Multiagent Systems (SMA-UPC)

1. Introduction (to Agents and Multiagent Systems)

Javier Vázquez-Salceda
SMA-UPC

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
https://kemlg.upc.edu

Multiagent Systems (SMA-UPC)

Origins

- **Trends in Computer Science**
- **Agents and Multiagent Systems**
- **2 views of the Field**

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
https://kemlg.upc.edu

Computing now-a-days

- Internet Technology
  - Internet 2.0, Broadband access, exploding usage…
- Mobile "Telephony" Technology
  - 3G, iMode, WAP, Wireless PDAs, Bluetooth…
- Software Technology
  - JavaBeans, Soap, UDDI, JINI…
- Web Technology
  - XML, RDF, Servlets, JavaBeans, "Semantic Web"
- AI
  - Reasoning, Knowledge Representation, Agents…

Origins of MAS

- Five ongoing trends have marked the history of computing [M. Wooldridge]:
  - *ubiquity*;
  - *interconnection*;
  - *intelligence*;
  - *delegation*; and
  - *human-orientation*

5 trends (1 of 3)

- Ubiquity
  - The continual reduction in cost of computing capability has made it possible to introduce processing power into places and devices that would have once been uneconomic
  - As processing capability spreads, computation (and intelligence of a sort) becomes ubiquitous
- Interconnection
  - Computer systems today no longer stand alone, but are networked into large distributed systems
  - Since distributed and concurrent systems have become the norm, some researchers are putting forward theoretical models that portray computing as primarily a process of interaction

5 trends (2 of 3)

- Intelligence
  - The complexity of tasks that we are capable of automating and delegating to computers has grown steadily, to the limits that we can define as *intelligent*.
- Delegation
  - Computers are doing more for us – without our intervention
  - We are *giving control* to computers, even in safety critical tasks
- Human orientation
  - The movement away from machine-oriented views of programming toward concepts and metaphors that more closely reflect the way we ourselves understand the world
  - Programmers conceptualize and implement software in terms of higher-level – more human-oriented – abstractions

5 trends (3 of 3)

- Delegation and Intelligence imply the need to build computer systems that can act effectively on our behalf
- This implies:
  - The ability of computer systems to act *independently*
  - The ability of computer systems to act in a way that *represents our best interests* while interacting with other humans or systems
- Interconnection and Distribution have become core motifs in Computer Science
- But Interconnection and Distribution, coupled with the need for systems to represent our best interests, implies:
  - Systems that can *cooperate* and *reach agreements* (or even *compete*) with other systems that have different interests (much as we do with other people)

Computer Science progression

- These issues were not studied in Computer Science until recently
- All of these trends have led to the emergence of a new field in Computer Science: *multiagent systems*
- Wooldridge says that programming has progressed through:
  - machine code;
  - assembly language;
  - machine-independent programming languages;
  - sub-routines;
  - procedures & functions;
  - abstract data types;
  - objects;

to *agents*.

Agents and Multiagent Systems

- An agent is a computer system that is capable of *independent* action on behalf of its user or owner (figuring out what needs to be done to satisfy design objectives, rather than constantly being told)
- A multiagent system is one that 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

Agents and Multiagent Systems

- Building Agents, we address questions such as:
  - How do you state your preferences to your agent?
  - How can your agent compare different deals from different vendors? What if there are many different parameters?
  - What algorithms can your agent use to negotiate with other agents (to make sure you get a good deal)?
- In Multiagent Systems, we address questions such as:
  - How can cooperation emerge in societies of self-interested agents?
  - What kinds of languages can agents use to communicate?
  - How can self-interested agents recognize conflict, and how can they (nevertheless) reach agreement?
  - How can autonomous agents coordinate their activities so as to cooperatively achieve goals?

Agent Design, Society Design

- Two key problems:
  - How do we build agents capable of independent, autonomous action, so that they can successfully carry out tasks we delegate to them?
  - How do we build agents that are capable of interacting (cooperating, coordinating, negotiating) with other agents in order to successfully carry out those delegated tasks, especially when the other agents cannot be assumed to share the same interests/goals?
    - The first problem is *agent design* [in this course we cover this in *3. Reasoning in Agents*].
    - The second is *society design* (micro/macro) [in this course we cover this in *4. Multiagent Systems Design*].

