Aspects of Semantic ETL

Nath, Rudra Pratap

ASPECTS OF SEMANTIC ETL

BY
RUDRA PRATAP DEB NATH

DISSERTATION SUBMITTED 2020

AALBORG UNIVERSITY
DENMARK

Aspects of Semantic ETL

Ph.D. Dissertation

Rudra Pratap Deb Nath

Dissertation submitted June, 2020

Dissertation submitted:	June, 2020
PhD supervisor:	Professor Torben Bach Pedersen Aalborg University, Denmark
AAU PhD Co-Supervisor:	Professor Katja Hose Aalborg University, Denmark
UPC PhD Supervisor:	Professor Oscar Romero Universitat Politècnica de Catalunya, Spain
AAU PhD Committee:	Professor Kristian Torp (chairman) Aalborg University, Denmark Associate Professor Olaf Hartig Linköping University, Sweden Associate Professor Panagiotis Vassiliadis University of Ioannina, Greece
PhD Series:	Technical Faculty of IT and Design, Aalborg University
Department:	Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-657-1

Published by:
Aalborg University Press
Kroghstræde 3
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2020

Abstract

Business Intelligence (BI) tools support making better business decisions by analyzing available organizational data. Data Warehouses (DWs), typically structured with the Multidimensional (MD) model, are used to store data from different internal and external sources processed using Extract-Transformation-Load (ETL) processes. On-Line Analytical Processing (OLAP) queries are applied on DWs to derive important business-critical knowledge. DW and OLAP technologies perform efficiently when they are applied on data that are static in nature and well organized in structure. Nowadays, Semantic Web (SW) technologies and the Linked Data (LD) principles inspire organizations to publish their semantic data, which allow machines to understand the meaning of data, using the Resource Description Framework (RDF) model. One of the reasons why semantic data has become so successful is that managing and making the data available is low effort and does not rely on a sophisticated schema.

In addition to traditional (non-semantic) data sources, the incorporation of semantic data sources into a DW raises the additional challenges of schema derivation, semantic heterogeneity, and schema and data management model over traditional ETL tools. Furthermore, most SW data provided by business, academic and governmental organizations includes facts and figures, which raises new requirements for BI tools to enable OLAP-like analyses over those semantic (RDF) data. In this thesis, we 1) propose a layer-based ETL framework for handling diverse semantic and non-semantic data sources by addressing the challenges mentioned above, 2) propose a set of high-level ETL constructs for processing semantic data, 3) implement appropriate environments (both programmable and GUI) to facilitate ETL processes and evaluate the proposed solutions. Our ETL framework is a semantic ETL framework because it integrates data semantically. The following paragraphs elaborate on these contributions.

We propose SETL, a unified framework for semantic ETL. The framework is divided into three layers: the Definition Layer, ETL Layer, and Data Warehouse Layer. In the Definition Layer, the semantic DW (SDW) schema, sources, and the mappings among the sources and the target are defined.

In the ETL Layer, ETL processes to populate the SDW from sources are designed. The Data Warehouse Layer manages the storage of transformed semantic data. The framework supports the inclusion of semantic (RDF) data in DWs in addition to relational data. It allows users to define an ontology of a DW and annotate it with MD constructs (such as dimensions, cubes, levels, etc.) using the Data Cube for OLAP (QB4OLAP) vocabulary. It supports traditional transformation operations and provides a method to generate semantic data from the source data according to the semantics encoded in the ontology. It also provides a method to connect internal SDW data with external knowledge bases. Hence, it creates a knowledge base, composed of an ontology and its instances, where data are semantically connected with other external/internal data. We develop a high-level Python-based programmable framework, SETL for performing the tasks mentioned above. A comprehensive experimental evaluation comparing SETL to a solution made with traditional tools (requiring much more hand-coding) on the use case of Danish Agricultural and Business datasets shows that SETL provides better performance, programmer productivity, and knowledge base quality. The comparison between SETL and Pentaho Data Integration (PDI) in processing a semantic source shows that SETL is 13.5% faster than PDI. In addition to be faster than PDI, it deals with semantic data as first-class citizen, while PDI does not contain specific operators for semantic data.

On top of SETL, we propose $SETL_{CONSTUCT}$ where we define a set of high-level ETL tasks/operations to process semantic data sources. We divide the integration process into two layers: the Definition Layer and Execution Layer. The Definition Layer includes two tasks that allow DW designers to define target (SDW) schemas and the mappings between (intermediate) sources and the (intermediate) target. To create mappings among the sources and target constructs, we provide a mapping vocabulary called Source-To-Target Mapping (S2TMAP). Different from other ETL tools, we propose a new paradigm: we characterize the ETL flow transformations at the Definition Layer instead of independently within each ETL operation (in the Execution Layer). This way, the designer has an overall view of the process, which generates metadata (the mapping file) that the ETL operators will read and parametrize themselves with automatically. In the Execution Layer, we propose a set of high-level ETL operations to process semantic data sources. In addition to the cleansing, join, and data type based transformation for semantic data, we propose operations to generate multidimensional semantics at the data level and to update the SDW to reflect changes in the sources. In addition, we extend $SETL_{CONSTRUCT}$ for enabling automatic ETL execution flow generation (we call it $SETL_{AUTO}$). Finally, we provide an extensive evaluation to compare the productivity, development time, and performance of $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ with the previous framework SETL. The evaluation shows that $SETL_{CONSTRUCT}$ improves considerably over SETL in terms of productivity, development time, and performance. The evaluation shows that 1) $SETL_{CONSTRUCT}$ uses 92% fewer Number of Typed Characters (NOTC) than SETL, and $SETL_{AUTO}$ further reduces the Number of Used Concepts (NOUC) by another 25%; 2) using$SETL_{CONSTRUCT}$, the development time is almost cut in half compared to SETL, and is cut by another 27% using $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ is scalable and has similar performance compared to SETL.

Finally, we develop a GUI-based semantic BI system $SETL_{BI}$ to define, process, integrate, and query semantic and non-semantic data. In addition to the Definition Layer and the ETL Layer, $SETL_{BI}$ has the OLAP Layer, which provides an interactive interface to enable self-service OLAP analysis over the semantic DW. Each layer is composed of a set of operations/-tasks. We give an ontology to formalize the intra- and inter- layer connections among the components of the layer. The ETL Layer extends the the Execution Layer of $SETL_{CONSTUCT}$ by adding operations to process non-semantic data sources. We demonstrate the system using Bangladesh population census 2011 datasets.

The final solution of this thesis is the BI tool $SETL_{BI}$. $SETL_{BI}$ facilitates (1) DW designers with little/no SW knowledge to semantically integrate semantic and/or non-semantic data and analyze it in OLAP style, and (2) SW users with basic MD background to define MD views over semantic data for enabling OLAP-like analysis. Additionally, SW users can enrich the generated SDW's schema with RDFS/OWL constructs. Taking the framework as a base point, researchers can aim to develop further interactive and automatic integration framework for SDWs. This project bridges the traditional BI technologies and SW technologies which in turn will open the door of further research opportunities like developing machine-understandable ETL and warehousing techniques.

Resumé

Business Intelligence (BI) værktøjer understøtter at tage bedre forretningsbeslutninger, ved at analysere tilgængelige organisatoriske data. Data Warehouses (DWs), typisk konstrueret med den Multidimensionelle (MD) model, bruges til at lagre data fra forskellige interne og eksterne kilder, der behandles ved hjælp af Extract-Transformation-Load (ETL) processer. On-Line Analytical Processing (OLAP) forespørgsler anvendes på DWs for at udlede vigtig forretningskritisk viden. DW og OLAP-teknologier fungerer effektivt, når de anvendes på data, som er statiske af natur og velorganiseret i struktur. I dag inspirerer Semantic Web (SW) teknologier og Linked Data (LD) principper organisationer til at offentliggøre deres semantiske data, som tillader maskiner at forstå betydningen af denne, ved hjælp af Resource Description Framework (RDF) modellen. En af grundene til, at semantiske data er blevet succesfuldt, er at styringen og udgivelsen af af dataene er nemt, og ikke er afhængigt af et sofistikeret skema.

Ud over problemer ved overførslen af traditionelle (ikke-semantiske) databaser til DWs, opstår yderligere udfordringer ved overførslen af semantiske databaser, såsom skema nedarvning, semantisk heterogenitet samt skemaet for data repræsentation over traditionelle ETL værktøjer. På den anden side udgør en stor del af den semantiske data der bliver offentliggjort af virksomheder, akademikere samt regeringer, af figurer og fakta, der igen giver nye problemstillinger og krav til BI værktøjer, for at gøre OLAP lignende analyser over de semantiske data mulige. I denne afhandling gør vi følgende: 1) foreslår et lag-baseret ETL framework til at håndterer multiple semantiske og ikke-semantiske datakilder, ved at svare på udfordringerne nævnt herover, 2) foreslår en mængde af ETL operationer til at behandle semantisk data, 3) implementerer passende miljøer (både programmerbare samt grafiske brugergrænseflader), for at lette ETL processer og evaluere den foreslåede løsning. Vores ETL framework er et semantisk ETL framework, fordi det integrerer data semantisk. Den følgende sektion forklarer vores bidrag.

Vi foreslår SETL, et samlet framework for semantisk ETL. Frameworket er splittet i tre lag: et definitions-lag, et ETL-lag, og et DW-lag. Det seman-

tiske DW (SWD) skema, datakilder, samt sammenhængen mellem datakilder og deres mål, er defineret i definitions-laget. I ETL-laget designes ETL-processer til at udfylde SDW fra datakilderne. DW-laget administrerer lagring af transformerede semantiske data. Frameworket understøtter inkluderingen af semantiske (RDF) data i DWs ud over relationelle data. Det giver brugerne mulighed for at definere en ontologi for et DW og annotere med MD-konstruktioner (såsom dimensioner, kuber, niveauer osv.) ved hjælp af Data Cube til OLAP (QB4OLAP) ordforrådet. Det understøtter traditionelle transformations operationer, og giver en metode til at generere semantiske data fra de oprindelige data, i henhold til semantikken indkodet i ontologien. Det muliggør også en metode til at forbinde interne SDW data med eksterne vidensbaser. Herved skaber det en vidensbase, der er sammensat af en ontologi og dets instanser, hvor data er semantisk forbundet med andre eksterne / interne data. Vi udvikler et høj niveau Python-baseret programmerbart framework for at udføre de ovennævnte opgaver. En omfattende eksperimentel evaluering, der sammenligner SETL med en traditionel løsning (hvilket krævede meget manuel kodning), om brugen af danske landbrugs- og forretnings datasæt, viser at SETL præsterer bedre, programmør produktivitet og vidensbase kvalitet. Sammenligningen mellem SETL og Pentaho Data Integration (PDI) ved behandling af en semantisk kilde viser, at SETL er 13,5% hurtigere end PDI.

Udover SETL, foreslår vi $SETL_{CONSTRUCT}$ hvor vi definerer et sæt ETL-operationer på højt niveau til behandling af semantiske datakilder. Vi deler integrationsprocessen i to lag: Definitions-lag og eksekverings-lag. Definitions-laget indeholder to opgaver, der giver DW designere muligheden for at definere (SDW) skemaer, og kortlægningerne mellem kilder og målet. For at oprette kortlægning mellem kilderne og målene, leverer vi et kortlægnings ordforråd kaldet Source-to-Target Mapping (S2TMAP). Forskelligt fra andre ETL-værktøjer foreslår vi et nyt paradigme: vi karakteriserer ETL-flowtransformationerne i definitions-laget i stedet for uafhængigt inden for hver ETL-operation (i eksekverings-laget). På denne måde har designeren et overblik over processen, som genererer metadata (kortlægningsfilen), som ETL operatørerne vil læse og parametrisere automatisk. I eksekverings-laget foreslår vi en mængde høj niveau ETL-operationer til at behandle semantiske datakilder. Udover rensning, sammenføjning og datatypebaseret transformationer af semantiske data, foreslår vi operationer til at generere multidimensionel semantik på data-niveau og operationer til at opdatere et SDW for at afspejle ændringer i kilde-dataen. Derudover udvider vi $SETL_{CONSTRUCT}$ for at muliggøre automatisk ETL-eksekveringsstrømgenerering (vi kalder det $SETL_{AUTO}$). Endelig leverer vi en omfattende evaluering for at sammenligne produktivitet, udviklingstid og ydeevne for scon og $SETL_{AUTO}$ med den tidligere ramme SETL. Evalueringen viser, at $SETL_{CONSTRUCT}$ forbedres markant i forhold til SETL med hensyn til produktivitet, udviklingstid og ydeevne. Evalueringen viser, at 1) $SETL_{CONSTRUCT}$ bruger 92% færre antal indtastede tegn (NOTC) end SETL, og $SETL_{AUTO}$ reducerer antallet af brugte begreber (NOUC) yderligere med 25%; 2) ved at bruge $SETL_{CONSTRUCT}$, er udviklingstiden næsten halveret sammenlignet med SETL, og skæres med yderligere 27% ved hjælp af $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ er skalerbar og har lignende ydelse sammenlignet med SETL.

Til slut udvikler vi et GUI-baseret semantisk BI system $SETL_{BI}$ for at definere, processere, integrere og lave forespørgsler på semantiske og ikke-semantiske data. Ud over definitions-laget og ETL-laget, har $SETL_{BI}$ et OLAP-lag, som giver en interaktiv grænseflade for at muliggøre selvbetjenings OLAP analyser over det semantiske DW. Hvert lag er sammensat af en mængde operationer/opgaver. Vi udarbejder en ontologi til at formalisere intra-og ekstra-lags forbindelserne mellem komponenterne og lagene. ETL-laget udvider eksekverings-laget af $SETL_{CONSTUCT}$ ved at tilføje operationer til at behandle ikke-semantiske datakilder. Vi demonstrerer systemet ved hjælp af Bangladesh population census 2011 datasættet.

Sammenfatningen af denne afhandling er BI-værktøjet $SETL_{BI}$. $SETL_{BI}$ fremmer (1) DW-designere med ringe / ingen SW-viden til semantisk at integrere semantiske og / eller ikke-semantiske data og analysere det i OLAP stil, og (2) SW brugere med grundlæggende MD-baggrund til at definere MD-visninger over semantiske data, der aktiverer OLAP-lignende analyse. Derudover kan SW-brugere berige det genererede SDW-skema med RDFS / OWL-konstruktioner. Med udgangspunkt i frameworket som et grundlag kan forskere sigte mod at udvikle yderligere interaktive og automatiske integrationsrammer for SDW. Dette projekt bygger bro mellem de traditionelle BI-teknologier og SW-teknologier, som igen vil åbne døren for yderligere forskningsmuligheder som at udvikle maskinforståelige ETL og lagerteknikker.

Resum

Les eines d'Intel·ligència Empresarial (BI), conegudes en anglès com Business Intelligence, donen suport a la millora de la presa de decisions empresarials mitjançant l'anàlisi de les dades de l'organització disponibles. Els magatzems de dades, o data warehouse, (DWs), típicament estructurats seguint el model Multidimensional (MD), s'utilitzen per emmagatzemar dades de diferents fonts, tant internes com externes, processades mitjançant processos Extract-Transformation-Load (ETL). Les consultes de processament analític en línia (OLAP) s'apliquen als DW per extraure coneixement crític en l'àmbit empresarial. Els DW i les tecnologies OLAP funcionen de manera eficient quan s'apliquen sobre dades de natura estàtica i ben estructurades. Avui en dia, les tecnologies de la Web Semàntica (SW) i els principis Linked Data (LD) inspiren les organitzacions per publicar les seves dades en formats semàntics, que permeten que les màquines entenguin el significat de les dades, mitjançant el llenguatge de descripció de recursos (RDF). Una de les raons per les quals les dades semàntiques han tingut tant d'èxit és que es poden gestionar i fer que estiguin disponibles per tercers amb poc esforç, i no depenen d'esquemes de dades sofisticats.

A més de les fonts de dades tradicionals (no semàntiques), la incorporació de fonts de dades semàntiques en un DW planteja reptes addicionals tals com derivar-hi esquema, l'heterogeneïtat semàntica i la representació de l'esquema i les dades a través d'eines d'ETL. A més, la majoria de dades SW proporcionades per empreses, organitzacions acadèmiques o governamentals inclouen fets i figures que representen nous reptes per les eines de BI per tal d'habilitar l'anàlisi OLAP sobre dades semàntiques (RDF). En aquesta tesi, 1) proposem un marc ETL basat en capes per a la gestió de diverses fonts de dades semàntiques i no semàntiques i adreçant els reptes esmentats anteriorment, 2) proposem un conjunt d'operacions ETL per processar dades semàntiques, i 3) la creació d'entorns apropiats de desenvolupament (programàtics i GUIs) per facilitar la creació i gestió de DW i processos ETL semàntics, així com avaluar les solucions proposades. El nostre marc ETL és un marc ETL semàntic perquè Es capaç de considerar e integrar dades de forma semàntica. Els següents paràgrafs elaboren sobre aquests contribucions.

Proposem SETL, un marc unificat per a ETL semàntic. El marc es divideix en tres capes: la capa de definició, la capa ETL i la capa DW. A la capa de definició, es defineixen l'esquema del DW semàntic (SDW), les fonts i els mappings entre les fonts i l'esquema del DW. A la capa ETL, es dissenyen processos ETL per popular el SDW a partir de fonts. A la capa DW, es gestiona l'emmagatzematge de les dades semàntiques transformades. El nostre marc dóna suport a la inclusió de dades semàntiques (RDF) en DWs, a més de dades relacionals. Així, permet als usuaris definir una ontologia d'un DW i anotar-la amb construccions MD (com ara dimensions, cubs, nivells, etc.) utilitzant el vocabulari Data Cube for OLAP (QB4OLAP). També admet operacions de transformació tradicionals i proporciona un mètode per generar semàntica de les dades d'origen segons la semàntica codificada al document ontologia. També proporciona un mètode per connectar l'SDW amb bases de coneixement externes. Per tant, crea una base de coneixement, composta per un ontologia i les seves instàncies, on les dades estan connectades semànticament amb altres dades externes / internes. Per ferho, desenvolupem un mètode programàtic, basat en Python, d'alt nivell, per realitzar les tasques esmentades anteriorment. S'ha portat a terme un experiment complet d'avaluació comparant SETL amb una solució elaborada amb eines tradicional (que requereixen molta més codificació). Com a cas d'ús, hem emprat el Danish Agricultural dataset, i els resultats mostren que SETL proporciona un millor rendiment, millora la productivitat del programador i la qualitat de la base de coneixement. La comparació entre SETL i Pentaho Data Integration (PDI) mostra que SETL és un 13,5% més ràpid que PDI. A més de ser més ràpid que PDI, tracta les dades semàntiques com a ciutadans de primera classe, mentre que PDI no conté operadors específics per a dades semàntiques.

A sobre de SETL, proposem $SETL_{CONSTUCT}$ on definim un conjunt de tasques d'alt nivell / operacions ETL per processar fonts de dades semàntiques i orientades a encapsular i facilitar la creació de l'ETL semàntic. Dividim el procés d'integració en dues capes: la capa de definició i la capa d'execució. La capa de definició inclou dues tasques que permeten definir als dissenyadors de DW esquemes destí (SDW) i mappings entre fonts (o resultats intermedis) i l'SDW (potencialment, altres resultats intermedis). Per crear mappings entre les fonts i el SDW, proporcionem un vocabulari de mapping anomenat Source-To-Target Mapping (S2TMAP). A diferència d'altres eines ETL, proposem un nou paradigma: les transformacions del flux ETL es caracteritzen a la capa de definició, i no de forma independent dins de cada operació ETL (a la capa d'execució). Aquest nou paradigma permet al dissenyador tenir una visió global del procés, que genera metadades (el fitxer de mapping) que els operadors ETL individuals llegiran i es parametritzaran automàticament. A la capa d'execució proposem un conjunt d'operacions ETL d'alt nivell per processar fonts de dades semàntiques. A més de la neteja,

la unió i la transformació per dades semàntiques, proposem operacions per generar semàntica multidimensional i actualitzar el SDW per reflectir els canvis en les fonts. A més, ampliem $SETL_{CONSTRUCT}$ per permetre la generació automàtica de flux d'execució ETL (l'anomenem $SETL_{AUTO}$). Finalment, proporcionem una àmplia avaluació per comparar la productivitat, el temps de desenvolupament i el rendiment de $SETL_{CONSTRUCT}$ i $SETL_{AUTO}$ amb el marc anterior SETL. L'avaluació demostra que $SETL_{CONSTRUCT}$ millora considerablement sobre SETL en termes de productivitat, temps de desenvolupament i rendiment. L'avaluació mostra que 1) $SETL_{CONSTRUCT}$ utilitza un 92% menys de caràcters mecanografiats (NOTC) que SETL, i $SETL_{AUTO}$ redueix encara més el nombre de conceptes usats (NOUC) un altre 25%; 2) utilitzant $SETL_{CONSTRUCT}$, el temps de desenvolupament es redueix gairebé a la meitat en comparació amb SETL, i es redueix un altre 27 % mitjançant $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ es escalable i té un rendiment similar en comparació amb SETL.

Finalment, desenvolupem un sistema de BI semàntic basat en GUI $SETL_{BI}$ per definir, processar, integrar i consultar dades semàntiques i no semàntiques. A més de la capa de definició i de la capa ETL, $SETL_{BI}$ té una capa OLAP, que proporciona una interfície interactiva per permetre l'anàlisi OLAP d'autoservei sobre el DW semàntic. Cada capa està composada per un conjunt d'operacions / tasques. Per formalitzar les connexions intra i inter-capes dels components de cada capa, emprem una ontologia. La capa ETL amplia l'execució de la capa de $SETL_{CONSTUCT}$ afegint operacions per processar fonts de dades no semàntiques. Per últim, demostrem el sistema final mitjançant el cens de la població de Bangladesh (2011).

La solució final d'aquesta tesi és l'eina $SETL_{BI}$. $SETL_{BI}$ facilita (1) als dissenyadors del DW amb pocs / sense coneixements de SW, integrar semànticament les dades (semàntiques o no) i analitzar-les emprant OLAP, i (2) als usuaris de la SW els permet definir vistes sobre dades semàntiques, integrar-les amb fonts no semàntiques, i visualitzar-les segons el model MD i fer anàlisi OLAP. A més, els usuaris SW poden enriquir l'esquema SDW generat amb construccions RDFS / OWL. Prenent aquest marc com a punt de partida, els investigadors poden emprar-lo per a crear SDWs de forma interactiva i automàtica. Aquest projecte crea un pont entre les tecnologies BI i SW, i obre la porta a altres oportunitats de recerca com desenvolupar tècniques de DW i ETL comprensibles per les màquines.

Acknowledgements

Throughout this thesis, this is my favorite section because it allows me to express my gratitude from the deepest region of my heart to those who enriched me by their co-operations, inspirations, knowledge, wisdom, experiences, and positive vibes during my PhD journey.

Firstly, I would like to express my heartiest gratitude to my supervisor, Professor Torben Bach Pedersen. I do not whether he was special to me, or generally, he is like that. Either I am lucky, or as a result of my previous good deeds, I found him as a supervisor. He plays a significant role as a supervisor and true guardian during my PhD journey. I truly appreciate his professionalism, sharp memory, outstanding skills in understanding my academic and personal issues, and instant replies to the queries. His concrete and constructive comments, analytical skills, and visionary thinking helped me to shape my research skills and guided me in the right direction. Next, I would like to express my sincere gratitude to my co-supervisor, Professor Katja Hose. Since the first day of my journey, she has been supportive and taught me how to adapt with new social and research environments, do research in a structured and disciplined way. Her expertise, ideas, experience, wisdom, and organizing skills added new dimensions in my thought domain and improved my research quality. After that, I express my gratitude to my Host University co-supervisor Professor Oscar Romero. I think he is one of the very few persons who can understand my thinking and formulate it. I enjoyed each and every moment of working with him. He knows how to bring out my full potential. His encouragement, energy, knowledge, viewpoints, sense of humor, smiley face, ideas, analytical skills significantly improved my research skills and motivated me a lot. In short, I learned a lot of things in terms of academic, professional, and personal aspects from my supervisors, which helped me to bring this thesis in this condition.

Further, I would like to thank all my colleagues and friends both at Aalborg University and Universitat Politècnica de Catalunya. A special thank goes to Assorciate Professor Christina Thomsen for helping me in all IT4BI-DC related administrative tasks. I would also remember Ilkcan Keles, Suela Isai, Aamir Saleem, and S. R. Bhuvan for supporting me with their experiences and suggestions during my personal issues. I express my appreciation and gratitude to Helle Schroll and Helle Westmark for their support regarding the administrative matters. A special thank goes to Emil Riis Hansen for translating the abstract of this thesis in Danish. Professor Oscar Romero also deserves another thank for translating the abstract in Catalan. My thanks go to the anonymous reviewers of the papers included in this thesis. I would also like to give thanks to my PhD committee members: Professor Kristian Torp (Aalborg University), Associate Professor Olap Hartig (Linkoping University), and Associate Professor Panagiotis Vassiliadis (University of Ioannina).

I enjoyed working in both the Danish and Spanish environments. I made a lot of friends during this PhD journey. I appreciate Danish cultures, traditions, laws (especially, Janteloven, law of Jante), and their mentality. Therefore, My special thanks go to the Danish and Spanish governments to allow me to continue my PhD in their respective countries.

Finally, I would like to offer my gratitude to the energy of God and God Himself for everything, regardless of good and bad. Thanks for directing me in the paths of beyond duality and equal vision.

Giving thanks to all again mentioned above, I end this section.

Rudra Pratap Deb Nath
Aalborg, June 11, 2020

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

Thesis Details

Thesis Title:	Aspects of Semantic Data
Ph.D. Student:	Rudra Pratap Deb Nath
Supervisors:	Prof. Torben Bach Pedersen, Aalborg University
	Prof. Katja Hose, Aalborg University
	Prof. Oscar Romero, Universitat Politècnica de Catalunya

The main body of the thesis consists of the following papers.

[A] Rudra Pratap Deb Nath, Katja Hose, Torben Bach Pedersen, and Oscar Romero. "SETL: A programmable semantic extract-transform-load framework for semantic data warehouses.". In: *Information Systems*, Vol.68, pp. 17–43, Elsevier, 2017.

[B] Rudra Pratap Deb Nath, Oscar Romero, Torben Bach Pedersen, and Katja Hose. "High Level ETL for Semantic Data Warehouses". Submitted to: *Semantic Web Journal*, IOS Press, 2020.

[C] Rudra Pratap Deb Nath, Katja Hose, Torben Bach Pedersen, Oscar Romero, and Amrit Bhattacharjee. "$SETL_{BI}$: An Integrated Platform for Semantic Business Intelligence ". In: *Proceedings of the World Wide Web Conference 2020, Taipei* .

In addition to the main papers, Paper D has also been included. Paper A is the extended version of Paper D, thus Paper D can be considered as a subset of Paper A.

[D] Rudra Pratap Deb Nath, Katja Hose, and Torben Bach Pedersen: "Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses". In : *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP, DOLAP 2015, Melbourne, VIC, Australia*, pp.15-24, ACM, 2015.

This thesis has been submitted for assessment in partial fulfilment of the PhD degree. The thesis is based on the submitted or published scientific papers, which are listed above. Parts of the papers are used directly or indirectly in the summary of the thesis. As a part of the assessment, co-author statements have been made available to the assessment committee and those are also available at the at the Technical Faculty of IT and Design at Aalborg University and the Department of Service and Information System Engineering at Universitat Politècnica de Catalunya. The permission for using the published articles in the thesis has been obtained from the corresponding publishers with the conditions that they are cited and DOI pointers and/or copyrights/credits are placed prominently in the references.

Rudra Pratap Deb Nath
Aalborg University, June 11, 2020

Part I

Thesis Summary

Aspects of Semantic ETL

1 Introduction

1.1 Background and Motivation

Business Intelligence (BI) supports organizations by providing techniques and tools to derive intelligent business decisions by analyzing their data [31]. In a BI system, data come from different data sources and are integrated into a Data Warehouse (DW) for analytical purposes [35]. The integration process for extracting data from different sources, translating it according to the underlying semantics of the DW, and loading it into the DW is known as Extract-Transform-Load (ETL). Typically, a DW is designed following the Multidimensional (MD) model, where data are viewed in an n-dimensional space, generally known as a data cube, composed of facts (the cells of the cube) and dimensions (the axes of the cube) [39]. Dimensions are organized into hierarchies (composed of a number of levels) to explore and (dis)aggregate fact measures (e.g., numerical data) at various levels of detail. For example, the *Administrative* hierarchy ($ADM5 \rightarrow ADM4 \rightarrow ADM3 \rightarrow ADM2 \rightarrow ADM1$) of the *Geography* dimension allows to (dis)aggregate the *population* of Bangladesh at various administrative levels of detail. The MD model enables Online Analytical Processing (OLAP) [35] queries, where data cubes are explored through user-friendly interfaces. Hence, OLAP has been widely used by business and non-technical users for data analysis and decision-making [54].

DW and OLAP technologies perform efficiently when they are applied on data that are static in nature and well organized in structure [1]. Nowadays, Semantic Web (SW) technologies and the Linked Data (LD) principles inspire organizations to publish their business-related data, which allows machines to understand the meaning of data. Therefore, besides analyzing internal data available in a DW, it is often desirable to incorporate (external) semantic data sources into the DW to derive the needed business knowledge. Moreover, most SW data provided by international and governmental organizations include facts and figures, which can be described in an MD manner to enable OLAP-like analyses [32]. The semantics of data in semantic sources are specified by using Internationalized Resource Identifiers (IRIs), providing common terminology, semantically linking with published information, and providing further knowledge to allow reasoning [6].

Semantic data is expressed in Resource Description Framework (RDF) [57], where each resource is described by a set of statements called triples. One of the drawbacks of semantic data is that they are often described without any schema (i.e., only instances are delivered, not the schema) or with an incomplete schema. Moreover, different sources describe the same domain in their own implicit schema, introducing semantic heterogeneity problems.

In addition to the traditional (non-semantic) data, the incorporation of those (external) semantic data into a DW raises additional challenges of schema derivation, semantic annotation, semantic heterogeneity, as well as schema and data management model over traditional DW technologies and ETL tools. The main drawback of a state-of-the-art Relational Database Management System (RDBMS)-based DW is that it is strictly schema dependent and less flexible to evolving business requirements. To cover new business requirements, every step of the development cycle needs to be updated to cope with the new requirements. This update process is time-consuming and costly and is sometimes not adjustable with the current setup of the DW; hence, it introduces the need for a novel approach.

In summary, the limitations of traditional ETL tools to process semantic data sources are: (1) they do not fully support semantic-aware data, (2) they are entirely schema dependent (i.e., cannot handle data expressed without any schema), (3) they do not focus on meaningful semantic relationships to integrate data from disparate sources, and (4) they neither support to capture the semantics of data nor support to derive new information by active inference and reasoning on the data [43].

Thus, a DW with semantic data sources in addition to traditional data sources requires more powerful techniques to define, integrate, transform, update, and load data semantically, which are the research challenges of this thesis.

1.2 Semantic ETL

To build a DW system with heterogeneous (semantic and non-semantic) data sources, the integration process should be able to deal with data semantics as a first-class citizen. Semantic Web (SW) technologies fulfill these needs as they allow to add semantics on both data and schema level in the integration process. SW technologies aim at presenting and exchanging data in a machine-readable and understandable format.

On the SW, RDF is used for presenting and exchanging data in a machine-readable format. In the RDF data model, information is presented in a set of statements, called RDF triples. An RDF triple has three components: subject, predicate, and object and is formally defined as: $(v_1, v_2, v_3) \in (I \cup B) \times I \times (I \cup B \cup L)$, where I, B, L are a set of International Resource Identifiers (IRIs), a set of blank nodes and a set of literals, respectively, and $(I \cap B \cap L) = \emptyset$ [25]. An IRI is a string used to uniquely identify a resource on the Web. A blank node serves as a locally-scoped identifier for a resource that is unknown to the outer world, and literals are a set of lexical values [25]. An RDF triple represents a relationship between its subject and object described by its predicate.

To express richer constraints on data, formal languages such as RDF Schema (RDFS) [58] and the Web Ontology Language (OWL) [44] are used in combination with the RDF data model to define constraints that data must meet. Moreover, the expressivity of the source with MD semantics can be defined using the Data Cube (QB) [14] and Data Cube for OLAP (QB4OLAP) [19] vocabularies. Therefore, users can define the schema of a DW semantically. [39] refers to an MD DW that is semantically annotated both at the schema and data level as a semantic DW (SDW). An SDW is based on the assumption that the schema can evolve and be extended without affecting the existing structure. Hence, it overcomes the problems triggered by the evolution of an RDBMS-based data warehousing system. An SDW is considered as a Knowledge Base (KB), typically composed of a TBox and an ABox [39]. The TBox defines a domain in terms of concepts, properties, and terminological axioms whereas the ABox consists of assertions of the TBox [16]. Throughout this summary, "target schema", "SDW schema", and "target TBox" are used interchangeably.

To populate an SDW from different data sources, the integration process should consider the semantics of the data sources. Besides semantic sources, an SDW should support non-semantic sources by defining a semantic layer on top of the non-semantic sources. Besides cleansing and other traditional transformations (renaming, join, grouping, sorting, date/string/numerical transformations, etc.), the integration process should annotate MD semantics at the data level as well. Furthermore, changes to the data should be recorded and the semantic resources should be linked to external knowledge bases. Once an SDW is built, the next task is to derive business knowledge by evaluating OLAP queries on it.

Therefore, creating an SDW from heterogeneous (semantic and non-semantic) data sources requires an integration process. Figure 1 illustrates the concept of semantic ETL. A semantic ETL process is an integration process that fuses traditional DW and SW technologies to support, orchestrate, and manage all integration tasks semantically to create an MD SDW from diverse sources and to enable OLAP queries on it. The data in an MD SDW should be

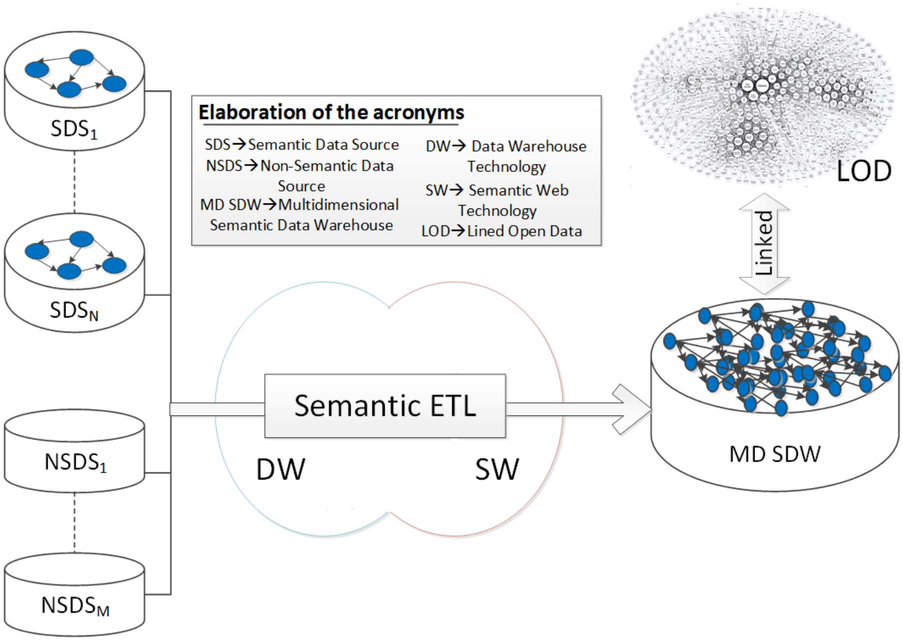

SDS$_1$
SDS$_N$
NSDS$_1$
NSDS$_M$
Elaboration of the acronyms
SDS→Semantic Data Source
NSDS→Non-Semantic Data Source
MD SDW→Multidimensional Semantic Data Warehouse
DW→ Data Warehouse Technology
SW→ Semantic Web Technology
LOD→Lined Open Data
LOD
Linked
Semantic ETL
DW
SW
MD SDW

Fig. 1: Understanding the concept of Semantic ETL

linked with external knowledge bases available in Linked Open Data(LOD) [1]. In summary, the semantic ETL covers the following aspects:

1. Definition of an SDW TBox with MD semantics.[2]
2. Data extraction from semantic and non-semantic sources.
3. Definition of source-to-target mappings to automatically parameterize the ETL operations.
4. A set of high-level conceptual ETL constructs (tasks/operations) to integrate semantic sources into an SDW.
5. Data transformation according to the (MD) semantics encoded into the target TBox and the source-to-target mapping file.
6. Update of the SDW to reflect the data changed in the source over time.
7. Linking the internal data with external KBs.
8. Loading the transformed data into a triple store.

[1]https://lod-cloud.net/

[2]For a semantic ETL process, this aspect is a precondition step. Since there are no standard tools to annotate an SDW with MD semantics in a correct and standard way, this aspect is included as an integral part of the semantic ETL framework presented in this thesis.

9. Enabling OLAP queries on the SDW.[3]

1.3 Objectives of the Thesis

The hypothesis of this thesis is:

"In the context of highly heterogeneous data integration processes, considering semantics as first-class citizen facilitates the integration process by providing automation and lower entry barriers for non-technical users."

The overall objective of this thesis is to develop a semantic ETL framework. The following incremental sub-objectives (i.e., they build on top of the previous ones) to meet this overall objective are defined:

1. Propose and develop an end-to-end semantic ETL framework to integrate semantic and non-semantic sources into an SDW: The framework should allow users to define the target TBox with MD semantics, extract semantic and non-semantic data, transform data according to the semantics of the target TBox, link data with external KBs, and load data into a triple store. The framework should be implemented using a high-level language so that ETL developers can accomplish different integration tasks considering semantics. A comprehensive experimental evaluation of the framework should be performed. This objective covers the following aspects of semantic ETL pointed in Section 1.2: 1, 2, 5, 7, and 8.

2. Propose and develop a prototype of a set of basic high-level ETL constructs (tasks/operations) to process semantic data sources and integrate semantic data into an SDW: Here, a set of ETL constructs (tasks/-operations) is proposed so that other integration tools can utilize those components to enable semantic integration. Further, three types of dimensional updates defined by Ralph Kimball in [34] should be applied in the context of an SDW. Importantly, there should be a mechanism to overcome manual mappings (typical of ETL tools) at the ETL operation level. The prototype should be evaluated in terms of productivity, development time and performance by comparing it with the framework proposed in Objective 1. This objective covers the following aspects of semantic ETL pointed in Section 1.2: 1, 2 (partly - only for semantic data sources), 3, 4 , 5, 6, 7, and 8.

3. Propose and develop a prototype of a semantic BI tool (like traditional BI tools) that supports, orchestrates, and manages all integration tasks and enables self-service OLAP analysis over an SDW: This GUI-based BI tool should enrich the framework of Objective 1 with the high-level

[3]In a traditional ETL, this aspect is outside of an ETL process. However, this aspect is included here to validate the quality of the produced MD semantic data.

ETL constructs discussed in Objective 2 and additionally add an OLAP interface to enable self-service OLAP-analysis over an SDW. This objective covers the following aspects of semantic ETL pointed in Section 1.2: 1, 2, 3, 4, 5, 6, 8, and 9.

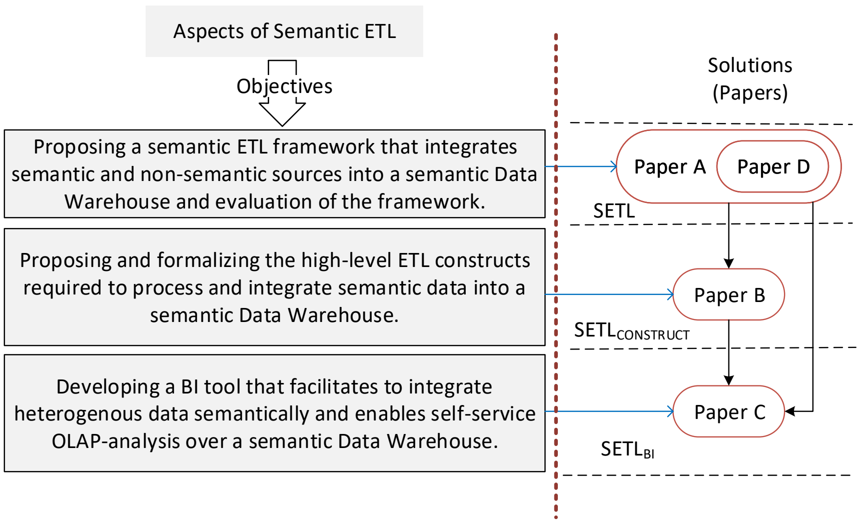

Aspects of Semantic ETL
Objectives
Proposing a semantic ETL framework that integrates semantic and non-semantic sources into a semantic Data Warehouse and evaluation of the framework.
Proposing and formalizing the high-level ETL constructs required to process and integrate semantic data into a semantic Data Warehouse.
Developing a BI tool that facilitates to integrate heterogenous data semantically and enables self-service OLAP-analysis over a semantic Data Warehouse.
Solutions (Papers)
Paper A
Paper D
SETL
Paper B
SETL_CONSTRUCT
Paper C
SETL_BI

Fig. 2: The connections between the objectives and the outcomes of the thesis.

Figure 2 shows the connection between the objectives and the solutions presented in this thesis. The right side of the figure shows how the papers published in the context of this thesis build on top of and relate to each other. For a complete reference of the papers here listed, check the Part II of this thesis. Paper A proposes a semantic ETL framework to integrate semantic and non-semantic data sources that sets the foundations for the rest of the papers. Paper A is an extended version of the initial idea published in paper D. Therefore, readers can skip Paper D when reading the thesis cover to cover. Paper B introduces a set of high-level conceptual ETL constructs (tasks/operations) to process semantic data sources. Paper C provides a semantic BI tool that enables semantic integration and self-service OLAP analysis.

1.4 Structure of the Thesis Summary

The summary is divided into three parts: 1) the first part presents a semantic ETL framework for integrating heterogeneous data in SDWs, 2) the second part proposes a set of high-level ETL constructs (operations/tasks) to process semantic data sources, and 3) finally, the third part provides a semantic BI tool and its application. Section 2 discusses the research related to the semantic ETL. Section 3 presents the first part of the thesis whereas Section 4 and Section 5 summarize the second and third parts of the thesis respectively. The contribution of the thesis are summarized in Section 6. Finally, Section 7 points to future work.

2 State of the Art

Nowadays, combining BI and SW technologies has become an emerging research topic as it opens the door to interesting research opportunities. As a DW deals with both internal and (increasingly) external data presented in heterogeneous formats, especially in RDF format, semantic issues should be considered in the integration process [1]. On the other hand, the availability of SW data gives rise to new requirements for BI tools to enable OLAP-style analysis over this type of data. Therefore, the existing research related to semantic ETL is divided into two lines: 1) on the one hand, the use of SW technologies to physically integrate heterogeneous sources and 2) on the other hand, enabling OLAP analysis over SW data.

A prominent research work following the first research line is [51], which presents an ontology-based approach for enabling the construction of ETL flows. At first, the schema of both the sources and the DW are defined by a common graph-based model, named the datastore graph. Then, the semantics of the datastore graphs of the data sources and the DW are described by generating an (OWL-based) application ontology, and the mappings between the sources and the target are established through that ontology. They address in this way the heterogeneity issues among the source and target schemata and finally demonstrated how the use of an ontology enables a high degree of automation from the source to the target attributes, along with the appropriate ETL transformations. Nonetheless, computationally complex ETL operations like slowly changing dimensions and the annotation of the application ontology with MD semantics are not addressed in this research. Therefore, OLAP queries cannot be applied on the generated DW.

Another existing work [5] aligned to this line has proposed a methodology describing some important steps required to make an SDW, which enables to integrate data from semantic databases. This approach also misses the annotation of SDW with MD semantics. [52] has proposed an approach to support data integration tasks in two steps: 1) constructing ontologies from XML and relational sources and 2) integrating the derived ontologies by means of existing ontology alignment and merging techniques. However, ontology alignment techniques are complex and error-prone. [4] presents a semantic ETL framework at the conceptual level. This approach utilizes the SW technologies to facilitate the integration process and discusses the use of different available tools to perform different steps of the ETL process. [2] presents a method to spatially integrate a Danish Agricultural dataset and a Danish Business dataset using an ontology. The approach uses SQL views and other manual processes to cleanse the data and Virtuoso for creating and storing integrated RDF data. [36] presents UnifiedViews, an open-source ETL framework that supports management of RDF data. Based on the SPARQL queries, they define four types of Data Processing Units (DPUs): Extractor, Transformer, Loader, and Quality Assessor. However, the DPUs do not support to generate MD RDF data.

About the second research line mentioned, a prominent study related to it is [42], which outlines a semi-automatic method for incorporating SW data into a traditional MD data management system for OLAP analysis. The method the authors proposed here allows an analyst to accomplish the following tasks: 1) designing the MD schema from the TBox of an RDF dataset, 2) extracting the facts from the ABox of the dataset and populating the MD fact table, and 3) producing the dimension hierarchies from instances of the fact table and the TBox to enable MDX queries over the generated DW. However, the generated DW no longer preserves the SW data principles defined in [27]; thus, OLAP analysis directly over SW data is yet to be addressed. With the purpose of fixing this issue, [11] introduces the notion of a *lens*, called the analytical schema, over the RDF dataset. An analytical schema is a graph of classes and properties where each node of the schema presents a set of facts that can be analyzed by traversing the reachable nodes. [28] presents a self-service OLAP endpoint for an RDF dataset. This approach first superimposes an MD schema over the RDF dataset. Then, a semantic analysis graph is generated on top of that MD schema where each node of the graph presents an analysis situation corresponding to an MD query, and an edge indicates to a set of OLAP operations.

Both [11] and [28] require either a *lens* or a semantic analysis graph to define MD views over an RDF dataset. Since most published SW data contains facts and figures, W3C recommends the Data Cube (QB) [14] vocabulary to standardize the publication of SW data with MD semantics. Although QB is appropriate to publish statistical data and several publishers (e.g., [47]) have already used the vocabulary for publishing statistical datasets, it has limitations to define MD semantics properly. The QB4OLAP [19] vocabulary enriches QB to support MD semantics by providing constructs to define 1) a cube structure in terms of different level of dimensions, measures, and attaching aggregate functions with measures and 2) a dimension structure in terms of levels, level attributes, relationships and the cardinality of relationships among the levels, and hierarchies of the dimension. Therefore, MD data can be published either by enriching data already published using QB with dimension levels, level members, dimension hierarchies, and the association of aggregate functions to measures without affecting the existing observations (for example, [56]) or using QB4OLAP from scratch (for example, [22], [30], and [39]).

In [33], the authors have presented an approach to enable OLAP operations on a single data cube published using the QB vocabulary and shown the applicability of their OLAP-to-SPARQL mapping in answering business questions in the financial domain. However, their OLAP-to-SPARQL mapping may not result in the most efficient SPARQL query and requires additional efforts and a longer time to get the results as they consider that the cube is queried on demand and the DW is not materialized. The authors in [54, 55] presented a semi-automatic method to enrich the QB dataset with QB4OLAP constructs. Nevertheless, the OLAP interface of their system requires end users to be familiar with either QL [10] or complex SPARQL queries to run OLAP queries.

After analyzing the main efforts on the two research lines identified when bridging DW and SW, we can draw some conclusions. Although each paper described above addresses one or multiple aspects of a semantic ETL framework, there is no single platform that supports all (target definition, source-to-target mappings generation, ETL generations, target population, evolution, update and OLAP-analysis). Further, there is no clear definition of how to orchestrate such steps into a single end-to-end flow. This thesis aims at developing a semantic ETL framework that supports, orchestrates, and manages all integration tasks to create an SDW and enable OLAP queries on it. To successfully reach the aim of the thesis, the framework has been designed in an incremental manner. First, a Python based programmable semantic ETL framework, named SETL is proposed in [39]. This is a kind of semantic version of pygramETL [53], a programmable ETL framework for traditional DWs. SETL provides various modules, classes, and methods to 1) extract semantic-aware data, geospatial data, and relational data, 2) integrate source data semantically using a target TBox defined with/without MD semantics, 3) store the data as semantic data, and 4) linking internal SDW data with external knowledge bases. Then, on top of SETL, $SETL_{CONSTRUCT}$ is proposed in [17] where the set of basic high-level ETL tasks/operations to process semantic data are discussed. Finally, a complete BI tool named $SETL_{BI}$ is developed in [16], which facilitates end-users to integrate heterogeneous data semantically and query on integrated data using interactive graphical interfaces.

3 A Semantic ETL Framework for Semantic Data Warehouses

This section is a summary of Paper A [39].

3.1 Motivation and Problem Statement

In addition to their internal organizational data, companies are (increasingly) showing interest in the knowledge encoded in large semantically annotated repositories that are locally or publicly available such as DBpedia [3], Freebase [8], Eurostat [4] or the datasets available in Linked Open Data (LOD). For example, a European company may want to analyze their internal product sales based on population and sex in different regions, which is available in Eurostat [17]. The use of SW technologies together with traditional DW technologies supports this open world scenario by addressing the limitations of current database-centric ETL tools discussed in Section 1.1. Therefore, the DW community is showing increasing interest in using SW technologies in the integration process. In [1, 4, 51], authors have used SW technologies to design conceptual frameworks of different phases of ETL processes; however, no one has so far offered an integrated framework to build a Semantic DW (SDW) that covers all integration phases, namely target definition, source to target mappings, (semantic and non-semantic) source extraction, transformation, linking, and loading [39]. In this section, a semantic ETL framework named SETL is presented. SETL allows to process and integrate heterogeneous data semantically by bridging SW and DW technologies. As discussed later, SETL supports all integration phases.

The unique contributions of SETL are: 1) It allows users to define the intensional knowledge (schema) of an SDW using ontological constructs. Users can annotate the schema with MD constructs (facts, dimensions, levels, cube, level attribute, and so on) using the QB [14] and QB4OLAP [19] vocabularies. 2) It allows processing both semantic and non-semantic data sources. A non-semantic data source is processed by creating a semantic layer on top of it. 3) It provides functionality to generate data (in the form of RDF triples) from diverse sources according to the semantics encoded in the schema of the SDW. 4) It also provides functionality to connect internal data with other internal/external data semantically [39]. A high-level Python-based programmable framework is developed for performing the integration tasks mentioned above. The programmable framework facilitates ETL developers by providing a higher abstraction level (in the form of modules, classes, and methods) that lowers the entry barriers. The following sections describe the use case used in this section, the architecture of SETL, the description of SETL components, and an overall discussion.

3.2 Use Case Description

This section describes the source datasets and the final target TBoxes for integrating and understanding the MD knowledge of those datasets. The three

[4]https://ec.europa.eu/eurostat/web/nuts/linked-open-data

datasets used in this paper are: 1) a Danish Agricultural Dataset (DAD) [21], 2) a Danish Business Dataset (DBD) [13], and 3) the European Union (EU) Farm Subsidy (Subsidy) dataset (`https://data.farmsubsidy.org/`). The datasets are integrated in two steps: 1) first, DAD and DBD are integrated to produce a SDW called SETLKB, and 2) then an MD SDW is produced by integrating the Subsidy dataset and the produced SETLKB. In this chapter, the focus is on the MD SDW part of the process; therefore, readers are directed to the original papers [15, 39] for the first step of integration.

The EU Farm subsidies for Denmark are collected from `https://data.farmsubsidy.org/`. This dataset has two MS Access database tables: *Subsidy* and *Recipient*. The *Subsidy* table contains the amount of subsidies given to the recipients in different years, whereas the *Recipient* table contains the information of who receive the subsidies. In total, there are 4,392,390 and 342,786 distinct rows in the *Subsidy* and *Recipient* tables respectively.

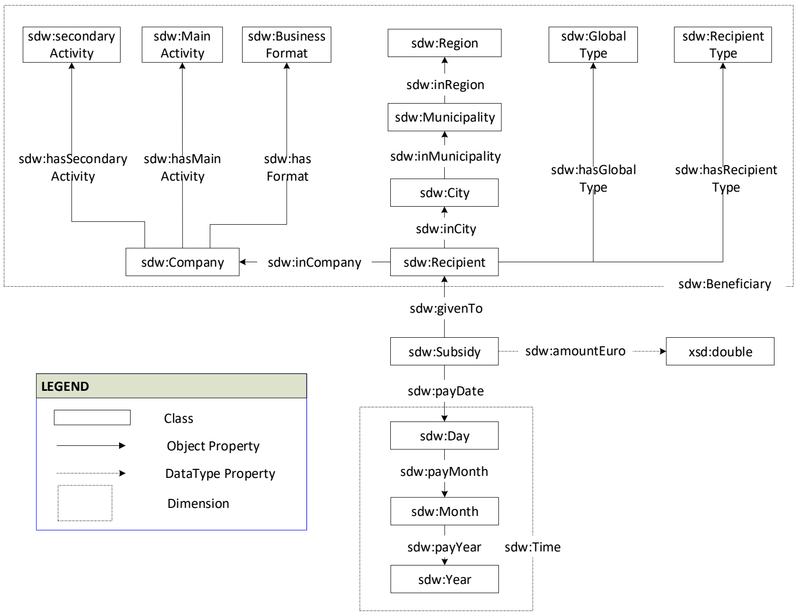


Fig. 3: The target TBox to multidimensionally integrate Subsidy and SETLKB datasets. For the sake of simplicity, the remaining data properties are not shown. (Adopted from [39])

The final target TBox, to multidimensionally integrate Subsidy and SETLKB datasets, is shown in Figure 3. Here, `sdw:Subsidy` is the central concept, and it provides a general description of the features for the facts of the MD SDW and the measure of facts (`sdw:amountEuro`) is shown as a datatype property. There are two dimensions in this SDW: `sdw:Time` and `sdw:Benificiary`. In Figure 3, the levels of each dimension are shown as a concept, and the hierarchies of each dimension—how the levels of each dimension are connected to each other through object properties to create different branches from `sdw:Subsidy`—are also shown.

3.3 SETL Overview

This section outlines the Semantic ETL Framework (SETL in short). To design an SDW, SETL follows a demand-driven approach. The two steps of the demand-driven approach are: 1) *requirements engineering:* identifying and analyzing the requirements of business users and decision makers, and 2) *data integration:* building the target TBox and the ETL based on the gathered requirements [23]. The first step is beyond the scope of this thesis, and SETL focuses on the second step. In short, the integration steps SETL supports are: defining a target TBox for the SDW based on the given requirements, extracting data from multiple heterogeneous data sources, transforming source data into RDF triples following the semantics encoded in the target TBox, linking data internally and externally, and loading the data into a triple store, and/or publishing the data on the Web as Linked Data [26, 29].

As shown in Figure 4, SETL is divided into three layers (separated by red-colored dotted lines): the Definition Layer, ETL Layer, and Data Warehouse Layer. In the Definition Layer, an ETL designer defines the target TBox, data sources, and the mappings among the source and target TBoxes. Using the *SDW TBox Definition* component, the designer defines the target TBox based on the requirements; the QB4OLAP vocabulary is used to annotate the target TBox with MD semantics. The *Define Mapping* component is used to map between a source and the target TBox. To create a semantic layer on top of a non-semantic data source, the *TBox Extraction* component is used to extract a TBox from a non-semantic data source. The *R2RML* component is used to generate RDB to RDF Mapping Language (R2RML) mappings for a non-semantic source, which is later used to generate RDF triples for the source using an R2RML engine.

In the ETL Layer, the designer can design an ETL process to create the ABox of the SDW from the available sources. The *Extraction* component retrieves data from the sources, which are further cleansed and formatted by the *Traditional Transformation* component. Then, the *Semantic Transformation* component transforms the data into RDF triples according to the semantics encoded in the target TBox. As a sub-task, *Semantic Transformation* stores the IRI information of each resource/data in the Provenance graph to preserve the uniqueness of each resource. Internal data can be linked with other external knowledge bases using the *External Linking* component. The RDF triples can be dumped as a local file using the *SaveToFile* component. Finally, the *Load* component loads the RDF triples, directly from the *Semantic Transformation* component or from the dumped file, into a triplestore that can be queried

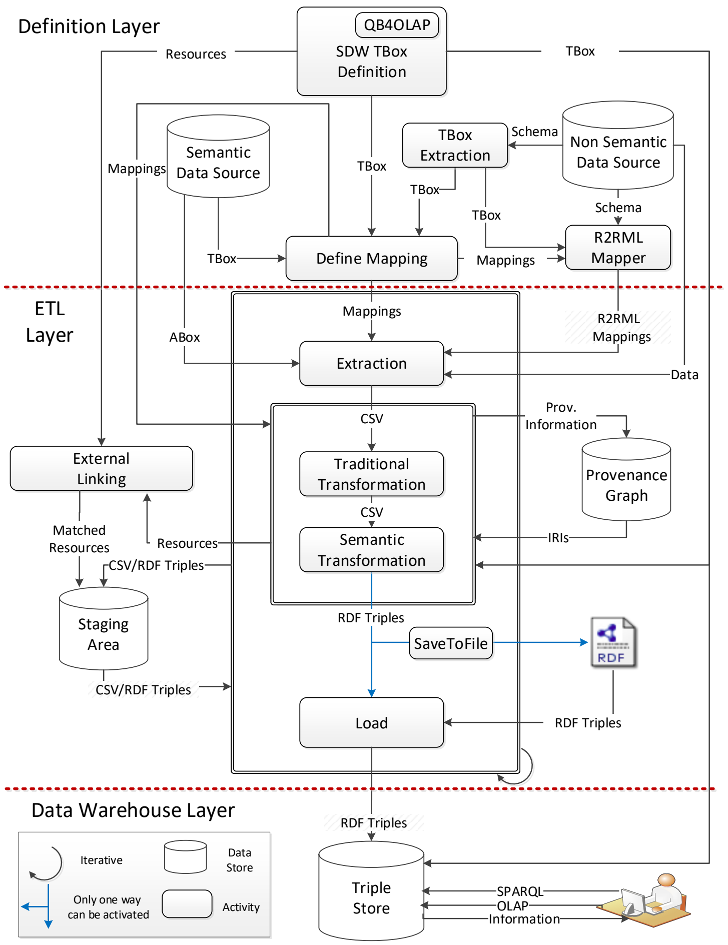


Fig. 4: SETL architecture. (Adopted from [39])

by end-users using OLAP or SPARQL queries. The intermediate results from each phase are stored in a data staging area. As ETL is an iterative block and repeated for each flow, a curved arrow is used in Figure 4. The Data Warehouse Layer stores the RDF triples. Next, the components of each layer are described in detail.

3.4 Component Description

This section describes the components of the Definition and ETL Layers.

The Definition Layer

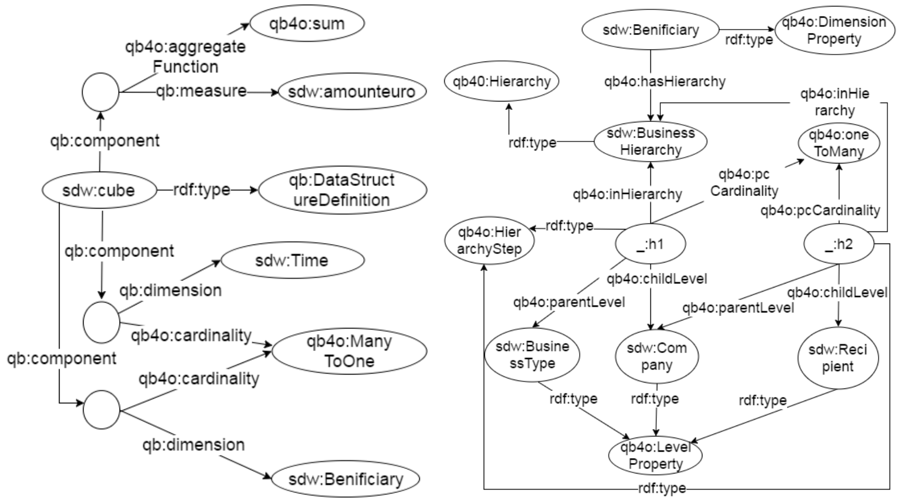
The Definition Layer is composed of four components: *SDW TBox Definition*, *TBox Extraction*, *Define Mapping* and *R2RML Mapper*, and an overview of each component is presented in the following paragraphs.

SDW TBox Definition As data sources are defined heterogeneously, it is vital to capture the semantics of the data at the conceptual level. To design an SDW, SETL uses ontological constructs (for example, Figure 3 shows the ontology of the SDW of our use case). SETL uses RDFS/OWL to define the formal semantics encoded in the ontology (TBox). In OWL, `owl:Class` and `rdfs:Property` are used to define concepts and properties of the TBox. The hierarchical relationships among the concepts and properties are presented by `rdfs:SubClassOf` and `rdfs:SubPropertyOf`, respectively, and `rdfs:domain` and `rdfs:range` are used to relate properties with concepts. As an SDW is defined with multidimensional constructs, SETL uses QB and QB4OLAP vocabularies to enrich the TBox with MD semantics.

In QB4OLAP, `qb:DataStructureDefinition` is used to define a cube/ cuboid structure in terms of dimensions/levels, measures, and attributes. A dimension (defined by `qb:DimensionProperty`) is composed of hierarchies, and a hierarchy (defined by `qb4o:Hierarchy`) logically orders the levels (defined by `qb4o:LevelProperty`) as a means of establishing parent-child relationships among the levels. A hierarchy is composed of hierarchy steps (defined by `qb4o:HierarchyStep`). Each hierarchy step defines the parent and child levels (through `qb4o:parentLevel` and `qb4o:childLevel` properties), the roll-up relationship and the cardinality between the levels (using the properties `qb4o:rollup` and `qb4o:pcCardinality`, respectively). The readers are directed to [18] for further details.

Figure 5a shows the cube structure `sdw:cube` of our use case. The structure is composed of QB components (defined by a blank node in the figure). Each QB component represents either a measure or a dimension. The aggregate functions (e.g., `qb4o:sum`) associated with the measure (e.g., `sdw:amounteuro`) also include in the measure component. A dimension component includes a dimension (e.g., `sdw:Beneficiary`) and its cardinality. The measure can be analyzed according to different dimensions, e.g., `sdw:Beneficiary`, `sdw:Time`. Figure 5b shows a fragment of the `sdw:Beneficiary` dimension. The `sdw:BusinessHierarchy` hierarchy (`sdw:Recipient` $\rightarrow$ `sdw:Company` $\rightarrow$ `sdw:BusinessType`) is composed of two hierarchy steps (defined by blank nodes `_:h1` and `_:h2`). Each hierarchy step defines its parent level, child level, and the cardinality of their relationship.

Define Mapping A mapping defines the relationships between the elements (either concepts or properties) of a source TBox and the corresponding ele-



(a) A fragment of the cube structure. **(b)** Beneficiary dimension definition.

Fig. 5: Description of a part of the cube structure and a dimension of our running example using QB4OLAP. (Reproduced from [39])

ments of the target TBox. Given a source and a target TBox T_S and T_T, a mapping can formally be defined as

$$Map(T_S, T_T) = \{(e_{s1}, e_{t1}, type) | e_{s1} \in T_S, e_{t1} \in T_T, type \in \{\equiv, \sqsubseteq, \sqsupseteq\}\} \quad (1)$$

Each 3-tuple $(e_{s1}, e_{t1}, type)$ in $Map(T_S, T_T)$ denotes that an element e_{s1} of T_S is mapped to an element e_{t1} of T_T with the relationship *type*. The relationship can be either an *equivalence* relationship ($e_{s1} \equiv e_{t1}$) or a *subsumption* relationship ($e_{s1} \sqsubseteq e_{t1}$) or a *supersumption* relationship ($e_{s1} \sqsupseteq e_{t1}$) [38]. OWL provides `owl:equivalentClass` for equivalence, and `rdfs:subClassOf`, and `rdfs:subPropertyOf` for *subsumption* and *supersumption* relationships.

TBox Extraction As mappings are done at the TBox level, a further step is necessary if the given source is either a non-semantic or a semantic source defined without schema. Currently, we focus on the extraction from non-semantic sources (e.g., relational, XML, and object-oriented). Extraction from a semantic source, where only instances are delivered, not the schema, is considered as a future work, addressed in Section 4. A TBox extraction process f from a data source D is defined as $f : D \rightarrow \mathcal{TB}$ where $\mathcal{TB}$ is the TBox derivation of the data source D. SETL follows the derivation techniques described in [52]. As an example, here we present the TBox extraction process for a relational data source RD as $f_{rd} : S \rightarrow \mathcal{TB}$, where S is the schema of RD. S is composed of a set of tables $\mathcal{T}$ and a set of columns $C = \{C_t | C_t \textit{ is a set of columns of a table } t \in T\}$. Table 1 summarizes the mappings between the elements of RD and the corresponding OWL constructs. Each table $t \in \mathcal{T}$ is an OWL class, each column (except foreign key column) is a datatype property, and each foreign key $FK_{t,t'}$ is an object property. The domain and range of the properties are shown within a square bracket in Table 1. Here, $type(c)$ returns the data type of a column. To extract a TBox from an XML source, readers are directed to [52].

x	$f_{rd}(x)$
$\forall t \in \mathcal{T}$	`owl:class`
$\forall c \in C_t$	`owl:DatatypeProperty` `[rdfs:domain =` $f_{rd}(t)$, `rdfs:range=`type(c)]
$\forall FK_{t,t'} \in C_t$	`owl:ObjectProperty` `[rdfs:domain =` $f_{rd}(t)$, `rdfs:range=`$f_{rd}(t')$]

Table 1: TBox extraction function from RD. (Reproduced from [39])

R2RML Mapper R2RML is the W3C standard to express customized mappings from relational data sources to RDF [49]. The *R2RML* component is used to map the relational data to RDF using the extracted TBox. since it is possible that not all the input data is relevant, the *R2RML* component also takes the output of the *Define Mapping* component as input. The mappings are done by users. Note that, to map from other sources (XML, CSV, JASON) to RDF, RML (`https://rml.io/specs/rml/`), which is the superset of R2RML proposed to support other structured formats, can be used.

The ETL Layer

The ETL Layer is composed of five components: *Extraction*, *Traditional Transformation*, *Semantic Transformation*, *External Linking*, and *Load* component. Details are discussed below.

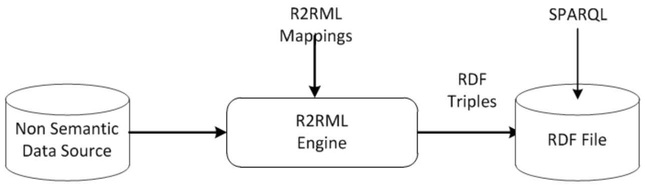


Fig. 6: The process of extraction from `Non Semantic Data Source`. (Reproduced from [39])

Extraction SETL allows extracting data from the available semantic and non-semantic sources. SPARQL [24] is the W3C standard pattern matching language for querying a semantic source. It uses SELECT queries to retrieve data and CONSTRUCT queries to produce an RDF graph (composed of a set of RDF triples). SETL applies SPARQL queries to a local RDF file or through a SPARQL endpoint to extract data from a semantic source. Figure 6 illustrates the process of data extraction from a non-semantic source. There a semantic version of the non-semantic source is created using an R2RML engine. An R2RML engine takes a relational database and an R2RML mapping file as input and generates an RDF dataset based on the mapping file [7]. Then, the RDF version of the non-semantic source can be queried using SPARQL queries.

Once the data is extracted, the next task is to transform the extracted data (given in the CSV format) according to the semantics encoded in the target TBox. SETL divides the transformation tasks into two components: *Traditional Transformation* and *Semantic Transformation*.

Traditional Transformation The *Traditional Transformation* component includes operations from traditional ETL tools to performs the formatting and cleansing tasks. SETL considers the following transformation tasks: duplicate data removal, data normalization, renaming attributes, recalculating data, integrity constraints checking, data filtering, unique identifier generation, default value handling, and sorting/grouping/summarizing data. This component outputs a set of tables, where each table corresponds to a concept construct (e.g., an OWL class, an MD level, an MD cube/dataset) of the target TBox, and the columns and records of each table represent the properties/level attributes and instances of that concept.

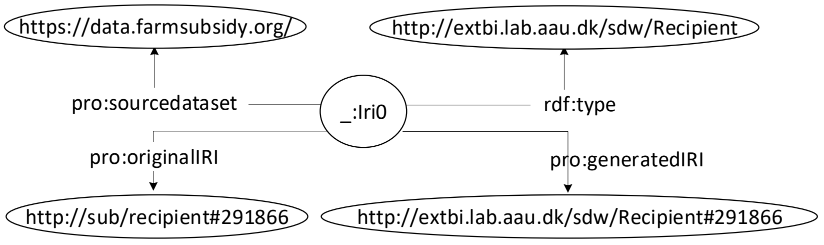

https://data.farmsubsidy.org/
http://extbi.lab.aau.dk/sdw/Recipient
pro:sourcedataset
_:Iri0
rdf:type
pro:originalIRI
pro:generatedIRI
http://sub/recipient#291866
http://extbi.lab.aau.dk/sdw/Recipient#291866

Fig. 7: An entry into the Provenance Graph. (Adopted from [39])

Semantic Transformation The *Semantic Transformation* component takes each output table produced by the *Traditional Transformation* component and transforms it into RDF data with MD semantics. The process is described in Algorithm 1. The algorithm takes the following inputs: a table (*table*), the name of a column in *table* (*resourceKey*) to be used to uniquely identify each

Algorithm 1: createTriples. (Adopted from [39])

```
Input: $table, resourceKey, onto, mapping, provGraph, dataset_{name}$
Output: $T$
begin
1   $D \leftarrow createDictionary(table)$
2   $concepts \leftarrow getMatchingConcepts(table, mapping)$
3   foreach $c \in concepts$ do
4       foreach $r \in D$ do
5           $sub \leftarrow createInstanceIRI(c, resourceKey(r), provGraph)$
6           $T.addtriple(sub, \texttt{rdf:type}, c)$
7           if $(type(c) = \texttt{qb4o:LevelProperty})$ then
8               $T.addtriple(sub, \texttt{qb4o:memberOf}, c)$
9           if $(type(c) = \texttt{qb:Observation})$ then
10              $T.addtriple(sub, \texttt{rdf:type}, \texttt{qb:Observation})$
11              $T.addtriple(sub, \texttt{qb:dataSet}, dataset_{name})$
12          foreach $(attribute, value_a) \in r$ do
13              if $(value_a != NULL)$ then
14                  $prop \leftarrow getMatchingProperty(attribute,$
                    $mapping, property(c, onto))$
15                  $T.addTriple(sub, prop, createObject(value_a))$
16  return $T$
```

row, a target TBox (*onto*), a source-to-target mapping file (*mapping*), a provenance graph (*provGraph*), and the name of the dataset ($dataset_{name}$) to be created. The first step is to create a temporal structure, a dictionary D (line 1), which is composed of *(key, value)* pairs. Each *(key, value)* pair represents a distinct row r in *table*. *key* corresponds to r's *resourceKey* value, and *value* presents a list of $(attribute, value_a)$ pairs, where each pair present one of r's cell. The next task is to identify the set of concepts in the target TBox that map to *table* (line 2). Then, the algorithm runs for each target mapped concept $c \in C$. For each entry in D, it creates a new resource and a triple defining the new resource as an instance of c (lines 5-6). The *createInstanceIRI(c, resourceKey(r), provGraph)* function first checks the availability of an existing IRI for the instance in the *Provenance Graph provGraph*; if it finds one, then returns it, otherwise creates a new one and updates the *Provenance Graph*. The *Provenance Graph* is an RDF graph that stores the meta information of the produced IRIs. Figure 7 illustrates an example entry of the IRI `http://extbi.lab.aau.dk/sdw/Recipient#291866`, its type in the target TBox, its original source dataset and IRI in the *Provenance Graph*. Based on the MD nature of c, the algorithm add QB4OLAP triples to define the MD semantics in the resource (lines 7-11). Furthermore, for each $(attribute, value_a)$, a triple is created if $value_a$ is not null and a mapping between $attribute$ and a property of c exists (lines 12-15). The function $createObject(value_a)$ returns either the literal $value_a$ or a resource based on the type of property (datatype or object).

