Semantic Metadata for Supporting Exploratory OLAP

Varga, Jovan

DOI (link to publication from Publisher):
10.5278/vbn.phd.engsci.00172

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

SEMANTIC METADATA FOR SUPPORTING EXPLORATORY OLAP

BY
JOVAN VARGA

DISSERTATION SUBMITTED 2016

AALBORG UNIVERSITY
DENMARK

Semantic Metadata for Supporting Exploratory OLAP

Ph.D. Dissertation

Jovan Varga

Dissertation submitted November, 2016

A thesis submitted to Universitat Politècnica de Catalunya, BarcelonaTech (UPC) and the Faculty of Engineering and Science at Aalborg University (AAU), in partial fulfillment of the requirements within the scope of the IT4BI-DC programme for the joint Ph.D. degree in computer science. The thesis is not submitted to any other organization at the same time.

Dissertation submitted: November, 2016

PhD supervisor: Assoc. Prof. Oscar Romero
Universitat Politècnica de Catalunya,
BarcelonaTech, Barcelona, Spain

AAU Ph.D. Supervisor: Prof. Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

AAU Ph.D. Co-Supervisor: Assoc. Prof. Christian Thomsen
Aalborg University, Aalborg, Denmark

PhD committee: Associate Professor Peter Dolog (chairman)
Aalborg University, Denmark

Professor Stefano Rizzi
Università di Bologna, Italy

Associate Professor Robert Wrembel
Poznan University of Technology, Poland

UPC PhD Committee: Assoc. Prof. Robert Wrembel
Poznan University of Technology, Poznań, Poland

Prof. Stefano Rizzi
University of Bologna, Bologna, Italy

Prof. Rafael Berlanga
Universitat Jaume I, Castellón, Spain

PhD Series: Faculty of Engineering and Science, Aalborg University
ISSN (online): 2246-1248
ISBN (online): 978-87-7112-840-6

Published by:
Aalborg University Press
Skjernvej 4A, 2nd floor
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2016

Abstract

On-Line Analytical Processing (OLAP) is an approach widely used for data analysis. OLAP is based on the multidimensional (MD) data model where factual data are related to their analytical perspectives called dimensions and together they form an n-dimensional data space referred to as data cube. MD data are typically stored in a data warehouse, which integrates data from in-house data sources, and then analyzed by means of OLAP operations, e.g., sales data can be (dis)aggregated along the location dimension. As OLAP proved to be quite intuitive, it became broadly accepted by non-technical and business users. However, as users still encountered difficulties in their analysis, different approaches focused on providing user assistance. These approaches collect situational metadata about users and their actions and provide suggestions and recommendations that can help users' analysis. However, although extensively exploited and evidently needed, little attention is paid to metadata in this context. Furthermore, new emerging tendencies call for expanding the use of OLAP to consider external data sources and heterogeneous settings. This leads to the Exploratory OLAP approach that especially argues for the use of Semantic Web (SW) technologies to facilitate the description and integration of external sources. With data becoming publicly available on the (Semantic) Web, the number and diversity of non-technical users are also significantly increasing. Thus, the metadata to support their analysis become even more relevant.

This PhD thesis focuses on metadata for supporting Exploratory OLAP. The study explores the kinds of metadata artifacts used for the user assistance purposes and how they are exploited to provide assistance. Based on these findings, the study then aims at providing theoretical and practical means such as models, algorithms, and tools to address the gaps and challenges identified. First, based on a survey of existing user assistance approaches related to OLAP, the thesis proposes the analytical metadata (AM) framework. The framework includes the definition of the assistance process, the AM artifacts that are classified in a taxonomy, and the artifacts organization and related types of processing to support the user assistance. Second, the thesis proposes a semantic metamodel for AM. Hence, Resource Descrip-

tion Framework (RDF) is used to represent the AM artifacts in a flexible and re-usable manner, while the metamodeling abstraction level is used to overcome the heterogeneity of (meta)data models in the Exploratory OLAP context. Third, focusing on the schema as a fundamental metadata artifact for enabling OLAP, the thesis addresses some important challenges on constructing an MD schema on the SW using RDF. It provides the algorithms, method, and tool to construct an MD schema over statistical linked open data sets. Especially, the focus is on enabling that even non-technical users can perform this task. Lastly, the thesis deals with queries as the second most relevant artifact for user assistance. In the spirit of Exploratory OLAP, the thesis proposes an RDF-based model for OLAP queries created by instantiating the previously proposed metamodel. This model supports the sharing and reuse of queries across the SW and facilitates the metadata preparation for the assistance exploitation purposes. Finally, the results of this thesis provide metadata foundations for supporting Exploratory OLAP and advocate for greater attention to the modeling and use of semantics related to metadata.

Resumé

On-Line Analytical Processing (OLAP) er en bredt anvendt tilgang til dataanalyse. OLAP er baseret på den multidimensionelle (MD) datamodel, hvor faktuelle data relateres til analytiske synsvinkler, såkaldte dimensioner. Tilsammen danner de et n-dimensionelt rum af data kaldet en *data cube*. Multidimensionelle data er typisk lagret i et *data warehouse*, der integrerer data fra forskellige interne datakilder, og kan analyseres ved hjælp af OLAP-operationer. For eksempel kan salgsdata disaggregeres langs sted-dimensionen. OLAP har vist sig at være intuitiv at forstå og er blevet taget i brug af ikketekniske og forretningsorienterede brugere. Nye tilgange er siden blevet udviklet i forsøget på at afhjælpe de problemer, som denne slags brugere dog stadig står over for. Disse tilgange indsamler metadata om brugerne og deres handlinger og kommer efterfølgende med forslag og anbefalinger, der kan bidrage til brugernes analyse. På trods af at der er en klar nytteværdi i metadata (givet deres udbredelse), har stadig ikke været meget opmærksomhed på metadata i denne kotekst. Desuden lægger nye fremspirende teknikker nu op til en udvidelse af brugen af OLAP til også at bruge eksterne og uensartede datakilder. Dette har ført til Exploratory OLAP, en tilgang til OLAP, der benytter teknologier fra *Semantic Web* til at understøtte beskrivelse og integration af eksterne kilder. Efterhånden som mere data gøres offentligt tilgængeligt via Semantic Web, kommer flere og mere forskelligartede ikketekniske brugere også til. Derfor er metadata til understøttelsen af deres dataanalyser endnu mere relevant.

Denne ph.d.-afhandling omhandler metadata, der understøtter Exploratory OLAP. Der foretages en undersøgelse af de former for metadata, der benyttes til at hjælpe brugere, og af, hvordan sådanne metadata kan udnyttes. Med grundlag i disse fund søges der løsninger til de identificerede problemer igennem teoretiske såvel som praktiske midler. Det vil sige modeller, algoritmer og værktøjer. På baggrund af en afdækning af eksisterende tilgange til brugerassistance i forbindelse med OLAP præsenteres først rammeværket Analytical Metadata (AM). Det inkluderer definition af assistanceprocessen, en taksonomi over tilhørende artefakter og endelig relaterede processeringsformer til brugerunderstøttelsen. Dernæst præsenteres en seman-

tisk metamodel for AM. Der benyttes *Resource Description Framework* (RDF) til at repræsentere AM-artefakterne på en genbrugelig og fleksibel facon, mens metamodellens abstraktionsniveau har til formål at nedbringe uensartetheden af (meta)data i Exploratory OLAPs kontekst. Så fokuseres der på skemaet som en fundamental metadata-artefakt i OLAP, og afhandlingen tager fat i vigtige udfordringer i forbindelse med konstruktionen af multidimensionelle skemaer i Semantic Web ved brug af RDF. Der præsenteres algoritmer, metoder og redskaber til at konstruere disse skemaer sammenkoblede åbne statistiske datasæt. Der lægges særlig vægt på, at denne proces skal kunne udføres af ikke-tekniske brugere. Til slut tager afhandlingen fat i forespørgsler som anden vigtig artefakt inden for bruger-assistance. I samme ånd som Exploratory OLAP foreslås en RDF-baseret model for OLAP-forespørgsler, hvor førnævnte metamodel benyttes. Modellen understøtter deling og genbrug af forespørgsler over Semantic Web og fordrer klargørelsen af metadata med øje for assistance-relaterede formål. Endelig leder resultaterne af afhandlingen til fundamenterne for metadata i støttet Exploratory OLAP og opfordrer til en øget opmærksomhed på modelleringen og brugen af semantik i forhold til metadata.

Resum

El processament analític en línia (OLAP) és una tècnica ampliament utilitzada per a l'anàlisi de dades. OLAP es basa en el model multidimensional (MD) de dades, on dades factuals es relacionen amb les seves perspectives analítiques, anomenades dimensions, i conjuntament formen un espai de dades n-dimensional anomenat cub de dades. Les dades MD s'emmagatzemen típicament en un data warehouse (magatzem de dades), el qual integra dades de fonts internes, les quals posteriorment s'analitzen mitjançant operacions OLAP, per exemple, dades de vendes poden ser (des)agregades a partir de la dimensió ubicació. Un cop OLAP va ser provat com a intuïtiu, va ser ampliament acceptat tant per usuaris no tècnics com de negoci. Tanmateix, donat que els usuaris encara trobaven dificultats per a realitzar el seu anàlisi, diferents tècniques s'han enfocat en la seva assistència. Aquestes tècniques recullen metadades situacionals sobre els usuaris i les seves accions, i proporcionen suggerències i recomanacions per tal d'ajudar en aquest anàlisi. Tot i ésser extensivament emprades i necessàries, poca atenció s'ha prestat a les metadades en aquest context. A més a més, les noves tendències demanden l'expansió d'ús d'OLAP per tal de considerar fonts de dades externes en escenaris heterogenis. Això ens porta a la tècnica d'OLAP exploratori, la qual es basa en l'ús de tecnologies en la web semàntica (SW) per tal de facilitar la descripció i integració d'aquestes fonts externes. Amb les dades essent públicament disponibles a la web (semàntica), el nombre i diversitat d'usuaris no tècnics també incrementa significativament. Així doncs, les metadades per soportar el seu anàlisi esdevenen més rellevants.

Aquesta tesi doctoral s'enfoca en l'ús de metadades per suportar OLAP exploratori. L'estudi explora els tipus d'artefactes de metadades utilitzats per l'assistència a l'usuari, i com aquests són explotats per proporcionar assistència. Basat en aquestes troballes, l'estudi preté proporcionar mitjans teòrics i pràctics, com models, algorismes i eines, per abordar els reptes identificats. Primerament, basant-se en un estudi de tècniques per assistència a l'usuari en OLAP, la tesi proposa el marc de treball de metadades analítiques (AM). Aquest marc inclou la definició del procés d'assistència, on els artefactes d'AM són classificats en una taxonomia, i l'organització dels artefactes i ti-

pus relacionats de processament pel suport d'assistència a l'usuari. En segon lloc, la tesi proposa un metamodel semàntic per AM. Així doncs, s'utilitza el Resource Description Framework (RDF) per representar els artefactes d'AM d'una forma flexible i reusable, mentre que el nivell d'abstracció de metamodel s'utilitza per superar l'heterogeneitat dels models de (meta)dades en un context d'OLAP exploratori. En tercer lloc, centrant-se en l'esquema com a artefacte fonamental de metadades per a OLAP, la tesi adreça reptes importants en la construcció d'un esquema MD en la SW usant RDF. Proporciona els algorismes, mètodes i eines per construir un esquema MD sobre conjunts de dades estadístics oberts i relacionats. Especialment, el focus rau en permetre que usuaris no tècnics puguin realitzar aquesta tasca. Finalment, la tesi tracta amb consultes com el segon artefacte més rellevant per l'assistència a usuari. En l'esperit d'OLAP exploratori, la tesi proposa un model basat en RDF per consultes OLAP instanciant el metamodel previament proposat. Aquest model suporta el compartiment i reutilització de consultes sobre la SW i facilita la preparació de metadades per l'explotació de l'assistència. Finalment, els resultats d'aquesta tesi proporcionen els fonaments en metadades per suportar l'OLAP exploratori i propugnen la major atenció al model i ús de semàntica relacionada a metadades.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisors Dr. Oscar Romero, Dr. Torben Bach Pedersen, and Dr. Christian Thomsen for their constant guidance, support, patience, time, and energy invested throughout my PhD studies. Individually and as a team, they created an amazing research environment where I was both challenged and encouraged to contribute novel solutions for the research gaps identified within my field of work. Dr. Oscar Romero, my Home University advisor, introduced me to the world of research and helped me shape my research skills from very basic principles to addressing advanced research challenges. His vision, knowledge, energy, brightness, and understanding motivated me to always take an extra step needed to push the boundaries of my research field. Dr. Torben Bach Pedersen, my Host University advisor, also influenced my research path from the beginning and helped me analyze my research goals from a broader perspective. His expertise, experience, ideas, and wisdom gave me unique insights that elevated the quality of my research. Dr. Christian Thomsen, my Host University co-advisor, assured that my research skills are maximally refined. His viewpoint, competence, conciseness, and attention to detail directly complemented my development as a researcher. All my advisors made me strive for excellence and believe that hard work pays off. Along with being exceptional professionals, they are admirable people with excellent interpersonal skills. I truly appreciate the opportunity to have had them as my advisors.

Furthermore, I would like to thank Dr. Alberto Abelló, my Home University doctoral programme coordinator, for his support and direction throughout my PhD studies. His insight on our research group seminars helped me to learn and improve, and his assistance with all the administrative procedures was invaluable. Moreover, I truly appreciate the opportunity to collaborate with Dr. Alejandro A. Vaisman and Dr. Lorena Etcheverry that resulted in two important publications that are a part of my thesis. Our interesting discussions brought new and useful insights, especially about the Semantic Web technologies. Additionally, I would like to thank all of the researchers that I had a chance to discuss with as well as the anonymous reviewers that

provided useful feedback and ideas during the work on my PhD thesis.

Importantly, I also thank my colleagues from the DTIM research group, Universitat Politècnica de Catalunya, Aalborg University, the IT4BI-DC programme, University of Bologna, and everybody I shared the office space with. Everything was easier with such great colleagues on my side.

From my whole heart, I wish to thank and devote this thesis to my family. Their endless moral support, encouragement, and motivation made this work possible. Special appreciation and gratitude goes to my parents that have sacrificed everything for my brothers and me, and invested their lives in making us be the men with true values and right priorities in life. Dad, this one is for you! Mom, one more victory for our team! Thanks to my brothers and their families for keeping my heart full and knowing that they are always on my side, no matter what. Furthermore, my admiration and gratitude goes to my most experienced supporter, my grandmother Ljubica whose life advices and example motivated me to never give up. Thanks to my uncles and their families and special thanks to my uncle Veljko for his support and suggestions that helped me in making decisions.

I would also like to thank my friends who are like my extended family. I am truly thankful to my friend and colleague Petar for always being there. As a true friend, he was there to celebrate accomplishments and provide encouragement during challenging times. I would also like to thank my friends back home and all around the world who encouraged me during my PhD journey.

Finally, I thank and praise my Lord Jesus Christ for giving me this opportunity and helping me along the way, step-by-step, day by day. I am grateful that God put all these amazing people on my path and gave me strength, endurance, persistence, courage, wisdom, and everything I needed on this journey. Glory to Jesus!

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC).

Contents

Chapter 1

Introduction

1 Background and Motivation

Business Intelligence (BI) is an area focusing on tools and techniques for data analysis to support decision-making [83]. Traditionally, data in BI systems come from in-house data sources and are integrated in a data warehouse (DW) [64] that keeps their history and supports their analysis. In a DW, data are conformed to the multidimensional (MD) model [53] where factual data (i.e., *facts*) relate to one or more analytical perspectives (i.e., *dimensions*) comprising the n-dimensional space typically referred to as *data cube*. Dimensions are hierarchically structured enabling that fact *measures* (i.e., numerical data) are (dis)aggregated along hierarchical elements representing different granularity *levels*. The MD model is the foundation for On-Line Analytical Processing (OLAP) [2], an approach where data cubes are navigated (e.g., data granularity is changed) via user-friendly interface of OLAP tools (e.g., to discover the sales value for a certain region over a time period). Hence, OLAP became widely spread and popular among business and non-technical users for data analysis and decision-making [2].

Novel trends in recent years bring challenges to the traditional BI systems and OLAP. The big data phenomena with its 3+ Vs causes the rapid data size growth (i.e., volume), speed of data generation (i.e., velocity), and diversification of data formats (i.e., variety) [50, 110] among others (e.g., veracity [50, 99]). Furthermore, the abundance of new publicly available data (e.g., see the Open Data initiative[1]) requires that BI settings and OLAP include non-controlled, heterogeneous, and volatile data sources in an on-demand manner so that end users can include them in their analysis [1]. For instance, considering average salaries from public statistics together with the internal sales results may help making further decisions. More than just data-related

[1] `http://okfn.org/opendata/`

challenges, novel settings spring up new and diverse (often non-skilled) users on the Web that want to analyze data in a similar fashion (e.g., data enthusiast users - educated but non-technical users [44]). All these and other factors call for novel BI and OLAP solutions as discussed in Ad-hoc and Collaborative BI [20], Self-Service BI [1], Live BI [24], On-Demand BI [31], Open BI [74], Situational BI [71], Exploratory OLAP [4] and other approaches focusing on one or more tasks in this spectrum. Throughout this thesis, these innovative solutions are denoted as *next generation BI systems* and the focus on OLAP in these settings is referred to as *Exploratory OLAP*.

To deal with data variety and external data sources, next generation BI systems turn to the Semantic Web (SW) technology stack [19]. Using the Resource Description Framework (RDF) [28] as the SW backbone to represent data in terms of triples comprising a graph where each triple has semantics (contained in its edge, and one or both nodes definitions) opens new possibilities for addressing data sharing, reuse, and integration on the Web. This is especially promoted with the Linked Data initiative [21,45] defining the principles on how to publish and interlink the data on the SW so that the existing semantics is reused and extended. Furthermore, the semantics enables reasoning and supports automatic data processing. The Linked Data and Open Data initiatives also bring plenty of publicly available data (e.g., European statistics) that are referred to as Linked Open Data. Thus, the SW both provides the technological means and motivates data publishing. Well-known OLAP analysis can facilitate the user analysis of SW data and vice-versa, SW data can give semantics to and/or enrich existing data cubes. In the context of external, non-controlled, and heterogeneous data sources, a special attention is on performing OLAP directly over the SW data by using the MD semantics defined in dedicated RDF vocabularies, namely the RDF Data Cube (QB) [27] and QB4OLAP [33]. Data modeled according to QB4OLAP can be automatically queried with traditional OLAP operations [34] and thus explored in familiar fashion by the broad group of OLAP users. Yet, many challenges remain as we discuss in the sequel.

Although OLAP facilitates user's data analysis, users still encounter difficulties when analyzing data cubes [15]. Thus, different user assistance approaches for query (e.g., [15,38]) and query session (e.g., [10]) recommendations, results visualization (e.g., [18]), and exploration using human language (e.g., [75]) were proposed among others for traditional BI and OLAP. These assistance approaches typically collect situational metadata about the user and/or system and use different user assistance algorithms to process them. The need for user assistance is even more emphasized with the novel tendencies such as Exploratory OLAP [4] which promote that non-expert users should be able to navigate the existing data cubes as well as extend them with external data without the help of technical support colleagues. Hence, Exploratory OLAP demands even more complex and advanced approaches

so that metadata become a first-class citizen decoupled from assistance algorithms. Nevertheless, although evidently needed, the metadata for user assistance are still addressed in an ad-hoc manner so that the identification of relevant metadata artifacts and their systematic modeling, management, and processing are typically neglected.

2 The Metadata Lifecycle

The metadata are the fuel for user assistance algorithms in traditional BI as well as next generation BI and other systems. Thus, the way how the metadata are represented directly influences the assistance possibilities. The metadata can enable that both the metadata and assistance algorithms can be reused among different systems, especially if metadata refer to the same application domain, e.g., OLAP. Nonetheless, there are several phases in the metadata lifecycle that need to be performed to achieve this goal. The phases are illustrated in Figure 1.1 and we further explain each of them.

Phases of the metadata lifecycle:

- *Metadata definition.* The lifecycle starts with the metadata definition phase where all the necessary metadata artifacts need to be identified. The artifacts depend on the application domain and exploitation goals.
- *Metadata modeling.* After defining the metadata artifacts, the next phase is their modeling. It needs to define the level of abstraction, the level of detail, the modeling notation, and related design decisions.
- *Metadata management.* Once the metadata are defined and modeled, the implementation details about which technology to be used for the creation of the metadata repository should be made. This enables systematic metadata management as opposed to ad-hoc means like text files and logs.
- *Metadata population.* Having the metadata repository prepared, the next phase includes the process(es) for its population. The complexity of these techniques can range from simple ones where metadata instances are explicitly defined to very complex processes that monitor the system and/or user to generate the metadata.
- *Metadata exploitation.* Finally, once the metadata repository is (being) populated, they should be exploited to bring benefits, i.e., assist the user. Furthermore, the metadata (e.g., user actions) can be processed to generate additional metadata (e.g., user profiles derived from user actions) that facilitate the final exploitation (i.e., user assistance).

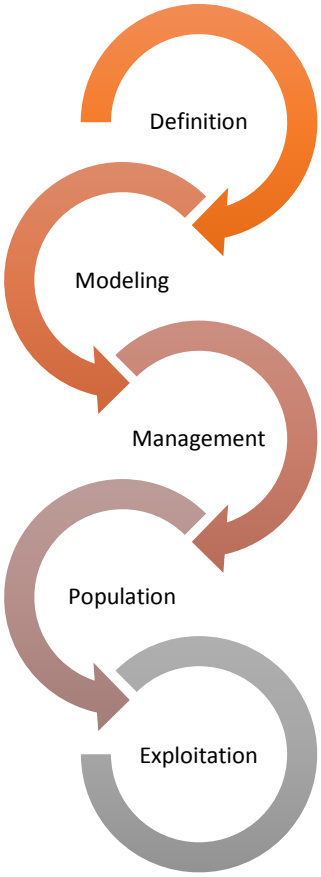

Fig. 1.1: The lifecycle of Analytical Metadata

We next discuss the research questions related to metadata for supporting Exploratory OLAP so that the user can perform her analysis in the easiest way possible. These research questions are considered with respect to the different phases of the metadata lifecycle. We focus on the following research questions that are addressed in this thesis.

What metadata are needed? Existing user assistance approaches in the OLAP context mainly focus on the user assistance algorithms and their results, i.e., the kinds of assistance that the user receives. Little attention is devoted to the metadata and considering of the typical metadata artifacts used in OLAP systems. The resulting pitfall of this situation is that many metadata artifacts which can be used for the user assistance are often overlooked in the OLAP context. For instance, the user demographic information that are typ-

ically used in web recommender systems are rarely considered, although they could be highly beneficial for collaborative recommendations and similar cases. Thus, the identification of all metadata artifacts that can be used for supporting Exploratory OLAP are essential to enable more advance user assistance, yet this remains missing. This research question relates to the metadata definition phase.

How to model the metadata? Even with metadata artifacts defined and the same artifacts used in different systems, their modeling is still being overlooked. For instance, OLAP queries are typically kept in query logs in raw format, i.e., without being modeled at all. The lack of modeling hinders the processing in the same manner as storing of data in a plain text file instead of using a database. In the OLAP context, this drawback can especially be avoided thanks to the OLAP semantics (e.g., the MD schema and OLAP operations) that can be used for their representation. Furthermore, even if using a model, the metadata reuse among different systems can be further facilitated by using modeling formalisms that support sharing (e.g., RDF). This has been shown with the QB4OLAP representation of MD schema, yet poorly (if at all) exploited for other metadata artifacts such as queries. Thus, a *semantic metadata* model is needed for the Exploratory OLAP context. Finally, due to the heterogeneity of models used in next generation BI systems, there is a need for a flexible modeling approach that captures the shared semantics while still enabling flexible modeling for specific systems. This research question relates to the metadata modeling phase.

How to automate the metadata collection and discovery as much as possible? There are two ways to collect metadata. Either the user needs to state them explicitly (e.g., age) or the metadata should be collected as automatically as possible. Thus, the systems need to provide techniques, methods, and algorithms to generate metadata from available (meta)data and user's interaction with the system. Furthermore, the Linked Data settings open a new opportunity for the discovery of additional (meta)data from other sources linked to the current one. Thus, focusing on semantic metadata for supporting Exploratory OLAP brings up new tasks for building the metadata corpus. This research question relates to the metadata population phase.

How to exploit the metadata? The main goal of metadata exploitation for Exploratory OLAP is the user assistance. The semantic metadata for this context can be reused by existing assistance algorithms or used for developing novel ones. The semantic metadata can enable user assistance for OLAP on the SW which has not been much exploited by the existing approaches. This research question relates to the metadata exploitation phase.

Starting from the identified research questions in managing the semantic metadata for Exploratory OLAP lifecycle, this PhD thesis further aims at addressing these research questions and providing methods and tools for facilitating different phases of the lifecycle.

3 Thesis Overview

In this section, we provide an overview of each chapter in the thesis and outline the overall contributions. An overview of how the chapters build on and relate to each other is illustrated in Figure 1.2. Chapter 1 introduces the thesis and Chapter 6 concludes the thesis. Chapters 2 to 5 present the main content and contributions of the thesis. Appendices A and B are published papers that directly relate to and are mostly covered by specific chapters of the main content. The following subsections provide more details about Chapters 2 to 5 and Appendices A and B.

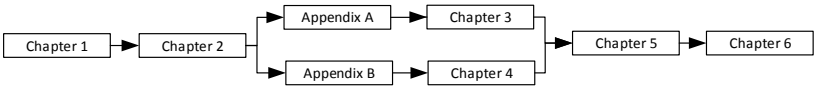

Fig. 1.2: Thesis content overview

3.1 Chapter 2: Towards Next Generation BI Systems: The Analytical Metadata Challenge

As data about data, metadata can refer to very diverse spectrum of concepts. For some concepts it is clear what metadata are, e.g., data describing textual, music, and video files. However, for other context it is not always unequivocal. For instance, user information can be considered as data to be analyzed or metadata describing users that analyze data (e.g., sales), which the system stores. Furthermore, not all metadata are relevant in all contexts. Thus, in order to elaborate on the topic of metadata, the context needs to be set first.

Chapter 2 introduces the context and presents the vision of metadata for user assistance in next generation BI systems. It defines the *analytical metadata framework* describing the metadata artifacts, related processes, and their role for the user assistance. The key idea in the chapter is that the set of metadata artifacts called analytical metadata should be considered and handled as a first-class citizen in *user-centric* BI systems in order to support the user data analysis such as OLAP. Furthermore, as next generation BI systems include external and heterogeneous data sources, analytical metadata should be semantic-aware and machine-processable so that different systems can automatically process them.

The settings envisioned for next generation BI systems are aligned with [1] and background given in Section 1. To identify analytical metadata artifacts, the chapter presents the results of a survey of user assistance approaches in OLAP and database contexts mainly focusing on query recommendation approaches. The approaches are analyzed with respect to an assistance pro-

cess that considers the *input* revealing the metadata artifacts (e.g., queries), the *paradigm* describing the abstraction level at which the artifacts are processed (e.g., query syntax level), the *scope* defining the profiling of artifacts needed for the assistance (e.g., user profiling as correlating of artifacts related to a user), and the assistance *goal* (e.g., query recommendations). The results specify the metadata artifacts used and types of user assistance provided, while outlining the need of considering user characteristics and feature-based paradigm (e.g., MD semantics as feature) for the user assistance. Moreover, the system profiling is identified as opening possibilities other than just user assistance.

With the metadata artifacts identified, this chapter further defines the *analytical metadata taxonomy* by taking an existing business oriented taxonomy [36], extending it with an additional category, and defining the technical interpretation of each of the 5 categories, namely definitional, data quality, navigational, lineage, and ratings. Then, the metadata artifacts, including the new ones to support automation, are classified into the categories. The metadata artifacts are the vocabulary, schema, characteristics, profiling metadata, query, query log, session, traceability metadata, preferences, and statistics. Finally, the framework discusses the processing techniques for analytical metadata with the specific emphasis on the automation possibilities where metadata artifacts are classified into explicit, inferred, and structural, while processing techniques are base processing, derivative processing, and goal-oriented processing. Overall, while the existing DW metadata solutions focus on a specific piece of metadata, e.g., Common Warehouse Metamodel [79] focusing mostly on DW schema, the analytical metadata framework provides a unified view of the metadata artifacts needed for supporting the user analytical tasks such as OLAP. Thus, it sets a base and opens new research challenges on metadata for the user assistance in next generation BI where Exploratory OLAP is one of the main data analysis approaches.

3.2 Chapter 3: Analytical Metadata Modeling for Next Generation BI systems

Once the analytical metadata artifacts are defined, the next task is their modeling. The metadata modeling brings the same benefits as the data modeling does in traditional data management systems versus, for example, keeping data in plain-text files. However, since analytical metadata need to be considered as a first-class citizen, managed in a dedicated repository [62], and represented in a flexible and reusable manner, their modeling requires significant attention. This is where the SW technologies can help and provide means to address this task. In this context, a comprehensive discussion and outlook on how SW technologies can benefit the Exploratory OLAP is given in [4]. In the same spirit, we propose the use of RDF to represent analyti-

cal metadata as it supports sharing, re-usability, and provides the necessary flexibility. Indeed, some metadata models related to certain analytical metadata artifacts already use RDF. For instance, the MD schema for OLAP can be modeled using QB/QB4OLAP vocabularies (see Chapter 4 for more details) while SPARQL queries can be represented with the LSQ data model [88]. Furthermore, the heterogeneity of next generation BI systems brings an additional challenge for analytical metadata modeling. Instead of a fixed universal model, analytical metadata should capture the common semantics while enabling the coexistence of different metadata models. Thus, we propose that the analytical metadata are represented at the metamodel abstraction level. In this direction, RDF also provides support as it can be used for ontological metamodeling (see [14]).

Hence, Chapter 3 proposes SM4AM: A Semantic Metamodel for Analytical Metadata as a metamodel of analytical metadata formalized in RDF. The chapter introduces the running example and the necessary background explaining analytical metadata, RDF, and ontological metamodeling. Then, the metamodel is presented and the metamodel elements are mapped to the analytical metadata artifacts. Each metamodel element is explained using the examples illustrating the metamodel, model, and instance abstraction levels. To support the proper use of the metamodel, the chapter also proposes a method on how it should be instantiated. The benefits of using SM4AM are showed on a use case considering two real-world data sets on the SW. In the use case, the schema and query analytical metadata artifacts are used to reduce the resulting search space when exploring external data sources and enable a metamodel-driven (meta)data exploration. Hence, in this chapter we show the practical benefits of using SM4AM. Our approach aims to harmonize the (meta)modeling as a well-established software engineering practice with the more recent SW settings that support flexible and reusable (meta)data representation, so that analytical metadata can bring maximal value for Exploratory OLAP and next generation BI systems.

3.3 Chapter 4: Dimensional Enrichment of Statistical Linked Open Data

The fundamental metadata artifact for enabling Exploratory OLAP is the *schema*. As discussed in Section 1, the schema needs to conform to the MD model and as the SW technologies are supporting Exploratory OLAP [4], this leads to representing schema as semantic metadata via QB4OLAP vocabulary [33]. QB4OLAP extends the RDF Data Cube (QB) vocabulary [27] with the necessary metadata concepts to be fully compliant with OLAP. Nevertheless, extending an existing QB data set with additional QB4OLAP metadata is a tiresome, labor-intensive, and error-prone process. More than just straight-forward transformation from QB to QB4OLAP semantics, this pro-

cess requires the discovery and/or definition of novel concepts for enriching the QB data set. Thus, the maximal possible automation of this process is required, especially considering that, in the spirit of Exploratory OLAP and next generation BI systems, end users should be able to perform this task on their own.

Chapter 4 presents the details on how to create a QB4OLAP data set starting from a QB one. In this context, it first introduces the necessary prerequisites for understanding QB and presents a running example used throughout the paper. Then, a detailed discussion on the QB limitations related to OLAP and how QB4OLAP remedies them is provided. The discussion outlines the benefits of additional metadata (i.e., schema) constructs that QB4OLAP brings. There is already a significant number of QB data sets on the SW, mostly published as statistical linked open data. Hence, to enable OLAP over these data sets, they should be enriched with QB4OLAP semantics. As this is a cumbersome task, the enrichment needs to be achieved as automatically as possible. Thus, the chapter further elaborates on the automation challenges related to the semantics and data. In this context, two techniques are proposed related to the additional QB4OLAP semantics, one for defining the aggregate functions of the measures and the other for the detection of candidate concepts for the construction of the dimension hierarchies. Then, the chapter presents a method defining the necessary steps for enriching a QB data set with QB4OLAP metadata (i.e., semantics) using the previous techniques for its automation. For the sake of comprehension, the steps are formulated as SPARQL queries. Furthermore, the chapter presents a tool called QB2OLAPem that is implemented for the enrichment process. The tool is used to evaluate our approach in the experiments with 25 users that needed to perform the same enrichment tasks with and without the tool. The results show that the tool facilitates and speeds-up the enrichment process, while in practice guarantees the correctness of the MD (i.e., QB4OLAP) schema produced. Using QB2OLAPem, even the non-technical users can perform the enrichment task and prepare a data set for the analysis with OLAP querying tools that can work over QB4OLAP data sets. As such, the approach presented in Chapter 4 represents a significant step towards Exploratory OLAP. Finally, while the introduced method presents the main enrichment tasks to facilitate the understanding, the chapter's appendix provides a fully formalized methodology using set theory. The methodology specifies the enrichment steps in terms of pre-conditions, post-conditions, and transformations performed that do not depend on the implementation decisions made.

Overall, the approach presented in Chapter 4 addresses the important challenges of MD modeling on the SW related to the schema metadata artifact. Moving towards the vision of fusion cubes [1] and empowered by the SW technologies, the presented approach supports the user in creating cor-

rect MD schemata in a user-friendly manner. Furthermore, it paves the path to having a stack of real-world data sets with their schemata around which other semantic metadata artifacts can be added to extend the support for Exploratory OLAP.

3.4 Chapter 5: SM4MQ: A Semantic Model for Multidimensional Queries

Once data are conforming to an MD schema on the SW, the next analytical metadata artifact to be considered for supporting Exploratory OLAP is the query. MD data are typically navigated via OLAP operations used in MD queries. Currently, most of the efforts in this context are devoted to OLAP algebras, however the query modeling is still overlooked. This gap becomes even more relevant when considering the initiatives like Open Data where plenty of publicly available data are analyzed by very diverse and mostly non-technical users. These users can certainly benefit from query reuse and sharing, and this is hardly achievable without a common query representation. Moreover, MD queries can also become the focus of analysis and, if made publicly available, be used for different exploitation purposes.

Thus, Chapter 5 proposes SM4MQ: A Semantic Model for Multidimensional Queries that is an RDF-based formalization of MD queries. The model is designed around QB4OLAP used for the representation of MD data on the SW. In particular, the model represents the set of most popular OLAP operations used in [34] where related algebraic formalization is also provided. In particular, the included OLAP operations are ROLL-UP, DRILL-DOWN, DICE, and SLICE. We model these operations and related MD queries with RDF to support their sharing and reuse on the SW. In this direction, the pieces of the model representing the OLAP operations include some additional concepts with respect to [34] to further facilitate sharing. For instance, ROLL-UP operation includes both from and to levels, while in principle the from level could be discovered from the sequence of OLAP operations. Furthermore, DICE and SLICE operations are split into two types, one using dimensions and other using measures, to provide atomicity and thus again facilitate sharing. Finally, the model captures the semantics of OLAP operations at the conceptual level as discussed in [34] and [29].

Moreover, the chapter also proposes a method to automate the exploitation of SM4MQ queries by means of SPARQL. The method is then exemplified on a use case to transform queries from SM4MQ to a vector representation. The vector representation can be used to compare queries using typical similarity functions (e.g., cosine similarity). To evaluate our approach, we developed a prototype that automatically transforms queries from SM4MQ to a vector representation and computes the similarities between queries. In this context, we used a set of 15 MD queries related to the chapter's running

example data set. We showed that their transformation to vectors can be automated thanks to SM4MQ while the vectors support their further exploitation. Thus, based on these results, the chapter recapitulates on the necessary user efforts and shows that even non-technical users can perform this task, as opposed to typical settings where system-specific query parsing can hardly be achieved without the support of IT people. Overall, this chapter shows that there are multiple benefits of semantic representation of MD queries for Exploratory OLAP that also opens the new possibilities for their exploitation on the SW.

3.5 Appendices A and B

The content of Appendix A is mostly already covered by Chapter 3 while Appendix B brings slightly more details about the tool proposed in Chapter 4. They are included in the thesis for completeness, as they have been published as separate papers. Appendix A presents the initial version of the SM4AM metamodel that was significantly extended in Chapter 3. Appendix B is a demo paper presenting the tool provided in Chapter 4 combined with other modules for enabling OLAP on statistical linked open data.

4 Structure of the Thesis

The results of this PhD thesis are reported inside the four main chapters of this document (i.e., Chapter 2 - Chapter 5). Each chapter is self-contained, corresponding to an individual research paper, and hence it can be read in isolation. There can be some overlaps of concepts, examples, and texts in the introduction and preliminaries sections of different chapters as they are formulated in relatively similar kind of settings. Due to different timelines in which these works have been done, as well as the research teams in which we have worked, there may also be some discrepancies in the terminology for some of the concepts. However, the terminology and notation used in each of the works have been clearly defined and formalized in each particular chapter.

The papers included in this thesis are listed below. Chapter 2 is based on Paper 1, Chapter 3 is based on Paper 2, Chapter 4 is based on Paper 3, Chapter 5 is based on Paper 4, Appendix A is based on Paper 5, and Appendix B is based on Paper 6.

1. Jovan Varga, Oscar Romero, Torben Bach Pedersen, Christian Thomsen. Towards Next Generation BI Systems: The Analytical Metadata Challenge, DaWaK 2014: 89-101

2. Jovan Varga, Oscar Romero, Torben Bach Pedersen, Christian Thomsen. Analytical Metadata Modeling for Next Generation BI systems. Submitted to a journal.

3. Jovan Varga, Alejandro A. Vaisman, Oscar Romero, Lorena Etcheverry, Torben Bach Pedersen, Christian Thomsen. Dimensional Enrichment of Statistical Linked Open Data. Web Semantics: Science, Services and Agents on the World Wide Web, 2016, vol. 40, p. 22-51

4. Jovan Varga, Ekaterina Dobrokhotova, Oscar Romero, Torben Bach Pedersen, Christian Thomsen. SM4MQ: A Semantic Model for Multidimensional Queries. To be submitted to a conference.

5. Jovan Varga, Oscar Romero, Torben Bach Pedersen, Christian Thomsen. SM4AM: A Semantic Metamodel for Analytical Metadata. DOLAP 2014: 57-66

6. Jovan Varga, Lorenan Etcheverry, Alejandro Vaisman, Oscar Romero, Torben Bach Pedersen, Christian Thomsen. QB2OLAP: Enabling OLAP on Statistical Linked Open Data. ICDE 2016: 1346-1349

Chapter 2

Towards Next Generation BI Systems: The Analytical Metadata Challenge

The paper has been published in the
Proceedings of the 16th International Conference on Data Warehousing and Knowledge Discovery, pp. 89-101 (2014). The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1007/978-3-319-10160-6_9

Abstract

Next generation Business Intelligence (BI) systems require integration of heterogeneous data sources and a strong user-centric orientation. Both needs entail machine-processable metadata to enable automation and allow end users to gain access to relevant data for their decision making processes. Although evidently needed, there is no clear picture about the necessary metadata artifacts, especially considering user support requirements. Therefore, we propose a comprehensive metadata framework to support the user assistance activities and their automation in the context of next

generation BI systems. This framework is based on the findings of a survey of current user-centric approaches mainly focusing on query recommendation assistance. Finally, we discuss the benefits of the framework and present the plans for future work.

1 Introduction

Next generation BI systems (BI 2.0 systems) shift the focus to the user and claim for a strong user-centric orientation. Through automatic user support functionalities, BI 2.0 systems must enable the user to perform data analysis tasks without fully relying on IT professionals designing/maintaining/evolving the system. Ideally, the end user should be as autonomous as possible and the system should replace the designer by providing maximal feedback with minimal efforts. This is even more important if we consider heterogeneous data sources. However, this scenario is yet far from being a reality. A research perspective on this new scenario can be found in [1]. As discussed in the paper, BI 2.0 systems should provide a global unified view of different data sources. To address new requirements, such as dynamic exploration of relevant data sources at the right-time, it is outlined that automatic information exploration and integration is a must. These characteristics raise the need for semantic-aware systems and machine-processable metadata.

Metadata in BI 2.0 systems are needed to support query formulation, relevant source discovery, data integration, data quality, data presentation, user guidance, pattern detection, mappings of business and technical terms, visualization, and any other automatable task that are to be provided by the system. However, current approaches address specific metadata needs in an ad-hoc manner and using customized solutions. A unified global view of the metadata artifacts needed to support the user is yet missing. In this paper we perform a survey to identify what user assistance functionalities should be supported and by means of which metadata artifacts their automation should be enabled. Identifying such metadata is mandatory to enable their systematic gathering, organization, and exploration.

Contributions. We propose a comprehensive metadata framework that supports user assistance activities and their automation in the context of BI 2.0 systems. Specifically, we describe the additional process of the user-system interaction so that the system does not only answer queries but supports the user during the interaction. We identify the main user assistance activities to be supported and the metadata artifacts to be gathered, modeled, and processed to enable the automation of such tasks. These results are based on a survey of specific approaches devoted to provide user assistance on activities like querying and visualization. Finally, we categorize the metadata artifacts to support the user assistance and their processing to enable

automation.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents a survey of user-centric approaches. Then, Section 4 defines the metadata framework according to our findings and finally, Section 5 concludes the paper and provides directions for future work.

2 Related Work

The main challenges of user assistance are highlighted in [62]. This paper outlines the increasing need for system support for user analytical tasks in the settings of fast growing, large-scale, shared-data environments. To assist the user with query completion, query correction, and query recommendations, the authors propose meta querying paradigms for advanced query management and discuss the challenge of query representation and modeling. This challenge motivated our research for metadata capturing queries and other related metadata artifacts for user assistance purposes.

Indeed, as discussed in [36], metadata are important for data warehousing users. The article gives a perspective about the metadata benefits for end users' understanding and usage of data warehouse and BI systems, especially for inexperienced ones. The authors present an end user metadata taxonomy consisting of *definitional*, *data quality*, *navigational*, and *lineage* categories. This is a first attempt to characterize end user metadata in BI systems that we aim at extending for a BI 2.0 context.

With the expansion of the Web, analytical data requirements have exceeded traditional data warehouse settings and now entail the incorporation of new and/or external data sources with unstructured or semi-structured data. In this environment, user assistance and its automation are even a greater need but also a challenge. By analyzing the architectural solutions provided in BI 2.0 systems, in the next paragraphs we discuss that metadata are to be one of the key resources for such task.

Some architectural solutions have been recently presented for BI 2.0 (e.g., [1] and [71]). These systems focus on supporting source discovery, data integration, and user guidance for large and often unstructured data sets. However, they do not provide much details about specific metadata artifacts.

[74] proposes the creation of a knowledge base to support data quality-awareness for the user assistance. This knowledge base is to be represented by means of metadata. Furthermore, in the BI Software-as-a-Service deployment model presented in [31], one of the essential business intelligence services is a meta-data service. It defines the business information to support information exchange and sharing among all other services. For metadata handling, these two approaches refer to a specific metadata framework [79].

Finally, the vision of next-generation visual analytics services is presented

in [76]. The challenges of visualization and data cleaning, data enrichment and data integration naturally match the goals of next generation BI systems. The authors analyze these challenges for structured data sets but they note that unstructured and semi-structured data represent even greater challenges. As discussed, these tasks raise the need for a common formalism. The metadata are to address such requirements.

Overall, BI 2.0 systems focus on end-to-end architectural solutions and typically pay little attention to describe how they deal with metadata. Indeed, most of these systems just mention the crucial role of metadata for the system overall success. As mentioned, [31] and [74] suggest the usage of Common Warehouse Metamodel (CWM) [79]. Nevertheless, CWM is a standard for interchange of warehouse metadata that provides means for describing data warehouse concepts but the support provided is incomplete for the BI 2.0 metadata artifacts discussed in the following sections (which we refer to as analytical metadata). In this line, the Business Intelligence Markup Language (BIML) Framework [70] presents the automation achieved by using metadata for tasks like data integration, but it does not cover the user assistance perspective. Hence, in order to gain insight on the needed artifacts, in the next section we focus on approaches addressing user assistance tasks, such as query recommendation, and describe in more detail their metadata needs and management.

3 A Survey of User-centric Approaches

As mentioned in the previous section, we subsequently discuss the approaches providing user support functionalities (typically, query recommendation), and primarily focusing on the metadata artifacts used and their exploitation. Thus, we focus on *analytical metadata* meant to support the user analytical tasks.

3.1 Methodology

As our work is motivated by [62], the search for relevant references started with it and the papers citing it / cited by it. We iteratively followed the references found looking for relevant approaches (on journals and conferences) and detecting the most relevant authors in this field. This search was complemented by the keyword searches on the main research related engines. Soon, the area of query recommendation proved to be the most related one and we repeated and refined our searches for relevant papers on this topic. During our search, our primary focus was to detect metadata artifacts used for the user assistance. Therefore, we selected those papers proposing concrete solutions (implementations and/or theoretical foundations for the user

assistance) providing enough details about the detection and definition of metadata artifacts. Due to the space limitation, we present a subset of papers representing the whole set of papers found.

3.2 Classification of the Surveyed Approaches

In the typical user-system interaction, the user poses a query, the system processes the query and returns the query result to the user. Throughout this process the user often needs assistance. In the reviewed approaches, we encountered various forms of user support. Figure 2.1 describes the additional process flows triggered to provide such support and refers to the main *metadata artifacts* collected and exploited.

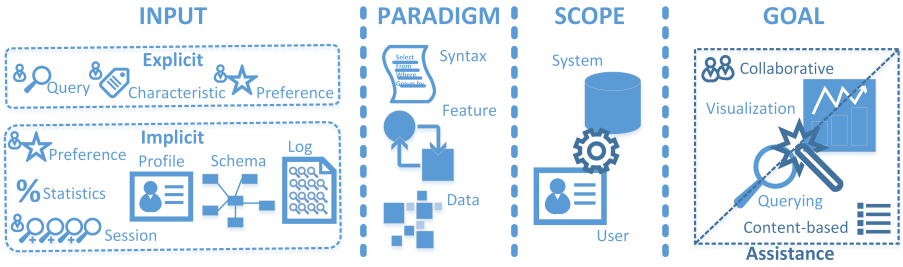

Fig. 2.1: Assistance Process

The system first gathers the needed **input** to be processed in order to achieve the ultimate **goal** of the user support (e.g., query recommendation). We further classify this process according to the level of abstraction or **paradigm** used to process the input data (syntactic, feature-based or data-based approaches) and the process **scope** (profiling the user, the system or both). Table 2.1 shows how the reviewed approaches were classified.

The **input** includes *metadata artifacts* that are defined in Table 2.2. As illustrated in Figure 2.1, we distinguish between *explicit* and *implicit* input. *Explicit* input represents data currently produced by the user that triggers the user assistance process of the system, whereas *implicit* input refers to input that was previously gathered (and stored in the metadata repository), either coming from the system or the user, as well as further metadata inferred from both. *Explicit* input detected in our survey are queries, preferences, and user characteristics. *Implicit* input are previously logged queries, typically stored in the sequence they were posed (i.e., sessions), stored user profiles, automatically inferred user preferences, detected system statistics, and the system schema.

Queries are the main input considered in all surveyed approaches. Processing queries is primarily performed at three abstraction levels or **paradigms**:

Table 2.1: Classification of the reviewed approaches

Approach	Explicit Input	Implicit Input	Paradigm	Scope	Goal	Assistance Techniques
SQL QueRIE Recommendations [9]	Query	Query Log, User Session	Syntax, Data	User Profiling	Querying	Collaborative
Similarity Measures for OLAP Sessions [11]	Query	User Session, Schema	Syntax, Feature (Schema)	/	Querying	Content-based
Predicting Your Next OLAP Query Based on Recent Analytical Sessions [15]	Query	Query Log, Schema	Syntax, Feature (Schema)	System Profiling	Querying	Content-based
A Personalization Framework for OLAP Queries [18]	Query, Preferences	User Profile	Feature (Preferences)	User Profiling	Visualization	Content-based
Query Recommendations for Interactive Database Exploration [26]	Query	Query Log, User Session	Data	User Profiling	Querying	Collaborative
Expressing OLAP Preferences [40]	Query, Preferences	Schema	Feature (Preferences)	User Profiling	Querying	Content-based
myOLAP: An Approach to Express and Evaluate OLAP Preferences [41]	Query, Preferences	Schema	Feature (Preferences)	User Profiling	Querying	Content-based
SnipSuggest: Context-Aware Autocompletion for SQL [63]	Query	Query Log	Syntax	System Profiling	Querying	Content-based
The Meta-Morphing Model Used in TARGIT BI Suite [75]	Query	Statistics, Preferences, Schema	Feature (Preferences, Statistics)	User Profiling	Querying, Visualization	Content-based
"You May Also Like" Results in Relational Databases [94]	Query	Query Log, Schema	Data, Feature (Schema)	User Profiling, System Profiling	Querying	Content-based, Collaborative
Recommending Join Queries via Query Log Analysis [108]	Query	Query Log	Syntax	System Profiling	Querying	Content-based
A Framework for Recommending OLAP Queries [38]	Query	Query Log, User Session	Syntax	System Profiling	Querying	Content-based
Meet Charles, Big Data Query Advisor [92]	Query	Statistics	Feature (SDL, Statistics)	System Profiling	Querying	Content-based

at the *syntax* level, at the *data* level, or modeled according to a certain *feature*. According to [62], *syntactic* processing happens when the syntactic structure of the query is the main facet to be explored (e.g., to combine fragments and propose new queries). For example, [15] presents a framework for recommending OLAP queries based on a probabilistic model of the user's behavior, which is computed by means of query similarities at the syntactic level. *Data-based* processing describes the query in terms of the data it retrieves. For example, in [26] the user queries are characterized by the retrieved tuples. Alternatively, *feature-based* processing models and stores the query in terms of a certain *feature*. *Features* encountered in the reviewed approaches are schema information, user preferences (and visual constraints), statistics, and ad-hoc languages to capture semantic fragments from the queries (e.g., the Segmentation Description Language in [92]). Interesting conclusions can be drawn for each paradigm. *Syntactic* approaches lack semantics and suffer from several drawbacks. First, differently formulated queries returning the same data cannot be identified as equivalent. Second, the natural interconnection be-

Table 2.2: Analytical Metadata Artifacts

Metadata Artifact	Definition	Example
Query	The user inquiry for certain data, disregarding the form it takes	What is the total quantity per product and location?
Preferences	The result set selection and/or representation prioritization	Preferred results of sales where amount is in between 1000 and 5000 range; Preferred representation is a pie chart
User characteristics	Explicitly stated data characterizing the user	Job position, department, office location
Query log	The list of all queries ever posed	{Query 1, query 2, query 3, query 4, ..., query N}
User session	Automatically detected sequence of queries posed by the user when analyzing or searching for certain data	<Query 1, query 3, query 7>
User profile	The set of user characteristics and preferences	Characteristics: User id – '1', job position – 'manager' Preferences: Preferred monthly over quarter overview
Statistics	Automatically detected data usage indicators	Product id 'P' searched in 23% (12345) of cases
System schema	The data model of the system	Dimension Tables: ProductDimension, DateDimension, LocationDimension Fact Tables: SalesFact

tween a series of queries in a single analytical session is lost (or cannot be easily represented). Third, the data granularity produced cannot be detected (as in general, a pure syntactic approach is performed). All in all, exploitation is limited due to the usage of recorded syntactic artifacts only. *Data-based* approaches characterize queries according to the data they retrieve, which entails similar deficiencies due to the lack of semantics gathered. Similar queries returning disjoint results due to some filtering conditions or data aggregation cannot be identified. Also, their interconnection cannot be easily represented. Nevertheless, the main deficiency of this paradigm is its questionable feasibility in the context of BI 2.0 systems, which typically consider Big Data settings with large amounts of data.

None of the two previous paradigms are powerful and flexible enough to fully capture the intention of the user and detect similar queries in a broader sense. This is the main goal behind the last paradigm, which opens new possibilities for addressing this challenge. The concept of a *feature* focuses on modeling the input query to gain additional semantics representing meaningful information that previous paradigms miss. Several current approaches can be classified according to these terms. For example, [11], [15], and [94] represent the queries in terms of the schema. Moreover, [75] proposes the use of the multidimensional model to both model the system schema and the queries posed. For recommendations, it uses recorded user actions and preferences or predefined settings. However, the recommendation potential based on multidimensional semantics such as hierarchical dimension organization is not fully exploited. Finally, other approaches such as [40, 41] introduce the means for the user to explicitly express her preferences when querying the data. However, the user is assumed to manually express the preferences.

The process **scope** defines the profiling need for user assistance purposes. We consider two scope types, *user profiling* that correlates (input) metadata artifacts with the user and *system profiling* that creates a general set of (input) metadata artifacts about the system. While metadata generated in both cases are then used for multiple user assistance purposes, most approaches focus on *user profiling* and few pay attention to *system profiling*, which opens new interesting possibilities, such as self-tuning systems.

The ultimate **goal** and the final step in the assistance process is the concrete user assistance produced. There are multiple forms of user assistance related to the different phases of user-system interaction. We generalize them into two major categories. The first category is *querying* assistance which covers various forms of user support when querying a database. The most typical querying assistance is query recommendation, but other tasks such as query completion, result selection, result recommendations, and join recommendations are identified in our survey. The second major category of user assistance is *visualization*. This assistance refers how to represent the query output in the most satisfactory way for the user. Although undoubtedly crucial for BI 2.0, little attention has been paid to this issue.

An orthogonal aspect to the two categories just discussed are the **assistance techniques** used to provide support. We typically talk about *collaborative* techniques, which entail assistance generation based on the metadata gathered for similar users, and *content-based* techniques, which only exploit the metadata related to a certain user for providing support.

Finally, since many ideas for database recommendation systems come from web solutions we aim at completing our survey by briefly positioning web recommender systems (e.g., see [6]) in terms of our classification. For generating personalized recommendations, web recommendation approaches typically rely on *user profiles*. They process user *queries* at the *data* level, i.e., according to the results retrieved, and profile users and, to some extent, systems. On this base, they provide *content-based*, *collaborative*, or hybrid recommendations. Additionally, as suggested in [6] and thoroughly elaborated in [7], current web recommendation systems should take into account context information for generating context-aware recommendations. In terms of our classification, the context is generally covered by either *user characteristics* and *user preferences*, or by data itself (e.g., in BI, the time/location information are typically covered by appropriate dimensions). Relevantly, web-based recommender systems strongly rely on *user characteristics* (mostly ignored in the database field) and profiles.

As result, by analyzing the described *user assistance process* (Figure 2.1), we identified currently used *metadata artifacts* (Table 2.2), outlined the importance of a *feature-based* approach for gathering and modeling metadata artifacts, remarked that *system profiling* can be used for more than just user assistance and highlighted the importance of *user characteristics* and profiling metadata.

4 The Analytical Metadata Challenge

The survey presented currently used metadata artifacts, their handling, and exploitation potential. Nevertheless, in the context of BI 2.0 systems there is a need for the automation of user assistance activities. Therefore, in this section, we propose a comprehensive framework to address this challenge. First, we present the metadata artifacts to be gathered in the Analytical Metadata (AM) repository and then we discuss how to gather, model, and process them in order to automate their management.

4.1 Analytical Metadata

AM are the set of metadata artifacts entailed by BI 2.0 systems to support the user decision making process. To clarify these artifacts we extend the end user metadata taxonomy from [36] as illustrated in Figure 2.2. The original taxonomy includes *definitional*, *data quality*, *navigational*, and *lineage* metadata categories that are business oriented and do not refer to technical artifacts. We define the technical interpretation of these categories, add a new category, and classify concrete metadata artifacts accordingly. The *definitional* category defines the integration schema, user characteristics, and a vocabulary of business terminology. The *data quality* category describes data set characteristics. The *navigational* category keeps evidence about how the user explores and navigates data. The *lineage* category captures the origin of data including data sources, transformations, and mappings. The *ratings* category covers metadata artifacts about user interests and data usage statistics.

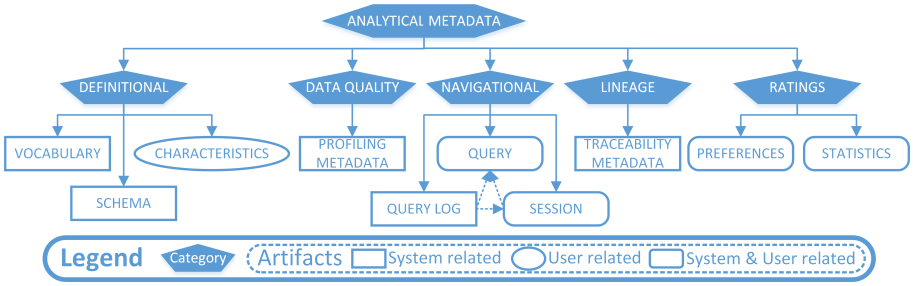

Fig. 2.2: Analytical Metadata Taxonomy

The schema, characteristics, query, session, query log, preferences, and statistics artifacts were introduced in Section 3. In this section, we classify these artifacts and introduce new ones to enhance automation. The metadata artifacts may refer to the system (e.g., schema), the user (e.g., characteristics) or both (e.g., a query defines user interests and refers to the schema structure).

