Semantic Enrichment of OLAP Cubes: Multi-dimensional Ontologies and Their Representation in SQL and OWL

Bernd Neumayr, Christoph Schütz, and Michael Schrefl

Johannes Kepler University Linz, Austria
{neumayr,schrefl,schuetz}@dke.uni-linz.ac.at

Abstract. A multi-dimensional ontology (MDO) enriches an OLAP cube with concepts that represent business terms in the context of data analysis. The formal representation of the meaning of business terms facilitates the unambiguous interpretation of query results as well as the sharing of knowledge among business analysts. In contrast to traditional ontologies, an MDO captures the multi-dimensional, hierarchical world view of business analysts. In this paper, we introduce a translation of MDO concepts to SQL in order to allow for the querying of a closed-world OLAP cube. We introduce a representation in OWL in order to determine subsumption hierarchies of MDO concepts using off-the-shelf reasoners.

Keywords: Business Intelligence, OLAP, Data Warehouse, Knowledge Representation and Reasoning.

1 Introduction

An OLAP cube is typically deployed on top of the asserted data in a data warehouse and provides a coherent, multi-dimensional and multi-granular view of (a fragment of) the data in the data warehouse. Analysts query OLAP cubes in order to gain insights into their businesses. When many business analysts work together understandability, reuse, and maintainability of multi-dimensional queries become key challenges. These problems can be tackled by defining a shared vocabulary of business terms.

The web ontology language OWL is a first candidate for a formal definition language for shared vocabularies or ontologies. OWL comes with automated reasoning support for subsumption and satisfiability checking over ontologies. Such automated reasoning support has proven vital for maintaining the consistency of large, collaboratively-engineered ontologies. Using traditional ontology languages such as OWL it is, however, difficult to represent the multi-granular and multi-dimensional conceptualizations of business analysts where abstract points at various granularities in a multi-dimensional space represent and quantify parts of the domain of interest.

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 624–641, 2013.

The introduction of the multi-dimensional ontology language overcomes the shortcomings of traditional ontology languages in the context of data warehousing. A multi-dimensional ontology (MDO) captures the multi-granular and multi-dimensional world-view of business analysts; it unambiguously defines business terms and explains the calculation of measures. Thus, an MDO facilitates the selection of facts as well as the interpretation of query results.

For closed-world querying, an OLAP cube is enriched with SQL views representing the concepts of the ontology. First, the definition of relational views over the OLAP cube allows for querying using SQL. This approach is akin to cube and dimensional views provided by Oracle OLAP [1]. Second, every MDO concept is translated into a relational view over the cube views and the dimensional views. This translation of MDO concepts into relational views is platform-independent. For increased performance, platform-specific logical and physical optimizations may be applied, which is out of the scope of this paper.

A representation of MDO concepts in OWL allows for automated subsumption and satisfiability checking, using off-the-shelf reasoners. Disjointness of transitive roll-up relationships is a fundamental property of roll-up hierarchies and difficult to represent in OWL 2 DL since functionality cannot be defined upon transitive object properties. In this paper we present a solution for this problem.

The MDO has been developed as part of the *Semantic Cockpit* (semCockpit) project [13,12], a joint effort between academia, BI industry, and public health insurers. In addition to introducing the representation of MDO concepts in SQL and OWL, in this paper, we extend the MDO language with disjunction and complement of concepts. Due to these extensions, the Datalog-based approach [12] is insufficient.

The remainder of the paper is organized as follows. In Sect. 2 we state the problem and motivate our approach. In Sect. 3 we define the abstract data model for OLAP cubes. In Sect. 4 we present the definition of MDO concepts over OLAP Cubes and introduce a representation in OWL as well as transformations into relational views for querying. In Sect. 5 we present a proof-of-concept prototype and discuss implementation issues. In Sect. 6 we review related work.

2 Motivation

An OLAP cube is a multi-dimensional model representing real-world facts at various levels of aggregation. Facts are quantified by measures and identified by entities from multiple dimensions. The dimensions have hierarchically organized levels which allow for the analysis of measures at different granularities. Figure 1 illustrates a multi-dimensional model for the analysis of medical treatments. In this model, the recorded measure are the costs which are available by the day for a particular acting doctor and insurant. Along the dimensions Time, Doctor, and Patient a business analyst may roll up the data in order to obtain the costs for coarser time periods and groups of acting doctors and insurants. As non-dimensional attribute, the number of inhabitants provides additional information about the city and country of a doctor.

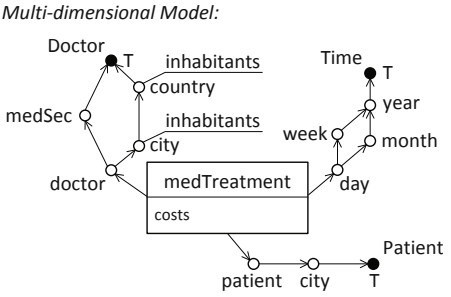

Fig. 1. A multi-dimensional model and its relational views for querying

Many data warehouse systems manage OLAP cubes in multi-dimensional data structures that are optimized for query performance. Some of these systems, for example, Oracle OLAP [1], provide relational views as a query interface for the business analyst. For example, the relational view representation of the multi-dimensional model in Fig. 1 comprises a fact view (medTreatment) which stores in column costs the costs at the most granular level. Measure view costs contains the costs at every level of granularity. The other columns (actDoc, time, insurant) of views medTreatment and costs reference dimension views (Doctor, Time, Patient) which associate entities of a dimension with a particular level. Another measure view (costsPerInsurant) abstracts from the Patient dimension and defines the costs per insurant as a derived measure for dimensions Doctor and Time at every level of granularity. Roll-up views (Doctor_rollup, Time_rollup, Patient_rollup) determine for each entity its superordinate entities, that is, the transitive and reflexive closure of the immediate parent entities. Entity views (CityE, CountryE) represent the non-dimensional attributes (inhabitants).

Working with the relational views over an OLAP cube can be cumbersome. Business analysts encode business terms directly into SQL queries. For example, the query in Fig. 2 retrieves the costs per insurant in the year 2010 by city of acting doctor where the acting doctor is in a big city of a small country. The analyst defines in the where-clause of the SQL query a big city as a city with

```
SELECT * FROM costsPerInsurant
WHERE actDoc IN (
  SELECT Doctor FROM Doctor_rollup
  WHERE Doctor_sup IN (
    SELECT CityE FROM CityE WHERE inhabitants>1000000 ) )
AND actDoc IN (
  SELECT Doctor FROM Doctor_rollup
  WHERE Doctor_sup IN (
    SELECT e_country FROM e_country WHERE inhabitants<20000000 ) )
AND actDoc IN (
  SELECT Doctor FROM Doctor WHERE Doctor_lvl = 'city' )
AND time = '2010'
```

Fig. 2. An SQL query over the views on the multi-dimensional model in Fig. 1

more than a million inhabitants and a small country as a country with less than twenty million inhabitants. For each query, the analyst must define the business terms from scratch.