External Linking The *External Linking* component links internal resources with external resources. For a given internal resource in, SETL provides an algorithm that extracts top k matching external resources either 1) from an external knowledge base by submitting a query to its endpoint or 2) by sending a web service request embedding in through the API (e.g., DBpedia lookup API). To find potential links, for each external resource ex, the Jaccard Similarity of between the semantic bags of in and ex are computed. The semantic bag of a resource consists of triples describing the resource [40, 41, 50]. The pair of the internal and external resources is considered as a match if the Jaccard Similarity exceeds a certain threshold. A triple with the `owl:sameAs` property is created to materialize the link in the dataset for each pair of matched internal and external resources.

Load SETL loads the transformed RDF triples and the triples generated in the `External Linking` component into a triple store (Jena TDB) or saves them to an RDF dump file. It supports two types of data loading: trickle load and bulk load. In the trickle load mode, data are transformed and fed to the triple store as it arrives without staging on a disk using SPARQL INSERT queries. Alternatively, the bulk load mode writes the transformed triples into a file on the disk, which is then loaded in the triple store in batch mode.

Tools⇒	**SETL**			**PDI (Kettle)**		
Task⇓	**Used Tools**	**Used Lnaguages**	**LOC**	**Used Tools**	**Used Languages**	**LOC**
TBox with MD semantics	Built-in SETL	Python	312	Protege, SETL	Python	312
Ontology Parser	Built-in SETL	Python	2	User Defined Class	Java	77
Semantic Data Extraction through SPARQL endpoint	Built-in SETL	Python, SPARQL	2	Manually extraction using SPARQL endpoint	SPARQL	N/A
Semantic Data Extraction from RDF Dump file	Built-in SETL	Python	2	N/A	N/A	N/A
Reading CSV/Database	Built-in Petl	Python	2	Drag & Drop	N/A	Number of used Activity: 1
Cleansing	Built-in Petl	Python	36	Drag & Drop	N/A	Number of used Activity: 19
IRI Generation	Built-in SETL	Python	2	User Defined Class	Java	22
Triple Generation	Built-in SETL	Python	2	User Defined Class	Java	60
External Linking	Built-in SETL	Python	2	N/A	N/A	N/A
Loading as RDF dump	Built-in SETL	Python	1	Drag & Drop	N/A	Number of used Activity: 1
Loading to Triple Store	Built-in SETL	Python	1	N/A	N/A	N/A
Total LOC for the complete ETL process			**401**	**471 LOC +4 N/A + 21 Activity**		

Table 2: Comparison between the ETL processes of SETL and PDI for SETLSDW. (Reproduced from [39])

3.5 Discussion

Each component of the SETL framework described in Section 3.3 is implemented using Python because it supports programmer productivity and has comprehensive standard libraries. For the implementation details, readers are referred to Paper A [39]. SETL is compared to the state-of-the-art, namely, ExtBI [2] and Pentaho Data Integration (PDI) [9]. As discussed in Section 3.2, a knowledge base, SETLKB, is created using SETL, which is used to make a comprehensive comparison with the process used in ExtBI. Readers are directed to Paper A [39] for the detailed experiment and evaluation. The following are the main advantages of SETL compared to ExtBI. 1) SETL makes the programmer more productive, as it uses only a fraction (570) of the Lines of Code (LOC) of 1879+5 N/A[5](by ExtBI) to complete the ETL process. 2) SETL only requires programming in Python instead of 7 different languages and tools. 3) The ETL process of SETL is run in a single pass, whereas ExtBI could not run the ETL process in a single pass because of using different tools and languages. 4) SETL produces a better knowledge base than the one produced by ExtBI in terms of consistency, completeness, and external linking.

In the second step of the use case data integration, discussed in Section 3.2, an MD SDW is produced using SETL, and the process is compared with PDI. Table 2 summarizes the tools, languages, and the LOC used to process integration tasks by SETL and PDI [6]. In total, PDI takes 471 LOC + 4 N/A + 21 Activity[7] to run the ETL process, whereas SETL takes only 401 LOC to run the complete ETL process. Hence, SETL creates an MD SDW with less number of LOC, user interactions than PDI where a user requires to build their own Java class, plugins, and manual tasks to enable semantic integration. As PDI does not allow all tasks related to semantic integration, a part of the ETL process generated by SETL is compared with PDI (Details are in [39]). SETL takes only 1644 seconds, where PDI takes 1903 seconds. Thus, SETL is 13.5% faster than PDI.

In summary, this chapter proposes a semantic ETL Framework to integrate semantic and non-semantic data sources. The experiments show that the programmable framework performs better in terms of programmer productivity, knowledge base quality, and performance. However, to use it, a user requires a programming background. Although it enables to create an ETL flow and provides methods by combining several tasks, there is a lack of a well-defined set of basic semantic ETL operators that allow users more control in creating their ETL process. Moreover, how to update the SDW to

[5]An N/A (Not Available) indicates that the task is done by either user interactions or manually or is not supported by the process. Therefore, the LOC cannot be counted.

[6]https://github.com/pentaho

[7]An activity is an operation directly supported by PDI with a drag and drop facility

synchronize it with the changes taking place in the sources is not discussed. Further, in a data lake /big data environment, the data may come from heterogeneous formats, and the use of the relational model as the canonical model may generate an overhead. Transforming JSON or XML data to relational to finally generate RDF can be avoided by using RDF as the canonical model instead. Further, several works have discussed the appropriateness of knowledge graphs for data integration purposes and, specifically, as a canonical data model [12]. An additional benefit of using RDF as a canonical model is that it allows adding semantics without being compliant to a fixed schema.

In Sections 4 and 5, the limitations identified in this discussion will be further elaborated. Specifically, Section 4 presents a set of semantic ETL operators that can be used to generate semantically rich and complex ETL flows. Doing so, the focus of the research moves from programmatic functions in Python to orchestrating high-level operators. These operators are further designed to use RDF as the canonical model. Also, some additional tasks to facilitate SDW management are introduced, such as the update of dimensional data to adapt to changes in the sources. In Section 5, a tool to perform high-level OLAP operations on top of an SDW is presented. This way, the user does not need to implement OLAP queries by means of SPARQL and can directly use operators such as roll-up or drill-down.

4 High-Level ETL Constructs for Processing Semantic Data

This section is a summary of Paper B [17].

4.1 Motivation and Problem Statement

Since there are a number of available RDF wrappers that can translate non-semantic resources to semantic resources, this section focuses on processing semantic sources and the usage of the RDF model as the canonical model in the integration process. The rest of this section explains the improvements introduced on top of SETL to remedy its main drawbacks identified in Section 3.5. Although SETL integrates semantic and non-semantic data sources, however, a set of high-level ETL constructs (tasks/operations) required to process semantic data is missing. As a consequence, in SETL, an ETL designer has to create specific handcode modules to deal with semantic data. Moreover, the output of the Define Mapping component in Figure 4 does not represent the complete scenario of the integration process, i.e., a mapping file needs to be separately created for each extraction and transformation module. Furthermore, there is a need to update the SDW to reflect the changes that occur in the sources over time. Finally, SETL is developed in Python and requires a programming background to accomplish the integration tasks. Therefore, users with no programming knowledge cannot use SETL.

Paper B defines the integration process based on two layers: the Definition and the Execution Layer and a set of high-level ETL constructs (tasks/operations) for each layer, with the aim of overcoming the drawbacks identified above for SETL. Different to SETL or other ETL tools, a new paradigm is proposed: the ETL flow transformations are characterized at the Definition Layer instead of independently within each ETL operation (in the Execution Layer). This is done by generating a mapping file that gives an overall view of the integration process. This mapping file is our primary metadata source and will be read by the ETL operations, orchestrated in the ETL flow (Execution Layer), to parametrize themselves. Thus, we are centralizing the creation of the required metadata to automate the ETL process in the Definition layer.

To create the mapping file, we use our proposed source-to-target mapping (S2TMAP) vocabulary. Besides other MD transformation operations, Paper B also proposes an ETL operation that allows applying three types of dimension updates (Type 1, Type 2, and Type 3) to accommodate evolution in an SDW, which were defined by Ralph Kimball in [34]. Furthermore, an ETL operation to extract a TBox from a semantic source defined without schema is also presented. Most importantly, an algorithm to automatically generate ETL data flows from the mappings file is proposed. Finally, a GUI-based prototype, named $SETL_{CONSTRUCT}$, based on the innovative elements mentioned above, is introduced. Hence, developers with no programming background can accomplish the integration tasks by interacting with the system's interfaces. Moreover, developers can create ETL flows either manually using drag-n-drop options or automatically using the auto ETL feature. The following sections present the overview of the integration process, the component description of each layer, and an overall discussion.

4.2 Layer-Based Semantic Data Integration Process

As discussed in Section 3.3, this thesis focuses on the *data integration* step (step 2) of a demand-driven approach. Here, we introduce a two-layered integration process. In the Definition Layer, a single source of metadata truth is defined. This includes: the target SDW, semantic representation of the source schemas, and a source to target mapping file. Relevantly, the metadata created represents the ETL flow at the schema level. In the Execution Layer, ETL data flows based on high-level operations are created. This layer executes the ETL data flows for instances (i.e., at the data level). Moving back to a kind of algebraic view of the ETL process gives freedom to the user to express richer and complex flows. Importantly, the ETL operations will read the metadata created and parameterize themselves automatically. Additionally, the Execution Layer automatically checks the correctness of the flow

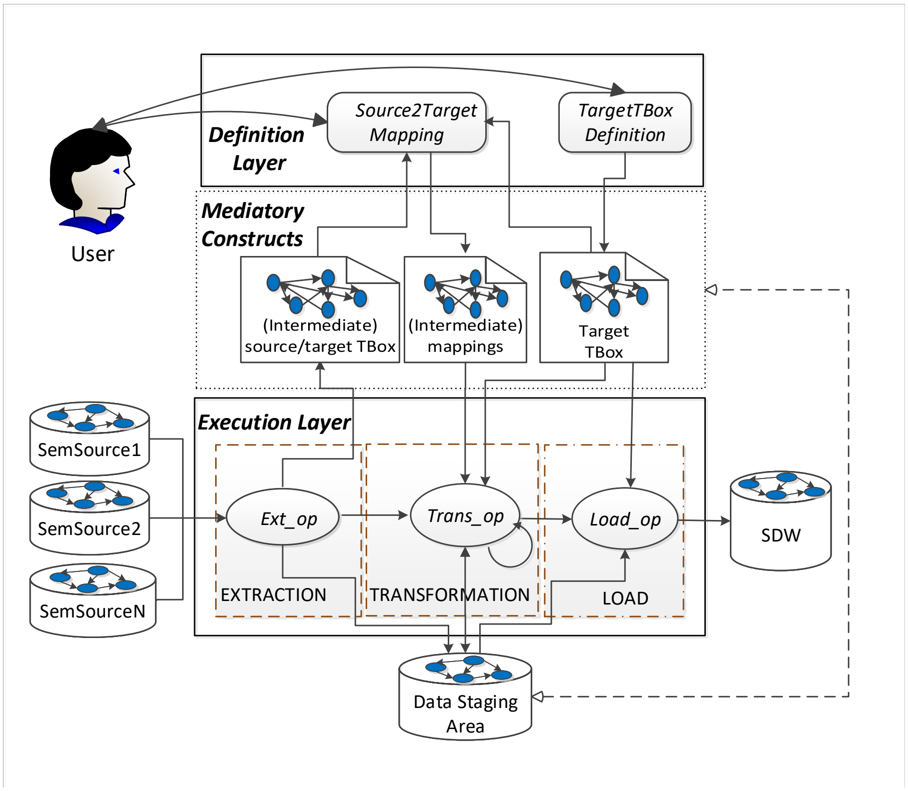


Fig. 8: The overall semantic data integration process. (Reproduced from [17])

created, by checking the compatibility of the output and input of consecutive operators. Overall, the data integration process requires the following four steps in the detailed order.

1. Defining the target TBox with MD semantics using QB and QB4OLAP constructs. In addition, the TBox can be enriched with RDFS/OWL classes and properties. However, $SETL_{CONSTRUCT}$ does not validate the correctness of the added semantics beyond the MD model. This step is done at the Definition Layer.

2. Extracting source TBoxes from the given sources. This step is done at the Definition Layer.

3. Creating mappings among source and target constructs to characterize ETL flows. The created mappings are stored using the S2TMAP vocabulary proposed. This step is also done at the Definition Layer, too.

4. Populating the ABox of the SDW implementing ETL flows. This step is done at the Execution Layer.

Operation Category	**Operation Name**	**Compatible Successors**	**Objectives**
Extraction	GraphExtractor	GraphExtractor, TBoxExtraction, TransformationOnLiteral, JoinTransformation, LevelMemberGenerator, ObservationGenerator, DataChangeDetector, UpdateLevel, Loader	It retrieves an RDF graph in terms of RDF triples from semantic data sources.
Transformation	TBoxExtraction		It derives a TBox from a given ABox.
	TransformationOnLiteral	TransformationOnLiteral, JoinTransformation, LevelMemberGenerator, ObservationGenerator, Loader	It transforms the source data according to the expressions described in the source-to-target mapping.
	JoinTransformation	TransformationOnLiteral, JoinTransformation, LevelMemberGenerator, ObservationGenerator, Loader	It joins two data sources and transforms the data according to the expressions described in the source-to-target mapping.
	LevelMemberGenerator (QB4OLAP construct)	Loader	It populates levels of the target with the input data.
	ObservationGenerator (QB4OLAP construct)	Loader	It populates facts of the target with the input data.
	DataChangeDetector	LevelMemberGenerator, UpdateLevel	It returns the differences between the new source dataset and the old one.
	UpdateLevel	Loader	It reflects the changes occurred in the source data to the target level.
	MaterializeInference	Loader	It enriches the SDW by materializing the inferred triples.
	ExternalLinking	Loader	It links internal resources with external KBs.
Load	Loader		It loads the data into the SDW.

Table 3: Summary of the ETL operations. (Reproduced from [17])

Figure 8 illustrates the whole integration process and how the constructs of each layer communicate with each other. There are two types of constructs: tasks and operations. On the one hand, a task requires user interactions with the interface of the system to produce an output. Intuitively, one may consider tasks as the required metadata to automate operations. On the other hand, from the given metadata, an operation automatically parameterizes itself and produces an output. The Definition Layer includes two tasks: *TargetTBoxDefinition* and *SourceToTargetMapping* and the *TBoxExtraction* operation. These two tasks respectively address the first and third steps of the integration process mentioned above, and the *TBoxExtraction* operation addresses the second step. This is the only operation shared by both layers (See the input of *SourceToTargetMapping* in the figure). Therefore, the Definition Layer creates three types of metadata: Target TBox (created by *TargetTBoxDefinition*), source TBoxes (create by *TBoxExtraction*), source-to-target mappings (created by *SourceToTargetMapping*). The Execution Layer covers the fourth step of the integration process and includes a set of operations to create data flows from the sources to the target. Figure 8 shows constructs (i.e., the Mediatory Constructs) used by the ETL constructs to communicate between them. These mediatory constructs store the required metadata created to automate the process. In the figure, Ext_{op}, $Trans_{op}$ and $Load_{op}$ are the set of extraction, transformation, and load operations used to create the flows at the instance level. The ETL data flows created in the Execution Layer are automatically validated and to do that, the concept of compatible operations is defined. Precisely, if an operation O_1's output is accepted by O_2, , then we say O_2 is compatible with O_1 and express it as $O_1 \rightarrow O_2$.

Further, as discussed in Section 3.5, traditional ETL tools (e.g., PDI) do not have ETL operations supporting the creation of an SDW. Here, a set of ETL operations are proposed. Table 3 summarizes the operations introduced based on their category, name, compatible successors, and objectives. Next, we discuss the details of each layer.

4.3 The Definition Layer

This layer contains two tasks (*TargetTBoxDefinition* and *SourceToTargetMapping*) and one operation *TBoxExtraction*. An overview of each construct is presented below.

TargetTBoxDefinition The objective of this task is to define a target TBox with MD semantics. The two main constructs of the MD structure are dimensions and cubes, which can be described using the QB and QB4OLAP vocabularies. Section 5 described how QB and QB4OLAP are used to describe a target TBox with MD semantics. The same principles hold in this task. Importantly, this task guarantees that any SDW target schema generated is QB4OLAP-compliant. In addition, the TBox can be enriched with RDFS/OWL classes and properties. However, for this additional enrichment, this task does not guarantee the correctness of the added semantics. Listing 1 represents the `sdw:Time` dimension and `sdw:Subsidy` cube of the use case described in Section 3.2 in QB4OLAP. Importantly, we extended the QB4OLAP vocabulary to enable updates in dimensions (for more details see [17]). Therefore, the `qb4o:updateType` property is added to each level attribute (lines 36, 40, 44, 48, 52, and 56).

Listing 1: QB4OLAP representation of `sdw:Time` dimension and `sdw:Subsidy` cube. (Reproduced from [17])

```
1 PREFIX sdw: <http://extbi.lab.aau.dk/ontology/sdw/>
2 PREFIX rdf:     http://www.w3.org/1999/02/22-rdf-syntax-ns#
3 PREFIX rdfs:    http://www.w3.org/2000/01/rdf-schema#
4 PREFIX qb: <http://purl.org/linked-data/cube#>
5 PREFIX qb4o: <http://purl.org/qb4olap/cubes#>
6
7 ## Time Dimension
8 sdw:Time rdf:type qb:DimensionProperty;
```

```
         rdfs:label "Time Dimension";
         qb4o:hasHierarcy sdw:TimeHierarchy.

# Dimension Hierarchies
sdw:TimeHierarchy rdf:type qb4o:Hierarchy;
                  rdfs:label "Time Hierarchy";
                  qb4o:inDimension sdw:Time;
                  qb4o:hasLevel  sdw:Day, sdw:Month, sdw:Year.



# Hierarchy levels
sdw:Day rdf:type qb4o:LevelProperty;
                 rdfs:label "Day Level";
                 qb4o:hasAttribute sdw:dayId, sdw:dayName.
sdw:Month rdf:type qb4o:LevelProperty;
                 rdfs:label "Month Level";
                 qb4o:hasAttribute sdw:monthId, sdw:monthName.
sdw:Year rdf:type qb4o:LevelProperty;
                 rdfs:label "Year Level";
                 qb4o:hasAttribute sdw:yearId, sdw:yearName.
sdw:All rdf:type qb4o:LevelProperty;
                 rdfs:label "ALL".

# Level attributes
sdw:dayId rdf:type qb4o:LevelAttribute;
                 rdfs:label "day ID";
                 qb4o:updateType qb4o:Type2;
                 rdfs:range xsd:String.
sdw:monthId rdf:type qb4o:LevelAttribute;
                 rdfs:label "Month ID";
                 qb4o:updateType qb4o:Type2;
                 rdfs:range xsd:String.
sdw:yearId rdf:type qb4o:LevelAttribute;
                 rdfs:label "year ID";
                 qb4o:updateType qb4o:Type2;
                 rdfs:range xsd:String.
sdw:dayName rdf:type qb4o:LevelAttribute;
                 rdfs:label "day Name";
                 qb4o:updateType qb4o:Type1;
                 rdfs:range xsd:String.
sdw:monthName rdf:type qb4o:LevelAttribute;
                 rdfs:label "Month Name";
                 qb4o:updateType qb4o:Type1;
                 rdfs:range xsd:String.
sdw:yearName rdf:type qb4o:LevelAttribute;
                 rdfs:label "year Name";
                 qb4o:updateType qb4o:Type1;
                 rdfs:range xsd:String.

#rollup relations
sdw:payMonth rdf:type qb4o:RollupProperty.
sdw:payYear rdf:type qb4o:RollupProperty.
sdw:payAll rdf:type qb4o:RollupProperty.

# Hierarchy Steps
_:ht1 rdf:type qb4o:HierarchyStep;
      qb4o:inHierarchy sdw:TimeHierarchy;
      qb4o:childLevel sdw:Day;
      qb4o:parentLevel sdw:Month;
      qb4o:pcCardinality qb4o:OneToMany;
      qb4o:rollup sdw:payMonth.


_:ht2 rdf:type qb4o:HierarchyStep;
      qb4o:inHierarchy sdw:TimeHierarchy;
      qb4o:childLevel sdw:Month;
      qb4o:parentLevel sdw:Year;
      qb4o:pcCardinality qb4o:OneToMany;
      qb4o:rollup sdw:payYear.

## Subsidy Cube
sdw:amounteuro rdf:type qb:MeasureProperty;
               rdfs:label "subsidy amount"; rdfs:range xsd:Double.
sdw:SubsidyStructure rdf:type qb:DataStructureDefinition;
                     qb:component[qb4o:level sdw:Recipient];
                     qb:component[qb4o:level sdw:Day];
                     qb:component[qb:measure sdw:amounteuro; qb4o:aggregateFunction qb4o:sum, qb4o:avg].

# Subsidy Dataset
sdw:Subsidy rdf:type qb:Dataset;
                        rdfs:label "Subsidy dataset";
                        qb:structure sdw:SubsidyStructure;
```

TBoxExtraction After defining a target TBox, the next step is to extract source TBoxes. Typically, in a semantic source, the TBox and ABox of the source are provided. Therefore, no external extraction task/operation is required. However, sometimes, the source contains only the ABox, and no TBox. In that scenario, an extraction process is required to derive a TBox from the ABox. We formally present the TBox extraction operation from a given ABox $ABox$ as $f_{ABox2TBox} : ABox \rightarrow TBox$. The derived $TBox$ is defined in terms of the following TBox constructs: A set of concepts C, a set of concept taxonomies H, a set of properties P, and the sets of property domains (D) and ranges (R). The following steps describe the process to derive each TBox element for $TBox$.

1. C: By checking the unique objects of the triples in *ABox* where `rdf:type` is used as a predicate, C is identified.

2. H: The taxonomies among concepts are identified by checking the instances they share among themselves. Let C_1 and C_2 be two concepts. One of the following taxonomic relationships holds between them: 1) if C_1 contains all instances of C_2, then we say C_2 is a subclass of C_1 (C_2 `rdfs:subClassOf` C_1); 2) if they do not share any instances, they are disjoint (C_1 `owl:disjointWith` C_2); and 3) if C_1 and C_2 are both a subclass of each other, then they are equivalent (C_1 `owl:equivalentClass` C_2).

3. P, D, R: By checking the unique predicates of the triples, P is derived. A property $p \in P$ can relate resources with either resources or literals. If the objects of the triples where p is used as predicates are IRIs, then p is an object property; the domain of p is the set of the types of the subjects of those triples, and the range of p is the types of the objects of those triples. If the objects of the triples where p is used as predicates are literals, then p is a datatype property; the domain of p is the set of types the subjects of those triples, and the range is the set of data types of the literals.

SourceToTargetMapping Once the target and source TBoxes are defined, the next task is to characterize the ETL flows at the Definition Layer by creating source-to-target mappings. Because of the heterogeneous nature of source data, mappings among sources and the target should be done at the TBox level. In principle, mappings are constructed between sources and the target; however, since mappings can get very complicated, we allow to create a sequence of *SourceToTargetMappings* whose subsequent input is generated by the precedent operation. The communication between these operations is by means of a materialized intermediate mapping definition and its meant to

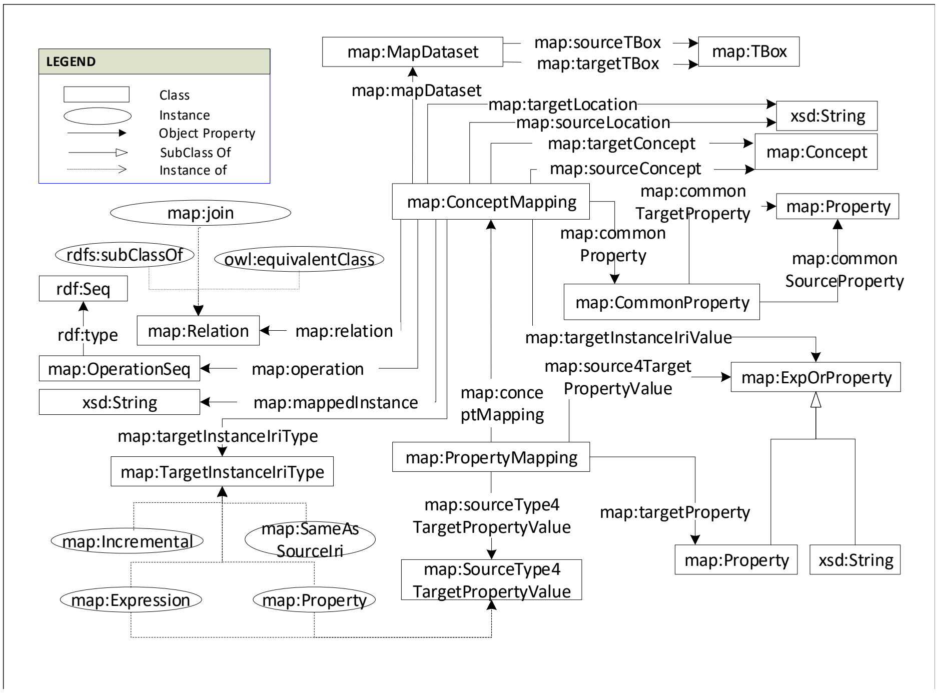


Fig. 9: Pictorial summary of the key terms and their relationship of the S2TMAP vocabulary. (Reproduced from [17])

facilitate the creation of very complex flows (i.e., mappings) between source and target.

A source-to-target mapping is constructed between a source and a target TBoxes, and it is composed of a set of concept-mappings. A concept-mapping defines 1) a relationship (equivalence, subsumption, supersumption, or join) between a source and the corresponding target concept, 2) which source instances are mapped (either all or a subset defined by a filter condition), 3) the rule to create the IRI for target concept instances, 4) the source and target ABox locations, 5) the common properties between two concepts if their relationship is join, 6) the sequence of ETL operations required to process the concept-mapping, and 7) a set of property-mappings for each property having the target concept as a domain. A property-mapping defines how a target property is mapped from either a source property or an expression over properties. The formal definition of a source-to-target mapping is given in [17]. To implement the source-to-target mappings, a Source-To-Target Mapping (S2TMAP) vocabulary is proposed. Figure 9 is the pictorial presentation of the key terms used in S2TMAP. Table 4 describes the key terms of S2TMAP. Listing 2 presents a snapshot of a mapping between a source and a target TBoxes.

Listing 2 An S2TMAP representation of a source-to-target mapping. (Adopted from [17])

```
PREFIX onto: <http://extbi.lab.aau.dk/ontology/>
PREFIX bus:  <http://extbi.lab.aau.dk/ontology/business/>
PREFIX sub:  <http://extbi.lab.aau.dk/ontology/subsidy/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX map:  <http://extbi.lab.aau.dk/ontology/s2tmap/>
PREFIX sdw:  <http://extbi.lab.aau.dk/sdw>
PREFIX : <http://extbi.lab.aau.dk/ontology/s2map/example#>

## MapDataset
:mapDataset1 rdf:type map:Dataset;
  rdfs:label "Map-dataset for subsidy and subsidyMD ontology";
  map:sourceTBox "/map/subsidyTBox.ttl";
  map:targetTBox "/map/subsidyMDTBox.ttl".

#concept-mapping: Populating the sdw:Recipient level
:Recipient_RecipientMD   rdf:type map:ConceptMapping;
  rdfs:label "Level member generation";
  map:mapDataset :mapDataset1;
  map:sourceConcept sub:Recipient;
  map:targetConcept sdw:Recipient;
  map:sourceLocation "/map/subsidy.nt";
  map:targetLocation "/map/sdw";
  map:relation owl:equivalentClass;
  map:mappedInstance "All";
  map:targetInstanceIRIValueType map:Property;
  map:targetInstanceIRIValue sub:recipientID;
  map:operation _:opSeq1.
_:opSeq1 rdf:type rdf:Seq;
  rdf:_1 map:LevelMemberGenerator;
  rdf:_2 map:Loader.

# property-mappings under :Recipient_RecipientMD
:companyId_company rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for companyId";
  map:conceptMapping :Recipient_RecipientMD;
  map:targetProperty sdw:hasCompany;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:companyId;
:cityId_city rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for cityId";
  map:conceptMapping :Recipient_RecipientMD;
  map:targetProperty sdw:inCity;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:city;
.........
..
```

4.4 The Execution Layer

In the Execution Layer, ETL flows are constructed to populate an MD SDW. Table 3 summarizes the set of ETL operations. Developers can use either of the following options: (i) The recommended option is that given a TBox construct, a mapping file and the IRI graph, the automatic ETL execution flow generation process (described in Section 4.5) will automatically extract the parameter values from the mapping file. (ii) They can manually set input parameters at the operation level. Here, we give a high-level overview of each operation category-wise.

Extraction

Extraction is the data retrieval process from sources. Here, we introduce two extraction operation for semantic sources: (i) *GraphExtractor* - to form/extract

[8]Key terms are the concepts and properties used in the vocabulary. Here, properties are presented with the italic form to distinguish them from the classes of the vocabulary.

Category	Key Term [8]	Description
Map-Dataset (For each mapping between a source and a target TBox, we create a map-dataset)	map:MapDataset	A source-to-target mapping is created as an instance of this class.
	map:sourceTBox, map:targetTBox	A map dataset is connected to its source and target TBoxes through the map:sourceTBox and map:targetTBox properties.
	map:sourceLocation, map:targetLocation	A map dataset points to the locations of the source and target ABoxes using the map:sourceLocation and map:targetLocation properties.
Concept-Mapping (For each mapping between a source and a target concept, we create a concept-mapping)	map:ConceptMapping	A concept-mapping between a source and target concept is initialized by this class.
	map:Concept	A map concept can be either an OWL concept or an MD level or a qb dataset.
	map:sourceConcept, map:targetConcept	The source and target concepts of a concept-mapping is connected through the map:sourceConcept and map:targetConcept properties.
	map:mapdataset	This property connects a concept-mapping to its map dataset.
	map:relation	Through it, a relationship between the source and target concept is defined. A relationship can be either rdfs:subClassOf, or rdfs:subPropertyOf or owl:equivalentClass.
	map:mappedInstance	It indicates whether all source instances or a subset of them are mapped to the target.
	map:targetInstanceIri Type	It indicates the rule to create target instance IRIs. It can be created either incrementally, or by using source IRI or by an expression or based on a source property value.
	map:targetInstanceIri Value	It defines the expression or the source property to be used to create target instance IRIs.
	map:commonProperty, map:sourceCommon Property, map:target CommonProperty	These properties are used to define the common properties between the source and target concepts of a concept-mapping.
	map:operation	Using this operation a concept-mapping indicates a sequence of ETL operations by which it will be processed.
Property-Mapping (For each property related to the target concept in a concept-mapping, we define a property-mapping associated with it)	map:PropertyMapping	It is used to define a property-mapping between a target property and a source property or an expression.
	map:Property	It is used to instantiate a property. The property can be either a data type property, or an object property, or a level attribute, or a roll-up relation, or even a level property.
	map:sourceType4Target PropertyValue	It defines the source type of a target property. It can either be a source property or an expression over the properties.
	map:source4Target PropertyValue	It defines either an expression or the source property to be used to create the target property-values.
	map:conceptMapping	A property-mapping is connect to a concept-mapping using this property.

Table 4: Description of the key terms of S2TMAP vocabulary.

an RDF graph from a semantic source, and (ii) *TBoxExtraction* - to derive a TBox from a semantic source described in Section 5. As such, this is the only operation of the Execution Layer generating metadata stored in the Mediatory Constructs.

GraphExtractor(Q, G, outputPattern, tABox) Since the data integration process proposed in this paper uses RDF as the canonical model, we extract/-generate RDF triples from the sources with this operation. GraphExtractor is functionally equivalent to SPARQL CONSTRUCT queries [37]. Given a query pattern *Q*, an RDF graph *G*, output triple Patterns *outputPattern*, and the output location *tABox*, *GraphExtractor* operation performs a pattern matching operation over the given source *G* (i.e., finds a map function binding the variables in query pattern Q to constants in G), and then, for each binding, it creates triples according to the triple templates in *outputPattern*. Finally, the operation stores the output in the path *tABox*.

Transformation

Transformation operations transform the extracted data according to the semantics of the SDW. Here, we define the following semantic-aware ETL transformation operations.

TransformationOnLiteral(sConstruct, tConstruct, sTBox, sABox propertyMappings, tABox) As described in the *SourceToTargetMapping* task, a target property (in a property-mapping) of a construct (i.e., a level, a QB dataset, or a concept) can be mapped to either a source concept property or an expression over the source properties. An expression allows arithmetic operations, datatype (string, number, and date) conversion and processing functions, and group functions (sum, avg, max, min, count) as defined in SPARQL [24]. Given a source and target TBox construct *sConstruct* and *tConstruct*, a source TBox and ABox *tABox* and *sABox*, a set of property-mappings *propertyMappings*, and the output location *tABox*, this operation transforms (or directly returns) the *sABox* triple objects according to the expressions (defined through `map:source4TargetPropertyValue`) in *propertyMappings* and stores the triples in *tABox*. This operation first creates a SPARQL SELECT query based on the expressions defined in *propertyMappings*, and then, on top of the SELECT query, it forms a SPARQL CONSTRUCT query to generate the transformed ABox for *tConstruct*.

JoinTransformation(sConstruct, tConstruct, sTBox, tTBox, sABox, tABox, comProperty, propertyMappings) A TBox construct (a concept, a level, or a QB dataset) can be populated from multiple sources. Therefore, an operation is necessary to join and transform data coming from different sources. Two constructs of the same or different sources can only be joined if they share some common properties. Given two constructs *sConstruct* and *tConstruct*, their TBoxes *sTBox* and *tTBox*, their instances *sABox* and *tABox*, a set of common properties *comProperty*, and their set of property-mappings *propertyMappings*, this operation joins two constructs based on *comProperty*, transforms their data based on the expressions (specified through `map:source4TargetPropertyValue`) defined in *propertyMappings*, and updates *tABox* based on the join result. It creates a SPARQL SELECT query joining two constructs using either `AND` or `OPT` features, and on top of that query, it forms a SPARQL CONSTRUCT query to generate the transformed *tABox*.

LevelMemberGenerator(sConstruct, level, sTBox, sABox, tTBox, iriValue, iriGraph, propertyMappings, tABox) In QB4OLAP, dimensional data are physically stored in levels. A level member, in an SDW, is described by a unique IRI and its semantically linked properties (i.e., level attributes and rollup properties). This operation generates data for a dimension schema. Given a source construct *sConstruct*, a target level[9] *level*, the source TBox and ABox *sTBox* and *sABox*, the SDW's TBox *tTBox*, a rule[10] to create IRIs for its level members *iriValue*, the IRI graph[11] within which to look up IRIs, *iriGraph*, a set of property-mappings *propertyMappings*, and the target ABox location *tABox*, *LevelMemberGenerator* operation generates QB4OLAP-compliant triples for the level members of *level* based on the semantics encoded in *tTBox* and stores them in *tABox*.

ObservationGenerator(dataset, tTBox, sABox, iriValue, iriGraph, propertyMappings, tABox) In QB4OLAP, an observation represents a fact. A fact is uniquely identified by an IRI, which is defined by a combination of several members from different levels and contains values for different measure properties. This operation generates data for a cube schema defined. Given a target dataset *dataset*, the SDW TBox *tTBox*, its source instances *sABox*, a rule to create IRIs for observations *iriValue*, the IRI graph *iriGraph*, its property-mappings set *propertyMappings*, and the target ABox location *tABox*, *ObservationGenerator* generates QB4OLAP-compliant triples for observations of *dataset* based on the semantics encoded in *tTBox* and stores them in *tABox*.

ChangedDataCapture(nABox, oABox, flag) In a real-world scenario changes occur in a semantic source both at the schema and instance level. Therefore, an SDW needs to take action based on the changed schema and instances. The adaption of the SDW TBox with the changes of source schemas is an analytical task and requires the involvement of domain experts, therefore, it is out of the scope of this thesis. Here, only the changes at the instance level are considered. From the given inputs, *ChangedDataCapture* outputs either 1) a set of new instances (in the case of SDW evolution, i.e., *flag=0*) or 2) a set of updated triples —the existing triples changed over time— (in the case of SDW update, i.e., *flag=1*) and overwrites *oABox*. This is done by means of the set difference operation. This operation must then be connected to either *LevelMemberGenerator* to create the new level members or *updateLevel* (described below) to reflect the changes in the existing level members.

[9]A level is termed as a level property in QB4OLAP, therefore, we use the term "level" and "level property" interchangeably.

[10]A rule can be either a source property, an expression or incremental, as described in Section 4.3.

[11]The IRI graph is an RDF graph that keeps a triple for each resource in the SDW with their corresponding source IRI.

UpdateLevel(level, updatedTriples, sABox tTBox, tABox, propertyMappings, iriGraph) Based on the triples updated in the source ABox *sABox* for the level *level* (generated by *ChangedDataCapture*), this operation updates the target ABox *tABox* to reflect the changes in the SDW according to the semantics encoded in the target TBox *tTBox* and *level* property-mappings *propertyMappings*. Here, we address three update types (Type1-update, Type2-update, and Type3-update), defined by Ralph Kimball in [34] for a traditional DW, in an SDW environment. The update types are defined in *tTBox* for each level attribute of *level* (as discussed in Section 4.3). As we consider only instance level updates, only the objects of the source updated triples are updated. To reflect a source updated triple in *level*, the level member using the triple to describe itself, will be updated. In short, the level member is updated in the following three ways: 1) A Type1-update simply overwrites the old object with the current object. 2) A Type2-update creates a new version for the level member (i.e., it keeps the previous version and creates a new updated one). It adds the validity interval for both versions. Further, if the level member is a member of an upper level in the hierarchy of a dimension, the changes are propagated downward in the hierarchy, too. 3) A Type3-update overwrites the old object with the new one. Besides, it adds an additional triple for each changed object to keep the old object. The subject of the additional triple is the instance IRI, the object of the triple is the old object, and the predicate is *concat(oldPredicate, "oldValue")*.

Listing 3 describes how different types of updates work by considering two members of the `sdw:Recipient` level (lines 1-10). Suppose the address of the second member (lines 7-10) is changed to "Torninglundvej 23" from "Torninglundvej 9". A Type1-update simply overwrites it (line 22). A Type2-update creates a new version (lines 50-56). Both old and new versions contain validity information (lines 46-48 and lines 54-56). A Type3-update overwrites the old address (line 34) and adds a new triple to record the old address (line 35).

Listing 3 An example of different types of updates.

```
 1 sdw:Recipient Level
 2
 3 sdw:Recipient#76291 rdf:type qb4o:LevelMember;
 4                     qb4o:member sdw:Recipient;
 5                     sdw:name "R. Nielsen";
 6                     sdw:add "Lyngbyvej 369".
 7 sdw:Recipient#86646 rdf:type qb4o:LevelMember;
 8                     qb4o:member sdw:Recipient;
 9                     sdw:name "A. Jochimsen";
10                     sdw:add "Torninglundvej 9".
11
12 After type1 update
13
14 sdw:Recipient#76291 rdf:type qb4o:LevelMember;
15                     qb4o:member sdw:Recipient;
16                     sdw:name "R. Nielsen";
17                     sdw:add "Lyngbyvej 369".
18
19 sdw:Recipient#86646 rdf:type qb4o:LevelMember;
20                     qb4o:member sdw:Recipient;
21                     sdw:name "A. Jochimsen";
22                     sdw:add "Torninglundvej 23".
```

```

After type3 update

sdw:Recipient#76291 rdf:type qb4o:LevelMember;
                    qb4o:member sdw:Recipient;
                    sdw:name "R. Nielsen";
                    sdw:add "Lyngbyvej 369".

sdw:Recipient#86646 rdf:type qb4o:LevelMember;
                    qb4o:member sdw:Recipient;
                    sdw:name "A. Jochimsen";
                    sdw:add "Torninglundvej 23";
                    sdw:add_oldValue "Torninglundvej 9".
After type2 update
sdw:Recipient#76291 rdf:type qb4o:LevelMember;
                    qb4o:member sdw:Recipient;
                    sdw:name "R. Nielsen";
                    sdw:add "Lyngbyvej 369".

sdw:Recipient#86646 rdf:type qb4o:LevelMember;
                    qb4o:member sdw:Recipient;
                    sdw:name "A. Jochimsen";
                    sdw:add "Torninglundvej 9";
                    sdw:fromDate ‘‘0000-00-00";
                    sdw:toDate ‘‘2016-09-25";
                    sdw:status ‘‘Expired".

sdw:Recipient#86646_2016_09_26 rdf:type qb4o:LevelMember;
                               qb4o:member sdw:Recipient;
                               sdw:name "A. Jochimsen";
                               sdw:add "Torninglundvej 23";
                               sdw:fromDate ‘‘2016-09-26";
                               sdw:toDate ‘‘9999-12-31";
                               sdw:status ‘‘Current".
```

MaterializeInference(ABox, TBox) This operation infers new information that has not been explicitly stated in an SDW. It analzses the semantics encoded into the SDW and enriches the SDW with the inferred triples. A subset of the OWL 2 RL/RDF rules, which encodes part of the RDF-Based Semantics of OWL 2 [48], are considered here. The reasoning rules can be applied over the TBox *TBox* and ABox *ABox* separately, and then together. Finally, the resulting inference is asserted in the form of triples, in the same spirit as how SPARQL deals with inference.

ExternalLinking (intResource, externalDataSource) This operation links the internal resource *intResource* with the resources from external data source *externalDataSource*. This operation applies the process described in Section 4.2.

Load

Loader(tripleSet, tsPath) An SDW is represented in the form of RDF triples and the triples are stored in a triple store (e.g., Jena TDB). Given a set of RDF triples *tripleSet* and the path of a triple store *tsPath*, this operation loads *triplesSet* in the triple store.

4.5 Automatic ETL Execution Flow Generation

Since ETL flows are characterized at the Definition Layer by means of the source-to-target mappings file, the ETL flows in the Execution Layer can be generated automatically. This way, we can guarantee the created flows are sound and free the user from creating the ETL flows. Note that this is possible because the Mediatory Constructs contain all the required metadata to automate the process.

Algorithm 2: CreateETL (Reproduced from [17])

```
Input: TargetConstruct $x$, MappingFile $G$, IRIGraph $G_{IRI}$
Output: A set of ETL flows $E$
begin
1   $node_{target} \leftarrow$ find$(x, G)$
2   $?c_{maps} \leftarrow$findConMap$((node_{target}, G)$
    /* In $G$ : $node_{target} \xleftarrow{\texttt{map:targetConcept}} ?c_{maps}$ ; $|?c_{maps}| \geq 0$ */
3   $E \leftarrow$ createSet()
4   if $(?c_{maps} \neq \emptyset)$ then
5       foreach $c \in ?c_{maps}$ do
6           $s_c \leftarrow$ createStack()
7           $s_c \leftarrow$ CreateAFlow$(c, s_c, G, G_{IRI})$
8           $E$.add$(s_c)$
9   return $E$
```

Algorithm 2 shows the steps required to create ETL data flows to populate a target construct (a level, a concept, or a dataset). As inputs, the algorithm takes a target construct x, the mapping file G, and the IRI graph G_{IRI}, and it outputs a set of ETL data flows E. At first, it locates x in G (line 1) and gets the concept-mappings where $node_{target}$ participates (line 2). As a (final) target construct can be populated from multiple sources, it can be connected to multiple concept-mappings, and for each concept-mapping, it creates an ETL data flow by calling the function[12] CreateAFlow (lines 5-8). Algorithm 3 generates an ETL data flow for a concept-mapping c and recursively does so if the current concept-mapping source element is connected to another concept-mapping as, until it reaches a source element. Algorithm 3 recursively calls itself and uses a stack to preserve the order of the partial ETL data flows created for each concept-mapping. Eventually, the stack contains the whole ETL data flow between the source and target schema.

Algorithm 3 works as follows. The sequence of operations in c is pushed to the stack after parameterizing it (lines 1-2). Algorithm 4 parameterizes each operation in the sequence, as described in Section 6 and returns a stack of parameterized operations. As inputs, it takes the operation sequence, the concept-mapping, the mapping file, and the IRI graph. For each operation,

[12]Here, functions used in the algorithms are characterized by a pattern over G, shown at its side as comments in the corresponding algorithm.

Algorithm 3: CreateAFlow (Reproduced from [17])

```
Input: ConceptMapping $c$, Stack $s_c$, MappingFile $G$, IRIGraph $G_{IRI}$
Output: Stack $s_c$
begin
1   $ops \leftarrow$ findOperations$(c, G)$
    /* In $G$ : $c \xrightarrow{\texttt{map:operation}} ops$ */
2   $s_c$. push(Parameterize$(ops, c, G, G_{IRI})$)
    /* Push parameterized operations in $s_c$ */
3   $scon \leftarrow$ findSourceConcept$(c, G)$
    /* In $G$ : $c \xrightarrow{\texttt{map:sourceConcept}} scon$ */
4   $?scon_{map} \leftarrow$ findConMap$(scon, G)$
    /* In $G$ : $scon \xleftarrow{\texttt{map:targetConcept}} ?scon_{map}$ ; $|?scon_{map}| \leq 1$ */
5   if $(|?scon_{map}| = 1)$ then
6       CreateAFlow$(nc \in ?scon_{map}, s_c, G)$
        /* recursive call with $nc$ */
7   $s_c$.push(StartOp)
    /* Push the ETL start operation to $s_c$ */
8   return $s_c$
```

it uses the parameterizeOp(op, G, loc(cm), G_{IRI}) function to automatically parameterize op from G (as all required parameters are available in G) and push the parameterized operation in a stack (line 2-7). Note that, for the first operation in the sequence, the algorithm uses the source ABox location of the concept-mapping (line 4) as an input, whereas for the remaining operations, it uses the output of the previous operation as input ABox (line 6). Finally, Algorithm 4 returns the stack of parameterized operations.

Then, Algorithm 3 traverses to the adjacent concept-mapping of c connected via c's source concept (line 3-4). After that, the algorithm recursively calls itself for the adjacent concept-mapping (line 6). Note that, here, we set a restriction: except for the final target constructs, all the intermediate source and target concepts can be connected to at most one concept-mapping. This constraint is guaranteed when building the metadata in the Definition Layer. Once there are no more intermediate concept-mappings, the algorithm pushes a dummy starting ETL operation (*StartOp*) (line 7) to the stack and returns it. *StartOp* can be considered as the root of the ETL data flows that starts the execution process. The stacks generated for each concept-mapping of $node_{target}$ are added to the output set E (line 8 in Algorithm 2).

Algorithm 4: PARAMETERIZE (REPRODUCED FROM [17])

```
Input: Seq $ops$, ConceptMapping $cm$, MappingFile $G$, IRIGraph $G_{IRI}$
Output: Stack, $s_{op}$
begin
1   $s_{op} \leftarrow$ createStack()
2   for ($i = 1$ to length($ops$)) do
3       if ($i = 1$) then
4           parameterizeOp($op[i], G$, loc($cm$), $G_{IRI}$)
5           $s_{op}$.push($op[i]$)
6       parameterizeOp($op[i], G$, outputPath($op[i-1]$), $G_{IRI}$)
7       $s_{op}$.push($op[i]$)
8   return $s_{op}$
```

4.6 Discussion

A GUI-based prototype, named $SETL_{CONSTRUCT}$ is developed by implementing the approach presented in this chapter. Like other traditional tools (e.g., PDI), $SETL_{CONSTRUCT}$ allows users to create ETL flows by dragging, dropping, and connecting the ETL operations. Therefore, developers do not require a programming background. Furthermore, it provides an interface to create automatic ETL flows discussed in Section 4.5 (we call it $SETL_{AUTO}$). Both systems are compared with the previous programmable framework SETL. To evaluate the completeness (with regard to real use cases) and correctness of the ETL constructs, an MD SDW is created with the use case described in Section 3.2. Table 5 summarizes the effort required by the developer to create different ETL tasks using SETL, $SETL_{CONSTRUCT}$, and $SETL_{AUTO}$. The developer effort is measured in terms of the Number of Typed Characters (NOTC) for SETL and in the Number of the Used Concepts[13] (NOUC) for SETL and $SETL_{AUTO}$. The evaluation shows that $SETL_{CONSTRUCT}$ improves considerably over SETL in terms of productivity, development time, and performance. The evaluation shows that 1) $SETL_{CONSTRUCT}$ uses 92% fewer NOTC than SETL, and $SETL_{AUTO}$ further reduces NOUC by another 25%; 2) using $SETL_{CONSTRUCT}$, the development time is almost cut in half compared to SETL, and is cut by another 27% using $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ is scalable and has similar performance compared to SETL [17]. Readers are directed to [17] for a detailed explanation of this evaluation.

In summary, this chapter proposes a set of high-level ETL constructs to

[13]A `concept` in $SETL_{CONSTRUCT}$ is a pop-up window used to define a basic element of an ETL construct (e.g., a concept-mapping, property-mapping, map-dataset, cube structure, dimension, dimension, an ETL operation, etc.)

Tool ⇒		SETL				$SETL_{CONSTRUCT}$				$SETL_{AUTO}$			
						NOUC	Components of each Concept			NOUC	Components of each concept		
ETL Task ⇓	**Sub Construct**	**Procedures/ Data Structures**	**NUEP**	**NOTC**	**Task/Operation**		**Clicks**	**Selections**	**NOTC**		**Clicks**	**Selections**	**NOTC**
TBox with MD Semantics	Cube Structure	Concept(), Property(), BlankNode(), Namespaces(), conceptPrope-rtyBindings(), createOntology()	1	665	TargetTBox-Definition	1	9	9	7	1	9	9	7
	Cube Dataset		1	67		1	2	2	18	1	2	2	18
	Dimension		2	245		2	3	4	12	2	3	4	12
	Hierarchy		7	303		7	4	6	18	7	4	6	18
	Level		16	219		16	4	5	17	16	4	5	17
	Level Attribute		38	228		38	4	3	15	38	4	3	15
	Rollup Property		12	83		12	2	1	7	12	2	1	7
	Measure Property		1	82		1	4	3	25	1	4	3	25
	Hierarchy Step		12	315		12	6	6	6	12	6	6	6
	Prefix		21	74		21	1	0	48	21	1	0	48
Total (for the Target TBox)			**1**	**24509**		**101**	**382**	**342**	**1905**	**101**	**382**	**342**	**1905**
Mapping Generation	Mapping Dataset	N/A	0	0	SourceTo-Target-Mapping	5	1	2	6	3	1	2	6
	Concept-mapping	dict()	17	243		23	9	10	0	18	12	10	15
	Property-mapping	N/A	0	0		59	2	2	0	44	2	2	0
Total (for the Mapping File)			**1**	**2052**		**87**	**330**	**358**	**30**	**65**	**253**	**274**	**473**
Semantic Data Extraction from an RDF Dump file	N/A	query()	17	40	GraphExtractor	17	5	0	30	1	Auto Generation		
Semantic Data Extraction through a SPARQL endpoint	N/A	ExtractTriples-FromEndpoint()	0	100	GraphExtractor	0	5	0	30	0	Auto Generation		
Cleansing	N/A	Built-in Petl functions	15	80	Transformation-OnLiteral	5	12	1	0	1	Auto Generation		
Join	N/A	Built-in Petl functions	1	112	Join-Transformation	1	15	1	0	1	Auto Generation		
Level Member Generation	N/A	createDataTripleToFile(), createDataTripleTo-TripleStore()	16	75	LevelMember-Generator	16	6	6	0	1	Auto Generation		
Observation Generation	N/A	createDataTripleToFile(), createDataTripleTo-TripleStore()	1	75	Observation-Generator	1	6	6	0	1	Auto Generation		
Loading as RDF Dump	N/A	insertTriplesIntoTDB(), bulkLoadToTDB()	0	113	Loader	0	1	2	0	0	Auto Generation		
Loading to a triple store	N/A	insertTriplesIntoTDB(), bulkLoadToTDB()	17	153	Loader	17	1	2	0	1	Auto Generation		
Auto ETL Generation	N/A	N/A	N/A	N/A	CreateETL	1	N/A	N/A	N/A	1	21	16	0
Total (for the Execution Layer Tasks)			**1**	**2807**		**57**	**279**	**142**	**363**	**58**	**21**	**16**	**0**
Total (for the whole ETL Process)				**29358**		**245**	**991**	**826**	**2298**	**224**	**656**	**632**	**2378**

Table 5: Comparison among the productivity of SETL, $SETL_{CONSTRUCT}$, and $SETL_{AUTO}$ for the SETLSDW. (Reproduced from [17])

integrate semantic data sources. The overall integration process uses RDF as the canonical model, and it is divided into the Definition and Execution Layer. In the Definition Layer, developers create the metadata (target and source TBoxes, and source-to-target mappings), while in the Execution Layer, the data flows are generated. We propose a set of high-level ETL operations for semantic data that can be used to create ETL flows in the Execution Layer. As we characterize the transformations in the Definition Layer in terms of source-to-target mappings at the schema level, we are able to propose an automatic algorithm to generate ETL data flows in the Execution Layer.

Further, the developed prototype, $SETL_{CONSTRUCT}$, facilitates users to accomplish integration tasks by using GUIs. Therefore, no programming knowledge is required. However, $SETL_{CONSTRUCT}$ only considers semantic data as source data. As explained in our overall vision presented in Chapter 3, it should consider non-semantic data as well. Besides, an ontology showing the connections among the ETL constructs of different layers would be advisable. On top of that, an OLAP tool connecting automatically to SDWs and enabling traditional OLAP is required to lower the entry barrier for OLAP users, not proficient in SW formalisms, such as RDF and SPARQL. Indeed, the current prototype presented in this chapter requires being able to trigger SPARQL queries with OLAP-like semantics to perform OLAP to the SDW. In Section 5, a semantic ETL and Business Intelligence tool, named $SETL_{BI}$, is proposed and developed to address the limitations just discussed.

5 A Semantic BI Tool and Its Application

This section is a summary of Paper C [16].

5.1 Motivation and Problem Statement

The popularity of the SW and LD encourages organizations to organize and publish data using the RDF model. This growth poses new requirements to Business Intelligence (BI) tools to integrate those data into an MD DW and enable On-Line Analytical Processing (OLAP)-like analysis over RDF data [16]. Although [42] outlines a semi-automatic method to incorporate RDF data into a traditional RDBMS-centric MD DW, the generated DW no longer maintains the SW data principles. Therefore, native OLAP-style analysis over an RDF dataset remains unaddressed. In [11, 28, 55], the authors have proposed approaches to enable OLAP analysis over RDF data. However, either they do not allow users to create their own OLAP queries or require end-users to be familiar with QL [10] or complex SPARQL queries.

Existing research focuses on either one or more parts of the aimed problem (i.e., semantic data integration and enabling OLAP analysis on the integrated data), but there is no single solution that supports all the steps to integrate heterogeneous data semantically in a DW and enabling OLAP queries on it. In this chapter, a GUI-based semantic BI tool named Semantic ETL and Business Intelligence ($SETL_{BI}$) is presented, which allows users to define, map, process, integrate, and query data semantically. $SETL_{BI}$ extends $SETL_{CONSTRUCT}$ by 1) adding operations to process non-semantic data sources in the ETL Layer, 2) adding an OLAP Layer that allows users to create OLAP queries using a GUI, without requiring SPARQL knowledge, 3) providing an ontology, which shows the compatibility among the constructs of different layers, and 4) evaluating and demonstrating the system with a new use case (Bangladesh population and housing census). The following sections present the architecture of $SETL_{BI}$, a use case description, demonstration, and an overall discussion.

5.2 Architecture of the System

Figure 10 shows that $SETL_{BI}$ is organized into three layers: the Definition Layer, ETL Layer, and OLAP Layer. First two layers, described in Sections 3 and 4, are used to semantically integrate data from heterogeneous sources and map to the Definition and Execution Layers presented in chapter 4. The orange-colored constructs are the new ETL constructs introduced over Paper B. The OLAP Layer enables the self-service analysis over the produced

MD SDW. The connections among the ETL constructs (tasks/operations) of inter- and intra-layers are shown, in Figure 10, by using an ontology, where a class represents an ETL construct (a task/operation) of a layer and an arrow represents the relationship between two constructs of layers. The following paragraphs describe the layers in detail.

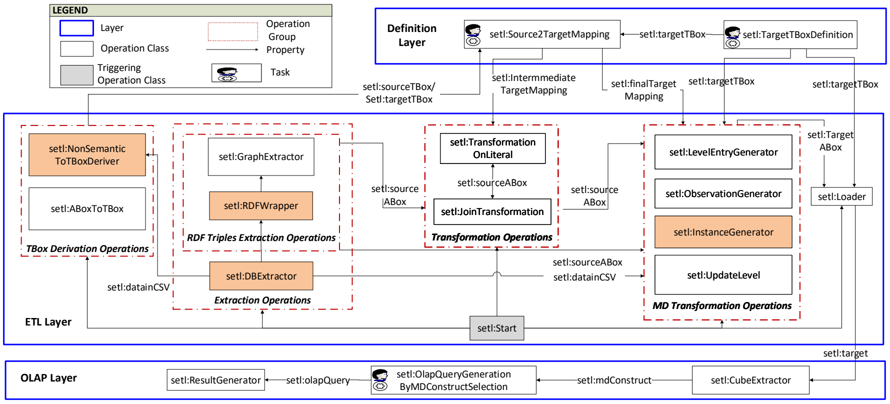


Fig. 10: The ontology shows the constructs of each layer and their connections and depicts the overall workflow in each layer. Only operations having a direct incoming arrow from `setl:Start` can be a part of an ETL flow. (Adopted from [16])

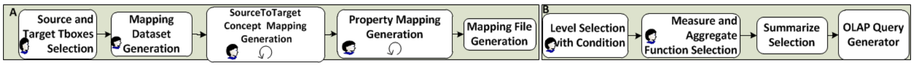

A
Source and Target Tboxes Selection
Mapping Dataset Generation
SourceToTarget Concept Mapping Generation
Property Mapping Generation
Mapping File Generation
B
Level Selection with Condition
Measure and Aggregate Function Selection
Summarize Selection
OLAP Query Generator

Fig. 11: The workflows for Source2Target Mapping(A) and the OLAP Query Generation (B) operation. The rectangles, user icons and curved arrows represent automatic, user-required and iterative tasks. (Reproduced from [16])

The Definition Layer is composed of two tasks: `setl:TargetTBoxDefinition` and `setl:SourceToTargetMapping`. The first task supports designing the TBox of an SDW with MD semantics, and users can define/edit new/existing data structures, cubes, cuboids, levels, level attributes, roll-up properties, measures using a GUI. Internally, the task translates the user's input into equivalent QB, QB4OLAP, and RDFS/OWL statements and produces an RDF file. The second task supports creating mappings across the constructs of (intermediate) source and target by providing a GUI. The overall workflow to create a source-to-target mapping is illustrated in Figure 11. At first, a user selects the source and target TBoxes to be mapped, then create a mapping dataset. Under a mapping dataset, the user can create one or more concept-mappings, and each concept mapping contains one or more property-mappings. Internally, the task interprets the user's input with our own OWL-based Source-to-Target Mapping (S2TMAP) vocabulary and produces a mapping file in RDF.

The ETL Layer includes a set of ETL operations. Based on their functionality, we divide them into five groups: *TBox Derivation Operations*, *Extraction Operations*, *Transformation Operations*, *MD Transformation Operations*, and *Load operations* [16]. There are two operations in the *TBox Derivation Operations* group: 1) `setl:NonSemanticToTBoxDeriver`: It allows to derive TBoxes from non-semantic (CSV, XML, JSON and Database) sources. This ETL construct is missing in Paper B and introduced here to support non-semantic sources. 2) `setl:ABoxToTBoxDeriver`: It derives TBox from an RDF file containing only assertions. In the *Extraction Operations* group, `setl:GraphExtractor` extracts RDF triples from a semantic source based on a query. It is equivalent to *GraphExtraction* operation in Paper B. To support non-semantic source, this group introduces two additional ETL operation of the set of ETL constructs proposed in Paper B: `setl:RDFWrapper` and `setl:DBExtractor`. The `setl:RDFWrapper` operation converts the data of a non-semantic (CSV, XML, JSON, Database) into equivalent RDF triples. As `setl:RDFWrapper` needs RML definitions, we give a GUI in the Definition Layer to define RML definitions[14]. `setl:DBExtractor` extracts data from a RDBMS Database source based on a query and outputs in CSV Format. The *Transformation Operations* group supports group functions (sum, avg, count) as well as string, numeric, and date based transformation on RDF property-values according to the intermediate mappings using `setl:TransformationOnLiteral` and joins between two RDF sources using `setl:JoinTransformation`. In the *MD Transformation Operations* group, `setl:LevelMemberGenerator`, `setl:ObservationGenerator`, and `setl:InstanceGenerator` create level members, observations and instances respectively to generate the ABox of a SDW according to the semantics encoded in the TBox. The `setl:InstanceGenerator` operation is also an extension over Paper B. This operation is required to create the instances of an OWL concept. Each operation in the *MD Transformation Operations* group supports both RDF and CSV input. `setl:UpdateLevel` updates the target level members to reflect the changes in a source to the target. In the *Load operations* group, `setl:Loader` loads RDF data into either a local RDF file or Jena TDB triplestore. The GUI of the ETL Layer allows users to drag and drop operations to create ETL flows.

Given either a local RDF file or a SPARQL endpoint containing an SDW, the OLAP Layer allows users to create OLAP queries using a GUI. This layer shows the cubes contained in the SDW. By selecting a cube, users can retrieve the cube structure composed of dimensions, hierarchies, levels, measures and aggregate functions. Then, users create and issue OLAP queries to explore and aggregate measures at various level of details by selecting different cube components. Figure 11 (B) illustrates how to create an OLAP query. Other OLAP operations, such as slice and dice queries are generated by adding

[14]This version only allows R2RML

conditions on the selected levels. Internally, the system translates the OLAP query generated from the selections into an equivalent SPARQL query [20]. Therefore, no SPARQL or QL knowledge is necessary.

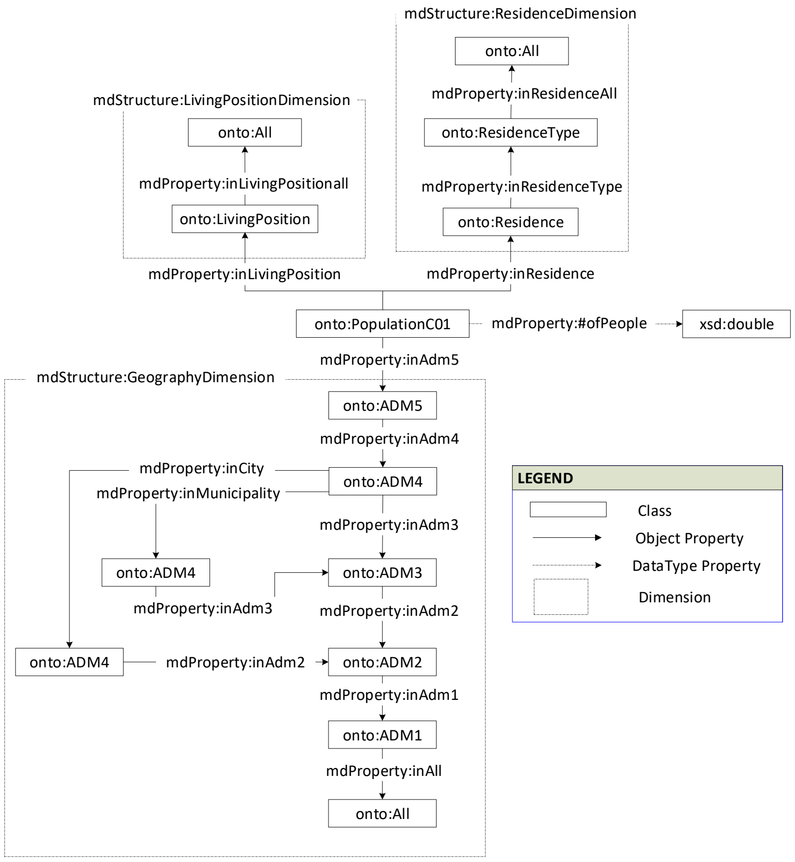


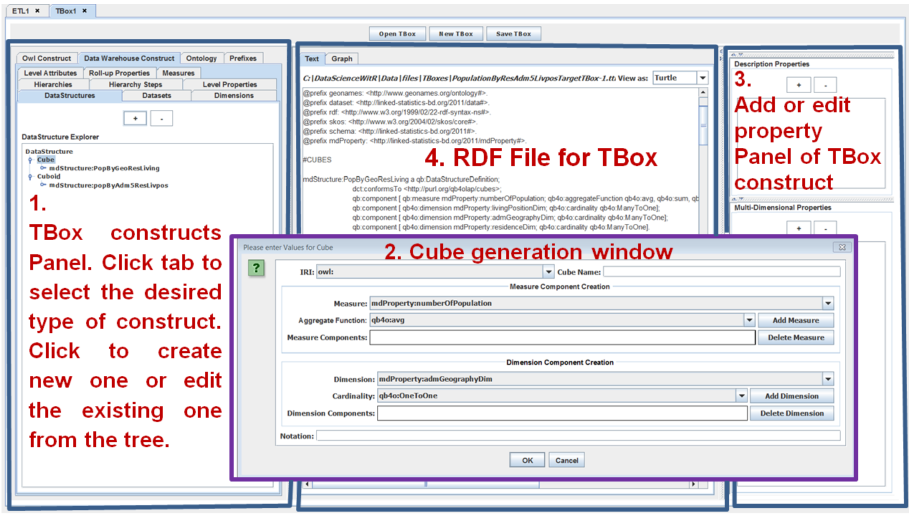
Fig. 12: The ontology of the Bangladesh census 2011 dataset C01.

5.3 A Use Case Description

This section describes the source datasets and the target TBox for integrating and understanding the encoded knowledge of those datasets. We consider the population and housing Census 2011 datasets from Bangladesh Bureau of Statistics (BBS). The datasets are available in `http://203.112.218.65:8008/Census.aspx?MenuKey=89`. In these datasets, the entire population and housing stock of Bangladesh are recorded based on characteristics such as geographic, demographic, social and economic, household, and family status. There are 15 datasets (CO1 to C015) given in tabular format. The objective is to publish those census data in an OLAP-compliant LOD format to enable decision making. From the datasets, we create 50 levels and 18 dimensions and a cube from each dataset. All cubes are independent, however they share some dimensions among them. Figure 12 represents the schema of the dataset C01, where the population is counted based on geography, living position, and residence dimensions. In Figure 12, the concept `onto:PopulationC01` represents the factual concept of the dataset and `mdProperty:#ofPeople` is the measure. The dataset contains three dimensions (shown by a box with dotted line): `mdStructure:GeographyDimension`, `mdStructure:ResidenceDimension`, and `mdStructure:LivingPositionDimension`. Each level of the dimensions is represented by a concept and the hierarchies of the each dimensions are also shown.

5.4 Demonstration

This section demonstrates how an ETL developer uses $SETL_{BI}$ to create an SDW for the use case. As the data are given in PDF format, the first task is to convert them into CSV format before using the system. Then, (s)he performs the following steps in order.


