(Please:) **Have ReSpecT for LogOp** (!)

Ronaldo Menezes
Department of Computer Science
Florida Institute of Technology
rmenezes@cs.fit.edu

Andrea Omicini Mirko Viroli
DEIS
Università degli Studi di Bologna a Cesena
{aomicini,mviroli}@deis.unibo.it

Conceptual problem

- Lots of results from the field of *objective* coordination models (CM) and languages
  - Lots of different models and technologies
  - Apparently, general purpose, but
  - Actually, providing solutions for classes of coordination problems
- Given a CM
  - Which (classes of) problems does it effectively solve?
  - How does it relate/compare to other CM?
- Given a complex coordination problem
  - Does it exist one CM that solves it *effectively, as a whole*?
  - If not, how can different CMs be combined to do it?

The Case Study: ReSpecT vs. LogOp

- Linda-based models for complex distributed environments
  - LogOp & ReSpecT
- Both extends the expressive power of the Linda coordination space, where
  1. There might be multiple spaces, but accessible one at a time
  2. The behaviour of a space is fixed once and for all
- LogOp
  - Introduces new primitives, without altering tuple spaces
  - Addresses (1)
- ReSpecT
  - Extends tuple spaces to tuple centres, with no new primitives
  - Mainly addresses first (2), and then (1)
- Different solutions, different levels of abstractions

The basic Linda formal model

- Monospace

$$\pi ::= out \mid in \mid rd, \quad \alpha ::= \pi(x), \quad P ::= 0 \mid \alpha.P$$

$$S ::= 0 \mid x \mid P \mid S \parallel S$$

$$\begin{aligned} out(x).P \parallel S &\longrightarrow P \parallel S \parallel x \\ rd(x).P \parallel S \parallel x &\longrightarrow P \parallel S \parallel x \\ in(x).P \parallel S \parallel x &\longrightarrow P \parallel S \end{aligned}$$

- Multispace: $x \Rightarrow <ts,x>$

LogOp

- Abstract syntax of LogOp primitives and processes

$$
\begin{array}{lcl}
\pi & ::= & out \mid in \mid rd \\
\lambda & ::= & ts \wedge \ldots \wedge ts \mid ts \otimes \ldots \otimes ts \mid ts \vee \ldots \vee ts \\
\alpha & ::= & \pi(\lambda, x) \\
P & ::= & 0 \mid \alpha.P
\end{array}
$$

  - One single LogOp operation can address more tuple spaces
  - Through composition via logical operators

- Abstract model of a LogOp space

$$S ::= 0 \mid \langle ts, x \rangle \mid P \mid S \parallel S$$

  - A "soup" of processes and localised tuples

Informal Semantics

- These rules say what tuples may be exchanged according to logical operators (*admissible* tuples)

$$\sigma(\pi(\bigwedge_{ts \in Ts} ts, x), \prod_{ts \in Ts} \langle ts, x \rangle)$$

$$\sigma(\pi(\bigotimes_{ts \in Ts} ts, x), \langle ts_0, x \rangle) \quad \text{with} \quad ts_0 \in Ts$$

