Requirement-driven Design and Optimization of Data-Intensive Flows

Data have become number one assets of today's business world. Thus, its exploitation and analysis attracted the attention of people from different fields and having different technical backgrounds. Data-intensive flows are central processes in today's business intelligence (BI) systems, deploying different technologies to deliver data, from a multitude of data sources, in user-preferred and analysis-ready formats. However, designing and optimizing such data flows, have been known as a burdensome task, typically left to the manual efforts of a BI system designer. These tasks have become even more challenging for next generation BI systems, where data flows typically need to combine data from in-house transactional storages, and data coming from external sources, in a variety of formats (e.g., social media, governmental data, news feeds). These challenges largely indicate a need for boosting the automation of the design and optimization of data-intensive flows.

This PhD thesis aims at providing automatable means for managing the lifecycle of data-intensive flows. In particular, the thesis first proposes an approach (CoAl) for incremental design of data-intensive flows, by means of multi-flow consolidation. CoAl not only facilitates the maintenance of data flow designs in front of changing information needs, but also supports the multi-flow optimization of data-intensive flows, by maximizing their reuse. Next, in the data warehousing (DW) context, we propose a complementary method (ORE) for incremental design of the target DW schema, along with systematically tracing the evolution metadata, which can further facilitate the design of back-end data-intensive flows (i.e., ETL processes). The thesis then studies the problem of implementing data-intensive flows into deployable formats of different execution engines, and proposes the BabbleFlow system for translating logical data-intensive flows into executable formats, spanning single or multiple execution engines. Lastly, the thesis focuses on managing the execution of data-intensive flows on distributed data processing platforms, and to this end, proposes an algorithm (H-WorD) for supporting the scheduling of data-intensive flows by workload-driven redistribution of data in computing clusters.

The overall outcome of this thesis is an end-to-end platform for managing the lifecycle of data-intensive flows, called Quarry. The results of this thesis largely contribute to the field of data-intensive flows in today's BI systems, and advocate for further attention by both academia and industry to the problems of the design and optimization of data-intensive flows.

Petar Jovanovic

Requirement-driven Design and Optimization of Data-Intensive Flows

Requirement-driven Design and Optimization of Data-Intensive Flows

Petar Jovanovic

PhD Thesis 2016

Supervised by:

Alberto Abelló, Oscar Romero, and Toon Calders

Petar Jovanovic

Requirement-driven Design and Optimization of Data-Intensive Flows

Ph.D. Dissertation

Petar Jovanovic

Dissertation submitted July, 2016

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

To my family

Mojoj porodici

Thesis submitted:	July, 2016
UPC Main Ph.D. Supervisors:	Prof. Alberto Abelló Prof. Oscar Romero Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain
ULB Ph.D. Supervisor:	Prof. Toon Calders Université Libre de Bruxelles, Brussels, Belgium
PhD Series:	Barcelona School of Informatics, Universitat Politècnica de Catalunya, BarcelonaTech

Printed in Spain, 2016

Curriculum Vitae

Petar Jovanovic

Petar Jovanovic graduated with honors from high-school gymnasium Miloš Savković, in 2006, with specialization in math and natural sciences, in his hometown of Aranđelovac (Serbia).

In August 2010, he graduated Software Engineering, at the School of Electrical Engineering, University of Belgrade. In October 2010, he continued his education at the Barcelona School of Informatics (FIB), Universitat Politècnica de Catalunya (UPC), and graduated Master in Computing, with specialization in Information Systems, in July 2011.

In October 2011, he decided to pursue his PhD studies under the supervision of professors Alberto Abelló and Oscar Romero inside the Group of Information Modelling and Processing (MPI), at the department of Service and Information System Engineering (ESSI), at Universitat Politècnica de Catalunya. In 2013, he has been awarded a four year PhD grant by the Spanish Ministry of Education, Culture, and Sport, under the FPU grant program. During his master and PhD program, he has been working on the problems of requirement-driven design and optimization of data-intensive flows in Business Intelligence systems.

His research interests mainly fall into the business intelligence field, namely: Big Data management, Management and optimization of data-intensive flows, Data warehousing, ETL, Distributed data processing, MapReduce, Hadoop ecosystem.

During his PhD studies, with MPI research group he has been part of the research project funded by the Spanish Ministry of Science and Inovation (TIN2011-24747), working on the development of new techniques for information integration (2012-2014). Additionally, with his supervisors Alberto Abelló and Oscar Romero, he has participated in the collaborative research project between Universitat Politècnica de Catalunya and HP Labs, Palo Alto, Califor-

nia (USA). From the HP Labs' side, the project was led and supervised by the HP Labs' senior researcher Alkis Simitsis. This project focused on automating the logical design and optimization of hybrid data flows, enabling engine independent design and composition of data flows spanning different execution and storage engines, hence enabling inter-engine management and optimization of hybrid data flows. The project also included two research stays (July - September 2012, and March - July 2013) at the HP Labs facilities in Palo Alto, California, where Petar was also working on the development of HP Labs' proprietary Hybrid Flow Management System (HFMS).

In 2014, Petar joined the Erasmus Mundus Joint Doctorate program of Information Technologies for Business Intelligence, Doctoral College (IT4BI-DC), and continued his PhD studies in cohort with Université Libre de Bruxelles (ULB). Professor Toon Calders from ULB has joined the supervision of his PhD thesis as a host co-advisor.

As part of his joint PhD studies, Petar performed two research stays at Université Libre de Bruxelles, his host university, working with professor Toon Calders (September - December 2014, and August - November 2015). For both of these research stays, he was awarded with competitive research stay grants by the Spanish Ministry of Education, Culture, and Sport, under the FPU Estancias Breves grant program.

He has published 13 peer-reviewed publications, including 3 JCR indexed journal articles, 1 Springer LNCS journal article, 2 research track, full conference papers, 3 full workshop papers, and 4 tool demonstrations. During the collaborative research project with HP Labs, he also co-authored 2 patent applications with the researchers from HP Labs (Alkis Simitsis and Kevin Wilkinson).

He has been one of the initiators and developer inside the Quarry project at Universitat Politècnica de Catalunya, which aims to automate the complex and time-consuming task of the incremental data warehouse (DW) design from high-level information requirements. This project is one of the main contributions of his PhD work, and during the years, the project has gathered several researchers, PhD, master, and bachelor students.

Petar has also participated in teaching and advisory work at Universitat Politècnica de Catalunya. During spring semesters of 2013/2014, 2014/2015, and 2015/2016, he has taught lab sessions of the Data Warehousing course, in the Master in Innovation and Research in Informatics (MIRI), at Barcelona School of Informatics. In addition, in 2016, he has also taught lab sessions of the Big Data Management and Analytics (BDMA) postgraduate program, at UPC School of Professional and Executive Development. Besides that, he has also co-advised 7 master thesis and 1 final (bachelor) degree project.

Abstract

Data have become number one assets of today's business world. Thus, its exploitation and analysis attracted the attention of people from different fields and having different technical backgrounds. Data-intensive flows are central processes in today's business intelligence (BI) systems, deploying different technologies to deliver data, from a multitude of data sources, in user-preferred and analysis-ready formats. However, designing and optimizing such data flows, to satisfy both users' information needs and agreed quality standards, have been known as a burdensome task, typically left to the manual efforts of a BI system designer. These tasks have become even more challenging for next generation BI systems, where data flows typically need to combine data from in-house transactional storages, and data coming from external sources, in a variety of formats (e.g., social media, governmental data, news feeds). Moreover, for making an impact to business outcomes, data flows are expected to answer unanticipated analytical needs of a broader set of business users' and deliver valuable information in near real-time (i.e., at the right time). These challenges largely indicate a need for boosting the automation of the design and optimization of data-intensive flows.

This PhD thesis aims at providing automatable means for managing the lifecycle of data-intensive flows. The study primarily analyzes the remaining challenges to be solved in the field of data-intensive flows, by performing a survey of current literature, and envisioning an architecture for managing the lifecycle of data-intensive flows. Following the proposed architecture, we further focus on providing automatic techniques for covering different phases of the data-intensive flows' lifecycle. In particular, the thesis first proposes an approach (CoAl) for incremental design of data-intensive flows, by means of multi-flow consolidation. CoAl not only facilitates the maintenance of data flow designs in front of changing information needs, but also supports the multi-flow optimization of data-intensive flows, by maximizing their reuse. Next, in the data warehousing (DW) context, we propose a complementary method (ORE) for incremental design of the target DW schema, along with systematically tracing the evolution metadata, which can further facilitate the design of back-end data-intensive flows (i.e., ETL processes). The thesis then studies the problem

of implementing data-intensive flows into deployable formats of different execution engines, and proposes the BabbleFlow system for translating logical data-intensive flows into executable formats, spanning single or multiple execution engines. Lastly, the thesis focuses on managing the execution of data-intensive flows on distributed data processing platforms, and to this end, proposes an algorithm (H-WorD) for supporting the scheduling of data-intensive flows by workload-driven redistribution of data in computing clusters. The overall outcome of this thesis is building an end-to-end platform for managing the lifecycle of data-intensive flows, called Quarry. The techniques proposed in this thesis, plugged to the Quarry platform, largely facilitate the manual efforts, and assist users of different technical skills in their analytical tasks. Finally, the results of this thesis largely contribute to the field of data-intensive flows in today's BI systems, and advocate for further attention by both academia and industry to the problems of the design and optimization of data-intensive flows.

Acknowledgments

In the following few lines, I would like to thank to all the people with whom I shared the path towards the completion of this PhD thesis, and who helped me grow both personally and professionally.

Initially, I would like to thank my advisors from Universitat Politècnica de Catlunya, Dr. Alberto Abelló and Dr. Oscar Romero, for their constant guidance and support throughout my master and PhD studies. Their inspiring ideas and their knowledge and expertise helped me shape my research path and bring a real novelty into the field. But most importantly, their abundant patience and the collaborative atmosphere made me persist on the path and improve the quality of my work. I also want to thank my advisor from Université Libre de Bruxelles, Dr. Toon Calders, whose support and guidance in the second part of my PhD thesis have been really valuable to conclude this work. I especially want to thank him for hosting me and working together during my research stays at Université Libre de Bruxelles.

In addition to my advisors, during my PhD studies I had a fortune and a real honor to meet and collaborate with Dr. Alkis Simitsis, a senior researcher from HP Labs. Alkis' expertise in the field of ETL and data-intensive flows helped me advance and clarify my doubts about the topic. Alkis also hosted me twice during my research stays at HP Labs in Palo Alto. I would first like to thank him for giving me this great opportunity, and for all the valuable discussions, advices and encouragements that have kept me believe in my ideas and my work.

Moreover, I want to thank all my colleagues from the DTIM group and the IT4BI-DC program at Universitat Politècnica de Catlunya, and all my colleagues from Université Libre de Bruxelles. They have always been there to help, and really made me feel like part of the team. Thank you guys for all the collaborative work we have done together, and for all the inspiring discussions about research, life, politics.... during our lunch and coffee breaks.

A special thanks to all the bachelor and master students that I have co-advised during the last five years, for their great efforts, and the hard work they invested into building our Quarry platform.

I would like to thank to the anonymous reviewers as well as to researchers

from many conferences, who provided great ideas and priceless feedback in different stages of this work.

I would like to express my deepest gratitude to my family, to whom I devote this thesis. To my mom and dad, for their unconditional love, support, and foremost sacrifice during all these years. They have always been a rock to hold on and inspiring examples of hard work, honesty, and lifelong values. I thank them for believing in me, and for encouraging me to follow my path and grow into an independent person I am today. I am especially grateful to my big sister, Nevena, who has always been my tailwind and my reality check. I thank her for her love, support, advices, talks, and for making me a stronger and better person.

Last but not least, I would also like to thank my friends. To my friend Mila, who, although remotely, has been a real support and a second sister during all these tough years. I would also like to thank my colleague and a great friend Jovan, for always being there when I needed help and for all the valuable talks and memories we have shared. I would especially like to thank my friend, flatmate, and really more like a Spanish brother, Silverio, with who I worked, discussed, argued, laughed, joked, lived and shared many unforgettable lifetime moments in the past five years.

Thanks to all the people that contributed to my personal and professional life!

This work has been possible thanks to the Spanish FPU grant FPU12/04915, and FPU Grants for Research Stays (Estancias Breves) EST13/00944 and EST14/00548. Parts of this work have been partially supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2011-24747, and by the Secreteria d'Universitats i Recerca de la Generalitat de Catalunya under 2014 SGR 1534.

Contents

Thesis Details

Thesis Title:	Requirement-driven Design and Optimization of Data-Intensive Flows
Ph.D. Student:	Petar Jovanovic
Supervisors:	Prof. Alberto Abelló, Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain (UPC Main Supervisor) Prof. Oscar Romero, Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain (UPC Co-Supervisor) Prof. Toon Calders, Université Libre de Bruxelles, Brussels, Belgium (ULB Supervisor)

The main body of this thesis consist of the following papers.

[1] Petar Jovanovic, Oscar Romero, Alberto Abelló. A Unified View of Data-Intensive Flows in Business Intelligence Systems: A Survey. Trans. Large-Scale Data- and Knowledge-Centered Systems, InPress (2016)

[2] Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló. Incremental Consolidation of Data-Intensive Multi-Flows. IEEE Trans. Knowl. Data Eng. 28(5): 1203-1216 (2016)

[3] Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló, Daria Mayorova. A requirement-driven approach to the design and evolution of data warehouses. Inf. Syst. 44: 94-119 (2014)

[4] Petar Jovanovic, Alkis Simitsis, Kevin Wilkinson. Engine independence for logical analytic flows. ICDE 2014: 1060-1071

[5] Petar Jovanovic, Oscar Romero, Toon Calders, Alberto Abelló. H-WorD: Supporting Job Scheduling in Hadoop with Workload-driven Data Redistribution, accepted for publication at 20th East-European Conference on Advances in Databases and Information Systems, August 2016.

In addition to the main papers, the following peer-reviewed publications have also been made.

- Journal articles:

  [1] Vasileios Theodorou, Petar Jovanovic, Alberto Abelló, Emona Nakuçi. Data generator for evaluating ETL process quality. Inf. Syst. InPress (2016)

- Conference and workshop full papers:

  [2] Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló. Integrating ETL Processes from Information Requirements. DaWaK 2012: 65-80

  [3] Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló. ORE: an iterative approach to the design and evolution of multi-dimensional schemas. DOLAP 2012: 1-8

  [4] Emona Nakuçi, Vasileios Theodorou, Petar Jovanovic, Alberto Abelló. Bijoux: Data Generator for Evaluating ETL Process Quality. DOLAP 2014: 23-32

  [5] Rizkallah Touma, Oscar Romero, Petar Jovanovic: Supporting Data Integration Tasks with Semi-Automatic Ontology Construction. DOLAP 2015: 89-98

- Tool demonstrations:

  [6] Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló. Requirement-Driven Creation and Deployment of Multidimensional and ETL Designs. ER Workshops 2012: 391-395

  [7] Alkis Simitsis, Kevin Wilkinson, Petar Jovanovic. xPAD: a platform for analytic data flows. SIGMOD Conference 2013: 1109-1112

  [8] Petar Jovanovic, Alkis Simitsis, Kevin Wilkinson. BabbleFlow: a translator for analytic data flow programs. SIGMOD Conference 2014: 713-716

  [9] Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló, Héctor Candón, Sergi Nadal. Quarry: Digging Up the Gems of Your Data Treasury. EDBT 2015: 549-552

This thesis has been submitted for assessment in partial fulfillment 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 extended summary of the thesis.

Chapter 1

Introduction

1 Background and Motivation

We have certainly entered the era in which data volumes are uncontrollably generated with immense speeds, by humans, as well as by machines. To make the situation clearer, until 2014 we had that "the amount of data collected in the last two years is higher than the amount of data collected since the dawn of time" [11]. At the same time, data and even more the knowledge extracted from them, have become number one assets in businesses world-wide, namely "the gems of the enterprise" [146]

The companies thus strive to obtain information from these data, and acquire useful knowledge for making their business and strategic decisions. Traditional business intelligence (BI) systems were introduced to support such strategic needs of business users [119]. Intuitively, a BI system typically "refers to a set of tools and techniques that enable a company to transform its business data into timely and accurate information for the decisional process, to be made available to the right persons in the most suitable form" [139].

For enabling smoother data analysis, a variety of data transformations need to be applied over raw data. Besides well-studied relational algebra operations (e.g., selection, join, Cartesian product) [171], other, more complex, and typically ad hoc built operations are additionally required to conform the raw data to the formats suitable to end users' analytical needs (e.g., data cleaning, pivot, data mappers, or in general operators with black-box semantics [186]). As pointed out in [186], the situation becomes more challenging, as these data transformations typically need to be combined in a workflow.

A prominent solution studied in the past twenty years for enabling data analysis over historical data is data warehousing (DW). The first and the most complete definition of a DW is given by Bill Inmon in 1992 [82]: "A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection

of data in support of management's decision making process". Intuitively, a DW system assumes a unified data storage, typically modeled following the multidimensional (MD) paradigm [140], to support analytical needs of business users. In the back stage, a DW system includes a batched data workflow known as extract-transform-load (ETL) process, responsible for orchestrating the flow of data from relevant data sources towards a data warehouse. The complexity and high costs of building an ETL process have been largely discussed both by practitioners and researchers in the peak years of data warehousing [169, 185], reporting that the correct ETL process implementation can take up to 80% of the entire DW project [170].

The situation, however, has become unbearable for traditionally built BI systems [119], given the inherent complexity that the new data characteristics (a.k.a., BigData) brought into picture. As initially defined [81], the three main challenges, also known as the 3 V's of BigData are: large volumes of data being collected (i.e., volume), growing speed in which data are arriving into systems (i.e., velocity), and plethora of different formats in which these data are supplied (i.e., variety).

At the same time, the high dynamics of global markets demand moving from typical long-term and strategic only, towards medium and short term (i.e., tactical and operational) decision making, such that it "provides the right information to the right people at the right time so they can make a positive impact on business outcomes" [50]. In addition, businesses have become more situation-dependent, thus combining in-house data with external data sources (e.g., govermental data, social networks, open data initiatives) has become essential for providing useful, context-aware knowledge to decision makers [110]. Lastly, there is also a demand for more agility in the BI systems development in order to more easily accommodate new, unanticipated, or changed needs of a broader set of business users [34, 79].

These challenges inevitably required changes in the roadmap for previously long-lasting, BI and DW projects [119], which typically, as other software projects, required long requirement elicitation cycles, followed by several, mostly manual rounds of reconciliation and redesigning, before all the business needs were met [146].

Being the most complex part of BI projects, but at the same time the most critical processes in today's BI applications, problems related to data workflows, like ETL processes, still occupy the attention of both researchers and practitioners. For meeting complex requirements of next generation BI systems [87], we often need an effective combination of the traditionally batched ETL processes that populate a DW from integrated data sources, and more real-time and operational data flows that integrate source data and provide results to users at runtime. Thus, throughput this document, we use the term data-intensive flow (DIF), or simply, data flow, for capturing in a more general manner the variety of data workflows in the next generation BI systems.

2 The Lifecycle of Data-intensive Flows

Data-intensive flows are critical processes in BI applications with the common goal of delivering the data in user-preferred and analysis-ready formats, from a multitude of data sources. In general, a DIF starts by extracting data from individual data sources, cleans and conforms extracted data to satisfy certain quality standards and business requirements, and finally brings the data to end users.

Similar to other software products, the lifecycle of a DIF in BI applications typically consists of several phases. We further define the lifecycle of DIFs (see Figure 1.1), and give an overview of its phases, along with example approaches that either fully or partially facilitate them.

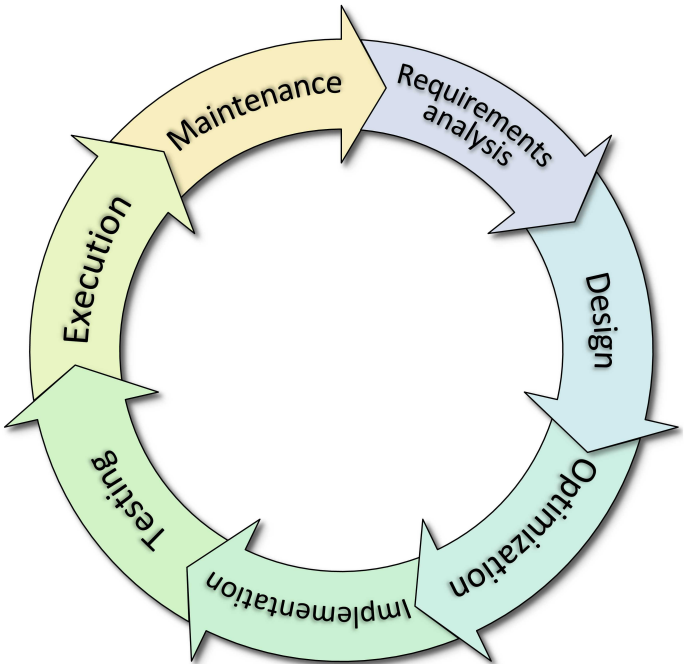


Fig. 1.1: The lifecycle of DIFs in BI applications

- Requirements analysis. The lifecycle of DIFs starts by eliciting and analyzing user requirements. In the context of data-centric (analytical) applications, user requirements are typically related to the analytical needs of end users, hence they express the information or knowledge a user wants to acquire from input data. We further refer to them as information requirements.

- Design. The next phase is dedicated to the conceptual and logical design of DIFs that will answer the information requirements of end users. Having a plethora of different commercial and open-source design tools in the market, the research has been rather focused on providing a unified way for the conceptual or logical design of data flows.

- Optimization. Being time-critical processes in BI application, a special attention is dedicated to the optimization of DIFs. Thus, once a DIF is designed to answer user information requirements, the following phase deals with optimizing the data flow considering other, non-functional, requirements, e.g., performance, reliability, fault-tolerance, to mention a few [176]. Consequently, a considerable amount of research has been dedicated to the problems of non-functional requirements of DIFs, especially to the problems of optimizing their performance.

- Implementation. Furthermore, once a DIF is designed to satisfy both information and non-functional requirements, in the following phase a designer needs to select the execution engine(s) for executing DIFs. Consequently, conceptual or logical DIFs need to be translated to the format of the engine of choice (e.g., SQL, PIG scripts, Java/Scala/Python code).

- Testing. Given the inherent complexity of DIFs in BI applications, verifying and testing DIFs for satisfying both information and non-functional requirements represents an important part of the DIF lifecycle. Testbed platforms with benchmarking and syntactic data generation functionalities are needed to support the DIF evaluation tasks.

- Execution. Once ready, the execution of DIFs needs to be properly scheduled and monitored at the selected execution engine(s). Today, execution environments are typically distributed (e.g., organized in a form of a computing cluster), and shared among various applications coming from one or many users. Thus, proper scheduling and runtime optimization are essential for achieving better performance of DIFs and better resource utilization of a computing cluster.

- Maintenance. Lastly, once in place, DIFs are subject to changes, either due to the changed end users' information requirements, or to changes of the considered data sources. Thus, maintenance techniques, capable of efficiently accommodating such changes to the existing DIFs are needed.

2.1 Research Problems and Challenges

The businesses today are undoubtedly in a need for the means to support and automate the complex DIF lifecycle. To this end, this PhD thesis focuses

on studying the current field of DIFs in BI applications, and providing the techniques and tools for facilitating different phases of the DIF lifecycle.

Current research approaches have dealt with some of the problems in the DIF lifecycle. To this end, we start by surveying the current state of the art in data-intensive flows. We present the results of this survey in detail, later in Chapter 2, while here we give a brief overview of the main remaining challenges for the lifecycle of DIFs.

Requirements analysis. Analyzing user requirements and systematically incorporating them into a design of DIFs has been typically overlooked in real BI projects, and left to more ad hoc techniques, leading many of them to failure. Some attempts towards systematic requirements analysis have been undertaken (e.g., [65, 196]), but they still require long processes of collecting the requirements at different levels of abstraction and their analysis before manually incorporating them into a DIF design. Other works identified this problem and proposed some automation to such time-lasting process (e.g., [146]). However, the automation required lowering the level of abstraction for defining information requirements, tightening them to the multidimensional model notation (i.e., facts and dimensions of analysis). Recently, an extensive study has been performed [60] to identify how decision support system approaches fit the traditional requirements engineering framework (i.e., [132]). As a result, the authors identified a need for a systematic and structured requirements analysis process for further raising the level of automation for the design of decision support systems (including the design of DIFs), while at the same time keeping the requirements analysis aware of all stakeholder needs.

Design. In reality, DIFs are currently mostly built in an engine-specific manner, in a variety of tools, each with its own specific proprietary format [128]. These tools typically do not provide any automation for the design and optimization of DIFs. Thus, we identify a need for providing the re-engineering support for the DIF's design, with the aim of raising the abstraction of physical DIFs to more logical and conceptual levels. This will further facilitate engine-independent analysis and optimization of DIFs, as well as their redeployment to the same or different execution engine(s). Current research appraoches for the DW design were mainly focused either on providing a unified notation for modeling DIFs (e.g., [12, 180, 190]), or on supporting their design, e.g., by providing design guidelines [96], frameworks for assising designers [189, 13], or by automating the design of DIFs [44, 146]. However, these approaches still lack the automation support for identifying more complex data transformations, typical for analytic needs of today's business users, like data cleaning, machine learning algorithms, etc.

Optimization. When it comes to the optimization of DIFs, most of the current approaches (e.g., [158, 26, 78, 101]) have focused on optimizing the performance of data flow executions, typically following the principles of traditional relational query optimization [84]. Others have also considered other

non-functional requirements, e.g., reliability, fault-tolerance, recoverability, to mention a few [161]. In addition, the traditional problem of multi-query optimization [150] has also been identified as advantageous for optimizing today's DIFs, especially for the case of batched data flows, for which the joint execution by means of operation and data reuse is found to be especially beneficial [24, 94]. However, current approaches have dealt with the problem focusing on a limited set of relational algebra operations, thus not taking into account the variability and complexity of data transformations typically required in today's DIFs. Moreover, the current approaches for multi-flow optimization have not studied the benefits of equivalence flow transformations (i.e., operation reordering) for boosting the reuse of data and operations among DIFs.

Implementation. Approaches here have mostly dealt with the problem of deciding the execution engine(s) and implementation for DIFs [102, 181]. Some approaches also focused on automating the translation of a conceptual design of DIFs to the logical and physical level, thus generating an executable code for their execution (e.g., by means of model transformations from UML [120], or from BPMN [13, 195]). However, these approaches were mostly tied to a specific model transformation and thus to a single predefined executable format for DIF implementation. However, in today's BI applications, complex DIFs can span multiple execution engines, and thus require flexibility for the deployment over different execution engines [162].

Testing. Even though testing is an important phase of the DIF lifcycle, it is one of the least supported phases in the literature. The approaches that automate the design of DIFs, do incorporate the verification and validation checks for assuring the correctness of DIFs and their compliance to information requirements. However, testing DIFs for other (non-functional) requirements has been mainly overlooked and left to the manual efforts of a designer. To this end, some approaches further focused on supporting benchmarking of DIFs (e.g., [157, 159, 25]), while others also tackled the problem of automatic data generation to support evaluating the quality of DIFs (e.g., [178]). However, there is still a lack of a unified (end-to-end) solution (i.e., a testbed) for evaluating DIFs for both information and non-functional requirements.

Execution. In the context of scheduling the execution of DIFs, one of the biggest challenges, especially in the case of distributed data processing systems, is boosting the proximity of input data to the data processing tasks, for reducing network traffic costs (a.k.a., data locality). Currently used data processing engines mostly rely on simple techniques for scheduling DIFs (e.g., Capacity [3] and Fair [4] scheduling in Hadoop ecosystem [191]), which focus on exploiting data locality and thus bringing data processing tasks to the resources where their input data is stored (i.e., query shipping). However, with "blindly" favoring query shipping (over data shipping), skewed distribution of data in the cluster can severely affect the performance of DIFs. Therefore, many research attempts have dealt with this problem, typically making a bal-

ance between query shipping and data shipping, e.g., [194, 72]. Data shipping, on the other side, can also affect the execution of data processing tasks in a DIF, as they are additionally deferred waiting for their input data to arrive. Therefore, distributed data processing systems require more advanced scheduling techniques that would (based on the estimated or predicted workload of executing DIFs), schedule data processing tasks to optimize the performance of DIFs, and as well, schedule the data shipping actions in advance to avoid additionally deferring tasks' executions.

Maintenance. While many appraoches have dealt with either manual or automated design of DIFs, they typically do not provide any support for the redesign of DIFs in front of different evolution changes. Some of the current approaches have focused on dealing with the changes of involved data sources (e.g., [13, 129]). However, there is a lack of support for the evolution of DIFs in front of new, changed, or removed information requirements. We thus identify a need for supporting incremental design of DIFs and the means to efficiently accommodate the changed information needs of end users in the existing DIF design. Moreover, when it comes to the DW context, no existing evolution approaches up to now have dealt with the problem of target schema and ETL process evolution, at the same time, hence they have typically overlooked how the changes at the DW schema affect the design of the back-end DIFs, i.e., ETL process.

Starting from the identified challenges in managing the lifecycle of DIFs, this PhD thesis further aims at addressing some of these challenges and providing methods and tools for facilitating different phases of the DIF lifecycle, namely, design, optimization, implementation, execution, and maintenance.

3 Structure of the Thesis

The results of this PhD thesis are reported inside the five main chapters of this document (i.e., Chapter 2 - Chapter 6). Each chapter is self-contained, corresponding to an individual research paper, and hence it can be read in isolation. There can be some overlaps of concepts, examples, and texts in the introduction and preliminaries sections of different chapters as they are formulated in relatively similar kind of settings. Due to different timelines in which these works have been done, as well as the research teams in which we have worked, there may also be some discrepancies in the terminology for some of the concepts. However, the terminology and notation used in each of the works have been clearly defined and formalized in each particular chapter. One important difference is the term used for denoting a concept of DIFs. In some works they are actually called data-intensive flows (chapters 2 and 3), in some analytic flows (Chapter 5), while in others they are termed more specifically to the scenario in which they are deployed (e.g., ETL processes

in Chapter 4 and Appendix A, and MapReduce jobs in Chapter 6). All of them refer to the general concept of data-intensive flows, defined earlier in this chapter. In addition, Appendix A refers to a published tool demonstration of the Quarry platform, which deploys the modules resulted from this PhD thesis' work.

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

1. Petar Jovanovic, Oscar Romero, Alberto Abelló. A Unified View of Data-Intensive Flows in Business Intelligence Systems: A Survey, accepted with revision for publication in Trans. Large-Scale Data- and Knowledge-Centered Systems, (2016)

2. Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló. Incremental Consolidation of Data-Intensive Multi-Flows. IEEE Trans. Knowl. Data Eng. 28(5): 1203-1216 (2016)

3. Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló, Daria Mayorova. A requirement-driven approach to the design and evolution of data warehouses. Inf. Syst. 44: 94-119 (2014)

4. Petar Jovanovic, Alkis Simitsis, Kevin Wilkinson. Engine independence for logical analytic flows. ICDE 2014: 1060-1071

5. Petar Jovanovic, Oscar Romero, Toon Calders, Alberto Abelló. H-WorD: Supporting Job Scheduling in Hadoop with Workload-driven Data Redistribution, accepted for publication at 20th East-European Conference on Advances in Databases and Information Systems, August 2016.

6. Petar Jovanovic, Oscar Romero, Alkis Simitsis, Alberto Abelló, Héctor Candón, Sergi Nadal. Quarry: Digging Up the Gems of Your Data Treasury. EDBT 2015: 549-552

4 Thesis Overview

In this section, we provide a brief overview of the results of this PhD thesis. In particular, we first give a holistic overview of the thesis' contributions to the management of the DIF lifecycle (i.e., the Quarry project), and then we provide an overview of the contributions of each chapter of this thesis.

Quarry is the backbone project of this PhD thesis. Its main goal is to provide a system for facilitating and automating different phases of the lifecycle of analitical infrastructures, like DW systems. In particular, Quarry is envisioned

as an end-to-end platform for assisting users of various technical skills in managing the incremental design and deployment of DIFs, by automating the physical design of a DIF from high-level information requirements. Quarry provides means for efficiently accommodating the design of a MD schema and DIFs to new or changed information needs of its end-users. Finally, Quarry facilitates the deployment of the generated designs over an extensible list of execution engines. Deployed designs are then available for further fine tuning, scheduling, and execution. The overview of Quarry's functionalities are illustrated in Figure 1.2.

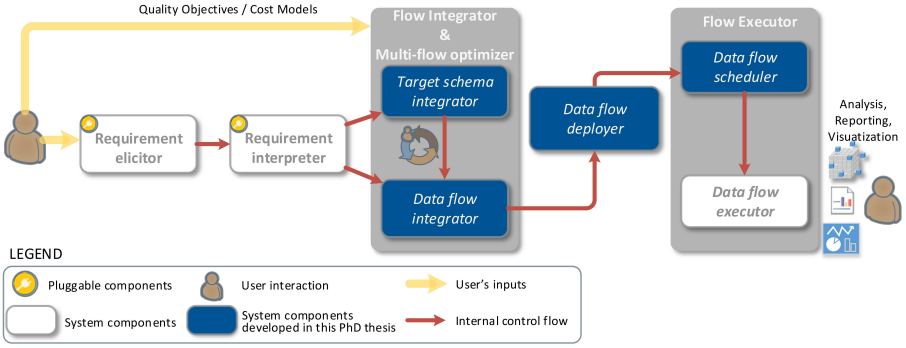


Fig. 1.2: The Quarry project

Quarry starts either from high-level information requirements expressed in terms of analytical needs of end users (e.g., [65, 146]), or the engine (implementation) specific formats (e.g., SQL query, scripts; [93]). Information requirements collected from end users (requirement elicitor) are further translated to the logical designs of DIFs that answer such requirements (requirement interpreter). In addition, in the case of a DW system, a target MD schema design is generated to support the storage of the data processed by the given DIF (i.e., ETL process). Independently of the way end users translate their information requirements into the corresponding logical designs (e.g., automatically [146, 18], or manually [13, 189]), Quarry provides automated means for integrating these designs into a unified design satisfying all requirements met so far (flow integrator & multi-flow optimizer).

On the one hand, data flow integrator incrementally consolidates DIFs satisfying individual information requirements, and facilitates the maintenance of DIFs in front of new, changed, or removed information requirements. While building a unified data-intensive multi-flow satisfying all the requirements, data flow integrator accounts for the cost of produced data flows and optimizes DIFs for joint (multi-flow) execution. For instance, by maximizing the reuse of data and operations and hence, maximizing the throughout of DIFs' execution.

On the other hand, target schema integrator provides means for facilitating the incremental design of the target MD schema, hence accommodating the new or changed information requirements of end users to the existing design. Importantly, while incrementally building the target schema, target schema integrator keeps traces of the transformations needed to adapt the existing schema to the changed information requirements, which can then be used for conforming the back stage DIFs (i.e., ETL processes).

Quarry further facilitates the implementation and deployment of designed DIFs, by means of translating them to the physical formats executable on different execution engines (flow deployer). Flow deployer also enables the multi-engine execution of DIFs, by providing the capablities of decomposing a DIF, and hence supporting approaches for execution engine selection (e.g., [102, 162]). Finally, Quarry assists the execution of deployed DIFs (flow executor), through optimally scheduling them over the available resources (data flow scheduler). Furthermore, data flow executor is in charge of submitting the execution of scheduled DIFs to the selected execution engine(s) and monitoring their progress. As a result of their execution, DIFs supply end users with valuable data ready for further analysis and exploitation, e.g., online analytical processing (OLAP), reporting, data mining, visualization.

More details about the complete Quarry system [90] are provided in Appendix A of this document. In the following subsections we give an overview of the Quarry components that resulted from this PhD thesis, which are further explained in detail in the corresponding chapters of this document.

4.1 Chapter 2: A Unified View of Data-Intensive Flows in Business Intelligence Systems: A Survey (The State of the Art)

Being a rather complex and broad field of research, problems related to DIFs have been typically addressed individually by different research approaches. Moreover, depending on the application requirements of DIFs, they are mostly separately studied for the case of back-end, batched ETL processes in DW systems, and more real-time and operational data flows that integrate source data at runtime. However, the next genaration BI systems require effective combination of the traditional, batched decision making processes, with on-the-fly DIFs to satisfy analytic needs of today's business users (e.g., [9, 42, 73]). For example, a part of the in-house company's sales data, periodically loaded to a DW by an ETL process, can at runtime be joined with up-to-date Web data to make more context-aware business decisions. To build such demanding systems, one must first have a clear understanding of the foundation of different data-intensive flow scenarios and the challenges they bring.

To this end, in Chapter 2, we present a comprehensive survey of today's research on data-intensive flows and the related fundamental fields of the database theory. The study is based on a set of dimensions describing the important chal-

lenges of data-intensive flows in the next generation BI setting. On the one hand, the results of this survey provide us with a clear understanding of the foundations of DIFs, while on the other hand, identified characteristics help us define the main remaining challenges in the field of DIFs. Finally, as the main outcome of this survey, we outlined the envisioned architecture of next generation BI systems, focusing on managing the lifecycle of data-intensive flows, which represents the framework of this PhD thesis study.

4.2 Chapter 3: Incremental Consolidation of Data-Intensive Multi-flows (Data Flow Integrator)

A data integration process combines data residing in various data sources and provides a unified view of this data to a user [106]. For instance, in a data warehousing (DW) context, data integration is implemented through back-end, batched ETL processes. Generally, data integration processes are typically represented in a form of a DIF that extracts, cleans, and transforms data from multiple, often heterogeneous data sources, and finally delivers data to users for further analysis or visualization. As we discuss in Chapter 2, there are various remaining challenges related to the design and optimization of DIFs. In Chapter 3, we focus on the challenges of incremental design and maintenance of DIFs.

A major problem that BI decision-makers face relates to the evolution of business requirements. These changes are more frequent at the early stages of a BI project [34] and in part, this is due to a growing use of agile methodologies in data flow design and BI systems in general [79]. But changes may happen during the entire BI lifecycle. Having an up-and-running DW system satisfying an initial set of requirements is still subject to various changes as the business evolves. The DIFs, as other software artifacts, do not lend themselves nicely to evolution events and in general, due to their complexity, maintaining them manually is hard. The situation is even more critical in today's BI settings, where on-the-fly decision making requires faster and more efficient adaptation to changes. Changes in business needs may result in new, changed, or removed information requirements. Thus having an incremental and agile solution that can automatically absorb occurred changes and produce a flow satisfying the complete set of requirements would largely facilitate the design and maintenance of DIFs.

At the same time, in an enterprise environment, data is usually shared among users with varying technical skills and needs, involved in different parts of a business process. Typical real-world data-intensive workloads have high temporal locality, having 80% of data reused in a range from minutes to hours [36]. However, the cost of accessing these data, especially in distributed scenarios, is often high [24], while intertwined business processes often imply overlapping of data processing. For instance, a sales department may analyze the

revenue of the sales for the past year, while finance may be interested in the overall net profit. Computing the net profit can largely benefit from the total revenue already computed for the sales department and thus, it could benefit from the sales data flow too. Such data flow reuse could result in a significant reduction in design complexity, but also in intermediate flow executions and thus, in total execution time too [24].

In Chapter 3, we present Consolidation Algorithm (CoAl), our approach to efficient, incremental consolidation of DIFs. CoAl deals with design evolution by providing designers with an agile solution for the design of DIFs. CoAl can assist the early stages of the design process when for only a few requirements we need to build a running data flow from scratch. But, it also helps during the entire lifecycle of a DIF, when the existing data flow must be efficiently accommodated to satisfy new, changed, or removed information requirements. CoAl reduces design complexity with aggressive information and software reuse. Per requirement, it searches for the largest data and operation overlap in the existing data flow design. To boost the reuse of existing design elements when trying to satisfy a new information requirement (i.e., when integrating a new data flow), CoAl aligns the order of data flow operations by applying generic equivalence rules. Note that in the context of data integration, the reuse of both data and code besides reducing flow complexity, might also lead to faster execution, better resource usage, and higher data quality and consistency [24, 94, 148].

CoAl also accounts for the cost of produced data flows when searching for opportunities to integrate new data flows. CoAl uses a tunable cost model to perform multi-flow, logical optimization, while creating a unified flow design that satisfies all information requirements. Here, we focus on maximizing the reuse of data and operations, but the algorithm can be configured to work with different cost models, taking into account different quality factors of data flows (e.g., [161]).

4.3 Chapter 4: A Requirement-Driven Approach to the Design and Evolution of Data Warehouses (Target Schema Integrator)

As mentioned earlier, DW systems have been widely recognized to successfully support strategic decision making. The most common design approach suggests building a centralized decision support repository (like a DW) that gathers the organization's data and which, to better support its analytical tasks, follows a multidimensional (MD) design [140]. The MD design is distinguished by the fact/dimension dichotomy, where facts represent the subjects of analysis and dimensions show different perspectives from which the subjects can be analyzed. In the context of DW systems, the batched, periodically executed DIFs (i.e., ETL processes) typically populate a unified DW storage. Thus,

the design and evolution of the target schema represents an additional design challenge, having that a design of DIFs is largely dependent on the constraints implied by the target DW schema (see Chapter 2).

While the design of DW systems has been typically considered as static, the dynamic nature of the businesses today requires a more agile approach for building the components of a DW system. Typically, assuming that all information and business requirements are available from the beginning and remain intact is not realistic. Indeed, the complexity of the monolithic approach for building a DW satisfying all information requirements, has also been largely characterized in the literature as a stumbling stone in DW projects, and different techniques, like the Data Warehouse Bus Architecture [97], were proposed as a step-by-step approach for building a DW. However, such design guidelines still assume a tremendous manual effort from a DW architect and hence, DW experts still encounter the burdensome and time-lasting problem of translating end user's information requirements into an appropriate DW design.

Therefore, automating the design process would largely support DW designers during the complex and time-lasting DW projects. Moreover, automation is also recognized as a way to assure the inherent contraints of a DW design (i.e., MD integrity constraints of target schemas [114]). Various approaches thus recognized the need for automation, and proposed the solution for generating the MD schema design either from data source (i.e., supply-driven; [168, 131, 142]), or from information requirements (i.e., demand-driven; [69, 115, 146]). Other approaches have also proposed hybrid solutions, combining both information about data sources and information requirements (e.g., [113]). However, the evolution of the design and integration of new information requirements is mostly overlooked, or at best left to be done manually, following the proposed guidelines (e.g., [69, 115]).

To cope with such complexity (both at the beginning of the design process and when potential evolution events occur), in Chapter 4, we present a semi-automatic method called ORE, for creating DW designs in an iterative fashion based on a given set of information requirements. We start from MD interpretations of individual requirements and present an iterative method that consolidates them into a single, unified MD schema design satisfying the entire set of information requirements. ORE is useful for the early stages of a DW project, when we need to create an MD schema design from scratch, but it can also serve during the entire DW lifecycle to accommodate potential evolution events. The correctness of the proposed MD schemata is guaranteed according to the MD integrity constraints [114]. While incrementally building a MD schema design of a DW, ORE systematically traces the meta data about the required schema transformation actions in the case of evolution changes. On the one side, such metadata is used for enabling designers to reconsider some of the previous design choices, while on the other side, it also provides a valuable information for adapting back-end ETL processes to the evolution

changes. Moreover, being highly correlated artifacts in a DW system, having semi-automatic methods for incremental design of MD schema and ETL processes (i.e., ORE and CoAl), both methods can benefit from the information inferred by one another. For instance, the aggregation and normalization levels of the produced MD schema could be considered, since this would affect the way the appropriate ETL process is tailored (i.e., trade-offs between materialized views and OLAP querying). Similarly, checkpointing or bottlenecks detected at the ETL level may cause some changes at the MD schema for the sake of performance.

4.4 Chapter 5: Engine Independence for Logical Analytic Flows (Data Flow Deployer)

As we previously discussed, the design of DIFs may be done either from high level information requirements, in which case the design is first typically done at the conceptual or logical (engine independent) level, or in the case of smaller, ad hoc projects, the design is typically done directly at the physical (engine specific) level. In both cases, given modern enterprise environments, answering business needs in the most efficient manner, may require complex DIFs that integrate data sources from diverse sets of storage repositories, as well as computing from different execution engines, typically deployed by today's enterprises, such as Map-Reduce systems, stream processing systems, statistical analysis engines.

On the one side, creating and deploying such complex DIFs for different execution engines is labor-intensive, time-consuming, and error prone. On the other side, having a DIF spanning different execution engines often conceals possibilities for more extensive optimization actions, as engines typically consider parts of these flows in isolation, hence the holistic picture of such (hybrid [163]) data flows is not available. Thus, data flow designers are in a need for a notion of engine independence for logical analytic flows. Just as logical data independence insulates a data modeler from physical details of the relational database, there are benefits in designing flows at a logical level and using automation to implement the flows.

In Chapter 5, we present a system (called BabbleFlow) that provides engine independence for DIFs. We first describe a language for encoding flows at a logical level, namely xLM. Furthermore, we study how a previously designed logical flow (e.g., by means of our CoAl approach), can be automatically translated to a physical flow (and executable code) for a targeted processing engine(s). Lastly, we also show how existing, physical flows, written for specific processing engines, can be imported, composed in a single (hybrid) flow, and converted to a logical flow that is engine independent.

Other tools, like ETL design GUIs, offer to some extent separation between logical design and implementation, but the design is still tool-specific. Our

work here goes beyond that. We envision hybrid data flow processors, like Hybrid Flow Management System (HFMS) proposed in [164], where logical flows may span multiple engines, which can be seen as peers enabling data and function shipping between all. Such functionalities largely support inter-engine optimization of these hybrid flows. Besides optimization, one might decompose a single, large, complex DIF into smaller subflows to reduce contention in a workload or to improve maintainability of the flow. Another, not unusual scenario for hybrid data-intensive ecosystems is to have an algorithm encoded for one engine (e.g., written in Map-Reduce) that one wishes to apply to data in a different engine (e.g., database). So, rather than shipping the data to the algorithm, our data flow translation techniques enable shipping the algorithm to the data.

In Chapter 5, our focus is not on specific flow processors or use cases. Rather, our goal is to support, on the one side, creating complex (hybrid) data flows in an engine independent manner, as well as to support deployment of previously built DIFs over a variety of execution engines.

4.5 Chapter 6: Supporting Job Scheduling with Workload-driven Data Redistribution (Data Flow Scheduler)

Once the designed DIFs are deployed to the selected execution engines, one may assume that their execution can be handed to the engine specific planner and executor (e.g., Apache Hadoop YARN [191]). While this is true in most of the cases, still a flow designer may choose not to simply rely on the provided functionality of these data processing engines. Consider for instance distributed data processing systems, like Hadoop, which have emerged as a necessity for processing, in a scalable manner, large-scale data volumes in clusters of commodity resources. While they typically provide fault-tolerant, reliable, and scalable platforms for distributed data processing, the network traffic is identified as a bottleneck for the performance in most of these systems [85].

Current scheduling techniques in Hadoop typically follow a query shipping approach where the tasks are brought to their input data, hence data locality is exploited for reducing network traffic. However, such scheduling techniques make these systems sensitive to the specific distribution of data in the cluster, and when skewed, it can drastically affect the performance of data processing applications. At the same time, underlying distributed data storage systems, which are typically independent of the application layer, do not consider the imposed workload when deciding data placements in the cluster. For instance, Hadoop Distributed File System (HDFS) places data block replicas randomly in the cluster following only the data availability policies, hence without a guarantee that data will be uniformly distributed among DataNodes [156]. Some of them provide certain rebalancing techniques, which are however still blindly done, without considering the real workload imposed by executing DIFs.

In Chapter 6, we address these challenges and present a workload-driven approach for data redistribution, called H-WorD, to support scheduling of DIFs. H-WorD leverages on having a complete overview of the cluster workload and automatically decides on a better redistribution of workload and data. We exemplify our approach using a well-known MapReduce model [43], implemented inside the Apache Hadoop system [191].

In particular, H-WorD comprises an algorithm for supporting scheduling of DIFs with workload-driven data redistribution. H-WorD starts from a set of previously profiled DIFs (e.g., MapReduce jobs) that are planned for execution in the cluster; e.g., a set of jobs currently queued for execution in a batch-queuing grid manager system. The cluster workload is initialized following commonly used scheduling techniques (i.e., exploiting data locality, hence performing query shipping). Then, H-WorD iteratively reconsiders the current workload distribution by proposing different execution scenarios (e.g., executing map tasks on nodes without local data, hence performing also data shipping). In each step, it estimates the effect of a proposed change to the overall cluster workload, and only accepts those that potentially improve certain quality characteristics.

We focus here on improving the overall makespan of the jobs that are planned for execution. As a result, after selecting execution scenarios, H-WorD identifies the tasks of DIFs that would require data shipping (i.e., transferring their input data from a remote node). On the one hand, our techniques can be used offline, complementary to existing MapReduce scheduling techniques, to automatically instruct redistribution of data beforehand. On the other hand, our final goal here envisions more sophisticated scheduling techniques where H-WorD can be used on the fly, to take advantage of a priori knowing potentially needed data transfers, and leveraging on idle network cycles to schedule such data transfers in advance, without deferring other tasks' executions.

5 Contributions

The first contribution of this PhD thesis is the study of the current field of DIFs. In particular, we survey the existing approaches focusing on the characteristics that best describe the challenges of moving towards next-generation BI systems. We analyze both the foundational work of the database theory, and recent approaches for DIFs. As a result, we envision an architecture for managing the complexity of the DIF lifecycle in the next generation BI setting. The results of this study further help us to identify the remaining challenges for the DIF lifecycle that require further attention.

Furthermore, we map the main contributions of this PhD thesis to the corresponding phases of the DIF lifecycle that they facilitate.

- Design. In the context of the design of DIFs, we provide support for re-engineering DIFs coming from a variety of engine-specific formats, in order to enable further engine-independent composition, analysis, and optimization of DIFs. In particular, we first define a logical (engine-independent) format, called xLM, as well as an extensible translation method for converting flows coming from different engines into a logical format. Moreover, we also provide support for the composition of hybrid data flows, by means of building a unified logical DIF, that in reality can span multiple storage and execution engines.

- Optimization. We provide support for multi-flow optimization of DIFs. In particular, we provide a novel algorithm, called CoAl for consolidating DIFs. We define generic methods for reordering and comparing data flow operations, which are applied while searching for a consolidation solution to increase data and operation reuse.

- Implementation. We propose an extensible method, called BabbleFlow, for supporting the implementation and deployment of DIFs on a variety of execution engines. Such method can be used complementary with engine selection or multi-engine optimization approaches to assist the optimal deployment of DIFs over the selected engines.

- Execution. We provide support for scheduling the execution of DIFs. In particular, we introduce a novel algorithm, named H-WorD, that led by the real workload of DIFs, decides on data redistribution that would improve the performance of executing these DIFs in a distributed data processing system.

- Maintenance. Lastly, we also largely support the maintenance of DIFs in front of the evolving business needs. In particular, we present a semi-automatic approach to the incremental design of data-intensive flows (i.e., CoAl). We define a framework for accommodating new, changed, or removed information requirements into an existing DIF design. Moreover, in the case of new information requirements, the CoAl algorithm tackles the problem of integrating new data flows from the context of data and code reuse. In addition, in the DW context, we also tackle the problem of supporting the evolution of a target MD schema design in front changed business needs. We propose a novel method, called ORE for accommodating changed business needs to the existing MD schema design, by means of efficiently integrating target MD schemata coming from different information requirements. Lastly, we introduce the traceability metadata structure to systematically record valuable information about the MD schema integration, thus providing valuable information for supporting the redesign of back-end ETL processes.

Chapter 2

A Unified View of Data-Intensive Flows in Business Intelligence Systems: A Survey

The paper has been accepted for publication in the
in Transactions on Large-Scale Data- and Knowledge-Centered Systems, 2016.
The layout of the paper has been revised.
DOI: TBD

Abstract

Data-intensive flows are central processes in today's business intelligence (BI) systems, deploying different technologies to deliver data, from a multitude of data sources, in user-preferred and analysis-ready formats. To meet complex requirements of next generation BI systems, we often need an effective combination of the traditionally batched extract-transform-load (ETL) processes that populate a data warehouse (DW) from integrated data sources, and more real-time and operational data flows that integrate source data at runtime. Both academia and industry thus must have a clear understanding of the foundations of data-intensive flows and the challenges of moving towards next generation BI environments. In this chapter we present a survey of today's research on

data-intensive flows and the related fundamental fields of database theory. The study is based on a proposed set of dimensions describing the important challenges of data-intensive flows in the next generation BI setting. As a result of this survey, we envision an architecture of a system for managing the lifecycle of data-intensive flows. The results further provide a comprehensive understanding of data-intensive flows, recognizing challenges that still are to be addressed, and how the current solutions can be applied for addressing these challenges.

1 Introduction

Data-intensive flows are critical processes in today's business intelligence (BI) applications with the common goal of delivering the data in user-preferred and analysis-ready formats, from a multitude of data sources. In general, a data-intensive flow starts by extracting data from individual data sources, cleans and conforms extracted data to satisfy certain quality standards and business requirements, and finally brings the data to end users.

In practice, the most prominent solution for the integration and storage of heterogeneous data, thoroughly studied in the past twenty years, is data warehousing (DW). A DW system assumes a unified database, modeled to support analytical needs of business users. Traditionally, the back stage of a DW system comprises a data-intensive flow known as the extract-transform-load (ETL) process responsible of orchestrating the flow of data from data sources towards a DW. ETL is typically a batch process, scheduled to periodically (e.g., monthly, daily, or hourly) load the target data stores with fresh source data. In such a scenario, limited number of business users (i.e., executives and managers) are expected to query and analyze the data loaded in the latest run of an ETL process, for making strategic and often long-term decisions.

However, highly dynamic enterprise environments have introduced some important challenges into the traditional DW scenario.

- Up-to-date information is needed in near real-time (i.e., right-time [70]) to make prompt and accurate decisions.
- Systems must provide the platform for efficiently combining in-house data with various external data sources to enable context-aware analysis.
- Systems must be able to efficiently support new, unanticipated needs of broader set of business users at runtime (i.e., on-the-fly).

These challenges have induced an important shift from traditional business intelligence (BI) systems and opened a new direction of research and practices. The next generation BI setting goes by various names: operational BI (e.g., [40, 50]), live BI (e.g., [42]), collaborative BI (e.g., [19]), self-service BI (e.g., [7]), situational BI (e.g., [110]). While these works look at the problem from

different perspectives, in general, they all aim at enabling the broader spectrum of business users to access a plethora of heterogeneous sources (not all being under the control of the user's organization and known in advance), and to extract, transform and combine these data, in order to make right-time decisions. Consequently, here, we generalize these settings and use the common term next generation BI, while for the old (DW-based) BI setting we use the term traditional BI. An interesting characterization of the next generation BI setting is given by Eckerson [50]: "...operational BI requires a just-in-time information delivery system that provides the right information to the right people at the right time so they can make a positive impact on business outcomes."

Obviously, in such a scenario periodically scheduled batch loadings from preselected data stores have become unrealistic, since fresh data are required in near real-time for different business users, whose information needs may not be known in advance. In fact, effectively integrating the traditional, batched decision making processes, with on-the-fly data-intensive flows in next generation BI systems is discussed to be important to satisfy analytic needs of today's business users (e.g., [9, 42, 73]). For example, a part of in-house company's sales data, periodically loaded to a DW by an ETL, can be at runtime crossed with the Web data to make context-aware analysis of business decisions. To build such demanding systems, one must first have a clear understanding of the foundation of different data-intensive flow scenarios and the challenges they bring.

From a theoretical perspective, handling data heterogeneity has been separately studied in two different settings, namely data-integration and data exchange. Data integration has studied the problem of providing a user with a unified virtual view over data in terms of a global schema [106]. User queries over the global schema, are then answered by reformulating them on-the-fly in terms of data sources. On the other side, data exchange has studied the problem of materializing an instance of data at the target that reflects the source data as accurately as possible and can be queried by the user, without going back to the original data sources [55].

A recent survey of ETL technologies [186] has pointed out that the data exchange problem is conceptually close to what we traditionally assume by an ETL process. Intuitively, in an ETL process, we also create an instance at the target, by means of more complex data transformations (e.g., aggregation, filtering, format conversions, deduplication). However, the trends of moving towards the next generation BI settings have brought back some challenges initially studied in the field of data integration, i.e., requiring that the user queries should be answered by extracting and integrating source data at runtime. Moreover, the next generation BI settings brought additional challenges into the field of data-intensive flows (e.g., low data latency, context-awareness).

Right-time decision making processes demand close to zero latency for data-intensive flows. Hence the automated optimization of these complex flows is

a must [42, 78], not only for performance, but also for other quality metrics, like fault-tolerance, recoverability, etc. [161]. Considering the increasing complexity of data transformations (e.g., machine learning, natural language processing) and the variety of possible execution engines, the optimization of data-intensive flows is one of the major challenges for next generation BI systems.

Even though the above fields have been studied individually in the past, the literature still lacks a unified view of data-intensive flows. In this chapter, we aim at studying the characteristics of data-intensive flows in next generation BI systems. We focus on analyzing the main challenges in the three main stages when executing data-intensive flows, i.e., (1) data extraction, (2) data transformation, and (3) data delivery. Having low data latency as an important requirement of data-intensive flows in the next generation BI setting, we additionally analyze the main aspects of data flow optimization. We analyzed these four areas inside the two main scenarios for data-intensive flows: (a) periodically executed, batched processes that materialize and load data at the target data store for future analysis (extract-transform-load - ETL), and (b) on-the-fly, instantaneous data flows executed on demand upon end-users' query (extract-transform-operate - ETO).

In addition, we identify that a well-known BigData challenge, namely, one of the so called 3 V's [81] (i.e., massive volumes of data), is an important one also for the design and even more deployment of data-intensive flows in next-generation BI systems. However, the approaches that deal with such challenge represent a separate and rather extensive field of research, which is out of scope of this study. We thus refer an interested reader to [35] for more detailed overview of the approaches dealing with the BigData challenges.

As the first result, we identify the main characteristics of data-intensive flows, focusing on those that best describe the challenges of moving towards the next generation BI setting. Then, in terms of these characteristics we classify the approaches, both from the foundational works and more recent literature, tackling these characteristics at different levels. On the one hand, the results provide us with a clear understanding of the foundations of data-intensive flows, while on the other hand, identified characteristics help us defining the challenges of moving towards the next generation BI setting.

Finally, as the main outcome of this study, we outlined the envisioned architecture of next generation BI systems, focusing on managing the complete lifecycle of data-intensive flows.

Contributions. In particular, our main contributions are as follows.

- We analyzed current approaches, scrutinizing the main aspects of data-intensive flows in today's BI environments.
- We define the main characteristics of data-intensive flows, focusing on

those that best describe the challenges of a shift towards the next generation BI.

- In terms of the dimensions defined from these characteristics, we analyze both the foundational work of database theory, and recent approaches for data-intensive flows, at different levels of these dimensions.
- Resulting from this study, we envision an architecture for managing the complexity of data-intensive flows in the next generation BI setting.
- Finally, we indicate the remaining challenges for data-intensive flows, which require further attention from both academia and industry.

