Αναπαράσταση Γνώσης (Knowledge Representation)

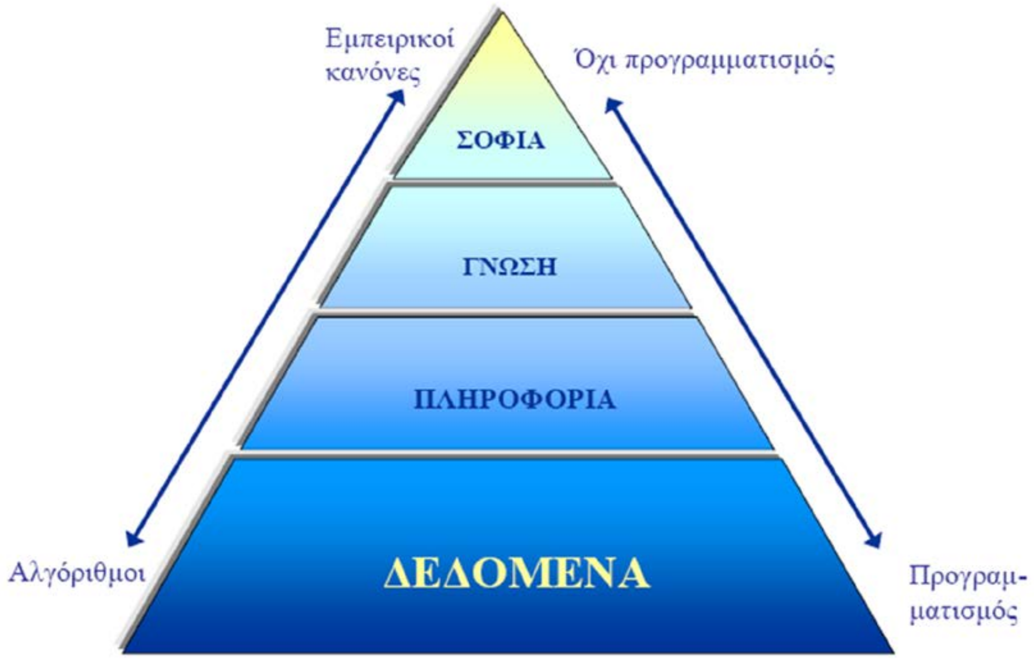
Μια οντολογική σκοπιά
(Ontological engineering)

Πιτικάκης Μάριος, Ph.D.

Αναπαράσταση Γνώσης

- Σύνολο συντακτικών και σημασιολογικών παραδοχών, οι οποίες καθιστούν δυνατή την περιγραφή ενός κόσμου.
- Η φυσική γλώσσα είναι ακατάλληλη για αναπαράσταση γνώσης, γιατί υπάρχει
  - Αμφισημία/Ασάφεια (ambiguity)
  - Ερμηνεία από τα συμφραζόμενα (context)
- Χρειάζεται ένας μονοσήμαντος και τυποποιημένος συμβολισμός
  - Επακριβής αναπαράσταση γνώσης
  - Συνδυασμός με μηχανισμό εξαγωγής συμπερασμάτων (**inference mechanism**)
- Μία μέθοδος αναπαράστασης γνώσης έχει:
  - *Συντακτικό (syntax):* σύμβολα και κανόνες συνδυασμού τους
  - *Σημασιολογία (semantics):* καθορισμός των εννοιών που αποδίδονται στα σύμβολα και στους συνδυασμούς που επιτρέπει το συντακτικό.

Από την επεξεργασία δεδομένων στα Συστήματα Γνώσης



- Γνώση (knowledge) θεωρείται η κατανόηση που αποκτάται μέσω εμπειρίας ή μελέτης και συμπεριλαμβάνει όλες τις πληροφορίες, τις εμπειρίες, τις ικανότητες, τις δεξιότητες και την κοινή λογική
- Σοφία (wisdom) είναι η ικανότητα να χρησιμοποιεί κάποιος τη γνώση όσο το δυνατόν αποδοτικότερα (αναθεώρηση, μάθηση, διορατικότητα, πρόβλεψη).

Είδη γνώσης

- **Αντικείμενα (objects):** Αναπαράσταση των αντικειμένων ενός κόσμου και των σχέσεων μεταξύ τους
  - Σημασιολογική γνώση (semantic knowledge), ιεραρχικά δομημένη.
- **Συμβάντα (events):** Αναπαράσταση των ενεργειών και της χρονικής ακολουθίας με την οποία συμβαίνουν, καθώς και τις σχέσεις αίτιου-αποτελέσματος.
  - Επεισοδιακή γνώση (episodical knowledge).
  - Προσωπικές εμπειρίες ενός ατόμου, οργανωμένες χρονικά και χωρικά σε επεισόδια και όχι σε έννοιες ή σχέσεις.
- **Εκτέλεση ενεργειών (performance):** Αναπαράσταση των δεξιοτήτων για το πώς κάποιος κάνει πράγματα (εκτελεί εργασία ή διεκπεραιώνει διαδικασία)
  - Διαδικαστική γνώση (procedural knowledge).
- **Μετα-γνώση (meta-knowledge):** Αναπαράσταση της γνώσης για το τι γνωρίζει κάποιος και πότε πρέπει να το εφαρμόσει.
  - Συνώνυμη της *σοφίας* (wisdom).

Συλλογιστική (Reasoning)

- Είναι αναπόσπαστο συστατικό της νοημοσύνης
- Συλλογιστική: μέθοδος με την οποία τμήματα υπάρχουσας γνώσης συνδυάζονται μεταξύ τους ώστε να παράγουν νέα γνώση ή να εξάγουν συμπεράσματα.
- Κάθε μέθοδος αναπαράστασης της γνώσης έχει τις δικές της συλλογιστικές. Οι πιο γνωστές συλλογιστικές (για συστήματα κανόνων) είναι:
  - *Συνεπαγωγή (deduction)*
  - *Επαγωγή (induction)*
  - *Απαγωγή (abduction)*
- Εξελιγμένες συλλογιστικές που χρησιμοποιούνται στα συστήματα γνώσης:
  - Συλλογιστική των Μοντέλων (model-based reasoning)
  - Ποιοτική Συλλογιστική (qualitative reasoning)
  - Συλλογιστική των Περιπτώσεων (case-based reasoning)
  - Συλλογιστική με αναλογίες (analogical reasoning)

Εξαγωγή Συμπερασμάτων (inference)

- Εξαρτάται από:
  - τη συλλογιστική
  - τη στρατηγική αναζήτησης στη γνώση ενός προβλήματος
- **Στρατηγική αναζήτησης:** Ο τρόπος με τον οποίο έχει δομηθεί και κωδικοποιηθεί η γνώση προκειμένου να δώσει λύση σε ένα πρόβλημα. Υλοποιείται με διάφορους τρόπους:
  - **Οδηγούμενη από τους στόχους (goal driven ή top-down):** Ξεκινάμε από πιθανά συμπεράσματα και φτάνουμε στις αιτίες που τα στηρίζουν.
  - **Οδηγούμενη από τα δεδομένα (data driven ή bottom-up):** Ξεκινάμε από τα δεδομένα του προβλήματος και φτάνουμε σε συμπεράσματα.
- Οι συλλογιστικές υλοποιούνται από έναν ή περισσότερους εναλλακτικούς μηχανισμούς εξαγωγής συμπερασμάτων (inference mechanisms)

Εξαγωγή Συμπερασμάτων (inference)

- Μηχανισμός Εξαγωγής Συμπερασμάτων: Αλγόριθμος που προσδιορίζει τι συνεπάγεται από την πληροφορία που έχει προστεθεί στη Βάση Γνώσης
  – Οι ερωτήσεις που γίνονται στη βάση χρησιμοποιούν αυτό τον μηχανισμό
- Ένας μηχανισμός εξαγωγής συμπερασμάτων είναι:
  – ορθός/ασφαλής (**sound**) αν κατασκευάζει μόνο προτάσεις που εξάγονται από τη Βάση Γνώσης
  – πλήρης (**complete**) αν κατασκευάζει όλες τις προτάσεις που εξάγονται από τη Βάση Γνώσης

- Τα βήματα που απαιτούνται για να δημιουργηθεί μια πρόταση από ένα σύνολο προτάσεων της Βάσης Γνώσης ονομάζεται απόδειξη (**proof**)

Εξαγωγή Συμπερασμάτων (inference)

Δύο μεγάλες κατηγορίες συστημάτων εξαγωγής συμπερασμάτων:

- **Αιτιολόγηση προς τα πίσω** (backward chaining): Ξεκινάμε από πιθανά συμπεράσματα, τα οποία διατυπώνουμε ως ερωτήσεις και φτάνουμε στις αιτίες που τα υποστηρίζουν (απόδειξη θεωρημάτων, απάντηση ερωτήσεων) (π.χ. Prolog)
- **Αιτιολόγηση προς τα εμπρός** (forward chaining): Ξεκινάμε από τα δεδομένα του προβλήματος και φτάνουμε σε συμπεράσματα, αποφάσεις ή ενέργειες (π.χ. CLIPS, JESS).

Είδη Συλλογιστικής

- ***Συνεπαγωγική συλλογιστική (deductive reasoning):*** *Εξάγει συμπεράσματα βασισμένη στους κλασικούς μηχανισμούς εξαγωγής συμπερασμάτων της λογικής.*
  - *Δεδομένου του κανόνα:* Όλα τα σκυλιά του Κώστα είναι καφέ
  - *και του γεγονότος:* Αυτά τα σκυλιά είναι του Κώστα
  - *Συμπέρασμα που εξάγεται:* Αυτά τα σκυλιά είναι καφέ
- ***Επαγωγική συλλογιστική (inductive reasoning****): αφορά την εξαγωγή γενικών συμπερασμάτων από ένα σύνολο παραδειγμάτων.*
  - *Δεδομένων των γεγονότων:*
  - Το σκυλί Α είναι του Κώστα και είναι καφέ.
  - Το σκυλί Β είναι του Κώστα και είναι καφέ. …
  - *Κανόνας που εξάγεται:* Όλα τα σκυλιά του Κώστα είναι καφέ.
- ***Απαγωγική συλλογιστική (abductive reasoning):*** *Εξαγωγή συμπερασμάτων κατά την οποία, με δεδομένα μία βάση γνώσης και μερικές παρατηρήσεις (observations) επιχειρείται η εύρεση υποθέσεων οι οποίες μαζί με τη βάση γνώσης εξηγούν τις παρατηρήσεις.*
  - *Δεδομένου του κανόνα:* Όλα τα σκυλιά του Κώστα είναι καφέ
  - *και του αποτελέσματος:* Τα σκυλιά είναι καφέ
  - *Υπόθεση που γίνεται:* Αυτά τα σκυλιά είναι του Κώστα

Σύγκριση Συλλογιστικών

Επαγωγή vs. Απαγωγή

- Και οι 2 εμπεριέχουν τη δημιουργία και τον έλεγχο υποθέσεων
- Στην επαγωγή, η υπόθεση αφορά ένα γενικό κανόνα που εξηγεί τα γεγονότα
  - Η ορθότητα του κανόνα-υπόθεση απαιτεί μεγάλο αριθμό παρόμοιων καταστάσεων
- Στην απαγωγή η υπόθεση αφορά ένα συγκεκριμένο γεγονός
  - Για την εξαγωγή συμπεράσματος αρκεί μία μόνο κατάσταση

Συνεπαγωγή vs. Απαγωγή

- Στη συνεπαγωγή, το συμπέρασμα που εξάγεται είναι λογικό επακόλουθο του γενικού κανόνα και του γεγονότος που καταγράφεται ως αληθές
  - Όταν ο γενικός κανόνας δεν είναι απολύτως βέβαιος, τότε δεν μπορεί να εφαρμοστεί Π.χ. Σχεδόν όλα τα σκυλιά του Κώστα είναι καφέ
- Στην απαγωγή, η υπόθεση είναι μία από τις πιθανές και δεν είναι απόλυτα αληθής
  - Όταν ο γενικός κανόνας δεν είναι απολύτως βέβαιος, εξακολουθεί να είναι εφαρμόσιμη
  - Το γεγονός-υπόθεση εξακολουθεί να αποτελεί μια πιθανή εξήγηση της παρατήρησης

Γλώσσες Αναπαράστασης Γνώσης

- Μια γλώσσα αναπαράστασης γνώσης (KR language) ορίζεται από το συντακτικό (syntax) και τη σημασιολογία της (semantics)
- Η Λογική είναι μια γλώσσα αναπαράστασης γνώσης
- Ύπαρξη διαφόρων λογικών:
  – Προτασιακή Λογική (Propositional Logic)
  – Λογική πρώτης τάξεως (First Order Logic) ή Κατηγορηματική λογική (Predicate Logic)

Λογική

- Η Προτασιακή Λογική υποθέτει ότι ο κόσμος αποτελείται από γεγονότα που είτε είναι αληθή είτε όχι
- Στην **λογική πρώτης τάξης** έχουμε τις εξής παραδοχές:
  – Ο κόσμος αποτελείται από **αντικείμενα** (objects) με συγκεκριμένες ταυτότητες. Τα αντικείμενα έχουν **ιδιότητες** που τα ξεχωρίζουν από άλλα αντικείμενα
  – Τα αντικείμενα συμμετέχουν σε **σχέσεις/συσχετίσεις** (relations) με άλλα αντικείμενα.
- Αυτές οι παραδοχές κάνουν την Λογική Πρώτης Τάξης πιο ισχυρή από την Προτασιακή Λογική

Λογική

Παράδειγμα Παραστήστε σε πρωτοβάθμια κατηγορηματική λογική το νόημα των παρακάτω προτάσεων:

- Όλοι οι προγραμματιστές είναι ευτυχισμένοι
  - $\forall x(\ \text{Programmer}(x) \rightarrow \text{Happy}(x)\ )$
- Κάθε προγραμματιστής γνωρίζει μία (πιθανώς διαφορετική) γλώσσα προγραμματισμού
  - $\forall x(\ \text{Programmer}(x) \rightarrow \exists y(\text{Language}(y) \land \text{Knows}(x,y))\ )$
- Υπάρχει μία γλώσσα προγραμματισμού που τη γνωρίζουν όλοι οι προγραμματιστές
  - $\exists y(\ \text{Language}(y) \land \forall x(\text{Programmer}(x) \rightarrow \text{Knows}(x,y)\ )\ )$

