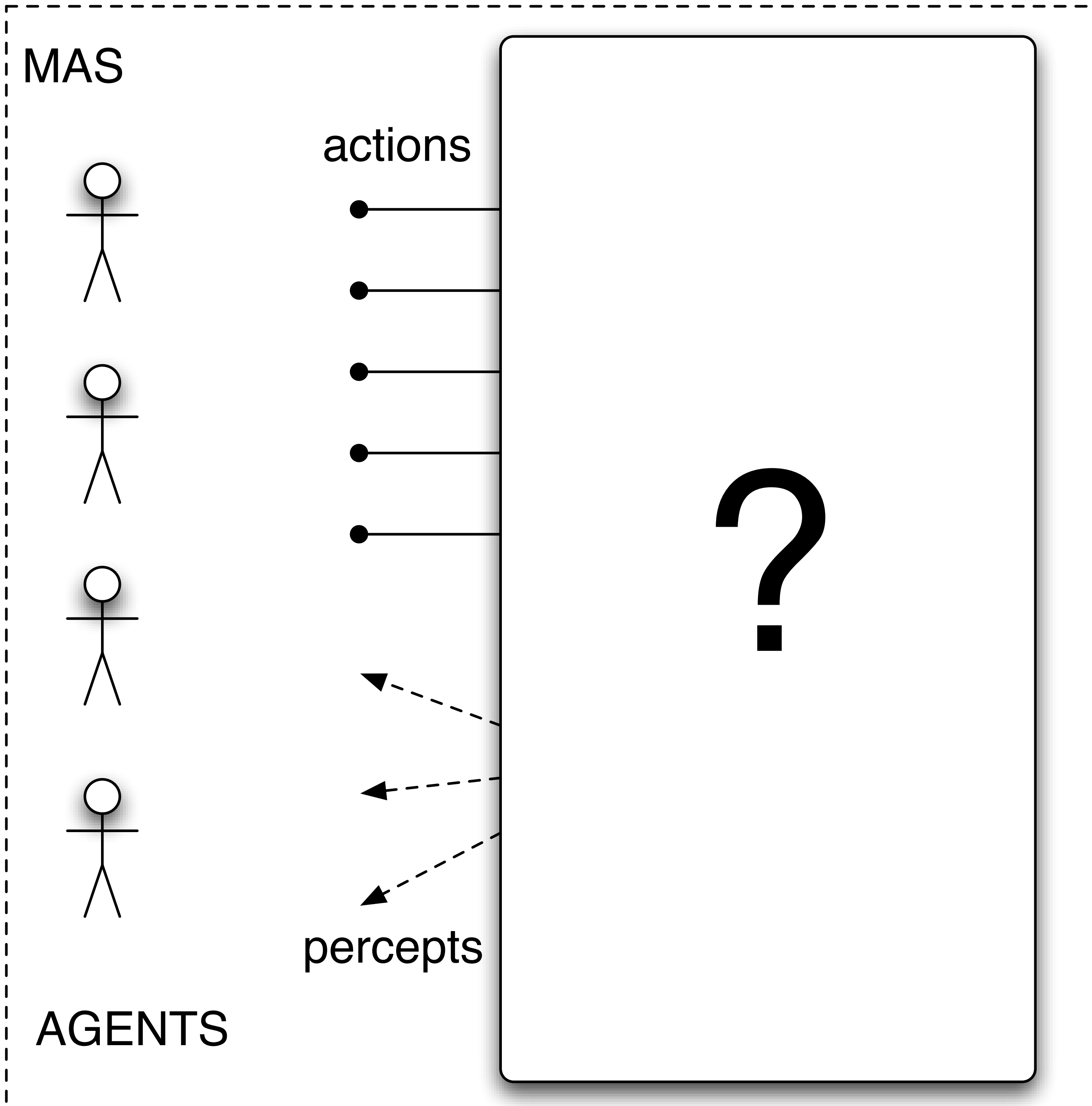
agent

ARTIFACTS
ARE IN THE
MAINSTREAM

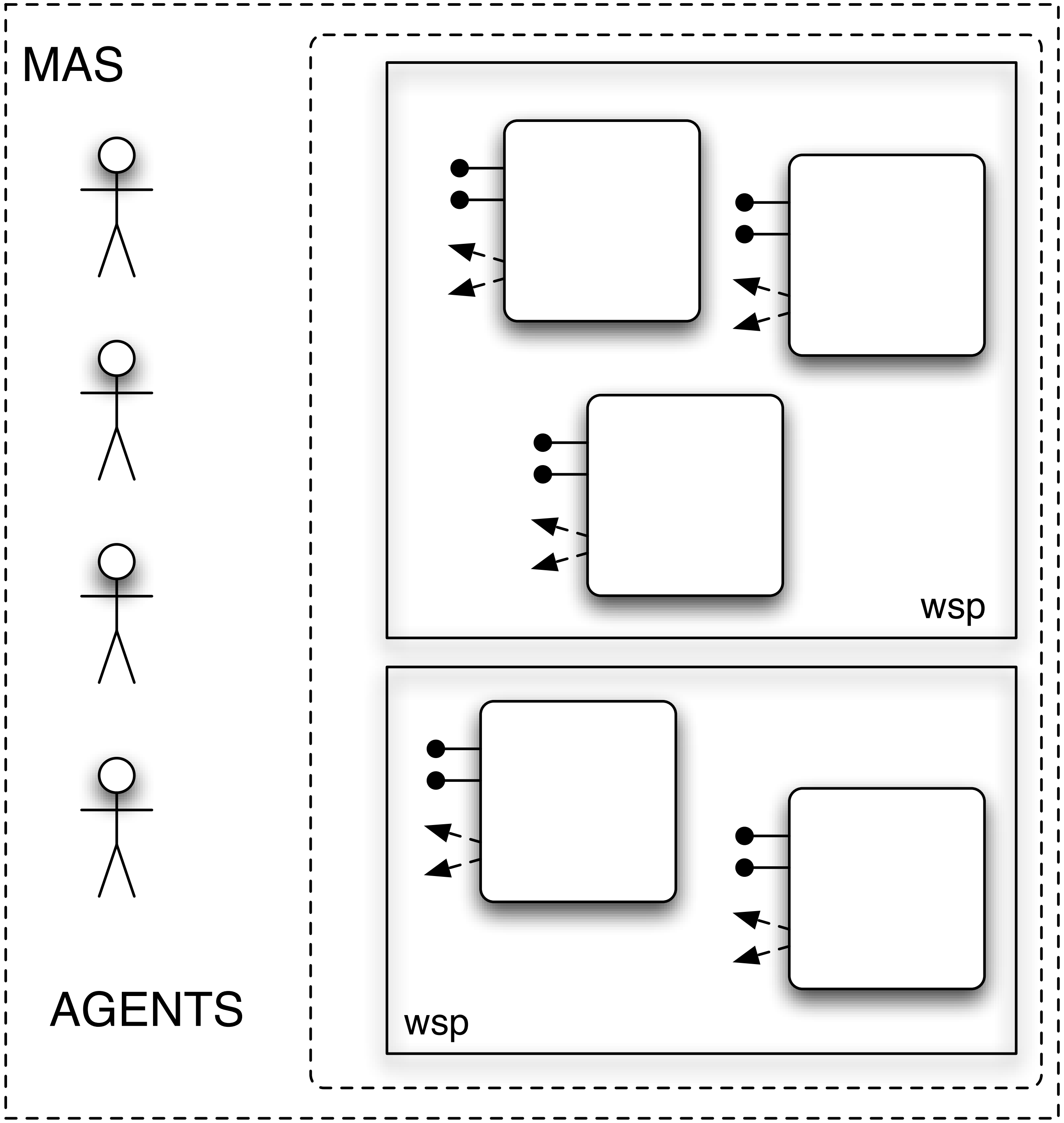
...not really, actually...



