Building intelligent agents

(A4M33MAS/autumn 2011/lecture #4)

Peter Novák

Agent Technology Center, Department of Cybernetics

Czech Technical University

October 11th 2011

Czech Technical University

agent technology center

Cognitive agents revisited

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. $\rightsquigarrow$ agent's memory

The problem

- How to build systems involving mentallistic concepts?
- What are the general principles and guidelines to follow?
- Why building such systems matters?
- What are the main problems we face when building such systems?
- What is the state-of-the-art in this field?

Lecture outline

1 Motivation & basic concepts

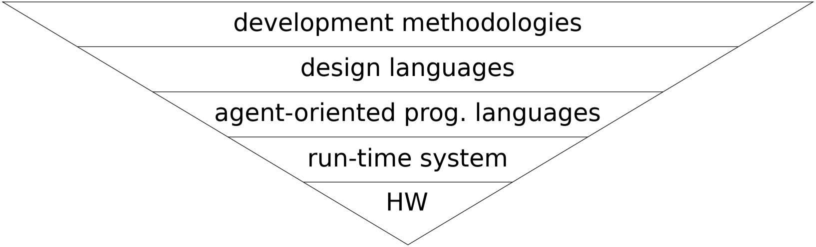
2 Agent-oriented software engineering
- Introduction
- Frameworks
- Tropos methodology
- Formal specification of agents

3 Agent-oriented programming
- Introduction
- Agent programming languages
- BDI design patterns

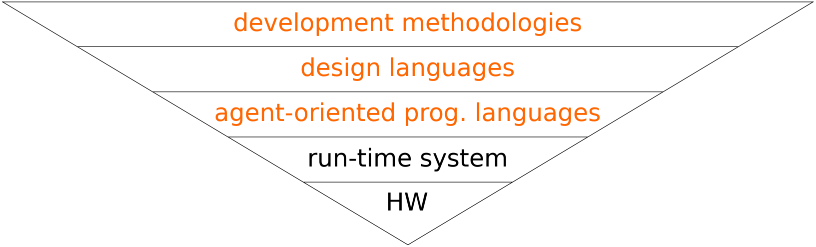
4 Conclusion

Motivation & basic concepts

Agent engineering



Agent engineering



Why "agent-oriented"?

Embodied agents in dynamic & unstructured environments!

- social ⇝ communication ⇝ language ⇝ knowledge representation, reasoning
- autonomy ⇝ decision making, robust & modular implementation
- proactive ⇝ opportunistic ⇝ non-deterministic, parallel
- reactive ⇝ interruptible

traditional approaches perform poorly in such contexts

1. interruptions & reactivity ⇝ exceptions vs. context restore
2. non-determinism vs. structure ⇝ declarative languages (?)
3. modularity vs. the above ⇝ elaboration tolerance, compositionality
4. parallelism vs. the above ⇝ separation vs. interactions
5. KR&R ⇝ logic-based approaches

Why "agent-oriented"?

Embodied agents in dynamic & unstructured environments!

- social ⇝ communication ⇝ language ⇝ knowledge representation, reasoning
- autonomy ⇝ decision making, robust & modular implementation
- proactive ⇝ opportunistic ⇝ non-deterministic, parallel
- reactive ⇝ interruptible

traditional approaches perform poorly in such contexts

1. interruptions & reactivity ⇝ exceptions vs. context restore
2. non-determinism vs. structure ⇝ declarative languages (?)
3. modularity vs. the above ⇝ elaboration tolerance, compositionality
4. parallelism vs. the above ⇝ separation vs. interactions
5. KR&R ⇝ logic-based approaches

AO software engineering

Highly parallel non-deterministic interruptible behaviours relying on relatively heavy weight knowledge representation and reasoning.

How to model systems in terms of mentalistic concepts?

- knowledge, beliefs
- goals
- obligations
- plans
- roles
- speech-acts

What is the right methodology?

