Agents & Artifacts
A Meta-Model for Agent-Oriented Computing

Andrea Omicini
andrea.omicini@unibo.it

Alma Mater Studiorum—Università di Bologna a Cesena

WOA 2007 School

Next in Line. . .

The Change is Widespread

- [Zambonelli and Parunak, 2003]

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

Computer science & software engineering are going to change

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

The Next Crisis of Software

The Scenario of the Crisis

Computing systems

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

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

Complex software systems. . .

. . . are 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—boundaries are useful 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]

Next in Line. . .

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

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 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 name you like to use – agent, computee, shrumpf –
    - agents are the last step in the encapsulation process
- Agents could in principle be reused under a variety of situations
- Agents have control over their own state
- Agents are active
  - they cannot – are not to – be invoked
  - agent's control is encapsulated

Evolution of Programming Languages: The Picture

Agent-Oriented Programming

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

Whatever we think of 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

Objects vs. Agents [Odell, 2002] I

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

Objects vs. Agents [Odell, 2002] II

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

Emergence

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

Objects vs. Agents [Odell, 2002] III

Analogies from nature and society

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

Towards the Coexistence of Agents and Objects

- 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, . . .

Next in Line. . .

Convergence Towards Computational 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

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
- Most of those area focus on individual agents, whereas MAS obviously emphasise the *multiplicity* of the agents composing a system
- The MAS area is today an independent research field & scientific community [Omicini and Poggi, 2006]

Summing up

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

A MAS Agent is Autonomous

A MAS agent is goal / task-oriented

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

A MAS agent pursues its goal / task. . .

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

So, what is new here?

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

A MAS Agent is Interactive

Limited perception, limited capabilities

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

A MAS agent lives not in isolation

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

Key-abstractions for MAS

- agents
- society
- environment

The Notion of Agent is Multi-faceted

Many reliable scientific sources

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

Finally, defining the agent notion

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

The meta-model is incomplete

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

Next in Line. . .

Agents are not the Only Abstractions Needed

Activity Theory is a social-organisational theory [Vygotsky, 1978, Nardi, 1996] providing a very general framework for conceptualising human activities—how people learn, how society evolves—based on the concept of human *activity* as the fundamental unit of analysis

Basic Abstractions: Agents plus Artifacts

- Adopting AT as a conceptual framework for MAS social activities has led to recognise that *agents are not the only basic abstractions* to model and build MAS [Ricci et al., 2003]
- Artifacts, too, are necessary [Ricci et al., 2006b]
  - to enable and constrain agent actions
  - to mediate agent interactions with other agents and with the environment
  - to model and shape MAS *environment*
  - in general, to improve agent ability to achieve their individual and social goals

Relevance of AT Research in MAS

Artifacts are essential—in MAS, too

- AT investigation is relevant in MAS since it points out that artifacts are essential to enable and govern agent actions and interactions within a MAS
  - by enhancing agent capabilities to act
  - by constraining both individual and social activities in a MAS

Role of environment

- AT emphasises the fundamental role of the *environment* in the development of complex systems
- Also, AT suggests that artifacts are the essential tools [Weyns et al., 2006, Viroli et al., 2005]
  - to model MAS environment
  - to shape it so as to make it favourable to the development of collaborative activities

Coordination Artifacts

Artifacts for collaboration and coordination

- *Coordination artifacts* are artifacts used in the context of collaborative activities, mediating the interaction among actors involved in the same social context [Ricci et al., 2003]
- Coordination artifacts can be either *embodied* or *disembodied*, referring to respectively physically or cognitive/psychological artifacts
- Coordination artifacts are social artifacts shared by agents in a MAS, which are meant to enable and govern the interaction among agents, and between agents and their environment

Coordination artifacts & media

- Coordination artifacts represent a straightforward generalisation of the notion of coordination medium, as coming from fields like coordination models and languages and distributed AI
- Examples include abstractions like tuple spaces, channels, blackboards, but also pheromone infrastructures, e-institutions, . . .

AT Layers for MAS

co-construction — agents understand and reason about the (social) objectives (goals) of the MAS, and build up a model of the social tasks required to achieve them—this also involves identifying interdependencies and interactions to be faced and managed

co-operation — agents design and build the coordination artifacts—either embodied (coordination media) or disembodied (plans, interaction protocols, etc.)—which are useful to carry on the social tasks and to manage the interdependencies and interactions devised out at the previous (co-construction) stage

co-ordination — agents use the coordination artifacts: then, the activities meant at managing interdependencies and interactions—either designed a-priori or planned at the co-operation stage—are enforced/automated

Levels of Use of Artifacts

Co-ordination: both intelligent and non-intelligent agents could coordinate

Any agent (either intelligent or not) can simply exploit artifacts to achieve its own goals by simply taking artifacts as they are, and use them

Co-operation: intelligent agents could change artifacts to change MAS

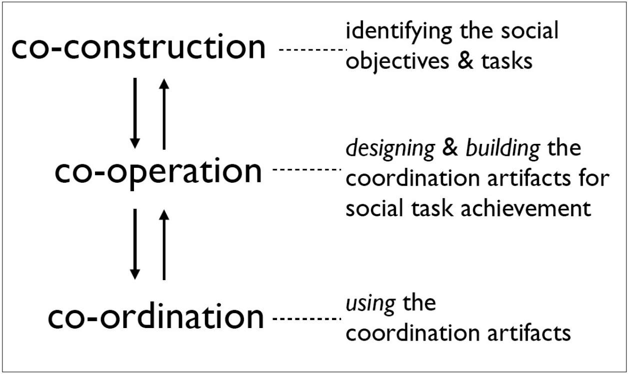
Intelligent agents could possibly reason about the nature of the artifacts as well as on the level of achievement of their goals, and take the chance to change or adapt the artifacts, or even to create new ones whenever useful and possible as the result of either an individual or a social activity

Co-operation: MAS engineers could embody social intelligence in artifacts

In the same way, MAS engineers can use artifacts to embody the "social intelligence" that actually characterises the systemic/synergistic (as opposed to compositional) vision of MAS [Ciancarini et al., 2000], but also to observe, control, and possibly change MAS social behaviour

AT Layers for MAS Collaboration: The Picture

AT Layers for MAS: The Picture



Next in Line. . .

Cognition Outside Agents

Distributed Cognition [Kirsh, 1999] is a branch of cognitive sciences which proposes that *human cognition* and *knowledge representation* rather than solely confined within individuals is *distributed* across individuals, tools and *artifacts* in the *environment*

Cognition & knowledge representation do not belong to agents only

- Objects & tools in the environment may participate to the cognitive processes
- Structure of MAS environment may explicitly represent knowledge

Cognition & knowledge representation are distributed in the environment

- Artifacts are essential parts of the MAS cognitive processes
- Cognitive artifacts encapsulate knowledge as explicitly represented

Agent View vs. MAS View

Personal / agent view

- Once artifacts are exploited, they change the way in which agents act and reason about action

System / MAS view

- In order to understand and possibly evaluate agent (social) action within a MAS, one should consider agent(s)+artifact(s) altogether

MAS Environment is Structured

(Cognitive) artifacts shape MAS environment

- Artifacts determine the structure of MAS environment
- Knowledge is distributed in the environment, and encapsulated within cognitive artifacts
- Structure of the environment, and knowledge it contains, affect the activities of agents within MAS

MAS Action & Coordination

MAS coordination depends on environment structure

- Environment structure changes the nature of agent action
- Environment structure affects agent mutual interaction
- Environment structure modifies the way agents coordinate in a MAS
- Environment structure should be designed to
  - help agent actions to achieve their goals
  - help epistemic, complementary, coordinative agent actions easier / effective

Next in Line. . .

