

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



- 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



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



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



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



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

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?

Open MAS

- No hypothesis on the agent life & behaviour

Distributed MAS

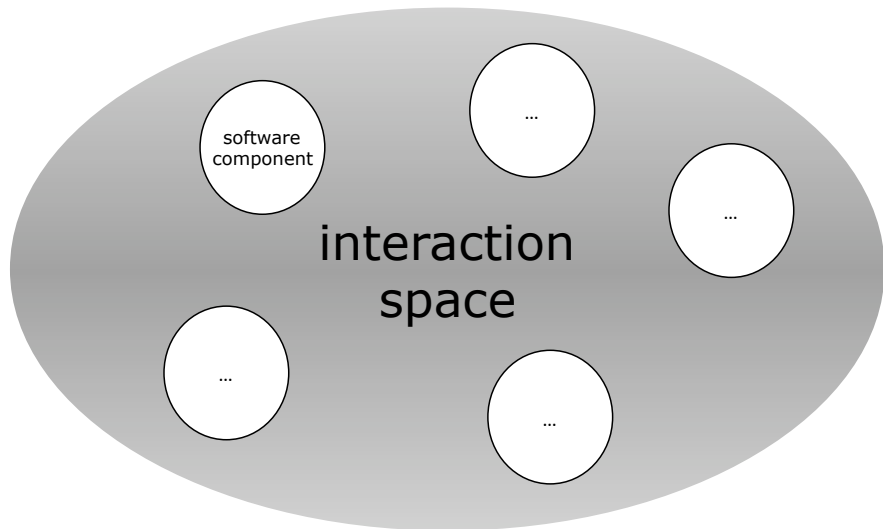
- No hypothesis on the agent location & motion

Heterogeneous MAS

- No hypothesis on the agent nature & structure



The Space of Interaction



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

Non-compositionality

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



Compositionality vs. Non-compositionality

Compositionality

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

Non-compositionality

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



Non-compositionality

Issues

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

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 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 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 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]

