Computing with Space

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 2016/2017

About these Slides

- the following slides borrow a lot from the tutorial "Spatial Multi-Agent Systems" held at the 18th European Agent Systems Summer School (EASSS 2016)
- original slides can be found at

  `http://www.slideshare.net/andreaomicini/spatial-multiagent-systems`
- developed with the help of Stefano Mariano and Mirko Viroli

Next in Line. . .

Humans & Space

Environment awareness, measure & modelling

For early humans

- awareness of the surrounding environment as the premise to self-awareness [Martelet, 1998]
- measure and modelling of the environment as on of the premises to goal-driven actions

Space & activity

For humans, space is a conceptual tool

- to model the environment where we live
- to organise resources and activities
- and their dynamics as well

Spatial Reasoning I

Representing space

- the way in which we represent space mutually affects the way in which we can *reason about* space
- in some way not well understood, yet
  - for instance, [Li and Gleitman, 2002] and [Levinson et al., 2002] explicitly express two clearly contrasting views on the matter

Spatial Reasoning II

Reasoning on space

- humans have their own ways to represent space, and to process spatial information [Byrne and Johnson-Laird, 1989]
- spatial reasoning is a feature of the intelligent process—humans are not alone [Gentner, 2007, Haun et al., 2006]
- human spatial reasoning is may be either qualitative or quantitative—but is mostly *approximated* [Dutta, 1988]

! for a well-organised account of how humans (learn to) represent and reason about space see [Clements and Battista, 1992]

Focus on. . .

Geometry

Modelling space

Abstracting away from our perception of reality

- basic geometric concepts (point, line, angle, circle, . . .)
- basic geometric shapes (triangle, rectangle, trapezoid, . . .)
- from Babylonians to Egyptians to Greeks
- Babylonian *astronomy*
- Greek *geometry*

Euclidean Geometry

Axiomatic approach to geometry

- Euclide's *Elements* [Kline, 1972]
- geometry [Aiello et al., 2012]
  - no longer seen "as a set of empirical observations and practical methods for measuring distances, area of land, etc."
  - instead conceived as "an abstract mathematical theory, which, while rooted in the perceived reality, had nevertheless its own, absolute right of existence and development"
- representing space and reasoning on space through axioms, theorems, proofs, . . .

Geometry & Space beyond Human Perception

Modelling "imaginary" space

- beyond Euclidean space
  - physical space may not satisfy Euclid's fifth postulate
  - non-euclidean geometry
  - Riemann (elliptic), Bolyai & Lobachevsky (hyperbolic), ...
- representing *true* physical space beyond our direct sensorial-cognitive perception

[Aiello et al., 2012]

> *That discovery was an unsurpassed manifestation of the superiority of the abstract, logical approach of mathematics over the empirical approach underpinning the natural sciences, at least when it comes to comprehending such fundamental physical concepts as space and time.*

Focus on. . .

From Euclid to Tarski [Aiello et al., 2012] I

Basic question

Morris Kline, Foreword [Russell, 1956]

> *What geometrical knowledge must be the logical starting point for a science of space and must also be logically necessary to the experience of any form of externality?*

Axiomatic investigations of the foundations of geometry

- axiomatic approach to geometry from Euclid to Peano
- studying relationships between axiomatic systems & basic geometric notions
- (sound) *axiomatic* foundation for geometry by Hilbert [Hilbert, 1950]

From Euclid to Tarski [Aiello et al., 2012] II

Analytic method in geometry of space

- Descartes' *coordinatisation* of the Euclidean space
- coordinate systems to solve purely geometric problems by using purely algebraic methods
- geometry as a study not of figures, but of *transformations* (Klein's Erlangen program [Balbiani et al., 2007]) preserving fundamental geometrical properties
- abstract *algebraic* foundation for geometry

From Euclid to Tarski [Aiello et al., 2012] III

Logical foundation of geometry

- *elementary geometry* as first-order theory of Euclidean geometry [Tarski, 1959]
- elementary geometry can be developed axiomatically using just two geometric relations—betweenness and equidistance
- elementary geometry like field of reals via *coordinatisation*
- completeness and decidability of the first-order theory of the field of reals implies an explicit decision procedure for the elementary geometry
- (non efficient) *algorithm* for deciding the truth of any statement in Euclidean geometry

Modal Logics

Non-classical logics

- modelling, analysing, and reasoning about space with non-classical logics
- *modal logics* have proven to be particularly interesting, since
  - they can be more *specific*
  - they have better computational behaviour—very often *decidable*

Modal Logics of Topology I

Distance & metric

- basic problem: *distance*
- approach: notions of *metric*

Topology

- *neighbourhood* as a generalisation of metric
- from neighbourhood to *topology* [Singer and Thorpe, 1967]
- "alongside algebra and geometry, topology became one of the fundamental branches of contemporary mathematics" [Aiello et al., 2012]

Modal Logics of Topology II

Logic and topology

- modal operators reinterpreted
  - $\Box$ *(necessarily)* as the interior operator
  - $\Diamond$ *(possibly)* as the closure operator of a topological space
- S4 [Bull and Segerberg, 1984] is *sound* and *complete* with respect to topological semantics [McKinsey and Tarski, 1944]
- S4 is the *modal logic* of any Euclidean space

Mathematical Morphology [Aiello et al., 2012]

- mathematical morphology (MM) analyses *shape*, *spatial information*, *image processing*
- any mathematical theory dealing with shapes can contribute to MM
- *modal morphologic* from the similarity between the algebraic properties of MM operators and of modal operators
- efficient for spatial reasoning
  - to guide the exploration of space, in a focus of attention process, and for recognition and interpretation tasks

Summary

- we have math and logic tools to *represent*, *analyse*, and *reason* about space
- we can *compute* about space and its organisation with diverse levels of efficiency

Next in Line. . .

Focus on. . .

Distributed Systems: Recap

A new space for software components

- physical distribution of computational systems
  - distribution of *computational units*, *communication channels*, *data*
- local networks building the first virtual spaces
- Internet and IP-based locations for first "global" virtual spaces
- WWW as the first global space shared by agents and humans
  - a *knowledge-intensive* space

Middleware: Recap

Structured environments for software components

- *middleware* as operating systems for distributed systems
  - *software infrastructure* giving structure to *distributed environments*
  - possibly supporting *mobility* [Fuggetta et al., 1998a]
- EAI horizontal integration to build *aggregated environments*

Mapping logical distribution upon physical distribution

- middleware
  - provides topological notions for distributed systems
  - maps logical distribution upon physical distribution

JADE [Bellifemine et al., 2007] provides agent programmers with the topological notions of container and platform to represent *locality* for agents

! *sometimes, however, physical distribution is what actually matters*

Distributed Systems & Situated Computations

Situated distributed systems

- physical distribution of computational systems is essential to cope with the distributed nature of many working environments
- . . . as well as with the need for situated computation
  - that is, computations occurring locally where either perception or action are taking place
  - either elaborating on perception, driving action, or both
- essentially, when system requirements mandate for situated computations within a distributed physical environment, situated distributed systems are the only way out
  - e.g., disaster recovery scenarios, environmental monitoring, crowd steering, . . .
- Internet of Things (IoT) just makes the need for situated computation unescapable

Pervasive Systems: Recap

Physical space

- pervasive systems emphasise the role of physical environment
- physical distribution of computational systems to cope with the *distributed nature* of physical application scenarios
- a distributed pervasive system is
  - is part of our surroundings
  - generally lacks of a human administrative control
  - is typically unstable

Example: Home Systems

Systems built around. . .

