Enhancing OLAP Analysis with Web Cubes

Lorena Etcheverry[1] and Alejandro A. Vaisman[2]

[1] Instituto de Computación, Universidad de la República, Uruguay
lorenae@fing.edu.uy
[2] Université Libre de Bruxelles
avaisman@ulb.ac.be

Abstract. Traditional OLAP tools have proven to be successful in analyzing large sets of enterprise data. For today's business dynamics, sometimes these highly curated data is not enough. External data (particularly web data), may be useful to enhance local analysis. In this paper we discuss the extraction of multidimensional data from web sources, and their representation in RDFS. We introduce Open Cubes, an RDFS vocabulary for the specification and publication of multidimensional cubes on the Semantic Web, and show how classical OLAP operations can be implemented over Open Cubes using SPARQL 1.1, without the need of mapping the multidimensional information to the local database (the usual approach to multidimensional analysis of Semantic Web data). We show that our approach is plausible for the data sizes that can usually be retrieved to enhance local data repositories.

1 Introduction

Business intelligence (BI) comprises a collection of techniques used for extracting and analyzing business data, to support decision-making. Decision-support systems (DSS) include a broad spectrum of analysis capabilities, from simple reports to sophisticated analytics. These applications include On-Line Analytical Processing (OLAP) [9], a set of tools and algorithms for querying large multidimensional databases usually called data warehouses (DW). Data in a DW come from heterogeneous and distributed operational sources, and go through a process, denoted ETL (standing for Extraction, Transformation, and Loading). In OLAP, data are usually perceived as a *cube*. Each cell in this data cube contains a measure or set of measures representing facts and contextual information (the latter called *dimensions*). For some data-analysis tasks (e.g., worldwide price evolution of a certain product), the data contained the DSS do not suffice. External data sources (like the web) can provide useful multidimensional information, although usually too volatile to be permanently stored in the DW. We now present, through a use case, the research problems that appear in this scenario, and our approach for a solution to some of them.

1.1 Motivation and Problem Statement

A new electronics retail store is running a promotional campaign to improve sales on an specific segment of the digital cameras market: amateur digital SLR

				Time							
				Q3-2011							
				November 2011				December 2011			
Product			**Locat.**	*profit*	*#sales*	*unitPrice*	*unitCost*	*profit*	*#sales*	*unitPrice*	*unitCost*
Canon	T3i	Kit 18-55	NJ	870	10	870	783	736	8	875	783
			NY	1044	12	870	783	609	7	870	783
		Body only	NJ	340	4	850	765	375	5	840	765
			NY	425	5	850	765	300	4	840	765
			WA	1020	12	850	765	510	6	850	765
	T3	Kit 18-55	CA	780	13	460	400	480	8	460	400
			NJ	1200	15	480	400	560	7	480	400
Nikon	D3100	Kit 18-55	CA	630	10	610	547	945	15	610	547
			NY	732	12	608	547	366	6	608	547
			WA	340	5	615	547	189	3	610	547
		Kit 55-200	NY	750	6	725	600	1500	12	725	600
		Body only	CA	400	8	500	450	250	5	500	450
			NY	385	7	505	450	330	6	505	450
	D5100	Kit 18-55	NJ	1215	15	810	729	688	8	815	729
		Body only	CA	456	6	746	670	456	6	746	670

Fig. 1. A Sales Data Cube

cameras. The company sales products from several manufacturers and wants to find out candidates for "best deals" kind of offer. In today's business, web information is crucial for this. Price policies must take into account current deals found on the Internet (e.g., number of available offers, shipping policies, expected delivery time), as well as user opinions and product features. Jane, the data analyst of the company manually searches the web, querying different sites, and building spreadsheets with the collected data. Then she analizes local data at the DSS, together with data from web sources. This procedure is not only inefficient but also imprecise. Jane needs flexible and intelligent tools to get an idea of what is being offered on the web. We propose to make Jane's work more productive, by semi-automatically extracting multidimensional information from web data sources. This process produces what we denote **web cubes**, which can then be related to local OLAP cubes through a set of operators that we study in this paper. A key assumption is that web cubes only *add* knowledge for decision-making, and are not aimed at replacing precise information obtained from traditional DSS. Thus, we do not need these data to be complete, not even perfectly sound: Jane only needs a "few good answers" [17] to enhance her analysis, and our approach takes this into account. In a nutshell, web cubes are data cubes (obtained from web data sources) expressed using an RDF [10] vocabulary. We show through an use case how web cubes could be used to enhance existing DSS.

The case starts with Jane using her **local** DSS to analyze sales of digital cameras. From local cubes she produces a multidimensional report (Figure 1), actually a data cube with dimensions Product, Geography, and Time, and measures profit, #sales, unitPrice, and unitCost. From the report, Jane identifies that the sales of Canon T3 and T3i cameras have dropped in December. She conjectures that probably these products are being offered on the web at better prices, thus she decides to build a web cube to retrieve information about offers of these camera models. To start the process of building web cubes, Jane specifies her information requirements, which in this particular case are: price, delivery time and shipping costs of new Canon T3 and T3i DSLR cameras. She will try to obtain sales facts with these three measures, if possible with the same dimensions than the local cube. We assume that there is a catalogue of web data sources, with metadata that allows deciding which sources are going to be queried, the query mechanisms available for each source, and the format of the results.

