

# Interaction, Complexity, Coordination

## Nature-inspired Coordination Models for Complex Distributed Systems

Andrea Omicini  
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)  
ALMA MATER STUDIORUM – Università di Bologna

Fribourg, Switzerland  
20 November 2014



- 1 Premises
- 2 Interaction & Coordination
- 3 Classes of Coordination Models
- 4 Tuple-based Coordination Models

# Outline

- 1 Premises
- 2 Interaction & Coordination
- 3 Classes of Coordination Models
- 4 Tuple-based Coordination Models

# Scenarios for Concurrent / Distributed Systems

## Action & interaction

- Concurrency / parallelism
  - *Multiple* independent activities / loci of control
  - *Acting* simultaneously
  - Processes, threads, actors, active objects, *agents*...
- Distribution
  - Activities running on *different* and heterogeneous execution *contexts* (machines, devices, ...)
- Social interaction
  - *Dependencies* among activities
  - Collective goals involving activities *coordination* / cooperation
- Environment interaction
  - Interaction with external *resources*
  - Interaction within the *time-space* fabric
  - Interaction in *context*

# Complexity & Interaction I

An essential source of complexity for computational systems is  
**interaction**

[Goldin et al., 2006]

The power of interaction [Wegner, 1997]

*Interaction is a **more powerful** paradigm than rule-based **algorithms** for computer-based solving, overtiring the prevailing view that all computing is expressible as algorithms.*

# Complexity & Interaction II

## Intelligence & interaction [Brooks, 1991]

*Real computational systems are not rational agents that take inputs, compute logically, and produce outputs. . . It is hard to draw the line at what is **intelligence** and what is **environmental interaction**. In a sense, it does not really matter which is which, as all **intelligent systems** must be **situated** in some world or other if they are to be useful entities.*

## A conceptual framework for interaction [Milner, 1993]

*. . . a **theory of** concurrency and **interaction** requires a **new conceptual framework**, not just a refinement of what we find natural for sequential [algorithmic] computing.*

# Complexity & Interaction III

## Interactive computing [Wegner and Goldin, 1999]

- Finite computing agents that interact with an environment are shown to be more expressive than Turing machines according to a notion of expressiveness that measures problem-solving ability and is specified by observation equivalence
- Sequential interactive models of objects, agents, and embedded systems are shown to be more expressive than algorithms
- Multi-agent (distributed) models of coordination, collaboration, and true concurrency are shown to be more expressive than sequential models

# (Non) Algorithmic Computation I

## Elaboration / computation

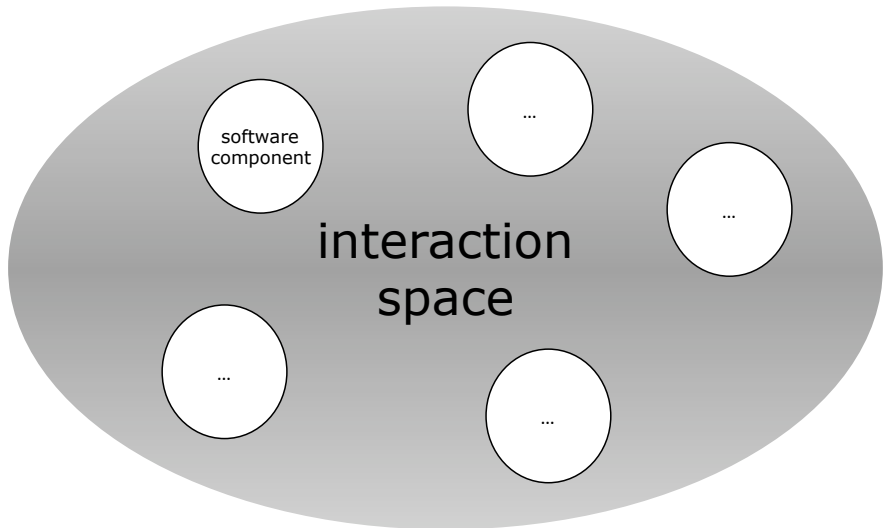
- Turing Machine (TM)
  - gets an input, elaborates it, throws an output
  - no interaction during computation
- Black-box algorithms
- Church's Thesis and computable functions
  - in short, a function is computable *iff* can be computed by a TM
  - so, all computable functions are computable by a TM

# (Non) Algorithmic Computation II

## Beyond Turing Machines

- Turing's *choice machines* and *unorganised machines* [Wegner and Goldin, 2003]
- Wegner's Interaction Machines [Goldin et al., 2006]
- Examples: AGV, Chess oracle [Wegner, 1997]

# The Space of Interaction



# Components of an Interactive System

## What is a component of an interactive system?

- A computational abstraction characterised by
  - an independent computational activity
  - I/O capabilities
- Two independent dimensions
  - elaboration / *computation*
  - *interaction*