Overview of contents

- A general introduction to Conceptual Modeling (Εννοιολογική Μοντελοποίηση) and Knowledge Representation (Αναπαράσταση Γνώσης)
- Ontology-Driven Conceptual Modeling
- The ontology development process & methodology and some guidelines
- Introduction to the Semantic Web
- Logic and Reasoning with ontologies
- Rule-based systems

Introduction to Conceptual Modeling

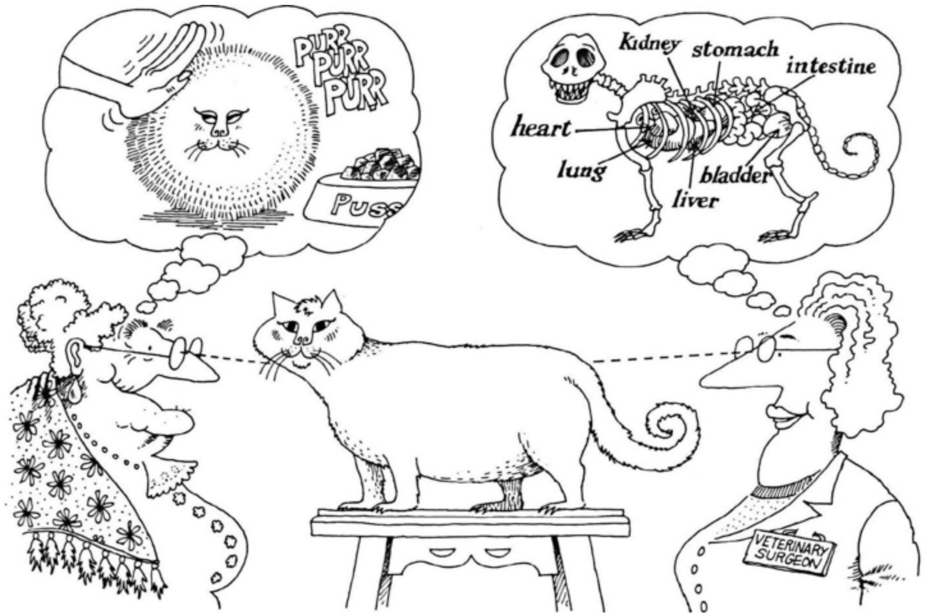
(Εισαγωγή στην Εννοιολογική Μοντελοποίηση)

Information Modeling

- Information modeling means “models **of** information”
  (NOT “models made by information”)
- We are interested in models of information about the real world, or ***somebody’s conception*** of the real world.
- Applications in Computer Science and Engineering:
  - **Databases** -- using semantic data models to build databases;
  - **Software Engineering** -- requirements modeling, using diagrammatic (structured, OO) techniques to build requirement specifications; software process modeling using finite state machines, state charts, rules, Petri nets to build process models;
  - **Artificial Intelligence** -- knowledge representation using logic-based notations, description logics, semantic networks etc. to build **knowledge bases**.
  - …

Conceptual Modeling

- **Goal**: improve models and tools for representing information and processes
- **Why**: narrow the gap between concepts in the real world and their representation in conceptual models
- **How**: identify and formalize (τυπική αναπαράσταση) modeling primitives (θεμελιακά στοιχεία), like generic relationships, for more accurate and intuitive descriptions of real-world concepts

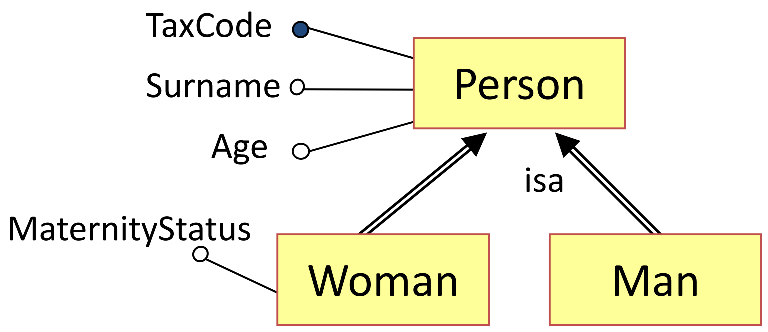


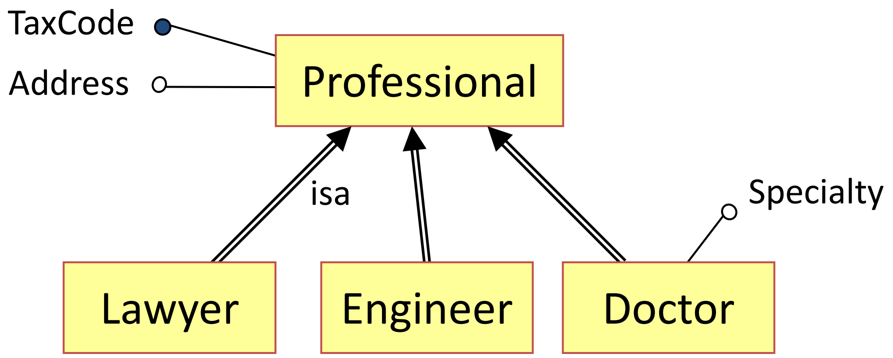
Characterizing Conceptual Models

- An **Entity** (οντότητα) represents classes of objects that have properties in common and an autonomous existence e.g. *City, Employee*. An instance of an entity is an object in the class represented by the entity e.g. *Heraklion, Papadakis*.
- A **Relashionship** (σχέση) represents logical links between two or more entities e.g. *Residence*.
- **Attributes** (χαρακτηριστικά) describe the elementary properties of entities or relationships.

Generalizations

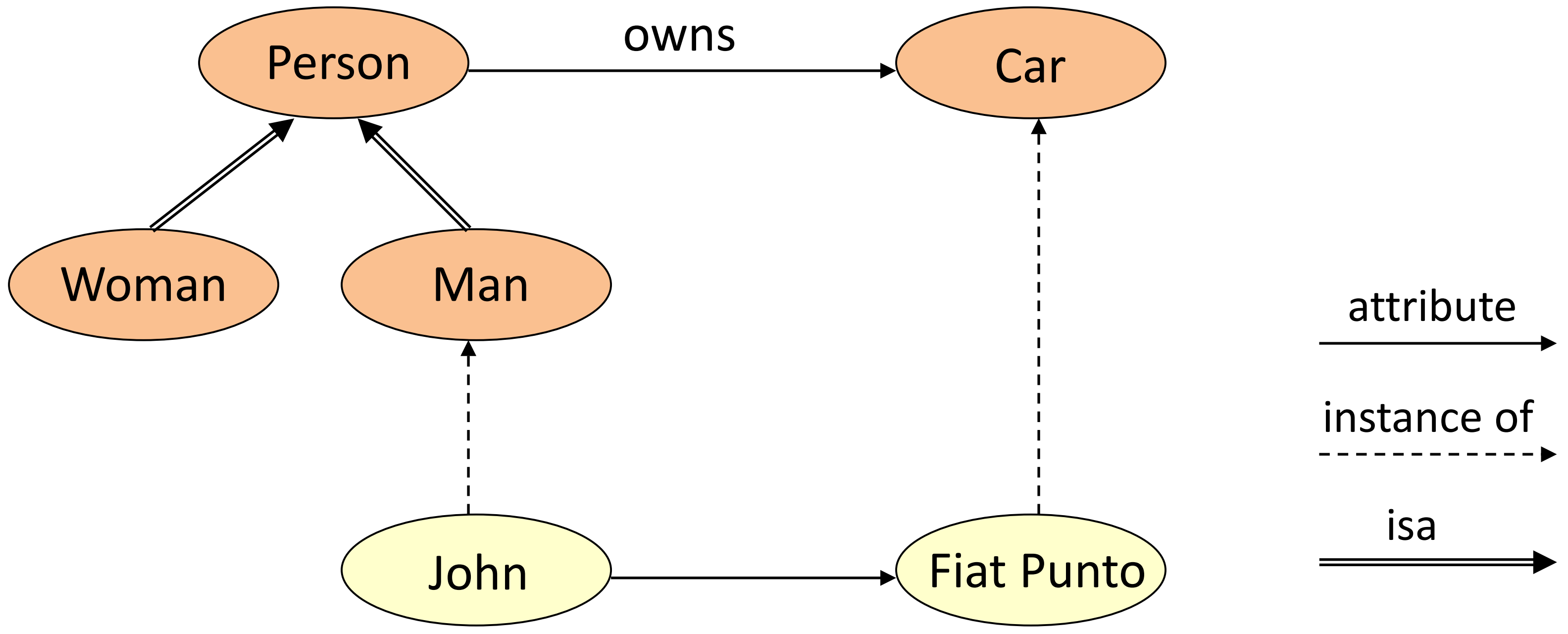
- ***Generalizations*** *(γενικεύσεις)* represent logical links between an entity E, known as ***parent*** entity, and one or more entities E1,...,En called ***child*** entities, of which E is more general. We say that E is a ***generalization*** of E1,...,En and that the entities E1,...,En are ***specializations*** *(ειδικεύσεις)* (noted **isa** i.e. subClassOf) of E.
- Every instance of a child entity is also an instance of the parent entity.
- Every property of the parent entity (attribute, relationship or other generalization) is also a property of a child entity. This is known as ***inheritance*** (*κληρονομικότητα*).





Simple Example

- Objects are structured along three main hierarchies:
  - the generalization/specialization hierarchy
  - the aggregation(attribute) hierarchy
  - the classification hierarchy



Several Generic Relationships

- **Classification**: an instance to its class (e.g. John and person)
- **Generalization**: a superclass to subclasses (e.g. person and employee) i.e. is-a or is-subclass-of or is-superclass-of
- **Aggregation**: composites formed from components (e.g. a car has body and engine) i.e. is-part-of
- **Materialization**: a class of categories (e.g. models of cars) and a class of more concrete objects (e.g. individual cars)
- **Ownership**: an owner class (e.g. persons) and a property owned (e.g. cars)
- **Grouping**: a member class (e.g. players in a team) to a grouping class (e.g. teams)
- **Viewpoint**: partial information about a class from a particular standpoint
- **Generation**: new output entities emerging from input entities
- **Versioning**: an object class and its time-varying versions
- **Role**: an object class (e.g. persons) and a role class (e.g. employees), describing dynamic states for the object class

Conceptual Design

Design choices:

- Should a concept be modeled as an entity or an attribute?
- Should a concept be modeled as an entity or a relationship?
- Identifying relationships: Binary or ternary? Aggregation? Composition?
  - **Aggregation** implies a relationship where the child class can exist independently of the parent class. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.
  - **Composition** implies a relationship where the child class cannot exist independent of the parent class. Example: House (parent) and Room (child). Rooms don't exist separate to a House.

Some rules of thumb (εμπειρικοί κανόνες)

- If a concept has significant properties and/or describes classes of objects with an autonomous existence, it is appropriate to represent it as an entity.
- If a concept has a simple structure, and has no relevant properties associated with it, it is convenient to represent it with an attribute of another concept to which it refers.
- If a concept provides a logical link between two (or more) entities, it is convenient to represent it with a relationship.
- If one or more concepts are particular cases of another concept, it is convenient to represent them in terms of a generalization relationship (isa).

Example

Consider the **address** of a person. Is it an entity, relationship, or attribute?

- Consider a telephone company database, which has to keep track of how many and what type of phones are available in any one household, who lives there (there may be several phone bills going to the same address) etc. for this case, address is probably best treated as an <u>entity</u>.
- Or, consider an employee database, where for each employee you maintain personal information, such as the address. Here address is best represented as an <u>attribute</u>.
- Or, consider a police database where we want to keep track of a person’s whereabouts, including the address (i.e., address from Date1 to Date2, address from Date2 to Date3, etc.) Here, address is treated best as a <u>relationship</u>.

Qualities for a Conceptual Schema

- **Correctness**. Conceptual schema uses correctly the constructs made available by the conceptual model. As with programming languages, the errors can be ***syntactic*** or ***semantic***.
- **Completeness**. Conceptual schema represents all data requirements and allows for the execution of all the operations included in the operational requirements.
- **Readability**. Conceptual schema represents the requirements in a way that is natural and easy to understand. Therefore, the schema must be self-explanatory; for example, by choosing suitable names for concepts.
- **Minimality**. Schema avoids *redundancies*, e.g., data that can be derived from other data.

Conceptual Modeling Strategies

- The design of a conceptual schema for a given set of requirements is an engineering process and, as such, can use design strategies from other disciplines like:
  - Top-down
  - Bottom-up
  - Middle-out
  - Mixed

A Comprehensive Method for Conceptual Design

1. **Analysis of requirements**

   (a) Construct a glossary of terms.

   (b) Analyze the requirements and eliminate all ambiguities.

   (c) Arrange the requirements in groups.

2. **Basic step**

   Identify the most relevant concepts and represent them in a skeleton schema.

3. **Decomposition step** (to be used if appropriate or necessary). Decompose the requirements with reference to the concepts present in the skeleton schema.

A Comprehensive Method for Conceptual Design

4. **Iterative step**

   (a) Refine concepts in the schema, based on the requirements.

   (b) Add new concepts to the schema to describe any parts of the requirements not yet represented.

5. **Integration step**

   Integrate the various subschemas into a general schema with reference to the skeleton schema.

