MULTIAGENT SYSTEMS NAIL106

Roman Neruda, Martin Pilát
MFF UK Praha, 2018

Bibliography:

- M. Wooldridge: An Introduction to Multiagent Systems, (2nd ed), 2009.
- S. Russell, P. Norvig: Artificial Intelligence: A Modern Approach, (3rd ed), 2009.
- G. Weiss (ed): Multiagent Systems (2nd ed), 2013.
- Y. Shoham, K. Leyton-Brown: Multiagent systems: Algorithmic, Game-Theoretic, and Logical Foundations, 2009. [online]
- David Ensley, Jon Kleinberg: Networks, Crowds, and Markets: Reasoning about a highly connected word [online]

Topics

- Agents, environments, abstract architectures.
- Reactive a planning agents, hybrid architectures.
- Logic and reasoning based agents, BDI.
- Communication, speech acts, ACL.
- Ontologies, OWL, KIF.
- Distributed problem solving, cooperation.
- Multiagent interactions, Nash equilibria, Pareto efficiency.
- Resource alloacation, auctions, negotiation.
- MAS design methodologies, Gaia, roles.
- MAS languages and environments, JADE.

What is not (much) covered

- What exactly is an agent in different contexts
- Agent learning and MAS learning
- Modal, temporal logics and similar interesting math-logical issues
- Robotics
- Planning
- Artificial life simmulations
- ABM - Agent Based Model and applications in sociology, economy, …

The seminars

- BattleCode
  - The Battlecode programming competition is a unique challenge that combines battle strategy, software engineering and artificial intelligence.
- JADE
  - JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases.
- AgentSpeak & Jason
  - AgentSpeak is an agent-oriented programming language. It is based on logic programming and the BDI architecture for (cognitive) autonomous agents.

1. Introduction

Visions, specifics, libels of related fields, basic notions

Propaganda or trends of informatics

- Ubiquity
  - From computer halls to tables, to pockets and fridges
- Connectivity and distributivity
  - Internet, clouds
  - Computing as interaction(s)
- Intelligence
  - Complexity of problems to solve is rising
- Delegation
  - Web searches, fly-by-wire
- Ease of use for humans
  - GUI, personalization

Multiagent systems

- A new computer science field has risen in 90s
- Agent is a (computer) system capable to work independently on behalf of its user
- Multiagent system consists of agents that are communicating with each other, most often by exchanging messages over a computer network
- How to create such independent agents able to perform delegated tasks?
- How to create a system of agents with cooperation, coordination and negotionation?

Software engineering

- Agents as one of possible paradigms of programming:
  - Machine code/assembler
  - Structured programs
  - Objects
  - CORBA – distributed objects
  - …
- Mastering complex distributed systems of communicating components
- Objects do it for free, agents because they want to.

Distributed, concurrent systems

- In distributed systems area the research (for decades) deals with theory, programming languages and methods for description and design of systems with multiple components. Many problems have been successfully identified and solved
  - Mutex, deadlock, …

- Agents are automous, the coordination mechanisms are not fully defined beforehand, everything is happening – and must be solved – in run time.
- Agenti have their own interests, they do not have to share a common goal. It is necessary to study mechanisms of negotiations and dynamic coordination.

Artificial intelligence

- Traditional view:
  - MAS are part of AI.
- Russell, Norvig (AIMA):
  - The goal of AI is the design of intelligent agents.
- Bold view:
  - AI is part of MAS.
- Hardcore view (Etzioni):
  - MAS is 1% AI, 99% computer science.
- MAS use AI techniques (knowledge representation, planning)
- MAS emphasize social aspects (cooperation, negotiation), which were overlooked by traditional AI.

Game theory, economics

- Already von Neumann and Turing in 40s …
- Game theory is an important theoretical approach to study MAS
  - Nash equilibrium, …

- MAS question (will question?) the notion of rational agent, which is a key concept of game theory.
- Mathematical foundations of game theory deal mostly with existence questions, MAS emphasize computational aspects (complexity, practical usage).

Sociology

- The key concept for MAS are agent societies.
- Sociology studies human societies.
- Similarly to AI and human intelligence, MAS can seek inspirations in sociology.
- But not necessarily (cf. e.g. many areas of AI not-really related to human intelligence, game theory, airplanes vs birds, …).
- Sociology (but also ecology and other science disciplines) like to use agents for their simulation models (Agent Based Model).

2. Inteligent agents

Better definitions (many of them), methodologies, models

How to stars a MAS?

- To know how to create at least one agent
- The key problem of agent design is … action selection:
  - What the agent should do in a given moment based on information from environment.
- Agent architecture is then a software architecture that enables the process of decision – action selection.
  - … 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. (Maes, 1991)

OK, but what is an agent?

(Franklin, Graesser, 1996 – Is it an agent or just a program?)

- MuBot (mobile agent Crystaliz, Inc)
  - *"The term agent is used to represent two orthogonal concepts. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning."*
- AIMA (Russell and Norvig)
  - *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."*
- Pattie Maes, MIT Media Lab
  - *"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."*

… or?

- KidSim, Apple
  - *"Let us define an agent as a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller."*
- Barbara Hayes-Roth, Stanford Knowledge system lab
  - *"Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions."*

... or more ...

- IBM, Intelligent Agent strategy white paper
  - *"Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."*
- SodaBot , MIT AI Lab
  - *"Software agents are programs that engage in dialogs [and] negotiate and coordinate transfer of information."*
- Brustoloni
  - *"Autonomous agents are systems capable of autonomous, purposeful action in the real world."*
- Software agents mailing list FAQ
  - *"This FAQ will not attempt to provide an authoritative definition ..."*

Let us summarize

- Franklin:
  - *An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*
- Wooldridge, Jennings:
  - *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 delegated objectives.*


Prostředí poskytuje:
Vjemy, Feedback
Agent: Vnímání,
Rozhodování,
Akce
Senzory,
Efektory
imgflip.com

Environment

- *Fully/Partially observable*
  - Agent can fully observe a complete environment state via its sensors.
  - Can not.
- *Static / Dynamic*
  - Environment is changing only as a result of agent actions.
  - Also in other ways.
- *Deterministic / Nondeterministic*
  - Each action has one and only one result.
  - Has not.
- *Discrete / Continuous*
  - An environment has a fixed number of states.
  - More than that.
- Unfortunately, most interesting environments, like real world or internet, are continuous, non-deterministic, dynamic and partially observable.

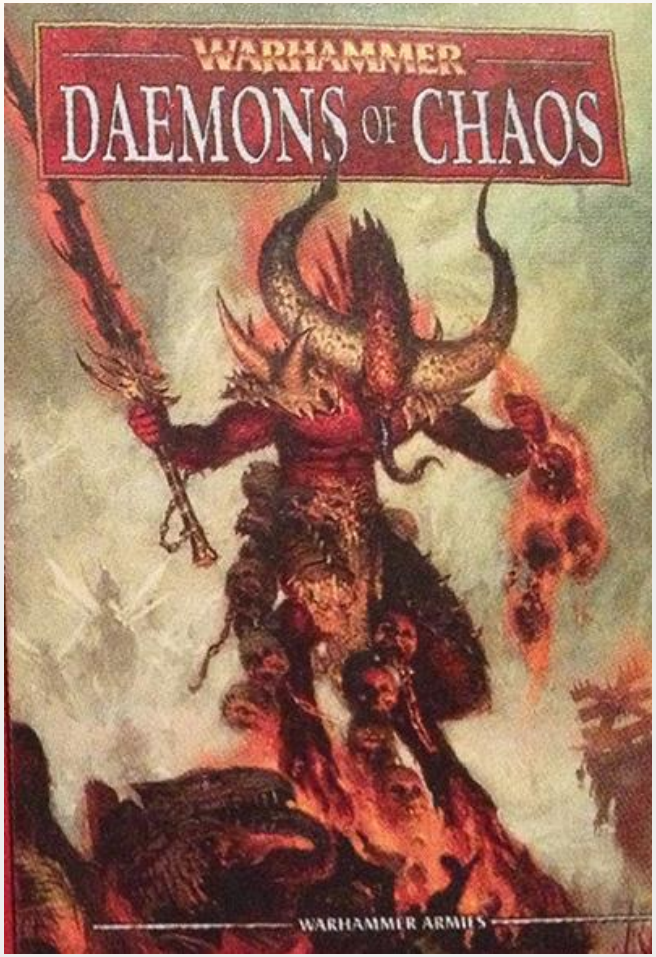
Notes to an agent

- Autonomy:
  - Human vs. Java method
  - Agent is somewhere in between – should be able to choose autonomously the way how to solve an assigned problem, i.e. choice of subgoals, not of the main goal (delegation).
- Deliberation:
  - Agent has a repertoire of available actions, not all can be executed always, not all makes sense always
  - The key problem for an agent is to select the best action in order to fulfill its goals.
- Agent architecture is then an architecture of a software system embedded in the environment, that serves to action selection (embeded decision-making system).

Example: Thermostat

- Not-so-intelligent agent
- Situated in environment (room)
- 2 perceptions: cold, OK
- 2 actions: off, on
- Decision unit:
  - cold => on
  - OK => off
- Non-deterministic environment (open doors, …)

Example: software demon



- Such as unix xbiff
- Situated in an environment of operating system
- Senses through executing commands and signals …
- Has software actions – run an email client, change icon on a screen …
- Decision algorithm is on the same level as thermostat

Intelligent agent

- Reactive
  - Senses the environment, is able to react in real (reasonable) time to its changes.
- Proactive
  - Has its own goals, is able to fulfill them actively.
- Social
  - Is able to communicate with other agents, or people.

- To achieve pure reactive or pure planning/goal-oriented behavior is relatively simple. What is difficult, is to achieve the balance between reactivity and proactivity.

Intentional system

- Oftentimes, when speaking about agents, we assign the so-called ***mental states*** to them (they believe something, want something, desire something, hope for something, …)
- Daniel Dennet defines ***intentional system*** as an entity whose behavior can by predicted by assigning properties as beliefs, desires, and rational wit.
- Hierarchy: I.S. of the first order, second order (beliefs and desires about beliefs and desires, … third order …
- It is interesting to note that we in fact use intentional systems (such as when arranging a meeting for given time and space, and then meeting a friend there)

Example: Electric switch as an intentional system

- Shoham: *The electric switch in a room is a (very) cooperating agent which by his own will can conduct electric current, but doing it so only if it believes that we actually want it. By switching the switch, we communicate our wish to it.*
- Consistent, elegant, concise, nicely describes the switch actions.
- And yet, many people consider this childish or absurd.
- In this case it is because we (probably) have another, still precise, but simpler way how to describe a switch.

Intentional stance

- Physical stance:
  - For description and functionality prediction only laws of physics are sufficient (I throw a stone, no need to talk about its desires to find out where it will land. I need only its weight, velocity, law of gravity)
- Design stance:
  - In more complex systems I may not know all exact physical laws, but it is enough to understand why they were created, what is their purpose. (teleological explanation)
  - (I know how an alarm clock works, do not have to know all related physical laws, still know how to set the alarm clock, and predict when it will ring)
- Intentional stance:
  - I use the intentional system view for prediction.

Intentional stance

- Sometimes the physical/design description is not available.
- Sometimes it is but it is not practical for description and behavior prediction
  - E.g. Even if I have an electronic scheme of a computer, it is difficult to derive why a window on a screen opens after clicking on some desktop icon.
- In many situations, the intentional stance is simpler than alternatives.
- From the computer science point of view, it is an abstraction in order to handle the complexity of the problem.
- For many programmers/computer scientists, the usage of intentional systems to program agents, is the (most) important feature of multi-agent systems.

And how about Daniel Dennet?

- The problem if there exists an intentionality of entities by itself, or if the intentionality is realized only in the process of interpretation.
- Dennett thinks that there is no intentionality by itself, without interpretation.

3. Abstract agent architectures

States, environment, agents

Environment

- We try to formalize an abstract view on an agent and its interactions with the environment.
- Environment can be in one of finite number of discrete states
- E = {e, e', … }
- Even if the environment were not really discrete, we discretize it
- This is a standard presumption for diverse modeling approaches, nothing unusual
- And, every continuous environment can be modelled by a discrete environment with arbitrary precision, nice.

