

Definitions & Goals for Distributed Systems

Distributed Systems

Andrea Omicini

<mailto:andrea.omicini@unibo.it>

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

Academic Year 2025/2026



- 1 Prologue
- 2 Definitions & Issues
- 3 Goals
- 4 Conclusion



Next in Line...

1 Prologue

2 Definitions & Issues

3 Goals

4 Conclusion



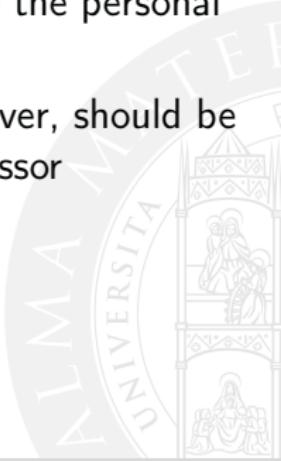
Our Goals Here

- the **goal for students**, here, is to be exposed to the **classical general view** on distributed systems that the technical and scientific community has developed in the last decades
- by providing the most common *definitions* of distributed systems
- along with the *goals* that are generally recognised to represent the target of distributed systems engineers
- also recapping some of the cases presented till now



Disclaimer

- the following slides borrow a great deal from the slides coming with the book adopted as the basic one for this course [Tanenbaum and van Steen, 2017]
- material from those slides (including pictures) has been re-used in the following, and integrated with new material according to the personal view of the professor
- every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of this course's professor



Next in Line...

1 Prologue

2 Definitions & Issues

3 Goals

4 Conclusion



Distributed System: Classic Definitions I

A distributed system can be defined as...

... a collection of *independent* computers that *appears* to its **users** as a *single coherent system* [Tanenbaum and van Steen, 2017]

User's view

- a possible definition, accounting for an *observational*, user-oriented view
- we may also call it the **computer scientist** definition of a distributed system
- from an engineering viewpoint, it is a sort of *a posteriori* definition



Distributed System: Classic Definitions II

A distributed system can be defined as...

... a collection of *autonomous* computational entities *conceived* as a *single coherent system* by its **designer**

Engineer's view

- another possible definition, accounting for a *constructive*, design-oriented view
- we may also call it the **computer engineer** definition of a distributed system
- from an engineering viewpoint, it is a sort of *a priori* definition



Distributed System: Classic Definitions III

Remarks

- a distributed system is made of a multiplicity of *components*
 - independent / autonomous computational entities (computers, microprocessors, ...)
 - ! no definition focus specifically on either computational processes or computing devices, here
 - *no assumptions* on their individual nature, structure, behaviour, ...
 - heterogeneity
- a distributed system can be seen as a single coherent system
 - according to either the user's view or the engineer's view—or both
 - need for *coherence* over multiplicity and heterogeneity



Distributed System: Another Definition

A distributed system can be defined as...

... one in which components located at *networked* computers
communicate and *coordinate* their actions only by passing
messages [Coulouris et al., 2012]

Remarks

This definition focusses on the following features of distributed systems:

- physical *distribution* of components
- the role of *interaction*—network-based communication and coordination
- uncoupling in terms of *control*

