Interactive Multidimensional Modeling of Linked Data for Exploratory OLAP[☆]

Enrico Gallinucci[a], Matteo Golfarelli[a], Stefano Rizzi[a,*], Alberto Abelló[b], Oscar Romero[b]

[a] *DISI – University of Bologna, Bologna, Italy*
[b] *ESSI – Universitat Politècnica de Catalunya, Barcelona, Spain*

Abstract

Exploratory OLAP aims at coupling the precision and detail of corporate data with the information wealth of LOD. While some techniques to create, publish, and query RDF cubes are already available, little has been said about how to contextualize these cubes with situational data in an on-demand fashion. In this paper we describe an approach, called iMOLD, that enables non-technical users to enrich an RDF cube with multidimensional knowledge by discovering aggregation hierarchies in LOD. This is done through a user-guided process that recognizes in the LOD the recurring modeling patterns that express roll-up relationships between RDF concepts, then translates these patterns into aggregation hierarchies to enrich the RDF cube. Two families of aggregation patterns are identified, based on associations and generalization respectively, and the algorithms for recognizing them are described. To evaluate iMOLD in terms of efficiency and effectiveness we compare it with a related approach in the literature, we propose a case study based on DBpedia, and we discuss the results of a test made with real users.

Keywords: Multidimensional modeling, Data warehouse design, Linked data, Exploratory OLAP

1. Introduction and Motivation

The goal of business intelligence (BI) is to transform operational data into information and knowledge to be used for decision making. To this end, data are periodically extracted, cleaned, transformed, and fed into a repository called data warehouse (DW), where they are stored in the form of multidimensional

[☆]This work was partly supported by the EU-funded project TOREADOR (contract n. H2020-688797).

[*]Corresponding author
Email addresses: enrico.gallinucci2@unibo.it (Enrico Gallinucci), matteo.golfarelli@unibo.it (Matteo Golfarelli), stefano.rizzi@unibo.it (Stefano Rizzi), aabello@essi.upc.edu (Alberto Abelló), oromero@essi.upc.edu (Oscar Romero)

cubes to be analyzed by decision makers using OLAP (On-Line Analytical Processing) front-ends. Though BI techniques have been enormously accelerating and improving the decision making process for almost two decades, companies face today a highly dynamic environment which requires including external factors (e.g., market or socioeconomic data) in the decision-making process. So, while traditional OLAP is based on *stationary data* (i.e., reliable data owned by the company itself), the recent years are witnessing a push towards enriching the DW with external *situational data* [1] in an on-demand fashion. Situational data are retrieved from outside the borders of the corporate information system and may not be fully reliable; they are used to contextualize stationary data and their lifespan is typically short and out of the company control. Unfortunately, traditional DW methods and techniques have been proved to be inappropriate when dealing with situational data and several visionary papers underline the need to develop original architectures and approaches to support this new scenario [2, 3, 4, 5].

When accessing situational data and integrating them in the decision-making process, knowing the data semantics is important; semantic web technologies, and ontologies in particular, are strong candidates to this end. In this direction, a new approach called *exploratory OLAP* [6] has recently emerged to describe the convergence of OLAP and the semantic web. Inspired by the Linked Open Data (LOD) initiative [7], exploratory OLAP advocates for publishing, sharing, and linking semantic multidimensional data to enable cross-domain OLAP analyses. This approach is beneficial to smart users (e.g., data scientists) who have a basic knowledge of multidimensional modeling and a strong skill on the application domain, and whose goal is to write ad hoc queries on situational data in a self-service fashion, i.e., without resorting to the help of IT people.

Two main challenges lie behind this approach. On the one hand, the semantic web relies on RDF for publishing, sharing and linking data; thus, the user needs to explore RDF repositories and build her own cubes by means of SPARQL, the *de facto* standard to express RDF queries. On the other hand, to support correct aggregations during OLAP analyses, these RDF cubes must be well-formed from the multidimensional point of view [8]. Both issues require a high degree of specialization, so enriching OLAP analyses with situational data is still hardly possible for non-technical users.

To address these challenges, exploratory OLAP adopts a *publish-enrich-query* paradigm:

1. **Publish**. This stage entails publishing and sharing RDF cubes, either starting from a corporate DW or from available RDF data sets. Several state-of-the-art efforts support this stage. A cornerstone in this direction is QB4OLAP [9], an RDF vocabulary to publish and share multidimensional data. QB4OLAP is an extension of the standard vocabulary for RDF cubes (QB) that enables OLAP analyses (otherwise impossible with QB alone). On top of that, several approaches provide means to generate RDF cubes from non-semantic sources (e.g., [10, 11]). In particular, the

QB2OLAPem tool enriches available QB-compliant RDF data sets with QB4OLAP constructs [8]. Alternatively, expert users may directly generate QB4OLAP-compliant RDF cubes [10].

2. **Enrich**. In order to provide 5-stars quality open data (`www.w3.org/DesignIssues/LinkedData.html`), RDF cubes must be *linked* to other RDF data sets that contextualize the data (mainly by means of synonym or subsumption axioms) and facilitate exploring and cross-referencing data among different sources. Automatic link generation has already been identified as a crucial step for the success of LOD, given its time-consuming and error-prone nature [7]; nevertheless, current approaches pay little attention to linking, which still needs to be done manually.

3. **Query**. Once RDF cubes have been published and enriched, they should be queried in an OLAP fashion. Recent works have developed OLAP-like languages on top of SPARQL (e.g., CQL), so that a non-technical user can query RDF cubes by applying high-level multidimensional operations such as roll-up and drill-down [12, 13].

The approach we propose in this paper, named *iMOLD* (Interactive Multidimensional Modeling of Linked Data), addresses the *enrich* stage of exploratory OLAP. Its goal is to contextualize a QB4OLAP-compliant RDF cube with situational multidimensional knowledge by discovering aggregation hierarchies in LOD. This is done in an interactive way, by first letting the user explore LOD in the light of her view of the business, and then recognizing structural patterns (called *aggregation patterns* from now on) that correspond to potential hierarchies.

While some existing ontology construction approaches address the problem of building hierarchies, they do not focus on *multidimensional* aggregation hierarchies and rather follow linguistic or statistical approaches [14, 15, 44], so the resulting hierarchies do not fulfill the three summarization conditions established by the state-of-the-art literature [16]. So, to our knowledge, no other approach supports the automation of the *enrich* stage. The original contributions of this work can be summarized as follows:

1. We propose an approach for enriching QB4OLAP-compliant RDF cubes with aggregation hierarchies discovered in LOD through a user-guided process.

2. We identify five aggregation patterns commonly found in ontologies.

3. We provide algorithms for recognizing these patterns in LOD and translating them into hierarchies.

The paper outline is as follows. After providing in Section 2 the necessary background on multidimensional modeling and LOD modeling, Section 3 gives

an overview of the iMOLD approach. Section 4 presents the different aggregation patterns, while Section 5 describes in detail the core techniques we use to recognize these patterns in LOD and translate them into hierarchies. Section 6 proposes a case study for iMOLD, presents the results of an evaluation test involving real users, discusses the approach efficiency, and compares it with a related approach in the literature. Finally, Section 7 summarizes the related work and Section 8 concludes the paper.

2. Background

2.1. Linked Data Modeling

The LOD initiative was envisioned by Tim Berners-Lee as "published data that can be machine-readable, its meaning is explicitly defined, it is linked to other external data sets and can be linked to from other external data sets" [17]. In LOD, *Universal Resource Identifiers* (URIs) are used as names for available resources (note the *universal* scope of URIs as identifiers) and the HTTP protocol should be used to dereference URIs so that people can locate and look up those names.

The formalism used to describe and link resources is the *Resource Description Framework* language (RDF), a W3C recommendation. The basic RDF block is the triple, a binary relationship between a subject and an object; i.e., <*subject predicate object*>. The subject and the predicate must be resources (i.e., identified by a URI), whereas the object can be either a resource or a literal (i.e., a constant value such a string or an integer).

A set of RDF triples form an RDF graph; it is usual to refer to RDF graphs as ontologies. RDF Schema (RDFS), a W3C recommendation, was introduced to express basic constraints on RDF triples. By means of the RDFS core classes (namely rdfs:Resource, rdfs:Class, rdfs:Literal, rdf:Property, and rdf:Statement) and of some predefined properties, one can distinguish between instances and classes (by using the rdf:type property), express property and class taxonomies by means of inclusion statements (by means of rdfs:subClassOf and rdfs:subPropertyOf), and type properties by specifying the allowed classes at its domain and range (by means of rdfs:domain and rdfs:range). The W3C recommends using the *SPARQL Protocol and RDF Query Language* to query RDF ontologies. Typically, RDF published graphs are exposed by means of (publicly available) SPARQL endpoints.

2.2. Multidimensional Modeling

The multidimensional model is the core of DW and OLAP applications. It is close to the way of thinking of data analyzers, who are used to spreadsheets; thus, it helps them understand data by smoothly supporting typical business analyses and enabling complex queries to be formulated with small effort even by non-IT users. Intuitively, the multidimensional model represents data under the metaphor of a cube whose cells correspond to events that occurred in the business domain. Each event is quantified by a set of measures; each axis of the

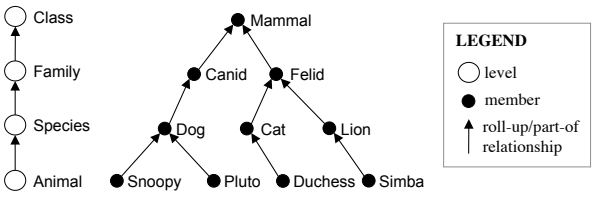

Figure 1: An example of multidimensional modeling: levels and roll-up relationships (left), members and part-of relationships (right)

cube corresponds to a relevant dimension for analysis, typically associated to a hierarchy of levels that further describe its potential aggregation. Our specific goal in this work is to discover hierarchies in LOD, for this reason we will focus on the modeling of hierarchies. In the following, we introduce the basic concepts we will use to this end.

Definition 1 (Hierarchy). *An* aggregation hierarchy *(or, briefly, a* hierarchy*) is a directed tree of* levels *rooted in a dimension. Each arc models a* roll-up relationship $u = (l, m, l')$ *between two levels, a* child l *and a* parent l'*, and has semantics* m.[1] *Each level has a domain made by a set of* members*. The roll-up relationship* u *abstracts a many-to-one* part-of relationship *on the members of* l *and* l'*, such that each member of* l *is part of exactly one member of* l'*.*

Following the reusability principle, in this work we use the QB4OLAP [9] vocabulary (namespace `qb4o`, `purl.org/olap#`) to annotate the hierarchies identified on RDF data. Although many datasets published in the LOD adopt the RDF Data Cube Vocabulary (QB, `www.w3.org/TR/vocab-data-cube`), the latter lacks in providing the constructs needed to enable OLAP analyses [9]. QB4OLAP does this by extending QB with additional resources; in particular, it allows structuring the cube dimensions in hierarchies and levels, relate measures with aggregation functions, and represent observations at different aggregation levels, enabling roll-up and drill-down operations over RDF-based data. As a result, QB4OLAP captures all the multidimensional modeling features presented in Definition 1. More specifically, QB4OLAP represents a hierarchy level as an instance of class qb4o:LevelProperty. A roll-up relationship between two levels is represented through an instance of class qb4o:HierarchyStep, which is linked to the two corresponding qb4o:LevelPropertys by means of the qb4o:parentLevel and qb4o:childLevel properties. Finally, the multiplicities of roll-up relationships are specified through the qb4o:Cardinality class.

[1] Considering also the semantics m in the definition of roll-up relationship is necessary to cope with the case in which the same two levels are involved in two different relationships (e.g., persons can roll-up to cities according to two different semantics, namely *lives in* and *was born in*).

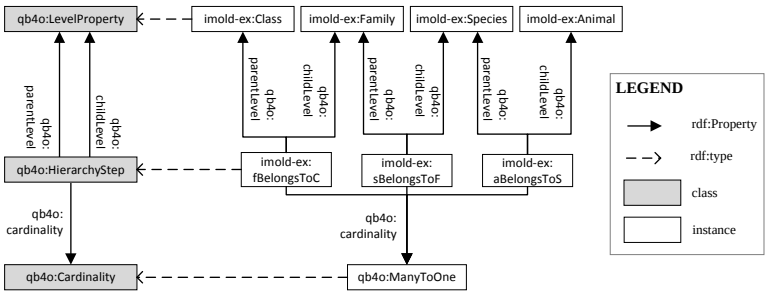

Figure 2: The QB4OLAP representation of the hierarchy of Figure 1

Example 1. *Consider the sample hierarchy in Figure 1. Levels are shown as white circles; for instance,* Species *rolls-up to (i.e., is a child of)* Family*. Members are shown as black circles; for instance,* Canid *and* Felid *(member instances of* Family*) are part of member* Mammal *(member instance of* Class*). In the QB4OLAP representation of this hierarchy (Figure 2), level* Species *is represented by object* imold-ex:Species*, instance of* qb4o:LevelProperty*; the three roll-up relationships (namely* imold-ex:aBelongsToS*,* imold-ex:sBelongsToF*, and* imold-ex:fBelongsToC*) are instances of* qb4o:HierarchyStep *and have many-to-one multiplicity. This specific hierarchy will be used throughout the paper as a working example; in particular, we will show how it can be discovered by the user by exploring LOD to enrich any RDF cube that stores observations about animals.*

In the literature, hierarchy discovery is typically done at design-time in the context of *supply-driven design* (i.e., mainly based on the schema of the source data [18]) and consider well-structured data sources described, for instance, by Entity/Relationship diagrams and relational schemata; there, hierarchies can be identified by simply following functional dependencies (FDs) at the schema level, like in [19]. Our scenario poses further challenges: (i) the modeling heterogeneity of LOD and the impossibility of describing the multiplicity of properties in the RDFS vocabulary make hierarchy discovery more complex since FDs must be identified at the instance level; (ii) iMOLD operates at querying time rather than at design time; (iii) it can be regarded as a *mixed* approach, because it combines access to data —typical of supply-driven design— with user interaction —typical of demand-driven design (i.e., mainly based on the user requirements).

3. Approach Overview

The basic idea of iMOLD is to enrich an RDF cube through a user-guided process that explores an *External Ontology* (EO) and discovers hierarchies from the LOD stored there, to connect them to the cube dimensions. This is done by recognizing in the EO the recurring modeling patterns that express roll-up relationships between RDF concepts and translating them into hierarchies, to

be stored locally within an *Internal Ontology* (IO). Overall, iMOLD takes in input a QB4OLAP-compliant RDF cube and an EO, and returns in output a QB4OLAP-compliant RDF cube whose dimensions have been extended with multidimensional hierarchies to enable more effective OLAP analyses.

