Agent-Oriented Programming for Developing Modern Software Systems

Andrea Santi
a.santi@unibo.it

Alma Mater Studiorum—Università di Bologna
Ph.D in Electronics, Computer Science and Telecommunications\textsuperscript{1}- XXV Ciclo

Second Year Ending Seminar
21/10/2011
# Outline

1. **Introduction**
   - Context
   - Objectives
   - Method

2. **Multi-Agent Oriented Programming**
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3. **The simpAL Project**
   - Background Metaphor
   - Programming Model

4. **Conclusions**
# Outline

## 1 Introduction
- Context
- Objectives
- Method

## 2 Multi-Agent Oriented Programming
- Context
- The JaCa platform
- The JaCaMo Framework
- Weaknesses and Limitations

## 3 The simpAL Project
- Background Metaphor
- Programming Model

## 4 Conclusions
Developing Modern Software Systems: Challenges

- Developing modern software systems is an hard task

- Almost every modern application has to deal with issues such as
  - Concurrency
  - Distribution
  - Decentralisation of control
  - ...

- The free lunch is over [Sutter and Larus, 2005]

- Towards a paradigm change in computer science [Zambonelli, 2004]
How to Program Such Systems?

- It is now important to introduce higher-level programming abstractions [Sutter and Larus, 2005]
  - Easing the development of modern software systems
  - Like OO abstractions help(ed) build large component-based programs

- Proliferation of new programming languages/frameworks aiming at ease this task
  - Rooted on not so mainstream (yet) programming models
    - The Actor [Hewitt et al., 1973] model one is a main example
  - Scala [Wampler and Payne, 2009], Groovy [König and Glover, 2007], Clojure [Hickey, 2011], etc.
The idea of Agent-Oriented Programming is not new
- The first paper about AOP is dated 1993 [Shoham, 1993]..
- ... and since then many APLs and platforms have been proposed [Bordini et al., 2005, Bordini et al., 2009, Bordini et al., 2006]

Main acceptations are the (D)AI contexts
- Agents as a special purpose technique to build intelligent systems

No significant impacts on mainstream research in programming languages and software development

Emphasis put on theoretical issues

No focus on principles of general-purpose computer programming
Our Perspective

AOP can be exploited for programming modern software systems in general [Ricci and Santi, 2011a]

- Extending object/function-oriented programming
  - Thanks to an higher-level of abstraction

- Tackling main challenges of modern software development
  - Concurrency
  - Decentralization of control
  - Autonomy
  - Adaptivity
  - ...

Outline

1. Introduction
   - Context
   - Objectives
   - Method

2. Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3. The simpAL Project
   - Background Metaphor
   - Programming Model

4. Conclusions
Objectives

Background Objective

Explore AOP as a high-level general purpose programming paradigm providing a set of abstractions meant to simplify the design of concurrent, distributed, reactive/interactive programs.

In particular we are interested in

- Identify the essential concepts and features of the paradigm
  - Focussing on principles of general-purpose computer programming

- Inject/re-frame modern programming languages features in AOP
  - Typing, reuse, inheritance, etc.

- AOP’s impact on the set of programming support tools
  - IDE, debuggers, profilers, etc.
Outline

1 Introduction
   - Context
   - Objectives
   - Method

2 Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3 The simpAL Project
   - Background Metaphor
   - Programming Model

4 Conclusions
How Achieve the Stated Objectives?

- A container project named simpAL [Ricci and Santi, 2011b]

- Concerning the design and development of
  - A new generation agent-oriented programming language (i.e. simpAL)
  - Related platforms / infrastructures / tools

- Starting from the construction of a solid background
  - Studying state of the art of
    - Agent architectures
    - Agent programming languages, programming models
  - Making contributions in the context of Multi-Agent Oriented Programming
Outline

1. Introduction
   - Context
   - Objectives
   - Method

2. Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3. The simpAL Project
   - Background Metaphor
   - Programming Model

