

Agents & MAS: An Introduction

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
ALMA MATER STUDIORUM—Università di Bologna a Cesena

European Agent Systems Summer School
31 August 2009

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 1 / 130

Complex Software Systems Toward a Paradigm Change

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 2 / 130

Complex Software Systems Toward a Paradigm Change

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 3 / 130

Complex Software Systems Toward a Paradigm Change

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 4 / 130

Complex Software Systems Toward a Paradigm Change

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

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 3 / 130

Complex Software Systems Toward a Paradigm Change

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 4 / 130

Complex Software Systems Toward a Paradigm Change

The Next Crisis of Software

The Scenario of the Crisis

Computing systems

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

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 5 / 130

Complex Software Systems

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

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 5 / 130

Complex Software Systems

Andrea Omicini (Università di Bologna) Introduction to MAS EASSS 2009, 31/9/2009 6 / 130

Complex Software Systems

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 II

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 III

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 execution 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 ongoing

- Computer Science is changing
- Software Engineering is changing
- a (sort of) paradigm shift is occurring—a *revolution*, maybe? [Kuhn, 1996]

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

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 (CALed)	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 become 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

Philosophical Differences [Odell, 2002] II

Instance-level features

- 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

Philosophical Differences [Odell, 2002] IV

Emergence

- 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

Philosophical Differences [Odell, 2002] III

Small in time

- 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

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 EASSS courses

- So, stay tuned!

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 **Towards Agents**
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

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 abandonment

- 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

The Economical Environment Dimension II

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

Agent-Oriented Middleware & Infrastructures

- Promoting agent-oriented technologies through integration with existing object-oriented middleware & infrastructures
- Creating a no-cost space for agent technologies
- Notions like e.g. *coordination as a service* [Viroli and Omicini, 2006], with coordination media made available to components of any sort
 - where (agent) coordination technologies are no longer “sold” as whole packages
 - whose choice do not require any design commitment
 - where however agents represent the most effective choice for most components
- allow agent metaphors to add their value to existing systems with no assumption on the component model

Outline

Convergence Towards The Agent

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

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

On the Notion of Agent 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, ...

Encapsulating intelligence

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

Symbolic intelligence

- Classic AI promoted the so-called *symbolic* acceptance 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

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 acceptance 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: 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



A DAI Agent Represents the World: Consistency II

Reactivity vs. proactivity

- 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 I

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

A DAI Agent Deliberates & Plans II

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 both important and potentially 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 to invocations
- An agent is never invoked, and can deliberate whether to respond or not to any stimulus

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]
- *Reactive architectures* come from here

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 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

Autonomy as the Core of Agents

Lex Parsimoniae: Autonomy

- Autonomy as the essential feature of agents
- Let us see whether other typical agent features may follow / descend from this somehow

Computational Autonomy

- Agents are autonomous as they *encapsulate* (the thread of) *control*
- Control does not pass through agent boundaries
 - only data (knowledge, information) crosses agent boundaries
- Agents have no interface, cannot be controlled, nor can they be invoked
- Looking at agents, MAS can be conceived as an aggregation of multiple distinct *loci* of control interacting with each other by exchanging information

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

(Autonomous) Agents (Pro-)Act

Action as the essence of agency

- The etymology of the word *agent* is from the Latin *agens*
- So, agent means "the one who acts"
- Any coherent notion of agency should naturally come equipped with a model for agent actions

Autonomous agents are pro-active

- Agents are literally active
- Autonomous agents encapsulate control, and the rule to govern it
- Autonomous agents are pro-active by definition
 - where pro-activity means "making something happen", rather than waiting for something to happen

Agents are Situated

The model of action depends on the context

- Any “ground” model of action is strictly coupled with the context where the action takes place
- An agent comes with its own model of action
- Any agent is then strictly coupled with the environment where it lives and (inter)acts
- Agents are in this sense are intrinsically *situated*

Are Agents Reactive?