. . . home networks

- no way to ask people to act as a competent network / system administrator

$\rightarrow$ home systems should be self-configuring and self-maintaining in essence

. . . personal information

- huge amount of heterogeneous personal information to be managed
- coming from heterogeneous sources from inside and outside the home system

. . . our personal, domestic *space*

- resources and activities at home are logically organised around our notion of domestic space
- which home systems need to model, adapt to, and fruitfully exploit

Situatedness: Recap I

Situated action [Suchman, 1987]

- (intelligent) actions are *purposeful*, and *not abstract*
- "every course of action depends in essential ways upon its material and social circumstances"
- situatedness is in short the property of systems of being *immersed* in their environment
- that is, of being capable to *perceive* and *produce* environment change, by suitably dealing with environment events
- mobile, adaptive, and pervasive computing systems have emphasised the key role of situatedness for nowadays computational systems [Zambonelli et al., 2011]

Situatedness: Recap II

Situatedness & context-awareness

- computational systems are more and more affected by their physical nature
- this is not due to a lack of abstraction: indeed, we are suitably layering systems – including hardware and software layers –, where separation between the different layers is clear and well defined
- instead, this is mostly due to the increasingly complex requirements for computational systems, which mandate for an ever-increasing context-awareness [Baldauf et al., 2007]
- defining the notion of *context* is a complex matter [Omicini, 2002], with its boundaries typically blurred with the notion of *environment*, in particular in the field of *multi-agent systems* (MAS) [Weyns et al., 2007]

Situatedness: Recap III

Context-awareness: environment

- situatedness of computational systems – MAS in particular – requires awareness of the environment where the systems are deployed and are expected to work
- computational systems and their components need to understand their working environment, its *nature*, its *structure*, the *resources* it makes available for use, the possible *issues* that may harm the systems in any way

Situatedness: Recap IV

Context-awareness: space & time

- in particular, mobile computing scenarios have made clear that *situatedness* of computational systems nowadays requires at least *awareness of the spatio-temporal fabric*
- computational systems and their components need to know *where* it is working, and *when*, in order to effectively perform its function
- in its most general acceptation, then, any (working) environment for nowadays non-trivial computational systems is first of all made of space and time

Summary

- distributed systems originate from *physical distribution* of computation, communication, and data
- middleware provides logical topology upon physical distribution
- pervasive systems and IoT make situated computation a requirement, and *situated distributed systems* the only viable approach
- nowadays non-trivial computational systems needs to be space-aware

Focus on. . .

Moving Code

Sometimes passing data is not enough

- sometimes we would like to change the place where the code is executed—for load balancing, security, scalability, . . .
- sometimes we do not like to separate the data from the code to be executed on them (e.g., objects, agents)

→ then, passing data between processes is no longer enough

→ code should be passed

Reasons for Migrating Code

Process migration

- traditionally, code is moved along with the whole computational context
- moving code is typically moving processes [Milojicic et al., 2000]

Why?

- load balancing
- minimising communication
- optimising perceived performance
- improving scalability
- flexibility through dynamic configurability
- improving fault tolerance

Models for Code Migration

There is much more than just moving code

- what do we move along with a program?
- execution status, pending signals, data, . . .

Understanding code mobility [Fuggetta et al., 1998b]

- a process can be thought as three segments

  code segment the set of the executable instructions of the process

  resource segment the set of the references to the external resources needed by the process—like files, printers, devices, other processes, . . .

  execution segment the store for the execution state of the process—with private data, stack, program counter

- depending on what is moved along with the code, we can classify different types of code mobility

Weak Mobility

The bare minimum for code migration

- only the code segment is transferred
- possibly along with some initialisation data

Main idea

- the code can be executed every time *ex novo*
- so, we do not care about any computational context
- or, maybe, the computational context we need is the target one

Main benefit

- the only requirement is that the target machine can execute the code
- weak mobility is very simple
- it has no particular restrictions or further requirements to be implemented

Strong Mobility

Moving execution context

- the execution segment is transferred along with the code segment

Main benefit

- a process can be stopped, moved, and then restart on another machine

Requirements

- strong mobility is very demanding
- technological environment should allow for it

Sender- vs. Receiver- Initiated Migration I

Sender-initiated migration

- migration is initiated where the code resides / is being currently executed
- examples: search-bots, mobile agents
- servers should know clients, and ensure security of resources

→ more complex interaction scheme

Sender- vs. Receiver- Initiated Migration II

Client-initiated migration

- migration is initiated by the target machine, requiring a new behaviour to be added
- examples: Java Applets, JavaScript chunks
- just a few resources on clients need to be secured
- clients may also be anonymous

→ less complex interaction scheme

Separate vs. Target Process Execution

Weak mobility and execution of mobile code

- in the case of weak mobility, one may execute the mobile code on either the target process or a separate process
- for instance, Java Applets are executed in the browser's address space

→ no need for inter-process communication at the target machine

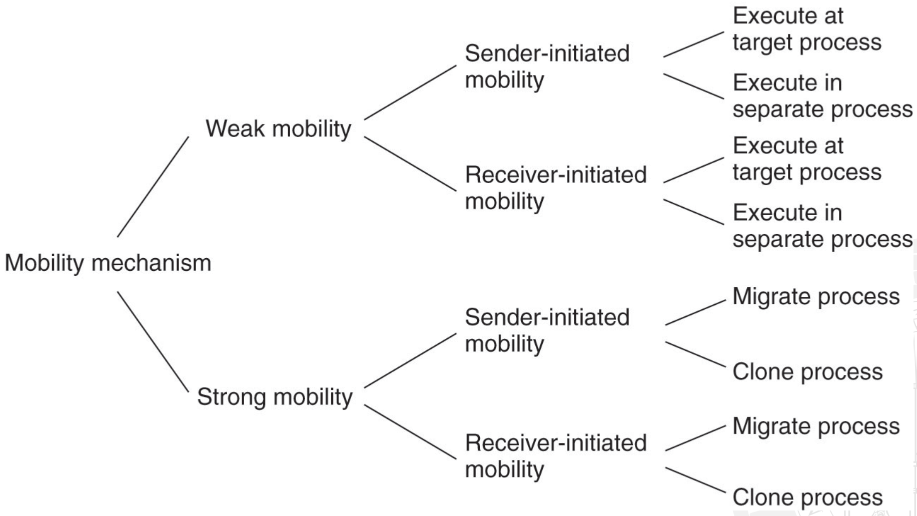
- main problem: protection against malicious or buggy code execution
- solution: assigning mobile code execution to a separated process

Cloning vs. Migrating

Strong mobility can be supported also by remote cloning

- cloning yields an exact copy of the original process, executed on the target machine
- cloned process is executed in parallel to the original process, on different machines
- example: In UNIX, forking a child process and let it execute on a remote machine
- cloning is an alternative to migration
- cloning in some sense improve distribution transparency, in that the processes are transparently replicated on many different machines

Models for Code Migration


Mobility mechanism
Weak mobility
Sender-initiated mobility
Execute at target process
Execute in separate process
Receiver-initiated mobility
Execute at target process
Execute in separate process
Strong mobility
Sender-initiated mobility
Migrate process
Clone process
Receiver-initiated mobility
Migrate process
Clone process

Alternatives for code migration [Tanenbaum and van Steen, 2007]

Migration and Local Resources

Migration and the resource segment

- till now, we have only accounted for migration of the code and execution segments
- main problem: resources might not be as easy to move around as code and variables
- example: a huge database might in theory be moved across the network, but in practice it will not
- either references need to be updated, or resources need to be moved

Two issues

- how does the resource segment refer to resources?
- how does the resource relate with the hosting machine?

How does the resource segment refer to resources?

Process-to-resource binding

binding by identifier need of a resource with a given name—e.g., via an URL, or a local ID

binding by value need of a resource based on its value—e.g., code libraries

binding by type need of a resource based on its type—typically, local devices like printers, monitors, . . .

How does the resource relate with the hosting machine?