Multiagent Systems is Interdisciplinary

- The field of Multiagent Systems is influenced and inspired by many other fields:
  - Philosophy
  - Logic
  - Game Theory
  - Economics
  - Social Sciences
  - Ecology
- This can be both a strength (infusing well-founded methodologies into the field) and a weakness (there are many different views as to what the field is about)

2 Views of the Field

- *Agents as a paradigm for software engineering*: Software engineers have derived a progressively better understanding of the characteristics of complexity in software. It is now widely recognized that *interaction* is probably the most important single characteristic of complex software
- Over the last two decades, a major Computer Science research topic has been the development of tools and techniques to model, understand, and implement systems in which interaction is the norm

2 Views of the Field

- *Agents as a tool for understanding human societies*: Multiagent systems provide a novel new tool for simulating societies, which may help shed some light on various kinds of social processes.
- This has analogies with the interest in “theories of the mind” explored by some artificial intelligence researchers

Standards: FIPA (*www.fipa.org*)

- International Agent Standard
  - Started in 1996 to provide agent technology specifications.
  - Part of IEEE (since 2005) as 11th standards committee.
- Includes standards for
  - Communication: Agent Communication Languages, Content Languages, Semantic Framework
  - Infrstructure: directories, message transport, naming, etc…
- Recent trends
  - Moved toward web technology (XML, RDF, HTTP)
  - Plug and Play architectures
  - Moves for Java standard
- Next phase
  - Verification
  - Significant take-up
  - Demonstration of Value

Hot topic: Open Service Environments

- Explosion of Agent technology with new uses for Open Service Environments
- Automation of Services
  - Proactive, responsible, intelligent, peer to peer
- Dynamic Composition of Services
  - Automated discovery, automated coordination, “Just in Time” Enterprises, Virtual Companies
- Semantics
  - HTML won’t do anymore
  - “Semantic Web”
  - Service-level semantics
  - Semantics for E-commerce
  - Service-Oriented Architectures’ frameworks

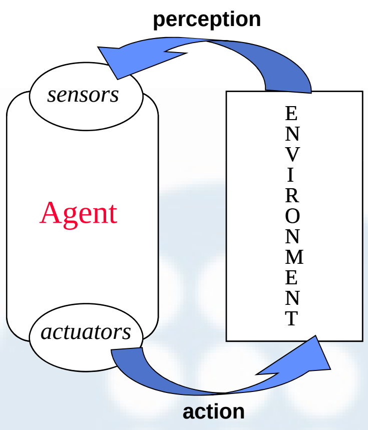
Agent types and architectures

- Agent properties
- Environment properties
- Agent types
- Abstract architecture

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
https://kemlg.upc.edu

Agent Properties
Autonomy

- *An agent is a computer system capable of* autonomous action *in some environment in order to meet its* design objectives
- Usually the environment is *complex* and *dynamic*, and agents should interact with it in real time.



- o Main property: Autonomous

capable of acting independently, exhibiting control over their internal state

Agent Properties
Autonomy, Flexibility

- Trivial (non-interesting) agents:
  - thermostat
- Def. 2: *An* intelligent agent *is a computer system capable of* flexible *autonomous action in some environment*
- By *flexible*, we mean:
  - *reactive* (response capability)
  - *pro-active* (taking initiative)
  - *social* (interacting with others)

Agent Properties
Reactivity

- If a program’s environment is guaranteed to be fixed, the program need never worry about its own success or failure – program just executes blindly
  - Example of fixed environment: compiler
- The real world is not like that: things change, information is incomplete. Many (most?) interesting environments are *dynamic*
- Software is hard to build for dynamic domains: program must take into account possibility of failure – ask itself whether it is worth executing!
- 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)

Agent Properties
Proactiveness

- Reacting to an environment is easy (e.g., stimulus → response rules)
- But we generally want agents to *do things for us*
- Hence *goal directed behavior*
- Pro-activeness = generating and attempting to achieve goals; not driven solely by events; taking the initiative
- Recognizing opportunities

Agent Properties
Social Ability

- The real world is a *multi*-agent environment: we cannot go around attempting to achieve goals without taking others into account
- Some goals can only be achieved with the cooperation of others
- Similarly for many computer environments: witness the Internet
- *Social ability* in agents is the ability to interact with other agents (and possibly humans) via some kind of *agent-communication language*, and perhaps cooperate with others

Agent Properties
Balancing Reactive and Goal-Oriented Behavior

- We want our agents to be reactive, responding to changing conditions in an appropriate (timely) fashion
- We want our agents to systematically work towards long-term goals
- These two considerations can be at odds with one another
- **Reactivy vs. Deliberation balance**
  - Designing an agent that can balance reactivity and deliberation (reason about long term goals) remains an open research problem