Agent

- Agent has at its disposal, a finite number of actions
- A = {a, a', …}
- How does the agent interact with the environment:
  - Environment is in some initial state
  - Agent chooses an action
  - Environment can respond by transition in several possible states
  - Environment responds by changing its state into one particular state from the set of possible states.
  - We do not know which one in advance
  - Based on the environment state, agent selects further action
  - …

Run

- **Run** of an agent in an environment is a sequence of alternating environment states and actions
- $r = e_0, a_0, e_1, a_1, e_2, a_2, \ldots a_{n-1}, e_n$
- Let:
- R be a set of all possible finite sequences on E and A
- R_A is a subset of those sequences from R that end by an action from A
- R_E is a subset of those sequences from R that end by a state of the environment

A state transformer function

- The influence of an agent to the environment is modelled by a state transformer function:
- T: R_A->2^E
  - T maps run of an agent (ending by an action) to the set of environment states (those states that can be a result of the final action)
- Environments have history
  - The next environment state does not depend on the last agent action only, but also on previous actions and states.
- Environments are non-deterministic
  - We do not know in which state the environment will be after the action execution.

Environment again

- If for some r in $R_{A,}$ the T(r) is an empty set, we say that the system has **finished** a run.
- Thus, there are no more possible states after r
- From now on let us consider that all runs will allways end.
- **The Environment** is a triple:
- Env = (E,e_0,T)
  - E is a set of environment states
  - e_0 is the initial state
  - T is the state transformer function

Agent again

- Agent is a function mapping runs (ending by environment states) to actions:
- Ag: R_E -> A
  - Thus, the agent is deciding which action to select based on the complete history of the system
  - Agent is deterministic
- Let AG be the set of all agents
- A System is a pair containing an agent and an environment
- Each system has a corresponding set of possible runs
- R(Ag,Env) is a set of runs of an agent Ag in the environment Env
  - Consider only runs that end, i.e. T(r)=O

Agent runs in environment

- We say that a sequence $(e_0,a_0,e_1,a_1,e_2, \ldots)$ is a **run of agent Ag in environment Env=(E,e_0,T),** iff:
  1. e_0 is the initial state of Env,
  2. $a_0 = Ag(e_0)$,
  3. For each $u>0$:
  - $e_u \in T((e_0,a_0,\ldots,a_{u-1}))$ and
  - $a_u = Ag((e_0,a_0,\ldots,e_u))$
- Ag_1 and Ag_2 are **functionally equivalent** with respect to Env, if and only if the following holds:
  - $R(Ag_1,Env) = R(Ag_2,Env)$
- Ag_1 and Ag_2 are **simply functionally equivalent**, if they are functionally equivalent with respect to all Env.

Pure reactive agent

- Pure reactive (or tropist) agent:
- Ag: E -> A
- Reacts without taking history into account, the action selection mechanism is based on current environment state only.
- For every pure reactive agent, there exists a standard agent which is functionally equivalent.
- (Of course,) the opposite does not hold.
- Thermostat:

```
Ag(e) = off, if e=temp OK
Ag(e) = on, elsewhere
```

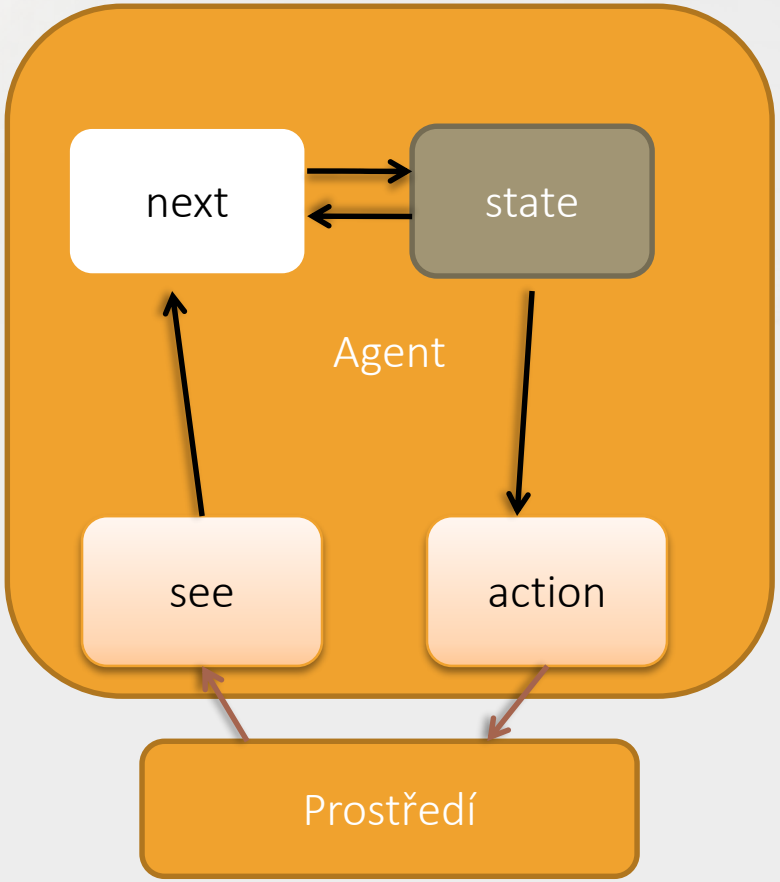
Agent with a state

  - Let us not define technicalities of the agent state representation, consider a state to be a piece of information, typically about environment state, and its history, saved in some internal data structure that helps the agent in action selection process.

- I – the set of all internal states of an agent
- Per – the set of all perceptions of an agent
- see – a function of agent perception see: E -> Per
- action: I -> A
- next: I x Per -> I
- Every agent with a state can be transformed into functionally equivalent standard agent.

Agent with a state at work

- Agent starts in internal initial state i_0
- Percepts the environment and generates a perception
  - see(e)
- Updates its inner state according to
  - next(i_0,see(e))
- Selects an action akci
  - action(next(i_0,see(e)))
- Executes the action, and starts another iteration



What should an agent do, and how to tell it

- We do not want fixed and hard-wired agents
- The idea is to tell the agent what to do, but not how to do it
- A popular way how to do it in AI/CS is to define the problem/goal indirectly by means of some kind of measure of success of an agent
- Utility – objective function evaluating environment states
- u: E -> $\mathscr{R}$
- Agent goal is to reach environment states with high utility value

Utility of a run

- Overall success of an agent can be based on:
  - The utility of the worst state reached by an agent
  - Average utility of states visited by an agent
    - Not clear what is better, usually this is task dependent
- To evaluate individual states (of the environment) can be short-sighted, but how to evaluate longer-term utility to judge the success of an agent
- Utility of a run r:
- u: R -> $\mathscr{R}$
- Better for agents running independently for a longer time

Example: Tileword

  - Agents are on a checkboard, moving to 4 directions
  - There are holes, obstacles and blocks on the checkboard
  - Agent goal is to push blocks into holes
  - The environment is dynamic – holes, obstacles and blocks are appearing and disappearing at random
- $u(r) = N_f/N_a$,
  - N_f – number of holes filled by blocks by an agent during a run r
  - N_a – number of all holes in a run r
- And this is repeated several times
- Agent has to react to environment changes
  - (the block I am pushing suddenly disappears)
- Agent should be able to make use of opportunities
  - (new block appears next to me)

Maximization of expected utility

- If the function u(r) has an upper bound, we can talk about maximization:
- Optimal agent should maximize its expected utility
- P(r|Ag,Env) – probability of a run r of agent Ag in nevironment Env
- $\sum P(r|Ag,Env) = 1$, for all $r \in R(Ag,Env)$
- Ag_{opt} optimal agent:
  - $Ag_{opt} = \arg\max \{Ag \in AG\} \sum u(r)\, P(r|Ag,Env)$
- Which is a nice definition, but it does not provide clues how to design such an optimal agent
- And sometimes it can be complicated to define the utility function as well

Predicate specification of a task

- Utility is a mapping to the set of {0,1}
  - Run is successful if u(r)=1
- F is predicate specification (we do not exactly specify how it looks for now):
  - F(r) is true, if and only if u(r)=1
- The environment of a task is then (Env,F):
  - Env is the environment
  - F: a mapping R ->{0,1}
- TE is a set of all environments of a task
- The environment of a task then specifies:
  - The properties of a system (via Env)
  - Criteria if the agent fulfills the task (via F)

The environment of a task

- RF(Ag,Env) is a set of runs of agent Ag in environment Env, that satisfy F
  - RF(Ag,Env) = {r | r ϵ R(Ag,Env) & F(r)}
- When does agent Ag solves the task (Env,F):
  - Pessimist: RF(Ag,Env) = R(Ag,Env)
    - I.e., all runs satisfy F
  - Optimist: ∃ r ϵ R(Ag,Env) tak, že F(r)
    - I.e., at least one run satisfies F
  - Realist:
    - Let us extend T such that it includes probability distribution over all possible results (and so over all runs)
    - The success of an agent can be then measured by a probability of satisfying F:
    - P(F | Ag,Env) = ∑ P(r | Ag,Env) for r ϵ RF(Ag,Env)

Types of tasks (from life)

- To find something(Achievment tasks)
  - The goal is to reach any state from a goal set G
  - G is a set of states from E takových such that F(r) is true if at least one state from G appears in the run r
  - Agent is successful if all his runs end in a state from G
  - Example: Pretty much any task from AI (searching a solution)
- To keep something (Maintenance tasks)
  - Agent must avoid some environment states
  - B is a set of states, such that F(r) is false if any of states from B appears in the run r
  - Example: games, B are losing states, the environment is the opponent
- Combination: for example achieve state from G but avoid states from B

4. Deductive reasoning agent

Induction, deduction, agent as a theorem prover

Classical AI approach

- Classical way how AI creates an „inteligent system“:
  - Symbolic representation of environment and behaviour
    - We will focus on representation by means of logical formulae
  - Syntactic manipulation with such a representation
    - This corresponds to logical deduction or theorem proofs
  - Thus, the theory about the agent behavior (cf. previous talk) is in fact a program – an executable specification providing concrete agent actions
- The transduction problem: How to represent the world
  - *"Grau, teurer Freund, ist alle Theorie, und Grün des Lebens goldner Baum."*
  - Computer vision, natural language processing, learning, …
- The representation and reasoning problem:
  - How to represent knowledge, theorem provers, planners …

By the way

- Deduction: derivation of conclusions that are true, based on the fact that preconditions are true
  - general -> specific
    - Sylogisms: „All humans are mortal, Sokrates is human, thus, Sokrates is mortal. “
- Induction: if the preconditions are true, then the conclusion is more likely true than not.
  - special case -> more general
    - Police says that S. is a killer (/seen by two witnesses/left fingerprints/confessed), then, S. is a killer.
- Sherlock Holmes is using induction, while saying it is deduction.
- Mathematical induction is in fact deduction.

Agent as a theorem prover

- L be a set of formulae in the first order logics,
- $D=2^L$ is a set of data-bases of formulae L,
- The inner state DB of an agent is then DB ϵ D.
- Deliberation is done by means of deduction/derivation rules P in the underlying mathematical logics
  - DB $\vdash_P$ f – a formula f can be derived from DB (only by using deduction) rules P
- see: S -> Per
- next: D x Per -> Per
- action: D -> A according to rules P

Action selection as proving

```
Function action(DB:D) returns action A
begin
   for each a ϵ A do
      If DB ⊢P Do(a) then return a
   for each a ϵ A do
      If DB ! ⊢P ¬Do(a) return a
return null
end
```

- Returns action that can be proven via Do(a)
- Or, tries to find a consistent action (i.e. not in a contradiction with DB)

Example: Vacuum world 3x3

- In(x,y) – agent is on (x,y)
- Dirt(x,y) – there is dirt
- Facing(d) – agent faces direction d
- Action deduction:
- Vacuum clean:
  - In(x,y)& Dirt(x,y) => Do(suck)
- And browse the world, e.g., 00-01-02-12-11-10-…

- In(0,0) & Facing(north) & not Dirt(0,0) => Do(forward)
- In(0,1) & Facing(north) & not Dirt(0,1) => Do(forward)
- In(0,2) & Facing(north) & not Dirt(0,2) => Do(turn)
- In(0,2) & Facing(east) => Do(forward)
- …

Pros and cons

- Elegant and with beautiful semantics
- Not really practical
- Takes a long time, if ends at all
  - Computable rationality: agent derives someting, based on the environment current state, but in the meantime, the environment changes and the action may not be optimal any more. That is bad for rapidly changing environments.
- Sometimes it is hard to find a good see() function
  - How to translate picture into formulae
  - How to represent temporal data
- Some elements of this approach have been used in other architectures, as we will see, e.g. in the next talk about Practical reasoning

5. Praktical reasoning agent

Beliefs-Desires-Intentions


OH, JEFF... I LOVE YOU, TOO... BUT...

