Evolution of Computational Systems: The Paradigm Shift

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 2013/2014

The Change is Widespread

- [Zambonelli and Parunak, 2003]
- Today software systems are essentially different from "traditional" ones
- The difference is widespread, and not limited to some application scenarios

Computer science & software engineering are going to change

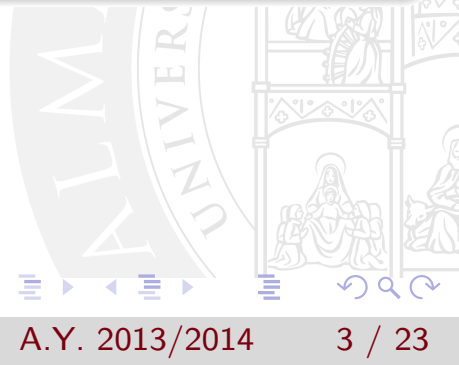
- dramatically
- complexity is too *huge* for traditional CS & SE abstractions
  - like object-oriented technologies, or component-based methodologies

The Next Crisis of Software

The Scenario of the Crisis

Computing systems

- will be anywere
  - will be embedded in every environment item/ object
- always connected
  - wireless technologies will make interconnection pervasive
- always active
  - to perform tasks on our behalf



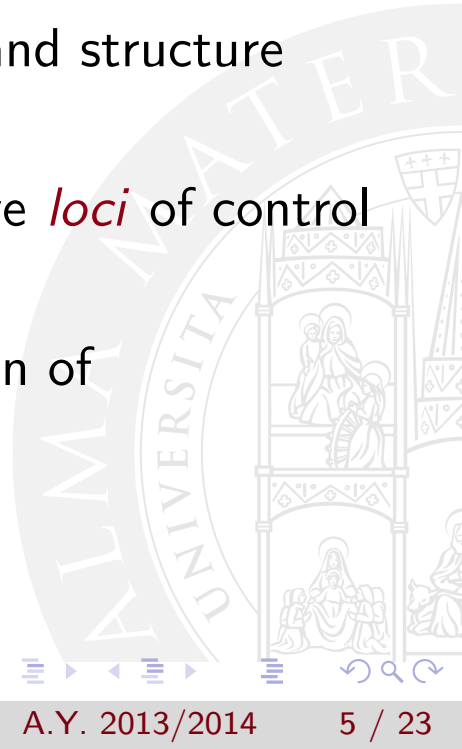
Impact on Software Engineering

Which impact on the design & development of software systems?

- Quantitative
  - in terms of computational units, software components, number of interconnections, people involved, time required, ...
  - current processes, methods and technologies do not scale
- Qualitative
  - new software systems are *different* in kind
  - new features never experimented before

Novel Features of Complex Software Systems

- Situatedness
  - computations occur within an environment
  - computations and environment mutually affect each other, and cannot be understood separately
- Openness
  - systems are permeable and subject to change in size and structure
- Locality in control
  - components of a system are autonomous and proactive *loci* of control
- Locality in interaction
  - components of a system interact based on some notion of spatio-temporal compresence on a *local* basis



Examples

Fields like

- distributed artificial intelligence
- manufacturing and environmental control systems
- mobile computing
- pervasive / ubiquitous computing
- Internet computing
- peer-to-peer (P2P) systems

have already registered the news, and are trying to account for this in technologies and methodologies

Situatedness—Examples

Control systems for physical domains

- manufacturing, traffic control, home care, health care systems
- explicitly aim at managing / capturing data from the environment through event-driven models / event-handling policies

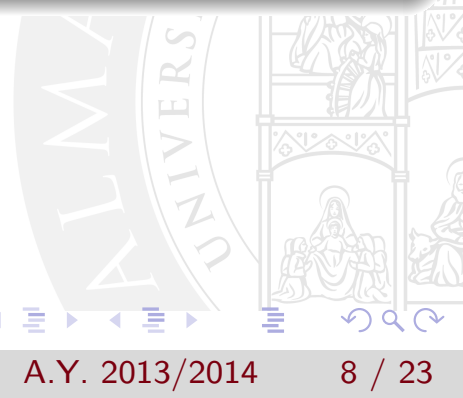
Sensor networks, robot networks

- are typically meant to sense, explore, monitor and control partially known / unknown environments

Situatedness I

Situated action [Suchman, 1987]

- the notion of *situated action* stresses the relationship between an action and its context of performance
- actions are performed in a context: which affects the actions, and is affected by them
- the notion of *environment* is what is typically used here to denote the (computational) context



Situatedness II

Environment as a first-class entity

- the notion of *environment* is explicit
- components / computations interact with, and are affected by the environment
- interaction with the environment is often explicit, too

Is this new?

- every computation always occurred in some context
- however, the environment is *masked* behind some "wrapping" abstractions
- environment is not a *primary* abstraction

Situatedness III

Does masking / wrapping work?