# Which Components?

## Open systems

- No hypothesis on the component's life & behaviour

## Distributed systems

- No hypothesis on the component's location & motion

## Heterogeneous systems

- No hypothesis on the component's nature & structure

# Basics of Interaction

## Component model

A simple component exhibits

**Computation** Inner behaviour of a component

**Interaction** Observable behaviour of a component as *input* and *output*

## Coupling across component's boundaries

- Control
- Information
- Time & Space – internal / computational vs. external / physical

## Information-driven interaction

**Output** shows part of its state outside

**Input** bounds a portion of its own state to the outside

# Compositionality vs. Non-compositionality

An informal view by [Wegner, 1996]

## Compositionality

- Sequential composition  $P1; P2$
- $behaviour(P1; P2) = behaviour(P1) + behaviour(P2)$

## Non-compositionality

- Interactive composition  $P1|P2$
- $behaviour(P1|P2) = behaviour(P1) + behaviour(P2) + \mathbf{interaction}(P1, P2)$
- Interactive composition is more than the sum of its parts

# Non-compositionality

## Issues

- Compositionality vs. formalisability
  - A formal notion of model is required for stating any compositional property
  - However, formalisability does not require compositionality, and does not imply predictability
  - *Partial formalisability* may allow for proof of properties, and for partial predictability
- Emergent behaviours
  - Fully-predictable / formalisable systems do not allow by definition for emergent behaviours
- Formalisability vs. expressiveness
  - Less / more formalisable systems are (respectively) more / less expressive in terms of potential behaviours

# Basic Engineering Principles for Interactive Systems I

## Principles

- Abstraction
  - Problems should be faced / represented at the most suitable level of abstraction
  - Resulting “abstractions” should be expressive enough to capture the most relevant problems
  - Conceptual integrity
- Locality & encapsulation
  - Design abstractions should embody the solutions corresponding to the domain entities they represent
- Run-time vs. design-time abstractions
  - Incremental change / evolution
  - On-line engineering [Fredriksson and Gustavsson, 2004]
  - (Cognitive) Self-organising systems [Omicini, 2012]

# Basic Engineering Principles for Interactive Systems II

## Issues

- What is the most suitable abstraction level to deal with interaction?
- Which sort of dedicated SE abstractions could encapsulate the logic of the management of interaction?
- Which sort of general conceptual framework could generally deal with all interaction-related issues?
- Is there any general-purpose middleware technology that could reify such a framework?

# Outline

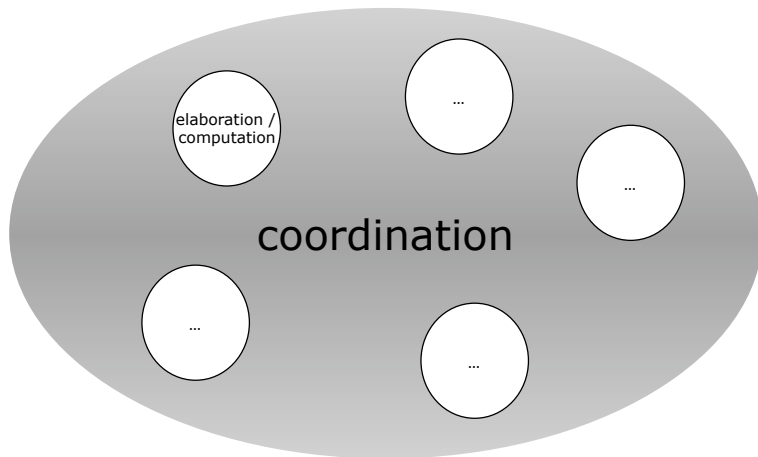
- 1 Premises
- 2 Interaction & Coordination**
- 3 Classes of Coordination Models
- 4 Tuple-based Coordination Models

# Harnessing the Complexity of Interaction

- **Coordination** is *managing interaction* [Wegner, 1997]
- **Coordination models** work by *constraining* the space of interaction [Wegner, 1996]
- So, in principle, *coordination abstractions* and *technologies* can help **harnessing** the intricacies of interaction in the engineering of complex software systems

# What is Coordination?

Coordination: ruling the space of interaction



# New Perspective on Computational Systems

## Programming languages

- Interaction as an *orthogonal dimension*
- Languages for interaction / coordination

## Software engineering

- Interaction as an *independent design dimension*
- Coordination patterns

## Artificial intelligence

- Interaction as a new *source for intelligence*
- Social intelligence

# Coordination as Constrained Interaction [Wegner, 1996]

Coordination models harness the complexity of interaction by disciplining the *space of admissible interactions* [Omicini, 1999]

- the **coordination primitives** provided by a coordination model constrain the possible agent *actions*
- the **coordination media** provided by a coordination model constrain the possible *targets* of agent actions

## Coordination models as *constrainers* of interaction

A coordination *model* constrains its coordinables (agents) to interact via a limited yet (possibly) expressive set of *primitives* acting on a class of pre-defined *media* of coordination, whose computational behaviour defines the *laws* of coordination

