

Towards Intelligent Agents

Frontiers of Multimedia Systems

Andrea Omicini
andrea.omicini@unibo.it

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

ATER 2013/2014

Outline of Part I. From Object to Agents

- 1 Evolution of Computational Systems
- 2 Evolution of Programming Languages
- 3 Evolution of Middleware



Outline of Part II. Agents

4 The Many Agents Around

5 Definitions

Outline of Part III: Intelligent Agents

- 6 Basics
- 7 Intentional Systems
- 8 Agents with Mental States
- 9 Intentions and Practical Reasoning
- 10 BDI Agents



Part I

From Object to Agents: The Paradigm Shift



Outline

- 1 Evolution of Computational Systems
- 2 Evolution of Programming Languages
- 3 Evolution of Middleware



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

- are / 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

Impact on Software Engineering

Which impact on the design & development of software systems?

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

Novel Features of Complex Software Systems

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

Examples

Fields like

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

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

Situatedness—Examples

Control systems for physical domains

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

Sensor networks, robot networks

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

Situatedness I

Situated action [Suchman, 1987]

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

Situatedness II

Environment as a first-class entity

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

Is this new?

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

Situatedness III

Does masking / wrapping work?

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

Situatedness IV

Trend in CS and SE

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

Openness—Examples

Critical control systems

- unstopable 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]

Outline

- 1 Evolution of Computational Systems
- 2 Evolution of Programming Languages
- 3 Evolution of Middleware



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 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] 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

Philosophical Differences [Odell, 2002] IV

Emergence

- Object-based systems are essentially predictable
- Multi-agent systems are intrinsically unpredictable and non-formalisable and typically give rise 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

Outline

- 1 Evolution of Computational Systems
- 2 Evolution of Programming Languages
- 3 Evolution of Middleware



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

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. *ontology* or *coordination as a service* [Viroli and Omicini, 2006], which are made available to components of any sort
 - where (agent) 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

Part II

Agents



Outline

4 The Many Agents Around

5 Definitions



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* 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

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 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: 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

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

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 both important and potentially misleading, since they are apparently intuitive, and everybody thinks he/she understands them exactly

A DAI Agent Learns

What is () 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]
- *Reactive architectures* come from here

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

Summing Up I

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

Summing Up II

Meta-model is incomplete

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



Outline

4 The Many Agents Around

5 Definitions



Autonomy as the Foundation of the Definition of Agent I

Lex Parsimoniae: Autonomy

- Autonomy as the only fundamental and defining feature of agents
- Let us see whether other typical agent features follow / descend from this somehow

Autonomy as the Foundation of the Definition of Agent II

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

“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
 - mobility, however, 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* required 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

Definition (Agent)

Agents are *autonomous computational entities*

genus agents are computational entities

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

Agents are

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

Part III

Intelligent Agents



Outline

- 6 Basics
- 7 Intentional Systems
- 8 Agents with Mental States
- 9 Intentions and Practical Reasoning
- 10 BDI Agents



Intelligent Agents

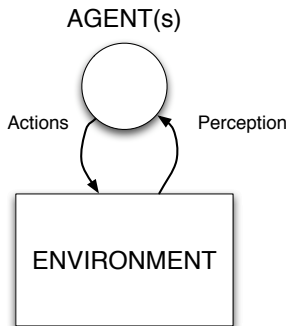
- According to a classical definition, an intelligent agent is a computational system capable of *autonomous* action and *perception* in some environment

Reminder: 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

Actions and Perception

- Actions are supposed to change the environment in order to meet agents design objectives
- Perception is a process by which the agent recognises the state of the environment, so as to be able to adapt its behaviour to it



[Russell and Norvig, 2002]

- The perception function is the agent's ability to observe its environment
 - Output of the perception function is a **percept**
 $Perception : E \rightarrow Per$
which maps environment states to percepts
- The Action function represents the agent's decision making process
 - Output of the action function is an action
 $Action : Per^* \rightarrow A_c$
which maps sequences of percepts to actions