Goals in MAS

We draw from the literature on cognitive interpretation of (social) action [Conte and Castelfranchi, 1995]

Agents have goals

strong agency Agents have explicitly-represented goals

weak agency Agents have implicitly-represented / encoded goals

Artifacts have functions

- Artifacts have no internal goals
- Artifacts have a pre-designed function
- An artifact is associated with an external goal (its destination) by agents in the act of using it

Agents & Artifacts Interacting

Aspects of agent-artifact relationship

use An agent can use an artifact, according to its use goal, associating it with a destination

- **aware use** because the agent is aware of the artifact's function
- **unaware use** because the artifact's use is encoded in the agent by the programmer / designer

selection An agent can select an artifact for future use, according to its use-value goal, reasoning about its possible future destinations and use goals

construction / manipulation An agent can modify an artifact to adapt its function to some required use-value goals and to its possible future destinations

- or, an agent can create *ex-novo* a new artifact with an agent-designed function according to some required use-value goals and to its possible future destinations

MAS Engineers Designing Agents

Basic choices to make in agent design

- Should an agent be aware of artifact's behaviour and structure, and of how to use them?
  - should an agent be able to reason and deliberate about artifact use?
- Should an agent be aware of artifact's function and possible uses?
- Should an agent be able to act over artifacts to modify them and adapt their function?
  - should an agent be able to create *ad hoc* artifacts *ex novo*?
- Should a MAS engineer be able to act over artifacts to modify them and adapt their function, or, to create new artifacts, *at run-time*?

MAS Engineers Designing Artifacts

Basic issues in artifact design

- How should an artifact be made in order to be ready for agent's use?
  - either aware, or unaware
  - possibly, within an open system
- How should an artifact be made in order to be ready for agent's evaluation and selection?
- How should an artifact be made in order to be ready for agent's modification and adaptation?
- How should MAS environment be structured in order to allow artifact run-time creation and modification?
  - by agents and MAS engineers?

Next in Line. . .

The Gap in CSCW [Schmidt and Simone, 2000]

CSCW aims at automating human cooperative work through computational procedures

Two diverging strategies are currently emerging in CSCW

automation stressing computational procedures to automate coordination of activities

flexibility stressing the flexibility of computational procedures with respect to intelligent coordination by collaborating actors

Automation of Collaboration Activities in MAS

Coordinative artifacts for automation of MAS collaboration

- *Coordinative artifacts* rule MAS collaboration, working more as constrainers rather than as commanders
- Coordinative artifacts structure MAS common field of work, as specialised abstractions automatising and making collaboration efficient
- As constrainers, coordinative artifacts define and govern the space of the admissible articulation of MAS collaboration activities
- On the other hand, they do not impose a pre-defined course of actions, promoting flexibility of intelligent agent coordination, and respecting agent autonomy

Flexibility of Collaboration Activities in MAS

Mutual awareness for flexibility of MAS collaboration

- Shared MAS environment should be structured as the MAS *common field of work* to allow agents to mutually perceive each other's activities (*mutual awareness*)
- MAS common field of work can reveal / conceal portions of MAS collaboration activities to the agents
- Mutual awareness promotes opportunistic alignment and improvisation of agent activities, and ensure *flexibility* to MAS collaboration

Next in Line. . .

The Logocentric Philosophical Bias in MAS

Agent capacity of language as the main sign of agent intelligence?

- Research on MAS still dwells on the *logocentric bias* [Hewes and Arcos, 1993]
- Intelligent *use of tools by agents* is typically neglected
  - as a stunning example, FIPA (Foundation for Intelligent Physical Agents) just ignore pragmatic / physical agent actions, and only focuses on agent communication actions

Agent ability of developing and using tools as a sign of agent intelligence

- A notion of *tool* for agents is needed
- Agents should be *able to use tools*
- Intelligent agents should be *able to forge & adapt tools*
- A theory of physical / pragmatical action should be developed for agents, as refined as the one for communication actions
- Such a theory should focus on tool use / creation by agents
  - The notion of *Agens Faber* goes along this very direction [Omicini et al., 2006a]

Tools, Agents, and the Tooling Test

Use of tools should be a feature for agents in a MAS

- As ethologists for animal intelligence [Povinelli, 2000], MAS researchers should be able to measure intelligence of agents by making them face problems that require the use of tools to be solved
- A sort of tool-equivalent of the Turing test for agents using tools should be defined, aimed at evaluating agent intelligence in terms of the ability to exploit tools
  - a sort of "Tooling Test for Agent Intelligence" [Wood et al., 2005]
- Agent intelligence should then be measured by both the agent ability to communicate and by agent ability to use tools
  - the two abilities should be somehow strictly related, and "co-evolve" in some sense—a common theory of agent action could be of use here

Next in Line. . .

Autonomy as the Foundation of the Definition of Agent

Lex Parsimoniae: Autonomy

- Autonomy as the only fundamental and definitory feature of agents
- Let us see whether other typical agent features 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

(Autonomous) Agents (Pro-)Act

Action as the essence of agency

- The etimology 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 *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 in the previous slide
- However, this issue will be even clearer when facing the issue of artifacts and environment design

(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 is not a definitory feature

- 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 Do not Need a Goal or a Task

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

Goal-/Task-Orientedness is not a Definitory Feature for Agents

Example: finite-state automaton with encapsulated control

- An agent might be a finite-state automaton
- Encapsulating control as an independent thread
- Equipped with state transition rules
- The criteria for the govern of control would there be embodied in terms of (finite) states and state transition rules

Goal-orientedness and task-orientedness are just possible features for agents

- They are not definitory features anyway

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
- *Intelligent autonomous agents* clearly make sense
  - intelligence, however, is *not* required for an agent to be autonomous

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

Agents in the A&A Meta-model

Definition (A&A Agent)

An A&A agent is an *autonomous computational entity*

genus agents are computational entities

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

A&A agents are *autonomous*

- 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*

Next in Line. . .