ETL1
TBox1
Open TBox
New TBox
Save TBox
Owl Construct
Data Warehouse Construct
Ontology
Prefixes
Level Attributes
Roll-up Properties
Measures
Hierarchies
Hierarchy Steps
Level Properties
DataStructures
Datasets
Dimensions
+
-
DataStructure Explorer
DataStructure
Cube
mdStructure:PopByGeoResLiving
Cuboid
mdStructure:popByAdm5ResLivpos
1. TBox constructs Panel. Click tab to select the desired type of construct. Click to create new one or edit the existing one from the tree.
Text
Graph
4. RDF File for TBox
Description Properties
3. Add or edit property Panel of TBox construct
Multi-Dimensional Properties
2. Cube generation window
Please enter Values for Cube
?
Measure Component Creation
Add Measure
Delete Measure
Dimension Component Creation
Add Dimension
Delete Dimension
OK
Cancel

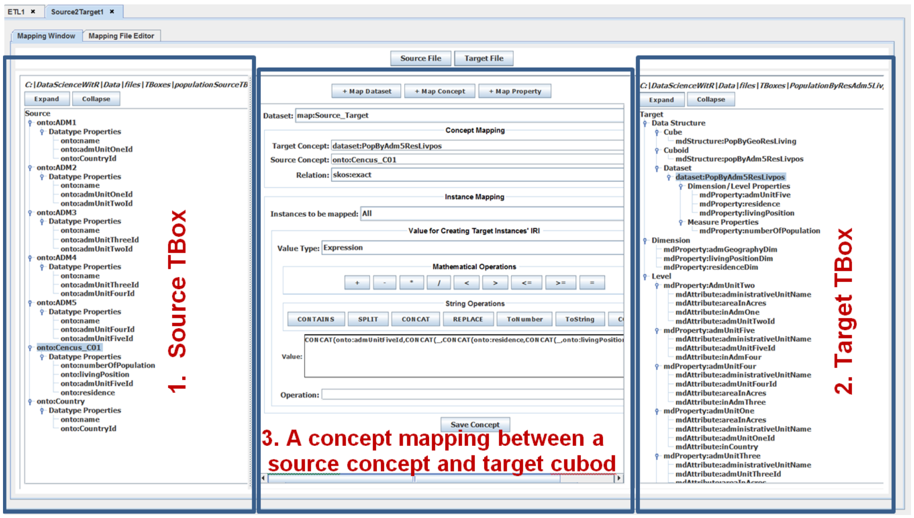
C:|DataScienceWitR|Data|files|TBoxes|PopulationByResAdm5LivposTargetTBox-1.tt View as: Turtle

```
@prefix geonames: <http://www.geonames.org/ontology#>.
@prefix dataset: <http://linked-statistics-bd.org/2011/data#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.
@prefix schema: <http://linked-statistics-bd.org/2011#>.
@prefix mdProperty: <http://linked-statistics-bd.org/2011/mdProperty#>.

#CUBES

mdStructure:PopByGeoResLiving a qb:DataStructureDefinition;
        dct:conformsTo <http://purl.org/qb4olap/cubes>;
        qb:component [ qb:measure mdProperty:numberOfPopulation; qb4o:aggregateFunction qb4o:avg, qb4o:sum, qb
        qb:component [ qb4o:dimension mdProperty:livingPositionDim; qb4o:cardinality qb4o:ManyToOne];
        qb:component [ qb4o:dimension mdProperty:admGeographyDim; qb4o:cardinality qb4o:ManyToOne];
        qb:component [ qb4o:dimension mdProperty:residenceDim; qb4o:cardinality qb4o:ManyToOne].
```

IRI: owl:
Cube Name:
Measure: mdProperty:numberOfPopulation
Aggregate Function: qb4o:avg
Measure Components:
Dimension: mdProperty:admGeographyDim
Cardinality: qb4o:OneToOne
Dimension Components:
Notation:

Fig. 13: The interface to define Target TBox. (Reproduced from [16])

The Definition Layer At first, the developer creates the target TBox using the GUI shown in Figure 13. The rightmost panel is used to create TBox constructs (e.g., cubes, dimensions, levels, hierarchies, OWL classes, properties, etc.). A pop-up window to create a cube is shown in panel 2. (S)he can edit any existing constructs using panel 3. Panel 4 shows the RDF triples generated internally from the user inputs. The next task is to derive the TBox from a CSV file using `setl:NonSemanticToTBoxDeriver`. Once source and target TBoxes are defined, then, (s)he needs to create a source-to-target mapping. At first, (s)he has to create a map-dataset; then under the map-dataset, the designer can create a concept-mapping using the GUI shown in Figure 14. The leftmost panel shows the source TBox and the rightmost one shows the target TBox. The middle panel is used to define the different constructs of a concept-property mapping. Once the button **+Map Property** is clicked the property-mapping panel will be appeared.



```
C:|DataScienceWitR|Data|files|TBoxes|populationSourceTB
Source
  onto:ADM1
    Datatype Properties
      onto:name
      onto:admUnitOneId
      onto:CountryId
  onto:ADM2
    Datatype Properties
      onto:name
      onto:admUnitOneId
      onto:admUnitTwoId
  onto:ADM3
    Datatype Properties
      onto:name
      onto:admUnitThreeId
      onto:admUnitTwoId
  onto:ADM4
    Datatype Properties
      onto:name
      onto:admUnitThreeId
      onto:admUnitFourId
  onto:ADM5
    Datatype Properties
      onto:name
      onto:admUnitFourId
      onto:admUnitFiveId
  onto:Cencus_C01
    Datatype Properties
      onto:numberOfPopulation
      onto:livingPosition
      onto:admUnitFiveId
      onto:residence
  onto:Country
    Datatype Properties
      onto:name
      onto:CountryId
```

Dataset: map:Source_Target

Concept Mapping
Target Concept: dataset:PopByAdm5ResLivpos
Source Concept: onto:Cencus_C01
Relation: skos:exact

Instance Mapping
Instances to be mapped: All

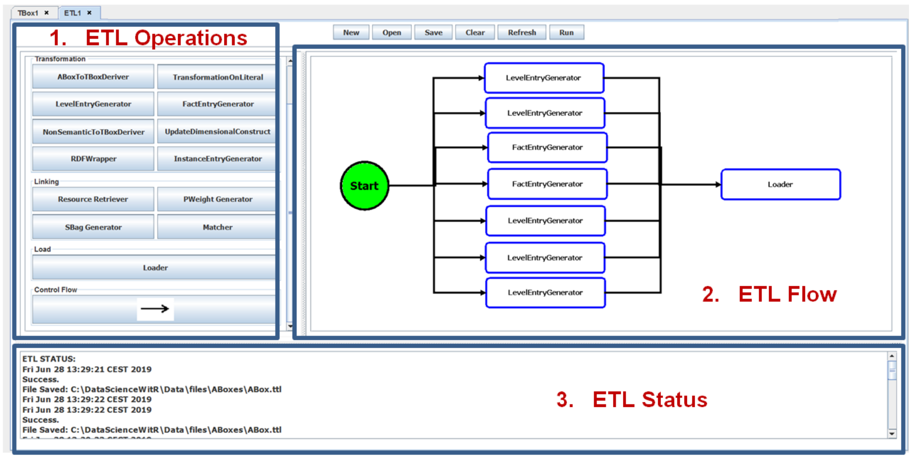
Value for Creating Target Instances' IRI
Value Type: Expression
Mathematical Operations
String Operations
Value: CONCAT(onto:admUnitFiveId,CONCAT(_,CONCAT(onto:residence,CONCAT(_,onto:livingPositio
Operation:

```
C:|DataScienceWitR|Data|files|TBoxes|PopulationByResAdm5Liv
Target
  Data Structure
    Cube
      mdStructure:PopByGeoResLiving
    Cuboid
      mdStructure:popByAdm5ResLivpos
    Dataset
      dataset:PopByAdm5ResLivpos
        Dimension/Level Properties
          mdProperty:admUnitFive
          mdProperty:residence
          mdProperty:livingPosition
        Measure Properties
          mdProperty:numberOfPopulation
  Dimension
    mdProperty:admGeographyDim
    mdProperty:livingPositionDim
    mdProperty:residenceDim
  Level
    mdProperty:AdmUnitTwo
      mdAttribute:administrativeUnitName
      mdAttribute:areaInAcres
      mdAttribute:inAdmOne
      mdAttribute:admUnitTwoId
    mdProperty:admUnitFive
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitFiveId
      mdAttribute:inAdmFour
    mdProperty:admUnitFour
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitFourId
      mdAttribute:areaInAcres
      mdAttribute:inAdmThree
    mdProperty:admUnitOne
      mdAttribute:areaInAcres
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitOneId
      mdAttribute:inCountry
    mdProperty:admUnitThree
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitThreeId
```

Fig. 14: The interface to define mappings. (Reproduced from [16])

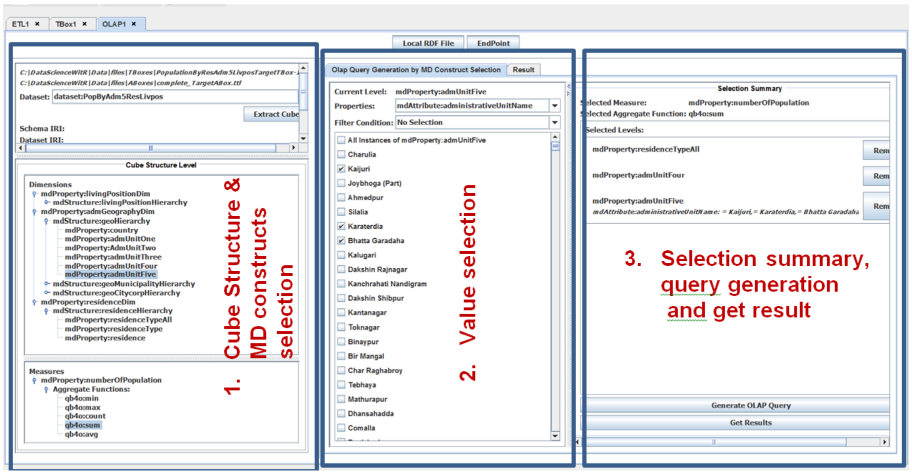
The ETL Layer The next task is to create the ETL data flows. The GUI of the ETL Layer is shown in Figure 15. The leftmost panel shows different ETL operations that can be dragged and dropped to panel 2 to create ETL data flows. After executing the flows, the status of the ETL flows will be shown in panel 3. Potentially, the developer may decide to automatically generate the ETL flows from the metadata created in the Definition Layer as explained in Section 4.5.

The OLAP Layer Once the SDW is created, the next task is to issue the OLAP queries using the GUI of the OLAP Layer, shown in Figure 16. Panel 1 shows the cube structure in terms of dimensions, hierarchies, levels and measures with the associated aggregate functions. The developer can create queries by clicking different levels of hierarchies and measures. (S)he can



```
ETL STATUS:
Fri Jun 28 13:29:21 CEST 2019
Success.
File Saved: C:\DataScienceWitR\Data\files\ABoxes\ABox.ttl
Fri Jun 28 13:29:22 CEST 2019
Fri Jun 28 13:29:22 CEST 2019
Success.
File Saved: C:\DataScienceWitR\Data\files\ABoxes\ABox.ttl
```

Fig. 15: The interface to create ETL flows. (Reproduced from [16])


ETL1
TBox1
OLAP1
Local RDF File
EndPoint
Extract Cube
Cube Structure Level
Olap Query Generation by MD Construct Selection
Result
Selection Summary
Rem
Generate OLAP Query
Get Results
1. Cube Structure & MD constructs selection
2. Value selection
3. Selection summary, query generation and get result

C:\DataScienceWitR\Data\files\TBoxes\PopulationByResAdm5LivposTargetTBox
C:\DataScienceWitR\Data\files\ABoxes\complete_TargetABox.ttl
Dataset: dataset:PopByAdm5ResLivpos
Schema IRI:
Dataset IRI:

```
Dimensions
  mdProperty:livingPositionDim
    mdStructure:livingPositionHierarchy
  mdProperty:admGeographyDim
    mdStructure:geoHierarchy
      mdProperty:country
      mdProperty:admUnitOne
      mdProperty:AdmUnitTwo
      mdProperty:admUnitThree
      mdProperty:admUnitFour
      mdProperty:admUnitFive
    mdStructure:geoMunicipalityHierarchy
    mdStructure:geoCitycorpHierarchy
  mdProperty:residenceDim
    mdStructure:residenceHierarchy
      mdProperty:residenceTypeAll
      mdProperty:residenceType
      mdProperty:residence
Measures
  mdProperty:numberOfPopulation
    Aggregate Functions:
      qb4o:min
      qb4o:max
      qb4o:count
      qb4o:sum
      qb4o:avg
```

Current Level: mdProperty:admUnitFive
Properties: mdAttribute:administrativeUnitName
Filter Condition: No Selection

- [ ] All Instances of mdProperty:admUnitFive
- [ ] Charulia
- [x] Kaijuri
- [ ] Joybhoga (Part)
- [ ] Ahmedpur
- [ ] Silalia
- [x] Karaterdia
- [x] Bhatta Garadaha
- [ ] Kalugari
- [ ] Dakshin Rajnagar
- [ ] Kanchrahati Nandigram
- [ ] Dakshin Shibpur
- [ ] Kantanagar
- [ ] Toknagar
- [ ] Binaypur
- [ ] Bir Mangal
- [ ] Char Raghabroy
- [ ] Tebhaya
- [ ] Mathurapur
- [ ] Dhansahadda
- [ ] Comalla

Selected Measure: mdProperty:numberOfPopulation
Selected Aggregate Function: qb4o:sum
Selected Levels:
mdProperty:residenceTypeAll
mdProperty:admUnitFour
mdProperty:admUnitFive
mdAttribute:administrativeUnitName: = Kaijuri, = Karaterdia, = Bhatta Garadaha

Fig. 16: The interface to create and run OLAP queries. (Reproduced from [16])

filters the selection based on different conditions. The middle panel enables it. The summary of the selections will be shown in panel 3. Then, (s)he can run the query and get result of the query.

5.5 Prototype Specifications

The system is developed in Java 8. All GUIs are implemented in Standard Widget Toolkit (SWT). Jena 3.4.0 is used to process, store, and query RDF and as a triple store, Jena TDB is used. A comprehensive video explain-ing the functionalities of $SETL_{BI}$ is available at `http://extbi.cs.aau.dk/SETLBI/index.php` and `https://www.youtube.com/watch?v=9-a4MVHqZow&feature=emb_logo`. The source codes, examples, and developer manual are available at `https://github.com/bi-setl/SETL`.

5.6 Discussion

This chapter presented an end-to-end semantic BI tool, $SETL_{BI}$. $SETL_{BI}$ enriches the SETL framework described in Section 3 with the ETL constructs described in Section 4. In addition, it introduces new operations to the set of ETL operations described in Section 4.4 to enable non-semantic sources in the integration process. On top of that, it enables native OLAP-like analysis over an SDW. To show the relevance of $SETL_{BI}$ in real-world cases, we have used it to create an MD SDW for the Bangladesh population and housing census 2011 data. It has been shown how users can use the different layers of $SETL_{BI}$ to accomplish their integration tasks. In summary, it fulfills all the aspects of an end-to-end semantic ETL framework, as discussed in Section 1.3. $SETL_{BI}$ bridges the two lines of research discussed in Section 2: It facilitates (1) DW designers with little/no SW knowledge to semantically integrate semantic and/or non-semantic data and analyze it in OLAP style, and (2) SW users with basic MD background to define MD views over semantic data for enabling OLAP-like analysis [16].

6 Summary of Contributions

The overall objective of the thesis was to propose and develop a semantic ETL framework covering the nine aspects pointed out in Section 1.2. This thesis fulfills the objective incrementally. It proposes: (1) a framework to integrate semantic and non-semantic data into SDWs, (2) a set of basic high-level ETL constructs for processing semantic data and an approach to automate the execution flows, and (3) develops a GUI-based BI tool that allows integrating semantic and non-semantic data into an SDW and enables native OLAP-queries on SDWs. The main body of the thesis is composed of four papers. Paper A is published as an extension of Paper D; therefore, to present the contributions of each paper coherently, Paper D is listed first. A summary of the contributions of each paper is given below.

- Paper D [15] proposes SETL, a Python-based programmable semantic ETL framework. The framework supports the inclusion of semantic (RDF) data into DWs in addition to relational data. It allows users to define an ontology for a DW. Note that this version of SETL does not allow users to create a DW with multidimensional semantics. It provides a semantic transformation algorithm to generate semantic data from the source data according to the semantics encoded in the ontology. It also provides a method to connect internal DW data with external knowledge bases. Hence, it creates a knowledge base composed of an ontology and its instances where data are semantically connected with other external/internal data. A comprehensive experimental evaluation comparing SETL to a solution using traditional tools on the use case of Danish Agricultural and Business datasets shows that SETL provides better performance, programmer productivity, and knowledge base quality [15].

- Paper A [39] significantly extends Paper D [15]. Here, we redefine the architecture of SETL by 1) dividing it into three layers: the Definition Layer, ETL Layer, and Data Warehouse Layer, 2) adding a subcomponent *QB4OLAP* for defining a target TBox (schema) with multidimensional (MD) semantics, 3) adding a semantic layer on top of non-semantic sources, and 4) adding a provenance graph to capture provenance information of IRIs. The design and implementation of the framework are presented separately. The design section includes the formal description of the components of the architecture. The version of the semantic transformation algorithm in Paper D is updated to reflect the MD semantics encoded in the target TBox at the data level. The developed Python-based programmable framework helps developers by providing a higher-level abstraction that lowers the entry barriers. Thus, one of the contributions is to automatically map the high-level abstraction to executable code [39]. For evaluating SETL, an SDW is created with MD semantics by integrating the Danish Business dataset and an EU subsidy dataset using SETL and compare the process with other tools/solutions. The evaluation shows that SETL represents a considerable improvement over the competing solutions/tools in terms of programmer productivity, knowledge base quality, and performance. SETL sets the foundations for the rest of the papers.

- Paper B [17] outlines an integration process, focusing on the usage of RDF as the canonical model, to integrate data from semantic data sources into an SDW. The overall integration process is divided into two layers: the Definition Layer and the Execution Layer. This paper proposes a set of high-level ETL constructs (tasks/operations) for each layer to process semantic data. The Definition Layer includes two tasks and one ABox-to-TBox derivation operation. The two tasks allow ETL designers to define target schemas and mappings between semantic data sources and the target schemas, whereas the ABox-to-TBox operation derives a TBox from a given ABox. A mapping vocabulary called S2TMAP is proposed here to enable semantic annotations of the mappings. In the Execution Layer, a set of high-level ETL operations are proposed to process semantic data sources. In addition to the cleansing, join, group, and data type based transformations for semantic data, operations are proposed to create multidimensional semantics at the data level and to update the SDW to reflect the changes of source data. Here, we show how different types of updates [35] used in traditional DW can be applied to an SDW. Different from SETL and other ETL tools, this layer proposes a new paradigm: the ETL flow transformations are characterized at the Definition Layer instead of independently within each ETL operation (in the Execution Layer). Thus, we are able to propose an algorithm to automatically generate ETL data flows in the Execution Layer. Finally, a GUI-based prototype, named $SETL_{CONSTRUCT}$ is developed by implementing all the ETL constructs proposed here. We provide an extensive evaluation to compare the productivity, quality, and performance of the proposed ETL operations with the previous framework SETL [39]. The evaluation shows that $SETL_{CONSTRUCT}$ improves considerably over SETL in terms of productivity, development time, and performance. The evaluation shows that 1) $SETL_{CONSTRUCT}$ uses 92% fewer NOTC than SETL, and $SETL_{AUTO}$ (the extension of $SETL_{CONSTRUCT}$ that enables automatic ETL execution flow generation) further reduces NOUC by another 25%; 2) using $SETL_{CONSTRUCT}$, the development time is almost cut in half compared to SETL, and is cut by another 27% using $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ is scalable and has similar performance compared to SETL [17].

- Paper C [16] presents the $SETL_{BI}$ (Semantic Extract-Transform-Load and Business Intelligence) integration platform that (1) covers all phases of integration: target definition, source-to-target mappings, semantic and non-semantic source extraction, data transformation, target population, and update and (2) enables OLAP-like analyses over an SDW. $SETL_{BI}$ is organized into three layers: the Definition Layer, ETL Layer, and OLAP Layer. Each layer is composed of a set of tasks/operations. The intra- and inter-layer connections among the constructs of layers are presented using an ontology. In addition to the ETL operations proposed in the Execution Layer of Paper B, the ETL Layer includes operations to process non-semantic data sources. The OLAP Layer provides a GUI to enable self-service OLAP analysis over an SDW. We use the Bangladesh 2011 population census data as a use case to demonstrate $SETL_{BI}$. Using $SETL_{BI}$, we generate an SDW with census data that can be analyzed using OLAP queries.

The problem that motivated the thesis was twofold: 1) the integration of semantic and non-semantic data into an MD DW semantically, and 2) enabling OLAP analysis over SW data. The solutions to this problem will fuse the benefits of BI and SW technologies; thus, both BI and SW communities will benefit. We set a hypothesis for the thesis *"In the context of highly heterogeneous data integration processes, considering semantics as first-class citizen facilitates the integration process by providing automation and lower entry barriers for non-technical users."* In this direction, as an output, Paper A provides a Python-based programmable framework SETL that provides a number of powerful modules, classes, and methods. SETL allows ETL developers to integrate semantic and non-semantic data programmatically. Paper B provides a GUI-based prototype $SETL_{CONSTRUCT}$, which allows non-technical users to integrate semantic data in an SDW where the RDF model is used as the canonical model to make semantics as first-class citizen. Importantly, $SETL_{CONSTRUCT}$ allows automatic ETL data flows generation. Paper C provides the final contribution of this thesis: the BI tool $SETL_{BI}$. $SETL_{BI}$ facilitates (1) DW designers with little/no SW knowledge to semantically integrate semantic and/or non-semantic data and analyze it in OLAP style, and (2) SW users with basic MD background to define MD views over semantic data for enabling OLAP-like analysis [16]. Additionally, SW users can enrich the generated SDW's schema with RDFS/OWL constructs. Using the developed tool we generated an MD SDW using real-world data. Therefore, we empirically prove the hypothesis of this thesis by creating a system that fulfills all identified aspects of semantic ETL.

7 Future Work Directions

Due to the popularity and high acceptance rate of SW and LD, nowadays, technologies are shifting from a database centric concept to a knowledge base centric concept, where semantic data play an important role. This project will be a guideline for accelerating this shifting process. Taking the proposed framework SETL as a foundation, researchers can aim to develop further interactive and automatic integration frameworks for the Big Data and Data Lake environments. This project bridges the traditional BI technologies and SW technologies, which in turn will open the door for further research opportunities like developing machine-understandable ETL and warehousing techniques. In the following, we give specific pointers to future work.

SETL uses a demand-driven approach to design an (MD) SDW, i.e., based on user requirements, a DW designer designs the TBox of an SDW. The framework can be extended to follow a source-driven approach where different source schemas are aligned and merged to produce a single source of truth (the target TBox). Different ontology matching and merging techniques can be utilized to (semi)automatically create the target TBox.

$SETL_{CONSTRUCT}$ allows users to create source-to-target mappings manually. However, an algorithm can be developed to create the mappings (semi-) automatically. Another extension of this work is to define the formal semantics of the operations using an ontology and mathematically prove the correctness and completeness of the proposed ETL constructs.

To enable the utmost benefit of Exploratory OLAP, the system should explore SW search engines, data hubs, and knowledge bases and dynamically suggest relevant data sources to enrich MD constructs and evolve the target schema.

We use the Jena TDB triplestore to load an SDW. Other triple stores and/or graph databases can be explored to get better loading and query performance. Also, exploring the physical optimization of named graphs can be an interesting extension.

The source-to-target mappings act as a mediator to generate data according to the semantics encoded in the target. Therefore, the next extension of this work is to extend the framework from purely physical to also virtual data integration where instead of materializing all source data in the DW, ETL processes will run based on users' demands. When considering virtual data integration, it is important to develop query optimization techniques for OLAP queries on virtual semantic DWs, similar to the ones developed for virtual integration of data cubes and XML data [45, 46, 59].

The next task also includes publishing the Bangladesh census and economic data in OLAP-complaint LOD format by linking it with other knowledge base resources to show the advantages of SDW over the existing solutions.

Another interesting work will be to apply this layer-based integration process in the Big Data and Data Lake environment. The metadata produced in the Definition Layer can be used as the basis to develop advanced catalog features for Data Lakes.

References

[1] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *IEEE transactions on knowledge and data engineering*, vol. 27, no. 2, pp. 571–588, 2014.

[2] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Joint International Semantic Technology Conference*. Springer, 2014, pp. 178–186.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A Nucleus for a Web of Open Data," in *The semantic web*. Springer, 2007, pp. 722–735.

[4] S. K. Bansal, "Towards a Semantic Extract-Transform-Load (ETL) Framework for Big Data Integration," in *Big Data*, 2014, pp. 522–529.

[5] L. Bellatreche, S. Khouri, and N. Berkani, “Semantic Data Warehouse Design: From ETL to Eeployment A La Carte,” in *International Conference on Database Systems for Advanced Applications*. Springer, 2013, pp. 64–83.

[6] R. Berlanga, O. Romero, A. Simitsis, V. Nebot, T. B. Pedersen, A. Abelló, and M. J. Aramburu, “Semantic Web Technologies for Business Intelligence,” in *Business Intelligence Applications and the Web: Models, Systems and Technologies*. IGI global, 2012, pp. 310–339.

[7] M. H. Bhuiyan, A. Bhattacharjee, and R. P. D. Nath, “DB2KB: A Framework to Publish a Database as a Knowledge Base,” in *2017 20th International Conference of Computer and Information Technology (ICCIT)*. IEEE, 2017, pp. 1–7.

[8] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1247–1250.

[9] M. Casters, R. Bouman, and J. Van Dongen, *Pentaho Kettle solutions: Building Open Source ETL Solutions with Pentaho Data Integration*. John Wiley & Sons, 2010.

[10] C. Ciferri, R. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi, “Cube Algebra: A Generic User-centric Model and Query Language for OLAP Cubes,” *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 9, no. 2, pp. 39–65, 2013.

[11] D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatiş, “RDF Aanalytics: Lenses over Semantic Graphs,” in *WWW*. ACM, 2014, pp. 467–478.

[12] P. Cudré-Mauroux, “Leveraging Knowledge Graphs for Big Data Integration: the XI Pipeline,” *Semantic Web*, no. Preprint, pp. 1–5.

[13] “Danish Central Company Registry (CVR) Data, http://cvr.dk/.”

[14] R. Cyganiak, D. Reynolds, and J. Tennison, “The RDF Data Cube Vocabulary,” *World Wide Web Consortium*, 2014.

[15] R. P. Deb Nath, K. Hose, and T. B. Pedersen, “Towards A Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses,” in *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*, 2015, pp. 15–24.

[16] R. P. Deb Nath, K. Hose, T. B. Pedersen, O. Romero, and A. Bhattacharjee, “SETLBI: An Integrated Platform for Semantic Business Intelligence,” in *Companion Proceedings of the Web Conference 2020*, 2020, pp. 167–171.

[17] R. P. Deb Nath, O. Romero, T. B. Pedersen, and K. Hose, “High-Level ETL for Semantic Data Warehouses,” in *Submitted to Semantic Web Journal*, 2020.

[18] L. Etcheverry, S. S. Gomez, and A. Vaisman, “Modeling and Querying Data Cubes on the Semantic Web,” *arXiv preprint arXiv:1512.06080*, 2015.

[19] L. Etcheverry, A. Vaisman, and E. Zimányi, “Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP,” in *DaWak*, 2014, pp. 45–56.

[20] L. Etcheverry and A. A. Vaisman, “Efficient Analytical Queries on Semantic Web Data Cubes,” *Journal on Data Semantics*, vol. 6, no. 4, pp. 199–219, 2017.

[21] "Ministry of Food, Agriculture and Fisheries of Denmark, http://en.fvm.dk/."

[22] L. Galárraga, K. A. M. Mathiassen, and K. Hose, "QBOAirbase: The European Air Quality Database as an RDF Cube," in *International Semantic Web Conference (Posters, Demos & Industry Tracks)*, 2017.

[23] M. Golfarelli and S. Rizzi, *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill, Inc., 2009.

[24] S. Harris, A. Seaborne, and E. Prud'hommeaux, "SPARQL 1.1 Query Language," *W3C recommendation*, vol. 21, no. 10, p. 778, 2013.

[25] A. Harth, K. Hose, and R. Schenkel, *Linked Data Management*. CRC Press, 2014.

[26] O. Hartig, K. Hose, and J. F. Sequeda, "Linked Data Management." 2019.

[27] T. Heath and C. Bizer, "Linked Data: Evolving the Web into a Global Data Space," *Synthesis lectures on the semantic web: theory and technology*, vol. 1, no. 1, pp. 1–136, 2011.

[28] M. Hilal, C. G. Schuetz, and M. Schrefl, "An OLAP Endpoint for RDF Data Analysis Using Analysis Graphs." in *ISWC*, 2017.

[29] K. Hose and R. Schenkel, "RDF Stores," in *Encyclopedia of Database Systems*. Springer Publishing Company, 2017.

[30] 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*. Springer, 2014, pp. 114–132.

[31] C. S. Jensen, T. B. Pedersen, and C. Thomsen, "Multidimensional Databases and Data Warehousing," *Synthesis Lectures on Data Management*, vol. 2, no. 1, pp. 1–111, 2010.

[32] E. Kalampokis, B. Roberts, A. Karamanou, E. Tambouris, and K. A. Tarabanis, "Challenges on Developing Tools for Exploiting Linked Open Data Cubes." in *SemStats@ ISWC*, 2015.

[33] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *Extended Semantic Web Conference*. Springer, 2012, pp. 87–101.

[34] R. Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, Inc., 1996.

[35] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, and B. Becker, *The Data Warehouse Lifecycle Toolkit*. John Wiley & Sons, 2008.

[36] T. Knap, P. Hanečák, J. Klímek, C. Mader, M. Nečaskỳ, B. Van Nuffelen, and P. Škoda, "UnifiedViews: An ETL Tool for RDF Data Management," *Semantic Web*, vol. 9, no. 5, pp. 661–676, 2018.

[37] E. V. Kostylev, J. L. Reutter, and M. Ugarte, "CONSTRUCT Queries in SPARQL," in *18th International Conference on Database Theory (ICDT 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[38] J. Li, J. Tang, Y. Li, and Q. Luo, "Rimom: A Dynamic multistrategy ontology Alignment framework," *IEEE Transactions on Knowledge and data Engineering*, vol. 21, no. 8, pp. 1218–1232, 2008.

[39] R. P. D. Nath, K. Hose, T. B. Pedersen, and O. Romero, "SETL: A Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses," *Information Systems*, vol. 68, pp. 17–43, 2017.

[40] R. P. D. Nath, H. Seddiqui, and M. Aono, "Resolving Scalability Issue to Ontology Instance Matching in Semantic Web," in *2012 15th International Conference on Computer and Information Technology (ICCIT)*. IEEE, 2012, pp. 396–404.

[41] R. P. D. Nath, M. H. Seddiqui, and M. Aono, "An Efficient and Scalable Approach for Ontology Instance Matching," *JCP*, vol. 9, no. 8, pp. 1755–1768, 2014.

[42] V. Nebot and R. Berlanga, "Building Data Warehouses with Semantic Web Data," *Decision Support Systems*, vol. 52, no. 4, pp. 853–868, 2012.

[43] V. Nebot, R. Berlanga, J. M. Pérez, M. J. Aramburu, and T. B. Pedersen, "Multidimensional Integrated Ontologies: A Framework for Designing Semantic Data Warehouses," in *Journal on Data Semantics XIII*. Springer, 2009, pp. 1–36.

[44] "OWL, Web Ontology Language, www.w3.org/TR/owl-ref/."

[45] D. Pedersen, J. Pedersen, and T. B. Pedersen, "Integrating XML Data in the TARGIT OLAP System," in *Proceedings. 20th International Conference on Data Engineering*. IEEE, 2004, pp. 778–781.

[46] D. Pedersen, K. Riis, and T. B. Pedersen, "Query Optimization for OLAP-XML Federations," in *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, 2002, pp. 57–64.

[47] I. Petrou and G. Papastefanatos, "Publishing Greek Census Data as linked open data," *ERCIM News*, vol. 96, 2014.

[48] A. Polleres, A. Hogan, R. Delbru, and J. Umbrich, "RDFS and OWL Reasoning for Linked Data," in *Reasoning Web International Summer School*. Springer, 2013, pp. 91–149.

[49] "R2RML: RDB to RDF Mapping Language https://www.w3.org/TR/2012/REC-r2rml-20120927/."

[50] M. H. Seddiqui, S. Das, I. Ahmed, R. P. D. Nath, and M. Aono, "Augmentation of Ontology Instance Matching by Automatic Weight Generation," in *2011 World Congress on Information and Communication Technologies*. IEEE, 2011, pp. 1390–1395.

[51] D. Skoutas and A. Simitsis, "Ontology-Based Conceptual Design of ETL Processes for both Structured and Semi-structured Data," *IJSWIS*, vol. 3, no. 4, pp. 1–24, 2007.

[52] M. Thenmozhi and K. Vivekanandan, "An Ontological Approach to Handle Multidimensional Schema Evolution for Data Warehouse," *International Journal of Database Management Systems*, vol. 6, no. 3, p. 33, 2014.

[53] C. Thomsen and T. Bach Pedersen, "pygrametl: A Powerful Programming Framework for Extract-Transform-Load Programmers," in *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*, 2009, pp. 49–56.

[54] J. Varga, "Semantic Metadata for Supporting Exploratory OLAP," *PhD Thesis, Universitat Politècnica de Catalunya*, 2017.

[55] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, and C. Thomsen, “QB2OLAP: Enabling OLAP on Statistical Linked Open Data,” in *ICDE*. IEEE, 2016, pp. 1346–1349.

[56] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen, “Dimensional Enrichment of Statistical Linked Open Data,” *Journal of Web Semantics*, vol. 40, pp. 22–51, 2016.

[57] “W3C. Resource Description Framework, http://www.w3.org/RDF/.”

[58] “RDFS. Resource Description Framework Schema, http://www.w3.org/TR/rdf-schema/.”

[59] X. Yin and T. B. Pedersen, “Evaluating XML-Extended OLAP Queries Based on Physical Algebra,” *Journal of Database Management (JDM)*, vol. 17, no. 2, pp. 85–116, 2006.

Part II

Papers

Paper A

SETL: A Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses

Rudra Pratap Deb Nath, Katja Hose, Torben Bach Pedersen, and Oscar Romero

The paper has been published in the
Information Systems
Vol. 64, pp. 17–43, 2017. DOI: 10.1016/j.is.2017.01.005

Abstract

In order to create better decisions for business analytics, organizations increasingly use external structured, semi-structured, and unstructured data in addition to the (mostly structured) internal data. Current Extract-Transform-Load (ETL) tools are not suitable for this "open world scenario" because they do not consider semantic issues in the integration processing. Current ETL tools neither support processing semantic data nor create a semantic Data Warehouse (DW), a repository of semantically integrated data. This paper describes our programmable Semantic ETL (SETL) framework. SETL builds on Semantic Web (SW) standards and tools and supports developers by offering a number of powerful modules, classes, and methods for (dimensional and semantic) DW constructs and tasks. Thus it supports semantic data sources in addition to traditional data sources, semantic integration, and creating or publishing a semantic (multidimensional) DW in terms of a knowledge base. A comprehensive experimental evaluation comparing SETL to a solution made with traditional tools (requiring much more hand-coding) on a concrete use case, shows that SETL provides better programmer productivity, knowledge base quality, and performance.

1 Introduction

Business Intelligence (BI) tools support intelligent business decisions by analyzing available organizational data. Data Warehouses (DWs) are used to store large data volumes from different operational databases in enterprises, and On-Line Analytical Processing (OLAP) queries are applied on DWs to answer business analytical questions. Extract-Transform-Load (ETL) is the backbone process of a DW, and the ETL design and deployment takes up to 80% of the time in DW projects [49]. To support data analyses or OLAP queries on it, the underlying schema of a DW is represented using the Multidimensional (MD) model. In the MD model, data are categorized as either facts with associated numerical measures or dimensions that characterize the facts [28]. This model represents any interesting observation of the domain (i.e., measures) in its context (i.e., dimensions) [1].

Nowadays, the Web is also an important source of information. Moreover, Semantic Web (SW) technologies and the Linked Data (LD) principles inspire organizations to publish and share their data using the Resource Description Framework (RDF) [61]. The role of data semantics in such sources is defined by different facets: using Internationalized Resource Identifiers (IRIs) to uniquely identify resources globally, providing common terminology, semantically linking published information, and providing further knowledge to allow reasoning [9]. The number of semantic data sources is ever increasing over time, and the growth rate of the data in the SW is faster than the computational power of computers [32]. As a result, besides analyzing internal data available in a DW, it is essential to incorporate external data from various (semantic) sources into the DW to derive the needed business knowledge. For example, companies want to include product reviews, customer complains, competitors' status from the Web in their analytical process besides the internal sales and customer data [1].

The inclusion of external data, especially RDF data, however, raises several challenges for integration and transformation in comparison to the traditional ETL process. One of the drawbacks of using RDF data in the corporate analysis process is that data sometimes do not have any schema (e.g., only instances are delivered with implicit schema) [12], have a poor schema where not all the schema constructs are explicitly defined (e.g., only taxonomies are depicted), or complex schema (e.g., other more expressive ontological languages, such as OWL, are used). Unlike the relational data model, the RDF data model does not impose a common schema for all instances. In contrast, it provides flexible means to tackle different schema information as well as its evolution. This flexible nature of the RDF data model makes it suitable for representing and exchanging data in the SW. However, different sources may describe the same data in different ways, introducing semantic heterogeneity problems. Therefore, to build a successful DW system with heterogeneous data, the integration process should be able to deal with data semantics as a first-class citizen. Traditional ETL tools are unable to process such external data because they (1) do not support semantic data sources, i.e., they are not prepared to deal with semantic heterogeneity, (2) are entirely schema-dependent, and (3) do not focus on meaningful semantic relationships to integrate data from disparate sources [7]. Thus, a DW with both internal and external (semantic) data requires more powerful tools to define, enrich, integrate, and transform data semantically.

Until now, the DW community has timidly used SW technologies for considering the semantic issues in a DW [64], e.g., for data integration or describing the DW. SW technology aims at converting the 'Web of Documents' to the 'Web of Data' where data are presented and exchanged in a machine-readable and understandable format. In the SW, RDF is used for presenting and exchanging data in a machine-readable format. To express richer constraints on data, formal languages such as RDF Schema (RDFS) [62] and Web Ontology Language (OWL) [37] can be used in combination with the RDF data model to define Knowledge Bases (KBs). Although [1, 7, 50] have used SW technologies to design conceptual frameworks of different phases of ETL processes, no one has so far offered an integrated and implemented framework to build a Semantic DW (SDW) that covers all of the phases of updating of sources, extraction, validation, and integration in one integrated platform.

This paper presents Semantic ETL (SETL), a unified framework for processing and integrating data semantically by bridging SW and DW technologies. SETL uses and extends SW tools and standards to overcome the limitations of the traditional ETL tools. Using SETL, the BI community can benefit by including semantic annotated data in their analytical processes and the SW community can benefit by having an MD view over semantic data for enabling OLAP-like analysis. Hence, it supports publishing better quality RDF datasets as well. The novel contributions of this paper are:

- We propose SETL, a unified framework for semantic ETL. The main tasks can be conducted within the framework are given below:

  (a) In addition to traditional relational data, SETL allows including semantically annotated data (RDF data) in the analytical process. To process a `Non Semantic Data Source`, it builds a semantic layer on top of the source.

  (b) To integrate data from disparate data sources, the user can define the intensional knowledge of SDW in the form of a terminology (TBox) [6] using the ontological constructs. In addition, it provides functionality to annotate the TBox with MD constructs, such as, dimensions, facts, levels, etc. Here, we use the QB4OLAP vocabulary [21] to define the MD constructs.

  (c) SETL provides functionality to produce semantic data (in RDF triples format) from the source data according to the (MD) semantics encoded in the TBox of the SDW.

  (d) SETL creates a SDW, a KB, composed of a TBox annotated with/without MD constructs and an ABox (the instances of the TBox). It also provides functionality to semantically connect internal data with other internal/external data.

- We develop a high-level Python-based programmable framework that provides a number of powerful modules, classes, and methods for performing the tasks mentioned above. It facilitate developers by providing a higher abstraction level that lowers the entry barriers. Thus, one of the contributions is to automatically map the high-level abstraction to executable code.

- Using SETL, we perform a comprehensive experimental evaluation by producing an MD SDW that integrates a `Semantic` and `Non Semantic Data Sources`. The evaluation shows that SETL improves considerably over the competing solutions/tools in terms of programmer productivity, KB quality, and performance.

This paper very significantly extends an earlier workshop paper [17] by (1) adding a sub-component *QB4OLAP* in the architecture of SETL that allows to define a target TBox with MD constructs, (2) adding a semantic layer on top of a `Non Semantic Data Source`, (3) extending the use case for creating an MD SDW which integrates a `Semantic Data Source` and a `Non Semantic Data Source`, (4) updating the semantic transformation algorithm to produce RDF triples according to MD constructs, (5) introducing a provenance graph for tracking how the IRIs used in the SDW are generated, and (6) comparing the experiment with other ETL tools/solutions.

The remainder of the paper is organized as follows. We discuss the terminologies and the notations used throughout the paper in Section 2. Section 2 details the source datasets and the target TBoxes of the SDW that we use as the running example. Section 3 gives an overview of SETL and its components. Section 5 describes the Definition Layer of SETL framework. The ETL Layer of the framework is described in Section 6. Section 7 describes the implementation of the framework in details. We evaluate SETL in terms of productivity, quality, and performance in Section 4. Section 5 describes related work. Finally, we conclude and give pointers to future work in Section 6.

2 Preliminary Definitions

In this section, we give the definitions of the notions and terminologies used throughout the paper.

RDF Graph An RDF graph can be represented as a set of statements, called RDF triples. The three elements of a triple are subject, predicate, and object, respectively, and a triple represents a relationship between its subject and object by its predicate. Let I, B, and L be the sets of IRIs, blank nodes, and literals, respectively, where we denote the set of RDF terms $(I \cup B \cup L)$ as T and $(I \cap B \cap L) = \emptyset$. An IRI is an unique identifier that can be used to identify a resource globally (Web-scope). Blank nodes serve as locally-scoped identifier for resources that are unknown to the outside world. Literal are a set of lexical values enclosed with inverted commas. An RDF triple is defined as a 3-tuple (s, p, o), where $s \in (I \cup B)$, $p \in I$, and $o \in (I \cup B \cup L)$. An RDF graph G is a set of RDF triples, where $G \subseteq (I \cup B) \times I \times T$ [24].

Knowledge Base A Knowledge Base (KB) is typically composed of two Components: TBox and ABox. The TBox introduces the domain terminology. The ABox is the set of assertions representing individuals or instances. The ABox assertions must follow the TBox [6]. In this paper, we assume the components of a KB are described by a set of RDF triples, i.e., a KB is an RDF graph without distinguishing classes and instances.

The TBox is defined as a 3-tuple:

$$TBox = (C, P, A^O)$$

where C, P, and A^O are the sets of concepts, properties, and terminological axioms, respectively. A concept provides a general description of the features for similar types of resources. We use the terms "concept" and "class" interchangeably. Similar to other ontological formalisms, in this paper, we distinguish between object and datatype properties, depending on the RDF element used as object in the RDF triple. An object property relates concept instances, represented as IRIs, while a datatype property is used to associate instances to literals. A^O describes a domain's concepts, properties, and the relationships among them. RDF-schema (RDFs) and the Web Ontology Language (OWL) provide basic constructs to define the TBox of a KB.

3 A Use case

This section describes the source datasets, and the TBoxes for integrating and understanding the knowledge of those datasets, used as the running example in this paper. We consider three Datasets: a Danish Agricultural dataset (DAD) [22], a Danish Business dataset (DBD) [14], and a European Union (EU) Farm Subsidy (`Subsidy`) dataset and integrate the datasets in two steps. First, we integrate the DAD with DBD to (re-)produce a SDW (called SETLKB) from the earlier paper [4]. Second, by integrating the `Subsidy` dataset and the produced SETLKB (which is considered as an external `Semantic Data Source`), we build the final MD SDW. In our context, the integration of different sources is achieved in an iterative and incremental manner. As proposed in [2], each iteration integrates a new source with the results of the previous integration (a single existing source table in the first iteration).

We build the SDW instead of a traditional DW because we want to publish the resulting DW on the Web in a format that other sources can easily integrate and make use of. SDW follows Linked Open Data (LOD) principles and provides IRIs that other people and sources can dereference and therefore receive information from them without having to issue an analytical query. Besides, it provides links to other Web accessible datasets. Therefore, in principle, we could even issue queries involving remote data at other sources, which is something that a traditional DW cannot handle. Thus, our approach enables analyzing situational data (from external datasets) to personalize the user analysis, as discussed in [1, 2].

The DAD consists of three smaller Datasets: *Field*, *Organic Field*, and *Field Block*. All the datasets are available in Shape [48] format. The *Field* dataset has 9 attributes and contains all registered fields in Denmark. This dataset overall contains information about 641,081 fields. The *Organic Field* dataset has 12 attributes and contains information about 52,060 organic fields. The *Field Block* dataset has 12 attributes for 314,648 field blocks [4].

The DBD is provided in CSV format. The dataset consists of two sub Datasets: *Company* and *Participant*. The *Company* dataset consists of 59 attributes and contains information about 603,667 companies and 659,639 production units. The *Participant* dataset describes the relations that exist between a participant and a legal unit [4]. Figure A.1 shows how the datasets are connected to each other.

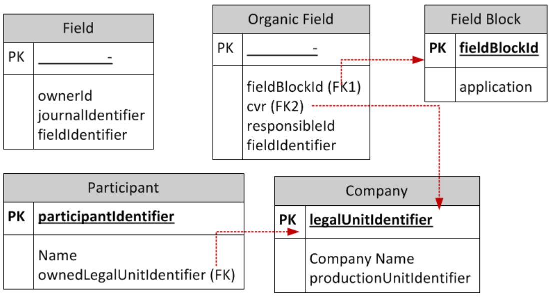


Fig. A.1: The schemas of the DAD and DBD datasets. *Field*, *Field Block* and *Organic Field* are spatially connected [4].

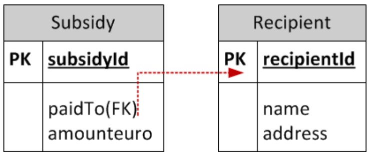


Fig. A.2: The conceptual schema of the Subsidy dataset. For simplicity, we do not show all columns.

Every year the EU provides subsidies to the farms of its member countries. We collect EU Farm subsidies for Denmark from `http://data.farmsubsidy.org/index.html?prefix=Raw/2014/`. The dataset contains two MS Access database Tables: *Recipient* and *Subsidy*. The *Recipient* table contains the information of 342,786 recipients who receive the subsidies and the *Subsidy* table contains the amount of subsidy given to the recipients in different years. In total, the *Subsidy* table contains 4,392,390 distinct records. The conceptual schema of the `Subsidy` dataset is shown in Figure A.2.

To integrate the DAD and DBD datasets, we need to define a TBox of the SDW. The TBox of the SDW is illustrated in Figure A.3. This is an updated version of the TBox (ontology) described in [4]. We start the description from the `bus:Ownership` concept. This concept contains information about the owners of companies, the type of the ownership, and the start date of the company ownership. This concept is related to the `bus:Owner` concept through the `bus:isOwnedBy` property. The `bus:Owner` includes the details of the owners. The `bus:Ownership` has a one-to-many relationship with the `bus:Company` concept and they are related to each other through the property `bus:hasCompany`. The `bus:Company` concept is related to the concepts `bus:BusinessFormat` and `bus:ProductionUnit`. The concepts `bus:Company` and `bus:ProductionUnit` are also related to the `bus:Activity` concept through one main activity and/or through one to three secondary activities. Each company and each production unit have a postal address and an official address. Therefore, both concepts are related with the `bus:Address` concept. Each address has an address feature and it is contained within a par-

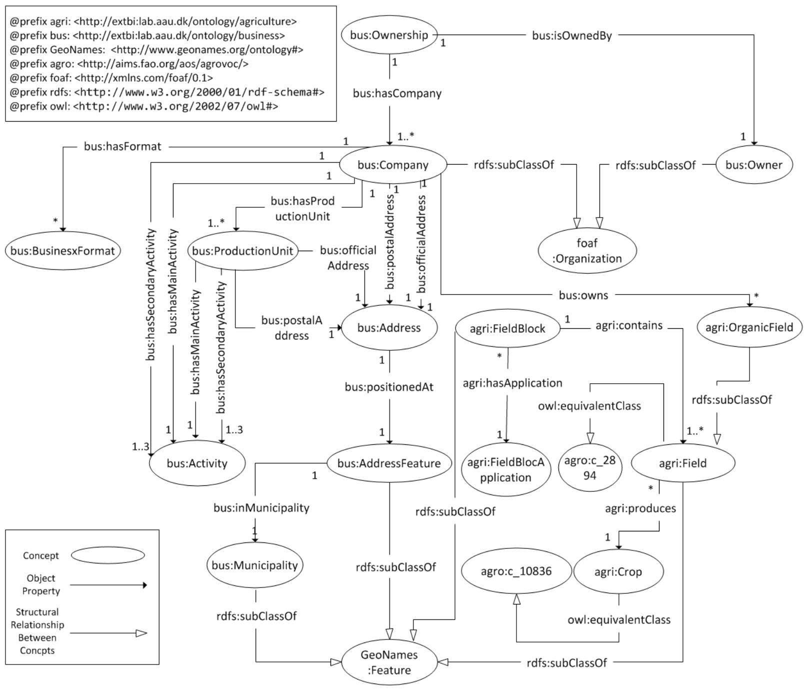


Fig. A.3: The TBox for the Danish Agricultural dataset and Danish Business dataset. The arrows show the cardinality of the relationship. Because of the large number of concept data type properties, they are not included in the figure.

ticular municipality. The `bus:Company` concept is also related to the concept `agri:OrganicField` through the relation `bus:owns` as each company may contain one or more organic fields. Through this relation, the business and agricultural datasets are connected. The `agri:Field` concept defines the structure for all the registered agricultural fields of Denmark. Thus, `agri:OrganicField` is a subclass of `agri:Field`. The `agri:Field` is also equivalent to the UN definition of a European field. A field produces a crop, thus, `agri:Field` is connected to `agri:Crop`. A field is contained within a field block. Each field block has an application. The concepts `bus:AddressFeature`, `bus:Municipality`, `agri:Field`, `agri:FieldBlock` are defined as subclasses of the `GeoNames:Feature` concept in the GeoNames ontology. According to the semantics encoded in the TBox, we produce a SDW named SETLKB.

We create an MD SDW by integrating the produced SETLKB and the `Subsidy` dataset. The *Recipient* table in the `Subsidy` dataset contains the information of recipient id, name, address, and etc. From the SETLKB, we can extract information of the owner of a company who may receive the EU farm subsidy. The TBox of the SDW is shown in Figure A.4, where the

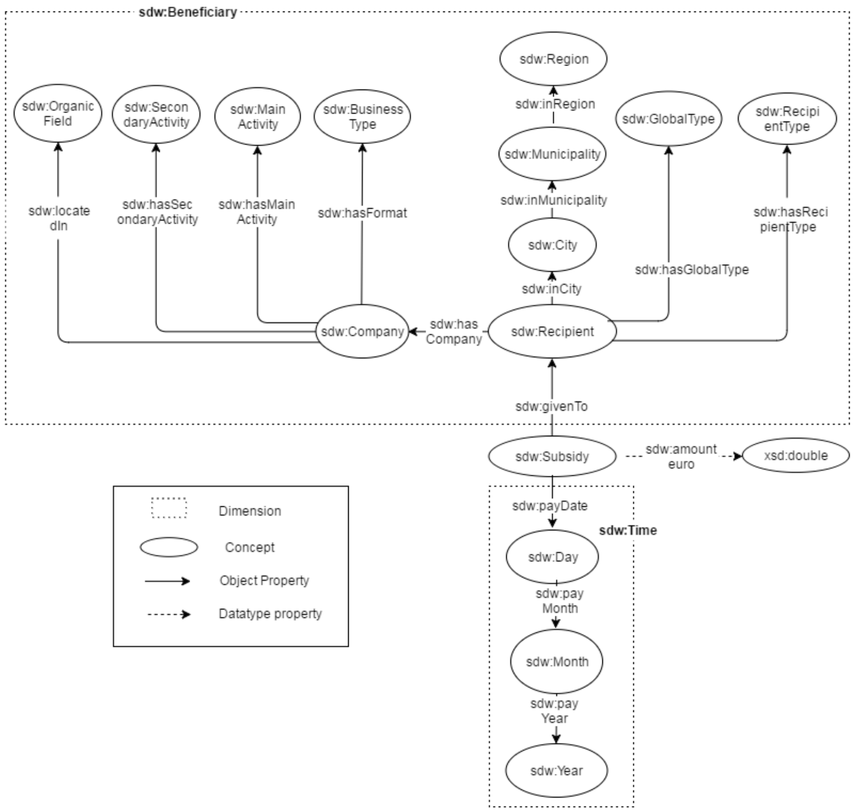


Fig. A.4: The TBox of our example SDW. Because of the large number, not all data properties of the dimensions/levels are shown.

concept `sdw:Subsidy` represents the factual concept[1] of the MD SDW and `sdw:amounteuro` is the measure. The SDW has two Dimensions: `sdw:Beneficiary` and `sdw:Time`. The dimensions are shown by a box with dotted line in Figure A.4. Here, each level of the dimensions is represented by a concept and the hierarchies of the dimensions, i.e., how the levels of the dimensions are connected to each other through object properties, are also shown.

4 SETL Framework Overview

In this section, we present the overview of our Semantic ETL framework (SETL in short). SETL follows a demand-driven approach to design a (MD) SDW. The first step of this approach is to identify and analyze the informa-

[1]A factual concept is the concept that provides a general description of the features for the facts of the DW.

tion requirements of business users and decision makers. Then, based on the gathered requirements, the next two steps of the DW are to build the DW schema and to build the ETL [23]. As the data in a SDW should be semantically connected, we assume the SDW to be a KB and it allows to produce an MD schema for the SDW to benefit from OLAP. The semantic ETL process populates the SDW from the data sources according to the semantics captured in the schema.

Our discussion focuses on SETL's architecture and its main components that support the different steps of creating a SDW. Requirement engineering (RE) is the process of identifying the needs of involved stakeholders and modeling and documenting those requirements in a form that is comprehensible, analyzable and communicable [34]. RE in the DW is itself a research topic and it is beyond the scope of this paper. We direct readers to [23, 34] for RE. The main steps that SETL supports are: defining a TBox for the SDW based on the domain of interest, extracting data from multiple heterogeneous data sources, transforming the source data into RDF triples according to the target TBox, linking the data internally and externally, and loading the data into a triple store, and/or publishing the data on the Web as Linked Data (LD).

Figure A.5 illustrates how the components of SETL framework are connected to each other. We divide the framework into three Layers: Definition Layer, ETL Layer, and Data Warehouse Layer. The red-colored dashed lines in Figure A.5 separate the layers. In the Definition Layer, the SDW schema, sources, and the mappings among the sources and the target are defined. The *SDW TBox Definition* component is used to define a TBox describing the relevant data and the SDW schema based on the requirements. The sub-component *QB4OLAP* is used to define the MD semantics of the SDW. Using the *Define Mapping* component, the user can define the mappings between a source TBox and the target TBox. To map between a `Non Semantic Data Source` and the target TBox, the *TBox Extraction* component generates a TBox representing the `Non Semantic Data Source`. In the ETL Layer, an ETL process to populate the SDW from sources are designed. The *Extraction* component extracts data from multiple sources, the *Traditional Transformation* component cleanses and formats (e.g., unique value generation, null value handling, noisy data filtering etc.) the extracted data and stores them in a `Staging Area`. The `Staging Area` is a storage used to keep the intermediate results of the ETL sub-processes. Then, the *Semantic Transformation* converts the data into RDF triples according to the target TBox. As a sub-task *Semantic Transformation* stores the meta information of concepts, properties, instances, and IRIs used in the SDW into the `Provenance Graph`. The *External Linking* component links the resources in the created RDF dataset to other external resources. The *SaveToFile* component writes the created RDF dataset into a file on disk. Finally, the *Load* component can either directly load the

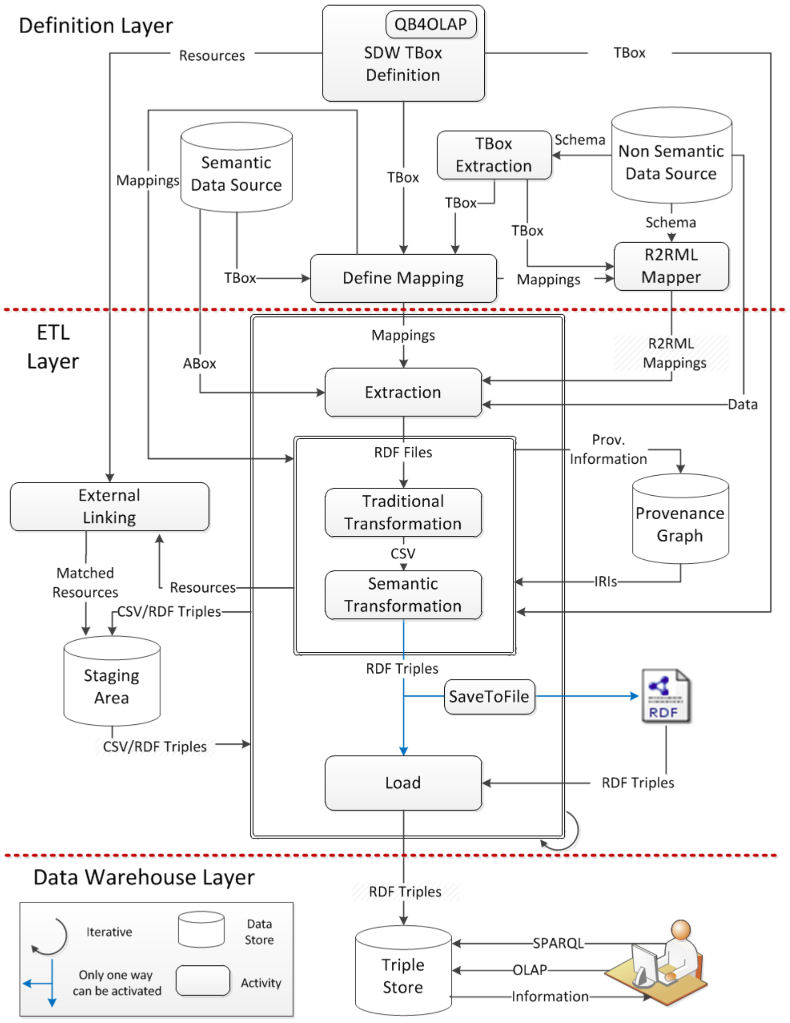


Fig. A.5: SETL architecture.

RDF triples created by the *Semantic Transformation* component or load from the RDF dump file into the `triple store`, which can directly be queried by the user. Extraction-Transformation-Load is an iterative block repeated for each ETL flow (shown as a curved arrow in Figure A.5). The Data Warehouse Layer concerns where the transformed semantic data should be stored. SPARQL queries can be used to analyze the stored data. The SDW defined in MD fashion can also be integrated with an OLAP tools to perform OLAP queries. Here, we focus on building an ETL process. The following sections describe the Definition Layer and ETL Layer along with their components in more details.

5 Definition Layer

In this section, we describe the Definition Layer of SETL. This layer integrates different components required to define the schema of a SDW, to define the different sources that feed data in the SDW, and to define the mappings between the sources and the target. The following sections describe the different components of the Definition Layer.

5.1 SDW TBox Definition

The data in a SDW are semantically connected with other internal and/or external data. Therefore, capturing the semantics of the data at the conceptual level is indispensable. SETL uses ontological constructs to design a SDW (in our context the SDW TBox) because of the following reasons. First, ontologies allow for a semantic integration of disparate data sources as we can explicitly define how two concepts of an ontology are structurally related and how they are associated with different properties. Second, given the cubes share dimensions, it facilitates drill-across operations [3] by basic graph operations during the integration of cubes. Third, it is machine-readable and allows us to automate the ETL modelling tasks. Fourth, compared to other representations, it is easier to evolve the DW. Fifth, it allows to preserve the semantics of the `Semantic Data Sources`.

As mentioned in Section 2, some standard languages, such as RDFS or OWL can be used to describe a TBox. They both provide basic constructs to define the formal semantics of the TBox. OWL uses `owl:Class` and `rdfs:Property` to define concepts and properties of the TBox, uses `rdfs:subClassOf` and `rdfs:subPropertyOf` to define hierarchical relationships among concepts and among properties, respectively, and uses `rdfs:domain` and `rdfs:range` to associate properties with concepts.

On-Line Analytical Processing (OLAP) is a technology to analyze the data available in a DW to support decision making [59]. As OLAP is on-line, it should provide answers quickly. To enable OLAP queries, the DW is represented using the MD model. The central attraction of the MD model of a business is its simplicity and the easy and intuitive way of making analytical queries [29]. In the MD model, data are viewed in an n-dimensional space, usually known as a data cube, composed of facts (the cells of the cube) and dimensions (the axes of the cube). A fact is the interesting thing or process to be analyzed (for example, analysis of subsidy given by the EU) and the attributes of the fact are called measures (e.g., amount of subsidy), usually represented as numeric values. A dimensions is organized into hierarchies, composed of a number of levels, which permit users to explore and aggregate measures at various levels of detail. For example, the *Address* hierarchy (*recipient* → *Municipality* → *Region* → *Country*) of the *Beneficiary* dimension allows to aggregate the subsidy amount at various levels of detail. Therefore, to enable OLAP, we support the MD representation of the SDW.

Although the version of the framework described in the workshop paper [17] allows to define a TBox using OWL, it does not support to define MD constructs. We extend this version by adding the *QB4OLAP* sub-component to support MD constructs. To describe the MD semantics at the TBox level, we use the QB4OLAP vocabulary [20]. QB4OLAP is used to annotate the TBox with MD constructs and is based on the RDF Data Cube (QB) which is the W3C standard to publish MD data on the Web [15]. The QB is mostly used for analyzing statistical data and does not adequately support OLAP MD constructs. Therefore, we direct to QB4OLAP. Figure A.6 depicts the QB4OLAP vocabulary [20]. The terms prefixed with "qb:" are from the original QB vocabulary, and QB4OLAP terms are prefixed with "qb4o:" and displayed with gray background. 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 *pro* means that *pro* is an RDF property with domain A and range B. White triangles represent sub-classes or sub-properties.

In QB4OLAP, the concept `qb:DataStructureDefinition` is used to define the structure of a cube in terms of dimensions, measures, and attributes. To define dimensions, levels, level-attributes and hierarchies, the concepts `qb4o:DimensionProperty`, `qb4o:LevelProperty`, `qb4o:LevelAttribute`, and `qb4o:Hierarchy` are used, respectively. The association between a level-attribute and a level is defined by `qb4o:hasAttribute` property. The hierarchies are connected with dimensions via the property `qb4o:hasHierarchy`. Hierarchies are composed of pairs of levels, which are defined by the concept `qb4o:HierarchyStep`. The reason of defining the hierarchy as a collection of pairs is to establish a rollup relationship between the levels of a pair. A rollup relation is defined by `qb4o:RollupProperty` and the cardinality of the rollup relation is stated using the `qb4o:pcCardinality`. Each pair of levels is connected to a rollup relation via `qb4o:rollup`. The role of the

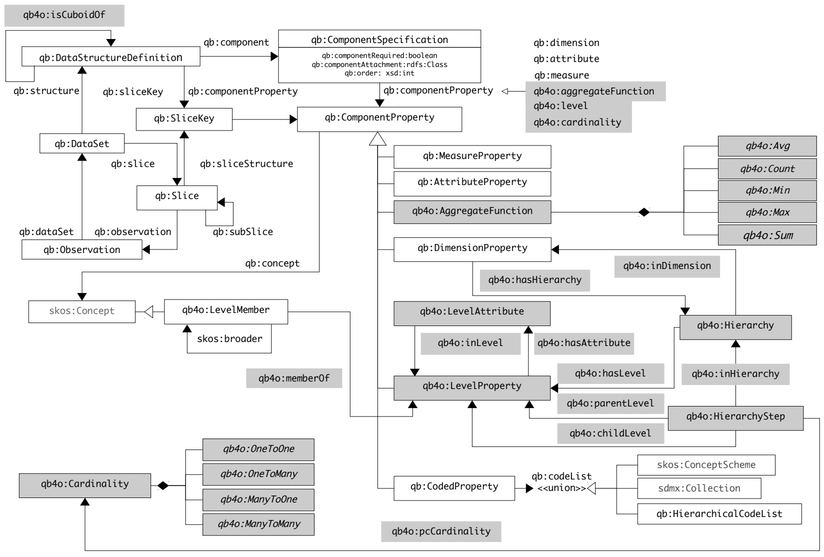
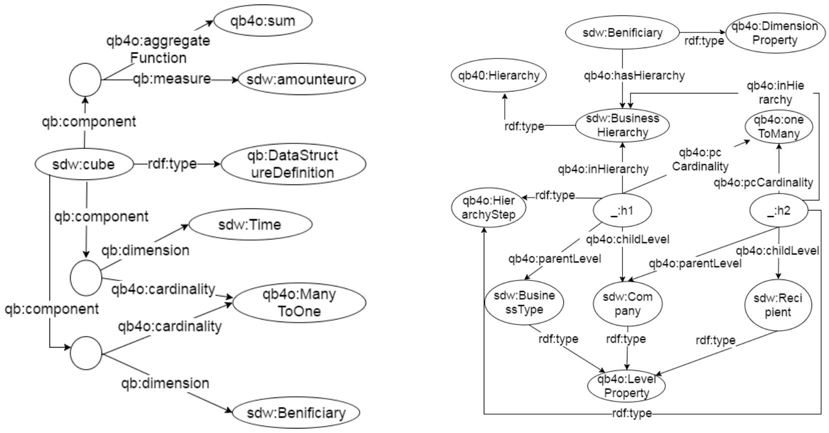


Fig. A.6: QB4OLAP vocabulary.

levels of a pair is distinguished using the property `qb4o:parentLevel` and `qb4o:childLevel`. Pairs are connected with hierarchies through the property `qb4o:inHierarchy` [20]. Therefore, the set of concepts C and the set of properties P in the TBox contains $C_m \subset C$ and $P_m \subset P$ created from the QB4OLAP constructs, where
$C_m = \{\texttt{qb:DataStructureDefinition}, \texttt{qb4o:DimensionProperty}, ..\}$
and $P_m = \{\texttt{qb4o:hasAttribute}, \texttt{qb4o:hasHierarchy}, \texttt{qb4o:inHierarchy}, ..\}$.

Figure A.7a shows a part of the QB4OLAP cube structure for our use case. A cube may have several dimensions and measures. The cube (`sdw:cube`) has a set of components (blank nodes), which represent different measures, dimensions, attributes, levels, and so on. A measure has a property to store numerical values in the dataset (e.g., `sdw:amount`) and an aggregation function (e.g., `qb4o:sum`). A measure can be analyzed according to different dimensions, e.g., `sdw:Beneficiary`, `sdw:Time`. Figure A.7b illustrates a segment of `sdw:Benificiary` dimension. It has a hierarchy named `sdw:BusinessHierarchy`, which is composed of two hierarchy steps, namely `_:h1` and `_:h2`. Each hierarchy-step maintains the roles between the levels it contains, e.g., `_:h1` contains two levels, namely, `sdw:BusinessType`, and `sdw:Company`, and the parent and the child of `_:h1` are `sdw:BusinessType`, and `sdw:Company`, respectively. The cardinality of the step is defined by



(a) A fragment of the cube structure. **(b)** Beneficiary dimension definition.

Fig. A.7: Description of a part of the cube structure and a dimension of our running example using QB4OLAP.

`qb4o:pcCardinality`.

SETL allows users to define a TBox manually. It allows to define various ontological constructs, such as, concepts, properties, and blank nodes individually as well as to capture how they relate to each other. The user first defines concepts and properties individually and then explicitly connects particular concepts to a set of properties. The MD constructs, such as, dimensions, levels, factual concepts, hierarchies, cube structures, cubes are represented by annotating the concepts of the TBox using the QB4OLAP vocabulary. In order to denote a particular MD construct, the concept is annotated as a member of corresponding QB4OLAP class. For example, the triples (`sdw:Beneficiary rdf:type owl:Class, qb4o:DimensionProperty.`) represent that the concept `sdw:Beneficiary` is annotated as a dimension of the target TBox. At first, the user defines the concepts for dimensions, levels, level attributes, measures, and hierarchies, and then based on the defined concepts, the structure of the data cube, dataset and factual concept are defined. Internally, SETL indexes the set of properties connected to a particular concept. As user can also define a TBox using other ontology editors, such as Protege, SETL also allows to parse a given TBox.

5.2 Data Source

A data source is the source of data that can be used to populate a DW. We define a data source as a 2-tuple:

$$D = (ds, type)$$

where ds is the formal definition of the data source and $type$ is the type of the data source. The types of a data source can be either a `Semantic Data Source` (*ss*) or a `Non Semantic Data Source` (*ns*).

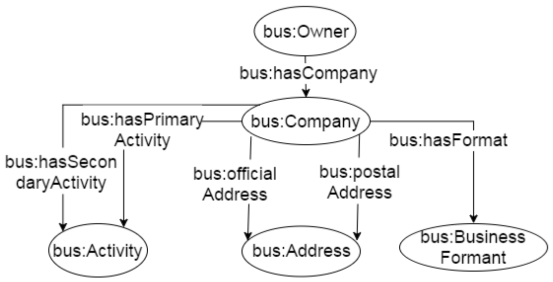


Fig. A.8: A segment of SETLKB TBox.

Semantic Data Source. A `Semantic Data Source` is a typical KB, which is discussed in Section 2. Therefore, a `Semantic Data Source` is defined as $D = (KB, ss)$. Figure A.8 depicts a part of TBox shown in Figure A.3. The semantic graph to logically define a part of TBox and ABox of Figure A.8 is depicted in Figure A.9. In Figure A.9, the segment of the graph above the vertical dashed line represents the TBox and below the dashed line represents sample instances (i.e., ABox). The semantic graph (RDF graph) is considered as a directed label graphs which conceptualize an RDF dataset. In the graph, subjects and objects of RDF triples are drawn by the labeled vertices and predicates are shown as directed labeled edges. To differentiate between the TBox and ABox, we draw the instances in ABox with rectangle, and literal-vertices with simple text.

Typically, an RDF dataset is physically stored as an RDF dump file or in a triple store, for instance, Jena TDB, Virtuoso, Sesame store [10].

Non Semantic Data Source. A `Non Semantic Data Source` can be a relational database (RD), a shapefile, an XML file, an object-oriented database, or a CSV. However, the current version of SETL supports RD, shapefile, and CSV, and the framework is designed in a way that allows it to be easily extended to support other formats as well.

For an RD, $D = (\mathcal{R}, ns)$, and $\mathcal{R}$ can be further defined as $\mathcal{R} = (\mathcal{S}, \mathcal{D})$, where $\mathcal{S}$ and $\mathcal{D}$ represent the schema and data of $\mathcal{R}$. To give the formal definition of a RD, we use the notation from [56]. An RD is composed of a set of tables $\mathcal{T}$. Each $t \in \mathcal{T}$ can be defined by 2-tuples: $t = ((C_t, PK_t, FK_t), R_t)$, where
$-(C_t, PK_t, FK_t)$ denotes the schema of the table t
$-C_t$ is the set of all columns of t
$-PK_t$ is the set of all primary key columns of t
$-FK_t$ is the set of foreign keys
$-R_t$ is the set of rows in t , which represents the data of table t .
Each foreign key $FK_{t,t'} \in FK_t$ is a subset of one or more columns and holds

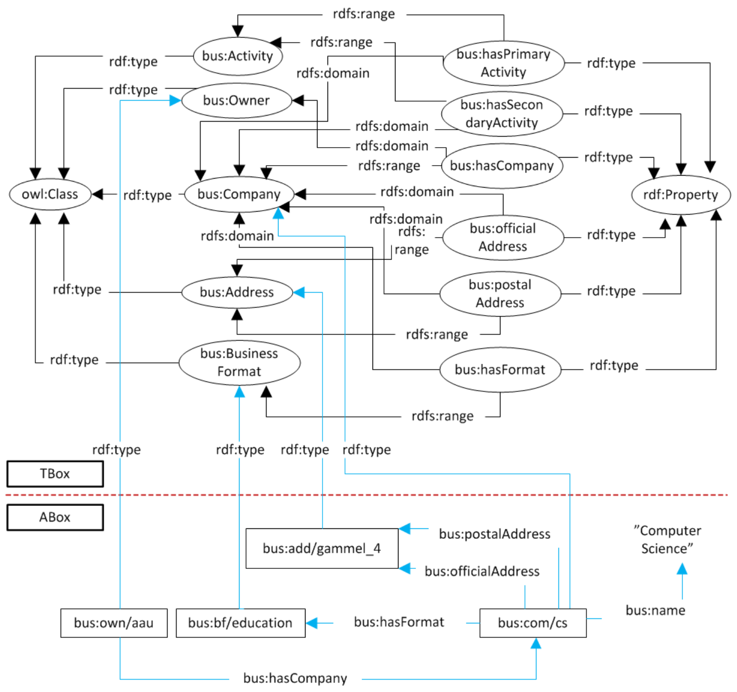


Fig. A.9: Semantic graph to represent a portion of TBox and ABox of Figure A.8.

$\forall FK_{t,t'} \in FK_t, \forall r \in R_t : \exists t' \in \mathcal{T}, \exists r' \in R_{t'}$ *where* $r(FK_{t,t'} = r'(PK_{t'}))$, i.e., each foreign key of a table must be a primary key of same/other table.

(a) Subsidy

subsidyId	paidTo	amount
10611390	366894	617308
10611402	362146	6310

(b) Recipient

recipientId	name	address
362146	Jan'S Værksted	Gammel 4
366894	Videncentret For Landbrug	Agro Park 15

Table A.1: The sample data of `Subsidy` dataset

Therefore, considering all tables $\mathcal{T}$ in the RD, D becomes

$$D = ((\bigcup_{t\in\mathcal{T}}(C_t, PK_t, FK_t)), (\bigcup_{t\in\mathcal{T}}(R_t))), ns).$$

Figure A.2 shows the conceptual schema of the `Subsidy` dataset. Table A.1 shows some sample data of *Subsidy* and *Recipient* tables. We can define the `Subsidy` database as

$\texttt{Subsidy} = ((\{(\{\texttt{subsidyId}, \texttt{paidTo}, \texttt{amount}\}, \{\texttt{subsidyId}\}, \{\texttt{paidTo}\}),$
$(\{\texttt{recipientId}, \texttt{name}, \texttt{address}\}, \{\texttt{recipientId}\}, \{\})\}), (\{(\text{“}10611390\text{”}, \text{“}3668$

94",“617308"),(“10611402",“362146",“6310")},{(“362146",“Jan'SVrksted", “Gammel4"),(“366894",“VidencentretForLandbrug",“AgroPark15")}),ns).