Autonomy

- As we know, the main point about agents is they are *autonomous*: capable of acting independently, encapsulating control over their internal state.
 - Thus, an agent is a computer system capable of autonomous actions in some environment
- What's about agent's behaviour? The most typically-mentioned features are
 - Reactivity
 - Situatedness
 - Pro-activeness
 - Social ability

Reactivity

- Application in real world domains are characterised by highly dynamic conditions: situations change, information is incomplete, resources are scarce, the actions performed are not deterministic in their effects
- A *reactive* system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it—in time for the response to be useful
- *Purely reactive agents* decide what to do without reference to their history
- Reaction is entirely based on the present, with no reference to past states
 - e.g. *stimolous* $\rightarrow$ *response* rules
 - *action* : $E \rightarrow A_c$
 - A thermostat is a purely reactive agent

Situatedness

- Reactive models and state-less agency are not enough for entities engaged in dynamic environments
- Such entities continually face with external events requiring adequate services and behavioural responses
- 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 by the very actions it is capable of

Proactiveness

- We want agents able to intelligently adapt to their environment
 - autonomously (re)adapting to changes
 - recognising opportunities
 - goal-oriented behaviour
- *Proactiveness* is a generative approach
 - agents generate and attempt to achieve their objectives
 - encapsulate control, and the rule to govern it
 - not driven solely by stimuli
 - take the initiative and make something happen, rather than waiting for something to happen

Social Ability

- MAS can be conceived as an articulated world
 - where we cannot go around attempting to achieve goals without taking other entities into account
- Some goals can only be achieved with the cooperation of others
- Thus, *Social ability* in agents is the ability to interact with other agents (and possibly humans) via some kind of agent-communication means
 - Speech-Acts
 - Artifact based Interactions
 - Signals
 - Environment traces
 - ...
- *Coordination* is social: cooperation, collaboration, but also competition with others

Agents with State

- To face these growing complexities, stateful agents can be conceived
- These agents have some internal data structure, which is typically used to record information about the environment state and history
- Let I be the set of all internal states of the agent. The *perception* function is the same:

$$\text{Perception} : E \rightarrow \text{Per}$$

The action-selection function action is now defined as a mapping:

$$\text{Action} : I \rightarrow A_c$$

from internal states to actions.

- An additional function *next* is introduced, which maps an internal state and percept to an internal state: $\text{Next} : I \times \text{Per} \rightarrow I$

Implementing an Agent with Internal State

Agent Control Loop Version 1

```
1: while true
2:   observe the environment state 'e' and generates a perception(e);
3:   update internal state model:
      i ::= next(i, perception(e));
4:   select an action to execute:
      action(i);
5: end while
```

- We now have the problem to define such agent states
- How to build an effective *internal state model*?
- How to make it run? How to update it?

Outline

- 6 Basics
- 7 Intentional Systems**
- 8 Agents with Mental States
- 9 Intentions and Practical Reasoning
- 10 BDI Agents



Intentional Systems

- An idea is to refer to human attitudes as *intentional notions*
- When explaining human activity, it is often useful to make statements such as the following:
 - Felipe got rain tires because he believed it was going to rain
 - Lewis is working hard because he wants to win his first world championship
- These statements can be read in terms of *folk psychology*, by which human behaviour can be explained and can be *predicted* through the attribution of mental attitudes, such as believing and wanting (as in the above examples), hoping, fearing, and so on.

The Intentional Stance

- The philosopher – cognitive scientist – Daniel Dennett coined the term *Intentional System* to describe entities '*whose behaviour can be predicted by the method of attributing to it belief, desires and rational acumen*' [Dennett, 1971].
- Dennett identifies several grades of intentional systems:
 - ➊ A first-order intentional system has beliefs, desires, etc.
 - Felipe believes P
 - ➋ A second-order intentional system has beliefs, desires, etc. about beliefs, desires, etc. both its own and of others
 - Felipe believes that Lewis believes P
 - ➌ A third-order intentional system is then something like
 - Felipe believes that Lewis believes that Felipe believes P
 - ➍ ...

The Intentional Stance in Computing

What entities can be described in terms of intentional stance?