WORK ENVIRONMENT IN A&A

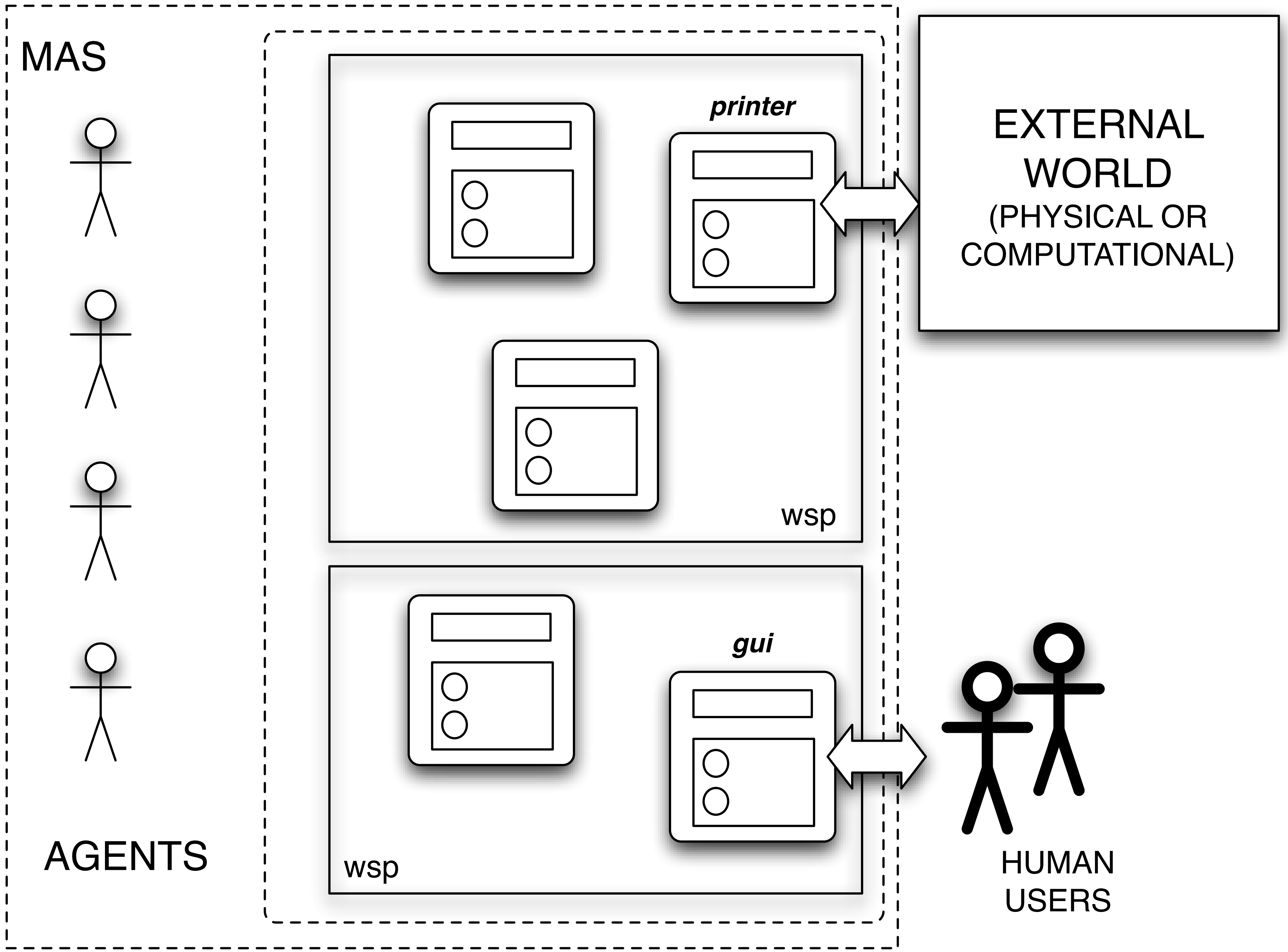


WORK ENVIRONMENT IN A&A

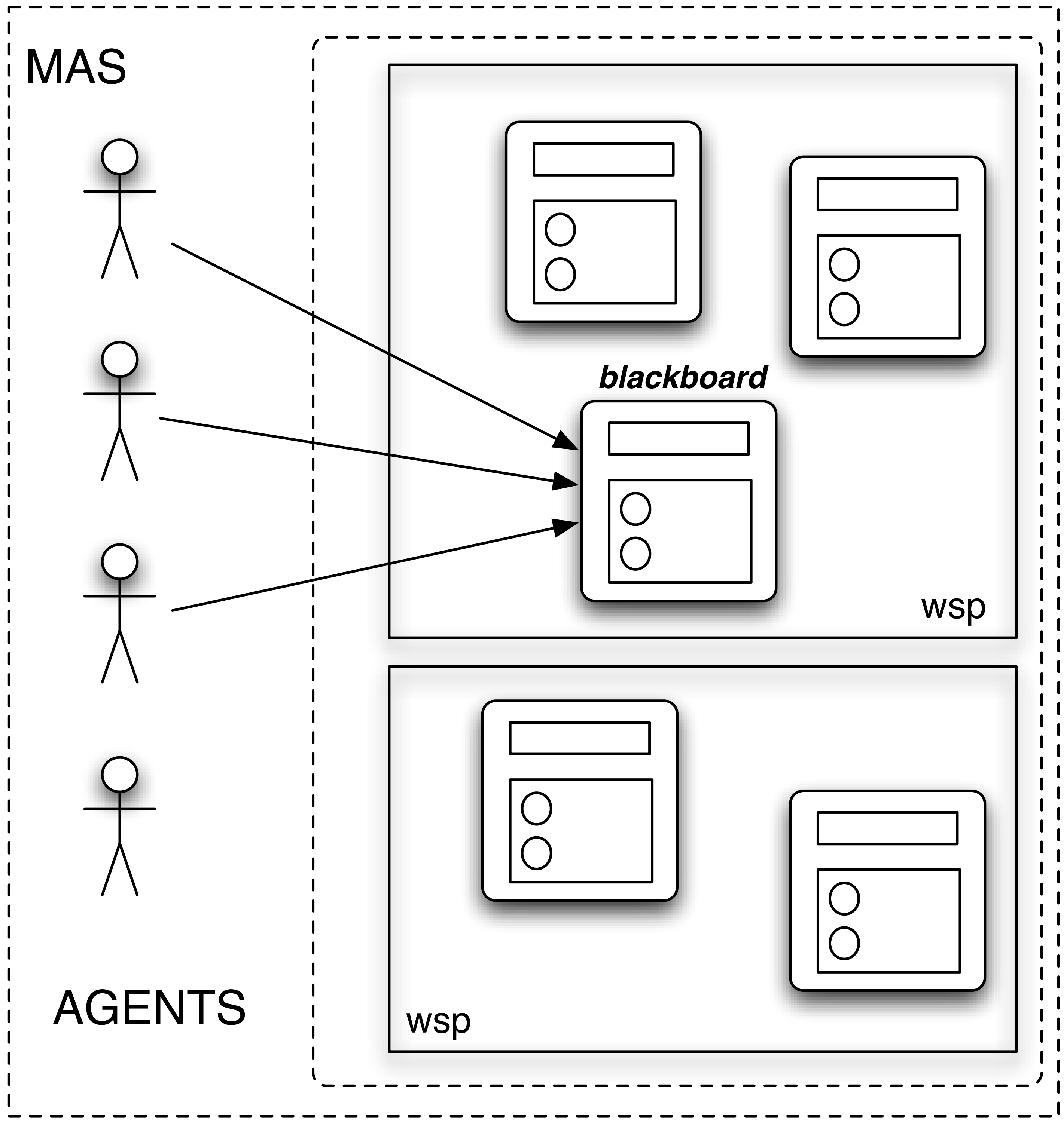


- Abstraction
  - encapsulation
  - information hiding
- Modularization
  - extendibility
  - reuse