Artifacts in the A&A Meta-model

Definition (A&A Artifact)

An A&A artifact is a *computational entity* aimed at the *use* by A&A agents

- **genus** artifacts are computational entities
- **differentia** artifacts are aimed to be used by agents

Artifacts are *to be used* by agents

- From use, many other features stem
  - which are either essential or desirable, but need not to be used as definitory ones

Artifacts Have a Function

Artifacts are designed for use

- Being *aimed at* the agent's *use*, artifacts are *designed* to serve some purpose
  - and built as such
- When designed, they are then associated by design to their *function*
- Artifact function does not necessarily determine the actual use of the artifact by an agent
  - however, it incorporates the *aim* of the artifact designer, envisioning the artifact as potentially serving agent's purposes

Artifacts are transparent & predictable

transparency In order to be used by agents, artifact function should be available to / understood by agents

predictability In order to promote agent's use, artifact behaviour should be predictable

Artifacts Are Not Autonomous

Artifacts are designed to serve

- Artifacts are designed to *serve* some agent's purpose
  - not to follow their own path of action
- An artifact has an embodied function, made repeatedly and predictably available to agents
- An artifact is a tool in the "hands" of agents
  - it does not need to be self-governed, it just has to be "governed" by agents when they use it

Artifacts Are (Computationally) Reactive

Artifacts are *reactive* in terms of control

- Artifacts behave in response to agent use
  - the behaviour of an artifact just needs to emerge when it is used by an agent
- In terms of control, an artifact just needs to be *reactive*
  - or, to behave as it were
- What about reaction to change?
  - should artifacts be reactive to environment change?

Artifacts Have Operations and Interfaces

Agents use artifact operations

- In order to be used, artifacts should make *operations* available to agents
- Operations change an artifact's state, make it behave and produce the desired effects on the environment
- Either explicitly or implicitly, an artifact exhibits its *interface* to agents, as the collection of the operations made available

Artifacts are Situated

Artifacts & Agent Actions

- Being used, artifacts are the primary target / means of agent's action
  - action is what makes agents strictly coupled with the environment
- Artifact's function is expressed in terms of change to the environment
  - what the artifact actually *does* when used
- Artifact's model, structure & behaviour are *expressed* in terms of agent's actions and *environment*
  - artifacts are *situated*

Artifacts are reactive to change

- Along the same line used for agents, artifacts are then supposedly *reactive to change*
  - since they are structurally reactive in computational terms, this comes for free—unlike (proactive) agents

Artifacts Are Not Agents

Agents vs. artifacts

- Agents are autonomous, artifacts are not
- Agents encapsulate control, artifacts do not
- Agents are proactive, artifacts are not
- Agents are opaque, artifacts are transparent
- Artifacts are predictable, agents are not
- Agents may have a goal / task, artifacts do not
- Artifacts have a function, agents have not
- Agents use artifacts, but cannot use agents
- Agents speak with agents, but cannot speak with artifacts
- Agents are designed to govern, artifacts are designed to serve

Artifacts in the A&A Meta-model

Definition (A&A Artifact)

An A&A artifact is a *computational entity* aimed at the *use* by A&A agents

genus: artifacts are computational entities

differentia: artifacts are aimed to be used by agents

Artifacts are *to be used* by agents

- From use, many other features stem
  - artifacts have a function, are computationally reactive, are situated and reactive to change, are not autonomous, are transparent and predictable, have operations and interface for agent's use
  - artifacts are not agents

Next in Line. . .

Artifacts & Environment

Artifacts as mediators

- Artifacts mediate between agents and the environment
- Artifacts embody the portion of the environment that can be designed and controlled to support MAS activities

Artifacts as representatives of MAS environment

- As an observable & controllable part of the environment, artifacts can be monitored along with the development of MAS activities
  - to evaluate overall MAS performance
  - to keep track of MAS history
  - to influence MAS behaviour and evolution

Artifacts for environment design

- Artifacts are the essential tools
  - for modelling MAS environment
  - to shape MAS environment so as to make it favourable to the development of MAS social activities

Artifacts as Enablers and Constrainers of MAS Activities

- As mediating tools, artifacts have both an *enabling* and a *constraining* function

enablers
: artifacts expand out agent's ability to manipulate and transform different objects

constrainers
: the environment is perceived and manipulated by agents through the artifact not 'as such' but within the limitations set by the artifact itself

- A simple example: an agent-oriented printer driver

enabler
: enables agents to use a printer, along with a number of its options

constrainer
: limits in general agent interaction with the printer to some well-defined interaction patterns

Desirable Features of A&A Artifacts

How do we like artifacts?

- Artifacts could exhibit a number of relevant features, which would in principle enhance MAS engineers / agents ability to use them for their own purposes [Omicini et al., 2006a]
  - *inspectability*
  - *controllability*
  - *malleability / forgeability*
  - *predictability*
  - *formalisability*
  - *linkability*
  - *distribution*

A&A Artifacts: Inspectability

- The state of an artifact, its content (whatever this means in a specific artifact), its operations, interface and function might be all or partially available to agents through *inspectability*
- Whereas in closed MAS this information could be hard-coded in the agent—the artifact engineer develops the agents as well—, in open MAS third-party agents should be able to dynamically join a society and get aware at run-time of the necessary information about the available artifacts
- Also, artifacts are often in charge of critical MAS behaviour [Omicini et al., 2004a]: being able to inspect a part or the whole of an artifact features and state is likely to be a fundamental capability in order to understand and govern the dynamics and behaviour of a MAS

A&A Artifacts: Controllability

- *Controllability* is an obvious extension of the inspectability property
- The operational behaviour of an artifact should then not be merely inspectable, but also *controllable* so as to allow MAS engineers (or even intelligent agents) to monitor its proper functioning
  - it should be possible to stop and restart an artifact working cycle, to trace its inner activity, and to observe and control a step-by-step execution
- In principle, this would largely improve the ability of monitoring, analysing and debugging the operational behaviour of an artifact at execution time, and of the associated MAS social activities as well

A&A Artifacts: Malleability

- Also related to inspectability, *malleability* (also called *forgeability*) is a key-feature in dynamic MAS scenarios, when the behaviour of artifacts could require to be modified dynamically in order to adapt to the changing needs or mutable external conditions of a MAS
- Malleability, as the ability to change the artifact behaviour at execution time, is seemingly a crucial aspect in on-line engineering for MAS, and also a perspective key-issue for self-organising MAS

A&A Artifacts: Predictability

- Differently from agents—which as autonomous entities have the freedom of behaving erratically, e.g. neglecting messages—, artifact operations, interface and function description can be used as the stable basis for a contract between an artifact and an agent
- In particular, the description of the artifact function could provide precise details of the outcomes of exploiting the artifact, while description of the artifact operations, interface and behaviour should make the behaviour of an artifact *predictable* for an agent

A&A Artifacts: Formalisability

- The predictability feature can be easily related with *formalisability*
- Due to the precise characterisation that can be given to an artifact behaviour, until reaching e.g. a full operational semantics model—for instance, as developed for coordination artifacts in [Omicini et al., 2004b]—it might be feasible to automatically verify the properties and behaviour of the services provided by artifacts, for this is intrinsically easier than services provided by autonomous agents

A&A Artifacts: Linkability

- Artifacts can be used encapsulate and model reusable services in a MAS
- To scale up with complexity of an environment, it might be interesting to compose artifacts, e.g. to build a service incrementally on top of another, by making a new artifact realising its service by interacting with an existing artifact
- To this end, artifacts should be able to invoke the operation of another artifact: the reply to that invocation will be transmitted by the receiver through the invocation of another operation upon the caller

A&A Artifacts: Distribution

- Differently from an agent, which is typically seen as a point-like abstraction conceptually located to a single node of the newtwork, artifacts can also be distributed
- In particular, a single artifact can in principle be used to model a distributed service, accessible from more nodes of the net
- Using linkability, a distributed artifact can then be conceived and implemented as a composition of linked, possibly non-distributed artifacts—or viceversa, a number of linked artifacts, scattered through a number of different physical locations could be altogether seen as a single distributed artifact
- Altogether, distribution and linkability promote the *layering* of artifact engineering—as sketched in [Molesini et al., 2006b]

Next in Line. . .

