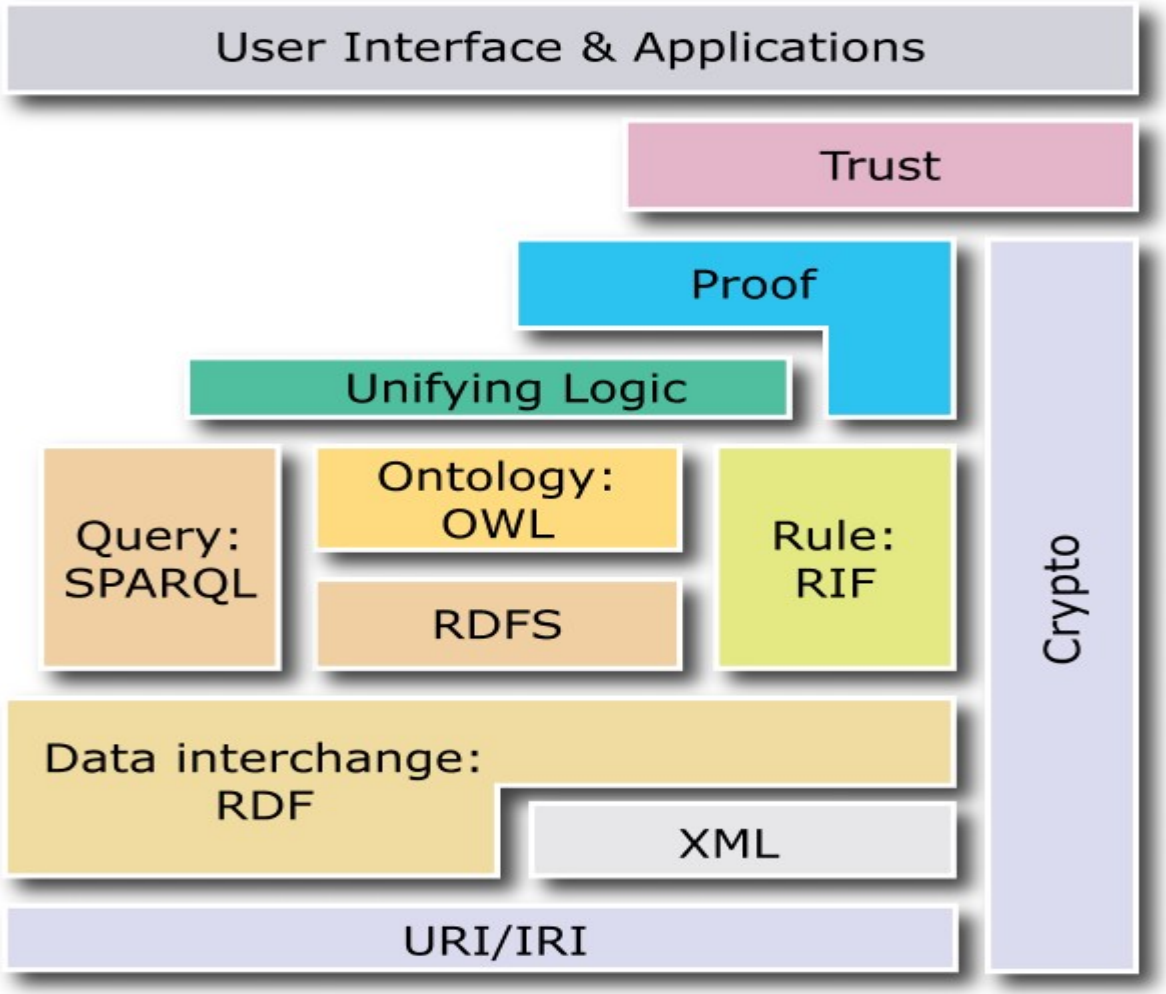
An Introduction to RDF Schema

Acknowledgement

- This presentation is based on the excellent RDF primer by the W3C available at http://www.w3.org/TR/rdf-primer/ and http://www.w3.org/2007/02/turtle/primer/ .