# Coordination in Distributed Programming I

## Coordination model as a *glue*

*A coordination model is the glue that binds separate activities into an ensemble [Gelernter and Carriero, 1992]*

## Coordination model as an *agent interaction framework*

*A coordination model provides a framework in which the interaction of active and independent entities called agents can be expressed [Ciancarini, 1996]*

# Coordination in Distributed Programming II

## Issues for a coordination model [Ciancarini, 1996]

A coordination model should cover the issues of

- *creation* and *destruction* of agents,
- *communication* among agents, and
- spatial *distribution* of agents, as well as
- *synchronization* and *distribution* of their *actions* over time

# Coordination: A Meta-model I

## A constructive approach [Ciancarini, 1996]

The components of a coordinated system are

**Coordination entities** — the entities whose mutual interaction is ruled by the model, also called the *coordinables*, or *agents*

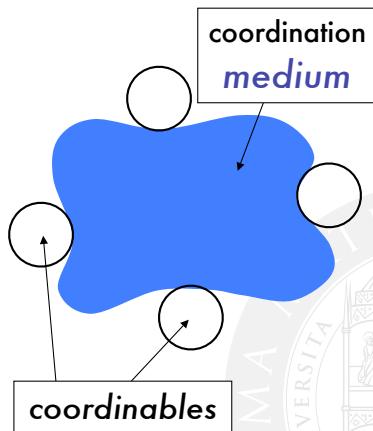
**Coordination media** — the abstractions enabling and ruling interaction among coordinables

**Coordination laws** — the rules governing the observable behaviour of coordinables, the behaviour coordination media, as well as their interactions of any sort

# Coordination: A Meta-model II

## The *medium of coordination*

- “fills” the interaction space
- enables / promotes / governs the admissible / desirable / required interactions among the interacting entities
- according to some *coordination laws*
  - enacted by the behaviour of the medium
  - defining the semantics of coordination



# Agents / Coordinables

## Original definition [Ciancarini, 1996]

*These are the entity types that are coordinated. These could be Unix-like processes, threads, concurrent objects and the like, and even users.*

**examples** Processes, threads, objects, human users, agents, ...

**focus** Observable behaviour of the coordinables

**question** Are we anyhow concerned here with the internal machinery / functioning of the coordinable, in principle?

# Coordination Media

## Original definition [Ciancarini, 1996]

*These are the media making communication among the agents possible. Moreover, a coordination medium can serve to aggregate agents that should be manipulated as a whole. Examples are classic media such as semaphores, monitors, or channels, or more complex media such as tuple spaces, blackboards, pipelines, and the like.*

**examples** Semaphors, monitors, channels, tuple spaces, blackboards, pipes, ...

**focus** The core around which the components of the system are organised

**question** Which are the possible computational models for coordination media?

# Coordination Laws I

## Original definition [Ciancarini, 1996]

*A coordination model should dictate a number of laws to describe how agents coordinate themselves through the given coordination media and using a number of coordination primitives. Examples are laws that enact either synchronous or asynchronous behaviors or exploit explicit or implicit naming schemes for coordination entities.*

# Coordination Laws II

- Coordination laws rule the observable behaviour of coordination media and coordinables, as well as their interaction
    - a notion of (*admissible interaction*) *event* is required to define coordination laws
  - The interaction events are (also) expressed in terms of
    - the *communication language*, as the syntax used to express and exchange data structures
- examples* tuples, XML elements, FOL terms, (Java) objects, ...
- the *coordination language*, as the set of the admissible interaction primitives, along with their semantics
- examples* in/out/rd (Linda), send/receive (channels), push/pull (pipes), ...

# Coordination as a General System Issue I

## Coordination is not just an issue of computational systems

[Malone and Crowston, 1994]

*Research in this area uses and extends ideas about coordination from disciplines such as computer science, organization theory, operations research, economics, linguistics, and psychology*

## A general purpose definition

[Malone and Crowston, 1994]

*coordination can be seen as the process of managing dependencies among activities*

# Coordination as a General System Issue II

- More generally, activities can be extended to include everything that generate *events* in a system, what *makes things happen*
- In systems of any sorts, *complexity* comes from dependencies
- When distributed systems are modelled as MAS (multi-agent systems), dependencies are the basic entity of the meta-model that brings about coordination

# Coordination as a Multi-Agent System Issue I

## Agent-based engineering for complex software systems

- *Encapsulation of control* makes agents overcome the abstraction gap of OOP [Odell, 2002]
- *Agent* abstractions inherently deal with *distribution* [Jennings, 2000]
- Agent *autonomy* — along with the associated agent features such as sociality, pro-activeness, situatedness, and the like – make agent-oriented software engineering (AOSE) a viable approach for engineering complex software systems [Jennings, 2001]
- Extra agent features such as *mobility* and *intelligence* make AOSE the most effective approach available for the engineering of intelligent and pervasive distributed systems [Zambonelli and Omicini, 2004]

