



EH2750 Computer Applications in Power Systems, Advanced Course.

Lecture 5

Professor Lars Nordström, Ph.D.
 Dept of Industrial Information & Control systems, KTH
larsn@ics.kth.se



Acknowledgement

- These slides are based largely on a set of slides provided by:

*Professor Rosenschein of the Hebrew University
 Jerusalem, Israel*
 and
Dr. Georg Groh, TU-München, Germany.

- Available at the Student companion site of the Introduction to Multi Agent Systems book



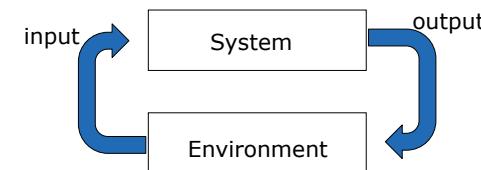
Outline of the Lecture

- Repeating where we are right now
 - Intelligent Agents of various types
 - How to make agents think and plan
- Constraint Satisfaction Problems
 - A variant of planning problems (still in one agent)
- Multi-agent interactions
 - Some concepts for cooperation
- Agent Communication
 - Ontologies, XML, RDF and OWL



What is an Intelligent Agent?

- The main point about agents is they are *autonomous*: capable of acting independently, exhibiting control over their internal state
- Thus: *an intelligent agent is a computer system capable of flexible autonomous action in some environment in order to meet its design objectives*





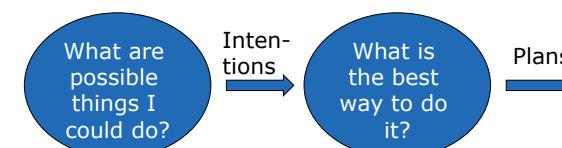
The discussion so far

- Chapter 2 describes the idea of agents that perform tasks in an environment and sets some definitions
- Chapters 3, 4, & 5 describe three different approaches to describing and developing the apparent Intelligence in the agents.
 - Chapter 3 – Deductive Reasoning Agents
 - Chapter 4 – Practical Reasoning Agents
 - Chapter 5 - Reactive (and Hybrid Agents)
- In the Excerpt from the AI book used in Lecture #4 we took a look at planning and searching
- Today we start looking at the Multi in Multi-agent systems



Practical Reasoning

- Human practical reasoning consists of two activities:
 - *deliberation*
deciding *what* state of affairs we want to achieve
 - *means-ends reasoning*
deciding *how* to achieve these states of affairs
- The outputs of deliberation are *intentions*



Practical Reasoning Agent

```

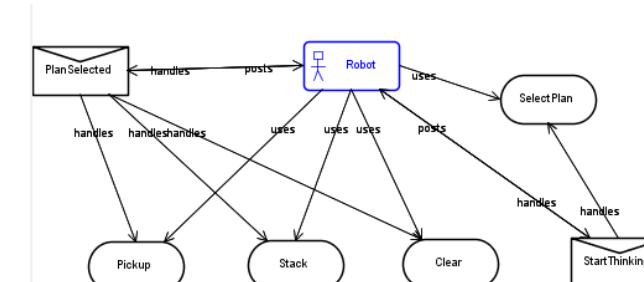
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  inputs: percept, a percept
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation

  state  $\leftarrow$  UPDATE-STATE(state, percept)
  if seq is empty then do
    goal  $\leftarrow$  FORMULATE-GOAL(state)
    problem  $\leftarrow$  FORMULATE-PROBLEM(state, goal)
    seq  $\leftarrow$  SEARCH(problem)
  action  $\leftarrow$  FIRST(seq)
  seq  $\leftarrow$  REST(seq)
  return action

```



How this can look in JACK





Outline of the Lecture

- Repeating where we are right now
 - Intelligent Agents of various types
 - How to make agents think and plan
- Constraint Satisfaction Problems
 - A variant of planning problems (still in one agent)
- Multi-agent interactions
 - Some concepts for cooperation
- Agent Communication
 - Ontologies, XML, RDF and OWL



