Coordination Models & Languages

Multiagent Systems LS
Sistemi Multiagente LS

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2009/2010
Elements of Multi-agent Systems Engineering

Coordination: A Meta-model

Enabling vs. Governing Interaction

Classifying Coordination Models

Introduction to (Tuple-based) Coordination
  • Tuple-based Coordination & Linda
Scenarios for Multi-Agent Systems

Issues

- Concurrency / Parallelism
  - Agents are multiple independent activities / loci of control . . .
  - . . . active simultaneously

- Distribution
  - Activities running on different and heterogeneous execution contexts (machines, devices, . . .)

- “Social” Interaction
  - Dependencies among agent activities
  - Collective goals involving activities coordination / cooperation

- “Environmental” Interaction
  - Interaction with external resources
  - Interaction within the time-space fabric
Scenarios for Multi-Agent Systems

Issues

- **Concurrency / Parallelism**
  - Agents are multiple independent activities / loci of control . . .
  - . . . active simultaneously

- **Distribution**
  - Activities running on different and heterogeneous execution contexts (machines, devices, . . .)

- **“Social” Interaction**
  - Dependencies among agent activities
  - Collective goals involving activities coordination / cooperation

- **“Environmental” Interaction**
  - Interaction with external resources
  - Interaction within the time-space fabric
Issues

- **Concurrency / Parallelism**
  - Agents are multiple independent activities / loci of control . . .
  - . . . active simultaneously

- **Distribution**
  - Activities running on different and heterogeneous execution contexts (machines, devices, . . .)

- **“Social” Interaction**
  - Dependencies among agent activities
  - Collective goals involving activities coordination / cooperation

- **“Environmental” Interaction**
  - Interaction with external resources
  - Interaction within the time-space fabric
Scenarios for Multi-Agent Systems

Issues

- **Concurrency / Parallelism**
  - Agents are multiple independent activities / loci of control . . .
  - . . . active simultaneously

- **Distribution**
  - Activities running on different and heterogeneous execution contexts (machines, devices, . . .)

- **“Social” Interaction**
  - Dependencies among agent activities
  - Collective goals involving activities coordination / cooperation

- **“Environmental” Interaction**
  - Interaction with external resources
  - Interaction within the time-space fabric
# Issues

- **Concurrency / Parallelism**
  - Agents are multiple independent activities / loci of control . . .
  - . . . active simultaneously

- **Distribution**
  - Activities running on different and heterogeneous execution contexts (machines, devices, . . .)

- **“Social” Interaction**
  - Dependencies among agent activities
  - Collective goals involving activities coordination / cooperation

- **“Environmental” Interaction**
  - Interaction with external resources
  - Interaction within the time-space fabric
Basic Engineering Principles

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 / evolutions
  - On-line engineering
  - (Cognitive) Self-organising systems
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 / evolutions
  - On-line engineering
  - (Cognitive) Self-organising systems
Basic Engineering Principles

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 / evolutions
  - On-line engineering
  - (Cognitive) Self-organising systems
Basic Engineering Principles

**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 / evolutions
  - On-line engineering
  - (Cognitive) Self-organising systems
Which Components?

Open MAS
- No hypothesis on the agent life & behaviour

Distributed MAS
- No hypothesis on the agent location & motion

Heterogeneous MAS
- No hypothesis on the agent nature & structure
Which Components?

Open MAS
- No hypothesis on the agent life & behaviour

Distributed MAS
- No hypothesis on the agent location & motion

Heterogeneous MAS
- No hypothesis on the agent nature & structure
### Which Components?

