Engineering Virtual Knowledge Graphs

Diego Calvanese

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

Department of Computing Science
Umeå University, Sweden

Ontopic s.r.l.

Huawei Strategy and Technology Workshop (STW)

26–28 September 2022 – Shenzhen, China & Online

The problem of data access

In large organization data management is a complex challenge:

- Many different data sets are created independently.
- The data is heterogeneous in the way it is represented and structured.
- Data are often stored across different sources (possibly controlled by different people / organizations).

The problem of data access

However, complex data processing pipelines (e.g., for analysis, monitoring and prediction) require to **access in an integrated and uniform way** such large, richly structured, and heterogeneus data sets.

How can we address the complexity of data access?

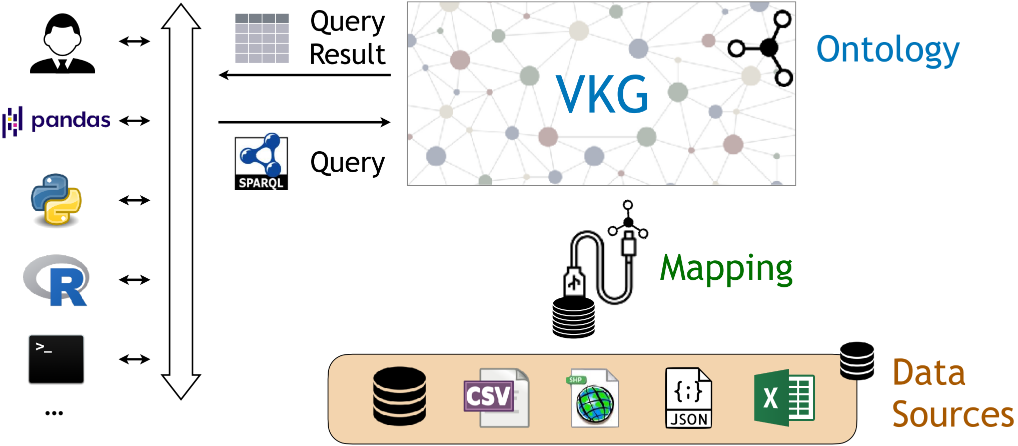
We combine three key ideas:

1. Expose to users/applications the data in very flexible data model, making use of terms the users are familiar with
   $\leadsto$ **Knowledge Graph** whose vocabulary is expressed in a **domain ontology / global schema**.
2. **Map the data sources to the global schema** in order to provide the data for the KG.
3. Exploit **virtualization**, i.e., the KG is not materialized, but kept virtual.

This gives rise to the **Virtual Knowledge Graph** (**VKG**) approach to data access,
also called **Ontology-based Data Access** (**OBDA**).
[Xiao et al. 2018, IJCAI]

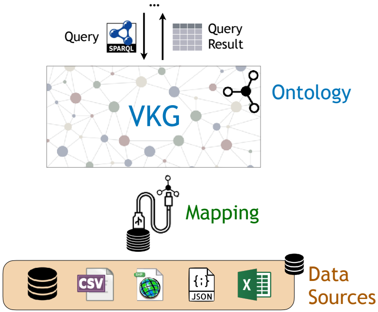
unibz

Virtual Knowledge Graph (VKG) architecture



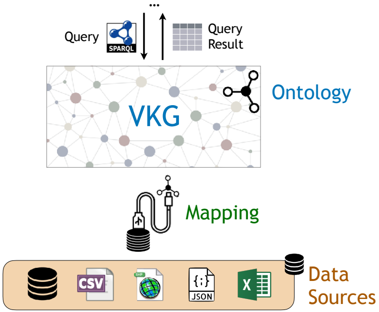
Why an ontology?

An ontology is a structured formal representation of concepts and their relationships that are relevant for the domain of interest.



- In the VKG setting, the ontology has a twofold purpose:
  - It defines a vocabulary of terms to denote classes and properties that are familiar to the user.
  - It extends the data in the sources with background knowledge about the domain of interest, and this knowledge is machine processable.
- One can make use of custom-built domain ontologies.
- In addition, one can rely on standard ontologies, which are available for many domains.

Why a Knowledge Graph for the global schema?

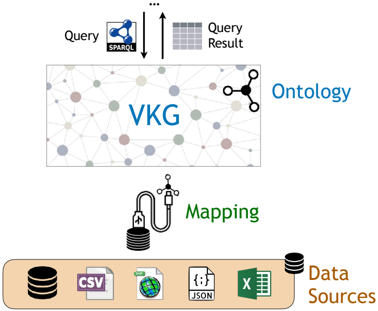


Traditional approaches to data management rely on the relational model.

A Knowledge Graph, instead:

- Does not require to commit early on to a specific structure.
- Can better accommodate heterogeneity.
- Can better deal with missing / incomplete information.
- Does not require complex restructuring operations to accommodate changes or new information.

Why mappings?

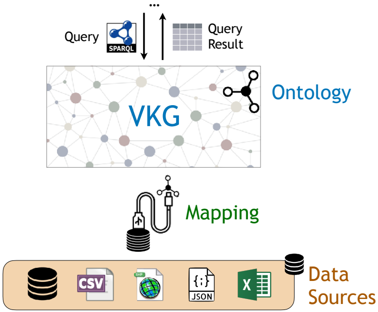


Traditional approaches to data access/integration rely on mediators, which are specified through complex code.