Distributed System: Working as One, Looking as One

Issues: coherence & uniformity

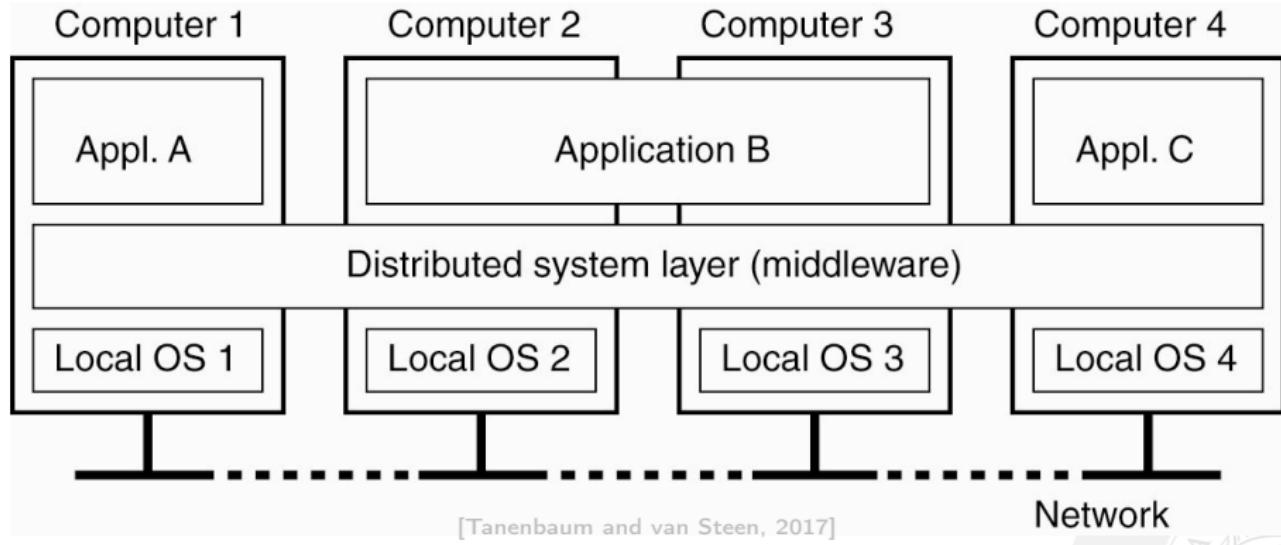
collaboration in order to achieve **coherence**, many *autonomous* entities should *collaborate*—that is, work altogether as a single (coherent) system

amalgamation in order to achieve **uniformity**, many *heterogeneous* entities should *amalgamate*—that is, look altogether as a single (uniform) system

- question: how do we achieve that in general?
- is there a general solution?



An Architectural View of Distributed Systems: Middleware



- a distributed system is organised around a **middleware**
- the *middleware layer* extends over multiple machines, and offers each application the *same interface*

An Architectural View of Distributed Systems: Old vs. New

Moving from a view of non-distributed computational systems. . .

- . . . by trying to extend the same old interpretation of systems
 - good for preserving good habits
 - bad when looking for new ideas and new problems
- ! *middleware* has obviously a role to play here



Middleware: A Principled Solution

Collaboration & amalgamation via middleware

Through *separation*, the **middleware layer**...

- ... enables *meaningful interaction* between autonomous distributed components
 - communication issues like the language for messages and its semantic interpretation
- ... hides differences in technology, structure, behaviour
 - providing both applications and components with a common shared interface—in the same way as operating systems



Why Should We Build a Distributed System?

A distributed system is *easy to build*

- hardware, software, and network components are easy to find & use
 - and to be *put together* somehow
- however, at a first sight, distribution apparently introduces *problems*, rather than solving them
 - ? why should we build a system as a distributed one?
 - ! in particular, given that it is **not** easy to make a distributed system *actually work*
 - e.g., pitfalls of distributed systems

[Tanenbaum and van Steen, 2017]



Pitfalls of Distributed Systems

False assumptions made by first time developer (by L. Peter Deutsch)

- the network is reliable
- the network is secure
- the network is homogeneous
- the topology does not change
- latency is zero
- bandwidth is infinite
- transport cost is zero
- there is one administrator

Such (false) assumptions typically produces all mistakes in the engineering of distributed systems

Pitfalls of Distributed Systems: Remarks

Such (false) assumptions . . .

. . . relate to properties that unique to distributed systems:

- reliability of the network
- security of the network
- heterogeneity of the network
- topology of the network
- latency
- bandwidth
- transport costs
- administrative domains

When engineering non-distributed systems, such problems are likely not to show up

Next in Line...

- 1 Prologue
- 2 Definitions & Issues
- 3 Goals
- 4 Conclusion



Making Distributed System Worth the Effort

Four goals for a distributed system...

- making (distributed, remote) *resources available* for use
- allowing *distribution* of resources to be **hidden** whenever unnecessary
- promoting **openness**
- promoting **scalability**

... plus, a fifth goal

- promoting **situatedness**



Focus on...

