Agent-Oriented Software Engineering

PORTO Methodology

AIAD 2013/2014

António Castro and Eugénio Oliveira

Contents

• What is AOSE?
• Main Existing Methodologies
• Tropos
• AUML
• Gaia
• Porto
What is AOSE?

- “A methodological approach for the development of software oriented or based on agents”
  (Giorgini and Henderson-Sellers)

- A methodology has two main components:
  - A Description of approach processes.
  - Documentation and products/artifacts.

- All existing proposals are from the academic world, are not complete (either in processes or artifacts/products) nor applicable to a wider scope of application domains.

Main existing methodologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Characteristics</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>TROPOS</td>
<td>Analysis/Design; Supports Verification/validation</td>
<td>Paolo Giorgini et al.</td>
</tr>
<tr>
<td>MAS-CommonKADS</td>
<td>Analysis/Design; Guidelines for Verification/Validation</td>
<td>Carlos Iglesias et al.</td>
</tr>
<tr>
<td>PASSI</td>
<td>Analysis/Design/Implementation; Supports Verification/validation;</td>
<td>Massimo Cossentino</td>
</tr>
<tr>
<td>PROMETHEUS</td>
<td>Analysis/Design; Supports Verification/validation; QQ # agts.</td>
<td>Lin Padgham et al.</td>
</tr>
<tr>
<td>GAIA</td>
<td>Analysis/Design; Does not Support Verification/validation; &lt;= 100 agts.</td>
<td>Franco Zambonelli et al.</td>
</tr>
<tr>
<td>ADELFÉ</td>
<td>Analysis/Design/Implementation; Supports Verification/validation;</td>
<td>Carole Bernon et al.</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Analysis/Design; Does not Support Verification/validation;</td>
<td>Francisco Garrio et al.</td>
</tr>
<tr>
<td>INGENIAS</td>
<td>Analysis/Design/Implementation; Supports Verification/validation;</td>
<td>Juan Pavón et al.</td>
</tr>
<tr>
<td>PORTO</td>
<td>Analysis/Design/Implementation/Testing/Validation</td>
<td>Antonio Castro et al.</td>
</tr>
</tbody>
</table>
TROPOS


Main Characteristics

• BDI specific
• Develops the system using BDI constructs
• Emphasis on early requirement analysis
Early Requirements Analysis

- Model needs and objectives of various system stakeholders in the human organization
- Model the system environment
- Model interaction of system-to-be with human stakeholders and system environment

Example Actor Diagram

[Diagram showing interactions between Citizen, Museum, Visitor, and PAT]
Example Goal Diagram

Main Concepts

- Use actor to represent agents, roles and positions
- Use goals, plans, resources, dependencies, capabilities and beliefs throughout development
Strengths and Weaknesses

• Strength
  – Good support for early requirement analysis

• Weakness
  – BDI specific, less general
  – Diagrams may not scale

AUML

J. Odell, H. Van Dyke Parunak and B. Bauer
Representing Agent Interaction Protocols in UML
AUML

- Not a methodology
- Standards to represent design of agent systems
- Familiar to industry developers
- Widely adopted

GAIA

F. Zambonelli, N. Jennings, M. Wooldridge
Developing Multiagent Systems: the Gaia Methodology
ACM Transactions on Software Engineering and Methodology, Vol. 12, No. 3, July 2003
Identifies the main services — coherent blocks of activity in which agents will engage — that are required to realize the agent's roles and properties.

Separating, when possible, the organizational independent aspects (detected in analysis) from the organizational dependent ones (derived from the adoption of a specific organizational structure).

The goals of the organizations that constitute the overall system and their expected global behavior.

The rules that the organization should respect and enforce in its global behavior.

In terms of its topology and control regime. Can also exploit catalogues of organizational patterns.

Overview

Analysis Phase

- **Organizations**: determine whether multiple organizations have to coexist in the system and become autonomous interacting MAS.

- **Environmental Model**: abstract computational resources, such as variables or tuples, made available to the agents for sensing (read), for effecting (change) or for consuming (extract)
Analysis Phase

- **Preliminary Role Model**: identify the “basic skills” that are required by the organization to achieve its goals, as well as the basic interactions that are required for the exploitation of these skills.