Outline. In Section 2, we first introduce an example scenario used to support our discussions throughout this chapter. We then in Section 3, describe the methodology used in our study, and outline the main study setting. Next, in Section 4 we discuss the process of defining the dimensions that are further used for studying data-intensive flows. In Sections 5 - 8, we analyze different approaches from data-intensive flows inside the previously defined dimensions. In Section 9 we provide the overall discussion and introduce an envisioned architecture of a system for managing data-intensive flows in the next generation BI setting, while in Section 7, we conclude this chapter.

2 Example Scenario

We first introduce an example scenario to support discussions throughout this chapter and to motivate our study. Our example scenario is motivated by the data model introduced for the BigData benchmark (a.k.a. BigBench) in [62], which extends the TPC-DS benchmark[1] for the context of BigData analytics. Notice that we adapted their scenario to make the examples more intuitive and suitable to our discussions. In particular, besides the typical operations found in relational database systems, i.e., Join, Selection (Filter), Aggregation (Aggr.), Sort, and Distinct (Remove Duplicates), in the following example scenarios, we also introduce more complex operations typically found in today's data-intensive flows; that is, (1) User Defined Functions (UDF) that may implement either simple arithmetic expressions or complex, typically black-box operations, (2) Match that implements more relaxed join or lookup semantics (e.g., using approximate string matching), and (3) Sentiment Analysis that typically applies natural language processing techniques for extracting subjective (opinion) information from the Web sources (e.g., forums, social networks).

In general, we consider a simple case of a retail company that has different databases that support its daily operational processes. These databases

[1] `http://www.tpc.org/tpcds/spec/tpcds_1.1.0.pdf` (last accessed 4/4/2014)

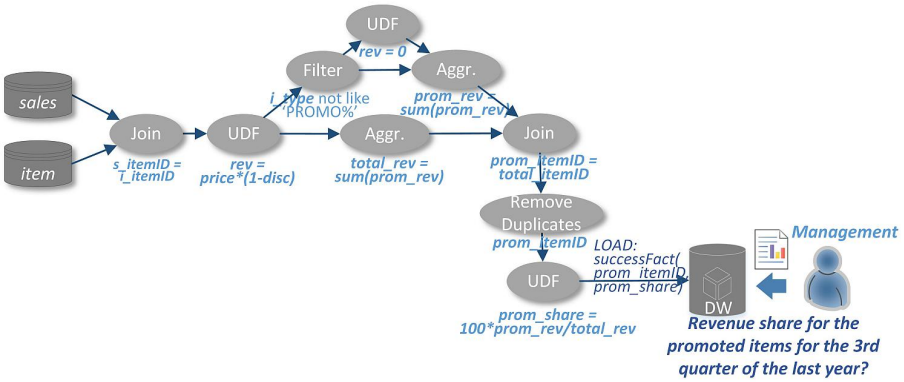


Fig. 2.1: Example 1.1: ETL to analyze revenue share from a promotion

cover the information about different items (i.e., products) offered for the sale, company's customers, their orders, shipping information, etc. Periodically, the company launches campaigns and puts some product on a promotion.

Scenario 1. Originally, the company has used a traditional BI system with a centralized DW that is loaded by means of different ETL flows (one of which is conceptually depicted in Figure 2.1). Users in this scenario are typically upper management executives that analyze the enterprise-wide data (e.g., `items` and their `sales`) to make decisions for making strategic actions (e.g., launching promotional campaigns).

Example 1.1. In the specific example in Figure 2.1, the ETL flow is periodically executed to load the DW with information about the percentage of the revenue share made from the items that were on the promotion. Quarterly, the management analyzes how the previous business decisions on promotional campaigns affected the revenue.

Scenario 2. While the above setting has served the company well in having a periodical feedback about the previous strategic decisions, today's dynamic markets require more prompt reaction to the potentially occurring problems (e.g., hourly or daily). The company thus noticed that instead of waiting for the sales data to analyze the success of the promotional campaign, they can potentially benefit from the opinions that customer may leave about the campaign and product items, in the form of `reviews` over the Web (e.g., social networks) and react faster to improve the potential revenue. Moreover, the company also noticed that such an analysis should be decentralized to the regional and local representatives and available to a broader set of users involved in the business process. As fast decisions are needed, the users must be able to make them at right time (i.e., "...before a problem escalates into a crisis or a fleeting opportunity disappears..." [50]). We consider the two following

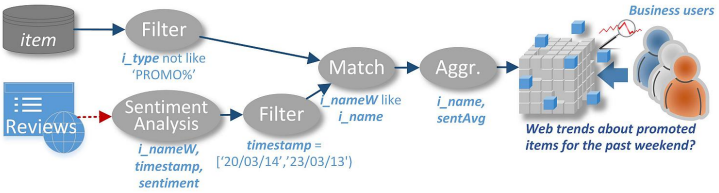


Fig. 2.2: Example 2.1: ETO to predict the success of a promotion

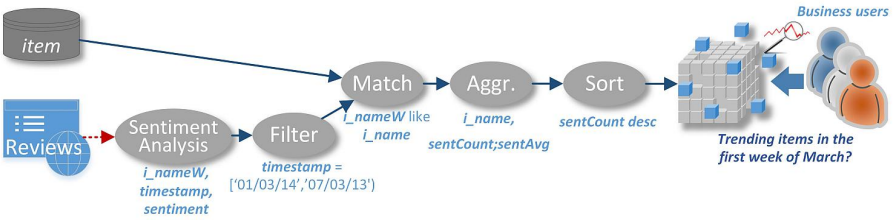


Fig. 2.3: Example 2.2: ETO to analyze the trends for launching a new promotion

business requirements posed on-the-fly, and the two data-intensive flows that answer them, conceptually depicted in Figure 2.2 and Figure 2.3.

Example 2.1. In the former example (Figure 2.2), a regional manager decides to analyze the potential success of the promotional campaign launched on Friday, after the first weekend, by inspecting the sentiment (i.e., opinions) of the real and potential customers about the product `items` that are included in the promotion.

Example 2.2. The latter example (Figure 2.3), on the other hand, analyzes the currently trending product `items` and user opinions about them for deciding which items to include in the next promotional campaign.

In both cases, business users interactively make on-the-fly and context-aware analysis, in order to quickly react and improve their business decisions.

3 Methodology

We further introduce the methodology used for studying data-intensive flows and outline the resulting study setting. We start by introducing the process of selecting the literature to be included in this study. The study further includes three consecutive phases, which we explain in more detail in the following subsections.

3.1 Selection process

The literature exploration started with the keyword search of the relevant works inside the popular research databases (i.e., Scopus[2] and Google Scholar[3]). In Phase I, we focused on the keywords for finding seminal works in the DW and ETL field (i.e., "data warehousing", "ETL", "business intelligence"), as well as the most relevant works on the next generation BI (i.e., "next generation business intelligence", "BI 2.0", "data-intensive flows", "operational business intelligence"). While in the case of traditional DW and ETL approaches we encountered and targeted the most influential books from the field (e.g., [96, 82]) and some extensive surveys (e.g., [186]), in the case of the next generation BI approaches, we mostly selected surveys or visionary papers on the topic (e.g., [7, 9, 42, 40, 73]). Furthermore, the following phases included keyword search based on the terminology found in the created study outline (see Figure 2.4), as well as the identified dimensions (see Figure 2.5).

Rather than being extensive in covering all approaches, we used the following criteria for prioritizing and selecting a representative initial set of approaches that we studied.

- The relevance of the works to the field of data-intensive flows and the related topics, i.e., based on the abstract/preface content we discarded the works that did not cover the topics of interest.
- The importance of the works, i.e., number of citations, importance of the venue (e.g., ranking of the conference[4] or impact factor of the journal[5]).
- The maturity of the works, i.e., extensiveness and completeness of a theoretical study or a survey, experimental results, applicability in the real world.

Furthermore, we also followed the snowballing technique and included, previously not found, but relevant approaches referenced from the initial ones.

3.2 Phase I (Outlining the study setting).

First phase included the review of the seminal works on traditional data-intensive flows, ETL, and data warehousing in general; as well as the relevant works discussing the next generation BI systems and their main challenges on moving toward (near) real-time data analysis.

As a result, we outline the setting for studying data-intensive flows. Specifically, in our study we aim at analyzing two main scenarios of data-intensive flows present in today's BI settings, namely:

[2] https://www.scopus.com/
[3] https://scholar.google.com
[4] CORE conference ranking: http://portal.core.edu.au/conf-ranks/
[5] Thomas Reuters Impact Factor: http://wokinfo.com/essays/impact-factor/

- extract-transform-load (ETL). In the traditional BI setting, data are extracted from the sources, transformed and loaded to a target data store (i.e., a DW). For posing analytical queries (e.g., OLAP), the business users in such a scenario solely rely on the data transferred in periodically scheduled time intervals, when the source systems are idle (e.g., at night time).

- extract-transform-operate (ETO). Next generation BI has emerged as a necessity of companies for combining more instantaneous decision making with traditional, batched processes. In such a scenario, a user query, at runtime, gives rise to a data flow that accesses the data sources and alternatively crosses them with already loaded data to deliver an answer.

Kimball and Caserta [96] introduced the following definition of an ETL process "A properly designed ETL system extracts data from the source systems, enforces data quality and consistency standards, conforms data so that separate sources can be used together, and finally delivers data in a presentation-ready format so that application developers can build applications and end users can make decisions."

Being general enough to cover the setting of data-intensive flows studied in this chapter (i.e., both previous scenarios), we follow this definition and first divide our study setting into three main stages, namely:

i Data extraction. A data-intensive flow starts by individually accessing various (often heterogeneous) data sources, collecting and preparing data (by means of structuring) for further processing.

ii Data transformation. Next, the main stage of a data-intensive flow transforms the extracted data by means of cleaning it for achieving different consistency and quality standards, conforming and combining data that come from different sources.

iii Data delivery. This stage is responsible for ensuring that the data, extracted from the sources, transformed and integrated are being delivered to the end user in a format that meets her analytical needs.

Related fields. To complete the data-intensive flows lifecycle, in addition to the main three stages, we revisit two fields closely related to data-intensive flows, i.e., data flow optimization and querying.

iv Data flow optimization considers data-intensive flow holistically and studies the problem of modifying the given data flow, with the goal of satisfying certain non-functional requirements (e.g., performance, recoverability, reliability). Obviously, the optimization problem is critical for data flows in today's BI systems, where the data delivery is often required in the near real-time manner.

In addition, for the completeness of the overall picture, we briefly analyze what challenges the two main scenarios in data-intensive flows (i.e., ETL and ETO) bring to querying.

v The requirements elicitation and analysis (i.e., querying) stage mainly serves for posing analytical needs of end users over the available data. This stage is not actually part of a data-intensive flow execution, but depending on the scenario (i.e., either ETO or ETL), can respectively come as a preceding or subsequent stage for a data-intensive flow execution. At the lower level of abstraction, end users' analytical needs are typically expressed in terms of queries (e.g., SQL), programs (e.g., ETL/MapReduce jobs), or scripts (e.g., Pig Scripts), which are then automatically translated to data-intensive flows that retrieve the needed data. The typical challenges of querying the data in the next generation BI setting concern the ability of the system to adapt and complement users' analytical needs by means of discovering related, external data, and the usability of a BI system for posing analytical needs by end-users. The former challenge may span from the traditional DW systems that typically answer user's OLAP queries solely by exploiting the data previously loaded into a DW (by means of an ETL process), to situation-(context-)aware approaches that considering end user queries, explore, discover, acquire, and integrate external data [7, 9]. Regarding the latter challenge, we can also observe two extreme cases: traditional querying by means of standard, typically declarative query languages (e.g., SQL, MDX), and approaches that enable users to express their (often incomplete) analytical needs in a more natural and human-preferred manner (e.g., keyword search, natural language). Recently, some researchers have proposed more flexible (semi-structured) query language (SQL++) for querying a variety of both relational and new NoSQL databases that may store data in a variety of formats, like JSON or XML [125].

  Other approaches also tackled the problem of providing a higher level of abstraction for posing information requirements, more suitable for business users. As analyzing information requirements and systematically incorporating them into a design of data-intensive flows has been typically overlooked in practice, initial efforts were mostly toward systematic requirements analysis in BI and DW projects (e.g., [65, 196]). However, such approaches still require long processes of collecting the requirements at different levels of abstraction and their analysis, before manually incorporating them into a data-intensive flow design. Thus, they obviously cannot be applied in ETO scenarios, where the generation of data-intensive flows to fulfill end user analytical needs is expected in near real-time. Other works identified such problem and proposed certain automation to such time-lasting process (e.g., [146]). However, the automation required lowering the level

of abstraction for defining information requirements, tightening them to the multidimensional model notation (i.e., facts and dimensions of analysis). As an extension to this approach the authors further tried to raise the level and provide an abstraction of the data sources' semantics in terms of a domain ontology, with its graph representation. This has partially hidden the model specific details from business users, and allowed them to pose information requirements using a domain vocabulary. Recent work in [60] conducted an extensive study of decision support system approaches, identifying how they fit the traditional requirements engineering framework (i.e., [132]). As a result, the authors identified a need for systematic and structured requirements analysis process for further raising the level of automation for the design of decision support systems (including the design of data-intensive flows), while at the same time keeping the requirements analysis aware of all stakeholder needs. We have discussed here the main challenges that requirements elicitation and analysis introduces to the field of data-intensive flows in the next generation BI setting, but we omit further analysis as it falls out of the scope of this work.

As a result, we define a blueprint for studying data-intensive flows, depicted in Figure 2.4. Going from top down, we depict separately the two main scenarios of data-intensive flows studied in this chapter (i.e., ETL and ETO). Going from left to right, the first part of Figure 2.4 (i.e., A, E) depicts the characteristics of the data extraction stage in terms of the complexity that different input data types bring to a data-intensive flow; then the following part (i.e., B, F) covers the characteristics of the data transformation stage; the penultimate part (i.e., C, G) covers the data delivery stage, while the last (most right) part (i.e., D, H) covers querying. Being rather a holistic field (i.e., taking into account the complete data-intensive flow), the optimization spans all stages of data-intensive flows and it is depicted at the bottom of Figure 2.4.

3.3 Phase II (Analyzing the characteristics of data-intensive flows).

This phase included the review of the works that scrutinize the characteristics of data-intensive flows in both previously defined scenarios, i.e., ETL and ETO. This phase aimed at characterizing data-intensive flows in terms of the features that best indicate the movement toward the next generation BI setting.

To this end, we performed an incremental analysis of the included works to discover the features of data-intensive flows they have tackled. We started from the papers that individually covered the traditional and the next generation BI settings. The identified features are then translated into the dimensions for studying data-intensive flows (see Figure 2.5). As new dimensions are discovered, the related papers are reconsidered to analyze their assumptions regarding the new dimensions. Each discovered dimension determines the levels, supported or envisioned by analyzed approaches, in which these approaches

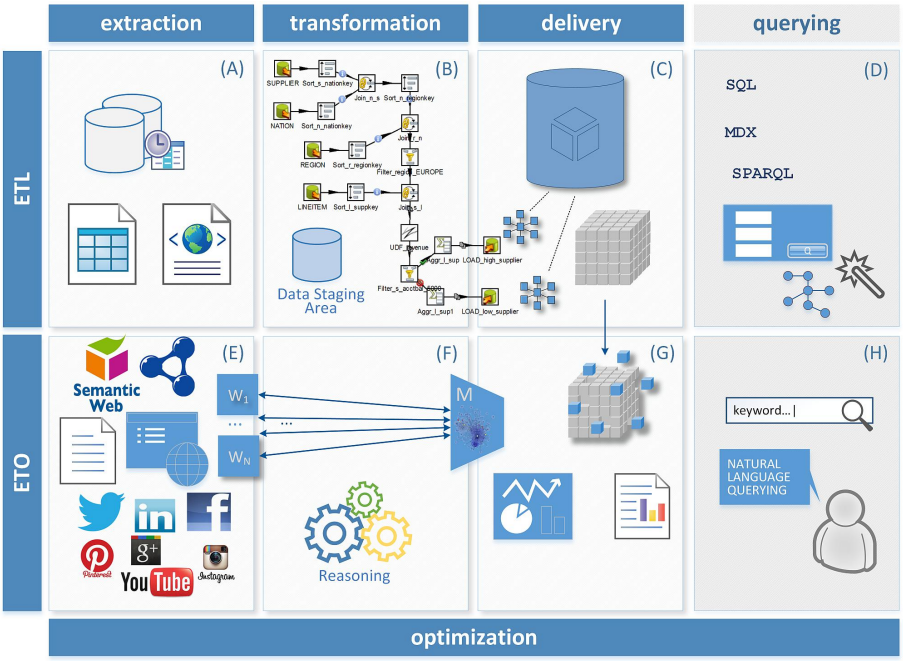


Fig. 2.4: Study setting for data-intensive flows

attain the corresponding feature of data-intensive flows. Eventually, we converged to a stable set of dimensions, which can be further used for studying and classifying the approaches of data-intensive flows.

In Section 4, we discuss in more detail the process of discovering dimensions for studying data-intensive flows, and further provide their definitions.

3.4 Phase III (Classification of the reviewed literature).

In this phase, we further extend the study to the works that more specifically cover the previously discussed areas of data-intensive flows (i.e., data extraction, data transformation, data delivery, and data flow optimization). We classify the reviewed approaches using the previously defined dimensions, which build our study setting (see Figure 2.5), and present the results of this phase in Sections 5 - 8. We summarize the classified approaches of the three main stages (i.e., data extraction, data transformation, and data delivery) respectively in Tables 2.1 (page 39) - 2.3 (page 49), and the optimization approaches in Table 2.4 (page 53). We mark the level of the particular dimension (i.e., challenge)

that each approach achieves or envisions (i.e., `Low`, `Medium`, or `High`)[6].

In addition, we also classify the approaches in Tables 2.1 - 2.4 based on the fact if they are potentially applicable in ETL, ETO, or both ETL and ETO scenarios.

Finally, for each reviewed approach we define the technology readiness level, focusing on the first four levels of the European Commission scale [51].

- `TRL1` ("basic principles observed"), refers to work that either based on practical use cases or reviewed literature observes the basic principles that should be followed in practice (e.g., guidelines, white or visionary papers)
- `TRL2` ("technology concept formulated"), refers to work that provide theoretical underpinnings of the studied area, which are not always directly applicable in practice, but represent an important foundation for principles that should be followed in practice (e.g., the database theory works on data exchange and data integration).
- `TRL3/TRL4` ("experimental proof of concept"/"technology validated in lab"), refers to the system-oriented work that provide the proof of concept solution for an observed principle from the previous two levels, validated either over synthetic (`TRL3`) or real-world use cases (`TRL4`).

4 Defining dimensions for studying data-intensive flows

For each area in the outlined study setting for data-intensive flows (Figure 2.4), we discuss in more detail, and further provide the definitions of the dimensions through which the reviewed works on data-intensive flows are analyzed (see Figure 2.4). Then, in the following sections 5 - 8, we discuss in more detail the works specifically covering each of the studied areas.

4.1 Data Extraction

The most commonly discussed challenge when extracting data (e.g., [7, 96]) is related to the format in which the data are provided, i.e., structuredness.

Structuredness determines the level, in which data in data sources under analysis follow a certain model, constraints or format. It spans from highly structured data that follow strictly defined models and ensures certain constraints over data (`High`), like relational (see top left of Figure 2.4); then semi-structured data that are represented in a repetitive [83], standard and easily

[6] The exception to this are the approaches from the data flow optimization area, for which we introduced levels that more precisely describe the consequences of their placement inside the corresponding dimensions. Moreover, in the cases when the approach is completely independent of the level for a particular dimension, we mark it as non-applicable (`N/A`).

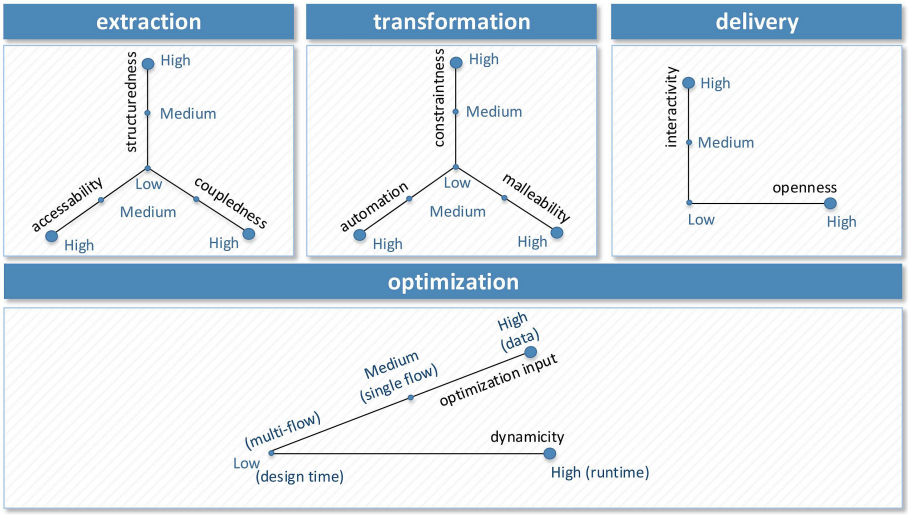


Fig. 2.5: Dimensions for studying data-intensive flows

parsable format, but that do not enforce strong constraints over data (`Medium`), like XML, CSV, RDF (see middle left in Figure 2.4); and unstructured data in a free-form (textual or non-textual) that require smarter techniques for extracting real value from it (`Low`), like free text, photos, x-rays, etc. (see bottom left of Figure 2.4). □

Other characteristics of this stage are related to the degree in which BI applications need to be coupled with source systems when extracting the data, i.e., coupledness, and the reliability of accessing these systems, i.e., accessibility.

Coupledness determines the level, in which data-intensive flows depend on a specific knowledge or components obtained from data sources under analysis. It spans from typical ETL processes that consolidate organization's ("in-house") operational sources with predictable access policies (e.g., DBMS, ERP systems; see top left of Figure 2.4), using extraction algorithms (e.g., incremental DB snapshots) that strongly depend on information from source systems (`High`), e.g., materialized views DW solutions, triggers, source components modifications; then the systems that use logs, timestamps or other metadata attached to data sources (`Medium`); and the scenarios where we cannot expect any in-advance knowledge about the data sources, but the system needs to discover and integrate them on-the-fly [9] (`Low`), e.g., Web data, Link Open Data. □

Accessability determines the level, in which one can guarantee a "non-stop" access to certain data sources. It spans from "in-house", highly available data (`High`), like ERP, ODS systems; then the external data usually provided by data providers that under SLAs can guarantee certain accessibility to data (`Medium`);

and external data sources that are completely out of the organization's control (`Low`), like open, situational, or Web data. □

4.2 Data Transformation

In this stage, the most common issue is related to the complexity of data transformations inside a flow (e.g., [186]), and more specifically to the degree in which we can automate the design of a data-intensive flow, i.e., automation. While the design of an ETL process is known to be a demanding task (e.g., [170]; see the example ETL process in the top middle of Figure 2.4) and its automation (even partial) is desirable and has been studied in the past (e.g., [120, 146]), ETO depends on fully automated solutions for answering user queries at runtime (e.g., by means of reasoning; see Figure 2.4).

Automation determines the level, in which one can automate the design of data-intensive flows. It spans from the works that propose modeling approaches to standardize the design of data flows, especially in the context of ETL processes (`Low`), e.g., [190, 180, 12]; then approaches that provide guidelines and/or frequently used patterns to facilitate the design of a data-intensive flow (`Medium`), e.g., [96, 189]; and approaches that attempt to fully automate the generation of data-intensive flows as well as their optimization (`High`), e.g., [146, 52, 44]. □

Other important characteristics that distinguish ETO from a traditional ETL process are the degree of data constraintness that a flow must ensure, and the flexibility (i.e., malleability) of a data-intensive flow in dealing with the changes in the business context.

Constraintness determines the level, in which data-intensive flows must guarantee certain restrictions, constraints, or certain level of data quality over the output data. It spans from fully constrained data, usually enforcing the MD model (MD integrity constraints [114]), and high level of data cleanness and completeness required to perform further data analysis, like OLAP (`High`); then, data-intensive flows that may provide ad-hoc structures for answering user queries (e.g., reports), without a need to enforce the full completeness and cleanness of data (`Medium`); and as an extreme case we consider the concept of data lakes where no specific schema is specified at load time, but rather flexible to support different analysis over stored and shared data at read-time (`Low`).□

Malleability determines the level in which a system is flexible in dealing with the changes in the business context (e.g., new/changed data sources under analysis, new/changed/removed information requirements). It spans from the traditional DW settings where data sources as well as information requirements are static, typically gathered in advance. and are added manually to the analysis only at design time, while any change would require the redesign of a complete data-intensive flow (`Low`) [96]; then systems that tackle the incremental evolution of data-intensive flows in front of new requirements and data

sources (`Medium`) [89]; and dynamic approaches that consider discovering new, usually external data at runtime (`High`) [9]. □

4.3 Data Delivery

For delivering the transformed data at the target, we have identified two main characteristics that distinguish ETL from ETO, namely the interactivity of data delivery as perceived by the end-user; and the openness of the delivered data, which refers to the degree in which the approaches assume the delivered information to be complete (i.e., closed vs. open world assumption).

Interactivity determines the level in which a system interacts with the end-user when executing a data-intensive flow and delivering the data at the output. It spans from traditional ETL processes that typically deliver the data (i.e., materialize the complete data to load a DW) in a batched, asynchronous process, without having an interaction with an end-user (`Low`), then approaches that based on the overall cost, select data to be partially materialized (e.g., loaded in a batch to materialized views), and those that are queried on-the-fly (`Medium`); and finally completely interactive approaches that assume on-the-fly data flows which deliver the data to answer user queries for immediate use only, e.g., for visualization (`High`). □

Openness determines the level, in which the delivered data are considered open to different interpretations. It spans from closed-world assumption approaches typical for traditional databases, where the data are considered complete and any answer to a user query is determined (`Low`), to open-world assumption approaches, where due to the assumption that data may be incomplete, an answer to a user query can be either determined if there exist data that can prove such an answer, or "unknown" in the case where there is no data to determine its truthfulness (`High`). □

4.4 Optimization of data-intensive flows

Finally, we also discuss the low data latency as an important requirement for today's data-intensive flows.

Optimizing data flows has been one of the main topics in database research [84]. In this context, we discuss the following two aspects.

Optimization input. This dimension refers to the level at which the optimization is provided. It spans from optimizing the way that input `data` is stored and processed (e.g., data fragmentation) in order to achieve optimal execution (e.g., parallelizing data flow execution); then optimizing the execution of `single data flows` by means of modifying the execution of the data flow (e.g., operation reordering, different implementations) to achieve the optimal execution of a data flow; and finally the overall optimization of a `multi-flow`,

where the goal is to achieve the optimal execution for a set of data flows, rather than optimizing the execution of a single flow. □

Dynamicity. Another dimension for studying the optimization of data-intensive flows relates to the overhead introduced by the optimization and thus determines the level of dynamicity of the data flow optimization process. In the traditional DW systems, the design and optimization of ETL processes is done at the `design time`, once while the process is then executed periodically, many times. This obviously allows for a higher overhead of the optimization process and taking into account different metadata (e.g., statistics from the previous executions of the same flow). On the other hand, an ETO flow must be optimized at `runtime`, when the analytical query is issued, which introduces additional challenges into the optimization techniques, especially regarding the way the statistics are gathered and exploited for optimization. □

5 Data Extraction

In the initial stage of data-intensive flows, the source systems are identified and accessed for extracting the relevant data. In the data extraction stage, we focused on analyzing the three following challenges that characterize the shift towards the next generation BI settings, namely coupledness, accessibility, and structuredness, which are subsequently discussed in the following subsections.

5.1 Structuredness

The seminal work on DW systems (e.g., [82]), although mentioned external unstructured data as an important opportunity for building DW system, have not in particular tackled the challenges that they bring to the design of the ETL pipelines. Furthermore, some of the first (purely relational) approaches for the DW system design in the literature (i.e., using materialized views in [174]), as well as the later extraction techniques (e.g., [104, 109]) assumed purely relational data sources, thus only supported `High` structuredness of input data. [96] considers `Medium` stucturedness of input data, by providing the practical guidelines for accessing external data in Web logs or flat files, and semi-structured data (i.e., XML; see Figure 2.4(A)).

Example. In our running example, the ETL flow depicted in Figure 2.4 reads data from the transactional systems supporting daily sales operations. Notice that besides the challenges that heterogeneity in data sources (both structural and semantic [166]) brings to the design of an ETL pipeline, well-structured data sources do not require any special technique for obtaining the data before the transformation and cleaning starts. However, today, using a diversity of external and often unstructured data sources is becoming inevitable and thus the techniques for extracting such data have attracted the attention

of both academy and industry. In our example scenario, we can see that introducing unstructured data (e.g., free text `reviews` from the Web) into the analytical processes of the company (see Figures 2.2 and 2.3) required additional data processing for extracting relevant information from these sources. Specifically, `Sentiment Analysis` based on natural language processing (NLP) is performed over textual data from the Web to extract customer opinions about the `items` and the campaign. □

In research, different techniques (e.g., text mining [56], NLP [105, 57], sentiment analysis) are proposed to discover data patterns and extraction rules for extracting relevant data from natural language documents and transform unstructured data into more explicitly structured formats [30] (e.g., graphs, trees, relational model). There, approaches are hence able to deal with the `Low` structuredness of input data. However, at the same time, they may assume a considerable latency overhead to the execution of the complete data pipeline and thus introduce an additional challenge to data-intensive flows. Interestingly, the linked data movement [20], on the other side, proposes that large amounts of external data are already provided in more structured (semi-structured) formats and semantically interlinked (e.g., using RDF), in order to facilitate the situation-aware analysis [110] and data exploratory actions [9]. These approaches assume `Medium` structuredness of input data, ready for the analytical processes carried out by data-intensive flows.

5.2 Coupledness

First relational approaches for designing a DW by means of a set of materialized views (e.g., [174]) in general allowed very efficient refreshments processes, by applying the well-known view maintenance techniques, to either compute incremental changes in the sources or a complete "rematerialization" of a view. Both approaches issue queries (maintenance queries) over the data sources, extract the answer, and load the corresponding views. Such approaches, however, required a `High` coupledness to source systems and soon became unrealistic to support the demands of enterprises to include a variety of external data sources into their analytical processes.

Example. As we can see from our example scenario, the retail company initially relied mainly on the internal data sources (e.g., the information about the `sold items`), which are periodically transferred to a central DW (see Example 1.1). To lower the data transferred in every execution of the ETL flow, the designers have built the flow to only extract the `sales` and `item` data that are inserted to the sources after the last ETL execution (i.e., snapshot difference). For efficiently finding the differences in two snapshots of the source data, the tight (`High`) coupledness to the considered data sources is needed. On the other side, in the scenario illustrated in Examples 2.1 and 2.2 (i.e., Figures 2.2 and 2.3, respectively), some data sources (i.e., `item reviews` from the Web) are

not under the control of the company and moreover they may not be known in advance as their choice depends on the current user needs. Thus, obviously we cannot depend on having strong knowledge of these kinds of data sources. □

In the context of modern ETL processes, in [186], the author revisits the approaches for finding the difference of the consecutive snapshots of source databases (e.g., [104, 109]). However, these snapshot techniques (e.g., timestamps, triggers, interpreting the source's logs) still required certain control over the known source systems (i.e., `Medium` coupledness). Web-oriented systems have further imposed more relaxed environments for extracting data located on disparate Web locations. The most common solution introduced for performing data integration from disparate sources includes building specialized software components, called wrappers, for extracting data from different Web sources (e.g., [61]; see Figure 2.4(E)). Wrappers are typically used in combination with another component (namely mediator, see Figure 2.4(F)), which, based on a user query, invokes individual wrappers, and combines (i.e., integrates) data they return to answer the input query. However, the wrapper/mediator architecture still requires a `Medium` coupledness, as wrapper design highly relies on the specific technology of data source systems, while the changes in the source systems typically require reconsidering the wrappers' design. Finally, as we mentioned above, to make the enormously growing data volumes on the Web available and potentially useful for the enterprise analytical and data discovery actions, the linked and open data movement (e.g., [20]) has proposed a completely uncoupled environment (i.e., `Low` coupledness) with the general idea of having huge amounts of distributed data on the Web semantically interlinked and preferably provided in easily parseable formats (e.g., XML, RDF), see Figure 2.4(E). Approaches that argue for such `Low` coupled scenarios, envision architectures that can take the advantage of existing logic-based solutions for enabling data exploration actions over the external sources [9], and provide more context-aware data analysis [7].

A separate branch of realated research that strongly argues for `High` decoupling of data sources in data-intensive flows is Complex Event Processing (CEP) [37]. The idea here is on enabling on-the-fly processing and combining of data coming in greater speed and typically from external data sources, with the goal of detecting different correlations or anomalies happening in the "external world". Thus, the CEP systems typically decouples from the technical level information of the data sources (e.g., sensor readings), and rather aims at detecting events at the application level (e.g., correlations, anomalies). CEP is rather extensive and separate fields of research, and to this end, we here give its high level overview in terms of our analysis dimensions, while for the specific approaches and applications we refer the reader to the survey in [37], which compares and studies in detail the state of the art approaches in this field.

5.3 Accessability

The practical guidelines for efficiently building an ETL process in [96] proposes a pre-step of profiling data sources for quality, completeness, fitness, and accessibility. Apart from transactional data sources, dominant in traditional DW scenarios [82], with typically `High` accessability or at least predictable access policies (e.g., nightly time windows), nowadays a vast amount of potentially useful data for an enterprise is coming from remote data sources, over the global networks, like forums, social networks, and Web in general, [7], see Figure 2.4(E).

Example. Going back to our example scenario, we can notice that in the traditional DW environment, the company builds the system based on the previously elicited business needs and accordingly incorporates internal data sources (e.g., `item` and `sales`) into their analytical processes. ETL flows (e.g., see Figure 2.1) are designed and tested in advance for periodically extracting the data from pre-selected data sources, relying on the `High` or predictable availability of these sources. Conversely, the ETO flows in Figures 2.2 and 2.3 cannot rely on accessing the data sources at all times, due to remote access (i.e., Web) and moreover as they can be selected on-the-fly. □

Even though in the linked (open) data movement information about quality of external data are envisioned to be in the form of catalogs [20], the access to these data at any moment still cannot be guaranteed (`Low` accessability), which brings a new challenge to the process of data extraction in this scenario. In this context, the authors in [7] study the concept of situational data, which are usually external to an organization control and hence without a guaranteed access, but which in fact play an important role in today's context-aware decision making. The authors thus propose a "data as a service" solution, where envisioned systems will have a registry of possible data providers, and using Web service interface partially automate the process of finding the most appropriate and currently accessible data source.

5.4 Discussion

We summarize the results of studying the challenges of data extraction stage (i.e., the classification of the representative approaches) in Table 2.1. As expected, we have found more matured (i.e., $TRL \geqslant 2$) works dealing with this stage in the traditional BI setting (i.e., ETL), considering tighter coupledness to source systems, relying on high accessibility, and expecting structured data. Several approaches have opened the issue of dynamically accessing external and unstructured data, focusing mostly on data coming from the Web, while the majority considered structured (relational) or at most semi-structured (XML) data.

Data extraction is however an important stage to be reconsidered for to-

Table 2.1: Classification of data extraction approaches

Data extraction					
ETL vs. ETO	Approaches AUTHORS, YEAR, [NAME,] REF.	TRL	Dimensions		
			struct.	access.	coupl.
ETL	Inmon, 1992, [82]	1	High	High	High
	Theodoratos & Sellis, 1999, [174]	2			
	Labio et al. 1996, [104] Lindsay et al. 1987, [109]	3	High	N/A	Medium
	Kimball & Caserta, 2004, [96]	1	Medium	Medium	High
ETO	Feldman & Sanger, 2007, [56] Buneman et al., 1997, [30]	1	Low	N/A	N/A
	Laender et al., 2002, [105]	1	Low	Low	Medium
	Bizer et al., 2009, Linked Data, [20] Cugola and Margara, 2012, CEP, [37]	1	Medium	Low	Low
	Abelló et al., 2013, Fusion Cubes, [7] Abelló et al., 2015, [9]	1	Low	Low	Low
	G.-Molina et al., 1997, TSIMMIS, [61]	4	Medium	High	Medium

day's data-intensive flows, especially taking into account new loosely coupled BI environments [7]. The main challenges of these (mostly envisioned) ecosystems with low coupledness relate to the fact that data sources are outside of the organization control, and often not even known in advance. Thus, the efficient techniques to discover the relevant data must be deployed. We can benefit here from the known techniques proposed to explore the contents on the Web (e.g., Web crawling). Moreover, being external data sources, the systems become very sensitive to very probable variability of data formats, as well as the undefined semantics of data coming from these sources. To overcome the semantics heterogeneity gap between the data sources, and to automate discovering and extracting the data from them, we propose to use the semantic-aware exploratory mechanisms [9].

Furthermore, as we can see from our example scenario, specialized data processing (e.g., natural language processing, sentiment analysis, text mining) should be also considered to extract the relevant information from these, often unstructured, data sources. However, such complex data transformations typically affect the data latency in a data-intensive flow, hence in most of the current approaches this kind of input data transformation has been considered as part of a pre-processing step. An alternative to this, following the principles of linked (open) data [20], is to have data published in at least semi-structured formats (e.g., XML, CSV, RDF), which largely facilitates their further exploitation.

6 Data Transformation

After data are extracted from selected (often heterogeneous) sources, the flow continues with transforming the data for satisfying business requirements and considered quality standards. Data transformation is characterized as the main stage of a data-intensive flow by most of the approaches [96, 186]. The main dimensions we analyze in the data transformation stage are automation, malleability, and constraintness.

As previously mentioned, from early years, managing heterogeneous data has brought the attention of database community, and some fundamental works stem from the database theory field (i.e., data integration (e.g., [106, 182]) and data exchange (e.g., [55])) to tackle this problem from different perspectives.

Both data exchange and data integration problems are based on the concept of schema mappings, which in general can be seen as assertions that define the correspondences between source and target schema elements.

In general, a parallelism can be drawn between the theoretical problems of data exchange and data integration, and what we today consider as data-intensive flows. Similar observation has been discussed in [186]. The author compares data exchange to the traditional DW setting, where data transformations in the ETL pipeline can be generally seen as schema mappings of the data exchange setting. However, as also noted in [186], schema mappings, as defined by these theoretical approaches, are typically limited to simple transformations over data and do not efficiently support typical transformations of data-intensive flows (e.g., grouping, aggregation, or "black-box" operations), nor the diversity of data sources (i.e., only relational or XML data formats have been considered).

6.1 Malleability

Data-intensive flows, as other software artifacts, do not lend themselves nicely to evolution events, and in general, maintaining them manually is hard. The situation is even more critical in the next generation BI settings, where on-the-fly decision making requires faster and more efficient adapting to changed domain context, i.e., changed data sources or changed information needs.

For considering the former problem, we revisit the foundational works on data exchange and data integration, which introduced two schema mappings approaches, i.e., global-as-view (GAV) and local-as-view (LAV) [59, 106].

In the GAV approach, the elements of the global schema are characterized in terms of a query over the source schemata, which further enables less complex query answering by simply unfolding global queries in terms of the mapped data sources. However, GAV mappings lack flexibility in supporting the evolution of data source schemata, as any change on the sources may potentially invalidate

all the mapping assertions (i.e., `Low` malleability). An example of this approach is the wrapper/mediator system [61].

Example. As we discussed, in Scenario 1, the company elicits the business needs prior to designing the DW and ETL flows (e.g., see Figure 2.1). In the case a new data source is added, the redesign of the system is performed offline before the ETL flows are run again. However, notice that in the second scenario (see Examples 2.1 and 2.2) the flexibility of the system for adding new data sources must be supported in an "instant" manner, as business needs are provided on-the-fly and often require a prompt response. □

As opposed to GAV, LAV schema mappings characterize the elements of source schemata in terms of a query over the global schema. LAV mappings are intuitively used in the approaches where changes in dynamic data source schema are more common (e.g., [98]) as it provides `High` malleability of the data integration systems. We can thus observe that the LAV approach fits better the needs of the next generation BI setting, where the variability and number of data sources cannot be anticipated (e.g., [7, 9]). However, the higher flexibility of LAV mappings brings the issues of both, the complexity of answering the user queries and the completeness of the schema mappings. Generally, in LAV, answering user queries posed in terms of a global schema implies the same logic as answering queries using materialized views, which is largely discussed as a computationally complex task [75].

Several approaches further worked on generalizing the concept of schema mappings by supporting the expressive power of both LAV and GAV, i.e., both-as-view (BAV) [116], and global-and-local-as-view (GLAV) [59].

However, as we discussed before such approaches are hardly applicable to the complex data-intensive flows. In the context of the traditional DW systems, some works have studied the management of data-intensive flows (i.e., ETL process) in front of the changes of data source schemata. In [129] the authors propose a framework for impact prediction of schema changes for ETL workflow evolution. Upon the occurred change, the ETL flow is annotated with (pre-defined) actions that should be taken, and the user is notified in the case that the specific actions require user involvement. Other approaches (e.g., [89]) have dealt with automatically adapting ETL flows to the changes of user's information needs. For each new information requirement, the system searches for the way to adapt the existing design to additionally answer the new requirement, by finding the maximal overlapping in both data and transformation. Lastly, some approaches have also dealt with the problem of adapting DW systems to the changes of the target DW schema. Being a "non-volatile collection of data" [82], the evolution changes of the target DW schema are typically represented in terms of different versions of a DW (i.e., multiversion DW). In particular, the most important issue was providing a transparent querying mechanisms over different versions of DW schemata (i.e., cross-version querying; [118, 66]). For instance, a solution proposed in [66] suggested keeping

track of change actions to further enable answering the queries spanning the validity of different DW versions. These approaches provide a certain (`Medium`) level of malleability for data-intensive flows, but still lack the full automation of the evolution changes or applicability in the case of unpredictable complexity of data transformations.

In addition, after data warehousing was established as a de facto way to analyze historical data, the need for more timely data analysis has also emerged in order to support prompter detection of different anomalies coming from data. This led researches to rethink the current DW architecture and make it more malleable to combine both traditionally mid-term and long-term, with "just-in-time" analysis. This brought the idea of (near) real-time data warehousing systems. Several approaches discussed the main requirements of such systems and proposed architectural changes to traditional DW systems for satisfying these new requirements (e.g., [28, 188]). For instance, besides the main requirement of data freshness, [188] has also indicated minimal overhead of the source system and scalability in terms of input data sources, user queries, and data volumes, as relevant for these systems. They however pointed out the contradiction between users need for maximal data freshness and completeness, and the high overhead of the traditional DW workflows that often require costly data cleaning and transformations. To this end, the approaches in [28] and [188] discuss both conceptual and technological changes that would balance the delays in traditional ETL processes. In practice, SAP Business Warehouse [117] is an example of such system. It provides certain flexibility to traditional DW systems for enabling on-the-fly analysis at different levels of data, i.e., summarized and loaded to a DW, consolidated operational data (operational data store), or even directly over the transactional data. Their goal is to enable more real-time data warehousing and a possibility of also including fresh, up-to-date transactional data to the analysis. Even though the (near) real-time DW approaches bring more malleability (`Medium`) to data analysis by combining historical and on-the-fly analysis, included data are still typically coming from the in-house and predefined data sources.

6.2 Constraintness

What further distinguishes data exchange [55, 99] from the original data integration setting, is that the target schema additionally entails a set of constraints that must be satisfied (together with schema mappings) when creating a target instance (i.e., `High` constraintness).

Example. In Figure 2.1, we can notice that for loading data into a DW, data-intensive flow must ensure a certain level of data quality to satisfy constraints entailed by the DW (e.g., `Remove Duplicates` in Figure 2.1 removes the repetitive `itemIDs` for loading the `successFact` table into a DW). On the other side, data-intensive flows in the next generation BI settings (see Figures

2.2 and 2.3), due to their time constraints typically cannot afford to ensure full data quality standards, but is often sufficient to deliver partially cleaned (i.e., "right") data, at the right time to an end-user [50]. □

The work on generalizing schema mappings (GLAV) in [59] also discusses the importance of adding support for defining the constraints on global schema, but no concrete solution has been provided. In the data integration setting, although some works did study query answering in the presence of integrity constraints on global schema [32], (i.e., `Medium` constraintness), most of the prominent data integration systems (e.g., [61, 98]) typically do not assume any constraints in the target schema (i.e., `Low` constraintness). Furthermore, as we discussed in the DW context, the design of an ETL process is affected by the integrity constraints typical in a dimensionally modeled DW schema (see Figure 2.4(C)).

When working with data from unstructured data sources, one may face two different problems: (1) how to extract useful information from data in unstructured formats and create more structured representation; and (2) how to deal with incomplete and erroneous data occurred due to lack of strict constraints in source data models. The latter problem becomes even more challenging when the target data stores entail strict constraints as we discussed above. While the former problem is usually handled by means of data extraction techniques discussed in the Section 5, the latter is solved at the data transformation stage, where data are cleaned to fulfill different quality standards and target constraints. As expected, such a problem has brought the attention of researchers in the data exchange (e.g., [53]) and data integration (e.g., [47]) fields. In the modern DW systems, target data quality and `High` constraintness is usually guaranteed as the result of the process called data cleaning. Data cleaning deals with different data quality problems detected in sources, e.g., lack of integrity constraints at sources, naming and structural conflicts, duplicates [137].

6.3 Automation

It is not hard to see from the previously discussed problem of data cleaning and the flows in the example scenario, that today's data-intensive flows require more sophisticated data transformations than the ones (mainly based on logics) assumed by fundamental approaches of data exchange and data integration. At the same time, higher automation of the data flow design is also required to provide interactive, on-the-fly, analysis.

Example. Loading a DW may require complex data cleaning operations to ensure the entailed constraints. Obviously, complete automation of the design of such data-intensive flows is not realistic and thus the designers in Scenario 1 usually rely on a set of frequent data transformations when building ETL flows (e.g., `Join`, `UDF`, and `Remove Duplicates` in Figure 2.1). But, in Scenario 2, such an assisted design process is not sufficient, as the flows for

answering users' on-the-fly queries (e.g., see Examples 2.1 and 2.2) must be created instantaneously. This, together with the requirement for lower data latency, restricted such flows to more lightweight operations (e.g., `Filter` or `Match` in Figure 2.2). □

Different design tools are available in the market and provide often overlapping functionalities for the design and execution of data-intensive flows (mostly ETL; see for example Figure 2.4(B)). The complexity and variability of data transformations has introduced an additional challenge to the efforts for providing a commonly accepted modeling notation for these data flows. Several works have proposed different ETL modeling approaches, either ad-hoc [190], or based on well-known modeling languages, e.g., UML in [180] or BPMN in [12, 195]. However, these modeling approaches do not provide any automatable means for the design of an ETL process (i.e., `Low` automation). Some approaches (e.g., from UML [120], or from BPMN [13]) are further extended to support certain (i.e., `Medium`) automation of generating an executable code from the conceptual flow design, by means of model transformations (i.e., Model-driven design).

The design of an ETL process is on the other side described as the most demanding part of a DW project. As reported in [170] ETL design can take up to 80% of time of the entire DW project. In [96] the authors give some practical guidelines for a successful design and deployment of an ETL process, but without any automatable means (i.e., `Low` automation), still, a considerable manual effort is expected from a DW designer. In [189], the framework that uses the ad-hoc modeling notation from [190] is proposed to assist the ETL design, along with the palette of frequently used ETL patterns (i.e., `Medium`).

Several approaches went further with automating the conceptual design of ETL processes. On the one hand, in [167], the authors introduced the design approach based on Semantic Web technologies to represent the DW domain (i.e., source and target data stores), showing that this would further enable automation of the design process by benefiting from the automatic reasoning capabilities of an ontology. [146], on the other hand, assumes that only data sources are captured by means of a domain ontology with associated source mappings. Both DW and ETL conceptual designs are then generated to satisfy information requirements posed in the domain vocabulary (i.e., ontology). Finally, [18] entirely rely on an ontology, both for describing source and target data stores, and corresponding mappings among them. Integration processes (ETL) are then also derived at the ontological level based on the type of mappings between source and target concepts (e.g., equality, containment). However, even though these approaches enable `High` automation of the data flow design, they work on a limited set of frequent ETL operations.

In parallel, in the field of data exchange, [52] proposes a tool (a.k.a. Clio) that automatically generates correspondences (i.e., schema mappings) among schemas without making any initial assumptions about the relationships between them, nor how these schemas were created. Such a generic approach

thus supports `High` automation in creating different schema mappings for both data integration and data exchange settings. [44] went further to provide the interoperability between tools for creating declarative schema mappings (e.g., Clio) and procedural data-intensive tools (e.g., ETL). Still, such schema mappings either cannot tackle grouping and aggregation or overlook complex transformations typical in today's ETL processes. The next generation BI settings, however, cannot always rely on the manual or partially automated data flow design. Moreover, unlike ETL, ETO cannot completely anticipate end user needs in advance and thus besides the `High` level of automation, the design process must also be agile to efficiently react in front of new or changed business needs (e.g., [143]).

Table 2.2: Classification of data transformation approaches

Data transformation					
ETL vs. ETO	Approaches AUTHORS, YEAR, [NAME,] REF.	TRL	Dimensions		
			autom.	malleab.	constr.
ETL	Fagin et al., 2003, [55] Kolaitis, 2005, [99]	2	N/A	N/A	High
	Rahm & Hai Do, 2000, [137]	1			
	Kimball & Caserta, 2004, [96]	1	Low	Low	High
	Vassiliadis et al., 2002, [190] Trujillo & L.-Mora, 2003, [180] Wilkinson et al., 2010, xLM, [195] El Akk. et al., 2012, BPMN-ETL [12]	2	Low	Low	High
	Muñoz et al., 2009, [120] El Akkaoui et al., 2013, [13] Vass. et al., 2003, ARKTOS II, [189] Papastefanatos et al., 2009, [129] Morzy & Wrembel, 2004, [118] Golfarelli et al., 2006, [66]	3	Medium	Medium	High
	Skoutas & Simitsis, 2007, [167] Bellatreche et al., 2013, [18]	3	High	Low	High
	Fagin et al., 2009, Clio, [52]	4			
ETL & ETO	McDon. et al., 2002, SAP BW, [117]	4	Medium	Medium	High
	Romero et al., 2011, GEM, [146] Dessloch et al., 2008, Orchid, [44] Jovanovic et al., 2012, CoAl, [89]	3	High	Medium	High
ETO	G.-Molina et al., 1997, TSIMMIS, [61]	4	High	Medium	Low
	Kirk et al., 1995, Inf. Manifold, [98]	3	High	High	Low
	Romero & Abelló, 2014, [143] Abelló et al., 2014, [9]	1			
	McBrien & Pou., 2003, BAV, [116] Friedman et al., 1999, GLAV, [59] Calì et al., 2004, [31]	2	N/A	High	Medium

6.4 Discussion

As we have seen, in the next generation BI setting (i.e., ETO), where data sources are often external to the organization control and moreover discovered

dynamically based on current user needs; a more flexible environment is needed for efficiently supporting adding new data sources to analytical processes. The local-as-view (LAV) schema mapping approaches are more suitable in order to support the required level of malleability [116, 59]. In the LAV approach, plugging new data sources requires defining a mapping of the new source schemata to the global schema, without affecting the existing mappings. However, as we can see in Table 2.2, currently, most of these techniques are still purely theoretical (i.e., $TRL = 2$), while the high complexity and intractability of LAV approaches have been widely discussed, and hence represent a serious drawback for using LAV mappings in near real-time BI environments.

On the other side, some approaches (e.g., [52]) have worked on automating the creation of such schema mappings, which can be widely applicable for supporting answering information requirements on-the-fly. Even though we notice the lower requirement for the cleanness and constraintness of output data in the next generation BI setting (see Table 2.2), which would typically result with lower complexity of a data flow, today's BI applications do require rather complex data analytics, which are typically not supported in the schema mapping approaches. Some approaches try to extend this by automating the flow design (e.g., [146, 18]), but still with very limited and predefined operation sets (i.e., ETL & ETO). Therefore, automating the creation of more complex data-intensive flows (e.g., machine learning algorithms), by means of exploiting different input data characteristics or using metadata mechanisms is still lacking.

7 Data Delivery

After data from multiple sources are cleaned, conformed, and combined together, a data-intensive flow delivers the data in the format suitable to the user needs either for visualization, or further analysis and querying. In the data delivery stage, we focus on analyzing the two following dimensions, namely, interactivity and openness, subsequently discussed in the following sections.

7.1 Interactivity

One of the important decisions that should be made while building data-intensive flows is the interactivity of the system when delivering data at the output.

Example. Notice that the ETL flow in Figure 2.1 is designed to periodically transfer the complete data about the `item sales` in a batched back-end process, so that users may later analyze the subset of these data depending on their needs (e.g., slicing it only to the `sales` in the third quarter of the last). ETO flows in Figures 2.2 and 2.3, however, instantly deliver the data from

the sources that are currently asked by the user (e.g., trends of the past weekend, and trending product `items` from the first week of March, respectively). Moreover, such ETO flows are typical examples of answering ad-hoc and one-time analytical queries, thus storing their results is usually not considered as beneficial. □

Going back again to the fundamental work on data exchange, the data delivery in this setting is based on computing a solution, i.e., a complete instance of the target schema that satisfies both, schema mappings and constraints of the target schema. The queries are then evaluated over this solution to deliver the answer to the user. However, due to incompleteness of data and/or schema mappings, there may be more than one, and theoretically an infinite number of valid solutions to the data exchange problem [55]. Answering user queries in such a case would result in evaluating a query over all possible solutions (i.e., finding the certain answer). To overcome the obvious intractability of query answering in data exchange, a special class of solutions (universal solutions), having a homomorphism into any other possible solution, is proposed.

Like in the data exchange setting, materializing the complete data at the target for the purpose of later answering user queries without accessing the original data sources, is also considered in the later approaches for designing a data warehouse (see Figure 2.4(G)), i.e., `Low` interactivity. As we discussed in Section 5, a DW has been initially viewed as a set of materialized views (e.g., [174]). Similarly, in this case the database specific techniques (e.g., incremental view maintenance) are studied to minimize the size of the data materialized in each run of refreshment flows (maintenance queries). However, as DW environments have become more demanding both considering the heterogeneity and volume of data, it became unrealistic to consider a DW solely as a set of materialized views. In addition, many approaches have further studied the modeling and the design of a target DW schema, which should be able to support analytical needs of end users. This has resulted in the field of multidimensional (MD) modeling that is based on fact/dimension dichotomy. These works belong to a broader field of MD modeling and design that is orthogonal to the scope of this chapter, and thus we refer readers to the survey of MD modeling in [140] and the overview of the current design approaches covered by Chapter 6 in [69].

Conversely, the data integration setting, as discussed in Section 6, does not assume materializing a complete instance of data at the target, but rather interactively answering individual user queries (e.g., through a data cube or a report; see Figure 2.4(G)) posed in terms of a global (virtual) schema (i.e., `High` interactivity). A query is reformulated at runtime into queries over source schemata, using schema mappings (e.g., [61, 98, 106, 75, 182]). Another examples of `High` interactivity are Complex Event Processing and Data Stream Processing systems. Besides the differences these systems have (see their comparison in [37]), the common property of these systems is that they provide on-the-fly delivery of data, with typically low latency, and for the one-time use

only (e.g., monitoring stocks, fraud detection), without a need to materialize such data.

In [63], in the context of peer data management, a hybrid solution is proposed based on both data exchange and data integration. The `Medium` interactivity, by partially materializing data and using a virtual view over the sources (peers), has been proposed. To this end, the solution presents schema dependencies that can be used both for computing the core and query answering.

The "right" (`Medium`) level of interactivity is also discussed to be crucial in the next generation BI setting (e.g., [40]) where a partial materialization is envisioned for a subset of data with low latency and low freshness requirements (i.e., for which we can rely on the last batched ETL run). Following the similar idea, [134] proposes a framework for combining data-intensive flows with user query pipelines and hence choosing the optimal materialization point (i.e., `Medium` interactivity) in the data flow, based on different cost metrics (e.g., source update rates and view maintenance costs). Another field of research also follows the `Medium` level of interactivity, and proposes an alternative to traditional ETL processes, where row data are first loaded to the target storage, and later, typically on-demand, transformed and delivered to end-users, i.e., extract-load-transform (ELT). For instance, an example ELT approach in [193] proposes an ELT architecture that deploys traditional database mechanisms (i.e., hierarchical materialized views) for enabling on-demand processing of fresher row data previously bulk loaded into a DW.

7.2 Openness

As we have seen, incompleteness in source data (especially in the today's Web oriented environments, see Scenario 2 in Section 3), brings several challenges to data-intensive flows. In addition, when integrating and delivering the data at the target, due to possibly incomplete (or non-finite) data, the choice between two main assumptions should be made, i.e., closed world assumption (CWA) or open world assumption (OWA). This choice depends on different characteristics of both, the considered data sources and the expected target. For the systems, like in-house databases or traditional data warehouse systems, where the completeness of data can be assumed, CWA is preferable since in general we do not anticipate discovering additional data (i.e., `Low` openness). On the other side, when we assume incomplete data at the sources of analysis, we can either follow CWA and create a single finite answer from incomplete data (e.g., by means of data cleaning process), or OWA which would in general allow multiple and possibly an infinite number of interpretations of the answer at the target, by considering also dynamically added data to the analysis [9] (i.e., `High` openness).

Example. In our example scenarios, in the traditional BI setting (Scenario 1), the analysis of the revenue share depends solely on the data about the `item`

`sales`, currently transferred to DW by means of the ETL process depicted in Figure 2.1. On the other hand, the next generation BI setting in Scenario 2, should assume a more open environment, where at each moment depending on the end user needs (e.g., following trends and opinions about `items` as in Examples 2.1 and 2.2) the system must be able to dynamically discover the sources from which such an information can be extracted (e.g., `reviews` in forums). □

Similarly, [130] revisits two main paradigms for the Semantic Web: (1) Datalog, that follows the CWA, and (2) Classical (standard logics) paradigm that follows OWA. An important conclusion of this work is that the Datalog paradigm as well as CWA is more suitable for highly structured environments in which we can ensure completeness of the data, while the Classical paradigm and OWA provide more advantages in loosely coupled environments, where the analysis should not only be limited to the existing (i.e., "in-house") data.

Moreover, coinciding arguments are found in the fields of data integration and data exchange. Intuitively, CWA (i.e., `Low` openness) is more suitable assumption in data exchange, where query answering must rely solely on data transferred from source to target using defined schema mappings and not on the data that can be added later [108]. Conversely, more open scenario is typically expected in data integration systems [46], where additional data can be explored on-the-fly and added to the analysis [9] (i.e., `High` openness). However, the high complexity of query answering under the OWA [130], raises an additional challenge to the latency of data-intensive flows, which is critical in next generation BI systems.