1 Prologue

2 Definitions & Issues

3 Goals

- Resource Availability

- Transparency
- Openness
- Scalability
- Situatedness

4 Conclusion



Making Resources Available

Resources are physically distributed

- a good reason to build a distributed system is to make them **distributed resources** *available* as they would belong to a single system

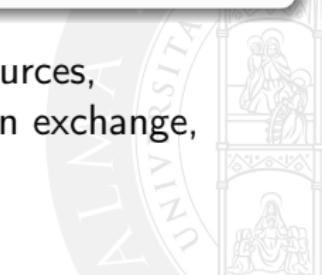
What is a resource?

Anything that ...

- ... could be in any way connected to a computational system
- ... anyone could legitimately use

E.g., printers, scanners, storage devices, distributed sensors, ...

- by making interaction possible between users and resources, distributed systems are *enablers* of sharing, information exchange, collaboration, ...
- e.g., grid systems [Foster et al., 2001]



Focus on...

1 Prologue

2 Definitions & Issues

3 Goals

- Resource Availability
- **Transparency**
- Openness
- Scalability
- Situatedness

4 Conclusion



Distribution Transparency

Physical distribution is not a feature, sometimes

- a(nother) good reason to build a distributed system is to make **physical distribution** *irrelevant* from the user's viewpoint

Transparency

- hiding non-relevant properties of the system's components and structure is called *transparency*
- there exists a number of different and useful sorts of transparency, according to the property hidden to the user's perception

By hiding non-relevant properties to users, distributed systems provide users with a *higher level of abstraction*



Types of Transparency in a Distributed System

access hide differences in data *representation* and in the access to resources

location hide the *location* of a resource

migration hide the *change of location* of a resource

relocation hide the *motion* of a resource

replication hide that a resource is *replicated*

concurrency hide the *sharing* of a resource by multiple users

failure hide the *failure* and subsequent *recovery* of a resource

Access Transparency

Heterogeneity in representation and use

- different *data representation*
- different *component structure*
- different *resource usage interface / protocols*

All of them should be hidden from the user's view, whenever possible & non-relevant

Providing a homogeneous view over heterogeneity

- a distributed system should hide heterogeneity, by providing uniform/homogeneous access to data, components, resources

Location Transparency

Location of users and resources

- often, the physical location of a resource is not relevant for its use by the users—and, viceversa, the location of users is often irrelevant for the resource
 - e.g., the position of a storage facility, or of a single printer in a cluster of printers in a lab

Hiding physical distribution of users and resources

- distributed systems should hide physical distribution, whenever possible & non-relevant

Naming is essential

- there should exist a system of logical *identifiers*, not bound to physical location
 - e.g., URLs

Migration Transparency

Resources might be *mobile*

- locations change within a distributed system
- which has to maintain its coherence anyway

A distributed system should allow *migration* of resources

- without losing coherence
- without losing functionality

Also *users* might move

- some years ago this was a sort of secondary aspect
- nowadays, user's mobility is typically a core issue in the modelling and engineering of artificial systems

Relocation Transparency

Resources should be still accessible when moving

migration should not prevent users to access resources, *while they are changing their location*

relocation transparency is in some sense a specialised, *on line* version of migration transparency

A distributed system could be useful to *allow access to mobile resources by mobile users*, by hiding changes in location (migration transparency), even while changes are actually occurring (relocation transparency)

Replication Transparency I

Sometimes *replication* helps

Like, for instance,

- in providing a *local copy* of a *resource* to local agents/users—so as to
 - reduce bandwidth consumption
 - provide faster access
- in promoting *tolerance to failures*
 - through *redundancy*



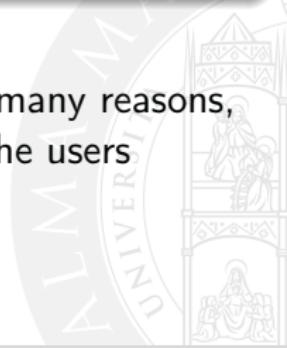
Replication Transparency II

Whatever the motivation behind replication ...

... replication is not something a user should worry about

- all replicas should be accessible in the same, *transparent* way
- so they should have the *same name*
- and should be essentially in the *same state*, so to be *apparently one* and the same thing for each and every user

A distributed system could exploit *replication* techniques for many reasons, but should be able at the same time to make it *invisible* to the users



Concurrency Transparency I

Activity in a distributed systems involves *independent* entities

- users and resources are *distributed*, and work *autonomously*, in a *concurrent* way
- for instance, two users may try to exploit the same resource at the same time
- typically, no user need to be bothered with these facts—like, “another user is accessing the same database you are accessing just now”, who cares?