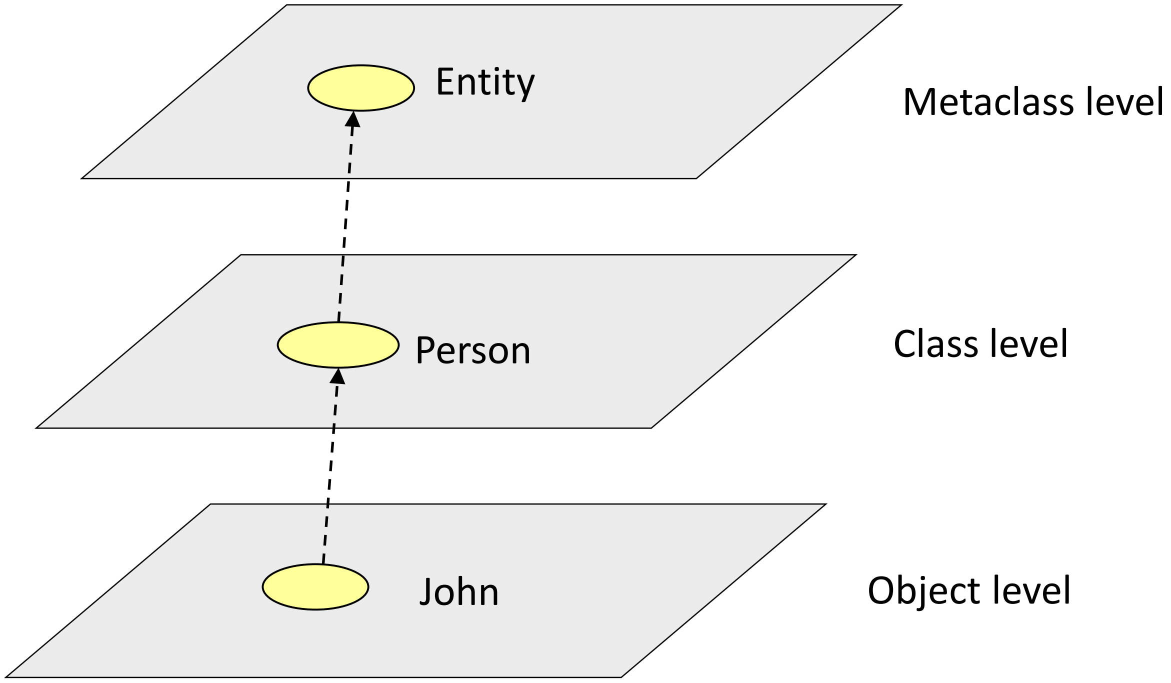
6. **Quality analysis**

   Verify the correctness, completeness, minimality and readability of the schema.

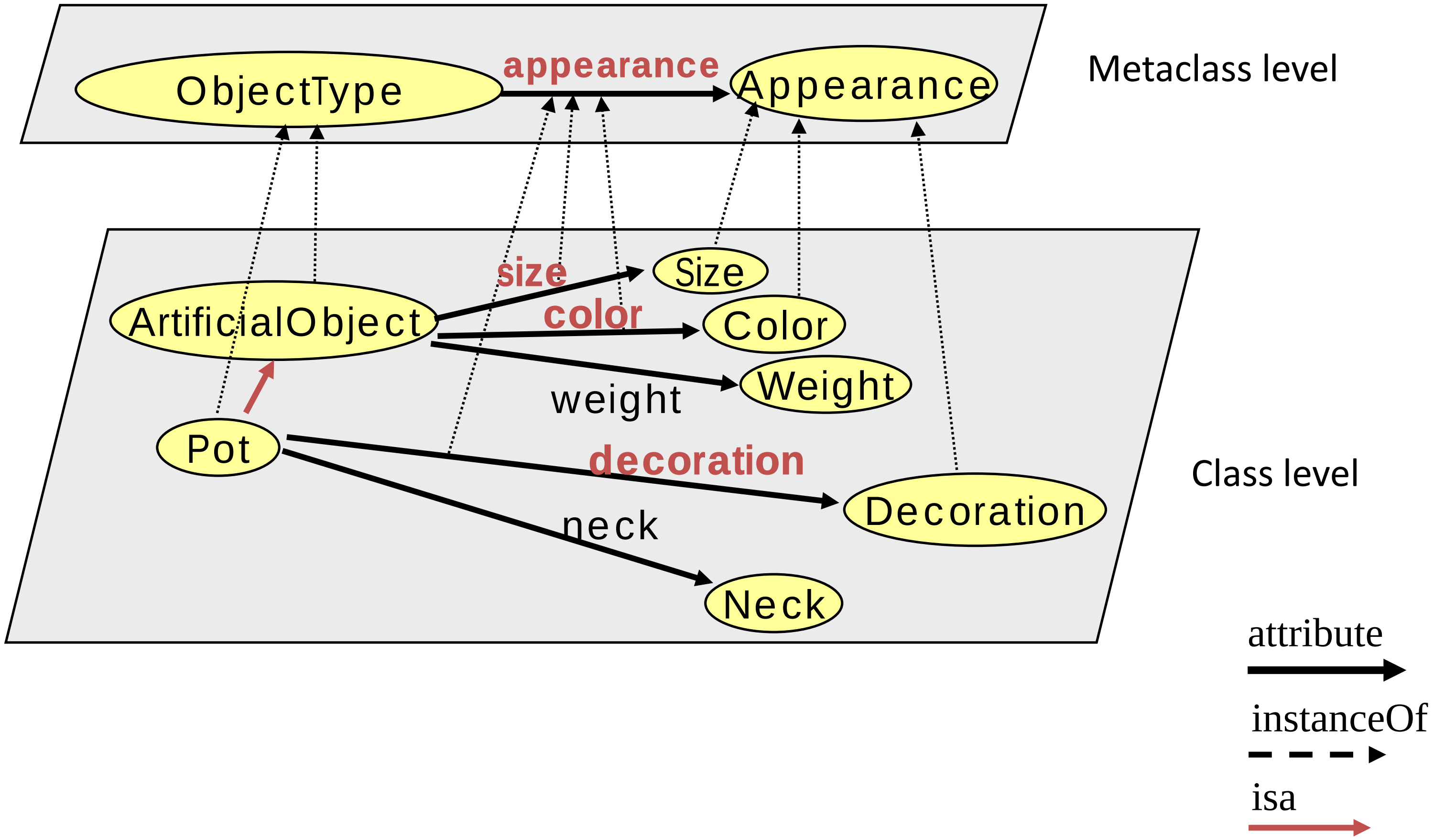
What is Metamodeling?

- “Meta” means literally “after” in Greek.
- In Computer Science, the term is used heavily and with several different meanings:
  - In Databases, metadata means “data about data” and refer to things such as a data dictionary, a repository, or other descriptions of the contents and structure of a data source;
  - In Conceptual Modeling, metamodel is a model of a data model, e.g., an ER model of the relational model, or an ER model of the ER model.
- Data are modelled by metadata (“schemata”, “classes”,…) which are parts of the metamodel; these are instances of metadata which are parts of a metamodel, etc.

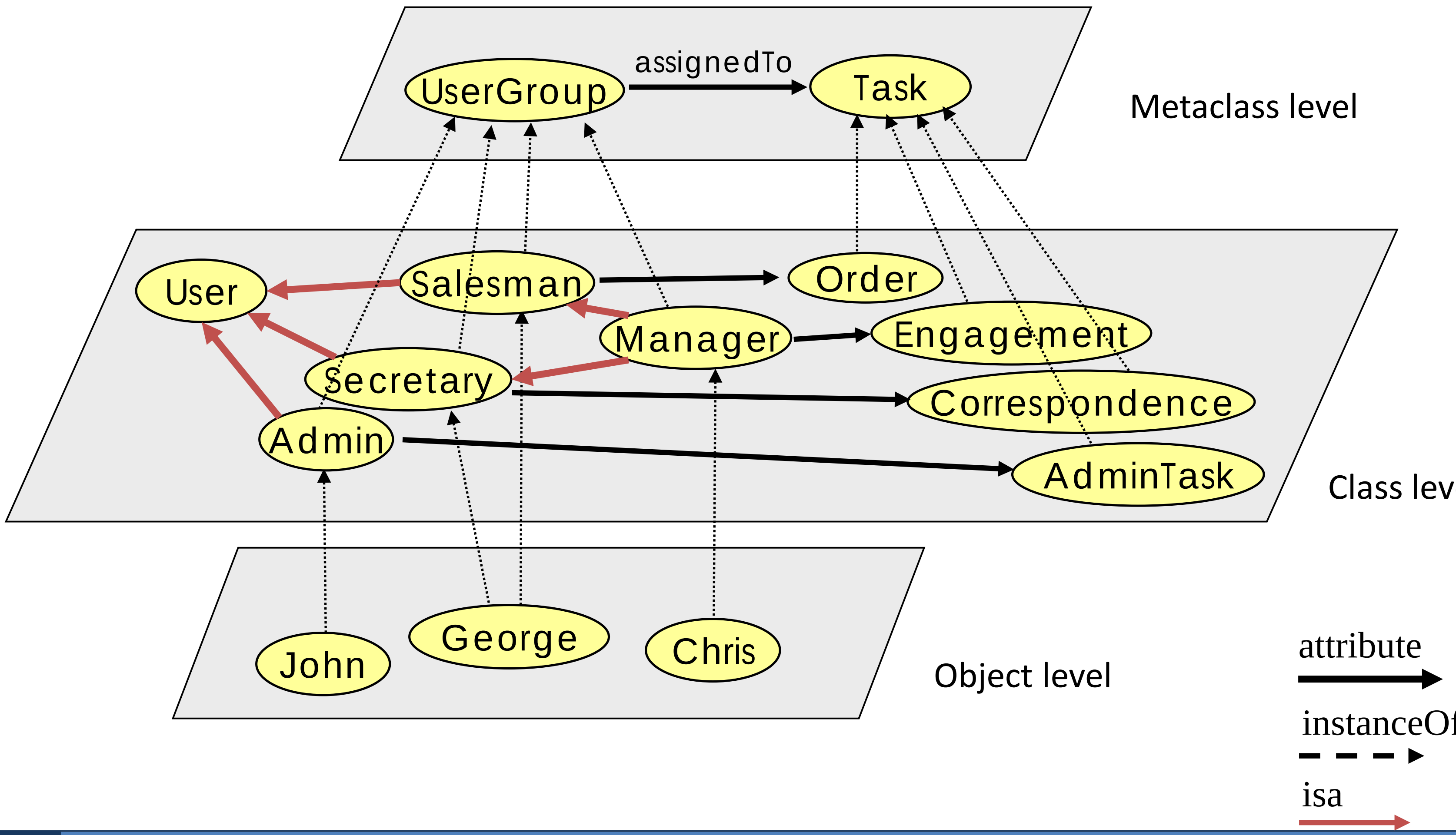
Structural Reflection



Example 1


appearance
ObjectType
Appearance
Metaclass level
size
Size
color
Color
ArtificialObject
weight
Weight
Pot
decoration
Decoration
neck
Neck
Class level
attribute
instanceOf
isa

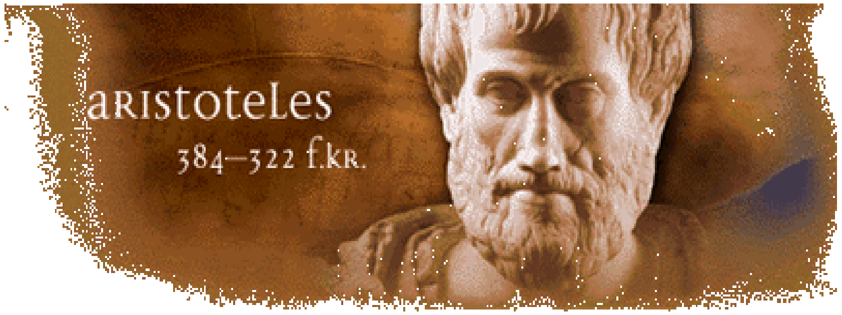
Example 2


UserGroup
assignedTo
Task
Metaclass level
User
Salesman
Order
Manager
Engagement
Secretary
Correspondence
Admin
AdminTask
Class level
John
George
Chris
Object level
attribute
instanceOf
isa

Ontology-Driven Conceptual Modelling

What is Ontology?

- A discipline of Philosophy
  - Meta-physics dates back to Aristotle (Metaphysics, IV, 1)
    - Tries to answer the questions: *What characterizes beings? Eventually, what is being?* How should things be classified?
  - Ontology dates back to 17th century
- The science of what is ("being qua being")
- Ontology derives from the Greek word "on" (being) and "logos" (word/speech, reason, or study-of).



What is *an* Ontology?

"A *formal*, *explicit* specification of a *shared conceptualization*" – Gruber (1994)

must be machine readable & understandable

types of concepts and constraints must be clearly defined

not private to some individual, but accepted by a group or community

an abstract model of phenomena in the world formed by identifying the relevant concepts of those phenomena

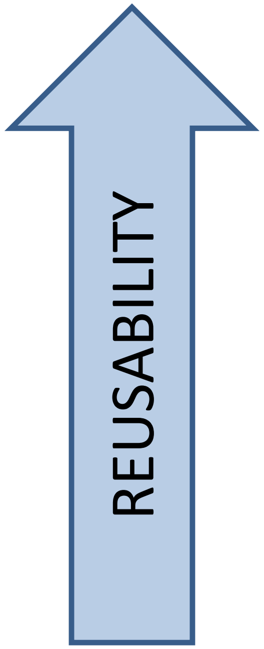
Important Distinctions

- A taxonomy – also from Greek "*taxis*" (arrangement, order, division) and "*nomos*" (law) - is usually applied to structure existing objects and assumed to be a single hierarchical architecture, while an ontology can apply to constructs and can have different kinds of architectures.
- An ontology is more than a taxonomy or classification of terms. Ontologies include richer relationships between terms. It is these rich relationships that enable the expression of domain-specific knowledge. This is a key distinction.

Kinds of ontologies

Ontologies can be classified according to their degree of reusability.

- General/Common ontologies
- Meta-ontologies
  - reusable across domains
- Domain ontologies
  - related to concepts in a specific domain
- Task ontologies
  - a systematic vocabulary of the terms used to solve problems associated with given tasks
- Domain-task ontology
  - task ontology reusable in a specific domain
- Application ontology
  - necessary knowledge for modeling a particular application domain



Why develop an ontology ?

- To **share a common understanding** of the structure of information
  - among people
  - among software agents
- To enable **reuse** of domain knowledge
  - to avoid “re-inventing the wheel”
  - to introduce standards to allow interoperability

Purpose and Benefits

Fundamentally, ontologies are used to improve communication between either humans or computers. Broadly, these may be grouped into the following three areas:

- to assist in **communication** between humans. Here, an unambiguous but informal ontology may be sufficient.
- to achieve **interoperability** among computer systems achieved by translating between different modelling methods, paradigms, languages and software tools. Here, the ontology is used as an interchange format.
- to improve the process and/or quality of engineering software systems.