Web data are available in many formats. Each one of these formats can be accessed using different mechanisms. For example RDF data can be published via SPARQL [16] endpoints, or extracted from HTML pages that publish RDFa [1] (among other formats), while XML data may be the result of querying RESTful web services (also known as web APIs). Tabular data may be extracted from HTML tables or retrieved from data sharing platforms, such as Google Fusion Tables[1]. In this paper we do not deal with the problem of retrieving web cubes. Well-known Information Retrieval and Natural Language Processing techniques can be used for this. For integrating the information retrieved from the data sources, and for representing the web cubes that are built after data extraction, we propose to use RDF as the data model. For the latter task, we devised a vocabulary called *Open Cubes* that we present in Section 3. It is highly possible that not all of Jane's requirements can be satisfied. For example, data could be obtained at an aggregation level different from the requested one. Or may be incomplete. We do not deal with these issues in this paper. Continuing with our use case, let us suppose that from `www.overstock.com`, Jane obtaines the following following RDF triples.

```
@prefix dc: <http://purl.org/dc>
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix schema: <http://schema.org/>

<http://www.overstock.com/.../5700610/product.html> dc:title
''Canon EOS Rebel T3i 18 D55mm IS II Digital SLR Camera Kit Overstock'' .
_:node16bt8fdb2x1 rdf:type schema:Product .
_:node16bt8fdb2x1 schema:name
''Canon EOS Rebel T3i 18 D55mm IS II Digital SLR Camera Kit''.
_:node16bt8fdb2x1 schema:offers _:node16bt8fdb2x2 .
_:node16bt8fdb2x2 rdf:type schema:Offer .
_:node16bt8fdb2x2 schema:price ''USD 850.82'' .
```

From these triples, a web cube is built (represented using the Open Cube vocabulary). Figure 2 shows this cube, in report format. Note that in this example, the new cube has the same dimensions than the cube in Figure 1, but different measures. Also notice that we have maximized the number of returned results, leading to the presence of null values (denoted by '-' in Figure 2), which should be replaced by appropriate values (e.g., 'unknown state') to guarantee the correctness of the results when performing OLAP operations. Jane now wants to compare the price of each product in the local cube, with the price for the same product in the web cube. This requires using OLAP operators like roll-up, slice, dice, or drill-across, *over web Cubes.* Jane then realizes that in the Geography dimension of the web cube, data are presented per city instead of per state (which is the case in the local cube). Thus, she needs to transform the web cube to the same level of detail as the local cube, taking both cubes to the country level, using a roll-up operation in the Geography dimension. After this, both cubes can be merged, and Jane can compare the prices in the local cube with those found

[1] `http://www.google.com/fusiontables/Home/`

							Time			
							Q3-2011			
							December 2011			
Product				**Geography**			*unitPrice*	*deliveryTime*	*shippingCost*	
SLR Camera	Canon	T3i	Kit 18-55	USA	-	-	850.82	10	0	(1)
					NY	Amityville	799.95	-	19.95	(2)
					-	-	760.00	5	0	(3)
			Body only	USA	-	-	672.99	-	0	(4)
		T3	Kit 18-55	USA	NJ	Somerset	466.82	-	-	(5)
					-	-	476.99	7	0	(6)

Fig. 2. A web cube in report format

				Time					
				Q3-2011					
				December 2011					
Product			**Geography**	*profit*	*#sales*	*unitPrice*	*unitCost*	*deliveryTime*	*shippingCost*
Canon	T3i	Kit 18-55	USA	672.5	15	872.5	783.0	-	-
				-	-	803.6		7.5	19.95
		Body only	USA	395.0	15	843.3	765.0	-	-
				-	-	673.0		-	0.0
	T3	Kit 18-55	USA	520.0	15	470.0	400.0	-	-
				-	-	471.9		7.0	0.0
Nikon	D3100	Kit 18-55	USA	500.0	24	609.3	547.0	-	-
		Kit 55-200	USA	1500	12	725	600	-	-
		Body only	USA	290.0	11	502.5	450.0	-	-
	D5100	Kit 18-55	USA	688	8	815	729	-	-
		Body only	USA	456	6	746	670	-	-

Fig. 3. Results of merging local and web cubes

on the Internet (grey rows). Figure 3 shows the result (Section 4 shows how web cubes can be mapped to the multidimensional model of the local cube).

Contributions. The following research questions arise in the scenario described above: Is it possible to use web data to enhance local OLAP analysis, without the burden of incorporating data sources and data requirements into the existent DSS life-cycle? What definitions, data-models, and query mechanisms are needed to accomplish these tasks? Our main goal is to start giving answers to the some of these questions. Central to this goal is the representation and querying of multidimensional data over the Semantic Web. Therefore, our main contributions are: (a) We introduce Open Cubes, a vocabulary specified using RDFS that allows representing the schema and instances of OLAP cubes, which extends and makes workable other similar proposals, since it is not only devised for data publishing, but for operating over RDF representation of multidimensional data as well (Section 3); (b) We show how typical OLAP operators can be expressed in SPARQL 1.1 using the vocabulary introduced in (a). We give an algorithm for generating SPARQL 1.1. `CONSTRUCT` queries for the OLAP operators, and show that implementing these operators is feasible. The basic assumption here is that web cubes are composed of a limited number of instances (triples) of interest (Section 4); (c) We sketch a mapping from a web cube to the multidimensional (in what follows, MD) model, in order to be able to operate with the local cubes. We do this through an example that shows how web cubes can be exported to the local system, using the Mondrian OLAP server[2] (Section 4).