- Much of the material in this presentation is verbatim from the above Web site.

The Semantic Web “Layer Cake”



Important Assumption

- The following slides assume that you have a basic understanding of the concepts of **object**, **class** and **meta-class** as used in object-oriented formalisms (data models, programming languages etc.).

- If you do not, please read the introductory papers
  - Renate Motschnig-Pitrik, John Mylopoulos: Classes and Instances. Int. J. Cooperative Inf. Syst. 1(1): 61-92 (1992)
  - "Ontology Development 101: A Guide to Creating Your First Ontology" from http://ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html

RDF Schema

- RDF is a data model that provides a way to express simple **statements** about **resources**, using named **properties** and **values**.
- The **RDF Vocabulary Description Language (or RDF Schema or RDFS)** is a language that can be used to define the **vocabulary** (i.e., the **terms)** to be used in an RDF graph.
- The RDF Vocabulary Description Language is used to indicate that we are describing specific **kinds** or **classes** of resources, and will use specific **properties** in describing those resources.
- The RDF Vocabulary Description Language 1.0 is an **ontology definition language** (a **simple** one, compared with other languages such as OWL; **we can only define taxonomies and do some basic inference about them**).
- The RDF Vocabulary Description Language is like a **schema definition language** in the relational or object-oriented data models (hence the alternative name RDF Schema – we will use this name and its shorthand RDFS mostly!).

RDF Schema (cont'd)

- **The RDF Schema concepts are themselves provided in the form of an RDF vocabulary**; that is, as a specialized set of predefined RDF resources with their own special meanings.

- The resources in the RDF Schema vocabulary have URIrefs with the prefix `http://www.w3.org/2000/01/rdf-schema#` (associated with the QName prefix `rdfs:`).

- Vocabulary descriptions (schemas, ontologies) written in the RDF Schema language are **legal RDF graphs**. In other words, we use **RDF to represent RDFS information**.

Classes

- A basic step in any kind of description process is identifying the various **kinds of things** to be described. RDF Schema refers to these “kinds of things” as **classes**.

- A **class** in RDF Schema corresponds to the generic concept of a **type** or **category**, somewhat like the notion of a class in object-oriented programming languages such as Java or object-oriented data models.

Defining Classes

- Suppose an organization `example.org` wants to use RDF Schema to provide information about **motor vehicles, vans and trucks.**

- To define classes that represent these categories of vehicles, we write the following statements (triples):

```
ex:MotorVehicle rdf:type rdfs:Class .

ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
```

- **In RDFS, a class** `C` **is defined by a triple of the form**

```
C rdf:type rdfs:Class .
```

using the predefined class `rdfs:Class` **and the predefined property** `rdf:type`.

Defining Instances

- Now suppose `example.org` wants to define an individual car (e.g., the company car) and say that it is a motor vehicle.

- This can be done with the following RDF statement:

```
exthings:companyCar rdf:type ex:MotorVehicle .
```

rdf:type

- The predefined property `rdf:type` is used as a predicate in a statement

  `I rdf:type C`

  to declare that **individual** `I` **is an instance of class** `C`**.**

- In statements of the form

  `C rdf:type rdfs:Class .`

  `rdf:type` is used to declare that **class** `C` **(viewed as an individual object) is an instance of the predefined class** `rdfs:Class`.

Defining Classes (cont'd)

- **Defining a class explicitly is optional**; if we write the triple

  `I rdf:type C`

  then `C` is **inferred** to be a class (an instance of `rdfs:Class`) in RDFS.

Notation

- **Class names** will be written with an initial uppercase letter.

- **Property and instance names** are written with an initial lowercase letter.

Defining Subclasses

- Now suppose `example.org` wants to define that vans and trucks are **specialized kinds** of motor vehicle.
- This can be done with the following RDF statements:

```
ex:Van   rdfs:subClassOf   ex:MotorVehicle .
ex:Truck   rdfs:subClassOf   ex:MotorVehicle .
```

- The predefined property `rdfs:subclassOf` is used as a predicate in a statement to declare that **a class is a specialization of another more general class.**
- A class can be a **specialization of multiple superclasses** (e.g., the graph defined by the `rdfs:subclassOf` property is a **directed graph** not a tree).

