# SM4MQ: A Semantic Model for Multidimensional Queries

Jovan Varga[1(✉)], Ekaterina Dobrokhotova[1], Oscar Romero[1], Torben Bach Pedersen[2], and Christian Thomsen[2]

[1] Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain
{jvarga,oromero}@essi.upc.edu, ekaterina.dobrokhotova@est.fib.upc.edu
[2] Aalborg Universitet, Aalborg, Denmark
{tbp,chr}@cs.aau.dk

**Abstract.** On-Line Analytical Processing (OLAP) is a data analysis approach to support decision-making. On top of that, Exploratory OLAP is a novel initiative for the convergence of OLAP and the Semantic Web (SW) that enables the use of OLAP techniques on SW data. Moreover, OLAP approaches exploit different metadata artifacts (e.g., queries) to assist users with the analysis. However, modeling and sharing of most of these artifacts are typically overlooked. Thus, in this paper we focus on the query metadata artifact in the Exploratory OLAP context and propose an RDF-based vocabulary for its representation, sharing, and reuse on the SW. As OLAP is based on the underlying multidimensional (MD) data model we denote such queries as MD queries and define *SM4MQ*: A Semantic Model for Multidimensional Queries. Furthermore, we propose a method to automate the exploitation of queries by means of SPARQL. We apply the method to a use case of transforming queries from *SM4MQ* to a vector representation. For the use case, we developed the prototype and performed an evaluation that shows how our approach can significantly ease and support user assistance such as query recommendation.

**Keywords:** Semantic Web · Vocabulary · OLAP · Query modeling

## 1 Introduction

On-Line Analytical Processing (OLAP) is a well-established approach for data analysis to support decision-making [14]. Due to its wide acceptance and successful use by non-technical users, novel tendencies endorse broadening of its use from solutions working with in-house data sources to analysis considering external and non-controlled data. A vision of such settings is presented as Exploratory OLAP [1] promoting the convergence of OLAP and the Semantic Web (SW). The SW provides a technology stack for publishing and sharing of data with their semantics and many public institutions, such as Eurostat, already use it to make their data publicly available. The Resource Description Framework (RDF) [7] is the backbone of the SW representing data as directed triples that form a graph where each triple has its semantics defined. Querying of RDF data is supported by SPARQL [17], the standard query language for RDF.

To facilitate data analysis, OLAP systems typically exploit different metadata artifacts (e.g., queries) to assist the user with analysis. However, although extensively used, little attention is devoted to these metadata artifacts [23]. This originates from traditional settings where very few (meta)data are open and/or shared. Thus, [23] proposes the Analytical Metadata (AM) framework, which defines AM artifacts such as schema and queries that are used for user assistance in settings such as Exploratory OLAP. In this context, analysis should be collaborative and therefore these metadata artifacts need to be open and shared. Thus, SW technologies are good candidates to model and capture these artifacts.

A first step for (meta)data sharing among different systems is to agree about (meta)data representation, i.e., modeling. As RDF uses a triple representation that is generic, the structure of specific (meta)data models is defined via RDF vocabularies providing semantics to interpret the (meta)data. Thus, the AM artifacts are modeled in [22] proposing SM4AM: a semantic metamodel for AM. Due to the heterogeneity of systems, the metamodel abstraction level is used to capture the common semantics and organization of AM. Then, metadata models of specific systems are defined at the model level instantiating one or more AM artifacts. For instance, the schema artifact for Exploratory OLAP can be represented using the QB4OLAP vocabulary to conform data to a multidimensional (MD) data model for OLAP on the SW [24]. QB4OLAP further enables running of MD queries to perform OLAP on the SW [25]. However, the representation of these queries to support their sharing, reuse, and more extensive exploitation on the SW is yet missing. Thus, in the present paper we propose a model for MD queries and explain how it supports sharing, reuse, and can also be used to facilitate metadata processing, e.g., for user assistance exploitations such as query recommendations. In particular, the contributions of this paper are:

– We propose *SM4MQ*: A Semantic Model for MD Queries formalized as an RDF-based vocabulary of typical OLAP operations (see [21]). The model captures the semantics of common OLAP operations at the conceptual level and supports their sharing and reuse via the SW.
– We define a method to automate the exploitation of *SM4MQ* queries by means of SPARQL. The method is exemplified on a use case to transform a query from *SM4MQ* to a vector representation. The use case shows an example of generating vectors (forming a matrix) as analysis-ready data structures that can be used by existing recommender systems [2] and approaches to query recommendations (e.g., [6]).
– We developed a prototype and used a set of MD queries to evaluate our approach for the chosen use case. The evaluation shows that *SM4MQ* significantly eases and supports the automation of query exploitation by means of SPARQL.

The paper is organized as follows. The next section explains the preliminaries of our approach. Then, Sect. 3 proposes the MD query model. Section 4 presents the proposed method and the related use case. Section 5 discusses the use case evaluation results. Finally, Sect. 6 discusses the related work and Sect. 7 concludes the paper.

## 2 Background

In this section, we introduce the necessary preliminaries and a running example used throughout the paper. First, we explain the MD model and the most popular OLAP operations. Then, we discuss the use of SW and QB4OLAP for MD models. The formalization of QB4OLAP concepts and OLAP operations can be found in [10] and in the present paper we provide the necessary intuition for understanding the proposed query model. The running example is incrementally introduced in each of the subsections.

