Agent-Oriented Programming as a Paradigm for Developing Modern Software Systems

Andrea Santi\textsuperscript{1}, Alessandro Ricci\textsuperscript{1}

a.santi@unibo.it, a.ricci@unibo.it

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

Cesena, 29/05/2012
1. About me

2. Introduction

3. Background Metaphor

4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations

5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
Who is Andrea Santi?

- I’m a 26 years old Italyn Ph.D student
  - http://www.alice.unibo.it/xwiki/bin/view/AndreaSanti/

- Education information:
  - Master degree in Computer Science Engineering

- My department
  - DEIS Università di Bologna ...
    - .. but will soon be DISI

- Where I actually work:
  - apiCE laboratory, Cesena

- My supervisor:
  - Alessandro Ricci
  - http://www.alice.unibo.it/xwiki/bin/view/AlessandroRicci/
Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
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]
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.
Agent-Oriented Programming (AOP): the Current Situation

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

- 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
  - ...

Our Perspective
Main Research Lines

Two main ongoing research lines

- Exploiting agent-oriented abstractions to develop real-world programs
  - Stressing existing technologies:
    - JaCa platform [Ricci and Santi, 2011b] & friends
    - JaCaMo framework [Boissier et al., 2012]
  - Applying them in modern application domains
    - Pointing out: outcomes, weaknesses ...

- Devising a new language – simpAL [Ricci and Santi, 2011a]
  - Focus on principles of general-purpose computer programming
  - Injecting/re-framing modern programming languages features in AOP
  - Supported by a set of adequate tools
    - IDE, debuggers, profilers, etc.
Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
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
Multi-Agent Oriented Programming

Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work

A. Santi (DEIS)  Agent-Oriented Programming (AOP)  29/05/2012  14 / 82
Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
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
    - http://jason.sourceforge.net/
  - CArtAgO [Ricci et al., 2009]
    - A framework for building agents’ working environments

- Applied in several application contexts
  - Mobile [Santi et al., 2011]
  - SOA/WS [Piunti et al., 2009]
  - Ambient-Intelligence [Sorici et al., 2011]
  - ...
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
The JaCa Programming Model in a Glance

Applications realized in CArtAgO [Ricci et al., 2009] working environments where a set of Jason [Bordini et al., 2007] agents work together and interact creating, sharing and exploiting a dynamic set of artifacts.

- **BDI Jason Agents** encapsulate the execution and the control of tasks that characterize the application scenario.
- **Working environments** used as a first class abstraction:
  - Encapsulating the resources and tools needed by agents
  - Allowing the design of a world aimed at the agent’s use
A JaCa Application Abstract Representation

MAS

ENVIRONMENT

Sphere of visibility and influence

WORK ENVIRONMENT

Sphere of visibility and influence
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 promote agent designed 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
Working Environments

- A first-class concept

- A software/computational layer
  - Defining the context of agents tasks
  - Providing and encapsulating (by means of artifacts)
    - Resources and services
    - State and processes
    - Ruling and mediating function

- Properly designed by the MAS engineer that defines
  - The structure and the topology of the environment
    - Organised in term of workspaces
  - And its desired behaviour
Artifacts in a Nutshell

Figure: [Ricci et al., 2009].
Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
The JaCa-Android Platform

- Proper porting of the standard JaCa platform in the Android context
- Open-source project
  - http://jaca-android.sourceforge.net/
- AOP for tackling new challenges in mobile application development
  - Concurrency
  - Asynchronous interactions
    - Web sites/Services, social-networks, messaging/mail clients, etc.
  - Context-sensitive and user-centric behavior
    - Geographical position, presence/absence of connectivity..
- Seamless integration with existing Android applications
The JaCa-Android Platform: an Abstract Representation

JaCa (Jason+CArtAgO)

Android Framework
(Dalvik Virtual Machine + Libraries)

Linux kernel

JaCa Android artifacts

SMSManager
Calendar
GPSArtifact
ActivityGUI
MyArtifact

JaCa-Android app

JaCa-services
shared workspace
Figure: Two different kinds of SMS notifications: (a) notification performed using the standard Android status bar, and (b) notification performed using the ViewerArtifact.
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).
Possible Projects Concerning JaCa-Android

- **Platform improvements**
  - Realization of a general layer for interacting with external services
  - Integration with interesting external services (e.g. Google Calendar, Maps, etc.)
  - Refactorization and optimizations

- **Realization of interesting mobile applications**
  - Starting from a interesting idea you can have
  - Aiming at showing the benefits of the programming model adopted

