Disclaimer

This presentation is based on papers, tutorials, and presentations by

Tim Berners-Lee, Ludger van Elst, Asun Gomez-Perez, Nicola Guarino, Frank van Harmelen, James Hendler, Ian Horrocks, Markus Krötzsch, Alan Rector, Thomas Roth-Berghofer, Michael Sintek, Vagan Terziyan, and (probably) others
Overview

Part I  Ontologies and the Semantic Web (LvE)
  - Motivation for Ontologies in the Semantic Web
  - Characteristics of Ontologies

Part II  Semantic Web Ontology Modeling (MS)
  - Representation Languages – RDF/S, OWL
  - Queries, Rules and Inferencing

Part III  Tools & Applications (MS)
  - Protégé
  - Semantic Search, SmartWeb, NEPOMUK

Part IV  Challenges (MS+LvE)
  - Sharing, Stability, Formality
Vision of the Semantic Web

The entertainment system was belting out the Beatles’ “We Can Work It Out” when the phone rang. When Pete answered, his phone turned the sound down by sending a message to all the other local devices that had a volume control. His sister, Lucy, was on the line from the doctor’s office: “Mom needs to see a specialist and then has to have a series of physical therapy sessions. Biweekly or something. I’m going to have my agent set up the appointments.” Pete immediately agreed to share the chauffeuring. At the doctor’s office, Lucy instructed her Semantic Web agent through her handheld Web browser. The agent promptly retrieved information about Mom’s prescribed treatment from the doctor’s agent, looked up several lists of providers, and checked for the ones in-plan for Mom’s insurance within a 20-mile radius of her home and with a rating of excellent or very good on trusted rating services. It then began trying to find a match between available appointment times (supplied by the agents of individual providers through their Web sites) and Pete’s and Lucy’s busy schedules.


Example: Doctor’s Appointment

Characteristics of the Example (1):

**Communication between Various Actors**

- **Machines**
  - Services, agents (living in the WWW)
  - communicating with machines
  - communicating with humans

- **Humans**
  - With individual goals, levels of expertise, ...
  - Communicating with humans
    - Direct
    - Mediated
  - Communicating with machines

Characteristics of the Example (2):

**Various Domains**

- **Temporal**: Schedules, ...
- **Location**: Addresses
- **Medical**: Disease, treatment
- **Financial**: Payment
- **Assessments**: Ratings
- **Organizational Structures**: Hospitals & doctors
- Informative text on various topics
- ...

---

*FOIS 2008 Tutorial: Ontologies and the Semantic Web: Building Blocks and Challenges*
Characteristics of the Example (3): Openness

- Same entities in the world may have different names and identifiers
  - concrete entities like persons, but also abstract concepts like diseases
- Potentially conflicting statements and data
- Agents and services may steadily enter or leave the scene

Some Derived Requirements

- **Shared Knowledge**
  - On the instance level: *When is the appointment?*
  - On the conceptual level: *What does it mean to have an appointment?*
  - (at least) partially shared between computers and humans

- **Formal Knowledge**
  - To allow agents/services to reason on knowledge
Ontologies in Knowledge-based Systems (1)

- Coined in the late 80s, early 90s within the Knowledge Sharing and Reuse Effort

- E.g., Gruber (1993): "explicit specification of a conceptualization"

- Goal: Re-use of knowledge
  - Neches et al. (1991): "In this article, we present a vision of the future in which the idea of knowledge sharing is commonplace. If this vision is realized, building a new system will rarely involve constructing a new knowledge base from scratch. Instead, the process of building a knowledge-based system will start by assembling reusable components."
  - Consider: Knowledge-based systems in the 80s have not the same characteristics as (envisioned) Semantic Web applications!

Ontologies in Knowledge-based Systems (2)

“A formal, explicit specification of a shared conceptualisation.” (Studer et al., 1998)

- Conceptualisation: an abstract model of some phenomenon in the world,
- Explicit: the type of concepts used and the constraints on their use are explicitly defined
- Formal: machine readable
  - A spectrum of levels: from controlled vocabularies to higher-order logics
- Shared: consensual knowledge
  - Size of "consensus groups": Small groups, big groups, ..., world
  - Types of actors: Machines, software engineers, end-users, ...
Classical Definition of the O-Word from an Information Systems Perspective

"An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e., its ontological commitment to a particular conceptualization of the world."