### 2.1 Multidimensional Model and OLAP Operations

The MD model organizes data in terms of *facts*, i.e., data being analyzed, and *dimensions*, i.e., analytical perspectives [14]. Dimensions consist of *levels* representing different data granularities that are hierarchically organized into dimension *hierarchies*. Levels can have *attributes* that further describe them. Facts contain *measures* that are typically numerical values being analyzed. Data conforming to an MD schema are referred to as a *data cube* that is being navigated (e.g., data granularity is changed) via OLAP operations. For instance, Fig. 1 illustrates an MD schema created for the European Union asylum applicants data set available in a Linked Data version of the Eurostat data[1]. In the data set, the number of asylum applications as a measure, can be analyzed according to the age, sex, type of application (Asyl_app), destination country (Geo), country of origin (Citizenship), and month of application (RefPeriod) levels of related dimensions. Moreover, the data can be aggregated from months to quarters and likewise to years, from country of origin to continent, and from destination country to continent or government type as additional levels in related dimensions.

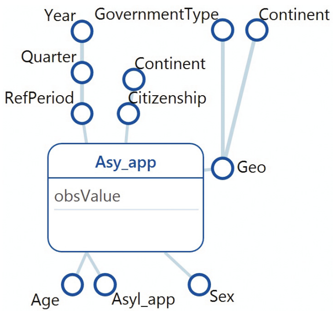

**Fig. 1.** Asylum data set schema (in DFM Notation [12])

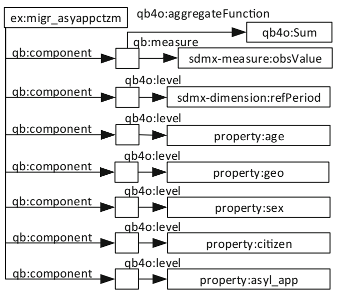

**Fig. 2.** Asylum data set QB4OLAP schema representation

[1] http://eurostat.linked-statistics.org/.

To navigate a data cube, OLAP operations are used and different OLAP algebras have been proposed [18]. In the present paper, we consider the set of OLAP operations used in [9,10] that are defined at the conceptual level as discussed in [8]. The considered OLAP operations are described in the following.

The *ROLL-UP* operation aggregates data from a finer granularity level to a coarser granularity level in a dimension hierarchy of a data cube. For instance, in case of the schema in Fig. 1 data can be aggregated from the month level to the year level for the RefPeriod dimension. Similarly, the *DRILL-DOWN* operation as its inverse disaggregates data from a coarser granularity level to a finer granularity level in a dimension hierarchy of a data cube. Furthermore, the *DICE* operation takes a data cube and applies a boolean condition expressed over a level (attribute) and/or measure value over it. For instance, for the schema in Fig. 1 a user may be interested in number of asylum applications only for the years 2009 and 2010. Finally, the *SLICE* operation removes a dimension or a measure from a data cube. For instance, a user may not be interested in the age of the applicants and thus remove the related dimensions.

## 2.2 The Semantic Web Technologies

As mentioned in the introduction, the SW technologies provide means for flexible (meta)data representation and sharing. RDF, which is the SW backbone, represents data in terms of directed subject - predicate - object triples that comprise an RDF graph where subjects and objects are nodes and predicates are edges. Subject and predicate are represented with IRIs, i.e., unique resource identifiers on the SW, while objects can either be IRIs or literal values. Furthermore, RDF supports representation of the data semantics via the rdf:type property. In the context of sharing, the Linked Data initiative [13] strongly motivates interlinking of RDF data on the SW to support identification of related/similar/same concepts. Finally, the RDF data can be queried with SPARQL [17], the standardized query language for RDF, which supports their systematic exploration.

To support the publishing of MD data and their OLAP analysis directly on the SW, two RDF vocabularies were proposed, namely the RDF Data Cube (QB) and QB4OLAP vocabularies. As the former vocabulary was primarily designed for statistical data sets, the latter one was proposed to extend QB with necessary concepts to fully support OLAP. A detailed discussion on this is presented in [24] where it is also explained how existing QB data sets can be enriched with the QB4OLAP semantics. Thus, in the present paper we consider QB4OLAP for the representation of the MD data on the SW and Fig. 2 illustrates how the MD schema from Fig. 1 can be represented with QB4OLAP. Note that for simplicity reasons we represent only the finest granularity levels.

Once a data cube is published using QB4OLAP, the OLAP operations from the previous subsection can be performed. However, a metadata model for representing these queries is yet missing. Such a model can be created by instantiating the SM4AM metamodel (see [21,22] for the initial and extended SM4AM versions, respectively). The metamodel represents the query AM artifact with several meta classes that we explain next. First, the sm4am:UAList element is a complex element that combines atomic elements that include data exploration actions (i.e., sm4am:DataExplorationAction with its sm4am:ManipulationAction subclass). Thus, the metamodel elements can be instantiated as an MD query that combines OLAP operations and we present the details in the next section.

## 3 A Semantic Model for Multidimensional Queries

In this section, we define *SM4MQ* as an RDF-based vocabulary to represent the introduced OLAP operations. The model is created by instantiating the related SM4AM metamodel elements. It is built around the QB4OLAP model for representing an MD schema and explained with examples related to the running example schema.