4. Conclusions
Outline

1 Introduction
   - Context
   - Objectives
   - Method

2 Multi-Agent Oriented Programming
   - Context
     - The JaCa platform
     - The JaCaMo Framework
     - Weaknesses and Limitations

3 The simpAL Project
   - Background Metaphor
   - Programming Model

4 Conclusions
Objectives

- Investigating the adoption of state of the art agent-oriented technologies for programming Multi-Agent System (MAS)

- Focus on
  - Agent-Oriented Programming Languages (APLs)
  - Agent-based frameworks

- Aiming at
  - Construct a background knowledge on state of the art
    - Agent architectures
    - APLs, programming models
  - Immediately applying agent-based technologies in relevant domains
  - Improving current agent-based technologies
The Belief Desire Intention (BDI) Agent Model

Introduced by Rao and Georgeff [Rao and Georgeff, 1995]

Reference model for the development of rational/intelligent agents

The BDI model promotes agent design rooted upon:

- **Belief** knowledge the agent has about the world and other agents
- **Desire** all the possible objectives the agent *might like* to accomplish
- **Intention** the set of objectives the agent *has decided* to work towards
Agents’ Control Architecture
Outline

1. Introduction
   - Context
   - Objectives
   - Method

2. Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3. The simpAL Project
   - Background Metaphor
   - Programming Model

4. Conclusions
The JaCa platform

- **Our base reference platform/infrastructure**
  - For this macro-research line

- **Born from the synergistic integration of**
  - *Jason* [Bordini et al., 2007]
    - A BDI-based agent-oriented programming language
  
  - CArtAgO [Ricci et al., 2009]
    - A framework for building agents' working environments
    - [http://cartago.sourceforge.net](http://cartago.sourceforge.net)

- **Applied in several application contexts**
  - Mobile [Santi et al., 2011]
  - SOA/WS [Piunti et al., 2009]
  - Ambient-Intelligence [Sorici et al., 2011]
Figure: Two different kinds of SMS notifications: (a) notification performed using the standard Android status bar, and (b) notification performed using the ViewerArtifact.
JaCa-Android in Action: SMS Notification Manager 2/2

00  !init.
01
02  +!init
03  <- focus("SMSArtifact");
04    focus("NotificationManager");
05    focus("ViewerArtifact").
06
07  +sms_received(Source, Message)
08    : not (state("running") & session(Source))
09    <- showNotification("jaca.android:drawable/notification",
10       Source, Message, "jaca.android.sms.SmsViewer", Id);
11    append(Source, Message).
12
13  +sms_received(Source, Message) : state("running") & session(Source)
14    <- append(Source, Message).
Outline

1 Introduction
   - Context
   - Objectives
   - Method

2 Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3 The simpAL Project
   - Background Metaphor
   - Programming Model

4 Conclusions
The JaCaMo Framework

- A comprehensive multi-agent oriented programming approach [Boissier et al., 2012]

- Taking into account three different programming dimensions
  - Agent, environment, and organization dimensions

- Built upon the synergistic integration of
  - Jason [Bordini et al., 2007]
  - CArtAgO [Ricci et al., 2009]
  - MOISE [Hübner et al., 2007]

- Defining semantic links among concepts of the different programming dimensions at the meta-model and programming levels

- Providing a uniform programming model for MAS programming
Outline

1. Introduction
   - Context
   - Objectives
   - Method

2. Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3. The simpAL Project
   - Background Metaphor
   - Programming Model

4. Conclusions
Weaknesses and Limitations

- Weak support for modularise active behaviour
  - A main issue: absence of hierarchical structure for plans

- Still no *explicit* notion of type
  - Neither for agents nor for artifacts

- Lack of a seamless integration with object/functional-oriented layer
  - We are currently using a custom library
Outline

1. Introduction
   - Context
   - Objectives
   - Method

2. Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3. The simpAL Project
   - Background Metaphor
   - Programming Model

4. Conclusions
Outline

1 Introduction
   - Context
   - Objectives
   - Method

2 Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3 The simpAL Project
   - Background Metaphor
   - Programming Model

