M2RML:
Mapping Multidimensional Data to RDF

by

Saleh Ghasemi

B.Sc. (Hons., Computing Science), Staffordshire University, 2011

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY

Spring 2014

Approval

Name: **Saleh Ghasemi**

Degree: **Master of Science**

Title of Thesis: ***M2RML: Mapping Multidimensional Data to RDF***

Examining Committee: **Chair:** Qianping Gu
Professor

Wo-Shun Luk
Senior Supervisor
Professor

Jian Pei
Supervisor
Professor

Fred Popowich
Examiner
Professor

Date Defended/Approved: January 07 2014

Partial Copyright Licence

Abstract

In this thesis we provide a mapping language that facilitates mapping of multidimensional data to RDF datasets using Data Cube Vocabulary (DCV), W3C candidate for publishing statistical data on the web in RDF format. RDF is W3C standard for data interchange which allows data, structured and semi-structured with different underlying schemas to be mixed, exposed and shared among different applications.

The language design is similar to recently published R2RML used for mapping relational datasets to RDF datasets having similar core elements. As we design the mapping language general enough to work with various data sources of statistical data we also provide a framework specific to publishing OLAP cubes based on our mapping language. In this framework we address selective issues of DCV and propose an extension which improves DCV for representing multidimensional data and allows one-to-one mapping between an OLAP cube and RDF/QB elements.

Keywords: Open Linked Data; Resource Description Framework (RDF); Data Cube Vocabulary; Multidimensional Data; OLAP

Dedication

To my family especially my wife, without whose support and patience this work would have never been completed.

Acknowledgements

I should thank my senior supervisor Prof. Wo-Shun Luk for all his support and guidance in all phases of developing this work from idea generation and problem identification to completing it to this point. There is no way I can thank him for being so understanding of and flexible with my constantly changing situation.

I should also thank my defence examiner Prof. Fred Popowich and my supervisor Prof. Jian Pei for their constructive feedback on this work and their suggestions to make this work even better and more complete.

I would also like to thank my wife for her continuous support, encouragement and patience at all times as well as my father and mother without whom I could have never reached this point in my academic, professional, and personal life.

Table of Contents

List of Tables

List of Figures

List of Acronyms

CSV	Comma-Separated Values
DCV	Data Cube Vocabulary
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
M2RML	Multidimensional to RDF Mapping Language
MD	Multidimensional
OLAP	Online Analytical Processing
OWL	Web Ontology Language
R2RML	RDB to RDF Mapping Language
RDF	Resource Description Framework
RDFS	RDF Schema
SCOVO	Statistical Core Vocabulary
SDMX	Statistical Data and Metadata Exchange
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Chapter 1. Introduction

1.1. Web of Data and Linked Data

The World Wide Web has provided us with a global information space and changed the way we share information by making it easier to publish and access documents (Bizer, Heath, & Berners-Lee, 2009). Hypertext links allow users to traverse this global information space using web browsers and search engines direct us to the most relevant documents by indexing them and analyzing the structure of links between them (Brin & Page, 1998). This functionality is a key feature of the Web's unconstrained growth and is enabled by generic, open and extensible nature of the Web (Jacobs & Walsh, 2004)(Bizer, Heath, & Berners-Lee, 2009).

These principles that led to existence of the Web containing huge amounts of documents have not been applied to data until recently. Traditionally data published on the Web could be accessed via raw CSV dumps, HTML tables or XML; however, this is being changed as the Web has evolved to be a global information space containing documents and data rather than just documents. This evolution is created by a set of best practices for publishing and connecting structured data on the Web called Linked Data (Bizer, Heath, & Berners-Lee, 2009).

An increasing number of data providers have adopted these principles which have led to creation of a Web of Data as a global data space containing billions of assertions (Bizer, Heath, & Berners-Lee, 2009). The Web of data enables new kind of applications such that a user can start by browsing a dataset and navigate links that lead to data in another dataset; or a user can use web of data crawlers to collect aggregated data from different sources and then write expressive queries on that or build domain specific applications that access multiple and various datasets related to that domain. As mentioned in (Bizer, Heath, & Berners-Lee, 2009) unlike Web 2.0 in which users were restricted to use a fixed set of data sources, the Web of data allows them to access an

unlimited global data space which enables them to deliver more complete answers to questions as more data becomes published and available on the Web.

Linked data in brief is using the Web to create typed links between data from different sources. In comparison to web of documents where HTML documents are linked by un-typed hyperlinks, web of data includes data in Resource Description Framework (RDF) (Graham & Caroll, 2004) format which allows typed links between resources or things (Bizer, Heath, & Berners-Lee, 2009).

Linked data principles were defined by Tim-Berners Lee in (Lee, 2006) as below:

- Use URIs as names for things.
- Use HTTP URIs, so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL the RDF query language).
- Include links to other URIs, so that they can discover more things.

Figure 1.1 taken from (Obitko, 2007) shows the Semantic Web Architecture or Semantic Web Stack. As mentioned in (Abraham, 2013), the figure illustrated by Tim-Berners Lee shows the languages and standard technologies organized to make the semantic web a reality. Each layer exploits capabilities of the layer below and the whole stack is evolving. According to (Abraham, 2013) the layer also shows that semantic web is an extension of the web of document or classical Hyper-Text web and not a replacement.

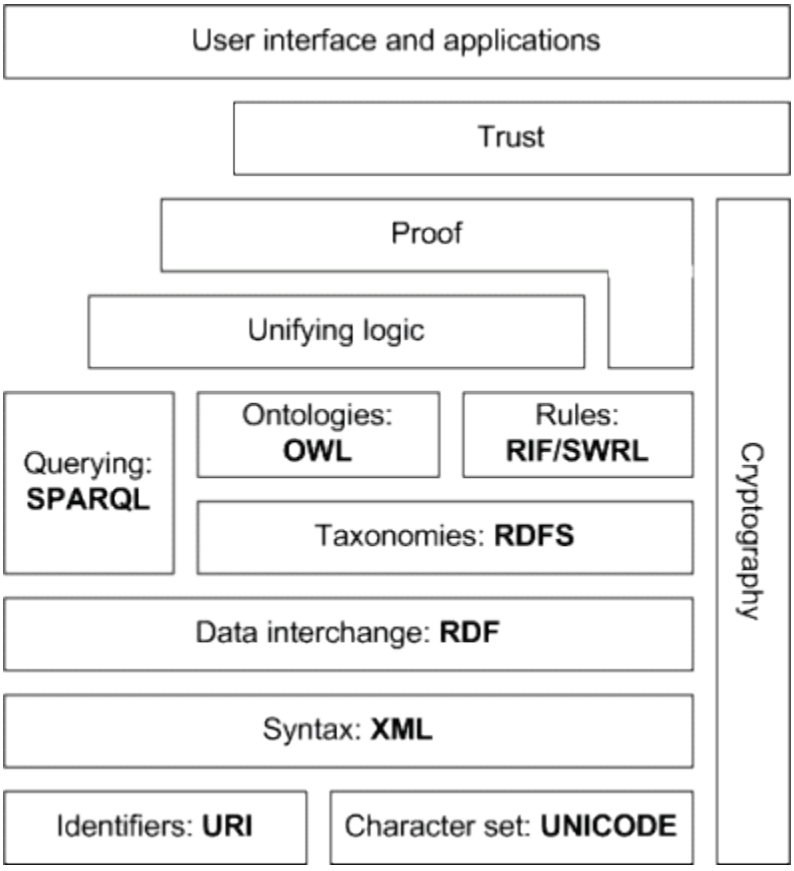

Figure 1.1. Semantic Web Stack

The layers up to OWL are standardized and can be used to implement semantic web applications however implementation of the top layers is still unclear (Abraham, 2013). Hence the stack can be divided into three sets of technologies Hyper-Text web technologies, semantic web technologies and unrealized semantic web technologies.

Hyper-Text technologies are 1) URIs to uniquely identify things and resource on the web, 2) UNICODE to represent and manipulate text in different languages and 3) XML abbreviated for Extended Markup Language to represent semantic web documents in a structured way.

Semantic Web Technologies are in the middle and are all standardized by W3C except RIF/SWRL. These technologies can be used to create semantic web applications and include 1) RDF (Resource Description Framework) as a simple graph-based model for representing data on the web, 2) RDFS (RDF Schema) which provides a vocabulary or schema to maintain proper structure of an RF document and to represent a proper

hierarchy of classes and properties, 3) OWL (Web Ontology Language) is used to add more constraints and meaning to the RDF representation and 4) SPARQL (a recursive acronym for SPARQL Protocol and RDF Query Language) an RDF query language for querying RDF documents and RDF data stored in a database which is used by semantic web applications to retrieve RDF data.

The third layer at top of the stack is called 'Unrealized Semantic Web Technologies' in (Abraham, 2013)and includes technologies that are not standardized yet or are ideas to be implemented to completely create semantic web applications. Such technologies are 1) RIF/SWRL (Rule Interchange Format/Semantic Web Rule Engine) is used to add rules to RDF data and to represent information that cannot be expressed directly by OWL, 2) Cryptography to ensure RDF statements coming from semantic web applications are coming from proper and identified sources, 3) Trust for statement support that premises com from trusted sources and relying on formal language to retrieve new information and 4) User Interface to make semantic web applications more user-friendly for humans.

Linked data uses two technologies fundamental to the web, Uniform Resource Identifiers (URI) and Hypertext Transform Protocol (HTTP) and another technology critical to the Web of Data, RDF or Resource Description Framework. URIs as mentioned above, allow us to uniquely identify things in the world, a subset of which Uniform Resource Locators (URL) used in the Web for uniquely locating HTML documents. HTTP is used for looking up things/entities by dereferencing URIs; and RDF provides a generic and graph-based model to structure and link data that describe things in the world (Bizer, Heath, & Berners-Lee, 2009).The following section provides more information on the RDF data model.

1.2. RDF and RDF Vocabularies

Resource Description Framework (RDF) is a simple graph-based data model (Graham & Caroll, 2004). RDF represents data in subject-predicate-object format called a triple. The subject of a triple is a resource identified by a URI and object of a triple can be a resource or a string literal. The predicate in a triple represents the relation between

subject and the object and is again identified by a URI. The best way to explain this is by introducing an example as below taken from (Bizer, Heath, & Berners-Lee, 2009).

Below we have shown a set of subject-predicate-object which forms a triple. RDF links (Bizer, Cyganiak, & Heath, 2007) take the form of RDF triples where subject of the triple is identified by a URI in namespace of a dataset and object of the triple identified by a URI in another (Bizer, Heath, & Berners-Lee, 2009). The example set of triple shows a membership relation between the subject and our object. It states that the resource 'Tim-Berners Lee' identified by the URI 'http://www.w3.org/People/Berners-Lee/card#i' is a member of 'http://dig.csail.mit.edu/data#DIG' group. Dereferencing these URIs over HTTP we will get back the resource descriptions from their respective servers. The predicate in this example is 'http://xmlns.com/foaf/0.1/member' which if dereferenced we will get back definition of the link type 'member' described in RDF using RDF Vocabulary Definition Language, RDFS (Brickley & Guha, 2004), that we introduce below.

- Subject: http://dig.csail.mit.edu/data#DIG
- Predicate: http://xmlns.com/foaf/0.1/member
- Object: http://www.w3.org/People/Berners-Lee/card#i

The RDF Vocabulary Definition Language (RDFS) (Brickley & Guha, 2004) and the Web Ontology Language (OWL) (McGuinness & Harmelen, 2004) are two main vocabularies that provide a basis for creating other vocabularies to describe things and relations. Vocabularies are themselves represented in RDF and are comprised of a set of classes (type of things) and properties (type of links). Vocabularies are built upon RDF an OWL for modeling domains of interest depending on expressivity of the model (OWL is more expressive than RDFS). According to (Bizer, Heath, & Berners-Lee, 2009) everyone can build and publish vocabularies on Web of Data which in turn can be connected to other vocabularies by linking classes and properties of the them. This provides a way to create a mapping between vocabularies.

1.3. Publishing Relational Data using R2RML

According to (Bizer & Cyganiak, 2006) most structured data is stored in relational databases and will still be maintained in relational databases in mid-future in spite of the progress and advancement in field of RDF and XML. However it would be nice and sometime required to link the data we already have to other datasets and information on the web. In fact the full potential of Linked Data initiative would highly depend on how easy is to publish the data that we already have, mostly in relational databases. This is where R2RML (Das, Sundara, & Cyganiak, 2012), Relational to RDF Mapping Language comes into play.

R2RML is a descriptive language that allows describing customized mappings from a relational database to a target RDF vocabulary. It provides means to describe the relation between the data that already exists in a relational database and the data that would be available once it's published in RDF. R2RML is itself a vocabulary and an RDF graph. It allows different implementation types; R2RML processors, the applications that provide accessibility to relational data in RDF format can materialize the output RDF data or can represent it as a virtual RDF graph while original data still resides in the relational database.

1.4. Multidimensional Data

Statistical data is a main element of interesting mash-ups and visualization we see. In another important usage and role, analysis of statistical data allows policy makers and managers to see trends, make predictions, plan and adjust their plan along the way (Cyganiak, Field, Gregory, Halb, & Tennison, 2010). So much of data we currently have is statistical data (Cyganiak, Field, Gregory, Halb, & Tennison, 2010) which we would like to publish based on linked data principals to get benefit. According to (Cyganiak, Field, Gregory, Halb, & Tennison, 2010) publishing these data will allow both publishers and third parties to build trust by annotating and referencing the data on the web.

A number of governments and international organizations have started to publish their data in RDF, some of which use Linked Data principles (Vrandecic, Lange, Hausenblas, Bao, & Ding, 2010). These organizations and governments deal with statistical data at national or global level. Rise of Web of Data created an increasing need for sharing and accessing and using open statistical data on the web (Hausenblas, Halb, Raimond, Feigenbaum, & Ayers, 2009) and several standards and frameworks were created by companies, organizations or standard bodies such as W3C. Such standards are SDMX (International Organisation for Standardisation, 2005), SCOVO (Hausenblas, Halb, Raimond, Feigenbaum, & Ayers, 2009) and the recently published Data Cube Vocabulary by W3C which is built upon SDMX, SCOVO and SKOS (Miles & Bechhofer, 2009) vocabularies.

Statistical Data and Metadata Exchange (SDMX) was an initiative in 2001 by seven international organizations (BIS, ECB, Eurostat, IMF, OECD, World Bank and the UN) to make publishing and consumption of statistical data more efficient. These organizations all collect considerable amount of data and also publish data at global and international level. Organizations like the U.S. Federal Reserve Board, the European Central Bank, Eurostat, the WHO, the IMF, and the World Bank use SDMX (Statistical Data and Metadata Exchange) to publish their data on the web.

The Statistical Core Vocabulary (SCOVO) was introduced in the paper “SCOVO – Using Statistics on the Web of Data” at European Semantic Web Conference in 2009(Hausenblas, Halb, Raimond, Feigenbaum, & Ayers, 2009). The paper proposed a framework for modeling, representing and sharing statistical data on the web. Following SCOVO, Data Cube Vocabulary (DCV) was proposed by (Cyganiak, Reynolds, & Tennison, 2010) to standardize publishing of statistical and multidimensional data on the web. This work was then continued in W3C Government Linked Data Working Group and became a W3C candidate recommendation in June 2013 (Cyganiak & Reynolds, 2013).

DCV builds upon SDMX for core statistical data model and content oriented guidelines, SCOVO for core statistical structures and SKOS (Simple Knowledge Organization System) for concept schemes. DCV allows representation of multidimensional data from different sources such as OLAP Cubes, spreadsheets and

survey data. According to (Cyganiak & Reynolds, 2013) DCV is the core foundation which supports extension vocabularies to enable more enhanced publication of multidimensional datasets.

1.5. Publishing Multidimensional Data using Data Cube Vocabulary

As relational databases have been the dominant method for storing structured data, OLAP cubes are probably the most common approach for multidimensional data and data analysis. As mentioned above Data Cube Vocabulary is designed to publish multidimensional data on the web however it's designed general enough so that data from other sources like spreadsheets and surveys can also be published.

Similar to the situation with relational data, with increasing interest in Web of Data it would be nice to have access to multidimensional data we already have in OLAP cubes with which DCV would help as a target and standardized vocabulary however there is a lack of tool such as R2RML for multidimensional data. This thesis is about design and implementation of a mapping language that allows us to describe mapping of existing multidimensional data in OLAP cubes to RDF datasets in Data Cube Vocabulary.

The main contribution of our work is Multidimensional to RDF Mapping Language (M2RML), a mapping language/vocabulary similar to R2RML but for multidimensional data which allows us to describe the relation between existing multidimensional data we already have and the target RDF dataset which can be published and shared on the Web of Data. M2RML is designed to work with DCV as the target vocabulary while it allows mapping from different source such as OLAP cubes, spreadsheets and surveys.

As OLAP cube is the most common source of multidimensional data we focus on one-to-one mapping of OLAP cube elements with DCV elements and discuss limitations of DCV for representing such data. To solve these limitations we provide an extension to Data Cube Vocabulary which facilitates this one-to-one-mapping. Unlike other approaches to extend DCV for representing OLAP cubes we design our elements based on abstract classes of DCV to make our vocabulary/extension compliant with DCV.

1.6. Organization of this Document

The rest of this document is organized as follows.

In chapter 2 we discuss related work in publishing and interacting with relational and multidimensional data on Web of Data. In chapter 3 we analyze Data Cube Vocabulary more in detail, discussing main elements of the vocabulary, what they represent and how different elements are related to each other.

In chapter 4 we review a general framework and architecture for transforming multidimensional data to linked data as well as general frameworks for publishing and consuming multidimensional linked data.

In chapter 5 we introduce M2RML, our mapping language and we discuss its elements in detail. As well we provide details on our approach to design the language and its differences and similarities with R2RML.

In chapter 6 we provide a use case and represent AdventureWorks_DW2012 OLAP cube as an RDF/QB graph. We will also identify limitations of Data Cube Vocabulary in representing multidimensional data and we introduce our extension to the vocabulary for overcoming these limitations which enables one-to-one mapping of OLAP cubes and RDF/QB datasets. Finally in chapter 7 we will conclude our work by summarizing our contributions to the field and discussing a possible future work.

Chapter 2. Related Work

Literature in this area of work can be divided into three groups, vocabularies, triplification of relational/multidimensional data and frameworks for publishing statistical data. In the Vocabularies section below we talk about Data Cube Vocabulary, related vocabularies to DCV and their relation; in Triplification and Publication of Statistical Data we discuss related works where the main focus is on mapping of data to RDF datasets, generating RDF triples as well as methods and approaches for publishing data. Papers in this section can be both similar and at the same time distinctive. This is because some use Data Cube Vocabulary as is and focus on mapping approach/scheme while some others propose new vocabularies or extensions to Data Cube Vocabulary for publishing multidimensional data as needed.

2.1. RDF Vocabularies

Publishing and representing statistical data as RDF started by SCOVO and continued by its successor Data Cube Vocabulary (Ruback, Pesce, Manso, Ortiga, Salas, & Casanova, 2013). According to DCV Document (RDB2RDF Working Group, 2012), the Data cube vocabulary is originated outside of W3C and then continued within Government Linked data group. The original DCV document or its draft was published in 2010 by (Cyganiak, Reynolds, & Tennison, The RDF Data Cube vocabulary, 2010) in which it's mentioned motivation of DCV was limitations of SCOVO and that the design of DCV is informed by SCOVO as it's used to build the core statistical structures. In fact every SCOVO dataset can be expressed with DC Vocabulary according to (Cyganiak, Reynolds, & Tennison, 2010).

Another building block of DCV is SDMX which was an initiative started in 2001 by organizations who mostly deal with statistical data at national or global level. DCV uses

SDMX for its core information models as well as content oriented guidelines. Following is a more detailed description of each vocabulary.

2.1.1. Data Cube Vocabulary (DCV)

One of the most important related tools available in semantic web area is Data Cube Vocabulary, a W3C candidate recommendation for publishing multidimensional and statistical data. The vocabulary is in turn built upon other existing vocabularies among which we can mention:

- SKOS for concept schemes
- SCOVO for core statistical structures
- SDMX for content-oriented guidelines and core information model

DCV provides a vocabulary including elements for representing a multidimensional dataset which consists of measured values represented across group of dimensions and along with their related metadata. Building upon SDMX and SCOVO it also provides standard and shared code-lists, content-oriented guidelines and concept schemes as members and available values for a dimension attribute. Beside these, it also provides means for defining slices and groups of related observations as well as dataset metadata and provenance information (using Dublin Core vocabulary).

2.1.2. SDMX

Statistical Data and Metadata Exchange was an initiative by seven international organizations (BIS, ECB, Eurostat, IMF, OECD, World Bank and the UN) in 2001 to make publishing, sharing and using of statistical data and metadata more efficient. According to DCV document these organizations all collect considerable amount of data usually at national level and publish it at national and beyond national boundaries. SDMX has produced two standards/specifications which is widely accepted and employed by large organizations such as UN Statistical Commission. Data Cube Vocabulary according to its documentation is built upon the code information model of SDMX.

2.1.3. SCOVO

The Statistical Core Vocabulary (SCOVO) was introduced in the paper "SCOVO – Using Statistics on the Web of Data" at ESWC in 2009(Hausenblas, Halb, Raimond, Feigenbaum, & Ayers, 2009). The paper proposes a framework for modeling, representing and sharing statistical data on the web and is used as the core of Data Cube Vocabulary. In the paper authors introduce elements for representing datasets, dimensions, items, events and etc. which later are refined as elements of Data Cube Vocabulary. The website for this vocabulary, http://vocab.deri.ie/scovo, is last modified at 9 August 2012 which mentions the vocabulary is deprecated and users should instead use Data Cube Vocabulary.

2.1.4. SKOS

Simple Knowledge Organization System (SKOS) is a common data model for sharing and linking knowledge organization systems such as thesauri, classification schemes, and subject heading systems and taxonomies on the web. It is a W3C recommendation and is built upon RDF and RDFS vocabulary. The latest SKOS data model is formally defined as an OWL Full ontology and it expresses data as RDF triples. The SKOS data model view knowledge systems as concept schemes comprised of a set of concepts which can be shared between applications. Concepts can be linked and mapped together in other concept schemes as well as being grouped into collections to be labelled and/or ordered. Data Cube Vocabulary uses SKOS concept schemes as code-lists for available and acceptable values of a dimension attribute.

2.2. Triplification of Relational Data

Most of the works and approaches that involve OLAP cubes and RDF/QB cubes, try to build multidimensional model from published RDF/QB cubes and/or define a new vocabulary to support mapping of OLAP cube elements and further OLAP cube operations on RDF/QB datasets.

As mentioned in (Ruback, Pesce, Manso, Ortiga, Salas, & Casanova, 2013) most of the work in publishing data using DCV use relational databases as their source and

less is done on OLAP cubes as the source of data. As we mentioned before OLAP is one of the most used approaches for analytical data processing and there is already vast amount data modeled and analyzed using this approach. Hence, there is need for tools and frameworks which ease transformation and publishing of data from OLAP cubes as source of multidimensional data.

This thesis is strongly influenced by R2RML in two respects. First and foremost, it shows us how to construct, in a concrete way, a mapping language. Secondly, it is together with other related works in triplification and mapping of relational data such as Triplify (Auer, Dietzold, Lehmann, Hellmann, & Aumueller, 2009) in a form of simple script, D2R Server (Bizer & Cyganiak, 2006) as a standalone solutions and Virtuoso RDF Views (Erling, 2009) as an integrated enterprise tool.

R2RML (RDB2RDF Working Group, 2012), RDB to RDF Mapping Language is W3C recommendation for expressing mappings of data in relational databases to RDF datasets. R2RML became a recommendation in September 2012 after going through extensive community review and revision which made it stable enough for wide spread in commodity software (Franzon, 2012).

R2RML provides a model to produce triple maps that can be processed for generating triples based on their specific subject, predicate and object mappings. It also provides elements to map data coming from different sources such as relational databases, RDF dumps, SQL Queries and views while allowing templates for generating URIs of subject, predicate and object of an RDF triple. We use the approach in R2RML for building a mapping language for mapping multidimensional data to RDF datasets. This similar structure to a standard makes the language familiar, more understandable and more adoptable.

The input to R2RML is a database schema and the output of it is an RDF dataset corresponding to the input schema and with classes and properties (types and predicates) of the target vocabulary. Every mapping is specific to a source schema and a target vocabulary and every mapping is itself an RDF graph represented by RDF triples. R2RML allows a mapping author to define flexible and customizable views over the relational data. According to the document an R2RML processor can materialize

resulting RDF triples from mapping or can expose them virtually as views while data still resides in the relational database.

On importance of having such a mapping language, Richard Cyganiak, one of the editors of (RDB2RDF Working Group, 2012) mentions that mapping languages such as R2RML bridge the gap between large amount of data already stored/available in relational databases and semantic web; which is aligned with the goal of interconnecting all the data and make semantic web and its technologies more useful.

According to R2RML document (RDB2RDF Working Group, 2012) the specification has also a companion that defines a direct mapping from relational databases to RDF (RDB2RDF Working Group, 2012). As mentioned in the document “In the direct mapping of a database, the structure of the resulting RDF graph directly reflects the structure of the database, the target RDF vocabulary directly reflects the names of database schema elements, and neither structure nor target vocabulary can be changed”. The flexibility and customizability in R2RML enables us to generate a default mapping which can then be enriched by a mapping author. This flexibility makes R2RML the relaxed version of the direct mapping and Eric Prud’hommeaux, a major contributor to development of R2RML, believes this makes it easier for users to adopt and start using R2RML while direct mapping would be more a long-term goal and for future usages. (Franzon, 2012)

Figure 2.1 shows overall view of R2RML classes in which TriplesMap is a central element in R2RML which defines a resource map by specifying specifications of the resource as a SubjectMap and its related property-value sets as PredicateObjectMap.