- **Preliminary Interaction Model**: captures the dependencies and relationships between the various roles in the MAS organization, in terms of one protocol definition for each type of inter role interaction.

**Organizational Rules**: responsibilities of the organization as a whole. These are safety (invariants that must be respected) and liveness (dynamics of the organization) organizational rules.

Fig. 5. Conference management: Multilevel hierarchy.

**Environment Model**
Architectural Design

- **Organizational Structure**: identify the appropriate organizational structure, including, topology and control regime.

- **Organizational Patterns**: catalogue of possible modular and composable "organizational structures" that will help the designer.

**Completion of Role and Interaction Models**: (1) define all the activities in which a role will be involved, (2) define organizational roles, (3) complete the definition of the protocols required by the application and (4) define organizational protocols.
Detailed Design

- **Agent Model**: to define the agent model it is necessary to identify which agent classes are to be defined to play specific roles and how many instances of each class have to be instantiated in the actual system. The model can be defined using a simple diagram (or table) specifying, for each class, which roles will map it. In addition, the model can document the instances of a class that will appear in the MAS.

- **Services Model**: identify the services associated with each agent class, or equivalently, with each of the roles to be played by the agent classes. For each service it is necessary to document its properties: inputs, outputs, preconditions and postconditions. The services are derived from the list of protocols, activities, responsibilities and liveness properties of the roles it implements.
Scopes and Limitations

• Does not directly deal with particular modeling techniques. It proposes but does not commit to, specific techniques for modeling (e.g., roles, environment, interactions). In the future: “... AUM is a useful companion to GAIA.”

• Does not directly deal with implementation issues. The outcome is a detailed but technology-neutral specification. Should be easy to implement with, for example, a FIPA-compliant agent system.

• Does not explicitly deal with activities of requirements capturing and modeling. In the future: “... integrate methods and techniques from goal-oriented analysis.”

Summary of Strengths

• Expressive except does not model knowledge in system

• Architecturally independent, allow diverse technologies

• Simple and easy to learn
Summary of Weaknesses

• Does not cover full life-cycle
• Unfamiliar notation for developers
• Lack of CASE tool and other support

PORTO better than GAIA


PORTO Overview/3

Requirements Analysis
Advantages

- Modelling specifications in terms of actors, their roles, their goals and dependencies, is more similar to the AOCC organisation.
- In subdividing the system: identifying the specific organisations and sub-organisations dedicated to the achievement of a specific goal.
- In preliminary role model: identifying the basic skills (functionalities and competences) required by the organisation to achieve its goals.
- In preliminary interaction model: identifying the basic interactions that are required for the exploitation of the basic skills.

Table 6: RosterCrewMonitor preliminary role

| Role schema: RosterCrewMonitor |
| Description: This preliminary role involves monitoring the crew status for events related to crew members not reporting for duty and/or flights with open positions. After detecting one of these events the RosterCrewMonitor will request a solution from the organisation. It should be able to trace previous requests, avoiding duplicates, and it receives a message regarding the status of the request. |
| Protocols and activities: CheckForNewCrewEvents, UpdateCrewEventStatus |
| Permissions: |
| sends: CrewSignON // to obtain all who did not report for duty |
| sends: Paging // to obtain all flights with open positions |
| Responsibilities: |
| Live: |
| RosterCrewMonitor = (CheckForNewCrewEvents) and (UpdateCrewEventStatus) |
| Safety: |
| • successful_connection_with_CrewSignON = true |
| • successful_connection_with_Paging = true |
| • new_crew_request <> existing_unclaimed_crew_request |

Table 7: Preliminary protocol: informCrewEvents

| Protocol name: informCrewEvents |
| Initiator role(s): RosterCrewMonitor |
| Partner role(s): CrewFind |
| Description: |
| After an event has been detected it is necessary to find an available crew member to fill the open position. For that it is necessary to send details about the open position so that an available crew member might be found. |
| Input: Open position information |
| Output: Yes, I will try to find a solution OR No. I cannot process the request (see safety conditions on CrewFind role). |

Analysis – Preliminary Role & Interaction
Table 5.7 Liveness rules (relations)

Liveness rule or relations

\[ \text{applyCrewSolution}(\text{CrewAssign}(\text{crew}(x))) \rightarrow \text{repCrewStatus}(\text{CrewAssign}(\text{crew}(x))) \]