From a functional point of view, the user locates a concept of interest in a selected EO (e.g., the concept of city on DBpedia), then she uses it as a starting point to build her hierarchies. The enriching scenario can be subdivided into two iterative phases. In the **acquisition** phase, if the concept of interest is not already present in the IO or it is not satisfactorily modeled (either because it is outdated or misaligned with the user's current requirements), the user can search for aggregation patterns in the EOs, build her own hierarchies by selecting the concepts of interest, provide appropriate names to levels, and add the results to the IO. The techniques we propose to address this phase are described in detail and experimentally evaluated in the remainder of the paper. In the **integration** phase, the levels of the newly-created hierarchies are populated with members (extracted from the EO through SPARQL queries) and linked to the RDF cube by identifying the inter-member mappings between the hierarchy levels and the cube dimensions. To identify these mappings we use a similarity function based on the Levenshtein distance between the member names; however, as various research communities have investigated this issue over the years, more sophisticated solutions could be found by checking the literature about *record linkage* [20], *entity identification* [21], *approximate joins* [22] and *ontology/schema matching* [23, 24]. Eventually, the matching couples identified are linked by means of the owl:sameAs property.

A key feature of LOD is that of creating connections between different ontologies. In an ontology, this connection is provided by pointing to objects of a different ontology with their original URIs. An example is the triple <dbpedia:Barcelona rdf:type yago:City108524735>, specified in DBPedia, which reuses a class defined in YAGO, therefore providing a link between the two ontologies. In iMOLD, the connectivity of LOD is exploited to enable users to jump from the currently explored EO to a different one whenever, by recognizing an aggregation pattern, a concept with a different namespace is reached. This transition can be seamlessly made by launching the next searches for patterns on the SPARQL endpoints of both ontologies and then merging the results; however, for the sake of simplicity, in this paper we will restrict to consider a single endpoint.

The remainder of this section is focused on the IO, which is the container of the multidimensional knowledge discovered by the users through the exploration of the EOs; specifically, for each hierarchy it models its levels, its roll-up relationships (with their multiplicity), and the mappings of these concepts into the EO. The structure of the IO relies on the existing vocabularies that already propose a solution in these contexts. However, the mere reuse of these vocabularies is not sufficient, as we need to extend the original vocabularies with custom classes and properties. To this end, we define two new namespaces, imold (`big.csr.unibo.it/imold#`) and imold-ex (`big.csr.unibo.it/imold-ex#`), to create additional classes and properties that are either domain-independent or

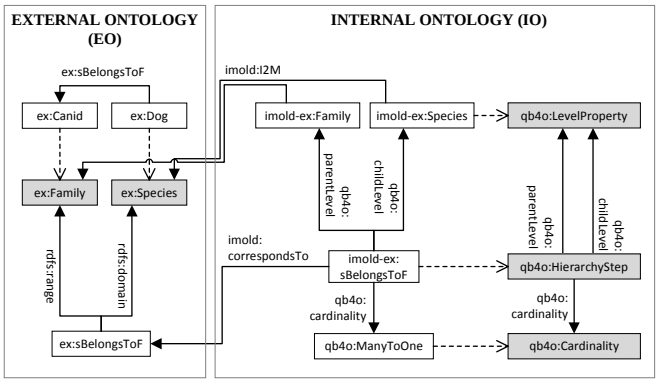

Figure 3: The Internal Ontology and an instance for the example in Figure 1 (a property at the class level is represented as an arc linking the domain class to the range class, which implies the correct definition of the property by means of the rdfs:domain and rdfs:range properties)

domain-dependent, respectively.

More specifically, as mentioned in Section 2.2, we reuse QB4OLAP to represent hierarchies. The mapping of levels and roll-up relationships into the EO (i.e., the identification of the original classes and properties from which these concepts are extracted) is made through four custom properties, imold:I2M, imold:L2M, imold:S2M and imold:correspondsTo, which materialize the link between the IO and the EO (see Figure 3). Noticeably, imold:I2M, imold:L2M, and imold:S2M are used to retrieve the members of each level when hierarchies are populated with data (further details on how these properties are used are given in Section 5). A complete glossary for the classes and properties used in the IO is included in Table 11 in the Appendix.

Example 2. *Figure 3 shows an instance of the IO and EO for our animal hierarchy. The* ex *namespace used here refers to the EO being explored. For instance, level* imold-ex:Species *in the IO is related to the corresponding class of the EO,* ex:Species*, via property* imold:I2M*, thus expressing the fact that the members of level* Species *are the instances of* ex:Species *(e.g.,* ex:Dog*). Similarly, the* imold-ex:sBelongsToF *roll-up relationship in the IO is related to the* ex:sBelongsToF *property in the EO.*

4. Aggregation Patterns in Ontologies

Our approach to discover hierarchies consists in recognizing aggregation patterns in RDF data. Table 1 summarizes the five patterns that can give rise to roll-up relationships; each cell shows how each hierarchy-related concept (as defined in Section 2) is mapped into an RDF construct. In particular, those five patterns imply the creation of three types of mappings between level members and RDF concepts:

Table 1: Hierarchy-related concepts and their representations within the different aggregation patterns

Concept	Patt. (A1)	Patt. (A2)	Patt. (A3)	Patt. (G1)	Patt. (G2)
Parent Level	Class	Class	Datatype/Class	Powertype	Powertype
Roll-up rel.	Assoc.	—	—	—	—
Child Level	Class	Class	Class/Datatype	Class	Powertype
Parent member	Instance	Instance	Literal/Instance	Class	Class
Part-of rel.	Assoc.	Assoc.	Assoc.	Instantiation	General.
Child member	Instance	Instance	Instance/Literal	Instance	Class

- *Instances-to-members* (I2M): the members of a level correspond to the instances of the RDF class(es) that correspond to that level; this is the case for both levels in (A1) and (A2), for one of the levels in (A3), and for the child level in (G1).
- *Literals-to-members* (L2M): the members of a level correspond to literals, whose datatype corresponds to that level; this is the case for one of the levels in (A3).
- *Subclasses-to-members* (S2M): the members of a level correspond to the subclasses of one or more classes; this is the case for the parent level in (G1) and for both levels in (G2).

As briefly mentioned in Section 3, these mappings are coded in RDF by connecting each level l in the IO to one or more classes or datatypes in the EO, which allows for retrieving the members of that level when hierarchies are populated with data. More specifically, (i) for I2M mappings, l is linked to the class(es) whose instances are members of l using property imold:I2M property (e.g., imold-ex:Family in Figure 3); (ii) for L2M mappings, l is linked to the datatype(s) whose literals are members of l using property imold:L2M property; (iii) for S2M mappings, l is linked to the class(es) whose subclasses are members of l using property imold:S2M.

In the following patterns are described in more detail, using Figure 4 as a general reference and the sample hierarchy depicted in Figure 1 as an example.

4.1. Association-Based Patterns

In these patterns, the roll-up relationships entailed by hierarchies are modeled as associations, which in RDF are represented using properties. Specifically, as exemplified in Figure 4, this can be done in three ways:

(A1) Pattern (A1) corresponds to two classes related by an RDF property (e.g., ex:Family and ex:Class related by ex:fBelongsToC in Figure 5.a). Two I2M mappings are here determined: the two classes are mapped into hierarchy levels connected by a roll-up relationship, and the class instances are mapped into level members.

(A2) Pattern (A2) allows the RDF property not to exist at the level of classes but only of instances (e.g., between ex:Animal and ex:Species in Figure 5.a).

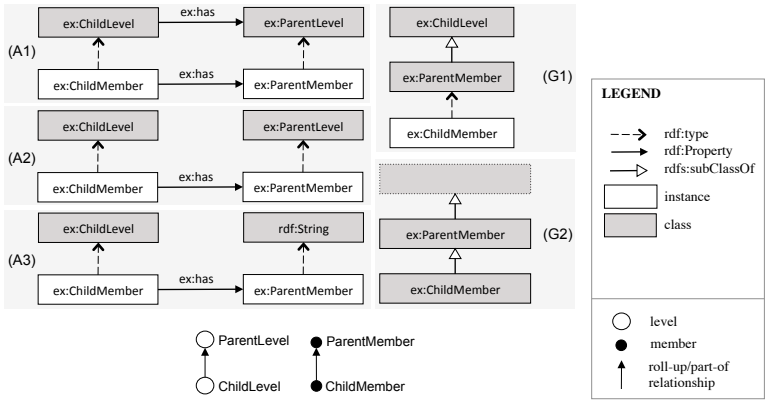

Figure 4: RDF aggregation patterns (top) and their multidimensional translation (bottom)

This pattern arises because of the incompleteness of LOD; indeed, in cross-domain ontologies, associations are rarely defined at the model level, i.e., through rdfs:domain and rdfs:range properties. As in (A1), two I2M mappings are determined.

(A3) Pattern (A3) occurs when there is no class modeling the parent (child) level but only a datatype, so its members correspond to literals rather than to instances (i.e., it has an L2M mapping instead of an I2M one). In this case, the name of the parent (child) level must be either provided by the user or derived from the name of the property.

We recall from Section 2.2 that roll-up relationships have many-to-one multiplicity. Since multiplicities are not explicitly represented in RDF, when an association-based pattern is recognized we have to sample the instances (e.g., by counting how many instances of ex:Class are related to each instance of ex:Family): if the average cardinality of the association between the domain and the range is close to one on the side of the range, then the pattern indeed corresponds to a roll-up relationship. Of course, this requires to assume that, though EOs may be incomplete and not fully correct, their data are statistically representative.

Finally, we observe that the association may have been modeled in one direction or the other. For instance, the association between ex:Family and ex:Class in Figure 5.a could also have been modeled with the RDF property ex:hasFamily, where ex:Class is the domain and ex:Family is the range, as depicted in Figure 6. For this reason, the child and parent roles can be inverted with regard to those shown in Figure 4. Even in this case, the roll-up relationships can be correctly derived by checking the average cardinality of the association as mentioned above to find that there is a one-to-many (rather than many-to-one) association between the domain and the range.

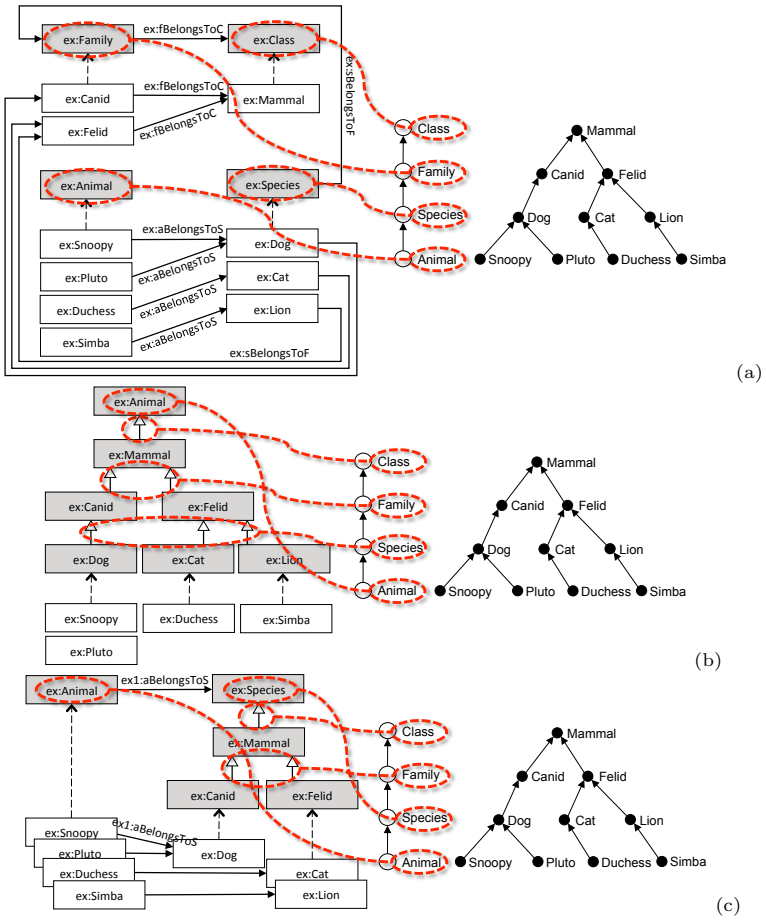

Figure 5: The hierarchy in Figure 1 modeled in RDF using associations (a), generalizations (b), and a mix of the two (c); in thick dashed lines, the correspondence between names of ontology concepts and names of hierarchy levels

4.2. Generalization-Based Patterns

The second group of patterns is based on generalization. Generalizations express an *is a* semantics that induces a subsumption between sets of instances of related classes —for instance, in Figure 5.b, ex:Felid generalizes ex:Cat and ex:Lion since the set of mammals instances of ex:Felid is superset of the set of mammals instances of ex:Cat and ex:Lion, and the same holds for ex:Canid and ex:Dog. But then, mammals can be grouped into felids and canids, or into cats, lions, and dogs, and the former grouping is coarser than the latter, which in OLAP terms translates to a part-of relationship between members Cat + Lion and Felid on the one hand, between Dog and Canid on the other. This suggests that there is roll-up relationship between two different levels, whose members correspond to classes. To find the names for these levels we recall that, from

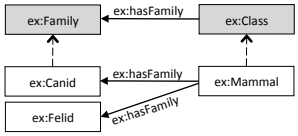

Figure 6: Alternative RDF representation for the association between ex:Family and ex:Class

a conceptual point of view, generalizations can be grouped depending on the criteria used, which in the UML terminology is called a powertype. A *powertype* is a metaclass[2] whose instances are subclasses of a given class; for instance, the specialization of class Person into subclasses Male and Female has powertype Gender, i.e., a metaclass with Male and Female as instances. In our example, the two powertypes involved are Species and Family, and give their names to the child and parent levels, respectively.

More specifically:

(G1) Pattern (G1) corresponds to two classes related by an rdfs:subClassOf property. Here the superclass and its instances are mapped into the child level and into its members, respectively, through an I2M mapping; the powertype and the subclasses are mapped into the parent level and into its members, respectively, through an S2M mapping. For instance, as depicted in Figure 5.b, class ex:Animal is transitively specialized into ex:Dog, ex:Cat, and ex:Lion based on powertype Species. Therefore, these three subclasses (i.e, subsets) give rise to three parent members of level Species, and their instances (ex:Snoopy, ex:Pluto, etc.) to child members of level Animal.