Constraint Satisfaction problems

- Formally, a Constraint Satisfaction Problem (CSP) is
 - A set of variables x_1, x_2, \dots, x_n
 - All within a domain d_1, d_2, \dots, d_n
 - A set of constraints c_1, c_2, \dots, c_m
- A set of assigned values (to one or more of) the variable(s) is a state.
 - E.g.
 - $x_1 = 23, x_2 = 3$ is the state $\{23, 3\}$



Solution to a CSP

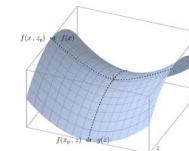
1. All variables have been assigned a value from their respective Domain – *complete assignment*
2. All constraints hold – *consistent assignment*



CSP – Different Characteristics

- Discrete variables with Finite Domains
 - Map colouring (typical example)
 - Circuit switching
- Infinite domains
 - E.g. Scheduling of flights
- Continuous variables
 - Linear constraints – optimisation problem....

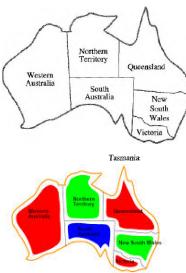
$$\begin{aligned} & \text{minimize}_{x} && f(x) \\ & \text{subject to} && g_i(x) \leq 0, \quad i = 1, \dots, m \\ & && h_i(x) = 0, \quad i = 1, \dots, p \end{aligned}$$





CSP in discrete finite domains

- Classic example – map coloring



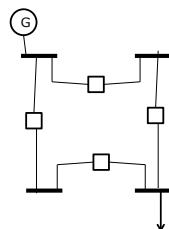
Solutions are assignments satisfying all constraints, e.g.
 $[WA=\text{red}, NT=\text{green}, Q=\text{red}, NSW=\text{green}, V=\text{red}, SA=\text{blue}, T=\text{green}]$

- Color the map of Australia
- Using the colors Red, Green, Blue
- No neighbours can have the same color
- CSP formulation
 - x_i = color of state i
 - $D = \{\text{Red, Green, Blue, Null}\}$
 - $x_i \neq x_j$ if $x_i = N(x_j)$



Or, if you wish

- CSP formulation for Switching problem

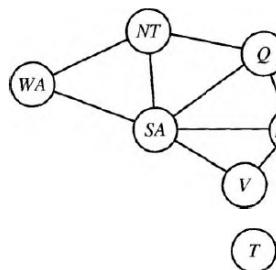


- Supply all load in the grid
- Switches can be on or off
- No loops
- CSP formulation
 - x_i = state of Switch i
 - $D = \{\text{breaking, conducting, Null}\}$
 - $c_1 = \text{not}(x_1 \wedge x_2 \wedge x_3 \wedge x_4)$



Back to Australia

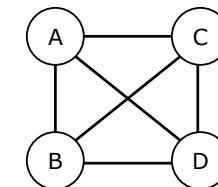
- Constraint Graph for Australia coloring problem



- It turns out, that the structure of the problem can be useful for finding the solution.
- This includes studying the types and degrees of constraints.



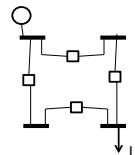
And the Switching problem





Types of constraints

- Unary constraints involve a single variable e.g.,
 - SA ≠ green
- Binary constraints involve pairs of variables
 - SA ≠ WA
- Higher order involves 3 or more variables
 - $\text{not}(x_1 \wedge x_2 \wedge x_3 \wedge x_4)$
- More advanced constraints
 - Use cost metrics for a variable
 - Powerflows for instance?
 - Constrained optimization problem



So, why all this?

- CSPs can be seen as search problems
 - States are defined by values assigned this far
 - Initial state: empty assignment {}
 - Successor function:
 - Assign value to a variable that is OK with constraints
 - Goal test: complete assignment with all constraints satisfied
- Note that every solution appears at depth n
 - use depth-first search



But, wait – don't be too fast

- What is the complexity of a completely naive solution?

$$O(n!d^n)$$

- Because for every variable you must test any color and then test the constraints and goal fulfilment.
- But that is stupid!



