

Engineering Multi-Agent Systems in LuCe



Andrea Omicini

LIA - DEIS - Università di Bologna

(joint work with Enrico Denti)

A place for LP...

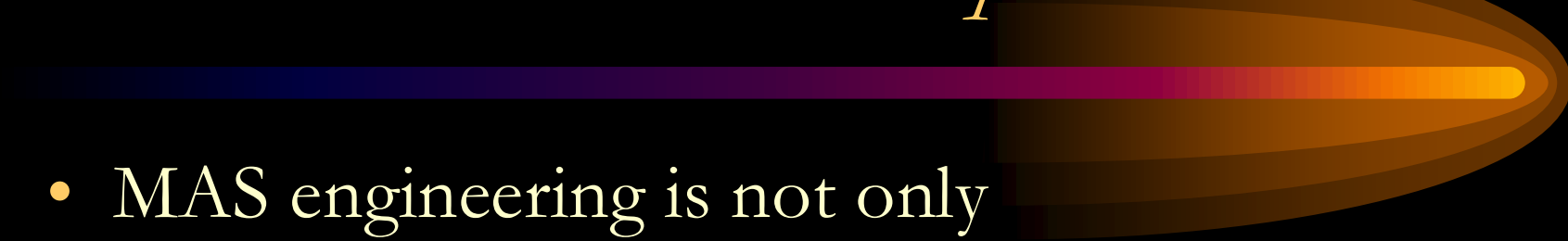


- OO-based SE
 - weakness in programming in the large
 - integration of OO and LP
- Component-based SE
 - small, heterogeneous components
 - LP for small specialised components
- Agent-based SE
 - declarative KR → beliefs
 - inference → autonomy/intelligence

LP as a natural source for...

- Computation languages
 - building logic-based agents
 - engineering individual agents
- Communication languages
 - representing agent-shared information
 - putting agents together in a MAS
- *Are we done with LP in MAS engineering?*

The point is that...



- MAS engineering is not only
 - engineering individual agents
 - putting agents together
- but also
 - engineering *agent societies* as first-class entities
 - in open, heterogeneous and unpredictable environments like the *Internet*

The goal of this work is to show how...