(G2) In pattern (G2) child members correspond to classes rather than to instances in the LOD. This may happen because of incompleteness or because of a different level of abstraction chosen by the ontology designer. In this case, the different classes corresponding to child members (e.g., ex:Canid and ex:Felid in Figure 5.b) would be generalized into a superclass (e.g., ex:Mammal) that would be the corresponding parent member, while both the parent and child levels (ex:Class and ex:Family, respectively) correspond to powertypes — i.e., two S2M mappings would be determined.

Powertypes are not made explicit in RDF. In principle, they could be explicitly represented at the metalevel, i.e., using metaclasses (e.g., class ex:Dog could be an instance of metaclass ex:Species). However, this type of metamodeling is extremely rare in LOD; for instance, at the time of writing, DBpedia has no metaclasses, while in YAGO only 5 classes are instances of (very generic) metaclasses (e.g., class rdfs:Property is an instance of metaclass rdfs:Resource). So, the user has to provide names for the levels corresponding to powertypes

[2] A *metaclass* is a class whose instances are classes.

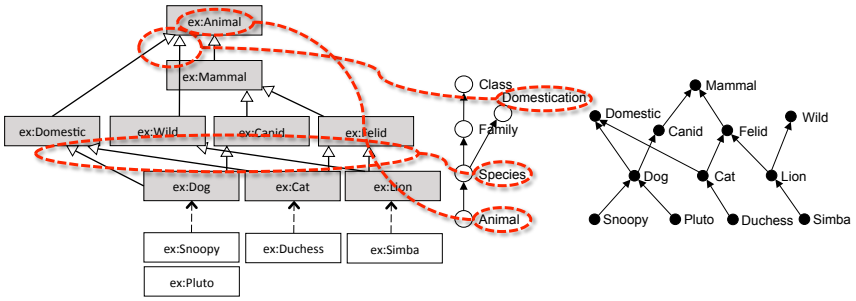

Figure 7: Double specialization for classes ex:Dog, ex:Cat, and ex:Lion

upon recognizing a generalization-based pattern. Remarkably, differently from association-based pattern, generalization-based ones do not require data to be queried, nor do they rely on probabilistic assumptions like those made for association multiplicities.

We finally note that, in case of an overlapping powertype (i.e., individuals are instances of more than one subclass and subsets are not disjoint), a many-to-many relationship arises; since in multidimensional modeling only many-to-one relationships are normally considered (assumption ♯2 in Section 4), we will not consider this case. Besides, there is also a possibility of having a multiple specialization (i.e., the same class can be a specialization of several superclasses), which can be interpreted as multiple many-to-one relationships with different semantics thus generating a branch in the hierarchy. For instance, Figure 7 shows how the generalization of ex:Dog, ex:Cat, and ex:Lion into ex:Domestic and ex:Wild leads to creating a branch towards level Domestication.

5. Acquisition

In this section, the core phase of iMOLD is described in detail; its goal is to discover hierarchies by recognizing aggregation patterns on an EO and translating them into hierarchies in the IO. We recall that, in the exploratory OLAP paradigm, each user may be interested in different portions of data (i.e., in different hierarchies). Thus, a full automation would be undesirable; iMOLD supports and speeds up the discovery process, but the user drives it by selecting relevant concepts at each step. Every recognition is based on a starting class c and on a direction dir:

- Class c is the entry point for pattern recognition, and is chosen by the user within the EO as a class of her interest for the current session. This class corresponds to a hierarchy level l, so it is mapped into the IO by creating an instance imold-ex:c of qb4o:LevelProperty; imold-ex:c is linked to c either via property imold:I2M (if its mapping is I2M, e.g., Species in Figure 3.b) or via property imold:S2M (if its mapping is S2M).

- Direction *dir* can be either *outbound* or *inbound*; given c and *dir*, we recognize the patterns by exploring the triples where c (or its instances) is either the subject (*dir* = 'outbound') or the object (*dir* = 'inbound'). Noticeably, any relationship a between subject s and object o is equally recognized and modeled in the IO either by starting from s and moving to o in the outbound direction, or by starting from o and moving to s in the inbound direction.

The recognition process is done in a breadth-first fashion, i.e., c is completely analyzed in its relationships with other classes or datatypes; the roll-up relationships selected by the user lead to new classes, from which the user can iteratively perform new searches. No recursive recognition is triggered to keep the system performance in line with the real-time requirement typical of OLAP applications; though it may happen that the user selects one exploration direction that leads her to miss some potentially interesting concept lying ahead in a different direction, iMOLD incorporates the possibility of "backtracking" along the exploration path to take some previously neglected paths, so there is no preclusion to a nearly-exhaustive exploration —it is actually up to the user. While Sections 5.1 and 5.2 show, respectively, how a single association-based or generalization-based pattern can be recognized, Section 5.3 explains how patterns can be repeatedly recognized and chained by describing an iteration of acquisition.

5.1. Recognition of Association-Based Patterns

The goal of these patterns is to determine whether a property p involving c can be mapped into a roll-up relationship, where the domain and range of p are mapped into a child and a parent level (or vice versa) in the hierarchy.

Definition 2 (Association). *An* association *is a triple $a = (d, p, r)$ where p is a property, d is a class that represents the domain of p, and r is either a class or a datatype that represents the range of p. Association a is characterized by its* right cardinality *$rightCard(a)$, i.e., the average number of distinct instances of r linked to each instance of d through p, and by its* left cardinality *$leftCard(a)$, i.e., the average number of distinct instances of d linked to each instance of r through p. Given $a = (d, p, r)$, we denote with a^{-1} its inverse, $a^{-1} = (r, p, d)$.*

An association a is a roll-up relationship if its multiplicity is either many-to-one or one-to-many; in particular, a corresponds to a roll-up relationship $u = a$ if its multiplicity is many-to-one, to a roll-up relationship $u = a^{-1}$ if its multiplicity is one-to-many. As already mentioned, since the RDFS vocabulary does not provide means to describe the multiplicity of a property, the only way to determine the multiplicity of a is through a statistical analysis at the instance level, which means inspecting the relationships in which the instances of d and r are involved. To decrease the complexity of the search, at the price of introducing some uncertainty, we adopt a (user-defined) parameter *maxCard* (default 1000) that acts as an upper bound to the number of instances of d (r)

Algorithm 1 Recognize Association-Based Patterns

Input pt: a pattern (either A1, A2, or A3), EO: an external ontology, IO: the internal ontology, c: a starting class, dir: a direction (either 'outbound' or 'inbound'), $maxCard$ and $multTol$: the search parameters
Output R: a set of roll-up relationships

1: **if** $card(c) \leq maxCard$ **then** ▷ Random offset for query q
2: $offset \leftarrow 0$
3: **else**
4: $offset \leftarrow Random(0, card(c) - maxCard)$
5: $R \leftarrow \emptyset$ ▷ Initialize R
6: $q \leftarrow Query(c, dir, pt, maxCard, offset)$ ▷ Create q...
7: $A \leftarrow Execute(EO, q)$ ▷ ...and execute it against EO
8: **for each** $a \in A$ **do** ▷ Find the roll-up relationships in A
9: **if** $rightCard(a) \leq multTol$ **then** ▷ If a is many-to-one...
10: $R \leftarrow R \cup \{a\}$ ▷ ...add it to R
11: **else if** $leftCard(a) \leq multTol$) **then** ▷ If a is one-to-many...
12: $R \leftarrow R \cup \{a^{-1}\}$ ▷ ...add its inverse to R
13: $UpdateIO(IO, R)$ ▷ Update IO with the roll-up relationships in R
14: **return** R

to be sampled. Besides, to cope with possible errors in the EO, we let the user specify a tolerance *multTol* (default 1.1) to determine whether an association can be considered as many-to-one or one-to-many.

The pseudocode for recognizing association-based patterns is shown in Algorithm 1. We start by randomly generating an *offset*, aimed at inducing some randomness in the selection of the sample from the instances of c (lines 1—4); to this end, we count the number of instances of c, $card(c)$, using a simple SPARQL query. Then, a SPARQL query q is generated by function *Query* (line 6); given a starting class c and an offset, q returns a set A of associations involving c in direction dir, together with the left and right cardinality of each association $a \in A$. The specific form of q depends on the pattern pt and on the search parameters; for instance, this is the query generated for pattern (A1) in the outbound direction (see Table 12 in the Appendix for an explanation of the variables):

```
SELECT ?p ?class (?nProp/?nO AS ?rightCard) (?nProp/?nS AS ?leftCard) ?nO ?nS
WHERE
{   SELECT ?p ?class (COUNT(*) AS ?nProp) (COUNT(DISTINCT(?o)) AS ?nO)
           (COUNT(DISTINCT(?s)) AS ?nS)
    WHERE
    {   ?p rdfs:domain ?c .                          ▷ Step 1: retrieve the properties of c
        ?p rdfs:range ?class .
        ?s a ?c .                                    ▷ Step 2: retrieve the property instances
        ?s ?p ?o .
        ?o a ?class
    }
    GROUP BY ?p ?class      ▷ Step 3: group the property instances to get the list of associations
}
```

The query for pattern (A2) does not work at the class level, and adds a check on the maximum number of instances of c to avoid overloading the endpoint:

```
SELECT ?p ?class (?nProp/?nO AS ?rightCard) (?nProp/?nS AS ?leftCard) ?nO ?nS
WHERE
{   SELECT ?p ?class (COUNT(*) AS ?nProp) (COUNT(DISTINCT(?o)) AS ?nO)
           (COUNT(DISTINCT(?s)) AS ?nS)
    WHERE
```

```
{   {   SELECT ?s                                   ▷ Step 1: select instances of c
        WHERE
        {   ?s a ?c .
        }
        LIMIT ?maxCard
        OFFSET ?offset
    } .
    ?s ?p ?o .                                      ▷ Step 2: retrieve the properties of each s
    ?o a ?class .                                   ▷ Step 3: retrieve the classification of each o
}
GROUP BY ?p ?class      ▷ Step 4: group the property instances to get the list of associations
}
```

When pattern (A3) is recognized (i.e., when literals are inspected), at Step 3 the class of ?o, ?class, is replaced by its datatype, str(datatype(?o)). Note that this pattern can only be applied in the outbound direction, because literals can only be objects in RDF triples.

Function *Execute* (line 7) submits q to the SPARQL endpoint of the EO. In the lines from 8 to 12, the associations in A are filtered according to their multiplicities and added to R. Threshold $multTol$ is applied to left and right cardinalities to determine if each association a can be considered as either many-to-one or one-to-many (lines 9 and 11). Then, all roll-up relationships in R are added to the IO using procedure $UpdateIO$ (line 13). The mapping of each $u = (l, p, l')$ into the IO is carried out as follows:

- l and l' are mapped into two instances of qb4o:LevelProperty; the members of l and l' correspond to the instances of the domain and range of p, so a connection with the corresponding classes in the EO is established via property imold:I2M (or imold:L2M in case of a literal);
- property p is mapped into an instance of qb4o:HierarchyStep, properly linked to the two corresponding qb4o:LevelPropertys, and also linked to its counterpart on the EO by means of the imold:correspondsTo property;

Example 3. *Consider again the portion of EO depicted in Figure 5.a, and let* ex:Species *be the starting class. If pattern (A1) is selected and the outbound direction is taken, a set A of associations is returned (line 7 of Algorithm 1) including $a =$* (ex:Species, ex:sBelongsToF, ex:Family). *Then the cardinalities of these associations are checked; let $rightCard(a)$ be lower than the tolerance $multTol$, so a is found to be a roll-up relationship and is added to the IO together with level* Species. *Now,* ex:Family *may be picked by the users to further extend the hierarchy, leading to discover the roll-up relationship to level* Class. *On the other hand, if the inbound direction were taken selecting pattern (A2), the association $a' =$* (ex:Animal, ex:aBelongsToS, ex:Species) *would be discovered and found to be a roll-up relationship as well.*

5.2. Recognition of Generalization-Based Patterns

Generalization-based patterns are cheaper to recognize than association-based ones because (i) no query at the instance level is required and (ii) the only inter-class link that must be considered is rdfs:subClassOf. On the other hand,

their interpretation is less intuitive, because the transformations applied to concepts move them along the instantiation-classification dimension. Whereas distinct classes always correspond to distinct levels in association-based patterns, in generalization-based ones distinct subclasses that belong to the same superclass can be grouped together to become members of a single level, which corresponds to the powertype of the subclasses. Furthermore, differently from association-based patterns, generalization-based ones always require additional information from the user to give names to powertypes.

Definition 3 (Generalization). *A* generalization *is an association* $g = (d, p, r)$ *where* d *and* r *are the subclass and the superclass, respectively, and* $p =$ rdfs:subClassOf. *We denote with* $PT(g)$ *the (user-provided) powertype to which* g *belongs.*

Consistently with our acquisition approach, the generalizations g involving a given class c are detected by navigating the rdfs:subClassOf properties according to direction dir: the superclasses of c are found by bounding d to c and taking $dir =$ 'outbound', while the subclasses of c are found by bounding r to c and taking $dir =$ 'inbound'. In both cases, an interaction with the user is necessary to filter out non-relevant generalizations; more specifically:

- A class c may be specialized according to different powertypes. Since powertypes are normally not modeled in RDF, when operating in the inbound direction the user must manually select the subclasses of c that belong to the powertype of interest and provide its name.
- Multiple specialization is allowed in RDF. Thus, when operating in the outbound direction, the user must manually select one or more superclasses of interest when searching for generalizations of c.

To create multi-level hierarchies, generalizations must be iteratively navigated. This can be done adopting either (i) a top-down strategy, where a general concept is selected first and the inbound direction is followed to iteratively find its subclasses; or (ii) a bottom-up strategy, where a detailed concept is selected first and the outbound direction is followed to iteratively find its superclasses; or (iii) a mix of these two. In any case, the identification of a powertype leads to creating a new level in the IO.

