RBAC for MAS Coordination Infrastructure

Alessandro Ricci, Mirko Viroli, Andrea Omicini

DEIS - Università di Bologna

Sede di Cesena

(Italy)

Outline

- Context & goals: modelling security issues on top of an agent coordination infrastructure
- Role-Based Access Control (RBAC) Model Overview
- RBAC in MAS infrastructure through Agent Coordination Contexts (ACC)
- Embedding RBAC in TuCSoN infrastructure

Context

- Software engineering perspective
  - agents and MAS as engineering paradigm
  - MAS *organisation* and *coordination* as basic engineering dimensions
- Security issues in the context of MAS organisation and coordination models
  - focus on access control
  - (organisation | coordination | security) relationships
    - shaping (specifying, constraining) the agent interaction space
    - security as *normative* counterpart of coordination
    - static (organisation) + dynamic (coordination) management
- *Infrastructure* level
  - devising structures & services useful for engineering systems on top

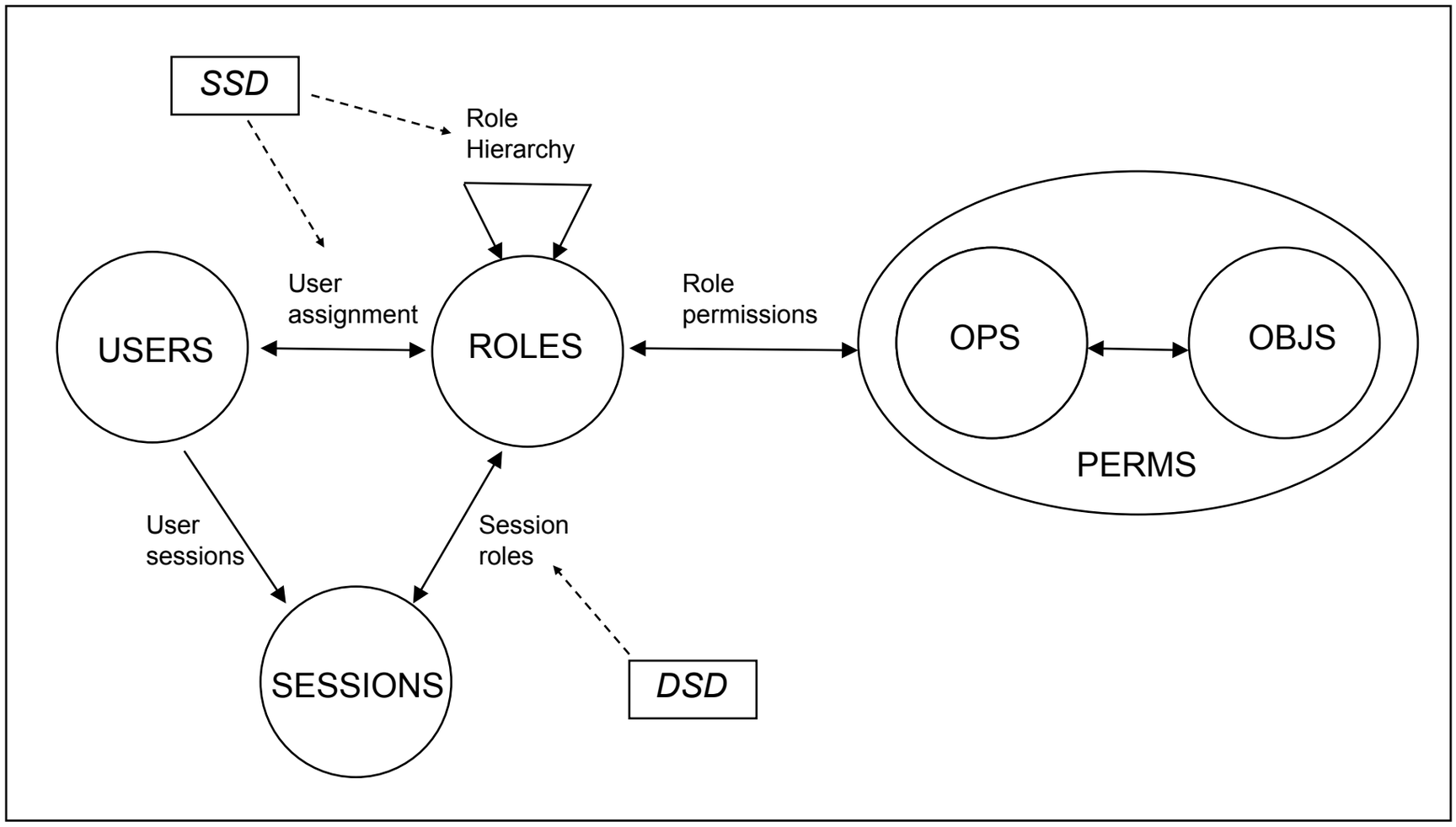
Goals

- Modelling access control within a coordination / organisation context
- Integrating a Role-Based Access Control model (RBAC) model on an existing agent based coordination Infrastructure (TuCSoN)

RBAC Model Overview

- State-of-the-art models for engineering security in complex information systems
- Access-control policies formulated in terms of *roles*
- Specifying and enforcing forms of separation of duty (static and dynamic)
  - User-role, inter-role and role-resource relationships
- Flexible administration of security policy
  - security policy encapsulation
  - dynamic / open environment