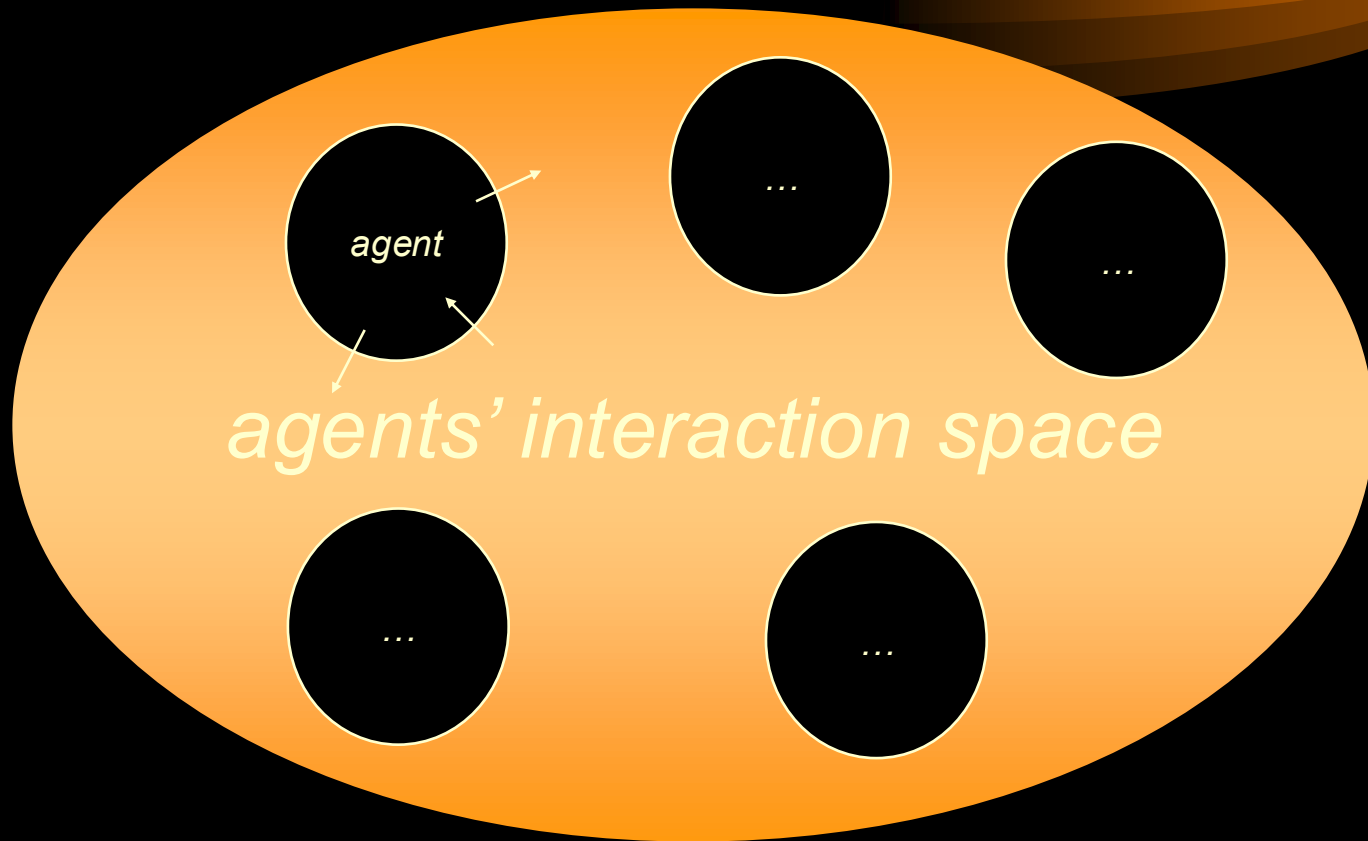


- Logic-based
 - abstractions
 - *logic tuple centre*
 - languages
 - ReSpecT
 - technologies
 - LuCe
- can effectively support the engineering of *agent societies* in *Internet-based MAS*