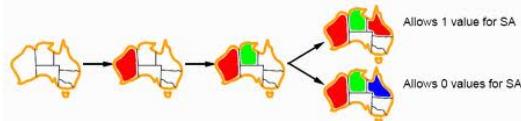
Commutativity

- The order in which assignments are made is not important.
- Consider only one variable at each node.
 - No point to worry about color of WA when you are selecting the color at SA
- Use Backtracking if searching fails.
 - Success function is:
 - Assign value to variable x_i from d_i
 - If not possible unless constraints are broken
 - Go back to x_{i-1} and assign alternate value from domain d_{i-1}



"Generic" Heuristics

- Based on our knowledge of the constraint graph we can choose which is the next node to assign a variable to.
- Minimum Remaining values (MRV)
 - Pick the Node with the least number of available values.
 - This avoids searching for solutions



Degree Heuristic

- But where to start?
 - Select the Node with the most constraints, highest degree* in constraint graph.



* Number of connecting edges in the Graph.



Backtracking Search

```

function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING( csp )
function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure

```

Figure 5.3 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. The functions *SELECT-UNASSIGNED-VARIABLE* and *ORDER-DOMAIN-VALUES* can be used to implement the general-purpose heuristics discussed in the text.



Things to take away

- Constraint Satisfaction Problems can be solved as searches
- Analysis of the problem structure can provide us with generic heuristics
- Planning with Backtracking is a key method for cooperative planning

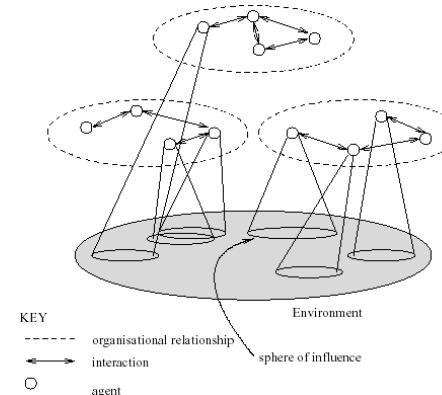


Outline of the Lecture

- Repeating where we are right now
 - Intelligent Agents of various types
 - How to make agents think and plan
- Constraint Satisfaction Problems
 - A variant of planning problems (still in one agent)
- Multi-agent interactions
 - Some concepts for cooperation
- Agent Communication
 - Ontologies, XML, RDF and OWL



Multi-agent Systems



Multi-agent Systems

Contains a number of agents...

- ...which interact through communication...
- ...are able to act in an environment...
- ...have different "spheres of influence" (which may coincide)...
- ...will be linked by other (organizational) relationships



Working Together

- Why and how do agents work together?
- Important to make a distinction between:
 - *benevolent agents*
 - *self-interested agents*



Benevolent Agents

- If we “own” the whole system, we can design agents to help each other whenever asked
- In this case, we can assume agents are *benevolent*: our best interest is their best interest
- Problem-solving in benevolent systems is *cooperative distributed problem solving* (CDPS)
- *Benevolence simplifies the system design task enormously!*



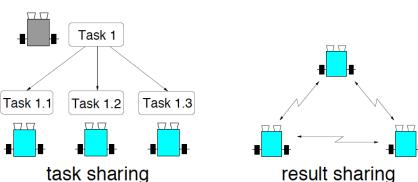
Self-Interested Agents

- If agents represent individuals or organizations, (the more general case), then we cannot make the benevolence assumption
- Agents will be assumed to act to further their own interests, possibly at expense of others
- Potential for *conflict*
- May complicate the design task enormously



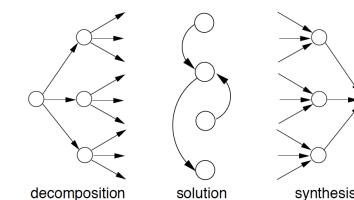
Benevolent Agents Task Sharing and Result Sharing

- Two main modes of cooperative problem solving:
 - *task sharing*: components of a task are distributed to component agents
 - *result sharing*: information (partial results, etc.) is distributed



Benevolent Agents Cooperative Distributed Problem Solving

- CDPS is concerned with investigation of:
 - Problem subdivision
 - Sub-Problem distribution
 - Result synthesis
 - Optimization of problem solver coherence
 - Optimization of problem solver coordination





Benevolent Agents Coherence

Coherence: Refers to “how well the MAS behaves as a unit along some dimension of evaluation”. Coherence may be measured in terms of

