CM40212: Internet Technology

Julian Padget
Tel: x6971, E-mail: jap@cs.bath.ac.uk

Semantic Web

Acknowledgements: Grigoris Antoniou and Frank van Harmelen (A Semantic Web Primer)

November 24, 2011
Content

1 Introduction

2 Ontology Engineering

3 RDF and RDFS (Antoniou & van Harmelen)

4 Processing RDF (Antoniou & van Harmelen)

5 XML reaction

6 Summary
Content

1 Introduction

2 Ontology Engineering

3 RDF and RDFS (Antoniou & van Harmelen)

4 Processing RDF (Antoniou & van Harmelen)

5 XML reaction

6 Summary
Web Services: Challenges?

1. Service description: interface? function?
2. Service advertisement: where?
3. Service discovery: matching + brokerage
4. Service contracts: governance
5. Service composition: workflow definition and re-use
6. Orchestration vs. Choreography
7. Provenance: automation of audit trails

Semantic issues stem from 1, 3 and 5. Need:

- Means to define concepts—ontologies
- Language to express ontologies
  - RDF
  - OWL(-S)
- Means to process semantic descriptions
Web Services: Challenges?

1. Service description: interface? function?
2. Service advertisement: where?
3. Service discovery: matching + brokerage
4. Service contracts: governance
5. Service composition: workflow definition and re-use
6. Orchestration vs. Choreography
7. Provenance: automation of audit trails

Semantic issues stem from 1, 3 and 5. Need:

1. Means to define concepts—ontologies
2. Language to express ontologies
   1. RDF
   2. OWL(-S)
3. Means to process semantic descriptions
Web Services: Challenges?

1. Service description: interface? function?
2. Service advertisement: where?
3. Service discovery: matching + brokerage
4. Service contracts: governance
5. Service composition: workflow definition and re-use
6. Orchestration vs. Choreography
7. Provenance: automation of audit trails

Semantic issues stem from 1, 3 and 5. Need:

1. Means to define concepts—ontologies
2. Language to express ontologies
   1. RDF
   2. OWL(-S)
3. Means to process semantic descriptions
Web Services: Challenges?

1. Service description: interface? function?
2. Service advertisement: where?
3. Service discovery: matching + brokerage
4. Service contracts: governance
5. Service composition: workflow definition and re-use
6. Orchestration vs. Choreography
7. Provenance: automation of audit trails

Semantic issues stem from 1, 3 and 5. Need:

1. Means to define concepts—ontologies
2. Language to express ontologies
   1. RDF
   2. OWL(-S)
3. Means to process semantic descriptions
Change in the Web

- Web content is mostly intended for human readers
- Mostly inaccessible to programs
- Keyword-based search engines have programmatic interfaces, but have limitations:
  - High recall, low precision
  - Low or no recall
  - Results are highly sensitive to vocabulary
  - Results are single Web pages
  - Human involvement is necessary to interpret and combine results
  - Results of Web searches are not readily accessible by other software tools
Change in the Web

- Web content is mostly intended for human readers
- Mostly inaccessible to programs
  - Keyword-based search engines have programmatic interfaces, but have limitations:
    - High recall, low precision
    - Low or no recall
    - Results are highly sensitive to vocabulary
    - Results are single Web pages
    - Human involvement is necessary to interpret and combine results
    - Results of Web searches are not readily accessible by other software tools
Change in the Web

- Web content is mostly intended for human readers
- Mostly inaccessible to programs
- Keyword-based search engines have programmatic interfaces, but have limitations:
  - High recall, low precision
  - Low or no recall
  - Results are highly sensitive to vocabulary
  - Results are single Web pages
  - Human involvement is necessary to interpret and combine results
- Results of Web searches are not readily accessible by other software tools
From Web to Semantic Web

- The meaning of web content is not machine accessible: lack of semantics
- It is simply difficult, for a machine, to distinguish between different meanings:
  - I am a lecturer of computer science.
  - I am an assistant professor of computer science
- Step 1: Represent web content in a form that is more easily machine-processable
- Step 2: Use intelligent techniques to take advantage of these representations
- The Semantic Web will gradually evolve out of the current Web and co-exist with it
- Foundation of semantics is the establishment of ontologies
From Web to Semantic Web

- The meaning of web content is not machine accessible: lack of semantics
- It is simply difficult, for a machine, to distinguish between different meanings:
  - I am a lecturer of computer science.
  - I am an assistant professor of computer science
- Step 1: Represent web content in a form that is more easily machine-processable
- Step 2: Use intelligent techniques to take advantage of these representations
- The Semantic Web will gradually evolve out of the current Web and co-exist with it
- Foundation of semantics is the establishment of ontologies
From Web to Semantic Web

- The meaning of web content is not machine accessible: lack of semantics
- It is simply difficult, for a machine, to distinguish between different meanings:
  - I am a lecturer of computer science.
  - I am an assistant professor of computer science
- **Step 1:** Represent web content in a form that is more easily machine-processable
- **Step 2:** Use intelligent techniques to take advantage of these representations
- The Semantic Web will gradually evolve out of the current Web and co-exist with it
- Foundation of semantics is the establishment of ontologies
From Web to Semantic Web

- The meaning of web content is not machine accessible: lack of semantics
- It is simply difficult, for a machine, to distinguish between different meanings:
  - I am a lecturer of computer science.
  - I am an assistant professor of computer science
- **Step 1:** Represent web content in a form that is more easily machine-processable
- **Step 2:** Use intelligent techniques to take advantage of these representations
- The Semantic Web will gradually evolve out of the current Web and co-exist with it
- Foundation of semantics is the establishment of ontologies
From Web to Semantic Web

- The meaning of web content is not machine accessible: lack of semantics
- It is simply difficult, for a machine, to distinguish between different meanings:
  - I am a lecturer of computer science.
  - I am an assistant professor of computer science
- **Step 1:** Represent web content in a form that is more easily machine-processable
- **Step 2:** Use intelligent techniques to take advantage of these representations
- The Semantic Web will gradually evolve out of the current Web and co-exist with it
- Foundation of semantics is the establishment of ontologies
From Web to Semantic Web

- The meaning of web content is not machine accessible: lack of semantics
- It is simply difficult, for a machine, to distinguish between different meanings:
  - I am a lecturer of computer science.
  - I am an assistant professor of computer science
- **Step 1:** Represent web content in a form that is more easily machine-processable
- **Step 2:** Use intelligent techniques to take advantage of these representations
- The Semantic Web will gradually evolve out of the current Web and co-exist with it
- Foundation of semantics is the establishment of ontologies

The meaning of web content is not machine accessible: lack of semantics.