Definitional. The *vocabulary* and *schema* artifacts are fundamental for all the other categories, i.e., all other metadata artifacts should be defined in terms of them. *Vocabulary* defines business terms, their relationships, and their mappings to the integration schema. Its primary role is to act as a reference terminology where to map all gathered metadata artifacts. It can efficiently be represented with an ontology [93] that is machine-processable and enables the automatic reasoning needed for the automation of user support. Next, in the context of BI systems, we propose the *schema* to be represented by means of the multidimensional (MD) model [64]. As discussed in [80], the MD model is mature and well-founded and has key applicability in data warehousing, on-line analytical processing (OLAP), and increasingly in data mining. It captures analytical perspectives by means of facts and dimensions. In this context, it defines necessary constraints and is covered by the MD algebra [84] that together determine potential user actions. For example, if a user analyzes data on a `Month` level, just based on the *schema* she can be suggested to change the granularity to the `Day` or `Year` level even if no one performed this analysis before. Lastly, the *user characteristics* artifact is borrowed from web recommender systems and defines the user by capturing explicitly asserted information that cannot be automatically detected (e.g., job position, age, etc.). It is typically stored as unstructured data and if defined in terms of the *vocabulary* it can be used for metadata processing, e.g., as parameter for recommending algorithms, pattern detection, etc.

Data quality. To tackle *data quality* from a technical point of view, we propose metadata profiling processes (e.g., as explained in [77]) to gain insight into data. *Profiling metadata* characterize data sets like value range, number of values, number of unique values, sparsity, and similar metrics. This way data can be automatically annotated so that domain experts are provided with quality evidences for the data used. For example, inaccurate analytical results might come from data sparsity, i.e., a non-representative data sample.

Navigational. In compliance with the MD representation suggested for the *schema*, we propose the MD model as modeling feature (see previous section for further details) used to capture *queries*, *logs*, and *sessions*. Moreover, to better capture the user intentions, we propose to represent *queries* as ETL flows. As elaborated in [55], a query can be represented as a directed graph of operators. In turn, each operator is characterized as its input and output schema, which should follow the MD principles. This solution is more generic than a typical query definition and enables representation of more complex transformations (e.g., *rollup*), supports lineage, and can be represented as a graph that facilitates manipulations in comparison to declarative queries. Although more powerful, managing such a complex representation is more demanding (e.g., computing similarities or containment between ETL flows) and remains as an open challenge.

Lineage. For *lineage*, we propose the *traceability metadata* artifact that must

capture the information about data sources, transformations performed when migrating data from the sources, and mappings to the integration *schema* (e.g., see [55, 57]). This way the system may provide the user with explanations about how an analytical value is computed and from what sources.

Ratings. The *user preferences* artifact can be manually stated, e.g., using a preference algebra [41]. Although this option might suit advanced users, whenever possible, it is preferable to automatically detect them by appropriate processing techniques over other metadata artifacts. For example, we may infer from the *queries* gathered that the user systematically applies some filtering predicates when navigating the data. Finally, the *statistics* artifact represents data usage indicators that can be considered as a kind of query profiling, i.e., to keep evidence about what data are more explored, as well as more complex indicators (e.g., the most popular combinations of fact and dimension tables [75]).

4.2 Automation and Processing

For the efficient management and storage of the AM we alternatively categorize its artifacts into the categories illustrated in Figure 2.3. These categories elaborate on how to gather each artifact, its level of processing automation, and exploitation purposes. This categorization must be used to guide the AM storing organization.

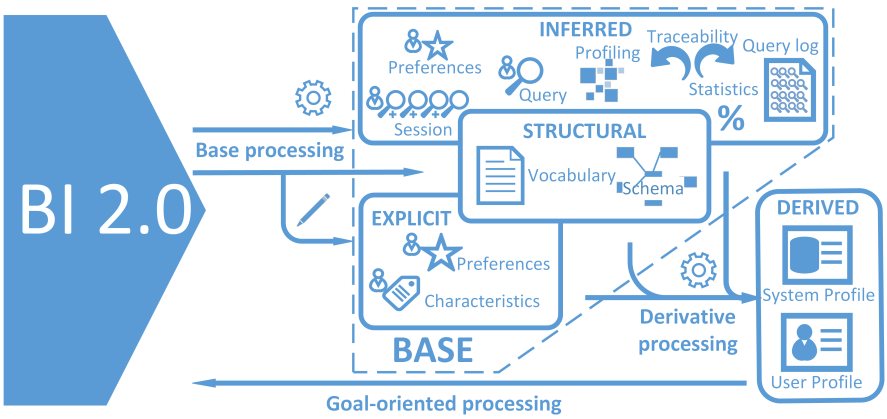

Fig. 2.3: Analytical Metadata Processing Categories

In the categorization, we talk about **structural**, **explicit**, **inferred**, and **derived** metadata. As discussed earlier, the *schema* and *vocabulary* artifacts are fundamental for all the other artifacts and for this reason we refer to them as structural artifacts. Typically, the system designers define and maintain these artifacts. Explicit artifacts (i.e., *user preferences* and *characteristics*) are

those explicitly stated by the user and not automatically detected by the system. Contrary, inferred artifacts are automatically gathered by the system and thus, they can be automatically detected and stored without the explicit help of the user. This category mainly refers to *query logs* and *sessions*, but also to automatically detected *preferences*, data usage *statistics*, as well as the *profiling* and *traceability* metadata artifacts. Inferred artifacts are feature-based, i.e., either modeled according to the MD *schema* or in terms of the definitional *vocabulary*. The structural, explicit, and inferred categories jointly represent the minimal set of information about user actions and interests to be gathered for automating user assistance within the system. For this reason, we refer to these three categories as the **base** metadata, which determine the exploitation possibilities of the AM.

The remaining **derived** metadata category results from processing base metadata according to certain exploitation purposes (typically user and/or system profiling). The produced *user/system profiles* are materialized pieces of derived metadata typically aimed at improving the performance of the algorithms used to provide user assistance. For example, a collaborative recommending system requires to compute similar users. Computing similar users must be performed from base metadata artifacts and it can be previously materialized as derived metadata (i.e., user profiling) in order to improve the response time of the recommending system. Note that system profiling opens new possibilities for system self-tuning capabilities.

Automation implies processing flows (denoted as arrows in Figure 2.3) to populate and exploit the AM artifacts. Our goal here is not to define concrete algorithms but to point out the metadata management and processing flows to be considered when implementing the metadata repository. Consequently, we talk about **base**, **derivative**, and **goal-oriented processing**. The base processing populates the base metadata (i.e., the structural, explicit, and inferred artifacts) by interacting with the BI system. Specifically, the explicit metadata is stated by the user, whereas inferred artifacts are automatically detected and gathered from the system. Structural metadata is typically maintained by the system administrator. Contrary, the derivative processing populates the derived metadata from the base metadata gathered (i.e., derivative processing happens within the AM repository). Finally, goal-oriented processing exploits both the base and derived metadata in order to give concrete user assistance. It represents the purpose of the AM storage and exploitation. The aims of goal-oriented processing are, for example, query recommendations or visualization techniques, whereas final products are recommended queries, graphs/charts, and potentially self-tuning database actions for database optimizations (e.g., data usage *statistics* can serve to trigger indexing of some frequently used attribute).

All in all, due to the expected large volumes of metadata in these systems and the demanding processing capabilities described, the AM reposi-

tory should be implemented in a dedicated subsystem.

5 Conclusions and Future Work

We have presented a comprehensive metadata framework to support user assistance and its automation in the context of BI 2.0 systems. The framework is based on the findings of a survey of existing user-centric approaches where we describe the user assistance process and identify assistance activities and metadata artifacts needed. It proposes an AM repository by categorizing the metadata artifacts to support the user assistance and their processing to enable automation. By introducing the subsets of automatically inferred and derived metadata artifacts with corresponding processing techniques, our framework motivates and directs the automation of the user assistance process which is one of the key requirements of BI 2.0 systems. As metadata artifacts are described on a high abstraction level, the framework is a base to support user assistance features over BI 2.0 heterogeneous data sources. Moreover, since AM also capture the information about the system usage, they can serve for other purposes typically overlooked, such as system self-tuning and optimization.

In our future work, we plan to define the metamodel of AM and provide an implementation of the AM support for query recommendation.

Acknowledgments.

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC) and it has been partly supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2011-24747.

Chapter 3

Analytical Metadata Modeling for Next Generation BI Systems

The paper is submitted to a Journal. This paper is an extended version of our previous workshop paper [103], which is presented in Appendix A.

Abstract

Business Intelligence (BI) systems are extensively used as in-house solutions to support decision-making in organizations. Recent trends in this context advocate for exploiting different metadata artifacts (e.g., schema and queries) to assist the user in data analysis. However, as the metadata management and organization of such artifacts are typically overlooked, the Analytical Metadata (AM) framework has been proposed defining the needed metadata. This framework especially applies in the setting of next generation BI (BI 2.0) systems, which integrate traditional in-house with external sources.

In this paper, we discuss the relevance and benefits of the AM modeling. We propose SM4AM, a Semantic Metamodel for Analytical Metadata that is an RDF-based representation of AM. In these settings, we claim for ontological metamodeling as the proper solution, instead of a fixed universal model, due to the (meta)data models heterogeneity in BI 2.0, while RDF supports sharing and flexible metadata representation. Furthermore, we provide a method to instantiate our metamodel. Finally, we present a use case from the Semantic Web domain and discuss how SM4AM, and the schema and query artifacts, can help traversing different models instantiating our metamodel and enable innovative means to explore external repositories in what we call metamodel-driven (meta)data exploration.

1 Introduction

Traditional BI systems enable data analysis to support decision-making. Data are typically extracted from operational (and controlled) sources, transformed, and loaded into a data warehouse (DW). In the DW, data are usually organized according to a schema conforming to the multidimensional (MD) model [64] and analyzed by user-friendly OLAP front-ends. Following the spirit of OLAP, which enables data analysis for non-technical users, recent trends advocate for the exploitation of schema and other metadata artifacts (e.g., queries) to assist the user when exploring the wealth of data contained in the DW. However, the modeling, organization, and management of these metadata artifacts are typically not systematically addressed and handled in system-specific manners. As a first step to remedy this, the Analytical Metadata (AM) framework [104] has been proposed and, based on a survey of the current state-of-the-art, it defines the user assistance process, the needed metadata artifacts as well as their processing to enable automatic user assistance when exploring and analyzing the DW. The AM framework is especially defined for the context of next generation BI (BI 2.0) systems where external sources need to be incorporated and the use of Semantic Web (SW) technologies is encouraged.

BI 2.0 aims to expand the analysis scope beyond traditional settings where users query prepared data from already known data sources. Publicly available data on the web (e.g., Open data[1]) are to be analyzed in the same fashion (e.g., with a pivot table) as in typical BI settings. These data are explored either *per se* or used to enrich traditional sources. The SW technologies further provide means to represent these data in a machine-readable format to support their automatic processing based on their semantics [4]. In these settings, the number and diversity of users are also constantly growing. Unlike traditional BI systems where data are typically analyzed by trained professionals, new possibilities attract an increasing number of individuals and groups of non-expert users (e.g., [1, 62]). Thus, the AM artifacts also need to be shared/reused and automatically processed to enable user assistance for BI 2.0.

Data and metadata modeling approaches are widely applied in software engineering and database domains to enable systematic data organization and automation. For example, conforming data to the MD model enables automatic data aggregations [53]. Likewise, modeling of AM is necessary and highly desired, especially considering the SW context. Indeed, some recent approaches already model certain AM artifacts. For instance, instead of keeping queries in logs, [88] represents queries according to a query metadata model and stores them in a common repository. However, strict modeling is

[1] `https://okfn.org/opendata/`

hardly applicable for the BI 2.0 context due to a high heterogeneity of models and sources, e.g., relational and graph data models. Thus, in the present paper we present a metadata modeling approach to capture the semantics of AM artifacts in an RDF-based metamodel. We use the metamodel abstraction level due to the models heterogeneity and RDF [28] to support sharing and reuse necessary for BI 2.0 settings. Initiatives such as Linked Data (see [21]) promote that data are published in RDF and interlinked with other sources in order to facilitate data re-usage. We claim that this approach can also be used for AM. Moreover, the RDF Schema vocabulary [23] built on top of RDF enables representation of different *ontological modeling layers* (see [65]). Hence, our approach is *ontological metamodeling* [14] and it includes a method defining steps to instantiate a metamodel with models that likewise have instances. We discuss its applicability and benefits based on a use case for two real-world data sets.

Contributions. Overall, the main contributions of our work are as follows:

- We present a Semantic Metamodel for Analytical Metadata (*SM4AM*), an RDF-based metamodel for AM. The metamodel formalizes both system and user related metadata artifacts needed to enable user assistance in BI 2.0 systems.
- Given the challenge of metamodeling in RDF, we provide a method defining detailed steps on how to use the metamodel for instantiating system-specific metadata models.
- We present a use case from the SW domain with two real-world data sets related to countries that shows the benefits of using *SM4AM* to reduce the (meta)data search space in a metamodel-driven (meta)data exploration.

The present paper is a significant extension of an earlier workshop paper [103]. We extended and simplified the metamodel, added a detailed method for the metamodel instantiation, and presented a use case to show the practical benefits of *SM4AM*.

The rest of the paper is organized as follows. Section 2 explains the necessary prerequisites to understand our approach and presents a running example. Then, the complete metamodel is presented in Section 3. Section 4 defines a method comprising of steps for instantiation of an ontological metamodel and Section 5 elaborates on the application level examples. Finally, related work is discussed in Section 6 while Section 7 concludes the paper.

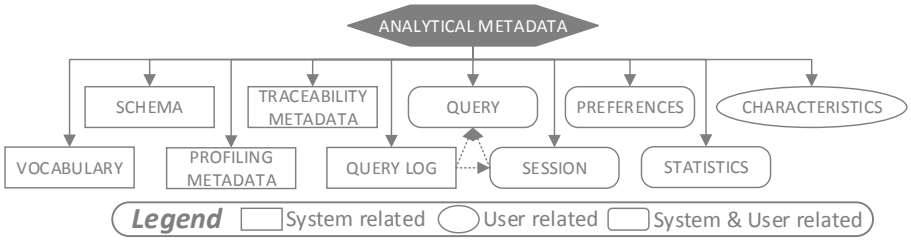

Fig. 3.1: Analytical Metadata Artifacts

2 Background and Running Example

In this section, we introduce the background necessary for understanding of our approach. In particular, we provide details about AM, RDF, and ontological metamodeling. Moreover, we present the running example used throughout the paper.

2.1 Analytical Metadata

The AM framework is presented in [104] where we explain the role of AM for the user assistance in BI 2.0 systems. It includes the AM taxonomy, defined according to a survey, that represents a set of metadata artifacts for this context. Figure 3.1 illustrates the AM artifacts classified into system and/or user related artifacts that are defined as follows. *Vocabulary* defines business terms, their relationships, and their mappings to the integration schema. *Schema* represents the data model (i.e., integration schema) while *profiling metadata* capture technical characteristics of the data set. *Traceability metadata* describe information about data sources, performed transformations, and mappings to the integration schema. Further, *query* represents a user inquiry for certain data (disregarding the form it takes), *query log* is a list of all queries ever posed, and *session* is a sequence of queries posed by the user performing a certain analysis. *Preferences* refer to the user preferences about the result set selection (e.g., the year of analysis) and/or representation prioritization (e.g., visualization chart). Finally, *statistics* captures data usage indicators (e.g., most queried piece of data) while *characteristics* capture the explicitly stated information about the users (e.g., name, job position, etc.).

Due to a high diversity of BI 2.0 systems, the metadata models between systems are heterogeneous and volatile. Thus, there is a need for a common yet flexible solution that enables identification of similar and correlation of related concepts while maintaining a high degree of customization entailed by specific systems. For these reasons, we formalize AM in RDF and, following good habits in software engineering, at the *metamodel* abstraction level. We next explain both choices.

2.2 Resource Description Framework

The means for flexible (meta)data representation range from XML[2] that in principle only tags the (meta)data elements, to semantically rich but computationally complex approaches such as the OWL[3] ontology language. In this spectrum, we follow the principle of least power (see [97]) and our choice is RDF as it provides an acceptable expressivity regardless of its simplicity. RDF is very flexible for capturing data semantics as most information can be naturally represented as RDF triples. A triple consists of a subject, a predicate, and an object, and represents a directed binary relationship (a predicate) between two resources (a subject and an object) or a resource (a subject) and a literal (an object). The subject and predicate are typically represented with IRIs[4] that enable their unique identification, while the object can be an IRI or a literal value. The set of triples is usually grouped into an RDF graph. Furthermore, RDF vocabularies are typically used to define the semantics of IRIs and enable their (re)use across the (Semantic) Web. As discussed in [21], RDF is a standardized data model where data access is simplified as data are self-describing, thus supporting the same concepts reuse in independent systems.

A particularly important vocabulary for modeling in RDF is RDF Schema (RDFS) [23], which is an extension of the RDF vocabulary. Although quite simple, RDF and RDFS (jointly referred as RDF(S)) represent formalisms that can be used for data integration, mappings of business and technical terms, incorporation of external and heterogeneous sources, and other. In the context of metadata, we particularly outline the typing possibilities via the `rdf:type` property. As in an RDF graph both data and metadata with their classes and instances are stored together, the typing is convenient to semantically distinguish the metadata and data instances. Indeed, novel approaches such as [109] use `rdf:type` to reduce the search space and extract the schema from data. Moreover, as types (i.e., classes) are kept together with the instances, the RDF models are extensible to include new types and their instances. RDF[5] graphs can be queried with the SPARQL query language [82]. It applies pattern matching techniques to retrieve sub-graphs (i.e., set of triples) that fit the pattern (i.e., the query). Furthermore, SPARQL supports federated queries[6] for retrieving results from more than one data source. It provides a powerful framework for working with RDF graphs.

All previous features motivate our choice of RDF for the modeling of metadata in a diverse environment such as BI 2.0. Even existing non-RDF

[2] `http://www.w3.org/XML/`

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

[4] `http://en.wikipedia.org/wiki/Internationalized_resource_identifier`

[5] For the sake of simplicity, note that from here on RDF should be read as RDF(S).

[6] `http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/`

metadata repositories can be included if an RDF middleware is created. Furthermore, RDF is widely applied in the Linked Data initiative which is accepted by a significant number of participants including the industry, e.g., Swirrl[7], and public government institutions, e.g., the European Union in the case of Linked Open Data[8]. Linked Data represent a valuable wealth of information as many types of data including geographical, media, government, education, retail and commerce, user generated content and social media, are a part of the Web of Data represented in RDF (see [45]). An overview of Linked Data and RDF can be found in [21]. The user assistance metadata relate to Linked Data in two ways. First, Linked Data sources are typically non-controlled and AM can support user assistance for their analysis. Second, Linked Data interlinking principles can also be applied to correlate the metadata of different systems and thus enhance the user assistance possibilities. For instance, the profiles of the same user in different systems can be identified based on the same user characteristics and used for the personalization of user interaction with all of them. Notice that RDF is already a mean for capturing different types of metadata (e.g., describing music, images, videos, etc.)[9].

2.3 Ontological Metamodeling

As explained in [51] where typical modeling abstraction levels (metamodel, model, and instance levels) are discussed, the metamodel is convenient for the settings where heterogeneous models can be created as instances of a metamodel. Indeed, as discussed in [104], existing BI approaches use similar metadata artifacts for user assistance typically representing them in ad-hoc manners (e.g., logs). Even if metadata models exist, the heterogeneity of BI 2.0 hinders the use of a single model. Thus, our idea is to represent the AM artifacts at the metamodel abstraction level capturing the common semantics. Then, the system-specific metadata models can be defined as instances of the metamodel elements, both when creating new models or enriching the existing ones. The common semantics enables that the metadata elements of different models sharing the semantics (i.e., having the same metamodel type) can be automatically detected via the metamodel. For instance, we can automatically identify query models in different systems and prepare them for the alignment. Once identified, the full alignment can be achieved manually as models are typically small (in comparison to the volume of instances) or in (semi-)automatic ways via entity matching and/or ontology alignment techniques (e.g., [98]). Hence, the metamodeling abstraction level enables the reduction of the metadata search space and facilitates the alignment of the

[7] http://www.swirrl.com/

[8] http://ec.europa.eu/digital-agenda/en/open-data-0

[9] http://dublincore.org/

metadata models. In case of AM that directly extends the user assistance possibilities.

The use of ontologies and SW technologies for metamodeling is already applied for capturing of context information in Web 3.0 as presented in [30] and a detailed survey on the relation between ontologies and metamodels is found in [46]. As we also use RDF in this context, our approach can be considered as *ontological metamodeling* aiming to define meta types for a certain domain (see [14]). Meta types are to be instantiated with types of the specific model, which in turn will have its instances. In the present paper, we use the terms *meta class* for meta type and *class* for type. In RDF, the class-instance relation between a meta class and a class, and of a class with a class instance can be defined with the *rdf:type* property. As discussed in [65], this way we can distinguish between *ontological metamodel layers*. In RDF modeling there is no restriction that an instance cannot be a class at the same time. For example, in Figure 3.2 we can express that `MDLevel` (i.e., a level in an MD schema) is a class and an instance at the same time. In the figure, we use an unnamed namespace for custom concepts.

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2
3 :MDLevel rdf:type :SchemaComponent .
4 :refArea rdf:type :MDLevel .
```

Fig. 3.2: Class and Instance Concept Example

In the context of BI 2.0, an especially important characteristic of RDF models is their extensibility. Novel concepts can easily be incorporated and the metamodel can evolve according to the needs. This mechanism is already used in SW environments and BI 2.0 systems can strongly benefit from it. Furthermore, when having several metadata models related to the metamodel we can more efficiently sample metadata as the search space for the same elements is significantly reduced (see [3]) when starting from the metamodel types. This eliminates the need for classical sampling that is too computationally expansive for the large metadata volumes expected in BI 2.0.

2.4 Running Example

The World Bank provides financial and technical support for developing countries around the world. It publishes data about its projects, indicators about the countries in development, and related information as World Bank Open Data on its website. The data are also available for download via the World Bank API and they were used for the creation of the World Bank Linked Data (WBLD)[10] data set. In the spirit of Linked Data, WBLD is also

[10] `http://worldbank.270a.info/.html`

linked with other data sets (e.g., DBpedia[11]). Throughout the paper we build the examples around the Population (Total)[12] indicator data set from WBLD. This data set contains data about the populations of countries per year. It is modeled according to the RDF Data Cube (QB) vocabulary [27] and can be enriched with additional QB4OLAP concepts to enhance OLAP analysis (see [106]). The schema with QB4OLAP semantics is illustrated in Figure 3.3.

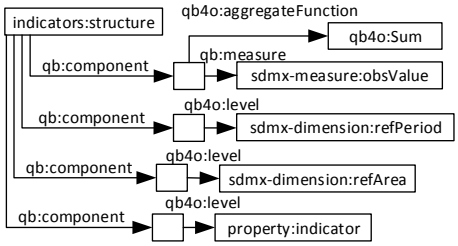

Fig. 3.3: QB4OLAP Schema of the Running Example Data set

For the ease of understanding, the figure only illustrates the elements relevant for the examples. The schema structure is defined by the indicators:structure element. It further relates to the four components (one measure with an aggregate function and three levels), sdmx-measure:obsValue with qb4o:Sum, sdmx-dimension:refPeriod, sdmx-dimension:refArea, and property:indicator. Note that these components are linked with the data set structure via blank nodes (see [28]). This structure and organization originate from the use of QB(4OLAP) for the data representation. Data instances conforming to this schema are created as instances of qb:Observation (see [106] for more details) and an example of population of Serbia for year 2011 is illustrated Figure 3.4.

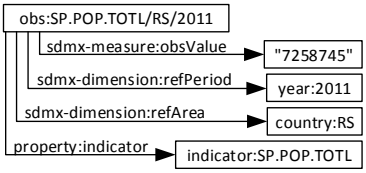

Fig. 3.4: Data Example of the Running Example Data set

3 The *SM4AM* Metamodel

In this section we present *SM4AM*, a metamodel of AM formalized in RDF. We first explain the general design principles of the metamodel followed by

[11] http://wiki.dbpedia.org/

[12] http://worldbank.270a.info/dataset/SP.POP.TOTL.html

the detailed elaboration of the metamodel elements.

3.1 The Metamodel Design

SM4AM formalizes the AM artifacts in a unified metamodel. Our aim is to capture atomic building elements for the artifacts. Thus, the AM artifacts are captured either directly, i.e., by a one-to-one mapping of an artifact to the metamodel element, or indirectly where an artifact is represented with more than one metamodel element. As some artifacts are more coarse grained than others (e.g., session vs. query), we also define complex metamodel elements that organize some of the atomic building elements into structurally organized collections (e.g., a schema organizing schema components). This way, different system-specific metadata models can be created by instantiating atomic elements that can be combined into an instance of a complex element. The complete metamodel is illustrated in Figure 3.5.

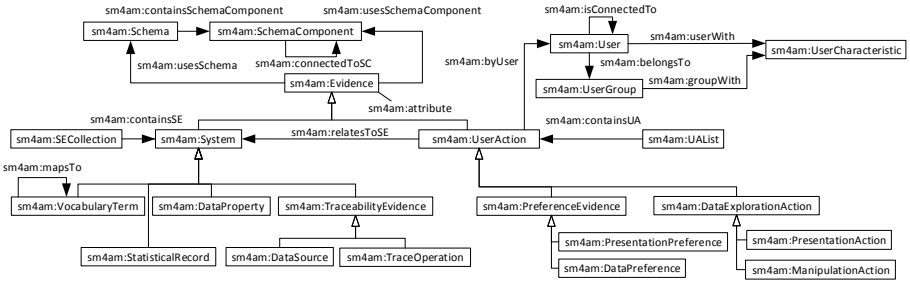

Fig. 3.5: *SM4AM*: A Semantic Metamodel for Analytical Metadata

As explained in Section 2.1 (see also Figure 3.1), AM artifacts are related to the system, user, or both system and user. Therefore, the metamodel is designed around pieces of evidence about the system, user, or both. A piece of evidence is represented with the sm4am:Evidence abstract meta class that is the super class for all pieces of evidence. This abstract meta class is further sub categorized into two (also abstract) meta classes, sm4am:System representing a piece of evidence related to the schema and sm4am:UserAction capturing both elements about explicit user actions over the schema (e.g., queries) and elements inferred from these actions (e.g., user preferences). Each piece of evidence is related to the schema while only sm4am:UserAction related ones also affiliate to the user. The sm4am:UserAction element is linked to sm4am:System via sm4am:relatesToSE (i.e., relates to the system evidence) in order to capture situations when a user action relates to the system element, e.g., a preference over a vocabulary term.

Importantly, each piece of evidence can have attributes (i.e., sm4am:attribute) which is how we model the situation where a property links a class

with a datatype. They are intended for relating different values with a piece of evidence (e.g., a value of a certain statistical record). Concrete properties and datatypes will be defined/specified at the model level. In the next subsections, we explain the remaining metamodel elements starting with the schema and user related elements, and then discussing all the pieces of evidence. The metamodel elements are related to the corresponding AM artifacts (see Section 2.1) to enhance the understanding and semantics of each metamodel element. Table 3.1 summarizes how AM artifacts are covered by *SM4AM*. Note that in the examples we use sm4am namespace for the metamodel elements, while the *@prefix ex: <http://www.example.org/>* prefix and others are used for the model and instance element examples. In the examples, we visually separate the metamodel, model, and instance levels.

Table 3.1: Capturing AM artifacts with *SM4AM* elements

AM Artifact	SM4AM element	System/User related
Vocabulary	sm4am:VocabularyTerm	System
Schema	sm4am:Schema sm4am:SchemaComponent	System
Profiling metadata	sm4am:DataProperty	System
Traceability metadata	sm4am:DataSource sm4am:TraceOperation	System
Query log	sm4am:UAList sm4am:PresentationAction	System
Query	sm4am:ManipulationAction sm4am:UAList	Both
Session	sm4am:UAList	Both
Preferences	sm4am:PresentationPreference sm4am:DataPreference	Both
Statistics	sm4am:StatisticalRecord sm4am:UserCharacteristic	Both
Characteristics	sm4am:User sm4am:UserGroup	User

3.2 Schema and User Related Elements

Schema Related Elements The *schema* AM artifact is modeled with the following two meta classes: sm4am:SchemaComponent represents schema components and sm4am:Schema refers to the schema as a whole organizing the components. Each piece of evidence (i.e., sm4am:Evidence) relates to these meta classes via sm4am:usesSchemaComponent and sm4am:usesSchema properties, respectively. The sm4am:containsSchemaComponent property links the schema (i.e., sm4am:Schema) with schema components (i.e., sm4am:SchemaComponent), while sm4am:connectedToSC interlinks the schema components (i.e., sm4am:SchemaComponent). This design is a generalization of a typical case where a complete integration schema, e.g., a database schema or an RDF graph consists of components that can be mutually connected, e.g., interlinked tables of a relational database or nodes of an RDF graph.

For the running example data set, the schema metadata are represented with the QB4OLAP vocabulary (see Section 2.4). Figure 3.6 illustrates a portion of QB4OLAP elements at the model level instantiating *SM4AM* and the

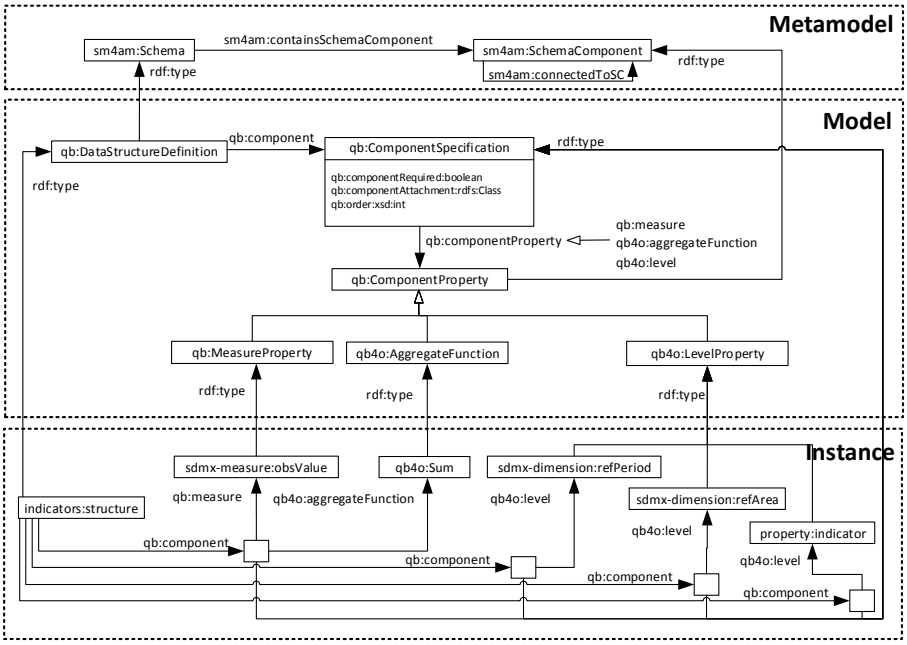

Fig. 3.6: Schema Elements

corresponding portion of the Population data set schema at the instance level. The same way as QB4OLAP is used in the example, other vocabularies and models of different systems representing schema can be related to *SM4AM*.

User Related Elements The (user) *characteristics* AM artifact is modeled with the following meta classes. First, sm4am:UserCharacteristic stands for a specific user characteristic. Second, sm4am:UserGroup models a group of users with the characteristics to which it is linked via sm4am:groupWith. Third, a user is represented with the sm4am:User meta class. She can have several characteristics linked via sm4am:userWith, she can be connected to other users via sm4am:isConnectedTo, and can belong to a user group via sm4am:belongsTo. By now, user characteristics have typically been overlooked in the BI area. Existing approaches mostly focus on the user actions (e.g., queries). Inspired by the web recommender systems we believe that user characteristics are necessary to enable better user assistance possibilities in BI 2.0 [104]. Moreover, different social networks emphasize the need for keeping track of the user interconnections. The BI 2.0 systems need to follow this direction and benefit from these metadata for the user assistance features.

Figure 3.7 exemplifies a simple model for the user metadata and its instances as two journalists exploring the running example data set. In particu-

lar, the model element examples are designed for the members (ex:OrganizationMember instantiating the sm4am:User meta class) of non-profit organizations (ex:NonProfitOrganization instantiating the sm4am:UserGroup meta class) that are interested in exploring the countries' populations. A member has an ID (i.e., ex:ID), a profession (i.e., ex:Profession), and a country of origin (i.e., ex:CountryOfOrigin) as instances of the characteristics (i.e., sm4am:UserCharacteristic). On the other hand, a non-profit organization gathers members that are of certain professions and from certain countries. All model elements are interlinked with the related properties as illustrated in the figure. This model example has as instances two persons (i.e., ex:Person1 and ex:Person2) with their IDs (i.e., ex:ID1 and ex:ID2, respectively) and countries of origin (i.e., ex:Spain and ex:Denmark, respectively), who are both journalists (i.e., ex:Journalist). These persons belong to a European journalist organization (i.e., ex:EuropeanJournalists) that gathers the journalists from Spain and Denmark.

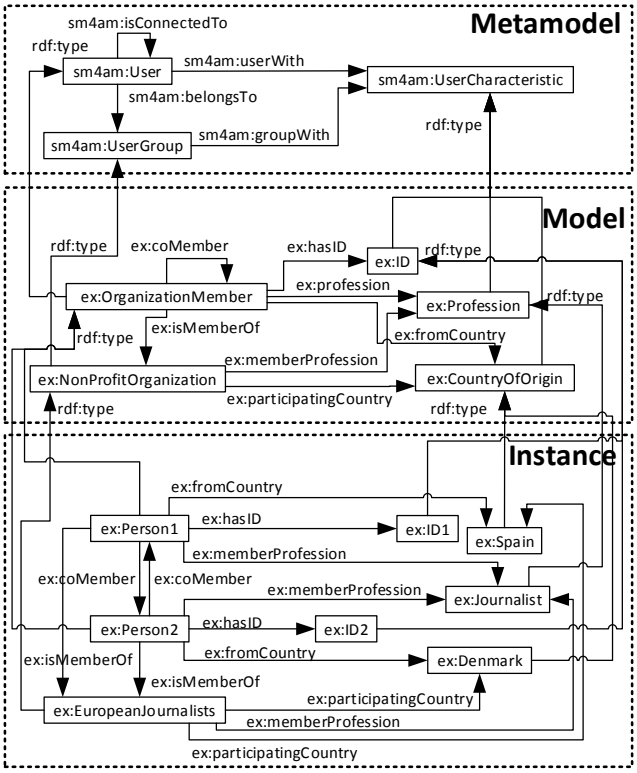

Fig. 3.7: User Elements

3.3 User Action Related Elements

Data Exploration Action Related Elements Several AM artifacts are modeled by sub classes of the sm4am:UserAction meta class as we explain in the sequel. All user action related elements can be related to a user via sm4am:byUser. The *query*, *session*, and *query log* AM artifacts are considered as data exploration actions representing the explicit user actions when analyzing data (e.g., an operation in a query). As these artifacts are of different granularities (e.g., a query log consists of queries) and we focus to capture the atomic elements that can be composed in more complex structures, the sm4am:DataExplorationAction meta class with its subclasses represent the atomic elements that can be organized in a user action list (i.e., sm4am:UAList). The subclasses are sm4am:ManipulationAction capturing the actions for data handling (e.g., change of data granularity) and `sm4am:PresentationAction` describing the actions for data presentation (e.g., a diagram type selection). In general, what is to be considered as atomic elements depends on the model instantiating the metamodel (e.g., an MD operation can be part of an MD query).

Figure 3.8 illustrates the atomic data exploration elements and examples of model instance levels. Note that we include sm4am:SchemaComponent with its instances to make the example more comprehensive. Moreover, for simplicity reasons we do not explicitly show the links from a data exploration action to a schema component and to a user performing the action (at the metamodel level they are sm4am:usesSchemaComponent of the sm4am:Evidence meta class and sm4am:byUser of the sm4am:userAction meta class shown in Figure 3.5). In Figure 3.8, we capture a situation where a journalist querying the Population data set of the running example performs a selection over year 2011 and chooses to visualize the results with a certain chart instance. Therefore, the model level contains ex:SelectionOperation that is an instance of sm4am:ManipulationAction. The ex:SelectionOperation element has the ex:processingTime property linking it with the number of milliseconds taken for its processing that is expressed with xsd:decimal[13]. Moreover, ex:SelectionOperation also has ex:sValue property linking it with the xsd:integer datatype that defines the value for the selection over qb4o:LevelProperty that is linked to ex:SelectionOperation via ex:sLevel. In this case, the ex:processingTime and ex:sValue properties are instances of sm4am:attribute of the metamodel, while ex:sLevel is an instance of sm4am:usesSchemaComponent of the metamodel (see Figure 3.5). More details on how to instantiate attributes and properties are provided in Section 4. The rest of the model demonstrates an example of a sm4am:PresentationAction instance that is ex:ChartSelection with the ex:dwellTime and ex:chartIRI properties that link the class with the number of milliseconds that the user spends on analyzing the chart captured with xsd:decimal, and an IRI defining the chart type as xsd:anyURI. As illustrated in Figure

[13] `http://www.w3.org/TR/rdf11-concepts/`

3.8, the metadata instances of this model are the concrete ex:Selection1 over sdmx-dimension:refPeriod for year 2011 that was processed for 50 milliseconds, and the ex:ChartSelect1 operation instance for a certain (i.e., http://exa...) IRI where the user spent 30000 milliseconds in analyzing the chart.

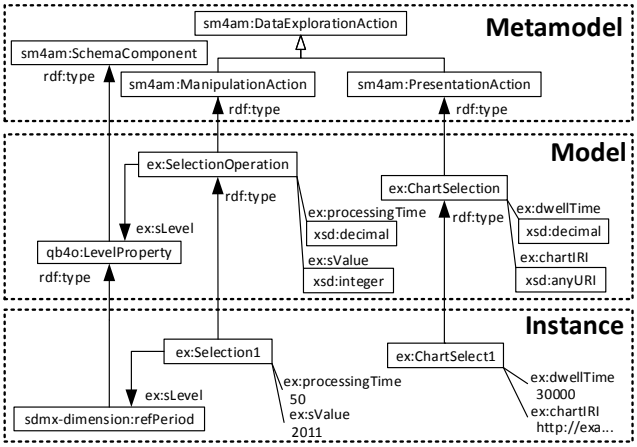

Fig. 3.8: Data Exploration Action Elements

Preference Related Elements The *preferences* AM artifact is modeled with preferences evidence (i.e., sm4am:PreferenceEvidence) category capturing pieces of evidence for the personalization of the user data analysis that can either be stated explicitly, or that can be implied from explicit user actions. They capture the pieces of evidence that enable personalization of the user interaction with the system. We divide them into the following two categories, sm4am:PresentationPreference capturing preferences regarding the data presentation, typically visualization affinities, and sm4am:DataPreference modeling the information about the data interests that can be exploited for the result personalization and similar purposes.

Figure 3.9 depicts the preference related elements of the metamodel with examples of model and instance levels. The same remarks about sm4am:SchemaComponent and sm4am:byUser apply here as in the data exploration example. We capture a situation where a journalist analyzing the Population data set from the running example states her preference for the year 2011 over other years while the system detects her preference (i.e., implicit preferences) for a certain chart type as implied from her previous data exploration. Hence, the model level defines a data ordering preference expression (i.e., ex:DOPreferenceExpression). This expression defines a preference over qb4o:LevelProperty to which it is linked via ex:pLevel similarly to the previous example. It has the ex:inferred, ex:priority, and ex:prefValue properties that link it with

the boolean value true if the preference is inferred, a decimal value defining the priority of the preference important for the ranking of the preferences, and a value that is preferred for the analysis, respectively. All properties are instances of sm4am:attribute as in the previous example. Furthermore, the instance level example illustrates an instance of this preference for the year 2011 over the reference period level which is not inferred and has the priority 5. The figure also includes similar examples for the instantiation of sm4am:PresentationPreference and the corresponding properties.

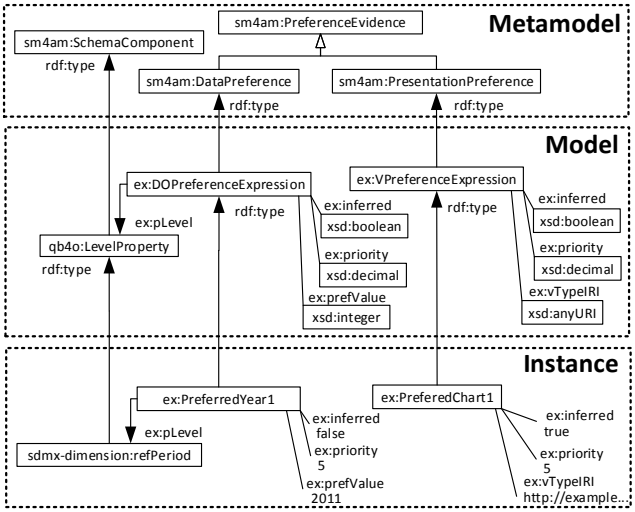

Fig. 3.9: Preference Evidence Elements

User Action List Element After explaining the atomic elements of user actions, we now provide more details about the sm4am:UAList (i.e., user action list) meta class for composing them into ordered lists that represent different concepts. For instance, a *query* can be represented as an ordered list of: i) one or more sm4am:ManipulationActions, ii) optionally one or more sm4am:PreferenceEvidence, and iii) one or more sm4am:PresentationActions. At the model level, the instances of user action elements should instantiate attributes (e.g., the ordering) to determine their organization inside of an instance of user action list. Furthermore, sm4am:UAList can be instantiated (at the model level) to model *sessions*, *query logs*, and *exploration patterns* (e.g., querying patterns) depending on the exploitation needs (e.g., query recommendation). The models depend on the concrete systems.

Figure 3.10 illustrates the use of user action list meta class and examples of model and instance levels. We also use additional meta classes introduced above. We capture a situation where a journalist analyzing the Population

data set from the running example wants to retrieve the population of countries for the year 2011 where result values are formatted with two decimal places. At the model level, this query is defined as ex:Query that is an instance of sm4am:UAList. It illustrates the instances of the previous two user action elements in a complex structure. In this particular case, it includes ex:SelectionOperation, ex:ProjectionOperation, and ex:FormattingExpression with their attributes and the necessary schema elements. Furthermore, the instance level shows an example of the metadata instances for this particular case.

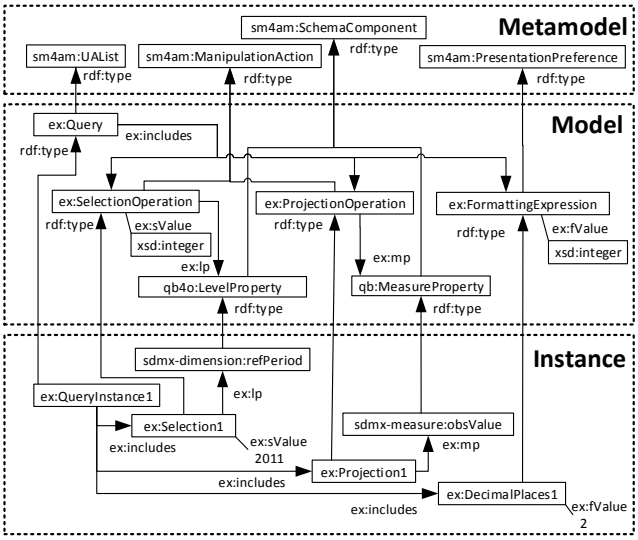

Fig. 3.10: User Action List Element

3.4 System Related Pieces of Evidence

Traceability Related Elements The *traceability metadata* AM artifact is modeled with two subclasses of the sm4am:TraceabilityEvidence meta class that represent atomic metamodel elements. The first one is sm4am:DataSource capturing the source where the data come from. The second one is sm4am:TraceOperation and it represents an operation that can be performed over data or metadata before reaching the data/metadata repository. Note that the use of sm4am:attribute at the model level is useful for indicating if the data source is internal/external and trusted/not-trusted. Moreover, it can be used to link to the particular data values for the integration schema.

Figure 3.11 illustrates the example of the meta classes with the model and instance level examples. We capture DBpedia as a Linked Open Data source

instantiating sm4am:DataSource and the adding of the region level as an operation performed over the Population data set schema from the running example. The ex:LinkedDataSource class instantiates the sm4am:DataSource meta class. For the instantiation of the sm4am:TraceOperation meta class, we use a model example inspired by [57] where integration operations (ex:IntegrationOperation) are modeled as metadata and the insert level (ex:InsertLevel) is one of them. The ex:InsertLevel class has a property ex:levelIRI that links it to the xsd:anyURI datatype. This property is created as an instance of sm4am:attribute as in previous examples. The instance level includes ex:DBPedia as an instance of ex:LinkedDataSource and ex:InsertRegion1 as an instance of ex:InsertLevel that is linked with an IRI representing the region.

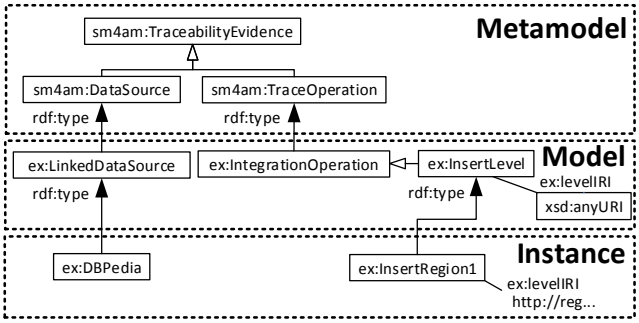

Fig. 3.11: Traceability Evidence Elements

Profiling Related Elements The *profiling metadata* AM artifact is modeled with the sm4am:DataProperty meta class representing technical quality characteristics (e.g., cardinality values). These metadata are typically obtained from data profiling processing in order to enhance the user understanding of the data set. Specific data properties are then defined at the model level depending on the particular system.

Figure 3.12 illustrates the example of model and instance levels for the sm4am:DataProperty meta class. We capture the cardinalities between two levels, namely reference area (i.e., country) and region, in the Population data set schema from the running example. We again include sm4am:SchemaComponent and its model and instance elements to enhance the understanding. The model defines the ex:Cardinality class that relates to the qb4o:LevelProperty class twice, one for the child (i.e., from) and other for the parent (i.e., to) level. The instance level example includes instances of these classes, where ex:CardInst1 has the values 240 on the child (i.e., from) side and 10 on the parent (i.e., to) side, for the pair of sdmx-dimension:refArea and ex:Region levels.

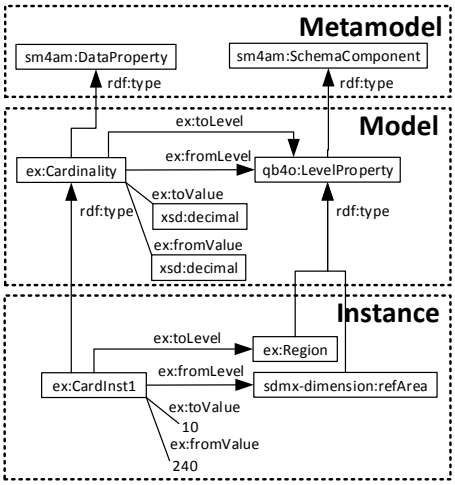

Fig. 3.12: Data Property Element

Vocabulary Related Element The *vocabulary* AM artifact is modeled with the sm4am:VocabularyTerm meta class that represents a vocabulary entry as a building block for vocabulary construction. Moreover, the sm4am:mapsTo property links two vocabulary terms and defines the mapping between them (e.g., a synonym relation). The sm4am:VocabularyTerm meta class is instantiated at the model level with the concrete vocabulary entry types and their links that, in turn, have their instances.

Figure 3.13 depicts examples of model and instance levels of sm4am:VocabularyTerm. We capture a situation where the terms "Place" and "Location" are used as business terms for the sdmx:dimension:refArea schema element of the Population data set schema from the running example. As before, we include the sm4am:SchemaComponent meta class with its instances to make the example more comprehensive. In particular, the model level defines the concept of business term, i.e., ex:BusinessTerm, that can have its synonyms via ex:synonym, and relates to qb4o:LevelProperty via the ex:relatesTo property. Instances of this model are ex:Place and ex:Location that are synonyms and that relate to sdmx:dimension:refArea from the Population data set from the running example.

Statistics Related Element The *statistics* AM artifact is modeled with sm4am:StatisticalRecord as an atomic element for constructing statistics. As before, sm4am:attribute should be instantiated for linking statistical records with their values. Then, the model level should be used to define the class representing type of statistical indicators that are related to the specific numerical datatype as their value, while the instance level keeps track of the indicator instances

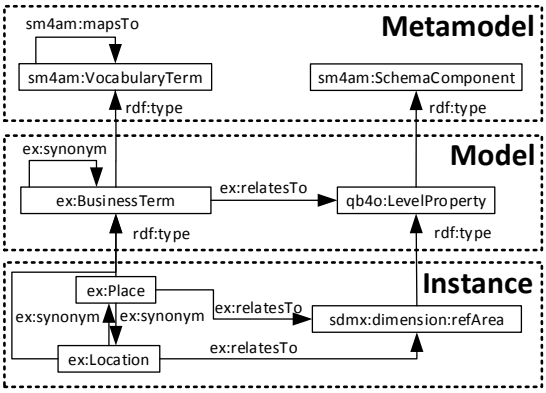

Fig. 3.13: Vocabulary Term Element

and their value. Note that the values for the sm4am:StatisticalRecord related metadata should come from system monitoring.

Figure 3.14 presents the example of model and instance levels for sm4am:StatisticalRecord. We capture the usage counter of the cases where the observed value measure and the referenced area level combination of the Population data set from the running example are used together. As before, we include sm4am:SchemaComponent with its model and instance elements to enhance the understanding. As the model example, the ex:LevelMeasurePair class instantiates sm4am:StatisticalRecord. It is linked to the two instances of sm4am:SchemaComponent, particularly qb4o:LevelProperty and qb4o:MeasureProperty that are the elements of the QB4OLAP schema at the model level. Moreover, the ex:LevelMeasurePair class is linked with xsd:decimal via the ex:value property that is an instance of sm4am:attribute. The instances of this model example are the usage counter (i.e.,ex:UsageCounter1) for the pair of sdmx-measure:obsValue and sdmx:dimension:refArea that has the value 190.

Complex System Related Elements After explaining the atomic system related elements, we provide more details about the sm4am:SEList (i.e., system evidence list) meta class for composing them into ordered lists that represent different concepts. Atomic elements are composed into a complex structure via the sm4am:containsSE property. The attributes (i.e., `sm4am:attribute`) at the model level should be used to determine structural organization (e.g., ordering). For instance, we can have a complex *trace* composed of data sources and traceability operations aligned in an ordered trace structure. Similarly, we can also have a vocabulary composed of vocabulary terms, statistics composed of statistical records, and a data profile composed of data properties.

Figure 3.15 illustrates an example where the elements from Figure 3.11 are aligned in the ordered trace structure. We capture the trace about the

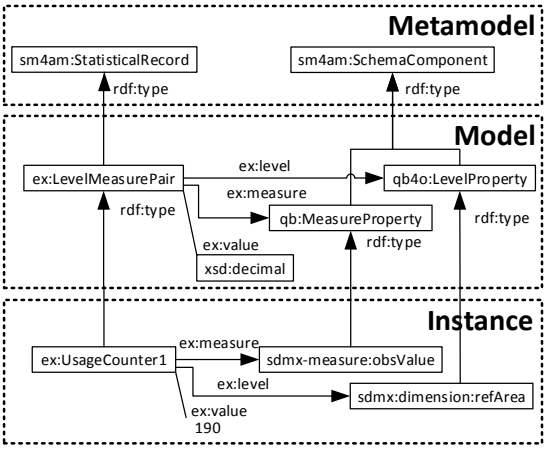

Fig. 3.14: Statistical Record Element

adding of the region level from the World Bank Linked Data source to the Population data set schema from the running example. In particular, the sm4am:SEList meta class is instantiated with the ex:MDIntegration class (also inspired by [57]). This class is then linked with both classes from the previous example, i.e., ex:LinkedDataSource via the ex:source property and ex:IntegrationOperation via the ex:operation property. All these classes are then instantiated in a concrete trace evidence where ex:RegionIntegration describes a trace where ex:DBPedia is a source and ex:InsertRegion1 is the traceability operation performed. The example also includes the same property from the previous example.

4 A Method for Instantiating *SM4AM*

One of the challenges when using an RDF metamodel is to ensure that it is used in a proper and consistent way for the creation of system-specific metadata models. This is a lesson learned from our experience with other RDF-based vocabularies (e.g., [106]) where we noticed that they can be used in inconsistent manners. Thus, the precise steps about how *SM4AM* should be instantiated must be defined, especially considering the context of RDF (meta)modeling. Hence, this method can be used as basis to implement other metamodeling solutions. The ultimate goal is to enable as uniform as possible use of a metamodel and thereby better integration and exploitation possibilities of different models. We first consider the modeling steps after which we provide additional considerations and elaborate on how to extend a metamodel if needed.

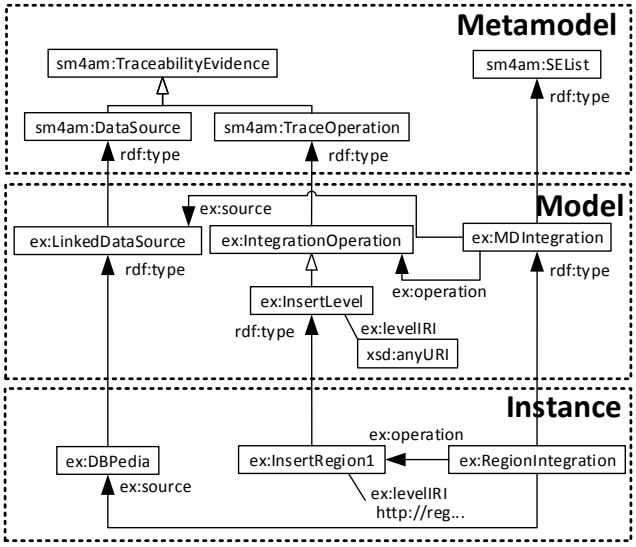

Fig. 3.15: Trace Element

4.1 Modeling Guidelines

In the metamodel we talk about meta classes, properties, and attributes. For creating a model, these elements should be instantiated as follows.

Metamodel class usage guidelines All meta classes in *SM4AM* are defined as instances of rdfs:Class. The definition of traceability related meta classes from Figure 3.15 is shown in Figure 3.16.

```
1 # Metamodel level
2 sm4am:SEList rdf:type rdfs:Class .
3 sm4am:TraceabilityEvidence rdf:type rdfs:Class .
4
5 sm4am:DataSource rdfs:subClassOf sm4am:TraceabilityEvidence .
6 sm4am:TraceOperation rdfs:subClassOf sm4am:TraceabilityEvidence .
```

Fig. 3.16: Example of Meta Class Definitions

For instantiating a meta class at the model level the step is:

- Create a triple ex:ModelClass rdf:type sm4am:MetaClass, where ex:ModelClass is the new class at the model level, sm4am:MetaClass is the *SM4AM* meta class that is to be instantiated, and the rdf:type property links the previous two in an instance-type relation.

The example triples instantiating the previously defined meta classes are illustrated in Figure 3.17.

```
# Model level
ex:MDIntegration rdf:type sm4am:SEList .
ex:LinkedDataSource rdf:type sm4am:DataSource .
ex:IntegrationOperation rdf:type sm4am:TraceOperation .

ex:InsertLevel rdfs:subClassOf ex:IntegrationOperation .
```

Fig. 3.17: Example of Classes Instantiating Meta Classes

Then in turn, model classes have their instances. For instantiating a class at the instance level the step is:

- Create a triple ex:ClassInstance rdf:type ex:ModelClass, where ex:ClassInstance is an instance of ex:ModelClass and the rdf:type property links the previous two in an instance-type relation.

The example triples instantiating the previously defined model classes are illustrated in Figure 3.18.

