Agent Coordination Context: From Theory to Practice

Alessandro Ricci, Mirko Viroli, Andrea Omicini
DEIS, Università degli Studi di Bologna
Sede di Cesena (Italy)

AT2AI Context

- MAS Middleware
  - Infrastructure support for coordination and organisation in agent societies

Outline

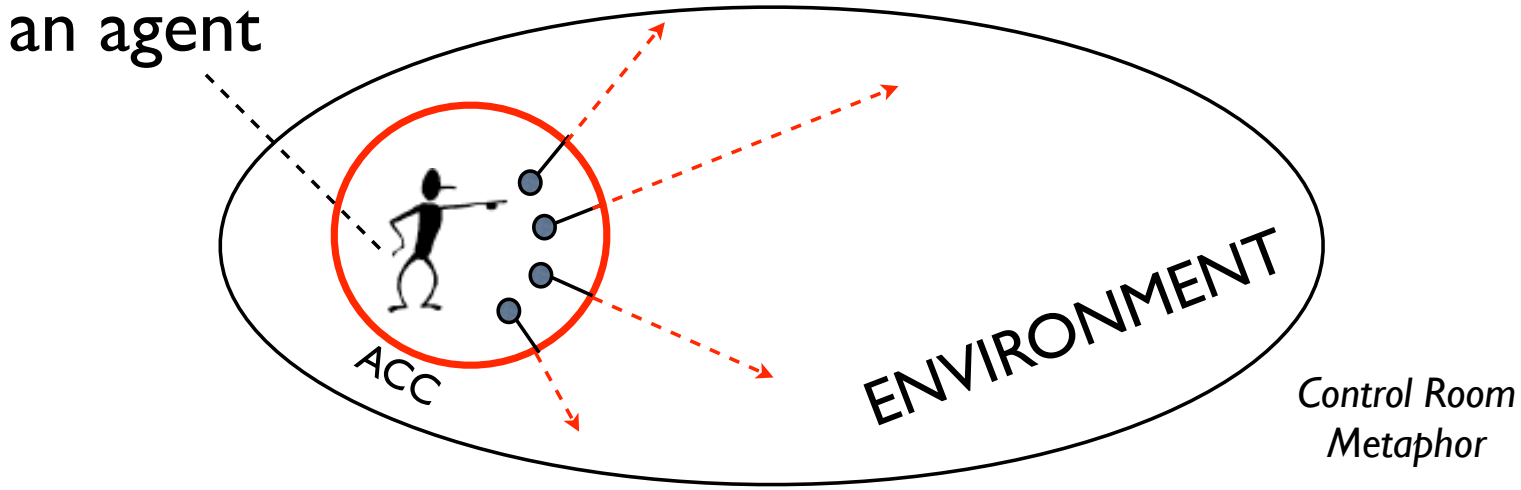
- The Context: Engineering MAS Organisation and Coordination in Synergy
- The Agent Coordination Context (ACC) Notion
- From Theory to Practice: ACC in TuCSoN
- Simple case study: Constrained CNP
- Future work

Engineering MAS Social Dimensions

- First class abstractions for MAS Organisation and Coordination
  - Specifying and managing organisation structures and rules
    - >> Roles, groups/societies, resources
    - >> Static and dynamic Relationships
  - Engineering agent interaction space
    - >> *E.g.: Coordination artifacts*
  - *Security from* Organisation & Coordination Synergy
- Continuum from design to *runtime*
  - "Keeps the abstractions alive" motto
  - Fundamental role of the infrastructures
    - >> From enabling to *governing* infrastructure

The Agent Coordination Context (ACC) Abstraction

- First class abstraction to model agent environment and agent/environment interaction
  - Runtime Interface (from inside-out)
    - >> What actions & perceptions the agent can do inside the environment
    - >> How agent actions affect the environment
  - Engineering abstraction



Control Room Metaphor

ACC for Organisations

- Representing the set of *roles* activated and played (dynamically) inside a structured organisation
- *Working session* inside an organisational context
  - Entering by acquiring an ACC
  - Using the ACC
  - Leaving by releasing the ACC
- *Formal contract* between the agent and the Institution (organisation)
- First class abstraction to map design abstractions (roles, groups,...) at runtime
  - Specification
  - Enactment/Enforcing