The explicit definition of business terms in concept views allows for the reuse of business term definitions in several queries and facilitates the formulation of these queries. For example, the query in Fig. 3 references a concept view (actDocInBigCityAndSmallCountry) that already defines an acting doctor in a big city of a small country. In the reformulated query, a natural join with the concept view replaces the subqueries from the original query (Fig. 2). Consequently, when formulating the query, the analyst is freed of defining these business terms.

```
SELECT * FROM costsPerInsurant
  NATURAL JOIN actDocInBigCityAndSmallCountry
WHERE actDoc IN (
  SELECT Doctor FROM Doctor WHERE Doctor_lvl = 'city' )
AND time = '2010'
```

Fig. 3. A reformulation with concept views of the SQL query from Fig. 2

Besides their use in the formulation of queries, concept views also facilitate the definition of derived measures. A derived measure, as opposed to a base measure with asserted measure values, applies some measurement instruction to the available measure values. For example, a business analyst may derive the costs per rural insurant (Fig. 4) from the costs using the concept view that defines rural patients (ruralPatient).

```
CREATE VIEW costsPerRuralInsurant AS
SELECT AVG(mt.costs) AS costsPerRuralInsurant, d.sup AS Doctor, t.sup AS Time
FROM medTreatment mt NATURAL JOIN Doctor_rollup d
  NATURAL JOIN Time_rollup t NATURAL JOIN ruralPatient rp
GROUP BY d.sup, t.sup
```

Fig. 4. Definition of a derived measure using concept views

Rather than defining concept views in SQL, we propose a multi-dimensional ontology (MDO) language. The MDO language allows for an organization-wide coherent definition of business terms. In order to share definitions of business terms among business analysts there must be some sort of classification which facilitates the retrieval of concepts. Automated reasoners may organize the concepts in subsumption hierarchies. As a prerequisite for the use of existing, off-the-shelf automated reasoners, the MDO concepts must be formalized in a standard ontology language, for example, the web ontology language OWL. Thus, in this paper, we also provide a translation of MDO concepts into OWL which allows for their automated classification.

3 OLAP Cubes

In this section we describe the abstract data model of OLAP cubes together with a concrete notation and representations in OWL and SQL. The notion of *cube* is used differently: (a) for a set of facts at some granularity, (b) for a set of base and derived facts at different granularities (data cube), and (c) a set of related data cubes. We use the term OLAP cube in the third sense.

3.1 Abstract Data Model

The MDO language allows for the specification of schema and instance of OLAP cubes. Table 1 defines, in the left column, the concrete MDO syntax for the specification of an OLAP cube schema. Throughout the remainder of this paper, variables are *emphasized* and terminal symbols are written in sans-serif typeface. The schema of an OLAP cube consists of entity classes, dimensions, dimension roles, dimension spaces, and measures. An entity class (Row 1) defines a set of attributes $attr_1, \ldots, attr_n$ with data types $dt_1, \ldots, dt_n$. Attributes are also referred to as descriptors or non-dimensional attributes. A dimension (Row 2) consists of a set of levels organized in a level roll-up hierarchy with a single top level and a single bottom level. A level refers to an entity class with an entity class being referred to by at most one level per dimension. A level-range restricted dimension (Row 3) is defined over a dimension d and restricts the set of levels of d to those levels that are between or equal to a given from-level l' and a given to-level l''.

A dimension space is the multi-dimensional space that corresponds to the cartesian product of a list of dimensions. Within a dimension space, a dimension may play several dimension roles (Row 4). A dimension space (Row 5) is defined over a set of dimension roles $dr_1, \ldots, dr_n$ for level-range restricted dimensions $lrd_1, \ldots, lrd_n$. The level is the role that an entity class plays within a dimension. The overall set of dimension roles of an OLAP cube constitutes a universal dimension space. A measure (Row 6) is defined for a dimension space. The measure values depend functionally on the dimension roles in the dimension space.

Table 2 illustrates, in the left column, the definition of example fragments of an OLAP cube schema. Entity class CityE (Row 1) has an attribute inhabitants of data type integer. This entity class CityE is associated with the city level of the Doctor dimension (Row 2). In the hierarchy of the Doctor dimension, the city level is the superordinate level of doctor. The level-range restricted dimension Doctor_CityCountry (Row 3), the Doctor dimension to the levels from city up to country. The acting doctor (actDoc, Row 4) is a role of the doctor dimension. The actDoc role is used in dimension space ds1. In this dimension space, the actDoc role is defined for the restricted level range from city to country. The dimension space ds1 has a measure costs (Row 6).

An instance of an OLAP cube (Table 3) comprises entities and nodes. These individuals are described by their connections and properties. An entity (Row 1) is member of an entity class *ecl* and assigns data values to the attributes of the

