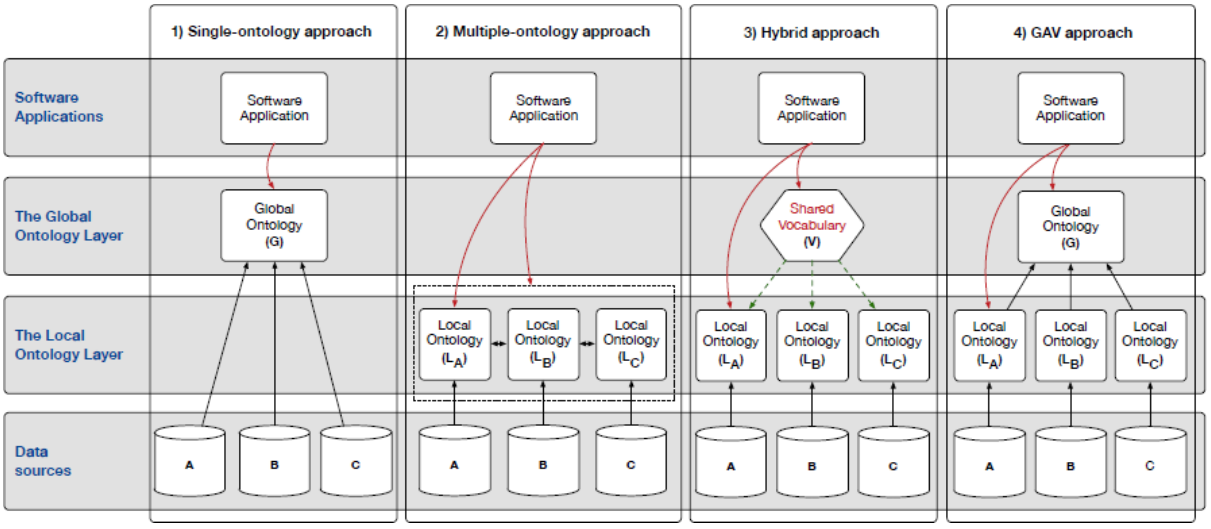
Semi-Automatic Information and Knowledge Systems - Oral Exam Topics 2021S

1 Ontology-based data Integration

Ontology-based data integration (OBDI) refers to the use of (potentially several layers of) ontologies that capture implicit knowledge across heterogeneous data sources to achieve semantic interoperability between these sources.



- Data sources - (heterogeneous) local data repositories, which need to be integrated.
- The local ontology layer – ontologies which represent the content of each individual data source repository.
- The global ontology layer – ontologies which are semantically sufficiently broad to represent the data from all data sources to be integrated.
- The software applications layer - applications, which access the data integrated with OBDI.

Single-ontology OBDI

Relies on a single global ontology to integrate all data sources. Implementation steps:

1) Define a single global ontology *G* and
2) transform source data from *A*, *B*, and *C* into the global ontology *G*.

\+ straightforward implementation
\- susceptible to changes in data sources
\- only allows access in global data

Multiple-Ontology OBDI

Involves a local ontology per data source and an alignment of these ontologies with each other using semantic mappings. Implementation steps:

1) Create local ontologies L_A, L_B, L_C for data sources *A*, *B*, and *C*, respectively.

2) Transform source data of *A*, *B*, and *C* according to their respective local ontologies, and
3) Establish semantic mappings between related ontologies.

+ Allows access to local and aggregated ontologies
+ supports (simple) mappings via semantic relations
- Needs to provide a new local ontology and its mapping to existing local ontologies each time a new data source is connected

Hybrid OBDI

is characterized by definitions of a local ontology per data source. This approach defines a shared vocabulary to be used and extended within local ontologies. Implementation steps:

1) Define a shared vocabulary *V* that contains basic terms/concepts of the domain,
2) Create three local ontologies L_A, L_B, L_C f by using and/or extending the shared vocabulary *V* for data sources *A*, *B*, and *C* respectively (local ontologies already are specializations of the shared vocabulary, so no mapping needed for that), and
3) Transform/annotate source data from *A*, *B*, and *C* according to local ontologies L_A, L_B and L_C.

+ Allows access to (restructured) local ontologies based on the shared vocabulary
+ Supports (simple) mappings via vocabulary refinement
- It is not possible to preserve the original local ontologies if they are not compatible with the shared vocabulary

GAV OBDI

The GAV OBDI requires the definition of one local ontology per data source and mapping definition between local and global data sources. Implementation steps:

1) Creation of three independent local ontologies L_A, L_B, and L_C (or reuse of existing local ontologies) for data sources *A*, *B*, and *C* respectively.
2) Transformation of source data in local sources *A*, *B*, and *C* according to local ontologies L_A, L_B, and L_C.
3) Development of a global ontology *G*. *G* represents a set of common concepts relevant to the local ontologies, and
4) Definition of independent mappings between a local repository (i.e., L_A, L_B, and L_C) and the global ontology *G* to facilitate data transformation from local ontologies to the global ontology.