Mappings, instead:

- Provide a declarative specification, and not code.
- Are easier to understand, and hence to design and to maintain.
- Support an incremental approach to integration.
- Are machine processable, hence are used in query answering and for query optimization.

Why virtualization?



Materialized data access / integration relies on extract-transform-load (ETL) operations, to load data into an integrated data store / data warehouse / materialized KG.

In the virtual approach, instead:

- The data stays in the sources and is only accessed at query time.
- No need to construct a large and potentially costly materialized data store and keep it up-to-date.
- Hence the data is always fresh wrt the latest updates at the sources.
- One can rely on existing data infrastructure and expertise.

Outline

1. Virtual Knowledge Graphs (VKGs) for Data Access
2. Components of the VKG Architecture
3. Query Answering in VKGs
4. The Ontop System
5. Designing a VKG System
6. Conclusions

Outline

1 Virtual Knowledge Graphs (VKGs) for Data Access

2 Components of the VKG Architecture

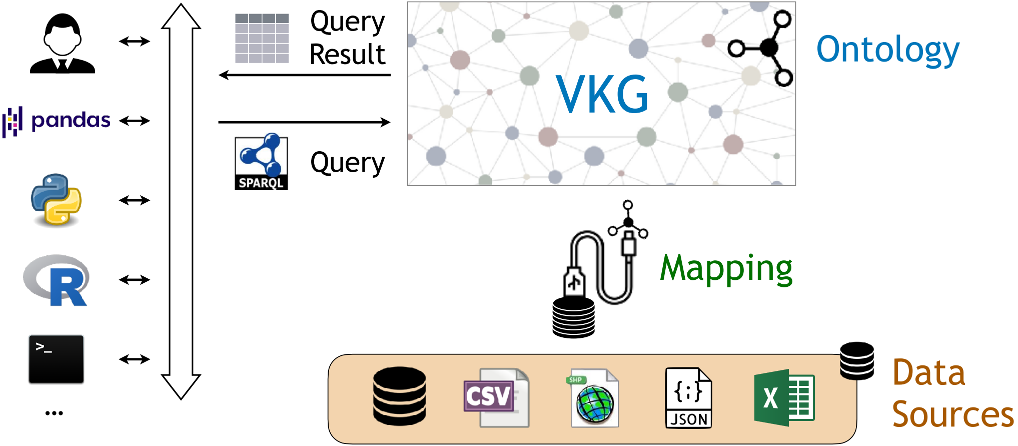
3 Query Answering in VKGs

4 The Ontop System

5 Designing a VKG System

6 Conclusions

Virtual Knowledge Graph architecture



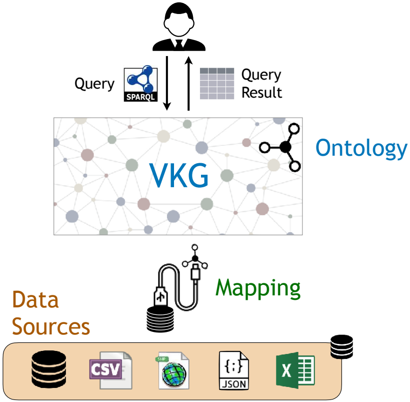
Engineering a VKG solution – Which languages?

Which are the right languages for the components of the VKG framework?

We need to consider the **tradeoff between expressive power and efficiency**, where efficiency with respect to the data is the key aspect to consider.

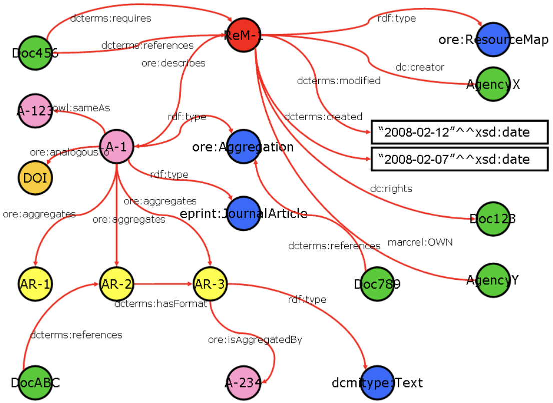
The W3C has standardized languages suitable for VKGs:

1. Knowledge graph: expressed in **RDF** [W3C Rec. 2014]
2. Ontology $\mathcal{O}$: expressed in **OWL 2 QL** [W3C Rec. 2012]
3. Mapping $\mathcal{M}$: expressed in **R2RML** [W3C Rec. 2012]
4. Query: expressed in **SPARQL** [W3C Rec. 2013]



RDF – Data is represented as a graph

The graph consists of a set of **subject-predicate-object triples** relating objects to other objects or values and to classes.



