Towards Exploratory OLAP on Linked Data

S. Rizzi[1], E. Gallinucci[1], M. Golfarelli[1], A. Abelló[2], and O. Romero[2]

[1] DISI, University of Bologna, Italy
[2] ESSI, Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract. In the context of exploratory OLAP, coupling the information wealth of linked data with the precision and detail of corporate data can greatly improve the effectiveness of the decision-making process. In this paper we outline an approach that enables users to extend the hierarchies in their corporate cubes through a user-guided process that explores selected linked data and derives hierarchies from them. This is done by identifying in the linked data the recurring modeling patterns that express roll-up relationships between RDF concepts and translating them into multidimensional knowledge.

Keywords: OLAP, linked data, ontologies

1 Introduction and Overview

Business intelligence (BI) techniques have been enormously accelerating and improving the decision making process in companies for two decades. However, the recent years witness a push towards two directions: on the one hand, enriching the decisional process by including, besides data extracted from the corporate sources, also external data coming for instance from the web; on the other hand, enabling data enthusiasts to build their own reports on-the-fly, without any support from ICT people. The resulting approach is often called *exploratory OLAP* because external data must be discovered and acquired [2].

When accessing external data and integrating them in the decision-making process, knowing the data semantics is important; ontologies may obviously offer a strong contribution in this direction [2]. A relevant role in this context is played by *linked data*, whose shared, structured, and interlinked nature should make them easily accessible and searchable. Unfortunately, linked data are often chaotic and badly organized, especially from the schema point of view, which often prevents users from taking full advantage of the informative wealth expressed by linked data. On the other hand, multidimensional modeling appears to be an effective approach to so-called *small analytics on big data* (essentially, aggregate queries on large volumes of data), because it enables users to get a comprehensive yet synthetic picture of the information of interest.

The work of this paper is inspired by the *fusion cubes* vision [1], where a corporate, stationary data warehouse can be dynamically extended by the users on a self-service basis, by including some external situational data. In particular, the goal of our approach, named *iMOLD* (Interactive Multidimensional Modeling of Linked Data), is to enable users to extend the hierarchies in the corporate

cubes through a user-guided process that explores selected linked data, derives hierarchies from them, and populates these hierarchies with data. This is done by identifying in the linked data the recurring modeling patterns that express roll-up relationships between RDF concepts and translating them into multidimensional knowledge, to be stored locally and shared with every user for reuse purposes. The knowledge base built and maintained by the system to store such information is called *Internal Ontology* (IO)[3], whereas any source of linked data will be referred to as an *External Ontology* (EO).

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 typical scenario that we envision can be subdivided into three iterative phases.

1. **Assessment:** the user accesses the IO to check whether the concept of interest is already present and which hierarchies have already been built around it, and she can decide to reuse the information previously acquired either by herself or by others.
2. **Acquisition:** if the concept of interest is not present in the IO or it is not satisfactorily modeled, the user can search for aggregation patterns in the EOs, build her own multidimensional schema by selecting the concepts of interest, and integrate the results into the IO.
3. **Integration:** the user launches a set of system-generated queries that create and populate new dimension tables with the data selected; these tables are then integrated with those in the corporate cubes to enable richer analyses.

The first phase is mainly a matter of delivering a smart user interface for effectively browsing the IO. The third phase requires to automatically create and execute some SPARQL queries to extract data, to transform and load them into ad hoc dimension tables, and to establish a correlation between rows in these dimension tables and rows in the corporate dimension tables. The only demanding step in the third phase is the last one, which requires inter-member mappings to be found; though this problem has not been deeply investigated in the literature, some existing approaches could be adapted to implement it (e.g., [4, 9, 6]). The second phase, on which this paper is focused, is the one that raises the most challenging and novel research issues.

Remarkably, while several previous papers address the problem of building multidimensional schemata starting from source data, most of them are meant to be used at design-time in the context of so-called *supply-driven design* and consider well-structured data sources (e.g., Entity/Relationship diagrams and relational schemata) where hierarchies can be easily detected by following functional dependencies (FDs, represented, respectively, by many-to-one relationships and by foreign keys). Conversely, our approach operates at query-time to integrate the corporate cubes with situational data; besides, the modeling heterogeneity

[3] The description of the IO structure is not given here for space reasons; we just mention that it reuses the QB4OLAP model to represent OLAP cubes in the RDF format [5] and the SM4AM metamodel to store metadata on a QB4OLAP cube [11].