Other Agent Properties
(desireable, not mandatory)

- *mobility*
  - the ability of an agent to move around an electronic network
- *veracity*
  - an agent will not knowingly communicate false information
- *benevolence*
  - agents do not have conflicting goals, and that every agent will therefore always try to do what is asked of it
- *rationality*
  - agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit
- *learning/adaption*
  - agents improve performance over time

Environment properties
Accessible vs. inaccessible

- An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state
- Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible
- The more accessible an environment is, the simpler it is to build agents to operate in it

Environment properties
Deterministic vs. non-deterministic

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

Environment properties
Episodic vs. non-episodic

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

Environment properties
Static vs. dynamic

- A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent
- A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control
- Other processes can interfere with the agent's actions (as in concurrent systems theory)
- The physical world is a highly dynamic environment

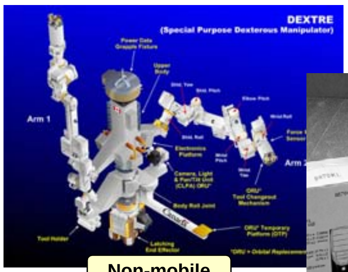
Environment properties
Discrete vs. continuous

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

Agent types
Physical (embodied) Agents vs. Software Agents

- Software agents’ environment is a virtual one
  - Single machine, intranet, internet
  - Interact with other software agents, with sw modules, services
  - Interact with humans through human interfaces
- Physical agents or embodied agents
  - Interact with real world (sensors, actuators connected to real world)
  - Problems of perception and action
  - Best known example: *Robots*.

Agent types
Robots



Non-mobile

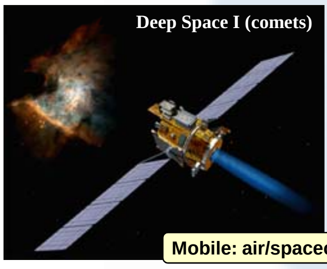
Lunokhod (Moon)

Mobile: weeled

Spirit (Mars)

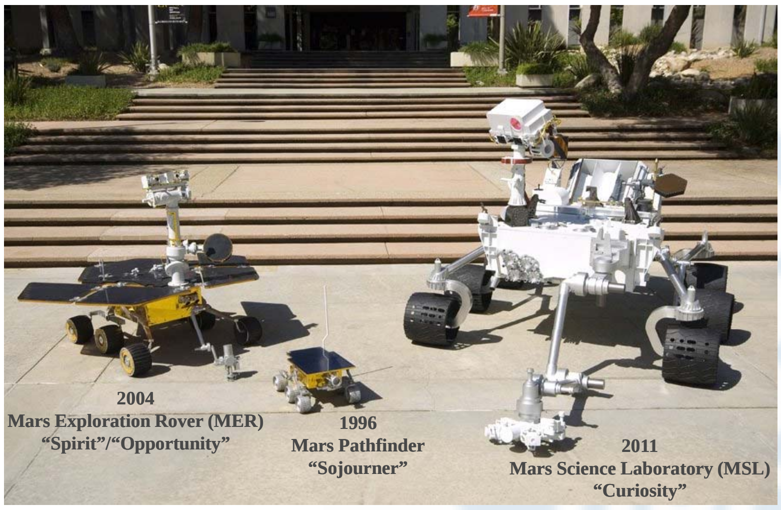
SONY aibo

Mobile: legged


Deep Space I (comets)

Mobile: air/spacecrafts

Agent types
Example of state-of-art Agent technology: Mars Robots



Agent Types
Software agents

- **Internet agents** (search and information extraction/management from Internet)
- **Collaborative agents** (they coordinate with other agents to solve a common task)
  - To solve problems too complex for a single agent
  - To solve problemes distributed in nature
  - To interconnect already existing, heterogeneous systems (→ **Agentification**)
- **Interface agents** (they collaborate with a human user to solve a task, or to act on behalf of the user.
- **Mobile SW agents** (they can move from one computer to another)

Agent types
Internal architecture

- **Purely Reactive Agents** (with no internal state)
- **Reactive Agents with internal state**
- **Delliberative Agents** (goal-oriented behaviour)
- **Hybrid Agents** (combine reactive and delliberative behaviour)