Object property:

`<A-1> ore:describes <ReM-1> .`

Data property:

`<ReM-1> :created "2008-02-07" .`

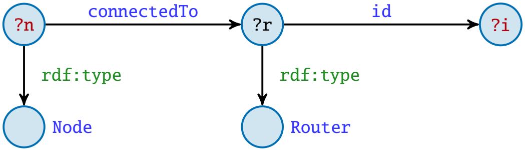
Class membership:

`<ReM-1> rdf:type ore:ResourceMap .`

SPARQL query language

- Is the standard query language for RDF data. [W3C Rec. 2008, 2013]
- Core query mechanism is based on **graph matching**.

```
SELECT ?n ?i
WHERE { ?n rdf:type Node .
        ?n connectedTo ?r .
        ?r rdf:type Router .
        ?r id ?i .
      }
```



Additional language features (SPARQL 1.1):

- `UNION`: matches one of alternative graph patterns
- `OPTIONAL`: produces a match even when part of the pattern is missing
- complex `FILTER` conditions
- `GROUP BY`, to express aggregations
- `MINUS`, to remove possible solutions
- property paths (regular expressions)

The OWL 2 QL ontology language

- **OWL 2 QL** is one of the three standard sub-languages of the very expressive standard ontology language OWL 2. [W3C Rec. 2012]
- It is considered a lightweight ontology language:
  - controlled expressive power
  - efficient inference
- Optimized for accessing large amounts of data
  - Queries over the ontology can be rewritten into SQL queries over the underlying relational database (First-order rewritability).
  - Logical consistency of ontology and data can also be checked by executing SQL queries over the underlying database.

Constructs of OWL 2 QL

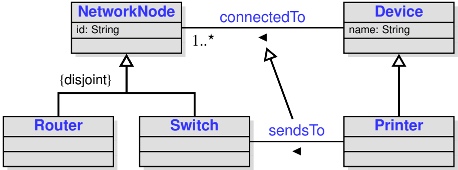
In an OWL 2 QL ontology, one can express knowledge about the classes and properties in the domain of interest by means of various types of assertions.

• Subclass assertions	Router `rdfs:subClassOf` NetworkNode
• Class disjointness	NetworkNode `owl:disjointWith` User
• Domain of a property	connectedTo `rdfs:domain` User
• Range of a property	connectedTo `rdfs:range` NetworkNode
• Subproperty assertions	sendsTo `rdfs:subPropertyOf` connectedTo
• Inverse properties	accesses `owl:inverseOf` isAccessedBy
• Mandatory participation to a property	... `owl:someValuesFrom` ...

OWL 2 QL ontologies and conceptual data models

There is a close correspondence between OWL 2 QL and conceptual modeling formalisms, such as UML Class Diagrams and ER Schemas.

Router `rdfs:subClassOf` NetworkNode	subclass
Router `owl:disjointWith` Switch	disjointness
connectedTo `rdfs:domain` Device	domain
connectedTo `rdfs:range` NetworkNode	range
sendsTo `rdfs:subPropertyOf` connectedTo	sub-association
... `owl:someValuesFrom` ...	mandatory participation



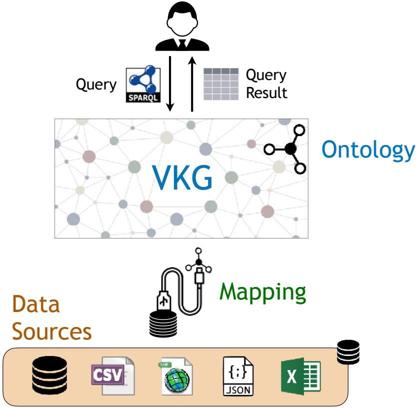
In fact, to visualize an OWL 2 QL ontology, we can use standard UML Class Diagrams / ER Schemas.

The role of mappings

In the VKG framework, the mapping encodes how the data in the sources should be used to create the Virtual Knowledge Graph, which is formulated in the vocabulary of the ontology.

VKG defined from the mapping and the data.

- Queries are answered with respect to the ontology and the data of the VKG.
- The data of the VKG is not materialized (it is virtual!).
- Instead, the information in the ontology and the mapping is used to translate queries over the ontology into queries formulated over the sources.



Note: The graph is **always up to date** wrt the data sources.

Mapping language

The **mapping** consists of a set of assertions of the form

SQL Query ⇝ Class membership assertion

SQL Query ⇝ Property membership assertion

Intuition behind the mapping

The answers returned by the SQL Query in the left-hand side are used to create the objects (and values) that populate the Class / Property in the right-hand side.

Note: The mapping contains also a mechanism to transform values retrieved from the database into objects of the VKG (thus solving the so-called impedance mismatch).

unibz

Outline

1. Virtual Knowledge Graphs (VKGs) for Data Access
2. Components of the VKG Architecture
3. Query Answering in VKGs
4. The Ontop System
5. Designing a VKG System
6. Conclusions

Query answering in VKGs

In VKGs, we want to answer queries formulated over the ontology, by using the data provided by the data sources through the mapping.

- The ontology contains **domain knowledge** that can be used to enrich answers.