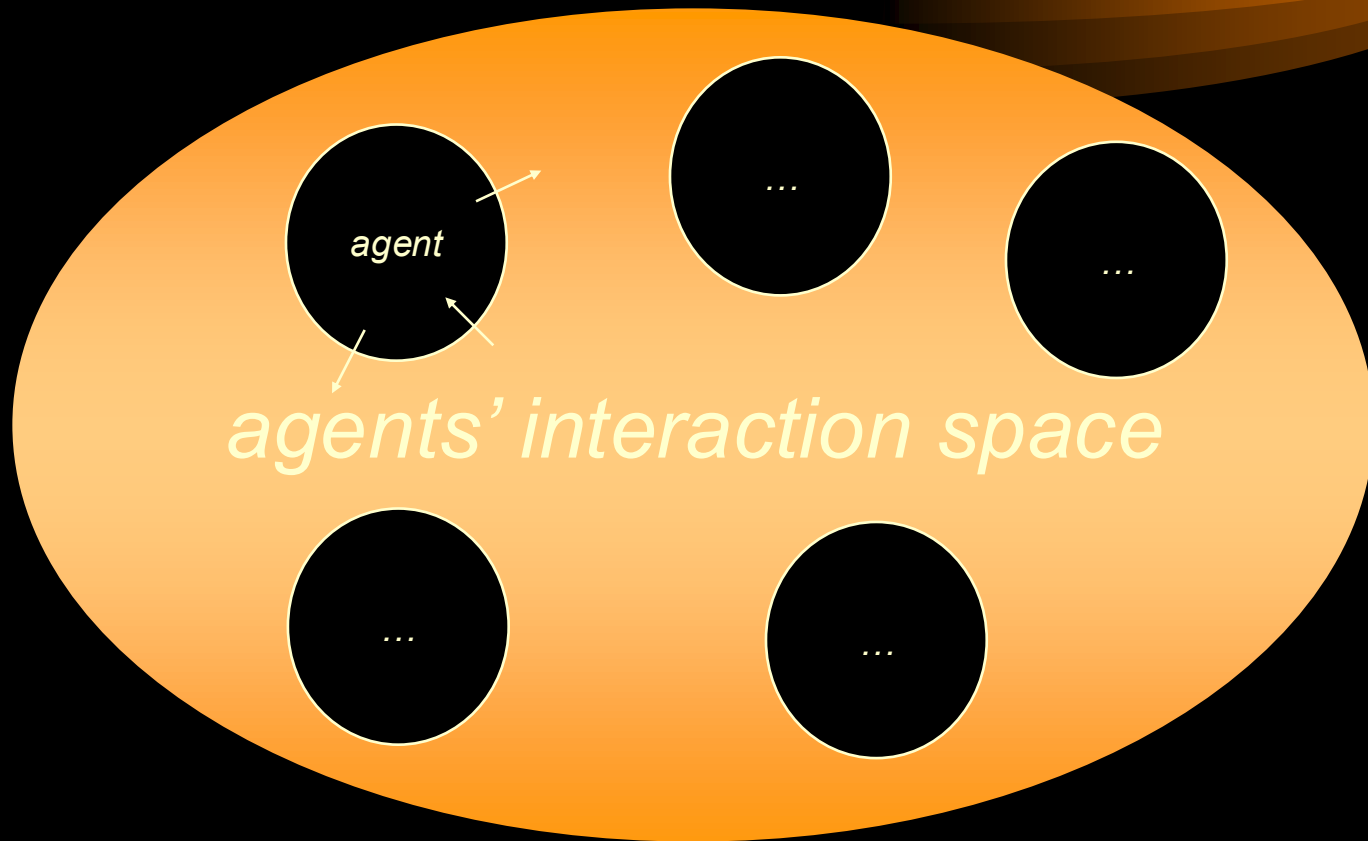
MAS engineering: design by task

- First step: defining
 - *individual tasks* – can be defined in terms of a single agent's task
 - *social tasks* – cannot be (or, are not naturally) defined in terms of any single agent's task
- Task-oriented design
 - delegation of *responsibility*
 - to individual agents
 - to societies of agents