Levels of Use of Artifacts

Co-ordination: both intelligent and non-intelligent agents could coordinate

Any agent (either intelligent or not) could simply exploit artifacts to achieve its own goals by simply taking artifacts as they are, and use them

Co-operation: intelligent agents could change artifacts to change MAS

Intelligent agents could possibly reason about the nature of the artifacts as well as on the level of achievement of their goals, and take the chance to change or adapt the artifacts, or even to create new ones whenever useful and possible as the result of either an individual or a social activity

Co-operation: MAS engineers could embody social intelligence in artifacts

In the same way, MAS engineers can use artifacts to embody the "social intelligence" that actually characterises the systemic/synergistic (as opposed to compositional) vision of MAS , but also to observe, control, and possibly change MAS social behaviour [Ciancarini et al., 2000]

Agents & Artifacts Interacting

Aspects of agent-artifact relationship

use An agent can use an artifact, according to its use goal, associating it with a destination

- **aware use** because the agent is aware of the artifact's function
- **unaware use** because the artifact's use is encoded in the agent by the programmer / designer

selection An agent can select an artifact for future use, according to its use-value goal, reasoning about its possible future destinations and use goals

construction / manipulation An agent can modify an artifact to adapt its function to some required use-value goals and to its possible future destinations

- or, an agent can create *ex-novo* a new artifact with an agent-designed function according to some required use-value goals and to its possible future destinations

A Model of A&A Artifacts

Rational exploitation of artifacts by intelligent agents

- In order to allow for its rational exploitation by intelligent agents, an A&A artifact possibly exposes
  - a *usage interface*
  - *operating instructions*
  - a *function description*

A&A Artifacts: Usage Interface

Agents, artifacts & operations

- One of the core differences between artifacts and agents is the concept of *operation*
- An operation is the means by which an artifact provides agents with a service or function
- An agent executes an action over an artifact by invoking an artifact operation
- Execution possibly terminates with an *operation completion*, typically representing the outcome of the invocation, which the agent comes to be aware of in terms of *perception*

usage interface The set of operations provided by an artifact defines what is called its *usage interface*

- which (intentionally) resembles interfaces of services, components or objects—in the object-oriented acceptation of the term

A&A Artifacts: Operating Instructions

Artifact's manuals for intelligent agents

- Operations cannot be invoked in any order
- Artifact's state & behaviour, along with the effects of agent's actions on the environment via the artifact, depend on the execution order of operations

operating instructions *Operating instructions* are a description of the procedure an agent has to follow to meaningfully interact with an artifact over time

- which should of course be coupled with usage interface

- Operating instructions are a description of the possible *usage protocols*, i.e. sequences of operations that can be invoked on the artifact, in order to exploit its function
- Besides a syntactic information, they can also embed some sort of semantic information for rational agents
  - rational agents can use such information for their practical reasoning
- Artifacts are conceptually similar to devices used by humans
  - operation instructions play for agents a role similar to a manual for a human—which a human reads to know how to use the device on a step-by-step basis, and depending on the expected outcomes he/she needs to achieve

A&A Artifacts: Function Description

Agents, artifacts & function

- Agents should be provided with a description of the functionality provided by the artifact
    - which agents essentially use for artifact selection

function description Artifacts could then be equipped with a *function description* (or, a *service description*), (formally) describing the function / service that the artifact is designed to provide agents with

- differently from operating instructions, which describes *how* to exploit an artifact, function description describes *what* to obtain from an artifact

An example

When modelling a sensor wrapper as an artifact, we may easily think of the operations for sensor activation and inspection as described via usage interface and operations instructions, while the information about the sensory function itself being conveyed through function description of the sensor wrapper

Next in Line. . .

MAS in the A&A Meta-model

Definition (A&A MAS)

An A&A MAS is a *computational systems* made of agents and artifacts

genus MAS is computational system

differentia its basic components are agents and artifacts

A constructive definition

- Based on the previous definitions
- Also based on on the (primitive) notion of system as well

A&A MAS are Situated

MAS & situatedness

- MAS are made of agents & artifacts
- Both agents & artifacts are situated computational entities
- As an obvious consequence, MAS are *situated computational systems*

MAS & environment

- A MAS is always immersed within an environment
- A MAS cannot be conceived / modelled / designed in a separate way with respect to its environment

A&A MAS have a Behaviour

MAS & activity

- MAS are made of agents & artifacts
- Agents are pro-active, artifacts are reactive
- Agents are autonomous entities, artifacts have functions

→ In the overall, a MAS has a behaviour that results from the interaction of autonomous, self-governing entities (agents) and reactive, functional entities (artifacts)

MAS Interaction in the A&A Meta-model

Admissible interactions *within* a MAS

- MAS are made of agents & artifacts
- Two fundamental entities give raise to four different sorts of admissible interactions

communication agents *speak* with agents
usage/operation agents *use* artifacts
composition artifacts *link* with artifacts
presentation artifacts *manifest* to agents

MAS interactions with the environment

- Defining a system is to define a boundary—the same holds for a MAS, of course
- Interactions occur within and without the boundaries
  - MAS interaction with the environment
- Depending on the desired level of abstraction, we may attribute environment interactions to either individual agents & artifacts, or to the MAS as a whole

Delimiting a MAS

MAS boundaries

- Our definition allows us to understand whether a computational system is a MAS
- It mostly define the class of the MAS in the A&A meta-model

What is an open system?

- How can we determine / recognise the boundaries of an open MAS?
- On the engineering side, how can we design an open MAS?
  - what should we actually design when designing a MAS?
  - what should anyway account for / account not?

Essence of a single MAS

MAS characteristic

- To define one single MAS, we need a characterising criterion
- The very notion of system means there is a coherent way to interpret the overall set of components as a whole, and to determine whether a given component belongs to a given MAS
- Characterising a single MAS then means firstly to define a criterion according to which an agent / an artifact could be said either to belong or not to a given MAS
  - hopefully in a univocal way
  - possibly dynamically depending on a number of parameters, like time, state of components, state of MAS, state of the environment, . . .

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, academical languages covering only some of the issues are proposed
  - finally, new well-founded languages are defined, which cover new spaces adequately and coherently

[a] SE here is taken in its broadest acceptation 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

Next in Line. . .

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]
  - see Rosenschein's slides for some basic agent architectures
- 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 other
  - a "language of pragmatical actions" should define how an agent can act over its environment
- A full-fledged agent language should account for both languages
  - so little 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

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

Programming the Interaction Space

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]

Enabling / governing / constraining the space of MAS interaction

- A number of inter-disciplinary fields of study insist on the space of (system) interaction
    - coordination
    - organisation
    - security

Coordination

Coordination in short

- Many different definitions around
  - we will talk about this later on 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" acceptation
  - 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 [Schumacher, 2001, Omicini and Ossowski, 2003]

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 & pragmatical actions
  - e.g. RBAC-MAS [Omicini et al., 2005a]

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- plus a run-time acceptation
- But tends to enforce a “negative” interpretation over interaction
    - “this is not allowed”
- It is then 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
  - could artifacts, and the A&A meta-model help on this?