RBAC Architecture



"Proposed NIST Standard for RBAC" (Ferraiolo, Sandhu, Gavrila, Kuhn, Chandramouli)
ACM Transactions on Information System and Security, 4(3), 2001

How to embed

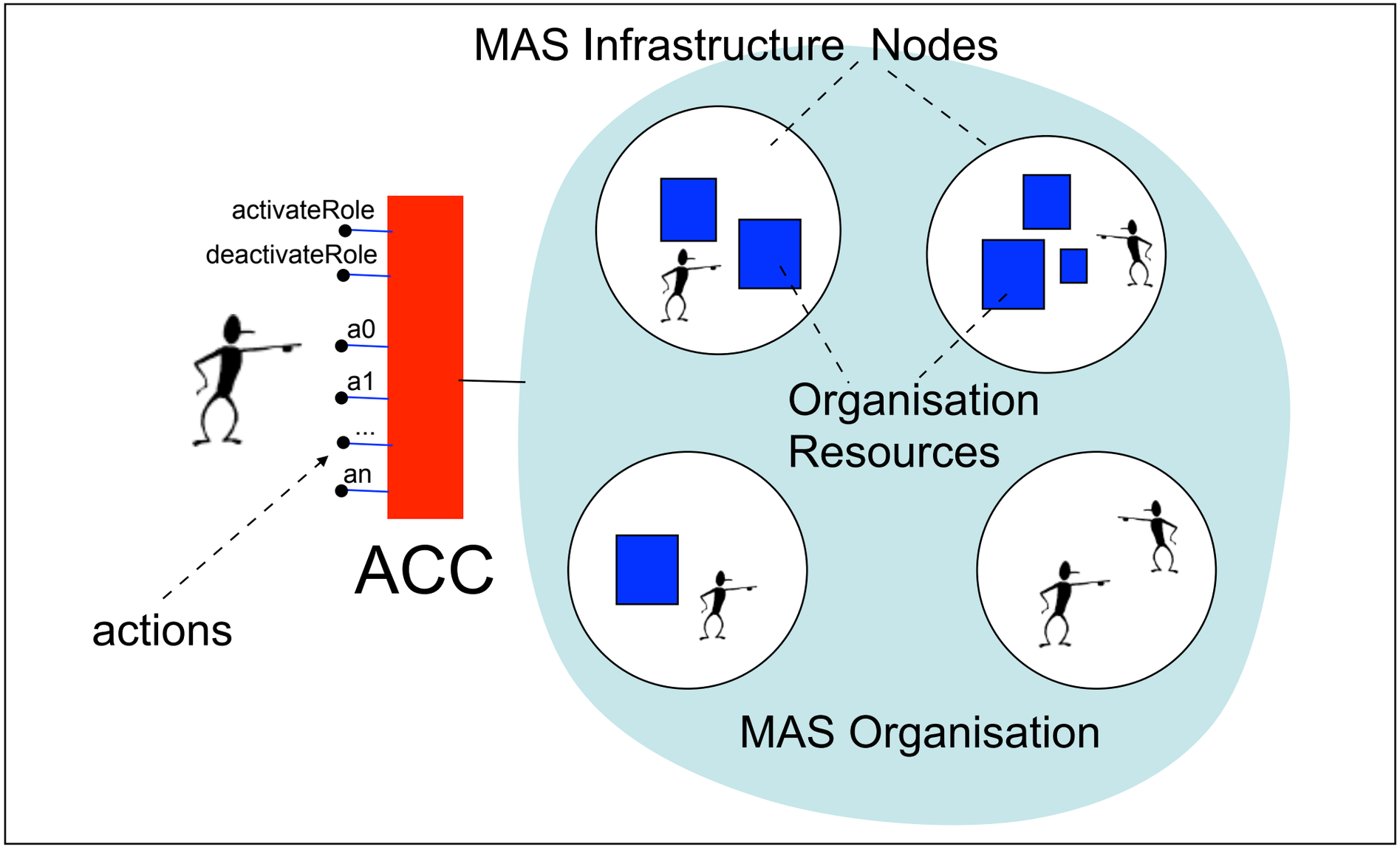
an RBAC-like architecture in

in the agent world

(in MAS models / infrastructures) ?

Agent Coordination Contexts (ACC)

- Infrastructure abstraction modelling / engineering agent *presence* inside an environment