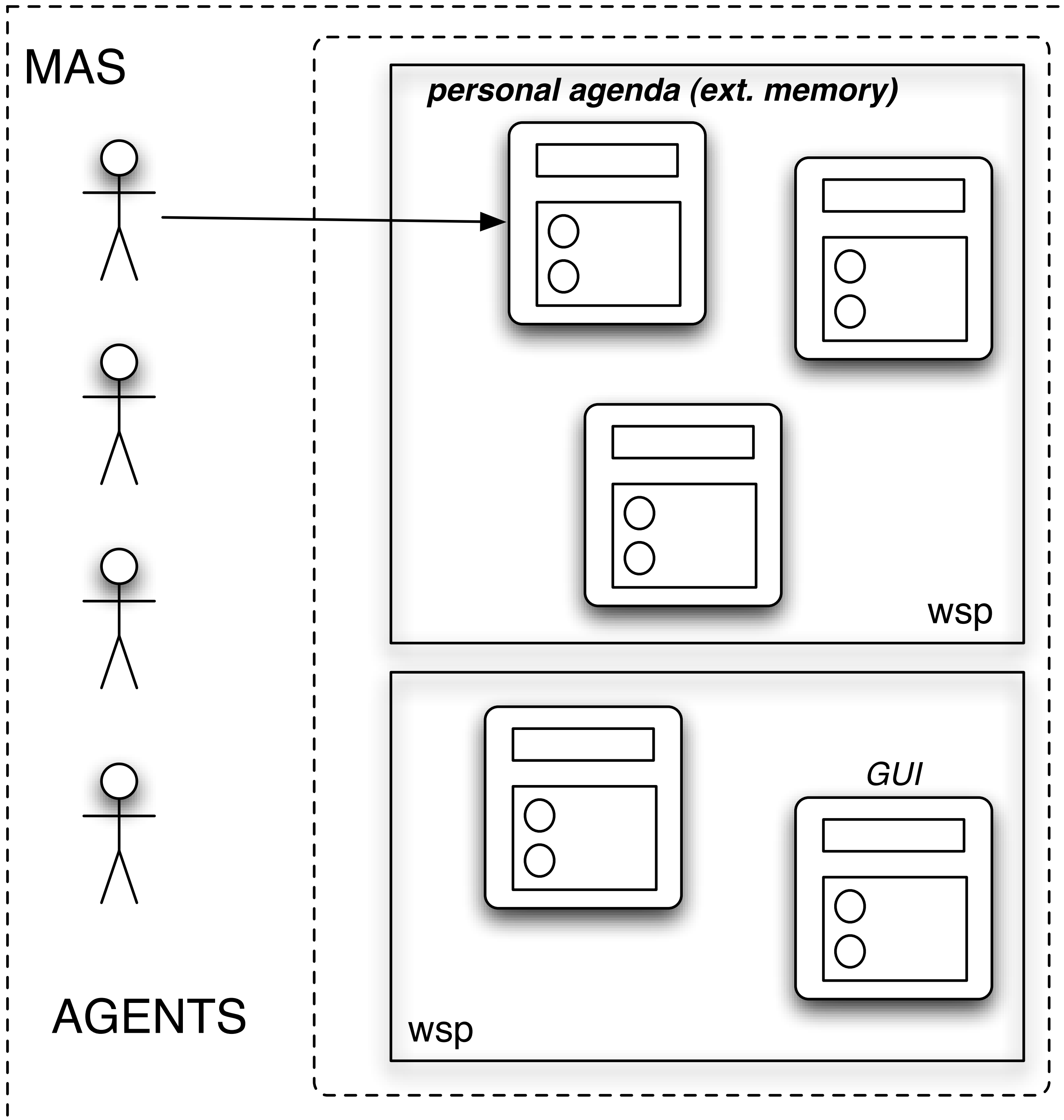
WORK ENVIRONMENT IN A&A



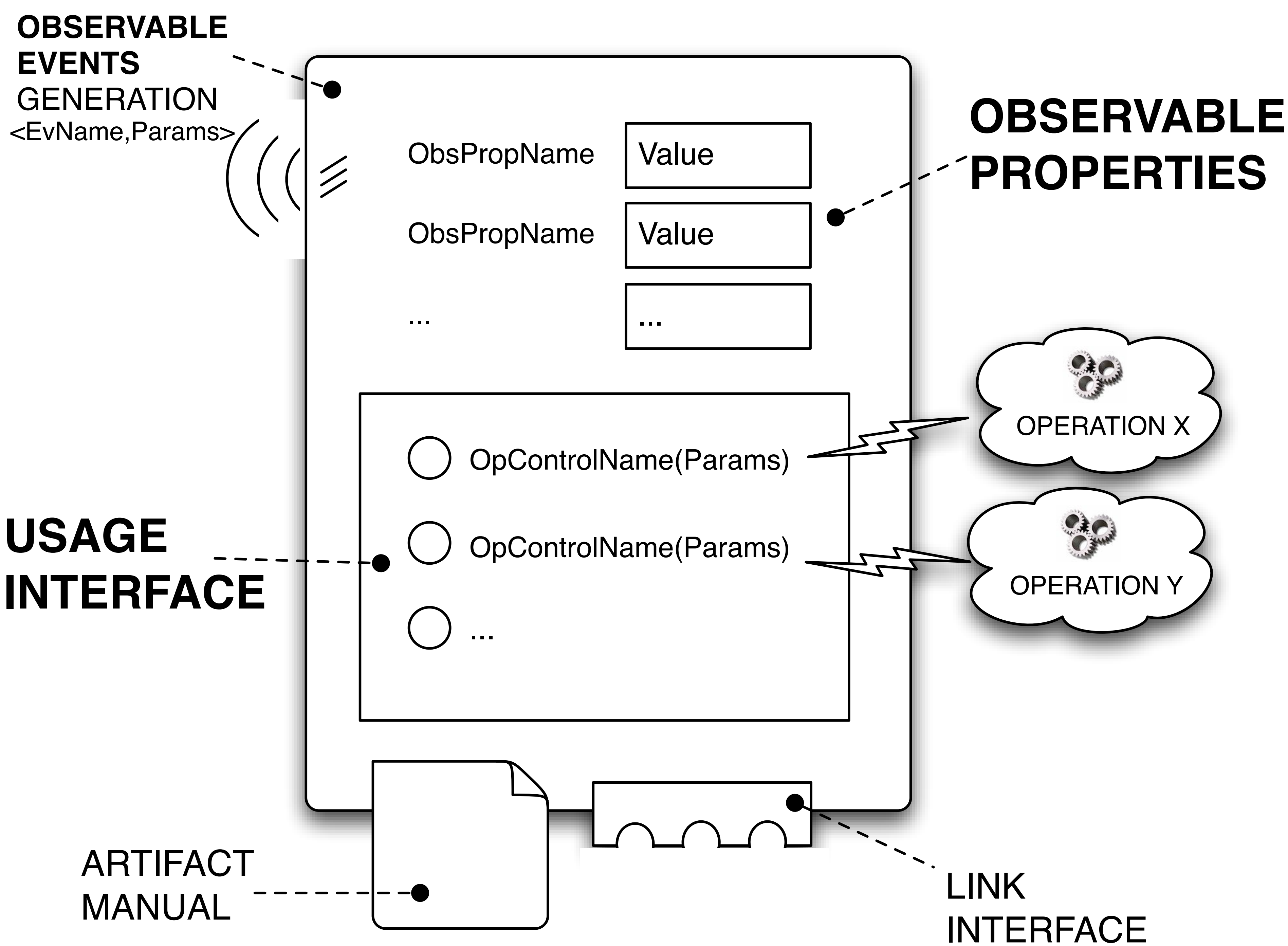
WORK ENVIRONMENT IN A&A



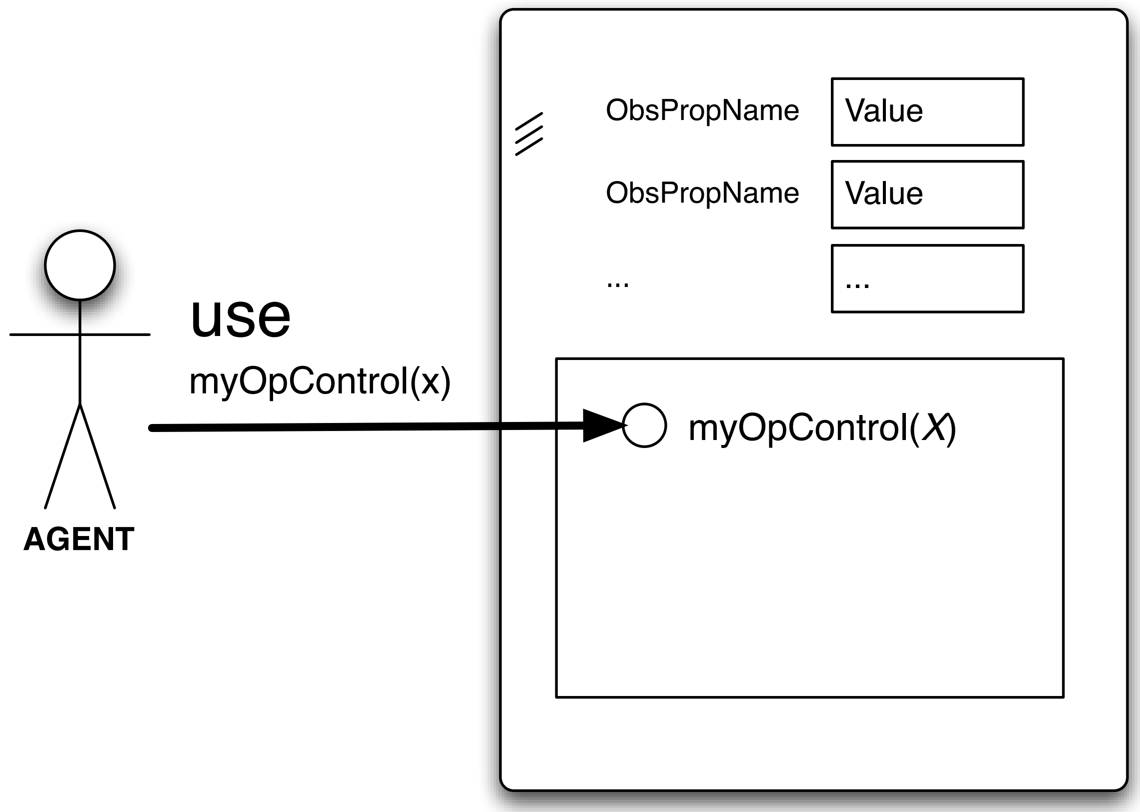
WORK ENVIRONMENT IN A&A



ARTIFACT COMPUTATIONAL MODEL
- “COFFEE MACHINE METAPHOR” -

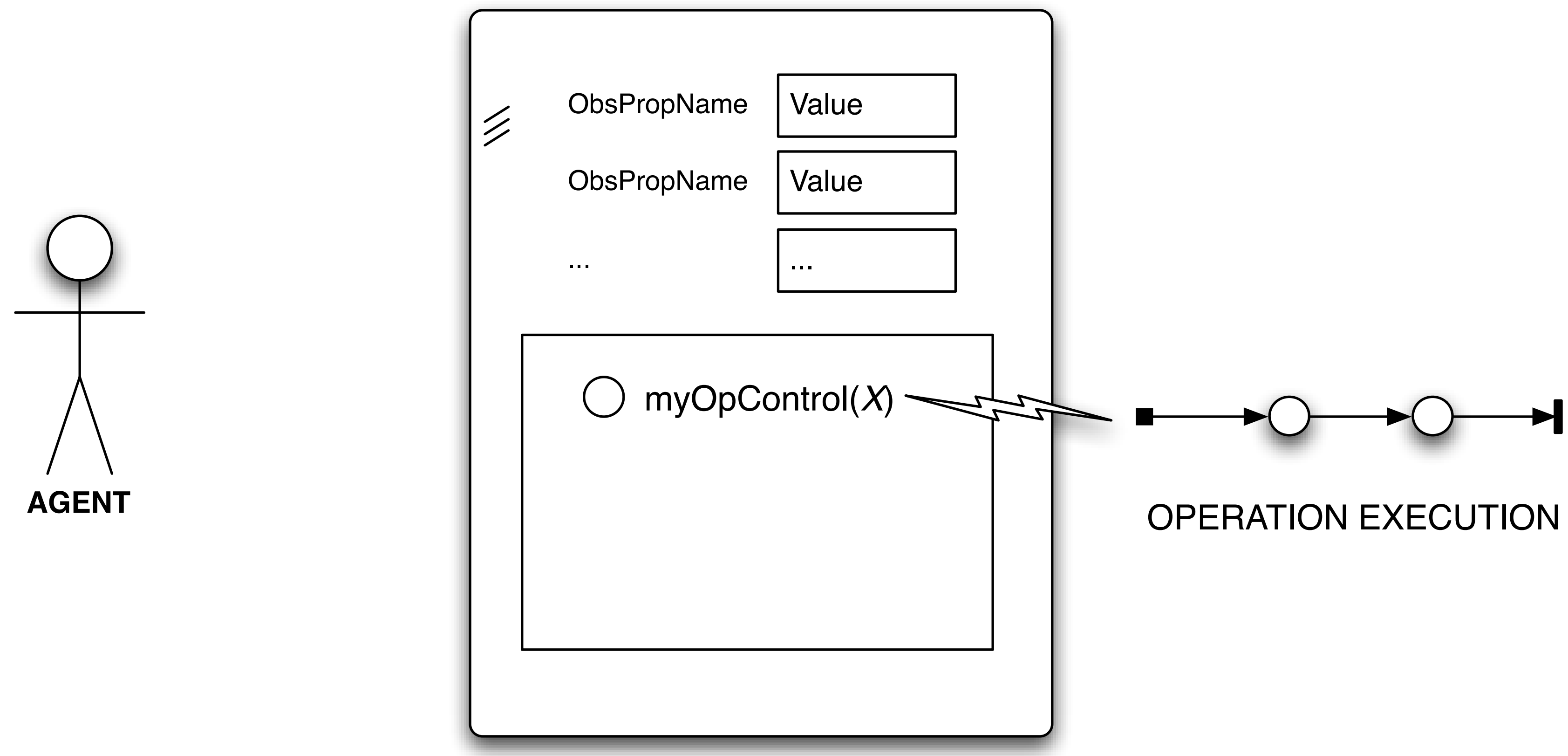


INTERACTION MODEL: **USE** & OBSERVATION



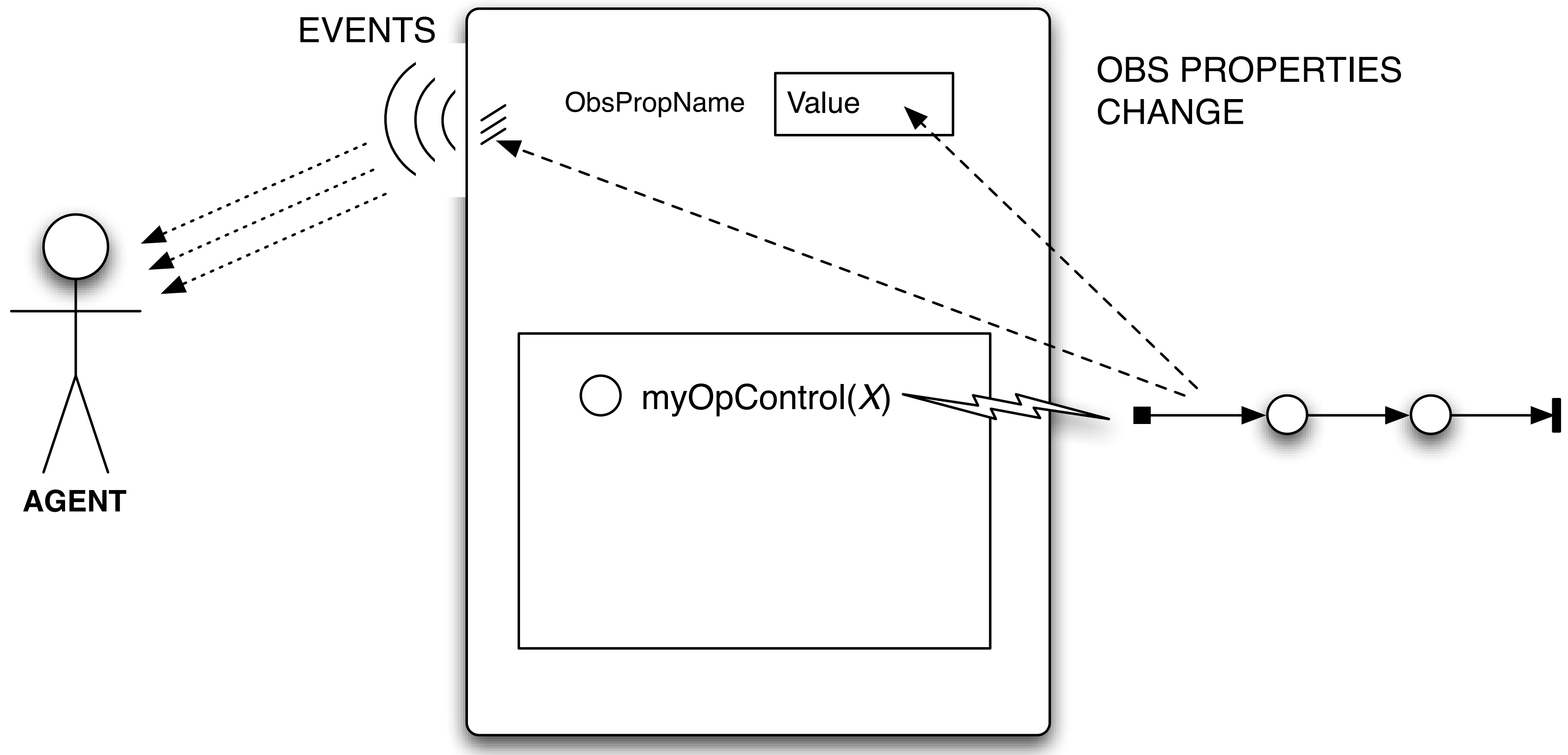
- `use` action
  - acting on op. controls to trigger op execution
