Multiagent Systems

2. Abstract Architectures for Agents

B. Nebel, C. Becker-Asano, S. Wölfl

Albert-Ludwigs-Universität Freiburg

2nd of May 2014

Multiagent Systems

2nd of May 2014 — 2. Abstract Architectures for Agents

2.1 General information

General information

- Recommended reading:
  - Wooldridge, An Introduction to MultiAgent Systems - Second Edition, Wiley & Sons, 2009.
  - Russell & Norvig, Artificial Intelligence: A Modern Approach, third edition, Prentice Hall, 2010.
  - Bordini, Hübner, & Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason, Wiley & Sons, 2007
- Software:
  - JASON: `http://jason.sourceforge.net/wp/`
  - Intro to ROS: `http://www.ros.org/wiki/ROS/Introduction`

Website

Up-to-date information

www.informatik.uni-freiburg.de/~ki/teaching/ss14/multiagent-systems/

2.2 Agents (once again)

- Agents as intentional systems

What is an agent?

Definition 2 (Wooldridge, p. 21)

An agent is a computer system that is **situated** in some **environment**, and that is capable of **autonomous action** in this environment in order to meet its **design objectives**

- Adds the notion of free will or **intention** to agent design
- When explaining human activity, we use statements like the following:

  > Janine took her umbrella because she believed it was raining and she wanted to stay dry. (Wooldridge)

- **folk psychology** used to explain human behavior based on **attitudes** such as *believing*, *wanting*, *hoping*, *fearing*, ...

The (virtual) agent MAX

MAX, the Multimodal Assembly eXpert:

- developed at the VR and AI group at Bielefeld University since 2003
- since 2007 promoted in the Cluster of Excellence CITEC

Figure: The MAX agent, taken from http://www.excellence-initiative.com/bielefeld-cognitive-interaction-technology

Some applications of multiagent systems: MAX?

Two major areas of application:

- Distributed systems (agents as processing nodes)
- Personal software assistants (aiding the user)

A variety of subareas:

- Workflow/business process management
- Distributed sensing
- Information retrieval and management
- Electronic commerce
- Human-computer interfaces
- Virtual environments
- Social simulation
- ...

Intentional Systems

Daniel Dennet coined the term **intentional system** to describe entities "whose behavior can be predicted by the method of attributing belief, desires and rational acumen".
(Dennett, 1987; after Wooldridge, p. 31)

"A first-order intentional system has beliefs and desires (etc.) but no beliefs and desires about beliefs and desires. ... A second-order intentional system is more sophisticated; it has beliefs and desires (and no doubt other intentional states) about beliefs and desires (and other intentional states) – both those of others and its own." (Dennet, 1987, p. 243)

Intentional stance applied to a light switch?

Intentional stance $\Rightarrow$ ascribing **beliefs, free will, intentions, consciousness, abilities** or **wants** to others, even to machines.

"It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires."
But: "...it does not *buy us anything*, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behavior." (Yoav Shoham, 1990)

So then, why Agents?

- The more we know about a system, the less we need to rely on animistic, intentional explanations of its behavior
- But with very complex systems, a mechanistic explanation may not be practicable
- Thus, we use intentional notions as **abstraction tools** providing us with a **convenient** and **familiar** way to describe, explain, and predict the behavior of complex systems
- Abstractions commonly used in computer science:
  - procedural abstraction
  - abstract data types
  - objects

  Agents and agents as intentional systems represent just another powerful abstraction

2.3 Abstract Architectures for Agents

- Standard agents
- State-based agents
- Utility
- Expected Utility
- Special types of tasks

States and Actions

Assume the environment may be in any of a finite set E of discrete, instantaneous states:

$$E = \{e, e', \ldots\}.$$

Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment.

$$Ac = \{\alpha, \alpha', \ldots\}$$

A **run**, r, of an agent in an environment E is a sequence of interleaved environment states and actions:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

Runs

Let ...

- $\mathcal{R}$ be the set of all such possible **finite sequences** (over E and Ac);
- $\mathcal{R}^{Ac}$ be the subset of these that **end with an action**; and
- $\mathcal{R}^{E}$ be the subset of these that **end with an environment state**.