Classes and Instances

- The **meaning** of the predefined property `rdfs:subClassOf` in a statement of the form

  `C1 rdfs:subClassOf C2`

  is that any instance of class `C1` is also an instance of class `C2`.

- **Example:** If we have the statements

  ```
  ex:Van   rdfs:subClassOf   ex:MotorVehicle .
  exthings:myCar rdf:type ex:Van .
  ```

  then RDFS allows us to **infer** the statement

  ```
  exthings:myCar rdf:type ex:MotorVehicle .
  ```

Properties of `rdfs:subClassOf`

- The `rdfs:subClassOf` property is **reflexive** and **transitive**.

- **Examples:**
  - If we have a class `ex:MotorVehicle` then RDFS allows us to **infer** the statement

    ```
    ex:MotorVehicle rdfs:subClassOf ex:MotorVehicle .
    ```

  - If we have the statements

    ```
    ex:Van rdfs:subClassOf ex:MotorVehicle .
    ex:MiniVan rdfs:subClassOf ex:Van .
    ```

    then RDFS allows us to **infer** the statement

    ```
    ex:MiniVan rdfs:subClassOf ex:MotorVehicle .
    ```

RDF Schema Predefined Classes

- The group of resources that are RDF Schema classes **is itself a class** called `rdfs:Class`. **All classes are instances of this class.**

- In the literature, classes such as `rdfs:Class` that have other classes as instances are called **meta-classes**.

- All things described by RDF are called **resources**, and are instances of the class `rdfs:Resource`.

- `rdfs:Resource` **is the class of everything. All other classes are subclasses of this class.** For example, `rdfs:Class` is a subclass of `rdfs:Resource`.

Class and Instance Information as a Graph

The Graph in Triple Notation

```
ex:MotorVehicle rdf:type rdfs:Class .
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .

ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .

ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle .
```

Properties

- In addition to defining the specific classes of things they want to describe, user communities also need to be able to define specific **properties** that characterize those classes of things (such as `author` to describe a book).

Defining Properties

- **A property can be defined by stating that it is an instance of the predefined class** `rdf:Property`.

- Example:

```
ex:author rdf:type rdf:Property .
```

- Then, property `ex:author` can be used as a **predicate in an RDF triple** such as the following:

```
ex:john ex:author ex:book123 .
```

Defining Properties (cont'd)

- **Defining a property explicitly is optional**; if we write the RDF triple

  `S P O .`

  then `P` is **inferred** to be a property by RDFS.

Properties (cont'd)

- **Properties are resources too** (this makes RDF and RDFS different than many other KR formalisms).
- **Therefore, properties can appear as subjects or objects of triples.**
- Example (provenance):

```
ex:author prov:definedBy ke:john
ke:john prov:defined ex:author
```

- We will see many more examples like the above.

Properties (cont'd)

- In RDFS property definitions are **independent** of class definitions. In other words, a property definition can be made without any reference to a class.

- Optionally, properties can be declared to apply to certain instances of classes by defining their **domain and range.**

Domain and Range

- RDFS provides vocabulary for describing **how properties and classes are intended to be used together** in RDF data.

- The `rdfs:domain` predicate can be used to indicate that a particular property applies to instances of a designated class (i.e., it defines the **domain** of the property).

- The `rdfs:range` predicate is used to indicate that the values of a particular property are instances of a designated class (i.e., it defines the **range** of the property).

Example

```
ex:Book rdf:type rdfs:Class .
ex:Person rdf:type rdfs:Class .

ex:author rdf:type rdf:Property .

ex:author rdfs:domain ex:Book .
ex:author rdfs:range ex:Person .
```

Domain and Range (cont'd)

- For a property, we can have **zero, one, or more than one** domain or range statements.

Domain and Range (cont'd)

- **No domain or no range statement**: If no range statement has been made for property `P`, then **nothing has been said about the values** of this property. Similarly for no domain statement.