The pseudocode for recognizing generalization-based patterns —working for both the top-down and bottom-up strategies— is shown in Algorithm 2. The first operation (line 2) is the generation of a SPARQL query q that returns the set S of either the superclasses or the subclasses of c, according to direction dir. For instance, the query generated for $dir =$ 'inbound' is as follows:

```
SELECT DISTINCT ?type
WHERE
{   ▷ Retrieve the subclasses of c
    ?type rdfs:subClassOf ?c .
}
```

Algorithm 2 Recognize Generalization-Based Patterns

Input EO: an external ontology, IO: the internal ontology, c: a starting class, dir: a direction (either 'outbound' or 'inbound'), H: a hierarchy including a level l_c corresponding to c
Output R: a set of roll-up relationships

1: $R \leftarrow \emptyset$ ▷ Initialize R
2: $q \leftarrow Query(c, dir)$ ▷ Create q...
3: $S \leftarrow Execute(EO, q)$ ▷ ...and execute it against EO
4: **for each** $s \in S$ **do** ▷ Find the roll-up relationships corresponding to S
5: **if** dir ='inbound' **then** ▷ Moving top-down: s is a subclass of c
6: $g = (s, \text{rdfs:subClassOf}, c)$
7: **if** H only includes level l_c **then** ▷ First iteration
8: $R \leftarrow \{(l_c, \text{subClassOf}, PT(g))\}$
9: **else** ▷ Other iterations
10: $R \leftarrow \{(l_c, \text{subClassOf}, PT(g)), (PT(g), \text{subClassOf}, Par(l_c))\}$
11: **else** ▷ Moving bottom-up: s is a superclass of c
12: $g = (c, \text{rdfs:subClassOf}, s)$
13: $R \leftarrow \{(l_c, \text{subClassOf}, PT(s))\}$
14: $UpdateIO(IO, R, H, dir)$ ▷ Update IO with the roll-up relationships in R
15: **return** R

Note that the EO may possibly include transitive generalizations (e.g., class ex:Canid in Figure 5.b could be explicitly declared as a subclass of both ex:Mammal and ex:Animal). Though this is not shown above for simplicity, the actual query detects transitive rdf:subClassOf properties and excludes them.

Then, function *Execute* (line 3) submits q to the SPARQL endpoint of the EO. While mapping each class of S into a generalization g is trivial because it simply depends on the direction dir (lines 6 and 12), mapping each g into a roll-up relationship in R and then into the IO via procedure $UpdateIO$ is more complex, as the way the mapping is done also depends on the current state of the hierarchy, H. To explain how this is done, for simplicity we restrict to the case in which pattern (G1) is recognized, i.e., the members of the bottom level of the hierarchy to be built correspond to instances (the mapping process for pattern (G2) is similar); specifically, we refer to the portion of EO depicted in Figure 5.b. We start by describing the process for the top-down strategy (dir = 'inbound', lines 5—10), leaning on Figure 8.

0. Before the first iteration, the user has selected the starting class c (ex:Animal in Figure 8.a) which has been mapped by $UpdateIO$ into a level of the IO (i.e., an instance of qb4o:LevelProperty) as already described.

1. At the first iteration (line 7), pattern (G1) is recognized and the subclasses of c are found (e.g., ex:Mammal in Figure 8.b). For each powertype gs declared by the user, a new level l_{gs} (Class in the example) is created by $UpdateIO$ as an instance of qb4o:LevelProperty and the corresponding subclasses are mapped into the IO as members of l_{gs} via property imold:S2M. Moreover, the imold:I2M property of the level l_c corresponding to c is pushed down to the subclasses of c (in the example, from ex:Animal to ex:Mammal and its siblings), to denote that the members of l_c are currently represented by all the instances of these subclasses. Finally, a new roll-up relationship from l_c to l_{gs} is created as an instance of qb4o:HierarchyStep.

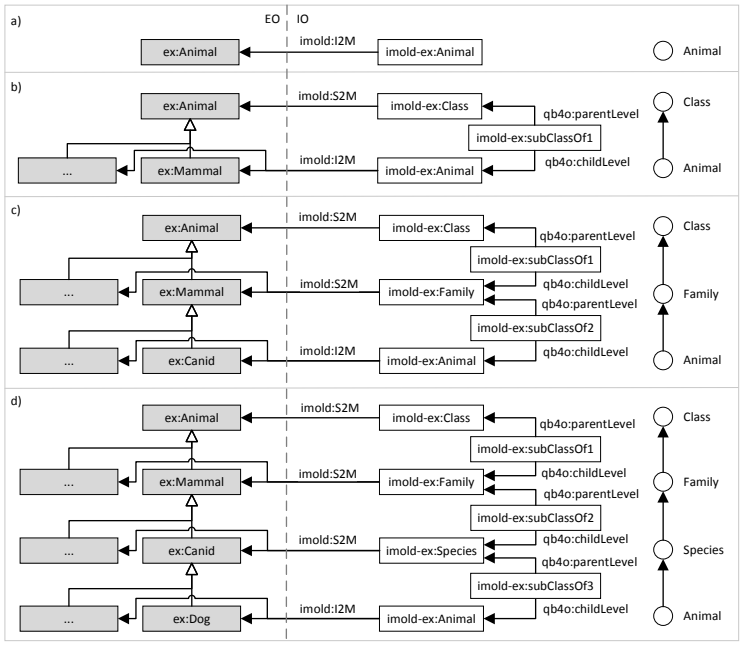

Figure 8: Top-down strategy for recognizing generalization-based patterns; the link between the instances in the IO and their classes (i.e., qb4o:LevelProperty and qb4o:HierarchyStep) is omitted for simplicity

2. At each following iteration (line 9), let $Par(l_c)$ be the parent of l_c in H. For instance, taking Figure 8.c, let now c correspond to ex:Canid, which implies that l_c and $Par(l_c)$ correspond to imold-ex:Animal and imold-ex:Family, respectively. Even in this case, after the subclasses of c have been found (e.g., ex:Dog), a new level l is created by $UpdateIO$ for each powertype declared by the user (e.g., Species). However, since the new level must be inserted in the hierarchy between l_c and $Par(l_c)$, the existing roll-up relationship from l_c to $Par(l_c)$ is replaced by two roll-up relationships, one from l_c to l_{gs} and one from l_{gs} to $Par(l_c)$.

Note that, when a top-down strategy is followed to discover a hierarchy H, the domain of H (meant as the domain of the finest level in H) progressively gets more selective as new levels are explored. For instance, in the example of Figure 8, from all animals it shrinks to all canids. So, if *all* animals are actually relevant to the user, she has to explore all the other subclasses (by explicitly navigating through reptiles, birds, etc.). This is because Algorithm 2 adopts a "lazy", class-wise approach, i.e., it maps into the IO (as members) only the classes that are explicitly selected by the user. Remarkably, the algorithm can be easily modified to implement a more "eager", level-wise approach where exploration is implicitly carried out by levels. For instance, at the iteration depicted in Figure 8.c, this means enabling the user to select the subclasses of

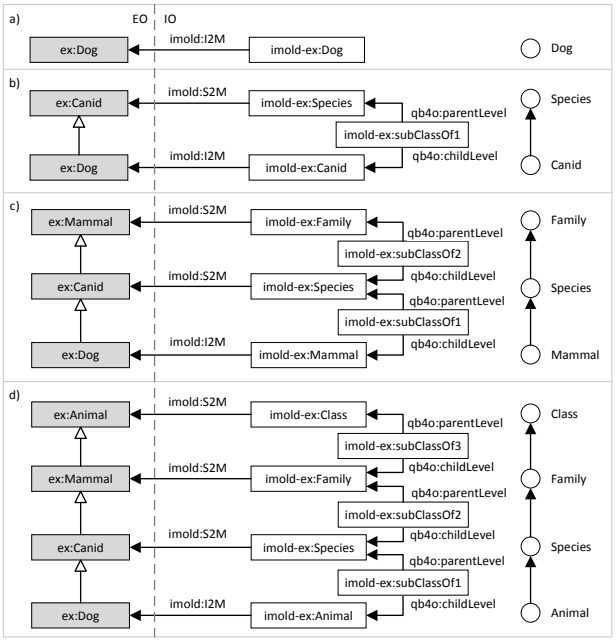

Figure 9: Bottom-up strategy for recognizing generalization-based patterns

interest not only from the subclasses of ex:Canid, but even from those of ex:Felid and all the other families.

To conclude this section, we briefly exemplify the process for the bottom-up strategy (dir = 'outbound', lines 11—13). Let c be the starting class, s one of its superclasses, and gs its powertype. Differently from the top-down case, here the hierarchy domain progressively enlarges as new levels are discovered, so when the roll-up relationship from l_c to l_{gs} is added, l_c must be renamed to s. Let for instance ex:Dog be the starting class c, initially represented in the IO with a level l_c named Dog as depicted in Figure 9. If ex:Canid is selected in the EO as a relevant superclass of ex:Dog, with powertype Species, a new level named Species is created, l_c is renamed to Canid, and a new roll-up relationship from Canid to Species is added.

5.3. Chaining Patterns

Association- and generalization-based patterns cover different but coexisting aspects of an EO. This means that, starting from any class in the EO, both kinds of patterns can be recognized to detect (and map to the IO) roll-up relationships. However, because of the different types of mappings they use between classes and levels (I2M, L2M, or S2M), a mix of association- and generalization-based patterns is feasible only under specific conditions. So let c be a class or datatype in the EO that has been mapped into a level of the IO during a previous acquisition iteration; Table 2 shows which patterns involving c can be

Table 2: Patterns that can be recognized depending on the mapping type of c

Mapping type	Patt. (A1)	Patt. (A2)	Patt. (A3)	Patt. (G1)	Patt. (G2)
L2M	✗	✗	✗	✗	✗
S2M	✗	✗	✗	inbound/drill-down	✓
I2M	✓	✓	outbound	roll-up	✗

Algorithm 3 Acquisition Iteration

Input pt: a pattern (either A1, A2, or A3), EO: an external ontology, IO: the internal ontology, c: a starting class, dir: a direction (either 'outbound' or 'inbound'); H: a hierarchy including a level l_c corresponding to c; $maxCard$ and $multTol$: the search parameters
Output R: a set of roll-up relationships

1: $R \leftarrow \emptyset$ ▷ Initialize R
2: **if** $mappingType(c)$ ='S2M' **then**
3: **if** (pt ='G1' **and** dir ='inbound') **or** (pt ='G2') **then**
4: $R \leftarrow RecognizeGeneralizationBasedPatterns(EO, IO, c, dir, H)$
5: **else if** $mappingType(c)$ ='I2M' **then**
6: **if** (pt ='A1') **or** (pt ='A2') **or** (pt ='A3' **and** dir ='outbound') **then**
7: $R \leftarrow RecognizeAssociationBasedPatterns(pt, EO, IO, c, dir, maxCard, multTol)$
8: **else if** pt ='G1' **then**
9: $R \leftarrow RecognizeGeneralizationBasedPatterns(EO, IO, c, dir, H)$
10: **if** l_c is in an association-based hierarchy **and** dir ='inbound' **then**
11: $PushDownAssociation(c, R)$
12: **return** R

further recognized depending on the mapping type previously used for c. The rationale is that the same level in the IO can only have one mapping to the EO. Thus, based on the kinds of mappings each pattern generates at the parent and child levels, the following situations can arise:

- L2M mappings do not enable any further pattern to be recognized. This is because literals are not identified by URIs, but they are simple values used for descriptive purposes (e.g., the label of an instance, or the age of a person) and their semantics depends on the property that references them.
- S2M mappings are used by generalization patterns only, thus they are incompatible with association-based ones. Also, notice that (G1) can be recognized only in the inbound direction to detect a finer level of aggregation (i.e., a drill-down); this is because (G1) requires the S2M level (c) to be the parent level, while the mapping of the child level has type I2M.
- I2M mappings are the only ones that enable the further recognition of patterns of both kinds. In particular, all association-based patterns can be recognized (with the exception of (A3) in the inbound direction for the aforementioned reasons), while (G1) is enabled only to detect a coarser level. Again, this is because (G1) requires the I2M level to be the child level, while the mapping of the parent level is S2M.

A procedural view of a single iteration of acquisition is provided in Algorithm 3. As already said, the patterns enabled depend on the mapping type of c (lines 2 and 5); pattern recognition that in Table 2 are not allowed return an empty set. As S2M mappings are used by generalization-based patterns only, further

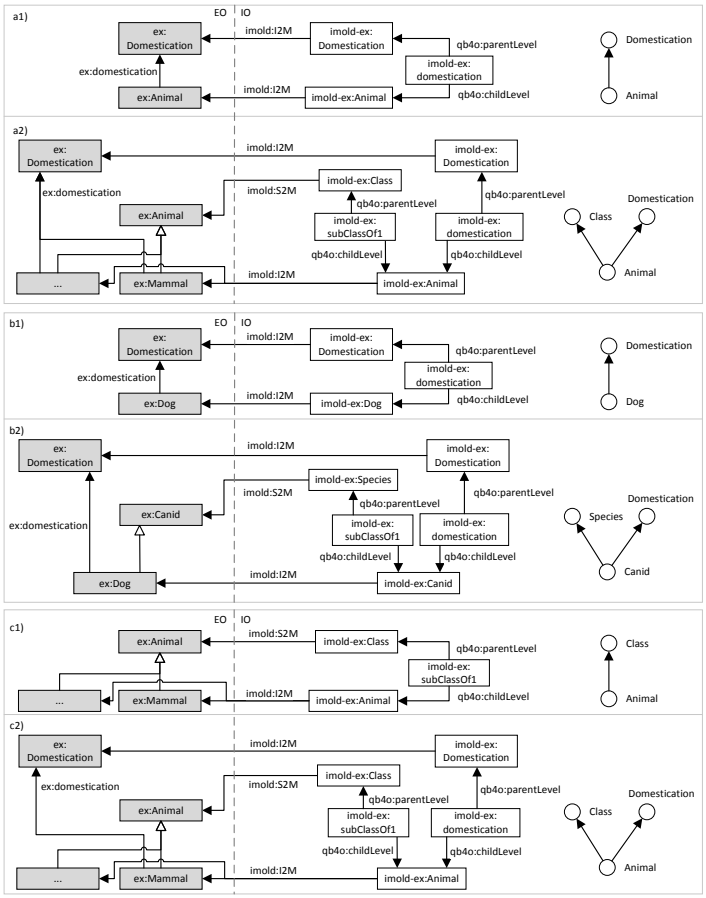

Figure 10: Examples of combined recognition of association- and generalization-based patterns

recognitions simply trigger Algorithm 2. The real mixing takes place starting from I2M mappings (lines 6—11). The execution of Algorithms 1 and 2 in the allowed situations creates no particular issues (lines 7 and 9), while some further note is useful to explain what happens when the hierarchy has been detected by recognizing association-based patterns and is extended with (G1) in the 'inbound' direction (line 10). As explained in Section 5.2 with reference to the top-down strategy, pattern (G1) applies a transformation on l_c, as its members become the instances of the subclasses of c instead of the instances of c (i.e., the I2M mapping from c is pushed down to its subclasses). Thus, for the final hierarchy to be consistent, the associations involving c that represent roll-up relationships must also be pushed down to its subclasses; this is done by function $PushDownAssociation$ (line 11). In practice, for the sake of simplicity we assume that if $a = (d, p, r)$ is a roll-up relationships, then $a' = (d', p, r)$ is still a roll-up relationship where d' is a subclass of d (the same for r). To ensure

the correctness of this operation, however, the detection of association-based patterns should be repeated on the subclasses of d (or r) to verify that the roll-up relationships still hold on at least one of them.