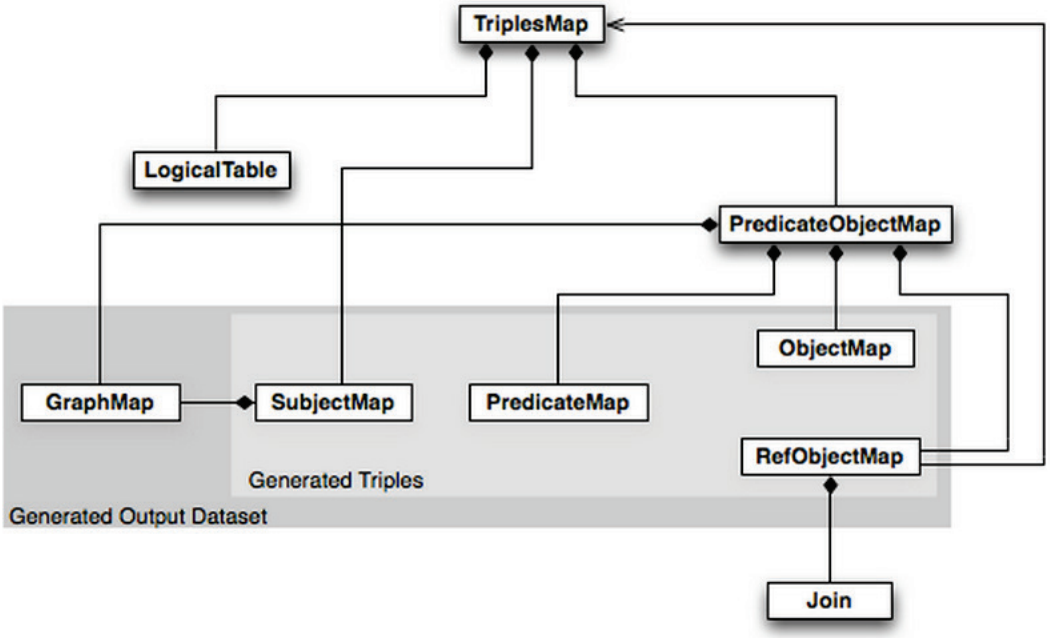

Figure 2.1. Overview of R2RML

Similar to R2RML, in 2004 (Bizer & Seaborne, 2004) proposed a declarative language called D2RQ to represent mapping of non-RDF application specific data to RDF data in RDFS or OWL ontologies. The motivation behind their work was the growing need for RDF data to access live, legacy non-RDF data without replicating the whole database in RDF. Similar to R2RML resulting RDF data/triples can be materialized or represented as virtual RDF graphs/views. Figure 2.2 shows application architecture using D2RQ mapping language.

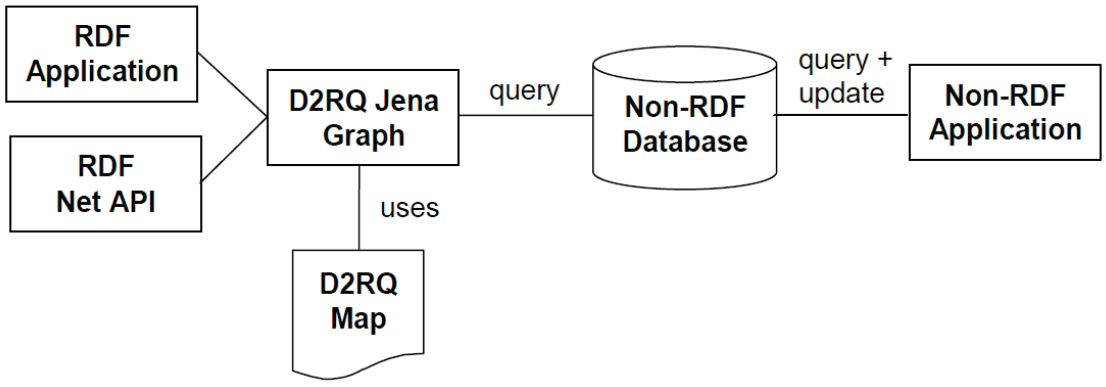

Figure 2.2. Application Architecture using D2RQ

In 2006, Bizer and Cyganiak introduced a standalone solution to publish and expose relational data in RDF called D2R Server (Bizer & Cyganiak, 2006) which uses D2RQ as its mapping language. As shown in figure 2.3, their solution makes relational data accessible to web agents, RDF browsers, SPARQL clients and other linked data users to access underlying data with acceptable response times resulted from on-the-fly translation and mapping of relational data facilitated by D2RQ mapping language. According to them a ClassMap is a central objet in D2RQ similar to TriplesMap in R2RML which describes mapping of set of entities in a database to similar classes/types of resources in RDF graph. Further each ClassMap has a set of PropertyBridges to represent properties and values corresponding to that ClassMap. D2R Server has a tool enabling users to automatically generate default mappings using table names as class/type names, column names as property names, and column values as property values which can later be edited and customized.

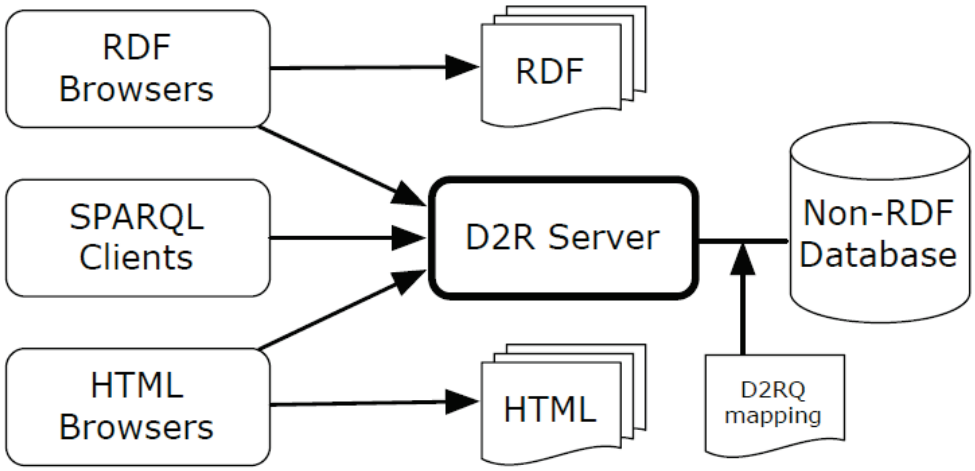

Figure 2.3. Overview of Framework using D2R Server

For the same reason Richard Cyganiak believes a mapping language for relational data to RDF data is important, we believe having a mapping language for multidimensional and statistical data to RDF data is important. As we mentioned in the previous chapter there are large amount of data hosted and analytically processed by OLAP engines. This mapping language in the same way can enable web agents and RDF data consumers to access statistical data and enrich their analysis using available data from other datasets on the web. Since we use the same approach and core elements of R2RML in our proposed mapping language, it enables users to materialize

resulting RDF graph or expose it as virtual RDF views/graphs and makes it easier for user to adopt and use the language.

2.3. Triplification of Multidimensional Data

In this section we discuss other related works which explicitly focus on mapping and publishing of multidimensional and statistical data.

Noticeable and related works in Triplification of multidimensional data are (Ruback et. al., 2013), (Moreira & de Freitas Jorge, 2012) and (Salas, Martin, Mota, Breitman, Auer, & Casanova, 2012) which all assume relational databases as their data source. In (Ruback, Pesce, Manso, Ortiga, Salas, & Casanova, 2013) authors propose a mediation architecture featuring a catalogue of linked data cubes created according to linked data principles. They mention that the catalogue is just the description of data cube which doesn't include fact records or observations and therefore it's not a complete materialization of the underlying cube from relational database. Figure 2.4 taken from their paper shows the mediation architecture they propose.

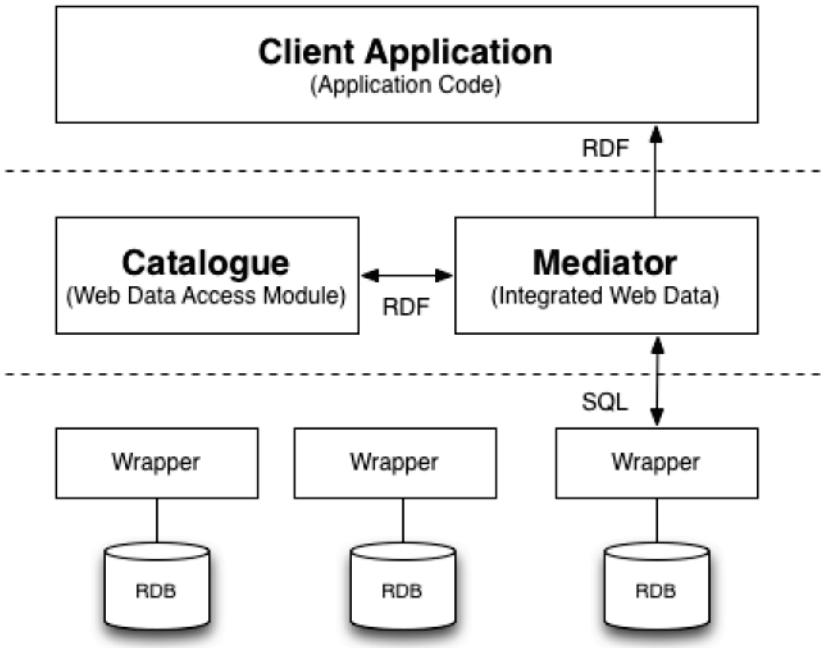

Figure 2.4. Mediation Architecture for Statistical Linked Data

In this architecture the catalogue includes cube definitions and metadata such as dimensions and attributes but leaves the fact records or observations in the database layer. A client application accesses the catalogue to select a desired cube and requests facts data from the mediator. The mediator then accesses the wrapper based on the cube selected by the application to retrieve the facts. They assumed data in relational databases is not necessarily in star schema and transformation of data to star schema useful to OLAP systems depends on the wrappers which access relational databases using SQL and transform requested fact records to Data Cube Vocabulary observation RDF triples. In this paper they have focused on overall architecture of mediator and don't go into details of processing a request for accessing data in RDF cube format from an underlying star schema or OLAP cube.

In (Salas et. al., 2012) authors present a plug-in for OntoWiki (Auer, Dietzold, & Riechert, 2006) called OLAP2DataCube which again transforms data from a relational database with star schema. Their mapping scheme uses foreign keys and relationships between fact and dimension tables. Using OntoWiki as their platform they benefit from having a user interface and other OntoWiki features like ontology validation and exposing data as an end-point.

In a different work (Moreira & de Freitas Jorge, 2012) proposes a method to translate SPARQL queries to MDX which can be used to access data in data warehouses and OLAP cubes using SPARQL. Unfortunately we don't have more detail on this work as it's in Portuguese and we couldn't find an English version. However assuming existence of such tool and assuming availability of OLAP cube metadata in RDF format we can have a system that stores and processes fact data in an OLAP system and exposes and give access to data via SPARQL end-point.

2.4. Vocabularies and Frameworks for Publishing and Using Multidimensional Data in RDF

Following these, now we discuss other related works where in addition to publishing multidimensional and statistical data using Data Cube Vocabulary, authors

propose new vocabularies or extensions to DCV to enable or enhance OLAP operations on linked data.

In (Etcheverry & Vaisman, 2012) authors work with multidimensional tables and introduce a vocabulary called Open Cube to enable OLAP operations such as rollup and drilldown on multidimensional data in RDF format. The vocabulary introduced is based on RDFS and the motivation behind creating a new vocabulary is mentioned as limitations of DCV in 1) granularity of dimensions (as OLAP dimensions can include related attributes with different granularities such as a date dimension with year, month and day attributes whereas DCV dimensions has single granularity), 2) Representing hierarchies at dimension level, schema level or instance level and 3) Differences between concepts such as slices (where in OLAP cube a slice definition is fixing one or multiple dimension but in DCV a slice is fixing only one dimension).

The authors later in another work (Etcheverry & Vaisman, 2012) address the issue with Open Cube Vocabulary as not compatible with and not being able to republish datasets that are already published in Data Cube Vocabulary. In this paper they present another vocabulary called QB4OLAP, this time as an extension of Data Cube Vocabulary which supports OLAP operations on statistical RDF data while being compatible with cubes already published in Data Cube Vocabulary. They mention the compatibility of OLAP operations and datasets in published using DCV comes with a low price of only adding dimension information. In this paper they present additional elements required in the vocabulary to enable OLAP operations as levels, relationship between members and levels and also relationship between levels and dimensions. In this work a hierarchy is defined by a set of related levels.

In (Kampgen & Harth, 2011)authors introduce a framework for mapping statistical data in RDF to multidimensional data in OLAP cube. The motivation behind their approach is that as statistical data is become more available everyday on the web as 1) Different pieces of a desired dataset might be distributed on the web, 2) Dataset endpoints might not be always available and 3) Different schemas might have been used for publishing datasets which makes identifying equal elements of a dataset a challenge.

Therefore they mention it would be more efficient to transform data in RDF to multidimensional data and load them into OLAP systems for further analysis. Unlike(Etcheverry & Vaisman, 2012) and instead of creating a new vocabulary, Kampgen and Harth consider data published in Data Cube Vocabulary as their source data and try to map elements of data cube vocabulary to common elements of an OLAP cube. In their mapping of QB (an RDF cube in Data Cube Vocabulary) to OLAP cube elements there is no direct equivalent in DCV for elements such as hierarchies and levels. This was also mentioned in (Etcheverry & Vaisman, 2012) as a limitation of DCV. In our work we identified and address this limitation too however instead of coming up with a whole new vocabulary we will add a few required elements to DCV to solve the problem with:

- Single granularity of dimensions
- Representation of hierarchies, levels and members
- As well as representation of the relation between dimensions, hierarchies, levels and members in a similar way they exist in OLAP cube model

Following their 2011 paper, authors in (Kampgen, O'Riain, & Harth, 2012) show how OLAP operations can be performed on linked data published in Data Cube Vocabulary. In this work they address drawbacks of their previous model as 1) Dependency on a ROLAP engine to execute OLAP queries on multidimensional data loaded from published linked data cubes and 2) Difficulty of maintaining converted multidimensional data up to date. If a single fact is modified or changed in the RDF/QB cube the proposed ETL process should be performed again to generate the data in multidimensional model or to update the changed record.

To overcome these shortcomings they present how OLAP operations can be nested to form an OLAP query and how that query can be translated into a SPARQL to retrieve and generate required fact RDF triples from RDF/QB cube. Although they present a new scheme, they discuss that their approach is still limited as they have assumed a single hierarchy per dimension; a common limitation addressed by other previously mentioned papers.

As can be seen, in all these works authors consider a multidimensional model but assume data exists in a relational database which is different from our approach in

which we think the data is in multidimensional format and coming from an OLAP cube under management of an OLAP server.

There are two implications due to the difference in architecture. First, according to our model, there is a constraint on the kind of information we can access from the server. For example, we cannot get access to the relational data in star schema. The only information accessible by a program is the data and metadata of the data cube via XMLA. Secondly, using OLAP cube as source and considering that fact data can be exposed as virtual RDF graph, we can leave fact data and observations in OLAP cube which results in storage efficiency and as well we can shift aggregation/summarization load into OLAP engine (rather than on RDF data) and just represent summarized data in RDF to be linked to other summarized/aggregated or non-aggregated data in web of data. This is desirable as OLAP cubes are designed to deal with such calculations and there are already tools do such tasks efficiently whereas RDF storage schemes and SPARQL are general and not as efficient as OLAP engines in aggregation, data summarization and data analysis.

Chapter 3. Analyzing Data Cube Vocabulary

Data Cube Vocabulary (DCV) is an RDF vocabulary to provide necessary means to publish statistical and dimensional data in semantic web and RDF format. Based on the document supporting this vocabulary (Cyganiak & Reynolds, 2013), it is designed to be general and thus it can be used for statistical and dimensional, survey data, spreadsheets and OLAP data cubes.

In the same document it is also mentioned that extensions to this vocabulary might be needed to support publication of additional context to statistical and dimensional data. One of these extensions can be a model to make OLAP cubes data easily publishable in RDF format. The motivations behind this extension are listed below:

- Many statistical and dimensional data is processed using OLAP cubes
- DCV is general and doesn't provide enough to
    - Publish OLAP cube data in a way that includes all necessary elements
    - Perform operations such as drill-down and roll-up in OLAP engines

Having a vocabulary which can define crucial elements of an OLAP cube and their relations would help data analysts to have advantages associated with linked data while having some depth to their analysis as available in OLAP cubes/engines.

Following defines elements of DCV, how it can provide a one to one mapping to OLAP cube elements, what they can represent and how elements are related together.

3.1. Data Cube Vocabulary

DCV is intended to be for statistical datasets which are defined as follows. A statistical dataset is a set of observed values collected from a logical space meaning that the observation can be viewed from different aspects. The values gathered are called measures (e.g. life expectancy, price, quantity, etc.) and aspects are called dimensions

(e.g. time, age, geography, etc.). These values can also have metadata associated with them called attributes providing information like how they are measured and the unit in which they are measured/represented.

The statistical dataset organized in this way is called a cube consisting of dimensions, measures and attributes which in data cube vocabulary are called components. Dimension components are used to identify observations. We can identify an observation having values for all dimensions. Measures represent the phenomenon measured and attribute components are used to interpret and understand measures better.

3.2. Data Cube Vocabulary Elements

3.2.1. Dataset

Datasets in DCV are represented by a resource representing the entire data set. This resource gives a brief description on what the dataset contains/represents and will have a property which defines how the dataset is structured.

DCV-Dataset is represented by a qb:Dataset class and has a property qb:structure whose value defines its structure. Following is an example from (Cyganiak & Reynolds, The RDF Data Cube Vocabulary - W3C Candidate Recommendation, 2013).

```
eg:dataset-le1 a qb:DataSet;

      rdfs:label "Life expectancy"@en;

      rdfs:comment "Life expectancy within Welsh Unitary
authorities - extracted from Stats Wales"@en;

      qb:structure eg:dsd-le1 .
```

3.2.2. Data Structure Definition

A data structure definition defines how the dataset is structured and defines the elements of the dataset and optionally their orders. Elements of a statistical dataset can be dimensions, measures and attributes.

DCV-Data Structure Definition is represented as qb:DataStructureDefinition and lists dimensions, measures and attributes of a dataset. A qb:DataStructureDefinition serves as value of qb:structure property of a qb:Dataset resource. Following is an example from (Cyganiak & Reynolds, 2013):

```
eg:dsd-le a qb:DataStructureDefinition;

      # The dimensions

      qb:component [ qb:dimension eg:refArea;
qb:order 1 ];

      qb:component [ qb:dimension eg:refPeriod;
qb:order 2 ];

      qb:component [ qb:dimension sdmx-dimension:sex;
qb:order 3 ];

      # The measure(s)

      qb:component [ qb:measure eg:lifeExpectancy];

      # The attributes

      qb:component [ qb:attribute sdmx-
attribute:unitMeasure;

            qb:componentRequired "true"^^xsd:boolean;

            qb:componentAttachment qb:DataSet;] .
```

3.2.3. Component Property

Component property is an abstract class that represents elements of a statistical cube. Dimensions, measures and attributes are subclasses of this class. A component property provides the following information:

- The concept being represented

- The nature of the component
- The type or code list used to represent the values

Sometimes the same concept can be represented as two different components. For instance the measuring unit of an age can be represented as a dimension or an attribute. To make the concept reusable as either of the components DCV provides a class to represent concepts and a property which links the concept to the component property.

Possible values for a component property might be drawn from a list or a collection, in which case they can be defined using Concept Schemes, Collections or Hierarchical Code Lists in DCV. These collections of values would be attached to the component using a property. The comprising elements of these collections/lists would be all concepts. (It would be a good idea to define a class and link all comprising concepts to that class so that it can be used for defining possible range of values of the component).

DCV-Concept being represented by a qb:ComponentProperty is defined as a skos:Concept and instances of this class can serve as values of skos:concept property of a qb:ComponentProperty. Having instances of skos:Concept class enables us to define range of accepted values for the qb:ComponentProperty using rdfs:range property of the component (qb:ComponentProperty).

DCV-Component Property is represented as a qb:ComponentProperty and instances of this class would serve as value of qb:componentProperty (sub-properties qb:dimension, qb:measure, qb:attribute) property of a qb:ComponentSet (sub-class qb:DataStructureDefinition).

DCV-Concept Scheme is represented by skos:ConceptScheme class and instances of this class serve as values of qb:codeList property of a qb:ComponentProperty (or sub-classes qb:DimensionProperty, qb:MeasureProperty, qb:AttributeProperty).

DCV-Hierarchical Code List is represented by qb:HierarchicalCodeList and similar to qb:ConceptScheme, instances of this class server as values of qb:codeList property of a qb:ComponentProperty.

3.2.4. Dimension, Measure, Attribute

Dimensions represent aspects for which data/values is measured, for example we can study life expectancy from aspects such as time, gender and geography. As an example we can have a value for life expectancy (in number of years) for male and females during years, 2000-2010 in Asian countries.

The values gathered are called measures (e.g. life expectancy, price, quantity, etc.). Measure components represent the phenomenon measured.

Dimension components are used to identify observations. We can identify an observation having values for all dimensions. Attribute components are used to interpret and understand measures better.

We mentioned before that the same concept can be represented as different components in Data Cube Vocabulary. For instance if the data is as in table 3.1 and all records have the same measure type (year in this example) then we can have the attribute attached once at the dataset level for all observations however if we have data as in table 3.2 then the unit column can be a dimension itself.

Table 3.1. A Measure with same Unit for all Records

Age Unit	Age
Year	2
Year	31
Year	65

Table 3.2. A Measure with different Units for Records

Age Unit	Age
Month	2
Day	31
Year	65

For such cases where age unit is the same for all records, we are able to define the measuring unit at dataset level. In Data Cube vocabulary by default values of attribute components will be attached to each observation which is called the normalized representation. However it is also possible to define attach attributes to other levels such as the dataset, slices or to each individual measure.

As mentioned in the DCV document, normalized representation allows each observation to stand alone so that SPARQL endpoints can interpret observation without requiring referring to other elements. On the other hand, the non-normalized way allows reducing some redundancy in representing data. To enable this abbreviated structure, in component specification, we use a property that defines component attachments and the value of which would be the desired attachment level (to which we want to attach the attribute).

DCV-Component Specification is represented by qb:ComponentSpecification and instances of this class will serve as qb:component property of a qb:DataStructureDefinition. In the simplest case qb:ComponentSpecification references the corresponding qb:ComponentProperty (usually using one of the sub properties qb:dimension, qb:measure or qb:attribute), however it's also possible to have it more complex.

DCV-Dimension, Measure and Attributes are specific qb:ComponentProperties and are represented by qb:DimensionProperty, qb:MeasureProperty and qb:AttributeProperty. Instances of these classes will serve as values of qb:dimension, qb:measure and qb:attribute property of a qb:ComponentSpecification.

DCV-Component Specifications can have other properties like qb:componentRequired and qb:order and also qb:componentAttachment whose value should be an instance of class qb:Attachable and represents the attachment level (as discussed above).

We should note that qb:DataStructureDefinition and qb:ComponentSpecification are sub-classes of qb:ComponentSet. Elements qb:DimensionProperty, qb:MeasureProperty and qb:AttributeProperty are sub-classes of qb:ComponentProperty. Properties qb:dimension, qb:measure, qb:attribute are sub properties of qb:componentProperty. Property qb:componentProperty is a property of qb:ComponentSet with rdfs:range as qb:ComponentProperty.

3.2.5. Concept Scheme

Although we briefly discussed Concept Schemes and Code Lists we will have another look at them in this section.

Dimensions of a dataset should have defined unambiguous values. These values should be typed values (like xsd:integer, xsd:dateTime, etc.) or drawn from code lists which will be defined by a RDF resource.

The code list might already exist as SDMX Content Oriented Guidelines (COGs) in which case we just link it to the dimension using a property. If COGs are not helpful and the list is not defined already in other datasets, we use recommended SKOS vocabulary to create such lists.

The individual members of the Concept Scheme would be SKOS concepts and as mentioned before, it's good to have a separate class representing list members. This class will be used to define range of values a dimension can have. To represent hierarchies with SKOS, we should use skos:narrower and skos:broader properties for members of the list.

Hierarchical Code List is similar to SKOS Concept Scheme in which property names are different. These code lists can represent hierarchies too by using similar properties to SKOS narrower and broader properties.

DCV-Concept Scheme, as mentioned before, is defined by skos:ConceptScheme, instances of which will serve as values of qb:codeList property of qb:ComponentProperty (and sub-classes qb:DimensionProperty, qb:MeasureProperty, qb:AttributeProperty). skos:prefLabel property gives a name to the code, skos:note describes it and skos:notation is used to give the code a short name. skos:ConceptScheme uses a property skos:hasTopConcept to link to individual members of the list. Values of this property will be instances of skos:Concept class.

DCV-Concept is defined by skos:Concept class and instances of which can be values of qb:hasTopConcept property of a skos:ConceptScheme. A skos:Concept can be included in a skos:ConceptScheme using skos:inScheme property, values of which will be an instance of skos:ConceptScheme. And if the skos:Concept instance is the top concept of the list, we will use the property skos:topConceptOf whose value is a skos:ConceptScheme. skos:narrower and skos:broader will be used to define parent or child levels.

According to SKOS vocabulary “The property skos:hasTopConcept is, by convention, used to link a concept scheme to the SKOS concept(s) which are topmost in the hierarchical relations for that scheme. However, there are no integrity conditions enforcing this convention.” So the graph below is consistent with SKOS:

```
<MyScheme> skos:hasTopConcept<MyConcept> .

<MyConcept> skos:broader<AnotherConcept> .

<AnotherConcept> skos:inScheme<MyScheme>
```

DCV-Hierarchical Code List is defined by qb:HierarchicalCodeList and instances of this class can serve as values of qb:codeList property of a qb:ComponentProperty (and sub-classes qb:DimensionProperty, qb:MeasureProperty and qb:AtributeProperty). Individual members of the code list will be skos:concepts and are linked to qb:HierarchicalCodeList using its qb:hierarchyRoot property which plays the same role as skos:HasTopConcept. Hierarchies in qb:HierarchicalCodeList are built using qb:parentChildProperty property.

Following is an example of a skos:ConceptScheme, its individual members and the class representing all constituting members:

```
sdmx-code:sex a skos:ConceptScheme;

      skos:prefLabel "Code list for Sex (SEX) - codelist
scheme"@en;

      rdfs:label "Code list for Sex (SEX) - codelist
scheme"@en;

      skos:notation "CL_SEX";

      skos:note "This  code list provides the gender."@en;

      skos:definition <http://sdmx.org/wp-
content/uploads/2009/01/02_sdmx_cog_annex_2_cl_2009.pdf> ;

      rdfs:seeAlso sdmx-code:Sex ;

      sdmx-code:sex skos:hasTopConcept sdmx-code:sex-F ;

      sdmx-code:sex skos:hasTopConcept sdmx-code:sex-M .



sdmx-code:Sex a rdfs:Class, owl:Class;

      rdfs:subClassOf skos:Concept ;

      rdfs:label "Code list for Sex (SEX) - codelist
class"@en;

      rdfs:comment "This  code list provides the
gender."@en;

      rdfs:seeAlso sdmx-code:sex .



sdmx-code:sex-F a skos:Concept, sdmx-code:Sex;

      skos:topConceptOf sdmx-code:sex;

      skos:prefLabel "Female"@en ;

      skos:notation "F" ;

      skos:inScheme sdmx-code:sex .
```

```
sdmx-code:sex-M a skos:Concept, sdmx-code:Sex;
      skos:topConceptOf sdmx-code:sex;
      skos:prefLabel "Male"@en ;
      skos:notation "M" ;
      skos:inScheme sdmx-code:sex .
```

Following is how the qb:ConceptScheme created is linked to a qb:DimensionProperty:

```
eg:sex a qb:DimensionProperty, qb:CodedProperty;
      qb:codeList sdmx-code:sex ;
      rdfs:range sdmx-code:Sex .
```

3.2.6. Slice

In defining the cube and its components we mentioned that dimensions are like different aspects from which we can look at data and we also mentioned having a set of values for all dimensions we can identify an observation. Now what if we want to take a look at data from one or more specific aspects?

