Conceptually-grounded Mapping Patterns for Virtual Knowledge Graphs (Extended Abstract)

D. Calvanese\textsuperscript{1,2}, A. Gal\textsuperscript{3}, D. Lanti\textsuperscript{1}, M. Montali\textsuperscript{1}, A. Mosca\textsuperscript{1} and R. Shraga\textsuperscript{4}

\textsuperscript{1}Free-University of Bozen-Bolzano, Bolzano, Italy
\textsuperscript{2}Umeå University, Umeå, Sweden
\textsuperscript{3}Technion – Israel Institute of Technology, Haifa, Israel
\textsuperscript{4}Khoury College of Computer Science, Northeastern University, Boston, Massachusetts

Abstract

Knowledge Graphs (KGs) have been gaining momentum recently in both academia and industry, due to the flexibility of their data model, allowing one to access and integrate collections of data of different forms. Virtual Knowledge Graphs (VKGs), a variant of KGs originating from the field of Ontology-based Data Access (OBDA), are a promising paradigm for integrating and accessing legacy data sources. The main idea of VKGs is that the KG remains virtual: the end-user interacts with a KG, but queries are reformulated on-the-fly as queries over the data source(s). To enable the paradigm, one needs to define declarative mappings specifying the link between the data sources and the elements in the VKG. In this work, we try to investigate common patterns that arise when specifying such mappings, building on well-established methodologies from the area of conceptual modeling and database design.

Keywords

Virtual Knowledge Graphs, Ontology-based Data Access, Mapping patterns, Data Integration

1. Introduction

Data integration and access to legacy data sources are key challenges for contemporary organizations. In the whole spectrum of data integration and access solutions, the approach based on Virtual Knowledge Graphs (VKGs) is gaining momentum, especially when the underlying data sources to be integrated come in the form of relational databases (DBs)\textsuperscript{[1]}. VKGs replace the rigid structure of tables with the flexibility of a graph that incorporates domain knowledge and is kept virtual, eliminating redundancies. A VKG specification consists of three main components: (i) data sources (in the context of this paper, constituted by relational DBs), where the actual data are stored; (ii) a domain ontology, capturing the relevant concepts, relations, and constraints of the domain of interest; and (iii) a set of mappings, linking the data sources to the ontology. A critical bottleneck in this setting lies in the definition and management of mappings. In this work, we focus on this issue by proposing a comprehensive catalog of mapping patterns that emerge when linking data to ontologies. Our catalog is based on the (somehow
reasonable) assumption that both the ontology and the DB schema are derived from a conceptual analysis of the domain of interest. The resulting knowledge may stay implicit, or may lead to an explicit representation in the form of a structural conceptual model, which can be represented using well-established notations such as UML, ORM, or E-R. On the one hand, this conceptual model provides the basis for creating a corresponding domain ontology through a series of semantic-preserving transformation steps. On the other hand, it can trigger the design process that finally leads to the deployment of an actual DB. The whole view is depicted in Figure 1.

Our catalog is built on well-established methodologies and patterns studied in data management (e.g., W3C direct mappings (W3C-DM)\(^1\) and extensions), data analysis (e.g., algorithms for discovering dependencies), and conceptual modeling (e.g., relational mapping techniques).

The idea of mapping patterns is not new. For instance, work in [2] is closely related to ours, as it also introduces a catalog of mapping patterns. However, there are some key differences with our approach. One difference is that we consider KGs (with ontologies), whereas that work focuses on property graphs without an ontology. More importantly, in [2] and in the related literature, patterns are not formalized or grounded to a specific conceptual representation, but are rather informally specified and discussed in a "by-example" fashion. On the contrary, each of our patterns explicitly and non-ambiguously specifies the link between the conceptualization and the DB instance, which is the one arising from applying well-known semantics-preserving transformations studied in the area of DB design.