Example 4. *Figure 10 shows an example for the three possible scenarios of pattern chaining. The first scenario is the most complex one, in which the hierarchy has been detected through association-based patterns (Figure 10.a1) and is then extended by recognizing (G1) in the 'inbound' direction (Figure 10.a2) from one of the classes —in this case from* ex:Animal. *Consistently with the top-down strategy (Figures 8.a and 8.b), a roll-up relationship from* Animal *to* Class *is added. Also, notice that the* ex:domestication *association is not considered anymore from* ex:Animal *to* ex:Domestication *but from the subclasses of* ex:Animal *to* ex:Domestication. *In the second scenario, the hierarchy has been detected through association-based patterns (Figure 10.b1) and is extended by recognizing (G1) in the 'outbound' direction from* ex:Dog *(Figure 10.b2). In the third scenario, the existing hierarchy has been detected through generalization-based patterns (Figure 10.c1) and is extended by recognizing (A2) in the 'outbound' direction from* ex:Mammal *(Figure 10.c2).*

5.4. User Experience

Implementing the acquisition phase of iMOLD poses one more important challenge, that is, how to provide a clean and simple user interface that hides to some extent the complexity of the pattern recognition process to let the user focus on the discovery of hierarchies. We preliminarily note that the variety, volume, and veracity aspects of public ontologies make a *complete* automation of the acquisition process unfeasible, due to the inherent difficulties in reliably inferring the quality and relevance of the data explored and to the huge size of the exploration space. Additionally, there is a large degree of subjectivity in judging both the quality and the relevance of ontological data, so these evaluations may change from user to user, and some concepts (e.g., powertypes) must be explicitly provided by the user. As a result, we claim that the data exploration process must be necessarily supervised by the user to ensure the effectiveness of acquisition.

Acquisition builds on two basic operations:

- **Select a concept of interest**: with this operation, the user locates a starting class in a selected EO to enable the subsequent recognition of patterns. In practice, the user simply has to provide a search string and the ontology to query; then the system queries the SPARQL endpoint and retrieves a list of matching classes. The matching classes are shown to the user, ranked by decreasing cardinalities (i.e., number of instances). As the user selects one starting class, her selection is stored in the IO as a hierarchy level.

- **Recognize a pattern**: this operation aims at discovering a hierarchy involving the starting class c. Once the user has selected one specific pattern

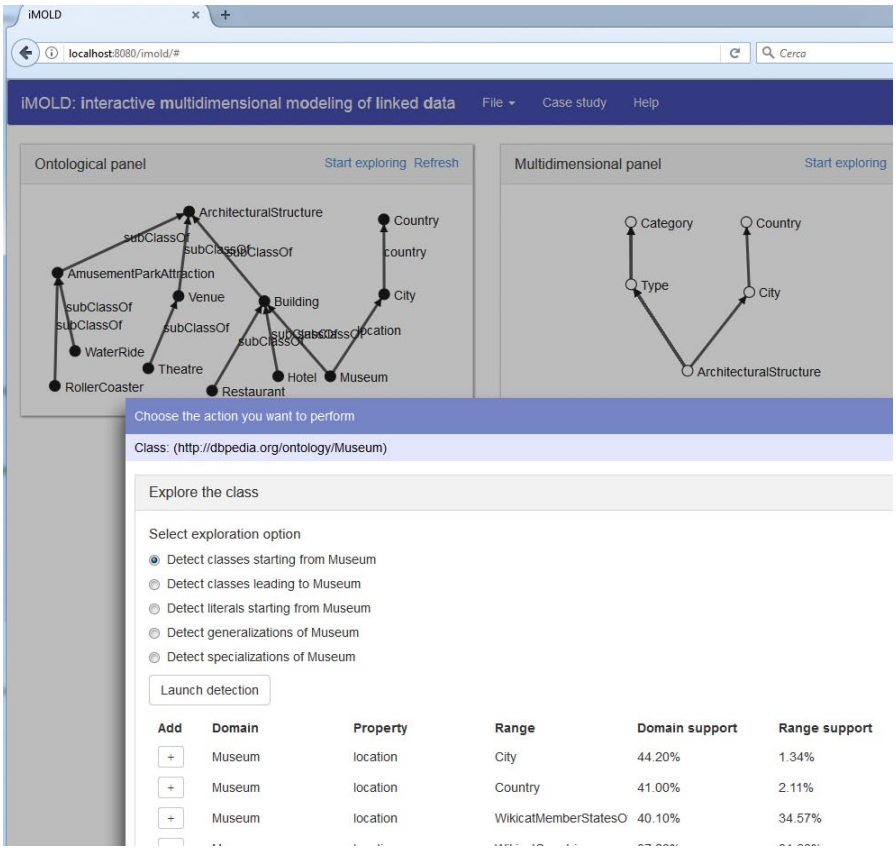

Figure 11: The user interface of iMOLD: in the background, the ontological (left) and the multidimensional (rigth) panel; in the foreground, the wizard for ontology-driven pattern recognition

to be recognized, the SPARQL endpoint of the EO is queried to find new roll-up relationship and update the IO accordingly. The system shows the list of candidate relationships, ranked by their support.[3] Then the user selects one relationship u, so that she can iteratively trigger pattern recognition using the range of u as the starting class.

Note that, in principle, aggregation patterns could be made transparent to users by applying in sequence the recognition algorithms for all five patterns and in both directions. While this is surely feasible from a technical point of view, we claim that the resulting user experience would be made ineffective and unsatisfactory by the huge number of resulting candidate relationships.

[3]Let u be a roll-up relationship and c be the level of u corresponding to the starting class; the support $supp(u, c)$ of u in c is the percentage of instances of c that are involved in u.

To increase the effectiveness and the inclusiveness of the user experience, iMOLD proposes two alternative (and freely interchangeable) interaction approaches, both supporting the basic operations described above but tailored on users with different background and expertise. Both approaches are supported by the combined view shown in the background window of Figure 11, which includes two panels: an *ontological panel* where the user has a picture of the classes and properties of the EO (represented with black dots and labeled arcs, respectively), and a *multidimensional panel* where the user sees how classes and properties translate to a hierarchy using a graphical notation inspired by the Dimensional Fact Model [25] (levels and roll-up relationships are represented with white dots and directed arcs, respectively).

- The *ontology-driven experience* is oriented to users who have good familiarity with ontologies and semantic web. Here the focus is set on the EO, a low-level view of the IO is provided, and a more intense user interaction is required. The user has a strict control over the acquisition phase; she can precisely specify the aggregation patterns to recognize and finely tune the search parameters (i.e., $maxCard$ and $multTol$) to refine the search based on her preferences. By clicking on a class in the ontological panel, a wizard for pattern recognition is launched (see the foreground window in Figure 11).
- The *OLAP-driven experience* is targeted to users who have good familiarity with the field of data warehousing and multidimensional modeling. Here the focus is set on the hierarchy being built, a high-level view of the IO is provided, and a lower degree of interaction is required. In this case pattern recognition and search parameters are transparent to the user, who simply interacts by iteratively selecting, given a level l of interest on the multidimensional panel, one or more multidimensionally-inspired operations:
  - the search for a parent level of l, which triggers the recognition of association-based patterns and that of generalization-based ones with top-down strategy;
  - the search for a child level of l, which triggers the recognition of association-based patterns;
  - the extension of the domain of l, which triggers the recognition of generalization-based patterns: with bottom-up strategy first, to find the parent levels of l (i.e., the superclasses of the class corresponding to l), and then with top-down strategy to find new members for l and its parent levels (i.e., the subclasses of the superclasses previously found).

6. Case Studies and Evaluation

To demonstrate the potential of our approach, in this section we provide an extensive discussion comprising case studies and evaluations in terms of effec-

tiveness and efficiency. Section 6.1 presents a first case study in which a fictional user discovers a hierarchy, one step at a time, by exploring a small portion of DBpedia. Section 6.2 presents a second case study aimed at demonstrating an end-to-end solution, where a dimension of a real-world RDF cube is extended with new levels which are then used to formulate OLAP queries on the extended cube. In Section 6.3 we propose a more quantitative evaluation by discussing the results of some tests made with real users. Then we discuss the performance of iMOLD in Section 6.4, and finally we compare it with a related approach in the literature in Section 6.5.

To evaluate iMOLD we chose to focus on the so-called *cross-domain* ontologies, as their chaotic status makes it harder for both humans and machines to efficiently extract structured and coherent knowledge. While the generic theme of this kind of ontologies opens to a broad set of examples in different domains, some degree of uncertainty and incorrectness appears in the data. Indeed, differently from small and confined ontologies that are typically developed in-house and provide a reliable representation of the area of interest, big and cross-domain ontologies are more inclined to errors and imprecisions, especially if they are built through a collaborative effort across the web. In particular, we focus the case study on DBpedia, one of the best-known public ontologies available on the web, which is also compliant with the LOD principles and covers a wide range of domains.

The prototype we built (see Figure 11) is implemented as a web application (available at `semantic.csr.unibo.it/imold`) and fully supports the user experience described in Section 5.4; also, a guide to replicate the case studies is available. The back-end functionalities are implemented in Java; in particular, we extensively rely on the Jena Library, as it provides solid APIs for the communication with remote SPARQL endpoints and for in-memory manipulation of a local ontology. As to the IO, we currently store it within a simple RDF file; however, the migration to a triplestore is planned in future evolutions of the prototype. We used Javascript to implement the user interface and adopted the D3 library for the graphical visualization of the IO.

6.1. Case Study: Recognizing Patterns

We consider a social BI scenario in which a fictional user has collected reviews about museums in an RDF cube and, to be better analyze them, is interested in discovering a hierarchy rooted in museums. The goal is to aggregate the set of available observations (i.e., the mentions of a museum within the text of reviews) to obtain aggregate data (e.g., a measure of popularity as the total number of mentions, or a measure of appreciation as the average sentiment expressed by the reviewers). The most obvious candidate as a starting concept is class dbo:Museum, which currently hosts 5341 instances. A few simple queries on dbo:Museum show that the distinct properties that make use of its instances as either domain or range in the triple are 120 and 131, respectively, for a total of 344687 instances. Although not impressive, these numbers prove that a manual approach to identify the properties that map into roll-up relationships would be at least wearying, if not unfeasible.

Table 3: Detecting the parents of Museum

p (property/semantics)	r (range/parent)	$supp(a, d)$	$supp(a, r)$
dbo:location	dbo:City	40.9%	1.4%
dbo:location	dbo:Country	40.6%	4.8%
dbo:location	yago:MemberStatesOfTheUnitedNations	32.6%	36.6%
dbo:location	dbo:Settlement	32.5%	36.6%
dbo:location	yago:MemberStatesOfNATO	18.6%	85.2%
dbo:location	yago:CountriesBorderingTheAtlanticOcean	17.3%	38.6%
dbp:location	yago:English-speakingCountriesAndTerritories	17.0%	21.8%
dbp:visitors	xsd:integer	14.9%	—
dbp:type	yago:WikicatArtMuseumsAndGalleries	11.6%	5.88%
dbp:type	yago:WikicatArtMuseumsAndGalleries	10.0%	5.88%

To begin the exploration of the EO following the ontology-driven experience, we assume that the user clicks on the dbo:Museum class on the ontological panel and activates the recognition of association-based patterns in the outbound direction to find the parents of level Museum. The results of the recognition of association-based patterns are shown in Table 3, where each row corresponds to a many-to-one association $a = (d, p, r)$ with domain $d =$ dbo:Museum and range r, i.e., to a roll-up relationship with semantics p from Museum to r; $supp(a, d)$ is the percentage of instances of dbo:Museum for which a is present, while $supp(a, r)$ is the percentage of instances of r for which a is present.

Among the roll-up relationships where r is a class (i.e., those discovered with patterns (A1) and (A2)), the most relevant property is dbo:location, which provides the geographical position of each museum. Interestingly, this property links to instances that belong to different classes (see rows 1–7 of Table 3); this is due to multiple classification, which is widely used in DBpedia. The iMOLD interface helps the user in this case by orienting her towards the classes with most references (i.e., higher support). As to the associations where r is a literal (i.e., those discovered with pattern (A3)), property dbp:visitors (which indicates the number of visitors of the museums) could be interesting for analysis. For the sake of the example, we assume that the user selects (dbo:Museum, dbo:location, dbo:City). The following recognition of association-based patterns from dbo:City leads to finding new associations, among which the most relevant is (dbo:City, dbo:location, dbo:Country). Assuming that this one is selected, a 3-level linear hierarchy is eventually built: Museum, City and Country.

Back to our social BI scenario, we now assume that the user wants to broaden its scope from museums only to other places of interests that are subject to reviews, such as hotels and restaurants. In the context of the OLAP-driven experience, she can click on the Museum level of the multidimensional panel and activate the operation to extend the domain of that level. As a consequence, the generalizations $g = (d, p, r)$ with domain $d =$ dbo:Museum and property $p =$ rdfs:subClassOf are explored to find the superclasses r of dbo:Museum, then for each r its other subclasses d' are explored to find candidate new members for level Museum. The results are shown in Table 4. Class dbo:Museum has only one superclass, dbo:Building. The user can now select the siblings that

Table 4: Extending the domain of level Museum

r (superclass/new name of level)	d' (sibling class/member for new level)	$card(d')$
dbo:Building	dbo:Castle	1389
	dbo:HistoricBuilding	8413
	dbo:Hospital	3019
	dbo:Hotel	1285
	dbo:Library	1005
	dbo:Prison	952
	dbo:ReligiousBuilding	4349
	dbo:Restaurant	1158
	dbo:ShoppingMall	2597
	dbo:Skyscraper	4

relate the most to her area of interest (e.g., dbo:Hotel and dbo:Restaurant) and provide the name of the powertype (in this case, Type could be an option). As a result, the hierarchy is changed by (i) renaming level Museum into Building, whose members are now the union of the instances of dbo:Museum, dbo:Hotel, and dbo:Restaurant; and (ii) adding level Type as a parent of Building, with members Museum, Hotel, and Restaurant.