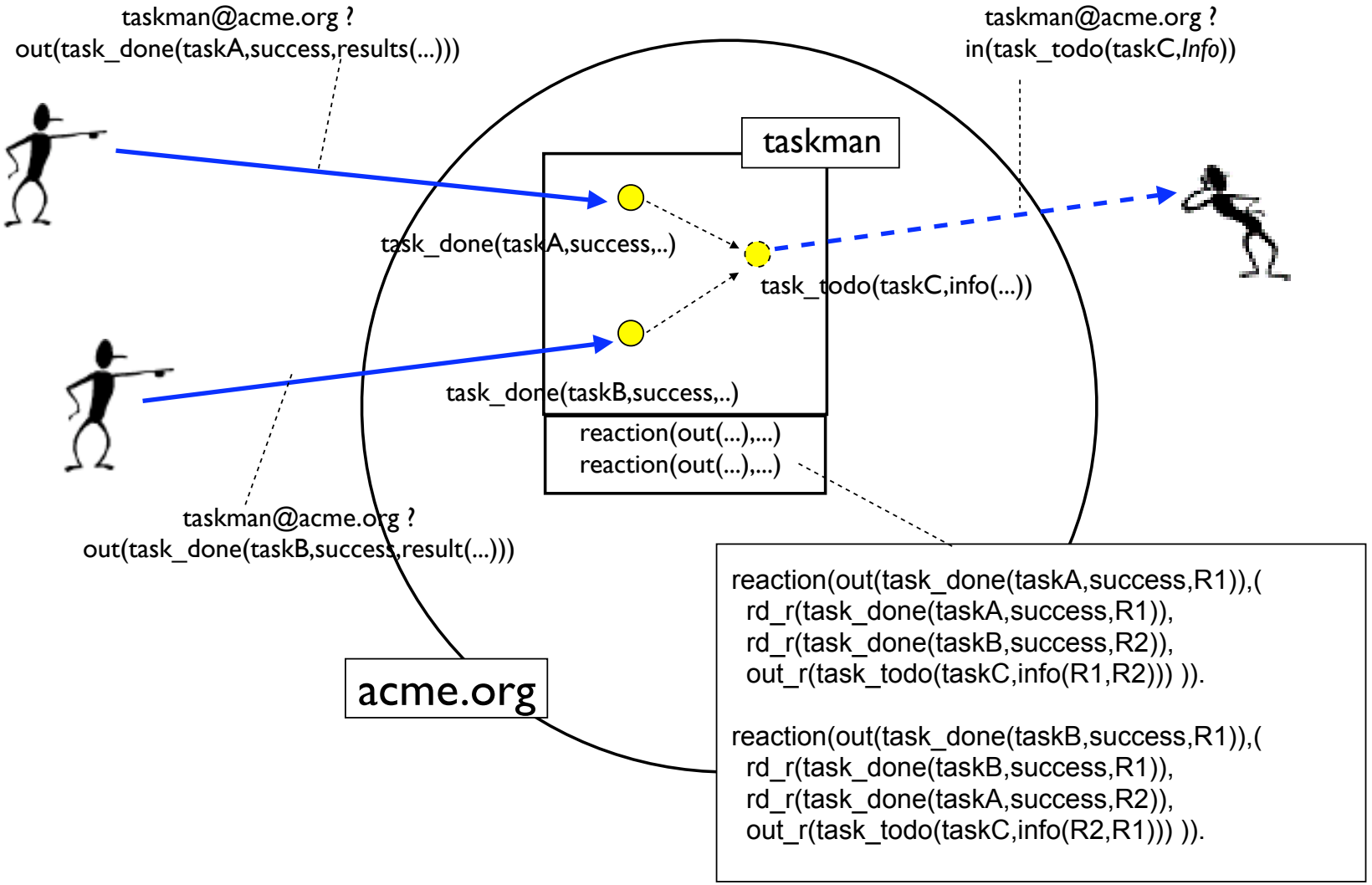
ACC for Security

- Modelling and enacting *Role-based Access Control*
  - RBAC-like approach
    - >> Reference model for engineering access control in complex information systems
  - Policies ruling agent actions / operations (access) on organisational resources
- RBAC benefits
  - Separation of Duty
  - Flexible Security Administration
  - Open & dynamic contexts
    - >> Dynamic activation/deactivation of role