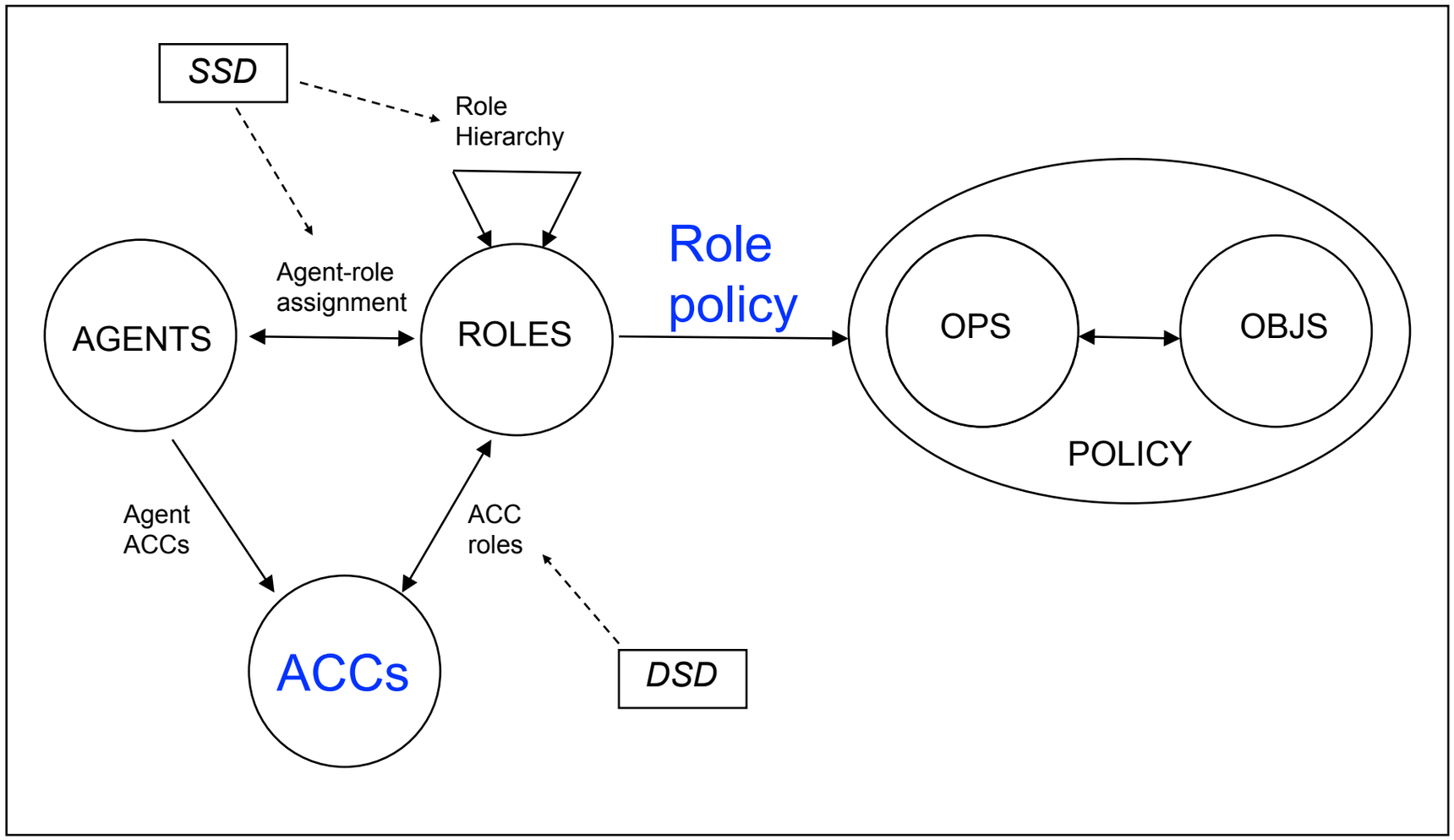
ACC characterisation

- [ACC as an *stateful runtime interface (contract)*]
  - Defining the space of agent (allowed) *actions* inside an environment
  - Enforcing patterns of actions / protocols

- [ACC as an *organisational abstraction*]
  - Representing the set of dynamic roles the agent is playing inside an organisational environment
  - Each agent has an ACC for each organisation where playing
  - roles can be activated / deactivated at runtime in the ACC