<table>
<thead>
<tr>
<th>Open MAS</th>
<th>Distributed MAS</th>
<th>Heterogeneous MAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No hypothesis on the agent life &amp; behaviour</td>
<td>No hypothesis on the agent location &amp; motion</td>
<td>No hypothesis on the agent nature &amp; structure</td>
</tr>
</tbody>
</table>
The Space of Interaction

interaction space

software component

...
Algorithmic Computation

Elaboration / Computation

- Turing Machine
- Black box algorithms
- Church and computable functions

Beyond Turing Machines

- Wegner’s Interaction Machines
- Examples: AGV, Chess oracle
Algorithmic Computation

Elaboration / Computation
- Turing Machine
- Black box algorithms
- Church and computable functions

Beyond Turing Machines
- Wegner’s Interaction Machines
- Examples: AGV, Chess oracle
Basics of Interaction

A simple sequential machine
- Output: shows part of its state outside
- Input: bounds a portion of its own state to the outside

Coupling across component’s boundaries
- Information
- Time – internal / sequential vs. external / entropic
Basics of Interaction

A simple sequential machine
- Output: shows part of its state outside
- Input: bounds a portion of its own state to the outside

Coupling across component’s boundaries
- Information
- Time – internal / sequential vs. external / entropic
Compositionality vs. Non-compositionality

Compositionality

- Sequential composition $P_1; P_2$
- $\text{behaviour}(P_1; P_2) = \text{behaviour}(P_1) + \text{behaviour}(P_2)$

Non-compositionality

- Interactive composition $P_1|P_2$
- $\text{behaviour}(P_1|P_2) = \text{behaviour}(P_1) + \text{behaviour}(P_2) + \text{interaction}(P_1, P_2)$
- Interactive composition is more than the sum of its parts
Compositionality vs. Non-compositionality

**Compositionality**

- Sequential composition \( P_1; P_2 \)
  \[
  \text{behaviour}(P_1; P_2) = \text{behaviour}(P_1) + \text{behaviour}(P_2)
  \]

**Non-compositionality**

- Interactive composition \( P_1|P_2 \)
  \[
  \text{behaviour}(P_1|P_2) = \text{behaviour}(P_1) + \text{behaviour}(P_2) + \text{interaction}(P_1, P_2)
  \]
  Interactive composition is more than the sum of its parts
Non-compositionality

Issues

- Compositionality vs. formalisability
- Emergent behaviours
- Formalisability vs. expressiveness
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]

Issues for a coordination model

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 [Ciancarini, 1996]
Coordination in Distributed Programming

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]

Issues for a coordination model

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 [Ciancarini, 1996]
### 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]

### Issues for a coordination model

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 [Ciancarini, 1996]
Coordination in Distributed Programming

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]

Issues for a coordination model

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 [Ciancarini, 1996]
What is Coordination?

Ruling the space of interaction

coordination

elaboration / computation

...
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
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
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
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
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
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
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
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
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
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
A constructive approach

Which are the components of a coordination system?

Coordination entities: Entities whose mutual interaction is ruled by the model, also called the *coordinables*.

Coordination media: Abstractions enabling and ruling agent interactions.

Coordination laws: Rules defining the behaviour of the coordination media in response to interaction.
A constructive approach

Which are the components of a coordination system?

Coordination entities  Entities whose mutual interaction is ruled by the model, also called the coordinables

Coordination media  Abstractions enabling and ruling agent interactions

Coordination laws  Rules defining the behaviour of the coordination media in response to interaction
A constructive approach

Which are the components of a coordination system?

**Coordination entities** Entities whose mutual interaction is ruled by the model, also called the *coordinables*

Coordination media Abstractions enabling and ruling agent interactions

Coordination laws Rules defining the behaviour of the coordination media in response to interaction
A constructive approach

Which are the components of a coordination system?

**Coordination entities** Entities whose mutual interaction is ruled by the model, also called the *coordinables*

**Coordination media** Abstractions enabling and ruling agent interactions

**Coordination laws** Rules defining the behaviour of the coordination media in response to interaction
A constructive approach

Which are the components of a coordination system?

**Coordination entities**  Entities whose mutual interaction is ruled by the model, also called the *coordinables*

**Coordination media**  Abstractions enabling and ruling agent interactions

**Coordination laws**  Rules defining the behaviour of the coordination media in response to interaction
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? 
→ This issue will be clear when comparing Linda & TuCSoN 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?