- **Example:** If we have only the triple

```
exstaff:frank ex:hasMother exstaff:frances .
```

then nothing can be inferred from it regarding resources `exstaff:frank` and `exstaff:frances`.

Domain and Range (cont'd)

- **One domain statement:** If we have

  `P rdfs:domain D .`

  then we can **infer** that when `P` is applied to some resource, this resource is an instance of class `D`.

- **One range statement:** If we have

  `P rdfs:range R .`

  then we can **infer** that when `P` is applied to some resource, the value of `P` is an instance of class `R`.

Examples

- If we have

```
ex:hasMother rdfs:domain ex:Person .

exstaff:frank ex:hasMother exstaff:frances .
```

then we can infer:

```
exstaff:frank rdf:type ex:Person.
```

- If we have

```
ex:hasMother rdfs:range ex:Person .

exstaff:frank ex:hasMother exstaff:frances .
```

then we can infer

```
exstaff:frances rdf:type ex:Person.
```

Domain and Range (cont'd)

- **Two domain or range statements:** If we have

```
P rdfs:range C1 .
P rdfs:range C2 .
```

  then we can **infer** that the values of property `P` are instances of both `C1` and `C2`. Similarly, for two domain statements.

- **Example:** If we have

```
ex:hasMother rdfs:range ex:Female .
ex:hasMother rdfs:range ex:Person .
exstaff:frank ex:hasMother exstaff:frances .
```

  then we can infer that `exstaff:frances` is an instance of both `ex:Female` and `ex:Person`.

Another Example

```
ex:Human rdf:type rdfs:Class .
ex:hasParent rdf:type rdf:Property .
ex:hasParent rdfs:domain ex:Human .
ex:hasParent rdfs:range ex:Human .

ex:Tiger rdf:type rdfs:Class .
ex:hasParent rdfs:domain ex:Tiger .
ex:hasParent rdfs:range ex:Tiger .

ex:tina ex:hasParent ex:john .
```

What new triples can we infer from the above? Is anything wrong?

Another Example (cont'd)

- Intuitively: Tina and John are inferred to be both humans and tigers. There is nothing wrong with this! **We have not said anywhere that humans cannot be tigers and vice versa (nor can we say this in RDFS).**

- However, one might not want to do this kind of modeling in RDFS if Tina and John were meant to be humans.

Datatypes for Ranges

- The `rdfs:range` property can also be used to indicate that the value of a property is given by a **typed literal.**

- **Example:**

```
ex:age rdf:type rdf:Property .
ex:age rdfs:range xsd:integer .
```

- Optionally, we can also assert that `xsd:integer` is a **datatype** as follows:

```
xsd:integer rdf:type rdfs:Datatype .
```

An Example using Turtle Syntax

```
ex:registeredTo a rdf:Property;
   rdfs:domain ex:MotorVehicle;
   rdfs:range ex:Person.

ex:rearSeatLegRoom a rdf:Property;
   rdfs:domain ex:PassengerVehicle;
   rdfs:range xsd:integer.

ex:Person a rdfs:Class.

xsd:integer a rdfs:Datatype.
```

Specializing Properties

- RDF Schema provides a way to **specialize properties** (similarly with classes). This specialization relationship between two properties is described using the predefined property `rdfs:subPropertyOf`.

- **Example:**

```
ex:driver rdf:type rdf:Property .
ex:primaryDriver rdf:type rdf:Property .
ex:primaryDriver rdfs:subPropertyOf ex:driver .
```

Specializing Properties (cont'd)

- If resources `S` and `O` are connected by the property `P1` and `P1` is a subproperty of property `P2`, then RDFS allows us to **infer** that `S` and `O` are also connected by the property `P2`.

- **Example:** If we have the statements

```
ex:john ex:primaryDriver ex:companyCar
ex:primaryDriver rdfs:subPropertyOf ex:driver .
```

  then we can infer

```
ex:john ex:driver ex:companyCar .
```

Specializing Properties (cont'd)

- `rdfs:subPropertyOf` is **reflexive** and **transitive**.

