



ARTIFICIAL INTELLIGENT SYSTEMS

(BMA-EL-IZB-LJ-RE 1. YEAR 2024/2025)

MULTI-AGENT SYSTEMS

Simon Dobrišek


Copyrights all reserved © 2024 – University of Ljubljana, Faculty of Electrical Engineering

LECTURE TOPICS

- Introduction and definitions
- Agents as Intentional Systems
- Abstract architectures for agents
- Utility of agents
- Communication between agents
- Agent programming models
- Examples of multi-agent system platforms



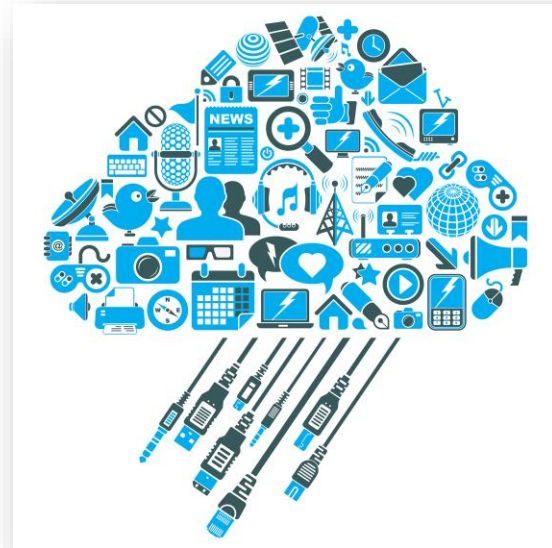
INTRODUCTION AND DEFINITIONS

- Machine code
 - Assembly language
 - Platform independent programming languages
 - Subroutines
 - Procedures and Functions
 - Abstract data types
 - Objects
 - ...
 - Subjects / Agents
- 



OTHER IMPORTANT TRENDS IN COMPUTING

- Grid computing
- Ubiquitous computing
- Semantic web
- Cloud computing
- Generative pre-trained language models
- ...



MULTI-AGENT SYSTEMS AND CLOUD COMPUTING

- Multi-agent systems as well as cloud computing support the development of open distributed systems.
- Cloud computing is focused on developing **interoperable infrastructures** and tools for secure and reliable **sharing** of computing resources.
- Multi-agent systems are focused on developing concepts, methodologies and algorithms for **autonomous problem solvers** that can act flexibly in uncertain and dynamic environments in order to achieve their objectives.

AUTONOMOUS AGENTS

- An agent is a computer system that is capable of independent (**autonomous**) action on behalf of its user or owner.
- An agent is able to **autonomously decide** on what needs to be done to achieve those objectives.
- They do not need to be constantly told what steps must be done to achieve the objective.
- An agent is a system situated in, and part of, a technical or natural environment, which senses state of that environment, and acts on it in pursuit of its own agenda.

MULTI-AGENT SYSTEMS

- A multi-agent system consists of a number of agents, which interact with one-another.
- In the most general case, agents will be acting on behalf of users with different goals and motivations.
- To successfully interact, they will require the ability to **cooperate**, **coordinate**, and **negotiate** with each other, much as people do.
- Multi-agent systems can be used to solve complex problems that are difficult or impossible for an individual agent or a monolithic system to solve.

APPLICATION AREAS

- Autonomous ground, air and space probes
- Autonomous Internet Explorers
- Network communication systems
- Computer games and graphics
- Artificial life
- Defence systems
- Transport and logistics systems
- Smart surveillance system
- ...

THE MICRO AND MACRO PROBLEMS

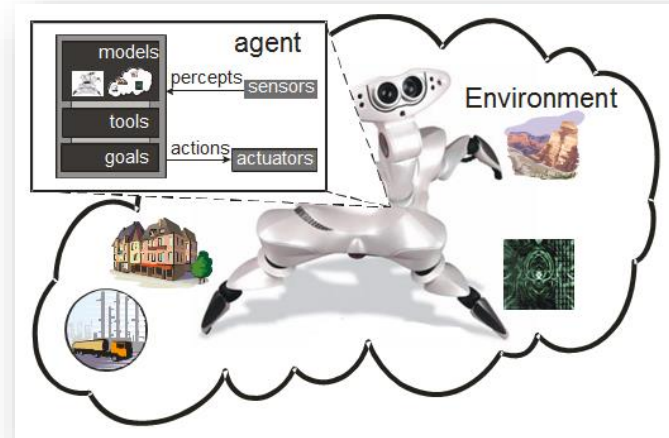
- Individual agent design:

How do we build agents that are capable of independent, autonomous action in order to successfully carry out the tasks that we delegate to them?