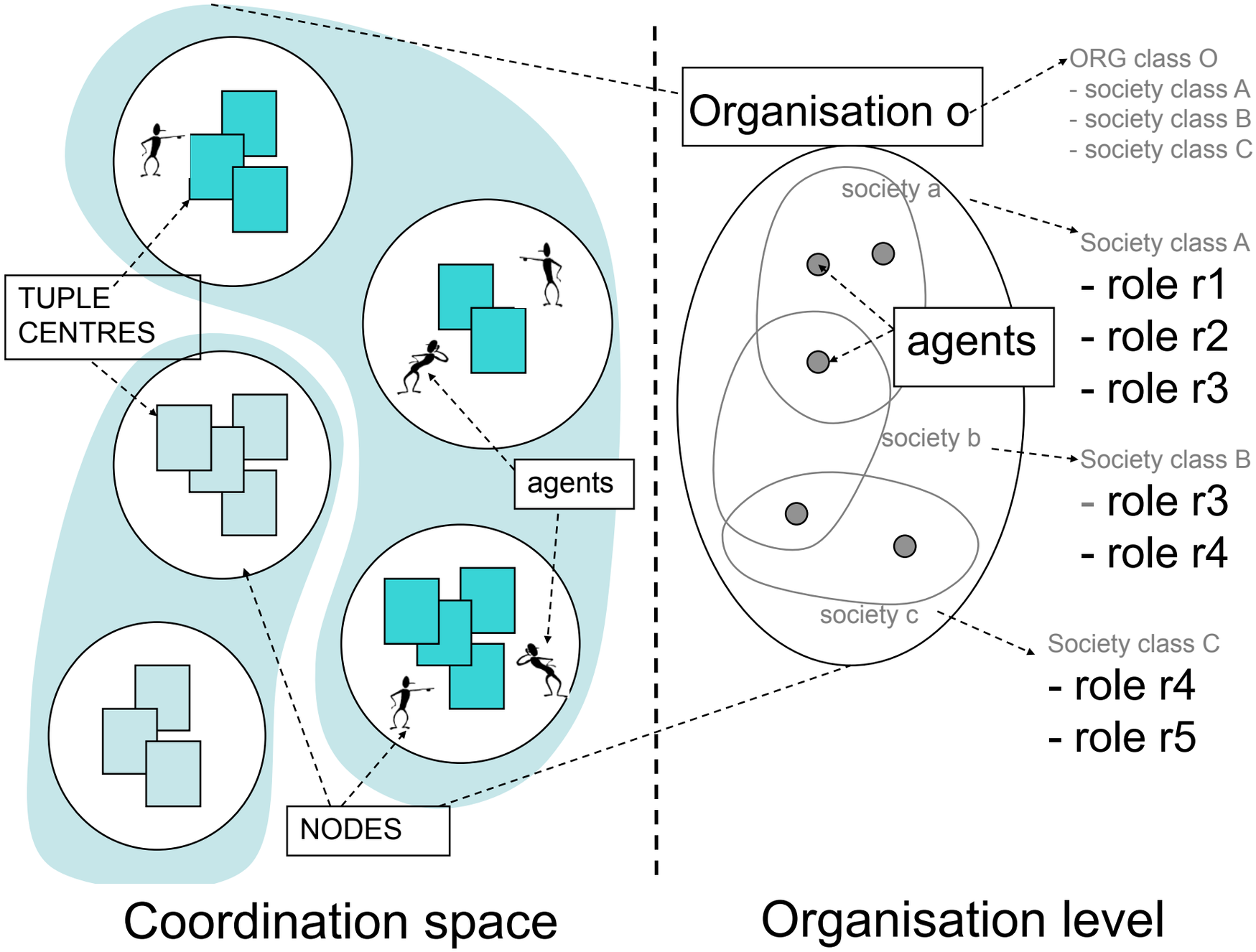
- [An ACC as a *working session*]
  - An agent obtains an ACC to start playing inside an organisation, and then release it when finishing