Protocol `applyCrewSolution` must necessarily be executed by role `CrewAssign` for a specific crew solution `crew(x)` before role `CrewAssign` can execute protocol `repCrewStatus` for that crew solution.

\[ \text{requestCrew}(\text{CrewFind}(\text{request}(x))) \rightarrow \text{applyCrewSolution}(\text{CrewFind}(\text{crew}(x))) \]

Protocol `requestCrew` must necessarily be executed by the role `CrewFind` for a specific request `request(x)` before role `CrewFind` can execute protocol `applyCrewSolution` for the solution found.

Table 5.8 Safety rules (constraints)

<table>
<thead>
<tr>
<th>Safety rules or constraints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>((-\text{(RosterCrewMonitor)}\text{&amp;CrewFind}))</td>
<td>Role <code>RosterCrewMonitor</code> and role <code>CrewFind</code> can never be played concurrently by the same entity.</td>
</tr>
<tr>
<td>((-\text{(RosterCrewMonitor)}\text{&amp;CrewAssign}))</td>
<td>Role <code>RosterCrewMonitor</code> and role <code>CrewAssign</code> can never be played concurrently by the same entity.</td>
</tr>
</tbody>
</table>
Organisational Rules and Structure

Table 5.9 Topologies and control regime

<table>
<thead>
<tr>
<th>Organization</th>
<th>Topology</th>
<th>Control Regime</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOCC</td>
<td>Multilevel hierarchy</td>
<td>Mixed: cooperative and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>authoritative.</td>
</tr>
<tr>
<td>Crew Manager</td>
<td>Multilevel hierarchy</td>
<td>Work specialization.</td>
</tr>
<tr>
<td>Aircraft Manager</td>
<td>Multilevel hierarchy</td>
<td>Work specialization.</td>
</tr>
<tr>
<td>Passenger Manager</td>
<td>Hierarchy</td>
<td>Work specialization.</td>
</tr>
</tbody>
</table>

Organisational Rules and Structure

Table 5.10 Organization structure for passenger recovery

Statement/Comment

∀i, OperationalControlSupervisor \xrightarrow{control} PaxApply [i]

This means that the role OperationalControlSupervisor has an authoritative relationship with role PaxApply, controlling, in this case, all the actions of role PaxApply. Specifically, role PaxApply needs approval from OperationalControlSupervisor before applying the solution. Please note that role OperationalControlSupervisor is shared between this sub-organization and Aircraft Manager sub-organization.

∀i, OperationalControlSupervisor \xrightarrow{depends_on} PaxFind [i]

This means that the role OperationalControlSupervisor relies on resources or knowledge (a solution found to solve a passenger problem) from the role PaxFind to accomplish its task (i.e., to authorize or not authorize the assignment of a specific solution).

∀i, j, PaxFind [i] \xrightarrow{depends_on} PaxMonitor [j]

This means that the role PaxFind relies on resources or knowledge (an event related to a passenger problem) from the role PaxMonitor to accomplish its task (i.e., to find a solution to the passenger problem).
Arch. Design - Combined Representation

Arch. Design - Completing role model

Table 12  RosterCrewMonitor role

Role Schema: RosterCrewMonitor

Description: Monitors the crew roster for events related to crew members not reporting for duty and/or flights with open positions. After detecting one of these events, it will request a solution from the organization. Traces previous requests and avoids duplicates, until it receives a message regarding the status of the request.

Protocols and Activities: CheckForNewCrewEvents, UpdateCrewEventStatus, informCrewEvent, reportCrewEventStatus

Permissions:

reads: CrewSignON // to obtain all who did not report for duty

reads: Pairings // to obtain all flights with open positions.