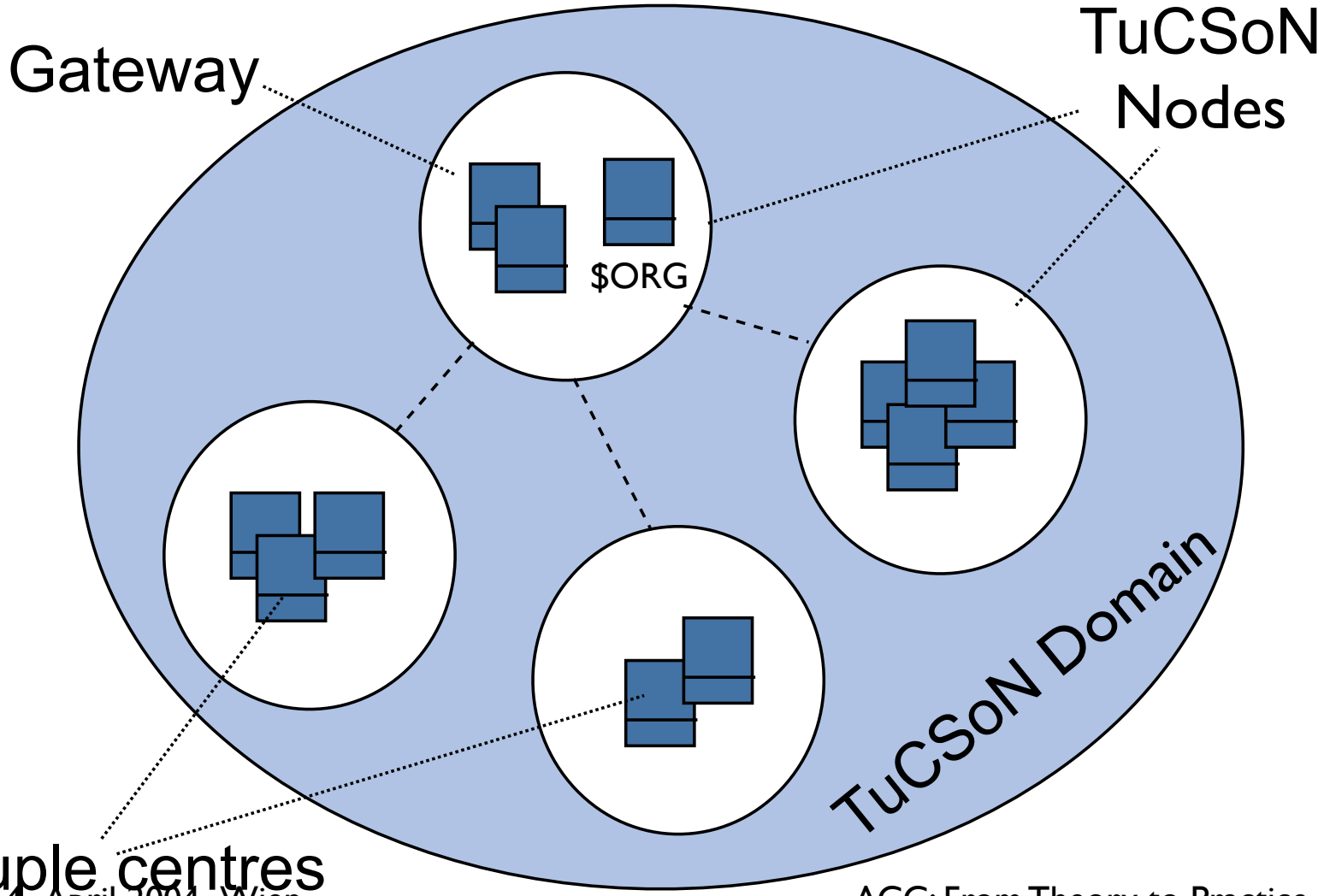
ACC in Practice: Extending TuCSoN

- TuCSoN Coordination Infrastructure
  - Supporting agent coordination by means of first class runtime coordination abstractions (*coordination artifacts)*
    - >> Tuple Centre coordination model
    - >> General purpose customisable coordination artifacts
      - From semaphors, synchronisers, to workflow engines
    - >> Activity Theory as a meta-model (Ricci et al, ESAW 2002)
  - Coordination artifacts as organisation resources shared and (concurrently) accessed/exploited by agents
  - Domain-based topology
    - >> Gateway node + Places nodes