- How to analyse systems?
- How to design systems?

AO programming languages

Highly parallel non-deterministic interruptible behaviours relying on relatively heavy weight knowledge representation and reasoning.

What is the computational model we should employ for building non-deterministic, parallel and interruptible systems?

- plan encoding
- plan instantiation
- plan execution
- monitoring
- replanning
- failure handling
- reasoning
- integration

What is the system semantics?

- how to: design $\rightsquigarrow$ implement $\rightsquigarrow$ execute?
- How to verify?

A4M33MAS/Lecture #4

Agent-oriented software engineering

What is AOSE?

- methods and tools for supporting development of agent and multi-agent systems oriented software engineering
- modelling languages for the specification of MAS
- techniques for requirements elicitation and analysis
- architectures and methods for designing agents and their organizations
- platforms for implementation and deployment of MAS
- validation and verification methods

AOSE frameworks

Modelling frameworks:

- Tropos
- MaSE
- AUML
- AML
- . . .

Methodologies:

- Tropos
- Gaia
- Prometheus
- MaSE
- . . .

Special purpose methodologies & modelling tools directed towards:

- emergent systems
- mobile agents
- swarm intelligence

Tropos: overview

Tropos is an agent-oriented software engineering (AOSE) methodology that covers the whole software development process.

- requirements -driven software development approach $\leadsto$ exploits goal analysis and actor dependencies analysis
- covers also the very early phases of requirements analysis $\leadsto$ deeper understanding of the environment & interactions between software and human agents
- spans from early analysis down to agent-oriented programming languages issues
- uses mentalistic notions (agent, role, goals, plans, etc.) $\leadsto$ from early analysis down to the actual implementation.

Tropos language

Basic concepts:

- Actor
  - intentional entity: role, position, agent (human or software)
  - agent is an actor which occupies a position covering (several) roles played by the agent
- Goal
  - strategic interest of an actor
  - is associated to an actor.
    - *hard*: clear satisfaction criteria
    - *soft*: qualitative “soft” criteria
- Task
  - a course of action (plan/process) associated with a goal and used to satisfy it by execution

Tropos language (cont.)

Basic concepts (cont.):

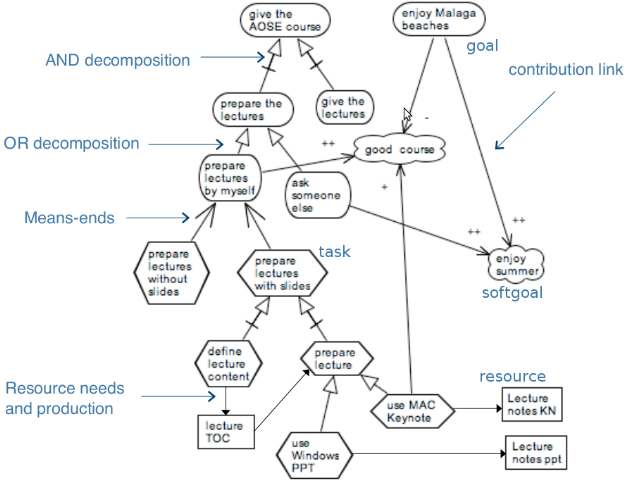
- Resource
  - physical, or informative non-intentional entity
  - can be *used*, *produced*, or *shared*
- Social dependency (between two actors)
  - one actor depends on another to accomplish a goal, execute a task, or deliver a resource
  - the content can be a goal/task/resource

Tropos language (cont.)