- Agent society design:

How do we build agents that are capable of interacting (cooperating, coordinating, negotiating) with other agents in order to successfully carry out the tasks that we delegate to them, particularly when the other agents cannot be assumed to share the same interests/goals?

WHAT IS AN AGENT?



- An agent is a computer system capable of autonomous action in some environment, in order to achieve its delegated goals.
- An agent is in a close-coupled, continual interaction with its environment

sense – decide – act – sense – decide – act ...

EXAMPLES OF TRIVIAL AGENTS

- Thermostat:
 - The aim is to maintain ambient temperature.
 - Actions are switching the heating on and off.
- Email reader:
 - The aim is to monitor the email and communication.
 - Actions are fetching emails from email servers and displaying them in a graphical user interface.
- They are trivial because the **decision making** they do is trivial.

INTELLIGENT AGENTS

- An intelligent agent is exhibiting the following three main types of behaviour:
 - **Reactivity** - A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful).
 - **Proactiveness** - Going beyond only reacting to events and changes in the environment, for example, recognising opportunities and goal directed behaviour.
 - **Social Ability** - The ability to interact with other agents (and possibly humans) via cooperation, coordination, and negotiation, as well as the ability to communicate

AGENT COOPERATION

- Cooperation is working together as a **team** to achieve a shared goal.
- Often prompted either by the fact that no single agent can achieve the goal alone, or that cooperation will obtain a better result (e.g., getting result faster).
- Coordination is managing the **interdependencies** between activities.
- For example, if there is a **non-sharable resource** that two agents want to use, then they **need to coordinate**.

AGENTS AND OBJECTS

○ Objects:

- encapsulate some state;
- communicate via message passing;
- have methods, corresponding to operations that may be performed on their states.

○ Agents:

- embody stronger notion of autonomy
(they decide for themselves whether or not to perform an action on request from another agent),
- are capable of complex and flexible behaviour
(they are reactive, pro-active, and social),
- are not passive service providers.

PROPERTIES OF AGENT ENVIRONMENTS

- **Accessibility** - An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.
- **Determinism** - 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.
- **Episodism** - In an episodic environment, the performance of an agent is dependent on a number of discrete episodes.
- **Dynamism** – A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control
- **Discrete vs. continuous** - An environment is discrete if there are a fixed, finite number of actions and percepts in it.

AGENTS AS INTENTIONAL SYSTEMS

- When explaining human activity, we use statements like the following:

Mary took her umbrella because she believed it was raining and she wanted to stay dry.

- These statements make use of a folk psychology, by which human behavior is predicted and explained by attributing attitudes such as **believing**, **wanting**, **hoping**, **fearing**, ...

AGENTS AS INTENTIONAL SYSTEMS

- The term intentional system was coined by the philosopher **Daniel Dennett** to describe entities

*“whose behaviour can be predicted by the method of attributing **belief**, **desires** and rational intelligence”.*

- A **first-order** intentional system has beliefs and desires about their environment and other agents but no beliefs and desires about their beliefs and desires.
- A **second-order** intentional system is more sophisticated; it has beliefs and desires about its beliefs and desires.

WHAT CAN BE DESCRIBED WITH THE INTENTIONAL STANCE?

- A trivial example of a light switch.

A light switch can be seen as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it **believes that we want** it transmitted and not otherwise.

Flicking the switch is simply our way of communicating our desires to them.



WHAT CAN BE DESCRIBED WITH THE INTENTIONAL STANCE?

- In simple systems, which are well understood, such descriptions do not provide anything useful.
- But with very **complex systems**, a mechanistical explanation of its behaviour may not be practicable.
- As computer systems are getting more and more complex, we need **more powerful abstractions** and metaphors to explain their operation.
- The intentional stance is such an abstraction.

POST-DECLARATIVE PROGRAMMING MODEL

- In **procedural programming**, in a computer program it is stated exactly what a system should do.
- In **declarative programming**, it is stated only what need to be achieved and some general information about the relationships between objects is given. A built-in control mechanism then figures out what to do by itself.
- In **agent programming**, a high-level description of the delegated goal is given, and the control mechanism should figure out what to do, where it acts in accordance with some built-in model of rational agency.