Guarino (1998)

Communication: Semiotic Triangle

Problem: Potential diversity on all three corners!
Problems with Knowledge Sharing

Why Formality Facilitates Sharing (1)
Why Formality Facilitates Sharing (2):

Conceptualization $C$

Language $L$

Intended models $I_k(L)$

Ontology

Models $M(L)$

Why Formality Facilitates Sharing (3):

Communication Requires Overlap

Conceptualization $C$

Language $L$

Intended models $I_A(L)$

Intended models $I_B(L)$

Ontology

Models $M(L)$
Why Formality Facilitates Sharing (4)

Conceptualization $C$

Language $L$

Intended models $I_A(L)$

Intended models $I_B(L)$

Models $M(L)$

Why Formality Facilitates Sharing (4): Ontologies Exclude (Some) Models

Conceptualization $C$

Language $L$

Intended models $I_A(L)$

Intended models $I_B(L)$

Models $M(L)$

Ontology
Human and machine communication

Adapted from [Maedche et al., 2002]

Ontology

Description

exchange symbol, e.g. via natural language
"Mom", "Mrs. Bundy"

Internal models

Formal models

Commit

Ontology

Symbol

Concept

Meaning Triangle

Things/Entities

The Semantic Web Needs Different Multiple Levels of Formality and Sharing

Sharing scope

Word

Group

Personal

Folksonomies

Web Directories

Ontologies

Group Directories

Tags

File Folders

Personal Information Models

Level of formality/expressivity
Beware of the Subtle Interactions!

- **Formalization is expensive** in terms of time and money
  - requires: „use time“ >> „formalization time“ i.e., high stability required
  - but: stability mostly externally given
- **Formality allows for sharing** (explicitness, precision)
  - prerequisites formal training
  - possibly keeps away agents from participation
  - wide sharing scope increases costs of negotiation
Ontologies are Meant to Capture Application-independent Knowledge

- An ontology as part of a Knowledge Base contains facts assumed to be always true by a community of users, in virtue of the agreed-upon meaning of the vocabulary used.
  - Application-independent knowledge is typically also more stable!
  - What a community considers as truth has a higher chance to be stable.

Layering of Ontologies

- Ontology Language
- Top-level Ontology specializes
- Mid-level Ontology instantiates
- Domain Ontology
Layering of Ontologies in the SW: Actors & Responsibilities

- Ontology Language
- W3C
- Philosophers
- Domain Experts

Modes of Development

- **top-down**
  - define the most general concepts first and then specialize them

- **bottom-up**
  - define the most specific concepts and then organize them in more general classes

- **combination**
  - define the more salient concepts first and then generalize and specialize them
THESEUS MEDICO: Ontology Hierarchy

Representational Ontology (RDFS, OWL, …)

- time, space, organization, person, event

Information Element Ontology
(images, text documents, videos)
MPEG7

Clinical Ontologies
- clinic, doctor, nurse, patient
- medical case
- Electronic Health Record
- Diagnosis-Related Groups
- information from DICOM

Medical Ontologies
- FMA
- Disease Ontology
- Physiology Ontology

Annotation Ontology

The EPOS Ontology Space

Corporate Ontology Level

OMO1

OMO2

OMO3

Organizational Memory Ontology Level

PIM1

PIM2

PIM3

PIM4

PIM5

PIM6

PIM7

PIM8

PIM9

PIM10

PIM11

Personal Information Model Level

Native Structure Level

Inherit/Leverage
Task-oriented Mapping

Level of Formality & Sharing Scope
How Are Ontologies Applied?

- Preserve knowledge
- Provide views and navigation structures for manual browsing
- Facilitate natural language access
- Provide background knowledge for query expansion or query rewriting
- Enable management of non-textual media
- Support retrieval and integration of information from different, distributed sources

Ontology Languages/Representational Ontologies

- **Ontology Languages** (representational ontologies): define the *vocabulary* with which ontologies are represented

- **Neutral with respect to world entities**: provide a representational framework without making claims about the world

Further Characteristics of Representational Ontologies

- **Open/closed-world semantics**
  - How is "not knowing" handled?