Tuple Centres as Coordination Artifacts

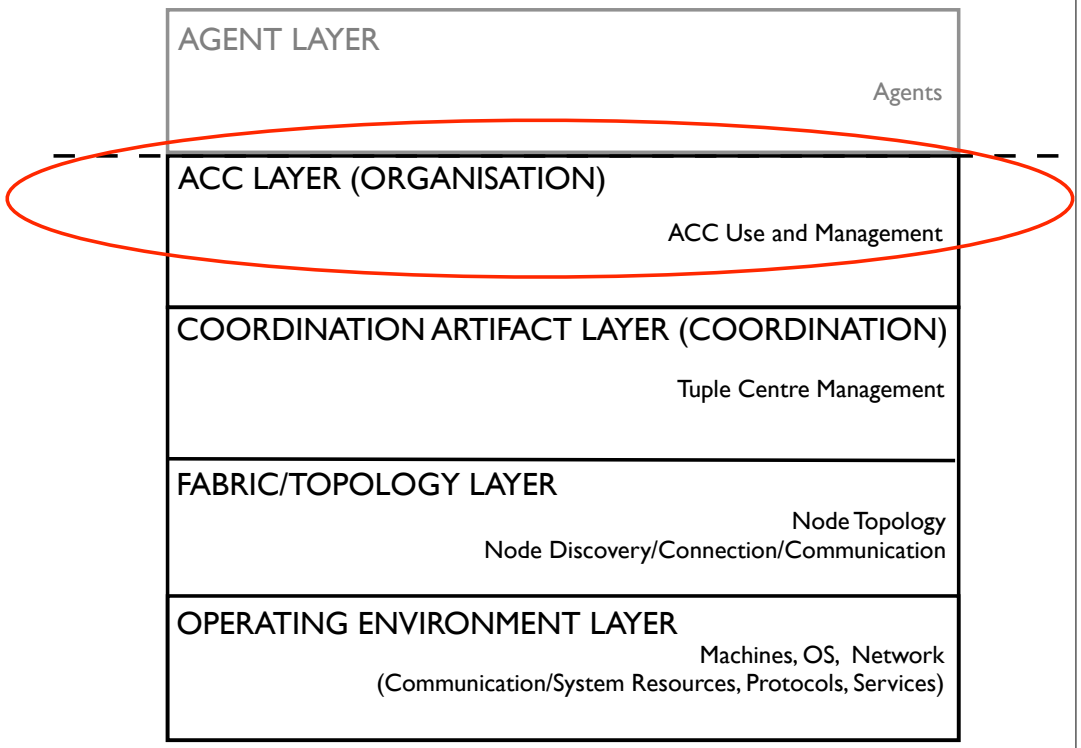


TuCSoN Space Overview



The ACC Extension

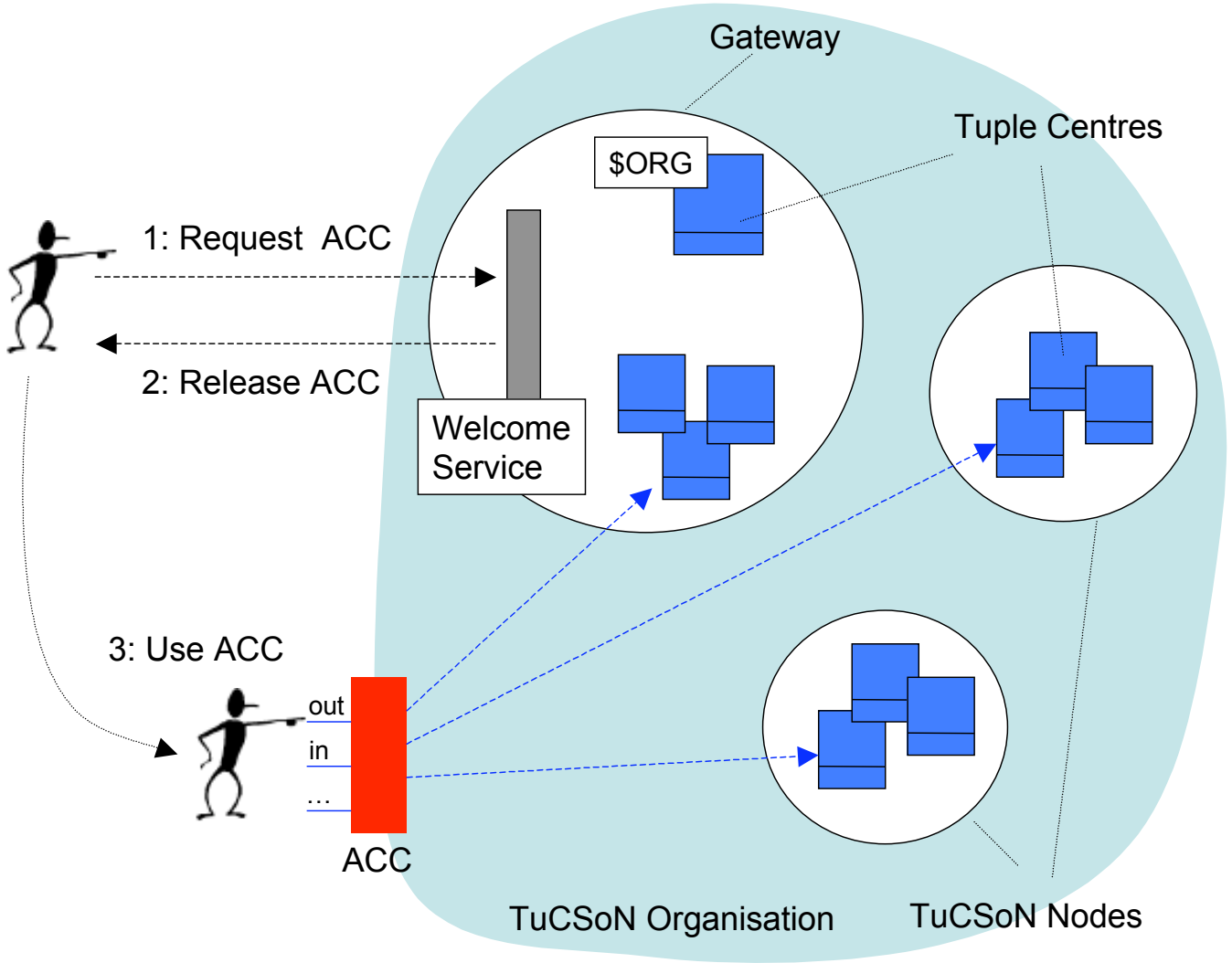
- Introducing an organisation layer
  - ACC for ruling access to tuple centres
  - Modelling organisation & security in synergy with coordination
    >> SODA methodology