ABSTRACT ARCHITECTURES FOR AGENTS

- Let us assume that the environment may be in any of a finite set E of discrete, instantaneous states.

$$E = \{e, e', \dots\}$$

- Agents then have a set Ac of possible actions available to them, which transform the state of the environment.

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

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

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

ENVIRONMENT BEHAVIOUR

- A **state transformer** function τ represents behaviour of the environment and maps runs to (next) states.

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

- Note that environments can be **history dependent** or **non-deterministic**.
- If $\tau(r) = \emptyset$ there are no possible successor states to a run r , so the run has ended (“*Game over*”).
- An environment Env is then a triple

$$Env = \langle E, e_o, \tau \rangle$$

where E is set of environment states, e_o is initial state; and τ is state transformer function.

AGENT BEHAVIOUR

- Agent is a function Ag which maps runs to (next) actions.

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

- An agent thus makes a decision about what action to perform based on the history of the **system** that it has witnessed to date.
- 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

$$\mathcal{R}(Ag, Env)$$

AGENT SYSTEM

- Formally, a sequence

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \alpha_2, e_3, \dots)$$

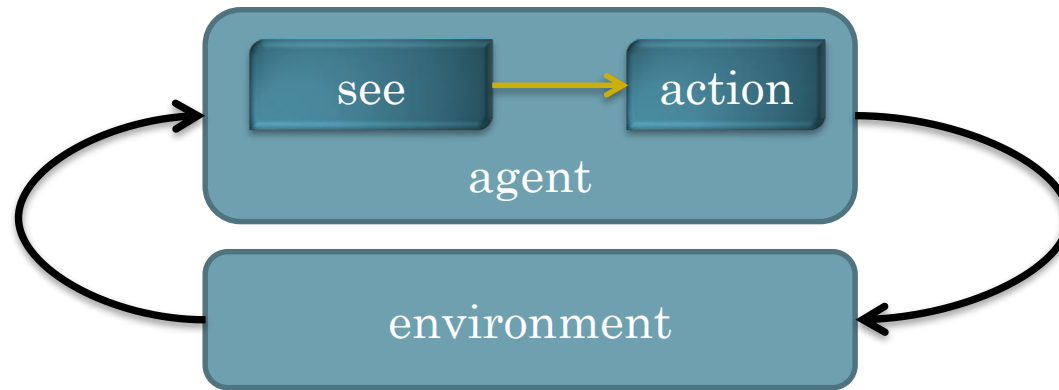
represents a run of an agent Ag in environment $Env = \langle E, e_0, \tau \rangle$ if :

- e_0 is the initial state of Env
- $\alpha_0 = Ag(e_0)$; and
- for $u > 0$

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

GENERALIZATION OF AGENT SYSTEMS

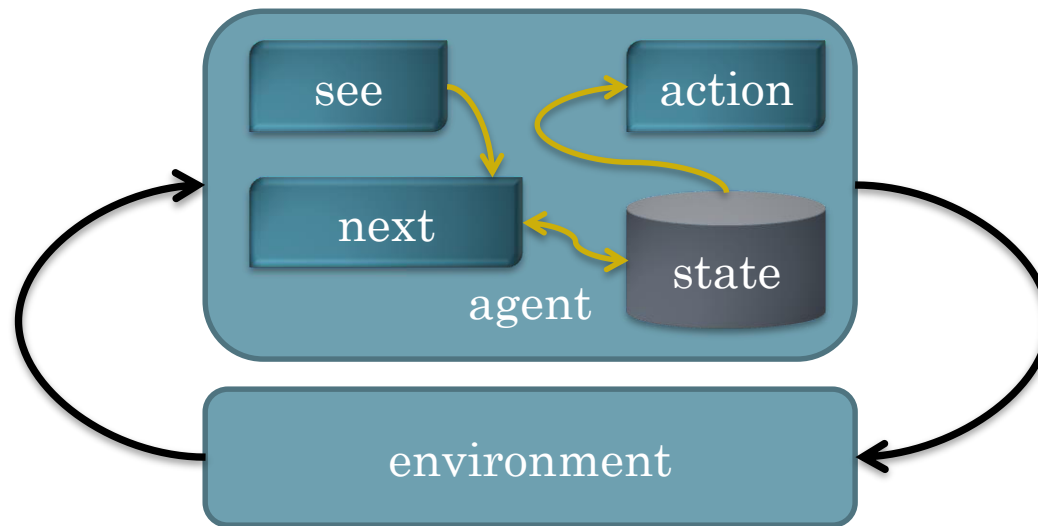
- Purely **reactive agents** in a **perception** system



- Such 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.
- In such a system, the *see* function maps environment states to **percepts** and the *action* function maps a **sequence of percepts to actions**.

