On the Cognitive Use of the Environment through Artifacts

Mirko Viroli
DEIS, Cesena
Alma Mater Studiorum - Università di Bologna
mirko.viroli@unibo.it
joint work with Alessandro Ricci, Andrea Omicini

Outline

- The framework of artifacts for modelling/engineering the environment
- Cognitive selection and use of artifacts
  - impact on agent programming
- A ready-to-use incarnation
  - Prolog-Java programs as cognitive agents
  - TuCSoN tuple centres as artifacts
- An example

ENVs for rational agents!

- ENV is emerging as a key concept in MASs!!!
- Filling the "Agent/ENV gap"
  - MAS research: intentional stance + social interaction
  - ENV research: providing services to black-box agents
- Main challenge
  - A true theory of agent-to-ENV interaction..
- We address the problem at 2 levels
  - Modelling:
  - how to model an ENV from a rational agent viewpoint?
  - Engineering
  - how to design a good ENV for rational agents?

Human Environments

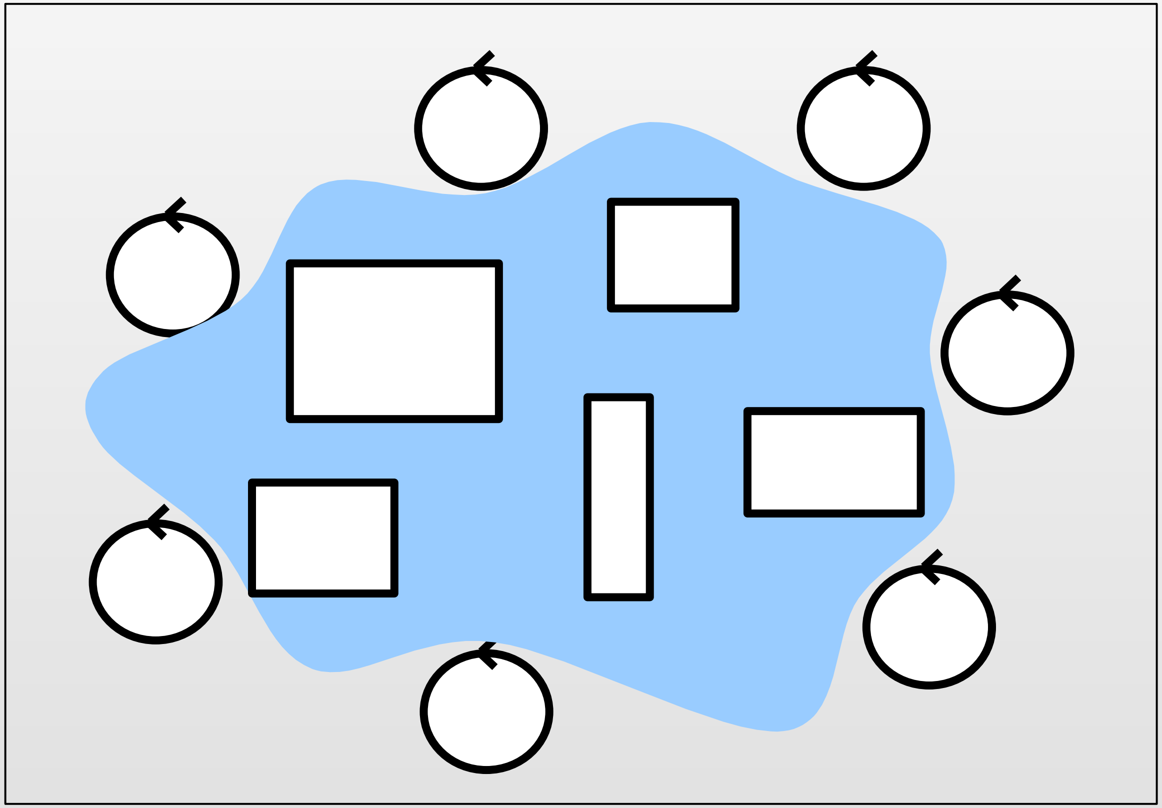
- We take the setting of Activity Theory (AT)
  - Theory of human working activity
- Main Observation
  - "human activities can be understood only by considering both humans and their **context/environment**, seen *as set of mediating artifacts they use*"
- What are these artifacts anyway?
  - Disembodied ones: languages and protocols
  - Embodied ones: maps, checklists, blackboards, communication media, semaphores, ...
- Note:
  - Which cognitive process when using artifacts?

Agent Environments

- The same framework is likely fruitful for agents
- Standard approach
  - agents implicitly use disembodied artifacts
  - ➢ language (speech acts) and protocols
  - ➢ now moving to institutional aspects..
- We investigate explicit use of (embodied) artifacts
  - entities of the environment (that are not agents)
  - agents may exploit them to achieve goals
- MASs applications are already full of artifacts!
  - resources: physical resources, third-party Web services
  - coordination: blackboards, connectors, stigmergic ground
  - organization: e-institutions, agent coordination contexts,..