- Human beings are prone to provide an intentional stance to almost anything
 - sacrifices for ingratiating gods benevolence
 - animism
- 'Ascribing mental qualities like beliefs, intentions and wants to a machine is sometimes correct if done conservatively and is sometimes necessary to express what is known about its state [...] it is useful when the ascription helps us understand the structure of the machine, its past or future behaviour, or how to repair or improve it' [McCarthy, 1979]

Outline

- 6 Basics
- 7 Intentional Systems
- 8 Agents with Mental States**
- 9 Intentions and Practical Reasoning
- 10 BDI Agents



Agents as Intentional Systems

- As computer systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation
- With complexity growing, mechanistic / low level explanations become impractical
- The intentional notions can be adopted as abstraction tools, providing us with a convenient and familiar way of describing, explaining, and predicting the behaviour of complex systems
- The idea is to use the **intentional stance** as an abstraction in computing in order to explain, understand, drive the behaviour—then, crucially, program computer systems

Agents as Intentional Systems

Strong Notion of Agency

- Early agent theorists start from a (strong) notion of agents as intentional systems
- Agents were explained in terms of mental attitudes, or mental states
- In their social abilities, agents simplest consistent description implied the intentional stance
- Agents contain an explicitly-represented – *symbolic* – model of the world (written somewhere in the working memory)
- Agents make decision on what action to take in order to achieve their goals via *symbolic* reasoning

When?

Mental states are a worth abstraction for developing agents to effectively act in a class of application domains characterized by various practical limitations and requirements [Rao and Georgeff, 1995]

- At any instant of time there are many different ways in which an environment may evolve (the environment is not deterministic)
- At any instant of time there are many actions or procedures the agent may execute (the agent is not deterministic, too)
- At any instant of time the agent may want to achieve several objectives
- The actions or procedures that (best) achieve the various objectives are dependent on the state of the environment (i.e., on the particular situation, context)
- The environment can only be sensed locally
- The rate at which computations and actions can be carried out is within reasonable bounds to the rate at which the environment evolves

Goal-Oriented & Goal-Directed Systems

- There are two main families of architectures for agents with mental states

Teleo-Reactive / Goal-Oriented Agents are based on their internal design model and their control mechanism. The Goal is not figured within the internal state but it is an 'end state' for agents internal state machine

Deliberative / Goal-Directed Agents are based on a further symbolyc reasoning about Goals, which are explicitly represented and processed aside the control loop

Conceiving Agents with Mental States

Modelling agents based on **mental states**...

- ... eases the development of agents exhibiting complex behaviour
- ... provides us with a familiar, non-technical way of understanding and explaining agents
- ... allows the developer to build MAS by taking the perspective of a cognitive entity engaged in complex tasks – what would **I** do in the same situation?
- ... simplifies the construction, maintenance and verification of agent-based applications
- ... is useful when the agent has to communicate and interact with users or other system entities

Conceiving agents with mental states

- “The intentional stance is the strategy of interpreting the behaviour of an entity (person, animal, artifact, whatever) by treating it as if it were a rational agent who governed its ‘choice’ of ‘action’ by a ‘consideration’ of its ‘beliefs’ and ‘desires’ ”.
- “The scare-quotes around all these terms draw attention to the fact that some of their standard connotations may be set aside in the interests of exploiting their central features: *their role in practical reasoning*, and hence in the prediction of the behaviour of practical reasoners” [Dennett, 2007].

Agents with Mental States

Agents with Mental States

- Agents governing their behaviour on the basis of internal states that mimic cognitive mental states

Epistemic States representing agents knowledge (of their world)

- i.e., Percepts, Beliefs

Motivational States representing agents objectives

- i.e., Goals, Desires

- The process of selecting an action to execute based on the actual mental states (i.e., $action(next(i, perception(e)))$) is called **Practical Reasoning**

Outline

- 6 Basics
- 7 Intentional Systems
- 8 Agents with Mental States
- 9 Intentions and Practical Reasoning**
- 10 BDI Agents



Practical and Epistemic Reasoning

What Practical Reasoning is

- Practical reasoning is reasoning directed towards actions—the process of figuring out what to do in order to achieve what is desired
- “Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes.” [Bratman, 1987]

