Programming Languages for Distributed Systems as Multiagent Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2015/2016

Outline

Outline

Paradigm Shifts in Software Engineering

New classes of programming languages

- New classes of programming languages come from paradigm shifts in Software Engineering
  - 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

The Problem of PL & SE Today

Things are running too fast

- New classes of programming languages emerge too fast from the needs of real-world software engineering
- However, technologies (like programming language frameworks) require a reasonable amount of time (and resources, in general) to be suitably developed and stabilised, before they are ready for SE practise

→ Most of the time, SE practitioners have to work with languages (and frameworks) they know well, but which do not support (or, incoherently / insufficiently support) required abstractions & mechanisms

→ This makes methodologies more and more important with respect to technologies, since they can help covering the "abstraction gap" in technologies

Outline

An Example: CORBA & Distributed Objects

OOP technologies moving too slow

- As soon as OOP moved out of academia to enter SE practises, new needs had already emerged
- Distribution of software applications required new solutions, and created new spaces for programming languages
- Distributed objects were the first answer, and distributed infrastructures like CORBA were developed
- On the one hand, new (classes of) languages like IDL were introduced
- On the other hand, the development of a stable & reliable technology was so slow, that the first "usable" CORBA implementation (3.0) came too late, and never established itself as the standard reference technology

Another Example: Java & Web Technologies

- What is the standard framework for distributed systems today?
  - Java, for distributed objects
  - The Web, for most distributed applications
- None of them, however, was born for this
  - Java was born as a programming language
    - today Java is typically conceived as a platform, or a distributed framework
  - The Web was born as a mere concept, implemented via HTML pages, server & browsers
    - today the Web is a sort of cluster of related technologies in ultra-fast growth
- Both of them suffer from a *lack of conceptual coherence*
  - in Java, syntax and basic language mechanisms are the only glue
  - in Web technologies, the client / server pattern is the only unifying model
  - conceptual integrity is lost in principle

Outline

Outline

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.
  - behaviour-based architecture in JADE [BCG07]
  - the BDI (Belief, Desire, Intention) architecture [RG91]
- 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

Intra-agent languages, Inter-agent languages

- Agent programming languages should be either / both

intra-agent languages languages to define (agent) computational behaviour

inter-agent languages languages to define (agent) interactive behaviour

Example: Agent Communication Languages (ACL)

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

Agents Without Agent Languages

What if we do not have an agent language available?

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

Outline

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 [Sch01, OO03]

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 [Sch01, OO03]

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 and power
  - typically based on the notion of *role*
  - requiring a model of communicative & pragmatical actions
  - e.g. RBAC-MAS [ORV05]

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 artefacts, and the A&A meta-model help on this?

Outline

Outline

1. Spaces for Programming Languages in Software Engineering
   - Paradigm Shifts
   - Examples
2. Spaces for Programming Languages in Multiagent Systems
   - Programming Agents
   - Programming MAS
3. Spaces for Programming Languages in the A&A Meta-model
   - Generality
   - Environment, Coordination, Organisation & Security
4. Remarkable Cases of (Programming) Languages for Multiagent Systems

The A&A Meta-model in Short

A&A: A conceptual framework for MAS modelling & engineering

Based on the conceptual foundations discussed in the previous block of slides, the A&A meta-model is a conceptual framework characterised in terms of three basic abstractions [ORV08]:

agents represent pro-active components of the systems, encapsulating the autonomous execution of some kind of activities inside some sort of environment

artefacts represent passive components of the systems such as resources and media that are intentionally constructed, shared, manipulated and used by agents to support their activities, either cooperatively or competitively

workspaces are the conceptual containers of agents and artefacts, useful for defining the topology for the environment and providing a way to define a notion of locality

Artefacts in the A&A Meta-model

Definition (A&A Artefact)

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

genus artefacts are computational entities

differentia artefacts are aimed to be used by agents

Artefacts are *to be used* by agents

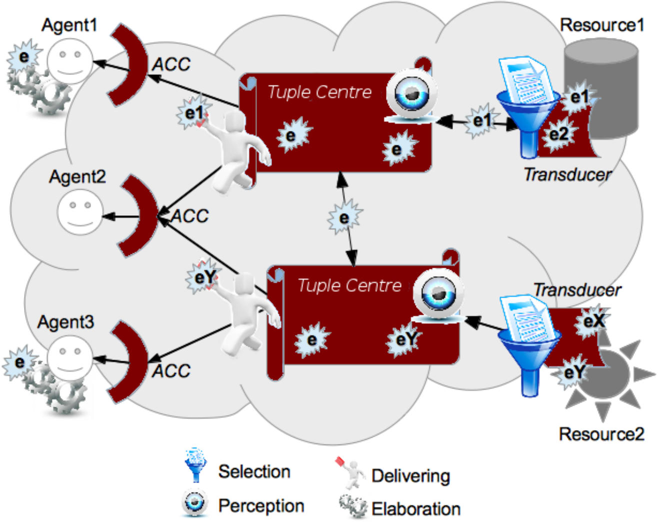
- From use, many other features stem
  - which are either essential or desirable, but need not to be used as definitory ones

Artefacts in the TuCSoN Architecture I

Examples

- Coordination media
- Agent Coordination Contexts (ACC)
- Transducers

Artefacts in the TuCSoN Architecture II



MAS Interaction Space in the A&A Meta-model

MAS interaction & A&A

- Agents *speak* with agents
- Agents *use* artefacts
- Artefacts *link* with artefacts
- Artefacts *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 compute and languages to interact?