Basic relations between entities:

- Decomposition
  - AND decomposition
  - OR decomposition
  - goal $\rightsquigarrow$ subgoals
  - task $\rightsquigarrow$ subtasks
- Means-ends
  - a task (mean) used to achieve a goal (end)
- Contribution
  - a goal/task/softgoal contributes to the satisfaction of a softgoal
- Resource need
  - a task/goal needs a resource
- Resource production
  - a task/goal produces a resource

Example model



Tropos methodology

Phases:

1. Early requirements (*social domain*)
   - socio- and organizational setting is analyzed and the most relevant actors and their relationships are identified
2. Late requirements (*system in the domain*)
   - the system is introduced as a new actor of the social domain and analyzed in terms of *Tropos* concepts
3. Architectural design (*analysis/decomposition*)
   - the *actor system* is designed
   - subactors are introduced and goals/task are assigned
   - agents are identified
   - agent capabilities are identified
4. Detailed design (*detailed design*)
   - capabilities, protocols, and agent's tasks/plan are specified in detail

Tropos methodology

Phases:

1. Early requirements (*social domain*)
   - socio- and organizational setting is analyzed and the most relevant actors and their relationships are identified
2. Late requirements (*system in the domain*)
   - the system is introduced as a new actor of the social domain and analyzed in terms of *Tropos* concepts
3. Architectural design (*analysis/decomposition*)
   - the *actor system* is designed
   - subactors are introduced and goals/task are assigned
   - agents are identified
   - agent capabilities are identified
4. Detailed design (*detailed design*)
   - capabilities, protocols, and agent's tasks/plan are specified in detail

Tropos methodology

Phases:

1. Early requirements (*social domain*)
   - socio- and organizational setting is analyzed and the most relevant actors and their relationships are identified
2. Late requirements (*system in the domain*)
   - the system is introduced as a new actor of the social domain and analyzed in terms of *Tropos* concepts
3. Architectural design (*analysis/decomposition*)
   - the *actor system* is designed
   - subactors are introduced and goals/task are assigned
   - agents are identified
   - agent capabilities are identified
4. Detailed design (*detailed design*)
   - capabilities, protocols, and agent’s tasks/plan are specified in detail

Tropos methodology

Phases:

1. Early requirements (*social domain*)
   - socio- and organizational setting is analyzed and the most relevant actors and their relationships are identified
2. Late requirements (*system in the domain*)
   - the system is introduced as a new actor of the social domain and analyzed in terms of *Tropos* concepts
3. Architectural design (*analysis/decomposition*)
   - the *actor system* is designed
   - subactors are introduced and goals/task are assigned
   - agents are identified
   - agent capabilities are identified
4. Detailed design (*detailed design*)
   - capabilities, protocols, and agent's tasks/plan are specified in detail

Tropos methodology

Phases:

1. Early requirements (*social domain*)
   - socio- and organizational setting is analyzed and the most relevant actors and their relationships are identified
2. Late requirements (*system in the domain*)
   - the system is introduced as a new actor of the social domain and analyzed in terms of *Tropos* concepts
3. Architectural design (*analysis/decomposition*)
   - the *actor system* is designed
   - subactors are introduced and goals/task are assigned
   - agents are identified
   - agent capabilities are identified
4. Detailed design (*detailed design*)
   - capabilities, protocols, and agent's tasks/plan are specified in detail

Temporal & epistemic logics recap.

Capture the properties of an agent system:

1. evolution of the system in time
2. structure and component relationships of the internal state
   - beliefs, desires, intentions, obligations, commitments, etc.

↓

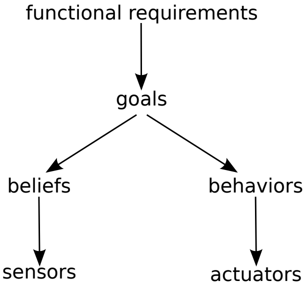
evolution of beliefs, desires, etc. in time

Logics and SW engineering

Role of logics in software engineering

Temporal & epistemic logics provide means to capture important fragments of system specification and lifecycle.