- **Examples**:
  - If we have the property `ex:driver` then RDFS allows to infer the triple

    ```
    ex:driver rdfs:subPropertyOf ex:driver .
    ```

  - If we have the triples

    ```
    ex:primaryDriver rdfs:subPropertyOf ex:driver .
    ex:driver rdfs:subPropertyOf ex:isResponsibleFor .
    ```

    then RDFS allows us to **infer** the triple

    ```
    ex:primaryDriver rdfs:subPropertyOf ex:isResponsibleFor .
    ```

Important (Tricky?) Details

- **A class may be a member of its own class extension (i.e., an instance of itself).**

  Example: `rdfs:Class rdf:type rdfs:Class .`

- **A property may be applied to itself.**

  Example: `rdfs:domain rdfs:domain rdf:Property .`

- The semantics of RDF and RDFS are formalized appropriately so that we do not have problems with these features (details later in the course).

Some Utility Properties

- `rdfs:label`
- `rdfs:comment`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`

The Property `rdfs:label`

- `rdfs:label` is an instance of `rdf:Property` that may be used to provide a **human-readable version of a resource's name**.

- The `rdfs:domain` of `rdfs:label` is `rdfs:Resource`. The `rdfs:range` of `rdfs:label` is `rdfs:Literal`.

- Multilingual labels are supported using the **language tagging** facility of RDF literals.

The Property `rdfs:comment`

- `rdfs:comment` is an instance of `rdf:Property` that may be used to provide a **human-readable description of a resource.**

- The `rdfs:domain` of rdfs:comment is `rdfs:Resource`. The `rdfs:range` of `rdfs:comment` is `rdfs:Literal`.

- Multilingual documentation is supported through use of the **language tagging** facility of RDF literals.

The Property `rdfs:seeAlso`

- `rdfs:seeAlso` is an instance of `rdf:Property` that is used to indicate a resource that might provide additional information about the subject resource.

- A triple of the form `S rdfs:seeAlso O` states that the resource `O` may provide additional information about `S`. It may be possible to retrieve representations of `O` from the Web, but this is not required. When such representations may be retrieved, no constraints are placed on the format of those representations.

- The `rdfs:domain` of `rdfs:seeAlso` is `rdfs:Resource`. The `rdfs:range` of `rdfs:seeAlso` is `rdfs:Resource`.

The Property `rdfs:isDefinedBy`

- `rdfs:isDefinedBy` is an instance of `rdf:Property` that is used to indicate a resource defining the subject resource. This property may be used to indicate an RDF vocabulary in which a resource is described.

- A triple of the form `S rdfs:isDefinedBy O` states that the resource `O` defines `S`. It may be possible to retrieve representations of `O` from the Web, but this is not required. When such representations may be retrieved, no constraints are placed on the format of those representations. `rdfs:isDefinedBy` is a subproperty of `rdfs:seeAlso`.

- The `rdfs:domain` of `rdfs:isDefinedBy` is `rdfs:Resource`. The `rdfs:range` of `rdfs:isDefinedBy` is `rdfs:Resource`.

RDFS vs. Types in OO Languages and Data Models

- The **scope** of an attribute description in most programming languages is **restricted to the class or type** in which it is defined.

- In RDFS, on the other hand, property definitions are, by default, **independent** of class definitions, and have, by default, **global** scope (although they may optionally be declared to apply only to certain classes using domain and range specifications).

- Since they are resources, properties are **first-class citizens** in RDF. But notice the following **asymmetry**:
  - The class `rdf:Property` has as instances all properties (similarly with `rdfs:Class` which has as instances all classes).
  - There is no top class for the `rdfs:subPropertyOf` relationship e.g., `rdfs:topProperty` (unlike `rdfs:Resource` which is the top class for the `rdfs:subClassOf` relationship).

RDFS vs. Types (cont'd)

- **Benefits of the RDF approach:** One can start with a property definition and then extend it to other uses that might not have been anticipated.