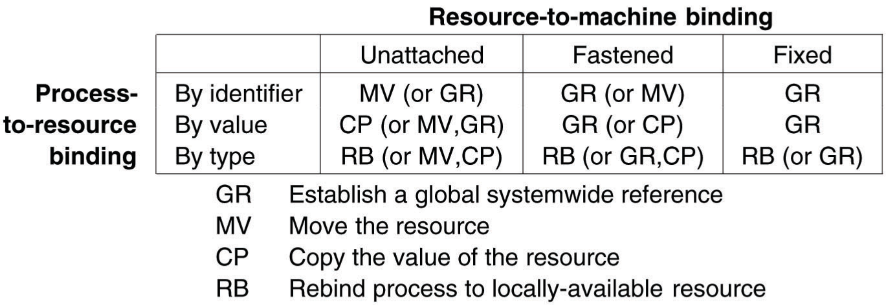
Resource-to-machine binding

unattached resources
: resources that can be easily moved between different machines—like, files associated to the migrating code

fastened resources
: resources that can be moved, but at a cost—like, a local database

fixed resources
: resources bounded to a specific machine—like, a monitor

Code Migration and Local Resources



		Resource-to-machine binding		
		Unattached	Fastened	Fixed
Process-to-resource binding	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV,GR)	GR (or CP)	GR
	By type	RB (or MV,CP)	RB (or GR,CP)	RB (or GR)

GR	Establish a global systemwide reference
MV	Move the resource
CP	Copy the value of the resource
RB	Rebind process to locally-available resource

Actions to be taken with respect to the references to local resources when migrating code to another machine

[Tanenbaum and van Steen, 2007]

Summing Up

Code migration

- code may move through distributed machines for a number of good reasons
- different types of code mobility are possible, depending on either the application needs or the technology constraints

Focus on. . .

Computational Geometry [de Berg et al., 2000]

Computing with geometry to solve problems

- using *space* to *represent problems*—either spatial or non spatial ones
- using *geometry* to make them *computable*
- using *algorithms* to compute solutions

Example

- finding the nearest coffee shop in the campus here in Cesena
- finding it for any point in the campus
- dividing the campus in regions around each coffee shop, including all the points for which each shop is the nearest one
- how do we compute this?

Geographical Information Systems (GIS)

Computing with geography

- representing geographic information
- capturing / storing / checking / displaying data representing positions on the Earth—maybe other planets, too, in the (near) future

→ the notion of space is clearly defined in GIS

- more generally

  > *A geographic information system is a computer-based system that supports the study of natural and man-made phenomena with an explicit location in space. To this end, the GIS allows data entry, data manipulation, and production of interpretable output that may provide new insights about the phenomena.* [Huisman and de By, 2009]

Virtual Reality (VR)

Activity in a virtual environments

- artificial worlds—with artificial *space*
- digitally-created, represented exclusively as *computational environments*
- where “real people” can actually perform sensorimotor and cognitive activity

Definition [Fuchs et al., 2011]

> *Virtual reality is a scientific and technical domain that uses computer science (1) and behavioural interfaces (2) to simulate in a virtual world (3) the behaviour of 3D entities, which interact in real time (4) with each other and with one or more users in pseudo-natural immersion (5) via sensorimotor channels.*

! *here, interaction and immersion are the key concepts*

Gaming

Virtual worlds for gaming

- gaming is the most prominent application of virtual reality
  - e.g., Microsoft Kinect is one of the most well-known applications of VR
- gaming platforms nowadays provide the means for building whole virtual worlds
  - e.g., Unity3D, Unreal Engine

  that can be explored from a wide range of diverse devices

Focus on. . .

Location-based Services (LBS)

Adding a virtual layer to physical space

- LBS use information about the position of a user / device to provide information, entertainment, security
  - e.g., AroundMe, Pokemon Go
- *mixed reality* is an old term possibly coming back to use
  - yet, with an uncertain meaning

Augmented Reality (AR)

Integrating virtual and physical space

- merging real-world information with *context-sensitive* digital content in a meaningful way [Furht, 2011]
  - the point here is that the virtual and physical layers mutually affect each other dynamically
- gamification can easily lead to any sort of relevant application—medical, civic, educational, touristic, . . .
  - e.g., GeoZombie [Prandi et al., 2016]—a step further (and before) Pokémon Go

Focus on. . .

Altogether Now: Spatial Computing I

Whenever space cannot be abstracted away from computation

[Shekhar et al., 2016]

During the "Computing Media and Languages for Space-Oriented Computation" workshop in Dagstuhl (`http://www.dagstuhl.de/06361`), three classes of spatial systems were identified

- *distributed systems*, where space is either a means or a resource—a.k.a. intensive computing, or *coping with space*
- *situated systems*, where location in space (and time) is essential for computation—a.k.a. *embedded in space*
- *spatial systems*, where space is fundamental to the application problem, is explicitly represented and manipulated, and is essential to express the result of a computation—a.k.a. *representing space*

Altogether Now: Spatial Computing II

Spatial computers, spatial computing [Beal et al., 2011]

- a spatial computer – or, *spatial computing system* – is a computational system where (the logic of) space is essential in representing the problem, define computation, and express the result
- spatial computing is any form of computation where (the logic of) space is relevant to express and perform computation

Spatial Computing Languages (SCL)

Languages for spatial computing

Spatial Computing Languages (SCL)

- were born to address the issue of bringing space *into* programming languages
- allowing programmers to explicitly deal with space-related aspects *at the language level*

Proto

Archetypical SCL example

Proto [Viroli et al., 2013]

- is a purely functional language with a LISP-like syntax
- uses a continuous space abstraction (the *amorphous medium*) to model the notion of spatial computer
- has mathematical operations on space-time as primitives
- has programs specified in terms of geometric computations and information flow on a topological space

Proto *is a Domain-Specific Language (DSL) for representing the description of aggregate device behaviour in a spatial computer* [Viroli et al., 2013]

- it embeds the aggregate computing model [Beal et al., 2015] as an original paradigm to organise spatial computations
- it works as a sort of benchmark for SCL of any sort

Expressiveness of SCL [Beal et al., 2011] I

The Abstract Device Model

In order to allow comparison between different SCL, the Abstract Device Model (ADM) is defined in [Beal et al., 2013], working along three basic criteria

communication region — the spatial "coverage" of the communication—e.g., global vs. neighbourhood

communication granularity — the number of receivers—e.g., unicast vs. multicast

code mobility — the relationships between the running code of different devices

Expressiveness of SCL [Beal et al., 2011] II

Types of space-time operations

Furthermore, three classes of operators are required in SCL in order to achieve a kind of *spatial Turing equivalence* [Beal, 2010]

measure space — to translate spatial properties into computable information—e.g. sensing GPS position

manipulate space — to translate back information into modifications of spatial properties—e.g. starting a motion engine

compute (on) space — any kind of “spatial-pointwise” computation

A fourth class (*physical evolution*) looks more like an assumption about the dynamics that a given program/device must/can consider—e.g. a robot may move while a computation runs

Expressiveness of SCL [Beal et al., 2011] III

The "T-Program"

As a reference *benchmark* to compare the expressiveness of different SCL, the "T-Program" is proposed in [Beal et al., 2013] w.r.t. a set of independent moveable devices

i cooperatively create a local coordinate system
ii move or grow devices into a T-shaped structure
iii determine its center of gravity, then draw a ring around it

SCL Requirements

Hence it requires all the three classes of operators afore-mentioned: stage *(i)* requires *measuring space* capabilities; stage *(ii)* requires *manipulating space* capabilities; stage *(iii)* requires both *compute over space* capabilities and, again, measuring capabilities.

Expressiveness of SCL [Beal et al., 2011] IV

Local coordinate system

Setting a local coordinate system basically amounts to

i choosing an origin node
ii making it *spread* a vector tuple to neighbours
iii (recursively) making them increment such a vector
iv forwarding it to neighbours

Expressiveness of SCL [Beal et al., 2011] V

T-shaped structure

To arrange nodes (tuple centres) so as to form a T-shaped structure, it is required to

i define spatial constraints representing the T

ii make every node move so as to satisfy them

Thus, the basic mechanism needed at the VM level is motion monitoring and control

Expressiveness of SCL [Beal et al., 2011] VI

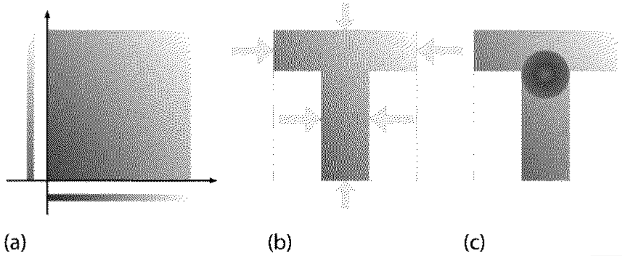
Center of gravity

To first compute the focal point (FC) of the T-shape, then draw a sphere around it, two basic mechanisms are needed, both similar to the *neighbourhood spreading* previously shown

- a *bidirectional* neighbourhood spreading to collect replies to sent messages—allowing to aggregate all the node's coordinates and count them
- a *spherical multicast* to draw the ring pattern

Expressiveness of SCL [Beal et al., 2011] VII

Figure 5. The "T program" reference example exercises the three main classes of space-time operations: measurement of space-time to organize local coordinates (a) and compute the center of gravity (c), manipulation of space-time to move devices into a T-shaped structure (b), and pattern computation to make a ring around the center of gravity (c).