→ This issue will be clear when comparing Linda & TuCSoN 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?

→ This issue will be clear when comparing Linda & TuCSoN agents
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?

→ This issue will be clear when comparing Linda & TuCSoN 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?

→ This issue will be clear when comparing Linda & TuCSoN agents
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?

→ This issue will be clear when comparing Linda tuple spaces & ReSpecT tuple centres
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?

→ This issue will be clear when comparing Linda tuple spaces & ReSpecT tuple centres
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?
  → This issue will be clear when comparing Linda tuple spaces & ReSpecT tuple centres
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?
          → This issue will be clear when comparing Linda tuple spaces & ReSpecT tuple centres
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**: Semaphores, 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?
  - This issue will be clear when comparing Linda tuple spaces & ReSpecT tuple centres
Coordination Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 (Linda), send/receive (channels), push/pull (pipes)
Coordination Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 asmissible interaction primitives, along with their semantics
    examples: in/out, read/write (channels), push/pull (pipes)
Coordination Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 Laws

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 define the behaviour of the coordination media in response to interaction
  - a notion of (admissible interaction) event is required to define a model
- Coordination laws are 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 asmissible interaction primitives, along with their semantics
    examples: in/out/rd (Linda), send/receive (channels), push/pull (pipes), ...
What Do We Ask to a Coordination Model?

- to provide high-level abstractions and powerful mechanisms for distributed system engineering
- to enable and promote the construction of open, distributed, heterogeneous systems
- to intrinsically add properties to systems independently of components
  - e.g. flexibility, control, intelligence, ...
Examples of Coordination Mechanisms I

Message passing

- communication among peers
- no abstractions apart from message
- no limitations
  - the notion of protocol could be added as a coordination abstraction
- no intrinsic model of coordination
- any pattern of coordination can be superimposed – again, protocols
Agent Communication Languages

- Goal: promote information exchange
- Examples: Arcol, KQML
- Standard: FIPA ACL
- Semantics: ontologies
- **Enabling communication**
  - ACLs *create* the space of inter-agent communication
  - they do not allow to *constrain* it
- No coordination, again, if not with protocols
Examples of Coordination Mechanisms III

Service-Oriented Architectures

- Basic abstraction: service
- Basic pattern: Service request / response
- Several standards
- Very simple pattern of coordination
### Web Server

- Basic abstraction: resource (REST/ROA)
- Basic pattern: Resource request / representation / response
- Several standards
- Again, a very simple pattern of coordination
- Generally speaking, objects, HTTP, applets, JavaScript with AJAX, user interface
  - a multi-coordinated systems
  - “spaghetti-coordination”, no value added from composition
- **How can we “fill” the space of interaction to add value to systems?**
  - so, how do we get value from coordination?
Middleware

- Goal: to provide global properties across distributed systems
- Idea: fill the space of interaction with abstractions and shared features
  - interoperability, security, transactionality, . . .
- Middleware can contain coordination abstractions
  - but, it can contain anything, so we need to look at specific middleware
CORBA

- Goal: managing object interaction across a distributed systems in a transparent way
- Key features: ORB, IDL, CORBAServices…
- However, no model for coordination
  - just the client-servant pattern
- However, it can provide a shared support for any coordination abstraction or pattern
Enabling interaction

- ACL, middleware, mediators...
- enabling communication
- enabling components interoperation
- no models for coordination of components
  - no rules on what components should (not) say and do at any given moment, depending on what other components say and do, and on what happens inside and outside the system
Governing interaction

- ruling communication
- providing concepts, abstractions, models, mechanisms for meaningful component integration
- governing mutual component interaction, and environment-component interaction
- in general, a model that does
  - rule what components should (not) say and do at any given moment
  - depending on what other components say and do, and on what happens inside and outside the system
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
A Classical Example: Manifold

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
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
- “Procedure-call” coordination
- Control-oriented coordination
- Data-oriented coordination
An Evolutionary Pattern?

