MANCHESTER 1824

The University of Manchester

OWL: An Ontology Language for the Web

Sean Bechhofer

School of Computer Science,

University of Manchester, UK

http://www.cs.manchester.ac.uk

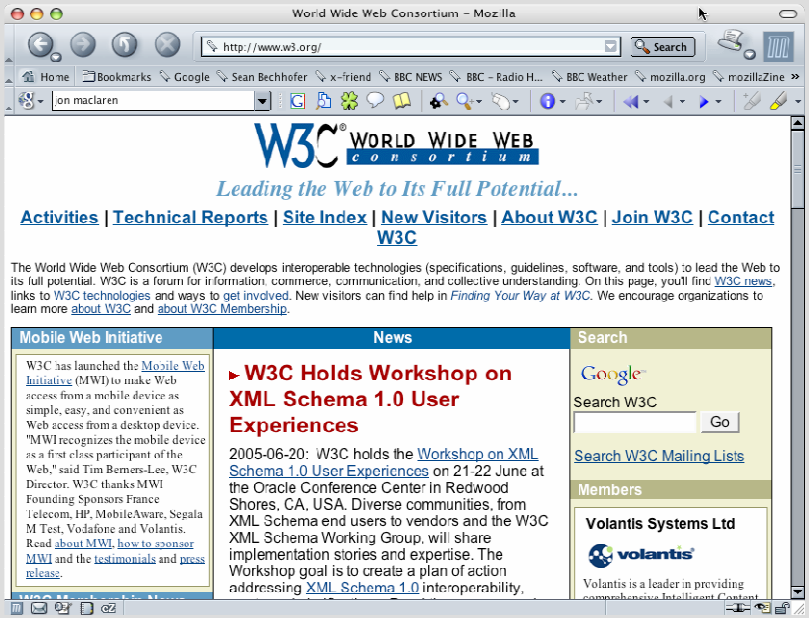
Tutorial Topics

- Context
- A Brief history of OWL
- The OWL Language
- Developments and Extensions

The Semantic Web Vision

- The Web was made possible through established **standards**
  - **TCP/IP** for transporting bits down a wire
  - **HTTP & HTML** for transporting and rendering hyperlinked text
- **Applications** able to exploit this common infrastructure
  - Result is the WWW as we know it
- **1st generation** web mostly handwritten HTML pages
- **2nd generation** (current) web often machine generated/active
  - Both intended for direct human processing/interaction
- In **next generation** web, **resources** should be more accessible to automated processes
  - To be achieved via **semantic markup**
  - **Metadata** annotations that describe content/function
- Coincides with Tim Berners-Lee's vision of a **Semantic Web**

What is the Problem?



- Consider a typical web page
- Markup consists of:
  - rendering information (e.g., font size and colour)
  - Hyper-links to related content
- Semantic content is accessible to humans but not (easily) to computers…
- Requires (at least) NL understanding

A Semantic Web — First Steps

- Make web resources more accessible to automated processes
- Extend existing rendering markup with semantic markup
  - Metadata annotations that describe content/function of web accessible resources
- Use Ontologies to provide vocabulary for annotations
  - New terms can be formed by combining existing ones
  - "Formal specification" is accessible to machines
- A prerequisite is a standard web ontology language
  - Need to agree common syntax before we can share semantics
  - Syntactic web based on standards such as HTTP and HTML

Technologies for the Semantic Web

- Metadata
  - Resources are marked-up with descriptions of their content. No good unless everyone **speaks the same language**;
- Terminologies
  - provide shared and common vocabularies of a domain, so search engines, agents, authors and users can communicate. No good unless everyone **means the same thing**;
- Ontologies
  - provide a shared and common understanding of a domain that can be communicated across people and applications, and will play a major role in supporting information exchange and discovery.

Object Oriented Models

- Many languages use an “object oriented model” with
- **Objects**/Instances/Individuals
  - Elements of the domain of discourse
- **Types**/Classes/Concepts
  - Sets of objects sharing certain characteristics
- **Relations**/Properties/Roles
  - Sets of pairs (tuples) of objects
- Such languages are/can be:
  - Well understood
  - Formally specified
  - (Relatively) easy to use
  - Amenable to machine processing

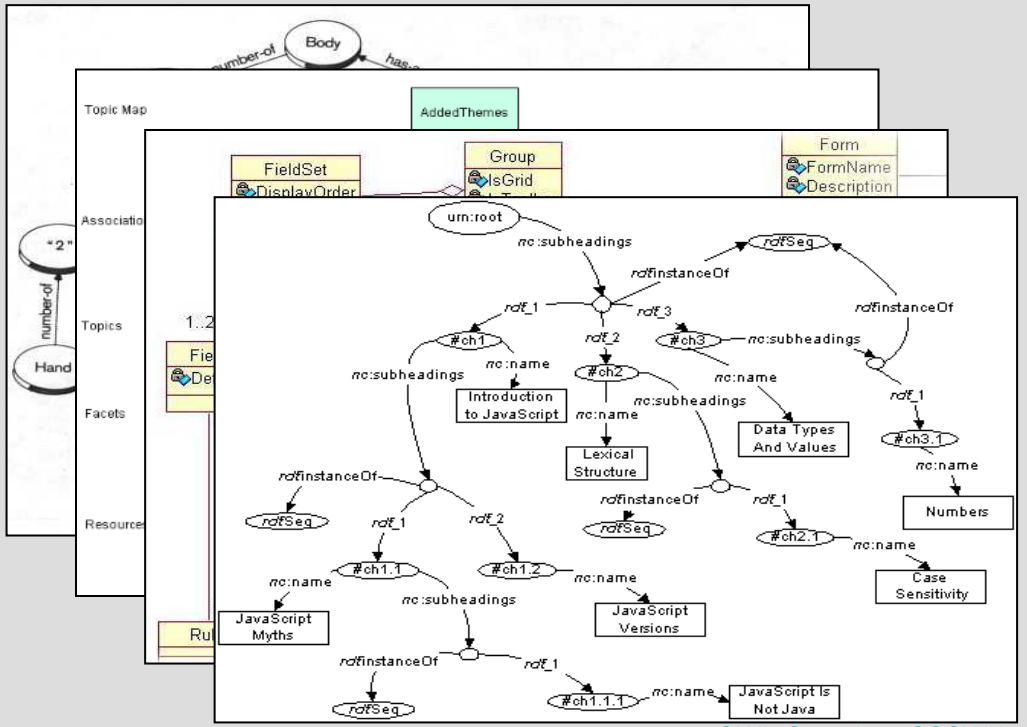
Structure of an Ontology

Ontologies typically have two distinct components:

- Names for important concepts in the domain
  - Elephant is a concept whose members are a kind of animal
  - Herbivore is a concept whose members are exactly those animals who eat only plants or parts of plants
  - Adult_Elephant is a concept whose members are exactly those elephants whose age is greater than 20 years
- Background knowledge/constraints on the domain
  - Adult_Elephants weigh at least 2,000 kg
  - All Elephants are either African_Elephants or Indian_Elephants
  - No individual can be both a Herbivore and a Carnivore

Ontology Languages

- There are a wide variety of languages for “Explicit Specification”
  - Graphical Notations
    - Semantic Networks
    - Topic Maps
    - UML
    - RDF



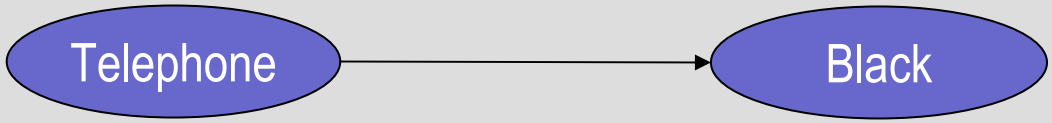
Ontology Languages

- There are a wide variety of languages for "Explicit Specification"
  - Graphical Notations
    - Semantic Networks
    - Topic Maps
    - UML
    - RDF
  - Logic Based
    - Description Logics
    - Rules
    - First Order Logic
    - Conceptual Graphs

Every gardener likes the sun.
(Ax) gardener(x) => likes(x,Sun)
You can fool some of the people all of the time.
(Ex)(At) (person(x) ^ time(t)) => can-fool(x,t)
You can fool all of the people some of the time.
(Ax)(Et) (person(x) ^ time(t) => can-fool(x,t)
All purple mushrooms are poisonous.
(Ax) (mushroom(x) ^ purple(x)) => poisonous(x)
No purple mushroom is poisonous.
~(Ex) purple(x) ^ mushroom(x) ^ poisonous(x)
(Ax) (mushroom(x) ^ purple(x)) => ~poisonous(x)
There are exactly two purple mushrooms.
(Ex)(Ey) mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ~(x=y) ^ (Az)
(mushroom(z) ^ purple(z)) => ((x=z) v (y=z))
Clinton is not tall.
~tall(Clinton)

Why Semantics?

- What does an expression in an ontology **mean**?
- The semantics of a language can tell us precisely how to interpret a complex expression.
- Well defined semantics are vital if we are to support machine interpretability
  - They remove ambiguities in the interpretation of the descriptions.



?

Formal Languages

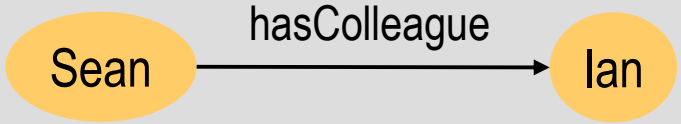
- The degree of formality of ontology languages varies widely
- Increased formality makes languages more amenable to **machine processing** (e.g. automated reasoning).
- The formal semantics provides an **unambiguous** interpretation of the descriptions.

RDF

- RDF stands for Resource Description Framework
- It is a W3C Recommendation
  - http://www.w3.org/RDF
- RDF is a graphical formalism (+ XML syntax + semantics)
  - for representing metadata
  - for describing the semantics of information in a machine-accessible way
- Provides a simple data model based on triples.

The RDF Data Model

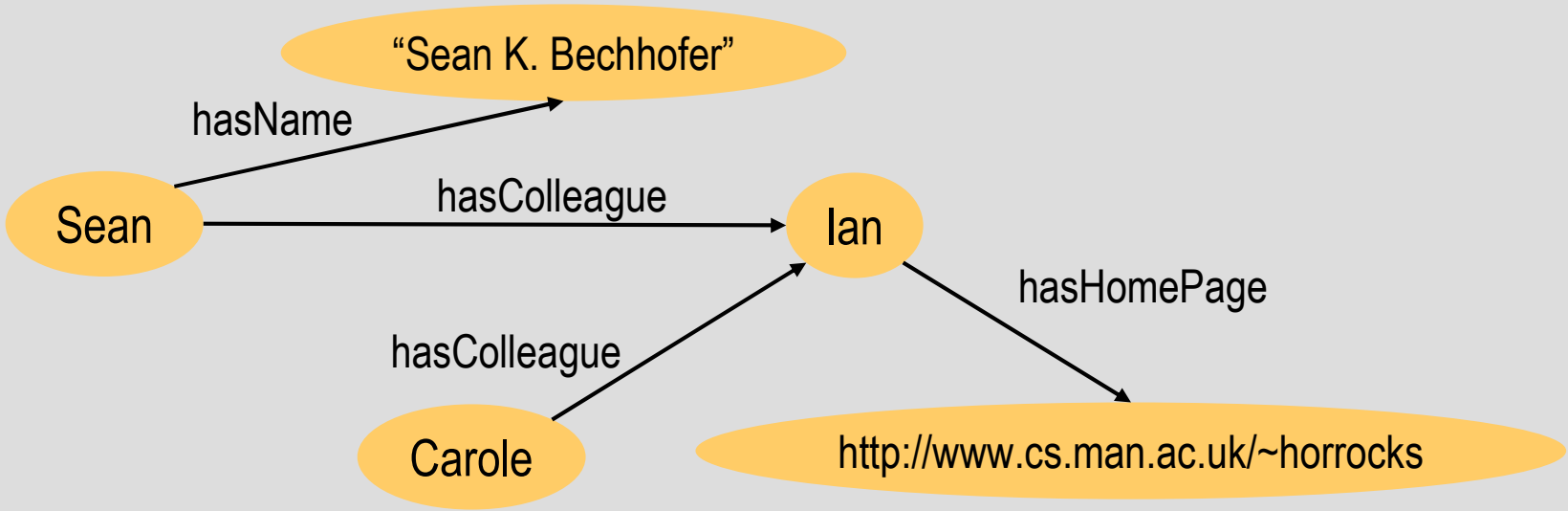
- Statements are <subject, predicate, object> triples:
  - `<Sean,hasColleague,Ian>`
- Can be represented as a graph:



- Statements describe properties of resources
- A resource is any object that can be pointed to by a URI:
  - The generic set of all names/addresses that are short strings that refer to resources
  - a document, a picture, a paragraph on the Web, http://www.cs.man.ac.uk/index.html, a book in the library, a real person (?), isbn://0141184280
- Properties themselves are also resources (URIs)

Linking Statements

- The subject of one statement can be the object of another
- Such collections of statements form a directed, labeled graph



- Note that the object of a triple can also be a "literal" (a string)