- Solution quality
- resource usage
- conceptual clarity of operation
- performance degradation if unexpected failure occurs



Benevolent Agents Coordination

• **Coordination:** “The degree...to which [the agents] can avoid ‘extraneous’ activity [such as] ...synchronizing and aligning their activities”

→ Poor coordination if

- Agents clobber each other’s sub-goals
- Lots of communication (no mutual predictability (e.g. by expressive models of each other))
- Destructive interference if conflict



Self-Interested Agents



Utilities and Preferences

- Assume we have just two agents: $Ag = \{i, j\}$
- Agents are assumed to be **self-interested**: they *have preferences over how the environment is*
- Assume $\Omega = \{\omega_1, \omega_2, \dots\}$ is the set of “outcomes” that agents have preferences over
- We capture preferences by *utility functions*:

$$u_i = \Omega \rightarrow \mathbf{R}$$

$$u_j = \Omega \rightarrow \mathbf{R}$$

- Utility functions lead to *preference orderings* over outcomes:

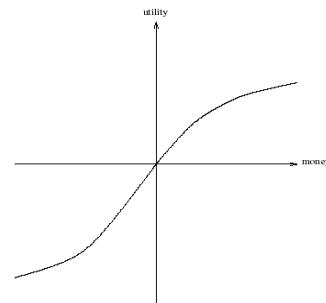
$$\omega \succeq_i \omega' \text{ means } u_i(\omega) \geq u_i(\omega')$$

$$\omega \succ_i \omega' \text{ means } u_i(\omega) > u_i(\omega')$$



What is Utility?

- Utility is *not* money (but it is a useful analogy)
- Typical relationship between utility & money:



Multiagent Encounters

- We need a model of the environment in which these agents will act...
 - agents simultaneously choose an action to perform, and as a result of the actions they select, an outcome in Ω will result
 - the *actual* outcome depends on the *combination* of actions
 - assume each agent has just two possible actions that it can perform, C ("cooperate") and D ("defect")
- Environment behavior given by *state transformer function*:

$$\tau : \underbrace{\mathcal{A}_c}_{\text{agent } i \text{ 's action}} \times \underbrace{\mathcal{A}_c}_{\text{agent } j \text{ 's action}} \rightarrow \Omega$$



Multiagent Encounters

- Here is a state transformer function:
 $\tau(D, D) = \omega_1 \quad \tau(D, C) = \omega_2 \quad \tau(C, D) = \omega_3 \quad \tau(C, C) = \omega_4$
 (This environment is sensitive to actions of both agents.)
- Here is another:
 $\tau(D, D) = \omega_1 \quad \tau(D, C) = \omega_1 \quad \tau(C, D) = \omega_1 \quad \tau(C, C) = \omega_1$
 (Neither agent has any influence in this environment.)
- And here is another:
 $\tau(D, D) = \omega_1 \quad \tau(D, C) = \omega_2 \quad \tau(C, D) = \omega_1 \quad \tau(C, C) = \omega_2$
 (This environment is controlled by j .)



Rational Action

- Suppose we have the case where *both* agents can influence the outcome, and they have utility functions as follows: $u_i(\omega_1) = 1 \quad u_i(\omega_2) = 1 \quad u_i(\omega_3) = 4 \quad u_i(\omega_4) = 4$
 $u_j(\omega_1) = 1 \quad u_j(\omega_2) = 4 \quad u_j(\omega_3) = 1 \quad u_j(\omega_4) = 4$
- With a bit of abuse of notation:
 $u_i(D, D) = 1 \quad u_i(D, C) = 1 \quad u_i(C, D) = 4 \quad u_i(C, C) = 4$
 $u_j(D, D) = 1 \quad u_j(D, C) = 4 \quad u_j(C, D) = 1 \quad u_j(C, C) = 4$
- Then agent i 's preferences are:
 $C, C \succeq_i C, D \quad \succ_i \quad D, C \succeq_i D, D$
- " C " is the *rational choice* for i .
 (Because i prefers all outcomes that arise through C over all outcomes that arise through D .)



