MULTIAGENT SYSTEMS 2E – CHAPTER 1
INTELLIGENT AGENTS

Michael Wooldridge

Department of Computer Science

University of Oxford, UK

mjw@cs.ox.ac.uk

Defining Agency

- What are the characteristics of an agent?
- This is a question that gets a lot of discussion! (Compare *object*.)
- For our purposes, we find it useful to introduce two increasingly strong notions of agency:
  - *weak* agency;
    (primarily the *software agents* community)
  - *strong* agency;
    (primarily AI).

A Weak Notion of Agency

An agent is a hardware or (more usually) software-based computer system that enjoys the following properties:

- *autonomy*

  agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;

- *social ability*

  agents interact with other agents (and possibly humans) via some kind of *agent communication language*;

- *reactivity*

  agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;

- *pro-activeness*

  agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by *taking the initiative*.

- (A simple way of conceptualising an agent is as a kind of UNIX-like software daemon.)
- If you take any of these attributes away, then you end up with software you already have. . .
- Think of (weak) agents as human-like 'assistants' or 'drones' that are limited in their abilities:
  - you can give them tasks to do, and they can go away and cooperate with other agents to achieve these tasks;
  - also, they are capable of taking the initiative in a limited way, like a human secretary would.
- The weak notion of agency *buys us something*: a useful computational metaphor and abstraction tool.

A Strong Notion of Agency

- For some researchers — particularly those in AI — the term ‘agent’ has a stronger meaning.
- These researchers generally mean an agent to be a computer system that, in addition to having the properties identified above, is either conceptualised or implemented using concepts usually applied to people:
  - mentalistic notions (belief, desire, obligation, choice, . . . ;
  - rationality;
  - veracity;
  - adaptability/learning.

Micro *versus* Macro Issues

- In building an agent-based system, there are 2 sorts of issues to be addressed:
  - *micro* issues:

    how do we design and build an agent that is capable of acting autonomously, reactively, pro-actively in a time-constrained domain?
  - *macro* issues:

    how do we get a society of agents to cooperate effectively (i.e., maximising their *coherence* and *coordination*)?

Agent Architectures

- We want to build agents, that enjoy the properties of autonomy, reactiveness, pro-activeness, and social ability that we talked about earlier.
  - How do we do this?
  - What software/hardware structures are appropriate?
  - What is an appropriate separation of concerns?
- This is the area of *agent architectures*.

- Maes defines an agent architecture as:

  ‘[A] particular methodology for building [agents]. It specifies how … the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions … and future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology.’

- Kaelbling considers an agent architecture to be:

  ‘[A] specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a general methodology for designing particular modular decompositions for particular tasks.’

- Most of the rest of this presentation will be taken up with a discussion of the various kinds of agent architecture that have been developed (primarily in AI).

Reasoning Agents

- The classical approach to building agents is to view them as a particular type of knowledge-based system, and bring all the associated (discredited?!) methodologies of such systems to bear.
- This paradigm is known as *symbolic AI*.
- We define a deliberative agent or agent architecture to be one that:
  - contains an explicitly represented, symbolic model of the world;
  - makes decisions (for example about what actions to perform) via symbolic reasoning.

- The idea of deliberative agents is highly seductive:

  To get an agent to realise some theory of agency, simply give it representation of this theory in rules, frames, semantic nets or whatever symbolic knowledge representation scheme you fancy, and get it to shift some symbols...

- Now if if one aims to build an agent in this way, then there are at least two important problems to be solved:

  1. The transduction problem: that of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful.
  2. The representation/reasoning problem: that of how to symbolically represent information about complex real-world entities and processes, and how to get agents to reason with this information in time for the results to be useful.

- The former problem has led to work on vision, speech understanding, learning, etc.
- The latter has led to work on knowledge representation, automated reasoning, automatic planning, etc.
- Despite the immense volume of work that these problems have generated, most researchers would accept that neither is anywhere near solved.
- Even seemingly trivial problems, such as commonsense reasoning, have turned out to be extremely difficult

- The underlying problem lies with the complexity of symbol manipulation algorithms in general: many (most) search-based symbol manipulation algorithms of interest are *intractable*.
- Because of these problems, some researchers have looked to alternative techniques for building agents; we look at these later.
- First, we consider efforts made within the symbolic AI community to construct agents.

