Semantic Summary

Lecture 1:

- Good engineers convert problems into products.
- Managers talk about products (buying, managing, selling)
- Advanced users installing APIs, administrating, using product as part of their professional activity, and understanding WHAT the product is.
- VIP engineers invent new problems.
- Semantic web:
    - Allow machines to understand data.
    - Ease sharing and mixing data.
    - Extend the world wide web rather than replace it.
- Semantic web: "An **extension** of the current Web in which information is given well defined meaning, better enabling computers and people to work in cooperation."
- Semantic is used to simplify communication between people.
- Semantic = meaning.
- **Relational understanding** means a child know what to do and can explain why.
- **Instrumental understanding** means a child knows a rule or procedure and has the ability to use it.
- **Understanding:** the process of connecting (linking) the conceptual items of new knowledge and skills coming from external sources to the ones already stored as a personal knowledge and skills.
- **Resource Identifier:** unique identifiers used to locate a particular resource on the network. URI (Uniform Resource Identifier) / URL (Uniform Resource Locator).
- **Markup language:** characters or codes embedded in text which indicate structure, semantic meaning, or advice on presentation.
- **Semantic annotations** are specific sort of metadata, which provides information about particular domain objects, values of their properties and relationships, in a machine-processable, formal and standardized way.
- Ontologies make metadata interoperable and ready for efficient sharing and reuse. machines. They are used as a form of agreement-based knowledge representation about the world.
- What is semantic web for?
    - Integrating
    - Searching

Lecture 2:

- Use ontologies to specify meaning of annotations.
    - Ontologies provide a vocabulary of terms.
    - New terms can be formed by combining existing ones.
    - Meaning (semantics) of such terms is formally specified.
    - Can also specify relationships between terms in multiple ontologies.
- Semantic technologies are digital tools that represent meanings and knowledge.
- Semantic technology as a software technology allows the meaning of information to be known and processed at execution time.

Lecture 3:

- HTML:
    - Tags defined.
    - Static.
    - Not case sensitive.
    - No storage of data.
- XML:
    - Can make my own tags.
    - Dynamic.
    - Case sensitive.
- Web page vs web portal.
- node = label + contents
- DTD (Document Type Definition) describe the grammar and structure of permissible XML trees.
- Agree on language used to define meanings:
    - Flexible and extensible.
        - New terms can be formed by combining existing ones.
    - Meaning (semantics) of such terms is formally specified.
- The origin of ontology:
    - : Onto: = existence or being real.
    - : Logia: = science or study.
- An ontology consists of:
    - A **vocabulary** used to describe some domain.
    - An **explicit specification** of the **intended meaning** of the vocabulary.
- Ontology should:
    - Capture a **shared understanding** of a domain of interest.
    - Provide a **formal** and **machine understandable** model.
    - An ontology describes the things we want to talk about, including both objects and relationships.

- Ontology example:
    - Concept:
        - Conceptual entity of the domain.
    - Attribute :
        - Property of a concept.
    - Relation:
        - Relationship between concepts or properties.
    - Axiom (constraints)
        - Coherent description between concepts / properties / relations via logical expressions.
- Why develops an ontology?
    - To **share** common understanding of the structure of information.
    - To enable **reuse** of domain knowledge.
    - To make domain assumptions **explicit**.
    - To **separate** domain knowledge from the operational knowledge.
- Ontology components:
    - **Concepts:** set of entities within a domain.
    - **Individuals:** instances or objects
    - **Relations:** interactions between concepts or concepts' properties.
    - **Axioms:** explicit rules to constrain the use of concepts.
- RDF statement:
    - **Subject** of an RDF statement is a resource.
    - **Predicate** of an RDF statement is a property of a resource.
    - **Object** of an RDF statement is the value of a property of a resource.
        - Ex: Predicate(subject, object).

<u>Lecture 4:</u>

- Ontology Main Elements:
    - Defining concepts in the domain **(classes)**
        - In most representations, members of classes must be individuals.
        - Classes can be subsumed by or can subsume other classes **sub classes** and **super classes** .
        - Some ontologies consist **only of a class hierarchy** these are called **taxonomy**.
    - 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.
        - Individuals are **instances** or **objects**.
        - They are usually concrete or abstract.
        - Two individuals may be equivalent.