Payoff Matrices

- We can characterize the previous scenario in a *payoff matrix*:

		<i>i</i>	
		defect	coop
<i>j</i>	defect	1	4
	coop	1	4

- Agent *i* is the *column player*
- Agent *j* is the *row player*



Dominant Strategies

- Given any particular strategy (either *C* or *D*) of agent *i*, there will be a number of possible outcomes
- We say s_1 *dominates* s_2 if every outcome possible by *i* playing s_1 is preferred over every outcome possible by *i* playing s_2
- A rational agent will never play a dominated strategy
- So in deciding what to do, we can *delete dominated strategies*
- Unfortunately, there isn't always a unique undominated strategy



Nash Equilibrium

- In general, we will say that two strategies s_1 and s_2 are in Nash equilibrium if:
 - under the assumption that agent *i* plays s_1 , agent *j* can do no better than play s_2 ; and
 - under the assumption that agent *j* plays s_2 , agent *i* can do no better than play s_1 .
- *Neither agent has any incentive to deviate from a Nash equilibrium*
- Unfortunately:
 - Not every interaction scenario has a Nash equilibrium*
 - Some interaction scenarios have more than one Nash equilibrium*



Competitive and Zero-Sum Interactions

- Where preferences of agents are diametrically opposed we have *strictly competitive* scenarios
- Zero-sum encounters are those where utilities sum to zero:
$$u_i(\omega) + u_j(\omega) = 0 \quad \text{for all } \omega \text{ in } \Omega$$
- Zero sum implies strictly competitive
- Zero sum encounters in real life are very rare ... but people tend to act in many scenarios as if they were zero sum



The Prisoner's Dilemma

- Two men are collectively charged with a crime and held in separate cells, with no way of meeting or communicating. They are told that:
 - if one confesses and the other does not, the confessor will be freed, and the other will be jailed for three years
 - if both confess, then each will be jailed for two years
- Both prisoners know that if neither confesses, then they will each be jailed for one year



The Prisoner's Dilemma

- Payoff matrix for prisoner's dilemma:

		<i>i</i>	
		defect	coop
<i>j</i>	defect	2	1
	coop	4	3

- Top left: If both defect, then both get punishment for mutual defection
- Top right: If *i* cooperates and *j* defects, *i* gets sucker's payoff of 1, while *j* gets 4
- Bottom left: If *j* cooperates and *i* defects, *j* gets sucker's payoff of 1, while *i* gets 4
- Bottom right: Reward for mutual cooperation



The Prisoner's Dilemma

- The *individual rational* action is **defect**. This guarantees a payoff of no worse than 2, whereas cooperating guarantees a payoff of at most 1.
- So defection is the best response to all possible strategies: both agents defect, and get payoff = 2
- But *intuition* says this is *not* the best outcome: Surely they should both cooperate and each get payoff of 3!



The Prisoner's Dilemma

- This apparent paradox is *the fundamental problem of multi-agent interactions*. It appears to imply that *cooperation will not occur in societies of self-interested agents*.
- Real world examples:
 - nuclear arms reduction ("why don't I keep mine. . . ")
 - free rider systems — public transport;
- The prisoner's dilemma is *present everywhere*.
- Can we recover cooperation?
 - Well, yes we can introduce auctions, negotiations and argumentation. More on this next lecture!



Outline of the Lecture

- Repeating where we are right now
 - Intelligent Agents of various types
 - How to make agents think and plan
- Constraint Satisfaction Problems
 - A variant of planning problems (still in one agent)
- Multi-agent interactions
 - Some concepts for cooperation
- Agent Communication
 - Ontologies, XML, RDF and OWL



Agent Communication

- The traditional computer sciences view on communication in concurrent systems is focused on solving synchronization of multiple processes.
- Example:
- Processes p_1 and p_2 ; shared variable v ;
 p_1 reads v ;
 p_2 reads v ;
 p_2 updates v ;
 p_1 updates v ;
 \rightarrow updates by p_2 are lost;



Agent Communication II

Object oriented view on communication: Object o_2 invokes method m on object o_1 : Java: **$o_1.m(arg)$**

- o_2 has control over invocation. o_1 must invoke m .

Agent view on communication: Agent a_2 asks (sends event in JACK) agent a_1 to perform action α . (a_2 makes a request).