[Beal et al., 2013]

Other SCL and Issues I

- a survey over SCL can be found in [Beal et al., 2013]
- diverse classes of SCL are listed
  - amorphous computing
  - biological
  - agent-based
  - wireless sensor networks (WSN)
  - pervasive computing
  - swarm and modular robotics
- and compared against the T-program

Other SCL and Issues II

Main issue

- the reference framework defined in [Beal et al., 2013] perfectly works in terms of language expressiveness when focussing on spatial issues
  - many diverse sort of SCL languages can be analysed and compared
  - so, just to be clear, it is perfect to its purpose
- however, we miss a general conceptual framework where all spatial languages could fit altogether
  - so as to make it possible, for instance, where do they *generally* belong in the engineering of complex software systems
  - e.g., TOTA [Mamei and Zambonelli, 2009] and Proto [Viroli et al., 2013] cannot really stay in the same place
- a more expressive conceptual framework for SCL – and spatial computing in general – would be dearly needed

Summary

- spatial computing is a sort of “landscape framework” for the many sorts of spatial computations and systems around
- an already-long list of SCL are available, which can be analysed and compared through the conceptual framework provided in [Beal et al., 2013]
- a more general framework for finding the right place for SCL in the general-purpose SE process would be needed

Next in Line. . .

Focus on. . .

Agents as Autonomous Entities: Recap [Omicini et al., 2008] I

Agent

Agents are *autonomous computational entities*

genus	agents are computational entities
differentia	agents are autonomous, in that they encapsulate control along with a criterion to govern it

Agents are *autonomous*

- from autonomy, many other features stem
  - autonomous agents *are* interactive, social, proactive, and situated
  - they *might* have goals or tasks, or be reactive, intelligent, mobile
  - they live within MAS, and *interact* with other agents through *communication actions*, and with the environment with *pragmatical actions*

Agents and Space I

Autonomy

- autonomy is an essential feature for distributed systems
  - coupling of control is maybe the main potential problem in distributed systems
  - uncoupling is required to prevent delays, deadlocks, faults
  - computational autonomy ensures uncoupling of control, since agents encapsulate control
- this is particularly clear in pervasive systems, where instability makes autonomy an essential feature

$\rightarrow$ spatial distribution mandates for autonomy

Agents and Space II

Reactiveness to change

- the ability to be sensitive to environment change is obviously the generalisation of the reactiveness to spatial changes in the surrounding of an agent
- which might be change of
    - either the relative or absolute position / motion of the agent
    - the position / motion of resources in the surrounding
    - the structure of the spatial environment

Agents and Space III

Situatedness

- the property of an agent of being immersed within its surrounding environment [Ferber and Müller, 1996]
  - is somehow a generalisation of reactiveness
  - is connected to context-awareness
  - is particularly relevant in term of spatio-temporal situatedness

Agents and Space IV

Mobility

- the ability of an agent of changing its own position within either logical or physical space, by either software or physical agents
- is particularly relevant since it is associated to autonomy—agents can say “Go!” [Odell, 2002]
- may require the ability to move autonomously, through e.g., sensorimotor actions
  - but may also be associated to agents hosted by mobile devices
- may obviously benefit by any form of spatial representation and reasoning

Agents and Space V

Intelligence

! we do not discuss (artificial) intelligence here

- however, intelligent agents are possibly the most suitable vessel for *spatial reasoning* [Kray, 2001]—including
  - languages for spatial representation
  - logics for spatial reasoning

Agents & Spatial Reasoning I

Spatial reasoning by cognitive agents

- cognitive abilities of intelligent agents can be in principle exploited for spatial reasoning [Dutta, 1988]
  - e.g., the ability to separately handle and properly use epistemic knowledge is an obvious benefit when dealing with spatial information
- diverse logics can be *embedded* into an agent architecture, so as to provide for different ways to reason about space
  - fuzzy logic in [Dutta, 1988]
  - hybrid logic for common sense reasoning in [Bandini et al., 2007]
  - propositional logic in [Bennett, 1992]

Agents & Spatial Reasoning II

Spatial reasoning by physical agents

- actually, the most effective work on spatial reasoning till now comes from robotics [de Berg et al., 2000]
  - robot teams [Moratz and Wallgrün, 2003]
  - robot swarms [Hamann, 2010]
  - robots as MAS [Williams and Sukhatme, 2012]
- a huge flow of literature, including spatial self-organisation [Zambonelli, 2004], robot coordination [Moratz and Wallgrün, 2003], etc.

Focus on. . .

MAS: Basic Abstractions

Agents in MAS

- agents live within an agent *society*
- agents live immersed within an agent *environment*

Basic design abstractions for MAS [Weyns et al., 2007]

agents	are the autonomous components of the systems, embodying the *designed* activities
society	is meant to model and govern the *relationships* among agents
environment	models the either virtual or physical context where the agents are situated, capturing both the effect of agent activities and the unpredictable change brought about by *non-designed* activities

Society and Coordination

Agent society

- autonomous agents encapsulate designed activities in a MAS
- agents live within an agent society
- an agent society models and governs mutual agent relationships

Interaction and coordination

- when agent relationships are expressed in terms of *mutual dependencies*, their govern is a matter of *coordination* [Malone and Crowston, 1994]
- *coordination models* and *languages* [Ciancarini et al., 2000] rule agent interaction within agent societies and MAS
- *coordination abstractions* – a.k.a. coordination media – model agent societies, and govern agent social relationships by *ruling their mutual interaction*

Coordination Models for MAS: Recap I

Coordination model as a glue

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

Coordination model as an agent interaction framework

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

Coordination Models for MAS: Recap II

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]

Examples

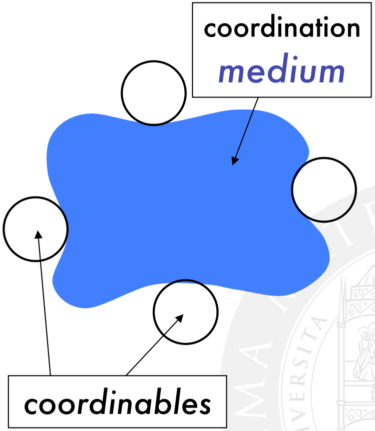
- blackboard-based systems [Corkill, 1991]
- LINDA coordination model [Gelernter and Carriero, 1992, Gelernter, 1985]

... along with its many derivative, a.k.a. *tuple-based* coordination models [Rossi et al., 2001, Omicini, 1999]

Coordination Models for MAS: Recap III

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 Models for MAS: Recap IV

Coordination: a meta-model [Ciancarini, 1996]

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

Coordination Models for MAS: Recap V

Example

For instance, in tuple-based coordination models

coordination entities are the *agents* interacting by exchanging *tuples* via `out`, `rd`, `in` *coordination primitives*

coordination media are the *tuple spaces* where tuples are written (`out`), read (`rd`), and consumed (`in`) by agents

coordination laws are the rules of model, such as

- *pattern matching* between tuples and templates in `rd` and `in`
- *suspensive semantics* for no match
- *non-determinism* for multiple matches