Agent architectures

- **Abstract architecture for agents**
- **Architectures for Multiagent systems**

Knowledge Engineering and Machine Learning Group
UNIVERSITAT POLITÈCNICA DE CATALUNYA
https://kemlg.upc.edu

Abstract Architecture for Agents

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

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

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

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

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

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

Abstract Architecture for Agents

- Let:
  - R be the set of all such possible finite sequences (over E and Ac)
  - R^{Ac} be the subset of these that end with an action
  - R^{E} be the subset of these that end with an environment state

State Transformer Functions

- A *state transformer* function represents behavior of the environment:

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

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

Agents

- Agent is a function which maps runs to actions:

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

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

Systems

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

Systems

- Formally, a sequence

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

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

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

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

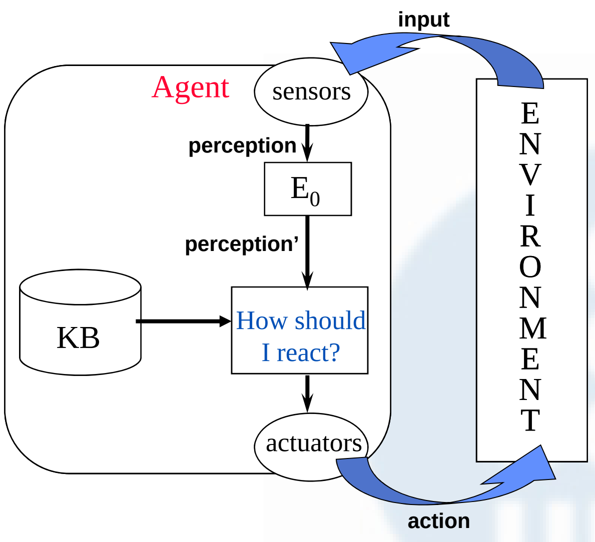
Purely Reactive Agents

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

$$action : E \to Ac$$

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

Purely Reactive Agents



Purely Reactive Agents

```
function pra(percept) returns (action)
 static rules
        state  ← interpret-input(percept)
        rule   ← rule-match(state,rules)
        action ← rule-action[rule]
        return  action
```

Formally…

- We define 2 functions
  - The *see* function is the agent's ability to observe its environment,
  - The *action* function represents the agent's decision making process

- *Output* of the *see* function is a *percept*:

$$see : E \to Per$$

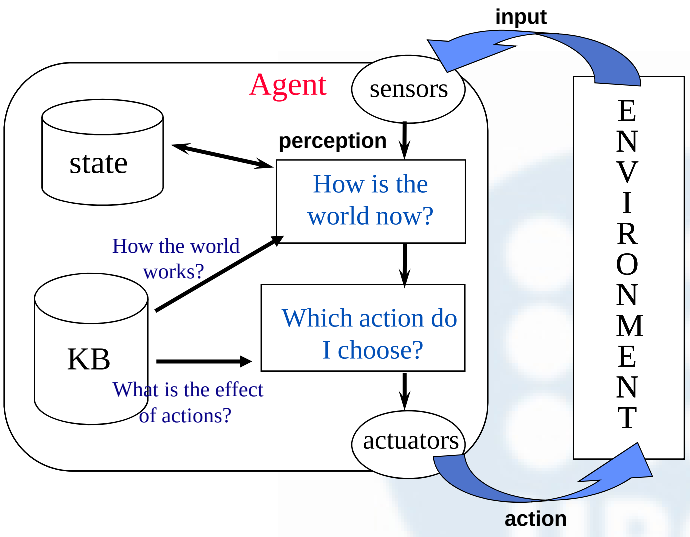
which maps environment states to percepts,

- and *action* is now a function

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

which maps sequences of percepts to actions

Reactive Agents with internal state



Reactive Agents with internal state

```
Function reactive-agent-with-state(percept) returns action
Static    state ;a world description
          rules ;a set of, e.g., if-then rules

state  ⟵ update-state(state,percept)
rule   ⟵ rule-match(state,rules)
action ⟵ rule-action[rule]
state  ⟵ update-state(state,action)

return ⟵ action
```

