Programming Languages for 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 2012/2013
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
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
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

\(^a\)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
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
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 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]
  - other agent architectures could in principle be supported
- 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

<table>
<thead>
<tr>
<th>Agent computation vs. agent interaction / coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Agents have both an internal behaviour and an observable, external behaviour</td>
</tr>
<tr>
<td>- this reproduce the “computation vs. interaction / coordination” dichotomy of standard programming languages</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>- so, what is new here?</td>
</tr>
<tr>
<td>- Agent autonomy is new</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
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
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 and power
  - typically based on the notion of role
  - requiring a model of communicative & pragmatisical actions
  - e.g. RBAC-MAS [Omicini et al., 2005]
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?
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
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 compute and languages to interact?
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
  - *intra-artifact* languages, to compute within artifacts, and
  - *inter-artifact* languages, to interact with agents and other artifacts
Intra-artifact languages

- 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

Inter-artifact languages

- 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 cognitional 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, ...
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
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
[Molesini et al., 2006]

- 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
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
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
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, 2007]
  - 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: Situated/Timed Tuple Centres
  - [Omicini et al., 2007, Casadei and Omicini, 2009]
    - coordination abstractions reactive to passing of time and external events
    - Timed ReSpecT for time-aware coordination policies
    - Situated ReSpecT for environment-related coordination policies
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
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
Our examples

- 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]
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
Agent Oriented Programming Languages (AOP) II

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, 2007, 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., 2005]
Non-Agent Programming Languages I

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 [Denti et al., 2005]
  - **simpA** extending Java towards A&A agents & artifacts: examples in simpA
  - **Java/TuCSoN** simple Java agents using TuCSoN tuple centres and ACC
  - **Jason/CArtAgO** Jason agents using CArtAgO artifacts [Ricci et al., 2006b, Ricci et al., 2007]
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


In Müller, J. P. and Zambonelli, F., editors, Agent-Oriented Software Engineering VI, volume 3950 of LNCS, pages 81–93. Springer.

5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA’06), CONCUR’06, Bonn, Germany, 31 August 2006. Post-proceedings.


Web Intelligence and Agent Systems, 5(2):161–175.


CArtAgO: A framework for prototyping artifact-based environments in MAS.

Objective Coordination in Multi-Agent System Engineering. Design and Implementation, volume 2039 of LNCS.
Springer.