The MAS picture

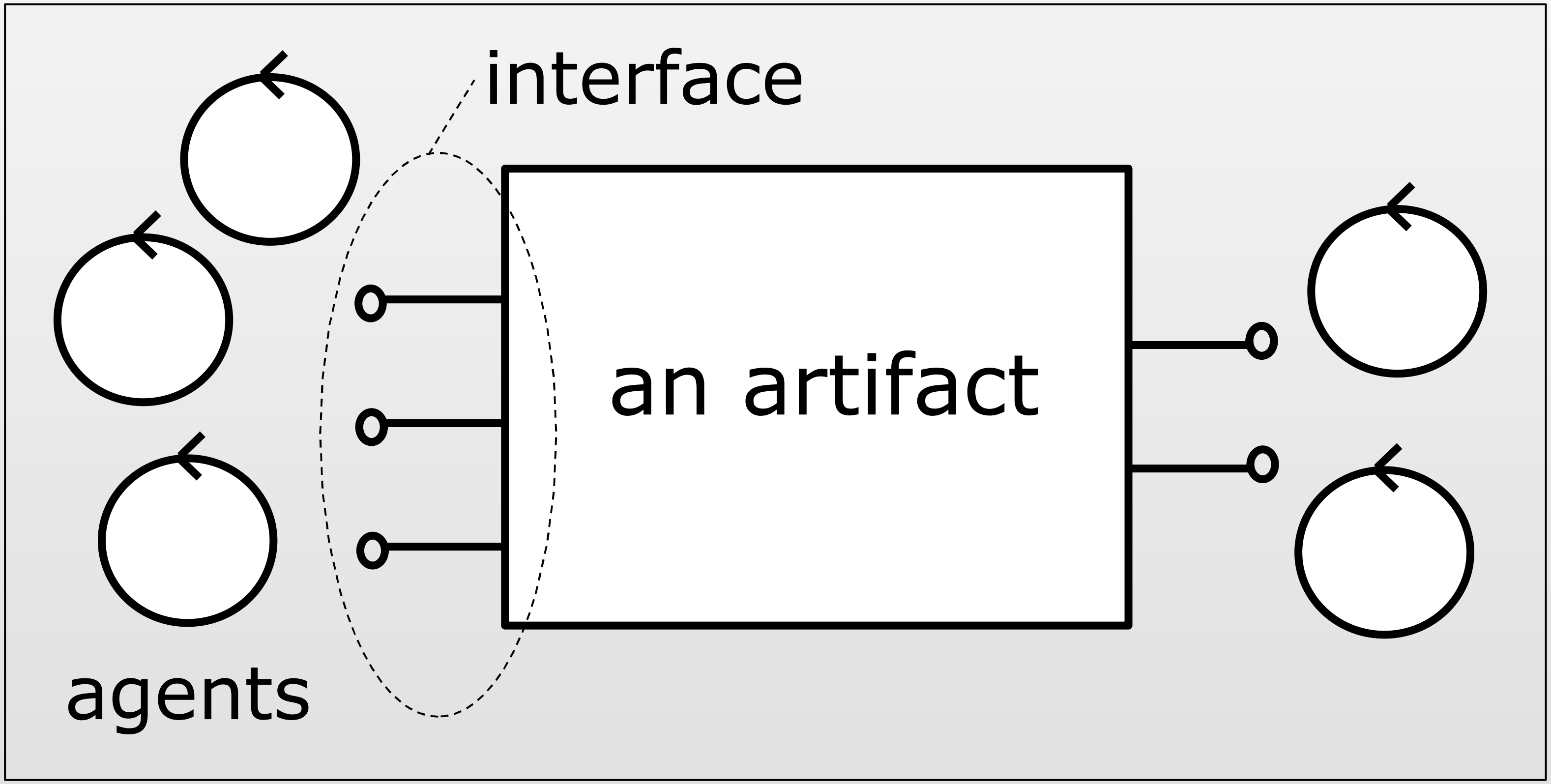
- A rational agent may achieve its goals by either:
  - communicating with other agents (e.g. by goal-delegation)
  - interacting with artifacts in the ENV



[AAMAS2004]
[E4MAS2005]
[PROMAS2005]
[KER2005]

Agents vs. Artifacts

- Differently from other agents:
  - artifacts have an interface by which operations can be executed (artifacts cannot say no!)
  - back to objects? Somehow...



ENV model, artifact model

How a rational agent models an artifact?

- We look for the minimum set of features

1. A mechanism to **interact** with artifacts
   - usage interface (**UI**): which interaction modality?
   - ➢ a set of operations, which are invoked and then completes

3. A mechanism to **select** an artifact to use
   - function description (**FD**): why using that artifact?
   - ➢ a description of what can be expected from the artifact

4. A mechanism to correctly **use** an artifact
   - operating instructions (**OI**): how using that artifact?
   - ➢ a description of the procedure to use the artifact

Intentional stance

- To reason about cognitive exploitation of artifacts, we need to resort to the so-called intentional stance (a main pillar of AOSE)
  - to understand, analyse, and predict a complex system it is useful to ascribe to it mental properties such as beliefs, desires, intentions, goals, hopes, fears,..
- Applied to agents, it does not mean they MUST be internally built as such
  - they are not required to explicitly represent the above properties, and behave accordingly
- It is just an interpretation mean
  - maybe more useful if the agent is a BDI one

More on FD and OI

- Assume a general model for rational agents
  - beliefs (+ intentions) explicitly represented
  - awareness of the artifacts existence
  - scheduling actions and perceiving their completions
  - some computability power (e.g. logic agents)
- FD: why using the artifact?
  - described e.g. by a list of triples
    - preconditions on beliefs and intentions, effects on beliefs
  - an artifact can realise many functions..
  - .. each defines a role for the agent while interacting
- OI: how using the artifact?
  - described by a transition system, i.e. a relation
    - OldOIState x (Action x Precondition x Effect) x NewOIState
  - it is an operational semantics for an OI language...
  - ...also seen as a precompiled plan to use