Table 1. Representation of an OLAP cube schema in OWL and as relational views

	MDO	OWL	Relational
(1)	CREATE ENTITY CLASS ecl ($attr_1$ dt_1, ..., $attr_n$ dt_n);	$ecl \sqsubseteq$ Entity $\exists attr_1.\top \sqsubseteq ecl$ ... $\exists attr_n.\top \sqsubseteq ecl$ $\top \sqsubset \forall attr_1.dt$ ... $\top \sqsubset \forall attr_n.dt$ $\top \sqsubseteq \leqslant 1 attr_1$ ... $\top \sqsubseteq \leqslant 1 attr_n$	ecl(<u>ecl</u>, $attr_1$, ..., $attr_n$)
(2)	CREATE DIMENSION d WITH LEVELS l_1 ecl_1, ..., l_n ecl_n AND HIERARCHY l'_1 UNDER l''_1, ..., l'_m UNDER l''_m;	$d \sqsubseteq$ Node $\exists$atLevel.$\{l_1\} \equiv d \sqcap \exists$roleOf.$ecl_1$ ... $\exists$atLevel.$\{l_n\} \equiv d \sqcap \exists$roleOf.$ecl_n$ directlyRollsUpTo(l'_1, l''_1) ... directlyRollsUpTo(l'_m, l''_m)	d_lvl(<u>d_lvl</u>,entityclass) d_lvlparent(<u>d_lvl, d_lvl_sup</u>) d_lvlrollup(<u>d_lvl, d_lvl_sup</u>) d(<u>d, d_lvl</u>) d_parent(<u>d, d_sup</u>) d_rollup(<u>d, d_sup</u>) insert into d_lvl values (l_1,ecl_1); ... insert into d_lvl values (l_n,ecl_n); insert into d_lvlparent values (l'_1,l''_1); ... insert into d_lvlparent values (l'_m,l''_m);
(3)	CREATE LEVELRANGERESTRICTED DIMENSION lrd AS $d[l'..l'']$;	$lrd \equiv d \sqcap \exists$atLevel.($\exists$rollsUpTo$^-$.$\{l'\} \sqcap$ $\exists$rollsUpTo.$\{l''\}$)	create view lrd as select * from d where d_lvl in ((select d_lvl_sup as d_lvl from d_lvlrollup where d_lvl = 'l''') intersect (select d_lvl from d_lvlrollup where d_lvl_sup = 'l'''));
(4)	CREATE DIMENSION ROLE dr OF d;	$\top \sqsubseteq \leqslant 1 dr$ $\exists dr.\top \sqsubseteq$ Point $\top \sqsubseteq \forall dr.d$	... dr varchar not null references d ...
(5)	CREATE DIMENSION SPACE ds AS ($dr_1 lrd_1, \ldots, dr_n lrd_n$);	$ds \equiv$ Point$\sqcap \exists dr_1.lrd_1 \sqcap \cdots \sqcap \exists dr_n.lrd_n$	create view ds as select * from (select d_1 as dr_1 from lrd_1) natural join ... natural join (select d_n as dr_n from lrd_n); — d_i is the dimension referred to by dr_i
(6)	CREATE MEASURE msr FOR ds;	$\exists msr.\top \sqsubseteq ds$	msr(<u>$dr_1, \ldots, dr_n$</u>, msr) — $dr_1, \ldots, dr_n$ are the dimension roles of ds

entity class ecl. A node (Row 2) is member of a dimension and defined at a particular level of the dimension. The nodes of a dimension are organized in a node roll-up hierarchy (Row 3) which corresponds to the level roll-up hierarchy. A node refers to a member of the entity class that the node's level refers to. The node is the role that the entity plays within the dimension. An entity may be referred to by at most one node per dimension.

In addition to entities and nodes, an OLAP cube comprises points. A point is member of a dimension space ds. The extension of a dimension space ds, that is, the set of member points, corresponds to the cartesian product of the dimension extensions of the dimension roles $dr_1, \ldots, dr_n$ of dimension space ds. A point is described by measure values. A point is identified by its coordinates $nd_1, \ldots, nd_n$, one for each of the dimension roles $dr_1, \ldots, dr_n$ of dimension space ds. Each

Table 2. Example fragments of an OLAP cube schema

	MDO	OWL	Relational
(1)	CREATE ENTITY CLASS CityE (inhabitants integer)	CityE $\sqsubseteq$ Entity $\exists$CityE_inhabitants.$\top \sqsubseteq$ CityE $\top \sqsubseteq \leqslant 1$CityE_inhabitants	CityE(CityE, inhabitants)
(2)	CREATE DIMENSION Doctor WITH LEVELS doctor DoctorE, city CityE AND HIERARCHY doctor UNDER city;	Doctor $\sqsubseteq$ Node $\exists$atLevel.{doctor} $\equiv$ Doctor $\sqcap \exists$roleOf.DoctorE $\exists$atLevel.{city} $\equiv$ Doctor $\sqcap \exists$roleOf.DoctorE $\top \sqsubseteq \leqslant 1$rollsUpTo_doctor $\top \sqsubseteq \leqslant 1$rollsUpTo_city directlyRollsUpTo(doctor,city)	Doctor_lvl(Doctor_lvl,entityclass) Doctor_lvlparent(Doctor_lvl, Doctor_lvl_sup) Doctor_lvlrollup(Doctor_lvl, Doctor_lvl_sup) Doctor(Doctor, Doctor_lvl) Doctor_parent(Doctor, Doctor_sup) Doctor_rollup(Doctor, Doctor_sup) insert into "Doctor_lvl" values ('doctor','DoctorE'); insert into "Doctor_lvl" values ('city','CityE'); insert into "Doctor_lvlparent" values ('doctor','city');
(3)	CREATE LEVELRANGERE-STRICTED DIMENSION Doctor_CityCountry AS Doctor[city..country];	Doctor_CityCountry $\equiv$ Doctor $\sqcap$ $\exists$atLevel.($\exists$rollsUpTo$^-$.{city} $\sqcap \exists$rollsUpTo.{country})	create view "Doctor_CityCountry" as select * from "Doctor" where "Doctor_lvl" in ((select "Doctor_lvl_sup" as "Doctor_lvl" from "Doctor_lvlrollup" where "Doctor_lvl" = 'city') intersect (select "Doctor_lvl" from "Doctor_lvlrollup" where "Doctor_lvl_sup = 'country'));
(4)	CREATE DIMENSION ROLE actDoc OF Doctor;	$\top \sqsubseteq \leqslant 1$actDoc $\exists$actDoc.$\top \sqsubseteq$ Point $\top \sqsubseteq \forall$actDoc.$Doctor$	... "actDoc" varchar not null references "Doctor" ...
(5)	CREATE DIMENSION SPACE ds1 AS (actDoc Doctor_CityCountry, time Time_MonthYear);	ds1 $\equiv$ Point $\sqcap$ $\exists$actDoc.Doctor_CityCountry $\sqcap \exists$time.Time_MonthYear	create view "ds1" as select * from (select "Doctor" as "actDoc" from "Doctor_CityCountry") natural join (select "Time" as "time" from "Time_MonthYear");
(6)	CREATE MEASURE costs FOR ds1;	$\exists$costs.$\top \sqsubseteq$ ds1	costs(actDoc,time,costs)

coordinate refers to one of nodes $nd_1, \ldots, nd_n$ of the dimensions $d_1, \ldots, d_n$ that correspond to the dimension roles $dr_1, \ldots, dr_n$. A link between point, measure, and value is also referred to as fact. A fact assigns a value val to a measure msr of a point p.

3.2 OWL Representation and Relational Views

For the purpose of automated subsumption checking over MDO concepts using off-the-shelf reasoners, we introduce a representation of MDO in OWL 2 DL[1]. This OWL representation does not represent the full semantics of OLAP cubes. For the purpose of management of shared business terms, an incomplete representation of OLAP cubes is sufficient.