It is simply difficult, for a machine, to distinguish between different meanings:

- I am a lecturer of computer science.
- I am an assistant professor of computer science

**Step 1:** Represent web content in a form that is more easily machine-processable

**Step 2:** Use intelligent techniques to take advantage of these representations

The Semantic Web will gradually evolve out of the current Web and co-exist with it

Foundation of semantics is the establishment of ontologies
Need for Ontologies

In order to be able to describe and discover web services, we need a common set of terms.

- An ontology is a formal specification of a set of terms.

Need for Ontologies

In order to be able to describe and discover web services, we need a common set of terms.

- An ontology is a formal specification of a set of terms.

What is an ontology?

Depends on subject and use, but common features are:

- A formal description of (the relevant parts) of a domain: "the nature of things, and the relationships between them"
- A set of classes (concepts) and their hierarchical relations
- A set of properties (slots or roles), defining arbitrary relations
- The same property may be ascribed to several independent classes
- Constraints — restrictions on properties (type, number)
- Individuals — some concrete instances of classes
What is an ontology?

Depends on subject and use, but common features are:

- A formal description of (the relevant parts) of a domain: "the nature of things, and the relationships between them"
- A set of classes (concepts) and their hierarchical relations
- A set of properties (slots or roles), defining arbitrary relations
- The same property may be ascribed to several independent classes
- Constraints — restrictions on properties (type, number)
- Individuals — some concrete instances of classes
What is an ontology?

Depends on subject and use, but common features are:

- A formal description of (the relevant parts) of a domain: "the nature of things, and the relationships between them"
- A set of classes (concepts) and their hierarchical relations
- A set of properties (slots or roles), defining arbitrary relations
- The same property may be ascribed to several independent classes
- **Constraints** — restrictions on properties (type, number)
- **Individuals** — some concrete instances of classes
What is an ontology?

Depends on subject and use, but common features are:

- A formal description of (the relevant parts) of a domain: “the nature of things, and the relationships between them”

- A set of classes (concepts) and their hierarchical relations

- A set of properties (slots or roles), defining arbitrary relations

- The same property may be ascribed to several independent classes

- Constraints — restrictions on properties (type, number)

- Individuals — some concrete instances of classes
What is an ontology?

Depends on subject and use, but common features are:

- A formal description of (the relevant parts) of a domain: "the nature of things, and the relationships between them"

- A set of classes (concepts) and their hierarchical relations

- A set of properties (slots or roles), defining arbitrary relations

- The same property may be ascribed to several independent classes

- **Constraints** — restrictions on properties (type, number)

- **Individuals** — some concrete instances of classes
What is an ontology?

Depends on subject and use, but common features are:

- A formal description of (the relevant parts) of a domain: “the nature of things, and the relationships between them”
- A set of classes (concepts) and their hierarchical relations
- A set of properties (slots or roles), defining arbitrary relations
- The same property may be ascribed to several independent classes
- **Constraints** — restrictions on properties (type, number)
- **Individuals** — some concrete instances of classes
Example ontology

- Wordnet is a live domain-neutral ontology: http://wordnetweb.princeton.edu/perl/webwn
- Words are nodes in a network of relationships, for example:
  - hyponym: more specialized concepts
  - meronym: parts of this concept
  - hypernym: generalized concept
  - holonym: part of something larger
  - etc.
Example ontology

- Wordnet is a live domain-neutral ontology: 
  http://wordnetweb.princeton.edu/perl/webwn
- Words are nodes in a network of relationships, for example:
  - hyponym: more specialized concepts
  - meronym: parts of this concept
  - hypernym: generalized concept
  - holonym: part of something larger
  - etc.
Kinds of ontology

- Lightweight → heavyweight
  1. Controlled vocabularies
  2. Glossaries
  3. Thesauri
  4. Informal Is-a hierarchy
  5. Formal Is-a hierarchy
  6. Formal instances
  7. Frames
  8. Value restriction
  9. General logic constraints
What Is “Ontology Engineering”?

Ontology Engineering: Defining terms in the domain and relations among them

- Defining concepts in the domain (classes)
- Arranging the concepts in a hierarchy (subclass-superclass hierarchy)
- Defining which attributes and properties (slots) classes can have and constraints on their values
- Defining individuals and filling in slot values
What is “Ontology Engineering”? 

Ontology Engineering: Defining terms in the domain and relations among them

- Defining concepts in the domain (classes)
- Arranging the concepts in a hierarchy (subclass-superclass hierarchy)
- Defining which attributes and properties (slots) classes can have and constraints on their values
- Defining individuals and filling in slot values
What Is “Ontology Engineering”? 

Ontology Engineering: Defining terms in the domain and relations among them

- Defining concepts in the domain (classes)
- Arranging the concepts in a hierarchy (subclass-superclass hierarchy)
- Defining which attributes and properties (slots) classes can have and constraints on their values
- Defining individuals and filling in slot values
What Is “Ontology Engineering”?

Ontology Engineering: Defining terms in the domain and relations among them

- Defining concepts in the domain (classes)
- Arranging the concepts in a hierarchy (subclass-superclass hierarchy)
- Defining which attributes and properties (slots) classes can have and constraints on their values
- Defining individuals and filling in slot values
Why use an ontology?

- To share common understanding of the structure of information among people or software components
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge (through ontology construction)

However:

Ontologies do not usually succeed in being application independent and often require adaptation for use in a new application.
Why use an ontology?

- To share common understanding of the structure of information among people or software components
- To enable reuse of domain knowledge
  - To make domain assumptions explicit
  - To separate domain knowledge from the operational knowledge
  - To analyze domain knowledge (through ontology construction)
- However:
  
  "Ontologies do not usually succeed in being application independent and often require adaptation for use in a new application."
Why use an ontology?

- To share common understanding of the structure of information among people or software components
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge (through ontology construction)

However:

Ontologies do not usually succeed in being application independent and often require adaptation for use in a new application.
Why use an ontology?

- To share common understanding of the structure of information among people or software components
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge (through ontology construction)

However:

Ontologies do not usually succeed in being application independent and often require adaptation for use in a new application.
Why use an ontology?

- To share common understanding of the structure of information among people or software components
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge (through ontology construction)

However:

Ontologies do not usually succeed in being application independent and often require adaptation for use in a new application.
Why use an ontology?

- To share common understanding of the structure of information among people or software components
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge (through ontology construction)

However:

Ontologies do not usually succeed in being application independent and often require adaptation for use in a new application.
Why use an ontology?