[2] `http://mondrian.pentaho.com/documentation/schema.php`

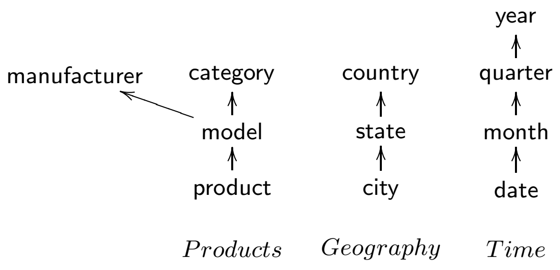
2 Preliminaries

RDF and SPARQL. The Resource Description Framework (RDF) [10] is a data model for expressing assertions over resources identified by an universal resource identifier (URI). Assertions are expressed as triples *subject - predicate - object*, where *subject* are always resources, and *predicate* and *object* could be resources or strings. *Blank nodes* are used to represent anonymous resources or resources without an URI, typically with a structural function, e.g., to group a set of statements. Data values in RDF are called *literals* and can only be *objects*. A set of RDF triples or *RDF dataset* can be seen as a directed graph where *subject* and *object* are nodes, and *predicates* are arcs. Many formats for RDF serialization exist. The examples presented in this paper use Turtle [2]. RDF Schema (RDFS) [3] is a particular RDF vocabulary where a set of reserved words can be used to describe properties like attributes of resources, and to represent relationships between resources. Some of these reserved words are rdfs:range `[range]`, rdfs:domain `[dom]`, rdf:type `[type]`, rdfs:subClassOf `[sc]`, and rdfs:subPropertyOf `[sp]`.

SPARQL is the W3C standard query language for RDF [16]. The query evaluation mechanism of SPARQL is based on subgraph matching: RDF triples are interpreted as nodes and edges of directed graphs, and the query graph is matched to the data graph, instantiating the variables in the query graph definition. The selection criteria is expressed as a graph pattern in the `WHERE` clause, consisting basically in a set of triple patterns connected by the '.' operator. The SPARQL 1.1 specification [5], with status of working draft at the moment of writing this paper, extends the power of SPARQL in many ways. Particularly relevant to our work is the support of aggregate functions and the inclusion of the `GROUP BY` clause.

OLAP. In OLAP, data are organized as hypercubes whose axes are *dimensions*. Each point in this multidimensional space is mapped through *facts* into one or more spaces of *measures*. Dimensions are structured in *hierarchies* of *levels* that allow analysis at different levels of aggregation. The values in a dimension level are called *members*, which can also have properties or *attributes*. Members in a dimension level must have a corresponding member in the upper level in the hierarchy, and this correspondence is defined through so-called rollup functions. In our running example we have a cube with sales data. For each sale we have four measures: quantity of products sold, profit, price and cost per product (see Figure 1, which shows a cube in the form of a report). We have also three dimensions: Product, Geography (geographical location of the point of sale), and Time. Figure 4 shows a possible schema for each dimension, and for the sales facts. We can see that in dimension Geography, cities aggregate over states, and states over countries. A well-known set of operations are defined over cubes. We present (for clarity, rather informally) some of these operations next, following [9] and [19].

Roll-Up. Summarizes data at a higher level in the hierarchies of a dimension. Given a cube $\mathcal{C}$, a dimension $D \in \mathcal{C}$, such that d_l is the lowest level of D in $\mathcal{C}$; a dimension level $du \in D$ such that $d_l \preceq d_u$ (meaning that dl is lower than du



SALES[product, city, date] $\rightarrow$ [$profit$, $\#sales$, $unitPrice$, $unitCost$]

Fig. 4. Examples of dimensions (above) and fact schemas (below)

in the hierarchy of D), and a set of aggregate functions f, Roll-Up($\mathcal{C}, D, d_u, f$) returns a new cube $\mathcal{C}'$ whose dimensions and levels are the same than in $\mathcal{C}$, except for the lowest level of D in $\mathcal{C}'$, which becomes d_u. Measures in $\mathcal{C}'$ are the result of applying each function in f to the corresponding measures in C. Aggregation is performed according with the rollup function associated with the relation $d_l \preceq d_u$. As an example, consider the cube C in Figure 5a with dimensions Product and Time and measure qtySold. Figure 5b shows the result of Roll-Up(C, Time, month, SUM). (If the cube has more than one measure, and different aggregate funnctions f_i are applied, pairs (measure,f_i) must be specified.

Slice. Removes one dimension from a cube $\mathcal{C}$. Intuitively, given a cube C, a dimension $D \in \mathcal{C}$, and an aggregation function f, the result of applying the Slice operation is a new cube $\mathcal{C}' =$ Slice($\mathcal{C}, D, f$) whose dimensions are the ones of $\mathcal{C}$, except for D. Measures in $\mathcal{C}'$ are the result of applying the aggregation function f to the measure in $\mathcal{C}$. Figure 5c shows the result of Slice(C, Product, SUM). (The same as above holds if there is more than one measure in $\mathcal{C}$)).