Table 2.3: Classification of data delivery appraoches

Data delivery				
ETL vs. ETO	Approaches AUTHORS, YEAR, [NAME,] REFERENCE	TRL	Dimensions	
			interac.	open.
ETL	Golfarelli & Rizzi, 2009, [69] Fagin et al., 2005, Data Exchange, [55] Libkin, 2006, [108] Theodoratos & Sellis, 1999, [174]	2	Low	Low
ETL & ETO	Dayal et al., 2009, [41]	1	Medium	Low
	Qu & Dessloch, 2014, [134] Waas et al., 2013, ELT, [193]	3		
	Giacomo et al., 2007, [63]	2	Medium	Medium
ETO	Cugola & Margara, CEP, 2012, [37]	1	High	Medium
	Abelló et al., 2013, Fusion Cubes, [7] Abelló et al., 2015, [9]	1	High	High
	Lenzerini, 2002, Data Integration, [106] Halevy, 2001, [75] Ullman, 1997, [182] Doan et al., 2012, Data Integration [46]	2		
	Garcia-Molina et al., 1997, TSIMMIS, [61] Kirk et al., 1995, Information Manifold, [98]	3		

7.3 Discussion

The outcome of studying the data delivery stage of data-intensive flows can be seen in Table 2.3. We observed that the same principles for data delivery (i.e., levels of the studied dimensions in Table 2.3) are followed in approaches of traditional data exchange [55] and DW settings [69] (i.e., ETL). At the same time, we also noticed that the similar principles of the data integration setting [106, 46] are envisioned for next generation BI systems in some of the studied approachs [7, 9] (i.e., ETO), while others propose a mixed approach [41] (i.e., ETL & ETO).

Such observations strongly indicated that the underpinnings for building a system for managing data-intensive flows in the next generation BI setting should be searched in the theoretical field of data integration.

Such a trend has been indeed followed in some of the recent approaches. For example, the idea of creating a unified view over relevant data sources (i.e., the main principle of the data integration setting), is revisited by some approaches by creating a common domain vocabulary and integrating it with existing data sources (e.g., [167, 146]). There, the use of a domain ontology to reconcile the languages of business and IT worlds when building a BI system has been proposed. In [146], an ontology is used in combination with schema mappings to automatically generate ETL pipelines to satisfy information requirements previously expressed in terms of an ontology by an end user. The approach works with a predefined set of data sources which is, as suggested, suitable for building a DW system, but as data in today's BI settings are coming from disparate and external data sources, the challenge of capturing their semantics under a common vocabulary brings additional challenges. To this end, Semantic Web technologies are discussed (e.g., [9]) as a solution both for capturing the semantics and further interactive exploration of data, facilitated by the automatic reasoning mechanisms.

8 Optimization of data-intensive flows

Optimizing the execution of data-intensive flows, is a necessity, especially taking into account the next generation BI setting that often requires the "right-time" delivery of information.

8.1 Optimization input

The problem of data flow optimization has been considered from early years of databases from different perspectives, where each of these perspectives may affect different parts of a data flow.

- `Data.` The optimization of a data flow execution can be achieved by

transforming the structure of the original dataset (e.g., by means of data partitioning [95]). However, notice that simply transforming a dataset would not achieve a better performance, unless the execution model of a data flow is able to exploit such a transformation (e.g., distributed or parallel data processing [43]).

- `Data flow.` The most typical case considers the optimization of data flow execution by changing the way data are processed, while ensuring the equivalent semantics of the resulting dataset. Such techniques stem from the early years of databases, where minimizing data delivery time by changing the order and selecting the most optimal algorithm for operations applied over input data has been studied under the name of query optimization [84]. Moving to the DW environment, which assumes more complex data transformations than the ones in relational algebra, has opened a new field of study dealing with optimizing ETL processes (e.g., [158]). In fact, similar principles to those introduced in query optimization (i.e., generating semantically equivalent execution plans for a query by reordering operations, and then finding a plan with a minimal cost) have been applied in [158] and extended to the context of ETL flows.

  Another work [78, 138] has based operation reordering (i.e., plan rewrites) on automatically discovering a set of extensible operation properties rather than relying solely on algebraic specifications, in order to enable reordering of complex ("black-box") operators. While low data latency is desirable for ETL processes, due to limited time windows dedicated to the DW refreshment processes, in the next generation BI setting, having data-intensive flows with close to zero latency is a must. Other techniques include: choosing the optimal implementation for the flow operations [181], selecting the optimal execution engine for executing a data flow [162, 102], data flow fragmentation and pipelining [95, 164].

- `Multi-flow.` In other scenarios, especially in the case of shared execution resources, the optimization goal may suggest optimizing the overall execution of a set of data flows, rather than only the execution of an individual flow. Approaches that deal with this problem fall in two categories. On the one hand, some approaches assume having a detailed knowledge of included data flows and thus try to exploit it and optimize the overall execution, by means of finding shared parts of data workloads and reusing common execution and data [150, 64, 89]. Other approaches however assume only a high level knowledge of included data flows (e.g., input data size, execution time, high-level flow complexity, time constraints for the flow execution; [133]). In such cases, the optimization of data flows proceeds by selecting the best scheduling for the execution of data-intensive flows, while the further optimization of individual data flows is left to an

engine-specific optimizer [164].

8.2 Dynamicity

While the challenges due to the higher complexity of data transformation has been largely addressed [158, 78], proposed cost-based techniques often require certain statistics metadata available for a given data flow in order to find the optimal configuration. However, this is typically not the case and gathering and managing such statistics is not an easy task [74]. [74] proposes a statistics collection framework, by defining a set of necessary statistics, as well as gathering methods. However, this approach although powerful assumes a case of ETL process flow, where data flows are typically designed and optimized in advance (i.e., at `design time`), while the statistics gathering depends on the previous execution of the same ETL process.

Notice that the majority of the optimization approaches discussed in the previous subsection also assume a static case (see Table 2.4), where data flows are optimized once, at `design time`, and then executed many times. The exception to this are approaches that besides statically optimizing a data flow, also provide dynamic optimization of data flow executions in terms of `runtime` scheduling (i.e., [164, 133, 95]).

Some optimization approaches however propose on-the-fly gathering of statistics, more suitable for the next generation data flow setting, and applying data flow optimization steps at `runtime`. The approach in [42] proposes performing micro-benchmarks for building models to estimate the costs of operations using different implementations or executing them on different engines. They show how to deal both with the conventional (relational algebra) operators as well as with complex data transformations typical for the next generation data flows (e.g., sentiment or text analysis). The importance of using more accurate statistics for optimizing data flows in dynamic, cloud-scale environments has been also discussed in [29]. To deal with uncertainty when optimizing running data flows they propose an approach that continuously monitors the execution of data flows at `runtime`, gathers statistics, and re-optimizes data flows on-the-fly to achieve better performance. The focus here is however on the distributed computational model, where execution times are often higher than in the centralized systems due to necessary synchronization costs, thus the re-optimization overheads are typically considered as negligible.

8.3 Discussion

In Table 2.4, we summarize the outcome of studying data flow optimization approaches in this section. It is easy to see that a static (design time) optimization of data flows has been largely studied in academia. While most approaches worked on the problem of extending traditional query optimization

Table 2.4: Classification of data flow optimization approaches

Data flow optimization				
ETL vs. ETO	Approaches AUTHORS, YEAR, [NAME,] REFERENCE	TRL	Dimensions	
			input	dynamicity
ETL	Simitsis et al., 2005, [158] Hueske et al., 2012, [78] RheinInder et al., 2015, SOFA, [138] Tziovara et al., 2007, [181] Simitsis et al., 2005, [162] Kougka et al., 2015, [102] Halasipuram et al., 2014, [74]	3	Data flow	Design time
	Giannikis et al., 2014, SharedDB, [64] Jovanovic et al., 2016, CoAl, [89]	3	Multi-flow	Design time
ETO	Karagiannis et al., 2013, [95]	3	Data & Data flow	Runtime
	Dayal et al., 2011, [42] Bruno et al., 2013, [29]	3	Data flow	Runtime
	Jarke & Koch, 1984, [84]	2		
	Simitsis et al., 2013, HFMS, [164]	2	Multi-flow	Runtime
	Polo et al., 2014, [133]	3		

techniques [84] to support more complex data flow operations [158, 78], they typically overlook the importance of having the needed statistics of input data and data flow operations to perform cost-based data flow optimization. Such design time optimization approaches require higher overhear and are hence mostly applicable to the traditional BI settings (i.e., ETL). Some of the recent approaches insist on the importance of having accurate statistics for creating an optimal execution of a data flow, both for design time [74] and runtime scenarios [29]. Still, the challenges for efficiently gathering and exploiting such statistics metadata for optimizing data-intensive flows remain due to the required close to zero overhead of an optimization process and the "right-time" data delivery demands in the next generation BI settings (i.e., ETO). To this end, the existing algorithms proposed for efficiently capturing the approximate summaries out of massive data streams [107], should be reconsidered here and adopted for gathering approximate statistics for data-intensive flows over large input datasets.

9 Overall Discussion

Finally, in this section, we summarize the main observations made from the results of our study and propose the high level architecture for managing the lifecycle of data-intensive flows in the next generation BI setting. We also give further directions for the topics that require special attention of the research community when studying data-intensive flows.

We have observed some general trends in studying the fields related to data-

intensive flows. We focused on the fields of data exchange, data integration, as well as ETL, and ETO. The need for managing heterogeneous data has appeared ever since the database systems start being more broadly used (e.g., federated databases in the 80's [76]). Besides, even though the system in [155] from the 70's is argued to be the first approach that followed the principles of data exchange, the data exchange problem has not been formally defined until the early 00's [54]. Likewise, the problem of data integration is studied from the 90's [61, 182], while the strong theoretical overview of the field is given in the early 00's [106]. Along with these theoretical works, the concept of the traditional DW setting was defined by Bill Inmon in the early 90's [82]. ETL, as a separate and rather complex process, however, appeared in the late 90's and the early 00's to replace simple refreshment processes for a DW modeled as a set of materialized views [186]. We can, however, notice the disparity among the trends of studying these fields in the past, showing that they have focused on solving isolated issues.

In the recent years, business environments became more complex, dynamic and interconnected, hence more interactive analytic systems to support daily decision making upon the combination of various (external or internal) data sources, have become a necessity. Moreover, as discussed throughout this chapter, today's BI environments require efficiently combining these individual solutions for the problem at hand. To this end, in this chapter, we have given a unified view of data-intensive flows, focusing on the challenges that next generation BI setting has brought. Currently, even though many works under different names (i.e., from different perspectives) have envisioned and/or proposed conceptual frameworks for next generation BI ecosystems (e.g., [7, 19, 40, 42, 50, 110]), we still lack an end-to-end solution for managing the complete lifecycle of data-intensive flows. Going back to Tables 2.1, 2.2, and 2.3, we can observe a certain overlapping of levels of different dimensions between the theoretical problem of data exchange and data warehousing approaches (i.e., ETL), as well as between data integration and data-intensive flows in the next generation BI setting (i.e., ETO).

After drawing a parallelism between the principles of data-intensive flows and fields of data integration and data exchange, we discuss how the knowledge of the studied fields can be applied for building data-intensive flows in the next generation BI setting.

9.1 Architecture for managing the lifecycle of data-intensive flows in next generation BI systems

We additionally observed in Tables 2.1 - 2.3 that the majority of works supporting the idea of the next generation BI setting in fact belong to level 1 of technical readiness ($TRL = 1$), as they are mostly visionary works that analyze the challenges of the next generation BI setting from different perspectives.

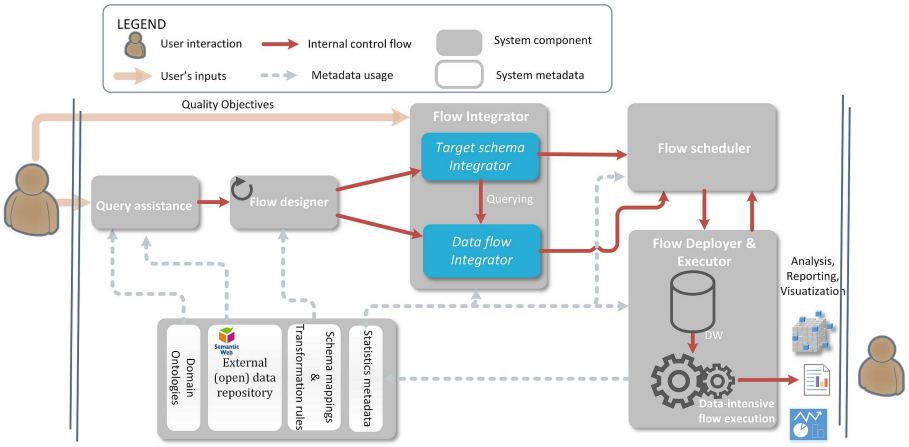


Fig. 2.6: Architecture for managing the lifecycle of data-intensive flows

However, we still lack a complete picture of all the aspects of data-intensive flows in this new setting, and to this end, we envision here an architecture of a system for managing data-intensive flows (Figure 2.6).

The proposed architecture depicts at high level the main outcome of our study. It points out the main processing steps which need to be considered during the lifecycle of a data-intensive flow. Moreover, the architecture captures in a holistic way the complete lifecycle of data-intensive flows, and as such, it can be seen as a roadmap for both academia and industry toward building data-intensive flows in next generation BI systems. In what follows, we discuss in more detail different architectural modules, and if available, we point out example approaches that tackle the challenges of such modules.

Going from left to right in Figure 2.6, we observe several modules covering different parts of the data-intensive flow lifecycle.

We start with the Query Assistance module that should provide an intuitive interface to end users when expressing their information requirements. On the one hand, it should raise the usability of the BI system, for a broader set of business users. This module should provide a business-oriented view over the included data sources (e.g., by means of domain ontologies like in [146, 90, 18]). On the other hand, the Query Assistance module should also facilitate the low coupledness of data sources and be able to efficiently connect to a plethora of external data source repositories. These, preferably semantically enriched data sources (e.g., linked (open) data; [20]), should supplement user analysis with context-aware data and thus raise the openness of the delivered results (see Section 7).

Previously expressed information requirements further need to be automat-

ically translated in an appropriate data flow (i.e., the Flow Designer module) that will satisfy such information requirements. Flow Designer must provide robustness for such loosely coupled systems in dealing with data sources with low (non-guaranteed) accessibility. Furthermore, to support the high automation of the Flow Designer module, the system should first revisit the existing approaches for automatic schema mapping creation (e.g., [52]). Obviously, these approaches must be extended with more complex data transformations. First, supporting low structuredness and extracting useful data from unstructured data sources on-the-fly should be largely supported, as dynamic systems cannot always rely on having a batched preprocessing step doing so. Furthermore, more complex data analysis (e.g., machine learning algorithms) should be supported in these data flows. Here, we can benefit from exploiting different data and flow characteristics (e.g., by revisiting the previously studied field of intelligent assistants for data analysis [153]). Lastly, the Flow Designer module should automatically accommodate the flow to ensure the required level of data quality and output data constraintness (typically in contradiction with required data latency) [177]. Importantly, such a design process must be iterative to support high malleability of the data flow design in front of new, changed, or removed data sources or information requirements.

In the case of partially materializing data, as suggested in [40], the Flow Designer module should be aware or be able to reconstruct the target schema, where data are loaded in a previously executed batch process, and further queried when interactively answering information requirements. Thus, the final data flow ready for deployment must be integrated from the combination of data flows that directly access data sources and querying previously materialized target data (i.e., the Flow Integrator module). Notice that finding the optimal level of partial materialization is still a challenge and must be decided using previously collected flow statistics and following desired quality objectives.

Next, the optimization of data-intensive flows should be efficiently supported at different levels of the flow lifecyle (see Section 8. Initially, when the integrated data flow is created to answer a user's requirement at hand, optimization of a data flow should be done in combination with selecting the optimal partial materialization of data [134]. Furthermore, having multiple data-intensive flows answering different requirements of end-users waiting for execution, the system requires an optimal schedule for running these data flows over the shared computational resources (e.g., shared, multi-tenant cluster), i.e., Flow Scheduler module. Lastly, the automatic optimization means must be also provided when deploying data flows, for selecting an optimal execution engine (e.g., [162, 102]), as well as for providing the lower level, engine-specific, optimization of a data flow (i.e., the Flow Deployer module).

From Figure 2.6 we can observe that the automation of the design and optimization of data-intensive flows, as well as the query assistance, must be largely facilitated by means of different metadata artifacts (i.e., schema map-

pings, domain ontology, flow and statistics). Indeed, the use of metadata for automating the design of the next generation data warehouse systems (DW 2.0) has been previously discussed [83], while recently the main challenges of matadata in the analytical process of the next generation BI systems have been studied in [184].

Finally, as an important remark, we want to draw a parallelism of the envisioned architecture depicted in Figure 2.6, and the traditional architecture of centralized database management systems (DBMS). First, using declarative (SQL) queries in a DBMS, end users pose their analytical needs to the system. While based on the traditional database theory, the semantic optimizer is responsible for transforming a user query into an equivalent one with a lower cost, in next generation BI systems, user queries need to be additionally transformed and enriched to access external data by means of data exploration processes (i.e., Query Assistance; [9]). Furthermore, similarly to the syntactic optimizer in the traditional DBMSs, Flow Designer needs to translate an information requirement to a sequence of operations (i.e., syntactic tree), which represents a logical plan of a data-intensive flow execution. The execution plan should be typically optimized for an individual execution. However, in next generation BI systems, a data-intensive flow could also be integrated with other data flows for an optimized multi-flow execution (i.e., Flow Integrator). This conceptually resembles the well-known problem of multi-query optimization [150], but inevitably brings new challenges considering the complexity of data flow operations, which cannot always be presented using algebraic specifications [78]. Moreover, the Flow Integrator module should also transform input execution plan and optimize it considering partial materialization of data, similarly to the query rewriting techniques for answering queries using materialized views [75]. Following the traditional DBMS architecture, an integrated execution plan is then optimally scheduled for execution, together with the rest of the data flows in the system (i.e., Flow Scheduler). Lastly, the logical data flow is translated into the code of a selected execution engine (e.g., [93, 102]), physically optimized considering available access structures, and finally deployed for execution (i.e., Flow Deployer & Executor). Similarly to the concept of the database catalog, throughout the lifecycle of a data-intensive flow, different metadata artifacts need to be available (e.g., schema mappings, transformation rules, statistics metadata; see Figure 2.6) to lead the automatic design and optimization of a data flow.

The parallelism drawn above finally confirms us that the underpinnings of data-intensive flows in next generation BI systems should be analyzed in the frame of the traditional DB theory field. Nevertheless, as we showed through our study, the inherent complexity of today's business environments (e.g., data heterogeneity, high complexity of data processing) must be additionally addressed, and comprehensively tackled to provide end-to-end solutions for managing the complete lifecycle of data-intensive flows.

10 Conclusions

In this chapter, we studied data-intensive flows, focusing on the challenges of the next-generation BI setting. We analyzed the foundational work of database theory tackling heterogeneity and interoperability (i.e., data exchange and data integration), as well as the recent approaches both in the context of DW and next generation BI systems.

We first identified the main characteristics of data-intensive flows, which built the dimensions of our study setting, and further studied the current approaches in the frame of these dimensions and determined the level the studied approaches attain in each of them.

As the main outcome of this study, we outline an architecture for managing the complexity of data-intensive flows in the next generation BI setting. We discuss in particular different components that such an architecture should realize, as well as the processes that the data-intensive flow lifecycle should carry out.

Finally, within the components of the envisioned architecture, we point out the main remaining challenges that the next generation BI setting brings to managing data-intensive flows, and which require special attention from both academia and industry.

Acknowledgements. This work has been partially supported by the Secreteria d'Universitats i Recerca de la Generalitat de Catalunya under 2014 SGR 1534, and by the Spanish Ministry of Education grant FPU12/04915.

Chapter 3

Incremental Consolidation of Data-Intensive Multi-flows

The paper has been published in the
IEEE Transactions on Knowledge and Data Engineering, 28(5): pp. 1203-1216 (2016). The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1109/TKDE.2016.2515609

Abstract

Business intelligence (BI) systems depend on efficient integration of disparate and often heterogeneous data. The integration of data is governed by data-intensive flows and is driven by a set of information requirements. Designing such flows is in general a complex process, which due to the complexity of business environments is hard to be done manually. In this chapter, we deal with the challenge of efficient design and maintenance of data-intensive flows and propose an incremental approach, namely CoAl, for semi-automatically consolidating data-intensive flows satisfying a given set of information requirements. CoAl works at the logical level and consolidates data flows from either high-level information requirements or platform-specific programs. As CoAl integrates a new data flow, it opts for maximal reuse of existing flows and applies a customizable cost model tuned for minimizing the overall cost of a unified so-

lution. We demonstrate the efficiency and effectiveness of our approach through an experimental evaluation using our implemented prototype.

1 Introduction

The complexity of business environments constantly grows, both with regard to the amount of data relevant for making strategic decisions and the complexity of included business processes. Today's dynamic and competitive markets often imply rapid (e.g., near real-time) and accurate decision making. Relevant data are stored across a variety of data repositories, possibly using different data models and formats, and potentially crossed with numerous external sources for various context-aware analysis. A data integration process combines data residing on different sources and provides a unified view of this data for a user [106]. For example, in a data warehousing (DW) context, data integration is implemented through extract-transform-load (ETL) processes. Generally, an ETL process represents a data-intensive flow (or simply, data flow) that extracts, cleans, and transforms data from multiple, often heterogeneous data sources and finally, delivers data for further analysis.

There are various challenges related to data flow design. Here we consider two: design evolution and design complexity.

A major challenge that BI decision-makers face relates to the evolution of business requirements. These changes are more frequent at the early stages of a DW design project [34] and in part, this is due to a growing use of agile methodologies in data flow design and BI systems in general [79]. But changes may happen during the entire DW lifecycle. Having an up-and-running DW system satisfying an initial set of requirements is still a subject to various changes as the business evolves. The data flows populating a DW, as other software artifacts, do not lend themselves nicely to evolution events and in general, due to their complexity, maintaining them manually is hard. The situation is even more critical in today's BI settings, where on-the-fly decision making requires faster and more efficient adapting to changes. Changes in business needs may result in new, changed, or removed information requirements. Thus having an incremental and agile solution that can automatically absorb occurred changes and produce a flow satisfying the complete set of requirements would largely facilitate the design and maintenance of data-intensive flows.

In an enterprise environment data is usually shared among users with varying technical skills and needs, involved in different parts of a business process. Typical real-world data-intensive workloads have high temporal locality, having 80% of data reused in a range from minutes to hours [36]. However, the cost of accessing these data, especially in distributed scenarios, is often high [24]. At the same time, intertwined business processes may also imply overlapping of data processing. For instance, a sales department may analyze the revenue

of the sales for the past year, while finance may be interested in the overall net profit. Computing the net profit can largely benefit from the total revenue already computed for the sales department and thus, it could benefit from the sales data flow too. The concept of reusing partial results is not new. Software and data reuse scenarios in data integration have been proposed in the past, showing that such reuse would result in substantial cost savings, especially for large, complex business environments [148]. Data flow reuse could result in a significant reduce in design complexity, but also in intermediate flow executions and thus, in total execution time too [24].

In this chapter, we address these challenges and present an approach to efficient, incremental consolidation of data-intensive flows. Following common practice, our method iterates over information requirements to create the final design. In doing that, we show how to efficiently accommodate a new information requirement to an existing design and also, how to update a design in lieu of an evolving information requirement. To this end, we describe a Consolidation Algorithm (CoAl) for data-intensive flows. Without loss of generality, we assume that starting with a set of information requirements, we create a data flow per requirement. The final design satisfying all requirements comprises a multi-flow. As 'coal' is formed after the process and extreme compaction of layers of partially decomposed materials[1], CoAl processes individual data flows and incrementally consolidates them into a unified multi-flow.

CoAl deals with design evolution by providing designers with an agile solution for the design of data flows. CoAl assists the early stages of the design process when for only a few requirements we need to build a running data flow from scratch. But, it also helps during the entire flow lifecycle when the existing multi-flow must be efficiently accommodated to satisfy new, removed, or changed information requirements.

CoAl reduces design complexity with aggressive information and software reuse. Per requirement, it searches for the largest data and operation overlap in the existing data flow design. To boost the reuse of existing design elements when trying to satisfy a new information requirement (i.e., when integrating a new data flow), CoAl aligns the order of data flow operations by applying generic equivalence rules. Note that in the context of data integration, the reuse of both data and code (e.g., having a single computation shared by multiple flows as depicted in Figure 3.4) besides reducing flow complexity, might also lead to faster execution, better resource usage, and higher data quality and consistency [148, 94].

In addition, since data-intensive flows comprise critical processes in today's BI systems, CoAl accounts for the cost of produced data flows when searching for opportunities to integrate new data flows. CoAl uses a tunable cost model to perform multi-flow, logical optimization to create a unified flow design that

[1] src. Wikipedia

satisfies all information requirements. Here, we focus on maximizing the reuse of data and operations, but the algorithm can be configured to work with different cost models, taking into account different quality factors of data flows (e.g., [161]).

As a final remark, CoAl works at the logical level and is therefore applicable to a variety of approaches that generate logical data flows from information requirements expressed either as high level business objects (e.g., [18, 13, 146]) or in engine specific languages (e.g., [92]).

In particular, our main contributions are as follows.

- We present a semi-automatic approach to the incremental design of data-intensive flows.
- We introduce a novel consolidation algorithm, called CoAl, that tackles the data flow integration problem from the context of data and code reuse, while at the same time taking into account the cost of the produced data flow design.
- We present generic methods for reordering and comparing data flow operations that are applied while searching for the consolidation solutions that will increase data and operation reuse.
- We experimentally evaluate our approach by using an implemented prototype. A set of empirical tests have been performed to assess the CoAl's efficiency and improvements in overall execution time.

A short version of this chapter was published in Jovanovic et al. [88].

Outline. Section 2 describes a running example and formalizes the problem at hand. Section 3 discusses the main challenges: operations reordering and comparison. Section 4 presents the CoAl algorithm. Section 5 reports on our experimental findings. Sections 6 and 7 discuss related work and conclude the chapter, respectively.

2 Overview

2.1 Running Example

Figure 4.1 shows an abstraction of the TPC-H schema [5]. Figure 3.2 illustrates four example information requirements extracted from TPC-H queries. In a sense, our example here is adapted by reverse engineering the use case described by the TPC-H schema and queries.

We create a data flow per each requirement in Figure 3.2 (see Figures 3.3 and 3.6). In the literature, there are many methods dealing with such task, either manually (e.g., [18, 13]) or automatically (e.g., [146]). Independent of a method

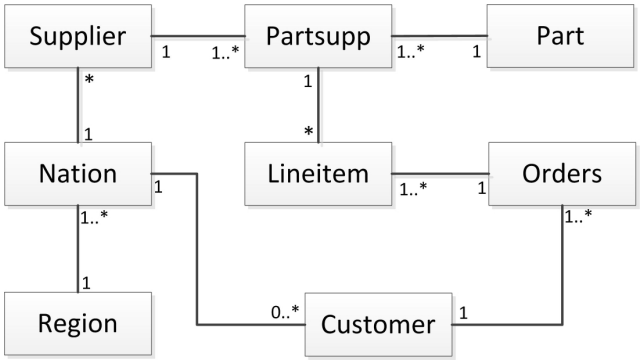


Fig. 3.1: TPC-H Schema

for creating data flows from single requirements, CoAl focuses on the problem of integrating these flows into a unified flow that satisfies all requirements.

Consider the DIF-1 and DIF-2 data flows depicted in Figure 3.3 that satisfy the requirements IR_1 and IR_2, respectively. We define the referent data flow as the integrated multi-flow satisfying a number of requirements already modeled (we start from IR_1) and the new data flow as the flow satisfying the new requirement (IR_2).

In terms of graphical notation, the squares represent source or target data stores, whereas the circles represent data flow operations. Operations of each data flow are uniquely named using the following notation:
`OPNAME = OPTYPE+OPID+";"+{FLOWIDs}`.

Note that the flow IDs at the end define the set of data flows that share the given operation. They are optional and can be omitted for single (non integrated) data flows.

Observe that the DIF-1 and DIF-2 data flows have a number of common operations. CoAl exploits this and creates an alternative, equivalent data flow

IR_1: Revenue of the sales for the parts ordered in the past year, per quarter.

IR_2: Net profit of the sales for the parts ordered in the last year, per quarter.

IR_3: Top 10 automobile industry customers based on the quantity of shipped parts, ordered in the last year.

IR_4: Sorted list of quantities of parts shipped to Spanish customers, ordered in the last year.

Fig. 3.2: Information Requirements

satisfying both requirements IR_1 and IR_2, such that the reuse of the data stores and operations of DIF-1 is maximal (see Figure 3.4). We then continue and integrate the remaining requirements to create a unified multi-flow that satisfies all four requirements (i.e., IR_1-IR_4), see Figure 3.7.

2.2 Preliminaries and Notation

We build upon past work on ETL workflow formalization [158] and model generic data flows as follows. A data-intensive flow (*DIF*) is formally defined as a directed acyclic graph consisting of a set of nodes (V), which are either data stores (DS) or operations (O), while the graph edges (E) represent a directed data flow among the nodes of the graph ($v_1 \prec v_2$). We write:

$DIF = (\mathrm{V}, \mathrm{E})$, such that: $\mathrm{V} = \mathrm{DS} \cup \mathrm{O}$,

$\forall e \in \mathrm{E} : \exists (v_1, v_2), v_1 \in \mathrm{V} \wedge v_2 \in \mathrm{V} \wedge v_1 \prec v_2$

In the rest of this chapter we use the terms 'flow' and 'graph' interchangeably, while the above formalization of a data flow holds for an individual data flow, as well as for an integrated multi-flow.

Data store nodes (DS) can represent either a source data store (DS_S, e.g., input DB table, file, etc.) or a result data store which in general is not necessarily materialized (DS_R, e.g., output DB table, file, report, virtual cube, etc.), i.e., $\mathrm{DS} = \mathrm{DS}_S \cup \mathrm{DS}_R$. Data store nodes are defined by a schema (i.e., finite list of attributes) and a connection to a source or a target storage for respectively extracting or loading the data. Furthermore, we formally define a data flow operation as a quintuple:

$$o = (\mathbb{I}, \mathbb{O}, \mathbb{S}, \mathrm{Pre}, \mathrm{Post})$$

- $\mathbb{I}$ represents a set of input schemata, where each schema (I_i) characterizes an input from a single predecessor operation and is defined with a finite set of attributes coming from that operation (i.e., $\mathrm{I} = \{a_1, .., a_{n_I}\}$). This definition is generic in that it allows the arbitrary input arity of flow operations.
  Example. Notice in Figure 3.3 that some operations like `UDF3` are unary and thus have only one input schema, while operations like `Join2` are binary and expect two input schemata. □

- $\mathbb{O}$ represents a set of output schemata, where each schema (O_i) characterizes an output to a single succeeding operation and is defined with a finite set of attributes (i.e., $\mathrm{O} = \{a_1, .., a_{nO}\}$).
  Example. The operations in DIF-12 of Figure 3.4 can have either a single output schema (e.g., `UDF3`, `Filter1`, and `Aggr4`), or as it is the case of `Join2` two equivalent output schemata sending the same data to two different subflows. □

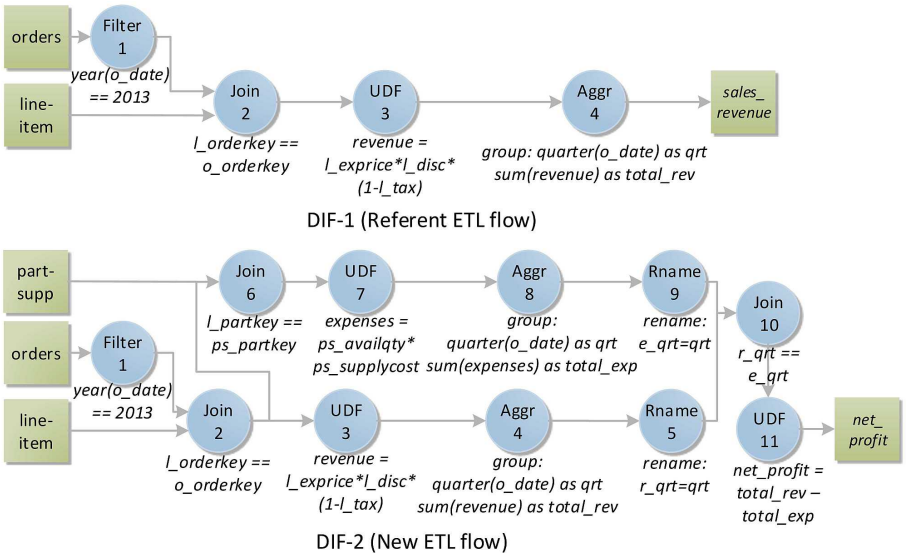


Fig. 3.3: Example data-intensive flows for IR_1 and IR_2

- $\mathbb{S}$ represents the formalization of operator's semantics, i.e., a finite set of expressions that interprets the processing semantics of an operator. Due to their inherent complexity and diversity, in order to automate the processing of data flow operations (e.g., comparison, discovery of different operation properties), we express the semantics of a generic data flow operation as a finite set of normalized expression trees. That is, a binary search tree, alphanumerically ordered on the expression elements (i.e., operators, function calls, variables, and constants), whilst respecting the valid order of operators when evaluating the expression. CoAl assumes that the semantics' formalization is generic, and similar formalization techniques (e.g., [189]) can be used seamlessly in our approach.

  Example. To express the semantics of data flow operations in DIF-1 (Figure 3.3) we build the normalized expression trees showed in Figure 3.5. □

To further determine how the operations' semantics affect their interdependence in a data flow, we use a set of data flow operation properties. Relying solely on a set of algebraic properties of data flow operations (e.g., [124, 158]) is not enough to take full advantage of the potential for the data flow analysis. Different 'physio-logical' properties that can be extracted from a data flow, additionally boost the automation of the flow analysis and equivalent operation reordering. For example, such idea has been introduced for the context of optimizing generalized MapReduce data flows by extracting properties like

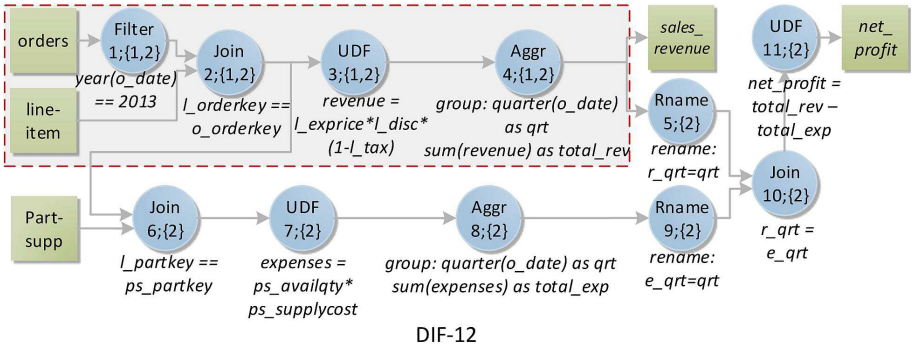


Fig. 3.4: Integrated data-intensive multi-flow (IR_1-IR_2)

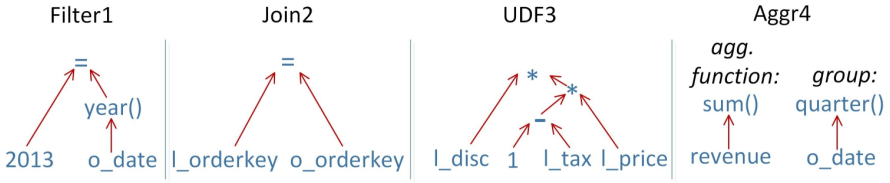


Fig. 3.5: Normalized expression trees (DIF-1 operations)

attribute values [78]. Additionally motivated by this work, we extend and generalize the idea of having a customizable set of properties characterizing data flow operations. In this chapter, we considered the following set of operation properties:

- Schema (S). Attributes being used, emitted, or removed by an operation.
- Values (V). Attributes whose values are used or produced by an operation.
- Order (O). Indicator if the order of the tuples (i.e., rows or records) in the processed dataset affects the results or is being produced by an operation.

We have analyzed the applicability of these properties over the types of operations in the example ETL tools; both a commercial, i.e., Oracle Warehouse Builder (OWB 11.2), and an open source data integration (ETL) tool, i.e., Pentaho Data Integration (PDI 5.0). This analysis has showed us that these three properties cover frequently used operations, while our approach is generic and allows the extension to other categories of data processing operations as needed.

Furthermore, in terms of these three properties, for each instance of a data flow operation, we define pre- (Pre) and post-conditions (Post) of a data flow

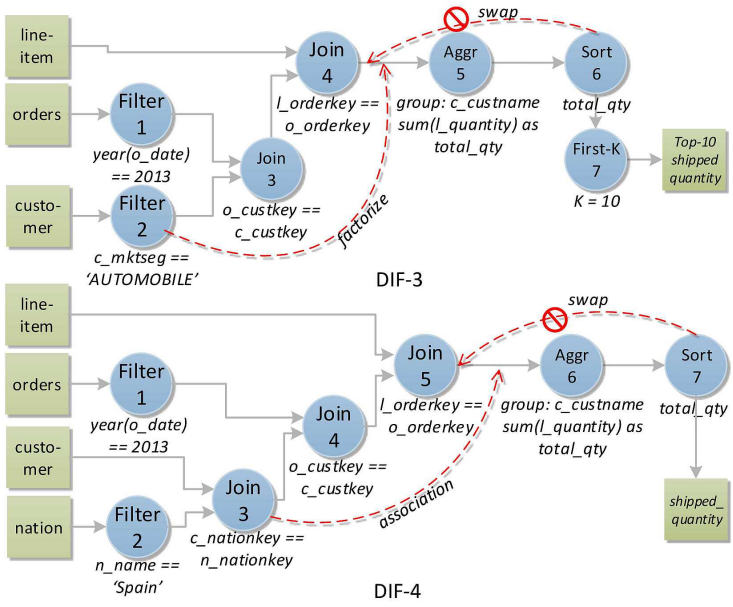


Fig. 3.6: Example data-intensive flows for IR_3 and IR_4

operation.

Depending on the properties that an operation "consumes" (i.e., the results of an operation are affected by the specific value of that input property), we define the pre-conditions of an operation as follows:
Pre = $(S_{pre}, V_{pre}, O_{pre})$, such that:

- S_{pre} (consumed schema) is a subset of attributes of input schemata ($\mathbb{I}$) that are used by the operation.
- V_{pre} (consumed values) is a subset of attributes of the consumed schema ($V_{pre} \subseteq S_{pre}$) whose values are used by the operation.
- O_{pre} (consumed order) is a boolean indicator that specifies whether the results of the operation processing are affected by an order of its input dataset.

Example. The `UDF3` operation of DIF-1 in Figure 3.3 uses the attributes `l_exprice`, `l_disc`, and `l_tax` from the input (i.e., $S_{pre} = \{l\_exprice, l\_disc, l\_tax\}$), and moreover it uses their values for computing the value of the output attribute `revenue` (i.e., $V_{pre} = \{l\_exprice, l\_disc, l\_tax\}$). On the other hand, notice that the `Rename` operations in DIF-2, require the attribute `qrt` at the input (i.e., $S_{pre} = \{qrt\}$), while the value of that attribute is not used by these operations (i.e., $V_{pre} = \varnothing$). Likewise, the value of the `revenue` attribute

resulting from the `UDF` operation, is not affected by the order of the input dataset (i.e., $O_{pre} = false$). □

In a similar way, but now depending on the properties an operation "produces" (i.e., generates the specific value of that property at the output), we define the post-conditions of a data flow operation as:
Post = $(S_{post\_gen}, S_{post\_rem}, V_{post}, O_{post})$, such that:

- S_{post_gen} (generated schema) is a finite set of new attributes that the operation generates at the output.
- S_{post_rem} (removed schema) is a subset of input attributes that the operation removes from the output.
- V_{post} (produced values) is a finite set of attributes whose values are either produced or modified by the operation.
- O_{post} (produced order) is a boolean indicator that specifies whether the operation processing generates a specific order of the output dataset.

Note that we need to distinguish two sets (i.e., S_{post_gen} and S_{post_rem}) to specify the schema property of a post-condition in order to determine the dependency of operations in a data flow. We clarify this when discussing the generic equivalence rules in Section 3.1.

Example. The operation `Aggr4` of DIF-1 in Figure 3.3 modifies the schema provided at the input and produces the new schema at the output, removing all the input attributes (i.e., $S_{post\_rem} = \{\#all\}$) and generating the grouping attribute `qrt` and the aggregated attribute `total_revenue` (i.e., $S_{post\_gen} = \{qrt, total\_revenue\}$). `Aggr4` also produces the new value for the aggregated `revenue` attribute and the value of the grouping attribute `qrt` (i.e., $V_{post} = \{qrt, total\_revenue\}$), and affects the order of the output dataset (i.e., $O_{post} = true$). □

For extending the set of considered "physio-logical" properties, an instantiation of each property must be defined both at the input (Pre) and the output (Post) of each data flow operation. If an operation type does not consume/produce a property, the corresponding instantiation is empty (or `false`, see the order property).

Finally, notice that the Pre and Post conditions of a data flow operation provide the needed knowledge to determine the dependencies among operations when performing equivalence transformations for reordering operations in a generic data flow. We discuss this in more detail in Section 3.1.

2.3 Problem Statement

We formalize the problem of the incremental data flow consolidation, by introducing the three main design operations to integrate, remove, and change a data flow.

Integrate a data flow ($\cdot_{int}$): When a new information requirement comes, we need to integrate the data flow that satisfies it, into the existing data flow. Considering that a data flow at the logical level is modeled as a directed acyclic graph (DAG), in the context of integrating new data flow, at each step we assume two graphs:

- Referent graph. An existing multi-flow satisfying the n current information requirements.
  $DIF_{ref} = (\mathrm{V}_{ref}, \mathrm{E}_{ref}) : DIF_{ref} \vDash \{IR_1, .., IR_n\}$
- New graph. A data flow satisfying the upcoming requirement.
  $DIF_{new} = (\mathrm{V}_{new}, \mathrm{E}_{new}) : DIF_{new} \vDash IR_{new}$

In addition, for each information requirement (IR_i) and a data flow (DIF_{ref}), we define a requirement subgraph function (i.e., $DIF_i = (V_i, E_i) = G(DIF_{ref}, IR_i)$), such that G returns a subgraph DIF_i of DIF_{ref}, if the requirement IR_i can be satisfied by DIF_{ref} using DIF_i. Otherwise, it returns $NULL$.

Example. Notice that the shaded subgraph within the multi-flow DIF-12 in Figure 3.4, ending in the sales_revenue data store, is a requirement subgraph satisfying IR_1. □

Intuitively, to integrate two data flow graphs ($DIF_{ref} \cdot_{int} DIF_{new}$), we look for the maximal overlapping of their nodes (i.e., data sources and operations). As a result, the integrated multi-flow (i.e., DIF_{int}) must logically subsume the requirement subgraphs of all the requirements satisfied by DIF_{ref} and DIF_{new} and consequently satisfy the entailed information requirements.

We define the integrate design operation as:
$DIF_{int} = DIF_{ref} \cdot_{int} DIF_{new} = (\mathrm{V}_{int}, \mathrm{E}_{int})$, s.t.:
$\forall IR_i, i = 1, .., n : G(DIF_{int}, IR_i) <> NULL$,
$G(DIF_{int}, IR_{new}) <> NULL$.
Thus, we say that: $DIF_{int} \vDash \{IR_1, .., IR_n, IR_{new}\}$

Example. The multi-flow integrated from IR_1-IR_4 is shown in Figure 3.7, where the overlapping operations are shown inside the shaded areas. □

Remove a data flow ($\cdot_{rem}$): In the case a user wants to remove an information requirement (IR_{rem}) from her analysis (i.e., $DIF_{ref} \vDash \{IR_1, .., IR_n\}$), we need to remove its requirement subgraph (i.e., $DIF_{rem} = (V_{rem}, E_{rem}) = G(DIF_{ref}, IR_{rem})$), without affecting the satisfiability of other requirements. Formally:
$DIF_{int} = DIF_{ref} \cdot_{rem} DIF_{rem}$, s.t.:
$\forall IR_i, i = \{1, .., n\} \backslash \{rem\} : G(DIF_{int}, IR_i) <> NULL$,
$((V_{rem} \backslash \bigcup_{\forall i \in \{1,..,n\} \backslash \{rem\}} V_i) \cap V_{int}) = \varnothing$,
$((E_{rem} \backslash \bigcup_{\forall i \in \{1,..,n\} \backslash \{rem\}} E_i) \cap E_{int}) = \varnothing$.

To this end, while integrating data flows, CoAl maintains metadata, consisting of two maps that for each node (datastore or operation) and edge of the integrated multi-flow graph, keeps a share counter for the total number of

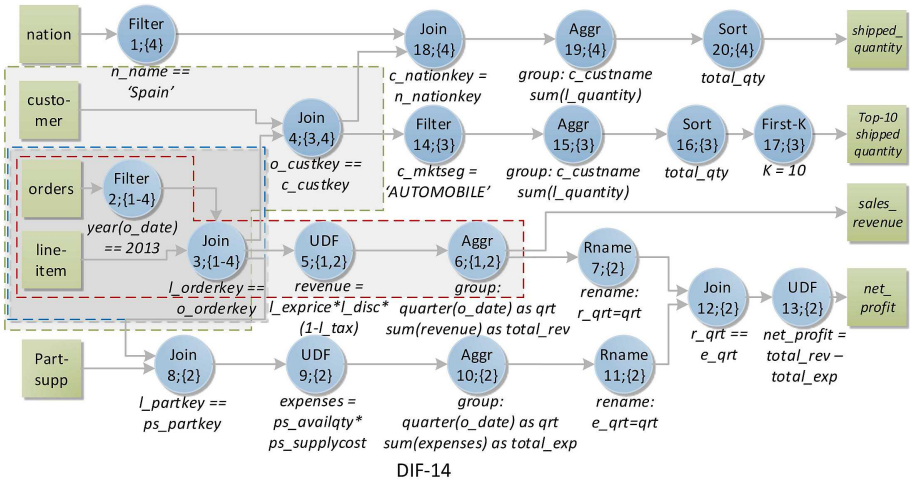


Fig. 3.7: Integrated data-intensive multi-flow (IR_1-IR_4)

input data flows that use that node (i.e., $\forall v \in V_{int} : \exists c_v > 0$) or edge (i.e., $\forall e \in E_{int} : \exists c_e > 0$). For target data store nodes that satisfy requirements at hand, CoAl also keeps the requirement identifier.

Thus, when a user decides to remove an information requirement, the system will search through its requirement subgraph, starting from a target node, and decrements the share counter of the visited nodes and edges. If the counter drops to zero, the system will remove the node or the edge from the graph, as it is not used by any of the remaining input data flows anymore.

Change a data flow ($\cdot_{chg}$): Similarly, changing an information requirement ($IR_{chg} \rightsquigarrow IR_{chg'}$), results in modifying the subgraph of a referent data flow that satisfies this requirement ($DIF_{chg} \rightsquigarrow DIF_{chg'}$), while preserving the satisfiability of other requirements in the analysis. Intuitively, the operation for changing a data flow can be reduced to the sequence of the previous two operations, i.e., remove and integrate. Formally:
$DIF_{int} = DIF_{ref} \cdot_{chg} (DIF_{chg} \rightsquigarrow DIF_{chg'}) = (DIF_{ref} \cdot_{rem} DIF_{chg}) \cdot_{int} DIF_{chg'}$.

Next, we discuss the challenges in data flow consolidation and present the CoAl algorithm.

3 Data Flow Consolidation Challenges

To search for the overlapping between DIF-12 and DIF-3 (i.e., Figures 3.4 and 3.6, respectively), we first find `orders` and `lineitem` as the shared source data stores. Then, starting from these nodes we proceed with comparing operations going further in the encompassing subgraphs towards the result data store

nodes, respectively, `Top-10 shipped quantity` and `sales_revenue`. Taking into account the previous example, we notice several challenges that arise when searching the overlapping operations in the referent and the new data flows.

1. Going from the `orders` nodes in DIF-12 and DIF-3, notice that after the common Filter operations we identify Join operations in both data flows, `Join2` and `Join3`, respectively. However, even though one input of these Join operations coincides in both flows, the second input differs, and thus we do not proceed with comparing these operations.
   Incremental advancement. To guarantee the semantic overlapping of two data flows, we must proceed incrementally starting from the common source nodes. That is, for comparing any two operations of two data flows, we must ensure that the predecessors of both operations coincide.

2. Although we do not compare the two Join operations, we continue our search. Note that in DIF-3 there is another Join operation (i.e., `Join4`) and that it is possible to exchange the order of this operation with the previously discussed `Join3` without affecting the semantics of the DIF-3 data flow. As a result, we find a larger set of overlapping operations between DIF-3 and DIF-12.
   Operation reordering. Operation reordering under a set of equivalence rules is a widely used optimization technique, e.g., for pushing selective operators early in a flow. When comparing data flows we can benefit from such technique to boost finding the maximal overlapping of operations whilst keeping the equivalent semantics of input data flows.

3. In each step, like after reordering the Join operations in DIF-3, when we find that the predecessors of two operations coincide, we proceed with comparing these two operations. Thus the comparison of data flow operations arises a third challenge in consolidating data flows considering the inherent complexity and variety of data flow operations.
   Operation comparison. To integrate the operations of two data flows we need to compare them to ensure that they provide the equivalent dataset at the output. To automate their comparison we need to formalize the semantics of data flow operations.

Before presenting CoAl that solves the first challenge, we discuss the theoretical aspects of the last two challenges.

3.1 Operation reordering

Operation reordering has been widely studied in the context of data flow optimization, both for traditional relational algebra operators (e.g., [124]) and generic data flows (e.g., [158, 78]). Different reordering scenarios have proven to improve the performance of data flows (e.g., pushing selective operations

early in the flow). Conversely, we observe that reordering techniques can also be used for consolidating data flows by finding the maximal overlapping of flow operations. Thus, here we introduce a set of data flow transformations and generic equivalence rules which ensure that these transformations lead to a semantically equivalent data flow.

Following the previously proposed set of flow transformations in the context of ETL processes [158], in CoAl we extend this set considering also the associative property of n-ary operations (e.g., Join) and thus rely on the following four flow transformations used for reordering the operations.

- Swap. Applied to a pair of adjacent unary operations, it interchanges the order of these operations.
- Distribute/Factorize. Applied on a unary operation over an adjacent n-ary operation, it respectively distributes the unary operation over the adjacent n-ary operation or factorize several unary operations over the adjacent n-ary operation.
- Merge/Split. Applied on a set of adjacent unary operations, it respectively merges several operations into a single unary operation or splits a unary operation into several unary operation.
- (Re-)associate. Applied on a pair of mutually associative n-ary operations, it interchanges the order in which these operations are executed.

Example. Examples of these transformations applied to integrate DIF-3 and DIF-4 into the referent data flow (Figure 3.4) are showed in Figure 3.6. □

Furthermore, we define here the equivalence rules applicable to a generic set of data flow operations, which must hold in order to guarantee a semantically equivalent data flow after performing the previous reordering transformations. Our generic equivalence rules are expressed in terms of the operation properties defined in Section 2.2 (i.e., schema, values, order). Notice that the equivalence rules defined in terms of these properties can be conservative in some cases and prevent some valid reorderings, but whenever applied, they do guarantee the semantic equivalence of a reordered data flow.

Let's consider two adjacent operations o_A and o_B in a data flow, such that o_A precedes o_B (i.e., $o_A \prec o_B$). Thus, to ensure the equivalence transformation, CoAl checks if there is no conflict of the properties among these two operations, i.e., if the following constraints hold.

- Schema conflict. We must guarantee that all attributes of the schemata used by operations o_A and o_B are available also after the operation reordering.
  $(S_{post\_rem_B} \cap S_{pre_A} = \varnothing) \wedge (S_{post\_gen_A} \cap S_{pre_B} = \varnothing)$

Example. In DIF-2 (Figure 3.3), notice that `Aggr4` generates the attribute `qrt` that is accessed by the `Rname5` operation, and thus there is a conflict of the schema property, which prevents the swap between these two operations □

- Values conflict. We must guarantee that none of the attributes' values used by one operation are modified by another operation.
  $(V_{post_B} \cap V_{pre_A} = \varnothing) \wedge (V_{post_A} \cap V_{pre_B} = \varnothing)$
  Example. In DIF-3 (Figure 3.6), `Aggr5` generates the value of attribute `total_qty` that is consumed by the `Sort6` operation, and thus there is a conflict of the value property, which prevents the swap between these two operations. □

- Order conflict. We must guarantee that if the results of one operation are affected by a specific order of an input dataset, another operation does not modify the order of the dataset at the output.
  $(O_{post_B} \Rightarrow \neg O_{pre_A}) \wedge (O_{post_A} \Rightarrow \neg O_{pre_B})$
  Example. For the operations `Sort6` and `First-K7`[2] of DIF-3, CoAl finds an order conflict, since `Sort6` affects the order of the tuples in the output dataset, while the results of `First-K7` are obviously affected by the order of the input dataset. □

Besides these, if both o_A and o_B are n-ary operations, and hence CoAl tries to apply the (re-)associate transformation, we must also ensure that o_A and o_B are mutually associative operations.
Example. Notice that the equi-join operations (e.g., `Join2` and `Join6` operations in DIF-12 in Figure 3.3) are associative, as well as the typical set union and intersection operations, whilst for example outer joins (left, right, and full) and set difference in general do not satisfy associative property and cannot be reordered using the (re-)associate transformation. □

Finally, only if for all the above conditions, CoAl detects no conflict, it can consider reordering o_A and o_B.

Proof. For the proof that the first three flow transformations (i.e., swap, distribute/factorize, and merge/split) lead to a semantically equivalent data flow, we refer the reader to the work of Simitsis et. al, [158]. Furthermore, the proof that the association transformation leads to a semantically equivalent data flow, when applied over the operations that satisfy the associative property, is based on the assumption of the equivalence of the output schemata and the output datasets before and after the association is applied.

1. Schema equivalence. Regardless of the order of the two adjacent n-ary operations that satisfy the associative property, the equivalent output

[2]The First-K operation keeps only the top K tuples of the input dataset (e.g., `LIMIT` in SQL syntax).

schemata are provided at the output (e.g., concatenated schemata in the case of equi-join, or equivalent schemata in the case of set union and set intersection).

2. Dataset equivalence. The associative property, if satisfied by the two adjacent n-ary operations, guarantees the equivalence of the output datasets before and after the (re-)associate transformation is applied.

3.2 Operations comparison

Another challenge for consolidating two data flows is finding the matching operations between these flows. In general, operations o_A and o_B, only match if they imply the equivalent semantics and the subgraphs having o_A and o_B as their sinks provide equal datasets.

We consider four possible outcomes of comparing two operations, o_A and o_B, from a referent and a new data flow, respectively (see Figure 3.8).

(1) Full match: the compared operations are equivalent (i.e., $o_A \equiv o_B$). In that case, we can consolidate the two operations as a single one in the integrated multi-flow.

Example. In DIF-1 and DIF-2 of Figure 3.3 we find some fully matching operations, e.g., `Filter1` or `UDF3`. □

(2) Partial (referent) match: the results of o_B are subsumed by the results of o_A, thus o_B can partially benefit from the transformations already performed by o_A (i.e., $o_B \sqsubseteq o_A$). Then, both operations can be partially collapsed as depicted in Figure 3.8(2). Furthermore, the consolidation of the partially matched operation o_B may involve an additional transformation (i.e., $o_{B'}$) for obtaining the original output data.

(3) Partial (new) match: the results of o_A are subsumed by the results of o_B (i.e., $o_A \sqsubseteq o_B$). Similarly, o_A can benefit from the transformations performed by o_B.

Example. Consider an alternative scenario where the requirement IR_2 only takes into account urgent orders, i.e., `Filter1` uses `year(o_date)=2013 AND o_orderprior = '1-URGENT'`. Then, we would find a partial match of the Filter operations in DIF-1 and DIF-2, since `Filter1` in DIF-2 could partially benefit from `Filter1` in DIF-1 and would need an additional filter using `o_orderprior = '1-URGENT'`. □

(4) No match: Finally, it may happen that neither o_B nor o_A can benefit from one another, (i.e., $o_A \not\sqsubseteq\not\sqsupseteq o_B$). Then, the two operations cannot be consolidated. Thus, we introduce a fork in the already matched subflow, as shown in Figure 3.8(4). The fork in such case requires the copy-partition functionality (i.e., parallel flows).

Following the notation in Section 2.2, here we formalize the comparison of two operations o_A and o_B as follows:

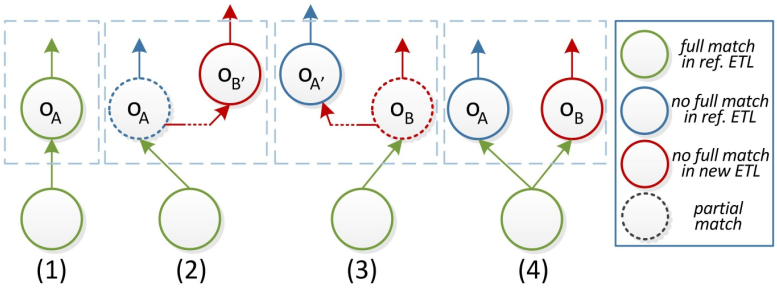


Fig. 3.8: Integration of data flow operations

- $o_A(\mathbb{I}_A, \mathbb{O}_A, S_A, Pre_A, Post_A) \equiv o_B(\mathbb{I}_B, \mathbb{O}_B, S_B, Pre_B, Post_B)$ iff $\mathbb{I}_A = \mathbb{I}_B \wedge \mathbb{O}_A = \mathbb{O}_B \wedge S_A \equiv S_B$;
- $o_A(\mathbb{I}_A, \mathbb{O}_A, S_A, Pre_A, Post_A) \sqsupseteq o_B(\mathbb{I}_B, \mathbb{O}_B, S_B, Pre_B, Post_B)$ iff $\exists o_{B'}(\mathbb{I}_{B'}, \mathbb{O}_{B'}, S_{B'}, Pre_{B'}, Post_{B'}) : \mathbb{I}_A = \mathbb{I}_B \wedge \mathbb{O}_A = \mathbb{I}_{B'} \wedge \mathbb{O}_{B'} = \mathbb{O}_B \wedge S_B \equiv S_A \circ S_{B'}$;

Intuitively, we find the equivalence (i.e., full match) of two data flow operations, if their input and output schemata coincide, while at the same time the operations define the equivalent semantics (i.e., $S_A \equiv S_B$). To find the partial match between two operation o_A and o_B (i.e., cases (2) and (3) in Figure 3.8), we need to check if the result of one operation (o_B) can be partially obtained from the results of another operation (o_A), i.e., if the results of o_B are subsumed by the results of o_A. To this end, we look for a new operation (o'_B) so that the semantics of operations o_A and o'_B can be functionally composed to imply the equivalent semantics as the operation o_B and hence provide the same result dataset. Note that in general, finding the subsumption among the expressions is known to be a challenging problem. Thus for arbitrary operation expressions, we rely on the current state of the art for reasoning over the expressions and assume that they are in conjunctive normal form.

4 Consolidation Algorithm

We now present the CoAl algorithm for data flow consolidation.

Intuitively, the incremental advancement property defined in the previous section, requires that for solving the problem of integrating any two data flow graphs, by maximizing the reuse of their data and operations, we first need to recursively solve the subproblems of integrating their subgraphs, starting from the data sources, and following a topological order of nodes in the graphs.

Given the clear ordering and dependencies between these subproblems, we formulate the problem of integrating data-intensive flows as a dynamic pro-

gramming problem. We devise a bottom-up, iterative variant of the algorithm that efficiently solves the problem in our case.