# Coordination as a Multi-Agent System Issue II

## A MAS meta-model [Mariani and Omicini, 2014b]

- **Activities** are the *goal-directed/oriented* proceedings resulting into *actions* of any sort, which “make things happen” in a MAS
    - through actions, activities in a MAS are *social* [Castelfranchi, 1998] and *situated* [Suchman, 1987]
    - in MAS, activities are modelled through the *agent* abstraction
  - **Environment change** represents the (possibly unpredictable) variations in MAS environment
    - environment is usually modelled through the *resource* abstraction, as a non-goal-driven entity producing events and/or reactively waiting for requests to perform its function
  - Since activities *depend* on other activities (*social dependencies*), as well as on environment change (*situated dependencies*)
- **Dependencies** motivate and cause *interaction*, both social and situated, based on the sort of dependency taking place

# An Event-based View of MAS as Interacting Systems

MAS can be seen as event-based systems, where

- agents encapsulate *internal events*
- environment models *external events* through dedicated abstraction
- ! social abstractions of some sort take care of dealing with *interaction*
  - agent-agent interaction
  - agent-environment interaction
  - environment-environment interaction
  - and, any other more complex pattern of interaction

taking care of their mutual *dependencies*, by *coordinating* the many resulting flows of events [Malone and Crowston, 1994]

# Coordination in MAS

Given that

- artefacts are the *reactive* abstractions in MAS [Omicini et al., 2008]
- *coordination artefacts* are the social abstractions in charge of managing dependencies [Malone and Crowston, 1994]

the social abstractions in charge of coordinating multiple event flow according to their mutual dependencies in MAS are the *coordination artefacts* [Omicini et al., 2004]

## Coordination models in MAS

The role of coordination models in MAS [Ciancarini et al., 2000] is to provide event-driven *coordination media* as the *coordination artefacts* governing event coordination in MAS

# Benefits of Coordination in MAS as Event-Based Systems

- Since all events are *uniformly* represented through the same general event model – the same language for notifications –, *coordination artefacts* can be used to deal with both *social* and *situated* dependencies, governing agent-agent, agent-environment, environment-environment interaction through the same set of coordination abstractions, languages, and mechanisms [Mariani and Omicini, 2014a]
- Coordination artefacts provide a specific, *event-based computational model* for dealing with event observation, manipulation, and coordination
- Coordination artefacts *encapsulate the logic for the coordination* of multiple related flows of events, thus counterfeiting inversion of control

# Outline

- 1 Premises
- 2 Interaction & Coordination
- 3 Classes of Coordination Models**
- 4 Tuple-based Coordination Models



# Two Classes for Coordination Models

## Control-oriented vs. Data-oriented Models

- Control-driven vs. Data-driven Models  
[Papadopoulos and Arbab, 1998]

**Control-oriented** Focus on the *acts* of communication

**Data-oriented** Focus on the *information* exchanged during communication

- Several surveys, no time enough here
- Are these really *classes*?
  - actually, better to take this as a criterion to observe coordination models, rather than to separate them

# Control-oriented Models I

## Processes as black boxes

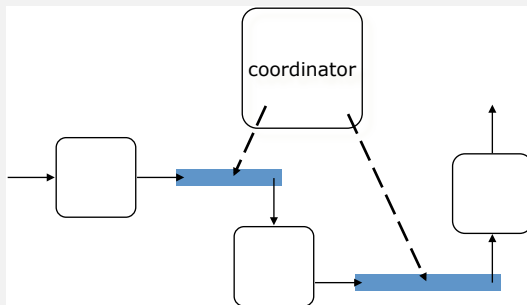
- I/O ports
- events & signals on state

## Coordinators...

- ...create coordinated processes as well as communication channels
- ...determine and change the topology of communication
- Hierarchies of coordinables / coordinators are possible

# Control-oriented Models II

## Coordinators as meta-level communication components



# Control-oriented Models III

## General features

- High flexibility, high control
- Separation between communication / coordination and computation / elaboration
- Examples
  - RAPIDE [Luckham et al., 1995]
  - Manifold [Arbab et al., 1993]
  - ConCoord [Holzbacher, 1996]
  - Reo [Arbab, 2004, Dastani et al., 2005]

# A Classical Example: Manifold [Arbab et al., 1993]

## Main features

- coordinators
- control-driven evolution
  - events without parameters
- stateful communication
- coordination via topology
- fine-grained coordination
- typical example: sort-merge

# Control-oriented Models: Impact on Design

## Which abstractions?

- Producer-consumer pattern
- Point-to-point communication
- Coordinator
- Coordination as configuration of topology

## Which systems?

- Fine-grained granularity
- Fine-tuned control
- Good for small-scale, closed systems

# An Evolutionary Pattern?

## Paradigms of sequential programming

- Imperative programming with “goto”
- Structured programming (procedure-oriented)
- Object-oriented programming (data-oriented)

## Paradigms of coordination programming

- Message-passing coordination
- Control-oriented coordination
- Data-oriented coordination