[1] `http://www.w3.org/TR/owl2-overview/`

Table 3. Representation of an OLAP cube instance in OWL and as relational views

	MDO (open world)	**OWL** (open world)	**Relational** (closed world)
(1)	CREATE ENTITY e OF ecl ($attr_1 = val_1, \ldots, attr_n = val_n$);	$ecl(e)$ $attr_1(e, val_1)$ $\ldots$ $attr_n(e, val_n)$	insert into ecl (ecl, $attr_1$, $\ldots$, $attr_n$) values (e, val_1, $\ldots$, val_n);
(2)	CREATE NODE nd OF d AT l AS ROLEOF e;	$d(nd)$ atLevel(nd, l) roleOf(nd, e)	insert into d(d,d_lvl) values (e,l);
(3)	CREATE DIRECT ROLLUP FROM nd TO nd';	directlyRollsUpTo(nd, nd')	insert into d_parent (d, d_sup) values (nd,nd'); — d is the dimension of nd

The OLAP cube representation in OWL builds on a set of generic OWL classes and properties as described in Fig. 5. Object property directlyRollsUpTo represents roll-up hierarchies of nodes and levels alike. No distinction is required between the different kinds of hierarchies. Object property rollsUpTo represents the transitive and reflexive closure of directlyRollsUpTo. Since OWL does not provide a means for defining an object property as transitive-reflexive closure of another property, an incomplete work-around is provided as follows. The rollsUpTo property is characterized as transitive and directlyRollsUpTo defined as a sub-property; each node and level is specified to roll-up to itself.

Table 1, in the middle column, shows the OWL equivalents, in Description Logics notation, for the MDO OLAP cube schema definitions. Table 3, in the middle column, shows the OWL equivalents for the MDO OLAP cube instance definitions. Typically, only a small subset of the entities and nodes of an OLAP

Class: **Entity**	(1)
Class: **Node** SubClassOf: directlyRollsUpTo Self and roleOf some Entity and atLevel some Level and rollsUpTo only Node	(2)
Class: **Level** SubClassOf: directlyRollsUpTo Self and rollsUpTo only Level	(3)
Class: **Point**	(4)
DisjointClasses: Entity, Node, Level, Point	(5)
ObjectProperty: **directlyRollsUpTo** SubPropertyOf: rollsUpTo	(6)
ObjectProperty: **rollsUpTo** Characteristics: Transitive	(7)
ObjectProperty: **roleOf** Characteristics: Functional Domain: Node Range: Entity	(8)
ObjectProperty: **atLevel** Characteristics: Functional Domain: Node Range: Level	(9)
for the set of all entity classes $\{ecl_1, \ldots, ecl_n\}$: DisjointClasses: ecl_1, $\ldots$, ecl_n	(10)
for the set of all dimensions $\{d_1, \ldots, d_n\}$: DisjointClasses: d_1, $\ldots$, d_n	(11)
for the set of all entities $\{e_1, \ldots, e_n\}$: DifferentIndividuals: e_1, $\ldots$, e_n	(12)

Fig. 5. OWL classes and properties for the representation of OLAP cubes (in Manchester syntax)

cube are represented as individuals in the MDO. An OLAP cube further consists of points which are, however, not explicitly represented at the individual level of an MDO.

We do not make any assumptions about the physical storage of OLAP cubes or the algorithms for calculating and materializing derived measures. We regard OLAP cubes as data structures at the logical level which allow to directly query derived measure values and to abstract from the calculation of measure values. We employ SQL as language for querying OLAP cubes via relational views.

Table 1, in the right column, shows the relational views for the OLAP cube schema definitions. The primary keys of a view are underlined. The primary key values of the entity views, also referred to as entity IDs, are assumed to be globally unique. Some of the relational views are derived from other views. The lvlrollup views (Row 2) are the transitive and reflexive closure of the lvlparent views, the rollup views are the transitive and reflexive closure of the parent views. The definition of these views is omitted due to space considerations. Table 3, in the right column, shows the relational views for the OLAP cube instance definitions.

4 Defining MDO Concepts over OLAP Cubes

In this section we introduce a language for describing business terms as used in data analysis in the form of concepts in a multi-dimensional ontology. We specify the syntax of entity concept, dimensional concept, and multi-dimensional concept descriptions together with their representation in SQL and in OWL.

There are two main tasks concerning MDO concepts: First, MDO concepts should be directly usable in querying the OLAP cube. Second, MDO concepts should be organized in subsumption hierarchies to simplify their collaborative management and use.

For querying the OLAP cube we make the closed world assumption and consider the OLAP cube as the single interpretation (in a model-theoretic sense) of the MDO and assume that the OLAP cube is consistent with the MDO (that is, a model of the MDO). The translation of MDO concepts to SQL *concept views* does not only allow to query the OLAP cube but also defines their interpretation. That is, the semantics of the MDO concept description language is informally specified in terms of SQL. To facilitate translation to SQL views we only allow acyclic concept definitions.

For deriving subsumption hierarchies we make the open world assumption, because subsumption hierarchies should not change when the OLAP cube instance changes. To automate subsumption checking, MDO concepts are translated to OWL 2 DL and subsumption checking is delegated to OWL reasoners.

The signature of a concept is given by its name and indicates the sort of individuals over which it is defined. The extension or interpretation of a concept may be specified in one the following ways: The extension of a *primitive concept* is asserted or derived outside the MDO and is treated as black box for reasoning. The extension of a *defined concept* is derived according to a concept

expression which are the basis for automated subsumption checking. The extension of an *SQL-defined concept* is specified by an SQL query over the enriched OLAP cube. For subsumption checking, SQL-defined concepts are treated as primitive concepts with their associated SQL query being ignored.

4.1 Entity Concepts

An entity concept ec is defined over an entity class ecl, its domain, and is interpreted by a subset of the members of this entity class. An entity concept is either primitive (see Table 4 Row 1), SQL-defined (Row 2), or defined (3). Primitive concepts are defined or asserted outside the MDO. SQL-defined concepts are defined within the MDO by an SQL query which is ignored for reasoning and thus not part of the transformation to OWL. Defined concepts are defined by an entity concept expression in one of the following forms: (3a) a singleton concept consisting of a single entity, (3b) an attribute-value restriction, for example, concept bigCity in Table 5, (3c) the disjunction of entity concepts, for example, concept bigOrSmallCity, (3d) the conjunction of entity concepts, (3e) or the complement of an entity concept. Attributes, entities, and entity concepts referred to in an entity concept definition all belong to the entity class which serves as domain of the defined concept.

