Agent-Oriented Programming

(A3M33UI Spring 2010 - Lecture #12)

Peter Novák
Agent Technology Center, Department of Cybernetics
Czech Technical University

May 4th 2010
Agents

**Taxonomy of agent types** (in order of increasing capability):

1. simple reflex agents
2. reflex agents with state
3. goal-based agents
4. utility-based agents
5. ... learning agents
Introduction

Agents

**Taxonomy of agent types** (in order of increasing capability):

1. simple reflex agents
2. reflex agents with state
3. goal-based agents
4. utility-based agents $\leftarrow$ lecture #10
5. ... learning agents $\leftarrow$ lecture #11
Taxonomy of agent types (in order of increasing capability):

1. simple reflex agents
2. reflex agents with state
3. goal-based agents ← lecture #12, now!
4. utility-based agents ← lecture #10
5. ... learning agents ← lecture #11
Goal-based agents

goals + state + actions’ consequences $\rightsquigarrow$ action selection

$\downarrow$

(reactive) planning, means-end reasoning

$\downarrow$

agent programming languages
Goal-based agents

goals + state + actions’ consequences $\mapsto$ action selection

(reactive) planning, means-end reasoning

agent programming languages

Peter Novák · Agent Technology Center, Department of Cybernetics Czech Technical University

May 4th 2010
Lecture overview

1. Intelligent agents revisited

2. Agent-oriented programming
   - Motivation
   - Theoretical foundations
   - Agent programming languages
   - AgentSpeak(L)/Jason
   - APLs: the context

3. Agent-oriented software engineering

4. Conclusion
Intelligent agents are embodied artificial entities assumed to be autonomous, proactive, reactive, as well as socially able.

- **autonomy**: the agent acts without its user’s intervention, *agent is not an object* (in OOP sense)
- **proactiveness**: the agent acts *purposefully*, i.e., towards reaching goals
- **reactivity**: the agent is responsive to changes of the environment
- **social ability**: it is capable to communicate, coordinate and cooperate with its users and peers

**communication ⇝ knowledge representation!**
Cognitive agents

**cognitive/knowledge intensive agent**

employ cognitive processes, such as knowledge representation and reasoning as the basis for decision making and action selection. I.e., they construct and maintain a *mental state*.

**mental state**

agent’s internal explicit representation of the environment, itself, its peers, etc. ~* agent’s memory
Cognitive agents (cont.)

mental state

goals

beliefs

perceptions

action selection

actions

beliefs a database of agent’s information about itself, the world (environment), other agents, etc.

⇝ NOW

goals description of states the agent “wants” to bring about

⇝ FUTURE

How to select actions leading from NOW to the FUTURE

Peter Novák · Agent Technology Center, Department of Cybernetics Czech Technical University

May 4th 2010 · 7/31
Cognitive agents (cont.)

mental state

goals

beliefs

perceptions

action selection

actions

beliefs a database of agent’s information about itself, the world (environment), other agents, etc.

⇝ NOW

goals description of states the agent “wants” to bring about

⇝ FUTURE

How to select actions leading from NOW to the FUTURE
Definition (planning)

... is the process of generating (possibly partial) representations of future behavior prior to the use of such plans to constrain or control that behavior. The outcome is usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents.

(The MIT Encyclopedia of the Cognitive Sciences)

to arrive to a valid plan, in the worst case, the planner has to explore all the possible action sequences!!!

→ very high computational complexity... 😊
The problem

How to select actions leading from NOW to the FUTURE?

plan - execute - monitor cycle

1. plan from the current state to a goal state(s)
2. sequentially execute actions from the plan
3. monitor success of action execution
   ▶ in the case of action failure, (re-)plan again (goto 1)

speed of planning vs. environment dynamics

planning $\succ$ environment can perform relatively well

planning $\prec$ environment can lead to fatal inefficiencies

⇝ the system “suffocates” in (re-)planning

Peter Novák · Agent Technology Center, Department of Cybernetics Czech Technical University
The problem

How to select actions leading from NOW to the FUTURE?

plan - execute - monitor cycle

1. plan from the current state to a goal state(s)
2. sequentially execute actions from the plan
3. monitor success of action execution
   ▶ in the case of action failure, (re-)plan again (goto 1)

speed of planning vs. environment dynamics

planning $\overset{\text{speed}}{\succ}$ environment can perform relatively well
planning $\overset{\text{speed}}{\prec}$ environment can lead to fatal inefficiencies
$\leadsto$ the system “suffocates” in (re-)planning
The idea: reactive planning

reactive planning

Instead of plan-execute-monitor cycle, select partial plans reactively on the ground of the current state of the world.

Reactive planning agent:

current beliefs + future goals $\leadsto$ choose from a plan library
Belief-Desire-Intention (BDI)

⇝ a reactive planning agent architecture. Establishes intentions as a first-class concept.

structural decomposition:

- **(B)eliefs**: reflect agent’s static beliefs about its environment, itself, its peers, etc. (now)
- **(D)esires**: descriptions of situations the agent wants to bring about (future)

system dynamics: ~ from now to the future

- **(I)ntentions**: courses of action, plans, the agent commits to
  - partial plans of action that the agent is committed to execute in order to fulfill the goals
Means-end reasoning

explicit partial plans $\rightsquigarrow$ *recipes* for how to proceed from now to the future

means-end reasoning $\rightsquigarrow$ to reach an end, there are means to employ...