Table 5: Extending the domain of level Type

r (superclass/new name of level)	d' (sibling class/member for new level)	$card(d')$
dbo:ArchitecturalStructure	dbo:AmusementParkAttraction	499
	dbo:Infrastructure	87966
	dbo:MilitaryStructure	4371
	dbo:SportFacility	9120
	dbo:Tower	1868
	dbo:Tunnel	82
	dbo:Venue	5157

The domain extension operation can be iteratively activated to explore a larger part of the EO. For instance, extending level Type leads to searching for the siblings of dbo:Building; the results are shown in Table 5. If dbo:AmusementParkAttraction and dbo:Venue are selected as relevant concepts and the new powertype is named Category, the hierarchy is further extended by (i) renaming Building into ArchitecturalStructure and adding new members (i.e., the instances of dbo:AmusementParkAttraction and dbo:Venue); and (ii) adding level Category as a parent of Type, with members Building, AmusementParkAttraction, and Venue. The final shape of the hierarchy is the one shown in Figure 11.

6.2. Case Study: End-to-End Solution

While in the previous section we have shown how patterns can be recognized in an EO to build a hierarchy, here we demonstrate how iMOLD can be used within an end-to-end solution to effectively enrich a real-world RDF cube, and how the resulting enriched cube can then be queried.

We rely on the open data published by the World Bank financial institution (`http://www.worldbank.org`), which supports developing countries through loans for strategic projects. The World Bank Linked Data (WBLD, `http:`

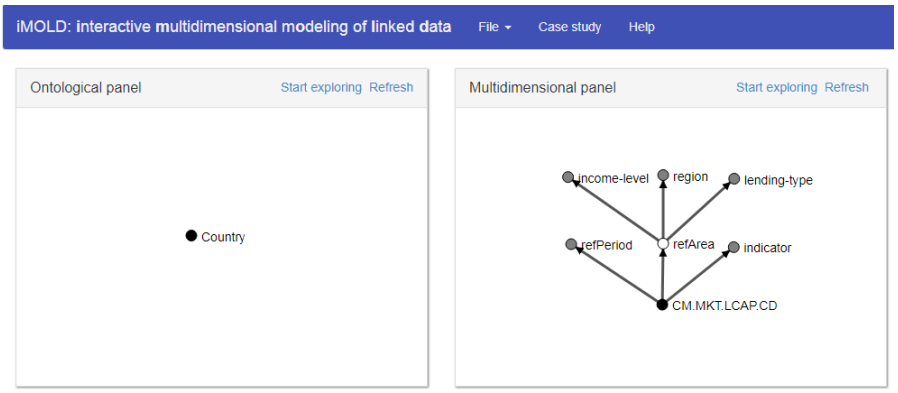

Figure 12: The WBLD cube loaded in iMOLD; on the right, the black circle represents the set of observations, while the grey ones are levels not linked to the EO

`//worldbank.270a.info`) translates some of these open data to RDF using the QB vocabulary. In particular, we use the CM.MKT.LCAP.CD cube, which provides the market capitalization in US dollars per country and year (`https://data.worldbank.org/indicator/CM.MKT.LCAP.CD`), also linking each country to its counterpart on DBpedia. A previous work shows how this cube can be translated from QB to QB4OLAP using the QB2OLAPem tool [8]. An excerpt of the cube is shown in the Appendix.

Figure 12 shows the WBLD cube loaded in iMOLD. In the multidimensional panel, the black dot represents the set of observations (i.e., market capitalizations), which are identified by three dimensions: refPeriod (i.e., years), refArea (i.e., countries) and indicator (a dimension with a single member, "CM.MKT.LCAP.CD", useful when multiple cubes representing different indicators are joined together); the refArea can be rolled up to either income-level, region, or lending-type, whose members are codes defined by the World Bank. Since refArea is the only level for which a mapping to DBpedia is provided, every other level is colored in grey; indeed, the ontological panel displays only the Country class, which is the one referenced by refArea.

From here, the user can start enriching the cube by searching patterns on the country level. For instance, the recognition of association-based patterns from dbo:Country in the outbound direction finds a roll-up relationship with dbo:Currency, which can be added as a new level in the IO.

Now the IO can be exploited to formulate queries on the enriched cube. To this end, in [26] the authors show how OLAP queries can be formulated in SPARQL by relying on the QB4OLAP vocabulary. In our case, however, we must keep in mind that the IO extends the cube only from the intensional point view (i.e., level Currency is created, but its members are not imported from the EO). Thus, two solutions are possibile: (i) download members from the IO and add them to the cube; and (ii) formulate a *federated* SPARQL query, which allows to simultaneously query the local store and a remote endpoint. While

Table 6: Samples of average market capitalizations by Currency and refPeriod obtained with a federated SPARQL query on the enriched WBLD cube; the "gov-uk" namespace abbreviates http://reference.data.gov.uk/id/year/.

Currency (EO)	refPeriod (IO)	Avg. market cap.
dbr:Euro	gov-uk:2012	3.58E+11
dbr:Euro	gov-uk:2011	3.14E+11
dbr:Euro	gov-uk:2010	3.71E+11
...	...	...
dbr:Pound_sterling	gov-uk:2012	3.02E+12
dbr:Pound_sterling	gov-uk:2011	2.90E+12
dbr:Pound_sterling	gov-uk:2010	3.11E+12
...	...	...
dbr:United_States_dollar	gov-uk:2012	1.38E+11
dbr:United_States_dollar	gov-uk:2011	1.26E+11
dbr:United_States_dollar	gov-uk:2010	1.38E+11
...	...	...

the first solution yields the best results in terms of performance and consistency (in that the results are not susceptible to the EO being updated over time), it introduces scalability issues and an implementation effort. So here we opt for the second solution, which allows to query the enriched cube on-the-fly. For instance, a query that calculates the average market capitalization by Currency and refPeriod can be formulated as follows:

```
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX qb4o: <http://purl.org/qb4olap/cubes#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT ?currencyEO ?yearIO (AVG(?marketCap) as ?ag1)
    WHERE {
        ?o a qb:Observation .                    ▷ Step 1: retrieve observations and local level members
        ?o qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> ;
            <http://purl.org/linked-data/sdmx/2009/measure#obsValue> ?marketCap ;
            <http://purl.org/linked-data/sdmx/2009/dimension#refArea> ?yearIO ;
            <http://purl.org/linked-data/sdmx/2009/dimension#refPeriod> ?countryIO .
        ?countryIO owl:sameAs ?countryEO .              ▷ Step 2: retrieve remote level members
        SERVICE <http://dbpedia.org/sparql> {
            ?countryEO dbo:currency ?currencyEO .
            ?currencyEO a dbo:Currency
        } .
    }
GROUP BY ?currencyEO ?yearIO
```

where the SERVICE clause extends the results obtained in Step 1 by navigating RDF concepts on the remote endpoint. The execution time of this query is 10 seconds, and is mainly due to the navigation from the RDF concepts in the cube to the ones on the SPARQL endpoint of DBpedia[4]. A sample of the query results is shown in Table 6.

[4]The performance of federated queries is highly dependent on the amount of data that *jumps* between endpoints. The query shown presents an intuitive formulation, but it can be optimized by reversing Step 1 and 2, thus moving the SERVICE before the retrieval of observations. The execution time already refers to the optimized version of the query.

Table 7: Average results of user test for the three tasks (times are expressed in minutes)

Figure	SPARQL-based task	Ontology-driven task	OLAP-driven task
Total time	71.6	26.7	21.8
- Exploration time	-	19.6	14.7
- System time	-	3.0	3.3
- Browsing time	-	4.1	3.8
Num. interactions	11	27	25
Score (1..5)	2.3	3.4	3.1

6.3. User Evaluation

To give a quantitative assessment of the benefits of our approach and compare the effectiveness of the ontology-driven and OLAP-driven scenarios, we conducted a set of tests with a group of 21 users, mainly PhD and master students with basic or advanced knowledge of ontologies and multidimensional modeling. The goal of the tests was to evaluate the execution of the same task (i.e., discover a hierarchy starting from a concept selected from the DBpedia ontology) according to the two interaction scenarios of iMOLD.

To prepare the tests, three concepts in different application domains were chosen (namely banks, museums, and screenwriters), and for each concept a task was stated using either the multidimensional or the ontological terminology (e.g., *extend the domain of level* Museum *by discovering a multi-level hierarchy to include also hotels and other types of architectural constructions, then extend the hierarchy to aggregate these constructions according to the currencies of the countries they are located in*). Each user was asked to read an instruction sheet explaining the iMOLD goals, how to operate the interface in both scenarios, and the five aggregation patterns; then, (s)he was asked to execute two tasks, one for each interaction scenario, on two randomly-chosen domains. To determine a testing baseline, we also asked to execute an additional task using plain SPARQL to the users who had some knowledge of this language; in this case, the users were relieved from the construction of the hierarchies in the IO and only had to manually draw the inferred hierarchy.

Table 7 shows the average results obtained for each task. By monitoring the user activities on the prototype, we split the time taken to complete each task (*total time*) into three blocks: *exploration time* (i.e., the time taken by the user to check the status of the hierarchy and choose the next operator to be applied), *system time* (i.e., the time taken by the system to recognize one or more patterns and provide the list of candidate relationships), and *browsing time* (i.e., the time taken by the user to browse these relationships and select one or more of them). The table also shows the average *number of interactions* made by each user, referring to the number of SPARQL queries, of pattern recognitions, and of multidimensional operations, respectively for SPARQL-based, ontology-driven, and OLAP-driven tasks. Finally, the hierarchies discovered by each user have been manually evaluated for correctness and completeness and scored on a scale from 1 (worst result) to 5 (best result); row *score* shows the average results.

In SPARQL-based tasks, users have mostly failed in achieving an acceptable result despite spending a considerable amount of time. The main difficulties

Table 8: Average system and decision time for each pattern

Pattern	Direction	System time	Browsing time
(A1)	inbound	2	5
(A1)	outbound	2	8
(A2)	inbound	15	18
(A2)	outbound	7	38
(A3)	outbound	3	26
(G12)	inbound	2	13
(G12)	outbound	2	7

encountered are the design of the queries to recognize association-based patterns (an activity that took away almost half of the total time) and the interpretation of generalization-based patterns: although most users built a correct hierarchy with two levels, no one was able to build a correct hierarchy with three levels. This is due to the fact that mere query results obviously do not support users in finding the correct interpretation key.

Conversely, the iMOLD prototype enabled users to adequately complete the exercise in a short time. Indeed, manually writing the queries requires a lot more time than simply launching the queries automatically written by iMOLD; so, in the same time iMOLD users employed to acquire a comprehensive picture of the ontology, SPARQL users could only catch a very partial glimpse of it. Users spent less time on OLAP-driven tasks than on ontology-driven ones (22 minutes against 27), showing that the OLAP-driven experience succeeds in reducing the amount of work for the user. Interestingly, however, most users declared to feel more comfortable with the ontology-driven experience, especially when combining patterns was necessary: in fact, the low-level view adopted for the former experience reduces the potential misinterpretations of the multidimensional translation of the patterns, thus enabling users to better identify the correct relationships. This is reflected in the results, where the average score for ontology-driven tasks is slightly better than that of the OLAP-driven ones (3.4 against 3.1); one more clue in this direction comes from the fact that the best results in the OLAP-driven task came from the users who had already carried out the ontology-driven one (3.3 against 2.8).

6.4. Performance Evaluation

Table 8 shows the average system and browsing times for each single pattern, as measured during the user tests (with the parameters for association-based patterns set to their default values). Patterns (G1) and (G2) are merged into (G12) since (i) the query to detect generalizations is the same for both patterns, and (ii) from the user's perspective, the difference between the two is transparent.

Noticeably, there is a clear difference between patterns (A1) and (A2), with the former taking considerably less time than the latter. The reason is that pattern (A1) accesses a considerably lower number of properties of the starting class (only those for which both rdfs:domain and rdfs:range are specified, i.e., about 4% of the properties in DBpedia); even the browsing time of pattern

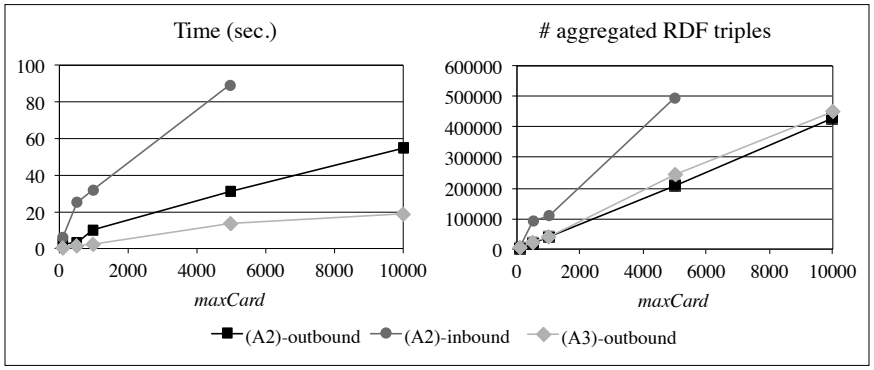

Figure 13: System time and number of RDF triples aggregated for association-based patterns, with reference to class dbo:City

(A1) is lower, due to the lower number of results. Overall, the slower pattern to be executed is (A2) in the inbound direction. This is expected, because many-to-one relationships in EOs tend to be expressed from subjects to objects rather than vice versa (i.e., one-to-many relationships as the one in Figure 6 are less used); as a result, the number of triples to be considered by the SPARQL query in the inbound direction is typically higher. On the other hand, pattern (A2) in the outbound direction turns out to be the one with the highest browsing time. This is also expected, as this pattern is the one that statistically returns the highest number of candidate relationships.

Among the parameters for association-based patterns, the only one that impacts on the system time is *maxCard*, which limits the number of instances of the starting class to be accessed (whereas *multTol* is used to filter out the obtained associations). Figure 13 shows, for different values of *maxCard* and for each pattern (except pattern (A1), which is statistically not significant), the system times and the number of RDF triples aggregated by the SPARQL queries with reference to class dbo:City, whose 21k instances are used in about 5 millions of triples). First of all, we observe that the results are consistent with the previous observation about pattern (A2) being the most expensive one in the inbound direction; the performance for $maxCard = 10000$ is not shown because the SPARQL endpoint is unable to process the huge amount of RDF triples involving the instances of dbo:City as objects. In general, the time required to execute the patterns is proportional to the number of RDF triples aggregated by the respective queries. For this reason, the sampling technique proves to be quite useful —if not necessary— to obtain the results in a reasonable time. The obvious drawback, however, is that of obtaining a partial view of the associations available from the starting class, especially if the value of *maxCard* is set too low. Nonetheless, we remark that the performance of patterns mostly depends on the computational power of the SPARQL endpoint; therefore, a good strategy for the user would be to first get an overview from a small sample of the data and eventually request more complete results if this overview is deemed interesting.