+ Allows access to (original) local ontologies based on the shared vocabulary
+ Supports complex mappings via queries and rules
- Rather high implementation efforts due to the need of defining mappings between global and local ontologies

Research Gaps

Despite the availability of OBDI and all its variants choosing the most appropriate OBDI variant, as well as particular suitable technologies is still challenging. Appropriate choices are mainly determined by the specific characteristics of the problem setting, such as data source heterogeneity or mapping complexity between the data sources.

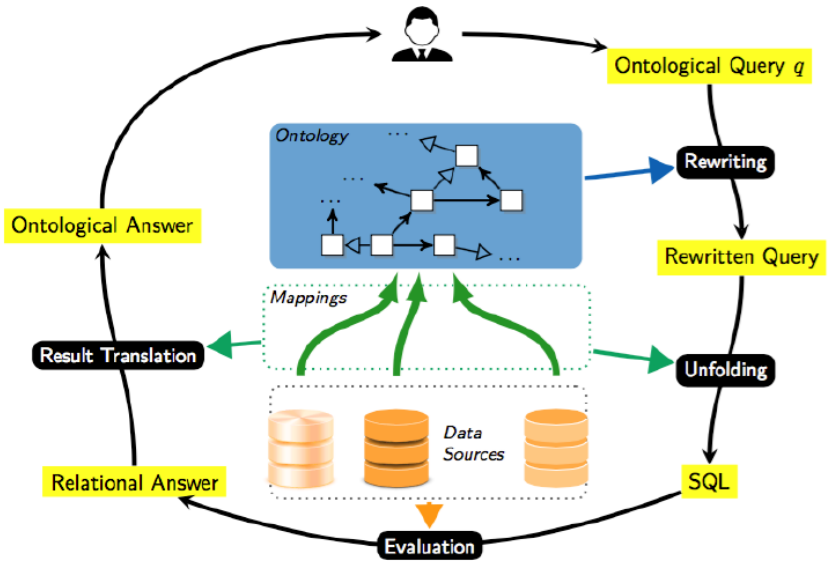
2 Ontology-based data access

Ontology Based Data Access (OBDA) is an approach to data access in which an ontology is used to mediate between data consumers and data sources.

OBDA Main Elements

- **Data consumers**: express information needs in their own terms without any prior knowledge about the way the data is organised at the source and receive answers in the same intelligible form.
- **Ontology**: Is 'a single point of semantic data access' for data consumers, i.e., a conceptual view. It provides global vocabulary over the data sources
- **Mappings:** declarative specifications that relate ontological terms with queries over the underlying data. Specify how to populate the ontology from the data

OBDA Process



OBDA systems automatically translate ontological queries, i.e., SPARQL, into database queries, i.e., SQL, and delegate execution of SQL queries to the database systems hosting the data, then translate the data retrieved from the databases into ontological terms.

OBDA is a virtual approach, providing an access layer on top of databases while leaving the data in its original stores

Bonus explanation for extra points:

Here's a short explanation of what's happening under the hood if you're interested. You can hear more about this in the Semantic Technologies course.

The ontology we use here is OWL QL, a very small subset of OWL 2 - it's much less expressive than the ontologies you made in the exercises. OWL QL has some very nice properties, mainly that it's first order rewritable and query answering is in AC_0 (sub-polynomial) in data complexity. It's still NP-complete in combined complexity (a reduction to graph colorability is simple to show), but in practice, the data is much larger than the TBox (see: statoil example). FO-rewritability means we can translate rewritten queries to SQL.

OWL QL can be rewritten using the PerfectRef algorithm which incorporates the OWL QL TBox into the query. RDF-S and OWL QL are really the only description logics where this is possible (OWL EL, another lightweight DL, is already too expressive). The rewritten query is a union of conjunctive queries (UCQ; basically a FO formula with positive existentials in DNF), and is exponential in the size

of the TBox in the worst case. But remember, the data is much larger than the TBox, so we don't really care about that.

The UCQ is then rewritten to SQL via the mappings. In essence, the SQL query is a union-statement. From the Statoil paper, as an example:

```
SELECT f(ID) AS x FROM ExpWBore
 UNION
SELECT f(W.ID) AS x FROM WellBore W, Purpose P
WHERE W.PurpID = P.ID AND P.Name = "Shallow".
```

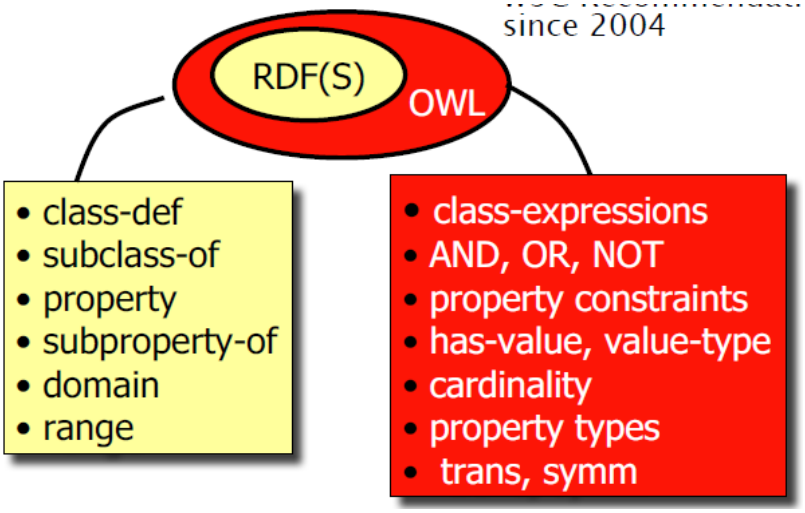
The SQL is then evaluated over the relational data sources. Here, we can make use of the hyper-optimized RDBMS. We can then return the answers via the mappings in "ontology form".