Situatedness and reactivity come hand in hand

- Any model of action is strictly coupled with the context where the action takes place
- Any action model requires an adequate *representation* of the world
- Any *effective* representation of the world requires a *suitable* balance between environment *perception* and representation
- Any effective action model requires a suitable balance between environment perception and representation
 - however, any non-trivial action model requires some form of perception of the environment—so as to check action pre-conditions, or to verify the effects of actions on the environment
- Agents in this sense are supposedly *reactive* to change

Are Autonomous Agents Reactive?

Reactivity as a (deliberate) reduction of proactivity

- An autonomous agent could be built / choose to merely react to external events
- It may just wait for something to happen, either as a permanent attitude, or as a temporary opportunistic choice
- In this sense, autonomous agents may also be reactive

Reactivity to change

- Reactivity to (environment) change is a different notion
- This mainly comes from early AI failures, and from robotics
- It stems from agency, rather than from autonomy—as discussed above

(Autonomous) Agents Change the World

Action, change & environment

- Whatever the model, any model for action brings along the notion of *change*
 - an agent acts to change something around in the MAS
- Two admissible targets for change by agent action
 - agent** an agent could act to change the state of another agent
 - since agents are autonomous, and only data flow among them, the only way another agent can change their state is by providing them with some information
 - change to other agents essentially involves *communication actions*
 - environment** an agent could act to change the state of the environment
 - change to the environment requires *pragmatical actions*
 - which could be either physical or virtual depending on the nature of the environment

Autonomous Agents are Social

From autonomy to society

- From a philosophical viewpoint, autonomy only makes sense when an individual is immersed in a society
 - autonomy does not make sense for an individual in isolation
 - no individual alone could be properly said to be autonomous
- This also straightforwardly explain why any program in any sequential programming language is not an autonomous agent *per se* [Graesser, 1996, Odell, 2002]

Autonomous agents live in a MAS

- Single-agent systems do not exist in principle
- Autonomous agents live and interact within agent societies & MAS
- Roughly speaking, MAS are the only “legitimate containers” of autonomous agents

Autonomous Agents are Interactive

Interactivity follows, too

- Since agents are subsystems of a MAS, they interact within the global system
 - by essence of systems in general, rather than of MAS
- Since agents are autonomous, only data (knowledge, information) crosses agent boundaries
- Information & knowledge is exchanged between agents
 - leading to more complex patterns than message passing between objects

Autonomous Agents Have Need a Goal, or a Task, or ...

Agents govern MAS computation

- By encapsulating control, agents are the main forces governing and pushing computation, and determining behaviour in a MAS
- Along with control, agent should then encapsulate the *criterion* for regulating the thread(s) of control

Autonomy as self-regulation

- The term "autonomy", at its very roots, means self-government, self-regulation, self-determination
 - "internal unit invocation" [Odell, 2002]
- This does *not* imply in any way that agents *needs* to have a goal, or a task, to be such—to be an agent, then
- However, this *does* imply that autonomy captures the cases of goal-oriented and task-oriented agents
 - where goals and tasks play the role of the criteria for governing control

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

"Weak" Notion of Agent

Four key qualities [Wooldridge and Jennings, 1995]

Weak agents are

- Autonomous
- Proactive
- Reactive (to change)
- Social

Are Autonomous Agents Intelligent?

Intelligence helps autonomy

- Autonomous agents have to self-determine, self-govern, ...
- Intelligence makes it easy for an agent to govern itself
- While intelligence is not mandatory for an agent to be autonomous
 - however, *intelligent autonomous agents* clearly make sense

Are Autonomous Agents Mobile?

Mobility is an extreme form of autonomy

- Autonomous agents encapsulate control
- At the end of the story, control might be independent of the environment where an agent lives—say, the virtual machine on which it runs
- *Mobile autonomous agents* clearly make sense
 - even though mobility is not required for an agent to be autonomous

Do Autonomous Agents Learn?

Learning may improve agent autonomy

- By learning, autonomous agents may acquire new skills, improve their practical reasoning, etc.
- In short, an autonomous agent could learn how to make a better use out of its autonomy
- *Learning autonomous agents* clearly make sense
 - learning, however, is not needed for an agent to be autonomous

“Strong” Notion of Agent

Mentalistic notion [Wooldridge and Jennings, 1995]

Strong agents have mental components such as

- Belief
- Desire
- Intention
- Knowledge
- ...