GENERALIZATION OF AGENT SYSTEMS

- A further generalization are agents with **states**



- Such agents have some internal data structure, which is typically used to record information about the environment state and history.
- In such an agent, the *next* function maps an **internal state** and percept to an internal state and the action function maps **internal states to actions**.

UTILITY OF AGENTS

- To tell agents what to do without telling them how to do it, a **utility function** should be introduced.
- One possibility is to associate utilities with individual states — the task of the agent is then to bring about states that maximize utility.
- Task specification is then a function

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

which associated a real number with every environment state.

- A disadvantage is that it is difficult to specify a **long term view** when assigning utilities to individual states.

ASSIGNING UTILITY TO AGENT RUNS

- Another possibility is assigning a utility not to individual states, but to **runs** themselves:

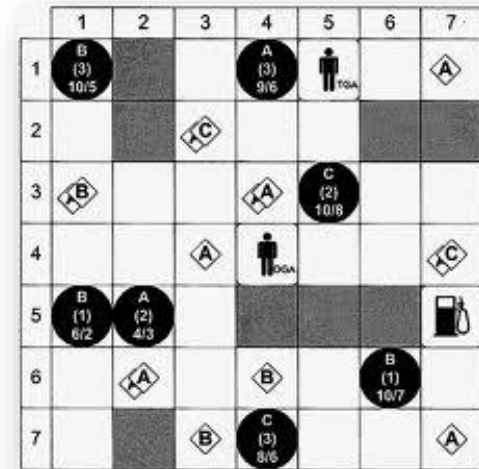
$$u: \mathcal{R} \rightarrow \mathbb{R}$$

- Such an approach takes an inherently long term view.
- Other possibility is to incorporate **probabilities** of different states emerging etc.
- The utility functions are often hard to be defined as people do not normally think in terms of explicit utilities.

EXAMPLE:

UTILITY IN THE TILEWORLD

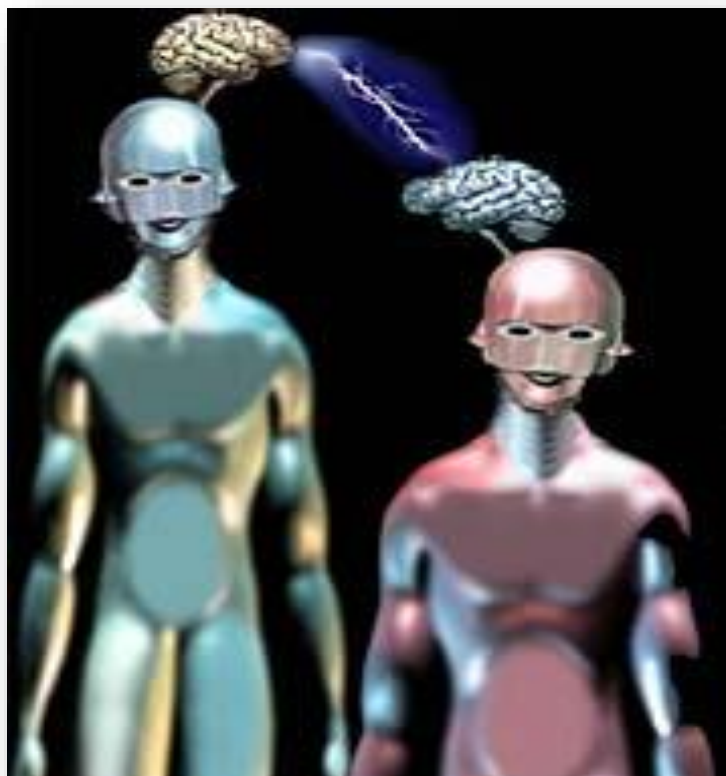
- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes.
- An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it.
- Holes have to be filled up with tiles by the agent. An agent scores points by filling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



UTILITY IN THE TILEWORLD EXAMPLE

- The utility function of the “Tileworld” example can be defined as the **ratio** between the number of holes filled and the number of holes that have appeared during the run.
- If an agent fills all the holes, the utility equals one, and if it fills no holes, the utility equals zero.
- A **probability of a run** of an agent in a given environment can also be introduced, and the expected utility of the agent can then be calculated.
- The expected utility of an agent is the sum of the utilities of all the possible agent runs multiplied by their probabilities.
- The optimal agent in an environment is the one that maximizes expected utility.