- **(No) Unique-name assumption**
  - What can be inferred from distinct resource IDs?

- **Descriptive vs. prescriptive**
  - What is the role of schema information?
Further Characteristics of Representational Ontologies (1)

- **Open/closed-world semantics**
  - **Closed-world:**
    what is not currently known to be true is false
  - **Open-world:**
    if a statement cannot be inferred from what is expressed in the system, then it cannot be inferred to be false (often: unknown); needed in incomplete systems

Further Characteristics of Representational Ontologies (2)

(No) Unique-name assumption

- **UNA:**
  One object (concept or instance) has exactly one identifier.
  Individuals with different names are indeed different individuals.

- **No-UNA:**
  Each object may have multiple identifiers.
  - Suppose we state that each wine is produced by at most one winery, and that a given wine is produced by two wineries.
    - With No-UNA, a reasoner does not flag an error.
    - Instead it infers that the two winery resources are equal!
Further Characteristics of Representational Ontologies (3)

Descriptive vs. prescriptive

- **Prescriptive**: database schema-like, allows for checking if instances conform to ontology
  
  *Carlo the Cat is author of a book* => *Error!*

- **Descriptive**: schema information as additional description of resources
  
  *Carlo the Cat is author of a book* => *Carlo is also a person.*

W3C Developed Standards for Many AI KR Approaches

<table>
<thead>
<tr>
<th>Semantic Networks (BUT...)</th>
<th>Frame Language (BUT...)</th>
<th>KR Logic-Lite (BUT...)</th>
<th>Next up: OPS5-ish (BUT...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDF</td>
<td>RDF Schema</td>
<td>OWL</td>
<td>Sem Web Rules Language</td>
</tr>
</tbody>
</table>
Part II

Ontology Modeling in the Semantic Web

Three Levels of Markup on the Web

- Content: RDF/S, OWL
- Structure: XML
- Form: HTML

WWW document
HTML is a Language for Presenting Documents

HTML tags are presentation-oriented.

```
<em>Thomas R. Roth-Berghofer</em>
<hr>
Postfach 20 80
<hr>
67608 Kaiserlautern
```

**Example**

Focus: Presentation instead of structuring

fixed document type

---

XML Describes Logical Structure of a Document (and, thus, of its Content)

Focus: structure instead of presentation

Visual presentation can be defined using one or more templates (XSL stylesheets), independent of actual document

Own definition of elements: XML standard only defines basic syntactical structure such as

- which tags are allowed
- which attributes a tag can have
- etc.

Extended linking possibilities such as XLink and XPointer (compared with HTML)
XML is Content-oriented

XML for Knowledge Representation

- No data format itself, but it is a foundation for defining those
- Definition of self-describing data
- Standardized, open and license-free format (not proprietary)
  - Text is one of the few data types universally supported by all platforms
- Human readable (also important for long-term archiving!)
- Used by enterprises of different sectors for structured data and information exchange
- Allows (easier) integration of information coming from different sources into uniform documents
Semantic Web Layer Cake

- Language independence
- Uniform identifiers

Semantic Web Layer Cake

- Extensible Markup Language XML as exchange format and syntactical foundation
- Namespaces
- Basic validation
Semantic Web Layer Cake

- Resource Description Framework RDF: Data model
- RDF Schema: Types, classes / subclasses, inheritance

Increasing formalization
Increasing expressiveness
Semantic Web Layer Cake

- Axioms
- Rules

Semantic validation

FOIS 2008 Tutorial: Ontologies and the Semantic Web: Building Blocks and Challenges
Semantic Web Layer Cake

- Securing transport of knowledge

Semantic Web Layer Cake

- Web of Trust
Identification, URLs, URIs

Need of Identification …

- … arises in everyday situations
  - „Real“ world:
    - Bob
    - The moon
    - FOIS conference
    - The weather today
  - On the Web:
    - Uniform Resource Locator (URL) identifies web pages using access mechanisms such as protocols, domains, hierarchies („Location“ on the network)
What do we do With „Stuff“ not on the Web?