Dice. Selects a subset of the instances of a cube. Intuitively, given a cube $\mathcal{C}$, a dimension $D \in \mathcal{C}$, and a formula σ over the levels in D, the operation Dice is a new cube $\mathcal{C}' =$ Dice($\mathcal{C}, D, \sigma$) whose dimensions are same as in C, and such that the elements in $\mathcal{C}$ are the ones that satisfy σ. Assuming that $date1 \leq date2 \leq date3$, Figure 5d shows the result of Dice(C, Time, $date > date1$).

3 OLAP Cubes Specification in RDF

We now introduce Open Cubes, a vocabulary specified using RDFS that allows representing the schema and instances of OLAP cubes. Recently, an RDF vocabulary called Data Cube [4] has been proposed that supports the exchange of statistical data according to the ISO standard SDMX. [3] Although OLAP and statistical databases are very similar concepts, they differ in the problems

[3] `http://sdmx.org/`

		prod1	prod2
month1	date1	1	3
	date2	3	2
month2	date3	4	3

(a) Cube C

	prod1	prod2
month1	4	5
month2	4	3

(b) $RollUp(C, Time, month, sum)$

month1	date1	4
	date2	5
month2	date3	7

(c) $Slice(C, Product, sum)$

		prod1	prod2
month1	date2	3	2
month2	date3	4	3

(d) $Dice(C, Time, date > date1)$

Fig. 5. Applying OLAP operations to a cube

they address [18]. In particular, the Data Cube vocabulary does not provide the means to perform OLAP operations over data. This and other issues will be further discussed in Section 5.

Open Cubes is based on the multidimensional data model presented in [6], whose main concepts are dimensions and facts [4].

Dimensions have a schema and a set of instances; the schema contains the name of the dimension, a set of *levels* and a *partial order* defined among them; a dimension level is described by **attributes**; a dimension instance contains a set of partial functions, called *roll-up functions*, that specify how *level members* are aggregated. **Facts** also have a schema and instances; the former contains the name of the fact, a set of levels and a set of *measures*; the latter is a partial function that maps points of the schema into measure values. Figure 6 graphically presents the most relevant concepts in the Open Cubes vocabulary, where bold nodes represent classes and regular nodes represent properties. Labelled directed arcs between nodes represent properties with a defined domain and range among concepts in the vocabulary. We omit properties whose range is a literal value.

In Open Cubes, the class `oc:Dimension` and a set of related levels, modelled by the `oc:Level` property, represent dimension schemas. A partial order among levels is defined using properties `oc:parentLevel` and `oc:childLevel`, while the attributes of each level member are modelled using the `oc:Attribute` property. A fact schema is represented by the class `oc:FactSchema` and a set of levels and measures, which are modelled using the properties `oc:Level` and `oc:Measure` respectively. For each measure the aggregation function that can be used to compute the aggregated value of the measure, can be stated using the `oc:hasAggFunction` property. As an example, Figure 8 shows RDF triples (in Turtle notation) that represent the schemas of the Products dimension and the Sales fact from the report in Figure 7, using the Open Cubes vocabulary. The prefixes `oc` and `eg` represent the Open Cube vocabulary and the URI of the RDF graph of this example, respectively.

Dimension instances are modelled using a set of level members, which are represented by the `oc:LevelMember` class. The properties `oc:parentLevelMember`

[4] We could have used a more complex model, like [7]. However, the chosen model is expressive enough to capture the most usual OLAP features.

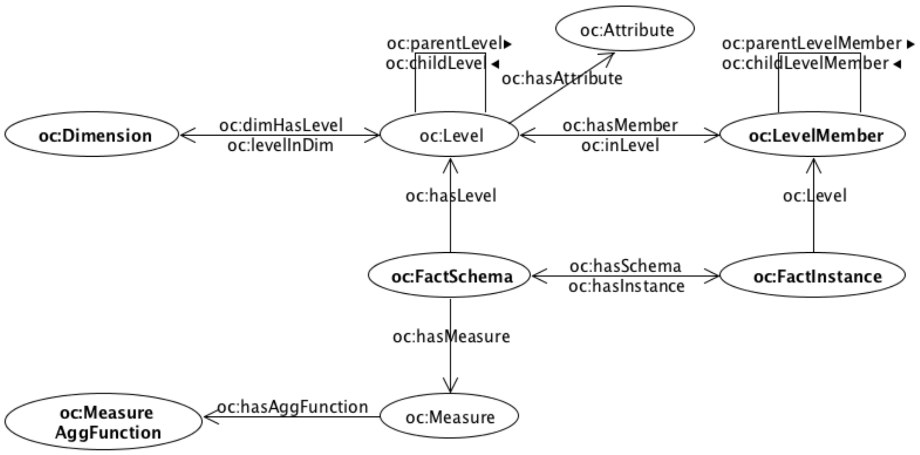


Fig. 6. Open Cubes vocabulary: classes and properties

							Time			
							Q3-2011			
							December 2011			
							date1		date2	
Product				**Geography**			price	qtySold	price	qtySold
DSLR Camera	Canon	T3i	Kit 18-55	USA	OR	Portland	714.54	2	714.54	3
		T3	Kit 18-55	USA	NJ	Somerset	466.82	5	466.82	5
						Jersey City	480	4	480	3