What Epistemic Reasoning is

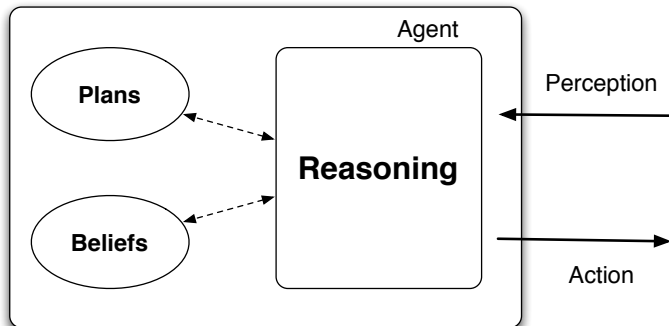
- Epistemic reasoning is reasoning directed towards knowledge—the process of updating information, replacing old information (no longer consistent with the world state) with new information

Practical Reasoning

What Practical Reasoning is

- Cognitive practical reasoning consists of two main activities
 - Deliberation** when the agent makes decision on what state of affairs the agent desire to achieve
 - Means-Ends Reasoning** when the agent makes decisions on how to achieve these state of affairs
- The output of the Deliberation phase are the *Intentions*—what agent desires to achieve, or what he desires to do
- The output of Means-Ends is in selecting a given course of actions—the workflow of actions the agent need to do to achieve the Goals

Basic Architecture of a Mentalistic Agent



Intentions and Practical Reasoning I

- ➊ Intentions pose problems for agents who need to determine ways of achieving them. If I have an intention to ϕ , you would expect me to devote resources to deciding how to bring about ϕ .
- ➋ Intentions provide a *filter* for adopting other intentions, which must not conflict. If I have an intention to ϕ , you would not expect me to adopt an intention ψ such that ϕ and ψ are mutually exclusive.
- ➌ Agents track the success of their intentions, and are inclined to try again if their attempts fail. If an agent's first attempt to achieve ϕ fails, then all other things being equal, it will try an alternative plan to achieve ϕ .
- ➍ Agents believe their intentions are possible. That is, they believe that there is at least some way that the intentions could be brought about.

Intentions and Practical Reasoning II

- 5 Agents do not believe they will not bring about their intentions. It would not be rational for me to adopt an intention to ϕ if I believed ϕ was not possible.
- 6 Under certain circumstances, agents believe they will bring about their intentions. It would not normally be rational of me to believe that I would bring my intentions about; intentions can fail. Moreover, it does not make sense that if I believe ϕ is inevitable that I would adopt it as an intention.
- 7 Agents need not intend all the expected side effects of their intentions. If I believe $\phi \rightarrow \psi$ and I intend that ϕ , I do not necessarily intend ψ also. (Intentions are not closed under implication.)
 - This last problem is known as the side effect or package deal problem. I may believe that going to the dentist involves pain, and I may also intend to go to the dentist – but this does not imply that I intend to suffer pain!

Intentions vs. Desires

- The adoption of an intention follows the rise of a given Desire (i.e. follows the adoption of a given Goal);
- Desires and Intentions are different concepts:
 - “My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires [...] before it is settled what I will do”.
 - “In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions.” [Bratman, 1990]

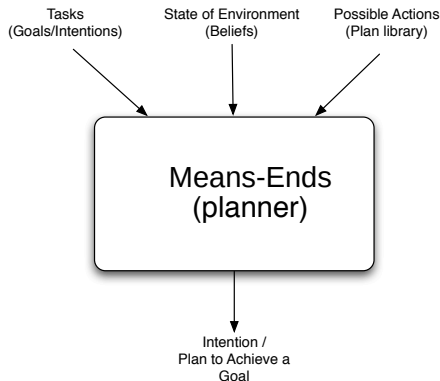
What Means-Ends Reasoning is

Means-Ends Reasoning

- The basic idea is to provide agents with three kinds of Representations:
 - Representation of Goal/Intention to achieve
 - Representation of Actions – Plans – in repertoire
 - Representation of Environment
- Given the environmental conditions, means-ends reasoning will find a Plan to achieve the adopted Goal

Mean-Ends Reasoning

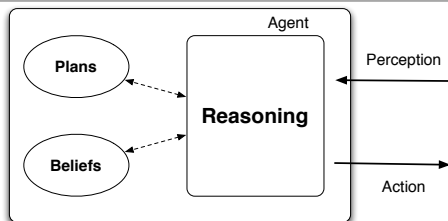
- The selected intention is an emergent property, reified at runtime by selecting a given plan for achieving a given goal.



