Interaction & Coordination in Distributed Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
`andrea.omicini@unibo.it`

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2014/2015

Outline

Outline

Scenarios for Concurrent / Distributed Systems

Issues

- Concurrency / Parallelism
  - Multiple independent activities / loci of control
  - Active simultaneously
  - Processes, threads, actors, active objects, agents. . .
- Distribution
  - Activities running on different and heterogeneous execution contexts (machines, devices, . . .)
- "Social" Interaction
  - Dependencies among activities
  - Collective goals involving activities coordination / cooperation
- "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 / evolution
  - On-line engineering [FG04]
  - (Cognitive) Self-organising systems [Omi12]

Which Components?

Open systems

- No hypothesis on the component's life & behaviour

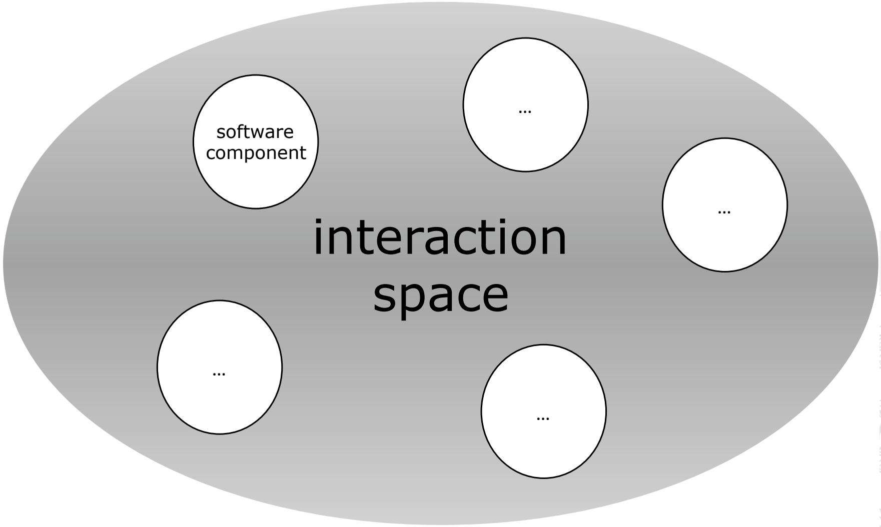
Distributed systems

- No hypothesis on the component's location & motion

Heterogeneous systems

- No hypothesis on the component's nature & structure

The Space of Interaction



Components of an Interactive System

What is a component of an interactive system?

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

(Non) Algorithmic Computation I

Elaboration / computation

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

(Non) Algorithmic Computation II

The power of interaction [WG03]

Real computational systems are not rational agents that take inputs, compute logically, and produce outputs... It is hard to draw the line at what is intelligence and what is environmental interaction. In a sense, it does not really matter which is which, as all intelligent systems must be situated in some world or other if they are to be useful entities. [Bro91]

... a theory of concurrency and interaction requires a new conceptual framework, not just a refinement of what we find natural for sequential [algorithmic] computing. [Mil93]

(Non) Algorithmic Computation III

Beyond Turing Machines

- Turing's *choice machines* and *unorganised machines* [WG03]
- Wegner's Interaction Machines [GSW06]
- Examples: AGV, Chess oracle [Weg97]

Basics of Interaction

Component model

A simple component exhibits

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

Coupling across component's boundaries

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

Information-driven interaction

Output	shows part of its state outside
Input	bounds a portion of its own state to the outside

Compositionality vs. Non-compositionality

Compositionality

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

Non-compositionality

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

Non-compositionality

Issues

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

Outline

Coordination in Distributed Programming

Coordination model as a glue

A coordination model is the glue that binds separate activities into an ensemble [GC92]

