Programming Languages for Multiagent Systems

Multiagent Systems LS

Sistemi Multiagente LS

Andrea Omicini

andrea.omicini@unibo.it

Ingegneria Due

Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008

Paradigm Shifts in Software Engineering

New classes of programming languages

- New classes of programming languages come from paradigm shifts in Software Engineering[1]
  - 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

---

[1] SE here is taken in its broadest acceptation as the science of building

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

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

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?

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

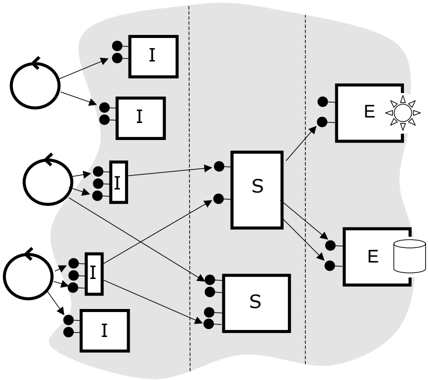
- Governing the interaction space essentially means coordination, organisation & security
- More or less the same holds for building agent workspace
- As a result, artifacts are our main places to model & engineer coordination, organisation & security in MAS

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
- Our example: Agent Coordination Contexts (ACC)
  - first-order logic (FOL) rules [Ricci et al., 2006a]
  - process algebra denotation [Omicini et al., 2006]

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
- Our example: Tuple Centres [Omicini and Denti, 2001]
  - coordination abstractions for MAS coordination
  - logic tuple centres for coordinative / awareness artifacts
  - ReSpecT tuple centres for A&A [Omicini, 2006]

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
- Our example: ReSpecT
  - first-order logic (FOL) language
  - semantics given operationally [Omicini, 2006]
  - 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) 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
- 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
- Our (limited) example: Timed Tuple Centres [Omicini et al., 2005b]
  - coordination abstractions reactive to the passage of time
  - Timed ReSpecT for time-aware coordination policies

Agent Communication Languages (ACL)

Speech acts

- Inspired by the work on human communication
- Communication 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

- Our examples, working as standard protocols for information exchange between agents

  KQML: Knowledge Query Manipulation Language
  `http://www.cs.umbc.edu/kqml/` [Labrou and Finin, 1997]
  FIPA ACL: FIPA Agent Communication Language
  `http://www.fipa.org/repository/aclspecs.html`
  [FIPA ACL, 2002]

Agent Oriented Programming Languages (AOP)

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
- Our examples

  3APL Programming language for cognitive agents
  `http://www.cs.uu.nl/3apl/`
  [Dastani et al., 2004, Dastani et al., 2005]

  Jason Java-based interpreter for an extended version of AgentSpeak(L) for programming BDI agents
  `http://jason.sourceforge.net/`
  [Rao, 1996, Bordini and Hübner, 2006]

Artifact Programming Languages: Coordination

Languages to program social / environment artifacts

- Our example: ReSpecT
  - Programming language for cognitive agents
    `http://respect.alice.unibo.it/`
    [Omicini, 2006, Omicini and Denti, 2001]
- Tuple centres as coordinative artifacts
  - programmable tuple spaces
  - encapsulating coordination policies
- Logic tuple centres as awareness artifacts
- ReSpecT tuple centres as social artifacts
  - 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

Artifact Programming Languages: Organisation / Security

Languages to program individual artifacts

- Our example: Agent Coordination Context (ACC)
  - individual artifact
  - associated to each individual agent in a MAS
  - filtering every interaction of its associated agent
- RBAC-MAS as the organisational model [Omicini et al., 2006]
- Languages for policy specification & enaction
  - logic-based [Ricci et al., 2006a]
  - process algebra [Omicini et al., 2005a]

Non-Agent Programming Languages

Building the agent abstraction layer

- Our examples

  Prolog programming logic agents in Prolog

  Java programming simple agents in Java: examples in TuCSoN

Agents using artifacts

- Our examples

  tuProlog logic agents using ReSpecT tuple centres: examples in tuProlog
  `http://tuprolog.alice.unibo.it/` [Denti et al., 2005]

  simpA extending Java towards A&A agents & artifacts: examples in simpA
  `http://www.alice.unibo.it:16080/projects/simpa/`

  Java/TuCSoN simple Java agents using TuCSoN tuple centres and ACC

  Jason/CArtAgO Jason agents using CArtAgO artifacts
  `http://www.alice.unibo.it:16080/projects/cartago/`
  [Ricci et al., 2006b, Ricci et al., 2006c]

Bibliography I

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

- Dastani, M., van Riemsdijk, B., Dignum, F., and Meyer, J.-J. C. (2004).
  A programming language for cognitive agents: Goal directed 3APL.
  In Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., editors, *Programming Multi-Agent Systems*, volume 3067 of *Lecture Notes in Computer Science*, pages 111–130. Springer.
  1st International Workshop, PROMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited Papers.

- Dastani, M., van Riemsdijk, B., and Meyer, J.-J. C. (2005).
  Programming multi-agent systems in 3APL.
  In Bordini, R. P., Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 39–67. Springer.

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

Bibliography II

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

Labrou, Y. and Finin, T. (1997).
Semantics and conversations for an agent communication language.
In Huhns, M. N. and Singh, M. P., editors, *Readings in Agents*, pages 235–242.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Omicini, A. (2006).
Formal ReSpecT in the A&A perspective.
In Canal, C. and Viroli, M., editors, *5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06)*, pages 93–115, CONCUR 2006, Bonn, Germany. University of Málaga, Spain.
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 III

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.

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. (2006).
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 IV

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

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.

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 V

Ricci, A., Viroli, M., and Omicini, A. (2006b).
Construenda est CArtAgO: Toward an infrastructure for artifacts in MAS.
In Trappl, R., editor, *Cybernetics and Systems 2006*, volume 2, pages 569–574, Vienna, Austria. 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.

Ricci, A., Viroli, M., and Omicini, A. (2006c).
CArtAgO: An infrastructure for engineering computational environments in MAS.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2006)*, pages 102–119, AAMAS 2006, Hakodate, Japan.

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

Programming Languages for Multiagent Systems

Multiagent Systems LS

Sistemi Multiagente LS

Andrea Omicini
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008