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

Paradigms of coordination programming
- “Procedure-call” 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. Elements of Multi-agent Systems Engineering
2. Coordination: A Meta-model
3. Enabling vs. Governing Interaction
4. Classifying Coordination Models
5. Introduction to (Tuple-based) Coordination
   - Tuple-based Coordination & Linda
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
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
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
The Tuple-space Meta-model

The basics

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

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

- **tuples** ordered collections of possibly heterogeneous information chunks
  - examples: \( p(1), \text{printer('HP',dpi(300))), [0,0.5], \)
  \( \text{matrix(m0,3,3,0.5)}, \)
  \( \text{tree\_node(node00,value(13),left(_),right(node01))}, \ldots \)

- **templates / anti-tuples** specifications of set / classes of tuples
  - examples: \( p(X), [?\text{int},?\text{int}], \text{tree\_node(N)}, \ldots \)

- **tuple matching mechanism** the mechanism by which tuples are said to “match” templates
  - examples: pattern matching, unification, \ldots \)
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 by which tuples are said to “match” templates
- examples: pattern matching, unification, ...
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 by which tuples are said to “match” templates

- examples: pattern matching, unification, ...
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 by which tuples are said to “match” templates
- examples: pattern matching, unification, ...
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: \( p(X), \) \([?\text{int},?\text{int}]\), tree_node(\text{N}), . . .

**tuple matching mechanism** the mechanism by which tuples are said to “match” templates

- examples: pattern matching, unification, . . .
Communication Language

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

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

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

- examples: \( p(X) \), \([?\text{int},?\text{int}]\), \( \text{tree_node(N)} \), ... 

**tuple matching mechanism** the mechanism by which tuples are said to “match” templates

- examples: pattern matching, unification, ...
Linda: The Communication Language [Gelernter, 1985]

Communication Language

**tuples** ordered collections of possibly heterogeneous information chunks
- examples: \( p(1), \text{printer('HP',dpi(300))}, [0,0.5], \text{matrix(m0,3,3,0.5)} \),
  \text{tree
  _node(node00,value(13),left(_),right(node01))}, \ldots

**templates / anti-tuples** specifications of set / classes of tuples
- examples: \( p(X), [\text{?int,?int}], \text{tree
  _node(N)} \), \ldots

**tuple matching mechanism** the mechanism by which tuples are said to “match” templates
- examples: pattern matching, unification, \ldots

\textbf{out(T)}

- \texttt{out(T)} puts tuple T in to the tuple space

\textbf{examples} \hspace{1em} \texttt{out(p(1))}, \texttt{out(0,0.5)}, \texttt{out(course(’Denti Enrico’, ’Poetry’, hours(150)))}...
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(’Denti Enrico’,Title,hours(X))) ...
**rd(TT)**

- **rd(TT)** retrieves a tuple matching template TT from 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(’Ricci Alessandro’, ’Operating Systems’, hours(X)))...
A First Example: Sharing a Pool of Printers

The model
- Each printer in the pool is represented by a number PrinterNo
- An available printer is represented by a tuple availablePrinter(PrinterNo)

The protocol
- Each agent willing to print asks for a tuple availablePrinter(N)
- When an available printer is assigned to the agent, the corresponding tuple is removed
- When the agent has done with printing, it puts the tuple back in the tuple space
A First Example: Sharing a Pool of Printers

The model

- Each printer in the pool is represented by a number PrinterNo
- An available printer is represented by a tuple availablePrinter(PrinterNo)

The protocol

- Each agent willing to print asks for a tuple availablePrinter(N)
- When an available printer is assigned to the agent, the corresponding tuple is removed
- When the agent has done with printing, it puts the tuple back in the tuple space
A First Example: Sharing a Pool of Printers

The model
- Each printer in the pool is represented by a number PrinterNo
- An available printer is represented by a tuple availablePrinter(PrinterNo)

The protocol
- Each agent willing to print asks for a tuple availablePrinter(N)
- When an available printer is assigned to the agent, the corresponding tuple is removed
- When the agent has done with printing, it puts the tuple back in the tuple space
A First Example: Sharing a Pool of Printers