RDF Syntax

- RDF has an XML syntax that has a specific meaning:
- Every `Description` element describes a resource
- Every attribute or nested element inside a `Description` is a `property` of that `Resource`
- We can refer to resources by URIs

```
<Description about="some.uri/person/sean_bechhofer">
  <hasColleague resource="some.uri/person/ian_horrocks"/>
  <hasName rdf:datatype="&xsd;string">Sean K. Bechhofer</hasName>
</Description>
<Description about="some.uri/person/ian_horrocks">
  <o:hasHomePage>http://www.cs.mam.ac.uk/~horrocks</o:hasHomePage>
</Description>
<Description about="some.uri/person/carole_goble">
  <o:hasColleague resource="some.uri/person/ian_horrocks"/>
</Description>
```

RDF(S): RDF Schema

- RDF gives a formalism for meta data annotation, and a way to write it down in XML, but it does not give any special meaning to vocabulary such as subClassOf or type
  - Interpretation is an arbitrary binary relation
- RDF Schema extends RDF with a schema vocabulary that allows you to define basic vocabulary terms and the relations between those terms
  - Class, type, subClassOf,
  - Property, subPropertyOf, range, domain
  - it gives “extra meaning” to particular RDF predicates and resources
  - this “extra meaning”, or semantics, specifies how a term should be interpreted

RDF(S) Examples

- RDF Schema terms (just a few examples):
  - Class; Property
  - type; subClassOf
  - range; domain
- These terms are the RDF Schema building blocks (constructors) used to create vocabularies:
  - `<Person,type,Class>`
  - `<hasColleague,type,Property>`
  - `<Professor,subClassOf,Person>`
  - `<Carole,type,Professor>`
  - `<hasColleague,range,Person>`
  - `<hasColleague,domain,Person>`

RDF/RDF(S) "Liberality"

- No distinction between classes and instances (individuals)

  `<Species,type,Class>`

  `<Lion,type,Species>`

  `<Leo,type,Lion>`

- Properties can themselves have properties

  `<hasDaughter,subPropertyOf,hasChild>`

  `<hasDaughter,type,familyProperty>`

- No distinction between language constructors and ontology vocabulary, so constructors can be applied to themselves/each other

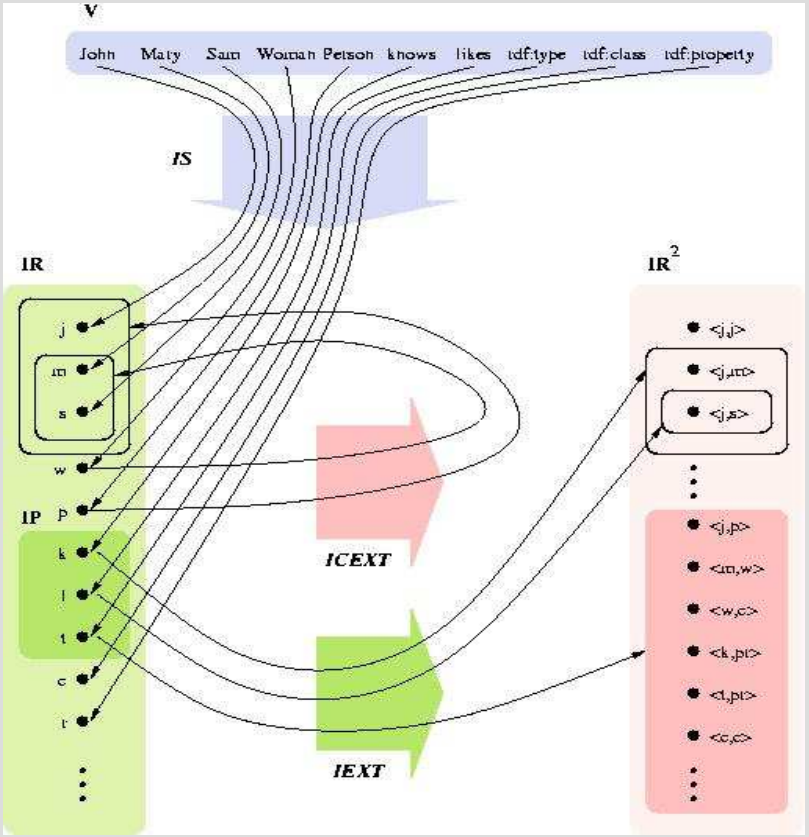
  `<type,range,Class>`

  `<Property,type,Class>`

  `<type,subPropertyOf,subClassOf>`

RDF/RDF(S) Semantics

- RDF has “Non-standard” semantics given by RDF Model Theory (MT)
  - IR, a non-empty set of resources
  - IS, a mapping from V into IR
  - IP, a distinguished subset of IR (the properties)
  - IEXT, a mapping from IP into the powerset of IR£IR
- Class interpretation ICEXT induced by IEXT(IS(type))
  - ICEXT(C) = {x | (x,C) 2 IEXT(IS(type))}
- RDF(S) adds constraints on models
  - {(x,y), (y,z)} µ IEXT(IS(subClassOf))) (x,z) 2 IEXT(IS(subClassOf))