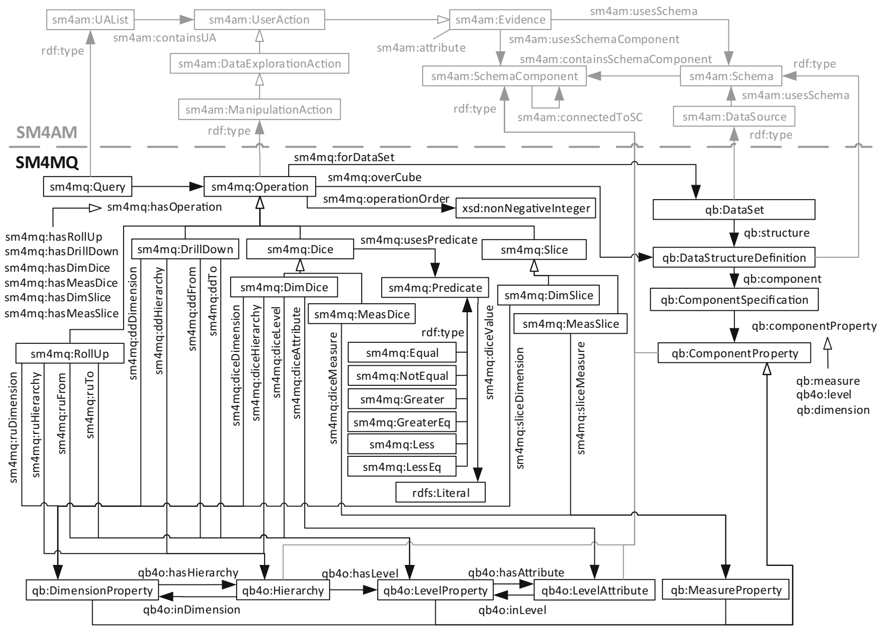

**Fig. 3.** A semantic model for multidimensional queries

### 3.1 MD Query Model

Figure 3 illustrates the complete *SM4MQ* query model. Furthermore, the figure also shows how *SM4MQ* relates to the SM4AM metamodel concepts and reuses concepts from the QB, QB4OLAP, and RDFS vocabularies. The central concept of the model is sm4mq:Query representing a query. A query can be related to a simply ordered set of OLAP operations (i.e., subclasses of sm4mq:Operation) via subproperties of sm4mq:hasOperation. OLAP operations are organized in a simply ordered set as each of them directly relates to the query and they are mutually ordered, e.g., the order of ROLL-UP and DRILL-DOWN operations is relevant to determine the final granularity of the data cube. Each operation relates to a data cube schema (i.e., qb:DataStructureDefinition) via sm4mq:overCube and to a data set (i.e., qb:DataSet) to which data belong to via sm4mq:forDataSet. This way, operations belonging to a single query can operate over different schemata and data sets (inspired by the federated queries mechanism in SPARQL). In the next subsections, we explain each of the OLAP operations. Note that we follow the formalization given in [9] and we also enrich the model with additional information needed to facilitate sharing.

## 3.2 ROLL-UP and DRILL-DOWN Operations

Following the definition in [9], ROLL-UP (i.e., sm4mq:RollUp) is represented with a data cube schema (i.e., qb:DataStructureDefinition), a dimension (i.e., qb:DimensionProperty), and a level to roll-up to (i.e., qb4o:LevelProperty). In addition to these concepts, *SM4MQ* also represents the level from which the roll-up is performed, its order in the query, and the dimension hierarchy (i.e., qb4o:Hierarchy) used. The related properties are illustrated in Fig. 3. In general, the roll-up from and hierarchy concepts can be inferred from a sequence of OLAP operations, however their explicit representations makes the ROLL-UP operation model self-contained such that it can be easily shared. The *SM4MQ* representation of the ROLL-UP example from Sect. 2 of aggregating data from the month to the year level over the running example schema is illustrated in Fig. 4. The example shows the ROLL-UP instance and also includes the related *SM4MQ* concepts (depicted in gray).

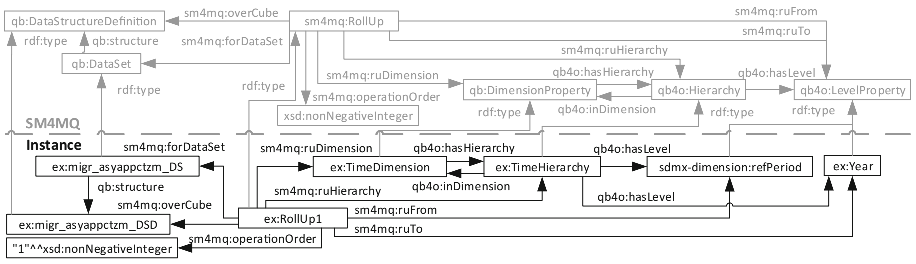

**Fig. 4.** ROLL-UP instance example

Following the definition in [9], DRILL-DOWN (i.e., sm4mq:DrillDown) is represented with a data cube schema (i.e., qb:DataStructureDefinition), a dimension (i.e., qb:DimensionProperty), and a level to drill-down to (i.e., qb4o:LevelProperty). In addition to these concepts, *SM4MQ* also represents the level from which the drill-down is performed, its order in the query, and the dimension hierarchy (i.e., qb4o:Hierarchy) used for the same reasons as in the case of ROLL-UP. An example of DRILL-DOWN is analogous to the ROLL-UP one and we omit it for space reasons.

