Agent Architectures

- Agent architecture is a software design for an agent.
- Perception – State – Decision – Action
- Control flow between operations
- Operations on data structures
- Key data structures

Agent Architectures – Pattie Maes (1991)

- [A] particular methodology for building agents.
- Algorithms and techniques that support this methodology.
- The agent architecture encompasses techniques determining the actions the agent can take based on the current internal state of the agent and the current situation.
- How these modules should be made to interact.
- The construction of a set of component 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.
- Agent architectures typically specify how the agent can be decomposed into a particular methodology for building agents.
Chapter 3: An Introduction to Multiagent Systems

Types of Agents

- **Symbolic Reasoning Agents** (1956–present): The purest expression, propose that agents use explicit logical reasoning in order to decide what to do. Problems with symbolic reasoning led to a reaction against this—led to the reactive agents movement, 1985–present.


- **Hybrid Agents** (1990–present): Symbolic reasoning agents perform via symbolic reasoning. The classical approach to building agents is to view them as a particular type of knowledge-based system. This paradigm is known as symbolic AI. Problems with symbolic reasoning led to a reaction against this—led to the reactive agents movement, 1985–present.

- **Symbolic Reasoning Agents** (1995–present): Symbolic reasoning agents perform via symbolic reasoning. The classical approach to building agents is to view them as a particular type of knowledge-based system. This paradigm is known as symbolic AI. Problems with symbolic reasoning led to a reaction against this—led to the reactive agents movement, 1985–present.

- **Representing the Environment Symbolically**: Representing the environment symbolically.

- **The Transduction Problem**: The problem of translating the real world into an accurate, symbolic description, in time for that description to be useful.

- **Types of Agents**: Symbolic reasoning agents perform via symbolic reasoning. The classical approach to building agents is to view them as a particular type of knowledge-based system. This paradigm is known as symbolic AI. Problems with symbolic reasoning led to a reaction against this—led to the reactive agents movement, 1985–present.
Therepresentation/reasoningproblem 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.

Problems with Symbolic Approaches

• Because of these problems, some researchers have looked to alternative techniques for building agents.
• Many researchers accept that neither problem is resolved.

Deductive Reasoning Agents

• Use logic to encode a theory defining the best action to perform in any given situation.
• Use logical entailment to encode a theory defining the best action for a theory defining a set of rules.

let:

\[ \Delta \vdash \rho \phi \] mean that \( \phi \) can be proved from \( \Delta \) using \( \rho \).

\( \Delta \) be the set of actions the agent can perform.

\( \rho \) be the set of actions the agent can perform.

\( \Delta \) be a logical database that describes the current state of the world.

\( \phi \) be a logical database that describes the current state of the world.

\( \Delta \) ⊢ \( \rho \phi \) mean that \( \phi \) can be proved from \( \Delta \) using \( \rho \).

\( \Delta \) be the set of actions the agent can perform.

\( \rho \) be the best action defining the set of rules.

end-for

end-for

return null

/* no action found */

foreach \( \alpha \in \Delta \) do

if \( \Delta \vdash \rho \phi \) then return \( \alpha \)

end-for

foreach \( \alpha \in \Delta \) do

if \( \Delta \nvdash \rho \neg \phi \) then return \( \alpha \)

end-for

return null

/* no action found */

Action Selection via Theorem Proving

Automated Planning

... Knowledge representation, automated reasoning, and planning.

... Knowledge representation, automated reasoning, and planning.

The Representation/Reasoning Problem
An Example: The Vacuum World

- Goal is for the robot to clear up all dirt.

- Use 3 domain predicates in this exercise:
  - $\text{Dirt}(x, y)$
  - $\text{Facing}(d, x, y)$
  - $\text{Agent}(x, y)$

- Possible actions:
  - $\text{Do(front)}$
  - $\text{Do(right)}$
  - $\text{Do(left)}$

- Rules for determining what to do:
  - In $(0, 0)$ and facing north, and no dirt:
    - Do(front)
  - In $(0, 1)$ and facing north, and no dirt:
    - Do(front)
  - In $(0, 2)$ and facing north, and no dirt:
    - Do(right)
  - In $(0, 2)$ and facing east:
    - Do(front)

- Decision making assumes a static environment.
- Decision making via theorem proving is complex (maybe even undecidable).
- Decision making via planning is calculable rationally.

Problems

- How to convert video camera input to $\text{Dirt}(0, 1)$?
- Decision making assumes a static environment.
- Calculative rationality.
- Calculative rationality.
- Calculative rationality.

Using these rules (+ other obvious ones), starting at $(0, 0)$ the robot will clear up dirt.

- And so on:
  - $\text{In}(0, 0, z) \land \text{Facing}(east) \rightarrow \text{Do(front)}$
  - $\text{In}(0, 0, z) \land \text{Facing}(north) \land \text{Dirt}(0, 0, z) \rightarrow \text{Do(left)}$
  - $\text{In}(0, 0, z) \land \text{Facing}(north) \land \text{Dirt}(0, 1, z) \rightarrow \text{Do(front)}$
  - $\text{In}(0, 0, z) \land \text{Facing}(north) \land \text{Dirt}(0, 2, z) \rightarrow \text{Do(right)}$

- ... and so on!
Approaches to Overcoming these Problems

- weakenthelogic;
- usesymbolic,non-logicalrepresentations;
- shifttheemphasisofreasoningfromrun timeto
  design time.
- use symbolic, non-logical representations;
- weaken the logic;

Agent0

Eachagent in Agent0 has 4 components:

- a set of commitment rules;
- a set of initial commitments (things the agent will do);
- a set of initial beliefs;
- a set of capabilities (things the agent can do).