Coordination model as an agent interaction framework

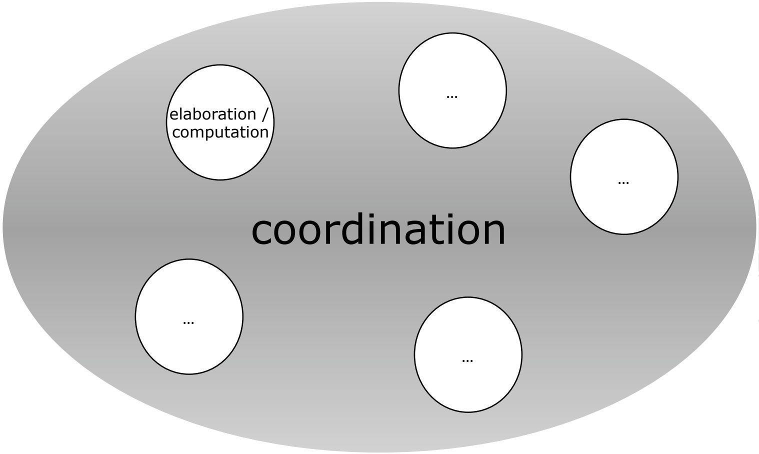
A coordination model provides a framework in which the interaction of active and independent entities called agents can be expressed [Cia96]

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 [Cia96]

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

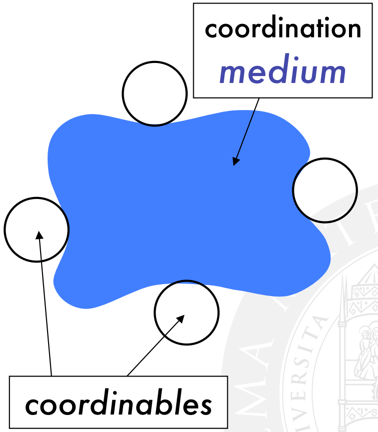
Artificial intelligence

- Interaction as a new source for intelligence
- Social intelligence

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 [Cia96]

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 interaction among coordinables

Coordination laws Laws ruling the observable behaviour of coordination media and coordinables, and their interaction as well

Coordinables

Original definition [Cia96]

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 [Cia96]

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 I

Original definition [Cia96]

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

Coordination Laws II

- Coordination laws rule the observable behaviour of coordination media and coordinables, as well as their interaction
  - a notion of (*admissible interaction*) *event* is required to define coordination laws
- The interaction events are (also) expressed in terms of
  - the communication language, as the syntax used to express and exchange data structures

  examples tuples, XML elements, FOL terms, (Java) objects, ...
  - the coordination language, as the set of the asmissible interaction primitives, along with their semantics

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

Outline

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 “real” 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

Outline

Two Classes for Coordination Models

Control-oriented vs. Data-oriented Models

— Control-driven vs. Data-driven Models [PA98]

Control-oriented Focus on the *acts* of communication

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

— Several surveys, no time enough here

— Are these really *classes*?

- actually, better to take this as a criterion to observe coordination models, rather than to separate them

Control-oriented Models I

Processes as black boxes

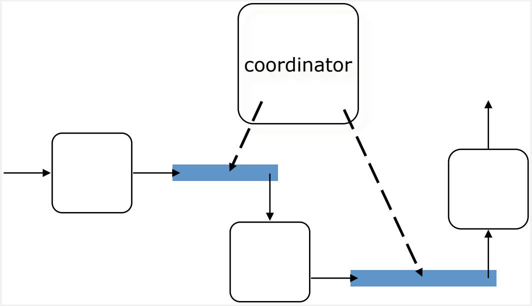
- I/O ports
- events & signals on state

Coordinators...

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

Control-oriented Models II

Coordinators as meta-level communication components



Control-oriented Models III

General features

- High flexibility, high control
- Separation between communication / coordination and computation / elaboration
- Examples
  - RAPIDE [LKA$^+$95]
  - Manifold [AHS93]
  - ConCoord [Hol96]
  - Reo [Arb04, DAdB05]

A Classical Example: Manifold [AHS93]

Main features

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

Control-oriented Models: Impact on Design

Which abstractions?

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

Which systems?

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

An Evolutionary Pattern?