Then the **state transformer function** τ represents behavior of the environment.

Definition 3: State transformer function τ

The state transformer function τ maps each run $r \in \mathcal{R}^{Ac}$ to a subset of E (even the empty set):

$$\tau : \mathcal{R}^{Ac} \to \mathcal{P}(E)$$

(from runs to environment states)

(with $\mathcal{P}(E)$ denoting the **power set** of E.)

Environments

An environment *Env* is then defined as follows:

Definition 4: Environments

An environment Env is given as the triple $Env = \langle E, e_0, \tau \rangle$ where

- E is the set of environment states,
- $e_0 \in E$ is the initial state, and
- τ is the state transformer function.

Note that **environments** are:

- history dependent
- non-deterministic

If $\tau(r) = \emptyset$, there are no possible successor states to r, so we say the run has **ended**.

Agents

Definition 5: Agent Ag

An agent Ag is a function which maps any run $r \in \mathcal{R}^E$ to an action $\alpha \in Ac$:

$$Ag : \mathcal{R}^E \to Ac$$

(from runs to actions)

- Agents choose actions depending on (environment) states
- With AG defined as the **set of all agents**, a **system** is defined as the pair (Ag, Env) with $Ag \in AG$
- Denote **runs** of a system by $\mathcal{R}(Ag, Env)$ and assume they are all terminate (and thus finite)

Behavioral equivalency

Definition 6: Behavioral equivalence

Two agents Ag_1 and Ag_2 are called **behavioral equivalent** with respect to **environment** Env iff

$$\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$$

If this is true for **any environment** Env, then they are simply called **behaviorally equivalent**

Putting it all together now

Formally, a sequence

$$(e_o, \alpha_0, e_1, \alpha_1, e_2, \ldots)$$

represents a **run of agent** Ag in environment $Env = \langle E, e_0, \tau \rangle$ if:

1. e_0 is the initial state of Env
2. $\alpha_0 = Ag(e_0)$; and
3. for $u > 0$,

$$e_u \in \tau((e_o, \alpha_0, \ldots, \alpha_{u-1})) \quad \text{where}$$
$$\alpha_u = Ag((e_0, \alpha_0, \ldots, e_u))$$

Purely reactive agents

A **purely reactive agent**:

- bases its decision only on the present state of the environment
- does **not** take history into account
- is an example of the "Behaviorist" model of activity, in that actions are solely based on stimulus-response schemata

Definition 7: Purely reactive agent

A purely reactive agent Ag_r maps the current state $e \in E$ to an action $\alpha \in Ac$:

$$Ag_r : E \to Ac$$

Purely reactive agent example

Properties of purely reactive agents:

- Every purely reactive agent can be mapped to an agent defined on **runs**, i.e. a **standard agent**
- The reverse is usually **not** true

Example: (old-style, non-NEST) thermostat

- Two environment states e_0 = "temperature OK" and e_1 = "temperature not OK"
- Ag defined as:

$$Ag(e) = \begin{cases} \text{heater off}, & \text{if } e = e_0 \\ \text{heater on}, & \text{if } e = e_1 \end{cases}$$

Perception and action

Agent model so far rather simple, but still many design choices need to be made to achieve **concrete agent architectures**

- data structures?
- operations on them?
- control flow?

Do you remember this Figure?

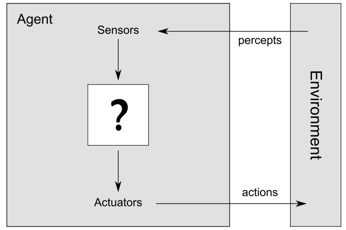

Figure: An agents interacts with an environment through sensors and actuators (after Russel & Norvig, p. 35)

Perception

Perception can be modeled as follows:

- Define function $see : E \to Per$ and $action : Per^* \to Ac$ where:
  - Per is non-empty set of **percepts** that the agent can obtain through its sensors
  - see describes process of perception and $action$ defines **decisions** based on **percept sequences**
- Agent definition now becomes $Ag = \langle see, action \rangle$