Table 9: Comparison of NB16 [27] and iMOLD on DBpedia

Approach	# roll-up relationships	% agreement
NB16	4148	31%
iMOLD	19484	7%

6.5. Comparative Evaluation

To the best of our knowledge, as discussed in Section 7, most other approaches in the literature cannot be directly compared with ours since they either operate on different data formats (e.g., XML), or address different phases in the exploratory OLAP process (i.e., publish and query). The only exception is the approach in [27], which we will call NB16, where aggregation hierarchies are discovered in LOD using a statistical model. In this section we compare NB16 with iMOLD using DBpedia as a test dataset. To carry out a fair comparison, we operate the following restrictions:

#1 We only compare the two approaches in terms of the results they produce. Indeed, since NB16 is completely automated and in [27] it is evaluated on a local dump of the whole EO, while iMOLD requires some user interaction and relies on queries executed against a remote SPARQL endpoint, a comparison in terms of efficiency would be pointless.

#2 We restrict iMOLD to search for the aggregation patterns that are also searched by NB16, i.e., (A1), (A2), and (A3) in outbound direction.

#3 We exclude from the results of iMOLD the roll-up relationships involving concepts that are not part of the core dump of DBpedia (e.g., `dbpedia.org/class/yago/*` and `www.wikidata.org/entity/*`), because these are not considered by NB16.

Note that, as a relevant consequence of restriction #2, in this test we do not count the roll-up relationships deriving from the application of generalization-based patterns (e.g., the one from ArchitecturalStructure to Type in Figure 11), because they cannot be discovered by NB16.

Our experimental setup consisted in launching NB16 and iMOLD on the same set of 432 starting classes (with all search parameters in iMOLD set to their defaults). The results are summarized in Table 9, which shows the number of roll-up relationships discovered by the two approaches and the percentage agreement (i.e., how many of the relationships discovered by one approach have been discovered by the other as well). From a merely quantitative point of view, it emerges that iMOLD discovers more relationships than NB16. Though this could be interpreted as a limit of the statistical model, it ultimately depends on the different goals of the two approaches. Indeed, NB16 aims at returning a set of RDF cubes (each comprising dimensions, measures, and hierarchies), so the discovery of hierarchies is just a part of a (fully-automated) process; conversely, iMOLD is specifically focused on the discovery of hierarchies (because RDF cubes are previously created during the publish stage of exploratory OLAP).

Table 10: Some sample relationships found by NB16 and iMOLD with dbo:Agent as starting class; the last two columns indicate from which approach(es) each relationship has been discovered

p (prop./semantics)	r (range/parent)	$supp(a, d)$	$leftCard(a)$	$rightCard(a)$	NB16	iMOLD
dbo:nationality	dbo:Country	16.6%	253.32	1.04	yes	yes
dbo:genre	dbo:MusicGenre	3.8%	188.66	2.28	yes	no
dbo:careerStation	dbo:CareerStation	7.2%	1.00	7.18	yes	no
dbo:successor	dbo:Governor	0.1%	1.33	1.08	yes	no
dbo:currentMember	dbo:Agent	9.5%	3.31	1.35	yes	no
dbo:careerStation	dbo:TimePeriod	7.6%	7.18	1.00	no	yes

For the sake of completeness we point out that, if restriction #3 is removed, iMOLD retrieves around 317000 associations.

Table 9 shows that more than two thirds of the relationships discovered by NB16 are not discovered by iMOLD. In the following we discuss the reasons for this, with the support of Table 10 which lists some examples of relationships detected by the two approaches using dbo:Agent as a starting class.

- In 18% of cases, the reason why a relationship is discovered by NB16 and not by iMOLD is that it has a very low support, so it is missed by our sampling algorithm. For instance, this is the case for the relationship between dbo:Agent and dbo:Governor.

- In 1% of cases (for instance, the relationship between dbo:Agent and dbo:Agent), the relationship creates a cycle in the hierarchy —which is intentionally excluded in iMOLD since cyclic hierarchies are seldom used in multidimensional modeling.

- In 50% of cases, the missing relationships have been discarded by iMOLD because they have been (correctly) labeled as either one-to-one or many-to-many. In fact, to estimate the multiplicity of an association a from d to r NB16 heuristically considers the ratio between the cardinality of d in a and the cardinality of r in a; if this ratio is greater than 1, NB16 labels the association as many-to-one. Of course this heuristics may fail: for instance, Table 10 shows that each agent is related to an average of more than two music genres, so clearly the association is many-to-many; nevertheless, NB16 labels it as many-to-one because the cardinality of agents (67434) is higher than the one of genres (815). NB16 also extends this heuristics to three-level hierarchies, which are created whenever a chain of two associations $a = (d, p, r)$ and $a' = (d', p', r')$ is found where $r = d'$ and the cardinality of d in a is higher than that of r' in $a + a'$. This introduces additional errors, because either a or a' might have one-to-many or many-to-many multiplicity. For instance, the association from dbo:Agent to dbo:CareerStation is one-to-many, but NB16 includes it in a three-level hierarchy dbo:Agent/dbo:CareerStation/dbo:NumberOfGoals.

To conclude this comparison, we remark that the effectiveness of iMOLD is confirmed by the fact that, despite restrictions #2 and #3, 93% of the roll-

up relationships it discovers are not discovered by NB16 (e.g., the one from dbo:Agent to dbo:TimePeriod).

7. Related Work

There have been several efforts towards automating the discovery of multidimensional schemata from available data. This approach is known as *supply-driven design* and consists of exploring the data sources in order to identify potential aggregation patterns that would allow to arrange data in a multidimensional fashion. [18] provides a comprehensive discussion on different techniques used to identify dimension hierarchies from available data. In all cases, discovering FDs is the cornerstone to automatically build hierarchies. Typically, most approaches look for FDs at the schema level, since instance-based approaches are computationally expensive for real scenarios [28].

Besides traditional approaches assuming the existence of a conceptual representation of the domain (e.g., an E/R or UML class diagram) or a well-formed logical relational database schema, some efforts have focused on less structured data models such as XML or logics-based formalisms. As mentioned, all of them focus on FD discovery at the schema level. The most relevant works related to ours are [29, 30]. [29] identifies FDs from XML schemata represented as a graph. The graphical representation of the XML schema facilitates finding the FDs, which is examined in the direction expressed by the arcs and according to cardinalities included in the dependency graph. Cardinalities are either provided or inferred from key attributes. Where no cardinality information can be inferred they shift to an instance-based approach by querying schema-compliant XML documents. [30] redefines the concept of FD for logics-based formalisms under open-world assumption. The paper presents new inference algorithms to identify FDs and, based on them, aggregation hierarchies. However, this approach works at the schema level (i.e., instances are accessed).

Now that a huge amount of data is available in the LOD cloud, providing techniques for its effective exploration is becoming more and more important. In this area researches have focused on providing intuitive and effective techniques for visualizing [31] and navigating [32] LOD, on delivering a high level and conceptual view of large LOD clouds [33], and on integrating and learning information from them [34]. In this line, several efforts focused on deploying multidimensional schemata on LOD to facilitate its exploration and visualization, according to the cube metaphor, have recently emerged (e.g., [35, 36, 37, 38]). However, only [27] presents an approach to automate the discovery of dimension hierarchies on LOD. Similarly to our approach, the authors aim at automatically discovering multidimensional conceptual patterns (i.e., resembling a multidimensional star schema) that summarize LOD based on probabilistic graphical models. They propose the use of the statistics about the instance data to generate the multidimensional schemata and therefore to identify hierarchies.

Following the visionary ideas behind concepts such as *small analytics for big data* [3], *fusion cubes* [2], *drill-beyond* [4], or the *global cube* [5], OLAP and multidimensional analyses are highlighted as a perfect match for assisting

and supporting the user when exploring the Web of Data. However, there is still a lack of research to automate the discovery of multidimensional schemata and enable automatic data exploration/crossing in the LOD setting. As a first step in this direction, a conceptual framework to perform exploratory OLAP over LOD has been proposed in [39]; the idea is to derive the multidimensional schemata from different data sources in order to run OLAP queries on the respective SPARQL endpoints. A key point recognized by the authors is that, due to the large volume and complexity of public knowledge bases, a user-guided process for multidimensional schema detection must be implemented. A similar problem occurs in [8], where, hierarchies are identified by computing FDs from instances. To avoid the inherent high computational complexity of identifying FDs from instances [28], the authors start the search from QB data sets where factual data and dimensional data (without hierarchies) are explicitly available. We believe iMOLD to be the first significative advance towards this goal. As already mentioned, iMOLD follows an instance-based approach but it neglects the problems identified in [28] by avoiding a pure supply-driven approach and incorporating the user to lead the process (as typically done in demand-driven approaches). As a consequence, iMOLD can be regarded as a *mixed* approach based on aggregation patterns. Unlike [27], we do not follow a probabilistic approach but one based on data modeling patterns (as typical of software engineering approaches) extracted from instances. Furthermore, by involving the user to guide the search we reduce the computational complexity of sampling instances, unlike [39, 8].

Also the areas of *ontology matching*, which aims at defining mappings among schema or ontology elements that are semantically related [40], and that of *instance matching*, which aims at determining if two resources are the same real-world entity [41], are related to our work. In particular, [42] presents a classification of the main techniques used for ontology matching: (i) terminological techniques, based on string matching, e.g., based on Levenstein or edit distance, (ii) structural techniques, which exploit the neighborhood of the concepts to determine their similarity and (iii) semantic techniques, which benefit from deduction and inference rules [43]. Although the latest yield good results, they are not that popular due to their computation complexity. Overall, these areas aim at interlinking concepts and instances, respectively, from two different ontologies in order to align them. Our approach follows a broader approach known as *ontology construction* where matching, mapping and merging operations are required [14]. More precisely, we benefit from the well-defined multidimensional constructs and semantics to construct ontological hierarchies from where to analyze already available factual data. The result of our process is the construction of a semantically correct hierarchy, whereas ontology/instance matching can only guarantee the correctness of the matching at hand (and not that of the whole hierarchy). Thus, the hierarchies we build fulfill the three necessary summarization conditions established by the state-of-the-art literature [16]. Conversely, other ontology construction approaches do not focus on multidimensional hierarchies (e.g., [15]) and rather follow linguistic [14] or statistical [44] approaches. In this sense, one may consider our approach as a domain-

specific ontology construction problem (i.e., finding implicit multidimensional hierarchies) that, to our knowledge, has not been tackled before.

Finally, as to data quality, several proposals have been made to address the issues of FD violations (e.g., [45]) and missing values (e.g., [46]), which are recurrent when data are extracted and integrated from the Web [47]. With LOD gaining popularity over the last decade, many approaches have focused on the assessment and management of data quality issues in the world of semantic web. [48] is a comprehensive survey that discusses the quality measures proposed by more than 20 papers over a period of ten years; our approach already deals with measures of accuracy and completeness, but it could be extended to consider additional qualities of the explored data. An interesting work in this area is SWIQA [49], a framework to assess a series of quality rules in LOD, including FDs and missing values; data quality requirements are expressed as SPARQL query templates based on SPIN (`www.w3.org/Submission/spin-sparql/`), a W3C standard to represent SPARQL queries as RDF triples. Another inspiring web-based tool is ProLOD [50], which profiles a LOD source and provides useful insights about the available contents and schemata. Both tools, however, carry out their analyses on a local snapshot of the data, therefore are not directly applicable to query SPARQL endpoints.

8. Conclusion

In this paper we have presented iMOLD, an approach to discover aggregation hierarchies at exploration time to integrate the RDF cubes with situational data. We assume these situational data come in the form of LOD in RDF format, whose lack of concrete schema presents important challenges. Specifically, we have identified five different aggregation patterns that go beyond the classical FD approach for multidimensional design, and also consider the possibility of defining hierarchies based on taxonomies and combinations of both.

Though iMOLD is a significant step towards integrating LOD into RDF cubes, some relevant aspects still need to be considered. First of all, there is a possibility that some hidden FDs are present in LOD; for instance, with reference to our working example, each animal may be associated to its species and family, but an explicit connection between species and families may be missing in the source ontology. In this case, using algorithms for instance-based discovery of approximate FDs (e.g., TANE [51]) would enable hierarchies to be more accurately reconstructed. One more issue is related to automatic recognition and management of cycles in source data, which can give rise either to shared hierarchies (e.g., a building is located in a country and was designed by an architect who was born in a country, but the two countries may be different) or to convergences (e.g., a museum is located in a country and is managed by an institution of the same country). Finally, while the description of the iMOLD approach we gave is user-independent, to effectively support knowledge reuse and collaboration in multi-user and multi-session environments the IO should be extended to include also the metadata that relate each portion of hierarchy with the users who are currently using it for their session. In this way, while

Table 11: Glossary for the classes and properties in the Internal Ontology

URI	Definition
qb4o:LevelProperty	Defines a generic level (e.g., Day)
qb4o:HierarchyStep	Defines a roll-up relationship between two instances of qb4o:LevelProperty
imold:I2M	(rdfs:domain qb4o:LevelProperty; rdfs:range rdfs:Class; rdfs:subPropertyOf rdfs:seeAlso) Links an instance of qb4o:LevelProperty to one or more classes in the EO; it states that the members of the level are the instances of the linked classes.
imold:L2M	(rdfs:domain qb4o:LevelProperty; rdfs:range rdfs:Datatype; rdfs:subPropertyOf rdfs:seeAlso) Links an instance of qb4o:LevelProperty to a datatype in the EO; it states that the members of the level are the instances of the LOD-type.
imold:S2M	(rdfs:domain qb4o:LevelProperty; rdfs:range rdfs:Class; rdfs:subPropertyOf rdfs:seeAlso) Links an instance of qb4o:LevelProperty to one or more classes in the EO; it states that the members of the level are the subclasses of the linked classes
imold:correspondsTo	(rdfs:domain qb4o:HierarchyStep; rdfs:range rdf:Property; rdfs:subPropertyOf rdfs:seeAlso) Links a qb4o:HierarchyStep to the corresponding property in the EO; it is also used to link a qb4o:LevelMember to the corresponding owl:Thing (either a class or an instance) in the EO

at each acquisition iteration the IO is updated with all the roll-up relationships discovered for reuse purposes, during each session a user could create some sort of "local" hierarchies either by selecting some levels and roll-up relationships that were previously discovered, or by actively exploring some new areas of the EOs.