- plans (recipes) are selected from a pre-encoded plan library
Is it a good idea?

Benefits:
- facilitates quick response to changes in the environment
- allows layering of the system
  - knowledge representation + reasoning vs. plan selection
- plans can be encoded in the design time
  - better control on what the system does $\leadsto$ software engineering

Shortcomings:
- (potentially) too much control of the system (is this still AI?)
- no (straightforward) guarantees that the means leads to reaching the end (unlike planning!)
- plan selection is quite greedy $\leadsto$ the system can get stuck
BDI deliberation

Execution cycle:

1. collect new events
2. collect applicable plans (options w.r.t. events/goals)
3. select a plan $\pi$ (deliberate and update intentions)
4. execute $\pi$
5. drop failed/unachievable plans

following plans $\rightsquigarrow$ the agent acts with an intention.

Intentional stance

Given its beliefs, we should be able to predict what a rational agent will do in a given situation to reach its goals. If the agent turns out to behave according to such a pattern, the agent can be seen as having an intention to follow the right course of action. We can ascribe it an intentional stance.
BDI deliberation

Execution cycle:

1. collect new events
2. collect applicable plans (options w.r.t. events/goals)
3. select a plan $\pi$ (deliberate and update intentions)
4. execute $\pi$
5. drop failed/unachievable plans

following plans $\leadsto$ the agent acts with an intention.

Intentional stance

Given its beliefs, we should be able to predict what a rational agent will do in a given situation to reach its goals. If the agent turns out to behave according to such a pattern, the agent can be seen as having an intention to follow the right course of action. We can ascribe it an intentional stance.
Agent-oriented programming

Promotes programming with mentalistic notions and intentional stance as an abstraction. Provides a realization of the BDI agent architecture in pragmatic programming languages.

AOP system:

1. a logical system for mental states
2. an interpreted programming language
3. an ‘agentification’ process
Agent-oriented programming

Promotes programming with **mentalistic notions** and **intentional stance** as an abstraction. Provides a realization of the BDI agent architecture in pragmatic programming languages.

**AOP system:**

1. a logical system for *mental states*
2. an *interpreted programming language*
3. an ‘*agentification’* process
Agent programming languages

Why APLs?
- software engineering → systems modeling
- clear properties (expressivity, verification, etc.)

What is an APL?
- operationalization of a BDI architecture
- provides an underlying architecture & *tools for encoding the system dynamics*
- syntax & model of execution (semantics)
AgentSpeak(L)/Jason

- programming language for BDI agents
- notation based on logic programming
- AgentSpeak(L) \(\rightarrow\) an abstract programming language
- Jason \(\rightarrow\) operational semantics for AgentSpeak
- incorporates Prolog for reasoning about beliefs
- various pragmatic extensions like external actions (Java)
- also a platform for developing multi-agent systems

http://jason.sourceforge.net/
AgentSpeak syntax

**Beliefs**
represent the information available to an agent (e.g., about the environment or other agents)

\[
\text{location}(\text{Object, Coordinate}), \text{at}(\text{Coordinate})
\]

**Goals**
represent states of affairs the agent wants to bring about (come to believe, when goals are used declaratively)

- **achievement goals:** achieve a state
  \[
  !\text{write}(\text{book}), \text{at}(\text{Coordinate})
  \]

- **test goals:** retrieve information from the belief base
  \[
  ?\text{at}(\text{location}(\text{Object, Coordinate}))
  \]
Agent reacts to events by executing plans.

agent program = set of rules

\[
\text{triggering event} : \text{context} \leftarrow \text{body}.
\]

**triggering event**: (perceived) change/event to handle

\(+b, -b, +!g, -!g, +?g, -?g \mapsto\) implicit goals!

**context**: circumstances in which the plan can be used

\(\land, \lor, \neg\)

**body**: the course of action to be used to handle the event if the context is believed true at the time a plan is being chosen to handle the event

\(\mapsto\) a means to an end
1. perceive the environment
   - receive communication from other agents
   - select ‘socially acceptable’ messages
2. update the belief base
3. select an event to handle
4. retrieve all relevant plans
5. determine the applicable plans
6. select one applicable plan
7. select an intention for further execution
8. execute one step of an intention
Jason (example)

+green_patch(Rock)
  : not battery_charge(low)
  <- ?location(Rock,Coordinates);
      !at(Coordinates);
      !examine(Rock).

+!at(Coords)
  : not at(Coords) & safe_path(Coords)
  <- move_towards(Coords); !at(Coords).
Contingency plans & failure

Contingency plans $\leadsto$ multiple rules + single triggering event:

+!at(Coords)
  : not at(Coords) & safe_path(Coords)
  $\leftarrow$ move_towards(Coords); !at(Coords).