Society and Space

Spatial dependencies

- agent activities can be situated—first of all on their spatio-temporal features
- activities in a situated MAS are logically organised also based on space

→ agent activities may depend on each other on a spatial basis

Spatial coordination

- dealing with spatial dependency between agent activities mandates for *spatial coordination*
- *space-aware* coordination models [Mariani and Omicini, 2013] are required
- many remarkable examples have emerged in the last years

GeoLinda I

- two approaches to detect movement patterns
  - virtual ⇒ localisation and communication infrastructure creating a virtual model of the physical world ⇒ people and mobile objects regularly update their location
  - physical ⇒ coordination protocols between people and objects, both carrying wireless devices with restricted communication range ⇒ movement detection based on discovery protocols, no global info
- the physical approach can be implemented by distributed tuple spaces ⇒ tuples (dis)appearance based on overlapping of devices' communication range (*physical synchronization*) ⇒ limited number of movement patterns recognisable
  - no relative locations / directions—e.g., arriving from the left
  - detection precision depending on communication range—e.g., too big ⇒ coarse-grained detection

GeoLinda II

- GeoLinda [Pauty et al., 2007] associates a volume to each tuple (*tuple's shape*) and a volume to each reading operation(*addressing shape*)—e.g., sphere, cylinder, cone, box, sector, and point
- a reading operation is released when the shape of a matching tuple intersects with the addressing shape of the operation
- the programmer defines a tuple's shape *relatively* to the location and the *orientation* of the device which publishes this tuple
- similarly, he defines the addressing shape of a reading operation relatively to the location and orientation of the device which executes this operation

LIME I

- LIME (*Linda in a Mobile Environment*) [Murphy et al., 2006] deals with both physical and logical mobility
  - physical mobility involves the movement of mobile hosts
  - logical mobility is concerned with the movement of mobile agents—processes able to migrate from host to host while preserving code and state
- coordination takes place through *transiently shared* tuple spaces, that ties together physical and logical mobility
- movement, logical or physical, results in *implicit* changes of the tuple space accessible to the individual components
  - the system, not the application, is responsible for managing movement and the tuple space *restructuring* associated with connectivity changes
- tuple spaces are permanently bound to mobile agents and mobile hosts

Lime II

- transient sharing dynamically re-computes the set of *locally accessible* tuples in such a way that, for each mobile agent, the content of its local space gives the appearance of having been *merged* with those of the other mobile agents which are currently *co-located*

TOTA I

- *Tuples On The Air* (TOTA) [Mamei and Zambonelli, 2009] is a middleware and programming model for supporting *adaptive context-aware activities* in pervasive and mobile computing scenarios
- the key idea in TOTA is to rely on *spatially distributed tuples*, propagated across a network on the basis of application-specific rules
- a tuple can be "injected" into the network from any node and, after cloning itself, can diffuse across the network according to tuple-specific *propagation rules*
- once a tuple is spread over the network, it can be perceived as a single distributed data structure that we call a *tuple field* – made up by all the tuples created during the propagation of the injected tuple – to draw an analogy with physical fields (e.g., gravitational), which have different values (in TOTA, tuples) at different points in space (in TOTA, network nodes)

TOTA II

- the middleware takes care of propagating the tuples and adapting their values in response to the dynamic changes that can (possibly) occur in the network topology
- TOTA assumes the presence of a peer-to-peer network of possibly mobile nodes, each running a local instance of the TOTA middleware
- each TOTA node holds references to a limited set of *neighbour nodes* and can *communicate directly* only with them
- the structure of the network, as determined by the neighbourhood relations, is automatically maintained and updated by the nodes to support dynamic changes, either due to nodes' mobility or to their birth/death
- TOTA tuples can be defined at the application level and are characterized by a *content* `C`, a *propagation rule* `P`, and a *maintenance rule* `M`, hence `T = (C, P, M)`

TOTA III

content C
: an ordered set of typed elements representing the information carried on by the tuple

propagation rule P
: determines how the tuple should be distributed across the network; propagation typically consists in a tuple *(i)* cloning itself, *(ii)* being stored in the local tuple space, then *(iii)* moving to neighbour nodes: however, different kinds of propagation rules can determine the "scope" of the tuple – e.g., the distance at which such tuple should be propagated, the spatial direction of propagation, etc. – and how such propagation can be affected by the presence or the absence of other tuples in the system; in addition, the propagation rule can determine how the tuple's content C should change during propagation

TOTA IV

maintenance rule `M` determines how a tuple should react to *events* occurring in the environment—including flow of time; on the one hand, maintenance rules can preserve the proper spatial structure of tuple field despite network dynamics—thanks to TOTA middleware constantly monitoring the network local topology and the income of new tuples, eventually re-propagating tuples; on the other hand, tuples can be made *time-aware*, e.g., to support temporary tuples or tuples that slowly "evaporate"—in the spirit of pheromones [Parunak, 1997]

SAPERE I

- SAPERE (Self-aware Pervasive Service Ecosystems) [Zambonelli et al., 2015] is a EU STREP Project founded within the EU 7FP—FP7-ICT-2009.8.5: Self-awareness in Autonomic Systems
- SAPERE considers modelling and architecting a *pervasive* service environment as a non-layered *spatial* substrate – made up of networked SAPERE nodes – laid above the actual network infrastructure
- such substrate embeds the basic "laws of nature" (*Eco-Laws* in SAPERE terminology) that rule the activities of the system
- any individual (e.g., devices, users, software services) has an associated semantic representation inside the ecosystem called *Live Semantic Annotation* (LSA)

SAPERE II

- LSAs are semantic annotations expressing information with same expressiveness as standard frameworks like RDF, whose abstract syntax is $i : [p_1 = v_1, \ldots, p_n = v_n]$ where i is the unique (ecosystem-wide) LSA identifier, p_i is the name of a property, and v_i is the associated value (any atomic or structured data)—p_i need not be different, as some property can be multi-valued
- an LSA *pattern* P should be initially understood as an LSA which may have some variable (written inside a pair of curly brackets) in place of one (or more) values – similar to a LINDA template – and an LSA L is said to match the pattern P if there exists a substitution of variables to values that applied to P gives L—differently from LINDA, the *matching* mechanism here is semantic and *fuzzy*
- eco-laws define the basic policies to rule sorts of *virtual chemical reactions* among LSAs

SAPERE III

- an eco-law is of the kind $P_1 + \ldots + P_n \xrightarrow{r} P'_1 + \ldots + P'_m$ where: *(i)* the left-hand side (reagents) specifies patterns that should match the LSAs $L_1, \ldots, L_n$ to be extracted from the space; *(ii)* the right-hand side (products) specifies patterns of LSAs which are accordingly to be inserted back in the space; and *(iii)* rate expression r is a numerical positive value indicating the *average frequency* at which the eco-law is to be fired
- as far as *topology* is concerned, the framework imposes that an eco-law applies to LSAs belonging to the same space, and constrains products to be inserted in that space or in a *neighbouring* one (to realise space-space interaction)

Spatial Tuples

Spatial Tuples [Ricci et al., 2016] is an extension of the basic tuple-based model for distributed multi-agent system coordination, where

- tuples are (conceptually) *placed* in the physical world and possibly *move*
  - tuples have both a *location* and an *extension*
- the behaviour of *coordination primitives* may depend on the spatial properties of the coordinating agents
- the tuple space can be conceived as a virtual layer *augmenting physical reality*, provided that
  - physical reality is enriched by digital information situated in some physical position
  - visually-augmented reality is perceived by human users by means of specific devices, such as smart-glasses, head-mounted-display, or even smartphones

Spatial ReSpecT: Recap I

Goal of Spatial ReSpecT [Mariani and Omicini, 2013]

Understanding the basic mechanisms of spatial coordination is a fundamental issue for coordination models and languages in order to govern situated interaction in the *spatio-temporal fabric*

- devising out
  - the *fundamental abstractions*
  - the *basic mechanisms*
  - the *linguistic constructs*

  required to *generally* enable and promote space-aware coordination
- defining their embodiment in terms of
  - the *tuple centre* coordination medium [Omicini and Denti, 2001]
  - the ReSpecT coordination language

Spatial ReSpecT: Recap II

Requirements of spatial coordination

Spatial coordination requires

- spatial situatedness
- spatial awareness

of the coordination media; this translates in a number of *technical requirements*

Spatial ReSpecT: Recap III

Situatedness

A *space-aware coordination abstraction* should at any time be associated to an absolute positioning, both physical and virtual

In fact

- *software abstractions* may move along a *virtual space* – typically, the network – which is usually *discrete*
- whereas *physical devices* move through a *physical space*, which is mostly *continuous*

However, software abstractions may also be hosted by mobile physical devices, thus *share* their motion.