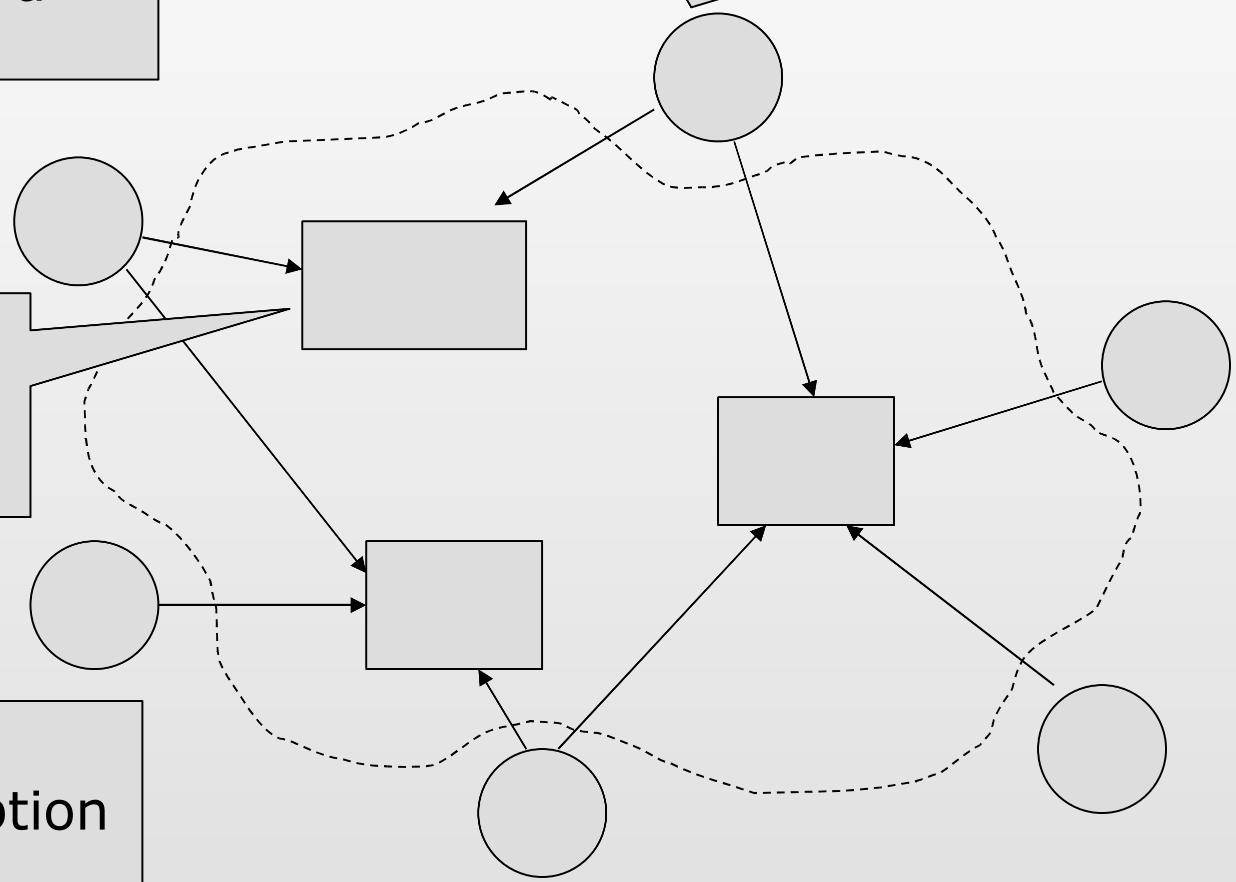
Degrees of cognition

- Programmed use
  - agents exploit artifacts without any cognition about that
- Cognitive use
  - agents do have a representation of the OI state (in beliefs)
  - use it to step-by-step select actions to execute
  - accordingly exploit preconditions and effects
- Cognitive selection & use
  - also have a representation of the FD for some artifacts
  - decide which is compatible with current beliefs + intentions
- ➢ Engineering principles promoting opennes and cognition..
  - design artifacts along with FDs and OIs!
  - let them be inspected through the Usage Interface!

A Framework in TuCSoN

TuCSoN infrastructure: provides coordination services to agents in a distributed setting

Agents:
S/W components
Java-based



Tuple Centre:
Programmable tuple space

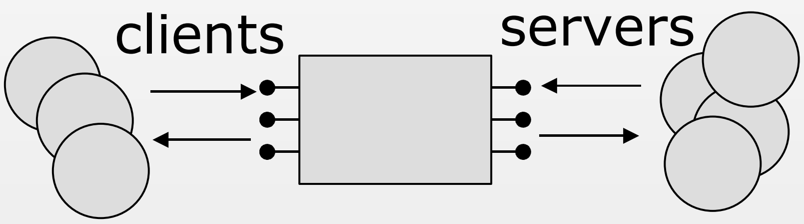
Interactions:
production/consumption of tuples

Details..

- TuCSoN coordination artifacts
  - [TuCSoN by DEIS @ SourceForge]
  - UI: *out(tuple)*, *in(tuple)*, *rd(tuple)*
  - FD: as tuples *fd(Role,BelPre,BelEff,IntPre).*
  - OI: as tuples *oi(Role, OIState)*
- Prolog(-Java) agents
  - [tuProlog by DEIS @ SourceForge]
  - Java programs holding a Prolog theory
  - beliefs and behaviour
  - interactions: execution of TuCSoN operations

A case-study

- Blackboard for knowledge sharing
  - CLIENTs:
    - put a request for some information, then
    - retrieve a reply, then
    - remove the request
  - SERVERs:
    - read pending requests, then
    - put replies or ignore them



- We stick to the “cognitive use” scenario
  - the agent knows that he wants to interact with the artifact being a client or server
  - he inspects operating instructions, and then execute them
- We realise general OI-players!!
  - the two kinds of agent differ only in their initial motivations

A language for OIs

- OIs are a sort of manual for using a device
  - see [Viroli&Ricci@AAMAS2004]
  - once one decides to use a device, he takes the manual and follows instructions
  - in which language should them be written?
- We realised a process algebraic language in Prolog
  - action execution: act(Act,Precondition,Effect)
  - parallel (//) and choice (+) binary composition
  - sequential composition: [OI_1,OI_2,..,OI_n]
  - an operator for recursion: rec(X,OI)
- The OI semantics expressed by a predicate
  - transition(oldOI,Act,newOI,Pre,Eff)
  - either already known to the agent...
  - .. or its clauses could be even dynamically inspected