We argue that this foundational grounding paves the way for a variety of VKG design scenarios, depending on which information artifacts are available, and which ones must be produced. For example, our patterns could be used to validate existing mappings, or to automatically generate (i.e., bootstrap) ontology and mappings when only the DB is available. In fact, specific patterns have been proposed also in relation to ontology and mapping bootstrapping, for which a variety of tools and approaches have been developed in the last two decades [3, 4, 5, 6, 7]. The approaches in the literature differ in terms of the overall purposes of bootstrapping (e.g., OBDA, data integration, ontology learning, checking of DB schema constraints using ontology reasoning), the adopted ontology and mapping languages (e.g., OWL 2 profiles or RDFS as ontology languages, and R2RML or custom languages for the specification of mappings), the different focus on direct and/or complex mappings, and the assumed level of automation. The majority of the most recent approaches closely follow W3C-DM, deriving ontologies that mirror the structure of the input DB.

\(^1\)http://www.w3.org/TR/rdb-direct-mapping/
Table 1
Semantics of the \textit{DL-Lite}_R constructs that involve datatypes.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax Element</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top domain</td>
<td>( T_V )</td>
<td></td>
<td>( \Delta^T_V )</td>
</tr>
<tr>
<td>Literal</td>
<td>( \ell \in \text{NL} )</td>
<td>“george”</td>
<td>( \ell^2 \in \Delta^T_V )</td>
</tr>
<tr>
<td>Datatype</td>
<td>( T_i )</td>
<td>xsd:int</td>
<td>( T^T_i \subseteq \Delta^T_V )</td>
</tr>
<tr>
<td>Data property name</td>
<td>( d \in \text{ND} )</td>
<td>hasName</td>
<td>( d^T \subseteq \Delta^T_O \times \Delta^T_V )</td>
</tr>
<tr>
<td>Data property domain</td>
<td>( \delta(d) )</td>
<td>( \delta(\text{hasName}) )</td>
<td>( { x \in \Delta^T_V \mid \exists v \in \Delta^T_V : (x, v) \in d^T } )</td>
</tr>
<tr>
<td>Data property range</td>
<td>( \rho(d) )</td>
<td>( \rho(\text{hasName}) )</td>
<td>( { v \in \Delta^T_V \mid \exists o \in \Delta^T_O : (o, v) \in d^T } )</td>
</tr>
<tr>
<td>Data property negation</td>
<td>( \neg d )</td>
<td>( \neg \text{hasName} )</td>
<td>( \Delta^T_O \times \Delta^T_V \setminus d^T )</td>
</tr>
</tbody>
</table>

The remainder of the paper is structured as follows: Section 2 introduces the notation and basic notions on VKGs, Section 3 presents (an extract of) our catalog of mapping patterns, and Section 4 concludes the paper.

### 2. Preliminaries

We use the \textbf{bold} font to denote tuples, e.g., \( x, y \), are tuples. When convenient and non-ambiguous, we treat tuples as sets and use set operators on them. We assume familiarity with standard notions and languages from DBs \cite{8}, such as SQL or E-R diagrams.

A VKG specification is a triple \( \langle T, M, S \rangle \) where \( T \) is an ontology (or TBox), \( M \) a set of mappings, and \( S \) the schema of a DB (with constraints, e.g., primary and foreign keys). In VKGs, the ontology is formulated in OWL 2 QL\(^2\), but for conciseness we use its Description Logic (DL) counterpart, \textit{DL-Lite}\(_R\) \cite{9}, here slightly enriched to handle datatypes.

We fix the following enumerable, pairwise-disjoint sets: \textbf{NI} of individuals, \textbf{NL} of literal values, \textbf{NC} of class names, \textbf{NP} of object property names, and \textbf{ND} of data property names.