AGENT0 and PLACA

- Much of the interest in agents from the AI community has arisen from Shoham's notion of *agent oriented programming* (AOP)
- AOP a 'new programming paradigm, based on a societal view of computation'.
- AOP embodies an unashamedly *strong* notion of agency!
- The key idea that informs AOP is that of directly programming agents in terms of intentional notions like belief, commitment, and intention.
- The motivation behind such a proposal is that, as we humans use the intentional stance as an *abstraction* mechanism for representing the properties of complex systems.

  In the same way that we use the intentional stance to describe humans, it might be useful to use the intentional stance to program machines.

- Shoham suggested that a complete AOP system will have 3 components:
  - a logic for specifying agents and describing their mental states;
  - an interpreted programming language for programming agents;
  - an 'agentification' process, for converting 'neutral applications' (e.g., databases) into agents.

  Results only reported on first two components.

  Relationship between logic and programming language is *semantics*.
- We will skip over the logic(!), and consider the first AOP language, AGENT0.

- AGENT0 is implemented as an extension to LISP.
  Each agent in AGENT0 has 4 components:
  - a set of capabilities (things the agent can do);
  - a set of initial beliefs;
  - a set of initial commitments (things the agent will do); and
  - a set of *commitment rules*.
- The key component, which determines how the agent acts, is the commitment rule set.

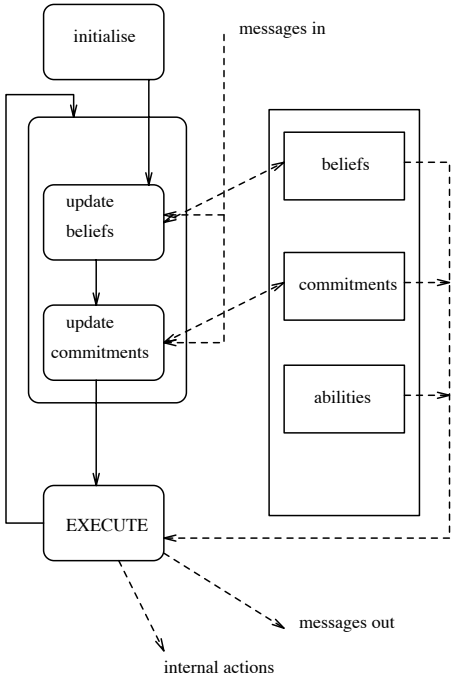
- Each commitment rule contains
  - a *message condition*;
  - a *mental condition*; and
  - an action.
- On each 'agent cycle' ...

  The message condition is matched against the messages the agent has received;

  The mental condition is matched against the beliefs of the agent.

  If the rule fires, then the agent becomes committed to the action (the action gets added to the agents commitment set).

- Actions may be
  - *private* — corresponding to an internally executed subroutine, or
  - *communicative*, i.e., sending messages.
- Messages are constrained to be one of three types:
  - 'requests' to commit to action;
  - 'unrequests' to refrain from actions;
  - 'informs', which pass on information.


initialise
messages in
update beliefs
update commitments
beliefs
commitments
abilities
EXECUTE
messages out
internal actions

- A commitment rule:

```
COMMIT(
  ( agent, REQUEST, DO(time, action)
  ), ;;; msg condition
  ( B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyaction)]
  ), ;;; mental condition
  self,
  DO(time, action)
)
```

- This rule may be paraphrased as follows:

  if I receive a message from *agent* which requests me to do *action* at *time*, and I believe that:

  – *agent* is currently a friend;

  – I can do the action;

  – at *time*, I am not committed to doing any other action,

  then commit to doing *action* at *time*.

- This trivial example gives a flavour of AGENT0 programs.

- AGENT0 provides support for multiple agents to cooperate and communicate, and provides basic provision for debugging...

- ... it is, however, a *prototype*, that was designed to illustrate some principles, rather than be a production language.

- A more refined implementation was developed by Thomas,
- Her Planning Communicating Agents (PLACA) language was intended to address one severe drawback to AGENT0: the inability of agents to plan, and communicate requests for action via high-level goals.
- Agents in PLACA are programmed in much the same way as in AGENT0, in terms of *mental change* rules.