Practical reasoning

- Inspired by human decision processes
- Theoretical reasoning (cf. Sokrates) results only in what we think about the world
- Practical reasoning leads to actions
- Two phases:
  - Deliberation
    - What we want to achieve
      - I want to graduate
  - Means-Ends reasoning
    - How to achieve the goal
      - Have to create a plan how to graduate
- And all this should not take too long

Intentions

- Intention is such a state of the world that the agent wants to achieve
- Intentions in agent lead to actions (in order to achieve the state of the world), then a reasoning follows which results in a plan
- Intentions persist:
  - Until the agent achieves them,
  - Until the agent starts believing they cannot be achieved
  - Until the reasons leading to the intention disappear
- Intentions constrain further deliberation
- Intentions influence what the agent will believe in the future

Desires

- Desire represents one of possible intentions
  - *My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires … before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions. (Bratman, 1990)*
- Desires are motivational states of an agent
- Do you recall Daniel Dennett?

Beliefs

- **Beliefs** are agent's knowledge, they represent its information state.
- We do not call them knowledge in MAS, in order to emphasize that:
  - They are subjective from the point of view of the agent,
  - They are not necessarily true,
  - They can change in the future.
- Beliefs can contain inference rules allowing forward chaining

B ... D ... I

- Consider some (any) explicit representation of the sets of beliefs, desires, and intentions, such as symbolic, but for now the technical details are not important.
- B
  - Variable for agent current beliefs,
  - Bel is then a set of all possible beliefs.
- D
  - Variable for desires,
  - Des is a set of all desires.
- I
  - Variable for intentions,
  - Int is a set of all intentions.

Deliberation

- Optins generating function:
  - options: $2^{Bel} \times 2^{Int} \rightarrow 2^{Des}$
- Filter function = selection from options:
  - filter: $2^{Bel} \times 2^{Des} \times 2^{Int} \rightarrow 2^{Int}$
- Belief refresh function:
  - brf: $2^{Bel} \times Per \rightarrow 2^{Bel}$

Means-Ends Reasoning, or Planning

- The process of deciding how to reach a goal (intention) based on available means (actions).
- Input:
  - Goal = intention
  - Current environment state = agent beliefs
  - Actions available for agent
- Output:
  - Plan = sequence of actions,
  - When the agent executes the actions, the goal is fulfilled

STRIPS

- Nilsson, Fikes, 1971
- Model of the world = a set of first order logic formulae
- The set of action schemes:
  - Preconditions
  - Effects:
    - Add – facts that will be (newly) true after the action
    - Delete – facts no longer true
- Planning algorithm:
  - Finds differences between the goal and current state of the world
  - Decreases the differences by application of suitable action
  - …
  - This is nice, but not very practical, the algorithms often iterates a lot over low-level details

The block world

- Predicates:
  - On(x,y) = x is on y
  - OnTable(x,y) = x is on a table
  - Clear(x) = nothing is on x
  - Holding(x) = robot holds x in his robotic arm
  - ArmEmpty = robot arm is empty
- Initial state:
  - {Clear(A), On(A,B), OnTable(B), OnTable(C), Clear(C)}
- Goal:
  - {OnTable(A), OnTable(B), OnTable(C)}

The block worls – actions

- Stack(x,y)
  - Pre {Clear(y), Holding(x)}
  - Del {Clear(y), Holding(x)}
  - Add {ArmEmpty, On(x,y)}
- UnStack(x,y)
  - Pre {On(x,y), Clear(x), ArmEmpty)}
  - Del {On(x,y), ArmEmpty)}
  - Add {Holding(x), Clear(y)}
- Pickup(x)
  - Pre {OnTable(x), Clear(x), ArmEmpty)}
  - Del {OnTable(x), ArmEmpty)}
  - Add {Holding(x)}
- PutDown(x)
  - Pre {Holding(x)}
  - Del {Holding(x)}
  - Add {ArmEmpty, OnTable(x)}

Definition of plans

- Set of actions Ac = $\{a_1, \ldots, a_n\}$
- Descriptor of an action a is $[P_a, D_a, A_a]$:
  - P_a set of FOL formulae – precondition of action a
  - D_a the delete set of effects after action a (expressed in FOL)
  - A_a the add set of effects after action a (in FOL),
  - For simplicity, these sets contain only grounded atomic formulae – no logical conjuctions, …, no variables
- The planning problem is then $[B_0, O, G]$:
  - B_0 – initial beliefs of an agent
  - $O = \{[P_a, D_a, A_a]: a \in Ac\}$ descriptors for all actions
  - G = set of FOL formulae representing the goal

Definition of planning

- Plan p is a squence $(a_1,\ldots,a_n)$, $a_i \in Ac$
  - Plan p for planning problem $[B_0,O,G]$ determins a sequence of belief databases $B_0, B_1, \ldots, B_n$:
    - $B_i = (B_{i-1} \setminus D_{ai}) \cup A_{ai}$; for i = 1, …, n
  - Plan p is admissible for $[B_0,O,G]$ ⇔
    - $D_{i-1} \models P_{ai}$; for each i = 1, …, n
  - Plan p is correct for $[B_0,O,G]$ ⇔
    - p is admissible for $[B_0,O,G]$ and
    - $D_{i-1} \models G$.
- Planning: for $[B_0,O,G]$ either find a correct plan, or say it does not exist.

Few more defintions

- Plan = the set of plans (nad Ac)
- *pre(p)* = precondition of plan p
- *body(p)* = body of plan p, a sequence of actions
- *empty(p)* = Boolean function, is the plan p empty
- *execute(p)* = procedure executing the plan (all actions sequentially)
- *hd(p)* = the first action in the plan p body
- *tail(p)* = actions of p from the second one to the last one
- *sound(p,I,B)* = plan p is correct for sets of intentions (goals) I and beliefs B

Agent planning function

- plan: $2^{Bel} \times 2^{Int} \times 2^{Ac} \rightarrow Plan$
- Agent does not have to construct plans on-line, because it can be time consuming
- Oftentimes, the *plan()* is implemented by means of library of plans
- Then, it is sufficient to iterate through the plan library once, and check if:
- The preconditions of a plan correspond to agent current beliefs
- The effects of a plan correspond to the goal

Implementation

- B:= B_0; I:= I_0
- while true do
  - v:= see(); B:= brf(B,v); D:= options(B,I); I:= filter(B,D,I);
  - p = plan(B,I,Ac);
  - while not (empty(p) or succeed(I,B or impossible (I,B)) do
    - a:= hd(p); execute(a); p:= tail(p);
    - v:= see(); B = brf(B,v);
    - if reconsider(I,B) then D:= options(B,I); I:= filter(B,D,I) endif
    - if not sound(p,I,B) then p = plan(B,I,Ac);
  - endwhile
- endwhile

Commitment of agents

- Mechanisms when to abandon the goal = commitment strategies:
- Blind commitment
  - Agent will continue to maintain an intention until it believes it has achieved the intention.
- Single-minded commitment
  - Agent will continue to maintain an intention until it believes either the intention has been achieved, or it is no longer possible to achieve it.
- Open minded
  - The intention persists until the agent believes it is still possible to achieve it.

Commitment to plans/intentions

- Agent is committed to one plan only, i.e.
  - It will end if:
    - Believes the goal has been achieved
    - Believes the goal is not possible to achieve
    - The plan is empty
- succeeded(I,B) = I holds under assumption of B
- impossible(I,B) = I cannot hold assuming B

Commitment to goals

- When should agent stop and reconsider the intention?
- Classical dilemma – deliberation takes time, and the environment can change during the process
  - Agent, who does not reconsider intentions, can happen to be be committed to goals that are no longer possible to achieve.
  - Agent, who reconsiders to often, can be too busy to actually solve the current intention and might not achieve anything.
- meta-control (Wooldridge, Parsons, 1999)
  - function reconsider(), which is computationally simpler than the reconsidering itself
  - Observation reconsider() is working well if every time it proposes reconsidering, the agent actually changes the intention after deliberation process

„To boldly go where no man has gone before.“

- Classical extreme solutions to the dilemma:
  - Bold agent – reconsider intentions only after it ends execution of the current plan (i.e. never stops the plan to reconsider)
  - Cautious agent – reconsiders after every action of the plan
- Level of boldness – how many actions to execute between reconsiderations
- Dynamism of the environment – rate of world change, how many times the environment can change during one agent cycle
- Agent efficiency = achieved intentions / all intentions
  - When the world is changing slowly, bold agents are efficient
  - When the world is changing rapidly, cautious agents outperform bold agents

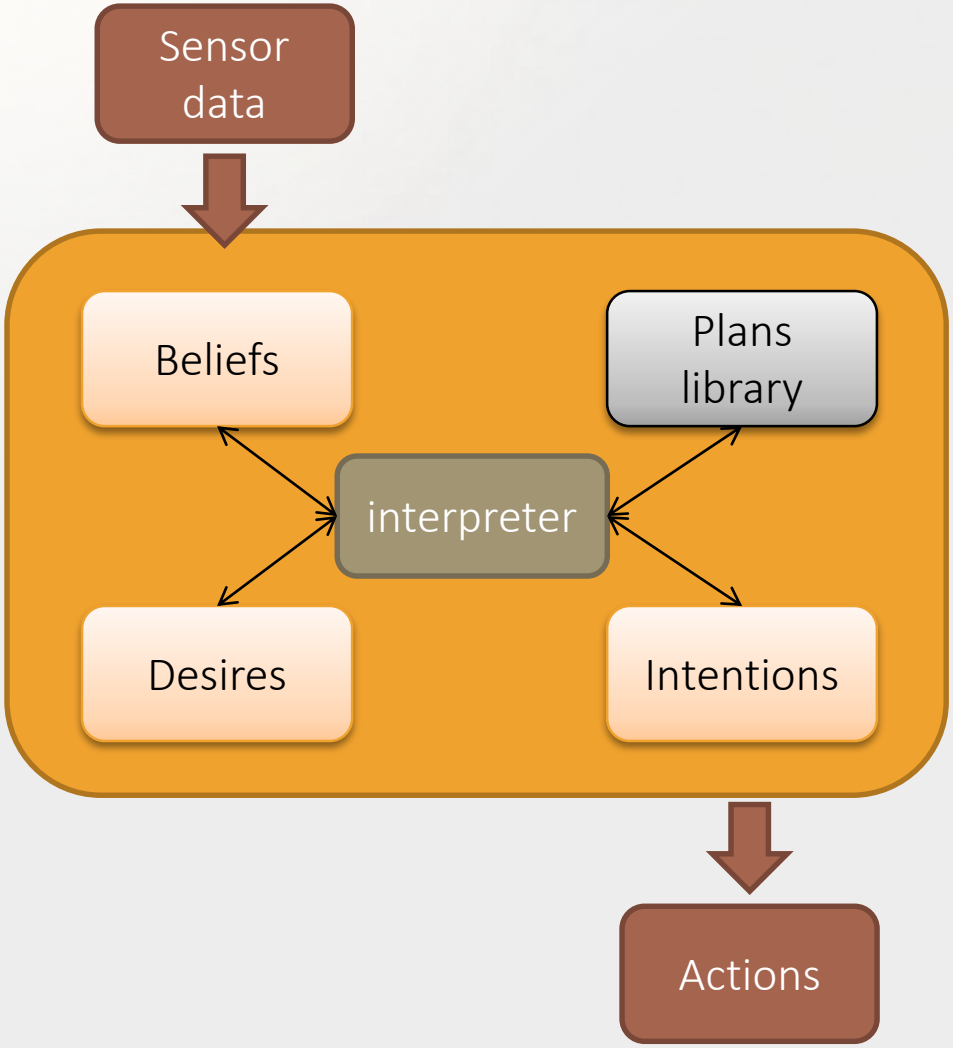
6. Procedural Reasoning System

BDI architecture in practice

PRS

- (Georgeff et al, 80s), Stanford
- The first implementation of BDI architecture
- Maybe the most successful agent architecture, reimplemented in many cases and environments
  - AgentSpeak/Jason
  - Jam
  - Jack
  - JADEX
- And also used in practice
  - OASIS – Sydney air traffic control system
  - SPOC (single point of contact) bussinness processes organization
  - SWARMM – air force flight simmulator