To do this we have to fix some of our dimensions and look at data from the desired aspect(s). For example we will fix our time dimension and look at data by gender (time and age would be our dimensions); this way we will have observations grouped by gender for specific time(s); like number of workers in 2008 for each gender (male, female, etc.). Fixing some dimensions and looking at data from other dimensions is defines slices.

Having slices enables us to look at subset of data which might ease to see trends and how measures are changed by changing values of a dimension for a fixed set of values for other dimensions.

Slices in the vocabulary have a structure linked to them called the Slice Key and in which we define which dimensions would be fixed. And then in the Slice definition itself we mention the specific values of the dimensions fixed. Slice key is attached to a Data Structure Definition and Slice is linked to the dataset itself so that we know what Slices the Dataset includes. We can also directly attach observations to the Slice using a property.

DCV-Slice Key is defined by qb:SliceKey and is linked to a Slice using its qb:sliceStructure property. A qb:SliceKey defines fixed dimensions using qb:componentProperty whose value will be instances of qb:DimensionProperty. A qb:SliceKey is linked to a qb:DataStructureDefinition using its qb:sliceKey property.

DCV-Slice is defined by qb:Slice and instances of this class will be linked to a qb:Dataset using qb:slice property. A qb:Slice will use its qb:sliceStructure property to link with a qb:SliceKey. Fixed dimensions specified in q:SliceKey and their values are defined in qb:Slice just by including the dimensions as properties and their specific values and their property values. A qb:Slice also lists containing qb:Observations using its property qb:observation.

Chapter 4. Architecture of Multidimensional Linked Data Applications

In this chapter we will discuss a general architecture for publishing and consuming multidimensional data as linked data on the web. As well we will identify components involved in the process of publishing and consuming multidimensional linked data and we will discuss the component's role in the process.

4.1. Overall Architecture for Mapping/Transforming Data

The following diagram shows the elements involved in converting a multidimensional source of data to RDF/QB graph:

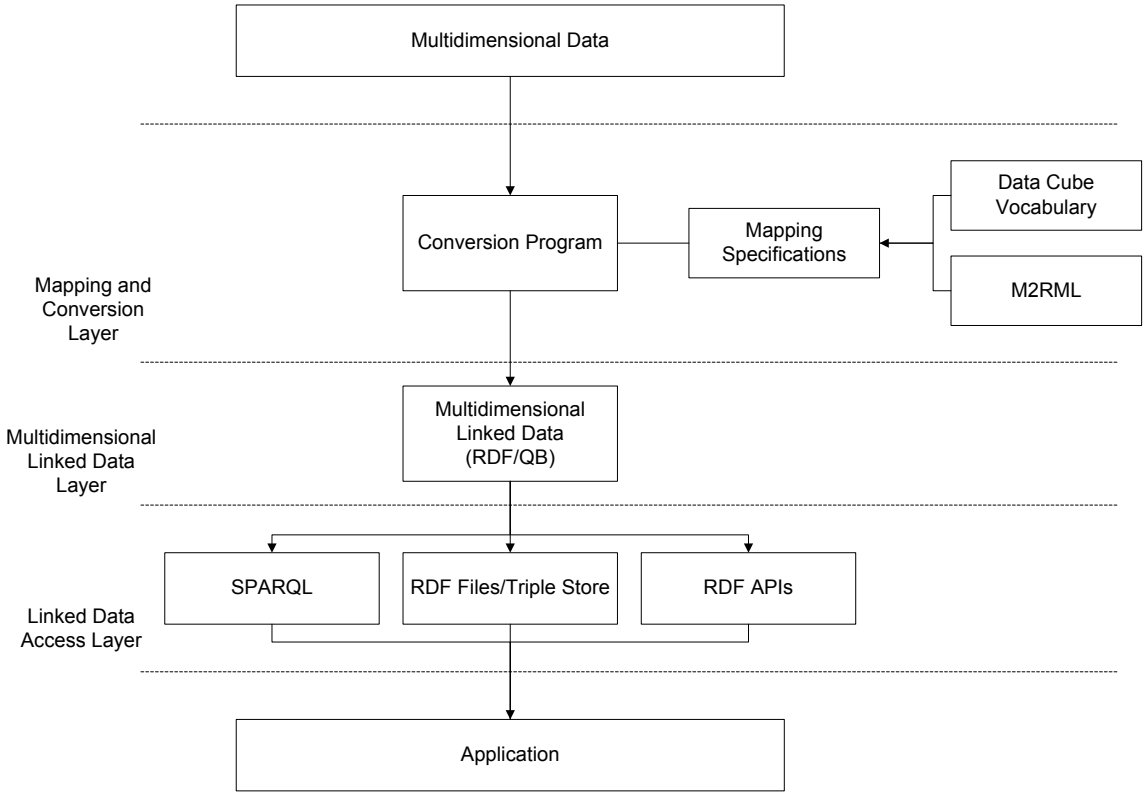

Figure 4.1 Overall Architecture for Accessing Multidimensional Linked Data

4.1.1. Source Layer

The source layer stores multidimensional data to be converted and can be an OLAP cube, MDX Queries (Slices, Sub-Cubes, etc.), Spreadsheet, Multidimensional Web Tables or other sources of multidimensional data.

Here we consider OLAP servers as our source of analytical data to be published on the web. OLAP is one of the most common approaches for processing data for analytical tasks.

OLAP cube is the source of all data and metadata. It includes the cube model which specifies dimensions, hierarchies, levels, members and measures as well as the values in the cells (measured values).

An OLAP cube can be accessed using query languages such as MDX, and XML or by using other APIs such as olap4j(an open java API for OLAP) which uses an XMLA diver to access OLAP servers. All these methods allow us to retrieve data and metadata

from the cube. Each vendor can have a specific driver/access method for its specific implementation of dimensional model. XMLA, XML for analysis, (Microsoft Corp., 2013) is the standard way to access OLAP cubes which allows applications to be vendor agnostic and can be used to interact with OLAP cubes from different sources/vendors. XMLA can be used to build generic drivers that can connect to different OLAP engines. For instance olap4j(Hyde, 2013)includes a generic XMLA driver that can connect to Mondrian, SAP BW and Microsoft SQL Server Analysis Services.

4.1.2. Mapping and Transformation Layer

The mapping/transformation layer which sometimes is referred to as ETL (Extraction, Transform, and Load) layer is where the data conversion happens and consists of the following elements:

- Target Vocabulary (Data Cube Vocabulary)
- Mapping Language/Vocabulary(M2RML)
- Mapping Specifications (written with M2RML and DCV)
- Conversion Script/Program

Target Vocabulary(Data Cube Vocabulary)

Data Cube Vocabulary defines a cube model which is designed to be general enough so that it can be used for publishing different formats of statistical data (survey data, OLAP Cubes, spreadsheet).

Data Cube Vocabulary is used to represent statistical and analytical data in graph/RDF form. It provides classes and properties for representing dimensions, measures, slices and etc. The resulting graph will be represented in this vocabulary which means the converted elements are instances of classes and properties of Data Cube Vocabulary.

Mapping Language/Vocabulary (M2RML)

In order to map OLAP cubes to RDF/QB graph, we will use our mapping language, M2RML, which defines how OLAP elements are translated to RDF/QB elements (classes, properties). There exists a similar mapping language (R2RML, a

W3C Recommendation) to map data in relational databases to RDF triples and to provide an RDF view over relational databases which we will use as a reference for our mapping language so that it can be more familiar and be adopted more easily by users.

Having an OLAP Cube and the mapping language, one can use them as inputs to a program and get an RDF/QB graph as the output. The graph in turn can be navigated, explored and linked to other datasets.

Conversion Graph (Mapping Specifications)

Conversion graph includes specifications for generating RDF/QB graph triples. It uses M2RML vocabulary to specify how source data should be converted to instances of DCV classes and properties. This graph includes mapping specifications for elements such as qb:DataSet, qb:DimensionProperty, qb:MeasureProperty, qb:Observations and etc in Data Cube Vocabulary. This graph will then be used in the conversion script/graph to generate resulting triples.

Conversion Script/Program

Conversion script/program is the application/program that generates the resulting triples forming the resulting graph. This script or program reads mapping specifications from conversion graph and based on that generates the triples that form an RDF/QB graph. This graph will be equivalent to the source OLAP cube. This graph can then be stored or exposed for further data exploration/analysis in semantic web/linked data.

4.1.3. Destination Layer

Destination layer has the generated graph as the final output of the mapping/translation procedure. Generated RDF triples will be the equivalent graph representation of OLAP Cube. As this graph is built using Data Cube Vocabulary, its nodes and properties will be instances of classes and properties in the Data Cube Vocabulary.

Once the OLAP cube is transformed into a RDF/QB graph, there would be several ways to interact with the data in the graph form. As mentioned in (Auer et. al, 2010) the data in graph form can be:

- Queried by the user using SPARQL (query access)
- Browsed using a browser or a crawler client (entity-level access)
- or can be dumped and stored in a RDF/Triple store (access-via-dump)

As can be seen the resulting graph from data conversion layer can then be stored in RDF/Triple stores or can be exposed so that it can be queried using SPARQL or be consumed or explored using RDF APIs.

Further applications can be built to consume the generated graph stored in RDF/Triples stores or text files and using RDF APIs, SPARQL to do various tasks such as analysis, visualization or exploration of data.

4.2. Overall Architecture for Publishing Multidimensional Data

Similar to figure 5.1 in (Heath & Bizer, 2011) which shows linked data publishing options and workflows for relational data, the figure below shows how multidimensional data can be published on the web.

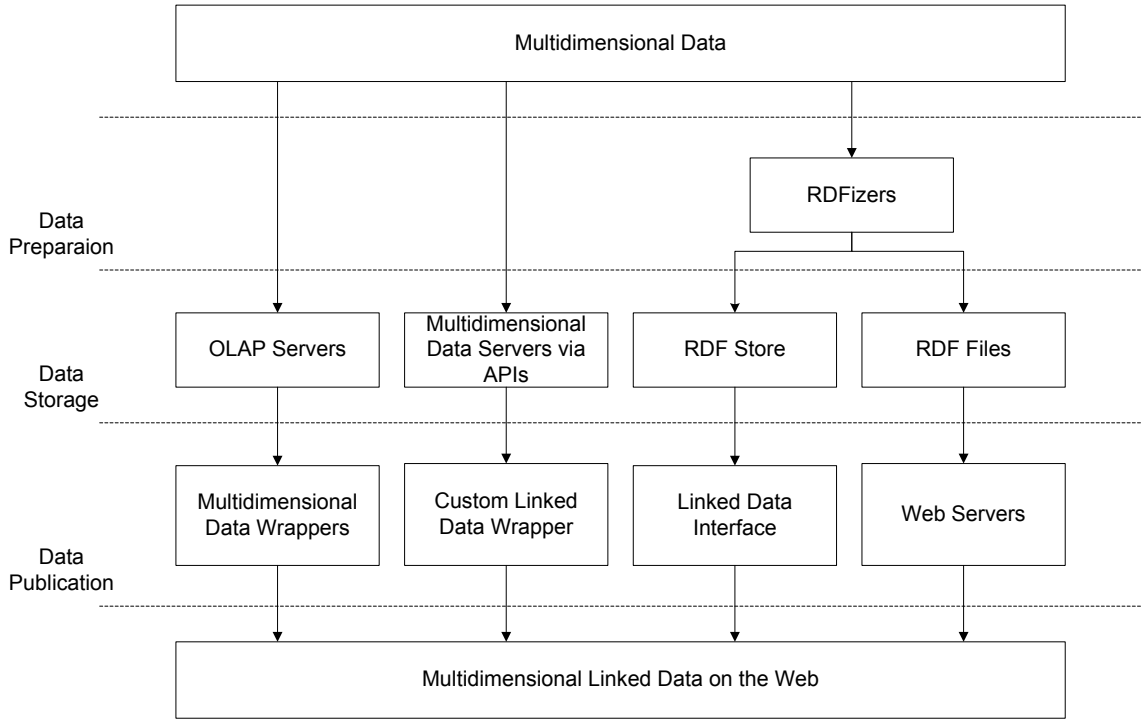

Figure 4.2. General Framework/Architecture for Publishing Multidimensional Data

As can be seen the source layer contains multidimensional data in various formats. Based on where the storage type of data we will have different approaches to publish multidimensional data.

If the data is stored in multidimensional data stores such as OLAP cubes/engines OLAP cube wrappers can be used to map multidimensional data to RDF/QB dataset and expose and publish the resulting dataset based on Linked data principles.

If multidimensional data is stored in a third party data store and can be accessed using third party APIs we might need a custom wrapper that can map and convert retrieved multidimensional data to RDF/QB datasets for exposing and publishing it on the web.

It is also possible to have multidimensional data as static spreadsheet files or multidimensional web pages on the web. In this case RDFizers will be used to map and convert the data to be stored in a RDF store or plain RDF files. If the resulting RDF/QB dataset is stored in a RDF store it will be exposed to the web using the RDF store or an

RDF end-point if the RDF store doesn't have one. If the resulting RDF/QB dataset is stored in plain RDF files it can be directly server on the web using web servers or can be loaded into a RDF store and then exposed to the web.

4.2.1. Additional Considerations for Publishing Linked Data

As mentioned in (Heath & Bizer, 2011) regardless of storage type of existing data other factors mentioned such as data volume and change frequency of data would affect the method/pattern for publishing linked data.

Data Volume

If the amount data we would like to publish is small we might consider storing the resulting RDF/QB graph in static RDF files and avoid complexities and costs of setting up required tools. In this case we might need to handle some data management tasks manually.

For larger RDF graphs we can store different parts/entities of dataset in different files and load or retrieve them upon request.

Change Frequency

Another factor that should be considered for publishing data is how frequent our data change in the source. If source data is static and rarely changes the best way might be to store RDF data in plain files and expose it to the web using web servers.

However if the data changes frequently in multidimensional source of data it would be better to work with tools that help with management of data such as RDF stores. In case of having OLAP systems or third party sources with APIs to access data it would be better to work with OLAP cube wrappers or custom wrappers to avoid interfering with source data.

4.3. Overall Architecture for Consuming Multidimensional Data

The figure below show the overall architecture for consuming published linked data. As can be seen we have three layers for

- The source layer that has multidimensional data
- The mapping and transformation layer
- The linked data layer that holds converted RDF/QB datasets
- The application layer that request multidimensional data

The application layer can request multidimensional data in two ways, request using an MDX query (slice, sub-cube, etc.) or request data using SPARQL query.

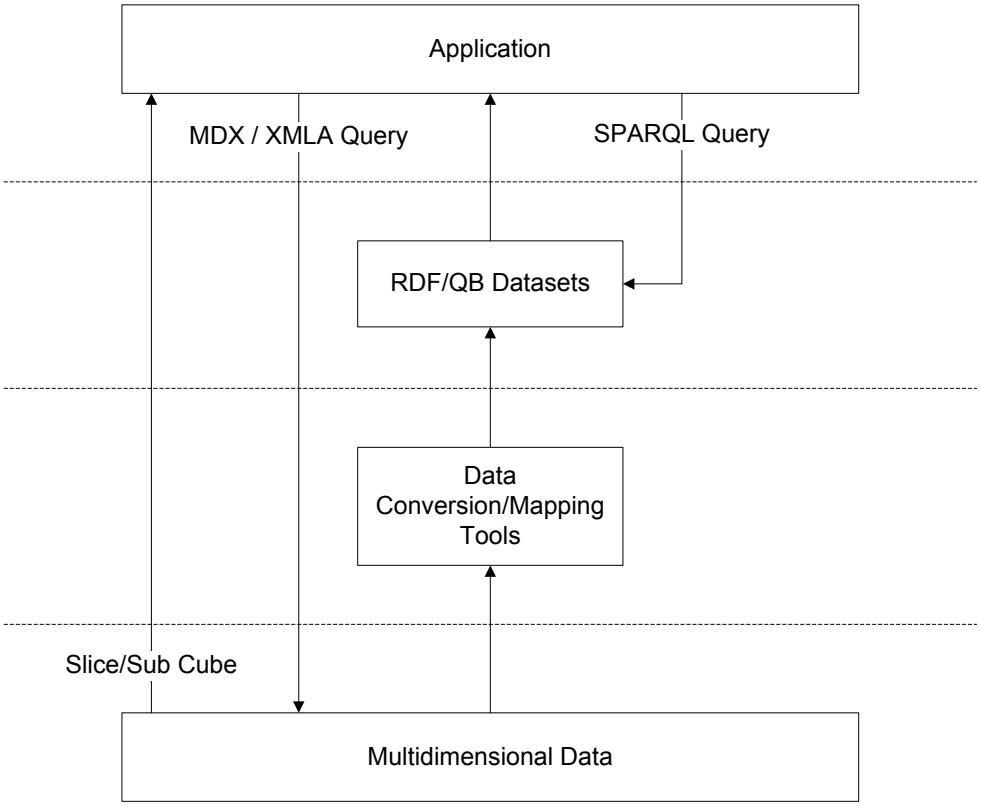

Figure 4.3. General Framework/Architecture for Consuming Multidimensional Linked Data

4.3.1. Requesting Data using MDX or XMLA Queries

If the application sends an MDX query, the query will be passed to the source layer which will execute the query. Result of the executed query is a slice, sub-cube or a smaller sub set of multidimensional data in the source. The result will be passed to the conversion layer which uses metadata passed along with results to map and transform the result of MDX query to an RDF/QB dataset. The resulting RDF/QB dataset will then be passed to the linked data layer and to the application accordingly.

4.3.2. Requesting Data using SPARQL Queries

The application can also query data using SPARQL in which case if the RDF/QB dataset is available at linked data layer the result will be passed to the application from this layer. However if the requested multidimensional linked data is not available in the linked data layer, the query should be translated to an MDX query and sent to the source to retrieve the result-set requested by the application. This result-set will then be transformed to RDF/QB dataset similar to the previous scenario where application sends an MDX query.

Translating SPARQL queries to MDX queries is out of scope of this thesis however there is a work on that by (Moreira & de Freitas Jorge, 2012).

Chapter 5. M2RML – Multidimensional to RDF Mapping Language

In this chapter we present M2RML, the Multidimensional to RDF Mapping Language, which facilitates building custom mappings for transforming multidimensional source data to target RDF dataset in Data Cube Vocabulary.

The mapping language is designed based on R2RML standard and therefore there are similarities and differences between the two which we will discuss in the following sections. The reason to design M2RML based on R2RML is that:

- We can ensure its elements and approach is similar to a standard in semantic web which may make it easier to be approved or accepted as a standard later by W3C
- Its similarity to R2RML (as a tool already used by users) allows users to understand and adopt it easier and faster

On the other hand it is designed such that:

- It enables expressing mappings from all data source types (OLAP Cubes, Spreadsheets, Survey data, etc.)
- It can be used for deriving transformation details programmatically/systematically
- It can be flexible, expandable and possible to be improved
- It can be used for developing tools to ease the mapping and transformation process of multidimensional data in future

In this chapter first we will compare relational model and multidimensional model and then we move on to design ofM2RML elements; at the end of chapter we compare M2RML with R2RML and we summarize similarities and differences between them.

5.1. Mapping/Transformation Overview

In order to better understand the mapping language we have to have an overview of the mapping process and see what elements are involved and how they are related to each other.

Similar to any other mapping language/scheme, in M2RML, we would have an input, an output and a mapping language or transformations process in the middle tier. In M2RML the source data is multidimensional data and can be from OLAP cubes, spreadsheets, surveys or any other source like multidimensional web tables that express statistics or reports or etc. The middle layer consists of mappings expressed in M2RML using it's classes and properties which we will discuss later; and finally the output layer consists of target RDF triples in Data Cube Vocabulary which will form a RDF/QB dataset/graph.

In comparison, in R2RML the input is structured data in relational database and the output is RDF triples that together will form an RDF graph/dataset in a target vocabulary. R2RML accepts source data from relational databases in form of a table, a view or a query. In the middle tier mappings are expressed in R2RML using its classes and properties and finally the output layer consists of RDF triples created in target vocabulary which can be any vocabulary. The resulting triples or output of the conversion process represents rows of data in the source.

So as can been seen a difference between R2RML and M2RML in higher level is that any vocabulary can be used as target vocabulary of output RDF triples whereas in M2RML, Data Cube Vocabulary is the target vocabulary and it is fixed as classes and properties are designed specifically for Data Cube Vocabulary. In fact M2RML can be separated to two parts: 1) A part which includes abstract and general classes which is based on and similar to R2RML. Most of the elements in this part are elements of R2RML generalized to be used for any kind of source data (and not just relational data); and 2) The other part is built upon the abstract classes in the first part and is created specifically to map multidimensional data as source and assumes Data Cube Vocabulary as the specific target vocabulary.

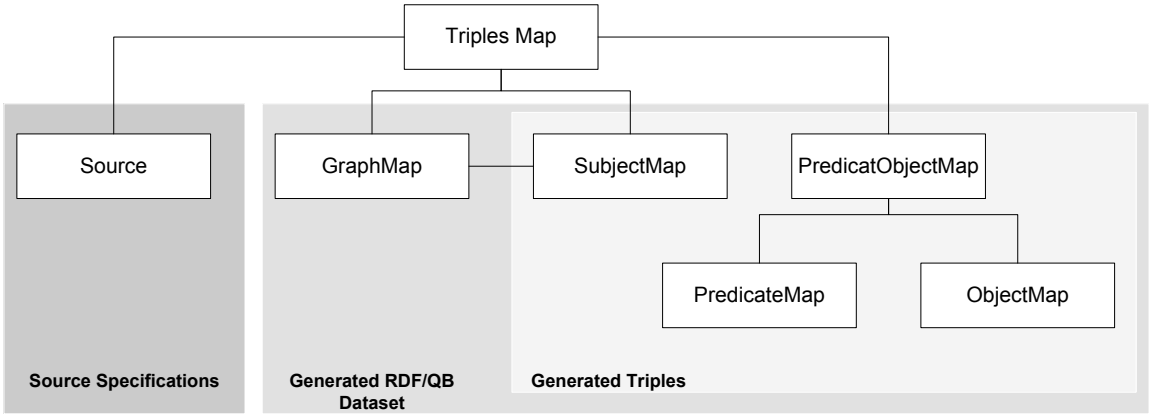

Figure 5.1. Overview of M2RML

5.2. Abstract Classes in M2RML

Abstract classes in M2RML are designed to provide a general mapping scheme that can be used for mapping any type of source data. Later we build upon these abstract classes to have more specific classes we need for mapping multidimensional data to RDF triples in DC vocabulary.

Abstract classes in M2RML are like abstract or super classes in Object Oriented programming. They provide a basis for building more specific classes with common characteristics. The abstract classes will include these common characteristics and will later be used to build more specific sub classes. Sub classes in RDF are similar to extended/derived/child classes in object oriented programming.

M2RML abstract classes are similar to some classes in R2RML which are generalized and has the dependency on relational data removed. Following components are defined general as abstract classes in M2RML:

5.2.1. Map Element

This element in M2RML defines our desired mapping as a whole and acts as a container for all other elements. Each map is targeted to generate a specific set of RDF triples and includes required specifications for generating our desired triples. In M2RML the map class is identified by m2r:TriplesMap and includes the following:

- Source data specifications
- Subject, predicate and object specifications

In RDF subject, predicate and object of a triple are all RDF terms which will further help us to define an abstract class for these elements in M2RML. In the following sections we introduce each of these classes included in a map.

```
m2r:TriplesMap

      rdf:type rdfs:Class, owl:Class;

      rdfs:label "Abstract class representing mapping of a
source to destination triples";

      rdfs:seeAlso rr:TriplesMap.
```

Basically each mapping is represented by a TriplesMap which includes specifications of the source, subject, predicate and object of the target triple. It should also be mentioned that each TriplesMap will have exactly one source specification and exactly one subject specification however it can have multiple predicate and objet specifications as we can have multiple property-value pairs for a specific subject in RDF.

5.2.2. Source Data Specifications

Source data specifications includes the following elements and helps us to identify source of data to be transformed into RDF triples:

- Source name
- Source type
- Source connection information

Source of data can include any type of data. It's designed to be abstract enough so that users can represent any source for any kind of data mapping. Source name is self descriptive. Source type helps us to identify type of data source such as relational data, multidimensional, etc.

The source class is called m2r:Sourceand is connected to an m2r:TriplesMap using m2r:source property.

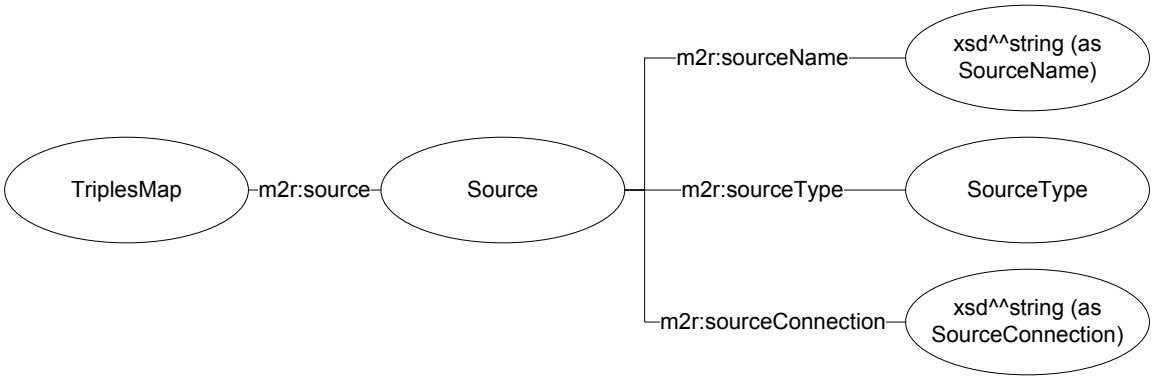

Figure 5.2. Source Specfications in M2RML

Source name is a string and is connected to m2r:Source using m2r:sourceName property.

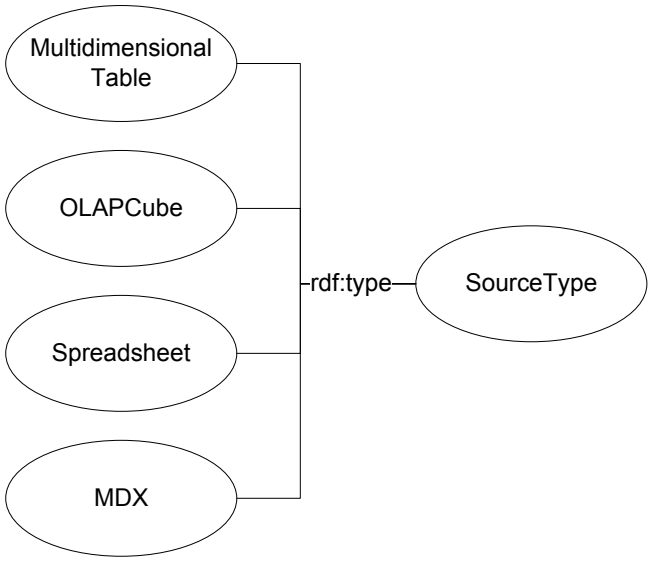

Figure 5.3. M2RML SourceTypes

The class representing source type is m2r:SourceType which is linked to m2r:TriplesMap using m2r:sourceType property. Instances of m2r:SourceType class are the following:

- m2r:MultidimensionalTable represents a general multidimensional source of data such as statistical and summary tables on web pages that include multidimensional data.
- m2r:OLAPCube as its name shows, represents an OLAP Cube
- m2r:Spreadsheet for multidimensional data in spreadsheet files

- m2r:MDX for an MDX query that returns multidimensional data as result; such as a slicer axis or a sub-cube

We have also defined another property for specifying connection information to the source. The property is called m2r:sourceConnection and its value is a string which can be used in an application to connect to original source of data.