Intelligent agents and mental components

Intelligent autonomous agents are naturally (and quite typically) conceived as strong agents

Summing Up

Agents are *autonomous computational entities*

- Agents encapsulate control along with a criterion to govern it
- From autonomy, many other features (more or less) 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*

Intelligent agents are core components for complex systems

- Strong vs. weak notion of agency
- Mobility

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 **Building Agents & MAS**
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

Paradigm Shifts in Software Engineering

New classes of programming languages

- New classes of programming languages come from paradigm shifts in Software Engineering^a
 - new meta-models / new ontologies for artificial systems build up new spaces
 - new spaces have to be “filled” by some suitably-shaped new (class of) programming languages, incorporating a suitable and coherent set of new abstractions
- The typical procedure
 - first, existing languages are “stretched” far beyond their own limits, and become cluttered with incoherent abstractions and mechanisms
 - then, academic languages covering only some of the issues are proposed
 - finally, new well-founded languages are defined, which cover new spaces adequately and coherently

^aSE here is taken in its broadest acceptance as the science of building software system, rather than the strange “theoretically practical” discipline you find at ICSE... Otherwise, one may easily see the thing the other way round

The Problem of PL & SE Today

Things are running too fast

- New classes of programming languages emerge too fast from the needs of real-world software engineering
 - However, technologies (like programming language frameworks) require a reasonable amount of time (and resources, in general) to be suitably developed and stabilised, before they are ready for SE practise
- Most of the time, SE practitioners have to work with languages (and frameworks) they know well, but which do not support (or, incoherently / insufficiently support) required abstractions & mechanisms
- This makes methodologies more and more important with respect to technologies, since they can help covering the “abstraction gap” in technologies

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 **Building Agents & MAS**
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

The Agent Abstraction

MAS programming languages have *agent* as a fundamental abstraction

- An agent programming language should support one (or more) agent definition(s)
 - so, straightforwardly supporting mobility in case of mobile agents, intelligence somehow in case of intelligent agents, . . . , by means of well-defined language constructs
- Required agent features play a fundamental role in defining language constructs

Agent Architectures

MAS programming languages support agent *architectures*

- Agents have (essential) features, but they are built around an *agent architecture*, which defines both its internal structure, and its functioning
- An agent programming language should support one (or more) agent architecture(s)
 - e.g., the BDI (Belief, Desire, Intention) architecture [Rao and Georgeff, 1991]
 - A good introduction [Wooldridge, 2002]
- Agent architectures influence possible agent features

Agent Observable Behaviour

MAS programming languages support agent *model of action*

- Agents act
 - through either communication or pragmatical actions
- Altogether, these two sorts of action define the admissible space for agent's observable behaviour
 - a *communication language* defines how agents speak to each others
 - a "language of pragmatical actions" should define how an agent can act over its environment
- Agent programming languages should account for *both* sorts of languages
 - not so much work on languages of pragmatical actions, however

Agent Behaviour

Agent computation vs. agent interaction / coordination

- Agents have both an internal behaviour and an observable, external behaviour
 - this reproduce the "computation vs. interaction / coordination" dichotomy of standard programming languages [Wegner, 1997, Gelernter and Carriero, 1992]
- **computation** the inner functioning of a computational component
- **interaction** actions determining the observable behaviour of a computational component
 - so, what is new here?
- Agent autonomy is new
 - the observable behaviour of an agent as a computational component is *driven / governed* by the agent itself
 - e.g., intelligent agents do practical reasoning—reasoning about actions—so that computation "computes" over the interaction space—in short, agent *coordination*

Agent (Programming) Languages

Languages *to be*, languages *to interact*

- Agent programming languages should be either / both
 - *languages to be* languages to define (agent) computational behaviour
 - *languages to interact* languages to define (agent) interactive behaviour

Example: Agent Communication Languages (ACL)

- ACL are the easiest example of agent languages "to interact"
 - they just define how agents speak with each other
 - however, these languages may have some requirements on internal architecture / functioning of agents

Agents Without Agent Languages

What if we do not have an agent language available?