```
# Instance level
ex:RegionIntegration rdf:type ex:MDIntegration .
ex:DBPedia rdf:type ex:LinkedDataSource .
ex:InsertRegion1 rdf:type ex:InsertLevel .
```

Fig. 3.18: Example of Class Instances

Metamodel property usage guidelines The situation with properties at the metamodel level is a bit more complex. To be consistent with the RDF semantics and enable property typing between metamodel and model levels, each property in *SM4AM* is considered as both rdf:Property and rdfs:Class. This way, at the metamodel level it is used as property to link the meta classes and at the same time it can be instantiated at the model level with a property. Note that examples of similar property formulation can be found in the QB [27] and QB4OLAP [106] vocabularies. For the ease of distinction between properties at the metamodel from the model level, in the rest of this section we refer to a property at the metamodel level as a *meta property*. Furthermore, we refer to a property at the model level only as a *property* and to an instance of a property at the metadata instance level as a *property instance*.

Hence, all meta properties in *SM4AM* are defined as instances of rdf:Property and instances of rdfs:Class. Moreover, as they are properties we also define their domain and range meta classes using the rdfs:domain and rdfs:range properties respectively. The definition of the sm4am:containsSE meta property from Figure 3.5 is shown in Figure 3.19.

As a property should be an instance of a meta property, its domain and range classes (at the model level) are to be instances of the domain and range meta classes of the meta property. For instantiating any of the meta properties at the model level the steps are:

```
# Metamodel level
sm4am:containsSE rdf:type rdfs:Class, rdf:Property;
                 rdfs:domain sm4am:SEList;
                 rdfs:range sm4am:System .
```

Fig. 3.19: Example of Meta Property Definition

- Create a triple ex:modelProperty rdf:type rdf:Property, where ex:modelProperty is the new property at the model that is created, rdf:Property is the concept of concept of property from RDFS, and the rdf:type property links the previous two in an instance-type relation.
- Create a triple ex:modelProperty rdf:type sm4am:MetaProperty, where ex:modelProperty is the new property at the model that is created, sm4am:MetaProperty is any of the meta properties of *SM4AM*, and the rdf:type property links the previous two in an instance-type relation.
- Create a triple ex:modelProperty rdf:domain ex:ModelClass, where ex:modelProperty is the new property at the model that is created, ex:ModelClass is a class at the model level instantiating the domain meta class of the meta property, and the rdfs:domain property links the previous two in a property-domain class relation.
- Create a triple ex:modelProperty rdf:range ex:ModelClass, where ex:modelProperty is the new property at the model that is created, ex:ModelClass is a class at the model level instantiating the range meta class of the meta property, and the rdfs:range property links the previous two in a property-range class relation.

The example triples instantiating the previously defined meta property are illustrated in Figure 3.20.

```
# Model level
ex:source rdf:type rdf:Property, sm4am:containsSE;
                 rdfs:domain ex:MDIntegration;
                 rdfs:range ex:LinkedDataSource .
ex:operation rdf:type rdf:Property, sm4am:containsSE;
                 rdfs:domain ex:MDIntegration;
                 rdfs:range ex:IntegrationOperation .
```

Fig. 3.20: Example of Properties Instantiating a Meta Property

Then in turn, instances of the model properties have their instances. For instantiating any of the properties at the instance level the step is:

- Create a triple ex:ClassInstance1 ex:modelProperty ex:ClassInstance2, where ex:ClassInstance1 is an instance of the model class that corresponds to the property domain, ex:ClassInstance2 is an instance of the model class that corresponds to the property range, and the ex:modelProperty property links the previous two in a relation with the semantics defined by the property at the model level.

The example triples instantiating the previously defined model properties are illustrated in Figure 3.21.

```
# Instance level
ex:RegionIntegration ex:source ex:DBPedia .
ex:RegionIntegration ex:operation ex:InsertRegion1 .
```

Fig. 3.21: Example of Property Instances

Metamodel attribute usage guidelines As briefly explained in Section 3, the concept of *attribute* in *SM4AM* represents a meta property that links a meta class with a data type. The concrete data type should be defined at the model level and, therefore, this meta property defines only the domain while the range remains undefined. The definition of sm4am:attribute is related to the sm4am:Evidence meta class in *SM4AM*. Hence, sm4am:attribute is defined with the triples presented in Figure 3.22.

```
# Metamodel level
sm4am:attribute rdf:type rdfs:Class, rdf:Property;
                rdfs:domain sm4am:Evidence .
```

Fig. 3.22: Metamodel Attribute Definition

When it is instantiated with a property at the model level, the domain of the property should be a class that is an instance of the subclass of the sm4am:Evidence meta class (that is the domain of the attribute at the metamodel level). Furthermore, the range of the property refers to the class of a particular data type. For instantiating an attribute at the model level the steps are:

- Create a triple ex:modelProperty rdf:type rdf:Property, where ex:modelProperty is the new property at the model that is created, rdf:Property is the concept of property from RDFS, and the rdf:type property links the previous two in an instance-type relation.
- Create a triple ex:modelProperty rdf:type sm4am:attribute, where ex:modelProperty is the new property at the model that is created, sm4am:attribute is the attribute concept of *SM4AM*, and the rdf:type property links the previous two in an instance-type relation.
- Create a triple ex:modelProperty rdf:domain ex:ModelClass, where ex:modelProperty is the new property that is created at the model level, ex:ModelClass is a class at the model level instantiating one of the sm4am:Evidence subclasses, and the rdfs:domain property links the previous two in a property-domain class relation.

- Create a triple ex:modelProperty rdf:range ex:DataType, where ex:modelProperty is the new property at the model that is created, ex:DataType is a class defining the data type at the model level (e.g., referring to the one of the RDF-compatible XSD types [28]), and the rdfs:range property links the previous two in a property-range class relation.

The example triples instantiating the previously defined attribute are illustrated in Figure 3.23. Continuing the previous examples, in Figure 3.23 we define the ex:levelIRI attribute for the ex:InsertLevel that has the *xsd:anyURI* datatype (see [28]) and indicates the IRI of a new level that will be added to the schema.

```
1 # Model level
2 ex:levelIRI rdf:type rdf:Property, sm4am:attribute;
3              rdfs:domain ex:InsertLevel;
4              rdfs:range xsd:anyURI .
```

Fig. 3.23: Example of Properties Instantiating an Attribute

Having the model defined, the instance level will contain the class instance linked by the property instance with a related (datatype) value. For instantiating an attribute at the instance level the step is:

- Create a triple ex:ClassInstance1 ex:modelProperty value, where ex:ClassInstance1 is an instance of the model class that corresponds to the property domain, value is a value of the datatype that corresponds to the property range, and the rdf:modelProperty property links the previous two in a relation with the semantics defined by the property at the model level.

The example triples that instantiate the previously defined attribute as the model properties are illustrated in Figure 3.24.

```
1 # Instance level
2 ex:InsertRegion1 ex:levelIRI <http://regionIRIexample...> .
```

Fig. 3.24: Attribute Instantiation

4.2 Additional Considerations

Rather than just instantiating a metamodel with a new model, the use of RDF enables linking of existing models with the metamodel as shown in Section 5. This can be done by revert order of the previous steps. Furthermore, automation of the related tasks is crucial to enable stable populating of the metadata repository which is necessary for the user assistance exploitation tasks. The metadata modeling is a starting point for addressing the automation. Note that even though the automation is desired, in certain cases the user might

still want to state some of these metadata manually, e.g., the expert user can formulate her preferences manually, and if so this should be enabled by the system.

Regarding the metamodel extension, if changes to the metamodel are inevitable, *SM4AM* can be easily extended since it is RDF-based and the metamodel elements are kept together with their instances. The new elements can be added to the metamodel just by creating the new concepts (i.e., nodes and properties) and linking them to the existing ones, i.e., creating new RDF triples. Whenever possible, as a good practice for this context, we strongly recommend that the new elements should be added as subclasses of the abstract metamodel classes (e.g., `sm4am:Evidence`). This will maintain compatibility with the existing approaches while extending *SM4AM*.

In addition to the elements explicitly captured in *SM4AM*, more metadata are contained implicitly. For instance, the statistics about the user actions can be retrieved by counting metadata instances and processing them. We consider this kind of metadata as *derived metadata* and it is up to the specific systems how to exploit this possibility [104].

5 Application Level Use Case

Unlike traditional BI systems that use closed in-house solutions for handling metadata (if any), BI 2.0 settings call for working with external sources and metadata in a reusable and flexible manner. In the sequel, we illustrate how ontological metamodeling of AM can support the reduction of (meta)data search space in a metamodel-driven (meta)data exploration. Thus, we present a use case of using *SM4AM* with two real-world data sets on the SW and their metadata. First, we describe the use case settings. Then, we explain the following benefits of using *SM4AM*: using the schema and query AM artifacts to reduce the metadata search space, and using the query AM artifact to reduce the data search space. Finally, we summarize the overall benefits of *SM4AM* for the use case. The use case shows that *SM4AM* can be used not only with new metadata models but also with already existing ones.

5.1 Use Case Settings and Example Scenario

For the use case settings, we consider two Linked Data sources on the SW. The first data source is WBLD introduced in Section 2.4. It includes data sets whose schemata are modeled according to the QB vocabulary [27]. Note that in contrast to Section 2.4, for the use case we focus directly on the initial QB representation of the data sets and explain its relation to *SM4AM*. A user querying WBLD and analyzing different statistics about countries (e.g., countries population as in the running example) first faces the challenge of

learning how the data are organized, i.e., learning the schemata of the data sets. Furthermore, the user would like to learn more about these countries. The additional information such as countries' description, names, geographical coordinates, are available in DBpedia that is an RDF representation of data published on Wikipedia, the free encyclopedia on the web, and covers very wide range of topics. However, instead of getting all the available information about countries that can be overwhelming, the user is interested in what other users searched (i.e., queried) about countries. A portion of SPARQL queries over DBpedia is available in the LSQ data set [88] and they are modeled according to the LSQ data model. However, the user again faces a challenge of finding only the queries that relate to countries. Here, to facilitate the analysis, the user can reduce the metadata and data search space if using *SM4AM* as illustrated in Figure 3.25. In the figure, the schema metadata of WBLD and the query metadata of LSQ are linked to *SM4AM*. The schema related links enable direct discovery of schema metadata of the WBLD data sets. Furthermore, the query related links, together with DBpedia country IRIs obtained from WBLD, enable discovering only those queries from LSQ, and thereby only those data from DBpedia related to these countries. The details about the benefits achieved this way are discussed in Sections 5.2 to 5.4.

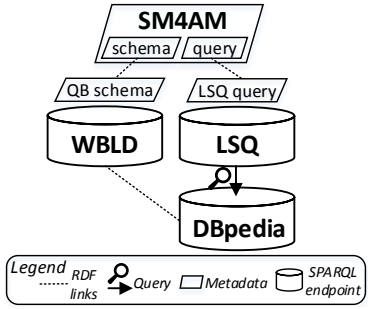

Fig. 3.25: Use Case Settings

Furthermore, Figure 3.26 illustrates the metadata metamodel, model, and instance levels and exemplifies data for the use case settings. For the sake of simplicity, sp:Query with its subclasses can instantiate sm4am:UAList. However, note that more elements of the LSQ data model can be related to several *SM4AM* elements. In particular, sd:Feature and lsqv:JoinVertex with its subclasses can instantiate sm4am:ManipulationAction, lsqv:Execution can instantiate sm4am:StatisticalRecord, and all the properties relating any of the previous classes with xsd datatypes can instantiate sm4am:attribute. The use of rdfs:Resource – the top meta element for all the resources on the SW – in LSQ is an example of use of metamodel and model classes in an RDF model support-

ing our claims. However, rdfs:Resource should not instantiate any *SM4AM* element, and instead the properties linking it with sp:Query and lsqv:Execution indicate that it can be replaced with the instances of sm4am:SchemaComponent in case of lsqv:mentionsObject, lsqv:mentionsPredicate, and lsqv:mentionsSubject, sm4am:Source in case of sd:endpoint, and sm4am:User in case of lsqv:agent. Note that all the prefixes related to LSQ data model elements can be found in [88]. Moreover, data sets in WBLD are modeled according to the QB vocabulary and Figure 3.26 also shows the linking of QB concepts with *SM4AM*. Considering these settings, we next explain how using *SM4AM* for metamodel-driven (meta)data exploration.

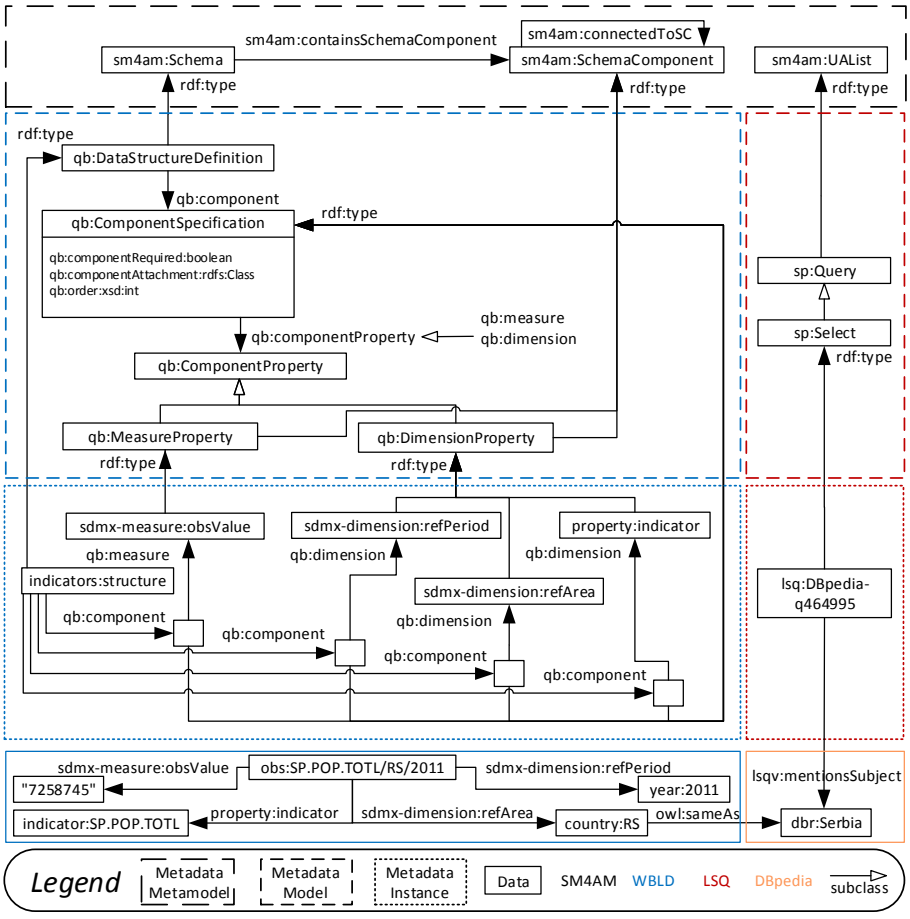

Fig. 3.26: Use Case Internals

5.2 Using Schema to Reduce the Metadata Search Space

To analyze a data set on the SW, the user typically needs to get familiar with the data organization. This is typically done by learning about schema models or ontologies, i.e., instances of the AM schema artifact. However, retrieving only the schema related triples can be a tedious task if the user does not know where to start exploring. For example, Table 3.2 shows the numbers of triples for WBLD.[14]

Table 3.2: WBLD Triple Numbers

Total number of triples	174M
Metadata number of triples	280K
Number of data sets	>9K
Number of schemata	59
Total number of dimensions	81
Total number of measures	70

The total number of triples indicates that non-guided exploration is a burdensome task. Even if focusing on all available metadata (that include non-AM artifacts), the task still requires significant manual efforts as the non-technical user does not know how to discover schema related triples (i.e., the triples related to qb:DataStructureDefinition in WBLD). However, these efforts can be significantly reduced if the schemata have additional semantics linking to *SM4AM* such that schema and schema components can be automatically retrieved. In case of the QB vocabulary used in WBLD, this can be achieved by defining that the schema is an instance of sm4am:Schema and that the dimension and measure are instances of sm4am:SchemaComponent as illustrated in Figure 3.27.

```
1 qb:DataStructureDefinition rdf:type sm4am:Schema .
2 qb:DimensionProperty rdf:type sm4am:SchemaComponent .
3 qb:MeasureProperty rdf:type sm4am:SchemaComponent .
```

Fig. 3.27: Triples Adding SM4AM Semantics to QB

Thus, additional semantics requires the creation of only 3 triples and enables automatic retrieval of schema, dimensions, and measure comprising 210 IRIs – 59 for schemata, 81 for dimensions, and 70 for measures (see Table 3.2). Furthermore, these IRIs can be automatically retrieved with Query 1 retrieving schemata and Query 2 retrieving dimensions and measures. Note that the number of schemata is much smaller than the number of data sets, as different data sets reuse the same schema. This way, the search space is narrowed from 174 millions of total triples (280 thousands of metadata triples) to 210 IRIs related to directly discovered schemata and their components.

[14]The values are rounded and come from WBLD website (http://worldbank.270a.info/about.html) or are retrieved by querying the WBLD SPARQL endpoint.

Moreover, this can be done for any data set which has a schema linked to *SM4AM*. Furthermore, a schema can typically be visualized and explored in user-friendly tools, e.g., the QB schema can be automatically redefined to QB4OLAP and visualized with QB4OLAP explorer[15] as explained in [102].

Query 1
Retrieve All Schemata

```
SELECT DISTINCT ?schema
WHERE {
 ?schema a ?modelLevelSchema .
 ?modelLevelSchema a sm4am:Schema .
}
```

Query 2
Retrieve All Components of Schemata

```
SELECT DISTINCT ?schemaComponent
WHERE {
 ?schemaComponent a ?modelLevelSchemaComponent .
 ?modelLevelSchemaComponent a sm4am:SchemaComponent .
}
```

5.3 Using Query to Reduce the Metadata Search Space

As statistics about countries are the main focus of WBLD, countries represent the main analytical perspective of these statistics. WBLD keeps track of 214 countries and provides their IRIs in both WBLD and DBpedia via owl:sameAs links (identifying the same resources) that can be retrieved with Query 3. Thus, the user can retrieve additional data (i.e., triples) about countries from DBpedia using the DBpedia IRIs. However, this can still be overwhelming as illustrated in Table 3.3 where the total number of country related triples, as well as average, maximum, and minimum number of triples per country are shown for these 214 countries on the DBpedia SPARQL endpoint.

Query 3
Retrieve WBLD Countries

```
SELECT DISTINCT ?c ?cc
WHERE {
 ?c a <http://dbpedia.org/ontology/Country> .
 ?c <http://www.w3.org/2002/07/owl#sameAs> ?cc .
FILTER regex ( str(?cc), 'dbpedia.org')
}
```

Table 3.3: DBpedia Country Related Triple Numbers

#triples	Value
Total	2,358,094
Average	11,019.13
Max	467,819
Min	1

Thus, instead of exploring all the available triples related to a country, the user may be interested in what other users were interested in, i.e., may want

[15] https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/explorer

to reuse some of the other user queries. A portion of queries over DBpedia is available in the LSQ data set. However, the data set includes more than a million queries where approximately 740,000 are over the DBpedia endpoint that can likewise be overwhelming if the user is not familiar with the LSQ data model of queries. Nevertheless, if the LSQ data model is linked to *SM4AM* there are two important benefits. First, the user can reduce the search space for finding the relevant queries. Second, the *SM4AM* semantics supports correlation of schema metadata in WBLD with query metadata in DBpedia (i.e., LSQ). In our use case settings (see Section 5.1), this can be achieved with a single triple stating that sp:Query is an instance of sm4am:UAList (see Figure 3.28).

```
1 sp:Query rdf:type sm4am:UAList .
```

Fig. 3.28: A Triple Adding SM4AM Semantics to LSQ

Query 4 illustrates how *SM4AM* metadata can automatically be exploited in metadata models where query artifact is linked to *SM4AM*. Note that "?parameter?" represents an IRI parameter that is used for the filtering of queries and in our use case settings should be replaced with the DBpedia country IRI.

Query 4
Retrieving Queries Related to an IRI

```
1 SELECT DISTINCT ?query
2 WHERE {
3  ?query ?p ?parameter? .
4  ?query a ?modelLevelQuery .
5  ?modelLevelQuery a sm4am:UAList .
6 }
```

This way, 1908 queries can be retrieved for 214 countries and Figure 3.29 illustrates the number of queries per country. The maximum number of queries per country is 98 for Germany, the average is approximately 9 queries, while there are only 10 countries with no related queries. The top 20 countries with the highest number of queries are presented in Table 3.4. These queries include searches like what are the country description, names, geographical coordinates, language, homepage, images, DBpedia class types, etc. Thus, the use of *SM4AM* and metadata supports reducing of the query search space from the total of 740,000 of queries to approximately 9 queries per country in average.

5.4 Using Query to Reduce the Data Search Space

Furthermore, we next discuss how the query (i.e., metadata) search space reduction supports the reduction of the data search space. Thus, we analyze

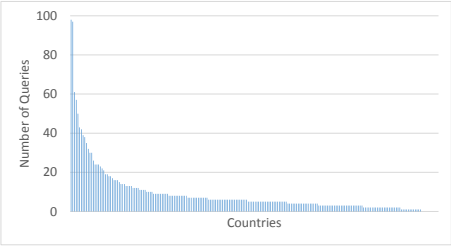

Fig. 3.29: Number of Queries per Country

Table 3.4: Number of Queries per Country

Country IRI	#Queries	Country IRI	#Queries
http://dbpedia.org/resource/Germany	98	http://dbpedia.org/resource/South_Africa	32
http://dbpedia.org/resource/United_States	97	http://dbpedia.org/resource/Canada	30
http://dbpedia.org/resource/Moldova	61	http://dbpedia.org/resource/South_Korea	30
http://dbpedia.org/resource/Italy	57	http://dbpedia.org/resource/Oman	26
http://dbpedia.org/resource/United_Kingdom	50	http://dbpedia.org/resource/Argentina	24
http://dbpedia.org/resource/Greece	43	http://dbpedia.org/resource/Japan	24
http://dbpedia.org/resource/Malta	42	http://dbpedia.org/resource/Philippines	24
http://dbpedia.org/resource/Poland	39	http://dbpedia.org/resource/Spain	23
http://dbpedia.org/resource/France	38	http://dbpedia.org/resource/Netherlands	22
http://dbpedia.org/resource/Hong_Kong	35	http://dbpedia.org/resource/Ireland	21

the result size for the queries previously retrieved. The LSQ data model considers four types of SPARQL queries – SELECT, ASK, DESCRIBE, and CONSTRUCT. Out of the 1908 retrieved queries, 1719 are SELECT, 143 are ASK, and 46 are CONSTRUCT (there are no DESCRIBE queries). CONSTRUCT queries create triples and thus are typically not used for analysis purposes. On the other hand, SELECT and ASK queries are typically used for data analysis but the latter ones only retrieve a boolean value per query. Thus, we further discuss the result sizes of the remaining 1719 SELECT queries. Similar as in the LSQ data model, by the result size we consider the number of results for the SELECT clause of the query and Figure 3.30 illustrates the percentage of queries for different result size ranges.

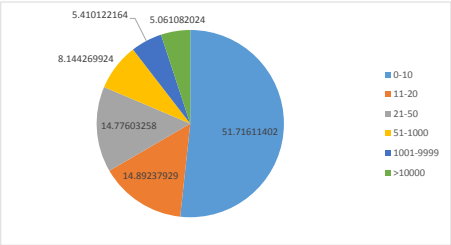

Fig. 3.30: Percentage of Countries per Result Size Range

The experiment results show that half of the queries return 10 or less

results, while there are approximately 15% of the queries both in case of 11 to 20 results as well as 21 to 50 results. Thus, 80% of the queries retrieve 50 or less results and only 5% retrieve 10,000 or more results (note that 10,000 results per query is the default limit on DBpedia SPARQL endpoint). The results illustrate that in the great majority of cases the volume of query results are small and can easily be manually analyzed by the user. Thus, more than just reducing metadata search space, the use of *SM4AM* in this case also reduces the data search space.

For instance, a user who runs Query 5 to search for the population of Serbia over years in the running example data set, can use the DBpedia IRI for Serbia (i.e., http://dbpedia.org/resource/Serbia) available in WBLD (via owl:sameAs and retrieve 9 queries related to Serbia from LSQ (see Query 4). The predicates used in these queries and their result size are presented in Table 3.5.

Query 5
Retrieve Population of Serbia over Years

```
Select ?year ?population
Where {
 ?o a qb:Observation .
 ?o qb:dataSet <http://worldbank.270a.info/dataset/SP.POP.TOTL> .
 ?o sdmx-dimension:refArea <http://worldbank.270a.info/classification/country/RS> .
 ?o sdmx-dimension:refPeriod ?year .
 ?o sdmx-measure:obsValue ?population .
}
```

Table 3.5: LSQ Queries with their Predicates and Result Size for Serbia

Query IRI	Predicate(s)	#Results
http://lsq.aksw.org/res/DBpedia-q464995	http://dbpedia.org/ontology/abstract	1 (1 predicate)
http://lsq.aksw.org/res/DBpedia-q560256	http://dbpedia.org/property/officialLanguages rdfs:comment	1 (1 predicate)
http://lsq.aksw.org/res/DBpedia-q628022	foaf:depiction foaf:homepage	3 (3 predicates)
http://lsq.aksw.org/res/DBpedia-q1054262	rdf:type	34 (1 predicate)
http://lsq.aksw.org/res/DBpedia-q588873	rdf:type	34 (1 predicate)
http://lsq.aksw.org/res/DBpedia-q1034866	rdfs:label rdfs:comment	12 (1 predicate)
http://lsq.aksw.org/res/DBpedia-q250234	foaf:depiction foaf:homepage rdfs:label	3 (3 predicates)
http://lsq.aksw.org/res/DBpedia-q964249	wgs84_pos:lat wgs84_pos:long	12 (3 predicates)
http://lsq.aksw.org/res/DBpedia-q1073992	http://dbpedia.org/property/redirect	0

5.5 Use Case Summary

Overall, the use case shows that even basic use of *SM4AM* facilitates and automates the search for metadata concepts in the data set. Consequently, it further supports both metadata and data exploration by means of narrowing the scope of metadata and data that user needs to analyze (i.e., query). A summary of the optimizations in each step is shown in Figure 3.31. Finally, *SM4AM* provides semantics that can be used for combined use of metadata

artifacts from different data sets. The use case illustrates the benefits of simple *SM4AM* use with existing metadata models for different metadata artifacts and greater gains could be achieved in several ways. For example, the queries with empty or excessive results can be removed in pre-processing and even query recommendations algorithms can be applied. Moreover, *SM4AM* can be used to identify the same metadata artifacts represented with different metadata models that can then be aligned manually or by applying ontology matching and/or entity resolution techniques.

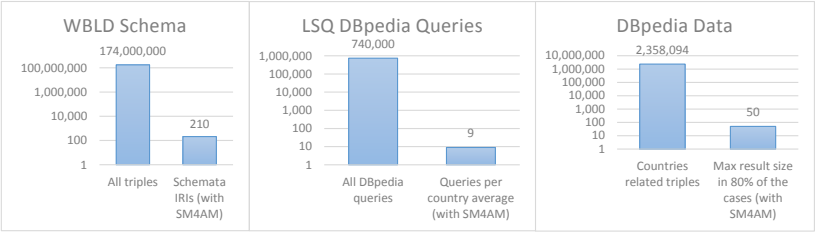

Fig. 3.31: Search Space Reductions with *SM4AM*

6 Related Work

The AM artifacts captured in *SM4AM* are only partially covered in the existing metadata solutions related to the BI area. Therefore, we discuss below a number of representative approaches in terms of *query*, *query log*, *(user) session*, *(user) preferences*, *schema*, *statistics*, *traceability*, *vocabulary*, *user characteristics*, and *profiling* metadata that are the AM artifacts explained in Section 2.1. Furthermore, we consider if any of the solutions are RDF-based like *SM4AM*. Note that more detail on AM, RDF, and ontological metamodeling are already discussed in Section 2.

As BI 2.0 solutions do not elaborate on the concrete metadata artifacts needed for the user assistance, some more insights on this matter can be found in the existing goal oriented (e.g., query recommendation) approaches providing such support in the classical database and BI settings. Moreover, we consider the metadata modeling solutions in these traditional settings. Therefore, we discuss three groups of approaches and discuss representative papers of each group.

The first group are *classical database user assistance* approaches. For instance, [63] and [26] that assist the user in the exploration of very large data sets via SQL. Typically, these approaches use *queries*, *query logs*, and *user sessions*. They analyze the queries based on their syntax and/or result data, compare query sessions in the search for the next potential query, and explore query logs to discover queries and sessions relevant for the recommen-

dations. The lack of semantics in this case limits the assistance possibilities. For instance, differently formulated similar queries cannot be match due to the syntax mismatch, or similar queries that only differ in filtering conditions will not be found mutually related due to the result differences. Additionally, approaches in this group use *user preferences*. For example, in [66] user preferences are applied over atomic query conditions and their complex combinations in SQL queries. Overall, this group mainly focuses on the metadata related to the user actions.

Because of it is analysis-oriented nature, the DW and OLAP fields are two significant contributors providing user assistance. Thus, the second group includes *OLAP user assistance* approaches. Similarly to the previous group, some of these approaches like [15, 38] also use *queries*, *query logs*, and *user sessions*. Additionally, other approaches also use *user preferences* in OLAP settings, e.g., for the visualization assistance with OLAP queries as in [18]. However, compared to the previous group, the *OLAP user assistance* group includes approaches that additionally apply user preferences over the *schema* as in [41] or use the *schema* for guiding natural language processing [75]. Furthermore, [75] also exploits various *statistics* for the user assistance features. Although this group extends the scope of the metadata artifacts used in traditional relational approaches, the *traceability*, *vocabulary*, *user characteristics*, and *profiling* metadata still remain generally unexploited. It is important to notice that none of the two groups discussed by now provide many details about the metadata used. Solutions like these model the metadata in a proprietary and ad-hoc manner, that is not easily reusable nor extensible, and to the best of our knowledge none of the approaches is RDF-based.

The third group focuses on *general metadata modeling* approaches. Compared to the previous two groups, these approaches are aim at gathering metadata and provide details about the metadata artifacts modeled. The most complete solution in this group is the Common Warehouse Metamodel (CWM) [79], which is a standardized solution for metadata modeling and management. However, as it is mostly intended for the interchange of metadata in data warehousing settings, it only partially covers the AM artifacts. Namely: the *schema*, *traceability*, and *vocabulary* metadata. Furthermore, the CWM is not RDF-based which requires significantly greater efforts for the correlation with the models instantiating it and much less flexibility for the changes. Another metamodel solution is presented in [72] that focuses on the traceability between the data sources and the target schema and their relation with the user requirements in the data warehouse context. Likewise, this solution represents a specialized metamodel that only covers the *traceability* metadata. Again, it is intended for data warehousing while the AM focus on providing assistance during the data analysis task. Finally, an approach to model user preferences is proposed in [68] that is again not RDF-based and captures only the *user preferences* and *schema* metadata. Table 3.6 illustrates

the comparison of all previous approaches with *SM4AM*. Unlike any of the earlier approaches, *SM4AM* supports all of the AM metadata artifacts.

Table 3.6: Related Work Comparison

	Query	Query Log	Session	Preferences	Schema	Statistics	Traceability	Vocabulary	User Charact.	Profiling	RDF-based
SnipSuggest [63]	√	√	√								
DB Explore [26]	√	√	√								
Personalize DB [66]	√			√							
Recommend OLAP [38]	√	√	√								
Predict OLAP [15]	√	√	√								
Personalize OLAP [18]	√			√							
myOLAP [41]	√			√	√						
Meta-Morphing [75]	√			√	√	√					
Preference Metamodel [68]				√	√						
Trace Metamodel [72]							√				
CWM [79]					√		√	√			
SM4AM	√	√	√	√	√	√	√	√	√	√	√

7 Conclusions

Motivated by the need to better support and assist the user experience in the context of BI 2.0 systems, in this paper we have presented *SM4AM*: an RDF-based metadata metamodel. Using ontological metamodeling, *SM4AM* has been designed as a flexible solution that can be easily shared among heterogeneous BI 2.0 systems. Being represented in a semantic-aware format it supports the metadata processing automation. Furthermore, the creation of the system-specific metadata models is supported by the proposed method for metamodel instantiation. The practical benefits of *SM4AM* for narrowing the (meta)data search space in a metamodel-driven (meta)data exploration are shown on a use case with two real-world data sets. Therefore, our approach provides foundations for the metadata modeling and organization needed for the user assistance engines in BI 2.0 systems that was missing until now. As discussed in the related work, to the best of our knowledge no previous approach captures all AM in a single unified metamodel.

The contributions of this paper open up several research lines for developing novel *user-centric* approaches. The next step in our future work is to create a metadata model by instantiating *SM4AM* and implement the metadata repository of our own BI 2.0 solution described in [56]. Then, we will develop the metadata processing techniques for gathering the metadata instances (e.g., by monitoring) and devise our own solutions for user assistance. We will also explore the creation and exploitation of derived metadata concepts from the ones existing in the repository.

Acknowledgments

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate ”Information Technologies for Business Intelligence - Doctoral College“ (IT4BI-DC) and it has been partially supported by the Secretaria d’Universitats i Recerca de la Generalitat de Catalunya under 2014 SGR 1534.

1 Prefixes

Prefixes (other than sm4am) used throughout the paper are specified in Figure 32.

```
@prefix ex:<http://www.example.org/>
@prefix qb:<http://purl.org/linked-data/cube#>
@prefix qb4o:<http://purl.org/qb4olap/cubes#>
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>
@prefix indicators:<http://worldbank.270a.info/dataset/world-bank-indicators/structure>
@prefix sdmx-measure:<http://purl.org/linked-data/sdmx/2009/measure#>
@prefix sdmx-dimension:<http://purl.org/linked-data/sdmx/2009/dimension#>
@prefix property:<http://worldbank.270a.info/property/>
@prefix obs:<http://worldbank.270a.info/dataset/world-bank-indicators/>
@prefix year:<http://reference.data.gov.uk/id/year/>
@prefix coutnry:<http://worldbank.270a.info/classification/country/>
@prefix indicator:<http://worldbank.270a.info/classification/indicator/>
@prefix xsd:<http://www.w3.org/2001/XMLSchema#>
@prefix lsq:<http://lsq.aksw.org/res/>
@prefix sp:<http://spinrdf.org/sp#>
@prefix lsqv:<http://lsq.aksw.org/vocab#>
@prefix dbr:<http://dbpedia.org/resource/>
@prefix foaf:<http://xmlns.com/foaf/0.1/>
@prefix wgs84_pos:<http://www.w3.org/2003/01/geo/wgs84_pos#>
```

Fig. 32: Prefixes

Chapter 4

Dimensional Enrichment of Statistical Linked Open Data

The paper has been published in the
Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Volume 40: pp. 22-51 (2016). The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1016/j.websem.2016.07.003

Abstract

On-Line Analytical Processing (OLAP) is a data analysis technique typically used for local and well-prepared data. However, initiatives like Open Data and Open Government bring new and publicly available data on the web that are to be analyzed in the same way. The use of semantic web technologies for this context is especially encouraged by the Linked Data initiative. There is already a considerable amount of statistical linked open data sets published using the RDF Data Cube Vocabulary (QB) which is designed for these purposes. However, QB lacks some essential schema constructs (e.g., dimension levels) to support OLAP. Thus, the QB4OLAP vocabulary has been proposed to extend QB with the necessary constructs and be fully compliant with OLAP. In this paper, we focus on the enrichment of an existing QB data set with QB4OLAP semantics. We first thoroughly compare the two vocabu-

laries and outline the benefits of QB4OLAP. Then, we propose a series of steps to automate the enrichment of QB data sets with specific QB4OLAP semantics; being the most important, the definition of aggregate functions and the detection of new concepts in the dimension hierarchy construction. The proposed steps are defined to form a semi-automatic enrichment method, which is implemented in a tool that enables the enrichment in an interactive and iterative fashion. The user can enrich the QB data set with QB4OLAP concepts (e.g., full-fledged dimension hierarchies) by choosing among the candidate concepts automatically discovered with the steps proposed. Finally, we conduct experiments with 25 users and use three real-world QB data sets to evaluate our approach. The evaluation demonstrates the feasibility of our approach and shows that, in practice, our tool facilitates, speeds up, and guarantees the correct results of the enrichment process.

1 Introduction

On-Line Analytical Processing (OLAP) is a well-established approach for data analysis to support decision making that typically relates to Data Warehouse (DW) systems. It is based on the multidimensional (MD) model which places data in an *n*-dimensional space, usually called a *data cube*. In this way, a user can analyze data along several dimensions of interest. For instance, a user can analyze sales data according to time and location (dimensions). The simplicity of the MD model specially fits the business users who navigate and analyze the MD data by means of OLAP operations (typically via a graphical user interface).

A large number of MD models in the literature are based on the data cube metaphor [42, 47, 107]. Historically, DW and OLAP have been used as techniques for data analysis within an organization, using mostly commercial tools with proprietary formats. However, initiatives like Open Data[1] and Open Government[2] are pushing organizations to publish MD data using standards and non-proprietary formats. Although several open source platforms for business intelligence (BI) have emerged in the last decade, an open format to publish and share cubes among organizations is still missing. The *Linked Data* [45] initiative promotes sharing and reusing data on the web using *semantic web* (SW) standards and domain ontologies expressed in the Resource Description Framework (RDF) as the basic data representation layer for the SW [28], or in languages built on top of RDF (e.g., RDF-Schema [23] and OWL[3]).

Two main approaches can be found in the literature concerning OLAP analysis of SW data. The first one consists in extracting MD data from the

[1] `http://okfn.org/opendata/`

[2] `http://opengovdata.org/`

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

web and loading them into traditional data management systems for OLAP analysis. The second one explores data models and tools that allow publishing and performing OLAP analysis directly over MD data on the SW. We discuss both approaches in detail in Section 7 and in this paper we follow the second one.

Statistical data sets on the SW are usually published using the *RDF Data Cube Vocabulary*[4] (also denoted as QB), the current W3C standard. There is already a considerable amount of data sets published using QB. However, as we explain later, QB lacks (among other shortcomings) the structural metadata needed to automate the translation of OLAP operations into the underlying technology storing the MD data. For example, DWs have been typically implemented using relational technology and the definition of a *well-formed* MD schema allows the automatic translation of OLAP operations into SQL queries. To address this challenge, a new vocabulary, denoted as QB4OLAP, has been proposed [33]. QB4OLAP allows reusing data already published in QB just by adding the needed MD schema semantics (e.g., the hierarchical structure of the dimensions) and the corresponding instances that populate the dimension levels. Thus, the main task that we address in this paper is *the enrichment of an existing QB data set with additional QB4OLAP semantics*. Once a data cube is published using QB4OLAP, users will be able to operate over it, not only through queries written in SPARQL [82] (the standard query language for RDF), but also by using a high-level OLAP query language [102]. Such a language allows OLAP users to query data cubes directly on the SW, without any knowledge of SPARQL or RDF, since OLAP queries and operations can be automatically translated into SPARQL, taking advantage of the structural metadata provided by QB4OLAP[5]. In addition, a language like this makes it easier to develop graphic tools, typically used to exploit data cubes.

Enriching an existing QB data set with QB4OLAP semantics implies a labor-intensive, repetitive, and error-prone task. Thus, it must be performed as automatically as possible. For instance, hierarchical structures of the dimensions can be discovered from the source data and metadata, and from external data. Once discovered, the structure must be populated with the members of hierarchy levels. In this paper, we present a method and a tool to facilitate the enrichment process. The method minimizes the user effort, by automatically detecting new potential semantics and performing otherwise time-consuming tasks, leaving to the user the task of providing the MD semantics that cannot be inferred.

Contributions. Our main contributions are:

• An in-depth comparison between the QB and QB4OLAP vocabularies, outlining the novel benefits of the latter.

[4] `http://www.w3.org/TR/vocab-data-cube/`

[5] A prototype is available at `http://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/`

• Techniques to automate (a) the association between measures and aggregate functions, by means of metadata; and (b) the discovery of dimension hierarchy schema and instances, based on an algorithm that detects implicit MD semantics.

• A method defining the steps described as SPARQL queries to semi-automatically enrich data already published in QB with dimensional (meta)-data compliant with the QB4OLAP vocabulary.

• QB2OLAPem, a tool that in an iterative fashion implements the method and the algorithm for the detection of implicit MD semantics. The tool enables the user to semi-automatically produce a QB4OLAP description of a QB data cube with minimal manual effort.

• An evaluation of our approach based on the experiments conducted with 25 users and the use of three real-world QB data sets. The evaluation shows that our approach is feasible and that, in practice, QB2OLAPem reduces the enrichment time and user efforts and guarantees that the QB4OLAP schema created is correct.

The remainder of the paper is organized as follows. Section 2 explains the basic concepts used throughout the paper. Section 3 discusses the limitations of the QB vocabulary, with respect to its capability to represent MD data. This section also presents the QB4OLAP vocabulary, which addresses these limitations. Section 4 studies the automation challenges and provides possible solutions for the two most important ones. Section 5 describes the proposed enrichment method while Section 6 presents the approach evaluation. Finally, Section 7 discusses related work and we conclude in Section 8.

2 Preliminaries

In this section we present the basic concepts on OLAP and SW data models followed by a detailed elaboration on QB based on the running example used throughout the paper.

2.1 OLAP

In OLAP, data are organized as *hypercubes* whose axes are called *dimensions*. Each point in this MD space is mapped into one or more spaces of *measures*, representing *facts* that are analyzed along the cube's dimensions. Dimensions are structured in *hierarchies* that allow analysis at different aggregation *levels*. The actual values in a dimension level are called *members*. A *Dimension Schema* is composed of a non-empty finite set of levels. We denote by '$\rightarrow$' a partial order on these levels, with a unique bottom, and a unique top, the latter being a distinguished level denoted as *All*, whose only member is called

all. We denote by '$\rightarrow^*$' the reflexive and transitive closure of '$\rightarrow$'. Levels can have *attributes* describing them. A *Dimension Instance* assigns a set of dimension members to each dimension level in the dimension schema. For each pair of levels (l_j, l_k) in the dimension schema, such that $l_j \rightarrow l_k$, a relation (denoted as *rollup*) is defined, associating members from level l_j with members of level l_k. In a rollup relationship, l_j is called the *child* level and l_k the *parent* level. In practice, to guarantee the correct aggregation of the measure values, rollup relations actually become functions. *Cardinality constraints* on these relations are then used to restrict the number of level members related to each other [101]. A *Cube Schema* is defined by a set of dimensions and a set of measures, and for each measure a *default aggregate function* is specified. Each dimension is represented by a level, defining the *granularity* of the cube. The cube composed by the bottom levels of each dimension is called a *base cube*. All other cubes are called *cuboids*. A *Cube Instance*, corresponding to a cube schema, is a partial function mapping coordinates from dimension instances (at the cube's granularity level) into measure values.

A well-known set of operations can be defined over cubes [29]. For example, given a cube $\mathcal{C}$, a dimension $D \in \mathcal{C}$, dimension levels $l_l, l_u \in D$ such that $l_l \rightarrow^* l_u$, and an aggregate function F_{agg}, $RollUp(\mathcal{C}, D, l_u, F_{agg})$ returns a new cube where measure values are aggregated along D, from the current level l_l up to a level l_u, using F_{agg}. Analogously, $DrillDown(\mathcal{C}, D, l_l, F_{agg})$ disaggregates previously summarized data, from the current level l_u down to a level l_l and can be considered the inverse of *RollUp*. Note that we do not need to use the starting levels as parameters of these operations, because we assume that they are applied over the 'current' aggregation level of the cube, thus they would be redundant, since the cube 'knows' the current aggregation level for dimension D. $Slice(\mathcal{C}, D, F_{agg})$ receives a cube $\mathcal{C}$, a dimension $D \in \mathcal{C}$, and an aggregate function F_{agg}, and returns a new cube, with the dimension D removed from the original schema, such that measure values are aggregated along D up to level *All* before removing the dimension, using F_{agg}. Note that in all cases, F_{agg} could be omitted if the default aggregate function is used. Finally, given a cube $\mathcal{C}$, and a first order formula σ over levels and measures in $\mathcal{C}$, $Dice(\mathcal{C}, \sigma)$ returns a new cube with the same schema, and whose instances are the instances in $\mathcal{C}$ that satisfy σ. For example, given a *Sales* data cube, with dimensions Time and Location, the query "Total sales by region, in December 2015" can be expressed with the following sequence of operations: $C1 := Dice(Sales, Time.month = "12-2015")$; $C2 := Rollup(C1, Location, Location.region)$. The first operation takes as input the *Sales* data cube, and produces another data cube C1, whose cells contain only sales data corresponding to December, 2015; C1 is then summarized by geographical region.

2.2 RDF and the Semantic Web

The basic construct of RDF is a triple, of the (s, p, o) form, where s stands for *subject*, p for *predicate*, and o for *object*. In general, s, p, and o are *resources*, identified with internationalized resource identifiers (IRIs). An object can also be a data value, denoted as a *literal* in RDF, or a *blank node*, typically used to represent anonymous resources. Subjects can also be represented by blank nodes. A set of RDF triples is called an *RDF graph*. An *RDF data set* is a collection of RDF graphs, comprising one default RDF graph with no name, and zero or more *named graphs* (a named graph is a graph with a name, typically an IRI or a blank node). Graph names must be unique within an RDF data set.

In addition, the *RDF Schema* (RDF-S) [23] is composed by a set of reserved keywords which define classes, properties, and hierarchical relationships between them. For example, the triple (r, `rdf:type`, c) explicitly states that r is an instance of c, and it also implicitly states that object c is an instance of `rdfs:Class`. Many formats for RDF serialization exist. In this paper we use Turtle [17].

SPARQL 1.1 [82] is the W3C standard query language for RDF. The query evaluation mechanism of SPARQL is based on subgraph matching: RDF triples are interpreted as nodes and edges of directed graphs, and the query graph is matched to the data graph, instantiating the variables in the query graph definition. The selection criteria is expressed as a graph pattern in the `WHERE` clause. Relevant to OLAP queries, SPARQL 1.1 supports aggregate functions and the `GROUP BY` clause, a feature not present in previous versions.

From here on we assume that the reader is familiar with RDF and SPARQL concepts.

2.3 QB: The RDF Data Cube Vocabulary

As mentioned before, QB is the W3C recommendation to publish statistical data and metadata in RDF. QB is based on the main components of the SDMX information model [91], proposed by the Statistical Data and Metadata eXchange initiative (SDMX)[6] for the publication, exchange, and processing of statistical data. The elements with white background in Figure 4.1 depict the QB vocabulary. Capitalized terms represent RDF classes and non-capitalized terms represent RDF properties. Capitalized terms in italics represent classes with no instances. An arrow with black triangle head from class A to class B, labeled *rel* means that *rel* is an RDF property with domain A and range B. White triangles represent sub-classes or sub-properties. The range of a property can also be denoted using ":". For better comprehension, we next

[6] `http://SDMX.org`

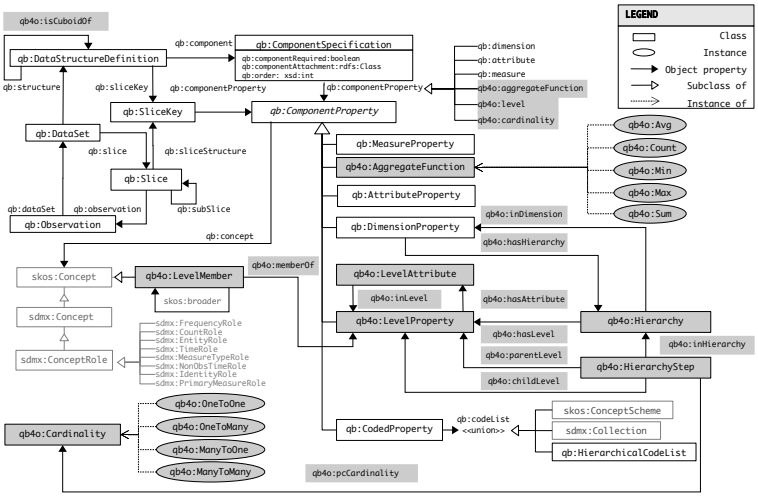

Fig. 4.1: QB (cf. [27]) and QB4OLAP vocabularies

introduce the running example that is used to explain the QB elements and then followed throughout the paper.

We use data published by the World Bank[7], a financial institution supporting developing countries, basically through loans for strategic projects. Free and open access to data about these countries is provided through the World Bank Open Data (WBOD), that includes a collection of indicators[8] measured for different countries and regions across time. Data are available in tabular, RDF, and many other formats depending on the particular portion of the data set. World Bank Linked Data (WBLD) is a Linked Data data set created from WBOD data via its rdf-ization (where needed) and it is annotated with the QB vocabulary. The WBLD is organized in four subsets, stored in different files, including demographic and financial indicators, projects and operations, and climate data. Additionally, there is a VoiD[9] file which contains metadata that describe the data sets. Moreover, a SPARQL endpoint[10] is also available to query the WBLD. Our running example is based on the "Market capitalization of listed companies (current US$)" indicator (CM.MKT.LCAP.CD)[11], where market capitalization refers to the share price times the number of shares outstanding. Each indicator is provided as a QB data set, i.e., as an instance of the class `qb:DataSet`.

The schema of a QB data set is specified by means of the *data structure definition* (DSD), an instance of the class `qb:DataStructureDefinition`. This

[7] `http://www.worldbank.org`
[8] `http://data.worldbank.org/indicator`
[9] `http://semanticweb.org/wiki/VoID`
[10] `http://worldbank.270a.info/sparql`
[11] `http://data.worldbank.org/indicator/CM.MKT.LCAP.CD`

specification is composed of a set of *component* properties, instances of subclasses of the `qb:ComponentProperty` class, representing *dimensions*, *measures*, and *attributes*. Component properties are not directly related to the DSD: the `qb:ComponentSpecification` class is an intermediate class typically instantiated as RDF blank nodes, that allows specifying additional attributes for a component in a DSD (e.g., a component may be tagged as *required* (i.e., mandatory), using the `qb:componentRequired` property). The different components that belong to a component specification are linked using specific properties that depend on the type of the component: `qb:dimension` for dimensions, `qb:measure` for measures, and `qb:attribute` for attributes. Component specifications are linked to DSDs via the `qb:component` property. Note that a DSD can be shared by different QB data sets (and each QB data set is linked to its DSD) by means of the `qb:structure` property. Example 1 presents the triples that represent the DSD of our running example.

Example 1
The DSD of the running example is defined in the file `meta.rdf`[12] and looks as follows.

```
1  @prefix qb: <http://purl.org/linked-data/cube#> .
2  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3  @prefix sdmx-dimension: <http://purl.org/linked-data/sdmx/2009/dimension#> .
4  @prefix sdmx-measure: <http://purl.org/linked-data/sdmx/2009/measure#>.
5
6  <http://worldbank.270a.info/dataset/world-bank-indicators/structure>
7    a qb:DataStructureDefinition ;
8    qb:component [
9      a qb:ComponentSpecification ;
10     qb:dimension <http://worldbank.270a.info/property/indicator> ;
11     qb:order "1"^^xsd:int],
12     [
13     a qb:ComponentSpecification ;
14     qb:dimension sdmx-dimension:refArea ;
15     qb:order "2"^^xsd:int],
16     [
17     a qb:ComponentSpecification ;
18     qb:dimension sdmx-dimension:refPeriod ;
19     qb:order "3"^^xsd:int],
20     [
21     a qb:ComponentSpecification ;
22     qb:measure sdmx-measure:obsValue ;
23     qb:order "4"^^xsd:int] .
```

This DSD is composed of three dimensions: `<http://worldbank.270a.info/property/indicator>` (lines 9-11), representing an indicator, `sdmx-dimension:refArea` (lines 13-15) which represents the geographical reference area, and `sdmx-dimension:refPeriod` (lines 17-19) which represents the time period. The measure of the data set is the generic `sdmx-measure:obsValue` predicate (lines 21-23).

Instances of the running example data set are described in an RDF graph

[12] `http://worldbank.270a.info/data/meta/meta.rdf`

contained in the file `CM.MKT.LCAP.CD.rdf`.[13] Such instances are called *observations* in the QB vocabulary. *Observations* (in OLAP terminology, *facts*) are instances of the `qb:Observation` class and represent points in an MD data space indexed by *dimensions*. They are associated with *data sets* (instances of the `qb:DataSet` class), through the `qb:dataSet` property. Each observation can be linked to a value in each dimension of the DSD via instances of the `qb:DimensionProperty` class; analogously, values for each observation are associated to measures via instances of the `qb:MeasureProperty` class; and instances of the `qb:AttributeProperty` class are used to associate attributes to observations. Example 2 presents the triples of an observation from our running example.

Example 2
The triples representing an observation corresponding to the market capitalization for Serbia in 2012 (we do not repeat prefixes previously defined).

```
1  @prefix property: <http://worldbank.270a.info/property/> .
2  @prefix indicator: <http://worldbank.270a.info/classification/indicator/>.
3  @prefix country: <http://worldbank.270a.info/classification/country/> .
4  @prefix year: <http://reference.data.gov.uk/id/year/> .
5
6  <http://worldbank.270a.info/dataset/world-bank-indicators/
7    CM.MKT.LCAP.CD/RS/2012> a qb:Observation ;
8    qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> ;
9    property:indicator indicator:CM.MKT.LCAP.CD ;
10   sdmx-dimension:refArea country:RS ;
11   sdmx-dimension:refPeriod year:2012 ;
12   sdmx-measure:obsValue 7450560827.04874 .
```

Note that each of the RDF properties defined as components of the data set DSD (Example 1) is used here to link the observation with either dimension members or measure values. In particular, the recorded value for `CM.MKT.LCAP.CD` indicator is linked to the observation via the `sdmx-measure:obsValue` predicate (line 12), and the semantics of this measure is given by the indicator linked to the observation via the `property:indicator` predicate (line 9).

To give further semantics to the components of a DSD, they may be associated with concepts in an ontology. For this, we can make use of the property `qb:concept`, to link components in a DSD, with instances of the class `skos:Concept` defined in the SKOS vocabulary.[14] More specifically, this property can be used to link component properties (i.e., dimensions or measures), with standard concepts defined in the SDMX guidelines (e.g., reference area, frequency, etc.) [90]. We illustrate this in Example 3, to define the dimension `sdmx-dimension:refPeriod`.

[13] `http://worldbank.270a.info/data/world-development-indicators/CM.MKT.TRAD.CD.rdf`

[14] `http://www.w3.org/TR/skos-reference/`

Example 3
An excerpt of the triples that define the dimension `sdmx-dimension:refPeriod`, and associate it with the SDMX concept `sdmx-concept:refPeriod` (a SKOS concept) is shown below.

```
1  @prefix sdmx-concept: <http://purl.org/linked-data/sdmx/2009/concept#> .
2  @prefix rdf: <http://www.w3.org/1992/02/22-rdf-syntax-ns#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4  @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
5
6  sdmx-dimension:refPeriod a qb:DimensionProperty, rdf:Property ;
7          rdfs:range rdfs:Resource;
8          qb:concept sdmx-concept:refPeriod ;
9          rdfs:label "Reference Period"@en ;
10         rdfs:comment """The period of time or point in time to which the
11         measured observation is intended to refer."""@en .
12
13 sdmx-concept:refPeriod a sdmx:Concept, skos:Concept ;
14         rdfs:label "Reference Period"@en ;
15         rdfs:comment """The period of time or point in time to which the
16         measured observation is intended to refer."""@en;
17         skos:inScheme sdmx-concept:cog .
```

Linking the dimension `sdmx-dimension:refPeriod` with the concept `sdmx-concept:refPeriod` (line 8) allows to give semantics to this dimension.

Finally, *slices*, as defined in QB, represent subsets of observations, not as operators over an existing cube, but as new structures and new instances (observations) in which one or more values of dimension members are fixed. The structure of a slice is defined using a DSD and an instance of the `qb:SliceKey` class. The class `qb:Slice` allows grouping the observations that correspond to a particular slice (using the `qb:observation` property) and the structure of each slice is attached using the `qb:sliceStructure` property.

3 Representing Multidimensional Data in RDF

Although appropriate to represent and publish statistical data, QB has a set of limitations when it comes to represent an MD model for OLAP. Thus, in this section we elaborate on these limitations of QB, introduce the QB4OLAP vocabulary that extends QB with the necessary concepts, discuss some QB4OLAP design decisions, and provide hints about the use of QB4OLAP.[15]

3.1 Limitations of QB

Lack of support for an OLAP dimension structure Although QB allows representing hierarchical relationships between *level members* in the *dimension*

[15]Parts of the material in this section have previously appeared in [33, 100]. However, the contents of [33] have been updated and now refer to newer versions of QB4OLAP. Further, the examples based on WBLD are new. Finally, we remark that [100] is a tutorial on QB4OLAP, produced for the 2015 edition of the Business Intelligence Summer School.

instances, it does not provide a mechanism to represent an OLAP dimension structure (i.e., the dimension levels and the relationships between levels). That means, QB allows stating that Serbia is a narrower concept than Europe, but not that Serbia is a Country, Europe is a Continent, and that countries aggregate to continents. To represent hierarchical relationships between dimension members, the semantic relationship `skos:narrower` should be used, with the following meaning: If two concepts `A` and `B` are such that `A skos:narrower B`, `B` represents a narrower concept than `A` (e.g., `continent skos:narrower country`).

Additional information that can be used to build *dimension instances* is scattered across many graphs. For example, we can obtain information about `country:RS` (Serbia) from the graph in the file `countries.rdf`.[16] Example 4 shows the triples that can be obtained about Serbia.

Example 4
The triples about the dimension member Serbia, obtained from `countries.rdf`.