- Self-standing things:
    - Things that can exist on their own nouns.
- Modifiers:
    - Properties or values.
    - Things that modify ("inhere") in other things.
    - Roughly adjectives and adverbs.
- Why reuse other ontologies?
    - To save the **effort**.
    - To **interact** with the tools that use other ontologies.
    - To use ontologies that **have been validated** through use in applications.
- There are two ways of specifying values for modifiers:
    - Value **partitions** (classes that partition a quality) [dangerousness]
    - Value **sets** (individuals that enumerate all states of a quality) [gender]
- Define Property:
    - Identify the domain and range constraints for properties:
        - **Animal** eats **LivingThing**
            - domain: **Animal** range: **LivingThing**
- Models of development:
    - **Top down** define the most general concepts first and then specialize them.
    - **Bottom up** define the most specific concepts and then organize them in more general classes.
    - **Combination** defines the more salient concepts first and then generalize and specialize them.
- **An instance of a subclass is an instance of a superclass.**
- Classes can overlap arbitrarily.
- Disjoint classes are classes that they cannot have common instances.
- Equivalent classes: other classes or groups are equivalent to the selected class.
- **Nouns** are objects (concepts).
- **Verbs** and **prepositions** are relationships.
- Types of properties:
    - Fundamental properties (data properties)
        - attributes of instances of the class.
    - Relations to other object (object properties)
- Golden Rules:
    - A **subclass** inherits all the slots from the **superclass**.
    - If a class has multiple **super classes**, it inherits slots from all of them.
    - Make the cardinality range smaller.
    - Replace a class in the range with a subclass.
    - If some property links individual a to individual b, then its inverse property will link individual b to individual a.

- Define the values of the data property:
    - Slot cardinality
        - defines how many values a slot can have.
        - **Minimum and maximum** value - a range of values for a numeric slot.
            - Minimum cardinality 1 means that the slot must have a value.
            - Minimum cardinality 0 means that the slot value is optional.
            - Maximum cardinality 1 means that the slot can have at most one value **(single valued slot)**.
            - Maximum cardinality greater than 1 means that the slot can have more than one value **(multiple valued slot)**.
        - **Default** value - the value a slot has unless explicitly specified otherwise.
            - A default value can be changed.
            - The default value is a common value for the slot but is not a required value.
    - Slot value type
        - what types of values can fill in the slot?

Ontology	**OO class structure**
Reflects the structure of the **world**.	Reflects the structure of the **data** and **code**.
Is often about **structure** of concepts.	Is usually about **behavior** (methods).
Actual physical representation is **not** an issue.	Describes the **physical representation** of data (long int , char, etc.).

- Open world reasoning (Negation as contradiction)
    - Anything might be true unless it can be proven false.
- Closed world reasoning (Negation as failure)
    - Anything that cannot be found is false.

Database	Ontology
Closed world assumption (CWA)	Open world assumption (OWA)
Unique name assumption (UNA)	No UNA
Schema behaves as constraints on structure of data.	Ontology axioms behave like implications (inference).

Lecture 5:

- RDF = Resource Description Framework.
- Structured information → array/vector.
- Unstructured information → text/webpages.
- RDF triples:
    - Subject predicate object
    - A Resource (Subject) is anything that can have a URI: URIs or blank nodes.
    - A Property (Predicate) is one of the features of the Resource: URIs.
    - A Property value (Object) is the value of a property, which can be literal or another resource: URIs, literal, blank nodes (anonymous label).
    - This is drawn as a directed graph.
- Graph data:
    - Resource nodes: A resource is anything that can have things said about it.
    - Literal nodes: a literal is the same as a value.
- Description is written before a resource.
- Creator is written before an object, 's' is a specific namespace prefix as reference to ontology where predicates are defined.
- A namespace is a URI that is defined once at the beginning and used in sentences later without referring to URI again.
- The subject of one triple can be the object of another.
- RDF Container Elements:
    - <Bag>: a list of members without order; duplicates allowed.
    - <Seq>: a list of members with order; duplicates allowed.
    - <Alt>: a list of members that only one can be selected.
- Blank node is called anonymous resource in RDF
    - We may not know the URI.
    - There is no point in minting a URI for them.
    - To identify a blank node there should be an underscore before it.

Lecture 6:

- RDFS describes the **vocabulary** of the RDF data model.
- RDFS → Sets of individuals sharing properties called classes.
- RDF → Individual objects in the domain.
- Core properties:
    - rdf:type , which relates a resource to its class.
    - rdfs:subClassOf , which relates a class to one of its super classes.
    - rdfs:subPropertyOf , relates a property to one of its super properties.
    - rdfs:domain , which specifies the domain of a property P.
    - rdfs:range , which specifies the range of a property P.
- A class may have multiple super classes.
- To identify a super class there should be a hashtag before it.
- If there is a property that the domain and range is not stated, then it will use that of super class.

Lecture 7:

- The **Web Ontology Language (OWL)** is a family of knowledge representation languages or ontology languages for engineering ontologies.
- **OWL DL (Description Logic)** designed to provide the maximum expressiveness possible while retaining computational completeness and
- RDFS is too weak in describing resources with sufficient details,
    - No localized range and domain constraints.
    - No cardinality constraints.
    - No transitive, inverse, or symmetrical properties.
    - Disjoint classes.
    - Disjoint classes.
- Why OWL?
    - Class membership.
    - Equivalence of classes.
    - Consistency.
    - Classification.
- OWL extends vocabulary and adds axioms to express **more complex relations of classes and properties**.
- Subclasses as we know them from RDFS: rdfs:subClassOf.
- owl:Thing is the most general class, which contains everything.
- owl:Nothing is the empty class.
- Class inclusion:
    - President **rdfs:subClassOf** ex:Politician
- Class Enumeration:
    - A class description of the "enumeration" kind is defined with the **owl:oneOf** property that represents exactly the enumerated individuals, no more, no less.
    - The list of individuals is typically represented by RDF construct **rdf:parseType="Collection"**
    - Collection has **limited** values, whether container has **unlimited** values.
- Class intersection:
    - The **owl:intersectionOf** links a class to a list of class descriptions.
- Class union:
    - The **owl:unionOf** is used.
- Class complement:
    - The **owl:complementOf** is used.