Table 4. MDO entity concepts and their representation in OWL and SQL

	MDO	in **OWL**	in **SQL**
	CREATE ENTITY CONCEPT ec FOR ecl	$ec \sqsubseteq ecl$	$ec(\underline{ecl})$
(1)	AS PRIMITIVE;		
(2)	AS SQL: $sqlquery$;		create view ec AS $sqlquery$;
(3)	AS	$ec \equiv$	create view ec as
(3a)	e;	$\{e\}$	select ecl from ecl where $ecl = $ 'e';
(3b)	$attr\ \theta\ value$; — θ is some comparison operator	$\exists attr.dt[\theta\ value]$ — dt is the datatype of $attr$	select ecl from ecl where $attr\ \theta\ value$;
(3c)	UNION OF ($ec_1, \ldots, ec_n$);	$ec_1 \sqcup \cdots \sqcup ec_n$	(select ecl from ec_1) union ... union (select ecl from ec_n);
(3d)	INTERSECTION OF ($ec_1, \ldots, ec_n$);	$ec_1 \sqcap \cdots \sqcap ec_n$	(select ecl from ec_1) intersect ... intersect (select ecl from ec_n);
(3e)	NOT ec';	$ecl \sqcap \neg ec'$	(select ecl from ecl) except (select ecl from ec');

Entity concepts can easily be represented in traditional ontology languages with the translation of MDO entity concepts to OWL being straightforward (see middle column of Table 4). Representing entity concepts as SQL views is also straightforward (see third column of Table 4). Every concept is represented by a

Table 5. Examples of MDO entity concepts

MDO	in OWL	in SQL
CREATE ENTITY CONCEPT bigCity FOR CityE AS inhabitants > 100000;	bigCity $\sqsubseteq$ CityE bigCity $\equiv$ $\exists$CityE_inhabitants.integer[> 100000]	create view "bigCity" as select "CityE" from "CityE" where "inhabitants" > 100000
CREATE ENTITY CONCEPT bigOrSmallCity FOR CityE AS UNION OF (bigCity, smallCity);	bigOrSmallCity $\sqsubseteq$ CityE bigOrSmallCity $\equiv$ bigCity $\sqcup$ smallCity	create view "bigOrSmallCity" as (select "CityE" from "bigCity") union (select "CityE" from "smallCity");

concept view with the concept name as view name and a single attribute named after the entity class.

4.2 Dimensional Concepts

A dimensional concept dc is defined over a level-range-restricted dimension lrd, its domain, and is interpreted by a subset of the nodes in this domain. Interpretations of dimensional concepts have the *hierarchy property*: if a node is in the interpretation of a dimensional concept, then all its sub- and superordinate nodes within the domain of the concept are also in the interpretation of the concept. First experience shows that enforcing the hierarchy property simplifies working with and understanding dimensional concepts.

Table 6. MDO dimensional concepts and their representation in OWL and SQL

	MDO	OWL	Relational/SQL
	CREATE DIMENSIONAL CONCEPT dc FOR lrd	$dc \sqsubseteq lrd$	$dc(\underline{d})$ — d is the dimension of lrd
(1)	AS PRIMITIVE;		
(2)	AS SQL: $sqlquery$;		create view dc as $sqlquery$;
(3)	AS	$dc \equiv$	create view dc as
(3a)	nd;	$\{nd\}$	select d from d where $d =$ 'nd';
(3b)	$d : ec$;	$d \sqcap \exists$roleOf.ec	select d from d where d in (select * from ec);
(3c)	dc'*;	$\exists$rollsUpTo_$l.dc'$ — l is a level of dc'	select d from d_rollup where d_sup in (select d from dc');
(3d)	$dc'[lrd]$	$dc' \sqcap lrd$	select d from dc' natural join lrd
(3e)	UNION OF (dc_1, ..., dc_n);	$dc_1 \sqcup \cdots \sqcup dc_n$	(select d from dc_1) union ... union (select d from dc_n);
(3f)	INTERSECTION OF (dc_1, ..., dc_n);	$dc_1 \sqcap \cdots \sqcap dc_n$	select d from dc_1 natural join ... natural join dc_n;
(3g)	NOT dc';	$lrd \sqcap \neg dc'$	(select d from lrd) except (select d from dc')

A dimensional concept is described in one of the following ways: (1) as primitive, (2) by an SQL-view over the OLAP cube, or (3) by a concept expression in one of the following forms, (3a) a single node such that only this node is in the interpretation of the concept, (3b) by a reference to an entity concept such that each node that refers to an entity satisfying the entity concept is in the interpretation of the defined concept, for example, concept doc_bigCity in Table 8, (3c) by hierarchy expansion of some concept such that each node of the dimension that is in the interpretation of the concept or some direct or indirect successor node thereof is in the interpretation, concept DocInBigCity, (3d) by level range restriction of some concept, (3e) by disjunction of concepts defined for the same level-range or (3f) by conjunction of concepts, for example, concept DocInBgCtySmCntry, (3g) the complement of a concept with regard to its domain (with the open world interpretation of complement).

Table 7. Representing disjointness of sub-dimensions

for each level l:	for example, for level country:
$\mathsf{rollsUpTo\_}l \sqsubseteq \mathsf{rollsUpTo}$	$\mathsf{rollsUpTo\_country} \sqsubseteq \mathsf{rollsUpTo}$
$\exists \mathsf{atLevel}.\exists \mathsf{rollsUpTo}.\{l\} \equiv \exists \mathsf{rollsUpTo\_}l.\top$	$\exists \mathsf{atLevel}.\exists \mathsf{rollsUpTo}.\{\mathsf{country}\} \equiv \exists \mathsf{rollsUpTo\_country}.\top$
$\top \sqsubseteq \forall \mathsf{rollsUpTo\_}l.\exists atLevel.\{l\}$	$\top \sqsubseteq \forall \mathsf{rollsUpTo\_country}.\exists \mathsf{atLevel}.\{\mathsf{country}\}$
$\top \sqsubseteq \leqslant 1 \mathsf{rollsUpTo\_}l$	$\top \sqsubseteq \leqslant 1 \mathsf{rollsUpTo\_country}$
for each node nd at level l:	for example, for node austria at level country:
$\exists \mathsf{rollsUpTo}.\{nd\} \equiv \exists \mathsf{rollsUpTo\_}l.\{nd\}$	$\exists \mathsf{rollsUpTo}.\{\mathsf{austria}\} \equiv \exists \mathsf{rollsUpTo\_country}.\{\mathsf{austria}\}$

The domain of a defined concept may be derived from its concept expression. The domain of a concept conjunction is the intersection of domains of the constituent concepts. The domain of the complement of a concept is that of the concept. Concepts that are constructed by a single node or by reference to an entity concept are flat, that means that their domain is restricted to a single level namely those of the node(s). The domain of a concept defined by hierarchical expansion is that of the concept but expanded to the bottom level of the dimension.