An OWL 2 QL TBox \( T \) is a finite set of inclusion axioms of the form \( B \sqsubseteq C, q \sqsubseteq r, \rho(d) \sqsubseteq f, \) or \( d \sqsubseteq v, \) where \( B, C \) are classes, \( q, r \) are object properties, \( d \) is a data property, \( f \) is a datatype expression, \( \rho(d) \) is a \textit{data property range expression}, and \( v \) is a \textit{data property expression}. These are defined according to the following grammar, where \( A \in \text{NC}, d \in \text{ND}, p \in \text{NP}, \delta(d) \) is a \textit{data property domain expression}, and \( T_1, \ldots, T_n \) are the RDF datatypes:

\[
\begin{align*}
B & \rightarrow A \mid \exists r \mid \delta(d) \\
C & \rightarrow T_C \mid B \mid \neg B \\
q & \rightarrow p \mid p^- \\
f & \rightarrow \top_D \mid T_1 \mid \cdots \mid T_n \\
r & \rightarrow q \mid \neg q \\
v & \rightarrow d \mid \neg d
\end{align*}
\]

In the rules above, \( T_C \) denotes the “top” element for concepts and \( \top_D \) the one for data values (called \textit{literals} in the RDF terminology). An OWL 2 QL ABox \( A \) is a finite set of assertions of the form \( A(a), p(a, b), \) or \( d(a, \ell), \) where \( A \in \text{NC}, p \in \text{NP}, d \in \text{ND}, a \) and \( b \) are individuals in \textbf{NI}, and \( \ell \in \textbf{NL}. \) We call the pair \( K = \langle T, A \rangle \) an OWL 2 QL Knowledge Graph (KG).

Similarly to first-order logic, the semantics of \textit{DL-Lite}_R KGs is given through Tarski-style interpretations \( \mathcal{I} = \langle \Delta^T_O, \Delta^T_V, \cdot^T \rangle, \) where \( \Delta^T_O \) is a non-empty \textit{domain of objects}, \( \Delta^T_V \) is a non-

\(^{2}\)http://www.w3.org/TR/owl2-overview/
empty domain of values, and \( \mathcal{I} \) is an interpretation function. Table 1 reports the semantics for the constructs involving datatypes. The other constructs are defined as in standard DL-Lite\( ^R \) [9]. As usual [10], we say that an interpretation \( \mathcal{I} \) satisfies a KG \( \mathcal{K} \), denoted by \( \mathcal{I} \models \mathcal{K} \), if \( \mathcal{I} \) satisfies the ABox assertions and the inclusion axioms in \( \mathcal{K} \).

Mappings. Mappings specify how to populate classes and properties of the ontology with individuals and values constructed from the data in the underlying DB. In other words, mappings provide the ABox that, together with a given TBox, realizes a KG. In VKGs, the adopted language for mappings in real-world systems is R2RML\(^3\), but for conciseness we use here a more convenient abstract notation inspired by the literature [11]: a mapping \( m \) is a pair of the form \( \langle s : Q(x), t : L(t(x)) \rangle \), where \( Q(x) \) is a SQL query with answer variables \( x \) over the DB schema \( \mathcal{S} \), called source query, and \( L(t(x)) \) is a list of target atoms of the form \( C(t_1(x_1)), p(t_1(x_1), t_2(x_2)), \) or \( d(t_1(x_1), t_2(x_2)) \), where \( C \in \mathcal{NC}, p \in \mathcal{NP}, d \in \mathcal{ND}, \) and \( t_1(x_1) \) and \( t_2(x_2) \) are terms that we call templates. We express source queries in relational algebra, omitting answer variables under the assumption that they coincide with the variables used in the target atoms.

Intuitively, a template \( t(x) \) in the target atom of a mapping corresponds to an R2RML string template\(^4\), and is used to generate an IRI (hence, an object identifier) or an RDF literal, starting from DB values retrieved by the source query in that mapping. For the examples, we use the concrete syntax from the Ontop VKG system [6], in which the source query is expressed in SQL and each target atom is expressed as an RDF triple pattern with templates. The answer variables of the source query occurring in the target atoms are distinguished by enclosing them in curly brackets \{ \ldots \}. The following is an example mapping expressed in such syntax:

```
source   SELECT ssn FROM person
target   ex:pers/{ssn} a ex:Person .
```