Programming Languages for Artefacts

Artefacts as MAS computational entities

- Artefacts are computational entities
  - with a *computational* (internal) *behaviour*
  - and an *interactive* (observable) *behaviour*
- Artefact programming languages are required
  - possibly covering *both* aspects
  - intra-artefact languages, to compute within artefacts, and
  - inter-artefact languages, to interact with agents and other artefacts

Programming Languages for Artefacts: Computation

Intra-artefact languages

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

Programming Languages for Artefacts: Interaction

Inter-artefact languages

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

Outline

Programming Languages for Artefacts: The Environment

Artefacts & MAS Environment

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

Artefacts for coordination, organisation & security

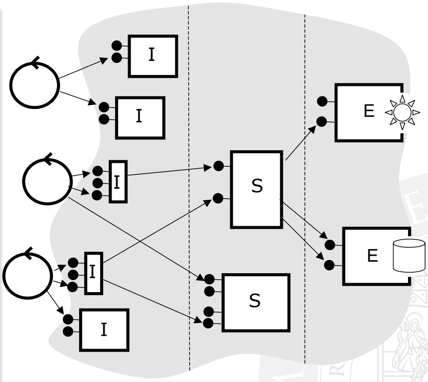
- Governing the interaction space essentially means coordination, organisation & security
- More or less the same holds for building agent workspace
- As a result, artefacts are our main places to model & engineer coordination, organisation & security in MAS

Layering Agent Workspace

A conceptual experiment

A layered taxonomy [MORD06]

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



Artefacts for MAS Organisation / Security

Individual artefacts

- Individual artefacts are the most natural place where to rule individual agent interaction within a MAS
  - on the basis of organisational / security concerns
- If an individual artefact 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

- Example: Agent Coordination Contexts (ACC)
  - infrastructural abstraction associated to each agent entering a MAS

Artefact Languages for MAS Organisation / Security

Languages for individual artefacts

- 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 [RVO06a]
  - process algebra denotation [ORV06]

Declarative does not mean static, actually

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

Artefacts for MAS Coordination

Social artefacts

- Social artefacts 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 artefacts, 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 [OD01]
  - coordination abstractions for MAS coordination
  - logic tuple centres for coordinative / awareness artefacts
  - ReSpecT tuple centres for A&A [Omi07]

Artefact Languages for MAS Coordination

Languages for social artefacts

- Typically operational, event-driven languages for our "dynamic" perception of coordination
  - interaction happens, the artefact has just to capture interaction and to react appropriately
- Example: ReSpecT
  - first-order logic (FOL) language
  - semantics given operationally [Omi07]
  - ongoing work on multiset rewriting semantics (with Maude)

Operational does not mean static, too

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

Artefacts for MAS Environment

Environment artefacts

- Environment artefacts are the most natural place where to rule interaction between a MAS and its environment
  - on the basis of artefact reactivity to change
- 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
- Example: Situated Tuple Centres [ORV07, CO09, MO13]

Outline

Agent Communication Languages (ACL) I

Speech acts

- Inspired by the study of human communication
- Communication is based on direct exchange of messages between agents
  - specifying agent communicative actions
- Speaking agent acts to change the world around
  - in particular, to change the belief of another agent
- Every message has three fundamental parts

  performative the pragmatics of the communicative action

  content the syntax of the communicative action

  ontology the semantics of the communicative action

Agent Communication Languages (ACL) II

Examples

- Examples, working as standard protocols for information exchange between agents

  KQML Knowledge Query Manipulation Language
  `http://www.cs.umbc.edu/kqml/` [LF97]

  FIPA ACL FIPA Agent Communication Language
  `http://www.fipa.org/repository/aclspecs.html` [FIP02]

Agent Oriented Programming Languages (AOP) I

Programming languages for cognitive agents

- Mentalistic agents
  - either BDI or other cognitive architectures
- Facilities and structures to represent internal knowledge, goals, ...
- Architecture to implement practical reasoning

Examples

3APL Programming language for cognitive agents
`http://www.cs.uu.nl/3apl/` [DvRDM04, DvRM05]

Jason Java-based interpreter for an extended version of AgentSpeak(L) for programming BDI agents
`http://jason.sourceforge.net/` [Rao96, BH06]

Artefact Programming Languages: Coordination

Languages to program social / environment artefacts

- Example: ReSpecT
  - Programming language for cognitive agents
    http://respect.alice.unibo.it/ [Omi07, OD01]
- Tuple centres as coordinative artefacts
  - programmable tuple spaces
  - encapsulating coordination policies
- Logic tuple centres as awareness artefacts
- ReSpecT tuple centres as social artefacts
  - ReSpecT as the event-driven, logic-based language to program tuple centres behaviour
  - Timed ReSpecT as an event-driven language to react to environment change