### 3.3 DICE Operation

Following the definition in [9], DICE (i.e., sm4mq:Dice) is represented with a data cube schema (i.e., qb:DataStructureDefinition) and a boolean condition (i.e., sm4mq:Predicate) over a dimension (i.e., qb:DimensionProperty) or a measure (i.e., qb:MeasureProperty). We represent these two cases separately for the atomicity of operations that also facilitates sharing. Thus, in *SM4MQ* we create two subclasses for DICE, sm4mq:DimDice as DICE applied over a dimension and sm4mq:MeasDice as DICE applied over a measure. The former one relates to a dimension, hierarchy, level, and optionally level attribute, while the latter relates to a measure. For both cases we define the order and consider a set of relational predicates that includes equals to (i.e., sm4mq:Equal), not equals to (i.e., sm4mq:NotEqual), greater than (i.e., sm4mq:Greater), greater than or equal (i.e., sm4mq:GreaterEq), less than (i.e., sm4mq:Less), and less than or equal (i.e., sm4mq:LessEq). Each specific relational operator is an instance of the sm4mq:-Predicate class (similarly to the case of aggregate functions in QB4OLAP), and it is related to rdfs:Literal used for the representation of the concrete values. The *SM4MQ* representation of the DICE example from Sect. 2 where a user is interested in number of asylum applications only for the years 2009 and 2010 for the running example schema is illustrated in Fig. 5. The example shows the DICE instance and includes the related *SM4MQ* concepts (depicted in gray).

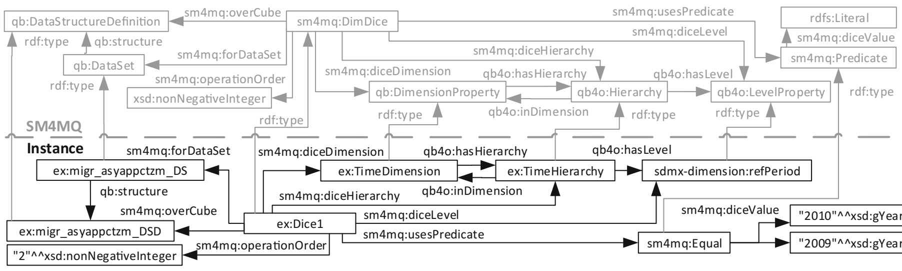

**Fig. 5.** DICE instance example

### 3.4 SLICE Operation

Following the definition in [9], SLICE (i.e., sm4mq:Slice) is represented with a data cube schema (i.e., qb:DataStructureDefinition) and a dimension (i.e., qb:-DimensionProperty) or a measure (i.e., qb:MeasureProperty). Again, we represent these two cases separately for the atomicity of operations that also facilitates sharing. Thus, in *SM4MQ* we create two subclasses for SLICE, sm4mq:DimSlice as SLICE applied over a dimension and sm4mq:MeasSlice as SLICE applied over a measure. For both cases we also define the order. The *SM4MQ* representation of the SLICE example from Sect. 2 where a user is not interested in the age of the applicants and thus removes the related dimensions for the running example schema (see Fig. 1) is illustrated in Fig. 6. The example shows the SLICE instance and also includes the related *SM4MQ* concepts (depicted in gray).

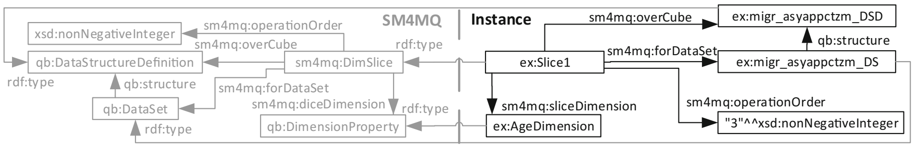

**Fig. 6.** SLICE instance example

## 4 Exploiting SM4MQ

In this section, we discuss the benefits of having a semantic model for MD queries. First, we discuss why modeling and capturing of the MD query semantics is essential for their exploitation. We then propose a method on how *SM4MQ* semantics can serve to automate the exploitation of *SM4MQ* queries by means of SPARQL. We also present a use case that shows how can the method be used to define the query transformations from *SM4MQ* to a vector representation.

### 4.1 Modeling and Semantics

We first explain the challenges behind the current state of the art to represent queries and explain how *SM4MQ* overcomes them. MD queries in existing approaches are typically stored in query logs [23]. From logs, queries are parsed to extract semantics needed for further processing. The parsing is dependent on the particular technology used (e.g., SQL) where different patterns need to be applied to identify OLAP operations introduced in Sect. 2. Once identified, the OLAP operations are represented with internal data structures and any processing of the queries is directly dependent on these internals. In the context of Exploratory OLAP [1] and next generation BI systems [23], this situation leads to several challenges:

1. *Repetitive model designing of MD queries* – Instead of considering query metadata as a first-class citizen and conforming them to a dedicated model, repetitive efforts are invested into designing ad-hoc query representations for each system.
2. *Repetitive adjustments for exploitation* – The use of hard-coded and ad-hoc query models hinders the use of existing algorithms for their exploitation. As the internals on query representation are typically not available, existing algorithms need to be adjusted to each system-specific query model.
3. *Burdensome query sharing* – Overlooking of query modeling obstructs query reuse among different systems. This becomes especially relevant, considering Exploratory OLAP and public data sets on the SW where not only data but also queries can be shared among the users. Moreover, once modeled, queries can be made publicly available so that users can exploit them for different purposes.
4. *The need for IT people support* – Working with internal query representation requires technical skills that are not characteristic of OLAP end-users.