- **Shortcoming:** In RDFS, it is not possible to say, for example, that if the property `ex:hasParent` is used to describe a resource of class `ex:Human`, then the range of the property is also a resource of class `ex:Human`, while if the property is used to describe a resource of class `ex:Tiger`, then the range of the property is also a resource of class `ex:Tiger`. **This can be done in ontology languages that we will define later in the course.**

RDFS vs. Types (cont'd)

- RDF Schema descriptions are not **prescriptive** in the way programming language type declarations typically are.

- **Example:** If a programming language declares a class `Book` with an `author` attribute having values of type `Person`, this is usually interpreted as a group of **constraints.**
  - The language will not allow the creation of an instance of `Book` without an `author` attribute.
  - The language will not allow an instance of `Book` with an `author` attribute that does not have a `Person` as its value.
  - If author is the **only** attribute defined for class `Book`, the language will not allow an instance of `Book` with some other attribute.

RDFS vs. Types (cont'd)

- RDF Schema provides schema information as additional **descriptions** of resources, but **does not prescribe how these descriptions should be used by an application.**

- RDF Schema only allows us to **infer new triples** as we specified earlier.

- Example:

```
ex:author rdf:type rdf:Property .
ex:author rdfs:range ex:Person .
```

- What can we infer?

RDFS vs. Types (cont'd)

- This schema information might be used in different ways by an application:
  - As a **constraint** in the same way that a programming language might: it will ensure that any `ex:author` property has a value of the `ex:Person` class. But this functionality needs to be developed by the application itself!
  - As **additional information** about the data it receives: if it receives some RDF data that includes an `ex:author` property whose value is a resource of unspecified class, it can use the schema-provided statement to **infer** that the resource must be an instance of class `ex:Person`. In this case, the inference functionality is offered by any RDF Schema implementation.

RDFS vs. Types (cont'd)

- Depending on how an application interprets the property descriptions, a description of an instance might be considered valid either **without** some of the schema-specified properties or with **additional** properties.

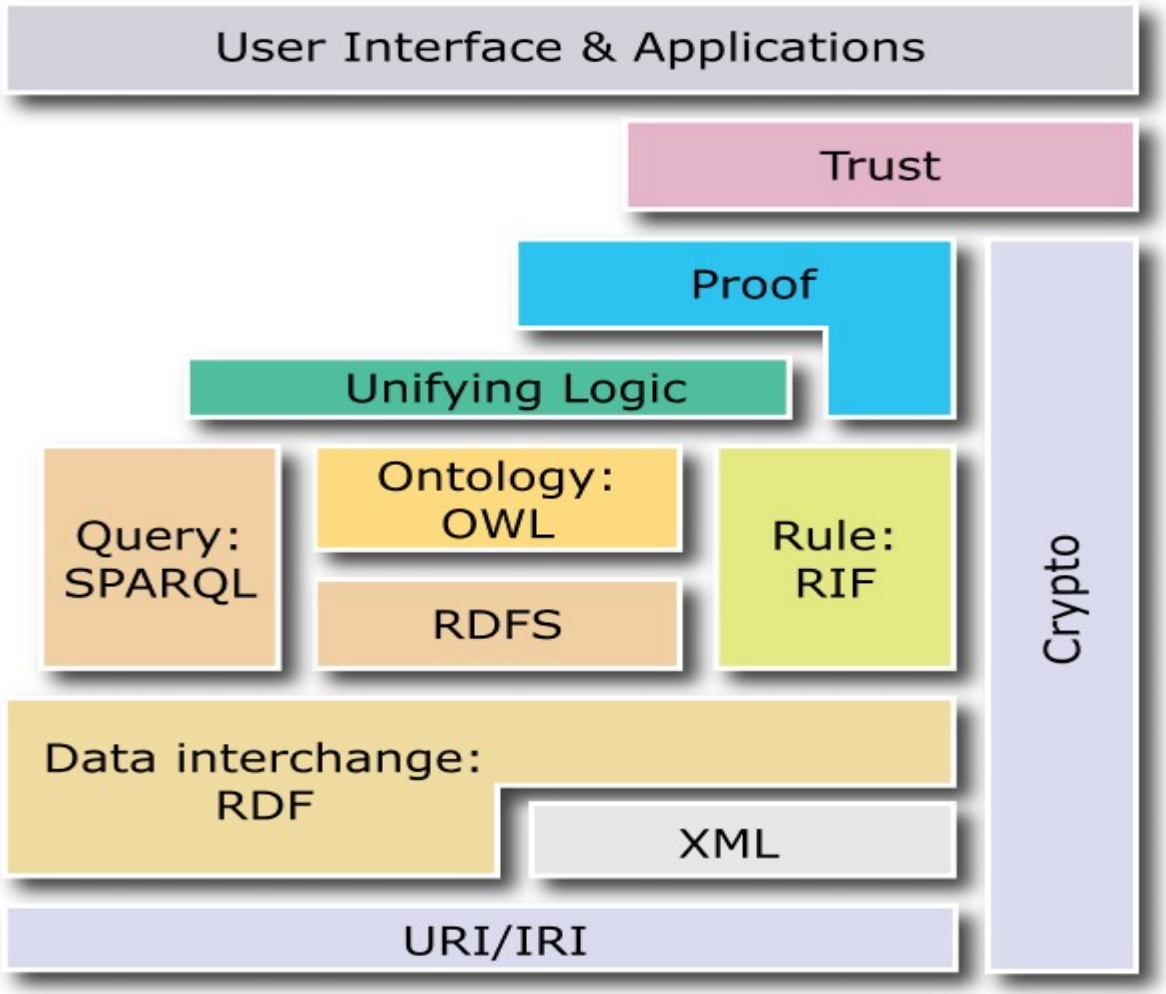
- **Examples:**
  - There might be an instance of `ex:Book` without an `ex:author` property, even if `ex:author` is described as having a domain of `ex:Book`.
  - There might be an instance of `ex:Book` with an `ex:technicalEditor` property, even though the schema describing class `ex:Book` does not describe such a property.