Concurrency Transparency II

Concurrency in activities should be *hidden* to users

- while shared access to resources could be done cooperatively, it is often the case that users should access competitively to resources
- a distributed system could take care of this, by defining access policies governing concurrent sharing of resources
- possibly, transparently to the users



Concurrency & Consistency

The problem of *consistency*

- when many users access the same resource concurrently, *consistency* of its state is in jeopardy—but should be ensured anyway
- a distributed system should take care of ensuring resource safety even under concurrent accesses
- as usual, transparently to the users

A distributed system should take care of allowing transparent concurrent access to resources, while ensuring consistency of resources



Failure in Distributed Systems

Failure in a distributed system is any failure *anywhere* in the system

- “*You know you have [a distributed system] when the crash of a computer you've never heard of stops you from getting any work done.*” (L. Lamport)
- distribution might be either a source of problems or a blessing
- it means that a failure could occur anywhere, but also that a part of the system is likely keep on working
 - *distributed failure* is hard to control
 - *partial failure* is possible, and much better than *total failure* of centralised systems

Distribution should work as a *feature*



Failure Transparency

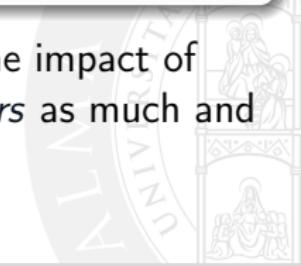
What does this mean?

- masking failures under realistic assumptions
- hiding failure of resources to users

Being failure transparent is a hard problem

- e.g., the problem of *latency*
 - how do we distinguish between a dead resource and a very slow one?
 - is “silence” from a resource originated by slowness, deliberate choice, resource failure, or network failure?

A distributed system should exploit distribution to *reduce the impact of partial failure* onto the overall system, *hiding failures to users* as much and often as possible.



Degree of Transparency in a Distributed System I

Hiding distribution is not always the best idea

- for instance, users may move and be subject to different time zones—this could lead to funny situations, if hidden, such as a download ending before it started
- also, you may appreciate to know where the file server is located (US, Japan, Europe?) when choosing from where you will download the new Ultra HD Pokémon movie from home
 - unless it is replicated in such a way that it does not matter



Degree of Transparency in a Distributed System II

Trade-off between transparency and information

- it typically concerns performance—yet, it is not limited to this
- location-awareness is often a feature
- every engineer should find out the precise *degree of transparency* its distributed system should feature, by taking into account other issues like performance, understandability, . . .



Focus on...

1 Prologue

2 Definitions & Issues

3 Goals

- Resource Availability
- Transparency
- **Openness**
- Scalability
- Situatedness

4 Conclusion



Openness

What is openness?

- essentially, *the property of a system to work with a number and sort of components that is not set once and for all at design time*
 - ← open systems are typically *designed to be open*
 - open systems are fundamentally *unpredictable*

Designing over unpredictability requires predictable items

- something needs to be *a priori* shared between the system and the (potential) components
- like, e.g., standard rules for services syntax and semantics, message interchange, ...

Interfaces for Open Systems

Interfaces to specify syntax for accessing services

- IDL (Interface Definition Languages) to define how *interfaces* are specified
- they capture syntax, rather than semantics—often, they do not specify the *protocol*, too



Issues for Open Systems

Non-functional issues

interoperability measures how easy / difficult is to make one component / system work with different ones based on some standard-based specifications

portability measures how much an application (or, a portion of it) can be moved to a different distributed system and keep working

extensibility measures how easy / difficult is to add new components and functionality to an existing distributed system



Focus on...