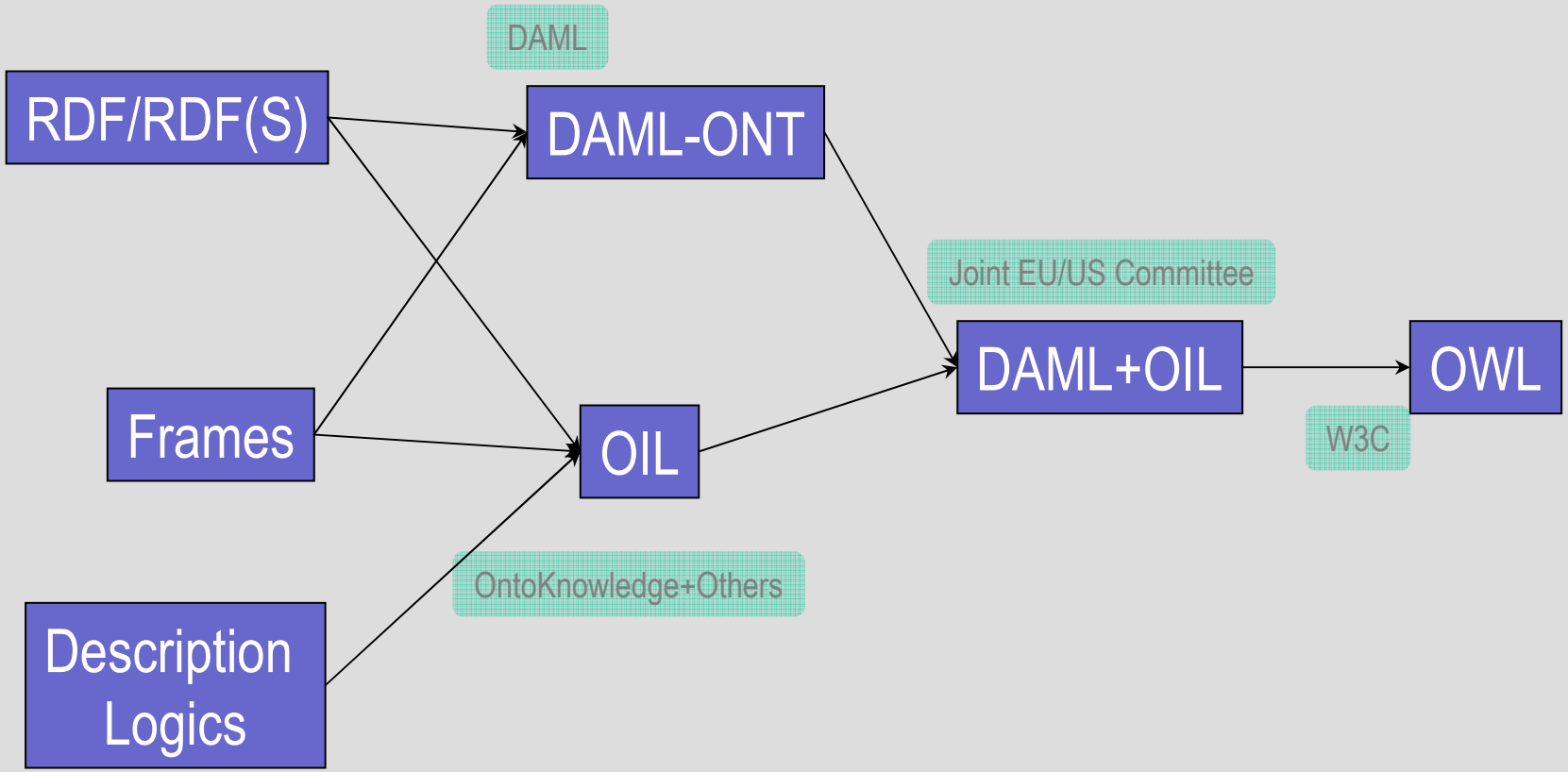
Problems with RDF(S)

- RDF(S) is too weak to describe resources in sufficient detail
  - No localised range and domain constraints
    - Can't say that the range of hasChild is Person when applied to Persons and Elephant when applied to Elephants
  - No existence/cardinality constraints
    - Can't say that all *instances* of Person have a mother that is also a Person, or that Persons have exactly 2 parents
  - No transitive, inverse or symmetrical properties
    - Can't say that isPartOf is a transitive property, that hasPart is the inverse of isPartOf or that touches is symmetrical
- Difficult to provide reasoning support
  - No "native" reasoners for non-standard semantics
  - May be possible to reason via FO axiomatisation

Solution

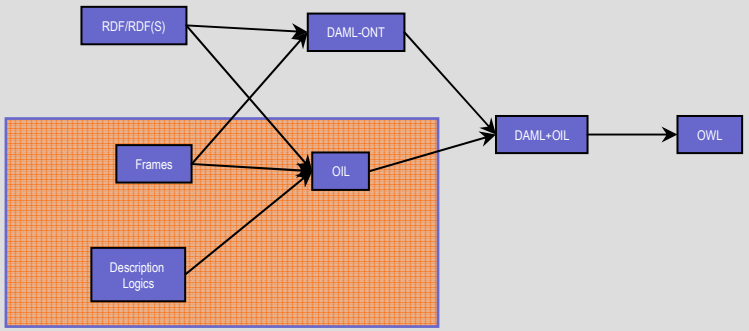
- Extend RDF(S) with a language that has the following desirable features identified for Web Ontology Language
  - Extends existing Web standards
    - Such as XML, RDF, RDFS
  - Easy to understand and use
    - Should be based on familiar KR idioms
  - Of "adequate" expressive power
  - Formally specified
    - Possible to provide automated reasoning support
- That language is **OWL**.

The OWL Family Tree



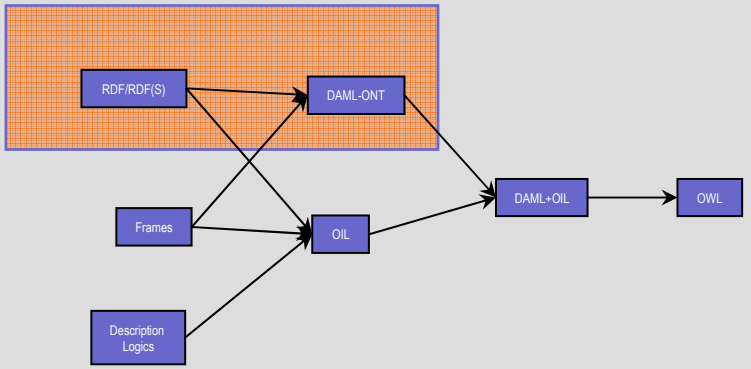
A Brief History of OWL

- **OIL**
  - Developed by group of (largely) European researchers (several from EU OntoKnowledge project)
  - Based on frame-based language
  - Strong emphasis on formal rigour.
  - Semantics in terms of Description Logics
  - RDFS based syntax



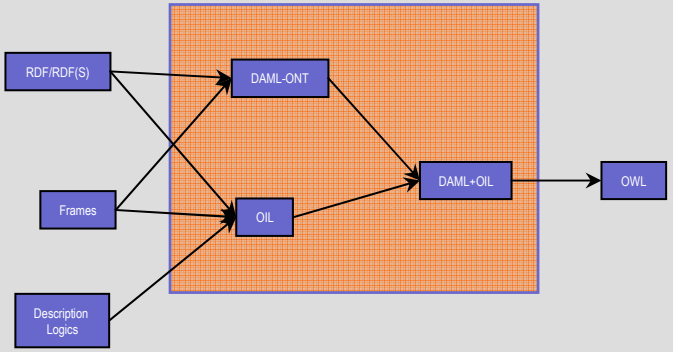
A Brief History of OWL

- **DAML-ONT**
  - Developed by DAML Programme.
    - Largely US based researchers
  - Extended RDFS with constructors from OO and frame-based languages
  - Rather weak semantic specification
    - Problems with machine interpretation
    - Problems with human interpretation