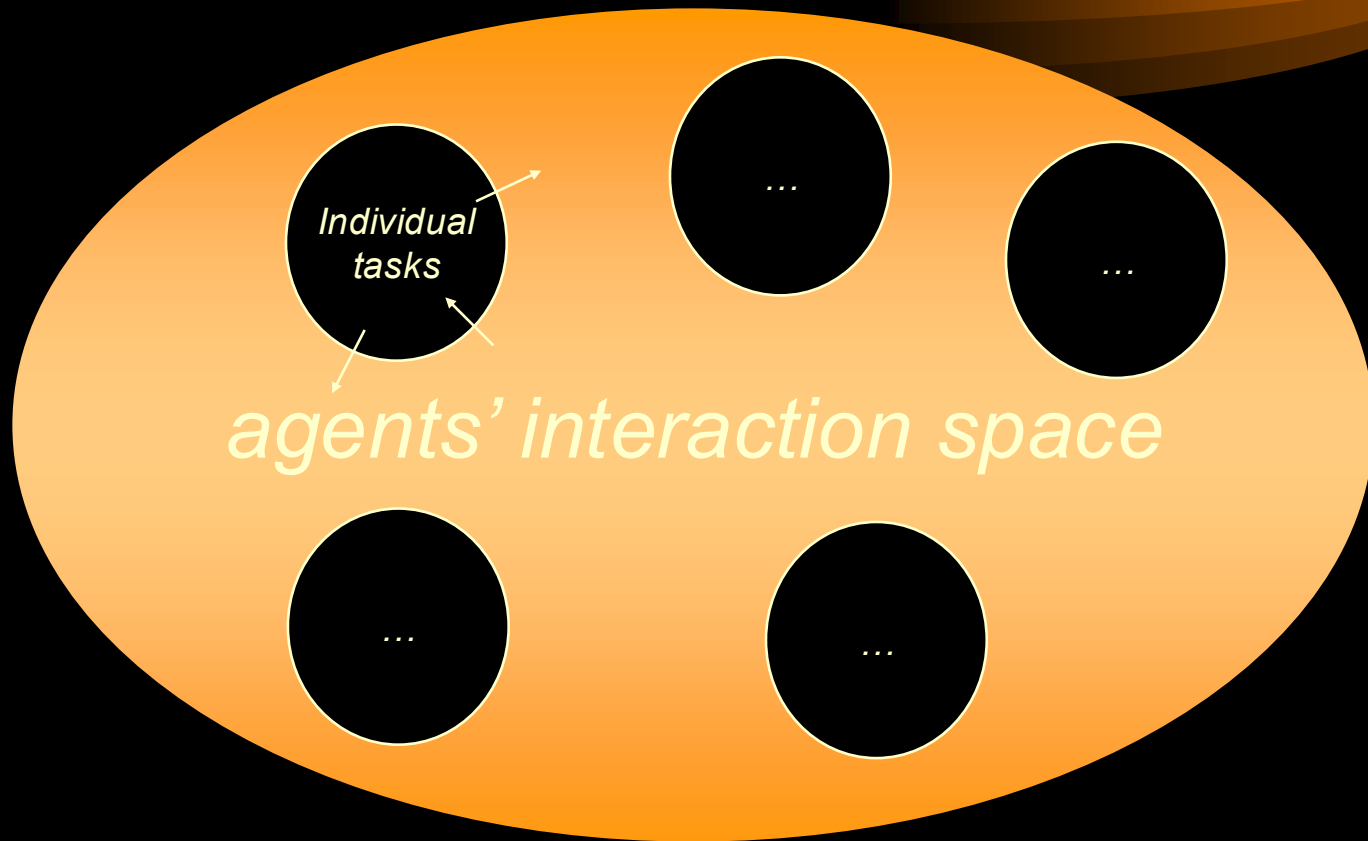
Depicting a MAS



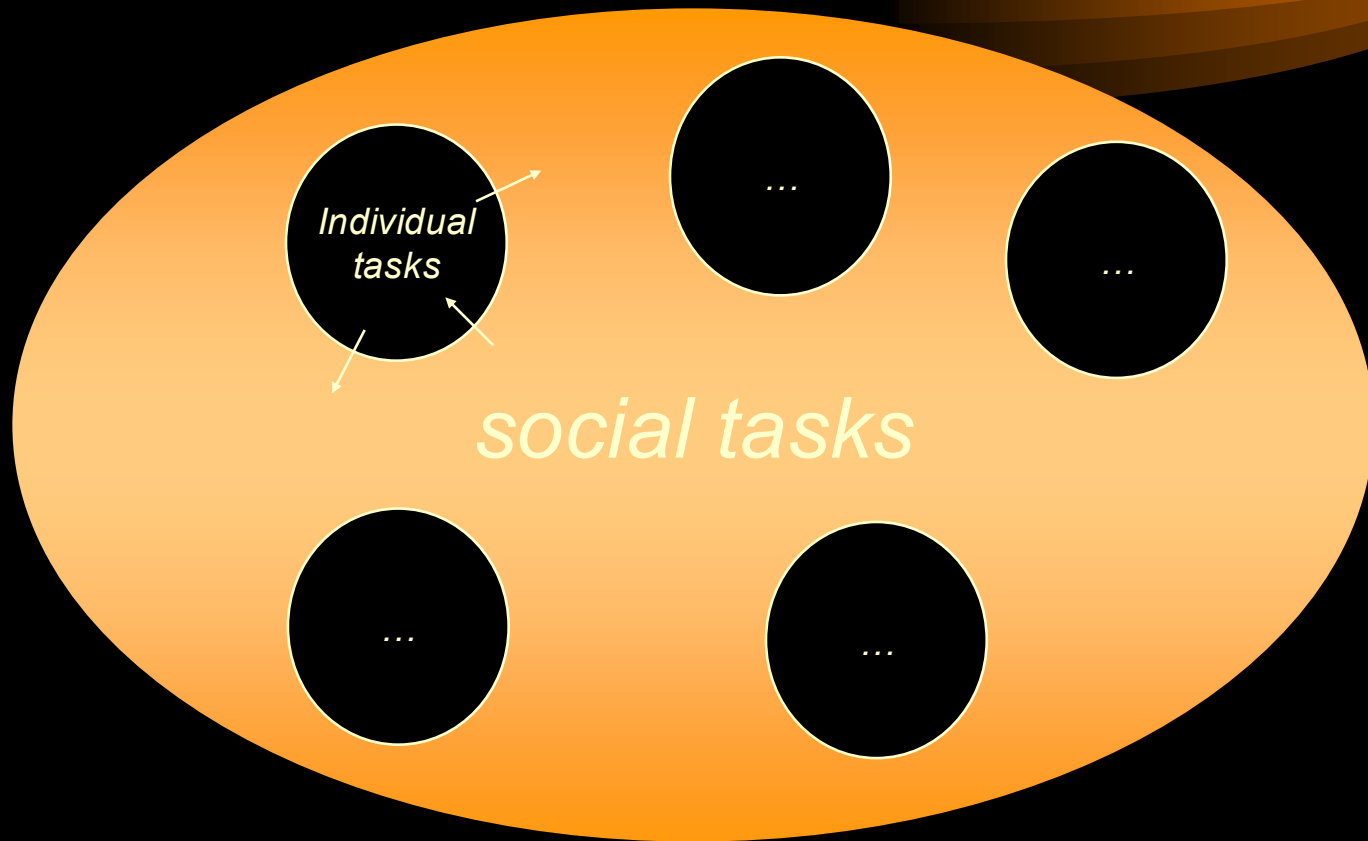
A place for a task



A place for a task



A place for a task



How do we build social tasks?



- typical solution
 - *ad hoc* meta-level agent
 - elegant? effective? necessary? feasible?
 - a *symptom*, rather than a solution
 - no suitable specific abstractions
- a more effective approach
 - exploiting suitable *coordination models* and *systems* to effectively shape the agents' interaction space

A coordination model...

- is
 - a *conceptual framework* to *model* the space of (agent) interaction
- can be described in terms of
 - *coordinables* - the entities whose mutual interaction is ruled by the model
 - *coordination media* - the abstractions enabling and ruling the interaction among the coordinables
 - *coordination laws* - the rules governing the interaction among coordinables as well as the behaviour of the coordination media

Building agent societies



- agent societies
 - built around coordination media
- social rules
 - expressed in terms of coordination laws
- social behaviours
 - results from the behaviour of the coordination media in response to agents' interaction protocols

Agent coordination via tuple centres

- information-driven coordination
 - agents coordinate by producing/acquiring/consuming information
 - standard tuple space interface
- ruling agent interaction/communication
 - by controlling agent-triggered communication events
 - by programming tuple centre's behaviour in response to communication events in terms of *reaction specification tuples*

Tuples in tuple centres

- communication
 - in term of *(ordinary) tuples*
 - defining the tuple centre's *content*
 - in the tuple centre's *tuple space*
- coordination
 - in terms of *specification tuples*
 - defining the tuple centre's *behaviour*
 - in the tuple centre's *specification space*

Logic tuple centres

- communication through *logic tuples*
 - looking like a logic tuple space (*nothing new here*)
 - tuple space as the *theory of the communication*
- coordination through logic tuples
 - specification tuples as logic tuples, too
 - specification space as the *theory of coordination*

ReSpecT

- Reaction Specification Tuples
- a logic-based language defining both
 - a *reaction language*, to define computations within tuple centres, called *reactions*
 - a *specification language*, to associate reactions to agent-triggered communication events

An example (a)