- An example mental change rule:

```
(((self ?agent REQUEST (?t (xeroxed ?x)))
 (AND (CAN-ACHIEVE (?t xeroxed ?x)))
      (NOT (BEL (*now* shelving)))
      (NOT (BEL (*now* (vip ?agent))))
 ((ADOPT (INTEND (5pm (xeroxed ?x)))))
 ((?agent self INFORM
       (*now* (INTEND (5pm (xeroxed ?x)))))))
```

- Paraphrased:

  if someone asks you to xerox something, and you can, and you don't believe that they're a VIP, or that you're supposed to be shelving books, then

  – adopt the intention to xerox it by 5pm, and

  – inform them of your newly adopted intention.

Concurrent METATEM

- Concurrent METATEM is a multi-agent language in which each agent is programmed by giving it a *temporal logic* specification of the behaviour it should exhibit
- These specifications are executed directly in order to generate the behaviour of the agent.
- Temporal logic is classical logic augmented by *modal operators* for describing how the truth of propositions changes over time.

- For example...

$$\Box \text{important(agents)}$$

means "it is now, and will always be true that agents are important"

$$\Diamond \text{important()}$$

means "sometime in the future, will be important"

$$\blacklozenge \text{important(Prolog)}$$

means "sometime in the past it was true that Prolog was important"

$$(\neg \text{friends(us)}) \,\mathcal{U}\, \text{apologise(you)}$$

means "we are not friends until you apologise"

$$\bigcirc \text{apologise(you)}$$

means "tomorrow (in the next state), you apologise".

- is a framework for *directly executing* temporal logic specifications.
- The root of the concept is Gabbay's *separation theorem*:

  Any arbitrary temporal logic formula can be rewritten in a logically equivalent *past future* form.
- This *past future* form can be used as *execution rules*.
- A program is a set of such rules.
- Execution proceeds by a process of continually matching rules against a "history", and *firing* those rules whose antecedents are satisfied.
- The instantiated future-time consequents become *commitments* which must subsequently be satisfied.

- Execution is thus is process of iteratively generating a model for the formula made up of the program rules.
- The future-time parts of instantiated rules represent *constraints* on this model.
- An example program: the resource controller...

  $\forall$ x ◎ ask(x) $\Diamond$ give(x)

  $\forall$ x,y give(x) $\wedge$ give(y) (x=y)
- First rule ensure that an 'ask' is eventually followed by a 'give'.
- Second rule ensures that only one 'give' is ever performed at any one time.
- There are algorithms for executing programs that appear to give reasonable performance.
- There is also *separated normal form*.

- provides an operational framework through which societies of processes can operate and communicate.
- It is based on a new model for concurrency in executable logics: the notion of executing a logical specification to generate individual agent behaviour.
- A system contains a number of agents (objects), each object has 3 attributes:
  - a name;
  - an interface;
  - a program.

- An object's interface contains two sets:
  - environment predicates — these correspond to messages the object will accept;
  - component predicates — correspond to messages the object may send.
- For example, a 'stack' object's interface:

  stack(pop, push)[popped, stackfull]

  {pop, push} = environment preds

  {popped, stackfull} = component preds
- If an agent receives a message headed by an environment predicate, it accepts it.
- If an object satisfies a commitment corresponding to a component predicate, it broadcasts it.

- To illustrate the language in more detail, here are some example programs...
- Snow White has some sweets (resources), which she will give to the Dwarves (resource consumers).
- She will only give to one dwarf at a time.
- She will always eventually give to a dwarf that asks.
- Here is Snow White, written in :

Snow-White(ask)[give]:

$$\bigcirc\!\!\!\!\bullet\ \text{ask}(x) \quad \Diamond\ \text{give}(x)$$

$$\text{give}(x) \wedge \text{give}(y) \quad (x = y)$$

- The dwarf ‘eager’ asks for a sweet initially, and then whenever he has just received one, asks again.

  eager(give)[ask]:

	ask(eager)
◎ give(eager)	ask(eager)

- Some dwarves are even less polite: ‘greedy’ just asks every time.

  greedy(give)[ask]:

  □ ask(greedy)

- Fortunately, some have better manners; 'courteous' only asks when 'eager' and 'greedy' have eaten.

