# The Evolution of Computational Systems: Foundations of Agent Oriented Computing

Multiagent Systems LS
Sistemi Multiagente LS

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008

## 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

## 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

## Does masking / wrapping work?

## Situatedness II

- 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]

### Trend in CS and SE

## Situatedness III

- 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

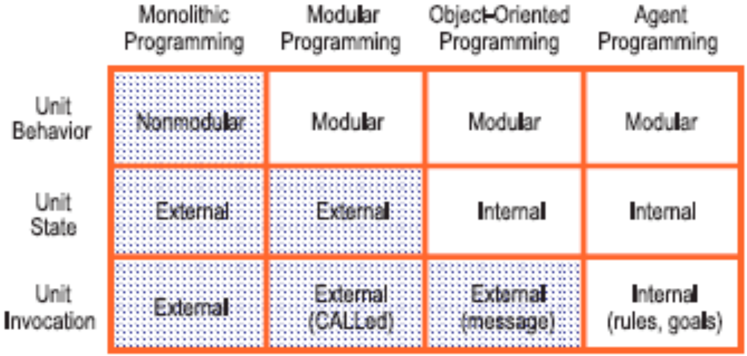
- made of autonomous components
- locally interacting with each other
- immersed in an environment—both components and the system as a whole
- system / component boundaries are blurred—they are conceptual tools until they work

### 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]

## Evolution of Programming Languages: The Picture

- [Odell, 2002]



| | Monolithic Programming | Modular Programming | Object-Oriented Programming | Agent Programming |
|---|---|---|---|---|
| Unit Behavior | Nonmodular | Modular | Modular | Modular |
| Unit State | External | External | Internal | Internal |
| Unit Invocation | External | External (CALLed) | External (message) | Internal (rules, goals) |

## Evolution of Programming Languages: Dimensions

### Historical evolution

- Monolithic programming
- Modular programming
- Object-oriented programming
- Agent programming

### Degree of modularity & encapsulation

- Unit behaviour
- Unit state
- Unit invocation

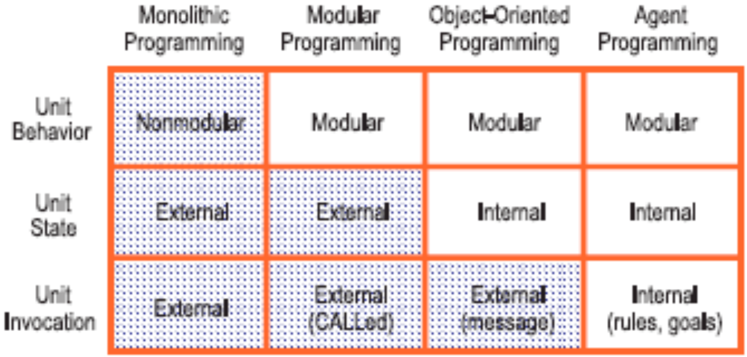
## Monolithic Programming

- The basic unit of software is the whole program
- Programmer has full control
- Program's state is responsibility of the programmer
- Program invocation determined by system's operator
- Behaviour could not be invoked as a reusable unit under different circumstances
  - modularity does not apply to unit behaviour

# Evolution of Programming Languages: The Picture

## Monolithic Programming

Encapsulation? There is no encapsulation of anything, in the very end



| | Monolithic Programming | Modular Programming | Object-Oriented Programming | Agent Programming |
|---|---|---|---|---|
| Unit Behavior | Nonmodular | Modular | Modular | Modular |
| Unit State | External | External | Internal | Internal |
| Unit Invocation | External | External (CALLed) | External (message) | Internal (rules, goals) |

## The Prime Motor of Evolution

### Motivations

- Larger memory spaces and faster processor speed allowed program to became more complex

### Results

- Some degree of organisation in the code was required to deal with the increased complexity