- To share common understanding of the structure of information among people or software components
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge (through ontology construction)

However:

*Ontologies do not usually succeed in being application independent and often require adaptation for use in a new application.*
Ontology-Development Process

Ideally:
- determine scope
- consider reuse
- enumerate terms
- define classes
- define properties
- define constraints
- create instances

In reality — an iterative process with feedback between succeeding phases.
Ontology Engineering versus Object-Oriented Modelling

An ontology:
- Reflects the structure of the world
- Is often about structure of concepts
- Actual physical representation is not an issue

An OO class structure:
- Reflects the structure of the data and code
- Is usually about behaviour (methods)
- Describes the physical representation of data (long int, char, etc.)
Ontology Engineering versus Object-Oriented Modelling

An ontology:
- Reflects the structure of the world
- Is often about structure of concepts
- Actual physical representation is not an issue

An OO class structure:
- Reflects the structure of the data and code
- Is usually about behaviour (methods)
- Describes the physical representation of data (long int, char, etc.)
Ontology Engineering versus Object-Oriented Modelling

An ontology:
- Reflects the structure of the world
- Is often about structure of concepts
- Actual physical representation is not an issue

An OO class structure:
- Reflects the structure of the data and code
- Is usually about behaviour (methods)
- Describes the physical representation of data (long int, char, etc.)
Ontology Engineering versus Object-Oriented Modelling

An ontology:
- Reflects the structure of the world
- Is often about structure of concepts
- Actual physical representation is not an issue

An OO class structure:
- Reflects the structure of the data and code
- Is usually about behaviour (methods)
- Describes the physical representation of data (long int, char, etc.)
Ontology Engineering versus Object-Oriented Modelling

An ontology:
- Reflects the structure of the world
- Is often about structure of concepts
- Actual physical representation is not an issue

An OO class structure:
- Reflects the structure of the data and code
- Is usually about behaviour (methods)
- Describes the physical representation of data (long int, char, etc.)
Ontology Engineering versus Object-Oriented Modelling

An ontology:
- Reflects the structure of the world
- Is often about structure of concepts
- Actual physical representation is not an issue

An OO class structure:
- Reflects the structure of the data and code
- Is usually about behaviour (methods)
- Describes the physical representation of data (long int, char, etc.)
Class Inheritance

- Classes usually constitute a **taxonomic hierarchy** (a subclass-superclass hierarchy)
  
- A class hierarchy is usually an IS-A hierarchy:
  
  - An instance of a subclass is an instance of a superclass
  
- If you think of a class as a set of elements, a subclass is a subset
Class Inheritance

- Classes usually constitute a **taxonomic hierarchy** (a subclass-superclass hierarchy)
- A class hierarchy is usually an IS-A hierarchy:
  - An instance of a subclass is an instance of a superclass
- If you think of a class as a set of elements, a subclass is a subset
Class Inheritance

- Classes usually constitute a **taxonomic hierarchy** (a subclass-superclass hierarchy)
- A class hierarchy is usually an IS-A hierarchy:
  - An instance of a subclass is an instance of a superclass
- If you think of a class as a set of elements, a subclass is a subset
Defining Slots and Properties

- Slots in a class definition describe attributes of instances of the class and relations to other instances.

- Each unit has level, pre-requisites, lecturer, etc.

- Types of properties:
  - **intrinsic** properties: learning objectives,
  - **extrinsic** properties: unit title and code
  - **parts**: content, coursework
  - **objects**: lecturer

- Simple and complex properties:
  - simple properties (attributes): contain primitive values (strings, numbers)
  - complex properties: contain (or point to) other objects (e.g., a manufacturer)
Defining Slots and Properties

- Slots in a class definition describe attributes of instances of the class and relations to other instances.
- Each unit has level, pre-requisites, lecturer, etc.

Types of properties:

- **intrinsic** properties: learning objectives,
- **extrinsic** properties: unit title and code
- **parts**: content, coursework
- **objects**: lecturer

Simple and complex properties:

- **simple properties** (attributes): contain primitive values (strings, numbers)
- **complex properties**: contain (or point to) other objects (e.g., a manufacturer)
Defining Slots and Properties

- Slots in a class definition describe attributes of instances of the class and relations to other instances.
- Each unit has level, pre-requisites, lecturer, etc.
- Types of properties:
  - intrinsic properties: learning objectives,
  - extrinsic properties: unit title and code
  - parts: content, coursework
  - objects: lecturer
- Simple and complex properties:
  - simple properties (attributes): contain primitive values (strings, numbers)
  - complex properties: contain (or point to) other objects (e.g., a manufacturer)
Defining Slots and Properties

- Slots in a class definition describe attributes of instances of the class and relations to other instances.
- Each unit has level, pre-requisites, lecturer, etc.

Types of properties:

- **intrinsic** properties: learning objectives,
- **extrinsic** properties: unit title and code
- **parts**: content, coursework
- **objects**: lecturer

Simple and complex properties:

- **simple properties** (attributes): contain primitive values (strings, numbers)
- **complex properties**: contain (or point to) other objects (e.g., a manufacturer)
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has *multiple* superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has multiple superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
  - Allow miscellaneous further uses
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code.*

- If a class has **multiple** superclasses, it inherits slots from all of them: *Some units are available to more than one program.*

- Domain of a slot: the class (or classes) of instances that can have the slot

- Range of a slot: the class (or classes) to which slot values belong

- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range

- Inverse slots

  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has *multiple* superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: If a unit has a title and a code, a Masters level unit also has a title and a code
- If a class has multiple superclasses, it inherits slots from all of them: Some units are available to more than one program
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has **multiple** superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has multiple superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has *multiple* superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but
  - Allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has **multiple** superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but
  - Allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has multiple superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Slot and Class Inheritance

- A subclass inherits all the slots from the superclass: *If a unit has a title and a code, a Masters level unit also has a title and a code*
- If a class has **multiple** superclasses, it inherits slots from all of them: *Some units are available to more than one program*
- Domain of a slot: the class (or classes) of instances that can have the slot
- Range of a slot: the class (or classes) to which slot values belong
- Property constraints (facets) describe or limit the set of possible values for a slot: type, cardinality, range
- Inverse slots
  - Example: Unit and Lecturer are inverse slots
  - Inverse slots contain redundant information, but
  - Allow acquisition of the information in either direction
  - Enable additional verification
  - Allow presentation of information in both directions
Defining Classes and a Class Hierarchy

- There is no single correct class hierarchy
- But there are some guidelines... The question to ask is: "Is each instance of the subclass an instance of its superclass?"

- Multiple Inheritance
  - A class can have more than one superclass
  - A subclass inherits slots and facet restrictions from all the parents
  - Different systems resolve conflicts differently

- Disjoint Classes
  - Classes are disjoint if they cannot have common instances
  - Disjoint classes cannot have any common subclasses either
Defining Classes and a Class Hierarchy

- There is no single correct class hierarchy
- But there are some guidelines... The question to ask is:
  "Is each instance of the subclass an instance of its superclass?"