Issues:
- I do not have a URL
- My employer wants to pay my salary, e.g., by money transfer
- My family doctor wants to associate medical history records with me and my health insurance company

Solutions for everyday life:
- Social security number
- Tax payer’s account number
- Health insurance number

Identifiability on the Web: Uniform Resource Identifier

Uniform Resource Identifiers (URIs) can be generated:
- by anyone
- at any time

URIs can denote (thus, identify) anything, i.e., resources

Resources may be
- accessible on the Web
  - such as Web pages or documents
- not accessible in the Web
  - such as people, companies, or books
- abstract concepts
  - such as creator or publisher

Via http, https, ftp, ftps, gopher, mailto …
URIs

- URI Format: absolute URI with optional fragment identifier, e.g.: 
  http://www.example.com/kbs/kb1#name
- Abbreviated with XML namespace mechanism, e.g.: example:name
- URI with namespace only meant for disambiguation of elements and attribute names in an XML document (not: modularization)
- URI does not guarantee existence of a document (i.e., URI is not necessarily a URL)

Identity Problem

- Decentralized generation of names (URIs) allows for multiple assignments of different URIs to same resource
- In general there is no way to find out if two URIs denote the same resource
RDF – Resource Description Framework

RDF Introduction

- RDF consists of two parts:
  - RDF model
    - a set of triples / a graph
  - RDF syntax
    - different serializations (mainly, but not exclusively, in XML)
- RDF Schema: definition of vocabularies
  - simple ontologies
  - for RDF
  - and using RDF
RDF Data Model

- **Resources**
  - Any object that can be referred to is a resource
  - Resources have URIs

- **Examples:**
  - A web page, part of a web page, or a collection of web pages
  - Objects that are not accessible over the web directly

- **Properties (Slots)**
  - Define relations to other resources or to values (literals)

- RDF/S is very similar to semantic networks and frame/object-oriented languages (but: different kind of formal semantics)

---

**An Example**

http://www.w3.org/Home/Lassila

XML

```xml
<Creator>
  <uri>http://www.w3.org/Home/Lassila</uri>
  <name>Ora Lassila</name>
</Creator>
```

XML

```xml
<Document uri="http://www.w3.org/Home/Lassila"
  Creator="Ora Lassila"/>
```
RDF – A Simple Example

**Statement**
- “Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.”

**Structure**
- Resource (subject) http://www.w3.org/Home/Lassila
- Property (predicate) http://www.schema.org/#Creator
- Value (object) “Ora Lassila”

Directed graph with named edges

```
http://www.w3.org/Home/Lassila s:Creator Ora Lassila
```

RDF Syntax

- RDF data model does not enforce specific syntax
- Specification suggests several XML-based variants

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
    <s:createdWith rdf:resource="http://www.w3c.org/amaya"/>
  </rdf:Description>
</rdf:RDF>
```
Resulting Graph

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
    <s:createdWith rdf:resource="http://www.w3c.org/amaya"/>
  </rdf:Description>
</rdf:RDF>
```

Properties are Represented by URIs

```
http://www.w3.org/Home/Lassila
```

- Note:
  - Disambiguation / Discernability of own and other's predicates
  - Anything that has a URI is a resource. Thus, one can make statements about properties
Example RDF Vocabulary: "Dublin Core"

- Dublin Core Metadata Element Set is a standard for cross-domain information resource description
- Current Version: 1.1
- Consists of 15 main elements:
  - title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage and rights
- URL: [http://dublincore.org](http://dublincore.org)

Referencing

- Statement:
  - "The publisher of the website with URL [http://www.michael-sintek.de](http://www.michael-sintek.de) is Michael Sintek, whose email address is sintek@dfki.de."
  
```xml
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/"

    <rdf:Description about="http://www.michael-sintek.de">
        <dc:title>Homepage Michael Sintek</dc:title>
        <dc:publisher rdf:resource="mailto:sintek@dfki.de"/>
    </rdf:Description>

    <rdf:Description about="mailto:sintek@dfki.de">
        <dc:title>Michael Sintek</dc:title>
    </rdf:Description>
</rdf:RDF>
```
Reification:
Statements About Statements

- Example:
  - “Ralph Swick believes that Ora Lassila is the publisher of resource http://www.w3.org/Home/Lassila.”