Artefact Programming Languages: Organisation / Security

Languages to program individual artefacts

- Example: Agent Coordination Context (ACC)
  - individual artefact
  - associated to each individual agent in a MAS
  - filtering every interaction of its associated agent
- RBAC-MAS as the organisational model [ORV06]
- Languages for policy specification & enaction
  - logic-based [RVO06a]
  - process algebra [ORV05]

Non-Agent Programming Languages I

Building the agent abstraction layer

- Examples

  Prolog programming logic agents in Prolog

  Java programming simple agents in Java: examples in TuCSoN

Non-Agent Programming Languages II

Agents using artefacts

- Examples

  tuProlog logic agents using ReSpecT tuple centres: examples in tuProlog `http://tuprolog.apice.unibo.it/` [DOR05]

  simpA extending Java towards A&A agents & artefacts: examples in simpA `http://simpa.apice.unibo.it/`

  Java/TuCSoN simple Java agents using TuCSoN tuple centres and ACC `http://tucson.apice.unibo.it/`

  Jason/CArtAgO Jason agents using CArtAgO artefacts `http://cartago.apice.unibo.it/` [RVO06b, RVO07]

References I

Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood.
Developing Multi-Agent Systems with JADE.
Wiley, February 2007.

Rafael H. Bordini and Jomi F. Hübner.
BDI agent programming in AgentSpeak using *Jason* (tutorial paper).
In Francesca Toni and Paolo Torroni, editors, *Computational Logic in Multi-Agent Systems*, volume 3900 of *Lecture Notes in Computer Science*, pages 143–164. Springer, April 2006.
6th International Workshop, CLIMA VI, London, UK, June 27-29, 2005. Revised Selected and Invited Papers.

References II

Matteo Casadei and Andrea Omicini.
Situated tuple centres in ReSpecT.
In Sung Y. Shin, Sascha Ossowski, Ronaldo Menezes, and Mirko Viroli, editors, *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, volume III, pages 1361–1368, Honolulu, Hawai'i, USA, 8–12 March 2009. ACM.

Enrico Denti, Andrea Omicini, and Alessandro Ricci.
Multi-paradigm Java-Prolog integration in tuProlog.
Science of Computer Programming, 57(2):217–250, August 2005.

References III

Mehdi Dastani, Birna van Riemsdijk, Frank Dignum, and John-Jules Ch. Meyer.
A programming language for cognitive agents: Goal directed 3APL.
In Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3067 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 2004.
1st International Workshop, PROMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited Papers.

Mehdi Dastani, Birna van Riemsdijk, and John-Jules Ch. Meyer.
Programming multi-agent systems in 3APL.
In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 39–67. Springer, 2005.

References IV

Foundation for Intelligent Physical Agents (FIPA).
Agent Communication Language Specifications, 2002.

Yannis Labrou and Tim Finin.
Semantics and conversations for an agent communication language.
In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 235–242. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

References V

Stefano Mariani and Andrea Omicini.
Space-aware coordination in ReSpecT.
In Matteo Baldoni, Cristina Baroglio, Federico Bergenti, and Alfredo Garro, editors, *From Objects to Agents*, volume 1099 of *CEUR Workshop Proceedings*, pages 1–7, Turin, Italy, 2–3 December 2013. Sun SITE Central Europe, RWTH Aachen University.
XIV Workshop (WOA 2013). Workshop Notes.

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

References VI

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

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

References VII

Andrea Omicini and Sascha Ossowski.
Objective versus subjective coordination in the engineering of agent systems.
In Matthias Klusch, Sonia Bergamaschi, Peter Edwards, and Paolo Petta, editors, *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*, pages 179–202. Springer, 2003.

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

References VIII

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

Andrea Omicini, Alessandro Ricci, and Mirko Viroli.
Timed environment for Web agents.
Web Intelligence and Agent Systems, 5(2):161–175, August 2007.

References IX

Andrea Omicini, Alessandro Ricci, and Mirko Viroli.
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17(3):432–456, December 2008.
Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.

Anand S. Rao.
AgentSpeak(L): BDI agents speak out in a logical computable language.
In Walter Van de Velde and John W. Perram, editors, *Agents Breaking Away*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996.
7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), Eindhoven, The Netherlands, 22-25 January 1996, Proceedings.

References X

Anand S. Rao and Michael P. Georgeff.
Modeling rational agents within a BDI architecture.
In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

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

References XI

Alessandro Ricci, Mirko Viroli, and Andrea Omicini.
Construenda est CArtAgO: Toward an infrastructure for artifacts in MAS.
In Robert Trappl, editor, *Cybernetics and Systems 2006*, volume 2, pages 569–574, Vienna, Austria, 18–21 April 2006. Austrian Society for Cybernetic Studies.
18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), 5th International Symposium "From Agent Theory to Theory Implementation" (AT2AI-5). Proceedings.

References XII

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

- Michael Schumacher.
  Objective Coordination in Multi-Agent System Engineering. Design and Implementation, volume 2039 of *LNCS*.
  Springer, April 2001.

Programming Languages for Distributed Systems as Multiagent Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2015/2016