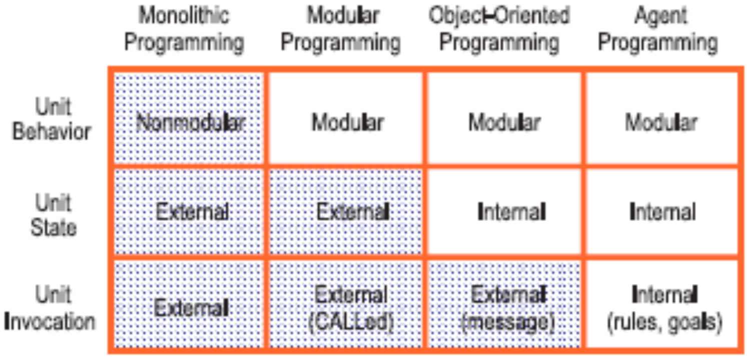
## Modular Programming

- The basic unit of software are structured loops / subroutines / procedures / ...
  - this is the era of procedures as the primary unit of decomposition
- Small units of code could actually be reused under a variety of situations
  - modularity applies to subroutine's code
- Program's state is determined by externally supplied parameters
- Program invocation determined by CALL statements and the likes

# Evolution of Programming Languages: The Picture

## Modular Programming

Encapsulation? Encapsulation applies to *unit behaviour* only



| | Monolithic Programming | Modular Programming | Object-Oriented Programming | Agent Programming |
|---|---|---|---|---|
| Unit Behavior | Nonmodular | Modular | Modular | Modular |
| Unit State | External | External | Internal | Internal |
| Unit Invocation | External | External (CALLed) | External (message) | Internal (rules, goals) |

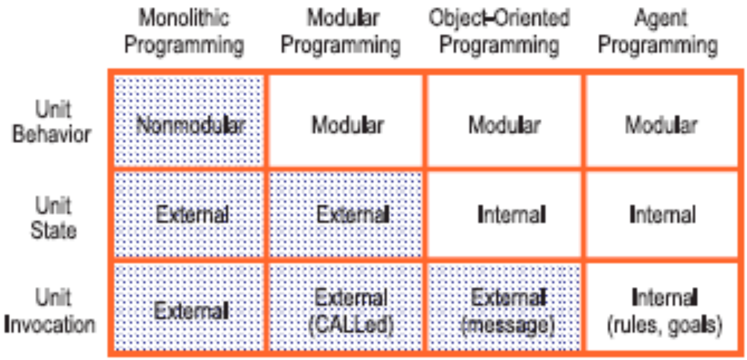
## Object-Oriented Programming

- The basic unit of software are objects & classes
- Structured units of code could actually be reused under a variety of situations
- Objects have local control over variables manipulated by their own methods
  - variable state is persistent through subsequent invocations
  - object's state is encapsulated
- Object are passive—methods are invoked by external entities
  - modularity does not apply to unit invocation
  - object's control is not encapsulated

# Evolution of Programming Languages: The Picture

## Object-Oriented Programming

Encapsulation? Encapsulation applies to unit *behaviour* & *state*



| | Monolithic Programming | Modular Programming | Object-Oriented Programming | Agent Programming |
|---|---|---|---|---|
| Unit Behavior | Nonmodular | Modular | Modular | Modular |
| Unit State | External | External | Internal | Internal |
| Unit Invocation | External | External (CALLed) | External (message) | Internal (rules, goals) |