The model

- Each printer in the pool is represented by a number PrinterNo
- An available printer is represented by a tuple availablePrinter(PrinterNo)

The protocol

- Each agent willing to print asks for a tuple availablePrinter(N)
- When an available printer is assigned to the agent, the corresponding tuple is removed
- When the agent has done with printing, it puts the tuple back in the tuple space
A First Example: Sharing a Pool of Printers

The model

- Each printer in the pool is represented by a number PrinterNo
- An available printer is represented by a tuple availablePrinter(PrinterNo)

The protocol

- Each agent willing to print asks for a tuple availablePrinter(N)
  - When an available printer is assigned to the agent, the corresponding tuple is removed
  - When the agent has done with printing, it puts the tuple back in the tuple space
A First Example: Sharing a Pool of Printers

The model

- Each printer in the pool is represented by a number `PrinterNo`.
- An available printer is represented by a tuple `availablePrinter(PrinterNo)`.

The protocol

- Each agent willing to print asks for a tuple `availablePrinter(N)`.
- When an available printer is assigned to the agent, the corresponding tuple is removed.
- When the agent has done with printing, it puts the tuple back in the tuple space.
A First Example: Sharing a Pool of Printers

The model
- Each printer in the pool is represented by a number PrinterNo
- An available printer is represented by a tuple availablePrinter(PrinterNo)

The protocol
- Each agent willing to print asks for a tuple availablePrinter(N)
- When an available printer is assigned to the agent, the corresponding tuple is removed
- When the agent has done with printing, it puts the tuple back in the tuple space
First Example: The Tuple Space

The initial state

- Each printer in the pool is represented by a number PrinterNo.
- All printers are initially available, so there are as many availablePrinter(PrinterNo) tuples as printers in the pool.

State

- At each instant in the working cycle, there are as many availablePrinter(PrinterNo) in the tuple space as there are available printers.
First Example: The Tuple Space

The initial state

- Each printer in the pool is represented by a number $\text{PrinterNo}$
- All printer are initially available, so there are as many $\text{availablePrinter(PrinterNo)}$ tuple as printers in the pool

State

- At each instant in the working cycle, there are as many $\text{availablePrinter(PrinterNo)}$ in the tuple space as there are available printers
The initial state

- Each printer in the pool is represented by a number `PrinterNo`
- All printers are initially available, so there are as many `availablePrinter(PrinterNo)` tuples as printers in the pool

State

- At each instant in the working cycle, there are as many `availablePrinter(PrinterNo)` in the tuple space as there are available printers
First Example: The Tuple Space

The initial state
- Each printer in the pool is represented by a number PrinterNo
- All printers are initially available, so there are as many availablePrinter(PrinterNo) tuple as printers in the pool

State
- At each instant in the working cycle, there are as many availablePrinter(PrinterNo) in the tuple space as there are available printers
First Example: The Tuple Space

The initial state

- Each printer in the pool is represented by a number PrinterNo
- All printer are initially available, so there are as many availablePrinter(PrinterNo) tuple as printers in the pool

State

- At each instant in the working cycle, there are as many availablePrinter(PrinterNo) in the tuple space as there are available printers
First Example: The Agent Protocol

Agents printing with ins and outs

printingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
!, printingAgent.

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```
printingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
!, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```prolog
printingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
!, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```prolog
printingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
    !, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```prolog
printingAgent :-
  getSomethingToPrint(Doc),
  in(availablePrinter(N)),
  print(document(Doc),printer(N)),
  out(availablePrinter(N)),
!, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```
paintingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc), printer(N)),
    out(availablePrinter(N)),
!, paintingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```plaintext
printingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
!, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```
printingAgent :-
    getsomethingtotoPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
!, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```prolog
printingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
!, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
# First Example: The Agent Protocol