Create, read, update CrewEvents Class

Responsibilities:

Liveness:

RosterCrewMonitor =

(CheckForNewCrewEvents ∧ informCrewEvent) ∨

(reportCrewEventStatus ∧ UpdateCrewEventStatus)

Safety:

• successful_connection_with_CrewSignON = true
• successful_connection_with_Pairings = true
• successful_connection_with_CrewEvents = true
• new_crew_request <> existing_unclosed_crew_request
Completing interaction model

Table 13 sendCrewSolution protocol definition

<table>
<thead>
<tr>
<th>Protocol name: sendCrewSolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiator role(s): CrewFind</td>
</tr>
<tr>
<td>Partner role(s): OperationsScheduleManager</td>
</tr>
</tbody>
</table>

**Input:**
Crew solution information, namely the list of best crew members who can fill the open positions.

**Output:**
OK if authorized by the Operations Schedule Manager or NOT OK if not authorized.

**Description:**
If a solution (or a list of solutions) is found it is necessary to inform the Operations Schedule Manager, who has control over applying the solution or not.

**Extrinsic:**
The Operations Schedule Manager partner

Final Role, Interaction and Environment
Table 5.12 Agent Model (partial)

**Agent classes/roles**

\[ \text{OpMonitor}^{1..n} \rightarrow \text{RosterCrewMonitor, RosterAircraftMonitor, PaxMonitor} \]

This means that agent class \( \text{OpMonitor} \) will be defined to play the roles \( \text{RosterCrewMonitor, RosterAircraftMonitor, and PaxMonitor} \), and that we will have between one and \( n \) instances of this class in our MAS (\( n \) depends on the functional and physical distribution adopted).

\[ \text{OpAssign}^{1..n} \rightarrow \text{CrewAssign, AircraftAssign, PaxApply} \]

This means that agent class \( \text{OpAssign} \) will be defined to play the roles \( \text{CrewAssign, AircraftAssign and PaxApply} \), and that we will have between one and \( n \) instances of this class in our MAS (\( n \) depends on the functional and physical distribution adopted).

\[ \text{OpSupervisor}^{1..n} \rightarrow \text{OperationalControlSupervisor} \]

This means that agent class \( \text{OpSupervisor} \) will be defined to play the role \( \text{OperationalControlSupervisor} \), and that we will one instance of this class in our MAS.
Table 5.13 Services (partial) agent class OpMonitor

Service Description

Service: Monitor Crew Events.
Input: current date, crew slack time, pairing slack time.
Output: A list of dutyID, crewNumber, pmsgNumber, listOpenPositions, eventId.
Pre-condition: Successful connection with CrewSignON and Pairing resources.
Post-condition: A new crew event that has to be different from an existing unclosed one.

Service: Update crew event status.
Input: eventId, eventStatus.
Output: Number of records updated.
Pre-condition: Successful connection with CrewEvents resource.
Post-condition: Successful update of the CrewEvents resource.

Table 5.14 Service (partial) agent class OpSupervisor

Service Description

Service: Obtain crew solution authorization.
Input: List of crew members to be assigned.
Output: Authorization status (OK or NOT OK).
Pre-condition: At least one crew solution found.
Post-condition: User confirms or does not confirm authorization.

Service: Request crew solution application.
Input: Authorized list of crew members to be assigned.
Output: Request status (YES = solution can be applied, NO = solution cannot be applied).
Pre-condition: Authorization status = OK.
Post-condition: User sees status of the request on the screen.
Table 5.15 Some concepts and actions from MASDIMA

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrewEvent</td>
<td>Characterizes a crew event that initiates the process of finding a crew</td>
</tr>
<tr>
<td></td>
<td>solution.</td>
</tr>
<tr>
<td>CrewSolutionList</td>
<td>Characterizes a list of crew solutions proposed by the agents that are</td>
</tr>
<tr>
<td></td>
<td>specialists in crew problems to the OpCrewManager corresponding to the</td>
</tr>
<tr>
<td></td>
<td>CFP initiated after a crew event has been detected.</td>
</tr>
<tr>
<td>CrewSolution</td>
<td>Characterizes the crew solution chosen by agent OpCrewManager, that will</td>
</tr>
<tr>
<td></td>
<td>be presented to the OpSupervisor for authorization.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyCrewSolution</td>
<td>Action of applying the crew solution after it has been authorized.</td>
</tr>
<tr>
<td>UpdateEventStatus</td>
<td>Action of making the status update of a crew/aircraft or passenger event.</td>
</tr>
</tbody>
</table>

Table 5.17 Mapping (partial) of JADE behaviours and Services in MASDIMA

Service: MonitorCrewEvents
JADE Behavior: Ticker
FIPA/JADE IP: fipa-request
Protocol implementation name: request-solution