Fig. 7. Sample sales fact instances

```
eg:products rdf:type oc:Dimension;
    oc:dimHasLevel eg:product;
    oc:dimHasLevel eg:model;
    oc:dimHasLevel eg:manufacturer;
    oc:dimHasLevel eg:category.

eg:product rdf:type oc:Level.
eg:model rdf:type oc:Level.
eg:manufacturer rdf:type oc:Level.
eg:category rdf:type oc:Level.
eg:product oc:parentLevel eg:model.
eg:model oc:parentLevel
                                oc:manufacturer.
eg:model oc:parentLevel eg:category.
```

Products dimension schema

```
eg:sales rdf:type oc:FactSchema;
 oc:hasLevel eg:product;
 oc:hasLevel eg:city;
 oc:hasLevel eg:date;
 oc:hasMeasure eg:price;
 oc:hasMeasure eg:qtySold;

eg:price rdf:type oc:Measure;
 oc:hasAggFunction avg.

eg:qtySold rdf:type oc:Measure;
 oc:hasAggFunction sum.
```

Sales fact schema

Fig. 8. Expressing schemas using Open Cubes vocabulary

and `oc:childLevelMember` represent the roll-up functions that specify the navigation among level members. Instances of `oc:FactInstance` class represent fact instances. The set of level members and the values of each measure are related to the fact instance using pre-defined properties of type `Level` and `Measure`

```
‘t3i kit 18−55’ oc:inLevel
    eg:product;
    oc:parentLevelMember ‘t3i’.
‘t3i’ oc:inLevel eg:model;
    oc:parentLevelMember ‘Canon’;
    oc:parentLevelMember ‘DSLR camera’.

‘t3 kit 18−55’ oc:inLevel eg:product;
    oc:parentLevelMember ‘t3’.

‘t3’ oc:inLevel eg:model;
    oc:parentLevelMember ‘Canon’;
    oc:parentLevelMember ‘DSLR camera’.

‘Canon’ oc:inLevel eg:manufacturer.

‘DSLR camera’ oc:inLevel eg:category.
```

Products dimension instances

```
eg:sales_i1 rdf:type
            oc:FactInstance ;
    oc:hasSchema eg:sales ;
    eg:product ‘t3i kit 18−55’ ;
    eg:date ‘date1’ ;
    eg:city ‘Portland’ ;
    eg:price ‘‘714.54’’^^xsd:decimal;
    eg:qtySold ‘‘3’’^^xsd:integer .

eg:sales_i2 rdf:type
            oc:FactInstance ;
    ....
eg:sales_i3 rdf:type
            oc:FactInstance ;
    ....
```

Sales fact instances

Fig. 9. Expressing instances using Open Cubes vocabulary

respectively. For example, Figure 9 shows how the instances in Figure 7 can be represented using the Open Cubes vocabulary. Is it worth noting that subjects in RDF triples should be either blank nodes or URIs. For clarity, we use constant values between quotation marks to represent the identifier of each level member. These constant values should be replaced by URIs that uniquely identify each of the level members. The `oc:hasFactId` property allows to provide a literal that uniquely identifies each fact instance within a collection of cubes or multidimensional database. Due to space reasons we omit the `oc:childLevelMember` relationships between the level members, and only show one complete fact instance RDF representation (the first tuple in Figure 7) .

4 Operating with Web Cubes

The operators introduced in Section 2 can be implemented in SPARQL 1.1 as we show next. Let us start with a couple of examples.

Roll-up: This operation encompasses two tasks: (i) creating the schema of the new cube; (ii) populating it. For instance, consider the Sales cube representation in Open Cubes (Figures 8 and 9), and a new cube SalesByMonth = Roll-Up($Sales, Time, month, price$, AVG, qty, SUM). Figure 10 shows the specification in Open Cubes of the SalesByMonth schema and a SPARQL 1.1 query that populates it. It is important to remark that new IRIs must be generated to identify each of the fact instances.

Slice: Slicing out the Geography dimension from the Sales cube of in Figures 8 and 9, returns a new cube SalesWithoutGeo that satisfies SalesWithoutGeo = $Slice(Sales, Geography$, AVG, SUM). Figure 11 shows the Open Cubes specification of the SalesWithoutGeo schema, and a SPARQL 1.1 query that populates it.

4.1 A General Algorithm for Roll-Up over Open Cubes

We now show an algorithm for the Roll-up operation, probably the most used one in the OLAP setting. We define the following functions: (a) $newVar(i)$

generates unique SPARQL variable names; (b) *value*(v) returns the value stored in variable v; (c) *levels*(s) returns all the levels in a schema s (i.e., all the values of ?l that satisfy s `oc:hasLevel` ?l); (d) *measures*(s) returns all the measures in a schema s (all the values of ?m that satisfy s `oc:hasMeasure` ?m); and (e) *aggFunction*(m) returns the aggregation function of measure m (all the values of ?f that satisfy m `oc:hasAggFunction` ?f). Also assume that there is a level dl_o in dimension d such that there exists a path between dl_o and dl_r which contains only arcs with type `oc:parentLevel`. The function $levelsPath(d_1, d_2)$ retrieves the ordered list of levels in the path between d_1 and d_2 (including both levels). Also assume that it is possible to access and modify different parts of a SPARQL query via the properties: *resultFormat*, *graphPattern*, and *groupBy*, among others. The *add*(s) function appends s to a particular part of the query.

```
eg:salesMonth rdf:type oc:FactSchema;
     oc:hasLevel eg:product; oc:hasLevel eg:city;
     oc:hasLevel eg:month; oc:hasMeasure eg:price;
     oc:hasMeasure eg:qtySold .
```