OI semantics

```
transition(act(A,pre(Pre),eff(Eff)),A,zero,Pre,Eff).
transition(act(A,pre(Pre)),A,zero,Pre,[]).
transition(act(A,eff(Eff)),A,zero,[],Eff).
transition(act(A),A,zero,[],[]).
transition([Act],A,zero,Pre,Eff):-!,transition(Act,A,zero,Pre,Eff).
transition([Act,Act2],A,Act2,Pre,Eff):-!,transition(Act,A,zero,Pre,Eff).
transition([Act|S],A,S,Pre,Eff):-transition(Act,A,zero,Pre,Eff).

transition(S1+S2,A,R1,Pre,Eff):-transition(S1,A,R1,Pre,Eff),!.
transition(S1+S2,A,R2,Pre,Eff):-transition(S2,A,R2,Pre,Eff).
transition(S1//S2,A,R1//S2,Pre,Eff):-transition(S1,A,R1,Pre,Eff),!.
transition(S1//S2,A,S1//R2,Pre,Eff):-transition(S2,A,R2,Pre,Eff).

transition(rec(X,S),A,R,Pre,Eff):-
        copy_term(S,S2),transition(rec(X,S,S2),A,R,Pre,Eff).
transition(rec(X,S,X),A,rec(X,S,R),Pre,Eff):-
        !,copy_term(S,S2),transition(S2,A,R,Pre,Eff).
transition(rec(X,S,R),A,rec(X,S,R2),Pre,Eff):-transition(R,A,R2,Pre,Eff).
```

OI semantics (1/2)

```
transition(act(A,pre(Pre),eff(Eff)),A,zero,Pre,Eff).
transition(act(A,pre(Pre)),A,zero,Pre,[]).
transition(act(A,eff(Eff)),A,zero,[],Eff).
transition(act(A),A,zero,[],[]).

transition([Act],A,zero,Pre,Eff):-
                !,transition(Act,A,zero,Pre,Eff).
transition([Act,Act2],A,Act2,Pre,Eff):-
                !,transition(Act,A,zero,Pre,Eff).
transition([Act|S],A,S,Pre,Eff):-
                transition(Act,A,zero,Pre,Eff).
```

OI semantics (2/2)

```
transition(S1+S2,A,R1,Pre,Eff):-
                    transition(S1,A,R1,Pre,Eff),!.
transition(S1+S2,A,R2,Pre,Eff):-
                    transition(S2,A,R2,Pre,Eff).
transition(S1//S2,A,R1//S2,Pre,Eff):-
                    transition(S1,A,R1,Pre,Eff),!.
transition(S1//S2,A,S1//R2,Pre,Eff):-
                    transition(S2,A,R2,Pre,Eff).

transition(rec(X,S),A,R,Pre,Eff):-
          copy_term(S,S2),transition(rec(X,S,S2),A,R,Pre,Eff).
transition(rec(X,S,X),A,rec(X,S,R),Pre,Eff):-
          !,copy_term(S,S2),transition(S2,A,R,Pre,Eff).
transition(rec(X,S,R),A,rec(X,S,R2),Pre,Eff):-
          transition(R,A,R2,Pre,Eff).
```

OI semantics

Semantics of choice S1+S2:

- If S1 admits a transition to R1, then R1 is a next possible state
- Otherwise, try with S2

```
transition(act(A
transition(act(A
transition(act(A
transition(act(A
transition([Act],
transition([Act,Act2],A,           ):-!,transition(Act,A,zero,Pre,Eff).
transition([Act|S],A,S        Eff):-transition(Act,A,zero,Pre,Eff).

transition(S1+S2,A,R1,Pre,Eff):-transition(S1,A,R1,Pre,Eff),!.
transition(S1+S2,A,R2,Pre,Eff):-transition(S2,A,R2,Pre,Eff).
transition(S1//S2,A,R1//S2,Pre,Eff):-transition(S1,A,R1,Pre,Eff),!.
transition(S1//S2,A,S1//R2,Pre,Eff):-transition(S2,A,R2,Pre,Eff).

transition(rec(X,S),A,R,Pre,Eff):-
        copy_term(S,S2),transition(rec(X,S,S2),A,R,Pre,Eff).
transition(rec(X,S,X),A,rec(X,S,R),Pre,Eff):-
        !,copy_term(S,S2),transition(S2,A,R,Pre,Eff).
transition(rec(X,S,R),A,rec(X,S,R2),Pre,Eff):-transition(R,A,R2,Pre,Eff).
```

Client and Server

```
// CLIENT before

bel(using_artifact(art@localhost)).
bel(role(client)).
bel(id('id1')).

bel(unknown(temp)).
bel(unknown(pressure)).
```

```
// CLIENT after

bel(using_artifact(art@localhost)).
bel(role(client)).
bel(id('id1')).

bel(val(temp,5)).
bel(val(pressure,21)).
```

```
// SERVER

bel(using_artifact(art@localhost)).
bel(role(server)).
bel(id('id2')).

bel(val(wind,-1)).
bel(val(temp,5)).
bel(val(pressure,21)).
```

Client and Server OIs

```
oi(client,rec(x,[
        act(        out(request(Id,Property)),
                    pre([holds(id(Id)),holds(unknown(Property))])
        ),
        act(        rd(reply(Property,Value)),
                    eff([bel(val(Property,Value)),nbel(unknown(Property))])
        ),
        act(        in(request(Id,Property))),
        x
])).
```

```
oi(server,rec(x,[
          act(      rd(request(Id,Property))),
          ([
              act(  out(reply(Property,Value)),
                    pre([holds(val(Property,Value))])),
              x
          ]+x)
])).
```