  - **synchronisation point** with artifact time/state

INTERACTION MODEL: **USE** & OBSERVATION



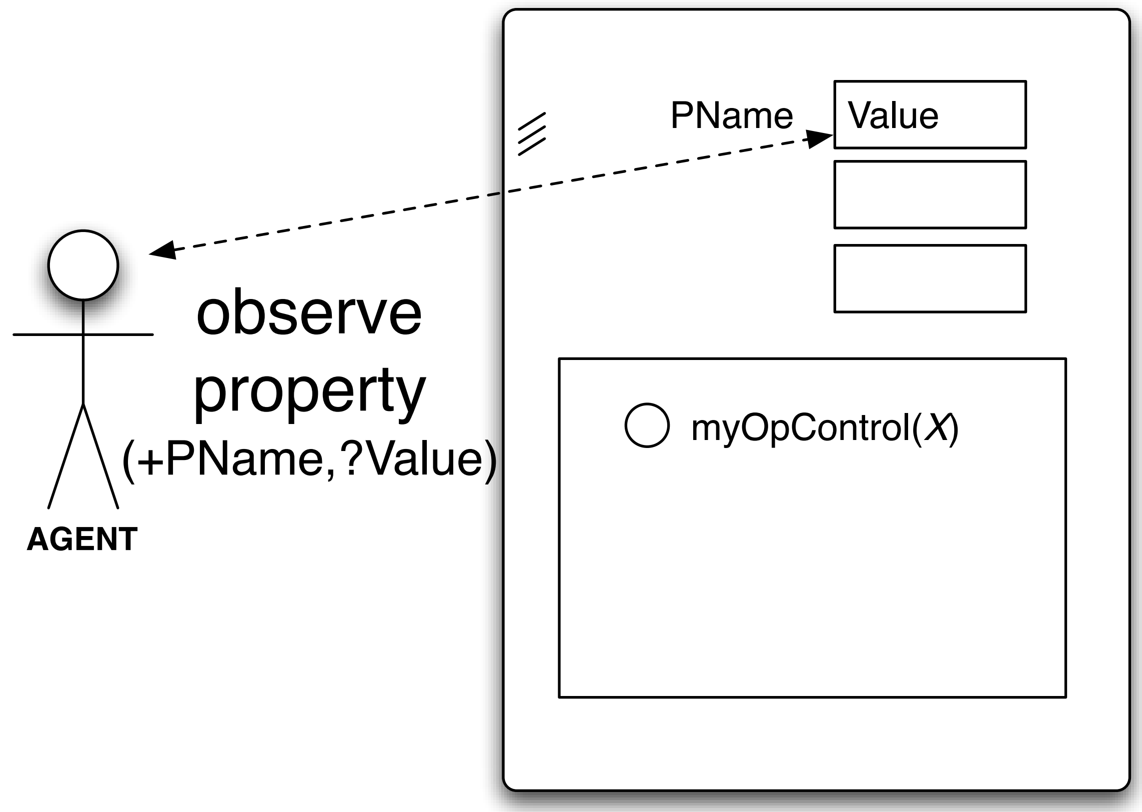
- artifact operation execution
  - asynchronous wrt agent
  - possibly a process structured in multiple atomic steps

INTERACTION MODEL: **USE** & OBSERVATION



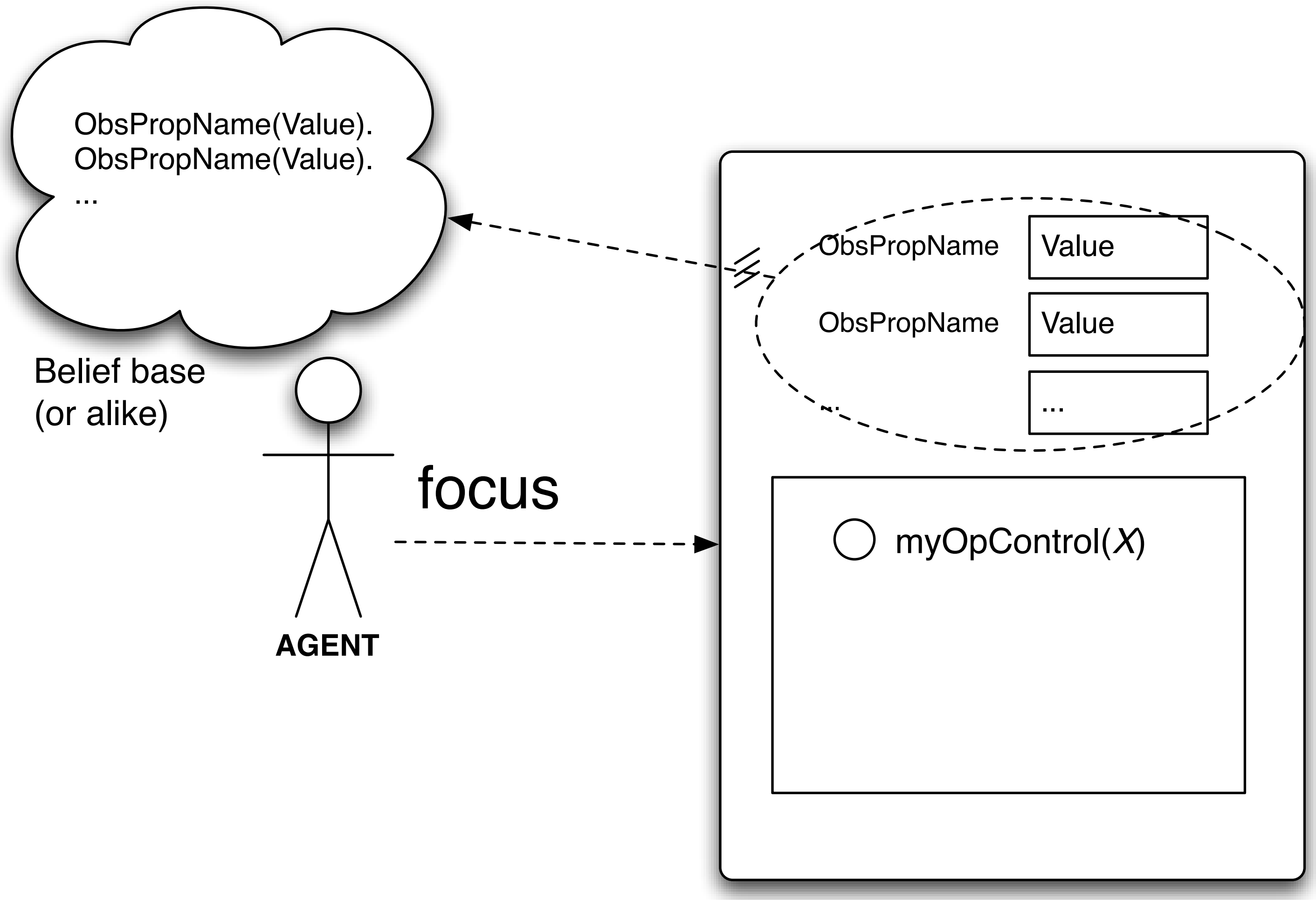
- observable effects
  - observable events & changes in obs property
  - perceived by agents either as (external) events