In particular, CoAl starts with two data flow graphs, the referent (DIF_{ref}) and the new (DIF_{new}), and proceeds following a topological order of the data flow operations in a graph, starting from the matched leaf (source) nodes. CoAl iteratively searches larger matching opportunities between their operations by applying the generic equivalence rules (Section 3.1), hence considering reordering without modifying the semantics of the involved data flows. At each iteration, CoAl compares two operations (one from each data flow), and continues only if a full match is found (Section 3.2). This guarantees the following two invariants:

(I_1): At each step, only one pair of operations of referent and new data flows can be partially or fully matched.

(I_2): A new match is added to the set of matched operations if and only if the operations themselves match and their input flows have been previously fully matched.

In addition, when searching for next operations to match, the following invariant must also hold:

(I_3): An operation can be reordered to be matched next, if and only if such reordering does not change the semantics (i.e., output) of a data flow.

The consequence of these invariants is that a pair of matched operations is eventually consolidated in the output, integrated multi-flow, if the flows they belong to can be reordered so that their children are fully matched.

Thus, the correctness of the CoAl algorithm, in the sense that it integrates input data flows without affecting their outputs, is guaranteed by the three characteristics of the algorithm –i.e., operation comparison, incremental advancement, and equivalent operation reordering– discussed in Section 3, and demonstrated respectively with the invariants I_1, I_2, and I_3.

Example. We illustrate different steps of CoAl, using the scenario of integrating data flow DIF-3 (Figure 3.6), into the referent DIF-12 multi-flow (Figure 3.4). □

In general, CoAl comprises four phases (see Algorithm 1): i) search for the next operations to match; ii) compare the next operations; iii) reorder input data flows if a match has been found; and iv) integrate the solution with the lowest estimated cost. The first three phases are executed in each iteration of CoAl, while the last one is executed once, when no matching possibility is left.

Before detailing the four phases, we present the main structures maintained by CoAl while looking for the final consolidation solution.

1. matchQ: A priority queue that contains pairs of referent and new data flows with currently overlapping areas which can be potentially extended with new matching operations.
   matchQ ::= *matchDIFPair, matchQ*|*matchDIFPair;*

Algorithm 1 CoAl

inputs: DIF_{ref}, DIF_{new}; output: DIF_{int}

```
 1: altList := {[[DIF_ref,DIF_new,∅,∅],cost_noInt]};                    ▷ no int. alternative
 2: matchQ := matchLeafs(DIF_ref,DIF_new);
 3: while matchQ ≠ ∅ do
 4:     matchDIFPair[DIF′_ref,DIF′_new,allMatches,lastMatches] := dequeue(matchQ);
 5:     [matchOpsPair[o_ref,o_new],edge_ref] := dequeue(lastMatches);
 6:     nextOps_ref := findNextForMatch(DIF′_ref,o_ref,edge_ref);          ▷ Algorithm 2
 7:     nextOps_new := findNextForMatch(DIF′_new,o_new,edge_o′new);        ▷ Algorithm 2
 8:     for all pair(o′_ref ∈ nextOps_ref, o′_new ∈ nextOps_new) do
 9:         if o′_ref ≡ o′_new ∨ o′_ref ⊑ o′_new ∨ o′_new ⊑ o′_ref then
10:             DIF″_ref := reorder(DIF′_ref,o′_ref,o_ref);
11:             DIF″_new := reorder(DIF′_new,o′_new,o_new);
12:             insert(allMatches, [[o′_ref,o′_new],intInfo]);
13:             if o′_ref ≡ o′_new  then                                   ▷ full match
14:                 for i:=1 to deg(o′_ref) enqueue(lastMatches, [[o′_ref,o′_new],edge_ref_i]);
15:                 enqueue(matchQ, [DIF″_ref,DIF″_new,allMatches,lastMatches]);
16:             else if o′_ref ⊑ o′_new ∨ o′_new ⊑ o′_ref then              ▷ partial match
17:                 if lastMatches = ∅ then                  ▷ no further matchings avail.
18:                     insert(altList, [[DIF″_ref,DIF″_new,allMatches,lastMatches],cost″]);
19:                 else
20:                     enqueue(matchQ, [DIF″_ref,DIF″_new,allMatches,lastMatches]);
21:                 end if
22:             end if
23:         end if
24:     end for
25:     if no matching found then
26:         if lastMatches = ∅ then                              ▷ no further matchings avail.
27:             insert(altList,[[DIF′_ref,DIF′_new,allMatches,lastMatches],cost′]);
28:         else
29:             enqueue(matchQ, [DIF′_ref,DIF′_new,allMatches,lastMatches]);
30:         end if
31:     end if
32: end while
33: bestAlt := findMin(altList);
34: DIF_int := integrate(bestAlt);
```

2. altList: A list of all alternative solutions ending up in a partial or full overlapping of two data flows (referent and new), together with the estimated costs of such consolidation solution.
   $altList$ ::=[matchDIFPair,cost],$altList$| [$matchDIFPair$, cost];

Each element of matchQ and altList contains information of integrated data flows, i.e.,
$matchDIFPair$ ::= [DIF_{ref}, DIF_{new}, $allMatches$, $lastMatches$];

- A pair of data flows (DIF_{ref} and DIF_{new}), potentially reordered for such integration.
- Pairs of pointers to all matched operations (allMatches), with information about the matching type and integration (intInfo).
  $allMatches$ ::= [$matchOpsPair$, intInfo], $allMatches$|
  [$matchOpsPair$, intInfo];
- A queue with pairs of pointers to the last matched operations (lastMatches), and an out-edge in a referent graph ($edge_{ref}$).
  $lastMatches$ ::= [$matchOpsPair$, $edge_{ref}$], $lastMatches$|
  [$matchOpsPair$, $edge_{ref}$];

$matchOpsPair$ ::=[o_{ref}, o_{new}];

CoAl first initializes the list of alternative solutions by adding the alternative with no integration of DIF_{new} and DIF_{ref}, together with the cost of having these two data flows separately, i.e., $cost_{noInt}$ (Step 1).

CoAl then starts by searching for the matching leaf (source) nodes of new and referent data flows (i.e., matchLeafs, Step 2). The source data stores are compared based on their main characteristics, i.e., source type, source name or location, and extracted attributes. CoAl initializes matchQ with the pair of the referent and new data flows, together with the found matching pairs of source data stores (i.e., initially both allMatches and lastMatches).

Example. When integrating DIF-3 (Figure 3.6), into the referent flow, i.e., DIF-12 (Figure 3.4), we first identify common source nodes of the two data flow graphs (i.e., `orders` and `lineitem`). □

The four phases of CoAl are as follows:

i) Search for the next operations to match. At each iteration, we consider extending a single pair of currently overlapping data flows from the priority queue (matchQ) with a new pair of matching operations. For a dequeued pair of data flows (i.e., dequeue, Step 4), we identify the operations in these flows to be compared next, starting from a pair of previously identified full matches (i.e., o_{ref} and o_{new} dequeued from lastMatches; dequeue, Step 5). Finding operations to be compared next in both data flows is performed by means of two calls to the function FindNextForMatch (i.e., Algorithm 2)

Algorithm 2 FindNextForMatch

```
inputs: DIF, o_cur, out-edge; output: nextOps
 1: nextOps := ∅;
 2: for all path ∈ findPathsToForks(DIF, o_cur, out-edge) do
 3:     for i:=1 to length(path) do
 4:         if canReorder(path,i) ∧ fulfillsI2(path[i]) then
 5:             insert(nextOps,path[i]);
 6:         end if
 7:     end for
 8: end for
 9: return nextOps;
```

in steps 6 and 7). In FindNextForMatch we apply the generic equivalence rules explained in Section 3.1, and find the operations that can be pushed down towards the last fully matched operations, thus fulfilling I_2 (i.e., canReorder and fulfillI2, Step 4 in Algorithm 2). Notice that we search until we reach the operation that has multiple outputs (i.e., findPathsToForks, Step 2), since swapping operations down a fork would affect the semantics of other branches in a data flow.
Example. For the fully matching nodes `orders`, of DIF-12 and DIF-3, we find the following sets of operations to be compared next: $orders_{DIF-12}$ = {`Filter1`, `Join2`}; $orders_{DIF-3}$ = {`Filter1`, `Join3`, `Join4`}. □

ii) Compare the next operations. CoAl then compares each pair of operations from the previously produced sets (i.e., $nextOps_{ref}$ and $nextOps_{new}$), using the comparison rules discussed in Section 3.2. Depending on the result, it identifies: (a) a full match, $o_{ref} \equiv o_{new}$ (Step 2a); (b) a partial match, $o_{new} \sqsubseteq o_{ref} \vee o_{ref} \sqsubseteq o_{new}$ (Step 16) or (c) no match, $o_{ref} \between o_{new}$.
Example. From the two sets of operations that can be compared next, we find two full matches between `Filter1`(DIF-12) and `Filter1`(DIF-3), and `Join2`(DIF-12) and `Join4`(DIF-3). □

It may also happen that no matching is found for any pair of operations (Step 25).

iii) Reorder the input data flows. If CoAl finds a (full or partial) match, it proceeds (if needed) with operation reordering to align the input data flows and enable integration of the previously matched operations, i.e., to satisfy I_2, (i.e., reorder, steps 10 and 11).
Example. Following the above example, for the full match of the Filter operations in DIF-12 and DIF-3, no additional operation reordering is necessary and CoAl directly adds `Filter1` to the current maximum overlapping area (i.e., I_2 is satisfied). But, for the full match between `Join2`(DIF-12) and `Join4`(DIF-3), CoAl must perform operation reordering (i.e., (re-)associate `Join4` down `Join3` in DIF-3), so that the `Join4` operation could be matched next (I_2). □

CoAl then extends the overlapping of input data flows with matching pair of operations to allMatches, together with their integration information (i.e., insert, Step 12). Next, based on the type of the previously found match, CoAl proceeds as follows:

- For a full match, it enqueues back to priority queue the two data flows (possibly reordered) to further extend the matching in the next iterations (i.e., enqueue, Step 15), starting from the last matched pair of operations added (i.e., lastMatches). Notice that CoAl needs to enable the search in all possible output branches of the referent data flow, thus we enqueue back the currently matched operations once for each of the next out-edges to be followed from the previously matched operations (see Step 14).
- For a partial match, if there are no other previously matched operations from which it can extend matching (Step 17), CoAl estimates the cost of the current solution and inserts it, along with its cost, to the list of alternatives (Step 18). Otherwise, it enqueues back to matchQ the two data flows to further extend the matching in other branches. (i.e., enqueue, Step 20).

Similarly, if no match is found (Step 25), this solution along with its estimated cost is also added to the list of potential alternatives only if it is not possible to further extend the matching.

Example. In the given example (i.e., DIF-3 and DIF-12), this occurs after we match the join operations (i.e., `Join2` from DIF-12 and `Join4` from DIF-3). Going further in data flows DIF-12 and DIF-3, we cannot find any matching operation that can enlarge the common areas of these two data flows. Thus, we add the currently matched data flows as an alternative solution, resulting in an integrated multi-flow branching after the matched join operation (i.e., `Join3;{1-4}` in Figure 3.7) □

Otherwise, CoAl continues matching in other branches (i.e., Step 29).

The algorithm finishes the matching process when all operations of input data flows are explored and compared (i.e., no more elements in the matchQ).

iv) Integrate an alternative solution. After all iterations finish, CoAl analyzes the list of the found alternatives, looks for the one with the lowest estimated cost (i.e., findMin, Step 33), and integrates it using the integration information (i.e., integrate, Step 34).

Finally, CoAl returns the integrated multi-flow (i.e., DIF_{int}).

4.1 Computational complexity

To integrate a referent ($DIF_{ref} = (V_{ref}, E_{ref})$) and a new ($DIF_{new} = (V_{new}, E_{new}$)) data flow, the CoAl algorithm at first glance indicates at worst quadratic complexity (in terms of $|V_{ref}| \cdot |V_{new}|$), due to the Cartesian product of operations that can be compared next (see Step 8). However, there are several characteristics that either directly from the invariants of the CoAl algorithm or from empirical experiences show that this is not a realistic upper bound of the algorithm.

- Under the assumption that input data flows are compact in terms that they do not have redundant operations (i.e., operations of a single flow that can be fully matched; see Section 3.2), it is impossible that multiple alternative paths branching from a single operation in a multi-flow completely match with a path of another data flow.
- The search is led by the size of paths in the new data flow (i.e., $|p_{new}|^{avg}$), which is typically shorter than the paths in the referent data flow.
- When searching next operations to compare (i.e., Algorithm 2), due to conflicting dependencies among operations (see Section 3.1), it is also unrealistic that all the operations in the paths of encompassing requirement subgraphs can be reordered to be compared next, which drastically reduces the actual number of comparisons inside the loop.

We further analyze the complexity of the CoAl algorithm based on the search space of the algorithm while looking for the next operations to match. That is, we take into account the number of the main loop iterations (see Step 3 in Algorithm 1), and for each loop, the number of operations searched to be compared next in each loop (see Algorithm 2). While the cost of Algorithm 2 for a new data flow in bounded by the maximal size of new data flow graphs, the cost for a referent one grows iteratively as the size of the data flow graph grows, hence we take the latter one into account when estimating the complexity of the CoAl algorithm. Additionally, notice that the number of the main loop iterations (Step 3) depends on the number of elements previously enqueued to the matchQ and lastMatched, which occurs only when we find a full match between two data stores or two operations (see steps 2 and 2a in Algorithm 1). In the worst case for the complexity, we can find a full match for all operations in a path of a new data flow, i.e., the path is completely subsumed by the referent flow.

We start by estimating the number of iterations of the main loop. The complexity for a single path of new data flow (i.e., p_{new}) can be obtained as follows:

$c(p_{ref} \cdot_{int} p_{new}) = \sum_{i=1}^{|p_{new}|} deg(o_{ref_i})$ □

If we consider an average outdegree of a referent data flow graph (i.e., $deg^{avg}(DIF_{ref})$):

$c(p_{ref} \cdot_{int} p_{new}) = \sum_{i=1}^{|p_{new}|} deg(o_{ref_i}) = |p_{new}|^{avg} \cdot deg^{avg}(DIF_{ref})$ □

Furthermore, CoAl performs such search for all paths starting from the previously matched source data stores, i.e., maximally for $|\mathrm{DS}_{new_S}| \cdot |\mathrm{DS}_{ref_S}|$ paths.

$c_1(DIF_{ref} \cdot_{int} DIF_{new}) = c_1 =$

$= |\mathrm{DS}_{new_S}| \cdot |\mathrm{DS}_{ref_S}| + |\mathrm{DS}_{new_S}| \cdot |p_{new}|^{avg} \cdot deg^{avg}(DIF_{ref})$ □

Using graph theory, we can further represent the average outdegree of a directed graph (i.e., $DIF_{ref} = (V_{ref}, E_{ref})$) as $\frac{|E_{ref}|}{|V_{ref}|}$. At the same time, the average length of a source-to-target path $|p_{new}|^{avg}$ can be obtained as the average depth of a new graph. Assuming that a graph resulting from a single information requirement is a tree, and the tree is balanced, we can obtain its depth as $\log_{\frac{|E_{new}|}{|V_{new}|-|\mathrm{DS}_{new_S}|}} |DS_{new_S}|$. That is:

$c_1 = |\mathrm{DS}_{new_S}| \cdot |\mathrm{DS}_{ref_S}| + |\mathrm{DS}_{new_S}| \cdot (\log_{\frac{|E_{new}|}{|V_{new}|-|\mathrm{DS}_{new_S}|}} |\mathrm{DS}_{new_S}|) \cdot \frac{|E_{ref}|}{|V_{ref}|}$ □

Next, for each loop, we estimate the complexity of searching for the next operations to compare in the referent data flow graph. Starting from a pair of last matched operations, we search in all paths, and identify the next operations that are candidates to be reordered and compared next (see Algorithm 2). Such search in general resembles the tree traversal with the last matched operation as a root and target data stores as leaf nodes. We estimate the size of such tree and thus the complexity of its traversal with the average depth of a tree (i.e., the average length of a source-to-target path), multiplied by the average outdegree of the graph. Again, based on the graph theory, we can express such estimations in terms of the size of a referent data flow graph. That is:

$c_2 = depth^{avg}(DIF_{ref}) \cdot deg^{avg}(DIF_{ref}) = (\log_{\frac{|E_{ref}|}{|V_{ref}|-|\mathrm{DS}_{ref_R}|}} |\mathrm{DS}_{ref_S}|) \cdot \frac{|E_{ref}|}{|V_{ref}|}$ □

Thus, we estimate the complexity of the algorithm as:

$c_{int} = c_1 \cdot c_2 =$

$= (|\mathrm{DS}_{new_S}| \cdot |\mathrm{DS}_{ref_S}| + |\mathrm{DS}_{new_S}| \cdot (\log_{\frac{|E_{new}|}{|V_{new}|-|\mathrm{DS}_{new_S}|}} |\mathrm{DS}_{new_S}|) \cdot \frac{|E_{ref}|}{|V_{ref}|}) \cdot$

$\cdot ((\log_{\frac{|E_{ref}|}{|V_{ref}|-|\mathrm{DS}_{ref_R}|}} |\mathrm{DS}_{ref_S}|) \cdot \frac{|E_{ref}|}{|V_{ref}|})$ □

Assuming that the average size (i.e.,$|E_{new}|$, $|V_{new}|$) and the number of source data stores (i.e., $|\mathrm{DS}_{new_S}|$) in a new data flow is constant, the theoretical complexity for the problem of integrating data flows can be given as a function of the size of the referent data flow (i.e., $|E_{ref}|$), the number of its data sources (i.e., $|\mathrm{DS}_{ref_S}|$), and its average outdegree (i.e., $\frac{|E_{ref}|}{|V_{ref}|}$).

5 Evaluation

5.1 Prototype

CoAl works at the logical level and integrates data flows coming from either high level business requirements (e.g., [146]) or platform-specific programs (e.g., queries, scripts, ETL tool metadata; [93]). We built a prototype that implements the CoAl algorithm. The prototype is integrated into a larger ecosystem for the design and deployment of data flows from information requirements [90]. Communication with different external modules of the ecosystem for import/export of data flows is enabled using a platform-independent representation of a data flow, namely xLM (i.e., XML encoding of data flow metadata [160]). Data flows in other languages could be translated to/from xLM using external tools (e.g., [92]).

5.2 Experimental setup

We selected a set of 15 data flows, translated from the referent TPC-H benchmark queries[3]. Notice that even though the TPC-H benchmark provides a relatively small set of input queries, such set covers different sizes and complexities of input data flows and suffices to demonstrate the functionality of the CoAl algorithm. Thus the obtained results are generalizable to other inputs. CoAl's flexibility to deal with different complexities of data flow operations is previously showed in sections 2 and 3. Considered data flows (similar to those presented in Section 2), span from only 4 to the maximum of 20 operations, performing various data transformations, i.e., filters, joins, projections, aggregations, user defined functions. More information about the selected queries can be found in the TPC-H specification [5]. We translated the selected SQL queries, into the platform-independent representation that CoAl understands (i.e., xLM).

To cover a variety of input scenarios (i.e., different orders in which input data flows are provided), we have considered different permutations of incoming data flows. Since the total number of different permutations for the chosen 15 queries is not tractable (i.e., 15!, > 1307 billions), we have randomly sampled, a uniformly distributed set of 1000 permutations and obtained the average values of the observed numbers. For each permutation, we have simulated the incremental arrival of input data flows, starting by integrating the first 2 data flows, and then incrementally adding the other 13.

[3]Selected TPC-H queries: q1, q2, q3, q4, q5, q6, q9, q10, q11, q13, q16, q17, q18, q19, q21. Other queries are discarded due to limitations of the available external SQL translation module.

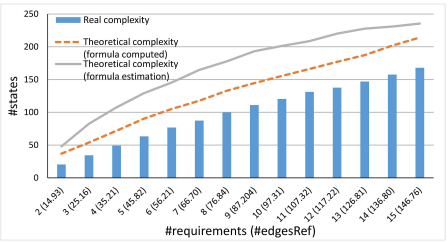


Fig. 3.9: Search space exploration

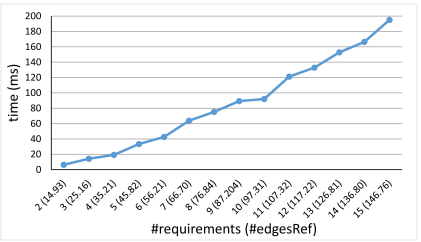


Fig. 3.10: CoAl's execution time

5.3 Scrutinizing CoAl

We first analyzed the distribution of the values obtained in the considered sample of permutations. For all of them we confirmed a positive (right) skew, which indicated the stability of our data flow integration algorithm. Thus, in the reminder of this section, we report the mean values obtained from the considered permutations.

Search Space. As shown in Figure 3.9 (*Real complexity*), the search space (i.e., #states refers to the complexity in Section 4.1) grows linearly with the number of input data flows. For input flows of an average size of 15 operations, the number of states considered starts from only several when integrating 2 data flows and go up to the maximum of 170 states when integrating the 15^{th} data flow (Figure 3.9 reports the average values).

We additionally estimated the theoretical computational complexity of CoAl for given inputs, following the formula in Section 4.1 and compared it to the obtained real values (see Theoretical complexity (formula estimated) in Figure 3.9). We observed that the overestimation and a slight deviation of the tendency of the formula estimated complexity comes from the generalizations adopted for estimating the average depth and the degree of a data flow graph. Fitting the formula with the average values directly computed in the execution (see Theoretical complexity (formula computed) in Figure 3.9), showed the co-

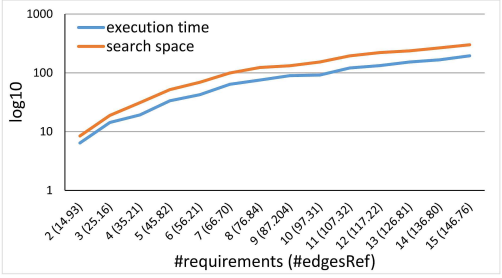


Fig. 3.11: Space/time correlation

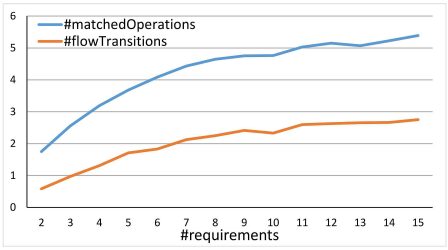


Fig. 3.12: CoAl characteristics

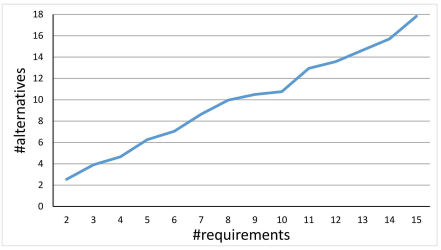


Fig. 3.13: Alternative solutions

inciding tendency with the real complexity and smaller overestimation resulted from the averaged values.

On the other side, following the complexity discussion in Section 4.1, we analyzed the time needed to complete the search in terms of the size of the referent data flow (i.e., number of edges). This analysis showed that the time also grows linearly following the size of a referent data flow (see Figure 3.10), starting from only 6.4 ms with the initial size of a referent data flow (i.e., 15 edges), and going up to the maximum of 195ms when integrating the fifteenth data flow over the referent data flow with 147 edges. These values showed a very low overhead of CoAl, making it suitable for today's right-time BI applications. Moreover, such results further show CoAl's scalability for larger input complexities. We additionally analyzed the correlation of the time and the search space (see Figure 3.11), and showed that the growth of execution time follows the same (linear) trend as the complexity growth, which validated our complexity estimations discussed in Section 4.1.

Algorithm characteristics. We also studied the behavior of CoAl internal characteristics. Figure 3.12 shows how the number of matched operations (#matchedOperations) and the number of flow transitions (#flowTransitions), related to the I_2 invariant, are affected by the size of the problem. The average number of matches increases from 1 to 4 (excluding the matched data sources), until the sixth integrated data flow, and then this trend slows only up to 5 matched for the following data flows. This happens because an integrated flow may impose branching (multiple outputs) for the subflows shared among the input data flows (see Section 4). Such behavior restricts the operation reorderings from one branch down the fork, as it would change the semantics of the shared subflow, and thus of all the dependent branches.

This trend is also confirmed by the number of different flow transitions (i.e., number of different operation reorderings; see Figure 3.12). This further showed that in the basic integration scenario different orders of incoming data flows might produce different integration solution (although all of them will be semantically equivalent). Notice, however, that tracing the metadata of original data flows and all integration alternatives (whose number grows linearly with

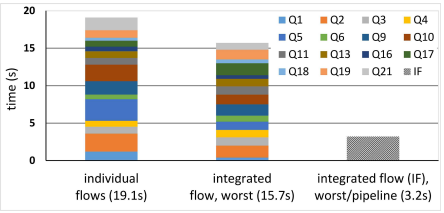


Fig. 3.14: Performance gains (worst integration case)

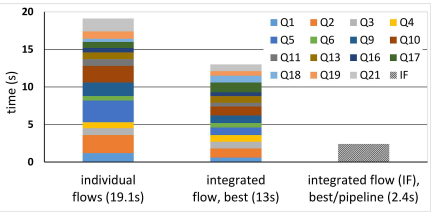


Fig. 3.15: Performance gains (best integration case)

the size of the problem; see Figure 3.13), would facilitate the maintenance of integrated multi-flows and the revision of some integration choices.

Improvement in the overall execution time. Additionally, by reviewing the integrated multi-flows for the considered sample of order permutations, we have identified a certain variation of the result characteristics (i.e., a relative standard deviation of the output size is around 20%), and thus we isolated two permutations whose outputs we further analyzed, i.e., (1) the best case - among the considered permutations, the permutation that produces the largest overlapping (i.e., the most matched operations) between the input data flows, and (2) the worst case - among the considered permutations, the permutation that produces the smallest overlapping (i.e., the least number of matched operations) between the input data flows. For these two cases, we analyzed the execution time of the multi-flow after integrating all 15 data flows from the input, and compared it with the total execution time for the 15 individually executed data flows. Notice that we do not present here optimal solutions in terms of performance, but rather analyze how different degrees of data and transformation reuse affect the overall execution time of a data-intensive multi-flow. For these experiments we ran data flows in Pentaho Data Integration tool using a dataset of 10k to 20k tuples generated from the TPC-H data generator.

The results are illustrated in Figures 3.14 and 3.15 for the worst and the best overlapping case, respectively. We first individually executed the data flows from the input, and observed that it took 19.1s in total to execute 15 data flows with the maximum of 2.4s for executing the data flow of Q2 from TPC-H. We further executed the integrated solutions of the best and the worst case, as explained above. Notice in Figures 3.14 and 3.15 that some data flows are penalized by the integration (e.g., Q17), i.e., their execution time increased due to unfavorable reordering of more selective operations (e.g., filters over joins) to achieve larger overlapping with the referent data flow. Conversely, some larger data flows (e.g., Q2 and Q5) largely benefit from the integration by reusing already performed data processing of other data flows. Note that in both cases the overall execution time of the integrated multi-flow decreased. The best case solution (see Figure 3.15) took 13s for the overall execution,

whilst the worst case solution (see Figure 3.14) took 15.7s. We thus observed approx. 31.9% of improvement of the overall execution time for the best and 17.8% for the worst case, which finally confirms our initial assumptions.

We additionally confirmed that the improvement of the overall execution time is correlated with the amount of overlapping (i.e., number of shared data and operations). For instance, the improvement in the overlapping size from the worst to the best integration case discussed above (i.e., 51 in the worst to 79 in the best case) showed to be approx. proportional to the improvement of the overall execution time for these two cases.

Furthermore, integration of multiple data flows enabled extra optimization inside the considered execution tool, by allowing the pipelined execution of the unified multi-flow. This finally resulted with 2.4s of the overall execution pipeline for the best case, and 3.2s for the worst case. Notice that we report these results for showing one of the benefits of integrating different data flows (see shaded bars in Figures 3.14 and 3.15), while for the fair comparison we used the results not taking into account the advantage of the enabled pipeline execution.

6 Related Work

From the early years, the reuse in data integration field has been encouraged and envisioned as beneficial [148], since organizations typically perform data integration efforts that involve overlapping sets of data and code. However, such problem has also been characterized as challenging due to inherently complex enterprise environments. Different guidelines and approaches have been proposed to tackle this issue in various scenarios.

Schema mapping management. The data exchange and data integration problems set the theoretical underpinnings of the complexity of what we today call an ETL process [186]. Schema mappings, as a set of logical assertions relating the elements of source and target schemata, play a fundamental role in both data exchange [52] and data integration [106] settings. Intuitively, schema mappings are predecessors to more complex ETL transformations [186]. Various approaches and tools dealt with automating schema mapping creation (e.g., [52, 44]), while others further proposed high-level operations over the set of mappings, i.e., composition, inversion, and merge, (e.g., [16]). We find the problems of composing and merging schema mappings especially interesting for the context of data flow consolidation in terms of reducing information redundancy and minimality. The remarks of these works motivated our research, but moving towards more complex scenarios of today's BI introduced new challenges both regarding the complexity of schema mappings (e.g., grouping, aggregation, or "black-box" operations) and the diversity of data sources that these approaches could not support (i.e., only relational or XML data

formats have been previously considered). Conversely, we propose generic solutions for both operation reordering and operation comparison challenges that solve the problem for an arbitrary set of data flow operations.

Workflow optimization. Equivalence rules used in data flow optimization can be conveniently applied in the context of consolidating data flows to maximize the data and operation sharing (see Section 3.1). Both traditional query optimization [124] and multi-query optimization approaches [94] focus on performance and consider a different subset of operations than those typically encountered in complex data flows (e.g., operations with "black-box" semantics). Recently, more attention has been given to solving the data flow optimization problem for a generic set of complex operations (e.g., [158, 78]). In the former work, the problem of ETL optimization has been modeled as a state-space search problem [158], with a set of generic equivalence transitions used to generate new (eventually optimal) states. Such equivalence transitions inspired those presented in Section 3.1 (i.e., swap, factorize/distribute, merge/split), but state generation is based solely on the information about the schemata used and produced by ETL operations. We propose a less conservative approach where we distinguish three different properties of data flow operations (i.e., schema, value, and order) and thus we are able to detect more promising reordering (optimization) opportunities. On the other side, the later approach [78] does consider the attributes' value as an important operation property, but overlooks the dataset order. The main reason is that the approach focuses solely on the set of second order operations written in an imperative language for a big data analytics system (e.g., Map, Reduce, Cross, etc.).

Recent optimization approaches (e.g., [64]) discuss the problem of finding the optimal global query plan for a set of input queries by means of data and operation sharing. Another approach considers two non-orthogonal challenges when looking for the optimal data flow design: operation sharing and reordering [64]. However, unlike CoAl, this approach focuses mainly on the tradeoffs of using these two approaches in the context of data flow optimization and does not study how operation reordering can enhance and maximize data and operation sharing among different data flows. Moreover, these approaches are limited to the typical relational algebra operators, while CoAl provides a generic framework for the comparison and reordering of arbitrary data flow operations (see Section 2).

Data flow design. The modeling and design of data-intensive flows is a thoroughly studied area, both in the academia [13, 189, 190] and industry, where many tools available in the market often provide overlapping functionalities for the design and execution of these flows. However, neither the research nor the available tools provide the means for automatically adapting data flow designs to changing information requirements.

To the best of our knowledge, the only work tackling the integration of ETL processes is in Albrecht and Naumann [14]. The authors propose a set of

high level operators for managing the repository of ETL processes. However, the work lacks the formal definition and automatic means for such operators. Additionally, the authors do not consider the incremental consolidation of data flows led by information requirements.

7 Conclusions and Future Work

We have presented CoAl, our approach to facilitate the incremental consolidation of data-intensive multi-flows. CoAl starts from data flows that satisfy single information requirements. Iteratively, CoAl identifies different possibilities for integrating new data flows into the existing multi-flow, focusing on the maximal data flow reuse. Finally, CoAl suggests a unified data flow design evaluating it with the user-specified cost model.

We have developed a prototype that implements the complete functionality of CoAl. We used it to evaluate the efficiency, scalability, and the quality of the output solutions of our approach. We also report the improvement of the overall execution time as well as other benefits of integrated multi-flows.

The final goal of our overall work is to provide an end-to-end platform for self-managing the complete lifecycle of BI solutions, from information requirements to deployment and execution of data-intensive flows [90].

8 Acknowledgments

This work has been partially supported by the Secreteria d'Universitats i Recerca de la Generalitat de Catalunya under 2014 SGR 1534, and by the Spanish Ministry of Education grant FPU12/04915.

Chapter 4

A Requirement-Driven Approach to the Design and Evolution of Data Warehouses

The paper has been published in the
Journal of Information Systems, Volume 44: pp. 94-119 (2014). The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1016/j.is.2014.01.004

Abstract

Designing data warehouse (DW) systems in highly dynamic enterprise environments is not an easy task. At each moment, the multidimensional (MD) schema needs to satisfy the set of information requirements posed by the business users. At the same time, the diversity and heterogeneity of the data sources need to be considered in order to properly retrieve needed data. Frequent arrival of new business needs requires that the system is adaptable to changes. To cope with such an inevitable complexity (both at the beginning of the design process and when potential evolution events occur), in this chapter we present a semi-automatic method called ORE, for creating DW designs in an iterative fashion based on a given set of information requirements. Requirements are

first considered separately. For each requirement, ORE expects the set of possible MD interpretations of the source data needed for that requirement (in a form similar to an MD schema). Incrementally, we build the unified MD schema that satisfies the entire set of requirements and meet some predefined quality objectives. We have implemented ORE and performed a number of experiments to study our approach. We have also conducted a limited-scale case study to investigate its usefulness to designers.

1 Introduction

Data warehousing ecosystems have been widely recognized to successfully support strategic decision making in complex business environments. One of their most important goals is to capture the relevant organization's data provided through different sources and in various formats with the purpose of enabling analytical processing of such data. The most common design approach suggests building a centralized decision support repository (like a DW) that gathers the organization's data and which, due to its analytical nature, follows a multidimensional (MD) design. The MD design is distinguished by the fact/dimension dichotomy, where facts represent the subjects of analysis and dimensions show different perspectives from which the subjects can be analyzed (e.g., we can analyze shopping orders for customers and/or suppliers). Furthermore, the design of the extract-transform-load (ETL) processes responsible for managing the data flow from the sources towards the DW constructs, must also be considered.

Complex business plans and dynamic, evolving enterprise environments often result in a continuous flow of new information requirements that may further require new analytical perspectives or new data to be analyzed. Due to the dynamic nature of the DW ecosystem, building the complete DW design at once is not practical. Also, assuming that all information and business requirements are available from the beginning and remain intact is not realistic either. At the same time, for constructing a DW design (i.e., its MD schema) the heterogeneity and relations among existing data sources need to be considered as well.

The complexity of the monolithic approach for building a DW satisfying all information requirements has also been largely characterized in the literature as a stumbling stone in DW projects (e.g., see [97]). As a solution to this problem, a step-by-step approach for building a DW has been proposed in [97] (a.k.a. Data Warehouse Bus Architecture). This approach starts from data marts (DM) defined for individual business processes and continues exploring the common dimensional structures, which these DMs may possibly share. To facilitate this process, a matrix as the one shown in Table 4.1 is used, which relates DMs (i.e., their subsumed business requirements) to facts and dimen-

Table 4.1: The DW Bus Architecture for IR1-IR5

	Customer	Supplier	Nation	Region	Orders	Part	Partsupp	Lineitem_dim
ship. qty.(IR1)	√	√	√		√		√	
profit(IR2)		√	√					√
revenue(IR3)		√	√	√		√	√	
avil. stock val.(IR4)		√	√			√		
ship. prior.(IR5)	√				√			√

sions implied by each DM. Such matrix is used for detecting how dimensions (in columns) are shared among facts of different DMs (in rows), i.e., if a fact of a DM in the row x is analyzed from a dimension of the column y, there is a tick in the intersection of x and y. The content of the matrix in Table 4.1 follows our running example based on the TPC-H benchmark [5], which is introduced in more detail in Section 2. We consider five information requirements (i.e., IR1-IR5) and for each of them a single DM. Each requirement analyzes some factual data (e.g., `IR3` analyzes the `revenue`), from different perspectives (e.g., `revenue` is analyzed in terms of parts –i.e., `partsupplier-part` hierarchy– , and supplier's region –i.e., `supplier-nation-region` hierarchy–). Finally, based on this matrix, different DMs are combined into an MD schema of a DW. However, such design guidelines still assume a tremendous manual effort from the DW architect and hence, DW experts still encounter the burdensome and time-lasting problem of translating the end-user's information requirements into the appropriate MD schema design.

Automating such process has several benefits. On the one hand, it supports the complex and time-consuming task of designing the DW schema. On the other hand, results automatically produced guarantee that the MD integrity constraints [114] are met as well as some DW quality objectives used to guide the process [147]. Accordingly, several works tried to automate the process of generating MD schemas (e.g., [168, 131, 142]). However, for the sake of automation, these approaches tend to overlook the importance of information requirements and focus mostly on the underlying data sources. Such practices require an additional manual work in conforming the automatically produced MD designs with the actual user requirements, which often does not scale well for complex scenarios. For example, it has been shown that even for smaller data source sizes the number of potential stars produced by means of a blind search of MD patterns over the sources is huge [142]. Consequently, it is not

feasible to assume that the DW architect will be able to prune and filter such results manually.

To the best of our knowledge only three works went further and considered integrating the information requirements in their semi-automatic approaches for generating MD models [69, 115, 146]. At different levels of detail, these approaches support transforming every single requirement into an MD model to answer such requirement. However, how to integrate such individual MD models into a single, compact MD view is left to be done manually (although [69, 115] introduce strict manual guidelines in their approaches to assist the designer). Our experiments, described in Section 5, have shown that integrating MD requirements, is not an easy task and this process must also be supported by semi-automatic tools.

In this chapter, we present a semi-automatic method for Ontology-based data warehouse REquirement-driven evolution and integration (ORE). ORE complements the existing methods (e.g., [168, 131, 142]) and assists on semi-automatically integrating partial MD schemas (each representing a requirement or a set of requirements) into a unified MD schema design. Moreover, ORE could also be used to integrate existing MD schemas of any kind (e.g., as in [68]). ORE starts from a set of MD interpretations (MDIs) of individual requirements (which resemble the rows of Table 4.1). Intuitively, an MDI is an MD characterization of the sources that satisfies the requirement at hand (see Section 2.2 for further details). Iteratively, ORE integrates MDIs into a single MD schema which satisfies all requirements so far. Importantly, ORE generates MD-compliant results (i.e., fulfilling the MD integrity constraints) and determines the best integration options according to a set of quality objectives, to be defined by the DW designer. To guarantee so, ORE systematically traces valuable metadata from each integration iteration. In all this process, the role of the data sources is crucial and ORErequires a characterization of the data sources in terms of a domain ontology, from where to automatically explore relationships between concepts (e.g., synonyms, functional dependencies, taxonomies, etc.) by means of reasoning.

Our method, ORE, is useful for the early stages of a DW project, where we need to create an MD schema design from scratch, but it can also serve during the entire DW lifecycle to accommodate potential evolution events. As we discuss later on, in the presence of a new requirement, our method does not create an MD design from scratch, rather it can automatically absorb the new requirement and integrate it with the existing MD schema so far.

Contributions. The main contributions of our work are as follows.

- We present a semi-automatic approach, ORE, which, in an iterative fashion, deals with the problem of designing a unified MD schema from a set of information requirements.
- We introduce novel algorithms for integrating MD schemata, each sat-

isfying one or more requirements. Results produced are guaranteed to subsume all requirements so far, preserve the MD integrity constraints, and meet the user defined quality objectives.

- We introduce the traceability metadata structure to systematically record information about the current integration opportunities both for finding the best integration solution w.r.t. the chosen quality objectives and linking the final MD schema to the data sources by means of ETL processes.
- We experimentally evaluate our approach by using a prototype. A set of empirical tests have been performed to assess the output correctness, a characterization of ORE's internal algorithms and the manual effort for providing an integrated MD design from several requirements and in turn, demonstrate the need for automated design approaches like ORE.

Outline. The rest of this chapter is structured as follows. Section 2 presents an abstract overview of our approach through an example case based on the TPC-H schema [5]. Sections 3 and 4 formally present the main structures and stages our method goes through for integrating new information requirements into a unified MD schema. In Section 5 we provide the theoretical validation of our approach, while Section 5 presents the experimental evaluation of ORE (both of its internals and from the end-users perspective). Section 6 discusses the related work, while Section 8 concludes the chapter and discusses our future directions.

2 Overview of our Approach

In this section, we first introduce an example case based on the TPC-H schema and formalize the notation used throughout the chapter. Then, we present the overview of our solution to semi-automatically create an MD schema from a set of information requirements.

2.1 Running example

Our example scenario is based on the schema provided by the TPC-H benchmark [5]. The abstraction of the schema is illustrated in Figure 4.1. The TPC-H is a decision support benchmark which, besides the schema, also provides a set of business oriented queries (from here on, information requirements - IR). For the sake of our example, let us assume a subset of five information requirements, which are the following:

- IR1: The total quantity of the parts shipped from Spanish suppliers to French customers.

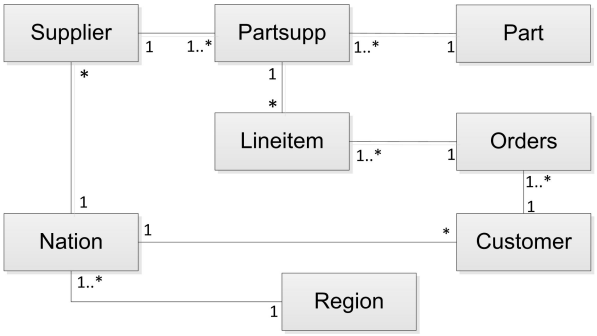


Fig. 4.1: TPC-H Schema

- IR2: For each nation, the profit for all supplied parts, shipped after 01/01/2011.
- IR3: The total revenue of the parts supplied from East Europe.
- IR4: For German suppliers, the total available stock value of supplied parts.
- IR5: Shipping priority and total potential revenue of the parts ordered before certain date and shipped after certain date to a customer of a given segment.

2.2 Formalizing Information Requirements

In this section, we describe how information requirements are formalized in order to be automatically processed. Requirements posed during the DW project lifecycle differ from usual user requirements in other software projects in that they typically have analytical flavor. For this reason, it has been previously discussed that information requirements can be elegantly represented in terms of the MD model as data cubes (e.g., see [69, 145]), and they are usually elicited with natural language template similar to the following one: "I want to analyze `fact` (e.g., revenue or profit) from `dimensions` (e.g., time, customer) where `descriptors` (e.g., previous year)". This is a straightforward enunciation of the cube-query metaphor [97] in natural language, which distinguishes two basic kinds of concepts, namely dimensional and factual concepts.

Thus, from an information requirement we can extract two kinds of information: (1) which data the user wants to analyze (i.e., factual data) and (2) from which perspectives (i.e., the multidimensional space conformed by the dimensions of analysis), which may contain descriptors (i.e., dimension attributes). We further formalize the data cube in terms of a grammar, which

can be conveniently used to serialize the content such as textual requirements or MD interpretations in a machine readable form (e.g., XML, JSON, etc.).

In the domain that is being analyzed, we can identify the concepts that retrieve the data asked by the requirement, i.e., `domain_concept`.

`Fact` (`F`) is the core MD concept in the domain and represents the focus of the analysis. It usually contains a set of numerical measures (`M`), e.g., revenue, quantity, extended price, which can be analyzed and aggregated from different perspectives (dimensions), using the appropriate aggregation functions (`AggFunc`), e.g., `SUM`, `AVERAGE`, `MAX`. Additionally, numerical measures can be derived by applying arithmetic operations (`ArithmOp`) or built-in functions (`DFunc`) over basic measures. Formally:

```
DFunc := 'sqrt'|'abs'|...;
AggFunc := 'SUM'|'AVG'|'COUNT'|'MEDIAN'|...;
ArithmOp := '+'|'-'|'/'|'x'|...;
Expr := Expr,ArithmOp,M|M;
M := DFunc(M)|Expr|< domain_concept >;
F := AggFunc(F)|M;
```

The multidimensional space (`S`) is determined by the analysis dimensions (i.e., $S = D_1 \times ... \times D_n$). Each `dimension` (`D`) represents a perspective from which we analyze facts. In general, a single dimension consists of a hierarchy of partially ordered set ($<$) of `levels` (`L`). In a data cube, dimensions appear at a given level of detail (a.k.a. dimension level). Note that a meaningful cube can only refer to a single level per dimension. A function (`EFunc`) can be applied on levels to extract additional or normalized information -e.g., month('2001-09-28 01:00:00') would return '09'. Additionally, descriptors (`Dsc`) may be defined over the dimension levels and used inside of logic predicates, -e.g., 'year = 2012'-. Note that the `literal` in the following grammar rules stands for any numerical or textual constant value. Formally:

```
Oper := '>'|'<'|'>='|'<='|'='|'!='|'IN'|'MATCHES'|...;
EFunc := 'day'|'month'|...;
Dsc := Operand, Oper, Operand;
Operand := L|< literal >;
L : = < domain_concept >|EFunc(L)|Dsc;
D := D≺L|L
S := S×D|D
```

Given the above formalizations, we represent an information requirement or data cube (`IR`) as a fact (`F`) and (at least) one dimension (usually a list of dimensions), conforming its multidimensional space. Formally:

```
IR := F,S;
```

Various approaches have tackled the issue of modeling the MD data cube for each requirement at hand. Basically, they identify the needed data sources' subset that retrieves the data to answer each IR (intuitively a query) and map it into an MD interpretation (i.e., assign MD roles to each concept in the subset) to guarantee its compliance with the MD model. Eventually, they conform MD schemas answering all the requirements. Each of these methods makes different assumptions and achieves different degrees of automation but, to our knowledge, only three of them largely automate the process of identifying, for each requirement, the MD cube that answers such requirement [69, 115, 146]. However, none of these provides automatic means to integrate several requirements into a single MD schema.

ORE starts from the MD knowledge extracted from each requirement (e.g., by means of any of the previously mentioned MD schema design approaches) and aims to incrementally derive a unified MD schema satisfying the entire set of requirements.

For the purpose of formalizing our approach, we define here the notion of MD interpretation, regardless of the approach used to obtain these MD interpretations.

An `MD interpretation` (MDI) represents a subset of the data sources' concepts placed in a valid MD space that answers the `IR` at hand. Thus, an *MDI* can be formally defined as a tagged graph, meeting `IR` and satisfying MD integrity constraints, such as

$MDI = (V, E, role : V \rightarrow_{\texttt{IR}} \{L, F\})$ such that $MDI \models MD_{ic}$

where V is the set of nodes, corresponding to source concepts, and E, the set of edges representing the associations between these concepts at the sources. Note that for a single MDI, "$\rightarrow_{\texttt{IR}}$" is a complete function and thus each node (V) plays an MD role according to `IR` (i.e., it is either a level or a fact, see the grammar introduced above). Additionally, the arrangement of nodes (V) and edges (E) in an MDI must satisfy the MD integrity constraints (MD_{ic}). Consequently, ORE relies on the previous approaches and assumes MDIs that are sound (i.e., that satisfy the MD integrity constraints) and complete (i.e., that satisfy the requirement at hand). Following the work in [141], we define here the MD integrity constraints that need to hold in order to satisfy both the soundness and completeness of the input MDIs.

- Information requirements. Information requirements (`IR`) are expected to have an analytical layout, i.e., having the subject of the analysis (fact) and the perspectives from which the subject is analyzed (dimensions). Such requirements resemble the natural language template provided earlier in this section.

- The multidimensional space arrangement. The dimensions of a requirement must arrange the MD space in which the factual data of a requirement is depicted, i.e., each instance of factual data is identified by a point in each of its analysis dimensions.

- The base concept. A minimal set of levels which functionally determine a fact must guarantee that given a point in each of the dimensions, such set of points determines one and only one instance of factual data.

- Data summarization. Data summarization must be correct which is ensured by applying necessary conditions for summarization, discussed in [114]: (1) Disjointness (the sets of objects to be aggregated must be disjoint); (2) completeness (the union of subsets must constitute the entire set); and (3) compatibility of the dimension, the type of measure being aggregated and the aggregation function.

Furthermore, an MDI contains two kinds of concepts. Those explicitly demanded in the input requirement and those implicitly needed to properly relate the explicitly demanded concepts in order to produce a single cube. Thus, the latter depend on the data sources implementation and do not appear in $\mathtt{IR}$. From here on, we refer to them as intermediate concepts. It is worth noting that initially, intermediate concepts may not have a strict MD role associated with them. Therefore, considering the potential ambiguity of business requirements and the diversity of the underlying associations in the data sources, several MDIs may result from a single requirement (each of them, capturing different semantics) [141]. We write $\mathbb{MDI}_{\mathtt{IR}}$ to refer to the set of all MDIs satisfying IR.

In [146], we describe our approach to generate and validate such set of MDIs per requirement (as the ones shown in Figure 4.2). However, ORE aims to be robust enough to process any valid MD cube-like form (i.e., respecting the fact/dimension dichotomy) generated manually or automatically from information requirements through any of the current approaches available in the literature. In fact, the only modification needed to couple another approach to ORE would be to serialize its output into the propriety format that ORE can automatically process.

Example. Each requirement, defined at the beginning of Section 2 (i.e., IR1-IR5), gives rise to a set of one or more MDIs (see Figure 4.2). Levels are depicted with white, and facts with gray boxes. The attributes of levels or facts are placed inside the corresponding boxes. The empty levels or facts (i.e., the boxes without any attributes) represent intermediate concepts and in the case their MD knowledge is ambivalent (i.e., either playing a factual or dimensional role, the MD integrity constraints are preserved) they are depicted as both level and fact. Consider now IR3, in plain text, and the MDIs produced for that requirement. There, we distinguish between concepts ex-

plicitly demanded (i.e., `revenue, parts`, and `region name`) and intermediate concepts (i.e., `partsupplier`, and `nation`).In this case, although `nation` is intermediate concept, its role is set to dimensional as to preserve the MD integrity constraints. However, `partsupplier` remains as an ambivalent concept since its MD interpretation has not been explicitly set in the requirement and cannot be unequivocally inferred without further information (either as factual or dimensional data the MDI satisfies the MD integrity constraints). Similarly, three out of the five requirements in our example (IR1, IR2 and IR3) contain ambivalent concepts and thus, they produce several MDIs (e.g., IR3 would produce two, one where `partsupplier` plays a factual role and thus, it can be used to find measures of potential interest, and another one where it plays a dimensional role and consequently, it is used as an analysis perspective). □

2.3 Formalizing the Problem

MDIs, by themselves, cannot serve as a final MD schema. Each of them answers a single requirement (i.e., cube-query) and may show unnecessary information (e.g., intermediate concepts may be of no interest at all and only be relevant for the ETL process). Intuitively, starting from a set of input requirements (each of them represented as an $\mathbb{MDI}_{\mathtt{IR}}$) and the data sources, the problem at hand looks for overlapping subsets of the input MDIs such that produce a single MD schema and meet some quality objectives.

Thus, we start from $\{\mathbb{MDI}_{\mathtt{IR}_i} \mid i = 1..m\}$ and following an iterative approach, we derive the unified MD schema satisfying the complete set of requirements. In order to lead the integration process, ORE uses a cost model (CM) which is defined with a set of parametric cost formulae ($\mathbb{CF}$) that evaluate some DW quality factors (QF) (e.g., structural complexity). A cost model together with its formulae is a parameter of ORE and can be alternatively customized by the designer to consider other quality factors (e.g., see [154]).

Without loss of generality, and for the sake of presentation, we assume that the resulting unified MD schema is a set of star schemas ($\mathtt{SS}$) where each star may answer one or more requirements. Similar to MDIs, we define $\mathtt{SS}$ as follows:

$$SS = (V, E, role : V \rightarrow_{IR_1 \ldots IR_m} \{D_1, \ldots, D_n, F\}), \text{ such that } SS \models MD_{ic})$$

Where m is the number of input requirements and n the number of dimensions in the output. This definition is similar to that of MDI, but guarantees that the MD role assigned to each node honors all the input requirements (i.e., "$\rightarrow_{IR_1,\ldots,IR_m}$"). Moreover, an MD schema, unlike a data cube, contains dimension hierarchies, which correspond to sets of levels with a partial, strict order ($\prec$) between them (e.g., nation $\prec$ region). Next, we search for a set of star schemas ($\mathtt{SS}$) such that

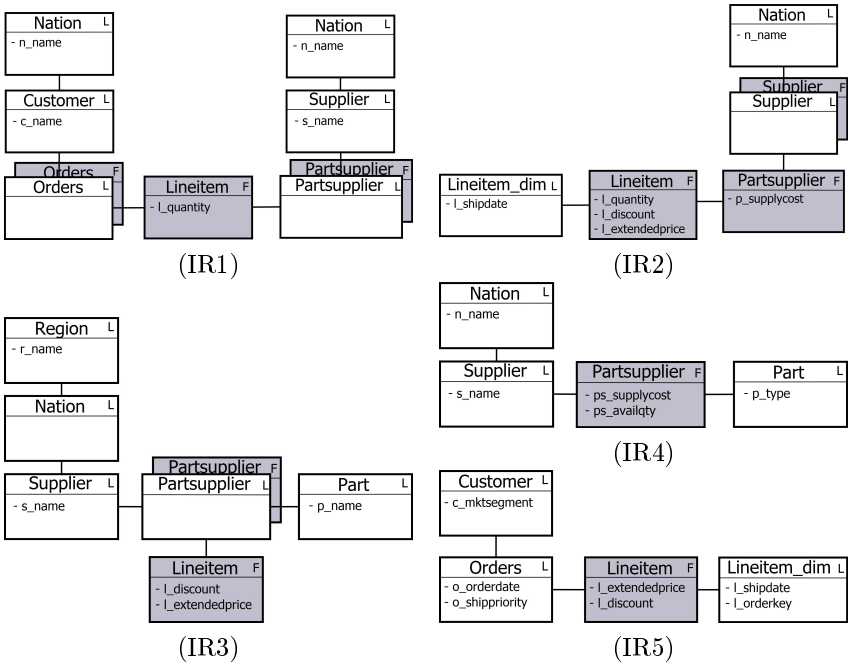


Fig. 4.2: Single MD interpretations for IR1-IR5

$$\forall i = 1..m : \mathbb{MDI}_{IR_i} \subseteq_q \mathbb{SS}$$

where $\subseteq_q$ implies $\exists MDI_k \in \mathbb{MDI}_{IR_i}$ and $\exists SS_j \in \mathbb{SS}$ such that:

$$V_{MDI_k} \subseteq V_{SS_j},$$
$$E_{MDI_k} \subseteq E_{SS_j},$$
$$\forall (a,b) \in E_{MDI_k} : role_{MDI_k}(a) = role_{MDI_k}(b) \Rightarrow role_{SS_j}(a) = role_{SS_j}(b)$$

Intuitively, each $\mathbb{SS}$ is a superset of the nodes, and edges of, at least, one MDI per requirement. Each connected subgraph with the same tagging in all corresponding MDIs gives rise to either a fact or one dimension in the star. Each $\mathbb{SS}$ is generated in accordance with the defined quality factors (QF) which are implemented by the chosen cost model (CM). Below, we further elaborate on how to express CM and meet the quality objectives (Section 2.4.2).

Example. Starting from the $\mathbb{MDI}_{IR1} \ldots \mathbb{MDI}_{IR5}$ identified for our requirements (see Figure 4.2), we aim at producing an MD schema meeting certain quality objectives (e.g., minimal structural complexity). ORE follows an iterative approach. Thus, it would start integrating the MDIs from IR1 and IR2 and produce a partial result (see Figure 4.3). Then, iteratively we integrate the

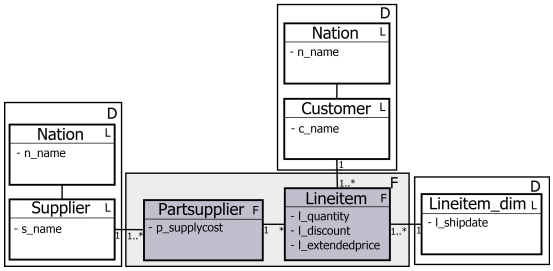


Fig. 4.3: MD schema satisfying IR1&IR2

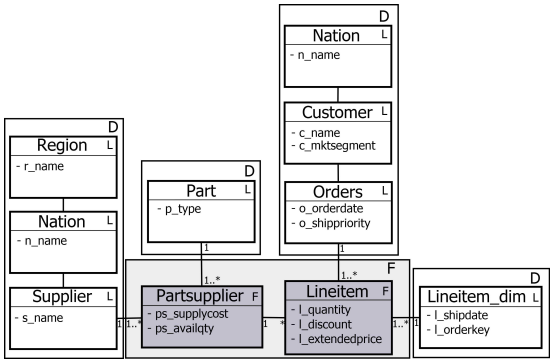


Fig. 4.4: MD schema satisfying IR1-IR5

remaining requirements. At each iteration, the quality factors QF are evaluated to choose among alternative integration options. Eventually, our method produces a single $\mathbb{SS}$ satisfying the input requirements and meeting the chosen quality objectives (see Figure 4.4). □

2.4 ORE in a Nutshell

This section presents the core components of our method (i.e., ORE). We first describe the inputs to our system and then, the processing stages of ORE.

2.4.1 Inputs

Data sources. Disparate internal and external sources may be of interest for an organization during the decision making processes. To boost the integration of new information requirements spanning diverse data sources into the final MD schema design, we capture the semantics (e.g., concepts, properties) of the available data sources in terms of an OWL ontology. The main role of the data sources ontology is supporting the integration of heterogeneous sources. Moreover, the use of an ontology, as proposed in [144], allows us to automati-

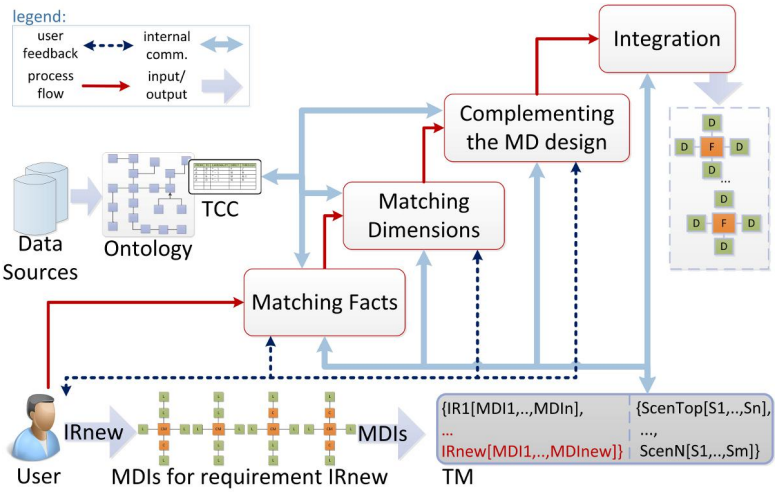


Fig. 4.5: Abstract representation of ORE stages

cally infer (by means of reasoning) relations between concepts (e.g., synonyms, functional dependencies, taxonomies, etc.). In the literature, many works have previously proposed to tackle data integration by means of ontologies. In this chapter, we assume the ontology is already available (how to obtain such ontology is out of the scope). For example, we can create or maintain a domain ontology as proposed in [142, 167].