```
1  @prefix dbpedia: <http://dbpedia.org/resource/> .
2  @prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
3  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
4  @prefix dcterms: <http://purl.org/dc/elements/1.1/> .
5  @prefix region: <http://worldbank.270a.info/classification/region/> .
6  @prefix income: <http://worldbank.270a.info/classification/income-level/> .
7  @prefix lending: <http://worldbank.270a.info/classification/lending-type/> .
8
9  <http://worldbank.270a.info/classification/country> skos:hasTopConcept
10   country:RS .
11
12 country:RS
13   a skos:Concept, <http://dbpedia.org/ontology/Country> ;
14   skos:inScheme <http://worldbank.270a.info/classification/country> ;
15   skos:topConceptOf <http://worldbank.270a.info/classification/country> ;
16   skos:notation "RS" ;
17   skos:exactMatch country:SRB ;
18   skos:prefLabel "Serbia"@en ;
19   property:region region:ECS ;
20   property:admin-region region:ECA ;
21   property:income-level income:UMC ;
22   property:lending-type lending:IBD ;
23   dbpedia:capital "Belgrade"@en ;
24   geo:lat "20.4656"^^xsd:float ;
25   geo:long "44.8024"^^xsd:float ;
26   foaf:page <http://data.worldbank.org/country/RS> ;
27   ...
28   dcterms:created "2012-02-29T00:00:00Z"^^xsd:dateTime ;
29   dcterms:issued "2013-11-04T13:37:18Z"^^xsd:dateTime .
30
31 country:SRB skos:exactMatch country:RS ; skos:notation "SRB" .
```

Some of the triples provide information that can be used to define dimension hierarchies, when producing the QB4OLAP representation. Line 13 states that `country:RS` is a country as it says that this IRI is of type `<http://dbpedia.org/ontology/Country>`. Lines 19 and 20 state that Serbia belongs to two different regions[17]: `region:ECS` (Eu-

[16] `http://worldbank.270a.info/data/meta/countries.rdf`
[17] `http://worldbank.270a.info/classification/region`

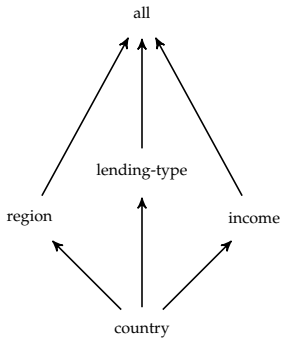

Fig. 4.2: Dimension levels and hierarchies with bottom level country.

rope & Central Asia (all income levels)) and `region:ECA` (Europe & Central Asia (developing only)), respectively. Lines 21 and 22 provide information about the income level and the type of lending Serbia is eligible for, respectively.[18]

We have said that a typical OLAP user explores data, e.g., performing aggregations along dimension hierarchies. For example, she would like to compute the total capitalization by region, income level, or lending type, given that these data are available in the data set, as Examples 1 through 4 show. However, we can also see that these data are given at the instance level, that means, no hierarchical structure is defined for the refArea dimension. This is because QB does not allow to define aggregation paths. Nevertheless, it is clear that from the information available, we could infer and build a dimension hierarchy. A possible structure for a geographical dimension (like refArea) is shown in Figure 4.2. We can see that there are five levels, namely country (i.e., initial refArea), region, lending-type, income, and, following traditional MD design, a distinguished level All which has only one level member, denoted as all. These levels are organized into three hierarchies: a *geographical* hierarchy country $\rightarrow$ region $\rightarrow$ All, a *lending type* hierarchy country $\rightarrow$ lending-type $\rightarrow$ All, and an *income* hierarchy country $\rightarrow$ income $\rightarrow$ All.

Lack of support for aggregate functions QB does not provide native support to represent aggregate functions. Many OLAP operations change the granularity level of the data represented in a data cube (e.g., a rollup over the Time dimension from the Month level up to the Year level). This involves aggregating measure values along dimensions, using the aggregate function defined for each measure. These aggregate functions depend on the nature

[18]These concepts are defined in `http://worldbank.270a.info/classification/income-level` and `http://worldbank.270a.info/classification/lending-type`.

of the measure (i.e., additive, semi additive, non additive [101]). The ability to link each measure with an aggregate function is therefore crucial and, although present in OLAP tools, it is not considered in QB.

Lack of support for descriptive attributes In the MD model, the instances (members) of each dimension level usually contain a set of real-world concepts with similar characteristics. Further, the schema of each level is composed of a set of *attributes* that describe the characteristics of their members (e.g., the level country may have the attributes countryName, surface, etc.) and one or more *identifiers* [101]. QB does not provide a mechanism to associate a set of attributes with a dimension level. This affects the expressiveness and efficiency of some OLAP operations, in particular, *Dice*, which filters a cube according to a Boolean condition. For example, to obtain a cube containing just data about Serbia, without descriptive attributes we would need to filter such data using the *IRI* representing Serbia, instead of the proper string. This would not only be unnatural for a user, but also highly inefficient. Note that the `qb:AttributeProperty` class, used in QB to associate attributes to observations as mentioned before, differs from descriptive level attributes as defined in QB4OLAP, and cannot be used in the way explained in the example above. Section 3.4 illustrates the use of descriptive attributes in a *Dice* operation.

3.2 The QB4OLAP Vocabulary

QB4OLAP[19] extends QB with a set of RDF terms and the rationale behind QB4OLAP includes:

- QB4OLAP must be able to represent the most common features of the MD model. The features considered are based on the MultiDim model [101].
- QB4OLAP must allow to operate over already published observations which conform to DSDs defined in QB, without the need of rewriting the existing observations. Note that in a typical MD model, observations are the largest part of the data while dimensions are usually orders of magnitude smaller.
- QB4OLAP must include all the metadata needed to automatically generate SPARQL queries implementing OLAP operations. In this way, OLAP users do not need to know SPARQL (which is the case of typical OLAP users) and even wrappers for OLAP tools can be developed to query RDF data sets directly (we give an example of this in Section 3.4).

The elements with gray background in Figure 4.1 depict the QB4OLAP vocabulary. Moreover, original QB terms are prefixed with "`qb:`"; QB4OLAP

[19] `http://purl.org/qb4olap/cubes`

terms are prefixed with "qb4o:". In addition to the QB graphical notation, QB4OLAP introduces ellipses representing class instances and dashed arrows representing rdf:type relationships.

As already mentioned, dimension hierarchies and levels are first-class citizens in an MD model for OLAP. Therefore, QB4OLAP focuses on their representation and several classes and properties are introduced to this end. QB4OLAP represents the structure of a data set in terms of *levels* and measures, instead of *dimensions* and measures (which is the case of QB), thus allowing us to specify the granularity level considered for each dimension. *Dimension levels* are represented in QB4OLAP in the same way that QB represents dimensions: as classes of properties. The class qb4o:LevelProperty represents dimension levels. Declaring it as a sub-class of qb:ComponentProperty allows specifying the schema of the cube in terms of dimension levels, using qb:DataStructureDefinition. To represent *aggregate functions* the class qb4o:AggregateFunction can be used. The property qb4o:aggregateFunction associates measures with aggregate functions, and, together with the concept of component sets, allows a given measure to be associated with different aggregate functions in different cubes. Given the structure described above, in QB4OLAP, fact instances (observations) map *level members* to measure values. It is also worth noting that, in general, each fact is related with at most one level member, for each level that participates in the fact. However, there are cases where this restriction does not hold, yielding so-called many-to-many dimensions [101]; thus, to support these dimensions, the property qb4o:cardinality can be used to represent the cardinality of the relationship between a fact and a level.

Example 5 shows how the cube in our running example would look like in QB4OLAP (we explain how we came up with this schema in Section 5). Figure 4.3 presents the definition of the prefixes (not included in the examples so far) that we use in the sequel.

```
1 @prefix classification: <http://worldbank.270a.info/classification/> .
2 @prefix dataset: <http://worldbank.270a.info/dataset/> .
3 @prefix qb4o: <http://purl.org/qb4olap/cubes#> .
4
5 #QB4OLAP schema and instances
6 @prefix schema:
7 <http://www.fing.edu.uy/inco/cubes/schemas/world-bank-indicators#> .
8 @prefix instances:
9 <http://www.fing.edu.uy/inco/cubes/instances/world-bank-indicators#> .
```

Fig. 4.3: RDF prefixes to be used in the examples

Example 5
The new DSD for the running example data cube defined using QB4OLAP.

```
schema:QB4O_CM_MKT_LCAP_CD
        a qb:DataStructureDefinition ;
        qb:component [ qb:measure sdmx-measure:obsValue;
                       qb4o:aggregateFunction qb4o:Sum ] ;
        qb:component [ qb4o:level property:indicator ] ;
        qb:component [ qb4o:level sdmx-dimension:refArea ] ;
        qb:component [ qb4o:level sdmx-dimension:refPeriod ] .

property:indicator a qb4o:LevelProperty.
sdmx-dimension:refArea a qb4o:LevelProperty.
sdmx-dimension:refPeriod a qb4o:LevelProperty.
sdmx-measure:obsValue a qb:MeasureProperty.

dataset:CM.MKT.LCAP.CD qb:structure
                       schema:QB4O_CM_MKT_LCAP_CD.
```

The DSD is defined in terms of dimension levels such that the dimension properties of the original QB cube are declared as instances of `qb4o:LevelProperty` and considered the lowest levels in the dimension hierarchy. Thus, we avoid rewriting the observations. Readers familiar with OLAP technology may note that `property:indicator` refers to the MDX's *Measures* dimension. MDX is a de facto standard language for OLAP (see [101] for details).

To represent *dimension hierarchies* the `qb4o:Hierarchy` class is introduced. The relationship between dimensions and hierarchies is represented via the property `qb4o:hasHierarchy` and its inverse `qb4o:inDimension`. To support the most common conceptual models, we need to allow declaring that a level may belong to different hierarchies, and that each level may have a different set of parent levels. Also, the relationship between level members may have different cardinality constraints (e.g., one-to-many, many-to-many, etc.). The class `qb4o:HierarchyStep` allows this by means of the reification of the parent-child relationship between two levels in a hierarchy. Each hierarchy step is linked to its two component levels using the `qb4o:childLevel` and the `qb4o:parentLevel` properties, respectively, and is attached to the hierarchy it belongs to, using the `qb4o:inHierarchy`. The `qb4o:pcCardinality` property allows representing the cardinality constraints of the relationships between level members in this step, using members of the `qb4o:Cardinality` class, whose instances are depicted in Figure 4.1. Example 6 illustrates the above.

Example 6
The definition of the geographical dimension `schema:geoDim` according to Figure 4.2.

```
schema:geoDim a qb:DimensionProperty ;
  rdfs:label "Geographical dimension"@en;
  qb4o:hasHierarchy schema:geoHier, schema:lendingHier,
                    schema:incomeHier.
```

We now define each hierarchy, declare to which dimension it belongs, and which levels it traverses. We have three hierarchies in our schema, namely `schema:geoHier`,

`schema:lendingHier`, and `schema:incomeHier`. We just show the first of them, the other ones are analogous.

```
schema:geoHier a qb4o:Hierarchy ;
    rdfs:label "Geographical Hierarchy"@en ;
    qb4o:inDimension schema:geoDim;
    qb4o:hasLevel sdmx-dimension:refArea, schema:region, schema:geoAll.
```

Next, we define the base (i.e., finest granularity) level for the geographical dimension, that means, the one whose instances compose the observations, and the upper levels in each hierarchy. Note that the former are defined to be compatible with QB, but as *levels* instead of dimensions. The example shows only the geographical dimension while construction of other dimensions is analogous where only the All level is added to each of them.

```
# Base levels
sdmx-dimension:refArea a qb4o:LevelProperty;
    rdfs:label "country level"@en.

#Upper hierarchy levels
schema:region a qb4o:LevelProperty;
   rdfs:label "Geographical regions"@en.
schema:lendingtype a qb4o:LevelProperty;
   rdfs:label "Lending type level"@en.
schema:income a qb4o:LevelProperty;
   rdfs:label "Income level"@en.
schema:geoAll a qb4o:LevelProperty;
   rdfs:label "All reference areas"@en.
```

Finally, the hierarchy steps (i.e., parent-child relationships) are defined. Again, we just show the ones corresponding to the `schema:geoHier` hierarchy.

```
_:hs1 a qb4o:HierarchyStep;
    qb4o:inHierarchy schema:geoHier;
    qb4o:childLevel sdmx-dimension:refArea;
    qb4o:parentLevel schema:region;
    qb4o:pcCardinality qb4o:ManyToOne.

_:hs2 a qb4o:HierarchyStep;
   qb4o:inHierarchy schema:geoHier;
   qb4o:childLevel schema:region;
   qb4o:parentLevel schema:geoAll;
   qb4o:pcCardinality qb4o:ManyToOne.
```

To represent *level attributes*, QB4OLAP provides the class of properties `qb4o:LevelAttribute`, linked to `qb4o:LevelProperty` via the `qb4o:hasAttribute` property. Instances of this class are used to link level instances with attribute values. Example 7 shows the definition of an attribute for the `sdmx-dimension:refArea` dimension level.

Example 7
Definition of a level attribute.

```
sdmx-dimension:refArea qb4o:hasAttribute
    schema:capital.
schema:capital a qb4o:LevelAttribute;
    rdfs:range xsd:string .
```

We assume that we add the attribute `schema:capital` to the `sdmx-dimension:refArea` dimension level.

At the instance level, *dimension level members* are represented as instances of the class `qb4o:LevelMember`, which is a sub-class of `skos:Concept`. Members are attached to the levels they belong to, using the property `qb4o:memberOf`, which resembles the semantics of `skos:member`. Rollup relationships between members are expressed using the property `skos:broader`, conveying the idea that hierarchies of level members should be navigated from finer granularity concepts up to coarser granularity concepts. Example 8 shows some examples of dimension members for the dimension `schema:geoDim`.

Example 8
The details for the dimension members corresponding to Serbia.

```
country:RS a qb4o:LevelMember ;
    qb4o:memberOf sdmx-dimension:refArea ;
    skos:broader lending:IBD ;
    skos:broader income:UMC ;
    skos:broader region:ECS ;
        skos:prefLabel "Serbia"@en .

lending:IBD a qb4o:LevelMember ;
    qb4o:memberOf schema:lending ;
    skos:broader instance:geoAll ;
        skos:prefLabel "IBRD"@en .

income:UMC a qb4o:LevelMember ;
        qb4o:memberOf schema:income ;
        skos:broader instance:geoAll ;
        skos:prefLabel "Upper middle income"@en .

region:ECS a qb4o:LevelMember ;
    qb4o:memberOf schema:region ;
    skos:broader instance:geoAll ;
        skos:prefLabel "Europe & Central Asia (all income levels)"@en .

instance:geoAll a qb4o:LevelMember ;
    qb4o:memberOf schema:geoAll ;
    skos:prefLabel "Geo ALL"@en .
```

Note that, for attribute instances, we need to link IRIs corresponding to level members, with attribute values. In our example for the geographical dimension:

```
country:RS schema:capital "Belgrade"^^xsd:string .
```

3.3 Discussion: From QB observations to QB4OLAP

Let us now comment on some decisions underlying the QB4OLAP design. As we have already mentioned, observations in QB are specified as dimension properties, while in QB4OLAP they are specified as level properties, to allow defining hierarchies, as usual in OLAP. Therefore, in order to be able to work with existing QB observations, a new DSD is defined in terms of

QB4OLAP *dimension levels*. This saves the cost that would imply adding, for each observation, triples for linking the observation with *level* members using newly defined level properties. We thus propose to define, in the new DSD, a *level property* for each *dimension property* in the existing DSD, and consider the former as the bottom level of each corresponding dimension in the new DSD. Of course, as a consequence, the same elements that in QB are considered dimensions, in QB4OLAP play the role of dimension *levels*, as in the case of `sdmx-dimension:refPeriod` and `sdmx-dimension:refArea`.

Further, instead of defining a new concept in QB4OLAP to represent dimensions, QB4OLAP uses the QB class `qb:DimensionProperty`. This does not produce a semantic contradiction, given that the QB specification states that this class is "The class of component properties which represent the dimensions of the cube", a definition that still holds in the QB4OLAP interpretation. In addition, since level members in QB4OLAP are instances of the class `skos:Concept`, we can use existing QB dimension members to populate QB4OLAP dimension levels. To accomplish this, IRIs that represent dimension members have to be linked with level members via the `qb4o:memberOf` property.

3.4 Using QB4OLAP

We have mentioned that one of the main advantages of using QB4OLAP instead of QB to represent MD data on the SW, is that QB4OLAP allows us to write high-level OLAP queries and automatically translate them into SPARQL. This way, OLAP users may exploit MD data directly over the web, and query them without any knowledge of SPARQL, or without the need of exporting these data to a relational repository. The obvious consequence is an enhancement of the usability of data published on the web. Giving complete details of how this can be achieved is beyond the scope of this paper, but we would like to at least convey the main idea through an example query.

Let us consider the query "Total market capitalization of listed companies, grouped by income level, in the period [2010,2012]." This is a typical OLAP query involving two main operations, as described in Section 2.1: First, a selection of the values corresponding to the years mentioned in the query (a *Dice* operation). Second, an aggregation, using the SUM aggregate function, along the geographical dimension, using the `schema:incomeHier` up to the `schema:income` level. This can be expressed, in a high-level, cube-based, conceptual algebra like the one proposed in [29] (using the operations defined in Section 2.1), as follows:

```
1 $C1 := DICE (<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD>,
2       (timeDim.refPeriod.yearNumber >= 2010) AND
3       (timeDim.refPeriod.yearNumber <= 2012) );
4 $C2 := ROLLUP ($C1, geoDim, geoDim.income);
```

In the syntax above, we use the notation *dimension.level.attribute*, to represent the dimension's structure. Also, to be concise, we omitted the F_{agg} parameter in the expression for `ROLLUP` (as indicated in Section 2.1), because we assume it is the only one defined for this cube in the DSD. Finally, the variables $C1$ and $C2$ store the results of the operations in the right hand sides of the expressions. With the help of QB4OLAP metadata (which, e.g., describes the hierarchical structure), this query can be automatically translated to the following SPARQL expression:

```
SELECT ?year ?income SUM(xsd:integer(?measureValue)) AS ?sumMeas
FROM <http://www.fing.edu.uy/inco/cubes/schemas/wbld>
FROM <http://www.fing.edu.uy/inco/cubes/instances/wbld>
WHERE {
SERVICE <http://worldbank.270a.info/sparql>
    {?o a qb:Observation ;
        qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD>;
        sdmx-dimension:refArea ?country;
        sdmx-dimension:refPeriod ?year;
        sdmx-measure:obsValue ?measureValue.
        }
    ?country skos:broader ?income.
    ?income qb4o:memberOf schema:income.
    ?year schema:yearNumber ?yearNum
    FILTER (?yearNum >= 2010 && ?yearNum <= 2012 )
}
GROUP BY ?year ?income
```

4 Automating Metadata Definition

Considering that QB4OLAP brings benefits in terms of additional schema constructs that are necessary for state-of-the-art OLAP analysis, we next discuss the possibilities for introducing these enhancements into existing QB data sets. Currently, a considerable number of data sets are published in QB. Thus, in this section we elaborate on how to define and/or discover new concepts (e.g., dimension levels) that can be used for enriching existing QB data sets. As this can be a very cumbersome, error-prone, and labor-intensive task, we especially focus on its maximal possible automation such that it involves the least possible user intervention. To achieve that, we take advantage of the semantics that is explicitly or implicitly present in the data set or in external data sources. Once discovered, these concepts are used for the enrichment method explained in the next section. In this section, we discuss the automation challenges and how far they can be solved in a semi-automatic way, followed by solutions that we propose for the two most relevant tasks, namely *the definition of aggregate functions* and *the discovery of the dimension hierarchy schema and instances*.

4.1 Automation Challenges

The enrichment tasks needed to turn a QB into a QB4OLAP data set must be done at the *metadata* (e.g., the cube schema concepts) and *data* (e.g., the cube instances) levels. Simply stated, we need to build the hierarchical structure of the dimensions involved (i.e., metadata enrichment) and populate this structure with actual data (i.e., data enrichment). Further, the enrichment includes associating aggregate functions with measures which is also a metadata-related task.

Exploiting the existing QB semantics (i.e., metadata) and the analysis of the data set instances (i.e., data) enable the automatic discovery of potentially new metadata concepts (e.g., new dimension levels). These metadata concepts can be suggested to the user that needs to select the concepts of her interests and, if needed, provide the minimum possible input about missing semantics and specific situations (e.g., data conflicts). The new metadata then support the rest of the enrichment process. For instance, in Example 4 we can see that a `country` is related to a `region` via the `property:region` property. When this property is identified as a parent-child relationship between two levels in a dimension, the rollup instances can be automatically created for all `country` and `region` instances (using the `skos:broader` property).

Performing OLAP analysis directly over SW data in the RDF and Linked Data settings is likely to bring certain challenges. This is due to the fact that, unlike in the traditional DWs settings where data are prepared by a well-defined and complex ETL process, the Linked Data settings do not guarantee clean and formatted data. On the contrary, working in a Linked Data environment typically involves external data sources where it is not rare to find incomplete and imperfect data. These problems directly influence the automation possibilities and user involvement is typically required to fully enrich a data set. The user needs to choose and/or to add semantic information to the data set, or to manage problems that can be present in the data sets. The less these situations occur, the higher level of automation can be achieved and vice versa. Next, we describe these challenges starting with the semantic-related challenges – partial and imperfect semantics – and then we address data-related challenges–partial and imperfect data.

Partial semantics This challenge arises when the data set does not contain enough schema information for the enrichment tasks. For instance, a data set may lack information about the aggregate functions that can be applied over measures or the semantics for building the dimension hierarchies.

This challenge is the most relevant for our approach and needs to be tackled. The problem can be addressed either by enrichment from external sources, or by manual user intervention. In the next subsections we discuss two possible approaches for defining the additional semantics needed for

enrichment tasks.

Imperfect semantics This challenge comprises different cases that occur when semantics obstructs automation. For instance, when the same semantics is represented with different concepts, the same concepts have different semantics, and other conflicts that may arise in schema comparison [52]. This is called *semantic heterogeneity* [43]. For example, we may have different currencies, different measure units, etc. As other cases of imperfect semantics, we can mention incorrectly defined semantics (e.g., when a continent is located in a city), and outliers, i.e., unexpected semantic concepts that cannot be aligned with the rest (e.g., cantons as a geographical concept that is not used in most of the cases).

This problem should be addressed by detecting the potential deviations in semantics (e.g., the same concept playing different roles) and enabling the user to address these cases. These situations are typically solved in the data set cleaning and transformation stages [52] while our approach focuses on semantic enrichment.

Partial data Partial data may affect aggregation, a key task in OLAP analysis. Missing data may raise many challenges. For example, aggregating data to the continent level depends on the availability of data about all the belonging countries.

To tackle this problem, missing data can possibly be imported from external sources. However, since it is often the case that the data set in hand contains the only available data (e.g., only some countries in the EU are covered), we focus on constructing the cube on top of the *available pieces of data*. In the case that it is possible to detect the problems caused by missing data, the user should be notified (e.g., marking the aggregated values as incomplete/partial in case of missing values).

Imperfect data Many different cases can illustrate this problem, where data instances cause difficulties to achieve automation. One of these cases is *data heterogeneity* where not all data are well formatted or do not satisfy explicit or implicit constraints. For example, in RDF it is impossible to impose data instances to satisfy MD integrity constraints (see Section 4.3). Imperfect data may also include data errors and data instances that do not satisfy constraints but still represent correct information.

This challenge generates a wide spectrum of cases and we focus on *cardinality detection* based on data analysis. Other cases, like the detection of the data instances that do not match their type(s) (e.g., instead of an expected integer we find a string), out-of-range values, and similar cases should be detected and reported to the user. Then, the user can discard or, if possible,

fix these cases. In this context, an extensive overview of methodologies for data quality assessment and improvement can be found in [16].

4.2 Associating measures with aggregate functions

To enable automatic navigation along dimension hierarchies, each measure in the data cube needs to have an associated aggregate function. Since QB does not allow to provide this information, the QB data set must be enriched with a mapping of measures to aggregate functions. We call this mapping *MAggMap*. Not every aggregate function can be applied to a measure and give a valid result. Defining the appropriate aggregate function depending on the measure type is a well-known problem in the literature related to the summarizability problem in OLAP and statistical databases [69]. In that context, measure types are flow (e.g., monthly sales value), stock (e.g., inventory of a product), and value-per-unit (e.g., product item price). For instance, while it makes sense to compute the sum of the monthly sales by year, it does not make sense to sum a product's unit price over time. This summarizability condition is called *type compatibility*, i.e., the compatibility between the measure type (i.e., its semantics), the measure category (i.e., temporal or non-temporal), and the aggregate function's type. This condition, together with *disjointness* and *completeness* (see next section), is necessary to guarantee correct data summarization [69].

The large variety of measure and aggregate function types makes the compatibility check a tedious task that can hardly be fully automated. Even in such a case, the user would still need to choose among different options. Therefore, the user must be involved in this process. However, this involvement can be guided and semi-automated based on the compatibility definition presented in [69]. We address the interested reader to [104] for further details.

In this paper, we assume that the user explicitly provides the *MAggMap* mapping, which is a needed input for our enrichment tasks (see Section 5). We define it as [measure IRI, aggregate function IRI] pairs. For example, [`sdmx-measure:obsValue`, `qb4o:Sum`]. In our current implementation (see Section 6), we suggest a default aggregate function (e.g., sum) but it can be changed by the user.

4.3 Discovering dimensional data

Dimensional concepts consist of dimensions, hierarchies, levels, and level attributes. Briefly, dimensions contain different levels of aggregation (i.e., dimension levels), which are organized in hierarchies (in short, each hierarchy corresponds to a path of rollup relationships) and may contain attributes (see Section 2.1). Enriching a QB data set with dimensional data implies properly

identifying all these constructs and annotating them according to QB4OLAP. Current OLAP state-of-the-art identifies dimensional concepts from functional dependencies (FDs) [37]. Arranging the dimensional concepts according to FDs guarantees the summarizability disjointness and completeness and these are necessary conditions to guarantee the summarizability correctness. Accordingly, FDs must be guaranteed between facts and dimensions (e.g., between market capitalization and geographical dimension) and between the levels forming dimension hierarchies (e.g., between the `country` and `region` levels). [87] discusses the role of FDs for automatic MD modeling and how to discover them for Description Logics (DL). To discover FDs, the most widespread technique consists of sampling data to identify functional properties that fulfill the underlying many-to-one[20] cardinality of the relationship. Briefly, many-to-one (i.e., m:1) cardinalities require that every child level instance is related to one parent level instance (e.g., `Serbia` is related to the `Europe & Central Asia (ECS)` region), while each parent level instance can be related to one or more child level instances and these sets (e.g., countries and regions) do not mutually overlap [73]. These guarantee completeness and disjointness. A comprehensive and detailed overview of the summarizability challenges in MD modeling is presented in [73]. Dimension hierarchies whose properties satisfy many-to-one cardinalities guarantee a correct summarizability as the aggregate values at parent levels (e.g., region) include all related child level instances (e.g., countries) and no parent level instance is without child level instance(s).[21] Furthermore, there is no double-counting of child level instances at the parent levels. Many-to-one cardinalities enable the automation of the MD design [86] where the potential new levels can be discovered by detecting these cases in data instances.

In some expressive languages, such as the OWL 2 RL profile[22] based on DL-Lite [13], it is possible to state that a property is functional. However, most available RDF data sets omit such definitions. Therefore, in the spirit of [86], we analyze the instances to identify FDs from data. To avoid the inherent computational complexity discussed in [54], we benefit from the QB semantics by considering the QB dimensions as the initial set of dimension levels from which to start building richer dimensional structures. Thus, the QB dimensions serve as the starting point from where to discover possible new dimension levels, hierarchies, and level attributes based on FDs. Given the relevance of this step for MD modeling, we provide an algorithm for the detection of implicit FDs by discovering functional properties (i.e., satisfying a many-to-one cardinality) for dimension levels. Moreover, one-to-one (i.e., 1:1) cardinalities are identified to detect potential dimension level

[20]Note that "many" stands for "one or more instances".

[21]Note that we do not consider the special case of non-covering dimensions where there could exist parent level instances with no child level instances.

[22]`https://www.w3.org/TR/2008/WD-owl2-profiles-20081008/#OWL_2_RL`

attributes. We only consider linear hierarchies since, in practice, complex hierarchies (see [81] for more details) with many-to-many cardinalities are typically transformed to linear hierarchies with many-to-one cardinalities. The pseudo code is presented in Algorithm 1. The algorithm runs over an implicit RDF graph.

Algorithm 1: Detect implicit MD semantics

```
   Input: L, minCompl, minCard; // initial levels set (i.e., former QB dimensions), minimum
          completeness, and minimum cardinality parameters, respectively
   Output: allL, allP, allHS, allLLA; // all levels, all properties, all hierarchy steps, and
           all level-level attribute pairs, respectively
1  begin
2      allL = L;
3      allP = ∅;
4      allHS = ∅;
5      allLLA = ∅;
6      foreach level ∈ allL following a bottom-up order do
7          foreach property ∈ getProperties(level) do
8              if getCardinality(level, property, minCompl, minCard) = m : 1 then
9                  parentLevel = getObjectElement(level, property);
10                 if noCycles(level, parentLevel) then
11                     allL ∪= parentLevel;
12                     allP ∪= property;
13                     allHS ∪= (level, property, parentLevel);
14             else if getCardinality(level, property, minCompl, minCard) = 1 : 1 then
15                 levelAttribute = getObjectElement(level, property);
16                 allLLA ∪= (level, levelAttribute);
```

The algorithm starts from the set of original QB data set *dimensions* (L), which from now on we call *initial levels*. Moreover, it also takes the *minCompl* and *minCard* parameters, which are later discussed in Algorithm 2. The output of the algorithm are the sets of all levels ($allL$), rollup properties ($allP$), all hierarchy steps ($allHS$), and all [level, level attribute] pairs ($allLLA$) available in the input QB data set. The set of all levels is initially populated with the initial levels set (line 2), while the other sets are initially empty (see lines 3 to 5). For each level (line 6), e.g., the `country` level, we check all of its properties (line 7), e.g., the region property, and infer their cardinalities. We iterate over the levels by following a bottom-up approach; i.e., we start from the finer (e.g., the `country` level) and later visit coarser granularity levels (e.g., the `region` level). Details on how to retrieve the property cardinality are shown in Algorithm 2. If a property yields a *many-to-one* cardinality (line 8) its object (i.e., the RDF property range) is considered as a potential coarser granularity level to rollup to. Therefore, a potential new *parent* level is retrieved in line 9. Importantly, to guarantee the MD integrity constraints, before adding this new parent level to the set of all levels, we check that this addition does not produce cycles (line 10), i.e., that the current level cannot be reached from the newly identified parent level (e.g., that there is no direct or indirect rollup

relationship from `region` to `country`). If there are no cycles, we add the new parent level to the set of all levels (line 11). Then, in lines 12 and 13, the property and the hierarchy step triples are added to the corresponding sets. Otherwise, if the property cardinality is one-to-one (line 14), the new concept is considered as a level attribute (e.g., `label`), and it is added to the set of [level, level attribute] pairs (lines 15 and 16). The output sets of Algorithm 1 are later consumed as inputs in our enrichment tasks (see Section 5).

Algorithm 2: Get cardinality for a property

```
   Input: l, p, minCompl, minCard;      // level, property, minimum completeness, and minimum
          cardinality parameters
   Output: cardinality;                 // cardinality of the property p for the level l
1  begin
2      to − oneFromChild = 0;      // number of to-one property instances from the child side
3      to − oneFromParent = 0;     // number of to-one property instances from the parent side
4      to − manyFromParentSet = ∅;     // set of parent level instances for to-many property
       instances from the parent side
5      foreach li ∈ getInstances(l);                                   // li - level instance
6      do
7          if countSubjectPropertyInstances(li, p) = 1 then
8              to − oneFromChild ++;
9              parentLI = getObjectElement(li, p);
10             if countObjectPropertyInstances(parentLI, p) = 1 then
11                 to − oneFromParent ++;
12             else if countObjectPropertyInstances(parentLI, p) > 1 then
13                 to − manyFromParentSet ∪= parentLI ;
14     if to − oneFromChild ≥ getInstanceNumber(l) * minCompl then
15         if to − oneFromChild / minCard ≥ to − manyFromParentSet.Size() then
16             cardinality = m : 1
17         else if to − oneFromParent ≥ getInstanceNumber(l) * minCompl then
18             cardinality = 1 : 1
19     else
20         cardinality = m : m;                                  // other cardinality value
```

Algorithm 2 determines a property cardinality using simple SPARQL queries to retrieve the number of property instances related to a subject (line 7) or an object (lines 10 and 12). This algorithm takes as input the level (e.g., the `country` level) and the property (e.g., the region property) for which it must retrieve the cardinality. Moreover, it also needs the minimum completeness and disjointness *minCompl* and the minimum cardinality *minCard* parameters as inputs. The former defines the minimal required percentage of $to-one$ relationships for the total number of level instances, e.g., a value 0.90 means that at least 90% of countries need to have one and only one region property. This way, there might be some level instances that have none or more than one property instances. Although non-complete / non-disjoint properties stand against the conditions discussed earlier in the present section, this is needed to identify conceptual FDs that, due to imperfect and /or

partial data, do not hold for all the data. Following the idea presented in [96], by means of these two parameters we identify *quasi-FDs* (which is often the case in Linked Data and RDF data sets). We say that a property is a *quasi-FD* if most of the data satisfy the FD (e.g., 98% of level instances are associated to exactly one property instance). The second parameter, *minCard*, defines the minimum average number of child level instances per parent level instance (e.g., $minCard = 5$ meaning at least 5 countries per region). The values for these parameters should be empirically defined depending on the domain and data set quality (see Section 6).

The algorithm proceeds as follows. The local variable $to-oneFromChild$ (line 2) holds the number of $to-one$ properties from child to parent instances (e.g., the number of cases where there is only one region property per `country`); analogously, $to-oneFromParent$ (line 3) holds the number of $to-one$ properties from parent to child instances (e.g., the number of cases where for a `region` instance there is only one region property from a `country` instance to that `region` instance), and $to-manyFromParentSet$ (line 4) holds the set of parent instances that are in $to-many$ relationships (e.g., the `region` instances that are related to more than one `country` via the region property). For all instances of a given level (line 6), e.g., all `country` instances, we count the ones that have only one instance of a given property (line 8), e.g., the region property. In this case, the algorithm retrieves the level instance on the other side of the property (e.g., the `region` instance) and checks its cardinality (lines 10 and 12). Note that this check differs from the first one (line 7), since here the level instance (e.g., the `region` instance) is used as a property object while in the first one, the input level instance is used as a subject. In case of $to-one$ property instances (lines 10 and 11), we count them; in case of $to-many$ instances, we add them to the set (lines 12 and 13) so that we can count them at the end (line 15), since the property instances will be repeated for child instances with the same parent instance (e.g., several `country` instances are related to the same `region` instance). Finally, we determine the cardinality in lines 14 – 20.

We next show how Algorithm 2 can be implemented with the following SPARQL queries. We consider that the queries use an RDF graph that contains a QB4OLAP level (e.g., `?levelIRI? a qb4o:LevelProperty`) and a set of QB4OLAP level members (e.g., `levelMemberIRI1 a qb4o:LevelMember`) belonging to this level (i.e., `levelMemberIRI1 qb4o:memberOf ?levelIRI?`). Furthermore, we use the following parameter values $minCompl = 100$ and $minCard = 2$. Hence, all properties for the level members can be retrieved with Query 1. The query takes the graph and level IRIs as parameters. Note that the elements between two '?' in the queries represent parameters that should be replaced with the corresponding IRI values and the prefixes used in queries are following:

```
prefix qb: <http://purl.org/linked-data/cube#>
prefix qb4o: <http://purl.org/qb4olap/cubes#>
```

Query 1
Get properties for level members.

```
# INPUT: ?qb4oGraphIRI? - the new QB4OLAP graph IRI, and
# ?levelIRI? - the level IRI

SELECT DISTINCT ?p
FROM ?qb4oGraphIRI?
WHERE {
 ?levelMember ?p ?o .
{ SELECT DISTINCT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember a qb4o:LevelMember .
  ?levelMember qb4o:memberOf ?levelIRI? . } } }
```

For a chosen property, we first need to check if it is a $to-one$ property, i.e., each level member is related to one and only one instance of the property. Query 2 performs this check. In addition to the previous ones, the query also takes the property IRI as parameter.

Query 2
Check if the property is to-one.

```
# INPUT: ?qb4oGraphIRI? - the new QB4OLAP graph IRI,
# ?levelIRI? - the level IRI, and
# ?propertyIRI? - the property IRI

ASK { {
SELECT (COUNT (?levelMember) AS ?lmWithUniqueObject )
FROM ?qb4oGraphIRI?
WHERE { { #get #unique object per level member for the input property
 SELECT ?levelMember (COUNT (DISTINCT ?obj) AS ?uniqueObjNum)
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember ?propertyIRI? ?obj .
  {SELECT ?levelMember
  FROM ?qb4oGraphIRI?
  WHERE {
   ?levelMember a qb4o:LevelMember .
   ?levelMember qb4o:memberOf ?levelIRI? .}}
 } GROUP BY ?levelMember }
 FILTER ( ?uniqueObjNum = 1) } }
 { #get the total #level members for a level
 SELECT (COUNT (DISTINCT ?lm) AS ?totalLevelMemberNumber)
 FROM ?qb4oGraphIRI?
 WHERE {
  ?lm a qb4o:LevelMember .
  ?lm qb4o:memberOf ?levelIRI? . } }
 FILTER (?lmWithUniqueObject = ?totalLevelMemberNumber) }
```

If Query 2 returns true, the property is $to-one$ and we can check if it is 1:1 or m:1 with Queries 3 and 4, respectively. The parameters are the same as for the previous query.

Query 3
Check if the property is 1:1.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?levelIRI? – the level IRI, and
# ?propertyIRI? – the property IRI

ASK { { #get the #object per level member for the input property
SELECT (COUNT (DISTINCT ?levelMember) AS ?totalLevelMemberNumber)
(COUNT (DISTINCT ?obj) AS ?uniqueObjNum)
FROM ?qb4oGraphIRI?
WHERE {
 {SELECT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember a qb4o:LevelMember .
  ?levelMember qb4o:memberOf ?levelIRI? . } }
 ?levelMember ?propertyIRI? ?obj . } }
 FILTER (?uniqueObjNum = ?totalLevelMemberNumber) }
```

Query 4
Check if the property is m:1.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?levelIRI? – the level IRI, and
# ?propertyIRI? – the property IRI

ASK { { #get the #object per level member for the input property
SELECT (COUNT (DISTINCT ?levelMember) AS ?totalLevelMemberNumber)
(COUNT (DISTINCT ?obj) AS ?uniqueObjNum)
FROM ?qb4oGraphIRI?
WHERE {
 {SELECT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember a qb4o:LevelMember .
  ?levelMember qb4o:memberOf ?levelIRI? . } }
 ?levelMember ?propertyIRI? ?obj . } }
FILTER (?uniqueObjNum < ?totalLevelMemberNumber/2)
{ #check that objects are not literals
SELECT (COUNT (DISTINCT ?obj2) AS ?notLiteralObj)
FROM ?qb4oGraphIRI?
WHERE {
 {SELECT ?lm
 FROM ?qb4oGraphIRI?
 WHERE {
  ?lm a qb4o:LevelMember .
  ?lm qb4o:memberOf ?levelIRI? . } }
  ?lm ?propertyIRI? ?obj2 .
 FILTER isIRI(?obj2) } }
FILTER (?uniqueObjNum = ?notLiteralObj) }
```

Our algorithms consider settings where the input QB data set contains implicit MD semantics, i.e., where the levels have properties that link them with coarser granularity levels inside the data set. If this is not the case, we can use existing IRIs or look for external IRIs (e.g., the IRI for Serbia on DBpedia[23]) to search for the necessary semantics from external data sets.

[23] `http://dbpedia.org`

If this is not possible, the user should define these IRIs manually. Further, we assume that in the input QB data set, all observations are at the same level of granularity for each dimension which is the case most of the time. Then, on top of these levels we build new dimension hierarchies. Special situations, where there might exist observations at different granularities, must be treated manually in a data preparation step. This situation can be detected with Algorithm 1 if it identifies a rollup property between instances of an initial level.

5 Enrichment Method

Taking advantage of the QB4OLAP vocabulary and the algorithms introduced in Section 4.3, we now propose a method to enrich an input QB graph[24] with additional MD semantics. This method presents a set of detailed enrichment steps. For the sake of comprehension, each step is described as a SPARQL query showing the precise enrichment and transformations. The queries take the specified parameters, use an input QB graph and incrementally create the new QB4OLAP graph by generating the necessary triples. Since this method requires some user actions, the overall enrichment process is semi-automatized. The method consists of two phases:

1. *Redefinition phase* which syntactically transforms the input QB graph into QB4OLAP constructs and, given the required input (see Section 4.2), specifies aggregate functions for measures.
2. *Enrichment phase* which, given a set of required inputs (see Section 4.3), enriches the QB4OLAP graph generated by the redefinition phase with additional MD semantics.

For the ease of understanding, this section introduces the main ideas for the enrichment tasks to be accomplished. In addition, 1 provides a fully formalized, more general, and detailed enrichment methodology, which is agnostic of the implementation decisions made and further specifies the preconditions, post-conditions, and transformations to be conducted by each step in terms of set theory. Thus, the method presented in this section can be considered as a possible solution to cover the steps defined by the methodology. In this section, we first introduce some preliminaries for understanding the method. Next, each phase is defined in terms of queries to be performed that taking the input parameters produce the output triples. Finally, we provide some additional considerations.

[24]For simplicity of presentation, we assume that all triples related to the input QB data set are in a single RDF graph.

5.1 Method Preliminaries

The method uses two RDF graphs, namely the *QB graph* (i.e., the set of triples defining the QB cube structure and instances) and the *QB4OLAP graph* (analogous to the QB graph definition). These graphs are assumed to be compliant with the QB and QB4OLAP vocabularies, respectively. According to the QB and QB4OLAP definitions, we further identify two sets of RDF triples in each graph: the set of triples describing the QB or QB4OLAP *cube schema* and the set describing the *cube instances*.

According to the QB definition (see Section 2), the *QB cube schema* consists of the triples involving the following classes and related properties: the QB *dataset*[25] (i.e., `qb:DataSet`), *structure* (i.e., `qb:DataStructureDefinition`), *dimensions* (i.e., `qb:DimensionProperty`), and *measures* (i.e., `qb:MeasureProperty`). Following QB's notation, the cube structure is defined as a set of dimensions and measures via the cube components (i.e., `qb:ComponentSpecification`). An example of QB cube schema extracted from our running example (see Section 2.3) is presented in Example 9.

Example 9
QB cube schema triples.

```
1 <http://worldbank.270a.info/dataset/world-bank-indicators/structure>
2  a qb:DataStructureDefinition ;
3  qb:component  [ qb:dimension sdmx-dimension:refArea ] ;
4  qb:component [ qb:measure sdmx-measure:obsValue ] .
5 sdmx-dimension:refArea a qb:DimensionProperty .
6 sdmx-measure:obsValue a qb:MeasureProperty .
7 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet ;
8  qb:structure
9     <http://worldbank.270a.info/dataset/world-bank-indicators/structure> .
```

Lines 1 – 4 relate to the QB cube structure, line 5 to dimensions, line 6 to measures, and line 7 to the dataset. The dataset is related to the cube structure in lines 8 – 9.

QB cube instances contain triples related to the QB *dimension instances* (extracted from the observations with Query 9 as explained later) and *observations* (i.e., `qb:Observation`). As discussed before, observations represent measure values for the fixed dimension instances determined by the cube structure. An example of QB cube instances is presented in Example 10.

Example 10
QB cube instance triples.

```
1 data:world-bank-indicators/CM.MKT.LCAP.CD/RS/2012
2   a qb:Observation ;
3   sdmx-dimension:refArea country:RS ;
4   sdmx-measure:obsValue 7450560827.04874 ;
5   qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> .
```

[25]Note that in the present section the term "dataset" refers to qb:DataSet.

Line 1 – 2 define an observation. Line 3 specifies a dimension instance and line 4 defines a measure value of the observation. The observation relates to the cube schema structure indirectly (see lines 7 – 9 of Example 9) via `qb:dataSet` in line 5.

Analogously, we next define the QB4OLAP cube schema and instances. The *QB4OLAP cube schema* consists of the triples involving the following classes and related properties:

- *dataset* (i.e., `qb:DataSet`),
- *structure* (i.e., `qb:DataStructureDefinition`),
- *dimensions* (i.e., `qb:DimensionProperty`),
- *measures* (i.e., `qb:MeasureProperty`),
- dimension *levels* (i.e., `qb4o:LevelProperty`),
- dimension *level attributes* (i.e., `qb4o:LevelAttribute`),
- dimension *hierarchies* (i.e., `qb4o:Hierarchy`),
- *hierarchy steps* (i.e., `qb4o:HierarchyStep`),
- predefined set of *aggregate functions* (i.e., `qb4o:AggregateFunction`), and
- predefined set of possible *cardinalities* (i.e., `qb4o:Cardinality`).

The *QB4OLAP cube instances* contain the triples related to the QB4OLAP cube *level instances*, *rollup relationships* between child and parent level instances (represented with `skos:broader`), *observations* (i.e., `qb:Observation`), and *level attribute values* (being literals or IRIs).

Examples of the QB4OLAP cube schema and instances are presented below in the method definition. The examples are based on the running example (see Section 2.3). We consider the scenario where the input graph contains implicit MD semantics (e.g., a country is linked to a region but this is not explicitly stated as a rollup relationship since this cannot be described in QB). Other scenarios are discussed in Section 5.4. For the sake of simplicity, we define the steps as SPARQL INSERT queries assuming that from the original input QB graph we build a *new* QB4OLAP graph (which is implicitly created with the first SPARQL INSERT query). Note that the elements between two '?' in the queries represent parameters that should be replaced with the IRI values specified at each step. Moreover, all the examples of the query results follow up on one another. In addition to the prefixes introduced in the previous section, the queries also use the following prefix:

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

5.2 Redefinition Phase

Redefinition of a cube schema. We start by building the new QB4OLAP cube schema. We proceed incrementally and first we perform a syntactic transformation from QB to QB4OLAP constructs, while the complete QB4OLAP

cube schema is formed after the Enrichment phase (see Section 5.3). The QB cube schema triples defining the cube dataset, structure, dimension levels, and measures are added to the QB4OLAP cube schema in the following way. First, dimensions are redefined as levels in the QB4OLAP cube schema. Next, we copy the measures from the QB graph to the QB4OLAP one. Then, we define the new cube schema structure, assign to it both levels and measures, and add it to the QB4OLAP cube structure. Finally, we copy the dataset definition to the QB4OLAP cube schema and assign the new cube schema structure to it. This transformation can be performed with Query 5. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the QB dataset IRI, and the new QB4OLAP structure IRI.

Query 5
Redefinition of a cube schema.

```
1  # INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
2  #?qbGraphIRI? – the input QB graph IRI,
3  #?dsIRI? – the dataset IRI, and
4  #?dsdIRI? – the data structure definition IRI
5
6  INSERT INTO ?qb4oGraphIRI? {
7  ?dsIRI? a qb:DataSet .
8  ?dsIRI? qb:structure ?dsdIRI? .
9  ?dsdIRI? a qb:DataStructureDefinition .
10 ?dsdIRI? qb:component ?bl . ?bl qb4o:level ?d .
11 ?dsdIRI? qb:component ?bm . ?bm qb:measure ?m .
12 ?d a qb4o:LevelProperty .
13 ?m a qb:MeasureProperty . }
14 FROM ?qbGraphIRI?
15 WHERE {
16 ?dsd a qb:DataStructureDefinition .
17 ?dsIRI? qb:structure ?dsd .
18 ?dsd qb:component  ?bl . ?bl qb:dimension ?d .
19 ?dsd qb:component  ?bm . ?bm qb:measure ?m . }
```

Thus, at this point, we have obtained the initial QB4OLAP cube schema. An example of a QB4OLAP cube schema is shown in Example 11 which illustrates the result of Query 5. Note that we use the `newG` namespace for the new QB4OLAP graph.

Example 11
Resulting triples of Query 5.

```
1  <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
2                                        qb:structure newG:newDSD .
3  newG:newDSD a qb:DataStructureDefinition ;
4    qb:component  [ qb4o:level sdmx-dimension:refArea ] ;
5    qb:component  [ qb:measure sdmx-measure:obsValue ] .
6  sdmx-dimension:refArea a qb4o:LevelProperty .
7  sdmx-measure:obsValue a qb:MeasureProperty .
```

Lines 1 and 2 illustrate the triples related to the cube dataset. Results in lines 3, 4, and 5 define the new cube schema structure and add a level and a measure as components

to it. Line 6 redefines the dimension from Example 9 as a QB4OLAP level, while line 7 illustrates the measure from Example 9 copied to the new QB4OLAP graph.

Specification of an aggregate function. Next, we need to specify an aggregate function per measure. Note that possible aggregate functions are predefined by QB4OLAP. The inputs for this task are the QB4OLAP graph IRI, the QB dataset IRI, and the *MAggMap* mapping; i.e., the [measure IRI, aggregate function IRI] pair (see Section 4.2). The aggregate function is specified as a triple that relates the aggregate function IRI with the component of the cube schema structure related to the measure. This triple is added to the QB4OLAP cube schema and it can be performed with Query 6. In case of more than one measure, the query should be run for each measure.

Query 6
Specification of an aggregate function.

```
1  # INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
2  # ?dsIRI? – the dataset IRI, and
3  # ?measureIRI?
4  # ?aggregateFunctionIRI? – the aggregate function IRI
5
6  INSERT INTO ?qb4oGraphIRI? {
7  ?comp qb4o:aggregateFunction ?aggregateFunctionIRI? }
8  FROM ?qb4oGraphIRI?
9  WHERE {
10 ?dsd  a  qb:DataStructureDefinition .
11 ?dsIRI? qb:structure  ?dsd .
12 ?dsd  qb:component  ?comp .
13 ?comp qb:measure ?measureIRI? . }
```

An example of the updated QB4OLAP cube schema is presented in Example 12.

Example 12
Resulting triples of Query 6.

```
1  <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
2  <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> qb:structure newG:newDSD .
3  newG:newDSD a qb:DataStructureDefinition ;
4    qb:component  [ qb4o:level sdmx-dimension:refArea ] ;
5          qb:component  [ qb:measure sdmx-measure:obsValue ;
6                          qb4o:aggregateFunction qb4o:Sum ] .
7  sdmx-dimension:refArea a qb4o:LevelProperty .
8  sdmx-measure:obsValue a qb:MeasureProperty .
```

Line 6 presents the aggregate function that is assigned to a measure by the grouping mechanism via a blank node. In this case, the SUM (i.e., `qb4o:Sum`) aggregate function.

Definition of a dimension. As part of the automatic redefinition, to build QB4OLAP-compliant dimension hierarchies, a new dimension for each initial level needs to be defined. As explained in Section 3, QB4OLAP reuses the `qb:DimensionProperty`, however with different semantics than in QB: while

in the latter a dimension represents a point at a fixed granularity, QB4OLAP considers a dimension to contain points at different level granularities. Therefore, in QB4OLAP, a QB dimension becomes a dimension level (see Query 5) and a dimension represents a set of levels that are hierarchically organized. The inputs for this task are the QB4OLAP graph IRI and the dimension IRI, and it can be performed with Query 7 that should be run for each dimension.

Query 7
Definition of a dimension.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI and
# ?dimensionIRI? – the dimension IRI

INSERT INTO ?qb4oGraphIRI? {
?dimensionIRI? a qb:DimensionProperty . }
```

An example of the triple added to the updated QB4OLAP cube schema is presented in Example 13.

Example 13
Resulting triple of Query 7.

```
newG:geoDimension a qb:DimensionProperty .
```

The triple presents a dimension for the `sdmx-dimension:refArea` initial level.

Definition of a hierarchy. Once the dimensions are created, we need to create a hierarchy for each dimension. A hierarchy represents the set of hierarchically ordered levels in the dimension. Once created, it needs to be linked with the corresponding dimension and the initial level. Thus, the inputs for this task are the QB4OLAP graph IRI, the hierarchy IRI, the dimension IRI, and the level IRI. This can be performed with Query 8 that should be run for each hierarchy.

Query 8
Definition of a hierarchy.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?hierarchyIRI? – the hierarchy IRI,
# ?dimensionIRI? – the dimension IRI, and
# ?levelIRI? – the level IRI

INSERT INTO ?qb4oGraphIRI? {
?hierarchyIRI? a qb4o:Hierarchy .
?dimensionIRI? qb4o:hasHierarchy ?hierarchyIRI? .
?hierarchyIRI? qb4o:inDimension ?dimensionIRI? .
?hierarchyIRI? qb4o:hasLevel ?levelIRI? . }
```

An example of the triples added to the updated QB4OLAP cube schema is presented in Example 14.

Example 14
Resulting triples of Query 8.

```
1 newG:geoHierarchy a qb4o:Hierarchy .
2 newG:geoDimension qb4o:hasHierarchy newG:geoHierarchy .
3 newG:geoHierarchy qb4o:inDimension newG:geoDimension .
4 newG:geoHierarchy qb4o:hasLevel newG:region .
```

Line 1 illustrates a new hierarchy being created. Triples in lines 2 and 3 link the hierarchy with a dimension, and to a level in line 4.

Populating level members of an initial level. Finally, at the end of the redefinition phase, we populate level members for the initial levels of the QB4OLAP graph schema. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the level IRI, and the QB dataset IRI. This can be performed with Query 9 that should be run for each initial level.

Query 9
Populating level members of an initial level.

```
1  # INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
2  # ?qbGraphIRI? – the input QB graph IRI,
3  # ?levelIRI? – the level IRI, and
4  # ?dsIRI? – the dataset IRI
5
6  INSERT INTO ?qb4oGraphIRI? {
7  ?levelMember a qb4o:LevelMember .
8  ?levelMember qb4o:memberOf ?levelIRI? . }
9  FROM ?qbGraphIRI?
10 WHERE { {
11  SELECT DISTINCT ?levelMember WHERE {
12    ?o a qb:Observation .
13    ?o qb:dataSet ?dsIRI? .
14    ?o ?levelIRI? ?levelMember . } } }
```

An example of level member triples added to the QB4OLAP graph instances is presented in Example 15.

Example 15
Resulting triples of Query 9.

```
1 country:RS a qb4o:LevelMember .
2 country:RS qb4o:memberOf sdmx-dimension:refArea .
```

Line 1 illustrates a level member extracted from the observations and line 2 links it to the level it belongs to.

5.3 Enrichment Phase

Once the cube schema is redefined in terms of QB4OLAP, we next focus on its enrichment with new dimensional concepts to construct richer hierarchies. At this point, we assume that a pre-process to discover potential new levels and level attributes has been carried out. For example, this could be done

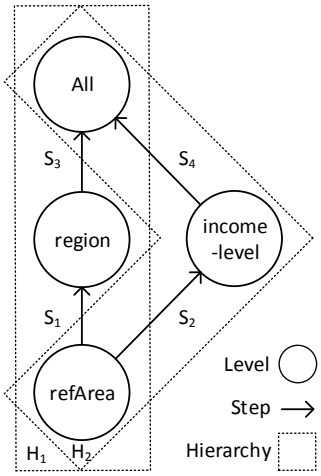

Fig. 4.4: Three-level Hierarchies Construction

with the algorithms proposed in Section 4.3. Starting from an initial dimension level we explain the construction of two three-level hierarchies. This scenario is illustrated in Figure 4.4. Starting from the `refArea` level that belongs to the hierarchy H1 we first add a new level - `region`. For this we need to add the `region` level to the hierarchy H1, create the hierarchy step S1, add both levels to S1 (`refArea` as child and `region` as parent), and add S1 to H1. Then, we add one more level, `income-level`, on top of `refArea`. As conceptually the new level belongs to a new hierarchy (i.e., it does not belong to the same rollup path as `region`), we need to create a new hierarchy H2, create a new step S2, add `refArea` (as child) and `income-level` (as parent) to S2, and add S2 to H2. Moreover, H2 needs to be added to the same dimension that H1 already belongs to. This notation is fixed by QB4OLAP. Finally, we create the mandatory `All` level for the dimension and accordingly link the `region` and `income-level` levels to it via two new hierarchy steps S3 and S4 that are created and added to H1 and H2, respectively. The process is as follows.

Creating, populating, and linking a new parent level. First, we need to create a new level (e.g., `region`) and add it as a parent level to the child level (e.g., `refArea`). Moreover, we need to link all the members of the child level with the corresponding members of the new parent level. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the child level IRI, and the new parent level (i.e., property) IRI. This can be performed with Query 10. To build the hierarchy illustrated in Figure 4.4, the query should be run for both `region` and `income-level` levels that are added as parent levels for the `refArea` level.

Query 10
Creating, populating, and linking a new parent level.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?qbGraphIRI? – the input QB graph IRI,
# ?levelIRI? – the existing level IRI, and
# ?propertyIRI? – the property IRI (i.e., new level)

INSERT INTO ?qb4oGraphIRI? {
?propertyIRI? a qb4o:LevelProperty .
?obj a qb4o:LevelMember .
?obj qb4o:memberOf ?propertyIRI? .
?levelMember2 skos:broader ?obj2 . }
WHERE { {
 SELECT DISTINCT ?obj
 FROM ?qbGraphIRI?
 WHERE { {
  SELECT ?levelMember
  FROM ?qb4oGraphIRI?
  WHERE {
   ?levelMember a qb4o:LevelMember .
   ?levelMember qb4o:memberOf ?levelIRI? .  }}
  ?levelMember ?propertyIRI? ?obj .  }}
{ SELECT ?levelMember2 ?obj2
  FROM ?qbGraphIRI?
  WHERE {
  {SELECT ?levelMember2
   FROM ?qb4oGraphIRI?
   WHERE {
    ?levelMember2 a qb4o:LevelMember .
    ?levelMember2 qb4o:memberOf ?levelIRI? .  }}
  ?levelMember2 ?propertyIRI? ?obj2 .
 } GROUP BY ?levelMember2 ?obj2 } }
```

An example of triples for two new levels, their level members, and linking of the level members of the new parent levels with the level members of the child level added to the QB4OLAP graph is presented in Example 16.