INTERACTION MODEL: USE & **OBSERVATION**



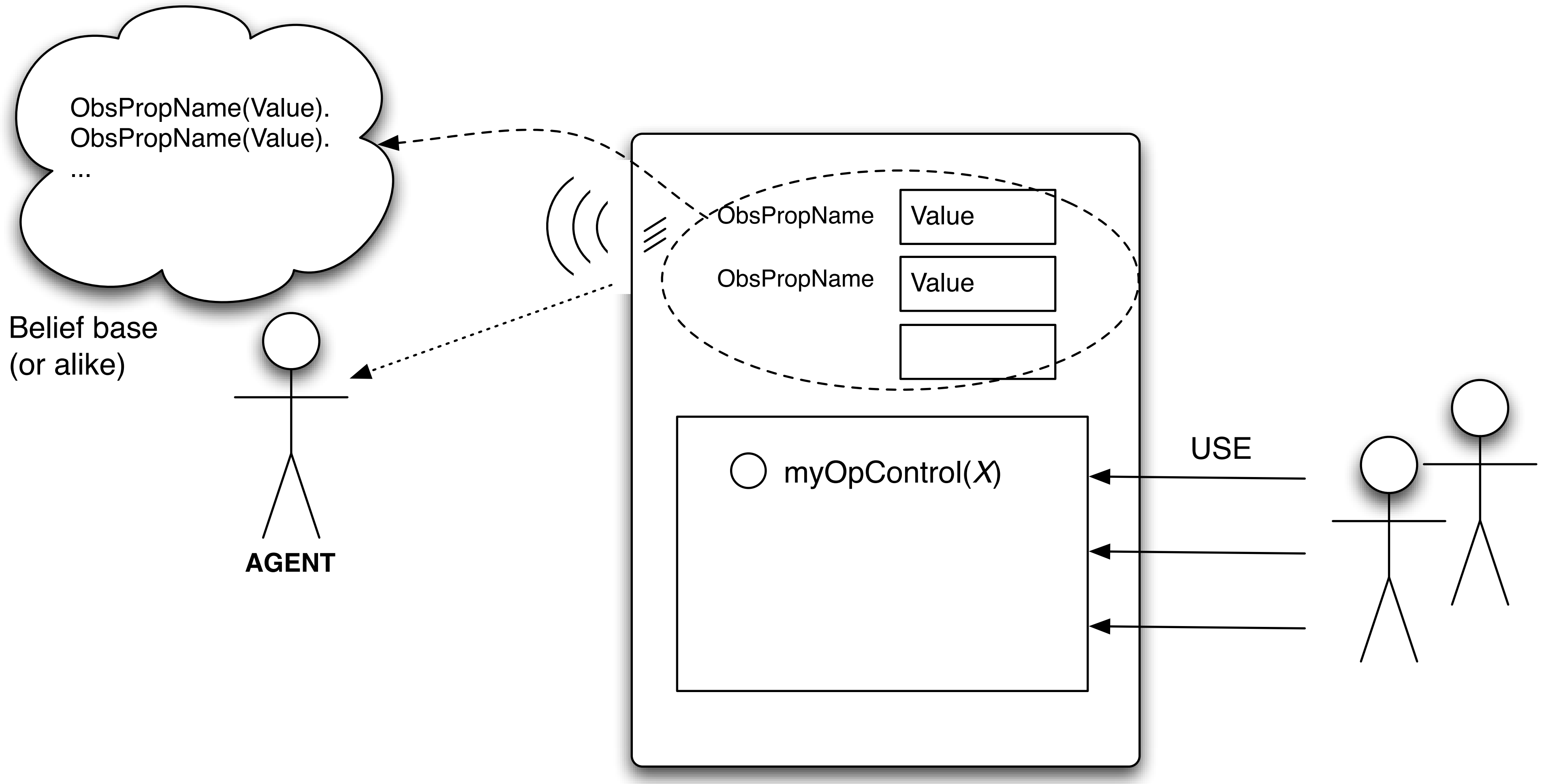
- `observeProperty` action
  - value of an obs. property as action feedback
  - *no interaction*

INTERACTION MODEL: USE & **OBSERVATION**



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

INTERACTION MODEL: USE & **OBSERVATION**



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

ARTIFACT COMPUTATIONAL MODEL HIGHLIGHTS

ARTIFACT COMPUTATIONAL 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

ARTIFACT COMPUTATIONAL 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

ARTIFACT COMPUTATIONAL 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

EXAMPLES OF ARTIFACTS

- Common tools and resources in MAS
  - blackboards, tuple centres, synchronisers,...
  - maps, calendars, shared agenda,...
  - data-base, shared knowledge base,...
  - hardware res. wrappers
  - GUI artifacts
  - Web Services
  - ...

  - principled way to design / program / use them inside MAS

CArtAgO

CArtAgO

- CArtAgO computational model + platform / infrastructure
  - concrete computational & programming model for artifacts
    - API available in Java
    - to be integrated with agent programming platforms
  - runtime environment for executing (possibly distributed) artifact-based environments
  - Java-based programming model for defining artifacts

CArtAgO

- CArtAgO computational model + platform / infrastructure
  - concrete computational & programming model for artifacts
    - API available in Java
    - to be integrated with agent programming platforms
  - runtime environment for executing (possibly distributed) artifact-based environments
  - Java-based programming model for defining artifacts
- 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

CArtAgO

- CArtAgO computational model + platform / infrastructure
  - concrete computational & programming model for artifacts
    - API available in Java
    - to be integrated with agent programming platforms
  - runtime environment for executing (possibly distributed) artifact-based environments
  - Java-based programming model for defining artifacts
- 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

...AND FRIENDS

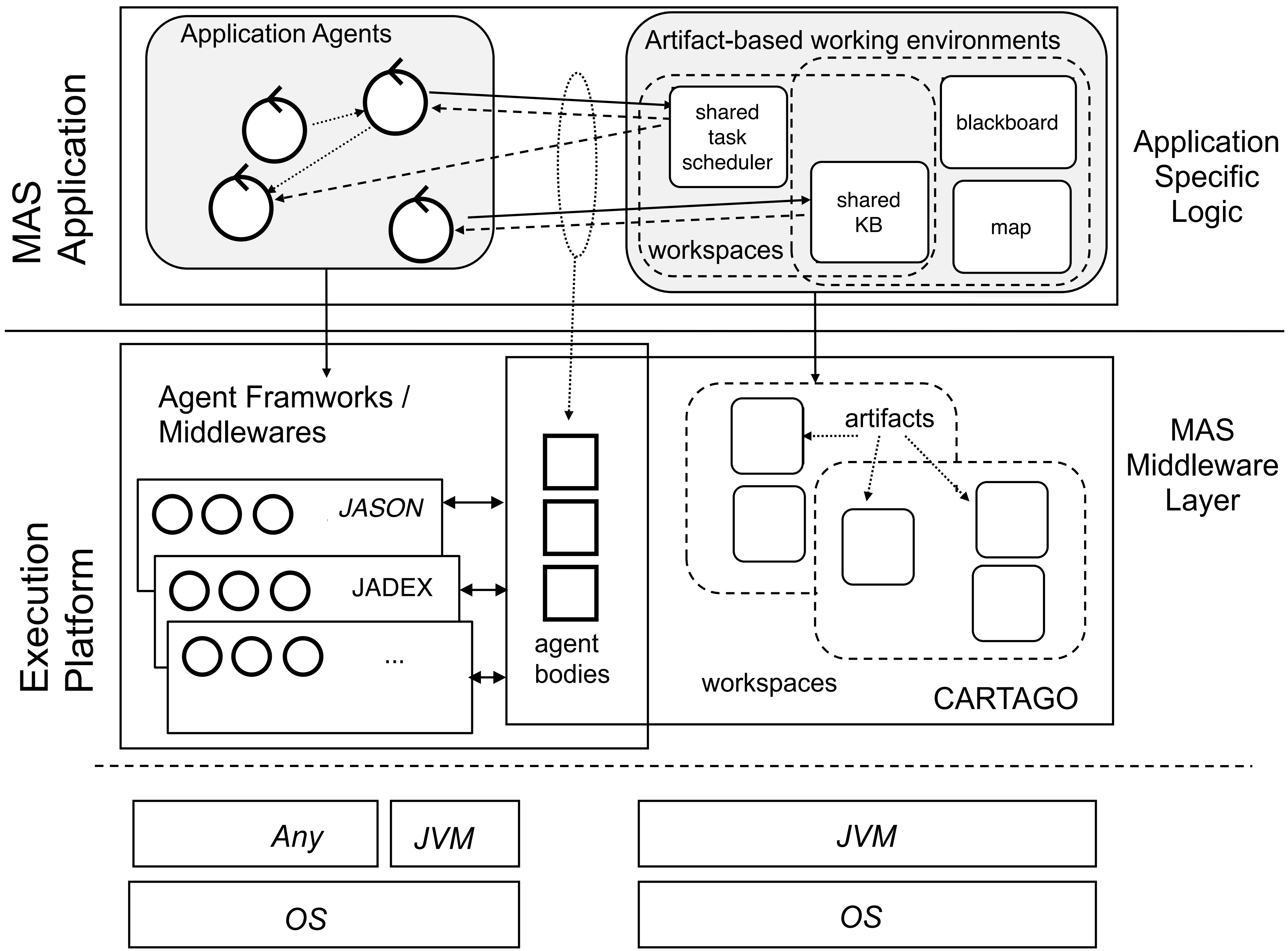
...AND FRIENDS

- Integration with existing agent languages & platforms
  - available bridges: ***Jason***, **Jadex**, simpA
    - ongoing: **2APL**