Note that we do not need to materialize the knowledge graph to answer queries. The FO-rewritability is what makes this possible. Imagine if you had gigabytes of data from different sources you had to upload to a GraphDB repository. This would be highly impractical. Also, SPARQL is easier to write than SQL if you have a defined vocabulary, so we don't need IT experts any more to write our queries. We also don't have as much of an integration effort (this is handled by the mappings). Most of the cost shifts to maintaining the ontology and the mappings (it's still pretty expensive).

Advantages/Disadvantages

+ End-user specific vocabulary for query specification
+ Layered on top of databases without requiring customisation
    + Virtual (as opposed to materialized) data access thus providing up-to-date answers
- Creating ontologies and mappings is time-consuming
    - Solution: partial bootstrapping from DB schema
- Query efficiency over large and distributed data sets
    - Solution: query optimization
- Usability beyond SPARQL
    - Graphical query interfaces

3 Advanced OWL modelling (examples)



OWL: used for specifying heavyweight ontologies. Full-fledged knowledge representation language for the web. Provides a wider range of ontological constructs and avoids some of the potential confusion in RDF-S.

Basic OWL features: classes, individuals, ObjectProperties, DatatypeProperties, class Hierarchies (owl:subClassOf), equivalence (owl:sameAs), difference (owl:differentFrom), several individuals different (owl:AllDifferent), closed Classes “nominals” (owl:oneOf), cardinalityRestrictions owl:([min|max])*Cardinality.

What follows here is a short description of OWL 2 (full) features.

Some terminology: Individuals are instances of classes in the ontology. Satisfiability means that there exists a model of the KB consisting of TBox (set of axioms) and the ABox (individual assertions). We can also only consider the satisfiability of the ABox/TBox individually.

- Declaration of individuals (owl:NamedIndividual)
- Class Disjointness: In OWL 1 and 2, there is a possibility of declaring two classes to be disjoint (owl:disjointWith). In OWL 2, we can also declare a set of classes to be mutually disjoint (owl:AllDisjointClasses)
- Union Disjoint Classes: In OWL 1, a class could be defined as the union of two classes, i.e. (Animal or Plant) is a LivingThing. In OWL 2, we can declare a class to be the disjoint union of a set of other classes (owl:disjointUnionOf). Note that this is only syntactic sugar, this can be expressed in multiple axioms of disjointness and union.
- Disjoint Properties: Two properties are disjoint if there are no two individuals that are interlinked by both properties (owl:propertyDisjointWith)
- Negative Property Assertions: Allows making an assertion that no relationship exists, for a particular property, between an individual and another individual or value (therefore applicable both to object and data properties; owl:propertyDisjointWith)
- Keys: Individuals can be identified uniquely with keys. Can be defined as object, data or a combination of properties (owl:hasKey)
- Data types: OWL 2 supports most XML schema data types.
- Comments and metadata: Can be provided by OWL annotations. An OWL annotation simply associates property-value pairs with parts of an ontology, or the entire ontology itself.
- Existential Restrictions: Basically an existential quantifier (owl: someValuesFrom). Means that there must be some relation to another object of the specified class.
- Universal Restriction: Basically a universal quantifier (owl:allValuesFrom). Means that all objects of a relation must be of a certain class. Is trivially satisfiable if there is no such relation for an instance. Common pitfall to only include a universal restriction without a value restriction.
- Qualified Cardinality restriction (== Combining Existential and Universal Restrictions): Specifies the exact number of objects of a class the class must be in a relation in.
- Value Restriction: A owl:hasValue restriction describes an anonymous class of individuals that are related to another specific individual along a specified property.
- Property chaining: Infer the existence of a property from a chain of properties (owl:propertyChainAxiom)

4 Ontology design patterns

An ontology design pattern (ODP) is a reusable successful solution to a recurrent modeling problem. Similar to OOP design patterns in programming (see: Gang of Four).

Specifically, we discussed:

- Part-of:

    - Transitive modeling of part-whole relation - transitivity returns all direct and indirect sub-parts
    - Component-wise modeling: Distinguish between parts and proper parts. Relation of proper parts is not transitive, and implies a simple has-part relation, which is transitive.
    - Extension: Time-indexed part-of (with n-ary relations)
- N-ary relations: One class extra to model the interaction of an n-ary relation.
- Roles:
    - Roles as classes: Roles are described at TBox level. ANTI-PATTERN
    - Roles as individuals: Roles are described at ABox level
    - Roles as properties: the semantics of “having a role” is embedded in a the name of a property
    - Roles as n-ary relations: expressive but larger taxonomy of classes and properties

We also discussed direct and indirect use of ODPs:

Direct:

- Advantages:
    - Stability and sustainability
    - Modularity
    - Interoperability
    - ODPs are unlikely to change
    - Easy to re-design in case of changes
- Disadvantages:
    - Dependency on external module
    - Mitigated risk of stability

Indirect:

- Advantages:
    - Dependency on external changes is limited to alignment axioms
    - Easier to re-design for fixing issues due to external changes
    - Stability and sustainability
    - Modularity and interoperability
- Disadvantages:
    - Effort for replicating the ODP implementation

5 Ontology Pitfall evaluation with patterns

Pitfalls are potential errors or problems when modelling ontologies. They could represent or lead to an error, but are not necessarily errors. For example, pitfalls might not represent an error depending on:

- Modelling decisions
- Context or scope of the ontology
- Ontology requirements

Not all pitfalls are equally important:

- minor: it does not represent a problem but correcting it makes the ontology better organized and user friendly
- important: Though not critical for ontology function, it is important to correct this type of pitfall
- critical: it is crucial to correct the pitfall. Otherwise, it could affect the ontology consistency, reasoning or applicability etc

Pitfall catalogue classification by dimensions:

- Structural Dimension
    - Ontology Language
    - Wrong Inference
    - Modelling Decisions
- Functional Dimension
    - Application context

    - Requirements Completeness
    - Real World Modelling or Common Sense
- Usability-profiling
    - Ontology Clarity
    - Ontology Understanding
    - Ontology Metadata

Examples of pitfalls:

- Creating synonyms as classes (“Cascade” and “Waterfall”)
- Defining wrong inverse relationships (C_1 p_1 C_2 and C_2 p_2 C_3 and p_1 inverseOf p_2)
- Missing inverse relationships (C_1 p_1 C_2 and C_2 p_2 C_1 but not p_1 inverseOf p_2)
- Swapping intersection and union (logical vs. linguistic “and”)
- Defining a relationship inverse to itself (p inverseOf p)
- Creating a property chain with just one property

OOPS! keeps a catalogue of ontologies reviewed by ontology experts. Submitted ontologies are scanned for pitfalls identified by experts in the ontology catalogue.

6 Knowledge graph/ontology quality metrics and dimensions

Some terminology: In description logics, a knowledge graph/knowledge base is the combination of an ABox and a TBox. A model of a KB is an interpretation that satisfies both the ABox and the TBox. Note that an interpretation can satisfy both the ABox and TBox individually, but not at the same time. Depending on the context, a KG can also be something that only consists of data, but includes information derived from the knowledge component (in our case, the ontology/TBox).

Data quality: commonly conceived as a multi-dimensional construct with a popular definition “fitness for use”.

Goals of Data Quality Assessment:

- Fix quality issues in the KG
- Root cause analysis: identify source of quality issues
    - quality issues in the source data
    - arising due to data integration
    - incorrect use of vocabularies
    - linking to data from untrustworthy sources
- Assess and ultimately improve KG quality

Data quality categories with their dimensions for KGs:

- Accessibility: Criteria determine how the data can be accessed
    - Availability
    - Interlinking
    - Licensing
    - Performance
    - Security
- Intrinsic: Criteria that are independent of the user’s context. They focus on whether information correctly (semantically and syntactically), compactly and completely represents the real world and whether information is logically consistent in itself.
    - Syntactic validity
    - Completeness

    - Consistency
    - Semantic Accuracy
    - Conciseness
- Contextual: Dimensions that are highly dependent on the context of the task at hand and must be assessed by taking into account such context.
    - Trustworthiness
    - Relevancy
    - Understandability
    - Timeliness
- Representation: Capture aspects related to the design of the data and in which form data is available.
    - Interoperability
    - Versatility
    - Representational Conciseness
    - Interpretability

The four categories and their dimensions are related to each other, so we can not consider them separately.

Tools for evaluating data quality:

- SHACL
- Crowdsourcing; uComp Protégé plugin

7 Data Validation with SHACL

One of the weaknesses of semantic web technologies is that the schema-less nature makes optimization difficult.

Why should we describe and validate RDF?

- For producers
    - Developers can understand the contents they are going to produce
    - They can ensure they produce the expected structure
    - Advertise and document the structure
    - Generate interfaces
- For consumers
    - Understand the contents
    - Verify the structure before processing it
    - Query generation and optimization

Why is RDF-S not enough?

- RDF “schema” is a misnomer, should be “RDF Vocabulary Definition Language”
- Very limited expressivity
- Not the right semantics for **data** validation
- Invalid data leads to more inferred invalid data

Why is OWL not enough?

- De facto a constraint language. Logical contradiction leads to invalid data
- Very expressive, but targeted at logical modeling, not at validity constraints
- Not the right semantics for **data** validation
- OWA
- No Unique Name Assumption

SHACL (SHApe Constraint Language) constraints are written in RDF. Uses SPARQL underneath. Validating RDF graphs against a set of shapes. The shapes in a RDF graph are called “Shapes Graph”, the graph to be validated the “Data Graph”.