Example 16
Resulting triples of Query 10.

```
1 newG:region a qb4o:LevelProperty .
2 region:ECS a qb4o:LevelMember .
3 region:ECS qb4o:memberOf newG:region .
4 country:RS skos:broader region:ECS .
5
6 newG:income–level a qb4o:LevelProperty .
7 income:UMC a qb4o:LevelMember .
8 income:UMC qb4o:memberOf newG:income–level .
9 country:RS skos:broader income:UMC .
```

Line 1 illustrates the new level definition for the `region` level. Lines 2 and 3 exemplify a new level member and its linking to the `region` level, respectively. Then, line 4 links a child level member (i.e., `country:RS`) to the new parent level member (i.e., region:ECS) and this way creating a rollup relationship between them. Finally, lines 6-9 reflect the same definition for `income-level`.

Definition of a hierarchy step in an existing hierarchy. Having a new parent level defined and added to the QB4OLAP graph (both schema and

instance parts), we next need to create a hierarchy step that determines the order of these levels in a hierarchy. In this context, we explain two particular cases. The first, simpler, case is to add a new parent level to a level that is either the only level in a hierarchy or that is the last (i.e., coarsest) parent level in the hierarchy. The inputs for this task are the QB4OLAP graph IRI, the child level IRI, the new parent level (i.e., property) IRI, the hierarchy IRI, and the hierarchy step IRI. This can be performed with Query 11. To build the hierarchy illustrated in Figure 4.4, the query should be run for the `region` level that is added as parent level to the `refArea` level.

Query 11
Definition of a hierarchy step in an existing hierarchy.

```
1  # INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
2  # ?levelIRI? – the existing level IRI,
3  # ?propertyIRI? – the property IRI (i.e., new level),
4  # ?hierarchyIRI? – the hierarchy IRI, and
5  # ?hslRI? – the hierarchy step IRI (typically blank node)
6
7  INSERT INTO ?qb4oGraphIRI? {
8  ?hslRI? a qb4o:HierarchyStep .
9  ?hslRI? qb4o:inHierarchy ?hierarchyIRI? .
10 ?hslRI? qb4o:parentLevel ?propertyIRI? .
11 ?hslRI? qb4o:childLevel ?levelIRI? .
12 ?hslRI? qb4o:pcCardinality qb4o:ManyToOne .
13 ?hierarchyIRI? qb4o:hasLevel ?propertyIRI? .}
```

Example triples for creating a hierarchy step in an existing hierarchy are presented in Example 17.

Example 17
Resulting triples of Query 11.

```
1 _:newHierarchyStep a qb4o:HierarchyStep .
2 _:newHierarchyStep qb4o:inHierarchy newG:geoHierarchy .
3 _:newHierarchyStep qb4o:parentLevel newG:region .
4 _:newHierarchyStep qb4o:childLevel sdmx−dimension:refArea .
5 _:newHierarchyStep qb4o:pcCardinality qb4o:ManyToOne .
6 newG:geoHierarchy qb4o:hasLevel newG:region .
```

Line 1 defines the new hierarchy step and lines 2 – 5 link the hierarchy step with its hierarchy, parent level, child level, and cardinality, respectively. Finally, line 6 adds the new parent level to the existing hierarchy.

Definition of a hierarchy step while creating a new hierarchy. The second, more complex, case is to add a parent level to a child level that already has a parent level (in one or more hierarchies). In this case, for each hierarchy where there is a parent level, create a new hierarchy. Then, replicate the hierarchy steps and add the corresponding levels such that the child level is the only or the last (i.e., coarsest) parent level in the hierarchy. Finally, add the new parent level to the new hierarchy and create a new hierarchy step with

the child level. In case that identical (i.e., duplicate) new hierarchies would be created from different existing hierarchies, only one new hierarchy should be created. Thus, in the context of Figure 4.4, adding the `income-level` level as parent to the `refArea` level (i.e., S2 in H2) can be performed with Query 12. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the child level IRI, the new parent level (i.e., property) IRI, the hierarchy IRI, and the hierarchy step IRI. Note that if there was a longer sequence of levels (and hierarchy steps) to be replicated, Query 12 would need to be extended.

Query 12
Definition of a hierarchy step with creating a new hierarchy.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?qbGraphIRI? – the input QB graph IRI,
# ?levelIRI? – the existing level IRI,
# ?propertyIRI? – the property IRI (i.e., new level),
# ?hierarchyIRI? – the hierarchy IRI, and
# ?hslRI? – the hierarchy step IRI (typically blank node)

INSERT INTO ?qb4oGraphIRI? {
?hierarchyIRI? a qb4o:Hierarchy .
?hierarchyIRI? qb4o:inDimension ?d .
?d qb4o:hasHierarchy ?hierarchyIRI? .
?hslRI? a qb4o:HierarchyStep .
 ?hslRI? qb4o:inHierarchy ?hierarchyIRI? .
 ?hslRI? qb4o:parentLevel ?propertyIRI? .
 ?hslRI? qb4o:childLevel ?levelIRI? .
 ?hslRI? qb4o:pcCardinality qb4o:ManyToOne .
?hierarchyIRI? qb4o:hasLevel ?propertyIRI? .
?hierarchyIRI? qb4o:hasLevel ?levelIRI? }
FROM ?qbGraphIRI?
WHERE {
 ?h a qb4o:Hierarchy .
 ?h qb4o:hasLevel ?levelIRI? .
 ?h qb4o:inDimension ?d .
 ?d a qb:DimensionProperty .
}
```

Example triples for creating a hierarchy step in a new hierarchy are presented in Example 18.

Example 18
Resulting triples of Query 12.

```
newG:incomeHierarchy a qb4o:Hierarchy .
newG:incomeHierarchy qb4o:inDimension newG:geoDimension .
newG:geoDimension qb4o:hasHierarchy newG:incomeHierarchy .
_:newHierarchyStep2 a qb4o:HierarchyStep .
_:newHierarchyStep2 qb4o:inHierarchy newG:incomeHierarchy .
_:newHierarchyStep2 qb4o:parentLevel newG:income-level .
_:newHierarchyStep2 qb4o:childLevel sdmx-dimension:refArea .
_:newHierarchyStep2 qb4o:pcCardinality qb4o:ManyToOne .
newG:incomeHierarchy qb4o:hasLevel sdmx-dimension:refArea .
newG:incomeHierarchy qb4o:hasLevel newG:income-level .
```

Lines 1 – 3 define the new hierarchy and link it with the existing dimension. Then, lines 4 – 8 create a new hierarchy step and link it with its hierarchy, parent level,

child level, and cardinality, respectively. Finally, lines 9 and 10 add both levels to the hierarchy.

In any case, when adding a hierarchy step, cycles need to be avoided in the hierarchy definition and this can be achieved following the rationale of Algorithm 1.

Definition of the `All` level and its level member. Finally, the mandatory `All` level with its `all` level member must top all the dimension hierarchies. Thus, we first need to add the `All` level to each dimension. The inputs for this task are the QB4OLAP graph IRI, the `All` level IRI, and the `all` level member IRI. This can be performed with Query 13.

Query 13
Definition of the `All` level and its level member.

```
1 # INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
2 # ?allLevelIRI? – the All level IRI, and
3 # ?allLevelMemberIRI? – the all level member IRI
4
5 INSERT INTO ?qb4oGraphIRI? {
6 ?allLevelIRI? a qb4o:LevelProperty .
7 ?allLevelMemberIRI? a qb4o:LevelMember .
8 ?allLevelMemberIRI? qb4o:memberOf ?allLevelIRI? .}
```

Example triples for creating the `All` level and its `all` level member are presented in Example 19.

Example 19
Resulting triples of Query 13.

```
1 newG:geoALL a qb4o:LevelProperty .
2 newG:geoALLmember a qb4o:LevelMember .
3 newG:geoALLmember qb4o:memberOf newG:geoALL .
```

Line 1 illustrates the new level definition of the all level for the geographical dimension. Line 2 defines its level member and line 3 links the member to the level.

Linking of the `all` level member with the lower level members. After creating the `All` level and its `all` level member for a dimension, all the coarsest levels of each hierarchy belonging to the dimension, must be linked to the `All` level. Furthermore, their level members must be related to the `all` level member. The inputs for this task are the QB4OLAP graph IRI, the `All` level IRI, and the child level IRI. This can be performed with Query 14. In the context of the hierarchy illustrated in Figure 4.4, the query should be run for both `region` and `income-level` that need to be linked to the `All` level.

Query 14
Linking of the `all` level member with the lower level members.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?levelIRI? – the child level IRI, and
# ?allLevelMemberIRI? – the all level member IRI

INSERT INTO ?qb4oGraphIRI? {
?levelMember skos:broader ?allLevelMemberIRI? . }
FROM ?qb4oGraphIRI?
WHERE {
?levelMember a qb4o:LevelMember .
?levelMember qb4o:memberOf ?levelIRI? . }
```

Example triples that link the child level members with the `all` level member are presented in Example 20.

Example 20
Resulting triples of Query 14.

```
1 region:ECS skos:broader newG:geoALLmember .
2 income:UMC skos:broader newG:geoALLmember .
```

Lines 1 and 2 illustrate linking of the `region` and `income-level` level members with the `all` level member in the geographical dimension, respectively.

Once the `All` levels (one per dimension) and their `all` level members are created and linked, we can use Query 11 to link both `region` and `income-level` to the `All` level. This way we can create the S3 and S4 hierarchy steps in Figure 4.4. The addition of the `All` levels for the remaining two dimensions is analogous.

Definition of a level attribute and its linking to a level. Additionally to the construction of the dimension hierarchies, the levels may have level attributes that can also be discovered with Algorithm 1. Thus, level attributes can be added to the QB4OLAP graph. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the level IRI, and the attribute IRI. This can be performed with Query 15.

Query 15
Definition of a level attribute and its linking to a level.

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?qbGraphIRI? – the input QB graph IRI,
# ?levelIRI? – the level IRI, and
# ?propertyIRI? – the property IRI (i.e., the attribute)

INSERT INTO ?qb4oGraphIRI? {
 ?propertyIRI? a qb4o:LevelAttribute .
 ?propertyIRI? qb4o:inLevel ?levelIRI? .
 ?levelIRI? qb4o:hasAttribute ?propertyIRI? .
 ?levelMember ?propertyIRI? ?obj . }
FROM ?qbGraphIRI?
WHERE {
 {SELECT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE{
```

```
16   ?levelMember a qb4o:LevelMember .
17   ?levelMember qb4o:memberOf ?levelIRI? . } }
18  ?levelMember ?propertyIRI? ?obj . }
```

Example triples of a level attribute, its linking with a level, and specifying the value for a level member are presented in Example 21.

Example 21
Resulting triples of Query 15.

```
1 skos:prefLabel a qb4o:LevelAttribute .
2 skos:prefLabel qb4o:inLevel sdmx-dimension:refArea .
3 sdmx-dimension:refArea qb4o:hasAttribute skos:prefLabel .
4 country:RS skos:prefLabel ''Serbia''@en .
```

Line 1 defines a level attribute and lines 2 and 3 link the level attribute and the corresponding level. Then, line 4 exemplifies the value of the attribute for `country:RS`.

Copying of observations. The output of the previous steps (i.e., queries) is the complete QB4OLAP schema graph and partial QB4OLAP instance graph. To complete the latter one, the observations need to be copied to the QB4OLAP graph. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, and the QB dataset IRI. This can be performed with Query 16.

Query 16
Copying of observations.

```
1  # INPUT: ?qb4oGraphIRI? - the new QB4OLAP graph IRI,
2  # ?qbGraphIRI? - the input QB graph IRI, and
3  # ?dsIRI? - the dataset IRI
4
5  INSERT INTO ?qb4oGraphIRI? {
6  ?oIRI a qb:Observation .
7  ?oIRI ?prop ?val .  }
8  FROM ?qbGraphIRI?
9  WHERE {
10  ?oIRI a qb:Observation .
11  ?oIRI qb:dataSet ?dsIRI? .
12  ?oIRI ?prop ?val . }
```

Example triples of a QB observation copied to the QB4OLAP graph are presented in Example 22.

Example 22
Resulting triples of Query 16.

```
1 <http://worldbank.270a.info/dataset/world-bank-indicators/
2                                   CM.MKT.LCAP.CD/RS/2012>
3   a qb:Observation ;
4   qb:dataSet dataset:CM.MKT.LCAP.CD ;
5   property:indicator indicator:CM.MKT.LCAP.CD ;
6   sdmx-dimension:refArea country:RS ;
7   sdmx-measure:obsValue 7450560827.04874 ;
```

Lines 1-3 define an observation and the rest of the triples link observation with the dataset, levels, and measure value for `country:RS`.

5.4 Additional considerations

We want to position our approach with respect to two QB concepts that might be considered relevant in our context, namely `qb:AttributeProperty` and `qb:Slice`. The former refers to the attributes describing the observations (e.g., the type of the measure). We consider these attributes as metadata rather than elements of the schema. Our proposal is to capture this information, by means of analytical metadata (e.g., the SM4AM analytical metadata [103]). For instance, different attributes can be defined as instances (via `rdf:type`) of `sm4am:DataProperty` and kept as additional metadata. Regarding the latter, `qb:Slice`, we focus on the base cuboid, without considering higher level cuboids, since they are derivable from the base cuboid.

For the creation of the IRIs of the new objects, we recommend to use the W3C best practices.[26]

A final consideration relates to whether or not the QB4OLAP enrichment should entail the construction of a new graph. For simplicity of presentation, we assume so far that the method steps always build a new graph. Nevertheless, the newly created QB4OLAP data cube schema can be used for the exploration of the existing observations. As the existing QB dimension properties used in the observations are now redefined as QB4OLAP dimension level properties, the new schema that defines dimension structures and includes the aggregate functions for measures can be used to interpret observations and aggregate measure values. Moreover, the existing graph can be changed such that its `qb:DataSet` points only to the newly defined `qb:DataStructureDefinition` representing the QB4OLAP data cube schema.

6 Evaluation

To evaluate our approach, we have built the tool called *QB2OLAP Enrichment Module* (QB2OLAPem) that redefines a QB data set in terms of the QB4OLAP vocabulary. The resulting QB4OLAP data set can then be enriched as discussed in the previous section by relating measures with aggregate functions and by discovering new potential dimensional concepts that can then be used to extend the MD knowledge described in the QB4OLAP data set. To do so, QB2OLAPem follows the steps described in the method. Our tool was used in a set of experiments aimed at validating our approach and in this section we present the results. We first briefly introduce QB2OLAPem. Then, we explain the evaluation setting and rational. Finally, we present the evaluation results for the applicable quality characteristics of the Quality in use and the Product quality models of the ISO/IEC 25000 [49].

[26] `http://www.w3.org/2011/gld/wiki/223_Best_Practices_URI_Construction`

6.1 QB2OLAP Enrichment Module

QB2OLAPem is developed in Java using JRE 1.8.0_60, Apache Jena 2.13.0 for working with RDF graphs, SWT[27] for the GUI, and Windows 8.1 as OS. It implements the algorithm to discover implicit dimensional concepts from Section 4 and the enrichment method from Section 5. The enrichment steps are performed in an iterative and interactive fashion. QB2OLAPem automatically retrieves potentially new dimension levels and level attributes, lists them to the user who can choose the ones of her interest, and automatically enriches the data set based on the user choices. The user can also configure an aggregate function for each measure. QB2OLAPem visualizes the cube structure for the user and enables her to automatically generate the QB4OLAP triples once the enrichment is finished. This way, QB2OLAPem enables user-friendly and (semi-)automatic enrichment of QB data sets with additional QB4OLAP semantics. Furthermore, the output triples produced by QB2OLAPem can be straightforwardly loaded into an SW-based OLAP engine and allow traditional OLAP analysis (i.e., by means of a high-level interface in terms of an MD algebra that automatically translates these high-level operators in terms of SPARQL). Thus, it lowers the entry barrier for data analysis on SW data for non-expert users (e.g., traditional OLAP users). For further details, [102] reports on how to connect QB2OLAPem with an OLAP engine. From here on we focus on QB2OLAPem, whose process flow is presented in Figure 4.5.

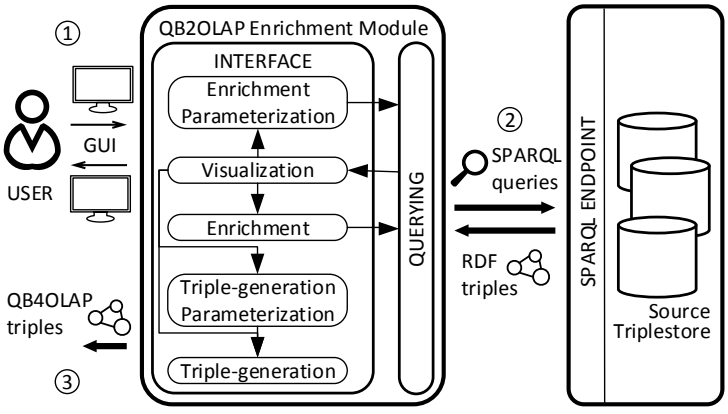

Fig. 4.5: QB2OLAP Enrichment Module Process Flow

The figure illustrates three kinds of external processes, as well as the QB2OLAPem internal process flow. The first kind of external process is the user–system interaction, where the user is guided by the Interface element to iteratively perform the semi-automatic enrichment using the QB2OLAPem

[27] https://www.eclipse.org/swt/

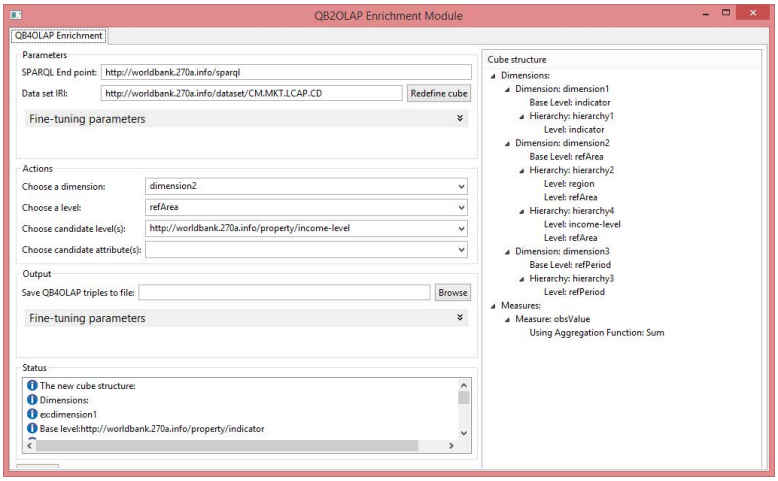

Fig. 4.6: QB2OLAP Enrichment Module Screenshot

GUI. The second kind of external process is the system–SPARQL endpoint interaction where QB2OLAPem queries external triple stores to identify implicit potential MD concepts that could be used for the enrichment. The last kind of external process is the generation of the QB4OLAP triples that can then be used in other tools (e.g., an OLAP engine, a SPARQL endpoint, etc.). These three kinds of external processes are coordinated by the QB2OLAPem internal process flow as follows. In the user–system interaction, the user initiates the enrichment process by specifying the SPARQL endpoint, the QB data set IRI, and possibly additional fine-tuning parameters in the Enrichment Parameterization activity. Then, the Querying element triggers the SPARQL queries to retrieve the RDF triples specifying the initial QB cube structure. This structure is then redefined in terms of QB4OLAP (see Section 5.2). The results are then visualized to the user in the Visualization activity. Moreover, the Querying element also runs the queries to discover candidate enrichment concepts (see Section 4.3). The user can then again set the fine-tuning parameters and optionally restart the process, perform the enrichment in the Enrichment activity, or decide to generate triples in the Triple-generation activity, where she can optionally first set the fine-tuning parameters for triple generation in the Triple-generation Parameterization activity. After each enrichment, the Querying element again runs the necessary SPARQL queries to discover new candidate enrichment concepts if any. The enrichment process finishes when the user decides to generate the QB4OLAP triples as the final result. The QB2OLAPem GUI is illustrated in Figure 4.6 and more details about the tool can be found in [102].

6.2 Evaluation Setting and Rational

For the evaluation of our approach, we adopt quality characteristics from the Quality Model of the ISO/IEC 25000 – System and Software Quality Requirements and Evaluation (SQuaRE) series of standards [49] that are applicable to our usage scenario. It includes the Quality in use model and the Product quality model that specify quality characteristics and subcharacteristics for which we define metrics specific for our tool. Table 4.1 presents the details.

The main target group of users for our tool is OLAP practitioners that are well familiar with the data cube abstraction paradigm (see Section 2.1). Typical OLAP users are not proficient in, for example, SQL or other query languages. Instead, they analyze data using graphical user interfaces of different OLAP engines. Such high-level automation is enabled by the definition of a correct MD schema (see Section 2.1), which allows OLAP users to abstract their actions from data manipulation languages (such as SQL or SPARQL) and use high-level interfaces based on an MD algebra not demanding IT expertise [101]. In the same spirit, our tool supports OLAP users to enrich the QB schema with a graphical environment. Thus, more than just analyzing data as in traditional OLAP settings, QB2OLAPem enables OLAP users to participate in discovering interesting analysis perspectives and construct the MD schema according to their requirements / needs. Moreover, although the users with both SW and OLAP skills could perform this manually, our tool lowers the barrier such that the users non-familiar with SW technologies and even non-technical users can perform the enrichment. Therefore, we conduct the experiments with 25 users including 4 OLAP experts, 4 SW experts, and 17 users having some knowledge of both. The expert users are PhD students[28] and a master student[29] doing research in the OLAP and SW areas. The non-expert users include a PhD student familiar with both areas and students of a UPC-BarcelonaTech master[30] that includes a Data Warehousing / OLAP course and a Semantic Web course. All of them had a 20 minutes tutorial in QB / QB4OLAP and some recall on DW / OLAP to facilitate the understanding of the template SPARQL queries provided to perform the manual enrichment as well as the overall objective.

For the experiments with the users we consider the running example data set. The users should perform the enrichment according to a predefined scenario. Currently, the alternative to using our tool is that the user manually explores a QB data set, discovers enrichment concepts, and constructs the QB4OLAP graph by using SPARQL while, also manually, storing and defining RDF triples. Thus, we compare this case as an alternative to using the tool for the same tasks. The users are asked to both use the tool and

[28] `https://it4bi-dc.ulb.ac.be/`

[29] `http://it4bi.univ-tours.fr/it4bi/`

[30] `http://www.fib.upc.edu/en/masters/miri.html`

Table 4.1: Quality Characteristics, Subcharacteristics, and Metrics

Model	Characteristics	Subcharacteristics	Metrics
Quality in use - The evaluation of the interaction with a system	**Effectiveness** - Accuracy and completeness with which users can enrich a schema	/	The percentage of users producing (not) complete and (not) sound schemata according to a predefined scenario, both with and without the tool
	Efficiency - The actions that the user needs to perform to enrich the schema	/	The number of user actions and time contributing to redefinition/enrichment tasks according to a predefined scenario, and the total number of user actions and total time
	Satisfaction - How much is the user satisfied with using the tool	**Usefulness** - How useful does the user consider the tool	Survey statistics
Product quality - The software product quality properties	**Functional suitability** - The functions specified by the tool	**Functional completeness** - Functions that cover the specified tasks	The percentage of cases where the user produces a complete schema according to the underlying QB data set and a predefined scenario
		Functional correctness - Correctness of the tool results	The percentage of cases where the user produces a sound schema according to the MD model
	Performance efficiency - The amount of resources used for the enrichment	**Time-behavior** - The processing time	(i) Redefinition time to: a) Retrieve the initial cube structure, b) Retrieve the level members, and (ii) Enrichment time: a) Properties / Level members retrieval time
		Capacity - The limits of the tool	The number of observations (characterizing the data set size) that can be handled by the tool
	Usability - How much can a tool be used for the enrichment with effectiveness, efficiency, and satisfaction	**Learnability** - Degree to which specified users can learn to use the system with effectiveness, efficiency, and satisfaction	Statistics about how many different types of users are able to use all (or partial) tool functions (e.g., redefinition, adding new level, etc.)
		Operability - The attributes of the system that make it easy to operate and control the enrichment	The time that the user takes for performing the actions in the tool

Table 4.2: The Schema Transformation

Data set	#dimensions	t_{schema} (s)	#observations	#members	$t_{members}$ (s)	#properties	$t_{properties}$ (s)
Market Capitalization	3	0.91	2360	146	0.04	2364	1.08
Renewable Energies	6	1.18	537447	155	1.05	786	1.05
Asylum seekers	7	0.94	499369	286	1.4	796	1.90

follow a guided tutorial that provides template SPARQL queries to manually perform the enrichment. They are provided with detailed guidelines in both cases. Moreover, the order of the cases is alternated among users. After the enrichment, the users deliver the results and their feedback. Later, for the evaluation of the tool performance, we consider two more data sets from another Linked Data source and of different sizes. All the data sets are loaded into a local endpoint after a cleaning process in which observations with pre-aggregated and zero measure values are removed. Moreover, we applied additional transformations, e.g., conversion from literals to IRIs to enable further search for new levels. The local SPARQL endpoint is provided by a Virtuoso 7 server running on an Intel Core i5 CPU at 2.6 GHz and 8 GB of RAM machine with Windows 8.1 OS. The details and results of the experiments are presented in the following subsections.

6.3 Quality in Use Evaluation

The quality in use metrics are evaluated over the "Market Capitalization" QB data set from our running example. It is an indicator data set of 1.9 MB in size extracted from the WBLD source. All the WBLD indicator data sets have the same cube structure and the average size is of approximately 1.15 MB. Thus, "Market Capitalization" is considered as a representative data set for WBLD. Table 4.2 includes its main features such as the number of dimensions in the schema (#dimensions), the number of observations which in fact characterizes the size of the data set (#observations), and the total number of dimension members for all dimensions (#members), along with other data sets and results relevant for the next subsection.

The scenario that the users need to perform is to redefine the schema according to the QB4OLAP vocabulary, check if certain properties can be considered as candidates for the enrichment, create three-level hierarchies (i.e., hierarchies consisting of the initial QB level, one new level on top of the initial one, plus the All level) for two different dimensions and optionally add a level attribute to a level. Figure 4.4 summarizes the enrichment to be done for each of these dimensions: define levels, a hierarchy for each rollup path, hierarchy steps for each adjacent pair of nodes in the same path, and

Table 4.3: Effectiveness Results

Option / Case	Manual		Tool	
Sound & Complete	16%		84%	
Incomplete and/or Not Sound	84%		16%	
	Incomplete	100%	Incomplete	100%
	Not sound	47.62%	Not sound	0%

a dimension wrapping all of them (see Section 5). The evaluation guidelines provide a predefined scenario and thus, ask the participants to enrich the data set with some specific levels. The quality in use characteristics specified in Table 4.1 focus on the user efforts to perform these tasks with and without the tool. Detailed guidelines for both i) the manual and ii) tool enrichment were given[31]. The guidelines include i) the set of predefined SPARQL queries with the parameters that need to be used and ii) the specification of tool actions with the parameters that need to be used, respectively. The users have one hour and a half to perform the manual enrichment and half an hour with the tool.

Effectiveness. The results of the metrics for the effectiveness characteristics are presented in Table 4.3. The table presents the percentage of users producing sound and complete schemata with and without the tool. In this context, completeness refers to whether the user managed to complete all the tasks required to fulfill the requirements in the guidelines within the time available, while soundness indicates whether the schema produced is correct in terms of MD modeling (see Section 6.4 for details). The results show that, even with very detailed guidelines, only 4 (including just 2 SW experts) out of 25 users managed to manually create a sound and complete schema. The rest of users did not manage to manually complete all the tasks and almost half of them also created errors in this process undermining the schema soundness. These errors are typically a consequence of copy/paste actions and include the same concept defined as both level and level attribute, or hierarchy and hierarchy step, adding the same level to different dimensions (and their hierarchies), adding the same level members to several levels, adding duplicate hierarchies, etc. Oppositely, 4 users did not complete the task but the partial output produced was correct (i.e., generated sound but incomplete schemata). The results are much better when using the tool. 21 participants created sound and complete schemata. Out of the 4 not completing the task, none produced an incorrect schema. This shows that in practice *the tool guarantees the schema soundness*.

Efficiency. Regarding the efficiency characteristic of the Quality in use model, the results of our experiments with users are presented in Table 4.4. The table represents the average number of effective actions (i.e., queries or

[31]The guidelines given to the evaluation participants can be found at: `http://www.essi.upc.edu/~jvarga/qb2olapem.html`

Table 4.4: Efficiency Results

Task / Parameter	Redefinition		Enrichment	
	Manual	Tool	Manual[32]	Tool
Effective Actions	11.2	5.6	14	9.28
Total Actions	17	6.32	17	9.92
Effective Time (min)	30.02	4.18	43.32	7.15 (3.75)
Total Time (min)	45.5	4.72	52.38	7.64 (3.75)
Ratio of Effective and Total Actions	65%	89%	82%	94%

tool interactions) contributing to the task (i.e., redefinition/enrichment) and the total number of actions (which include unnecessary actions to complete the task, such as errors). Moreover, the table illustrates the average effective and total time. At the bottom, the table shows the ratio between the effective and total actions, and it is used for the calculation of the effective time with respect to the total time. Furthermore, note that the time for the manual enrichment only refers to the 4 users who finished (thus, it does not consider the 21 users who did not finish the enrichment task). In this context, we added in brackets the values referring to these four users when performing the enrichment with the tool (since they were among the most skilled ones). Thus, Table 4.4 illustrates that *the tool reduces the necessary time* for the redefinition and enrichment *by at least 7 times* even when users are provided with detailed guidelines for manually performing these tasks.

Satisfaction. Finally, we discuss the results of a survey answered by the users that provide insights about the satisfaction characteristic of the Quality in use model[33]. The average user rating of how much they like the tool, how easy it is to use the tool, how useful the tool is, and if they would use it in the future, are all over 4 in a scale from 1 to 5. The average user rating on the helpfulness of the manual and tool guidelines is 3.98 and 4.1, respectively. However, only 5 users (including 2 SW experts) consider that they would be able to formulate the queries without guidelines with a certainty of 4 or 5. However, this is proven wrong since even with the guidelines only 4 users were able to generate a sound and complete schema as discussed earlier. In an open-ended question section, more than 10 users, including an SW expert, stated that the manual part is too repetitive, error-prone, and hard to perform even with the guidelines. Thus, *QB2OLAPem was appreciated and considered needed and helpful*. The main reason for this is that generating a QB4OLAP data set from an available QB data set does not consist of purely syntactical transformations but requires triggering queries to identify and validate potential new dimensional concepts (i.e., levels). Such queries need to guarantee the MD integrity constraints and also require a solid OLAP knowledge

[32]The values refer to the 4 users finishing the manual enrichment. The other 21 users did not finish.

[33]The survey can be found at: `http://www.essi.upc.edu/~jvarga/qb2olapem.html`

to avoid making mistakes. Additionally, the users also pointed out some improvements to be done in our tool. Mainly interface enhancements, e.g., changing the way of choosing new concepts and adding the capacity to delete / remove concepts (which is not currently present). We plan to make the changes / add these features to QB2OLAPem in the future.

6.4 Product Quality Evaluation

The product quality evaluation focuses on the tool properties. In the following paragraphs we discuss the results for each of the characteristics with its subcharacteristics.

Functional suitability. The functional suitability is measured in terms of the *completeness* and *correctness* subcharacteristics. In our context, the functional completeness refers to the tool capacity to transform a QB data set into a QB4OLAP one as well as its capacity for extending the QB4OLAP data set with relevant dimensional data from all the available potential dimensional concepts. The functional correctness refers to the correctness of the QB4OLAP data set produced, i.e., in the MD context, if the underlying MD schema produced satisfies the MD integrity constraints discussed in Section 4.3. Next, we explain how both criteria are met via QB2OLAPem.

The ***correctness*** of an MD schema has been exhaustively studied in the DW/OLAP community (e.g., [73]). Based on these findings, the MD integrity constraints were identified and, in turn, several methods to automatically model diverse data (following a given data model) in terms of the MD model have been proposed (the reader is addressed to [85] for a detailed survey on this topic). The automatic identification of factual data (i.e., facts and measures) still results challenging for many domains. However, there is a clear consensus on discovering dimensional data based on FDs. Arbitrarily looking for FDs in the MD context is known to be computationally expensive [54]. In our case, QB2OLAPem exploits the QB semantics and uses the QB dimension concept as valid starting point of analysis from where to look for FDs. To do so, QB2OLAPem follows a traditional approach but adapted to the SW technologies [4]: it looks for many-to-one relationships by applying the algorithms presented in Section 4.3. However, like most automatic modeling approaches, instead of detecting and constructing all possible dimension hierarchies, the detection of new dimension levels and attributes is performed in an iterative fashion each time the user selects a new level to be added. In MD terms, QB2OLAPem applies a hybrid *data-driven* and *requirement-driven* approach [85]; i.e., effectively combining the discovery of MD knowledge hidden in the available data set (i.e., FDs are identified for all the initial QB dimensions) with the interest of the user (from there on, the user is able to enrich the MD schema with dimensional data that are semantically meaningful and of her interest). This approach is widely acknowledged as the

most appropriate way to proceed in automatic MD modeling. For example, in our running example, the `region` level is meaningful to be added on top of the `reference area` (referring to country) level to conform a richer hierarchy within the same dimension (and thus, the user most probably would choose this enrichment), while this is probably not the case for the `creator` level even though it is identified as an FD and proposed by our tool as a potential rollup relationship from `reference area`. With the default settings (next we discuss how to relax them), possible new levels are only available for the `reference area`. Out of the total 20 outbound distinct properties, 6 satisfy the many-to-one cardinality and only two are a meaningful choice for the construction of hierarchies, namely, `region` and `income level`. Although there are some properties that are candidates for the next coarser granularity level for these two new levels, none of them would be meaningful from the user point of view. These enrichment possibilities are based on the data available in the data set and more meaningful enrichment concepts can be found by exploring external sources (e.g., DBpedia can provide more concepts that are meaningful for constructing coarser levels for countries).

Additionally, due to imperfect data (see Section 4.1) it might happen that some properties are not proposed as FDs (e.g., due to incorrect data) when conceptually they should be. This issue, typical from the SW and similar scenarios, has also been studied in the recent past (e.g., [4]). To deal with these situations, we also consider *quasi FDs* (see Section 4.3). In any case, selecting a quasi FD to enrich the schema requires cleaning the data set to meet the FD. Otherwise, the resulting MD schema would not guarantee a correct data summarization (see [86]). In the case of our running example, Figure 4.7 illustrates the effect of reducing the percentage of instances that must satisfy the many-to-one cardinality for the `reference area`. When the percentage is reduced to 80% (i.e., the *minCompl* parameter in Algorithm 2), there is an additional candidate property (`lending type`), that is also meaningful for the construction of a new hierarchy. This is a consequence of imperfect data as not all members of the `reference area` level are countries. Removing these errors, `lending type` turns out to be a functional property and thus a rollup candidate. Accordingly, these errors must be addressed to guarantee correct data aggregations if this level is chosen. The next potentially meaningful level is found at 40% but it surely does not make sense to build a hierarchy where massive cleaning should be done. Therefore, the value of 80% represents an empirically based threshold for the discovery of new levels in the case of our running example. Overall, the running example shows a good quality for the construction of dimension hierarchies as the first two properties completely satisfy the many-to-one cardinality and the third one does so for at least 80% of the cases.

Accordingly, we say that *QB2OLAPem is functionally correct because all dimensional data proposed are based on FDs*. This is guaranteed by the SPARQL

queries internally triggered by QB2OLAPem (see Section 4.3). This is shown in Table 4.3 where no incorrect schema is produced using the tool. Note this is one of the main advantages of our tool in front of the manual enrichment. The tool guarantees the overall correctness of all the enrichment steps (queries) triggered, which cannot be guaranteed in the manual enrichment.

The ***completeness*** of the MD schema produced is guaranteed by several means. The first step is a syntactical transformation from QB to QB4OLAP constructs (i.e., the redefinition phase). There, the QB dimension concept is redefined in terms of QB4OLAP constructs, i.e., as a level within a hierarchy and belonging to a dimension. The basic QB4OLAP structure is then enriched by relating each measure with an aggregate function (see Section 5.2) and with relevant dimensional data selected by the user from that automatically discovered by the tool. The additional dimensional data added are conformed according to QB4OLAP and thus, to the good MD modeling practices (in terms of levels, hierarchies, dimensions, and dimension attributes).

We say that *QB2OLAPem is functionally complete because all relevant MD data from the QB data set are included in the QB4OLAP one, all measures are related to an aggregate function, and all the available FDs are discovered and proposed to the user as potential dimensional enrichment*. As discussed in Section 5.2, for each QB data structure QB2OLAPem retrieves all the initial QB measures and dimensions and all of them are redefined in terms of QB4OLAP. Also, QB2OLAPem guarantees that each QB4OLAP measure is associated with an aggregated function (if none is selected, SUM is used by default). About the enrichment phase (see Section 5.3), the dimensional enrichment is based on the detection of many-to-one cardinalities based on the analysis of instances (see Section 4.3). QB2OLAPem guarantees that all the instances are covered by the newly defined levels (i.e., every level instance is member of a level) and that all potential functional properties, i.e., all potential new levels, are identified and proposed to the user. This is done in two steps, QB2OLAPem exhaustively finds all FDs for the initial dimension levels (i.e., the former QB dimensions). Later, for each additional level chosen by the user, QB2OLAPem exhaustively searches for new potential FDs starting from it.

Thus, we say *QB2OLAP is complete with regard to the available data and the user requirements stated* (as hybrid automatic MD modeling tools do). Note that the manual enrichment proposed guarantees the same degree of completeness since the SPARQL template queries provided exhaustively search for FDs and it depends on the user to properly use them. The only difference in this respect is that QB2OLAPem guarantees an aggregate function will be linked to each measure, whereas the manual enrichment cannot guarantee so. Also, that in a given time frame the user is able to complete more tasks (this is shown in Table 4.3 where the degree of incompleteness with regard to the requirements given in the guidelines is lower when using the tool).

Performance efficiency. To evaluate the tool's performance, we conducted

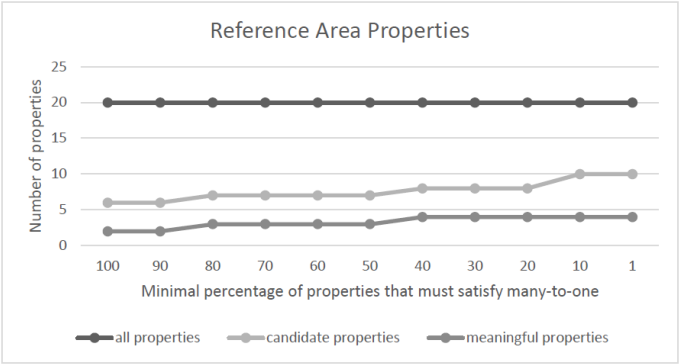

Fig. 4.7: Reference Area Candidate Properties

the experiments considering two more data sets in addition to the one used in the previous section. In particular, we include "Asylum statistics"[34](of 1.2 GB of size) and "Renewable energies (wind, solar, hydro, tidal, wave and ocean, geothermal energy, energy from biomass)"[35](660 MB) from Eurostat Linked Data. In addition to the characteristics explained in the previous subsection, Table 4.2 shows the following characteristics relevant for the performance measurement: the time to retrieve the QB schema (t_{schema}), the time to retrieve level members ($t_{members}$), the total number of properties retrieved for level members (#properties), and the time to retrieve these properties ($t_{properties}$).

The ***time-behavior*** subcharacteristic relates to the tool performance for two main tasks, the schema redefinition and enrichment. Thus, for the schema redefinition (see Section 5.2) we measure the QB schema retrieval time and the time for acquiring dimension members. As shown in Table 4.2, the schema retrieval time is approximately the same for all data sets and it does not depend on the number of observations (i.e., the data set size). This advantage originates from the fact that the QB schema typically represents a small part of the entire data set. The schema can be retrieved with a simple query where the central schema node (i.e., schema structure definition) is linked to a dimension or a measure via a component node, i.e., with only two hops (see Query 5). This maps to a path join [89] and can be efficiently solved with indexing and index-based join operations [37]. Furthermore, acquiring the level members depends on the number of observations from where they are retrieved (see Query 9). Nevertheless, as shown in Table 4.2, *even for large data sets* (e.g., half a million observations and 286 level members) *the retrieval time is still small* (around 1 sec), demonstrating the scalability. Additionally, Table 4.5 shows that, *inside a single data set, the time for acquiring level mem-*

[34] `http://eurostat.linked-statistics.org/data/migr_asyappctzm`
[35] `http://eurostat.linked-statistics.org/data/nrg_107a.rdf`

Table 4.5: Level Member Statistics

Data set	Level	#Members	$t_{members}$ (s)	#Properties	$t_{properties}(s)$
Market Capitalization	indicator	1	0.01	16	0.01
Market Capitalization	refArea	120	0.02	2348	0.91
Market Capitalization	refPeriod	25	0.01	0	0.16
Renewable Energies	indic_nrg	81	0.15	486	0.54
Renewable Energies	freq	1	0.22	0	0.01
Renewable Energies	product	15	0.18	90	0.08
Renewable Energies	geo	34	0.15	204	0.25
Renewable Energies	unit	1	0.14	6	0.01
Renewable Energies	refPeriod	23	0.21	0	0.16
Asylum seekers	citizen	157	0.25	620	1.00
Asylum seekers	freq	1	0.14	0	0.01
Asylum seekers	geo	34	0.12	132	0.23
Asylum seekers	asyl_app	2	0.28	8	0.02
Asylum seekers	age	6	0.25	24	0.04
Asylum seekers	refPeriod	83	0.13	0	0.59
Asylum seekers	sex	3	0.25	12	0.02

bers is similar regardless of their number. Again, this is the result of indexing techniques used by triple stores.

Furthermore, regarding the schema enrichment (see Section 5.3), Table 4.2 shows the total number (#properties) and the retrieval time ($t_{properties}$) for all the properties related to the level members of all the initial levels of the previously introduced data sets. Moreover, Table 4.5 provides more details about the number of properties related to the level members of a level in each data set. All values refer to the initial schemata. From the tables we can note that *the retrieval time does not depend on the number of observations but on the number of level members and their properties*. Our tool runs a query for each level member to retrieve its properties. Thus, the complexity of such queries is that of *adjacency* queries [12,67] and in our case, in the worst case, it directly depends on the out degree of each level member (LevelMemberOutDegree) of each level; i.e.:

$$\Sigma_1^{levels}\Sigma_1^{members} LevelMemberOutDegree$$

When retrieving the properties, the tool acquires the property range objects that automatically become new level members (if the property is chosen by the user). The tool then automatically iterates in the same way for these new level members. Thus, there is no need for additional queries to retrieve new level members. In the worst case, if the user chooses all potential functional properties identified by the tool, the total number of properties for the 3 data sets would be 47, 23, and 18, respectively. Nevertheless, the user is the last responsible for choosing relevant properties for creating new levels. Table 4.6 shows a real scenario for the running example data set. There, the user chooses to enrich the data set with three new levels (the bottom three rows) out of the seven proposed by the tool as functional properties.

Table 4.6: Market Capitalization Levels Statistics

Level	#Members	$t_{members}$ (s)	Avg Out-degree	#Properties	$t_{properties}$ (s)
indicator	1	0.01	16.00	16	0.01
refArea	120	0.02	19.57	2348	0.91
refPeriod	25	0.01	0.00	0	0.16
income-level	5	0.07	0.00	0	0.03
lending-type	4	0.07	0.00	0	0.02
region	7	0.07	0.00	0	0.04

Table 4.7: QB2OLAPem Usability Results

Tasks/User Group	SW	OLAP	Non-experts
Redefinition	100%	100%	100%
Enrichment	100%	75%	81.25%
Redefinition (min)	1.88	1.5	6.15
Enrichment (min)	2.75	4.75	9.47

The ***capacity*** subcharacteristic is measured by the number of observations (characterizing the size of the data set) that the tool can handle. Table 4.2 illustrates that *the tool can process data sets of different sizes*. Thus, the schema transformation is feasible and efficient even for large data sets by benefiting from indexing techniques to deploy efficient access on SPARQL endpoints [22].

Usability. In this context, we focus on the learnability and operability subcharacteristics. The results come from the experiments with users explained in Section 6.3. The ***learnability*** is measured with the percentage of different users who managed to use different tool functionalities. The first two rows of Table 4.7 show the percentage of users who successfully performed the redefinition and enrichment tasks with the tool, respectively. All the users successfully performed the redefinition. One OLAP expert and three non-expert users missed to perform the complete enrichment, although all of them performed at least one enrichment action. Thus, *all the users were capable of using both types of tool functionalities*. Moreover, the ***operability*** subcharacteristic is shown in the bottom two rows of the table that illustrate the average time that the different user types took for redefinition and enrichment, respectively. We can note that with higher expertise the necessary time for performing the tasks is reduced. Nevertheless, in all cases the redefinition and enrichment are at least 7 times faster than with the manual approach (see Section 6.3) showing *a high operability degree*.

Overall, the evaluation showed that QB2OLAP facilitates the enrichment and it is appreciated and considered needed and helpful by different users. The tool speeds up the enrichment process and guarantees the schema soundness and completeness in practice.

7 Related Work

As mentioned in Section 1, there are two main lines of research addressing OLAP analysis of SW data, namely (1) extracting MD data from the SW and loading them into traditional MD data management systems for OLAP analysis; and (2) performing OLAP-like analysis directly over SW data, e.g., over MD data represented in RDF. We next discuss them in some more detail.

Relevant to the first line (and also in some sense related to the methodology we present in this paper) are the works by Nebot and Llavori [78] and Kämpgen and Harth [58]. The former proposes a semi-automatic method for on-demand extraction of semantic data into an MD database. In this way, data could be analyzed using traditional OLAP techniques. The authors present a methodology for discovering facts in SW data, and populating an MD model with such facts. They assume that data are represented as an OWL ontology. The proposed methodology has four main phases: (1) Design of the MD schema, where the user selects the subject of analysis that corresponds to a concept of the ontology, and then selects potential dimensions. Then, she defines the measures, which are functions over data type properties; (2) Identification and extraction of facts from the instance store according to the MD schema previously designed, producing the base fact table of a DW; (3) Construction of the dimension hierarchies based on the instance values of the fact table and the knowledge available in the domain ontologies (i.e., the inferred taxonomic relationships) and also considering desirable OLAP properties for the hierarchies; (4) User specification of MD queries over the DW. Once queries are executed, a cube is built. Then, typical OLAP operations can be applied over this cube.

Kämpgen and Harth [58] study the extraction of statistical data published using the QB vocabulary into an MD database. The authors propose a mapping between the concepts in QB and an MD data model, and implement these mappings via SPARQL queries. There are four main phases in the proposed methodology: (1) Extraction, where the user defines relevant data sets which are retrieved from the web and stored in a local triple store. Then, SPARQL queries are performed over this triple store to retrieve metadata on the schema, as well as data instances; (2) Creation of a relational representation of the MD data model, using the metadata retrieved in the previous step, and the population of this model with the retrieved data; (3) Creation of an MD model to allow OLAP operations over the underlying relational representation. Such model is expressed using XML for Analysis (XMLA)[36], which allows the serialization of MD models and is implemented by several OLAP clients and servers; (4) Specification of queries over the DW, using OLAP client applications.

[36] `http://xmlforanalysis.com`

The proposals described above are based on traditional MD data management systems, thus they capitalize the existent knowledge in this area and can reuse the vast amount of available tools. However, they require the existence of a local DW to store SW data. This restriction clashes with the autonomous and highly volatile nature of web data sources as changes in the sources may lead not only to updates on data instances but also in the structure of the DW, which would become hard to update and maintain. In addition, these approaches solve only one part of the problem, since they do not consider the possibility of directly querying *à la* OLAP MD data over the SW.

The second line of research tries to overcome the drawbacks of the first one, exploring data models and tools that allow publishing and performing OLAP-like analysis directly over SW MD data. Terms like *self-service BI* [1], *Situational BI* [71], *on-demand BI*, or even *Collaborative BI*, refer to the capability of incorporating situational data into the decision process with little or no intervention of programmers or designers. The web, and in particular the SW, is considered as a large source of data that could enrich decision processes. Abelló et al. [1] present a framework to support self-service BI, based on the notion of *fusion cubes*, i.e., MD cubes that can be dynamically extended both in their schema and their instances, and in which data and metadata can be associated with quality and provenance annotations.

To support the second approach mentioned above, the RDF Data Cube vocabulary [27] aims at representing, using RDF, statistical data according to the SDMX information model. Although similar to traditional MD data models, the SDMX semantics imposes restrictions on what can be represented using QB. Etcheverry and Vaisman [33] proposed QB4OLAP, an extension to QB that allows to represent analytical data according to traditional MD models, also presenting a preliminary implementation of some OLAP operators (RollUp, Dice, and Slice), using SPARQL queries over data cubes specified using QB4OLAP. These two approaches have been thoroughly discussed in Sections 2 and 3. In [48], Ibragimov et al. present a framework for Exploratory OLAP over Linked Open Data sources, where the MD schema of the data cube is expressed in QB4OLAP and VoID. Based on this MD schema the system is able to query data sources, extract and aggregate data, and build an OLAP cube. The MD information retrieved from external sources is also stored using QB4OLAP.

Kämpgen et al. [59, 60] attempt to override the lack of structure in QB, discussed in Section 3, defining an OLAP data model on top of QB and other related vocabularies, e.g., some proposed ISO extensions to SKOS.[37]. They propose to represent each level as an instance of a class `skosclass:ClassificationLevel` and organize levels in hierarchies via stating the depth of

[37] `http://www.w3.org/2011/gld/wiki/ISO_Extensions_to_SKOS`

each level in the hierarchy using the `skosclass:depth` property. The proposed representation restricts levels to participate in only one hierarchy and does not provide support for level attributes. They also propose a mechanism for implementing some OLAP operators over these extended cubes, using SPARQL queries, but restricting to data cubes with only one hierarchy per dimension.

Related to the SKOS extensions mentioned above, and realizing that SKOS is insufficient to represent the needs of statistical classifications and concept management, a new proposal to address those needs was issued, and denoted as XKOS.[38] XKOS attempts to capture the main semantic relationships like generalization, specialization, part-of, among other ones, with properties like `xkos:generalizes`, `xkos:specializes`, `xkos:isPartOf`, respectively. At the time of writing the present paper, this proposal is in a preliminary status.

For an exhaustive study of the possibilities of using SW technologies for OLAP, we refer the reader to the survey by Abelló et al. [4].

8 Conclusion

The approach presented in this paper opens new possibilities for performing OLAP analysis in Linked Data and SW contexts. After thoroughly elaborating on the significant benefits that QB4OLAP brings in terms of additional schema constructs that are necessary for the state-of-the-art OLAP analysis, we have elaborated on how to, as automatically as possible, introduce these enhancements into an existing QB data set. We have proposed the use of metadata to automate the association between measures and aggregate functions, and the algorithm for the detection of implicit MD semantics to automate the discovery of dimension hierarchy schema and instances, since these are the two most relevant semantic enhancements of QB4OLAP. The enrichment task is formalized in a semi-automatic method that defines steps described as SPARQL queries to create a new enriched QB4OLAP graph. Moreover, we have presented QB2OLAPem implementing the method and the algorithm for the detection of implicit MD semantics. QB2OLAPem enables the user to enrich a QB data set with minimal user efforts. Finally, the evaluation of our approach using three real-world QB data sets of different sizes demonstrates its feasibility in practice. Furthermore, the experiments conducted with 25 users show that, in practice, QB2OLAPem facilitates, speeds up, and guarantees the correct results of the enrichment process.

In the future, we plan to extend our approach to automatically identify the data heterogeneity cases and inspired by [57] explore the possibilities to integrate different QB schemata into a single QB4OLAP schema.

[38] `http://rdf-vocabulary.ddialliance.org/xkos.html`

Acknowledgments

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC) and it has been partially supported by the Secretaria d'Universitats i Recerca de la Generalitat de Catalunya, the grant number 2014 SGR-1534. Alejandro Vaisman was partially supported by PICT-2014 Project 0787, funded by Argentina's Scientific Agency.

1 Enrichment Methodology

Here, we provide a fully formalized, more general, and detailed enrichment methodology, which is agnostic of the implementation decisions made. The methodology specifies the pre-conditions, post-conditions, and transformations to be conducted by each step in terms of set theory. The method presented in Section 5 is an instantiation of it.

The methodology considers the inputs listed below. These inputs can be generated from the outputs of Algorithm 1 (i.e., *allL*, *allP*, *allHS*, and *allLLA*) as follows. We can construct the dimension hierarchies (i.e., build structures like the one in Figure 4.2) and generate the inputs needed by the methodology by following the detected hierarchy steps (the *allHS* output of Algorithm 1). Each of these structures is considered a dimension (from here on, *dimension structure*) that has a name assigned to it, e.g., derived from the initial level name. A bottom-up traversal of the paths in this structure defines the hierarchies. On top of each dimension structure, the `All` level should be automatically added as the highest granularity level for all hierarchies. Thus, we obtain the inputs required for the enrichment methodology as follows.

- *NewLevSet*, the set of the new level IRIs (e.g., `schema:region`), generated as the difference between the set of all levels and the set of initial levels.
- *DimSet*, the set of dimension IRIs (e.g., `schema:geoDim`), generated to include an element for each dimension structure.
- *HierSet*, the set of hierarchy IRIs (e.g., `schema:geoHier`), generated for each distinct traversal path in the dimension structures.
- *LAttSet*, the set of new level attribute IRIs (e.g., `schema:capital`), extracted from the *allLLA* output of Algorithm 1.
- *HStepSet*, the set of hierarchy step IDs, i.e., IRIs or blank node identifiers (e.g., `_:hs1`), generated for each occurrence of a hierarchy step in hierarchies.
- *MapLAttribute2L*, a mapping of level attribute IRIs to level IRIs (e.g., (`schema:capital`, `sdmx-dimension:refArea`)), generated from the *allLLA* output of Algorithm 1, as pairs of IRIs.
- *MapH2D*, a mapping of hierarchy IRIs to dimension IRIs (e.g., (`schema:geoHier`, `schema:geoDim`)), extracted from the dimension structure as pairs of IRIs.
- *MapH2L*, a mapping of hierarchy IRIs to level IRIs (e.g., (`schema:geoHier`, `schema:region`)), extracted from the dimension structure as pairs of IRIs.
- *MapHStep2ParentL*, a mapping of hierarchy step IDs to parent level IRIs (e.g., (`_:hs1`, `schema:region`)), extracted from the dimension structure as pairs of IRIs.
- *MapHStep2ChildL*, a mapping of hierarchy step IDs to child level IRIs (e.g., (`_:hs1`, `sdmx-dimension:refArea`)), extracted from the dimension structure as pairs of IRIs.

• *MapHStep2H*, a mapping of hierarchy step IDs to hierarchy IRIs (e.g., (`_:hs1`, `schema:geoHier`)), extracted from the dimension structure as pairs of IRIs.

• *MapHStep2C*, a mapping of hierarchy step IDs to cardinality IRIs (e.g., (`_:hs1`, `qb4o:ManyToOne`)), extracted from the dimension structure as pairs of IRIs.

• *MapChild2Parent*, a mapping of child level member IRIs to parent level member IRIs (e.g., (`country:RS`, `region:ECS`)), extracted from the data set(s) as pairs of IRIs according to the hierarchy step structures.

• *MapLInstance2L*, a mapping of level member IRIs to level IRIs (e.g., (`country:RS`, `sdmx-dimension:refArea`)), extracted from the data set(s) as pairs of IRIs for all levels.

• *MapLInstance2LAInstance*, a mapping of level member IRIs to the level attribute IRI–level attribute value (that is IRI or literal) pairs (e.g., (`country:RS`, (`schema:capital`, ‘‘`Belgrade`’’`8sd:string`))), extracted from the data set(s) based on the *MapLAttribute2L* mapping as, IRI–a pair of IRIs, or, IRI–an IRI and literal pair, pairs.

Taking advantage of the QB4OLAP vocabulary, we next propose a methodology to enrich a QB data set. The methodology defines the steps that need to be performed for the input QB data set in order to produce an output data set that is enriched with QB4OLAP semantics (e.g., dimension hierarchies). The methodology steps are fully automatized considering that the inputs discussed above are provided. Taking into account that the inputs generation involves some user actions, the overall enrichment process is semi-automatized. The methodology steps are:

1. Redefinition of the cube schema.
2. Specification of the aggregate functions.
3. Definition of the dimension hierarchies.
4. Annotation of the cube instances.

In this section, we first introduce preliminaries for the formal definition of each step. Then, each step is defined in terms of the input, tasks to be performed, and output that it produces.

1.1 Methodology Preliminaries

We first formally define a QB graph.

Definition 1
A QB graph G_{qb} is a set of RDF triples, i.e., an RDF graph, defined as follows.

• $G_{qb} = S_{qb} \cup I_{qb}$, where S_{qb} and I_{qb} are sets of triples that define the QB cube schema and instances, respectively.

- $S_{qb} = DS_{qb} \cup DSD_{qb} \cup D_{qb} \cup M_{qb}$, where DS_{qb}, DSD_{qb}, D_{qb}, and M_{qb} are subsets of triples that define the cube data set, the cube structure, the cube dimensions, and the cube measures, respectively. Following QB's notation, the cube structure (i.e., DSD_{qb}) is defined as a set of dimensions and measures (in QB terminology, *the cube components*).
- $I_{qb} = DI_{qb} \cup O_{qb}$, where DI_{qb} is the subset of triples that defines all dimension instances, while O_{qb} is the subset of triples that defines the observations. As discussed before, observations represent measure values for the fixed dimension instances determined by DSD_{qb} related to the data set DS_{qb}.

The considered elements of the QB graph are the ones needed in our approach to create a QB4OLAP graph while the other ones are omitted. Triple examples of S_{qb} that are extracted from the running example (see Section 2.3) are presented in Example 23.

Example 23
S_{qb} triples.