...AND FRIENDS

- Integration with existing agent languages & platforms
  - available bridges: ***Jason***, **Jadex**, simpA
    - ongoing: **2APL**
- Outcome
  - developing open and heterogenous MAS
  - different perspective on *interoperability*
    - sharing and working in a common work environment
    - common data-model based on Object-Oriented or XML-based data structures

CArtAgO ARCHITECTURE



DEFINING ARTIFACTS IN CArtAgO

DEFINING ARTIFACTS IN CArtAgO

- Single class extending `alice.cartago.Artifact`

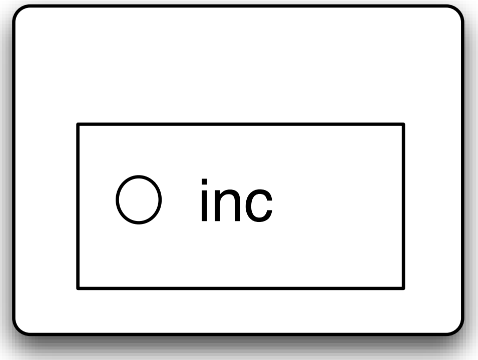
DEFINING ARTIFACTS IN CArtAgO

- Single class extending `alice.cartago.Artifact`
- Specifying the operations
  - atomic: **`@OPERATION`** methods
    - name+params -> usage interface control
    - no return value
  - structured
    - linear composition of atomic operation steps composed dynamically
  - `init` operation
    - automatically executed when the artifact is created

DEFINING ARTIFACTS IN CArtAgO

- Single class extending `alice.cartago.Artifact`
- Specifying the operations
  - atomic: **`@OPERATION`** methods
    - name+params -> usage interface control
    - no return value
  - structured
    - linear composition of atomic operation steps composed dynamically
  - `init` operation
    - automatically executed when the artifact is created
- Specifying artifact state
  - instance fields of the class

SIMPLE EXAMPLE #1



USAGE INTERFACE:

inc: [op_exec_completed]

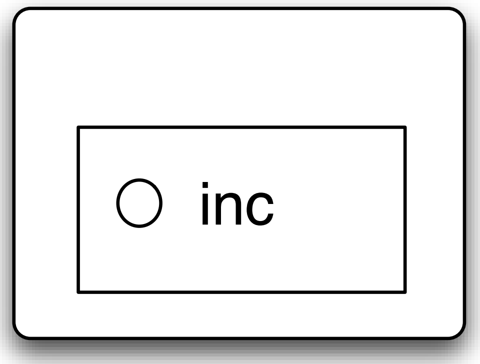
```
public class Count extends Artifact {
  int count;

  @OPERATION void init(){
    count = 0;
  }

  @OPERATION void inc(){
     count++;
  }
}
```

ARTIFACT OBSERVABLE EVENTS

- Observable events
  - generated by **signal** primitive
  - represented as labelled tuples
    - event_name(Arg0,Arg1,...)

- Automatically made observable to...
  - the agent who executed the operation
  - all the agents observing the artifact

SIMPLE EXAMPLE #2



USAGE INTERFACE:

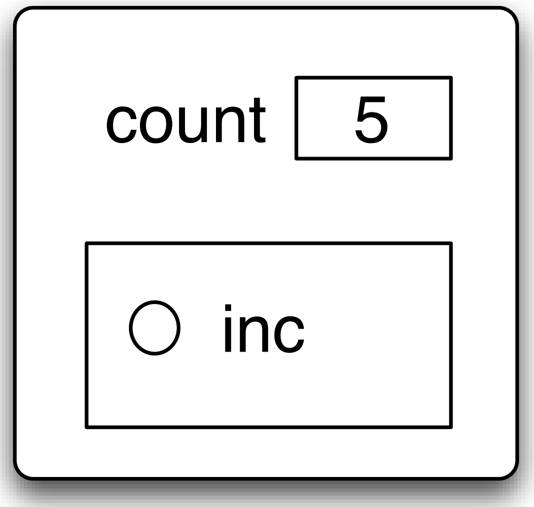
inc: [new_count_value,
op_exec_completed]

```
public class Count extends Artifact {
  int count;

  @OPERATION void init(){
    count = 0;
  }

  @OPERATION void inc(){
     count++;
     signal("new_count_value", count);
   }
}
```

ARTIFACT OBSERVABLE PROPERTIES

- Observable properties
  - declared by **defineObsProperty** primitive
    - characterized by a property name and a property value
  - internal primitives to read / update property value
    - **updateObsProperty**
    - **getObsProperty**
- Automatically made observable to all the agents observing the artifact

SIMPLE EXAMPLE #3



OBSERVABLE PROPERTIES:

count: int

USAGE INTERFACE:

inc: [op_exec_completed]

```
public class Count extends Artifact {

  @OPERATION void init(){
    defineObsProperty("count", 0);
  }

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

OPERATION CONTROLS WITH GUARDS

- Specifying *guards* in operation controls
  - guards as boolean functions defining a condition over artifact (observable) state

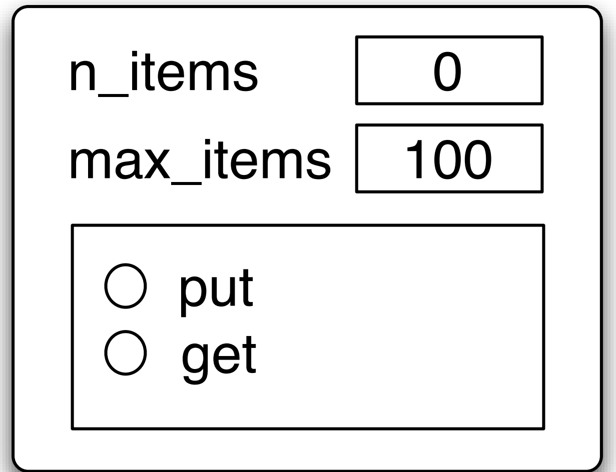
```
@OPERATION(guard="myGuard") void myOp(Param p){
  ...
}

@GUARD boolean myGuard(Param p){
   /* evaluating the condition */
}
```

  - the operation control is enabled if the condition is evaluated to true
    - otherwise the operation control is disabled
- use actions acting upon disabled controls are suspended
  - blocking behaviour for the use action

EXAMPLE: BOUNDED-BUFFER FOR P/C SCENARIOS

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

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

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

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

EXAMPLE: BOUNDED-BUFFER FOR P/C SCENARIOS



OBSERVABLE PROPERTIES:

n_items: int+
max_items: int

Invariants:
n_items <= max_items

USAGE INTERFACE:

put(*item*:Item) / (n_items < max_items):
[op_exec_completed]

get / (n_items >= 0) :
[*new_item*(item:Item), op_exec_completed]

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

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

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

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

MORE ON ARTIFACTS

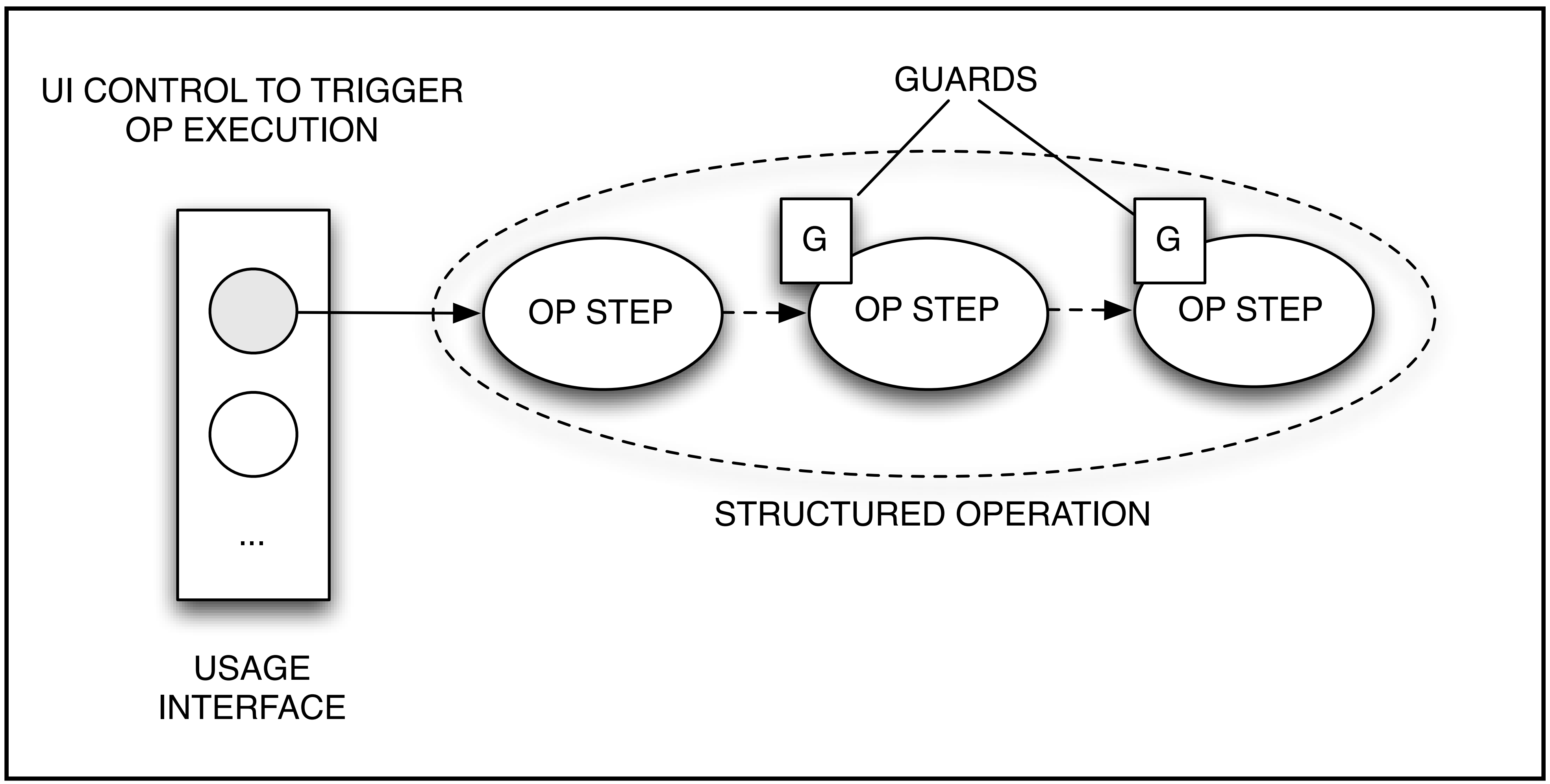
- Structured operations
  - specifying operations composed by chains of atomic operation steps
  - to support the concurrent execution of multiple operations on the same artifact
    - by interleaving steps
- Linkability
  - dynamically composing / linking multiple artifacts together
- Artifact *manual*
  - machine-readable description of artifact functionality and operating instructions

STRUCTURED OPERATIONS

- Complex operations as chains of guarded atomic operation step execution
  - `@OPSTEP` methods

STRUCTURED OPERATIONS

- Complex operations as chains of guarded atomic operation step execution
  - `@OPSTEP` methods



STRUCTURED OPERATIONS

- Complex operations as chains of guarded atomic operation step execution
  - `@OPSTEP` methods

- Guards
  - boolean expression over the artifact state
    - once enabled, the operation step is executed as soon as the guard is evaluated to true

> Multiple structured operations can be executed concurrently on the same artifact by interleaving their steps
  - with only one step executed at a time

EXAMPLE: A (CENTRALIZED) TUPLE SPACE

