Lecture 2: Intelligent Agents

An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
What is an Agent?

1. What is an Agent?

The main point about agents is they are autonomous: capable of acting independently, exhibiting control over their internal state.

Thus: an agent is a computer system capable of autonomous action in some environment.
Lecture 2
An Introduction to Multiagent Systems

Trivial (non-interesting) agents:
- thermostat;
- UNIX daemon (e.g., biff).

An intelligent agent is a computer system capable of flexible autonomous action in some environment.

By flexible, we mean:
- reactive;
- pro-active;
- social.

- UNIX daemon (e.g., biff).
- thermostat;

Trivial (non-interesting) agents:
If a program’s environment is guaranteed to be fixed, it need never worry about its own success or failure — it just executes blindly. If a program’s environment is guaranteed to be fixed, the time for the response to be useful 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!

Example of fixed environment: compiler.

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).

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).
1.2 Proactiveness

Reactivity is easy (e.g., stimulus response)

Recognising opportunities.

Pro-activeness = generating and attempting to achieve goals; not driven solely by events; taking the initiative.

- Hence goal directed behaviour.
- But we generally want agents to do things for us.
- Reacting to an environment is easy (e.g., stimulus response)

Hence goal directed behaviour.

- Reacting to an environment is easy (e.g., stimulus response)
- Recognising opportunities.
- Pro-activeness = generating and attempting to achieve goals; not driven solely by events; taking the initiative.
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.
Other properties, sometimes discussed in the context of agency:

- Learning/adaptation: agents improve performance over time.
- 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 inssofar as its beliefs permit.
- 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;
- Mobility: the ability of an agent to move around an electronic network;
- Other Properties

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
2.1 Agents and 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.

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
Main differences:

- **Agents are active:**

  Types of behavior:
  
  The standard object model has nothing to say about such capable of flexible (reactive, pro-active, social) behavior, and

- **Agents are smart:**

  Perform an action on request from another agent:
  
  In particular, they decide for themselves whether or not to agents embody stronger notion of autonomy than objects, and

- **Agents are autonomous:**

  An introduction to Multiagent Systems
Objects do it for free...

- agents do it for money.
- agents do it because they want to.
Aren't agents just expert systems by another name?

Expert systems typically embody disembodied 'expertise' about some domain of discourse (e.g., blood diseases).

Example: MYCIN knows about blood diseases in humans.

- MYCIN has a wealth of knowledge about blood diseases, in the form of rules.
- A doctor can obtain expert advice about blood diseases by giving MYCIN facts, answering questions, and posing queries.

MYCIN: "I have anemia..."
Doctor: "You have a rare blood disease..."
MYCIN: "I don't know..."
Doctor: "You have a rare blood disease..."
MYCIN: "I don't know..."

It has a wealth of knowledge about blood diseases in humans.

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Main differences:

- agents situated in an environment

Some real-time (typically process control) expert systems are

MYCIN is not aware of the world — only information obtained by asking the user questions.

MYCIN does not operate on patients.

- agents act: an introduction to multiagent systems
Aren't agents just the AI project? Isn't building an agent what AI is all about? Aren't agents just the AI project?

2.3 Intelligent Agents and AI

So, don't we need to solve all of AI to build an agent...? All aims to build systems that can (ultimately) understand natural language, recognize and understand scenes, use common sense, think creatively, etc — all of which are very hard.
When building an agent, we simply want a system that can choose the right action to perform, typically in a limited domain. We do not have to solve all the problems of AI to build a useful agent.

Oren Etzioni, speaking about the commercial experience of NETBOT, Inc:

We made our agents dumber and dumber and dumber... until finally they made money.

A little intelligence goes a long way!

When building an agent, we simply want a system that can choose the right action to perform, typically in a limited domain.

Oren Etzioni, speaking about the commercial experience of NETBOT, Inc:

We made our agents dumber and dumber and dumber... until finally they made money.

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

3 Environments

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
As we have already mentioned, a deterministic environment is

\[ \text{D} \]

an action.

is no uncertainty about the state that will result from performing

one in which any action has a single guaranteed effect — there

As we have already mentioned, a deterministic environment is

\[ \text{D} \]

Non-deterministic environments present greater problems for the

agent designer.