Formally…

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

$$see : E \rightarrow Per$$

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

$$action : I \rightarrow Ac$$

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

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

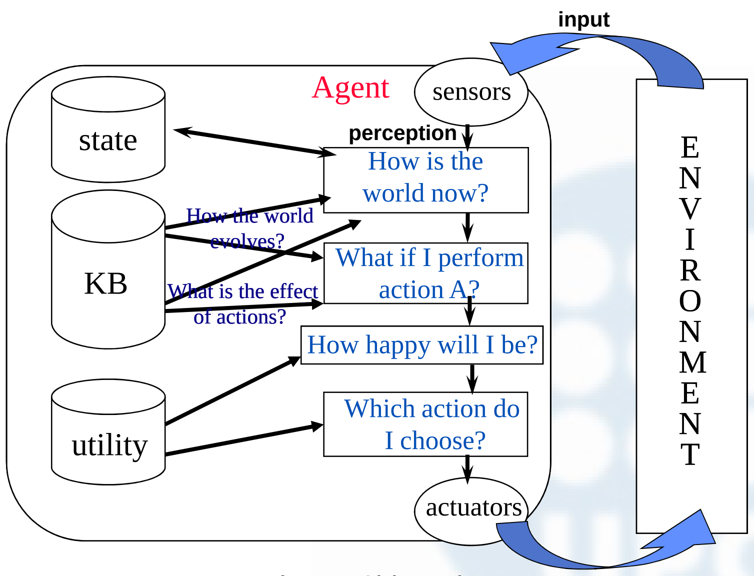
Formally…

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

Tasks for Agents

- We build agents in order to carry out *tasks* for us
- The task must be *specified* by us…
- But we want to tell agents what to do *without* telling them how to do it
- One possibility: associate *utilities* with individual states — the task of the agent is then to bring about states that maximize utility

Delliberative Agents (with expected utilities)



Utility Functions over States

- A task specification is a function

$$u : E \rightarrow \#$$

which associates a real number with every environment state

- But what is the value of a *run*…
  - minimum utility of state on run?
  - maximum utility of state on run?
  - sum of utilities of states on run?
  - average?

- Disadvantage: difficult to specify a *long term* view when assigning utilities to individual states
(One possibility: a *discount* for states later on.)

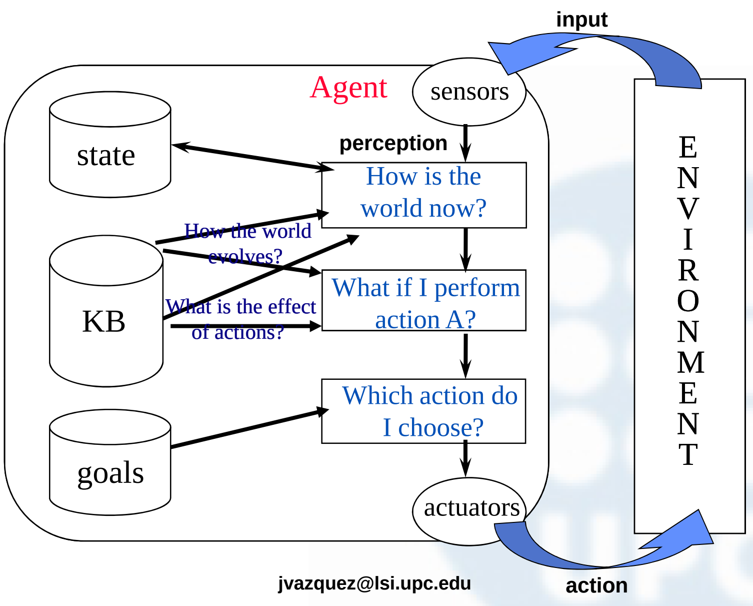
Utilities over Runs

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

$$u : R \rightarrow \#$$

- Such an approach takes an inherently *long term* view
- Other variations: incorporate probabilities of different states emerging
- Difficulties with utility-based approaches:
  - where do the numbers come from?
  - we don't think in terms of utilities!
  - hard to formulate tasks in these terms

Delliberative Agents (with explicit goals)



Delliberative Agents (with explicit goals)

```
Function reactive-agent-with-goals(percept) returns action
Static    state ; a world description
          rules ;a set of, e.g., if-then rules
          goals ;a list of goal states

state ← update-state(state,percept)
appliable-rules ← rule-match(state,rules)
possible-actions ← rule-action[rule]
action ← goal-oriented-selection[possible-actions]
state ← update-state(state,action)

return    action
```

Formally…

- It gets far more complex to do a proper formalization
  - Goal semantics
  - Relationship between goals, action and states
  - Relationship between perception and knowledge
- [We will see this in *3. Reasoning in Agents*]