Next in Line. . .

MAS Interaction Space in the A&A Meta-model

MAS interaction & A&A

- Agents *speak* with agents
- Agents *use* artifacts
- Artifacts *link* with artifacts
- Artifacts *manifest* to agents
  - these four sentences completely describe interaction *within* a MAS in the A&A meta-model
- What about programming languages now?
  - what about languages to be and languages to interact?

Programming Languages for Artifacts

Artifacts as MAS computational entities

- Artifacts are computational entities
  - with a *computational* (internal) *behaviour*
  - and an *interactive* (observable) *behaviour*
- Artifact programming languages are required
  - possibly covering *both* aspects
  - *to be* artifact, and *to interact* with agents and other artifacts

Programming Languages for Artifacts: Computation

Languages to be for artifacts

- Artifact computational behaviour is reactive
  - artifact languages should essentially be *event-driven*
- Artifacts belong to the agent interaction space within a MAS
  - artifact languages should be able to compute over MAS interaction
- Given the prominence of interaction in computation, artifact languages are likely to embody *both* aspects altogether

Programming Languages for Artifacts: Interaction

Languages to interact for artifacts

- Artifact interactive behaviour deals with agents and artifacts
  - artifact languages should provide operations for agents to use them
  - artifact languages should provide links for artifacts to link with them
- Artifacts work as mediators between agents and the environment
  - artifact languages should be able to react to environment events, and to observe / compute over them
- In the overall, artifacts may subsume agent's pragmatical actions, as well as environment's events & change
  - thus providing the basis for an engineering discipline of MAS interaction

Programming Languages for Artifacts: A&A Features

A&A artifact features in languages

- An artifact language may deal with artifact's usage interface
- An artifact language may deal with artifact's operating instructions
- An artifact language may deal with artifact's function description

Other artifact features in languages

- An artifact language may allow an artifact to be inspectable, controllable, malleable/forgeable, linkable, . . .

Programming Languages for A&A Agents

A&A agents deal with artifacts

- An agent programming language may deal with artifact's usage interface for artifact use
- An agent programming language may deal with artifact's operating instructions for practical reasoning about artifacts
- An agent programming language may deal with artifact's function description for artifact selection

Other features for agent programming languages

- An agent programming language may allow an A&A agent to inspect, control, forge, compose, . . . , artifacts of a MAS

Programming Languages for Artifacts: The Environment

Artifacts & MAS Environment

- Artifacts are our conceptual tools to model, articulate and shape MAS environment
  - to govern the agent interaction space
  - to build up the agent workspace

Artifacts for coordination, organisation & security

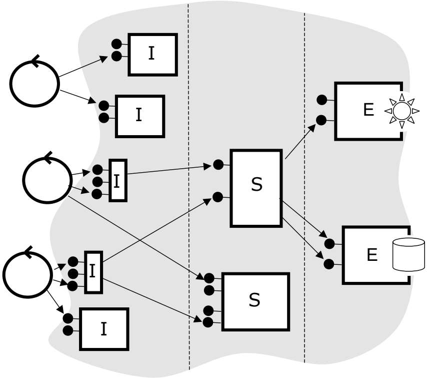
- Governing the interaction space essentially means coordination, organisation & security
- More or less the same holds for building agent workspace
- As a result, artifacts are our main places to model & engineer coordination, organisation & security in MAS

Layering Agent Workspace

A conceptual experiment

A layered taxonomy

- Individual artifacts
  - handling a single agent's interaction
- Social artifacts
  - handling interaction among a number of agents / artifacts
- Environment artifacts
  - handling interaction between MAS and the environment



Artifacts for MAS Organisation / Security

Individual artifacts

- Individual artifacts are the most natural place where to rule individual agent interaction within a MAS
  - on the basis of organisational / security concerns
- If an individual artifact is the only way by which an agent can interact within a MAS

  organisation there, role, permissions, obligations, policies, etc., should be encapsulated

  security working as a filter for any perception / action / communication between the agent, MAS and the environment

  autonomy it could work as the harmoniser between the clashing needs of agent autonomy and MAS control

  boundaries it could be used as a criterion for determining whether an agent belongs to a MAS
- Our example: Agent Coordination Contexts (ACC)
  - infrastructural abstraction associated to each agent entering a MAS

Artifact Languages for MAS Organisation / Security

Languages for individual artifacts

- Declarative languages (KR-style) for our "quasi static" perception of organisation
- Formal languages (like process algebras) for action / policy denotation
- Operational languages for modelling actions
- Example: Agent Coordination Contexts (ACC)
  - first-order logic (FOL) rules [Ricci et al., 2006a]
  - process algebra denotation [Omicini et al., 2006b]

Declarative does not mean static, actually

- organisation structure may change at run-time
- agents might reason about (organisation) artifacts, and possibly adapt their own behaviour, or change organisation structures

Artifacts for MAS Coordination

Social artifacts

- Social artifacts are the most natural place where to rule social interaction within a MAS
  - on the basis of (objective) coordination concerns
- Coordination policies could be distributed upon social artifacts, and there encapsulated

  inspectability there, coordination policies could be explicitly represented and made available for inspection

  controllability functioning of coordination engine could be controllable by engineers / agents

  malleability coordination policies could be amenable to change by agents / engineers
- Example
  - Tuple centres [Omicini and Denti, 2001] in TuCSoN [Omicini and Zambonelli, 1999] & MARS [Cabri et al., 2000]
  - JavaSpaces [Sun Microsystems, 2003] & TSpaces [Wyckoff et al., 1998]
  - e-institutions—e.g. AMELI middleware [Esteva et al., 2004]

Artifact Languages for MAS Coordination

Languages for social artifacts

- Typically operational, event-driven languages for our "dynamic" perception of coordination
  - interaction happens, the artifact has just to capture interaction and to react appropriately
- Examples
  - ReSpecT for tuple centres [Omicini, 2007]
  - Java for JavaSpaces [Sun Microsystems, 2003] & MARS [Cabri et al., 2000]
  - JavaScript when modelling browser users as agents

Operational does not mean static, too

- coordinative behaviour may change at run-time
- agents might reason about (coordination) artifacts, and possibly adapt their own behaviour, or change coordination policies

Artifacts for MAS Environment

Environment artifacts

- Environment artifacts are the most natural place where to rule interaction between a MAS and its environment
  - on the basis of artifact reactivity to change
  - and of their ability to manifest them to agents
- Spatio-temporal fabric as a source of events
  - **time** time events for temporal concerns
  - **space** spatial events for topological concerns
- Resources as sources of events and targets of actions
  - like a database, or a temperature sensor
- Examples
  - Timed Tuple Centres [Omicini et al., 2005b]
  - Sensor wrappers

Next in Line. . .