The key component, which determines how the agent acts, is the commitment rule set.

Commitment Rules

- a message condition;
- a mental condition;
- an action.

Eachcommitment rule contains:

The key component, which determines how the agent acts, is the commitment rule set.

AGENT0 and PLACA

Yoav Shoham introduced "agent-oriented programming" in 1990:

"new programming paradigm, based on a societal view of computation.,"

You introduced "agent-oriented programming" in 1990:

"new programming paradigm, based on a societal view of computation.,"

- a set of beliefs (things the agent believes);
- a set of intentions (things the agent intends to do);
- a set of capabilities (things the agent can do).
Chapter 3: An Introduction to Multiagent Systems

Agent Decision Cycle

• On each decision cycle...

The message condition is matched against the beliefs of the agent the message was received from.

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 (the action gets added to the agent's commitment set).

The action may be an internally executed computation or an externally executed computation.

Messages are constrained to be one of three types:

• Sending messages.

• Communicative messages.

• Private messages.

Messages which pass on information.

“Requests” to return from actions;

“Requests” to commit to actions;

“Unrequests” to return from actions;

“Unrequests” to commit to actions;

“Requests” to return from actions.

A commitment rule:

\[
\text{COMMIT}\left(\text{agent, REQUEST, DO}(\text{time, action})\right), \quad \text{msgcondition}(\text{B, [now, Friendagent] AND CAN(self, action) AND NOT[time, CMT(self, any action)]}, \quad \text{mentalcondition}(\text{self, DO}(\text{time, action}))
\]

On each decision cycle...

EXECUTE

update beliefs

update commitments

messages in

messages out

internal actions

update beliefs

update commitments

messages in

messages out

internal actions

update beliefs

update commitments

messages in

messages out

internal actions
• This rule may be paraphrased as follows:

  - if I receive a message from an agent which requests me to do action at time, and I believe that:
    - agent is currently a friend;
    - at time, I am not committed to doing any other action,
  
  then commit to doing action at time.

---

• A more refined implementation was developed by Thomas, for her 1993 doctoral thesis.

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.
A MetateM program is a collection of rules. The instantiated future-time consequents become commitments which must subsequently be satisfied.

• A Concurrent MetateM system contains a number of agents (objects); each object has 3 attributes:
  - a name;
  - an interface;
  - a MetateM program.

A MetateM program is a collection of rules whose antecedents are satisfied. Execution proceeds by a process of continually matching rules against a "history" and firing those rules whose antecedents are satisfied.

For example...

- An example MetateM program: the resource controller...
  - ask(x) ⇒ ♦ give(x)
  - give(x) ∧ give(y) ⇒ (x=y)
  - ask(x) ⇒ ¬ give(x) 

- First rule ensures that an "ask" is eventually followed by a "give":
  - Second rule ensures that only one "give" is ever performed at any one time.
  - Example MetateM program:

For example...

- An introductory MetateM program:
  - important(agents) means "it is now, and will always be true that agents are important"
  - important(ConcurrentMetateM) means "sometime in the future, ConcurrentMetateM will be important"
  - important(ConcurrentMetateM) means "it is now, and will always be true that ConcurrentMetateM are important"
  - important(agents) means "we are not friends until you apologize"

• MetateM program:

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
An agent's interface contains two sets:

- messages the agent will accept;
- messages the agent may send.

For example, a stack's objects' interface:

- messages the agent may send
- messages the agent will accept

An agent's interface contains two sets:

- messagestheagentwillaccept;
- messagestheagentmaysend.

Snow White & The Dwarves

To illustrate the language Concurrent MetaM, here are some example programs...

Snow White has some sweets (resources), which she will always eventually give to a dwarf that asks. She will only give to one dwarf at a time. Whenever he has just received one, asks again.

Every time.

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

The dwarfs 'eager' asks for a sweet initially and then whenever he has just received one, asks again.

To illustrate the language Concurrent MetaM:

For example, a stack's objects' interface:

- messages the agent may send
- messages the agent will accept

An agent's interface contains two sets:

- messages the agent will accept;
- messages the agent may send.

For example, a stack's objects' interface:

- messages the agent may send
- messages the agent will accept

To illustrate the language Concurrent MetaM, here are some example programs...

Snow White has some sweets (resources), which she will always eventually give to a dwarf that asks. She will only give to one dwarf at a time. Whenever he has just received one, asks again.

Every time.

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

The dwarfs 'eager' asks for a sweet initially and then whenever he has just received one, asks again.

To illustrate the language Concurrent MetaM:

For example, a stack's objects' interface:

- messages the agent may send
- messages the agent will accept

An agent's interface contains two sets:

- messages the agent will accept;
- messages the agent may send.

For example, a stack's objects' interface:

- messages the agent may send
- messages the agent will accept

To illustrate the language Concurrent MetaM, here are some example programs...
Fortunately, some have better manners: courteous. And finally, 'shy' will only ask for a sweet when no-one else has just asked.

(shy(give)]ask:

\begin{align*}
\text{ask}(\text{shy}) & \iff \Box \text{give}(\text{shy}) \\
\text{ask}(x) & \iff \Box \text{give}(\text{shy}) & \setminus \text{ask}(\text{shy}) \\
\text{shy} & \iff \Box \text{give}(\text{shy})
\end{align*}

\text{else has just asked.}