Thus, preparing these queries (e.g., extracting the relevant semantics) requires the support of IT people. In the OLAP context, the data preparation by means of a correct ETL process may take up to 80% of the entire DW project as reported by Gartner [20]; illustrating the enormous efforts even from trained professionals.

The *SM4MQ* model for MD queries captures the MD semantics at the conceptual abstraction level using the SW technologies. Thus, it overcomes the previous challenges in the following way:

1. *SM4MQ* is a model of MD queries covering the OLAP operations specified in Sect. 2 that are commonly accepted and used in OLAP systems. Furthermore, the use of RDF makes it flexible to be extended with additional operations. Moreover, it can be linked with other RDF-based models using Linked Data principles (see Sect. 2).
2. Being an RDF-based model, *SM4MQ* provides a common semantics which exploitation algorithms can query in a standardized way via SPARQL.
3. The conforming of queries to *SM4MQ* directly supports their sharing via Linked Data principles, i.e., publishing on the SW in the RDF format. This way, different systems can make their queries available and reusable.
4. The semantics of MD queries captured at the conceptual abstraction level is understandable even for non-technical OLAP end-users [10,24]. It can automatically be transformed into different data structures (e.g., vectors) via SPARQL and thereby benefit from algorithms working over the related structures (e.g., computing cosine similarity between vectors).

Once MD queries are represented with *SM4MQ*, we can use off-the-shelf tools (i.e., triple stores) to store and query them in a standardized way via SPARQL.

### 4.2 Automating SM4MQ Exploitation

Instead of extracting and parsing queries like in typical settings, *SM4MQ* supports automation of query exploitations as their RDF representation can be directly retrieved via SPARQL. Using this benefit, we propose a method that automates the transformation of queries from *SM4MQ* to other query representations. We also provide a use case where we exemplify our claims by transforming *SM4MQ* queries into analytical vectors that can be used to perform advanced analysis such as query comparison and undertake recommendations. Vector-based representations have been typically used to compute similarities (e.g., the cosine similarity) that have been widely used in recommender systems [2]. Hence, several of the state of the art query recommendation approaches such as [6] use vectors to represent queries and compute similarities (see [3]). Next, we explain the method (sub)tasks and exemplify each task on our use case.

**Task 1. Choosing the analytical structure** is the initial task where the target data structure needs to be chosen. The analytical structure then directly determines the following tasks that define its *analytical features* and populate the analytical structure. An analytical feature is a set of one or more model elements (i.e., nodes or edges) representing a model characteristic, e.g., an operation used in a query, that is relevant for the desired analysis such as comparison of queries.

*Use Case.* We focus on a vector representation as an analytical structure.

**Task 2. Defining analytical features** identifies the elements of a given model (e.g., *SM4MQ*) that should be considered as analytical features. As an analytical feature can involve one or more model elements, it can either be *atomic*, i.e., consisting of one model element, or *composite* including two or more model elements. We next explain the two subtasks of Task 2.

***Task 2.1. Selecting model element(s) to form analytical features*** is the task where one or more model elements are selected to form an atomic or composite feature, respectively. It is performed for each analytical feature. Here, a single or several connected elements of the model are selected to be considered as an atomic or a composite feature, respectively.

*Use Case.* Analytical features will be a part of the vector representation, i.e., they will be represented with vector elements. As the whole vector will represent a query, sm4mq:Query is not an analytical feature. Instead, the analytical features to be defined are different OLAP operations, i.e., the subclasses of sm4mq:Operation, with their successor nodes (e.g., schema elements such as level). Thus, each OLAP operation is a composite analytical feature. Considering that all the queries are run over the same data cube, the data set and schema elements are omitted, as well as the operation order.

***Task 2.2. Defining the level of detail for each analytical feature*** relates to the model elements forming analytical features. For each analytical feature, the level of detail needs to be defined for each of the model elements forming that feature. There are two possible levels of detail. One is the *basic level of detail*, meaning that the model element in an analytical feature should be considered without its instances. The other option is the *comprehensive level of detail* where a model element in an analytical feature should be considered together with its instances.

*Use Case.* In a vector representation, this means that a model element with the base level of detail takes a single vector element, e.g., if an operation is used or not. Accordingly, in case of a model element with the comprehensive level of detail, there is a vector element for each instance of the model element, e.g., one vector element for each possible dimension that can be used in an operation. This way, a vector length is defined by vector elements needed for the model elements in each analytical feature.