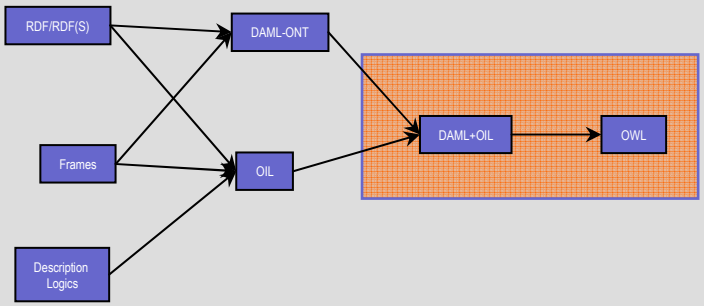
A Brief History of OWL

- **DAML+OIL**
  - Merging of DAML-ONT and OIL
  - Basically a DL with an RDFS-based syntax.
  - Development was carried out by "Joint EU/US Committee on Agent Markup Languages"
  - Extends ("DL subset" of) RDF
- DAML+OIL submitted to W3C as basis for standardisation
  - Web-Ontology (**WebOnt**) Working Group formed



A Brief History of OWL

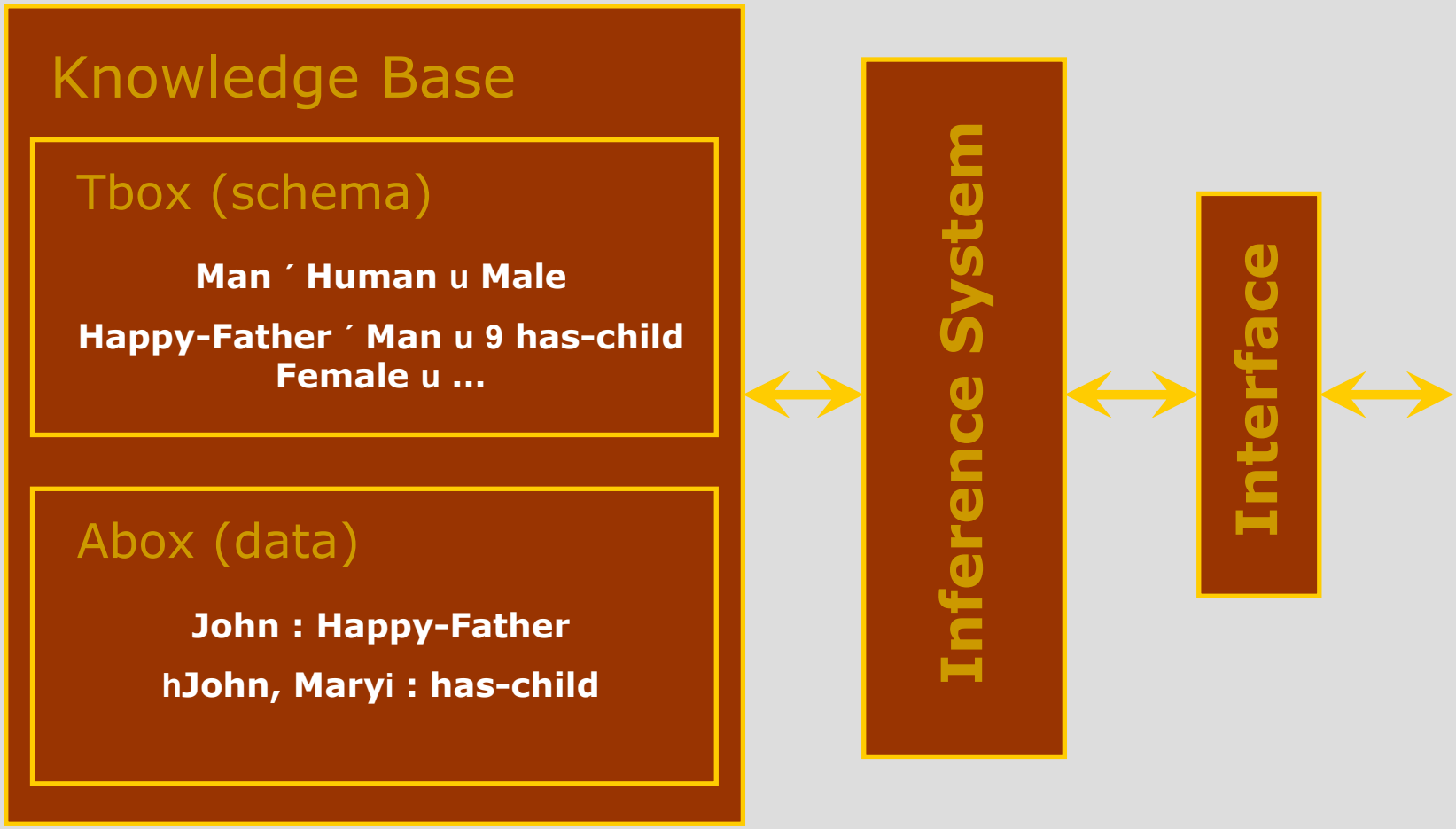
- **OWL**
  - W3C Recommendation (February 2004)
  - Based largely on the DAML+OIL specification from March 2001.
  - Well defined RDF/XML serializations
  - Formal semantics
    - First Order
    - Relationship with RDF
  - Comprehensive test cases for tools/implementations
  - Growing industrial takeup.



Aside: Description Logics

- A family of logic based Knowledge Representation formalisms
  - Descendants of semantic networks and KL-ONE
  - Describe domain in terms of concepts (classes), roles (relationships) and individuals
- Distinguished by:
  - Formal semantics (typically model theoretic)
    - Decidable fragments of FOL
    - Closely related to Propositional Modal & Dynamic Logics
  - Provision of inference services
    - Sound and complete decision procedures for key problems
    - Implemented systems (highly optimised)

DL Architecture



A Brief History of DLs

Phase 1:

- Incomplete systems (Back, Classic, Loom, . . .)
- Based on structural algorithms

Phase 2:

- Development of tableau algorithms and complexity results
- Tableau-based systems for Pspace logics (e.g., Kris, Crack)
- Investigation of optimisation techniques

Phase 3:

- Tableau algorithms for very expressive DLs
- Highly optimised tableau systems for ExpTime logics (e.g., FaCT, DLP, Racer)
- Relationship to modal logic and decidable fragments of FOL

A Brief History of DLs

Phase 4:

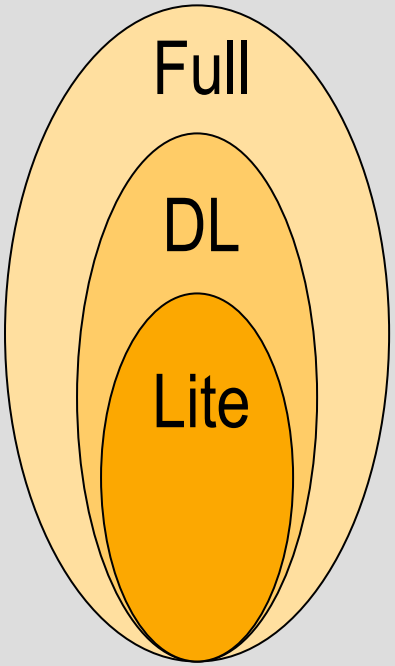
- Mature implementations
- Mainstream applications and Tools
  - Databases
    - Consistency of conceptual schemata (EER, UML etc.)
    - Schema integration
    - Query subsumption (w.r.t. a conceptual schema)
  - Ontologies and Semantic Web (and Grid)
    - Ontology engineering (design, maintenance, integration)
    - Reasoning with ontology-based markup (meta-data)
    - Service description and discovery
- Commercial implementations
  - Cerebra system from Cerebra
  - RacerPro from Racer Systems GmbH