- the system eventually achieves, resp. always maintains goals ($\Diamond$, $\Box$)
- perceiving a sensor in the next step leads to belief update ($\bigcirc$)
- upon holding a belief, a goal should be adopted, resp. dropped ($\bigcirc$)
- a goal sometimes, resp. always triggers a behavior ($\Diamond$, $\Box\bigcirc$)
- behaviors eventually lead to fulfillment of goals ($\Diamond$)



Modeling goals: achievement

$$\mathsf{G}\Diamond\varphi \longrightarrow \Diamond\mathsf{B}\varphi$$

$$\mathsf{G}\Diamond\varphi\,\mathcal{U}\,\mathsf{B}\varphi$$

ACHIEVEMENT-GOAL:

- $\mathsf{B}\varphi_{adopt} \wedge \neg\mathsf{G}\Diamond\varphi \longrightarrow \mathsf{G} \oplus \Diamond\varphi$
- $\mathsf{G}\Diamond\varphi \wedge \mathsf{B}\varphi_{drop} \longrightarrow \mathsf{G} \ominus \Diamond\varphi$
- $\mathsf{G}\Diamond\varphi \wedge \mathsf{B}\varphi \longrightarrow \mathsf{G} \ominus \Diamond\varphi$
- $\mathsf{G}\Diamond\varphi \longrightarrow \mathsf{E} \oslash \mathsf{behavior}_{\varphi}$

Modeling goals: maintenance

$$\mathsf{G}\Box\varphi \wedge \mathsf{B}\neg\varphi \longrightarrow \Diamond\Box\mathsf{B}\varphi$$

MAINTENANCE-GOAL:

- $\mathsf{B}\varphi_{adopt} \wedge \neg\mathsf{G}\Box\varphi \longrightarrow \mathsf{G} \oplus \Box\varphi$
- $\mathsf{G}\Box\varphi \wedge \mathsf{B}\varphi_{drop} \longrightarrow \mathsf{G} \ominus \Box\varphi$
- $\mathsf{G}\Box\varphi \wedge \neg\mathsf{B}\varphi \longrightarrow \mathsf{E} \oslash \mathsf{behavior}_{\varphi}$

Specification & verification

specification φ vs. program $\mathcal{P}$

decomposition/refinement $\rightsquigarrow$ agent-oriented programming
verification $\rightsquigarrow$ model checking

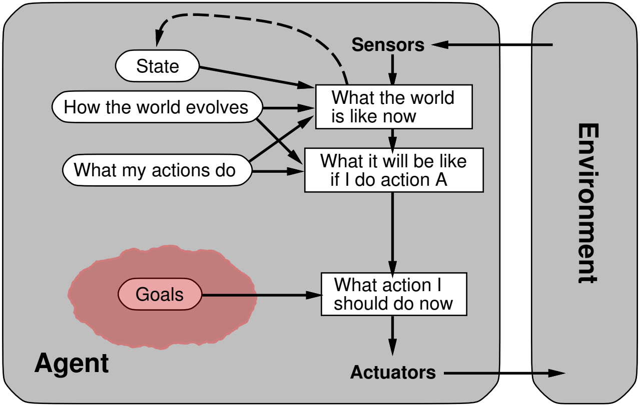
model checkers:

LTL: e.g., SPIN, etc.

CTL/CTL*: e.g., NuSMV, UPAAL, etc.

Agent-oriented programming

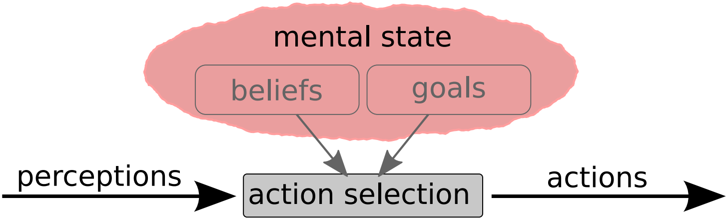
Goal-oriented agents



goals + state + actions' consequences $\rightsquigarrow$

action selection

Structure of cognitive agents



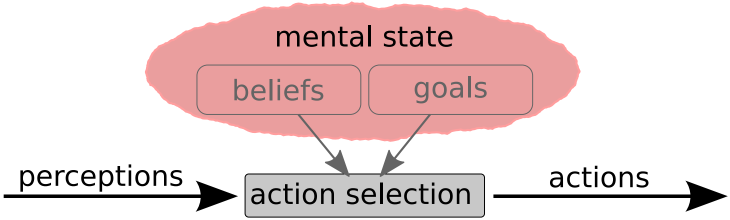
beliefs a database of agent's information about itself, the world (environment), other agents, etc.
⇝ NOW