Purpose and Benefits

Systems Engineering Benefits:

- *Re-Usability:* the ontology is the basis for a formal encoding of the important entities, attributes, processes and their inter-relationships in the domain of interest. This formal representation may be a re-usable and/or shared component in a software system.
- *Search*: querying and browsing semantically enriched information sources, serving as an index into a knowledge repository.
- *Reliability:* a formal representation also makes possible the automation of consistency checking, resulting in more reliable software.

Purpose and Benefits

Systems Engineering Benefits:

- *Specification:* the ontology can assist the process of identifying requirements and defining a specification for an IT system (knowledge based, or otherwise).
- *Maintenance*: use of ontologies in system development, or as part of an end application, can render maintenance easier in a number of ways.
- *Knowledge Acquisition:* using an existing ontology as the starting point and basis for guiding knowledge acquisition when building knowledge-based systems may increase speed and reliability.

Ontology languages and Conceptual Data Models

- An ontology language usually introduces **concepts** (classes, entities), **properties** of concepts (slots, attributes, roles), **relationships** between concepts (associations), and additional **constraints**.
- Ontology languages may be simple (e.g., having only concepts and taxonomies), frame-based (only concepts and properties), or logic-based (e.g. OWL).

Ontology Engineering vs Object-Oriented Modeling

An ontology

- reflects the structure of the **world**
- is often about the **structure** of concepts
- formal ontological classes are **sets** which can be defined in terms of other classes, attributes and associations
- property is a general term for attributes and associations.
- Open World Assumption

An OO class structure

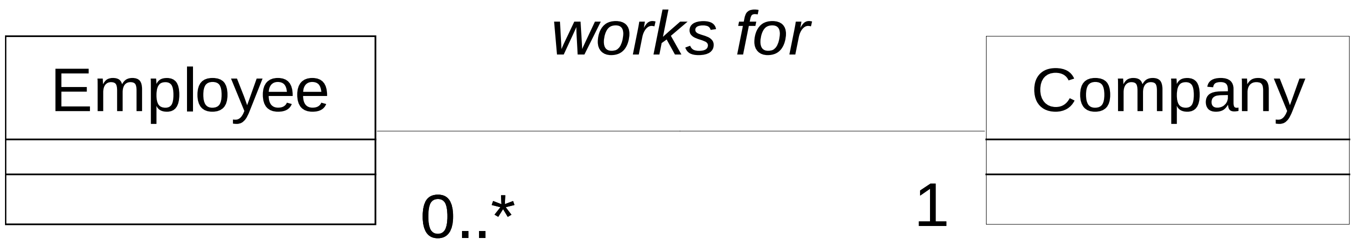
- reflects the structure of the **data and code**
- is usually about **behavior** (methods)
- Object-oriented classes are **templates** for constructing objects.
- attributes and associations have different meanings.
- Closed World Assumption

Open World Assumption means that the lack of knowledge of a fact does not immediately imply knowledge of the negation of a fact.

Closed World Assumption: what is not currently known to be true, is false.

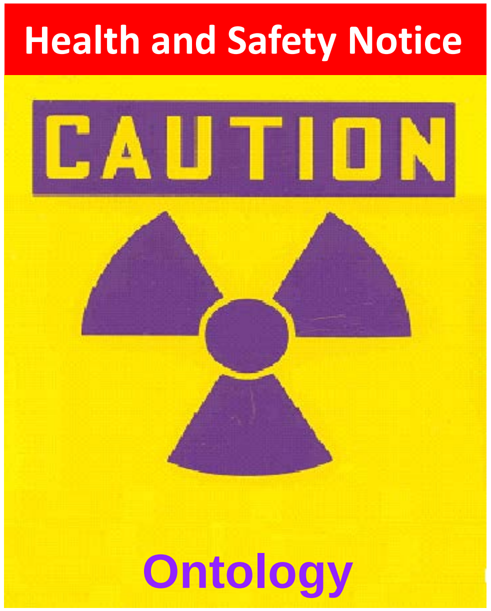
Logic: Open versus Closed

- Open (monotonic) logic gives different answers to queries than a closed logic.



- Suppose that John is the only Employee (no company info):
  - Closed world: violates constraint
  - Open world: John exists but the company he works for is not known.
- John works for Google and HP:
  - Closed world: violates constraint
  - Open world: Google and HP are the same! (inconsistency resolved!)

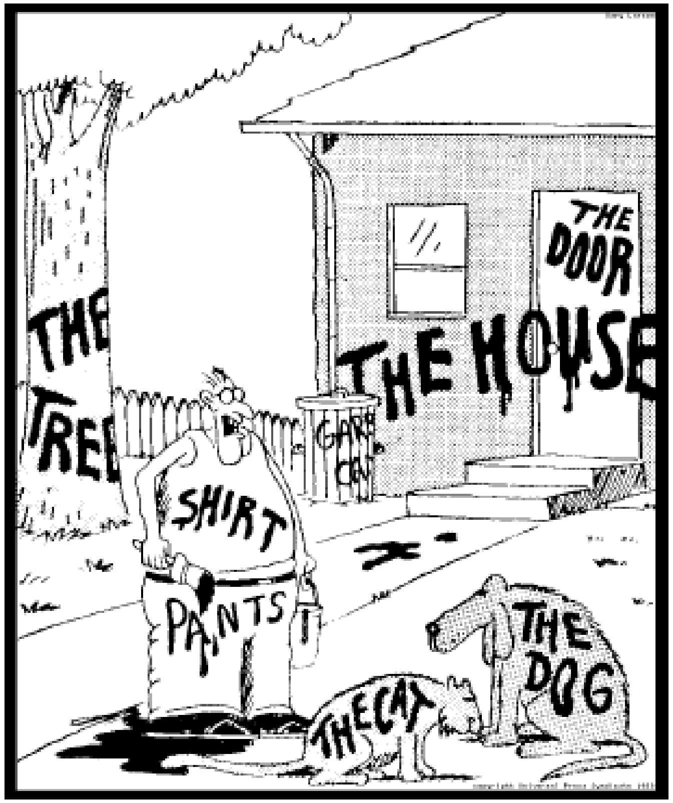
Ontology development



Disclaimer: This is not the only way that ontologies can be developed. In fact, many other possibilities exist.

Motivation

- Why create an ontology?
  - Sharing a common understanding of the information in a knowledge domain
  - Improving interoperability among applications that use the domain
  - Making domain assumptions explicit i.e. facilitate knowledge management and make changes (evolution) easier
  - Enabling re-use of the knowledge domain and allow new users to learn about the domain



"Now! *That* should clear up a few things around here!"

A complex process

- Ontology construction is not yet a well understood process
  - Size
  - Complexity
  - Shared understanding within a group of people with different backgrounds

- There is **no one correct** way to model a domain. There are always viable alternatives.

- There is not a single correct methodology for building an ontology. Best solutions depends on:
  - applications using the ontology
  - anticipated extensions

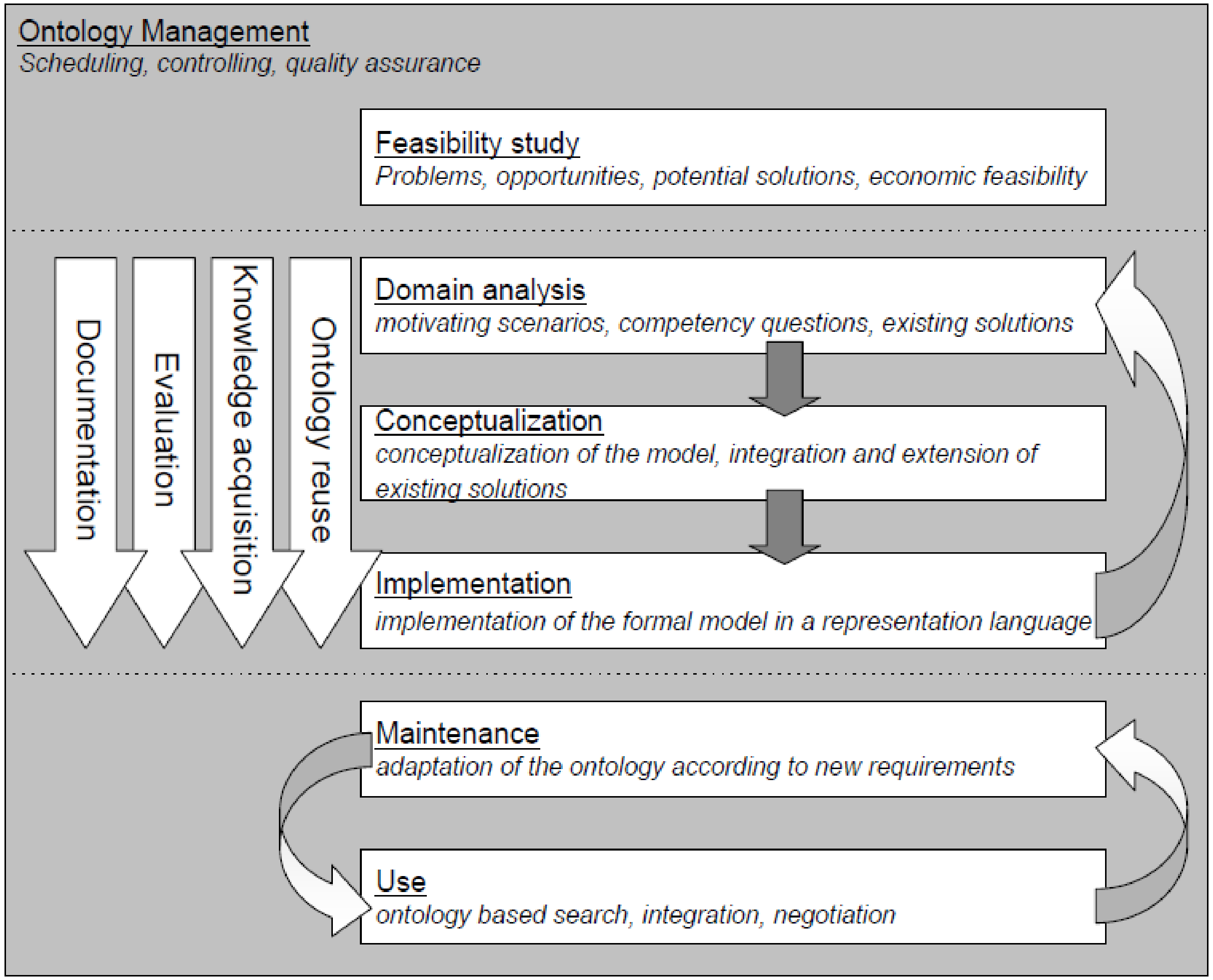
Ontology engineering definition

- "The set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies".*

- 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** classes can have
  - defining **constraints** on the property values
  - defining **individuals** and filling in values

**Source: Gómez-Pérez, A. et. al.: Ontological Engineering. Advanced Information and Knowledge Processing. Springer, 2003*

METHONTOLOGY

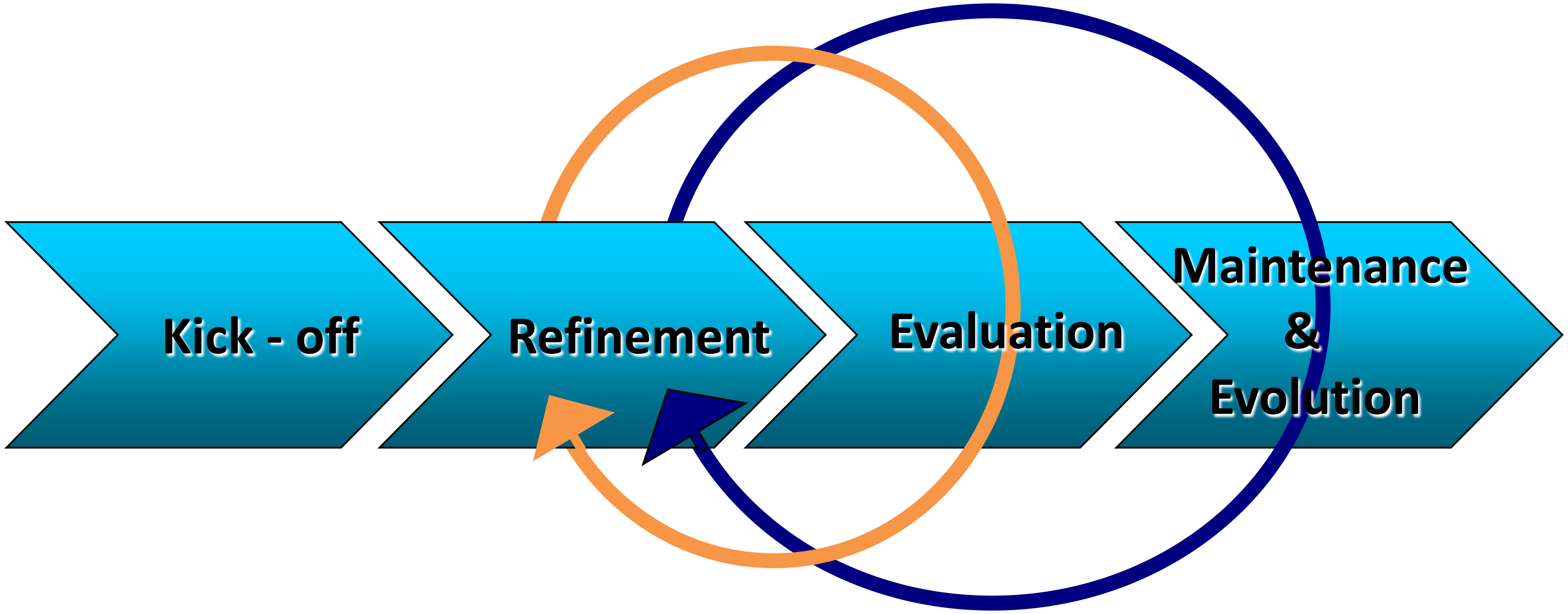

Ontology Management
Scheduling, controlling, quality assurance
Feasibility study
Problems, opportunities, potential solutions, economic feasibility
Domain analysis
motivating scenarios, competency questions, existing solutions
Conceptualization
conceptualization of the model, integration and extension of existing solutions
Implementation
implementation of the formal model in a representation language
Maintenance
adaptation of the ontology according to new requirements
Use
ontology based search, integration, negotiation
Documentation
Evaluation
Knowledge acquisition
Ontology reuse

Source: METHONTOLOGY, Gómez-Pérez, A. ,1996.