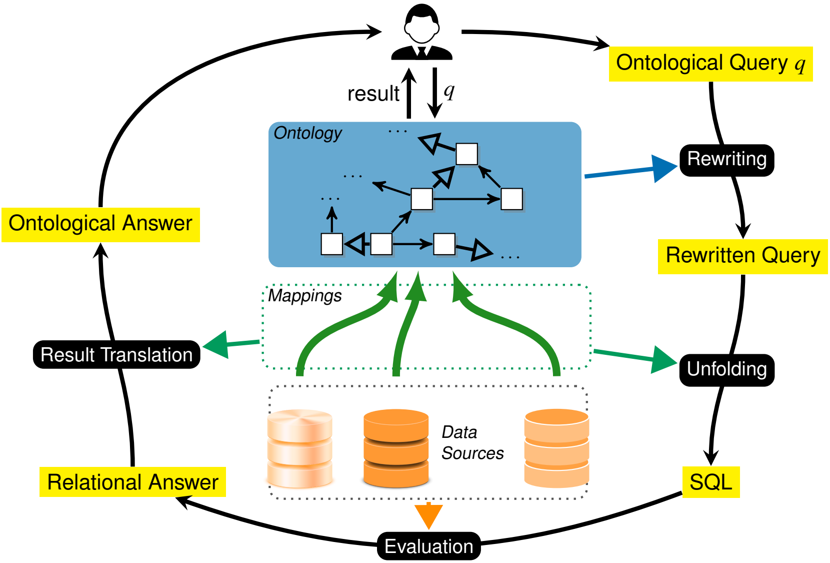
  Example: Suppose that our data contains LJ-2025 among the Printers, and that the ontology states that each Printer is a NetworkDevice.
  If we ask for all NetworkDevices, we should return also LJ 2025, considering both the data and the knowledge in the ontology.

- The **mapping** encodes the information of how to translate a query over the ontology into a query over the **database**.

A VKG query answering engine has to take into account all these types of information.

Query answering by query rewriting

Query answering by query rewriting



Outline

1. Virtual Knowledge Graphs (VKGs) for Data Access
2. Components of the VKG Architecture
3. Query Answering in VKGs
4. The Ontop System
5. Designing a VKG System
6. Conclusions

The *Ontop* system

ontop

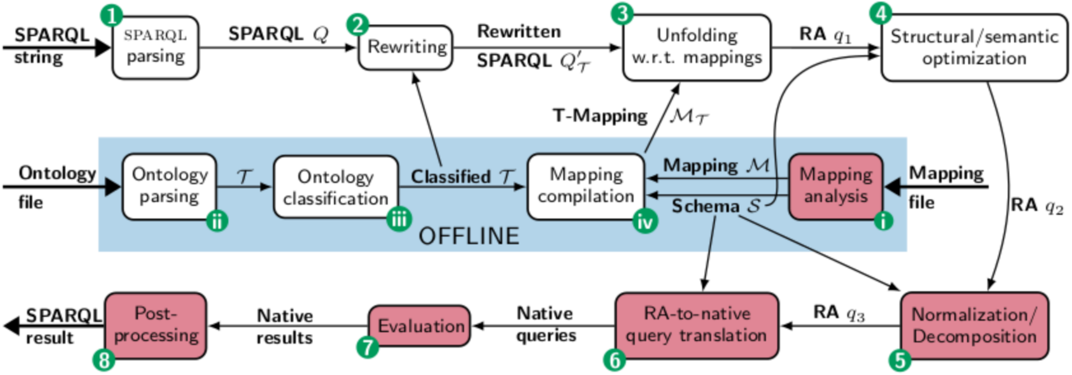
https://ontop-vkg.org/

- State-of-the-art VKG system.
- Addresses the key challenges in query answering of scalability and performance.
- Compliant with all relevant Semantic Web standards:

  RDF, RDFS, OWL 2 QL, R2RML, SPARQL, and GeoSPARQL.
- Supports all major relational DBMSs:

  Oracle, DB2, MS SQL Server, Postgres, MySQL, Teiid, Dremio, Denodo, etc.
- Open-source and released under Apache 2 license.

Query answering in *Ontop*



unibz

Developer community

unibz

ON TOP IC

Birkbeck
UNIVERSITY OF LONDON

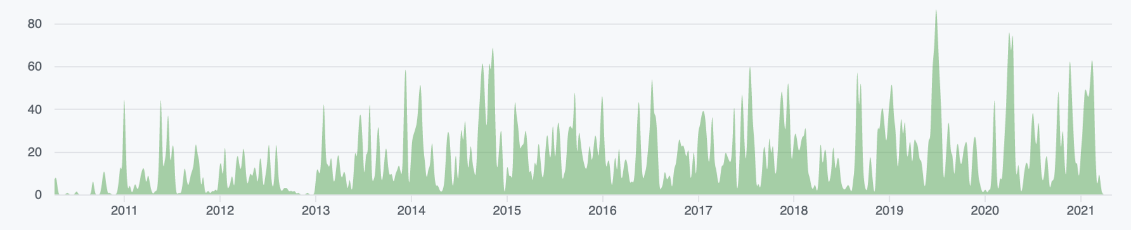
UiO : University of Oslo

HELLENIC REPUBLIC
National and Kapodistrian
University of Athens

UNIVERSITÄT
LEIPZIG

ontotext

POLITECNICO
MILANO 1863



Outline

1. Virtual Knowledge Graphs (VKGs) for Data Access
2. Components of the VKG Architecture
3. Query Answering in VKGs
4. The Ontop System
5. Designing a VKG System
6. Conclusions

Designing the ontology

- Ontologies might become very large and complex, hence designing an ontology is a challenging task.

- In many domains (e.g., the biomedical one) ontologies are developed independently by trained experts and already available to be re-used.

- Relying on “standardized ontologies” also enables interoperability across different domains.

- However, ontology design is a well investigated task, and methodologies and supporting tools are readily available. See, e.g.,
  - the series of *Workshops on Ontology Design Patterns* [`http://ontologydesignpatterns.org/`];
  - the OntoClean methodology for ontology analysis based on formal, domain-independent properties of classes [Guarino & Welty 2009].