COMMUNICATION BETWEEN AGENTS



COMMUNICATION BETWEEN AGENTS

- Agents communicative acts are based in the **theory of speech acts**.
- This theory deals with languages as a means by which people achieve their goals and intentions, where spoken statements are regarded as a kind of physical actions, which sometimes even appear to change the state of the world.
- Several specifications have been proposed for agent communication systems, such as FIPA (*Foundation for Intelligent Physical Agents*)
- FIPA Agent Communication specifications deal with Agent Communication Language (ACL) messages, message exchange interaction protocols, speech act theory-based communicative acts and content language representations.

SPEECH ACTS

- Various different speech acts have been identified:
 - **representatives**: committing a speaker to the truth of the expressed proposition, e.g., “*It is raining.*”,
 - **directives**: causing the hearer to take a particular action, e.g., “*Please make the tea!*”,
 - **commissives**: committing the speaker to doing something, e.g., “*I promise to ...*”,
 - **expressives**: expressing the speaker's attitudes and emotions towards the proposition, e.g., “*Thank you!*”,
 - **declarations**: changing the reality in accordance with the proposition of the declaration, e.g. “*I declare a war on you!*”, or “*I find you guilty as charged*” ,

TYPOLGY OF SPEECH ACTS

- In general, a speech act can be seen to have two components:
 - **a performative verb**: (e.g., request, inform, inquire, ...)
 - **propositional content**: (e.g., “*the window is clean*”)

- speech act = “*Please clean the window!*”
performative = request
content = “*the window is clean*”
- speech act = “*The windows is clean.*”
performative = inform
content = “*the window is clean*”
- speech act = “*Is the window clean?*”
performative = inquire
content = “*the window is clean*”

SEMANTICS OF SPEECH ACTS

- Speech acts can be seen as operators that **change the beliefs** of the speaker and hearer.
- For example, the request speech act $request(s,h,a)$ could change the state of minds of the speaker s and hearer h as follows.
 - Before the speech act:
 - s believes h can do action a ,
 - s believes h believes can do action a ,
 - s believes h wants action a .
 - After the speech act:
 - h believes s believes s wants action a .

AGENT COMMUNICATION LANGUAGES

- Several specifications of agent communication languages (ACLs) have been proposed.
- One of the first such languages and is Knowledge Query and Manipulation Language (KQML) developed by the DARPA knowledge sharing initiative in the early 1990s
- The specification comprises two parts:
 - *the knowledge query and manipulation language (KQML); and*
 - *the knowledge interchange format (KIF)*

KNOWLEDGE QUERY AND MANIPULATION LANGUAGE

- KQML is an agent communication language that defines various acceptable ‘**communicative verbs**’, or performatives, such as:
 - ask-if (*“Is it true that ...?”*)
 - perform (*“Please perform the following action ...”*)
 - tell (*“It is true that ...”*)
 - reply (*“My answer is that”*)
- KIF is a language for expressing **message content**.

EXAMPLE KQML/KIF DIALOGUE

A to B: (ask-if
 (> (size chip1) (size chip2)))
B to A: (reply true)
B to A: (tell (= (size chip1) 20))
B to A: (tell (= (size chip2) 18))

FIPA SPECIFICATION

- More recently, the Foundation for Intelligent Physical Agents (FIPA) proposed several agent standards and specifications.
- The core of the specification is an ACL that is similar to KQML.
- FIPA specifications include 20 performative verbs, a message tracking and housekeeping protocol and the actual content of the message, e.g.,

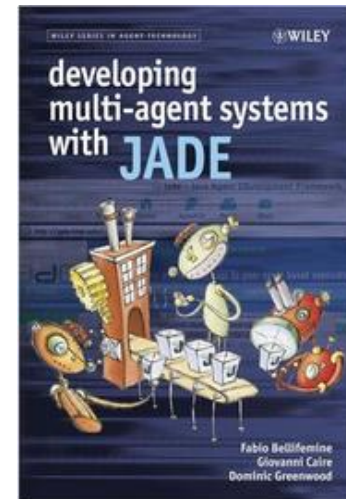
```
(inform
  :sender      agent1
  :receiver    agent5
  :content      (price good200 150)
  :language    sl
  :ontology     hpl-auction
)
```