Multiple Inheritance

- A class can have more than one superclass
- A subclass inherits slots and facet restrictions from all the parents
- Different systems resolve conflicts differently

Disjoint Classes

- Classes are disjoint if they cannot have common instances
- Disjoint classes cannot have any common subclasses either
Defining Classes and a Class Hierarchy

- There is no single correct class hierarchy
- But there are some guidelines... The question to ask is:
  
  "*Is each instance of the subclass an instance of its superclass?*"

- Multiple Inheritance
  - A class can have more than one superclass
  - A subclass inherits slots and facet restrictions from all the parents
  - Different systems resolve conflicts differently

- Disjoint Classes
  - Classes are disjoint if they cannot have common instances
  - Disjoint classes cannot have any common subclasses either
Defining Classes and a Class Hierarchy

- There is no single correct class hierarchy
- But there are some guidelines... The question to ask is: "Is each instance of the subclass an instance of its superclass?"

- Multiple Inheritance
  - A class can have more than one superclass
  - A subclass inherits slots and facet restrictions from all the parents
  - Different systems resolve conflicts differently

- Disjoint Classes
  - Classes are disjoint if they cannot have common instances
  - Disjoint classes cannot have any common subclasses either
Defining Classes and a Class Hierarchy

- There is no single correct class hierarchy
- But there are some guidelines... The question to ask is: "Is each instance of the subclass an instance of its superclass?"

- Multiple Inheritance
  - A class can have more than one superclass
  - A subclass inherits slots and facet restrictions from all the parents
  - Different systems resolve conflicts differently

- Disjoint Classes
  - Classes are disjoint if they cannot have common instances
  - Disjoint classes cannot have any common subclasses either
Limiting the Scope

- An ontology should not contain all the possible information about the domain.
- No need to specialize or generalize more than the application requires.
- No need to include all possible properties of a class.
  - Only the most salient properties.
  - Only the properties that the applications require.
- Aim: to model precisely enough.
Limiting the Scope

- An ontology should not contain all the possible information about the domain
- No need to specialize or generalize more than the application requires
- No need to include all possible properties of a class
  - Only the most salient properties
  - Only the properties that the applications require
- Aim: to model *precisely* enough
Limiting the Scope

- An ontology should not contain all the possible information about the domain.
- No need to specialize or generalize more than the application requires.
- No need to include all possible properties of a class.
  - Only the most salient properties.
  - Only the properties that the applications require.

- Aim: to model *precisely* enough.
Limiting the Scope

- An ontology should not contain all the possible information about the domain
- No need to specialize or generalize more than the application requires
- No need to include all possible properties of a class
  - Only the most salient properties
  - Only the properties that the applications require
- Aim: to model *precisely* enough
Content

1. Introduction

2. Ontology Engineering

3. RDF and RDFS (Antoniou & van Harmelen)

4. Processing RDF (Antoniou & van Harmelen)

5. XML reaction

6. Summary
Drawbacks of XML

- XML is a universal metalanguage for defining markup.
- Provides a uniform framework for interchange of data and metadata between applications.
- But no way to describe the semantics (meaning) of data.
- No meaning associated with the nesting of tags: each application interprets structure for itself.

Example representation of Julian Padget lectures Internet Technology:

```xml
<course name="Internet Technology">
  <lecturer>Julian Padget</lecturer>
</course>

<lecturer name="Julian Padget">
  <teaches>Internet Technology</teaches>
</lecturer>
```

- Opposite nesting, but same information.
Basic Ideas of RDF

- Basic building block: **object-attribute-value** triple
  - Called a *statement*
  - Sentence about me is such a statement
- An XML syntax had been defined for RDF
  - Inherits benefits of XML
  - Other syntactic representations of RDF possible
- The fundamental concepts of RDF are:
  - resources
  - properties
  - statements
- Allows us to express the relationships:
Basic Ideas of RDF

- Basic building block: **object-attribute-value** triple
  - Called a *statement*
  - Sentence about me is such a statement

- An XML syntax had been defined for RDF
  - Inherits benefits of XML
  - Other syntactic representations of RDF possible

- The fundamental concepts of RDF are:
  - resources
  - properties
  - statements

- Allows us to express the relationships:
Basic Ideas of RDF

- Basic building block: object-attribute-value triple
  - Called a *statement*
  - Sentence about me is such a statement
- An XML syntax had been defined for RDF
  - Inherits benefits of XML
  - Other syntactic representations of RDF possible
- The fundamental concepts of RDF are:
  - resources
  - properties
  - statements

- Allows us to express the relationships:

```
Internet Technology

lecturer

Julian Padget

```

```
teaches

```
Basic Ideas of RDF

- Basic building block: **object-attribute-value** triple
  - Called a *statement*
  - Sentence about me is such a statement
- An XML syntax had been defined for RDF
  - Inherits benefits of XML
  - Other syntactic representations of RDF possible
- The fundamental concepts of RDF are:
  - resources
  - properties
  - statements
- Allows us to express the relationships:

![Diagram](image-url)

- Internet Technology
- Julian Padget
- Teaches
- Lecturer
Resources

- A resource as an object, a “thing” we want to talk about:
  - For example: authors, books, publishers, places, people, hotels
- Building block comes from web technology
- Every resource has a URI, a Universal Resource Identifier
- A URI can be:
  - A URL (Web address) or
  - some other kind of unique identifier
Resources

- A resource as an object, a “thing” we want to talk about:
  - For example: authors, books, publishers, places, people, hotels

- Building block comes from web technology

- Every resource has a URI, a Universal Resource Identifier

- A URI can be:
  - A URL (Web address) or
  - some other kind of unique identifier
Resources

- A resource as an object, a “thing” we want to talk about:
  - For example: authors, books, publishers, places, people, hotels
- Building block comes from web technology
- Every resource has a URI, a Universal Resource Identifier
  - A URI can be:
    - A URL (Web address) or
    - some other kind of unique identifier
Resources

- A resource as an object, a “thing” we want to talk about:
  - For example: authors, books, publishers, places, people, hotels

- Building block comes from web technology

- Every resource has a URI, a Universal Resource Identifier

- A URI can be:
  - A URL (Web address) or
  - some other kind of unique identifier
Properties

- Properties are a special kind of resources
- Used to describe relations between resources
  - For example: “written by”, “age”, “title”, etc.
- Properties also identified by URIs
- Advantages of using URIs:
  - Global, worldwide, unique naming scheme
  - Reduces the homonym problem for distributed data representation
Properties

- Properties are a special kind of resources
  - Used to describe relations between resources
    - For example: “written by”, “age”, “title”, etc.
  - Properties also identified by URIs
- Advantages of using URIs:
  - Global, worldwide, unique naming scheme
  - Reduces the homonym problem for distributed data representation
Properties

- Properties are a special kind of resources
- Used to describe relations between resources
  - For example: “written by”, “age”, “title”, etc.
- Properties also identified by URIs

Advantages of using URIs:
- Global, worldwide, unique naming scheme
- Reduces the homonym problem for distributed data representation
Properties