DL Semantics

- **Model theoretic** semantics. An interpretation consists of
  - A domain of discourse (a collection of objects)
  - Functions mapping
    - classes to sets of objects
    - properties to sets of pairs of objects
  - Rules describe how to interpret the constructors and tell us when an interpretation is a model.
- In a DL, a class description is thus a characterisation of the individuals that are members of that class.

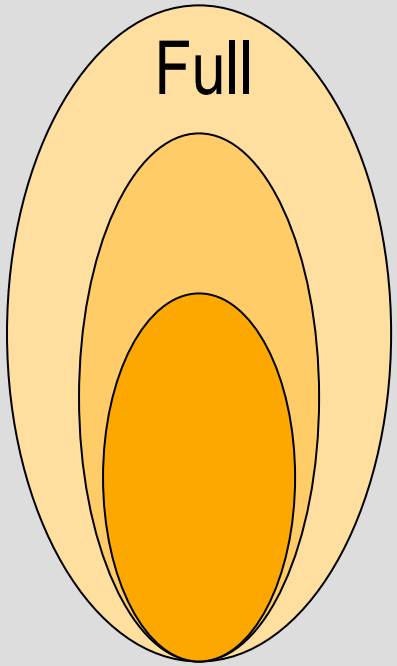
OWL Layering

- Three species of OWL
  - OWL Full is the union of OWL syntax and RDF
  - OWL DL restricted to FOL fragment (¼ DAML+OIL)
    - Corresponds to *SHOIN*(D_n) Description Logic
  - OWL Lite is "simpler" subset of OWL DL
- Syntactic Layering
- Semantic Layering
  - OWL DL semantics = OWL Full semantics (within DL fragment)
  - OWL Lite semantics = OWL DL semantics (within Lite fragment)
- DL semantics are **definitive**
  - In principle: correspondence proof
  - But: if Full disagrees with DL (in DL fragment), then Full is wrong



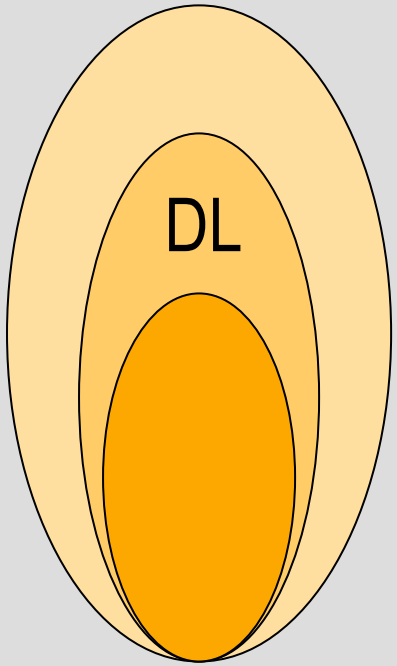
OWL Full

- No restriction on use of OWL vocabulary (as long as legal RDF)
  - Classes as instances (and much more)
- RDF style model theory
  - Reasoning using FOL engines
    - via axiomatisation
  - Semantics should correspond with OWL DL for suitably restricted KBs



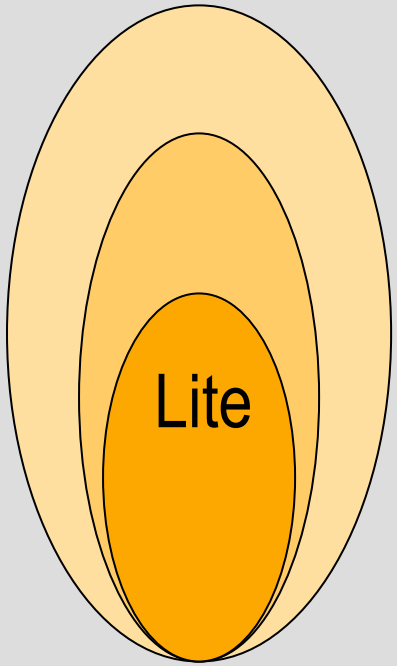
OWL DL

- Use of OWL vocabulary restricted
  - Can't be used to do "nasty things" (i.e., modify OWL)
  - No classes as instances
  - Defined by abstract syntax + mapping to RDF
- Standard DL/FOL model theory (definitive)
  - Direct correspondence with (first order) logic
- Benefits from years of DL research
  - Well defined semantics
  - Formal properties well understood (complexity, decidability)
  - Known reasoning algorithms
  - Implemented systems (highly optimised)



OWL Lite

- Like DL, but fewer constructs
  - No explicit negation or union
  - Restricted cardinality (zero or one)
  - No nominals (oneOf)
- Semantics as per DL
  - Reasoning via standard DL engines (+datatypes)
    - E.g., FaCT, RACER, Cerebra, Pellet



OWL Syntaxes

- Abstract Syntax
  - Used in the definition of the language and the DL/Lite semantics
- OWL in RDF (the “official” concrete syntax)
  - RDF/XML presentation
- XML Presentation Syntax
  - XML Schema definition

OWL Class Constructors

- OWL has a number of operators for constructing class expressions.
- These have an associated semantics which is given in terms of a domain:
  - Δ
- And an interpretation function
  - *I*:*concepts* ! $\wp(\Delta)$
  - *I*:*properties* ! $\wp(\Delta \pounds \Delta)$
  - *I*:*individuals* ! Δ
- *I* is then extended to concept expressions.

OWL Class Constructors

Constructor	Example	Interpretation
Classes	Human	*I(Human)*
intersectionOf	intersectionOf(Human Male)	*I(Human)* Å *I(Male)*
unionOf	unionOf(Doctor Lawyer)	*I(Doctor)* [*I(Lawyer)*
complementOf	complementOf(Male)	Δ n *I(Male)*
oneOf	oneOf(john mary)	{*I(john)*, *I(mary)*}

MANCHESTER 1824

The University of Manchester

OWL Class Constructors

Constructor	Example	Interpretation
someValuesFrom	restriction(hasChild someValuesFrom Lawyer)	{xj9y.hx,yi2*I(hasChild)*Æ y2*I(Lawyer)*}
allValuesFrom	restriction(hasChild allValuesFrom Doctor)	{xj8y.hx,yi2*I(hasChild)*) y2*I(Doctor)*}
minCardinality	restriction(hasChild minCardinality (2))	{x|#hx,yi2*I(hasChild)* ¸ 2}
maxCardinality	restriction(hasChild maxCardinality (2))	{x|#hx,yi2*I(hasChild)* · 2}

OWL Axioms

- Axioms allow us to add further statements about arbitrary concept expressions and properties
  - Subclasses, Disjointness, Equivalence, transitivity of properties etc.
- An interpretation is then a model of the axioms iff it satisfies every axiom in the model.