The representation of dimensional concepts in OWL (see the middle column in Table 6) comes with one key challenge. OWL 2 DL does not allow to express that a transitive property (such as rollsUpTo) maps to exactly one object in a given range (for example, to one node of one level). But, the essential characteristics of roll-up hierarchies of data warehouse dimensions are that the rollsUpTo-relationship between nodes is transitive, and each node of a level rolls up to exactly one node of each higher level (to which the former level rolls up). Without these semantics of roll-up hierarchies in data warehousing being captured, the subsumption hierarchy determined by an OWL reasoner will be sound, but incomplete. More specific, it will not recognize that if two concepts dc and dc' are disjoint, their hierarchical expansions, dc^* and dc'^* will be disjoint, too.

To cope with this limitation of OWL, we include redundant, derived information in the OWL representation of levels and nodes. For each level l we introduce a functional roll-up property and assert for every named node nd at level l that all its descendant nodes roll up to nd via this functional roll-up property (see Table 7). Using such derived functional roll-up properties for defining hierarchical expansion of dimensional concepts (see Table 6 Row *3c* and the second row in Table 8) allows the reasoner to derive disjointness of concepts which are defined by hierarchical expansion.

Table 8. Examples of MDO dimensional concepts

MDO	in **OWL**	in **SQL**
CREATE DIMENSIONAL CONCEPT doc_bigCity FOR Doctor_city AS Doctor:bigCity;	doc_bigCity $\sqsubseteq$ Doctor_city doc_bigCity $\equiv$ Doctor $\sqcap$ $\exists$roleOf.bigCity	create view "doc_bigCity" as select "Doctor" from "Doctor" where "Doctor" in (select * from "bigCity")
CREATE DIMENSIONAL CONCEPT DocInBigCity FOR Doctor_doctorCity AS doc_bigCity*;	DocInBigCity $\sqsubseteq$ Doctor_doctorCity DocInBigCity $\equiv$ $\exists$rollsUpTo_city.doc_bigCity	create view "DocInBigCity" as select "Doctor" from "Doctor_rollup" where "Doctor_sup" in (select * from "bigCity")
CREATE DIMENSIONAL CONCEPT DocInBgCtySmCntry FOR Doctor_doctorCity AS INTERSECTION OF(DocInBigCity, DocInSmallCountry);	DocInBgCtySmCntry $\sqsubseteq$ Doctor_doctorCity DocInBgCtySmCntry $\equiv$ DocInBigCity $\sqcap$ DocInSmallCountry	create view "DocInBgCtySmCntry" as (select "Doctor" from "DocInBigCity") intersect (select "Doctor" from "DocInSmallCountry")

The translation of dimensional concepts to SQL views is presented in Table 6. A dimensional concept is represented by a single-column concept view which holds a subset of the extension of the node view of the level-range restricted dimension which represents its domain. Since entity IDs (primary key values in entity views) are also used as node IDs, entity concept views can easily be joined with node views (see 3b).

4.3 Multi-dimensional Concepts

A multi-dimensional concept mc is defined over a dimension space ds, its domain. Multi-dimensional concepts have an *inner* interpretation and an *outer* interpretation. The inner interpretation contains all points that have *exactly* the dimension roles of the dimension space and fulfill the restrictions on these dimension roles. The outer interpretation contains all points that have *at least* the dimension roles of the dimension space and fulfill the restrictions on these dimension roles.

The concept expression of a multi-dimensional concept (md-concept) is given in one of the following ways: (1) as primitive, (2) by an SQL query over the relational representation of the underlying OLAP cube (3) by a concept expression in one of the following forms, (3a) a tuple of references to dimensional concepts for some dimension roles in which case all points that satisfy the dimensional concepts in the indicated dimension roles are in the interpretation,

Table 9. MDO multi-dimensional concepts and their representation in OWL and SQL

	MDO	in **OWL**	in **SQL** (inner interpretation)
	CREATE MULTIDIMENSIONAL CONCEPT mc FOR ds ...	$mc \sqsubseteq ds$ — $dr_1, \ldots, dr_n$ are the dimension roles of ds. d_i is the dimension of dr_i.	$mc(\underline{dr_1, \ldots, dr_n})$
(1)	AS PRIMITIVE;		
(2)	AS SQL: *sqlquery*;		create view mc as *sqlquery* ;
(3)	AS	$mc \equiv$	create view mc as
(3a)	$(dr_1{:}dc_1, \ldots, dr_n{:}dc_n)$;	$\exists dr_1.\{nd_1\} \sqcap \cdots \sqcap \exists dr_n.\{nd_n\}$	select * from ds where dr_1 in (select * from dc_1) and ... and dr_n in (select * from dc_n);
(3b)	mc'^*;	see text	select * from ds s, mc' c where (s.dr_1, c.dr_1) in (select * from d_1_rollup) and ... and (s.dr_n,c.dr_n) in (select * from d_n_rollup) ;
(3c)	$mc'[ds]$	$mc' \sqcap ds$	select * from mc' natural join ds;
(3d)	UNION OF (mc_1, ..., mc_m);	$mc_1 \sqcup \cdots \sqcup mc_m$	(select * from mc_1) union ... union (select * from mc_m);
(3e)	INTERSECTION OF (mc_1, ..., mc_m);	$mc_1 \sqcap \cdots \sqcap mc_m$	select * from mc_1 natural join ... natural join mc_m;
(3f)	NOT mc';	$ds \sqcap \neg mc'$	(select * from ds) except (select * from mc')
(3g)	$msr\ \theta\ value$	$\exists msr.[\theta\ value]$	select s.* from ds s natural join msr m where $msr\ op\ value$;

for example, leadDocInBgCtySmCntry in Table 10, (3b) by hierarchy expansion of an md-concept, such that each point that is in the interpretation of the md-concept or a descendent thereof is in the interpretation, for example, concept leadDocInBgCtySmCntryIn2010, (3c) by restriction to a dimension space which is restricted to a smaller granularity range, (3d) by disjunction of md-concepts with the same domain or (3e) by conjunction of md-concepts, (3f) by complement of an md-concept (open world interpretation), or (3g) by a boolean expression over measure-value comparisons of measures applied to a point, for example, expensiveDoctorCombos.