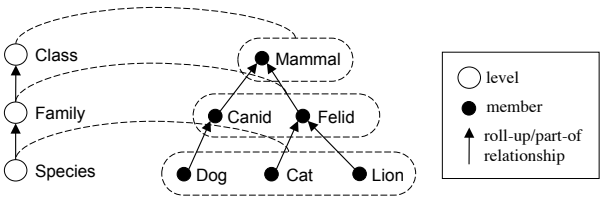

Fig. 1. An example of multidimensional modeling: levels and roll-up relationships (left), members and part-of relationships (right)

of linked data and the impossibility of describing the multiplicity of properties in the RDFS vocabulary make hierarchy detection more complex. Most of the approaches that consider ontologies and semi-structured sources work at the schema level to avoid sampling or querying instances [10]. A solution that stands in the middle is the one proposed in [12], which is based on XML schema but, in some cases, infers many-to-one relationships from instances. Finally, in [11] an instance-based approach is sketched but only basic rules are described and no formal definitions of their expressiveness is provided.

2 Aggregation Patterns in Ontologies

The multidimensional model is the core of data warehouse and OLAP applications. Our specific goal in this work is to detect hierarchies in linked data, for this reason we will focus on the modeling of hierarchies.

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 $u = (l, p, l')$ *between two levels, a* child l *and a* parent l'*, and has semantics* p*. Each level has a domain made by a set of* members*. The roll-up relationship* u *induces 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'*.*

Example 1. As a working example we use the 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 remainder of this section we describe the two main aggregation patterns in RDF data that can give rise to roll-up relationships. To this end we assume that, though the EOs may be incomplete and not fully correct, its data are statistically representative.

OLAP aggregation hierarchies express part-whole relationships, which in RDF are commonly modeled using properties. Thus, as exemplified in Figure 2, pattern (a) corresponds to two classes (e.g., ex:Family and ex:Class in Figure 3) whose instances are related by an RDF property (because of the incompleteness

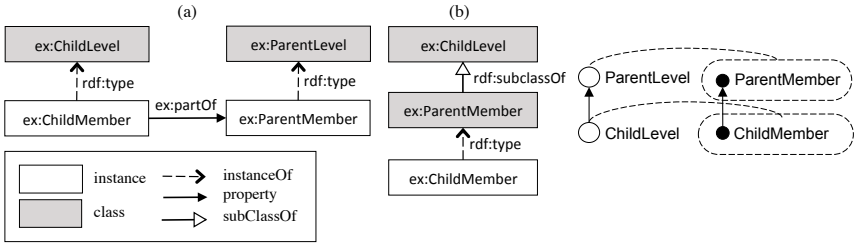

Fig. 2. The two main RDF aggregation patterns (left) and their multidimensional counterpart (right)

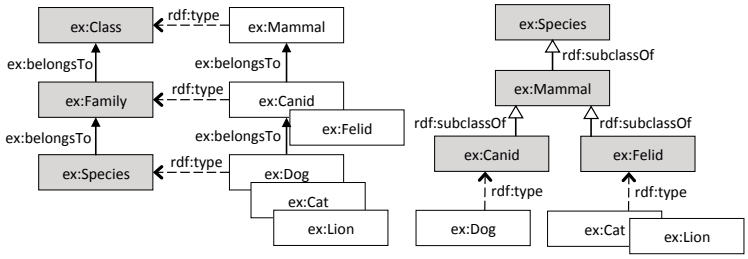

Fig. 3. The hierarchy in Figure 1 modeled in RDF using associations (left) and generalizations (right)

of linked data, we cannot assume that this property also exists at the level of classes). Noticeably, the ontology designer may have modeled the association in one direction or the other. For instance, the aggregation between ex:Family and ex:Class in Figure 3 could also have been modeled with the RDF property ex:hasFamily, where ex:Class is the domain and ex:Family is the range. For this reason, the child and parent roles can be inverted with reference to those shown in Figure 2. Another variant of this pattern arises when the instances are associated to a data type instead of a class (i.e., we have literals instead of objects); in this case there is no class modeling the parent (child) level, so its name must be provided by the user.

Unfortunately, we cannot rely on the existence of multiplicities in RDF; thus, in pattern (a) we have to sample the associated instances to check the proportion of existing many-to-one relationships (e.g., we retrieve from the SPARQL endpoint how many instances of Class are related to each instance of Family). Then, these patterns are identified only if the average cardinality of the association is close to 1 on the side of the parent, i.e., if an *approximate FD* holds [8].

The second possibility for modeling an aggregation hierarchy in RDF is based on generalization. Generalizations express an *is a* semantics that induces a subsumption between sets of instances of related classes; so, for instance, Mammal generalizes Canid and Felid since the set of mammal species is a superset of the set of species instances of Canid and Felid. But then, species can be grouped into

mammals (and reptiles), or into canids and felids, and the former grouping is coarser than the latter, which in OLAP terms translates to a part-of relationship between members Canid + Felid and Mammal. This suggests that there is roll-up relationship between two different levels, which we will call Family and Class.