```
1 <http://worldbank.270a.info/dataset/world-bank-indicators/structure>
2  a qb:DataStructureDefinition ;
3  qb:component  [ qb:dimension sdmx-dimension:refArea ] ;
4  qb:component [ qb:measure sdmx-measure:obsValue ] .
5 sdmx-dimension:refArea a qb:DimensionProperty .
6 sdmx-measure:obsValue a qb:MeasureProperty .
7 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet ;
8  qb:structure
9     <http://worldbank.270a.info/dataset/world-bank-indicators/structure> .
```

Lines 1 - 4 belong to DSD_{qb}, line 5 to D_{qb}, line 6 to M_{qb}, and line 7 to DS_{qb}. The data set DS_{qb} is related to the cube structure DSD_{qb} in lines 8 – 9 and this triple belongs to DS_{qb}.

Triple examples of I_{qb} are presented in Example 24.

Example 24
I_{qb} triples.

```
1 data:world-bank-indicators/CM.MKT.LCAP.CD/RS/2012
2   a qb:Observation ;
3   sdmx-dimension:refArea country:RS ;
4   sdmx-measure:obsValue 7450560827.04874 ;
5   qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> .
```

Lines 1 – 5 define an observation O_{qb}. Line 3 specifies a dimension instance that belongs to DI_{qb} and line 4 defines a measure value of O_{qb}. Note that this observation relates to the cube schema structure (i.e., DSD_{qb}) indirectly (see lines 7 – 9 of Example 23) via `qb:dataSet` in line 5.

Analogously, we next define the output QB4OLAP graph G_{qb4o} in terms of sets of triples.

Definition 2
A QB4OLAP graph G_{qb4o} is defined as follows.

- $G_{qb4o} = S_{qb4o} \cup I_{qb4o}$, where S_{qb4o} and I_{qb4o} are sets of triples that define the QB4OLAP cube schema and instances, respectively.
- $S_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup D_{qb4o} \cup M_{qb4o} \cup L_{qb4o} \cup LA_{qb4o} \cup H_{qb4o} \cup HS_{qb4o} \cup AF_{qb4o} \cup C_{qb4o}$, where the subsets of triples are:
  - DS_{qb4o} defining the cube data set;
  - DSD_{qb4o} defining the cube structure;
  - D_{qb4o} defining the cube dimensions;
  - M_{qb4o} defining the cube measures;
  - L_{qb4o} defining the dimension levels;
  - LA_{qb4o} defining the dimension level attributes;
  - H_{qb4o} defining the dimension hierarchies;
  - HS_{qb4o} defining the hierarchy steps;
  - AF_{qb4o} a predefined set of aggregate functions; and
  - C_{qb4o} a predefined set of possible cardinalities.
- $I_{qb4o} = LI_{qb4o} \cup O_{qb4o} \cup LAI_{qb4o}$, where LI_{qb4o} is the subset of triples that defines level instances and rollup relationship instances between child and parent level instances, O_{qb4o} is the subset of triples that defines the observations, and LAI_{qb4o} is the subset of triples that defines level attribute values.

Patterns of triples and examples of QB4OLAP sets are presented below in the methodology step definitions. The examples are based on the running example (see Section 2.3). By using this formalization we define our methodology considering the scenario where the input data set contains implicit MD semantics (e.g., a country is linked to a region) that is not explicitly stated, i.e., without the semantics that this is a rollup relationship in an MD hierarchy between a country level and a region level. Other scenarios are discussed in Section 5.4.

1.2 Redefinition of the cube schema

We start by building the new cube schema S_{qb4o}. For simplicity of presentation, when defining the steps we assume that from the input QB graph G_{qb}, we build a *new* QB4OLAP graph G_{qb4o}. We proceed incrementally and in the first step we build the schema S^1_{qb4o} (the complete cube schema S_{qb4o} is the output of Step 3). S^1_{qb4o} contains the sets DS_{qb4o} (defining the cube data set), DSD_{qb4o} (defining the new cube schema structure), L_{qb4o} (defining the QB4OLAP dimension levels), and M_{qb4o} (defining the measures). We populate these sets from S_{qb}, starting from the dimensions in D_{qb} that are redefined as levels in L_{qb4o}. Next, we copy the measures from M_{qb} to M_{qb4o}. Then, we define the new cube schema structure, assign it both levels and measures,

and add it to the DSD_{qb4o}. Finally, we copy the data set definition to DS_{qb4o} and assign the new cube schema structure to it. The details of the step are presented below.

Step 1. ***Redefinition of the cube schema:***
INPUT: S_{qb}
OUTPUT: S^1_{qb4o}

Step 1.1. *Redefine input dimensions as levels*:

- $L_{qb4o} \cup= \{createLevel(d), d \in D_{qb}\}$, where *createLevel* is a function redefining a dimension d as a level l, i.e., taking as argument a triple defining a *qb:DimensionProperty*, and producing a triple defining a *qb4o:LevelProperty*.

Triples pattern added to L_{qb4o}:
`qbDimensionIRI a qb4o:LevelProperty`, where `qbDimensionIRI` is the existing IRI of the QB dimension redefined as a QB4OLAP level. For instance, a triple related to the running example:

```
1 sdmx-dimension:refArea a qb4o:LevelProperty .
```

Step 1.2. *Copy input measures*:

- $M_{qb4o} \cup= \{m,\ m \in M_{qb}\}$, where m is a measure triple in M_{qb} that is added to M_{qb4o}. These triples are defining instances of the class *qb:MeasureProperty*.

Triples pattern added to M_{qb4o}:
`qbMeasureIRI a qb:MeasureProperty`, where `qbMeasureIRI` is the existing IRI of the QB measure copied. For instance, a triple related to the running example:

```
1 sdmx-measure:obsValue a qb:MeasureProperty .
```

Step 1.3. *Define the new cube schema structure and add to it levels and measures as components*:

- $DSD_{qb4o} \cup= \{createDSD()\}$, where *createDSD* is a function that creates a new cube schema structure triple, i.e., defining a new *qb:DataStructureDefinition*.

Triples pattern added to DSD_{qb4o}:
`dsdIRI a qb:DataStructureDefinition`, where `dsdIRI` is the newly defined IRI of the new schema structure definition. For instance, a triple related to the running example:

```
1 newG:newDSD a qb:DataStructureDefinition .
```

- $DSD_{qb4o} \cup= \{createComponent(lm), lm \in L_{qb4o} \cup M_{qb4o}\}$, where *createComponent* is a function that creates a schema structure component (in this case, a blank) node to which a level or measure is related. It receives a triple lm

defining a *qb4o:LevelProperty* or a *qb:MeasureProperty* and produces a triple c defining a *qb:ComponentProperty*.

Triples pattern added to DSD_{qb4o}:
`dsdIRI qb:component [qb4o:level qbDimensionIRI]; dsdIRI qb:component [qb:measure qbMeasureIRI]`. Here, `dsdIRI`, `qbDimensionIRI`, and `qbMeasureIRI` are the IRIs previously introduced. Note that we include dsdIRI for better understanding. For instance, triples related to the running example:

```
1 newG:newDSD qb:component  [ qb4o:level sdmx-dimension:refArea ] ;
2             qb:component  [ qb:measure sdmx-measure:obsValue ] .
```

Step 1.4. *Create the QB4OLAP cube schema*:

- $S^1_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup L_{qb4o} \cup M_{qb4o}$. Initially, we add to DS_{qb4o} the data set definition triple and the triple linking the data set to the DSD. Then, S^1_{qb4o} represents a union of DS_{qb4o} and the previous sets (i.e., DSD_{qb4o}, L_{qb4o}, and M_{qb4o}) with no additional triples pattern.

Triples pattern added to DS_{qb4o}:
`dsdIRI a qb:DataSet` and `dsIRI qb:structure dsdIRI`, where `dsIRI` and `dsdIRI` are the IRIs of the data set and the new schema structure definition, respectively. For instance, triples related to the running example:

```
1 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
2                                qb:structure newG:newDSD .
```

Thus, at this point, we have obtained S^1_{qb4o}.

Triple examples of S^1_{qb4o} are summed up in Example 25 to illustrate the overall result of Step 1. Note that we use the `newG` namespace for the new graph G_{qb4o}.

Example 25
Resulting triples of Step 1.

```
1 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
2                                qb:structure newG:newDSD .
3 newG:newDSD a qb:DataStructureDefinition ;
4   qb:component  [ qb4o:level sdmx-dimension:refArea ] ;
5   qb:component  [ qb:measure sdmx-measure:obsValue ] .
6 sdmx-dimension:refArea a qb4o:LevelProperty .
7 sdmx-measure:obsValue a qb:MeasureProperty .
```

Lines 1 and 2 illustrate the output of **Step** 1.4. **Step** 1.3. results in lines 3, 4, and 5 define the new cube schema structure and add a level and a measure as components to it. Line 6 redefines the dimension from Example 23 as a QB4OLAP level (result of **Step** 1.1.), while line 7 illustrates the measure from Example 23 copied in **Step** 1.2.

1.3 Specification of the aggregate functions

In this step we perform the first QB4OLAP enrichment by specifying an aggregate function for each measure. Note that possible aggregate functions

are predefined by QB4OLAP. The inputs of this step are S^1_{qb4o} and a mapping *MAggMap* from a measure $m \in M_{qb4o}$ to an aggregate function $a \in AF_{qb4o}$ (introduced in Section 4.2). The aggregate function is specified as a triple that relates a with the component of the cube schema structure related to m. This triple is added to DSD_{qb4o} and this enrichment is represented as the updated cube schema S^2_{qb4o} that is the step output. The details of the step are presented below.

Step 2. ***Specification of the aggregate functions***
INPUT: S^1_{qb4o}, *MAggMap*
OUTPUT: S^2_{qb4o}

Step 2.1. *Specify an aggregate function for each measure component of the cube schema structure*:

- $DSD_{qb4o} \cup= \{addAggFunction(getComponent(m), MAggMap(m)), m \in M_{qb4o}\}$, where:

– *getComponent* is a function that, given a measure m, returns the schema structure component c related to it (i.e., m is a triple defining a *qb:MeasureProperty* and c is a triple defining a *qb:ComponentProperty* (typically a blank node)).

– *MAggMap* is a mapping function from an input measure m (same as above) to the aggregate function a (i.e., a is a triple defining a *qb4o:AggregateFunction* (a predefined QB4OLAP aggregate function)).

– *addAggFunction* is a function that links an aggregate function a to the corresponding component c (a and c are the ones defined above).

Triples pattern added to DSD_{qb4o}:
`dsdIRI qb:component [qb:measure qbMeasureIRI; qb4o:aggregateFunction afIRI]`, where `dsdIRI` and `qbMeasureIRI` are the IRIs previously introduced and `afIRI` is the IRI of the aggregate function. Note that only `qb4o:aggregateFunction afIRI` is new in the pattern while the rest of the pattern refers to the earlier defined triples. For instance, triples related to the running example:

```
1 newG:newDSD qb:component
2         [ qb:measure sdmx-measure:obsValue ;
3         qb4o:aggregateFunction qb4o:Sum ] .
```

Step 2.2. *Create new partial cube schema*:

- $S^2_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup L_{qb4o} \cup M_{qb4o}$. S^2_{qb4o} represents a union of updated DSD_{qb4o} and the DS_{qb4o}, L_{qb4o}, and M_{qb4o} sets with no additional triples pattern.

Triple examples of S^2_{qb4o} are presented in Example 26. It is a follow-up of the previous example (i.e., Example 25).

Example 26
Resulting triples of Step 2.

```
1 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
2 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> qb:structure newG:newDSD .
3 newG:newDSD a qb:DataStructureDefinition ;
4   qb:component  [ qb4o:level sdmx-dimension:refArea ] ;
5         qb:component  [ qb:measure sdmx-measure:obsValue ;
6                         qb4o:aggregateFunction qb4o:Sum ] .
7 sdmx-dimension:refArea a qb4o:LevelProperty .
8 sdmx-measure:obsValue a qb:MeasureProperty .
```

Line 6 presents the aggregate function that is assigned to a measure by the grouping mechanism via a blank node. We use the SUM (i.e., `qb4o:Sum`) aggregate function as an example, while in general it depends on the input mapping (i.e., *MAggMap*).

1.4 Definition of the dimension hierarchies

We now construct the dimension hierarchies. As explained in Section 3, QB4OLAP reuses the `qb:DimensionProperty`, however with different semantics than in QB: while in the latter, a dimension represents a point at a fixed granularity, QB4OLAP considers a dimension to contain points at different granularities. Therefore, in QB4OLAP, a QB dimension becomes a dimension level (see Step 1) and a dimension represents a set of levels that are hierarchically organized. Thus, *the final cube schema* S_{qb4o} is created by updating S^2_{qb4o} and adding of additional sets of triples defining the new semantics. For their definition we use certain additional inputs about the structure of dimensions (e.g., about new dimension levels and dimension hierarchies). In the preface of this Appendix we have shown how to compute the inputs to this step.

The general process is as follows. First, we update the set of levels L_{qb4o} obtained in Step 1, with the new levels that will define the dimension hierarchies. These new levels are specified in *NewLevSet* as a set of IRIs. Each level can have one or more level attributes defined in the LA_{qb4o} set of triples (see Definition 2) added to the final cube schema S_{qb4o}. They are specified by the input set of IRIs *LAttSet*. Once we have all the levels and their attributes, we use the input *DimSet* (a set of dimension IRIs) to create the new set of dimensions D_{qb4o} in the complete cube schema S_{qb4o}. Then, the set of hierarchies H_{qb4o} that represents the hierarchically ordered levels in the dimensions is added to S_{qb4o}. For this, the input *HierSet* (a set of hierarchy IRIs) is used. Finally, we add to S_{qb4o} the set of hierarchy steps HS_{qb4o} that represents parent-child relationships between two levels. For this, we use the input *HStepSet*, a set of hierarchy step IDs.

Once levels, level attributes, dimensions, hierarchies, and hierarchy steps are defined, we must correlate them using the QB4OLAP properties. For this, we use the mappings *MapLAttribute2L* through *MapHStep2C* in the input list (see the Appendix preface). The *MapLAttribute2L* mapping associates level

attributes with levels (defining both, the "direct" association, and its "inverse", e.g., telling that a level has an attribute, and that an attribute belongs to a dimension level, respectively); *MapH2D* links hierarchies with dimensions; *MapH2L* links hierarchies with levels; *MapHStep2ParentL* associates a hierarchy step with its parent level, while *MapHStep2ChildL* does the same with its child level. Finally, *MapHStep2H* associates a hierarchy step with the hierarchy and *MapHStep2C* links a hierarchy step with its corresponding cardinality.

The output of this step is *the complete cube schema* S_{qb4o}. The details of the step are presented below.

Step 3. **Definition of the dimension hierarchies**
INPUT: S^2_{qb4o}, *NewLevSet*, *DimSet*, *HierSet*, *LAttSet*, *HStepSet*, *MapLAttribute2L*, *MapH2D*, *MapH2L*, *MapHStep2ParentL*, *MapHStep2ChildL*, *MapHStep2H*, *MapHStep2C*
OUTPUT: S_{qb4o}

Step 3.1. *Define new levels*:

• $L_{qb4o} \cup= \{createNewLevel(newL), newL \in NewLevSet\}$, where *createNewLevel* is a function that, for each level IRI in *NewLevSet*, produces the corresponding triple using the *qb4o:LevelProperty*.

Triples pattern added to L_{qb4o}:
`lIRI a qb4o:LevelProperty`, where `lIRI` is the IRI of the new level. For instance, a triple related to the running example:

```
1 newG:region a qb4o:LevelProperty .
```

Step 3.2. *Define level attributes*:

• $LA_{qb4o} \cup= \{createLevelAttribute(attr), attr \in LAttSet\}$, where *createLevelAttribute* is a function that, for each attribute IRI in *LAttSet*, produces the corresponding triple using the *qb4o:LevelAttribute*.

Triples pattern added to LA_{qb4o}:
`laIRI a qb4o:LevelAttribute`, where `laIRI` is the IRI of the new level attribute. For instance, a triple related to the running example:

```
1 newG:label a qb4o:LevelAttribute .
```

Step 3.3. *Define dimensions*:

• $D_{qb4o} \cup= \{createDimension(dim), dim \in DimSet\}$, where *createDimension* is a function that, for each dimension IRI in *DimSet*, produces the corresponding triple using the *qb:DimensionProperty*.

Triples pattern added to D_{qb4o}:
`dIRI a qb:DimensionProperty`, where `dIRI` is the IRI of the new dimension. For instance, a triple related to the running example:

```
1 newG:geoDimension a qb:DimensionProperty .
```

Step 3.4. *Define hierarchies*:

• $H_{qb4o} \cup= \{createHierarchy(hier), hier \in HierSet\}$, where *createHierarchy* is a function that, for each hierarchy IRI in *HierSet*, produces the corresponding triple using the *qb4o:Hierarchy*.

Triples pattern added to H_{qb4o}:
`hIRI a qb4o:Hierarchy`, where `hIRI` is the IRI of the new hierarchy. For instance, a triple related to the running example:

```
1 newG:geoHierarchy a qb4o:Hierarchy .
```

Step 3.5. *Define hierarchy steps*:

• $HS_{qb4o} \cup= \{createHStep(hStep), hStep \in HStepSet\}$, where *createHStep* is a function that, for each hierarchy step ID in *HStepSet*, produces the corresponding triple using the *qb4o:HierarchyStep*. Since QB uses blank nodes for representing reification, we follow the same principle and use blank node identifiers for hierarchy steps in the examples.

Triples pattern added to HS_{qb4o}:
`hsID a qb4o:HierarchyStep`, where `hsID` is the ID of the new hierarchy step. For instance, a triple related to the running example:

```
1 _:newHierarchyStep a qb4o:HierarchyStep .
```

Step 3.6. *Associate dimension levels with level attributes*:

• $L_{qb4o} \cup= \{linkLevelWithAttr(MapLAttribute2L(la), la), la \in LA_{qb4o}\}$, where *linkLevelWithAttr* is a function that receives a pair (l, la), where l is an instance of *qb4o:LevelProperty* and *la* is an instance of *qb4o:LevelAttribute*, and produces a triple *lla* telling that the level l has the level attribute *la*. *MapLAttribute2L* is a mapping that, given a level attribute, returns the level to which the level attribute belongs.

Triples pattern added to L_{qb4o}:
`lIRI qb4o:hasAttribute laIRI`, where `lIRI` and `laIRI` are the IRIs of the level and level attribute, respectively. For instance, a triple related to the running example:

```
1 newG:region qb4o:hasAttribute newG:label .
```

• $LA_{qb4o} \cup= \{linkAttrWithLevel(la, MapLAttribute2L(la)), la \in LA_{qb4o}\}$, where *linkAttrWithLevel* is a function that receives a pair (la, l), where *la* and l have the same meaning as above, and produces a triple *lal* telling that the level attribute *la* belongs to the level l.

Triples pattern added to LA_{qb4o}:
`laIRI qb4o:inLevel lIRI`, where `laIRI` and `lIRI` are the IRIs of the level

attribute and level, respectively. For instance, a triple related to the running example:

```
newG:label qb4o:inLevel newG:region .
```

Step 3.7. *Associate dimensions with hierarchies*:

• $D_{qb4o} \cup= \{linkDimWithHier(MapH2D(h), h), h \in H_{qb4o}\}$, where *linkDimWithHier* is a function that receives a pair (d, h), where d is an instance of *qb:DimensionProperty* and h is an instance of *qb4o:Hierarchy*, and produces a triple dh telling that the dimension d has the hierarchy h. *MapH2D* is a mapping that, given a hierarchy, returns the dimension to which the hierarchy belongs.

Triples pattern added to D_{qb4o}:
`dIRI qb4o:hasHierarchy hIRI`, where `dIRI` and `hIRI` are the IRIs of the dimension and hierarchy, respectively. For instance, a triple related to the running example:

```
newG:geoDimension qb4o:hasHierarchy newG:geoHierarchy .
```

• $H_{qb4o} \cup= \{linkHierWithDim(h, MapH2D(h)), h \in H_{qb4o}\}$, where *linkHierWithDim* is a function that receives a pair (h, d), where h and d have the same meaning as above, and produces a triple hd telling that the hierarchy h belongs to the dimension d.

Triples pattern added to H_{qb4o}:
`hIRI qb4o:inDimension dIRI`, where `hIRI` and `dIRI` are the IRIs of the hierarchy and dimension, respectively. For instance, a triple related to the running example:

```
newG:geoHierarchy qb4o:inDimension newG:geoDimension .
```

Step 3.8. *Associate hierarchies with levels*:

• $H_{qb4o} \cup= \{linkHierWithL(h, MapH2L(h)), h \in H_{qb4o}\}$, where *linkHierWithL* is a function that receives a pair (h, l), where h is an instance of *qb4o:Hierarchy* and l is an instance of *qb4o:LevelProperty*, and produces a triple hl telling that the hierarchy h has the level l. *MapH2L* is a mapping that, given a hierarchy, returns the level(s) it contains.

Triples pattern added to H_{qb4o}:
`hIRI qb4o:hasLevel lIRI`, where `hIRI` and `lIRI` are the IRIs of the hierarchy and level, respectively. For instance, triples related to the running example:

```
newG:geoHierarchy qb4o:hasLevel newG:region .
newG:geoHierarchy qb4o:hasLevel sdmx-dimension:refArea .
```

Step 3.9. *Associate hierarchy steps with the hierarchy, levels, and cardinalities*:

• $HS_{qb4o} \cup= \{linkHStepWithH(hs, MapHStep2H(\ hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithH* is a function that receives a pair (hs, h), where *hs* is an instance of *qb4o:HierarchyStep* and *h* is an instance of *qb4o:Hierarchy*, and produces a triple *hsh* telling that the hierarchy step *hs* belongs to the hierarchy *h*. *MapHStep2H* is a mapping that, given a hierarchy step, returns the related hierarchy.

Triples pattern added to HS_{qb4o}:
`hsID qb4o:inHierarchy hIRI`, where `hsID` and `hIRI` are the hierarchy step ID and the hierarchy IRI, respectively. For instance, a triple related to the running example:

```
1 _:newHierarchyStep qb4o:inHierarchy newG:geoHierarchy .
```

• $HS_{qb4o} \cup= \{linkHStepWithParentL(hs, MapHStep2ParentL(hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithParentL* is a function that receives a pair (hs, pl), where *hs* is an instance of *qb4o:HierarchyStep* and *pl* is an instance of *qb4o:LevelProperty*, and produces a triple *hspl* telling that the hierarchy step *hs* has *pl* as parent level. *MapHStep2ParentL* is a mapping that, given a hierarchy step, returns the related parent level.

Triples pattern added to HS_{qb4o}:
`hsID qb4o:parentLevel plIRI`, where `hsID` and `plIRI` are the hierarchy step ID and the parent level IRI, respectively. For instance, a triple related to the running example:

```
1 _:newHierarchyStep qb4o:parentLevel newG:region .
```

• $HS_{qb4o} \cup= \{linkHStepWithChildL(hs, MapHStep2ChildL(hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithChildL* is a function that receives a pair (hs, cl), where *hs* is an instance of *qb4o:HierarchyStep* and *cl* is an instance of *qb4o:LevelProperty*, and produces a triple *hscl* telling that the hierarchy step *hs* has *cl* as child level. *MapHStep2ChildL* is a mapping that, given a hierarchy step, returns the related child level.

Triples pattern added to HS_{qb4o}:
`hsID qb4o:childLevel clIRI`, where `hsID` and `clIRI` are the hierarchy step ID and the child level IRI, respectively. For instance, a triple related to the running example:

```
1 _:newHierarchyStep qb4o:childLevel sdmx-dimension:refArea .
```

• $HS_{qb4o} \cup= \{linkHStepWithC(hs, MapHStep2C(\ hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithC* is a function that receives a pair (hs, c), where *hs* is an instance of *qb4o:HierarchyStep* and *c* is an instance of *qb4o:Cardinality*, and produces a triple *hsc* telling that the hierarchy step *hs* has *c* as cardinality. *MapHStep2C* is a mapping that, given a hierarchy step, returns the related cardinality.

Triples pattern added to HS_{qb4o}:
`hsID qb4o:pcCardinality cIRI`, where `hsID` and `cIRI` are the hierarchy

step ID and the cardinality IRI, respectively. For instance, a triple related to the running example:

```
1 _:newHierarchyStep qb4o:pcCardinality qb4o:ManyToOne .
```

Step 3.10. *Create the complete cube schema*:

• $S_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup D_{qb4o} \cup M_{qb4o} \cup L_{qb4o} \cup LA_{qb4o} \cup H_{qb4o} \cup HS_{qb4o} \cup AF_{qb4o} \cup C_{qb4o}$. S_{qb4o} represents a union of all its subsets with no additional triples pattern. Note that AF_{qb4o} and C_{qb4o} are predefined by QB4OLAP.

As we have already said, the output of this step is the complete cube schema S_{qb4o}. Triple examples of new triples in S_{qb4o} are summed up in Example 27.

Example 27
Resulting triples of Step 3.

```
1          #Create a level, a level attribute, a dimension, a hierarchy,
2          #and a hierarchy step
3         newG:region a qb4o:LevelProperty .
4         newG:label a qb4o:LevelAttribute .
5         newG:geoDimension a qb:DimensionProperty .
6         newG:geoHierarchy a qb4o:Hierarchy .
7         _:newHierarchyStep a qb4o:HierarchyStep .
8
9          #Link the level and level attribute
10        newG:region qb4o:hasAttribute newG:label .
11        newG:label qb4o:inLevel newG:region .
12
13         #Link dimensions and hierarchies
14        newG:geoDimension qb4o:hasHierarchy newG:geoHierarchy .
15        newG:geoHierarchy qb4o:inDimension newG:geoDimension .
16
17         #Link the hierarchy and levels, and hierarchy step
18         # with all related instances
19        newG:geoHierarchy qb4o:hasLevel newG:region .
20        newG:geoHierarchy qb4o:hasLevel sdmx-dimension:refArea .
21        _:newHierarchyStep qb4o:inHierarchy newG:geoHierarchy .
22        _:newHierarchyStep qb4o:parentLevel newG:region .
23        _:newHierarchyStep qb4o:childLevel sdmx-dimension:refArea .
24        _:newHierarchyStep qb4o:pcCardinality qb4o:ManyToOne .
```

Line 3 shows a triple defining a new level, as a result of **Step** 3.1. Analogously, line 4 shows a triple defining a new level attribute as a result of **Step** 3.2. Lines 5 and 6, define a dimension and a hierarchy, resulting from applying **Step** 3.3. and **Step** 3.4., respectively. Then, line 7 defines a hierarchy step as a result of **Step** 3.5. Lines 10 – 11 link a level and a level attribute as the product of **Step** 3.6., and lines 14 – 15 link a dimension and a hierarchy showing the result of **Step** 3.7. Finally, lines 19 – 20 associate a hierarchy with its levels as results of **Step** 3.8. while lines 21 – 24 link a hierarchy step and its hierarchy, levels, and cardinality as the triples produced in **Step** 3.9.

1.5 Annotation of the cube instances

Once the new cube schema is defined, we need to link the level members with their schema definitions, i.e., populate the dimension level instances. Moreover, we link the level members with their level attribute instances and copy the observations to the new QB4OLAP graph. The first input of the step is *MapChild2Parent* that maps child level members to their parent level members. Next input is *MapLInstance2L* that maps level members to their levels. The *MapLInstance2LAInstance* input maps level members and level attribute instances. Then, the set of cube instances $I_{qb} = DI_{qb} \cup O_{qb}$, where DI_{qb} defines dimension instances and O_{qb} defines observations. The last input is the complete new cube schema S_{qb4o}. The output of the step is the set of new cube instances $I_{qb4o} = LI_{qb4o} \cup O_{qb4o} \cup LAI_{qb4o}$. The details of the step are presented below.

Step 4. **Annotation of the cube instances**
INPUT: MapChild2Parent, MapLInstance2L, MapLInstance2LAInstance, I_{qb}, S_{qb4o}
OUTPUT: I_{qb4o}

Step 4.1. *Copy dimension instances from* G_{qb} *as base level instances in* G_{qb4o}:

• $LI_{qb4o} \cup= \{copyAsLevelInstance(di), di \in DI_{qb}\}$, where *copyAsLevelInstance* is a function that receives a triple *di* representing a dimension instance in QB and returns a triple *li* defining the subject (i.e., an IRI) of the triple *di* as a level instance in QB4OLAP, using the *qb4o:LevelMember* property.

Triples pattern added to LI_{qb4o}:
`liIRI a qb4o:LevelMember`, where `liIRI` is the IRI of the level instance obtained from *di*. For instance, a triple related to the running example:

```
1 country:RS a qb4o:LevelMember .
```

Step 4.2. *Add coarser granularity level instances*:

• $LI_{qb4o} \cup= \{MapChild2Parent(li), li \in LI_{qb4o}\}$, where *MapChild2Parent* is a mapping that, for a given dimension level instance, returns its corresponding parent level member. This object is then defined as a level instance, using the *qb4o:LevelMember* property.

Triples pattern added to LI_{qb4o}:
`liIRI a qb4o:LevelMember`, where `liIRI` is the IRI of the new level instance, returned by *MapChild2Parent*. For instance, a triple related to the running example:

```
1 region:ECS a qb4o:LevelMember .
```

Step 4.3. *Copy observations*:

• $O_{qb4o} \cup= \{o, o \in O_{qb}\}$, where *o* is an observation from O_{qb} that is added to O_{qb4o}.

Triples pattern added to O_{qb4o}:
`oIRI a qb:Observation`, `oIRI qb:dataSet dsIRI`, `oIRI lIRI liIRI`, and `oIRI mIRI mvalue`, where `oIRI`, `dsIRI`, `lIRI`, `liIRI`, and `mIRI` are the IRIs of the observation, data set, level, level instance, and measure, respectively, while `mvalue` is a literal representing measure value. For instance, triples related to the running example:

```
<http://worldbank.270a.info/dataset/world-bank-indicators/
                                    CM.MKT.LCAP.CD/RS/2012>
  a qb:Observation ;
  qb:dataSet dataset:CM.MKT.LCAP.CD ;
  property:indicator indicator:CM.MKT.LCAP.CD ;
  sdmx-dimension:refArea country:RS ;
  sdmx-measure:obsValue 7450560827.04874 ;
```

Step 4.4. *Specify the level for each level instance*:

• $LI_{qb4o} \cup = \{linkToLevel(li, MapLInstance2L(li)), li \in LI_{qb4o}\}$, where *linkToLevel* is a function that receives a pair (li, l), where li is an instance of *qb4o:LevelMember* and l is an instance of *qb4o:LevelProperty*, and produces a triple *lil* telling that the level member li belongs to the level l. *MapLInstance2L* is a mapping that, given a level instance, returns the level it belongs to.

Triples pattern added to LI_{qb4o}:
`liIRI qb4o:memberOf lIRI`, where `liIRI` and `lIRI` are the IRIs of the level member and level, respectively. For instance, a triple related to the running example:

```
country:RS qb4o:memberOf sdmx-dimension:refArea .
```

Step 4.5. *Specify rollup (i.e., parent–child) relationships between level instances*:

• $LI_{qb4o} \cup = \{linkRollUps(MapChild2Parent(li), li), li \in LI_{qb4o}\}$, where *linkRollUps* is a function that receives a pair (pli, cli), where pli and cli are instances of *qb4o:LevelMember*, and produces a triple *pcli* telling that cli rolls-up to pli using the *skos:broader* property.

Triples pattern added to LI_{qb4o}:
`cliIRI skos:broader pliIRI`, where `cliIRI` and `pliIRI` are the IRIs of the child and parent level instances, respectively. For instance, a triple related to the running example:

```
country:RS skos:broader region:ECS .
```

Step 4.6. *Add level attribute instances*:

• $LAI_{qb4o} \cup = \{addLevelAttInstance(li, MapLInstance2LAInstance(li)), li \in LI_{qb4o}\}$, where *addLevelAttInstance* is a function that receives li and a pair (la, lai), where li is an instance of *qb4o:LevelMember*, la is an instance of *qb4o:LevelAttribute*, and lai is a level attribute value (IRI or literal), and produces a

triple *lilalai* telling that *li* has an attribute *la* with the value *lai*. *MapLInstance2LAInstance* is a mapping that, for a given dimension level instance, returns its level attribute–level attribute value pair(s).

Triples pattern added to LAI_{qb4o}:
`liIRI laIRI laiIRI` or `liIRI laIRI laiLiteral`, where `liIRI`, `laIRI`, and `laiIRI` are the IRIs of the level instance, level attribute, and level attribute value, respectively, and `laiLiteral` is a literal representing level attribute value. For instance, a triple related to the running example:

```
1 country:RS schema:capital ''Belgrade''^^xsd:string .
```

Step 4.7. *Create new cube instances*:

• $I_{qb4o} = LI_{qb4o} \cup O_{qb4o} \cup LAI_{qb4o}$. I_{qb4o} represents a union of LI_{qb4o} (i.e., the level members), O_{qb4o} (i.e., observations), and LAI_{qb4o} (i.e., the level attribute instances) with no additional triples pattern.

The output of this step is I_{qb4o}. Triple examples of I_{qb4o} are summed up in Example 28. This example follows our running example and is an extension of previous ones.

Example 28
Resulting triples of Step 4.

```
1  country:RS a qb4o:LevelMember .
2  region:ECS a qb4o:LevelMember .
3  <http://worldbank.270a.info/dataset/world-bank-indicators/
4                                                CM.MKT.LCAP.CD/RS/2012>
5    a qb:Observation ;
6    qb:dataSet dataset:CM.MKT.LCAP.CD ;
7    property:indicator indicator:CM.MKT.LCAP.CD ;
8    sdmx-dimension:refArea country:RS ;
9    sdmx-measure:obsValue 7450560827.04874 ;
10
11 country:RS qb4o:memberOf sdmx-dimension:refArea .
12 country:RS skos:broader region:ECS .
13 country:RS schema:capital ''Belgrade''^^xsd:string .
```

Result examples of **Step** 4.1. and **Step** 4.2. are illustrated in lines 1 and 2, respectively. Lines 3 – 9 illustrate copying of the part of observation from Example 2 as result example of **Step** 4.3. Then, line 11 presents the result example of **Step** 4.4. Finally, line 12 illustrates the result example of **Step** 4.5. and line 13 the result example of **Step** 4.6.

Chapter 5

SM4MQ: A Semantic Model for Multidimensional Queries

The paper is to be submitted to a conference.

Abstract

On-Line Analytical Processing (OLAP) is a data analysis approach to support decision-making. On top of that, Exploratory OLAP is a novel initiative for the convergence of OLAP and the Semantic Web (SW). This convergence enables the use of OLAP techniques on external data, such as the SW, to analyze the publicly available data in a user-friendly manner. Moreover, OLAP approaches exploit different metadata artifacts (e.g., queries) to assist the user with the analysis. However, modeling and sharing of most of these artifacts are typically overlooked. Thus, in this paper we focus on the query metadata artifact in the Exploratory OLAP context. As OLAP is based on the underlying multidimensional (MD) data model we denote such queries as MD queries and propose SM4MQ: A Semantic Model for Multidimensional Queries. SM4MQ is an RDF-based formalization of MD queries and it captures semantics of the related OLAP operations at the conceptual level. Thus, it enables sharing and reuse of these queries on the SW. Furthermore, we propose a method to automate the exploitation of queries by means of SPARQL (the standard query language for RDF). We apply our method to a use case of transforming a query from SM4MQ to a vector representation that enables computing of their similarities (e.g., using cosine similarity). For this use case, we also developed a prototype and used a set of MD queries to perform an evaluation. This way, we exemplify practical benefits of using SM4MQ to automate exploitation of MD queries. Overall, this paper provides foundations

for the modeling and sharing of MD queries on the SW that as well facilitate their further processing, e.g., for user assistance purposes.

1 Introduction

On-Line Analytical Processing (OLAP) is a well-established approach for data analysis to support decision-making [2]. Due to its wide acceptance and successful use by non-technical users, novel tendencies endorse broadening of its use from solutions working with in-house data sources to analysis considering external and non-controlled data. A vision of such settings is presented as Exploratory OLAP [4] promoting the convergence of OLAP and the Semantic Web (SW). The SW provides a technology stack for publishing and sharing of data with their semantics and many public institutions, such as Eurostat, already use it to make their data publicly available. The Resource Description Framework (RDF) [28] is the backbone of the SW representing data as directed triples that form a graph where each triple has its semantics defined. Querying of RDF data is supported by SPARQL [82], the standard query language for RDF.

To facilitate data analysis, OLAP systems typically exploit different metadata artifacts (e.g., queries) to assist the user with analysis. However, although extensively used, little attention is devoted to these metadata artifacts [104]. This originates from traditional settings where very few (meta)data are open and/or shared. Thus, [104] proposes the Analytical Metadata (AM) framework, which defines AM artifacts such as schema and queries that are used for user assistance in settings such as Exploratory OLAP. In this context, analysis should be collaborative and therefore these metadata artifacts need to be open and shared among different systems. Thus, SW technologies are good candidates to model and capture these artifacts.

A first step for (meta)data sharing among different systems is to agree about (meta)data representation, i.e., modeling. As RDF uses a triple representation that is generic, the structure of specific (meta)data models is defined via RDF vocabularies providing semantics to interpret the (meta)data. Thus, the AM artifacts are modeled in [105] proposing the SM4AM metamodel. Due to the heterogeneity of systems, the metamodel abstraction level is used to capture the common semantics and organization of AM. Then, metadata models of specific systems are defined at the model level instantiating one or more AM artifacts. For instance, the schema artifact for Exploratory OLAP can be represented using the QB4OLAP vocabulary to conform data to a multidimensional (MD) data model for OLAP on the SW [106]. QB4OLAP further enables running of MD queries to perform OLAP on the SW [102]. However, the representation of these queries to support their sharing, reuse, and more extensive exploitation on the SW is yet missing. Thus, in the present paper

we propose a model for MD queries and explain how it not only supports sharing and reuse but can also be used to facilitate metadata processing, e.g., for user assistance exploitations such as query recommendations.

In particular, the contributions of this paper are:

- We propose *SM4MQ*: A Semantic Model for MD Queries formalized as an RDF-based representation of typical OLAP operations. The model captures the semantics of common OLAP operations at the conceptual level and supports their sharing and reuse via the SW.
- We define a method to automate the exploitation of *SM4MQ* queries by means of SPARQL. The method is exemplified on a use case to transform a query from *SM4MQ* to a vector representation. The use case shows an example of generating vectors (forming a matrix) as analysis-ready data structures that are typically used in recommender systems to compare different items [8] and existing approaches such as [25] use vectors for query recommendations.
- We developed a prototype and used a set of MD queries to evaluate our approach for the chosen use case. The evaluation shows that even non-technical users can conduct this task thanks to the automatically generated SPARQL queries based on the *SM4MQ* semantics.

The remainder of the paper is organized as follows. The next section explains the preliminaries of our approach. Then, Section 3 proposes the MD query model. Section 4 defines a method to automate the exploitation of *SM4MQ* queries and presents the use case of transforming these queries into a vector representation. Section 5 discusses the results of the performed evaluation. Finally, Section 6 discusses the related work and Section 7 concludes the paper.

2 Background

For understanding our approach, we introduce the necessary preliminaries and a running example used throughout the paper. First, we explain the MD model and the most popular OLAP operations. Then, we discuss the use of SW and QB4OLAP for MD models. The formalization of QB4OLAP concepts and OLAP operations can be found in [34] and in the present paper we provide the necessary intuition for understanding the proposed query model. The running example is incrementally introduced in each of the subsections.

2.1 Multidimensional Model and OLAP Operations

The MD model organizes data in terms of *facts*, i.e., data being analyzed, and *dimensions*, i.e., analytical perspectives [53]. Dimensions consist of *levels* rep-

resenting different data granularities that are hierarchically organized into dimension *hierarchies*. Levels can have *attributes* that further describe them. Facts contain *measures* that are typically numerical values being analyzed. Data conforming to an MD schema are referred to as a *data cube* that is being navigated (e.g., data granularity is changed) via OLAP operations. For instance, Figure 5.1 illustrates an MD schema (using DFM notation [39]) created for the European Union asylum applicants data set available in a Linked Data version of the Eurostat data[1]. In the data set, the number of asylum applications as a measure, can be analyzed according to the age, sex, type of application (Asyl_app in the figure), destination country (Geo in the figure), country of origin (Citizenship in the figure), and month of application (RefPeriod in the figure) levels of related dimensions. Furthermore, the data can be aggregated from months to quarters and likewise to years, from country of origin to continent, and from destination country to continent or government type as additional levels in related dimensions.

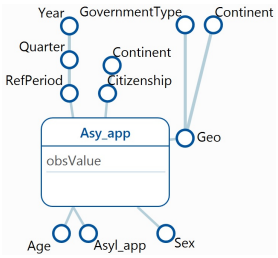

Fig. 5.1: Asylum Data set Schema

To navigate a data cube, OLAP operations are used and different OLAP algebras have been proposed as discussed in [84]. In the present paper, we consider the set of OLAP operations used in [34] and [32] that are defined at the conceptual level as discussed in [29]. The considered OLAP operations and their semantics are described in the following.

The *ROLL-UP* operation aggregates data from a finer granularity level to a coarser granularity level in a dimension hierarchy of a data cube. For instance, in case of the schema in Figure 5.1 data can be aggregated from the month level to the year level for the RefPeriod dimension. Similarly, the *DRILL-DOWN* operation as its inverse disaggregates data from a coarser granularity level to a finer granularity level in a dimension hierarchy of a data cube. Furthermore, the *DICE* operation takes a data cube and applies a boolean condition expressed over a level (attribute) and/or measure value over it. For instance, for the schema in Figure 5.1 a user may be interested in number of asylum applications only for the years 2009 and 2010. Finally, the *SLICE* operation removes a dimension or a measure from a data cube. For

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

instance, a user may not be interested in the age of the applicants and thus remove the related dimensions.

2.2 The Semantic Web Technologies

As mentioned in the introduction, the SW technologies provide means for flexible (meta)data representation and sharing. RDF that is the SW backbone, represents data in terms of directed subject - predicate - object triples that comprise an RDF graph where subjects and objects are nodes and predicates are edges. Subject and predicate are represented with IRIs, i.e., unique resource identifiers on the SW, while objects can either be IRIs or literal values. Furthermore, RDF supports representation of the data semantics via the `rdf:type` property. In the context of sharing, the Linked Data initiative [45] strongly motivates interlinking of RDF data on the SW to support identification of related / similar / same concepts. Finally, the RDF data can be queried with SPARQL [82], the standardized query language for RDF, which supports their systematic exploration.

To support the publishing of MD data and their OLAP analysis directly on the SW, two RDF vocabularies were proposed, namely the RDF Data Cube (QB) and QB4OLAP vocabularies. As the former vocabulary was primarily designed for statistical data sets, the latter one was proposed to extend QB with necessary concepts to fully support OLAP. A detailed discussion on this is presented in [106] where it is also explained how existing QB data sets can be enriched with the QB4OLAP semantics. Thus, in the present paper we consider QB4OLAP for the representation of the MD data on the SW and Figure 5.2 illustrates how the MD schema from Figure 5.1 can be represented with QB4OLAP. Note that for simplicity reasons we represent only the finest granularity levels.

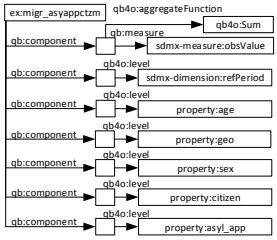

Fig. 5.2: Asylum Data set QB4OLAP Schema Representation

Once a data cube is published using QB4OLAP, the OLAP operations introduced in the previous subsection can be performed. However, a metadata model for representing these queries is yet missing. Such a model can be created by instantiating the SM4AM metamodel (see [105]). The metamodel represents the query AM artifact with several meta classes that we explain next.

First, the sm4am:UAList element is a complex element that combines atomic elements that include data exploration actions (i.e., sm4am:DataExploration-Action with its sm4am:ManipulationAction subclass). Thus, the metamodel elements can be instantiated as an MD query that combines OLAP operations and we present the details in the next section.

3 A Semantic Model for Multidimensional Queries

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

3.1 MD Query Model

Figure 5.3 illustrates the complete *SM4MQ* query model. Furthermore, the figure also shows how *SM4MQ* relates to the SM4AM metamodel concepts and reuses concepts from the QB, QB4OLAP, and RDFS vocabularies. The central concept of the model is sm4mq:Query representing a query. A query can be related to a simply ordered set of OLAP operations (i.e., subclasses of sm4mq:Operation via subproperties of sm4mq:hasOperation, namely sm4mq:-hasRollUp, sm4mq:hasDrillDown, sm4mq:hasDimDice, sm4mq:hasMeasDice, sm4-mq:hasDimSlice, and sm4mq:hasMeasSlice. OLAP operations are organized in a simply ordered set as each of them directly relates to the query and they are mutually ordered, e.g., the order of ROLL-UP and DRILL-DOWN operations is relevant to determine the final granularity of the data cube. Each operation relates to a data cube schema (i.e., qb:DataStructureDefinition) via sm4mq:over-Cube and to a data set (i.e., qb:DataSet) to which data belong to via sm4mq:-forDataSet. This way, operations belonging to a single query can operate over different schemata and data sets (inspired by the federated queries mechanism in SPARQL). In the next subsections, we explain each of the OLAP operations. Note that we follow the formalization given in [32] and we also enrich the model with additional information needed to facilitate sharing.

3.2 ROLL-UP and DRILL-DOWN Operations

Following the definition in [32], ROLL-UP (i.e., sm4mq:RollUp) is represented with a data cube schema (i.e., qb:DataStructureDefinition), a dimension (i.e., qb:DimensionProperty), and a level to roll-up to (i.e., qb4o:LevelProperty). In addition to these concepts, *SM4MQ* also represents the level from which the roll-up is performed, its order in the query, and the dimension hierarchy (i.e., qb4o:Hierarchy) used. The related properties are illustrated in Figure 5.3. In

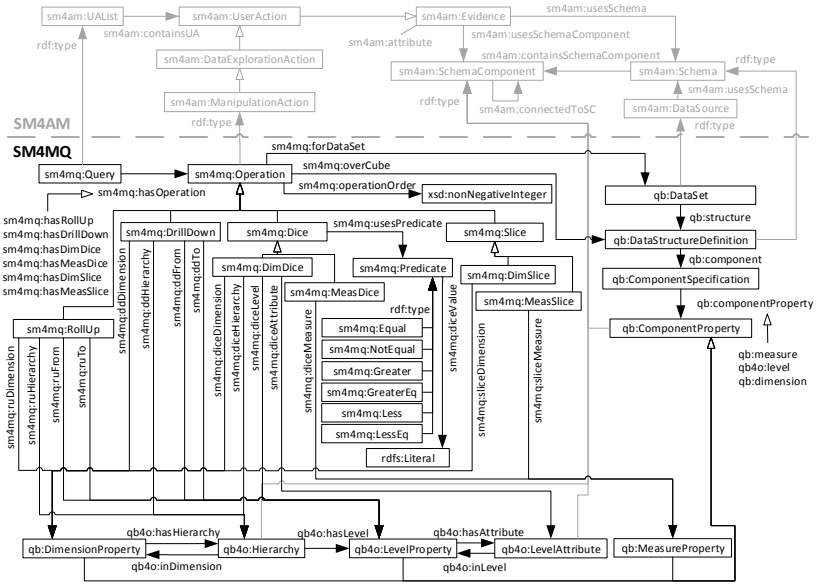

Fig. 5.3: A Semantic Model for Multidimensional Queries

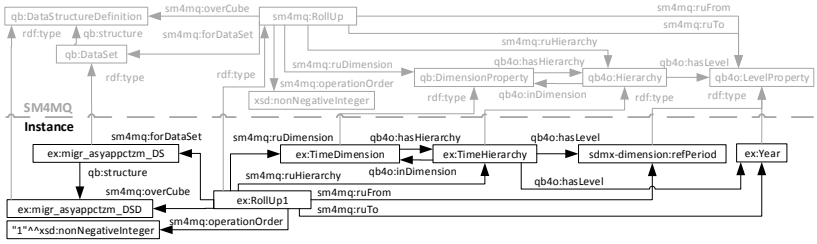

Fig. 5.4: ROLL-UP Instance Example

general, the roll-up from and hierarchy concepts can be inferred from a sequence of OLAP operations, however their explicit representations makes the ROLL-UP operation model self-contained such that it can be easily shared. The *SM4MQ* representation of the ROLL-UP example from Section 2 of aggregating data from the month to the year level over the running example schema is illustrated in Figure 5.4. The example shows the ROLL-UP instance and also includes the related *SM4MQ* concepts (depicted in gray). The top left unshaded triple shows the data set for ex:RollUp1 and the two triples below the data structure definition and the operation order, respectively. Moreover, the unshaded triples to the right from ex:RollUp1 show its dimension, hierarchy, and from and to levels, respectively.

Following the definition in [32], DRILL-DOWN (i.e., sm4mq:DrillDown)

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

3.3 DICE Operation

Following the definition in [32], DICE (i.e., sm4mq:Dice) is represented with a data cube schema (i.e., qb:DataStructureDefinition) and a boolean condition (i.e., sm4mq:Predicate) over a dimension (i.e., qb:DimensionProperty) or a measure (i.e., qb4o:MeasureProperty). We represent these two cases separately for the atomicity of operations that also facilitates sharing. Thus, in *SM4MQ* we create two subclasses for DICE, sm4mq:DimDice as DICE applied over a dimension and sm4mq:MeasDice as DICE applied over a measure. The former one relates to a dimension, hierarchy, level, and optionally level attribute, while the latter relates to a measure. For both cases we define the order and consider a set of relational predicates that includes equals to (i.e., sm4mq:Equal), not equals to (i.e., sm4mq:NotEqual), greater than (i.e., sm4mq:Greater), greater than or equal (i.e., sm4mq:GreaterEq), less than (i.e., sm4mq:Less), and less than or equal (i.e., sm4mq:LessEq). Each specific relational operator is an instance of the sm4mq:Predicate class (similarly to the case of aggregate functions in QB4OLAP), and it is related to rdfs:Literal used for the representation of the concrete values. The *SM4MQ* representation of the DICE example from Section 2 where a user is interested in number of asylum applications only for the years 2009 and 2010 for the running example schema (see Figure 5.1) is illustrated in Figure 5.5. The example shows the DICE instance and also includes the related *SM4MQ* concepts (depicted in gray). The top left unshaded triple shows the data set for ex:Dice1 and the two triples below the data structure definition and the operation order, respectively. Moreover, the unshaded triples to the right from ex:Dice1 show its dimension, hierarchy, level, and predicate, respectively. The two unshaded triples on the bottom right link the predicate with related values.

3.4 SLICE Operation

Following the definition in [32], SLICE (i.e., sm4mq:Slice) is represented with a data cube schema (i.e., qb:DataStructureDefinition) and a dimension (i.e., qb:DimensionProperty) or a measure (i.e., qb4o:MeasureProperty). Again, we represent these two cases separately for the atomicity of operations that also facilitates sharing. Thus, in *SM4MQ* we create two subclasses for SLICE,

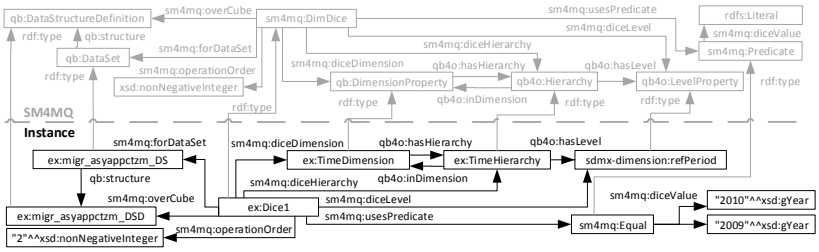

Fig. 5.5: DICE Instance Example

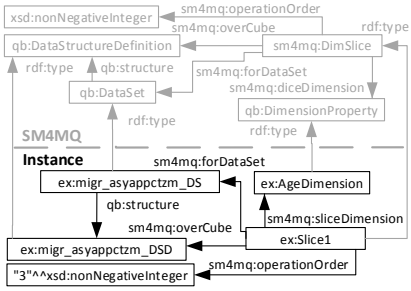

Fig. 5.6: SLICE Instance Example

sm4mq:DimSlice as SLICE applied over a dimension and sm4mq:MeasSlice as SLICE applied over a measure.For both cases we also define the order. The *SM4MQ* representation of the SLICE example from Section 2 where a user is not interested in the age of the applicants and thus removes the related dimensions for the running example schema (see Figure 5.1) is illustrated in Figure 5.6. The example shows the SLICE instance and also includes the related *SM4MQ* concepts (depicted in gray). The top left unshaded triple shows the data set for ex:Slice1 and the two triples below the data structure definition and the operation order, respectively. Moreover, the top right unshaded triple relates ex:Slice1 to the dimension.

4 Exploiting SM4MQ

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

4.1 Modeling and Semantics

We first explain the challenges behind the current state-of-the-art to represent queries and explain how *SM4MQ* overcomes them. MD queries in existing approaches are typically stored in query logs [104]. From logs, queries are parsed to extract semantics needed for further processing. The parsing is dependent on the particular technology used (e.g., SQL) where different patterns need to be applied to identify OLAP operations introduced in Section 2. Once identified, the OLAP operations are represented with internal data structures and any processing of the queries is directly dependent on these internals. In the context of Exploratory OLAP [4] and next generation BI systems [104], this situation leads to several challenges:

1. *Repetitive model designing of MD queries* – Instead of considering query metadata as a first-class citizen and conforming them to a dedicated model, repetitive efforts are invested into designing ad-hoc query representations for each system.

2. *Repetitive adjustments for exploitation* – The use of hard-coded and ad-hoc query models hinders the use of existing algorithms for their exploitation. As the internals on query representation are typically not available, existing algorithms need to each time be adjusted to a specific query model used in a system.

3. *Burdensome query sharing* – Overlooking of query modeling obstructs query reuse among different systems. This becomes especially relevant, considering Exploratory OLAP and public data sets on the SW where not only data but also queries can be shared among the users. Moreover, once modeled, queries can be made publicly available so that users can exploit them for different purposes.

4. *The need for IT people support* – Working with internal query representation requires technical skills that are not characteristic of OLAP end-users. Thus, preparing these queries (e.g., extracting the relevant semantics) requires the support of IT people. In the OLAP context, the data preparation by means of a correct ETL process may take up to 80% of the entire DW project as reported by Gartner [95]; illustrating the enormous efforts even from trained professionals.

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

1. *SM4MQ* is a model of MD queries covering the OLAP operations specified in Section 2 that are commonly accepted and used in OLAP systems. Furthermore, the use of RDF makes it flexible to be extended with

additional operations. Moreover, it can be linked with other RDF-based models using Linked Data principles (see Section 2).

2. Being an RDF-based model, *SM4MQ* provides a common semantics which exploitation algorithms can query in a standardized way via SPARQL.

3. The sharing of queries conforming to *SM4MQ* directly supports their sharing via Linked Data principles, i.e., publishing on the SW in the RDF format. This way, different systems can open their queries to become available.

4. The semantics of MD queries captured at the conceptual abstraction level is understandable even for non-technical OLAP end-users [34,106]. It can automatically be transformed into different data structures (e.g., vectors) via SPARQL, the standard query language for RDF [82], and thereby benefit from algorithms working over the related structures (e.g., computing cosine similarity between vectors).

Once MD queries are represented with *SM4MQ*, we can use off-the-shelf tools (i.e., RDF triple stores) to store and query them in a standardized manner by using SPARQL. For example, queries that contain the roll-up from Figure 5.4 can be retrieved with Query 1.

Query 1
Retrieve Queries Containing the Figure 5.4 Roll-up