<table>
<thead>
<tr>
<th>Agents printing with ins and outs</th>
</tr>
</thead>
<tbody>
<tr>
<td>printingAgent :-</td>
</tr>
<tr>
<td>getSomethingToPrint(Doc),</td>
</tr>
<tr>
<td>in(availablePrinter(N)),</td>
</tr>
<tr>
<td>print(document(Doc),printer(N)),</td>
</tr>
<tr>
<td>out(availablePrinter(N)),</td>
</tr>
<tr>
<td>!, printingAgent.</td>
</tr>
</tbody>
</table>

### Features

- **Very simple agent protocol** – agents concerned only with printing, not with choosing / sharing / competing
- **Clean world representation** – observing the tuple space is observing a portion of the actual system state
- **Coordination (such as synchronisation)** is mostly delegated to the coordination medium
First Example: The Agent Protocol

Agents printing with ins and outs

```prolog
printingAgent :-
    getSomethingToPrint(Doc),
    in(availablePrinter(N)),
    print(document(Doc),printer(N)),
    out(availablePrinter(N)),
    !, printingAgent.
```

Features

- Very simple agent protocol – agents concerned only with printing, not with choosing / sharing / competing
- Clean world representation – observing the tuple space is observing a portion of the actual system state
- Coordination (such as synchronisation) is mostly delegated to the coordination medium
Linda Extensions: Predicative Primitives

inp(TT), rdp(TT)

- both inp(TT) and rdp(TT) retrieve tuple T matching template TT from the tuple space

= in(TT), rd(TT) (non-)destructive reading, non-determinism, and syntax structure is maintained

\neq in(TT), rd(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: Predicative Primitives

inp(TT), rdp(TT)

- both inp(TT) and rdp(TT) retrieve tuple T matching template TT from the tuple space

= in(TT), rd(TT) (non-)destructive reading, non-determinism, and syntax structure is maintained

≠ in(TT), rd(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
**inp(TT), rdp(TT)**

- both `inp(TT)` and `rdp(TT)` retrieve tuple `T` matching template `TT` from the tuple space

\[ = \text{in}(TT), \text{rd}(TT) \]

(non-)destructive reading, non-determinism, and syntax structure is maintained

\[ \neq \text{in}(TT), \text{rd}(TT) \]

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: Predicative Primitives

\( \text{inp}(TT), \text{rdp}(TT) \)

- both \( \text{inp}(TT) \) and \( \text{rdp}(TT) \) retrieve tuple \( T \) matching template \( TT \) from the tuple space
  
  \( \equiv \text{in}(TT), \text{rd}(TT) \): (non-)destructive reading, non-determinism, and syntax structure is maintained

\( \not\equiv \text{in}(TT), \text{rd}(TT) \): 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
Tuple-based Coordination & Linda

Linda Extensions: Predicative Primitives

**inp(TT), rdp(TT)**

- both `inp(TT)` and `rdp(TT)` retrieve tuple `T` matching template `TT` from the tuple space
- \( = in(TT), \ rd(TT) \) (non-)destructive reading, non-determinism, and syntax structure is maintained
- \( \neq in(TT), \ rd(TT) \) 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

**in_all(TT), rd_all(TT)**

- Linda primitives (including predicative ones) deal with a 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.

- Many other bulk primitives have been proposed and implemented to address particular classes of problems.
Linda Extensions: Bulk Primitives

**in_all(TT), rd_all(TT)**

- Linda primitives (including predicative ones) deal with a 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
- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Bulk Primitives

**in**\_all(TT), **rd**\_all(TT)**

- Linda primitives (including predicative ones) deal with a 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 tuples 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
- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Bulk Primitives

**in_all(TT), rd_all(TT)**

- Linda primitives (including predicative ones) deal with a 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
- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Bulk Primitives

**in\_all(TT), rd\_all(TT)**

- Linda primitives (including predicative ones) deal with a 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
- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Bulk Primitives

**in_all(TT), rd_all(TT)**

- Linda primitives (including predicative ones) deal with a 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
- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Bulk Primitives

**in_all(TT), rd_all(TT)**
- Linda primitives (including predicative ones) deal with a 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
- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Bulk Primitives