If $e_1 \neq e_2 \in E$ and $see(e_1) = see(e_2)$ we call e_1 and e_2 **indistinguishable**

Perception example

- Let x = 'the room temperature is OK' and y = 'Merkel is chancelor' be the only two facts that describe environment
- Then we have $E = \{\underbrace{\{\neg x, \neg y\}}_{e_1}, \underbrace{\{\neg x, y\}}_{e_2}, \underbrace{\{x, \neg y\}}_{e_3}, \underbrace{\{x, y\}}_{e_4}\}$
- If **percepts** of thermostat are p_1 (too cold) and p_2 (OK), then **indistinguishable states** occur (unless Merkel makes room chilly)

$$see(e) = \begin{cases} p_1, & \text{if } e = e_1 \lor e = e_2 \\ p_2, & \text{if } e = e_3 \lor e = e_4 \end{cases}$$

- We write $e \sim e'$ (equivalence relation over states)
- The coarser these equivalence relations, the less effective is perception (if $|\sim| = |E|$, then the agent is **omniscient**)

Perception and action, state-based agents (1)

Three new functions:

1. the *see* function, the agent's ability to perceive its environment

Definition 8: The *see* function
It maps environment states $e \in E$ to percepts $p \in Per$:

$$see : E \rightarrow Per$$

2. the *action* function to represent the agent's (internal) decision making

Definition 9: The *action* function
It maps internal states $i \in I$ to actions $\alpha \in Ac$:

$$action : I \rightarrow Ac$$

3. a function *next* to update the agent's internal state-based on the current percept

Perception and actions, state-based agents (2)

Definition 10: The *next* function
It maps an internal state $i_{old} \in I$ and a percept $p \in Per$ to a new internal state $i_{new} \in I$:

$$next : I \times Per \rightarrow I$$

The behavior of a **state-based agent** is described as follows:

1. The agent starts in some initial state e_0
2. After perceiving environment state e it generates a percept $p = see(e)$
3. Its internal state is updated by $next(i_0, p)$
4. Finally, the agent chooses an action calculating the result of $action(next(i_0, p))$
5. Loop!

State-based agents

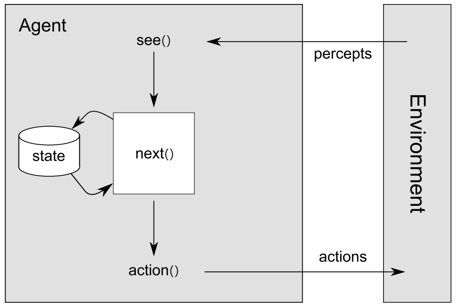

Figure: An agent that maintains a state (after Wooldrige, p. 37, and Russel & Norvig, p. 35)

⇒ State-based agents are **no more expressive** than **standard agents**.
They are **behaviorally equivalent**!
(Wooldridge, p. 38)

Task specification & utility

Agents should perform a task on our behalf:

- Task specified by us
- Tell agent **what** to do, but **not how** (exactly)
- How can the agent choose among alternative actions?

⇒ Utility functions over states
The agent has to bring about states that maximize utility.
First possibility:

Definition 11: Task specification
A **task specification** is a function u associating a real number with every environment state:

$$u : E \rightarrow \mathbb{R}$$

Utilities over Runs

With **task specification**, what is the utility of a run?

- minimum utility of visited states?
- maximum utility of visited states?
- Average utility of visited states?
- ...

Better idea:

Definition 12: Utility over Runs

Utility is assigned to runs:

$$u : \mathcal{R} \to \mathbb{R}$$

Takes a **long term view** and can be **extended by** incorporating **probabilities** of different states emerging into account.

Problems with Utility-based Approaches

Certain problems have been discussed in the literature:

- Where do the numbers come from?
- People don't think in terms of utilities $\Rightarrow$ difficult to specify tasks in these terms

Nevertheless, certain scenarios can be modeled with utilities.

The Tileworld

- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes.
- Agent can move in four directions, up, down, left, or right.
- If agent is located next to a tile, it can push it.
- Goal: Agent has to fill as many holes with tiles as possible.
- The more holes are filled the higher the score.
- TILEWORLD changes with random appearance and disappearance of holes.