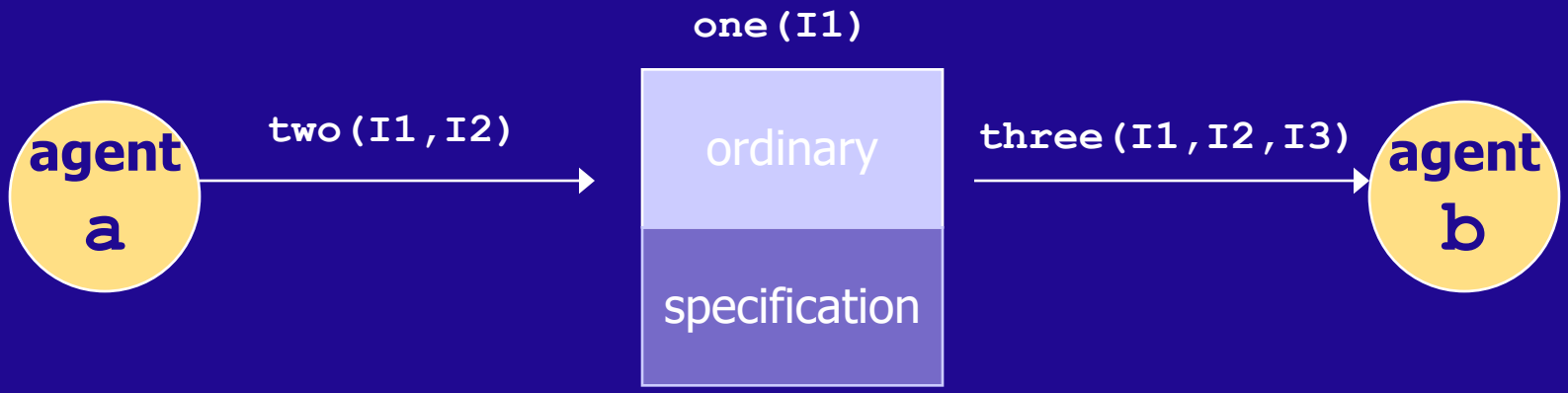
$$\sigma(\pi(\bigvee_{ts \in Ts} ts, x), \prod_{ts' \in Ts'} \langle ts', x \rangle) \quad \text{with} \quad \{\} \subset Ts' \subseteq Ts$$

- AND: transfer tuples from/to "all" spaces
- XOR: transfer a tuple from "one" space
- OR: transfer tuples from "some" space (1 or more)

ReSpecT

- Tuple centre
  - Tuple space + programmable behaviour
    - Default behaviour = tuple space
  - Tuple space + specification space
- Behaviour
  - In response to interaction events
    - *Atomic*
  - Extending the behaviour of basic Linda primitives
    - No new primitives added
  - In terms of *reactions*
    - Specified in terms of (logic) tuples
- Reactions
  - Access triggering events
  - Access/modify tuple space
  - Compute
  - Send tuples to other tuple centres
    - `out_tc`
  - *Atomically*
    - 2 levels of atomicity

Showing some ReSpecT




```
reaction(out(two(I1,I2)), (
    current_agent(a), in_r(two(I1,I2)),
    out_r(one(I1)), out_r(one(I2))
))
```

LogOp upon ReSpecT

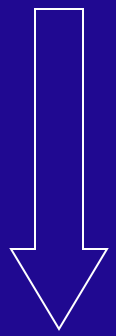
- ReSpecT
  - Empowers the coordination space with programmability
  - Sort of assembly language for handling interaction
- LogOp
  - Empowers agents with more expressive coordination primitive
  - Sort of high-level coordination language
- Possible architecture of a LogOp-like language upon ReSpecT
  - Special tuples in the standard primitives to represent LogOp operations
    - `in(and(TCList, Tuple, Answer))`
  - A `logOp` tuple centre repersenting the LogOp kernel
    - `logOp ? in(and(TCList, Tuple, Answer))`
  - Other tuple centres simply instrumented to deal with `logop` "requests"

Strong vs. Weak Synchrony

- Different views of the coordination space
  - As a synchronous multi-space
    - Strong synchrony
  - As multi-synchronous space
    - Weak synchrony
  - Example: `in` + AND
- LogOp *seems* to promote strong synchrony
  - That is why we "pleased" you…
  - Two interpretations / possible implementations available
- ReSpecT *intentionally* promotes weak synchrony
  - Tuple centres loosely coupled via `out_tc`
  - Strong synchrony requires lots of work
    - `in_tc`, `rd_tc`?