4 Conclusions
Abstractions and Metaphors for Computer Programming

- **OOP metaphor: real-world objects**
  - Objects have properties and states
  - Objects can communicate and respond to communications

- **Actors metaphor: anthropomorphic inspiration**
  - A set of light-weight processes (actors)
  - Interacting only via asynchronous message-passing

- **Our Agent-Oriented Programming metaphor**
  - Anthropomorphic inspiration too
  - A&A meta-model: mimicking human cooperative work environments from Activity Theory [Nardi, 1996] and Distributed Cognition
  - BDI Agent Model [Rao and Georgeff, 1995]
Background Metaphor: an Abstract Representation
Outline

1 Introduction
   • Context
   • Objectives
   • Method

2 Multi-Agent Oriented Programming
   • Context
   • The JaCa platform
   • The JaCaMo Framework
   • Weaknesses and Limitations

3 The simpAL Project
   • Background Metaphor
   • Programming Model

4 Conclusions
Agents

- Autonomous, state-full task- and event-driven entity
- Based on a BDI architecture
- Creating and co-using artifacts for supporting their activities
  - besides direct communication

Autonomous behaviour structured in terms of

- Role
  - The set of tasks an agent must be able to do inside an organisation
- Tasks
  - Unit of work that has to be executed
  - Defining what have to be executed
- Plans
  - Actions to be performed for achieving a task
  - Defining how to execute a task
- Agent Script
  - Collection of concrete plans for playing a role
Working Environments

- Structured in terms of multiple (distributed) workspace
- Provide agents the tools and resource they need by means of artifacts
- Artifacts are non-autonomous function-oriented entities
  - Controllable and observable by agents
  - Functionalities provided by artifact operations
  - State exposed by means of observable properties
An Agent Role Example

```java
role MyRole extends BasicRole {
    task MyTask {
        aParam: int; // primitive type
        myList: List<String>; // an object
        myCounterTool: CounterInterface; // an artifact
    }
    task AnotherTask { ... }
}
```

**Figure:** An example of role definition in simpAL
Agent Script

Defines *how* play one or more roles by means of concrete plans

```java
script MyScript implements MyRole {
    st: String;
    myTool: MyTool;

    plan MyPlanA
        task: MyTask
        context: (aParam > 0 && st.equals("test")) {...}

    plan MyPlanB
        task: MyTask
        context: aParam <= 0
        using: myCounterTool, myTool {...}

    plan MyPlanC
        task: AnotherTask {...}

    ...
}
```

*Figure:* An example of agent script definition in simpAL
Mixed Reactive and Proactive Behavior

- A plan body is composed by a set of action rules
  - Event : Context ⇒ Action [label]

- Easily mixing of reactive/proactive behavior
  - Combining Event and Context attributes

```plaintext
plan MixedPlan task: MyTask {
  myBel : String;
  myTool->doAction1();
  myTool->doAction2();
  myTool->doAction3();
  ...
  +critical_condition : myBel == "state1" ⇒ doRepairAction1()
  +critical_condition : myBel == "state2" ⇒ doRepairAction2()
  +new_info : true ⇒ println("New info retrieved" + new_info)
}
```

**Figure:** An example of mixed reactive and proactive behavior in simpAL
simpAL Type System

Agent Side

- Agent type ↔ role(s) it implements
  - Set of tasks the agent is able to perform
- Agent role: the *contract* identifying a certain set of tasks
  - Analogous to the interface for OOP
  - Basic unit for agent sub-typing/hierarchies relations
  - An agent can play/implement multiple roles

Artifact Side

- Artifact type ↔ its artifact model (usage interface), including
  - Operations
  - The set of its observable properties
- Sub-typing and inheritance rooted on the notion of Artifact type
Inheritance, Polymorphism and Error Checking

**Inheritance**
We can define inheritance relations for what concerns
- Agent roles/scripts
- Artifact models and concrete implementations

**Polymorphism**
We are able to support polymorphism at different levels
- Role level
  - Agents implementing the same role in different ways
- Agent script level
  - Tasks performed in different ways on the base of contextual conditions
- Artifact and artifact model level