- Assumption:
  - In order to make a statement about another statement, we need to transform that statement into a resource, i.e., make it addressable.

Example: Reification

- Statement:
  - “Ralph Swick believes that Ora Lassila is the publisher of http://www.w3.org/Home/Lassila.”
RDFS and OWL:
Ontology Languages for the Semantic Web

Characteristics of RDF/S
What does RDF Schema add to RDF?

• Defines **vocabulary** for RDF
• Organizes this vocabulary in a **typed hierarchy**
  • Class, subClassOf, type
  • Property, subPropertyOf
  • domain, range

---

Things RDF/S Can’t Do (1)

- **Local scope of properties**
  - `rdfs:range` defines the range of a property (e.g., eats) for all classes
  - In RDF Schema we cannot declare range restrictions that apply to some classes only
  - E.g., we cannot say that cows eat only plants, while other animals may eat meat, too
Things RDF/S Can’t Do (2)

- **Disjointness of classes**
  - Sometimes we wish to say that classes are disjoint (e.g., male and female)

- **Boolean combinations of classes**
  - Sometimes we wish to build new classes by combining other classes using union, intersection, and complement
  - E.g., person is the disjoint union of the classes male and female

Things RDF/S Can’t Do (3)

- **Cardinality restrictions**
  - E.g., a person has exactly two parents, a course is taught by at least one lecturer

- **Special characteristics of properties**
  - A property is the inverse of another property (like “eats” and “is eaten by”)
  - Transitive property (like “greater than”)
Characteristics of OWL

Tradeoff Between Expressive Power and Efficient Reasoning Support

- The richer the language is, the more inefficient the reasoning support becomes.
- Sometimes it crosses the border of non-computability.
- We need a compromise:
  - A language supported by reasonably efficient reasoners.
  - A language that can express large classes of ontologies and knowledge.
An OWL ontology may start with a collection of assertions for housekeeping purposes using the `owl:Ontology` element:

```xml
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

- `owl:imports` is a transitive property
- Imports and namespace declarations are very different ("importing" meaning vs. syntactical shorthands).
- Whether an import ontology is actually loaded is decided by each tool!

**Classes**

- Classes are defined using `owl:Class`
  - `owl:Class` is a subclass of `rdfs:Class`

- Disjointness is defined using `owl:disjointWith`:

```xml
<owl:Class rdf:about="#associateProfessor">
  <owl:disjointWith rdf:resource="#professor"/>
  <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```
Classes (2) Thing, Nothing, Equivalence

- **owl:Thing** is the most general class, which contains everything
- **owl:Nothing** is the empty class
- **owl:equivalentClass** defines equivalence of classes

```xml
<owl:Class rdf:ID="faculty">
  <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

Properties

- **In OWL there are two kinds of properties**
  - **Object properties**, which relate objects to other objects
    - e.g., `isTaughtBy`, `supervises`
  - **Data type properties**, which relate objects to datatype values
    - e.g., `phone`, `title`, `age`, etc.
Datatype Properties

- OWL makes use of XML Schema data types, using the layered architecture of the SW

```
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Object Properties

- User-defined data types

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
  <owl:domain rdf:resource="#course"/>
  <owl:range rdf:resource="#academicStaffMember"/>
  ...
</owl:ObjectProperty>
```
Inverse & Equivalent Properties

<owl:ObjectProperty rdf:ID="teaches">
  <rdfs:range rdf:resource="#course"/>
  <rdfs:domain rdf:resource="#academicStaffMember"/>
  <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="lecturesIn">
  <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>

Property Restrictions

- In OWL we can declare that the class C satisfies certain conditions
  - All instances of C satisfy the conditions
  - This is equivalent to saying that C is subclass of a class C', where C' collects all objects that satisfy the conditions
    - C' can remain anonymous

<owl:Class rdf:about="#redWine">
  <rdfs:subClassOf>
    <owl:Restriction>
      ...
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
Property Restrictions (2)

- A (restriction) class is achieved through an `owl:Restriction` element.
- This element contains an `owl:onProperty` element and one or more restriction declarations:
  - Cardinality restrictions
  - `owl:allValuesFrom` specifies universal quantification
  - `owl:hasValue` specifies a specific value
  - `owl:someValuesFrom` specifies existential quantification

Cardinality Restrictions

- We can specify minimum and maximum number using `owl:minCardinality` and `owl:maxCardinality`.
- It is possible to specify a precise number by using the same minimum and maximum number.
- For convenience, OWL offers also `owl:cardinality`.
Cardinality Restrictions (2)