- the tuple centre **table** contains

- tuples

chop(1)

chop(2)

in its tuple space

- tuple

reaction(**in**(**chops**(A,B) , (pre ,
in_r(**chop**(A)) , in_r(**chop**(B)) ,
out_r(**chops**(A,B)))) .

in its specification space

An example (b)

- agent **philo** tries to get its chopsticks
 - by *asking* to consume tuple **chops**(1,2)
 - by means of a
table?in(chops(1,2))
invocation

An example (c)

- tuple centre **table** reacts

- according to its behaviour specification

```
reaction(in(chops(A,B), (pre,  
                        in_r(chop(A)), in_r(chop(B)),  
                        out_r(chops(A,B)))).
```

- by *executing* reaction

```
pre, in_r(chop(1)), in_r(chop(2)),  
out_r(chops(1,2))
```

as a *transaction*, which is successful, so

- it *replaces* **chop(1)** and **chop(2)** with **chops(1,2)** in its tuple space

An example (d)

- agent **phil** actually gets its chopsticks
 - by actually *consuming* tuple
chops (1,2)
from **table**
 - and *unifying* it with **chops (A,B)**

Tuple centres: the benefits

- all the benefits of tuple-based coordination
- social rules encapsulated into coordination media
 - independent, *reusable* coordination component
 - *design uncoupling* between individual and societies
 - *social intelligence* outside the individuals

ReSpecT tuple centres: the benefits

- all the benefits of logic tuple spaces
- social rules as **ReSpecT** logic tuples
 - simple yet expressive
 - complex coordination laws by means of few tuples
 - Turing-powerful
 - any computable coordination law in a tuple centre
 - reaction execution model promotes *incrementality*
 - *inspectable/modifiable* behaviour specification
 - reasoning about communication and coordination
 - self-organising MAS

The LuCe coordination model

- a Linda-like coordination language
 - `out`, `in`, `rd`, `inp`, `rdp`
- a multiplicity of **ReSpecT** tuple centres
 - addressed by name (a ground term) transparently
- coordinables
 - conceptually, any agent accomplishing the coordination language
 - pragmatically, C, C++, Java, Prolog agents
 - heterogeneity addressed from the very beginning

Coordination technology

- *implementing* a coordination model is not enough
 - supporting the model's metaphors does not suffice
- a *coordination system* should *exploit* metaphors
 - to provide ad-hoc, specialised development tools
- *enabling* developers
 - to choose the desired abstraction level over the MAS' space of interaction
- the role of *coordination technology*

Coordination technology: LuCe inspectors



- tuple centre: sets
 - T of the ordinary tuples
 - W of the pending queries
 - S of the specification tuples
- T Inspector
 - communication from a data-driven viewpoint
- W Inspector
 - communication from a control-driven viewpoint
- S Inspector
 - coordination viewpoint
- providing *developers* with the most effective *level of abstraction* over a MAS

Towards a methodology

- *abstractions* drive the *design*
 - social tasks built around coordination media
 - ReSpecT tuple centres
- *technologies* drive the *development*
 - individual (observable) & social behaviours developed and tested via coordination tools
 - LuCe inspectors
- both impact on the process of *incremental refinement*

System technology: the LuCe run-time



- Java-based
 - easy integration with Internet tools (e.g. browser)
- Prolog-based
 - Prolog engine as a Java thread
 - **ReSpecT** Virtual Machine in a snap
 - intrinsic support for heterogeneity
 - “declarative” and “imperative” agents

Conclusions

- LP is a good source not only for
 - *computation languages*, to engineer individual agents
 - *communication languages*, to put agents together in a MAS
- but also for
 - *coordination components*, to be used as the core for engineering agent societies
 - *coordination technologies*, well-coping with real-world application environments like the Internet

Home pages



- ReSpecT

<http://lia.deis.unibo.it/Research/ReSpecT/>

- LuCe

<http://lia.deis.unibo.it/Research/LuCe/>

- this paper

<http://lia.deis.unibo.it/Research/DO-MAS99/>