Md-concepts are divided into primitive and defined. Defined md-concepts are divided into dimension-based and fact-based. A dimension-based md-concept is an md-concept that is defined only by reference to dimensional concepts. The hierarchical expansion, disjunction, conjunction, and complement of dimension-based md-concepts is also a dimension-based md-concept. A fact-based md-concept is an md-concept that is defined by predicates on measure values.

Non hierarchically-expanded md-concepts are mapped to OWL according to Table 9 and subsumption checking is delegated to the OWL reasoner. In order for subsumption checking over hierarchically-expanded dimension-based md-concepts to be delegated to the OWL reasoner, the md-concepts must be transformed into disjunctive normal form (DNF). The DNF of a dimension-based md-concept mc is a disjunction of tuples of references to dimensional concepts (Equation 1). The conjunction of tuples of references to dimensional

Table 10. Examples of MDO multi-dimensional concepts

MDO	in **OWL**	in **SQL**
CREATE MULTIDIMENSIONAL CONCEPT leadDocInBgCtySmCntry FOR ds_leadDoc AS leadDoc:DocInBgCtySmCntry;	leadDocInBgCtySmCntry $\sqsubseteq$ ds_leadDoc leadDocInBgCtySmCntry $\equiv$ $\exists$ds_leadDoc .DocInBgCtySmCntry	create view "leadDocInBgCtySmCntry" as select * from leadDoc where leadDoc in (select * from "DocInBgCtySmCntry")
CREATE MULTIDIMENSIONAL CONCEPT leadDocInBgCtySmCntryIn2010 FOR ds_leadDoc_time AS INTERSECTION OF (leadDocInBgCtySmCntry, timeIn2010)) ;	leadDocInBgCtySmCntryIn2010 $\sqsubseteq$ ds_leadDoc_time leadDocInBgCtySmCntryIn2010 $\equiv$ leadDocInBgCtySmCntry $\sqcap$ timeIn2010	create view "leadDocInBgStySmCntryIn2010" as select * from "leadDocInBgCtySmCntry" natural join "timeIn2010"
CREATE MULTIDIMENSIONAL CONCEPT expensiveDoctorCombos FOR leadDoc_actDoc_year AS medianCostsPerIns $>$ 1000	expensiveDoctorCombos $\sqsubseteq$ leadDoc_actDoc_year expensiveDoctorCombos $\equiv$ medianCostsPerIns.double[$>$ 1000.0]	create view "expensiveDoctorCombos" as select * FROM "leadDoc_actDoc_year" natural join "medianCostsPerIns" where "medianCostsPerIns" $>$ 1000;

concepts is rewritten to a single tuple where dimensional concepts that are referred to by the same dimension role are conjoined (Equation 2). The DNF of the hierarchical expansion of mc corresponds to the DNF of mc with the referred dimensional concepts being hierarchically expanded (Equation 3). The DNF of mc restricted to a dimension space ds that is restricted to a smaller granularity range ($ds = dr_1[l_1..l'_1], \ldots, dr_n[l_n..l'_n]$) corresponds to the DNF of mc with the referred dimensional concepts being restricted accordingly (Equation 4). The DNF of the complement of mc corresponds to a disjunction of tuples of references to dimensional concepts. In this disjunction, every term is only restricted in one dimensional concept with the others left unrestricted (Equation 5). Remember, disjunction of md-concepts is only possible over md-concepts with the same dimension space.

$$DNF(mc) = \bigvee_{i=1..m} (dr_1{:}dc_{1_i}, \ldots, dr_n{:}dc_{n_i}) \tag{1}$$

$$\begin{aligned}&(dr_1{:}dc_1, \ldots, dr_k{:}dc_k, \ldots, dr_n{:}dc_n) \wedge (dr_1{:}dc'_1, \ldots, dr_k{:}dc'_k) = \\ &(dr_1{:}(dc_1 \wedge dc'_1), \ldots, dr_k{:}(dc_k \wedge dc'_k), \ldots, dr_n{:}dc_n)\end{aligned} \tag{2}$$

$$DNF(mc^*) = \bigvee_{i=1..m} (dr_1{:}dc^*_{1_i}, \ldots, dr_n{:}dc^*_{n_i}) \tag{3}$$

$$DNF(mc[ds]) = \bigvee_{i=1..m} (dr_1{:}dc_{1_i}[l_1..l'_1], \ldots, dr_n{:}dc_{n_i}[l_n..l'_n]) \tag{4}$$

$$
\begin{aligned}
DNF(\neg(dr_1{:}dc_1, \ldots, dr_k{:}dc_k, \ldots, dr_n{:}dc_n)) = \\
(dr_1{:}\neg dc_1, \ldots, dr_k{:}\top_k, \ldots, dr_n{:}\top_n) \vee \cdots \vee \\
(dr_1{:}\top_1, \ldots, dr_k{:}\neg dc_k, \ldots, dr_n{:}\top_n) \vee \cdots \vee \\
(dr_1{:}\top_1, \ldots, dr_k{:}\top_k, \ldots, dr_n{:}\neg dc_n)
\end{aligned} \tag{5}
$$

5 Prototype Implementation

In this section we describe the implementation of the approach as part of the semCockpit system prototype. The approach is implemented using off-the-shelf OWL reasoners (HermiT [16]) and database technology (Oracle DB 11.2g), the Java programming language and the OWL API [6]. The prototype architecture is geared towards functionality, rapid prototyping and experimentation. Performance considerations are widely neglected.

MDOs are persisted in a relational database called MDO DB. The schema of the MDO DB implements the abstract syntax (not discussed in this paper) of the MDO language. Every MDO language construct is represented by one or more tables in the MDO DB and every MDO object is represented by one or more tuples. By using transactional support of the relational database management system, MDOs can be maintained concurrently by different analysts.

The concrete MDO syntax (as presented in this paper) is implemented as ANTLR4 grammar. An MDO parser translates MDO statements to insert-statements against the MDO DB.

The semCockpit data warehouse (semDWH) holds the semantically enriched OLAP cube and provides relational views on dimensions, (derived) facts and MDO concepts as specified in Sections 3 and 4. In order to allow for rapid prototyping and experimentation with different language features, the OLAP cube is directly implemented as a set of SQL views over the asserted data. In future work, we will investigate platform-specific logical and physical optimizations.

The MDO DB also holds the OWL and SQL representations of MDO objects. The translations from MDO syntax to OWL and SQL (as specified in Tables 4, 6, and 9) are implemented by stored procedures within the MDO DB. When inserting or updating an MDO object, inserts or updates to the tables holding the OWL and SQL representations are triggered. This allows for the inspection and alteration of OWL and SQL representations of MDO concepts prior to invoking the reasoner and prior to creating or replacing views in the semDWH.