The physical world can to all intents and purposes be regarded

as non-deterministic.
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.
The physical world is a highly dynamic environment. A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agents. A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.

Static vs. Dynamic.
A discrete environment is one where the actions and percepts are fixed. Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one. An environment is discrete if there are a fixed, finite number of actions and percepts in it. Russell and Norvig give a chess game.
When explaining human activity, it is often useful to make statements such as the following:

- Janine took her umbrella because she believed it was going to rain.
- Michael worked hard because he wanted to possess a PhD.
- When explaining human activity, it is often useful to make statements such as the following:

The attitudes employed in such folk psychological descriptions are called the *intentional notions*. The attitudes such as believing and wanting (as in the above examples), hoping, fearing, and so on, attributed to human behavior is predicted and explained through the intentional notions.

These statements make use of a **folk psychology** by which...
The philosopher Daniel Dennett coined the term intentional system to describe entities whose behaviour can be predicted by attributing beliefs, desires, and so on, to.

Dennett identifies different grades of intentional system:

- A first-order intentional system has beliefs and desires (etc.) but no beliefs and desires about beliefs and desires (and no doubt sophisticated; it has beliefs and desires about beliefs and desires (and so on, to own). A second-order intentional system is more sophisticated; it has beliefs and desires about beliefs and desires (and no doubt other intentional states) — both those of others and its own.

Is it legitimate or useful to attribute beliefs, desires, and so on, to computer systems?
An Introduction to Multiagent Systems

McCarthy argued that there are occasions when the intentional stance is appropriate. When applied to entities whose structure is incompletely known, most useful is ascription of mental qualities to machines, such as thermostats and computer operating systems, but is known structure. Ascription of mental qualities is most straightforward for machines of known structure: A computer, for example, is a machine whose past or future behaviour can be predicted by expressing an appropriate theory of mental qualities. Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. A computer, in a particular situation, may require mental qualities or the state of the machine to be expressed. Reasonably briefly what is actually known about a computer is the state of the machine and the user's expectations about it. It is perhaps never logically required for humans, but expressing reasonably briefly what is actually known about a person's past or future behaviour helps us understand the structure of the machine, its past or future about the machine that it expresses about a person. It is useful when the machine is legitimate when such an ascription expresses the same information about a machine that it expresses about a person. 

To ascribe beliefs, free will, intentions, consciousness, abilities, or wants to a machine is legitimate when such an ascription expresses the same information about a machine that it expresses about a person. It is useful when the machine is legitimate when such an ascription expresses the same information about a machine that it expresses about a person. It is useful when the machine is legitimate when such an ascription expresses the same information about a machine that it expresses about a person. It is useful when

http://www.csc.liv.ac.uk/~mjw/pubs/imas
Lecture 2: An Introduction to Multiagent Systems

What objects can be described by the intentional stance?

As it turns out, more or less anything can... consider a light switch:

But most adults would find such a description absurd!

'... flicking the switch is simply our way of communicating our desires. (Yoav Shoham)

\[ \text{flicking the switch = \text{transmitting current}} \]

\[ \text{when we believe we want it transmitted and not otherwise} \]

\[ \text{cooperative agent with the capability of transmitting current when we flick the switch as a (very)} \]

\[ \text{light switch} \]

Why is this?

http://www.csc.liv.ac.uk/~mjw/pubs/imas/22
The answer seems to be that while the intentional stance is such an abstraction, operation — low level explanations become impractical. But with very complex systems, a mechanistic explanation of its behaviour may not be practicable.

As computer systems become ever more complex, we need powerful abstractions and metaphors to explain their behaviour. Put crudely, the more we know about a system, the less we need mechanistic descriptions of its behaviour. (Yoav Shoham)

... it does not buy us anything, since we essentially description is consistent.

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
The intentional notions are abstraction tools, which provide us with a convenient and familiar way of describing, explaining, and predicting the behaviour of complex systems.

Remember: most important developments in computing are based on new abstractions and predicting the behaviour of complex systems. Agents, and agents as intentional systems, represent a further abstraction.

So agent theorists start from the (strong) view of agents as objects.