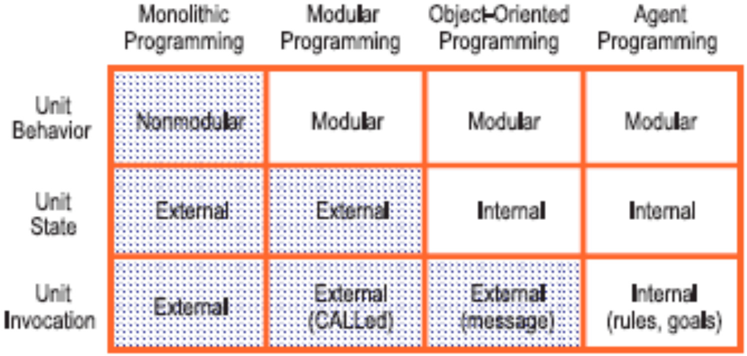
## Agent-Oriented Programming

- The basic unit of software are agents
  - encapsulating everything, in principle
    - by simply following the pattern of the evolution
  - whatever an agent is
    - we do not need to define them now, just to understand their desired features
- Agents could in principle be reused under a variety of situations
- Agents have control over their own state
- Agents are active
  - they cannot be invoked
  - agent's control is encapsulated

# Evolution of Programming Languages: The Picture

## Agent-Oriented Programming

Encapsulation? Encapsulation applies to unit *behaviour*, *state* & *invocation*



| | Monolithic Programming | Modular Programming | Object-Oriented Programming | Agent Programming |
|---|---|---|---|---|
| Unit Behavior | Nonmodular | Modular | Modular | Modular |
| Unit State | External | External | Internal | Internal |
| Unit Invocation | External | External (CALLed) | External (message) | Internal (rules, goals) |

## Features of Agents

### Before we define agents…

- …agents are *autonomous* entities
  - encapsulating their thread of control
  - they can say “Go!”
- …agents cannot be invoked
  - they can say “No!”
  - they do not have an interface, nor do they have methods
- …agents need to encapsulate a criterion for their activity
  - to self-govern their own thread of control

## Dimensions of Agent Autonomy

### Dynamic autonomy

- Agents are *dynamic* since they can exercise some degree of activity
  - they can say "Go!"
- From passive through reactive to active

### Unpredictable / non-deterministic autonomy

- Agents are *unpredictable* since they can exercise some degree of deliberation
  - they can say "Go!", they can say "No!"
  - and also because they are "opaque"—may be unpredictable to external observation, not necessarily to design
- From predictable through partially predictable to unpredictable

## Objects vs. Agents: Interaction & Control

### Message passing in object-oriented programming

- Data flow along with control
  - data flow cannot be designed as separate from control flow
- A too-rigid constraint for complex distributed systems. . .

### Message passing in agent-oriented programming

- Data flow through agents, control does not
  - data flow can be designed independently of control
- Complex distributed systems can be designed by designing information flow

## Agents Communication

### Agents communicate

- Interaction between agents is a matter of exchanging information
  - toward Agent Communication Languages (ACL)
- Agents can be involved in *conversations*
  - they can be involved in associations lasting longer than the single communication act
  - differently from objects, where one message just refer to one method

## Philosophical Differences [Odell, 2002] I

### Decentralisation

- Object-based systems are completely pre-determined in control. Control is essential centralised at design time
- Agent-oriented systems are essentially decentralised in control

### Multiple & dynamic classification

- Once created, objects typically have an unmodifiable class
- After creation, agents can change their role, task, goal, class, . . . , according to their needs and to the ever-changing structure of the surrounding environment

### Instance-level features

## Philosophical Differences [Odell, 2002] II

- Objects are class instances whose features are essentially defined by classes themselves once and for all
- Agents features can change during execution, by adaptation, learning, . . .

### Small in impact

- Loosing an object in an object-oriented system makes the whole system fail, or at least raise an exception
- Loosing an agent in a multi-agent system may lead to decreases in performance, but agents are not necessarily single points of failure

### Small in time

## Philosophical Differences [Odell, 2002] III

- Garbage collection is an extra-mechanism in object-oriented languages for taking advantage of disappearing objects
- Disappearing agents can simply be forgotten naturally, with no need of extra-mechanisms

### Small in scope

- Objects can potentially interact with the whole object space, however their interaction space is defined once and for all at design time: this defines a sort of local information space where they can retrieve knowledge from
- Agents are not omniscient and omnipotent, and typically rely on local sensing of their surrounding environment