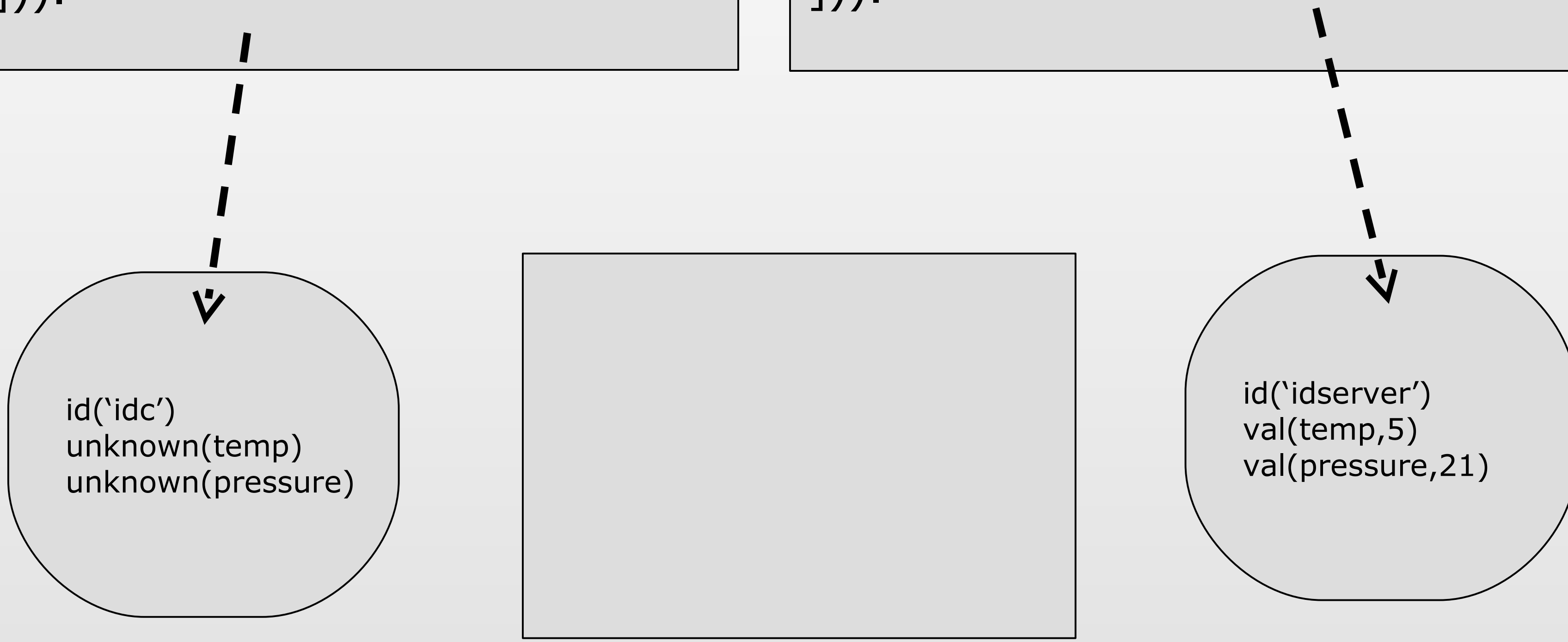
Agents as OI-players

```
bel(using_artifact(art@localhost)).       % Artifact to use
bel(role(client)).                        % Role to play
bel(id('id1')).                           % Identity

bel(unknown(temp)).                       % Knowledge
bel(unknown(pressure)).
bel(val(wind,-1)).

start :-  bel(using_artifact(Art)), bel(role(R)),
          exec(Art?rd(oi(R,S))),          % Inspect OIs
          updateOI(S),                    % Store OIs
          loop.

loop:-    bel(using_artifact(Art)),oi(S),
          transition(S,Act,S2,Pre,Eff),hold(Pre),  % Seek for next Act
          exec(Art?Act),                           % Execute Act
          apply(Eff), updateOI(S2),                % Update state
          !,loop.
loop.
```

Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(out(request(Id,Property))),
  act(rd(reply(Property,Value))),
  act(in(request(Id,Property))),x
])).
```

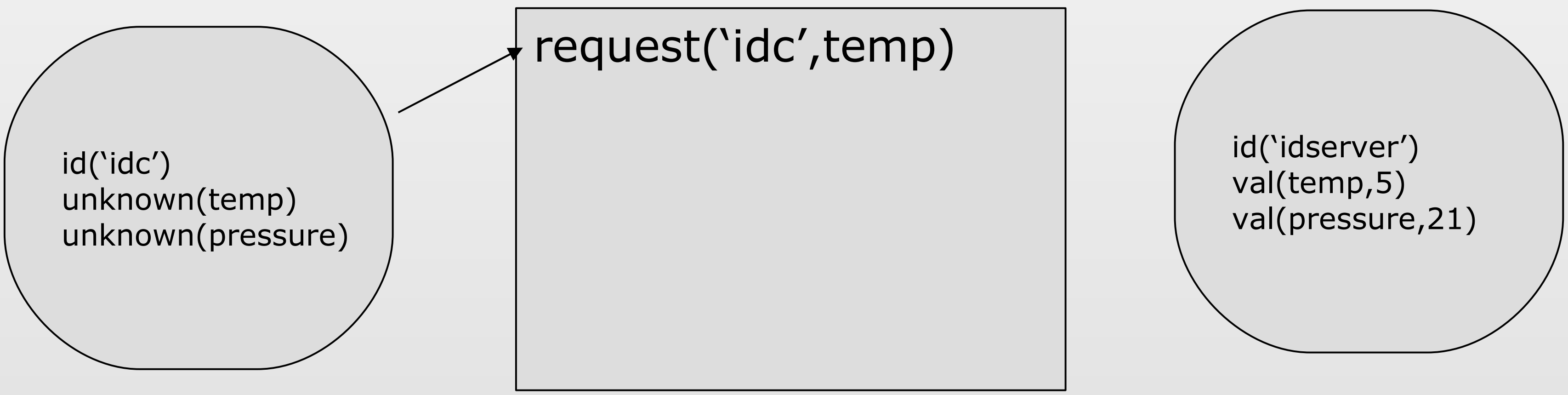
```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(rd(reply(temp,Value))),
  act(in(request(Id,temp))),x
])).
```

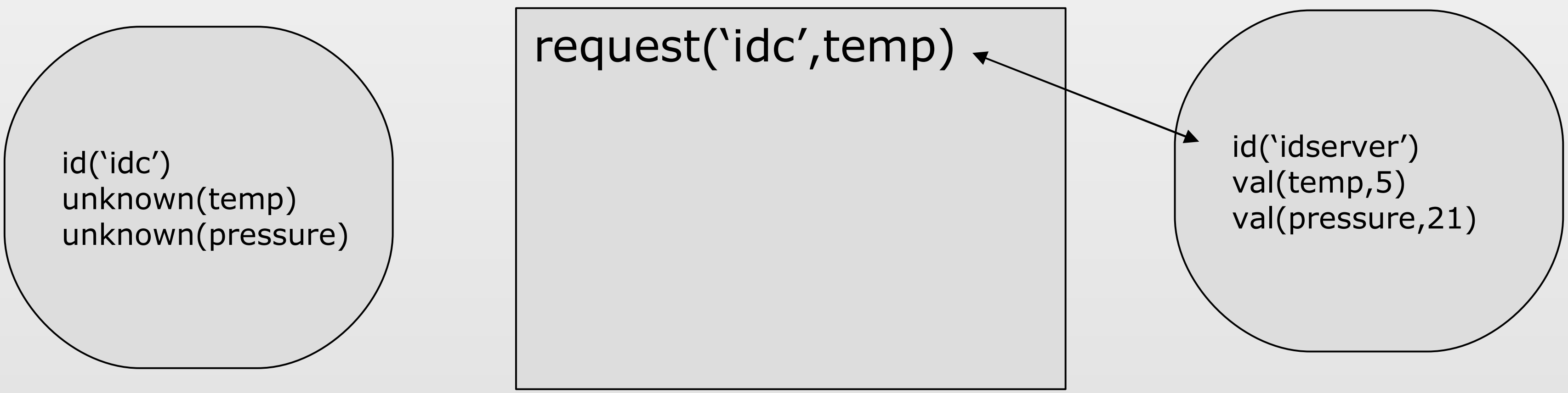
```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(rd(reply(temp,Value))),
  act(in(request(Id,temp))),x
])).
```

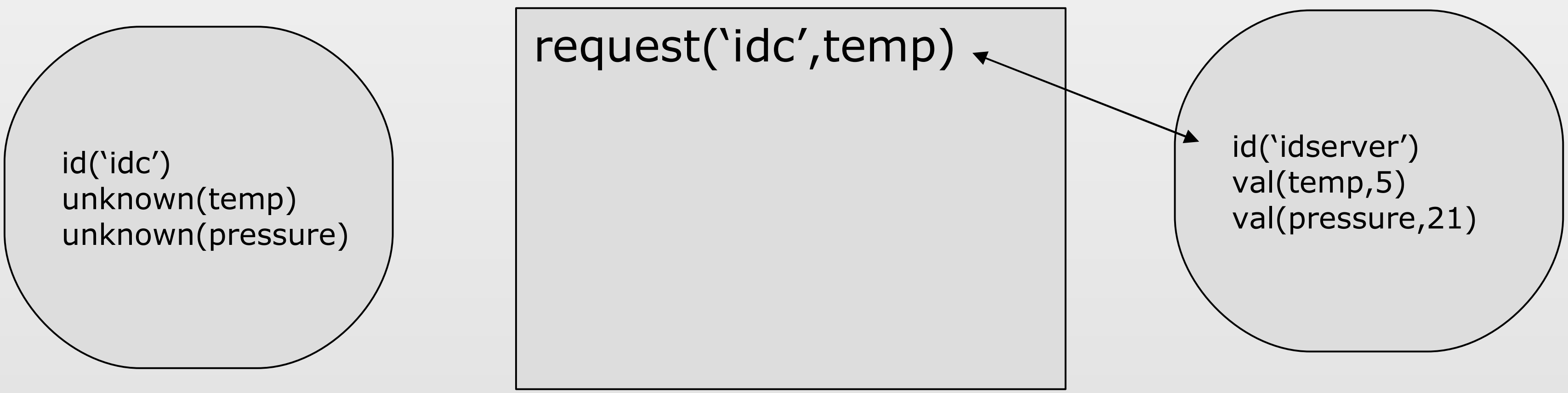
```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(rd(reply(temp,Value))),
  act(in(request(Id,temp))),x
])).
```

```
oi(server,rec(x,OI_server,[
  ([act(out(reply(temp,Value)),
     x]+x)
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(rd(reply(temp,Value))),
  act(in(request(Id,temp))),x
])).
```

```
oi(server,rec(x,OI_server,[
   ([act(out(reply(temp,Value)),
      x]+x)
])).
```

id(‘idc’)
unknown(temp)
unknown(pressure)

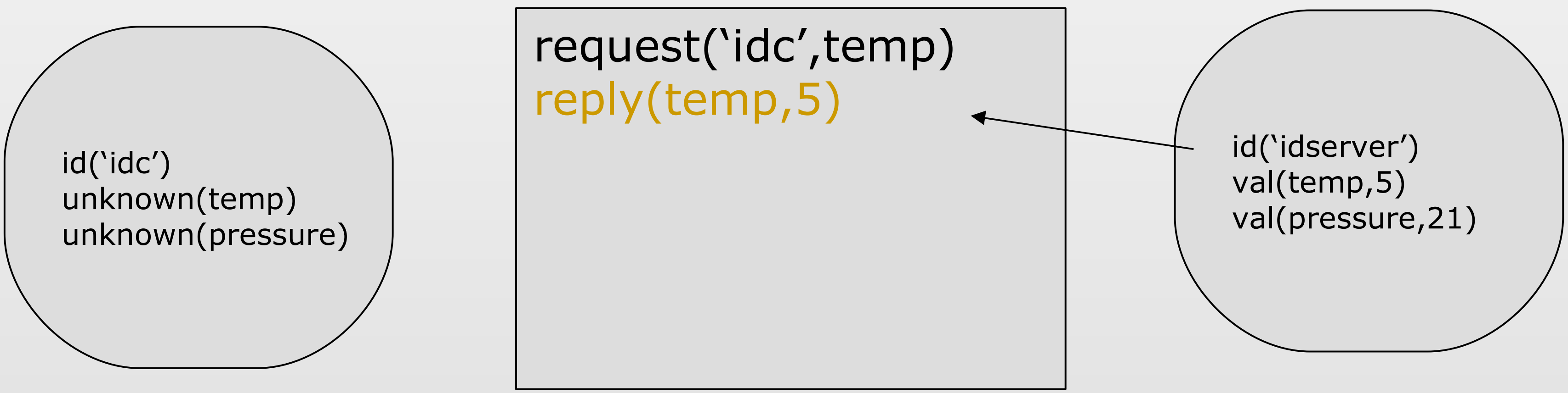
request(‘idc’,temp)
reply(temp,5)

id(‘idserver’)
val(temp,5)
val(pressure,21)

Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(rd(reply(temp,Value))),
  act(in(request(Id,temp))),x
])).
```

