EH2750 Computer Applications in Power Systems, Advanced Course.

Lecture 2

Professor Lars Nordström, Ph.D.
Dept of Industrial Information & Control systems, KTH
larsn@ics.kth.se

Acknowledgement

- These slides are based largely on a set of slides provided by:

  Professor Rosenschein of the Hebrew University Jerusalem, Israel

  and

  Dr. Georg Groh, TU-München, Germany.

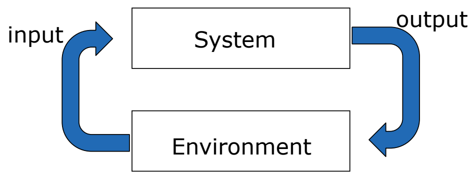
- Available at the Student companion site of the Introduction to Multi Agent Systems book

Outline of the Lecture

- Agent definition – a closer look *(Ch 2.1)*
- Beliefs Desires & Intentions –BDI *(Ch 2.4)*
- Formalising the agents *(Ch 2.5)*
- Agent decision making – Utility *(Ch 2.6)*
- Agent reasoning – deduction *(Ch 3)*

What is an Agent?

- The main point about agents is they are *autonomous*: capable of acting independently, exhibiting control over their internal state
- Thus: *an agent is a computer system capable of autonomous action in some environment in order to meet its design objectives*



Intelligent Agents

- Examples of agents that fit the definition:
  - Thermostat
  - UNIX daemon, Windows services
  - Controllers
- An intelligent agent is a computer system capable of flexible autonomous action in some environment in order to meet its design objectives
- With flexible, we mean:
  - reactive
  - pro-active
  - social

What does Reactive mean?

- If the environment is static, the program can execute as planned, for example
  - Parsing text-files
  - Compiling sourcecode into executable code.
- The real world is however dynamic
- It is difficult to build software program that accepts failure and constantly revises its "mission"
- A reactive system is one that keeps interacting with the environment constantly in order to determine if a certain action is appropriate – **this is very much a timing issue**

Proactive then, what's that

- Reacting to an environment is easy
  - Thermostat (again)
- But we want agents to do things for us, not just waiting for changes in the environment, we want them to be goal directed
- Pro-activeness is then the ability to generate and **work towards goals** not just waiting for a change.
- The simpler case is that we set the goal for the agent at design time.

Goal-oriented vs. Reactive behaviour

We want our agents to be reactive, responding to changing conditions in an appropriate (timely) fashion

and

We want our agents to systematically work towards long-term goals

- This is the same problem we humans face, long term goal or short-term reaction?
- These two considerations can be at odds with one another, and design this remains a open question for research and design.

ρ

Social then, what's that about?

- The real world is a multi-agent environment, remember the definition of MAS:

A multiagent system is one that consists of a number of agents, which interact with one-another. To successfully interact, they will require the ability to cooperate, coordinate, and negotiate with each other, much as people do

- Social ability in agents is the ability to interact with other agents to negotiate, cooperate and share information

Environments
Accessible vs. *inaccessible*

- An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state
- Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible
  - Subsets of the real-world can of course be made accessible
  - Measurements in a Power grid (U,I,P,Q, states, φ etc)
- The more accessible an environment is, the simpler it is to build agents to operate in it

Environments –
Deterministic vs. *non-deterministic*

- A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action
- The physical world can to all intents and purposes be regarded as non-deterministic
  - Again, subsets of the real world can appear deterministic
- Non-deterministic environments present greater problems for the agent designer

Environments
Episodic vs. *non-episodic*

- In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios
- Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes

Environments
Static vs. dynamic

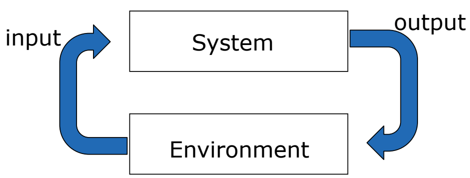
- A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent
- A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control
- Other processes can interfere with the agent's
- The real world is obviously a highly dynamic environment
  - But is a distribution grid a highly dynamic environment?