In the mapping above, the string ex: denotes a URI prefix, e.g., ex:Person is an abbreviation for the URI http://www.example.com/Person. Such mapping, when applied to a DB instance \( \mathcal{D} \) of \( \mathcal{S} \), populates the class ex:Person with IRIs constructed by replacing the answer variable ssn occurring in the target atom with the corresponding values assigned to that variable by the answers to the SQL source query evaluated over \( \mathcal{D} \). For instance, if the source query returns two answers that assign to the answer variable ssn respectively the values 1234 and 5678, then the mapping above produces the following RDF graph (expressed in the Turtle syntax\(^5\)), stating that individuals ex:pers/1234 and ex:pers/5678 are both instances of class ex:Person:

```
ex:pers/1234 a ex:Person .
ex:pers/5678 a ex:Person .
```

We denote by \( A_M(\mathcal{D}) \) the virtual ABox constructed through mappings \( M \) from a DB \( \mathcal{D} \). Given a VKG specification \( \langle \mathcal{T}, M, \mathcal{S} \rangle \) and a database instance \( \mathcal{D} \) of \( \mathcal{S} \), the KG \( \mathcal{K} = \langle \mathcal{T}, A_M(\mathcal{D}) \rangle \) is called the Virtual Knowledge Graph of \( \langle \mathcal{T}, M, \mathcal{S} \rangle \) through \( \mathcal{D} \). The qualifier “virtual” in the name derives from the fact that the virtual ABox \( A_M(\mathcal{D}) \) in a VKG setting is not materialized and stored somewhere. Query answering in VKGs, in fact, is carried out through query rewriting.

\(^3\)http://www.w3.org/TR/r2rml/
\(^4\)https://www.w3.org/TR/r2rml/#dfn-string-template
\(^5\)http://www.w3.org/TR/turtle/
and query unfolding techniques [11, 6]: user queries, expressed in SPARQL ⁶, get translated on-the-fly into equivalent SQL queries, which then are directly evaluated against the DB.

3. Mapping Patterns

In its basic form, a mapping pattern is a quadruple \( \langle \mathcal{C}, \mathcal{S}, \mathcal{M}, \mathcal{T} \rangle \), where \( \mathcal{C} \) is a conceptual model, \( \mathcal{S} \) a database schema, \( \mathcal{M} \) a set of mappings, and \( \mathcal{T} \) an (OWL 2 QL) ontology. In such pattern, the pair \( \langle \mathcal{C}, \mathcal{S} \rangle \) puts into correspondence a conceptual representation with one of its (many) admissible (i.e., formally sound [12, 13]) database schemata, like those prescribed by well-established database modeling methodologies. The pair \( \langle \mathcal{M}, \mathcal{T} \rangle \), instead, is formed by the DB ontology \( \mathcal{T} \), which is the OWL 2 QL encoding⁷ of the conceptual model \( \mathcal{C} \), and the set \( \mathcal{M} \) of mappings, providing the link between \( \mathcal{S} \) and \( \mathcal{T} \). The term “DB ontology” refers to an ontology whose concepts and properties reflect the constructs of the conceptual model, mirroring the structure of the relational database, as displayed in Figure 1.

Some of the more advanced patterns have a more complex structure, where pairs of conceptual models and/or pairs of database schemata are used in place of \( \mathcal{C} \) and \( \mathcal{S} \), respectively (e.g., the pattern “SHa” falls in this category). These patterns prescribe specific transformations to be applied to an input conceptual (resp., DB) schema, in order to obtain an output conceptual (resp., DB) schema. These output artifacts make explicit the presence of specific structures that are revealed through the application of the pattern itself. These structures can in turn enable further applications of patterns.