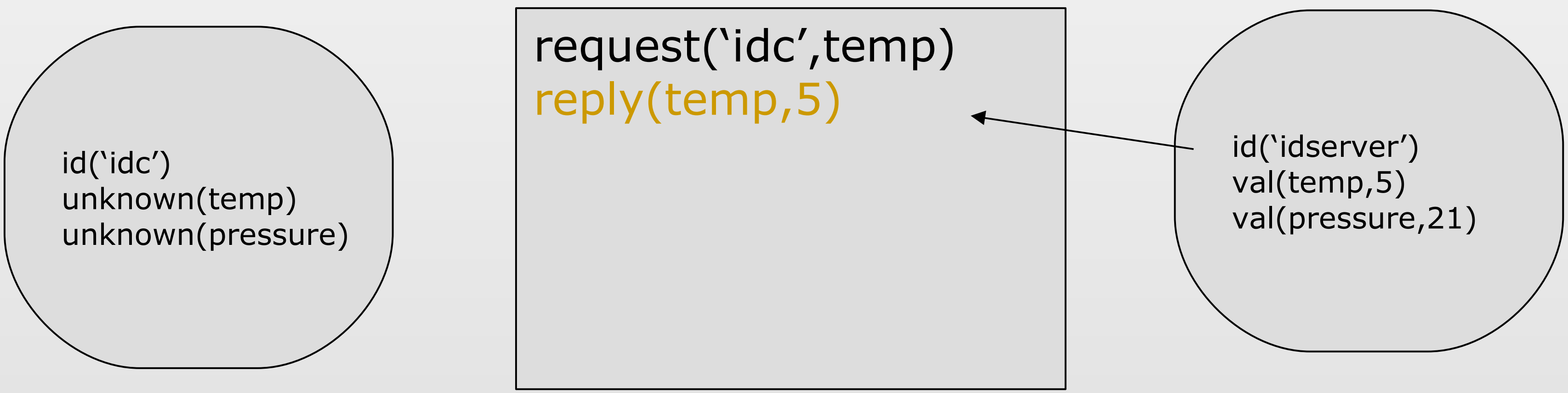
```
oi(server,rec(x,OI_server,[
   x
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(rd(reply(temp,Value))),
  act(in(request(Id,temp))),x
])).
```