1 Prologue

2 Definitions & Issues

3 Goals

- Resource Availability
- Transparency
- Openness
- **Scalability**
- Situatedness

4 Conclusion



Scalability

World-wide scale changes everything

- often, few realistic assumptions can be done on the actual “size” of a distributed system at design time
- there, *size* might mean actual size in number of components, but also in geographical distribution

Dimensions of scalability^[Neuman, 1994]

- a system scales up when the *number* of users and resources grows
- a system scales up when the *geographical distribution* of users and resources extends
- a system scales up when it spans over a growing number of distinct *administrative domains*

Scalability is an issue whenever any dimension of a distributed system changes its order of magnitude

Scalability Problems: Scaling with Respect to Size

Examples of scalability limitations^[Tanenbaum and van Steen, 2017]

centralised services a single server for all users

centralised data a single database for all distributed components

centralised algorithms complete information is assumed to be available in
one single place

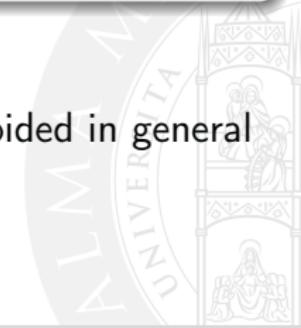


Centralisation

Making things centralised might be necessary

- even though a single server is a bottleneck, it could be a necessity in case of security problems, or, normative requirements
- even though a single collection of data is a bottleneck, it could still be needed if replication is insecure
- sometimes, the most efficient algorithm from a theoretical viewpoint might be a centralised one

However, *centralisation hinder scalability*, and should be avoided in general in distributed systems whenever possible



Decentralised vs. Centralised Algorithms I

The trouble with centralised algorithms^[Raynal, 2013]

- data should flow from the whole network to and from the place where the centralised algorithm works
 - the network would be overloaded
 - any transmission problem would cause problems to the overall algorithm
- only decentralised algorithms should be used in distributed systems



Decentralised vs. Centralised Algorithms II

Features of decentralised algorithms [Kshemkalyani and Singhal, 2011]

- no machine has complete information about the system state
- machines make decisions based only on local information
- failure of one machine does not ruin the algorithm
- there is no implicit assumption that a global clock exists



Geographical Scalability

Three basic techniques^[Neuman, 1994]

- hiding communication latency
 - asynchronous communication
- distribution
- replication



Scalability: Hiding Communication Latency I

The basic idea

Try to avoid wasting time waiting for remote responses to service requests whenever possible

Asynchronous communication

This basically means using *asynchronous communication* for requests whenever possible

- a request is sent by the application
- the application does not stop waiting for a response
- when a response come in, the application is interrupted and a handler is called to complete the request

Scalability: Hiding Communication Latency II

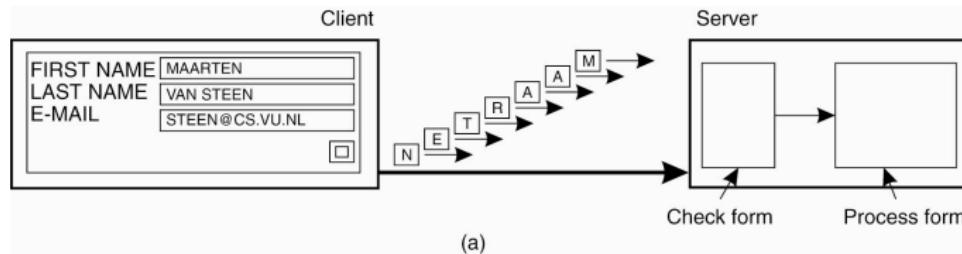
Problem

Sometimes, asynchronous communication is not feasible

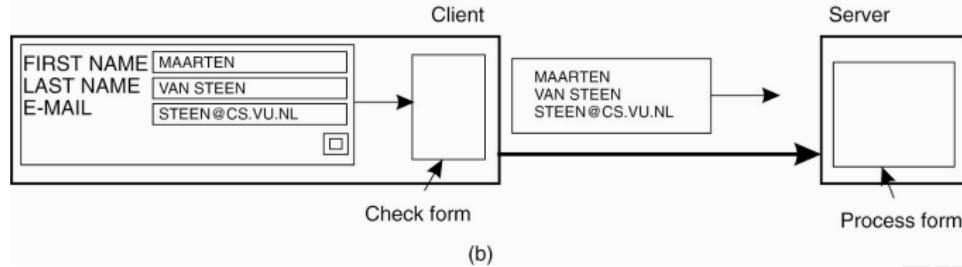
- like in Web application when a user is just waiting for the response
- alternative techniques like shipping code are needed—e.g., Javascript or Java Applets



Scalability: Code Shipping Example



(a)



(b)