- **Evaluate the JaCa-Android programming approach**
  - Realizing an application of interest both in JaCa-Android and Android
  - Comparing the two applications, pointing out differences, benefits, drawbacks
JaCa-Androbot 1/2

JaCa-Androbot

JaCa-Android

Middleware
Android

Android

Bluetooth

Middleware
LeJOS

LeJOS
JaCa-Androbot 2/2

![Robot Image]
JaCa-Fly
Possible Projects Concerning JaCa-Androbot and JaCa-Fly

Background Objective

Investigate the application of the agent paradigm (i.e. the JaCa approach) for programming simple robots

- Exploring higher-level abstraction to program their behavior
- With a software engineering perspective

Programming Robots in JaCa For Doing Some Interesting Task

1. Identification of scenarios/tasks of interest
   - E.g. an ongoing project studying QR-based navigation for the drone
2. Study of the related literature for the chosen scenario/task
3. Engineer the JaCa-based solution
4. Critical analysis of the obtained results
   - Benefits?
   - Weaknesses/limitations?
Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
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. About me

2. Introduction

3. Background Metaphor

4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations

5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
Exploring programming paradigms based on agent-oriented abstractions as evolution/specialization of mainstream ones

- Inspiration from ideas, models already introduced in AOP in D(AI)
  - BDI Agent Model [Rao and Georgeff, 1995]
  - A&A meta-model [Omicini et al., 2008]

- Exploring features that are essential in programming & sw engineering
  - Ignored in (D)AI

- Providing a seamless integration with existing mainstream paradigm
  - OO in particular
simpAL Project: Method

- How Achieve the Stated Objectives?
- simpAL Programming Language and...
- ... related tool/technology ecosystem
- Past/present related experiences
  - simpA agent-oriented Java-based Framework [Ricci et al., 2008]
  - JaCa & JaCaMo agent platform just presented
Background Metaphor: an Abstract Representation
Abstract Representation of a simpAL Program

- **Agents**
- **Organisation**
- **Workspaces A and B**
- **Artifacts**

**Operations:**
- Use
- Observe
- Communicate with

**Arrows:**
- Solid lines for use
- Dashed lines for observe
- Dash-dotted lines for communicate with
Main Coarse-Grain First-Class Abstractions

- **Agents**
  - Autonomous task-oriented + even-driven components
  - Computational model inspired to BDI
  - Communicate with other agents with speech acts
  - Use & observe the shared environment

- **Artifacts**
  - Non-autonomous entities, building blocks of the environment
  - Resources/tools created, shared and used by agents to do their tasks
    - A blackboard, a web page, a GUI, ...

- **Workspaces**
  - Logical containers of artifacts
  - Defining the topology of the (distributed) environments
And Objects?

- OOP Layer
  - Representing every data structure & related (transformational) computations

- All the data used inside agents, artifacts are represented in terms of OO abstractions

- Java model/language adopted in simpAL
Outline

1. About me
2. Introduction
3. Background Metaphor
4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations
5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
Beliefs

- Representing what the agent knows about
  - Itself (≈ private instance fields of an object)
  - About the observable state of the environment

- Variable-like typed information items

- Collected into the belief-base
Task as a First-Class Concept

- Pervasive concept used at the design level
- Specifying what job(s)/goal(s)/work to be performed by the agent
  - Can be long term
- To explicitly specify & structure the pro-active behaviour

- to-do: T2, T3, ...
- doing: T0, T1, ...
Task Taste in simpAL

```plaintext
task AchieveAndKeepTemperature {
    targetTemp: double;
}

task FindFiles {
    content: Content;
    fileList: List<File>;
}
```

Figure: Examples of simple tasks definition in simpAL.
Task Assignment

- Internal agent actions (self-assignment)
- Communicative action by other agents: do-task T

Dynamic Assignment

- do-task T4
- to-do: T2, T3, T4
- doing: T0, T1,...
Communicative Actions 1/2

- To assign tasks (do-task)
- To inform (tell)
- To ask for a belief (ask)
Communicative Actions 2/2

```plaintext
task AchieveAndKeepTemperature {
    targetTemp: double;
    understands {
        stop: boolean;
    }
}

(task FindFiles {
    content: Content;
    fileList: List<File>;
    understands {
        maxSize: double;
    }
}

Figure: Declaration of understandable messages for agents playing a role.
```
Grouping Set of Related Tasks: The Role Concept

role MemoryManager extends BasicRole {

  task AllocateMemory {
    blockSize : int;
    ...
  }

  task CollectGarbage {
    ...
  }

  ...