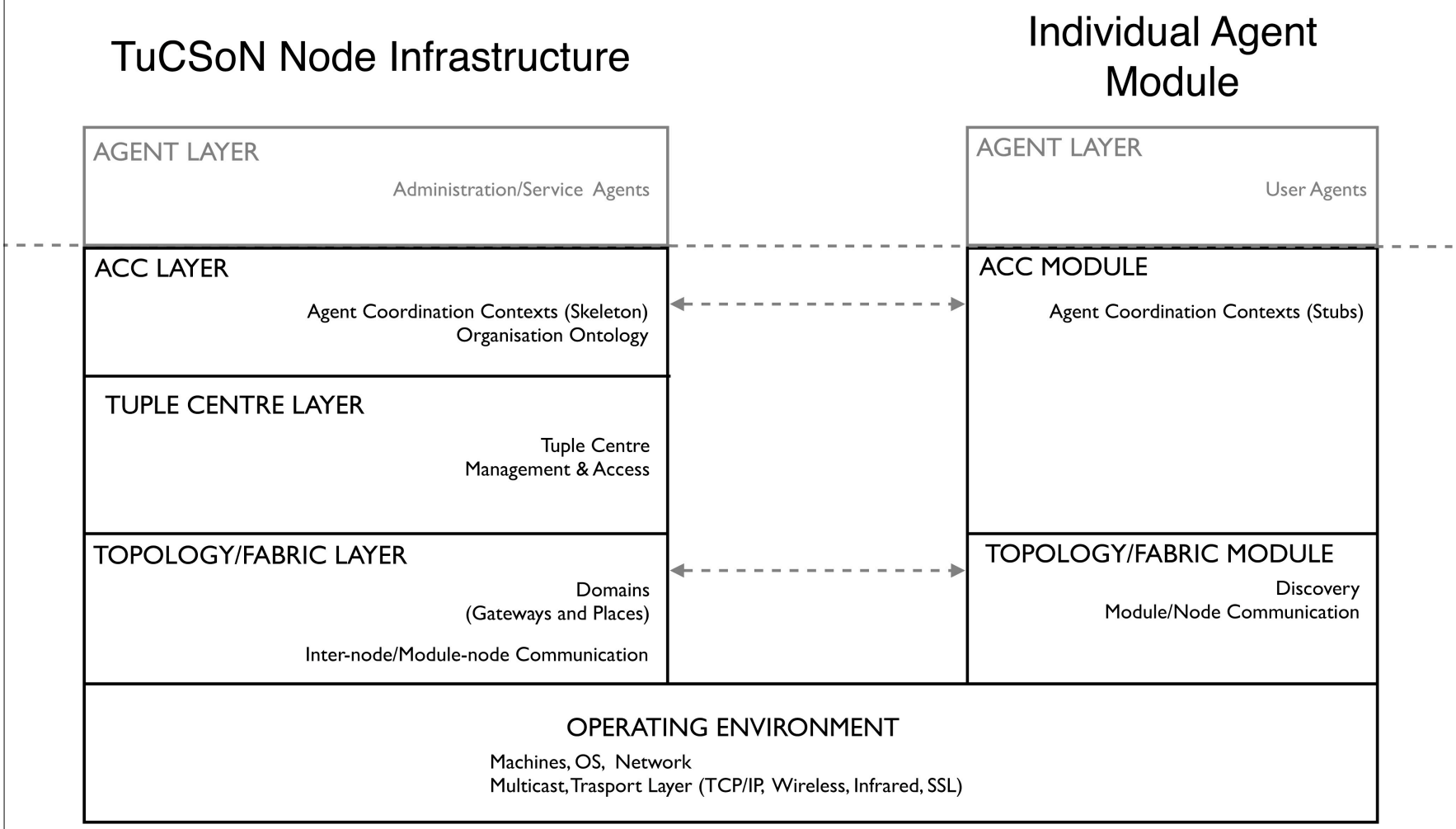
ACC Layer

- Managing ACCs dynamic creation, exploitation and destruction
- Support ACC *Negotiation*
  - An agent enters an organisation by requesting an ACC from a gateway node, specifying the roles to be activated
- Support ACC *Use*
  - Once obtained the ACC, the agent can participate to coordination activities by executing operations allowed by the ACC, by virtue of the activated roles