Information requirements (IR). Information requirements are pre-processed as previously explained in Section 2.2 and, for each requirement, an $\mathbb{MDI}_{IR}$ is generated. Thus, the input of ORE is a set of $\mathbb{MDI}$s (one per requirement). As discussed in Section 5, our prototype expects $\mathbb{MDI}$s serialized as XML files. Thus, our method is flexible and the designer can choose any approach to generate the $\mathbb{MDI}$s as far as the results are serialized in a propriety XML format (e.g., using XSLT), which can be directly processed by ORE.

Example. In addition to the $\mathbb{MDI}$s for requirements IR1-IR5 (see Figure 4.2), which have already been discussed in Section 2.2, we use a domain ontology that corresponds to the TPC-H data stores (see Figure 4.1). In our example, we followed the approach presented in [167] to automatically produce it. However, due to space considerations, we do not further elaborate on this and we refer the interested readers to that paper for additional details. Both, the TPC-H and the set of $\mathbb{MDI}$s in Figure 4.3 are the inputs of ORE. □

2.4.2 Stages

A schematic overview of our approach is illustrated in Figure 4.5. Our method semi-automatically integrates new information requirements and incrementally produces an MD schema satisfying the requirements so far. At the same time,

the set of operations that illustrates such integration step is identified (i.e., integration operations) and further weighted (see Table 4.2) according to the cost model to assist designer's choice on meeting the chosen quality objectives (e.g., minimal structural complexity).

Our method, ORE, comprises four stages, namely matching facts, matching dimensions, complementing the MD design, and integration (see Figure 4.5). The first three stages gradually match different MD concepts and explore new design alternatives. The last stage considers these matchings and designer's feedback to generate the final MD schema that accommodates a new IR. If a new requirement does not entail any modification on the current MD schema it means that it is subsumed by the previous requirements considered.

Throughout all these stages, we use an internal structure, namely traceability metadata (TM), for systematically tracing the MD knowledge integrated so far. With TM, we avoid overburdening the produced MD schema with insignificant information for the final (business) user. For example, this structure keeps the information about all alternative MDI_{IR}, while only one MDI_{IR} is chosen per requirement to be included in the final MD schema. We keep these alternatives because, in the future, due to new requirements, we may need to reconsider the integration strategy chosen for a given concept. More details about the TM structure are provided in Section 3. At each iteration, the TM grows with the integration of each requirement and we use a cost model to prioritize the most promising solutions and prune the rest. TM, along with the user feedback, provides the basis for obtaining the final MD schema.

Consequently, a model for determining the cost of the output alternative solutions is considered as a part of TM. Depending on the DW quality factors that the user is interested in, the cost model can be arbitrary selected by the designer to support end-user choice. Thus, ORE is not coupled to any specific cost model. For the purpose of explanations and prototyping we consider the structural complexity as a DW quality factor for building our cost model. The structural complexity is discussed in [154] and different metrics are provided to support the cost model. These metrics refer to the correlation between DW quality factors, like understandability, analizability and maintainability, and different structural characteristics of a DW schema, like number of dimensional and factual concepts, their attributes (i.e., measures or descriptors), and number of functional dependencies. Table 4.2 shows the set of theoretically and empirically validated weights that express how introducing different DW concepts (by means of different integration operations) affects the considered DW quality factors (in this case structural complexity). We use this set of weights as an example case for ORE, and refer the reader to [154] for more details about how these values are obtained and later validated. We build our example cost model with the following formula ($f \in \mathbb{CF}$) that calculates the overall cost (i.e., structural complexity) of the MD schema. Note that the equation is parametrized with the weights defined in Table 4.2 and considers

Table 4.2: Integration operations

DW concept	Operation name	Weight
Dimensional	insertLevel	0.21
	insertRollUp	0.27
	insertDimDescriptor	0.04
	MergeLevels	0.04×(♯insertDimDescriptor)
Factual	insertFact	0.31
	insertFactMeasure	0.36
	MergeFacts	0.36×(♯insertFactMeasure)
Factual/ Dimensional	renameConcept	0

the number of different structural elements of an MD schema.

$f = \#level \cdot 0.21 + \#rollUp \cdot 0.27 + \#dimDescriptor \cdot 0.04 +$
$+ \#fact \cdot 0.31 + \#factMeasure \cdot 0.36.$

Example. The MD schema satisfying $IR1 - IR5$, depicted in Figure 4.4, contains the following MD structural elements: two facts with five measures, eight levels with nine level attributes, and four roll-up relations between the levels. Taking into account the equation f, we can calculate the overall cost in terms of structural complexity of the MD schema as follows:

$f = 8 \cdot 0.21 + 4 \cdot 0.27 + 9 \cdot 0.04 + 2 \cdot 0.31 + 5 \cdot 0.36 = 5.54.$ □

Next, we give a high-level description of the four stages, during which ORE iteratively integrates each new requirement into the MD schema that satisfies the requirements so far, preserves the MD integrity constraints and meets the chosen quality objectives.

Stage 1: Matching facts. Facts are the core MD concepts. Thus, we start looking for potential matches between facts in the current IR and the MD schema at hand. Consequently, we first search for different alternatives to incorporate the new IR (i.e., $\mathbb{MDI}_{IR}$) into TM. If ORE does not succeed, a new SS is created to support the new requirement. For matching facts, ORE searches for fact(s) in TM that produce a compatible set of points in the MD space. As a result, different possibilities to match the factual concepts in IR with already processed requirements are identified, as well as the appropriate sets of integration operations. The costs of these integration possibilities are further weighted according to our cost model (see Table 4.2), as explained in the previous example.

Stage 2: Matching dimensions. After matching facts, ORE conforms the dimensions producing the MD space of the merged fact. Different matches among levels are considered and thus, the different valid integration possibilities are obtained. With each possibility, a different set of integration operations for conforming these dimensions is considered and weighted.

Stage 3: Complementing the MD Design. ORE further explores the domain ontology and searches for new analytical perspectives (e.g., proposing new MD

concepts of potential interest). Nevertheless, ORE never enriches the design without the designer's acknowledgement and it is up to her to extend the current schema with new MD concepts in the ontology (i.e., levels, descriptors, and measures). Consequently, the designer is asked to (dis)approve the integration of the discovered concepts into the final MD schema.

Stage 4: Integration. The MD schema is finally obtained in two phases. First, we identify possible groupings of concepts containing an equivalent MD knowledge and which are directly connected by means of an MD compliment relationship (i.e., adjacent concepts). Then, we collapse them to capture the minimal information relevant to the user. Nevertheless, the complete *TM* is still preserved in the background to assists further integration steps.

3 Traceability Metadata

Introducing traceability into software projects has been widely recognized as beneficial (e.g., [15]) for several reasons: (i) the comprehension and understanding of the final products, (ii) maintenance and reusability of the existing software and, (iii) identifying the parts of the final product that fulfill particular input requirements and hence, the impact that the changes in those requirements have on the final product.

For the design of DWs, however, the potential advantages of traceability have been largely overlooked. In a recent research work [111], the attention is given to the benefits of keeping traces during the DW lifecycle. The authors identify three kinds of valuable traces in the DW context: (i) traces coming from information requirements, (ii) traces coming from the underlying data sources, and (iii) traces linking elements in the MD conceptual models.

Through our iterative approach, we aim at keeping the similar set of traces to enhance the MD schema design. To do so, we introduce a structure, namely traceability metadata (*TM*), for systematically keeping the interesting information about the MD knowledge integrated so far.

In Figure 4.6, we present the conceptual model of our *TM* structure. For the sake of comprehension, it has been divided in four areas (1-4), describing four types of traceable information which helps to assess the impact that particular requirements have on the resulting MD design. In comparison to the work in [111], we keep the traces coming from information requirements, i.e., requirement-related metadata (see 1 in Fig. 4.6) and the traces linking elements in the MD conceptual models, i.e., resulting MD schema and MD integration metadata (see 2 and 3 in Fig. 4.6). Notice that in this chapter, we only discuss the metadata related to schema transformations that are essential for ORE and thus, the traces related to the underlying data sources and data, which are essential for building the ETL processes, are not considered here but in our overall framework for the design and evolution of DW (see Section 8).

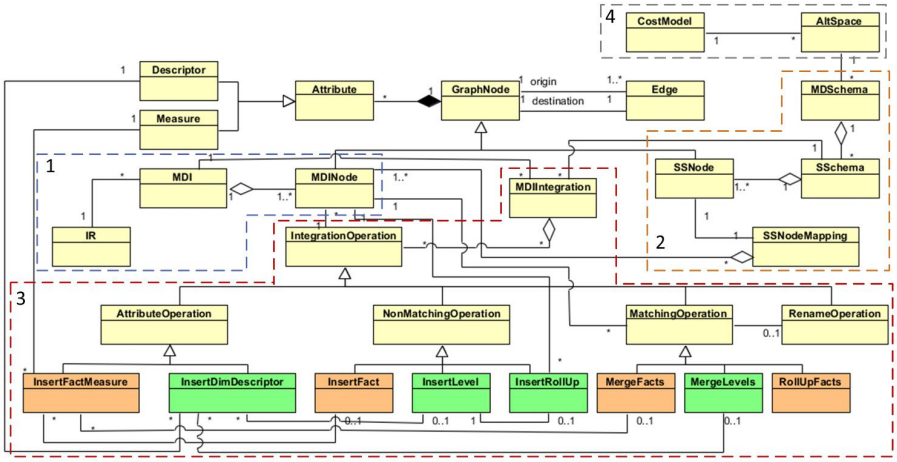


Fig. 4.6: *TM* conceptual model

Additionally, we keep the metadata about the cost model being used and w.r.t. the cost model, we produce and store a cost-based space of alternative solutions (i.e., 4). *TM* also assists ORE to respond to different changes that may occur during the DW lifecycle and to accordingly adapt the MD design of a DW to support such changes. *TM* stores the following information (1-4 in Figure 4.6):

1. Requirement-related metadata. As discussed in Section 2.2, IRs are represented as MDIs, which are modeled as tagged graphs. For this reason, it is mandatory that every `MDINode` is tagged with an MD role (either fact or level). Thus, in *TM*, for each *IR*, we store its $\mathbb{MDI}_{IR}$s as tagged nodes (`MDINode`) and edges (`Edge`). Furthermore, nodes (`GraphNode`) may contain attributes either dimensional (`Descriptor`, if the node is a level) or factual (`Measure`, if the node is a fact).

2. Resulting MD schema. An MD schema (`MDSchema`) may contain several star schemas (`SSchema`), which are essentially extended MDIs (see Section 2.2). Therefore, they are also implemented as tagged graphs (with nodes, `SSNode`, and edges `Edges`). Importantly, *TM* stores many different `MDSchema`s, each one reflecting different integration alternatives. These integration alternatives are described through mappings (`SSNodeMapping`) between `MDINode`s and `SSNode`s.

3. MD integration metadata. When an evolution event occurs (namely adding new information requirement), ORE efficiently accommodates the existing MD schemas to these changes and records their occurrences for assisting the user in future design steps. In *TM*, we store these changes as a set of operations (`IntegrationOperation`), which are necessary to integrate individual MD concepts (`MDINodes`) coming from new requirements

(i.e., new MDIs) into the existing MD design. There are several types of integration operations: (1) operations for inserting new MD concepts into the existing design (i.e., InsertFact, InsertLevel and InsertRollUp) (2) operations for enriching the existing concepts with new MD attributes (i.e., InsertFactMeasure and InsertDimDescriptor) and (3) operations for combining new MD concepts with the existing ones (i.e., MergeFacts, MergeLevels, RollupFacts and RenameOperation). According to Figure 4.6, in order to integrate two MDIs (MDIIntegration) we may need several integration operations (IntegrationOperation). Note that all operations are applied to one MDINode, except MatchingOperations and InsertRollUp that refer to an additional MDI. Furthermore, when inserting or merging, levels or facts, an additional set of operations may be identified (see bottom part of Figure 4.6). These operations can either insert new attributes to these concepts (i.e., InsertFactMeasure and InsertDimDescriptor), or when inserting a new level (insertLevel), to insert an additional roll-up relation (i.e., InsertRollUp).

4. Cost-based space of alternative solutions. Finally, *TM* also stores a reference cost model (CostModel) and a space of alternative solutions (AltSpace). When integrating new requirements, it may occur that several alternative scenarios appear to be valid outcomes of the MD design process (this is mainly due to ambivalent concepts; see Section 2.2). However, considering the chosen DW quality factors (see Section 2.4.2), not all of these output schemas would have the same overall score. In our example case the cost depends on the size and the structural complexity of each final solution. To assist the end user in efficiently determining the most suitable solution, in *TM* we maintain an ordered space of alternative solutions (AltSpace), according to the overall score that is calculated as explained in Section 2.4.2 by using the referenced cost model (CostModel). Thus, we compute, for each integration alternative, its cost. More specifically, each of the integration operations presented above adds a certain weight (see Table 4.2) to the overall cost of the solution. As discussed, to integrate two MDIs we may need several integration operations and consequently, ORE keeps track of the overall cost needed to integrate them, which is used to sort the space of alternatives.
We define this space as a partially ordered graph which is represented with a triplet ($\mathbb{S}, \mathbb{NB}, S_t$). $\mathbb{S}$ represents a set of valid scenarios in terms of final MD schemas and $\mathbb{NB}$ (Next Best) is a set of directed edges that order these alternative solutions inside the space. At any moment, the solution with the lowest overall cost can be identified as the top solution (S_t). In the case that the end user does not find the proposed top solution suitable for her needs, the $\mathbb{NB}$ edges going from that solution will lead her to its nearest alternatives, i.e., the next best solutions. Relevantly,

our search space only contains alternatives considering all requirements at hand (i.e., partial results are not kept). Moreover, considering the chosen cost model, the search space can be pruned to include only the top-N alternative solutions (see Section 5).

TM is a building block of ORE and it is constantly maintained to reflect and fulfill the set of IRs at hand. For example:

Adding an IR. Whenever a new requirement (*IR*) is posed, the *TM* is enriched with the missing metadata to answer such requirement. In particular, new $\mathbb{MDI}_{IR}$ that are identified for a given requirement are added to the *TM*. The added $\mathbb{MDI}_{IR}$ initiates a new iteration of ORE to identify how the requirement at hand can be integrated into the existing structures. To this end, we identify all parts of the existing schemas (i.e., `SSNodes`), where the MDIs of a new requirement can be integrated. Going from the identified `SSNodes`, through the `SSNodeMapping` we further identify possible relationships of the new MDIs with the MDIs of previous requirements. In the following section, we describe how an iteration of ORE is performed for such addition.

Example. Figure 4.3 shows the top solution S_t for integration of IR1 and IR2. If a new requirement comes (e.g., IR3), ORE will ask for $\mathbb{MDI}$ of IR3 and go through its four internal stages to match these MDIs with the current space of alternatives ($\mathbb{S}$). At the end, it will produce new space ($\mathbb{S}_{new}$) of MD schemas ($\mathbb{SS}_{new}$), each of them showing different integration alternatives, such that:
$\forall \mathbb{SS}_{new} \in \mathbb{S}_{new}, \forall i = 1..3 : \mathbb{MDI}_{IR_i} \subseteq_q \mathbb{SS}_{new}$
These new schemas are properly ordered (according to our cost model) and stored in the space of alternative solutions (i.e., `AltSpace`). □

Removing and changing an IR. Besides accommodating the current MD design to satisfy new IRs, *TM* also needs to react in response to a changed or disregarded requirement. In the case of disregarding the existing requirement, the requirement (*IR*) is removed together with its $\mathbb{MDI}_{IR}$. ORE is then relaunched for the remaining set of information requirements. When the exiting requirement is changed, ORE first removes the obsolete requirement and its MDIs and then adds a new (changed) requirement (*IR*) together with its $\mathbb{MDI}_{IR}$. The main process of ORE is then relaunched for the updated set of requirements. Note that relaunching ORE is not that costly at this point, considering the fact that the integration options for the existing MDIs are already stored in *TM* together with the user's feedbacks and preferences.

4 The ORE Approach

As shown in the previous section, at each moment, ORE with the information stored inside its *TM* structure satisfies the current set of requirements. The

Algorithm: ORE

inputs: $\mathbb{MDI}_{IR}$, $\mathbb{S}$, output: $\mathbb{S}_{new}$

1. $options := \emptyset$;
2. For each $\mathbb{SS}_{cur} \in \mathbb{S}$ do
   (a) For each $[MDI_i \in \mathbb{MDI}_{IR}, SS_j \in \mathbb{SS}_{cur}]$ do
      i. $matchedFactsOpers := FM(\text{getFact}(MDI_i), \text{getFact}(SS_j))$;
      ii. If $matchedFactsOpers \neq \{insertFact(F_{MDI_i})\}$ do
         A. $\mathbb{D}_{MDI_i} := \text{searchDimsOverFact}(F_{MDI_i})$;
         B. $\mathbb{D}_{SS_j} := \text{searchDimsOverFact}(F_{SS_j})$;
         C. For each $D_{MDI_i} \in \mathbb{D}_{MDI_i}, D_{SS_j} \in \mathbb{D}_{SS_j}$ do
            If $related(bottom(D_{MDI_i}), bottom(D_{SS_j}))$ then
            $matchedDimsOpers \cup = DM(\{bottom(D_{MDI_i})\}, \{bottom(D_{SS_j})\})$;
         D. $options \cup = [\mathbb{SS}_{cur} \backslash SS_j, SS_j, \text{matchedFactsOpers}, \text{matchedDimsOpers}]$;
3. For each $o \in findBestN(options)$ do
   (a) $\mathbb{S}_{new} \cup = applyOperations(o)$;
4. $\mathbb{S}_{new} := INT(complementingMDSchema(applyOperations(findTop(options))))$;
5. return $\mathbb{S}_{new}$;

space of alternative solutions is also created at the end to support users in finding the most suitable MD schema. For each new requirement ORE tries to find the valid correspondences among the MD concepts and to incorporate the new requirement into the existing MD design. At the same time, ORE attempts to meet previously set quality objectives and to produce the minimal cost of the output schema according to the proposed cost model (see Section 2.4.2). An exhaustive search inside the current space of alternatives is performed to achieve such goals. The main process of ORE is shown in the ORE algorithm.

Note that in order to uniquely reference the steps of the algorithms throughout the chapter we use the notation (Step "algorithm:step").

Internally, whenever a new requirement arrives (IR_{new}), ORE tries to match each MD interpretation (MDI_i) of that requirement with the schemas (SS_j) of each solution in the current space of alternatives ($\mathbb{S}$). The correspondences between an MDI_i and an SS_j are found through the matchings of their individual MD concepts, i.e., facts (FM call in Step ORE:2(a)i) and dimensions (DM call in Step ORE:2(a)iiC). For such matchings, ORE benefits from the ontology reasoning mechanisms to find relationships among the concepts from heterogeneous data sources, in an efficient and scalable manner (i.e., synonyms, taxonomies, direct and transitive associations). Notice that only the associations that preserve the MD integrity constraints are accepted. After all the operations to incorporate new concepts into the existing design have been found, ORE considers creating the new space of alternative solutions (Step ORE:3). The cost of each solution is calculated using the selected cost model as described in Section 2.4.2. Eventually, the set of solutions is pruned by taking into account their overall costs, such that only the best N solutions are kept for further consideration (the value of N can be parametrized by the user).

Algorithm: FM

inputs: F_{MDI_i}, F_{SS_j}, output: *intOps*

1. If $F_{MDI_i} == F_{SS_j}$ then $intOps := \{mergeFacts(F_{MDI_i}, F_{SS_j})\}$;
2. ElseIf $F_{MDI_i} \rightarrow F_{SS_j} \vee F_{SS_j} \rightarrow F_{MDI_i}$ then
   (a) If $F_{MDI_i} \rightarrow F_{SS_j} \wedge F_{SS_j} \leftarrow F_{MDI_i}$ then
   $intOps := \{mergeFacts(F_{MDI_i}, F_{SS_j}), renameConcept(F_{MDI_i}, F_{SS_j})\}$;
   (b) ElseIf $F_{MDI_i} \rightarrow F_{SS_j}$ then
      i. If $acceptableGranularity(MDI_i)$ then $intOps := \{rollupFacts(F_{MDI_i}, F_{SS_j})\}$;
   (c) Else // $F_{SS_j} \rightarrow F_{MDI_i}$
      i. If $acceptableGranularity(SS_j)$ then $intOps := \{rollupFacts(F_{SS_j}, F_{MDI_i})\}$;
3. Else
   (a) $\mathbb{D}_{MDI_i} :=$ searchDimsOverFact(F_{MDI_i});
   (b) $\mathbb{D}_{SS_j} :=$ searchDimsOverFact(F_{SS_j});
   (c) If $F_{MDI_i} \rightarrow \mathbb{D}_{SS_j} \vee F_{SS_j} \rightarrow \mathbb{D}_{MDI_i}$ then
      i. If $F_{MDI_i} \rightarrow \mathbb{D}_{SS_j} \wedge F_{SS_j} \rightarrow \mathbb{D}_{MDI_i}$ then
      $intOps := \{mergeFacts(F_{MDI_i}, F_{SS_j}), renameConcept(F_{MDI_i}, F_{SS_j})\}$;
      ii. ElseIf $F_{MDI_i} \rightarrow \mathbb{D}_{SS_j}$ then
         A. If $acceptableGranularity(MDI_i)$ then $intOps := \{rollupFacts(F_{SS_j}, F_{MDI_i})\}$;
      iii. Else // $F_{SS_j} \rightarrow \mathbb{D}_{MDI_i}$
         A. If $acceptableGranularity(SS_j)$ then $intOps := \{rollupFacts(F_{MDI_i}, F_{SS_j})\}$;
   (d) Else $intOps := \{insertFact(F_{MDI_i})\}$;
4. return *intOps*;

Among these N solutions ORE considers the top one to produce the final MD schema by complementing the current one with new knowledge and conforming its constructs respecting the minimal design properties (INT in Step ORE:4).

4.1 Matching facts

Respecting the MD integrity constraints described in Section 2.2, in order to match two facts, these should produce an equivalent set of points in the MD space. This is formally defined with the following condition (C):

(C): The fact $F_{MDI_i} \in MDI_i$ matches the fact $F_{SS_j} \in SS_j$ if and only if there is a bijective function f such that for each point x_{MDI_i} in the MD space arranged by the dimensions $\{D_1 \times D_2 \times .. \times D_n\}$ implied by F_{MDI_i}, there is one and only one point y_{SS_j} in the MD space arranged by the dimensions $(D'_1 \times D'_2 \times .. \times D'_m\}$ implied by F_{SS_j}, such that $f(x_{MDI_i}) = y_{SS_j}$.

The abstraction of the fact matching process that guarantees the fulfillment of the above condition (C) is described by the FM algorithm. Thus, FM tests whether C is satisfied by checking whether F_{MDI_i} (F_{SS_j}) functionally determines all the dimensions of F_{SS_j} (F_{MDI_i}), i.e., whether F_{MDI_i} (F_{SS_j}) is related by means of "1 - 1" or "* - 1" relationship to each dimension of F_{SS_j} (F_{MDI_i}) (Step FM:3c). If this happens, it is because either they share the same MD

space (Step FM:3(c)i) or one can be rolled up to the other (Steps FM:3(c)iiA and FM:3(c)iiiA). These relationships are searched by means of reasoning over the domain ontology (see Section 2.4.1). As a special case, such matching is achieved if both facts coincide or transitively if one fact functionally determines the other, which is cheaper to be checked. Therefore, as an optimization, we firstly check the latter (Steps FM:1 and FM:2, respectively). It is worth noting that if we roll up a fact, we must check whether the new granularity is still acceptable for the associated requirements.

If the full match between both facts is found, i.e., they are equal (Step FM:1) or they functionally determine each other (Steps FM:2a and FM:3(c)i), the *mergeFacts* operation is added, followed by *renameConcept*, if they are not equal. Alternatively, if one fact functionally determines the other (Steps FM:2(b)i and FM:2(c)i) or if it functionally determines all the dimensions of the other fact (Steps FM:3(c)iiA and FM:3(c)iiiA) the *rollupFacts* operation is identified to roll-up from the MD space of one fact to the MD space of the other (only when the coarser granularity is acceptable for all involved requirements). Otherwise, if FM cannot identify a valid matching for the fact F_{MDI_i}, it generates the *insertFact* operation (Step FM:3d).

Example (Step FM:1) Figure 4.7 shows the case when the fact Lineitem of the requirement $IR2$ matches the same fact in the existing TM. □

Example (Step FM:2) Figure 4.8 illustrates the case where the fact Partsupplier of $IR4$ matches the fact Lineitem that functionally determines it, which is identified in the ontology that captures the TPC-H data sources (see TPC-H schema in Figure 4.1). Therefore, the operation *rollupFacts*(*Lineitem, Partsupplier*) is identified. Notice that in this case, we still should check whether the new granularity allows to answer the requirements associated to SS_j. □

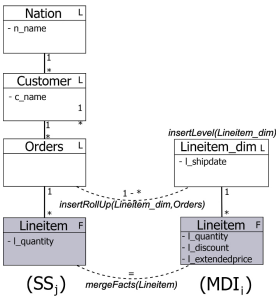


Fig. 4.7: Matching the MD Interpretation for IR2

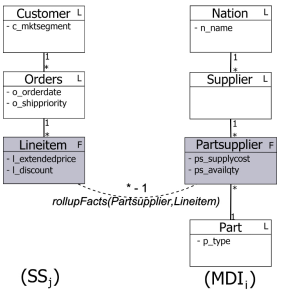


Fig. 4.8: Matching facts with *rollupFacts*

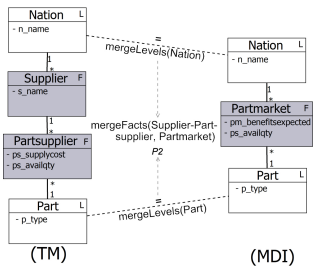


Fig. 4.9: Matching facts with *mergeFacts*

Example (Step FM:3c). Figure 4.9 shows an example inspired by TPC-H. A new PartMarket fact is introduced, which assesses the convenience of releasing a specific Part in a specific market (i.e., Nation). Even though we

may find no direct matching (i.e., Steps FM:1 and FM:2) between PartMarket and the Supplier-PartSupplier fact, their MD spaces do coincide as they both functionally determine the dimensions (i.e., Nation and Part) of each other. On the other side, in Figure 4.10, in the variation of the above example we have the case that the dimensions of the fact PartMarket are functionally determined by the Lineitem-PartSupplier fact but the opposite does not hold and thus, the $rollupFacts(\text{Lineitem} - \text{PartSupplier}, \text{PartMarket})$ operation should be added. As in the previous example, we still have to guarantee that the new granularity allows to satisfy the requirements associated to SS_j. □

Finally, all integration possibilities represented through the identified operations (i.e., *rollupFacts*, *mergeFact*, *renameConcept* or *insertFact*) are listed with the weights they add to the final solutions (see Table 4.2). ORE then creates an alternative solution for each of the integration possibilities. To prioritize the resulting solutions, the overall cost of each solution is evaluated according to predefined quality factors.

4.2 Matching dimensions

In this stage, ORE conforms the dimensions which form the MD spaces of the facts (F_{MDI_i}, F_{SS_j}) for which it previously found a matching that is not discarded by the user (Step ORE:2(a)iiC).

Since a single dimension D_x consists of a partially ordered set of individual levels (see Section 2.2), with single bottom (i.e., atomic) level and with "to-one" relationships among the levels, each dimension may be seen as a directed acyclic graph (DAG), where the vertices of the graph are individual levels and the directed edges between them are "to-one" relationships (i.e., functional dependencies). Note that we assume the dimension graph is guaranteed to be acyclic (this should be checked in a preliminar step), since the loops would violate the MD integrity constraints [114].

Having this in mind, the problem of matching dimensions may be seen as a graph matching problem. However, taking into account the MD context, ORE must additionally preserve the MD integrity constraints (see Section 2.2). Thus, we present here the DM algorithm, which solves the problem in our case.

DM is launched from the main (ORE) algorithm (Step ORE:2(a)iiC) for each pair of dimensions coming from MDI_i ($\mathbb{D}_{MDI_i}$) and SS_j ($\mathbb{D}_{SS_j}$) with the previously matched facts (F_{MDI_i}, F_{SS_j}) and bottom (atomic) levels connected with an MD compliant relationships (i.e., "=", "1-1", "1-*", and "*-1").

Considering the topological order of levels in each of the dimensions, DM starts by matching the atomic (*bottom*) levels of these dimensions.

In each call, DM searches for the matchings between all pairs of the candidate levels of MDI_i (i.e., $candidates_{MDI_i}$) and each candidate from the dimensions of SS_j (i.e., $candidates_{SS_j}$), starting from the bottom of each of them and

Algorithm: DM

inputs: $candidates_{MDI_i}$, $candidates_{SS_j}$, output: $intOps$

1. If $L_{MDI_i} == L_{SS_j}$ then
   $intOps \cup = \{mergeLevels(L_{MDI_i}, L_{SS_j})\} \uplus DM(getNext(L_{MDI_i}), getNext(L_{SS_j}))$;
2. ElseIf $L_{MDI_i} \rightarrow L_{SS_j} \vee L_{SS_j} \rightarrow L_{MDI_i}$ then
   (a) If $L_{MDI_i} \rightarrow L_{SS_j} \wedge L_{SS_j} \rightarrow L_{MDI_i}$ then
   $intOps \cup = \{mergeLevels(L_{MDI_i}, L_{SS_j}), renameConcept(L_{MDI_i}, L_{SS_j})\} \uplus$
   $DM(getNext(L_{MDI_i}), getNext(L_{SS_j}))$;
   (b) ElseIf $L_{SS_j} \rightarrow L_{MDI_i}$ then
   $intOps \cup = \{insertLevel(L_{MDI_i}), insertRollUp(L_{SS_j}, L_{MDI_i})\} \uplus DM(\{L_{MDI_i}\}, getNext(L_{SS_j}))$;
   (c) Else // $L_{MDI_i} \rightarrow L_{SS_j}$
   $intOps \cup = \{insertLevel(L_{MDI_i}), insertRollUp(L_{MDI_i}, L_{SS_j})\} \uplus DM(getNext(L_{MDI_i}), \{L_{SS_j}\})$;
3. Else // No matching for current levels
   (a) $intOps \cup = DM(\{L_{MDI_i}\}, getNext(L_{SS_j}))$;
   (b) $intOps \cup = \{insertLevel(L_{MDI_i})\} \uplus DM(getNext(L_{MDI_i}), \{L_{SS_j}\})$;
4. return $intOps$;

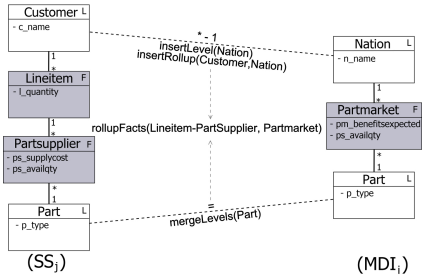


Fig. 4.10: Matching facts with *rollupFacts*

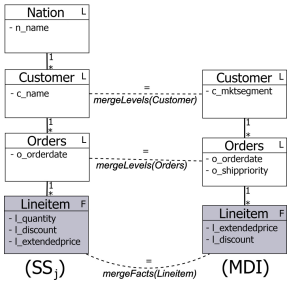


Fig. 4.11: Matching the MD Interpretation for IR5

hence, recursively moves forward through the corresponding hierarchy, depending on the multiplicity of the relationship found. For each pair of candidate levels, DM first checks if they exactly coincide (Step DM:1). If not, it may be that one functionally determines the other (Step DM:2). This can be either the case when both levels functionally determine each other, i.e., "1-1" relationship (Step DM:2a), or the case when only one of them functionally determines the other one, i.e., either "1-*" or "*-1" relationship (Steps DM:2b and DM:2c, respectively). Again, these relationships are identified by means of reasoning over the input ontology.

If the levels are equal or the "1-1" relationship is found (i.e., a full match), a recursive call moves forward in both hierarchies. However, when a "1-*" or "*-1" relationship is identified, DM can still explore the possibility that the level being in the to-one side matches the next one in the to-many side of the

relationship. Thus, it only moves forward in one of the hierarchies. Finally, if there is no relationship between the levels (Step DM:3), DM considers both alternatives, i.e., moving forward in either one or the other hierarchy. Thus, DM exhaustively visits all meaningful matchings for dimensions of MDI_i and of SS_j, and builds the set of possible integration options (*intOps*).

For each integration option, DM stores the information about the corresponding operations to be applied (see Table 4.2). When the full match between L_{MDI_i} and L_{SS_j} is found, either by equality or "1-1" relationship (Steps DM:1 and DM:2a), we consider either only the mergeLevels or the mergeLevels with the renameConcept operation to be applied, respectively.

Example. Figure 4.11 shows an integration possibility for the Orders dimension, in the case of integrating IR5 into the *TM* satisfying IR1-IR4. A full matching (i.e., equality) is then found for both Orders and Customer levels of the corresponding MDI_i. Thus, the mergeLevels operations are proposed for both Orders and Customer levels and they respectively involve insertDimDescriptor operations for transferring o_shippriority and c_mktsegment. □

On the other hand, if "1 - *" or "* - 1" relationships are identified (Step DM:2b and DM:2c), we consider inserting a roll-up relation (i.e., insertRollUp operation) in the star SS_j to a new inserted level.

Example. In Figure 4.7, for the "* - 1" matching found between levels Lineitem_dim and Orders, inserting a roll-up relation is proposed (i.e., insertRollUp(Lineitem_dim, Orders)), with the involved insertion of the level (Lineitem_dim). □

Similarly to the previous stage, we may identify different options to integrate levels in MDI_i with different candidate levels of the hierarchies from SS_j. Thus, we must consider the possibility of combining the current integration operations with all the alternatives of the successors (this is shown by the symbol $\uplus$ in the algorithm). Once all the integration options are found with their corresponding weights, ORE creates an alternative solution for each of the integration possibilities.

4.3 Complementing the MD design

After the previous two stages, ORE identifies the space of possible solutions for incorporating the new information requirement into the existing schema. As we discussed before, this space can be partially ordered considering user preferences and the cost of the individual solutions. In such a space we can obtain the top element that corresponds to the solution currently most suitable for the end user according to predefined quality objectives.

Considering the top solution found, in this stage ORE continues by analyzing the ontology to complement the future MD design with new analytically interesting concepts. This stage is optional and may be disregarded by the user. By exploring the functional dependencies ("to-one" relationships) in the

Algorithm: INT

input: $\mathbb{SS}_{top}$, output: $\mathbb{SS}_{new}$

1. $\mathbb{SS}_{new} = \emptyset$;
2. For each $SS_j \in \mathbb{SS}_{top}$ do
   (a) $SS_{new} := [\emptyset, \emptyset, \emptyset]$;
   (b) $seedF :=$ findFactualConcept(SS_j);
   (c) $\mathbb{F} :=$ group($seedF$);
   (d) setFact(SS_{new},collapse($\mathbb{F}$));
   (e) For each $(f, L_0) \in F_{SS_j} \wedge f \in \mathbb{F} \wedge L_0 \notin \mathbb{F}$ do
      i. $\mathbb{D} =$ group(L_0);
      ii. setDimension(SS_{new},collapse($\mathbb{D}$));
   (f) $\mathbb{SS}_{new} \cup = \{SS_{new}\}$
3. return $\mathbb{SS}_{new}$;

ontology, ORE identifies new levels for the previously conformed dimensions. Furthermore, different datatype properties in the ontology may also be identified either as measures of the existing facts or descriptive attributes of the levels. We distinguish two cases:

- If the property has a numerical data type (e.g., integer, double), we consider using it as a measure if and only if the domain concept of the property is identified as a fact.
- Otherwise, in the case that the domain concept of a property is identified as a level, our method suggests using the property as a new descriptor.

Different possibilities for enriching the current design are presented to the designer as different integration operations (i.e., insertLevel, insertFactMeasure, insertDimDescriptor). The designer may decide how to complement the MD design. As shown in [142], automation of such exploration process is highly achievable with polynomial complexity for the ontologies capturing the data source semantics.

Example. For the Orders dimension in Figure 4.11, ORE explores the ontology and proposes concept Region as a new top level of the given dimension. Also, ORE proposes different descriptors for the levels Orders, Customer, Nation, and Region; e.g., o_orderstatus, c_phone, and so on. □

4.4 Integration

Having the top solution for producing the MD schema identified in the first two stages and optionally complemented in the third stage, here, ORE produces the final MD schema. The integration process is described with the INT algorithm as follows.

The algorithm starts from the set of star schemas of the top solution ($\mathbb{SS}_{top}$) produced in the previous stages of ORE, which now additionally answers the *IR* at hand.

The factual and the dimensional concepts of each star (SS_j) in $\mathbb{SS}_{top}$ go through two phases, namely *grouping* and *collapsing*.

(I) Grouping. As the ontological concepts can be represented by a directed acyclic graph (DAG), these are combined together to produce different groups (subgraphs), so that all those in one group:

1. produce a connected subgraph and
2. have the same MD interpretation (i.e., all concepts are either factual or dimensional).

(II) Collapsing. Starting from these groups of concepts we obtain the final star schema. Inside each subgraph captured by a single group, we consider only the concepts currently required by the user, either provided with the requirement at hand or discovered when complementing the MD design in the ontology (i.e., Section 4.3). The concepts considered inside each group are then collapsed to produce one element (i.e., fact or dimension) of the final MD schema.

The INT algorithm first initializes the output MD schema ($\mathbb{SS}_{new}$, Step INT:1). Next, INT initializes the new star schema (SS_{new}) that results from integrating each star of $\mathbb{SS}_{top}$ (Step INT:2a), with the empty sets of vertices (i.e., facts or dimensions), edges (i.e., relationships among the MD concepts) and MD roles that the vertices play (see how SS is defined in Section 2.3). INT then searches for any factual concept (Step INT:2b) and starting from it, INT performs *grouping* of the factual concepts transitively related (Step INT:2c). As a result, the subgraph of related facts is produced and then *collapsed* in the next step (Step INT:2d) to produce a single fact added to SS_{new}. Starting from the collapsed fact, the levels arranging the dimensions over the new fact are explored (Step INT:2e) and starting from the atomic level (L_0), the adjacent levels are *grouped* in a similar way as the facts and *collapsed* to produce a single dimension. The dimensions produced are then also added to SS_{new}. Finally, a new star (SS_{new}) resulted from integrating each SS_j is added to the output MD schema (Step INT:2f).

In these two phases, we free the final schema from knowledge currently irrelevant to the designer's choices and conform the schema to meet the previously established DW quality objectives (i.e., minimal structural complexity). For example, collapsing the adjacent levels simplifies the corresponding dimensions and lowers the number of roll-up relationships, which has a significant influence in the overall structural complexity of the schema. The process of integration can optionally be assisted by the user, who can determine the level of integration by means of selecting the type of the finally produced schemas (i.e., star or snowflake).

While concepts with currently no interest to the designer may be hidden from the final MD schema design, TM structure still preserves all this knowledge for using it in future integration steps. Furthermore, as TM traces the

knowledge about the complete MD interpretations (i.e., including all the concepts) and also the mappings of these concepts to the data sources we can benefit from it when producing the appropriate data flow design (e.g., ETL process) to manage loading the data from the sources to the produced target MD schema. This is the part of our overall research work as it is discussed in Section 8 (e.g., see [88]) .

5 Theoretical Validation

In this section we present the theoretical validation of our approach where we examine the satisfaction of four major properties:

- Soundness. The resulting MD schema must satisfy the MD integrity constraints (see Section 2.2).
- Completeness. The set of information requirements integrated so far can be answered from the resulting MD schema.
- Commutativity. Independently of the order of the input information requirements, ORE must produce an equivalent MD schema at the output.
- Associativity. Independently of the order in which information requirements are integrated, ORE must produce an equivalent MD schema at the output.

Finally, we discuss the theoretical complexity of the problem of the requirement-driven DW design and our approach.

5.1 Soundness and Completeness

Here, we formally prove the soundness and completeness of ORE by analyzing the set of integration operations applied through its four stages (see Table 4.2).

Precondition. For each new requirement (IR), ORE starts from a set of MDIs ($\mathbb{MDI}_{IR}$) whose soundness (i.e., respecting the MD integrity constraints) and completeness (i.e., satisfying IR) are ensured by means of the constraints described in Section 2.2.

Trivial case. When the first requirement arrives (IR_{first}), ORE chooses an MDI_i ($MDI_i \in \mathbb{MDI}_{IR_{first}}$) such that MDI_i has the lowest overall cost in $\mathbb{MDI}_{IR_{first}}$ (w.r.t to the chosen cost model) and produces a MD schema by means of the INT algorithm. According to our precondition, all input MDIs are considered to be sound and complete. Notice that the soundness and completeness of the resulting MD schema is guaranteed because the INT algorithm focuses on presentation issues and does not affect semantics.

Hereinafter, we consider the general case when a new requirement is integrated into an existing (sound and complete) MD schema.

Invariant of the process. Given an $MDI_i \in \mathbb{MDI}_{IR_{new}}$ coming from a new requirement IR_{new} and a star SS_j from a current MD schema $\mathbb{SS}_{cur}$, ORE, through the four stages explained in the previous section, applies the set of transformations (i.e, integration operations, see Table 4.2) to produce the final MD schema. This process must guarantee the output schema soundness and completeness and thus we subsequently show that the results produced by each integration operation of ORE always results in a solution that preserves the MD integrity constraints (i.e., the operation is sound) and able to answer the current set of requirements (i.e., the operation is complete).

In what follows, we evaluate the above invariant by analyzing each integration operation individually. To analyze the soundness, for each operation, we test whether it only considers relationships that are compliant with the MD integrity constraints; whilst for the completeness, we test whether the newly created MD schema subsumes the complete MD knowledge (i.e., semantics) of the MDIs from which it was created.

Factual concepts operations

- *mergeFacts*(F_{MDI_i}, F_{SS_j}): ORE, in the FM algorithm (Section 4.1), merges facts $F_{MDI_i} \in MDI_i$ and $F_{SS_j} \in SS_j$ if and only if the MD spaces of facts F_{MDI_i} and F_{SS_j} are equivalent (see condition C in Section 4.1). From the precondition above (i.e., MDI_i is sound and complete), and by preserving the same MD space and keeping a single fact in the existing sound and complete star SS_j (invariant), it is not hard to see that *mergeFacts* operation also preserves the MD integrity constraints of SS_j and of the resulting MD schema (i.e., *mergeFacts* is sound). Moreover, the existing star SS_j, after merging two facts, still answers the previous requirements and additionally the new one through the existing or newly added measure attributes (i.e., *mergeFacts* is complete).

- *insertFact*(F_{MDI_i}): In the case that in the FM algorithm ORE does not find a fact with the equivalent MD space as the F_{MDI_i} fact of the new requirement, it inserts F_{MDI_i} and creates a new star SS_{new} in the output MD schema. Similar to the trivial case, *insertFact* operation does not affect the current set of stars in $\mathbb{SS}_{cur}$, but it creates a new star (SS_{new}) in the output MD schema, whose soundness is guaranteed by the soundness of the new MDI_i from which it is created (i.e., *insertFact* is sound). Consequently, as the existing set of stars ($\mathbb{SS}_{cur}$) is intact, it still answers the previous requirements, while the newly added star (SS_{new}) answers the requirement at hand (IR_{new}) (i.e., *insertFact* is complete).

- *insertFactMeasure*($m_{F_{MDI_i}}$, F_{SS_j}): When merging facts $F_{MDI_i} \in MDI_i$ and $F_{SS_j} \in SS_j$, in the FM algorithm, ORE may insert a new measure $m_{F_{MDI_i}} \in$

F_{MDI_i} into the existing fact F_{SS_j}. It is trivial to see that the *insertFactMeasure* operation does not affect the MD integrity constraints of F_{SS_j} and thus it does not violate the soundness of SS_j and of the output MD schema (i.e., *insertFactMeasure* is sound). Moreover, the existing star SS_j still answers the previous requirements and additionally the new one through the existing or newly added measure attributes (i.e., *insertFactMeasure* is complete).

Dimensional concepts operations

Note that ORE runs the DM algorithm (Section 4.2) only for the dimensions that form the MD spaces of the facts (F_{MDI_i}, F_{SS_j}) for which it previously found a matching. Additionally, ORE can introduce new analysis perspectives (i.e., dimensions) to the existing fact F_{SS_j}, but only if there is a relationship between the F_{SS_j} and the dimension at the data sources (i.e., in the ontology that captures them). However, this does not change the existing answerability of the fact but adds a new perspective through which it can be analyzed.

- *mergeLevel*(L_{MDI_i}, L_{SS_j}): ORE, in the DM algorithm (Section 4.2), merges levels $L_{MDI_i} \in MDI_i$ and $L_{SS_j} \in SS_j$ if and only if $L_{MDI_i} = L_{SS_j}$ or the "1-1" relationship between L_{MDI_i} and L_{SS_j} is found. By keeping the existing level L_{SS_j} in the sound and complete star SS_j (invariant), *mergeLevel* operation does not affect the MD constraints of SS_j but only potentially enriches the existing level L_{SS_j} with new level attributes (i.e., *mergeLevel* is sound). Moreover, the star SS_j, still answers the previous requirements and additionally the new one through the existing or newly added level descriptors (i.e., *mergeLevel* is complete).

- *insertLevel*(L_{MDI_i}): In the case that in the DM algorithm, ORE does not find any relationship of the level $L_{MDI_i} \in MDI_i$ with levels of a current hierarchy of the existing sound and complete star SS_j (invariant), it creates a new branch of the current hierarchy of SS_j and inserts a new level L_{MDI_i} by connecting it with the last matched level of that hierarchy. As DM only moves through valid dimension hierarchies of MDI_i and SS_j, *insertLevel* always connects newly inserted level to the last matched level using a "to-one" relation of the MDI_i hierarchy, and thus it preserves the MD integrity constraints of SS_j and of the output MD schema (i.e., *insertLevel* is sound). Moreover, as the lower levels remain intact, the star SS_j, still answers all the previous requirements and additionally the new one through the existing levels or newly added level in the existing hierarchy (i.e., *insertLevel* is complete).

- *insertRollUp*(L_{SS_j}, L_{MDI_i}): In the case that in the DM algorithm, ORE finds a "1-*" or "*-1" relationship between levels $L_{MDI_i} \in MDI_i$ and

$L_{SS_j} \in SS_j$, it adds a new level L_{MDI_i} to the existing sound and complete star SS_j (invariant) (by means of the *insertLevel* operation) and then inserts the roll-up relation to relate matching levels, i.e., L_{SS_j} and L_{MDI_i}. By considering only the relationships that fulfill the MD integrity constraints (i.e., "1-*" or "*-1"), *insertRollUp* preserves the MD integrity constraints of SS_j and of the output MD schema (i.e., *insertRollUp* is sound). Moreover, the star SS_j can still answer the previous requirements with the existing levels. Additionally, the new requirement can be answered with the existing levels or newly added level in the hierarchy by using the inserted roll-up relation (i.e., *insertRollUp* is complete).

- *insertDimDescriptor*($d_{L_{MDI_i}}, L_{SS_j}$): When merging levels $L_{MDI_i} \in MDI_i$ and $L_{SS_j} \in SS_j$, in the DM algorithm, ORE may insert a new level descriptor $d_{L_{MDI_i}} \in L_{MDI_i}$ into the fact F_{SS_j} of the existing, sound and complete star SS_j (invariant). It is trivial to see that the *insertDimDescriptor* operation does not affect the MD integrity constraints of L_{SS_j} and thus preserves the soundness of SS_j and of the output MD schema (i.e., insertDimDescriptor is sound). Moreover, the star SS_j, still answers the previous requirements and additionally the new one through the existing or newly added level descriptors (i.e., *insertDimDescriptor* is complete).

Factual/Dimensional concepts operations

- *renameConcept*(C_{MDI_i}, C_{SS_j}): In the case of merging two facts or two levels, ORE generates *renameConcept* operation to record that in order to integrate concepts C_{MDI_i} and C_{SS_j} it may be necessary to rename the new concept C_{MDI_i} to the name of the existing one C_{SS_j}. *renameConcept* is a syntactic change and thus it does not affect the MD integrity constraints of the existing star SS_j or of the output MD schema (i.e., *renameConcept* is sound). For the same reason, *renameConcept* operation neither affects the ability of the output MD schema to answer all the current requirements (i.e., *renameConcept* is complete).

From the above analysis, we can conclude that given a sound and complete $MDI_i \in \mathbb{MDI}_{IR_{new}}$ of a new requirement IR_{new} (precondition), and a sound and complete star $SS_j \in \mathbb{SS}_{cur}$ resulted from a single requirement (trivial case), applying any valid combination of integration operations in the ORE algorithm results in an output MD schema that preserves the MD integrity constraints (i.e., all operations are sound) and answers the entire set of requirements (i.e., all operations are complete). Thus, the initial invariant always holds. This further proves the soundness and completeness of our ORE approach and its algorithms. □

5.2 Commutativity and Associativity

Commutativity. Following the general algebra definition of commutativity, ORE needs to satisfy the following property (Note that the binary operator "$\cdot_{ORE}$" represents the application of ORE algorithm over the two information requirements, by means of first integrating the left requirement and then the right one):

$\forall i,j \in \{1..n\}, IR_i \cdot_{ORE} IR_j = IR_j \cdot_{ORE} IR_i$

This property sates that for any two requirements IR_i and IR_j, the order in which they are coming at ORE's input does not change the resulting space of alternative solutions.

Associativity. Similarly, if we follow the general algebra definition of associativity, ORE needs to satisfy the following property:

$\forall i,j,k \in \{1..n\}, (IR_i \cdot_{ORE} IR_j) \cdot_{ORE} IR_k = IR_i \cdot_{ORE} (IR_j \cdot_{ORE} IR_k)$

This property sates that for any three requirements in a fixed order (IR_i, IR_j, and IR_k), the order in which ORE integrates them (i.e., first IR_i and IR_j and then IR_k, or first IR_j and IR_k and then IR_i) does not change the resulting space of alternative solutions.

As stated before, when the new information requirement (IR_i) arrives, each of its MDIs is compared to each of the stars of all the MD schemata in the current space of alternative solutions. Only in the cases when the full matching between facts is found (i.e., "=" and "1-1") and when no matching is found at all, the FM algorithm creates a single solution by merging the matched facts (i.e., mergeFacts) or inserting the new fact into the output MD schema (i.e., insertFact), respectively. In all other cases, two different solutions are created: one, where the facts are merged into a single fact, through their dimensions or changing the fact's granularity (see Steps FM:2(b)i, FM:2(c)i, and FM:3); and another one where two different stars are created over the facts that are previously compared. Furthermore, the DM algorithm preserves all the alternative solutions previously generated by FM and potentially adds new ones by matching the dimensions. ORE stores all these alternative solutions inside TM (see Section 3). This characteristic of the ORE algorithm guarantees that no matter in which order the requirements arrive (commutativity), or in which order they are integrated (associativity), ORE exhaustively explores all integration options and thus it always produces the same set of solutions (i.e., MD schemata) at the output either by merging several stars into one or by creating separate stars. □

5.3 Computational complexity

We first analyze the complexity of the requirement-driven DW schema design problem. For each new information requirement (IR_i), we compare all the MDIs of that requirement ($MDI_1,..,MDI_p$) with all the stars ($SS_1,..,SS_q$) of

all alternative solutions ($\mathbb{SS}_1, .., \mathbb{SS}_r$). Furthermore, due to the different integration options explained in sections 4.1 and 4.2, we produce different alternative solutions that are compliant with the soundness and completeness properties discussed in the previous subsection. Considering the fact that for each new requirement we need to compare all its MDIs with all the current alternative solutions and find all the possible integration alternatives, it is not hard to see from the above that the general problem of the requirement-driven DW design is complex (clearly exponential). Formally, we can present it as follows:

If we assume a scenario where n information requirements are arriving at the input.

$IR_i, i = 1..n$

For each requirement IR_i, there can be m_i MD interpretations.

$m_i = |\mathbb{MDI}_{IR_i}|$

After the first requirement (IR_1) arrives, its MDIs are inserted into the space of alternative solutions $\mathbb{S}$. The size of $\mathbb{S}$ will correspond to the number of MDIs of IR_1, i.e., m_1.

$|\mathbb{S}| = m_1$

Next, when we integrate the second requirement (IR_2) into the existing space of alternative solutions $\mathbb{S}$, the size of $\mathbb{S}$ will correspond to the product of m_1 for IR_1 and m_2 for IR_2 multiplied by the coefficient of alternative solutions (i.e., γ_1). Note that when comparing a single *MDI* with one star *SS* there is at least one solution (i.e., $\gamma_i \geqslant 1$). However, there may be additional solutions which result from having different alternatives to integrate an *MDI* with *SS*. For example, in dimension matching algorithm in Step DM:3, if we do not find the direct matching between current levels we explore both the hierarchy of the first and the second dimension, which in turn can create more then one valid solution.

$|\mathbb{S}| = m_1 \cdot m_2 \cdot \gamma_1$

Moreover, the algorithm will perform $m_1 \cdot m_2$ comparisons of the MDIs of incoming requirement and the stars of the existing space of solutions.

Considering the above, after integrating requirement IR_n into the existing space of alternative solutions $\mathbb{S}$, the size of $\mathbb{S}$ will be the following.

$|\mathbb{S}| = (((m_1 \cdot m_2 \cdot \gamma_1) \cdot m_3 \cdot \gamma_2) \cdot .. \cdot m_{n-1} \cdot \gamma_{n-2}) \cdot m_n \cdot \gamma_{n-1}$

Let us consider that the average number of MDIs per requirement is m (i.e., $\forall m_i \in m_1..m_n, m_i \approx m$).

$|\mathbb{S}| = (((m \cdot m \cdot \gamma_1) \cdot m \cdot \gamma_2) \cdot .. \cdot m \cdot \gamma_{n-2}) \cdot m \cdot \gamma_{n-1} =$

Additionally, if we assume that each comparison produces in average γ alternative solutions (i.e., $\forall \gamma_j \in \gamma_1..\gamma_{n-1}, \gamma_j \approx \gamma$). Then, we can approximate the size of $\mathbb{S}$ as follows.

$|\mathbb{S}| = m^n \cdot \gamma^{n-2} \cdot \gamma = m^n \cdot \gamma^{n-1}$

Here, we can see that the total number of comparisons among the MDIs of incoming requirements and the stars of the existing space of solutions for n requirements will be $m^n \cdot \gamma^{n-2}$.

If we further assume a realistic scenario where the number of incoming requirements is big enough, we can then infer that $n \gg m$ and $n \gg \gamma$. Taking the above analysis into account, we can conclude that theoretical computation complexity for the problem of requirement-driven DW schema design is exponential (i.e., $O(2^n)$). □

As discussed in the chapter, and more specifically in sections 2.4 and 3, we introduce a cost model that evaluates some quality factors as a parameter of ORE. In our example scenario, we use the structural complexity of the output MD schema as a quality factor. We further use this cost model for introducing heuristics to tackle the inherent complexity of the problem at hand. To this end, the cost-based space of alternative solutions, explained in detail in Section 3, is created and pruned after each integration iteration to maintain only the top-N solutions.

Assuming that N is the limiting size of the space of alternative solutions $\mathbb{S}$, we revisit the analysis of the computational complexity of ORE.

With each new requirement IR_i, we need to perform the maximum of $m_i \cdot N$ comparisons between the MDIs of IR_i and the stars of $\mathbb{S}$. For n incoming requirements we perform the total of $m_1 \cdot N + .. + m_n \cdot N$ comparisons. If we again consider that m is the average number of MDIs per requirement, it is not hard to see that the total number of comparisons for n requirements is $m \cdot N \cdot n$.

As before, if we assume n being big enough, i.e., $n \gg m$, we can conclude that by introducing the cost-based heuristics, the computational complexity of ORE is guaranteed to be linear (i.e., $O(n)$). □

Such linear complexity, as we will experimentally show in the next section, can be further parametrized with the size of the space of alternative solutions (i.e., N). Moreover, as it will be also empirically shown, for a manageable size of N, ORE, in most of the cases does not miss the optimal solution w.r.t. chosen quality objectives.

6 Evaluation

In this section, we first describe a prototype system which implements ORE's functionalities. Next, we validate the correctness of the output MD schemata, obtained for the TPC-H benchmark example used in this chapter, by comparing these to the ones manually derived in the Star Schema Benchmark (SSB) [126]. Finally, we report on our experimental findings from both scrutinizing the ORE's prototype and performing an empirical case study with a group of participants. On the one hand, we scrutinize ORE to validate our theoretical conclusions w.r.t.: (1) the algorithm's complexity (in terms of the execution time), (2) quality of the results (in terms of the chosen quality factors), (3) the algorithm's characteristics (in terms of the number of performed integration operations), and (4) scalability (in terms of the growing amount of require-

ments). On the other hand, we evaluate the manual effort of the real end-users needed for the MD schema design w.r.t.: (1) elapsed time, (2) quality and accuracy of the results, and (3) the emotional response (by means of the user's confidence in the provided solutions), in order to show the inherent complexity of the design tasks when performed manually.

6.1 Prototype

As a proof of concept, we have built a prototype implementation of ORE. ORE expects two kinds of inputs: information requirements expressed as MDIs and an OWL ontology representing the sources. In order to integrate the end-to-end solution of our research work (more details in Section 8) and to automate our testing exercises, we linked ORE to GEM. GEM is our previous solution to produce MDIs from requirements, [146] (see Figure 4.12). Moreover, the output generated by GEM is guaranteed to be sound and complete. These two modules (i.e., GEM and ORE) are communicating through the TM module (Section 3). We have chosen MongoDB for implementing a storage repository for TM. We justify our choice with the fact that handling semistructured, document-oriented data (e.g., information requirements, MDIs, Star schema (SS)) is more efficient with NOSQL databases, like MongoDB. Moreover, the consistency tradeoffs (i.e., eventual consistency) that boost the scalability, availability and latency of a MongoDB, fit the TM's needs for data storage. Later in this section we show some experimental findings that confirm such design choice.

When a new requirement arrives, it is processed by GEM, which produces a set of MDIs describing the MD data cube representing such requirement and stores them into the TM repository. Next, the produced MDIs of a single requirement are read from TM, and serialized in a proprietary XML format understood by ORE. ORE then processes these MDIs as explained throughout this chapter and produces the alternative MD schemata, each one in terms of a set of stars (i.e., $\mathbb{SS}_i$), and stores them inside the *TM*.

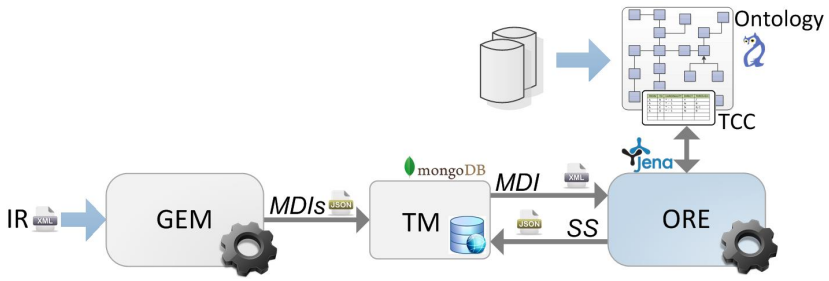


Fig. 4.12: Prototype setup

Figure 4.12 shows the architecture of our prototype. This architecture is

modular in that another module could be chosen to replace GEM if necessary, as far as the output MDIs are sound and complete (see Section 2.2) and they are expressed in ORE's proprietary XML format.