  courteous(give)[ask]:
  $((\neg \text{ask(courteous)}\ \mathcal{S}\ \text{give(eager)}) \wedge$
  $(\neg \text{ask(courteous)}\ \mathcal{S}\ \text{give(greedy)}))$
  ask(courteous)

- And finally, 'shy' will only ask for a sweet when no-one else has just asked.

  shy(give)[ask]:

	$\Diamond$ ask(shy)
◎ ask(x)	$\neg$ ask(shy)
◎ give(shy)	$\Diamond$ ask(shy)

Planning agents

- Since the early 1970s, the AI planning community has been closely concerned with the design of artificial agents.
- Planning is essentially automatic programming: the design of a course of action that will achieve some desired goal.
- Within the symbolic AI community, it has long been assumed that some form of AI planning system will be a central component of any artificial agent.
- Many planning algorithms have been proposed, and the theory of planning has been well-developed.

Reactive Architectures

- There are many unsolved (some would say insoluble) problems associated with symbolic AI.
- These problems have led some researchers to question the viability of the whole paradigm, and to the development of *reactive* architectures.
- Although united by a belief that the assumptions underpinning mainstream AI are in some sense wrong, reactive agent researchers use many different techniques.
- In this presentation, we start by reviewing the work of one of the most vocal critics of mainstream AI: Rodney Brooks.

Brooks — behaviour languages

- Brooks has put forward three theses:
  1. Intelligent behaviour can be generated *without* explicit representations of the kind that symbolic AI proposes.
  2. Intelligent behaviour can be generated *without* explicit abstract reasoning of the kind that symbolic AI proposes.
  3. Intelligence is an *emergent* property of certain complex systems.

- He identifies two key ideas that have informed his research:
  1. Situatedness and embodiment: 'Real' intelligence is situated in the world, not in disembodied systems such as theorem provers or expert systems.
  2. Intelligence and emergence: 'Intelligent' behaviour arises as a result of an agent's interaction with its environment. Also, intelligence is 'in the eye of the beholder'; it is not an innate, isolated property.

- To illustrate his ideas, Brooks built some based on his *subsumption architecture*
- A subsumption architecture is a hierarchy of task-accomplishing *behaviours*.
- Each behaviour is a rather simple rule-like structure.
- Each behaviour 'competes' with others to exercise control over the agent.
- Lower layers represent more primitive kinds of behaviour, (such as avoiding obstacles), and have precedence over layers further up the hierarchy.
- The resulting systems are, in terms of the amount of computation they do, *extremely* simple.

- Steels' Mars explorer system, using the subsumption architecture, achieves near-optimal cooperative performance in simulated 'rock gathering on Mars' domain:

  The objective is to explore a distant planet, and in particular, to collect sample of a precious rock. The location of the samples is not known in advance, but it is *known that they tend to be clustered.*

- Here are the behaviours for the Mars explorer (lower layers have priority):

1. *Obstacle avoidance*.

   (LOWEST LAYER — LEAST ABSTRACT).

2. *Path attraction*.

   If you see a heavily trodden path, follow it away from spaceship.

3. *Explore movement*.

   If I have no samples, move away from spaceship.

4. *Return movement*.

   If I have samples, and I can't see any denser concentration, then head towards spaceship.

5. *Random movement*.

   Move in a random fashion.

   (HIGHEST LAYER — MOST ABSTRACT).

Agre and Chapman — PENGI

- Agre & Chapman observed that most everyday activity is 'routine' in the sense that it requires little — if any — new abstract reasoning.
- Most tasks, once learned, can be accomplished in a routine way.
- Agre proposed that an efficient agent architecture could be based on the idea of 'running arguments'.
- The idea is that as most decisions are routine, they can be encoded into a low-level structure (such as a digital circuit), which only needs periodic updating.
- The idea is illustrated in PENGI: a simulated computer game, with the central character controlled using a scheme such as that outlined above.

Situated Automata

- A sophisticated approach is that of Rosenschein and Kaelbling
- In their *situated automata* paradigm, an agent is specified in a rule-like (declarative) language, and this specification is then compiled down to a digital machine, which satisfies the declarative specification.
- This digital machine can operate in a *provable time bound*.
- At the time of writing, the theoretical limitations of the approach are not well understood.
- The main difficulty appears to be that the more expressive the agent specification language, the harder it is to compile it.