- For either theoretical or practical reasons, it may happen
 - we may need an essential Prolog feature, or be required to use Java
- What we do need to do: (1) *define*
 - adopt an agent definition, along with the agent's required / desired features
 - choose agent architecture accordingly, and according to the MAS needs
 - define a model and the languages for agent actions, both communicative and pragmatical
- What we do need to do: (2) *map*
 - map agent features, architecture, and action model / languages upon the existing abstractions, mechanisms & constructs of the language chosen
 - thus building an *agent abstraction layer* over our non-agent language foundation

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

Programming the Interaction Space I

The space of MAS interaction

- Languages to interact roughly define the space of (admissible) MAS interaction
- Languages to interact should not be merely seen from the viewpoint of the individual agent (*subjective viewpoint*)
- The overall view on the space of (admissible) MAS interaction is the MAS engineer's viewpoint (*objective viewpoint*)
 - *subjective vs. objective* viewpoint over interaction [Schumacher, 2001, Omicini and Ossowski, 2003]

Programming the Interaction Space II

Enabling / governing / constraining the space of MAS interaction

- A number of inter-disciplinary fields of study insist on the space of (system) interaction
 - communication
 - dialogue / argumentation
 - coordination
 - organisation
 - security
 - e-institutions / normative systems
 - ...

Coordination

Coordination in short

- Many different definitions around
 - no time to explain everything in this course—we need to simplify, here
- In short, coordination is managing / governing interaction in any possible way, from any viewpoint
- Coordination has a typical “dynamic” acceptance
 - that is, enabling / governing interaction at execution time
- Coordination in MAS is even a more chaotic field
 - again, a useful definition to harness the many different acceptations in the field is subjective vs. objective coordination—the agent's vs. the engineer's viewpoint over coordination

Organisation

Organisation in short

- Again, a not-so-clear and shared definition
- It mainly concerns the structure of a system
 - it is mostly design-driven
- It affects and determines admissible / required interactions permissions / commitments / policies / violations / fines / rewards / ...
- Organisation is still enabling & ruling the space of MAS interaction
 - but with a more “static”, structural flavour
 - such that most people mix-up “static” and “organisation” improperly
- Organisation in MAS is first of all a model of responsibilities & power
 - typically based on the notion of *role*
 - requiring a model of communicative & pragmatic actions

Security

Security in short

- You may not believe it, but also security means managing interaction
 - you *cannot* see / do / say this, you *can* say / do / see that
- Typically, security has both “static” and “dynamic” flavours
 - a design-time plus a run-time acceptance
- But tends to enforce a “negative” interpretation over interaction
 - “this is not allowed”
- In this sense, it is dual to both coordination and organisation
- So, in MAS at least, they should to be looked at altogether

Coordination, Organisation & Security

Governing interaction in MAS

- Coordination, organisation & security all mean managing (MAS) interaction
- They all are meant to shape the space of admissible MAS interactions
 - to define its admissible space at design-time (organisation/security flavour)
 - to govern its dynamics at run-time (coordination/security flavour)
- An overall view is then required
 - MAS middleware & infrastructures, normative systems, e-institutions, ... , often try to approach the overall problem of handling the MAS interaction space with a uniform approach



Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics



Methodologies

Providing usable methodologies and tools

- Industry-level AOSE methodologies are needed
- AO development *process* should be addressed, too
- The issue of *tools* is a key one



Micro- vs. Macro-level in MAS

Micro-level: individual goals

- Individual agents are built around some individual goal/task, which represent the micro-level of MAS

Macro-level: social/global goals

- Agent societies / MAS are built around some individual goal/task, which represent the macro-level of MAS

Micro-level vs. Macro-level

- Making everything work smoothly is typically the main issue in governing interaction in MAS



Technologies

Developing usable technologies

- A lot of experimental work, a number of nice agent technologies developed
- Reliable agent technologies are under development
- Integration with common technology standards



Meta-models

How do we explain MAS to the masses – ordinary programmers, I mean?