SalesByMonth schema

```
CONSTRUCT { ?id oc:hasSchema eg:salesMonth . ?id eg:product ?prod .
   ?id eg:city ?city . ?id eg:month ?mon .
   ?id eg:price ?priceMonth . ?id eg:qtySold ?qtyMonth .}
WHERE{ {
SELECT ?prod ?city ?mon  (AVG(?price) AS ?priceMonth)
(SUM(?qtySold) AS ?qtyMonth)
(iri(fn:concat("http://example.org/salesInstances#", "sales","_",
 fn:substring-after(?prod,"http://example.org/salesInstances#"),"_",
 fn:substring-after(?city,"http://example.org/salesInstances#"),"_",
 fn:substring-after(?mon,"http://example.org/salesInstances#"))) AS ?id)
    WHERE {
         ?i oc:hasSchema sl:sales . ?i eg:product ?prod .
         ?i eg:city ?city . ?i eg:date ?date .
         ?i eg:price ?price . ?i eg:qtySold ?qty .
         ?date oc:parentLevelMember ?mon . ?mon oc:inLevel eg:month
   }GROUP BY ?prod ?city ?mon}}
```

SalesByMonth instances

Fig. 10. RollUp implementation over Open Cubes

```
eg:salesWithoutGeo rdf:type oc:FactSchema;
     oc:hasLevel eg:date; oc:hasLevel eg:city;
     oc:hasMeasure eg:price; oc:hasMeasure eg:qtySold.
```

SalesWithoutGeo schema

```
CONSTRUCT
{ ?id oc:hasSchema eg:salesWithoutGeo . ?id eg:city ?city .
   ?id eg:date ?date . ?id eg:price ?avgPrice .
   ?id eg:qtySold ?sumQtySold .
} WHERE {{
 SELECT ?city ?date (AVG(?price) AS ?avgPrice)
  (SUM(?qty) AS ?sumQtySold)
  (iri(fn:concat("http://example.org/salesInstances#", "salesSGeo","_",
 fn:substring-after(?city,"http://example.org/salesInstances#"),"_",
 fn:substring-after(?date,"http://example.org/salesInstances#"))) AS ?id)
  WHERE {
         ?i oc:hasSchema eg:sales .
         ?i eg:city ?city  .?i eg:date ?date .
         ?i eg:price ?price . ?i eg:qtySold ?qty .
   }GROUP BY ?city ?date }}
```

SalesWithoutGeo instances

Fig. 11. Slice implementation over Open Cubes

Algorithm 1 shows a general procedure to build the SPARQL query needed to populate a cube $\mathcal{C}' = \text{Roll-up}(\mathcal{C}, D, dl_r, F)$, according to the definitions of Section 2. Two SPARQL queries are needed: an inner query *qInner* that performs the GROUP BY, and an outer query *qOuter* that builds triples based on the retrieved values from the inner query. The algorithm incrementally builds both queries simultaneously, using the *add* function. Line 1 states that generated facts have s_r as schema. Line 2 adds a triple to the WHERE clause of the inner query, that allows selecting facts from schema s_o. Lines 3 through 11 project the members of each level in the schema into the result of both queries, also adding triples to the WHERE clause of the inner query and adding the variables that represent the level members to the GROUP BY clause in the inner query. Lines 12 through 18 do the same for measures. In lines 13 and 16 f represents the SPARQL function corresponding to the aggregate function for each measure and $f(value(m_i))$ is the string that should be included to calculate the aggregated value (e.g $\text{SUM}(?m)$ if $value(m_i) = ?m$). Lines 19 to 30 add the triples needed to navigate the dimension hierarchy. Line 22 adds to the inner query a triple that associates the level member with the fact instance, line 25 adds a triple that allows to check to which level the level member belongs to, and line 27 retrieves the parent level member of the current level. Line 30 adds the target level to the GROUP BY clause. Finally, line 33 sets the inner query as the WHERE clause of the outer query, which is returned in line 34. For clarity, we omitted the clause that generates the expression that binds variable ?id to a dynamically generated URI from the values in the fact.

Preliminary Results. We have implemented the roll-up operation. We generated over 24000 triples from synthetic data that represent instances of 3000 facts and dimension members. Those triples were loaded in Virtuoso Opensource 6.1.4[5]. SPARQL 1.1 queries that implement the roll-up operation were performed over the triples, retrieving results in less than 0.1 seconds. Our assumption is that web cubes will not be large, since they will contain very focused an current data from selected sources. Then, the operators over web cubes will be useful to avoid going back and forth from the local multidimensional representation.

4.2 Exporting Web Cubes to a Local DSS

We now sketch how web cubes can be exported into the Mondrian OLAP server. Multidimensional schemas in Mondrian are specified via an XML file, which also contains the directives that allow populating the schemas.

The general mechanism has two phases: structure definition and population. The definition phase takes a web cube schema as input and produces two outputs: (i) the SQL code that creates tables in a relational database[6], and (ii) an XML file that contains the schema definition of the cube, and the mappings that allow to populate the multidimensional schema with data stored in the relational database. The population phase takes as input a web cube instance, expressed

[5] `http://www.openlinksw.com/wiki/main/Main`

[6] Mondrian represents the cube in the relational model.

using Open Cubes, and produces SQL code that loads data into the database created in the definition phase. Figure 12 shows a portion of the XML file that represents the cube shown in Figure 2, closing the cycle of our running example: Jane requested the web cubes, which were retrieved, and represented in RDF using Open Cubes vocabulary; then she operated over the RDF representation of the web cube, and finally imported it to the local DSS, for joint analysis with the local cube.