Software Engineering Abstractions

- Software deals with *abstract entities*, having a real-world counterparts
  - Numbers, dates, names, persons, documents...
- How should we model them in terms of computational entities?
  - data, functions, objects, agents...
  - i.e., what are the abstractions that we have to use to build up software systems?
- This might depend on the technologies available
  - use OO abstractions for OO programming environments
  - maybe, use OO abstractions because they are more expressive, even for COBOL-based programming environments

Why Agent-Oriented Software Engineering?

- Software engineering is necessary to discipline software systems and software development processes
  - Any SE approach relies on a set of abstractions and on related methodologies and tools
- Agent-oriented computing introduces novel abstractions and asks for
  - Clearly define the set of abstractions available in the diverse stages of the software development process
  - Adapting the methodologies and producing new tools
- This firstly requires a well-defined and suitably expressive ontology
  - This is what A&A comes for

SE Issues in MAS I

Autonomy

- Control encapsulation as a dimension of modularity
- Conceptually simpler to tackle than a single (or multiple inter-dependent) locus of control

Situatedness

- Clear separation of concerns between
  - the active computational parts of the system (the agents)
  - the resources of / the events from the environment

Sociality

- Not a single characterising protocol of interaction
- Interaction as an additional SE dimension

SE Issues in MAS II

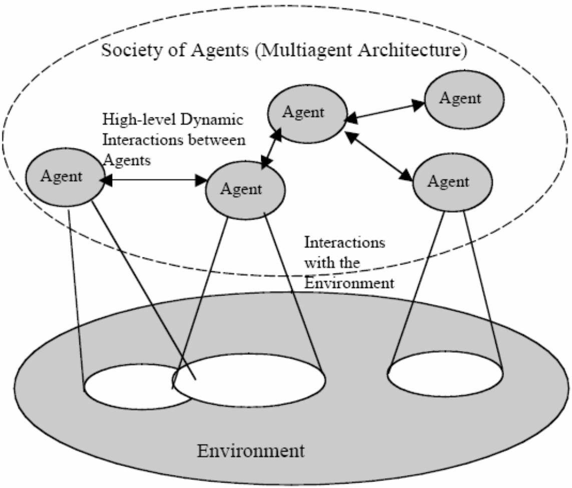
Openness

- Controlling self-interested agents, malicious behaviours, and badly programmed agents
- Dynamic re-organisation of software architecture

Mobility & Locality

- Additional dimension of autonomous behaviour
- Improve locality in interactions

Software Systems in terms of MAS



Agent-Oriented Abstractions

- The development of a MAS should fruitfully exploit *abstractions* coherent with the above characterisation
- It also should be able to drive software engineers in building

  agents
  : as autonomous entities, independent loci of control, situated in an environment, interacting with each other

  societies
  : as the subsystems in charge of relevant MAS behaviours / tasks / goals

  environment
  : as the world of resources agents perceive & act upon

Next in Line. . .

Agents & Artifacts

- When adopting the A&A meta-model, agents & artifacts constitute the basic building block both for
  - MAS analysis/modelling
  - MAS development
- Agents & artifacts can be assumed as the two fundamental abstractions for modelling MAS structure
  - Agents speaking with other agents
  - Agents using artifacts to achieve their objectives
  - Artifacts linking with artifacts to provide more complex functions
  - Artifacts manifesting to agents to enable reactive behaviours by agents and MAS

Engineering MAS with A&A

- Agents model individual activities
  - When grouped in societies, around social artifacts, agents capture articulated social activities
- Artifacts allow MAS engineers to
  - Model the environment as a first-class entity
  - Engineer the space of interaction among agents—not only mere conversations between agents, but complex agent interaction patterns
  - Enrich MAS design with social/organisational structure, topological models, as well as (complex) security models
- In particular
  - Social artifacts *glue* agents together, and build up agent societies
  - Individual artifacts mediate between individual agents and MAS
  - Environment artifacts wrap up and bring to the cognitive level of agents the resources of MAS and MAS environment as well

Changing the Level of Abstraction

Looking down

- Societies of agents are possibly assigned of social tasks
- Aggregates of artifacts could provide an articulated function
- At a higher level of abstraction, they could be seen as individual agents and artifacts, respectively

Looking up

- Individual agents may turn in agent societies at a subsequent stage of the engineering process just applying a decomposition principle
- Individual artifacts may turn in aggregates of linked artifact for the very same reason
- At a lower level of abstraction, they are no longer seen as individual agents and artifacts, respectively

Layering and MAS

Layering

- In general, layering is a fundamental principle of complex systems
- The engineering of non-trivial computational systems may benefit from the availability of layering mechanisms
- To this end, MAS models, abstractions, patterns and technologies should be suitably categorised and compared using a layered description
- Accordingly, agent-oriented processes and methods should support some forms of MAS layering, allowing engineers to design and develop MAS along different levels of abstractions

Next in Line. . .

The SODA Agent-Oriented Methodology

- SODA (Societies in Open and Distributed Agent spaces) is an agent-oriented methodology for the analysis and design of agent-based systems
- SODA focuses on inter-agent issues, like the engineering of societies and environments for MAS [Omicini, 2001]
- SODA adopts the A&A meta-model as the conceptual foundation for MAS software development process [Molesini et al., 2006a]
- SODA introduces a simple and expressive layering principle in order to manage the complexity of the system description [Molesini et al., 2006b]
- SODA adopts a tabular representations

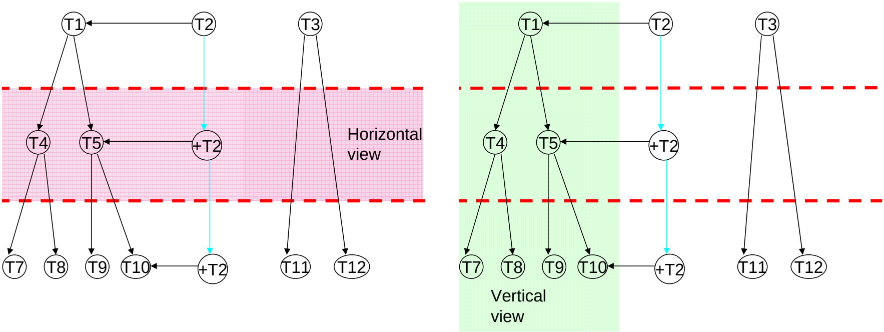
Layering in SODA

- Layering in SODA is based upon the two mechanisms of *zooming* and *projection*
- The zooming mechanism includes two kinds of zoom
  - *in-zoom*: when passing from an abstract layer down to a more detailed one
  - *out-zoom*: when passing from a detailed layer up to a more abstract one
- The *projection mechanism* projects the non-zoomed entities shared by two adjacent layers, so as to make it possible the internal consistency of one layer
- It is then possible to have two types of *system's view*

Horizontal view to analyse the system at one single level of detail

Vertical view to analyse one single abstract entity in its wholeness across layers

