The shaping of the agent-oriented mindset

Koen Hindriks
Delft University of Technology, The Netherlands

6-5-2014
Overview

- *From Rational BDI Agents to …*
- *From Gaia to …*
- *From AGENT-0 to …*
- *From jEdit to Eclipse*
- Some application areas
- Vision and Research Agenda
What is the agent-oriented mindset?

“In your view, what are the key concepts in the agent-oriented mindset? If you had to identify just one, then what would it be and why?”

The audience at EMAS mentioned:
- autonomy
- rational
- goal-directedness
- interaction
- social
- reactive/events
- environment
- robustness
- decentralization
- Intentional stance
ATAL and DALT

*From* Rational BDI Agents

*to* …
Arguably, most research on BDI agents is influenced by Rao and Georgeff’s 1991 paper about BDI Logic: *Modeling Rational Agents within a BDI-Architecture*. 2007 Winner of IFAAMAS Influential Paper Award
BDI Logic

Key notions introduced in BDI logic:

- Events, beliefs, goals, and intentions
- Goals must be “compatible” with beliefs
- Intention must be “compatible” with goals
- Agents do not procrastinate wrt their intentions
- Blind, single- & open-minded commitment strategies
Another influential paper of Rao from 1996: **AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language**
Agents are **autonomous** computer programs, capable of independent action in **environments** that are typically **dynamic** and **unpredictable**.
ATAL: The Foundational Era

• Mental states and mental attributes are one ("the"?) essential of AOP

• Stan Franklin, Art Graesser, 1997, *Is It an agent, or just a program?: A taxonomy for autonomous agents*, in: Intelligent Agents III
Theories of intelligent agents: How do the various components of an agent's cognitive makeup conspire to produce rational behaviour?

Architectures for intelligent agents: What structure should an artificial intelligent agent have?

Languages for intelligent agents: What are the right primitives for programming an intelligent agent?
Key ATAL Results: Architectures

- Layered architectures, e.g., \textbf{InteRRaP} agent model:

- \textbf{dMARS} architecture (MAS extension of PRS)
- Early work on coordination & organizations
Key ATAL Results: Languages

The landscape of agent frameworks presented and introduced @ATAL. Includes operational agent languages and logical models.

In a sense this landscape defines a space of agents that can be created (and thus a corresponding mindset).
Key ATAL Results

- Methodologies: **MaSE**, first version **agentTool**

- Early work with main focus on the **analysis** and **design** phase.

- Concepts introduced: **Role & Conversation**
Agent *metaphors* and technologies are increasingly adopted to harness and govern the *complexity* of today’s systems. The growing complexity of agent systems calls for models and technologies that promote system *predictability* and enable *feature discovery* and *verification*.
DALT CfP Topics

- Declarative agent communication & coordination languages
- Knowledge-based and knowledge-intensive MAS
- Modeling of agent rationality
- Declarative approaches to the engineering of MAS
- High level agent specification languages
- Formal methods for the specification and verification of MAS
- Computational logics in MAS
- Argumentation and dialectical systems in MAS
- Declarative paradigms for combining heterogeneous agents
- Declarative representation of policies and security in MAS
- Models of social interaction, trust, commitments & reputation
- Game theory and mechanism design for multi-agent systems
Key DALT Results: Languages

Languages

• Go!
• Ordered Choice Logic Programming
• Dynamic Logic Programming (goal modelling)
• Answer Set Programming
• JASDL: Combining Agent and Semantic Web Technologies (Jason extension)

Logics and Logical Models

• Logic for ignorance; for expectation & observation
Key DALT Results: BDI Extensions

- Cooperative BDI models, e.g., Coo-BDI.
- Beliefs:
  - Resource-Bounded Belief Revision & Contraction
- Declarative goals:
  - Dynamics of goals, Maintenance goals, Preferences, goal generation, goal change, goal interactions
- Norms, Organizations, Electronic Institutions:
  - Norm-aware architecture
  - Specifying and Enforcing Norms in Artificial Institutions
  - Social Norm Emergence in Virtual Agent Societies
This work differentiates: query, achieve, maintain, and perform goals.
This work differentiates Perform, Achievement, Maintain goals; T = drop/abort/succeed/fail
This work differentiates: query, achieve, maintain, and perform goals.
Key DALT Results: Interaction

- Models for verification of agent dialogues
- Compliance of agent interactions
- Social commitments
- Commitment-Based Protocols
AOSE

From Gaia to ...
The Gaia Methodology

The Gaia Methodology is a framework for the design and implementation of multi-agent systems (MAS). It consists of several steps and models that guide the development process.