Environments
Discrete vs. *continuous*

- An environment is discrete if there are a fixed, finite number of actions and percepts in it
- A chess game is an example of a discrete environment, and taxi driving an example of a continuous one
- Continuous environments have a certain level of mismatch with computer systems
- Discrete environments could *in principle* be handled by a kind of "lookup table"

What is an Agent?

- The main point about agents is they are *autonomous*: capable of acting independently, exhibiting control over their internal state
- An intelligent agent is a computer system capable of flexible autonomous action in some environment in order to meet its design objectives



Outline of the Lecture

- Agent definition – a closer look *(Ch 2.1)*
- Beliefs Desires & Intentions –BDI *(Ch 2.4)*
- Formalising the agents *(Ch 2.5)*
- Agent decision making – Utility *(Ch 2.6)*
- Agent reasoning – deduction *(Ch 3)*

Describing things

- How do you best describe the event of holding a stone in your hand and dropping it? Which terms do you use to explain the event?

- Concepts like:
  - Mass
  - Gravity
  - Force
- Are useful terms (obviously)

Descriptions like this is based on a *physical stance*

Describing things

- How do you best describe a computer programs execution of a control loop that suggest you to buy a pink striped shirt?

- Concepts like:
  - Thinks
  - Says
  - Asks
  - ”The computer asked if I was older than 40 and now it thinks I like pink shirts”

Agents as Intentional Systems

- When explaining human activity, it is often useful to make statements such as the following:

  "Janine took her umbrella because she believed it was going to rain and she did not want to ruin her hair."

- These statements make use of a *folk psychology*, by which human behavior is predicted and explained through the attribution of *attitudes*, such as **believing** and **desiring** like wanting (as above), hoping, fearing, and so on

- The attitudes employed in such folk psychological descriptions are called the **intentional** notions

19

Beliefs, Desires & Intentions - BDI

- When we describe Intelligent Agents it is convenient to talk about them as if they have:
  - Beliefs
    - Some image of the environment
    - E.g. Temperature measurement
  - Desires
    - Goals they wish to achieve
    - E.g Increase temperature
  - Intentions
    - Actions that the agent can take
    - Means by which to do something
    - Opening hot water valve

Outline of the Lecture

- Agent definition – a closer look *(Ch 2.1)*
- Beliefs Desires & Intentions –BDI *(Ch 2.4)*
- Formalising the agents *(Ch 2.5)*
- Agent decision making – Utility *(Ch 2.6)*
- Agent reasoning – deduction *(Ch 3)*

Formalised view of Agents

Abstract Architecture for Agents

- 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 is a sequence of interleaved environment states and actions:

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

Abstract Architecture for Agents

- 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
  - $\mathcal{R}^{E}$ be the subset of these that end with an environment state

State Transformer Functions

- A state transformer function represents behavior of the environment: $\tau : \mathcal{R}^{Ac} \to \wp(E)$
- Note that environments are…
  - *history dependent*
  - *non-deterministic*
- If $\tau(r)=\varnothing$, then there are no possible successor states to r. In this case, we say that the system has *ended* its run
- Formally, we say an environment Env is a triple $Env = \langle E, e_0, \tau \rangle$ where: E is a set of environment states, $e_0 \in E$ is the initial state, and τ is a state transformer function

Agents

- Agent is a function which maps runs to actions:

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

- An agent makes a decision about what action to perform based on the history of the system that it has witnessed to date. Let AG be the set of all agents

Systems

- A *system* is a pair containing an agent and an environment
- Any system will have associated with it a set of possible runs; we denote the set of runs of agent Ag in environment Env by $R(Ag, Env)$
- (We assume $R(Ag, Env)$ contains only *terminated* runs)

Systems

- Formally, a sequence

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$$

represents a run of an 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 $u > 0$,

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

Why are we talking about this?

Agents are implemented as software, i.e. Source code programmed by someone to execute on a computer – **so it's just a program!?!**

Well, we want to make sure that the program works as intended, that no circuit breakers are opened when they should not be.

So, we need to make sure that our design of this program is correct and complete and at the same time efficient– right?