  (There are some deep theoretical results which say that after a certain expressiveness, the compilation simply can't be done.)

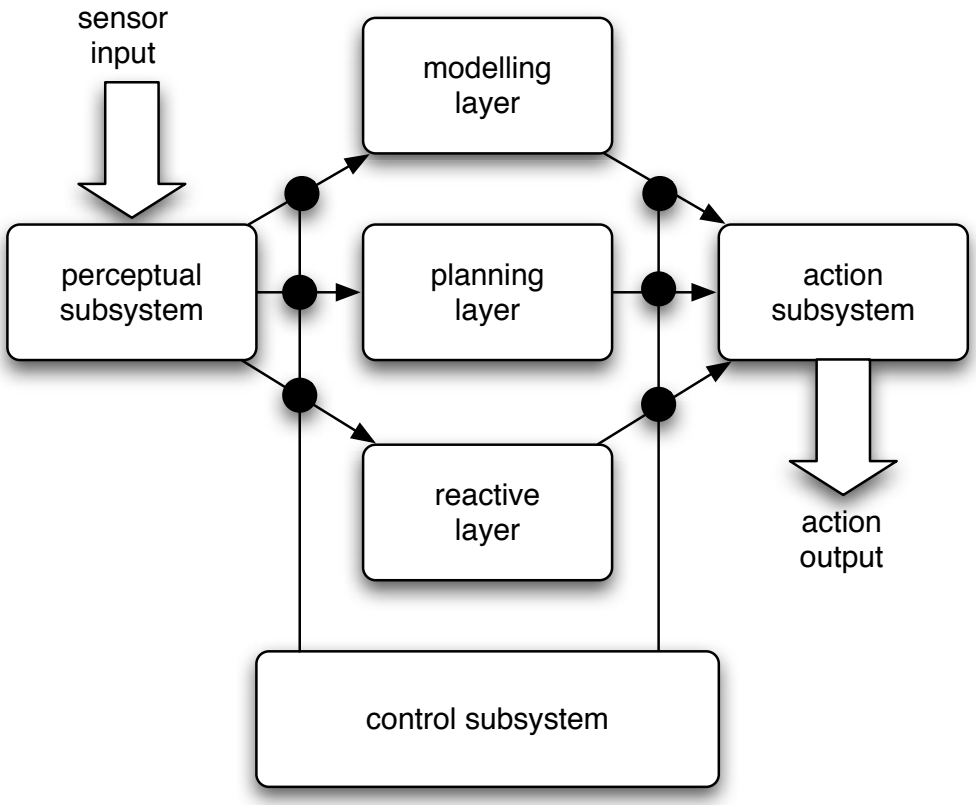
Hybrid Architectures

- Many researchers have argued that neither a completely deliberative nor completely reactive approach is suitable for building agents.
- They have suggested using *hybrid* systems, which attempt to marry classical and alternative approaches.
- An obvious approach is to build an agent out of two (or more) subsystems:
  - a *deliberative* one, containing a symbolic world model, which develops plans and makes decisions in the way proposed by symbolic AI; and
  - a *reactive* one, which is capable of reacting to events without complex reasoning.

- Often, the reactive component is given some kind of precedence over the deliberative one.
- This kind of structuring leads naturally to the idea of a *layered* architecture, of which TOURINGMACHINES and INTERRAP are examples.
- In such an architecture, an agent's control subsystems are arranged into a hierarchy, with higher layers dealing with information at increasing levels of abstraction.
- A key problem in such architectures is what kind control framework to embed the agent's subsystems in, to manage the interactions between the various layers.

Ferguson — TOURINGMACHINES

- The TOURINGMACHINES architecture consists of *perception* and *action* subsystems, which interface directly with the agent's environment, and three *control layers*, embedded in a *control framework*, which mediates between the layers.


sensor
input
modelling
layer
perceptual
subsystem
planning
layer
action
subsystem
reactive
layer
action
output
control subsystem

- The *reactive layer* is implemented as a set of situation-action rules, *à la* subsumption architecture.
- The *planning layer* constructs plans and selects actions to execute in order to achieve the agent's goals.
- The *modelling layer* contains symbolic representations of the 'cognitive state' of other entities in the agent's environment.
- The three layers communicate with each other and are embedded in a control framework, which use *control rules*.