# Data-oriented Models I

## Communication channel

- Shared memory abstraction
- Stateful channel

## Processes

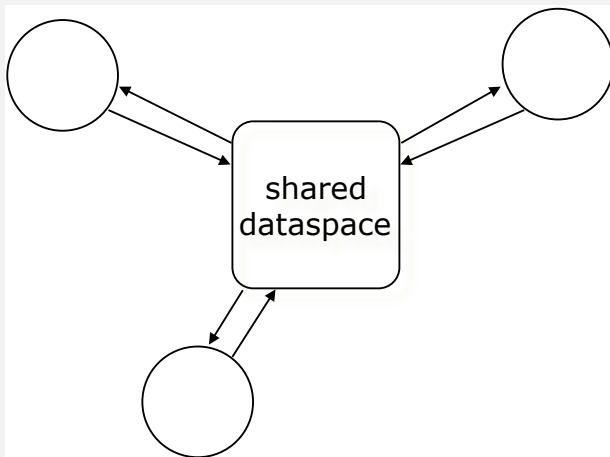
- Emitting / receiving data / information

## Coordination

- Access / change / synchronise on shared data

# Data-oriented Models II

## Shared dataspace: constraint on communication



# Data-oriented Models

## General features

- Expressive communication abstraction
- information-based design
- Possible spatio-temporal uncoupling
- No control means no flexibility??
- Examples
  - Gamma / Chemical coordination
  - Linda & friends / tuple-based coordination

# Outline

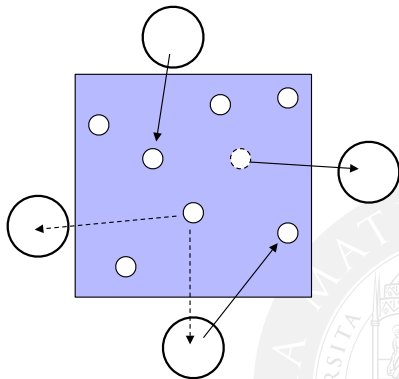
- 1 Premises
- 2 Interaction & Coordination
- 3 Classes of Coordination Models
- 4 Tuple-based Coordination Models**



# The Tuple-space Meta-model

## The basics

- *Coordinables* synchronise, cooperate, compete
  - based on *tuples*
  - available in the *tuple space*
  - by *associatively* accessing, consuming and producing tuples



# Tuple-based / Space-based Coordination Systems

## Adopting the constructive coordination meta-model [Ciancarini, 1996]

coordination media tuple spaces

- as multiset / bag of data objects / structures called *tuples*

communication language tuples

- as ordered collections of (possibly heterogeneous) information items

coordination language tuple space primitives

- as a set of operations to put, browse and retrieve tuples to/from the space

# Linda: The Communication Language [Gelernter, 1985]

## Communication Language

**tuples** ordered collections of possibly heterogeneous information chunks

- examples: `p(1)`, `printer('HP',dpi(300))`, `[0,0.5]`, `matrix(m0,3,3,0.5)`, `tree_node(node00,value(13),left(_),right(node01))`, ...

**templates / anti-tuples** specifications of set / classes of tuples

- examples: `p(X)`, `[?int,?int]`, `tree_node(N)`, ...

**tuple matching mechanism** the mechanism that matches tuples and templates

- examples: pattern matching, unification, ...

# Linda: The Coordination Language [Gelernter, 1985] I

## out(T)

- out(T) puts tuple T in to the tuple space

**success semantics** out(T) always succeeds—at the model level

**generative communication** tuple out(T) is permanently stored in the tuple space—until it is explicitly removed

**examples** out(p(1)), out(0,0.5), out(course('Antonio Natali', 'Poetry', hours(150))) ...

# Linda: The Coordination Language [Gelernter, 1985] II

## in(TT)

- `in(TT)` retrieves a tuple matching template TT from the tuple space

**destructive reading** the tuple retrieved is removed from the tuple centre

**non-determinism** if more than one tuple matches the template, one is chosen non-deterministically

**suspensive semantics** if no matching tuples are found in the tuple space, operation execution is suspended, and woken when a matching tuple is finally found

**examples** `in(p(X))`, `in(0,0.5)`, `in(course('Antonio Natali',Title,hours(X)) ...`

# Linda: The Coordination Language [Gelernter, 1985] III

## rd(TT)

- rd(TT) retrieves a tuple matching template TT from to the tuple space

**non-destructive reading** the tuple retrieved is left untouched in the tuple centre

**non-determinism** if more than one tuple matches the template, one is chosen non-deterministically

**suspensive semantics** if no matching tuples are found in the tuple space, operation execution is suspended, and awakened when a matching tuple is finally found