Appendix

The glossary for the IO and the variables used in the SPARQL queries are shown in Tables 11 and 12, respectively. In the following, an excerpt of the WBLD cube used in the case study of Section 6.2 is shown in Turtle syntax.

```
# PREFIXES #
@prefix qb4o: <http://purl.org/qb4olap/cubes#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix qb: <http://purl.org/linked-data/cube#>.
@prefix ex: <http://example.com/2017/10/03/09/40/11/cube#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix xmlns: <http://www.w3.org/2001/XMLSchema#>.
@prefix imold: <http://big.csr.unibo.it/imold#>.
@prefix dbo: <http://dbpedia.org/ontology/>.

# DATASET #
<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet .

# CUBE #
<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> qb:structure ex:structure .
ex:structure a qb:DataStructureDefinition ;
    qb:component [
        qb:measure <http://purl.org/linked-data/sdmx/2009/measure#obsValue>;
```

Table 12: Variables used in the SPARQL queries for detecting patterns

Name	Type	Pattern	Value
?c	input	ass., gen.	URI of c
?maxCard	input	ass.	parameter $maxCard$
?offset	input	ass.	query offset
?p	output	ass.	a property linked to c
?class	output	ass.	a class to which the instances o belong
?rightCard	output	ass.	right cardinality of a
?leftCard	output	ass.	left cardinality of a
?nO	output	ass.	range cardinality, i.e., number of distinct instances of $class$ involved in p
?nS	output	ass.	domain cardinality, i.e., number of distinct instances of c involved in p
?type	output	gen.	the superclass of c
?s	auxiliary	ass.	an instance of c
?o	auxiliary	ass.	an instance linked to s through p
?nProp	auxiliary	ass.	number of properties p connecting s to o

```
        qb4o:aggregateFunction qb4o:Sum ] ;
    qb:component [
        qb4o:level <http://worldbank.270a.info/property/indicator>;
        qb4o:cardinality qb4o:ManyToOne ];
    qb:component [
        qb4o:level <http://purl.org/linked-data/sdmx/2009/dimension#refArea>;
        qb4o:cardinality qb4o:ManyToOne ];
    qb:component [
        qb4o:level <http://purl.org/linked-data/sdmx/2009/dimension#refPeriod>;
        qb4o:cardinality qb4o:ManyToOne ].

# DIMENSIONS (excerpt) #
ex:dimension2 a qb:DimensionProperty;
    qb4o:hasHierarchy ex:hierarchy5.
ex:hierarchy5 a qb4o:Hierarchy;
    qb4o:hasLevel <http://purl.org/linked-data/sdmx/2009/dimension#refArea>;
    qb4o:hasLevel <http://worldbank.270a.info/property/region>;
_:hs3 a qb4o:HierarchyStep;
    qb4o:inHierarchy ex:hierarchy5;
    qb4o:childLevel <http://purl.org/linked-data/sdmx/2009/dimension#refArea>;
    qb4o:parentLevel <http://worldbank.270a.info/property/region>;
    qb4o:pcCardinality qb4o:ManyToOne.

# Dimension levels (excerpt) #
<http://purl.org/linked-data/sdmx/2009/dimension#refArea> a qb4o:LevelProperty.
<http://worldbank.270a.info/property/region> a qb4o:LevelProperty;
    imold:I2M dbo:Country.

# MEASURES (excerpt) #
<http://purl.org/linked-data/sdmx/2009/measure#obsValue> a qb:MeasureProperty.

# LEVEL MEMBERS (excerpt) #
<http://worldbank.270a.info/classification/country/IT> a qb4o:LevelMember;
    qb4o:memberOf <http://purl.org/linked-data/sdmx/2009/dimension#refArea>;
    skos:broader <http://worldbank.270a.info/classification/income-level/OEC>;
    skos:broader <http://worldbank.270a.info/classification/lending-type/LNX>;
    skos:broader <http://worldbank.270a.info/classification/region/ECS>;
    owl:sameAs <http://dbpedia.org/resource/Italy>.

<http://worldbank.270a.info/classification/region/ECS> a qb4o:LevelMember;
    qb4o:memberOf <http://worldbank.270a.info/property/region>.

# OBSERVATIONS (excerpt) #
<http://worldbank.270a.info/dataset/world-bank-indicators/CM.MKT.LCAP.CD/IT/2012>
```

```
a qb:Observation;
qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD>;
<http://purl.org/linked-data/sdmx/2009/measure#obsValue> 4804526464008mlns:decimal;
<http://worldbank.270a.info/classification/indicator/>
    <http://worldbank.270a.info/classification/indicator/CM.MKT.LCAP.CD>;
<http://purl.org/linked-data/sdmx/2009/dimension#refArea>
    <http://worldbank.270a.info/classification/country/IT>;
<http://purl.org/linked-data/sdmx/2009/dimension#refPeriod>
    <http://reference.data.gov.uk/id/year/2012>.
```

References

[1] V. Markl, Situational business intelligence, in: Proc. BIRTE, 2008.

[2] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J. Mazón, F. Naumann, T. B. Pedersen, S. Rizzi, J. Trujillo, P. Vassiliadis, G. Vossen, Fusion cubes: Towards self-service business intelligence, IJDWM 9 (2013) 66–88.

[3] M. Stonebraker, What does 'big data' mean?, http://cacm.acm.org/blogs, 2012.

[4] J. Eberius, M. Thiele, K. Braunschweig, W. Lehner, Drillbeyond: Enabling business analysts to explore the web of open data, PVLDB 5 (2012) 1978–1981.

[5] B. Kämpgen, S. Stadtmüller, A. Harth, Querying the global cube: Integration of multidimensional datasets from the web, in: Proc. EKAW, 2014, pp. 250–265.

[6] A. Abelló, O. Romero, T. B. Pedersen, R. B. Llavori, V. Nebot, M. J. A. Cabo, A. Simitsis, Using semantic web technologies for exploratory OLAP: a survey, IEEE Trans. Knowl. Data Eng. 27 (2015) 571–588.

[7] C. Bizer, T. Heath, T. Berners-Lee, Linked data - the story so far, Int. J. Semantic Web Inf. Syst. 5 (2009) 1–22.

[8] J. Varga, A. Vaisman, O. Romero, L. Etcheverry, T. Pedersen, C. Thomsen, Dimensional enrichment of statistical linked open data, Web Semantics: Science, Services and Agents on the World Wide Web 40 (2016).

[9] L. Etcheverry, A. Vaisman, QB4OLAP: A vocabulary for OLAP cubes on the semantic web, in: Proc. COLD, Boston, USA, 2012.

[10] R. P. D. Nath, K. Hose, T. B. Pedersen, Towards a programmable semantic extract-transform-load framework for semantic data warehouses, in: Proc. DOLAP, Melbourne, Australia, 2015, pp. 15–24.

[11] T. Komamizu, T. Amagasa, H. Kitagawa, H-SPOOL: A SPARQL-based ETL framework for OLAP over linked data with dimension hierarchy extraction, IJWIS 12 (2016) 359–378.

[12] L. Etcheverry, A. A. Vaisman, Querying semantic web data cubes, in: Proc. Alberto Mendelzon Int. Workshop on Foundations of Data Management, CEUR-WS.org, Panama City, Panama, 2016.

[13] B. Kämpgen, A. Harth, No size fits all - running the star schema benchmark with SPARQL and RDF aggregate views, in: Proc. ESWC, Montpellier, France, 2013, pp. 290–304.

[14] N. Choi, I. Song, H. Han, A survey on ontology mapping, SIGMOD Record 35 (2006) 34–41.

[15] J. David, F. Guillet, H. Briand, Matching directories and OWL ontologies with AROMA, in: Proc. CIKM, Arlington, USA, 2006, pp. 830–831.

[16] J. Mazón, J. Lechtenbörger, J. Trujillo, A survey on summarizability issues in multidimensional modeling, Data Knowl. Eng. 68 (2009) 1452–1469.

[17] C. Bizer, T. Heath, T. Berners-Lee, Linked data - the story so far, Int. J. Semantic Web Inf. Syst. 5 (2009) 1–22.

[18] O. Romero, A. Abelló, A survey of multidimensional modeling methodologies, IJDWM 5 (2009) 1–23.

[19] M. Golfarelli, D. Maio, S. Rizzi, Conceptual design of data warehouses from E/R schema, in: Proc. HICSS, Hawaii, USA, 1998, pp. 334–343.

[20] N. Koudas, S. Sarawagi, D. Srivastava, Record linkage: similarity measures and algorithms, in: Proc. SIGMOD, Chicago, Illinois, 2006, pp. 802–803.

[21] E. Lim, J. Srivastava, S. Prabhakar, J. Richardson, Entity identification in database integration, Inf. Sci. 89 (1996) 1–38.

[22] A. Das, J. Gehrke, M. Riedewald, Approximate join processing over data streams, in: Proc. SIGMOD, San Diego, California, 2003, pp. 40–51.

[23] J. Euzenat, P. Shvaiko, Ontology Matching, Second Edition, Springer, 2013.

[24] P. A. Bernstein, J. Madhavan, E. Rahm, Generic schema matching, ten years later, PVLDB 4 (2011) 695–701.

[25] M. Golfarelli, S. Rizzi, Data Warehouse design: Modern principles and methodologies, McGraw-Hill, 2009.

[26] M. Bouza, B. Elliot, L. Etcheverry, A. A. Vaisman, Publishing and querying government multidimensional data using QB4OLAP, in: Proc. LA-WEB, Minas Gerais, Brazil, 2014, pp. 82–90.

[27] V. Nebot, R. B. Llavori, Statistically-driven generation of multidimensional analytical schemas from linked data, Knowl.-Based Syst. 110 (2016) 15–29.

[28] M. R. Jensen, T. Holmgren, T. B. Pedersen, Discovering multidimensional structure in relational data, in: Proc. DaWaK, 2004, pp. 138–148.

[29] B. Vrdoljak, M. Banek, S. Rizzi, Designing web warehouses from XML schemas, in: Proc. DaWaK, 2003, pp. 89–98.

[30] O. Romero, D. Calvanese, A. Abelló, M. Rodriguez-Muro, Discovering functional dependencies for multidimensional design, in: Proc. DOLAP, ACM, 2009, pp. 1–8.

[31] C. Hirsch, J. Hosking, J. Grundy, Interactive visualization tools for exploring the semantic graph of large knowledge spaces, in: Proc. VISSW, volume 443, 2009.

[32] R. Mirizzi, A. Ragone, T. Di Noia, E. Di Sciascio, Semantic wonder cloud: exploratory search in DBpedia, Springer, 2010.

[33] S. Castano, A. Ferrara, S. Montanelli, Thematic exploration of linked data, in: Proc. VLDS, Seattle, WA, 2011, pp. 11–16.

[34] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, S. Decker, Sig.ma: Live views on the web of data, Web Semantics: Science, Services and Agents on the World Wide Web 8 (2010) 355–364.

[35] V. Nebot, R. B. Llavori, J. M. Pérez-Martínez, M. J. Aramburu, T. B. Pedersen, Multidimensional integrated ontologies: A framework for designing semantic data warehouses, Journal on Data Semantics XIII 13 (2009) 1–36.

[36] S. Khouri, I. Boukhari, L. Bellatreche, E. Sardet, S. Jean, M. Baron, Ontology-based structured web data warehouses for sustainable interoperability: requirement modeling, design methodology and tool, Computers in Industry 63 (2012) 799–812.

[37] B. Kämpgen, S. O'Riain, A. Harth, Interacting with statistical linked data via OLAP operations, in: Proc. of The Semantic Web Satellite Events, 2015, pp. 87–101.

[38] M. Meimaris, G. Papastefanatos, P. Vassiliadis, I. Anagnostopoulos, Efficient computation of containment and complementarity in RDF data cubes, in: Proc. EDBT, Bordeaux, France, 2016, pp. 281–292.

[39] D. Ibragimov, K. Hose, T. B. Pedersen, E. Zimányi, Towards exploratory OLAP over linked open data - a case study, in: Proc. BIRTE, Riva del Garda, Italy, 2014, pp. 114–132.

[40] I. F. Cruz, F. P. Antonelli, C. Stroe, Agreementmaker: Efficient matching for large real-world schemas and ontologies, PVLDB 2 (2009) 1586–1589.

[41] S. Araújo, J. Hidders, D. Schwabe, A. P. de Vries, SERIMI - resource description similarity, RDF instance matching and interlinking, in: Proc. OM, Bonn, Germany, 2011.

[42] R. Touma, O. Romero, P. Jovanovic, Supporting data integration tasks with semi-automatic ontology construction, in: Proc. DOLAP, Melbourne, Australia, 2015, pp. 89–98.

[43] Y. R. Jean-Mary, E. P. Shironoshita, M. R. Kabuka, Ontology matching with semantic verification, J. Web Sem. 7 (2009) 235–251.

[44] Z. Syed, T. Finin, Unsupervised techniques for discovering ontology elements from wikipedia article links, in: Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading, Association for Computational Linguistics, 2010, pp. 78–86.

[45] S. Kolahi, L. V. Lakshmanan, On approximating optimum repairs for functional dependency violations, in: Proc. ICDT, St. Petersburg, Russia, 2009, pp. 53–62.

[46] H. Müller, J.-C. Freytag, Problems, methods, and challenges in comprehensive data cleansing, Professoren des Inst. Für Informatik, 2005.

[47] G. Beskales, I. F. Ilyas, L. Golab, A. Galiullin, Sampling from repairs of conditional functional dependency violations, The VLDB Journal 23 (2014) 103–128.

[48] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, S. Auer, Quality assessment for linked data: A survey, Semantic Web 7 (2015) 63–93.

[49] C. Fürber, M. Hepp, Swiqa - a semantic web information quality assessment framework, in: ECIS, volume 15, Helsinki, Finland, 2011, p. 19.

[50] C. Böhm, F. Naumann, Z. Abedjan, D. Fenz, T. Grütze, D. Hefenbrock, M. Pohl, D. Sonnabend, Profiling linked open data with ProLOD, in: Proc. ICDEW, Long Beach, California, 2010, pp. 175–178.

[51] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen, TANE: An efficient algorithm for discovering functional and approximate dependencies, Comput. J. 42 (1999) 100–111.