goals description of states the agent "wants" to bring about
⇝ FUTURE

How to select actions leading from NOW to the FUTURE ?

⇝ **Planning!!!**

Structure of cognitive agents



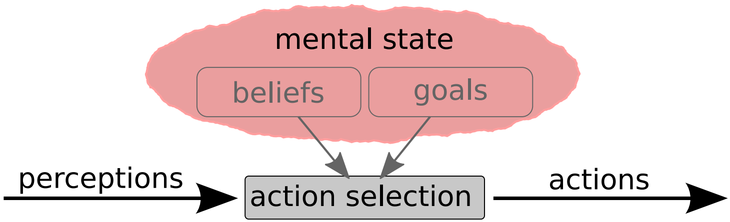
beliefs a database of agent's information about itself, the world (environment), other agents, etc.
⇝ NOW

goals description of states the agent "wants" to bring about
⇝ FUTURE

How to select actions leading from NOW to the FUTURE ?

⇝ **Planning!!!**

Structure of cognitive agents



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 ?

⇝ **Planning!!!**

Planning

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)

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)

The issue with planning

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

$\leadsto$ high computational complexity ($\approx$PSPACE) ☹

speed of planning vs. environment dynamics

planning $\overset{speed}{\succ}$ environment can perform relatively well

planning $\overset{speed}{\prec}$ environment can lead to fatal inefficiencies

$\leadsto$ the system "suffocates" in (re-)planning

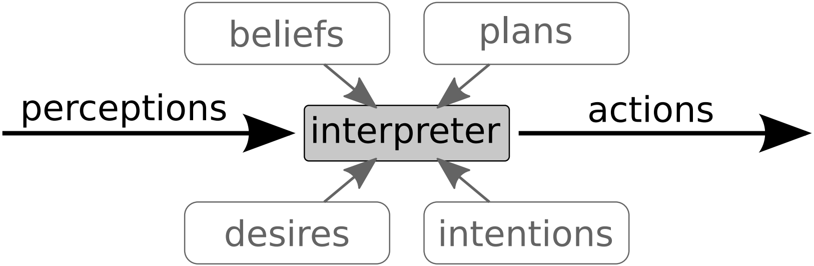
A way out: reactive planning & BDI

Structural decomposition:

- ***(B)eliefs:*** agent's static information about the world
- ***(D)esires:*** situations the agent wants to bring about
- ***(I)ntentions:*** courses of action, plans

System dynamics:

- ***reactive planning***: instead of plan-execute-monitor cycle, select partial plans reactively on the ground of the current state of the world, beliefs and goals



Agent-oriented programming

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

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

What can APLs do for us?

1. mentalistic abstractions for agent system specification
   - beliefs, desires, intentions, plans, practical reasoning rules, etc.,
   - operationalization of the BDI architecture
   - *tools for encoding the system dynamics*
2. agent-oriented language semantics
   - syntax & model of execution
   - loosely corresponds to temporal modal logics
3. means to tackle the pro-activity vs. reactivity problem
   - deliberation/planning vs. handling events & interruptions $\rightsquigarrow$ hybrid architectures

Historical overview

Hybrid architectures:	*– incomplete –*
1987: PRS	(Georgeff and Lansky)
1988: IRMA	(Bratman, Israel and Pollack)
1991: Abstract BDI architecture	(Rao and Georgeff)
1994: INTERRAP	(Müller and Pischel)