- Properties are a special kind of resources
- Used to describe relations between resources
  - For example: “written by”, “age”, “title”, etc.
- Properties also identified by URIs
- Advantages of using URIs:
  - Global, worldwide, unique naming scheme
  - Reduces the homonym problem for distributed data representation
Statements

- Statements assert the properties of resources
- A statement is an object-attribute-value triple comprising:
  - a resource
  - a property, and
  - a value
- Literals are atomic values (strings)
- Values can be resources or literals
Statements

- Statements assert the properties of resources
- A statement is an object-attribute-value triple comprising:
  - a resource
  - a property, and
  - a value

- Literals are atomic values (strings)
- Values can be resources or literals
Statements

- Statements assert the properties of resources
- A statement is an object-attribute-value triple comprising:
  - a resource
  - a property, and
  - a value
- Literals are atomic values (strings)
- Values can be resources or literals
Statements

- Statements assert the properties of resources
- A statement is an object-attribute-value triple comprising:
  - a resource
  - a property, and
  - a value
- Literals are atomic values (strings)
- Values can be resources or literals
Three Views of a Statement

- A statement can be:
  - A triple
  - A piece of a graph
  - A piece of XML code

- Thus a RDF document can be viewed as:
  - A set of triples
  - A graph (semantic net)
  - A XML document

- Statements as triples:
  (“Julian Padget”,
  http://www.mydomain.org/siteowner,
  http://www.cs.bath.ac.uk/ jap)
  - The triple \((x, P, y)\) can be read as a logical formula
    \(P(x, y)\), where
    - Binary predicate \(P\) relates object \(x\) to object \(y\)
    - RDF offers only *binary predicates* (properties)
Three Views of a Statement

- A statement can be:
  - A triple
  - A piece of a graph
  - A piece of XML code

- Thus a RDF document can be viewed as:
  - A set of triples
  - A graph (semantic net)
  - A XML document

- Statements as triples:

  - The triple \((x,P,y)\) can be read as a logical formula \(P(x,y)\), where
    - Binary predicate \(P\) relates object \(x\) to object \(y\)
    - RDF offers only *binary predicates* (properties)
Three Views of a Statement

- A statement can be:
  - A triple
  - A piece of a graph
  - A piece of XML code

- Thus a RDF document can be viewed as:
  - A set of triples
  - A graph (semantic net)
  - A XML document

- Statements as triples:
  - The triple \( (x, P, y) \) can be read as a logical formula \( P(x, y) \), where
    - Binary predicate \( P \) relates object \( x \) to object \( y \)
    - RDF offers only binary predicates (properties)
as graphs

- A directed graph with labeled nodes and arcs:

  ![Graph Diagram]

- **from** the resource (the subject of the statement)
- **to** the value (the object of the statement)

- Called a “semantic net” in AI
- The value of a statement may be a resource and it may be linked to other resources
- A set of triples as a semantic net:
as graphs

- A directed graph with labeled nodes and arcs:
  
  
  
  
  

  - **from** the resource (the subject of the statement)
  - **to** the value (the object of the statement)

- Called a “semantic net” in AI

- The value of a statement may be a resource and it may be linked to other resources

- A set of triples as a semantic net:
as graphs

- A directed graph with labeled nodes and arcs:
  - http://www.cs.bath.ac.uk/~jap
  - Julian Padget

- **from** the resource (the subject of the statement)
- **to** the value (the object of the statement)

- Called a “semantic net” in AI
- The value of a statement may be a resource and it may be linked to other resources

- A set of triples as a semantic net:
as graphs

- A directed graph with labeled nodes and arcs:
  
  ![Graph Diagram]

  - **from** the resource (the subject of the statement)
  - **to** the value (the object of the statement)

- Called a “semantic net” in AI

- The value of a statement may be a resource and it may be linked to other resources

- A set of triples as a semantic net:
Graphs are useful for human understanding but The semantic web requires machine-accessible and machine-processable representations. There is a 3rd representation based on XML: But XML is not a part of the RDF data model. Thus, serialisation of XML is irrelevant for RDF. For example:

```
1  <rdf:RDF
2   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:mydomain="http://www.mydomain.org/my-rdf-ns">
4    <rdf:Description
5      rdf:about="http://www.cs.bath.ac.uk/~jap">
6      <mydomain:site-owner>Julian Padget</mydomain:site-owner>
7    </rdf:Description>
8  </rdf:RDF>
```
as statement in XML II

- An RDF document is represented by an XML element with the tag `rdf:RDF`
- The content of this element is a number of descriptions using `rdf:Description` tags
- Every description makes a statement about a resource, identified in 3 ways:
  - an `about` attribute, referencing an existing resource
  - an `ID` attribute, creating a new resource
  - or, without a name, creating an anonymous resource
as statement in XML II

- An RDF document is represented by an XML element with the tag `rdf:RDF`
- The content of this element is a number of descriptions using `rdf:Description` tags
- Every description makes a statement about a resource, identified in 3 ways:
  - an **about** attribute, referencing an existing resource
  - an **ID** attribute, creating a new resource
  - or, without a name, creating an anonymous resource
as statement in XML II

- An RDF document is represented by an XML element with the tag `rdf:RDF`.
- The content of this element is a number of descriptions using `rdf:Description` tags.
- Every description makes a statement about a resource, identified in 3 ways:
  - an **about** attribute, referencing an existing resource
  - an **ID** attribute, creating a new resource
  - or, without a name, creating an anonymous resource
Representing n-ary predicates

- RDF uses only binary properties
  - This is a restriction because often we use predicates with more than 2 arguments
  - Binary predicates can simulate n-ary predicates:

**Example:** referee(X,Y,Z)

- X is the referee in a chess game between players Y and Z
- Introduce:
  - a new auxiliary resource chessGame
  - the binary predicates referee, player1, and player2
- Now can represent referee(X,Y,Z) as

```xml
chessGame X Y Z referee player1 player2
```
Representing n-ary predicates

- RDF uses only binary properties
  - This is a restriction because often we use predicates with more than 2 arguments
  - Binary predicates can simulate n-ary predicates:

- Example: referee(X,Y,Z)
  - X is the referee in a chess game between players Y and Z
  - Introduce:
    - a new auxiliary resource chessGame
    - the binary predicates referee, player1, and player2
  - Now can represent referee(X,Y,Z) as:
Representing n-ary predicates

- RDF uses only binary properties
  - This is a restriction because often we use predicates with more than 2 arguments
  - Binary predicates can simulate n-ary predicates:
- Example: referee(X,Y,Z)
  - X is the referee in a chess game between players Y and Z
  - Introduce:
    - a new auxiliary resource chessGame
    - the binary predicates referee, player1, and player2
  - Now can represent referee(X,Y,Z) as:

```
<chessGame>
  <referee X/>
  <player1 Y/>
  <player2 Z/>
</chessGame>
```
Representing n-ary predicates