1. **Requirements Statement**: This step involves defining the goals and objectives of the MAS.
2. **Roles Model**: Defines the roles and responsibilities of the agents within the system.
3. **Interactions Model**: Specifies how agents interact with each other.
4. **Agent Model**: Describes the functionality and behavior of individual agents.
5. **Services Model**: Outlines the services provided by the agents.
6. **Acquaintance Model**: Details the relationships and connections between agents.

The process flows from analysis to design, focusing on the development of these models in a structured manner.
An agent is an autonomous system, capable of interacting with other agents in order to satisfy its design objectives.
AOSE CfP Topics

- Methodologies for agent-oriented analysis and design
- Formal methods for agent-oriented systems specification, verification and validation
- Computer-Aided SE (CASE) tools for AOSE
- Standardization efforts for multi-agent systems
- Engineering large-scale agent systems
- Practical coordination and cooperation frameworks for agent systems and engineering MAS organizations
- How can legacy software architectures be integrated with agent- or multi-agent-oriented applications?
- Autonomy vs. dependability and robustness
- Goal-oriented design
- Qualities and trade-offs of agent-based architectures
Key AOSE Results

Methodologies

- INGENIAS, MASDK, O-MaSE, PASSI, Prometheus, SODA, Tropos

Agent-Based Design patterns

Interaction Protocols
Key AOSE Results: Methodologies

Graphical notation and diagrams for specifying design
- Many variations and extensions of UML notation (AUML)

Tropos

Prometheus agent acquaintance diagram

Check out covers of AOSE Proceedings.
Key AOSE Results: Methodologies

**Design processes** (phases in development life cycle):

- Requirements
- Analysis
- Design (Architectural & Detailed)
- Implementation
- Testing

**Conceptual (meta-)models:**

- capabilities, (soft) goals, plans, roles, interaction protocols, mission, …
Key AOSE Results: Design

Organization Centred Design (or OC-MAS)

**AGR (Agent/Group/Role):**
- Organizations provide normative specifications on behavior
- Make no assumptions about cognitive capabilities of agents
- Group is organizational unit in which members interact freely

Other organizational meta-models:
- MOISE+, TEAMS, ISLANDER, OperA
Support for Testing Phase

- SUNIT: A Unit Testing Framework for Test Driven Development of Multi-Agent Systems

- Tropos testing framework including a testing process model for goal-oriented testing

- Unit testing of plan based agent systems, with a focus on automated generation and execution of test cases
ProMAS

From AGENT-0
to …
In order to differentiate AOP from OOP it is perhaps useful, to go back to the roots of the AOP paradigm to identify its main components.

“An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments. These components are defined in a precise fashion, and stand in rough correspondence to their commonsense counterparts.”

- a precise theory regarding the particular mental category: the theory must have clear semantics ("No Notation without Denotation"), and should correspond to the commonsense use of the term;
- a demonstration that the component of the machine obeys the theory;
- a demonstration that the formal theory plays a nontrivial role in analyzing or designing the machine (or, to coin a new phrase, "No Notation without Exploitation").

From: Shoham, 1993, Agent-Oriented Programming, Artificial Intelligence
The success of agent oriented system design can only be guaranteed if we can bridge the gap between analysis and design and implementation. This, in turn, requires the development of fully fledged and general purpose programming technology so that the concepts and techniques of MAS can be easily and directly implemented.
The ProMAS workshop series aims at promoting and contributing to the establishment of multi-agent systems as a mainstream approach to the development of industrial-strength software.
ProMAS CfP Topics

- Agent programming languages
- Extensions of traditional languages for MAS programming
- Programming **mobile** agents
- Algorithms, techniques, or protocols relevant to multi-agent programming (e.g., coordination, cooperation, negotiation)
- Agent **communication** issues in multi-agent programming
- Programming **social**, organizational & normative aspects
- **Interoperability** and **standards** for MAS
- Formal methods and tools for **specification & verification**
- Applications of multi-agent programming languages (incl. legacy)
- **Benchmarks** and **testbeds** for comparing MAS programming languages and tools
- Generic **tools** and **infrastructures** for multi-agent programming
Key ProMAS Results: Languages

Programming Languages

• JACK (BDI extension of Java)
• CLAIM (Support for Mobile agents)
• AF-APL (Agent Factory)
• Jadex (BDI agents on top of JADE)
• METATEM (Executing Temporal Logic)
• JIAC (Java Intelligent Agent Componentware)
• Jazzyk (Support for heterogeneous KR)
• JaCaMo (Jason + CArtAgO + Moise)
### How are these APLs related?

A comparison from a high-level, conceptual point, not taking into account any practical aspects (IDE, available docs, speed, applications, etc).