Paradigms of sequential programming

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

Paradigms of coordination programming

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

Data-oriented Models I

Communication channel

- Shared memory abstraction
- Stateful channel

Processes

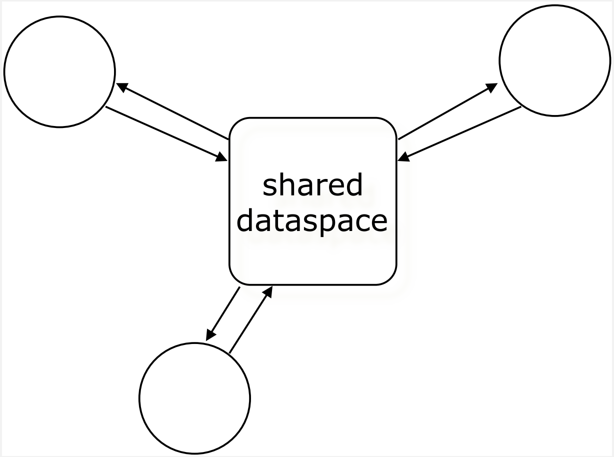
- Emitting / receiving data / information

Coordination

- Access / change / synchronise on shared data

Data-oriented Models II

Shared dataspace: constraint on communication



Data-oriented Models

General features

- Expressive communication abstraction

$\rightarrow$ information-based design

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

Summing Up

Coordination for distributed system engineering

- Engineering the space of interaction among components

Coordination as governing interaction

- Enabling vs. governing

Classes and features of coordination models

- Control-oriented vs. data-oriented models

References I

Farhad Arbab, Ivan Herman, and Per Spilling.
An overview of Manifold and its implementation.
Concurrency: Practice and Experience, 5(1):23–70, February 1993.

Farhad Arbab.
Reo: A channel-based coordination model for component composition.
Mathematical Structures in Computer Science, 14:329–366, 2004.

Rodney A. Brooks.
Intelligence without reason.
In John Mylopoulos and Ray Reiter, editors, *12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, volume 1, pages 569–595, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

References II

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

Mehdi Dastani, Farhad Arbab, and Frank S. de Boer.
Coordination and composition in multi-agent systems.
In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael J. Wooldridge, editors, *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 439–446, Utrecht, The Netherlands, 25–29 July 2005. ACM.

References III

Martin Fredriksson and Rune Gustavsson.
Online engineering and open computational systems.
In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organization*, pages 377–388. Kluwer Academic Publishers, 2004.

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

Dina Q. Goldin, Scott A. Smolka, and Peter Wegner, editors.
Interactive Computation: The New Paradigm.
Springer, September 2006.

References IV

Anne-Alexandra Holzbacher.
A software environment for concurrent coordinated programming.
In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models*, volume 1061 of *LNCS*, pages 249–266. Springer-Verlag, 1996.
1st International Conference (COORDINATION '96) Cesena, Italy, April 15–17, 1996.

David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann.
Specification and analysis of system architecture using Rapide.
IEEE Transactions on Software Engineering, 21(4):336–354, April 1995.

References V

Robin Milner.
Elements of interaction: Turing award lecture.
Communications of the ACM, 36(1):78–89, January 1993.

Andrea Omicini.
Agents writing on walls: Cognitive stigmergy and beyond.
In Fabio Paglieri, Luca Tummolini, Rino Falcone, and Maria Miceli, editors, *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, volume 20 of *Tributes*, chapter 29, pages 543–556. College Publications, London, December 2012.

References VI

George A. Papadopoulos and Farhad Arbab.
Coordination models and languages.
In Marvin V. Zelkowitz, editor, *The Engineering of Large Systems*, volume 46 of *Advances in Computers*, pages 329–400. Academic Press, 1998.

Peter Wegner.
Why interaction is more powerful than algorithms.
Communications of the ACM, 40(5):80–91, May 1997.

Peter Wegner and Dina Goldin.
Computation beyond Turing machines.
Communications of the ACM, 46(4):100–102, April 2003.

Interaction & Coordination in Distributed Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2014/2015