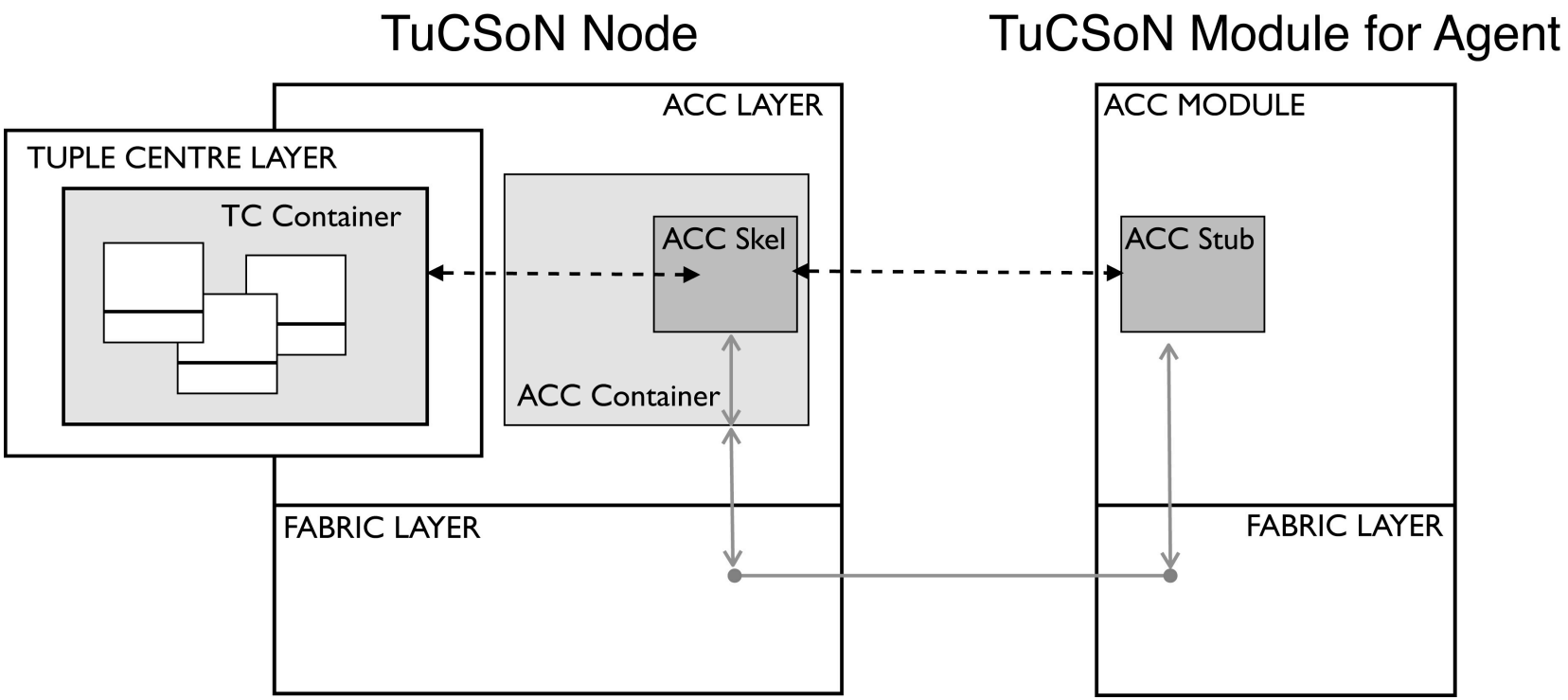
ACC Negotiation & Use



Architecture Overview



Design Details



The ACC as an Interface

- Operations to access tuple centres
  - Operations as first class objects

```
ACC Interface ::=
  doAction(+ACCAction,-ActionID )|
  action(+ActionID,?ACCAction )|
  actionState(+ActionID,?ActionState)|
  getACCSpec(-ACCSpec )|getACCState(-ACCState )

ACCAction ::=
  CoordinationOp | Tid ? CoordinationOp |
  Tid @ Node ? CoordinationOp

CoordinationOp ::=
  out(T)|in(TT)|rd(TT)|inp(TT)|rdp(TT)|
  set_spec(T)|get_spec(TT)
```

The ACC as a Contract

- Description of the relationships between the agent and the organisation
  - Agent Role(s) policy
- *State-full* runtime entity enforcing the contract
  - Representation of current role state
  - Notion of local time
  - Required to specify patterns/protocols