### Emergence

## Philosophical Differences [Odell, 2002] IV

- Object-based systems are essentially predictable
- Multi-agent systems are intrinsically unpredictable and non-formalisable and typically give raise to emergent phenomena

### Analogies from nature and society

- Object-oriented systems have not an easy counterpart in nature
- Multi-agent systems closely resembles existing natural and social systems

## Towards the Coexistence of Agents and Objects

### Final issues from [Odell, 2002]

- Should we *wrap* objects to *agentify* them?
- Could we really *extend objects* to make them agents?
- How are we going to *implement the paradigm shift*, under the heavy weight of legacy?
  - technologies, methodologies, tools, human knowledge, shared practises, . . .

### Answers are to be found in the remainder of the course

- So, stay tuned!

## Towards Seamless Agent Middleware

### The first question

- How are we going to *implement the paradigm shift*, under the heavy weight of legacy?

### Mainstreaming Agent Technologies

[Omicini and Rimassa, 2004]

- Observing the state of agent technologies nowadays
- Focussing on agent middleware
- Devising out a possible scenario

## The Technology Life-Cycle

### A successful technology from conception to abandon

- First ideas from research
- Premiere technology examples
- Early adopters
- Widespread adoption
- Obsolescence
- Dismissal

### Often, however, this does not happen

- New technologies fail without even being tried for real
- Which are the factors determining whether a technology will either succeed or fail?

# Dimensions of a Technology Shift

## Technology scenario has at least three dimensions

- Programming paradigm
  - new technologies change the way in which systems are conceived
- Development process
  - new technologies change the way in which systems are developed
- Economical environment
  - new technologies change market equilibrium, and their success is affected by market situations

## 3-D space for a success / failure story

- What will determine the success / failure of agent-based technologies?

## The Programming Paradigm Dimension

### Pushing the paradigm shift

- Evangelists gain space on media
- Technological geeks follow soon
- Drawbacks
  - too much hype may create unsupported expectations
  - perceived incompatibility with existing approaches
  - possible dangers for conceptual integrity

### Middleware for the paradigm shift

- Technology support to avoid unsupported claims
- Seamlessly situated agents vs. wrapper agents
  - communication actions towards agents
  - pragmatical actions towards objects
- This allows agents to be used in conjunction with sub-systems adopting different component models

# The Development Process Dimension

## Accounting for real-world software development

- Availability of development methods & tools is critical
  - No technology is to be widely adopted without a suitable methodological support
- Day-by-day developer's needs should be accounted, too

## Agent-Oriented Software Engineering Methodologies

- Adopting agent-based metaphors and abstractions to formulate new practises in software engineering
- Current state of AOSE methodologies
  - early development phases are typically well-studied
  - later phases are not, neither the tools, nor the fine-print details

## The Economical Environment Dimension I

### Innovation has to be handled with care

- Stakeholders of new technologies may enjoy advantages of early positioning
- However, they often focus too much on *novelty* and *product*, rather than on *benefits* and *service*
  - "We are different" alone does not help much
  - software is a quite peculiar product: nearly zero marginal cost, and almost infinite production capability

## The Economical Environment Dimension II

- Promoting agent-oriented technologies through integration with existing object-oriented middleware & infrastructures
- Creating a no-cost space for agent technologies
- Notions like *coordination as a service* [Viroli and Omicini, 2006]
  - where (agent) coordination technologies are no longer "sold" as whole packages
  - whose choice do not require any design commitment
- allow agent metaphors to add their value to existing systems with no assumption on the component model

## Convergence Towards The Agent

Many areas contribute their own notion of agent

- Artificial Intelligence (AI)
- Distributed Artificial Intelligence (DAI)
- Parallel & Distributed Systems (P&D)
- Mobile Computing
- Programming Languages and Paradigms (PL)
- Software Engineering (SE)
- Robotics

# On the Notion of Intelligence in AI

## Reproducing intelligence

- AI is first of all concerned with reproducing *intelligent processes* and *behaviours*, where
  - intelligent processes roughly denote *internal* intelligence—like understanding, reasoning, representing knowledge, . . .
  - intelligent behaviours roughly represent *external*, observable intelligence—like sensing, acting, communicating, . . .