FIPA PERFORMATIVE SPECIFICATIONS

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			

JAVA AGENT DEVELOPMENT FRAMEWORK

- JADE (Java Agent DEvelopment Framework) 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 support the debugging and deployment phases. (<https://jade.tilab.com>)
- A JADE-based system can be distributed across machines (which not need to share the same OS) and the configuration can be controlled via a remote GUI.
- The configuration can be even changed at run-time by moving agents
- Besides the agent abstraction, JADE provides a simple yet powerful **task execution and composition model**, peer to peer **agent communication** based on the **asynchronous message passing** paradigm from one machine to another, as and when required.



AGENT PROGRAMMING MODELS

- The usual software programming models are not always suitable for developing more complex intelligent agents in multi-agent systems.
- New, more advanced programming models emerge to solve particular problems in agent programming.
- An example of such a programming model is the **belief–desire–intention** (BDI) software model - it is called so, as it is characterised by the implementation of an agent's *beliefs*, *desires* and *intentions*.
- The BDI programming model allows a higher level of abstraction than the usual programming models and is closer to the human way of reasoning.

AGENT PROGRAMMING MODELS



- Several other agent programming models have been successfully developed within the agent research and development community and AI in general.
- One such example is a **cognitive architecture Soar** that enable creating representations and using appropriate forms of knowledge, such as procedural, declarative, episodic.
- Another such example is a cognitive architecture **OpenCog** for robot and virtual embodied cognition that defines a set of interacting components designed to give rise to human-equivalent artificial general intelligence.

BDI SOFTWARE MODEL

- The BDI software model implements the principal aspects of Bratman's theory of human practical reasoning.
- This model provides a mechanism for separating the activity of selecting a plan from the execution of currently active plans.
- Consequently, BDI agents are able to balance the time spent on deliberating about plans (choosing what to do) and executing those plans (doing it).
- Creating the plans in the first place (planning), is not within the scope of the model, and is left to the system designer and programmer.

ARCHITECTURAL COMPONENTS OF A BDI SYSTEM

- **Beliefs** represent the informational state of the agent, i.e., its beliefs about the world (including itself and other agents). Beliefs can also **include inference rules**, allowing forward chaining to lead to new beliefs.
- **Desires** represent the motivational state of the agent. They represent objectives (**goals**) or situations that the agent would like to accomplish or bring about.
- **Intentions** represent the deliberative state of the agent – what the agent has chosen to do. Intentions are desires to which the agent has to some extent committed, i.e., the agent has **begun executing a plan**.
- **Events** are triggers for reactive activity by the agent. An event may update beliefs, trigger plans or modify goals.