```xml
<owl:Class rdf:about="#course">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

- For each course there exists at least one property defining by whom the course is taught

Special Properties

- **owl:TransitiveProperty** (transitive property)
  - E.g. “has better grade than”, “is ancestor of”

- **owl:SymmetricProperty** (symmetry)
  - E.g. “has same grade as”, “is sibling of”

- **owl:FunctionalProperty** defines a property that has at most one value for each object
  - E.g. “age”, “height”, “directSupervisor”

- **owl:InverseFunctionalProperty** defines a property for which two different objects cannot have the same value
  - E.g. “emailAddress”
Boolean Combinations

- We can combine classes using Boolean operations (union, intersection, complement)

```xml
<owl:Class rdf:about="#courseMember">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:complementOf rdf:resource="#staffMember"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Boolean Combinations (2)

```xml
<owl:Class rdf:ID="peopleAtUni">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#staffMember"/>
    <owl:Class rdf:about="#student"/>
  </owl:unionOf>
</owl:Class>
```

- The new class is not a subclass of the union, but rather equal to the union
  - We have stated an equivalence of classes
Declaring Instances

- Instances of classes are declared as in RDF:

  ```
  <rdf:Description rdf:ID="949352">
    <rdf:type rdf:resource="#academicStaffMember"/>
  </rdf:Description>
  ```

  Abbreviation:

  ```
  <academicStaffMember rdf:ID="949352">
    <uni:age rdf:datatype="&xsd;integer">39</uni:age>
  </academicStaffMember>
  ```

OWL Is a Layered Language

- **OWL Lite:**
  - Classification hierarchy
  - Simple constraints

- **OWL DL:**
  - Maximal expressiveness
  - While maintaining tractability
  - Standard formalization

- **OWL Full:**
  - Very high expressiveness
  - Loosing tractability
  - Non-standard formalization
  - All syntactic freedom of RDF (self-modifying)
OWL Full

- It uses all the OWL languages primitives
- It allows the combination of these primitives in arbitrary ways with RDF and RDF Schema
- OWL Full is fully upward-compatible with RDF, both syntactically and semantically
- OWL Full is so powerful that it is **undecidable**
  - No complete (or efficient) reasoning support

OWL DL

- **OWL DL** (Description Logic) is a sublanguage of **OWL Full** that restricts application of the constructors from OWL and RDF
  - Application of OWL’s constructors’ to each other is disallowed
  - Therefore it corresponds to a well studied description logic
- **OWL DL** permits efficient reasoning support
- But we lose full compatibility with RDF:
  - Not every RDF document is a legal OWL DL document.
  - Every legal OWL DL document is a legal RDF document.
OWL Lite

- An even further restriction limits OWL DL to a subset of the language constructors
  - E.g., OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality
- The advantage of this is a language that is easier to
  - grasp, for users
  - implement, for tool builders
- The disadvantage is **restricted expressivity**.
- (The original hope that it’s formal and computational properties are much simpler turned out to be wrong.)

---

**Overview: Language Primitives in the Different OWL Layers**

**OWL Light**
- (sub)classes, individuals
- (sub)properties, domain, range
- conjunction
- (in)equality
- cardinality 0/1
- datatypes
- inverse, transitive, symmetric
- hasValue
- someValuesFrom
- allValuesFrom

**RDF Schema**

**OWL DL**
- Negation
- Disjunction
- Full Cardinality
- Enumerated types

**OWL Full**
- Allow meta-classes etc.
Typical OWL Reasoning Services

- **Class membership**
  - We want to deduce whether an object is instance of a class.

- **Classification**
  - We want to deduce all the subclass relationships between the existing classes in the ontology.

- **Equivalence of classes**
  - We want to deduce whether two classes are equivalent, i.e., they have the same extension.

- **Consistency of a class**
  - We want to check that some class does not have necessarily an empty extension.

- **Consistency of the ontology**
  - We want to check that the ontology admits at least a model, i.e., there is at least a possibility to have an instantiation of the domain compatible with the ontology.

---

**OWL vs. RDF/S**
OWL Compatibility with RDF Schema

- All varieties of OWL use RDF for their syntax
- Instances are declared as in RDF, using RDF descriptions
- OWL constructors are specializations of their RDF counterparts

OWL Compatibility with RDF Schema (2)

- Semantic Web design aims at *downward compatibility* with corresponding reuse of software across the various layers
- The advantage of full downward compatibility for OWL is only achieved for OWL Full, at the cost of computational intractability
Semantics of RDF/S and OWL

- OWL (Lite and DL)
  - Well-known DL semantics, open-world

- RDF/S
  - Official semantics: descriptive, open-world
  - Also allowed according to RDF Primer: prescriptive (i.e., database schema like, allows instances to be checked whether they conform to the ontology)

Queries, Rules, Reasoning
Reasoning in the Semantic Web

Queries, Rules, Inferences

- Queries:
  - Mainly just select existing knowledge, cannot compute ("infer") new knowledge
  - No recursion or other ways to express loops
  - Query languages for
    - RDF/S: SPARQL
    - OWL: OWL-QL, SPARQL-DL

- Rules:
  - Can be understood as named queries which can be reused in other rules (recursion)
  - Rule languages for
    - RDF/S: F-Logic/TRIPLE/..., Jena rules
    - OWL: DLP, SWRL, DL-safe rules
Kinds of Rules

- Various kinds of rules:
  - Logical rules
    - Open world, declarative
  - Logic programming rules
    - Approximation of logical semantics with operational aspects
    - Builtins, often closed world, semi-declarative
  - Procedural rules
    - Production rules, event-condition-action rules
    - Executable operations

Example Rule: “Berlin is in the EU”

- Rules:
  - X capital_of Y → X located_in Y (r1)
  - X located_in Y ∧ Y located_in Z → X located_in Z (r2)

- Query:
  - X located_in Y ?
  - will (among other answers) derive

Berlin located_in EU:
  - (kb) Berlin capital_of Germany
  - (r1) Berlin located_in Germany
  - (kb) Germany located_in EU
  - (r2) Berlin located_in EU

---

FOIS 2008 Tutorial: Ontologies and the Semantic Web: Building Blocks and Challenges
SPARQL

- SPARQL: Query Language for RDF
- W3C Candidate Recommendation (6 April 2006)
- http://www.w3.org/TR/rdf-sparql-query/
- Goals:
  - extract information in the form of URIs, blank nodes, plain and typed literals
  - extract RDF subgraphs
  - construct new RDF graphs based on information in the queried graphs
- Accompanied by “SPARQL Protocol for RDF”
  - W3C Candidate Recommendation (6 April 2006)