- Agents, societies, environment?
- What about agent society as a first-class entity? What about MAS environment?
- Some theoretical work ongoing
- E.g., A&A meta-model [Omicini et al., 2008], based on the notions of *artifacts* used by agents within *workspaces* relating topology to agent activity



Standards

Building a shared AO technology environment worldwide

- FIPA started a decade ago
- OMG and IEEE (FIPA committee) are taking the lead today

Applications

Sorry, a course on this is missing; but!

- Complex systems, ok
- Some papers, already [Pěchouček and Mařík, 2008]

Outline

- 1 Complex Software Systems
 - Toward a Paradigm Change
 - Away from Objects
- 2 Towards Agents
 - Moving Toward Agent Technologies
 - The Many Agents Around
- 3 Agents
 - Autonomy
 - Defining Agents
- 4 Building Agents & MAS
 - Paradigm Shifts
 - Programming Agents
 - Beyond Agents: Programming MAS
- 5 Issues in MAS
 - Current Lines of Development
 - Hot Topics

Self-*

Self-organising MAS

- Self-* techniques for MAS
- Mapping natural & social systems
- Self-* techniques for SE

Environment

Handling unpredictable environment

- Modelling environment as a first-class entity
- Environment abstractions
- From virtual environments to augmented reality

Knowledge-Intensive Systems

Agents for KIS





- Developing over distributed cognition
- within knowledge-intensive environments
- Both intelligence and mobility are features, along with self-* techniques
- From ontologies to semantic coordination

What did we say?

Motivations & Issues for Agents and MAS

- ... time for questions...
- ... thanks for listening...
- ... and enjoy EASSS 2009!!

Bibliography II





-  Graesser, S. F. A. (1996).
Is it an agent, or just a program?: A taxonomy for autonomous agents.
In Jörg P. Müller, Michael J. Wooldridge, N. R. J., editor, *Intelligent Agents III Agent Theories, Architectures, and Languages: ECAI'96 Workshop (ATAL) Budapest, Hungary, August 12–13, 1996 Proceedings*, volume 1193 of *Lecture Notes In Computer Science*, pages 21 – 35. Springer.
-  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.

Intelligent Complex Systems




Intelligent MAS

- Encapsulating intelligence
- within individual agents
- as well as within social abstractions
- also adopting self-adaptation techniques

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.
-  Gelernter, D. and Carriero, N. (1992).
Coordination languages and their significance.
Communications of the ACM, 35(2):97–107.






Bibliography III

-  Omicini, A. and Ossowski, S. (2003).
Objective versus subjective coordination in the engineering of agent systems.
In Klusch, M., Bergamaschi, S., Edwards, P., and Petta, P., editors, *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*, pages 179–202. Springer.
-  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., 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.



Bibliography IV

-  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.
-  Pěchouček, M. and Mařík, V. (2008).
Industrial deployment of multi-agent technologies: review and selected case studies.
Autonomous Agents and Multi-Agent Systems, 17(3):397–431.
Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.
-  Rao, A. S. and Georgeff, M. P. (1991).
Modeling rational agents within a BDI architecture.
In Allen, J. F., Fikes, R., and Sandewall, E., editors, *2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, San Mateo, CA. Morgan Kaufmann Publishers.

Bibliography V

-  Russell, S. J. and Norvig, P. (2002).
Artificial Intelligence: A Modern Approach.
Prentice Hall / Pearson Education International, Englewood Cliffs, NJ, USA, 2nd edition.
-  Schumacher, M. (2001).
Objective Coordination in Multi-Agent System Engineering. Design and Implementation, volume 2039 of LNCS.
Springer.
-  Viroli, M. and Omicini, A. (2006).
Coordination as a service.
Fundamenta Informaticae, 73(4):507–534.
Special Issue: Best papers of FOCLASA 2002.
-  Wegner, P. (1997).
Why interaction is more powerful than algorithms.
Communications of the ACM, 40(5):80–91.
-  Wooldridge, M. and Jennings, N. R. (1995).
Intelligent agents: Theory and practice.
Knowledge Engineering Review, 10(2):115–152.

Bibliography VI

-  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.

Agents & MAS: An Introduction

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
ALMA MATER STUDIORUM—Università di Bologna a Cesena

European Agent Systems Summer School
31 August 2009