– procedural abstraction;

– abstract data types;

– procedural abstraction;

The intentional notions are thus abstraction tools, which provide...
This is an important argument in favour of agents.

Programming computer systems?

Tool in computing — to explain, understand, and, crucially, explain their behaviour without having to understand how the system actually works.

So why not use the intentional stance as an abstraction tool for objects, ...?

Abstract mechanisms witness procedural abstraction, ADTs, ... This intentional stance is an abstraction tool — a convenient way to talk about complex systems, which allows us to predict and explain their behaviour without having to understand how the mechanism actually works.
Other 3 points in favour of this idea:

- **Characterising Agents**
- **Nested Representations**
- **Explaining Agents**

Characterising Agents
It provides us with a familiar, non-technical way of understanding agents.

Nested Representations
It gives us the potential to specify systems that include representations of other systems.

Explaining Agents
It is widely accepted that such nested representations are essential for agents that must cooperate with other agents.
This view of agents leads to a kind of post-declarative

Post-Declarative Systems
Later... We'll examine this point in detail later.
\[\text{interpretation in terms of the states of a process; return to this}\]
• In DS theory, knowledge is \textit{grounded} — given a precise

\[\text{the message,}\]

\[\text{IF process } i \text{ knows process } j \text{ has received message } m_1\]

\[\text{THEN process } i \text{ should send process } j \text{ message } m_2.\]

The rationale is that when constructing protocols, one often encounters reasoning such as the following:

• The development of knowledge-based protocols.

• In distributed systems theory, \textit{logics of knowledge} are used in

• We find that researchers from a more mainstream computing

\[\text{An aside...}\]
Assume the environment may be in any of a finite set \( E \) of \( n \) discrete, instantaneous states:

\[ e_0, e_1, \ldots, e_n \in E \]

Agents are assumed to have a repertoire of possible actions:

\[ \{ \ldots, \rho, \alpha \} = \mathcal{A} \]

A run, \( r \), of an agent in an environment is a sequence of

\[ \{ \ldots, \rho, a, \alpha \} = \mathcal{E} \]

Interleaved environment states and actions:

\[ \mathcal{E} \ni \mathcal{A} \]

Abstract Architectures for Agents
Let:

- \( E \) be the set of all possible finite sequences over \( E \) and \( A \); and
- \( A(c) \) for all \( c \)
- \( R \) be the set of all such possible finite sequences (over \( F \) and
- \( E \) be the subset of those that end with an environment state.

- \( R_e \) be the subset of those that end with an action; and
- \( R_a \) be the subset of those that end with an action; and
A state transformer function represents behaviour of the environment:

\[ f : E \rightarrow E \]

where: \( E \) is a set of environment states, \( E_0 \in E \) is the initial state.

Formally, we say an environment \( E_{\text{Environ}} \) is a triple \( \langle E, E_0, f \rangle \).

- History dependent.
- Non-deterministic.

If \( f \) is the empty function, then the environment has ended its run.

Note that environments are:

\[ \mathcal{E} \]

A state transformer function represents behaviour of the environment.
An agent makes a decision about what action to perform based on the history of the system that it has witnessed to date. Let $A_g$ be the set of all agents. An agent $A_g$ makes a decision about what action $a$ to perform based on the history $R_f$ of the system:

$$A_g : R_f \rightarrow a$$

An agent is a function which maps runs to actions.
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 \( A_g \) in environment \( E_n \) by \( R(A_g, E_n) \).

We assume \( R(A_g, E_n) \) contains only terminated runs. A system is a pair containing only terminated runs.
Formally, a sequence $e$ represents a run of an agent $Ag$ in environment $Env$ if:

$$((e^n_0, \ldots, e_0, e_0, e_0, e_0, e_0, \ldots))^\delta = e^n_{\xi}$$

where $ \xi \in \mathbb{N}$

3. for $n > 0$,

2. $Ag = e_0 \delta (e_0)^0$; and

1. $e_0$ is the initial state of $Env$.