From the example above we can conclude that here the classes in the EO correspond to members instead of levels. So, to find an expression of levels in this case, we must look farther. Indeed, though generalizations are binary relationships between pairs of classes, they can be grouped depending on the criteria used (*powertype* in UML terminology). Thus, in pattern (b), the class corresponding to the child level is specialized into subclasses (i.e., subsets), each corresponding to a parent member, using the parent level as a powertype. For instance, class Species is specialized into Canid and Felid based on powertype Family; these two subclasses give rise to two parent members, and their instances (Dog, Cat, and Lion) to child members. A variant of this pattern arises when child members are modeled as classes rather than instances in the linked data, which 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., Cat and Lion, whose instances could be for instance Felix and Simba) would be generalized into a superclass (Felid) that would be the corresponding parent member.

Powertypes are normally not made explicit in RDF, so the user will have to provide names for the levels corresponding to powertypes.

3 Acquisition

In this section we discuss the core phase of iMOLD, whose goal is to build aggregation hierarchies out of an EO by letting users choose the specific roll-up relationships of interest. As previously stated, the acquisition of multidimensional knowledge is done by detecting aggregation patterns on a selected EO. Exploring an EO in its entirety to find every potential roll-up relationship is clearly unfeasible. Some approaches have been devised for effectively exploring linked data using advanced visualization techniques (e.g., [7, 3]); in this work we adopt a basic approach for supporting the user interaction with the EO, and we simply require the user to first choose a class of interest c in an EO, to be mapped into a level l and used as an entry point for pattern detection.

Every detection is focused, besides on c, on a direction dir, which can be either *outbound* or *inbound*; this means that, given c and dir, we detect the patterns by exploring the triples where c (or its instances) is either the subject (dir = 'outbound') or the object (dir = 'inbound'). Indeed, given a relationship a between subject s and object o, a is equally detected 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. Pattern detection relies on a SPARQL query, to be directly submitted to the SPARQL endpoint of a selected EO. Each pattern detected is then mapped into a roll-up relationship. The detection process is done in a breadth-first fashion: this means that c is completely analyzed in its

Algorithm 1 Detect Pattern (a)

Input EO: an external ontology, c: a starting class, dir: a direction (either 'outbound' or 'inbound'); $multTol$: tolerance for giving -to-one multiplicity to an association
Output R: a set of roll-up relationships

```
1: R ← ∅                                              ▷ Initialize R
2: q ← Query(c, dir)                                  ▷ Create q...
3: A ← Execute(EO, q)                                 ▷ ...and execute it against EO
4: for each a ∈ A do                                  ▷ Find the roll-up relationships in A
5:     if rightCard(a) ≤ multTol then                 ▷ If a is -to-one...
6:         R ← R ∪ {a}                                ▷ ...add it to R
7:     else
8:         if leftCard(a) ≤ multTol then              ▷ If a is one-to-...
9:             R ← R ∪ {a^-1}                         ▷ ...add its inverse to R
10: return R
```

relationships with other classes or datatypes but no recursive detection is done to avoid an exponential explosion. The roll-up relationships selected by the user lead to new classes, from which the user can iteratively perform new searches.

A key feature of linked data is that of creating connections between different ontologies, so as to enable the reuse of knowledge. In an ontology, this connection is provided by mentioning 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 linked data is exploited to enable users to jump from the currently explored ontology to a different one whenever the application of a pattern detects an external concept (i.e., one whose namespace is different from the one of the starting concept). At this point, if the user wishes to continue the exploration of the external concept, she will be asked by the system whether she wants to jump to the ontology that concept belongs to. If so, the transition can be seamlessly made by launching the next searches for patterns on the SPARQL endpoint of the new ontology.

In the remainder of this section we give a short description of how the two main aggregation patterns seen in Section 2 can be detected on an RDF ontology.

3.1 Acquisition of Pattern (a)

Detecting this pattern means determining whether a property p involving c can be mapped to a roll-up relationship, where the domain and range of p are mapped to 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 the domain of p, and r is the range of p. Association a has a* right cardinality *$rightCard(a)$, i.e., the average number of distinct instances of r linked to an instance of d through p, and a* left cardinality *$leftCard(a)$, i.e., the average number of distinct instances of d linked to an instance of r through p. Given $a = (d, p, r)$, with $a^{-1} = (r, p, d)$ we denote its inverse.*

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. 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.