Typically, the tables of a RD are stored in a Relational Database Management System (RDBMS), for example, PostgreSQL, Oracle RDBMS, Microsoft SQL server etc.

A shapefile [48] is a file format that stores geometric location and attribute information of geographic features in vector format. Points, lines, and polygons can be used to describe a geographic feature. It can also be represented in the format of a table. This table is just like any other relational table except it contains a special geometry column. This column stores the actual geometry shape of geographic data. These are the edges and vertices locations that make up the spatial data. Typically, all the other columns in this table are the attributes information associated with the spatial data.

A CSV file is a comma separated values file, which allows data to be saved in a table structured format. Each line of the file is a row of the table and each row consists of one or more fields, separated by commas. Thus, shape and CSV files can be formalized using the notation of RD.

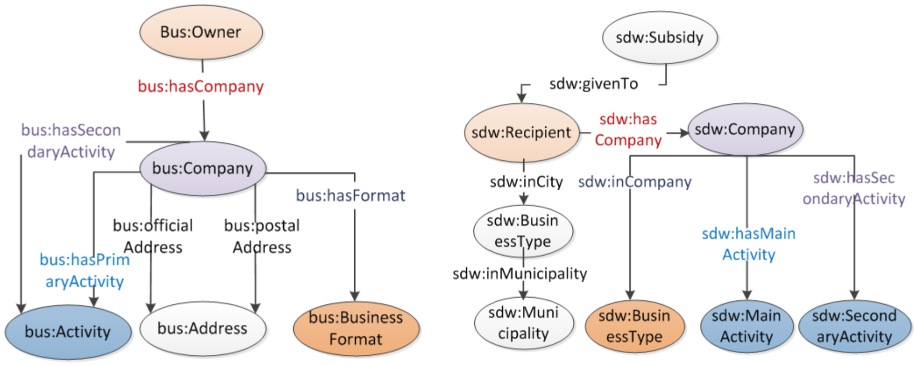
5.3 Define Mapping

A relationship between semantically similar but autonomously designed data needs to be established [54]. As data sources are highly heterogeneous in syntax, mapping should be done between sources and the target at the schema level. Mappings define the relationship between the elements (either concepts or properties) of a source and the corresponding elements in the target TBox [18].

Given two TBoxes T_S and T_T, a mapping maps an element in T_S to the corresponding element in T_T. T_S and T_T are called the source TBox and target TBox, respectively. We formally define an *TBox Mapping* as

$$Map(T_S, T_T) = \{(e_{s1}, e_{t1}, type_i) | e_{s1} \in T_S, e_{t1} \in T_T, type_i \in \{\texttt{skos:exact}, \texttt{skos:narrower}, \texttt{skos:broader}, \texttt{owl:sameAs}, \texttt{rdfs:subClassOf}, \texttt{rdfs:subPropertyOf}\}\}.$$

Each 3-tuple $(e_{s1}, e_{t1}, type_i)$ in $Map(T_S, T_T)$ represents that e_{s1} in T_S is mapped to e_{t1} in T_T with the relationship $type_i$. The relationship can be either *equivalence* relationship ($e_{s1} \equiv e_{t1}$) or *subsumption* relationship ($e_{s1} \sqsubseteq e_{t1}$) [33]. Different knowledge representation languages use different properties to represent *equivalence* and *subsumption* relationships. Therefore, the set of properties supporting the (*equivalence* and *subsumption*) relationships can be extended, but in the current version, we use, `skos : exact`, and `owl : sameAs` to represent equivalence relationship and `skos : narrower`, `skos : broader`, `rdfs:subClassOf`, and `rdfs : subPropertyOf` are used to represent *subsumption* relationship.



(a) T_S: Source TBox. **(b)** T_T: Target TBox.

Fig. A.10: Fragments of a source and the target TBoxes to be mapped. Same colors across the TBoxes indicate mapped elements.

Figure A.10 shows the fragments of a source TBox (Figure A.10a) and the target TBox (Figure A.10b). The mappings from the source TBox to the target TBox are shown in Table A.3. SETL allows users to define the mappings manually to keep them up to date with the needs of end-users.

Entity in T_S	Mapped entity in T_T	Relation
`bus:Owner`	`sdw:Recipient`	*subsumption* (`bus:Owner` $\sqsupseteq$ `sdw:Recipient`)
`bus:Activity`	`sdw:MainActivity`	*subsumption* (`bus:Activity` $\sqsupseteq$ `sdw:MainActivity`)
`bus:Activity`	`sdw:SecondaryActivity`	*subsumption* (`bus:Activity` $\sqsupseteq$ `sdw:SecondaryActivity`)
`bus:hasCompany`	`sdw:hasCompany`	*equivalence*
`bus:hasPrimaryActivity`	`sdw:hasPrimaryActivity`	*equivalence*
`bus:hasSecondaryActivity`	`sdw:hasSecondaryActivity`	*equivalence*
`bus:BusinessFormat`	`sdw:BusinessType`	*equivalence*
`bus:hasFormat`	`sdw:inCompany`	*equivalence*

Table A.3: Mapping example from the source TBox to the target TBox

Mapping for Semantic Data Source. As the schema of a `Semantic Data Source` is defined using a TBox and we assume that the TBox is integrated in the source, no additional step is required to define the mappings between the source and target TBoxes. However, as discussed in Section 1, sometimes RDF datasets do not include an explicit schema. In that case, a TBox should be derived from the RDF [12]. Currently, SETL does not provides this facility. It will be addressed in future.

Mapping for Non Semantic Data Source. In Figure A.5, it is shown that the *Define Mapping* component takes a source and the target TBoxes as input and outputs the mapped elements across the TBoxes. Thus, a further step is required to extract the TBox from a `Non Semantic Data Source`. The following paragraph defines how a TBox is defined as a semantic layer on top of the underlying `Non Semantic Data Source`.

TBox Extraction TBox Extraction is the process of constructing a TBox (semi-) automatically from a given data source [39]. We define the extraction process f from a data source D by:

$$f : D \to \mathcal{TB}$$

where $\mathcal{TB}$ is the TBox derived from D. Given the heterogeneous types of sources (e.g., relational, xml, object-oriented) from where the ontologies to be derived, an instance of f should be defined for each type of sources [56].

x	$f_{rd}(x)$
$\forall t \in \mathcal{T}$	`owl:class`
$\forall c \in C_t$	`owl:DatatypeProperty` [`rdfs:domain` = $f_{rd}(t)$, `rdfs:range`=type(c)]
$\forall FK_{t,t'} \in C_t$	`owl:ObjectProperty` [`rdfs:domain` = $f_{rd}(t)$, `rdfs:range`=$f_{rd}(t')$]

Table A.4: TBox extraction function from RD

Using the notation of a RD described in Section 5.2, we define the TBox extraction function for a RD as $f_{rd} : \mathcal{S} \to \mathcal{TB}$, where S is the schema of the given RD, composed of set of tables $\mathcal{T}$. For this function, we also use the notation and process from [56]. Table A.4 shows the mapping between the elements of a RD schema and corresponding OWL constructs. Each table $t \in \mathcal{T}$ of the RD is mapped to an OWL class, each column $c \in C_t$ is mapped to a datatype property, and each foreign key $FK_{t,t'} \in C_t$ is mapped to an object property. The additional properties (e.g., `rdfs:domain`, `rdfs:range`) related to the main property are shown within a square bracket in Table A.4. For example, for an OWL property, the domain and range are also shown in Table A.4. Figure A.11 shows the extracted TBox of the `Subsidy` database shown in Figure A.2.

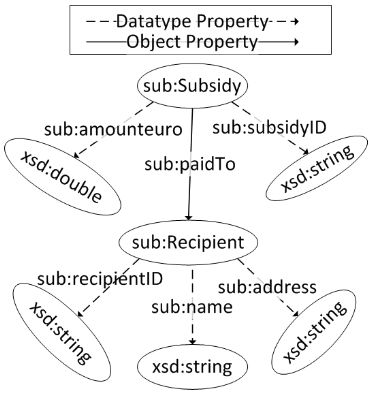


Fig. A.11: The extracted `Subsidy` TBox.

To map the relational data to RDF data using the extracted TBox the *R2RML Mapper* component is used. This component uses the R2RML mapping language, which is a W3C standard to define customized mappings from relational data to RDF [43]. This mapping is done by users. As all relational data may not be relevant to the target, we also consider the output of *Define Mapping* component in the *R2RML Mapper* (shown in Figure A.5) to map only the relevant data. An R2RML mapping is itself represented as an RDF graph. Listing A.1 shows a segment of the R2RML mapping between the `Subsidy` dataset (in Table A.1) and the `Subsidy` TBox (shown in Figure A.11). An R2RML mapping document consists of one or more structures called TripleMaps (e.g., `sub:subsidyMap` and `sub:recipentMap` in Listing A.1). The `rr:TriplesMap` class consists of three properties, namely, `rr:logicalTable`, `rr:subjectMap`, and `rr:predicateObjectMap`. Each TripleMap contains a reference to a logical table (e.g., Subsidy, Recipient) using the property `rr:logicalTable` (lines 5-6). A logical table can be a table, or a view or a SQL query. The property `rr:subjectMap` specifies the target class of TBox and the IRI generation process for each row (lines 7-10). The RDF triples generated from a row share the same subject. The property `rr:predicateObjectMap` defines a target property and the generation of the value of the property using `rr:objectMap`. The type of the RDF terms is either a constant, or a template, or a column value. A constant-valued term map always generates the same RDF terms. PredicateMaps are usually constant-valued (line 12). If the term map is a column-valued, the values in the specified column of the logical table will be used to generate the terms (lines 13-14). If it is template-valued, the terms will be generated based on the given template. A template is a string that concatenates several columns names or the base IRI and the values of one or more column names to generate unique IRIs. Line (8-9) creates IRIs for the subject of the triple using `rr:template` term map. The term map also have specified term type (IRI, Blank node, or Literal) [47].

Listing A.1: An R2ML mapping from Subsidy TBox to Subsidy relational data.

```
1  @prefix rr: <http://www.w3.org/ns/r2rml\#>.
2  @prefix sub: <$http://extbi.lab.aau.dk/ontology/sub/> .
3  sub:subsidyMap
4          a rr:TriplesMap ;
5          rr:logicalTable  [
6          rr:tableName 'Subsidy' ] ;
7          rr:subjectMap [
8                  rr:template 'http://extbi.lab.aau.dk/
9                  ontology/sub/subsidy/{subsidyId}' ;
10                 rr:class sub:Subsidy ; ] ;
11         rr:predicateObjectMap [
12                 rr:predicateMap [rr:constant sub:subsidId
13                 ];
14                 rr:objectMap [
15                         rr:column 'subsidyId' ;
16                         rr:termType rr:Literal ;] ;
17         ];
18         rr:predicateObjectMap [
19                 rr:predicateMap sub:paidTo ;
20                 rr:objectMap [
21                         rr:template 'http://extbi.lab.aau.
22                 dk/ontology/sub/recipient/{paidTo}' ; ] ;
23         ];
24         rr:predicateObjectMap [
            rr:predicateMap sub:amounteuro ;
            rr:objectMap [
                    rr:template "{amounteuro}.00" ;
                    rr:termType Literal ;] ;
        ];
sub:recipientMap
        a rr:TriplesMap ;
        ....
```

6 ETL Layer

In this section, we describe the ETL Layer of SETL. This layer integrates different components to design an ETL process to populate the SDW from different data sources. The following sections describe the different components in details.

6.1 Extraction

Extraction is the process of acquiring data from sources. In this section we describe how to extract data from `Semantic Data Sources` and `Non Semantic Data Sources`.

Extraction from Semantic Data Source. SPARQL is the standard pattern matching language for querying a `Semantic Data Source`. It uses SELECT query to retrieve the desired output from the source. The output of the SPARQL SELECT query is a bag of bindings for the variables in the SPARQL query pattern. Unlike SELECT query, the SPARQL CONSTRUCT query is used to construct the triples from the `Semantic Data Source`, i,e., the output of the query itself is an RDF graph. As we need to retrieve the triples from a `Semantic Data Source`, in this section, we discuss the semantics of the basic SPARQL CONSTRUCT query.

To define the basic semantics of SPARQL CONSTRUCT query, we need to introduce the notion of triple patterns and basic graph pattern (BGP). A triple pattern is an RDF triple which allows query variables in any position of the triple, i.e., $t_p \in (I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$, where V is a set of query variables that range over all RDF terms T, and $V \cap T = \emptyset$. A query variable $v \in V$ is led by the symbol $'?'$, e.g., $(?s\ \ p\ \ o)$. A BGP is a set of triple patterns connected via logical conjunctions. The SPARQL graph pattern expression is defined based on BGP.

The execution of triple patterns against a `Semantic Data Source` produces a set of solution mappings. A solution mapping μ is a partial function that maps V to T, $\mu : V \to T$. The domain of μ, denoted by $dom(\mu)$, is the subset of V for which μ is defined.

Let $\mathcal{K}$ and t_p be a `Semantic Data Source` and a triple pattern, and $var(t_p)$ denotes the set of query variables t_p contains. The evaluation of t_p against $\mathcal{K}$

is the set of all mappings that can map t_p to a triple contained in $\mathcal{K}$, i.e.,

$$[\![t_p]\!]_{\mathcal{K}} = \{\mu \mid dom(\mu) = var(t_p) \textit{ and } \mu(t_p) \in \mathcal{K}\}.$$

where $\mu(t_p)$ is the triple obtained by replacing the variables in t_p according to μ. The SPARQL CONSTRUCT query makes the class of queries whose inputs and answers are RDF graphs [31]. Therefore, we define the answer $\mathtt{ans}(t_p, \mathcal{K})$ of the SPARQL CONSTRUCT query as a set of triples matched t_p in $\mathcal{K}$, i.e.,

$$\mathtt{ans}(t_p, \mathcal{K}) = \{\mu(t_p) \mid \mu \in [\![t_p]\!]_{\mathcal{K}} \textit{ and } \mu(t_p) \in \mathcal{K}\}.$$

If $B = \{t_{p1}, t_{p2}, t_{p3}, ...\}$ is a BGP, then the evaluation of B over $\mathcal{K}$ is

$$[\![B]\!]_{\mathcal{K}} = \{\mu \mid dom(\mu) = var(B) \textit{ and } \mu(B) \subseteq \mathcal{K}\}.$$

Here, $\mu(B)$ is the set of triples obtained by replacing the variables in the triple patterns of B according to μ. Therfore, the answer $\mathtt{ans}(B, \mathcal{K})$ of the query is

$$\mathtt{ans}(B, \mathcal{K}) = \mu(B) = \bigcup_{t_{pi} \in B} \{\mu(t_{pi}) | \mu \in [\![B]\!]_{\mathcal{K}} \textit{ and } \mu(t_{pi}) \in \mathcal{K})\}.$$

Consider the `Semantic Data Source` shown in Figure A.9 and we want to construct the triples for the properties of `bus:Company`. Here, $t_{cp} = (?property\ \mathtt{rdfs:domain\ bus:Company})$, $dom(\mu) = ?property$. The evaluation of t_{cp} over DAB is

$$\begin{aligned} [\![t_{cp}]\!]_{DAB} = \{&\{?property \to \mathtt{bus:hasFormat}\},\\ &\{?property \to \mathtt{bus:officialAddress}\},\\ &\{?property \to \mathtt{bus:postalAddress}\},\\ &\{?property \to \mathtt{bus:hasPrimaryActivity}\},\\ &\{?property \to \mathtt{bus:hasSecondaryActivity}\}\}. \end{aligned}$$

The answer $\mathtt{ans}(t_{cp}, DAB)$ of the query is the set of following triples.

```
bus:hasFormat rdfs:domain bus:Company .
bus:officialAddress rdfs:domain bus:Company .
bus:postalAddress rdfs:domain bus:Company .
bus:hasPrimaryActivity rdfs:domain bus:Company .
bus:hasSecondaryActivity rdfs:domain bus:Company .
```

Typically, data from a `Semantic Data Source` is retrieved by applying queries to a local RDF file or through a SPARQL endpoint. When the output of a given query is very big or if it is required to extract all triples from the `Semantic Data Source`, a simple download strategy might fail because

Algorithm 5: TripleExtraction.

```
   Input: $endpoint, batchsize$
   Output: $tripleset$
   begin
1      $P \leftarrow retrieveDistinctProperty(endpoint)$
2      foreach $property\ p \in P$ do
3          $x \leftarrow retrieveNumberOfTriples(p, endpoint)$;
4          for $count \leftarrow 0$ to $x/batchsize$ do
5              $tripleset \leftarrow$
                 $tripleset + retrieveTriples(count, batchsize, endpoint)$;
6      return $tripleset$
```

some SPARQL endpoints restrict result sizes; DBpedia [16], for instance, does not allow more than 10,000 result triples at a time. To overcome this limitation, we design Algorithm 5. The algorithm takes a *sparqlEndpoint* URL and a *batchsize* (indicating how many triples to extract at a time) as parameters and returns the extracted triples. In line 1, the algorithm extracts all distinct properties in the KB. Then, for each property (lines 2-3), the algorithm determines the number of triples containing the property as predicate. Based on this number the algorithm determines how many queries need to be sent to receive all these triples and executes the queries (lines 4-5). This is done by sorting the triples by subject, using *batchsize* in the LIMIT clause, and using the iteration number multiplied by the batchsize as OFFSET.

Extraction from Non Semantic Data Source.

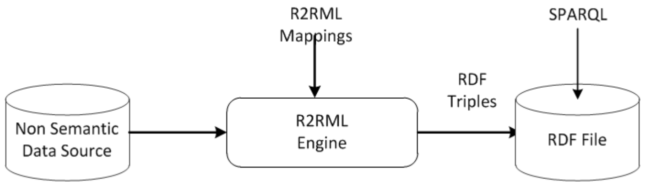


Fig. A.12: The process of extraction from `Non Semantic Data Source`.

Figure A.12 shows the process of extraction from a `Non Semantic Data Source`. At first, we make a semantic version of the `Non Semantic Data Source` using the *R2RML Engine*. The *R2RML Engine* component is typically an R2RML processor that takes a relational database and an R2RML mapping document as input and outputs an RDF graph according to the mapping document [63]. Hence, the data of the `Non Semantic Data Source` are converted into RDF dataset which can simply be queried using SPARQL queries discussed in the Section 6.1. The following triples are the sample output of the *R2ML Engine* component with the input Listing A.1 and Table A.1.

```
sub:subsidy/10611390 sub:subsidyId 10611390 .
sub:subsidy/10611390 sub:paidTo sub:recipient/366894 .
sub:subsidy/10611390 sub:amount 617308 .
sub:subsidy/10600402 sub:paidTo sub:recipient/362146 .
sub:subsidy/10600402 sub:amount 6010 .
......
```

6.2 Transformation

In the data transformation process, the extracted source data are transformed in a way such that it can easily be fed to the SDW. To ensure that data in the SDW are consistent with the semantics of its TBox, i.e., *ABox* follows *TBox*, we divide the transformation tasks into two Components: *Traditional Transformation*, and *Semantic Transformation*. The following sections detail the components.

Traditional Transformation. *Traditional Transformation* includes operations known from traditional ETL tools, such as cleansing the data and formatting the source data according to the target schema. This includes removing duplicate data, recalculating data, normalizing data, renaming attributes, checking integrity constraints, refining data, unique identifier generation, creating new attributes based on existing attributes, null value and default value handling, noisy data filtering, sorting data, and grouping/summarizing data [40]. In this component, the data related to each concept of the target TBox are stored in a separate table where the columns of the table represents the properties associated with the concept and the records of the table represent the instances of the concept. The transformed data are kept into a `Staging Area`. The `Staging Area` is an intermediate storage where the intermediate results of each process are stored. This can prevent the loss of transformed data in case of the failure of the loading process.

Semantic Transformation. *Semantic Transformation* includes operations to create RDF triples according to the semantics of the target TBox from the data output by the *Traditional Transformation* component.

Algorithm 6 describes the steps of converting a table (*table*) (from *Traditional Transformation*) into a set of RDF triples ($\mathbb{T}$). As additional inputs, the algorithm takes the name of an attribute in the table (*resourceKey*) that can be used to uniquely identify each row, a target TBox (*onto*), mapping files between the sources and the target (*mapping*), a *provenance graph* (*provGraph*) and $dataset_{name}$ is the name of the dataset to be created. It is an instance of `qb:Dataset`. The values of *resourceKey* will be used to create resource identifiers. The *provGraph* is required to search for an existing IRI and to store the information of how the IRIs are formulated and its original source (literal/IRI

Algorithm 6: createTriples.

```
Input: $table, resourceKey, onto, mapping, provGraph, dataset_{name}$
Output: $\mathbb{T}$
begin
1    $D \leftarrow makeDictionary(table)$
2    $concepts \leftarrow$ getMatchingConcepts($table, mapping$, concepts($onto$))
3    foreach $c \in concepts$ do
4        foreach $r \in D$ do
5            $sub \leftarrow createInstanceIRI(c, resourceKey(r), provGraph)$
6            if $!(propertyTable(table))$ then
7                $obj \leftarrow createConceptIRI(c, provGraph)$
8                $\mathbb{T}.addtriple(sub,$ rdf:type, $obj)$
9                if $(type(c) =$ qb4o:LevelProperty$)$ then
10                   $\mathbb{T}.addtriple(sub,$ qb4o:memberOf, $obj)$
11               if $(type(c) =$ qb:Observation$)$ then
12                   $\mathbb{T}.addtriple(sub,$ rdf:type, qb:Observation$)$
13                   $\mathbb{T}.addtriple(sub,$ qb:dataSet, $dataset_{name})$
14           foreach $(attribute, value) \in r$ do
15               if $(value! = NULL)$ then
16                   $prop \leftarrow getMatchingProperty(attribute,$
17                     $mapping, property(c, onto))$
18                   $\mathbb{T}.addTriple(sub, createPropertyIRI(c, prop, provGraph),$
                       $createObject(value))$
19   return $\mathbb{T}$
```

and dataset) information. We describe the *Provenance Graph* in Section 6.3.

At first, a dictionary D is created based on the input *table* (line 1). D consists of (key, value) pairs for each row r in *table*; the key corresponds to r's *resourceKey* value. The value in turn corresponds to a list of (attribute, value) pairs where each pair represents one of r's cells.

The next step is to determine which concepts in the target TBox the information contained in the table correspond to (line 2). Based on this information, the algorithm runs through each such concept c. Some tables in relational databases do not directly correspond to a concept, instead they represent many-to-many relationships between instances of concepts (entity types). Hence, only for those tables that directly correspond to a concept, the algorithm creates a new resource for each entry in the dictionary D and a triple (with `rdf:type` as predicate) that defines the new resource as an instance of the concept (lines 6-8). To create an instance IRI, the *createInstan-*

ceIRI() function (in line 5) first checks the *Provenance Graph* whether there is an existing IRI for the instance; if it finds an existing one, then returns it, else creates new one and updates the *Provenance Graph*. Different type of resources (i.e, concept, property, instance), takes different parameters to create new IRIs. Table A.5 shows the parameters to create different type of resources.

Type of IRI	Base IRIs	Concept Name	Property Name	Property Range Name	Value(s)	Resource Key(s)
Concept IRI	Y	Y				
Property IRI	Y	Y	Y			
Object Property value IRI	Y			Y	Y	
Instance IRI	Y	Y				Y

Table A.5: Required parameters for creating different types of IRIs

If the concept c is a `qb4o:LevelProperty`, then a triple saying that the new resource is a member of the level property c is added (lines 9-10). If the concept c is a factual concept, i.e., a `qb:Observation`, then the resource is added as a `qb:Observation` and the `qb:dataSet` of the resource is $dataset_{name}$ (lines 11-13). Furthermore, for each (*attribute*, *value*) pair for which the *attribute* is defined as a match of one of c's properties and *value* is not null, a triple encoding this information is created (lines 14-18). Depending on the type of property (data type or object), the object of the created triple either corresponds to a literal or a resource. For example, the values of `sdw:hasrecipient` are resources and the values of `sdw:amounteuro` are literals.

The computational complexity of Algorithm 6 depends on its constituents: the number of concepts in the target TBox matched with the input table, and the size of the given table. For each row of the input table, the *makeDictionary* operation creates an entry for the dictionary D. Therefore, line 1 takes $O(N)$ complexity, where N is the number of rows in the table. The loop depends on the number of concepts matched with the table. In the worst case, the number of concepts in the target TBox matched with the table is equivalent to the number of concepts in the target TBox. Let say C is the number of concepts in the target TBox matched with the input table. The for loop in line 4 is executed for each entry in D, therefore, it takes $O(N)$ time. The for loop in lines 14-18 is executed for each attribute of the table and it takes $O(M)$ time, where M is the number of attributes in table. Thus, the total time for Algorithm 6 is $O(N) + O(CNM) = O(CNM)$ in worst case, where C is the number of concepts in the target TBox, N is the size of the dictionary, and m is the average size of the entries of the dictionary. The best case is $O(NM)$ where, $C = 1$.

6.3 Provenance Graph

The provenance graph is an RDF graph that stores the meta information of concepts, properties, instances and IRIs of resources used in a SDW. At this version, we only consider the provenance of IRIs, and the provenance graph contains the type of resource it indicates in the target, its prefix (base IRI), its source dataset, and the original IRI or the original literal corresponding to each IRI. The type of a resource is either a concept, a property or an instance. For example, an IRI can be generated for a concept *Company*, for a property *hasCompany*, or for an instance of Company DanskBank. The prefix is the base IRI of the target. The graph is queried using SPARQL query. Figure A.13 shows how an IRI is represented in the IRI provenance graph. The blank node represents that the IRI is an instance of `sdw:Day`. The literal used to generate the IRI is shown as the value of `pro:originalLiteral` which is taken from the `subsidy` dataset. The URL of the `subsidy` dataset are shown as the value of `pro:sourcedataset`. The new generated IRI is shown as the value of the `pro:generatedIRI` property. This graph needs to be built by semantic-ETL processes.

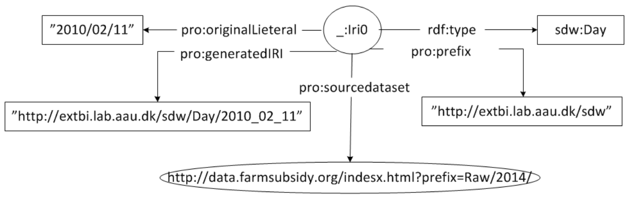

"2010/02/11"
pro:originalLieteral
_:Iri0
rdf:type
sdw:Day
pro:generatedIRI
pro:prefix
pro:sourcedataset
"http://extbi.lab.aau.dk/sdw/Day/2010_02_11"
"http://extbi.lab.aau.dk/sdw"
http://data.farmsubsidy.org/indesx.html?prefix=Raw/2014/

Fig. A.13: An IRI represented in the provenance graph.

6.4 External linking

Algorithm 7 formalizes the steps required to link an internal resource to external resources. It takes *intResource*, *externalDataSource flag*, and k as input parameters. *intResource* is a given internal resource that we want to find external links for, and *externalDataSource* can be either a keyword search API or the SPARQL endpoint of a KB. In our DOLAP workshop paper [17], we use the Sindice API [57] as a default API, However, the service of the API is currently stopped by its provider. Therefore, in this version, we replace the default API with the DBpedia Lookup API [2]). *flag* indicates whether the target external source is a search API ($flag = 0$) or an external KB ($flag = 1$), and k is the number of top k IRIs that we want to retrieve.

At first, the algorithm creates a semantic bag ($sb_{internal}$) for the internal resource (line 1). Such a semantic bag consists of triples describing the internal

[2]https://github.com/dbpedia/lookup

Algorithm 7: LinkToExternalResources.

```
Input: $intResource, externalDataSource, flag, k$
Output: $\mathbb{L}$
begin
1     $sb_{internal} \leftarrow semanticBag(intResource)$
2     if $flag = 0$ then
          $E \leftarrow search(intResource, externalDataSource, k)$
3     else
          $E \leftarrow search(intResource, externalDataSource, query, k)$
4     foreach $extResource \in E$ do
5         $extTriples \leftarrow retrieveTriples(extResource)$
6         $sb_{external} \leftarrow semanticBag(extTriples)$
7         if $match(sb_{internal}, sb_{external}) > \delta$ then
8             $alignedPairs \leftarrow alignedPairs \cup (intResource, extResource)$
9     $alignedPairs \leftarrow userInteraction(alignedPairs)$
10    foreach $pair \in alignedPairs$ do
11        $\mathbb{L}.addTriple(intResource,$ owl:sameAs, $extResource)$
12        $ER \leftarrow getEquivalentResource(extResource)$
13        foreach $equiResource \in ER$ do
14            $\mathbb{L}.addTriple(intResource,$ owl:sameAs, $equiResource)$
15    return $\mathbb{L}$
```

resource. Then, if *externalDataSource* is a search API, then a web service request embedding the *intResource* is sent through the API for the top *k* matching external resources (line 2). If *externalDataSource* is the SPARQL endpoint of a KB, then it submits a query to the endpoint for retrieving top *k* matching external resources (line 3). An example query is shown in Listing A.2. The query retrieves top-20 resources from a KB which are the subjects of triples whose objects are literals containing the string "obama" most. For each external resource (*extResource*), the algorithm retrieves triples describing it (line 5), creates a semantic bag ($sb_{external}$) from them (line 6), and compares the bag to the one created for the internal resource (line 7). If the Jaccard Simarity (see Equation A.1) between the two semantic bags exceeds a certain threshold δ, the internal and external resource are considered a match (lines 7-8).

$$J(sb_{internal}, sb_{external}) = |\frac{sb_{internal} \cap sb_{external}}{sb_{internal} \cup sb_{external}}| \tag{A.1}$$

After having identified all candidate pairs, the user can optionally interact with the system and filter pairs. Then, for each pair of internal and external resources, a triple with the `owl:sameAs` property[3] is created to materialize the link in the dataset (line 11). Finally, the algorithm also retrieves the set of resources from the KB that are already linked to the external resource via by `owl:sameAs` property and materializes the links to the internal resource as well (lines 12-14).

The computational complexity of Algorithm 7 depends on its constituents: the generation of semantic bags, the similarity computation, and the number of aligned pairs. A semantic bag of a resource is created from the RDF graph of the resource [45, 46]. The set of triples describing the resource defines the RDF graph of that resource. Therefore, a semantic bag creation takes $O(N)$ time complexity where N is the size of the graph. Hence, line 1 takes $O(N)$ time. The given internal source is compared to every external source. Therefore, the loop in line 4 is executed $O(K)$ times where K is the number of external sources, and since each *semanticBag* operation takes $O(N)$ time, the total time for all calls to *semanticBag* is $O(NK)$. The *match* operation matches the semantic bags of two resources and takes $O(N^2)$ time. Therefore, the total time of the loop lines 4-8 is $O(KN + KN^2) = O(KN^2)$. The maximum number of aligned pairs can be the equivalent of the number of external sources K. Hence, the total time for the loop in lines 10- 14 is $O(KR)$, where R is the average number of equivalent resources of each aligned external sources. Thus, the total time for Algorithm 7 is $O(N + KN^2 + KR) = O(K(N^2 + R))$, where K, N, R are the number of external resources, N is the average size of the RDF graphs, and R is the average number of resources same as with external resources.

Listing A.2: An example SPARQL query for keyword based searching.

```
SELECT ?sub (count(?sub) as ?count)
WHERE {
   ?sub rdf:type ?class.
   ?sub ?pro ?label.
   Filter regex(?label, "obama", "i")
}
group by ?sub
order by desc(?count)
limit 20
```

6.5 Load

This component loads the RDF triples created by *Semantic Transformation* into a triple store or saves to an RDF dump file. SETL currently uses Jena TDB [27] as a triple store and allows loading data batch-wise. It provides two kinds of load, namely, trickle load and bulk load. The trickle load mode transforms and feeds data as it arrives from the previous transformation step without staging on disk using SPARQL INSERT queries. As it provides concurrent

[3]An `owl:sameAs` property indicates that two IRI references refer to the same real world object. Hence, subject and object IRIs of a triple with `owl:sameAs` are considered linked.

processing and loading, therefore, it takes long time to store whole dataset because it includes process and loading time. In bulk load mode, the triples are written into a file on disk after transformation and then loaded batch-wise into the triple store.

7 Implementation

We implement the SETL framework, discussed in Sections 4, 5, 6, using Python [42]. The reasons for choosing Python are its comprehensive standard libraries and its support to programmer productivity [55]. The following sections discuss how we implement the main components of the framework.

7.1 Definition Layer

SDW TBox Definition . We define three Python Classes: *Concept*, *Property*, *BlankNode* to give the structure of concepts, properties and blank nodes of a TBox, respectively. These classes play a meta modeling role and the user can instantiate the classes to define their TBox constructs, such as, concept, property, and blank node. The user can define the MD constructs, such as, cube, dimensions, level, factual concept, and hierarchies by instantiating the *Concept* class. Listing A.3 shows a segment of Python script to create a TBox of our running example. It shows how to define different MD constructs, such as, dimension (lines 1-4), level (lines 5-7), hierarchy (lines 8-11), hierarchy step (lines 12-17), level attributes (lines 18-27), measure (lines 28-32), cube structure (lines 33-40), cube (lines 41-43), factual concept(43-44). To internally connect each concept with an explicit set of properties, such as data type property, level attribute, object property, functional property, inverse-functional property, SETL offers a method *conceptPropertyBinding()* which takes the list of all concepts, properties and blank nodes as parameters (line 51). The method *createTriples()* creates the triples according to the TBox (line 53).

Listing A.3: A segment of Python script to create the TBox of our running example.

```
 1 # Defining Time Dimension
 2 Time=Concept(name=‘PayDate’,_base_iri=‘sdw:’)
 3 Time.setrdfType([‘owl:class’,‘qb:DimensionProperty’])
 4 Time.setqb4oHasHierarchy(‘sub:TimeHierarchy’)
 5 Time.setrdfsLabel(‘‘‘PayDate"@en’)
 6 # Defining levels for Time Dimension
 7 Day=Concept(name=‘Day’,_base_iri=‘sdw:’)
 8 Day.setrdfType([‘owl:class’,‘qb4o:LevelProperty’])
 9 Day.setrdfsLabel(‘‘‘Day"@en’)
10 Month=Concept(name=‘Month’,_base_iri=‘sdw:’)
11 Day.setrdfType([‘owl:class’,‘qb4o:LevelProperty’])
12 Day.setrdfsLabel(‘‘‘Month"@en’)
13 #␣Defining␣a␣time␣hierarchy
14 TimeHier=Concept(name=‘TimeHier’,_base_iri=‘sdw:’)
15 TimeHier.setrdfType([‘owl:class’,‘qb4o:Hierarchy’])
16 TimeHier.setqb4oHasLevel([‘sdw:day’,‘sdw:Month’}}
17 TimeHier.setqb4oInHierarchy([‘b_t1’}}
18 # Defining a hierarcy step
19 b_t1=BlankNode(name=‘_:t1’)
```

```
b_t1.setrdfType('qb4o:HierarchyStep')
b_t1.setqb4oparentLevel('sub:Month')
b_t1.setqb4ochildLevel('sub:Day')
b_t1.setqb4opcCardinality('OneToMany')
b_t1.setqb4oinHierarchy('sdw:TimeHier')
# Defining level attributes
hasMonth=Property(name='hasMonth',_base_iri='sdw:')
hasMonth.setrdfType(['owl:ObjectProperty',
'qb4o:LevelAttribute'])
hasMonth.setrdfsRange('sdw:Month')
hasMonth.setrdfsDomain('sdw:Day')
paydate=Property(name='paydate',_base_iri='paydate')
paydate.setrdfType(['owl:objectProperty',
'qb4o:levelAttribute'])
paydate.setrdfsDomain('sdw:Subsidy')
paydate.setrdfsRange('sdw:Day')
amount=Property(name='amounteuro',_base_iri='sdw')
amount.setrdfType(['owl:datatypeProperty',
'qb:MeasureProperty'])
amount.setrdfsDomain('sdw:Subsidy')
amount.setrdfsRange('xsd:double')
# Defining a data structure definition
cubestruct=Concept(name='cubestruct',_base_iri='sdw:')
cubestruct.setrdfType(['owl:Class',
'qb:DataStructureDefinition'])
cubestruct.setqbComponent(['b_d','b_m'])
b_d=BlankNode(name='_d')
b_d.setqbdimension('sdw:Time')
b_m=BlankNode(name='_m')
b_m.setqbmeasure('sdw:amount')
# Defining a cube
cube=Concept(name='dataset',_base_iri='sdw:')
cube.setrdfType(['owl:Class','qb:DataSet'])
cube.setqbstructure('sdw:cubestruct')
#␣Defining␣factual␣concept
Subsidy=Concept(name='Subsidy',_base_iri='sdw:')
Subsidy.setrdfType(['owl:Class','qb:Observation'})
#␣Defining␣list␣for␣concepts,␣properties,␣and␣blank␣node
conceptList=list()
propertylist=list()
blankList=list()
#␣Extending␣the␣lists
conceptList.extend([Time,Day,TimeHier])
propertyList.extend([hasMonth,amount,paydate])
blankList.extend([b_t1,b_d,b_m])
#␣Establishing␣relationships␣among␣the␣elements
conceptPropertyBinding(conceptList,propertyList,blankList)
#␣Generating␣triples␣for␣the␣tbox
createTriple(conceptlist,propertyList,BlankList)
```

We also define a Python class named *ParseOntology*(*object*) to parse a given TBox. The user can instantiate it by passing the path of the given TBox. It provides several methods (*getConcepts(), getDataProperties(), getProperties()*, and etc.) to process the TBox.

Define Mapping. To define the mappings between the elements (concepts and properties) of a source TBox and the target TBox, we use a Python *dictionary* which is composed of (key, value) pairs. The key of a pair is the source element and the value of the pair is a list which contains the corresponding target element and relationship type (lines 1-8 in Listing A.4). The relationship type is presented using *sup*, *sub*, and *equi*. Here, *sup* means that the source concept is the super class of the target concept, *sub* means source concept is the sub class of the target concept, and *equi* means *equivalence* relationship. We also define the mappings between all source TBoxes and the target TBoxusing a Python *dictionary* whose keys are the source TBoxes names and the values are the individual source and target mapping dictionary (lines 9-10). Currently, SETL supports basic mapping, however, in the future we will allow it to support complex mappings, for example, mapping a property of a source TBox to a concept of the target TBox.

Listing A.4: Python script for defining mapping.

```
bus_sdw=dict()
bus_sdw={‘bus:Owner’:[‘sdw:Recipient’, sup],
‘bus:Activity’:[‘sdw:mainActivity’, ‘sup’],
‘bus:Activity’:[‘sdw:SecondaryActivity’,␣‘sup’],
‘bus:hasCompany’:[‘sdw:hasCompany’,‘equi’],
‘bus:hasPrimaryActivity’:[‘sdw:hasPrimaryActivity’,‘equi’]
,‘bus:BusinessFormat’:[‘sdw:BusinessType’,‘equi’],
‘bus:hasFormat’: [‘sdw:inCompany’,‘equi’]}
mapping=dict()
mapping[‘bus’]=bus_sdw
```

For processing a `Non Semantic Data Source`, users can define a TBox representing the source using the classes of *SDW TBox Definition*. Then, the mapping between the target and the extracted TBox can be defined using Python *dictionary*. For converting the `Non Semantic Data Source` into RDF triples based on R2RML mapping, we implement the simple algorithms of R2RML processor provided in [63].

7.2 ETL Layer

SETL enables the extraction of data from a KB through SPARQL endpoints, RDF files, relational databases, CSV files, and shapefiles. The framework is designed in a way that allows it to be easily extended to support other formats.

Extraction from a SPARQL endpoint. Using the SPARQLWrapper [51] library, we extract data/triples from a KB by applying queries through a SPARQL endpoint.

Extraction from an RDF file. We use the RDFLib library [44] to execute a query on a local RDF file. If the RDF file is too large to fit into main memory, we split the file into several smaller files, execute the query on the smaller files, and then combine the partial results. However, this method can only be used for simply queries, such as retrieving all triples with a particular predicate. More complex queries involving joins need to be handled differently, e.g., by loading the data into a local SPARQL endpoint first and using the method described in the previous paragraph.

Extraction from a relational database or a CSV file. SETL is based on Petl [40], which is a traditional ETL Python package including a rich set of methods to perform extraction, cleansing, and transformation based on relational databases and CSV files. Making use of these methods, SETL provides the functionality as well.

Extraction from a shapefile. To read data from a shapefile, we use the pyshape [41] library. Internally, SETL can store the data of a shapefile either in a file or in a database table. SETL provides methods to automatically create a table in a database based on the structure of a shapefile and to insert the data from the shapefile into the created table. It also provides a method to store the attribute information of a shapefile in CSV format.

Figure A.1 shows that there is no direct connection between *Field* and *OrganicField*, neither between *Field* and *FieldBlock*. In the target TBox (Figure A.3), however, we have defined connections, e.g., `agri:Field` and `agri:OrganicField` are connected through the `owl:sameAs` property and `agri:FieldBlock` and `agri:Field` are connected via the `agri:contains` property. These connections are computed based on a spatial join that SETL computes during the extraction process for pairs of shapefiles. We use the ArcPy [5] library to perform the spatial join analysis.

Transformation. To perform the operations related to *Traditional Transformation*, described in the 2nd paragraph of Section 6, we again use the Petl [40] library. For creating RDF triples according to the target TBox based on the data output from the *Traditional Transformation* component, we develop a Python method, named *createTriples(table, resourceKey, onto, mapping, provGraph, filepath)* that implements Algorithm 6, where the parameters maintain the same meanings as Algorithm 6. Additionally, it requires a parameter $filepath$, the location of the file, to store the generated RDF triples.

External Linking. We develop a method named *link(intResource, extSource, flag, k)* that implements Algorithm 7. We use Python built-in *requests* library to retrieve the top-k related resources of *inResource* through DBpedia Loopup API in JSON format. Sometimes the queries submitted to a SPARQL endpoint result in timeout error. For this reason, we download the RDF dump files of the selected KB and store them to a triple store locally for accessing it through local SPARQL endpoint.

Load. Bulk load stores the RDF triples from a file (created by $createTriple()$ method) to the triple store. To support bulk load, SETL offers two methods: *bulkTDBLoader (tdblocation, filePath, batchsize)* and *bulkSPARQLInsert(tdblocation, filePath, batchsize, database)*. The *BulkTDBLoader* is used to load data from a file on disk into a triple store using the jena TDBLoader command. The parameter *tdblocation* denotes the location of a TDB database, the *filePath* is the path to the triple file, and *batchsize* is the number of triples to be fed into the store at a time. The *BulkSPARQLInsert* is similar to *BulkTDBLoader* but defines SPARQL INSERT statements to load triples instead of using the jena TDBLoader command. It additionally requires the name of the database as parameter. Unlike bulk load, which loads the triples from a file, trickle load converts the cleansed and transformed data into triples and subsequently loads them into a triple store holding the triples in main memory. The $trickleload(table, resourceKey, onto, mapping, provGraph, tdblocation, batchsize)$ method has same parameters like $createTriple()$ method. In addition, it takes $tdblocation$ and $batchsize$ as input parameters to directly load the triples into the triple store.

8 Evaluation

To evaluate SETL, as discussed in Section 2, we first create a SDW called SETLKB applying the proposed ETL process on sources covering Danish agricultural and business information. Then, we create an MD SDW called SETLSDW using SETL by integrating SETLKB and the `Subsidy` dataset. We create SETLKB in order to be able to compare it with an existing dataset integrating DAD and DBD [4] that has been created in a manual process by using a broad range of different tools. In contrast, SETL allows to perform this process mostly automatically with minimal user interaction. This allows us to compare the different ETL processes as well as the RDF datasets themselves. Furthermore, to show the strength of SETL over the non-semantic traditional data integration tools, we also re-create SETLSDW using Pentaho Data Integration (PDI, also called Kettle) [11] and compare the productivity and the performance of the processes with SETL.

Table A.6 shows the description of SETLKB and SETLSDW. As DAD and DBD do not have any numerical measures, we do not model SETLKB in an MD fashion. On the other hand, SETLSDW is defined with MD constructs. It has 4,392,390 facts, 2 dimensions, 9 hierarchies, and 15 levels.

In the following, we refer to the dataset published by [4] as ExtBIKB. Our evaluation concentrates on three aspects: productivity, i.e., to what extent SETL eases the work of a user when performing an ETL task to produce a SDW, quality of the produced datasets, and performance, i.e., the time required to run the processes.

We run the experiment on a HP ProLiant DL385 server with an AMD Opteron(tm) processor 6376 with 32 cores (only 1 core is used in the experiments); it has 256 GB DDR3 RAM and is running Ubuntu 14.04.1 LTS (Trusty Tahr). The data is stored on an 1-TB SCSI disk running in an HP Smart Array.

Dataset	# of RDF Triples	# of Concepts	# of Datatype Properties	# of Object Properties	# of External Linkings	# of Dimensions	# of Levels	# of Level Attributes	# of Hierarchies	# of Observations
SETLKB	34,177,652	16	76	22	14,153	-	-	-	-	-
SETLSDW	54,995,678	54	63	16	3,538	2	15	58	9	4,392,390

Table A.6: Description of SETLKB and SETLDW.

8.1 Productivity

One of the main advantages of working with SETL is that all the phases of an ETL process can easily be maintained and implemented using Python. SETL offers high-level classes and functionality for development of semantic ETL processes. Using SETL the users can write their own code to create an ETL process in Python. Therefore, it allows extensibility. The modularized structure of SETL allows users to call different modules simply by passing arguments. Hence, it hides the complexity of the processes from the users. For example, to populate the SDW with transformed data, the user just write one line of code to call the method *createTriples(table, resourceKey, onto, mapping, provGraph, filepath)* by passing the required arguments, where the parameters maintain the same meanings as Algorithm 6. Concretely, SETL uses 224 lines of code to implement Algorithm 6. Hence, it provides a higher level abstraction to the user by hiding details. Table A.7 summarizes the tools, languages, and the Lines Of Code (LOC) used to produce the SETLKB dataset using SETL and the ExtBIKB dataset using the process described in [4]. To process a shapefile, SETL provides methods to automatically create and insert data into PostgreSQL. It uses only 2 lines of Python code. To perform spatial intersection operation between two shapefiles, SETL uses the ArcPy [5] library, which is faster than ArcGIS [19] which is used in [4]. On the other hand, the ETL process used in [4] converts the shapefile into PostgreSQL table using the PostgreSQL tool *shp2pgsql*. To make it compatible with vt-SQL, they have performed some find-replace actions. For performing, spatial operation, they use ArcGIS [19]. For filtering inconsistent values, unknown characters and for correcting candidate keys, we use different methods from the Petl library [40], which are efficient and easy to access. To cleanse our all dataset, SETL only takes 173 lines of code. On the other hand, the [4] uses different MySQL views and hand-crafted methods to cleanse the data, and it takes 360 LOC. To generate triples for all data sources, SETL takes only 109 LOC whereas [4] uses 1,511 LOC. SETL only takes two line of codes to call the external linking procedure. On the hand, [4] does not support any linking process. The last row of Table A.8 shows the total number of LOC. SETL takes in total 570 LOC to make the ETL process where 286 LOC is used to build the TBox. On the other hand, for ExtBI, we sum only the LOC of *Cleansing* and *Triple Generation* because we can not count the LOC for other processes, hence, the sum is $1879 + 5NA$. In summary, SETL is significantly more productive, using only a fraction of the LOC of $1879 + 5NA$ and using only Python instead of 7 different languages and tools.

PDI contains a rich set of data integration functionality to create an ETL solution for non-semantic data sources. However, it does not support any functionality to support semantic integration. We use PDI in combination with other tools and manual tasks to re-create SETLSDW. Table A.8 shows the comparison between the ETL processes of SETL and PDI to create SETLSDW. SETL provides built-in classes to annotate MD constructs with a TBox. On the other hand, PDI does not support to define a TBox. To create SETLSDW using PDI, we use the TBox created by SETL. SETL provides built-in classes to parse a given TBox and users can use different methods to parse the TBox based on their requirements. In PDI, we implement a Java class to parse the TBox created by SETL which takes an RDF file containing the definition of a TBox as input and outputs the list of concepts and properties contained in the TBox. The class also includes the functionality to internally connects the concepts and properties of the TBox. PDI is a non-semantic data integration tools, thus, it does not support to process semantic data. In this case, we manually extract data from SPARQL endpoints and materialize them in a relational database to further processing. PDI provides drag & drop functionality to process database and CSV files. On the other hand, SETL provides methods to extract semantic data either from a SPARQL endpoint or an RDF dump file batch-wise. SETL allows users to create an IRI by simply passing argument to the *createIRI ()* method. Moreover, SETL supports to update the provenance graph to store the provenance information of IRIs. On the other hand, PDI does not include any functionality to create IRIs for resources. We define a 23 line of Java class to enable the creation of IRIs for resources. To generate triples from the source data based on the MD semantics of the target TBox, SETL provides *createTriple()* method; users can just call it by passing required arguments. On the other hand, we develop a Java class of 60 lines to create the triples for the sources. PDI does not support to load data directly to a triple store which can easily be done by SETL. Finally, we could not run the ETL process of PDI automatically (i.e., in a single pass) to create the SETLSDW. We make it with significant number of user interactions. In total SETL takes 401 LOC to run the ETL, where PDI takes $471LOC + 4NA + 21Activity$. Thus, SETL creates an MD SDW with minimum number of LOC, user interactions comparing to PDI where users have to build their own Java class, plugin, and manual tasks to enable semantic integration.

Tools	**SETL**			**ExtBI**		
Task	**Used tools**	**Used language**	**LOC**	**Used tools**	**Used language**	**LOC**
TBox Definition	Built-in SETL	Python	286	Virtuoso, Protege	Not Applicable (NA)	NA
CSV	Built-in Petl	Python	2	MySQL,vtSQL	SQL	NA
Reading Shapefile	Built-in SETL	Python	3	ArcGIS,PostgreSQL, shp2pgsql	NA	NA
Spatial join	Built-in ArcPy	Python	2	ArcGIS	NA	NA
Cleansing	Built-in Petl	Python	173	MySQL, Google refine	SQL	360
Triple Generation	Built-in SETL	Python	2	Virtuoso	iSQL	1511
External Linking	Built-in SETL	Python	2	NA	NA	NA
Loading	Built-in SETL	Python	1	Virtuoso	iSQL	4
Total LOC for the complete ETL process			**570**	**1879 + 5 NA[4]**		

Table A.7: Comparison between the ETL processs of SETL and [4].

8.2 Quality

In this section, we examine the extent to which the produced triples are consistent, complete, interpretable, fresh, and semantically linked [53]. We compare the result created with SETL (SETLKB) to ExtBIKB.

Consistency. According to the definition, an object property relates instances of two classes, i.e, if x is a value of an object property, there must exist at least one triple whose subject is x. We found out that for a significant number of values for the object properties `bus:postalAddress` and

Tools	SETL			PDI (Kettle)		
Task	**Used Tools**	**Used Lnaguages**	**LOC**	**Used Tools**	**Used Languages**	**LOC**
TBox with MD semantics	Built-in SETL	Python	312	Protege, SETL	Python	312
Ontology Parser	Built-in SETL	Python	2	User Defined Class	Java	77
Semantic Data Extraction through SPARQL endpoint	Built-in SETL	Python, SPARQL	2	Manually extraction using SPARQL endpoint	SPARQL	NA
Semantic Data Extraction from RDF Dump file	Built-in SETL	Python	2	NA	NA	NA
Reading CSV/Database	Built-in Petl	Python	2	Drag & Drop	NA	Number of used Activity: 1
Cleansing	Built-in Petl	Python	36	Drag & Drop	NA	Number of used Activity: 19
IRI Generation	Built-in SETL	Python	2	User Defined Class	Java	22
Triple Generation	Built-in SETL	Python	2	User Defined Class	Java	60
External Linking	Built-in SETL	Python	2	NA	NA	NA
Loading as RDF dump	Built-in SETL	Python	1	Drag & Drop	NA	Number of used Activity: 1
Loading to Triple Store	Built-in SETL	Python	1	NA	NA	NA
Total LOC for the complete ETL process			**401**	**471 LOC +4 NA + 21 Activity**		

Table A.8: Comparison between the ETL processes of SETL and PDI for SETLSDW.

bus:officialAddress, ExtBIKB does not contain such information. There are 83,302 bus:officialAddresses and 34 bus:postalAddresses that are not used as a subject of any triple in ExtBIKB. In SETLKB, on the other hand, all bus:officialAddresses and bus:postalAddresses have correctly been related to the bus:Address concept. Hence, data consistency is completely maintained in SETLKB.

Information loss. In ExtBIKB, fields that do not produce any crop are not represented. This affects 2,650 instances of the concept agri:Field. Similarly, instances of agri:OrganicField that have the value "999999-99" for the agri:FieldBlock property are not represented in ExtBIKB. This affects 1,213 instances. For SETLKB on the contrary, we do not want to miss any agri:Field or agri:OrganicFields instance even though they do not have crop information. Hence, in the cleansing step, we replace the value "999999-99" with *NULL*. If an instance of a concept contains *NULL* or an unknown value for a property, the *createTriples ()* method (described in Section 6) simply does not produce a triple for that property. This is how SETLKB ensures data completeness.

In the original raw datasets, we can find out since when a legal unit belongs to an owner. This is not possible in ExtBIKB if an owner owns more than one company. We fix this issue by extending the target TBox. In addition, we also add the information related to the type of the owner, i.e., whether the owner is a person or another company, both are missing in ExtBIKB.

Furthermore, the main and secondary activity of a company or production unit are treated as the same activity in the ExtBIKB. In SETLKB, the concepts bus:Company and bus:ProductionUnit are connected to bus:Activity through two object properties: bus:hasMainAcitivity and bus:hasSecondaryActivity. Thus, SETLKB provides more information than ExtBIKB.

Redundant triples. If a property is inversely connected to another property, then storing the same types of triples for both properties, simply by interchanging the subject and predicate of the triples, makes a KB redundant. The properties `bus:belongsTo` and `bus:hasProductionUnit` are inversely connected to each other and relate the concepts `bus:Company` and `bus:ProductionUnit`. In ExtBIKB, triples are stored for both properties separately, which produces extra 659,639 triples. On the other hand, we create triples only for `bus:belongsTo` property. Hence, SETLKB reduces redundancy.

External linking. We enhance SETLKB by linking it to external knowledge bases at instance level. Section 6.4 describes the linking process. On the other hand, ExtBIKB is not connected with other external knowledge bases at instance level. We run the linking process only for instances of `agri:Crop`, `agri:FieldBlockApplication`, `bus:BusinessFormat`, and `bus:Municipality` because for instances of other concepts, DBpedia [16] does not contain relevant information. Note that the following results are produced by the automatic matching process without any pre-filtering by the user (which is mentioned as an optional step in Section 6.4). Table A.9 shows the different types of concepts for which we run the linking process, their total number of internal instances, the number of internal instances linked to the external resources, the number of external resources connected to the internal instances, and the accuracy rate of the linking. In total, 196 instances of SETLKB are linked to 9,618 external resources. To evaluate the linking process, we randomly chose 5% of the triples that linked internal and external resources and manually evaluated their correctness. The accuracy rate in Table A.9 shows the accuracy rate in percentage. We only consider the values of `rdfs:comment`, `rdfs:label` and `rdf:description` in the semantic bags of compared resources in order to make the process faster. One of the reasons having the low accuracy rate for some cases is the language. The description of each instance is translated from Danish to English using Google Translate API, which sometimes fails to give accurate translation.

Listing A.5: The SPARQL query that connects SETLKB with DBpedia to retrieve the description of top-3 municipalities where most of the companies belongs to.

```
PREFIX bus:<http://extbi.lab.aau.dk/ontology/business/>
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX owl:<http://www.w3.org/2002/07/owl#>

select ?municipality ?abstract
where {
        {select ?municipality count(*)
        where {
        ?company a bus:Company.
        ?company bus:officialAddress ?address.
        ?address bus:positionedAt ?addFeature.
        ?addFeature bus:inMunicipality ?municipality.}
Group by ?municipality
order by desc(count(*))
limit 3
}
?municipality owl:sameAs ?db.
SERVICE <http://dbpedia.org/sparql?default-graph-uri=
  http://dbpedia.org> { ?db dbo:abstract ?abstract .
  FILTER (lang(?abstract) = 'en')}
}
```

Because of being linked with external resources, users can analyze SETLKB based on external knowledge bases as well. Listing A.5 shows an SPARQL query that retrieves the descriptions of top-3 Danish municipalities from DBpedia with the most companies in them. The query returns eight entries. The first 3 entries describe Copenhagen, next 2 entries describe Aarhus, and the last 3 entries describe Aalborg. This query cannot be answered by ExtBIKB.

Concept	# of instances	Linked instances	External resources	Accuracy
`agri:Crop`	244	95	5,309	83%
`bus:BusinessFormat`	30	16	976	66%
`agri:FieldBlockApplication`	17	6	403	43%
`bus:Municipality`	98	79	2,930	73%
Total	**389**	**196**	**9,618**	**66.25%**

Table A.9: Linking local resources to external resources.

Based on these findings, we can conclude that SETLKB is more consistent, complete, and resourceful than ExtBIKB. Table A.10 summarizes some notable triple-wise differences between SETLKB and ExtBIKB.

Concept(c)/ Property(p)	# of Triples SETLKB	# of Triples ExtBIKB	Comments
Whole KB	34,177,652	32,457,657	SETLKB is the updated version.
External Linking	9,618	0	No instance is linked with external sources in ExtBIKB.
Company(c)	603,667	603,668	One duplicate triple in ExtBIKB.
Address(c)	499,850	497,938	In ExtBIKB, Address concept were not mapped with the postal address of Danish Business dataset.
Field(c)	52,060	50,842	ExtBIKB used poor cleansing techniques.
OrganicField(c)	613,903	611,093	ExtBIKB used poor cleansing techniques.
belongsTo(p)	659,638	659,640	Two duplicate triples in ExtBIKB.
hasProductionUnit(p)	0	659,640	belongsTo and hasProductionUnit are inversely connected.
hasOwnerType(p)	353,954	0	ExtBIKB did not include owner type information.

Table A.10: Comparison between SETLKB and ExtBIKB.

8.3 Performance

In this section, we discuss the time required for different sub-processes to create SETLKB and SETLSDW using SETL, the performance of different sub-process, and the comparison of performance with the previous solution [4] and PDI.

Runtime of the different phases. Figure A.14 and A.15 illustrate the time and the percentage of the total time that each phase of the ETL processes of SETLKB and SETLSDW respectively accumulated. The figures show both the time for each phase in separate and the overall runtime of the complete

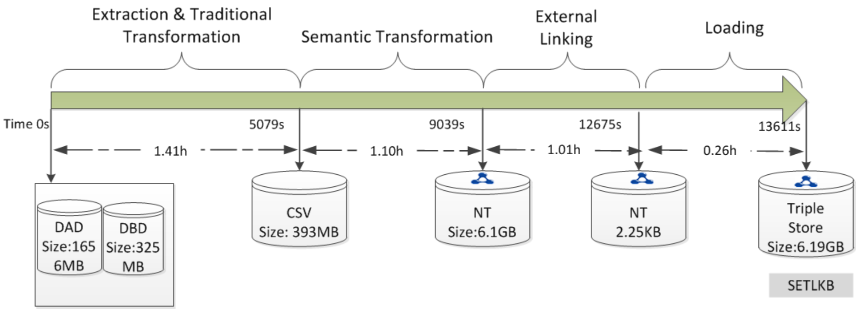


Fig. A.14: Time for the ETL process to create SETLKB. s and h stand for second and hour respectively.

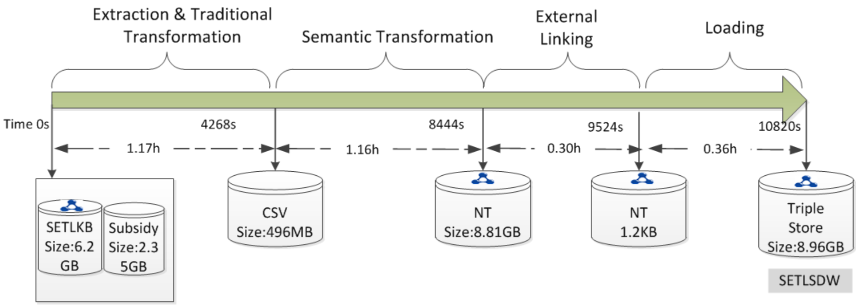


Fig. A.15: Time for the ETL process to create SETLSDW.

process. The times represent averages over 5 test runs. As SETL automatically connects all the phases, the end time of one phase becomes the starting time of the next. Figure A.14 and A.15 also show the materialized output with the data size of each phase.

Figure A.14 shows that the *Extraction* and *Traditional Transformation* phase take 37% of the total time. This phase takes the longest time comparing to other phases because of the following reasons. First, the DAD is given with three Shapefiles: *Field*, *Organic Field*, and *Field Block*; therefore, we need to do some preprocessing tasks to extract the attribute information from the files and to keep them into a PostgreSQL database. Second, we perform two spatial join operations: between *Field* and *Field Block*, and between *Field* and *Organic Field* datasets. To perform the spatial join operations, we use ArcPy library which takes 30% time of this phase which is out of our control. Third, we extract a TBox from the dataset and based on the source-target mappings, we keep the data into 9 smaller tables. We perform extensive cleansing operations for correcting candidate keys, fixing inconsistent and incomplete data, and filtering noisy data which are also time intensive. Fourth, we also fix and filter the noisy, inconsistent and incomplete data from Company and Participant datasets and split the two datasets into 19 smaller tables according to the source-target mappings which are also time consuming.

The *Semantic Transformation* process is time-intensive because all the data needs to be processed and matched to the TBox of the SETLKB. In Section 3.4, we discuss the computational complexity of *Semantic Transformation* which depends on the three nested loops. The first loop iterates for the number of concepts in the target TBox matched with the input table. The second loop repeats for each row of the input table. Further, each iteration creates an IRI for each row of the table and updates the `Provenance Graph`. The third loop iterates for each value of a row, and *Semantic Transformation* creates a separate triple for each value in the SDW according to the semantics encoded in the target TBox. On the other hand, the building block of a traditional DW is tuple which contain multiple values. Many triples of SDW are kept into a tuple of a traditional DW table. This is why our SDW takes longer time comparing to the traditional DW.

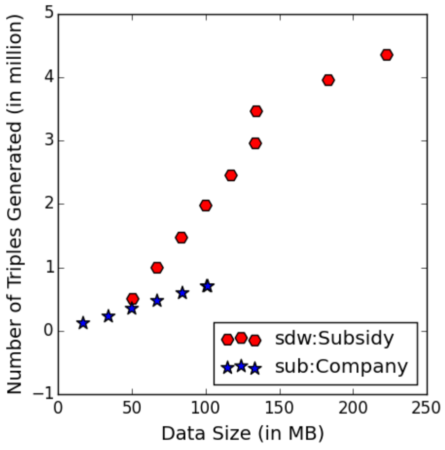
The total time required for the *Linking* process is 3,657 seconds. All the information of the resources are given in Danish, so we first translate the name of the resource into English, then use DBpedia Lookup API to retrieve the top 20 relevant resources. Eventually, we use our instance matcher described in Section 6.4 to link the internal and external resources. The computational complexity depends on the number of external resources to be matched, the average size of the RDF graphs of internal and external resources, and the number of resources same as external sources. As we consider the semantic bags of two resources in the matching operation, we need to retrieve the RDF graph of each external resource from DBpedia through its SPARQL endpoint. Moreover, the matching operation is time consuming because it takes $O(N^2)$ complexity, where N is the average size of the RDF graphs of internal and external resources. Table A.11 shows the time consumed for linking the instances of each concept. In summary, SETLKB is linked to 9,618 external resources. To load all triples at a time, the *BulkTDBLoader* takes 936 seconds which is almost 7% of the total time. For loading triples into a triplestore, we depend on the native loading commands of the underlying triplestore. The elapsed time depends on the internal strategy of the triplestore which is also out of control. We have discussed the performance of the different loading methods in the earlier workshop paper [17].

In Figure A.15, it is shown that the ETL process takes SETLKB and the `Subsidy` dataset as input. the *Extraction* and *Traditional Transformation* phase includes time for 1) Filtering the noisy data from the `Subsidy` dataset (2.3 GB in size), 2) converting the `Subsidy` dataset into RDF according to the R2RML mapping, 3) extracting data from the `Semantic Data Sources` through its SPARQL endpoints, and 4) stores them into the `Staging Area` based on the source-target mappings. The *Semantic Transformation* makes the RDF triples based on the MD semantics encoded in the target. Here, we only link the instances of `sdw:Municipality` with the external resources. In total, the ETL process for creating SETLSDW takes 10,820 seconds.

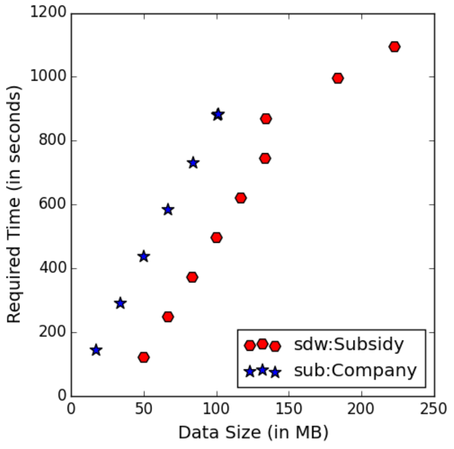
Concept	**Time(in seconds)**
`agri:Crop`	1,925
`bus:BusinessFormat`	504
`agri:FieldBlockApplication`	63
`bus:Municipality`	1,165
Total	**3,657**

Table A.11: Time required for the linking process.

Performance analysis of Semantic Transformation. Figure A.16 shows the performance of our *Semantic Transformation* method. To evaluate the method, we consider the factual concept `sdw:Subsidy` from SETLSDW and the central concept `sub:Company` from SETLKB because `sub:Company` also acts as a super class of the `sdw:Company` level of SETLSDW. Figure A.16a shows the number of triples generated with the increasing input data size, and Figure A.16b shows the processing time with the increasing input data size. With the increase of input data size, the number of generated triples (shown in Figure A.16a) and the processing time (shown in Figure A.16b) increase almost linearly for both concepts. In general, the processing time and the number of generated triples depends on the nature of the input data, e.g., whether it corresponds to a fact entry, a dimension entry or a level entry, how big the value of each attribute is. Figure A.16 summarizes that the number of triples generated per data size and the data processing rate for `sub:Company` are higher than that of `sdw:Subsidy`. This is because each entry of `sdw:Subsidy` only contains a measure property and the pointers to two dimensions, namely, `sdw:Beneficiary` and `sdw:Time`. Therefore, `sdw:Subsidy` is straightforward to process while `sub:Company` requires more effort as it involves 17 properties with relatively long strings, such as address, name, email, etc. Table A.12 reports the performance of *Semantic Transformation* for `sdw:Subsidy` and `sub:Company` using different comparison metrics. Triple generate rate of *Semantic Transformation* for `sdw:Subsidy` is 4.9 times higher than that of `sub:Company`. `sdw:Subsidy` has a higher data process rate (i.e., the amount of data that can be processed per time) than `sub:Company`. To process 1 MB input data, `sdw:Subsidy` takes 4.8 seconds while `sub:Company` takes 8.7 seconds. We also compare the number of triples per (1 MB) data size for both input and output data size. 1 MB of the output file generated for `sdw:subsidy` contains 6,134 triples where `sub:Company` contains 5,393 triples. The output file size for `sdw:Subsidy` is 31 times larger than the input data size. On the other hand, the output file for `sub:Company` is 12 times larger than its input file.



(a) Number of triples vs data size.



(b) Processing time of increasing data.

Fig. A.16: Performance of *Semantic Transformation* based on the number of generated triples and processing time with increasing data size.

Comparison between SETL and ExtBI [4] To make it comparable with the results of [4], we also run the computer on our local machine which is 2.10 GHz Intel Core i7-2600 processor with 8 GB RAM operated by Windows 7 and similar to the machine used by ExtBI. Table A.13 shows the time in seconds takes different sub-processes of the ETL processes. In this case, we do not run the *External Linking* process. SETL takes more time in data cleansing step comparing to ExtBI. This is because we handle the null value and other unknown values which are simply discarded in ExtBI. On the other hand, our *createTriple()* method takes less time to make the RDF dump although we create 2M more triples than ExtBI shown in Table A.10. Loading RDF triples more time as we load more triples than ExtBI. In summary, SETL takes 5,725.05 seconds where ExtBI takes 6,141.56 seconds, i.e., SETL takes 6.7% less time than ExtBI.

Comparison between SETL and PDI in processing `sdw:Subsidy` To compare the performance of SETL with a Non-semantic data integration tool, we choose PDI. We populate the SDW for `sdw:Subsidy` concept using PDI. Figure A.17 shows the ETL flow for `sdw:Subsidy` built on PDI. The *TBox Extraction*, *R2RML Mapping*, *Extraction* using SPARQL steps are ignored in this flow as those sub-processes cannot be implemented using PDI, i.e., the ETL run with PDI starts by taking the Subsidy dataset transformed into a semantic source. To run the process PDI takes 1,903 seconds. On other hand, SETL takes 2,221 seconds to complete the process which includes all sub-processes starting from *TBox Extraction* to *Load*. If we run the ETL using SETL by skipping he sub-processes skipped by PDI, then SETL takes only 1,644 seconds.

Comparison Metrics	sdw:Subsidy	sub:Company
Triple generate rate (triples/second)	39816.47	8089.80
Data process rate (data size/second)	0.22MB	0.11MB
Data process time (time to process unit data size (in MB))	4.80s	8.70s
Average time for generating a triple	25.20μs	120 μs
Number of triples per input unit data size (in MB)	190986.40	69961.85
Number of triples per output data size (in MB)	6134	5393
Output data size and input data size ratio	31	12

Table A.12: Comparison to process the factual concept and a level of the running example by *Semantic Transformation*.

Tools	Data Cleansing (excluding Spatial join)	Load Ontology	Load Mappings	Dump RDF	Load RDF	Total
ExtBI	603.35	1.01	12.35	4,684.82	840.04	6,141.56
SETL	726	1.05	0	4,208	970	5,725.05

Table A.13: Comparison between SETL and ExtBI [4].

SETL takes 577 seconds for the sub-processes *TBox Extraction*, *R2RML Mapping*, and *Extraction* from semantic source. Thus, SETL is 259 seconds (13.5%) faster than PDI.

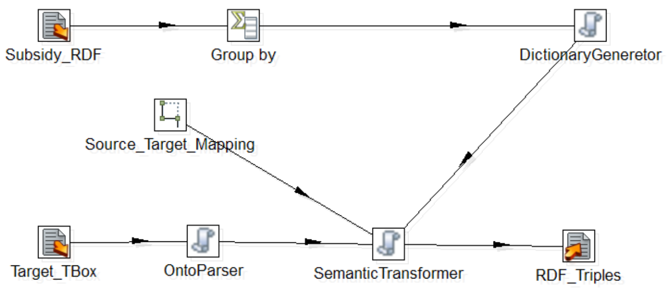


Fig. A.17: ETL flow of PDI to populate `sdw:Subsidy`.