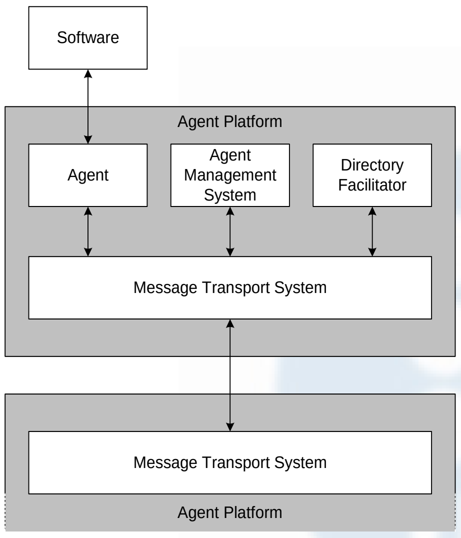
Multiagent Systems' architecture

- Agents in a multiagent system tend to interact through a middleware layer
- This middleware provides connectivity between agents, solving low-level connectivity issues
  - Communication methods
- Sometimes this middleware is called *agent platform*

Communication methods

- *Blackboard systems*
  - Agents communicate information through a common data structure, accessible by everybody
  - Problem: if there is no middleware to provide some concurrency, it tends to become a bottleneck.
- *Message passing*
  - Agents communicate directly by means of messages
  - The agent platform usually acts as message router
  - Common communication language (e.g. FIPA-ACL)
  - Common communication protocols (message format, steps in a communication)

FIPA Architecture for Agent Platforms



Components of an Agent Platform

- **Agent:** a program providing a list of services
- **Directory Facilitator (DF)** is an agent which provides a Yellow Pages service within the platform (knows the services that agents within the platform provide)
  - *register, deregister, modify, search*
- **Agent Management System (AMS)** is an agent controlling access and usage of the agent platform. It knows the platform and agents' "addresses" and provides a White Pages service (knows the routing addresses for agents within and in other platforms)
- **Message Transport Service (MTS)** is used to enable communication between agents in different platforms.

Agent Platform tasks

- Suspend temporally an agent execution
- Stop an agent execution
- Resume/continue agent execution
- Start an agent
- Platform resource management

Discussion about Agents

- **Agents vs. Objects**
- **Agents vs. Expert Systems**

Agents vs. Objects

- Are agents just objects by another name?
- Object:
  - encapsulates some state
  - communicates via message passing
  - has methods, corresponding to operations that may be performed on this state

Agents vs. Objects

- Main differences:
  - *agents are autonomous:*
    agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent
  - *agents are smart:*
    capable of flexible (reactive, pro-active, social) behavior, and the standard object model has nothing to say about such types of behavior
  - *agents are active:*
    a multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of active control
- As Wooldridge says:
  - objects do it for *free*…
  - …agents do it because they *"want"*
  - …agents do it for *"money"*

Agents vs. Expert Systems

- Aren't agents just expert systems with another name?
  - Expert systems are deliberative
  - e.g. MYCIN
- Main differences:
  - agents *situated in an environment:*
    MYCIN is not aware of the world — only information obtained is by asking the user questions
  - agents *act:*
    MYCIN does not operate on patients
- Some *real-time* (typically process control) expert systems *are* agents

References

[1] Luck, M., McBurney, P., Shehory, Onn, Willmott, S. "Agent Technology: Computing as interaction. A Roadmap to Agent Based Computing". Agentlink, 2005. ISBN 085432 845 9

[2] Wooldridge, M. "Introduction to Multiagent Systems". John Wiley and Sons, 2002.

[3] Russell, S. & Norvig, P. "Artificial Intelligence: A Modern Approach" Prentice-Hall Series in Artificial Intelligence. 2009
ISBN 0-13-103805-2

[4] Haddadi, A. "Communication and Cooperation in Agent Systems: A Pragmatic Theory" Lecture Notes in Artificial Intelligence #1056. Springer-Verlag. 1996. ISBN 3-540-61044-8

[5] Rosenschein, J. & Zlotkin, G. "Rules of Encounter. Designing Conventions for Automated Negotiation among Computers". MIT Press. 1994 ISBN 0-262-18159-2

[6] Weiss, G. "Multiagent Systems: A modern Approach to Distributed Artificial Intelligence". MIT Press. 1999. ISBN 0262-23203

These slides are based mainly in material from [2] and [1], with some additions from material by U. Cortés, J.Padget, A. Moreno and Steve Willmott