**examples** rd(p(X)), rd(0,0.5), rd(course('Alessandro Ricci', 'Operating Systems', hours(X)) ...

# Linda Extensions: Predicative Primitives

## `inp(TT)`, `rdp(TT)`

- both `inp(TT)` and `rdp(TT)` retrieve tuple `T` matching template `TT` from the tuple space  
 $= \text{in}(\text{TT}), \text{rd}(\text{TT})$  (non-)destructive reading, non-determinism, and syntax structure is maintained  
 $\neq \text{in}(\text{TT}), \text{rd}(\text{TT})$  suspensive semantics is lost: this *predicative* versions primitives just fail when no tuple matching `TT` is found in the tuple space  
`success / failure` predicative primitives introduce *success / failure semantics*: when a matching tuple is found, it is returned with a success result; when it is not, a failure is reported

# Linda Extensions: Bulk Primitives I

## `in_all(TT)`, `rd_all(TT)`

- All Linda primitives deal with one tuple at a time
  - some coordination problems require more than one tuple to be handled by a single primitive
- `rd_all(TT)`, `in_all(TT)` get all tuples in the tuple space matching with `TT`, and returns them all
  - no suspensive semantics: if no matching tuple is found, an empty collection is returned
  - no success / failure semantics: a collection of tuple is always successfully returned—possibly, an empty one
  - in case of logic-based primitives / tuples, the form of the primitive are `rd_all(TT,LT)`, `in_all(TT,LT)` (or equivalent), where the (possibly empty) list of tuples unifying with `TT` is unified with `LT`
  - (non-)destructive reading: `in_all(TT)` consumes all matching tuples in the tuple space; `rd_all(TT)` leaves the tuple space untouched

# Linda Extensions: Bulk Primitives II

- Many other bulk primitives have been proposed and implemented to address particular classes of problems [Rowstron, 1996]
- Adding a new set of primitives for any new coordination problem does not exactly fit open scenarios

# Linda Extensions: Multiple Tuple Spaces

`ts ? out(T)`

- Linda tuple space might be a bottleneck for coordination
- Many extensions have focussed on making a multiplicity of tuple spaces available to processes [Gelernter, 1989]
  - each of them encapsulating a portion of the coordination load
  - either hosted by a single machine, or distributed across the network
- Syntax required, and dependent on particular models and implementations
  - a space for tuple space names, possibly including network location
  - operators to associate Linda operators to tuple spaces
- For instance, `ts @ node ? out(p)` may denote the invocation of operation `out(p)` over tuple space `ts` on node `node`

# Main Features of Tuple-based Coordination

## Main features of the Linda model

**tuples** A tuple is an ordered collection of knowledge chunks, possibly heterogeneous in sort

**generative communication** until explicitly withdrawn, the tuples generated by coordinables have an independent existence in the tuple space; a tuple is equally accessible to all the coordinables, but is bound to none

**associative access** tuples in the tuple space are accessed through their content & structure, rather than by name, address, or location

**suspensive semantics** operations may be suspended based on unavailability of matching tuples, and be woken up when such tuples become available

# Features of Linda: Tuples

- A tuple is an ordered collection of knowledge chunks, possibly heterogeneous in sort
  - a record-like structure
  - with no need of field names
  - easy aggregation of knowledge
  - raw semantic interpretation: a tuple contains all information concerning an given item
- Tuple structure based on
  - arity
  - type
  - position
  - information content
- Anti-tuples / Tuple templates
  - to describe / define sets of tuples
- Matching mechanism
  - to define belongingness to a set

# Features of Linda: Generative Communication

## Communication orthogonality

- Both senders and the receivers can interact even without having prior knowledge about each others
- space uncoupling** no need to coexist in space for two processes to interact
- time uncoupling** no need for simultaneity for two processes to interact
- name uncoupling** no need for names for processes to interact

# Features of Linda: Associative Access

## Content-based coordination

- Synchronisation based on tuple content & structure
  - absence / presence of tuples with some content / structure determines the overall behaviour of the coordinables, and of the coordinated system in the overall
  - based on tuple templates & matching mechanism
- *Information-driven coordination*
  - patterns of coordination based on data / information availability
  - based on tuple templates & matching mechanism
- *Reification*
  - making events become tuples
  - grouping classes of events with tuple syntax, and accessing them via tuple templates

# Features of Linda: Suspensive Semantics I

## Blocking primitives

- `in` & `rd` primitives in Linda have a suspensive semantics
  - the coordination medium makes the primitives waiting in case a matching tuple is not found, and wakes it up when such a tuple is found
  - the coordinable invoking the suspensive primitive is expected to wait for its successful completion
- Twofold wait
  - in the **coordination medium** the operation is first (possibly) suspended, then (possibly) served: coordination based on absence / presence of tuples belonging to a given set
  - in the **coordination entity** the invocation may cause a wait-state in the invoker: hypothesis on the internal behaviour of the coordinable

# Features of Linda: Suspensive Semantics II

- Suspensive semantics is the basic mechanism for synchronisation in tuple-based models
- Synchronisation based on (un)availability of matching tuples / suitable information