PRS agent


Sensor data
Beliefs
Plans library
interpreter
Desires
Intentions
Actions

Plans in PRS

- Agent has a library of ready plans, representing its procedural knowledge
- No full planning, only choosing a plan from the library
- Plan
  - Goal – condition holding after the execution
  - Context – condition necessary to run
  - Body – actions to execute
- Plan body:
  - Not only linear sequence of actions
  - Plan can contain goals
    - Achive f,
    - achieve f or g,
    - keep achieving f until g

Planning in PRS

- Initialization:
  - Beliefs B_0 (prolog FOL atoms)
  - top-level goal
- Stack of intentions
  - The stack contains current goals in the state of partial completion
  - Interpreter takes an intention on top and searches the plan library for matching goal
  - Out of the matching plans, only some have their context consistent with current beliefs
  - These represent current options/desires

Deliberation in PRS

- Deliberation – selecting intention from desires
  - The original PRS had meta-plans
    - Plans about plans, they were modifying agent intentions
    - But it was too complicated
  - Utility
    - Every plan is evaluated by number representing expected utility
    - The plan with biggest utility is selected
- The selected plan is executed, which can result in adding more intentions on the stack …
- If a plan fails, agent chooses another intention from options, and continues

Example: JAM

```
GOALS:
  ACHIEVE blocks_stacked;

FACTS:
  FACT ON "Block5" "Block4";      FACT ON "Block4" "Block3";
  FACT ON "Block1" "Block2";      FACT ON "Block2" "Table";
  FACT ON "Block3" "Table";       FACT CLEAR "Block1";
  FACT CLEAR "Block5";            FACT CLEAR "Table";

Plan: {
  NAME: "Top-level plan"
  GOAL: ACHIEVE blocks_stacked;
  CONTEXT:
  BODY:     ACHIEVE ON "Block3" "Table";
            ACHIEVE ON "Block2" "Block3";
            ACHIEVE ON "Block1" "Block2";
}

Plan: {
  NAME: "Stack blocks that are already clear"
  GOAL: ACHIEVE ON $OBJ1 $OBJ2;
  CONTEXT:
  BODY:     ACHIEVE CLEAR $OBJ1;
            ACHIEVE CLEAR $OBJ2;
            PERFORM move $OBJ1 $OBJ2;
  UTILITY: 10;
  FAILURE: EXECUTE print "\n\nStack blocks failed!\n\n";
}

Plan: {
  NAME: "Clear a block"
  GOAL: ACHIEVE CLEAR $OBJ;
  CONTEXT: FACT ON $OBJ2 $OBJ;
  BODY:     ACHIEVE ON $OBJ2 "Table";
  EFFECTS: RETRACT ON $OBJ1 $OBJ;
  FAILURE: EXECUTE print "\n\nClearing block failed!\n\n";
}
```

Example: Jason

```
/* Initial beliefs */
// initially, I believe that there are some beers in the fridge
available(beer,fridge).
// my owner should not consume more than 10 beers a day
limit(beer,10).

/* Rules */
too_much(B) :-
.date(YY,MM,DD) &
.count(consumed(YY,MM,DD,_,_,_,B),QtdB) &
limit(B,Limit) &
QtdB > Limit.

/* Plans */
@h1
+!has(owner,beer)
: available(beer,fridge) & not too_much(beer)
<- !at(robot,fridge);
open(fridge);
get(beer);
close(fridge);
!at(robot,owner);
hand_in(beer);
// remember that another beer will be consumed
.date(YY,MM,DD); .time(HH,NN,SS);
+consumed(YY,MM,DD,HH,NN,SS,beer).

@h2
+!has(owner,beer)
: not available(beer,fridge)
<- .send(supermarket, achieve, order(beer,5));
!at(robot,fridge). // go to fridge and wait there.

@h3
+!has(owner,beer)
: too_much(beer) & limit(beer,L)
<- .concat("The Department of Health does not allow me ",
"to give you more than ", L,
" beers a day! I am very sorry about that!",M);
.send(owner,tell,msg(M)).

@m1
+!at(robot,P) : at(robot,P) <- true.3.4. EXAMPLE: A COMPLETE
        AGENT PROGRAM 63

@m2
+!at(robot,P) : not at(robot,P)
<- move_towards(P);
!at(robot,P).
// when the supermarket finishes the order, try the 'has'
// goal again

@a1
+delivered(beer,Qtd,OrderId)[source(supermarket)] : true
<- +available(beer,fridge);
!has(owner,beer).
// when the fridge is openned, the beer stock is perceived
// and thus the available belief is updated
@a2
+stock(beer,0)
: available(beer,fridge)
<- -available(beer,fridge).
@a3
+stock(beer,N)
: N > 0 & not available(beer,fridge)
<- +available(beer,fridge).
```

Example: ARTS

GOALS:
ACHIEVE PrepareLecture agents101 :PRIORITY 9 :DEADLINE 50;
ACHIEVE HaveLunch :PRIORITY 7 :DEADLINE 40;
ACHIEVE BorrowBook R&N :PRIORITY 2 :DEADLINE 30;
CONCLUDE LectureNotes agents101 myNotes;
PLAN: {NAME: ”Plan 1”; DOCUMENTATION: ”Prepare for lecture”;
CUE: ACHIEVE PrepareLecture $x, y;
PRECONDITION: TEST LectureNotes $x, y;
BODY:
EXECUTE revise-lecture $y :TIMEOUT 35;}

PLAN: {NAME: ”Plan 2”; DOCUMENTATION: ”Pickup a book from the library”;
CUE: ACHIEVE BorrowBook $x;
BODY:
EXECUTE goto library :TIMEOUT 10;
ACHIEVE Pickup $x;}

PLAN: {NAME: ”Plan 3”; DOCUMENTATION: ”Pick up something”;
CUE: ACHIEVE Pickup $x;
BODY:
EXECUTE pickup $x :TIMEOUT 2;}

PLAN: {NAME: ”Plan 4”;DOCUMENTATION: ”Have lunch”;
CUE: ACHIEVE HaveLunch;
BODY:
EXECUTE eat-sandwich :TIMEOUT 20;}

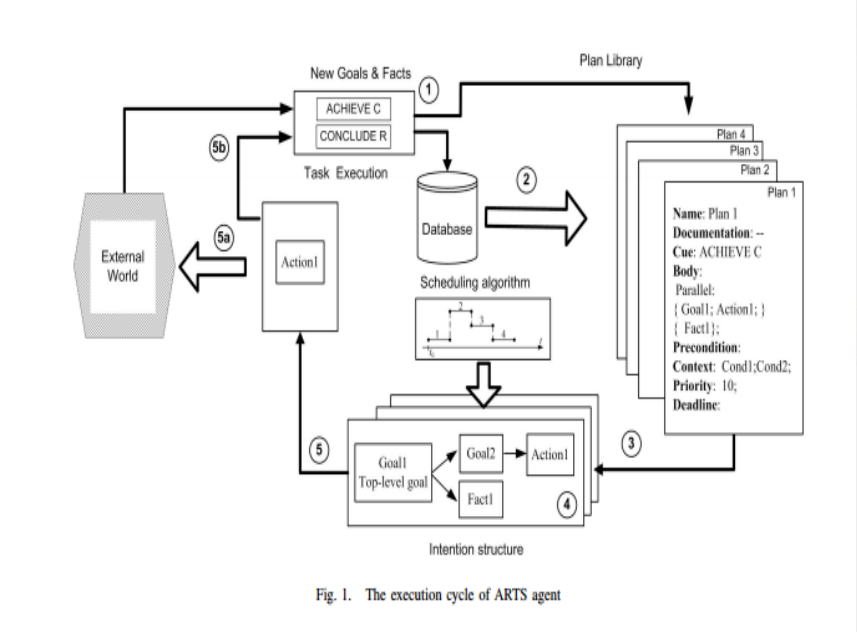


Fig. 1. The execution cycle of ARTS agent

ACHIEVE C	achieve condition C
CONCLUDE F	add fact F to the database
TEST C	test for the condition C
RETRACT F	retract fact F from database
WAIT C	wait until condition C is true

7. Reactive and hybrid architectures

Brooks subsumption architecture, horizontal and vertical layered architectures, Stanley

Reactive approach

- Problems with symbolic representation and reasoning
- 80s-90s – small changes of the symbolic approach are not enough
- Alternative paradigms in AI
- Rejection of symbolic representation with deduction based on syntactic manipulation
- Interakce - Intelligent behavior depends on the environment where the agent is situated
- Embodiment - Intelligent behavior is not just the logic, it is a product of embodied agent, agent with a body
- Emergence – intelligent behavior emerges by interaction of (many) simple behaviors

Reactive agents

- Behavioral – emphasize evolution and combination of individual behaviors
- Situated – agent is in environment, it is embodied
- Reactive
  - Agent mainly (only) reacts on the environment
  - Agent does not do deliberation
- Subsymbolic representation
  - Connectionism
  - Finite automata
  - Simple reactive If-THEN rules

Subsumption agenta architecture

- Probably the most successful from reactive approaches
- Rodney Brooks, 1991
- Inteligent behavior can be generated without symbolic representation
- Inteligent behavior can be generated without explicit abstract reasoning
- Inteligence is an emergent property of certain complex systems

Architecture

- Agents intelligence is realized via simple goal-oriented behaviors
  - Each behavior is an action selection mechanism
  - Each behavior receives perceptions and transforms them into actions
  - Each behavior is responsible for some goal
  - Each behavior is a simple rule-like structure
  - Each behavior competes with others for control over agent
  - Each behavior works in parallel to others
  - Behaviors are in the subsumption hierarchy defining their priorities

Action selection

- Subsumption mechanism reacts on the inputs:
  - Selecting rules corresponding to current situation
  - For each behavior that is applicable to current sitation (fires) it checks if there is a behavior with higher priority in the hierarchy
  - If not, the behavior is selected
  - If nothing is selected, no action is executed
- It is simple (but not so simple to program in the case of dozens of behaviors, the priorities can be tricky)
- It is fast (hw implementation, constant complexity)

Example: Steels' Mars explorer

- The goal is to collect precious rock samples on distant planet by a swarm of robotic explorers

- Means:
- The base transmits a navigation signal
- Communication is not necessary, it is sufficient to detect a gradient of the signal
- Every robot has radioactive crumbles for indirect communication with other robots

First iteration – random walk and return of one robot

- R1: if detect obstacle then change direction
- R2: if carying sample and at the base then drop sample
- R3: if carying sample and not at the base then travel up gradient
- R4: if detect a sample then pick sample up
- R5: if true the move randomly

- Priority!