- a_1 has control over whether it performs action α . Agents are autonomous.



Agent Communication III

- What agents can do:
 Perform communication acts
- Goal: Influence other agents:
 - To make them perform actions or
 - to make them believe something (change their belief)
- The receiving agent decides whether to perform action or believe proposition



Speech Acts

- Most treatments of communication in (multi-) agent systems borrow their inspiration from *speech act theory*
- Speech act theories are *pragmatic* theories of language, i.e., theories of language use: they attempt to account for how language is used by people every day to achieve their goals and intentions
- The origin of speech act theories are usually traced to Austin's 1962 book, *How to Do Things with Words*



Speech Acts in the agent community

- Based on the Speech Act theory, Agent Communication Languages have been developed.
- The two most known are
 - KQML - Knowledge Query Markup Language.
 - FIPA - ACL Agent Communication Language.
- These are not programming languages as such, but formalisations of communication acts that are useful to understand and specify agent interaction.



Speech Acts – some thoughts.

- Consider:
 - performative = request
content = "the door is closed"
speech act = "please close the door"
 - performative = inform
content = "the door is closed"
speech act = "the door is closed!"
 - performative = inquire
content = "the door is closed"
speech act = "is the door closed?"



Agent Communication Languages

- We now consider *agent communication languages* (ACLs) — standard formats for the exchange of messages
- An early example of an ACL is KQML, developed by the ARPA knowledge sharing initiative
KQML is comprised of two parts:
 - the knowledge query and manipulation language (KQML)
 - the knowledge interchange format (KIF)
- A later developed framework is the FIPA



KQML and KIF

- KQML is an 'outer' language, that defines various acceptable 'communicative verbs', or *performatives*
Example performatives:
 - ask-if ('is it true that. . .')
 - perform ('please perform the following action. . .')
 - tell ('it is true that. . .')
 - reply ('the answer is. . .')
- KIF is a language for expressing message *content*



FIPA

- More recently, the Foundation for Intelligent Physical Agents (FIPA) started work on a program of agent standards — the centerpiece is an ACL
- Basic structure is quite similar to KQML:
 - *performative*
20 performatives in FIPA
 - *housekeeping*
e.g., sender, etc.
 - *content*
the actual content of the message



FIPA, example of an performative

• Example:

```
(inform
  :sender
  :receiver
  :content
  :language
  :ontology
)
agent1
agent5
(price good200 150)
s1
hpl-auction
```

But this part then?



To communicate...

- ...the agents must understand each other
- To understand each other the agents must use common terms, an *Ontology* is a formal specification of such terms.



Specifications of Terms - XML

- A basic format for specifying information exchange is the XML (eXtended Markup Language)

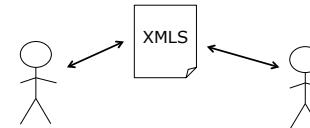
```
<?xml version="1.0" standalone="yes" ?>
<shop location="Birmingham" size="Large">
  <food>
    <Name>Apple</Name>
    <type>fruit</type>
    <cost>15</cost>
  </food>
  <food>
    <Name>Carrot</Name>
    <type>vegetable</type>
    <cost>10</cost>
  </food>
</shop>
```

- The structure of the information is decided by the author of the text file
- No rule checking is implemented in the format
- Data can be named with tags.
- The structure of the XML file is specified in an XML Schema (XMLS)
- By exchanging XMLS files, two agents can be made aware of possible terms.



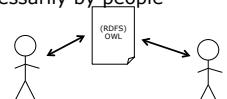
XML Schema

```
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema/XMLSchema"
  version="1.0">
  <xss:element name="EmployeeDetails">
    <xss:complexType>
      <xss:sequence>
        <xss:element name="MailAddressTo">
          <xss:complexType>
            <xss:sequence>
              <xss:element name="EmployeeName" type="xs:string"/>
              <xss:element name="Department" type="xs:string"/>
              <xss:element name="Job" type="xs:string"/>
              <xss:element name="Salary" type="xs:string"/>
            </xss:sequence>
            <xss:attribute name="EmployeeId" type="xs:string" use="required"/>
          </xss:complexType>
        </xss:element>
      </xss:sequence>
    </xss:complexType>
  </xss:element>
</xss:schema>
```