# Data- vs. Control-driven Coordination

- What if we need to start an activity after, say, at least  $N$  processes have asked for a resource?
  - More generally, what if we need, in general, to coordinate based on the coordinable actions, rather than on the information available / exchanged?
- Classical distinction in the coordination community
  - data-driven coordination vs. control-driven coordination
- In more advanced scenario, these names do not fit
  - *information-driven* coordination vs. *action-driven* coordination fits better
  - but we might as well use the old terms, while we understand their limitations

# Hybrid Coordination Models

- Generally speaking, control-driven coordination does not fit so well information-driven contexts, like Web-based ones, for instance
  - control-driven models like Reo [Arbab, 2004] need to be adapted to contexts like agent-based ones, mainly to deal with the issue of autonomy in distributed systems [Dastani et al., 2005]
  - control should not pass through the component boundaries in order to avoid coupling in distributed systems
- We need features of both approaches to coordination
  - *hybrid* coordination models [Omicini, 2000]
  - adding for instance a control-driven layer to a Linda-based one
- Examples
  - LGI/LGL [Minsky et al., 2001]
  - TOTA [Mamei and Zambonelli, 2004]
  - TuCSon [Omicini and Zambonelli, 1999]

- 1 Premises
- 2 Interaction & Coordination
- 3 Classes of Coordination Models
- 4 Tuple-based Coordination Models

# Bibliography I



Arbab, F. (2004).

Reo: A channel-based coordination model for component composition.  
*Mathematical Structures in Computer Science*, 14:329–366.



Arbab, F., Herman, I., and Spilling, P. (1993).

An overview of MANIFOLD and its implementation.  
*Concurrency: Practice and Experience*, 5(1):23–70.



Brooks, R. A. (1991).

Intelligence without reason.

In Mylopoulos, J. and Reiter, R., editors, *12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, volume 1, pages 569–595, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.



Castelfranchi, C. (1998).

Modelling social action for AI agents.

*Artificial Intelligence*, 103(1-2):157–182.



Ciancarini, P. (1996).

Coordination models and languages as software integrators.

*ACM Computing Surveys*, 28(2):300–302.

# Bibliography II



Ciancarini, P., Omicini, A., and Zambonelli, F. (2000).

Multiagent system engineering: The coordination viewpoint.

In Jennings, N. R. and Lespérance, Y., editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *LNAI*, pages 250–259. Springer.

6th International Workshop (ATAL'99), Orlando, FL, USA, 15–17 July 1999. Proceedings.



Dastani, M., Arbab, F., and de Boer, F. S. (2005).

Coordination and composition in multi-agent systems.

In Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M. P., and Wooldridge, M. J., editors, *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 439–446, Utrecht, The Netherlands. ACM.



Fredriksson, M. and Gustavsson, R. (2004).

Online engineering and open computational systems.

In Bergenti, F., Gleizes, M.-P., and Zambonelli, F., editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organization*, pages 377–388. Kluwer Academic Publishers.



Gelernter, D. (1985).

Generative communication in Linda.

*ACM Transactions on Programming Languages and Systems*, 7(1):80–112.

# Bibliography III



Gelernter, D. (1989).

Multiple Tuple Spaces in Linda.

In Odijk, E., Rem, M., and Syre, J.-C., editors, *PARLE '89 Parallel Architectures and Languages Europe*, volume II: Parall of *Springer Berlin Heidelberg*, pages 20–27, Eindhoven, The Netherlands. Springer.



Gelernter, D. and Carriero, N. (1992).

Coordination languages and their significance.

*Communications of the ACM*, 35(2):97–107.



Goldin, D. Q., Smolka, S. A., and Wegner, P., editors (2006).

*Interactive Computation: The New Paradigm*.

Springer.



Holzbacher, A.-A. (1996).

A software environment for concurrent coordinated programming.

In Ciancarini, P. and Hankin, C., editors, *Coordination Languages and Models*, volume 1061 of *LNCS*, pages 249–266. Springer-Verlag.

1st International Conference (COORDINATION '96) Cesena, Italy, April 15–17, 1996.

# Bibliography IV



Jennings, N. R. (2000).  
On agent-based software engineering.  
*Artificial Intelligence*, 117(2):277–296.



Jennings, N. R. (2001).  
An agent-based approach for building complex software systems.  
*Communications of the ACM*, 44(4):35–41.



Luckham, D. C., Kenney, J. J., Augustin, L. M., Vera, J., Bryan, D., and Mann, W. (1995).  
Specification and analysis of system architecture using Rapide.  
*IEEE Transactions on Software Engineering*, 21(4):336–354.



Malone, T. and Crowston, K. (1994).  
The interdisciplinary study of coordination.  
*ACM Computing Surveys*, 26(1):87–119.



Mamei, M. and Zambonelli, F. (2004).  
Programming pervasive and mobile computing applications with the TOTA middleware.  
In *Pervasive Computing and Communications*, pages 263–273.  
2nd IEEE Annual Conference (PerCom 2004), Orlando, FL, USA, 14–17 March 2004.  
Proceedings.