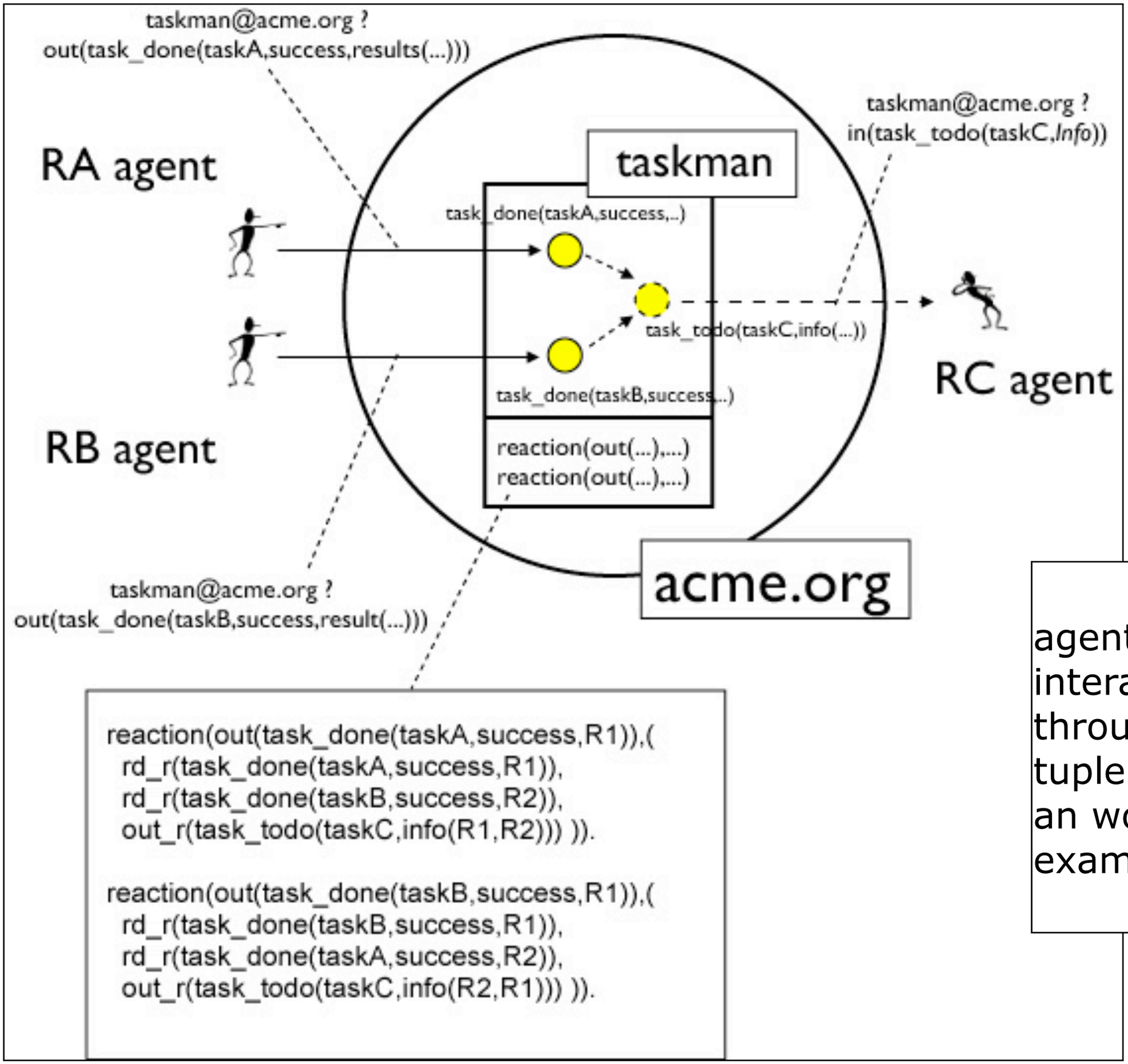
RBAC with ACCs



Applying the approach: TuCSoN Coordination Infrastructure

- TuCSoN = Tuple Centre Spread over the Network
- *Tuple centres* as general purpose customisable coordination artifacts shared and accessed by agents
  - programmable tuple space (~reactive logic based blackboards)
  - out, in, rd operations for inserting / retrieving logic tuples to / from tuple centres
  - coordinating behaviour specified with a ReSpecT program
  - get_spec, set_spec to change the coordinating behavcaviour dynamically
- Tuple centres are collected (distributed) in TuCSoN *nodes*

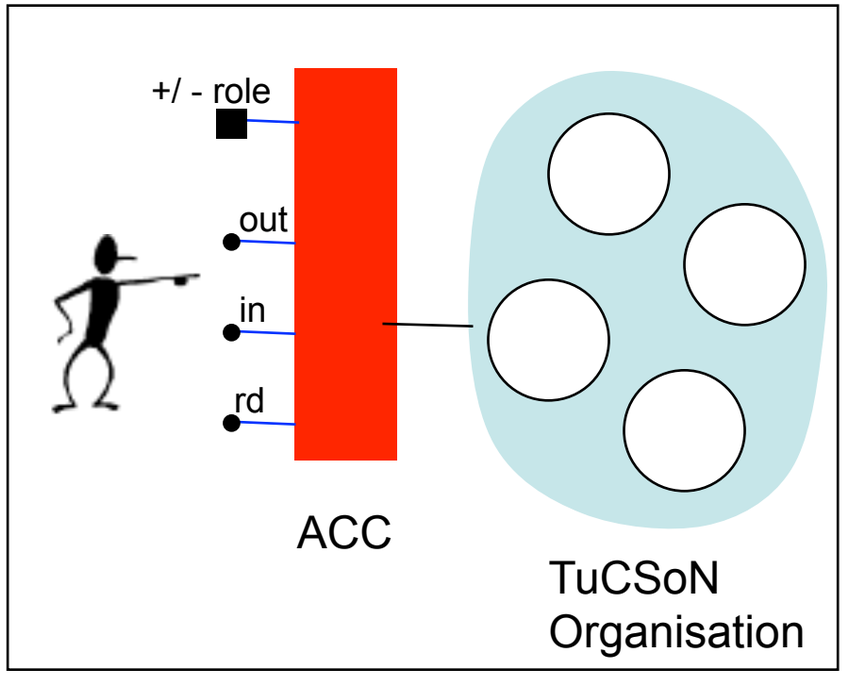

Organisation o
ORG class O
- society class A
- society class B
- society class C
society a
Society class A
- role r1
- role r2
- role r3
agents
society b
Society class B
- role r3
- role r4
society c
Society class C
- role r4
- role r5
TUPLE CENTRES
agents
NODES
Coordination space
Organisation level