Implementing a Practical Reasoning Agent

Agent Control Loop Version 2

```
1: while true
2:   observe the world;
3:   update internal world model (beliefs);
4:   deliberate which intention to adopt next;
5:   use means-ends reasoning to get a plan for the given intention;
6:   execute the plan;
7: end while;
```



Outline

- 6 Basics
- 7 Intentional Systems
- 8 Agents with Mental States
- 9 Intentions and Practical Reasoning
- 10 BDI Agents**



Implementing a Pratical Reasoning Agent

- Problem: Agents have Bounded Resources (bounded rationality)
 - Deliberation and Means-Ends processes are not for free: they have computational costs
 - The time taken to reason and the time taken to act are potentially unbounded, thereby destroying the reactivity and the promptness that is essential to agent survival (fitness)
- Suppose the agent
 - starts deliberating at t_0
 - begins means-ends at t_1
 - begins executing a plan at t_2
 - ends executing a plan at t_3
- Time for Deliberation is
$$t_{deliberation} = t_1 - t_0$$
- Time for Means-Ends reasoning is
$$t_{meansend} = t_2 - t_1$$

- Agents environments are supposed to be highly dynamic
 - Many concurrent changes may occur during agent decision-making as well as during the execution of plans
- The deliberated intention is worth when it is calculated—say, at t_0
 - At time t_1 , the agent will select a goal/intention that would have been optimal if it had been achieved at t_0
- The agent runs the risk that the intention selected is no longer optimal – or no longer achievable – by the time the agent has committed to it

So, this agent will have overall optimal behaviour in the following circumstances:

- ① When deliberation and means-ends reasoning take a vanishingly small amount of time
- ② When the world is guaranteed to remain (essentially) static while the agent is deliberating and performing means-ends reasoning, so that the assumptions upon which the choice of intention to achieve and plan to achieve the intention remain valid until the agent has completed both deliberation and means-ends reasoning
- ③ When an intention that is optimal when achieved at t_0 (the time at which the world is observed) is guaranteed to remain optimal until t_2 (the time at which the agent has found a course of action to achieve the intention)

BDI Agents

The abstract architecture proposed by [Rao and Georgeff, 1992] comprise three dynamic and global structures representing agent's *Beliefs*, *Desires* and *Intentions* (BDI), along with an input queue of events.

- Update (write) and Query (read) operations are possible upon the three structures
 - Update operation are subject to compatibility requirements
 - Formalised constraints hold upon the mental attitudes
- The events that the system is able to recognise could be either external (i.e., coming from the environment) or internal ones (i.e., coming from some reflexive action)
 - Events are assumed to be atomic and can be recognized after they have occurred

Implementing a BDI Agent

Agent Control Loop Version 3 [Rao and Georgeff, 1995]

```
1. initialize-state();
2. while true do
3.     options := option-generator(event-queue);
4.     selected-options := deliberate(options);
5.     update-intentions(selected-options);
6.     execute();
7.     get-new-external-events();
8.     drop-successful-attitudes();
9.     drop-impossible-attitudes();
10. end-while
```

- ➊ The agent initializes the internal states
- ➋ The agent enters the main loop
- ➌ The option generator reads the event queue and returns a list of options
- ➍ The deliberator selects a subset of options to be adopted and adds these to the intention structure
- ➎ The intentions to be adopted are filtered from the selected ones
- ➏ If there is an intention to perform an atomic action at this point in time the agent executes it
- ➐ Any external events that have occurred during the interpreter cycle are then added to the event queue (the same for internal events)
- ➑ The agent modifies the intention and the desire structures by dropping successful ones
- ➒ Finally, impossible desires and intentions are dropped too

More formally:

Agent Control Loop Version 4

1. $B := B_0$;
2. $I := I_0$;
3. *while true do*
4. *get new percept* Γ ;
5. $B := brf(\Gamma, B)$;
6. $I := deliberate(B)$;
8. $\pi := plan(B, I)$;
9. *execute*(π);
10. *end-while*

How does a BDI Agent Deliberate?