Presentation Conventions. We show the fragment of the conceptual model that is affected by the pattern in E-R notation (adopting the original notation by Chen [14]). To compactly represent sets of attributes, we use a small diamond in place of the small circle used for single attributes in Chen’s notation. For cardinality constraints we follow the “look-here” convention, that is, the cardinality constraint for a role is placed next to the entity participating in that role. In the DB schema, we use \( T(\mathbf{K}, \mathbf{A}) \) to denote a table with name \( T \), primary key consisting of the attributes \( \mathbf{K} \), and additional attributes \( \mathbf{A} \). Given a set \( U \) of attributes in \( T \), we denote by \( \text{key}_T(U) \) the fact that \( U \) form a key for \( T \). Referential integrity constraints (like, e.g., foreign keys) are denoted with arcs, pointing from the referencing attribute(s) to the referenced one(s). For conciseness, we denote sets of the form \( \{ o \mid \text{condition} \} \) as \( \{ o \} \text{condition} \). In order to express datatypes for data properties, we introduce two auxiliary functions: a function \( \tau \) that, given a DB attribute \( A \), returns the DB datatype of \( A \), and a function \( \mu \) that associates, to each DB datatype, a corresponding RDF datatype. For the definition of \( \mu \), we re-use the Natural Mapping⁸ correspondence provided by the R2RML recommendation. As a final note, following the E-R-diagrams convention, we assume a default \((1, 1)\) cardinality on attributes. For such a reason, in the DB schema we assume all attributes to be not nullable by default (using the SQL convention, declared as “\( \text{NOT NULL} \)”). An optional attribute \( A \) is instead denoted by adding \( \text{opt}(A) \) to the DB schema. Such notation extends in the natural way to a set \( \mathbf{A} \) of attributes.

⁶http://www.w3.org/TR/sparql11-query
⁷Modulo the expressivity of the OWL 2 QL language.
⁸https://www.w3.org/TR/r2rml/#natural-mapping
Table 2
An extract of our catalog of mapping patterns.

<table>
<thead>
<tr>
<th>Conceptual Model</th>
<th>DB Schema</th>
<th>Mappings</th>
<th>Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T_E(K, A)</td>
<td>s: T_E</td>
<td>{δ(d_A) ⊆ C_E, p(δ(d_A)) \subseteq μ(τ(A)), C_E \subseteq δ(δ(d_A)) }_δA, pA</td>
</tr>
<tr>
<td></td>
<td>T_R(K_E, A_V)</td>
<td>t: C_E(t_E(K)), {d_A(t_E(K), A) }_A</td>
<td></td>
</tr>
</tbody>
</table>

In case of optional attributes, for each optional attribute A’ of E, add an opt(A’) constraint to the DB schema and drop the corresponding inclusion axiom C_E \subseteq δ(δ(d_A)) from the ontology.

<table>
<thead>
<tr>
<th>Schema Relationship (SR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_E(K_E, A_V)</td>
</tr>
<tr>
<td>s: T_E</td>
</tr>
<tr>
<td>{δ(d_A) \subseteq C_E, p(δ(d_A)) \subseteq μ(τ(A)), C_E \subseteq δ(δ(d_A)) }_δA, pA</td>
</tr>
</tbody>
</table>

- In case of cardinality (\_\_\_\_) on role R_E (resp., R_F), the primary key of T_E is restricted to the attributes K_E (resp., K_F). In case both roles have cardinality (\_\_\_\_), either choice for the primary key is made, and the remaining attributes form a non-primary key in the logical schema.
- In case of cardinality (\_\_\_) on role R_E (resp., R_F), the inclusion dependency K_E \subseteq K_E (resp., K_F \subseteq K_R_F) holds in the schema, and the first (resp., second) inclusion axiom in the ontology holds in both directions. Note that when the maximum cardinality on role R_E (resp., R_F) is 1, the corresponding inclusion dependency is actually a foreign key.

Pattern Catalog. Table 2 shows an excerpt of our patterns, which we discuss in detail here.

**Schema Entity (SE).** This fundamental pattern describes the correspondence between an entity with a primary identifier and attributes in the DB schema, and a class and data properties in the ontology. The entity is expressed in the DB schema through a single table T_E with primary key K and other attributes A, as it is the norm in sound DB design practices. The mappings column explains how T_E is mapped into a corresponding class C_E. The primary key of T_E is employed to construct the IRIs of the objects that are instances of C_E, using a template t_E specific for that entity. Each relevant attribute of T_E is mapped to a data property of C_E, with suitable domain and range axioms. A mandatory participation constraint is added to each data property corresponding to a mandatory attribute.