Formally, a sequence $e$ represents a run of an agent $Ag$ in environment $Env$ if:

$$(e_0, e_1, e_2, \ldots)$$
Some agents decide what to do without reference to their history — they base their decision making entirely on the present. This is what we call purely reactive agents. A thermostat is an example of such an agent:

\[
\text{action} = \begin{cases} 
\text{on} & \text{if } \text{temperature } \text{OK} \\
\text{off} & \text{otherwise.}
\end{cases}
\]

We call such agents purely reactive.
Now introduce perception system:

 perceive
Lecture 2 An Introduction to Multiagent Systems

The see function is the agent's ability to observe its environment, whereas the action function represents the agent's ability to observe its environment.

Output of the see function is a percept:

- The see function is the agent's ability to observe its environment.

Making process:

- Whereas the see function represents the agent's ability to observe its environment.
We now consider agents that maintain state:

8 Agents with State
These agents have some internal data structure, which is typically used to record information about the environment state and history.

Let $I$ be the set of all internal states of the agent.

An additional function $\text{next}$ is introduced, which maps an internal state and percept to another internal state:

$$\text{next} : I \times \text{Per} \rightarrow I$$

The action-selection function $\text{action}$ is now defined as a mapping:

$$\text{action} : I \rightarrow \text{Ac}$$

The perception function $\text{see}$ for a state-based agent is unchanged:

$$\text{see} : E \rightarrow \text{Per}$$

- The set of all internal states of the agent, $I$, is typically used to record information about the environment state.
- These agents have some internal data structure, which is
8.1 Agent control loop

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

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
We build agents in order to carry out tasks for us. But we want to tell agents what to do without telling them how to do it.

• The task must be specified by us...

• The task must be specified by us...

• The task must be specified by us...

• The task must be specified by us...

9 Tasks for Agents
A task specification is a function

utility: task of the agent is then to bring about states that maximise the

One possibility: associate utility with individual states — the

9.1 Utilities: Functions over States

which associates a real number with every environment state.

\[ u : \mathbb{E} \rightarrow \mathbb{R} \]

One possibility: associate utilities with individual states — the

9.1 Utilities: Functions over States
But what is the value of a run?...

Disadvantage: difficult to specify a long term view when assigning utilities to individual states.

- average?
- sum of utilities of states on run?
- maximum utility of state on run?
- minimum utility of state on run?

One possibility: a discount for states later on.
Another possibility: assign a utility not to individual states, but to runs themselves:

\[ \mathcal{E} \leftarrow \mathcal{R} n \]

Difficulties with utility-based approaches:
- We don’t think in terms of utilities!
- Where do the numbers come from?
- Where do formulatate tasks in these terms.

Other variations: incorporate probabilistic utilities of different states

Emerging.

Such an approach takes an inherently long term view.

An Introduction to Multiagent Systems
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 simulate 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

Utility function defined as follows:

\[
\frac{\text{number of holes that appeared in } r}{\text{number of holes filled in } r} = (\lambda)n
\]

- Hole appearance and disappearance of holes.
- Number of holes as possible.
- An agent can move in four directions, up, down, left, or right, and
- An agent can move in four directions, up, down, left, or right, and
- An agent can move in four directions, up, down, left, or right, and
- An agent can move in four directions, up, down, left, or right, and
9.3 Expected Utility & Optimal Agents

Then optimal agent $A^\text{opt}$ in an environment $En\nu$ is the one that maximizes expected utility:

$$A^\text{opt} = \arg\max_{A^\text{env}} \mathbb{E}_{I \in I\nu_{A^\text{env}}} \sum_{a \in A} \mathbb{E}(I \mid A^\text{env}, I, r)p(r)u(a, I, r).$$

Note:
- Agent $A^\text{env}$ is placed in environment $En\nu$.
- Write $p(r)\mathbb{E}(I \mid A^\text{env}, I, r)$ to denote probability that run $r$ occurs when $I \in I\nu_{A^\text{env}}$.
Some agents cannot be implemented on some computers (a function $\operatorname{Ag}$ may need more than available memory to implement). Write $\operatorname{Ag}^m$ to denote the agents that can be implemented on machine (computer) $m$. Write $\operatorname{Ag}^\exists$ for agents that exist but may need more than available memory. Some agents cannot be implemented on some computers.