The general scheme for representing source specifications in a TriplesMap is as below:

```
TriplesMap

      m2r:source<source name as string>;

      m2r:source type <source type as an instance of
m2r:SourceType class>;

      m2r:sourceConnection<connection string to source data
as plain string>;

      rdfs:comment <comment as string>.
```

As an example we can have a TriplesMap with an OLAP cube as its source expressed in the following way:

```
ex:TriplesMap1

      rdf:type m2r:TriplesMap;

      m2r:source "AdventureWorks_DW2012 Cube"

      m2r:sourceType m2r:OLAPCube

      m2r:sourceConnection "OLAP Connection String to be
used for connecting to the cube"

      rdfs:comment "An example TriplesMap which has an OLAP
Cube as its source.".
```

5.2.3. Subject Specifications

The next required element of a map is subject specifications for properly generating subject of the desired triple. In M2RML the class m2r:SubjectMap includes

mapping specifications for subject of the target output triple(s). Following figure shows m2r:SubjectMap class and its properties:

Eachm2r:SubjectMap has at least two specifications for subject of the triple

- Type of the subject
- IRI template of the subject/resource

Type of a resource or subject in RDF is specified using rdf:type property. For instance in the triple below we have defined ex:CarA is of type ex:Car.

```
ex:CarA rdf:type ex:Car.
```

In M2RML, m2r:SubjectMap specifies type of a subject/resource by using m2r:class property of the m2r:SubjectMap. The value of this property must be an IRI which defines type of the subject.

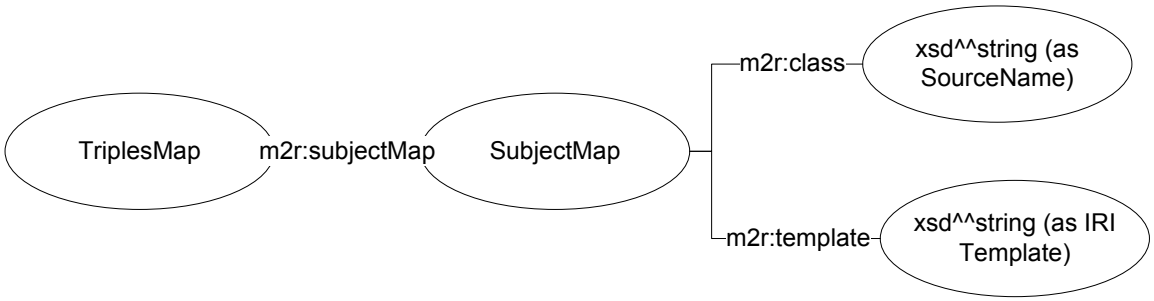

Figure 5.4. Subject Specifications in M2RML

Resources in RDF are identified using unique IRIs, therefore a SubjectMap will have a template to generate a unique IRI for subject term of the target triple. The example below shows a mapping in M2RML which generates a triple for a dimension having its type and the template of its IRI.

```
[] m2r:template
"http://data.example.com/dimension/{DimensionName}";

     m2r:class qb:DimensionProperty.
```

This example will use an OLAP Cube dimension to generate a triple defining a qb:DimensionProperty and it will use the dimension name to generate the IRIs. The resulting triple will look like:

```
<http://data.example.com/dimensions/ProductName> rdf:type
qb:DimensionProperty.
```

5.2.4. Predicate-Object Specifications

Predicate and objects (or Properties and values) are two other types of RDF terms in addition to subject of a triple. Each subject or resource in RDF can have multiple properties and values. Each property/predicate of a subject/resource should have a value/object. This is why these two terms are defined together as pairs of property-value or predicate-object.

Specifications of mapping to predicate-object pairs in M2RML are expressed using m2r:PredicateObjectMap class which in turn will include separate specifications for predicate and object of the target triple. A general form of anm2r:PredicateObjectMap is:

```
m2r:PredicateObjectMap

      <PredicateMap>

            <predicate specifications>

      <ObjectMap>

            <object specifications>
```

The following figure shows a PredicateObjectMap and properties that links it to PredicateMaps or ObjectMaps.

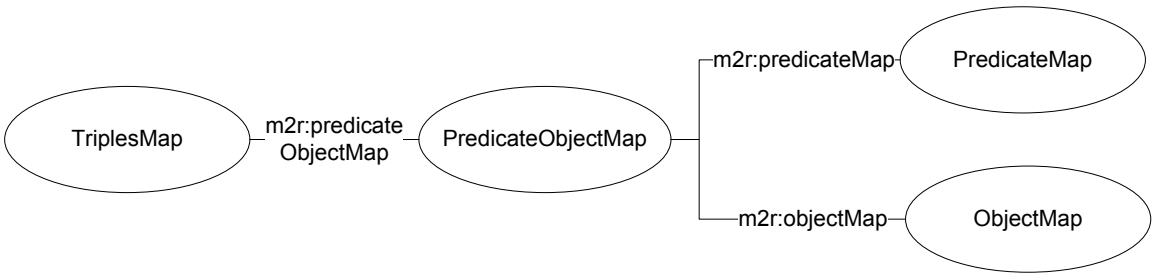

Figure 5.5. Predicate and Object Specifications in M2RML

As shown, m2r:PredicateObjectMap is connected to two elements PredicateMap and ObjectMap which contain specifications of predicate and object of the target triple. Predicate and object of a triple must be exactly one of the following:

- Constant-valued term map
- Element-valued term map
- Template-valued term map

PredicateObjectMap is connected to these two elements with the following properties. For predicate specification:

- If PredicateMap is a constant-valued TermMap, m2r:predicate is used as the property. The value for this property will be a constant IRI that will be the same in all generated triples.
- If PredicateMap is a column-valued or template-valued TermMap, m2r:predicateMap property will be used. The value of this property should be a PredicateMap that dynamically generates predicate of the triples.

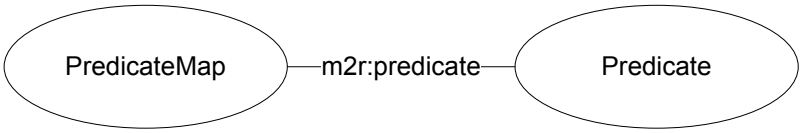

Figure 5.6. Predicate Map in M2RML

For object specification:

- If ObjectMap is a constant-valued TermMap, m2r:object is used as the property. The value for this property will be a constant IRI or a constant literal that will be the same in all generated triples.
- If ObjectMap is a column-valued or template-valued TermMap, m2r:objectMap property will be used. The value of this property should be an ObjectMap that dynamically generates the object of the triples.

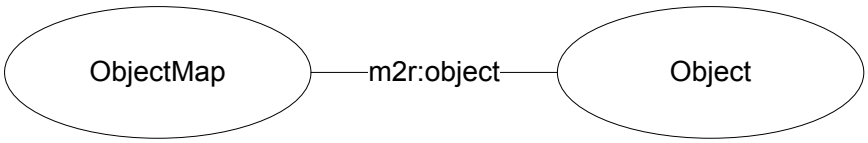

Figure 5.7. ObjectMap in M2RML

Therefore the general RDF form a predicate might look like:

```
PredicateObjectMap

      m2r:predicateMap<a PredicateMap>

      m2r:objectMap<an ObjectMap>
```

Or if constant-valued TermMaps are used it would look like:

```
PredicateObjectMap

      m2r:predicate<a constant predicate>

      m2r:object<a constant object>
```

The example below shows how a measure in an OLAP cube will be mapped to a qb:MeasureProperty in RDF Data Cube Vocabulary.

```
[] m2r:predicateMap [m2r:constant rdf:type];

      m2r:objectMap [m2r:constant qb:MeasureProperty].
```

The same map can be represented using constant-valued TermMaps in the following form:

```
[] m2r:predicate rdf:type;

      m2r:object qb:MeasureProperty.
```

The result of these two mapping will be the same and as below:

```
?x rdf:type qb:MeasureProperty.
```

As predicate and object of a triple are instance of an RDF term they can have related properties of an RDF term such as language tags, data types, etc. In the example below we map an OLAP Cube dimension attribute to an RDF triple and assign a language tag to its label. In the next section we will provide more information on m2r:TermMap and its properties.

```
[] m2r:predicate rdfs:label;

      m2r:objectMap [

            m2r:element "This is the label for dimension
attribute {AttributeName} in English language";
```

```
            m2r:language "en-us"].
```

The resulting generated triple will be:

```
?x rdfs:label "This is the label for dimension attribute
Product Category in English language"@en-us.
```

5.2.5. RDF Term (Subject, Predicate, Object) Specifications

As we mentioned before subject, predicate and object of a triple are all instances of RDF term therefore we can have a generalized class for these elements that represents their general and common specifications. This class in M2RML is identified by m2r:TermMap. A term map is used to define general properties of subject, predicate or object of a triple such as:

- Type of a term (IRI, blank node or literal)
- Data type of the term (string, integer, float, etc.)
- Language tag for label/description of the term
- And etc.

The figure below shows m2r:TermMap class and its properties. Each property is explained following this figure.

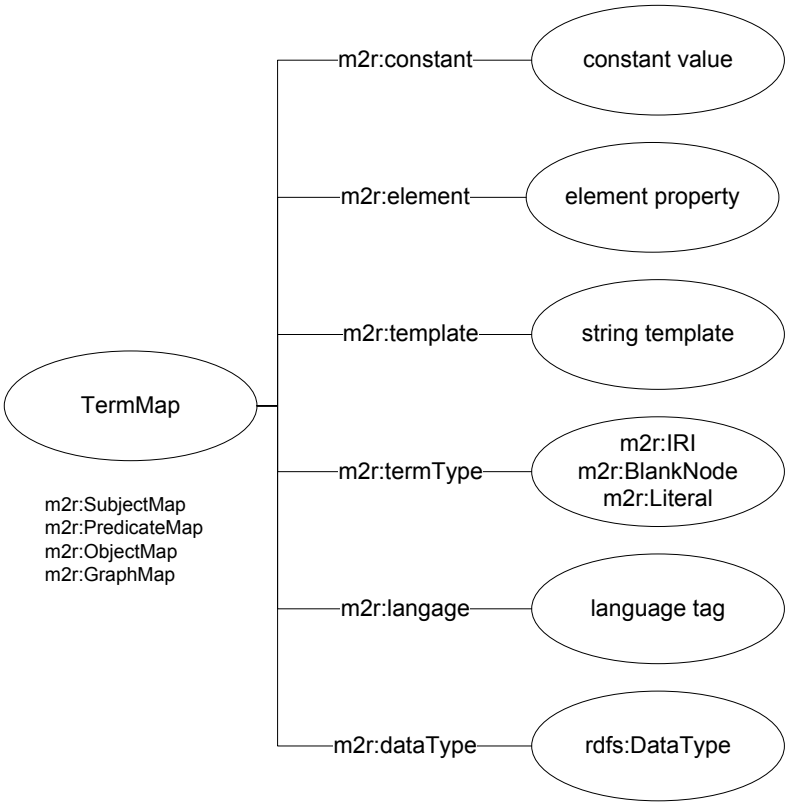

Figure 5.8. TermMap and its Properties

m2r:constant Property

m2r:constant is used when a constant valued term is generated for a resource regardless of the logical table row. According to (Das, Sundara, & Cyganiak, 2012) if a constant valued term map is subject, predicate or graph the constant value can be an IRI however if it's an object the constant value can be an IRI or a literal. There are shortcuts created to use with constant-valued term types as:

- m2r:subject for constant valued subject term
- m2r:predicate for constant valued predicate term
- m2r:object for constant valued object term
- m2r:graph for constant valued graph term

m2r:element Property

m2r:elementis used when the term is generated using elements such as dimensions or measures of multidimensional data. For instance if we want to generated

a qb:MeasureProperty we can use name of the measure in OLAP cube for this element. Using this property indicated that name of generated qb:MeasureProperty is derived or extracted from OLAP cube measure element.

m2r:template Property

m2r:template generates a term as an IRI using a template to define the IRI. The template string can include column names of a logical table row enclosed in curly braces. Values of these columns for a logical table row will be replaced in the template to build the IRI term.

m2r:termType Property

m2r:termType defines type of the RDF term to be generated (IRI, blank node or literal)

m2r:language Property

m2r:language allows adding language tags to literal RDF terms. The value of this property should be a valid language tag and the generated term having this property will be a language-tagged plain literal

m2r:dataType Property

m2r:dataType enables us to have typed literal terms however the term should be of literal type and should not be language-tagged literal. Value of this property is an IRI defining type of the literal

5.2.6. Assigning Triples to Named Graphs

RDF graphs are simply a set of RDF triples. Graph IRI that contain the RDF triple can be simply added to the RDF triple to form a quadruple in the following format:

```
<graph name><subject><predicate><object>
```

Same as subject, predicate and object of a triple, graph is also an RDF term and in R2RML and M2RML can be generated using a TermMap and connected to a TriplesMap using m2r:grpahMap property. If the TermMap is a constant-valued

TermMap we can use the constant shortcut property m2r:graph instead. Below is an example usage of GraphMaps to generate containing named graph of the term.

Figure 5.9. SubjectMap and GraphMap

5.2.7. Summary

In M2RML a TriplesMap is an abstract class defining a mapping from source data to target triples and includes the following elements:

- Source as abstract class for expressing source of data in the mapping
- SubjectMap as abstract class for expressing mapping specifications of subject of the target triple
- PredicateObjectMap as abstract class for expressing mapping specifications of predicate and object of the triple which in turn consists of:
    - PredicateMap
    - ObjectMap

These classes are connected to an m2r:TriplesMap using the following properties:

- A Source is connected to a TriplesMap using its m2r:source property
- A SubjectMap is connected to a TriplesMap using its m2r:subjectMap property
- PredicateObjectMaps are connected using m2r:predicateObjectMap of a TriplesMap
- PredicateMap is connected to a PredicateObjectMap using m2r:predicateObjectMap property
- ObjectMap is connected to a PredicateObjectMap using m2r:objectMap property

If we would like to represent a TriplesMap in RDF we would have:

```
ex:TriplesMap1

      rdf:type m2r:TriplesMap;
```

```
      m2r:source<source>;
      m2r:subjectMap<subjectMap>;
      m2r:predicateObjectMap<predicateObjectMap>;
            m2r:predicateMap<predicateMap>;
            m2r:objectMap<objectMap>.
```

Now that we have a general image of the mapping we discuss each element/part more in detail and see how they are used in designing M2RML for mapping multidimensional to RDF mapping language in the following sections.

5.3. DCV as the Target Vocabulary in M2RML

In this section we will present the specific classes and properties of M2RML designed for using Data Cube Vocabulary as the target vocabulary for mapping multidimensional data.

5.3.1. Design Considerations for DCV-Specific Elements

The general and abstract classes we discussed in the previous section are building blocks and basis of the classes and properties introduced in this section. Till this point in the chapter, the mapping language, M2RML, can be used with any source of data and data can be generated in any target vocabulary however from this point onward we will assume Data Cube Vocabulary as the standard for publishing multidimensional data and we will fix it as the target vocabulary.

Fixing Data Cube Vocabulary as our target RDF vocabulary allows us to introduce specific classes and properties that are natively connected to DCV elements and makes it easier for user to express mappings. These specific classes and properties will hold the information about elements of DCV as the target vocabulary and hold necessary mapping related information in them rather than leaving it to the user to specify target elements.

If we consider DCV as the target vocabulary and write some mappings in M2RML we can see which information will be always static to include in class and property definitions. For instance let's look at the examples below. We assume AdventureWorks_DW2012 OLAP cube is our source of multidimensional data and which we would like to map to an RDF/QB. We can have the following map:

```
ex:DataSetMap rdf:type m2r:TriplesMap;

      m2r:source "AdventureWorks_DW2012";

      m2r:sourceType m2r:OLAPCube;

      m2r:sourceConnection "An arbitrary OLAP cube
connection";

      rdfs:comment "Mapping to generate triples for the
qb:Dataset element".

      m2r:subjectMap [

            m2r:template "http://example.com/{CubeName}";

            m2r:class qb:DataSet];

      m2r:PredicateObjectMap [

            m2r:predicate qb:structure

            m2r:object
http://example.om/AdventureWorks_DW2012_DSD].
```

Here we have defined a TriplesMap to generate the qb:DataSet element of our desired RDF/QB dataset; it consists of source information as well as mapping specifications for subject, predicate and object of the triple. The result of this map would be:

```
http://example.com/AdventureWorks_DW2012

      rdf:type qb:DataSet;

      qb:structure ://example.om/AdventureWorks_DW2012_DSD.
```

This map has generated these two triples, one as the result of SubjectMap and the other as the result of PredicateObjectMap. Now let's assume we have another cube

for which we would like to create another map and generate related triples. If we compare these two maps we will notice that some elements are fixed in the maps:

- SubjectMap will always have the following:
    - m2r:class qb:DataSet which makes the subject an instance of qb:DataSet class
- PredicateObjectMap will always have:
    - m2r:predicate qb:structure as the PredicateMap
    - An object map representing or generating a qb:DataStructureDefinition resource

In fact every map representing or generating a qb:DataSet object will have the above elements and this makes it possible to create a type of map specific to qb:DataSet in Data Cube Vocabulary and allows us to embed the above characteristics in that.

The design scheme we would have is as follows. For each required and necessary element of Data Cube Vocabulary:

- Create a map type as rdf:subClassOf m2r:TriplesMap
- Create a SubjectMap representing the specific element type and embed the class information in it
- For each required property of the specific element:
    - Create a PredicateObjectMap representing the required predicate/property and embed the predicate
    - Embed the type of object/value of that property using m2r:objectClass property.

In order to demonstrate this we will represent the required classes and properties specific to qb:DataSet and we will re-write the map with our new classes and properties. The classes and properties specific to a qb:DataSet is as following.

TripleMaps

As we mentioned for each element of Data Cube Vocabulary we will have a specific map defined as sub class of the general/abstract TriplesMap class.

```
#The map class specific to qb:DataSet element
```

```
m2r:DataSetMap

      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing mapping of an OLAP Cube
to a qb:DataSet".
```

SubjectMaps

For each element of Data Cube Vocabulary we will have a specific SubjectMap which has the type of the element embedded in it and a specific property linking the map and the SubjectMap.

```
#The SubjectMap class and property specific to qb:DataSet
element

m2r:SubjectMapDataSet

      m2r:class qb:DataSet.



m2r:subjectMapDataSet

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:DataSetMap;

      rdfs:range m2r:SubjectMapDataSet.
```

As shown we have m2r:SubjectMapDataSet to be used as SubjectMap for mapping qb:DataSet elements and as we mentioned before it has the subject class information included so that user doesn't need to specify that in a mapping.

We also have a property m2r:subjectMapDataSet a sub property of m2r:subjectMap general/abstract property which has the range and domain properties to only link m2r:SubjectMapDataset ad m2r:DataSetMap instances.

PredicateObjectMaps

For each element of Data Cube Vocabulary we will have a designated PredicateObjectMap class and predicateObjectMap property.

```
#The PredicateObjectMap class and property specific to
qb:DataSet element

m2r:PropertyMapDSD

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate qb:structure;

      m2r:objectClass qb:DataStrcutureDefinition.


m2r:propertyMapDSD

      rdfs:subPropertyOf m2r:predicateObjectMap;

      rdfs:domain m2r:DataSetMap;

      rdfs:range m2r:PropertyMapDSD.
```

The designated class m2r:PredicateObjectMapDSD is a sub class of m2r:PredicateObjectMap class and includes the required target predicate of the DCV element. In addition it also includes the type of the target value in the target property-values pair using m2r:objectClass property. Here m2r:PredicateObjectMapDSD is specific to structure of the qb:DataSet and has the fixed predicate qb:structure which accepts an instance of qb:DataStrcutureDefinition as its value.

The property m2r:predicatObjectMapDSD connects a m2r:PredicateObjectMapDSD to a m2r:DataSetMap and same as a m2r:subjectMapDSD has rdfs:range and rdfs:domain properties restricting the domain to which it is applied and the type of values it accepts.

5.3.2. DCV Specific Classes and Properties

In the previous section we described the idea based on which we designed the specific classes. Here we take a closer look at specific map, subject map, property-object map and properties for main elements of Data Cube Vocabulary which enable us to map a multidimensional cube/slice to an RDF/QB graph.

DataSet

Part of specific classes and properties for qb:DataSet class. Here we present them again along with other classes and properties for the sake of completeness.

The figure below shows DCV-specific elements of M2RML for qb:DataSet class. The specific map type for a qb:DataSet is m2r:DataSetMap.

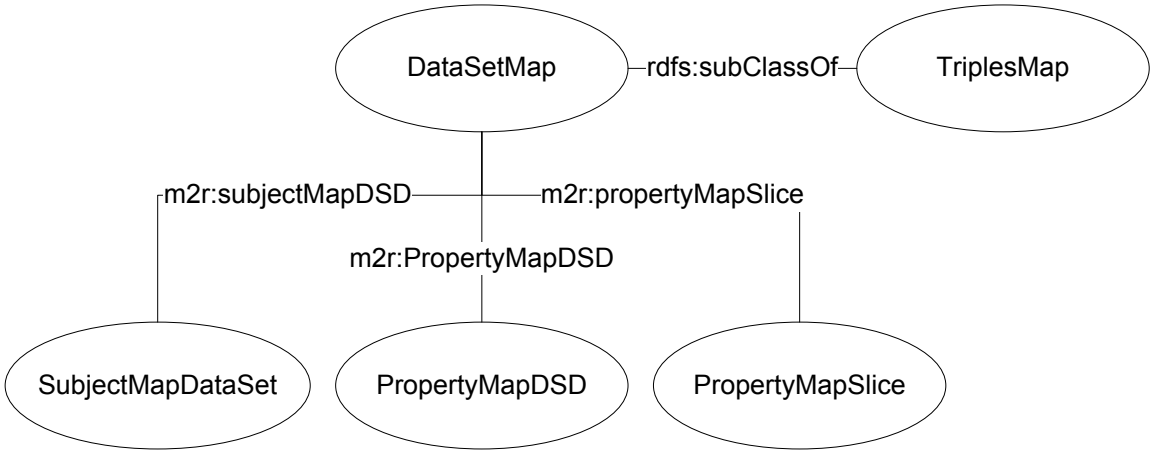

Figure 5.10. DataSetMap and its Properties

```
m2r:DataSetMap

        rdfs:subClassOf m2r:TriplesMap;

        rdfs:label "Class representing mapping of an OLAP Cube to a
qb:DataSet".
```

The specific subject map class is m2r:subjectMap which embeds qb:DataSet as subject type.

```
m2r:SubjectMapDataSet

        m2r:class qb:DataSet.
```

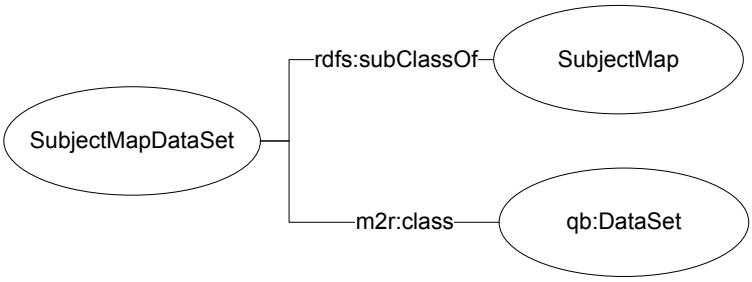

Figure 5.11 SubjectMapDataSet in M2RML

The m2r;SubjectMap class is connected to an m2r:DataSetMap class using m2r:subjectMapDataSet property.

```
m2r:subjectMapDataSet
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:DataSetMap;
        rdfs:range m2r:SubjectMapDataSet.
```

As a qb:DataSet is linked to qb:DataStructureDefinition (required) and qb:Slice (if required), two PredicateObjectMaps, m2r:PropertyMapDSD and m2r:PropertyMapSlice are created for these elements respectively.

The properties to link these two types of PredicateObjectMaps are m2r:propertyMapDSD and m2r:propertyMapSlice.