**in\_all(TT), rd\_all(TT)**

- Linda primitives (including predicative ones) deal with a tuple at a time
  - some coordination problems require more than one tuple to be handled by a single primitive
- \( \text{rd\_all(TT), in\_all(TT)} \) 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 \( \text{rd\_all(TT,LT), in\_all(TT,LT)} \) (or equivalent), where the (possibly empty) list of tuples unifying with TT is unified with LT
  - (non-)destructive reading: \( \text{in\_all(TT)} \) consumes all matching tuples in the tuple space; \( \text{rd\_all(TT)} \) leaves the tuple space untouched

- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Bulk Primitives

\textbf{in\_all}(TT), \textbf{rd\_all}(TT)

- Linda primitives (including predicative ones) deal with a tuple at a time
  - some coordination problems require more than one tuple to be handled by a single primitive
- \textbf{rd\_all}(TT), \textbf{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 \textbf{rd\_all}(TT,LT), \textbf{in\_all}(TT,LT) (or equivalent), where the (possibly empty) list of tuples unifying with TT is unified with LT
  - (non-)destructive reading: \textbf{in\_all}(TT) consumes all matching tuples in the tuple space; \textbf{rd\_all}(TT) leaves the tuple space untouched
- Many other bulk primitives have been proposed and implemented to address particular classes of problems
Linda Extensions: Multiple Tuple Spaces

**ts ? out(T)**

- Linda tuple space might be a bottleneck for coordination
- Many extensions have focused on making a multiplicity of tuple spaces available to agents
  - 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*
Linda Extensions: Multiple Tuple Spaces

\textbf{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 agents
  - 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, \texttt{ts@node ? out(p)} may denote the invocation of operation \texttt{out(p)} over tuple space \texttt{ts} on node \texttt{node}
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 agents
  - 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**
Linda Extensions: Multiple Tuple Spaces

\( \text{ts} \ ? \ \text{out}(T) \)

- Linda tuple space might be a bottleneck for coordination
- Many extensions have focused on making a multiplicity of tuple spaces available to agents
  - 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, \( \text{ts@node} \ ? \ \text{out}(p) \) may denote the invocation of operation \( \text{out}(p) \) over tuple space \( \text{ts} \) on node \( \text{node} \).
Linda Extensions: Multiple Tuple Spaces

Linda tuple space might be a bottleneck for coordination

Many extensions have focussed on making a multiplicity of tuple spaces available to agents
  - 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
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 agents
  - 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*
Linda Extensions: Multiple Tuple Spaces

**ts ? out(T)**

- Linda tuple space might be a bottleneck for coordination
- Many extensions have focused on making a multiplicity of tuple spaces available to agents
  - 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`
Linda Extensions: Multiple Tuple Spaces

\texttt{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 agents
  - 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, \texttt{ts@node ? out(p)} may denote the invocation of operation \texttt{out(p)} over tuple space \texttt{ts} on node \texttt{node}
Linda Extensions: Multiple Tuple Spaces

\texttt{ts ? out(T)}

- Linda tuple space might be a bottleneck for coordination
- Many extensions have focused on making a multiplicity of tuple spaces available to agents
  - 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, \texttt{ts@node ? out(p)} may denote the invocation of operation \texttt{out(p)} over tuple space \texttt{ts} on node \texttt{node}
**Main Features of Tuple-based Coordination**

<table>
<thead>
<tr>
<th>Main features of the Linda model</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tuples</strong></td>
<td>A tuple is an ordered collection of knowledge chunks, possibly heterogeneous in sort</td>
</tr>
<tr>
<td><strong>generative communication</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>associative access</strong></td>
<td>tuples in the tuple space are accessed through their content &amp; structure, rather than by name, address, or location</td>
</tr>
<tr>
<td><strong>suspensive semantics</strong></td>
<td>operations may be suspended based on unavailability of matching tuples, and be woken up when such tuples become available</td>
</tr>
</tbody>
</table>
### Main Features of Tuple-based Coordination

