What is an Agent?

Wide range of behavior and functionality in computing

- Active computational entity
  - With a persistent identity
  - Able to carry out a long-lived conversation
- Perceives, reasons about, and initiates activities in its environment
  - Deals with services
- Communicates (with other agents)
  - Loosely coupled
- Adaptive

Agents and Multiagent Systems for Services

Business partners are supported by agents

- Unlike objects, agents
  - Are proactive and autonomous—can say No!
  - Support loose coupling
- In addition, agents may
  - Cooperate or compete
  - Model users, themselves, and others
  - Dynamically use and reconcile ontologies

Modeling Agents: Artificial Intelligence

Emphasize mental (folk psychology) concepts to achieve simplicity of description

- Beliefs: agent's representation of the world
- Knowledge: (usually) true beliefs
- Desires: preferred states of the world
- Goals: consistent desires
- Intentions: goals adopted for action
  - Resources allocated
  - Sometimes incorporate persistence

Modeling Agents: Multiagent Systems

Emphasize interaction and autonomy and, hence, communication)

- Social: about collections of agents
- Organizational: about teams and groups
- Legal: about contracts and compliance
- Ethical: about right and wrong actions

Mapping Service-Oriented Computing to Agents

Agents capture the constraints of an open system

- Autonomy $\Rightarrow$ ability to enter into and enact contracts
  - Counterbalanced by establishing compliance
  - How can we check or enforce compliance?
- Heterogeneity $\Rightarrow$ ontologies
- Loose coupling $\Rightarrow$ communication
- Trustworthiness $\Rightarrow$ contracts, ethics, learning, incentives
- Dynamism $\Rightarrow$ break and form relationships via combinations of the above

Two Main Ways to Apply Agents

Agent-Oriented Software Engineering (AOSE)

- As modeling constructs
  - Standing in for stakeholders
  - To help in capturing their requirements as goals
- As runtime constructs, each
  - Representing a stakeholder
  - Acting on its behalf, reflecting its autonomous decision making to others

Economic Rationality

Applies to business services

- Three elements: an agent's
  - Performance measure (for itself), e.g., expected utility
  - Prior knowledge and current (ongoing) perceptions
  - Available actions
- Ideally, for each possible percept sequence, a rational agent
  - Acts to maximize its expected utility
  - On the basis of its knowledge and evidence from the percept sequence

Logic-Based Agents

Logical reasoning being a form of rationality

- An agent is a knowledge-based system
  - Represents a symbolic (as opposed to neural) model of the world
  - Declarative, hence, inspectable
  - Reasons symbolically via logical deduction
- Challenges:
  - Representing information symbolically
    - Easier in information environments than in general
  - Maintaining an adequate model of the world

Cognitive Architecture for an Agent

- Sensors and effectors map to services
  Communication infrastructure is messaging middleware

Exercise

Create an instance of the preceding diagram where the two agents are Amazon and a manufacturer

- When is it beneficial to employ agents in this setting?
- What is an illustration of loose coupling in this setting?

A Reactive Agent

The Sense-Decide-Act Loop

```
Environment e;
RuleSet r; //Could be the receive method of an actor
while (true) {
  state = senseEnvironment(e);
  a = chooseAction(state, r);
  e.applyAction(a);
}
```

Generic BDI (Belief-Desire-Intention) Architecture

Addresses how beliefs, desires and intentions are represented, updated, acted upon
Variant with just beliefs and goals is also prominent

```
Agent::run() {
Perception p;
p.run(); //start perception in own thread

while (true) {
  intention = getBestPlan();
  if (intention.execute()) // if achieved
    desires.remove(intention);
}

Perception::run(){
while (true) {
  a.beliefs.incorporateNewObservations(getInput(w));
  if (! a.currentPlanIsApplicable())
    a.stopCurrentPlan();
  sleep(someShortTime);
}
```

- Richer than sense-decide-act: decisions directly affect future decisions

Representing Services for Planning

IOPE (sometimes IOPR), goes beyond typical input-output signature

- Inputs: information the service requires
- Outputs: information the service produces
- Preconditions: constraints on the input
- Effects: effects on the environment
- Results (variant of effects): properties of the output

Composition as Planning

- Represent initial and goal states
- Represent each service as an action
  - Based on its IOPE specification
- A composed service: a plan that invokes constituent services
  - Inputs: outputs of previous services
  - Preconditions: true in initial state or made true by effects (results) of previous services
  - Effects not undone by subsequent services yield the goal state

Rules: Logical Representations

Marry declarative representation with computing

- Modular: easy to read and maintain
- Inspectable (by fact of being declarative): easy to understand
- Executable: no further translation needed
- Expressive: (commonly) Turing complete
  - Capture knowledge that would otherwise not be captured declaratively
  - Compare with relational calculus (classical SQL) or description logics (OWL)
- Declarative, although imperfectly so
  - Conflict handling is nontrivial and often ad hoc

Kinds of Rules

- ECA (Event-Condition-Action) or Reaction

```
on event
if condition
then perform action
```

- Derivation rules: special case of above, e.g., integrity constraints:

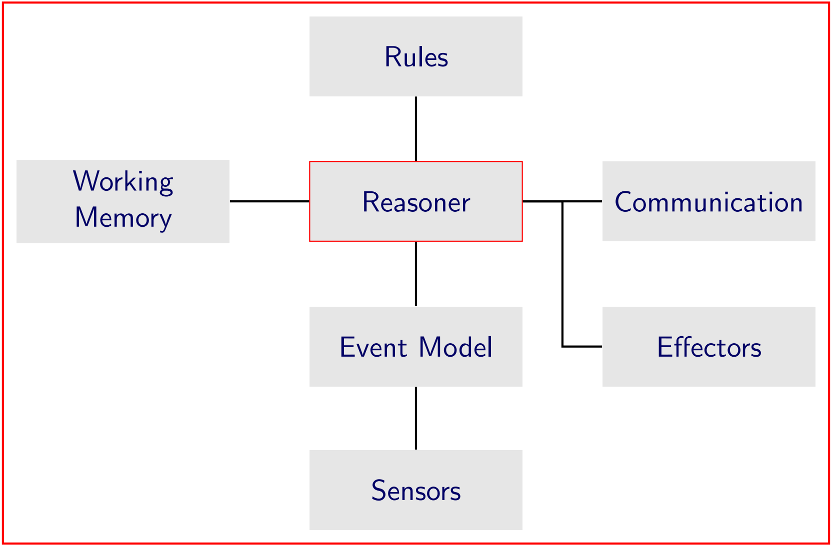
```
derive false
if error
```

- Inference rules

```
if antecedent
then consequent
```

  - Support multiple computational strategies
  - Forward chaining; backward chaining

Architecture of an ECA-Based Agent



Applying ECA Rules

- Capture protocols, enterprise policies, and heuristics as ECA rules
  - Examples?
- Combine with inference rules (to check if a condition holds)
- Modeling challenge
  - What is an event?
  - How to capture composite events by pushing event detection to lower layers

Example: ECA Rule

Identify predicates, variables, the do command, connectives

```
IF request (?x ?y ?z)    // event
   AND like (?x ?y)       // condition
THEN do(fulfill(?x ?z)) // action
```

- Watch out for relevant events
- If one occurs, check condition
- If condition holds, perform action

Example: Inference Rule

- Typical syntax indicating forward chaining

```
IF parent(?x ?y)
   AND parent (?y ?z)       // Antecedent
THEN grandparent (?x ?z) // Consequent
```

- Typical syntax indicating backward chaining

```
INFER grandparent (?x ?z) // Consequent
FROM parent(?x ?y)        // Antecedent
AND parent (?y ?z)
```

Example: Communication

Combining backward chaining and ECA

```
IF incoming-message(?x ?y ?z)
  AND policyA(?x ?y ?w)
  AND policyB(?x ?z ?v)
THEN send message(?x ?v ?w)
  AND assert internal-fact(?x ?v ?w)
```

- The policy stands for any internal decision making, usually defined as

```
INFER policyA(?x ?y ?w)
FROM ...

INFER policyB(?x ?z ?v)
FROM ...
```

Exercise: Communication

State the customer's rules to capture how it might interact with a merchant in a purchase protocol

- RFQ: request for quotes
- (Price) quote
- Accept or Reject
- Goods
- Payment
- Receipt

Typical Rule Syntax Limitations

- Antecedent may have conjunction or disjunction
- Antecedent may have generally not have negation (nontrivial)
- Consequent may have conjunction but not disjunction or negation—to avoid ambiguity of what to do
- Rules are not nested
- Generally no else clause

Applying Inference Rules

- Capture requirements naturally
- Elaboration tolerance requires defeasibility
  - Conclusions are not firm in the face of new information
  - Formulate general rules
  - Override rules to specialize them as needed
- Leads to logical nonmonotonicity
  - Easy enough operationally but difficult to characterize mathematically
  - Details get into logic programming with negation

Negation and Nonmonotonicity

- Strong negation, indicating falsity (i.e., nontruth)
  - Traditional, two-valued logic
  - Law of the excluded middle
- Weak negation, indicating absence of knowledge or absence of proof (depending upon the setting)
  - Goes beyond traditional, two-valued logic
  - A proposition and its strong negation may both be unknown
- Nonmonotonicity
  - Conclusions are retracted in light of additional information
  - Common in real-life reasoning
  - *Not* supported by traditional logic
  - Weak negation is an early approach to achieve nonmonotonicity

Conflicts and Priorities

- Rules can, and frequently do, conflict
  - An outcome of modular knowledge acquisition
  - Inadvertently enable two rules with contradictory conclusions
- Solution: expand rules to contain all applicable exception conditions
  - Unwieldy rules
  - Must redo each time new rules are stated
  - Can be impossible for users to understand $\Rightarrow$ a major motivation for rules in the first place
- Solution: assert which rule overrides another rule
  - *Specificity* based on predicates used: only generic basis for prioritizing one rule over another
    - Doesn't always apply
  - Rely on order in the rules program
    - Such an order may not exist
    - Nontrivial to maintain
  - Assert numeric (or categorical) weights on rules
    - Nontrivial to maintain
  - Assert rankings between rules
    - Nontrivial to maintain

Variables in Rules

For safety, do not introduce variables in action or consequent

- ECA rules introduce variables in event and condition
  - Free variable in action indicates perform action for each binding
- Inference rules introduce variables in antecedent
  - Free variable in consequent means assert it for each binding

Agents Summary

- Agents match requirements of open environments
- Agents go beyond objects and procedural programming
- Agent abstractions help express requirements in a natural manner
- Cognitive constructs for agents can be powerful
- Rules provide a simple means to construct information agents