SHACL Terminology:

- Target
  - Determine what resources are to be validated against a Shape
  - During the validation, targets are referred to as “focus nodes”
- Shape
  - A Shape determines how to validate a focus node based on the values of properties and other characteristics of the focus node.
  - Shape can be a NodeShape or a PropertyShape
  - NodeShape specify conditions that need to be met by property values; NodeShape can be used to group PropertyShapes
- Constraint Components
  - A set of predefined constraint components
  - “SHACL core vocabulary”
  - Users can generate new constraint components - domain-specific data validation languages
- Validation Report
  - An RDF graph with validation results
  - SHACL includes a vocabulary for describing results
- Shapes Graph and Data Graph
  - These are “roles”
  - Any graph can be declared to be a Shapes Graph or Data Graph

8 General Ontology Alignment Process

Ontology Alignment: bringing two ontologies into mutual agreement, making them consistent and coherent with one and another. It may often include a transformation of the source ontologies removing the unnecessary information and integrating missing information.

- Step 1: Feature Engineering
  Determine a list of features F: Extract characteristics of both ontologies, i.e. the features of their ontological entities (concepts C, relations E, and instances I) from intensional and extensional ontology definitions. Possible features:
  - Identifiers, i.e. strings with dedicated formats, such as URIs or labels
  - RDF/S Primitives: such as properties or subclass relations
  - OWL Primitives: such as an entity being the sameAs another entity
  - Derived Features: which constrain or extend simple RDFS primitives
- Step 2: Search Step Selection
  Most common methods:
  - compare all entities of O_1 with all entities of O_2: any pair is treated as a candidate mapping
  - or only compare entities of the same type (class, relation, instance)
  - or use heuristics to lower the number of candidate mappings using strategies such as random or label, or change propagation

- Step 3: Similarity Computation
  Based on the features F of the ontological entities we do the similarity computation for all pairs of candidates. Additional similarity measures exist.
- Step 4: Similarity Aggregation
  The individual similarity measures are weighted and combined. Some approaches for aggregation:
  - Averaging, weight = 1
  - Linear Summation
  - Linear Summation with negative evidence: weight can have a negative value
  - Sigmoid Function: emphasize high similarity and de-emphasize low similarity
- Step 5: Interpretation
  Aggregated similarity is compared to a threshold: every value above indicates an alignment.
  Determine the threshold:
  - constant,
  - $\max(sim_{agg}(e, f))$ - constant
  - $\max(sim_{agg}(e, f))$ (1 - p)
- Step 6: Iteration
  - Entities are similar if their position in the structure is similar
  - Structure similarity is expressed through the similarity of the other entities in the structure
  - Calculating the similarity for one entity pair, the similarity of the neighboring entity pairs are needed
  - In a first round only basic comparison methods (e.g. string similarity) are applied (or pre-given alignments are used)
  - In further rounds already computed pairs and use more sophisticated structural similarity measures
  - When to stop the iteration:
    - 1. fixed number of iterations
    - 2. fixed time constraint
    - 3. changes of alignments compared to a threshold

Evaluation

- Precision $p = \frac{\#correct-found-mappings}{\#found-mappings}$
- Recall $r = \frac{\#correct-found-mappings}{\#existing-mappings}$
- F-Measure $f_1 = \frac{2pr}{p+r}$
- Compliance measure: quality of identified alignments
- Performance measure: quality of algorithm in terms of computational resources
- User-related measure: overall subjective user satisfaction
- Task-related measure: quality of alignment for a certain use case or application scenario

9 Ontology Alignment Application Scenarios

Ontology Merging

- Two or more ontologies are combined into one target ontology
- By establishing alignments among entities we identify equal entities which we can merge

- Time resources are less critical
- Human post-processing is required
- Finally high quality requirements

Web Service Composition

- Agents or web services often use different representations of their domains resulting in different expressions on their goals, and their input or output
- Collaborate despite the heterogeneous representations
- Standard upper-level ontologies or ontology alignment
- Alignment needs to be fast, reliable, and correct
- Wrong results can lead to unjustified costs
- Sometimes user checks are possible

Query and Answer Mapping

- Users formulate a query in a specific query language based on a specific ontology
- Query is sent to a query engine
- To access heterogeneous information sources the query needs to be re-written for the target ontologie(s)
- For the presentation of the answers the results have to be transformed back again
- Rewriting / Mapping should be fast and fully automatic
- Users may tolerate wrong results as long as the correct results are returned as well

Reasoning

- New information is inferred from distributed and heterogeneous ontologies
- Time constraints are not critical (for both, alignment and inference tools)
- Quality of the alignments is very important
- Alignment needs to be done automatically
- Wrong results may trigger additional wrong results in a cascading manner
- Detection of conflicting inconsistencies is required

10 Scope and Purpose of the Ontology Alignment Evaluation Initiative

The goals of the Ontology Alignment Evaluation Initiative are:

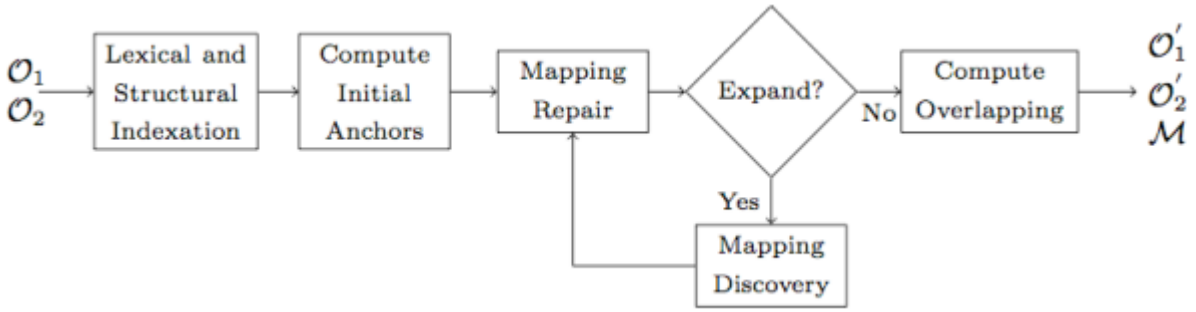
- assessing strengths and weaknesses of alignment/matching systems;
- comparing performance of techniques;
- increase communication among algorithm developers;
- improve evaluation techniques;
- most of all, helping improve the work on ontology alignment/matching.

OAEI is a yearly contest. Alignment tools are run on certain datasets on given domains. There exists a gold standard for the mappings and the tools' results are compared to the gold standards and the performances of the different tools are compared to each other.

11 How to use alignment tools like LogMap or AML

Both from the command line and LogMap via the web interface?

LogMap in a nutshell



LogMap is a highly scalable ontology matching system with ‘built-in’ reasoning and diagnosis capabilities

- lexical techniques for establishing initial mappings;
- structural and semantic methods for automatically refining or repairing them.

to efficiently produce alignments for large ontologies (focus on scalability and logical consistency).

1. Lexical Indices

For each class, LogMap takes the English name of each class, as well as any other alternative label (usually stored in the ontology as OWL annotations), and splits them into single words -> each label produces a set of words which is then introduced as the key in the inverted index to store the class identifier (an integer). Enhancing the indices: alternative labels from annotations in source ontologies, synonyms from WordNet, Stemming

2. Structural Indexation

Extended hierarchy of each ontology is computed using structural heuristics or an off-the-shelf DL reasoner (eg. HermiT and Condor) and consists of the following elements:

- The inferred hierarchy of the ontology, describing all the parent-child links between its classes.
- Disjointments explicitly declared in the ontology.
- Additional complex class axioms

3. Computation of Initial Anchors

The initial set of anchors is computed by intersecting the sets of keys in each inverted index : classes whose labels contain exactly the same words (disregarding the order). -> Reliable mappings, used as initial anchors for the iterative core of the algorithm.

4.a. Repair

A (possibly incomplete) reasoning algorithm (eg. Dowling and Gallier) is used to identify unsatisfiable classes with respect to the merge of the input ontologies and the set of mappings established so far. Structural indices of the input ontologies are used together with the established mappings to produce a propositional Horn theory. Remove some of the found mappings in order to eliminate such unsatisfiabilities yielding a set of ‘clean’ mappings

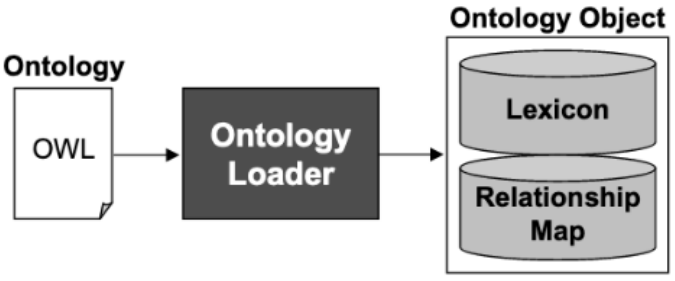
4.b. Discovery

New mappings are computed by matching the classes in the relevant contexts using ISUB (string similarity) -> match the neighbours of C1 in the ontology hierarchy of O1 to the neighbours of C2 in the hierarchy of O2. (If classes C1 and C2 are correctly mapped, then the classes semantically related to C1 in O1 are likely to be mapped to those semantically related to C2 in O2.)

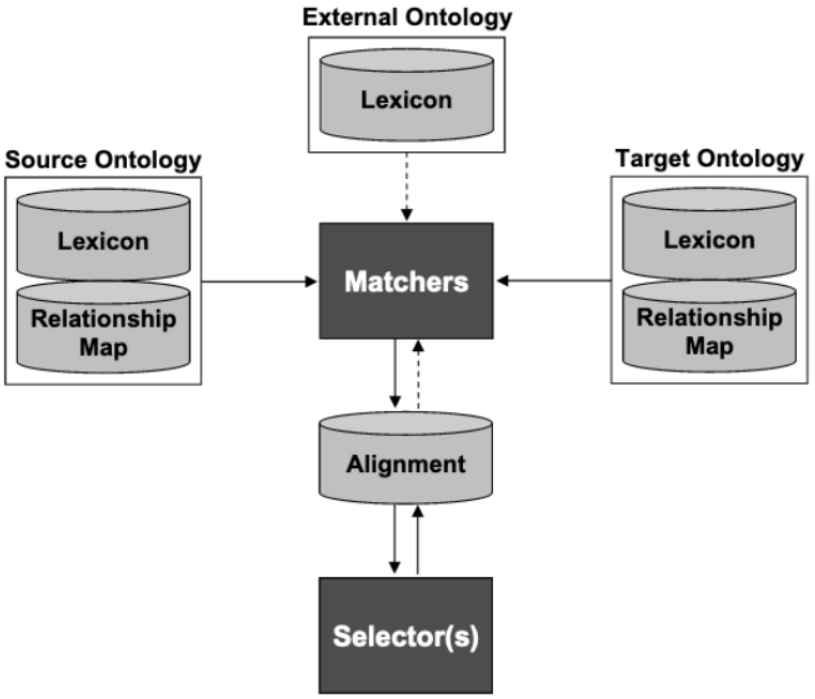
LogMap continues the iteration of repair and discovery steps until no context is expanded in the discovery step. The output of this process is a set of mappings that are likely to be ‘clean’ that is, it will not lead to logical errors when merged with the input ontologies.