Axiom	**Example**	**Interpretation**
SubClassOf	SubClassOf(Human Animal)	*I(Human)* µ *I(Animal)*
EquivalentClasses	EquivalentClass(Man intersectionOf(Human Male))	*I(Man)* = *I(Human)* Å *I(Male)*
DisjointClasses	DisjointClasses(Animal Plant)	*I(Animal)* Å *I(Plant)* = ;

OWL Individual Axioms

Axiom	Example	Interpretation
Individual	Individual(Sean type(Human))	*I(Sean)* 2 *I(Human)*
Individual	Individual(Sean value(worksWith Ian))	h*I(Sean)*,*I(Ian)*i2*I(worksWith)*
DifferentIndividuals	DifferentIndividuals(Sean Ian)	*I(Sean)* ≠ *I(Ian)*
SameIndividualAs	SameIndividualAs(GeorgeWBush PresidentBush)	*I(GeorgeWBush)* = *I(PresidentBush)*

OWL Property Axioms

Axiom	Example	Interpretation
SubPropertyOf	SubPropertyOf(hasMother hasParent)	*I(hasMother)* µ *I(hasParent)*
domain	ObjectProperty (owns domain(Person))	8x.hx,yi2*I(owns)*) x2*I(Person)*
range	ObjectProperty (employs range(Person))	8x.hx,yi2*I(employs)*) y2*I(Person)*
transitive	ObjectProperty(hasPart Transitive)	8x,y,z. *(*hx,yi2*I(hasPart)* Æ hy,zi2*I(hasPart))*) hx,zi2*I(hasPart)*

Semantics

- An interpretation *I* satisfies an axiom if the interpretation of the axiom is true.
- *I* satisfies or is a model of an ontology (or knowledge base) if the interpretation satisfies all the axioms in the knowledge base (class axioms, property axioms and individual axioms).
- *C* subsumes *D* w.r.t. an ontology *O* iff for *every* model *I* of *O*, $I(D) \mu I(C)$
- *C* is equivalent to *D* w.r.t. an ontology *O* iff for *every* model *I* of *O*, $I(C) = I(D)$
- *C* is satisfiable w.r.t. *O* iff there exists *some* model *I* of *O* s.t. $I(C) \neq ;$
- An ontology *O* is consistent iff there exists *some* model *I* of *O.*

Reasoning

- A reasoner makes use of the information asserted in the ontology.
- Based on the semantics described, a reasoner can help us to discover inferences that are a consequence of the knowledge that we've presented that we weren't aware of beforehand.
- Is this new knowledge?
  - What's actually in the ontology?

Reasoning

- Subsumption reasoning
  - Allows us to infer when one class is a subclass of another
  - B is a subclass of A if it is necessarily the case that (in all models), all instances of B *must* be instances of A.
  - This can be either due to an explicit assertion, or through some inference process based on an intensional definition.
  - Can then build concept hierarchies representing the taxonomy.
  - This is classification of classes.
- Satisfiability reasoning
  - Tells us when a concept is unsatisfiable
    - i.e. when there is no model in which the interpretation of the class is non-empty.
  - Allows us to check whether our model is consistent.

Instance Reasoning

- Instance Retrieval
  - What are the instances of a particular class C?
  - Need not be a named class
- Instantiation
  - What are the classes that x is an instance of?

Why Reasoning?

- Reasoning can be used as a design support tool
  - Check logical consistency of classes
  - Compute implicit class hierarchy
- May be less important in small local ontologies
  - Can still be useful tool for design and maintenance
  - Much more important with larger ontologies/multiple authors
- Valuable tool for integrating and sharing ontologies
  - Use definitions/axioms to establish inter-ontology relationships
  - Check for consistency and (unexpected) implied relationships
  - Already shown to be useful technique for DB schema integration
- For most DLs, the basic inference problems are decidable (e.g. there is some program that solves the problem in a finite number of steps)

Example

```
Class(pet_owner complete
  intersectionOf(person
                 restriction(has_pet someValuesFrom (animal))))
```

- A pet_owner is a person that has some pet that is an animal
- This is a complete definition, thus any person who has a pet that is an animal will be a pet_owner.
- If we know someone is a pet_owner, then we know that there must be some animal that is their pet: we may not know the name of this particular animal though.

Example

```
Class(giraffe partial animal
    restriction(eats allValuesFrom (leaf)))
```

- A giraffe is an animal that only eats leaves.
- This is a partial definition, thus every giraffe must have these characteristics, however there may be animals that eat only leaves that are **not** giraffes.

Necessary and Sufficient Conditions

- Classes can be described in terms of necessary and sufficient conditions.
  - This differs from some frame-based languages where we only have necessary conditions.
- Necessary conditions
  - Must hold if an object is to be an instance of the class
- Sufficient conditions
  - Those properties an object must have in order to be recognised as a member of the class.
  - Allows us to perform automated classification.

If it looks like a duck and walks like a duck, then it's a duck!

Example

```
Class(animal_lover complete
  intersectionOf(person restriction(has_pet minCardinality(3))))
```

- An animal_lover is a person that has at least 3 pets.
- All of these pets must be distinct individuals.
- Any person with 5 pets will be inferred to be an instance of this class.

Example

```
Class(newspaper complete
  unionOf(tabloid broadsheet))
```

- A newspaper is either a broadsheet or a tabloid.
- By default there is no mutual exclusion.
- If we know something is a newspaper we can infer that it must be either a broadsheet or a tabloid, but we may not know for sure which one it actually is (cf Open World).

Example

```
Individual(Mick type(male)
  value(reads Daily_Mirror)
  value(drives Q123_ABC))
```

- Mick is an individual and an instance of the class male.
- He is related to individuals Daily_Mirror and Q123_ABC via the properties reads and drives.

Common Misconceptions

- Disjointness of primitives
- Interpreting domain and range
- And and Or
- Quantification
- Closed and Open Worlds

Disjointness

- By default, primitive classes are not disjoint.
- Unless we explicitly say so, the description (Animal and Vegetable) is not inconsistent.
- Similarly with individuals -- the so-called Unique Name Assumption (often present in DL languages) does not hold, and individuals are not considered to be distinct unless **explicitly** asserted to be so.