Algorithm 1. Generates the SPARQL query that builds the Roll-Up instances

Input: s_o original schema, s_r new schema, dl_o level of $D \in s_o$, dl_r level of $D \in s_r$
Output: *qOuter* is a SPARQL CONSTRUCT query that creates the roll-up instances

```
 1: qOuter.graphPattern.add(?id , oc:hasSchema, $s_r$)
 2: qInner.graphPattern.add(?i , oc:hasSchema, $s_o$)
 3: for all $l \in L = levels(s_o)$ do
 4:    if $l \neq dl_o$ then
 5:       newVar($l_i$)
 6:       qOuter.resultFormat.add(?id , l, value($l_i$))
 7:       qInner.resultFormat.add(value($l_i$))
 8:       qInner.graphPattern.add(?i , l, value($l_i$))
 9:       qInner.groupBy.add(value($l_i$))
10:    end if
11: end for
12: for all $m \in M = measures(s_o)$ do
13:    f = aggFunction(m)
14:    newVar($m_i$); newVar($ag_i$)
15:    qOuter.resultFormat.add(?id , m, value($ag_i$))
16:    qInner.resultFormat.add(f(value($m_i$)) AS $ag_i$)
17:    qInner.graphPattern.add(?i , m, value($m_i$))
18: end for
19: for all $dl_i \in path = levelsPath(dl_o, dl_r)$ do
20:    newVar($lm_i$)
21:    if $dl_i = dl_o$ then
22:       qInner.graphPattern.add(?i , value($dl_i$), value($lm_i$))
23:    else
24:       newVar($plm_i$)
25:       qInner.graphPattern.add(value($plm_i$), oc:inLevel, value($dl_i$))
26:       if $dl_i \neq dl_r$ then
27:          qInner.graphPattern.add(value($lm_i$), oc:parentLevelMember,value($plm_i$) )
28:       end if
29:    end if
30: end for
31: newVar($lm_i$)
32: qInner.groupBy.add($plm_i$)
33: qInner.resulFormat.add($plm_i$)
34: qOuter.resultFormat.add(?id , $dl_r$, value($plm_i$))
35: qOuter.graphPattern.set(qInner)
36: return qOuter
```

```xml
<Schema>
<Cube name="WCubeSales">
<Table name="wcube_sales_fact"/>
<Dimension name="Products" foreignKey="product_id">
<Hierarchy hasAll="false" primaryKey="product_id">
<Table name="products"/>
<Level name="Model" column="model_id" uniqueMembers="true"/>
<Level name="Manufacurer" column="manuf_id" uniqueMembers="true"/>
<Level name="Category" column="category_id" uniqueMembers="true"/>
</Hierarchy>
</Dimension>
<Dimension name="Time" foreignKey="date">
 ...
</Dimension>
<Measure name="Unit Price" column="unit_price" aggregator="avg" />
<Measure name="Delivery Time" column="del_time" aggregator="avg" />
<Measure name="Shipping Cost" column="ship_cost" aggregator="avg" />
</Cube>
</Schema>
```

Sales schema representation using Mondrian

Fig. 12. Exporting Web Cubes to a local DSS

5 Related Work

Our work is highly related with the idea of *situational BI*, a term coined in [11]. Situational BI focuses on executing complex queries, mainly natural language queries, over unstructured and (semi-) structured data; in particular, unstructured text retrieved from documents is seen as a primary source of information. In the context of situational BI the process of augmenting local data with data retrieved from web sources is discussed in [12].

In [4] a vocabulary called RDF Data Cube(DC) is presented. This vocabulary is focused on representing statistical data, according to the SDMX data model. Although this underlying data model shares some terms with traditional multidimensional data models, the semantics of some of the concepts are different. An example of this is the concept of *slices*. Slices, as defined in the DC vocabulary, represent subsets of observations, fixing values for one or more dimensions. Slices are not defined in terms of an existing cube, they are defined as new structures and new instances (observations). An example can be found in [4], Section 7. The semantics of the slice operator in the MD model is quite different, as shown in Section 2. Besides, while dimensions and its hierarchical nature are first class citizens in MD models, DC dimensions are flat concepts that allow to identify observations at a single granularity. The DC vocabulary does not provide the constructs to explicitly represent hierarchies within dimensions, neither at the schema level (DataStructureDefinition) nor at the instance level. As the DC vocabulary adheres to Linked Data principles hierarchies within dimensions may be inferred from external hierarchies, whenever possible. For example, members of a dimension stated to be of type `foaf:Person` can be grouped according to their place of work using the `foaf:workplaceHomepage` property. This is clearly not enough to guarantee the capability of the model to support OLAP operations as roll-up, which need to represent hierarchical relationships within dimensions levels and level members. Some of the problems found when trying to map cubes expressed in the DC vocabulary into a multidimensional model are discussed in [8]. In light of the above, we decided to buid a new vocabulary from scratch, instead of extending DC.