Agent-Oriented Programming Languages:	*– incomplete –*
1990: AGENT-0	(Shoham)
1996: AgentSpeak(L)	(Rao)
1996: Golog	(Reiter, Levesque, Lesperance)
1997: 3APL	(Hindriks et al.)
1998: ConGolog	(Giacomo, Levesque, Lesperance)
2000: JACK	(Busetta et al.)
2000: GOAL	(Hindriks et al.)
2002: Jason	(Bordini, Hubner)
2003: Jadex	(Braubach, Pokahr et al.)
2008: BSM/Jazzyk	(Novák)
2008: 2APL	(Dastani)

The landscape

BDI programming systems

Theoretically oriented

- − declarative languages built from scratch $\leadsto$ new syntax
- + clear theoretical properties $\leadsto$ verification
- + declarative KR techniques
- − difficult 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

BDI: the underlying principles

Structure of agent's internal state

- beliefs $\rightsquigarrow \mathcal{B}$
- goals $\rightsquigarrow \mathcal{G}$
- intentions/plans $\rightsquigarrow \mathcal{I}$ (optional)

+ an interface to the environment $\rightsquigarrow \mathcal{E}$

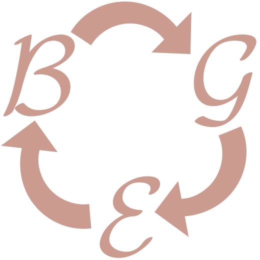
Minimal flow of information

1. agent perceives the environment and reflects it in the belief base
2. its beliefs about the world determine the goals it pursues
3. pursuing goals triggers behaviors aimed at fulfilling them

BDI: the underlying principles

Structure of agent's internal state

- beliefs $\rightsquigarrow \mathcal{B}$
- goals $\rightsquigarrow \mathcal{G}$
- intentions/plans $\rightsquigarrow \mathcal{I}$ (optional)
- \+ an interface to the environment $\rightsquigarrow \mathcal{E}$



Minimal flow of information

1. agent perceives the environment and reflects it in the belief base
2. its beliefs about the world determine the goals it pursues
3. pursuing goals triggers behaviors aimed at fulfilling them

Agent system architecture

$$\mathcal{A} = (\mathcal{B}, \mathcal{G}, \mathcal{E}, \mathcal{P})$$

robot in a 3D environment: search & deliver

Structure:

$\mathcal{B}$: belief base $(\models, \oplus, \ominus)$

$\mathcal{G}$: goal base $(\models, \oplus, \ominus)$

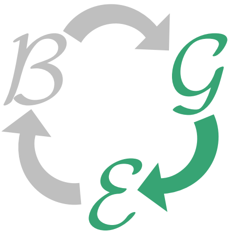
$\mathcal{E}$: interface to the environment $\rightsquigarrow$ body $(\models, \oslash)$

Basic capabilities:

FIND: $[\text{FIND}^*]\Diamond holds(item42)$

RUN_AWAY: $[\text{RUN\_AWAY}^*]\Diamond safe$

BD(I) design patterns: TRIGGER



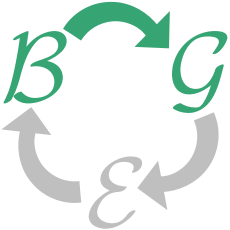
define TRIGGER($\varphi_{\mathbf{G}}, \tau$)
  when G$\models\varphi_{\mathbf{G}}$ **then** τ
end

running example (cont.)