6 Related Work

The MASTRO system [5] for ontology-based data access uses ontologies for querying (relational) databases. Similarly, the semCockpit approach employs the MDO as a "high-level, conceptual view over data repositories". In order to accommodate for the particularities of data warehousing and OLAP the semCockpit approach assumes complete data (closed world assumption) as a prerequisite for correct data aggregation. Furthermore, the MDO provides a set of primitives for the definition of the multi-dimensional and hierarchical space the multi-dimensional concepts are defined in.

Other work has investigated the difference between open-world ontologies and closed-world databases. Motik et al. [9] extend knowledge bases that are based on Description Logics with closed-world integrity constraints. Lim et al. [8] employ virtual views as a query interface for semantically-enriched relational data.

In the context of data warehousing and OLAP, the use of existing domain ontologies has been discussed from different perspectives. The AMDO method [15] allows for an automated derivation of conceptual multi-dimensional models from domain knowledge represented in ontologies. The DWOBS tool [7] supports the design of data warehouses from ontology-based operational databases. Pardillo and Mazón [14] propagate the use of ontologies to fix the shortcomings of traditional data warehouse design methods. In contrast, the semCockpit approach introduces MDOs specifically for knowledge sharing between business analysts.

The Multidimensional Integrated Ontology (MIO) [10] integrates various sources of knowledge and defines measures, dimensions, and hierarchies for OLAP over the different sources of knowledge. In contrast, MDOs do not integrate existing ontologies. Rather, MDOs define business terms, building on hierarchical and multi-dimensional primitives for querying traditional data warehouses.

The partitioning of the terminological box of an ontology [4] into a schema part and a view part, with distinct language constructs for either part, is of particular importance for MDOs. For an overview of the use of semantic web technologies in Business Intelligence we refer to Berlanga et al. [3].

7 Conclusion

A multi-dimensional ontology (MDO) is a formal representation of a conceptualization of a data analysis domain. This formal representation facilitates the unambiguous interpretation of query results and allows business analysts to share their knowledge in data analysis projects. By providing translations of MDO concepts to relational views, semantically-enriched OLAP cubes may be queried using SQL. By providing a representation of MDO in OWL, off-the-shelf reasoners may be used for the classification of business terms which is vital for the management of large vocabularies. Furthermore, the OWL representation of MDO concepts facilitates the integration of existing domain ontologies.

The MDO is the fundamental of ontology-driven comparative data analysis as investigated in the *Semantic Cockpit* (semCockpit) project. In the semCockpit system dimensions are extended by concept levels, the members of which are MDO concepts. In order to ensure summarizability the MDO concepts of a concept level must be pairwise disjoint. Checking disjointness of MDO concepts can be reduced to subsumption checking as provided by the approach presented in this paper. Semantic dimensions [2] allow for the use of external domain ontologies, for example, SNOMED CT, as dimensions of OLAP cubes. Having an OWL representation of MDO concepts serves as the basis for the seamless integration of semantic dimensions into the MDO. BI analysis graphs [11] extend the common set of OLAP operations, for example, slice and dice, with a set of operations for the navigation along subsumption hierarchies of MDO concepts.

References

1. Querying Dimensional Objects, Oracle OLAP User's Guide, 11g Release 2 (2010), http://docs.oracle.com/cd/E11882_01/olap.112/e17123/query.htm
2. Anderlik, S., Neumayr, B., Schrefl, M.: Using domain ontologies as semantic dimensions in data warehouses. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012. LNCS, vol. 7532, pp. 88–101. Springer, Heidelberg (2012)
3. Berlanga, R., Romero, O., Simitsis, A., Nebot, V., Pedersen, T.B., Abello, A., Aramburu, M.J.: Semantic web technologies for business intelligence (2011)
4. Buchheit, M., Nutt, W., Donini, F.M., Schaerf, A.: Refining the structure of terminological systems: Terminology = schema + views. In: Hayes-Roth, B., Korf, R.E. (eds.) AAAI, pp. 199–204. AAAI Press/The MIT Press (1994)
5. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web 2(1), 43–53 (2011)
6. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web 2(1), 11–21 (2011)
7. Khouri, S., Bellatreche, L.: DWOBS: Data warehouse design from ontology-based sources. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011, Part II. LNCS, vol. 6588, pp. 438–441. Springer, Heidelberg (2011)
8. Lim, L., Wang, H., Wang, M.: Unifying data and domain knowledge using virtual views. In: VLDB, pp. 255–266 (2007)
9. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. Journal of Web Semantics 7(2), 74–89 (2009)
10. Nebot, V., Llavori, R.B.: Building data warehouses with semantic web data. Decision Support Systems 52(4), 853–868 (2012)
11. Neuböck, T., Neumayr, B., Rossgatterer, T., Anderlik, S., Schrefl, M.: Multidimensional navigation modeling using BI Analysis Graphs. In: Castano, S., Vassiliadis, P., Lakshmanan, L.V., Lee, M.L. (eds.) ER Workshops 2012. LNCS, vol. 7518, pp. 162–171. Springer, Heidelberg (2012)
12. Neumayr, B., Anderlik, S., Schrefl, M.: Towards ontology-based olap: datalog-based reasoning over multidimensional ontologies. In: Song, I.Y., Golfarelli, M. (eds.) DOLAP, pp. 41–48. ACM (2012)
13. Neumayr, B., Schrefl, M., Linner, K.: Semantic Cockpit: An ontology-driven, interactive business intelligence tool for comparative data analysis. In: De Troyer, O., Bauzer Medeiros, C., Billen, R., Hallot, P., Simitsis, A., Van Mingroot, H. (eds.) ER 2011 Workshops. LNCS, vol. 6999, pp. 55–64. Springer, Heidelberg (2011)
14. Pardillo, J., Mazón, J.N.: Using ontologies for the design of data warehouses. International Journal of Database Management Systems 3(2), 73–87 (2011)
15. Romero, O., Abelló, A.: A framework for multidimensional design of data warehouses from ontologies. Data Knowledge Engineering 69(11), 1138–1157 (2010)
16. Shearer, R., Motik, B., Horrocks, I.: HermiT: A highly-efficient OWL reasoner. In: Dolbear, C., Ruttenberg, A., Sattler, U. (eds.) OWLED 2008. CEUR Workshop Proceedings, vol. 432. CEUR-WS.org (2008)