Methodology related to Software Engineering

On-to-Knowledge methodology

Methodology related to Knowledge Management Systems



On-to-Knowledge methodology

Ontology development is an iterative process.

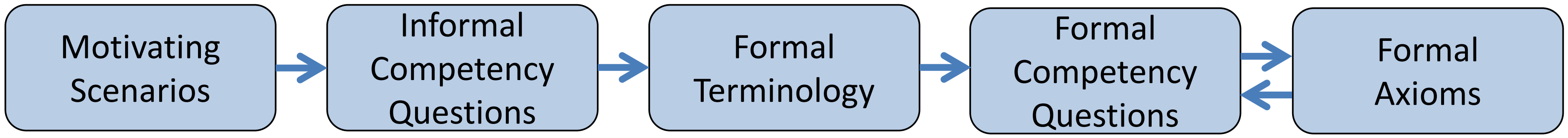
1. Kick-off phase: capturing the ontology
   a. Define the requirements specification
   b. Build a first draft version in a semi-formal description
2. Refinement phase: formal representation
3. Evaluation phase
4. Maintenance and Evolution phase

Kick-off phase

- What is the domain (scope) the ontology will cover?
- Who are the domain experts? Who will be the users?
- What is the ontology going to be used for?
- Brainstorming session
  - Involved people must collectively possess sufficient domain expertise
  - Quickly capture potentially relevant terms and phrases of the domain
- Identify knowledge sources
  - Existing ontologies
  - Domain experts (interviews, CQs, etc.)
  - Dictionaries, internal documents, etc.
- Potential users and usage scenarios
  - Identify roles
  - Identify how the ontology will be used
  - Identify terms and potential relations between them

Kick-off phase: Competency questions

- Competency questions (CQs): basic questions the ontology should be able to answer
  - Constitute expressiveness requirements
  - Indicate scope and content of the ontology
  - Used as reference during the evaluation
  - Ideally defined in a modular manner (higher level questions require the answer of lower level questions)
- Formal procedure for the generation and evaluation of the conceptualization



Kick-off phase: Ontology reuse

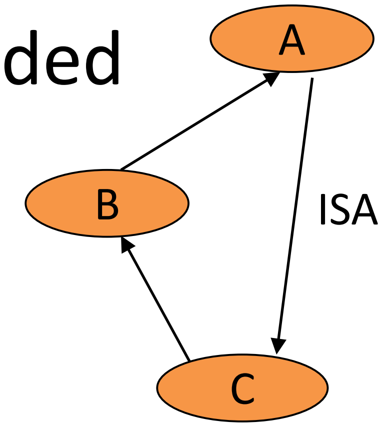
- Reusing existing ontologies
  - Worth considering what others have done
  - Due to compatibility requirements
- If applicable in the problem at hand:
  - Extend and refine to fit specific needs
  - Possibly integrate with other ontologies

Kick-off phase: Guidelines

- Design guidelines: provide guidelines for users not familiar with ontology modelling
  - Level of granularity (e.g. how many classes in each level)
  - Naming rules for concepts and relations
  - Concepts in the ontology should be close to objects (physical or logical) and relations in the target domain
- Clarity: the ontology should minimize ambiguity
- ISA modeling: a subclass represents a concept that is a specialization of the concept represented by the superclass
- Synonyms for the same concept should not represent different classes
- Siblings in the hierarchy must be in the same level of generality

Kick-off phase: Guidelines

- Cycles in the hierarchy should be avoided



- Multiple inheritance, disjoint classes

A new class or a new property value?

- A new class when
  - The concept is important in the domain
  - The concept becomes a restriction for some property in some other class
- A new property value when
  - The concept represents an extrinsic property of some class (number, color, location, etc.)

Kick-off phase: Guidelines

When to introduce a new class?

- When the concept to model
  - has additional properties
  - has different restrictions
  - participates in different relations

Modelling a concept as an instance or as a class?

- Individual instances represent the most specific concepts in the knowledge domain
- Depends on applications using the ontology

Kick-off phase: The different approaches

- Define the classes of the ontology
  - Top-down approach: Model the most general concepts first and refine them
  - Bottom-up approach: Find the most specific concepts and generalize to obtain higher level concepts
  - Middle-out approach: Identify the most important concepts and obtain the rest by specialization and generalization
- Pros and cons
  - Top-down: high quality, fine grained ontology but may not cover all domain information
  - Bottom-up: less quality, more complete ontology addressing all available domain information
  - Middle-out: more natural, concentrates on what is important and builds on from there, better controls the desired level of detail

Kick-off phase: Properties, relations & restrictions

- Define the relations of the ontology
  - Identify which domain terms represent relations or properties of some classes
  - A property should be attached to the most general class (inheritance)
  - Kinds of properties
    - Intrinsic (flavor, color)
    - Extrinsic (name, location)
    - Relations to other classes
- Restrictions on properties
  - Cardinality: how many values a property has
  - Value type: String, Number, Boolean, Class, etc.
  - Domain: the classes to which the property is attached
  - Range: the allowed classes, if the value-type is Class (object property)

Refinement phase

- Produce a formal representation of the ontology described in the previous phase
  - Refine the ontology
  - Choose a representation language (RDFS, OWL, etc.) based on:
    - Expressive power
    - Tool support for reasoning
  - Formalize the representation and produce a mature application-oriented ontology

Evaluation phase

- Evaluate the resulting ontology
  - Using as reference:
    - Requirements specifications
    - Competency questions
    - Usage scenarios
  - Test the ontology in the target application environment
    - Obtain usage patterns by monitoring user navigation
    - Recheck less frequently accessed parts for relevance
    - Possibly expand parts accessed with high frequency

Maintenance and Evolution phase

- Apply changes to the ontology while it evolves over time
  - Who is responsible for maintaining the ontology?
  - How the maintenance is going to be performed?
- Before each major change all possible effects must be thoroughly examined
- Thoroughly document according to purpose and scope.
  - One of the main barriers to effective knowledge sharing is the inadequate documentation
  - All important assumptions should be documented

Ontology Management

- Support for evolving ontologies in real-world applications and environments. Should consider issues like:
  - Storage
  - Change management
  - Alignment
  - Maintenance
  - Versioning

Ontology alignment-mapping

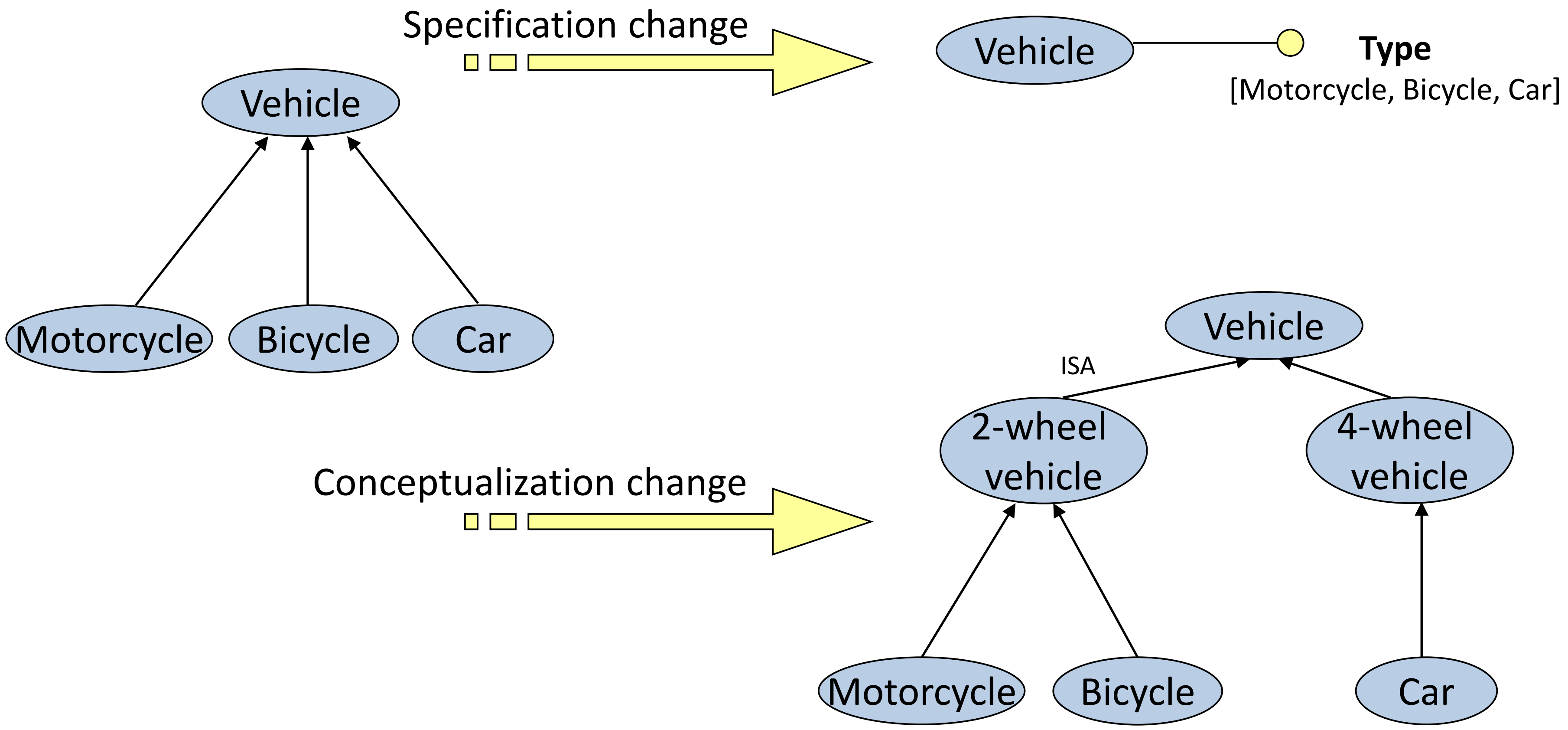
- **Definition:** Link different terminologies in domain specific ontologies
- **How:** Create bridges between separated pieces of knowledge

- What are bridges?
  - A connection between 2 sets of concepts (source set & target set)
  - Bridges are grouped into Maps
- A Map is a collection of bridges serving a single purpose
- Maps and Bridges form meta-ontologies or mapping ontologies
- Meta-ontologies together with domain ontologies may be used to perform cross-ontology information search and retrieval.

Support for ontology change

- Ontologies are not static but evolve over time
- Understanding the nature of potential changes is crucial
- Identifying the effects of potential changes and providing support for them is equally crucial
- Types of ontology changes:
  - Changes in the domain
    - Most frequent
    - Domain itself evolves over time
      - i.e. Merging of two university departments
  - Changes in the conceptualization
    - Depend on usage
    - Consensus needs to be reached many times over time
  - Changes in the specification
    - The format in which the conceptualization is formally recorded may be forced to change

Implication of ontology changes: An example



Effects of ontology changes

- Ontology evolution might cause incompatibilities: the original ontology cannot be replaced by the changed version without causing side-effects in the conforming data or the applications that use it
- Changes reflect on usage and the meaning and interpretation of data
- The real problem: Different meanings of incompatibility
  - The side effects caused by ontology changes depend on the use of the ontology
  - Failure to interpret all the data correctly through the changed ontology
- In ontologies built on top of other ontologies (e.g. imported ontologies), if the source ontologies change the resulting ones will be affected.

General guidelines for ontological engineering

- Try **NOT** to develop from scratch (re-use)
- Use existing data models and/or domain ontologies as a starting point
- Start with constructing an ontology of common concepts
- Make the purpose and context of the ontology explicit
  – E.g. data exchange between the medical domain
- Operational purpose/context with use cases
- Use multiple hierarchies to express different viewpoints on classes

General guidelines for ontological engineering

- Consider treating central relationships as classes
- Do not confuse properties with concepts
- Small ontologies are fine, as long as they meet their goal
- Don’t be overly ambitious: complete unified models are difficult
- Ontologies represent static aspects of a domain
  - Be careful when trying to include workflows
- Decide about the abstraction level of the ontology early on in the process.
  - E.g., use of the ontology only as a meta model

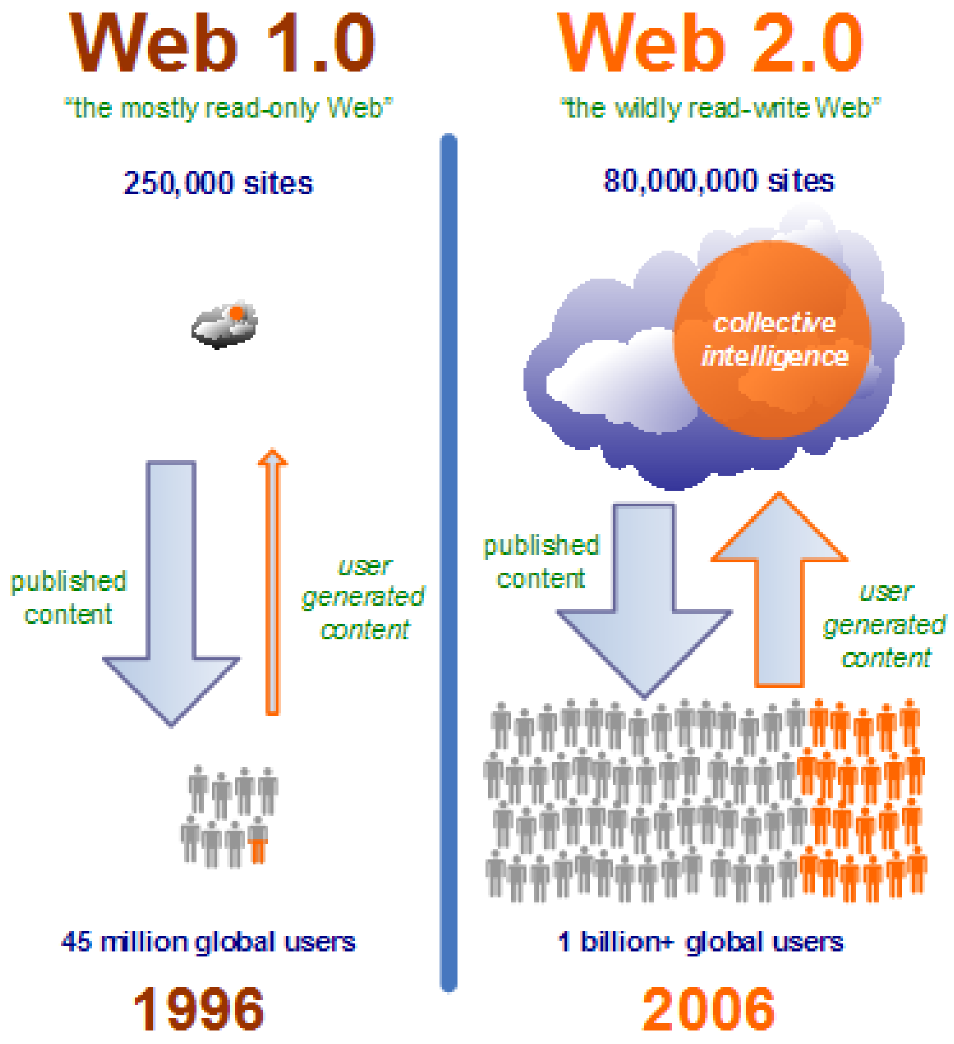
Introduction to the Semantic Web

(Εισαγωγή στον Σημασιολογικό Ιστό)