Therefore, we need a rigid (almost formal) way to talk about and design the program/software/agent

Purely Reactive Agents

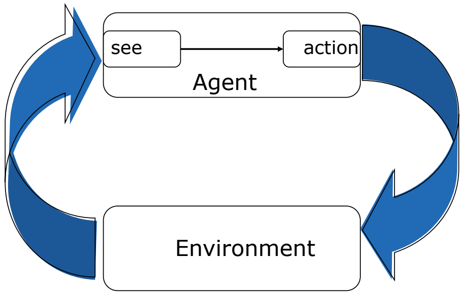
- Some agents decide what to do without reference to their history — they base their decision making entirely on the present, with no reference at all to the past
- We call such agents *purely reactive*:
- A thermostat is a purely reactive agent

$$action : E \to Ac$$

$$action(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise.} \end{cases}$$

Perception

- Now introduce the perception system:



Perception

- The *see* function is the agent's ability to observe its environment, whereas the *action* function represents the agent's decision making process
- *Output* of the *see* function is a *percept*:

$$see : E \to Per$$

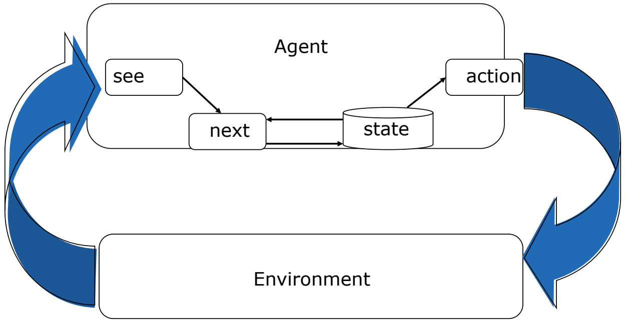
which maps environment states to percepts, and *action* is now a function

$$action : Per^* \to Ac$$

which maps sequences of percepts to actions

Agents with State

- We now consider agents that *maintain state*:



Agents with State

- These agents have some internal data structure, which is typically used to record information about the environment state and history.
  Let I be the set of all internal states of the agent.
- The perception function see for a state-based agent is unchanged:

$$see : E \rightarrow Per$$

The action-selection function $action$ is now defined as a mapping

$$action : I \rightarrow Ac$$

from internal states to actions. An additional function $next$ is introduced, which maps an internal state and percept to an internal state:

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

Agent Control Loop

1. Agent starts in some initial internal state i_0
2. Observes its environment state e, and generates a percept $see(e)$
3. Internal state of the agent is then updated via $next$ function, becoming $next(i_0, see(e))$
4. The action selected by the agent is $action(next(i_0, see(e)))$
5. Goto 2

Outline of the Lecture

- Agent definition – a closer look *(Ch 2.1)*
- Beliefs Desires & Intentions –BDI *(Ch 2.4)*
- Formalising the agents *(Ch 2.5)*
- Agent decision making – Utility *(Ch 2.6)*
- Agent reasoning – deduction *(Ch 3)*

Tasks for Agents

- We build agents in order to carry out *tasks* for us
- The task must be *specified* by us…
- But we want to tell agents what to do *without* telling them how to do it

Utility Functions over States

- One possibility: associate *utilities* with individual states — the task of the agent is then to bring about states that maximize utility
- A task specification is a function

$$u : E \to \mathbf{R}$$

which associates a real number with every environment state

Utility Functions over States

- But what is the value of a *run…*
  - minimum utility of a state on the run?
  - maximum utility of a state on the run?
  - sum of utilities of states on run?
  - average?
- Disadvantage: difficult to specify a *long term* view when assigning utilities to individual states

Utilities over Runs

- Another possibility: assigns a utility not to individual states, but to runs themselves:

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

- Such an approach takes an inherently long term view
  - We watch several runs and evaluate which is the best
  - Assumes that the environment is in some way predicatable
- Other variations: incorporate probabilities of different states emerging
- Difficulties with utility-based approaches:
  - where do the numbers come from?
  - we don't think in terms of utilities!
  - hard to formulate tasks in these terms

Tileworld example

- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes
- An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it
- Holes have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, with the aim being to fill as many as possible.