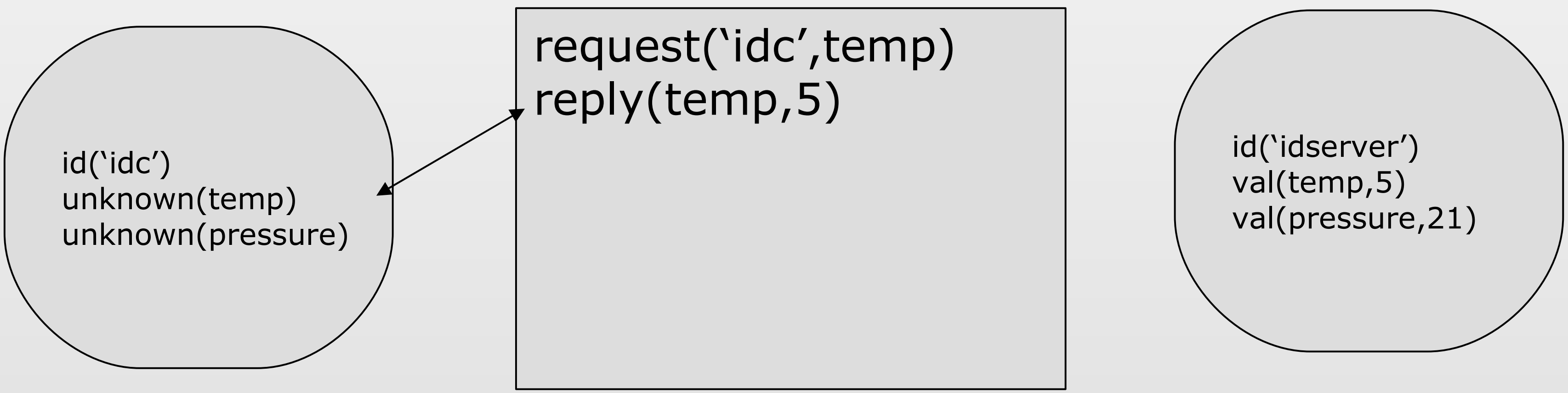
```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(rd(reply(temp,Value))),
  act(in(request(Id,temp))),x
])).
```

```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(in(request(Id,temp))),x
])).
```

```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```

id(‘idc’)
val(temp,5)
unknown(pressure)

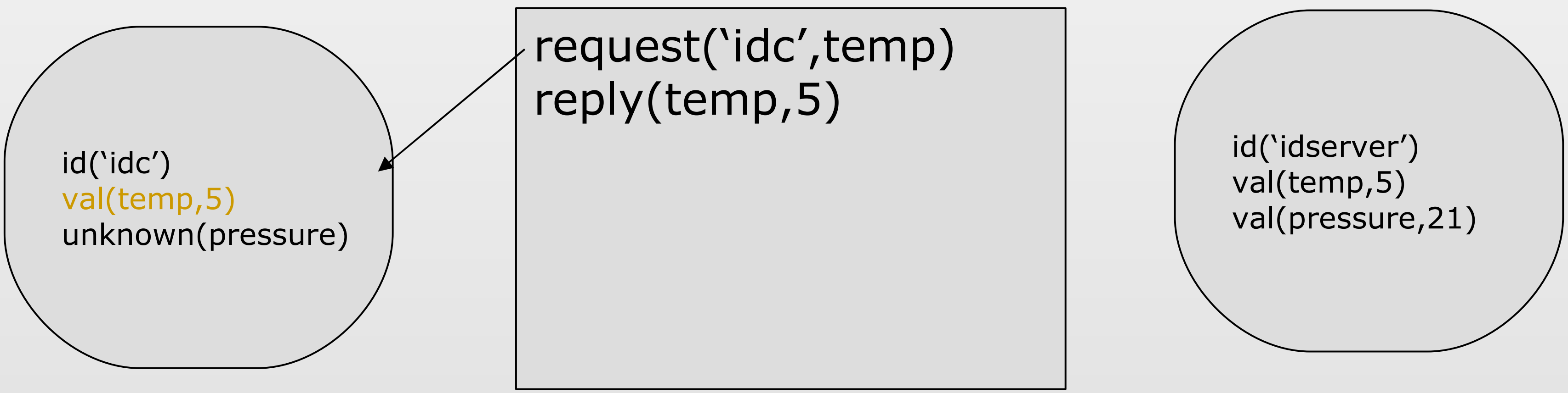
request(‘idc’,temp)
reply(temp,5)

id(‘idserver’)
val(temp,5)
val(pressure,21)

Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(in(request(Id,temp))),x
])).
```

```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```



Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(in(request(Id,temp))),x
])).
```

```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```

id(‘idc’)
val(temp,5)
unknown(pressure)

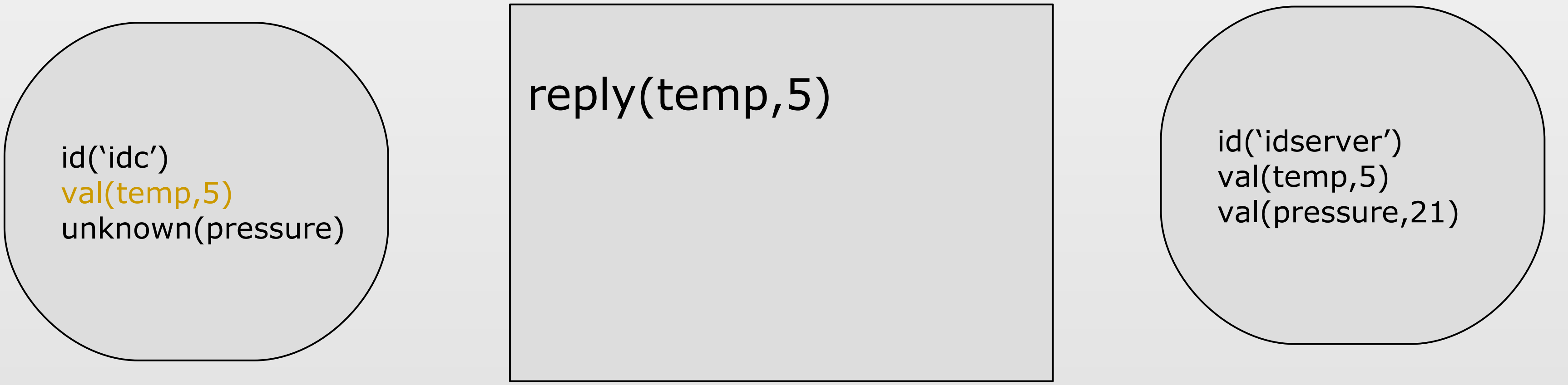
reply(temp,5)

id(‘idserver’)
val(temp,5)
val(pressure,21)

Client and Server OIs

```
oi(client,rec(x,OI_client,[
  act(out(request(Id,Property))),
  act(rd(reply(Property,Value))),
  act(in(request(Id,Property))),x
])).
```

```
oi(server,rec(x,OI_server,[
   act(rd(request(Id,Property))),
   ([act(out(reply(Property,Value)),
      x]+x)
])).
```



Features

- It scales with the number of clients and servers
- The artifact can be specialised
  - currently it is a simple blackboard
  - rules can be added to improve “effectiveness”

➢ Example: retracting replies
  - server replies remain indefinitely in the artifact...
  - .. should automatically retract them after a while!!!
  - can be realised in ReSpecT by rule:

```
reaction (out(reply(Property,Value)),(
                    current_time(Time),
                    rd_r(timeout(Timeout)),
                    ExpireTime is Time+Timeout
                    out_r_spec(time(ExpireTime), in(reply(Property,Value)))
)).
```

Conclusions

- Certain responsibilities are better delegated to artifacts, as specialised tools
- Thanks to features such as UI, FD and OI agents can exploit artifacts rationally
  - this can be smoothly realised using logic agents
- Future work in this direction
  - evaluating support in agent languages such as 3APL, Jason,.. and in full BDI frameworks
  - towards cognitive selection, use, manipulation, construction
  - integration with self-organisation

On the Cognitive Use of the Environment through Artifacts

Mirko Viroli, Alessandro Ricci
DEIS, Cesena
Alma Mater Studiorum - Università di Bologna
mirko.viroli@unibo.it
joint work with Andrea Omicini