IMPLEMENTATION OF BDI SYSTEM COMPONENTS

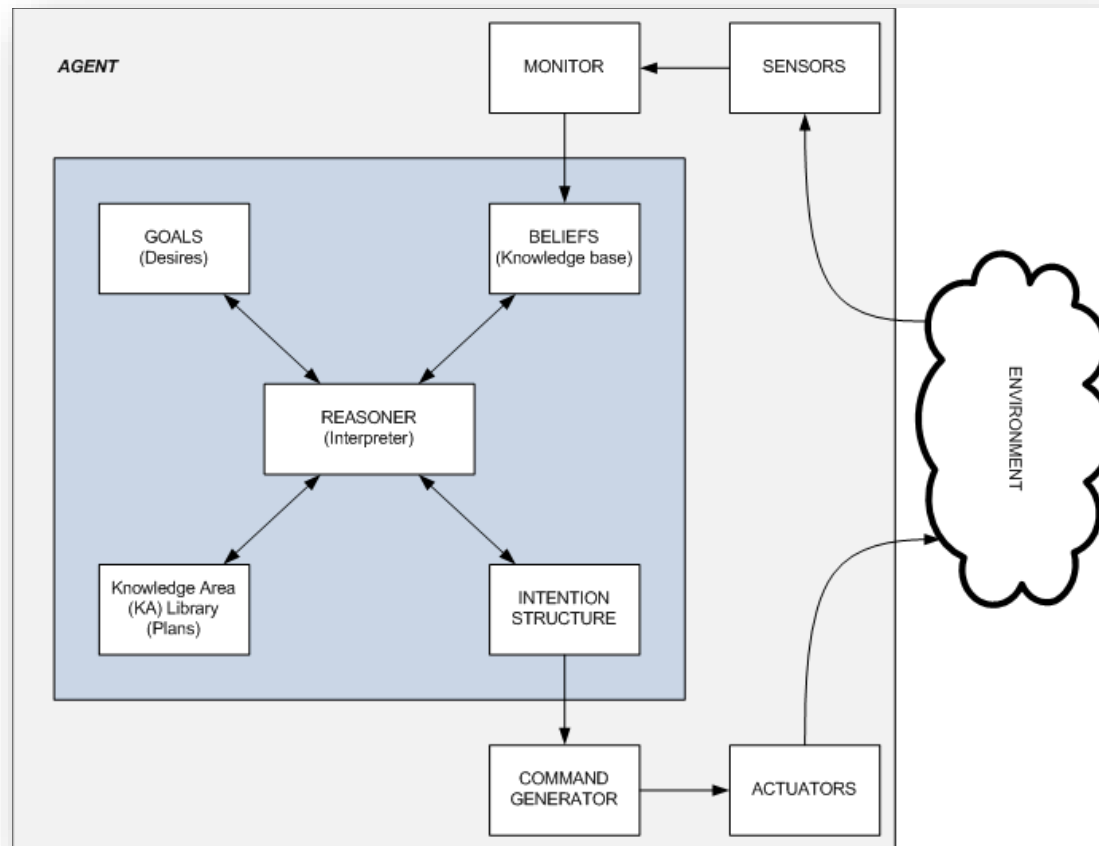
- **Beliefs** are usually stored in database (sometimes called a belief base or a **belief set**).
- **Desires** are organized as goals that have been adopted for active pursuit by the agent. The set of active desires must be consistent, e.g., one should not have **concurrent goals** to go to a party and to stay at home – even though they could both be desirable.
- **Intentions** are achieved by executing plans (recipes, sequences of actions) that could be initially only partially conceived, with details being filled in as they progress.
- **Events** may be generated externally via sensors, or may be generated internally to trigger decoupled updates or plans of activity.

BDI INTERPRETER PSEUDO CODE

- initialize-state
- repeat
 - options: option-generator (event-queue)
 - selected-options: deliberate (options)
 - update-intentions (selected-options)
 - execute()
 - get-new-external-events()
 - drop-unsuccessful-attitudes()
 - drop-impossible-attitudes()
- end repeat

PROCEDURAL REASONING SYSTEM

- A **procedural reasoning system** (PRS) is a framework for constructing real-time reasoning systems that can perform complex tasks in dynamic environments, and is based on the notion of a rational agent using the BDI software model.



BDI PROGRAMMING LANGUAGES

- Several programming languages that are **based on logic programming** and the BDI architecture for rational agents has been developed.
- Multi-agent platforms, such as JADE, usually do not include higher BDI programming languages.
- AgentSpeak is an example of an **agent-oriented** programming language.
- The language was originally called AgentSpeak(L), but became more popular as AgentSpeak, a term that is also used to refer to the **variants** of the original language.

AGENT-ORIENTED PROGRAMMING LANGUAGE AGENTSPEAK

- In 1996, Anand Rao created a logic-based agent programming language based on the BDI architecture and named it AgentSpeak(L).
- AgentSpeak is an **abstract agent programming language** aimed to help the understanding of the relation between practical implementations of the BDI architecture.
- It generalizes logic programming by adding agent's desire to satisfy the predicate, or to figure out whether it can or cannot be satisfied.
- It also incorporates a mechanism of simultaneous handling of several action plans that an agent creates or abandons depending on the generated events.

AGENTSPEAK SYNTAX AND SEMANTICS

- Beliefs are written as predicates (e.g., `location(robot, a)`).
- Goals are states of the system which the agent wants to bring about. Two types of goals are considered - an *achievement goal* and a *test goal*.
- An achievement goal, written as `!g(t)` states that the agent wants to achieve a state where `g(t)` is a true belief.
- A test goal, written as `?g(t)` states that the agent wants to test if the predicate `g(t)` is a true belief or not.
- Thus, if `location(robot, a)` is a belief (*being at the location a*), then `!location(robot, a)` is an achievement goal (*arrive to the location a*) and `?location(obstacle, a)` a test goal (*is there an obstacle at the location a*).