9.4 Bounded Optimal Agents

\begin{align}
\forall \delta \in \mathcal{A} \quad & \max_{r \in R} \delta \left( \sum_{t=0}^{n} (1 - \delta(\tau)) \right) \\
& \leq \max_{r \in R} \delta \left( \sum_{t=0}^{n} (1 - \delta(\tau)) \right) \\
& = \text{bounded optimal } \operatorname{Ag}_{\text{Opt}}
\end{align}
As a special case of assigning utilities to histories, one can assign \( 0 \) (false) or \( 1 \) (true) to a run. If a run is assigned \( 1 \), then the agent succeeds on that run; otherwise it fails.

Denote predicate task specifications by \( \Phi \).

Call these **predicate task specifications**.

Thus \( \Phi : \{ 0, 1 \} \rightarrow \).

\( \Phi \) is a special case of assigning utilities to histories to histories to histories.
A task environment specifies:

- the properties of the system the agent will inhabit;
- the criteria by which an agent will be judged to have either failed or succeeded.

Let $\mathcal{E}$ be the set of all task environments.

$\begin{align*}
\{0, 1\} & \leftarrow \forall : \mathcal{E} \\
\end{align*}$

A task environment is a pair $(\mathcal{E}_n, \Phi)$ where $\mathcal{E}_n$ is an environment, and $\Phi$ is a predicate over runs.

9.6 Task Environments
\[ \forall \text{Env} = (\forall \text{Ag}., \text{Env})^{\text{Ag}.} \]

We then say that an agent Ag succeeds in task environment Env if \[ \langle \text{Env}, \text{Ag} \rangle \in \mathcal{L} \] and \[ \{ I = (r) \}_{r \in \mathcal{R}} \text{ (and } \forall \text{Env} = (\forall \text{Ag}., \text{Env})^{\text{Ag}.} \} \]

Write \( \forall \text{Ag}. \) to denote set of all runs of the agent Ag in environment Env that satisfy: \[ \mathcal{R} \]
Let \( P_{r Ag Env} \) denote the probability that run \( r \) occurs if agent \( Ag \) is placed in environment \( Env \).

\[
\left( \bigwedge_{r \in \mathcal{R}(Ag, Env)} \right) p_{r Ag Env} \subseteq \left( \bigwedge_{r \in \mathcal{R}(Ag, Env)} p_r \right)
\]

Then the probability \( P(\bigwedge_{r \in \mathcal{R}(Ag, Env)} p_r) \) that \( \Phi \) is satisfied by \( Ag \) in \( Env \) is placed in environment \( Env \).

Let \( P(\bigwedge_{r \in \mathcal{R}(Ag, Env)} p_r) \) denote the probability that run \( r \) occurs if agent \( Ag \).

The Probability of Success

An Introduction to Multiagent Systems

Lecture 2
Two most common types of tasks are achievement tasks and maintenance tasks:

1. Achievement tasks are those of the form “achieve state of affairs $\phi$.”

2. Maintenance tasks are those of the form “maintain state of affairs $\psi$.”
An achievement task is specified by a set of "good" or "goal" states: $G \subseteq E$. The agent succeeds if it is guaranteed to bring about at least one of these states (we do not care which one — they are all considered equally good).

A maintenance goal is specified by a set of "bad" states: $B \subseteq E$. The agent succeeds in a particular environment if it manages to avoid all states in $B$ — it never performs actions which result in any state in $B$ occurring.
Agentsynthesis is automatic programming: goal is to have a program that will take a task environment, and from this task environment automatically generate an agent that succeeds in this environment.

**Synthesis algorithm is:**

Think of \( \top \) as being like \( \texttt{null} \) in JAVA.

\[
(\{\top\} \cup \mathcal{A}) \leftarrow \exists \mathcal{L} : \text{true}
\]

This environment is automatically generated from the task environment and from this task environment, an agent that succeeds in this environment will be generated.

A **agent synthesis** is automatic programming.
A synthesis algorithm $\text{syn}$ is sound if it satisfies the following condition:

\[ \text{Synthesis algorithm } \text{syn} \text{ is sound if it satisfies the following condition: } \]

\[ \forall \gamma \in \forall \delta \exists \gamma \forall \delta (\text{Env}_{} \land \text{Ag}_{} \Rightarrow \text{Env}_{} \land \text{syn} \)