```
public class SimpleTupleSpace extends Artifact {
  TupleSet tset;

  @OPERATION void init(){ tset = new TupleSet(); }

  @OPERATION void out(Tuple t){ tset.add(t);}

  @OPERATION void in(TupleTemplate tt){
    Tuple t = tset.removeMatching(tt);
    if (t!=null){
      signal("tuple",t);
    } else {
      nextStep("completeIN",tt);
    }
  }
  @OPSTEP(guard="foundMatch") void completeIN(TupleTemplate tt){
    Tuple t = tset.removeMatching(tt);
    signal("tuple",t);
  }
  @GUARD boolean foundMatch(TupleTemplate tt){
    return tset.hasTupleMatching(tt);
  }

  @OPERATION void inp(TupleTemplate tt){
    Tuple t = tset.removeMatching(tt);
    if (t!=null){
      signal("tuple_available",t);
    } else {
      signal("tuple_not_available");
    }
  }
  @OPERATION void rd(TupleTemplate tt){...}
  @OPERATION void rdp(TupleTemplate tt){...}
}
```

ON THE AGENT SIDE: AGENT ACTIONS

- 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,OpCntrName(Params),+Sensor,+Timeout,+Filter) **sense**(Sensor,?Perception,+Filter,+Timeout)
artifact pure 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

use
sense
focus
stopFocus

grab
release
```

+

basic set of artifacts available in each workspace

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

implementing non primitive actions:

`makeArtifact` => use `factory`

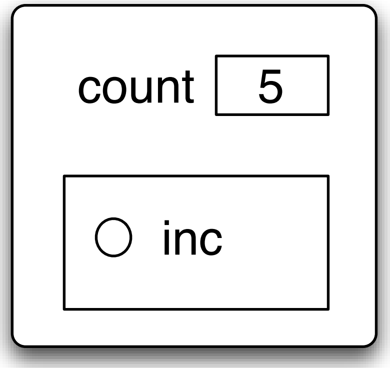
`lookupArtifact` => use `registry`

JASON API EXAMPLE

- C4Jason bridge
  - enabling Jason agents to work in CArtAgO workspaces
  - alice.c4jason.CEnvStandalone / alice.c4jason.CEnv Jason environment classes (for standalone / distributed artifact based environments)
  - alice.c4jason.CAgentArch as agent architecture class
- `cartago.*` internal actions library
  - `cartago.joinWSP / cartago.quitWSP`
  - `cartago.use / cartago.sense`
  - `cartago.focus / cartago.stopFocus /cartago.observeProperty`
  - `cartago.makeArtifact / cartago.lookupArtifact`
  - `...`
- Included also basic set of internal actions to manipulate Java objects as basic data type
  - `cartago.newObject / cartago.callObj`

A FIRST SIMPLE EXAMPLE

- Counter



OBSERVABLE PROPERTIES:

count: int

USAGE INTERFACE:

inc: [op_exec_completed]

```
MAS mas1 {

  environment:
    alice.c4jason.CEnvStandalone

  agents:
    observer agentArchClass alice.c4jason.CAgentArch;
    user agentArchClass alice.c4jason.CAgentArch #2;
}
```

```
package test;

public class Counter1 extends Artifact {
  @OPERATION void init(){
    defineObsProperty("count",0);
  }

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

```
// observer
!observe.

+!observe : true
    <-  cartago.makeArtifact("my_counter","test.Counter1", Count);
        cartago.focus(Count).

+count(V) : true
    <-  cartago.use(console,println("current count observed: ",V)).
```

```
// user
!use_count.

+!use_count : true
  <- ?counter_to_use(Counter) ;
     +cycle(0) ;
     !use_count(Counter).

+?counter_to_use(Counter) : true
  <- cartago.lookupArtifact("my_counter",Counter).

-?counter_to_use(Counter) : true
  <- .wait(100);
     ?counter_to_use(Counter).

+!use_count(C) : cycle(N) & N < 10
  <- -cycle(N);
     cartago.use(C,inc,mySensor0);
     cartago.sense(mySensor0,"operation_completed");
     !have_a_rest ;
     +cycle(N+1) ;
     !use_count(C).

+!use_count(C) : cycle(10).

+!have_a_rest : true
  <- .wait(10).
```

BOUNDED-BUFFER EXAMPLE: PRODUCERS & CONSUMERS IN JASON

PRODUCERS

```
!produce.

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

+!produceItems : true <-
  ?nextItemToProduce(Item);
  cartago.use(myBuffer,put(Item),5000);
  !produceItems.

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

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

CONSUMERS

```
!consume.

+!consume: true <-
  ?bufferToUse(Buffer);
  .print("Going to use ",Buffer);
  !consumeItems.

+!consumeItems : true <-
  cartago.use(myBuffer,get,s0,5000);
  cartago.sense(s0,new_item(Item),5000);
  !consumeItem(Item);
  !consumeItems.

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

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

EXAMPLE: GOOD OLD DINING PHILOSOPHERS

- Dining philosopher problem
  - N philosophers sharing and using N forks
    - philosophers repeatedly thinking and eating
    - to eat philosophers need 2 forks
    - a fork can be used by 1 philosopher at a time
  - avoiding interferences, deadlock, starvation
- Two classic solutions
  - centralized coordination
    - single Table coordination artifact
  - decentralized coordination
    - N Fork resource artifacts
    - proper usage protocol

DINING PHILO: SOLUTION #1

- Two basic type of artifacts
  - `Table` artifact coordination artifact
    - coordinating access to shared resources
  - `ForkDispenser` artifact
    - to allocate at the beginning forks number to philosophers
- Strategy for philosophers
  - after obtaining two fork numbers by interacting with the ForkDispenser, each philosopher agent repeatedly use the table artifact to get the forks and to release them after eating

DINING PHILO SOLUTION #1: THE MAS

```
MAS philosophers {
  environment:
   alice.c4jason.CEnvStandalone

  agents:
    waiter waiter.asl agentArchClass alice.c4jason.CAgentArch;
    philo philo.asl agentArchClass alice.c4jason.CAgentArch #5;
}
```

DINING PHILO SOLUTION #1: ARTIFACTS

```
public class ForkDispenser extends Artifact {

  private int nForks;
  private int forkIndex = 0;

  @OPERATION void init(int nforks){
    nForks = nforks;
    forkIndex = 0;
  }

  @OPERATION void getForkAssignment(){
    int next = (forkIndex+1)%nForks;
    signal("fork_assignment",forkIndex,next);
    forkIndex = next;
  }
}
```

```
public class Table extends Artifact {

  private boolean[] forks;

  @OPERATION void init(int nforks){
    forks = new boolean[nforks];
    for (int i = 0; i<forks.length; i++){
      forks[i]=true;
    }
  }

  @OPERATION(guard = "forksAvailable")
  void getForks(int firstFork, int secondFork){
    forks[firstFork] = forks[secondFork] = false;
    signal("forks_acquired");
  }

  @GUARD boolean forksAvailable(int firstFork,int secondFork){
    return forks[firstFork] && forks[secondFork];
  }

  @OPERATION void releaseForks(int firstFork, int secondFork){
    forks[firstFork] = forks[secondFork] = true;
  }

}
```

DINING PHILO SOLUTION #1: WAITER AGENT

```
!prepare_table.

+!prepare_table : true
  <-  cartago.use(console,println("Preparing the environment..."));
      cartago.makeArtifact("fork_disp","philo.ForkDispenser",[3]) ;
      cartago.makeArtifact("table","philo.Table",[3]) ;
      cartago.use(console,println("The environment is ready.")).
```

DINING PHILO SOLUTION #1: PHILOSOPHER AGENT

```
// initial goal
!go.

+!go
  <-  !discover_table(Table);
      +table(Table);
      !get_fork_assignment(F1,F2);
      +my_forks(F1,F2);
      !!do_my_job.

+!do_my_job
  <-  !think;
      !acquire_forks;
      !eat;
      !release_forks;
      !!do_my_job.

+!acquire_forks: my_forks(F1,F2) & table(T)
  <-  cartago.use(T,getForks(F1,F2),s0);
      cartago.sense(s0,forks_acquired).

+!release_forks: my_forks(F1,F2) & table(T)
  <-  cartago.use(T,releaseForks(F1,F2)).
```

```
+!think
  <-  .my_name(Name);
      cartago.use(console,println(Name," is thinking."));
      .wait(10+20*math.random).

+!eat
  <-  .my_name(Name);
      cartago.use(console,println(Name," is eating."));
      .wait(10+10*math.random).

+!discover_table(Table) : true
  <-  cartago.lookupArtifact("table",Table).
-!discover_table(Table) : true
  <-  .wait(10);
      !discover_table(Table).