  Set of tasks an agent implementing the role must be able to perform
  Modularizing the overall set of tasks
Typing Agents with Roles and Tasks

- Roles allow for introducing a notion of type for agents
  - Based on the tasks that they are able to do
  - Defining role hierarchies

- Two main outcomes
  - Error checking at compile time
    - An agent implementing a role $R$ can be assigned only of those tasks enlisted in $R$
  - Principle of substitutability
    - An agent playing the role $R$ can be substituted by an agent by any agent playing role $R$ or an extension $R'$
Plans as a First-Class Concept

- Plans specifies how to accomplish tasks
  - Modules of procedural knowledge (which actions to do and when)
  - At runtime collected in the agent plan library
- For the same tasks we can have multiple plans that can be used in different contexts

```
PLAN LIB:
plan P0 task: T0 {...}
plan P1 task: T1 {...}
...
to-do: T2, T3, ...
doing: T0, T1, ...
```
Plan Example in simpAL

```plaintext
plan p0 task: AchieveTemperature
  context: currenTemp < targetTemp
  using: myThermometer {
    ...
  }

plan p1 task: AchieveTemperature
  context: currenTemp > targetTemp
  using: myThermometer {
    ...
  }
```

**Figure**: Examples of simple plans definition in simpAL.
Grouping Plans: AgentScripts

- Encapsulating set of plans and beliefs that can be used to implement one or multiple roles
  - “Learnt” (fetched/compiled/executed) by agents at runtime
  - The agent “program”, finally

PLAN LIB:
plan P0 task: T0 {...}
plan P1 task: T1 {...}
...
Script taste in simpAL

```plaintext
script MyScript implements MyRole {
    /* long-term beliefs */
    value: double;
    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: Examples of a simple role definition in simpAL.
Error Checking & Reuse with Scripts

- Error checking
  - By implementing a role $R$, a script $S$ must have at least one plan for each task enlisted in $R$
  - ...

- Reuse & refinement
  - Scripts can be defined by extending existing scripts
  - Inheriting/refining the plans
The Artifact Abstraction

- To design & implement non-autonomous components
- Resources & tools from agents’ viewpoint
- Encapsulating some functionality/service
- Modularizing the agent environment
Artifacts Core Features

- Encapsulating
  - Operations ⇔ actions available to agents
  - Observable Properties ⇔ observable state of that environment block

- Can be used/observed concurrently by multiple agents
  - No need of any specific low-level synch mechanism
  - Operations with native transaction-like features
  - No control coupling (they are not *shared objects*)

- Modularizing the agent environment
Artifact Models & Templates 1/2

Figure: Example of an Artifact Model declaration and a related Artifact Template implementation.
Artifact Interfaces & Templates 2/2

- Typing artifact with artifacts models
  - Error-checking (also in agent scripts)
  - Principle of substitutability
    - For artifacts of the same or extended model

- Reuse/Refinement of the implementation of artifacts
  - Defining artifact templates from existing ones
Back to Agent Plans: The Plan Body

- Plan body as a set of
  - Local beliefs declaration (\(\sim\) like local variables)
  - Action rules: specifying which actions to execute and when
    - For artifacts of the same or extended model

Action rule model based on ECA rule

**Event**: Context \(\Rightarrow\) **Action** [label]

- **Event**: change of a belief (about env, about actions, ...)
- **Context**: condition over the belief base
- **Action**: internal + external (artifact operations)
Integrating Pro-Active & Reactive Behaviour in Plans 1/3

\begin{verbatim}
plan MyPlan task: MyTask using: counter@wsp1{
  inc
  inc
  println (msg: "done")
}
\end{verbatim}

**Figure**: Examples of a plan with a pure sequential behaviour.
plan MyPlan task: MyTask using: counter@wsp1{

updated countValue : countValue >= 0
=> println (msg:"done")

updated countValue : countValue < 0{
  action1
  action2
}
}

**Figure**: Examples of a plan with a pure reactive behaviour.
plan MyPlan task: Counting using: counter@wsp1 {
  inc
  inc
  inc
  updated countValue => exec: atomically {
    println (msg:"new value"+countValue)
  }
}

Figure: Examples of a plan with a mixed pro-active and reactive behaviour.
Agent Control Architecture

- Events flow through the Ext. Event queue.
- Int. Event Queue feeds into the SENSE stage.
- SENSE stage connects to the Belief Base and Plan Library.
- PLAN stage receives information from the Int. Event Queue and Ongoing Plans/Tasks.
- ACT stage is influenced by Actuators and Ongoing Plans/Tasks.
- Logical clock synchronizes the decision-making process.
Organisation Abstraction

