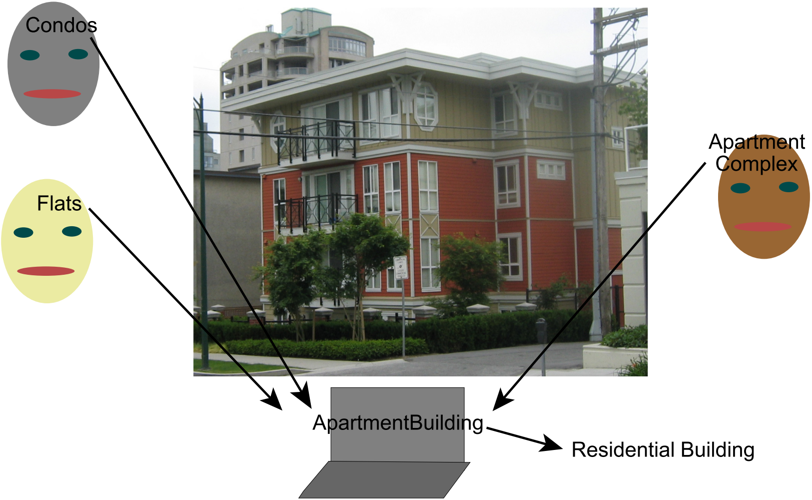
Knowledge Sharing

- A conceptualization is a map from the problem domain into the representation. A conceptualization specifies:
  - What sorts of individuals are being modeled
  - The vocabulary for specifying individuals, relations and properties
  - The meaning or intention of the vocabulary
- If more than one person is building a knowledge base, they must be able to share the conceptualization.
- An ontology is a specification of a conceptualization. An ontology specifies the meanings of the symbols in an information system.

Mapping from a conceptualization to a symbol



Semantic Web

- Ontologies are published on the web in machine readable form.
- Builders of knowledge bases or web sites adhere to and refer to a published ontology:
  - a symbol defined by an ontology means the same thing across web sites that obey the ontology.
  - if someone wants to refer to something not defined, they publish an ontology defining the terminology.
    Others adopt the terminology by referring to the new ontology.
    In this way, ontologies evolve.
  - Separately developed ontologies can have mappings between them published.

Challenges of building ontologies

- They can be huge: finding the appropriate terminology for a concept may be difficult.

Challenges of building ontologies

- They can be huge: finding the appropriate terminology for a concept may be difficult.
- How one divides the world can depend on the application. Different ontologies describe the world in different ways.
- People can fundamentally disagree about an appropriate structure.

Challenges of building ontologies

- They can be huge: finding the appropriate terminology for a concept may be difficult.
- How one divides the world can depend on the application. Different ontologies describe the world in different ways.
- People can fundamentally disagree about an appropriate structure.
- Different knowledge bases can use different ontologies.
- To allow KBs based on different ontologies to inter-operate, there must be mapping between ontologies.
- It has to be in user's interests to use an ontology.

Challenges of building ontologies

- They can be huge: finding the appropriate terminology for a concept may be difficult.
- How one divides the world can depend on the application. Different ontologies describe the world in different ways.
- People can fundamentally disagree about an appropriate structure.
- Different knowledge bases can use different ontologies.
- To allow KBs based on different ontologies to inter-operate, there must be mapping between ontologies.
- It has to be in user's interests to use an ontology.
- The computer doesn't understand the meaning of the symbols. The formalism can constrain the meaning, but can't define it.

Semantic Web Technologies

- XML the Extensible Markup Language provides generic syntax.
  $\langle tag \ldots / \rangle$ or
  $\langle tag \ldots \rangle \ldots \langle / tag \rangle$.
- URI a Uniform Resource Identifier is a name of an individual (resource). This name can be shared. Often in the form of a URL to ensure uniqueness.
- RDF the Resource Description Framework is a language of triples
- OWL the Web Ontology Language, defines some primitive properties that can be used to define terminology. (Doesn't define a syntax).

Main Components of an Ontology

- Individuals the things / objects in the world (not usually specified as part of the ontology)
- Classes sets of individuals
- Properties between individuals and their values

Individuals

- Individuals are things in the world that can be named. (Concrete, abstract, concepts, reified).
- Unique names assumption (UNA): different names refer to different individuals.
- The UNA is not an assumption we can universally make: "The Queen", "Elizabeth Windsor", etc.
- Without the determining equality, we can't count!
- In OWL we can specify:

  owl:SameIndividual(i_1, i_2)

  owl:DifferentIndividuals(i_1, i_3)

Classes

- A class is a set of individuals. E.g., house, building, officeBuilding
- One class can be a subclass of another

  owl:SubClassOf(*house*, *building*)

  owl:SubClassOf(*officeBuilding*, *building*)
- The most general class is owl:Thing.
- Classes can be declared to be the same or to be disjoint:

  owl:EquivalentClasses(*house*, *singleFamilyDwelling*)

  owl:DisjointClasses(*house*, *officeBuilding*)
- Different classes are not necessarily disjoint.
  E.g., a building can be both a commercial building and a residential building.