The pseudocode for detecting association-based patterns is shown in Algorithm 1. First of all, a SPARQL query q is generated by function *Query* (line 2); given a starting class c and a direction dir, q returns a set A of associations involving c in direction dir, together with the left and right cardinality of each association. The specific form of q depends on dir; for instance, this is the query generated in the outbound direction:

```
SELECT ?prop ?range (?nProp/?nObj AS ?rightCard) (?nProp/?nSubj AS ?leftCard) ?nObj ?nSubj
WHERE
{  SELECT ?prop ?range (COUNT(*) AS ?nProp) (COUNT(DISTINCT(?obj)) AS ?nObj)
          (COUNT(DISTINCT(?subj)) AS ?nSubj)
   WHERE {  ?subj rdf:type ?c .                          ▷ Step 1: select instances of c
            ?subj ?prop ?obj .                  ▷ Step 2: retrieve the associations of each subject
            ?obj rdf:type ?range . }            ▷ Step 3: retrieve the classification of each object
   GROUP BY ?prop ?range }          ▷ Step 4: group the instances to get the list of associations
```

Function *Execute* (line 3) submits q to the SPARQL endpoint of the EO. In the lines from 4 to 9, the associations in A are filtered according to their multiplicities and added to R. In lines 5 and 8, threshold $multTol$ is applied to right (left) cardinalities to determine if each association a is -to-one (one-to-).

We close this section by remarking that, due to the huge number of roll-up relationships potentially found, only the most relevant relationships in R should be returned to the user, being the relevance of a relationship defined in function of its support in the EO.

3.2 Acquisition of Pattern (b)

This pattern is cheaper to detect than the previous one because (i) no query at the instance level is required and (ii) the only inter-class property that must be considered is rdfs:subClassOf. On the other hand, it applies less intuitive transformations to classes: whereas distinct classes always correspond to distinct levels in pattern (a), in (b) 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.

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: (i) when operating inbound, the user must select the subclasses of c of interest; (ii) when operating outbound, the user must select the superclass of interest. In both cases, the user must provide the name of the powertype.

Though the algorithm pseudocode is not shown for space reasons, we mention that, when a generalization taxonomy is iteratively explored by the user, the process of mapping the patterns detected into levels of a hierarchy H is more complex than for pattern (a), because the way the mapping is done depends not only on *dir*, but also on the previous structure of H.

4 Implementation

The prototype we built to test iMOLD is implemented as a Java web application; we rely on the Jena Library for the communication with SPARQL endpoints and for in-memory manipulation of ontologies. The IO is stored within a simple RDF file. Finally, we used Javascript to implement the user interface. To increase the effectiveness of the user experience, two alternative interaction approaches are proposed to the user in our prototype: an *ontology-driven experience*, oriented to users who have good familiarity with ontologies and semantic web, where the focus is set on the EO and the user intentionally detects one pattern or the other; and an *OLAP-driven experience*, targeted to users who have good familiarity with multidimensional modeling, where patterns are transparent to users and the hierarchy is progressively built using OLAP-inspired operators.

References

1. Abelló, A., et al.: Fusion cubes: Towards self-service business intelligence. IJDWM 9(2), 66–88 (2013)
2. Abelló, A., et al.: Using semantic web technologies for exploratory OLAP: a survey. IEEE Trans. Knowl. Data Eng. 27(2), 571–588 (2015)
3. Castano, S., Ferrara, A., Montanelli, S.: Thematic exploration of linked data. In: Proc. VLDS. pp. 11–16 (2011)
4. Chang, K.C., Garcia-Molina, H.: Mind your vocabulary: Query mapping across heterogeneous information sources. In: Proc. SIGMOD. pp. 335–346 (1999)
5. Etcheverry, L., Vaisman, A.: QB4OLAP: A vocabulary for OLAP cubes on the semantic web. In: Proc. COLD. CEUR-WS.org (2012)
6. Golfarelli, M., Mandreoli, F., Penzo, W., Rizzi, S., Turricchia, E.: OLAP query reformulation in peer-to-peer data warehousing. Inf. Syst. 37(5), 393–411 (2012)
7. Hirsch, C., Hosking, J., Grundy, J.: Interactive visualization tools for exploring the semantic graph of large knowledge spaces. In: Proc. VISSW. vol. 443 (2009)
8. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: TANE: An efficient algorithm for discovering functional and approximate dependencies. Comput. J. 42(2), 100–111 (1999)
9. Jovanovic, P., Romero, O., Simitsis, A., Abelló, A., Mayorova, D.: A requirement-driven approach to the design and evolution of data warehouses. Information Systems 44, 94–119 (2014)
10. Romero, O., Calvanese, D., Abelló, A., Rodriguez-Muro, M.: Discovering functional dependencies for multidimensional design. In: Proc. DOLAP. pp. 1–8 (2009)
11. Varga, J., Romero, O., Pedersen, T.B., Thomsen, C.: SM4AM: A semantic metamodel for analytical metadata. In: Proc. DOLAP. pp. 57–66 (2014)
12. Vrdoljak, B., Banek, M., Rizzi, S.: Designing web warehouses from XML schemas. In: Proc. DaWaK. pp. 89–98 (2003)