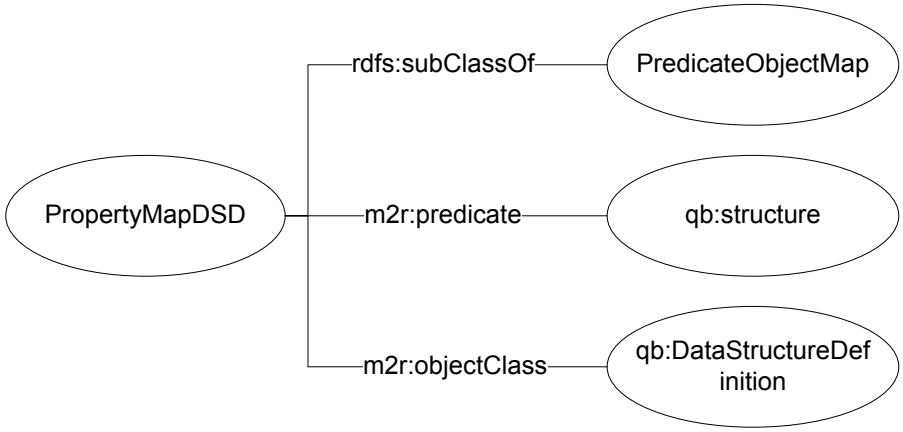

Figure 5.12 PropertyMapDSD for DataSetMap in M2RML

```
m2r:PropertyMapDSD
```

```
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate qb:structure;
        m2r:objectClass qb:DataStrcutureDefinition.

m2r:propertyMapDSD
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:DataSetMap;
        rdfs:range m2r:PropertyMapDSD.
```

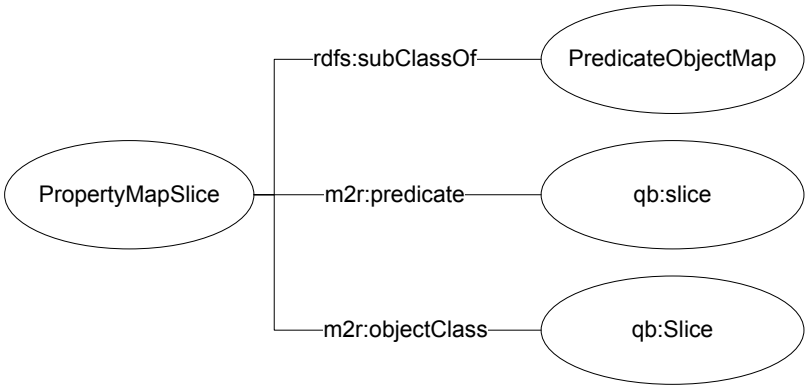

Figure 5.13 PropertyMapSlice for DataSetMap in M2RML

```
m2r:PropertyMapSlice
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate qb:slice;
        m2r:objectClass qb:Slice.

m2r:propertyMapSlice
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:DataSetMap;
        rdfs:range m2r:PropertyMapSlice.
```

Data Structure Definition

The specific map type for qb:DataStructureDefinition class is m2r:DataStructureDefinitionMap.

```
m2r:DataStructureDefinitionMap
        rdfs:subClassOf m2r:TriplesMap;
        rdfs:label "Class representing mapping of a qb:DataStructureDefinition".
```

The subject map for qb:DataStructureDefinition is m2r:SubjectMapDSD which embeds the class type of qb:DataStructureDefinition for subject of the triples it generates. It is connected to an m2r:DataStructureDefinitionMap using m2r:subjectMapDSD.

```
m2r:SubjectMapDSD
        m2r:class qb:DataStructureDefinition.


m2r:subjectMapDSD
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:DataStructureDefinitionMap;
        rdfs:range m2r:SubjectMapDSD.
```

Properties of a qb:m2r:DataStructureDefinition are qb:ComponentSpecification and qb:SliceKey represented by m2r:PropertyMapCompSpec and m2r:PropertyMapSliceKey. These Predicate ObjectMaps are linked to an m2r:DataStructureDefinitionMap using m2r:propertyMapCompSpec and m2r:propertyMapSliceKey respectively.

```
m2r:PropertyMapCompSpec
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate qb:component;
        m2r:objectClass qb:ComponentSpecification.
```

```
m2r:propertyMapCompSpec
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:DataStructureDefinitionMap;
        rdfs:range m2r:PropertyMapCompSpec.
```

```
m2r:PropertyMapSliceKey
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate qb:sliceKey;
        m2r:objectClass qb:SliceKey.

m2r:propertyMapSliceKey
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:DataStructureDefinitionMap;
        rdfs:range m2r:PropertyMapSliceKey.
```

Component Specification

The specific map for a qb:ComponentSpecification is m2r:ComponentSpecificationMap.

```
m2r:ComponentSpecificationMap
        rdfs:subClassOf m2r:TriplesMap;
        rdfs:label "Class representing mapping of a qb:ComponentSpecification".
```

Specific SubjectMap is m2r:SubjectMapCompSpec including class type of qb:ComponentSpecification for subject of triples it generates.

```
m2r:SubjectMapCompSpec
        m2r:class qb:ComponentSpecification.
```

```
m2r:subjectMapCompSpec
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:ComponentSpecificationMap;
        rdfs:range m2r:SubjectMapCompSpec.
```

A qb:ComponentSpecification has properties of type qb:componentProperty which are linked using qb:componentProperty property.

```
m2r:PropertyMapCompProperty
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate qb:componentProperty;
        m2r:objectClass qb:ComponentProperty.

m2r:propertyMapCompProperty
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:ComponentSpecificationMap;
        rdfs:range m2r:PropertyMapCompProperty.
```

Instances of qb:ComponentProperty are qb:DimensionProperty, qb:MeasureProperty and qb:AttributeProperty with respective properties qb:dimension, qb:measure and qb:attribute.

```
m2r:PropertyMapDimProperty
        rdfs:subClassOf m2r:PropertyMapCompProperty;
        m2r:predicate qb:dimension;
        m2r:objectClass qb:DimensionProperty.

m2r:propertyMapDimProperty
```

```
        rdfs:subPropertyof m2r:propertyMapCompProperty;
        rdfs:range m2r:PropertyMapDimProperty.
```

```
m2r:PropertyMapMeasureProperty
        rdfs:subClassOf m2r:PropertyMapCompProperty;
        m2r:predicate qb:measure;
        m2r:objectClass qb:MeasureProperty.

m2r:propertyMapMeasureProperty
        rdfs:subPropertyof m2r:propertyMapCompProperty;
        rdfs:range m2r:PropertyMapMeasureProperty.
```

```
m2r:PropertyMapAttributeProperty
        rdfs:subClassOf m2r:PropertyMapCompProperty;
        m2r:predicate qb:attribute;
        m2r:objectClass qb:AttributeProperty.

m2r:propertyMapAttributeProperty
        rdfs:subPropertyof m2r:propertyMapCompProperty;
        rdfs:range m2r:PropertyMapAttributeProperty.
```

Dimension Property

Specific map type for qb:DimensionProperty is m2r:DimensionPropertyMap.

```
m2r:DimensionMap
        rdfs:subClassOf m2r:TriplesMap;
```

```
        rdfs:label "Class representing mapping of a qb:DimensionProperty".
```

SubjectClass is m2r:SubjectMapDimension linked to m2r:DimensionMap using m2r:subjectMapDimension.

```
m2r:SubjectMapDimension
        m2r:class qb:DimensionProperty.

m2r:subjectMapDimension
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:DimensionPropertyMap;
        rdfs:range m2r:SubjectMapDimProperty.
```

A qb:DimensionProperty can have two properties for list of its members (code list) and class type of its members represented by m2r:PropertyMapCodeList and m2r:PropertyMapRangeClass. These are linked to m2r:DimensionMap using m2r:propertyMapCodeList and m2r:propertyMapRangeClass.

```
m2r:PropertyMapCodeList
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate qb:codeList;
        m2r:objectClass qb:ConceptScheme.

m2r:propertyMapCodeList
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:DimensionPropertyMap;
        rdfs:range m2r:PropertyMapCodeList.
```

```
m2r:PropertyMapRangeClass
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate rdfs:range;
        m2r:objectClass rdfs:Class.

m2r:propertyMapRangeClass
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:DimensionPropertyMap;
        rdfs:range m2r:PropertyMapRangeClass.
```

Measure Property

Here we have m2r:MeasureMap specific to qb:MeasureProperty.

```
m2r:MeasureMap
        rdfs:subClassOf m2r:Map;
        rdfs:label "Class representing mapping of a qb:MeasureProperty".
```

The SubjectMap and the property for linking it to m2r:MeasureMap.

```
m2r:SubjectMapMeasure
        m2r:class qb:MeasureProperty.

m2r:subjectMapMeasure
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:MeasureMap;
        rdfs:range m2r:SubjectMapMeasure.
```

Concept Scheme

Specific map for skos:ConceptSchemes used as possible list of values for qb:DimensionsProperty.

```
m2r:ConceptSchemeMap
        rdfs:subClassOf m2r:Map;
        rdfs:label "Class representing mapping of a skos:ConceptScheme".
```

The SubjectMap and the property to link it.

```
m2r:SubjectMapConceptScheme
        m2r:class skos:ConceptScheme.

m2r:subjectMapConceptScheme
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:ConceptSchemeMap;
        rdfs:range m2r:SubjectMapConceptScheme.
```

A skos:ConceptScheme mentions its top level members using skos:hasTopConcept property.

```
m2r:PropertyMapTopConcept
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate skos:hasTopConcept;
        m2r:objectClass skos:Concept.

m2r:propertyMapTopConcept
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:ConceptSchemeMap;
        rdfs:range m2r:PropertyMapTopConcept.
```

Range Class/Concept

Range classes or concepts are types of members of a dimension and is a property of qb:DimensionProperty. It has a specific map in M2RML m2r:ConceptMap.

```
m2r:ConceptMap
        rdfs:subClassOf m2r:TriplesMap;
        rdfs:label "Class representing mapping of a skos:Concept".
```

The SubjectMap and the property to link it.

```
m2r:SubjectMapConcept
        m2r:class skos:Concept.

m2r:subjectMapConcept
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:ConceptMap;
        rdfs:range m2r:SubjectMapConcept.
```

A concept can act like a member of a dimension or top concept of a skos:ConceptScheme using skos:inScheme property.

```
m2r:PropertyMapInScheme
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate skos:inScheme;
        m2r:objectClass skos:ConceptScheme.

m2r:propertyMapInScheme
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:ConceptMap;
        rdfs:range m2r:PropertyMapInScheme.
```

A skos:Concept can also be linked to other concepts in higher levels of a hierarchical structure using skos:broader property for which we have defined m2r:PropertyMapBroader class and m2r:propertyMapBroader property.

```
m2r:PropertyMapBroader
        rdfs:subClassOf m2r:PredicateObjectMap;
        m2r:predicate skos:broader;
        m2r:objectClass skos:Concept.

m2r:propertyMapBroader
        rdfs:subPropertyof m2r:predicateObjectMap;
        rdfs:domain m2r:ConceptMap;
        rdfs:range m2r:PropertyMapBroader.
```

Slice and Slice Key

A qb:SliceKey is linked to a qb:ComponentSpecification and has m2r:SliceKeyMap specific map type.

```
m2r:SliceKeyMap
        rdfs:subClassOf m2r:TriplesMap;
        rdfs:label "Class representing mapping of a qb:SliceKey". #needs ComSpec property which is already available.
```

The SubjectMap and the property to link it.

```
m2r:SubjectMapSliceKey
        m2r:class qb:SliceKey.

m2r:subjectMapSliceKey
        rdfs:subPropertyOf m2r:subjectMap;
        rdfs:domain m2r:SliceKeyMap;
        rdfs:range m2r:SubjectMapSliceKey.
```

5.3.3. Concluding M2RML and DCV Specific Elements

Now let's take a look at the result of having DCV specific elements in the vocabulary and see what benefits this mapping language will provide.

If we want to re-express the mapping for qb:DataSet element of our target RDF/QB dataset in the example we had in beginning of the section we would write it as below which is much simpler and less complicated than the mapping we had before:

```
ex:DataSetMap rdf:type m2r:DataSetMap;

      m2r:source "AdventureWorks_DW2012";

      m2r:sourceType m2r:OLAPCube;

      m2r:sourceConnection "An arbitrary OLAP cube
connection";

      rdfs:comment "Mapping to generate triples for the
qb:Dataset element".

      m2r:subjectMapDataSet

            m2r:template "http://example.com/{CubeName}";

      m2r:predicateObjectMapDSD [

            m2r:object
http://example.om/AdventureWorks_DW2012_DSD].
```

For all other commonly used and main elements of Data Cube Vocabulary such as qb:DataStructureDefinition, qb:Slice, qb:DimensionProperty, qb:MeasureProperty, etc. we will have specific classes and properties similar to what we showed in this section.

Having Data Cube Vocabulary fixed as our target vocabulary enables us to create specific classes and properties for mapping elements of Data Cube Vocabulary. These specific classes replace the complexity of the maps and make them more readable and maintainable by encapsulating repeating information in definition of classes in M2RML schema. Further creating DCV specific classes and properties can work as a guide for a developer to understand and figure out the required elements and

the relation between them easier and in clearer fashion. This is true as M2RML properties have rdfs:range and rdfs:domain properties in their definition.

5.4. Comparing R2RML and M2RML

Generalization of R2RML elements would be enough to efficiently express any mappings and we use them as abstract classes of M2RML vocabulary. As these classes in R2RML are bound to relational data mappings, we have just generalized them and removed their dependency on relational data to define classes that can be used in any kind of mapping; from any source to any destination vocabulary. This generalization is performed since we see definitions of these classes in R2RML are specific to relational data mapping.

The main difference between the two vocabularies is source of data and target vocabularies. R2RML is designed for expressing mappings of relational data to RDF whereas M2RML is designed for expressing mappings of multidimensional data to RDF. Considering target vocabularies, R2RML can have any vocabulary as target vocabulary however M2RML has Data Cube Vocabulary fixed as its target vocabulary.

Beside these differences one of the considerations in designing M2RML was to keep it similar to R2RML as a standard by W3C. As mentioned before this similarity makes it easier for users to adopt and start using it.

Chapter 6. OLAP Cube to RDF/QB Mapping (Case Study, Extending DCV)

In this chapter we represent mapping of an OLAP Cube to an RDF/QB dataset/graph. We will also discuss shortcomings of Data Cube Vocabulary for one to one mapping of OLAP cubes to RDF/QB graphs and as well, we will propose and extension to DCV to overcome the shortcomings and make the vocabulary richer.

6.1. Identifying and Mapping of Elements

In this section we first identify elements of both an OLAP Cube and a RDF/QB graph and then we use our mapping language to map these identified elements.

6.1.1. Anatomy of an OLAP Cube, RDF/QB

An OLAP Cube consists of one or more dimensions as well as one or more measures. Each dimension has at least a default hierarchy and can have multiple hierarchies. Further each hierarchy consists of one or more levels which in turn include members. Members can have properties like labels in different languages or description. So as we can see there is a hierarchical relationship between an OLAP cube elements:

- Cube
- Dimensions
- Hierarchies
- Levels
- Members
- Members properties
- Measures
- Measure Properties

A point to mention here is that once we connect to the cube, we should be able to get down to the lowest level of hierarchy and get all the metadata elements required for the cube. Beside this, we should also be able to identify the cube structure in a bottom-to-top approach as we have the hierarchical relationship between the elements.

On the other hand we have elements of the RDF/QB graph which consists of classes and properties to represent a cube. The relation between classes and properties is demonstrated in the figure below:

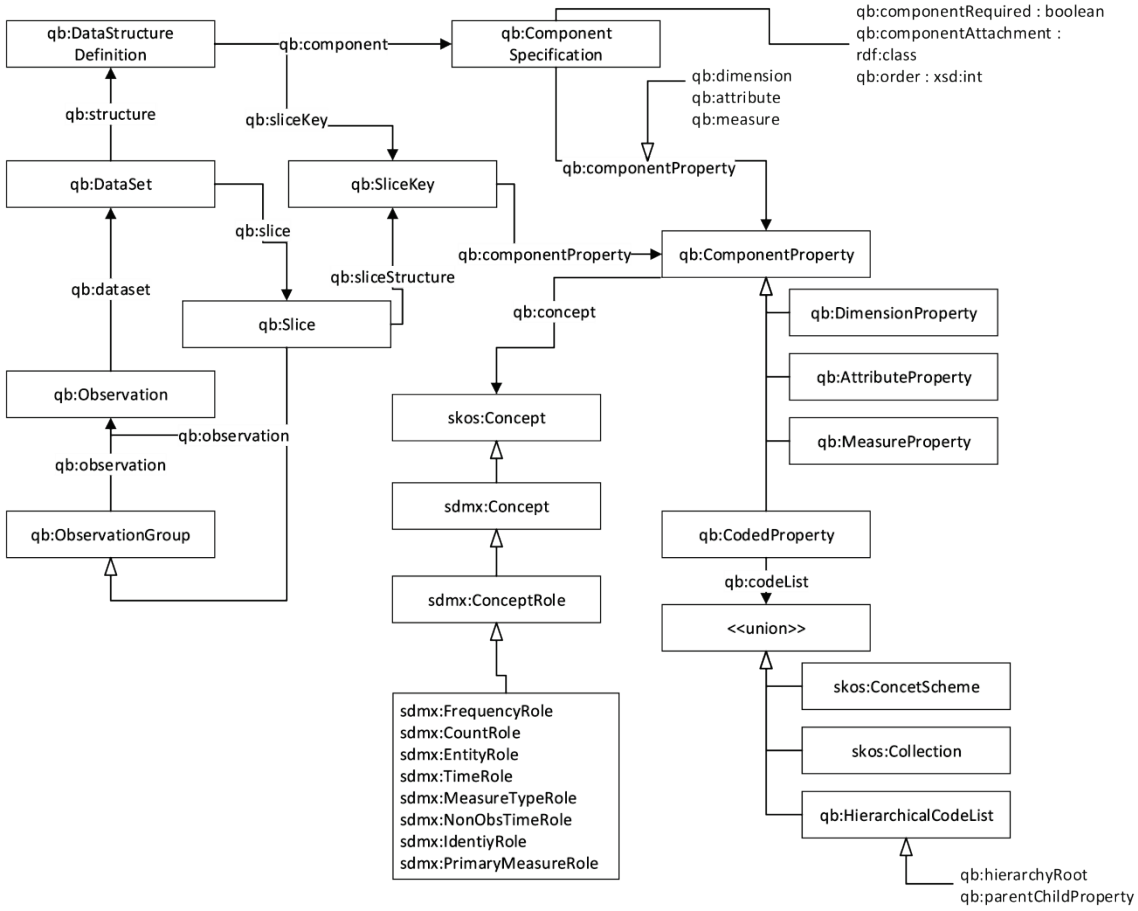

Figure 6.1. Classes and Properties of Data Cube Vocabulary

For more in-depth review of RDF/QB elements please refer to Chapter 3, Analyzing Data Cube Vocabulary.

6.1.2. Mapping OLAP Cube Elements to DCV Elements

In this section we map each element of an OLAP cube to a class in DCV (or a node/element of RDF/QB graph). In the table below, the first column represents an element of an OLAP Cube, the second column represents a class in Data Cube Vocabulary equivalent to the OLAP element, the third column represents minimum properties required for mapping (properties of the identified DCV class) and the forth column shows the range (or possible values) of a particular property. Mapping of each OLAP cube element is discusses more in detail after the table.

Table 6.1. Mapping OLAP Cube Elements to Data Cube Vocabulary Elements

OLAP Cube Element	DCV Class	Property	rdfs:range of property
Cube	qb:DataSet	qb:structure qb:slice	qb:DataStructureDefinition qb:Slice
	qb:DataStructureDefinition	qb:component qb:sliceKey	qb:ComponentSpecification qb:SliceKey
	qb:ComponentSpecification	qb:componentProperty qb:dimension, qb:measure, qb:attribute	qb:ComponentProperty
Dimension	-	-	-
Attribute	qb:DimensionProperty	qb:codeList rdfs:range	qb:ConceptScheme rdfs:Class, owl:Class
Hierarchy	-	-	-
Level	skos:ConceptScheme	skos:hasTopConcept	skos:Concept
Member	skos:Concept	skos:broader skos:inScheme	skos:Concept skos:ConceptScheme
Slice Specification	qb:Slice	qb:sliceStructure qb:DimensionProperty	qb:SliceKey skos:Concept
	qb:SliceKey	qb:componentProperty	qb:DimensionProperty
Measure	qb:MeasureProperty	rdfs:range	

In the following sub-sections we explain mapping of each element more in detail and we provide examples from Adventure Works DW 2012 dataset.

Cube

To represent an OLAP cube in RDF using Data Cube Vocabulary we will use the class qb:DataSet which according to W3C documentation consists of measured observations. DCV's qb:DataSet requires a qb:DataStrcutureDefinition which defines the structure of a qb:DataSet and specifies measures and dimensions in the dataset.

A qb:DataStructureDefinition is connected to qb:DataSet using qb:structure property. Hence as soon as we create a qb:Dataset to represent our OLAP cube we need to create a qb:DataStructureDefinition and connect it to the qb:DataSet using qb:structure property.

As mentioned in previous sections [refer to the section 'Analysis of DCV'] qb:DataStructureDefinition specifies measures and dimensions of the qb:DataSet using a property qb:component which accepts qb:ComponentSpecification as its value. As the name says qb:ComponentSpecification holds specifications of a component (measure/dimension).

The component is specified using qb:ComponentProperty class of DCV and is connected to the qb:ComponentSpecification object using the property qb:ComponentProperty. A qb:ComponentProperty has sub-classes qb:DimensionProperty, qb:MeasureProperty and qb:AttributeProperty which make it more easier/clearer to use and will be discussed later.

So let's assume we have the Adventure Works DW 2012 database as our source of OLAP cube. Following is how we map it to its RDF/QB equivalent:

```
ex:Adventure_Works_DW2012
      rdf:type qb:DataSet.
ex:dsd_Adventure_Works_DW2012
      rdf:type qb:DataStructureDefinition.
ex:Adventure_Works_DW2012
      qb:structure ex:dsd_Adventure_Works_DW2012.
```

Here we have defined ex:Adventure_Works_DW_2012 as a qb:Dataset. As can be seen we have used the cube name to name our qb:DataSet object. Then we have defined a qb:DataStructureDefinition for ex:Adventure_Works_DW2012, named it ex:dsd_Adventure_Works_DW2012 using cube name and connected it to ex:Adventure_Works_DW_2012 using qb:structure property.

Our data structure definition requires qb:ComponentSpecification for dimensions and measures (as can be seen in the table) however we'll discuss them in the following sections where we discuss dimensions and measures. Below is the naming convention we've used for our dataset and data structure definition as well as how our graph looks like at this stage:

Dataset Name: {CubeName}

DSD Name: 'dsd_' + {CubeName}

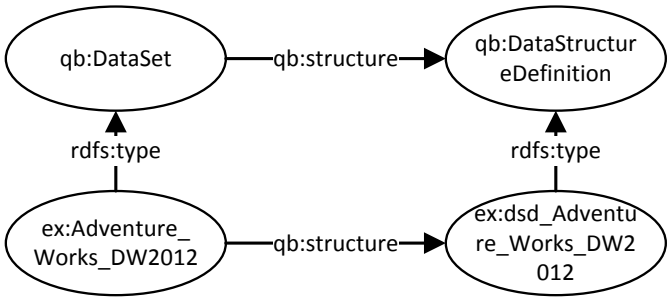

Figure 6.2 Maping an OLAP Cube to DCV Dataset and Data Strcuture Definition

Dimensions

A dimension in an OLAP cube consists of one or more attributes and one or more hierarchies built from attributes. In our example database, Adventure Words DW2012, dimensions are 'Product', 'Customer', 'Due Date', 'Ship Date' and etc. A dimension in OLAP cube is like a group of related attributes that represent different characteristic of a dimension/aspect. At this point of time there is no equivalent class or group of classes and properties to represent this element of an OLAP Cube. Without having this element mapped, we can still represent the OLAP cube however it would be beneficial to have such an element as we can group all related attributes together and as

we can have more meaningful hierarchies that show the relationship between dimensional attributes (equivalent to dimensions in RDF/QB).

Later when we present our extended RDF cube model (an extension to Data Cube Vocabulary) we will include a class to represent this OLAP cube element and to overcome lack of semantics in the current version of DCV.

Attributes

Attributes are like properties of members of a dimension; and a dimension can have multiple attributes.

In our example, if we consider the 'Product' dimension, some of the attributes are 'Product Line', 'Model Name', 'Product Name' and etc.

An attribute in a dimension of an OLAP cube can be mapped to a qb:ComponentProperty or to be more precise to a qb:DimensionProperty in Data Cube Vocabulary. Linking our qb:DimensionProperty object to the qb:DataStrcutureDefinition we have created before is by using qb:component property, qb:ComponentSpecification class and qb:dimension property (sub-property of qb:componentProperty). Instances of qb:ComponentSpecification can be blank nodes as in the W3C document however here we name them explicitly so that we can explain the structure and mapping more clearly.

We consider the 'Product Line' attribute of 'Product' dimension in Adventure Works DW2012 cube and we map them to their equivalent in DCV as follows:

```
ex-dim:Product_Line rdf:type qb:DimensionProperty.

ex:compSpec_Product_Line rdf:type
qb:ComponentSpecification.

ex:compSpec_Product_Line qb:dimension ex-dim:Product_Line.

ex:dsd_Adventure_Works_DW2012 qb:component
ex:compSpec_Product_Line.
```

We have mapped the 'Product Line' attribute to a qb:DimensionProperty and used the attribute name to name our object ex-dim:Product_Line. Then we have created an instance of qb:ComponentSpecification class, ex:compSpec_Product_Line using the

attribute name and connected it to the ex-dim:Product_Line using qb:dimension property. Further to connect this piece of graph to the earlier piece we used qb:component property to link it to our previously defined instance of qb:DataStructureDefinition, ex:dsd_Adventure_Works_DW2012.

Below is the naming convention we have used for naming new objects and the graph so far we have:

Dimension Name: {AttributeName}

Component Specification Name: 'compSpec_' + {AttributeName}

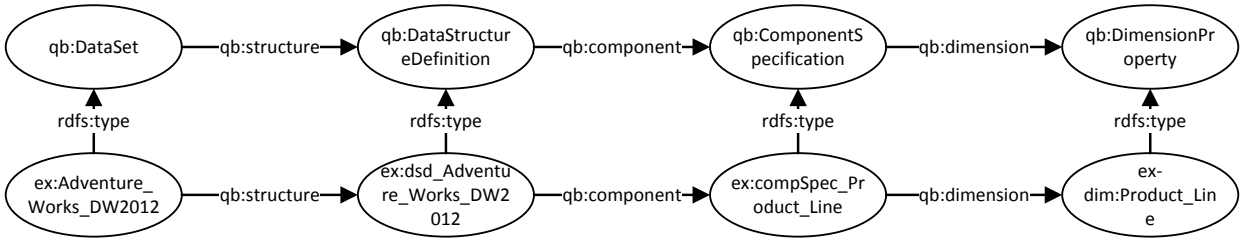

Figure 6.3. Mapping an OLAP Cube Dimension Attribute to DCV DimensionProperty

For each attribute we should have the same set of triples, therefore if we'd like to add other attributes like 'Model Line' and 'Product Name' as dimensions we should have:

```
#Triples for adding 'Model Line' as a dimension

ex-dim:Model_Name rdf:type qb:DimensionProperty.

ex:compSpec_Model_Name rdf:type qb:ComponentSpecification.

ex:compSpec_Model_Name qb:dimension ex-dim:Model_Name.

ex:dsd_Adventure_Works_DW2012 qb:component
ex:compSpec_Model_Name.
```

```
#Triples for adding 'Product Name' as a dimension

ex-dim:Product_Name rdf:type qb:DimensionProperty.
```

```
ex:compSpec_Product_Name rdf:type
qb:ComponentSpecification.

ex:compSpec_Product_Name qb:dimension ex-dim:Product_Name.

ex:dsd_Adventure_Works_DW2012 qb:component
ex:compSpec_Product_Name.
```

And our graph would look like:

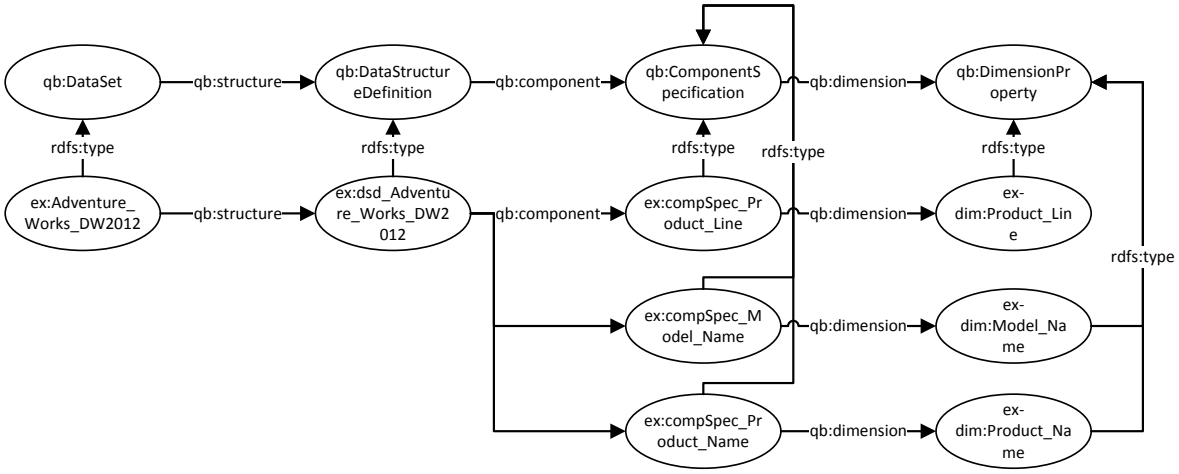

Figure 6.4. Mapping Multiple OLAP Cube Dimension Attributes to DCV qb:DimensionProperty

Hierarchy

Each dimension in an OLAP cube can have one or more hierarchy. Hierarchies are built from attributes and represent parent child relationship between attributes and their members. In fact an attribute in a dimension is considered a hierarchy and in XMLA and OLAP cube metadata they are actually called attribute hierarchies or simply hierarchies.