Spatial ReSpecT: Recap IV

Awareness

The position of the coordination medium should be available to the *coordination laws* it contains in order to make them capable of *reasoning about space*—so, to implement space-aware coordination laws

Also, space has to be embedded into the working cycle of the coordination medium

- a spatial event should be *generated* within a coordination medium, conceptually corresponding to *changes in space*
- then, such events should be *captured* by the coordination medium, and used to *activate* space-aware coordination laws

Spatial ReSpecT: Recap V

Tuple centres as general-purpose coordination abstractions

- technically, a tuple centre is a programmable tuple space, i.e., a tuple space whose behaviour in response to (coordination) events can be programmed so as to specify and enact any coordination policy [Omicini and Denti, 2001, Omicini, 2007]
- tuple centres can then be thought as *general-purpose coordination abstractions*, which can be suitably forged to provide specific coordination services

Spatial ReSpecT: Recap VI

Space-aware tuple centres

- the *location* of a space-aware tuple centre is obtained through the notion of current place
    - *i.e.*, the absolute position of the computational device where the coordination medium is running, or the domain name of node hosting the tuple centre
- *motion* is conceptually represented by two sorts of spatial events:
    - leaving from a starting place
    - stopping at an arrival place

  in any sort of space / place
- analogously to (coordination) operation and time events, it is possible to specify *reactions triggered by spatial events*—spatial reactions

Spatial ReSpecT: Recap VII

Space-aware coordination policies

As a result, *space-aware coordination policies* can be encapsulated in a spatial tuple centre, which

- can be programmed to *react to motion* in either a physical or a virtual space
- so as to enforce space-aware coordination policies

Summary

- agent societies are a basic brick for MAS modelling and engineering, including spatial aspects
- coordination models can be used to build agent societies
- coordination abstractions needs to include spatial concerns, so as to be capable of expressing *spatial coordination policies* ruling agent societies
- many coordination models nowadays deals with space in order to manage the complexity of systems such as pervasive intelligent systems

Focus on. . .

MAS Basic Abstractions: Recap

Agents in MAS

- agents live within an agent *society*
- agents live immersed within an agent *environment*

Basic design abstractions for MAS [Weyns et al., 2007]

agents	are the autonomous components of the systems, embodying the *designed* activities
society	is meant to model and govern the *relationships* among agents
environment	models the either virtual or physical context where the agents are situated, capturing both the effect of agent activities and the unpredictable change brought about by *non-designed* activities

Environment in MAS I

Environment as a first-class abstraction [Weyns et al., 2007]

- environment should be handled as an *explicit* part of MAS
- environment in MAS is a first-class abstraction with two roles
  - providing the surrounding conditions for agents to exist—supporting agent life and activity
  - providing an exploitable design abstraction for building MAS applications

Environment in MAS II

Middleware and infrastructure

- *environment abstractions* [Viroli et al., 2007]
    - provide agents with services useful for achieving individual and social goals
    - are supported by some underlying software *infrastructure* managing their creation and exploitation

$\rightarrow$ modelling and engineering MAS environment based on agent *middleware* and infrastructure

Environment in MAS III

Environment, middleware, and space

- environment for situated MAS means first of all space-time situatedness of agents, societies, and MAS as a whole

→ environment abstractions as provided by MAS middleware should deal with space

! coordination abstractions are middleware abstractions: most of the aforementioned coordination models (should) have an implementation provided via *coordination middleware*

? how do *actually existing* MAS middleware deal with spatial notions?

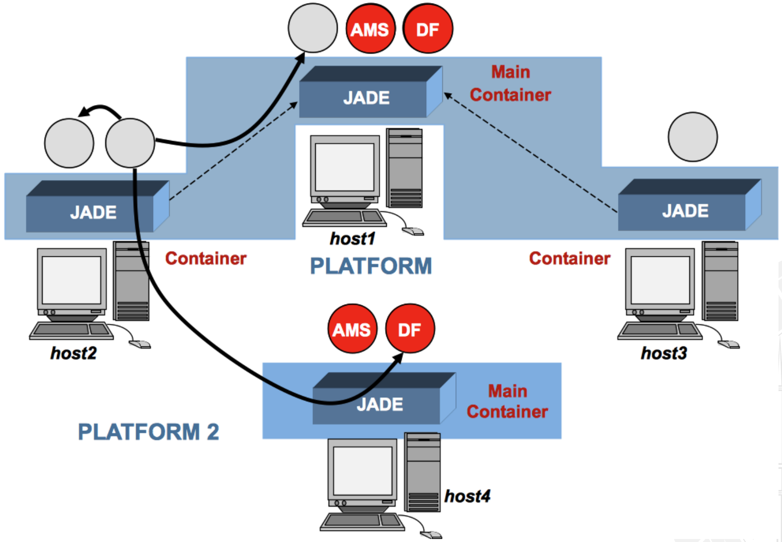
MAS Middleware and Space: Examples I

JADE [Bellifemine et al., 2007]

- JADE provides a distributed agent *platform*—where "distributed" means that a single (logical) JADE system can be split among different networked hosts
- the platform is FIPA compliant [Foundation for Intelligent Physical Agents, 2005]
- it supports intra-platform agent (strong) mobility
- it is composed by one (*main*) or more *containers*—one for each physical hosts of the platform
- each container provides a complete *runtime environment* for JADE agents execution—lifecycle management, message passing facilities, etc.

→ logical notion of locality and topology are provided via containers and platforms, with containers mapping on to physical distribution of hosts

MAS Middleware and Space: Examples II

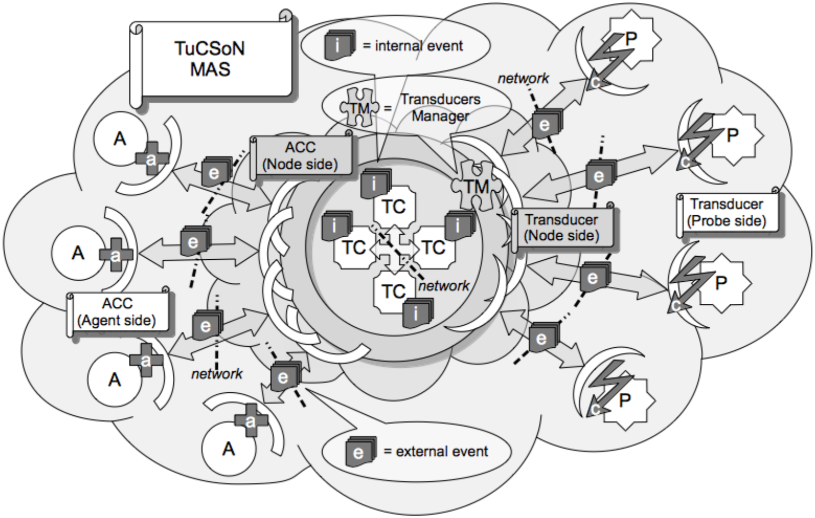


MAS Middleware and Space: Examples III

TuCSoN [Omicini and Zambonelli, 1999]

- TuCSoN exploits *spatial tuple centres* [Mariani and Omicini, 2013] as middleware abstractions for spatial coordination
- tuple centres are associated to a TuCSoN *node*, the basic brick of TuCSoN *topology*
- agent invocations of coordination primitive can be either *local* – towards the node tuple centres – or *global*—towards any tuple centre in the network
- TuCSoN agents and *environment resources* interact with MAS through boundary artefacts [Mariani and Omicini, 2015], which are
  - the architectural components representing agents as well as environmental resources within the MAS
  - in charge of associating agents and resources with *spatial information*