<table>
<thead>
<tr>
<th>Basic concepts: beliefs, action, plans, goals-to-do</th>
<th>Families of Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGENT-0(^1) (PLACA)  (\Leftrightarrow) Golog (=) 3APL(^3)</td>
<td>AgentSpeak(L), Jason(^2)</td>
</tr>
</tbody>
</table>

**Main addition: Declarative goals**

\[ 2APL \approx 3APL + \text{GOAL} \]

**Java-based Cognitive Agent Languages**

- AF-APL, JACK (commercial), Jadex, Jazzyk

**Logic Programming**

- METATEM

**Mobile Agents**

- CLAIM

---

\(^1\) mainly interesting from a historical point of view

\(^2\) from a conceptual point of view, we identify AgentSpeak(L) and Jason

\(^3\) without practical reasoning rules
Key ProMAS Results: Extensions

Programming Constructs: Goals & Modules

- **Capability** for Agent Modularization (Jadex)
- **Declarative goals** (goal-directed 3APL)
- **Modules** (GOAL)
- **Modularity and Compositionality** (Jason)

Capabilities: Planning, Learning, Organizing

- **Planning** (Jadex)
- **Organisations** (2OPL)
- **Reinforcement Learning** (GOAL)
An Agent is a Set of Modules

Built-in modules:

- **init** module:
  - Define global knowledge
  - Define initial beliefs & goals
  - Process “send once” percepts
  - Specify environment actions

- **main** module
  - Action selection strategy

- **event** module
  - Process percepts
  - Process messages
  - Goal management

User-defined modules.

```plaintext
init module{
    knowledge{
        ...
    }
    beliefs{
        %%% INITIAL BELIEFS ONLY IN INIT MODULE %%%
    }
    goals{
        ...
    }
    program{
        %%% PROCESS "SEND ONCE" PERCEPTS HERE %%%
    }
    actionspec{
        %%% SPECIFY ENVIRONMENT ACTIONS HERE %%%
    }
}

main module{
   % OPTIONAL knowledge section
   % NO beliefs section HERE!
   % OPTIONAL goal section (not advised in 'main')
   program{
       %%% ENVIRONMENT ACTION SELECTION HERE %%%
   }
}

event module{
   program{
       %%% PROCESS PERCEPTS HERE %%%
       %%% PROCESS MESSAGES HERE %%%
       %%% PERFORM GOAL MANAGEMENT HERE %%%
   }
}
```
Key ProMAS Results: Environment

Environment Modelling
• Artifacts: building blocks for environment modelling
  - Environment Interface Standard
  - PRESAGE: Simulation of Agent Societies

MAPC Agent Contest Competition
• 2005-2007: Gold Miners (??, Jason, JIAC)
  - 2008-2010: Cow Herders (JIAC, JIAC, JIAC)
  - 2011-2013: Mars Explorers (GOAL, Jason, Jason)
Design of a Generic Environment Interface

- Environment Interface
- APL Side
- Environment Side
- Environment Management System
- Platform
- Agents
- Environment Model
From jEdit to Eclipse
Key ProMAS Results: Tooling

Development Tools & Techniques

- DECAF, an MAS development toolkit
- Tracer Tool for debugging agents
- Debugging in AFAPL
- AIL – Agent Infrastructure Layer
- Toolipse: JIAC development tool in Eclipse
- MDL: Debugging using LTL (3APL)
- Model Checking Agent Programs (GOAL)
A Tooling Perspective on Agents

Infrastructurally, developing and running an agent requires a set of components:

- Editor/Parser (KRT)
- Reasoner (KRT)
- Interpreter
- Middleware
- Environment
- Debugger
- Verifier (e.g., MC)
Cognitive Agent Debugging Tool

Should provide support for three key stages:

1. Which events happened? Why action/plan selected?
2. Which decision made?
3. Which action performed? What changed as result?
Toolipse 2 (JIAC; editor)

Navigator
Outline
Agent World Editor
JADLEdit Editor
JADLEdit Browser
SeMA²
Semantic Service Matching
Jadex Control Center (Debug)

Navigator

Breakpoints

Views:
- BDI Viewer
- Agent Inspector
- Rule Engine
Agent Architecture and Cycle

Agent

Process percepts
- percept rules
- knowledge
- beliefs
- goals

Action selection
- action specification
- action rules / program

Cycle

Process percepts & messages
(= apply percept rules)

Select action
(= apply action rules)

Perform action
(= send to environment)

Update mental state
(= apply action specs + commitment strategy)

Environment
Jason Eclipse Plugin

MAS Console (running, debugging)
2APL Debug Perspective

Many views on agent components and state
GOAL’s Eclipse Debug Perspective

Agent processes overview (threads) & breakpoints tab

Code stepping

Mental state introspector

Rule evaluator

Output console

Interactive console
Empirical Work

Some empirical work (controlled studies with subjects) reported, e.g.:

Evaluation of a Conversation Management Toolkit for Multi Agent Programming
Engineering MAS (EMAS)
Vision
Increasing Demand for AI

- McKinsey: by 2025, machines will be able to learn, adjust, exercise judgment, and reprogram themselves.