TRIGGER($achieve(has(item42))$, FIND)

TRIGGER($maintain(keep\_safe)$, RUN_AWAY)

BD(I) design patterns: ADOPT/DROP

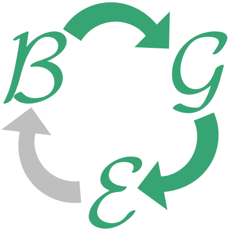


define ADOPT($\varphi_{\mathbf{G}}, \psi_{\oplus}$)
  when B⊨$\psi_{\oplus}$ **and not** G⊨$\varphi_{\mathbf{G}}$ **then** G $\oplus$ $\varphi_{\mathbf{G}}$
end

define DROP($\varphi_{\mathbf{G}}, \psi_{\ominus}$)
  when B⊨$\psi_{\ominus}$ **and** G⊨$\varphi_{\mathbf{G}}$ **then** G $\ominus$ $\varphi_{\mathbf{G}}$
end

BD(I) design patterns: ACHIEVE

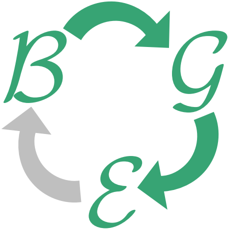
define ACHIEVE($\varphi_{\mathbf{G}}, \varphi_{\mathbf{B}}, \psi_{\oplus}, \psi_{\ominus}, \tau$)
TRIGGER($\varphi_{\mathbf{G}}, \tau$) |
ADOPT($\varphi_{\mathbf{G}}, \psi_{\oplus}$) |
DROP($\varphi_{\mathbf{G}}, \varphi_{\mathbf{B}}$) |
DROP($\varphi_{\mathbf{G}}, \psi_{\ominus}$)
end



running example cont.

ACHIEVE(
$achieve(has(item42))$,
$holds(item42)$,
$needs(item42)$,
$\neg needs(item42) \vee \neg exists(item42)$,
FIND)

BD(I) design patterns: MAINTAIN



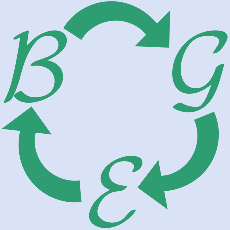
define MAINTAIN($\varphi_{\mathbf{G}}$, $\varphi_{\mathbf{B}}$, τ)
when not B$\models\varphi_{\mathbf{B}}$ **then** TRIGGER($\varphi_{\mathbf{G}}$, τ) |
ADOPT($\varphi_{\mathbf{G}}$, $\top$)
end

running example cont.

MAINTAIN($maintain(keep\_safe)$, $safe$, RUN_AWAY)

Putting it altogether

Robot program



PERCEIVE $\circ$
{
  MAINTAIN(
    $maintain(keep\_safe)$,
    $threatened$,
    RUN_AWAY) |

  ACHIEVE(
    $achieve(has(item42))$,
    $holds(item42)$,
    $needs(item42)$,
    $\neg needs(item42) \vee \neg exists(item42)$,
    FIND)
}

A4M33MAS/Lecture #4

Conclusion

Summary

1. Motivation & basic concepts
2. Agent-oriented software engineering
   - Introduction
   - Frameworks
   - Tropos methodology
   - Formal specification of agents
3. Agent-oriented programming
   - Introduction
   - Agent programming languages
   - BDI design patterns
4. Conclusion

Final thoughts

Agent-oriented software engineering

... provides a useful view on complex distributed systems. In the core, there is the idea of loose coupling of components and a strong emphasis on autonomy.

Agent-oriented programming

... is just one of the ways to tackle the problem of reactivity vs. deliberation.

- BDI architecture ⇝ modelling smart robotic and multi-robot systems

...both fields are a subject of an active on-going research, so the story is far from over.

Final thoughts

Agent-oriented software engineering

... provides a useful view on complex distributed systems. In the core, there is the idea of loose coupling of components and a strong emphasis on autonomy.

Agent-oriented programming

... is just one of the ways to tackle the problem of reactivity vs. deliberation.

- BDI architecture ⇝ modelling smart robotic and multi-robot systems

...both fields are a subject of an active on-going research, so the story is far from over.

The end

Thank you for your attention.

Questions?

Resources:

- ČVUT CourseWare: A4M33MAS
- http://www.troposproject.org/