## Symbolic intelligence

- Classic AI promoted the so-called *symbolic* acceptation of (artificial) intelligence
  - based on *mental representation* of the external environment
  - where the environment is typically oversimplified
  - and the agent is the only source of disruption

## On the Notion of Agent in AI

### Encapsulating intelligence

- Agents in AI have from the very beginning worked as the units encapsulating *intelligence*
  - *individual* intelligence
  - within the symbolic interpretation of intelligence

### Cognitive agents

- AI agents are essentially cognitive agents
  - they are *first* cognitive entities
  - *then* active entities
  - in spite of their very name, coming from Latin *agens* [*agere*]—the one who acts

## AI & Agents—A Note

### Reversing perspective [Omicini and Poggi, 2006]

- Today, results from AI and MAS research are no longer so easily distinguishable
- Agents and MAS have become the introductory metaphors to most of the AI results
  - as exemplified by one of the most commonly used AI textbooks [Russell and Norvig, 2002]
- Classic AI results on planning, practical reasoning, knowledge representation, machine learning, and the like, have become the most obvious and fruitful starting points for MAS research and technologies
- It is quite rare nowadays that new findings or lines of research in AI might ignore the agent abstractions at all
- Altogether, rather than a mere subfield of AI, agents and MAS could be seen as promoting a new paradigm, providing a new and original perspective about computational intelligence and intelligent systems

## On the Notion of Agent in DAI [Wooldridge, 2002]

### Overcoming the individual dimension

- no more a single unit encapsulating individual intelligence
- and acting *alone* within an oversimplified environment

### Social acceptation of agency

- agents are individuals within a society of agents
    - agents are components of a *multiagent system* (MAS)
- agents are distributed within a *distributed environment*

## Agent Features in DAI [O'Hare and Jennings, 1996]

A DAI agent. . .

- . . . has an explicit representation of the world
- . . . is situated within its environment
- . . . solves a problem that requires intelligence
- . . . deliberates / plans its course of actions
- . . . is flexible
- . . . is adaptable
- . . . learns

## A DAI Agent Represents the World: What?

### What should be represented?

- What is relevant? What is not relevant?
  - More precisely, which knowledge about the environment is relevant for an agent to effectively plan and act?
  - So, which portion of the environment should the agent explicitly represent somehow in order to have the chance to behave intelligently?

### Representation is partial

- Necessarily, an agent has a *partial* representation of the world
- Its representation includes in general both the current *state* of the environment, and the *laws* regulating its dynamics

# A DAI Agent Represents the World: How?

## The issue of Knowledge Representation (KR)

- How should an agent represent knowledge about the world?
- Representation is not neutral with respect to the agent's model and behaviour
  - and to the engineer's possibilities as well
- Choosing the right KR language / formalism
  - according to the agent's (conceptual & computational) model
  - multisets of tuples, logic theories, description logics, . . . ?

## A DAI Agent Represents the World: Consistency I

### Perception vs. representation

- Environment changes, either by agent actions, or by its own dynamics
- Even supposing that an agent has the potential to observe all the relevant changes in the environment, it can not spend all of its activity monitoring the environment and updating its internal representation of the world
- So, in general, how could consistency of internal representation be maintained? And to what extent?
  - in other terms, how and to what extent can an agent be ensured that its knowledge about the environment is at any time consistent with its actual state

### Reactivity vs. proactivity

## A DAI Agent Represents the World: Consistency II

- An agent should be *reactive*, sensing environment changes and behaving accordingly
- An agent should be *proactive*, deliberating upon its own course of actions based on its mental representation of the world
- So, more generally, how should the duality between reactivity and proactivity be ruled / balanced?