+!at(Coords)
  : not at(Coords) & no_safe_path(Coords) & not storm
  $\leftarrow$ fly_towards(Coords); !at(Coords).

+!at(Coords)
  : not at(Coords) & very_bad_weather
  $\leftarrow$ ask_for_teleport(Coords); ....

A plan failure triggers a goal-deletion event:

-!at(Coords)
  : very_bad_weather
  $\leftarrow$ !wait_for_good_weather.
Contingency plans & failure

Contingency plans $\rightarrow$ multiple rules + single triggering event:

+$!at$(Coords)
  : not $at$(Coords) & $safe\_path$(Coords)
  $\leftarrow$ $move\_towards$(Coords); $!at$(Coords).

+$!at$(Coords)
  : not $at$(Coords) & $no\_safe\_path$(Coords) & not storm
  $\leftarrow$ $fly\_towards$(Coords); $!at$(Coords).

+$!at$(Coords)
  : not $at$(Coords) & $very\_bad\_weather$
  $\leftarrow$ $ask\_for\_teleport$(Coords); ....

A plan failure triggers a goal-deletion event:

-$!at$(Coords)
  : $very\_bad\_weather$
  $\leftarrow$ $!wait\_for\_good\_weather$. 
Internal actions

Internal actions serve:

1. as a glue between Jason and external legacy code (Java)
   ```plaintext
   map.get_coords(Rock, Coords),
   .send(...), .print(...)
   ```

2. as a means to manually steer the deliberation cycle
   ```plaintext
   .desires(...), .drop_desires(...) 
   ```

```plaintext
+green_patch(Rock)
  : ~battery_charge(low) & .desires(at(_))
  ← .drop_desires(at(_));
      map.get_coords(Rock, Coords);
  !at(Coords);
  !examine(Rock).
```
MAS specification

Putting it all together:

\[
\text{MAS} = \text{agents} + \text{communication} + \text{environment}
\]

MAS mars_exploration_system {
    /* communication infrastructure (built-in) */
    infrastructure: Centralised

    /* interface to the environment (Java class) */
    environment: MarsRoverEnv

    /* agents in the MAS (Jason agents) */
    agents: Spirit; Opportunity; Beagle2;
}

Peter Novák · Agent Technology Center, Department of Cybernetics Czech Technical University
May 4th 2010
What can APLs do for us?

1. mentalistic abstractions for agent system specification
   such as beliefs, desires, intentions, plans, practical reasoning rules, etc.,

2. agent-oriented language semantics
   corresponding to temporal modal logics (?)

3. means to tackle the pro-activity vs. reactivity problem
   deliberation/planning vs. handling events & interruptions
   hybrid architectures
Agent-oriented programming
APLs: the context

Historical overview

Hybrid architectures:
1987: PRS
1988: IRMA
1991: Abstract BDI architecture
1994: INTERRAP

Agent-Oriented Programming Languages:
1990: AGENT-0
1996: AgentSpeak(L)
1996: Golog
1997: 3APL
1998: ConGolog
2000: JACK
2000: GOAL
2002: Jason
2003: Jadex
2008: BSM/Jazzyk
2008: 2APL

– incomplete –
(Georgeff and Lansky)
(Bratman, Israel and Pollack)
(Rao and Georgeff)
(Müller and Pischel)

– incomplete –
(Shoham)
(Rao)
(Reiter, Levesque, Lesperance)
(Hindriks et al.)
(Giacomo, Levesque, Lesperance)
(Busetta et al.)
(Hindriks et al.)
(Bordini, Hubner)
(Braubach, Pokahr et al.)
(Novák)
(Dastani)
BDI programming systems

Theoretically oriented

- declarative languages built from scratch \(\leadsto\) new syntax
- clear theoretical properties \(\leadsto\) verification
- declarative KR techniques
- no integration with external/legacy systems

AgentSpeak(L), 3APL, 2APL, GOAL, CAN, etc.

Engineering approaches

- layer of specialised language constructs over a robust mainstream programming language (Java) \(\leadsto\) code re-usability
- host language semantics
- KR in an imperative language
- easy integration with external systems and environments

JACK, Jadex
Agent engineering

development methodologies

- design languages
- agent-oriented prog. languages
- run-time system
- HW
Agent engineering

development methodologies

design languages

agent-oriented prog. languages

run-time system

HW

Peter Novák · Agent Technology Center, Department of Cybernetics Czech Technical University
AOSE frameworks

Modelling frameworks:
- Tropos
- MaSE
- AUML
- AML
- ...

Methodologies:
- Gaia
- Tropos
- Prometheus
- MaSE
- ...

Special purpose methodologies & modelling tools directed towards:
- emergent systems
- mobile agents
- swarm intelligence
Summary

1. Intelligent agents revisited

2. Agent-oriented programming
   - Motivation
   - Theoretical foundations
   - Agent programming languages
   - AgentSpeak(L)/Jason
   - APLs: the context

3. Agent-oriented software engineering

4. Conclusion
The end

Thank you for your attention.