+!get_fork_assignment(F1,F2) : true
  <-  cartago.lookupArtifact("fork_disp",FD);
      cartago.use(FD,getForkAssignment,s0);
      cartago.sense(s0,fork_assignment(F1,F2)).
-!get_fork_assignment(F1,F2) : true
  <-  .wait(10);
      !get_fork_assignment(F1,F2).
```

DINING PHILOSOPHERS: SOLUTION #2

- Fully decentralized solution
  - again a `ForkDispenser` artifact
    - to allocate at the beginning forks number to philosophers
  - `Fork` artifact representing the resource to acquire and release
    - 5 instances

DINING PHILO SOLUTION #2: ARTIFACTS

```
public class ForkDispenser extends Artifact {

  private int nForks;
  private int forkIndex = 0;

  @OPERATION void init(int nforks){
    nForks = nforks;
    forkIndex = 0;
  }

  @OPERATION void getForkAssignment(){
    int next = (forkIndex+1)%nForks;
    signal("fork_assignment",forkIndex,next);
    forkIndex = next;
  }
}
```

```
public class Fork extends Artifact {

  @OPERATION void init(int id){
    defineObsProperty("available",true);
    defineObsProperty("id",id);
  }

  @OPERATION(guard="isAvailable") void acquire(){
    updateObsProperty("available", false);
    signal("fork_acquired");
  }

  @GUARD boolean isAvailable(){
    return getObsProperty("available").booleanValue();
  }

  @OPERATION void release(){
    updateObsProperty("available", true);
  }
}
```

DINING PHILO SOLUTION #2: WAITER AGENT

```
!prepare_table.

+!prepare_table : true
  <-  cartago.use(console,println("Preparing the environment..."));
      !create_forks(0,3);
      cartago.makeArtifact("fork_disp","tools.ForkDispenser",[3]) ;
      cartago.use(console,println("The environment is ready.")).

+!create_forks(I,N) : I < N
  <-  .concat("fork",I,FN);
      cartago.makeArtifact(FN,"tools.Fork",[I]);
      !create_forks(I+1,N).

+!create_forks(N,N).
```

DINING PHILO SOLUTION #2: PHILOSOPHER AGENT

```
!go.

+!go
  <-  !get_fork_assignment(F1,F2);
      !sort_forks(F1,F2);
      !!do_my_job.

+!do_my_job
  <-  !think;
      !acquire_forks;
      !eat;
      !release_forks;
      !!do_my_job.

+!acquire_forks : my_forks(F1,F2)
  <-  cartago.use(F1,acquire,s0);
      cartago.use(F2,acquire,s0);
      cartago.sense(s0,fork_acquired);
      cartago.sense(s0,fork_acquired).


+!release_forks : my_forks(F1,F2)
  <-  cartago.use(F1,release);
      cartago.use(F2,release).
```

```
+!think
  <-  .my_name(Name);
      cartago.use(console,println(Name," is thinking."));
      .wait(10+20*math.random).

+!eat
  <-  .my_name(Name);
      cartago.use(console,println(Name," is eating."));
      .wait(10+10*math.random).

+!get_fork_assignment(F1,F2) : true
  <-  cartago.lookupArtifact("fork_disp",FD);
      cartago.use(FD,getForkAssignment,s0);
      cartago.sense(s0,fork_assignment(F1,F2)).

-!get_fork_assignment(F1,F2) : true
  <-  .wait(10);
      !get_fork_assignment(F1,F2).

+!sort_forks(F1,F2) : true
  <-  cartago.observeProperty(F1,id(Id1));
      cartago.observeProperty(F2,id(Id2));
      if (Id1 < Id2){
        +my_forks(F1,F2)
      } {
        +my_forks(F2,F1)
      }.
```

OPEN WORKSPACES & DISTRIBUTION

- Agents can dynamically join and quit workspaces
  - heterogeneous & “remote” agents
    - *Jason*, JADEX, simpA, etc.
  - in Jason MAS
    - `alice.c4jason.CEnv` environment class
- RBAC model for ruling agent access & use of artifacts
  - `security-registry` artifact to keep track of roles and role policies
    - making roles & policies observable and modifiable by agents themselves
- Distribution
  - agents can join and work concurrently in multiple workspaces at a time
  - workspaces can belong to different CArtAgO nodes

PART III

ONGOING WORK & AVAILABLE PROJECTS/THESES

GOAL-DIRECTED USE OF ARTIFACTS

- Objective
  - enabling intelligent agents to dynamically discover and use (and possibly construct) artifacts according to their individual / social objectives
  - *open* systems
    - systems with different kinds of aspects not defined a priory by MAS designers
- Toward fully autono(mic/mous) systems
  - exploring self-organizing systems based on intelligent agents
    - self-CHOP+CA
      - configuring, healing, optimizing, protecting + constructing, adapting

GOAL-DIRECTED USE: SOME CORE ASPECTS

- Defining an “agent-understandable” model & semantics for artifact manual
  - how to specify artifact functionalities
  - how to specify artifact operating instructions
- How to extend agent basic reasoning cycle including reasoning about artifacts
  - relating agent goals and artifact functions
  - relating agent plans and artifact operating instructions and function description
- Reference literature
  - Artificial Intelligent and Distributed AI
  - Semantic Web / Ontologies

EXTERNALIZATION & INTERNALIZATION

- Using artifacts to improve modularisation of agent programs
  - *externalizing* agent functionalities into the environment
    - artifacts as “external modules”
  - using the manual to *internalize* high-level plans to use the artifact
    - minimizing the burden on the agent programming side to explicitely implement low level usage protocols

EXISTING APPLICATIONS/ FRAMEWORKS BASED ON CArtAgO

- CArtAgO-WS
  - basic set of artifacts for building SOA/WS applications
    - interacting with web services
    - implementing web services
- ORA4MAS
  - exploiting artifacts to build MAS *organisational* infrastructure

CArtAgO 2.0

- Revisiting use action / operation mapping and semantics
  - use-action semantics directly mapped onto executed-operation semantics
  - introduction of action feedback parameters as output operation parameters
- Simplifying perception & observation
  - no more sensors
  - revisiting focus semantics
- Simplifying artifact programming API
  - no more operation steps

TUPLE SPACE REVISITED

```
public class SimpleTupleSpace extends Artifact {

  TupleSet tset;

  @OPERATION void init(){
    tset = new TupleSet();
  }

  @OPERATION void out(Tuple t){
    tset.add(t);
  }

  @OPERATION void in(TupleTemplate tt, ActionFeedbackParam<Tuple> res){
    await("foundMatch",tt);
    Tuple t = tset.removeMatching(tt);
    res.set(t);
  }

  @GUARD boolean foundMatch(TupleTemplate tt){
    return tset.hasTupleMatching(tt);
  }

 @OPERATION void inp(TupleTemplate tt, ActionFeedbackParam<boolean> found, ActionFeedbackParam<Tuple> res){
    Tuple t = tset.removeMatching(tt);
    if (res.set(t)){
      found.set(true);
      res.set(t);
    } else {
      found.set(false);
   }
 }

 @OPERATION void rd(TupleTemplate tt, ActionFeedbackParam<Tuple> res){...}
 @OPERATION void rdp(TupleTemplate tt, ActionFeedbackParam<boolean> found, ActionFeedbackParam<Tuple> res){...}
}
```

A CLOCK

```
public class Clock extends Artifact {

  private boolean stopped;

  @OPERATION void init(){
    defineObsProperty("nticks",0);
    stopped = false;
  }

  @OPERATION void start(){
    stopped = false;
    execOp(new Op("ticketing"));
  }

  @OPERATION void stop(){
    stopped = true;
  }

  @INTERNAL_OPERATION void ticketing(){
    while (!stopped){
      int nticks = getObsProperty("nticks").intValue();
      updateObsProperty("nticks", nticks+10);
      signal("tick");
      await_time(10);
    }
  }
}
```

AVAILABLE PROJECTS & THESES /1

- Extending CArtAgO
  - introducing a specific language for defining artifacts
    - using Java only for data-types
  - integration with other agent platforms
    - 2APL
  - working with/to CArtAgO 2.0
    - kernel, IDE, tools
- Applying Jason+CArtAgO
  - Jason+CArtAgO for SOA/WS
    - extending CArtAgO-WS
  - Jason+CArtAgO for Web-Based Computing (2.0,3.0,..)
    - client+server
  - MAS-based Autonomic Systems / Computing & Virtualization
    - MAS for automated management of virtual machines & virtual resources

AVAILABLE PROJECTS & THESES /2

- Defining JaCa
  - language+platform integrating Jason + CArtAgO + Java (for data-types)
- Goal-directed use of artifacts
  - models & languages for manual
  - artifacts in the loop of reasoning

SELECTED BIBLIOGRAPHY

- A. Ricci, M. Piunti, M. Viroli, and A. Omicini. 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, pages 259{288. Springer, 2009
- A. Ricci, M. Viroli, and A. Omicini. 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.
- A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems, 17 (3), Dec. 2008.
- D. Weyns, A. Omicini, and J. J. Odell. 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.
- M. Piunti, A. Ricci, L. Braubach, and A. Pokahr. Goal-directed interactions in artifact-based MAS: Jadex agents playing in CARTAGO environments. In Proc. of IAT (Intelligent Agent Technology) '08 Conference, 2008.
- J. F. Hubner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents: "Giving the organisational power back to the agents". Autonomous Agents and Multi-Agent Systems, 2009. DOI-URL: http://dx.doi.org/10.1007/s10458-009-9084-y.
- Michele Piunti, Andrea Santi, Alessandro Ricci. Programming SOA/WS Systems with BDI Agents and Artifact-Based Environments. Proceedings of AWESOME'09, International Workshop on Agents, Web Services and Ontologies, Integrated Methodologies - part of the Multi-Agent Logics, Languages, and Organisations (MALLOW) Federated Workshops - September 2009 - Torino
- Alessandro Ricci, Michele Piunti and Mirko Viroli. Externalisation and Internalization: A New Perspective on Agent Modularisation in Multi-Agent Systems Programming. Proceedings of LADS'09, International Workshop on LAnguages, methodologies and Development tools for multi-agent systemS, MALLOW'09 Torino