  - http://www.w3.org/TR/rdf-sparql-protocol/

---

SPARQL: Sample Query

- Data:
  - <http://purl.org/dc/elements/1.1/title>
  - "SPARQL Tutorial".

- Query:
  - SELECT ?title WHERE
    - { <http://example.org/book/book1>
      <http://purl.org/dc/elements/1.1/title> ?title . }

- Query Result:
  - title
  - "SPARQL Tutorial"
Queries and Rules for OWL

OWL: Inferencing, Queries, and Rules

- Inferencing in OWL usually restricted to
  - Classification (class hierarchy)
  - Concept satisfiability / ontology consistency
  - Realization (where do individuals belong)
  - (Entailment)

- Why OWL is not enough
  - OWL is not expressive enough:
    - $\forall X \forall Y \forall Z \ (\text{brother}(Y, Z) \land \text{father}(X, Y) \Rightarrow \text{uncle}(X, Z))$ cannot be expressed in OWL
  - OWL is static (used for knowledge representation, not programming)
  - OWL (DL) is decidable and thus cannot be used for general programming
  - OWL is not executed (it is not procedural)
  - Extensions (built-ins) difficult to add
Adding Rules to OWL

- Rules as FOL extension of OWL (declarative)
  - Most prominent example: SWRL
- Rules on top of OWL (semi-declarative):
  - Accessing OWL knowledge bases from logic programs, e.g. by querying external OWL reasoners
- Rules and OWL side-by-side (semi-declarative):
  - Candidate: F-Logic
  - Interaction with OWL using common fragment

SWRL

- SWRL: A Semantic Web Rule Language Combining OWL and RuleML
- W3C Member Submission (21 May 2004)
- http://www.w3.org/Submission/SWRL/
- Based on a combination of OWL DL and OWL Lite sublanguages of OWL with unary/binary Datalog → undecidable!
- Has model-theoretic semantics to provide the formal meaning for OWL ontologies including rules
- Example in human-readable syntax:
  - parent(?x,?y) \land brother(?y,?z) \Rightarrow uncle(?x,?z)
Part III

Tools & Applications

Protégé – Frames Version
Protégé – OWL Version

Semantic Search

FOIS 2008 Tutorial: Ontologies and the Semantic Web: Building Blocks and Challenges
Who got a red card in the final?

Ontology-based Analysis of the Question

Multimodal Answer
NEPOMUK: Social Semantic Desktop

Part IV

Challenges
Motivation is still up-to-date!

Every ontology is a treaty – a social agreement – among people with some common motive in sharing

Ontological Knowledge should Traverse a Complete KM Cycle

- Ontology Identification
- Ontology Application
- Ontology Development
- Ontology Distribution
- Ontology Acquistion
- Local Embedding

<table>
<thead>
<tr>
<th>Application Goals</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utility Evaluation</td>
<td></td>
</tr>
</tbody>
</table>

(Scientific and practical) Challenges:
- Many building blocks ( ) cannot be handled locally
- Managing ontologies is not a first order business task
Challenges

- **Sharing**
  - Collaboration
  - Ontology/KB Mapping/Merging
  - Reusability
- **Stability**
  - Ontology Evolution
- **Formality**
  - Representational ontology
  - Ontology design

Sharing Challenge I: Collaboration and Mapping/Merging

- Sharing: often ignored in real systems
  - No good tool support yet
  - Collaborative editing
    - Only few first systems/prototypes exist (e.g., collaborative version of Protégé and Neon)
  - Ontology mapping/merging
    - Several frameworks exist as scientific prototypes
    - But: not yet ripe for real use
      - Too much manual work needed
      - Only offline so far
Sharing Challenge II: Reusability of Ontologies

- Reusability: so far very low!
  - Reasons: different representational/foundational ontologies, underlying design assumptions, scope

First (technical) solution approaches:
- Modularization with named graphs plus views
  - Named graphs: no official standards yet; weak support in existing systems
  - Views (on named graphs/modules)
    - Allow adaptation for applications (and also end users)
    - But: only scientific prototypes exist (NRL in NEPOMUK; TRIPLE)

Stability Challenge: Ontology Evolution

- Hardly any support in existing systems (ontology/knowledge base stores)
  - Exception: OWL API
- Requires explicit support of changes and versioning
- Dependencies between ontologies and applications building upon them usually hinders ontology evolution (esp. when several distributed and independent systems are involved)
Formality Challenge I: Selecting a Representational Ontology

- **Expressiveness of the language**
  - Are the primitives available which are adequate for modeling my domain?
  - Unique name assumption or not?
  - Closed world assumption or not?

- **Computational features**
  - Decidable, sound, complete, efficient … reasoning algorithms

- **Compatibility with rest of the world**

- **Tool support**
  - Ontology/KB store
  - Reasoning (efficient?)
  - Versioning, distributed editing, “understandable” GUI
  - …

Formality Challenge II: Ontology Design

- **Trade-off between well-designed ontologies, effort to create them, and understandability**

- **Common mistake in current systems: same ontology**
  - Should be the well-designed ontology
  - Is used by various system components
  - Should be understandable for various end user groups

- **Proposed solution approach:**
  - View concept
  - Allows every actor (both system components and users) to deal with/see an adapted version of the ontology/knowledge base
Thank you for your attention!