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tuples</strong></td>
<td>A tuple is an ordered collection of knowledge chunks, possibly heterogeneous in sort</td>
</tr>
<tr>
<td>Generative Communication</td>
<td>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</td>
</tr>
<tr>
<td>Associative Access</td>
<td>Tuples in the tuple space are accessed through their content &amp; structure, rather than by name, address, or location</td>
</tr>
<tr>
<td>Suspensive Semantics</td>
<td>Operations may be suspended based on unavailability of matching tuples, and be woken up when such tuples become available</td>
</tr>
</tbody>
</table>
## Main Features of Tuple-based Coordination

**Main features of the Linda model**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tuples</strong></td>
<td>A tuple is an ordered collection of knowledge chunks, possibly heterogeneous in sort.</td>
</tr>
<tr>
<td><strong>generative communication</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>associative access</strong></td>
<td>Tuples in the tuple space are accessed through their content &amp; structure, rather than by name, address, or location.</td>
</tr>
<tr>
<td><strong>suspensive semantics</strong></td>
<td>Operations may be suspended based on unavailability of matching tuples, and be woken up when such tuples become available.</td>
</tr>
</tbody>
</table>
Main Features of Tuple-based Coordination

<table>
<thead>
<tr>
<th>Main features of the Linda model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tuples</strong></td>
</tr>
<tr>
<td><strong>generative communication</strong></td>
</tr>
<tr>
<td><strong>associative access</strong></td>
</tr>
<tr>
<td><strong>suspensive semantics</strong></td>
</tr>
</tbody>
</table>
**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
  - 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: 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
  - semantic interpretation: a tuple contains all information concerning a 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: 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
  - semantic interpretation: a tuple contains all information concerning a 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: 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
  - 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: 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
  - 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: 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
  - semantic interpretation: a tuple contains all information concerning a 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: 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
  - 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: 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
  - 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: 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
  - semantic interpretation: a tuple contains all information concerning a 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: 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
  - semantic interpretation: a tuple contains all information concerning a 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: 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
  - 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: 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
  - 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: 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
  - 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: 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
  - semantic interpretation: a tuple contains all information concerning a 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 (also called distributed naming): no need to coexist in space for two agents to interact
  - time uncoupling: no need for simultaneity for two agents to interact
  - name uncoupling: no need for names for agents to interact
Features of Linda: Generative Communication

- **Communication orthogonality**: both senders and the receivers can interact even without having prior knowledge about each others
  - space uncoupling (also called distributed naming): no need to coexist in space for two agents to interact
  - time uncoupling: no need for simultaneity for two agents to interact
  - name uncoupling: no need for names for agents to interact
Features of Linda: Generative Communication

- **Communication orthogonality**: both senders and the receivers can interact even without having prior knowledge about each others
  - space uncoupling (also called distributed naming): no need to coexist in space for two agents to interact
  - time uncoupling: no need for simultaneity for two agents to interact
  - name uncoupling: no need for names for agents to interact
Features of Linda: Generative Communication

- **Communication orthogonality**: both senders and the receivers can interact even without having prior knowledge about each others
  - space uncoupling (also called distributed naming): no need to coexist in space for two agents to interact
  - time uncoupling: no need for simultaneity for two agents to interact
  - name uncoupling: no need for names for agents 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: 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: 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: 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: 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: 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: 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: 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: 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

- **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

- **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

- `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

- The `in` and `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

- `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

- \texttt{in} & \texttt{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
  - \textit{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
  - \textit{in the coordination entity} the invocation may cause a wait-state in the invoker: hypothesis on the internal behaviour of the coordinable
1. Elements of Multi-agent Systems Engineering

2. Coordination: A Meta-model

3. Enabling vs. Governing Interaction

4. Classifying Coordination Models

5. Introduction to (Tuple-based) Coordination
   - Tuple-based Coordination & Linda
Bibliography I

Coordination models and languages as software integrators.

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

Coordination languages and their significance.

Coordination models and languages.
Conclusions

Coordination Models & Languages
Multiagent Systems LS
Sistemi Multiagente LS

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2009/2010