To our knowledge no previous work addresses the problem of expressing OLAP operators over multidimensional data structures expressed in RDF. Nevertheless the idea of using RDFS or OWL ontologies to represent multidimensional data structures has been already explored. There is a line of work that uses these kinds of ontologies as auxiliary artefacts in traditional DSS. In [15] the authors propose an OWL ontology that models OLAP cubes schemas to assist in the ETL process of a traditional DSS, while in [14] a variation of the former ontology is used to check the consistency of summarizations. Among the approaches to the problem of populating multidimensional schemas with semantic web data, in [13] a process that extracts facts from semantic web data sources is outlined, assuming that the multidimensional schema and the sources are annotated using a single ontology. This work, however, does not deal with the extraction of dimension hierarchies.

6 Conclusion and Open Problems

We have presented an scenario where DSS systems are enhanced with web cubes obtained from current data on the web. We defined a vocabulary to represent these multidimensional data in RDFS, and operations over this representation, to avoid exporting these cubes to a classic OLAP system. We also presented a general algorithm for creating the SPARQL 1.1 queries that implement the Roll-up operation. To the best of our knowledge, this is the first approach of this kind in the field of BI over the Semantic Web. Existing approaches are based on the idea of obtaining RDF data and exporting these data to traditional OLAP cubes. Future work includes developing the complete framework, which encompasses requirements specification, data acquisition, web cubes extraction and publication, and data analysis using web cubes.

Acknowledgement. This work has been partially funded by the LACCIR Project “Monitoring Protected Areas using an OLAP-enabled Spatio-Temporal Geographic Information System”. Alejandro Vaisman has been partially funded by the “Open Semantic Cloud for Brussels (OSCB)” Project, supported by the Brussels Capital Region, Belgium.

References

1. Adida, B., Birbeck, M.: RDFa Primer, Bridging the Human and Data Webs (2008), `http://www.w3.org/TR/xhtml-rdfa-primer/`
2. Beckett, D., Berners-Lee, T.: Turtle - Terse RDF Triple Language (2011), `http://www.w3.org/TeamSubmission/turtle/`
3. Brickley, D., Guha, R., McBride, B.: RDF Vocabulary Description Language 1.0: RDF Schema (2004), `http://www.w3.org/TR/rdf-schema/`
4. Cyganiak, R., Field, S., Gregory, A., Halb, W., Tennison, J.: Semantic Statistics: Bringing Together SDMX and SCOVO. In: Proc. of the WWW2010 Workshop on Linked Data on the Web, pp. 2–6. CEUR-WS.org (2010)
5. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language (2010), `http://www.w3.org/TR/sparql11-query/`

6. Hurtado, C.A., Mendelzon, A.O., Vaisman, A.A.: Maintaining Data Cubes under Dimension Updates. In: Proc. of the 15th International Conference on Data Engineering, ICDE 1999, pp. 346–355. IEEE Computer Society, Washington, DC (1999)
7. Hurtado, C.A., Gutiérrez, C., Mendelzon, A.O.: Capturing summarizability with integrity constraints in OLAP. ACM Transactions on Database Systems 30(3), 854–886 (2005)
8. Kämpgen, B., Harth, A.: Transforming statistical linked data for use in OLAP systems. In: Proc. of the 7th International Conference on Semantic Systems, I-Semantics 2011, New York, NY, USA, pp. 33–40 (2011)
9. Kimball, R.: The Data Warehouse Toolkit. J. Wiley and Sons (1996)
10. Klyne, G., Carroll, J.J., McBride, B.: Resource Description Framework (RDF): Concepts and Abstract Syntax (2004), http://www.w3.org/TR/rdf-concepts/
11. Löser, A., Hueske, F., Markl, V.: Situational Business Intelligence. In: Castellanos, M., Dayal, U., Sellis, T. (eds.) BIRTE 2008. LNBIP, vol. 27, pp. 1–11. Springer, Heidelberg (2009)
12. Löser, A., Nagel, C., Pieper, S.: Augmenting Tables by Self-supervised Web Search. In: Löser, A. (ed.) BIRTE 2010. LNBIP, vol. 84, pp. 84–99. Springer, Heidelberg (2011)
13. Nebot, V., Llavori, R.B.: Building data warehouses with semantic data. In: EDBT/ICDT Workshops. ACM International Conference Proceeding Series. ACM (2010)
14. Niemi, T., Niinimäki, M.: Ontologies and summarizability in OLAP. In: Proc. of the 2010 ACM Symposium on Applied Computing, SAC 2010, pp. 1349–1353. ACM, New York (2010)
15. Niinimäki, M., Niemi, T.: An ETL Process for OLAP Using RDF/OWL Ontologies. In: Spaccapietra, S., Zimányi, E., Song, I.-Y. (eds.) Journal on Data Semantics XIII. LNCS, vol. 5530, pp. 97–119. Springer, Heidelberg (2009)
16. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (2008), http://www.w3.org/TR/rdf-sparql-query/
17. Sequeda, J., Hartig, O.: Towards a query language for the web of data (a vision paper). CoRR, abs/1110.3017 (2011)
18. Shoshani, A.: OLAP and statistical databases: similarities and differences. In: PODS 1997, pp. 185–196. ACM, New York (1997)
19. Vassiliadis, P.: Modeling multidimensional databases, cubes and cube operations. In: SSDBM, pp. 53–62. IEEE Computer Society (1998)