As an example, we next define the level of detail in case of the ROLL-UP operation as an analytical feature. Here, there should be a vector element indicating if there is any ROLL-UP in the query (i.e., the basic level of detail) and for each dimension used in one or more ROLL-UPs there should be a sequence of vector elements (due to the comprehensive level of detail) including: an element identifying the dimension, an element for each dimension hierarchy, an element for each possible from-level, and an element for each possible to-level. Note that the aggregate function used in ROLL-UP is defined by the data structure definition and thus the same for all queries. For example, Table 1 illustrates a part

**Table 1.** A roll-up piece of vector instance

| ROLL-UP (Operation) | D1 (Dimension) | H1 (Hierarchy) | RefPeriod (From-level) | Quarter (From-level) | Year (From-level) | RefPeriod (To-level) | Quarter (To-level) | Year (To-level) | D2 (Dimension) | H2 (Hierarchy) | Citizenship (From-level) | Continent (From-level) | Citizenship (To-level) | Continent (To-level) | ... | D6 (Dimension) | H7 (Hierarchy) | Age (From-level) | Age (To-level) | DRILL-DOWN (Operation) | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 |  |

of the vector instance for the running example schema related to the ROLL-UP operation from Fig. 4. The non-zero values shown from the first vector element at the left specify that it is a ROLL-UP operation, over the dimension D1 and hierarchy H1, from the RefPeriod level to the Year level.

**Task 3. Populating the analytical structure** focuses on taking a query instance in *SM4MQ* and populating the chosen analysis-ready data structures. This task depends on the analytical feature definition from the previous task and can be automated with SPARQL query templates. It consists of the two subtasks that we explain in the sequel.

*Use Case.* This task refers to the population of all vector elements.

***Task 3.1. Retrieving model instances*** is the task where template SPARQL queries are defined to automatically retrieve the model element instances related to the analytical features. There are two types of templates. The first type retrieves all instances of the model elements that are nodes and belong to one or more analytical features. The related SPARQL query is shown in Query 1. Note that variables between two '?' are parameters that should be replaced with the related IRIs, e.g., node IRIs in the previous case. This way, all instances of model elements related to either atomic or composite analytical features are retrieved.

**Query 1** . *Retrieve Model Element Instances*

```
SELECT DISTINCT ?i
WHERE {
 i? rdf:type ?nodeIRI? . }
```

**Query 2** . *Retrieve Roll-Ups*

```
SELECT DISTINCT ?r ?d ?h ?fromL ?toL
WHERE {
 ?q? rdf:type sm4mq:Query ;
    sm4mq:hasRollUp ?r .
 ?r rdf:type sm4mq:RollUp ;
    sm4mq:ruDimension ?d ;
        sm4mq:ruHierarchy ?h ;
        sm4mq:ruFrom ?fromL ;
        sm4mq:ruTo ?toL . }
```

The other template type retrieves the instances of graph patterns that include both nodes and edges related to composite analytical features. This way, all model instances that are used in composite analytical features are retrieved. For instance, Query 2 retrieves the model elements related to ROLL-UP (as a composite analytical feature) of a particular query, where the ?q? parameter is the query IRI.

*Use Case.* Benefiting from the *SM4MQ* model, this task can be automated with Algorithm 1. The algorithm takes a metadata graph containing the *SM4MQ* queries and QB4OLAP schema for a data set and returns the matrix populated with vectors of all the queries. For simplicity of explanation we consider that the graph contains metadata for a single data set. In lines 2 and 3, the algorithm first retrieves queries and schema triples and this can be automatically performed with SPARQL queries based on the *SM4MQ* and QB4OLAP semantics. Then, line 4 initializes the matrix, i.e., defines the number of columns, based on the schema (see above for the vector instance structure). The rest of the algorithm belongs to the following task and is explained in the sequel.

***Task 3.2.*** **Computing values for each analytical feature** is the final task that takes the model instances previously retrieved and processes them according to the defined analytical features to populated the chosen analytical structure. This processing can be simple and generate values 1 or 0 to show if a model element instance has been used or not in an analytical feature, or be a customized function.

*Use Case.* To populate vector elements, we apply a simple computation and consider that the values of vector elements are 1 or 0 (meaning either the presence or absence of that element). Using this logic, lines 5 to 13 of Algorithm 1 populate the matrix as follows. For each query, a vector is created again based on the schema (see line 6), populated according to the OLAP operations used in the query (see lines 7 to 12), and added to the matrix (see line 13). The nested functions in lines 7 to 12 run SPARQL queries and return specific OLAP operations for an MD query. An example of a SPARQL query for *q.getRollUps*() used in line 7 is illustrated in Query 2. Finally, line 14 returns the resulting matrix. Thus, benefiting from the *SM4MQ* query modeling and semantics, the population of the matrix can be completely automated using Algorithm 1. The populated matrix can be used for computing similarities (e.g., cosine similarity) between vectors and other exploitations. In the next section, we present a prototype for our use case and discuss on practical benefits of our approach.