**Error Checking**
We are able to support error checking at compile time
- Agents implementation compliant to the roles it implements
- Artifact definition must adhere to its model specification
- ...
Outline

1. Introduction
   - Context
   - Objectives
   - Method

2. Multi-Agent Oriented Programming
   - Context
   - The JaCa platform
   - The JaCaMo Framework
   - Weaknesses and Limitations

3. The simpAL Project
   - Background Metaphor
   - Programming Model

4. Conclusions
Future Work

- Main efforts to be put on finalizing/improving the simpAL project
  - From the language perspective
    - Shape the basic set of features of the language
    - Finalize and formalize the type system
  - From the simpAL platform perspective
    - IDEs, debuggers, profilers, etc.
    - Finalize the work on the distributed Runtime Infrastructure

- Test simpAL effectiveness for developing modern software systems
  - In relevant application domains
  - Pointing out experienced outcomes, benefits and limitations

- Continue the exploration with the JaCaMo framework
  - As a reference multi-agent oriented programming approach for developing MAS
Conclusions

AGERE!@SPLASH

AGERE! Programming Systems, Languages, and Applications based on Actors, Agents, and Decentralized Control

- International workshop we organised to discuss our ideas in practice
- Held at the ACM conference SPLASH 2011
- Aiming at investigating Actors and Agents as post-OO programming paradigm

Bibliography II


*Groovy in action.*
Manning Pubs Co Series. Manning.

Activity theory and human-computer interaction.
*Context and consciousness: Activity theory and human-computer interaction,* pages 7–16.

Programming SOA/WS systems with BDI agents and artifact-based environments.
In *Proceedings of MALLOW-AWESOME Joint Workshop on Agents, Web Services and Ontologies, Integrated Methodologies.*
Bibliography V

BDI Agents: From Theory to Practice.
In First International Conference on Multi Agent Systems (ICMAS95).

Turtles, Termites and Traffic Jams. Explorations in Massively Parallel Microworlds.
MIT Press.

Environment programming in CArtAgO.
In Bordini, R. H., Dastani, M., Dix, J., and El Fallah-Seghrouchni, A.,


Conclusions

Considerations

- Decentralized mindset in programming [Resnick, 1994] thanks to
  - Encapsulation of control (into agents)
  - An interaction dimension rooted on two interaction typologies
    - Asynchronous message passing
    - Environment-mediated interactions

- An high-level rationale to structure autonomous behaviour
  - Through the notion of roles, tasks and plans

- Integration of pro-active and reactive behaviour
  - Thanks to the agent control architecture

- Avoidance of "everything is an X" perspective
  - Agents to model active parts, artifacts for passive ones
Inheritance: Notable Cases

- **R1** can be defined as an extension of role **R**
  - Inheriting all tasks defined in **R**
  - Adding/specializing tasks defined in **R**

- **Ag1** can be defined as an extension of an agent script **Ag**
  - Inheriting all plans defined in **Ag**
  - Adding further plans either to achieve
    - Task defined in the role implemented by **Ag**
    - New tasks defined in a role implemented by **Ag1** (and not by **Ag**)

- Similar inheritance relations can be defined also for the environment dimension
Error Checking and Polymorphism: Notable Cases

Error Checking

- An agent script must define at least one plan for each task defined in the role it implements
- Artifact definition must adhere to its model specification
- Duplicated constructs checks
- ...

Polymorphism

- The same task can be performed by different agents implementing the same role in different ways
  - Depending on the scripts that they are using
- Task assigned to an agent can be performed in a different way
  - Depending on current contextual conditions
- Artifacts can provide different implementations for the same model
Integration with Object-Oriented/Functional programming

- Agent and environment are coarse grain abstractions
  - Defining the high-level organization of the system
  - Not useful for defining
    - Data structures
    - Computational parts manipulating them

- OOP/Functional languages to define data structures and functions

- simpAL integration with OO programming languages is quite straightforward
  - Thanks to a clear separation between the agents/environment layer and the objects/functions one