Second iteration: better exploration

- What to do if I carry a sample:
- R6: if carrying a sample and at the base drop sample
- R7: if carrying a sample and not the base then drop 2 crumbs and travel up gradient
- R8: if sense crumb then pickup 1 crumb and travel up gradient

- Priority: R1 < R6 < R7 < R4 < R8 < R5

Agent network architecture

- Maes, 1991
- Every agent is a set of competence modules (resembling Brooks behaviors)
- Each module has
  - Pre-conditions
  - Post-conditions
  - Activation threshold (defining how relevant the module is with respect to current situation, works as a priority during action selection process)
- Modules are connected in a network based on their conditions
- Matching pre and post conditions represent oriented edges
- And there are further connections representing time precedence or conflicts
- Modules in the network are activated, and the most activated one is selected to determine the action

Limitations of reactive architectures

- Reactive agents do not create any kind of model of the world, they have to derive everything from the environment (thus, they sometimes have to change the environment, e.g. by radioactive crumbles)
- Reactive agents have only short-term view on the world – they act based on current state and local information only, it is difficult to consider global conditions and long-term goals
- Emergence of behaviors is not an ideal engineering approach to programming
- Potentially many layers of reactive behavior is difficults to design

Hybrid architectures

- It seems neither completely reactive nor completely deliberative architectures are ideal
- Hybrid architectures try to combine more components into an agent wishing for best of both worlds:
  - Deliberative/planning component(s) working on symbolic level, creates representations, plans
  - Reactive component(s) for immediate actions without complex computations
- These components are usually in a hierarchy where the reactive ones are given precedence over the deliberative ones

Hybrid architectures

- **Horizontal**
- Layers are connected to sensors and effectors in parallel
- Relatively simple, but the layers can influence each other
- mediator function – resolves conflict between layers (potential bottleneck)

- **Vertical**
- Layers are connected to sensors and effectors in serial manner
- One-pass
  - Natural ordering and hierarchy of behaviors
- Two-pass
  - Bottom-up go perceptions, top-down go actions
  - The flow resembles control in real-world companies

Concrete Example: Touring Machines

- Horizontal 3-layer architecture
- Modelling layer
  - Models the agent and environment, resolves conflicts, sets goals, sends the goals to the planning layer
- Planning layer
  - Proactive behavior, selects from pre-programmed plans similarly to the PRS approach
- Reactive layer
  - Classical reactive rules, fast, immediate reaction (obstacle avoidance, …)

Concrete Example: InteRRaP

- Vertical two-pass 3-layer architecture
- Cooperative planning layer
  - Social interactions
- Lokální plánovací vrstva
  - Day-to-day planning
- Behaviorální vrstva
  - Reactive
- Each layer has its own knowledge base on different abstraction level

Real wordl example: Stanley

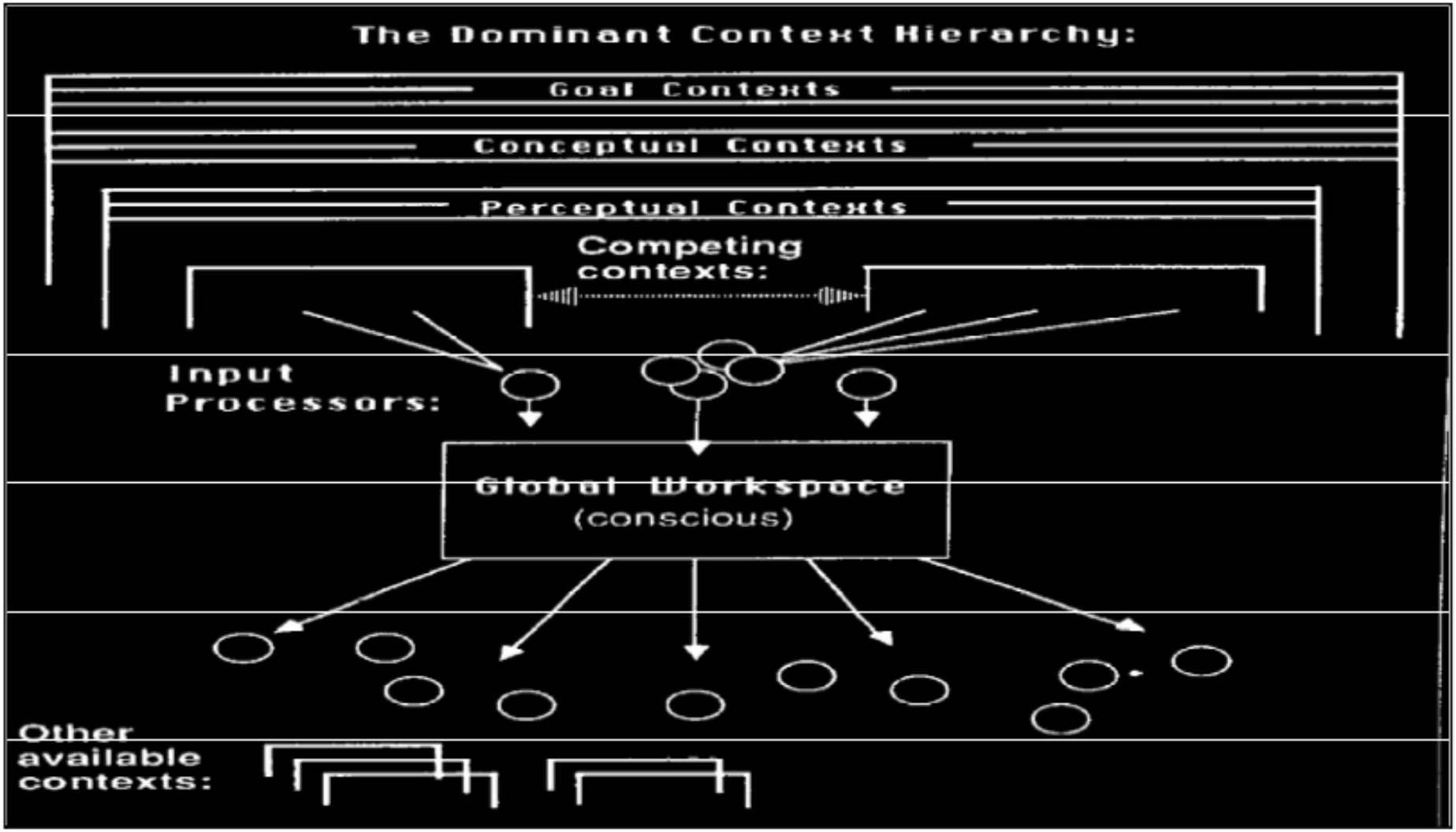
- Volkswagen Touareg R5
- Autonomous car, the father of todays Google cars and similar AUV
- Won DARPA Grand Challenge 2005 – 132 miles in Mojave desert
  - Sensor layer
  - Abstract perception layer
  - Planning and control layer (road plan and control of the car)
  - Vrstva rozhraní vozidla
  - Vrstva uživatelského rozhraní (panel, start)
  - Vrstva globálních služeb (filesystém, komunikace, hodiny)

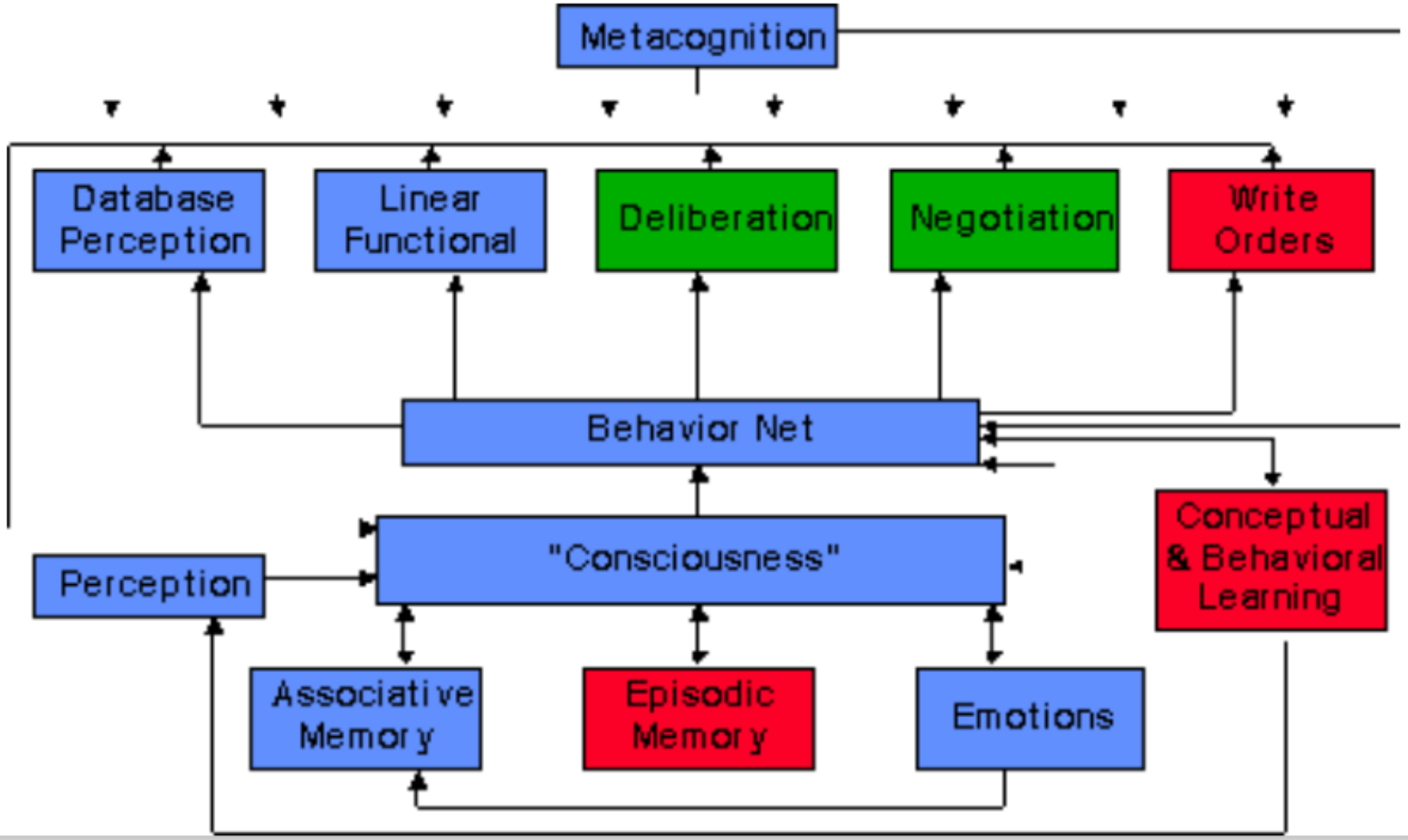
Concrete example: IDA

- S. Franklin - “Intelligent distribution agent”
  - Distribution relates to the application domain, which originally was a navy personnel distribution on their assignments, although the architecture is distributed itself as well
- Probably the most complicated agent architecture one can see
  - Which implies both pros and cons
  - Inspired by (one of many) theory of consciousness
- Combines many approaches from MAS and other fields of AI to achieve many complex tasks:
  - Action selection, memory, deliberation, emotions, …
  - Some choices were made ad hoc
  - It is difficult to tune the collection of heterogenous models to cooperate in an optimal way

IDA - principles

- Inspired by *global worskpace theory* (Baars 88-97)
  - Mind is a multi-agent system
  - Consists of many simple (ofte subconscious) processes realizing specialized operations
  - They communicate seldom, and via a shared memory of blackboard type, organized as associative array
  - The processes dynamically create higher-order coalitions
  - The coalition that is most applicable to current inputs will get to the “consciousness” (and is executed)
  - There is a hierarchy of “contexts” representing the world on different levels – context of goals, perceptions, senses, concepts, cultural context.


The Dominant Context Hierarchy:
Goal Contexts
Conceptual Contexts
Perceptual Contexts
Competing contexts:
Input Processors:
Global Workspace
(conscious)
Other available contexts:


Metacognition
Database Perception
Linear Functional
Deliberation
Negotiation
Write Orders
Behavior Net
Conceptual & Behavioral Learning
"Consciousness"
Perception
Associative Memory
Episodic Memory
Emotions

8. Ontologies

How to understand each other, from dictionaries to is-a hierachies, from Aristotle to programming

From agent to MAS

- We know how to design an agent in several different ways
- How should agent communicate and cooperate in MAS:
  - Knowledge representation
  - Common dictionaries
  - Standard messages
  - Predictable behavior during communication – communication protocols
  - Semantics of communication
  - Technical problems with communication
    - Distributivity
    - Mobility
    - Asynchronnicity
    - Unreliable communication channels

Ontologies

- Ontology (from Greek to όν being + λόγος, word) is part of philosophy dealing with nature of being, becoming, existence, or reality, and related basic philosophical notions. Aristotle is calling it the first philosophy, it is part of metaphysics, and dealing with the most general questions.
- Ontologies in computer science means explicit and formalized description of part of reality. It is usually a formal and declarative description containing glossary (definition of concepts) and thesaurus (definitions of relations among concepts). Ontologies are kind of dictionaries for storing and exchanging knowledge about some domain in a standard way.