Specifications of Terms - RDF

- Resource Description Framework uses XML syntax but adds more rules to the terms.
 - XML is more flexible = Less interoperable
 - RDF is more structured)= More interoperable
- A framework (not a language) for describing resources
 - Providing a model for data
 - Syntax to allow exchange and use of information stored in various locations
 - The point is to facilitate reading and correct use of information by *computers*, not necessarily by *people*



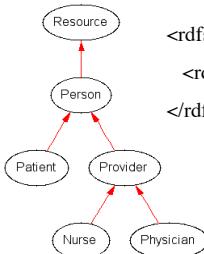
RDF Structure

- Described in RDF Schema (or now more popular OWL)
- Nodes are identified by URIs
 - E.g. <http://iec.ch/TC57/2001/CIM-schema-cim10#Wires>
- Elements in RDF files can be given more attributes
 - rdfs:Class
 - rdfs:Property
 - rdfs:subClassOf
 - rdf:type

```
<rdfs:Class rdf:ID="Switch">
  <rdfs:label>Switch</rdfs:label>
  <rdfs:subClassOf>
    rdfs:Resource
  </rdfs:subClassOf>
</rdfs:Class>
<rdfs:Class rdf:ID="Breaker">
  <rdfs:label>Breaker</rdfs:label>
  <rdfs:subClassOf>
    rdfs:Resource
  </rdfs:subClassOf>
</rdfs:Class>
<rdf:Property rdf:ID="Switch_NormalOpen">
  <rdfs:label>NormalOpen</rdfs:label>
  <rdfs:range>
    rdfs:Resource
  </rdfs:range>
  <rdfs:range>
    rdfs:Resource
  </rdfs:range>
</rdf:Property>
<rdf:Property rdf:ID="Breaker_AmpRating">
  <rdfs:label>AmpRating</rdfs:label>
  <rdfs:range>
    rdfs:Resource
  </rdfs:range>
  <rdfs:range>
    rdfs:Resource
  </rdfs:range>
</rdf:Property>
```



Simplified Schema, Healthcare example



```

<rdfs:Class rdf:ID="Provider">
  <rdfs:subClassOf rdf:resource="#Person"/>
</rdfs:Class>
  
```



RDF example

The **xmns:rdf** namespace, specifies that elements with the **rdf** prefix are from the namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns#".

The **xmns:cd** namespace, specifies that elements with the **cd** prefix are from the namespace "http://www.recsop.fake/cd#".

The **<rdf:Description>** element contains the description of the resource identified by the **rdf:about** attribute.

The elements: **<cd:artist>**, **<cd:country>**, **<cd:company>**, etc. are properties of the resource.

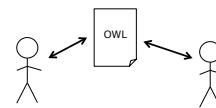
```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cd="http://www.recsop.fake/cd#">
  <rdf:Description
    rdf:about="http://www.recsop.fake/cd/Empire Burlesque">
      <cd:artist>Bob Dylan</cd:artist>
      <cd:country>USA</cd:country>
      <cd:company>Columbia</cd:company>
      <cd:price>10.90</cd:price>
      <cd:year>1985</cd:year>
    </rdf:Description>
    <rdf:Description
      rdf:about="http://www.recsop.fake/cd/Hide your heart">
        <cd:artist>Bonnie Tyler</cd:artist>
        <cd:country>UK</cd:country>
        <cd:company>CBS Records</cd:company>
        <cd:price>9.90</cd:price>
        <cd:year>1988</cd:year>
      </rdf:Description>
    .
    .
  </rdf:RDF>
  
```



Specification of Terms - OWL

- OWL Ontology Web Language
- Adds even more structure to the meta-data definitions
- Adds relation to Objects, so that Logic can be used to Infer facts about the data.



Outline of the Lecture

- Repeating where we are right now
 - Intelligent Agents of various types
 - How to make agents think and plan
- Constraint Satisfaction Problems
 - A variant of planning problems (still in one agent)
- Multi-agent interactions
 - Some concepts for cooperation
- Agent Communication
 - Ontologies, XML, RDF and OWL