Till now, we gave no indication on how reasoning procedures of deliberation and option generation can be made sufficiently fast to satisfy the real time demands placed upon the cognitive system.

Deliberation can be decomposed in two phases:

Option Generation understand what are the available alternatives

Deliberation choose (and filter) between the adoptable goals/intentions

Chosen options are then intentions, so the agents commit to the selected ones—and executes them.

Refining Deliberation Function

Option Generation the agent generates a set of possible alternatives; represents option generation via a function, options, which takes agent's current beliefs and current intentions, and from them determines a set of options (= desires).

Deliberation the agent chooses between *competing* alternatives, and commits to the intention to achieving them. In order to select between competing options, an agent uses a *filter* function.

- the strategy for deliberating between goals typically is in the hands of the agent developer
- most BDI programming platforms provide mechanisms to describe under which conditions some goal should inhibit the others (Goal Formulae)
- typically, such Goal Formulae are first-order logic predicates indicating contexts and trigger conditions
- Game Theory can enter the picture: i.e., Maximizing 'Expected Utilities'

BDI Agent Control Loop

Agent Control Loop Version 5

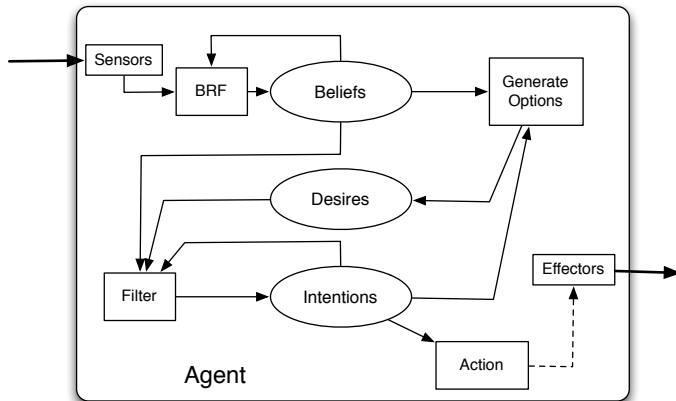
1. $B := B_0$;
2. $I := I_0$;
3. *while true do*
4. *get new percept* Γ ;
5. $B := brf(\Gamma, B)$;
6. $D := options(B, I)$;
7. $I := filter(B, D, I)$;
8. $\pi := plan(B, I)$;
9. *execute*(π);
10. *end-while*

Structure of BDI Systems

BDI architectures are based on the following constructs

- A set of Beliefs
- A set of Desires (or Goals)
- A set of Intentions
 - or better, a subset of the goals with an associated stack of plans for achieving them. These are the intended actions
- A set of Internal Events
 - elicited by a belief change (i.e., updates, addition, deletion) or by goal events (i.e. a goal achievement, or a new goal adoption)
- A set of External Events
 - perceptive events coming from the interaction with external entities (i.e. message arrival, signals, etc.)
- A Plan library (repertoire of actions) as a further (static) component

Basic Architecture of a BDI Agent [Wooldridge, 2002]



Post-Declarative Systems

Someone said that this approach leads to a kind of *post-declarative* programming

- In procedural programming, we say exactly what a system should do
- In declarative programming, we state something that we want to achieve, give the system general info about the relationships between objects, and let a built-in control mechanism (e.g., goal-directed theorem proving) figure out what to do
- with intentional agents, we give a very abstract specification of the system, and let the control mechanism figure out what to do, knowing that it will act in accordance with the built-in theory of agency

Actually, the BDI framework combines in an excellent way both (post)declarative structure and procedural knowledge in terms of plans.