Today’s Web

- Most of today’s Web content is suitable for human consumption
  - Even Web content that is generated automatically e.g. from databases
- Typical Web users
  - seeking and making use of information, searching for and getting in touch with other people, reviewing catalogs of online stores and ordering products by filling out forms
- The Web would not have been the huge success without search engines (mainly keyword-based search engines)

Web 1.0 vs Web 2.0



- Cooperation
- Dynamicity
- Decentralized change
- Heterogeneity
- Multimedia content

Key Problems of Today's Web

- Keyword-Based Search Engines
  - High recall, low precision. Results are highly sensitive to vocabulary
  - Results are single Web pages
  - Human involvement is necessary to interpret and combine results, which are not easily accessible by other software tools
- Consider a typical web page. The markup consists of:
  - Rendering information (e.g. font size and colour)
  - Hyper-links to related content

Key Problems of Today’s Web

- Semantic content is accessible to humans but not (easily) to computers. The meaning of Web content is not machine-accessible: lack of semantics
- Need to add “semantics”
  - Agreement on the meaning of annotations
  - Shared understanding of a domain of interest
  - Formal and machine manipulable model of a domain of interest
- Web 3.0 ? The “Next Generation Web”
  - machine accessible semantics
  - machine-accessible meaning of information (reasoning services)

Two ways for computer to "understand"

- **Smarter machines**
  - Such as
    - Natural Language processing (NLP)
    - Audio Processing
    - Image Processing (IP)
    - Video Processing
    - … many many more
  - They all work fine alone, the problem is combining them
    - E.g., NLP meets IP
  - Not the Semantic Web approach
- **Smarter Data**
  - The Semantic Web approach
  - Make data **easier** for machines to publish, share, find and understand

What is the Semantic Web?

- “The Semantic Web is not a separate Web, but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”

“The Semantic Web”, Scientific American Magazine, 2001

What is the Semantic Web?

Key concepts:

- An extension of the current Web…
  - … where **information and services** are given **well-defined** and **explicitly represented meaning**, …
  - … so that it can be **shared** and used by **humans and machines**, ...
  - ... better enabling them to work in cooperation
- How?
  - Promoting information exchange by **tagging web content** with machine processable descriptions of its meaning.
  - And **languages**, **technologies** and **infrastructure** to do this

First steps towards the Semantic Web vision

- 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
- A prerequisite is a standard web ontology language
  - Need to agree common syntax before we can share semantics
  - Syntactic web based on standards such as HTTP and HTML

The Role of Ontologies on the Web

- Ontologies provide a shared understanding of a domain: semantic interoperability
  - overcome differences in terminology
  - mappings between ontologies
- Ontologies are useful for improving the accuracy of Web searches
  - search engines can look for pages that refer to a precise concept in an ontology
- Web searches can exploit generalization/specialization information
  - If a query fails to find any relevant documents, the search engine may suggest to the user a more general query.
  - If too many answers are retrieved, the search engine may suggest to the user some specializations.

Who works on ontologies?

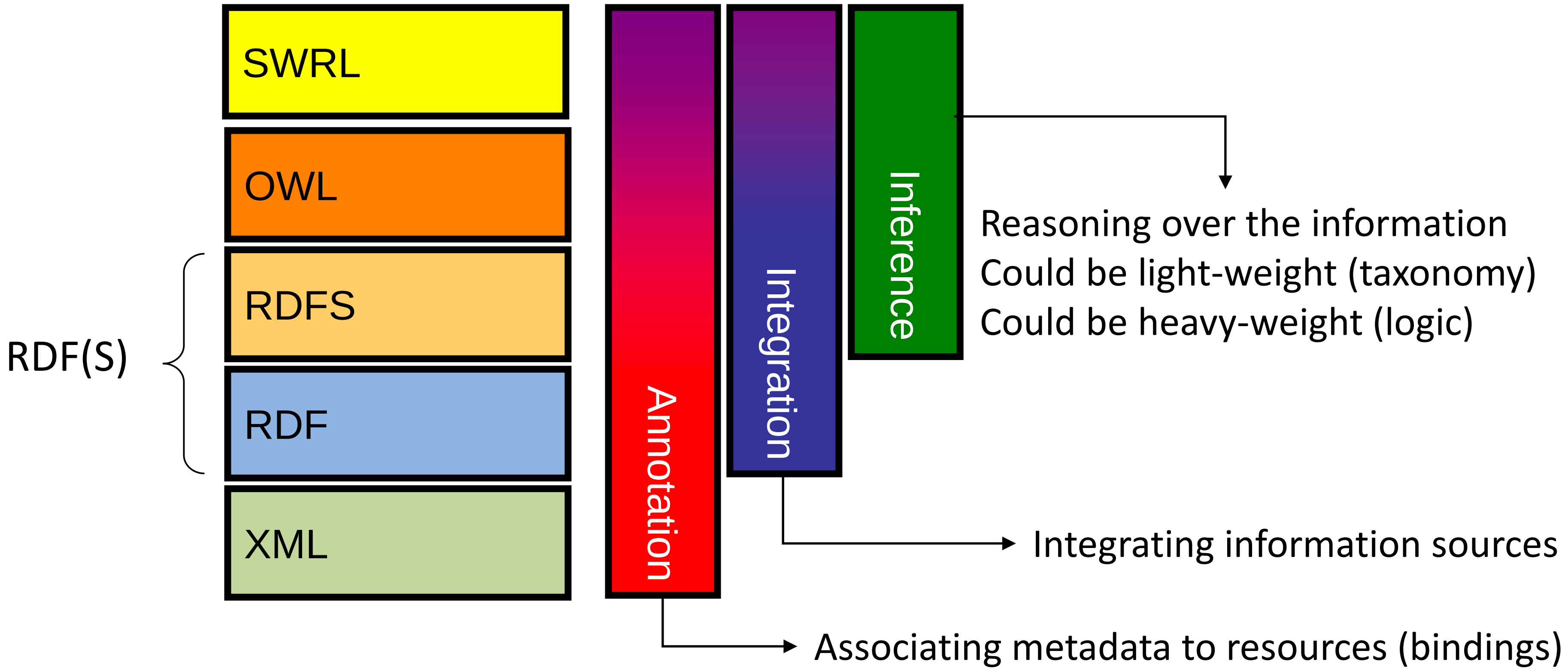
- Developed in **AI** to facilitate **knowledge sharing and reuse**
- A popular research topic in:
  - knowledge engineering
  - information systems
  - information retrieval
  - knowledge management
  - electronic commerce
  - multi-agent systems
  - natural language processing
  - bioinformatics and medical informatics
  - …

Requirements for an ontology language

- Extend existing Web standards
  - XML, RDF, RDFS, ...
- Easy to understand and to use
  - based on well known KR languages
- Formally specified
- Adequate expressive power
- Automatic support for reasoning

Ontology Languages

- A large amount of work on Semantic Web has concentrated on the definition of a collection or “stack” of languages.
  - Used to support the representation and use of metadata
  - Basic machinery that we can use to represent the extra semantic information needed for the Semantic Web


SWRL
OWL
RDFS
RDF
XML
RDF(S)
Annotation
Integration
Inference
Reasoning over the information
Could be light-weight (taxonomy)
Could be heavy-weight (logic)
Integrating information sources
Associating metadata to resources (bindings)

Ontology Languages

RDF / RDF Schema

- RDF is a data model for objects and relations between them
- RDF Schema describes properties and classes of RDF resources
- RDFS provides semantics for generalization hierarchies of properties and classes

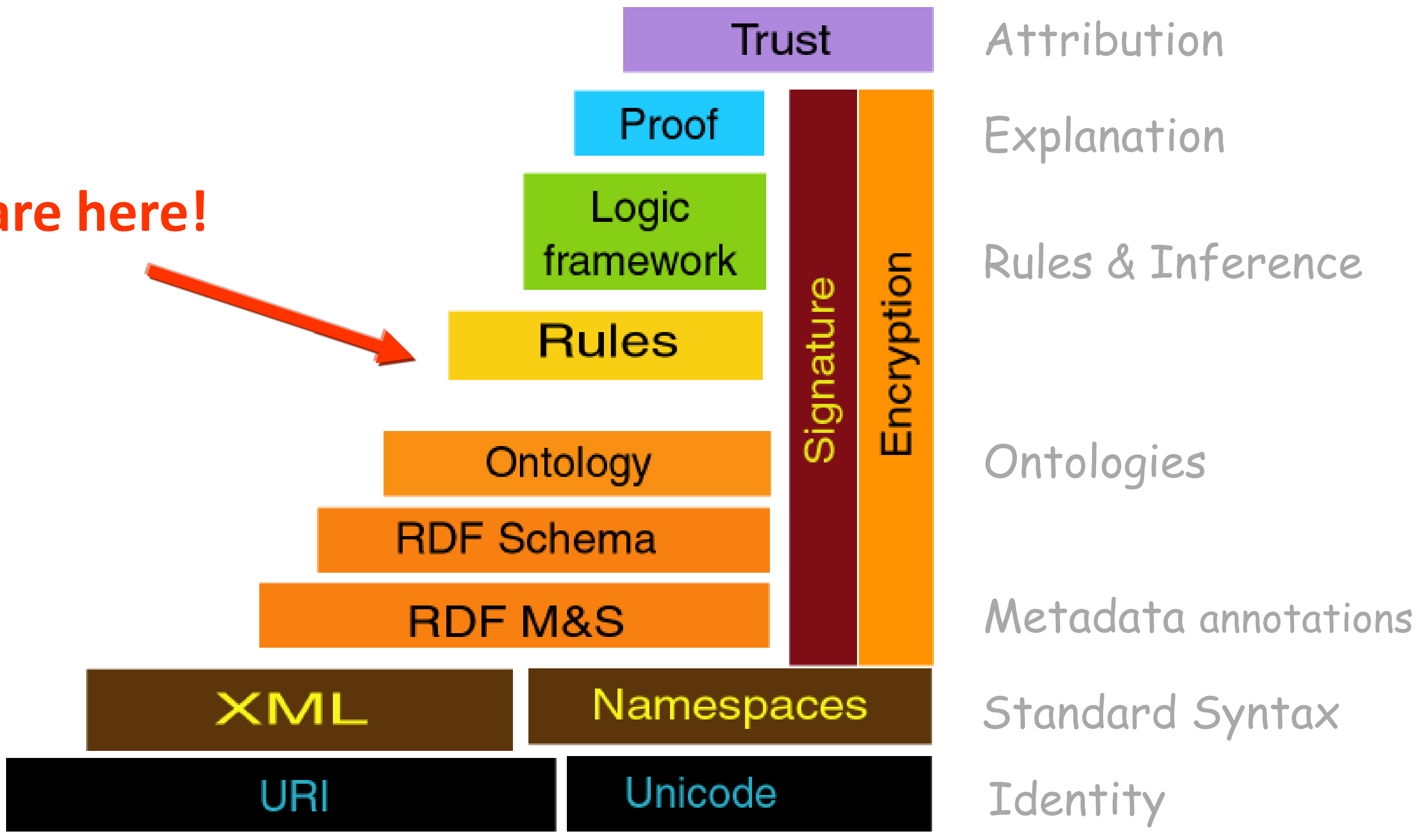
Ontology Languages

Web Ontology Language (OWL)

- A richer ontology language
  - relations between classes
    - e.g., disjointness
  - cardinality
    - e.g. “exactly one”
  - richer typing of properties
  - characteristics of properties
    - e.g. symmetry
  - ...

Layers of Languages

We are here!



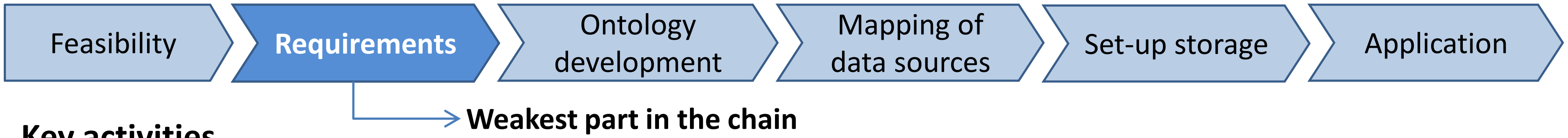
W3C

Semantic Web Layers

- XML layer
  - Syntactic basis
- RDF layer
  - RDF basic data model for facts
  - RDF Schema simple ontology language
- Ontology layer
  - More expressive languages than RDF Schema
  - Current standard: OWL
- Rules
  - SWRL (Semantic Web Rule Language)

Develop a Semantic Web application

Building a semantic application is feasible. However it requires deep technology insight and best practices for integration are still under development.