- Also, due to the correspondence with UML Class Diagrams / ER Schemas, we can rely on standard methodologies and tools adopted in software engineering or database design.

Designing VKG mappings

VKG mappings:

- Map complex queries to the target ontology.
- Overcome the abstraction mismatch between relational data and a knowledge graph.
- Are inherently more sophisticated than mappings used in other settings, such as schema matching [Rahm & Bernstein 2001] or ontology matching [Euzenat & Shvaiko 2007].

As a consequence:

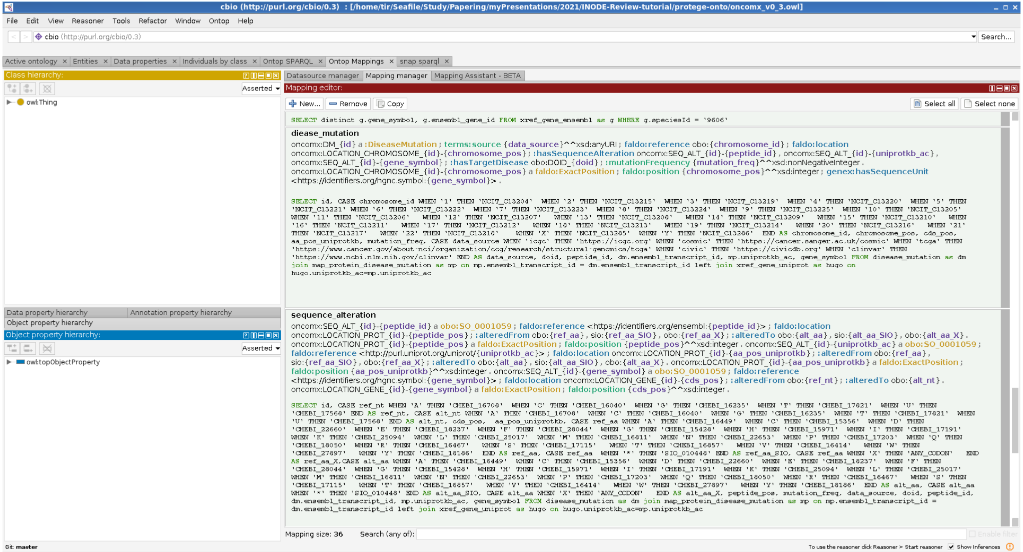
- The design of VKG mappings is an essentially manual effort that is **labor-intensive** and **error-prone**.
- Requires highly-skilled professionals [Spanos, Stavrou & Mitrou 2012].
- Writing mappings is challenging in terms of semantics, correctness, and performance.

Designing and managing mappings is the most critical bottleneck
for the adoption of the VKG approach.

Designing VKG mappings

Writing mappings manually is a **time-consuming** and **error-prone** task.

Designing VKG mappings



```
SELECT distinct g.gene_symbol, g.ensembl_gene_id FROM xref_gene_ensembl as g WHERE g.speciesId = '9606'
```

diease_mutation

```
oncomx:DM_{id} a :DiseaseMutation ; terms:source {data_source}^^xsd:anyURI ; faldo:reference obo:{chromosome_id} ; faldo:location
oncomx:LOCATION_CHROMOSOME_{id}-{chromosome_pos} ; :hasSequenceAlteration oncomx:SEQ_ALT_{id}-{peptide_id} , oncomx:SEQ_ALT_{id}-{uniprotkb_ac} ,
oncomx:SEQ_ALT_{id}-{gene_symbol} ; :hasTargetDisease obo:DOID_{doid} ; :mutationFrequency {mutation_freq}^^xsd:nonNegativeInteger .
oncomx:LOCATION_CHROMOSOME_{id}-{chromosome_pos} a faldo:ExactPosition ; faldo:position {chromosome_pos}^^xsd:integer ; genex:hasSequenceUnit
<https://identifiers.org/hgnc.symbol:{gene_symbol}> .
```

```
SELECT id, CASE chromosome_id WHEN '1' THEN 'NCIT_C13204'  WHEN '2' THEN 'NCIT_C13215'  WHEN '3' THEN 'NCIT_C13219'  WHEN '4' THEN 'NCIT_C13220'  WHEN '5' THEN
'NCIT_C13221' WHEN '6' THEN 'NCIT_C13222'  WHEN '7' THEN 'NCIT_C13223'  WHEN '8' THEN 'NCIT_C13224'  WHEN '9' THEN 'NCIT_C13225'  WHEN '10' THEN 'NCIT_C13205'
WHEN '11' THEN 'NCIT_C13206'   WHEN '12' THEN 'NCIT_C13207'   WHEN '13' THEN 'NCIT_C13208'   WHEN '14' THEN 'NCIT_C13209'   WHEN '15' THEN 'NCIT_C13210'   WHEN
'16' THEN 'NCIT_C13211'   WHEN '17' THEN 'NCIT_C13212'   WHEN '18' THEN 'NCIT_C13213'   WHEN '19' THEN 'NCIT_C13214'   WHEN '20' THEN 'NCIT_C13216'   WHEN '21'
THEN 'NCIT_C13217'   WHEN '22' THEN 'NCIT_C13218'    WHEN 'X' THEN 'NCIT_C13285'  WHEN 'Y' THEN 'NCIT_C13286'  END AS chromosome_id, chromosome_pos, cds_pos,
aa_pos_uniprotkb, mutation_freq, CASE data_source WHEN 'icgc' THEN 'https://icgc.org' WHEN 'cosmic' THEN 'https://cancer.sanger.ac.uk/cosmic' WHEN 'tcga' THEN
'https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga' WHEN 'civic' THEN 'https://civicdb.org' WHEN 'clinvar' THEN
'https://www.ncbi.nlm.nih.gov/clinvar' END AS data_source, doid, peptide_id, dm.ensembl_transcript_id, mp.uniprotkb_ac, gene_symbol FROM disease_mutation as dm
join map_protein_disease_mutation as mp on mp.ensembl_transcript_id = dm.ensembl_transcript_id left join xref_gene_uniprot as hugo on
hugo.uniprotkb_ac=mp.uniprotkb_ac
```