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

Expected Utility & Optimal Agents

- Write $P(r \mid Ag, Env)$ to denote probability that run r occurs when agent Ag is placed in environment Env
  Note:

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

- Then optimal agent $\mathrm{Ag_{opt}}$ in an environment Env is the one that maximizes expected utility:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}} \sum_{r \in \mathcal{R}(Ag,Env)} u(r) P(r \mid Ag, Env). \qquad (1)$$

Predicate Task Specifications

- A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run
- If a run is assigned 1, then the agent succeeds on that run, otherwise it fails
- Call these *predicate task specifications*
- Denote predicate task specification by Ψ.

$\Psi : R \rightarrow \{0, 1\}$.

Task Environments

- A *task environment* is a pair $\langle Env, \Psi \rangle$ where Env is an environment,

  $\Psi : R \rightarrow \{0, 1\}$

  is a predicate over runs.
  Let TE be the set of all task environments.
- A task environment specifies:
  - the properties of the system the agent will inhabit
  - the criteria by which an agent will be judged to have either failed or succeeded

Task Environments

- Write $R_\Psi(Ag, Env)$ to denote set of all runs of the agent Ag in environment Env that satisfy Ψ:

- We then say that an agent Ag succeeds in task environment $\langle Env, \Psi \rangle$ if

$$\mathcal{R}_\Psi(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r) = 1\}.$$

$$\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env)$$

- Meaning that all possible runs fulfill the statement

The Probability of Success

- Let $P(r \mid Ag, Env)$ denote probability that run r occurs if agent Ag is placed in environment Env
- Then the probability $P(\Psi \mid Ag, Env)$ that Ψ is satisfied by Ag in Env would then simply be:

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

46

Outline of the Lecture

- Agent definition – a closer look *(Ch 2.1)*
- Beliefs Desires & Intentions –BDI *(Ch 2.4)*
- Formalising the agents *(Ch 2.5)*
- Agent decision making – Utility *(Ch 2.6)*
- Agent reasoning – deduction *(Ch 3 (only 3.1))*

Agent Architectures

- We want to build agents, that enjoy the properties of autonomy, reactiveness, pro-activeness, and social ability that we talked about earlier
- This is the area of *agent architectures*
- Maes defines an agent architecture as:
  '[A] particular methodology for building [agents]. It specifies how… the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions… and future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology.'

Agents with State - repeated

- These agents have some internal data structure, which is typically used to record information about the environment state and history.
  Let I be the set of all internal states of the agent.
- The perception function *see* for a state-based agent is unchanged:

$$see : E \rightarrow Per$$

The action-selection function *action* is now defined as a mapping

$$action : I \rightarrow Ac$$

from internal states to actions. An additional function *next* is introduced, which maps an internal state and percept to an internal state:

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

So, how do we make the agent think?

- One straightforward way is to use logic
- Program the agent to be completely logical and use deduction to prove it's way to chosing which action to perform.

```
function action(i:I) returns α:A {
     for each α in A do {
          if(i using ρ proves Do(α)) {
               return α
          }
     }
     for each α in A do {
          if(i using ρ does not prove NOT(Do(α))) {
               return α
          }
     }
     return null
}
```

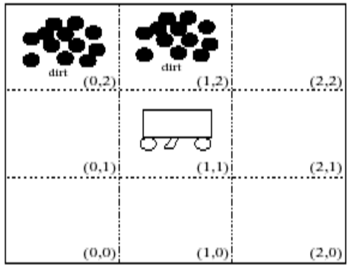
?

Example: The Vacuum World I

Agent's objective:
suck up all dirt

Possible actions: **α**
A={turn, forward, suck}
(*turn* = turn right 90 degrees)

Domain-Predicates (Facts)
In(x,y) Dirt(x,y) Facing(d)
(*d* from *{south, north, west, east}*)



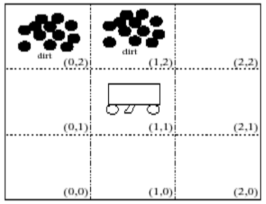
Agent's *next* function is: $next(\Delta, p) = \Delta \setminus old(\Delta) \cup new(\Delta, p)$

where $old(\Delta) = \{P(t_0, t_1, ..) \mid P(t_0, t_1, ..) \in \Delta \wedge P \in \{In, Dirt, Facing\}\}$

and $new : D \times Per \rightarrow D$ computes new Facts

Example: The Vacuum World II

- Agents database-rules:



Objective:

$$\underbrace{In(x,{}^{i}y) \wedge Dirt(x,{}^{i}y) \rightarrow Do(s{}^{i}uck)}_{\rho}$$

Traversal:

$$In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) \longrightarrow Do(forward)$$
$$In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) \longrightarrow Do(forward)$$
$$In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) \longrightarrow Do(turn)$$
$$In(0,2) \wedge Facing(east) \longrightarrow Do(forward)$$

and for all other rows accordingly

Deductive Agents – does that work?

- The idea of prooving theorems as a way of making decisions is logically sound and rigouros

Two challenges remain:

1. It is time consuming to program
2. It is time consuming to execute

- Applied in a human setting it is also rather rigid. Imagine a theorem:
  - I will buy the cheapest copy of Wooldrdige's book.
- Requires you to find a copy, check the price
  - Find next copy check price
  - Etc. until you have found all copies of the book
- People tend to use Practical reasoning

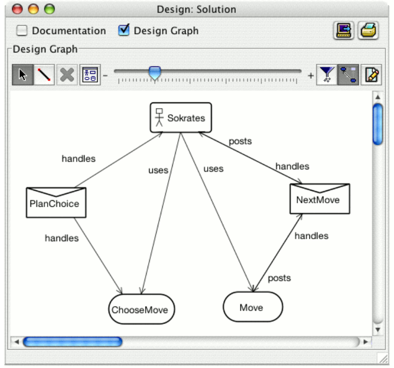
Outline of the Lecture

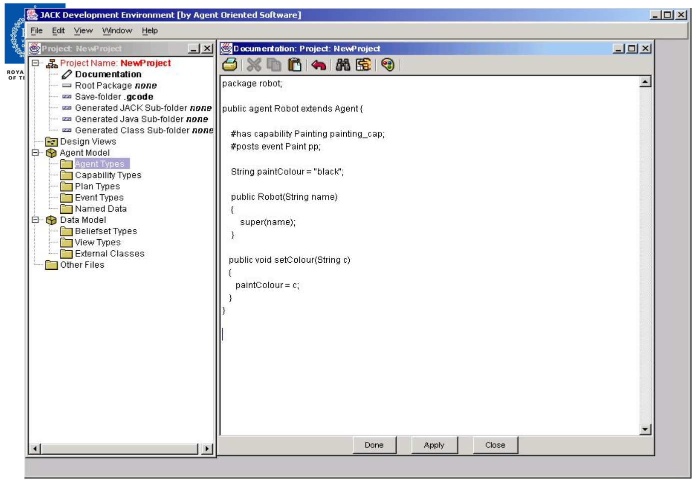
- Agent definition – a closer look *(Ch 2.1)*
- Beliefs Desires & Intentions –BDI *(Ch 2.4)*
- Formalising the agents *(Ch 2.5)*
- Agent decision making – Utility *(Ch 2.6)*
- Agent reasoning – deduction *(Ch 3)*

What is JACK

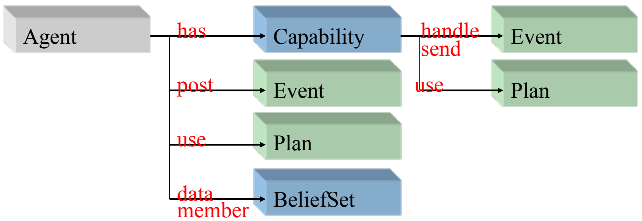
JACK Intelligent Agents is an *environment* for building, running and integrating commercial *Java-based* multi-agent software using a *component-based* approach.

Beginner friendly





JACK Architecture



Multiagent Systems in Power Systems

- In Multiagent Systems, we address questions such as:
  - How can cooperation emerge in societies of self-interested agents?
  - What kinds of languages can agents use to communicate?
  - How can self-interested agents recognize conflict, and how can they (nevertheless) reach agreement?
  - How can autonomous agents coordinate their activities so as to cooperatively achieve goals?