## A DAI Agent Solves Problems

### An agent has inferential capabilities

- New data representing a new solution to a given problem
- New knowledge inferred from old data
- New methods to solve a given problem
- New laws describing a portion of the world

### An agent can change the world

- An agent is equipped with actuators that provide it with the ability to affect its environment
- The nature of actuators depends on the nature of the environment in which the agent is immersed / situated
- In any case, agent's ability to change the world is indeed limited

# A DAI Agent Deliberates & Plans

## An agent has a goal to pursue

- A *goal*, typically, as a state of the world to be reached—something to achieve
- A *task*, sometimes, as an activity to be brought to an end—something to do

## An agent understands its own capabilities

- Its capabilities in terms of actions, pre-conditions on actions, effects of actions
- "Understands" roughly means that its admissible actions and related notions are somehow represented inside an agent, and there suitably interpreted and handled by the agent
- Perception should in some way interleave with action either to check action pre-conditions, or to verify action effects

## An agent is able to build a plan of its actions

- It builds possible plans of action according to its goal/task, and to its knowledge of the environment
- It deliberates on the actual course of action to follow, then acts consequently

## A DAI Agent is Flexible & Adaptable

Define *flexible*. Define *adaptable*.

- What do these words *exactly* mean?
- Adaptable / flexible with respect to what?
- Can an agent change its goal dynamically?
- Or, can it solve different problems in different contexts, or in dynamics contexts?
- Can an agent change its strategy dynamically?
- These properties are misleading, since they are apparently intuitive, and everybody thinks he/she understands them exactly

# A DAI Agent Learns

## What is (*not*) learning?

- Learning is not merely agent's change of state
- Learning is not merely dynamic perception—even though this change the agent's state and knowledge

## What could an agent learn?

- New knowledge
- New laws of the world
- New inferential rules?
  - new ways to learn?

## A number of areas insisting on this topic

Machine Learning, Abductive / Inductive Reasoning, Data Mining, Neural Networks, . . .

## DAI Agents: Summing Up

In the overall, a DAI agent has a number of important features

- It has a (partial) representation of the world (state & laws)
- It has a limited but dynamic perception of the world
- It has inferential capabilities
- It has a limited but well-known ability to change the world
- It has a goal to pursue (or, a task to do)
- It is able to plan its course of actions, and to deliberate on what to do actually
- Once understood what this means, it might also be flexible and adaptable
- It learns, regardless of how this term is understood

## A PL Agent is Autonomous in Control

### Complexity is in the control flow

- The need is to abstract away from control
- An agent encapsulates control flow
- An agent is an independent *locus* of control
- An agent is never invoked—it merely follows / drives its own control flow
- An agent is *autonomous* in control
  - it is never invoked—it cannot be invoked

## A PL Agent is neither a Program, nor an Object

### An agent is not merely a program

- A program represents the *only* flow of control
- An agent represents a single flow of control within a multiplicity

### An agent is not merely a “grown-up” object

- An object is invoked, and simply responds
- An agent is never invoked, and can deliberate whether to respond or not to any stimulus

## A P&D Agent is mobile [Fuggetta et al., 1998]

### An agent is not bound to the Virtual Machine where it is born

- Reversing the perspective
  - it is not that agents are mobile
  - it is that objects are not
- *Mobility* is then another dimension of computing, just uncovered by agents

### A new dimension requires new abstractions

- New models, technologies, methodologies
- To be used for reliability, limitations in bandwidth, fault-tolerance, . . .