sequence_alteration

```
oncomx:SEQ_ALT_{id}-{peptide_id} a obo:SO_0001059 ; faldo:reference <https://identifiers.org/ensembl:{peptide_id}> ; faldo:location
oncomx:LOCATION_PROT_{id}-{peptide_pos} ; :alteredFrom obo:{ref_aa} , sio:{ref_aa_SIO} , obo:{ref_aa_X} ; :alteredTo obo:{alt_aa} , sio:{alt_aa_SIO} , obo:{alt_aa_X} .
oncomx:LOCATION_PROT_{id}-{peptide_pos} a faldo:ExactPosition ; faldo:position {peptide_pos}^^xsd:integer . oncomx:SEQ_ALT_{id}-{uniprotkb_ac} a obo:SO_0001059 ;
faldo:reference <http://purl.uniprot.org/uniprot/{uniprotkb_ac}> ; faldo:location oncomx:LOCATION_PROT_{id}-{aa_pos_uniprotkb} ; :alteredFrom obo:{ref_aa} ,
sio:{ref_aa_SIO} , obo:{ref_aa_X} ; :alteredTo obo:{alt_aa} , sio:{alt_aa_SIO} , obo:{alt_aa_X} . oncomx:LOCATION_PROT_{id}-{aa_pos_uniprotkb} a faldo:ExactPosition ;
faldo:position {aa_pos_uniprotkb}^^xsd:integer . oncomx:SEQ_ALT_{id}-{gene_symbol} a obo:SO_0001059 ; faldo:reference
<https://identifiers.org/hgnc.symbol:{gene_symbol}> ; faldo:location oncomx:LOCATION_GENE_{id}-{cds_pos} ; :alteredFrom obo:{ref_nt} ; :alteredTo obo:{alt_nt} .
oncomx:LOCATION_GENE_{id}-{gene_symbol} a faldo:ExactPosition ; faldo:position {cds_pos}^^xsd:integer .
```

```
SELECT id, CASE ref_nt WHEN 'A' THEN 'CHEBI_16708'  WHEN 'C' THEN 'CHEBI_16040'  WHEN 'G' THEN 'CHEBI_16235'  WHEN 'T' THEN 'CHEBI_17821'  WHEN 'U' THEN
'CHEBI_17568' END AS ref_nt, CASE alt_nt WHEN 'A' THEN 'CHEBI_16708'  WHEN 'C' THEN 'CHEBI_16040'  WHEN 'G' THEN 'CHEBI_16235'  WHEN 'T' THEN 'CHEBI_17821'  WHEN
'U' THEN 'CHEBI_17568' END AS alt_nt, cds_pos,  aa_pos_uniprotkb, CASE ref_aa WHEN 'A' THEN 'CHEBI_16449'  WHEN 'C' THEN 'CHEBI_15356'  WHEN 'D' THEN
'CHEBI_22660'  WHEN 'E' THEN 'CHEBI_18237'  WHEN 'F' THEN 'CHEBI_28044'  WHEN 'G' THEN 'CHEBI_15428'  WHEN 'H' THEN 'CHEBI_15971'  WHEN 'I' THEN 'CHEBI_17191'
WHEN 'K' THEN 'CHEBI_25094'  WHEN 'L' THEN 'CHEBI_25017'  WHEN 'M' THEN 'CHEBI_16811'  WHEN 'N' THEN 'CHEBI_22653'  WHEN 'P' THEN 'CHEBI_17203'  WHEN 'Q' THEN
'CHEBI_18050'  WHEN 'R' THEN 'CHEBI_16467'   WHEN 'S' THEN 'CHEBI_17115'   WHEN 'T' THEN 'CHEBI_16857'   WHEN 'V' THEN 'CHEBI_16414'   WHEN 'W' THEN
'CHEBI_27897'   WHEN 'Y' THEN 'CHEBI_18186'  END AS ref_aa, CASE ref_aa  WHEN '*' THEN 'SIO_010448' END AS ref_aa_SIO, CASE ref_aa WHEN 'X' THEN 'ANY_CODON'   END
AS ref_aa_X,CASE alt_aa WHEN 'A' THEN 'CHEBI_16449'  WHEN 'C' THEN 'CHEBI_15356'  WHEN 'D' THEN 'CHEBI_22660'  WHEN 'E' THEN 'CHEBI_18237'  WHEN 'F' THEN
'CHEBI_28044'  WHEN 'G' THEN 'CHEBI_15428'  WHEN 'H' THEN 'CHEBI_15971'  WHEN 'I' THEN 'CHEBI_17191'  WHEN 'K' THEN 'CHEBI_25094'  WHEN 'L' THEN 'CHEBI_25017'
WHEN 'M' THEN 'CHEBI_16811'  WHEN 'N' THEN 'CHEBI_22653'  WHEN 'P' THEN 'CHEBI_17203'  WHEN 'Q' THEN 'CHEBI_18050'  WHEN 'R' THEN 'CHEBI_16467'   WHEN 'S' THEN
'CHEBI_17115'   WHEN 'T' THEN 'CHEBI_16857'   WHEN 'V' THEN 'CHEBI_16414'   WHEN 'W' THEN 'CHEBI_27897'   WHEN 'Y' THEN 'CHEBI_18186'  END AS alt_aa, CASE alt_aa
WHEN '*' THEN 'SIO_010448' END AS alt_aa_SIO, CASE alt_aa WHEN 'X' THEN 'ANY_CODON'   END AS alt_aa_X, peptide_pos, mutation_freq, data_source, doid, peptide_id,
dm.ensembl_transcript_id, mp.uniprotkb_ac, gene_symbol FROM disease_mutation as dm join map_protein_disease_mutation as mp on mp.ensembl_transcript_id =
dm.ensembl_transcript_id left join xref_gene_uniprot as hugo on hugo.uniprotkb_ac=mp.uniprotkb_ac
```