Strong Synchrony

- Which *precise* semantics?
  - Which dynamics?
  - Which atomicity props?
  - What about interference?
- Coordination space as a synchronous whole
  - À la Linda, in a sense



$$
\begin{aligned}
out(x).P \parallel S &\longrightarrow P \parallel S \parallel x \\
rd(x).P \parallel S \parallel x &\longrightarrow P \parallel S \parallel x \\
in(x).P \parallel S \parallel x &\longrightarrow P \parallel S
\end{aligned}
$$

$$\frac{\sigma(out(\lambda, x), S')}{out(\lambda, x).P \parallel S \longrightarrow P \parallel S \parallel S'}$$

$$\frac{\sigma(rd(\lambda, x), S')}{rd(\lambda, x).P' \parallel S \parallel S' \longrightarrow P \parallel S \parallel S'}$$

$$\frac{\sigma(in(\lambda, x), S')}{in(\lambda, x).P \parallel S \parallel S' \longrightarrow P \parallel S}$$

Weak Synchrony

- Can be defined as a mapping from LogOp primitives
  - nondeterministic choices (+) between sequences (;) of Linda primitives

$$
\begin{aligned}
\left|\pi(\bigwedge_{ts\in Ts} ts, x)\right|_{\mathcal{O}} &\triangleq \sum_{ts\in Ts} \pi(ts,x); \left|\pi(\bigwedge_{ts'\in Ts\setminus\{ts\}} ts', x)\right|_{\mathcal{O}} \\
\left|\pi(\bigotimes_{ts\in Ts} ts, x)\right|_{\mathcal{O}} &\triangleq \sum_{ts\in Ts} \pi(ts,x) \\
\left|\pi(\bigvee_{ts\in Ts} ts, x)\right|_{\mathcal{O}} &\triangleq \sum_{ts\in Ts} \left(\pi(ts,x) + \pi(ts,x); \left|\pi(\bigvee_{ts'\in Ts'\setminus\{ts\}} ts', x)\right|_{\mathcal{O}}\right)
\end{aligned}
$$

$$
\begin{aligned}
\left|out(ts_1 \wedge ts_2, x)\right|_{\mathcal{O}} &= out(ts_1,x); out(ts_2,x) + out(ts_2,x); out(ts_1,x) \\
\left|in(ts_1 \otimes ts_2, x)\right|_{\mathcal{O}} &= in(ts_1,x) + in(ts_2,x) \\
\left|rd(ts_1 \vee ts_2, x)\right|_{\mathcal{O}} &= rd(ts_1,x) + rd(ts_1,x); rd(ts_2,x) + \\
&\quad + rd(ts_2,x) + rd(ts_2,x); rd(ts_1,x)
\end{aligned}
$$

Weak vs. Strong: An example

$$S = in(ts_1 \wedge ts_2, x).out(ts_3, x) \parallel in(ts_1 \wedge ts_2, x).out(ts_3, x) \parallel \langle ts_1, x \rangle \parallel \langle ts_2, x \rangle$$

$$S \longrightarrow out(ts_3, x) \parallel in(ts_1 \wedge ts_2, x).out(ts_3, x)$$
$$\longrightarrow in(ts_1 \wedge ts_2, x).out(ts_3, x) \parallel \langle ts_3, x \rangle$$

$$S \longrightarrow_w in(ts_2).out(ts_3, x) \parallel in(ts_1 \wedge ts_2).out(ts_3, x) \parallel \langle ts_2, x \rangle$$
$$\longrightarrow_w in(ts_2).out(ts_3, x) \parallel in(ts_1).out(ts_3, x)$$

- Deadlock?
  - Closed systems: yes
  - Open systems: no way!

Lessons Learned

- Distributed computing still a mess even for experienced researchers
- Please be *both* formal *and* concrete
- Lots still to be learned in the CM field…
  - Several different levels of abstraction upon the interaction space
  - Many different approaches effective at different levels
  - Can be used together with *awareness*
- … and from there brought *everywhere* else
  - CSCW
  - WfMS
  - Agents
  - Web
  - Security
  - …