- Users expect a more personalized interaction with their devices and machines.
The Next Generation AI Engineers

Artificial Intelligence

... will need to develop complex intelligent and autonomous decision-making systems

... apply complex AI techniques:
- automated reasoning
- machine learning
- automated planning
- ...
The Next Generation AI Engineers

AI is going to make life easier for us...

... only if we make life easier for AI engineers.
Cognitive agent technology offers a powerful solution for developing the next generation autonomous decision-making systems.
Dagstuhl 2012 Roadmap

Challenge Areas

• BDI+ Agents
• Coordination & Organization
• Tooling & Benchmarks
• Agent technology & legacy
• Component-based agents

Key Issues

• How can we quantify benefits of agent technology?
• What is added value of agent technology?
• What are needs and issues faced by agent developers.
Let’s stop talking about BDI agents and let’s start talking only about Cognitive Agents

Why?
To increase adoption of our technology:
• No need anymore to explain what BDI stands for
• BDI agents carry too much an association of complex logic and rationality.
Engineering Intelligent Agents

Make it easy for programmers to unleash the power of AI techniques.

Why? We need more sophisticated agents to delegate decision-making and control to.

Cf. also intro of Nick Jennings, AOSE 1999:
• Agents are also being used as an overarching framework for bringing together the component AI sub-disciplines that are necessary to design and build intelligent entities.

Already proposals for planning, learning, and emotions.
## Combining AOP and Planning

<table>
<thead>
<tr>
<th><strong>GOAL</strong></th>
<th><strong>Planning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Knowledge</td>
<td>• Axioms</td>
</tr>
<tr>
<td>• Beliefs</td>
<td>• (Initial) state</td>
</tr>
<tr>
<td>• Goals</td>
<td>• Goal description</td>
</tr>
<tr>
<td>• Program Section</td>
<td>• x</td>
</tr>
<tr>
<td>• Action Specification</td>
<td>• Plan operators</td>
</tr>
</tbody>
</table>
Why? Show relevance of our computing paradigm to mainstream computing science.

Intro Nick Jennings AOSE 1999:

• Agents are being advocated as the next generation model for engineering complex, distributed systems.

Demonstrate that agent-orientation can solve key concurrency and distributed computing issues.
Methodologies & Languages

More effort needed to connect methodologies to agent programming languages.

Why?

- To stimulate adoption of our technology, we need to provide the complete package.
- Complete evaluation of methodologies cannot be done without considering target platform.
- Integration of the (slightly different) conceptual models of methodologies and APLs.
Why? Still many open issues:

- “Little has been done so far into transforming autonomy into a practical software property”
- Re-use agents, MAS, …?
- “Open Agent Systems ???”
- …
Mature Tooling Support

Why?
Tools should comply with current standards and provide ease of use to increase adoption.

- Complexity of the basic cycle of many agents is issue: not clear how to best present to user.
- Sophisticated debugging & testing tools!
Standardising EMAS

Standard interfaces for cognitive agents

Why?

- Facilitates easy exchange between platforms.
- Facilitates component-based approach to engineering MAS. Towards plugin Architecture?
- Objectives of AOSE, ProMAS, and DALT
Performance & Scalability

Need high-performing cognitive agents

Why?

- Performance issues are barrier to adoption
  - Develop efficient agent tools and interpreters that scale in practice.

- We need scalable systems (within reason) for, e.g., agent-based simulation.
Logic-based agents are simpler

Why? Prefer simplicity and elegance.

• Cycles and components of logic-based agent languages are fewer and simpler and therefore easier to understand by programmer.

• Java-based frameworks do not clearly demonstrate the power of the AOP paradigm: was it Java or the cognitive agent that made the difference?
Education is the first step

Teach the agent-oriented mind-set

Why? We need to train people to know how to apply our technology to ensure adoption.

Facilitate use of agent-oriented paradigm:
• Created and make available assignments and teaching materials
• Make tutorial materials widely available.
Multi-Agent Systems Project

Course Multi-Agent Systems:
Learn to program a multi-agent system

Develop logic-based agents programs:
- Apply reasoning technology (Prolog)
- Write agent programs (GOAL)
- Hands-on experience by various programming assignments.

Project Multi-Agent Systems:
CTF Competition in UT2004

- Control a team of bots by means of a multi-agent system.
- Compete at the end of the project.

Create fun assignments and projects! (UT3, competition)
Concluding Summary

- Let’s talk about Cognitive Agents from now on
- Easy access to powerful AI techniques
- Demonstrate AOP solves concurrency issues
- Integrate methodologies and APLs
- Address Concrete AOSE Issues
- Standard interfaces for cognitive agents
- Mature tooling for agent development
- Need high-performing cognitive agents
- Logic-based agents are simpler
- Teach the agent-oriented mindset