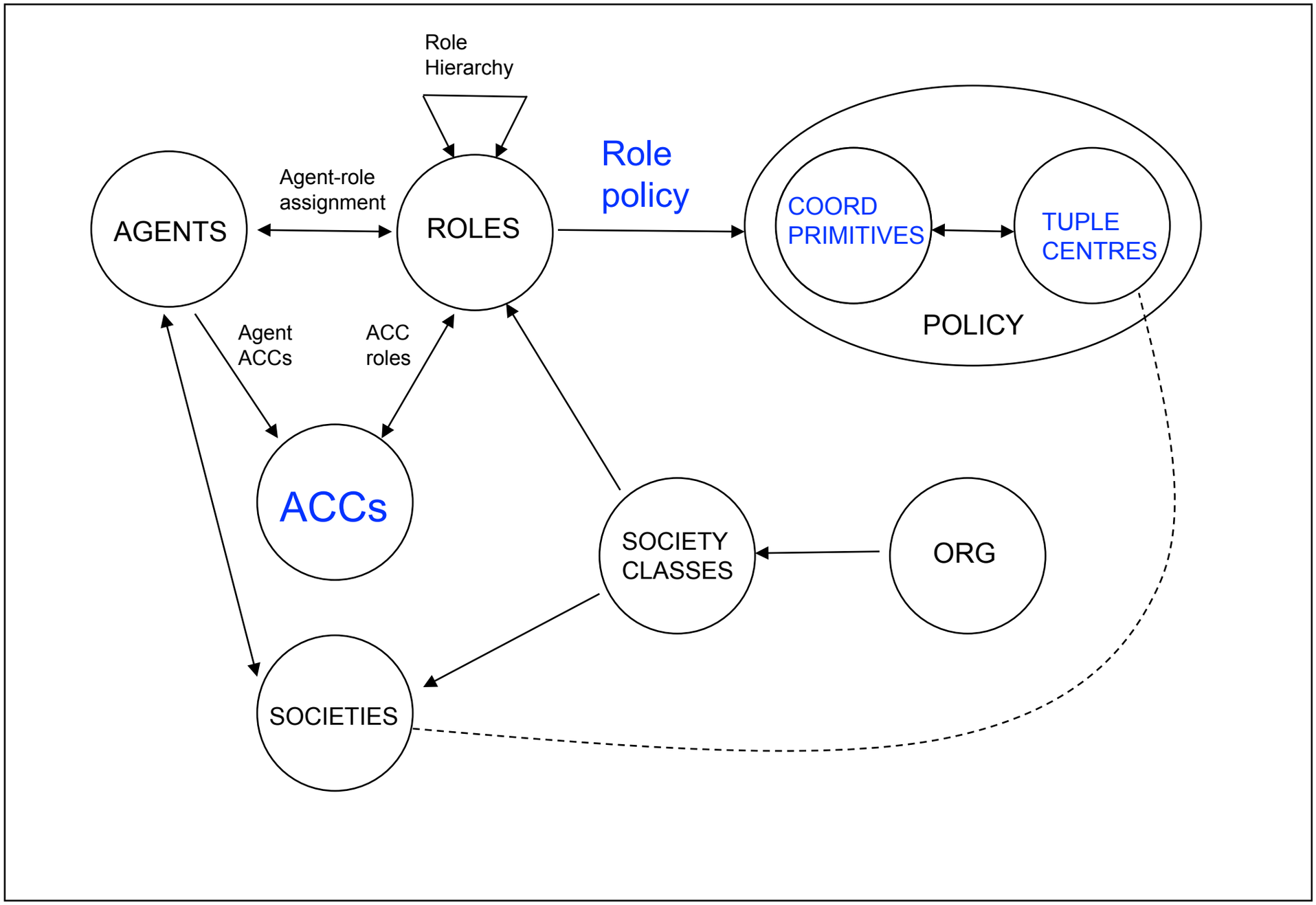
agents interacting through a tuple centre: an workflow example

ACC in TuCSoN

- ACC = runtime interface used by an agent for interacting with tuple centres
  - defines agent (allowed) operation on tuple centres
    - in, rd, out, inp, rdp
    - get_spec, set_spec



RBAC in TuCSoN



Role policy description

- Extending RBAC policy for handling protocols and context-awareness
  - Representation of current role state
  - Notion of local time, agent position, identity..

Policy expressed as a Prolog theory

```
can_do(CurrentState,Action,NextState)
                    :-Conditions.
```

Policy Examples

Resource Access

Student Role:

```
can_do(_,printer ? out(doc_to_print(type(txt),Document)),_).
```

Professor Role:

```
can_do(_,printer ? out(doc_to_print(type(_),Document)),_).
can_do(_,printer ? rd(config(_)),_).
```

Technician Role:

```
can_do(_,printer ? out(config(_)),_).
can_do(_,printer ? rd(config(_)),_).
```

Msg Box Service

User Role:

```
can_do(_,msg_box?out(msg(AgentID,Content)),_).
can_do(_,msg_box?in(msg(AgentId,Content)),_):-agent_id(AgentId).
```

Context awareness

Guest Role:

```
can_do(_,Action, _):- session_time(ST), ST < T.
can_do(_, Tid ? out(_), _):-local_node(Node).
```

Policy Examples

Resource Access

Student Role:

```
can_do(_,printer ? out(doc_to_print(type(txt),Document)),_).
```

Professor Role:

```
can_do(_,printer ? out(doc_to_print(type(_),Document)),_).
can_do(_,printer ? rd(config(_)),_).
```

Technician Role:

```
can_do(_,printer ? out(config(_)),_).
```

Msg Box Service

User Role:

```
can_do(_,msg_box?out(msg(AgentID,Content)),_).
can_do(_,msg_box?in(msg(AgentId,Content)),_):-agent_id(AgentId).
```

Context awareness

Guest Role:

```
can_do(_,Action, _):- session_time(ST), ST < T.
can_do(_, Tid ? out(_), _):-local_node(Node).
```

Policy Examples

Resource Access

Student Role:

```
can_do(_,printer ? out(doc_to_print(type(txt),Document)),_).
```

Professor Role:

```
can_do(_,printer ? out(doc_to_print(type(_),Document)),_).
can_do(_,printer ? rd(config(_)),_).
```

Technician Role:

```
can_do(_,printer ? out(config(_)),_).
```

Msg Box Service

User Role:

```
can_do(_,msg_box?out(msg(AgentID,Content)),_).
can_do(_,msg_box?in(msg(AgentId,Content)),_):-agent_id(AgentId).
```

Context awareness

Guest Role:

```
can_do(_,Action, _):- session_time(ST), ST < T.
can_do(_, Tid ? out(_), _):-local_node(Node).
```

A more involved example: A Constrained Contract Net Protocol

The Contract Net Protocol

- Task allocation from *masters* to *workers*
- Masters make a task announcement, workers provide their bids, and then the task is allocated to the winning bidder, chosen by the master
- *open systems* perspective