Properties

- A property is between an individual and a value.
- A property has a domain and a range.

  rdfs:domain(*livesIn*, *person*)

  rdfs:range(*livesIn*, *placeOfResidence*)

Properties

- A property is between an individual and a value.
- A property has a domain and a range.

  rdfs:domain(*livesIn*, *person*)

  rdfs:range(*livesIn*, *placeOfResidence*)
- An *ObjectProperty* is a property whose range is an individual.
- A *DatatypeProperty* is one whose range isn't an individual, e.g., is a number or string.

Properties

- A property is between an individual and a value.
- A property has a domain and a range.

  rdfs:domain(*livesIn*, *person*)

  rdfs:range(*livesIn*, *placeOfResidence*)
- An *ObjectProperty* is a property whose range is an individual.
- A *DatatypeProperty* is one whose range isn't an individual, e.g., is a number or string.
- There can also be property hierarchies:

  owl:subPropertyOf(*livesIn*, *enclosure*)

  owl:subPropertyOf(*principalResidence*, *livesIn*)

Properties (Cont.)

- One property can be inverse of another

  owl:InverseObjectProperties(*livesIn*, *hasResident*)

- Properties can be declared to be transitive, symmetric, functional, or inverse-functional.

Properties (Cont.)

- One property can be inverse of another

  owl:InverseObjectProperties(*livesIn*, *hasResident*)

- Properties can be declared to be transitive, symmetric, functional, or inverse-functional.
  (Which of these are only applicable to object properties?)

Properties (Cont.)

- One property can be inverse of another

  owl:InverseObjectProperties(*livesIn*, *hasResident*)

- Properties can be declared to be transitive, symmetric, functional, or inverse-functional.
  (Which of these are only applicable to object properties?)

- We can also state the minimum and maximal cardinality of a property.

  owl:minCardinality(*principalResidence*, 1)

  owl:maxCardinality(*principalResidence*, 1)

Property and Class Restrictions

- We can define complex descriptions of classes in terms of restrictions of other classes and properties.
  E.g., A homeowner is a person who owns a house.

Property and Class Restrictions

- We can define complex descriptions of classes in terms of restrictions of other classes and properties.
  E.g., A homeowner is a person who owns a house.

$$homeOwner \subseteq person \cap \{x : \exists h \in house \text{ such that } x \; owns \; h\}$$

Property and Class Restrictions

- We can define complex descriptions of classes in terms of restrictions of other classes and properties.
  E.g., A homeowner is a person who owns a house.

$$\mathit{homeOwner} \subseteq \mathit{person} \cap \{x : \exists h \in \mathit{house} \text{ such that } x \ \mathit{owns} \ h\}$$

owl:subClassOf(homeOwner,person)

owl:subClassOf(*homeOwner*,
  owl:ObjectSomeValuesFrom(*owns*, *house*))