MAS Middleware and Space: Examples IV



MAS Middleware and Space: Examples V

CArtAgO [Omicini and Zambonelli, 1999]

- CArtAgO [Ricci et al., 2007] is a Java-based framework and infrastructure based on the A&A (agents & artefacts) meta-model [Omicini et al., 2008]
- A&A
  - introduces artefacts as the tools [Nardi, 1996] that agents use to enhance their own capabilities, for achieving their own goals [Omicini et al., 2006]
  - artefacts can be used to (computationally) represent any kind of environmental resource within a MAS in a uniform way—from sensors to actuators, from databases to legacy OO applications, from real-world objects to virtual blackboards

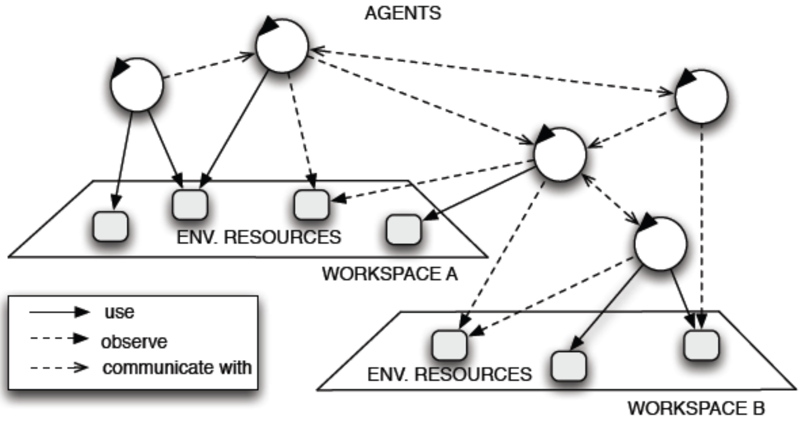
MAS Middleware and Space: Examples VI

CArtAgO architecture

CArtAgO main architectural components are

artefacts	are the basic bricks in the A&A meta-model, then the basic bricks in the CArtAgO framework, too; they work as the tools for MAS designers to properly model and implement the portion of the *environment* agents can control/should deal with
workspaces	play the role of the *topological* containers for agents and artefacts, representing the agent *working environments*: since every agent and every artefact are always associated with a workspace, workspaces can be used to define the scope of event generation/perception for agents and artefacts
agent bodies	are the architectural components enabling agent interaction with artefacts—thus, in the very end, *situatedness*, at least from the individual agent viewpoint
observable events	are defined by programmers, and are generated in response to specific operation invocations as well as observable states to monitor changes—including environment ones

MAS Middleware and Space: Examples VII



Summary

- agent environment is a basic brick for MAS modelling and engineering, including spatial aspects
- agent middleware is meant to provide agents and MAS programmers with the abstractions and tools to model and engineer complex systems with spatial features

Overall Summary

- math and logic tools to *represent*, *analyse*, and *reason* about space
- computing about space and its organisation
- different types of code mobility are possible
- spatial computing as a framework for spatial computations and systems
- MAS may provide a more general framework for spatial computing
- agent environment as a basic brick for MAS spatial aspects
- agent middleware as a source for the abstractions and tools to model and engineer complex systems with spatial features

References I

Aiello, M., Bezhanishvili, G., Bloch, I., and Goranko, V. (2012).
Logic for physical space: From antiquity to present days.
Synthese, 186(3):619–632.

Balbiani, P., Goranko, V., Kellerman, R., and Vakarelov, D. (2007).
Logical theories for fragments of elementary geometry.
In Aiello, M., Pratt-Hartmann, I., and Van Benthem, J., editors, *Handbook of Spatial Logics*, pages 343–428. Springer Netherlands, Dordrecht.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007).
A survey on context-aware systems.
International Journal of Ad Hoc and Ubiquitous Computing, 2(4):263–277.

Bandini, S., Mosca, A., and Palmonari, M. (2007).
Common-sense spatial reasoning for information correlation in pervasive computing.
Applied Artificial Intelligence, 21(4-5):405–425.

Beal, J. (2010).
A basis set of operators for space-time computations.
In *Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 91–97, Washington, DC, USA. IEEE Computer Society.

References II

Beal, J., Dulman, S., Usbeck, K., Viroli, M., and Correll, N. (2013).
Organizing the aggregate: Languages for spatial computing.
In *Formal and Practical Aspects of Domain-Specific Languages*, pages 436–501. IGI Global.

Beal, J., Michel, O., and Schultz, U. P. (2011).
Spatial computing: Distributed systems that take advantage of our geometric world.
ACM Transactions on Autonomous and Adaptive Systems, 6(2):11:1–11:3.

Beal, J., Pianini, D., and Viroli, M. (2015).
Aggregate programming for the Internet of Things.
Computer, 48(9):22–30.

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007).
Developing Multi-Agent Systems with JADE.
Wiley.

Bennett, B. (1992).
Spatial reasoning with propositional logics.
In Grouws, D. A., editor, *Principles of Knowledge Representation and Reasoning. Proceedings of the Fourth International Conference (KR '94*, volume 8, pages 51–62.
Macmillan Publishing Co, Inc.

References III

Bull, R. and Segerberg, K. (1984).
Basic modal logic.
In *Handbook of Philosophical Logic. Volume II: Extensions of Classical Logic*, volume 165 of *Synthese Library*, pages 1–88. Springer Netherlands, Dordrecht.

Byrne, R. M. and Johnson-Laird, P. (1989).
Spatial reasoning.
Journal of Memory and Language, 28(5):564–575.

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

Ciancarini, P., Omicini, A., and Zambonelli, F. (2000).
Multiagent system engineering: The coordination viewpoint.
In Jennings, N. R. and Lespérance, Y., editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *LNAI*, pages 250–259. Springer-Verlag.
6th International Workshop (ATAL'99), Orlando, FL, USA, 15–17 July 1999. Proceedings.

References IV

Clements, D. H. and Battista, M. T. (1992).
Geometry and spatial reasoning.
In *Handbook of research on mathematics teaching and learning: A project of the National Council of Teachers of Mathematics*, pages 420–464. Macmillan Publishing Co, Inc, New York, NY, England.

Corkill, D. D. (1991).
Blackboard systems.
AI Expert, 6(9):40–47.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. C. (2000).
Computational Geometry. Algorithms and Applications.
Springer.

Dutta, S. (1988).
Approximate spatial reasoning.
In *1st International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems – IEA/AIE '88*, volume 1 of *IEA/AIE '88*, pages 126–140, New York, New York, USA. ACM Press.

References V

Ferber, J. and Müller, J.-P. (1996).
Influences and reaction: A model of situated multiagent systems.
In Tokoro, M., editor, *2nd International Conference on Multi-Agent Systems (ICMAS-96)*, pages 72–79, Tokio, Japan. AAAI Press.

Foundation for Intelligent Physical Agents (2005).
FIPA home page.

Fuchs, P., Moreau, G., and Guitton, P., editors (2011).
Virtual Reality: Concepts and Technologies.
CRC Press.

Fuggetta, A., Picco, G. P., and Vigna, G. (1998a).
Understanding code mobility.
IEEE Transactions on Software Engineering, 24(5):342–361.

Fuggetta, A., Picco, G. P., and Vigna, G. (1998b).
Understanding code mobility.
IEEE Transactions on Software Engineering, 24(5):342–361.

References VI

Furht, B., editor (2011).
Handbook of Augmented Reality.
Springer New York, New York, NY.

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.

Gentner, D. (2007).
Spatial cognition in apes and humans.
Trends in Cognitive Sciences, 11(5):192–194.

Hamann, H. (2010).
Space-Time Continuous Models of Swarm Robotic Systems. Supporting Global-to-Local Programming, volume 9 of *Cognitive Systems Monographs.*
Springer Berlin Heidelberg, Berlin, Heidelberg.