History of ontologies in AI

- "Toward Principles for the Design of Ontologies Used for Knowledge Sharing”, T. Gruber, 1995
  - An ontology is a description (like a formal specification of a program) of the concepts and relationships that can formally exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set of concept definitions, but more general. And it is a different sense of the word than its use in philosophy.
- "The Semantic Web”, T. Berners-Lee, Scientific American Magazine, 2001.
  - “… remains largely unrealized”, 2006

Ontologies, real life example

- Anna: Have you heard 7777?
- Boris: No, what is it?
- Anna: It's a new CD from the band called SRPR, a kind of alt.rock with some electronic music, and the lyrics are just great.
- Boris: Oh, I see.
- 7777 is an album
- 7777 music falls into alt.rock, as well as electronica
- 7777 has songs, the artists (SRPR), music and lyrics authors …
- SRPR is a band, has members, they play instruments, …
- …

Ontologies in general

- Classes – things with something in common
- Instances (objects) – concrete individuals from the classes
- Properties
- Relations between classes
- Sub-class
  - Transitive relation
- Further properties and relations (background knowledge?)
- Structural part – this is usually called ontology
- Facts about concrete items
- Together they form a knowledge base

Ontology of ontologies

- From weaker informal systems, to strong formal ones
  - Dictionary (controlled vocabulary) – selected terms
  - Glossary – definition of meanings by means of selected terms, often in natural language
  - Thesaurus – definition of synonyms
  - Informal hierarchies – subclasses hiearchy more or less defined, Amazon, wiki
  - Formal is-a hierarchy – subsumption of classes in a formal way
  - Classes with properties
  - Value restrictions – every human has 1 mother
  - Arbitrary logical constrains – leads to complex reasoning algorithms

Different ontology of ontologies

- Application
  - Most common in practice, hardly re-usable
- Domain
  - Favorite part of research and application of semantic web, …
- Upper
  - Should be ultimate ontology describing everything
    - Like Thing, Living thing, Proof, Vegan, …
  - Serve as a base for domain ontologies
  - BFO, GFO, UFO
  - Wordnet, IDEAS – not suitable for formal reasoning and machine usage
  - Metodological objections – Wittgenstein, Tractatus logico-philosphicus

Ontological languages

- Formal declarative languages for knowledge representation
- Contain facts and reasoning rules
- Most often based on first-order logic, or description logic
- Frames
  - Historical predecessor of ontologies
  - Proposed by Minsky, favorite in classical AI, expert systems, …
  - Vizualization of human reasoning and language processing

Frame Terminology	OO Terminology
Frame	ObjectClass
Slot	Object property or attribute
TriggerAccessor, Mutator methods	Method

XML

- XML
  - Was not developed for ontology representation, but sometimes it is used so
  - Coming from www
  - Main advantage - new tag definition
  - XML tags then naturally represent dictionaries

```
<catalog>
<product typ="CD">
    <title>7777</title>
    <artist>SRPR</artist>
    <price currency="CZK">250</price>
</product>
<product typ="CD">
    <title>Dlask</title>
    <artist>SRPR</artist>
    <price currency="CZK">200</price>
</product>
</catalog>
```

RDF (resource definition framework)

- Standard knowledge representation tool (not only) for web
- Simple
  - Not very expressive
  - Simple (fast) reasoning algorithms
- Representing triples subject-predicate-object

MarriedTo(Karel,Jája), FatherOf(Karel,Péťa), FatherOf(Karel, Jíťa)

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ns="http://www.example.org/#"> <ns:Person
    rdf:about="http://www.example.org/#john"> <ns:hasMother
    rdf:resource="http://www.example.org/#susan" /> <ns:hasFather>
    <rdf:Description rdf:about="http://www.example.org/#richard">
    <ns:hasBrother rdf:resource="http://www.example.org/#luke" />
    </rdf:Description> </ns:hasFather> </ns:Person>
</rdf:RDF>
```

OWL - Web Ontology Language

- Also coming from (semantic) web
- Ver 1 vs ver 2
- It is a collection of several formalisms to describe ontologies
- Syntax can be in XML (or RDF, or functional notation …)
- An attempt to have formal, yet practically useful approach
- Description logic – a decidable fragment of FOL
  - In the beginning, IS-A was quite simple. Today, however, there are almost as many meanings for this inheritance link as there are knowledge-representation systems. (Ronald J. Brachman, What ISA is and isn't)
- Open world assumption - OWA
  - [The closed] world assumption implies that everything we don't know is false, while the open world assumption states that everything we don't know is undefined. (Stefano Mazzocchi, Closed World vs. Open World: the First Semantic Web Battle)

Description logics

- Concepts (classes), roles (properties, predicates), and individuals (objects)
- Axiom – logical expression about roles and concepts
  - This is different to frames or object-oriented programming (both frames and OO fully describe classes)
- It is in fact a family of logical systems, depending on admissible rules/axioms
- For example:
  - ACL – negation a class conjunction, limited existential quantifier, contstrains
  - Extensions: class disjunction, hierarchy of roles, transitivity of roles, …
- Terminological T-Box, Axiom A-box

OWL 1

- OWL-Lite (SHIF)
  - The simplest, closer to RDF
  - Many axiom constrains in order to maintain readability and fast machine processing
- OWL-DL (SHOIN)
  - Corresponds to DL
  - Alllows to express things such as - Two classes are disjunct
  - Complete, Decidable
- OWL-Full
  - The strongest expressive power
  - Many problems are undecidable for this subset, though

OWL 2 (SROIQ)

- OWL-EL
  - Polynomial reasoning time complexity
- OWL-QL
  - Specialized to queries in knowledge bases
- OWL-RL
  - Special form of axioms - rules

OWL2 Functional Syntax

```
Ontology(<http://example.org/tea.owl> Declaration( Class(
        :Tea ) ) )
```

OWL2 XML Syntax

```
<Ontology ontologyIRI="http://example.org/tea.owl" ...>
        <Prefix name="owl"
        IRI="http://www.w3.org/2002/07/owl#"/>
        <Declaration> <Class IRI="Tea"/> </Declaration>
        </Ontology>
```

Manchester Syntax

```
Ontology: <http://example.org/tea.owl> Class: Tea
```

RDF/XML syntax

```
<rdf:RDF ...> <owl:Ontology rdf:about=""/> <owl:Class
        rdf:about="#Tea"/> </rdf:RDF>
```

RDF/Turtle

```
<http://example.org/tea.owl> rdf:type owl:Ontology . :Tea
        rdf:type owl:Class .
```

KIF - Knowledge interface format

- Representation of knowledge in FOL
  - (salary 015-46-3946 widgets 72000)
  - (> (* (width chip1) (length chip1)) (* (width chip2) (length chip2)))
  - (interested joe `(salary ,?x ,?y ,?z))
  - (progn (fresh-line t) (print "Hello!") (fresh-line t))

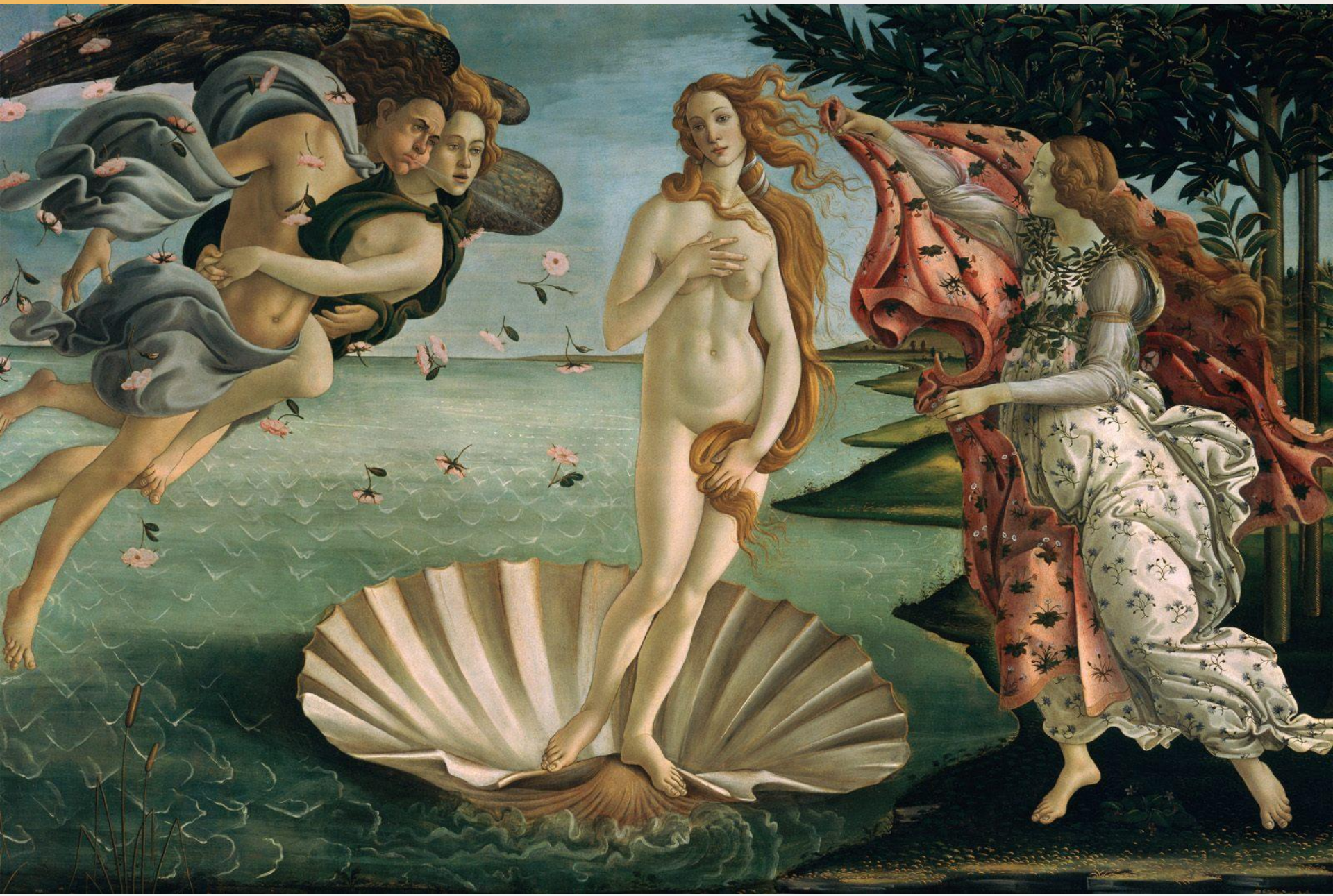
  - (forall (?F ?T)
    (=>
    (and
    (instance ?F Farmer)
    (instance ?T Tractor))
    (likes ?F ?T)))
  - (exists (?F ?T)
    (and
    (instance ?F Farmer)
    (instance ?T Tractor)
    (likes ?F ?T)))

DAML+OIL

- DAML – Darpa agent markup language
- OIL – Ontology Interchange language
- Predecessor of OWL
- Abandoned in 2006

9. Agent communication

Speech acts, KQML, ACL, KIF, protocols



Agent communication

- In OOP, communication means calling object methods with parameters
- Agents cannot directly make other agent to do something, or to change its inner variables
- Agents have to communicate – perform a communication act:
  - In order to exchange information,
  - In order to influence other agents to do something
- Other agents have their own agenda, goals, and it is up to them how they handle all the information, requests, queries from peers
- It is a beautiful day today

Speech acts

- Austin, 1962
  - some parts of language usage have character of actions, because they change the state of the world similarly to physical actions – they are speech acts
    - I pronounce you man and wife
    - I declare a war on Russia
  - This is tha pragmatic language theory – how the speech is used to achieve goals
- Verbs as request, inform, promise
- Locutinary acts
  - what was said,
  - Utterances - small part of language usage, such as sentence
    - Make me a tea
- Ilocutionary act
  - What was meant
  - Locution + perfomative meaing (query, request, …)
    - Ha asked me for tea
- Perloctionary act
  - What really happened
  - The effect of the speech act
    - She made me to make her a tea

Searle and his work on speech acts

- Example: SPEAKER request HEARER action
- standard I/O conditions
  - H can hear, it is not happening in the movie …
- Pre-conditions
  - What has to be true, so that S can choose this speech act:
    - H must be capable to perform the action, S believes that H is capable to perform the action, it is not clear that H would perform the action without asking
- Honesty
  - S really wants the action to be performed

Searle speech acts categories

- Older:
  - Request, Advice, Statement, Promise, …
- Newer
  - Assertives (Representatives) – informing the hearer
  - Directives – requesting an action from the hearer
  - Comissives – promise by a speaker
  - Expressives – speaker expresses a mental state, emotions, “thank you!”
  - Declarations – change the state of things, war, marriage
- Speech act should have
  - Performative verb – request, query, inform, …
  - Propositional content – the window is closed

Planning theory of speech acts

- Cohen, Perrault, 1979
  - *„… modelling [speech acts] in a planning system as operators defined … in terms of speakers and hearers beliefs and goals. Thus speech acts are treated in the same way as physical actions.“*
- STRIPS
  - preconditions,
  - postconditions
- Modal operators
  - beliefs,
  - abilities,
  - wants
- Thus, the semantics of speech acts is defined by means of precondition-delete-add approach from STRIPS

Example: Request and Inform

- Request (S,H,A)
  - Preconditions
    - Cando:
      - (S believe (H cando A))&(S believe(H believe (H cando A)))
    - Want:
      - (S believe (S want requestinstance))
  - Effect:
      - (H believe (S believe (S want A)))
- Inform(S,H,F)
  - Preconditions
    - Cando:
      - (S believe F)
    - Want:
      - (S believe (S want informinstance))
  - Effect:
    - (H believe (S believe F))

Agent communication languages

- 1990s: DARPA – Knowledge sharing effort (KSE)
  - KQML – knowledge query and manipulation language
    - Outer communication language (envelope of a letter)
    - Contains Ilocutionary part of the message
  - KIF – knowlewdge interchange format
    - Inner language,
    - propositional contents of the message
    - Knowledge representation

- FIPA ACL (foundation of physical agents, agent communication language)
  - Simplification of KQML, semantics, better system in performatives,
  - Practical implementation in JADE

KQML

- Perfomative
- Content
- Receiver
- Language
- Ontology