Mapping size: 36   Search (any of):

VKG mapping patterns

Several approaches and tools supporting the creation of mappings have been proposed, several of them based on mapping patterns.

We build on well-established methodologies and patterns studied in:

- data management – e.g., W3C Direct Mapping Specification, [Arenas et al. 2012] and extensions,
- data analysis – e.g., algorithms for discovering dependencies, and
- conceptual modeling.

We have defined a **catalog of VKG mapping patterns**, where for each pattern we **take into account all the available information**:

- the domain knowledge that is encoded in ontology axioms
- the relational DB schema with its constraints (keys, foreign keys, . . .)
- data stored in the DB, when available
- the conceptual schema at the basis of the relational schema

Two major groups of mapping patterns

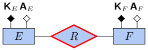
Schema-driven patterns

Are shaped by the structure of the DB schema and its explicit constraints.

Data-driven patterns

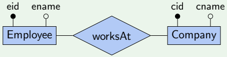
- Consider also constraints emerging from specific configurations of the data in the DB.
- For each schema-driven pattern, we identify a data-driven version:
  The constraints over the schema are not explicitly specified, but hold in the data.
- We provide also data-driven patterns that do not have a schema-driven counterpart.

Example of schema-driven patterns

E-R DIAGRAM	DB SCHEMA	MAPPING	ONTOLOGY
		Schema Entity (SE)	
	$T_E(\underline{\mathbf{K}}, \mathbf{A})$	s: T_E t: $C_E(\mathbf{iri}_E(\mathbf{K}))$, $\{d_A(\mathbf{iri}_E(\mathbf{K}), A)\}_{A \in \mathbf{K} \cup \mathbf{A}}$	$\{\exists d_A \sqsubseteq C_E\}_{A \in \mathbf{K} \cup \mathbf{A}}$
		Schema Relationship (SR)	
 $\mathbf{K}_E$ $\mathbf{A}_E$ $\mathbf{K}_F$ $\mathbf{A}_F$ E R F	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E)$ $T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$ $T_R(\underline{\mathbf{K}_{RE}, \mathbf{K}_{RF}})$	s: T_R t: $p_R(\mathbf{iri}_E(\mathbf{K}_{RE}), \mathbf{iri}_F(\mathbf{K}_{RF}))$	$\exists p_R \sqsubseteq C_E$ $\exists p_R^- \sqsubseteq C_F$

In case of $(\_, 1)$ cardinality on role R_E (resp., R_F), the primary key for T_R is restricted to the attributes $\mathbf{K}_{RE}$ (resp., $\mathbf{K}_{RF}$).

Conceptual



DB Schema

<u>eid</u>	name
15	Johnny

Employee

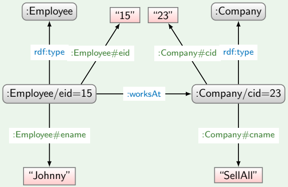
<u>employee</u>	<u>company</u>
15	23

worksAt

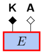
<u>c_id</u>	name
23	SellAll

Company

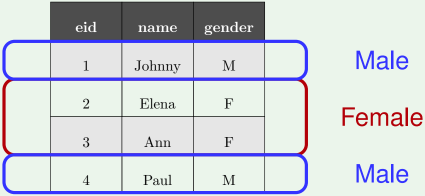
RDF (data only)



Example of a data-driven pattern [C., Gal, Lanti, et al. 2020]