- OWL contains three language constructs for combining class descriptions into class axioms:
    - **rdfs:subClassOf** : a class description is a subset of another class description.
    - **owl:equivalentClass** : a class description is exactly the same as another class description.
    - **owl:disjointWith** : a class description has no members in common with the other class description.
- There are two types of property in OWL:
    - **Object properties** (**owl:ObjectProperty**) relates individuals (instances) of two OWL classes.
    - **Datatype properties** (**owl:DatatypeProperty**) relates individuals (instances) of OWL classes to literal values.
- Transitive property:
    - If a property P is transitive, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via property P.
- Functional property:
    - Is a property that can have only one (unique) value y for each instance x.
    - Both object properties and datatype properties can be declared as "functional".
- Property restriction:
    - A (restriction) class is achieved through an **owl:Restriction** element.
    - This element contains an **owl:onProperty** element and one or more.
    - **owl:allValuesFrom** specifies universal quantification.
    - **owl:hasValue** specifies a specific value.
    - **owl:someValuesFrom** specifies existential quantification.
    - We can specify minimum and maximum number using **owl:minCardinality** and **owl:maxCardinality**.
- Property inherence:
    - (a subPropertyOf b) and (b inverseOf c) doesn't imply (a inverseOf c)
    - (a subPropertyOf b) and (b equivalentPropertyOf c) doesn't imply (a equivalentPropertyOf c)
    - (a subPropertyOf b) and (b type TransitiveProperty) doesn't imply (a type TransitiveProperty)
- Individuals:
    - **owl:sameAs** is used to state that two URI references refer to the same individual.
    - **owl:differentFrom** is used to state that two URI references refer to different individuals.

    - **owl:AllDifferent** provides an idiom for stating that a list of individuals is all different.
- OWL dialects:
    - OWL lite
    - OWL DL
    - OWL full

Lecture 8:

- **SPARQL** consists of two parts: query language and protocol.
- **SPARQL** is an RDF query language (semantic query language) for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format.
- **SPARQL** queries are not constrained to working within one database.
- A basic **SPARQL** query is simply a graph pattern with some variables.
- A **SPARQL** query consists of three parts:
    - Graph pattern: Specifying a graph pattern, which is just RDF using some variables.
    - Matching: When RDF data matches a specific graph pattern.
    - Binding: When a specific value in RDF is bound to a variable in a graph pattern.
- Variables denoted with **?**.
- Types of **SPARQL** queries:
    - SELECT: return a table of all X, Y, etc. satisfying the following conditions.
    - CONSTRUCT: Find all X, Y, etc. satisfying the following conditions and generate RDF statements, creating a new graph.
    - DESCRIBE: find all statements in the dataset that provide information about the following resource.
    - ASK: are there any X, Y, etc. satisfying the following conditions.
- A SPARQL query consists of a:
    - Body: pattern matching expression.
    - Head: processing of the variables.
- **PREFIX** keyword: describes prefix declarations for abbreviating URIs.
- When you use the abbreviation (foaf:name), it appends the string after the colon (:) to the URI that is referenced by the prefix string.
- Basic graph pattern:
    - Triple Pattern - similar to an RDF Triple (subject, predicate, object), but any component can be a query variable.
- Group pattern:
    - A set of graph patterns must all match.
- Value constraints:

    - **FILTERs** restrict the solutions of a graph pattern can restrict on using arithmetic comparisons or logical expressions.
    - **regex**: match only plain literals with no language tag.
        - Expression may be made case insensitive with the "i" flag.
- Optional graph patterns:
    - If optional part does not match, it creates no bindings but does not eliminate the solution.
- Patterns on Named Graphs:
- Solution Modifiers:
    - **ORDER BY**: used to sort the results (ASC or DESC).
    - **LIMIT**: indicates the maximum number of rows that should be returned.
    - **Distinct**: ensure solutions in the sequence are unique.
    - **Offset**: control where the solutions start from in the overall sequence of solutions.
    - **Reduced**

Lecture 9:

- Knowledge graph represents a collection of interlinked descriptions of entities, it combines characteristics of several data:
    - Database
    - Graph
    - Knowledge base
- GKG enhances Google Search in three main ways:
    - Find the right thing
        - Deals with the ambiguity of the language.
    - Summaries
        - Summarize relevant content around that topic, including key facts about the entity.
    - Deeper and broader information
        - Reveal new facts.
        - Anticipate what the next questions and provide the information beforehand (based on what other users asked).