At the time being there is no direct equivalent for dimension hierarchies in Data Cube Vocabulary. Data Cube Vocabulary uses skos:ConceptScheme and qb:HierarchicalCodeList classes to represent list of values for an element. Members in instances of these classes can then have parent-child relationship with members of other instances using skos:broader or skos:narrower properties. This representation is

more suitable for levels of a hierarchy in dimension of an OLAP cube and hence we can say there is no existing class in DCV that represent a hierarchy.

Having a hierarchy we can better represent semantics and relationships between different skos:ConceptSchemes or qb:HierarchicalCodeLists. We will present this as an extension to DCV later in the following chapters.

Level

Each hierarchy (that we discussed before) consists of one or more levels which define the relationship of members in them (levels). Each level is actually an attribute in the dimension that is used to define the relationship/hierarchy. In a parent-child relationship that a hierarchy represents the higher levels represent parent and usually are less granular however as we navigate down the hierarchy granularity of levels increases and they represent child levels. This relationship would exist for members of the levels as we can see in the following example. If we consider a date dimension, we can have year, month and date as attributes and we can have a hierarchy as below:

- Date Hierarchy:
    - Year
        - Month
            - Day

And if we consider possible members, we can have:

Table 6.2. Date Levels and Related Members

Year	Month	Day
2011	January,	1, 2, 3, …, 29, 30, 31
2011	February	1, 2, 3, …, 28
2011	…	…
2011	November	1, 2, 3, …, 29, 30
2011	December	1, 2, 3, …, 29, 30, 31
2012	January,	1, 2, 3, …, 29, 30, 31
2012	February	1, 2, 3, …, 28
2012	…	…

2012	November	1, 2, 3, …, 29, 30
2012	December	1, 2, 3, …, 29, 30, 31
2013	January,	1, 2, 3, …, 29, 30, 31
2013	February	1, 2, 3, …, 28
2013	…	…
2013	November	1, 2, 3, …, 29, 30
2013	December	1, 2, 3, …, 29, 30, 31

As shown in the table, we have following possible values for our date dimension attributes:

- Year:
  - 2011,
  - 2012,
  - 2013
- Month:
  - January,
  - February,
  - …,
  - November,
  - December
- Day:
  - 1
  - 2
  - 3
  - …
  - 28 or 30 or 31 (depending on the month)

And this means number of distinct values for our attributes would be:

- Year: 3 distinct value
- Month: 12 distinct value
- Day: 31 distinct value

As can be seen the top most attribute (year) has less granularity than its lower level attribute (month) which in turn is less granular than the lowest level attribute (day). In other words year has multiple months and a month has multiple days; a one-to-many relationship.

Now going back to Data Cube Vocabulary, a level can be represented using a skos:ConceptScheme or qb:HierarchicalCodeList and will be connected to its members using skos:hasTopConcept for skos:ConceptScheme and qb:hierarchyRoot for qb:HierarchicalCodeList. Once we have instances of skos:ConceptScheme or qb:HierarchicalCodeList we can connect them to a dimension (attribute or attribute hierarchy in an OLAP cube) using qb:codeList property.

Another point that should be mentioned is a good practice pointed out in W3C document for DC and is creating a separate rdfs:Class or owl:Class to represents objects/members of the level. Therefore when we are using a list of values for a dimension we restrict the values It can accept using rdfs:range property and the class we create for members of the list. The example below will make it clearer.

If we consider the ‘Product Model Line’ hierarchy of ‘Product’ dimension in our example cube, we have:

- Product Model Line (Hierarchy)
    - Level 1: Product Line (attribute)
    - Level 2: Model Name (attribute)
    - Level 3: Product Name (attribute)

Attributes used are ‘Product Line’, ‘Model Name’ and ‘Product Line’ which we have already mapped to qb:DimensionProperty however using them in a hierarchy as levels we will have:

```
#Triples to define Product Line level

ex-code:Product_Line rdf:type skos:ConceptScheme.

ex-class:Product_Line rdf:type rdfs:Class.

ex-class:Product_Line rdf:type owl:Class.
```

```
ex-dim:Product_Line qb:codeList ex-code:Product_Line.

ex-dim:Product_Line rdfs:range ex-class:Product_Line [add
this to script]
```

```
#Triples to define Model Name level

ex-code:Model_Name rdf:type skos:ConceptScheme.

ex-class:Model_Name rdf:type rdfs:Class.

ex-class:Model_Name rdf:type owl:Class.

ex-dim:Model_Name qb:codeList ex-code:Model_Name.

ex-dim:Model_Name rdfs:range ex-class:Model_Name [add this
to script]
```

```
#Triples to define Product Name level

ex-code:Product_Name rdf:type skos:ConceptScheme.

ex-class:Product_Name rdf:type rdfs:Class.

ex-class:Product_Name rdf:type owl:Class.

ex-dim:Product_Name qb:codeList ex-code:Product_Name.

ex-dim:Product_Name rdfs:range ex-class:Product_Name [add
this to script]
```

As shown, we have created objects ex-code:Product_Line, ex-code:Model_Line and ex-code:Product_Name as skos:ConceptScheme. Then we created classes ex-class:Product_Line, ex-class:Model_Name an ex-class:Product_Name for each concept scheme and at the end connected them to their respective dimensions using qb:codeList property. At the end using classes we created for each level, we restrict acceptable values of related dimension to instances of that class using rdfs:range property for the dimensions (attributes or attribute hierarchies of a dimension in OLAP cube).

It is also obvious that there is no object representing ‘Product Model Line’ hierarchy and there is no way we can identify relationships between levels with the

existing triples we have. This lack of semantics is resulted by not having the hierarchy mapped which will be addressed later (Although without a hierarchy we can identify relationship between levels/lists by looking at their members and relationships between members).

Following is the naming convention and how triples form a graph. Arrangement of nodes is changed to fit them into the page; as well different colors are used for different components to make the graph more readable and understandable and a legend for colors is added at the bottom of the image:

Level Name: {LevelName}

Class Name: {LevelName}

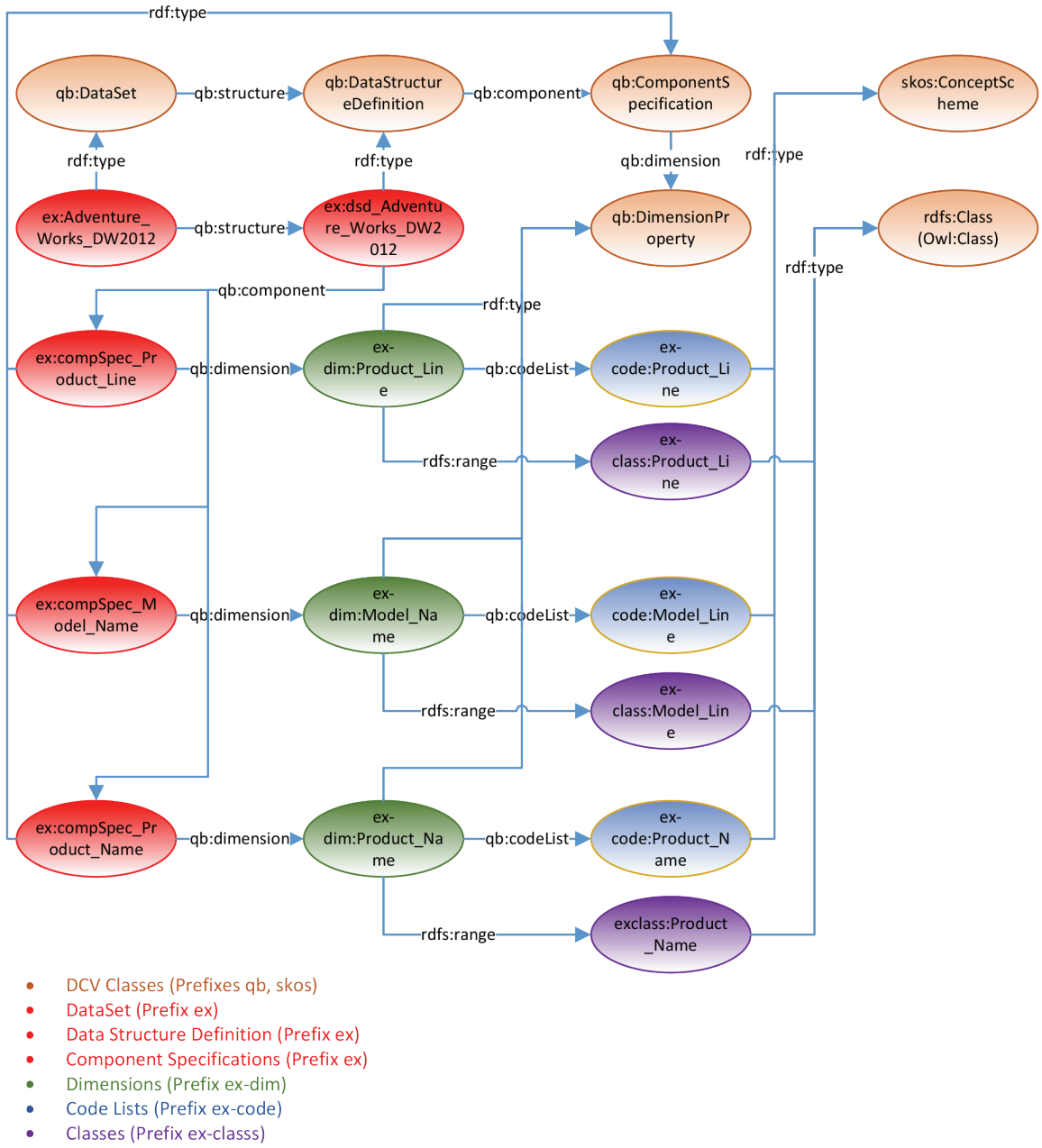

Figure 6.5. Mapping OLAP Cube Dimension Levels to SKOS ConceptSchemes

Member

As mentioned before each level has a list of values which are called members. Members in DCV can be instances of skos:Concept and will be linked to instances of skos:ConceptScheme and qb:HierarchicalCodeList using skos:hasTopConcept and qb:hierarchyRoot properties respectively. Members can also participate in parent-child

relationships using skos:broader and skos:narrower properties as well as qb:parentChildProperty which is currently the main method for finding relationships between levels/lists. Also each member will be defined as an instance of the class we created in the previous section for the code lists/levels.

Considering our example members of 'Product Line' level/attribute are 'Components', 'Accessory', 'Mountain' and they will be created using DCV as below:

```
#Triples to define member Accessory

ex-member:memberAccessory rdf:type skos:Concept.

ex-member:memberAccessory rdfs:label "Accessory".

ex-member:memberAccessory rdf:type ex-class:Product_Line.

ex-member:memberAccessory skos:inScheme ex-
code:Product_Line.

ex-code:Product_Line skos:hasTopConcept ex-
member:memberAccessory.
```

```
#Triples to define member Components

ex-member:memberComponents rdf:type skos:Concept.

ex-member:memberComponents rdfs:label "Components".

ex-member:memberComponents rdf:type ex-class:Product_Line.

ex-member:memberComponents skos:inScheme ex-
code:Product_Line.

ex-code:Product_Line skos:hasTopConcept ex-
member:memberComponents.
```

```
#Triples to define member Mountain

ex-member:memberMountain rdf:type skos:Concept.

ex-member:memberMountain rdfs:label "Mountain".
```

```
ex-member:memberMountain rdf:type ex-class:Product_Line.

ex-member:memberMountain skos:inScheme ex-
code:Product_Line.

ex-code:Product_Line skos:hasTopConcept ex-
member:memberMountain.
```

As shown each member is created as a skos:Concept and has the property rdfs:label which describes the name more appropriately. Further each created member is linked to the class representing available values of the list using rdf:type property and linked to the list/level itself using skos:inScheme and skos:hasTopConcept properties.

The naming convention we have used for members is:

Member Name: 'member' + {MemberName}

Member Label: {MemberName}

At this stage our graph would look like below:

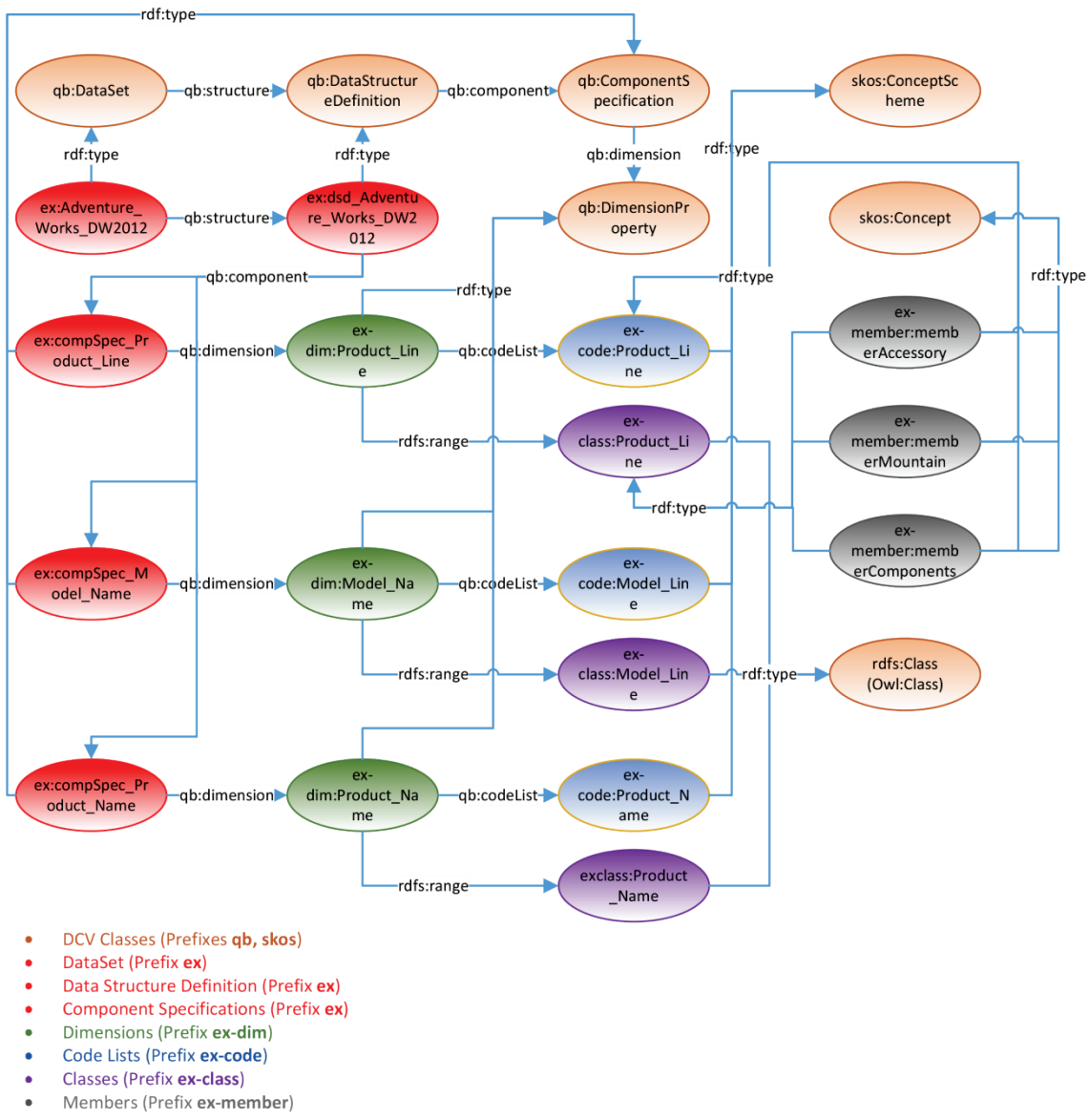

Figure 6.6. Mapping OLAP Cube Dimension Members to SKOS/DCV Concepts

Measures

Measure or facts are numerical values that would be aggregated using various functions; therefore their main characteristic is what they represent as a value and the aggregated function or the formula used to define them. In DCV there is a class qb:MeasureProperty that allows us to directly map OLAP measures. This class is sub class of the abstract class qb:ComponentProperty and is linked to a qb:ComponentSpecification via qb:measure property (sub property of

qb:componentProperty) which then links to the qb:DataStructureDefinition using qb:component property (similar to a qb:DimensionProperty).

DCV allows users to use measures in two different ways; simply use multiple measures linked to observations or define a measure dimensions that includes all measures and specify them for an observation separately [reference to DCV document example]. As mentioned in the official W3C documentation for DCV the first approach is more suitable for OLAP cubes and hence we will use the first approach to link our measures to the dataset (Although in OLAP we can have a measure dimension too).

Taking look at our example dataset we have multiple measure groups such as 'Internet Sales', 'Reseller Sales', 'Sale Quotas', etc. We should mention there is no direct equivalent element to OLAP measure groups at this time. [We might be able to address it] If we take a look at 'Internet Sales' group, we can identify measures such as 'Internet Sales Count', 'Internet Sales – Unit Price', 'Internet Sales – Order Quantity' and etc. if we look into design of our example cube we can identify aggregated functions used for these measures however at the moment there is no standard way to include aggregated function semantics to a measure in DCV; therefore what we do is simply trying to map each measure to qb:MeasureProperty in DCV and connect it to our RDF/QB.

Following triples are used to map measures 'Internet Sales - Order Quantity' and 'Internet Sales - Unit Price':

```
#Triples to define 'Order Quantity' measure

ex-measure:Order_Quantity rdf:type qb:MeasureProperty.

ex:compSpec_Order_Quantity rdf:type
qb:ComponentSpecification.

ex:compSpec_Order_Quantity qb:measure ex-
dim:Order_Quantity.

ex:dsd_Adventure_Works_DW2012 qb:component
ex:compSpec_Order_Quantity.
```

```
#Triples to define 'Unit Price' measure
```

```
ex-measure:Unit_Price rdf:type qb:MeasureProperty.

ex:compSpec_Unit_Price rdf:type qb:ComponentSpecification.

ex:compSpec_Unit_Price qb:measure ex-dim:Unit_Price.

ex:dsd_Adventure_Works_DW2012 qb:component
ex:compSpec_Unit_Price.
```

First we have defined an instance of class qb:MeasureProperty for each measure ex-measure:Order_Quantity and ex-measure:Unit_Price. Similar to instances of qb:DimensionProperty, qb:MeasurePropertys link to instances of qb:ComponentSpecification and hence we have created ex:compSpec_Unit_Price and ex:compSpec_Order_Quantity and linked them to their respective measure via qb:measure property. Then we linked these component specifications to our data structure definition ex:dsd_Adventure_Works_DW2012 which we created at the start using qb:component property.

The naming convention we have used and the graph we have so far is as below:

Measure Name: {MeasureName}

Component Specification Name: 'compSpec_' + {MeasureName}

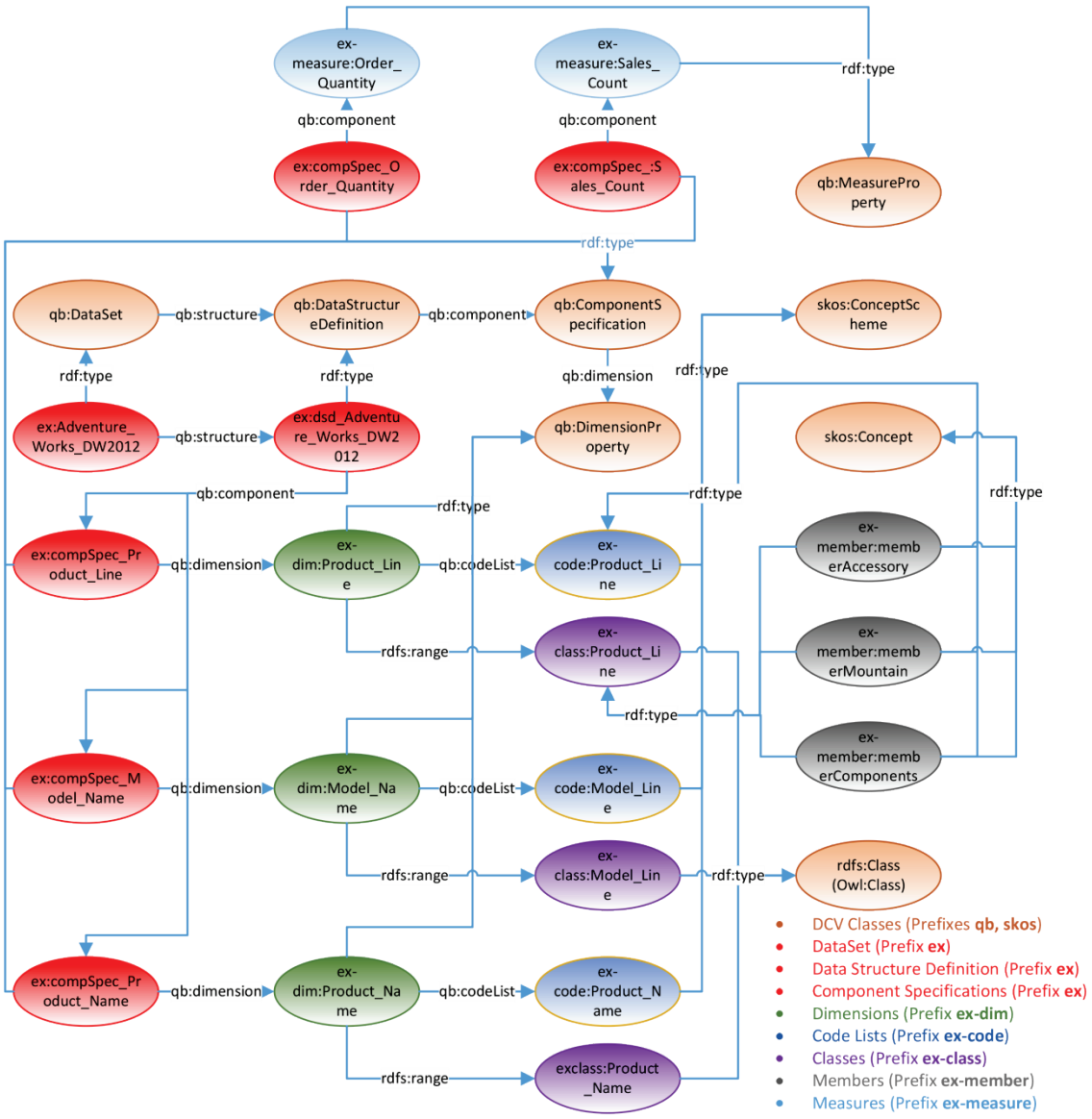

Figure 6.7. Mapping OLAP Cube Measure to DCV MeasureProperties

Slice

Slices are used in MDX queries to filter the data returned by the SELECT statement. In a slicer axis of an MDX query one can specify members of dimensions and the query will return the aggregated value for the intersection of those dimension members. If multiple members from a dimension are to be specified they should form a set and then include the set in the slicer axis in the WHERE clause(Microsoft Corp., 2013). For instance in the example query below we are filtering data for members

‘Accessory’ and ‘Mountain’ from ‘Product Line’ attribute in ‘Product’ dimension and ‘Canada’ from ‘Country’ attribute in ‘Customer’ dimension. The result of this query would be total sales count aggregated for ‘Accessory’ and ‘Mountain’ product line for all years.

```
SELECT

      {[Measures].[Internet Sales Count]} ON COLUMNS,

      [Date].[Calendar Year].MEMBERS ON ROWS

FROM

      [Analysis Services Tutorial]

WHERE (

      {[Product].[Product Model Lines].[Product
Line].&[Accessory], [Product].[Product Model
Lines].[Product Line].&[Mountain]}

      , [Customer].[Customer Geography].[Country-
Region].&[Canada])
```

The result of query would look like below and as we can see the slicer axis hasn’t changed what is returned on rows and columns but instead filtered values for the specified combination of members:

Table 6.3. Slice of an OLAP Cube

	Internet Sales Count
All	5874
CY 2005	6
CY 2006	31
CY 2007	2442
CY 2008	3395
CY 2009	(null)
CY 2010	(null)

The same concept exists in Data Cube Vocabulary. A slice is defined as an instance of class qb:Slice. This class has a property qb:sliceStructure which accepts an instance of class qb:SliceKey as value. A qb:SliceKey defines which dimensions would

be fixed and members of which dimensions would be specified in the slicer; it would be connected to qb:DimensionPropertys using qb:componentProperty property. The qb:Slice will then include specific values of specified dimensions (qb:DimensionProperty). At last qb:Slice will be linked to qb:DataSet and qb:SliceKey is linked to qb:DataStructureDefinition. Taking look at an example makes it clearer:

```
#Triples to define a slice

ex:AWD_Slice_1 rdf:type qb:Slice.

ex:AWD_SliceKey_1 rdf:type qb:SliceKey.

ex:AWD_Slice_1 qb:sliceStructure ex:AWD_SliceKey_1.

ex:AWD_SliceKey_1 qb:componentProperty ex-dim:Product_line.

ex:AWD_SliceKey_1 qb:componentProperty ex-dim:Country.

ex:AWD_Slice_1 ex-dim:Product_Line ex-
member:memberAccesory.

ex:AWD_Slice_1 ex-dim:Product_Line ex-
member:memberMountain.

ex:AWD_Slice_1 ex-dim:Country ex-member:memberCanada.

ex:dsd_Adventure_Works_DW2012 qb:sliceKey
ex:AWD_SliceKey_1.

ex:Adventure_Works_DW2012 qb:slice ex:AWD_Slice_1.
```

Here we have defined a qb:Slice as ex:AWD_Slice_1; the name is arbitrary and not like name of other elements that come from OLAP element properties; it is also numbered as we can have multiple slices per dataset. Same is true about naming ex:AWDW_SliceKey_1 which is a qb:SliceKey and linked to ex:AWDW_Slice_1 via qb:sliceStructure property. Then we have specified dimensions to be fixed in the slice, ex-dim:Product_Line and ex-dim:Country and linked them to ex:AWDW_SliceKey_1 using qb:componentProperty property. As the next step specific members of these dimensions ex-member:memberAccessory, ex-member:memberMountain and ex-member:memberCanada are linked to the slice ex:AWDW_Slice_1 using their respective qb:DimensionProperty. At last slice key and the slice are connected to data structure definition and to dataset respectively using qb:sliceKey and qb:slice properties.