- 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(_)),_).
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(_)),_).
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).
```

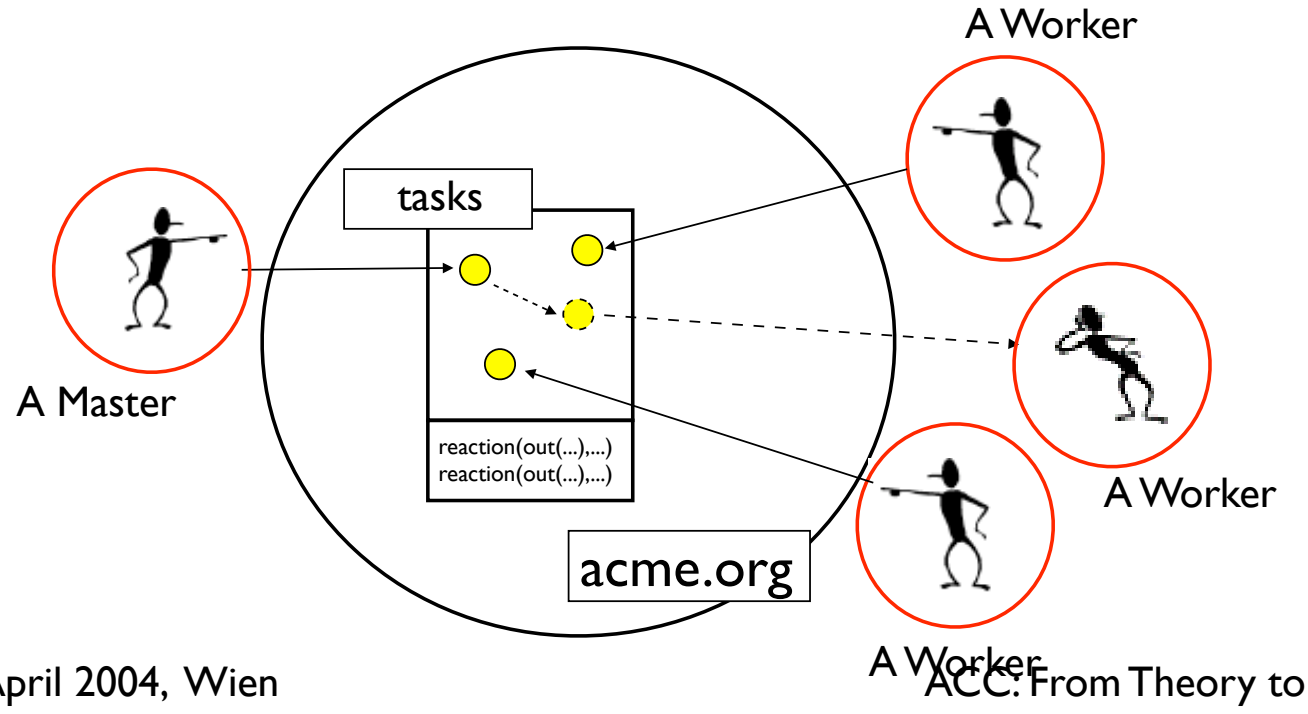
A more involved example: A Constrained Contract Net Protocol

- 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



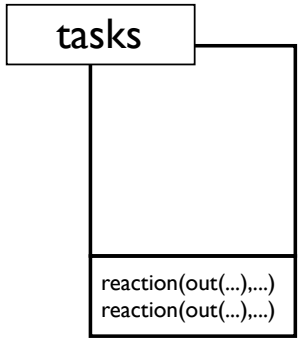
CNP Protocols & Coordination Laws

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)
```

ACC

```
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))}
```

ACC

```
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

- Separation of coordination concerns
  - ACC for ruling *individual* agent (inter)actions
    - >> Local/subjective perspective
    - >> Enforcing organisational constraints
  - Coordination artifacts for specifying/ruling *society* interactions
    - >> objective perspective
    - >> Enforcing coordination laws
- Engineering benefits
  - ACC as local (runtime) abstractions
  - Coordination artifacts as global (runtime) abstractions

Ongoing & Future Work

- Testing the approach with real world applications
  - Workflow Management Systems
- Exploiting ACC formal semantics based on process algebra for verification of properties
  - [Agent Coordination Contexts as Abstractions for the Formal Specification and Enactment of Coordination & Security Policies (Omicini, Ricci, Viroli) - Science of Compouter Progr. - to appear]
- Exploring the use of the ACC abstraction in other MAS infrastructures
  - JADE

AT2AI Some Related

- [*A Semantics for the Interaction of Agents with Coordination Artifacts* (Viroli, Ricci, Omicini)]
  - ACC and Coordination artifacts synergy
- [*An Ontological Approach to Harmonising Security Models for Open Services* (Juan Jim Tan et al.)]
  - ACC as a concrete way to realise (implement, deploy) the approach
- [*Combining Gaia and JADE for MAS Development* (Moraitis et al)]
  - Static vs. dynamic concept of roles & orgs
  - Close vs. Open approach to MAS org & security