- wrapping abstractions are often too simple to capture complexity of the environment
- when you need to sense / control the environment, masking it is not always a good choice
- environment dynamics is typically independent of system dynamics
  - the environment is often unpredictable and non-formalisable [Wegner, 1997]

Situatedness IV

Trend in CS and SE

- drawing a line around the system
- explicitly representing
  - what is inside in terms of component's behaviour and interaction
  - what is outside in terms of environment, and system interaction with the environment
- predictability of components vs. unpredictability of the environment
  - this dichotomy is a key issue in the engineering of complex software systems

Openness—Examples

Critical control systems

- unstoppable systems, run forever
- they need to be adapted / updated anyway, in terms of either computational or physical components
- openness to change, and automatic reorganisation are essential features

Systems based on mobile devices

- the dynamics of mobile devices is out of the system / engineer's control
- system should work without assumptions on presence / activity of mobile devices
- the same holds for Internet-based / P2P systems

Openness

Permeable boundaries

- drawing lines around "systems" does not make them isolated
- boundaries are often just conventional, thus allow for mutual interaction and side-effects

The dynamics of change

- systems may change in structure, cardinality, organisation, ...
- technologies, methodologies, but above all abstractions should account for modelling (possibly governing) the dynamics of change

Openness—Further Issues

Where is the system?

- where do components belong?
- are system boundaries for real?

"Mummy, where am I?"

- how should components become aware of their environment?
- when they enter a system / are brought to existence?

How do we control open systems?

- . . . where components come and go?
- . . . where they can interact at their will?

Local Control—Examples

Cellular phone network

- each cell with its own activity / autonomous control flow
- autonomous (inter)acting in a world-wide network

World Wide Web

- each server with its own (reactive) independent control flow
- each browser client with its own (proactive) independent control flow

Local Control

Flow of Control

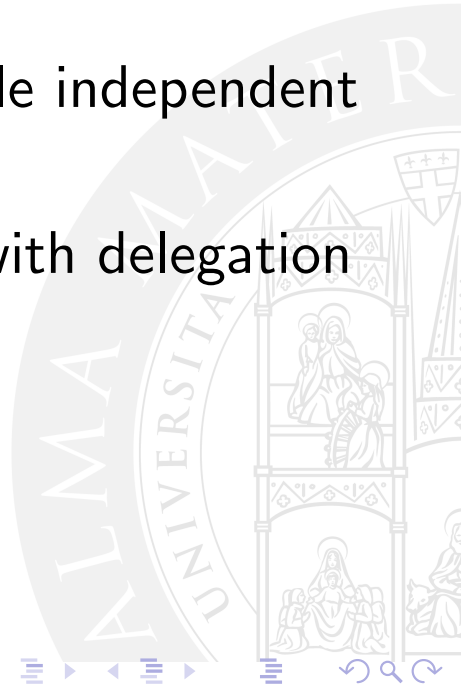
- key notion in traditional systems
- key notion in Computer Science
- multiple flows of control in concurrent / parallel computing
- however, not an immediate notion in complex software systems
  - a more general / powerful notion is required

Autonomy

- is the key notion here
- subsuming control flow / motivating multiple, independent flows of control
- at a higher level of abstraction

Local Control—Issues of Autonomy

- in an open world, autonomy of execution makes it easy for components to move across systems & environments
- autonomy of components more effectively matches dynamics of environment
- autonomy of executions is a suitable model for multiple independent computational entities
- SE principles of locality and encapsulation cope well with delegation of control to autonomous components



Local Interactions—Examples

Control systems for physical domains

- each control component is delegated a portion of the environment to control
- interactions are typically limited to the neighboring portions of the environment
- strict coordination with neighboring components is typically enforced

Mobile applications

- local interaction of mobile devices is the basis for "context-awareness"
- interactions are mostly with the surrounding environment
- interoperation with neighboring devices is typically enabled

Local Interactions

Local interactions in a global world

- autonomous components interact with the environment where they are located
  - interaction is limited in extension by either physical laws or logical constraints
- autonomous components interact openly with other systems
  - motion to and local interaction within the new system is the cheapest and most suitable model
- situatedness of autonomous components calls for context-awareness
  - a notion of locality is required to make context manageable

Summing Up

Complex software systems, then

- made of autonomous components
- locally interacting with each other
- immersed in an environment—both components and the system as a whole
- system / component boundaries are blurred—they are conceptual tools until they work

Change is going to happen soon

- Computer Science is going to change
- Software Engineering is going to change
- a paradigm shift is occurring—a *revolution*, maybe [Kuhn, 1996]

References I

Kuhn, T. S. (1996).
The Structure of Scientific Revolutions.
University of Chicago Press, 3rd edition.

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

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

References II

Zambonelli, F. and Parunak, H. V. D. (2003).
Towards a paradigm change in computer science and software engineering: A synthesis.
The Knowledge Engineering Review, 18(4):329–342.

Evolution of Computational Systems: The Paradigm Shift

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 2013/2014