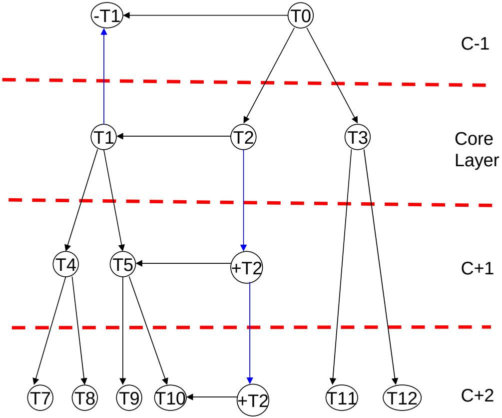
System's views



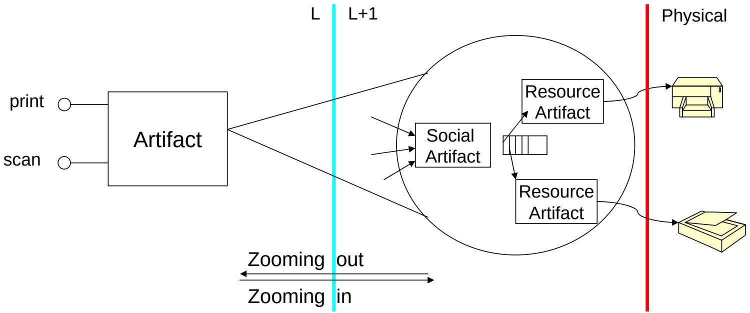
Layering in SODA

- In general, when working with SODA, we start from the *core layer*, labelled with “c”
- The core layer is always *complete* by definition
- In the other layers we find only the in/out zoomed entities and the projected entities
- In-zoomed layers are labelled with “c+1”, “c+2”
- Out-zoomed layers are labelled “c-1”, “c-2”...
- Projected entities are labelled with “+” if the projection is from a more abstract layer to a more detailed layer, “-” otherwise
- The only relations between layers are the *zooming relation* expressed by means of zooming tables
- Relations between entities belonging to different layers cause projection of such entities

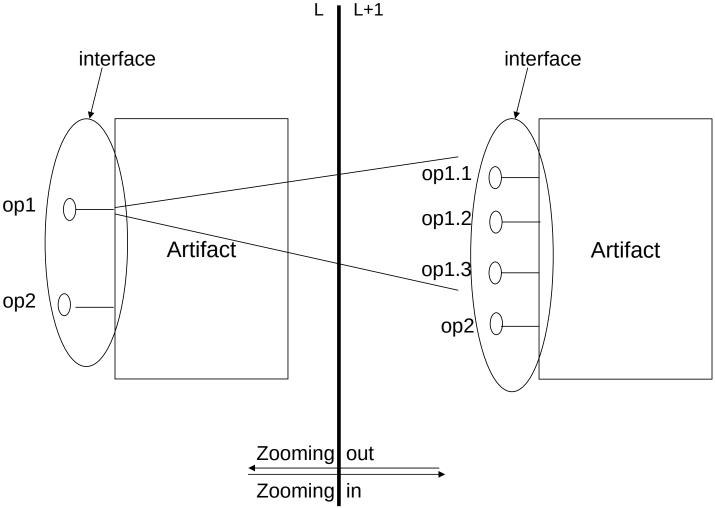
Example



Zooming Artifacts I



Zooming Artifacts II



Artifacts I

Agents are not alone in MAS

As humans are not alone in societies and organisations

Adopting artifacts as a basic brick for MAS

- defines a more articulated meta-model
  - allowing existing MAS to be re-interpreted
  - providing a new notion of agent intelligence
- mandates for new classes of programming languages
- impacts on the way in which MAS are engineered

Artifacts II

A&A Meta-model

- A&A adopts agents & artifacts as its basic abstractions
- A&A can be / is used as a coherent conceptual framework to re-define programming languages, MAS infrastructures, simulation frameworks, . . .

Artifacts & OO

- Artifacts can be used as the fundamental abstraction
  - to leverage OO languages, technologies, methodologies and tools in agent-oriented systems
  - in a conceptually well-founded way

Future Works I

Workspaces

- Topology is an essential feature in today non-trivial computational systems
- *Workspaces* are the A&A abstraction to capture topology

Infrastructure & Tools

- In order to promote a paradigm shift, suitable infrastructures should be defined
- along with coherent tools for software development
  - re-interpreting ReSpecT, TuCSoN and other infrastructures
  - building CArtAgO [Ricci et al., 2007b]

Future Works II

Simulation

- A more expressive meta-model for computational systems also implies more expressive power in simulation
- An A&A simulation framework is under construction and testing [Montagna et al., 2006]
- Current scenario: biological systems under the Systems Biology perspective [Montagna et al., 2007]

Future Works III

Self-*

- Autonomic Computing & Self-* Systems address many typical MAS concerns
- The role of interaction, locality, environment, . . . , is essential for self-* properties in both natural and artificial systems
- On-going experiments in MAS with self-* based on artifacts
  - Cognitive Stigmergy [Ricci et al., 2007a]

Acknowledgements

The aliCE Group in Cesena

- Alessandro Ricci & Mirko Viroli
- Antonio Natali & Enrico Denti
- Matteo Casadei, Maurizio Cimadamore, Luca Gardelli, Sara Montagna, Ambra Molesini, Elena Nardini, Enrico Oliva, Giulio Piancastelli, Michele Piunti
- Cesare Monti, Marco Fabbri, Michele Cabano, . . .

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.

Cabri, G., Leonardi, L., and Zambonelli, F. (2000).
Mars: A programmable coordination architecture for mobile agents.
IEEE Internet Computing, 4(4):26–35.

Ciancarini, P., Omicini, A., and Zambonelli, F. (2000).
Multiagent system engineering: The coordination viewpoint.
In Jennings, N. R. and Lespérance, Y., editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *LNAI*, pages 250–259. Springer-Verlag.
6th International Workshop (ATAL'99), Orlando, FL, USA, 15–17 July 1999. Proceedings.

Conte, R. and Castelfranchi, C., editors (1995).
Cognitive and Social Action.
University College London.

Bibliography II

Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., and Lluís Arcos, J. (2004).
AMELI: An agent-based middleware for electronic institutions.
In Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 236–243, New York, USA. IEEE Computer Society.

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.

Hewes, G. W. and Arcos, J. L. (1993).
A history of speculation on the relation between tools and languages.
In Gibson, K. R. and Ingold, T., editors, *Tools, Language & Cognition in Human Evolution*, pages 20–31. Cambridge University Press.

Bibliography III

Kirsh, D. (1999).
Distributed cognition, coordination and environment design.
In Bagnara, S., editor, *3rd European Conference on Cognitive Science (ECCS'99)*, pages 1–11, Certosa di Pontignano, Siena, Italy. Istituto di Psicologia, Consiglio Nazionale delle Ricerche.

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

Molesini, A., Omicini, A., Denti, E., and Ricci, A. (2006a).
SODA: A roadmap to artefacts.
In Dikenelli, O., Gleizes, M.-P., and Ricci, A., editors, *Engineering Societies in the Agents World VI*, volume 3963 of *LNAI*, pages 49–62. Springer.
6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised, Selected & Invited Papers.