AGENT SPEAK SYNTAX AND SEMANTICS

- When an agent acquires a new goal or notices a change in its environment, it may trigger additions or deletions to its goals or beliefs.
- Addition is denoted by the operator $+$ and deletion is denoted by the operator $-$.
- If $b(t)$ is a belief atom and $!b(t)$ and $?b(t)$ are an achievement and a test goal, then possible *triggering events* are:

$+b(t), -b(t), +!b(t), -!b(t), +?b(t), -?b(t)$

AGENTSPEAK LANGUAGE SYNTAX

- An agent has plans which specify the means by which an agent should satisfy a goal.
- A plan consists of a **head** and a **body**.
- The head of a plan consists of a **triggering event** and a **context**, separated by a **:**
- The triggering event specifies why the plan was triggered, i.e., the addition or deletion of a belief or goal.
- The context of a plan specifies those beliefs that should hold in the agent's set of base beliefs, when the plan is triggered.
- The body of a plan is a sequence of goals or actions. It specifies the goals the agent should achieve or test, and the actions the agent should execute.

```
triggering_event : context <- body.
```

EXAMPLE PROGRAM

- We want to write a plan for a cleaning robot that gets triggered when some waste appears on a particular position.
- If the robot is in the same position as the waste, it will perform the action of picking up the waste, followed by achieving the goal of reaching the bin location, followed by performing the primitive action of putting it in the bin.
- This plan can be written as follows.

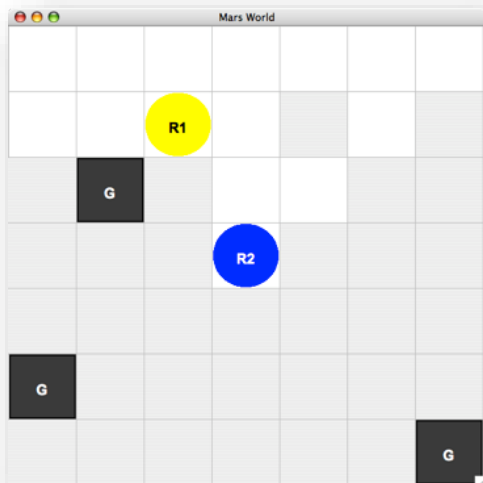
```
+location(waste,X):location(robot,X) &  
    location(bin,Y)  
  <- pick(waste);  
    !location(robot,Y);  
    drop(waste) .                                (P1)
```

EXAMPLE OF AGENT PLANS

```
+location(waste,X):location(robot,X) &  
    location(bin,Y)  
  <- pick(waste);  
    !location(robot,Y);  
    drop(waste). (P1)
```

```
+!location(robot,X):location(robot,X) <- true. (P2)
```

```
+!location(robot,X):location(robot,Y) &  
    (not (X = Y)) &  
    adjacent(Y,Z) &  
    (not (location(car, Z)))  
  <- move(Y,Z);  
    +!location(robot,X). (P3)
```



LIMITATIONS AND CRITICISMS

- The BDI software model is one example of a reasoning architecture for a **single rational agent**.
- The BDI model does not explicitly describe mechanisms for interaction with other agents and integration into a **multi-agent system**.
- BDI agents lack any specific mechanisms within the architecture to **learn from past behavior** and adapt to new situations.
- The model does not have (by design) any look ahead deliberation or **forward planning**.
- Classical decision theorists and planning research questions the necessity of having **all three attitudes**, however, on the other side, distributed AI researchers question whether the three attitudes are sufficient.

BDI PROGRAMMING MODEL LANGUAGES

- *JASON* (*AgentSpeak(XL)*)
(<http://jason.sourceforge.net>)
- JADEX
(<http://www.activecomponents.org>)
- JACK Agent Language (JAL)
(<https://aosgrp.com.au/jack/>)
- 2APL (*AgentSpeak*)
(<http://sourceforge.net/projects/apapl>)
- 3APL
(<http://www.cs.uu.nl/3apl/>)
- ...

QUESTIONS

- What is a multi-agent system?
- What is the difference between objects and agents in computer programming?
- Describe the abstract architectures for agents.
- Why the communication between agents is based on the Theory of speech acts?
- What is determined by the FIPA standard?
- Describe the Belief–desire–intention (BDI) software model.
- What are the basic characteristics of the abstract programming language AgentSpeak?