For describing the sources as an OWL ontology we followed the method proposed in [167], which largely automates the process for well-formed sources. ORE communicates with the ontology describing the sources by means of Jena API [6]. Depending on the complexity of such ontology, the requests for finding the relationship between the ontology concepts may be costly (e.g., [144]). Since such requests are frequent, in our implementation we used a component called Transitive Closure Cache (TCC) to facilitate the probing of the ontology. TCC receives requests from ORE for discovering the relationship between two concepts, e.g., (A,B). If no entry is found, it means that no previous iteration asked for the relationships of A. At this point, *TCC* accesses the ontology and looks for the requested relationship, but it also further explores the ontology to find the transitive closure for A through "1 - 1", "1 - *" or "* - 1" relationships and loads the corresponding entries to its structures. If a following request relating A, say (A,C), is posed, it will be answered from *TCC*. *TCC* works with a limited size of the internal memory and the overwriting is done following the least recently used cache algorithm.

6.2 Output validation

Following our running example scenario (i.e., the TPC-H benchmark), we performed a set of experiments to validate the reliability of ORE by examining the correctness of its resulting MD schemata. TPC-H is an ad hoc decision support benchmark, hence it provides a convenient source for analyzing data from different perspectives. Some efforts have been invested in adapting the schema of the TPC-H benchmark for MD purposes. In [126], the authors present the Star Schema Benchmark (SSB). They manually design an MD schema based on the TPC-H benchmark by applying traditional optimization techniques (e.g., denormalization) to the classical TPC-H schema. Here, we list the manual modifications of the TPC-H schema that are proposed in [126] and describe how each of these design choices is automatically supported in ORE.

- Combine the TPC-H `Lineitem` and `Orders` tables. The fact matching stage of ORE first merges the facts of different requirements that are placed in the same MD space. Furthermore, in the last stage (i.e., integration), ORE denormalizes the schema and combines the adjacent MD concepts into a single fact or dimension.
- Drop the `Partsupplier` table. As explained in [126], the `Partsupplier` table is removed from the design as it has a different temporal granularity from the `Lineitem` and `Orders`. The authors further discuss that `Partsupplier` can only be treated as a separate fact table belonging to

a different data mart. As a matter of fact, when in the fact matching stage, ORE does not find any existing fact whose MD space matches a fact from the new requirement, it creates a new star inside the existing MD schema with the new coming fact.

- Drop the `comment` attribute from `Lineitem`. We discuss in Section 2.2 that ORE only accepts MDIs that satisfy the MD integrity constraints. As explained in Section 2.2, one of the requirements to guarantee this is a correct data summarization which is further guaranteed among others, by the condition of Compatibility. This condition indeed ensures that the type of a measure being aggregated and the aggregation function are compatible, which guarantees that the textual attributes, like `l_comment`, will be left out from the fact table design.
- Drop the tables `Nation` and `Region` outboard to the dimensions `Customer` and `Supplier`. In [126], the authors drop the tables `Nation` and `Region` and add this information as attributes to the `Customer` and `Supplier` tables (i.e., `address`). Such design choice is widely supported in ORE's last stage (i.e., integration), where adjacent levels are combined and the dimension is denormalized and collapsed into one table.
- Adding the `Date` dimension. To adapt the schema to the standard DW design principles, the authors in [126] propose to complement the schema with the new `Date` dimension which is very common when analyzing the factual data such as `sales`. In ORE, after the fact and dimension matching stages, the new MD schema can be additionally tuned with the analytically interesting concepts in the stage of complementing the MD design. In this stage, the domain ontology is searched and the new valid MD concepts are proposed to the user. Likewise, the new dimension (e.g., `Date`) can be added to the MD design.

As an empirical proof, we ran ORE for a set of information requirements adapted from the queries that are provided by the TPC-H benchmark and we automatically obtained the same MD schema as SBB. The only exception is the `Partsupplier` fact which ORE, due to different granularity from other facts (e.g., `Lineitem`), separates in a different star. □

6.3 Experimental setup

The experimental setup involves a system called LEARN-SQL [8], which has been used at UPC-BarcelonaTech since 2003 to assess assignments related to exercises on the SQL language. The system is implemented using the open-source, learning management system, i.e., Moodle. The data gathered so far involve 2550 students in five different subjects and concerns more than 600 different SQL items (assignments). The students issued more than 150.000 submissions, which created a significant data source to be analyzed for gain-

ing an insight into the students' performance regarding different aspects of the subjects. Three main data sources are considered for the analysis in the LEARN-SQL system: (1) the operational DB deployed in PostgreSQL, (2) the students' data in XML, and (3) the evaluation results (per semester) in spreadsheet format. As a goal, a target MD data store should be provided to conveniently support the demands from different analytical requirements posed by the system users (e.g., lecturers).

For evaluating our approach, we conducted a set of experiments using the prototype being implemented (see Figure 4.12), with the aim to scrutinize ORE for measuring its performances (i.e., computational complexity, results quality, and different characteristics of the algorithm) and scalability (in terms of the number of requirements). As in the rest of this chapter, here we also use the structural complexity as the example quality factor for evaluating the cost of the output MD schemata. As an additional validation step for demonstrating the need for automating the MD schema design, we performed a set of empirical tests to measure the manual efforts of the real users when manually designing an MD schema from information requirements (in terms of the elapsed time, the quality and accuracy of provided solutions, and the emotional response of the users).

6.4 Scrutinizing ORE

As previously discussed in this chapter (see Section 5.3), the MD schema design from information requirements is a computationally complex problem due to the exponential growth of the space of possible solutions. However, as it can be seen in Figure 4.13, not all these solutions have the same cost when it comes to structural complexity of an MD schema. Therefore, we introduced a cost-based approach based on our cost model to prune the space of alternative solutions. Taking this into account, we used the previously described ORE's prototype and performed a series of experiments. These experiments aimed at validating the linear complexity of our cost-based solution (see Section 5.3) and generally the practical feasibility of our approach. We collected different indicators about ORE's performance and scalability, and at the same time, for our example scenario we aimed to show that ORE meets the chosen quality objectives, i.e., the resulting MD schema is with minimal structural complexity.

Inputs. As we explain shortly, we had a case study conducted with the help of a number of participants, real users of the LEARN-SQL system. During this process, one group of participants (i.e., DB lecturers) analyzed the domain ontology that captures the semantics of data sources in the LEARN-SQL system, and provided us with a set of thirteen requirements. Taking a look at them, the users most often wanted to analyze the success of students focusing on different indicators (e.g., final marks, outcomes of single assignments, results of the experiments executions over students' solutions, etc.) and from different

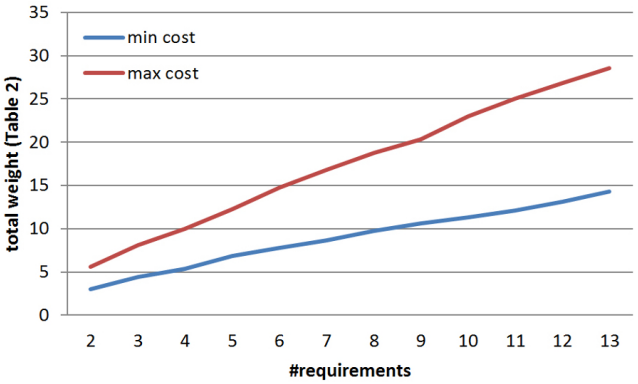


Fig. 4.13: Cost differences

perspectives (e.g., candidates, semesters, kind of assignment items, subjects, etc.). These requirements represent a real world case as they came from real users showing real demands on a real system. What these requirements also showed is that different users may be interested in performing the same or similar analysis which is a common case in real business environments. We use this set of thirteen requirements as input dataset for performing the experiments over ORE.

Experimental Methodology. Linking ORE to GEM allowed us to provide the collected requirements as inputs in our prototype setting (see Figure 4.12). From each individual requirement, GEM produced the corresponding set of *MDI*s, which then fed ORE together with the domain OWL ontology we created for the experiments with the users. We then ran ORE for different permutations of the input set of requirements. In each permutation, we simulated iterative arrivals of new requirements to the system i.e., starting from a single requirement a new requirement was considered in each iteration to be incorporated into the current MD schema design. Through a series of experiments that we explain shorty, we collected different indicators for each permutation and iterations (e.g., overall time per iteration, time spent on fact matching, time spent on dimension matching, overall time per complete permutation, #matching facts, #matching dimensions).

Experiments. Next, we discuss different experiments we performed over ORE in more detail and report our findings.

Time. Considering that in general the requirement-driven DW design problem is exponential, which we theoretically validated in Section 5.3, we performed a set of experiments following the above methodology and with varying input loads and constraints. The results are shown in Figure 4.14. As assumed, dealing with the complete space of solutions is expensive. After only ten require-

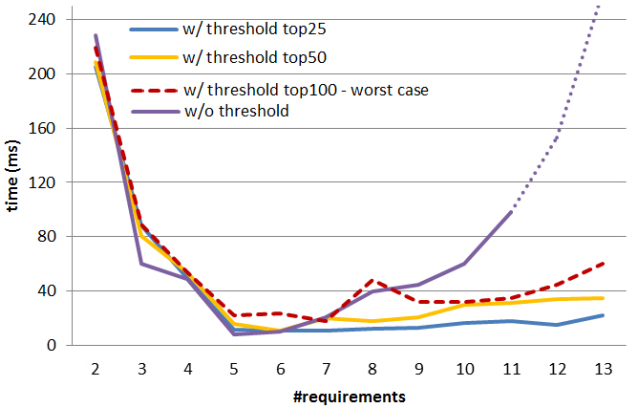


Fig. 4.14: Time comparison

ments (considering all the *MDI*s of those requirements) the time to iteratively integrate each requirement starts rapidly to grow and it later bursts for additional requirements (see dotted part of the purple line, i.e., w/o threshold, in Figure 4.14). However, as previously discussed, our approach is cost-based (in this experiment, we used as a quality factor the structural complexity of an output schema) and we only keep the top-N best solutions after integrating a new requirement. In the experiments, we used different values for N (i.e., 25, 50, and 100) to analyze how N affects the execution time of ORE. The linear complexity of our cost-based approach discussed in Section 5.3, we empirically confirm here after introducing the cost-based heuristics (i.e., keeping the top-N solutions). The additional tests with different thresholds (i.e., N) showed us that after new requirements come, the time tends to stay in a certain range depending on the number of top solutions kept (i.e., the value of N). Additionally, Figure 4.14 (dashed line), shows that peaks may appear since the size of inputs (i.e., #*MDI*s per requirement) is not limited. However, our cost-based pruning keeps the problem manageable even in the worst case, regardless the number of requirements already dealt with.

Another interesting observation is the higher latency that appears at the beginning, when integrating the first few requirements (see Figure 4.14). This comes from the fact that at the beginning we make a direct access to the ontology for checking different relationships among concepts, which tends to be costly. As we discuss in Section 6.1, in our prototype we introduce a component called Transitive Closure Cache (TCC), where we cache relationships among ontology concepts, so over time we avoid accessing the ontology directly as the relationships are mostly found in TCC. Notice in Figure 4.14 that even with the expensive initial accesses to the ontology the time needed for integrating new requirement stays within a manageable rate (200-240ms).

When measuring the times of different stages in ORE, we noticed that the average ratio of time spent on the fact matching stage (*FM*) appears to be significantly higher as opposed to the other stages. As an example, the average ratio of time between fact and dimension matching stages (*FM* and *DM*) is shown in Figure 4.15. Such behavior showed us that the real problem of integrating MD designs is indeed finding the matches of MD spaces of their factual concepts which is done during the FM algorithm. Moreover, matching the MD spaces of facts through all their dimensions tends to be more complex than finding the relation between the dimensions of the already matched facts.

Results Quality. Within the same set of experiments we also analyzed the cost of the output MD schemas that ORE produces w.r.t. the quality objectives we chose (i.e., minimal structural complexity). By testing different thresholds (i.e., values for N) we noticed that the optimal solution was always in the top-N (independent of the size of N) for the current setting. Clearly, as it usually happens with pruning techniques, at the moment there is no formal guarantee that pruning will always consider the optimal solution. At the same time, we also analyzed the alternative solutions with the highest cost. As one may notice in Figure 4.13, there is a significant and increasing difference of the costs between the optimal solution (i.e., min cost) that ORE finds and the worst solution (i.e., max cost) that may result from some other approach (e.g., manual design). Such observations additionally demonstrated the importance of automating the DW design process with the guarantee of meeting predefined quality objectives.

Algorithm's characteristics. We also studied the behavior of characteristics of ORE's algorithms. Figure 4.16 shows how the number of matchings of different concepts (factual and dimensional) is affected by the size of the problem. The number of matchings represents the average matchings found between *MDI*s of incoming requirement (1-13) and each of the alternative schemas at the output. Starting from scratch, ORE first, as it finds the empty *TM*, inserts the facts and dimensions and creates first designs at the output. Then, it exhaustively tries to match the facts and also their corresponding dimensions, as explained in Section 4. However, as the incrementally integrated design matures, there are lesser novel opportunities for matches, i.e., MD design becomes more complete. In Figure 4.16, one may notice that the number of the facts matched is proportionally lower then the number of levels, as we often have the situation of having a single fact and several of its dimensions. On the other hand, as we have already shown in Figure 4.15 time to perform the fact matching is still significantly higher from the time for dimension matching.

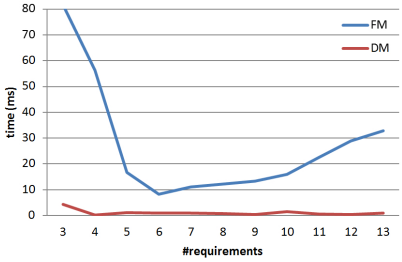


Fig. 4.15: Time per stage

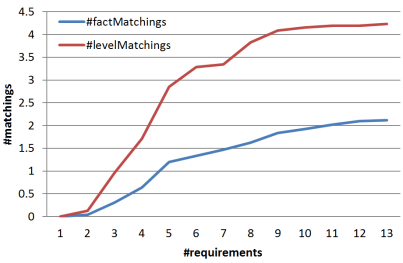


Fig. 4.16: Matchings found

Scalability. As we showed with the previous set of experiments, ORE is able to handle the growing amount of requirements, by using the introduced cost-based heuristics (i.e., top-N solutions). We first theoretically (in Section 5) and then practically (in this section) proved that the complexity of the approach by using these heuristics is linear and manageable regardless of the number of incoming requirements. Additionally, for the sake of analyzing the scalability and maintainability of the *TM* structure (see Section 3) for the growing amount of information requirements, we performed a set of tests where we measured the times for accessing the *TM* on MongoDB for reading and writing of different elements (e.g., *MDI*s). These tests showed us that the accessing times do not depend on the current size of the *TM* but mostly on the size of individual elements that are read or stored. As an example, reading MDIs from TM with the average size of five concepts (facts or dimensions) has the average latency of 5.2ms and it goes to maximum of 9ms for MDIs larger than 10 concepts, independent of the current size of *TM*.

Summary. This set of experiments demonstrated the feasibility of our semi-automatic approach for MD schema design. Furthermore, our cost-based heuristics (see Section 5.3) proved to maintain the linear complexity of the approach which makes it scalable regardless of the number of requirements and yet showed that ORE tends to generate the optimal solution. The main recommendation for the future users of the system would be to carefully choose the size of the space of alternative solutions which is used for the cost-based pruning (top-N). If N is too small (e.g., lower than 15), ORE does not always provide the optimal solution at the output. However, if N is too high (e.g., larger than 500), more solutions will be produced (including the optimal), but the performances will drastically drop. In fact, in our experimental scenario we showed that even with N in the rage [25,100] which is rather manageable, ORE always provides the optimal solutions at the output.

6.5 The LEARN-SQL Case Study

After evaluating the performance and observing different characteristics of ORE, we additionally conducted a series of empirical tests with the goal to measure the emotional aspects and the amount of human efforts needed for manually designing MD schema from requirements, as well as for incorporating new requirements into an existing schema.

Experimental methodology. These tests resembled the common practice of a DW design project and aimed at capturing all the stages of such process, from the collection of information requirements to the final design of the MD schema that satisfies them. We considered the following variables: (a) independent, i.e., the participants and the information requirements; (b) controlled, i.e., the object of the experiment (the domain ontology capturing data sources' semantics of the system); and (c) dependent, the observed parameters. We divide the observed parameters in three groups: (1)Time - How long did it take for participants to complete the task? (2) Accuracy and Quality - How many mistakes participants made? How far their solutions are from the ideal case? (3) Emotional response - How confident the participants felt about the completed tasks? The analysis was both quantitative and qualitative.

We considered two groups of users with different backgrounds.

1. Six DW and conceptual modeling experts - researchers and lecturers from the Department of Service and Information System Engineering (ESSI) at UPC BarcelonaTech.

2. Six graduate students following the DW course at the Facultat d'Informatica de Barcelona (FIB).

The tests were organized in three separate rounds as follows.

We first had a round where the seven real end-users of the LEARN-SQL system familiarized themselves with the vocabulary used to build the domain ontology. Next, these seven participants were asked to provide information requirements for analyzing different aspects of the system. Participants used the natural language template introduced in Section 2.2 as guidelines for providing the requirements. For example, the following is a requirement demanded by one of the lecturers: "Analyze failed experiments, from the point of view of the day of the week, student's group and difficulty of assignment, where the final mark is lower than 7." Each participant came up with maximum 2 requirements totalling in 13 requirements for the study.

In the second round, the live session is organized and the participants of both groups are introduced to the problem under the study. As some of the participants, especially among the students were not DW experts, this session provided some insights about the MD design principles, MD integrity constraints and the structural complexity of MD schema design as an example

quality factor used throughout this chapter. At the end of this session, several participants posed questions and after the discussion the participants were positive about being familiar with the problem under the study.

The final round, organized also as a live session, included two assignments of the manual DW design that the participants were solving. In the final round, both groups of participants were assessed.

Assignment 1 One of the information requirements produced in the first round was handed out to participants, namely: "I want to analyze the average valid processing outcome for each semester and the candidate where the candidate's subject is DABD."

Using the ontology graphical representation describing the sources (see Figure 4.17), the participants were asked to provide the MD interpretation of the ontology subset needed to answer the provided requirement (i.e., to identify the concepts that are needed to answer the requirement and determine a valid MD role(s) for such concepts -i.e., factual and/or dimensional-).

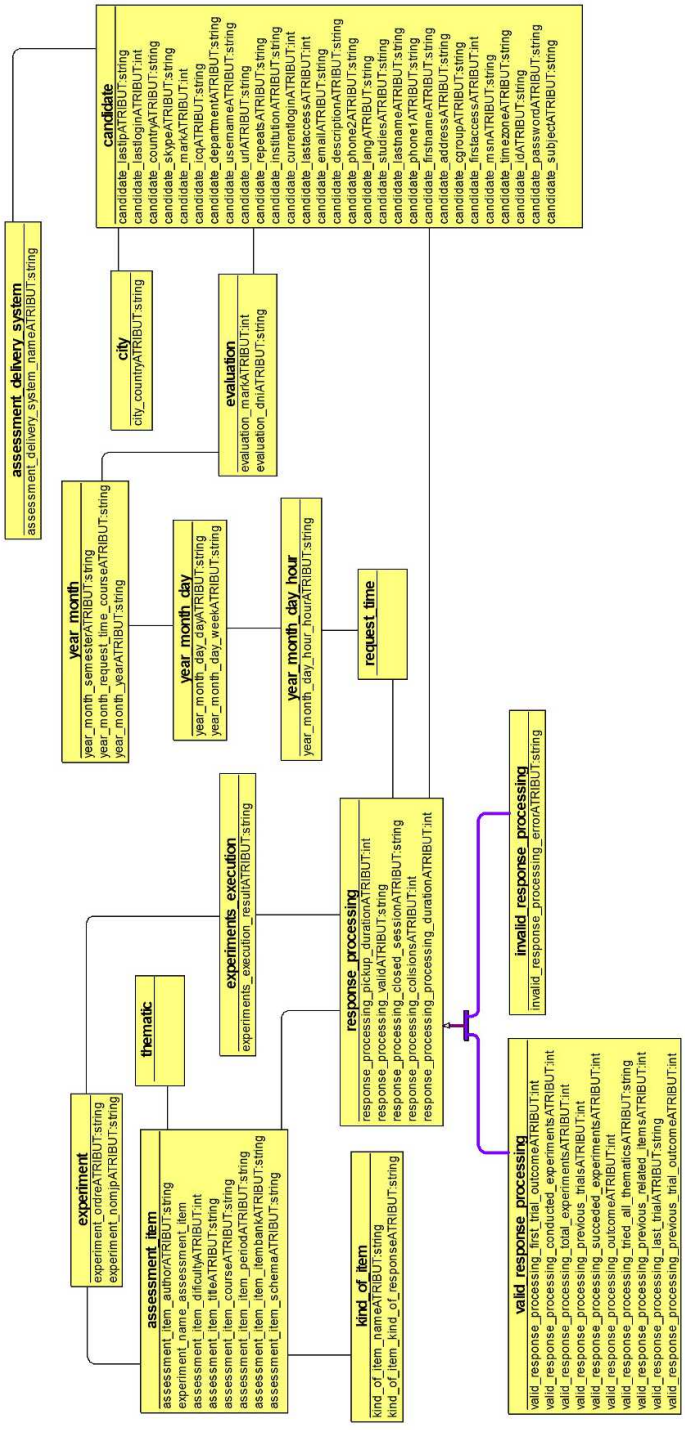


Fig. 4.17: LEARN-SQL ontology

To answer this assignment, the participants were asked to fill in a grid as the one presented in Figure 4.18, one per each valid MDI, and to record the start and the end time of the work. The time limit for this task was 10 minutes.

Concept / MD Context (F/D)	Evaluation	response_ processing	candidate	...	assessment_ item
Fact1	X				X
Dim1			X		
Dim2					X
...					

Fig. 4.18: Table for identifying *MDI*s for an *IR*

After the participants provided their solutions, the actual correct solutions for the given assignment were presented to the user (see Figure 4.19). Additionally, the reasons why several MDIs resulted from a single requirement are explained to the participants (i.e., existence of the intermediate concepts).

Assignment 2 In this assignment, we introduced a referent MD schema (see Figure 4.20) meeting a set of previous requirements. In this assignment, participants were asked to integrate the requirement considered in the previous assignment with the referent MD schema. Intuitively, they are asked to look for an MDI (out of the four produced in the previous assignment) such that it achieves the best integration with the referent schema according to the set of quality objectives, i.e., we asked them to minimize the structural complexity of the resulting MD schema, as discussed in Section 2.4.2. As a result, the participants were supposed to produce the final unified MD schema that additionally satisfies the new requirement, respects the MD integrity constraints and meets the quality objectives (i.e., minimal structural complexity of a MD schema). To answer this assignment the participants were asked to sketch the new MD schema on paper and record the starting and ending time. The time limit for this task was 15 minutes.

Results. Next, we report on our findings from the performed tests, in two parts. The first part aims at identifying the background of the users and the efforts invested in solving the given assignments. In the second part, we evaluate the solutions provided in the given assignments and investigate the possible reasons why the participants failed to provide the correct solutions.

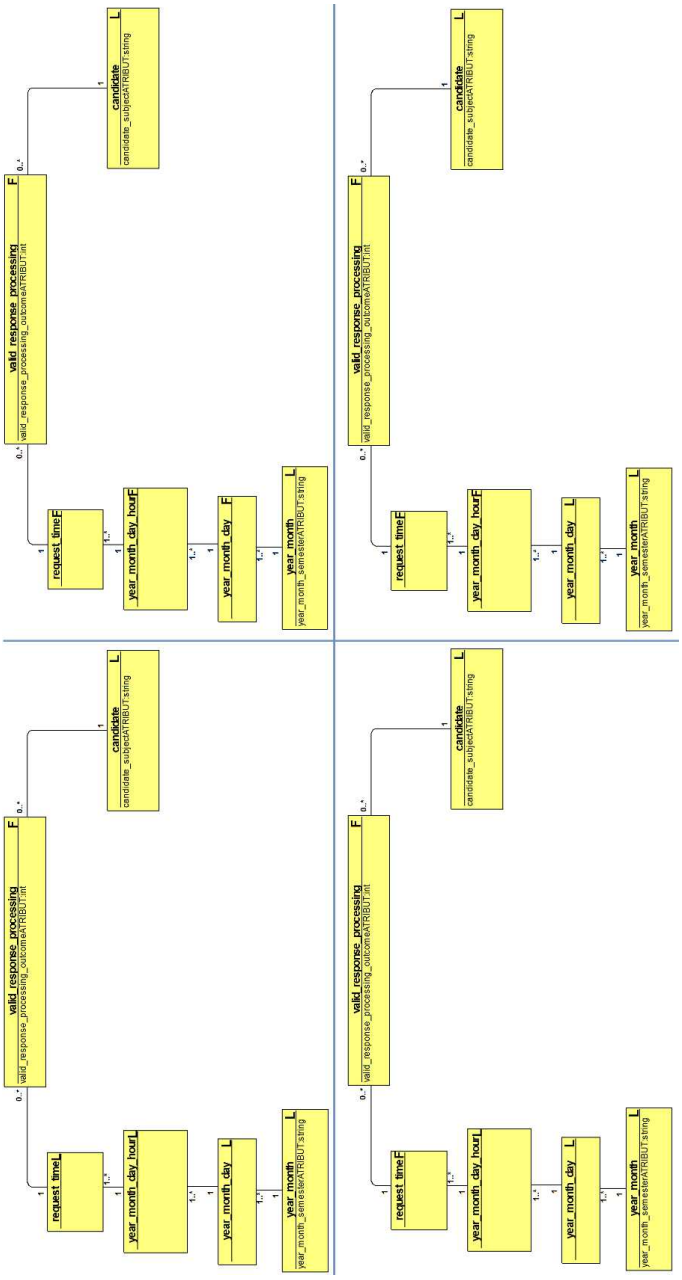


Fig. 4.19: *MD*Is resulted from *IR*

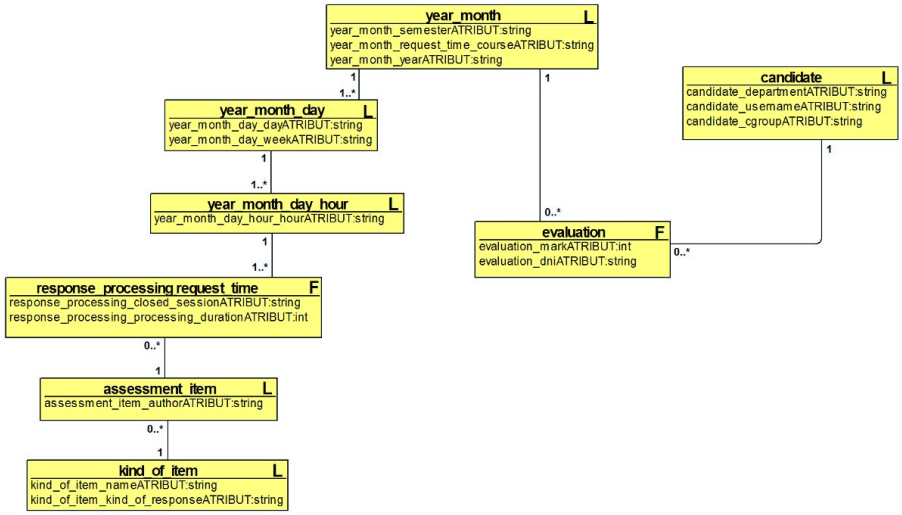


Fig. 4.20: Reference MD schema

Background analysis. The first group of six lecturers are database and conceptual modeling experts. However, four users were non experts when it comes to DW and ETL technical issues. This was an excellent opportunity to see how our methods could work with business users, who are familiar with the domain and the requirements, but sometimes lack the technical expertise to perform some pre-processing steps that would facilitate and advance their business analysis. Also, only one of them had seen LEARN-SQL internals before, which in a sense resembles practice where not all analysts are initially familiar with the system under the analysis. The second group of six users are graduate students following the DW course during their master studies and thus not well experienced users. However, as being part of the DW course, they showed a strong commitment towards the activity and wanted to perform well. The first and second rounds were used to smooth out these differences among our users and a second interviewing round afterwards showed that all twelve users were feeling comfortable with the technologies used in this experiment.

An expected observation among these two groups is the time/effort needed to finalize the given tasks. In average the students needed around twice as much time in comparison to lecturers (experts) (see Figure 4.21).

Design evaluation. Next, we evaluated the solutions resulting from the given assignments (see Figure 4.22). We focus here on the results of Assignment 2 (A2), which in a way resembles the work that ORE facilitates through its algorithms. A2 required that the participants looked for potential fact match-

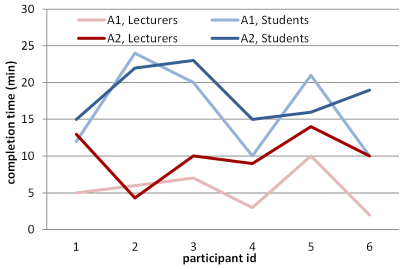


Fig. 4.21: Completion time for correct designs

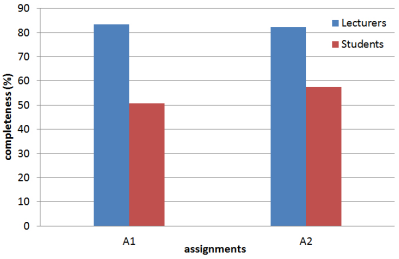


Fig. 4.22: Completeness of solutions with fixed time

ing. In principle, given that the `valid_response_processing` fact (in the new requirement) is a subclass of `response_processing` (which appears in the reference MD schema) users tended to integrate the new requirement through these facts. However, after a detailed analysis, they were supposed to recognize at least one more promising alternative: merging `evaluation` (in the reference MD schema) with the `valid_response_processing` (in the new requirement) as in the top-right MDI in Figure 4.19. These two facts share the same MD space (`year_month` × `candidate`) and thus, the facts could be properly combined.

Nevertheless, these were not the only two valid solutions. Indeed, the results provided for this assignment showed us that even a small example like this can produce many valid solutions, from which we should find the one with the lowest structural complexity. Figure 4.22 compares the level of completeness for lecturers and students. Note that among all participants only one lecturer provided solutions that were above 90% close to the ideal solution. We also observed that the average completeness percentage for the lecturers was around 83% and for the students it was lower - around 56%.

Along with the previously discussed difference in the amount of effort (i.e., elapsed time), the noticeable difference of the two groups (lecturers and students) in the quality and accuracy of the provided solutions showed us that the DW design problem is indeed complex and it requires high level of experience and expertise to achieve satisfactory results.

The most common error made by the participants was the attempt to match `valid_response_processing` with `response_process` by means of the ontology taxonomy they belong to. Such a matching may result in a valid solution that answers the new requirement, but as previously discussed, it is not the one with the lowest overall cost, in terms of structural complexity as an example quality factor, as it introduces new concepts to the design. Furthermore, intuitively, this solution should be disregarded because we are merging two facts at different granularities when there is another fact at the same granularity (i.e.,

`evaluation`).

Summary. These empirical tests aimed at simulating the environment where the users are supposed to provide the design of MD schema starting from the input information requirements. We started with a moderately small problem size and introduced time limits to simulate the complexity of larger DW design problem (the complexity of a task is related to the time given to complete it). As observed from the results, even for a small scenarios, the number of possible design solutions is fairly high. Moreover, the best solutions do not always appear to be the most obvious ones. Additionally, after filling a simple questionnaire at the end, most of the participants (including DB experts) were not confident about having found the best solution. After combining these findings with the results we obtained from scrutinizing ORE's prototype (i.e., low overall latency and high quality of generated results), we can conclude that the automation of the DW design is a necessity for dynamic business environments.

7 Related Work

Following a monolithic approach for building a DW is considered problematic and thus, manual guidelines were given to overcome this issue (e.g., DW Bus Architecture [97]). Some recent works (e.g., [71]) also study how the modern software engineering methodologies can be applied in the DW context. The authors in [71] define a set of methodological principles that should be followed during the design process and inspect their impact on different quality factors (e.g., reliability, robustness, etc.). Apart from traditional DW designing approaches (e.g., [67, 80, 115]), various works have studied the problem of adjusting the DW systems to changes of the business environments. For dealing with such an issue, different directions have been followed.

DW Evolution. The early approaches that fall into this category (e.g., [175, 172, 173] consider the DW as a set of materialized views over the source relations and propose the algorithms for either maintaining the existing set of views (i.e., view maintenance, [175]) or selecting new views to be materialized (i.e., view selection, [172, 173]), in order to answer new queries. Since a pure relational approach has been followed, these works use the equivalence transformation rules to enhance the integration of new queries into the existing view set. That makes these approaches not easily applicable to the current heterogeneous environments. In fact, as it has become clear in the last decade, DW are more complex than just a set of materialized views, especially when the ETL flows come into play. Additionally, these approaches mainly consider two traditionally correlated costs in DW design: (1) view maintenance cost, and (2) query evaluation cost. ORE, regarding new business oriented environments, complements these approaches by taking into account other quality

factors that additionally minimize the end-user's efforts (e.g., the structural complexity and understandability of the MD schema). Other DW evolution approaches (e.g., [21, 183]) maintain the up-to-dateness of the existing DW schemata in the event of a change by proposing a set of evolution operators (e.g., for addition/deletion of dimensions/levels/facts or their instances). Some (e.g., [135, 187]) additionally study the influence of different evolution changes on the quality factors of the DW (e.g., consistency and completeness). Similar problems have been studied for ETL flows too (e.g., [129]). One contribution of these works is the formal definition of specific alteration operators, which can be applied when an evolution change occurs. However, these works do not study how the information requirements actually affect such evolution changes. In this sense, our work complements them and starting from a given set of information requirements, it aims to automatically derive the changes of the current schema necessary to be applied for satisfying these new requirements.

Schema Versioning. Beside dealing with the changes by upgrading the existing schema, schema versioning approaches have also focused on keeping the trace of these changes by separately maintaining different versions of the schema (e.g., [17, 23, 66]). Some of them (e.g., [17]) in addition to the versions resulting from real world changes, also store and maintain the alternative versions which can simulate various operational/business scenarios and propose new analytical perspectives. Specifically, [66] deals with the problem of cross-version querying and introduces the concept of augmented schema, which keeps track of change actions to enable answering the queries spanning the validity of different versions. ORE also systematically keeps traces of the changes occurred so far, by using the TM structure (see Section 3). Contrarily, by maintaining TM, ORE aims at reproducing the final solution and/or potentially choosing an alternative integration option, but at any moment, the resulting MD schema must answer all the current requirements.

Incremental DW design and DW schema Integration. There are works that have studied the problem of incremental DW design (e.g., [122]). The authors here propose the approach for incremental design of DW by deriving new facts and dimensions from the existing ones by applying the set of deriving operations (i.e., dimension and measure transformation algebras). However, they do not specifically discuss the influence of information requirements nor their automatic integration into the DW design. On the schema integration side, there are works that use ontologies, to bridge the semantic gap among heterogeneous data (e.g., [167]). To deal with the integration of heterogeneous DW schemas, [179] proposes two approaches: loosely and tightly coupled. But, this work assumes that a structural matching among schemas exists and proposes the d-chase procedure (inspired by the chase procedure) for the chase of dimensional instances to ensure the consistency of the given matching.

Overall, the importance of information requirements into the DW design process has been generally overlooked. One recent work [112], proposes a

method to deal with the maintenance of a DW by identifying a set of semantic mappings between data sources and user requirements. In a sense, we find the motivation behind this work complementary to ours. However, the process of identifying such mappings as well as integrating them into the current design requires an extensive manual effort. In our work, we go a step further and automate a large part of this process as we discussed in the previous sections. Another exception is the work in [121] that starts from OLAP requirements expressed in sheet-style tables and later translates them to single star schemas. However, the integration proposed is based solely on union operations applied to the facts and dimension hierarchies. Moreover, this work does not consider the heterogeneity and complex relations that may exist in DW environments.

8 Conclusions and Future Work

In this chapter, we have presented ORE, a method to iteratively design the MD schema of a data warehouse from requirements. To the best of our knowledge, there is no other similar work dealing with such integration, which differs from ontology or generic integration research papers in that the MD integrity constraints are preserved and considered quality objectives are met.

This work represents an important step for fulfilling our final goal, i.e., to deliver a system for providing an end-to-end, requirement-driven solution for DW design problem. Our work builds on top of the GEM system [146]. GEM starts with the information requirements at hand and generates for each requirement separately a respective MD design along with the ETL operations that build the data flow. In another work [88], we present CoAl, an approach to deal with the problem of integrating the ETL processes from single information requirements.

In this chapter, we present our approach to incrementally design the MD schema from individual requirements. Starting from single requirements, we obtain a set of MD interpretations of the sources to answer such requirement (e.g., GEM). Incrementally, we build a unified MD schema satisfying the current set of requirements. At the same time, the details about the previous integration steps are traced by means of metadata to allow broader integration possibilities when new requirements arrive.

Clearly, both the MD schema and the ETL constructs are highly correlated. Our overall work is the first that considers them together. We plan to invest on that and that opens several possible future directions. As one example, it would be interesting to exploit the interdependence between ORE and CoAl through which each of them benefit from the relevant information inferred by the other approach. For instance, the aggregation and normalization levels of the produced schema could be considered, since this would effect the way the appropriate ETL process is tailored (i.e., trade-offs between materialized

views and OLAP querying). Similarly, checkpointing or bottlenecks detected at the ETL level may cause some changes at the MD schema for the sake of performance.

9 Acknowledgements

This work has been partly supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2011-24747.

Chapter 5

Engine Independence for Logical Analytic Flows

The paper has been published in the
Proceedings of the 30th IEEE International Conference on Data Engineering, pp. 1060-1071 (2014). The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1109/ICDE.2014.6816723

Abstract

A complex analytic flow in a modern enterprise may perform multiple, logically independent, tasks where each task uses a different processing engine. We term these multi-engine flows hybrid flows. Using multiple processing engines has advantages such as rapid deployment, better performance, lower cost, and so on. However, as the number and variety of these engines grows, developing and maintaining hybrid flows is a significant challenge because they are specified at a physical level and, so are hard to design and may break as the infrastructure evolves. We address this problem by enabling flow design at a logical level and automatic translation to physical flows. There are three main challenges. First, we describe how flows can be represented at a logical level, abstracting away details of any underlying processing engine. Second, we show how a physical flow, expressed in a programming language or some design GUI, can be imported and converted to a logical flow. In particular, we show how a hybrid

flow comprising subflows in different languages can be imported and composed as a single, logical flow for subsequent manipulation. Third, we describe how a logical flow is translated into one or more physical flows for execution by the processing engines. The chapter concludes with experimental results and example transformations that demonstrate the correctness and utility of our system.

1 Introduction

In a modern enterprise, answering a business question may require a complex, analytic data flow that integrates datasets and computation from a number of diverse repositories and processing engines. Conceptually, one may consider the flow as a single, logical computation and it may be modeled as such. However, a logical flow has many possible implementations, each serving a different purpose. The job of the flow designer is to create an implementation (or physical flow) that meets objectives for the flow and workload. But, over time, objectives may change, data volumes may increase rendering an implementation sub-optimal, the underlying infrastructure may change, or the logical flow may need modification. Creating and modifying physical flows is labor-intensive, time-consuming, and error prone. Because enterprises are now deploying a wide variety of systems, such as Map-Reduce systems, stream processing systems, statistical analysis engines, and even elastic computing, the trend is toward more of these complex, hybrid analytic flows. This will only increase the development and maintenance burden on IT departments.

What is needed is a notion of engine independence for logical analytic flows. Just as logical data independence insulates a data modeler from physical details of the relational database, there are benefits in designing flows at a logical level and using automation to implement the flows.

In this chapter, we present a system that does this. We focus on three main challenges. First, we describe a language for encoding flows at a logical level. Second, we show how an existing, physical flow written for one processing engine, is imported and converted to a logical flow that is engine independent. Third, given a logical flow, we show how to generate a physical flow (and executable code) for a targeted processing engine. These physical to logical and logical to physical translations also support hybrid flows, i.e., flows that involve multiple engines. Our flow translators are components in a larger system, called Hybrid Flow Management System (HFMS), that includes modules for design, optimization, and execution of complex analytic flows [164]. Other tools, like ETL design GUIs, offer some separation between design and implementation, but the design is tool-specific. Our work goes beyond that. HFMS logical flows span engines and, most importantly, the engines are peers enabling data and function shipping between all.

The logical, engine-independent flow gives a unified, end-to-end view of the entire analytic computation. From this logical view, there is a number of possible and practical flow transformations. These may alter the flow design, but not its semantics (functionality). HFMS allows a flow processor to manipulate a logical flow between the physical to logical and logical to physical translations. Optimizing the logical flow is one possible transformation [163]. Or, one might decompose a single, large, complex flow into smaller subflows to reduce contention in a workload or to improve maintainability of the flow [164]. Conversely, one might compose a series of individual, connected flows into one large flow to improve performance. Or, a flow processor might generate documents about the flow, either as pseudo-code or as a natural language description [151].

In this chapter, our focus is not on specific flow processors. Rather, our point is that a logical view of a flow simplifies and enhances flow processors. In fact, flow translation can be useful by itself. A not unusual scenario is to have an algorithm encoded for one engine (e.g., written in Map-Reduce) that you wish to apply to data in a different engine (e.g., database). Rather than ship the data to the algorithm, our flow translators enables shipping the algorithm to the data. Hence, our core contribution is that, by showing how to transform physical flows to logical flows and back, we enable new computations on hybrid flows that would otherwise be difficult to program over the original physical flows.

The next section formalizes the translation problem. Section 3 presents an overview of our system, including encoding of flows and dictionaries for mapping between logical and physical elements. Section 4 describes the physical to logical translation process and Section 6 describes logical to physical. Section 5 presents an evaluation of our system through use cases of flows running over three processing engines. The final sections discuss related work and conclusions.

2 Problem Formalization

2.1 Preliminaries

We represent an analytic flow Γ as an acyclic, parameterized digraph $\Gamma(U_\Gamma) = (V_\Gamma(U_\Gamma), E_\Gamma)$, where U_Γ is a finite set of properties of the vertices V_Γ of Γ. V_Γ are either operators, V_{op}, or data stores, V_{ds}, and the edges E_Γ model the data flow among the vertices. A special class of operators includes the connectors, $V_{cn} \subset V_{op}$, which are discussed in Section 4. The vertex properties capture information related to business requirements, Q, resource allocation, R, and characteristics, $\mathcal{C}$, like the vertex type $\mathcal{C}_{type}$ (e.g., sentimentMiner, join), implementation type $\mathcal{C}_{impl}$ (e.g., merge-sort join), engine used $\mathcal{C}_{eng}$ (e.g., Hadoop, database), etc. Therefore, the properties of a vertex v_j in

Γ are $U_\Gamma^j = Q_\Gamma^j \cup R_\Gamma^j \cup C_\Gamma^j$.

In general, the vertices of an analytic flow Γ may be assigned to a multiplicity of engines. The set of all engines used in Γ is represented as $\Phi_\Gamma = \bigcup_j C_{\Gamma eng}^j$, for all vertices $v_j \in V_\Gamma$. Connected vertices assigned to the same engine constitute a subflow of Γ. Hence, Γ may comprise a partially ordered set of such subflows $I_K = \{G_i\}$, $1 \leqslant i \leqslant K$ (K being the size of the set), each one having vertices assigned to a single engine. Each of these subflows is an acyclic digraph $G_i=(V_i, E_i)$. Their partial order in Γ is defined by the reachability of the flow. The reachability relation of Γ is the transitive closure of its edges set E_Γ, i.e., the set of all ordered pairs (x, y) of its vertices in V_Γ for which there exist vertices $v_1=x$, ..., $v_{|V_\Gamma|}=y$, such that $(v_{j-1}, v_j) \in E_\Gamma$, for all $1 < j \leqslant |V_\Gamma|$. Depending on the structure of Γ, we have three types of flows.

Definition 1
A analytic flow is a multi-flow iff $|I_K| > 1$.

Definition 2
A multi-flow is a hybrid flow iff $|\Phi_\Gamma|>1$.

Each of the K subflows of a multi-flow is called a single flow. An analytic flow itself may be a single flow too.

Definition 3
A analytic flow is a single flow iff $|I_K|=1$.

In what follows, we use the terms flow and graph interchangeably. We do the same for single flow and subflow.

2.2 Logical and physical flows

A logical flow is independent of an execution engine. The graph G_L representing such a flow contains vertices that do not necessarily have resource allocation information or some of their characteristics completed (e.g., implementation type).

A physical flow is a flow that is designed for a specific execution engine; e.g., specific RDBMS, specific map-reduce engine, specific ETL engine. The graph G_P representing such a physical flow contains the information needed to bound an operator for example to a specific implementation and engine.

2.3 Normalized flow

A flow running on an engine is expressed in a language, L, that the engine can execute. This language may be programming code or even metadata that the engine interprets. Given n languages, we would need $n \times (n-1)$ parsers to convert one language to another. We follow a different approach: we introduce

an intermediate language, L_N, that all other languages should be converted to first. Hence, we reduce the number of parsers needed to $2 \times n$. A previous use of this idea goes back to the early days of NL processing [152].

L_N has the following characteristics. It describes flows, their operators (schemata, semantics) and the interconnection among them. It captures additional operational properties at the flow and operator levels like resources required and physical characteristics. It can also represent various levels of abstraction. We use L_N as our logical flow language. Hence, a physical flow expressed in a language L_i is translated first to L_N and from there, it can be converted back to the same or to another language. In Section 3, we present an implementation of L_N, called xLM. This implementation, as we describe shortly, allows keeping logical constructs in L_N and also L_i constructs, i.e., engine specific details for multiple engines.

Definition 4
We call engine agnostic xLM, denoted as a-xLM, the xLM encoding of a logical flow G_L.

Definition 5
We call engine specific xLM, denoted as s-xLM, the xLM encoding of a physical flow G_P.

2.4 Dictionary

To enable conversions from one language to another and from physical to logical flows and vice versa, we keep a dictionary of mappings, D_M, between logical constructs and their valid physical incarnations in the supported languages. Example logical constructs are operators, functions, expressions, and data types. We denote as C_{L_i}, a construct expressed in a language L_i; e.g, for a logical construct we write C_{L_N}. We have two types of D_M mappings: (a) D_{MS}: (C_{L_i},C_{L_N}), these are, in general, one-to-one mappings used in physical to logical conversion, in order to identify the logical counterpart in L_N of a physical construct in L_i; and (b) D_{MO}: (C_{L_N},$\{C_{L_i}\}$), these are, in general, one-to-many mappings that map a logical construct to different physical implementations. (In Section 4.1, we discuss more complicated cases where more than one physical operator map to a single logical operator.)

In Section 3, we describe an implementation of D_M and Figure 5.5 shows example dictionary entries.

2.5 Conversion process

For converting a physical flow to a logical one and vice versa, we are using a mapping system defined as follows.

Definition 6
A mapping system is a triplet ($\mathcal{S}$,$\mathcal{T}$,$\mathcal{M}_{\mathcal{S},\mathcal{T}}$), where: $\mathcal{S}$ is a source graph, $\mathcal{T}$ is a target graph, and $\mathcal{M}_{\mathcal{S},\mathcal{T}}$ is a mapping between $\mathcal{S}$ and $\mathcal{T}$.

Having a physical graph, G_P, of a flow, we can use a mapping $\mathcal{M}_{G_P,G_L}$ to convert it to a logical graph G_L, by applying the mapping onto all its vertices V_P. An implementation of $\mathcal{M}_{G_P,G_L}$ may be: $\mathcal{M}_{G_P,G_L}(optype, engine, impl)$ that can be used to probe the dictionary D_M for getting a logical operator corresponding to the physical implementation *impl* of the physical operator *optype* in the engine *engine*. Hence, starting from source code we can use $\mathcal{M}_{G_P,G_L}$ to produce an engine agnostic graph encoded in a-xLM.

A reverse mapping, $\mathcal{M}_{G_L,G_P}$, converts a-xLM to s-xLM. If a specific implementation of an operator is not defined in the mapping, then the system chooses the most efficient one according to the cost model used; if none exists, the system propagates the error to the user. Note that the s-xLM produced can be expressed in a language different from the one used in the original flow. We also define a mapping $\mathcal{M}_{G_P,G'_P}$ for changing the implementation of operations in a physical flow.

Finally, the composition of mappings is allowed. For example, the composite mapping:

$$\mathcal{M}_{G_{P_2},G'_{P_2}}(\mathcal{M}_{G_L,G_{P_2}}(\mathcal{M}_{G_{P_1},G_L}))$$

describes how a physical flow G_{P_1} expressed in a language L_1 can be first converted to a logical flow G_L, then to a physical flow G_{P_2} expressed in another language L_2, and then, to a different incarnation G'_{P_2} that uses alternative implementation for a subset of its operations on the same engine.

2.6 Problem statements

Problem 1 (Physical to Logical)
Given an analytic flow Γ, we construct a logical flow G_L as follows:

$$G_L = cmp(\bigcup_{1 \leqslant i \leqslant K} \mathcal{M}_{G_{P_i},G_{L_i}}),\ \forall G_i \in I_K$$

That is, given an analytic flow Γ and a physical representation G_P of it, we produce a logical flow by converting first all single physical flows to logical flows, and then, we compose these flows to a unified, logical flow G_L. Flow composition, *cmp*, is described and solved in Section 4.2.1. □

Problem 2 (Logical to Physical)
Given a logical flow G_L, we construct an analytic flow Γ' as follows:

$$\Gamma' = \bigcup_{1 \leqslant j \leqslant K'} \mathcal{M}_{G_{L_j},G_{P_j}},\ \forall G_{L_j} \in dcmp(G_L)$$

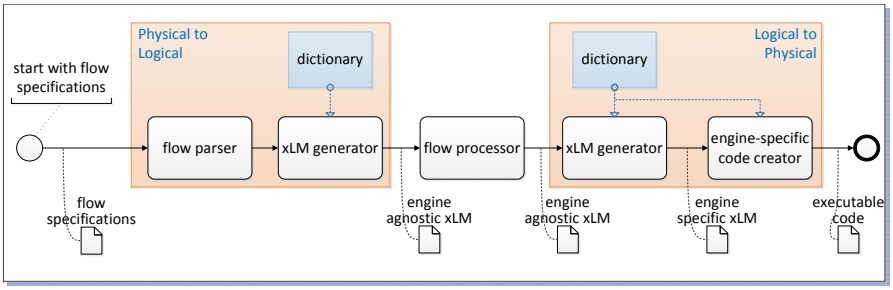


Fig. 5.1: Architecture of our solution

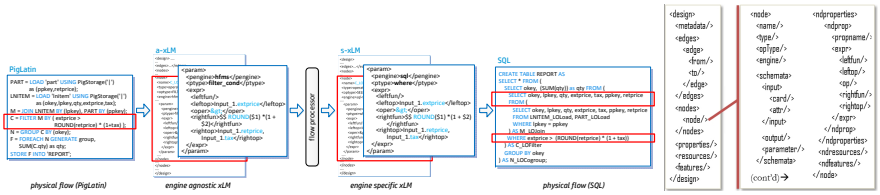


Fig. 5.2: Example process for translating a PigLatin script to SQLFig. 5.3: xLM elements

That is, given a logical flow G_L corresponding to a flow Γ, we produce a semantically equivalent multi-flow Γ', by decomposing first the flow G_L to single, logical flows G_{L_j}, $1 \leqslant j \leqslant K'$, each designed to run on a single engine, and then, we convert each G_{L_j} to a physical flow G_{P_j}. The poset of all G_{P_j} comprises Γ'. Notice that in general: (a) Φ'_Γ may differ from Φ_Γ, as Γ' may run on the same or a different set of engines than the original flow Γ; and (b) the number of single flows K in Γ and K' in Γ' may be different. Flow decomposition, *dcmp*, is described and solved in Section 6.1.1. □

The two problems may be connected or not; i.e., starting from an analytic flow, the end goal might be to produce only its logical abstraction (Problem 1) or to produce another implementation of it (a combination of both Problems 1 and 2).

3 Architecture

This section describes our system architecture and implementations of our flow language, L_N, and the dictionary, D_M.

3.1 System overview

An overview of our approach is illustrated in Figure 5.1. We start with the flow specifications (e.g., a script in a programming language or metadata that encodes a physical flow) and we first convert it to a logical graph, encoded in

engine agnostic xLM, a-xLM. This is the task of the 'physical to logical' module. There, we first parse the flow and then, we produce an xLM representation of the flow constructs using the dictionary. During parsing, it is possible, as we describe next, to collect statistics and cost estimates for the flow and its operations. Note also that the original flow may comprise more than one subflow (e.g., scripts) that may be written or be expressed in more than one programming language or forms. These are all translated into a single logical flow.

Next, a flow processor may transform the logical flow. Example processing modules are a flow optimizer (e.g., [163], [165], [162]), a collection statistics module (e.g., [163]), a flow execution scheduler (e.g., [164]), and so on. Detailing the different flow processors is out of the scope of this chapter.

The 'logical to physical' module converts the engine agnostic, logical flow into an engine specific flow according to the engine selections made either by a flow processor or a flow administrator and using the xLM mappings stored in the dictionary. In some cases, the engine specific flow may be further processed by a flow processor; e.g., to apply engine specific optimizations to the physical flow (not shown in Figure 5.1). Finally, an 'engine specific code generator' module translates the engine specific xLM to executable code that can be dispatched to the processing engines.

3.2 Example

Figure 5.2 depicts a simple example of flow translation. In this scenario, we start with a PigLatin script, parse it and with the help of the dictionary, we translate the PigLatin operators to engine agnostic operators expressed in xLM (a-XLM). Then, a flow processor may change the flow. Finally, we produce engine specific, SQL-specific here, xLM (s-xLM) and from there, we generate a SQL incarnation of the original script. The figure highlights the process for an example filter operation. We discuss this process in more detail in the rest of this chapter.

3.3 Flow encoding

HFMS uses xLM to serve as its language L_N [160]. xLM is a flow metadata language expressed, in its current implementation, in XML. It captures structural information of a flow, design metadata (e.g., functional and non-functional requirements, physical characteristics like resource allocation, positional information), operator properties (e.g., type, schemata, statistics, parameters and expressions needed for instantiating an operator, engine and implementation details, physical characteristics like memory budget), and so on.

xLM encodes a DAG and supports a rich set of operators, like relational algebra, analytic, machine learning or ETL-like operators. For the moment,

```
<node>                              (cnt'd)
<type>filter</type>                 <param>
...                                    <pengine>sql</pengine>
<param>                                <ptype>where</ptype>
   <pengine>hfms</pengine>             <expr>...</expr>
   <ptype>filter_cond</ptype>       </param>
   <expr>...</expr>                 <...>
</param>                            </node>
```

Fig. 5.4: Multiple language representations of a node in xLM

however, our flow processors do not address fixpoints or iterative computations over a set of operators. In addition, HFMS treats operators with unknown or incomplete semantics as black-boxes. Processing flows containing black-box operators might not be optimal (e.g., optimization actions would be rather conservative), but it would be at minimum correct, respecting the semantics of the data flow and the schemata involved.

The two main xLM structural components are design and node. Figure 5.3 shows a skeleton of design (left) and node (right). Design describes a flow as a graph with its vertices and edges. It also describes flow properties, resources used, and features (e.g., location coordinates of the GUI elements) captured as expressions; e.g., we express 'timeWindow=2h' and 'max(failures)=3' as:

```
'timeWindow = 2h':              'max(failures) = 3':
<leftop>timeWindow</leftop>     <leftfun>max</leftfun>
<oper>=</oper>                  <leftop>failures</leftop>
<rightop>2h</rightop>           <oper>=</oper>
                                <rightop>3</rightop>
```

Node describes a flow vertex with its name, type, operational type, engine, implementation, schemata, properties, resources, and features. The vertex type denotes if it is an operator or a data store. The operational type (*<opType>*) denotes the functionality of the operator; e.g., aggregator, tokenizer, sentiment miner. Flow vertices have input, output, and parameter schemata. Each schema represents a set of fields (*<attr>*) that have name, type, and properties (e.g., format, unit). Additional elements captured are properties like selectivity, throughput, path location; resources like i/o cost, allocated memory; and features like design coordinates.

xLM encodes both the logical and physical flows. Engine/language specific constructs can be nested in the corresponding element. For example, the xLM node for the filter of Figure 5.2 may contain multiple entries for its filter condition as in Figure 5.4. Based on the engine chosen for this filter, during flow conversion we may use the engine agnostic snippet (*pengine=hfms*, we use 'hfms' as a label of logical constructs) or the engine specific snippet (*pengine=sql*, here, 'sql' stands for generic SQL code, but it is also possible to specify a specific database engine) to produce the appropriate flow semantics. Beside expressions, the same logic is also followed to encode other engine spe-

cific elements like schemata attribute properties (e.g., data types) in different languages.

3.4 Dictionary

A logical operator may have multiple physical implementations either on a single engine or across multiple engines. For example, a join operator can be implemented as a nested loop or hash join in a database and as a replicated or skewed join in PigLatin. In order to achieve engine inter-operability and preserve flow semantics across multiple engines, we need a means for translating engine specific characteristics from one engine to another. To deal with this, we implemented a dictionary of mappings. In addition to keeping information useful for code interpretation and generation, the dictionary also contains attributes that can be used during flow processing, like the operator cost models specific to an implementation and engine.

The dictionary comprises: (a) categories; (b) language specific mappings; and (c) operation mappings (see Figure 5.5-left).

Categories describe in a machine-processable way the dictionary structure and mapping types. The mappings connect the different incarnations of flow constructs for multiple engines. Categories are used as an index and allow changing the dictionary at runtime without affecting our system's operation.

For preserving flow semantics across engines, we handle different data types, expressions, and operators with the language specific mappings in physical to logical conversion and the operation mappings in logical to physical translation.

The language specific mappings implement the D_{MS} mappings and are engine specific, as they capture the intrinsic characteristics of an engine and map them to a-xLM. Many engines separate the logical operator names from the internal, physical name corresponding to a specific implementation. For example, PigLatin FILTER translates into LOFilter when the script code translates into an execution plan. There is also a variety of representations for functions and operands used in expressions across engines; e.g., the function ROUND is invoked in PigLatin as a call to a library (org.apache.pig.builtin.ROUND). Differences also occur among data types, and thus, we convert engine specific data types to logical data types.

The operation mappings implement the D_{MO} mappings and describe the logical operations supported; e.g., operator and data store types. Figure 5.5-right shows an example entry in the dictionary for a FILTER operator, which has multiple implementations per engine and across engines. An operator entry has associated attributes, including: a logical operator name (e.g., `'hfms.name'= 'FILTER'`), a link to a cost model for computing the operator's cost, and template structures for the translation of the operator to a physical implementation. Figure 5.5-right shows example implementations: in PigLatin, `Filter`; in SQL, `Selection`; and in Pentaho PDI, an open source ETL tool, two implementa-

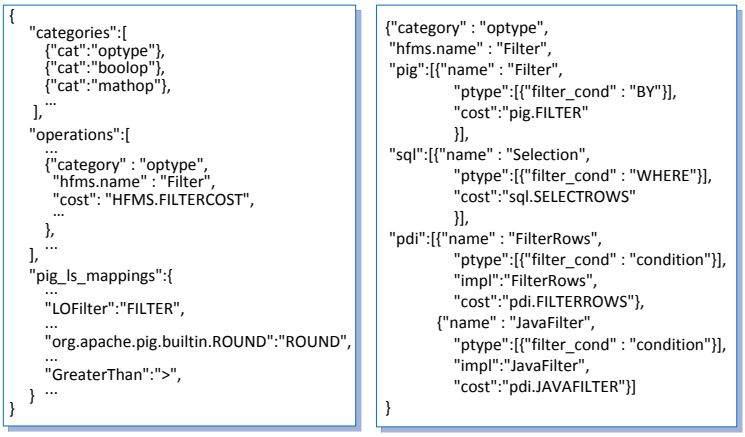


Fig. 5.5: Example entries in the dictionary

tions of filter, `FilterRows` and `JavaFilter`. The physical implementation details can be used in code generation or by a flow processor; e.g., for choosing the appropriate cost model for an operator. Other attributes stored for operators include links to code templates or implementation specific, physical properties like whether an operator is order preserving, parallelizable, streaming, etc.