- RDF uses only binary properties
  - This is a restriction because often we use predicates with more than 2 arguments
  - Binary predicates can simulate n-ary predicates:

- Example: referee(X, Y, Z)
  - X is the referee in a chess game between players Y and Z
  - Introduce:
    - a new auxiliary resource chessGame
    - the binary predicates referee, player1, and player2

- Now can represent referee(X, Y, Z) as:
Representing n-ary predicates

- RDF uses only binary properties
  - This is a restriction because often we use predicates with more than 2 arguments
  - Binary predicates can simulate n-ary predicates:

- Example: referee(X,Y,Z)
  - X is the referee in a chess game between players Y and Z
  - Introduce:
    - a new auxiliary resource chessGame
    - the binary predicates referee, player1, and player2
  - Now can represent referee(X,Y,Z) as:

```
X
  ^   ^   ^
  |   |   |
  referee player1 player2
 chessGame   Y   Z
```

This is a restriction because often we use predicates with more than 2 arguments.

Binary predicates can simulate n-ary predicates:

Example: referee(X,Y,Z)

X is the referee in a chess game between players Y and Z

Introduce:
- a new auxiliary resource chessGame
- the binary predicates referee, player1, and player2

Now can represent referee(X,Y,Z) as:
RDF is a universal language that lets users describe resources in their own vocabularies. RDF does not assume, nor does it define semantics of any particular application domain. The user can do so in RDF Schema using:

- Classes and Properties
- Class Hierarchies and Inheritance
- Property Hierarchies
RDF Schema I

- RDF is a universal language that lets users describe resources in their own vocabularies

- RDF does not assume, nor does it define semantics of any particular application domain

- The user can do so in RDF Schema using:
  - Classes and Properties
  - Class Hierarchies and Inheritance
  - Property Hierarchies
RDF Schema I

- RDF is a universal language that lets users describe resources in their own vocabularies.
- RDF does not assume, nor does it define semantics of any particular application domain.
- The user can do so in RDF Schema using:
  - Classes and Properties
  - Class Hierarchies and Inheritance
  - Property Hierarchies
RDF Schema II

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

**Diagram**

- **Person**
- **Author** subClassOf **Person**
- **Frank** type **Author**
- **Author** communicatesTo **Reader**
- **Reader** type **Author**
- **Lynda**
- **Reader** communicatesTo **Lynda**
- **Author** communicatesTo **Reader**
- **Frank** communicatesTo **Lynda**

**Labels**

- subClassOf
- domain
- range
- communicatesTo
- type
RDF Schema II

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

```
Author -> communicatesTo -> Reader
  \                        /  \
   v                      v  \\
Frank                     Lynda

Person

subClassOf
  \         /
   v       v
Author -> communicatesTo -> Reader
  \                        /  \
   v                      v  \\
Frank                     Lynda
```

- Defines vocabulary for RDF
- Organizes vocabulary in a typed hierarchy
  - Class, subClassOf, type
  - Property, subPropertyOf
  - domain, range
Critique of RDF(S)

- RDF has its idiosyncrasies and is not an optimal modelling language but
  - It is already a *de facto* standard
  - It has sufficient expressive power—at least for more layers to build on top
  - Using RDF offers the benefit that information maps unambiguously to a model
  - RDF Schema enables the description of specific domains
  - RDF Schema is a primitive ontology language: it offers certain modelling primitives with fixed meaning
  - Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
Critique of RDF(S)

- RDF has its idiosyncrasies and is not an optimal modelling language but
- It is already a *de facto* standard
  - It has sufficient expressive power—at least for more layers to build on top
  - Using RDF offers the benefit that information maps unambiguously to a model
  - RDF Schema enables the description of specific domains
  - RDF Schema is a primitive ontology language: it offers certain modelling primitives with fixed meaning
  - Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
Critique of RDF(S)

- RDF has its idiosyncrasies and is not an optimal modelling language but
- It is already a *de facto* standard
- It has sufficient expressive power—at least for more layers to build on top
  - Using RDF offers the benefit that information maps unambiguously to a model
  - RDF Schema enables the description of specific domains
  - RDF Schema is a primitive ontology language: it offers certain modelling primitives with fixed meaning
  - Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
Critique of RDF(S)

- RDF has its idiosyncrasies and is not an optimal modelling language but
- It is already a *de facto* standard
- It has sufficient expressive power—at least for more layers to build on top
- Using RDF offers the benefit that information maps unambiguously to a model
  - RDF Schema enables the description of specific domains
  - RDF Schema is a primitive ontology language: it offers certain modelling primitives with fixed meaning
  - Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
Critique of RDF(S)

- RDF has its idiosyncrasies and is not an optimal modelling language but
- It is already a *de facto* standard
- It has sufficient expressive power—at least for more layers to build on top
- Using RDF offers the benefit that information maps unambiguously to a model
- RDF Schema enables the description of specific domains
  - RDF Schema is a primitive ontology language: it offers certain modelling primitives with fixed meaning
  - Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
Critique of RDF(S)

- RDF has its idiosyncrasies and is not an optimal modelling language but

- It is already a *de facto* standard

- It has sufficient expressive power—at least for more layers to build on top

- Using RDF offers the benefit that information maps unambiguously to a model

- RDF Schema enables the description of specific domains

- RDF Schema is a primitive ontology language: it offers certain modelling primitives with fixed meaning

- Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
Critique of RDF(S)

- RDF has its idiosyncrasies and is not an optimal modelling language but
- It is already a *de facto* standard
- It has sufficient expressive power—at least for more layers to build on top
- Using RDF offers the benefit that information maps unambiguously to a model
- RDF Schema enables the description of specific domains
- RDF Schema is a primitive ontology language: it offers certain modelling primitives with fixed meaning
- Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions
Content

1. Introduction

2. Ontology Engineering

3. RDF and RDFS (Antoniou & van Harmelen)

4. Processing RDF (Antoniou & van Harmelen)

5. XML reaction

6. Summary
Semantics based on inference rules I

- Semantics in terms of RDF triples instead of restating RDF in terms of first-order logic
- ... and sound and complete inference systems
- This inference system consists of rules of the form:
  1. IF E contains certain triples
  2. THEN add to E certain additional triples

  where E is an arbitrary set of RDF triples

- Examples:
  - Property relations
    1. IF E contains the triple (?x,?p,?y)
    2. THEN E also contains (?p,rdf:type,rdf:property)
  - Class transitivity
    1. IF E contains the triples (?u,rdfs:subClassOf,?v) and
    2. (?v,rdfs:subclassOf,?w)
    3. THEN E also contains (?u,rdfs:subClassOf,?w)
Semantics based on inference rules II

Continued:

Type transitivity

1. IF $E$ contains the triples $(?x,$ rdf:type $,$ ?u $)$ and
2. $(?u,$ rdfs:subClassOf $,$ ?v $)$
3. THEN $E$ also contains the triple $(?x,$ rdf:type $,$ ?v $)$

Any resource $?y$ which appears as the value of a property $?p$ can be inferred to be a member of the range of $?p$

This shows that range definitions in RDF Schema are not used to restrict the range of a property, but rather to infer the membership of the range.