Beliefs

Beliefs

Agents knowledge is structured in Beliefs about the current state of the world

- they are informational units, typically implemented as ground sets of literals, possibly with no disjunctions or implications
- they should reflect only the information which is currently held (i.e. situated)
- they are expected to change in the future, i.e., as well as the environment changes

Plans

Plans

Plans represent the means the agent has to change the world, and to bring it closer to his desires

- they are language constructs, typically implemented in the form of procedural structures
- plans have a 'body', describing the workflow of activities (actions) that have to be executed for plan execution to be successful
- the conditions under which a plan can be chosen as an option are specified in an *invocation condition* (triggering event) and a *pre- or context- condition* (situation that must hold for the plan to be executable)

Intentions

Intentions

Intentions are emergent properties reified at runtime by selecting a given plan for achieving a given goal

- Represented 'on-line' using a run-time stack of hierarchically plans related to the ongoing adopted goals
- Similarly to how Prolog interpreter handle clauses
- Multiple intention stacks can coexist, either running in parallel, suspended until some condition occurs, or ordered for execution in some way

BDI Agents Programming Platforms

JASON (Brasil) <http://jason.sourceforge.net/> Agent platform and language for BDI agents based on AgentSpeak(L)

JADEX (Germany) <http://jadex.informatik.uni-hamburg.de/> Agent platform for BDI and Goal-Directed Agents

2APL (Netherlands) <http://www.cs.uu.nl/2apl/> Agent platform providing programming constructs to implement cognitive agents based on the BDI architecture

3APL (Netherlands) <http://www.cs.uu.nl/3apl/> A programming language for implementing cognitive agents

PRACTIONIST (Italy) <http://www.practionist.org/> Framework built on the Bratman's theory of practical reasoning to support the development of BDI agents

Part IV

Bibliography

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.



Bratman, M. E. (1987).

Intention, Plans, and Practical Reason.

Harvard University Press.



Bratman, M. E. (1990).

What is intention?

In Cohen, P. R., Morgan, J. L., and Pollack, M. E., editors, *Intentions in Communication*, pages 15–32. The MIT Press, Cambridge, MA.

Bibliography II



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



Dennett, D. (1971).
Intentional systems.
Journal of Philosophy, 68:87–106.



Dennett, D. (2007).
Intentional systems theory.
In Beckermann, A. and Walter, S., editors, *Oxford Handbook of the Philosophy of Mind*, chapter 19. Oxford University Press.



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

Bibliography III

-  Kuhn, T. S. (1996).
The Structure of Scientific Revolutions.
University of Chicago Press, 3rd edition.
-  McCarthy, J. (1979).
Ascribing mental qualities to machines.
In Ringle, M., editor, *Philosophical Perspectives in Artificial Intelligence*, Harvester Studies in Cognitive Science, pages 161–195.
Harvester Press, Brighton.
-  Odell, J. (2002).
Objects and agents compared.
Journal of Object Technologies, 1(1):41–53.

Bibliography IV



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.



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.

Bibliography V

Proceedings.



Rao, A. S. and Georgeff, M. P. (1992).

An abstract architecture for rational agents.

In Nebel, B., Rich, C., and Swartout, W. R., editors, *3rd International Conference on Principles of Knowledge Representation and Reasoning (KR '92)*, pages 439–449, Cambridge, MA, USA. Morgan Kaufmann. Proceedings.



Rao, A. S. and Georgeff, M. P. (1995).

BDI agents: From theory to practice.

In Lesser, V. R. and Gasser, L., editors, *1st International Conference on Multi Agent Systems (ICMAS 1995)*, pages 312–319, San Francisco, CA, USA. The MIT Press.

Bibliography VI



Russell, S. J. and Norvig, P. (2002).

Artificial Intelligence: A Modern Approach.

Prentice Hall / Pearson Education International, Englewood Cliffs, NJ, USA, 2nd edition.



Suchman, L. A. (1987).

Plans and Situated Actions: The Problem of Human-Machine Communication.

Cambridge University Press, New York, NYU, USA.



Viroli, M. and Omicini, A. (2006).

Coordination as a service.

Fundamenta Informaticae, 73(4):507–534.

Special Issue: Best papers of FOCLASA 2002.

Bibliography VII



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.



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.

Towards Intelligent Agents

Frontiers of Multimedia Systems

Andrea Omicini
andrea.omicini@unibo.it

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

ATER 2013/2014