Key activities

- Feasibility study.
- Identification of data to be integrated.
- Test cases
- Rough application architecture.

- Detailed description of application.
- Evaluation criteria.
- Detailed description of data sources.
- Application scenarios

- Logical model
- Ontology development
- Sample queries
- Documentation of ontology
- Maintainability of ontology

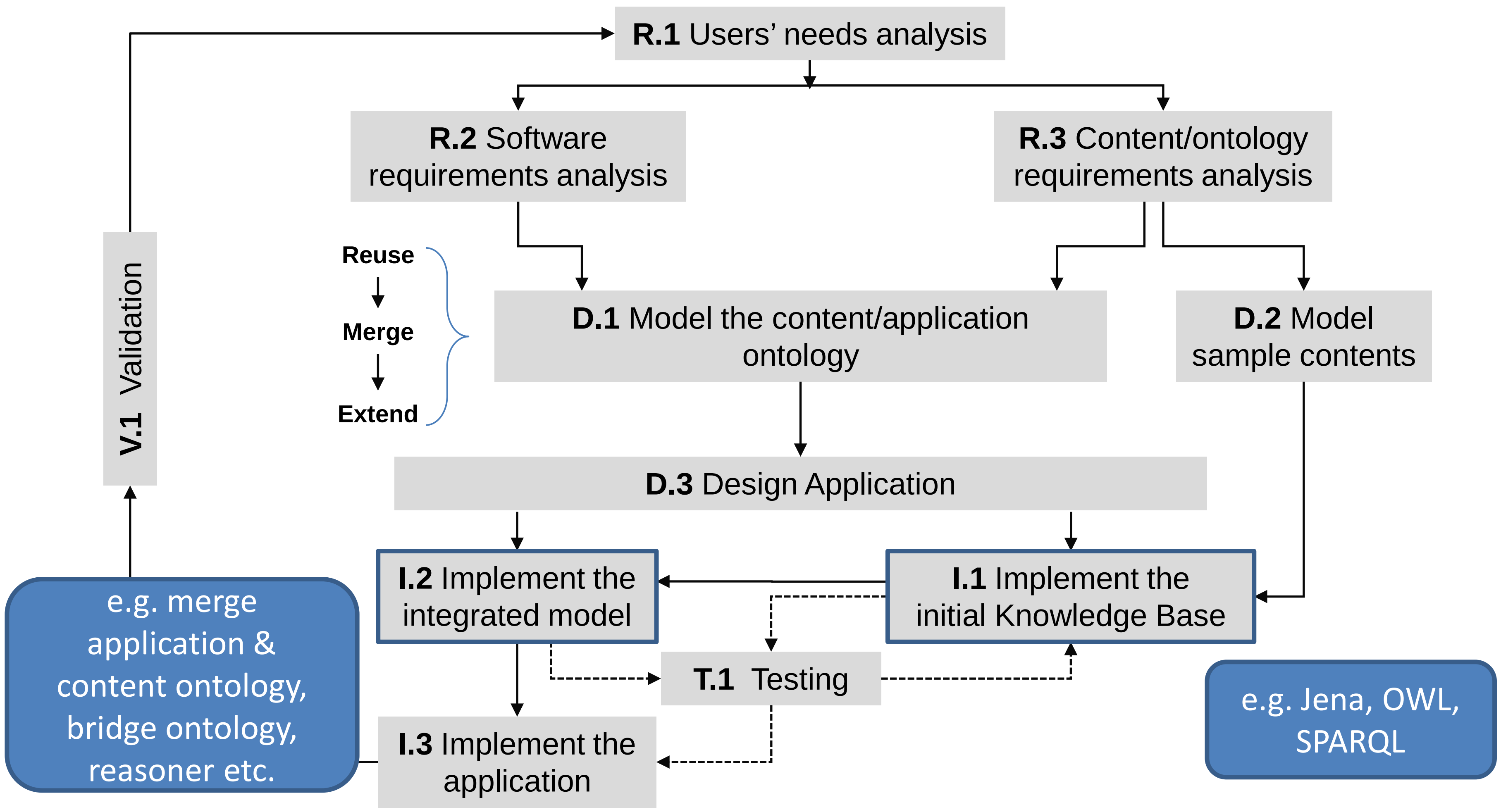
- Inserting data sources to ontology.
- Instantiation
- Ontology refinement.

- Integration with data sources.
- Scalable storage solution.
- Ontology management.
- Maintainability

- User interface.
- Process to support application scenarios.
- Infrastructure (possibly SOA).
- Proof of concept.

Stakeholders (who should do what)

Develop a Semantic Web application


R.1 Users' needs analysis
R.2 Software requirements analysis
R.3 Content/ontology requirements analysis
Reuse
Merge
Extend
D.1 Model the content/application ontology
D.2 Model sample contents
V.1 Validation
D.3 Design Application
I.2 Implement the integrated model
I.1 Implement the initial Knowledge Base
T.1 Testing
I.3 Implement the application
e.g. merge application & content ontology, bridge ontology, reasoner etc.
e.g. Jena, OWL, SPARQL

Logic and Reasoning with Ontologies

Key terms

- **Logic**: The discipline that studies the principles of reasoning. Step-by-step thinking about how a problem can be solved or a conclusion can be reached.
  - Formal languages for expressing knowledge
  - Well-understood formal semantics
  - Automated reasoners can deduce (infer) conclusions from the given knowledge
- **Inference**: A conclusion drawn from true or assumed-true facts. Derivation of new knowledge from existing knowledge and axioms within a single step.
- **Reasoning**: The capacity for consciously making sense of the world based on logic and evidence. Inferences are steps in reasoning.

A simple Inference example

prof(X) $\rightarrow$ faculty(X)

faculty(X) $\rightarrow$ staff(X)

prof(John)

We can deduce the following conclusions:

faculty(John)

staff(John)

prof(X) $\rightarrow$ staff(X)

Reasoners

- AI researchers develop automated inference systems to emulate human inference.
- A **reasoner** is a piece of software able to infer logical consequences from a set of asserted facts or axioms.
- The inference rules are commonly specified by an ontology language, and often by a description logic language.
- Many reasoners use first-order predicate logic to perform reasoning.

Reasoning Support for OWL

- Recently automatic reasoners found a new field of application in the Semantic Web. Semantics is a prerequisite for reasoning support.
- **Reasoners** play a **vital role** in developing and using an OWL ontology. (e.g. automated reasoners such as *Pellet, FaCT++, HerMiT* etc.)
- It's one of the reasons why a specification of ontologies/knowledge bases needs to be formal.
- List of OWL reasoners:
  - https://www.w3.org/2001/sw/wiki/OWL/Implementations
  - http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/

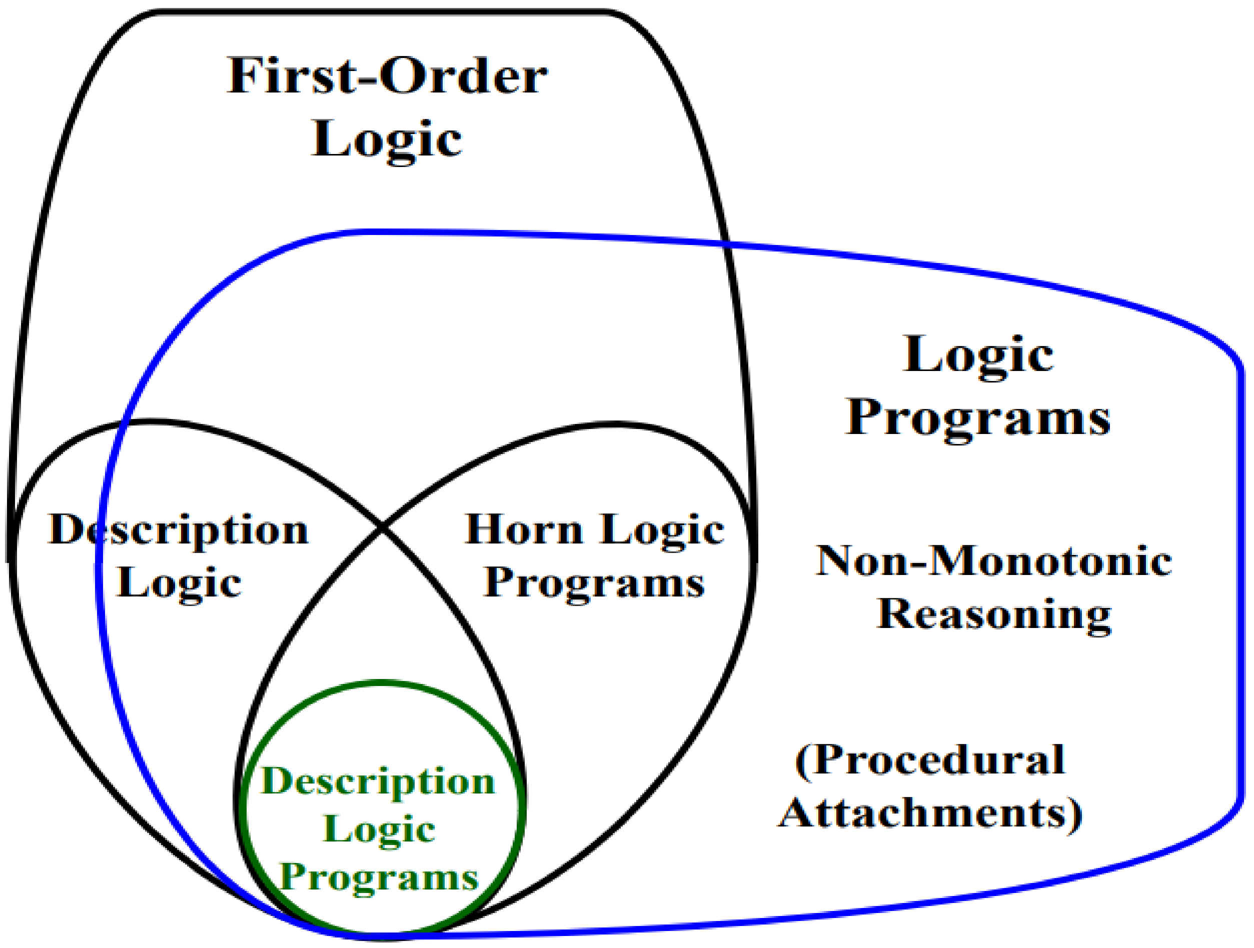
Reasoning Support for OWL

- Formal semantics and reasoning support are usually provided by
  - mapping an ontology language to a known logical formalism
  - using automated reasoners that already exist for those formalisms
- OWL is (partially) mapped on a description logic, and makes use of implemented OWL reasoners.
- Description logics are a subset of predicate logic (κατηγορηματική λογική) for which efficient reasoning support is possible.

Description logic

- **Description logics** (**DL**) are a family of formal knowledge representation languages. DLs are equipped with a formal, logic-based semantics. They are logics based on descriptions of concepts or terms.
- Many DLs are more expressive than propositional logic (προτασιακή λογική) but less expressive than first-order logic (FOL).
- The core reasoning problems for DLs are (usually) decidable, and efficient decision procedures have been designed and implemented for these problems.
- DLs are used in Artificial Intelligence to describe and reason about the relevant concepts of an application domain.
- It is important to provide a logical formalism for ontologies and the Semantic Web: the Web Ontology Language (OWL) is based on DLs.
- DLs, in general, are subsets of FOL.

Expressive overlaps



Knowledge Representation with DLs

- A knowledge base (KB) consists of two components: a TBox and an ABox.
  - TBox: In the TBox we define concepts (terms) of the application domain, their properties and their relations to each other. This is called schema knowledge.

Examples:

Woman ≡ Person ⊓ Female

GreekUniversityAlumni ≡ Person ⊓ ∃isAlumniOf.GreekUniversity

Man ⊑ Person

Man ⊑ ¬Woman

Male ⊑ ¬Female

Knowledge Representation with DLs

– ABox: In the ABox we make assertions about the individuals in the application domain: membership in classes and role filling (instance level knowledge).

Examples:

Female(ANNA),
hasChild(ANNA, JOHN),
(Person $\sqcap$ ¬Male)(ANNA),
((>3 hasChild) $\sqcap$ Male)(JOHN),
GreekUniversityAlumni(JOHN),
(>3 hasChild.(Person $\sqcap$ ∀isAlumniOf.GreekUniversity))(ANNA)

- The semantics of TBox or ABox sentences is defined by set theory (set membership, inclusion, equality, disjointness etc.). A concept in DL is a class in OWL. A role in DL is a property in OWL.

Translating DLs to FOL

- The FOL expression of

  $\text{Male} \sqsubseteq \neg\text{Female}$

  is

  $(\forall x)(\text{Male}(x) \Rightarrow \neg\text{Female}(x))$

- The FOL expression of

  $\forall\text{hasChild}.(\exists\text{isAlumniOf}.\text{GreekUniversity}) \sqsubseteq \text{ProudParent}$

  is

  $(\forall x)((\forall y)(\text{hasChild}(x, y) \Rightarrow (\exists z)(\text{isAlumniOf}(y, z) \land \text{GreekUniversity}(z))) \Rightarrow \text{ProudParent}(x))$

What an ontology reasoner does

- By reasoning we mean deriving (automatically) facts that are not explicitly expressed in the ontology or in a knowledge base.
- Some examples:
  - Consistency check
    e.g. disjointed classes *Human* and *Dog*, Individual: *Max* belongs both to *Human* and *Dog* → inconsistent ontology
    - Checking the consistency of a set of axioms is a mandatory first step before any other reasoning service
    - Fix the inconsistency before reasoning. Why? Because any consequence can be inferred from inconsistency
  - Subsumption check (class inferences)
    e.g. A bus driver is a person that drives a bus. A bus is a vehicle. A bus driver must be a driver → the subclass is inferred

What an ontology reasoner does

- Some examples:
  - Class membership
    e.g. If x is an instance of a class *B*, and *B* is a subclass of *A*, then we can infer that x is an instance of *A*
  - Equivalence of classes
    e.g. If class *A* is equivalent to class *B*, and class *B* is equivalent to class *C,* then *A* is equivalent to *C*, too
  - Classification
    e.g. certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A
  - Query answering e.g. retrieve instances, subclasses/superclasses etc.