As mentioned there is no specific naming convention for slice and slice keys. Also we have separated graph representation of slice so that it's more readable and understandable.

Slice Name: {ArbitraryName} + '_Slice' + {A Digit}

Slice Name: {ArbitraryName} + '_SliceKey' + {A Digit}

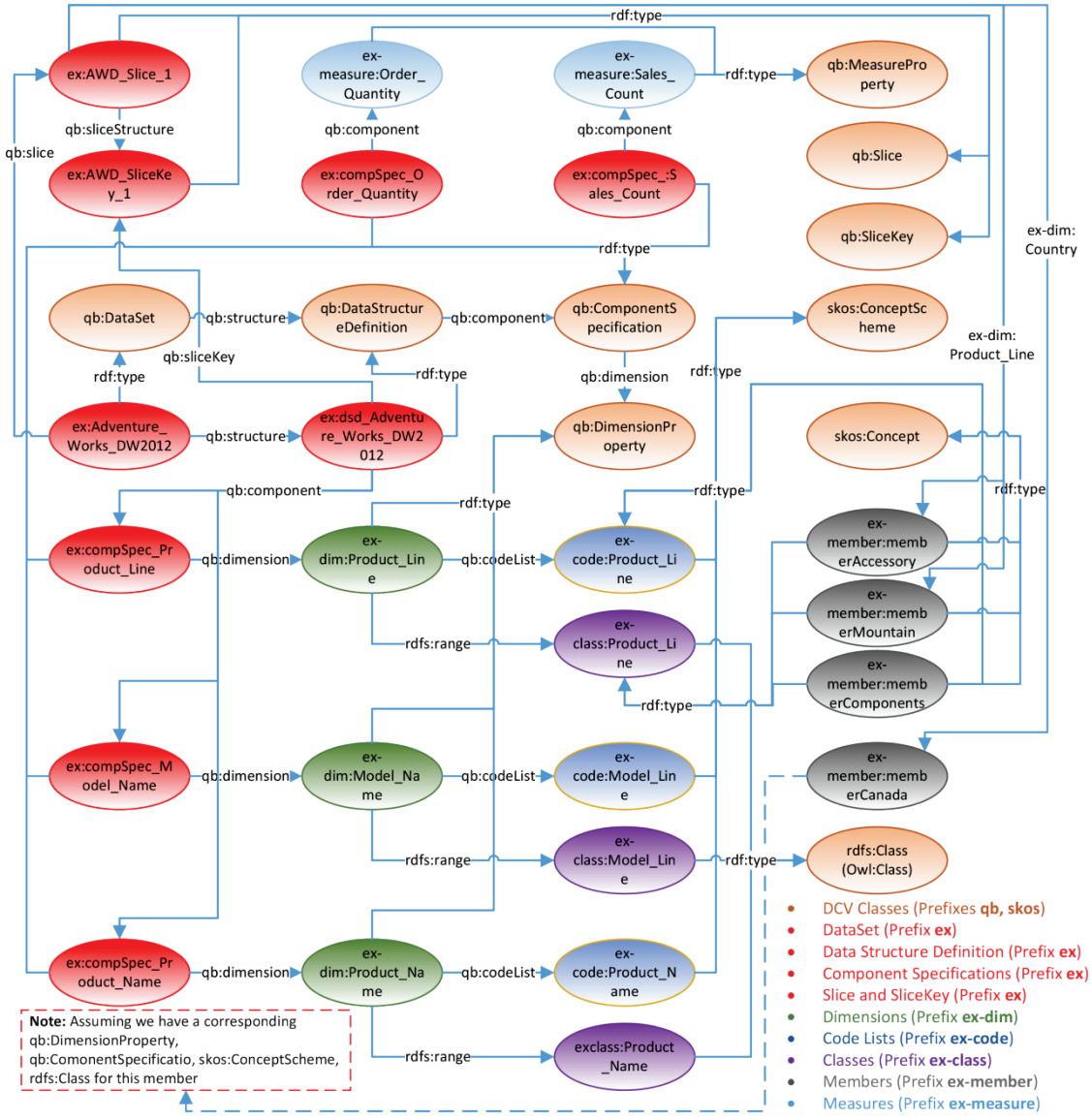

Figure 6.8. Mapping an OLAP Cube Slice to DCV Slice

6.2. Limitations of and Extending Data Cube Vocabulary

In this section we will first identify limitations of Data Cube Vocabulary based on the previous section in which we tried to map OLAP cube elements to Data Cube Vocabulary elements. Then in section 6.4.2 we discuss considerations we have made for extending DCV and designing the new elements and finally in section 6.4.3 we discuss in detail each problem identified in section 6.4.1 and provide our solution based on design considerations in section 6.4.2.

6.2.1. Limitations of Data Cube Vocabulary

In previous section of this chapter we tried to map a sample OLAP cube to an RDF/QB dataset using Data Cube Vocabulary however along the way we have identified some limitations with Data Cube Vocabulary listed below. These limitations are also mentioned in (Etcheverry & Vaisman, Enhancing OLAP Analysis with Web Cubes, 2012) but approached differently as discussed in Chapter 2, Related Work. We will discuss each of these more in detail in the next section.

- Singularity of Dimensions and their Granularity
- Multi-Level Hierarchal Relationships
- Multiple Hierarchies for a Dimension

In fact W3C document on Data Cube Vocabulary(Cyganiak & Reynolds, 2013) it is mentioned that the vocabulary is purposely designed general so that it can be used for multiple sources of multidimensional and statistical data. Sources such as OLAP cubes, spreadsheets and survey data are mentioned as examples of data sources that can be published using data cube vocabulary. It is also mentioned that extensions to this vocabulary are expected to satisfy needs of users for each scenario they might have.

Although the vocabulary is designed to be general we think some elements can be and added to the vocabulary to cope with these limitations. These extension elements are commonly used in all types of multidimensional data sources.

We believe extending Data Cube Vocabulary and adding the few missing elements:

- Makes the vocabulary richer in semantics by adding more classes and properties for common elements in a multidimensional dataset such as dimension groups and hierarchies.
- Adds more elements to data modellers toolbox to model their multidimensional data in DCV/RDF
- Makes it easier for data explorers and analysts to read and understand the data and relation between elements

6.2.2. Design Considerations for Extending Data Cube Vocabulary

In (Cyganiak & Reynolds, 2013)Data Cube Vocabulary as a standard defines a conformant data interchange as one that:

- Uses elements of DCV in a consistent way with their semantics as declared in DCV specification
- Doesn't use elements from other vocabularies instead of ones in DCV that could reasonably be used (although using those elements in addition to DCV elements is permissible).

In our approach we tried to be conformant to DCV considering above mentioned points. For extending the data cube vocabulary we tried to:

- Have no change to the existing elements
- Avoid replacing any element as use exiting elements as much as possible
- Introduce new elements based on abstract/general elements of the vocabulary
- Introduce only the necessary elements for accomplishing our goals and keeping new elements as few and at the same time as efficient as possible

In the next section we will discuss the problems in detail and we provide our solutions to them based on considerations defined here.

6.2.3. Extending Data Cube Vocabulary

In this section we discuss each of the problems mentioned in section 6.4.1 and we provide additional classes and properties with which we can overcome these limitations. At the end of this section we summarize our approach and present all additional classes and properties required.

Singularity of Dimensions and their Granularity

Dimensions or better said qb:DimensionPropertys in DCV has single granularity. For instance as we saw in our example in the previous section a qb:DimensionProperty can represent a single attribute. So if we have a date dimension in our OLAP cube with year, month and day attributes we have to map each attribute to a separate qb:DimensionProperty which can then represent only either of year, month and day.

There is no element in Data Cube Vocabulary that can bring these three dimensional attributes(year, month and day) together and contain multiple granularity levels. In our extension

In our extension we define a class qb-ext:DimensionGroup as a subclass of qb:ComponentSet which is defined as “Abstract class of things which reference one or more ComponentProperties” in (Government Linked Data Working Group, 2013). This class would work as a container of all related dimensions (qb:DimensionPropertys) and can simply work as an equivalent element to OLAP Cube dimension which includes multiple attributes.

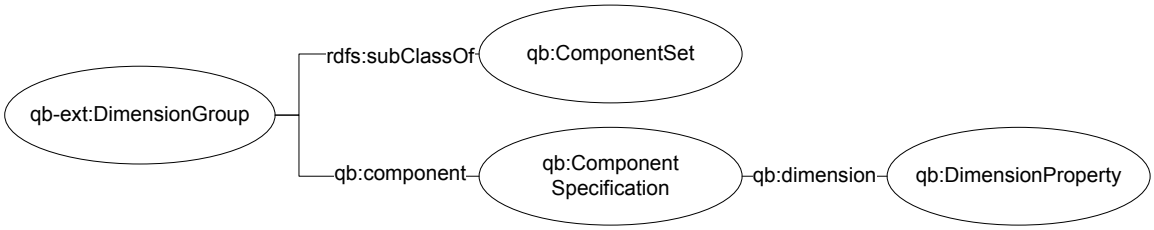

Figure 6.9. qb-ext:DimensionGroup for Grouping Related Dimension Attributes

```
qb-ext:DimensionGroup a rdfs:Class, owl:Class;

      rdfs:subClassOf qb:ComponentSet.
```

```
qb-ext:componentDimGroup a rdf:Property,
owl:ObjectProperty;

      rdfs:domain qb-ext:DimensionGroup;

      rdfs:range qb:ComponentSpecification.
```

Having this element helps us to solve the problem of singular granularity of dimensions in DCV. For Instance if we have the following triples:

```
ex-dim:Year rdf:type qb:DimensionProperty.

ex-dim:Month rdf:type qb:DimensionProperty.

ex-dim:Month rdf:type qb:DimensionProperty.
```

We can group them together to define a qb-ext:DimensionGroup that includes all three:

```
ex-dimgroup:Date

      qb:componentDimGroup  [qb:dimension ex-dim:Year];

      qb:componentDimGroup  [qb:dimension ex-dim:Month];

      qb:componentDimGroup  [qb:dimension ex-dim:Day].
```

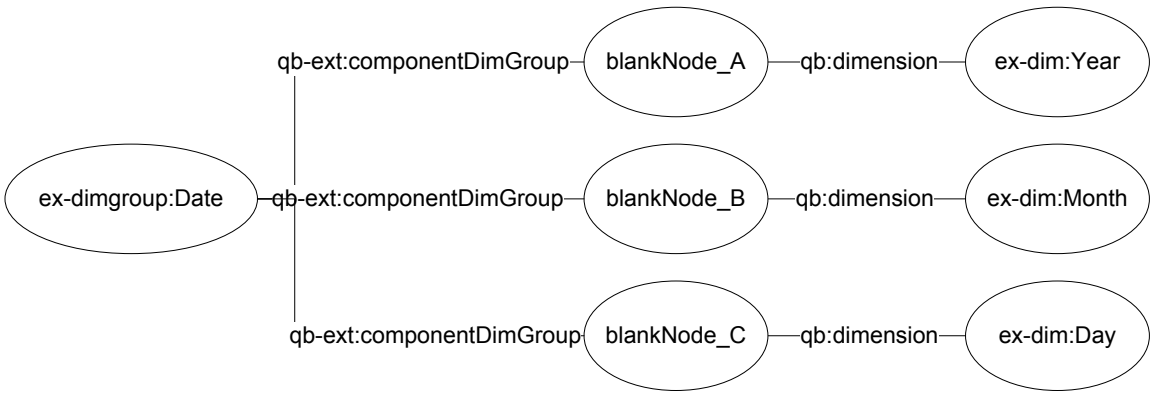

Figure 6.10 Grouping Related Dimensional Attributes using qb-ext:DimensionGroup

As can be see we have used the qb:dimension property to link our qb-ext:DimensionGroup element to each dimension. This is because as we mentioned qb-ext:DimensionGroup is a subclass of qb:ComponentSet which is linked to a qb:DimensionProperty object via qb:dimension property. It might worth mentioning that the qb:dimension property is itself a sub-property of qb:componentProperty.

To keep the extended model simple we don't connect qb-ext:DimensionGroup to neither a qb:DataStructureDefinition nor to qb:DataSet. In fact we think it's not necessary

as one can find qb-ext:DimensionGroups of a qb:DataSet or qb:DataStructureDefinition using a SPARQL query.

In future work one might even extend the extended model here and define classes and properties for the purpose of directly connecting qb-ext:DimensionGroup to qb:DataSet or qb:DataStructureDefinition.

It also worth mentioning that by introducing the new class there would be no change to how observations are represented. Each observation will still be identified using combination of values for qb:DimensionPropertys similar to OLAP cube where each cell in the cube has a value for dimension attributes.

Multi-Level Hierarchal Relationships

Multi-level hierarchies is one of the most common and useful features of OLAP engines. These hierarchies are defined to define relationships between multiple attributes of a hierarchy and to be able to aggregate data at different granularity levels easily.

Currently in DCV there is no way to express multi-level hierarchies at schema level. The two elements closest to representing hierarchical relationships are qb:HierarchicalCodeList and skos:ConceptScheme. However these classes act like a single level hierarchies and have direct link to their top-most level members/concepts. The only way to find the relationship in these hierarchies is by exploring relationship between members of the hierarchies using skos:broader/skos:narrower or qb:parentChild Relationship properties.

Having a structure that can define multi-level hierarchical relationships between dimensions at schema level, makes it much easier for users to define and find relationships between multiple dimensional attributes and their members.

After defining qb-ext:DimensionGroup that groups all related dimensional attributes together we can now define the relationship between them. To be able to define multi-level hierarchical relationships at schema level we have defined qb-ext:Hierarchy class. This class is similar to qb:DataStructureDefinition in design and requires qb:ComponentSpecifications and qb:ComponentProperties.

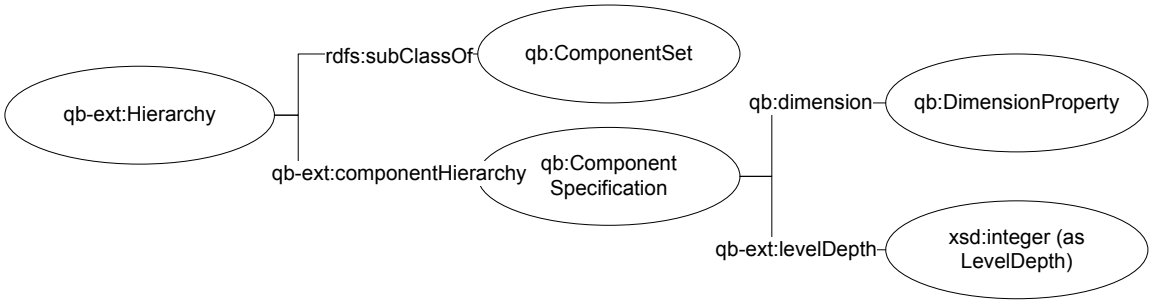

Figure 6.11 Multi-level Hierarchies using qb-ext:Hierarchy

```
qb-ext:Hierarchy a rdf:Class, owl:Class;

     rdfs:subClassOf qb:ComponentSet.
```

```
qb-ext:hierarchy a rdf:Property, owl:ObjectProperty;

     rdfs:domain qb-ext:DimensionGroup;

     rdfs:range qb-ext:Hierarchy.


qb-ext:componentHierarchy a rdf:Property,
owl:ObjectProperty;

     rdfs:domain qb-ext:Hierarchy;

     rdfs:range qb:ComponentSpecification.


qb-ext:levelDepth a rdf:Property, owl:ObjectProperty;

     rdfs:domain qb:ComponentSpeification;

     rdfs:range xsd:int.
```

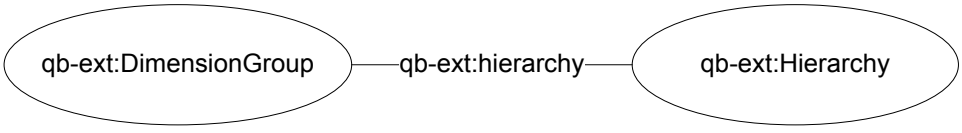

Figure 6.12 Linking qb-ext:DimensionGroup and qb-ext:Hierarchy

The qb-ext:Hierarchy is subclass of qb:ComponentSet mentioned previously and as shown in the figure is connected to qb:ComponentSpecification using qb-ext:componentHierarchy property which is a sub property of qb:componentProperty.

To specify qb:DimensionPropertys (or dimensional attributes in OLAP cube) qb:ComponentSpecifications are linked to qb:DimensionPropertys using qb:dimension property. Each qb:DimensionProperty servers as a level in the hierarchy depth of which is specified using qb-ext:levelDepth property. The value of this property will be a non-negative integer and should be in a way so that one can easily find the order/depth of levels.

As each qb:DimensionProperty has a qb:codeList property that links a skos:ConceptScheme to it we can easily identify members of each level and the relation between them. Further we have also introduced a property qb-ext:memberDepth that helps us quickly identify to which level of hierarchy this member belongs.

As an example we will have a hierarchical relationship between our date dimensional attributes established:

```
ex-hierarchy:SimpleDateHierarchy rdf:type qb-ext:Hierarcy;
      qb-ext:componentHierarchy [qb:dimension ex-dim:Year;
                                 qb-ext:levelDepth 1];
      qb-ext:componentHierarchy [qb:dimension ex-dim:Month;
                                 qb-ext:levelDepth 2];
      qb-ext:componentHierarchy [qb:dimension ex-dim:Day;
                                 qb-ext:levelDepth 3].
```

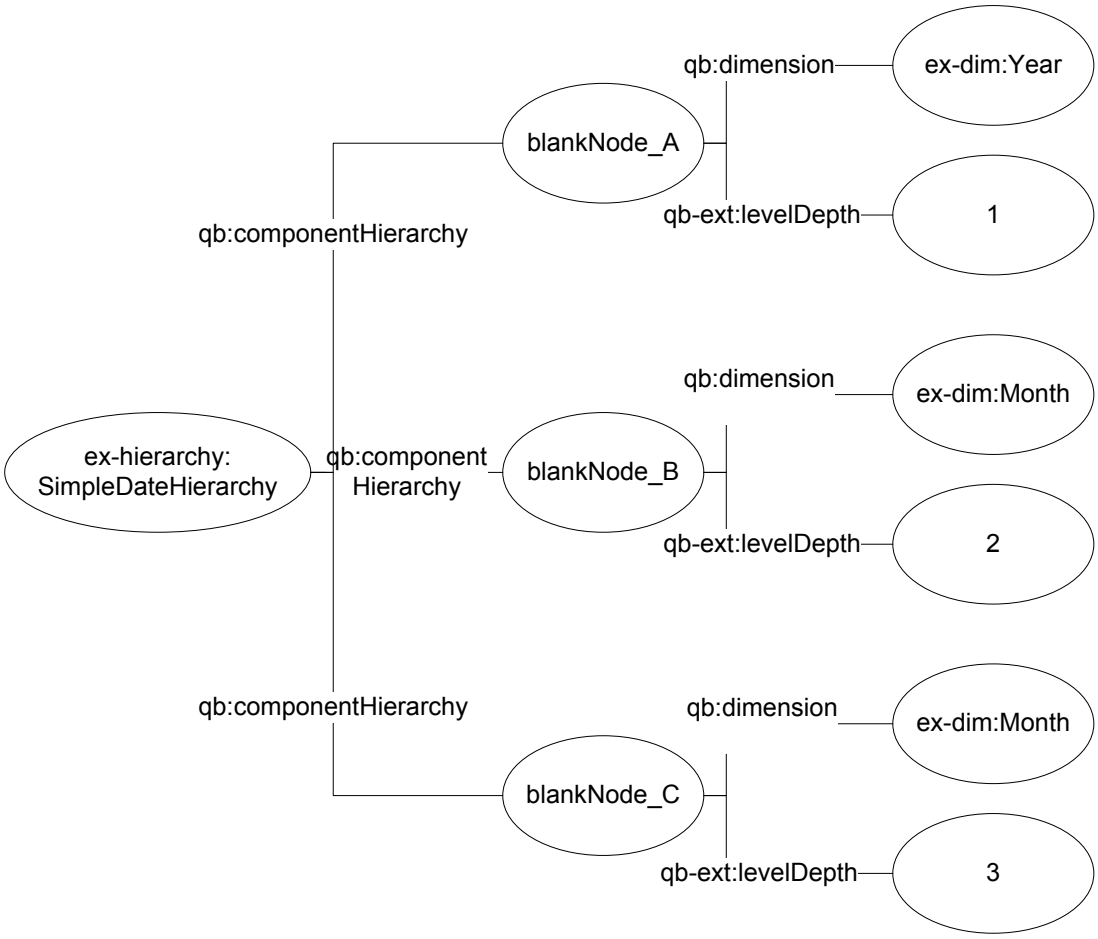

Figure 6.13 Graph representation of an example hierarchy

And then we include the hierarchy in the dimension group we have defined previously:

```
ex-dimgroup:Date qb-ext:hierarchy ex-
hierarchy:SimpleDateHierarchy.
```

Multiple Hierarchies for a Dimension

Another shortcoming with DCV which may be caused by the previous two mentioned limitations is that there is no way to have multiple multi-level hierarchies for a dimension. For example in OLAP cube we can have two different hierarchies for a date dimension such as calendar hierarchy and fiscal hierarchy. These two hierarchies have different levels and different members but both are included in a date dimension.

With limitations that DCV has for singular dimensional attributes and singe level hierarchies it is not possible to have such multiple multi-level hierarchies however using the classes and properties we added in the previous two sections we will be able to have multiple multi-level hierarchies for a group of related dimensional attributes.

To facilitate this feature we have already introduced all required properties and classes in the previous two sections. The only point we didn't mention is that each qb-ext:DimensionGroup can have multiple hierarchies attached to it. For instance if we want to define calendar date hierarchy and fiscal date hierarchy for our date dimension we can have:

```
ex-dimgroup:Date

      qb-ext:componentDimGroup    [qb:dimension ex-dim:Year];

      qb-ext:componentDimGroup    [qb:dimension ex-
dim:Month];

      qb-ext:componentDimGroup    [qb:dimension ex-dim:Date];

      qb-ext:componentDimGroup    [qb:dimension ex-
dim:FiscalYear];

      qb-ext:componentDimGroup    [qb:dimension ex-
dim:FiscalPeriod].
```

Then we can define two different hierarchies for our ex-dimgroup:Date dimension group:

```
ex-hierarchy:CalendarDateHierarchy rdf:type qb-
ext:Hierarcy;

      qb-ext:componentHierarchy [qb:dimension ex-dim:Year;

                        qb-ext:levelDepth 1];

      qb-ext:componentHierarchy [qb:dimension ex-dim:Month;

                        qb-ext:levelDepth 2];

      qb-ext:componentHierarchy [qb:dimension ex-dim:Date;

                        qb-ext:levelDepth 3].
```

and

```
ex-hierarchy:FiscalDateHierarchy rdf:type qb-ext:Hierarcy;

     qb-ext:componentHierarchy [qb:dimension ex-
dim:FiscalYear;

                    qb-ext:levelDepth 1];

     qb-ext:componentHierarchy [qb:dimension ex-
dim:FiscalPeriod;

                    qb-ext:levelDepth 2];

     qb-ext:componentHierarchy [qb:dimension ex-dim:Date;

                    qb-ext:levelDepth 3].
```

and finally attach both hierarchies to our dimension group:

```
ex-dimgroup:Date qb-ext:hierarchy ex-
hierarchy:CalendarDateHierarchy.



ex-dimgroup:Date qb-ext:hierarchy ex-
hierarchy:FiscalDateHierarchy.
```

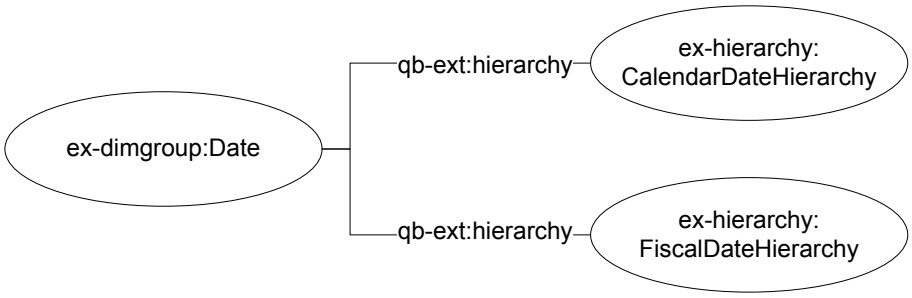

Figure 6.14 Linking multiple hierarchies to a qb-ext:DimensionGroup

As can be seen we have two hierarchies each with three dimensional attributes as levels but our dimension group ex-dimgroup:Date has five dimensional attributes. This is possible as we can reuse dimensional attributes (or qb:DimensionPropertys) in different hierarchies which is similar to how hierarchies are defined in OLAP cubes.

Summary

As we have seen with introducing only 2 additional classes and 3 additional properties we have overcome the limitations of Data Cube Vocabulary which enables us to have a one to one mapping between elements of an OLAP cube and an RDF/QB graph/dataset.

Addition of these classes and properties not only helped us with mapping OLAP cubes but also makes the vocabulary richer in semantics for publishing and using all types of multidimensional data. Grouping related dimensional attributes, multi-level hierarchies and multiple hierarchies per dimension is common for all sources of dimensional data and are used to enable critical OLAP operations such as drilldown and rollup.

With only five classes and properties added in total we kept the extension minimal but highly efficient and usable. Also all classes and properties added are based on abstract classes and properties of Data Cube Vocabulary which makes it easier to be used along with existing elements of the vocabulary. And as we mentioned we stayed conformant to the standard using existing elements of the vocabulary as much as possible and not replacing any existing element.

We should also mention that adding these elements doesn't have a single effect on all datasets that have published before. The elements are designed in a way to make the model more comprehensive with no change to the existing elements or semantics. Hence every dataset regardless of time of publication (already published or going to be published in future) can use these additional elements to make their RDF/QB dataset more complete, readable and useful.

```
#Classes:

qb-ext:DimensionGroup a rdfs:Class, owl:Class;

      rdfs:subClassOf qb:ComponentSet.

qb-ext:Hierarchy a rdf:Class, owl:Class;

      rdfs:subClassOf qb:ComponentSet.

#Properties:
```

```
qb-ext:hierarchy a rdf:Property, owl:ObjectProperty;

      rdfs:domain qb-ext:DimensionGroup;

      rdfs:range qb-ext:Hierarchy.


qb-ext:componentHierarchy a rdf:Property,
owl:ObjectProperty;

      rdfs:domain qb-ext:Hierarchy;

      rdfs:range qb:ComponentSpecification.


qb-ext:levelDepth a rdf:Property, owl:ObjectProperty;

      rdfs:domain qb:ComponentSpeification;

      rdfs:range xsd:int.
```

Chapter 7. Conclusion and Further Work

7.1. Contributions

We had three main contributions in this work which are listed and described as below:

- Multidimensional to RDF Mapping Language (M2RML)
- Extension to Data Cube Vocabulary (qb-ext vocabulary)
- A framework for one to one mapping of OLAP cubes to RDF/QB graphs

7.1.1. M2RML

M2RML is a mapping language and an RDF vocabulary which describes and facilitates mapping of multidimensional data to RDF/QB graphs written with Data Cube Vocabulary. M2RML also helps in systematic and automatic mapping of multidimensional data to RDF and has Data Cube Vocabulary as its main target vocabulary for representing multidimensional data.M2RML consists of abstract and DCV-Specific classes where some abstract classes are similar to some R2RML classes generalized and DCV-specific classes are specific and targeted to Data Cube Vocabulary. M2RML is designed in a way to encapsulate repeating information in maps and clarifying relations between Data Cube Vocabulary elements.