9 Related Work

This section discusses the state of the art in the area of semantic ETL to build a semantic DW and to integrate and publish data as Linked Open Data (LOD) from multiple sources.

Recently, the use of SW technology in data warehouses has become popular. Some approaches have used ontologies as a data integration method for mapping between sources and target whereas others have considered a data warehouse as a repository of ontologies and its instances.

In [49, 50], an OWL ontology is used to link the schemas of semi-structured and structured data to a target DW schema. At first, the schemas of the data sources and the DW are described by a common graph-based model named datastore graph. Then, an application ontology is constructed to describe the semantics of the data sources and the DW, and mapping between them can be done through that ontology. In this way, the heterogeneity issues among the schemas of sources and target have been resolved. In this work, it is assumed that source and target schemas are known beforehand. However, it does not consider the MD view over the DW. In [52], the authors have shown that an ontological approach can automate and minimize the maintenance cost of the evolution of a DW schema. Here, they define every design step of DW using ontology.

A framework for designing semantic DW has been proposed in [36]. Here, the usage of SW languages to integrate distributed DWs and to automate the ETL process of a DW has been discussed. The DW has been considered as a repository of ontologies and semantically annotated data sources. [8] proposes a methodology describing important steps required to create a semantic DW that enables to integrate data from semantic databases, which is composed of an ontology and its instances. A multidimensional (MD) framework has been proposed in [35] to analyze semantic data that are provided by the SW and that are annotated by application and domain ontologies. It allows to build and populate a DW from both the analyst requirements and the knowledge encoded in the ontologies. In [56], authors have proposed a solution for supporting data integration tasks by constructing ontologies from XML and relational sources and integrating the derived ontologies by means of existing ontology alignment and merging techniques. However, ontology alignment technique itself is a difficult and error-prone process [58].

In the past couple of years, publishing data in a machine-readable format has become more popular and (Linked Open Data) LOD has emerged as a way to share such data across Web sources. [60] presents a process to publish governmental data as LOD and [4] discusses how to spatially integrate and publish a Danish Agricultural dataset and a Danish Business dataset as LOD. The approach uses views to cleanse the data and Virtuoso for creating and storing RDF data. To integrate data from heterogeneous sources and publish those data as LOD, a semantic ETL framework is presented in [7] at conceptual level. The approach uses the semantic technology to facilitate the integration process and discusses the use of different tools to accomplish the different steps of the ETL process. To enable warehouse-style analysis over RDF data, some approaches [13, 26] propose the use of a native RDF data warehouse. To analyze an RDF graph, [13] introduced a lens named analytical schema, which is a graph of classes and properties. Each node of the schema represents a set of facts, which can be analyzed by exploring the reachable nodes. To enable OLAP-style analysis over Linked Data (LD), W3C recommend RDF Data Cube Vocabulary (QB vocabulary) to describe the RDF data in MD fashion [15]. However, it has limitations to fully define the traditional MD constructs and it is mostly used to present the statistical data as LD. In [21], authors propose QB4OLAP model by extending the QB model to annotate an existing graph with MD constructs. To enable OLAP over LD datasets which contains numerical values, even if they are not annotated either QB or QB4OLAP, in [30], authors proposed a SPARQL-based ETL framework.

Because of the decentralized structure and dynamic nature of LOD, sometimes it is required to integrate external sources in a federated way. Unlike DW, data federation avoids the need of materializing integrated data into a centralized warehouse. It emphasizes on logical rather than physical integration of data. In [38], authors have proposed a theoretically well-founded approach to the logical federation of OLAP and XML data sources. A set of query processing strategies for executing OLAP-like SPARQL queries over a federation of SPARQL endpoint has been proposed in [25]. However, federation techniques have an additional number of unique challenges on several levels. Participating sources in a federation are autonomous, i.e., they might be unavailable at some point or change their data. So, for query processing, federations introduce delays because of the distribution and there is no guarantee that the data is still available or in the same state as when the federation was created - both schema and data might have changed so that the integration rules might no longer be valid. So, a standard approach is to avoid federations and have a local copy of the data which is the focus of this study. This guarantees that the data will be available and an answer to a user query can be computed efficiently. Moreover, scalability is indeed a problem for analytical queries in federations [25].

As data become outdated due to updates in the sources, using different tools for extraction, validation, and integration becomes very tedious and time consuming. Hence, a framework is necessary that makes it possible to accomplish every step in a single platform. Although the state-of-the-art approaches provide different conceptual frameworks, there is no implemented programmable framework that facilitates the developers by providing required tools, classes, and methods to build ETL process with the goal of either creating a semantic multidimensional DW or publishing the semantically integrated data as LOD. Hence, this paper discusses a semantic ETL framework, SETL, which provides various modules, classes, and methods to extract semantic-aware data, geo-spatial data, and other traditional data, to integrate source data semantically using a target TBox defined with/without MD constructs, and to store the data as semantic data. SETL facilitates developers to build a semantic DW using a single platform instead of using different tools and a manual process.

10 Conclusion

Building a DW integrating internal and external data published in different formats requires semantic integration. Here, we have proposed and developed a programmable framework, named Semantic ETL (SETL) that facilitates users to build a (MD) SDW. SETL uses TBox as an underlying schema to integrate heterogeneous data sources. To annotate the target TBox with MD constructs, it uses the constructs of QB4OLAP model. It allows to process both `semantic` and `Non Semantic Data Sources`. In order to process a `Non Semantic Data Source`, it builds a semantic layer on top of the `Non Semantic Data Source`. Eventually, it stores the data as RDF triples based on the MD semantics encoded in the target TBox. It also allows to link the internal resources with external resources. In order to evaluate our framework, we produced a SDW by integrating a `semantic` and a `Non Semantic Data Sources`. In Section 4, we show that SETL has good performance for all the steps and is faster than the competing tools/solutions.

The model behind a triplestore is an RDF graph. Another issue is how to implement physically the store. We can choose graph, relational or Hadoop, but in this paper, we focus on increasing the abstraction level and thus the user productivity. Therefore, physical or platform-dependent optimizations remain for the future. Our future research also includes proposing a well-defined set of basic semantic ETL operators that can be combined to perform any semantic ETL operations. Besides, developing techniques to explore the Linked Open Data cloud for the related dimensions and level and linking them with the internal ones are also in our consideration.

References

[1] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, and A. Simitsis, “Using Semantic Web Technologies for Exploratory OLAP: A Survey,” *TKDE*, vol. 99, pp. 571–588, 2014.

[2] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J.-N. Mazón, F. Naumann, T. Pedersen, S. B. Rizzi, J. Trujillo, P. Vassiliadis *et al.*, “Fusion Cubes: Towards Self-Service Business Intelligence,” *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 9, no. 2, pp. 66–88, 2013.

[3] A. Abelló, J. Samos, and F. Saltor, “YAM2: A Multidimensional Conceptual Model Extending UML,” *Information Systems*, vol. 31, no. 6, pp. 541–567, 2006.

[4] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *JIST*, 2015.

[5] "ArcPy-ArcGIS Python Library, http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//000v00000001000000."

[6] F. Baader, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge university press, 2003.

[7] S. K. Bansal, "Towards a Semantic Extract-Transform -Load (ETL) Framework for Big Data Integration," in *Big Data*, 2014, pp. 522–529.

[8] L. Bellatreche, S. Khouri, and N. Berkani, "Semantic Data Warehouse Design: From ETL to Deployment à la Carte," in *DASFAA*, 2013, pp. 64–83.

[9] R. Berlanga, O. Romero, A. Simitsis, V. Nebot, T. B. Pedersen, A. Abelló, and M. J. Aramburu, "Semantic Web Technologies for Business Intelligence," in *Business Intelligence Applications and the Web: Models, Systems and Technologies*. IGI global, 2012, pp. 310–339.

[10] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," 2009.

[11] M. Casters, R. Bouman, and J. Van Dongen, *Pentaho Kettle solutions: Building Open Source ETL Solutions with Pentaho Data Integration*. John Wiley & Sons, 2010.

[12] K. Christodoulou, N. W. Paton, and A. A. Fernandes, "Structure Inference for Linked Data Sources using Clustering," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XIX*. Springer, 2015, pp. 1–25.

[13] D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatiş, "RDF Analytics: Lenses over Semantic Graphs," in *WWW*, 2014, pp. 467–478.

[14] "Danish Central Company Registry (CVR) Data, http://cvr.dk/."

[15] R. Cyganiak, D. Reynolds, and J. Tennison, "The RDF Data Cube Vocabulary," *World Wide Web Consortium*, 2014.

[16] "DBpedia, http://dbpedia.org/."

[17] R. P. Deb Nath, K. Hose, and T. B. Pedersen, "Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses," in *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*. ACM, 2015, pp. 15–24.

[18] M. Ehrig, "Ontology Alignment: Bridging the Semantic Gap, volume 4 of Semantic Web And Beyond Computing for Human Experience," 2007.

[19] "Esri, Arcgis. http://www.esri.com/software/arcgis."

[20] L. Etcheverry, S. S. Gomez, and A. Vaisman, "Modeling and Querying Data Cubes on the Semantic Web," *arXiv preprint arXiv:1512.06080*, 2015.

[21] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP," in *DaWak*, 2014, pp. 45–56.

[22] "Ministry of Food, Agriculture and Fisheries of Denmark, http://en.fvm.dk/."

[23] M. Golfarelli and S. Rizzi, *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill, Inc., 2009.

[24] A. Harth, K. Hose, and R. Schenkel, *Linked Data Management*. CRC Press, 2014.

[25] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi, "Processing Aggregate Queries in a Federation of SPARQL Endpoints," in *European Semantic Web Conference*. Springer, 2015, pp. 269–285.

[26] K. A. Jakobsen, A. B. Andersen, K. Hose, and T. B. Pedersen, "Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries," in *COLD*, 2015.

[27] "Apache Jena TDB, https://jena.apache.org/documentation/tdb/."

[28] C. S. Jensen, T. B. Pedersen, and C. Thomsen, "Multidimensional Databases and Data Warehousing," *Synthesis Lectures on Data Management*, vol. 2, no. 1, pp. 1–111, 2010.

[29] R. Kimball and M. Ross, *The Data Warehouse Toolkit: the Complete Guide to Dimensional Modeling*. John Wiley & Sons, 2011.

[30] T. Komamizu, T. Amagasa, and H. Kitagawa, "SPOOL: A SPARQL-Based ETL Framework for OLAP over Linked Data," in *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2015, p. 49.

[31] E. V. Kostylev, J. L. Reutter, and M. Ugarte, "CONSTRUCT Queries in SPARQL," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[32] M. S. Kotoulas, F. van Harmelen, and J. Weaver, "KR and Reasoning on the Semantic Web: Web-Scale Reasoning," in *Handbook of Semantic Web Technologies*. Springer, 2011, pp. 441–466.

[33] J. Li, J. Tang, Y. Li, and Q. Luo, "Rimom: A Dynamic Multistrategy Ontology Alignment Framework," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 8, pp. 1218–1232, 2009.

[34] A. Nasiri, E. Zimányi, and R. Wrembel, "Requirements Engineering for Data Warehouses." in *EDA*, 2015, pp. 49–64.

[35] V. Nebot and R. Berlanga, "Building Data Warehouses with Semantic Web Data," *DSS*, vol. 52, no. 4, pp. 853–868, 2012.

[36] V. Nebot, R. Berlanga, J. M. Pérez, M. J. Aramburu, and T. B. Pedersen, "Multidimensional Integrated Ontologies: a Framework for Designing Semantic Data Warehouses," *JDS*, vol. XIII, pp. 1–36, 2009.

[37] "OWL Web Ontology Language, www.w3.org/TR/owl-ref/."

[38] D. Pedersen, K. Riis, and T. B. Pedersen, "XML-Extended OLAP Querying," in *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*. IEEE, 2002, pp. 195–206.

[39] G. Petasis, V. Karkaletsis, G. Paliouras, A. Krithara, and E. Zavitsanos, "Ontology Population and Enrichment: State of the Art," in *Knowledge-driven multimedia information extraction and ontology evolution*. Springer-Verlag, 2011, pp. 134–166.

[40] "Petl - Extract, Transform and Load (Tables of Data), https://petl.readthedocs.org/en/latest/."

[41] "Pyshp, https://code.google.com/p/pyshp/."

[42] "Python, python.org as of 2015-03-25."

[43] "R2RML: RDB to RDF Mapping Language https://www.w3.org/TR/2012/REC-r2rml-20120927/."

[44] "Rdflib, https://github.com/RDFLib/rdflib."

[45] M. Seddiqui, R. P. D. Nath, M. Aono *et al.*, "An Efficient Metric of Automatic Weight Generation for Properties in Instance Matching Technique," *arXiv preprint arXiv:1502.03556*, 2015.

[46] M. H. Seddiqui, S. Das, I. Ahmed, R. P. D. Nath, and M. Aono, "Augmentation of Ontology Instance Matching by Automatic Weight Generation," in *Information and Communication Technologies (WICT), 2011 World Congress on*. IEEE, 2011, pp. 1390–1395.

[47] J. Sequeda, F. Priyatna, and B. Villazón-Terrazas, "Relational Database to RDF Mapping Patterns." in *WOP*. Citeseer, 2012.

[48] "ESRI, Shapefile Technical Description, INC, 1998."

[49] D. Skoutas and A. Simitsis, "Designing ETL Processes using Semantic Web Technologies," in *DOLAP*, 2006, pp. 67–74.

[50] ——, "Ontology-based Conceptual Design of ETL Processes for both Structured and Semi-structured Data," *IJSWIS*, vol. 3, no. 4, pp. 1–24, 2007.

[51] "SPARQLWrapper, http://rdflib.github.io/sparqlwrapper/."

[52] M. Thenmozhi and K. Vivekanandan, "An Ontological Approach to Handle Multidimensional Schema Evolution for Data Warehouse," *IJDMS*, vol. 6, no. 4, pp. 33–52, 2014.

[53] V. Theodorou, A. Abelló, and W. Lehner, "Quality Measures for ETL Processes," in *DaWaK*, 2014, pp. 9–22.

[54] A. D. H. Thi and B. T. Nguyen, "A Semantic Approach Towards CWM-Based ETL Processes," *Proceedings of I-SEMANTICS*, vol. 8, pp. 58–66, 2008.

[55] C. Thomsen and T. Bach Pedersen, "pygrametl: A Powerful Programming Framework for Extract-Transform-Load Programmers," in *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*. ACM, 2009, pp. 49–56.

[56] 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*. ACM, 2015, pp. 89–98.

[57] G. Tummarello, R. Delbru, and E. Oren, "Sindice. com: Weaving the open linked data," in *ISWC*, 2007.

[58] P. Tyl, "Ontology Matching for Web Services Composition," in *e-Technologies and Networks for Development*. Springer, 2011, pp. 94–103.

[59] A. Vaisman and E. Zimányi, *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[60] B. Villazón-Terrazas, L. M. Vilches-Blázquez, O. Corcho, and A. Gómez-Pérez, "A Methodological Guidelines for Publishing Government Linked Data," in *Linking government data*, 2011, pp. 27–49.

[61] "W3C. Resource Description Framework, http://www.w3.org/RDF/."

[62] "W3C. Resource Description Framework Schema, http://www.w3.org/TR/rdf-schema/."

[63] S. Zhou, Z. Xu, Y. Ni, and H. Zhang, "R2RML Processor for Materializing RDF View of Relational Data: Algorithms and Experiments," in *Web Information System and Application Conference (WISA), 2013 10th*. IEEE, 2013, pp. 450–455.

[64] M. E. Zorrilla, J.-N. Mazón, Ó. Ferrández, I. Garrigós, F. Daniel, and J. Trujillo, *Business Intelligence Applications and the Web: Models, Systems and Technologies*. Business Science Reference, 2012.

Paper B

High-Level ETL for Semantic Data Warehouses

Rudra Pratap Deb Nath, Oscar Romero, Torben Bach Pedersen,
and Katja Hose

The paper is submitted to *Semantic Web Journal*

Abstract

The popularity of the Semantic Web (SW) encourages organizations to organize and publish data using the RDF model. This growth poses new requirements to Business Intelligence (BI) technologies to enable On-Line Analytical Processing (OLAP)-like analysis over RDF data. The incorporation of semantic data into a Data Warehouse (DW) is not supported by the traditional Extract-Transform-Load (ETL) tools because they do not consider semantic issues in the integration process. In this paper, we propose a layer-based integration process and a set of high-level RDF-based ETL constructs required to define, map, extract, process, transform, integrate, update, and load (multidimensional) semantic data. Different to other ETL tools, we automate the ETL data flows by creating metadata at the schema level. Therefore, it relieves ETL developers from the burden of manual mapping at the ETL operation level. We create a prototype, named Semantic ETL Construct ($SETL_{CONSTRUCT}$), based on the innovative ETL constructs proposed here and it is also extended to generate automatic ETL data flows (we call it $SETL_{AUTO}$) . To evaluate $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$, we create a multidimensional semantic DW by integrating a Danish Business Dataset and an EU Subsidy dataset using it and compare it with the previous programmable framework $SETL_{PROG}$ in terms of productivity, development time and performance. The evaluation shows that 1) $SETL_{CONSTRUCT}$ uses 92% fewer Number of Typed Characters (NOTC) than $SETL_{PROG}$, and $SETL_{AUTO}$ further reduces the Number of Used Concepts (NOUC) by another 25%; 2) using $SETL_{CONSTRUCT}$, the development time is almost cut in half compared to $SETL_{PROG}$, and is cut by another 27% using $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ is scalable and has similar performance compared to $SETL_{PROG}$.

The layout has been revised.

1 Introduction

Semantic Web technologies enable adding a semantic layer over the data; thus, the data can be processed and effectively retrieved by both humans and machines. The Linked Data (LD) principles are the set of standard rules to publish and connect data using semantic links [23]. With the growing popularity of the SW and LD, more and more organizations natively manage data using SW standards, such as Resource Description Framework (RDF), RDF-Schema (RDFs), the Web Ontology Language (OWL), etc. [13]. Moreover, one can easily convert data given in another format (database, XML, JSON, etc.) into RDF format using an RDF wrapper. As a result, a lot of semantic datasets are now available in different data portals, such as DataHub[1], Linked Open Data Cloud [2] (LOD), etc. Most SW data provided by international and governmental organizations include facts and figures which give rise to new requirements for Business Intelligence (BI) tools to enable analyses in the style of Online Analytical Processing (OLAP) over those semantic data [30].

OLAP is a well-recognized technology to support decision making by analyzing data integrated from multiple sources. The integrated data are stored in a Data Warehouse (DW), typically structured following the Multidimensional Model (MD) that represents data in terms of facts and dimensions to enable OLAP queries. The integration process for extracting data from different sources, translating it according to the underlying semantics of the DW, and loading it into the DW is known as Extract-Transform-Load (ETL). One way to enable OLAP analysis over semantic data is by extracting those data and translating them according to the DW's format using a traditional ETL process. [40] outlines such a type of semi-automatic method to integrate semantic data into a traditional RDBMS-centric MD DW. However, the process does not maintain all the semantics of data as they are conveying in the semantic sources; hence the integrated data no more follow the SW data principles defined in [24]. The semantics of the data in a semantic data source is defined by 1) using Internationalized Resource Identifier (IRIs) to uniquely identify resources globally, 2) providing common terminology, 3) semantically linking with published information, and 4) providing further knowledge (e.g., logical axioms) to allow reasoning [7].

Therefore, considering semantic issues in the integration process should be emphasized. Moreover, initiatives such as Open Government Data[3] encourage organizations to publish their data using standards and non-proprietary formats [55]. The integration of semantic data into a DW raises the challenges of schema derivation, semantic heterogeneity, semantic annota-

[1]`https://datahub.io/`

[2]`https://lod-cloud.net/`

[3]`https://opengovdata.org/`

tion, linking as well as the schema and data management system over traditional DW technologies and ETL tools. The main drawback of a state-of-the-art Relational Database Management System (RDBMS)-based DW is that it is strictly schema dependent and less flexible to evolving business requirements. To cover new business requirements, every step of the development cycle needs to be updated to cope with the new requirements. This update process is time-consuming as well as costly and is sometimes not adjustable with the current setup of the DW; hence it introduces the need for a novel approach. The limitations of traditional ETL tools to process semantic data sources are: (1) they do not fully support semantic-aware data, (2) they are entirely schema dependent (i.e., cannot handle data expressed without predefined schema), (3) they do not focus on meaningful semantic relationships to integrate data from disparate sources, and (4) they neither support to capture the semantics of data nor support to derive new information by active inference and reasoning on the data.

Semantic Web technologies address the problems described above, as they allow adding semantics at both data and schema level in the integration process and publish data in RDF using the LD principles. On the SW, the RDF model is used to manage and exchange data, and RDFS and OWL are used in combination with the RDF data model to define constraints that data must meet. Moreover, QB [9] and QB4OLAP [15] vocabularies can be used to define data with MD semantics. [37] refers to an MD DW that is semantically annotated both at the schema and data level as a Semantic DW (SDW). An SDW is based on the assumption that the schema can evolve and be extended without affecting the existing structure. Hence, it overcomes the problems triggered by the evolution of an RDBMS-based data warehousing system. In [37], we proposed SETL (throughout this present paper, we call it $SETL_{PROG}$), a programmable semantic ETL framework that semantically integrates both semantic and non-semantic data sources. In $SETL_{PROG}$, an ETL designer has to create hand-code specific modules to deal with semantic data. Thus, there is a lack of a well-defined set of basic ETL constructs that allow developers having a higher level of abstraction and more control in creating their ETL process. In this paper, we propose a strong foundation for an RDF-based semantic integration process and a set of high-level ETL constructs that allows defining, mapping, processing, and integrating semantic data. The unique contributions of this paper are:

1. We structure the integration process into two layers: Definition Layer and Execution Layer. Different to SETL or other ETL tools, here, we propose a new paradigm: the ETL flow transformations are characterized once and for all at the Definition Layer instead of independently within each ETL operation (in the Execution Layer). This is done by generating a mapping file that gives an overall view of the integration

process. This mapping file is our primary metadata source, and it will be fed to the ETL operators, orchestrated in the ETL flow (Execution Layer), to parametrize themselves automatically. Thus, we are unifying the creation of the required metadata to automate the ETL process in the Definition layer. We propose an OWL-based Source-To-target Mapping (S2TMAP) vocabulary to express the source-to-target mappings.

2. We provide a set of high-level ETL constructs for each layer. The Definition Layer includes constructs for target schema[4] definition, source schema derivation, and source-to-target mappings generation. The Execution Layer includes a set of high-level ETL operations for semantic data extraction, cleansing, joining, MD data creation, linking, inferencing, and for dimensional data update.

3. We propose an approach to automate the ETL execution flows based on metadata generated in the Definition Layer.

4. We create a prototype $SETL_{CONSTRUCT}$, based on the innovative ETL constructs proposed here. $SETL_{CONSTRUCT}$ allows creating ETL flows by dragging, dropping, and connecting the ETL operations. In addition, it allows creating ETL data flows automatically (we call it $SETL_{AUTO}$).

5. We perform a comprehensive experimental evaluation by producing an MD SDW that integrates an EU farm Subsidy dataset and a Danish Business dataset. The evaluation shows that $SETL_{CONSTRUCT}$ improves considerably over $SETL_{PROG}$ in terms of productivity, development time, and performance. In summary: 1) $SETL_{CONSTRUCT}$ uses 92% fewer Number of Typed Characters (NOTC) than $SETL_{PROG}$, and $SETL_{AUTO}$ further reduces the Number of Used Concepts (NOUC) by another 25%; 2) using $SETL_{CONSTRUCT}$, the development time is almost cut in half compared to $SETL_{PROG}$, and is cut by another 27% using $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ is scalable and has similar performance compared to $SETL_{PROG}$.

The remainder of the paper is organized as follows. We discuss the terminologies and the notations used throughout the paper in Section 2. Section 3 explains the structure of the datasets we use as use case. Section 4 gives the overview of an integration process. The description of the Definition Layer and Execution Layer constructs are given in Sections 5 and 6, respectively. Section 7 presents the automatic ETL data flow generation process. In Section 8, we create an MD SDW for a use case using $SETL_{CONSTRUCT}$ and compare the process with $SETL_{PROG}$ using different metrics. The previous research related to our study is discussed in Section 9. Finally, we conclude and give pointers to future work in Section 10.

[4]Here, we use the terms “target" and “MD SDW" interchangeably.

2 Preliminary Definitions

In this section, we provide the definitions of the notions and terminologies used throughout the paper.

2.1 RDF Graph

An RDF graph is represented as a set of statements, called RDF triples. The three parts of a triple are subject, predicate, and object, respectively, and a triple represents a relationship between its subject and object described by its predicate. Each triple, in the RDF graph, is represented as $subject \xrightarrow{predicate} object$, where subject and object of the triple are the nodes of the graph, and the label corresponds to the predicate of the triple. Given I, B, and L are the sets of IRIs, blank nodes, and literals, and $(I \cap B \cap L) = \emptyset$, an RDF triple is (s, p, o), where $s \in (I \cup B)$, $p \in I$, and $o \in (I \cup B \cup L)$. An RDF graph G is a set of RDF triples, where $G \subseteq (I \cup B) \times I \times (I \cup B \cup L)$ [22].

2.2 Semantic Data Source

We define a semantic data source as a Knowledge Base (KB) where data are semantically defined. A KB is composed of two components, TBox and ABox. The TBox introduces terminology, the vocabulary of a domain, and the ABox is the assertions of the TBox. The TBox is formally defined as a 3-tuple: $TBox = (C, P, A^O)$, where C, P, and A^O are the sets of concepts, properties, and terminological axioms, respectively [4]. Generally, a concept[5] provides a general framework for a group of instances that have similar properties. A property either relates the instances of concepts or associates the instances of a concept to literals. Terminological axioms are used to describe the domain's concepts, properties, and the relationships and constraints among them. In this paper, we consider a KB as an RDF graph; therefore, the components of the KB are described by a set of RDF triples. Some standard languages such as Resource Description Framework Schema (RDFS) and Web Ontology Language (OWL) provide standard terms to define the formal semantics of a TBox. In RDFS, the core classes `rdfs:Class`, and `rdf:Property` are used to define the concepts and properties of a TBox; one can distinguish between instances and classes by using the `rdf:type` property, express concept and property taxonomies by using `rdfs:subClassOf` and `rdfs:subPropertyOf`, and specify the domain and range of properties by using the `rdfs:domain` and `rdfs:range` properties. Similarly, OWL uses `owl:Class` to define concepts and either `owl:DataTypeProperty` or `owl:ObjectProperty` for properties. In addition to `rdfs:subClassOf`, it uses `owl:equivalentClass` and

[5]In this paper, we use the terms "concept" and "class" interchangeably.

`owl:disjointWith` constructs for class axioms to give additional characteristics of classes. Property axioms define additional characteristics of properties. In addition to supporting RDFS constructs for property axioms, OWL provides `owl:equivalentProeprty` and `owl:inverseOf` to relate different properties, provides `owl:FunctionalProperty` and `owl:InverseFunctionalProperty` for imposing global cardinality constraints, and supports `owl:SymmetricProperty` and `owl:TransitivityProperty` for characterizing the relationship type of properties. As a KB can be defined by either language or both, we generalize the definition of C and P in a TBox T as $C(T) = \{c|\ type(c) \in \mathbb{P}(\{\texttt{rdfs:Class}, \texttt{owl:Class}\})\}$ and $P(T) = \{p|\ type(p) \in \mathbb{P}(\{\texttt{rdf:Property}, \texttt{owl:ObjectProperty}, \texttt{owl:DatatypeProperty}\})\}$, respectively, where $type(x)$ returns the set of classes of x, i.e., (x `rdf:type` $?type(x)$)— it returns the set of the objects of the triples whose subjects and predicates are x and `rdf:type`, respectively— and $\mathbb{P}(s)$ is the power set of s.

2.3 Semantic Data Warehouse

A semantic data warehouse (SDW) is a DW with the semantic annotations. We also considered it as a KB. Since the DW is represented with Multidimensional (MD) model for enabling On-Line Analytical Processing (OLAP) queries, the KB for an SDW needs to be defined with MD semantics. In the MD model, data are viewed in an n-dimensional space, usually known as a data cube, composed of facts (the cells of the cube) and dimensions (the axes of the cube). Therefore, it allows users to analyze data along several dimensions of interest. For example, a user can analyze sales of products according to time and store (dimensions). Facts are the interesting things or processes to be analyzed (e.g., sales of products) and the attributes of the fact are called measures (e.g., quantity, amount of sales), usually represented as numeric values. A dimension is organized into hierarchies, composed of several levels, which permit users to explore and aggregate measures at various levels of detail. For example, the *location* hierarchy (*municipality* $\rightarrow$ *region* $\rightarrow$ *state* $\rightarrow$ *country*) of the *store* dimension allows to aggregate the sales at various levels of detail.

We use the QB4OLAP vocabulary to describe the multidimensional semantics over a KB [15]. QB4OLAP is used to annotate the TBox with MD components and is based on the RDF Data Cube (QB) which is the W3C standard to publish MD data on the Web [12]. The QB is mostly used for analyzing statistical data and does not adequately support OLAP MD constructs. Therefore, in this paper, we choose QB4OLAP. Figure B.1 depicts the ontology of QB4OLAP [55]. The terms prefixed with "qb:" are from the original QB vocabulary, and QB4OLAP terms are prefixed with "qb4o:" and displayed with gray background. Capitalized terms represent OWL classes, and non-capitalized terms represent OWL properties. Capitalized terms in italics represent classes with no instances. The blue-colored square in the figure represents our extension of QB4OLAP ontology.

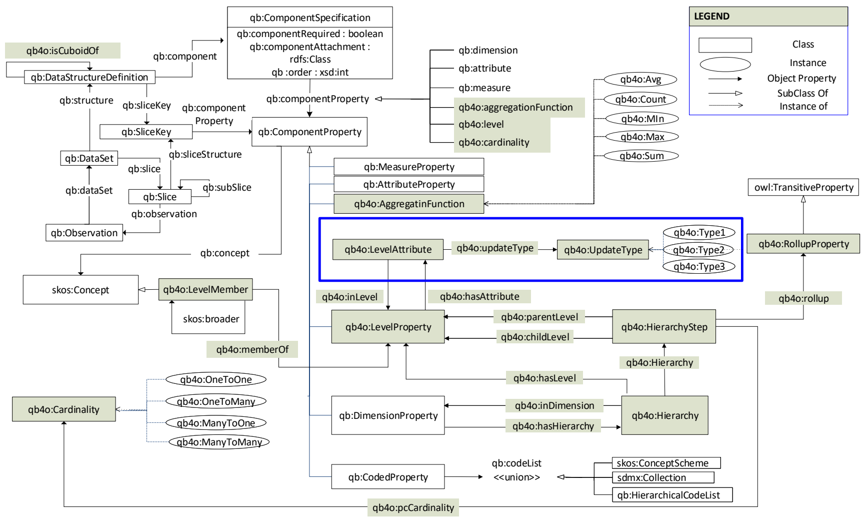


Fig. B.1: QB4OLAP vocabulary.

In QB4OLAP, the concept `qb:DataSet` is used to define a dataset of observations. The structure of the dataset is defined using the concept `qb:DataStructureDefinition`. The structure can be a cube (if it is defined in terms of dimensions and measures) or a cuboid (if it is defined in terms of lower levels of the dimensions and measures). The property `qb4o:isCuboidOf` is used to relate a cuboid to its corresponding cube. To define dimensions, levels and hierarchies, the concepts `qb4o:DimensionProperty`, `qb4o:LevelProperty`, and `qb4o:Hierarchy` are used. A dimension can have one or more hierarchies. The relationship between a dimension and its hierarchies are connected via the `qb4o:hasHierarchy` property or its inverse property `qb4o:inHierarchy`. Conceptually, a level may belong to different hierarchies; therefore, it may have one or more parent levels. Each parent and child pair has a cardinality constraint (e.g., 1-1, n-1, 1-n, and n-n.) [55]. To allow this kind of complex nature, hierarchies in QB4OLAP are defined as a composition of pairs of levels, which are represented using the concept `qb4o:HierarchyStep`. Each hierarchy step (pair) is connected to its component levels using the properties `qb4o:parentLevel` and `qb4o:childLevel`. A rollup relationship between two levels are defined by creating a property which is an instance of the concept `qb4o:RollupProperty`; each hierarchy step is linked to a rollup relationship with the property `qb4o:rollup` and the cardinality constraint of that relationship is connected to the hierarchy step using the `qb4o:pcCardinality` property. A hierarchy step is attached to the hierarchies it belongs to using the property `qb4o:inHierarchy` [15]. The concept `qb4o:LevelAttributes` is used to define attributes of levels. We extend this QB4OLAP ontology (the blue-colored box in the figure) to enable different types of dimension updates (Type 1, Type 2, and Type 3) to accommodate dimension update in an SDW, which are defined by Ralph Kimball in [32]. To define the update-type of a level attribute in the TBox level, we introduce the `qb4o:UpdateType` class whose instances are `qb4o:Type1`, `qb4o:Type1` and `qb4o:Type3`. A level attribute is connected to its update-type by the property `qb4o:updateType`. The level attributes are linked to its corresponding levels using the property `qb4o:hasAttribute`. We extend the definition of C and P of a TBox, T for an SDW as

$$C(T) = \{c|\ type(c) \in \mathbb{P}(\{\texttt{rdfs:Class}, \texttt{owl:Class}, \texttt{qb:DataStructureDef-}\textit{inition}, \texttt{qb:DataSet}, \texttt{qb:DimensionProperty}, \texttt{qb4o:LevelProperty}, \texttt{qb4o:Hierarchy}, \texttt{qb4o:HierarchyStep}\})\} \quad \text{(B.1)}$$

$$P(T) = \{p|\ type(p) \in \mathbb{P}(\{\texttt{rdf:Property}, \texttt{owl:ObjectProperty}, \texttt{owl:DatatypeProperty}, \texttt{qb4o:LevelAttribute}, \texttt{qb:MeassureProperty}, \texttt{qb4o:RollupProperty}\})\} \quad \text{(B.2)}$$

3 A Use Case

We create a Semantic Data Warehouse (SDW) by integrating two data sources, namely, a Danish Agriculture and Business knowledge base and an EU Farm Subsidy dataset. Both data sources are described below.

Description of Danish Agriculture and Business knowledge base The Danish Agriculture and Business knowledge base integrates a Danish Agricultural dataset and a Danish Business dataset. The knowledge base can be queried through the SPARQL endpoint `http://extbi.lab.aau.dk:8080/sparql/`. In our use case, we only use the business related information from this knowledge base and call it the Danish Business dataset (DBD). The relevant portion of the ontology of the knowledge base is illustrated in Figure B.2. Generally, in an ontology, a concept provides a general description of the properties and behavior for the similar type of resources; an object property relates among the instances of concepts; a data type property is used to associate the instances of a concept to literals.

We start the description from the concept `bus:Owner`. This concept contains information about the owners of companies, the type of the owner-

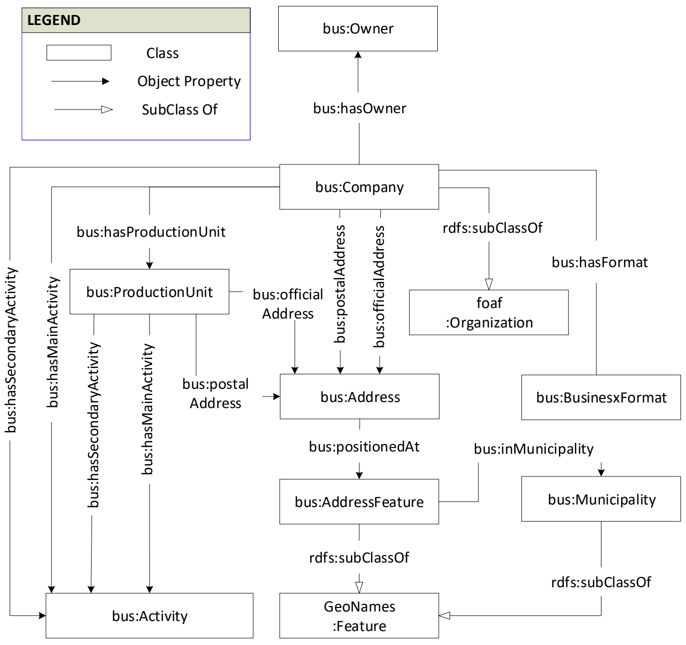


Fig. B.2: The ontology of the Danish Business dataset. Due to the large number of datatype properties, they are not included.

ship, and the start date of the company ownership. A company is connected to its owner through the `bus:hasOwner` property. The `bus:Company` concept is related to `bus:BusinessFormat` and `bus:ProductionUnit` through the `bus:hasProductionUnit` and `bus:hasFormat` properties. Companies and their production units have one or more main and secondary activities. Each company and each production unit has a postal address and an official address. Each address is positioned at an address feature, which is in turn contained within a particular municipality.

Description of the EU subsidy dataset Every year, the European Union provides subsidies to the farms of its member countries. We collect EU Farm subsidies for Denmark from `https://data.farmsubsidy.org/Old/`. The dataset contains two MS Access database tables: Recipient and Payment. The Recipient table contains the information of recipients who receive the subsidies, and the Payment table contains the amount of subsidy given to the recipients. We create a semantic version of the dataset using $SETL_{PROG}$ framework [36]. We call it the Subsidy dataset. At first, we manually define an ontology, to describe the schema of the dataset, and the mappings between the ontology and datatabase tables. Then, we populate the ontology with the

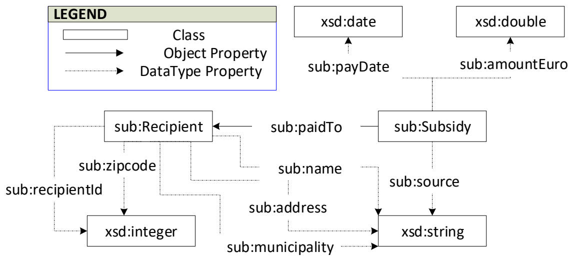


Fig. B.3: The ontology of the Subsidy dataset. Due to the large number of datatype properties, all are not included.

instances of the source database files. Figure B.3 shows the ontology of the Subsidy dataset.

Example 1. Listing B.1 shows the example instances of (`bus:Company`) the Danish Business dataset and (`sub:Recipient` and `sub:Subsidy`) the EU Subsidy dataset.

Listing B.1: Example instances of the DBD and the Subsidy dataset.

```
### Business Dataset
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bus: <http://extbi.lab.aau.dk/ontology/business/>
PREFIX company: <http://extbi.lab.aau.dk/ontology/
                                      business/Company#>
PREFIX activity: <http://extbi.lab.aau.dk/ontology/
                                      business/Activity#>
PREFIX businessType: <http://extbi.lab.aau.dk/ontology/
                                      business/BusinessType#>
PREFIX owner: <http://extbi.lab.aau.dk/ontology/
                                      business/Owner#>
## Examples of bus:Company instances
company:10058996 rdf:type bus:Company;
   bus:companyId 10058996;
   bus:name "Regerupgard v/Kim Jonni Larsen";
   bus:mainActivity activity:11100;
   bus:secondaryActivity activity:682040;
   bus:hasFormat businessType:Enkeltmandsvirksomhed:
   bus:hasOwner  owner:4000175029_10058996;
   bus:ownerName "Kim Jonni Larsen";
   bus:officialaddress "Valsomaglevej 117, Ringsted".
company:10165164 rdf:type bus:Company;
   bus:companyId 10165164;
   bus:name "Idomlund 1 Vindmollelaug I/S";
   bus:mainActivity activity:351100;
   bus:hasFormat businessType:Interessentskab;
   bus:hasOwner  owner:4000170495_10165164;
   bus:ownerName "Anders Kristian Kristensen";
   bus:officialaddress "Donskaervej 31,Vemb".
---------------------------------------------------------
---------------------------------------------------------
### Subsidy Dataset
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sub: <http://extbi.lab.aau.dk/ontology/subsidy/>
PREFIX recipient: <http://extbi.lab.aau.dk/ontology/
                                    subsidy/Recipient#>
PREFIX subsidy: <http://extbi.lab.aau.dk/ontology/
                                    subsidy/Subsidy#>
## Example of sub:Recipient instances.
recipient:291894 rdf:type sub:Recipient;
   sub:name "Kristian Kristensen";
   sub:address "Donskaervej 31,Vemb";
   sub:municipality "Holstebro";
   sub:recipientID 291894;
   sub:zipcode 7570.
## Example of sub:Subsidy instances.
```

```
subsidy:10615413 rdf:type sub:Subsidy;
    sub:paidTo recipient:291894;
    sub:amountEuro "8928.31";
    sub:payDate "2010-05-25";
```

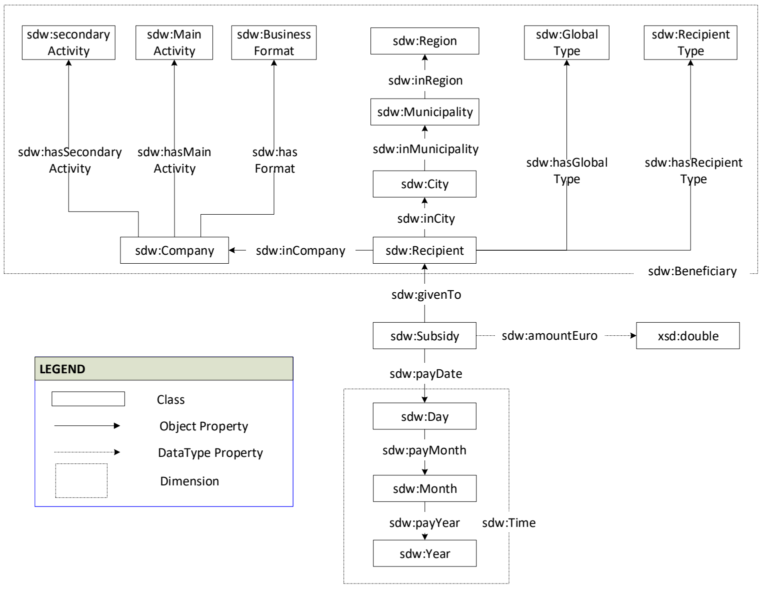


Fig. B.4: The ontology of the MD SDW. Due to the large number, data properties of the dimensions are not shown.

Description of the Semantic Data Warehouse Our goal is to develop an MD Semantic Data Warehouse (SDW) by integrating the Subsidy and the DBD datasets. The `sub:Recipient` concept in the Subsidy dataset contains the information of recipient id, name, address, etc. From `bus:Company` in the DBD, we can extract information of an owner of a company who received the EU farm subsidies. Therefore, we can integrate both DBD and Subsidy datasets. The ontology of the MD SDW to store EU subsidy information corresponding to the Danish companies is shown in Figure B.4, where the concept `sdw:Subsidy` represents the facts of the SDW. The SDW has two dimensions, namely `sdw:Benificiary` and `sdw:Time`. The dimensions are shown by a box with dotted-line in Figure B.4. Here, each level of the dimensions are represented by a concept, and the connections among levels are represented through object properties.

4 Overview of the Integration Process

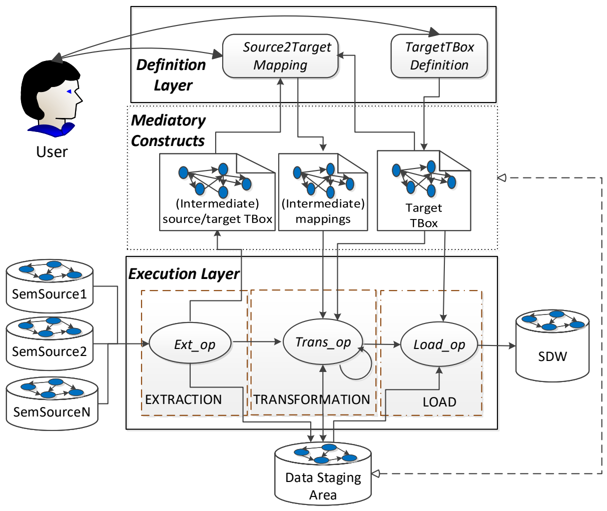


Fig. B.5: The overall semantic data integration process. Here, the round-corner rectangle, data stores, dotted boxes, ellipses, and arrows indicate the tasks, semantic data sources, the phases of the ETL process, ETL operations and flow directions.

In this paper, we assume that all given data sources are semantically defined and the goal is to develop an SDW. The first step of building an SDW is to design its TBox. There are two approaches to design the TBox of an SDW, namely source-driven and demand-driven [54]. In the former, the SDW's TBox is obtained by analyzing the sources. Here, ontology alignment techniques [35] can be used to semi-automatically define the SDW. Then, designers can identify the multidimensional constructs from the integrated TBox and annotate them with the QB4OLAP vocabulary. In the latter, SDW designers first identify and analyze the needs of business users as well as decision makers, and based on those requirements, they define the target TBox with multidimensional semantics using the QB4OLAP vocabulary. How to design a Target TBox is orthogonal to our approach. Here, we merely provide an interface to facilitate creating it regardless of whatever approach was used to design it.

After creating the TBox of the SDW, the next step is to create the ETL process. ETL is the backbone process by which data are entered into the SDW and the main focus of this paper. The ETL process is composed of three phases: extraction, transformation, and load. A phase is a sub-process of the ETL which provides a meaningful output that can be fed to the next phase as an input. Each phase includes a set of operations. The extraction operations extract data from the data sources and make it available for further processing as intermediate results. The transformation operations are applied on intermediate results, while the load operations load the transformed data into the DW. The intermediate results can be either materialized in a data staging area or kept in memory. A data staging area (temporary) persists data for cleansing, transforming, and future use. It may also prevents the loss of extracted or transformed data in case of the failure of the loading process.

As we want to separate the metadata needed to create ETL flows from their execution, we introduce a two-layered integration process, see Figure B.5. In the Definition Layer, a single source of metadata truth is defined. This includes: the target SDW, semantic representation of the source schemas, and a source to target mapping file. Relevantly, the metadata created represents the ETL flow at the schema level. In the Execution Layer, ETL flows based on high-level operations are created. This layer executes the ETL flows for instances (i.e., at the data level). Importantly, each ETL operation will be fed the metadata created to parameterize themselves automatically. Additionally, the Execution Layer automatically checks the correctness of the flow created, by checking the compatibility of the output and input of consecutive operators. Overall the data integration process requires the following four steps in the detailed order.

1. Defining the target TBox with MD semantics using QB and QB4OLAP constructs. In addition, the TBox can be enriched with RDFS/OWL classes and properties. However, we do not validate the correctness of the added semantics beyond the MD model. This step is done at the Definition Layer.

2. Extracting source TBoxes from the given sources. This step is done at the Definition Layer.

3. Creating mappings among source and target constructs to characterize ETL flows. The created mappings are stored using the S2TMAP vocabulary proposed. This step is also done at the Definition Layer.

4. Populating the ABox of the SDW implementing ETL flows. This step is done at the Execution Layer.

Figure B.5 illustrates the whole integration process and how the constructs of each layer communicate with each other. Here, we introduce two types of constructs: tasks and operations. On the one hand, a task requires developer interactions with the interface of the system to produce an output. Intuitively, one may consider the tasks output as the required metadata to automate operations. On the other hand, from the given metadata, an operation produces an output. The Definition Layer consists of two tasks (*TargetTBoxDefinition* and *SourceToTargetMapping*) and one operation (*TBoxExtraction*). These two tasks respectively address the first and third steps of the integration process mentioned above, while the *TBoxExtraction* operation addresses the second step. This is the only operation shared by both layers (see the input of *SourceToTargetMapping* in Figure B.5). Therefore, the Definition Layer creates three types of metadata: target TBox (created by *TargetTBoxDefinition*), source TBoxes (create by *TBoxExtraction*), and source-to-target mappings (created by *SourceToTargetMapping*). The Execution Layer covers the fourth step of the integration process and includes a set of operations to create data flows from the sources to the target. Figure B.5 shows constructs (i.e., the Mediatory Constructs) used by the ETL task/operations to communicate between them. These mediatory constructs store the required metadata created to automate the process. In the figure, Ext_{op}, $Trans_{op}$, and $Load_{op}$ are the set of extraction, transformation, and load operations used to create the flows at the instance level. The ETL data flows created in the Execution Layer are automatically validated by checking the compatibility of the operations. Precisely, if an operation O_1's output is accepted by O_2, then we say O_2 is compatible with O_1 and express it as $O_1 \rightarrow O_2$.

Since traditional ETL tools (e.g., PDI) do not have ETL operations supporting the creation of an SDW, we propose a set of ETL operations for each phase of the ETL to process semantic data sources. The operations are categorized based on their functionality. Table B.1 summarizes each operation with its corresponding category name, compatible successors, and its objectives. Next, we present the details of each construct of the layers presented in Figure B.5.

5 The Definition Layer

This layer contains two tasks (*TargetTBoxDefinition* and *Source2TargetMapping*) and one operation `TBoxExtraction`. The semantics of the tasks are described below.

TargetTBoxDefinition The objective of this task is to define a target TBox with MD semantics. There are two main components of the MD structure: dimensions and cubes. To formally define the schema of these components, we use the notation from [9] with some modifications.

Definition 1. A **dimension schema** can formally be defined as a 5-tuple $(D_{name}, \mathcal{L}, \rightarrow, \mathcal{H}, \mathcal{F}_{\mathcal{R}})$ where (a) D_{name} is the name of the dimension; (b) $\mathcal{L}$ is a set of level tuples (L_{name}, L_A) such that L_{name} is the name of a level and L_A is the set of attributes describing the level L_{name}. There is a unique bottom level (the finest granularity level) L_b, and unique top level (the coarsest one)

Operation Category	**Operation Name**	**Compatible Successors**	**Objectives**
Extraction	GraphExtractor	GraphExtractor, TBoxExtraction, TransformationOnLiteral, JoinTransformation, LevelMemberGenerator, ObservationGenerator, DataChangeDetector, UpdateLevel, Loader	It retrieves an RDF graph in terms of RDF triples from semantic data sources.
Transformation	TBoxExtraction		It derives a TBox from a given ABox.
	TransformationOnLiteral	TransformationOnLiteral, JoinTransformation, LevelMemberGenerator, ObservationGenerator, Loader	It transforms the source data according to the expressions described in the source-to-target mapping.
	JoinTransformation	TransformationOnLiteral, JoinTransformation, LevelMemberGenerator, ObservationGenerator, Loader	It joins two data sources and transforms the data according to the expressions described in the source-to-target mapping.
	LevelMemberGenerator (QB4OLAP construct)	Loader	It populates levels of the target with the input data.
	ObservationGenerator (QB4OLAP construct)	Loader	It populates facts of the target with the input data.
	DataChangeDetector	LevelMemberGenerator, UpdateLevel	It returns the differences between the new source dataset and the old one.
	UpdateLevel	Loader	It reflects the changes occurred in the source data to the target level.
	MaterializeInference	Loader	It enriches the SDW by materializing the inferred triples.
	ExternalLinking	Loader	It links internal resources with external KBs.
Load	Loader		It loads the data into the SDW.

Table B.1: Summary of the ETL operations.

denoted L_{All}, such that $(L_{All}, \emptyset) \in \mathcal{L}$; (c) $\rightarrow$ is a strict partial order on $\mathcal{L}$. The poset $(\mathcal{L}, \rightarrow)$ can be represented as a directed graph where each node represents an aggregation level $L \in \mathcal{L}$, and every pair of levels (L_i, L_j) in the poset are represented by a directed edge from the finer granularity level L_i to the coarser granularity level L_j, which means that L_i rolls up to L_j or L_j drills down to L_i. Each distinct path between the L_b and L_{All} is called a hierarchy; (d) $\mathcal{H}$ is a set of hierarchy tuples (H_{name}, H_L) where H_{name} is the name of a hierarchy and $H_L \subseteq \mathcal{L}$ is the set of levels composing the hierarchy; (e) $\mathcal{F}_{\mathcal{R}}$ is a set of roll-up relations. A roll-up relation is a mathematical relation that relates between the members of two adjacent levels in the hierarchy, i.e, $RUP_{(L_i,L_j)} \in \mathcal{F}_{\mathcal{R}} = \{(lm_i, lm_j) \mid lm_i \in L_i \; and \; lm_j \in L_j \; and \; (L_i, L_j) \in (\mathcal{L}, \rightarrow)\}$.

Example 2. Figure B.4 shows that our use case MD SDW has two dimensions: `sdw:Time` and `sdw:Ben`
`eficiary`. The dimension schema of `sdw:Time` is formally defined as follows:

1. $D_{name} =$ `sdw:Time`,

2. $\mathcal{L} = \{(\texttt{sdw:Day}, \langle \texttt{sdw:dayId}, \texttt{sdw:dayName} \rangle), (\texttt{sdw:Month}, \langle \texttt{sdw:monthId}, \texttt{sdw:monthName} \rangle), (\texttt{sdw:Year}, \langle \texttt{sdw:yearId}, \texttt{sdw:yearName} \rangle)\}$,

3. $(\mathcal{L}, \rightarrow) = \{(\texttt{sdw:Day}, \texttt{sdw:Month}), (\texttt{sdw:Month}, \texttt{sdw:Year}), (\texttt{sdw:Year}, \texttt{sdw:All})\}$,

4. $\mathcal{H} = \{\texttt{sdw:TimeHierarchy}, \{\texttt{sdw:Day}, \texttt{sdw:Month}, \texttt{sdw:Year}, \texttt{sdw:All}\}\}$, and

5. $\mathcal{F}_{\mathcal{R}} = \{$
$RUP_{(\texttt{sdw:Day}, \texttt{sdw:Month})} = \texttt{sdw:payMonth}$,
$RUP_{(\texttt{sdw:Month}, \texttt{sdw:Year})} = \texttt{sdw:payYear}$,
$RUP_{(\texttt{sdw:Year}, \texttt{sdw:All})} = \texttt{sdw:payAll}\}$.

Definition 2. A **cube schema** is a tuple $(C_{name}, \mathcal{D}_{l_b}, \mathcal{M}, \mathcal{F}_{\mathcal{A}})$, where (a) C_{name} is the name of the cube; (b) $\mathcal{D}_{l_b}$ is a finite set of bottom levels of dimensions, with $|\mathcal{D}_{l_b}| = n$, corresponding to n bottom levels of n dimension schemas different from each other; (c) $\mathcal{M}$ is a finite set of attributes called measures, and each measure $m \in \mathcal{M}$ has an associated domain $Dom(m)$; and (d) $\mathcal{F}_{\mathcal{A}}$ is a mathematical relation that relates each measure to one or more aggregate function in $\mathcal{A} = \{SUM, MAX, AVG, MIN, COUNT.\}$, i.e., $\mathcal{F}_{\mathcal{A}} \subseteq \mathcal{M} \times \mathcal{A}$.

Example 3. The cube schema of our use case, shown in Figure B.4, is formally defined as follows:

1. $C_{name} =$ `sdw:Subsidy`,

2. $D_L = \{\texttt{sdw:Day}, \texttt{sdw:Recipient}\}$,

3. $\mathcal{M} = \{$`sdw:amounteuro`$\}$, and

4. $\mathcal{F}_{\mathcal{A}} = \{($`sdw:amounteuro`$, SUM), ($`sdw:amounteuro`$, AVG)\})$.

In Section 2.3, we discussed how the QB4OLAP vocabulary is used to define different constructs of an SDW. Listing B.2 represents the `sdw:Time` dimension and `sdw:Subsidy` cube in QB4OLAP.

Listing B.2: QB4OLAP representation of `sdw:Time` dimension and `sdw:Subsidy` cube.

```
PREFIX sdw: <http://extbi.lab.aau.dk/ontology/sdw/>
PREFIX rdf:      http://www.w3.org/1999/02/22-rdf-syntax-ns#
PREFIX rdfs:     http://www.w3.org/2000/01/rdf-schema#
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX qb4o: <http://purl.org/qb4olap/cubes#>

## Time Dimension
sdw:Time rdf:type qb:DimensionProperty;
         rdfs:label "Time Dimension";
         qb4o:hasHierarcy sdw:TimeHierarchy.

# Dimension Hierarchies
sdw:TimeHierarchy rdf:type qb4o:Hierarchy;
         rdfs:label "Time Hierarchy";
         qb4o:inDimension sdw:Time;
         qb4o:hasLevel  sdw:Day, sdw:Month, sdw:Year.

# Hierarchy levels
sdw:Day rdf:type qb4o:LevelProperty;
        rdfs:label "Day Level";
        qb4o:hasAttribute sdw:dayId, sdw:dayName.
sdw:Month rdf:type qb4o:LevelProperty;
        rdfs:label "Month Level";
        qb4o:hasAttribute sdw:monthId, sdw:monthName.
sdw:Year rdf:type qb4o:LevelProperty;
        rdfs:label "Year Level";
        qb4o:hasAttribute sdw:yearId, sdw:yearName.
sdw:All rdf:type qb4o:LevelProperty;
        rdfs:label "ALL".

# Level attributes
sdw:dayId rdf:type qb4o:LevelAttribute;
       rdfs:label "day ID";
       qb4o:updateType qb4o:Type2;
       rdfs:range xsd:String.
sdw:monthId rdf:type qb4o:LevelAttribute;
       rdfs:label "Month ID";
       qb4o:updateType qb4o:Type2;
       rdfs:range xsd:String.
sdw:yearId rdf:type qb4o:LevelAttribute;
       rdfs:label "year ID";
       qb4o:updateType qb4o:Type2;
       rdfs:range xsd:String.
sdw:dayName rdf:type qb4o:LevelAttribute;
       rdfs:label "day Name";
       qb4o:updateType qb4o:Type1;
       rdfs:range xsd:String.
sdw:monthName rdf:type qb4o:LevelAttribute;
       rdfs:label "Month Name";
       qb4o:updateType qb4o:Type1;
       rdfs:range xsd:String.
sdw:yearName rdf:type qb4o:LevelAttribute;
       rdfs:label "year Name";
       qb4o:updateType qb4o:Type1;
       rdfs:range xsd:String.

#rollup relations
sdw:payMonth rdf:type qb4o:RollupProperty.
sdw:payYear rdf:type qb4o:RollupProperty.
sdw:payAll rdf:type qb4o:RollupProperty.

# Hierarchy Steps
_:ht1 rdf:type qb4o:HierarchyStep;
      qb4o:inHierarchy sdw:TimeHierarchy;
      qb4o:childLevel sdw:Day;
      qb4o:parentLevel sdw:Month;
      qb4o:pcCardinality qb4o:OneToMany;
      qb4o:rollup sdw:payMonth.
_:ht2 rdf:type qb4o:HierarchyStep;
```

```
      qb4o:inHierarchy sdw:TimeHierarchy;
      qb4o:childLevel sdw:Month;
      qb4o:parentLevel sdw:Year;
      qb4o:pcCardinality qb4o:OneToMany;
      qb4o:rollup sdw:payYear.
_:ht2 rdf:type qb4o:HierarchyStep;
      qb4o:inHierarchy sdw:TimeHierarchy;
      qb4o:childLevel sdw:Year;
      qb4o:parentLevel sdw:All;
      qb4o:pcCardinality qb4o:OneToMany;
      qb4o:rollup sdw:payAll.

## Subsidy Cube
sdw:amounteuro rdf:type qb:MeasureProperty;
      rdfs:label "subsidy amount"; rdfs:range xsd:Double.
sdw:SubsidyStructure rdf:type qb:DataStructureDefinition;
      qb:component[qb4o:level sdw:Recipient];
      qb:component[qb4o:level sdw:Day];
      qb:component[qb:measure sdw:amounteuro;
           qb4o:aggregateFunction qb4o:sum, qb4o:avg].
# Subsidy Dataset
sdw:SubsidyMD rdf:type qb:Dataset;
      rdfs:label "Subsidy dataset";
      qb:structure sdw:SubsidyStructure;
```

TBoxExtraction After defining a target TBox, the next step is to extract source TBoxes. Typically, in a semantic source, the TBox and ABox of the source are provided. Therefore, no external extraction task/operation is required. However, sometimes, the source contains only the ABox, and no TBox. In that scenario, an extraction process is required to derive a TBox from the ABox. We formally define the process as follows.

Definition 3. The TBox extraction operation from a given ABox, *ABox* is defined as $f_{ABox2TBox}(ABox) \rightarrow TBox$. The derived *TBox* is defined in terms of the following TBox constructs: a set of concepts C, a set of concept taxonomies H, a set of properties P, and the sets of property domains D and ranges R. The following steps describe the process to derive each TBox element for *TBox*.

1. C: By checking the unique objects of the triples in *ABox* where `rdf:type` is used as a predicate, C is identified.

2. H: The taxonomies among concepts are identified by checking the instances they share among themselves. Let C_1 and C_2 be two concepts. One of the following taxonomic relationships holds between them: 1) if C_1 contains all instances of C_2, then we say C_2 is a subclass of C_1 (C_2 `rdfs:subClassOf` C_1); 2) if they do not share any instances, they are disjoint (C_1 `owl:disjointWith` C_2); and 3) if C_1 and C_2 are both a subclass of each other, then they are equivalent (C_1 `owl:equivalentClass` C_2).

3. P, D, R: By checking the unique predicates of the triples, P is derived. A property $p \in P$ can relate resources with either resources or literals. If the objects of the triples where p is used as predicates are IRIs, then p is an object property; the domain of p is the set of the types of the

subjects of those triples, and the range of p is the types of the objects of those triples. If the objects of the triples where p is used as predicates are literals, then p is a datatype property; the domain of p is the set of types the subjects of those triples, and the range is the set of data types of the literals.

SourceToTargetMapping Once the target and source TBoxes are defined, the next task is to characterize the ETL flows at the Definition Layer by creating source-to-target mappings. Because of the heterogeneous nature of source data, mappings among sources and the target should be done at the TBox level. In principle, mappings are constructed between sources and the target; however, since mappings can get very complicated, we allow to create a sequence of *SourceToTargetMapping* definitions whose subsequent input is generated by the preceding operation. The communication between these operations is by means of a materialized intermediate mapping definition and its meant to facilitate the creation of very complex flows (i.e., mappings) between source and target.

A source-to-target mapping is constructed between a source and a target TBox, and it consists of a set of concept-mappings. A concept-mapping defines i) a relationship (equivalence, subsumption, supersumption, or join) between a source and the corresponding target concept, ii) which source instances are mapped (either all or a subset defined by a filter condition), iii) the rule to create the IRI for target concept instances, iv) the source and target ABox locations, v) the common properties between two concepts if their relationship is join, vi) the sequence of ETL operations required to process the concept-mapping, and vii) a set of property-mappings for each property having the target concept as a domain. A property-mapping defines how a target property is mapped from either a source property or an expression over properties. Definition 4 formally defines a source-to-target mapping.

Definition 4. Let T_S and T_T be a source TBox and a target TBox. We formally define a source-to-target mapping as a set of concept-mappings, wherein each concept-mapping is defined with a 10-tuple formalizing the elements discussed above (i-vii):

$SourceToTargetMapping(T_S, T_T) = \{(c_s, relation, c_t, loc_{c_s}, loc_{c_t}, mapIns, p_{map}, tin_{iri}, p_{com}, op)\}$.

The semantics of each concept-mapping tuple is given below.

- $c_s \in \mathcal{C}(T_S)$ and $e_{t_i} \in \mathcal{C}(T_T)$ are a source and a target concept respectively, where $\mathcal{C}(T)$ defined in Equation B.1.

- $relation \in \{\equiv, \sqsubseteq, \sqsupseteq, \bowtie\}$ represents the relationship between the source and target concept. The relationship can be either *equivalence* ($e_{s_i} \equiv e_{t_i}$), *supersumption* ($e_{s_i} \sqsupseteq e_{t_i}$), *subsumption* ($e_{s_i} \sqsubseteq e_{t_i}$), or *join* ($e_{s_i} \bowtie e_{t_i}$). A join relationship exists between two sources when there is a need to populate a target element (a level, a (QB) dataset, or a concept) from multiple sources. Since a concept-mapping represents a binary relationship, to join n sources, an ETL process requires $n-1$ join concept-mappings. A concept-mapping with a join relationship requires two sources (i.e., the concept-mapping source and target concepts) as input and updates the target concept according to the join result. Thus, for multi-way joins, the output of a concept-mapping is a source concept of the next concept-mapping. Note that, a join relationship can be natural join ($\bowtie$), right-outer join ($\rtimes$), or left-outer join ($\ltimes$).

- loc_{c_s} and loc_{c_t} are the locations of source and target concept ABoxes.
- $mapIns \in (\{All\} \cup FilterCondition)$ indicates which instances of the source concept to use to populate the target concept; it can either be all source instances or a subset of source instances defined by a filter condition.
- $p_{map} = \{(p_{c_s}, p_{c_t})\}$ is a set of property-mappings across the properties of c_s and c_t. p_{c_s} can be a property from $property(c_s)$ or an expression over the elements of $exp(property(c_s) \cup property(c_t))$ and p_{c_t} is a property from $property(c_t)$. Here, $property(c)$ returns the union of the set of properties which are connected with concept c either using the `rdfs:domain` or `qb4olap:inLevel` properties, or the set of rollup properties related to c. An expression allows to apply arithmetic operations and/or some high-level functions for manipulating strings, data types, numbers, dates defined as standard SPARQL functions in [21] over the properties.
- tin_{iri} indicates how the unique IRIs of target instances are generated. The IRIs can be either the same as the source instances, or created using a property of c_s, or using an expression from $exp(property(c_s))$, or in an incremental way.
- $p_{com} = \{(scom_i, tcom_i) | scom_i \in (property(e_{s_i}), tcom_i \in property(e_{t_i}))\}$ is a set of common property pairs. In each pair, the first element is a source property and the second one is a target property. p_{com} is required when the relationship between the source and target concept is a join.
- op is an ETL operation or a sequence of ETL operations required to implements the mapping element in the ABox level. When op is a sequence of ETL operations, the location of the input ABox location for the first operation in the sequence is loc_{c_s}; the subsequent operations in the sequence take the output of their preceding operation as the input ABox. This generation of intermediate results is automatically handled by the automatic ETL generation process described in Section 7.

In principle, an SDW is populated from multiple sources, and a source-to-target ETL flow requires more than one intermediate concept-mapping definitions. Therefore, a complete ETL process requires a set of source-to-target mappings. We say a mapping file is a set of source-to-target mappings. Definition 5 formally defines a mapping file.

Definition 5. $Mapping\ file = \bigcup_{i \in S} SourceToTargetMapping(T_i, T_j)$, where S is the set of all sources and intermediate results schemas, and j the set of all intermediate results and the target schemas.

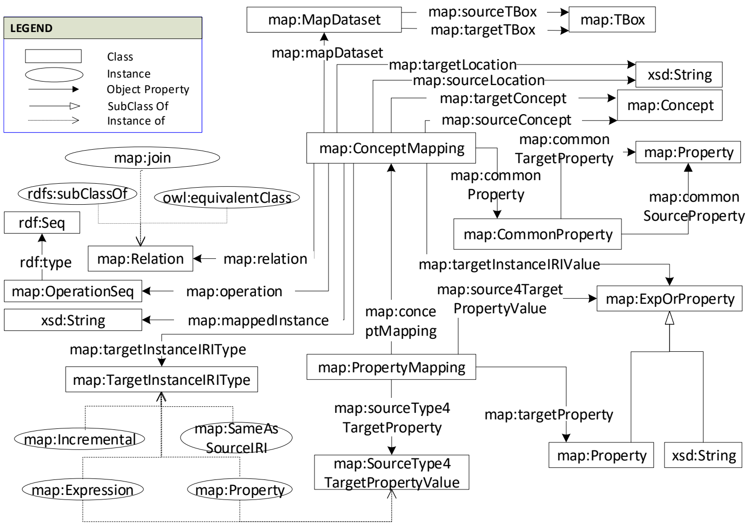


Fig. B.6: Graphical overview of key terms and their relationship to the S2TMAP vocabulary.

To implement the source-to-target mappings formally defined above, we propose an OWL-based mapping vocabulary: Source-to-Target Mapping (S2TMAP). Figure B.6 depicts the mapping vocabulary. A mapping between a source and a target TBox is represented as an instance of the class `map:MapDataset`. The source and target TBoxes are defined by instantiating `map:TBox`, and these TBoxes are connected to the mapping dataset using the properties `map:sourceTBox` and `map:targetTBox`, respectively. A concept-mapping (an instance of `map:ConceptMapping`) is used to map between a source and a target concepts (instances of `map:Concept`). A concept-mapping is connected to a mapping dataset using the `map:mapDataset` property. The source and target ABox locations of the concept-mapping are defined through the `map:sourceLocation` and `map:targetLocation` properties. The relationship between the concepts can be either *`rdfs:subClassOf`*, or *`map:join`*, or *`owl:e-`*

quivalentClass, and it is connected to the concept-mapping via the `map:relation` property. The sequence of ETL operations, required to implement the concept-mapping at the ABox level, is defined through an RDF sequence. To express joins, the source and target concept in a concept-mapping represent the concepts to be joined, and the join result is stored in the target concept as an intermediate result. In a concept-mapping, we, via `map:commonProperty`, identify the join attributes with a blank node (instance of `map:CommonProperty`) that has, in turn, two properties identifying the source and target join attributes; i.e., `map:commonSourceProperty` and `map:commonTargetProperty`. Since a join can be defined on multiple attributes, we may have multiple blank node definitions. The type of target instance IRIs is stated using the property `map:TargetInstanceIRIType`. If the type is either *map:Property* or *map:Expression*, then the property or expression, to be used to generate the IRIs, is given by `map:targetInstanceIRIvalue`.

To map at the property stage, a property-mapping (an instance of `map:PropertyMapping`) is used. The association between a property-mapping and a concept-mapping is defined by `map:conceptMapping`. The target property of the property-mapping is stated using `map:targetProperty`, and that target property can be mapped with either a source property or an expression. The source type of target property is determined through `map:sourceType4TargetProperty` property, and the value is defined by `map:source4TargetPropertyValue`.

Example 4. Listing B.3 represents a snippet of the mapping file of our use case MD SDW and the source datasets. In the Execution Layer, we show how the different segments of this mapping file will be used by each ETL operation.