```
SELECT DISTINCT ?q
WHERE {
 ?q rdf:type sm4mq:Query ;
    sm4mq:hasRollUp ?r .
 ?r rdf:type sm4mq:RollUp ;
    sm4mq:forDataSet ex:migr_asyappctzm_DS ;
    sm4mq:overCube ex:migr_asyappctzm_DSD ;
    sm4mq:ruDimension ex:TimeDimension ;
                sm4mq:ruHierarchy ex:TimeHierarchy ;
                sm4mq:ruFrom sdmx-dimension:refPeriod ;
                sm4mq:ruTo ex:Year .
}
```

We next propose a method to automatize transformations of queries or other metadata from our RDF-based model to other representations (i.e., models or structures) used for further exploitations, e.g., query comparison. The method is exemplified on a use case of creating a vector representation of the *SM4MQ* queries.

4.2 Automating SM4MQ Exploitation

SM4MQ supports automation of query exploitations in two ways. First, the MD semantics of queries can be directly retrieved via SPARQL, instead of extracting and parsing queries like in typical settings. Second, the conceptual abstraction level of OLAP operations in *SM4MQ* makes the model

understandable by non-technical OLAP users. Similar to how OLAP front-ends support the automation of queries exploring MD data based on the MD model, the *SM4MQ* model can support front-ends that automate exploration of MD queries. Thus, such front-ends release the user of even using SPARQL. For instance, an RDF-based model can be visualized as a graph and based on the selection of its elements (nodes or edges), SPARQL queries can be automatically generated to retrieve these elements and carry out automatic transformations.

Using these benefits, we next propose a method defining the steps to automate the transformation of queries from *SM4MQ* to other query representations. We also provide a use case where we exemplify our claims by transforming *SM4MQ* queries into analytical vectors that can be used to perform advanced analysis such as query comparison and undertake recommendations. Vector-based representations have been typically used to compute similarities (e.g., the cosine similarity) that have been widely used in recommender systems [8]. Therefore, several of the state-of-the-art query recommendation approaches such as [25] use vectors to represent queries and compute similarities (see [11]).

The method consists of the following tasks and subtasks:

1. Choosing the analytical structure,
2. Defining analytical features,
    a. Selecting model element(s) to form analytical features,
    b. Defining the level of detail for each analytical feature,
3. Populating the analytical structure,
    a. Retrieving model instances, and
    b. Computing values for each analytical feature.

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

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

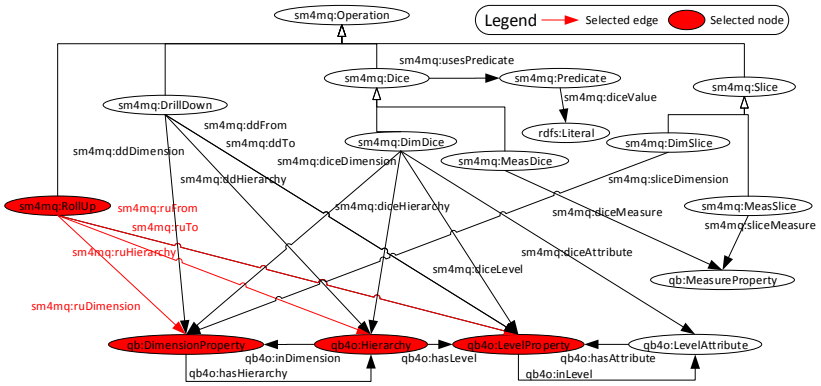

Fig. 5.7: Selection of a Composite Feature for ROLL-UP

Task 2a. Selecting model element(s) to form analytical features is the task where one or more model elements are selected to form an atomic or composite feature, respectively. Thus, it is performed for each analytical feature. *SM4MQ* can be visualized as graph and the user can select the model element(s). In this context, Figure 5.7 shows an example where the *SM4MQ* elements related to the subclasses of sm4mq:Operation are visualized as graph. Moreover, the red marked elements are select by the user to define the ROLL-UP operation as analytical feature. *Use Case.* Such analytical feature will be a part of the vector representation, i.e., will be represented with model elements, and this is precisely defined in the next subtask.

When defining an atomic or a composite analytical feature, the following reasoning applies:

- Starting from a chosen element of the model, in our case an MD query, visit all the adjacent successor nodes (i.e., adjacent neighbors reached via outgoing edges).
- If a successor node is to be considered alone, i.e., without its adjacent successor nodes, it is considered as an atomic feature.
- Otherwise, if a successor node is to be considered together with its successor nodes they form a composite feature. In this case, an edge used to reach a successor node identifies that successor node. Thus, a composite feature is a connected subgraph or the model graph.

Use Case. Analytical features will be a part of the vector representation, i.e., they will be represented with vector elements. The starting node to look for features in case of *SM4MQ* is sm4mq:Query. As the whole vector will represent a query, sm4mq:Query is not an analytical feature. Instead, the analytical features to be defined are different OLAP operations, i.e., the subclasses of sm4mq:Operation, with their successor nodes (e.g., schema elements such

Table 5.1: A Roll-up Piece of Vector Instance

ROLL-UP	D1	H1	RefPeriod	Quarter	Year	RefPeriod	Quarter	Year	D2	H2	Citizenship	Continent	Citizenship	Continent	...	D6	H7	Age	Age
1	1	1	1	0	0	0	0	1	0	0	0	0	0	0		0	0	0	0

as level). As we consider that all the queries are run over the same data cube, the data set and schema elements are omitted.

Task 2b. Defining the level of detail for each analytical feature relates to the model elements forming analytical features. For each analytical feature, the level of detail needs to be defined for each of the model elements forming that feature. There are two possible levels of detail. One is the *basic level of detail*, meaning that the model element in an analytical feature should be considered without its instances. The other option is the *comprehensive level of detail* where a model element in an analytical feature should be considered together with its instances. *Use Case.* In a vector representation, this means that a model element with the base level of detail takes a single vector element, e.g., if an operation is used or not. Accordingly, in case of a model element with the comprehensive level of detail, there is a vector element for each instance of the model element, e.g., one vector element for each possible dimension that can be used in an operation. This way, a vector length is defined by vector elements needed for the model elements in each analytical feature.

As an example, we next define the level of detail in case of the ROLL-UP operation as an analytical feature. Here, there should be a vector element indicating if there is any ROLL-UP in the query (i.e., the basic level of detail) and for each dimension used in one or more ROLL-UPs there should be a sequence of vector elements (due to the comprehensive level of detail) including: a vector element identifying the dimension, a vector element for each dimension hierarchy, a vector element for each possible from-level, and a vector element for each possible to-level. Note that the aggregate function used in ROLL-UP is defined by the data structure definition and thus the same for all queries. For example, Table 5.1 illustrates a part of vector instance for the running example schema related to the ROLL-UP operation from Figure 5.4. The non-zero values shown from the first vector element to right specify that it is a ROLL-UP operation, over the dimension D1 and hierarchy H1, from the RefPeriod level to the Year level.

Task 3. Populating the analytical structure focuses on taking a query instance in *SM4MQ* and populating the chosen analysis-ready data structures. This task depends on the analytical features definition from the previous task and can be automatized with SPARQL query templates. It consists of the two subtasks that we explain in the sequel. *Use Case.* For the vector representation, this means population of all vector element.

Task 3a. Retrieving model instances is the task where template SPARQL queries are defined to automatically retrieve the model elements instances related to the analytical features. There are two types of templates. One that retrieves all instances of the model elements that are nodes and belong to one or more analytical features. The related SPARQL query is shown in Query 2. Note that variables between two '?' are parameters that should be replaced with the related IRIs, e.g., node IRIs in the previous case. This way, all possible model element instances related to either atomic or composite analytical features are retrieved, i.e., the space of all possible model instances that can be used is defined.

Query 2
Retrieve Model Element Instaces

```
1 SELECT DISTINCT ?i
2 WHERE {
3  i? rdf:type ?nodeIRI? .
4 }
```

The other template retrieves the instances of graph patterns that includes both nodes and edges related to composite analytical features. This way, all model instances that are used in composite analytical features are retrieved. For instance, in case of the ROLL-UP elements selected in Figure 5.7, they can be retrieved with Query 3, where the ?q? parameter is a query IRI.

Query 3
Retrieve Roll-Ups for a Query

```
1  SELECT DISTINCT ?r ?d ?h ?fromL ?toL
2  WHERE {
3   ?q? rdf:type sm4mq:Query ;
4      sm4mq:hasRollUp ?r .
5   ?r rdf:type sm4mq:RollUp ;
6      sm4mq:ruDimension ?d ;
7                 sm4mq:ruHierarchy ?h ;
8                 sm4mq:ruFrom ?fromL ;
9                 sm4mq:ruTo ?toL .
10 }
```

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

Task 3b. Computing values for each analytical feature is the final task that takes the model instances previously retrieved and processes them according to the defined analytical features to populated the chosen analytical structure.

This processing can be simple and result with values 1 or 0 to show if a model element instance has been used or not in an analytical feature, or be a customized function. *Use Case.* To populate vector elements, we apply a simple computation and consider that the values of vector elements are 1 or 0 (meaning either the presence or absence of that element). Using this logic, lines 5 to 13 of Algorithm 3 populate the matrix as follows. For each query, a vector is created again based on the schema (see line 6), populated according to the OLAP operations used in the query (see lines 7 to 12), and added to the matrix (see line 13). The nested functions in lines 7 to 12 run SPARQL queries and return specific OLAP operations for an MD query. An example of a SPARQL query for *q.getRollUps*() used in line 7 is illustrated in Query 3. Finally, line 14 returns the resulting matrix. Thus, benefiting from the *SM4MQ* query modeling and semantics, the population of the matrix can be completely automatized using Algorithm 3 and encapsulated into high-level modules that even non-technical users may use.

Algorithm 3: Populate the matrix of queries

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

Once the matrix is populated with vector instances, it can be used to compute similarities (e.g., cosine similarity) between vectors. In the next section, we further discuss on this topic and provide examples of queries and their comparison results.

5 Evaluation

In this section, we present an evaluation of our approach. For this purpose, we implemented a prototype and used a set of 15 OLAP queries related to the running example data set. We next explain the details.

The prototype was developed in Java using JDK 8, Apache Jena 2.13.0 for working with RDF graphs, and Windows 8.1 as OS. It retrieves the (*SM4MQ*)

query and (QB4OLAP) schema metadata, transforms queries into vectors using Algorithm 3, and compares queries using the cosine similarity often used in this context as discussed in [11]. Assuming the existence of a single data set and its metadata on a SPARQL endpoint, it takes the endpoint address as a parameter and based on the *SM4MQ* and QB4OLAP semantics automatically retrieves the needed metadata to support creating a vectorized representation of the queries that will be used to compare queries.

We next discuss the user efforts required for this task. Considering the method in Section 4 the user needs to select features and define their level of detail, i.e., two user actions per model element. Thus, in the worst case, the user can select all the model elements and that can easily be performed by non-technical users. In the use case, the user needs 36 actions to define features representing OLAP operations. Next, the user introduces the SPARQL endpoint address and the vectors are automatically created and populated from the query models. We later discuss in more detail an example of how can these vectors be exploited. To illustrate the level of automation achieved in this process, we next discuss the number of automatically triggered queries by our tool. Once the user specified the SPARQL endpoint, our tool automatically triggered 91 SPARQL queries related to MD queries, 1 SPARQL query to retrieve all IRIs of MD queries and 6 SPARQL queries per MD query (one for each OLAP operation of *SM4MQ*). Furthermore, the tool also automatically triggered 32 SPARQL queries to retrieve the schema of the data set (the number of queries depends on the schema structure, e.g., how many dimension or hierarchies exist). To perform this, the execution of all the SPARQL queries took less than a second. Thus, our prototype automates, facilitates, and speeds up this process benefiting from the *SM4MQ* semantics for MD queries and also the QB4OLAP semantics for the MD schema, both being RDF-based models.

Furthermore, we next show an example of how these vectors can be exploited the tool computes the cosine similarity to detect similarities between queries (see [11]). In this context, we used 15 queries related the running example data set. The queries cover the following scenarios of comparing:

1. The same queries.
2. The queries that only differ in the filtering conditions.
3. The queries that only differ in the used granularities.
4. The queries that only differ in the used dimensions.
5. The queries that only differ in the OLAP operations used.
6. The queries that differ in both OLAP operations and granularities used.
7. The queries that differ in both OLAP operations and dimensions used.
8. Incrementally built queries.

Each scenario includes one or more queries and provides insights about how different changes in semantics influence the similarities of the queries com-

puted in this way. The query taken as a seed (i.e., Query 1) for all other queries considers *the total asylum applications submitted by the continent from which the applicant comes for year 2012, by sex, time, age, citizenship, and destination country*. Thus, starting from the initial cube, this query applies ROLL-UP from month to year over the time dimension, ROLL-UP from country of citizenship to the continent over the citizenship dimension, DICE to year 2012 over the time dimension, DRILL-DOWN back to citizenship country, and finally SLICE over the application type dimension. The results of the comparison of this query with other queries covering the scenarios are presented in Table 5.2. For each query, we provide a brief description of how it differs from Query 1.

Table 5.2: Query Similarities

Scenario	Query	Similarity
1	1. The same query	1
1	2. Different query with the same structure	1
2	3. Different filtering predicate	0.95
3	4. Time granularity changed	0.95
4	5. Dice over the geo instead of time dimension	0.81
5	6. Drill-Down changed to Dice	0.78
6	7. i) Time granularity changed and ii) Drill-Down changed to Dice	0.73
7	8. i) Dice over the geo and ii) Drill-Down changed to Dice	0.58
8	9. Only first operation of Query 1	0.49
8	10. First two operations of Query 1	0.65
8	11. First three operations of Query 1	0.82
8	12. First four queries of Query 1	0.95
8	13. Query 1 + additional Slice	0.98
8	14. Query 1 + additional Dice	0.92
8	15. Query 1 + additional Slice and Dice	0.90

The comparison of queries is metadata-based, i.e., it considers the MD semantics of *SM4MQ* query and QB4OLAP schema structure elements included in the vector structure. Thus, for example, Query 3 differs only in the predicate applied in DICE, disregarding the concrete filtering value. Next, in Query 4 we may notice that just a change of ROLL-UP granularity still keeps the queries highly similar, while in case of Query 5 the change of dimension used in an OLAP operation makes the queries more different. Furthermore, Queries 6, 7, and 8 show that when introducing even more difference the similarity decreases as expected, while keeping the trend that the dimension change has higher impact than the change of granularity. Finally, Queries 9 to 15 follow the scenario of incremental adding of OLAP operations where Queries 9 to 12 use already existing operations of Query 1 while Queries 13 to 15 add one or more additional operations to Query 1. It is interesting to notice that SLICE makes less difference than DICE as it affects smaller part of vector. This can be used for distinguishing between these to operations and, if needed, this can be readjusted by vector redesign.

Overall, our experiments show that *SM4MQ* not only enables the sharing of queries on the SW but its semantics can also be used to automatize the

SM4MQ query transformation into an analysis-ready structure. For instance, our prototype enables that the queries conforming to *SM4MQ* are automatically transformed into vectors as preparation for further processing. The matrix comprised of these vectors can be used to compute query similarities by applying typical similarity functions (e.g., cosine similarity) for this context. The whole process is facilitated such that even non-technical users can perform it.

6 Related Work

The modeling and representing of queries on the SW is just in its infancy. Just recently, in [88] the authors proposed an RDF-based model for representing SPARQL queries. They have used their model to represent a portion of queries over DBpedia and other public SPARQL endpoints with the following suggested use cases: generation of benchmarks, query feature analysis, query cashing, usability analysis, and meta-querying. Thus, there is a movement towards opening not only data but also metadata such as queries so that they could be explored with SPARQL. In this direction, although needed for the context of Exploratory OLAP [4], an MD query model is yet missing. By now, most of the efforts have been devoted to the schema modeling with vocabularies such as QB4OLAP (see [106]). Finally, another interesting use case that can be added to the ones proposed in [88] is the user assistance (e.g., query recommendations). Different MD query recommendation approaches have been proposed by now and a comprehensive overview can be found in [11]. However, little attention has been given to the query modeling and sharing, especially in the SW and Exploratory OLAP contexts.

7 Conclusions

In this paper, we have proposed *SM4MQ*, a Semantic Model for Multidimensional Queries. Using RDF and MD semantics, *SM4MQ* is a step towards Exploratory OLAP and sharing and reuse of MD queries on the SW. Furthermore, its semantics supports the automation of transforming the SM4MQ queries into other analysis-ready representations. In this context, we proposed a method to automate the transformation of queries and exemplified it for the use case of transforming *SM4MQ* to a vector representation. To evaluate our approach, we have developed a prototype implementing the method for the vector use case. In the evaluation, we have used queries over the running example data set. The experiments show that the transformation to a vector representation can be performed automatically supported by the *SM4MQ* semantics and similarities between queries can be identified even with such a simple query representation. In our future work, we plan to

work on the exploitation side of the queries, e.g., develop richer transformations to support advanced user support techniques such as [10].

8 Acknowledgments

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC) and it has been partially supported by the Secretaria d'Universitats i Recerca de la Generalitat de Catalunya under 2014 SGR 1534.

Chapter 6

Conclusions and Future Directions

Abstract

This chapter summarizes the conclusions and directions for future work presented in Chapters 2 - 5, Appendices A and B.

1 Summary of Results

This thesis presented our approach for modeling and exploiting semantic metadata for supporting Exploratory OLAP. The main goal of the thesis is to provide metadata means including models, methods, and tools to support the user in this context. The thesis focuses on next generation BI systems with especial emphasis on the Exploratory OLAP context. We aimed to create a synergy of OLAP as a well-establish data analysis approach in BI, (analytical) metadata management as a typically overlooked task for supporting users, and the SW technologies that provide means to cope with heterogeneity in these novel settings. Considering the research questions related to the metadata lifecycle, this thesis presents several approaches to address some of the existing challenges. Specifically, the thesis addresses the metadata definition and modeling phases, as well as providing techniques for the metadata population and exploitation phases for some of the metadata artifacts defined. In the sequel, we summarize contributions of each of the presented chapters.

Chapter 2 provided the context of metadata that we consider in the thesis. Thus, based on a survey of existing user assistance approaches in the OLAP domain, Chapter 2 defined the *analytical metadata framework* which identified the metadata artifacts used in this context. The artifacts were organized in

the *analytical metadata taxonomy* that provides technical categories of analytical metadata together with the belonging artifacts. The chapter also discussed the types of artifacts processing that are base processing, derivative processing, and goal-oriented processing. The main idea with these different processing types is to support automation of the metadata lifecycle. Accordingly, the metadata artifacts were also classified into explicit (i.e., manually defined), implicit (i.e., detected by the system), and derived (i.e., generated from the existing metadata). The metadata collection/generation should be as transparent as possible for the user and their processing should enable user assistance to facilitate the user analysis. Overall, Chapter 2 motivated the research of this thesis and set the direction that was followed. The key idea promoted is that the analytical metadata must be considered as first-class citizens to support *user-centric* BI systems and Exploratory OLAP.

Chapter 3 presented our approach for modeling of analytical metadata artifacts. Considering the settings of heterogeneous metadata models present in next generation BI systems and in the spirit of Exploratory OLAP, we argued for the use of the SW technologies for analytical metadata formalization. In particular, we chose RDF as it supports sharing and re-usability. RDF also provides the flexibility to interlink and extend different models. Furthermore, instead of claiming for one universal data model which would hardly be accepted, we proposed a metamodel capturing the common semantics of different models. RDF also fits this need as it can be used for the ontological metamodeling (see [14]). Thus, Chapter 3 proposed SM4AM: A Semantic Metamodel for Analytical metadata. The chapter presented extensive examples on the metamodel usage and proposed a method on how to instantiate an RDF-based metamodel as guidelines on how to use the metamodel in a correct manner. Finally, the chapter provided a use case with two real-world data sets from the SW domain on how the metamodel can be used with existing metadata models to narrow the (meta)data search space. Thus, we showed the practical exploitation possibilities of our metamodel.

Chapter 4 presented our approach focusing on the schema metadata artifact and its exploitation on the SW. The central focus of the chapter is on statistical linked open data sets modeled with the QB vocabulary that does not fully support OLAP. Thus, to enable OLAP over these data sets, the chapter proposed a method on how QB data sets can be enriched with additional semantics defined in the QB4OLAP vocabulary that extends QB to introduce the necessary concepts. Rather than just syntactic transformation, this enrichment requires the use of additional metadata concepts and represents a cumbersome and error-prone task. Thus, this chapter especially considered the possibilities for automating the enrichment and discussed the related challenges. In this context, we proposed two techniques to define and/or discover the necessary metadata concepts. These concepts were then used in the method defining the steps to enrich a QB data set with the additional

QB4OLAP semantics. The steps were formalized as SPARQL queries to facilitate the understanding. Chapter 4 then presented a tool developed to evaluate our approach and the experiments with 25 user were conducted in this context. The evaluation showed that the tool facilitates the enrichment. Moreover, the tool also in practice guaranteed the correctness of the produced MD schema. Finally, Chapter 4 included a methodology given in its appendix that was fully formalized using set theory and specified the enrichment steps in terms of pre-conditions, post-conditions, and transformations performed independently of the implementation. Overall, this chapter provided some fundamental means for enabling Exploratory OLAP such as creating an MD schema. The chapter also showed the benefits of using the SW technologies and Linked Data sources that provide additional (meta)data that can be used for the enrichment of existing data sets. This innovation especially relates to a concept of fusion cubes [1] that promotes the use of external data sources to enrich the user analysis. Last but not the least, the chapter demonstrated that all this can be performed in an automatic and user-friendly manner such that even non-technical users can create MD schemata on-the-fly.

Chapter 5 continued our research towards supporting Exploratory OLAP and supporting the user assistance. Once having data conforming to an MD schema on the SW, they are to be explored with MD queries that apply OLAP operations to navigate the data cube. Thus, the chapter proposed SM4MQ: A Semantic Model for Multidimensional Queries that is an RDF-based formalization of MD queries. The model included the most popular OLAP operations, being ROLL-UP, DRILL-DOWN, DICE, and SLICE as proposed in [34]. Following the algebraic formalization of [34] and [29], the semantics of the OLAP operations captured in the model was considered at the conceptual level. Furthermore, SM4MQ represented OLAP operations in a way that provides atomicity and supports their easier sharing and reuse on the SW. The chapter argued for the following benefits of representing MD queries in RDF. First, RDF supports their sharing and reuse. This is especially important considering the Open Data initiative that can also be applied to metadata such as queries. Second, the semantic representation and the conceptual abstraction level enable that SM4MQ queries can be easily transformed into other analysis-ready data structures and further exploited for different purposes. In this context, the chapter proposed a method to automate the transformation of the SM4MQ queries via SPARQL and exemplified the method for a vector representations. Moreover, the chapter presented a prototype implementing the method to transform queries from SM4MQ to a vector representation and compute their similarities using the cosine similarity function. The prototype was used with a set of MD queries to evaluate our approach showing that the transformation can be automatized thanks to the SM4MQ semantics and that this task can be performed even by non-technical users.

Appendix A presented an initial version of the SM4AM metamodel. It introduced the vision of using RDF for metadata representation and provided initial examples and discussion in this direction. It was significantly extended in Chapter 3 and therefore it does not provide new content. Appendix B presented a demonstration of the QB2OLAP tool that includes the QB2OLAPem tool presented in Chapter 4 as a module. It provided additional details on the QB2OLAPem internals and its use together with other complementary modules, giving the overall picture about the QB2OLAPem role and importance.

In summary, we position the contributions of each chapter with respect to the phases of the metadata lifecycle. In particular, Chapter 2 defined the analytical metadata framework including its metadata artifacts that contribute to the metadata definition phase of the metadata lifecycle. Moreover, this chapter also gave insight about the types of metadata processing related to the metadata population and exploitation phases. Next, Chapter 3 proposed modeling techniques to represent analytical metadata in a semantic-aware fashion and address the needs of Exploratory OLAP that contributed to the metadata modeling phase of the metadata lifecycle. Furthermore, Chapter 4 contributed to the several phases of the metadata lifecycle. First, it proposed a method for the definition of an MD schema on the SW that contributed to the metadata definition phase. Next, it provided means to automatically discover the needed metadata that contributed to the metadata population phase. Finally, the discovery of new metadata was driven by the existing schema information which was related to the metadata exploitation phase. In all these aspect, Chapter 4 focused on automation to facilitate these task and enable non-technical users to perform them. Moreover, Chapter 5 likewise brought contributions to several metadata lifecycle phases. One contributions was related to the metadata modeling phase in representing MD queries on the SW. The other contribution was related to the metadata exploitation phase, where it proposed a method to prepare the metadata for the further exploitation processing. In addition, Appendix B also illustrated the exploitation possibilities of the schema metadata artifacts related to the metadata exploitation phase. All in all, the thesis provided different models, algorithms, methods, techniques, and tools as the contributions towards the vision of Exploratory OLAP.

2 Future Research Directions

This PhD thesis opens several interesting and promising research directions. Thus, we next discuss the future research according to the chapters presented.

The approach presented in Chapter 2 is our vision about the role of metadata. By now we have devoted attention to the metadata definition, modeling, and to the processing and population of some metadata artifacts. Thus,

the next step in this direction is to consider other metadata artifacts, such as user preference that can be implicitly detected from user actions. This requires the preference metadata artifact modeling , detection (i.e., population processing), and exploitation for the personalization of user interaction with the system. Moreover, other (or ideally all) metadata artifacts should be used and represented together in a single model to enable even wider exploitation possibilities. In the context of derivative processing, the metadata artifacts can also be combined to generate more complex metadata artifacts such as system or user profiles. This would also require techniques to maintain the complex metadata consistent with respect to the metadata they were derived from. Finally, the exploitation of analytical metadata opens possibilities for systems self-tuning by, e.g., identifying the most used data pieces.

Chapter 3 motivates the use of ontological metamodeling for different exploitation purposes. We believe that using semantic metamodels can serve for novel approaches such as metamodel-driven data search, data analysis recommendations, and even entity resolution and matching techniques. The synergy of good practices from software engineering domain and flexibility of SW technologies can bring benefits for both areas. In this direction, Chapter 4 clearly shows the benefits of both MD modeling and (meta)data sharing and re-use on the SW. This approach opens further possibilities for combining different cubes on the SW and providing recommendations not only on data exploration but also on the schema enrichment. Finally, Chapter 5 outlines the importance of queries in the Exploratory OLAP context and demands further research efforts on the exploitation side, e.g., recommendation algorithms though to benefit from the open and shareable metadata.

Overall, we believe that this PhD thesis opens a new line of research where semantic metadata can bring benefits not only to next generation BI systems and Exploratory OLAP areas, but also to various novel Big Data systems working with data streams and schema-less data structures such as the Data Lake. The tremendous growth of data volumes and their availability require semantics for these data so that they can be automatically and on-the-fly processed. Moreover, this data volume growth also requires that the related metadata providing this semantics are also efficiently processed which requires novel metadata processing and management solutions.

References

[1] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J. Mazón, F. Naumann, T. B. Pedersen, S. Rizzi, J. Trujillo, P. Vassiliadis, and G. Vossen. Fusion cubes: Towards self-service business intelligence. *IJDWM*, 9(2):66–88, 2013.

[2] A. Abelló and O. Romero. On-line analytical processing. In *Encyclopedia of Database Systems*, pages 1949–1954. 2009.

[3] A. Abelló and O. Romero. Ontology driven search of compound IDs. *Knowl. Inf. Syst.*, 32(1):191–216, 2012.

[4] A. Abelló, O. Romero, T. B. Pedersen, R. B. Llavori, V. Nebot, M. J. A. Cabo, and A. Simitsis. Using semantic web technologies for exploratory OLAP: A survey. *IEEE Trans. Knowl. Data Eng.*, 27(2):571–588, 2015.

[5] S. Abiteboul, I. Manolescu, P. Rigaux, M. Rousset, and P. Senellart. *Web Data Management*. Cambridge University Press, 2012.

[6] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[7] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender Systems Handbook*, pages 217–253. 2011.

[8] C. C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016.

[9] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. SQL QueRIE Recommendations. *PVLDB*, 3(2):1597–1600, 2010.

[10] J. Aligon, E. Gallinucci, M. Golfarelli, P. Marcel, and S. Rizzi. A collaborative filtering approach for recommending OLAP sessions. *Decision Support Systems*, 69:20–30, 2015.

[11] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Similarity measures for OLAP sessions. *Knowl. Inf. Syst.*, 39(2):463–489, 2014.

[12] R. Angles. A comparison of current graph database models. In *Workshops Proceedings of the IEEE 28th International Conference on Data Engineering, ICDE 2012, Arlington, VA, USA, April 1-5, 2012*, pages 171–177, 2012.

[13] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite family and relations. *J. Artif. Intell. Res. (JAIR)*, 36:1–69, 2009.

[14] C. Atkinson and T. Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, 20(5):36–41, 2003.

[15] M. Aufaure, N. Kuchmann-Beauger, P. Marcel, S. Rizzi, and Y. Vanrompay. Predicting your next OLAP query based on recent analytical sessions. In *Data Warehousing and Knowledge Discovery - 15th International Conference, DaWaK 2013, Prague, Czech Republic, August 26-29, 2013. Proceedings*, pages 134–145, 2013.

[16] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3), 2009.

[17] D. Beckett and T. Berners-Lee. Turtle - Terse RDF Triple Language, 2011.

[18] L. Bellatreche, A. Giacometti, P. Marcel, H. Mouloudi, and D. Laurent. A personalization framework for OLAP queries. In *DOLAP 2005, ACM 8th International Workshop on Data Warehousing and OLAP, Bremen, Germany, November 4-5, 2005, Proceedings*, pages 9–18, 2005.

[19] R. Berlanga, O. Romero, A. Simitsis, V. Nebot, T. B. Pedersen, A. Abelló, and M. J. Aramburu. Semantic web technologies for business intelligence. *Business Intelligence Applications and the Web: Models, Systems, and Technologies*, 2012.

[20] H. Berthold, P. Rösch, S. Zöller, F. Wortmann, A. Carenini, S. Campbell, P. Bisson, and F. Strohmaier. An architecture for ad-hoc and collaborative business intelligence. In *Proceedings of the 2010 EDBT/ICDT Workshops, Lausanne, Switzerland, March 22-26, 2010*, 2010.

[21] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[22] P. A. Boncz, O. Erling, and M. Pham. Experiences with virtuoso cluster RDF column store. In *Linked Data Management.*, pages 239–259. 2014.

[23] D. Brickley and R. Guha. RDF schema 1.1. `http://www.w3.org/TR/rdf-schema/`, 2014.

[24] M. Castellanos, U. Dayal, and M. Hsu. Live business intelligence for the real-time enterprise. In *From Active Data Management to Event-Based Systems and More - Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthday*, pages 325–336, 2010.

[25] G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, and J. S. V. Varman. The querie system for personalized query recommendations. *IEEE Data Eng. Bull.*, 34(2):55–60, 2011.

[26] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *Scientific and Statistical Database Management, 21st International Conference, SSDBM 2009, New Orleans, LA, USA, June 2-4, 2009, Proceedings*, pages 3–18, 2009.

[27] R. Cyganiak and D. Reynolds. The RDF Data Cube Vocabulary (W3C Recommendation). `http://www.w3.org/TR/vocab-data-cube/`, January 2014.

[28] R. Cyganiak, D. Wood, and M.Lanthaler. Resource description framework (RDF): Concepts and abstract syntax. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`, 2014.

[29] C. D. de Aguiar Ciferri, R. R. Ciferri, L. I. Gómez, M. Schneider, A. A. Vaisman, and E. Zimányi. Cube algebra: A generic user-centric model and query language for OLAP cubes. *IJDWM*, 9(2):39–65, 2013.

[30] M. Erfani, M. Zandi, J. Rilling, and I. Keivanloo. Context-awareness in the software domain—a semantic web enabled modeling approach. *Journal of Systems and Software*, 121:345 – 357, 2016.

[31] M. Essaidi. ODBIS: towards a platform for on-demand business intelligence services. In *Proceedings of the 2010 EDBT/ICDT Workshops, Lausanne, Switzerland, March 22-26, 2010*, 2010.

[32] L. Etcheverry, S. A. Gómez, and A. A. Vaisman. Modeling and querying data cubes on the semantic web. *CoRR*, abs/1512.06080, 2015.

[33] L. Etcheverry and A. A. Vaisman. QB4OLAP: A vocabulary for OLAP cubes on the semantic web. In *Proceedings of the Third International Workshop on Consuming Linked Data, COLD 2012, Boston, MA, USA, November 12, 2012*, 2012.

[34] L. Etcheverry and A. A. Vaisman. Querying semantic web data cubes. In *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016*, 2016.

[35] L. Etcheverry, A. A. Vaisman, and E. Zimányi. Modeling and querying data warehouses on the semantic web using QB4OLAP. In *Data Warehousing and Knowledge Discovery - 16th International Conference, DaWaK 2014, Munich, Germany, September 2-4, 2014. Proceedings*, pages 45–56, 2014.

[36] N. Foshay, A. Mukherjee, and A. Taylor. Does data warehouse end-user metadata add value? *Commun. ACM*, 50(11):70–77, 2007.

[37] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

[38] A. Giacometti, P. Marcel, and E. Negre. A framework for recommending OLAP queries. In *DOLAP 2008, ACM 11th International Workshop on Data Warehousing and OLAP, Napa Valley, California, USA, October 30, 2008, Proceedings*, pages 73–80, 2008.

[39] M. Golfarelli, D. Maio, and S. Rizzi. The dimensional fact model: A conceptual model for data warehouses. *Int. J. Cooperative Inf. Syst.*, 7(2-3):215–247, 1998.

[40] M. Golfarelli and S. Rizzi. Expressing OLAP preferences. In *Scientific and Statistical Database Management, 21st International Conference, SSDBM 2009, New Orleans, LA, USA, June 2-4, 2009, Proceedings*, pages 83–91, 2009.

[41] M. Golfarelli, S. Rizzi, and P. Biondi. myolap: An approach to express and evaluate OLAP preferences. *IEEE Trans. Knowl. Data Eng.*, 23(7):1050–1064, 2011.

[42] L. I. Gómez, S. A. Gómez, and A. A. Vaisman. A generic data model and query language for spatiotemporal OLAP cube analysis. In *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 300–311, 2012.

[43] A. Y. Halevy. Why your data won't mix. *ACM Queue*, 3(8):50–58, 2005.

[44] P. Hanrahan. Analytic database technologies for a new kind of user: the data enthusiast. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 577–578, 2012.

[45] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool Publishers, 2011.

[46] B. Henderson-Sellers. Bridging metamodels and ontologies in software engineering. *Journal of Systems and Software*, 84(2):301–313, 2011.

[47] C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman. Maintaining data cubes under dimension updates. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, 1999*, pages 346–355, 1999.

[48] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi. Towards exploratory OLAP over linked open data - A case study. In *Enabling Real-Time Business Intelligence - International Workshops, BIRTE 2013, Riva del*

Garda, Italy, August 26, 2013, and BIRTE 2014, Hangzhou, China, September 1, 2014, Revised Selected Papers, pages 114–132, 2014.

[49] I. ISO. Iec25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. *International Organization for Standardization*, page 34, 2011.

[50] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, 2014.

[51] M. Jarke, M. A. Jeusfeld, H. W. Nissen, and C. Quix. Heterogeneity in Model Management: A Meta Modeling Approach. In *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, pages 237–253. 2009.

[52] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. *Fundamentals of data warehouses*. Springer, 2013.

[53] C. S. Jensen, T. B. Pedersen, and C. Thomsen. *Multidimensional Databases and Data Warehousing*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[54] M. R. Jensen, T. Holmgren, and T. B. Pedersen. Discovering multidimensional structure in relational data. In *Data Warehousing and Knowledge Discovery, 6th International Conference, DaWaK 2004, Zaragoza, Spain, September 1-3, 2004, Proceedings*, pages 138–148, 2004.

[55] P. Jovanovic, O. Romero, A. Simitsis, and A. Abelló. Integrating ETL processes from information requirements. In *Data Warehousing and Knowledge Discovery - 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings*, pages 65–80, 2012.

[56] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, H. Candón, and S. Nadal. Quarry: Digging up the gems of your data treasury. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 549–552, 2015.

[57] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, and D. Mayorova. A requirement-driven approach to the design and evolution of data warehouses. *Inf. Syst.*, 44:94–119, 2014.

[58] B. Kämpgen and A. Harth. Transforming statistical linked data for use in OLAP systems. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, pages 33–40, 2011.

[59] B. Kämpgen and A. Harth. No size fits all - running the star schema benchmark with SPARQL and RDF aggregate views. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pages 290–304, 2013.

[60] B. Kämpgen, S. O'Riain, and A. Harth. Interacting with statistical linked data via OLAP operations. In *The Semantic Web: ESWC 2012 Satellite Events - ESWC 2012 Satellite Events, Heraklion, Crete, Greece, May 27-31, 2012. Revised Selected Papers*, pages 87–101, 2012.

[61] S. Khouri, I. Boukhari, L. Bellatreche, E. Sardet, S. Jean, and M. Baron. Ontology-based Structured Web Data Warehouses for Sustainable Interoperability: Requirement Modeling, Design Methodology and Tool. *Computers in Industry*, 63(8):799–812, 2012.

[62] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for A collaborative query management system. In *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*, 2009.

[63] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010.

[64] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*. John Wiley & Sons, Inc., 1998.

[65] S. Koide and H. Takeda. MetaModeling in OOP, MOF, RDFS, and OWL. In *2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006) at the 5th International Semantic Web Conference (ISWC 2006)*, 2006.

[66] G. Koutrika and Y. E. Ioannidis. Personalization of Queries in Database Systems. In *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, pages 597–608, 2004.

[67] L. Kowalik. Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.*, 102(5):191–195, 2007.

[68] N. Kozmina and L. Niedrite. OLAP Personalization with User-Describing Profiles. In *Perspectives in Business Informatics Research - 9th International Conference, BIR 2010, Rostock Germany, September 29-October 1, 2010. Proceedings*, pages 188–202, 2010.

[69] H. Lenz and A. Shoshani. Summarizability in OLAP and statistical data bases. In *Ninth International Conference on Scientific and Statistical*

Database Management, Proceedings, August 11-13, 1997, Olympia, Washington, USA, pages 132–143, 1997.

[70] A. Leonard, M. Masson, T. Mitchell, J. Moss, and M. Ufford. Business Intelligence Markup Language. In *SQL Server 2012 Integration Services Design Patterns*, pages 301–326. 2012.

[71] A. Löser, F. Hueske, and V. Markl. *Situational Business Intelligence*, pages 1–11. 2009.

[72] A. Maté and J. Trujillo. A Trace Metamodel Proposal Based on the Model Driven Architecture Framework for the Traceability of User Requirements in Data Warehouses. *Inf. Syst.*, 37(8):753–766, 2012.

[73] J. Mazón, J. Lechtenbörger, and J. Trujillo. A survey on summarizability issues in multidimensional modeling. *Data Knowl. Eng.*, 68(12):1452–1469, 2009.

[74] J. Mazón, J. J. Zubcoff, I. Garrigós, R. Espinosa, and R. Rodríguez. Open business intelligence: on the importance of data quality awareness in user-friendly data mining. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012*, pages 144–147, 2012.

[75] M. Middelfart and T. B. Pedersen. The meta-morphing model used in TARGIT BI suite. In *Advances in Conceptual Modeling. Recent Developments and New Directions - ER 2011 Workshops FP-UML, MoRE-BI, Onto-CoM, SeCoGIS, Variability@ER, WISM, Brussels, Belgium, October 31 - November 3, 2011. Proceedings*, pages 364–370, 2011.

[76] K. Morton, M. Balazinska, D. Grossman, and J. D. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *PVLDB*, 7(6):453–456, 2014.

[77] F. Naumann. Data profiling revisited. *SIGMOD Record*, 42(4):40–49, 2013.

[78] V. Nebot and R. B. Llavori. Building data warehouses with semantic web data. *Decision Support Systems*, 52(4):853–868, 2012.

[79] Object Management Group. Common Warehouse Metamodel Specification 1.1, last accessed September, 2016. http://www.omg.org/spec/CWM/1.1/PDF/.

[80] T. B. Pedersen. Multidimensional modeling. In *Encyclopedia of Database Systems*, pages 1777–1784. 2009.

[81] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A foundation for capturing and querying complex multidimensional data. *Inf. Syst.*, 26(5):383–423, 2001.

[82] E. Prud'hommeaux and A. Seaborne. SPARQL 1.1 Query Language for RDF, 2011.

[83] S. Rizzi. Business intelligence. In *Encyclopedia of Database Systems*, pages 287–288. 2009.

[84] O. Romero and A. Abelló. On the need of a reference algebra for OLAP. In *Data Warehousing and Knowledge Discovery, 9th International Conference, DaWaK 2007, Regensburg, Germany, September 3-7, 2007, Proceedings*, pages 99–110, 2007.

[85] O. Romero and A. Abelló. A survey of multidimensional modeling methodologies. *IJDWM*, 5(2):1–23, 2009.

[86] O. Romero and A. Abelló. A framework for multidimensional design of data warehouses from ontologies. *Data Knowl. Eng.*, 69(11):1138–1157, 2010.

[87] O. Romero, D. Calvanese, A. Abelló, and M. Rodriguez-Muro. Discovering functional dependencies for multidimensional design. In *DOLAP 2009, ACM 12th International Workshop on Data Warehousing and OLAP, Hong Kong, China, November 6, 2009, Proceedings*, pages 1–8, 2009.

[88] M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A. N. Ngomo. LSQ: the linked SPARQL queries dataset. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, pages 261–269, 2015.

[89] M. Saleem and A. N. Ngomo. Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, pages 176–191, 2014.

[90] SDMX. Content Oriented Guidelines. `http://sdmx.org/?page_id=11`, 2009.

[91] SDMX. SDMX standards: Information model. `http://sdmx.org/wp-content/uploads/2011/08/SDMX_2-1-1_SECTION_2_InformationModel_201108.pdf`, 2011.

[92] T. Sellam and M. L. Kersten. Meet charles, big data query advisor. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[93] D. Skoutas and A. Simitsis. Ontology-based conceptual design of ETL processes for both structured and semi-structured data. *Int. J. Semantic Web Inf. Syst.*, 3(4):1–24, 2007.

[94] K. Stefanidis, M. Drosou, and E. Pitoura. You may also like" results in relational databases. In *Proceedings International Workshop on Personalized Access, Profile Management and Context Awareness: Databases, Lyon, France*, 2009.

[95] K. Strange. Etl was the key to this data warehouse's success. Gartner Research, CS-15-3143, 3 2002.

[96] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3):157–168, 2011.

[97] Tim Berners-Lee. Principles of Design, last accessed July, 2016. http://www.w3.org/DesignIssues/Principles.html.

[98] R. Touma, O. Romero, and P. Jovanovic. Supporting data integration tasks with semi-automatic ontology construction. In *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP, DOLAP 2015, Melbourne, VIC, Australia, October 19-23, 2015*, pages 89–98, 2015.

[99] M. F. Uddin, N. Gupta, et al. Seven v's of big data understanding big data to extract value. In *American Society for Engineering Education (ASEE Zone 1), 2014 Zone 1 Conference of the*, pages 1–5, 2014.

[100] A. A. Vaisman. Publishing OLAP cubes on the semantic web. In *Business Intelligence - 5th European Summer School, eBISS 2015, Barcelona, Spain, July 5-10, 2015, Tutorial Lectures*, pages 32–61, 2015.

[101] A. A. Vaisman and E. Zimányi. *Data Warehouse Systems - Design and Implementation*. Springer, 2014.

[102] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, and C. Thomsen. QB2OLAP: enabling OLAP on statistical linked open data. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 1346–1349, 2016.

[103] J. Varga, O. Romero, T. B. Pedersen, and C. Thomsen. SM4AM: A semantic metamodel for analytical metadata. In *Proceedings of the 17th International Workshop on Data Warehousing and OLAP, DOLAP 2014, Shanghai, China, November 3-7, 2014*, pages 57–66, 2014.

[104] J. Varga, O. Romero, T. B. Pedersen, and C. Thomsen. Towards next generation BI systems: The analytical metadata challenge. In *Data Warehousing and Knowledge Discovery - 16th International Conference, DaWaK 2014, Munich, Germany, September 2-4, 2014. Proceedings*, pages 89–101, 2014.

[105] J. Varga, O. Romero, T. B. Pedersen, and C. Thomsen. Analytical metadata modeling for next generation bi systems. *In submission*, 2016.

[106] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen. Dimensional enrichment of statistical linked open data. *J. Web Sem.*, 40:22–51, 2016.

[107] P. Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *10th International Conference on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998*, pages 53–62, 1998.

[108] X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 964–975, 2009.

[109] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao. Semantic SPARQL similarity search over RDF knowledge graphs. *PVLDB*, 9(11):840–851, 2016.

[110] P. Zikopoulos, C. Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

Paper A

SM4AM: A Semantic Metamodel for Analytical Metadata

The paper has been published in the
Proceedings of the 17th International Workshop on Data Warehousing and OLAP, pp. 57-66 (2014). The layout of the paper has been revised.
DOI: https://doi.org/10.1145/2666158.2666182

Abstract

Next generation BI systems emerge as platforms where traditional BI tools meet semi-structured and unstructured data coming from the Web. In these settings, the user-centric orientation represents a key characteristic for the acceptance and wide usage by numerous and diverse end users in their data analysis tasks. System and user related metadata are the base for enabling user assistance features. However, current approaches typically store these metadata in ad-hoc manners. In this paper, we propose a generic and extensible approach for the definition and modeling of the relevant metadata artifacts. We present SM4AM, a Semantic Metamodel for Analytical Metadata created as an RDF formalization of the Analytical Metadata artifacts needed for

user assistance exploitation purposes in next generation BI systems. We consider the Linked Data initiative and its relevance for user assistance functionalities. We discuss the metamodel benefits and present directions for future work.

1 Introduction

The analysis of ever-increasing data volumes represents an essential task for obtaining knowledge to support decision-making. Next generation BI systems (a.k.a., BI 2.0 systems) are characterized by heterogeneous, semi-structured and/or unstructured, and external data sources. Moreover, they claim for a strong *user-centric* orientation by supporting the user analytical processes in these settings. End users must be enabled to perform their analysis in the easiest and most efficient way possible. This entails different user assistance features provided by the system.

Major user support categories that we encountered in the literature (see [104]) are `querying` assistance, most typically represented by query recommendation, and `visualization` assistance for representing data in the most suitable way. To facilitate the user exploration of data, the system may suggest a query, help the user to compose a query, personalize the user results according to the user profile, visualize the data by user preferences, and similar. Most of these activities are based on the exploitation of metadata, such as queries, preferences, schema information, etc. Based on a survey conducted in [104], we defined the Analytical Metadata (AM) framework and discovered the metadata artifacts that are needed for user assistance functionalities. We especially elaborated on the need for automation of both metadata collection (e.g., by inference) and exploitation tasks by different processing techniques. AM are the means for BI 2.0 systems to understand and assist the user. For instance, user profiles enable the comparison of users and recommendations based on the similarities found. However, by now there is no unified formalization of AM and current approaches typically store pieces of metadata artifacts in ad-hoc manners that cannot be reused in other systems. They mostly focus on recommendation algorithms while metadata management and organization typically have been overlooked.

For the maximal benefit of AM, they need to be systematically organized and represented in a machine-readable format so that the system becomes *semantic-aware* and able to automatize the metadata processing. This is especially important considering the context of BI 2.0 systems where high frequency of changes, heterogeneity, and diversity of data sources need to be automatically overcome. The Resource Description Framework (RDF) has been widely applied as a semantic-aware formalism in the Semantic Web environments and can also bring benefits for the data analysis in BI 2.0 systems (see [4]). RDF represents data as simple subject-predicate-object triples that

are machine-processable and flexible for representation of different types of data. Most importantly, it brings a good ratio between expressiveness and computational complexity.

RDF is widely applied in the Linked Data initiative that is a significant participant of BI 2.0 settings. Linked Data addresses the challenge of linking independent and heterogeneous data sources typically available on the Web. It has been accepted by a significant number of participants including the industry, e.g., Swirrl[1], and public government institutions, e.g., the European Union in the case of Linked Open Data[2]. Linked Data represents a valuable wealth of information as many types of data including geographical, media, government, education, retail and commerce, user generated content and social media, are a part of the Web of Data represented in RDF (see [45]).

As external and heterogeneous data sources, Linked Data can highly benefit from user assistance functionalities for their exploration. Unlike internal sources where data organization is generally familiar, Linked Data sources are typically non-controlled and new for the user that needs guidance and recommendations for their analysis. Moreover, existing BI systems need to enrich their internal data with the information coming from the external data sources. Hence, the user again needs assistance on how to manage these external resources. In these scenarios, not only data but also metadata should be interlinked in order to be used for the user assistance. In this way, the user assistance can be achieved over different independent systems that are mutually linked. For instance, the user profile and characteristics can be represented in a form understandable for different systems and used for the personalization of user interaction with all of them. Notice that RDF is already a mean for capturing different types of metadata (e.g., about music, images, videos, etc.)[3].

Contributions. In this paper, we propose a generic and extensible approach for the definition and modeling of the metadata artifacts needed for user assistance purposes in BI 2.0 systems. We present SM4AM, a Semantic Metamodel for Analytical Metadata created as an RDF formalization of AM artifacts relevant for this context. The RDF representation provides semantics that enables machine-processing of these metadata. We consider the Linked Data initiative and its relevance for user assistance functionalities in BI 2.0 systems. The metamodel is elaborated through a set of examples that resemble a real use case scenarios. Finally, we provide a thorough discussion of the benefits and potential applications of the metamodel. Our focus is on how to store AM in a semantic-aware and reusable manner since that directly determines the future exploitation possibilities. Note that metadata processing and exploitation are out of the scope of this paper, and we address them only

[1] http://www.swirrl.com/

[2] http://ec.europa.eu/digital-agenda/en/open-data-0

[3] http://dublincore.org/

as examples to illustrate the potential use and benefits of SM4AM.

The rest of the paper is organized as follows. Section 2 briefly introduces RDF as a semantic-aware formalism and presents the running example coming from Linked Data. Next, the metamodel with prerequisites is presented in Section 3. Section 4 provides a discussion about the benefits of the approach. Finally, Section 5 presents the related work, while Section 6 concludes the paper and gives future work directions on how we plan to exploit SM4AM.

2 Towards Semantic-awareness

When focusing on BI 2.0 systems aiming at semantic-awareness and inclusion of external sources, RDF and Linked Data arise as the best fit option. Indeed, RDF[4] is very flexible for capturing data semantics and most information can be naturally represented through RDF triples. A triple consists of a subject, a predicate, and an object, and represents a binary relationship (a predicate) between two resources (a subject and an object) or a resource (a subject) and a literal (an object). An overview of Linked Data and RDF can be found in [21]. In short, Linked Data represents an initiative for the Web environment to provide a mechanism for semantic interlinking of data. Through semantic predicates, computers can interpret the meanings of a subject, an object, and the relation between them. This opens wide possibilities for exploitation and processing of machine-processable data with their description and meaningful relations. For example, the same concepts can be reused in several independent systems and as discussed in [21], data is self-describing and data access is simplified by RDF as standardized data model. Although quite simple, RDF and its extension RDFS[5] (RDF Schema) represent formalisms that can be used for relevant source discovery, data integration, mappings of business and technical terms, incorporation of external and heterogeneous sources, and other.

It is especially important to note that RDF and RDFS (jointly referred as RDF(S)) semantics enable reasoning possibilities. Table A.1, extracted from [5], represents RDF(S) statements and their First Order Logic translations (FOL).

By using this semantics, we can infer the knowledge that has not been explicitly stated. For instance, Figure A.1 illustrates a part of the schema information, certain (explicit) instances related to this schema, and the knowledge inferred from the previous two. Note that in the figure, we use Turtle[6] RDF notation, *rdf* and *rdfs* namespaces for RDF and RDFS respectively, and an unnamed namespace (represented with only ':') for custom elements.

[4] http://www.w3.org/TR/rdf11-concepts/
[5] http://www.w3.org/TR/rdf-schema/
[6] http://www.w3.org/TR/turtle/

Table A.1: RDF(S) statements and FOL [5]

RDF(S) statements	FOL translation
i rdf:type C	$C(i)$
i P j	$P(i,j)$
C rdfs:subClassOf D	$\forall X(C(X) \Rightarrow D(X))$
P rdfs:subPropertyOf R	$\forall X \forall Y(P(X,Y) \Rightarrow R(X,Y))$
P rdfs:domain C	$\forall X \forall Y(P(X,Y) \Rightarrow C(X))$
P rdfs:range D	$\forall X \forall Y(P(X,Y) \Rightarrow D(Y))$

```
# ...
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
# ...
# A part of the schema information
:Android rdfs:subClassOf :OperatingSystem .
:OptimizedFor rdfs:subPropertyOf :RunsOn .
:RunsOn rdfs:domain :OperatingSystem .
:RunsOn rdfs:range :Device .
# ...
# Instances (explicit facts asserted)
:KitKat rdf:type :Android .
:MobilePhone rdf:type :Device .
:KitKat :OptimizedFor :MobilePhone .
# ...
# Inferred knowledge!
:KitKat rdf:type :OperatingSystem .
:KitKat rdf:RunsOn :MobilePhone .
#...
```

Fig. A.1: Inferred Knowledge Example

As a running example, we will use the schema represented in Figure A.2. This schema is a piece of the Freebase[7] data set and will be used to show the relevance of the proposed metamodel for real data sets. By using this data set, we extend the analysis scope of our metamodel to include and consider a Linked Data source. In the figure, the prefixes *f*, *t*, *l*, and *m* relate to the *http://www.freebase.com/film/film/*, *http://www.freebase.com/type/*, *http://www.freebase.com/location/*, and *http://www.freebase.com/measurement_unit/* namespaces respectively.

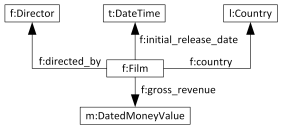

Fig. A.2: Running Example

The central concept is *f:Film*, which represents data about films. The *f:initial_release_date* property correlates *f:Film* to the *t:DateTime* class instance defining the film's initial release date. Next, the *f:directed_by* property relates *f:Film* with the film's *f:Director*. The *f:country* property correlates *f:Film* with

[7] http://www.freebase.com/

the *l:Country* of origin. Finally, *f:gross_revenue* links a film with the money amount (*m:DatedMoneyValue*) of the revenue. This schema information can be represented in Turtle RDF notation as illustrated in Figure A.3.

```
# Running Example
@prefix f: <http://www.freebase.com/film/film/> .
@prefix t: <http://www.freebase.com/type/> .
@prefix l: <http://www.freebase.com/location/> .
@prefix m: <http://www.freebase.com/measurement_unit/> .
# ...
f:Film f:initial_release_date t:DateTime .
f:Film f:directed_by f:Director .
f:Film f:country l:Country .
f:Film f:gross_revenue m:DatedMoneyValue .
# ...
```

Fig. A.3: Running Example in Turtle Notation

3 SM4AM

BI 2.0 settings need metadata to enable advanced user assistance functionalities. As different systems model and use ad-hoc metadata solutions, an approach for correlating similar concepts is strongly needed. As explained in [51] where typical (metamodel, model, and instance) modeling abstraction levels were discussed, the metamodel level is convenient for these kind of settings where heterogeneous models can be created as instances of the metamodel. Indeed, as we showed in [104], existing approaches use similar concepts for the user assistance. Therefore, in this section we define SM4AM, a Semantic Metamodel for Analytical Metadata that is to capture relevant metadata artifacts needed in the BI 2.0 context. Then, various systems can use the metamodel to instantiate specific models for their needs. To achieve semantic-awareness, our metamodel is represented in RDF since we follow the principle of least power (see [97]). This means that it provides an acceptable expressivity regardless of its simplicity. The notation for RDF is flexible and even though there is no explicit distinction between meta classes and classes (metamodel and model levels), it can be represented by the *rdf:type* property (see Table A.1). In RDF there is no restriction that an instance cannot be a class at the same time. For example, in Figure A.4 we could express that *Film* is a class and an instance at the same time. In the figure, we again use an unnamed namespace for custom concepts. Additionally, in the context of BI 2.0 systems, an especially interesting characteristic of RDF models is their extensibility. Novel concepts can be easily incorporated and the metamodel can evolve according to the needs. This mechanism is already used in Semantic Web environments and BI 2.0 systems can strongly benefit from it. Figure A.5 presents the metamodel. It illustrates the metamodel classes and their relationships that formalize the AM artifacts needed for the user assistance activities. The metamodel includes the QB4OLAP

part coming from [35], which is used for the representation of the schema as AM artifact. As explained in [35], shadings of the QB4OLAP elements visually distinguishes their RDF vocabularies. In the next subsection, we briefly outline AM and discuss QB4OLAP as the prerequisites for the metamodel understanding. The rest of the section explains in detail all other parts of the metamodel.