**Example:** A client registry table containing SSNs of clients, together with their name as an additional attribute, is mapped to a Client class using the SSN to construct its objects. In addition, the SSN and name are mapped to two corresponding data properties.

**Schema Relationship (SR).** This pattern describes the correspondence between a binary relationship without attributes and an OWL 2 QL object property, for the case where such relationship
is represented in the DB as a separate (usually, “many-to-many”) table. This pattern considers three tables $T_R$, $T_E$, and $T_F$, for which the set of columns in $T_R$ is partitioned into two parts $K_{RE}$ and $K_{RF}$ that are foreign keys to $T_E$ and $T_F$, respectively. The identifier of $T_R$ depends on the role cardinalities in the E-R model. The pattern captures how $T_R$ is mapped to an object property $p_R$, using the two parts $K_{RE}$ and $K_{RF}$ of the partition to construct respectively the subject and the object of the triples in $p_R$. The templates $t_{CE}$ and $t_{CF}$ must be those respectively used for building instances of classes $C_E$ corresponding to $T_E$ and $C_F$ corresponding to $T_F$.

**Example:** An additional table in the client registry stores the addresses of each client, and has a foreign key to a table with locations. The former table is mapped to an address object property, for which the ontology asserts that the domain is the class Person and the range an additional class Location, which corresponds to the latter table.

**Schema Relationship with Identifier Alignment (SRa).** This pattern is similar to pattern SR, but it comes with a modifier $a$, indicating that the pattern can be applied after the identifiers involved in the relationship have been aligned. The alignment is necessary because the foreign key in $T_R$ does not refer to the primary key $K_F$ of $T_F$, but to an alternative key $U_F$. Since the instances of the class $C_F$ corresponding to $T_F$ are constructed using the primary key $K_F$ of $T_F$ (cf. pattern SE), also the pairs that populate $p_R$ should refer in their object position to that primary key, which can only be retrieved via a join between $T_R$ and $T_F$ on the key $U_F$.

**Example:** The primary key of the table with locations is not given by the city and street, which are used in the table that relates clients to their addresses, but is given by the latitude and longitude of locations.

**Schema Hierarchy with Identifier Alignment (SHA).** This pattern handles the case where a hierarchy is specified and the child entity uses a primary identifier different from the one in the parent entity. In this situation, the foreign-key constraint can come in three different variants. In the depicted one, the foreign key in $T_F$ is over a non-primary key $K_{FE}$. The objects for $C_F$ have to be built out of $K_{FE}$, rather than out of the primary key of $T_F$. For this purpose, the pattern creates a view $V_F$ identical to $T_F$, except that $K_{FE}$ is the primary key. Also the foreign key relations are preserved. Such view might enable further applications of patterns.

**Example:** An ISA relation between entities Student and Person. Students are identified by their matriculation number, whereas persons are identified by their SSN.

### 4. Conclusions and Future Work

In this work, we have identified and formally specified a number of mapping patterns emerging when linking DBs to ontologies in a typical VKG setting. Our patterns are grounded in well-established practices of DB design, and render explicit the connection between the conceptual model, the DB schema, and the ontology. We envision that the organization in patterns can enable a number of relevant tasks, notably mapping bootstrapping for incomplete VKGs.

This work is only a first step, with respect to both categorization of patterns, and their actual use. Regarding the former, we are currently extending this initial catalog with more advanced “data-driven” patterns, which are patterns where the data component needs to be taken into account. Regarding the latter, we are investigating solutions to specific problems that need to be addressed when setting-up a VKG scenario, like the problem of mapping bootstrapping.
Acknowledgments

This research has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the Italian Basic Research (PRIN) project HOPE, by the EU H2020 project INODE (grant agreement 863410), and by the project MENS, funded through the 4th Call for Research of the Autonomous Province of Bolzano (IN221).9

References