Inference Example (using OWL)

- Input axioms
  1. Employee equivalentTo (CivilServant or Contractor)
  2. CivilServant disjointWith Contractor
  3. isEmployeeOf inverseOf hasEmployee
  4. isEmployeeOf domain Employee
  5. Person345 type CivilServant
  6. Person345 isEmployeeOf Organization121
- Some inferences

CivilServant subClassOf Employee	{ 1 }
Person345 type Employee	{ 1, 5 }, { 4, 6 }
Person345 type not Contractor	{ 2, 5 }
Organization121 hasEmployee Person345	{ 3, 6 }

Specializations of Predicate Logic

- RDF/S and OWL (OWL-Lite and OWL-DL, not OWL-Full) are specializations of predicate logic
  - correspond roughly to a description logic
- Trade-off between the expressive power and the computational complexity:
  - The more expressive the language, the more computationally expensive it becomes to draw conclusions and the less efficient the corresponding proof systems
  - Drawing certain conclusions may become impossible (non-computable)

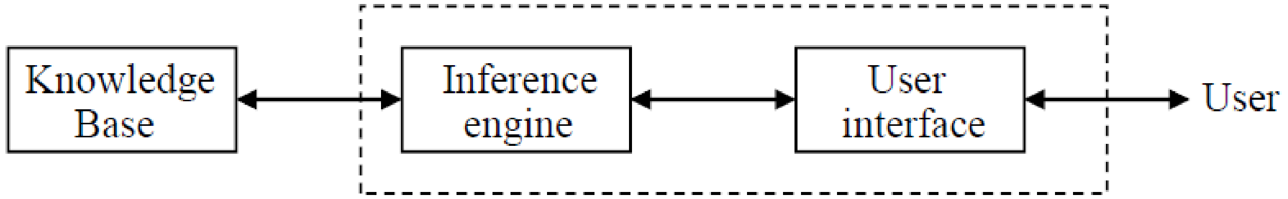
Rules and Rule-based systems

Έμπειρα Συστήματα (Expert Systems)

- Ομοιότητα με τον τρόπο που λειτουργούμε στις καθημερινές μας δραστηριότητες
- Βασικότερο πρόβλημα στην κατασκευή τέτοιων συστημάτων είναι η απόκτηση της γνώσης (knowledge acquisition) από τον άνθρωπο-ειδικό
- Έχουν πολλές εφαρμογές στον έλεγχο & σχεδίαση συστημάτων υποστήριξης αποφάσεων (π.χ. υποβοήθηση ιατρικής διάγνωσης και προτάσεις θεραπευτικής αγωγής, αυτόματος πιλότος, έλεγχος αντιδραστήρων, διαλογικά συστήματα, επιχειρηματικές αποφάσεις, χρηματοοικονομικά εργαλεία κλπ).
- Π.χ. ρομποτικά συστήματα:
  - θέση-κατάσταση (με χρήση αισθητήρων-sensors)
  - απόφαση/επιλογή κάποιας ενέργειας (π.χ. κίνηση)
  - εκτέλεση κάποιας ενέργειας
  - μετάβαση σε νέα θέση-κατάσταση κλπ.

Έμπειρα Συστήματα (Expert Systems)

- Δομή των εμπείρων συστημάτων
  - Μηχανή συμπερασμού (Inference engine)
  - Σύστημα επικοινωνίας με το χρήστη (User interface)
  - Βάση γνώσης (Knowledge base)



- Αναπαράσταση γνώσης μέσω if-then κανόνων ή κανόνων παραγωγής (production rules)
- Πλεονεκτήματα if-then κανόνων
  - Σταδιακή κατασκευή βάσης γνώσης
  - Διευκόλυνση απαντήσεων σε ερωτήσεις “πώς” και “γιατί”
  - Δυνατότητα επέκτασης για χειρισμό αβέβαιης πληροφορίας
  - Ικανοποιητικό μέσο κωδικοποίησης της γνώσης ενός ανθρώπου-ειδικού

Horn logic

- A Horn clause is a **logical** formula in a rule-like form which gives it useful properties (e.g. use in logic programming)
- A rule has the form: **A1, . . ., An → B**
  - **Ai** and **B** are atomic formulas
- There are 2 ways of reading such a rule:
  - Deductive rules: If **A1,..., An** are known to be true, then **B** is also true
  - Reactive rules: If the conditions **A1,..., An** are true, then carry out the action **B**

Description Logics vs Horn Logic

- Neither of them is a subset of the other
- It is impossible to assert that a person X who is brother of Y is uncle of Z (where Z is child of Y) in OWL
  - This can be done easily using rules:

    brother(X,Y), childOf(Z,Y) $\rightarrow$ **uncle(X,Z)**
- Rules cannot assert the information that a person is either a man or a woman
  - This information is easily expressed in OWL using disjoint union

Monotonic vs Non-monotonic Rules

Example: An online seller wants to give a special discount if it is a customer's birthday

Solution 1

R1: If birthday, then special discount

R2: If not birthday, then not special discount

- But what happens if a customer refuses to provide his birthday due to privacy concerns?

Solution 2

R2': If birthday is not known, then not special discount

- Solves the problem but: The premise of rule **R2'** is not within the expressive power of predicate logic
- **R2'** is a non-monotonic rule
- The new kind of rule system will find application in cases where the available information is incomplete

Monotonic Rules – Syntax

loyalCustomer(X), age(X) > 60 → discount(X)

- We distinguish some ingredients of rules:
  - variables which are placeholders for values: **X**
  - constants denote fixed values: **60**
  - predicates relate objects: **loyalCustomer, >**
  - functions which return a value for certain arguments: **age**

Semantic Web Rule Language (SWRL)

- SWRL is intended to be the rule language of the Semantic Web.
- It includes a high-level abstract syntax for Horn-like rules.
- All rules are expressed in terms of OWL concepts (classes, properties, individuals).
- SWRL rules are saved as part of the ontology.
- SWRL aims to overcome some limitations of OWL.

Semantic Web Rule Language (SWRL)

- A rule in SWRL has the form
  - $\mathbf{B_1, \ldots , B_n \rightarrow A_1, \ldots , A_m}$
  - Commas denote conjunction (σύζευξη) on both sides
  - $\mathbf{A_1, \ldots , A_m, B_1, \ldots , B_n}$ can be of the form **C(x)**, **P(x,y)**, **sameAs(x,y)**, or **differentFrom(x,y)** where **C** is an OWL description, **P** is an OWL property, and **x**, **y** can be variables, OWL individuals, or OWL data values

SWRL Properties

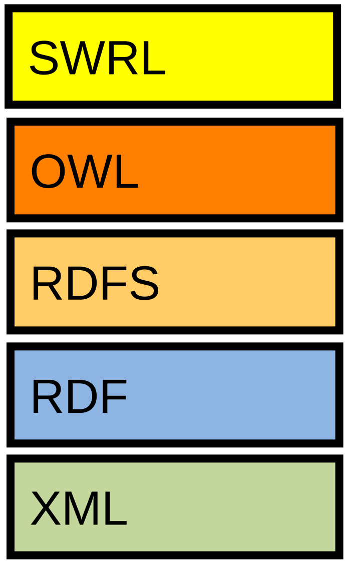
- Expressions, such as restrictions, can appear in the head or body of a rule
- This feature adds significant expressive power to OWL, but at the high price of undecidability
- SWRL combines/unites the expressivity of Description Logic and Horn Logic
- The challenge is to identify sublanguages of SWRL that find the right balance between expressive power and computational complexity
  - DL-safe rules

Why we need a rule language

- A rule language is needed for several reasons:
  - Existing rule sets can be reused.
  - Expressivity can be added to OWL
  - It is easier to read and write rules with a rule language.
- It is possible to combine OWL reasoning (e.g. inference capabilities through the OWL characteristics of properties, like inversion, symmetry and transitivity) and inference capabilities through the SWRL rules.
- SWRL can work with OWL reasoners and there is an increasing tool support (e.g. Pellet, HermiT, SWRLTab or SQWRLTab for Protégé).

Summary

- Horn logic is a subset of predicate logic that allows efficient reasoning, orthogonal to description logics
- Horn logic is the basis of monotonic rules
- Non-monotonic rules are useful in situations where the available information is incomplete
- SWRL combines OWL with Horn rules and is a much richer language



Rule engines

- A rule engine is an application (pattern matcher) which executes one or more actions based on **rules** and **facts** (e.g. from a knowledge base).
- Rule engines reduce an application’s maintenance as well as extensibility costs, lowering the complexity of components that implement complex business logic.
- Most rule engines use the **RETE** algorithm (or a version it) which is a **pattern matching algorithm** for implementing production rule systems. It is used to determine which of the system's rules should **fire** based on a set of facts.

Why use a Rule Engine

- Separates application from dynamic logic
  - Rules can be modified by different groups
  - No need to recompile or redeploy
  - All rules are in one place
- Declarative programming (Δηλωτικός Προγραμματισμός)
  - Human readable and anyone can easily modify rules.
- Centralization of knowledge e.g. repository of business policy
- Provides speed and scalability

Types of rule engines

- Types of rule engines (generally) differ in how rules are scheduled for execution.
- Most rules engines use **forward chaining**, which can be further divided into two types:
  - The first type processes so-called production/inference rules. These types of rules are used to represent behaviors of the type IF condition THEN action.
  - The other type of rule engine processes so-called reaction/Event Condition Action rules. The reactive rule engines detect and react to incoming events and process event patterns.

Types of rule engines

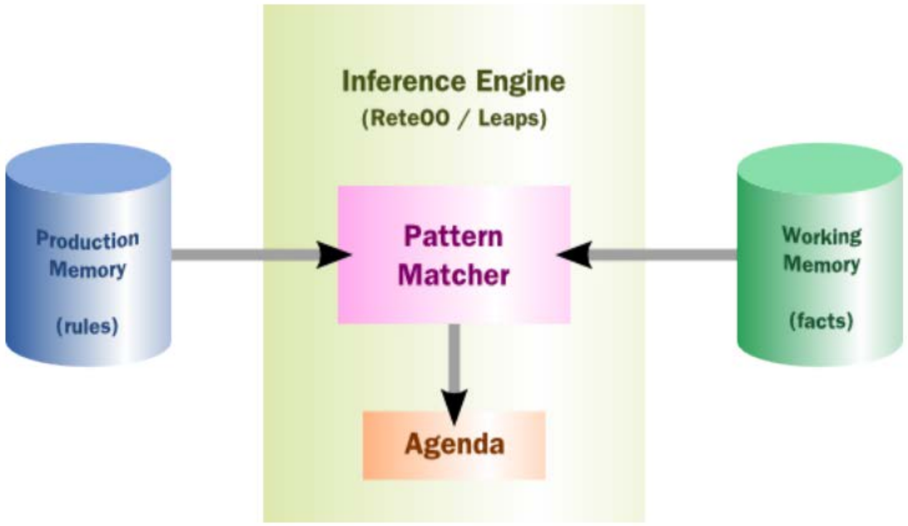
- Most popular commercial rule engines have both production and reaction rule capabilities, although they might emphasize one type over another.
- Some rules engines support **backward chaining**. In this case a rules engine seeks to resolve the facts to fit a particular goal. It is often referred to as being goal driven because it tries to determine if something exists based on existing information.
- Another kind of rule engine automatically switches between back- and forward-chaining several times during a reasoning run.

Popular business rule engines: Drools

- Drools is a Business Rules Management System (BRMS). It mainly consists of a core Business Rules Engine (BRE), a web authoring and rules management application (Drools Workbench) and an Eclipse IDE plug-in for core development in Java.
- Drools has an enhanced and optimized implementation of the RETE algorithm for object oriented systems (called ReteOO).
- The Business Rules Engine primarily uses the Drools Rule Language (DRL)
- Drools is a Rule Engine that uses the rule-based approach to implement an Expert System.

Popular business rule engines: Drools

- A Production Rule is a two-part structure using First Order Logic for reasoning over knowledge representation.
- The inference engine matches the rules against the facts (objects) in memory: when <conditions> then <actions>.
- The rules are loaded into the *Production Memory* and are available at all times.
  Facts are asserted into the *Working Memory* where they may then be modified or retracted. The *Agenda* manages the execution order of the conflicting rules using a conflict resolution strategy.



Other rule engines: Jess

- Jess is a rule engine and scripting environment written entirely in Java. Jess is small, light, and one of the fastest rule engines available.
- Jess allows the developers to build Java software that can reason using knowledge you supply in the form of declarative rules. Jess can straightaway manipulate and reason Java objects.
- Jess uses an enhanced version of the Rete algorithm to process rules.
- It also has an Eclipse IDE plug-in.

Other rule engines: OpenRules

- OpenRules is a general purpose Business Rules and Decision Management System (BRDMS). It allows subject matter experts and software developers to create, test, execute, and maintain enterprise-class decision support applications.
- The goal of OpenRules is to create domain-specific decision modeling constructs (diagrams, decision tables) oriented to business users/business analysts for decision modeling.
- Major Functional Components:
  - Rule Repository: Maintained by Business Analysts with Excel or Google Docs
  - Rule Engine: Executing Decision Models
  - Rule Learner: Discovering Rules, Predictive Analytics
  - Rule Solver: Constraint Satisfaction, Solving Optimization Problems
  - Rule Dialog: Developing dynamic rules-based Web Questionnaires
  - Finite State Machines: Event Processing

SWRL for Protégé

- The SWRLTab Plugin is a SWRLAPI-based plug-in (distributed within Protégé) that provides a set of graphical interfaces for managing SWRL rules (and SQWRL queries) in the Protégé Desktop Ontology Editor (version 5.0 and higher).
- It's a Protégé-OWL development environment for working with SWRL rules which can support editing and execution of rules.
- Its main components include:
  - A SWRL Editor for editing SWRL rules (and SQWRL queries).
  - A Drools-based SWRL Rule Engine and associated SWRL Drools Tab for executing SWRL rules.
  - A SQWRL Query Tab for executing SQWRL queries.
  - An OWL 2 RL Tab for interacting with SWRLAPI-based OWL 2 RL reasoners.

Short Protégé tutorial - example