Bibliography IV

Molesini, A., Omicini, A., Ricci, A., and Denti, E. (2006b).
Zooming multi-agent systems.
In Müller, J. P. and Zambonelli, F., editors, *Agent-Oriented Software Engineering VI*, volume 3950 of *LNCS*, pages 81–93. Springer.
6th International Workshop (AOSE 2005), Utrecht, The Netherlands, 25–26 July 2005. Revised and Invited Papers.

Montagna, S., Omicini, A., Ricci, A., and d'Inverno, M. (2007).
Modelling hematopoietic stem cell behaviour: An approach based on multi-agent systems.
In Allgöwer, F. and Reuss, M., editors, *2nd Conference "Foundations of Systems Biology in Engineering" (FOSBE 2007)*, pages 243–248, Stuttgart, Germany. Fraunhofer IBR Verlag.

Montagna, S., Ricci, A., and Omicini, A. (2006).
Agents & Artifacts for Systems Biology: Toward a framework based on TuCSoN.
In Genco, A., Gentile, A., and Sorce, S., editors, *Industrial Simulation Conference 2006 (ISC 2006)*, pages 25–32, Palermo, Italy. EUROSIS (The European Simulation Society) & ETI (The European Technology Institute).

Bibliography V

Nardi, B. A., editor (1996).
Context and Consciousness: Activity Theory and Human-Computer Interaction.
MIT Press.

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

Omicini, A. (2001).
SODA: Societies and infrastructures in the analysis and design of agent-based systems.
In Ciancarini, P. and Wooldridge, M. J., editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer-Verlag.
1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.

Bibliography VI

Omicini, A. (2007).
Formal ReSpecT in the A&A perspective.
Electronic Notes in Theoretical Computer Sciences, 175(2):97–117.
5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.

Omicini, A. and Denti, E. (2001).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.

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.

Bibliography VII

Omicini, A., Ossowski, S., and Ricci, A. (2004a).
Coordination infrastructures in the engineering of multiagent systems.
In Bergenti, F., Gleizes, M.-P., and Zambonelli, F., editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter 14, pages 273–296. Kluwer Academic Publishers.

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. (2005a).
An algebraic approach for modelling organisation, roles and contexts in MAS.
Applicable Algebra in Engineering, Communication and Computing, 16(2-3):151–178.
Special Issue: Process Algebras and Multi-Agent Systems.

Bibliography VIII

Omicini, A., Ricci, A., and Viroli, M. (2005b).
Time-aware coordination in ReSpecT.
In Jacquet, J.-M. and Picco, G. P., editors, *Coordination Models and Languages*, volume 3454 of *LNCS*, pages 268–282. Springer-Verlag.
7th International Conference (COORDINATION 2005), Namur, Belgium, 20–23 April 2005. Proceedings.

Omicini, A., Ricci, A., and Viroli, M. (2006a).
Agens Faber: Toward a theory of artefacts for MAS.
Electronic Notes in Theoretical Computer Sciences, 150(3):21–36.
1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.

Omicini, A., Ricci, A., and Viroli, M. (2006b).
Agent Coordination Contexts for the formal specification and enactment of coordination and security policies.
Science of Computer Programming, 63(1):88–107.
Special Issue on Security Issues in Coordination Models, Languages, and Systems.

Bibliography IX

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004b).
Coordination artifacts: Environment-based coordination for intelligent agents.
In Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 286–293, New York, USA. ACM.

Omicini, A. and Zambonelli, F. (1999).
Coordination for Internet application development.
Autonomous Agents and Multi-Agent Systems, 2(3):251–269.
Special Issue: Coordination Mechanisms for Web Agents.

Povinelli, D. J. (2000).
Folk Physics for Apes: The Chimpanzee's Theory of How the World Works.
Oxford University Press.

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 X

Ricci, A., Omicini, A., and Denti, E. (2003).
Activity Theory as a framework for MAS coordination.
In Petta, P., Tolksdorf, R., and Zambonelli, F., editors, *Engineering Societies in the Agents World III*, volume 2577 of *LNCS*, pages 96–110. Springer-Verlag.
3rd International Workshop (ESAW 2002), Madrid, Spain,
16–17 September 2002. Revised Papers.

Ricci, A., Omicini, A., Viroli, M., Gardelli, L., and Oliva, E. (2007a).
Cognitive stigmergy: Towards a framework based on agents and artifacts.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 124–140. Springer.
3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006.
Selected Revised and Invited Papers.

Ricci, A., Viroli, M., and Omicini, A. (2006a).
Agent coordination contexts in a MAS coordination infrastructure.
Applied Artificial Intelligence, 20(2–4):179–202.
Special Issue: Best of "From Agent Theory to Agent Implementation (AT2AI) – 4".

Bibliography XI

Ricci, A., Viroli, M., and Omicini, A. (2006b).
Programming MAS with artifacts.
In Bordini, R. P., Dastani, M., Dix, J., and El Fallah Seghrouchni, A., editors, *Programming Multi-Agent Systems*, volume 3862 of *LNAI*, pages 206–221. Springer.
3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers.

Ricci, A., Viroli, M., and Omicini, A. (2007b).
CArtAgO: A framework for prototyping artifact-based environments in MAS.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 67–86. Springer.
3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

Bibliography XII

Schmidt, K. and Simone, C. (2000).
Mind the gap! towards a unified view of CSCW.
In Dieng, R., Giboin, A., Karsenty, L., and De Michelis, G., editors, *Designing Cooperative Systems: The Use of Theories and Models*, volume 58 of *Frontiers in Artificial Intelligence and Applications*, Sophia Antipolis, France. IOS Press.
4th International Conference on the Design of Cooperative Systems (COOP 2000), Proceedings.

Schumacher, M. (2001).
Objective Coordination in Multi-Agent System Engineering – Design and Implementation, volume 2039 of *LNAI*.
Springer-Verlag.

Sun Microsystems (2003).
JavaSpaces 2.0 Home Page.
`wwws.sun.com/software/jini/`.

Bibliography XIII

Viroli, M., Omicini, A., and Ricci, A. (2005).
Engineering MAS environment with artifacts.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *2nd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2005)*, pages 62–77, AAMAS 2005, Utrecht, The Netherlands.

Vygotsky, L. S. (1978).
Mind and Society.
Harvard University Press.

Weyns, D., Omicini, A., and Odell, J. (2006).
Environment as a first-class abstraction in multi-agent systems.
Autonomous Agents and Multi-Agent Systems, Online First.
Special Issue: Environment for Multi-Agent Systems.

Wood, A. B., Horton, T. E., and Amant, R. S. (2005).
Effective tool use in a habile agent.
In Bass, E. J., editor, *2005 IEEE Systems and Information Engineering Design Symposium (SEADS 2005)*, pages 75–81, Charlottesville, VA, USA. IEEE.

Bibliography XIV

Wyckoff, P., McLaughry, S. W., Lehman, T. J., and Ford, D. A. (1998).
T Spaces.
IBM Journal of Research and Development, 37(3 - Java Techonology):454–474.

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 & Artifacts
A Meta-Model for Agent-Oriented Computing

Andrea Omicini
andrea.omicini@unibo.it

Alma Mater Studiorum—Università di Bologna a Cesena

WOA 2007 School