# A Robotic Agent is Physical & Situated

## A robot is a physical agent

- It has both a computational and a physical nature
  - complexity of physical world enters the agent boundaries, and cannot be confined within the environment

## A robot is intrinsically situated

- Its intelligent behaviour cannot be considered as such separately from the environment where the robot lives and acts
- Some intelligent behaviour can be achieved even without any symbolic representation of the world
  - non-symbolic approach to intelligence, or *situated action* approach [Brooks, 1991]

## A SE Agent is an Abstraction

### An agent is an abstraction for engineering systems

- It encapsulate complexity in terms of
  - information / knowledge
  - control
  - goal / task
  - intelligence?
  - mobility?
- Agent-Oriented Software Engineering (AOSE)
  - engineering computational systems using agents
  - agent-based methodologies & tools

## A MAS Agent is a Melting Pot

### Putting everything together

- The area of Multiagent Systems (MAS) draws from the results of the many different areas contributing a coherent agent notion
- The MAS area is today an independent research field & scientific community
- As obvious, MAS emphasise the *multiplicity* of the agents composing a system

### Summing up

- A MAS agent is an autonomous entity pursuing its goal / task by interacting with other agents as well as with its surrounding environment
- Its main features are
  - autonomy / proactivity
  - interactivity / reactivity / situatedness

## A MAS Agent is Autonomous

### A MAS agent is goal / task-oriented

- It encapsulates control
- Control is finalised to task / goal achievement

### A MAS agent pursues its goal / task. . .

- . . . proactively
- . . . not in response to an external stimulus

### So, what is new here?

- agents are goal / task oriented. . .
- . . . but also MAS as wholes are
- *Individual* vs. *global* goal / task
    - how to make them coexist fruitfully, without clashes?

# A MAS Agent is Interactive

## Limited perception, limited capabilities

- It depends on other agents and external resources for the achievement of its goal / task
- It needs to interact with other agents and with the environment [Agre, 1995]
  - communication actions & pragmatical actions

## A MAS agent lives not in isolation

- It lives within an agent *society*
- It lives immersed within an agent *environment*

## Key-abstractions for MAS

- agents
- society
- environment

## The Notion of Agent is Multi-faceted

### Many reliable scientific sources

- Many more or less convergent / divergent definitions
- A synthesis is currently ongoing in the MAS community

### Finally, defining the agent notion

- It is now possible. . .
- . . . but it is also insufficient, now
- to fully define MAS

### Meta-model is incomplete

- What about agent society?
- What about agent environment?

## Bibliography I

Agre, P. E. (1995).
Computational research on interaction and agency.
*Artificial Intelligence*, 72(1-2):1–52.
Special volume on computational research on interaction and agency, part 1.

Brooks, R. A. (1991).
Intelligence without representation.
*Artificial Intelligence*, 47:139–159.

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

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

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

O'Hare, G. M. and Jennings, N. R., editors (1996).
*Foundations of Distributed Artificial Intelligence*.
Sixth-Generation Computer Technology. John Wiley & Sons Ltd., hardcover edition.

# Bibliography II

Omicini, A. and Poggi, A. (2006).
Multiagent systems.
*Intelligenza Artificiale*, III(1-2):76–83.
Special Issue: The First 50 Years of Artificial Intelligence.

Omicini, A. and Rimassa, G. (2004).
Towards seamless agent middleware.
In *IEEE 13th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2004)*, pages 417–422, 2nd International Workshop "Theory and Practice of Open Computational Systems" (TAPOCS 2004), Modena, Italy. IEEE CS.
Proceedings.

Russell, S. J. and Norvig, P. (2002).
*Artificial Intelligence: A Modern Approach*.
Prentice Hall / Pearson Education International, Englewood Cliffs, NJ, USA, 2nd edition.
2nd Edition.

Viroli, M. and Omicini, A. (2006).
Coordination as a service.
*Fundamenta Informaticae*, 73(4):507–534.
Special Issue: Best papers of FOCLASA 2002.

## Bibliography III

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

Wooldridge, M. J. (2002).
*An Introduction to MultiAgent Systems*.
John Wiley & Sons Ltd., Chichester, UK.

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.

# The Evolution of Computational Systems: Foundations of Agent Oriented Computing

Multiagent Systems LS
Sistemi Multiagente LS

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008