Richer Schema Languages

- RDF Schema provides **basic capabilities** for describing RDF vocabularies, but additional capabilities are also possible, and can be useful.

- These capabilities may be provided through **further development of RDF Schema,** or in **other languages** (for example, **ontology languages** such as OWL).

The Semantic Web “Layer Cake”



Richer Schema Languages (cont'd)

- **Richer schema capabilities** that have been identified as useful and are provided in ontology languages include:
  - **cardinality constraints** on properties, e.g., that a Person has ***exactly one*** biological father.
  - specifying that a given property (such as `ex:hasAncestor`) is **transitive**, e.g., that if `A ex:hasAncestor B`, and `B ex:hasAncestor C`, then `A ex:hasAncestor C`.
  - specifying that a given property is a unique identifier (or **key**) for instances of a particular class.
  - specifying that two different classes (having different URIrefs) actually represent the same class.
  - specifying that two different instances (having different URIrefs) actually represent the same individual.
  - specifying **constraints on the range or cardinality of a property that depend on the class to which a property is applied**, e.g., being able to say that for a soccer team the `ex:hasPlayers` property has 11 values, while for a basketball team the same property should have only 5 values.
  - the ability to describe **new classes in terms of combinations** (e.g., unions and intersections) of other classes, or to say that two classes are disjoint (i.e., that no resource is an instance of both classes).
  - The ability to specify **domain and range restrictions** for properties when they are used with a certain class.
  - …

Readings

- Chapter 2 of the book “Foundations of Semantic Web Technologies” or Chapter 3 of the Semantic Web Primer available from http://www.csd.uoc.gr/~hy566/SWbook.pdf .

- The following material from the W3C Semantic Web Activity Web page on RDF http://www.w3.org/RDF/ especially:
  - RDF 1.1 Primer.
  - RDF 1.1: Concepts and Abstract Syntax
  - RDF Schema 1.1

- Check out the content published at the RDF and RDFS namespace URIs:
  - http://www.w3.org/1999/02/22-rdf-syntax-ns#
  - http://www.w3.org/2000/01/rdf-schema#

  where you will find RDFS descriptions of the RDF and RDFS vocabularies given in RDF/XML!