Implementation. The dictionary can be implemented in various ways. In our implementation, we use a single file in JSON format; but other formats are straightforward to use.

Maintenance. Modifying the dictionary is a semi-automated process. It is extensible to new engines and implementations. For adding a new language or modifying an existing one, we use a template dictionary instance. When we finish entering the details for the new/updated language, then we use an automated mechanism for updating the dictionary accordingly.

3.5 Error handling

Translation failures may occur for various reasons; e.g., unavailable mappings or non-supported operations in an engine, connection or machine failures, runtime errors especially as we probe an engine to get an explain plan or run a sample flow to get runtime statistics. To the extent possible, we catch these errors and propagate them back to the user.

Errors such as incorrect dictionary mappings are harder to catch. Akin to proving a compiler is correct, it is hard to formally prove correctness properties. However, we can provide the user with a crude test-and-learn mechanism [38] to validate a mapping through experimentation with example data. In Section 7.2, we discuss experiments on correctness.

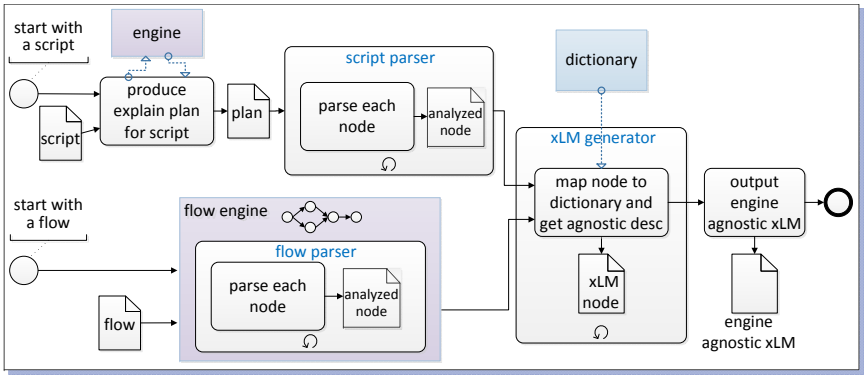


Fig. 5.6: Import single flow

4 Physical to Logical

This section deals with Problem 1 and describes how we convert a physical, analytic flow into an engine agnostic, logical flow. We distinguish two cases: single flows and multi-flows. Hybrid flows come as a variation of multi-flows.

4.1 Single flow

Converting a physical, single flow G_P to a logical flow G_L is described by a mapping $\mathcal{M}_{G_P,G_L}$. The main challenge here is to understand the semantics of a physical flow or else, of a script of execution code, and map it to a logical representation. For solving this, we first present how we parse execution code and convert it to xLM (see also Figure 5.6). Another challenge is how we deal with many-to-one mappings as in some languages a computation may require a different number of operators than in others. We can solve this either explicitly using the dictionary or implicitly as we discuss shortly.

One approach for parsing flow specifications would be to use a language specific parser to parse its source code. For some flows, like Map-Reduce programs, STORM programs, R scripts, etc., this might be the only solution. For several languages, e.g., SQL, there exist third-party parsers. In general, writing a language specific parser can be cumbersome. However, if the system provides an explain plan[1], as do database engines or Hadoop languages like PigLatin, we parse the plan instead of the code. We prefer using the explain plan because it is easier to parse and, coming from the engine, we know that it is syntactically correct. In addition, it provides extra information, which is not easy or sometimes even impossible to find by parsing the source code directly, such as cost

[1] An explain plan shows the physical operators and data flow an engine uses to execute a flow.

estimates per operator, input data sizes, operator implementation, etc. Our script parser analyzes the explain plan (or the source code if the plan is not available) and gets engine specific information for each operator or data store of the flow. The script parser is engine specific and is added to the system as a plug-in.

The script parser pipelines information for each operator or data store to the xLM generator. This generator probes the D_M dictionary to map physical to logical operators. According to the mappings found, the engine specific information is replaced accordingly. For example: the engine specific operator type is transformed to a logical operator; an engine specific expression is transformed to the form supported by the dictionary; the cost or data size estimates are used to feed the appropriate, implementation specific cost models that are essential for flow processing (e.g., optimization); and so on. Sometimes there are one-to-one mappings from one engine to another or to xLM. However, there are cases where the mappings are more complicated. For example, in PigLatin we can use two operators to specify aggregation (see Figure 5.2):

```
N = GROUP C BY (okey);
F = FOREACH N GENERATE group,SUM(C.qty)as qty;
```

Other programming languages perform the same calculation with a single operator. For example, in SQL we write:

```
SELECT SUM(qty)as qty FROM lnitm GROUP BY okey;
```

Many-to-one mappings can be resolved explicitly with a specific dictionary mapping or implicitly. For example, there are one-to-one mappings in the dictionary that map PigLatin.GROUPBY and PigLatin.FOREACH_GENERATE to logical GROUPER and PROJECT operators and from there to SQL.GROUPBY and SQL.PROJECT, respectively. Our language specific parser (i.e., the 'xLM flow processor' module in Figure 5.9) is enriched with additional smarts: when a logical GROUPER operator is followed by a logical PROJECT operator and iff the involved schemata match (e.g., PROJECT only uses the grouping attributes and the aggregates from the GROUPER) and the combination is valid, then we combine the two operators into one, generalized GROUPER. For no valid mappings, the process halts and asks for directions.

When all nodes have been processed, we output the engine agnostic a-xLM. This represents a logical flow abstraction.

Parsing the flow specification as described above is one way to create a logical flow. We use an alternative approach when an analytic flow has been created by a flow design tool like an ETL or a workflow tool. Instead of a script parser, it involves the engine itself (see left, bottom corner of Figure 5.6). This is more effective when access to the engine codebase is permitted. For example, we implemented this method in PDI, where we over-write the default printer method so that, for every flow node, we get the respective information (e.g., logical and physical) and pipeline it to the xLM generator [165]. The subsequent steps are as before.

4.2 Multi-flow import

If the input flow comprises a multiplicity of single flows, possibly written in different languages, we work as follows. To abstract away the intrinsic characteristics of each language, we first process each single flow separately to create their logical counterparts. Then, we deal with three problems: (a) identify appropriate connect points that link single flows within a multi-flow, (b) choose appropriate connector operators, and (c) compose them into a single flow. Formally, this process is defined as a composition, cmp, over the physical to logical mappings: $cmp(\bigcup_{i \in I_K} \mathcal{M}_{G_{P_i}, G_{L_i}})$.

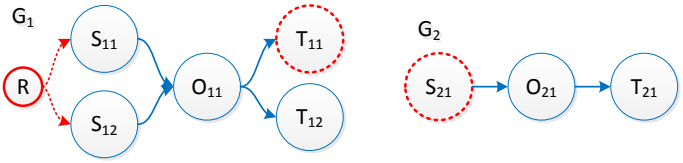
4.2.1 Composition

To deal with these problems, we formulate flow composition as a series-parallel graph (SPG) composition problem [49]. SPGs are created by composing two-terminal graphs (TTG). We may have parallel or series composition of two TTGs. The former produces a new TTG created from the disjoint union of the two graphs by merging their sources and their sinks to create a new source and sink, respectively, for the new TTG. The latter produces a TTG created from the disjoint union of the two graphs by merging the sink of the first graph with the source of the second.

Formally, a TTG is a triple (G, s, t), where $G = (V, E)$ is an acyclic digraph and s,$t \in V$; s is called a source and t a sink of G. Assume two TTGs $((V_1, E_1), s_1, t_1)$ and $((V_2, E_2), s_2, t_2)$. Assuming $t_1 = s_2$ (reads t_1 connects to s_2), the connect point is defined as $V_1 \cap V_2 = \{t_1\}$. Then, the series composition of the two TTGs is the TTG: $((V_1 \cup V_2, E_1 \cup E_2), s_1, t_2)$. Assuming $t_1 = t_2$ and $s_1 = s_2$ as connect points (i.e., $V_1 \cap V_2 = \{s_1, t_1\}$), the parallel composition of the two TTGs is the TTG: $((V_1 \cup V_2, E_1 \cup E_2), s_1, t_1)$.

In our case, a graph may have more than one source and/or sink vertices. However, it can be seen as a TTG if we work as follows. Considering a series of flows in a partial order, we begin with the first and last flows. If the first flow has more than one source, we add a dummy root vertex and connect the flow sources to the root. This dummy vertex has no semantics and used only for the composition. We do the same for the last flow if it has more than one sink vertex. (We work similarly if the partial order defines more than one 'first' or 'last' flows.) For all the other flows in the series, we define a single connect point between two connecting flows at a time. Hence, any flow in the series can now be seen as a TTG.

The above drawing shows an example involving two graphs G_1 and G_2. G_1 has two source nodes and by adding a dummy root, R, we create a single point of entry. For connecting G_1 to G_2 we need to identify a connect point. Assuming this is defined between vertices T_{11} and S_{21} –e.g., these two may represent the same physical storage, like a file– we are able to identify the TTGs involved in



this composition as the graphs $R \rightarrow \cdots \rightarrow T_{11}$ and $S_{21} \rightarrow \cdots \rightarrow T_{21}$.

We identify connect points in two ways. First, these can be explicitly defined by the user, e.g., in the form of metadata. Alternatively, these points can be inferred based on an analysis of the input flows. When the flows contain connectors (described shortly), these may connect flows to the same data storage, which is then automatically used as a connect point; e.g., one flow writes to a file or a table and a following flow reads from it. We can also discover compatible vertices between two flows, based on the similarity and compatibility of their output (for vertices in the first flow) and input (for vertices in the second flow) schemata. We process pairs of flows based on their execution order[2] and search for possible matchings between the sink vertices of the preceding flow and the source vertices of the following flow. Exact matches are candidate connect points. For approximate matches, we perform conservative schema refactoring by changing the names of the inputs and/or outputs. An application of this technique has been used for demonstrating composition of PigLatin scripts [151]. User feedback is requested if no match is found. Advanced schema matching and mapping techniques [136, 99] could be used too, but we consider this is as an interesting future direction.

Note that flow composition is not shown in Figure 5.6, which only shows import for single flows. For multi-flows, a 'flow composer' module analyzes the individual, logical graphs produced and performs the composition as discussed.

4.2.2 Connectors

As we identify candidate connect points, we also determine appropriate operators to realize the connections. At a logical level, a link between single flows is modeled in the flow with a connector operator. At a physical level, connectors delimit subflow boundaries and, possibly, engine barriers, i.e., operators on either side of a connector may run on different engines. Based on the connection semantics, a number of connection types are supported: pipeline, blocking, check point, and control point (Figure 5.7). Check points can be useful for recoverability or synchronization. Control points moderate the data flow according to a condition; e.g., wait until x% of the data has arrived, send batches

[2] Single flows may come as different scripts with metadata determining their execution order. Without such a metadata, the user defines the order. Here, without loss of generality we assume that the execution order is provided.

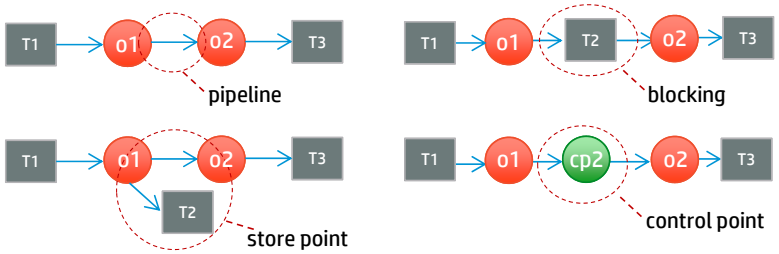


Fig. 5.7: Connection types

every y time units, etc. As a syntactic sugar, in the first three cases, an explicit connector operator may be omitted from the flow graph design.

As said, given a multi-flow, the connectors may be specified explicitly, through metadata associated with the flow, or determined implicitly. Implicit connectors are located during the physical to logical conversion by looking for operator patterns. Figure 5.8 illustrates two example types of subflow connectivity.

Figure 5.8-left shows an inter-engine connection from a PigLatin subflow to a SQL subflow. The connector executes on the consumer subflow and reads data from an HDFS file (many db providers offer such connectors). In such cases, the connect point between the two subflows can be inferred by looking into their sink and source nodes as discussed.

Figure 5.8-right shows two subflows where no connectors have been specified nor can any be inferred. In this case, we analyze the terminal nodes of the graphs and if we identify compatible vertices, we add an appropriate connector. If no match is found, then we ask the user to resolve the ambiguity. Suppose the user specifies a connector between HDFS $file1$ ($F2$) and database table $T1$. The figure shows two possible connectors. We can automatically enrich the producer subflow with a connector at its output, which will eventually be converted to PigLatin during code generation –see Section 6; this is shown in bold, red in the (I) solution. Alternatively, we can add extra operators packaged as glue code to read from the storage point of the first flow, $file1$, and load data to the matching, entry point of the second flow, $T1$; the (II) solution includes the parts of the scripts encircled by the blue dotted line. The choice between the two solutions depends on parameters like availability (not all engines support explicit connectors), costs, or user choice. Regarding costs of different connecting paths, we use a cost model based on a combination of micro-benchmarks and runtime statistics monitoring the system status (e.g., disk i/o, network congestion). A discussion on the costs is outside the scope of this chapter (more details in [163]).

To uniformly capture the different cases in $\mathcal{M}_{G_P,G_L}$, metadata for logical connectors includes semantics of these cases. A flow processor may exploit it

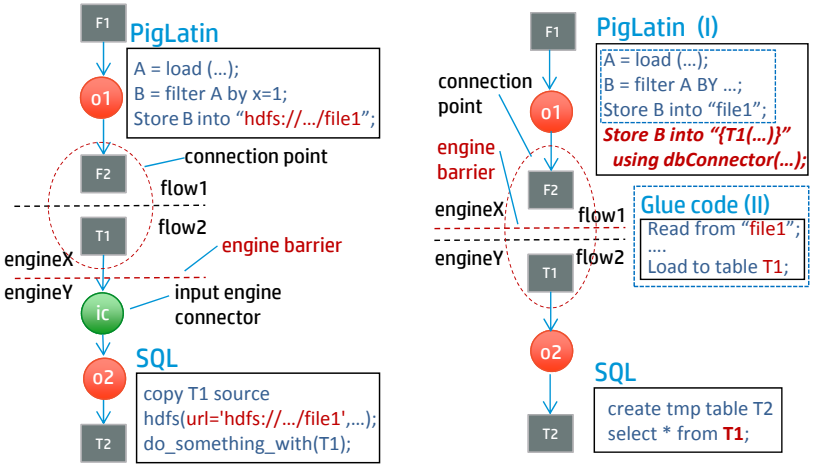


Fig. 5.8: Example connectors for hybrid flows

as needed. This metadata is also used in converting a logical connector to a physical connector. The metadata is captured as xLM properties like: pipeline or blocking connections; disk or memory storage; the type of the encapsulated data stores for blocking connectors; the flow lineage, in which single flow a part of the multi-flow originally belongs to; file paths; db connections; partitioning and clustering schema; etc. As an aside, in some cases it is reasonable to convert a blocking connector (e.g., a storage object) to a pipelined connector. For example, if the storage object is a temporary table in a SQL script and it only has one reader, then the flow that produces the data can be pipeline connected to the flow that consumes the table. But we need to be careful, since there may be other objects (e.g., other flows) that may use this storage.

5 Flow Processor

A flow processor takes as input a logical flow graph, performs some transformation on that graph, and produces as output a second logical graph that is functionally equivalent to the input but with different properties.

As discussed, the HFMS optimizer is one example of a flow processor. Another flow processor might be used to decompose a long flow into subflows or to compose multiple subflows into a single flow. A more detailed analysis of the modules that handle flow processing can be found elsewhere [164].

In the context of this chapter, an important function of a flow processor is to alter connectors. For example, flow composition would remove connectors. Flow decomposition would add connectors. Flow optimization may include function shipping, i.e., moving an operator from one engine to another, which

involves swapping the position of an operator and a connector. Retargeting a single flow from one engine to another involves modifying the metadata for the connector to the new engine. Once the connectors are determined, so are the engine boundaries. Since connectors link subflows on different engines, given the engine assignment, a physical flow can be generated. Note that the engine assignment can also be defined manually by a system administrator.

6 Logical to Physical

This section deals with Problem 2 and describes how we convert an engine agnostic, logical flow first into a physical, engine specific flow, and then, to executable code that can be dispatched to execution engines.

6.1 Creating an engine specific flow

A logical graph is engine agnostic, a-xLM. A flow processor determines on what engine(s) it will be executed and thus, we convert a-xLM to s-xLM using this information. If the entire flow has been assigned to a single engine, we proceed directly to the conversion. If the operators of the logical flow have been assigned to multiple engines, then before converting G_L to G_P, we decompose the logical flow, $dcmp(G_L)$, to single flows, G_{L_j}, each running on one engine (see Problem 2).

6.1.1 Decomposition

For a logical flow G_L=(V_L, E_L), the connectors between subflows assigned to different engines are engine barriers (see also Figure 5.8). For creating single flows, we split the logical graph at the engine barriers. The result of this split is a set of weak graph components, each one corresponding to a single flow. A weak component is the maximal subgraph in which all pairs of vertices in the subgraph are reachable from one another in the underlying subgraph. The weak components are stored in a list according to a topological sort of the original flow; each weak component, i.e., single flow, is prioritized according to the topological order of its vertices in the original, composed flow. These priorities will determine the execution order of the scripts produced in code generation, as we describe shortly.

6.1.2 Conversion

Converting a logical flow G_{L_j} to a physical flow G_{P_j} is described by a mapping $\mathcal{M}_{G_{L_j},G_{P_j}}$ (see also Figure 5.9). G_{L_j} is encoded in a-xLM, which describes the main flow structure, but also carries engine specific details captured during $\mathcal{M}_{G_P,G_L}$, like paths to data storage or design metadata. These details are now

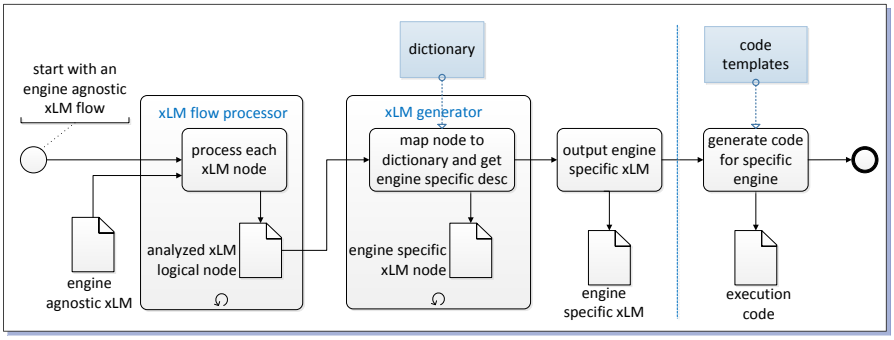


Fig. 5.9: Convert a-xLM to s-xLM and generate code

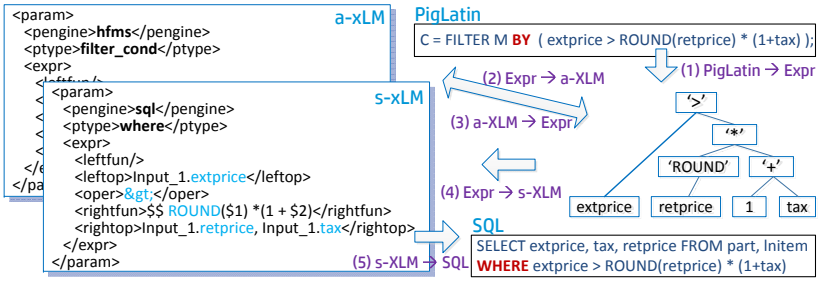


Fig. 5.10: Example expression tree usage

used for producing s-xLM. For space considerations, we omit an extensive description of the conversion of a-xLM constructs into s-xLM. The dictionary is a centerpiece in this process as it contains templates with default metadata information for operator representation in different engines as part of the language specific mappings, D_{MS}.

We elaborate on the conversion of expressions as an indicative example. Operators have parameter schemata describing their functionality. Typically, this information is represented with an expression. Expression formats vary across languages so, during the import phase, the original expressions are transformed into the logical format that our system understands. Now, we perform the reverse operation and convert them to a form that the targeted engine can process. There are two issues.

First, we determine the context of the expression. For example, an expression may describe a filter condition or may define a new projection schema; e.g., in SQL these correspond to conditions in the WHERE and SELECT clauses, respectively.

Second, apart from the metadata, we need to parse the expression itself. Expressions, like mathematic expressions or built-in functions, may be represented differently in different engines. For example, a conjunction may be

written as 'X AND Y' or 'AND(X,Y)' or 'X && Y', etc. Similarly, built-in functions may also differ from engine to engine. For example, SQRT(a) in SQL, Math.sqrt(a) in PDI, org.apache.pig.builtin.SQRT(a) in PigLatin, etc. Hence, during the conversion from physical to logical, we map the engine specific forms to logical forms as follows. Starting from an engine specific expression, we first build an expression tree, whose nodes are operators and built-in functions found in the expression, while the leaves of the tree are the attributes and constants included in the expression. Now, we reuse this tree for producing an engine specific expression (possibly for another engine). We use the dictionary to retrieve suitable mappings for the constructs of the expression (e.g., AND or &&, built-in functions) and the proper usage of such expressions (e.g., 'X AND Y' or 'AND(X,Y)').

Figure 5.10 shows a transformation for the FILTER of Figure 5.2. As discussed in Section 6, starting from PigLatin, we identify the expression describing its filter condition (step 1 in Figure 5.10), create an expression tree, and produce a-xLM (step 2). We now reuse the expression tree (step 3) for creating s-xLM for SQL (step 4), and finally, produce SQL code (step 5).

An overview of the conversion of a-xLM to s-xLM is depicted in Figure 5.9 (on the left of the vertical dashed line).

6.2 Code generation

This section describes how we generate code from s-xLM; see also Figure 5.9, on the right of the vertical dashed line.

We generate code separately for each of the decomposed, single engine specific flows; these tasks are done in parallel. In each single flow, we process the flow operations following a topological sort of the flow graph to ensure that before generating a code snippet all its prerequisites have been created. At the end, we produce orchestration code to package (e.g., as a shell script) the code pieces for execution.

Next, we describe a template mechanism that enables code generation and then, we give an example of code generation for SQL –other languages are omitted for space considerations.

6.2.1 Templates

For each operator implementation in the dictionary, we define a code template with formal parameters, which can be used to produce code for a graph construct. We distinguish two groups of templates: code and metadata templates. Code templates produce executable code directly when they are appropriately instantiated (e.g., a template for producing a SQL or PigLatin statement). Metadata templates produce meta-code interpretable by design tools, like ETL tools, which store the flow metadata in a specific format (e.g., in an XML or

JSON script). This format is later used for importing a flow into those external tools, and then at compile time, the tools create the appropriate execution code.

In our current implementation, we support a fairly large number of operators (more than a hundred) and it is relatively easy to register a new one. Our code templates are expressed in JavaScript and are invoked by a script engine at runtime. For the example FILTER of Figure 5.5, simplified example code templates for PigLatin and SQL are as follows:

```
function pig_FILTER(args) {
  return args.operator + " =\n  FILTER "
        + args.producer + "\n BY " + args.param + ";";
}
function sql_SELECTROWS(args) {
  w=(args.where=="")?"":" \nWHERE " + args.where;
  g=(args.groupby=="")?"":" \nGROUP BY " + args.groupby;
  h=(args.having=="")?"":" \nHAVING " + args.having;
  o=(args.orderby=="")?"":" \nORDER BY " + args.orderby;
  return "SELECT "+args.out+"\nFROM "+args.in+w+g+h+o+";";
}
```

When code is generated, the formal parameters are replaced by the actual parameters from the engine specific flow in order to generate a code snippet for the operator. The operator metadata also includes formal schemata for the input and output data sets, which specify the required and optional fields of each input record and the fields of the output records. In addition, there is a parameter schema that specifies the formal parameters, if any, needed for the operator. The data stores also have associated metadata, like a size estimate of the data set, as number of records, and a schema for the records. For data stores, the metadata also includes the location of the data store.

The code templates for the above, example filter operator can be instantiated as `Z = FILTER X BY Y` (PigLatin) and `SELECT Z' FROM X WHERE Y` (SQL), assuming that the filter operator has a parameter schema `Y`, a producer operation `X`, and a consumer operation `Z` with input schema `Z'`.

In addition to generating code for operators, we can also generate code that implements data flow between operators. There are two cases. If both producer and consumer operators use the same underlying execution engine, the execution form for that engine may have syntax for connecting the two operators, e.g., nested SQL clauses or pipelining Unix shell scripts. If no syntax is available, e.g., PigLatin, the output of the producer can be placed in a temporary data set which is then specified as an input to the consumer. If either the producer or the consumer is a connector, then, as we discussed in Section 4.2, connector code snippets are generated to transfer the dataset across engine.

6.2.2 Code generation - The SQL case

For space considerations, we describe code generation with code templates. We work similarly with metadata templates.

The designer may influence the style of generated code. For example, a nesting flow may run faster. But a decomposed flow splits the computation into shorter phases that use fewer resources and under certain conditions may reduce contention for system resources. Also, a designer might want to decompose a complicated, nested SQL query (e.g., perhaps a query generated by a reporting tool) into a script of sub-flows in order to resolve a data consistency issue (i.e., by exposing intermediate results). The final form of the executable depends on the designer's objectives. There are trade-offs among response time, throughput, maintainability, etc. in using these forms. In different ways and at different times, both styles can be useful.

HFMS offers an interactive environment, where a designer may explore and test alternatives for flow execution by indicating the degree of pipelining desired for the executable code. At one end of the spectrum, blocking may be specified. In this case, every operator is made into a blocking operator by simply inserting the output of the operator into a temporary table. At the other end of the spectrum, pipelining may be specified. In this case, the entire flow is generated –if possible– as a single SQL statement by using nested SQL. Note that this does not convert a blocking operator into non-blocking. For example, even sort operators are nested even though this operator is blocking. However, it enables the SQL engine to optimize the entire flow as a single statement which should produce more efficient code, but perhaps less readable or maintainable.

Between the two extremes, a user may specify pipelining that uses temporarily tables either more or less aggressively. We use heuristics to determine where to insert the temporary tables, like after a naturally occurring blocking operator, since no pipelining can occur with such an operator anyway. We also use estimates of the size of the datasets as well as available resources to determine whether to use temporary tables or not. For example a small data set may easily be buffered in memory to reduce the amount of I/O. A large data set may need to be spooled to disk even if pipelining is specified.

Under the hood, the user or a flow processor may choose a nesting strategy. We first split the flow graph at breaking points into query paths. Breaking points denote where a new subquery should start or a temporary table may be needed. Example breaking points are operators with multiple output (e.g., Router) and multiple input (e.g., Merger) paths. Then, we process each query path and construct a SQL subquery; depending on the nesting level, this may be a single query or it may contain temporary tables. If a query path contains a data storage (e.g., source table, intermediate table) then an appropriate DDL statement is constructed too.

For creating a nesting query, we use template placeholders in the SQL expression of each operator and afterwards, we replace it with an appropriately constructed subquery. For example, the SQL expressions for the operators of a subflow `T1`→`N2`→`N3` may be as follows:

```
T1: create table T1 ...; --T1 may already exist too
N2: select * from ##T1##;
N3: select * from ##N2##;
```

Following a topological sort, we resolve dependencies by replacing the placeholders `##T1##` and `##N2##` with the respective expressions. Based on the nesting level, we create the final query as follows. For maximum nesting, we have:

```
select * from (select * from T1) as N2;
```

For a lesser nesting level and assuming that `N2` is a breaking point, the templates for `N2` and `N3` would be:

```
N2: create temp table TEMP_N2 as select * from ##T1##;
N3: select * from ##N2##;
```

and the final query is:

```
create temp table TEMP_N2 as select * from T1;
select * from TEMP_N2;
```

Finally, we produce the final SQL expressions for tables, temporary tables, and the main (nested) query. Due to space considerations, we omit a formal description of this algorithm.

7 Evaluation

We demonstrate the utility of our techniques by showing representative results of our system behavior.

7.1 Preliminaries

Implementation. The system we described here comprises an API module for our hybrid flow management system (HFMS). The API is implemented in about 25K lines of Java code. The dictionary is implemented in JSON. xLM template operators are built with Apache Velocity 1.7. Code generation is performed with embedded JavaScript. For engine support, when we do not use code parsers, we parse execution plans that we get either by probing the engine (e.g., JDBC for SQL) or by using an external library of the language without connecting to the engine (e.g., PigServer for PigLatin). We also modified the open source codebase of PDI to make it xLM aware.

Methodology. In the evaluation, we focus on three languages: SQL (sql) running on a commercial DBMS, PigLatin (pig) running on Hadoop 0.20.2,

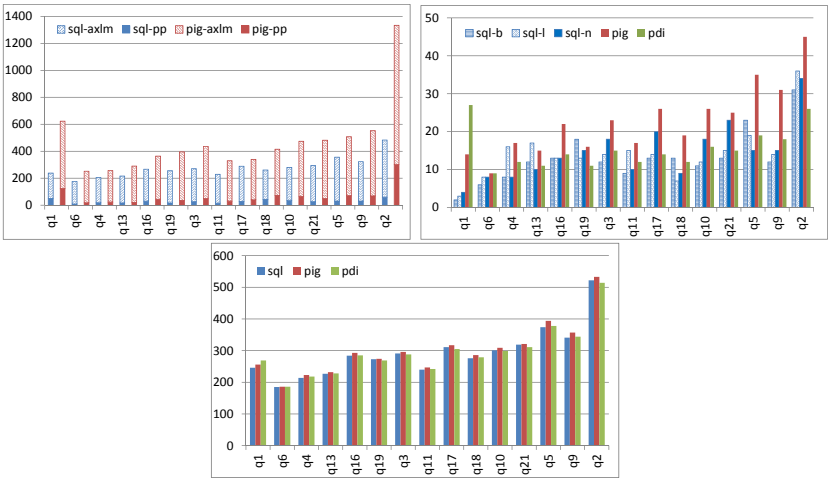


Fig. 5.11: Time analysis for 15 TPC-H flows (times in msec): (a) from SQL/Pig code to a-xLM, (b) from s-xLM to code, (c) from SQL code to xLM to code

and PDI flows (pdi) running on the community edition of Pentaho PDI 4.4.0. Our intention is to show that our techniques work for both declarative and procedural languages, but also for flows expressed as metadata.

We used flows written in all three languages implementing the logic of 15 TPC-H and 15 TPC-DS queries. We chose these as representative examples of transactional and analytical scenarios. We also used 10 custom made flows combining ETL and analytical logic and having their subflows implemented either in one (multi-flows) or in two (hybrid flows) languages.

7.2 Experiments

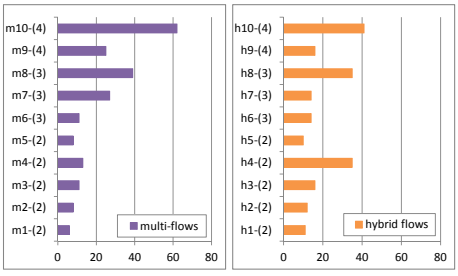


Fig. 5.12: Flow composition (times in msec)

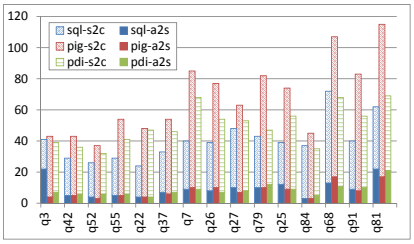


Fig. 5.13: From a-xLM to s-xLM to code (times in msec)

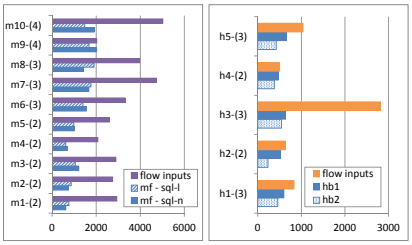


Fig. 5.14: Flow execution (times in msec)

7.2.1 Correctness

We evaluated the correctness of our methods in two ways: (a) comparing the code generated, and (b) examining the results produced by executing all scenarios used in all three code variants: SQL, PigLatin, and PDI.

Compare code. We compared the original code and execution plans (where available) to both the code produced and the execution plans the engines created for the new code. For a fair comparison, (a) we created snippets in the language they were originally written in and disabled any interaction with the flow processors, and (b) we brought the original and new snippets in the same format: same capital/small letter format, trim extra spaces and indentation, etc. We also round-tripped the translation of scripts starting from one language, converting them to the other two languages in a random order, and then back to the original language. All scripts compared were semantically equivalent. However, there were also differences as follows.

Different variable names: this is due to the code re-factoring done in composition for enabling schema matches and due to the operator and variable name substitution performed to add semantics to a script; e.g., change one-char names in PigLatin like `` `C=Join...' `` with `` `C_LOJoin=Join...' ``.

Different placing order: operators belonging to the same execution order class –i.e., operators without dependencies amongst them– may be placed differently; e.g., two operators $filter_1$ and $filter_2$ may be reordered as $filter_2$ and $filter_1$.

Different flavor of code: the code produced may have different nesting degree (e.g., number of intermediate tables).

Compare results. We ran all flows in their original configuration. We then produced new flows by varying combinations of the following attributes: target engines (single and multiple engines) and decomposition level (ranging from full nested to full blocking). For all scripts, the produced results after running code variants for the same flow were identical to the original, except, for row order, when it was not specified in the flow.

As a side note, the results we report here are indicative of our effort to

evaluate our translators. During implementation, we have gone through the language specification, translated each operator, and ran tests with it. Of course these tests are with specific input values and so, while great for catching obvious programming errors, they may miss data-dependent errors or subtle implementation differences between engines. In addition, we have not performed extensive tests to evaluate program equivalence in side effects or failure conditions.

7.2.2 Performance

We performed experiments focusing on the behavior of our system. All other features of HFMS (e.g., optimizer, workload manager) were disabled.

Physical to Logical. Figure 5.11(a) shows execution times for creating a logical graph starting from the 15 TPC-H flows written in sql and pig. For sql and pig the different phases during import are: parse a plan (pp) and create a logical graph (axlm). For pdi, since we changed its codebase to produce directly xLM, there is only one phase, axlm. For space considerations, we do not plot the results for pdi, but for the same flows they range between 5 and 20 seconds.

The flows in the figure are placed in increasing order of complexity. We consider complexity related to the flow size and type of operators performed; e.g., nesting or operators with more than one input or output schemata make parsing more complex. As seen, parsing gets slower with flow complexity.

Our parsers work differently. For sql and pig, some of the smarts in understanding the operations involved, nesting, etc. can be inferred from the plan. In pdi and when we parse code directly, some cycles are also spent for identifying the flow semantics, get extra information like runtime statistics, sizes of the involved data sets, etc. In addition, especially for pdi, some extra time (an avg 2-3sec per flow) is spent to ensure correct transfer of metadata associated with the flow design, not the semantics. A faster implementation is possible by changing the internals of PDI engine, but since we wanted to leave the lightest possible footprint and not be highly dependent from future versions of the engine, we compromised with paying an extra cost for operations not directly related to a-xLM creation.

We performed similar experiments with the 15 TPC-DS flows. The times for converting them to a-xLM starting from sql and pig follow the same trends as with the TPC-H flows. axlm dominates the import (an average of 497msec), while pp was cheaper (avg 88msec). Import for pdi was again slower, between 7 and 24 seconds, for the reasons we discussed.

We performed composition of 10 multi-flows (m) and 10 hybrid flows (h) (in pig and sql). We omit the results for each subflow import, since the trend is as before. Figure 5.12 shows the results of composing the individual subflows into a single logical graph. The y-axis shows the id of each flow and in a parenthesis, the number of its subflows, K. In most cases, composition takes less than

40msec. Time difference for same K relates to the identification of connection points (e.g., the hybrid h4-(2) comes with no pre-specified connectors).

Logical to Physical. The avg time for converting a-xLM to s-xLM, for the TPC-H flows tested was 3msec and for producing code from s-xLM was 23 for pig, 15 for pdi, 14 for sql (all times in msec). Figure 5.11(b) shows time needed to parse s-xLM and generate code for all engines pig, pdi, and sql –in three variants, blocking sql-b, moderate use of intermediate storage sql-i, and nesting sql-n. Again, the flows in the figure are placed in increasing order of complexity. In general, we produce sql faster than the other two languages, but on average, this is close to pdi code creation. For pig, we spend some extra cycles to identify n-to-m mappings and decompose operators like the GROUPER operation discussed in Section 4.1.

Figure 5.11(c) shows end-to-end time needed to convert a flow written in sql to xLM and from there either to pig, pdi or sql again. These times are pretty similar, and as the figure shows too, our system needs less than 500msec for a full translation. There is similar behavior when we start from pig (on avg less than 500msec) or pdi, with the latter being slower (5 to 20 seconds) due to the import cost of PDI flows.

We observed similar trends for the TPC-DS flows. These are more complex flows, so the conversion a-xLM to s-xLM was a bit slower, averaging 10msec for all s-xLM variants. Figure 5.13 shows the end-to-end times for the three languages tested.

For the 10 hybrid flows, the time to decompose each one to graphs assigned to pig and sql is less than 2msec. After that, code generation for each subflow is as in the other cases.

Execution. The focus here is on flow translators, not flow processors. However, an indicative example of how translation enables advanced opportunities in flow execution is shown in Figure 5.14. We tested 10 multi-flows and 5 hybrid flows comprising a varying number n of single flows, denoted as m_i-(n) and h_i-(n), respectively. Multi-flows were improved after composing them as one flow, either as sql-i or sql-n, as the database optimizes the whole. hb1 shows that just composition can improve hybrid flows, because each engine optimizes better its subflow. hb2 shows the effect of additional optimization performed to the entire flow (e.g., shipping operations from sql to pig). For space constraints, we show here only the sql part of the hybrid; pig is much slower (avg 1min) but is also affected positively by hb2. These are examples. A detailed study on the benefits in running processes as hybrid flows and in modifying a hybrid flow workload at runtime can be found in [164, 162].

7.2.3 Discussion

As seen here, and from our overall user experience, the translation is fairly fast and can be done at run time. If used by a user, the translation is interactive

and offers an 'instantaneous' feeling. If used by a flow processor, it does not add much to the latency of flow execution. This makes the flow translators a useful component of HFMS as it enables features like optimization (provides a single view of the end-to-end flow), workload management at a flow execution run time (enables new actions like flow decomposition and flow shipping when the targeted engine is unavailable or overloaded), scheduling (flexibility in code rewriting allows more options in cloud resource provisioning and scheduling), etc.

On the other hand, our translators are useful as an autonomous system too. An example use case is code migration. It can be used to migrate legacy routines to a new language (or even a new version of the language) or to convert code running on one system to another; e.g., from one database engine to another that may use different SQL syntax.

8 Related Work

Various systems generate a single type of executable code from a logical graph. Some systems generate code within a particular family of targets; e.g., database query systems generate SQL for a variety of database engines. Other systems generate executable code for more than one execution engine like ETL systems. But, these systems are inflexible in that they generate identical executable code for a given graph. Our system generates various executable forms from a single graph.

Research on federated database systems considered query processing across multiple database engines (e.g., Garlic [149], Multibase [39], Pegasus [48]). However, our work extends to a large variety of execution engines. Some systems consider hybrid execution in which queries span two execution engines, namely a database and a Map-Reduce engines; e.g., HadoopDB/Hadapt [10], PolyBase [45], Teradata Aster [58]. These are closer to our hybrid flows, but they are not designed as a general framework over multiple engines. They follow an ego-centric logic providing connectors to other engines, whereas we consider engines as equal peers.

There is research work in code generation for integration flows. Orchid converts declarative mapping specification into data flow specifications; but the focus is on ETL jobs [44]. GCIP generates code for integration systems like ETL and EAI, but not for other execution engines [27]. There is also work on SQL generation that applies simple rules to compose SQL queries (without subqueries) by identifying insert and create/drop table statements and eliminating intermediate steps [103]. Another work translates BPEL/SQL to BPEL/SQL using process graph model, but the focus is on business processes [192]. We differ for these works, since we can digest more complex flows written in other languages too, and can convert them back to multiple languages.

9 Conclusions

Our work is motivated by the emerging trend of using a diverse set of processing engines within a complex analytic flow. Implementing and maintaining these hybrid flows at a physical level is a burden and we propose using logical flows for engine independence. In this chapter, we describe how engine independence is supported by HFMS, our Hybrid Flow Management System. We show how our language, xLM, can encode flows, both logical and physical elements. We then show how a physical flow is imported into HFMS and translated to a logical flow that is not bound to a specific engine. That logical flow can then be translated back to a physical flow and then to executable code for another processing engine. These translations are enabled through a dictionary and engine specific mappings that are easy to extend to new engines and new operators. To represent links between subflows of a larger flow, HFMS has a set of connector operators that capture the semantics of the connections between subflows.

An evaluation of the HFMS translators was demonstrated using three processing engines, Hadoop/PigLatin, Pentaho PDI, and a SQL engine. Evidence of correctness is shown by round-tripping, i.e., given a flow for engine$_x$, HFMS translates it to a logical flow and then to a physical flow for engine$_x$ or engine$_y$ that produces identical results. Examples of the utility were presented including composition and decomposition.

We acknowledge there are language constructs that will be difficult to translate. For example, the semantics of user-defined functions (UDFs) vary across engines, so not all can be translated (e.g., some support only scalar, some aggregation, some are multi-valued, etc.). However, we believe our approach is sufficiently general to model and translate a sufficiently large class of analytic flows. Some of our next steps include incorporating additional engines of interest and enhancing the automation of flow composition by using advanced schema matching and mapping techniques [136, 99].

Chapter 6

H-WorD: Supporting Job Scheduling in Hadoop with Workload-driven Data Redistribution

The paper has been published in the
Proceedings of the 20th East-European Conference on Advances in Databases and Information Systems, pp. 306-320 (2016). The layout of the paper has been revised.
DOI: http://dx.doi.org/10.1007/978-3-319-44039-2_21

Abstract

Today's distributed data processing systems typically follow a query shipping approach and exploit data locality for reducing network traffic. In such systems the distribution of data over the cluster resources plays a significant role, and when skewed, it can harm the performance of executing applications. In this chapter, we address the challenges of automatically adapting the distribution

of data in a cluster to the workload imposed by the input applications. We propose a generic algorithm, named H-WorD, which, based on the estimated workload over resources, suggests alternative execution scenarios of tasks, and hence identifies required transfers of input data a priori, for timely bringing data close to the execution. We exemplify our algorithm in the context of MapReduce jobs in a Hadoop ecosystem. Finally, we evaluate our approach and demonstrate the performance gains of automatic data redistribution.

1 Introduction

For bringing real value to end-users, today's analytical tasks often require processing massive amounts of data. Modern distributed data processing systems have emerged as a necessity for processing, in a scalable manner, large-scale data volumes in clusters of commodity resources. Current solutions, including the popular Apache Hadoop [191], provide fault-tolerant, reliable, and scalable platforms for distributed data processing. However, network traffic is identified as a bottleneck for the performance of such systems [85]. Thus, current scheduling techniques typically follow a query shipping approach where the tasks are brought to their input data, hence data locality is exploited for reducing network traffic. However, such scheduling techniques make these systems sensitive to the specific distribution of data, and when skewed, it can drastically affect the performance of data processing applications.

At the same time, distributed data storage systems, typically independent of the application layer, do not consider the imposed workload when deciding data placements in the cluster. For instance, Hadoop Distributed File System (HDFS) places data block replicas randomly in the cluster following only the data availability policies, hence without a guarantee that data will be uniformly distributed among DataNodes [156]. To address this problem, some systems have provided rules (in terms of formulas) for balancing data among cluster nodes, e.g., HBase [1], while others like HDFS provided means for correcting the data balancing offline [156]. While such techniques may help balancing data, they either overlook the real workload over the cluster resources, i.e., the usage of data, or at best leave it to the expert users to take it into consideration. In complex multi-tenant environments, the problem becomes more severe as the skewness of data can easily become significant and hence more harmful to performance.

In this chapter, we address these challenges and present our workload-driven approach for data redistribution, which leverages on having a complete overview of the cluster workload and automatically decides on a better redistribution of workload and data. We focus here on the MapReduce model [43] and Apache Hadoop [191] as its widely used open-source implementation. However, notice that the ideas and similar optimization techniques as the ones proposed in this

chapter, adapted for a specific programming model (e.g., Apache Spark), could be applied to other frameworks as well.

In particular, we propose an algorithm, named H-WorD, for supporting task scheduling in Hadoop with Workload-driven Data Redistribution. H-WorD starts from a set of previously profiled MapReduce jobs that are planned for execution in the cluster; e.g., a set of jobs currently queued for execution in a batch-queuing grid manager system. It initializes the cluster workload, following commonly used scheduling techniques (i.e., exploiting data locality, hence performing query shipping). Then, H-WorD iteratively reconsiders the current workload distribution by proposing different execution scenarios for map tasks (e.g., executing map tasks on nodes without local data, hence performing also data shipping). In each step, it estimates the effect of a proposed change to the overall cluster workload, and only accepts those that potentially improve certain quality characteristics. We focus here on improving the overall makespan[1] of the jobs that are planned for execution. As a result, after selecting execution scenarios for all map tasks, H-WorD identifies the tasks that would require data shipping (i.e., transferring their input data from a remote node). Using such information, we can proactively perform data redistribution in advance for boosting tasks' data locality and parallelism of the MapReduce jobs.

On the one hand, the H-WorD algorithm can be used offline, complementary to existing MapReduce scheduling techniques, to automatically instruct redistribution of data beforehand, e.g., plugged as a guided rebalancing scheme for HDFS [2]. On the other hand, H-WorD can be used on the fly, with more sophisticated schedulers, which would be able to take advantage of a priori knowing potentially needed data transfers, and leveraging on idle network cycles to schedule such data transfers in advance, without deferring other tasks' executions.

Contributions. In particular, our main contributions are as follows.

- We present an automatic approach for workload-driven data redistribution in Apache Hadoop.
- We introduce a novel algorithm, H-WorD, which, led by the real workload in the cluster, decides on a data redistribution that would improve the performance of executing that workload.
- We evaluate our approach using well-known MapReduce benchmarks, for showing the effectiveness of the H-WorD algorithm.

Outline. The rest of the chapter is structured as follows. Section 2 introduces a running example used throughout this chapter. Section 3 discusses the motivation and presents the problem of data redistribution in Hadoop. Section

[1] We define makespan as the total time elapsed from the beginning of the execution of a set of jobs, until the end of the last executing job [22].

4 formalizes the notation and presents the H-WorD algorithm. In Section 5, we report on our experimental findings. Finally, sections 6 and 7 discuss related work and conclude this chapter, respectively.

2 Running Example

To illustrate our approach and facilitate the explanations throughout this chapter, we introduce a running example based on a set of three MapReduce WordCount[2] jobs, with different input data sets. A MapReduce job executes in two consecutive phases, namely map and reduce [43]. Map phase processes an input file from HDFS. The file is split in logical data blocks of the same size (e.g., 64MB or 128MB), physically replicated for fault tolerance, and distributed over the cluster nodes. Each data block is processed by a single map task.

Table 6.1: Example MapReduce jobs

job ID	file ID	size (MB)	#tasks	$dur^{mapTask}$ (s)	$dur^{mapInTransfer}$ (s)
1	f1	1920	15	40	6.34
2	f2	640	5	40	6.34
3	f3	1280	10	40	6.34

We profiled the example MapReduce jobs using an external tool, called Starfish [77]. Starfish can create job profiles on the fly, by applying sampling methods (e.g., while jobs are queued waiting for execution), or from previous jobs' executions. The portion of the profiles of the example jobs focusing on map tasks are presented in Table 6.1. We trace the number of map tasks, the average duration of each task ($dur^{mapTask}$), as well as the average duration of transferring its input data block over the network (i.e., $dur^{mapInTransfer}$).

Furthermore, we consider a computing cluster with three computing nodes, each with a capacity of 2CPUs and 2GB of memory, connected through the network with 100Mbps of bandwidth (see Figure 6.1). We deployed Hadoop 2.x on the given cluster, including HDFS and MapReduce. In addition, for simplifying the explanations, we configured HDFS for creating only one replica of each input data block. In Figure 6.1, we depict the initial distribution of the input data in the cluster. Note that each input data block is marked as DBX_{fid}, where X is an identifier of a block inside a file, and fid is the id of the file it belongs to.

For reasons of simplicity, we configured all example jobs to require containers (i.e., bundles of node resources) with 1CPU and 1GB of memory for accommodating each map and reduce task, i.e., mapreduce.map.memory.mb = mapreduce.reduce.memory.mb = 1024, and mapreduce.map.cpu.vcores = mapreduce.reduce.cpu.vcores = 1.

[2]WordCount Example: `https://wiki.apache.org/hadoop/WordCount`

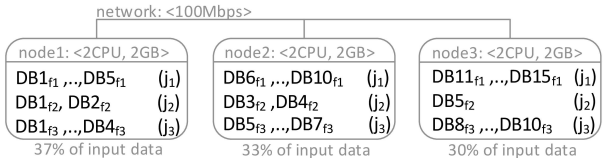


Fig. 6.1: Example cluster configuration and initial data distribution

3 The Problem of Skewed Data Distribution

We further applied the default scheduling policy of Hadoop (i.e., exploiting data locality) to our running example. An execution timeline is showed in Figure 6.2, where the x-axis tracks the start and end times of tasks and the y-axis shows the resources the tasks occupy at each moment. For clarity, we further denote a task t_i^j both with the task id i, and the job id j. Notice in Figure 6.1 that the job ids refer to groups of input data blocks that their map tasks are processing, which determines the placement of the map tasks in the cluster for exploiting data locality. First, from the timeline in Figure 6.2, we can notice that although the distribution of input data is not drastically skewed, it affects the execution of job 3, since for executing map task m_4^3, we need to wait for available computing resources on *node*1.

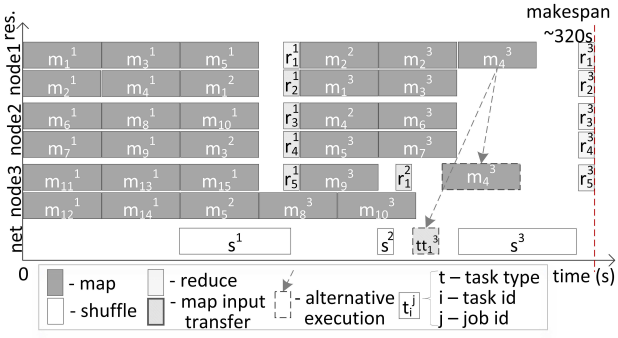


Fig. 6.2: Timeline of executing example MapReduce jobs, baseline

Furthermore, we can also observe some idle cycles on the computing resources (i.e., *node*3), that obviously could alternatively accommodate m_4^3, and finish the map phase of job 3 sooner. However, *node*3 does not contain the needed input data at the given moment, thus running m_4^3 on *node*3 would require transferring its input data (i.e., tt_1^3), which would also defer its execution (see alternative execution of m_4^3 in Figure 6.2).

Having such information beforehand, we could redistribute data in a way that would improve utilization of cluster resources, and improve the makespan.

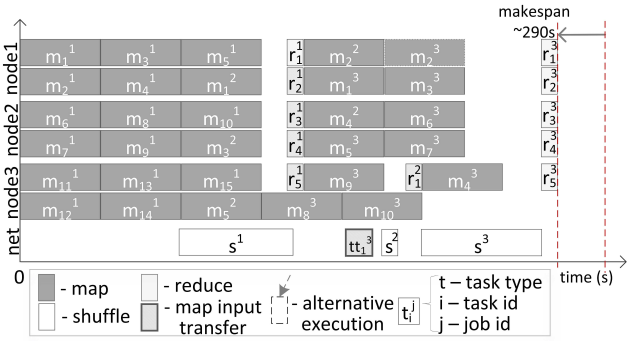


Fig. 6.3: Timeline of executing example MapReduce jobs, with data redistribution

Such data redistribution could be done offline before starting the execution of MapReduce jobs. However, note that there are also idle cycles on the network resource (e.g., between s^1 and s^2, and between s^2 and s^3). This is exactly where having more information about the imposed workload makes the difference. In particular, knowing that the higher workload of *node*1 can potentially affect the makespan of the jobs' execution, we could take advantage of idle network resources and plan for timely on the fly transferring of m_4^3's input data to another node, in overlap with other tasks' execution, and hence improve the execution makespan. Such alternative execution scenario is depicted in Figure 6.3.

We showcased here in a simple running example that in advance data redistribution can moderately improve the makespan. However, typical scenarios in Hadoop are much more complex, with larger and more complex cluster configurations, greater number of jobs, more complex jobs, and larger input data sizes. Thus, it is obvious that estimating the imposed workload over cluster resources and deciding on data and workload redistribution is intractable for humans and requires efficient automatic means. At the same time, in such real world scenarios improving resource utilization and minimizing the execution makespan is essential for optimizing the system performance.

We further studied how to automatically, based on the estimated workload, find new execution scenarios that would improve data distribution in the cluster, and hence reduce the makespan. Specifically, we focused on the following challenges:

- Resource requirements. For obtaining the workload that a job imposes over the cluster, we need to model cluster resources, input MapReduce jobs, and the resource requirements of their tasks.

- Alternative execution scenarios. We need to model alternative execution scenarios of MapReduce jobs, based on the distribution of input data in

a cluster and alternative destination resources for their tasks. Consequently, alternative execution scenarios may pose different resource requirements.

- Workload estimation. Next, we need an efficient model for estimating the workload over the cluster resources, for a set of jobs, running in certain execution scenarios.
- Data redistribution. Lastly, we need an efficient algorithm, that, using the estimated workload, selects the most favorable execution scenario, leading to a better distribution of data in a cluster, and to reducing the makespan.

4 Workload-driven Redistribution of Data

In this section, we tackle the previously discussed challenges, and present our algorithm for workload-driven redistribution of data, namely, H-WorD.

4.1 Resource requirement framework

In this chapter, we assume a set of previously profiled MapReduce jobs as input (see the example set of jobs in Table 6.1). Notice that this is a realistic scenario for batched analytical processes that are run periodically, hence they can be planned together for better resource utilization and lower makespan. For instance, in a grid manager system, a set of jobs are queued, waiting for execution, during which time we can decide on a proper distribution of their input data.

A set of MapReduce jobs is submitted for execution in a cluster, and each job j_x consists of sets of map and reduce tasks.

$$J := \{j_1, ..., j_n\},\ j_x := MT_x \cup RT_x \tag{6.1}$$

The set of all tasks of J is defined as $T_J = \bigcup_{x=1}^{n} j_x = \bigcup_{x=1}^{n} (MT_x \cup RT_x)$.

These tasks can be scheduled for execution in the cluster that comprises two main resource types, namely: computing resources (i.e., nodes; R_{cmp}), and communication resources (i.e., network; R_{com}).

$$R := R_{cmp} \cup R_{com} = \{r_1, ..., r_n\} \cup \{r_{net}\} \tag{6.2}$$

Each resource r (computing or communication) has a certain capacity vector $\boldsymbol{C}(r)$, defining capacities of the physical resources that are used for accommodating MapReduce tasks (i.e., containers of certain CPU and memory capacities, or a network of certain bandwidth).

$$\forall r \in R_{cmp}, \boldsymbol{C}(r) := \langle c_{cpu}(r), c_{mem}(r) \rangle; \forall r \in R_{com}, \boldsymbol{C}(r) := \langle c_{net}(r) \rangle \tag{6.3}$$

Each task t_i^j requires resources of certain resource types (i.e., computing and communication) during their execution. We define a resource type requirement

RTR_k of task t_i^j, as a pair $[S, d]$, such that t_i^j requires for its execution one resource from the set of resources S of type k ($S \subseteq R_k$), for a duration d.

$$RTR_k(t_i^j) := [S, d], st. : S \subseteq R_k \tag{6.4}$$

Furthermore, we define a system requirement of task t_i^j, as a set of resource type requirements over all resource types in the cluster, needed for the complete execution of t_i^j.

$$SR(t_i^j) := \{RTR_1(t_i^j), ..., RTR_l(t_i^j)\} \tag{6.5}$$

Lastly, depending on specific resources used for its execution, task t_i^j can be executed in several different ways. To elegantly model different execution scenarios, we further define the concept of execution modes. Each execution mode is defined in terms of a system requirement that a task poses for its execution in a given scenario (denoted $SR(t_i^j)$).

$$\mathcal{M}(t_i^j) := \{SR_1(t_i^j), ..., SR_m(t_i^j)\} \tag{6.6}$$

Example. The three example MapReduce jobs (job 1, job 2, and job 3; see Table 6.1), are submitted for execution in the Hadoop cluster shown in Figure 6.1. Cluster comprises three computing resources (i.e., $node1$, $node2$, and $node3$), each with a capacity of $\langle 2CPU, 2GB \rangle$, connected through a network of bandwidth capacity $\langle 100Mbps \rangle$. Map task m_1^1 of job 1 for its data local execution mode requires a container of computing resources, on a node where the replica of its input data is placed (i.e., $node1$), for the duration of 40s. This requirement is captured as $RTR_{cmp}(m_1^1) = [\{node1\}, 40s]$. □

4.2 Execution modes of map tasks

In the context of distributed data processing applications, especially MapReduce jobs, an important characteristic that defines the way the tasks are executed, is the distribution of data inside the cluster. This especially stands for executing map tasks which require a complete data block as input (e.g., by default 64MB or 128MB depending on the Hadoop version).