1. IF $E$ contains the triples $(?x,$ ?p $,$ ?y $)$ and $(?p,$ rdfs:range $,$ ?u $)$
2. THEN $E$ also contains the triple $(?y,$ rdf:type $,$ ?u $)$
Why an RDF Query Language?

- Several syntactic XML representations for same information
- Equivalent representations require multiple XQuery queries:
  - //uni:lecturer/uni:title if uni:title element
  - //uni:lecturer/@uni:title if uni:title attribute
- Both XML representations equivalent
- Understanding the semantics

```xml
1  <uni:lecturer rdf:ID="949352">
2   <uni:name>Julian Padget</uni:name>
3  </uni:lecturer>
4  <uni:professor rdf:ID="949318">
5   <uni:name>James Davenport</uni:name>
6  </uni:professor>
7  <rdfs:Class rdf:about="#professor">
8   <rdfs:subClassOf rdf:resource="#lecturer"/>
9  </rdfs:Class>
```

- A query for names of all lecturers should return both Julian Padget and James Davenport
Using select-from-where

- As in SQL
  - **select** specifies the number and order of retrieved data
  - **from** is used to navigate through the data model
  - **where** imposes constraints on possible answers

- Retrieve all phone numbers of staff members:
  
  ```
  select X,Y from {X}phone{Y}
  ```

- Here `X` and `Y` are variables, and `{X}phone{Y}` represents a resource-property-value triple
Implicit and explicit joins

- Retrieve all lecturers and their phone numbers:
  
  \[ \text{select } X, Y \text{ from lecturer}\{X\}.phone\{Y\} \]

- Implicit join: restrict the second query only to those triples, the resource of which is in the variable \( X \)
  
  - Here we restrict the domain of phone to lecturers
  - The dot denotes the implicit join

- Explicit join:
  
  - Retrieve the name of all courses taught by the lecturer with ID 949352:

  \[
  \begin{align*}
  &\text{select } N \text{ from course}\{X\}.\text{isTaughtBy}\{Y\}, \\
  &\quad \{C\}\text{name}\{N\} \text{ where } Y=\text{"949352\" } \text{and } X=C
  \end{align*}
  \]
Triple Stores

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata
- Implementation: special-purpose (e.g., Jena, JRDF, 4store) or on top of conventional SQL databases
- But mapping from RDF queries to SQL is not straightforward, hence
  - special purpose query languages
  - special purpose databases
- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted
- Disadvantage: unstructured data—serious impact on performance due to absence of regularity
- SPARQL: an RDF query language
- SPARQL queries consist of triple patterns, conjunctions, disjunctions, and optional patterns—c.f. earlier examples
Triple Stores

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata
- Implementation: special-purpose (e.g. Jena, JRDF, 4store) or on top of conventional SQL databases
- But mapping from RDF queries to SQL is not straightforward, hence
  - special purpose query languages
  - special purpose databases
- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted
- Disadvantage: unstructured data—serious impact on performance due to absence of regularity
- SPARQL: an RDF query language
- SPARQL queries consist of triple patterns, conjunctions, disjunctions, and optional patterns—c.f. earlier examples
Triple Stores

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata
- Implementation: special-purpose (e.g. Jena, JRDF, 4store) or on top of conventional SQL databases
- But mapping from RDF queries to SQL is not straightforward, hence
  - special purpose query languages
  - special purpose databases
- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted
- Disadvantage: unstructured data—serious impact on performance due to absence of regularity
- SPARQL: an RDF query language
- SPARQL queries consist of triple patterns, conjunctions, disjunctions, and optional patterns—c.f. earlier examples
Triple Stores

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata
- Implementation: special-purpose (e.g. Jena, JRDF, 4store) or on top of conventional SQL databases
- But mapping from RDF queries to SQL is not straightforward, hence
  - special purpose query languages
  - special purpose databases
- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted
- Disadvantage: unstructured data—serious impact on performance due to absence of regularity
- SPARQL: an RDF query language
- SPARQL queries consist of triple patterns, conjunctions, disjunctions, and optional patterns—c.f. earlier examples
Triple Stores

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata

- Implementation: special-purpose (e.g. Jena, JRDF, 4store) or on top of conventional SQL databases

- But mapping from RDF queries to SQL is not straightforward, hence
  - special purpose query languages
  - special purpose databases

- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted

- Disadvantage: unstructured data—serious impact on performance due to absence of regularity

- SPARQL: an RDF query language

- SPARQL queries consist of triple patterns, conjunctions, disjunctions, and optional patterns—c.f. earlier examples
Triple Stores

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata
- Implementation: special-purpose (e.g. Jena, JRDF, 4store) or on top of conventional SQL databases
- But mapping from RDF queries to SQL is not straightforward, hence
  - special purpose query languages
  - special purpose databases
- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted
- Disadvantage: unstructured data—serious impact on performance due to absence of regularity
- SPARQL: an RDF query language
  - SPARQL queries consist of triple patterns, conjunctions, disjunctions, and optional patterns—c.f. earlier examples
Triple Stores

- A purpose-built database for the storage and retrieval of Resource Description Framework (RDF) metadata

- Implementation: special-purpose (e.g. Jena, JRDF, 4store) or on top of conventional SQL databases

- But mapping from RDF queries to SQL is not straightforward, hence
  - special purpose query languages
  - special purpose databases

- Advantage: unstructured data—no need to design table structure in advance, so can handle whatever relations are asserted

- Disadvantage: unstructured data—serious impact on performance due to absence of regularity

- SPARQL: an RDF query language

- SPARQL queries consist of triple patterns, conjunctions, disjunctions, and optional patterns—c.f. earlier examples
Introduction

Ontology Engineering

RDF and RDFS (Antoniou & van Harmelen)

Processing RDF (Antoniou & van Harmelen)

XML reaction

Summary
XML developments

- Semantic technology (RDF, OWL) appears to supersede syntactic technology (HTML, XML)
- But switching to semantic mark-up and maintenance is costly
- Some tasks can be achieved using additional XML recommendations

XML reaction

- XQuery
- XPointer
- XLink
- XPath
- XSLT
XML developments

- Semantic technology (RDF, OWL) appears to supersede syntactic technology (HTML, XML)
- But switching to semantic mark-up and maintenance is costly
- Some tasks can be achieved using additional XML recommendations
XML developments