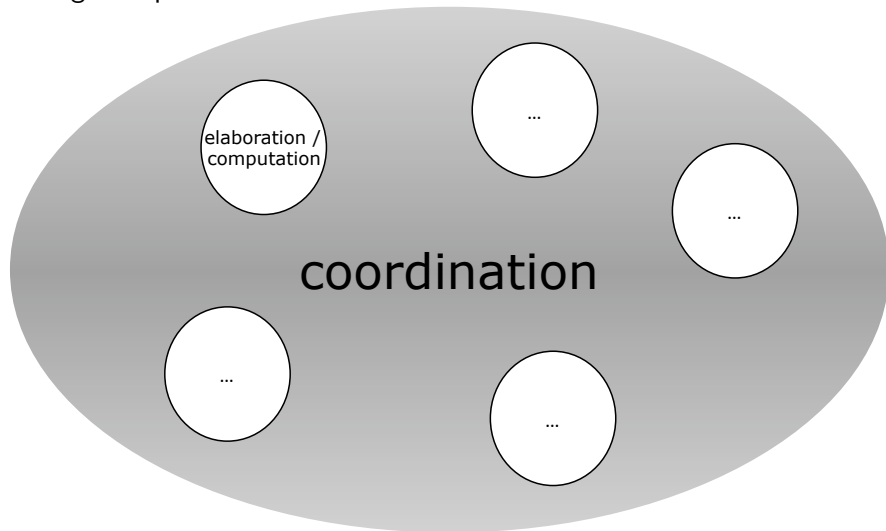
Issues for a coordination model

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



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



New Perspective on Computational Systems

Programming languages

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

Software engineering

- Interaction as an independent design dimension
- Coordination patterns

Artificial intelligence

- Interaction as a new source for intelligence
- Social intelligence



Coordination: Sketching a Meta-model

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



Coordination: Sketching a Meta-model

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



Coordination: Sketching a Meta-model

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



Coordination: Sketching a Meta-model

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



Coordination: Sketching a Meta-model

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



Coordination: Sketching a Meta-model

The *medium of coordination*

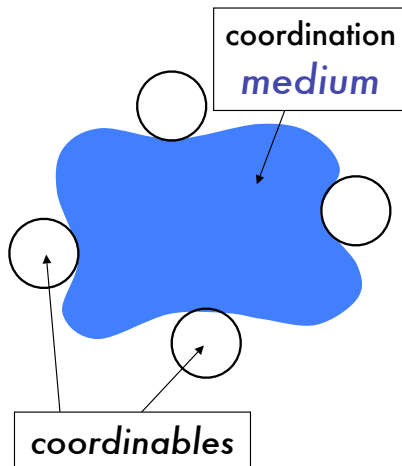
- “fills” the interaction space
- enables / promotes / governs the admissible / desirable / required interactions among the interacting entities
- according to some *coordination laws*
 - enacted by the behaviour of the medium
 - defining the semantics of coordination



Coordination: Sketching a Meta-model

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



Coordination: A Meta-model [Ciancarini, 1996]

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



Coordination: A Meta-model [Ciancarini, 1996]

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



Coordination: A Meta-model [Ciancarini, 1996]

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



Coordination: A Meta-model [Ciancarini, 1996]

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



Coordination: A Meta-model [Ciancarini, 1996]

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



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



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 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 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
 - the *coordination language*, as the set of the admissible interaction primitives, along with their semantics



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
 - the *coordination language*, as the set of the admissible interaction primitives, along with their semantics



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
 - the *coordination language*, as the set of the admissible interaction primitives, along with their semantics



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 admissible interaction primitives, along with their semantics

examples in/out/rd (Linda), send/receive (channels), push/pull (pipes), ...



Toward a Notion of Coordination Model

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



Examples of Coordination Mechanisms II

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



Examples of Coordination Mechanisms IV

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?



Examples of Coordination Mechanisms V

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



Examples of Coordination Mechanisms VI

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 vs. Governing Interaction I

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



Enabling vs. Governing Interaction II

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

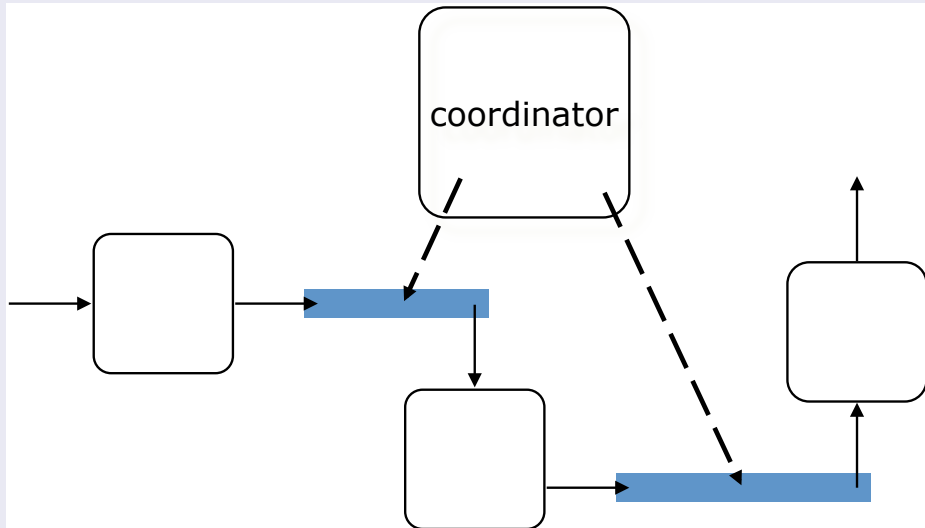
Coordinators. . .