The difference between letting (a) a server or (b) a client check forms as they are being filled [Tanenbaum and van Steen, 2017]

Scalability: Distribution

The basic idea

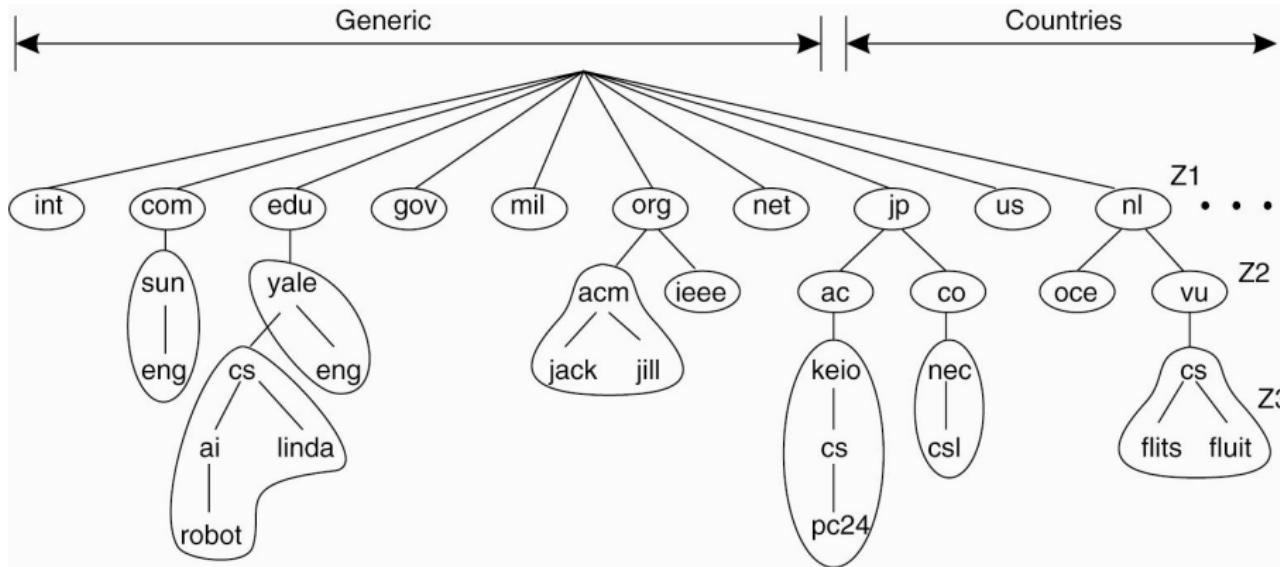
Taking a component, splitting it into parts, and spreading the parts across the system

Example: The Domain Name System (DNS)

- the DNS is hierarchically organised into a tree of *domains*
- domains are divided in non-overlapping *zones*
- the names in each zone are in charge of a single server
 - e.g., apice.unibo.it
- the *naming service* is thus distributed across several machines, without centralisation



Scalability: Distribution Example



An example of dividing the DNS space into zones

[Tanenbaum and van Steen, 2017]



Scaling Techniques: Replication

The basic idea

- when degradation of performance occurs, *replicating* components across a distributed system may increase availability and solve problems of latency
- replication typically involves making a copy of a resource from the original location to a location in the proximity of the (potential) users

Caching

- is a special form of replication
- caching is making a copy of a resource, like replication
- however, caching is a decision by the client of a resource, replication by the owner of a resource

The Problem of Consistency

Duplicating a resource introduces *consistency* problems

- involving both caching and replication
 - inconsistency is technically unavoidable, whenever copying a resource in a distributed setting
 - the point is *how much inconsistency* could a system tolerate, and how it could be hidden from users and components of a distributed system
- ! see *slides on consistency and replication*



Focus on...