Desiderata

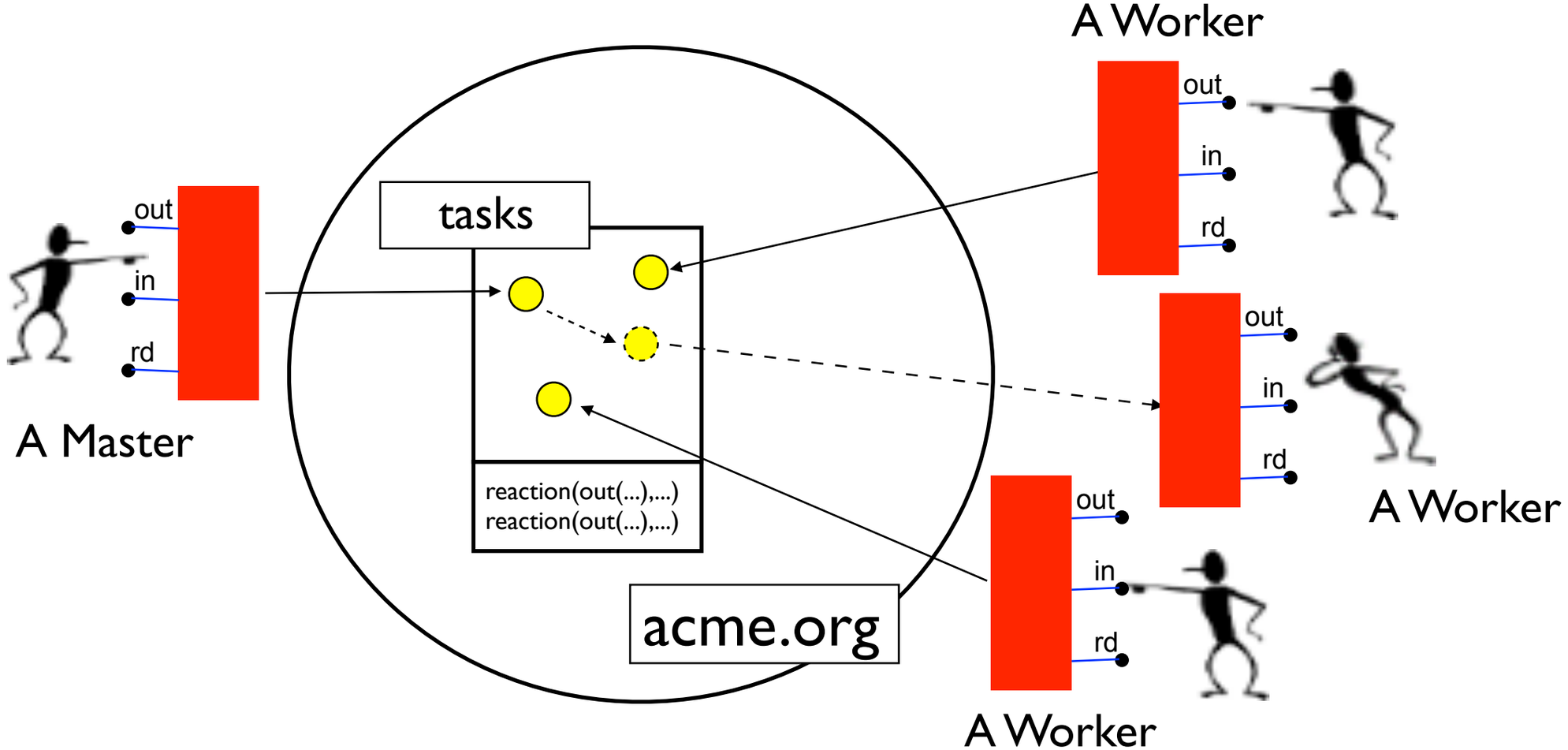
- explicitly specifying/enforcing master & worker protocols
- avoiding malicious/wrong interactions
  - openness challenge

CNP with ACCs and a Coordination Artifact

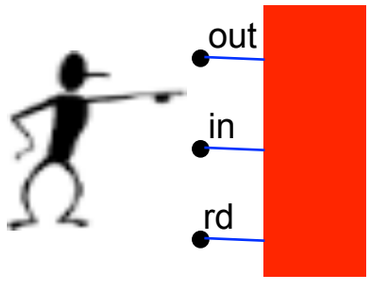
TuCSoN approach

A coordination artifact for supporting the task allocation coordination activity (`tasks` tuple centre)

ACC for specifying & ruling agent actions, playing master and worker roles

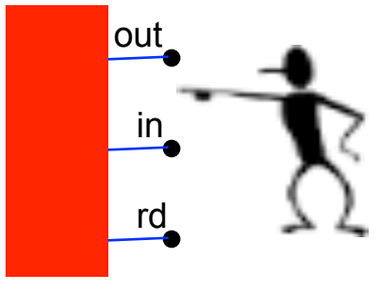


CNP Protocols & Coordination Laws



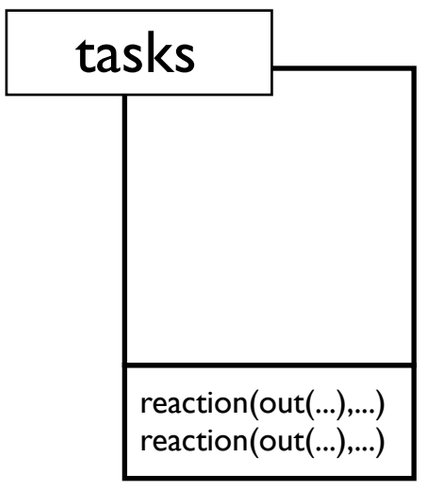
A Master

```
enterACC(acme.org,
       [organisation(acme), role(master,task_distribution)])
tasks ?  out(announcement(Task))
wait(ExpireTime)
tasks ?  in(bids(Task,BidList))
Bid ← selectWinner(BidList)
tasks ?  out(award(Task,Bid))
tasks ?  in(result(Task,Result)
```