Listing B.3: An S2TMAP representation of the mapping file of our use case.

```
PREFIX onto: <http://extbi.lab.aau.dk/ontology/>
PREFIX bus:  <http://extbi.lab.aau.dk/ontology/business/>
PREFIX sub:  <http://extbi.lab.aau.dk/ontology/subsidy/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX map:  <http://extbi.lab.aau.dk/ontology/s2tmap/>
PREFIX sdw:  <http://extbi.lab.aau.dk/sdw>
PREFIX : <http://extbi.lab.aau.dk/ontology/s2map/example#>
## MapDataset
:mapDataset1 rdf:type map:Dataset;
  rdfs:label "Map-dataset for business and subsidy ontology";
  map:sourceTBox "/map/businessTBox.ttl";
  map:targetTBox "/map/subsidyTBox.ttl".
:mapDataset2 rdf:type map:Dataset;
  rdfs:label "Map-dataset for subsidy and subsidyMD ontology";
  map:sourceTBox "/map/subsidyTBox.ttl";
  map:targetTBox "/map/subsidyMDTBox.ttl".
##ConceptMapping: Joining  Recipient and Company
:Recipient_Company rdf:type map:ConceptMapping;
  rdfs:label "join-transformation between
          bus:Company and sub:Recipient";
  map:mapDataset :mapDataset1;
  map:sourceConcept bus:Company;
  map:targetConcept sub:Recipient;
  map:sourceLocation "/map/dbd.nt";
  map:targetLocation "/map/subsidy.nt";
  map:relation map:rightOuterjoin;
  map:mappedInstance "All";
  map:targetInstanceIRIUniqueValueType map:SameAsSourceIRI;
  map:operation _:opSeq;
  map:commonProperty _:cp1, _:cp2.
_:opSeq  rdf:type rdf:Seq;
  rdf:_1 map:joinTransformation.
_:cp1 map:sourceCommonProperty bus:ownerName;
  map:targetCommonProperty sub:name.
_:cp2 map:sourceCommonProperty bus:officialAddress;
  map:targetCommonProperty sub:address.

#concept-mapping: Populating the sdw:Recipient level
:Recipient_RecipientMD   rdf:type map:ConceptMapping;
  rdfs:label "Level member generation";
  map:mapDataset :mapDataset2;
  map:sourceConcept sub:Recipient;
  map:targetConcept sdw:Recipient;
  map:sourceLocation "/map/subsidy.nt";
  map:targetLocation "/map/sdw";
  map:relation owl:equivalentClass;
  map:mappedInstance "All";
  map:targetInstanceIRIValueType map:Property;
  map:targetInstanceIRIValue sub:recipientID;
  map:operation _:opSeq1.
_:opSeq1 rdf:type rdf:Seq;
  rdf:_1 map:LevelMemberGenerator;
  rdf:_2 map:Loader.
#concept-mapping: Populating the cube dataset
:Subsidy_SubsidyMD rdf:type map:ConceptMapping;
  rdfs:label "Observation generation";
  map:mapDataset :mapDataset2;
  map:sourceConcept sub:Subsidy;
  map:targetConcept sdw:SubsidyMD;
  map:sourceLocation "/map/subsidy.nt";
  map:targetLocation "/map/sdw";
  map:relation owl:equivalentClass;
  map:mappedInstance "All";
  map:targetInstanceIRIUniqueValueType map:Incremental;
  map:operation _:opSeq2.
_:opSeq2 rdf:type rdf:Seq;
  rdf:_1 map:GraphExtractor;
  rdf:_2 map:TransformationOnLiteral;
  rdf:_3 map:ObservationGenerator;
  rdf:_4 map:Loader.
## property-mapping under :Recipient_Company
:companyID_companyID rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for companyID";
  map:conceptMapping :Recipient_Company;
  map:targetProperty sub:companyId;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue bus:companyId.
:businessType_businessType rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for business type";
  map:conceptMapping :Recipient_Company;
  map:targetProperty sub:businessType;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue bus:hasFormat.
:address_city rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for city";
  map:conceptMapping :Recipient_Company;
  map:targetProperty sub:cityId;
  map:sourceType4TargetPropertyValue map:Expression;
  map:source4TargetPropertyValue STRAFTER(sub:address,",").
:name_name rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for name";
  map:conceptMapping :Recipient_Company;
  map:targetProperty sub:name;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:name.
# property-mappings under :Recipient_RecipientMD
:companyId_company rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for companyId";
  map:conceptMapping :Recipient_RecipientMD;
  map:targetProperty sdw:hasCompany;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:companyId;
:cityId_city rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for cityId";
  map:conceptMapping :Recipient_RecipientMD;
  map:targetProperty sdw:inCity;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:city;
:name_name rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for name";
  map:conceptMapping :Recipient_RecipientMD;
  map:targetProperty sdw:name;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:name
# property-mappings under :Subsidy_SubsidyMD
:Recipient_recipientId rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for recipient in sdw:Subsidy";
  map:conceptMapping :Subsidy_SubsidyMD;
  map:targetProperty sdw:Recipient;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:paidTo.
:hasPayDate_Day rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for Day of sdw:SubsidyMD";
  map:conceptMapping :Subsidy_SubsidyMD;
  map:targetProperty sdw:Day;
  map:sourceType4TargetPropertyValue map:Expression;
  map:source4TargetPropertyValue
  "CONCAT(STR(DAY(sub:payDate)),"/",
   STR(MONTH(sub:payDate)),"/",STR(YEAR(sub:payDate)))".
:amountEuro_amountEuro rdf:type map:PropertyMapping;
  rdfs:label "property-mapping for amountEuro measure";
  map:conceptMapping :Subsidy_SubsidyMD;
  map:targetProperty sdw:amountEuro;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:amountEuro.
```

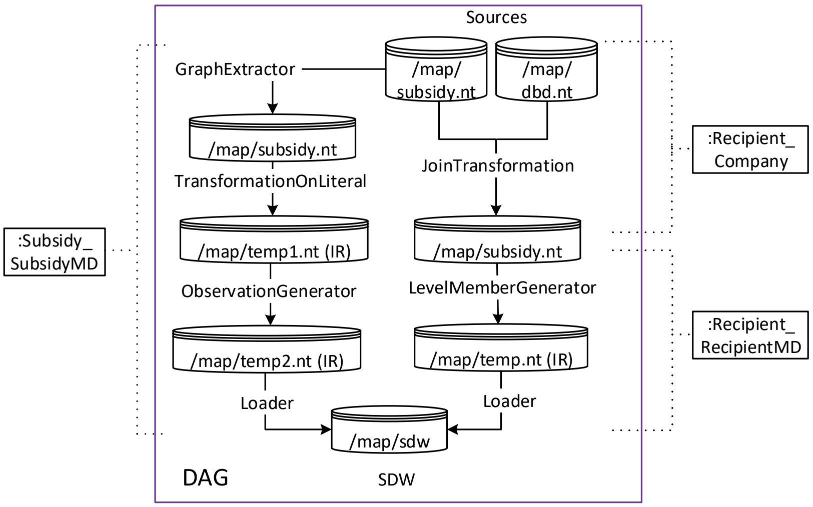


Fig. B.7: The conceptual presentation of Listing B.3.

A mapping file is a Directed Acyclic Graph (DAG). Figure B.7 shows the DAG representation of Listing B.3. In this figure, the sources, intermediate results and the SDW are denoted as nodes of the DAG and edges of the DAG represent the operations. The dotted-lines shows the parts of the ETL covered by concept-mappings, represented by a rectangle.

6 The Execution Layer

In the Execution Layer, ETL data flows are constructed to populate an MD SDW. Table B.1 summarizes the set of ETL operations. In the following, we present an overview of each operation category-wise. Here, we give the intuitions of the ETL operations in terms of definitions and examples. To reduce the complexity and length of the paper, we place the semantics of the ETL operations in Section A. In this section, we present the signature of each operation. That is, the main inputs required to execute the operation and the produced outcome. As an ETL data flow is a sequence of operations and an operation in the sequence communicates with its preceding and subsequent operations by means of materialized intermediate results, all the operations presented here have side-effects.

Developers can use either of the two following options: (i) The recommended option is that given a TBox construct *aConstruct* (a concept, a level, or a QB dataset) and a mapping file *aMappings* generated in the Definition Layer, the automatic ETL execution flow generation process will automatically extract the values for operations' parameters from *aMappings* (see Section 7 for a detailed explanation of the automatic ETL execution flow generation process). (ii) They can manually set input parameters at the operation level. In this section, we follow the following order to present each operation: 1) we first give a high-level definition of the operation; 2) then, we define how the automatic ETL execution flow generation process parameterizes the operation from the mapping file, and 3) finally, we present an example showing how developers can manually parameterize the operation. When introducing the operations and referring to its automatic parametrization, we will refer to *aMappings* and *aConstruct* as defined here. Note that each operation is bound to exactly one concept-mapping at a time in the mapping file (discussed in Section 7).

6.1 Extraction Operations

Extraction is process of data retrieval from the sources. Here, we introduce two extraction operations for semantic sources: (i) *GraphExtractor* - to form/extract an RDF graph from a semantic source and (ii) *TBoxExtraction* - to derive a TBox from a semantic source as described in Section 5. As such, *TBoxExtraction* is the only operation in the Execution Layer generating metadata stored in the Mediatory Constructs (see Figure B.5).

GraphExtractor(Q, G, outputPattern, tABox) Since the data integration process proposed in this paper uses RDF as the canonical model, we extract/generate RDF triples from the sources with this operation. *GraphExtractor* is functionally equivalent to SPARQL CONSTRUCT queries [34].

If the ETL execution flow is generated automatically, the automatic ETL execution flow generation process first identifies the concept-mapping *cm* from *aMappings* where *aConstruct* appears (i.e., in *aMapping*, $cm \xrightarrow{map:targetConcept}$

aConstruct) and the operation to process *cm* is *GraphExtractor* (i.e., *GraphExtractor* is an element in the operation sequence defined by the `map:operation` property). Then, it parametrizes *GraphExtractor* as follows: 1) *G* is the location of the source ABox defined by the property `map:sourceLocation` of *cm*; 2) *Q* and *outputPattern* are internally built based on the `map:mappedInstance` property, which defines whether all instances (defined by "All") or a subset of the instances (defined by a filter condition) will be extracted; 3) *tABox* is the location of target ABox defined by the property `map:targetLocation`. A developer can also manually set the parameters. From the given inputs, *GraphExtractor* operation performs a pattern matching operation over the given source *G* (i.e., finds a map function binding the variables in query pattern *Q* to constants in *G*), and then, for each binding, it creates triples according to the triple templates in *outputPattern*. Finally, the operation stores the output in the path *tABox*.

Example 5. Listing B.1 shows the example instances of the Danish Business Dataset (DBD). To extract all instances of `bus:Company` from the dataset, we use the *GraphExtractor(Q, G, outputPattern, tABox)* operation, where

1. *Q=((?ins,`rdf:type`,`bus:Company`) AND (?ins,?p,?v))*,[6]
2. *G*="/map/dbd.ttl",
3. *outputPattern= (?ins,?p,?v)*,
4. *tABox*="/map/com.ttl"[7].

Listing B.4 shows the output of this operation.

Listing B.4: Example of *GraphExtractor*.

```
company:10058996 rdf:type bus:Company;
    bus:name "Regerupgard v/Kim Jonni Larsen";
    bus:mainActivity activity:11100;
    bus:secondaryActivity activity:682040;
    bus:hasFormat businessType:Enkeltmandsvirksomhed:
    bus:hasOwner  owner:4000175029_10058996;
    bus:ownerName "Kim Jonni Larsen";
    bus:address "Valsomaglevej 117, Ringsted".
company:10165164 rdf:type bus:Company;
    bus:name "Idomlund 1 Vindmollelaug I/S";
    bus:mainActivity activity:351100;
    bus:hasFormat businessType:Interessentskab;
    bus:hasOwner  owner:4000170495_10165164;
    bus:ownerName "Anders Kristian Kristensen";
    bus:address "Donskaervej 31,Vemb".
```

TBoxExtraction is already described in Section 5, therefore, we do not repeat it here.

[6]To make it easily distinguishable, here, we use comma instead of space to separate the components of a triple pattern and an RDF triple.

[7]We present the examples in Turtle format for reducing the space and better understanding. In practice, our system prefers N-Triples format to support scalability.

6.2 Transformation Operations

Transformation operations transform the extracted data according to the semantics of the SDW. Here, we define the following semantic-aware ETL transformation operations: *TransformationOnLiteral, JoinTransformation, LevelMemberGenerator, ObservationGenerator, ChangedDataCapture, UpdateLevel, External linking,* and *MaterializeInference*. The following describe each operation.

TransformationOnLiteral(sConstruct, tConstruct, sTBox, sABox propertyMappings, tABox) As described in the *SourceToTargetMapping* task, a property (in a property-mapping) of a target construct (i.e., a level, a QB dataset, or a concept) can be mapped to either a source concept property or an expression over the source properties. An expression allows arithmetic operations, datatype (string, number, and date) conversion and processing functions, and group functions (sum, avg, max, min, count) as defined in SPARQL [21]. This operation generates the instances of the target construct by resolving the source expressions mapped to its properties.

If the ETL execution flow is generated automatically, the automatic ETL execution flow generation process first identifies the concept-mapping *cm* from *aMappings*, where *aConstruct* appears and the operation to process *cm* is *TransformationOnLiteral*. Then, the process parametrizes *TransformationOnLiteral* as follows: 1) *sConstruct* and *tConstruct* are defined by `map:sourceConcept` and `map:targetConcept`; 2) *sTBox* is the target TBox of *cm*'s map-dataset, defined by the property `map:sourceTBox`; 3) *sABox* is the location of the source ABox defined by `map:sourceLocation`; 3) *propertyMappings* is the set of property-mappings defined under *cm*; 4) *tABox* is the location of the target ABox defined by `map:targetLocation`. A developer can also manually set the parameters. From the given inputs, this operation transforms (or directly returns) the *sABox* triple objects according to the expressions (defined through `map:source4TargetPropertyValue`) in *propertyMappings* and stores the triples in *tABox*. This operation first creates a SPARQL SELECT query based on the expressions defined in *propertyMappings*, and then, on top of the SELECT query, it forms a SPARQL CONSTRUCT query to generate the transformed ABox for *tConstruct*.

Example 6. Listing B.5 (lines 16-19) shows the transformed instances after applying the operation *TransformationOnLiteral(sConstruct, tConstruct, sTBox, sABox, PropertyMappings, tABox)*, where

1. *sConstruct* = *tConstruct*=`sub:Subsidy`,
2. *sTBox*="/map/subsidyTBox.ttl",
3. *sABox*= source instances of `sub:Subsidy` (lines 47-50 in Listing B.1),

4. *propertyMappings* = lines 2-14 in Listing B.5,
5. *tABox*="/map/temp1.ttl".

Listing B.5: Example of *TransformationOnLiteral*.

```
 1 ## Property-mappings input
 2 :hasPayDate_Day rdf:type map:PropertyMapping;
 3  map:targetProperty sub:hasPayDate;
 4  map:sourceType4TargetPropertyValue map:Expression;
 5  map:source4TargetPropertyValue "CONCAT(STR(DAY(hasPayDate)),
 6  "/", STR(MONTH(hasPayDate)),"/",STR(YEAR(hasPayDate)))".
 7 :Recipient_recipientId rdf:type map:PropertyMapping;
 8  map:targetProperty sub:hasRecipient;
 9  map:sourceType4TargetPropertyValue map:Property;
10  map:source4TargetPropertyValue sub:hasRecipient.
11 :amountEuro_amountEuro rdf:type map:PropertyMapping;
12  map:targetProperty sub:amountEuro;
13  map:sourceType4TargetPropertyValue map:Expression;
14 map:source4TargetPropertyValue "xsd:integer(sub:amountEuro)".
15 ## sub:Subsidy instances after TransformationOnLiteral.
16 subsidy:10615413 rdf:type sub:Subsidy;
17  sub:hasRecipient recipient:291894;
18  sub:amountEuro 8928;
19  sub:hasPayData "25/25/2010".
```

JoinTransformation(sConstruct, tConstruct, sTBox, tTBox, sABox, tABox, comProperty, propertyMappings) A TBox construct (a concept, a level, or a QB dataset) can be populated from multiple sources. Therefore, an operation is necessary to join and transform data coming from different sources. Two constructs of the same or different sources can only be joined if they share some common properties. This operation joins a source and a target constructs based on their common properties and produce the instances of the target construct by resolving the source expressions mapped to target properties. To join *n* sources, an ETL process requires *n-1 JoinTransformation* operations.

If the ETL execution flow is generated automatically, the automatic ETL execution flow generation process first identifies the concept-mapping *cm* from *aMappings*, where *aConstruct* appears and the operation to process *cm* is *JoinTransformation*. Then it parameterizes *JoinTransformation* as follows: 1) *sConstruct* and *tConstruct* are defined by the `map:sourceConcept` and `map:targetConcept` properties; 2) *sTBox* and *tTBox* are the source and target TBox of *cm*'s map-dataset, defined by `map:sourceTBox` and `map:targetTBox`; 3) *sABox* and *tABox* are defined by the `map:sourceLocation` and `map:targetLocation` properties; 4) *comProperty* is defined by `map:commonProperty`; 5) *propertyMappings* is the set of property-mappings defined under *cm*.

A developer can also manually set the parameters. Once it is parameterized, *JoinTransformation* joins two constructs based on *comProperty*, transforms their data based on the expressions (specified through `map:source4TargetPropertyValue`) defined in *propertyMappings*, and updates *tABox* based on the join result. It creates a SPARQL SELECT query joining two constructs using either `AND` or `OPT` features, and on top of that query, it forms a SPARQL CONSTRUCT query to generate the transformed *tABox*.

Example 7. The recipients in `sdw:Recipient` need to be enriched with their company information available in the Danish Business dataset. Therefore, a join operation is necessary between `sub:Recipient` and `bus:Company`. The concept-mapping of this join is described in Listing B.3 at lines 19-37. They are joined by two concept properties: recipient names and their addresses (lines 31, 34-37). We join and transform `bus:Company` and `sub:Recipient` using *JoinTransformation(sConstruct, tConstruct, sTBox, tTBox, sABox, tABox, comProperty, propertyMappings)*, where

1. *sConstruct*= `bus:Company`,
2. *tConstruct*= `sub:Recipient`,
3. *sTBox*= "/map/businessTBox.ttl",
4. *tTBox*= "/map/subsidyTBox.ttl",
5. *sABox*= source instances of `bus:Company` (lines 13-29 in Listing B.1),
6. *tABox*= source instances of `sub:Recipient` (lines 40-45 in Listing B.1),
7. *comProperty* = lines 31, 34-37 in Listing B.3,
8. *propertyMappings* = lines 73-96 in Listing B.3.

Listing B.6 shows the output of the *joinTransformation* operation.

Listing B.6: Example of *JoinTransformation*.

```
## Example of sub:Recipient instances.
recipient:291894 rdf:type sub:Recipient;
    sub:name "Kristian Kristensen";
    sub:cityId "Vemb";
    sub:companyId company:10165164;
    sub:businessType businessType:Interessentskab.
```

LevelMemberGenerator(sConstruct, level, sTBox, sABox, tTBox, iriValue, iriGraph, propertyMappings, tABox) In QB4OLAP, dimensional data are physically stored in levels. A level member, in an SDW, is described by a unique IRI and its semantically linked properties (i.e., level attributes and rollup properties). This operation generates data for a dimension schema defined in Definition 1.

If the ETL execution flow is generated automatically, the automatic process first identifies the concept-mapping *cm* from *aMappings*, where *aConstruct* appears and the operation to process *cm* is *LevelMemberGenerator*. Then it parameterizes *LevelMemberGenerator* as follows: 1) *sConstruct* is the source construct defined by `map:sourceConcept`; 2) *level* is the target level[8] defined

[8]A level is termed as a level property in QB4OLAP, therefore, throughout this paper, we use both the term "level" and "level property" interchangeably.

by the `map:targetConcept` property; 3) *sTBox* and *tTBox* are the source and target TBoxes of *cm*'s map dataset, defined by the properties `map:sourceTBox` and `map:targetTBox`; 4) *sABox* is the source ABox defined by the property `map:sourceLocation`; 5) *iriValue* is a rule[9] to create IRIs for the level members and it is defined defined by the `map:TargetInstanceIriValue` property; 6) *iriGraph* is the IRI graph[10] within which to look up IRIs, given by the developer in the automatic ETL flow generation process; 7) *propertyMappings* is the set of property-mappings defined under *cm*; 8) *tABox* is the target ABox location defined by `map:targetLocation`.

A developer can also manually set the paramenters. Once it is parameterized, *LevelMemberGenerator* operation generates QB4OLAP-compliant triples for the level members of *level* based on the semantics encoded in *tTBox* and stores them in *tABox*.

Example 8. Listing B.3 shows a concept-mapping (lines 40-54) describing how to populate `sdw:Recipient` from `sub:Recipient`. Listing B.7 shows the level member created by the *LevelMemberGenerator(level, tTBox, sABox, iriValue, iriGraph, propertyMappings, tABox)* operation, where

1. *sConstruct*= `sub:Recipient`,
2. *level*= `sdw:Recipient`,
3. *sTBox*= "/map/subsidyTBox.ttl",
4. *sABox*= "/map/subsidy.ttl", shown in Example 7,
5. *tTBox* = "/map/subsidyMDTBox.ttl",
6. *iriValue* = `sub:recipientID`,
7. *iriGraph* = "/map/provGraph.nt",
8. *propertyMappings*= lines 98-115 in Listing B.3,
9. *tABox*="/map/temp.ttl".

Listing B.7: Example of *LevelMemberGenerator*.

```
PREFIX sdw: <http://extbi.lab.aau.dk/ontology/sdw/>
PREFIX recipient: <http://extbi.lab.aau.dk/ontology
                                /sdw/Recipient#>
PREFIX company: <http://extbi.lab.aau.dk/ontology
                                /sdw/Company#>
PREFIX city: <http://extbi.lab.aau.dk/ontology
                                /sdw/City#>
## Example of a recipient level member.
```

[9] A rule can be either a source property, an expression or incremental, as described in Section 5.

[10] The IRI graph is an RDF graph that keeps a triple for each resource in the SDW with their corresponding source IRI.

```
recipient:291894 rdf:type qb4o:LevelMember;
                 qb4o:memberOf sdw:Recipient.
                 sdw:name "Kristian Kristensen";
                 sdw:inCity city:Vemb;
                 sdw:hasCompany company:10165164.
```

ObservationGenerator(sConstruct, dataset, sTBox, sABox, tTBox, iriValue, iriGraph, propertyMappings, tABox) In QB4OLAP, an observation represents a fact. A fact is uniquely identified by an IRI, which is defined by a combination of several members from different levels and contains values for different measure properties. This operation generates data for a cube schema defined in Definition 2.

If the ETL execution flow is generated automatically, the way used by the automatic ETL execution flow generation process to extract values for the parameters of *ObservationGenerator* from *aMappings* is analogous to *LevelMemberGenerator*. Developers can also manually set the parameters. Once it is parameterized, the operation generates QB4OLAP-compliant triples for observations of the QB dataset*dataset* based on the semantics encoded in *tTBox* and stores them in *tABox*.

Example 9. Listing B.8 (lines 21-25) shows a QB4OL-AP-compliant observation create by the *ObservationGenerator(sConstruct, dataset, sTBox, sABox, tTBox, iriValue, iriGraph, propertyMappings, tABox)* operation, where

1. *sConstruct*=`sub:Subsidy`,
2. *dataset*= `sdw:SubsidyMD`,
3. *sTBox*="/map/subsidyTBox.ttl"
4. *sABox*= "/map/subsidy.ttl", shown in Example 6,
5. *tTBox* = "/map/subsidyMDTBox.ttl", shown in Listing B.2,
6. *iriValue* = "Incremental",
7. *iriGraph* = "/map/provGraph.nt",
8. *propertyMappings*= lines 8-19 in Listing B.8,
9. *tABox*= "/map/temp2.ttl".

Listing B.8: Example of *ObservationGenerator*.

```
1 PREFIX sdw: <http://extbi.lab.aau.dk/ontology/sdw/>
2 PREFIX subsidy: <http://extbi.lab.aau.dk/ontology
3                                /sdw/Subsidy#>
4 PREFIX recipient: <http://extbi.lab.aau.dk/ontology
5                                /sdw/Recipient#>
6 PREFIX day: <http://extbi.lab.aau.dk/ontology/sdw/Day#>
7 ## Property-Mappings
8 :recipientId_Recipient rdf:type map:PropertyMapping;
9   map:targetProperty sdw:Recipient;
```

```
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:hasRecipient.
:hasPayDate_Day rdf:type map:PropertyMapping;
  map:targetProperty sdw:Day;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:hasPaydate.
:amountEuro_amountEuro rdf:type map:PropertyMapping;
  map:targetProperty sdw:amountEuro;
  map:sourceType4TargetPropertyValue map:Property;
  map:source4TargetPropertyValue sub:amountEuro.
## Example of observations
subsidy:_01 rdf:type qb4o:Observation;
  inDataset sdw:SubsidyMD;
  sdw:hasRecipient recipient:291894;
  sdw:amountEuro "8928.00";
  sdw:hasPayData day:25/25/2010.
```

ChangedDataCapture(nABox, oABox, flag) In a real-world scenario changes occur in a semantic source both at the schema and instance level. Therefore, an SDW needs to take action based on the changed schema and instances. The adaption of the SDW TBox with the changes of source schemas is an analytical task and requires the involvement of domain experts, therefore, it is out of the scope of this paper. Here, only the changes at the instance level are considered.

If the ETL execution flow is generated automatically, the automatic process first identifies the concept-mapping *cm* from *aMappings*, where *aConstruct* appears and the operation to process *cm* is *ChangedDataCapture*. Then, it takes `map:sourceLocation` and `map:targetLocation` for *nABox* (new dimensional instances in a source) and *oABox* (old dimensional instances in a source), respectively, to parameterize this operations. *flag* depends on the next operation in the operation sequence.

Developers can also manually set the parameters. From the given inputs, *ChangedDataCapture* outputs either 1) a set of new instances (in the case of SDW evolution, i.e., *flag=0*) or 2) a set of updated triples —the existing triples changed over time— (in the case of SDW update, i.e., *flag=1*) and overwrites *oABox*. This is done by means of the set difference operation. This operation must then be connected to either *LevelMemberGenerator* to create the new level members or *updateLevel* (described below) to reflect the changes in the existing level members.

Example 10. Suppose Listing B.6 is the old ABox of `sub:Recipient` and the new ABox is at lines 2-10 in Listing B.9. This operation outputs either 1) the new instance set (in this case, lines 12-15 in the listing) or 2) the updated triples (in this case, line 17).

Listing B.9: Example of *ChangedDataCapture*.

```
1 ## New snapshot of sub:Recipient instances.
2 recipient:291894 rdf:type sub:Recipient;
3                  sub:name "Kristian Jensen";
4                  sub:cityId "Vemb";
5                  sub:companyId company:10165164;
6                  businessType:Interessentskab.
7 recipient:301894 rdf:type sub:Recipient;
8                  sub:name "Jack";
```

```
                  sub:cityId "Aalborg";
                  sub:companyId company:100000.
## New instances to be inserted
recipient:301894 rdf:type sub:Recipient;
                  sub:name "Jack";
                  sub:cityId "Aalborg";
                  sub:companyId company:100000.
## Update triples
recipient:291894 sub:name "Kristian Jensen".
```

UpdateLevel(level, updatedTriples, sABox tTBox, tABox, propertyMappings, iriGraph) Based on the triples updated in the source ABox *sABox* for the level *level* (generated by *ChangedDataCapture*), this operation updates the target ABox *tABox* to reflect the changes in the SDW according to the semantics encoded in the target TBox *tTBox* and *level* property-mappings *propertyMappings*. Here, we address three update types (Type1-update, Type2-update, and Type3-update), defined by Ralph Kimball in [32] for a traditional DW, in an SDW environment. The update types are defined in *tTBox* for each level attribute of *level* (as discussed in Section 2.3). As we consider only instance level updates, only the objects of the source updated triples are updated. To reflect a source updated triple in *level*, the level member using the triple to describe itself, will be updated. In short, the level member is updated in the following three ways: 1) A Type1-update simply overwrites the old object with the current object. 2) A Type2-update creates a new version for the level member (i.e., it keeps the previous version and creates a new updated one). It adds the validity interval for both versions. Further, if the level member is a member of an upper level in the hierarchy of a dimension, the changes are propagated downward in the hierarchy, too. 3) A Type3-update overwrites the old object with the new one. Besides, it adds an additional triple for each changed object to keep the old object. The subject of the additional triple is the instance IRI, the object of the triple is the old object, and the predicate is *concat(oldPredicate, "oldValue")*.

If the ETL execution flow is generated automatically, this operation first identifies the concept-mapping *cm* from *aMappings*, where *aConstruct* appears and the operation to process *cm* is *UpdateLevel*. Then it parameterizes *LevelMemberGenerator* as follows: 1) *level* is defined by the `map:targetConcept`; 2) sABox is old source data; 3) *updatedTriples* is the source location defined by `map:sourceLocation`; 4) *tTBox* is the target TBox of *cm*'s map-dataset, defined by the property `map:targetTBox`; 5) *tABox* is defined by `map:targetLocation`; 6) *propertyMappings* is the set of property-mappings defined under *cm*; 7) *iriGraph* is given by developer in the automatic ETL flow generation process.

Example 11. Listing B.10 describes how different types of updates work by considering two members of the `sdw:Recipient` level (lines 2-11). As the name of the second member (lines 7-11) is changed to "Kristian Jensen" from "Kristian Kristensen", as found in Listing B.9. A Type1-update simply overwrites the existing name (line 20). A Type2-update creates a new version (lines 39-46). Both old and new versions contain validity interval (lines 35-37) and (lines 44-46). A Type3-Update overwrites the old name (line 56) and adds a new triple to keep the old name (line 57).

Listing B.10: Example of different types of updates.

```
 1 ## sdw:Recipient Level
 2 recipient:762921 rdf:type qb4o:LevelMember;
 3                  qb4o:member sdw:Recipient;
 4                  sdw:name "R. Nielsen";
 5                  sdw:cityId city:Lokken;
 6                  sdw:hasCompany company:10165164.
 7 recipient:291894 rdf:type qb4o:LevelMember;
 8                  qb4o:memberOf sdw:Recipient.
 9                  sdw:name "Kristian Kristensen";
10                  sdw:cityId city:Vemb;
11                  sdw:hasCompany company:10165164.
12 ## After type1 update
13 recipient:762921 rdf:type qb4o:LevelMember;
14                  qb4o:member sdw:Recipient;
15                  sdw:name "R. Nielsen";
16                  sdw:cityId city:Lokken;
17                  sdw:hasCompany company:10165164.
18 recipient:291894 rdf:type qb4o:LevelMember;
19                  qb4o:memberOf sdw:Recipient.
20                  sdw:name "Kristian Jensen";
21                  sdw:cityId city:Vemb;
22                  sdw:hasCompany company:10165164.
23 ##After type2 update
24 recipient:762921 rdf:type qb4o:LevelMember;
25                  qb4o:member sdw:Recipient;
26                  sdw:name "R. Nielsen";
27                  sdw:cityId city:Lokken;
28                  sdw:hasCompany company:10165164.
29
30 recipient:291894 rdf:type qb4o:LevelMember;
31                  qb4o:memberOf sdw:Recipient.
32                  sdw:name "Kristian Kristensen";
33                  sdw:cityId city:Vemb;
34                  sdw:hasCompany company:10165164.
35                  sdw:fromDate ‘‘0000-00-00";
36                  sdw:toDate ‘‘2017-09-25";
37                  sdw:status ‘‘Expired".
38
39 recipient:291894\_2017\_09\_26 rdf:type qb4o:LevelMember;
40                             qb4o:memberOf sdw:Recipient.
41                             sdw:name "Kristian Jensen";
42                             sdw:cityId city:Vemb;
43                             sdw:hasCompany company:10165164;
44                             sdw:fromDate ‘‘2017-09-26";
45                             sdw:toDate ‘‘9999-12-31";
46                             sdw:status ‘‘Current".
47 ## After type3 update
48 recipient:762921 rdf:type qb4o:LevelMember;
49                 qb4o:member sdw:Recipient;
50                 sdw:name "R. Nielsen";
51                 sdw:cityId city:Lokken;
52                 sdw:hasCompany company:10165164.
53
54 recipient:291894 rdf:type qb4o:LevelMember;
55                  qb4o:memberOf sdw:Recipient.
56                  sdw:name "Kristian Jensen";
57                  sdw:name_oldValue "Kristian Kristensen";
58                  sdw:cityId city:Vemb;
59                  sdw:hasCompany company:10165164.
```

Besides the transformation operations discussed above, we define two additional transformation operations that cannot be run by the automatic ETL dataflows generation process.

ExternalLinking (sABox, externalSource) This operation links the resources of *sABox* with the resources of an external source *externalSource*. *externalSource* can either be a SPARQL endpoint or an API. For each resource *inRes* ∈

sABox, this operation extracts top k matching external resources either 1) submitting a query to *externalSource* or 2) by sending a web service request embedding *inRes* through the API (e.g., DBpedia lookup API). To find a potential link for each external resource *exRes*, the Jaccard Similarity of the semantic bags of *inRes* and *exRes* is computed. The semantic bag of a resource consists of triples describing the resource [38, 39, 50]. The pair of the internal and external resources is considered as a match if the Jaccard Similarity exceeds a certain threshold. A triple with the `owl:sameAs` property is created to materialize the link in *sABox* for each pair of matched internal and external resources.

MaterializeInference(ABox, TBox) This operation infers new information that has not been explicitly stated in an SDW. It analyzes the semantics encoded into the SDW and enriches the SDW with the inferred triples. A subset of the OWL 2 RL/RDF rules, which encodes part of the RDF-Based Semantics of OWL 2 [45], are considered here. The reasoning rules can be applied over the TBox *TBox* and ABox *ABox* separately, and then together. Finally, the resulting inference is asserted in the form of triples, in the same spirit as how SPARQL deals with inference.

6.3 Load

Loader(tripleSet, tsPath) An SDW is represented in the form of RDF triples and the triples are stored in a triple store (e.g., Jena TDB). Given a set of RDF triples *triplesSet* and the path of a triple store *tsPath*, this operation loads *triplesSet* in the triple store.

If the ETL execution flow is generated automatically, this operation first identifies the concept-mapping *cm* from *aMappings*, where *aConstruct* appears and the operation to process *cm* is *Loader*. Then, it takes values of `map:sourceLocation` and `map:targetLocation` for the parameters *tripleSet* and *tsPath*.

7 Automatic ETL Execution Flow Generation

We can characterize ETL flows at the Definition Layer by means of the source-to-target mapping file; therefore, the ETL data flows in the Execution Layer can be generated automatically. This way, we can guarantee that the created ETL data flows are sound and relieve the developer from creating the ETL data flows manually. Note that this is possible because the Mediatory Constructs (see Figure B.5) contain all the required metadata to automate the process.

```
Algorithm 8: CreateETL
  Input: TargetConstruct $x$, MappingFile $G$, IRIGraph $G_{IRI}$
  Output: A set of ETL flows $E$
  begin
1     $node_{target} \leftarrow$ find$(x, G)$
2     $?c_{maps} \leftarrow$findConMap$((node_{target}, G)$
      /* In $G:\ node_{target} \xleftarrow{\texttt{map:targetConcept}} ?c_{maps}$ ; $|?c_{maps}| \geq 0$ */
3     $E \leftarrow$ createSet()
4     if $(?c_{maps} \neq \emptyset)$ then
5         foreach $c \in ?c_{maps}$ do
6             $s_c \leftarrow$ createStack()
7             $s_c \leftarrow$ CreateAFlow$(c, s_c, G, G_{IRI})$
8             $E$.add$(s_c)$
9     return $E$
```

```
Algorithm 9: CreateAFlow
  Input: ConceptMapping $c$, Stack $s_c$, MappingFile $G$, IRIGraph $G_{IRI}$
  Output: Stack $s_c$
  begin
1     $ops \leftarrow$ findOperations$(c, G)$
      /* In $G:\ c \xrightarrow{\texttt{map:operation}} ops$ */
2     $s_c$. push(Parameterize$(ops, c, G, G_{IRI})$)
      /* Push parameterized operations in $s_c$ */
3     $scon \leftarrow$ findSourceConcept$(c, G)$
      /* In $G:\ c \xrightarrow{\texttt{map:sourceConcept}} scon$ */
4     $?scon_{map} \leftarrow$ findConMap$(scon, G)$
      /* In $G:\ scon \xleftarrow{\texttt{map:targetConcept}} ?scon_{map}$; $|?scon_{map}| \leq 1$ */
5     if $(|?scon_{map}| = 1)$ then
6         CreateAFlow$(nc \in ?scon_{map}, s_c, G)$
          /* recursive call with $nc$ */
7     $s_c$.push(StartOp)
      /* Push the ETL start operation to $s_c$ */
8     return $s_c$
```

Algorithm 8 shows the steps required to create ETL data flows to populate a target construct (a level, a concept, or a dataset). As inputs, the algorithm takes a target construct x, the mapping file G, and the IRI graph G_{IRI}, and it

Algorithm 10: Parameterize

```
Input: Seq $ops$, ConceptMapping $cm$, MappingFile $G$, IRIGraph $G_{IRI}$
Output: Stack, $s_{op}$
begin
1   $s_{op} \leftarrow$ createStack()
2   for ($i = 1$ to length($ops$)) do
3       if ($i = 1$) then
4           parameterizeOp($op[i], G$, loc($cm$), $G_{IRI}$)
5           $s_{op}$.push($op[i]$)
6       parameterizeOp($op[i], G$, outputPath($op[i-1]$), $G_{IRI}$)
7       $s_{op}$.push($op[i]$)
8   return $s_{op}$
```

outputs a set of ETL data flows E. At first, it locates x in G (line 1) and gets the concept-mappings where $node_{target}$ participates (line 2). As a (final) target construct can be populated from multiple sources, it can be connected to multiple concept-mappings, and for each concept-mapping, it creates an ETL data flow by calling the function[11] CreateAFlow (lines 5-8). Algorithm 9 generates an ETL data flow for a concept-mapping c and recursively does so if the current concept-mapping source element is connected to another concept-mapping as, until it reaches a source element. Algorithm 9 recursively calls itself and uses a stack to preserve the order of the partial ETL data flows created for each concept-mapping. Eventually, the stack contains the whole ETL data flow between the source and target schema.

Algorithm 9 works as follows. The sequence of operations in c is pushed to the stack after parameterizing it (lines 1-2). Algorithm 10 parameterizes each operation in the sequence, as described in Section 6 and returns a stack of parameterized operations. As inputs, it takes the operation sequence, the concept-mapping, the mapping file, and the IRI graph. For each operation, it uses the parameterizeOp(op, G, loc(cm), G_{IRI}) function to automatically parameterize op from G (as all required parameters are available in G) and push the parameterized operation in a stack (line 2-7). Note that, for the first operation in the sequence, the algorithm uses the source ABox location of the concept-mapping (line 4) as an input, whereas for the remaining operations, it uses the output of the previous operation as input ABox (line 6). Finally, Algorithm 10 returns the stack of parameterized operations.

Then, Algorithm 9 traverses to the adjacent concept-mapping of c connected via c's source concept (line 3-4). After that, the algorithm recursively

[11]Here, functions used in the algorithms are characterized by a pattern over G, shown at its side as comments in the corresponding algorithm.

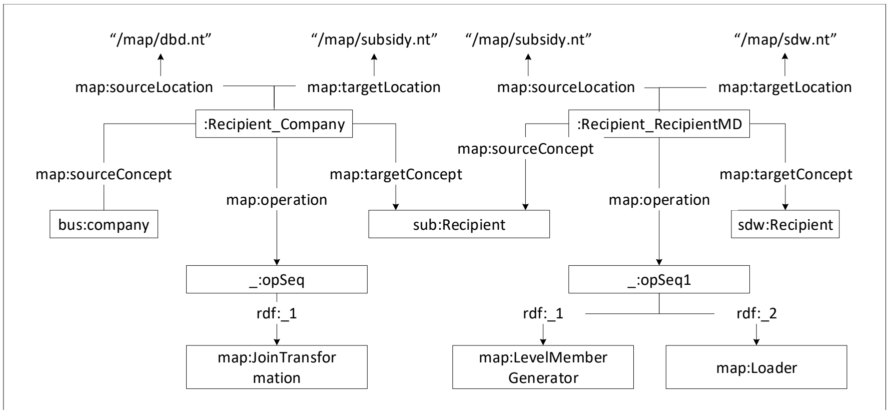


Fig. B.8: The graph presentation of a part of Listing B.3.

calls itself for the adjacent concept-mapping (line 6). Note that, here, we set a restriction: except for the final target constructs, all the intermediate source and target concepts can be connected to at most one concept-mapping. This constraint is guaranteed when building the metadata in the Definition Layer. Once there are no more intermediate concept-mappings, the algorithm pushes a dummy starting ETL operation (*StartOp*) (line 7) to the stack and returns it. *StartOp* can be considered as the root of the ETL data flows that starts the execution process. The stacks generated for each concept-mapping of $node_{target}$ are added to the output set E (line 8 in Algorithm 8). The following section shows this process with an example of our use case.

7.1 Auto ETL Example

In this section, we show how to populate `sdw:Recipient` level using CREATEETL (Algorithm 8). As input, CREATEETL takes the target construct `sdw:Recipient`, the mapping file Listing B.3 and the location of IRI graph "/map/provGraph.nt". Figure B.8 presents a part (necessary to explain this process) of Listing B.3 as an RDF graph. As soon as `sdw:Recipient` is found in the graph, the next task is to find the concept-mappings that are connected to `sdw:Recipient` through `map:targetConcept` (line 2). Here, $?c_{maps}$={`:Recipient_RecipientMD`}. For `:Recipient_RecipientMD`, the algorithm creates an empty stack s_c and calls CREATEAFLOW (`:Recipient_RecipientMD`, s_c, Listing B.3) (Algorithm 9) at lines 6-7.

CREATEAFLOW retrieves the sequence of operations (line 1) needed to process the concept-mapping, here: *ops=(LevelMemberGenerator; Loader)* (see Figure B.8). Then, CREATEAFLOW parameterizes *ops* by calling PARAMETERIZE (Algorithm 10) and then pushes the parameterized operations to s_c (line 2). PARAMETERIZE creates an empty stack s_{op} (line 1) and for each operation in *ops* it calls the PARAMETERIZE() method to parameterize the operation using the concept-mapping information from Listing B.3, the source location of the concept-mapping, and the IRI graph, and then it pushes the parameterized operation in s_{op} (lines 2-7). After the execution of the for loop in PARAMETERIZE (line 2-7), the value of the stack s_{op} is (*Loader*("/map/temp.nt", "/map/sdw"); *LevelMemberGenerator*(`sub:Recipient`,`sdw:Recipient`, "/map/subsidyTBox.ttl", "/map/subsidy.nt", "/map/subsidyMDTBox.ttl", `sub:recipientID`, "/map/provGraph.nt", propertyMappings[12], "/map/temp.nt")), which is returned to line 2 of CREATEAFLOW. Note that the output path of *LevelMemberGenerator(..)*[13] is used as the input source location of *Loader(..)*. CREATEAFLOW pushes the parameterized operations to s_c (line 2), hence s_c= (*LevelMemberGenerator(..); Loader(..)*).

Then, CREATEAFLOW finds the source concept of `:Recipient_RecipientMD` (line 3), which is `sub:Recipient`; retrieves the concept-mapping `:Recipient_Company` of `sub:Recipient` from Listing B.3 (line 4); and recursively calls itself for `:Recipient_Company` (lines 5-6). The operation sequence of `:Recipient_Company` (line 1) is (*JoinTransformation*) (see Figure B.8), and the call of PARAMETERIZE at line 2 returns the parameterized *JoinTransformation*(`bus:Company`, `Sub:Reicipient`, "/map/dbd.nt","/map/subsidy.nt", comProperty[14], propertyMappings[15]) operation, which is pushed to the stack s_c, i.e., s_c= (*JoinTransformation(...); LevelMemberGenerator(...); Loader(...)*). The source concept `bus:Company` is not connected to any other concept-mapping through the `map:targetConcept` property in the mapping file. Therefore, CREATEAFLOW skips lines 5 and 6. After that, it pushes the start operation (StartOp) in s_c, i.e., s_c= (*StartOp, JoinTransformation(..); LevelMemberGenerator(..); Loader(..)*) and returns it to CREATEETL (Algorithm 8) at line 7. Then, CREATEETL adds it to the set of ETL data flows E and returns it (line 9). Therefore, the ETL data flow to populate `sdw:Recipient` is *StartOp* ⇒*JoinTransformation(..)* ⇒*LevelMemberGenerator(..)* ⇒*Loader(..)*.

8 Evaluation

We created a GUI-based prototype, named *SETL*$_{CONSTRUCT}$ [13] based on the different high-level constructs described in Sections 5 and 6. We use Jena 3.4.0 to process, store, and query RDF data and Jena TDB as a triplestore. *SETL*$_{CONSTRUCT}$ is developed in Java 8. Like other traditional tools (e.g., PDI [8]), *SETL*$_{CONSTRUCT}$ allows developers to create ETL flows by dragging, dropping, and connecting the ETL operations. The system is demonstrated

[12]Here, lines 95-112 in Listing B.3.
[13](..) indicates that the operation is parameterized.
[14]lines 32,36-39 in Listing B.3.
[15]lines 76-93 in Listing B.3.

Tool ⇒		SETL				$SETL_{CONSTRUCT}$				$SETL_{AUTO}$			
						NOUC	Components of each Concept			NOUC	Components of each concept		
ETL Task ⇓	Sub Construct	Procedures/ Data Structures	NUEP	NOTC	Task/Operation		Clicks	Selections	NOTC		Clicks	Selections	NOTC
TBox with MD Semantics	Cube Structure	Concept(), Property(), BlankNode(), Namespaces(), conceptPropertyBindings(), createOntology()	1	665	TargetTBox-Definition	1	9	9	7	1	9	9	7
	Cube Dataset		1	67		1	2	2	18	1	2	2	18
	Dimension		2	245		2	3	4	12	2	3	4	12
	Hierarchy		7	303		7	4	6	18	7	4	6	18
	Level		16	219		16	4	5	17	16	4	5	17
	Level Attribute		38	228		38	4	3	15	38	4	3	15
	Rollup Property		12	83		12	2	1	7	12	2	1	7
	Measure Property		1	82		1	4	3	25	1	4	3	25
	Hierarchy Step		12	315		12	6	6	6	12	6	6	6
	Prefix		21	74		21	1	0	48	21	1	0	48
Total (for the Target TBox)			**1**	**24509**		**101**	**382**	**342**	**1905**	**101**	**382**	**342**	**1905**
Mapping Generation	Mapping Dataset	NA	0	0	SourceTo-Target-Mapping	5	1	2	6	3	1	2	6
	Concept-mapping	dict()	17	243		23	9	10	0	18	12	10	15
	Property-mapping	NA	0	0		59	2	2	0	44	2	2	0
Total (for the Mapping File)			**1**	**2052**		**87**	**330**	**358**	**30**	**65**	**253**	**274**	**473**
Semantic Data Extraction from an RDF Dump file	-	query()	17	40	GraphExtractor	17	5	0	30	1	Auto Generation		
Semantic Data Extraction through a SPARQL endpoint	-	ExtractTriples-FromEndpoint()	0	100	GraphExtractor	0	5	0	30	0	Auto Generation		
Cleansing	-	Built-in Petl functions	15	80	Transformation-OnLiteral	5	12	1	0	1	Auto Generation		
Join	-	Built-in Petl functions	1	112	Join-Transformation	1	15	1	0	1	Auto Generation		
Level Member Generation	-	createDataTripleToFile(), createDataTripleTo-TripleStore()	16	75	LevelMember-Generator	16	6	6	0	1	Auto Generation		
Observation Generation	-	createDataTripleToFile(), createDataTripleTo-TripleStore()	1	75	Observation-Generator	1	6	6	0	1	Auto Generation		
Loading as RDF Dump	-	insertTriplesIntoTDB(), bulkLoadToTDB()	0	113	Loader	0	1	2	0	0	Auto Generation		
Loading to a triple store	-	insertTriplesIntoTDB(), bulkLoadToTDB()	17	153	Loader	17	1	2	0	1	Auto Generation		
Auto ETL Generation	-	-	-	-	CreateETL	1	-	-	-	1	21	16	0
Total (for the Execution Layer Tasks)			**1**	**2807**		**57**	**279**	**142**	**363**	**58**	**21**	**16**	**0**
Total (for the whole ETL Process)				**29358**		**245**	**991**	**826**	**2298**	**224**	**656**	**632**	**2378**

Table B.2: Comparison among the productivity of $SETL_{PROG}$, $SETL_{CONSTRUCT}$, and $SETL_{AUTO}$ for the SETLSDW.

in [13]. On top of $SETL_{CONSTRUCT}$, we implement the automatic ETL execution flow generation process discussed in Section 7; we call it $SETL_{AUTO}$. The source code, examples, and developer manual for $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ are available at `https://github.com/bi-setl/SETL`.

To evaluate $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$, we create an MD SDW by integrating the Danish Business Dataset (DBD) [3] and the Subsidy dataset (`https://data.farmsubsidy.org/Old/`), described in Section 2. We choose this use case and these datasets for evaluation as there already exists an MD SDW, based on these datasets, that has been programatically created using $SETL_{PROG}$ [37]. Our evaluation focuses on three aspects: 1) productivity, i.e., to what extent $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ ease the work of a developer when developing an ETL task to process semantic data, 2) development time, the time to develop an ETL process, and 3) performance, the time required to run the process. We run the experiments on a laptop with a 2.10 GHz Intel Core(TM) i7-4600U processor, 8 GB RAM, and Windows 10.

8.1 Productivity

$SETL_{PROG}$ requires Python knowledge to maintain and implement an ETL process. On the other hand, using $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$, a developer can create all the phases of an ETL process by interacting with a GUI. Therefore, they provide a higher level of abstraction to the developer that hides low-level details and requires no programming background. Table B.2 summarizes the effort required by the developer to create different ETL tasks using $SETL_{PROG}$, $SETL_{CONSTRUCT}$, and $SETL_{AUTO}$. We measure the developer effort in terms of Number of Typed Characters (NOTC) for $SETL_{PROG}$ and in Number of Used Concepts (NOUC) for $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$. Here, we define a concept as a GUI component of $SETL_{CONSTRUCT}$ that opens a new window to perform a specific action. A concept is composed of several clicks, selections, and typed characters. For each ETL task, Table B.2 lists: 1) its sub construct, required procedures/data structures, number of the task used in the ETL process (NUEP), and NOTC for $SETL_{PROG}$; 2) the required task/operation, NOUC, and number of clicks, selections, and NOTC (for each concept) for $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$. Note that NOTC depends on the user input. Here, we generalize it by using same user input for all systems. Therefore, the total NOTC is not the summation of the values of the respective column.

To create a target TBox using $SETL_{PROG}$, an ETL developer needs to use the following steps: 1) defining the TBox constructs by instantiating the *Concept*, *Property*, and *BlankNode* classes that play a meta modeling role for the constructs and 2) calling *conceptPropertyBinding()* and *createTriples()*. Both procedures take the list of all concepts, properties, and blank nodes as parameters. The former one internally connects the constructs to each other, and the latter creates triples for the TBox. To create the TBox of our SETLSDW using $SETL_{PROG}$, we used 24,509 NOTC. On the other hand, $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ use the *TargetTBoxDefinition* interface to create/edit a target TBox. To create the target TBox of our SETLSDW in $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$, we use 101 concepts that require only 382 clicks, 342 selections, and 1905 NOTC. Therefore, for creating a target TBox, $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ reduce use 92% fewer NOTC than $SETL_{PROG}$.

To create source-to-target mappings, $SETL_{PROG}$ uses Python *dictionaries* where the keys of each dictionary represent source constructs and values are the mapped target TBox constructs, and for creating the ETL process it uses 23 dictionaries, where the biggest dictionary used in the ETL process takes 253 NOTC. In total, $SETL_{PROG}$ uses 2052 NOTC for creating mappings for the whole ETL process. On the contrary, $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ use a GUI. In total, $SETL_{CONSTRUCT}$ uses 87 concepts composed of 330 clicks, 358 selections, and 30 NOTC. Since $SETL_{AUTO}$ internally creates the intermediate mappings, there is no need to create separate mapping datasets and concept-mappings for intermediate results. Thus, $SETL_{AUTO}$ uses only 65 concepts requiring 253 clicks, 274 selections, and 473 NOTC. Therefore, $SETL_{CONSTRUCT}$ reduces NOTC of $SETL_{PROG}$ by 98%. Although $SETL_{AUTO}$ uses 22 less concepts than $SETL_{CONSTRUCT}$, $SETL_{CONSTRUCT}$ reduces NOTC of $SETL_{AUTO}$ by 93%. This is because, in $SETL_{AUTO}$, we write the data extraction queries in the concept-mappings where in $SETL_{CONSTRUCT}$ we set the data extraction queries in the ETL operation level.

To extract data from either an RDF local file or an SPARQL endpoint, $SETL_{PROG}$ uses the *query()* procedure and the *ExtractTriplesFromEndpoint()* class. On the other hand, $SETL_{CONSTRUCT}$ uses the *GraphExtractor* operation. It uses 1 concept composed of 5 clicks and 20 NOTC for the local file and 1 concept with 5 clicks and 30 NOTC for the endpoint.

$SETL_{PROG}$ uses different functions/procedures from the Petl Python library (integrated with SETL) based on the cleansing requirements. In $SETL_{CONSTRUCT}$, all data cleansing related tasks on data sources are done using *TransformationOnLiteral* (single source) and *JoinTransformation* (for multi-source). *TransformationOnLiteral* requires 12 clicks and 1 selection, and *JoinTransformation* takes 15 clicks and 1 selection. To create a level member and observation, $SETL_{PROG}$ uses $createDataTripleToFile()$ and takes 125 NOTC. The procedure takes all the classes, properties, and blank nodes of a target TBox as input; therefore, the given TBox should be parsed for being used in the procedure. On the other hand, $SETL_{CONSTRUCT}$ uses the *LevelMemberGenerator* and *ObservationGenerator* operations, and each operation requires 1 concept, which takes 6 clicks and 6 selections. $SETL_{PROG}$ provides procedures for either bulk or trickle loading to a file or an endpoint. Both procedures take 113 and 153 NOTC, respectively. For loading RDF triples either to a file or a triple store, $SETL_{CONSTRUCT}$ uses the *Loader* operation, which needs 2 clicks and 2 selections. Therefore, $SETL_{CONSTRUCT}$ reduces NOTC for all transformation and loading tasks by 100%.

$SETL_{AUTO}$ requires only a target TBox and a mapping file to generate ETL data flows through the *CreateETL* interface, which takes only 1 concept composed of 21 clicks and 16 selections. Therefore, other ETL Layer tasks are automatically accomplished internally. In summary, $SETL_{CONSTRUCT}$ uses 92% fewer NOTC than $SETL_{PROG}$, and $SETL_{AUTO}$ further reduces NOUC by another 25%.

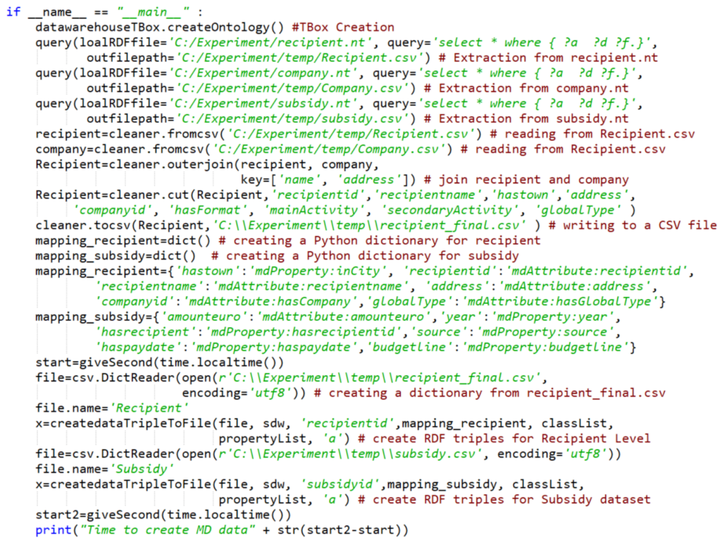
8.2 Development Time

We compare the time used by $SETL_{PROG}$, $SETL_{CONSTRUCT}$, and $SETL_{AUTO}$ to build the ETL processes for our use case SDW. As the three tools were developed within the same project scope and we master them, the first author conducted this test. We chose the running use case used in this paper and created a solution in each of the three tools and measured the development time. We used each tool twice to simulate the improvement we may obtain when we are knowledgeable about a given project. The first time it takes more time to analyze, think, and create the ETL process, and in the latter, we reduce the interaction time spent on analysis, typing, and clicking. Table B.3 shows the development time (in minutes) for main integration tasks used by the different systems to create the ETL processes.

Tool	**Iteration Number**	**Target TBox Definition**	**Mapping Generartion**	**ETL Design**	**Total**
$SETL_{PROG}$	1	186	51	85	322
$SETL_{PROG}$	2	146	30	65	241
$SETL_{CONSTRUCT}$	1	97	46	35	178
$SETL_{CONSTRUCT}$	2	58	36	30	124
$SETL_{AUTO}$	1	97	40	2	139
$SETL_{AUTO}$	2	58	34	2	94

Table B.3: ETL development time (in minutes) required for $SETL_{PROG}$, $SETL_{CONSTRUCT}$, and $SETL_{AUTO}$.

$SETL_{PROG}$ took twice as long as $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ to develop a target TBox. In $SETL_{PROG}$, to create a target TBox construct, for example a level l, we need to instantiate the *Concept()* class for l and then add its different property values by calling different methods of *Concept()*. $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ create l by typing the input or selecting from the suggested items. Thus, $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ also reduce the risk of making mistakes in an error-prone task, such as creating an ETL. In $SETL_{PROG}$, we typed all the source and target properties to create a map-

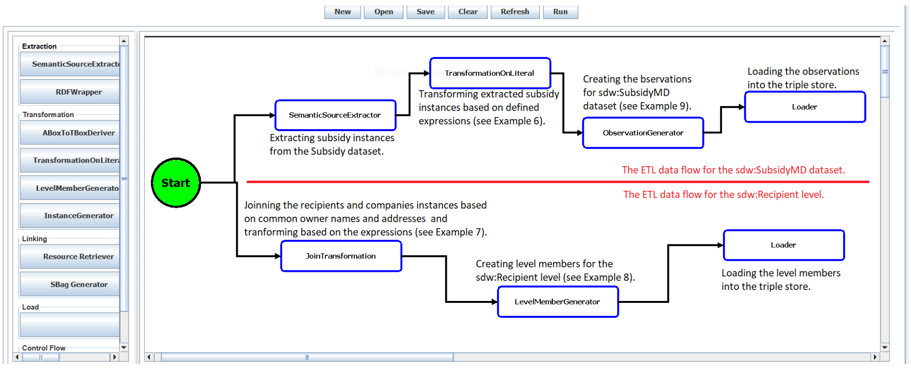


```
if __name__ == "__main__" :
    datawarehouseTBox.createOntology() #TBox Creation
    query(loalRDFfile='C:/Experiment/recipient.nt', query='select * where { ?a  ?d ?f.}',
          outfilepath='C:/Experiment/temp/Recipient.csv') # Extraction from recipient.nt
    query(loalRDFfile='C:/Experiment/company.nt', query='select * where { ?a  ?d ?f.}',
          outfilepath='C:/Experiment/temp/Company.csv') # Extraction from company.nt
    query(loalRDFfile='C:/Experiment/subsidy.nt', query='select * where { ?a  ?d ?f.}',
          outfilepath='C:/Experiment/temp/subsidy.csv') # Extraction from subsidy.nt
    recipient=cleaner.fromcsv('C:/Experiment/temp/Recipient.csv') # reading from Recipient.csv
    company=cleaner.fromcsv('C:/Experiment/temp/Company.csv') # reading from Recipient.csv
    Recipient=cleaner.outerjoin(recipient, company,
                                key=['name', 'address']) # join recipient and company
    Recipient=cleaner.cut(Recipient,'recipientid','recipientname','hastown','address',
        'companyid', 'hasFormat', 'mainActivity', 'secondaryActivity', 'globalType' )
    cleaner.tocsv(Recipient,'C:\\Experiment\\temp\\recipient_final.csv' ) # writing to a CSV file
    mapping_recipient=dict() # creating a Python dictionary for recipient
    mapping_subsidy=dict()  # creating a Python dictionary for subsidy
    mapping_recipient={'hastown':'mdProperty:inCity', 'recipientid':'mdAttribute:recipientid',
            'recipientname':'mdAttribute:recipientname', 'address':'mdAttribute:address',
            'companyid':'mdAttribute:hasCompany','globalType':'mdAttribute:hasGlobalType'}
    mapping_subsidy={'amounteuro':'mdAttribute:amounteuro','year':'mdProperty:year',
            'hasrecipient':'mdProperty:hasrecipientid','source':'mdProperty:source',
            'haspaydate':'mdProperty:haspaydate','budgetline':'mdProperty:budgetline'}
    start=giveSecond(time.localtime())
    file=csv.DictReader(open(r'C:\\Experiment\\temp\\recipient_final.csv',
                        encoding='utf8')) # creating a dictionary from recipient_final.csv
    file.name='Recipient'
    x=createdataTripleToFile(file, sdw, 'recipientid',mapping_recipient, classList,
                             propertyList, 'a') # create RDF triples for Recipient Level
    file=csv.DictReader(open(r'C:\\Experiment\\temp\\subsidy.csv', encoding='utf8'))
    file.name='Subsidy'
    x=createdataTripleToFile(file, sdw, 'subsidyid',mapping_subsidy, classList,
                             propertyList, 'a') # create RDF triples for Subsidy dataset
    start2=giveSecond(time.localtime())
    print("Time to create MD data" + str(start2-start))
```

Fig. B.9: The segment of the main method to populate the `sdw:SubsidyMD` dataset and the `sdw:Recipient` using *SETL*$_{PROG}$.

ping dictionary for each source and target concept pair. However, to create mappings, *SETL*$_{CONSTRUCT}$ and *SETL*$_{AUTO}$ select different constructs from a source as well as a target TBox and only need to type when there are expressions and/or filter conditions of queries. Moreover, *SETL*$_{AUTO}$ took less time than *SETL*$_{CONSTRUCT}$ in mapping generation because we did not need to create mappings for intermediate results.

To create ETL data flows using *SETL*$_{PROG}$, we had to write scripts for cleansing, extracting, transforming, and loading. *SETL*$_{CONSTRUCT}$ creates ETL flows using drag-and-drop options. Note that the mappings in *SETL*$_{CONSTRUCT}$ and *SETL*$_{AUTO}$ use expressions to deal with cleansing and transforming related tasks; however, in *SETL*$_{PROG}$ we cleansed and transformed the data in the ETL design phase. Hence, *SETL*$_{PROG}$ took more time in designing ETL compared to *SETL*$_{CONSTRUCT}$. On the other hand, *SETL*$_{AUTO}$ creates the ETL data flows automatically from a given mapping file and the target TBox. Therefore, *SETL*$_{AUTO}$ took only two minutes to create the flows. In short, *SETL*$_{PROG}$ is a programmatic environment, while *SETL*$_{CONSTRUCT}$ and *SETL*$_{AUTO}$ are drag and drop tools. We exemplify this fact by means of Figures B.9 and B.10, which showcase the creation of the ETL data flows for the `sdw:SubsidyMD` dataset and the `sdw:Recipient` level. To make it more readable and understandable, we add comments at the end of the lines of Figure B.9 and in each operation of Figure B.10. In summary, using *SETL*$_{CONSTRUCT}$, the development time is cut in almost half (41% less development time than *SETL*$_{PROG}$); and using *SETL*$_{AUTO}$ it is cut by another



Extracting subsidy instances from the Subsidy dataset.

Transforming extracted subsidy instances based on defined expressions (see Example 6).

Creating the bservations for sdw:SubsidyMD dataset (see Example 9).

Loading the observations into the triple store.

The ETL data flow for the sdw:SubsidyMD dataset.

The ETL data flow for the sdw:Recipient level.

Joinning the recipients and companies instances based on common owner names and addresses and tranforming based on the expressions (see Example 7).

Creating level members for the sdw:Recipient level (see Example 8).

Loading the level members into the triple store.

Fig. B.10: The ETL flows to populate the `sdw:SubsidyMD` dataset and the `sdw:Recipient` level using *SETL*$_{CONSTRUCT}$.

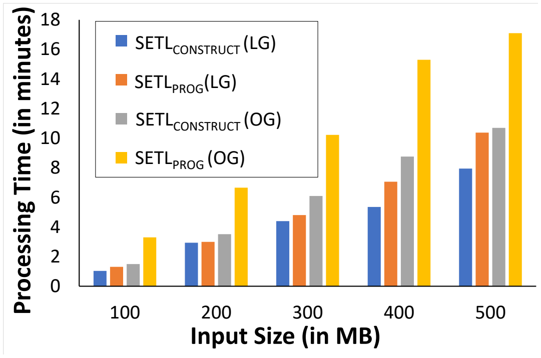


Fig. B.11: Comparison of *SETL*$_{CONSTRUCT}$ and *SETL*$_{PROG}$ for semantic transformation.Here, LG and OG stand for *LevelMemberGenerator* and *ObservationGenerator*.

27%.

Performance Metrics	**Systems**	**Extraction and Traditional Transformation**	**Semantic Transformation**	**Loading**	**Total Processing Time**
Processing Time (in minutes)	*SETL*$_{PROG}$	33	17.86	21	71.86
	SETL$_{CONSTRUCT}$	43.05	39.42	19	101.47
Input Size	*SETL*$_{PROG}$	6.2 GB (Jena TDB) + 6.1 GB (N-Triples)	496 MB (CSV)	4.1 GB (N-Triples)	
	SETL$_{CONSTRUCT}$	6.2 GB (Jena TDB) +6.1 GB (N-Triples)	6.270 GB (N-Triples)	4.1 GB (N-Triples)	
Output Size	*SETL*$_{PROG}$	490 MB (CSV)	4.1 GB (N-Tripels)	3.7 GB (Jena TDB)	
	SETL$_{CONSTRUCT}$	6.270 GB (N-Triples)	4.1 GB (N-Triples)	3.7 GB (Jena TDB)	

Table B.4: ETL execution time required for each sub-phase of the ETL processes created using *SETL*$_{PROG}$ and *SETL*$_{CONSTRUCT}$.

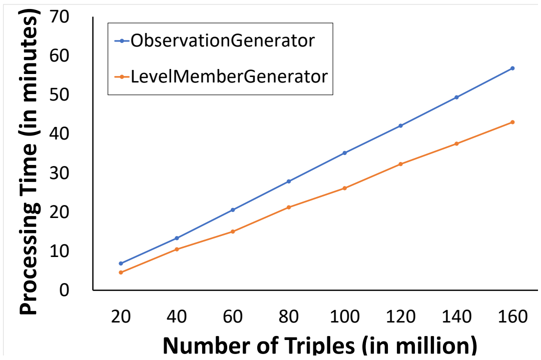


Fig. B.12: Scalability of *LevelMemberGenerator* and *ObservationGenerator*.

8.3 Performance

Since the ETL processes of $SETL_{CONSTRUCT}$ and $SETL_{AUTO}$ are the same and only differ in the developer effort needed to create them, this section only compares the performance of $SETL_{PROG}$ and $SETL_{CONSTRUCT}$. We do so by analyzing the time required to create the use case SETLSDW by executing the respective ETL processes. To evaluate the performance of similar types of operations, we divide an ETL process into three sub-phases: extraction and traditional transformation, semantic transformation, as well as loading and discuss the time to complete each.

Table B.4 shows the processing time (in minutes), input and output size of each sub-phase of the ETL processes created by $SETL_{PROG}$ and $SETL_{CONSTRUCT}$. The input and output formats of each sub-phase are shown in parentheses. The extraction and traditional transformation sub-phases in both systems took more time than the other sub-phases. This is because they include time for 1) extracting data from large RDF files, 2) cleansing and filtering the noisy data from the DBD and Subsidy datasets, and 3) joining the DBD and Subsidy datasets. $SETL_{CONSTRUCT}$ took more time than $SETL_{PROG}$ because its *TransformationOnLiteral* and *JoinTransformation* operations use SPARQL queries to process the input file whereas $SETL_{PROG}$ uses the methods from the Petl Python library to cleanse the data extracted from the sources.

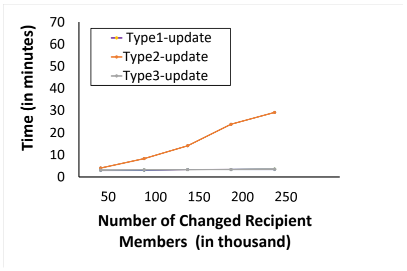
$SETL_{CONSTRUCT}$ took more time during the semantic transformation than $SETL_{PROG}$ because the operations of $SETL_{CONSTRUCT}$ take RDF (N-triples) files as input, which are larger in size compared to the CSV files (see Table B.4), the input format of the semantic transformation process of $SETL_{PROG}$. To ensure our claims, we run an experiment for measuring the performance of the semantic transformation procedures of $SETL_{PROG}$ and the operations of $SETL_{CONSTRUCT}$ by taking CSV files as inputs instead of RDF ones. Figure B.11 shows the processing time taken by *SETL$_{CONSTRUCT}$* operations and *SETL$_{PROG}$* procedures to create level members and observations with increasing input size. In the figure, LG and OG represent level member generator and observation generator operations (in case of *SETL$_{CONSTRUCT}$*) or procedures (in case of *SETL$_{PROG}$*).

In summary, to process an input CSV file with 500 MB in size, *SETL$_{CONSTRUCT}$* takes 37.4% less time than *SETL$_{PROG}$* to create observations and 23,4% less time than *SETL$_{PROG}$* to create level members. The figure also shows that the processing time difference between the corresponding *SETL$_{CONSTRUCT}$* operation and the *SETL$_{PROG}$* procedure increases with the size of the input. In order to guarantee scalability when using the Jena library, a *SETL$_{CONSTRUCT}$* operation takes the (large) RDF input in the N-triple format, divides the file into several smaller chunks, and processes each chunk separately. Figure B.12 shows the processing time taken by *LevelMemberGenerator* and *ObservationGenerator* operations with the increasing number of triples. We shows the scalability of the *LevelMemberGenerator* and *ObservationGenerator* because they creates data with MD semantics. The figure shows that the processing time of both operations increase linearly with the increase in the number of triples, which ensures that both operations are scalable. *SETL$_{CONSTRUCT}$* takes less time in loading than *SETL$_{PROG}$* because *SETL$_{PROG}$* uses the Jena TDB loader command to load the data while *SETL$_{CONSTRUCT}$* programmatically load the data using the Jena API's method.

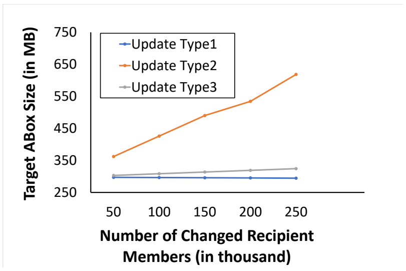
In summary, *SETL$_{PROG}$* and *SETL$_{CONSTRUCT}$* have similar performance (29% difference in total processing time). *SETL$_{CONSTRUCT}$* uses RDF as a canonical model, which makes it more general and powerful than *SETL$_{PROG}$*.

Besides the differences of the performances already explained in Table B.4, *SETL$_{CONSTRUCT}$* also includes an operation to update the members of the SDW levels, which is not included in *SETL$_{PROG}$*. Since the ETL process for our use case SDW did not include that operation, we scrutinize the performance of this specific operation of *SETL$_{CONSTRUCT}$* in the following.

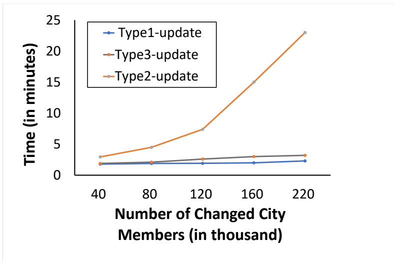
Performance analysis of UpdateLevel operation Figure B.13 shows the performance of the *UpdateLevel* operation. To evaluate this operation, we consider two levels: `mdProperty:Recipient` and `mdProperty:City`. We consider these two levels because `mdProperty:Recipient` is the largest level (in terms of size) in our use case, and `mdProperty:City` is the immediate upper level of `mdProperty:Recipient` in the `mdStructure:Address` hierarchy of the dimension `mdProperty:Beneficiary`. Therefore, we can record how the changes in cities propagate to recipients, especially in Type2-update. The `sdw:Recipient` level is the lowest granularity level in the `mdStructure:Address` hierarchy; therefore, changes in a recipient (i.e., a member of `sdw:Recipient`) only affect that recipient. Figure B.13a shows the processing time with the increasing number of recipients. As a Type2-update creates a new ver-



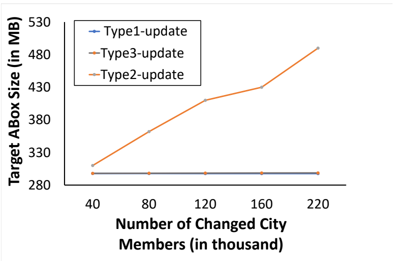
(a) Processing time of increasing changed recipient members.