```
(ask-one
  :content (PRICE IBM ?price)
  :receiver stock-server
  :language LPROLOG
  :ontology NYSE-TICKS
)
```

Parameters and performatives of KQML

- Content
- Force
- Reply-with
- In-reply-to
- Sender
- Receiver

- Achieve
- Advertise
- Ask-about, ask-one, ask-all, ask-if
- Break, sorry, error
- Broadcast
- Forward
- Recruit-all, -one
- Reply
- Subscribe

ACL

```
(inform
   :sender agent1
   :receiver agent2
   :content (price good2 150)
   :language sl
   ontology: hpl-auction
)
```

ACL performatives

- Request, request-when
- Inform, inform-if, inform-ref
- Subscribe
- Cfp
- Propose
- Proxy
- Refuse
- Reject-proposal
- Confirm, disconfirm
- Agree, cancel

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

10. Cooperation of agents

Contract net, BBS


Ceci n'est pas une pipe.

Working together …

- Agents have different goals, are autonomous
- Agents work in time, not hard-wired, decisions made at run-time, be capable of dynamic coordination

- Sharing tasks
- Sharing information

- Coherence – how well the system performs as a whole
- Coordination – how well agents minimize overhead activities related to synchronization, …

CDPS

- Cooperative distributed problem solving – CDPS
  - Lesser et al, 80s
  - Cooperation of individual agents when solving a problem exceeding their individual capabilities (information, sources)
  - Agents implicitely share a common goal, there are no conflicts – benevolence
  - Overall system performance is the measure of success
  - Agent helps the whole system even if it can be disadvantageous for it
  - Benevolence enormously simplifies the system design

CDPS vs. PPS vs. MAS

- CDPS differs from PPS (paralel problem solving, Bond, Gasser, 80s)
  - Focus on parallel solving,
  - Homogenous and simple processors
- Generally, agents in MAS are more complicated:
  - MAS is a society of agents with their own goals
  - They do NOT share a common goal
  - They should cooperate despite this
    - Why and how
    - How to identify and resolve conflicts
    - How to negotiate and bargain

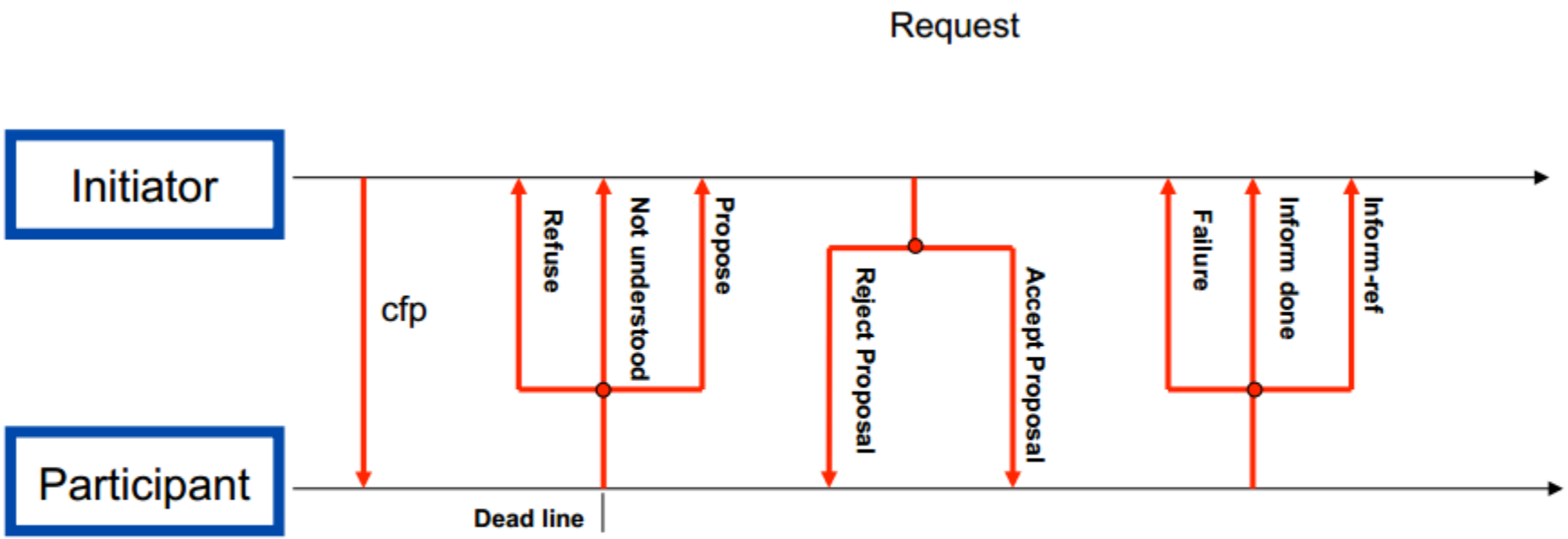
Task sharing and result sharing

- CDPS approach:
  - Problem decomposition
    - Hierarchical, recursive
    - How to decompose, who does the decomposition
    - ACTORS – for each sub-problem a new agent, till the instruction level
  - Sub-problem solutuion
    - Agents typically share some information during this
    - Agents might need to synchronize their actions
  - Solution synthesis
    - Hierarchical
- Task sharing – agreement of agents
- Result sharing – proactive or reactive

CNET

- Contract Net protokol, Smith and Davis, 1977
- Metaphore for task sharing via the contract mechanism
  - Recognition – agent recognizes it has a problem it cannot solve on its own, need to involve other agents
  - Announcement – agent broadcasts the announcement of the task including specification – description of the task (maybe executable), constraints (deadline, price, …)
  - Bidding – receiving agents decide if they want to participate, submit a tender
  - Awarding, Expediting – the agent in need selects a winner among bids and awards a contract
- Simple, can lead to hierarchical cascades of sub-contracting, was extensively studied, most implemented
- Other types of auctions possible (cf. the Auctions talk)

FIPA ACL CNET protocol



BBS - Blackboard systems

- BBS – the first original scheme for cooperative problem solving
- Results shared via the common data structure – the blackboard BB
  - Multiple agents (experts) sit around the BB, they can read and write there
  - The tasks are dynamically appearing on the BB
  - When an expert sees it can solve some task, will write the partial solution on BB
  - Until the final solution appears on the BB
- Requires mutual exclusion over BB – bottleneck
- Typically contain several abstraction levels, the BB can be structured into hierarchy
  - "Blackboard Architectures," AI Game Programming Wisdom, Volume 1, pp. 333 – 344)

BBS cont.

- Arbiter
  - Selects experts who can come to BB
  - Reactive, or considering plans maximizing expected utility
  - Responsible for higher-level problem solving (motivation)
- Experts
  - Agents to solve the problem by cooperation
  - React on the goals on BB
  - Execute actions when selected
- BB
  - Shared memory
  - The formalism for information representation is important
  - Typically for this paradigm, goals (and actions) are hierarchically ordered

Example – BBWar game

- BB
  - hash table – maps required capabilities to tasks
  - Open missions – tasks are publicized on the BB
- Experts
  - Solvers of various tasks in a hierarchy
  - List of capabilities and efficiency
- Example:
  - Commander agent seeks for ATTACK-CITY tasks, transforms them into multiple ATTACK-LOCATION tasks
  - Soldiers of various kinds seek for appropriate ATTACK-LOCATION missions

BBS – pros and cons

- Simple mechanism for agent coordination, cooperation, task and results sharing
- Experts do not need to know about other experts and still they can cooperate with them
- Messages on BB can (and typically are) rewritten – delegate tasks, create subtasks, change experts …
- Sometimes the BBS architecture is used for general communication of agents (every message is via BBS)
- Agents typically have to share the same architecture (to access the BB), and it can get crowded around BB (maybe distributed hash-tables can help)

Results sharing

- straightforward
- Besides trivial reasons, results sharing can help in solving these aspects:
  - Confidence
    - Independent solutions of identical problems can be compared
  - Completeness
    - Agents share their local views to create more global idea about the problem
  - Precision
    - Sharing can improve overall precision of the solution
  - Timely manner
    - An Obvious advantage of distributed approach is the time reduction

Example: FELINE

- Long time ago (before KQML or ACL), Wooldridge, Jennings, 1990
- Distributed expert system
- Sharing of knowledge, distribution of sub-tasks
- Each agent is a rule-based system
  - Skills – I can prove/contradict the following …
  - Interests – I am interested if the following is true or false …
- Communication
  - Sender, receiver, content (hypothesis + speech act)
  - Request, response, inform

Interaction of agents

- Agents can help each other, or obstruct
  - Robots can move a brick only by pushing from one side together
  - Robots crowd the entrance and cannot open the door
- Agents affect the environment
- Agents can create societies, subordinates, enemies
- It is important to know the types of interactions among agents in the particular MAS
- Otherwise, it is not possible to design efficient control mechanisms
- The simplest case – interaction of two rational (selfish) agents in an environment resembling a game

Self-interested agents cooperate

- Why should an agent be honest about its capabilities
- Why should agent finish an assigned task
- If the system is homogenous (such as completely designed by us), benevolence is good strategy
- But most often, the system contains agents with various interests
  - Conflict between the common goal and the goals of individual agents
  - Consider, e.g. the air traffic control
- Sometimes, the system is complicated, it is not explicitly clear what the common interest is
- Then it is better to consider selfish agents

What does a selfish agent want?

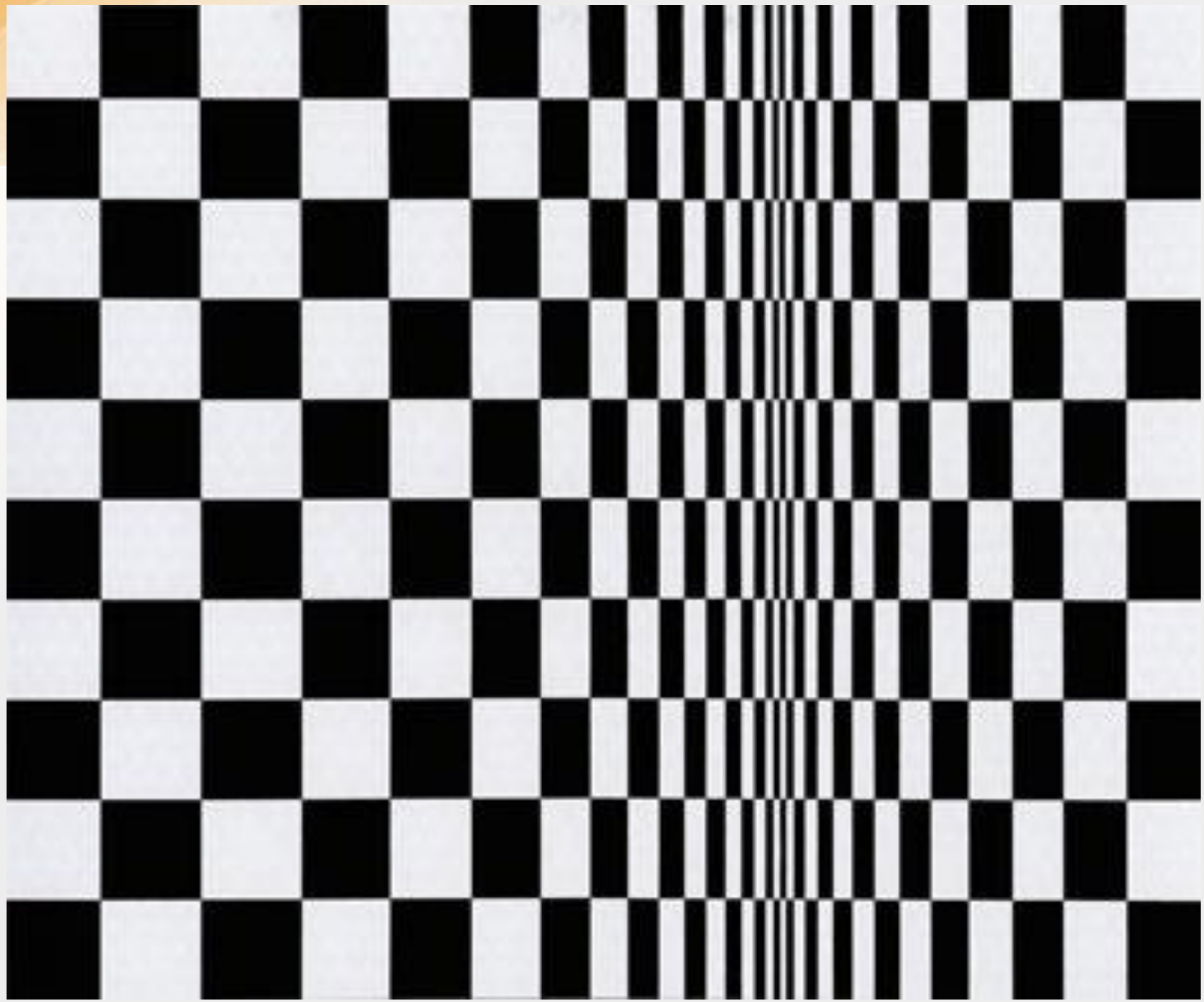
- To maximize its **expected utility**
- There is game theory, and many AI techniques to achieve this
- But other agents in the system want the same
- Each agent typically knows utilities fo its own actions
- Strategic thinking:
  - **Maximize your utility**
  - Considering everybody else (also) **act rationally**
  - This does NOT maximize the common utility
  - But it’s a robust strategy

Self-interested agent

- Benevolence not supported
- The set of possible outcomes $O=\{o_1, o_2, \ldots\}$,
  - Common for all (both) agents
- And preferences on O – utility function u: O->R
  - Different for each agent
  - Utility function sorts outcomes
- Remarks:
  - Money is not a good utility for humans
  - Non-linear utility function on money – different utility for rich/poor people
  - Extremes are symmetric

11. Agents interaction

Nashovo equilibrium, Pareto front, Prisoner's dilemma



Decision process as a 2-player game

- Both agents i and j influence the result
- – environment state change
  - e: A_i x A_j -> O
- Agent has a strategy s_i (s_j)
- Strategy s_i is dominant for agent i, if it provides better or same result than any other strategy of agent i, against all strategies of agent j
- Worst case scenario – opponent is rational, chooses the best alternative

Nash equilibrium

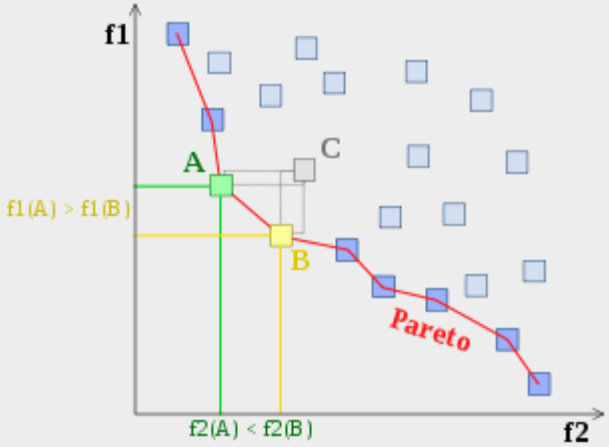
- Strategies s_1 and s_2 are in Nash equilibrium, if:
  - If agent i plays s_1, for agent j the best is to play s_2
  - If agent j plays s_2, agent i is best playing s_1
- I.e. s_1 and s_2 are mutually the best answer
- To find equilibria for n agents and m strategies takes m^n
- The definition is Nash equilibrium of pure strategies
  - But not every game has a Nash equilibrium in pure strategies
  - And some games have more of them

Nash theorem

- **Mixed strategy** – random selection between pure strategies
- **Nash theorem:** Every game with finite number of strategies has a Nash equilibrium in mixed strategies
- How difficult it is to find such an equilibrium – total search problem, since 2006 we know the problem is PPAD complete
  - PPAD-completness mean (oversimplified!) only a little bit less intractable than NP-completness.

Pareto optimality

- The strategy is Pareto-optimal/efficient, when no other strategy exists which would improve agent result without worsening the other agent result
- Non-Pareto-optimal solution can be improved without making other agent outcome worse



Social welfare

- Why don's actually agents maximize a common utility – social welfare
- = sum of utilities of all agents in the system
- But this is good in the cooperation scenarios only
- Typically,
  - when agents are from one team
  - Have one owner
  - Solve one task
  - The more homogenous the system, the better

Prisoner's dilemma?

- DD is Nash equilibrium
- DD is the only NON Pareto-optimal solution
- CC is the solution maximizing social wellfare
  - Tragedy of the commons
  - What does it mean to be rational?
  - Are people rational?
  - Shadow of the future – iterated – Axelrod, TFT strategy

i/j	D	C
D	2 / 2	0 / 5
C	5 / 0	3 / 3

Decision as voting

- Common utility – social welfare – aggregation of individual utilities
- Common candidate in elections – social choice – choosing the best candidate / utility
- **Example**:
  - A: o2>o1>o3,
  - B: o3>o2>o1,
  - C: o2>o3>o1
  - Social welfare: o2>o3>01
  - Social choice: o2

Voting shemes: Plurality

- Combine individual preferences to derive a social outcome
  - Each voter submits preferences
  - Each candidate gets one point for every preference ranking them first
  - Winner is the one with largest no. of points
- With two candidates, it is simple majority election
- With more candidates, it can happen that the winner is not a preferred candidate for majority of voters
  - Example o1 – 40%, o2 – 30%, o3 – 30%, winner is o1, but 60% did not want him
- Condorcet paradox – there are situations in which no matter which outcome we choose, a majority of voters will be unhappy with the result
- Tactical voting (do not follow my preferences, but vote against some candidate)

Example: Condorcet paradox

- V1: A > B > C
- V2: B > C > A
- V3: C > A > B

- No Condorcet winner

- Social welfare is cyclic, although individual preferences are linear

Other voting schemes

- Sequntial majority elections
  - Variant of plurality where players play pairwise „tournament rounds“, the winner moves further
  - Either linear or tree tournaments
  - The order of tournaments influences the election
- Borda count
  - Each voter submits its complete preferences, they are aggregated by counting the orders of all candidates
  - Used in real life sometimes: Slovenia, Nauru, Island
- Slater system
  - Optimize the preference aggregation process
  - Select a ranking of candidates to minimize the number of pairs of candidates such that the ranking disagrees with the pairwise majority vote on these two candidates
  - NP-complete

Properties of voting procedures

- Pareto property
  - If in everybody's preferences X > Y, then it should hold that X >sw Y
  - Holds for majority and Borda
  - Does not hold for sequential majority
- Condorcet winner
  - Condorcet winner is the candidate that beats opponents in pairwise comparisons, quite a strong thing.
  - Condorcet winner condition: Condorcet winner will be the overall voting
  - Sounds reasonable but holds only for sequential majority

Properties of voting procedures

- Independence of irrelevant alternatives (IIA)
  - Whether X >sw Y (i.e. X is ranked above Y in the social outcome) should depend only on relative orderings of X and Y in voters preferences.
  - Thus, when all preferences remain the ordering of X and Y, and maybe change something else, like preferences of other candidates X, Z or Z W, the relation X >sw Y should remain the same.
  - Does not hold for majority, sequential majority, neither Borda

Properties of voting procedures

- Unrestricted domain, or universality
  - a property of social welfare functions in which all preferences of all voters (but no other considerations) are allowed.
  - With unrestricted domain, the social welfare function accounts for all preferences among all voters to yield a unique and complete ranking of societal choices.
  - Thus, the voting mechanism must account for all individual preferences, it must do so in a manner that results in a complete ranking of preferences for society, and it must deterministically provide the same ranking each time voters' preferences are presented the same way.
- Dictatorship
  - The social outcome is determined by one of the voters – the dictator, whose preferences are taken as the social outcome
  - Non-dictatorship: No voter in the society is a dictator in the sense that, there does not exist a single voter *i* in the society such that for every set of orderings in the domain and every pair of distinct social states *x* and *y*, if voter *i* strictly prefers *x* over *y*, *x* is socially selected over *y*.

Arrow's theorem

- *For 3 and more candidates, no ranked voting electoral system can convert the ranked preferences of individuals into a community-wide (complete and transitive) ranking while also meeting a specified set of criteria:*
  - *unrestricted domain,*
  - *non-dictatorship,*
  - *Pareto efficiency, and*
  - *independence of irrelevant alternatives.*

Arrow's theorem simpler version

- *For elections with more than 2 candidates, the only voting procedure satisfying the Pareto condition and IIA is a* dictatorship*, in which the social outcome is in fact simply selected by one of the voters.*
- This is a *negative* result: there are fundamental limits to democratic decision making.
- But the interpretation that only working system is the dictatorship is totally wrong.

12. Auctions

English, Dutch, sealed-bid, Vickrey

Auction

- Mechanism, how to allocate (sparse) resources to agents
- Life:
  - eBay, Sothesby, …
  - Mining permits, mobile phone radio frequencies, …
  - Some games
- Computer science
  - Processor time, …
- Efficiency of auction:
  - Allocate the resources to agents that want them the most

Auction

- An auction is a market institution in which messages from traders include some price information—this information may be an offer to buy at a given price, in the case of a bid, or an offer to sell at a given price, in the case of an ask—and which gives priority to higher bids and lower asks.
- Seller
  - maximize the price
- Buyer
  - minimize the price
  - Has its own utility function
- Auction protocols
  - Winner – first price, second price, …
  - Open cry vs. sealed-bid
  - One or more rounds

First-price sealed bid

- One round, closed offers
- Highest bid wins, the offered price is paid
- Market price not estimated
- Other participants preferences not estimated
- Used for selling properties, treasury bonds, …
- Dominant strategy is to go epsilon bellow your utility

Vickrey

- Sealed bid, one round, second-price
- Highest bid is the winner, pays the second highest bid price
- Dominant strategy for the buyer is to offer its true value – why?
- Google AdWords, stamps, …

English

- Open cry, increasing price, first-price
- The most common one – antiques, artwork, internet, 95% of auctions are English
- Most common case where the buyer overshoots the price
- Dominant strategy – increase epsilons until the utility

Dutch

- Open cry, decreasing price, first-price
- Often used for perishable items – flowers, fish
- Sellers like it
- Not possible to estimate the market price
- Not possible to estimate other buyers preferences
- Dominant strategy – wait till utility minus epsilon is reached

Example

- Buyers utilities:
  - A – 80 Kč
  - B – 60 Kč
  - C – 30 Kč

- Different auctions:

- English – 61
- Dutch – 80 or 79ish
- FP Sealed bid – 80
- Vickrey – 60

- Ideal scenario with zero additional information and no cheating, …

Further auctions

- Combinatorial:
  - More items, subsets of them
  - Favourite for theory
- Paying all the bids:
  - Popular as tool for lobbing and bribes research, sport events (pay to run a marathon)
- Quiet:
  - Version of English on paper, cfp?
- Amsterdam:
  - Start English, when two buyers remain, switch to Dutch with double the price
- Tsukiji Tokyo fish market
  - Offers at once, conflicts by scissors-stone-paper