1 Prologue

2 Definitions & Issues

3 Goals

- Resource Availability
- Transparency
- Openness
- Scalability
- **Situatedness**

4 Conclusion



Situatedness I

What is situatedness? [Suchman, 2007]

- ! not really a *classic distributed system* goal
- **situatedness** is in short the property of systems of being *immersed* in their environment
- that is, of being capable to (timely) *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., 2015]



Situatedness 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*—as clearly emerging from the field of *multi-agent systems* [Weyns et al., 2007]

Situatedness III

Context-awareness: space & time

- mobile computing scenarios have made clear that *situatedness* of computational systems nowadays requires at least *awareness of the spatio-temporal fabric*
- that is, any non-trivial system needs 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 a computational systems is first of all made of **space** and **time**
 - ! what we call *temporal* and *spatial context*



Situatedness IV

Context-awareness: environment

- more generally, situatedness of computational systems nowadays requires *awareness of the environment* where the systems are deployed and are expected to work
- that is, any non-trivial system needs to understand somehow its 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 V

Knowledge-intensive environments (KIE)

- special and nowadays particularly relevant sorts of working environments are the so-called **knowledge-intensive environments**
- there, *knowledge* available in large amounts, and *distributed* in the working environment, is important (if not essential) for the activities of the systems
- accessing, understanding, and possibly injecting knowledge while interacting (locally) within the working environment is essential for most computational systems to properly perform their activities and achieve their goals



Situatedness & Distributed Systems I

Why distributed systems for situatedness?

- 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 computations*
- essentially, when the systems 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, live events...



Situatedness & Distributed Systems II

Openness & scalability

- openness of distributed situated systems is essential to deal with the *unpredictability* of complex environments
- scalability allows distributed situated systems to cope with working environments of *growing complexity*



Next in Line...

- 1 Prologue
- 2 Definitions & Issues
- 3 Goals
- 4 Conclusion



Lessons Learned I

Definitions...

- there are a number of different yet converging definitions for the notions of distributed system
- distributed systems are easy to build—which also means that they are easy to be *misbuilt*
- design goals for distributed systems include
 - resource availability
 - transparency
 - openness
 - scalability
- even though nowadays it is much about robustness, reliability, fault tolerance...

Open Questions I

- what is middleware?
- are we actually ready to deal with the general notion resources, and with situatedness as well?
- what is the place of IoT, here?



Definitions & Goals for Distributed Systems

Distributed Systems

Andrea Omicini

<mailto:andrea.omicini@unibo.it>

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

Academic Year 2025/2026



References |

[Baldauf et al., 2007] 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

<http://inderscience.metapress.com/content/1184787h28163t15/>

[Coulouris et al., 2012] Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2012).

Distributed Systems. Concepts and Design.

Pearson, 5th edition

<https://www.pearson.com/en-us/subject-catalog/p/distributed-systems-concepts-and-design/P200000003160/9780132143011>

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001).

The anatomy of the grid: Enabling scalable virtual organizations.

International Journal of High Performance Computing Applications, 15(3):200–222

DOI:10.1177/109434200101500302

[Kshemkalyani and Singhal, 2011] Kshemkalyani, A. D. and Singhal, M. (2011).

Distributed Computing: Principles, Algorithms, and Systems.

Cambridge University Press

<https://www.cambridge.org/us/universitypress/subjects/engineering/communications-and-signal-processing/distributed-computing-principles-algorithms-and-systems>

[Neuman, 1994] Neuman, B. C. (1994).

Scale in distributed systems.

In Casavant, T. L. and Singhal, M., editors, *Readings in Distributed Computing Systems*, pages 463–489. IEEE

CS Press, Los Alamitos, CA, USA

(APICe)

References II

[Omicini, 2002] 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
(APICe)

[Raynal, 2013] Raynal, M. (2013).

Distributed Algorithms for Message-Passing Systems.

Springer Berlin Heidelberg, Berlin, Heidelberg

DOI:10.1007/978-3-642-38123-2

[Suchman, 2007] Suchman, L. A. (2007).

Human–Machine Reconfigurations: Plans and Situated Actions, 2nd Edition.

Cambridge University Press, 2nd edition

<http://www.cambridge.org/9780521858915>

[Tanenbaum and van Steen, 2017] Tanenbaum, A. S. and van Steen, M. (2017).

Distributed Systems. Principles and Paradigms.

Pearson Prentice Hall, 3rd edition

<https://www.distributed-systems.net/index.php/books/ds3/>

[Weyns et al., 2007] 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

(APICe) DOI:10.1007/s10458-006-0012-0



References III

- [Zambonelli et al., 2015] 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
(APICe) DOI:10.1016/j.pmcj.2014.12.002