**Algorithm 1.** Populate the matrix of queries

```
   Input: graph;                                  // metadata graph with queries and schema
   Output: matrix;                                          // matrix representing queries
1  begin
2      queries = graph.getQueries();
3      schema = graph.getSchema();
4      matrix.init(schema);
5      foreach q ∈ queries do
6          vector.init(schema);
7          vector.addRollUps(q.getRollUps());
8          vector.addDrillDowns(q.getDrillDowns());
9          vector.addDimDices(q.getDimDices());
10         vector.addMeasDices(q.getMeasDices());
11         vector.addDimSlices(q.getDimSlices());
12         vector.addMeasSlices(q.getMeasSlices());
13         matrix.addVector(vector);
14     return matrix;
```

## 5 Use Case Evaluation

To evaluate our approach for the specified use case, we implemented a prototype and used a set of 15 MD queries related to the running example data set (see [21] for more details). The prototype includes the MD querying module (MDM) for the query generation and the to-Vector Transformation and Comparison module (VTC). MDM provides a GUI that enables a user to generate MD queries. It automatically generates *SM4MQ* query metadata as well as OLAP queries in Cube Query Language (CQL) [10] used by QB4OLAP explorer[2] for querying of QB4OLAP data cubes. The *SM4MQ* queries are stored in the SPARQL endpoint. Then, VTC retrieves the (*SM4MQ*) query and (QB4OLAP) schema metadata, transforms queries into vectors using Algorithm 1, and compares queries using the cosine similarity. Assuming the existence of a single data set and its metadata on the endpoint, VTC takes the endpoint address as a parameter and based on the *SM4MQ* and QB4OLAP semantics automatically retrieves the needed metadata. For space reasons, more implementation and evaluation details can be found in the technical report (see [21]) and next we focus on the key evaluation aspects.

One evaluation aspect of our approach for the selected use case is the size of the vector space, i.e., number of vector elements. As discussed in [2], for content-based recommender systems the pre-processing phase includes the definition of vector features. In the case of the running example data set, the vector defined by VTC is of size 180 and follows the structure (i.e., elements ordering) as defined in Sect. 4. Thus, without *SM4MQ* and VTC, this task needs to be performed *manually* entailing the tedious analysis of CQL algebra and exploration of data set schema. Once having the vector structure, the other evaluation aspect considers the population of the vector structures. Benefiting from *SM4MQ*, VTC automatically generates SPARQL queries to populate the vector for each query. Without *SM4MQ* and VTC, this task entails creation of a parser for CQL queries that populates the manually defined vector structures.

Finally, we show the degree of automation achieved in this process with the number of automatically triggered SPARQL queries by VTC. First, for the running example data set, VTC automatically triggered 32 SPARQL queries to retrieve its schema. The number of SPARQL queries in this context depends on the schema structure, e.g., how many dimensions or hierarchies exist. Furthermore, in the context of queries, we used a set of 15 *SM4MQ* queries and VTC automatically triggered 91 SPARQL queries related to *SM4MQ* queries including one SPARQL query to retrieve all IRIs of *SM4MQ* queries and 6 SPARQL queries for each *SM4MQ* query (one for each OLAP operation in *SM4MQ*). The execution of all the SPARQL queries took 2 s in average. Otherwise, if a user runs each SPARQL query manually using prepared templates and takes only 2 s/query in average, it would take 246 s. Thus, in this case VTC enables speed up of at least 100 times just for this task. Finally, VTC calculated similarity between the queries and the results indeed reflected similarities from OLAP and

[2] https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/queries.

business perspectives. Thus, we confirmed that our approach also significantly eases the exploitation of MD queries represented with *SM4MQ*.

## 6 Related Work

Typically, traditional approaches consider MD queries in terms of OLAP algebras, i.e., formalization of possible actions over a data cube. A thorough overview of such approaches is given in [18] where authors argue that an OLAP algebra needs to be closed (i.e., each operation produces a data cube), minimal (i.e., no operation can be expressed in terms of other operations), and complete (i.e., covering all the relevant operation). Thus, the algebras mainly focus on the correctness and satisfiability of queries in terms of underlying MD schemata. Furthermore, considering [18] as a backbone state of the art, the authors of [16] propose an approach for representing MD queries at the conceptual abstraction level by defining them as OCL constraints over UML schema representing data cube. Such a solution is a step towards interoperability of different platform specific models. However, the focus still remains on the validity of queries. On the other hand, our approach focuses on representing queries as metadata to support their sharing and reuse by means of SW technologies. As discussed in Sect. 2, *SM4MQ* is based on the OLAP algebra proposed in [8,9].

Furthermore, different OLAP query recommendation approaches typically store queries in logs and focus on their processing. According to a comprehensive overview found in [3], query in this context are represented with their syntactic (SQL) representation, resulting data, as vectors of features, sets of query fragments, or as graphs. The vector representation is used in several approaches, as for example in [6]. Moreover, some other approaches like [5] again use algebraic query representation. However, query representations are typically system specific that again hinders their sharing and reuse.

The modeling and representing of queries on the SW is just in its infancy. Just recently, in [19] the authors proposed an RDF-based model for representing SPARQL queries. They have used their model to represent a portion of queries over DBpedia and other public SPARQL endpoints with the following suggested use cases: generation of benchmarks, query feature analysis, query cashing, usability analysis, and meta-querying. Another interesting use case that can be added to the previous ones is the user assistance (e.g., query recommendations). Moreover, [11,15] propose vocabularies to represent SPARQL and SQL queries in RDF, respectively. Thus, there is a movement towards opening not only data but also metadata such as queries so that they could be explored with SPARQL. However, although needed for the context of Exploratory OLAP [1], an MD query model is still missing. By now, most of the efforts have been devoted to the schema modeling with vocabularies such as QB4OLAP (see [24]). Thus, the present paper proposes an RDF-based model to support sharing and reuse of MD queries on the SW that as well facilitates their exploitation.