AML (AgreementMakerLight)

- Agreement-MakerLight based on AgreementMaker framework and focused on computational efficiency and designed to handle very large ontologies
- AML has excellent run time results and (in many cases) the best F-measure
- Unlike AgreementMaker it is not designed to support user interaction, but it maintains the flexibility and extensibility of the original framework.
- Matching or aligning two input ontologies (one source ontology and one target ontology) consists in finding semantic relationships between the classes of the source ontology and the classes of the target ontology
- Only equivalence relationships, here called mapping
- Set of mappings between two ontologies is called an alignment
- Assign a numerical value to each mapping, reflects the semantic similarity between the entities (classes)
- extensibility: allows for inclusion of virtually any matching algorithm



← AML Loading Module



← Ontology Matching Module

12 Semantics in the Web: Achievements and Challenges

Goal: Understand the semantics of the information on the Web to enable intelligent systems to do a better job of processing Web documents

Achievements

- Some challenges were ignored in early Semantic Web research approaches: messiness of the real Web and lack of any organisation or semantics
- But it has brought important advancements to the Web of today:
    - in a sample of over 12 billion web pages, 2.5 billion pages (21%) use the schema.org format (2014).
    - Linked data powers media sites for organisations such as the BBC and New York Times; major libraries and museums around the world as well as governments and public organisations publish their data as linked data.
    - Large Web companies like Google, Yahoo!, Microsoft, Facebook, as well as numerous research projects are developing large knowledge graphs, which define, structure, and link hundreds of millions of entities, to enhance search, to provide better advertising match, to improve the answers of their AI-based personal assistants, etc
- The Web of today:
    - Commercial DBMSs (e.g. Oracle) provide native support for Semantic Web languages
    - Recommender companies use semantics and semantic tagging to improve quality and accuracy of recommendations (people who liked this, also liked…)
    - The WHO developed the International Classification of Diseases (ICD)

Challenges

- Modern semantic approaches leverage vastly distributed, heterogeneous data with needs-based, lightweight data integration.
- Take advantage of the coexistence of a myriad of different, sometimes contradictory, ontologies of varying levels of detail without assuming all-encompassing or formally correct ontologies.
- Textual data that is available on the Web, in hundreds of languages, to train artificially intelligent agents that will understand what users are trying to say in a given context and what information is most pertinent to users' goals at a given time
- In addition to the increasing amount of semantically annotated information on the Web, a lot more structured data is becoming available, from scientists and governments publishing data.
- Harnessing and exploiting this data, and understanding its diverse and often contradicting nature, to provide really meaningful services and to improve the quality of our lives
- Some of the semantic knowledge that researchers had to construct manually they can now learn automatically, tremendously increasing the scale of the use of semantics in understanding and processing Web data. Ontologies themselves might be learned or enhanced automatically.
- While manually constructed formal ontologies may often (but not always) be required to form a backbone of semantics for the Web, much of the content that puts "meat" on those bones is "scruffy" and imprecise, often statistically induced.
- "shallow" lightweight semantics (such as RDFa) may be more widely applicable

13 Lightweight Ontologies

RDFa is based on attributes (reusing some HTML attributes like href, src and adding new ones) bringing a couple of machine-readable hints to web pages.

Good Relations

GoodRelations is a lightweight ontology for exchanging e-commerce information - data about products, offers, points of sale, prices, terms and conditions.

It defines a data structure for e-commerce that is

- industry-neutral, i.e. suited for consumer electronics, cars, tickets, real estate, labor, services, or any other type of goods,
- valid across the different stages of the value chain, i.e. from raw materials through retail to after-sales services, and
- syntax-neutral, i.e. it should work in microdata, RDFa, RDF/XML, Turtle, JSON, OData, GData, or any other popular syntax.

This is achieved by using just four entities for representing e-commerce scenarios:

1. An agent (e.g. a person or an organization) → gr:BusinessEntity
2. An object (e.g. a camcorder, a house, a car,...) or service (e.g. a haircut), → gr:ProductOrService
3. A promise (offer) to transfer some rights (ownership, temporary usage, a certain license,..) on the object or to provide the service for a certain compensation (e.g. an amount of money), made by the agent and related to the object or service, → gr:Offering
4. A location from which this offer is available (e.g. a store, a bus stop, a gas station,…). → gr:Location

Used by a lot of small and large shops world-wide, like Google, Best Buy, OpenLink Software etc

Open Graph Protocol

OGP enables a web page to become a rich object in a social graph, RDFa-based simple vocabulary used by Facebook

basic metadata:

- og:title
- og:type
- og:image
- og:url

optional metadata:

- og:audio
- og:description
- og:determiner
- og:locale
- og:locale:alternate
- og:site_name
- og:video

14 How are you using semantic data?

Simplest answer: Google KG, Amazon Product KG. Also: there are a bunch of ontologies used by large companies to extract information from their product's metadata or to connect different parts of their businesses.