(b) Target size of increasing changed recipient members.



(c) Processing time of increasing changed city members.



(d) Target size of increasing changed city members.

Fig. B.13: Performance of *UpdateLevel* based on processing time and target ABox size with the changing of the `sdw:Recipient` and `sdw:City` members.

sion for each changed level member, it takes more time than a Type1-update and a Type3-update. A Type3-update takes more time than a Type1-update because it keeps the record of old property values besides the current ones. Figure B.13b shows how the size of the target ABox increases with the increasing number of recipients. The target ABox size increases linearly with the increasing number of recipients (see Figure B.13b) for Type2-update and Type-3 updates because they keep additional information. However, the target ABox size decreases with the increasing number of recipients for Type1-updates; this is because the current property-values are smaller than the older ones in size.

Figure B.13c shows the processing time when increasing the number of updated cities. Since `sdw:City` is the immediate upper level of `sdw:Recipient` in the `mdStructure:address` hierarchy, to reflect a changed city in the target ABox, a Type2-update creates new versions for itself as well as for the recipients living in that city. Therefore, a Type2-update takes more processing time in comparison to a Type1-update and a Type-3 update. Figure B.13d shows the target ABox size with the increasing number of cities. The figure shows that the target ABox size for Type2-updates increases slowly within the range from 120 to 160 (X-axis), and it indicates that the changed cities within this range contain fewer recipients than the cities in other ranges.

In overall, we conclude that $SETL_{PROG}$ and $SETL_{CONSTRUCT}$ have similar performance and $SETL_{CONSTRUCT}$ is a scalable framework. Moreover, $SETL_{CONSTRUCT}$ supports the SDW's update.

9 Related Work

Nowadays, combining SW and BI technologies is an emerging research topic as it opens interesting research opportunities. As a DW deals with both internal and (increasingly) external data presented in heterogeneous formats, especially in the RDF format, semantic issues should be considered in the integration process [1]. Furthermore, the popularity of SW data gives rise to new requirements for BI tools to enable OLAP-style analysis over this type of data [29]. Therefore, the existing research related to semantic ETL is divided into two lines: 1) on the one hand, the use of SW technologies to physically integrate heterogeneous sources and 2) on the other hand, enabling OLAP analysis over SW data.

One prominent example following the first research line is [52], which presents an ontology-based approach to enable the construction of ETL flows. At first, the schema of both the sources and the DW are defined by a common graph-based model, named the datastore graph. Then, the semantics of the datastore graphs of the data sources and the DW are described by generating an (OWL-based) application ontology, and the mappings between the sources and the target are established through that ontology. In this way this approach addresses heterogeneity issues among the source and target schemata and finally demonstrates how the use of an ontology enables a high degree of automation from the source to the target attributes, along with the appropriate ETL transformations. Nonetheless, computationally complex ETL operations like slowly changing dimensions and the annotation of the application ontology with MD semantics are not addressed in this work. Therefore, OLAP queries cannot be applied on the generated DW.

Another piece of existing work [6] aligned to this line of research proposes a methodology describing some important steps required to make an SDW, which enables to integrate data from semantic databases. This approach also misses the annotation of the SDW with MD semantics. [53] has proposed an approach to support data integration tasks in two steps: 1) constructing ontologies from XML and relational sources and 2) integrating the derived ontologies by means of existing ontology alignment and merging techniques. However, ontology alignment techniques are complex and error-prone. [5] presents a semantic ETL framework at the conceptual level. This approach utilizes the SW technologies to facilitate the integration process and discusses the use of different available tools to perform different steps of the ETL process. [3] presents a method to spatially integrate a Danish Agricultural dataset and a Danish Business dataset using an ontology. The approach uses SQL views and other manual processes to cleanse the data and Virtuoso for creating and storing integrated RDF data. [33] presents UnifiedViews, an open-source ETL framework that supports management of RDF data. Based on the SPARQL queries, they define four types of Data Processing Units (DPUs): Extractor, Transformer, Loader, and Quality Assessor. However, the DPUs do not support to generate MD RDF data.

In the second line of research, a prominent paper is [40], which outlines a semi-automatic method for incorporating SW data into a traditional MD data management system for OLAP analysis. The proposed method allows an analyst to accomplish the following tasks: 1) designing the MD schema from the TBox of an RDF dataset, 2) extracting the facts from the ABox of the dataset and populating the MD fact table, and 3) producing the dimension hierarchies from instances of the fact table and the TBox to enable MDX queries over the generated DW. However, the generated DW no longer preserves the SW data principles defined in [24]; thus, OLAP analysis directly over SW data is yet to be addressed. To address this issue, [10] introduces the notion of a *lens*, called the analytical schema, over an RDF dataset. An analytical schema is a graph of classes and properties, where each node of the schema presents a set of facts that can be analyzed by traversing the reachable nodes. [25] presents a self-service OLAP endpoint for an RDF dataset. This approach first superimposes an MD schema over the RDF dataset. Then, a semantic analysis graph is generated on top of that MD schema, where each node of the graph represents an analysis situation corresponding to an MD query, and an edge indicates a set of OLAP operations.

Both [10] and [25] require either a *lens* or a semantic analysis graph to define MD views over an RDF dataset. Since most published SW data contains facts and figures, W3C recommends the Data Cube (QB) [12] vocabulary to standardize the publication of SW data with MD semantics. Although QB is appropriate to publish statistical data and several publishers (e.g., [44]) have already used the vocabulary for publishing statistical datasets, it has limitations to define MD semantics properly. The QB4OLAP [16] vocabulary enriches QB to support MD semantics by providing constructs to define 1) a cube structure in terms of different level of dimensions, measures, and attaching aggregate functions with measures and 2) a dimension structure in terms of levels, level attributes, relationships, the cardinality of relationships among the levels, and hierarchies of the dimension. Therefore, MD data can be published either by enriching data already published using QB with dimension levels, level members, dimension hierarchies, and the association of aggregate functions to measures without affecting the existing observations

[55] or using QB4OLAP from scratch [18, 26, 37].

In [31], the authors present an approach to enable OLAP operations on a single data cube published using the QB vocabulary and shown the applicability of their OLAP-to-SPARQL mapping in answering business questions in the financial domain. However, their OLAP-to-SPARQL mapping may not result in the most efficient SPARQL query and requires additional efforts and a longer time to get the results as they consider that the cube is queried on demand and the DW is not materialized. While some approaches have proposed techniques to optimize execution of OLAP-style SPARQL queries in a federated setting [27], others have considered view materialization [17, 28]. The authors in [55] present a semi-automatic method to enrich the QB dataset with QB4OLAP terms. However, there is no guideline for an ETL process to populate a DW annotated with QB4OLAP terms.

After analyzing the two research lines, we can draw some conclusions. Although each approach described above addresses one or more aspects of a semantic ETL framework, there is no single platform that supports them all (target definition, source-to-target mappings generation, ETL generations, MD target population, and evolution). To solve this problem, we have proposed a Python-based programmable semantic ETL ($SETL_{PROG}$) framework [37] that provides a number of powerful modules, classes, and methods for performing the tasks mentioned above. It facilitates developers by providing a higher abstraction level that lowers the entry barriers. We have experimentally shown that $SETL_{PROG}$ performs better in terms of programmer productivity, knowledge base quality, and performance, compared to other existing solutions. However, to use it, developers need a programming background. Although $SETL_{PROG}$ enables to create an ETL flow and provides methods by combining several tasks, there is a lack of a well-defined set of basic semantic ETL constructs that allow users more control in creating their ETL process. Moreover, how to update an SDW to synchronize it with the changes taking place in the sources is not discussed. Further, in a data lake/big data environment, the data may come from heterogeneous formats, and the use of the relational model as the canonical model may generate an overhead. Transforming JSON or XML data to relational data to finally generate RDF can be avoided by using RDF as the canonical model instead. To this end, several works have discussed the appropriateness of knowledge graphs for data integration purposes and specifically, as a canonical data model [11, 48, 49]. An additional benefit of using RDF as a canonical model is that it allows adding semantics without being compliant to a fixed schema. The present paper presents the improvements introduced on top of $SETL_{PROG}$ to remedy its main drawbacks discussed above. As there are available RDF Wrappers (e.g., [14, 51]) to convert another format to RDF, in this paper, we focus on only semantic data and propose an RDF-based two-layered (Definition Layer and Execution Layer) integration process. We also propose a set of high-level

ETL constructs (tasks/operations) for each layer, with the aim of overcoming the drawbacks identified above for $SETL_{PROG}$. We also provide an operation to update an SDW based on the changes in source data. On top of that, we characterize the ETL flow in the Definition Layer by means of creating an RDF based source-to-target mapping file, which allows to automate the ETL execution flows.

10 Conclusion and Future Work

In this paper, we proposed a framework of a set of high-level ETL constructs to integrate semantic data sources. The overall integration process uses the RDF model as canonical model and is divided into the Definition and Execution Layer. In the Definition Layer, ETL developers create the metadata (target and source TBoxes, and source-to-target mappings). We propose a set of high-level ETL operations for semantic data that can be used to create ETL data flows in the Execution Layer. As we characterize the transformations in the Definition Layer in terms of source-to-target mappings at the schema level, we are able to propose an automatic algorithm to generate ETL data flows in the Execution Layer. We developed an end-to-end prototype $SETL_{CONSTRUCT}$ based on the high-level constructs proposed in this paper. We also extended it to enable automatic ETL execution flows generation (and named it $SETL_{AUTO}$). The experiment shows that 1) $SETL_{CONSTRUCT}$ reduces the Number of Typed Characters (NOTC) of $SETL_{PROG}$ by 92% and $SETL_{AUTO}$ reduces the Number of Used Concepts (NOUC) by 25% compared to $SETL_{CONSTRUCT}$; 2) using $SETL_{CONSTRUCT}$, the development time is cut in almost half compared to $SETL_{PROG}$, which is further cut by another 27% using $SETL_{AUTO}$; 3) $SETL_{CONSTRUCT}$ is scalable and have similar performance compared to $SETL_{PROG}$.

An extension of this work is to define the formal semantics of the operations using an ontology and mathematically prove the correctness and completeness of the proposed ETL constructs. The source-to-target mappings act as a mediator to generate data according to the semantics encoded in the target. Therefore, we plan to extend our framework from purely physical to also virtual data integration where instead of materializing all source data in the DW, ETL processes will run on demand. When considering virtual data integration, it is important to develop query optimization techniques for OLAP queries on virtual semantic DWs, similar to the ones developed for virtual integration of data cubes and XML data [41, 42, 56]. Another interesting work will be to apply this layer-based integration process in a Big Data and Data Lake environment. The metadata produced in the Definition Layer can be used as basis to develop advanced catalog features for Data Lakes. Furthermore, we plan to investigate how to integrate provenance information into the process [2]. Another aspect of future work is the support of spatial data [19, 20]. We will also develop a new implementation with scalable, parallel ETL computing and optimize the operations.

A Appendix

A.1 Semantics of the Execution Layer Operations

In this section, we provide the detailed semantics of the ETL operations described in Section 6. The operations depend on some auxiliary functions. Here, we first present the semantics of the auxiliary functions and then the semantics of the operations. We present each function and operation in terms of input Parameters and semantics. To distinguish the auxiliary functions from the operations, we use small capital letters to write an auxiliary function name, while an operation name is written in italics.

Auxiliary Functions

EXECUTEQUERY(*Q, G, outputHeader*) This function provides similar functionality as SPARQL SELECT queries.

Input Parameters: Let I, B, L, and V be the sets of IRIs, blank nodes, literals, and query variables. We denote the set of RDF terms $(I \cup B \cup L)$ as T for an RDF graph G. V is disjoint from T. A query variable $v \in V$ is prefixed by the symbol $'?'$. A query pattern Q can be recursively defined as follows[16]:

1. An RDF triple pattern t_p is a query pattern Q. A t_p[17] allows query variables in any positions of an RDF triple, i.e., $t_p \in (I \cup B \cup V) \times (I \cup V) \times (T \cup V)$.
2. If Q_1 and Q_2 are query patterns, then $(Q_1\ AND\ Q_2)$, $(Q_1\ OPT\ Q_2)$, and $(Q_1\ UNION\ Q_2)$ are also query patterns.
3. If Q is a query pattern and F_c is a filter condition then $(Q\ FILTER\ F_c)$ is a query pattern. A filter condition is constructed using elements of the set $(T \cup V)$ and constants, the equality symbol (=), inequality symbols ($<, \geq, \leq, >$), logical connectivities ($\neg, \vee, \wedge$), unary predicates like bound, isBlank, and isIRI plus other features described in [46].

G is an RDF graph over which Q is evaluated and *outputHeader* is an output header list, which is a list of query variables and/or expressions over the

[16]We follow the same syntax and semantics used in [43] for a query pattern.

[17]To make it easily distinguishable, here we use comma to separate the components of a triple pattern and an RDF triple.

variables used in Q. Here, expressions are standard SPARQL expressions defined in [21].

Semantics: For the query pattern Q, let μ be a partial function that maps $var(Q)$ to T, i.e., $\mu : var(Q) \to T$. The domain of μ, denoted by $dom(\mu)$, is the set of variables occurring in Q for which μ is defined. We abuse notation and say $\mu(Q)$ is the set of triples obtained by replacing the variables in the triple patterns of Q according to μ. The semantics of the query pattern are recursively described below.

1. If the query pattern Q is a triple pattern t_p, then the evaluation of Q against G is the set of all mappings that can map t_p to a triple contained in G, i.e., $[\![Q]\!]_G = [\![t_p]\!]_G = \{\mu \mid dom(\mu) = var(t_p) \ \wedge \ \mu(t_p) \in G\}$.
2. If Q is $(Q_1 \ AND \ Q_2)$, then $[\![Q]\!]_G$ is the natural join of $[\![Q_1]\!]_G$ and $[\![Q_2]\!]_G$, i.e., $[\![Q]\!]_G = [\![Q_1]\!]_G \bowtie [\![Q_2]\!]_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![Q_1]\!]_G, \ \mu_2 \in [\![Q_2]\!]_G,$ $\mu_1 \ and \ \mu_2 \ are \ compatible \ mappings, i.e., \forall ?x \in dom(\mu_1) \cap dom(\mu_2), \mu_1(?x) = \mu_2(?x)\}$.
3. If Q is $(Q_1 \ OPT \ Q_2)$, then $[\![Q]\!]_G$ is the left outer join of $[\![Q_1]\!]_G$ and $[\![Q_2]\!]_G$, i.e., $[\![Q]\!]_G = [\![Q_1]\!]_G ⟕ [\![Q_2]\!]_G = ([\![Q_1]\!]_G \bowtie [\![Q_2]\!]_G) \cup ([\![Q_1]\!]_G \setminus ([\![Q_2]\!]_G)$.
4. If Q is $(Q_1 \ UNION \ Q_2)$, then $[\![Q]\!]_G = [\![Q_1]\!]_G \cup [\![Q_2]\!]_G = \{\mu \mid \mu \in [\![Q_1]\!]_G \ or \ \mu \in [\![Q_2]\!]_G \ \}$.
5. If $Q = (Q_1 \ FILTER \ F_c)$, then $[\![Q]\!]_G = \{\mu \mid \mu \in [\![Q_1]\!]_G \ \wedge \ \mu \models F_c\}$, where a mapping μ satisfies a filter condition F_c, denoted by $\mu \models F_c$.

This function returns a set of tuples where each tuple contains the values of the variables and/or the values derived by solving the expressions defined in *outputHeader* according to μ.

GETPROPERTIESFROMEXPRESSIONS*(sTBox, exps)* An expression is a combination of one or more properties, operators, and SPARQL functions defined in [21]. This function returns a subset of properties of a source TBox that are used in the given expressions.

Input Parameters: *sTBox* is a TBox and *exps* is a set of expressions.

Semantics: For each $exp \in exps$, this function computes the intersection of the set of IRIs used in exp and the set of properties of *sTBox*. Then, it returns the union of all intersection sets. Here, *returnIRIs(exp)* returns the set of IRIs used in exp and $\mathcal{P}(t)$ is defined in Equation B.2. The semantic is defined as follows:

$$\text{GETPROPERTIESFROMEXPRESSIONS}(sTBox, exps) = \bigcup_{exp \in exps} returnIRIs(exp) \cap \mathcal{P}(sTBox).$$

This function returns a set of properties.

VALIDATEEXPRESSIONS*(exps,Q, flag)* The source expressions/properties (defined by `map:source4TargetPropertyValue`) in property-mappings contains properties. However, the *outputHeader* parameter of EXECUTEQUERY() allows only query variables. Therefore to extract data from an RDF graph using EXECUTEQUERY(), the properties used in the expressions/properties of property-mappings should be replaced by corresponding query variables. On the other hand, to match the expressions/properties used in *outputHeader* of EXECUTEQUERY() with the expressions/properties (defined by `map:source4TargetPropertyValue`) in property-mappings , the query variables used in expressions/properties need to be replaced by the corresponding properties. Given a set of expressions, a query pattern, and a flag, this function replaces the used term (either IRI or query variables) in expressions with alternative ones from the query pattern based on the flag and returns a list of the validated expressions.

Input Parameters: $exps$ is a list of expressions, Q is a query pattern, and *flag* indicates whether it will replace the IRIs (*flag*=1) or query variables (*flag*=0) used in $exps$.

Semantics: If *flag=1* the function replaces the $exps$ IRIs with the corresponding query variables used in Q. For each IRI *iri* used in an expression, VALIDATEEXPRESSIONS replaces *iri* with the object of the triple pattern whose predicate is *iri* in Q. Finally, it returns the list of expressions, where each expression does not contain any IRIs.

If *flag=0* the function replaces the $exps$ query variables with the corresponding predicate IRIs used in Q. For each query variable *?q* used in an expression, it replaces *?q* with the predicate of the triple pattern whose object is *?q* in Q. Finally, it returns the list of expressions, where each expression does not contain any query variables.

MAPPEDSOURCEINSTANCES*(sc, sTBox, sABox, propertyMappings)* This function returns a dictionary describing the instances of a source concept.

Input Parameters: *sc* is a source construct, $sTBox$ and $sABox$ are the source TBox and ABox, $propertyMappings$ is a set of property-mappings.

Semantics: At first, this function retrieves instances of *sc* with their corresponding properties and values. Here, we consider only those properties that are directly mapped to target properties and/or used in source expressions. Both the properties and expressions are defined in propertyMappings by `map:source4TargetPropertyValue`. Then, it creates a dictionary, a set of $(key, value)$ pairs. In a pair, *key* represents an instance IRI and $value$ in turn represents a set of (p_i, v_i) pairs, where p_i represents a property and v_i

represents a value for p_i. It is explained as follows:

$$\textsc{mappedSourceInstances}(sc, sTBox, sABox, propertyMapping) = dictionary(\\ \textsc{executeQuery}((?i, \texttt{rdf:}type^{18}, sc) AND (?i, ?p, ?v) FILTER\\ (?p \in \textsc{getPropertiesFromExpressions}(sTBox,\\ mappedExpressions(sc, propertyMappings)))), sABox, (?i, ?p, ?v))).$$

Here, *mappedExpressions(sc, propertyMappings)* returns the set of source properties/expressions (defined by `map:source4TargetPropertyValue`) used in *propertyMappings*. The *dictionary((?i, ?p, ?v))* function first groups the input tuples by *?i* and then for each instance $i \in ?i$, it creates a set of (p_i, v_i) pairs, where p_i is a property of i and v_i is the value for p_i. Finally, MAPPEDSOURCE-INSTANCES(..) returns the dictionary created.

GENERATEIRI*(sIRI, value, tType, tTBox, iriGraph)* Every resource in an SDW is uniquely identified by an IRI defined under the namespace of the SDW. This function creates an equivalent target IRI for a source resource. Additionally, it keeps that information in the IRI graph.

Input Parameters: $sIRI$ is the source IRI, $value$ is the literal to be used to ensure the uniqueness of the generated IRI, $tType$ is the type of the generated IRI in $tTBox$, $tTBox$ is the target TBox, and $iriGraph$ is the IRI graph. The IRI graph is an RDF graph that keeps a triple for each resource in the SDW with their corresponding source IRI.

Semantics: Equation B.3 formulates how to create the IRI. First, the function checks whether there is an equivalent IRI for $sIRI$ in $iriGraph$ using $lookup(sIRI, iriGraph)$. It returns the target IRI if it finds an existing IRI in *iriGraph*; otherwise, it generates a new IRI by concatenating *prefix(tTBox)* and $validate(value)$ for a target concept or property, or creates an instance IRI by concatenating *tType* and $validate(value)$. Here, *prefix(tTBox)* returns the namespace of $tTBox$; $concat()$ concatenates the input strings; $validate(value)$ modifies $value$ according to the naming convention rules of IRIs described in [47]; $\mathcal{C}(T)$ and $\mathcal{P}(T)$ are defined in Equation B.1 and B.2. Upon the creation of the IRI, this function adds an RDF triple (tIRI `owl:sameAs` sIRI) to *iriGraph*.

generateIRI(sIRI, value, tType, tTBox, iriGraph)=

$$\begin{cases} lookup(sIRI, iriGraph) & if\ lookup(sIRI, iriGraph)! = NULL \\ concat(tType, \text{“\#”}, validate(value)) & if\ lookup(value, iriGraph) = NULL \wedge tType \in (\mathcal{C}(tTBox) \cup \mathcal{P}(tTBox)) \\ concat(prefix(tTBox), \text{“\#”}, validate(value)) & if lookup(sIRI, iriGraph) = NULL \wedge tType \notin (\mathcal{C}(tTBox) \cup \mathcal{P}(tTBox)) \end{cases} \quad \text{(B.3)}$$

This function returns an IRI.

<u>TUPLESTOTRIPLES*(T, tc, Q, propertyMappings, (i, $exp_1, .., exp_n$))*</u> As the canonical model of our integration process is RDF model, we need to convert the instance descriptions given in the tuple format into equivalent RDF triples. This function returns a set of RDF triples from the given set of tuples.

Input Parameters: T is a set of tuples (1st normal form tabular format), tc is a target construct, Q is a query pattern, *propertyMappings* is a set of property-mappings, and $(i, exp_1, .., exp_n)$ is the tuple format.

Semantics: Equation B.4 shows the semantics of this function. Each tuple of T represents an instance-to-be of the target construct tc, and each expression (exp_i) and the value of the expression ($val(exp_i)$) represent the corresponding target property and its value of the instance. Here, $val(exp_i)$ is the value of an expression in a source tuple. As an expression can contain query variables and expressions used in *propertyMappings* do not use query variables, VALIDATEEXPRESSIONS*(exp_i, Q, 0)* replaces the query variables with equivalent properties from Q, and *return(x, propertyMappings)* returns the target construct mapped to source construct x from *propertyMappings*.

$$\begin{aligned}
&\textsc{tuplesToTriples}(T, tc, Q, propertyMappings, (i, exp_1, .., exp_n)) = \\
&\bigcup_{(val(i), val(exp_1), .., val(exp_n)) \in T} \{\{(val(i), \texttt{rdf:type}, tc)\}\} \cup \\
&\bigcup_{p=1}^{n} \{(val(i), return(\textsc{validateExpressions} \\
&(exp_p, Q, 0), propertyMappings), val(exp_i))\}\} \qquad \text{(B.4)}
\end{aligned}$$

This function returns a set of RDF triples.

Execution Layer Operations

This section gives the semantics of each operation category-wise.

Extraction Operations

<u>*GraphExtractor(Q, G, outputPattern, tABox)*</u> This operation is functionally equivalent to SPARQL CONSTRUCT queries to extract data from sources.

Input Parameters: Q is a query pattern as defined in EXECUTEQUERY(). G is an RDF graph over which Q is evaluated. The parameter *outputPattern* is a set of triple patterns from $(I \cup B \cup var(Q)) \times (I \cup var(Q) \times (T \cup var(Q))$, where $var(Q) \subseteq V$ is the set of query variables occurring in Q. Here, I, B, T, and V are the sets defined in EXECUTEQUERY(). The parameter *tABox* is the location where the output of this operation is stored.

Semantics: This operation generates the output in two steps: 1) First, Q is matched against G to obtain a set of bindings for the variables in Q as discussed in ExecuteQuery(). 2) Then, for each variable binding (μ), it instantiates the triple patterns in $outputPattern$ to create new RDF triples. The result of the query is a merged graph, including all the created triples for all variable bindings, which will be stored in $tABox$. Equation B.5 formally defines it.

GraphExtractor(Q, G, outputPattern, tABox) =

$$\bigcup_{t_h \in outputPattern} \{\mu(t_h) \mid \mu \in [\![Q]\!]_G \ \wedge \ \mu(t_h) \ is\ a\ well-formed\ RDF\ triple\}. \tag{B.5}$$

Transformation Operations

TransformationOnLiteral(sConstruct, tConstruct, sTBox, sABox,propertyMappings, tABox) This operation creates a target ABox from a source ABox based on the expressions defined in property-mappings.

Input Parameters: *sConstruct* and *tConstruct* are a source and target TBox construct, *sTBox* and *sABox* are the source TBox and ABox, *propertyMappings* is a set of property-mappings, and *tABox* is the output location.

Semantics: First, we retrieve the instances of *sConstruct ins(c)* from *sABox* using the executeQuery() function, which is formally described as follows:

ins(c)=executeQuery*(q(c), sABox,(?i,* validateExpressions*(list(cElements), q(c), 1))*.

Here:

- $c = sConstruct$.
- *q(c)=(((?i,* `rdf:type`, *c) AND (?i,?p,?v)) FILTER (?p* $\in$ *cProperties))* is a query pattern.
- *cProperties=* getPropertiesFromExpressions*(sTBox, cElements)* is the set of source properties used in source expressions defined in *propertyMappings*.
- *cElements=*executeQuery*((?pm,* `map:source4TargetPropertyValue`, *?sp), propertyMappings, ?sp)* is the set of source expressions in *propertyMappings* defined by `map:source4TargetPropertyValue`.
- validateExpressions*(list(cElements),q(c),1)* replaces the source properties used in *cElements* with the corresponding query variables from *q(c)* as *outputHeader* parameter executeQuery() does not allows any properties. Since validateExpressions takes a list of expressions as a parameter, list(cElement) creates a list for *scElements*.

Now, we transform all tuples of $ins(c)$ into equivalent RDF triples to get $tABox$, i.e.,

$output = \text{tuplesToTriples}(ins(c), tConstruct, q(c), (?i, exp_1, , .., exp_n))$.

The output of this operation is $output$, a set of RDF triples, which will be stored in $tABox$.

JoinTransformation(sConstruct, tConstruct, sTBox, tTBox, sABox, tABox, comProp, propertyMappings) This operation joins and transforms the instances of source and target based on property-mappings.

Input Parameters: *sConstruct* and *tConstruct* are a source and target[19] TBox construct, $sTBox$ and $tTBox$ are the source and target TBoxes;; $sABox$ and $tABox$ are the source and target ABoxes; *comProp* is a set of common properties between *tConstruct* and *sConstruct* and $propertyMapping$ is the set of property-mappings.

Semantics: At first, we create an ABox by taking the set union of instances (including their properties and values) of both *sConstruct* and *tConstruct* and apply a query on the ABox using the EXECUTEQUERY() function. The query pattern of the function joins the instances of two concepts based on their common properties and finally the function returns a set of target instances *ins(sc,tc)* with the values of the source expressions (defined by `map:source4TargetPropertyValue`) in *propertyMappings*. As the query pattern includes triple patterns from both *sConstruct* and *tConstruct*, the source expression can be composed of both source and target properties. This is described as follows:

*ins(tc,sc)=*EXECUTEQUERY *((q(tc) OPT q(sc)), union(extractIns(tc,tABox), extractIns(sc,sABox)), (?i,* VALIDATEEXPRESSIONS*(scElements, (q(tc) OPT q(sc)), 1)).*

Here:

- *tc=tConstruct* and *sc=sConstruct*.
- *q(tc)=((*$?i_{tc}$*,* `rdf:type`*, tc) AND (*$?i_{tc}, ?p_{tc}, ?v_{tc}$*)* tp_{com}*(*$?i_{tc}$*,"target", comProp) FILTER(*$?p_{tc} \in$ *tcProperties))* is a query pattern.
  - $tp_{com}(?i_{tc}, target, comProp)$ is a function that joins a triple pattern (i.e., $AND\ (?i_{tc}, scom_i, ?scom_i)$) in $q(tc)$ for each pair $(scom_i, tcom_i) \in comProp$.
  - *tcProperties=* GETPROPERTIESFROMEXPRESSIONS*(tTBox, scElements)* represents the set of source properties used in source expressions.
- *q(sc)=((*$?i_{sc}$*,* `rdf:type`*, sc) AND (*$?i_{sc}, ?p_{sc}, ?v_{sc}$*)* sp_{com}*(*$?i_{sc}$*,"source", comProp) FILTER (*$?p_{sc} \in$ *scProperties))* is the query pattern.
  - $sp_{com}(?i_{sc},\text{"source"}, comProp)$ is a function that joins a triple pattern (i.e., $AND\ (?i_{sc}, tcom_i, ?scom_i)$) in $q(sc)$ for each pair $(scom_i, tcom_i)$ $\in comProp$.
  - *scProperties=* GETPROPERTIESFROMEXPRESSIONS*(sTBox, scElements)* represents the set of source properties used in source expressions.

[19]The term target here does not mean target schema but in the sense it is defined in concept-mappings. Source and target both can be either source concepts or intermediate concepts (from an intermediate result).

- *scElements*=EXECUTEQUERY*((?pm,* `map:source4TargetPropertyValue`*, ?sp), propertyMappings, ?sp)* is the set of source expressions in *propertyMappings* defined by `map:source4TargetPropertyValue`.
- *extractInstance(c, abox)*=EXECUTEQUERY*(((?i, rdf:type, c) AND (?i,?p,?v)), abox,(?i, ?p, ?v))* retrieves the instances (with their linked properties and values) of the concept *c* from the given ABox *abox*.
- $union(s1, s2)$ returns the set union of two given sets $s1$ and $s2$.
- VALIDATEEXPRESSIONS*(list(scElements), q(sc,tc) ,1)* replaces the source properties used in *scElements* with the corresponding query variables from *q(tc,sc)* as the *outputHeader* parameter of EXECUTEQUERY() does not allow any properties. Since VALIDATEEXPRESSIONS takes a list of expressions as a parameter, list(cElement) creates a list for *scElements*.

Now, we transform all tuples of $ins(sc, tc)$ into equivalent RDF triples to get the transformed $tABox$, i.e.,

output=TUPLESTOTRIPLES*(ins(tc,sc), tConstruct, q(tc,sc), (?i,* $exp_1, , .., exp_n$*))*.

The output of this operation is $output$, a set of RDF triples, which will be stored in $tABox$.

LevelMemberGenerator(sConstruct, level, sTBox, sABox, tTBox, iriValue, iriGraph, propertyMappings, tABox) The objective of this operation is to create QB4OLAP-complaint level members.

Input Parameters: *sConstruct* is the source construct, *level* is a target level, *sTBox* and *sABox* are the source TBox and ABox, *iriValue* is the rule of creating level members' IRIs, *iriGraph* is the IRI graph, *proeprtyMappings* is a set of property-mappings, and *tABox* is the output location.

Semantics: First, we retrieve the instances of *sConstruct* with their properties and values, i.e.,

$$Ins = \textsc{mappedSourceInstances}(sConstruct, sTBox, sABox, propertyMappings) \quad (B.6)$$

To enrich *level* with the dictionary *Ins*, we define a set of triples *LM* by taking the union of *IdentityTriples* (for each *(i,pv)* pair in *Ins*) and the union of *DescriptionTriples* (for each (p_i, v_i) pair in *pv*). Equation B.7 defines *LM*.

$$LM = \bigcup_{(i,pv)\in Ins} (IdentityTriples \cup \bigcup_{(p_i,v_i)\in pv} DescriptionTriple) \quad (B.7)$$

Here:

$$IdentityTriples = \{(lmIRI, \texttt{rdf:type}, \texttt{qb4o:LevelMember}), (lmIRI, \texttt{qb4o:memberOf}, level)\} \quad (B.8)$$

$$lmIRI = \begin{cases} i & if\ iriValue = \text{“}sameAs\ SourceIRI\text{”} \\ \textsc{generate}\text{IRI}(i, resolve(i, pv, iriValue), range(level, tTBox, iriGraph), tTBox) & if\ range(level, tTBox)! = NULL \\ \textsc{generate}\text{IRI}(i, resolve(i, pv, iriValue), level, tTBox, iriGraph) & if\ range(level, tTBox) = NULL \end{cases} \tag{B.9}$$

$$DescriptionTriple = \begin{cases} \{(lmIRI, return(p_i, property-Mappings),\ \textsc{generate}\text{IRI}(v_i, iriValue(v_i), range((return(v_i, propertyMappings), tTBox), tTBox, iriGraph))\} & if\ targetType(return(v_i, propertyMappings), tTBox) \in \{rollupProperty, ObjectP-roperty\} \\ \{(lmIRI, return(p_i, propertyMappings), v_i)\} & if\ targetType(p_i, propertyMap-pings, tTBox) = levelAttribute \end{cases} \tag{B.10}$$

Each instance of *sConstruct* is a `qb4o:LevelMember` and a member of *level*; therefore, for each instance of *sConstruct*, *LM* in Equation B.7 includes two identity triples, described in Equation B.8. Equation B.9 describes how to create an IRI for a level member. As we described in Section 5, the rule for creating IRIs can be different types. If *iriValue* is "sameAsSourceIRI" then the function returns the source IRI; otherwise it resolves value of *iriValue* using the *resolve(i, pv, iriValue)* function—this function returns either the value of a property/expression or next incremental value—, and finally it creates a new IRI by calling the GENERATEIRI() function. As some datasets (e.g., [18]) use the IRI of the level to create the IRIs of its members, whereas others (Eurostat (`https://ec.europa.eu/eurostat/data/database`), Greece (`http://linked-statistics.gr/`) linked datasets) use the IRI of the range of the level, we generalize it using Equation B.9. If a range exists for the target level, the IRI of that range is used as a target type— the type of the resources for which IRIs will be created— for creating IRIs for the members of that level (second case in Equation B.9); otherwise, the level's IRI is used as a target type (third case in Equation B.9). Here, $range(level, tTBox)$ returns the range of *level* from *tTBox*.

Equation B.10 creates a triple for each (p_i, v_i) pair of *Ins*. If p_i corresponds to either an object property or a rollup property in *tTBox*, then a new IRI is created for v_i and the target type of the IRI is the range of the target construct that is mapped to p_i (first case in Equation B.10). Here, $return(x, mappings)$ function returns the target construct that is mapped to x from the set of property-mappings *mappings*; $targetType(x, tTBox)$ returns the type of a target construct x from *tTBox*; and $iriValue(v)$ retrieves the value to be used to create the IRI. If v is a literal, it simply returns it, otherwise, it splits v by either "/" or "#" and returns the last portion of v. If p_i corresponds to a level attribute (i.e., datatype property), then the object of the triple generated for the level member will be the same as v (second case in Equation B.10).

The output of this operation is LM and the operation stores the output in $tABox$.

ObservationGenerator(sConstruct, dataset, sTBox, sABox, tTBox iriValue, iriGraph, propertyMappings, tABox): This operation creates QB4OLAP-complaint observations from the source data.

Input Parameters: *sConstruct* is the source construct, *dataset* is a target QB dataset, *sTBox* and *sABox* are the source TBox and ABox, *iriValue* is the rule of creating level members' IRIs, *iriGraph* is the IRI graph, *proeprtyMappings* is a set of property-mappings, and *tABox* is the output location.

Semantics: First, we retrieve the instances of *sConstruct* with their properties and values using Equation B.6. To populate *dataset* with the dictionary *Ins*, we define a set of triples *OB* which is equivalent to *LM* in Equation B.7.

$$IdentityTriples = \{(oIRI, \texttt{rdf:type}, \texttt{qb:Observation}), (oIRI, \texttt{qb:dataset}, dataset)\} \tag{B.11}$$

$$oIRI = \begin{cases} i & if\ iriValue = \text{“}sameAs\ SourceIRI\text{”} \\ \textsc{generate}\text{IRI}(i, resolve(i, pv, iriValue), dataset, tTBox, iriGraph) & otherwise \end{cases} \tag{B.12}$$

$$DescriptionTriple = \begin{cases} \{(oIRI, return(p_i, propertyMappings),\ lmIRI_o)\} & if\ targetType(return(p_i, propertyMappings), tTBox) = LevelProperty \\ \{(oIRI, return(p_i, propertyMappings),\ v_i)\} & if\ targetType(return(p_i, propertyMappings), tTBox) = MeasureProperty \end{cases} \tag{B.13}$$

$$lmIRI_o = \begin{cases} \textsc{generate}\text{IRI}(v_i, iriValue(v_i), range(return(p_i, property-Mappings), tTBox), tTBox, iriGraph) & if\ range(return(p_i, propertyM appings), tTBox))! = NULL \\ generateIRI(v_i, iriValue(v_i), return(p_i, propertyMappings), tTBox, iriGraph) & if\ range(return(p_i, propertyM appings), tTBox)) = NULL \end{cases} \tag{B.14}$$

Each instance of *sConstruct* is a `qb:Observation` and the QB dataset of that observation is *dataset*; therefore, for each instance of *sConstruct*, *OB* in Equation B.7 includes two identity triples. Equation B.11 redefines Equation B.8 for *OB*. Equation B.12 describes how to create an IRI for an observation. If *iriValue* is "sameAsSourceIRI" then the function returns the source IRI;

otherwise it generates an observation IRI by calling the GENERATEIRI() function. Equation B.13 redefines Equation B.10 for OB. For each (p_i, v_i) in the dictionary, the equation creates a triple. If p_i represents a level in the target, the predicate of the triple is the corresponding level of p_i and the object of the triple is a level member of that level. Equation B.14 shows how to create the IRI for the object (level member) of that triple. If p_i represents a measure property in the target, the predicate of the triple is the corresponding target measure property of $tTBox$ and the object of the target triple is similar to v_i.

The output of this operation is OB, which will be stored in $tABox$.

ChangedDataCapture(nABox, oABox, flag) This operation triggers either the SDW evolution (i.e., enriched with new instances) or update (i.e., reflect the changes in the existing instances).

Input Parameters: *nABox* and *oABox* are the sets of triples that represent the new source data and the old source data, *flag* indicates the type of difference the operation will produced. If *flag*= *0*, the operation produces the new instances, and if *flag*= *1*, the operation produces the set of updated triples (i.e., the triples in $oABox$ updated in $nABox$).

Semantics: Here, *nABox* and *oABox* both are a set of triples where first element of each triple corresponds to a level member in the target, second element represents a property (either a level attribute or a rollup property) and the third element represents a value of the property for that instance. First, we retrieve the sets of instances from *nABox* and *oABox* using the EXECUTEQUERY() function, shown as follows:

$Ins_{nABox} = \text{EXECUTEQUERY}((?i, \texttt{rdf:type}, ?v), nABox, ?i)$

$Ins_{oABox} = \text{EXECUTEQUERY}((?i, \texttt{rdf:type}, ?v), oABox, ?i)$

The set difference of Ins_{nABox} and Ins_{oABox} gets the new instances, i.e., $Ins_{new} = Ins_{nABox} - Ins_{oABox}$. We get the description (properties and their values) of new instances using Equation B.15.

$$InsDes_{new} = \bigcup_{i \in Ins_{new}} \text{EXECUTEQUERY}((?s, ?p, ?v) \\ FILTER(?s = i)), nABox, (?s, ?p, ?v)) \quad \text{(B.15)}$$

To get the triples that are changed over time, we first remove the description of new instances (derived in Eq. B.15) from *newABox* i.e., $InsDes_{old} = newABox - InsDes_{new}$; then get the set of changed triples by taking the set difference of $InsDes_{old}$ and $oldABox$, i.e.,

$ChangedTriples = InsDes_{old} - oldABox$.

If (*flag*=*0*) then the output is *InsDes*$_{new}$, else *ChangedTriples*. The output overwrites *oABox*.

updateLevel(level, updatedTriples, sABox, tTBox, tABox, propertyMappings, iriGraph) This operation reflect the changes in source to the SDW.

Input Parameters: *level* is a target level, *updatedTriples* is the set of updated triples (generated by the *ChangedDataCapture* operation), *sABox* is the source data of *level*, *tTBox* and *tABox* are the target TBox and ABox, *propertyMappings* is the set of property-mappings, and *iriGraph* is the IRI graph.

Semantics: As we consider that the changes only occur at the instance level, not at the schema level, only the 3rd elements (objects) of the triples of *sABox* and *updatedTriples* can be different. For the simplicity of calculation, we define *sABox*, and *updatedTriples* as the following relations:
sABox = (instance, property, oldVal)
updatedTriples = (instance, property, newVal)

This operation updates $tABox$ by deleting invalid triples from $tABox$ and inserting new triples that reflect the changes. As SPARQL does not support any (SQL-like) update statement, we define two sets *DeleteTriples* and *InsertTriples* that contain the triples to be deleted from $tABox$ and inserted to $tABox$, respectively. To get the final $tABox$, we first take the set difference between $tABox$ and $DeleteTriples$ using Equation B.16 and then, take the union of $tABox$ and $InsertTriples$ using Equation B.17.

$$tABox = tABox - DeleteTriples \tag{B.16}$$

$$tABox = tABox \cup InsertTriples \tag{B.17}$$

To get the updated objects and old objects of triples for source instances, we take the natural join between $updateddTriples$ and $sABox$, i.e.,

$$NewOldValues = updatedTriples \bowtie sABox \tag{B.18}$$

We define *DeleteTriples* and *InsertTriples* as

$$DeleteTriples = \bigcup_{(i,p,nV)\in\ updatedTriples} del(i,p,nV) \tag{B.19}$$

$$InsertTriples = \bigcup_{(i,p,nV,oV)\in\ NewOldValues} in(i,p,nV,oV) \tag{B.20}$$

The $del(i,p,nV)$ and $in(i,p,nV)$ functions depends on the update type of the level attribute corresponding to p. We retrieve the update type of the level attribute from $tTBox$ using Equation B.21. The *updateType(prop,tTBox)* function returns the update type of a level attribute *prop* from *tTBox* (see the blue-colored rectangle of Figure B.1).

$$updateType \;=\; (updateType(return(p, propertyMappings), tTBox)) \tag{B.21}$$

In the following, we describe how the functions, *del(i, p ,nV)* and *in(i,p,nV,oV)* are defined for different values of *updateType*.

If *updateType*= Type-1 update

If *updateType* is *Type-1 update*, the new value of the property will replace the old one. Therefore, the triple holding the old value for the property will be deleted from $tABox$ and a new triple with the updated value of the property will be inserted into $tABox$. Equation B.22 defines $del(i, p, nV)$ for the Type-1 update. For an instance i in the source, the existing equivalent IRI in the target can be retrieved from the IRI graph *iriGraph*, i.e., $IRI_i = lookup(i, iriGraph)$. *return(p, propertyMappings)* in Equation B.22 returns the target level attribute that is mapped to p from *propertyMappings*.

$$del(i, p, nV) = \textsc{executeQuery}(((?i, ?p, ?val)FILTER\ (?i = IRI_i\ \&\&\ ?p = return(p, propertyMappings))), tABox, (?i\ ?p\ ?val)) \quad \text{(B.22)}$$

Equation B.23 describes how to create an RDF triple for the Type-1 update. First, we retrieve the equivalent value of oV in the target using Equation B.24. As the retrieved value can be either a literal or an IRI, we create the object of the triple by replacing the oV portion of that value with nV. The *replace(org_str, search_pattern, replace_pattern)* function updates the string *org_str* by replacing the substring *search_pattern* with *replace_pattern*. The subject of the output RDF triple is the level member equivalent to *i* and the predicate is the target property equivalent to *p*.

$$in(i, p, nV, oV) = \{(IRI_i, return(p, mapping), replace(targetValue(i, p), oV, nV)\} \quad \text{(B.23)}$$

where,

$$targetValue(i, p) = \textsc{executeQuery}((IRI_i, return(p, mapping), ?v), tABox, ?v) \quad \text{(B.24)}$$

If *updateType*= Type-3 update

Like the Type-1 update, Type-3 update also replaces the old property value with the current one. Besides, it also keeps the old property value as the latest old property value. Therefore, the triples that contain the current and old property value should be removed from the target. Equation B.25 defines *del(i,p,nV)* for the Type-3 update, which retrieves the triples containing both current and old property values.

As besides new value Type-3 keeps the latest old value of the property by adding another property, we define *in(i,p,nV,oV)* in Equation B.26 which creates a triple for the new value and a triple for the old one. For a property *property*, *concat(property, "_oldValue")* creates a property to contain the old value of the *property* by concatenating the "*_oldValue*" with *property*. The function $targetValue(i, p)$ returns the objects of triple whose subject and predicate are corresponds to *i* and *p*, defined in Equation B.24.

$$del(i,p,nV) = \textsc{executeQuery}(((?i,?p,?val)FILTER(?i = IRI_i \,\&\& \\ ?p \in (return(p, propertyMappings), concat(return(p, \\ propertyMappings), \text{“\_oldValue”}))), tABox, (?i,?p,?val)) \quad \text{(B.25)}$$

$$in(i,p,nV,oV) = \{(IRI_i, return(p, mapping), replace(targetValue(i,p), \\ oV, nV), (IRI_i, concat(return(p, propertyMappings), \text{“\_oldValue”}), \\ targetValue(i,p))\} \quad \text{(B.26)}$$

If *updateType*= Type-2 update

In Type-2 update, a new version for the level member is created (i.e., it keeps the previous version and creates a new updated version). Since the validity interval (defined by `type2:toDate`) and status (defined by `type2:status`) of the previous version need to be updated, triples describing the validity interval and status of the previous version should be deleted from $tABox$. Equation B.27 defines $del(i,p,nV)$ for Type-2 update. The first operand of the union in Equation B.27 retrieves the expired triples of IRI_i (the level member corresponding to i) from $tABox$. As the level of IRI_i can be an upper level in a hierarchy, the validity intervals and the status of the level members of lower levels referring to IRI_i need to be updated, too. Therefore, the current validity intervals and status of the associated level members will also be deleted. The second operand of union in Equation B.27 $getExpiredTriplesAll(IRI_i)$ returns the set of expired triples of the members of all lower levels referring to IRI_i (described in Equation B.28). The function $getTriplesImmediate(IRI_i)$ returns the set of expired triples of the members of immediate child level referring to IRI_i (described in Equation B.29).

$$del(i,p,nV) = \textsc{executeQuery}(((IRI_i,?p,?val)FILTER\ (?p \in (\texttt{type2:} \\ \texttt{toDate}, \texttt{type2:status}))), tABox, (IRI_i,?p,?val)) \cup \\ getExpiredTriplesAll(IRI_i) \quad \text{(B.27)}$$

$$getExpiredTriplesAll(IRI_i) = \\ \bigcup_{(i_c\ p\ v)\, \in\, getExpiredTriplesImmediate(IRI_i)} getExpiredTriplesAll(i_c) \quad \text{(B.28)}$$

$$getExpiredTriplesImmediate(IRI_i) = \textsc{executeQuery}((?i_c,?p,IRI_i)\ AND \\ (?i_c, \texttt{rdf:type}, \texttt{qb4o:LevelMember})\ AND\ (?i_c,?p,?v)\ FILTER\ (?p \in \\ (\texttt{type2:toDate}, \texttt{type2:status}))), tABox, (?i_c,?p,?val)) \quad \text{(B.29)}$$

To reflect the changes in the target, a new version of the level member (with the changed and unchanged property values) is created. Equation B.30 defines *in(i,p,nV,oV)* for Type-2 update. The IRI for the new version of IRI_i is $newIRI_i$ = *updateIRI*(IRI_i, *iriGraph*). The *updateIRI(iri, iriGraph)* function updates the existing IRI *iri* by appending the current date with it and returns the updated one. An RDF triple is generated for nV, where the object is composed by replacing the old value with the new value, the predicate is the target level attribute equivalent to *p*, and the subject is $newIRI_i$ (first operand of the union operation in Equation B.30). The call of *update2Insert*(iri_o, iri_n,*cp*) in Equation B.30 returns the triples required to reflect the changes in $tABox$. Here, iri_o and iri_n are the old and new version (IRI) of a level member and *cp* is the predicate of the updated triple (i.e., the property whose value has been changed). The values of `type2:status` for new and existing versions are set to “Current" and “Expired", respectively. The validity interval of existing version is ended on *sysdate()-1* and the validity interval of new version is from *sysdate()* to “9999-12-31". The *sysdate()* returns the current date in *year-month-day* format as we consider that at most one update operation can be applied on the SDW in a day. The value “9999-12-31" in the `type2:toDate` indicates that the instance is still valid; this is a usual notation in temporal databases [54]. As the new version of the level member also contains the triples for its unchanged properties, we retrieve the triples of a level member's old version iri_o with unchanged properties using *retrieveTriples*(iri_o, cp) which returns all triples of iri_o except the triples with the properties *cp* (changed property), `type2:toDate, type2:fromDate`, and `type2:status` (described in Equation B.32). We replace the IRI of old version with the new one using the *replace()* function. To propagate the current version to its descendants (i.e., the level members those are associated to iri_o); the function *updateAssociates*(iri_o, iri_n) creates new versions for the associated members of all lower levels in the hierarchy so that they refer to the correct version (described in Eq. B.33). The function *getAssociates*(iri_o) returns the set of triples that describe the members that are connected to iri_o with a property. For each member connected to iri_o, we create a new version of it and also recursively update the members dependent on it using Equation B.35.

$$\begin{aligned} in(i,p,nV,oV) = \{(&newIRI_i, return(p, propertyMappings),\\ &replace(targetValue(i,p), oV, nV)\} \cup\\ &update2Insert(IRI_i, newIRI_i, return(p,\\ &\qquad propertyMappings)) \end{aligned} \tag{B.30}$$

$$\begin{aligned}update2Insert(iri_o, iri_n, cp) = \{&(iri_o, \texttt{type2:status}, \text{“}Expired\text{”}),\\ &(iri_o, \texttt{type2:toDate}, sysdate() - 1), (iri_n, \texttt{type2:fromDate}, sysdate()),\\ &(iri_n, \texttt{type2:toDate}, \text{“}9999-12-31\text{”}), (iri_n, \texttt{type2:status}, \text{“}Current\text{”})\}\\ &\cup\ replace(retrieveTriples(iri_o, cp), iri_o, iri_n)) \cup\ updateAssociates(iri_o, iri_n)\end{aligned} \tag{B.31}$$

$$\begin{aligned}retrieveTriples(lm, cp) = \textsc{executeQuery}(((&lm, ?p, ?v)\ FILTER\ (?p \notin\\ &(cp, \texttt{type2:toDate}, \texttt{type2:fromDate}, \texttt{type2:status}))),\\ &tABox, (lm, ?p, ?v))\end{aligned} \tag{B.32}$$

$$updateAssociates(iri_o, iri_n) = \bigcup_{(i_c,p)\in getAssociates(iri_o)} recursiveUpdate(i_c, p) \tag{B.33}$$

$$\begin{aligned}getAssociates(iri_o) = \textsc{executeQuery}(((&?i_c, ?p, iri_o)\ AND\ (?i_c, \texttt{rdf:type},\\ &\texttt{qb4o:LevelMember})), tABox, (?i_c, ?p))\end{aligned} \tag{B.34}$$

$$\begin{aligned}recursiveUpdate(i_c, p) = \{&(updateIRI(i_c, iriGraph), p, i_c)\} \cup\\ &update2Insert(i_c, updateIRI(i_c, iriGraph), p)\end{aligned} \tag{B.35}$$

This operation updates the target ABox, $tABox$.

We refer to [37] for the semantics of the *ExternalLinking* operation and to [22] for the semantics of the *MaterializeInference* operation.

References

[1] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, and A. Simitsis, “Using Semantic Web Technologies for Exploratory OLAP: A Survey,” *IEEE transactions on knowledge and data engineering*, vol. 27, no. 2, pp. 571–588, 2014.

[2] K. Ahlstrøm, K. Hose, and T. B. Pedersen, “Towards Answering Provenance-Enabled SPARQL Queries Over RDF Data Cubes,” in *Joint International Semantic Technology Conference*. Springer, 2016, pp. 186–203.

[3] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *Joint International Semantic Technology Conference*. Springer, 2014, pp. 178–186.

[4] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge university press, 2003.

[5] S. K. Bansal, "Towards a Semantic Extract-Transform-Load (ETL) Framework for Big Data Integration," in *Big Data*, 2014, pp. 522–529.

[6] L. Bellatreche, S. Khouri, and N. Berkani, "Semantic Data Warehouse Design: From ETL to Eeployment A La Carte," in *International Conference on Database Systems for Advanced Applications*. Springer, 2013, pp. 64–83.

[7] R. Berlanga, O. Romero, A. Simitsis, V. Nebot, T. Pedersen, A. Abelló Gamazo, and M. J. Aramburu, "Semantic Web Technologies for Business Intelligence," 2011.

[8] M. Casters, R. Bouman, and J. Van Dongen, *Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration*. John Wiley & Sons, 2010.

[9] C. Ciferri, R. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi, "Cube Algebra: A Generic User-centric Model and Query Language for OLAP Cubes," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 9, no. 2, pp. 39–65, 2013.

[10] D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatiş, "RDF Aanalytics: Lenses over Semantic Graphs," in *WWW*. ACM, 2014, pp. 467–478.

[11] P. Cudré-Mauroux, "Leveraging Knowledge Graphs for Big Data Integration: the XI Pipeline," *Semantic Web*, no. Preprint, pp. 1–5.

[12] R. Cyganiak, D. Reynolds, and J. Tennison, "The RDF Data Cube Vocabulary," *W3C Recommendation, W3C (Jan. 2014)*, 2014.

[13] R. P. Deb Nath, K. Hose, T. B. Pedersen, O. Romero, and A. Bhattacharjee, "SETLBI: An Integrated Platform for Semantic Business Intelligence," in *Companion Proceedings of the Web Conference 2020*, 2020, pp. 167–171.

[14] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle, "RML: a generic language for integrated RDF mappings of heterogeneous data," 2014.

[15] L. Etcheverry, S. S. Gomez, and A. Vaisman, "Modeling and Querying Data Cubes on the Semantic Web," *arXiv preprint arXiv:1512.06080*, 2015.

[16] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP," in *DaWak*, 2014, pp. 45–56.

[17] L. Galárraga, K. Ahlstrøm, K. Hose, and T. B. Pedersen, "Answering Provenance-Aware Queries on RDF Data Cubes under Memory Budgets," in *International Semantic Web Conference*. Springer, 2018, pp. 547–565.

[18] L. Galárraga, K. A. M. Mathiassen, and K. Hose, "QBOAirbase: The European Air Quality Database as an RDF Cube," in *International Semantic Web Conference (Posters, Demos & Industry Tracks)*, 2017.

[19] N. Gür, J. Nielsen, K. Hose, and T. B. Pedersen, "GeoSemOLAP: Geospatial OLAP on the Semantic Web made easy," in *Proceedings of the 26th International Conference on World Wide Web Companion*, 2017, pp. 213–217.

[20] N. Gür, T. B. Pedersen, E. Zimányi, and K. Hose, "A Foundation for Spatial Data Warehouses on the Semantic web," *Semantic Web*, vol. 9, no. 5, pp. 557–587, 2018.

[21] S. Harris, A. Seaborne, and E. Prud'hommeaux, "SPARQL 1.1 Query Language," *W3C recommendation*, vol. 21, no. 10, 2013.

[22] A. Harth, K. Hose, and R. Schenkel, *Linked Data Management*. CRC Press, 2014.

[23] O. Hartig, K. Hose, and J. F. Sequeda, "Linked Data Management." 2019.

[24] T. Heath and C. Bizer, "Linked Data: Evolving the Web into a Global Data Space," *Synthesis lectures on the semantic web: theory and technology*, vol. 1, no. 1, pp. 1–136, 2011.

[25] M. Hilal, C. G. Schuetz, and M. Schrefl, "An OLAP Endpoint for RDF Data Analysis Using Analysis Graphs." in *ISWC*, 2017.

[26] 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*. Springer, 2014, pp. 114–132.

[27] ——, "Processing aggregate queries in a federation of sparql endpoints," in *European Semantic Web Conference*. Springer, 2015, pp. 269–285.

[28] ——, "Optimizing aggregate sparql queries using materialized rdf views," in *International Semantic Web Conference*. Springer, 2016, pp. 341–359.

[29] K. A. Jakobsen, A. B. Andersen, K. Hose, and T. B. Pedersen, "Optimizing rdf data cubes for efficient processing of analytical queries." in *COLD*, 2015.

[30] E. Kalampokis, B. Roberts, A. Karamanou, E. Tambouris, and K. A. Tarabanis, "Challenges on Developing Tools for Exploiting Linked Open Data Cubes." in *SemStats@ ISWC*, 2015.

[31] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *Extended Semantic Web Conference*. Springer, 2012, pp. 87–101.

[32] R. Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, Inc., 1996.

[33] T. Knap, P. Hanečák, J. Klímek, C. Mader, M. Nečaský, B. Van Nuffelen, and P. Škoda, "UnifiedViews: An ETL Tool for RDF Data Management," *Semantic Web*, vol. 9, no. 5, pp. 661–676, 2018.

[34] E. V. Kostylev, J. L. Reutter, and M. Ugarte, "CONSTRUCT Queries in SPARQL," in *18th International Conference on Database Theory (ICDT 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[35] J. Li, J. Tang, Y. Li, and Q. Luo, "Rimom: A Dynamic Multistrategy Ontology Alignment Framework," *IEEE Transactions on Knowledge and data Engineering*, vol. 21, no. 8, pp. 1218–1232, 2008.

[36] R. P. D. Nath, K. Hose, and T. B. Pedersen, “Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses,” in *Acm Eighteenth International Workshop on Data Warehousing and Olap (dolap 2015)*. Association for Computing Machinery, 2015.

[37] R. P. D. Nath, K. Hose, T. B. Pedersen, and O. Romero, “SETL: A Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses,” *Information Systems*, vol. 68, pp. 17–43, 2017.

[38] R. P. D. Nath, H. Seddiqui, and M. Aono, “Resolving Scalability Issue to Ontology Instance Matching in Semantic Web,” in *2012 15th International Conference on Computer and Information Technology (ICCIT)*. IEEE, 2012, pp. 396–404.

[39] R. P. D. Nath, M. H. Seddiqui, and M. Aono, “An Efficient and Scalable Approach for Ontology Instance Matching,” *JCP*, vol. 9, no. 8, pp. 1755–1768, 2014.

[40] V. Nebot and R. Berlanga, “Building Data Warehouses with Semantic Web Data,” *Decision Support Systems*, vol. 52, no. 4, pp. 853–868, 2012.

[41] D. Pedersen, J. Pedersen, and T. B. Pedersen, “Integrating XML Data in the TARGIT OLAP System,” in *Proceedings. 20th International Conference on Data Engineering*. IEEE, 2004, pp. 778–781.

[42] D. Pedersen, K. Riis, and T. B. Pedersen, “Query Optimization for OLAP-XML Federations,” in *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, 2002, pp. 57–64.

[43] J. Pérez, M. Arenas, and C. Gutierrez, “Semantics and Complexity of SPARQL,” in *International semantic web conference*, vol. 4273. Springer, 2006, pp. 30–43.

[44] I. Petrou and G. Papastefanatos, “Publishing Greek Census Data as linked open data,” *ERCIM News*, vol. 96, 2014.

[45] A. Polleres, A. Hogan, R. Delbru, and J. Umbrich, “RDFS and OWL Reasoning for Linked Data,” in *Reasoning Web. Semantic Technologies for Intelligent Data Access*. Springer, 2013, pp. 91–149.

[46] E. Prud, A. Seaborne *et al.*, “SPARQL Query Language for RDF,” 2006.

[47] L. Richardson and S. Ruby, *RESTful Web Services*. " O’Reilly Media, Inc.", 2008.

[48] J. Rouces, G. De Melo, and K. Hose, “Heuristics for Connecting Heterogeneous Knowledge via FrameBase,” in *European Semantic Web Conference*. Springer, 2016, pp. 20–35.

[49] ——, “FrameBase: Enabling Integration of Heterogeneous Knowledge,” *Semantic Web*, vol. 8, no. 6, pp. 817–850, 2017.

[50] M. H. Seddiqui, S. Das, I. Ahmed, R. P. D. Nath, and M. Aono, “Augmentation of Ontology Instance Matching by Automatic Weight Generation,” in *Information and Communication Technologies (WICT), 2011 World Congress on*. IEEE, 2011, pp. 1390–1395.

[51] J. F. Sequeda and D. P. Miranker, “Ultrawrap: SPARQL execution on relational data,” *Journal of Web Semantics*, vol. 22, pp. 19–39, 2013.

[52] D. Skoutas and A. Simitsis, “Ontology-Based Conceptual Design of ETL Processes for both Structured and Semi-structured Data,” *IJSWIS*, vol. 3, no. 4, pp. 1–24, 2007.

[53] M. Thenmozhi and K. Vivekanandan, "An Ontological Approach to Handle Multidimensional Schema Evolution for Data Warehouse," *International Journal of Database Management Systems*, vol. 6, no. 3, p. 33, 2014.

[54] A. Vaisman and E. Zimányi, *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[55] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen, "Dimensional Enrichment of Statistical Linked Open Data," *Journal of Web Semantics*, vol. 40, pp. 22–51, 2016.

[56] X. Yin and T. B. Pedersen, "Evaluating XML-Extended OLAP Queries Based on Physical Algebra," *Journal of Database Management (JDM)*, vol. 17, no. 2, pp. 85–116, 2006.

Paper C

$SETL_{BI}$: An Integrated Platform for Semantic Business Intelligence

Rudra Pratap Deb Nath, Katja Hose, Torben Bach Pedersen,
Oscar Romero, and Amrit Bhattacharjee

The paper has been published in the
WWW'20: Companion Proceedings of the Web Conference 2020
pp. 167–171, 2020. DOI: https://doi.org/10.1145/3366424.3383533

Abstract

With the growing popularity of Semantic Web technologies, more and more organizations natively manage data using Semantic Web standards, in particular RDF. This development gives rise to new requirements for Business Intelligence tools to enable analyses in the style of On-Line Analytical Processing (OLAP) over RDF data. In this demonstration, we therefore present the $SETL_{BI}$ (Semantic Extract-Transform-Load and Business Intelligence) integration platform that brings together the Semantic Web and Business Intelligence technologies. $SETL_{BI}$ covers all phases of integration: target definition, source to target mappings generation, semantic and non-semantic source extraction, data transformation, and target population and update. It facilitates Data Warehouse designers to build a semantic Data Warehouse, either from scratch or by defining a multi-dimensional view over existing RDF data sources, and further enables OLAP-style analyses.

1 Introduction

Business Intelligence tools allow integrating and analyzing data from multiple sources to facilitate business decisions, often in the style of On-Line Analytical Processing (OLAP). The integrated data are organized in a Data Warehouse, typically designed following the Multidimensional Model (MD), which represents data in terms of facts and dimensions. As source data can be structured, semi-structured, unstructured, or semantic, it is important to consider semantic issues in the integration process, which is typically ignored by traditional (RDBMS-centric) data integration tools [1]. On the other hand, initiatives such as Open Government Data (`https://opengovdata.org/`) encourage organizations to publish their data using standards and non-proprietary formats [16]. Semantic Web standards fulfill these needs as they allow adding semantics on both data and schema level in the integration process and publish data in RDF using Linked Data principles [5]. To bridge this gap, we present $SETL_{BI}$, a tool that combines Semantic Web and Business Intelligence technologies to define, process, integrate, and query semantic data.

In the remainder of this paper, we refer to a semantically annotated Data Warehouse as a *semantic* DW and represent it as a knowledge base in RDF composed of two components: ABox and TBox [12]. The TBox defines a domain in terms of concepts, properties, and terminological axioms whereas the ABox consists of assertions of the TBox. To define the expressivity of the knowledge base with MD semantics, $SETL_{BI}$ uses the Data Cube (QB) [4] and QB4OLAP [6] vocabularies. In doing so, we can elegantly define a TBox with essential Data Warehouse concepts, such as cube structures, dimensions, levels, level attributes, OLAP operations, and complex hierarchies. On the other hand, $SETL_{BI}$ also allows to enrich the TBox with RDFS/OWL classes and properties. To create the ABox from (non-)semantic sources, we introduce a set of basic semantic ETL operations that can be connected and pipelined to orchestrate an ETL flow from the data sources to the *semantic* DW. Finally, $SETL_{BI}$ provides an interactive interface to enable self-service OLAP analysis over the *semantic* DW.

In summary, $SETL_{BI}$ enables (i) users with a background in Data Warehousing but little-to-no background in Semantic Web technologies to semantically integrate semantic and/or non-semantic data and analyze it in OLAP-style, and (ii) users with basic background in Semantic Web and Data Warehouse technologies to define multidimensional views over semantic data and run OLAP-like analysis. Additionally, users can enrich the generated TBox with RDFS and OWL constructs.

2 Related Work

Nowadays, the fusion of Semantic Web and Data Warehouse technologies has become popular. There are two lines of research in this direction: 1) use of ontologies as a canonical data model to physically integrate heterogeneous sources, and 2) specific approaches enabling OLAP analysis over semantic data.

A prominent study [14] related to the first line presents an ontology-based approach for facilitating the construction of an ETL flow. At first, each schema of (structured or semi-structured) data sources and the data warehouse is described by a common graph-based model named the datastore graph. Then, an (OWL-based) application ontology is generated to describe the semantics of the datastore graphs of data sources and Data Warehouse, and mappings between the sources and the Data Warehouse are formulated through the ontology. In this way, they have resolved the heterogeneity issues among the source and target schemata and finally showed how the use of ontologies enables a high degree of automation of ETL flows. However, the integrated data are not annotated with MD semantics to enable OLAP queries.

Related work [13] addressing the second line of research presents a semi-automatic method for inclusion of Semantic Web data into a traditional MD data management system for OLAP analysis. The authors present a methodology which allows *a*) the analyst to design the MD schema from the TBox of an RDF dataset, *b*) to populate the MD fact table after extracting the facts from the ABox of the dataset, *c*) to generate the dimension hierarchies from instances of the fact tables and the TBox so that it enables MDX queries over the data warehouse. However, they have not shown how to update the Data Warehouse with the change of data in a RDF data source. Most importantly, the generated Data Warehouse no longer maintains the Semantic Web data principles; therefore, OLAP-style analysis directly over an RDF dataset remains unaddressed. To address this issue, [3] has introduced a notion of lens called the analytical schema over the RDF dataset, which is a graph of classes and properties. Each node of the schema presents a set of facts that can be analyzed by exploring the reachable nodes.

To enable self-service OLAP analysis over RDF datasets, an OLAP endpoint has been presented in [7]. At first, they superimpose an MD schema over an RDF dataset. Then, semantic analysis graphs are built on top of the MD schema where each node of the graph presents an analysis situation corresponding to a MD query and an edge corresponds to a set of OLAP operations. However, it does not allow end-users to create their own OLAP queries. In [3] and [7], analysts need to create either a lens or a semantic analysis graph to define MD view over a RDF dataset. As a major portion of the published (statistical) Linked Data contains facts and figures, W3C recommends QB [4] vocabulary as a standard to describe data in a MD fashion.

[10] has investigated OLAP operations on a single data cube published with QB vocabulary and shown the applicability of their OLAP-to-SPARQL mapping in answering business questions. Although QB is appropriate to publish statistical data, it has limitations to represent MD semantics properly. QB4OLAP [6] extends QB by providing constructs to define a dimension structure (in terms of levels, the relationship between the levels, and hierarchies of the dimension), a cube structure in terms of different levels of dimensions, measures, and attaching aggregate functions with measures. In [15] and [16], the authors presented a method to semi-automatically enrich the QB dataset with QB4OLAP constructs. However, to run OLAP queries using the OLAP interface of their system, it requires end users to be familiar with either QL [2] or complex SPARQL queries.

[8, 9] has proposed a set of query processing strategies for executing OLAP-like SPARQL queries over a federation of SPARQL endpoints. Here, they use QB4OLAP constructs to annotate the conceptual global schema with MD semantics. However, participating sources in a federation might be unavailable at some point. Data and schemata of the sources might have evolved since the federation was created; thus, integration rules might no longer be valid or history of the data will be lost. Therefore, the standard approach is to avoid federation and have a local copy of the data which is the focus of this research.

The related approaches discussed in this section address one or more parts of our aimed problem, but there is no single solution that supports all the steps (target definition, mappings, ETL generation, target population, evolution and update) necessary to integrate heterogeneous data semantically in a *semantic* DW and enabling OLAP queries on it. $SETL_{BI}$ bridges the two lines of research by defining the TBox of a *semantic* DW with MD semantics using QB and QB4OLAP constructs, providing RDF-based semantic integration operations to populate/evolve/update the ABox of the *semantic* DW from heterogeneous sources and allowing users to create OLAP queries by using different MD constructs of the *semantic* DW.

3 System Architecture

The data integration process using $SETL_{BI}$ requires the following steps: 1) defining a target TBox with MD semantics using QB and QB4OLAP constructs, 2) generating mappings from sources to target, 3) populating the target ABox from the available data sources, and 4) issuing OLAP queries on the *semantic* DW. Based on the integration steps, we organize $SETL_{BI}$ into three layers: the Definition Layer, ETL Layer, and OLAP Layer, see in Figure C.1.

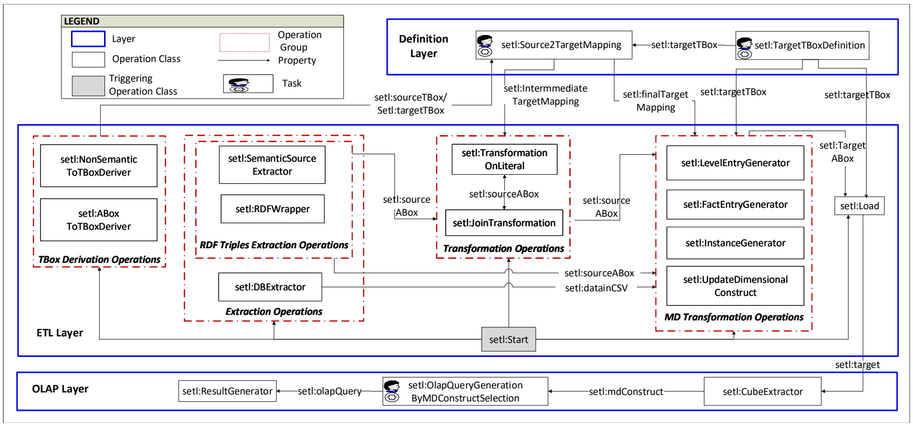


Fig. C.1: The ontology shows the components of each layer and their connections. The arrows from/to an operation group represent the arrows from/to each individual operation of the group.