- . . . 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

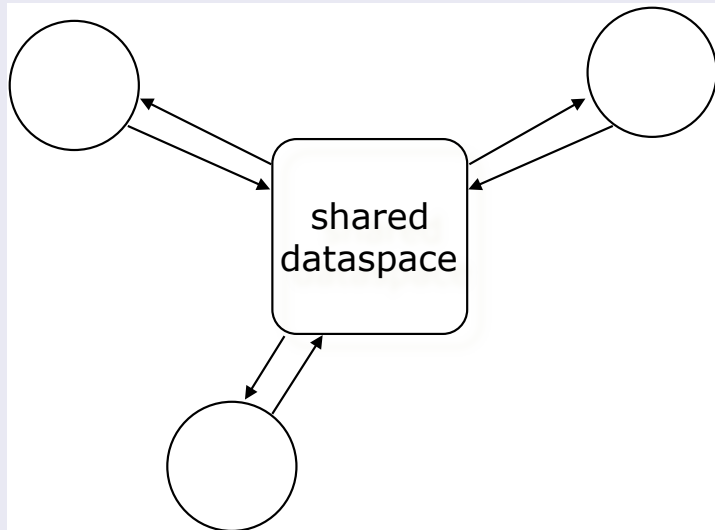
Coordination

- 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

- *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

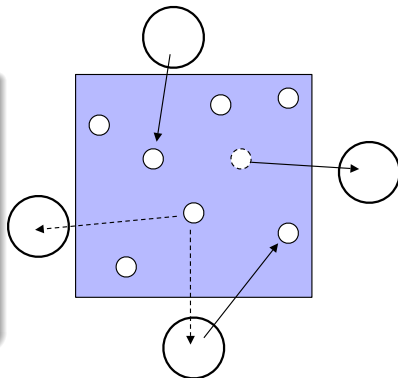
- *Coordinables* synchronise, cooperate, compete
 - based on *tuples*
 - available in the *tuple space*
 - by *associatively* accessing, consuming and producing tuples



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



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



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



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



Tuple-based / Space-based Coordination Systems

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

coordination media tuple spaces

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

communication language tuples

- as ordered collections of (possibly heterogeneous) information items

coordination language tuple space primitives

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



Linda: The Communication Language [Gelernter, 1985]

Communication Language

tuples ordered collections of possibly heterogeneous information chunks

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

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

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

tuple matching mechanism the mechanism 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, ...



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, ...



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, ...



Linda: The Coordination Language [Gelernter, 1985] I

out(T)

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

examples out(p(1)), out(0,0.5), out(course('Denti
Enrico', 'Poetry', hours(150))) ...



Linda: The Coordination Language [Gelernter, 1985] II

in(TT)

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

destructive reading the tuple retrieved is removed from the tuple centre

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

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

examples `in(p(X))`, `in(0,0.5)`, `in(course('Denti Enrico',Title,hours(X)) ...`



Linda: The Coordination Language [Gelernter, 1985] III

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 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 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 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 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 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

```
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

```
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

```
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

```
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

```
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

$= \text{in}(TT)$, $\text{rd}(TT)$ (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

`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



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



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 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
- `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: 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 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 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 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 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`



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



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



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



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



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 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 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 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 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 an given item
- Tuple structure based on
 - arity
 - type
 - position
 - information content
- Anti-tuples / Tuple templates
 - to describe / define sets of tuples
- Matching mechanism
 - to define belongingness to a set



Features of Linda: Generative Communication

- *Communication orthogonality*: both senders and the receivers can interact even without having prior knowledge about each others
 - space uncoupling (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

- `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




- 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

 Ciancarini, P. (1996).
Coordination models and languages as software integrators.
ACM Computing Surveys, 28(2):300–302.

 Gelernter, D. (1985).
Generative communication in Linda.
ACM Transactions on Programming Languages and Systems, 7(1):80–112.

 Gelernter, D. and Carriero, N. (1992).
Coordination languages and their significance.
Communications of the ACM, 35(2):97–107.

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



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