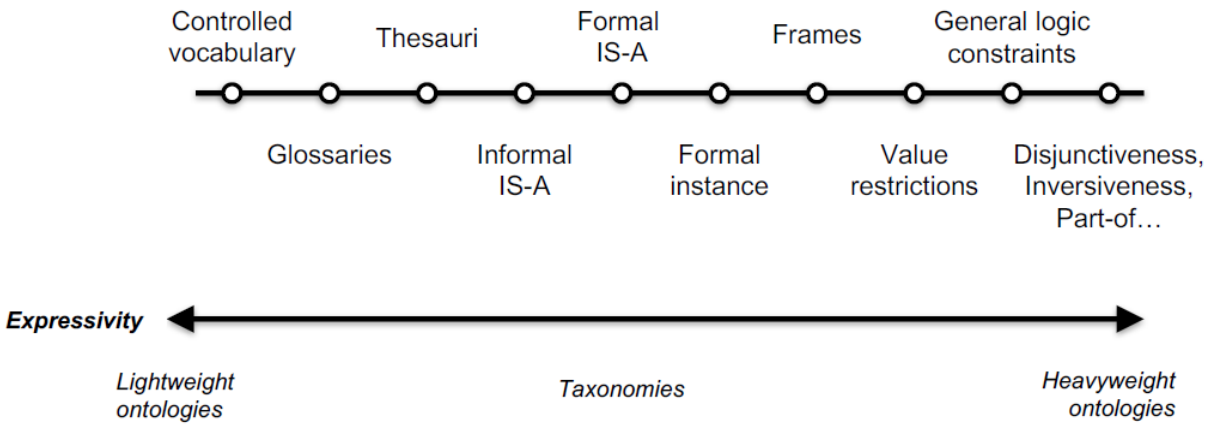
Another example is the BBC's ontology (https://www.bbc.co.uk/ontologies) that organizes all of its content. We might not be directly aware that semantic data is being used, but we encounter it basically every time we browse the world wide web.

15 Rich semantic interrelationships versus semantic snippets?

Semantic snippets are easier to integrate into e.g. websites and provide a simple machine-readable access to metadata. If we want to use rich semantic interrelationships or semantic snippets depends on the use-case, and our budget (time, personnel, money…).

semantic snippets as is RDFa

rich semantic interrelationships as in heavy-weight ontologies



16 Do we need large ontologies?

Again, the answer is: it depends. Large ontologies facilitate more powerful reasoning, but at a computational cost. Lightweight ontologies on the other hand can deliver similar results at lower cost. Interestingly, lightweight description languages have only emerged in the last 10-15 years, whereas heavyweight DLs have been around for much longer.

17 Semantic search?

Semantic search denotes search with meaning, as distinguished from lexical search where the search engine looks for literal matches of the query words or variants of them, without understanding the overall meaning of the query. Semantic search seeks to improve search accuracy by understanding the searcher's intent and the contextual meaning of terms as they appear in the searchable dataspace, whether on the Web or within a closed system, to generate more relevant results. Content that ranks well in semantic search is well-written in a natural voice, focuses on the user's intent, and considers related topics that the user may look for in the future.

Some authors regard semantic search as a set of techniques for retrieving knowledge from richly structured data sources like ontologies and XML as found on the Semantic Web. Such technologies enable the formal articulation of domain knowledge at a high level of expressiveness and could enable the user to specify their intent in more detail at query time.

18 Interesting tools you came across?

If you completed the exercise, you should know these tools:

- Protégé: Free to use ontology engineering UI tool. Supports a large number of plugins for SHACL, ONTOP, reasoners (HermiT, Pellet,...) to make life easier. Not really industry-grade, but very good for an open source tool. Has some weird error messages sometimes or just crashes.
- Ontop: OBDA tool. Works best with h2. Throws cryptic errors or simply doesn’t work if you don’t use one of the recommended RDBMS or don’t use the default settings of those. Mappings are also kinda wonky (recursion doesn’t really work).
- GraphDB: Graph database/Semantic Repository/Triple Store with SPARQL functionality. Really only used to upload the KG
- OOPS!: Online tool to scan an ontology for common pitfalls. Easy to use, good interface, and good explanation of the pitfalls (with references!). Highly recommended to use during ontology engineering.
- SHACL: Shape Constraint Language to validate the KG. There is a plugin for Protégé which is fairly simple to use. Overcomes the limitations of RDF-S and OWL for data validation
- AML: Ontology alignment tool. Works well out of the box, results are easy to interpret. Fast results, good placements in competitions regarding F1-score.
- LogMap: Ontology alignment tool. Uses a web interface and email to send results. Fast results, pretty good in competitions for EL ontologies.