```
1 # ...
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 # ...
4 :Film rdf:type :Art .
5 :Gladiator rdf:type :Film .
6 # ...
```

Fig. A.4: Class and Instance Concept Example

3.1 Prerequisites

Analytical Metadata

The AM defined in [104] represent a set of metadata artifacts supporting user assistance activities in the context of BI 2.0 systems. Figure A.6 illustrates the AM organization. Metadata artifacts are grouped into definitional, data quality, navigational, lineage, and ratings categories. The definitional category includes the *vocabulary* artifact defining business terms, their relationships, and their mappings to the integration schema; *schema* representing the data model; and *user characteristics* capturing the explicitly stated information about the users (e.g., name, job, address, etc.). The data quality category contains *profiling metadata* capturing technical characteristics of the data set. Further, the navigational category includes *query* as a user inquiry for certain data (disregarding the form it takes), *query log* as a list of all queries ever posed, and *user session* as a sequence of queries posed by the user when analyzing or searching for certain data. The lineage category includes the *traceability metadata* artifact describing information about data sources, performed transformations, and mappings to the integration schema. Finally, the ratings category keeps *user preferences* as the result set selection and/or representation prioritization (e.g., 2014 as the preferred release year of a film or bar chart as the preferred result visualization) and *statistics* as data usage indicators (e.g., most queried films). According to the survey conducted in [104], AM are the most relevant artifacts needed for the user assistance exploitation. In the present paper, we go one step further and formalize them into our SM4AM metamodel.

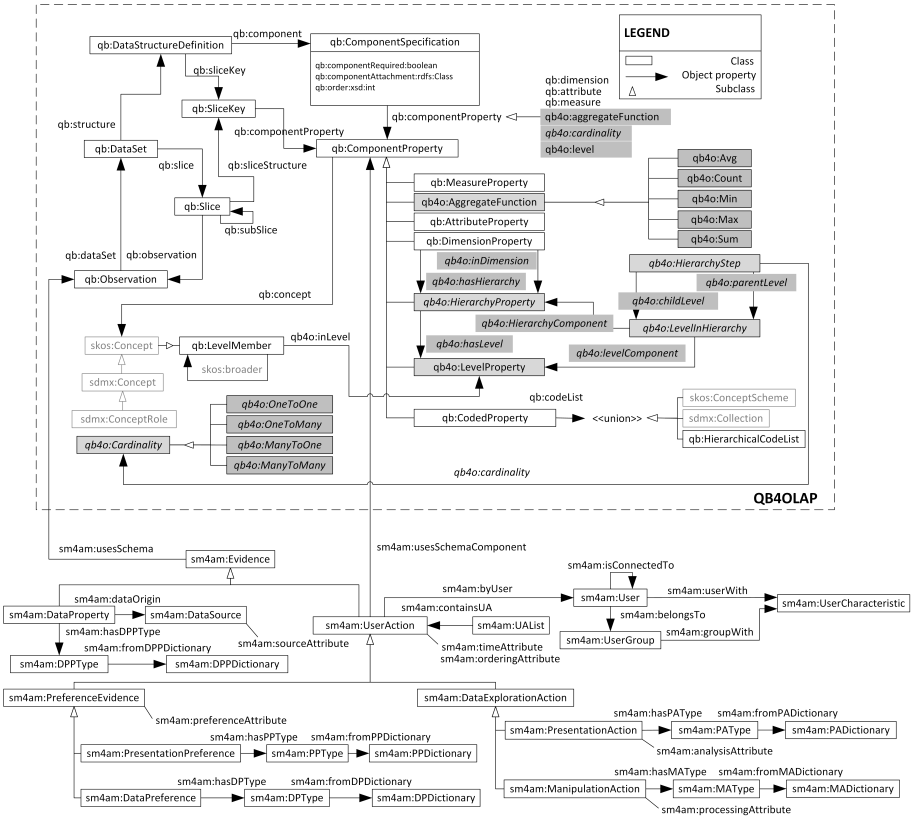

Fig. A.5: SM4AM: A Semantic Metamodel for Analytical Metadata (QB4OLAP [35])

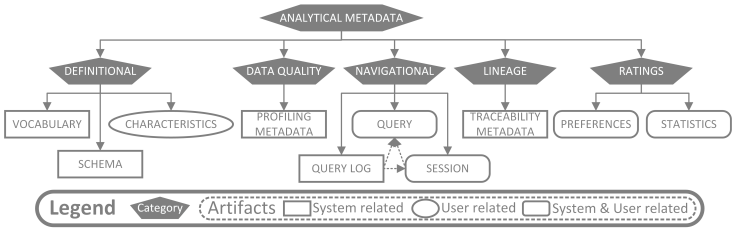

Fig. A.6: Analytical Metadata

Schema and Multidimensionality

The previously introduced AM include the *schema* artifact that should capture the integration schema. Data in BI systems are typically modeled in a multidimensional (MD) fashion and this should be reflected in the schema. As discussed in [80], the MD model [64] is mature and well-founded and has key applicability in data warehousing, OLAP, and increasingly in data mining. It captures analytical perspectives by means of facts and dimensions. In this context, it defines necessary constraints and is covered by the MD algebra [84] that determine potential user actions. For example, if a user analyzes data on the `Month` level, just based on the *schema* she can be suggested to change the granularity to the `Day` or `Year` level even if no one performed this analysis before. In this context, the authors of [35] have recently proposed QB4OLAP as an RDF-based multidimensional model schema (see the upper part of Figure A.5). For our metamodel, the concepts represented in QB4OLAP are considered as metamodel classes since their instances are classes belonging to the data model of a concrete data set which, in turn, will have its instances. We use [35] for representing the *schema* artifact that is then correlated to the rest of our metamodel. For instance, the running example can be considered as an implicit multidimensional schema where *f:Film* is the analyzed fact with the gross revenue (*m:DatedMoneyValue* class) as a measure, and *t:DateTime*, *f:Director*, and *l:Country* are the dimensions of the analysis. Different hierarchies with corresponding levels can be built on top of these dimensional subjects depending on the data set used. These concepts should be linked to the QB4OLAP [35] vocabulary, as for example in Figure A.7. Note that the *qb* namespace refers to the Data Cube[8] vocabulary. More details on how to use QB4OLAP for RDF data sets can be found in [35]. Moreover, insights about identification of MD schemas from RDF data can be found in [61] and [78].

```
# ...
f:Film     rdf:type qb:Observation .
m:DatedMoneyValue rdf:type qb:MeasureProperty .
t:DateTime     rdf:type qb:DimensionProperty .
f:Director     rdf:type qb:DimensionProperty .
l:Country     rdf:type qb:DimensionProperty .
# ...
```

Fig. A.7: Addition of MD Semantics Example

3.2 The Metamodel Elements

The metamodel captures AM artifacts either directly, i.e., by a one-to-one mapping of an artifact to the metamodel element, or indirectly where an artifact information is to be retrieved from more than one metamodel element.

[8] http://www.w3.org/TR/vocab-data-cube/

Therefore, for better understanding, we explain the metamodel in groups of elements according to the information they seize. Each group is represented with a figure (see Figures A.9 to A.13) depicting the corresponding part of the metamodel along with examples of potential instantiations of model and instance levels intended to illustrate how the metamodel can be used. In these figures, we visually separate the metamodel, model, and instance levels. The metamodel elements belong to our *sm4am* namespace, while we use the *@prefix ex: <http://www.example.org/>* namespace for examples of the model and instance elements. While the metamodel is common for different systems, the model and instance levels can vary depending on the particular system. Note that in the figures we abuse the notation (by using the same shapes for metamodel, model, and instance levels) in order to enhance the understanding. Nevertheless, these different abstraction levels are clearly separated to the corresponding boxes. If needed, they can also be represented textually in RDF like in Figures A.1, A.3, A.4, and A.7. Finally, to give the intuition on how the metadata of any of the three abstraction levels can be exploited for the user assistance in a real case scenario, we provide examples related to our running example introduced in Section 2.

Evidence Elements

As mentioned, the AM artifacts in our metamodel are modeled directly or indirectly. For example, a query typically consists of several operators that we keep individually. Therefore, our metamodel captures the AM artifacts through the pieces of evidence that capture information about the user and user's actions, or the system and its properties. When the pieces of evidence are on a finer granularity than the artifact itself, as in the previous example of the query, we consider the artifact indirectly modeled. Figure A.8 illustrates this flow in which AM evidence is being collected from BI 2.0 and Linked Data environments, after which it is formalized in our SM4AM metamodel so that it can be processed and enable user assistance exploitations, e.g., querying or visualization. Pieces of evidence that are collected refer to user queries, preferences, user characteristics, schema, and other AM artifacts. Therefore, the central focus of the metamodel is the *sm4am:Evidence* class (see Figure A.5). According to the evidence information type, it is subcategorized to *sm4am:DataProperty* and *sm4am:UserAction*. The first subcategory represents evidence about the data set such as data origin, resource usage, etc. The latter captures both explicit evidence of the user actions (e.g., queries) and implicit evidence automatically inferred from the user actions (e.g., detected visualization preference). *sm4am:UserAction* is characterized by two attributes, `sm4am:timeAttribute` defining the time parameter(s) related to the user actions (e.g., action duration), and `sm4am:orderingAttribute` defining different ordering characteristics for the sequences captured by *sm4am:UAList* that

is defined in Section 3.2. Each evidence will be stored according to its type as discussed in the next subsections.

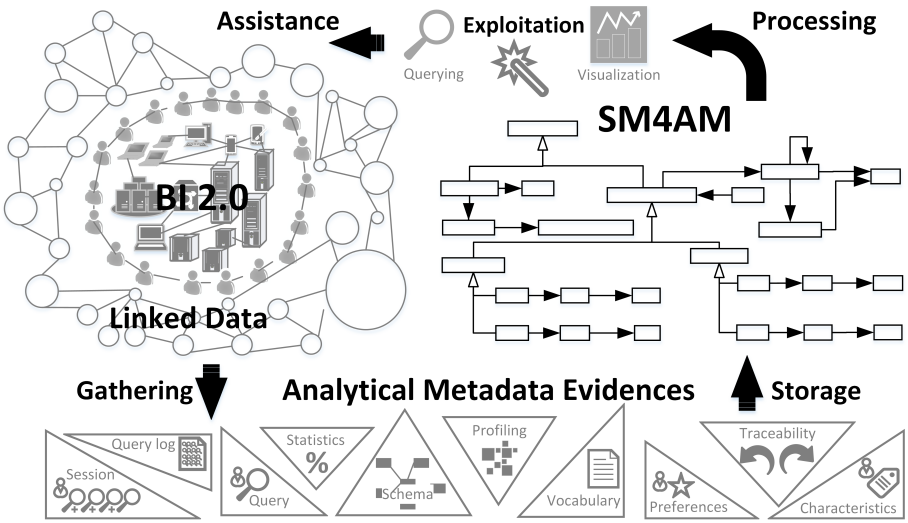

Fig. A.8: Evidence Usage Flow

Data Exploration Action Elements

The first evidence belonging to the *sm4am:UserAction* category is *sm4am:DataExplorationAction* or shortly DEA. It represents an atomic user operation over data. For instance, in the multidimensional context discussed in Section 3.1, an atomic operation can be a multidimensional operation. This and similar cases are modeled with *sm4am:ManipulationAction* that captures the actions for data handling (e.g., change of data granularity) with `sm4am:processingAttribute` representing the characteristics of the actions. Another DEA is *sm4am:PresentationAction* describing the actions for data presentation (e.g., diagram type selection) with `sm4am:analysisAttribute` that captures characteristics specific for visual data exploration. Both actions are related to their corresponding types, *sm4am:MAType* (i.e., manipulation action type) and *sm4am:PAType* (i.e., presentation action type) respectively, and this way the same type actions can be easily grouped. Furthermore, the types belong to the dictionaries *sm4am:MADictionary* and *sm4am:PADictionary*, respectively, that define the sets of potential actions depending on the concrete system, i.e., model instantiation. The actions, action types, and dictionaries are correlated with the corresponding properties (*sm4am:hasPAType*, *sm4am:hasMAType*, *sm4am:fromPADictionary*, *sm4am:fromMADictionary*). Figure A.9 illustrates the corresponding piece of metamodel, and its potential model and instance

level examples that we explain next.

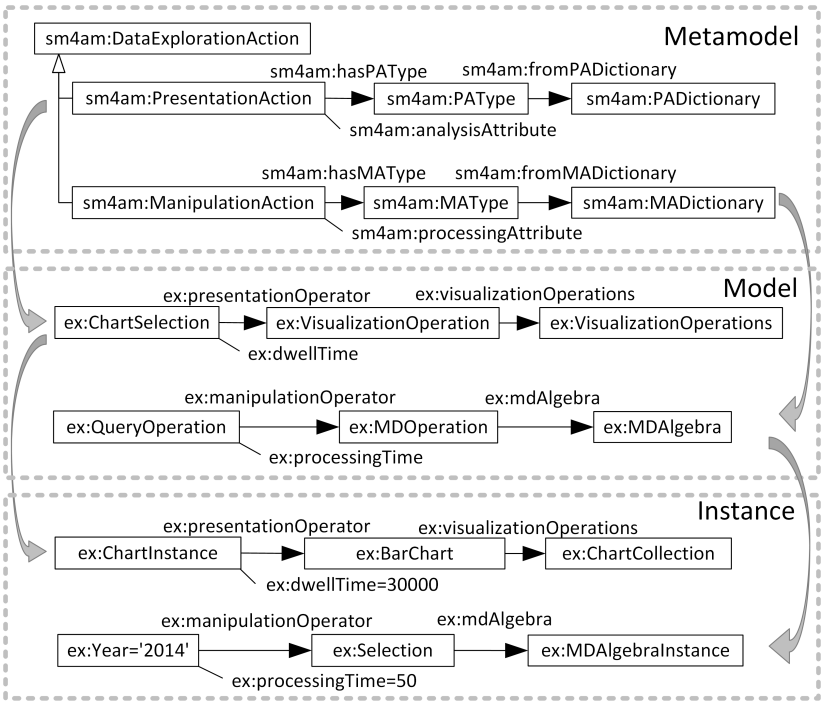

Fig. A.9: Data Exploration Action Elements

Example. The model level box represents the potential use of a multidimensional algebra (*ex:MDAlgebra*) and a visualization operations collection (*ex:VisualizationOperations*) as instances of *sm4am:MADictionary* and *sm4am:PADictionary* respectively. Furthermore, *ex:MDOperation* and *ex:VisualizationOperation* are instances of *sm4am:MAType* and *sm4am:PAType* respectively, while *ex:QueryOperation* and *ex:ChartSelection* exemplify instances of the *sm4am:ManipulationAction* and *sm4am:PresentationAction* metamodel classes respectively. Notice that *sm4am:processingAttribute* is instantiated as the *ex:processingTime* attribute that models the time the system needed to process the user action. In a similar way, *ex:dwellTime* models the time that the user spent in analyzing the returned results. The instance level of Figure A.9 illustrates how concrete chart instance (*ex:ChartInstance*) of the bar chart type (*ex:BarChart*) from the certain chart collection (*ex:ChartCollection*) would be stored. The same goes for the selection of year 2014 (*ex:Year='2014'*) as selection operation (*ex:Selection*) from the concrete multidimensional algebra (*ex:MDAlgebraInstance*).

Regarding the potential user assistance exploitation in the context of our running example, these metadata capture two situations. First, when the user

is querying the Film data set and looks for the films released in year 2014. Second, when the user selects a certain bar chart to analyze. These metadata can be used for various user assistance, e.g., recommendations of the similar searches related to the same or some other data set linked to the current one, suggestions of the alternative data visualization possibilities, etc. Once the metadata are captured and stored, different user assistance and other exploitation techniques can be applied.

Preference Evidence Elements

Another specialized evidence of *sm4am:UserAction* is *sm4am:PreferenceEvidence* capturing the user preferences regarding the data and their presentation. It is characterized by `sm4am:preferenceAttribute` that represents preference characteristics important for system processing (i.e., priority). The purpose of the preferences is to store information that enables the personalization of user interaction with the system and we divide them into the next two categories. *sm4am:PresentationPreference* captures evidence regarding the data presentation, typically visualization affinities, while *sm4am:DataPreference* keeps information about the data interests and their importance that can be exploited for the result personalization and similar purposes. As in the previous case, both categories have their corresponding type classes, *sm4am:PPType* (i.e., presentation preference type) and *sm4am:DPType* (i.e., data preference type), that are further related to the *sm4am:PPDictionary* and *sm4am:DPDictionary* dictionaries determining the potential types. All these concepts are correlated by the appropriate properties (*sm4am:hasPPType*, *sm4am:hasDPType*, *sm4am:fromPPDictionary*, *sm4am:fromDPDictionary*) as illustrated in Figure A.10. This figure depicts related part of the metamodel with potential model and instance levels for this context.

Example. The model level defines a data ordering preference expression (i.e., *ex:DOPreferenceExpression*) that corresponds to a data ordering preference type (i.e., *ex:DataOrderingPreference*) coming from a data preference algebra (i.e., *ex:DPAlgebra*). Further, an instance metadata level would possibly store *ex:PreferredYear='2014'* as a concrete preference expression, corresponding to a *ex:DimensionPreference* data preference operation of *ex:DPAlgebraInstance*. The similar examples for the instantiation of *sm4am:PresentationPreference* and related classes can be found in the figure with corresponding attributes and properties. Note that the model example contains two instantiations of *sm4am:preferenceAttribute*, the `ex:inferred` attribute defining whether the evidence is explicitly stated by the user or it is inferred by the system, and the `ex:priority` attribute defining the priority of the preference needed for the ranking between preferences, i.e., which preference has higher priority.

In the context of our running example, these metadata can represent the situations where the user explicitly states her preference towards the year

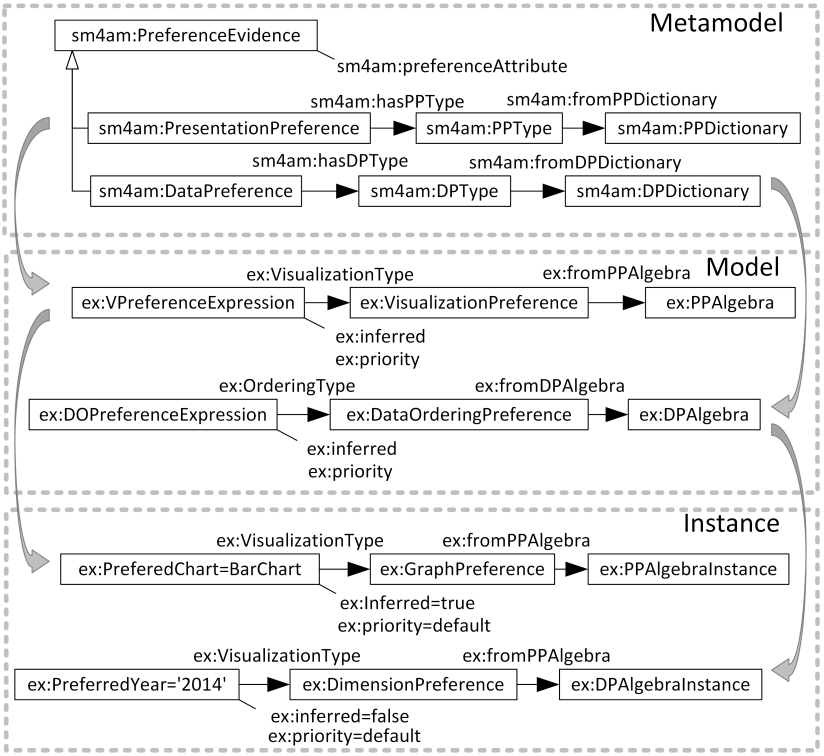

Fig. A.10: Preference Evidence Elements

2014 of the film's initial release date and where the system infers the user preference for bar charts from her previous analysis (e.g., a bar chart presentation of an average film revenue per year). These metadata can be used for the user assistance such as personalization of the results so that the ones related to the year 2014 always have higher priority and are shown before the others. For instance, this information can be used in the approach like [41]. In the context of Linked Data, this preference can be applied to any other data set linked to these concepts, e.g., music. Figure A.10 similarly includes *sm4am:PresentationPreference* related examples that can for instance be used to visualize data as bar charts whenever possible (as in [75]).

User Action List Element

After introducing simple *sm4am:UserAction*s, we now define the *sm4am:UAList* (i.e., user action list) class for composing them into ordered lists that represent different concepts. For instance, a query can be represented as an ordered list of: i) one or more *sm4am:ManipulationAction*s, ii) optionally one

or more *sm4am:PreferenceEvidence*, and iii) one or more *sm4am:PresentationAction*s. The *sm4am:timeAttribute* and *sm4am:orderingAttribute* attributes of the *sm4am:UserAction* evidence can be instantiated to keep the action time and the ordering in a list of user actions.

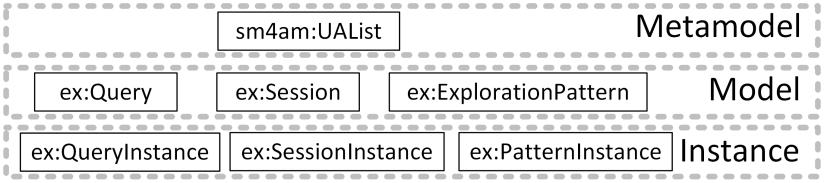

Fig. A.11: User Action List Element

Example. Figure A.11 illustrates the metamodel part and examples of model and instance levels for this context. The model level defines the concrete instances of the metamodel and as potential model level concepts we have chosen a query (*ex:Query*), a session (*ex:Session*), and an exploration pattern (*ex:ExplorationPattern*). These concepts are to be instantiated with concrete instances that are encountered in the system usage.

In the context of our running example, let us consider that the user searches for gross revenue of films from year 2014. Moreover, the user expresses the preference that the revenues over 10 million dollars are more preferred than the others. In this case, corresponding user actions (*sm4am:UserAction*) would be stored and organized as a *ex:Query*. On the instance level it would contain a projection multidimensional operation regarding the gross revenue fact, a selection multidimensional operation for the year 2014, after which it keeps the previous preference, and finally the visualization with bar chart as inferred from the examples in Section 3.2. Several correlated user actions (e.g., *sm4am:PresentationAction*s) posed at different points of the analysis can represent a session. Moreover, this way different patterns can be captured/detected and used for user assistance purposes. If it is detected that the selection of a *f:Film* from a specific *f:Director* is usually followed by the selection of *t:DateTime* (i.e., a director made the best films in a certain year), this selection of *t:DateTime* can be automatically suggested in the next occasion when the preceding sequence to this action is detected. The Linked Data environments open even greater assistance opportunities as the patterns can include user actions over several data sources. These metadata can be used for approaches like [9], [15], [38], and [63].

User Elements

While the classes discussed by now capture evidence about the user actions, several classes are used for capturing evidence about the user. *sm4am:User*

is the main class representing the user that can have several user characteristics (*sm4am:UserCharacteristic*). Moreover, the user can belong to one or more *sm4am:UserGroup*s that represent the collection of users with defined user characteristics. User related classes are correlated by the properties illustrated in Figure A.12. It is important to outline the importance of the *sm4am:isConnectedTo* property that serves for modeling of different types of user interconnections that can be defined on the model level.

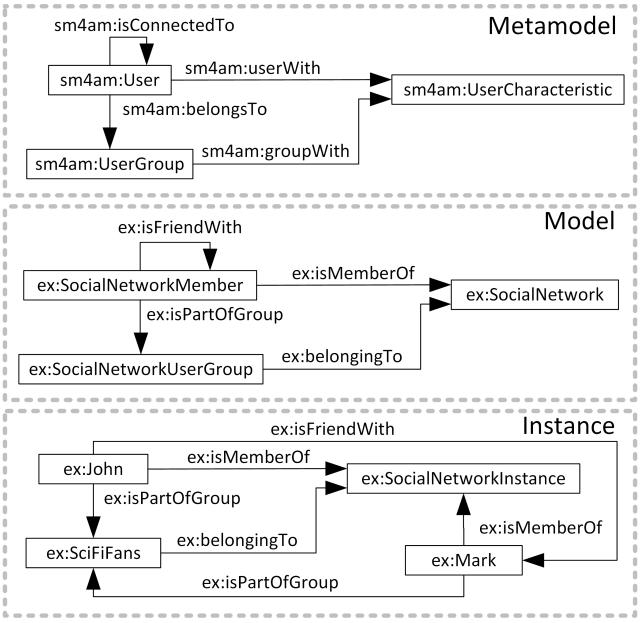

Fig. A.12: User Elements

Example. In BI 2.0 systems there are many different users and Figure A.12 illustrates an example related to a social network. On the model level, *ex:SocialNetworkMember* is characterized with *ex:SocialNetwork* as one of the user characteristics. A user belongs to *ex:SocialNetworkUserGroup* that also entails one or more user characteristics. The figure shows the instances of two users instantiating this model.

Many assistance possibilities are enabled by these metadata. For example, when querying the data set from our running example, recommendations of queries and chart types can be based on the user profile created as a set of user characteristics. User characteristics depend on the user base. For example, in case of a business user, characteristics are job position, company, and similar, while private user characteristics include age, gender, etc. Furthermore, different types of users belong to different social

networks, e.g., LinkedIn[9] in case of business users. User connections with other users and other characteristics can be used for collaborative recommendation techniques (see [6] for more details). In the context of Linked Data, the storage of user metadata in RDF format enables the linking of the same user accounts from different social networks and potential sharing and exploitation of these metadata. Current approaches typically focus on the user queries and sessions (i.e., user actions in our metamodel) for user assistance (e.g., [9], [15], [38]) and typically overlook the user characteristics. These metadata are especially important for the context-aware recommendations (see [7]) that are widely applied in the Web and still poorly exploited for BI 2.0 systems.

Data Property Elements

The pieces of evidence introduced by now have focused on the user and user actions. Although these pieces are our primary focus because of the user assistance, we also keep track of the data/system related evidence (i.e., the *sm4am:DataProperty* evidence) to enhance the user understanding of the explored data set. All data come from a certain data source and this information is captured by the *sm4am:DataSource* class with its `sm4am:sourceAttribute` defining specific attributes of the data source (e.g., external/internal flag). *sm4am:DataProperty* has *sm4am:DPPType* (i.e., data property type). All data property types are defined in the *sm4am:DPPDictionary* dictionary. These classes with the related properties are shown in Figure A.13 representing this piece of metamodel, with the examples of model and instance levels.

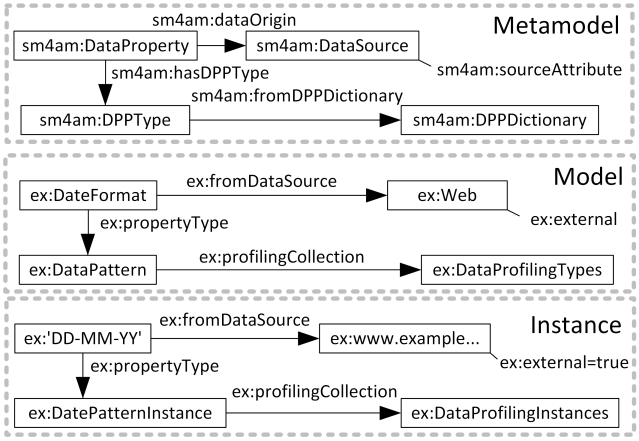

Fig. A.13: Data Property Elements

[9] www.linkedin.com

Example. As a model example we present the *ex:DateFormat* class which comes from a *ex:Web* data source and follows the *ex:DataPattern* type from the *ex:DataProfilingTypes* collection. This model is instantiated as shown in the figure.

Information that data are formatted by a certain pattern and come from a trusted data source can be exploited for the analysis of the data set and validity of stored instances. In the context of our running example, if for all films initial release date instances satisfy the pattern defined in Figure A.13, the user can be ensured that various aggregation functions can be performed over this dimension. For Linked Data that brings many external data sources, this information is essential for the user analysis when choosing among many different data sets. For example, the result inconsistencies may originate from distinct data patterns of different data sets. These metadata can be used for data quality portfolio that keeps track of the data accuracy, completeness, and similar.

4 Discussion

The SM4AM metamodel defined in the previous section captures pieces of evidence for the representation of AM (see [104]). To recapitulate, for representation of the *schema* artifact we use QB4OLAP [35], whereas *user characteristics* are defined by user related classes. Furthermore, *queries* and *preferences* are defined as ordered lists of concrete user actions (see Section 3.2 and Section 3.2). The *session* artifact is also covered as a specific user action list in Section 3.2. *Query log* information is captured through all the *queries* stored. *Profiling metadata* and *traceability metadata* are captured in the metamodel piece of Section 3.2. All previous pieces of evidence represent the *statistics* artifact. Finally, the RDF representation of AM allows that the whole metamodel plays the role of the *vocabulary* artifact. Assistance possibilities coming from AM storage and exploitation have been thoroughly discussed in [104].

Our SM4AM described in the RDF notation is one of the first steps towards user assistance support in the Linked Data context. These environments bring many new data published by the different entities where in particular government institutions provide Linked Open Data as publicly available information. The relation of Linked Data and BI 2.0 systems is becoming increasingly important. The area of BI is strongly influenced by the development of the Web and its trends. Hence, the BI environments must be open to handle and benefit from these new emerging resources. At the same time, well established BI principles are a sound foundation for addressing some similar challenges in the new environments. More concretely, we believe that the incorporation and integration of the external, often semi-structured, data with traditional BI systems can be successfully addressed by means of the Se-

mantic Web technologies such as ontologies (e.g., RDFS as a simple ontology language [5]) that support the mappings and automatic reasoning. If BI resources are represented by means of an ontology, they can be correlated with an external Linked Data source through ontology mapping techniques. An example of an RDF-based vocabulary representing a data warehouse schema can be found in [35] as mentioned before. On the other perspective, current Linked Data sources represent simple graph-organized data sets where querying possibilities are quite limited. If these data sets would be enriched with additional semantics and structure, where appropriate, a mature multidimensional model (as in the case of [35]) can be used for data sets exploration by existing techniques and mechanisms. As discussed in [104], we believe that the gathering and storage of AM would enable user assistance functionalities that have rarely, if at all, been supported in Linked Data environments. Moreover, the semantics present in the Linked Data architecture can be exploited for providing advanced user support functionalities based on the automatic reasoning over RDF schema and data structures. This would be a significant step towards BI 2.0 vision that we discussed in [104].

RDF for itself brings specific benefits even if not related to the Linked Data context. An especially important argument for its usage in BI 2.0 systems is the extensibility it enables. New metadata artifacts can be added to the metamodel just by creating new concepts and correlating them to the existing ones, i.e., creating new RDF triples. Moreover, all the model level instances can be directly linked to the metamodel classes and further, the instance level metadata can be directly related to the concrete models. The schema and instances are kept together so that the metamodel and model evolve together with the data and can be directly extracted at any point. Therefore, there is no need for additional external documentation efforts for schema changes.

Moreover, since there is a high variety of different systems and technologies in BI 2.0 environments, we propose SM4AM as a *metamodel*. Concrete models depend on the specific systems, and examples in Section 3 should have enhanced the understanding of how our approach helps to bridge this gap. A huge variety in the BI 2.0 ecosystem brings very different needs with respect to the types and numbers of users, queries, data sources, and other. These challenges can be met by joint efforts (e.g., creation and linking of specific models) around a high (metamodel) abstraction and a semantic-aware (RDF) formalism.

One of the strengths of our metamodel is the possibility of metadata reuse in various related data sets. For example, metadata captured in the example presented in Section 3.2 can be used for the user assistance in querying of IMDB[10] database. If properly linked and mapped, metadata captured in our examples can directly be used for assistance (e.g., query recommendation)

[10]http://http://www.imdb.com/

in IMDB exploration. In the context of Linked Data, the linking of concepts should be performed by the publisher. Thereby, the common vocabulary should be built by the joint effort of the participants.

The generalization and semantics that is to be achieved through the RDF formalization can be applied for the assistance in querying of data sets from different domains. Patterns captured by *sm4am:UAList* can be applied to the different multidimensional data sets sharing the same dimensions and/or semantics. For example, if the user always pairs two dimensions in her analysis (e.g., time and location) or uses specific sequence of dimension granularity (e.g., month and then year) these metadata can be applied for assistance within a data set containing these dimensions. The discovery of the data sets containing these dimensions should also be facilitated by linking mechanism of Linked Data.

Even though we have focused on the multidimensional data model, when discussing generalization, it is nonetheless important to notice that the metamodel can be used for other non-multidimensional data models. For that case, the schema metadata artifact, for which we use QB4OLAP [35] vocabulary, should be exchanged with an alternative data model, while the appropriate dictionaries of actions and characteristics should be provided for the rest of the metadata artifacts. For example, in the case of relational data set, all metadata artifacts can be adjusted to these settings e.g., relational algebra, preferences for relational algebra, etc.

Nevertheless, we acknowledge that there are also certain limitations currently carried by the RDF usage. Non-existence of cardinalities and open world assumption bring challenges that must be addressed for the incorporation of multidimensional semantics, e.g., the meaning of the aggregation in the open world assumption. Performance requirements (especially considering automatic reasoning) should also be considered in the future when providing implementation examples. However, the lack of explicit cardinalities could be addressed by the detection of implicit cardinalities extracted from the data, while performance is constantly improved by the advancement of both hardware and software technologies. These challenges will be addressed in our future work when exploiting the metamodel.

5 Related Work

The approaches for user assistance in the BI area typically use query logs, user sessions, and/or schema information, as for example the approaches in [9], [15], [26], [38], and [63]. They analyze the queries based on their syntax and/or result data, compare query sessions in the search for the next potential query, and analyze table relations for the querying assistance. Solutions like these store metadata on ad-hoc manners, in a none machine-processable

form, and provide limited assistance possibilities due to the lack of semantics. For example, similar queries that are differently formulated or return different results cannot be detected or compared. Furthermore, these approaches are mostly tied to the implementation technology and cannot be used in the combination with other solutions. Other approaches use additional metadata concepts for the user assistance. In [18], user profiles are used for visualization assistance where the profile is limited to certain visualization constraints and preferences. Moreover, in [40] the authors use preferences for the data analysis. Finally, the authors of [75] propose user assistance features for which they keep various statistics. However, neither of these approaches provides many details about the metadata used and, as adhoc metadata solutions, they have similar limitations to the ones discussed earlier.

A standardized solution for metadata management is presented in Common Warehouse Metamodel (CWM) [79] that is a standard for interchange of warehouse metadata. However, it does not fully cover AM artifacts (e.g., preferences) and is mostly intended for the interchange of data warehouse system metadata. Our metamodel mainly focuses on metadata needed for the user assistance purposes.

6 Conclusions and Future Work

SM4AM presented in this paper is a foundation for the management and organization of AM that is needed for user assistance purposes in BI 2.0 systems. To the best of our knowledge, no previous approach captures all these metadata in the single unified metamodel. Furthermore, our metamodel is described in RDF and thereby can easily be shared among various systems and technologies. It captures the metadata in a semantic-aware format so that they can be automatically processed. The paper is focused on the metadata management while exploitation of metadata will be elaborated in our future work as discussed in the next paragraph. The metamodel uses existing QB4OLAP proposal [35] as a solution for the RDF representation of the multidimensional data.

In our future work, as a first step towards the user assistance, we will use SM4AM to implement a dedicated metadata repository for AM artifacts. The repository will be a part of our own BI 2.0 system described in [57]. The primary focus will be on the metadata needed for query recommendation and personalization. Afterward, we will incrementally extend the repository, improve the user assistance functionalities, and explore novel exploitation possibilities of SM4AM such as system self-tuning.

7 Acknowledgments

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC) and it has been partly supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2011-24747.

Paper B

QB2OLAP: Enabling OLAP on Statistical Linked Open Data

The paper has been published in the
Proceedings of the 32nd IEEE International Conference on Data Engineering, pp. 1346-1349 (2016). The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1109/ICDE.2016.7498341

Abstract

Publication and sharing of multidimensional (MD) data on the Semantic Web (SW) opens new opportunities for the use of On-Line Analytical Processing (OLAP). The RDF Data Cube (QB) vocabulary, the current standard for statistical data publishing, however, lacks key MD concepts such as dimension hierarchies and aggregate functions. QB4OLAP was proposed to remedy this. However, QB4OLAP requires extensive manual annotation and users must still write queries in SPARQL, the standard query language for RDF, which typical OLAP users are not familiar with. In this demo, we present QB2OLAP, a tool for enabling OLAP on existing QB data. Without requiring any RDF, QB(4OLAP), or SPARQL skills, it allows semi-automatic transformation of a QB data set into a QB4OLAP one via enrichment

with QB4OLAP semantics, exploration of the enriched schema, and querying with the high-level OLAP language QL *that exploits the QB4OLAP semantics and is automatically translated to SPARQL.*

1 Introduction

OLAP analysis [101] is a well-established approach for decision making. Typically used in Data Warehousing (DW), OLAP relies on the MD model which represents data in terms of *facts* and *dimensions*. In short, dimensions conform the axes of an MD space in which a set of *measures* (associated to the fact) are represented. Dimensions provide appropriate contextual meaning to facts, and are organized as *hierarchies*, providing different *levels* of data aggregation. By means of an *MD algebra*, MD data are aggregated and disaggregated (through *roll-up* and *drill-down*, respectively), and filtered (through *slice* and *dice* operations), among other operations.

Initiatives like Open Data[1] are pushing organizations to publish MD data using standards and non-proprietary formats. Two main approaches can be followed for OLAP analysis of SW data. The first one aims at extracting MD data from the Web, and loading them into traditional DWs for OLAP analysis [58]. The second one (that we follow in our work) carries out OLAP-like analysis directly over MD data represented in RDF, following the notion of *self-service BI* [1].

Statistical data have traditionally been accessed and analyzed by means of OLAP [101]. In the SW, statistical data sets are usually published using the *RDF Data Cube Vocabulary*[2] (QB), a W3C recommendation since January, 2014. However, QB does not support the dimension hierarchies and aggregate functions needed for OLAP analysis. To address this challenge, a new vocabulary called QB4OLAP has been proposed [33]. QB4OLAP allows reusing data already published in QB by defining an MD schema containing the hierarchical structure of the dimensions (and the corresponding instances that populate the dimension levels). Once a data cube becomes published using QB4OLAP, we benefit from all the OLAP advances achieved in order to enable users to perform OLAP operations over the cube at a higher level of abstraction by using an OLAP algebra. In the demo, we present the *QB2OLAP* tool that can semi-automatically transform a QB data set into a QB4OLAP data set by enriching it with QB4OLAP semantics, explore the enriched schema (i.e., dimensions' structures and instances), and query the data set using a high-level OLAP language, denoted QL. *QB2OLAP* semi-automatically discovers dimension hierarchies to enrich the original data set, and automatically translates QL queries into SPARQL and executes them on an endpoint. Thus,

[1] http://okfn.org/opendata/

[2] http://www.w3.org/TR/vocab-data-cube/

QB2OLAP is a tool that facilitates data analysis, encouraging the use of MD data on the web. To our best knowledge, it is the first tool enabling native OLAP analysis on Statistical Linked Open Data.

Demo Use Case Mary is a journalist covering the European migration crisis. She wants to analyze historical migration data for the European Union (EU), and knows that these data[3] are provided by the statistical office of the EU (Eurostat) and are also available as Linked Open Data in QB format[4]. Mary wants to compute some basic filtering/ summaries, typical for OLAP, such as aggregate the origin nationality of immigrants per continent. However, due to the limited schema information, she soon realizes that it is not possible to perform OLAP operations. To do so, she would need to enrich the data set (e.g., with dimension hierarchies to roll-up through). Moreover, both enrichment and analysis require the use of SPARQL, a language that she cannot manage although she is quite proficient in OLAP. Fortunately, she knows about *QB2OLAP* and decides to use it to overcome her lack of technical knowledge on RDF, QB, and SPARQL. This demo shows how *QB2OLAP* can be used to achieve OLAP-like analysis over existing QB data sets and enable even wider analysis, e.g., analyze migration data according to the kind of political organization of the host countries. The original data set contains data about asylum applications from 2008 to 2014. For the demo purposes, we consider the subset of recent observations about asylum applications between 2013 and 2014, comprising approximately 80,000 observations.

2 Background: QB vs. QB4OLAP

A *QB data set* is a collection of so-called *observations* (in OLAP terminology *facts*) whose schema is specified by means of a *Data Structure Definition* (DSD) as an instance of the RDF class `qb:DataStructureDefinition`. This specification comprises a set of *component* properties representing *dimensions*, *measures*, and *attributes*, as shown below for a portion of the Eurostat data cube (RDF prefixes are omitted).

```
1 dsd:migr_asyappctzm rdf:type qb:DataStructureDefinition ;
2       qb:component [ qb:dimension sdmx-dimension:refPeriod] ;
3       qb:component [ qb:dimension property:age] ;
4       qb:component [ qb:dimension property:citizen] ;
5       ...
6       qb:component [ qb:measure sdmx-measure:obsValue] .
```

From an OLAP analyst's point of view, QB has the following limitations: (a) *No native support of dimension hierarchies.* OLAP operations rely on the organization of dimension members into hierarchies defined in terms of aggre-

[3] http://ec.europa.eu/eurostat/statistics-explained/index.php/Asylum_statistics
[4] http://eurostat.linked-statistics.org/

gation levels. QB only allows representing relationships between dimension instances. Thus, for example, although Mary knows that Nigeria aggregates to Africa, there is no way to express that Nigeria is a country, Africa a continent, and that countries aggregate to continents. (b) *No native support to represent aggregate functions.* Most OLAP operations aggregate measure values along dimensions in a cube using the default aggregate function defined for the measure, which is not present in QB. (c) *No support for descriptive attributes.* In the MD model, dimension levels are associated with a set of *attributes* that describe the characteristics of their members. Lack of descriptive attributes is not only awkward from a user's point of view, but also inefficient. For example, if Mary wants to ask only for applications from Nigeria, she would need to know the IRI representing Nigeria[5].

The QB4OLAP[6] vocabulary addresses the drawbacks above by representing the most common features of the MD model as shown in [35]. It is currently being used in several research projects concerning OLAP over RDF data. From a well-formed MD schema we can again automate most of the OLAP processing, as done in traditional DW settings. Importantly, QB4OLAP has been devised to operate over observations published in QB without the need of rewriting them. Typically, observations are the largest part of the data, while dimensions are usually orders of magnitude smaller.

A key difference between QB and QB4OLAP is that, in the latter, facts represent relationships *between dimension levels* and fact instances (observations) that map *level members* to measure values. The *dimension levels* are represented in the same way as *dimensions*, i.e., as *component* properties, and they can be linked to the DSD via the `qb4o:level` property. Similarly, *aggregate functions* are also *component* properties that are linked to the DSD via the `qb4o:aggregateFunction` property associating measures with aggregate functions. Moreover, QB4OLAP defines the `qb4o:cardinality` property that represents the cardinality of the relationship between a fact and a *dimension level*. Finally, *level attributes* can be linked to a *dimension level* via the `qb4o:hasAttribute` property. Below, we show the cube structure of the Eurostat data set, represented in QB4OLAP.

```
schema:migr_asyappctzmQB4O rdf:type qb:DataStructureDefinition;
    qb:component [ qb4o:level sdmx-dimension:refPeriod ;
                   qb4o:cardinality qb4o:ManyToOne ];
    qb:component [ qb4o:level property:citizen ;
                   qb4o:cardinality qb4o:ManyToOne ] ;
    ...
    qb:component [ qb:measure sdmx-measure:obsValue;
                   qb4o:aggregateFunction qb4o:sum ] ;
```

Furthermore, we also show a portion of the structure of the *Citizenship* dimension, discovered by the tool that we present later. We show the defini-

[5]Some data sets include a Label, although there is no guarantee about this.

[6]http://purl.org/qb4olap/cubes

tion of the dimension, the dimension levels (as part of the hierarchies), and hierarchy steps that represent roll-up relationship between levels. We point the interested reader to the QB4OLAP project's wiki[7] for details.

```
schema:citizenshipDim a qb:DimensionProperty ;
    qb4o:hasHierarchy schema:citizenshipGeoHier,
schema:citizenshipGeoHier a qb4o:Hierarchy ;
    qb4o:inDimension schema:citizenshipDim ;
    qb4o:hasLevel property:citizen, schema:continent, schema:citAll .
_:ih45 a qb4o:HierarchyStep ;
    qb4o:inHierarchy schema:citizenshipGeoHier ; qb4o:ChildLevel property:citizen ;
    qb4o:parentLevel schema:continent ; qb4o:pcCardinality qb4o:ManyToOne .
```

3 *QB2OLAP* Overview

QB2OLAP is organized in three main modules, *Enrichment*, *Exploration*, and *Querying*, as illustrated in Figure B.1. By using the *Enrichment* module, the user generates the QB4OLAP graph. Then, this semantics is exploited by the *Exploration* module enabling the user to explore the QB4OLAP schema and by the *Querying* module enabling OLAP analysis. The schema and level instance enrichment triples are loaded into a local SPARQL endpoint. All modules provide graphical interfaces. *QB2OLAP* automatically generates and triggers the necessary SPARQL queries and handles the result triples. The Querying module also gives the possibility to manually formulate SPARQL queries. Next, modules' details are explained.

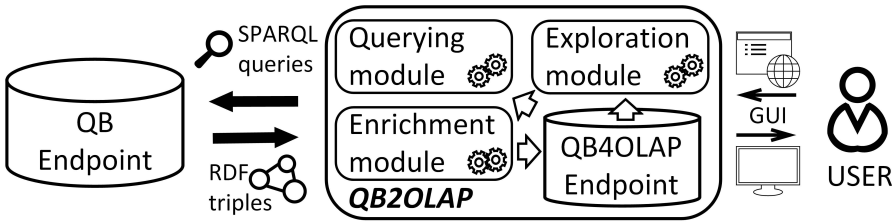

Fig. B.1: *QB2OLAP* Architecture

3.1 Enrichment module

The enrichment of the QB data set is a labor-intensive task that is semi-automatized in the Enrichment module [106]. The user is released of the burden to manually explore the data set, define dimension levels and hierarchies, and generate the corresponding QB4OLAP triples. Instead, the Enrichment module triggers the queries, performs the necessary processing, makes

[7] https://github.com/lorenae/qb4olap/wiki

suggestions for the user, and based on her choices enriches the schema. Thus, even an ordinary OLAP user can perform the enrichment on her own. The workflow of the Enrichment module is presented in Figure B.2.

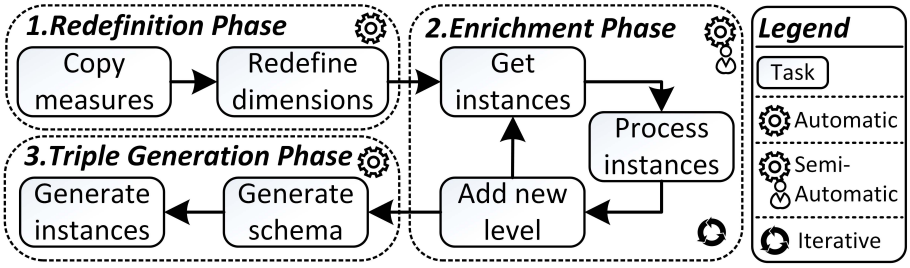

Fig. B.2: The Enrichment Module Workflow

The first phase is the *Redefinition Phase* where the input schema of the QB graph is adjusted according to the QB4OLAP semantics, i.e., dimensions are redefined as levels (e.g., `[qb:dimension property:citizen]` is redefined to `[qb4o:level property:citizen; qb4o:cardinality qb4o:ManyToOne]`) while measures are copied and an aggregate function is assigned to them (e.g., `[qb:measure sdmx-measure:obsValue]` to `[qb:measure sdmx-measure:obsValue; qb4o:aggregateFunction qb4o:sum]`). Starting from the levels of this redefined schema, the *Enrichment Phase* collects the level instances and their properties. A query is run for each level instance and the results are processed to discover the properties that represent *functional dependencies* (FD) which are typically used in MD modeling to automatically discover potential roll-up relationships [86]. Therefore, such properties are automatically suggested to the user as sound candidates for coarser granularity level(s) (e.g., `schema:continent` for `property:citizen`). The user then chooses out of the automatically discovered candidate properties the roll-up relationships of her interest and by doing so, we drastically prune the search space guided by the user preferences. The tasks of the *Enrichment Phase* are iteratively repeated until the user has added all desired levels and conformed the dimension hierarchies. When a new level is added, the dimension hierarchies are automatically constructed or updated (e.g., a portion of triples related to the previous levels `schema:citizenshipGeoHier a qb4o:Hierarchy; qb4o:hasLevel property:citizen, schema:continent`) and the new *candidate hierarchy levels* for the added level are again discovered. Finally, once the *Enrichment Phase* is over, the RDF triples are automatically generated for both the schema and schema instances in the *Triple Generation Phase*. The generated triples are then exploited in the Exploration and Querying modules. Additionally, the Enrichment module also enables configuring fine-tuning parameters for the aggregate function, level detection, and triple

generation. In the Linked Data dynamic context involving external and non-controlled data sources, the fine-tuning parameters that *QB2OLAP* offers are essential to deal with data quality issues, e.g., by searching for *quasi FDs* (i.e., an FD with an allowed error threshold).

The Enrichment module is implemented in Java 8. The Jena 2.13.0 library is used to manipulate RDF. QB and QB4OLAP graphs and the SPARQL endpoint are stored and run on Virtuoso 7 that is shared with the Exploration and Querying modules. The module interface is implemented in SWT.

3.2 Exploration and Querying modules

The *Exploration* module[8] allows to choose a data cube (represented in QB4-OLAP) among a collection of cubes stored in an endpoint and, in a user-friendly fashion, navigate its dimension structures and instances. Graphics allow to explore the dimension instances and group them in many ways (e.g., hierarchies, dimensions, etc.).

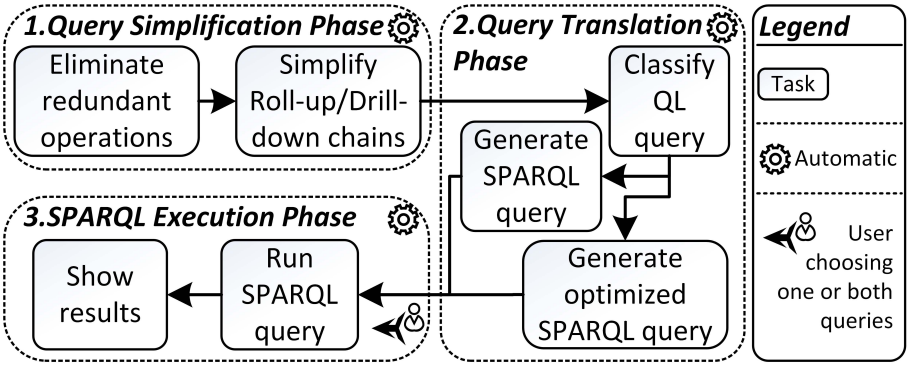

Fig. B.3: The Querying Module Workflow

The *Querying* module lets the user write QL queries (or load predefined example queries) in a query editor. Its workflow is presented in Figure B.3. QL follows the ideas introduced in the work by Ciferri et al. [29]. Basically, a QL program is a sequence of operations of the form (ROLLUP | SLICE | DRILLDOWN)* (DICE)*. Thus, we impose (for simplicity of processing) that dicing must always be written at the end of the QL program. In the *Query Simplification Phase* QL queries are then automatically simplified to produce better ones (e.g., the user may have included unnecessary operations, or written them in a non-optimal ordered sequence). The current implementation applies the following typical OLAP processing optimization rules: (a) perform SLICE operations as soon as possible, to reduce the size of intermediate

[8]https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/explorer

results; and (b) group all the ROLLUP and DRILLDOWN operations over the same dimension, and replace them with a single ROLLUP *from the dimension's bottom level* to the latest level reached by the sequence of ROLLUP/DRILLDOWN operation(s).

After simplifying and optimizing the QL query, it is automatically translated into a single SPARQL expression in the *Query Translation Phase* as explained next. ROLLUPs are implemented navigating the roll-up relationships between members, guided by the dimension hierarchy representation provided by the QB4OLAP metadata, and aggregations are performed using GROUP BY clauses. Navigation is performed through SPARQL graph patterns (corresponding to joins). Since SLICE removes dimensions, this requires measure values to be aggregated up to a single value in the dimension being sliced out. The mechanism for this is the same used as to compute a ROLLUP. Lastly, a DICE operation is associated with a condition over measures and/or attribute values, and its result filters out of cells in the cube that do not satisfy the condition. We implemented these conditions using SPARQL FILTER clauses[9]. This way, the QL query is classified according to the existing query patterns and two SPARQL queries are generated. Both are semantically equivalent and one represents the direct translation while the other is an alternative query generated using optimization heuristics thought to deal with some of the typical limitations of SPARQL endpoints. Finally, the user can choose to run either one or both queries and see the results in the *SPARQL Execution Phase*. The resulting cube is computed on-the-fly.

The Exploration and Querying modules are implemented in JavaScript and run on the Node.js platform. The interface of both modules is implemented with D3.js.

4 Demonstration

In the on-site demonstration, we will show how Mary, our journalist, can use the three modules of *QB2OLAP* to do her work. The following scenarios will be demonstrated.

Enrichment. Starting from the QB data set loaded into the endpoint, Mary can use the Enrichment module to interactively retrieve the cube structure and candidate properties pointing to the possible higher dimension levels. Using the graphical interface in Figure B.4, she is able to add new hierarchy levels to the cube. The cube structure is visualized as a tree that is updated after every change. Once all the levels are added, the triples representing the schema and level instances are loaded into the endpoint and used by the *Exploration and Querying* modules. We will also show that, in the presence of

[9]Details and examples of the translation process can be found in http://cs.ulb.ac.be/conferences/ebiss2015/files/slides/vaisman_ebiss2015.pdf

linked data sets, our tool is able to extract dimensional information (schema and instances) from other data sets (e.g., DBpedia).

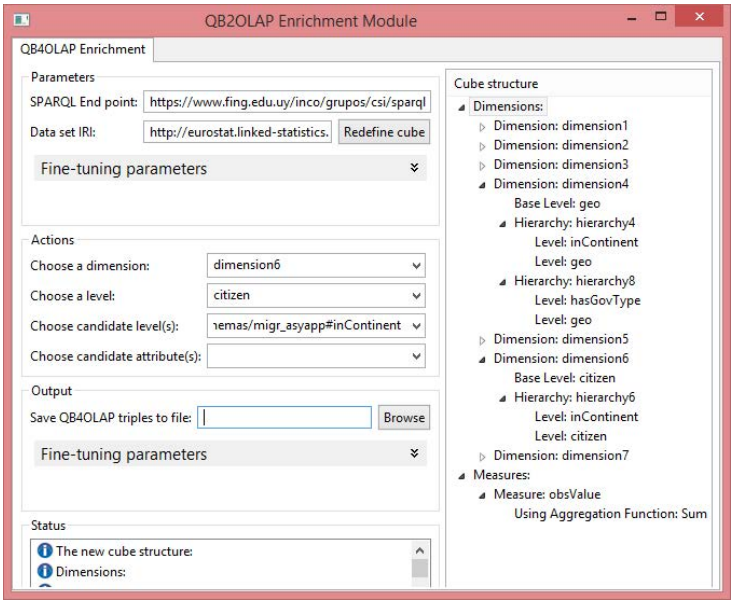

Fig. B.4: The Enrichment Example

Exploration and Querying. With the enriched data, Mary can now explore the cube dimensions, hierarchies, attributes, etc., through the graphical interface in Figure B.5. In the figure, Mary explores the dimensional cube data by clustering the instances according to their level value. Nodes represent level members (e.g., Syria) and edges represent roll-up relationships.

Once explored, Mary can write her own queries in QL (in the demo we include some predefined queries, which the audience can modify). For example, she can find the number of applications submitted by year by citizens from African countries whose destination is France, a query that could not be supported by the Eurostat site. The query (already simplified and rewritten) reads in QL:

```
PREFIX data: <http://eurostat.linked-statistics.org/data/>;
PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
QUERY
$C1 := SLICE (data:migr_asyappctzm, schema:asylappDim);
$C2 := ROLLUP ($C1, schema:citizenshipDim,schema:continent);
$C3 := ROLLUP ($C2, schema:timeDim, schema:year);
$C4 := DICE ($C3, (schema:citizenshipDim|schema:continent|
                schema:continentName = "Africa"));
$C5 := DICE ($C4, schema:destinationDim|property:geo|
                schema:countryName = "France");
```

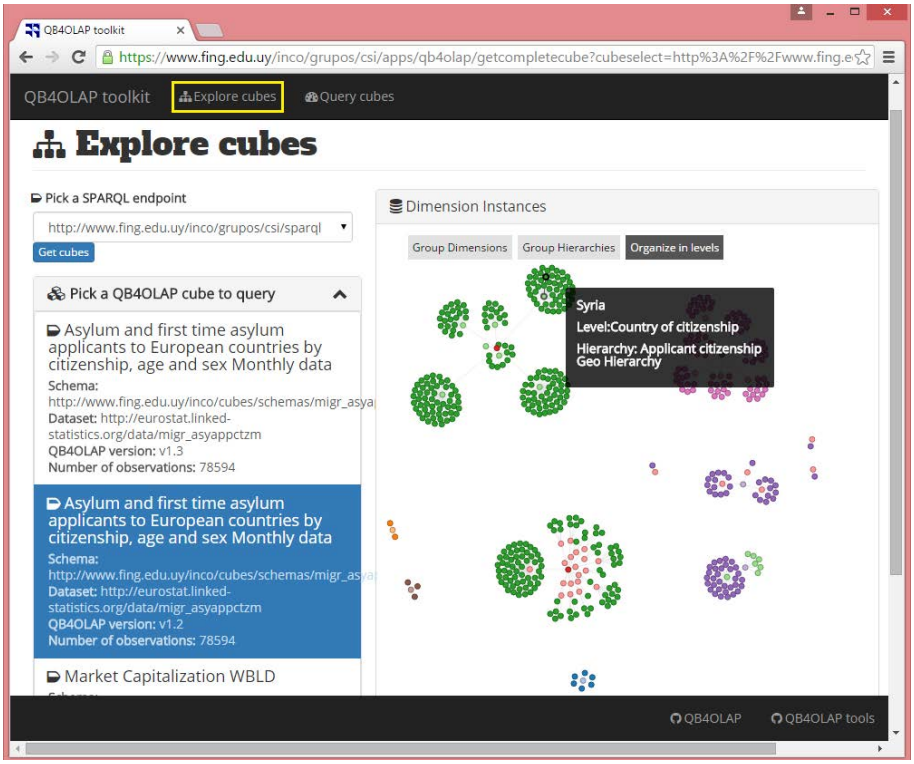

Fig. B.5: The Exploration Example

A key feature to promote the use of our proposal is to relieve OLAP users from the need of learning a new and complex language like SPARQL. QL provides a higher abstraction level that is more intuitive to typical OLAP users that only need to write relatively simple QL programs (e.g., the above query translates to more than 30 lines of SPARQL) using OLAP algebra operations. Thus, they have the flexibility to analyze data cubes on-the-fly, since QB4OLAP provides the metadata needed to automatically translate QL into SPARQL. Further, graphical OLAP tools can be developed, and translated first into a mediator language like QL, and then to SPARQL (we omit the SPARQL translation here, for space reasons).

ISSN (online): 2246-1248
ISBN (online): 978-87-7112-840-6

AALBORG UNIVERSITY PRESS