7.1.2. Extended Data Cube Vocabulary

Our extension to Data Cube Vocabulary helps in representing dimension groups with multiple levels of granularity consisting of related dimensional attributes as well as representing hierarchical relationships between dimensional attributes at schema level and enables us to have multiple hierarchies per dimension group without replicating and repeating elements. We have a minimalistic approach in designing the extension and we tried to be compliant with the DCV by maximizing our usage of DCV elements. The

result of this approach is addition of only two new classes and four new properties to Data Cube Vocabulary. Another great advantage of our extension is that no single change is required for already published datasets and users can easily add dimension group and hierarchical data to their published datasets.

7.1.3. OLAP Cube to RDF/QB Mapping Framework

The third contribution is our framework or guide for publishing/mapping OLAP cubes as/to RDF/QB datasets. This framework identifies equivalent elements of an OLAP cube and RDF/QB dataset and together with M2RML and extended data cube vocabulary provides a clearer, systematic and automatic method for publishing OLAP cubes as linked data. In this framework M2RML helps with describing the maps for generating required triples and extended DCV elements enables us to have a one to one mapping between elements.

7.2. Proposed (Potential) Aggregation Method for RDF/QB Graph

OLAP is a common approach for data analytics which lets users retrieve aggregated measures efficiently across multiple dimensions and at different levels. In contrast, RDF graphs and SPARQL are not optimized for analytical tasks and once the data is transformed into graph triples, the user might not be able to perform aggregations in reasonable time, especially for large datasets. There have been efforts to address this problem using RDF materialized views such as (Kämpgen & Harth, 2013) however they have their own advantages and disadvantages. A future work can provide a different way of tackling this issue and using the OLAP engine to perform the aggregations.

In this new approach a user can interact with the RDF/QB graph and navigate it; the classes and elements the user reaches in the graph will represent elements of an OLAP cube (as we have the one to one mapping with the extended vocabulary) which will in turn be used to identify dimensions, levels and measures to write an equivalent MDX query. This MDX query can then be sent to the OLAP engine to retrieve the aggregated value which can be returned and stored in the graph for further reference.

There would be no pre-aggregated values in the cube; as the user navigates the RDF/QB graph (using a front end application built upon this work and other used standards), the MDX queries will be formed and sent to the OLAP server. This way we avoid pre-aggregating values as it's not efficient especially for datasets with relatively large number of dimensions, hierarchies, levels and data; and as well we don't rely on limited features and performance of RDF model and SPARQL for analytical (aggregation) tasks.

One might criticize that in our model the OLAP cube and RDF/QB should co-exist and work together which is a valid statement; however our justification is that using both models together and making them interoperable enables us to get more out of the data while performing tasks more efficiently without sacrificing performance. Moreover we might really want to have our data in both models as each model provides us with different way of looking at our data (although they have things in common, RDF for linking data and OLAP for analytics); and again as one size doesn't fit all.

References

Abraham, G. (2013, March 2). *The Semantic Web Architecture.* Retrieved 1 25, 2014, from SaGe: Semantic Agents for Learning Style Prediction: http://semanticsage.blogspot.ca/2013/03/the-semantic-web-architecture.html?q=semnatic+web+stack

Auer, S., Dietzold, S., & Riechert, T. (2006). OntoWiki - A Tool for Social, Semantic Collaboration. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, et al. (Ed.), *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings* (pp. 736-749). Berlin / Heidelberg: Springer.

Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., & Aumueller, D. (2009). Triplify: Light-weight Linked Data Publication from Relational Databases. *Proceedings of the 18th International Conference on World Wide Web* (pp. 621-630). Madrid, Spain: ACM.

Auer, S., Feigenbaum, L., Miranker, D., Fogarolli, A., & Sequeda, J. (2010, June 8). *Use Cases and Requirements for Mapping Relational Databases to RDF.* Retrieved January 25, 2014, from W3C: http://www.w3.org/TR/2010/WD-rdb2rdf-ucr-20100608/

Bizer, C., & Cyganiak, R. (2006). D2R Server – Publishing Relational Databases on the Semantic Web. *Poster at the 5th International Semantic Web Conference (ISWC2006).* http://www4.wiwiss.fu-berlin.de/bizer/pub/Bizer-Cyganiak-D2R-Server-ISWC2006.pdf.

Bizer, C., & Seaborne, A. (2004). D2RQ-treating non-RDF databases as virtual RDF graphs. *Proceedings of the 3rd international semantic web conference (ISWC2004).*

Bizer, C., Cyganiak, R., & Heath, T. (2007). *How to Publish Linked Data on the Web.* Retrieved December 10, 2013, from http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst. , 5* (3), 1-22.

Brickley, D., & Guha, R. (2004). *RDF Vocabulary Description Language 1.0: RDF Schema - W3C Recommendation.* Retrieved December 12, 2013, from http://www.w3.org/TR/rdf-schema/

Brin, S., & Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Seventh International World-Wide Web Conference (WWW 1998).* Brisbane, Australia.

Cyganiak, R., & Reynolds, D. (2013). *The RDF Data Cube Vocabulary - W3C Candidate Recommendation.* Retrieved December 14, 2013, from http://www.w3.org/TR/vocab-data-cube/

Cyganiak, R., Field, S., Gregory, A., Halb, W., & Tennison, J. (2010). Semantic Statistics: Bringing Together SDMX and SCOVO. In C. Bizer, T. Heath, T. Berners-Lee, & M. Hausenblas (Ed.), *CEUR Workshop Proceedings.*

Cyganiak, R., Reynolds, D., & Tennison, J. (2010). *The RDF Data Cube vocabulary.* Retrieved December 1, 2013, from http://publishing-statistical-data.googlecode.com/svn/trunk/specs/src/main/html/cube.html

Das, S., Sundara, S., & Cyganiak, R. (2012). *R2RML: RDB to RDF Mapping Language - W3C Recommendation.* Retrieved November 02, 2013, from http://www.w3.org/TR/r2rml/

Erling, O. (2009). Automated Generation of RDF Views over Relational Data Sources with Virtuoso. http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSQL2RDF.

Etcheverry, L., & Vaisman, A. A. (2012). Enhancing OLAP Analysis with Web Cubes. *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications* (pp. 469-483). Heraklion: Springer-Verlag.

Etcheverry, L., & Vaisman, A. A. (2012). QB4OLAP: A Vocabulary for OLAP Cubes on the Semantic Web. In J. Sequeda, A. Harth, & O. Hartig (Ed.), *COLD, CEUR Workshop Proceedings.* CEUR-WS.org.

Franzon, E. (2012). *Transforming Relational Data to RDF – R2RML Becomes Official W3C Recommendation.* Retrieved December 9, 2013, from semanticweb.com: http://semanticweb.com/transforming-relational-data-to-rdf-r2rml-becomes-official-w3c-recommendation/

Government Linked Data Working Group. (2013, July 27). *The Data Cube Vocabulary Ontology.* Retrieved December 23, 2013, from http://publishing-statistical-data.googlecode.com/svn/trunk/specs/src/main/vocab/cube.ttl#

Graham, K., & Caroll, J. J. (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax - W3C Recommendation.* Retrieved December 10, 2013, from http://www.w3.org/TR/rdf-concepts/

Hausenblas, M., Halb, W., Raimond, Y., Feigenbaum, L., & Ayers, D. (2009). SCOVO: Using Statistics on the Web of Data. *Extended Semantic Web Conference , 5554*, 708-722.

Heath, T., & Bizer, C. (2011). *Linked Data: Evolving the Web into a Global Data Space.* Morgan & Claypool Publishers.

Hyde, J. (2013, January 17). Retrieved December 20, 2013, from olap4j: Open Java API for OLAP: http://www.olap4j.org/

International Organisation for Standardisation. (2005). *ISO/TS 17369:2005 - Statistical data and metadata exchange (SDMX).*

Jacobs, I., & Walsh, N. (2004). *Architecture of the World Wide Web, Volume One - W3C Recommendation.* Retrieved December 2013, from http://www.w3.org/TR/webarch/

Kämpgen, B., & Harth, A. (2013). No Size Fits All - Running the Star Schema Benchmark with SPARQL and RDF Aggregate Views. *ESWC*, (pp. 290-304).

Kampgen, B., & Harth, A. (2011). Transforming Statistical Linked Data for Use in OLAP Systems. *Proceedings of the 7th International Conference on Semantic Systems* (pp. 33-40). Graz, Austria: ACM.

Kampgen, B., O'Riain, S., & Harth, A. (2012). Interacting with Statistical Linked Data via OLAP Operations. *International Workshop on Linked APIs for the Semantic Web (LAPIS 2012)* .

Lee, T.-B. (2006). *Linked Data*. Retrieved December 2013, from http://www.w3.org/DesignIssues/LinkedData.html

McGuinness, D. L., & Harmelen, F. v. (2004). *OWL Web Ontology Language.* Retrieved December 12, 2013, from http://www.w3.org/TR/owl-features/

Microsoft Corp. (2013). *Specifying the Contents of a Slicer Axis*. Retrieved December 21, 2013, from http://technet.microsoft.com/en-us/library/ms146047.aspx

Microsoft Corp. (2013). *XML for Analysis Reference (XMLA)*. Retrieved December 20, 2013, from http://technet.microsoft.com/en-us/library/ms186604.aspx

Miles, A., & Bechhofer, S. (2009). *SKOS Simple Knowledge Organization System Reference - W3C Recommendation*. Retrieved November 29, 2013, from http://www.w3.org/TR/2009/REC-skos-reference-20090818/

Moreira, F. d., & de Freitas Jorge, E. M. (2012). SPARQL2MDX: Um Componente de Traduc ao de Consultas em Ontologia para Data Warehousing.

Obitko, M. (2007). *Semantic Web Architecture.* Retrieved 1 25, 2014, from Ontologies and Semantic Web: http://obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html%20Semantic%20Web%20Architecture

RDB2RDF Working Group. (2012). *A Direct Mapping of Relational Data to RDF, W3C Recommendation.* Retrieved December 9, 2013, from W3C Website: http://www.w3.org/TR/rdb-direct-mapping/

RDB2RDF Working Group. (2012). *R2RML: RDB to RDF Mapping Language, W3C Recommendation.* Retrieved November 02, 2013, from W3C Website: http://www.w3.org/TR/r2rml/

Ruback, L., Pesce, M., Manso, S., Ortiga, S., Salas, P. E., & Casanova, M. A. (2013). A mediator for statistical linked data. *In Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13)* (pp. 339-341). New York: ACM.

Salas, P. E., Martin, M., Mota, F. M., Breitman, K., Auer, S., & Casanova, M. A. (2012). OLAP2DataCube: An Ontowiki Plugin for Statistical Data Publishing. *Proceedings of the 2nd Workshop on Developing Tools as Plug-ins.* New York, NY, USA: ACM.

Vrandecic, D., Lange, C., Hausenblas, M., Bao, J., & Ding, L. (2010). Semantics of Governmental Statistics Data. *Proceedings of the WebSci10: Extending the Frontiers of Society On-Line.* Raleigh, NC, US.

Appendix A.

Extension to Data Cube Vocabulary (qb-ext)

This section includes our proposed extension to Data Cube Vocabulary as discussed in the thesis in chapter 6.The extension is written in RDF Turtle syntax.

```
@prefix qb-ext: <http://www.cs.sfu.ca/~sghasemi/qb-ext/>.

@prefix qb: <http://purl.org/linked-data/cube#>.

@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

@prefix owl: <http://www.w3.org/2002/07/owl#>.

@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

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


#Classes:

qb-ext:DimensionGroup a rdfs:Class, owl:Class;

      rdfs:subClassOf qb:ComponentSet.


qb-ext:Hierarchy a rdf:Class, owl:Class;

      rdfs:subClassOf qb:ComponentSet.


#Properties:

qb-ext:hierarchy a rdf:Property, owl:ObjectProperty;

      rdfs:domain qb-ext:DimensionGroup;

      rdfs:range qb-ext:Hierarchy.


qb-ext:componentHierarchy a rdf:Property, owl:ObjectProperty;

      rdfs:domain qb-ext:Hierarchy;
```

```
      rdfs:range qb:ComponentSpecification.


qb-ext:componentDimGroup a rdf:Property, owl:ObjectProperty;
      rdfs:domain qb-ext:DimensionGroup;
      rdfs:range qb:ComponentSpecification.


qb-ext:levelDepth a rdf:Property, owl:ObjectProperty;
      rdfs:domain qb:ComponentSpeification;
      rdfs:range xsd:int.
```

Appendix B.

Multidimensional to RDF Mapping Language (M2RML)

Following is the M2RML mapping language proposed and discussed in this thesis for mapping multidimensional data to RDF using Data Cube Vocabulary. The vocabulary is written in RDF Turtle syntax.

```
@prefix qb: <http://purl.org/linked-data/cube#>.

@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

@prefix owl: <http://www.w3.org/2002/07/owl#>.

@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

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

@prefix rr: <http://www.w3.org/ns/r2rml#>.


#created vocabularies

@prefix qb-ext: <http://www.cs.sfu.ca/~sghasemi/qb-ext#>.

@prefix m2r: <http://www.cs.sfu.ca/~sghasemi/m2rml#>.


###Classes:

#Abstract Classes

m2r:TriplesMap

      rdf:type rdfs:Class, owl:Class;

      rdfs:label "Abstract class representing mapping of a
source to destination triples";

      rdfs:seeAlso rr:TriplesMap.


m2r:TermMap
```

```
      rdf:type rdfs:Class, owl:Class;

      rdfs:label "Abstract class representing mapping of a
term in an RDF triple (subject, predicate, object)";

      rdfs:seeAlso rr:TermMap.


#General

m2r:template

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TermMap;

      rdfs:range xsd:string.


m2r:class

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TermMap;

      rdfs:range rdfs:Class.


m2r:constant

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TermMap;

      rdfs:range rdfs:Resource.


m2r:element

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TermMap.


m2r:termType
```

```
      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TermMap;

      rdfs:range m2r:IRI, m2r:BlankNode, m2r:Literal.


m2r:language

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:ObjectMap;

      rdfs:range xsd:String.


m2r:dataType

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:ObjectMap;

      rdfs:range rdfs:Datatype.


#Source Specs

m2r:Source

      rdf:type rdfs:Class, owl:Class;

      rdfs:label "Abstract class representing source element
for a mapping";

      rdfs:seeAlso rr:LogicalTable, rr:R2RMLView,
rr:BaseTableOrView.


m2r:SourceType

      rdf:type rdfs:Class, owl:Class;

      rdfs:label "Abstract Class representing source type of
the data to be mapped".
```

```
m2r:MultidimensionalTable

      rdf:type rdfs:SourceType;

      rdfs:label "SourceType representing a general
multidimensional table".



m2r:OLAPCube

      rdf:type rdfs:SourceType;

      rdfs:label "SourceType representing an OLAP cube".



m2r:Spreadsheet

      rdf:type rdfs:SourceType;

      rdfs:label "SourceType representing a spreadsheet".



m2r:MDX

      rdf:type rdfs:SourceType;

      rdfs:label "SourceType representing an MDX query that
returns multidimensional data as result such as a slicer
axis or a sub-cube".



m2r:source

      rdf:type rdfs:Property, owl:ObjectProperty;

      rdfs:label "Property that links a triples map to a
source";

      rdfs:domain m2r:TriplesMap;

      rdfs:range m2r:Source;

      rdfs:seeAlso rr:logicalTable, rr:tableName,
rr:sqlQuery.
```

```
m2r:sourceType

      rdf:type rdfs:Property, owl:ObjectProperty;

      rdfs:label "Property that defines type of a source
e.g. relational table, OLAP Cube, spreadsheet, etc.";

      rdfs:domain m2r:Source;

      rdfs:range m2r:SourceType;


#Subject Specs

m2r:SubjectMap

      rdf:type rdfs:Class, owl:Class;

      rdfs:subClassOf m2r:TermMap;

      rdfs:label "Class representing mapping of subject of a
triple".


m2r:subjectMap

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TriplesMap;

      rdfs:range m2r:SubjectMap.


m2r:subject

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TriplesMap;

      rdfs:range rdfs:Resource.


#Predicate and Object Specs

m2r:PredicateObjectMap

      rdf:type rdfs:Class, owl:Class;
```

```
      rdfs:label "Class representing mapping of predicate
and object of a triple".


m2r:PredicateMap

      rdf:type rdfs:Class, owl:Class;

      rdfs:subClassOf m2r:TermMap;

      rdfs:label "".


m2r:ObjectMap

      rdf:type rdfs:Class, owl:Class;

      rdfs:subClassOf m2r:TermMap.


m2r:predicateObjectMap

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:TriplesMap;

      rdfs:range m2r:PredicateObjectMap.


m2r:predicateMap

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:PredicateObjectMap;

      rdfs:range m2r:PredicateMap.


m2r:objectMap

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:PredicateObjectMap;

      rdfs:range m2r:ObjectMap.
```

```
m2r:predicate
      rdf:type rdf:Property, owl:ObjectProperty;
      rdfs:domain m2r:PredicateObjectMap;
      rdfs:range rdf:Property.

m2r:object
      rdf:type rdf:Property, owl:ObjectProperty;
      rdfs:domain m2r:PredicateObjectMap;
      rdfs:range rdfs:Resource.

#Term Types
m2r:IRI
      rdf:type rdfs:Class, owl:Class;
      rdfs:label "Class representing IRI term type".

m2r:BlankNode
      rdf:type rdfs:Class, owl:Class;
      rdfs:label "Class representing BlankNode term type".

m2r:Literal
      rdf:type rdfs:Class, owl:Class;
      rdfs:label "Class representing Literal term type".

#-----
#RDF/QB Specific Mappings
```

```
m2r:QBGraphMap

      rdf:type rdfs:Class, owl:Class;

      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing the graph resulted from
mapping of an OLAP Cube to RDF QB.".



m2r:olapConnectionString

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:subPropertyOf m2r:source;

      rdfs:domain m2r:QBGraphMap;

      rdfs:range xsd:string.



m2r:objectClass

      rdf:type rdf:Property, owl:ObjectProperty;

      rdfs:domain m2r:PredicateObjectMap;

      rdfs:range rdfs:Class.



#DataSet

m2r:DataSetMap

      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing mapping of an OLAP Cube
to a qb:DataSet".



m2r:SubjectMapDataSet

      m2r:class qb:DataSet.



m2r:subjectMapDataSet
```

```
      rdfs:subPropertyOf m2r:subjectMap;
      rdfs:domain m2r:DataSetMap;
      rdfs:range m2r:SubjectMapDataSet.

m2r:PropertyMapDSD
      rdfs:subClassOf m2r:PredicateObjectMap;
      m2r:predicate qb:structure;
      m2r:objectClass qb:DataStructureDefinition.

m2r:propertyMapDSD
      rdfs:subPropertyOf m2r:predicateObjectMap;
      rdfs:domain m2r:DataSetMap;
      rdfs:range m2r:PropertyMapDSD.

m2r:PropertyMapSlice
      rdfs:subClassOf m2r:PredicateObjectMap;
      m2r:predicate qb:slice;
      m2r:objectClass qb:Slice.

m2r:propertyMapSlice
      rdfs:subPropertyof m2r:predicateObjectMap;
      rdfs:domain m2r:DataSetMap;
      rdfs:range m2r:PropertyMapSlice.

#DataStructureDefinition
m2r:DataStructureDefinitionMap
```

```
      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing mapping of a
qb:DataStructureDefinition".


m2r:SubjectMapDSD

      m2r:class qb:DataStructureDefinition.


m2r:subjectMapDSD

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:DataStructureDefinitionMap;

      rdfs:range m2r:SubjectMapDSD.


m2r:PropertyMapCompSpec

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate qb:component;

      m2r:objectClass qb:ComponentSpecification.


m2r:propertyMapCompSpec

      rdfs:subPropertyof m2r:predicateObjectMap;

      rdfs:domain m2r:DataStructureDefinitionMap;

      rdfs:range m2r:PropertyMapCompSpec.


m2r:PropertyMapSliceKey

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate qb:sliceKey;

      m2r:objectClass qb:SliceKey.
```

```
m2r:propertyMapSliceKey

      rdfs:subPropertyof m2r:predicateObjectMap;

      rdfs:domain m2r:DataStructureDefinitionMap;

      rdfs:range m2r:PropertyMapSliceKey.


#ComponentSpecification

m2r:ComponentSpecificationMap

      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing mapping of a
qb:ComponentSpecification".


m2r:SubjectMapCompSpec

      m2r:class qb:ComponentSpecification.


m2r:subjectMapCompSpec

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:ComponentSpecificationMap;

      rdfs:range m2r:SubjectMapCompSpec.


m2r:PropertyMapCompProperty

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate qb:componentProperty;

      m2r:objectClass qb:ComponentProperty.


m2r:propertyMapCompProperty
```

```
      rdfs:subPropertyof m2r:predicateObjectMap;
      rdfs:domain m2r:ComponentSpecificationMap;
      rdfs:range m2r:PropertyMapCompProperty.

m2r:PropertyMapDimProperty
      rdfs:subClassOf m2r:PropertyMapCompProperty;
      m2r:predicate qb:dimension;
      m2r:objectClass qb:DimensionProperty.

m2r:propertyMapDimProperty
      rdfs:subPropertyof m2r:propertyMapCompProperty;
      rdfs:range m2r:PropertyMapDimProperty.

m2r:PropertyMapMeasureProperty
      rdfs:subClassOf m2r:PropertyMapCompProperty;
      m2r:predicate qb:measure;
      m2r:objectClass qb:MeasureProperty.

m2r:propertyMapMeasureProperty
      rdfs:subPropertyof m2r:propertyMapCompProperty;
      rdfs:range m2r:PropertyMapMeasureProperty.

m2r:PropertyMapAttributeProperty
      rdfs:subClassOf m2r:PropertyMapCompProperty;
      m2r:predicate qb:attribute;
      m2r:objectClass qb:AttributeProperty.
```

```
m2r:propertyMapAttributeProperty

      rdfs:subPropertyof m2r:propertyMapCompProperty;

      rdfs:range m2r:PropertyMapAttributeProperty.


#DimensionProperty

m2r:DimensionPropertyMap

      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing mapping of a
qb:DimensionProperty".


m2r:SubjectMapDimProperty

      m2r:class qb:DimensionProperty.


m2r:subjectMapDimProperty

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:DimensionPropertyMap;

      rdfs:range m2r:SubjectMapDimProperty.


m2r:PropertyMapCodeList

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate qb:codeList;

      m2r:objectClass qb:ConceptScheme.


m2r:propertyMapCodeList

      rdfs:subPropertyof m2r:predicateObjectMap;
```

```
      rdfs:domain m2r:DimensionPropertyMap;

      rdfs:range m2r:PropertyMapCodeList.


m2r:PropertyMapRangeClass

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate rdfs:range;

      m2r:objectClass rdfs:Class.


m2r:propertyMapRangeClass

      rdfs:subPropertyof m2r:predicateObjectMap;

      rdfs:domain m2r:DimensionPropertyMap;

      rdfs:range m2r:PropertyMapRangeClass.


#MeasureProperty

m2r:MeasurePropertyMap

      rdfs:subClassOf m2r:Map;

      rdfs:label "Class representing mapping of a
qb:MeasureProperty".


m2r:SubjectMapMeasureProperty

      m2r:class qb:MeasureProperty.


m2r:subjectMapDimProperty

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:MeasurePropertyMap;

      rdfs:range m2r:SubjectMapMeasureProperty.
```

```
#ConceptScheme

m2r:ConceptSchemeMap

      rdfs:subClassOf m2r:Map;

      rdfs:label "Class representing mapping of a
skos:ConceptScheme".


m2r:SubjectMapConceptScheme

      m2r:class skos:ConceptScheme.


m2r:subjectMapConceptScheme

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:ConceptSchemeMap;

      rdfs:range m2r:SubjectMapConceptScheme.


m2r:PropertyMapTopConcept

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate skos:hasTopConcept;

      m2r:objectClass skos:Concept.


m2r:propertyMapTopConcept

      rdfs:subPropertyof m2r:predicateObjectMap;

      rdfs:domain m2r:ConceptSchemeMap;

      rdfs:range m2r:PropertyMapTopConcept.


#RangeClass
```

```
#Concept

m2r:ConceptMap

      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing mapping of a
skos:Concept".



m2r:SubjectMapConcept

      m2r:class skos:Concept.



m2r:subjectMapConcept

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:ConceptMap;

      rdfs:range m2r:SubjectMapConcept.



m2r:PropertyMapInScheme

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate skos:inScheme;

      m2r:objectClass skos:ConceptScheme.



m2r:propertyMapInScheme

      rdfs:subPropertyof m2r:predicateObjectMap;

      rdfs:domain m2r:ConceptMap;

      rdfs:range m2r:PropertyMapInScheme.



m2r:PropertyMapBroader

      rdfs:subClassOf m2r:PredicateObjectMap;
```

```
      m2r:predicate skos:broader;

      m2r:objectClass skos:Concept.


m2r:propertyMapBroader

      rdfs:subPropertyof m2r:predicateObjectMap;

      rdfs:domain m2r:ConceptMap;

      rdfs:range m2r:PropertyMapBroader.


#SliceKey

m2r:SliceKeyMap

      rdfs:subClassOf m2r:TriplesMap;

      rdfs:label "Class representing mapping of a
qb:SliceKey". #needs ComSpec property which is already
available.


m2r:SubjectMapSliceKey

      m2r:class qb:Slice.


m2r:subjectMapSliceKey

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:SliceKeyMap;

      rdfs:range m2r:SubjectMapSliceKey.


#Slice

m2r:SliceKeyMap

      rdfs:subClassOf m2r:TriplesMap;
```

```
      rdfs:label "Class representing mapping of a qb:Slice".
#needs DimensionProperties as properties and members as
values that should be done in the actual mapping.


m2r:SubjectMapSlice

      m2r:class qb:Slice.


m2r:subjectMapSlice

      rdfs:subPropertyOf m2r:subjectMap;

      rdfs:domain m2r:SliceMap;

      rdfs:range m2r:SubjectMapSlice.


m2r:PropertyMapSliceStructure

      rdfs:subClassOf m2r:PredicateObjectMap;

      m2r:predicate qb:sliceStructure;

      m2r:objectClass qb:SliceKey.


m2r:propertyMapSliceStructure

      rdfs:subPropertyof m2r:predicateObjectMap;

      rdfs:domain m2r:SliceMap;

      rdfs:range m2r:PropertyMapSliceStructure.
```