Domain and Range

- OWL allows us to specify the domain and range of properties.
- Note that this is not interpreted as a constraint as you might expect.
- Rather, the domain and range assertions allow us to make inferences about individuals.
- Consider the following:
  - ObjectProperty(employs domain(Company) range(Person))
    Individual(IBM value(employs Jim))
- If we haven't said anything else about IBM or Jim, this is **not** an error. However, we can now infer that IBM is a Company and Jim is a Person.

And/Or and Quantification

- The logical connectives And and Or often cause confusion
  - Tea or Coffee?
  - Milk and Sugar?
- Quantification can also be contrary to our intuition.
  - Universal quantification over an empty set is true.
  - Sean is a member of restriction(hasChild allValuesFrom Martian)
  - Existential quantification may imply the existence of an individual that we don't know the name of.

Closed and Open Worlds

- The standard semantics of OWL makes an Open World Assumption (OWA).
  - We cannot assume that all information is known about all the individuals in a domain.
  - Facilitates reasoning about the intensional definitions of classes.
  - Sometimes strange side effects
- Closed World Assumption (CWA), or negation as failure.
  - If we can't deduce that x is an A, then we know it must be a (not A).
  - Facilitates reasoning about a particular state of affairs.

Extensions

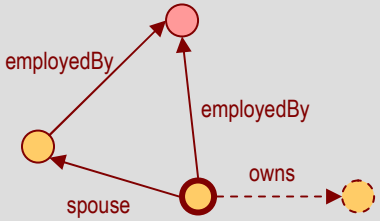
- OWL is not intended to be the answer to all our problems.
- There are things that we can't represent using OWL.
- Current work on extending OWL includes:
  - Rules
  - Richer Role Composition Axioms
  - Modularity

Extensions: SWRL

- OWL lacks a composition operator for properties, so we cannot defined relationships such as "uncle"
  - Your uncle is the brother of one of your parents.
- One way to overcome this is by the addition of rules.
- **SWRL**: A proposal for a Semantic Web Rule Language
- Extends OWL with Horn-like rules
- Rules can make use of OWL descriptions in both head and body
- W3C Member submission as of May 2004.
- Model-theoretic semantics (extension of OWL DL semantics).

Rules: SWRL

- Extends OWL expressivity, allowing inference of relations:
  - hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) → hasUncle(?x1,?x3)
  - An uncle is the brother of a parent.
- Extends rules to allow existential quantification in rule heads:
  - HighEarner(?x) ∧ spouse(?x, ?y) ∧ employedBy(?x, ?a) ∧ employedBy(?y, ?a) → some owns.FastCar(?x)
  - If you're a high earner and you have the same employer as your spouse, then you own a fast car.
- Rules can involve named individuals.
  - Beer(?x) → Happy(Sean)
  - If there are any instances of Beer, then Sean is happy.



Rules: Reasoning with SWRL

- Extending OWL in this way results in a language which is no longer decidable.
- Possible approaches to reaoning with SWRL include translation to First Order Logic, then using state of the art FOL provers.
  - May be unable to provide an answer.
- Alternatively use rule-based systems, which may not be able to cope so well with the OWL reasoning aspects.

Extensions: Complex Role Axioms

- Many applications (for example medicine) have requirements to specify interactions between roles:
  - A fracture located in part of the Femur is a fracture of the Femur.
- We **cannot** express such general patterns in OWL.
- Algorithms have been developed to support sound and complete reasoning in a DL extended with complex role inclusions
  - Providing support for a well-behaved extension of OWL.

Extensions: Query and Retrieval

- In standard DLs, reasoning is split into:
  - T-Box: reasoning about classes
  - A-Box: reasoning about instances
- T-Box reasoning is well understood, at least for languages like SHIQ (~OWL Lite)
  - e.g. subsumption & satisfiability testing
- Full A-Box reasoning is much more challenging
  - E.g. instance retrieval & instantiation

Extensions: Query Languages

- The basic DL reasoning tasks provide rather primitive queries:
  - Retrieve all instances of a particular class
  - Retrieve the classes this individual is known to be an instance of
- Richer query languages are clearly needed to support applications.
- OWL Query Language (OWL-QL) is a proposal for a query language (based on DQL (DAML Query Language))
- Query Example:
  - Query: (“Who owns a red car?”)
    Query Pattern: {(owns ?p ?c) (type ?c Car) (has-color ?c Red)}
    Must-Bind Variables List: (?p)
    May-Bind Variables List: (?c)

Extensions: Datatyping

- OWL Recommendation uses simple data types XML Schema
  - OWL reasoners only need to understand xsd:int and xsd:string.
- How to refer to user defined datatypes within OWL in RDF
  - User defined datatypes
  - Datatype predicates
- Reasoning with datatypes
  - Prototype implementations
- W3C Semantic Web Best Practices WG

Extensions: Modularity

- OWL provides a very simple mechanism for modularisation:
  - owl:import
- This pulls in another RDF graph and adds it to the graph of the importing ontology
- Applications may choose to add additional behaviour
  - E.g. don't allow users to edit imported ontologies.
- This is not really enough to support real modularisation
  - Ontology signatures or interfaces
  - Information Hiding
  - Distributed Reasoning

Tools

- Editors
  - OilEd, SWOOP, Protégé
  - Tend to present the user with “frame-like” interfaces, but allow richer expressions
  - Offer the possibility of using reasoners.
- Reasoners
  - DL style reasoners based on tableaux algorithms
    - Racer, FaCT++, Pellet, Cerebra
  - Based on rules or F-logic
    - F-OWL, E-Wallet…..
  - Based on translation to FOL
    - Hoolet
- APIs and Frameworks
  - Jena, WonderWeb OWL-API, Protégé OWL API

Summary

- OWL provides us with a rich language for defining ontologies.
- Builds upon RDF and RDF Schema
- Formal semantics
  - Provides an unambiguous interpretation of expressions and facilitates the use of reasoners.
  - Draws on years of DL research.
- Language extensions in the pipeline.
- Tools, both research and commercial, are emerging to support OWL
  - Reasoners
  - Editors
  - Query Engines

MANCHESTER 1824

The University of Manchester

Acknowledgements

- "If I have seen further, it is by standing on the shoulders of giants"
- Many thanks to all the people who I "borrowed" material from, in particular
  - Ian Horrocks, Frank van Harmelen, Alan Rector, Nick Drummond, Matthew Horridge
- and thanks to all those that *they* borrowed material from!
  - Too many to mention…

MANCHESTER 1824

The University of Manchester

Thank you!

http://www.w3.org/2004/OWL/