```
orgmodel MyOrgModel {
    workspace w1 {
        console: Console;
        c1: Counter;
        agent0: MyRoleA;
    }
    workspace w2 {
        console: Console;
        bb: Blackboard;
        agent1: MyRoleB;
    }
}
```

```
org MyOrg implements MyOrgModel {
    workspace w1 {
        c1 Counter startValue: 10;
        agent0
            script: MyScript
            task: MyTask(aParam: 10);
    }

    ...
}
```

**Figure:** Example of an OrgModel specification and a related Org implementation.
Distributed runtime infrastructure

Interactive programming environment
  - Integrating editor, compiler, launcher, debugger
  - Currently based on Eclipse ecosystem (Xtext...)

Libraries
  - System libraries + utility libraries
The simpAL Project
Programming Model Overview

simpAL Platform 2/3

A. Santi (DEIS)
Agent-Oriented Programming (AOP)
Outline

1. About me

2. Introduction

3. Background Metaphor

4. Multi-Agent Oriented Programming
   - Reference Platforms
   - The JaCa Platform in Relevant Application Domains
   - Weaknesses and Limitations

5. The simpAL Project
   - Programming Model Overview
   - Ongoing Work
Typing Extensions

Agents implementing multiple roles

- An agent $\text{Ag}$ implementing role $\text{R1}$ and $\text{R2}$ is of type $\text{T}$
  - Containing all the tasks defined in $\text{R1}$ and $\text{R2}$
  - Disambiguation among tasks with the same name
    - e.g. $\text{R1.MyTask}$ and $\text{R2.MyTask}$

Artifacts implementing multiple artifact models

- An artifact $\text{Art}$ implementing the model $\text{M1}$ and $\text{M2}$ is of type $\text{T}$
  - Containing all the operations/obs properties defined in $\text{M1}$ and $\text{M2}$
  - Disambiguation for constructs with the same name
    - e.g. $\text{M1.MyOp}$ and $\text{M2.MyOp}$
    - e.g. $\text{M1.MyObsProp}$ and $\text{M2.MyObsProp}$
Studying Inheritance 1/2

- An ArtifactModel `ArtMod' extending the ArtifactModel `ArtMod'
  - What does it inherits from `ArtMod`?
  - What it can extended from `ArtMod`?

- An Artifact `Art' extending the ArtifactModel `Art'
  - What does it inherits from `Art`?
  - What it can extended from `Art`?

- A Role `R' extending the Role `R'
  - What does it inherits from `R`?
  - What it can extended from `R`?

- An AgentScript `Ag' extending the AgentScript `Ag'
  - What does it inherits from `Ag`?
  - What it can extended from `Ag`?
An OrgModel $\text{OrgMod}'$ extending the OrgModel $\text{OrgMod}$
- What does it inherits from $\text{OrgMod}$?
- What it can extended from $\text{OrgMod}$?

An Organisation $\text{Org}'$ extending the Org $\text{Org}$
- What does it inherits from $\text{OrgMod}$?
- What it can extended from $\text{OrgMod}$?
simpAL Platform Extension

- IDE improvements
  - Specialised views
  - Context assist
  - ...

- Debugger improvements
  - For an easy inspection of simpAL programs

- Runtime platform improvements
  - For easy deployment of applications in distributed simpAL node
Possible Projects Concerning simpAL

- Concerning the compiler
  - Optimizations and refactoring
  - Move from Java to Xtend?

- Concerning the runtime
  - Improve the support for *seamlessly*
    - Deploy/execute/terminate distributed applications
  - Proper tool for inspect/debug distributed applications
  - Optimizations and refactoring

- Concerning the IDE
  - Improvements of the IDE features
  - Investigate a full synergy with the Xtext framework
  - Improve the editor synergy with the Eclipse ecosystem

Multi-agent oriented programming with jacamo.
*Science of Computer Programming.*
Special Issue on Agent-oriented Design methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.

A survey of programming languages and platforms for multi-agent systems.
In *Informatica 30*, pages 33–44.
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.

Artifacts in the A&A meta-model for multi-agent systems.
*Autonomous Agents and Multi-Agent Systems, 17 (3).*


Designing a General-Purpose Programming Language based on Agent-Oriented Abstractions: The simpAL Project.
In *Proc. of the 1st Int. Workshop on Agents and Actors Reloaded – AGERE*.

Agent-oriented computing: Agents as a paradigm for computer programming and software development.
In *Proc. of the 3rd Int. Conf. on Future Computational Technologies and Applications (Future Computing ’11)*, Rome, Italy. IARIA.