A Worker

```
enterACC(acme.org,
    [organisation(acme), role(worker,task_distribution])
tasks ?  rd(announcement(Task))
MyBid ← evaluate(Task)
tasks ?  in(bid(Task,MyBid,Answer))
if (Answer=='awarded') {
    Result ← perform(Task)
    tasks ?  out(result(Task,Result)}
```




```
reaction( in(bid(Task,MyBid,Answer )), ( pre,
  out_r(contractor(Task, MyBid)),
  in_r(bids(Task, L)),
  out_r(bids(Task, [MyBid|L])) )).

reaction( out(announcement(Task) ), (
  out_r(bids(Task,[])) )).

reaction( in(bids(Task,L) ), ( post,
  in_r(announcement(Task)) )).

reaction( out(award(Task,TheBid) ), (
  in_r(award(Task,TheBid)),
  in_r(contractor(Task,TheBid)),
  out_r(bid(Task,TheBid,awarded)),
  out_r(refuse_others(Task)) )).
```

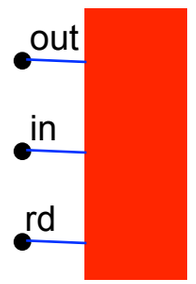
```
reaction( out(award(Task,TheBid) ),(
  in_r(award(Task,TheBid)),
  in_r(contractor(Task,TheBid)),
  out_r(bid(Task,TheBid,awarded)),
  out_r(refuse_others(Task)) )).

reaction( out_r(refuse_others(Task) ), (
  in_r(refuse_others(Task)),
  in_r(contractor(Task,TheBid)),
  out_r(bid(Task,TheBid,'not-awarded')),
              new_task_announcement(Task))) ).
  out_r(refuse_others(Task)) )).

reaction( out_r(refuse_others(Task) ), (
   in_r(refuse_others(Task)),
   no_r(contractor(_,_)) )).
```

ACC for Masters

```
enterACC(acme.org,
        [organisation(acme), role(master,task_distribution)])
tasks ?  out(announcement(Task))
wait(ExpireTime)
tasks ?  in(bids(Task,BidList))
Bid ← selectWinner(BidList)
tasks ?  out(award(Task,Bid))
tasks ?  in(result(Task,Result)
```

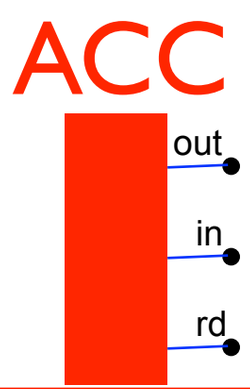



```
organisation(acme).
role(master,task_distribution).
can_do(init, tasks ? out(announcement(Task)), task_announced(Task)).
can_do(task_announced(Task), tasks ? rd(announcement(Task)),_).
can_do(task_announced(Task), tasks ? rd(bids(Task), )),_).
can_do(task_announced(Task), tasks ? in(bids(Task),BidList)),award_bidder(Task,BidList)).
can_do(award_bidder(Task,BidList), tasks ? out(award(Task,Bid)), get_result(Task,Bid)):-
     element(Bid,BidList).
can_do(get_result(Task,_)), tasks ? rd(result(Task,_)),_).
can_do(get_result(Task,_)), tasks ? in(result(Task,_)),init).
```

ACC for Workers

```
enterACC(acme.org,
    [organisation(acme), role(worker,task_distribution])
tasks ?  rd(announcement(Task))
MyBid ← evaluate(Task)
tasks ?  in(bid(Task,MyBid,Answer))
if (Answer=='awarded') {
    Result ← perform(Task)
    tasks ?  out(result(Task,Result))}
```

```
organisation(acme).
role(worker,task_distribution).
can_do(init, tasks ? rd(announcement(_)),_).
can_do(init, tasks ? in(bid(Task,_,awarded)),awarded(Task)).
can_do(awarded(Task), tasks ? out(result(Task,_),init)).
```



Outcome

Integration of security specification and enactment (access control in particular) within an organisation and coordination model

- conceptual integrity

Exploiting RBAC security properties in a MAS context

- separation of duty
- encapsulation of policy
- flexible and scalable management / administration
- supporting dynamic contexts
- supporting heterogeneity & openness

Ongoing work

Re-engineering some real world applications on top of TuCSoN already designed and developed without RBAC support

- Distributed Workflow Management Systems

Improving the prototype implementation (TuCSoN 2.0.0)

Defining a formal semantics (based on process algebra) for the RBAC architecture

- Integration with ACC formal model