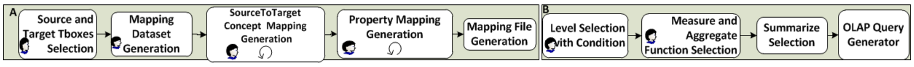

A
Source and Target Tboxes Selection
Mapping Dataset Generation
SourceToTarget Concept Mapping Generation
Property Mapping Generation
Mapping File Generation
B
Level Selection with Condition
Measure and Aggregate Function Selection
Summarize Selection
OLAP Query Generator

Fig. C.2: The workflows for Source2Target Mapping(A) and OLAP Query Generation (B) operation. The rectangles, user icons and curved arrows represent automatic, user-required and iterative tasks.

Each layer has a set of tasks and/or operations to accomplish certain integration steps. A task requires user interactions with the system's interface to produce an output while from the given inputs, an operation automatically produces an output. Intuitively, one may consider tasks as defining the required metadata to automate ETL operations. The Definition Layer covers the first two steps of the integration process and allows users to define the metadata (target schemas with MD semantics and semantic-aware mappings between the sources and target) of the integration process. The ETL Layer covers the third step of the integration process and includes a set of ETL operations to create data flows from sources to target. The OLAP Layer allows users to analyze *semantic* DW cubes using a GUI (the final step of the integration process). The inter- and intra-layer connections among the components (operations/tasks) of layers are shown in the ontology in Figure C.1 where each component is considered as a class and relationships between components are presented with arrows. In the following, we describe the layers in more detail.

The Definition Layer includes two tasks: `setl:TargetTBoxDefi nition` and `setl:Source2TargetMapping`. The former supports designing the TBox of a knowledge base with MD semantics and is implemented with a GUI where users can define/edit new/existing cubes, cuboids, dimensions, levels, level attributes, and measures. Internally, the operation annotates the user's input with QB, QB4OLAP and OWL constructs and generates an RDF file. Hence, it relieves TBox designers from the need to learn QB and QB4OLAP vocabularies. The latter also provides a GUI to create mappings across the constructs of (intermediate) source and target TBoxes. Intermediate mappings are required when there is a need of (string/numerical/ date) transformation on RDF literals or join of the sources. Hence, it relieves the user from the burden of manual mappings at the ETL operation level.

Figure C.1 shows how the intermediate mappings or final mappings are connected to the ETL Layer operations. The overall workflow of this operation is illustrated in Figure C.2 (A); internally, it creates a mapping file from the user's input in RDF format with our own OWL-based mapping vocabulary S2TMAP (`https://github.com/bi-setl/SETL`) that allows defining a property-level mapping under a concept-level mapping, which is in turn defined under a mapping dataset. Different to other ETL tools, this layer proposes a new paradigm: we characterize the ETL flow transformations at the Definition Layer instead of independently within each ETL operation (in the ETL layer). This way, the user has an overall view of the process, which generates metadata (the mapping file) that the ETL operators will read and parametrize themselves with automatically.

The ETL Layer is composed of a set of ETL operations. Based on their functionality, we categorize the operations into five groups: *TBox Derivation Operations*, *Extraction Operations*, *Transformation Operations*, *MD Transformation Operations*, and *Load operation*. In the *TBox Derivation Operations* group, `setl:NonSemanticToTBoxD eriver` derives TBoxes from non-semantic (CSV, XML, JSON and Database) sources and `setl:ABoxToTBoxDeriver` derives a TBox from an RDF file containing only assertions. In *Extraction Operations*, `setl:RDFWrapper` wraps up data from non-semantic sources to RDF triples; `setl:SemanticSourceExtractor` extracts RDF trip-les from an RDF data source, and `setl:DBExtractor` extract data in CSV format from a Database source. The *Transformation Operations* group supports numeric, string and date based transformation on the values of RDF properties according to the intermediate mappings using `setl:TransformationOnLiteral` and joins between two RDF sources using `setl:JoinTransformation`. The *MD Transformation Operations* group includes `setl:LevelEntryGenerator`, `setl: FactEntryGenerator`, and `setl:InstanceGenerator` to create level members, observations and instances to create the ABox of a *semantic* DW according to the semantics encoded in the TBox. Those operations support both RDF and CSV input. To reflect the changes in a source to the target, the `setl:UpdateDimensionalCo nstruct` operation updates the target level members accordingly. This operation supports three types of updates (Type 1, Type 2, and Type 3) defined by Ralph Kimball in [11] for a *semantic* DW. `setl:Load` loads RDF data into either a local RDF file or Jena TDB triple store. Users can drag and drop operations to create ETL flows.

The OLAP Layer takes a local RDF file or SPARQL endpoint containing a *semantic* DW, and allows users to create OLAP queries using a GUI. Users first extract the cube structure composed of dimensions, hierarchies, levels, measures and aggregate functions. Then, users create and issue OLAP queries to explore and aggregate measures at various level of details. Figure C.2 (B) shows how to create an OLAP query, similar to any traditional OLAP tool. Users can create slice and dice queries adding conditions on selected levels. Internally, we translate the OLAP query generated from the selections into an equivalent SPARQL query. Hence, the users are released of the burden of learning SPARQL. The system is developed in Java 8. All GUIs are implemented in SWT. To process, store and query RDF, we use Jena 3.4.0. As a triple store, we use Jena TDB.

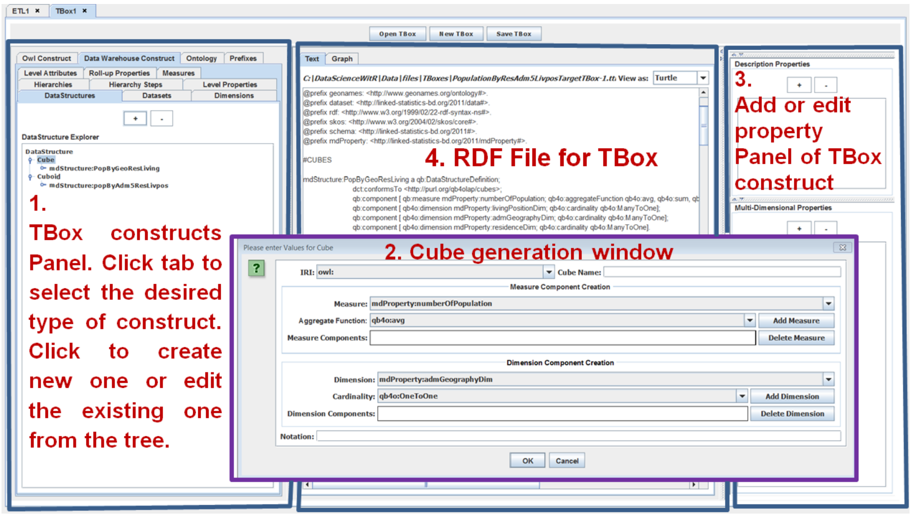
A comprehensive video of $SETL_{BI}$ and its functionality is available at `http://extbi.cs.aau.dk/SETLBI/index.php`. The source code is also available in `https://github.com/bi-setl/SETL`.

4 Demonstration

Inspired by the Linked Data principles, the Bangladesh Bureau of Statistics (BBS) wants to publish the 2011 population census in an OLAP-compliant Linked Open Data-format to enable decision making. The dataset consists of 12 smaller datasets where each of them contains approx. 130,000 observations. In this demonstration, we show how a user uses $SETL_{BI}$ layers to accomplish the integration steps (discussed in Section 3), starting with extracting census data in PDF format from the `http://203.112.218.65:8008/Census.aspx?MenuKey=89` BBS website and converting to CSV. Conference demo users can interact with the system as outlined below.

The Definition Layer: First, the user generates a TBox using the GUI shown in Figure C.3 which allows to create either a new TBox or edit an existing one. (S)he creates any TBox constructs using panel 1 and a cube using panel 2. The rightmost panel is the edit panel where any property with its corresponding values can be added or deleted. The middle panel shows the generated RDF model for the TBox in Turtle format.

The next task is to map the source and target TBoxes. First, the user creates a source TBox from the CSV files using `setl:NonSeman ticToTBoxDeriver` (ETL Layer). Figure C.4 shows how to map between source and target TBoxes. The left panel shows the tree of source TBox constructs while the right panel shows that of the target. The middle panel shows how to create a mapping between a source concept and a target cube. In short, this concept mapping tells under which map dataset it is, whether source instances are fully or partially mapped with the target, and how to generate target observation IRIs. Then, (s)he map properties of the source and target concepts by using



```
DataStructure
  Cube
    mdStructure:PopByGeoResLiving
  Cuboid
    mdStructure:popByAdm5ResLivpos
```

C:|DataScienceWitR|Data|files|TBoxes|PopulationByResAdm5LivposTargetTBox-1.tt View as: Turtle

```
@prefix geonames: <http://www.geonames.org/ontology#>.
@prefix dataset: <http://linked-statistics-bd.org/2011/data#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.
@prefix schema: <http://linked-statistics-bd.org/2011#>.
@prefix mdProperty: <http://linked-statistics-bd.org/2011/mdProperty#>.

#CUBES

mdStructure:PopByGeoResLiving a qb:DataStructureDefinition;
            dct:conformsTo <http://purl.org/qb4olap/cubes>;
            qb:component [ qb:measure mdProperty:numberOfPopulation; qb4o:aggregateFunction qb4o:avg, qb4o:sum, qb
            qb:component [ qb4o:dimension mdProperty:livingPositionDim; qb4o:cardinality qb4o:ManyToOne];
            qb:component [ qb4o:dimension mdProperty:admGeographyDim; qb4o:cardinality qb4o:ManyToOne];
            qb:component [ qb4o:dimension mdProperty:residenceDim; qb4o:cardinality qb4o:ManyToOne].
```

Please enter Values for Cube

IRI: owl: Cube Name:

Measure Component Creation

Measure: mdProperty:numberOfPopulation
Aggregate Function: qb4o:avg
Measure Components:

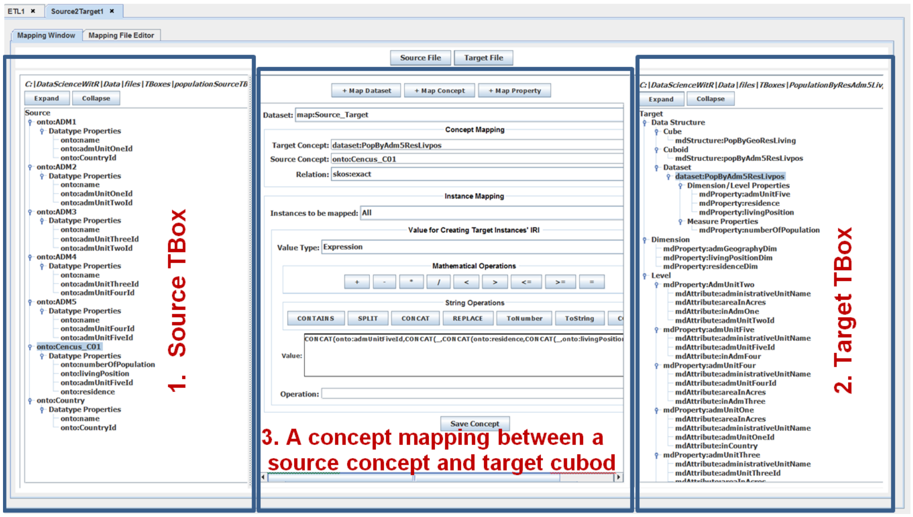
Dimension Component Creation

Dimension: mdProperty:admGeographyDim
Cardinality: qb4o:OneToOne
Dimension Components:

Notation:

Fig. C.3: The interface to define Target TBox.

Property Mapping window.



C:|DataScienceWitR|Data|files|TBoxes|populationSourceTB

```
Source
  onto:ADM1
    Datatype Properties
      onto:name
      onto:admUnitOneId
      onto:CountryId
  onto:ADM2
    Datatype Properties
      onto:name
      onto:admUnitOneId
      onto:admUnitTwoId
  onto:ADM3
    Datatype Properties
      onto:name
      onto:admUnitThreeId
      onto:admUnitTwoId
  onto:ADM4
    Datatype Properties
      onto:name
      onto:admUnitThreeId
      onto:admUnitFourId
  onto:ADM5
    Datatype Properties
      onto:name
      onto:admUnitFourId
      onto:admUnitFiveId
  onto:Cencus_C01
    Datatype Properties
      onto:numberOfPopulation
      onto:livingPosition
      onto:admUnitFiveId
      onto:residence
  onto:Country
    Datatype Properties
      onto:name
      onto:CountryId
```

Dataset: map:Source_Target

Concept Mapping

Target Concept: dataset:PopByAdm5ResLivpos
Source Concept: onto:Cencus_C01
Relation: skos:exact

Instance Mapping

Instances to be mapped: All

Value for Creating Target Instances' IRI

Value Type: Expression

Mathematical Operations

String Operations

Value: CONCAT(onto:admUnitFiveId,CONCAT(_,CONCAT(onto:residence,CONCAT(_,onto:livingPositio

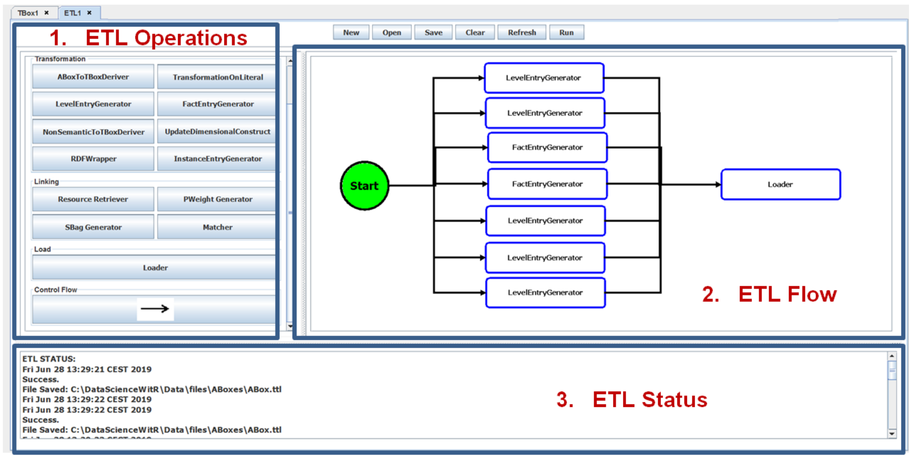
Operation:

C:|DataScienceWitR|Data|files|TBoxes|PopulationByResAdm5Liv

```
Target
  Data Structure
    Cube
      mdStructure:PopByGeoResLiving
    Cuboid
      mdStructure:popByAdm5ResLivpos
    Dataset
      dataset:PopByAdm5ResLivpos
        Dimension/Level Properties
          mdProperty:admUnitFive
          mdProperty:residence
          mdProperty:livingPosition
        Measure Properties
          mdProperty:numberOfPopulation
  Dimension
    mdProperty:admGeographyDim
    mdProperty:livingPositionDim
    mdProperty:residenceDim
  Level
    mdProperty:AdmUnitTwo
      mdAttribute:administrativeUnitName
      mdAttribute:areaInAcres
      mdAttribute:inAdmOne
      mdAttribute:admUnitTwoId
    mdProperty:admUnitFive
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitFiveId
      mdAttribute:inAdmFour
    mdProperty:admUnitFour
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitFourId
      mdAttribute:areaInAcres
      mdAttribute:inAdmThree
    mdProperty:admUnitOne
      mdAttribute:areaInAcres
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitOneId
      mdAttribute:inCountry
    mdProperty:admUnitThree
      mdAttribute:administrativeUnitName
      mdAttribute:admUnitThreeId
```

Fig. C.4: The interface to define mappings.

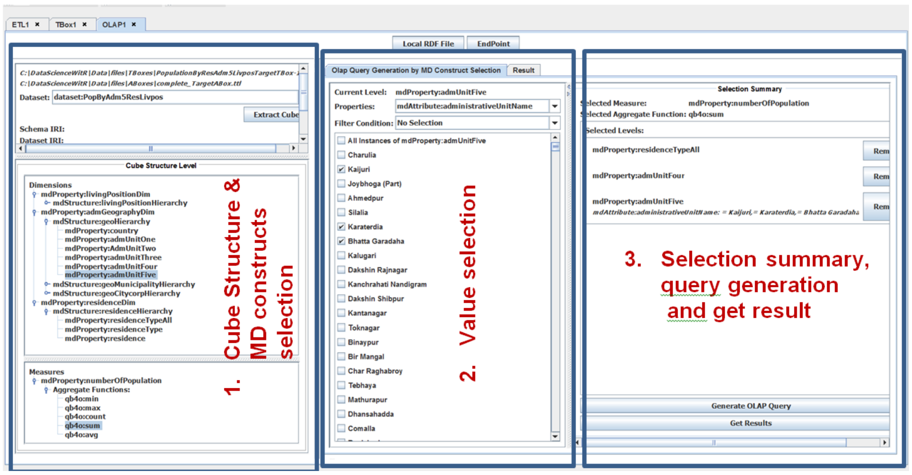
The ETL Layer: Figure C.5 shows how to create an ETL flow for populating the target ABox. *LevelEntryGenerator* and *FactEntryGenerator* are used to create Level members with their corresponding property values and observations according to the target semantics. The leftmost panel encapsulates the



```
ETL STATUS:
Fri Jun 28 13:29:21 CEST 2019
Success.
File Saved: C:\DataScienceWitR\Data\files\ABoxes\ABox.ttl
Fri Jun 28 13:29:22 CEST 2019
Fri Jun 28 13:29:22 CEST 2019
Success.
File Saved: C:\DataScienceWitR\Data\files\ABoxes\ABox.ttl
```

Fig. C.5: The interface to create ETL flows.

ETL operations. The user can drag and drop the operations in the ETL flow panel. The lowermost panel shows the ETL status.


ETL1
TBox1
OLAP1
Local RDF File
EndPoint
Extract Cube
Cube Structure Level
1. Cube Structure & MD constructs selection
Olap Query Generation by MD Construct Selection
Result
2. Value selection
Selection Summary
Rem
3. Selection summary, query generation and get result
Generate OLAP Query
Get Results

C:\DataScienceWitR\Data\files\TBoxes\PopulationByResAdm5LivposTargetTBox-
C:\DataScienceWitR\Data\files\ABoxes\complete_TargetABox.ttl
Dataset: dataset:PopByAdm5ResLivpos
Schema IRI:
Dataset IRI:

Dimensions
- mdProperty:livingPositionDim
  - mdStructure:livingPositionHierarchy
- mdProperty:admGeographyDim
  - mdStructure:geoHierarchy
    - mdProperty:country
    - mdProperty:admUnitOne
    - mdProperty:AdmUnitTwo
    - mdProperty:admUnitThree
    - mdProperty:admUnitFour
    - mdProperty:admUnitFive
  - mdStructure:geoMunicipalityHierarchy
  - mdStructure:geoCitycorpHierarchy
- mdProperty:residenceDim
  - mdStructure:residenceHierarchy
    - mdProperty:residenceTypeAll
    - mdProperty:residenceType
    - mdProperty:residence

Measures
- mdProperty:numberOfPopulation
  - Aggregate Functions:
    - qb4o:min
    - qb4o:max
    - qb4o:count
    - qb4o:sum
    - qb4o:avg

Current Level: mdProperty:admUnitFive
Properties: mdAttribute:administrativeUnitName
Filter Condition: No Selection

- [ ] All Instances of mdProperty:admUnitFive
- [ ] Charulia
- [x] Kaijuri
- [ ] Joybhoga (Part)
- [ ] Ahmedpur
- [ ] Silalia
- [x] Karaterdia
- [x] Bhatta Garadaha
- [ ] Kalugari
- [ ] Dakshin Rajnagar
- [ ] Kanchrahati Nandigram
- [ ] Dakshin Shibpur
- [ ] Kantanagar
- [ ] Toknagar
- [ ] Binaypur
- [ ] Bir Mangal
- [ ] Char Raghabroy
- [ ] Tebhaya
- [ ] Mathurapur
- [ ] Dhansahadda
- [ ] Comalla

elected Measure: mdProperty:numberOfPopulation
elected Aggregate Function: qb4o:sum
elected Levels:
mdProperty:residenceTypeAll
mdProperty:admUnitFour
mdProperty:admUnitFive
mdAttribute:administrativeUnitName: = Kaijuri, = Karaterdia, = Bhatta Garadaha

Fig. C.6: The interface to create and run OLAP queries.

The OLAP Layer: Then the user uses Figure C.6 to load the *semantic* DW from a SPARQL endpoint or a local RDF file. The leftmost panel shows the cube structure of the corresponding dataset. (S)he can roll-up, drill-down by clicking the desired dimension hierarchy levels. The middle panel allows slicing and dicing according to the property values. The right panel shows the summary of the selections. Then the user generate the equivalent OLAP query and finally click *Get Result* to show the result panel.

Acknowledgements

This research is partially funded by the European Commission through the Erasmus Mundus Joint Doctorate Information Technologies for Business Intelligence (EM IT4BI-DC), the Poul Due Jensen Foundation, and the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301B.

References

[1] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *IEEE TKDE*, vol. 27, no. 2, pp. 571–588, 2014.

[2] C. Ciferri, R. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi, "Cube algebra: A Generic User-Centric Model and Query Language for OLAP Cubes," *IJDWM*, vol. 9, no. 2, pp. 39–65, 2013.

[3] D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatiş, "RDF Analytics: Lenses over Semantic Graphs," in *WWW*. ACM, 2014, pp. 467–478.

[4] R. Cyganiak, D. Reynolds, and J. Tennison, "The RDF Data Cube Vocabulary," *W3C Recommendation, W3C (Jan. 2014)*, 2014.

[5] R. P. Deb Nath, K. Hose, and T. B. Pedersen, "Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses," in *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*, 2015, pp. 15–24.

[6] L. Etcheverry and A. A. Vaisman, "QB4OLAP: A New Vocabulary for OLAP Cubes on The Semantic Web," *COLD*, 2012.

[7] M. Hilal, C. G. Schuetz, and M. Schrefl, "An OLAP Endpoint for RDF Data Analysis Using Analysis Graphs." in *ISWC*, 2017.

[8] 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*. Springer, 2014, pp. 114–132.

[9] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi, "Processing Aggregate Queries in a Federation of SPARQL Endpoints," in *ESWC*, vol. 9088, 2015, pp. 269–285.

[10] B. Kämpgen, S. O'Riain, and A. Harth, "Interacting with Statistical Linked Data via OLAP Operations," in *ESWC*. Springer, 2012, pp. 87–101.

[11] R. Kimball, *The data warehouse toolkit: practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc., 1996.

[12] R. P. D. Nath, K. Hose, T. B. Pedersen, and O. Romero, "SETL: A Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses," *Information Systems*, vol. 68, pp. 17–43, 2017.

[13] V. Nebot and R. Berlanga, "Building Data Warehouses with Semantic Web Data," *Decision Support Systems*, vol. 52, no. 4, pp. 853–868, 2012.

[14] D. Skoutas and A. Simitsis, "Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data," *IJSWIS*, vol. 3, no. 4, pp. 1–24, 2007.

[15] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, and C. Thomsen, "QB2OLAP: Enabling OLAP on Statistical Linked Open Data," in *ICDE*. IEEE, 2016, pp. 1346–1349.

[16] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen, "Dimensional Enrichment of Statistical Linked Open Data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 40, pp. 22–51, 2016.

Paper D

Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses

Rudra Pratap Deb Nath, Katja Hose, and Torben Bach Pedersen

The paper has been published in the
Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP
pp. 15–24, 2015. DOI: http://dx.doi.org/10.1145/2811222.2811229

Abstract

In order to create better decisions for business analytics, organizations increasingly use external data, structured, semi-structured and unstructured, in addition to the (mostly structured) internal data. Current Extract-Transform-Load (ETL) tools are not suitable for this "open world scenario" because they do not consider semantic issues in the integration process. Also, current ETL tools neither support processing semantic-aware data nor create a Semantic Data Warehouse (DW) as a semantic repository of semantically integrated data. This paper describes SETL: a (Python-based) programmable Semantic ETL framework. SETL builds on Semantic Web (SW) standards and tools and supports developers by offering a number of powerful modules, classes and methods for (dimensional and semantic) DW constructs and tasks. Thus it supports semantic-aware data sources, semantic integration, and creating a semantic DW, composed of an ontology and its instances. A comprehensive experimental evaluation comparing SETL to a solution made with traditional tools (requiring much more hand-coding) on a concrete use case, shows that SETL provides better performance, knowledge base quality and porgrammer productivity.

1 Introduction

Business Intelligence (BI) tools support intelligent business decisions by analyzing available organizational data. Data Warehouses (DWs) are used to store the large data volumes from different operational databases in enterprises and On-Line Analytical Processing (OLAP) queries are applied on DWs to answer business questions. Extract-Transform-Load (ETL) is the backbone process of a DW. The strength of a DW depends on how good its ETL process is. Extraction retrieves data from appropriate data sources. Transformation converts the source data according to the target schema of the DW, typically either a star or snowflake schema. Loading stores the transformed data into the DW. DW/OLAP technologies perform efficiently when they are applied on data that are static in nature and well-organized in structure.

Nowadays, the Web is also an important source of business information. Moreover, Semantic Web (SW) technologies and the Linked Data (LD) principles inspire organizations to publish their business-related data using the Resource Description Framework (RDF) [29]. As a result, besides analyzing internal data available in a DW, it is often desirable to incorporate external data from various (semantic) sources into the DW to derive the needed business knowledge.

The inclusion of external data, especially RDF data, however, raises several challenges for integration and transformation in comparison to the traditional ETL process. One of the drawbacks of using RDF data in the corporate analysis process is that the data sometimes do not have any schema, or only have a poor or complex schema. Moreover, different sources describe the data in their own way, introducing heterogeneity problems. Therefore, to build a successful data warehouse system with these heterogeneous data, the integration process should emphasize the semantic relationship of the data. Traditional ETL tools are unable to process such external data because they (1) do not support semantic-aware data, (2) are entirely schema-dependent, (3) do not focus on meaningful semantic relationships to integrate data from disparate sources [4], and (4) do not support deriving new information by active inference and reasoning on the data. Thus, a DW with both internal and external (semantic) data requires more powerful tools to define, integrate and transform data semantically.

SW technology was introduced with the vision of converting the 'Web of Documents' to the 'Web of Data' where data are presented and exchanged in a machine-readable and understandable format, and data are integrated with each other semantically. The journey towards this vision is quite successful, and a number of standards, languages, and tools have been developed to express semantic-based metadata. To give a common framework for presenting and exchanging data in a machine-readable format in the SW, RDF is used for its simple and flexible format. In the RDF data model, information is represented in statements, called RDF triples. The three parts of a triple are subject, predicate, and object, respectively. An example triple is $(v_1, v_2, v_3) \in (I \cup B) \times I \times (I \cup B \cup L)$, where I, B, L are a set of IRIs, a set of Blank nodes and a set of Literals, respectively, and $(I \cap B \cap L) = \emptyset$ [12]. An IRI (Internationalized Resource Identifier) is a string used to uniquely identify a resource on the Web. To make the contents machine understandable, formal languages such as RDF Schema (RDFS) [30] and Web Ontology Language (OWL) [17] can be used in combination with the RDF data model to define schemata and ontologies. OWL also provides a rich set of vocabularies for defining meaningful relationships between resources and for deriving new information using description logic (DL) inferencing techniques.

Until now, the DW community has used SW technologies only for a few things [31], e.g., for data integration or describing the DW. Although [1, 4, 23] have used SW technologies to design conceptual frameworks of different phases of ETL processes, no one has so far offered an integrated and implemented framework to build a semantic DW that covers all of the phases of updating of sources, extraction, validation, and integration, in one integrated platform. This paper presents SETL, a unified framework for processing and integrating data semantically by bridging SW and DW technologies. SETL uses and extends SW tools and standards to overcome the limitations of the traditional ETL tools. The novel contributions of this paper are:

- We propose SETL, a unified Python-based programmable semantic ETL framework. SETL helps developers creating semantic DWs by offering following facilities: (a) In addition to traditional relational data, SETL allows including semantically annotated data (RDF data) in the analytical process. (b) SETL allows integrating and processing data from disparate data sources semantically. To integrate data from disparate data sources, the user can define an ontology for the DW. SETL also provides a method to produce semantic data (in RDF triples format) from the source data according to the semantics encoded in the ontology. (c) SETL creates a semantic DW, composed of an ontology and its instances, where data are semantically connected with other internal and/or external data.

- Using SETL, we perform a comprehensive experimental evaluation by (re-)producing a knowledge base (KB) that integrates a Danish Agricultural dataset and a Danish Business dataset. The evaluation shows that SETL improves considerably over an earlier solution (using standard technologies necessitating significant hand-coding) in terms of performance, knowledge base quality and programmer productivity.

The remainder of the paper is organized as follows. Section 2 outlines the source datasets and the target ontology of the DW that we use as the running example throughout the paper. Section 3 describes SETL and its components. We evaluate SETL in terms of performance, quality and productivity in Section 4. Section 5 describes related work. Finally, we conclude and give pointers to future work in Section 6.

2 A Use case

This section describes the source datasets, and the target ontology for integrating and understanding the knowledge of those datasets, used as the running example of the paper. We consider a Danish Agricultural dataset (DAD) [11] and a Danish Business dataset (DBD) [7]. By integrating the agricultural data with the business data and other external data, we (re-)produce a KB on which queries can be applied to get analytical information.

The Danish Agricultural dataset consists of three smaller datasets namely, *Field*, *Organic Field*, and *Field Block*. All the datasets are available in Shape [21] format. The *Field* dataset has 9 attributes and contains all registered fields in Denmark. This dataset overall contains information about 641,081 fields. The *Organic Field* dataset has 12 attributes and contains information about 52,060 organic fields. The *Field Block* dataset has 12 attributes for 314,648 field blocks [2].

The Danish Business dataset is provided in CSV format. The dataset consists of two sub-datasets namely, *Company* and *Participant*. The *Company* dataset consists of 59 attributes, and contains information about 603,667 companies and 659,639 production units. The *Participant* dataset describes the relations that exist between a participant and a legal unit [2]. Figure D.1 shows how the datasets are connected to each other.

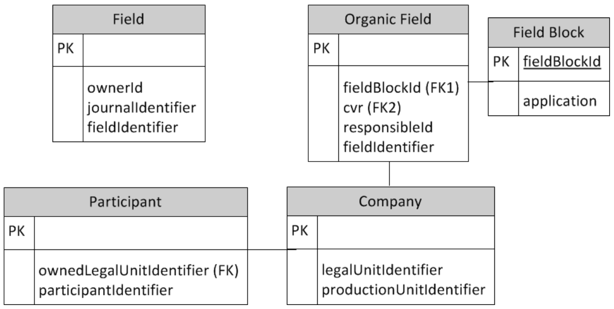


Fig. D.1: The schemas of the DAD and DBD datasets. Field, Field Block and Organic Field are spatially connected [2].

To integrate the datasets. we need to develop a target ontology. We choose using an ontology to design the DW schema rather than other models such

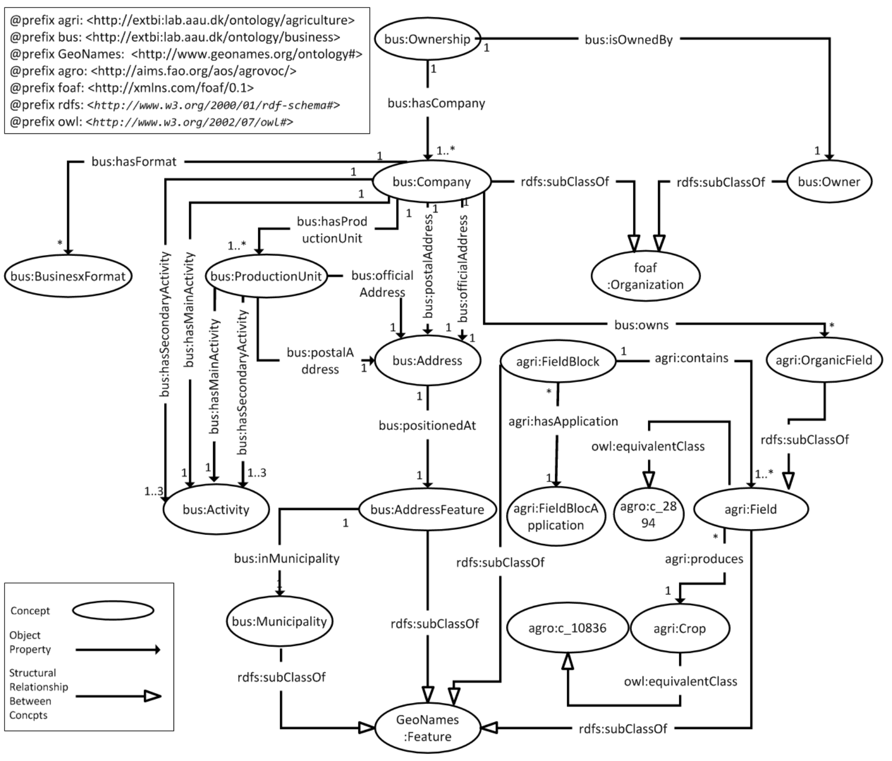


Fig. D.2: The target ontology of the Danish Agricultural dataset and Danish Business dataset. The number on an arrow indicates the cardinality of the relationship. Because of the large number of concept data type properties, they are not included in the figure.

as star or snowflake schema, or conceptual multidimensional model because an ontology allows to integrate disparate data sources semantically. In an ontology, a concept provides a general description of the properties and behavior for the similar type of resources; an object property relates among the instances of concepts; a data type property is used to associate the instances of a concept to literals. The target ontology used to integrate the Danish Agricultural dataset and the Danish Business dataset is illustrated in Figure D.2. This is an updated version of the ontology described in [2]. We start the description from the `bus:Ownership` concept. This concept contains information about the owners of companies, the type of the ownership, and the start date of the company ownership. This concept is related to the `bus:Owner` concept through the `bus:isOwnedBy` property. The `bus:Owner` includes the details of the owners. The `bus:Ownership` has a one-to-many relationship with the `bus:Company` concept and they are related to each other through the property `bus:hasCompany`. The `bus:Company` concept is related to the concepts `bus:BusinessFormat` and `bus:ProductionUnit`. The concepts bus:Company and bus:ProductionUnit are also related to the bus:Activity concept through one main activity and/or through one-three secondary activities. Each company and each production unit have a postal address and an official address. Therefore, both concepts are related with the bus:Address concept. Each address is positioned at an address feature, which is, in turn, contained within a particular municipality. The bus:Company concept is also related to the concept bus:OrganicField through the relation bus:owns as each company may contain one or more organic field. By this relation, the business and agricultural datasets are connected. The bus:Field concept contains all the registered agricultural fields of Denmark. Thus, bus:Organic Field is a subclass of bus:Field. The bus:Field is also equivalent to the UN definition of a European field. A field produces a crop. A field is contained within a field block. Each field block has an application. The concepts bus:AddressFeature, bus:Municipality, agri:Field, agri:FieldBlock are defined as subclasses of the GeoNames:Feature concept in the GeoNames ontology.

3 The SETL Framework

In this section, we present our Semantic ETL framework (SETL in short) that we implemented in Python. Our discussion focuses on SETL's architecture and its main methods that support the different phases of a semantic ETL process and produce a semantic data warehouse based on RDF triples. The main phases are: defining a target ontology based on the domain of interest, extracting data from multiple heterogeneous data sources, transforming the source data into RDF triples according to the target ontology, and loading the data into a triple store, and/or publishing the data on the Web as Linked Data.

Figure D.3 illustrates how SETL's components are connected to each other. SETL follows a demand-driven approach, i.e., a data warehouse is designed by identifying the information requirements of users and decision makers [5]. Based on the requirements, an ontology describing the relevant data and the target data warehouse can be defined using the *Ontology Definition* module. The *Define Mapping* module is used to define the mappings between the data sources and the target ontology. The *Extraction* module then extracts data from multiple sources, the *Traditional Transformation* module cleanses and formats the extracted data, and the *Semantic Transformation* converts the data into RDF triples according to the target ontology. The *External Linking* module links the resources in the created RDF dataset to other external resources. Finally, the *Writing* module can write the created RDF dataset into a file on disk and the *Loading* module can directly load the created RDF dataset into a triple store, which can directly be queried by the user. The following sections

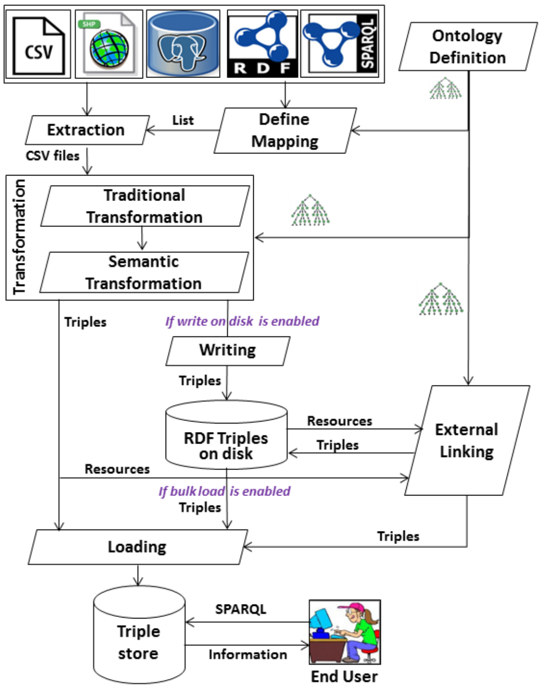


Fig. D.3: SETL architecture.

describe the components in more detail.

3.1 Ontology Definition

SETL uses an ontology to design a data warehouse schema rather than other models, such as a star schema, a snowflake schema, or a multidimensional model because of the following reasons. First, ontologies allow for a semantic integration of disparate data sources as we can explicitly define how two concepts of an ontology are structurally related, what type of relationship they have, and whether the relationship is symmetric, reflexive, or transitive. Second, unlike other models, multiple central concepts can easily be supported. Third, an ontology allows for including inference rules to derive new information based on the data that is available.

SETL allows to define various concepts, properties, and blank nodes of an ontology individually as well as to capture how they relate to each other. The user first defines concepts and properties in separate and then explicitly connects particular concepts to a set of properties. Internally, SETL indexes the set of properites connected to a particular concept. The following triples, for instance, represent the fact that `agri:produces` is an object property and its domain is the concept `agri:Field`.

```
agri:produces  rdf:type  owl:objectProperty.
agri:produces  rdfs:Domain  agri:Field.
```

Thus, internally each concept is explicitly connected to a set of properties, such as data type properties, object properties, functional and inverse functional properties.

3.2 Define Mapping

This component defines the mappings between the source datasets and the target ontology (defined using the *Ontology Definition* module described in Section 3.1). Based on this mapping, the extraction process retrieves data from different sources. We use methods from the Petl [18] library to define mappings between data sources and the target ontology.

3.3 Extraction

SETL enables the extraction of data from a knowledge base through SPARQL endpoints, RDF files, a search engine such as Sindice [27], relational databases, CSV files, and shapefiles. The framework is designed in a way that allows it to be easily extended to support other formats.

Extraction from a SPARQL endpoint. Using the SPARQLWrapper [24] library, we can extract data from a knowledge base by applying queries through a SPARQL endpoint. However, when the output of a given query is very big or if it is required to extract all triples from a knowledge base, a simple download strategy might fail because some SPARQL endpoints restrict result sizes; DBpedia [8], for instance, does not allow more than 10,000 result triples at a time.

Therefore, SETL provides a class named *ExtractTriplesFromEndpoint* which includes different methods to extract data and triples, for instance, by executing a query on a SPARQL endpoint. In a straightforward case, the query selects all triples but due to the result size limitation receives the results in batches. The different steps of the method are formalized in Algorithm 11. The algorithm takes a *sparqlEndpoint* URL and a *batchsize* (indicating how many triples to extract at a time) as parameters and returns the extracted triples. In line 1, the algorithm extracts all distinct properties in the knowledge base. Then, for each property (lines 2–3), the algorithm determines the number of triples containing the property as predicate. Based on this number the algorithm determines how many queries need to be sent to receive all these triples and executes the corresponding number of triples (lines 4–5). This is done by sorting the triples by subject, using *batchsize* in the LIMIT clause, and using the iteration number multiplied by the batchsize as OFFSET.

Algorithm 11: TripleExtraction.

```
    Input: $endpoint, batchsize$
    Output: $tripleset$
    begin
1       $P = retrieveDistinctProperty(endpoint)$ foreach $property\ p \in P$ do
3           $x = retrieveNumberOfTriples(p, endpoint)$;
4           for $count \leftarrow 0$ to $x/batchsize$ do
5               $tripleset+ = retrieveTriples(count, batchsize, endpoint)$;
6       return $tripleset$
```

Extraction from an RDF file. We use the RDFLib library [20] to execute a query on a local RDF file. If the RDF file is too large to fit into main memory, we split the file into several smaller files, execute the query on the smaller files, and then combine the partial results. However, this method can only be used for simply queries, such as retrieving all triples with a particular predicate. More complex queries involving joins need to be handled differently, e.g., by loading the data into a local SPARQL endpoint first and using the method described in the previous paragraph.

Extraction using semantic search engine. Currently, SETL supports extraction using the Sindice [27] search engine. To do so, the user provides a keyword or resource name that we want to retrieve information for. Based on this, SETL retrieves a list of relevant resource URLs from Sindice. The user can then manually select a list of URLs which are then used to retrieve triples involving them.

Extraction from a relational database or a CSV file. SETL is based on Petl [18], which is a traditional ETL Python package including a rich set of methods to perform extraction, cleansing, and transformation based on relational databases and CSV files. Making use of these methods, SETL provides the functionality as well.

Extraction from a shapefile. A shapefile [21] is a file format that stores geometric location and attribute information of geographic features in vector format. Points, lines, and polygons can be used to describe a geographic feature. To read data from a shapefile, we use the pyshape [19] library. Internally, SETL can store the data of a shapefile either in a file or in a database table. SETL provides methods to automatically create a table in a database based on the structure of a shapefile and to insert the data from the shapefile into the created table. It also provides a method to store the attribute information of a shapefile in CSV format.

Figure D.1 shows that there is no direct connection between *Field* and *OrganicField*, neither between *Field* and *FieldBlock*. In the target ontology

(Figure D.2), however, we have defined connections, e.g., `agri:Field` and `agri:OrganicField` are connected through the `owl:sameAs` property and `agri:FieldBlock` and `agri:Field` are connected via the `agri:contains` property. These connections are computed based on a spatial join that SETL computes during the extraction process for pairs of shapefiles. We use the ArcPy [3] library to perform the spatial join analysis.

3.4 Transformation

We divide the transformation tasks into two categories: *Traditional Transformation* and *Semantic Transformation*. *Traditional Transformation* includes operations known from traditional ETL tools, such as cleansing the data and formatting the source data according to the target schema. This includes removing duplicate data, normalizing data, renaming attributes, checking integrity constraints, refining data, creating new attributes based on existing attributes, sorting data, and grouping data [4]. SETL is using the Petl [18] library to implement the above mentioned operations.

Semantic Transformation includes operations to create RDF triples according to the target ontology based on the data output by the *Traditional Transformation* component. To perform this task, SETL provides two methods: *createTriples* and *trickleLoad*. The *createTriples* method transforms the data into triples and writes them to disk. Afterwards, the bulkload methods (discussed in Section 3.6) can be used to load the data into a triple store. The *trickleLoad* method converts the cleansed and transformed data into triples and subsequently loads them into a triple store holding the triples in main memory. The core mechanism of creating RDF triples creation is sketched in Algorithm 12.

Algorithm 12 describes the steps of converting a relational table (*table*) into a set of RDF triples (T). As additional input, the algorithm needs the name of an attribute in the table (*resourceKey*) that can be used to uniquely identify each row. In addition, its value will be used to create resource identifiers. At first, a dictionary D is created based on the input *table* (line 1). D consists of (key, value) pairs for each row r in *table*; the key corresponds to r's *resourceKey* value. The value in turn corresponds to a list of (attribute, value) pairs where each pair represents one of r's cells. The next step is to determine which concepts in the target ontology the information contained in the table correspond to (line 2). Based on this information, the algorithm runs through each such concept c. Some tables in relational databases do not directly correspond to a concept, instead they represent many-to-many relationships between instances of concepts (entity types). Hence, only for those tables that directly correspond to a concept, the algorithm creates a new resource for each entry in the dictionary and a triple (with `rdf:type` as predicate) that defines the new resource as an instance of the concept (lines 5–6).

Algorithm 12: createTriples.

```
Input: $table$, $resourceKey$
Output: $T$
begin
1    $D = makeDictionary(table)$ $concepts$ =
     getMatchingConcepts($table$,$conceptList$) foreach $c \in concepts$ do
4        foreach $r \in D$ do
5            if $!(propertyTable(table))$ then
6                $T.addtriple(getIRI(r),$ rdf:type, $getConceptIRI(c))$
7            foreach $(attribute, value) \in r$ do
8                if $value! = NULL$ and
                 $matchesDefinedProperty(attribute, c)$ then
9                    $T.addTriple(getIRI(r), getPropertyIRI(a, c),$
                       $createObject(value))$
10   return $T$
```

Furthermore, for each (*attribute*, *value*) pair for which the *attribute* is defined as a match of one of *c*'s properties and *value* is not null, a triple encoding this information is created (lines 7–9). Depending on the type of property (data type or object), the object of the created triple either corresponds to a literal or a resource.

3.5 External Linking

Algorithm 13 formalizes the steps required to link an internal resource to external resources. It takes *intResource*, *searchEngine*, and *k* as input parameters. *intResource* is a given internal resource that we want to find external links for, *searchEngine* is the link of a Semantic Web search engine (in our current implementation, we use Sindice [27]), and *k* is the number of top-k URLs that we want to retrieve.

At first, the algorithm creates a semantic bag ($sb_{internal}$) for the internal resource (line 1). Such a semantic bag consists of the objects of triples describing the internal resource. Then, the search engine is queried for the top-k matching external resources (line 2). For each external resource (*extResource*), the algorithm retrieves triples describing it (line 4), creates a semantic bag ($sb_{external}$) for them (line 5), and compares the bag to the one created for the internal resource (line 6). If the Jaccard simarity (see Equation D.1) between the two semantic bags exceeds a certain threshold δ, the internal and external resource are considered a match (lines 6–7).

Algorithm 13: LinkToExternalResources.

```
Input: $intResource, searchEngine, k$
Output: $T$
begin
1    $sb_{internal} = semanticBag(intResource)$
2    $E = search(intResource, searchEngine, k)$
3    foreach $extResource \in E$ do
4        $extTriples = retrieveTriples(extResource)$
5        $sb_2 = semanticBag(extTriples)$
6        if $match(sb_{internal}, sb_{external}) > \delta$ then
7            $alignedPairs \cup = (intResource, extResource)$
8    $alignedPairs = userInteraction(alignedPairs)$
9    foreach $pair \in alignedPairs$ do
10       $T.addTriple(intResource,$ owl:sameAs, $extResource)$
11       $ER = getEquivalentResource(extResource)$
12       foreach $equiResource \in ER$ do
13           $T.addTriple(intResource,$ owl:sameAs, $equiResource)$
14   return $T$
```

$$J(sb_{internal}, sb_{external}) = \left|\frac{sb_{internal} \cap sb_{external}}{sb_{internal} \cup sb_{external}}\right| \tag{D.1}$$

After having identified all candidate pairs, the user can optionally interact with the system and remove pairs. Then, for each pair of internal and external resources, a triple with the `owl:sameAs` property[1] is created to materialize the link in the dataset (line 10). Finally, the algorithm also retrieves the set of resources from the search engine that are already linked to the external resource and materializes the links to the internal resource as well (lines 11-13).

3.6 Loading

SETL currently uses Jena TDB [14] as a triple store and allows loading data batch-wise. SETL supports trickle load and bulk load. The trickle load mode transforms and feeds data as it arrives from the previous transformation step without staging on disk using SPARQL insert queries. In bulk load mode, the triples are written into a file on disk after transformation and then loaded batch-wise into the triple store. To support bulk load, SETL offers

[1]An `owl:sameAs` property indicates that two IRI references refer to the same real world object. Hence, subject and object IRI of a triple with `owl:sameAs` are considered linked.

two methods: *bulkTDBLoader (tdblocation, filePath, batchsize)* and *bulkSPARQLInsert(tdblocation, filePath, batchsize, database)*. The *BulkTDBLoader* is used to load data from a file on disk into a triple store using the TDBLoader. The parameter *tdblocation* denotes the location of a TDB database, the *filePath* is the path to the triple file, and *batchsize* is the number of triples to be fed into the store at a time. The *BulkSPARQLInsert* is similar to *BulkTDBLoader* but defines SPARQL insert statements to load triples instead of using the TDBLoader. It additionally requires the name of the database as parameter.

4 Evaluation

To evaluate SETL, we applied the proposed ETL process on sources covering Danish agricultural and business information. We chose this use case and these datasets for evaluation as there exists an RDF dataset based on these sources [2] that has been created in a manual process by using a broad range of different tools. In contrast, SETL allows to perform this process mostly automatically with minimal user interaction. This allows us to compare the different ETL processes as well as the RDF datasets themselves. In the following, we refer to the dataset published by [2] as ExtBIKB and the one created by SETL as SETLKB. Our evaluation concentrates on three aspects: the time required to run the processes, quality of the produced datasets, and productivity, i.e., to what extent SETL eases the work of a user when performing an ETL task to produce RDF datasets.

We run the experiment on a HP ProLiant DL385 server with an AMD Opteron(tm) processor 6376 with 32 cores (only 1 core is used in the experiments); it has 256 GB DDR3 RAM and is running Ubuntu 14.04.1 LTS (Trusty Tahr). The data is stored on an 1-TB SCSI disk running in an HP Smart Array.

4.1 Performance

In this section, we discuss the time required for different sub-processes to create SETLKB using SETL and the performance of different loading methods.

Runtime of the different phases. Figure D.4 illustrates the time that each phase of the ETL process requires; the figure shows both the time for each phase in separate and the overall runtime of the complete process. The times represent averages over 5 test runs. As SETL automatically connects all the phases, the end time of one phase becomes the starting time of the next. Figure D.4 also shows the materialized output of each phase.

The *Extraction* and *Traditional Transformation* phases take a relatively long time because they involve performing two spatial join operations for the agriculture datasets. The *Semantic Transformation* process is time-intensive be-

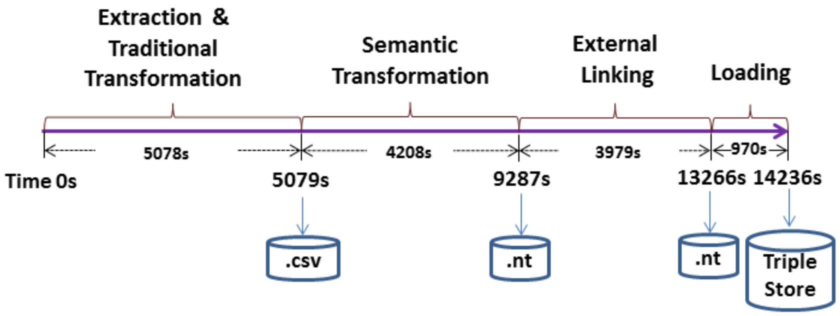


Fig. D.4: Required time for the ETL process.

cause all the data needs to be processed and matched to the schema of the target ontology. The total time required for the *Linking* process is 3,979 seconds. Table D.1 shows the time consumed for linking the instances of each concept. In summary, SETLKB is linked to 14,153 external resources. To load all triples at a time, the *BulkTDBLoader* takes 970 seconds. We discuss the performance of the different methods in the following section.

concept	**time(in seconds))**
Crop(C)	2,086
BusinessFormat(BF)	692
FieldBlockApplication(FBA)	95
Municipality(M)	1,106
total	**3,979**

Table D.1: Time required for the linking process.

Performance of different loading methods. As described in Section 3.6, SETL provides three different loading methods: *TrickleLoad*, *BulkTDBLoader*, and *BulkSPARQLInsert*. All these methods support batch-wise loading. In the following, we first compare the three methods based on their loading time. Then, we evaluate the performance of the trickle load in more detail and finally the bulk load.

Figure D.5 shows the loading times for the different loading methods for different batch sizes. To make the times comparable to trickle load, we add the time required to perform *Semantic Transformation* with the loading time of the bulk load's methods. The y-axis indicates the required time to load the set of triples in seconds, and the x-axis indicates the batch size, which is measured in number of triples. The results show that all methods take less time with increasing batch size. With a batch size of 6K, *BulkTDBLoader* takes 3.5 times more time than the other methods. However, as batch size increases from 6K to 1M, the runtime decreases significantly. With a batch size 1 million (1M) to 32 million (32M), the loading time of *BulkTDBLoader* becomes

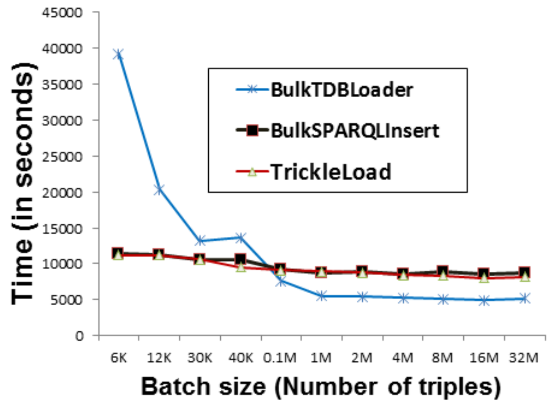


Fig. D.5: Total runtime for batch-wise loading.

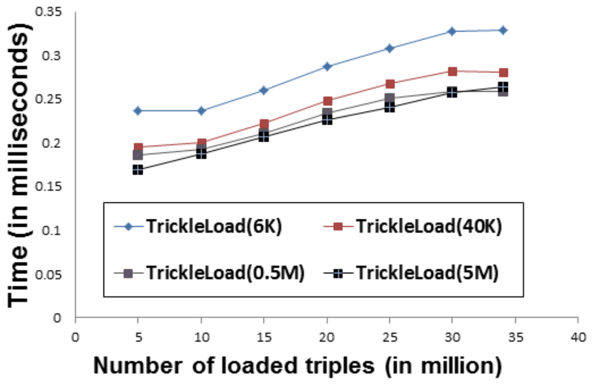


Fig. D.6: Loading time per triple for trickle load.

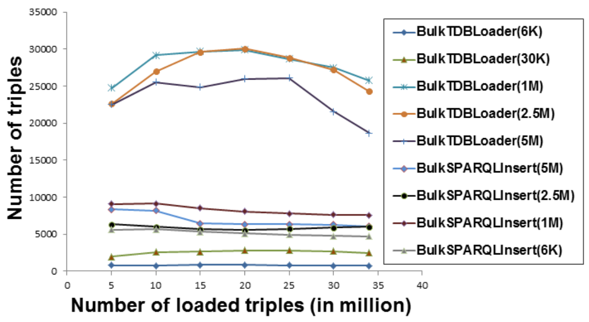


Fig. D.7: Load rate for bulk load.

nearly stable and *BulkTDBLoader* becomes two times faster than other two methods. This is because TDBLoader is especially implemented for loading big files; therefore, it performs better with larger batch sizes. On the other hand, *TrickleLoad* is slightly better than *BulkSPARQLInsert* in all cases. This is because *BulkSPARQLInsert* reads the triples from disk, where *TrickleLoad* uses memory as intermediate storage. In this experiment, we find that all methods take lowest time to load with a batch size of 16M. In summary, we recommend *BulkTDBLoader* for large files and *BulkSPARQLInsert* for smaller files. Trickle load is preferable when concurrent processing and loading is required.

Figure D.6 illustrates the time trickle load requires to load increasing numbers of triples. Here, the lines indicate different batch sizes. We calculate the time $t = T/N$, where T is the total time required to load N triples. The figure shows that with the increase in the number of loaded triples, the time increases almost linearly. For the trickle loads with the base sizes 6K and 40K, the constant startup cost is 10 million triples.

Figure D.7 shows the loading rate in terms of the number of triples loaded per second (TPS) for different batch sizes of the bulk load. The figure shows that with an increasing number of loaded triples, TPS increases until a certain point for *BulkTDBLoader*. After that point, TPS decreases with the increase in the number of loaded triples. This is mainly because of the internal indexing strategy of Jena TDB [14]. It also depends on the size of the triples to be loaded. On the other hand, for *BulkSPARQLInsert*, the number of triples per second gently decreases with the an increase in the number of loaded triples.

4.2 Quality

In this section, we examine the extent to which the produced triples are consistent, complete, interpretable, fresh, and semantically linked [26]. We compare the result created with SETL (SETLKB) to ExtBIKB.

Consistency. According to the definition, an object property relates instances of two classes, i.e, if x is a value of an object property, there must exist at least one triple whose subject is x. We found out that for a significant number of values for the object properties `bus:postalAddress` and `bus:officialAddress`, ExtBIKB does not contain such information. There are 83,302 `bus:officialAddresses` and 34 `bus:postalAddresses` that are not used as a subject of any triple in ExtBIKB. In SETLKB, on the other hand, all `bus:officialAddresses` and `bus:postalAddresses` have correctly been related to the `bus:Address` concept. Hence, data consistency is completely maintained in SETLKB.

Information loss. In ExtBIKB, fields that do not produce any crop are not represented. This affects 2,650 instances of the concept `agri:Field`. Similarly, instances of `agri:OrganicField` that have the value "999999-99" for the agri:FieldBlock property are not represented in ExtBIKB. This affects 1,213 instances. For SETLKB on the contrary, we do not want to miss any agri:Field or agri:OrganicFields instance even though they do not have crop information. Hence, in the cleansing step, we replace the value "999999-99" with *NULL*. If an instance of a concept contains *NULL* or an unknown value for a property, the *createTriples* method (described in Section 3.4) simply does not produce a triple for that property. This is how SETLKB ensures data completeness.

In the original raw datasets, we can find out since when a legal unit belongs to an owner. This is not possible in ExtBIKB if an owner owns more than one company. We fixed this issue by extending the target ontology. In addition, we also add the information related to the type of the owner, i.e., whether the owner is a person or another company, both are missing in ExtBIKB.

Furthermore, the main and secondary activity of a company or production unit are treated as the same activity in the ExtBIKB. In SETLKB, the concepts bus:Company and bus:ProductionUnit are connected to bus:Activity through two object properties: bus:hasMainAcitivity and bus:hasSecondaryActivity. Thus, SETLKB provides more information than ExtBIKB.

Redundant triples. If a property is inversely connected to another property, then storing the same types of triples for both properties, simply by interchanging the subject and predicate of the triples, makes a KB redundant. The properties bus:belongsTo and bus:hasProductionUnit are inversely connected to each other and relate the concepts bus:Company and bus:ProductionUnit. In ExtBIKB, triples are stored for both properties separately, which produces extra 659,639 triples. On the other hand, we create triples only for bus:belongsTo property. Hence, SETLKB reduces redundancy.

External linking. We enhance our knowledge base by linking it to other external resources at instance level. Section 3.5 describes the linking process. We run the linking process only for instances of agri:Crop (C), agri:FieldBlockApplication (FBA), bus:BusinessFormat (BF), and bus:Municipality (M) because for instances of other concepts, Sindice [27] does not contain relevant information. Note that the following results are produced by the automatic matching process without any pre-filtering by the user (which is mentioned as an optional step in Section 3.5). Table D.2 shows the different types of concepts for which we run the linking process, their total number of instances, the number of internal instances linked to external resources, the number of external resources connected to an internal instance, and the accuracy rate of the linking.

In total, 202 instances of SETLKB are linked to 14,153 external resources. To evaluate the linking process, we randomly chose 5% of the triples that linked internal and external resources and manually evaluated their correctness. The accuracy rate (AR) in Table D.2 shows the accuracy rate in percentage. One of the reasons having the low accuracy rate for some cases is the language. The description of each instance is translated from Danish to English using google translator API, which sometimes fails to give accurate translation.

Concept	# of inst.	Linked inst.	External resources	AR
C	244	97	7,602	68%
BF	30	19	1,770	39%
FBA	17	6	1,243	29%
M	98	80	3,538	57%
Total	**389**	**202**	**14,153**	**54.9%**

Table D.2: Linking local resources to external resources – AR stands for accuracy rate.

Based on these findings, we can conclude that SETLKB is more consistent, complete, and resourceful than ExtBIKB. Table D.3 summarizes some notable triple-wise differences between SETLKB and ExtBIKB.

Concept(c)/ Property(p)	# of triples SETLKB	# of triples ExtBIKB	Comments
Whole KB	34,177,652	32,457,657	SETLKB is the updated version.
External linking	14,153	0	No instance is linked with external sources in ExtBIKB.
Company(c)	603,667	603,668	One duplicate triple in ExtBIKB.
Address(c)	499,850	497,938	In ExtBIKB, Address concept were not mapped with the postal address of Danish Business dataset.
Field(c)	52,060	50,842	ExtBIKB used poor cleansing techniques.
OrganicField(c)	613,903	611,093	ExtBIKB used poor cleansing techniques.
belongsTo(p)	659,638	659,640	Two duplicate triples in ExtBIKB.
hasProductionUnit(p)	0	659,640	belongsTo and hasProductionUnit are inversely connected.
hasOwnerType(p)	353,954	0	ExtBIKB did not include owner type information.

Table D.3: Comparison between SETLKB and ExtBIKB

	SETL			ExtBI		
Task	**Used tools**	**Used language**	**LOC**	**Used tools**	**Used language**	**LOC**
Ontology Defining	Built-in SETL	Python	286	Virtuoso, Protege	Not Applicable	-
CSV	Build-in Petl	Python	2	MySQL,vtSQL	SQL	-
Reading Shapefile	Build-in SETL	Python	3	ArcGIS,PostgreSQL, shp2pgsql	Not Applicable	-
Spatial join	Built-in ArcPy	Python	2	ArcGIS	Not Applicable	-
Cleansing	Built-in Petl	Python	85	MySQL, Google refine	SQL	360
Triple Generation	Built-in SETL	Python	61	Virtuoso	iSQL	1511
Loading	build-in SETL	Python	4	Virtuoso	iSQL	4

Table D.4: Comparison between the ETL processs of SETL and [2]

4.3 Productivity

One of the main advantages of working with SETL is that all the phases of an ETL process can be easily maintained and implemented using Python. Table D.4 summarizes the tools, languages, and the lines of code (LOC) used to produce the SETLKB dataset using SETL and the ExtBIKB dataset using the process described in [2]. To process a shapefile, SETL provides methods to automatically create and insert data into PostgreSQL. It uses only 2 lines of Python code. To perform spatial intersection operation between two shapefiles, SETL uses the ArcPy [3] library, which is much more faster than ArcGIS [9] which is used in [2]. On the other hand, the ETL process used in [2] converts the shapefile into PostgreSQL table using the PostgreSQL tool *shp2pgsql*. To make it compatible with vtSQL, they performed some find-replace actions. For performing, spatial operation, they used ArcGIS [9]. For filtering inconsistent values, unknown characters and for correcting candidate keys, we use different methods from the Petl library [18], which are efficient and easy to access. It takes only 85 lines of code. On the other hand, the [2] used different MySQL views and hand-crafted methods to cleanse the data, and it took 360 LOC. To generate triples for all data sources, SETL takes only 85 LOC whereas [2] uses 1,511 LOC.

5 Related Work

This section discusses the state of the art in the area of semantic ETL to build a semantic DW and to integrate and publish data as Linked Open Data (LOD) from multiple sources.

Recently, the use of SW technology in data warehouses has become popular. Some approaches have used ontologies as a data integration method for mapping between sources and target whereas others have considered a data warehouse as a repository of ontologies and its instances.

In [22, 23], an OWL ontology is used to link the schemas of semi-structured and structured data to a target DW schema. At first, the schemas of the data sources and the DW are described by a common graph-based model named datastore graph. Then, an application ontology is constructed to describe the semantics of the data sources and the DW, and mapping between them can be done through that ontology. In this way, the heterogeneity issues among the schemas of sources and target have been resolved. In this work, it is assumed that source and target schemas are known beforehand. In [25], the authors have shown that an ontological approach can automate and minimize the maintenance cost of the evolution of a DW schema.

A framework for designing semantic DW has been proposed in [16]. Here, the usage of SW languages to integrate distributed DWs and to automate the ETL process of a DW has been discussed. The DW has been considered as a repository of ontologies and semantically annotated data sources. [5] proposes a methodology describing important steps required to create a semantic DW that enables to integrate data from semantic databases, which is composed of an ontology and its instances. A multidimensional (MD) framework has been proposed in [15] to analyze semantic data that are provided by the SW and that are annotated by application and domain ontologies. It allows to build and populate a DW from both the analyst requirements and the knowledge encoded in the ontologies. To enable warehouse-style analysis over RDF data, some approaches [6, 13] propose the use of a native RDF data warehouse. To analyze an RDF graph, [6] introduced a lens named analytical schema, which is a graph of classes and properties. Each node of the schema represents a set of facts, which can be analyzed by exploring the reachable nodes.

In the past couple of years, publishing data in a machine-readable format has become more popular and (Linked Open Data) LOD has emerged as a way to share such data across Web sources. [28] presents a process to publish governmental data as LOD and [2] discusses how to spatially integrate and publish a Danish Agricultural dataset and a Danish Business dataset as LOD. The approach uses views to cleanse the data and Virtuoso for creating and storing RDF data. To integrate data from heterogeneous sources and publish those data as LOD, a semantic ETL framework is presented in [4] at conceptual level. The approach uses the semantic technology to facilitate the integration process and discusses the use of different tools to accomplish the different steps of the ETL process.

As data become outdated due to updates in the sources, using different tools for extraction, validation, and integration becomes very tedious and time consuming. Hence, a framework is necessary that makes it possible to accomplish every step in a single platform. Although the state-of-the-art approaches provide different conceptual frameworks, there is no implemented programmable framework that facilitates the developers by providing required tools, classes, and methods to build ETL process with the goal of either creating a semantic DW or publishing the semantically integrated data as LOD. Hence, this paper discusses a semantic ETL framework, SETL, which provides various modules, classes, and methods to extract semantic-aware data, geo-spatial data, and other traditional data, to integrate source data semantically using a target ontology, and to store the data as semantic data. SETL facilitates developers to build a semantic DW using a single platform instead of using different tools and a manual process.

6 Conclusion

Building a DW with the organizational internal and external data published in different formats requires semantic integration. Here, we have proposed and developed a programmable framework, named semantic ETL (SETL) that facilitates users to build a semantic DW. SETL uses ontology as a underlying schema to integrate heterogeneous data sources. Besides traditional data format, SETL can process semantic-aware data. Eventually, it stores the data as RDF triples. It also allows to link the internal resources with external resources. In order to evaluate our framework, we (re-)produced a knowledge base by integrating a Danish Agricultural dataset and a Danish Business dataset using SETL; then we compared SETL's KB (SETLKB) and development process with that of ExtBI. In Section 4, we showed the performance of our trickle load and bulk load. Comparing to ExtBIKB, SETLKB KB is more consistent, informative and normalized. SETL also provides better productivity than the tools used by ExtBI.

We will augment the *Ontology Definition* module with QB4OLAP [10] vocabularies to enable OLAP-like analysis. Our future research includes automatic recommendation of an ontology for integration by analyzing the structures of the given data sources, automatic mapping between sources and target ontology, recommendation of relevant sources, and designing semantic ETL operations. We also have a plan to make the GUI version of SETL.

References

[1] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, and A. Simitsis, "Using Semantic Web Technologies for Exploratory OLAP: A Survey," *TKDE*, vol. 99, pp. 571–588, 2014.

[2] A. B. Andersen, N. Gür, K. Hose, K. A. Jakobsen, and T. B. Pedersen, "Publishing Danish Agricultural Government Data as Semantic Web Data," in *JIST*, 2015.

[3] "ArcPy-ArcGIS Python Library, http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//000v00000001000000."

[4] S. K. Bansal, "Towards a Semantic Extract-Transform -Load (ETL) Framework for Big Data Integration," in *Big Data*, 2014, pp. 522–529.

[5] L. Bellatreche, S. Khouri, and N. Berkani, "Semantic Data Warehouse Design: From ETL to Deployment à la Carte," in *DASFAA*, 2013, pp. 64–83.

[6] D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatiş, "RDF Analytics: Lenses over Semantic Graphs," in *WWW*, 2014, pp. 467–478.

[7] "Danish Central Company Registry (CVR) Data, http://cvr.dk/."

[8] "DBpedia, http://dbpedia.org/."

[9] "Esri, Arcgis. http://www.esri.com/software/arcgis."

[10] L. Etcheverry, A. Vaisman, and E. Zimányi, "Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP," in *DaWak*, 2014, pp. 45–56.

[11] "Ministry of Food, Agriculture and Fisheries of Denmark, http://en.fvm.dk/."

[12] A. Harth, K. Hose, and R. Schenkel, *Linked Data Management*. CRC Press, 2014.

[13] K. A. Jakobsen, A. B. Andersen, K. Hose, and T. B. Pedersen, "Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries," in *COLD*, 2015.

[14] "Apache Jena TDB, https://jena.apache.org/documentation/tdb/."

[15] V. Nebot and R. Berlanga, "Building Data Warehouses with Semantic Web Data," *DSS*, vol. 52, no. 4, pp. 853–868, 2012.

[16] V. Nebot, R. Berlanga, J. M. Pérez, M. J. Aramburu, and T. B. Pedersen, "Multidimensional Integrated Ontologies: a Framework for Designing Semantic Data Warehouses," *JDS*, vol. XIII, pp. 1–36, 2009.

[17] "OWL Web Ontology Language, www.w3.org/TR/owl-ref/."

[18] "Petl - Extract, Transform and Load (Tables of Data), https://petl.readthedocs.org/en/latest/."

[19] "Pyshp, https://code.google.com/p/pyshp/."

[20] "Rdflib, https://github.com/RDFLib/rdflib."

[21] "ESRI, Shapefile Technical Description, INC, 1998."

[22] D. Skoutas and A. Simitsis, "Designing ETL Processes using Semantic Web Technologies," in *DOLAP*, 2006, pp. 67–74.

[23] ——, "Ontology-based Conceptual Design of ETL Processes for both Structured and Semi-structured Data," *IJSWIS*, vol. 3, no. 4, pp. 1–24, 2007.

[24] "SPARQLWrapper, http://rdflib.github.io/sparqlwrapper/."

[25] M. Thenmozhi and K. Vivekanandan, "An Ontological Approach to Handle Multidimensional Schema Evolution for Data Warehouse," *IJDMS*, vol. 6, no. 4, pp. 33–52, 2014.

[26] V. Theodorou, A. Abelló, and W. Lehner, "Quality Measures for ETL Processes," in *DaWaK*, 2014, pp. 9–22.

[27] G. Tummarello, R. Delbru, and E. Oren, "Sindice. com: Weaving the open linked data," in *ISWC*, 2007.

[28] B. Villazón-Terrazas, L. M. Vilches-Blázquez, O. Corcho, and A. Gómez-Pérez, "A Methodological Guidelines for Publishing Government Linked Data," in *Linking government data*, 2011, pp. 27–49.

[29] "W3C. Resource Description Framework, http://www.w3.org/RDF/."

[30] "W3C. Resource Description Framework Schema, http://www.w3.org/TR/rdf-schema/."

[31] M. E. Zorrilla, J.-N. Mazón, Ó. Ferrández, I. Garrigós, F. Daniel, and J. Trujillo, *Business Intelligence Applications and the Web: Models, Systems and Technologies*. Business Science Reference, 2012.

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-657-1

AALBORG UNIVERSITY PRESS