E-R DIAGRAM	DB SCHEMA	MAPPING	ONTOLOGY
Clustering Entity to Class (CE2C)			
 $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$, $\mathit{partition}_{\mathcal{D}}(\mathbf{B}, E)$	$T_E(\mathbf{K}, \mathbf{A})$ $\mathsf{unique}_{T_E}(\mathbf{K})$ $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$ $\mathit{partition}_{\mathcal{D}}(\mathbf{B}, E)$ $\{V_{E_{\mathbf{v}}}(\mathbf{K}, \mathbf{A}) = \sigma_{\mathbf{B}=\mathbf{v}}(T_E)\}_{\mathbf{v} \in \pi_{\mathbf{B}}(T_E)}$	$\{s : \sigma_{\mathbf{B}=\mathbf{v}}(T_E)$ $t : C_E^{\mathbf{v}}(\mathbf{iri}_E(\mathbf{K}))\}_{\mathbf{v} \in \pi_{\mathbf{B}}(T_E)}$	$\{C_E^{\mathbf{v}} \sqsubseteq C_E\}_{\mathbf{v} \in \pi_{\mathbf{B}}(T_E)}$

K
A
E



eid	name	gender
1	Johnny	M
2	Elena	F
3	Ann	F
4	Paul	M

Male
Female
Male

Employee

Design scenarios for VKG mapping patterns

Depending on what information is available, we can consider different design scenarios where the patterns can be applied:

1. **Debugging of a VKG specification** that is already in place.
2. **Conceptual schema reverse engineering** for a DB that represents the domain of interest by using a given full VKG specification.
3. **Mapping bootstrapping** for a given DB and ontology that miss the mappings relating them.
4. **Ontology + mapping bootstrapping** from a given DB with constraints, and possibly a conceptual schema.
5. **VKG bootstrapping**, where the goal is to set up a full VKG specification from a conceptual schema of the domain.

The *Ontopic* spinoff of unibz

ONTOPIC

https://ontopic.ai/

First spin-off of the Free University of Bozen-Bolzano (funded in 2019).

- Ontopic Studio
  - Advanced tool for VKG design and deployment
  - Ensures scalability, reliability, and cost-efficiency at design and runtime of VKG solutions
  - Strong focus on usability
- Technical services
  - Technical support for Ontop and Ontopic Studio
  - Customized developments
- Consulting on adoption of VKG-based solutions for data access and integration

unibz

Outline

1. Virtual Knowledge Graphs (VKGs) for Data Access
2. Components of the VKG Architecture
3. Query Answering in VKGs
4. The Ontop System
5. Designing a VKG System
6. Conclusions

Conclusions

- The VKG approach is an innovative paradigm for data access and integration.
- The design of VKG mappings is a complex task, that currently is the major bottleneck in the wider adoption of the VKG paradigm.
- Mapping patterns are a promising approach for simplifying this complex task.
- In our work, we have identified a catalog of mapping patterns for the VKG framework and validated them in real-world use cases [C., Gal, Lanti, et al. 2020].
- We have defined algorithmic techniques to extract semantics from relational data sources, by automatically applying ontology mapping patterns to schema fragments [C., Gal, Haba, et al. 2021].
- Based on this, we have developed tools for the automatic generation of ontologies and mappings from relational data sources.

Thank you!

- E: calvanese@inf.unibz.it
- H: `http://www.inf.unibz.it/~calvanese/`

- *Ontop* website: `https://ontop-vkg.org/`
- Github: `https://github.com/ontop/ontop/`
- Facebook: `https://www.facebook.com/obdaontop/`
- Twitter: @ontop4obda
- *Ontopic* website: `https://ontopic.ai/`

References I

[1] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux & Juan Sequeda. *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. Available at http://www.w3.org/TR/rdb-direct-mapping/. World Wide Web Consortium, Sept. 2012.

[2] Diego C., Avigdor Gal, Naor Haba, Davide Lanti, Marco Montali, Alessandro Mosca & Roee Shraga. "ADaMaP: Automatic Alignment of Data Sources using Mapping Patterns". In: *Proc. of the 33rd Int. Conf. on Advanced Information Systems Engineering (CAiSE 2021)*. Vol. 12751. Lecture Notes in Computer Science. Springer, 2021, pp. 193–209. DOI: 10.1007/978-3-030-79382-1_12.

[3] Diego C., Avigdor Gal, Davide Lanti, Marco Montali, Alessandro Mosca & Roee Shraga. *Mapping Patterns for Virtual Knowledge Graphs*. CoRR Technical Report arXiv:2012.01917. arXiv.org e-Print archive, 2020.

[4] Jérôme Euzenat & Pavel Shvaiko. *Ontology Matching*. Springer, 2007.

[5] Nicola Guarino & Christopher A. Welty. "An Overview of OntoClean". In: *Handbook on Ontologies*. Ed. by Steffen Staab & Rudi Studer. International Handbooks on Information Systems. Springer, 2009, pp. 201–220. DOI: 10.1007/978-3-540-92673-3_9.

References II

[6] Erhard Rahm & Philip A. Bernstein. “A Survey of Approaches to Automatic Schema Matching”. In: *Very Large Database J.* 10.4 (2001), pp. 334–350.

[7] Dimitrios-Emmanuel Spanos, Periklis Stavrou & Nikolas Mitrou. “Bringing relational databases into the Semantic Web: A survey”. In: *Semantic Web J.* 3.2 (2012), pp. 169–209.

[8] Guohui Xiao, Diego C., Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati & Michael Zakharyaschev. “Ontology-Based Data Access: A Survey”. In: *Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. IJCAI Org., 2018, pp. 5511–5519. DOI: 10.24963/ijcai.2018/777.