# Bibliography V



Mariani, S. and Omicini, A. (2014a).

Coordination in situated systems: Engineering mas environment in TuCSon.

In Fortino, G., Di Fatta, G., Li, W., Ochoa, S., Cuzzocrea, A., and Pathan, M., editors, *Internet and Distributed Computing Systems*, volume 8729 of *Lecture Notes in Computer Science*, pages 99–110. Springer International Publishing.

7th International Conference on Internet and Distributed Computing Systems (IDCS 2014), Calabria, Italy, 22-24 September 2014, Proceedings.



Mariani, S. and Omicini, A. (2014b).

TuCSon coordination for MAS situatedness: Towards a methodology.

In Santoro, C. and Bergenti, F., editors, *WOA 2014 – XV Workshop Nazionale “Dagli Oggetti agli Agenti”*, volume 1260 of *CEUR Workshop Proceedings*, pages 62–71. Sun SITE Central Europe, RWTH Aachen University.



Milner, R. (1993).

Elements of interaction: Turing award lecture.

*Communications of the ACM*, 36(1):78–89.

# Bibliography VI



Minsky, N. H., Ungureanu, V., and Minsky, Y. M. (2001).  
Safe tuplespace-based coordination in multiagent systems.  
*Applied Artificial Intelligence: An International Journal*, 15(1):11–33.  
Special Issue: Coordination Models and Languages in AI.



Odell, J. J. (2002).  
Objects and agents compared.  
*Journal of Object Technology*, 1(1):41–53.



Omicini, A. (1999).  
On the semantics of tuple-based coordination models.  
In *1999 ACM Symposium on Applied Computing (SAC'99)*, pages 175–182, New York, NY, USA. ACM.  
Special Track on Coordination Models, Languages and Applications.



Omicini, A. (2000).  
Hybrid coordination models for handling information exchange among Internet agents.  
In Bonarini, A., Colombetti, M., and Lanzi, P. L., editors, *Workshop "Agenti intelligenti e Internet: teorie, strumenti e applicazioni"*, pages 1–4, 7th AI\*IA Convention (AI\*IA 2000), Milano, Italy.

# Bibliography VII



Omicini, A. (2012).

Agents writing on walls: Cognitive stigmergy and beyond.

In Paglieri, F., Tummolini, L., Falcone, R., and Miceli, M., editors, *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, volume 20 of *Tributes*, chapter 29, pages 543–556. College Publications, London.



Omicini, A., Ricci, A., and Viroli, M. (2008).

Artifacts in the A&A meta-model for multi-agent systems.

*Autonomous Agents and Multi-Agent Systems*, 17(3):432–456.

Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.



Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004).

Coordination artifacts: Environment-based coordination for intelligent agents.

In Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 286–293, New York, USA. ACM.

# Bibliography VIII



Omicini, A. and Zambonelli, F. (1999).  
Coordination for Internet application development.  
*Autonomous Agents and Multi-Agent Systems*, 2(3):251–269.  
Special Issue: Coordination Mechanisms for Web Agents.



Papadopoulos, G. A. and Arbab, F. (1998).  
Coordination models and languages.  
In Zelkowitz, M. V., editor, *The Engineering of Large Systems*, volume 46 of *Advances in Computers*, pages 329–400. Academic Press.



Rowstron, A. I. T. (1996).  
*Bulk Primitives in Linda Run-Time Systems*.  
PhD thesis, The University of York.



Suchman, L. A. (1987).  
*Plans and Situated Actions: The Problem of Human-Machine Communication*.  
Cambridge University Press, New York, NYU, USA.

# Bibliography IX



Wegner, P. (1996).

Coordination as constrained interaction.

In Ciancarini, P. and Hankin, C., editors, *Coordination Languages and Models. First International Conference, COORDINATION '96 Cesena, Italy, April 15–17, 1996. Proceedings*, Lecture Notes in Computer Science, pages 28–33, Cesena, Italy. Springer Berlin Heidelberg.



Wegner, P. (1997).

Why interaction is more powerful than algorithms.

*Communications of the ACM*, 40(5):80–91.



Wegner, P. and Goldin, D. (1999).

Mathematical models of interactive computing.

Technical report, Brown University, Providence, RI, USA.



Wegner, P. and Goldin, D. (2003).

Computation beyond Turing machines.

*Communications of the ACM*, 46(4):100–102.

# Bibliography X



Zambonelli, F. and Omicini, A. (2004).

Challenges and research directions in agent-oriented software engineering.

*Autonomous Agents and Multi-Agent Systems*, 9(3):253–283.

Special Issue: Challenges for Agent-Based Computing.

# Interaction, Complexity, Coordination

## Nature-inspired Coordination Models for Complex Distributed Systems

Andrea Omicini  
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)  
ALMA MATER STUDIORUM – Università di Bologna

Fribourg, Switzerland  
20 November 2014