OWL Class Constructors

owl:Thing $\equiv$ all individuals

owl:Nothing $\equiv$ no individuals

owl:ObjectIntersectionOf$(C_1, \ldots, C_k) \equiv C_1 \cap \cdots \cap C_k$

owl:ObjectUnionOf$(C_1, \ldots, C_k) \equiv C_1 \cup \cdots \cup C_k$

owl:ObjectComplementOf$(C) \equiv Thing \setminus C$

owl:ObjectOneOf$(I_1, \ldots, I_k) \equiv \{I_1, \ldots, I_k\}$

owl:ObjectHasValue$(P, I) \equiv \{x : x\ P\ I\}$

owl:ObjectAllValuesFrom$(P, C) \equiv \{x : x\ P\ y \rightarrow y \in C\}$

owl:ObjectSomeValuesFrom$(P, C) \equiv$

$\{x : \exists y \in C$ such that $x\ P\ y\}$

owl:ObjectMinCardinality$(n, P, C) \equiv$

$\{x : \#\{y | xPy$ and $y \in C\} \geq n\}$

owl:ObjectMaxCardinality$(n, P, C) \equiv$

$\{x : \#\{y | xPy$ and $y \in C\} \leq n\}$

OWL Predicates

rdf:type$(I, C) \equiv I \in C$
rdfs:subClassOf$(C_1, C_2) \equiv C_1 \subseteq C_2$
owl:EquivalentClasses$(C_1, C_2) \equiv C_1 \equiv C_2$
owl:DisjointClasses$(C_1, C_2) \equiv C_1 \cap C_2 = \{\}$
rdfs:domain$(P, C) \equiv$ if xPy then $x \in C$
rdfs:range$(P, C) \equiv$ if xPy then $y \in C$
rdfs:subPropertyOf$(P_1, P_2) \equiv xP_1y$ implies xP_2y
owl:EquivalentObjectProperties$(P_1, P_2) \equiv xP_1y$ if and only if xP_2y
owl:DisjointObjectProperties$(P_1, P_2) \equiv xP_1y$ implies not xP_2y
owl:InverseObjectProperties$(P_1, P_2) \equiv xP_1y$ if and only if yP_2x
owl:SameIndividual$(I_1, \ldots, I_n) \equiv \forall j \forall k\ I_j = I_k$
owl:DifferentIndividuals$(I_1, \ldots, I_n) \equiv \forall j \forall k\ j \neq k$ implies $I_j \neq I_k$
owl:FunctionalObjectProperty$(P) \equiv$ if xPy_1 and xPy_2 then $y_1 = y_2$
owl:InverseFunctionalObjectProperty$(P) \equiv$
if x_1Py and x_2Py then $x_1 = x_2$
owl:TransitiveObjectProperty$(P) \equiv$ if xPy and yPz then xPz
owl:SymmetricObjectProperty $\equiv$ if xPy then yPx

Knowledge Sharing

- One ontology typically imports and builds on other ontologies.
- OWL provides facilities for version control.
- Tools for mapping one ontology to another allow inter-operation of different knowledge bases.
- The semantic web promises to allow two pieces of information to be combined if
  - they both adhere to an ontology
  - these are the same ontology or there is a mapping between them.

Example: Apartment Building

An apartment building is a residential building with more than two units and they are rented.

Example: Apartment Building

An apartment building is a residential building with more than two units and they are rented.

```
Declaration(ObjectProperty(:numberOfunits))
FunctionalObjectProperty(:numberOfunits)
ObjectPropertyDomain(:numberOfunits :ResidentialBuilding)
ObjectPropertyRange(:numberOfunits
                    ObjectOneOf(:two :one :moreThanTwo))

Declaration(Class(:ApartmentBuilding))
EquivalentClasses(:ApartmentBuilding
    ObjectIntersectionOf(
        :ResidentialBuilding
        ObjectHasValue(:numberOfunits :moreThanTwo)
ObjectHasValue(:ownership :rental)))
```

Aristotelian definitions

Aristotle [350 B.C.] suggested the definition if a class C in terms of:

- Genus: the super-class
- Differentia: the attributes that make members of the class C different from other members of the super-class

"If genera are different and co-ordinate, their differentiae are themselves different in kind. Take as an instance the genus 'animal' and the genus 'knowledge'. 'With feet', 'two-footed', 'winged', 'aquatic', are differentiae of 'animal'; the species of knowledge are not distinguished by the same differentiae. One species of knowledge does not differ from another in being 'two-footed'."

Aristotle, *Categories*, 350 B.C.

Example: hotel ontology

Define the following:

- Room
- BathRoom
- StandardRoom - what is rented as a room in a hotel
- Suite
- RoomOnly

Example: hotel ontology

Define the following:

- Room
- BathRoom
- StandardRoom - what is rented as a room in a hotel
- Suite
- RoomOnly
- Hotel
- HasForRent
- AllSuitesHotel
- NoSuitesHotel
- HasSuitesHotel

Top-Level Ontology — Basic Formal Ontology (BFO)

```
if entity continues to exist through time then
    it is a continuant
    if it doesn't need another entity for its existence then
        it is an independent continuant
        if it has matter as a part then
            it is a material entity
            if it is a single coherent whole then
                it is an object
        else  it is an immaterial entity
    else  it is a dependent continuant
        if it a region in space  then
            it is a spatial region
        else if it is a property then
            if it is a property all objects have  then
                it is a quality
            . . . role . . . disposition . . . function . . .
```

Continuants vs Occurrents

- A continuant exists in an instance of time and maintains its identity through time.
- An occurrent has temporal parts.
- Continuants participate in occurrents.
- a person, a life, a finger, infancy: what is part of what?

Continuants vs Occurrents

- A continuant exists in an instance of time and maintains its identity through time.
- An occurrent has temporal parts.
- Continuants participate in occurrents.
- a person, a life, a finger, infancy: what is part of what?
- a holiday, the end of a lecture, an email, the sending of an email, the equator, earthquake, a smile, a laugh, the smell of a flower

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event:

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event: objects

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event: objects
- a flock of birds, the students in CS422, a card collection:

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event: objects
- a flock of birds, the students in CS422, a card collection: object aggregates

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event: objects
- a flock of birds, the students in CS422, a card collection: object aggregates
- a city, a room, a mouth, the hole of a doughnut:

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event: objects
- a flock of birds, the students in CS422, a card collection: object aggregates
- a city, a room, a mouth, the hole of a doughnut: site

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event: objects
- a flock of birds, the students in CS422, a card collection: object aggregates
- a city, a room, a mouth, the hole of a doughnut: site
- the dangerous part of a city, part of Grouse Mountain with the best view:

Continuants

- a pen, a person, Newtonian mechanics, the memory of a past event: objects
- a flock of birds, the students in CS422, a card collection: object aggregates
- a city, a room, a mouth, the hole of a doughnut: site
- the dangerous part of a city, part of Grouse Mountain with the best view: fiat part of an object.