References VII

Haun, D. B., Call, J., Janzen, G., and Levinson, S. C. (2006).
Evolutionary psychology of spatial representations in the hominidae.
Current Biology, 16(17):1736–1740.

Hilbert, D. (1950).
Foundations of geometry.
La Salle, Illinois, USA.

Huisman, O. and de By, R. A., editors (2009).
Principles of Geographic Information Systems. An Introductory Textbook.
ITC Educational Textbook Series. The International Institute for Geo-Information Science and Earth Observation (ITC), Enschede, The Netherlands, 3rd edition.

Kline, M. (1972).
Mathematical Thought from Ancient to Modern Times, volume 2.
Oxford University Press.

Kray, C. (2001).
The benefits of multi-agent systems in spatial reasoning.
In Russell, I. and Kolen, J., editors, *14th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2001)*, pages 552–556. AAAI Press.

References VIII

Levinson, S. C., Kita, S., Haun, D. B., and Rasch, B. H. (2002).
Returning the tables: language affects spatial reasoning.
Cognition, 84(2):155–188.

Li, P. and Gleitman, L. (2002).
Turning the tables: language and spatial reasoning.
Cognition, 83(3):265–294.

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

Mamei, M. and Zambonelli, F. (2009).
Programming pervasive and mobile computing applications: The TOTA approach.
ACM Transactions on Software Engineering and Methodology (TOSEM), 18(4):15:1–15:56.

References IX

Mariani, S. and Omicini, A. (2013).
Space-aware coordination in ReSpecT.
In Baldoni, M., Baroglio, C., Bergenti, F., and Garro, A., editors, *From Objects to Agents*, volume 1099 of *CEUR Workshop Proceedings*, pages 1–7, Turin, Italy. Sun SITE Central Europe, RWTH Aachen University.
XIV Workshop (WOA 2013). Workshop Notes.

Mariani, S. and Omicini, A. (2015).
Coordinating activities and change: An event-driven architecture for situated MAS.
Engineering Applications of Artificial Intelligence, 41:298–309.
Special Section on Agent-oriented Methods for Engineering Complex Distributed Systems.

Martelet, G. (1998).
Évolution et création, tome 1.
Editions du Cerf, Paris.

McKinsey, J. C. C. and Tarski, A. (1944).
The algebra of topology.
Annals of Mathematics, 45:141–191.

References X

Milojicic, D. S., Douglis, F., Paindaveine, Y., Wheeler, R., and Zhou, S. (2000).
Process migration.
ACM Computing Surveys, 32(3):241–299.

Moratz, R. and Wallgrün, J. O. (2003).
Spatial reasoning about relative orientation and distance for robot exploration.
In Kuhn, W., Worboys, M. F., and Timpf, S., editors, *Spatial Information Theory. Foundations of Geographic Information Science*, volume 2825 of *Lecture Notes in Computer Science*, pages 61–74. Springer Berlin Heidelberg.

Murphy, A. L., Picco, G. P., and Roman, G.-C. (2006).
LIME: A coordination model and middleware supporting mobility of hosts and agents.
ACM Transactions on Software Engineering and Methodology, 15(3):279–328.

Nardi, B. A., editor (1996).
Context and Consciousness: Activity Theory and Human-Computer Interaction.
MIT Press.

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

References XI

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

Omicini, A. (2002).
Towards a notion of agent coordination context.
In Marinescu, D. C. and Lee, C., editors, *Process Coordination and Ubiquitous Computing*, chapter 12, pages 187–200. CRC Press, Boca Raton, FL, USA.

Omicini, A. (2007).
Formal ReSpecT in the A&A perspective.
Electronic Notes in Theoretical Computer Science, 175(2):97–117.
5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.

Omicini, A. and Denti, E. (2001).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.

References XII

Omicini, A., Ricci, A., and Viroli, M. (2006).
Agens Faber: Toward a theory of artefacts for MAS.
Electronic Notes in Theoretical Computer Sciences, 150(3):21–36.
1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.

Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17(3):432–456.
Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

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

Parunak, H. V. D. (1997).
"Go to the ant": Engineering principles from natural agent systems.
Annals of Operation Research, 75(0):69–101.
Special Issue on Artificial Intelligence and Management Science.

References XIII

Pauty, J., Couderc, P., Banatre, M., and Berbers, Y. (2007).
Geo-Linda: a geometry aware distributed tuple space.
In *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, pages 370–377. IEEE.

Prandi, C., Roccetti, M., Salomoni, P., Nisi, V., and Nunes, N. J. (2016).
Fighting exclusion: a multimedia mobile app with zombies and maps as a medium for civic engagement and design.
Multimedia Tools and Applications, pages 1–29.

Ricci, A., Viroli, M., and Omicini, A. (2007).
CArtAgO: A framework for prototyping artifact-based environments in MAS.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 67–86. Springer.
3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

Ricci, A., Viroli, M., Omicini, A., Mariani, S., Croatti, A., and Pianini, D. (2016).
Spatial tuples: Augmenting physical reality with tuple spaces.
In *10th International Symposium on Intelligent Distributed Computing (IDC 2016)*.

References XIV

Rossi, D., Cabri, G., and Denti, E. (2001).
Tuple-based technologies for coordination.
In Omicini, A., Zambonelli, F., Klusch, M., and Tolksdorf, R., editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 4, pages 83–109. Springer.

Russell, B. A. W. (1956).
Essay on the Foundation of Geometry.
Dover Publications, New York, NY, USA.

Shekhar, S., Feiner, S. K., and Aref, W. G. (2016).
Spatial computing.
Communications of the ACM, 59(1):72–81.

Singer, I. M. and Thorpe, J. A. (1967).
Lecture Notes on Elementary Topology and Geometry.
Springer, New York.

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

References XV

Tanenbaum, A. S. and van Steen, M. (2007).
Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

Tarski, A. (1959).
What is elementary geometry?
In Henkin, L., Suppes, P., and Tarski, A., editors, *The axiomatic method, with special reference to geometry and physics*, pages 16–30. North-Holland, Amsterdam.

Viroli, M., Beal, J., and Usbeck, K. (2013).
Operational semantics of Proto.
Science of Computer Programming, 78(6):633–656.

Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K., and Zambonelli, F. (2007).
Infrastructures for the environment of multiagent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):49–60.

Weyns, D., Omicini, A., and Odell, J. (2007).
Environment as a first-class abstraction in multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):5–30.
Special Issue on Environments for Multi-agent Systems.

References XVI

Williams, R. K. and Sukhatme, G. S. (2012).
Probabilistic spatial mapping and curve tracking in distributed multi-agent systems.
In *2012 IEEE International Conference on Robotics and Automation (ICRA 2012)*, pages 1125–1130. IEEE.

Zambonelli, F. (2004).
Spatial computing and self-organization.
In *13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 4–8. IEEE Computer Society.

Zambonelli, F., Castelli, G., Ferrari, L., Mamei, M., Rosi, A., Di Marzo Serugendo, G., Risoldi, M., Tchao, A.-E., Dobson, S., Stevenson, G., Ye, Y., Nardini, E., Omicini, A., Montagna, S., Viroli, M., Ferscha, A., Maschek, S., and Wally, B. (2011).
Self-aware pervasive service ecosystems.
Procedia Computer Science, 7:197–199.
Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11).

References XVII

Zambonelli, F., Omicini, A., Anzengruber, B., Castelli, G., DeAngelis, F. L., Di Marzo Serugendo, G., Dobson, S., Fernandez-Marquez, J. L., Ferscha, A., Mamei, M., Mariani, S., Molesini, A., Montagna, S., Nieminen, J., Pianini, D., Risoldi, M., Rosi, A., Stevenson, G., Viroli, M., and Ye, J. (2015).
Developing pervasive multi-agent systems with nature-inspired coordination.
Pervasive and Mobile Computing, 17:236–252.
Special Issue "10 years of Pervasive Computing" In Honor of Chatschik Bisdikian.

Computing with Space

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 2016/2017