	HOLE		
	↑		
	TILE		
	Ag	TILE	HOLE

Utility in the Tileworld

Utility function defined as follows:

$$u(r) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

Thus:

- If agent fills **all** holes $\rightarrow$ utility $= 1$.
- If agent fills **no** holes $\rightarrow$ utility $= 0$.

Expected Utility of an agent

Let $P(r|Ag, Env)$ denote the **probability** that run r occurs when agent Ag is placed in environment Env.
Note:

$$\sum_{r \in \mathcal{R}(Ag,Env)} P(r|Ag, Env) = 1$$

Definition 13: Expected utility over runs

The expected utility EU of an agent Ag in environment Env (given P, u) is:

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag,Env)} u(r)P(r|Ag, Env).$$

Give example on blackboard

Optimal agents

Now we can define the **optimal agent** in an environment Env.

Definition 14: The Optimal Agent

The optimal agent Ag_{opt} in an environment Env is defined as the one that maximizes expected utility:

$$Ag_{opt} = arg \max_{Ag \in \mathcal{AG}} EU(Ag, Env)$$

Of course, the fact that it is **optimal** does not mean it **will** always be best; only that **on average**, we can expect it to do best.

Bounded optimal agents

Not every conceivable function $Ag : \mathcal{R}^E \to Ac$ can be implemented on a machine.
$\Rightarrow$ Define the class of bounded optimal agents:

Definition 15: Bounded optimal agents

Let
$\mathcal{AG}_m = \{Ag | Ag \in AG \wedge Ag \text{ implementable on machine } m\}$.
Then the **bounded optimal agent**, Ag_{bopt}, is defined with respect to m:

$$Ag_{bopt} = arg \max_{Ag \in \mathcal{AG}_m} EU(Ag, Env)$$

Predicate task specifications

Often more natural to define a **predicate** over runs:

- Idea: only assign **success** or **failure** to runs
- Assume u ranges over $\{0,1\}$, then run $r \in \mathcal{R}$ **satisfies** a task specification if $u(r) = 1$, else it **fails**

Define:

- $\Psi(r)$ iff $u(r) = 1$ and **task environment** $\langle Env, \Psi \rangle$ with $\mathcal{TE}$ the **set of all task environments**
- Let $\mathcal{R}_\Psi(Ag, Env) = \{r | r \in \mathcal{R}(Ag, Env) \wedge \Psi(r)\}$ be the set of runs of agent Ag that satisfy Ψ
  - Ag **succeeded** in task environment $\langle Env, \Psi \rangle$ iff $\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env)$
  - More **optimistic**, we may just require that $\exists r \in \mathcal{R}(Ag, Env)$ such that $\Psi(r)$

Extend state transformer function by probabilities, then:

$$P(\Psi | Ag, Env) = \sum_{r \in \mathcal{R}_\Psi(Ag, Env)} P(r | Ag, Env)$$

Achievement and maintenance tasks

Two very common types of tasks:

- "achieve state of affairs φ"
- "maintain state of affairs φ"

Achievement tasks:

- are defined by a set of **good states** $\mathcal{G} \subseteq E$.
- The agent **succeeds** if it is guaranteed to bring about **at least one** of these states.

Maintenance tasks:

- are defined by a set of **bad states** $\mathcal{B} \subseteq E$.
- The agent **succeeds** if it manages to avoid all states in $\mathcal{B}$.

More complex combinations exist.

2.4 Summary

- Thanks

Summary

- Discussed intentional stance & agents
- Introduced abstract agent architectures
- Environments, perception & action
- Purely reactive agents & agents with state
- Utility-based agents
- Task-based agents, achievement & maintenance tasks

$\Rightarrow$ Next time: **Deductive reasoning agents**

Acknowledgments

These lecture slides are partly based on the following slides:

- Dr. Michael Rovatsos, The University of Edinburgh
  http://www.inf.ed.ac.uk/teaching/courses/abs/abs-timetable.html
- Prof. Micheal Wooldridge, University of Oxford
  http://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/imas/distrib/pdf-index.html