- Semantic technology (RDF, OWL) appears to supersede syntactic technology (HTML, XML)
- But switching to semantic mark-up and maintenance is costly
- Some tasks can be achieved using additional XML recommendations

![XML developments diagram]

- XQuery
- XPointer
- XLink
- XPath
- XSLT
XML recommendations

- XSL: EXtensible Stylesheet Language
- XSLT: XSL Transformations.
- XPath is a language for finding information in an XML document:
  - XPath is a syntax for defining parts of an XML document
  - XPath uses path expressions to navigate in XML documents
  - XPath is a major element in XSLT
- XQuery: a kind of database language for extracting data from XML documents
- XLink: hyperlink creation in XML documents.
- XPointer: extends hyperlinks to point to more specific parts in the XML document.
XML recommendations

- XSL: EXtensible Stylesheet Language
- XSLT: XSL Transformations.
  - XPath is a language for finding information in an XML document:
    - XPath is a syntax for defining parts of an XML document
    - XPath uses path expressions to navigate in XML documents
    - XPath is a major element in XSLT
  - XQuery: a kind of database language for extracting data from XML documents
  - XLink: hyperlink creation in XML documents.
  - XPointer: extends hyperlinks to point to more specific parts in the XML document.
XML recommendations

- XSL: EXtensible Stylesheet Language
- XSLT: XSL Transformations.
- XPath is a language for finding information in an XML document:
  - XPath is a syntax for defining parts of an XML document
  - XPath uses path expressions to navigate in XML documents
  - XPath is a major element in XSLT
- XQuery: a kind of database language for extracting data from XML documents
- XLink: hyperlink creation in XML documents.
- XPointer: extends hyperlinks to point to more specific parts in the XML document.
XML recommendations

- XSL: EXtensible Stylesheet Language
- XSLT: XSL Transformations.
- XPath is a language for finding information in an XML document:
  - XPath is a syntax for defining parts of an XML document
  - XPath uses path expressions to navigate in XML documents
  - XPath is a major element in XSLT
- XQuery: a kind of database language for extracting data from XML documents
  - XLink: hyperlink creation in XML documents.
  - XPointer: extends hyperlinks to point to more specific parts in the XML document.
XML recommendations

- XSL: EXtensible Stylesheet Language
- XSLT: XSL Transformations.
- XPath is a language for finding information in an XML document:
  - XPath is a syntax for defining parts of an XML document
  - XPath uses path expressions to navigate in XML documents
  - XPath is a major element in XSLT
- XQuery: a kind of database language for extracting data from XML documents
- XLink: hyperlink creation in XML documents.
- XPointer: extends hyperlinks to point to more specific parts in the XML document.
XML recommendations

- XSL: EXtensible Stylesheet Language
- XSLT: XSL Transformations.
- XPath is a language for finding information in an XML document:
  - XPath is a syntax for defining parts of an XML document
  - XPath uses path expressions to navigate in XML documents
  - XPath is a major element in XSLT
- XQuery: a kind of database language for extracting data from XML documents
- XLink: hyperlink creation in XML documents.
- XPointer: extends hyperlinks to point to more specific parts in the XML document.
XPath Path Expressions

- Path expressions select document elements:
  - nodename: all child nodes of the named node
  - /: from the root node
  - //: from the current node that match the selection no matter where they are
  - .: current node
  - ../: parent of the current node
  - @: attributes
XPath example

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
  <book>
    <title lang="eng">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="eng">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

- `bookstore`: child nodes of `bookstore`
- `/bookstore`: the root element `bookstore`
- `bookstore/book`: books that are children of `bookstore`
- `//book`: all book elements in the document
- `bookstore //@lang`: books that are descendants of `bookstore`
- `//@lang`: all attributes named `lang`
XPath Predicates

- /bookstore/book[1]: first book that is the child of bookstore
- /bookstore/book[last()]: last book that is the child of bookstore
- /bookstore/book[last()-1]: last but one
- /bookstore/book[position()<3]: first two
- //title[@lang]: all title elements with attribute lang
- //title[@lang='eng']: all above, but with a value of 'eng'
- /bookstore/book[price>35.00]: books with prices greater than 35.00
- /bookstore/book[price>35.00]/title: titles of books with prices greater than 35.00
More XPath

- Selecting unknown nodes? Use * for nodes, @* for attributes
- Selecting multiple paths? Use | for conjunction (!)
- Axis: defines a set of nodes relative to current node
  - child::book
  - attribute::lang
  - ancestor::book, etc.
- XPath operators: arithmetic, numerical comparison, logical combinations
XLink and XPointer

- XLink allows the definition of links outside the linked files—recall embedded vs. external.
- XPointer extends Xlink to use XPath to reach inside XML documents.
- xlink:show controls whether in-line or new window.
- xlink:actuate controls when resource appears (e.g. onLoad, on Request).
- XPointer allows reference to an element with a particular id.
XQuery

- for $x$ in doc("books.xml")/bookstore/book
  where $x$/price>30
  return $x$/title

- Does for XML what SQL does for tables
- FLWOR ("For, Let, Where, Order by, Return"), or
- XPath equivalent:
  doc("books.xml")/bookstore/book[price>30]/title
- But can sort by addition of "order by $x$/title"
More XQuery

- Additionally, can filter content:
  
  ```xml
  <ul>
    { ... return <li>{data($x)}</li> ... }
  </ul>
  ```

- results in
  
  ```xml
  <ul>
    <li>Everyday Italian</li>
    <li>Harry Potter</li>
    <li>Learning XML</li>
    <li>XQuery Kick Start</li>
  </ul>
  ```

- Also: conditionals, selection, filtering, > 100 built-in functions + user-defined
Resources

“Never believe one source”

- XSLT Tutorial. See http://www.w3schools.com/xsl/default.asp
- XPath Tutorial. See http://www.w3schools.com/xpath/xpath_intro.asp
- XQuery Tutorial. See http://www.w3schools.com/xquery/default.asp
- XLink and XPointer Tutorial. See http://www.w3schools.com/xlink/default.asp
Content

1. Introduction
2. Ontology Engineering
3. RDF and RDFS (Antoniou & van Harmelen)
4. Processing RDF (Antoniou & van Harmelen)
5. XML reaction
6. Summary
Summary

- Ontology construction is a major topic in itself
  - Build it yourself for small, specialized domains
  - Find one to re-use and/or extend otherwise
- RDF is the assembler language of the semantic web
- Triples are a simple and effective way to capture, express and query semantic relationships
- Extensions of XML may be more appropriate for localized solutions, but while powerful syntactically, do not address semantic issues.
- The strength and the weakness of linked data is in the lack of structure.
References

[?] does not address semantic web topics. A useful source of information is [?].