## 7 Conclusion and Future Work

We have proposed *SM4MQ*, a Semantic Model for Multidimensional Queries. Using RDF and MD semantics, *SM4MQ* is a step towards Exploratory OLAP and sharing and reuse of MD queries on the SW. We also proposed a method to automate the transformation of the SM4MQ queries into other analysis-ready representations via SPARQL. The method is exemplified with the use case of transforming *SM4MQ* into a vector representation. To evaluate our approach, we have developed a prototype implementing the method for the vector use case. The evaluation showed that *SM4MQ* supports automation of transformation tasks that would otherwise require significant manual efforts (e.g., at least 100 times more just for SPARQL queries). In our future work, we plan to work on the exploitation side of the queries, e.g., develop richer transformations to support advanced user support techniques such as [4]. We also plan to apply our method to other use cases and support analytical feature definition via high-level GUIs.

**Acknowledgments.** This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate IT4BI-DC and it has been partially supported by the Secretaria d'Universitats i Recerca de la Generalitat de Catalunya, the grant number 2014 SGR-1534.

## References

1. Abelló, A., et al.: Using semantic web technologies for exploratory OLAP: a survey. IEEE Trans. Knowl. Data Eng. **27**(2), 571–588 (2015)
2. Aggarwal, C.C.: Recommender Systems - The Textbook. Springer, New York (2016)
3. Aligon, J., et al.: Similarity measures for OLAP sessions. Knowl. Inf. Syst. **39**(2), 463–489 (2014)
4. Aligon, J., et al.: A collaborative filtering approach for recommending OLAP sessions. Decis. Support Syst. **69**, 20–30 (2015)
5. Aufaure, M.-A., Kuchmann-Beauger, N., Marcel, P., Rizzi, S., Vanrompay, Y.: Predicting your next OLAP query based on recent analytical sessions. In: Bellatreche, L., Mohania, M.K. (eds.) DaWaK 2013. LNCS, vol. 8057, pp. 134–145. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40131-2_12
6. Chatzopoulou, G., et al.: The QueRIE system for personalized query recommendations. IEEE Data Eng. Bull. **34**(2), 55–60 (2011)
7. Cyganiak, R., et al.: Resource description framework (RDF): concepts and abstract syntax (2014). http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
8. de Aguiar Ciferri, C.D., et al.: Cube algebra: a generic user-centric model and query language for OLAP cubes. IJDWM **9**(2), 39–65 (2013)
9. Etcheverry, L., et al.: Modeling and querying data cubes on the semantic web. CoRR, abs/1512.06080 (2015)
10. Etcheverry, L., Vaisman, A.A.: Querying semantic web data cubes. In: AMW (2016)
11. Follenfant, C., Corby, O.: SQL abstract syntax trees vocabulary (2014). http://ns.inria.fr/ast/sql/index.html

12. Golfarelli, M., et al.: The dimensional fact model: a conceptual model for data warehouses. Int. J. Coop. Inf. Syst. **7**(2–3), 215–247 (1998)
13. Heath, T., Bizer, C., Data, L.: Evolving the Web into a Global Data Space. Morgan & Claypool Publishers, Seattle (2011)
14. Jensen, C.S., et al.: Multidimensional Databases and Data Warehousing. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, Seattle (2010)
15. Knublauch, H.: SPIN - SPARQL syntax (2013). http://www.spinrdf.org/sp.html
16. Pardillo, J., et al.: Extending OCL for OLAP querying on conceptual multidimensional models of data warehouses. Inf. Sci. **180**(5), 584–601 (2010)
17. Prud'hommeaux, E., Seaborne, A.: SPARQL 1.1 Query Language for RDF (2011)
18. Romero, O., Abelló, A.: On the need of a reference algebra for OLAP. In: Song, I.Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2007. LNCS, vol. 4654, pp. 99–110. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74553-2_10
19. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.-C.N.: LSQ: the linked SPARQL queries dataset. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 261–269. Springer, Cham (2015). doi:10.1007/978-3-319-25010-6_15
20. Strange, K.: ETL was the key to this data warehouse's success. Gartner Research, CS-15-3143, 3 (2002)
21. Varga, J.: SM4MQ materials (2016). http://www.essi.upc.edu/~jvarga/sm4mq-page.html
22. Varga, J., et al.: SM4AM: a semantic metamodel for analytical metadata. In: DOLAP, pp. 57–66 (2014)
23. Varga, J., Romero, O., Pedersen, T.B., Thomsen, C.: Towards next generation BI systems: the analytical metadata challenge. In: Bellatreche, L., Mohania, M.K. (eds.) DaWaK 2014. LNCS, vol. 8646, pp. 89–101. Springer, Cham (2014). doi:10.1007/978-3-319-10160-6_9
24. Varga, J., et al.: Dimensional enrichment of statistical linked open data. J. Web Sem. **40**, 22–51 (2016)
25. Varga, J., et al.: QB2OLAP: enabling OLAP on statistical linked open data. In: ICDE, pp. 1346–1349 (2016)