Service: FindCrew
JADE Behavior: Simpler
FIPA/JADE IP: fipa-request; fipa-contract-net
Protocol implementation name: request-solution; crew-solution-negotiation
Implementation - Development

```java
import java.util.Date;
import java.util.*;
public class MonitorCrowEventBehaviour extends TickeerBehaviour {
    public MonitorCrowEventBehaviour(Agent a, long period) {
        super(a, period);
    }
    // Monitors the crowSight and pairing resources
    protected void onTick() {
        Date dt = new Date("2009-09-10"); // date to monitor
        int crowBlack = 16; // crow black time
        int pairBlack = 10; // pairing black time

        // starts the service with the inputs
        CrowEvent ce = monitorResources(crowBlack, pairBlack);
        if (ce != null) {
            // starts the request solution protocol
        }

        // method that implements the SQL code to get
        // information from the environment
        private CrowEvent monitorResources(Date d, int cs, int ps) {
            CrowEvent ce = new CrowEvent();
            // Perform the SQL code to monitor the resources
            // returning the CrowEvent filled with information
            return ce;
        }
    }
}
```

Testing and Validation – Test Cases

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>Test Case</th>
<th>Test Case Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUN-023</td>
<td>Pedro</td>
<td>Detect flight problem and ask request solution.</td>
</tr>
<tr>
<td>Requirement(s)</td>
<td>RQ-017 and RQ-021</td>
<td></td>
</tr>
<tr>
<td>Preparation:</td>
<td>Flights in operation should appear in Flight Monitoring window.</td>
<td></td>
</tr>
<tr>
<td>Step number:</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>Step input:</td>
<td>An event should create a flight departure delay.</td>
<td></td>
</tr>
<tr>
<td>Step description:</td>
<td>Look at the Flight Problem window.</td>
<td></td>
</tr>
<tr>
<td>Step Expected Result:</td>
<td>The flight number, scheduled departure time, expected delay, number of violations and unsolved status, should appear.</td>
<td></td>
</tr>
<tr>
<td>Step Actual Result:</td>
<td>The same as the expected result.</td>
<td></td>
</tr>
<tr>
<td>Step number:</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>Step input:</td>
<td>An event should create a flight departure delay.</td>
<td></td>
</tr>
<tr>
<td>Step description:</td>
<td>Look at the Flight Map window.</td>
<td></td>
</tr>
<tr>
<td>Step Expected Result:</td>
<td>The flight affected should have a red circle blinking.</td>
<td></td>
</tr>
<tr>
<td>Step Actual Result:</td>
<td>The same as the expected result.</td>
<td></td>
</tr>
<tr>
<td>Step input:</td>
<td>The flight with the unsolved problem should appear in the Flight Problem window.</td>
<td></td>
</tr>
<tr>
<td>Step description:</td>
<td>The correct supervisor default values should appear for each dimension.</td>
<td></td>
</tr>
<tr>
<td>Step Expected Result:</td>
<td>The same as the expected result.</td>
<td></td>
</tr>
<tr>
<td>Step Actual Result:</td>
<td>The flight with solved status should appear in the Flight Problem window.</td>
<td></td>
</tr>
<tr>
<td>Step number:</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>Step input:</td>
<td>Click in the flight number.</td>
<td></td>
</tr>
<tr>
<td>Step description:</td>
<td>Click in the flight number.</td>
<td></td>
</tr>
<tr>
<td>Step Expected Result:</td>
<td>Should appear values for delays and cost for each dimension as well as the solution utility.</td>
<td></td>
</tr>
<tr>
<td>Step Actual Result:</td>
<td>The actions to be applied in the operational plan should appear for the dimensions.</td>
<td></td>
</tr>
<tr>
<td>Step number:</td>
<td>Test Result: PASS</td>
<td></td>
</tr>
<tr>
<td>Remarks:</td>
<td>In step 02 the clicking should stay for at least, 10 seconds.</td>
<td></td>
</tr>
</tbody>
</table>
Thanks for your attention!

Personal Web Pages
https://sites.google.com/site/antoniojmcastro

MASDIMA Project
http://www.disruptionmanagement.com

E-mail
antonio.castro@fe.up.pt