Data distribution. We first formalize the distribution of data in a cluster (i.e., data blocks stored in HDFS; see Figure 6.1), regardless of the tasks using these data. We thus define function f_{loc} that maps logical data blocks $DBX_{fid} \in \mathbb{DB}$ of input files to a set of resources where these blocks are (physically) replicated.

$$f_{loc} : \mathbb{DB} \rightarrow \mathcal{P}(R_{cmp}) \tag{6.7}$$

Furthermore, each map task m_i^j processes a block of an input file, denoted $db(m_i^j) = DBX_{fid}$. Therefore, given map task m_i^j, we define a subset of resources where the physical replicas of its input data block are placed, i.e., local resource set LR_i^j.

$$\forall m_i^j \in MT_J, LR_i^j := f_{loc}(db(m_i^j)) \tag{6.8}$$

Conversely, for map task m_i^j we can also define remote resource sets, where some resources may not have a physical replica of a required data block, thus

executing m_i^j may require transferring input data from another node. Note that for keeping the replication factor fulfilled, a remote resource set must be of the same size as the local resource set.

$$\forall m_i^j \in MT_J, \mathbb{RR}_i^j := \{RR_i^j | RR_i^j \in (\mathcal{P}(R_{cmp}) \backslash LR_i^j) \wedge |RR_i^j| = |LR_i^j|\} \quad (6.9)$$

Following from the above formalization, map task m_i^j can be scheduled to run in several execution modes. The system requirement of each execution mode of m_i^j depends on the distribution of its input data. Formally:

$$\forall m_i^j \in MT_J, \mathcal{M}(m_i^j) = \{SR_{loc}(m_i^j)\} \cup \bigcup_{k=1}^{|\mathbb{RR}_i^j|} \{SR_{rem,k}(m_i^j)\}, s.t.: \quad (10)$$
$$SR_{loc}(m_i^j) = \{[LR_i^j, d_i^{j,cmp}]\}; SR_{rem,k}(m_i^j) = \{[RR_{i,k}^j, d_{i,k}^{j,cmp}], [\{r_{net}\}, d_{i,k}^{j,com}]\}$$

Intuitively, a map task can be executed in the local execution mode (i.e., $SR_{loc}(m_i^j)$), if it executes on a node where its input data block is already placed, i.e., without moving data over the network. In that case, a map task requires a computing resource from LR_i^j for the duration of executing map function over the complete input block (i.e., $d_i^{j,cmp} = dur^{mapTask}$). Otherwise, a map task can also execute in a remote execution mode (i.e., $SR_{rem}(m_i^j)$), in which case, a map task can alternatively execute on a node without its input data block. Thus, the map task, besides a node from a remote resource set, may also require transferring input data block over the network. Considering that a remote resource set may also contain nodes where input data block is placed, hence not requiring data transfers, we probabilistically model the duration of the network usage.

$$d_{i,k}^{j,com} = \begin{cases} \frac{|RR_{i,k}^j \backslash LR_i^j|^2}{|RR_{i,k}^j|} \cdot dur^{mapInTransfer}, \text{if on the fly redistribution} \\ 0, \text{if offline redistribution} \end{cases} \quad (11)$$

In addition, note that in the case that data redistribution is done offline, given data transfers will not be part of the jobs' makespan (i.e., $d_{i,k}^{j,com} = 0$).

Example. Notice that there are three execution modes in which map task m_4^3 can be executed. Namely, it can be executed in the local execution mode $SR_{loc}(m_4^3) = \{[\{node1\}, 40s]\}$, in which case, it requires a node from its local resource set (i.e., $LR_4^3 = \{node1\}$). Alternatively, it can also be executed in one of the two remote execution modes. For instance, if executed in the remote execution mode $SR_{rem,2}(m_4^3) = \{[\{node3\}, 40s], [\{net\}, 6.34s]\}$, it would require a node from its remote resource set $RR_{4,1}^3 = \{node3\}$, and the network resource for transferring its input block to $node3$ (see dashed boxes in Figure 6.2). □

Consequently, selecting an execution mode in which a map task will execute, directly determines its system requirements, and the set of resources that it will potentially occupy. This further gives us information of cluster nodes that may require a replica of input data blocks for a given map task.

To this end, we base our H-WorD algorithm on selecting an execution mode for each map task, while at the same time collecting information about its resource and data needs. This enables us to plan data redistribution beforehand and benefit from idle cycles on the network (see Figure 6.3).

4.3 Workload estimation

For correctly redistributing data and workload in the cluster, the selection of execution modes of map tasks in the H-WorD algorithm is based on the estimation of the current workload over the cluster resources.

Algorithm 3 getWorkload

inputs: $SR(t_i^j)$; output: $W : R \rightarrow \mathbb{Q}$

```
1: for all r ∈ R do
2:     W(r) ← 0;
3: end for
4: for all [S,d] ∈ SR(t_i^j) do
5:     for all r ∈ S do
6:         W(r) ← W(r) + d/|S|;
7:     end for
8: end for
```

In our context, we define a workload as a function $W : R \rightarrow \mathbb{Q}$, that maps the cluster resources to the time for which they need to be occupied. When selecting an execution mode, we estimate the current workload in the cluster in terms of tasks, and their current execution modes (i.e., system requirements). To this end, we define the procedure getWorkload (see Alg. 3), that for map task t_i^j, returns the imposed workload of the task over the cluster resources R, when executing in execution mode $SR(t_i^j)$.

Example. Map task m_4^3 (see Figure 6.2), if executed in local execution mode $SR_{loc}(m_4^3)$, imposes the following workload over the cluster: $W(node1) = 40$, $W(node2) = 0$, $W(node3) = 0$, $W(net) = 0$. But, if executed in remote execution mode $SR_{rem,2}(m_4^3)$, the workload is redistributed to node3, i.e., $W(node1) = 0, W(node2) = 0, W(node3) = 40$, and to the network for transferring input data block to node3, i.e., $W(net) = 6.34$. □

Following from the formalization in Section 4.1, a resource type requirement of a task defines a set of resources S, out of which the task occupies one for its execution. Assuming that there is an equal probability that the task will be scheduled on any of the resources in S, when estimating its workload imposed over the cluster we equally distribute its complete workload over all the resources in S (steps 4 - 8). In this way, our approach does not favor any specific cluster resource when redistributing data and workload, and is hence agnostic to the further choices of the chosen MapReduce schedulers.

4.4 The H-WorD algorithm

Given the workload estimation means, we present here H-WorD, the core algorithm of our workload-driven data redistribution approach (see Alg. 4).

Algorithm 4 H-WorD

inputs: MT_J

1: $todo \leftarrow MT_J$;
2: for all $r \in R$ do $W(r) \leftarrow 0$; end for
3: for all $t \in MT_J$ do
4: $SR_{cur}(t) \leftarrow SR_{loc}(t)$;
5: $W_t \leftarrow \text{getWorkload}(SR_{cur}(t))$;
6: for all $r \in R$ do
7: $W(r) \leftarrow W(r) + W_t(r)$;
8: end for
9: end for
10: while $todo \neq \emptyset$ do
11: $t \leftarrow \text{nextFrom}(todo)$; $todo \leftarrow todo \backslash \{t\}$;
12: $SR_{new}(t) \leftarrow SR_x(t) | \text{q}(W + \Delta_{x,cur}) = \underset{SR_j(t) \in \mathcal{M}(t) \backslash \{SR_{cur}(t)\}}{\min} \left\{ \text{q}(W + \Delta_{j,cur}) \right\}$
13: if $\text{q}(W) > \text{q}(W + \Delta_{new,cur})$ then
14: $SR_{cur}(t) \leftarrow SR_{new}(t)$;
15: $W \leftarrow W + \Delta_{new}$;
16: end if
17: end while

H-WorD initializes the total workload over the cluster resources following the policies of the Hadoop schedulers which mainly try to satisfy the data locality first. Thus, as the baseline, all map tasks are initially assumed to execute in a local execution mode (steps 2 - 9).

H-WorD further goes through all map tasks of input MapReduce jobs, and for each task selects an execution mode that potentially brings the most benefit to the jobs' execution. In particular, we are interested here in reducing the execution makespan, and hence we introduce a heuristic function q(W), which combines the workloads over all resources, and estimates the maximal workload in the cluster, i.e., $\text{q}(W) = max_{r \in R}(W(r))$. Intuitively, this way we obtain a rough estimate of the makespan of map tasks executing in the cluster. Using such heuristic function balances the resource consumption in the cluster, and hence prevents increasing jobs' makespan by long transfers of large amounts of data.

Accordingly, for each map task, H-WorD selects an execution mode that imposes the minimal makespan to the execution of input MapReduce jobs (Step 12). The delta workload that a change in execution modes ($SR_{cur} \rightarrow SR_{new}$) imposes is obtained as: $\Delta_{new,cur} = \text{getWorkload}(SR_{new}(t)) - \text{getWorkload}\ (SR_{cur}(t))$.

Finally, for the selected (new) execution mode $SR_{new}(t)$, H-WorD analyzes if such a change in execution modes actually brings benefits to the execution of input jobs, and if the global makespan estimate is improved (Step 13), we assign the new execution mode to the task (Step 14). In addition, we update

the current total workload over the cluster due to changed execution mode of the map task (Step 15).

Example. An example of the H-WorD execution is shown in Table 6.2. After H-WorD analyzes the execution modes of task m_4^3, it finds that the remote execution mode $SR_{rem,2}(m_4^3)$ improves the makespan (i.e., 440 → 400). Thus, it decides to select this remote execution mode for m_4^3. □

Table 6.2: H-WorD algorithm: example of the improved makespan for task m_4^3

Workload	Initial	...	After task m_4^3	...
W(node1)	440	...	400	...
W(node2)	400	...	400	...
W(node3)	360	...	400	...
W(net)	0	...	15	...
Makespan: q(W)	440	...	400	...

It should be noted that the order in which we iterate over the map tasks may affect the resulting workload distribution in the cluster. To this end, we apply here a recommended longest task time priority rule in job scheduling [22], and in each iteration (Step 11) we select the task with the largest duration, combined over all resources. H-WorD is extensible to other priority rules.

Computational complexity. When looking for the new execution mode to select, the H-WorD algorithm at first glance indicates combinatorial complexity in terms of the cluster size (i.e., number of nodes), and the number of replicas, i.e., $|\mathbb{RR}_t| = \frac{|R_{cmp}|!}{(|R_{cmp}|-|LR_t|)!\cdot|LR_t|!}$. The search space for medium-sized clusters (e.g., 50-100 nodes), where our approach indeed brings the most benefits, is still tractable (19.6K-161.7K), while the constraints of the replication policies in Hadoop, which add to fault tolerance, additionally prune the search space.

In addition, notice also that for each change of execution modes, the corresponding data redistribution action may need to be taken to bring input data to the remote nodes. As explained in Section 3, this information can either be used to redistribute data offline before scheduling MapReduce jobs, or incorporated with scheduling mechanisms to schedule input data transfers on the fly during the idle network cycles (see Figure 6.3).

5 Evaluation

In this section we report on our experimental findings.

Experimental setup. For performing the experiments we have implemented a prototype of the H-WorD algorithm. Since the HDFS currently lacks the support to instruct the data redistribution, for this evaluation we rely on simulating the execution of MapReduce jobs. In order to facilitate the simulation of MapReduce jobs we have implemented a basic scheduling algorithm, following the principles of the resource-constrained project scheduling [100].

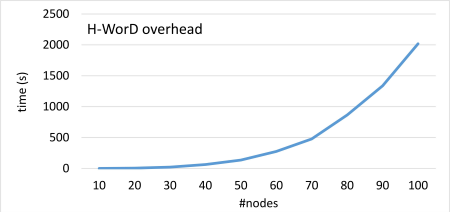


Fig. 6.4: H-WorD overhead (skew: 0.5, #jobs: 9)

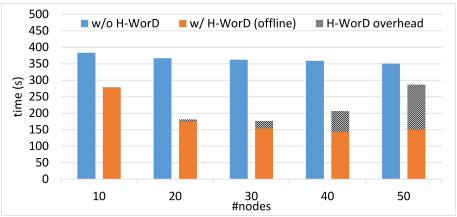


Fig. 6.5: Performance gains - #nodes (skew: 0.5, #jobs: 9)

Inputs. Besides WordCount, we also experimented with a reduce-heavy MapReduce benchmark job, namely TeraSort[3]. We started from a set of three profiled MapReduce jobs, two WordCount jobs resembling jobs 1 and 2 of our running example, and one TeraSort job, with 50 map and 10 reduce tasks. We used the Starfish tool for profiling MapReduce jobs [77]. When testing our algorithm for larger number of jobs, we replicate these three jobs.

Experimental methodology. We scrutinized the effectiveness of our algorithm in terms of the following parameters: number of MapReduce jobs, initial skewness of data distribution inside the cluster, and different cluster sizes. Notice that we define skewness of data distribution inside a cluster in terms of the percentage of input data located on a set of X nodes, where X stands for the number of configured replicas. See for example 37% skewness of data in our running example (bottom of Figure 6.1). This is important in order to guarantee a realistic scenario where multiple replicas of an HDFS block are not placed on the same node. Moreover, we considered the default Hadoop configuration with 3 replicas of each block. In addition, we analyzed two use cases of our algorithm, namely offline and on the fly redistribution (see Section 4.4). Lastly, we analyzed the overhead that H-WorD potentially imposes, as well as the performance improvements (in terms of jobs' makespan) that H-WorD brings.

Scrutinizing H-WorD. Next, we report on our experimental findings.

Note that in the presented charts we analyzed the behavior of our algorithm for a single parameter, while others are fixed and explicitly denoted.

Algorithm overhead. We first analyzed the overhead that the H-WorD algorithm imposes to scheduling of MapReduce jobs. Following the complexity discussion in Section 4.4, for small and medium-sized clusters (i.e., from 20 to 50 nodes), even though the overhead is growing exponentially (0.644s → 135.68s; see Figure 6.4), it still does not drastically delay the execution of input MapReduce jobs (see Figure 6.5).

Performance improvements. We further report on the performance improve-

[3]TeraSort: `https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html`

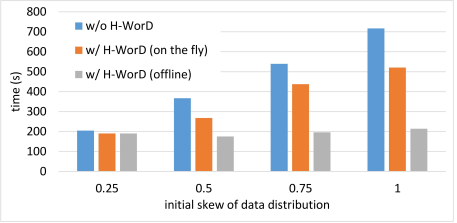


Fig. 6.6: Performance gains - data skewness (#nodes: 20, #jobs: 9)

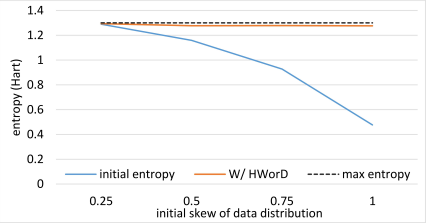


Fig. 6.7: "Correcting" skewness - entropy (#nodes: 20, #jobs: 9)

ments that H-WorD brings to the execution of MapReduce jobs.

- Cluster size. We start by analyzing the effectiveness of our approach in terms of the number of computing resources. We can observe in Figure 6.5 that skewed data distribution (50%) can easily prevent significant scale-out improvements with increasing cluster size. This shows another advantage of H-WorD in improving execution makespan, by benefiting from balancing the workload over the cluster resources. Notice however that the makespan improvements are bounded here by the fixed parallelism of reduce tasks (i.e., no improvement is shown for clusters over 40 nodes).

- "Correcting" data skewness. We further analyzed how H-WorD improves the execution of MapReduce jobs by "correcting" the skewness of data distribution in the cluster (see Figure 6.6). Notice that we used this test also to compare offline and on the fly use cases of our approach. With a small skewness (i.e., 25%), we observed only very slight improvement, which is expected as data are already balanced inside the cluster. In addition, notice that the makespan of offline and on the fly use cases for the 25% skewness are the same. This comes from the fact that "correcting" small skewness requires only few data transfers over the network, which do not additionally defer the execution of the tasks. However, observe that larger skewness (i.e., 50% - 100%) may impose higher workload over the network, which in the case of on the fly data redistribution may defer the execution of some tasks. Therefore, the performance gains in this case are generally lower (see Figure 6.6). In addition, we analyzed the effectiveness of our algorithm in "correcting" the data distribution by capturing the distribution of data in the cluster in terms of a Shannon entropy value, where the percentages of data at the cluster nodes represent the probability distribution. Figure 6.7 illustrates how H-WorD effectively corrects the data distribution and brings it very close ($\Delta \approx 0.02$) to the maximal entropy value (i.e., uniform data distribution). Notice that the initial entropy for 100% skew is in this case higher than 0, since replicas are equally distributed over 3 cluster nodes.

- Input workload. Lastly, we analyze the behavior of our algorithm in terms of the input workload, expressed in terms of the number of MapReduce jobs. We observed (see Figure 6.8) that the performance gains for various input load sizes are stable (∼48.4%), having a standard deviation of 0.025. Moreover, notice that data redistribution abates the growth of makespan caused by increasing input load. This demonstrates how our approach smooths the execution of MapReduce jobs by boosting data locality of map tasks.

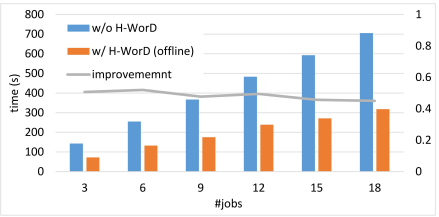


Fig. 6.8: Performance gains - workload (skew: 0.5, #nodes: 20)

Lastly, in Figure 6.5, we can still observe the improvements brought by data redistribution, including the H-WorD overhead. However, if we keep increasing the cluster size, we can notice that the overhead, although tractable, soon becomes severely high to affect the performance of MapReduce jobs' execution (e.g., 2008s for the cluster of 100 nodes). While these results show the applicability of our approach for small and medium-sized clusters, they at the same time motivate our further research towards defining different heuristics for pruning the search space.

6 Related Work

Data distribution. Currently, distributed file systems, like HDFS [156], do not consider the real cluster workload when deciding about the distribution of data over the cluster resources. By default, data are distributed randomly, without a guarantee that they will be balanced. Additional tools, namely balancer, are provided to balance data offline. However, such balancing is still done blindly, without considering the real usage of such data by the given workload.
Data locality. Hadoop's default scheduling techniques (i.e., Capacity [3] and Fair [4] schedulers), typically rely on exploiting data locality in the cluster, i.e., favoring query shipping. Moreover, other, more advanced scheduling proposals, e.g., [85, 197], to mention a few, also favor query shipping and exploiting data locality in Hadoop, claiming that it is crutial for performance of MapReduce jobs. The approach in [197] in addition proposes techniques that address the conflict between data locality and fairness in scheduling MapReduce jobs. For

achieving higher data locality, they delay jobs that cannot be accommodated locally to their data. While we agree that data locality is essential for boosting the execution of MapReduce jobs, these approaches overlook the fragileness of such techniques to skewed distribution of data in a cluster.
Combining data and query shipping. To address such problem, other approaches (e.g., [72, 194]) propose combining data and query shipping in a Hadoop cluster. In [72], the authors investigate on data locality, and claim that having a global overview of the executing tasks, rather than one task at a time as in current techniques, gives better opportunities for optimally scheduling tasks and selecting local or remote execution. [194], on the other side, uses a stochastic approach, and builds a model for predicting a workload over the cluster resources, when deciding on data locality for map tasks. We find the ideas and techniques for estimating resource workload of these approaches motivational for our work. However, these techniques do not leverage on the estimated workload to perform in advance data transfers and additionally boost data locality for map tasks.

Finally, the first approach that tackles the problem of adapting data placement to the workload is presented in [127]. We find this work especially interesting for our research. The authors argue for the benefits of having a data placement aware of the cluster workload. However, while the authors propose an efficient framework for load-aware data placement, they consider data placements for single jobs, in isolation. In addition, they propose different placement techniques depending on the job types. We, on the other side, propose more generic approach relying only on an information gathered from job profiles, and consider a set of different input jobs at a time.

7 Conclusions and Future Work

In this chapter, we have presented H-WorD, our approach for workload-driven redistribution of data in Hadoop. H-WorD starts from a set of MapReduce jobs and estimates the workload that such jobs impose over the cluster resources. H-WorD further iteratively looks for alternative execution scenarios and identifies more favorable distribution of data in the cluster beforehand. This way H-WorD improves resource utilization in a Hadoop cluster and reduces the makespan of MapReduce jobs. Our approach can be used for automatically instructing redistribution of data and as such is complementary to current scheduling solutions in Hadoop (i.e., those favoring data locality).

Our initial experiments showed the effectiveness of the approach and the benefits it brings to the performances of MapReduce jobs in a simulated Hadoop cluster execution. Our future plans focus on providing new scheduling techniques in Hadoop that take full advantage of a priori knowing more favorable data distribution, and hence use idle network cycles to transfer data to the

tasks in advance, without additionally deferring their executions.

8 Acknowledgements

This work has been partially supported by the Secreteria d'Universitats i Recerca de la Generalitat de Catalunya under 2014 SGR 1534, and by the Spanish Ministry of Education grant FPU12/04915.

Chapter 7

Conclusions and Future Directions

Abstract

In this chapter, we summarize the main results of this PhD thesis, presented in Chapters 2 - 6. In addition, we propose several promising future directions stemming from this thesis work.

1 Conclusions

In this PhD thesis, we presented our approach for managing different phases of the data-intensive flow lifecycle. The main goal of this thesis is to provide means for automating and facilitating burdensome tasks of the design and optimization of data-intensive flows. The thesis included a theoretical study, by means of a literature survey, aiming at indicating the main challenges in the filed of DIFs in today's BI applications. Considering the remaining challenges identified in the literature, this thesis further presented several approaches for facilitating the DIF lifecycle. In particular, we focus on design, optimization, implementation, execution, and maintenance of DIFs.

In what follows, we first summarize the contributions presented in chapters 2 - 6, and finally conclude the overall thesis.

Chapter 2 presented the survey of the current literature in the filed of DIFs. The goal of this survey was on studying the related work in the field, and consequently on identifying the remaining challenges and gaps in the design and optimization of DIFs. The first result provided a clear picture of the theoretical underpinnings of DIFs in today's BI applications. Moreover, the study further identified the main challenges that distinguish traditional, batched DIFs (e.g., ETL processes) that typically integrate in-house data for loading DW for further strategic analysis, and today's DIFs that require means for efficiently dis-

covering and integrating a plethora of both internal and external sources, in near real-time. We accordingly, using these challenges as dimensions, classified the current approaches based on the level they achieve in each of these dimensions. Finally, as the main outcome of this study, we proposed a high-level architecture for managing the burdensome lifecycle of DIFs. The architecture captures in a holistic way the complete lifecycle of DIFs, and as such, it can be seen as a roadmap for both academia and industry toward building DIFs in next generation BI systems. In particular, in this PhD thesis, the results of this survey served as a cornerstone for further studying in detail some of the remaining challenges in the filed of DIFs.

Chapter 3 presented our approach for dealing with the design, maintenance, and multi-flow optimization of DIFs. In particular, we presented the novel algorithm, called CoAl, which facilitates the incremental consolidation of DIFs, starting from data flows satisfying individual information requirements. CoAl iteratively searches for different possibilities of integrating new data flows into the existing multi-flow, focusing on maximizing data and operation reuse, and finally, proposes a unified solution satisfying all information requirements. In Chapter 3, we formalized the notion of DIFs, including the formalization of DIF's operation semantics. Furthermore, in terms of this notation we introduced novel techniques for operation comparison and operation reordering, which are used by the CoAl algorithm when searching for maximizing the DIF reuse. Finally, using the implemented prototype of the CoAl algorithm, we presented the evaluation results of our approach. The results presented in Chapter 3 demonstrated the efficiency and scalability of our approach, as well as the quality of produced solutions. In particular, we report the improvements of the overall execution time and other benefits of integrated multi-flows.

Chapter 4 presented our semi-automatic method, called ORE, for iteratively creating DW schema designs, based on information requirements. ORE starts from MD interpretations of single requirements, and incrementally builds a unified solution satisfying the complete set of information requirements. Chapter 4 formalized the notion of information requirements and their MD interpretations, as well as the MD schema, and further presented two main algorithms that support the integration of MD schemata, i.e., fact matching (FM) and dimension matching (DM). Apart from the main goal of providing semi-automatic support for the design and maintenance of MD schema, an important objective of this approach was to provide systematic collection of metadata during the entire evolution cycle, in order to "fuel" the process of design and adaptation of back-end DIFs (i.e., ETL processes) to the occurred evolution events. Given the generally high complexity of the MD schema integration process, caused by the plethora of different integration possibilities, ORE uses a customizable cost model for evaluating the resulted solutions, in order to prune the search space and offer a user only solutions that satisfy the previously set objectives. Using such heuristics, in Chapter 4, we lastly evaluated the performance and

different characteristics of the ORE method, demonstrating its efficiency in dealing with increasing problem size (i.e., number of input information requirements). In addition, we also reported the results of evaluating the benefits of our semi-automatic approach, by measuring the emotional aspects and the considerable amount of human efforts required for manually handling the design and evolution of a DW schema.

Chapter 5 presented our approach, called BabbleFlow, for supporting the implementation and deployment of DIFs, potentially spanning diverse execution and storage engines. In particular, the main objective of this work was to provide the engine independence for DIFs, enabling processing of DIF at the logical level, as well as their further deployment on a variety of execution engines. Chapter 5 thus introduced our logical encoding of DIFs, called xLM. We further presented the techniques for parsing DIFs from different execution engines into xLM, and also back to the same or different execution engine(s). In Chapter 5, we also demonstrated an important functionality of our approach for building hybrid DIFs, by means of decomposing a logical DIF, and generating code for specific subflows executing on different engines. Finally, we extensively evaluated our approach using data flows from TPC-H and TPC-DS benchmarks, as well as custom made data flows combining ETL and analytical functionalities. The results showed both the efficiency of our approach for translating DIF into deployable formats, as well as different benefits that it can bring to the optimization of hybrid DIFs.

Chapter 6 presented our approach, called H-WorD, for supporting the scheduling of execution of DIFs with workload-driven data redistribution. The main goal of this work was to build a system to analyze the workload that DIFs impose over execution resources, and decide on a better redistribution of input data that improves their overall makespan. In particular, we presented a model, exemplified with the MapReduce framework, which based on the placement of data inside the cluster, estimates DIFs' execution times and defines different execution modes in which data processing tasks can execute. Furthermore, the H-WorD algorithm uses such model for deciding more beneficial execution modes of data processing tasks, and hence the distribution of their input data in the cluster. We demonstrated two use cases of our approach, namely, offline, complementary to existing scheduling algorithm, where the redistribution of data is performed beforehand, and on the fly, which requires adaptation of current scheduling algorithms, to leveraging on idle network cycles, schedule needed data transfers on the fly, in advance, without deferring other task's executions. Finally, we evaluated our approach by means of simulating the MapReduce jobs executions, and using typical MapReduce benchmark examples from the Apache Hadoop distribution (i.e., WordCount and TeraSort). The results showed the efficiency of H-WorD in dealing with small and medium-sized clusters, while also indicated a need for introducing heuristics for pruning the search space when looking for the more beneficial data

distribution. More importantly, the evaluation results largely corroborated the benefits that improving the distribution of data in the cluster brings to the execution makespan of DIFs.

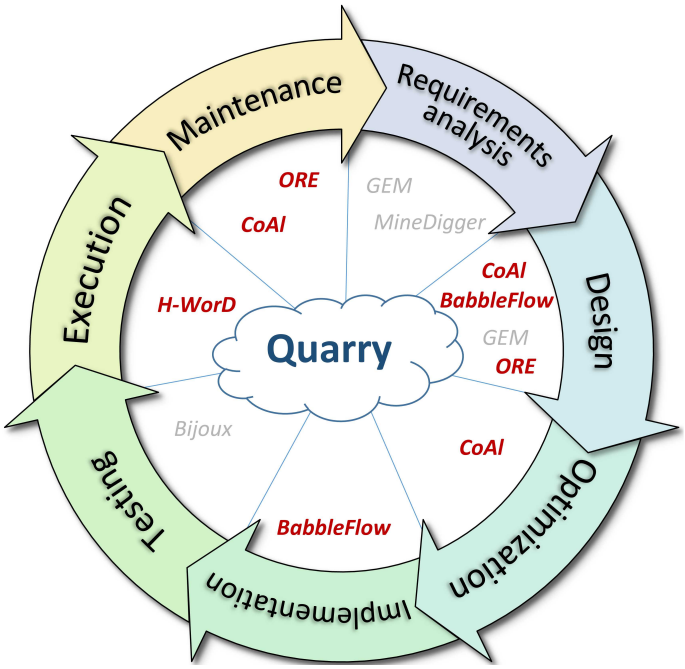


Fig. 7.1: The results of this PhD thesis inside the lifecycle of DIFs (Note: The modules in red are the main results of this PhD thesis whereas the gray ones resulted from collaborative work.)

Overall, the burdensome tasks of designing, deploying, and optimizing the execution of data-intensive flows in BI systems require considerable amount of human effort, which in today's hectic business environments is not always affordable, both in terms of money and time. Moreover, the maintenance of DIFs in front of changed information requirements demands a continuous support of a designer during the entire lifecycle. To this end, this PhD thesis proposed automated means for efficiently supporting tasks in different phases of the DIF lifecycle, namely design, optimization, implementation, execution, and maintenance (see Figure 7.1). In particular, we started from CoAl (Chapter 3) which facilitates the incremental design of DIFs and their maintenance in front of evolving analytical needs of end users. Moreover, CoAl also provides support for multi-flow optimization of DIF by means of maximizing the data and operation reuse among DIFs. Next, we proposed ORE (Chapter 4), which automates the incremental design of MD schema and at the same time systematically traces metadata of such process in order to further support the

design of back-end, batched DIFs in DW systems, and their adaptation to the evolution of the target schema. Furthermore, for implementing DIFs, we proposed BabbleFlow (Chapter 5), which provides support for deploying DIFs over single or multiple execution engines. Moreover, it also facilitates the design of DIFs, specially in the case when they span multiple execution or storage engines (i.e., hybrid data flows). Finally, we proposed H-WorD (Chapter 6), to facilitate the execution of DIFs, by means of supporting their scheduling with workload-driven data redistribution.

The main results of this PhD thesis were integrated as modules inside Quarry, our platform for managing the lifecycle of analytical infrastructures (see Appendix A). As we showed above, these modules automated to large extent the burdensome tasks during the DIF lifecycle, previously typically left to the manual efforts of BI system designer.

Moreover, during the last five years, the Quarry project has included collaboration with other researchers, bachelor, master , and PhD students, which overall resulted in modules that encompass the complete lifecycle of DIFs (see Figure 7.1). The GEM system [146] was in fact a springboard for the research path of this PhD thesis, introducing automatable means for analyzing information requirements of business users and designing MD schemata and ETL processes that satisfy them. GEM's developlemt and integration was the topic of my master thesis [86]. Next, MineDigger module provided graphical interface to the Quarry platform, largely facilitating the process of requirements elicitation and analysis for end users, and providing high usability of the complete Quarry platform. Its development and integration with Quarry platform was the topic of the bachelor degree project, done by Héctor Candón [33]. Lastly, a critical part of the DIF lifecycle, i.e., testing, was largely facilitated by Bijoux, the data generation tool for evaluating different quality objectives of DIFs [178]. The Bijoux module resulted from the collaborative work with a master student Emona Nakuci, working on her master thesis [123], and a fellow colleague PhD student, Vasileios Theodorou.

2 Future Directions

In this PhD thesis, we provided the means to fully or partially facilitate the phases of the DIF lifecycle. Nevertheless, there are several possible directions for future work inside the Quarry project.

Regarding the design of DIFs, driven by information requirements, although high automation is achieved, it required lowering the level of abstraction for providing the requirement, tightening the requirements to a multidimensional model. Further research is needed to bridge the gap between the typical ways end users define their requirements and how these requirements are further automatically translated into deployable DIFs.

In the context of DW systems, DIFs are highly correlated to the design and constraints imposed over the target MD schema. Thus, the design and maintenance of ETL processes must be instructed by the design decisions made at the MD schema side. We made a step toward this in our ORE approach. However, further study is required to handle and exploit such interdependence between the MD schema and ETL process design (i.e., ORE and CoAl), and explore the possibilities how each of these design processes can benefit from the relevant information inferred by the other process. For instance, the aggregation and normalization levels of the produced schema could be reconsidered, since this would affect the way the back-end ETL process is tailored (i.e., trade-offs between materialized views and OLAP querying). Similarly, checkpointing or bottlenecks detected at the ETL level may cause changes at the MD schema for the sake of performance.

The implementation and deployment of DIFs is largely supported by the automatable means for translating DIFs to executable formats, provided by BabbleFlow. On the one hand, the extension of such system for more high-level functional, procedural, or object-oriented languages (e.g., Java, Scala, Python) would require further work, and it would be immensely beneficial for applying the approach for modern data processing platforms, like Apache Spark, Hadoop, etc. On the other hand, further work in this topic should be directed toward enabling smarter deployment of DIFs, especially for deciding the target execution engine(s) of DIFs in the hybrid execution environments.

Lastly, further work is also required for building more sophisticated scheduling techniques for executing DIFs in distributed data processing systems. Our results in Chapter 6 showed that we can benefit from smarter data redistribution when scheduling DIFs. Thus the plan here is to study how such information can be further included inside the scheduling policies of data processing systems (e.g., Hadoop and Spark). Another interesting research direction is toward providing a more dynamic scheduling system. Instead of starting from a static set of DIFs, the scheduler should only use a high level notion of cluster workload (e.g., DIF types, arrival rates, service rates), and by means of prediction models, anticipate future workload over the cluster resources, and decide on the needed, in advance, data transfers, hence avoiding deferment of data processing tasks.

References

[1] Apache HBase. https://hbase.apache.org/. [02-March-2016].

[2] Cluster Rebalancing in HDFS. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Cluster+Rebalancing. [02-March-2016].

[3] Hadoop: Capacity Scheduler. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html. [04-March-2016].

[4] Hadoop: Fair Scheduler. https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/FairScheduler.htmll. [04-March-2016].

[5] TPC-H, last accessed March, 2015. http://www.tpc.org/tpch/.

[6] Apache Jena, last accessed September, 2013. http://jena.apache.org/.

[7] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J.-N. Mazón, F. Naumann, T. B. Pedersen, S. Rizzi, J. Trujillo, P. Vassiliadis, and G. Vossen. Fusion Cubes: Towards Self-Service Business Intelligence. IJDWM, 9(2):66–88, 2013.

[8] A. Abelló, E. Rodríguez, T. Urpí, X. B. Illa, M. J. Casany, C. Martín, and C. Quer. LEARN-SQL: Automatic Assessment of SQL Based on IMS QTI Specification. In ICALT, pages 592–593, 2008.

[9] A. Abelló, O. Romero, T. B. Pedersen, R. B. Llavori, V. Nebot, M. J. A. Cabo, and A. Simitsis. Using semantic web technologies for exploratory OLAP: A survey. IEEE Trans. Knowl. Data Eng., 27(2):571–588, 2015.

[10] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. HadoopDB: An architectural hybrid of mapreduce and DBMS technologies for analytical workloads. PVLDB, 2(1):922–933, 2009.

[11] A. Ailamaki. Running with scissors: Fast queries on just-in-time databases. In IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014, page 1, 2014.

[12] Z. E. Akkaoui, J.-N. Mazón, A. A. Vaisman, and E. Zimányi. BPMN-Based Conceptual Modeling of ETL Processes. In DaWaK, pages 1–14, 2012.

[13] Z. E. Akkaoui, E. Zimányi, J.-N. Mazón, and J. Trujillo. A BPMN-Based Design and Maintenance Framework for ETL Processes. IJDWM, 9(3), 2013.

[14] A. Albrecht and F. Naumann. METL: Managing and Integrating ETL Processes. In VLDB PhD Workshop, 2009.

[15] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. IEEE Trans. Software Eng., 28(10):970–983, 2002.

[16] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Foundations of schema mapping management. In PODS, pages 227–238, 2010.

[17] B. Bebel, J. Eder, C. Koncilia, T. Morzy, and R. Wrembel. Creation and management of versions in multiversion data warehouse. In SAC, pages 717–723, 2004.

[18] L. Bellatreche, S. Khouri, and N. Berkani. Semantic Data Warehouse Design: From ETL to Deployment à la Carte. In DASFAA (2), 2013.

[19] H. Berthold, P. Rösch, S. Zöller, F. Wortmann, A. Carenini, S. Campbell, P. Bisson, and F. Strohmaier. An architecture for ad-hoc and collaborative business intelligence. In EDBT/ICDT Workshops, 2010.

[20] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. Int. J. Semantic Web Inf. Syst., 5(3):1–22, 2009.

[21] M. Blaschka, C. Sapia, and G. Höfling. On schema evolution in multidimensional databases. In DaWaK, pages 153–164, 1999.

[22] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. Handbook on scheduling: from theory to applications. Springer Science & Business Media, 2007.

[23] M. Body, M. Miquel, Y. Bédard, and A. Tchounikine. A multidimensional and multiversion structure for OLAP applications. In DOLAP, pages 1–6, 2002.

[24] M. Böhm, D. Habich, and W. Lehner. Multi-flow optimization via horizontal message queue partitioning. In ICEIS, pages 31–47, 2010.

[25] M. Böhm, D. Habich, W. Lehner, and U. Wloka. DIPBench Toolsuite: A Framework for Benchmarking Integration Systems. In ICDE, pages 1596–1599, 2008.

[26] M. Böhm, U. Wloka, D. Habich, and W. Lehner. Workload-based optimization of integration processes. In CIKM, pages 1479–1480, 2008.

[27] M. Böhm, U. Wloka, D. Habich, and W. Lehner. GCIP: exploiting the generation and optimization of integration processes. In EDBT, pages 1128–1131, 2009.

[28] R. M. Bruckner, B. List, and J. Schiefer. Striving towards near real-time data integration for data warehouses. In DaWaK, pages 317–326, 2002.

[29] N. Bruno, S. Jain, and J. Zhou. Continuous cloud-scale query optimization and processing. PVLDB, 6(11):961–972, 2013.

[30] P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding Structure to Unstructured Data. In ICDT, pages 336–350, 1997.

[31] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. Inf. Syst., 29(2):147–163, 2004.

[32] A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In IJCAI, pages 16–21, 2003.

[33] H. Candón. The minecart project: A wee step towards bi 2.0. 2014. Bachelor Degree Project.

[34] D. Caruso. Bringing Agility to Business Intelligence, February 2011. Information Management, `http://www.information-management.com/infodirect/2009_191/business_intelligence_metadata_analytics_ETL_data_management-10019747-1.html`.

[35] C. L. P. Chen and C. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. Inf. Sci., 275:314–347, 2014.

[36] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. Proceedings of the VLDB Endowment, 5(12):1802–1813, 2012.

[37] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. ACM Comput. Surv., 44(3):15, 2012.

[38] T. H. Davenport. How to design smart business experiments. Harvard business review, 87(2):68–76, 2009.

[39] U. Dayal. Processing queries over generalization hierarchies in a multidatabase system. In VLDB, pages 342–353, 1983.

[40] U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson. Data integration flows for business intelligence. In EDBT, pages 1–11, 2009.

[41] U. Dayal, H. A. Kuno, J. L. Wiener, K. Wilkinson, A. Ganapathi, and S. Krompass. Managing operational business intelligence workloads. Operating Systems Review, 43(1):92–98, 2009.

[42] U. Dayal, K. Wilkinson, A. Simitsis, M. Castellanos, and L. Paz. Optimization of Analytic Data Flows for Next Generation Business Intelligence Applications. In TPCTC, pages 46–66, 2011.

[43] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113, 2008.

[44] S. Dessloch, M. A. Hernández, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating Schema Mapping and ETL. In ICDE, pages 1307–1316, 2008.

[45] D. J. DeWitt, A. Halverson, R. V. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling. Split query processing in polybase. In SIGMOD Conference, pages 1255–1266, 2013.

[46] A. Doan, A. Y. Halevy, and Z. G. Ives. Principles of Data Integration. Morgan Kaufmann, 2012.

[47] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. VLDB J., 18(2):469–500, 2009.

[48] W. Du, R. Krishnamurthy, and M. Shan. Query optimization in a heterogeneous DBMS. In VLDB, pages 277–291, 1992.

[49] R. J. Duffin. Topology of series-parallel networks. Journal of Mathematical Analysis and Applications, 10(2):303–318, 1965.

[50] W. W. Eckerson. Best practices in operational BI. Business Intelligence Journal, 12(3):7–9, 2007.

[51] European Commission. G. technology readiness levels (TRL), 2014.

[52] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In Conceptual Modeling: Foundations and Applications, 2009.

[53] R. Fagin, B. Kimelfeld, and P. G. Kolaitis. Probabilistic data exchange. Journal of the ACM (JACM), 58(4):15, 2011.

[54] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In ICDT, pages 207–224. Springer, 2003.

[55] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. Theor. Comput. Sci., 336(1):89–124, 2005.

[56] R. Feldman and J. Sanger. The text mining handbook: advanced approaches in analyzing unstructured data. Cambridge University Press, 2007.

[57] E. Ferrara, P. D. Meo, G. Fiumara, and R. Baumgartner. Web data extraction, applications and techniques: A survey. Knowl.-Based Syst., 70:301–323, 2014.

[58] E. Friedman, P. M. Pawlowski, and J. Cieslewicz. Sql/mapreduce: A practical approach to self-describing, polymorphic, and parallelizable user-defined functions. PVLDB, 2(2):1402–1413, 2009.

[59] M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans for Data Integration. In Intelligent Information Integration, 1999.

[60] S. García, O. Romero, and R. Raventós. DSS from an RE perspective: A systematic mapping. Journal of Systems and Software, 117:488 – 507, 2016.

[61] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. J. Intell. Inf. Syst., 8(2):117–132, 1997.

[62] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. BigBench: towards an industry standard benchmark for big data analytics. In SIGMOD Conference, pages 1197–1208, 2013.

[63] G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In PODS, pages 133–142, 2007.

[64] G. Giannikis, D. Makreshanski, G. Alonso, and D. Kossmann. Shared workload optimization. PVLDB, 7(6):429–440, 2014.

[65] P. Giorgini, S. Rizzi, and M. Garzetti. Grand: A goal-oriented approach to requirement analysis in data warehouses. DSS, 45(1):4–21, 2008.

[66] M. Golfarelli, J. Lechtenbörger, S. Rizzi, and G. Vossen. Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. Data Knowl. Eng., 59(2):435–459, 2006.

[67] M. Golfarelli, D. Maio, and S. Rizzi. The dimensional fact model: A conceptual model for data warehouses. Int. J. Cooperative Inf. Syst., 7(2-3):215–247, 1998.

[68] M. Golfarelli, F. Mandreoli, W. Penzo, S. Rizzi, and E. Turricchia. OLAP query reformulation in peer-to-peer data warehousing. Inf. Syst., 37(5):393–411, 2012.

[69] M. Golfarelli and S. Rizzi. Data Warehouse Design: Modern Principles and Methodologies. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2009.

[70] M. Golfarelli, S. Rizzi, and I. Cella. Beyond data warehousing: what's next in business intelligence? In DOLAP, pages 1–6, 2004.

[71] M. Golfarelli, S. Rizzi, and E. Turricchia. Modern software engineering methodologies meet data warehouse design: 4wd. In DaWaK, volume 6862 of LNCS, pages 66–79. Springer, 2011.

[72] Z. Guo, G. Fox, and M. Zhou. Investigation of data locality in mapreduce. In CCGrid, pages 419–426, 2012.

[73] L. M. Haas. Beauty and the Beast: The Theory and Practice of Information Integration. In ICDT, pages 28–43, 2007.

[74] R. Halasipuram, P. M. Deshpande, and S. Padmanabhan. Determining essential statistics for cost based optimization of an ETL workflow. In EDBT, pages 307–318, 2014.

[75] A. Y. Halevy. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.

[76] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. ACM Trans. Inf. Syst., 3(3):253–278, 1985.

[77] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In CIDR, pages 261–272, 2011.

[78] F. Hueske, M. Peters, M. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas. Opening the Black Boxes in Data Flow Optimization. PVLDB, 5(11):1256–1267, 2012.

[79] R. Hughes. Agile Data Warehousing: Delivering world-class business intelligence systems using Scrum and XP. IUniverse, 2008.

[80] B. Hüsemann, J. Lechtenbörger, and G. Vossen. Conceptual data warehouse modeling. In DMDW, page 6, 2000.

[81] IBM, P. Zikopoulos, and C. Eaton. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. McGraw-Hill Osborne Media, 1st edition, 2011.

[82] W. H. Inmon. Building the Data Warehouse. John Wiley & Sons, Inc., 1992.

[83] W. H. Inmon, D. Strauss, and G. Neushloss. DW 2.0: The architecture for the next generation of data warehousing: The architecture for the next generation of data warehousing. Morgan Kaufmann, 2010.

[84] M. Jarke and J. Koch. Query Optimization in Database Systems. ACM Comput. Surv., 16(2):111–152, 1984.

[85] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong. BAR: an efficient data locality driven task scheduling algorithm for cloud computing. In CCGrid, pages 295–304, 2011.

[86] P. Jovanovic. Integration of multidimensional and ETL design. 2011. Master Thesis.

[87] P. Jovanovic, O. Romero, and A. Abelló. A unified view of data-intensive flows in business intellignece systems: A sruvey. Trans. Large-Scale Data- and Knowledge-Centered Systems, 2013. InPress.

[88] P. Jovanovic, O. Romero, A. Simitsis, and A. Abelló. Integrating ETL processes from information requirements. In DaWaK, pages 65–80, 2012.

[89] P. Jovanovic, O. Romero, A. Simitsis, and A. Abelló. Incremental consolidation of data-intensive multi-flows. IEEE Trans. Knowl. Data Eng., 28(5):1203–1216, 2016.

[90] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, H. Candón, and S. Nadal. Quarry: Digging up the gems of your data treasury. In EDBT, pages 549–552, 2015.

[91] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, and D. Mayorova. A requirement-driven approach to the design and evolution of data warehouses. Inf. Syst., 44:94–119, 2014.

[92] P. Jovanovic, A. Simitsis, and K. Wilkinson. Babbleflow: a translator for analytic data flow programs. In SIGMOD Conference, pages 713–716, 2014.

[93] P. Jovanovic, A. Simitsis, and K. Wilkinson. Engine independence for logical analytic flows. In ICDE, pages 1060–1071, 2014.

[94] P. Kalnis and D. Papadias. Multi-query optimization for on-line analytical processing. Inf. Syst., 28(5):457–473, 2003.

[95] A. Karagiannis, P. Vassiliadis, and A. Simitsis. Scheduling strategies for efficient ETL execution. Inf. Syst., 38(6):927–945, 2013.

[96] R. Kimball and J. Caserta. The Data Warehouse ETL Toolkit. John Wiley & Sons, 2004.

[97] R. Kimball, L. Reeves, W. Thornthwaite, and M. Ross. The Data Warehouse Lifecycle Toolkit. J. Wiley & Sons, 1998.

[98] T. Kirk, A. Y. Levy, Y. Sagiv, D. Srivastava, and Others. The information manifold. In Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments, volume 7, pages 85–91, 1995.

[99] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In PODS, pages 61–75, 2005.

[100] R. Kolisch and S. Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. Springer, 1999.

[101] G. Kougka and A. Gounaris. Cost optimization of data flows based on task re-ordering. CoRR, abs/1507.08492, 2015.

[102] G. Kougka, A. Gounaris, and K. Tsichlas. Practical algorithms for execution engine selection in data flows. Future Generation Computer Systems, 45:133–148, 2015.

[103] T. Kraft, H. Schwarz, R. Rantzau, and B. Mitschang. Coarse-grained optimization: Techniques for rewriting SQL statement sequences. In VLDB, pages 488–499, 2003.

[104] W. Labio and H. Garcia-Molina. Efficient Snapshot Differential Algorithms for Data Warehousing. In VLDB, pages 63–74, 1996.

[105] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A Brief Survey of Web Data Extraction Tools. SIGMOD Record, 31(2):84–93, 2002.

[106] M. Lenzerini. Data Integration: A Theoretical Perspective. In PODS, pages 233–246. ACM, 2002.

[107] J. Leskovec, A. Rajaraman, and J. D. Ullman. Mining of massive datasets. Cambridge University Press, 2014.

[108] L. Libkin. Data exchange and incomplete information. In PODS, pages 60–69, 2006.

[109] B. G. Lindsay, L. M. Haas, C. Mohan, H. Pirahesh, and P. F. Wilms. A Snapshot Differential Refresh Algorithm. In SIGMOD Conference, pages 53–60, 1986.

[110] A. Löser, F. Hueske, and V. Markl. Situational Business Intelligence. In BIRTE, volume 27, pages 1–11, 2008.

[111] A. Maté and J. Trujillo. A trace metamodel proposal based on the model driven architecture framework for the traceability of user requirements in data warehouses. Inf. Syst., 37(8):753–766, 2012.

[112] A. Maté, J. Trujillo, E. de Gregorio, and I.-Y. Song. Improving the maintainability of data warehouse designs: modeling relationships between sources and user concepts. In DOLAP, pages 25–32, 2012.

[113] J. Mazón, J. Trujillo, and J. Lechtenbörger. Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms. Data Knowl. Eng., 63(3):725–751, 2007.

[114] J.-N. Mazón, J. Lechtenbörger, and J. Trujillo. A survey on summarizability issues in multidimensional modeling. Data Knowl. Eng., 68(12):1452–1469, 2009.

[115] J.-N. Mazón and J. Trujillo. An MDA approach for the development of data warehouses. Decision Support Systems, 45(1):41–58, 2008.

[116] P. McBrien and A. Poulovassilis. Data Integration by Bi-Directional Schema Transformation Rules. In ICDE, pages 227–238, 2003.

[117] K. McDonald, A. Wilmsmeier, D. C. Dixon, and W. Inmon. Mastering the SAP Business Information Warehouse. John Wiley & Sons, 2002.

[118] T. Morzy and R. Wrembel. On querying versions of multiversion data warehouse. In DOLAP, pages 92–101, 2004.

[119] L. T. Moss and S. Atre. Business intelligence roadmap: the complete project lifecycle for decision-support applications. Addison-Wesley Professional, 2003.

[120] L. Muñoz, J.-N. Mazón, and J. Trujillo. Automatic generation of ETL processes from conceptual models. In DOLAP, pages 33–40, 2009.

[121] A. Nabli, J. Feki, and F. Gargouri. Automatic construction of multidimensional schema from OLAP requirements. In AICCSA, page 28, 2005.

[122] P. Naggar, L. Pontieri, M. Pupo, G. Terracina, and E. Virardi. A model and a toolkit for supporting incremental data warehouse construction. In DEXA, pages 123–132, 2002.

[123] E. Nakuci. Data generation for the simulation of artifact-centric processes. 2014. Master Thesis.

[124] T. Neumann. Query optimization (in relational databases). In Encyclopedia of Database Systems, pages 2273–2278. Springer US, 2009.

[125] K. W. Ong, Y. Papakonstantinou, and R. Vernoux. The SQL++ semi-structured data model and query language: A capabilities survey of sql-on-hadoop, nosql and newsql databases. CoRR, abs/1405.3631, 2014.

[126] P. O'Neil and E. O'Neil and X. Chen. The Star Schema Benchmark, http://www.cs.umb.edu/~poneil/StarSchemaB.PDF (last access 21/09/2013).

[127] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: locality-aware resource allocation for mapreduce in a cloud. In SC, pages 58:1–58:11, 2011.

[128] A. S. Pall and J. S. Khaira. A comparative review of extraction, transformation and loading tools. Database Systems Journal, 4(2):42–51, 2013.

[129] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou. Policy-regulated management of ETL evolution. J. Data Semantics, 13:147–177, 2009.

[130] P. F. Patel-Schneider and I. Horrocks. Position paper: a comparison of two modelling paradigms in the Semantic Web. In WWW, pages 3–12, 2006.

[131] C. Phipps and K. C. Davis. Automating data warehouse conceptual schema design and evaluation. In DMDW, volume 58 of CEUR Workshop Proceedings, pages 23–32, 2002.

[132] K. Pohl. Requirements Engineering - Fundamentals, Principles, and Techniques. Springer, 2010.

[133] J. Polo, Y. Becerra, D. Carrera, J. Torres, E. Ayguadé, and M. Steinder. Adaptive MapReduce scheduling in shared environments. In IEEE/ACM CCGrid, pages 61–70, 2014.

[134] W. Qu and S. Dessloch. A real-time materialized view approach for analytic flows in hybrid cloud environments. Datenbank-Spektrum, 14(2):97–106, 2014.

[135] C. Quix. Repository support for data warehouse evolution. In DMDW, page 4, 1999.

[136] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.

[137] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. IEEE Data Eng. Bull., 23(4):3–13, 2000.

[138] A. Rheinländer, A. Heise, F. Hueske, U. Leser, and F. Naumann. Sofa: An extensible logical optimizer for udf-heavy data flows. Information Systems, 52(0):96 – 125, 2015.

[139] S. Rizzi. Business intelligence. In Encyclopedia of Database Systems, pages 287–288. 2009.

[140] O. Romero and A. Abelló. A Survey of Multidimensional Modeling Methodologies. IJDWM, 5(2):1–23, 2009.

[141] O. Romero and A. Abelló. Automatic Validation of Requirements to Support Multidimensional Design. Data Knowl. Eng., 69(9):917–942, 2010.

[142] O. Romero and A. Abelló. A framework for multidimensional design of data warehouses from ontologies. Data Knowl. Eng., 69(11):1138–1157, 2010.

[143] O. Romero and A. Abelló. Open Access Semantic Aware Business Intelligence. In E. Zimányi, editor, Business Intelligence, volume 172 of Lecture Notes in Business Information Processing, pages 121–149. Springer International Publishing, 2014.

[144] O. Romero, D. Calvanese, A. Abelló, and M. Rodriguez-Muro. Discovering functional dependencies for multidimensional design. In DOLAP, pages 1–8, 2009.

[145] O. Romero, P. Marcel, A. Abelló, V. Peralta, and L. Bellatreche. Describing analytical sessions using a multidimensional algebra. In DaWaK, volume 6862 of Lecture Notes in Computer Science, pages 224–239. Springer, 2011.

[146] O. Romero, A. Simitsis, and A. Abelló. GEM: Requirement-Driven Generation of ETL and Multidimensional Conceptual Designs. In DaWaK, volume 6862 of Lecture Notes in Computer Science, pages 80–95. Springer, 2011.

[147] R. Romero, J.-N. Mazón, J. Trujillo, M. A. Serrano, and M. Piattini. Quality of data warehouses. In Encyclopedia of Database Systems, pages 2230–2235. 2009.

[148] A. Rosenthal and L. J. Seligman. Data integration in the large: The challenge of reuse. In VLDB, pages 669–675, 1994.

[149] M. T. Roth, M. Arya, L. M. Haas, M. J. Carey, W. F. Cody, R. Fagin, P. M. Schwarz, J. T. II, and E. L. Wimmers. The garlic project. In SIGMOD Conference, page 557, 1996.

[150] P. Roy and S. Sudarshan. Multi-query optimization. In Encyclopedia of Database Systems, pages 1849–1852. 2009.

[151] C. P. Sayers, A. Simitsis, G. Koutrika, A. G. Gonzalez, D. T. Cantu, and M. Hsu. The farm: where pig scripts are bred and raised. In SIGMOD Conference, pages 1025–1028, 2013.

[152] R. C. Schank and L. G. Tesler. A conceptual parser for natural language. In Proceedings of the 1st International Joint Conference on Artificial Intelligence, Washington, DC, May 1969, pages 569–578, 1969.

[153] F. Serban, J. Vanschoren, J. Kietz, and A. Bernstein. A survey of intelligent assistants for data analysis. ACM Comput. Surv., 45(3):31, 2013.

[154] M. A. Serrano, C. Calero, H. A. Sahraoui, and M. Piattini. Empirical studies to assess the understandability of data warehouse schemas using structural metrics. Software Quality Journal, 16(1):79–106, 2008.

[155] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. ACM Trans. Database Syst., 2(2):134–174, 1977.

[156] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In MSST, pages 1–10, 2010.

[157] A. Simitsis, P. Vassiliadis, U. Dayal, A. Karagiannis, and V. Tziovara. Benchmarking ETL workflows. In TPCTC, pages 199–220, 2009.

[158] A. Simitsis, P. Vassiliadis, and T. K. Sellis. State-Space Optimization of ETL Workflows. IEEE Trans. Knowl. Data Eng., 17(10):1404–1419, 2005.

[159] A. Simitsis and K. Wilkinson. Revisiting ETL benchmarking: The case for hybrid flows. In TPCTC, pages 75–91, 2012.

[160] A. Simitsis and K. Wilkinson. The specification for xLM: an encoding for analytic flows, 2014.

[161] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. QoX-driven ETL design: reducing the cost of ETL consulting engagements. In SIGMOD Conference, 2009.

[162] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing analytic data flows for multiple execution engines. In SIGMOD Conference, pages 829–840, 2012.

[163] A. Simitsis, K. Wilkinson, and U. Dayal. Hybrid Analytic Flows - the Case for Optimization. Fundam. Inform., 128(3):303–335, 2013.

[164] A. Simitsis, K. Wilkinson, U. Dayal, and M. Hsu. HFMS: managing the lifecycle and complexity of hybrid analytic data flows. In ICDE, pages 1174–1185, 2013.

[165] A. Simitsis, K. Wilkinson, and P. Jovanovic. xPAD: a platform for analytic data flows. In SIGMOD Conference, pages 1109–1112, 2013.

[166] D. Skoutas and A. Simitsis. Designing ETL processes using semantic web technologies. In DOLAP, pages 67–74, 2006.

[167] D. Skoutas and A. Simitsis. Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. Int. J. Semantic Web Inf. Syst., 3(4):1–24, 2007.

[168] I.-Y. Song, R. Khare, and B. Dai. SAMSTAR: a semi-automated lexical method for generating star schemas from an entity-relationship diagram. In DOLAP, pages 9–16, 2007.

[169] K. H. Strange. Data Warehouse TCO: Don't Underestimate the Cost of ETL. Gartner Research, CS-15-2007, 2002.

[170] K. H. Strange. ETL Was the Key to This Data Warehouse's Success. Gartner Research, CS-15-3143, 2002.

[171] V. Tannen. Relational algebra. In Encyclopedia of Database Systems, pages 2369–2370. 2009.

[172] D. Theodoratos and M. Bouzeghoub. A general framework for the view selection problem for data warehouse design and evolution. In DOLAP, pages 1–8, 2000.

[173] D. Theodoratos, T. Dalamagas, A. Simitsis, and M. Stavropoulos. A randomized approach for the incremental design of an evolving data warehouse. In ER, pages 325–338, 2001.

[174] D. Theodoratos and T. K. Sellis. Designing Data Warehouses. Data Knowl. Eng., 31(3):279–301, 1999.

[175] D. Theodoratos and T. K. Sellis. Incremental design of a data warehouse. J. Intell. Inf. Syst., 15(1):7–27, 2000.

[176] V. Theodorou, A. Abelló, W. Lehner, and M. Thiele. Quality measures for etl processes: from goals to implementation. Concurrency and Computation: Practice and Experience, pages n/a–n/a, 2015. cpe.3729.

[177] V. Theodorou, A. Abelló, M. Thiele, and W. Lehner. POIESIS: a tool for quality-aware ETL process redesign. In EDBT, pages 545–548, 2015.

[178] V. Theodorou, P. Jovanovic, A. Abelló, and E. Nakuçi. Data generator for evaluating ETL process quality. Information Systems, pages –, 2016.

[179] R. Torlone. Two approaches to the integration of heterogeneous data warehouses. Distributed and Parallel Databases, 23(1):69–97, 2008.

[180] J. Trujillo and S. Luján-Mora. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In ER, pages 307–320, 2003.

[181] V. Tziovara, P. Vassiliadis, and A. Simitsis. Deciding the physical implementation of ETL workflows. In DOLAP, pages 49–56, 2007.

[182] J. D. Ullman. Information Integration Using Logical Views. In ICDT, pages 19–40, 1997.

[183] A. A. Vaisman, A. O. Mendelzon, W. Ruaro, and S. G. Cymerman. Supporting dimension updates in an OLAP server. Inf. Syst., 29(2):165–185, 2004.

[184] J. Varga, O. Romero, T. B. Pedersen, and C. Thomsen. Towards next generation BI systems: The analytical metadata challenge. In DaWaK, pages 89–101, 2014.

[185] P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In DMDW, page 12, 2000.

[186] P. Vassiliadis. A Survey of Extract-Transform-Load Technology. IJDWM, 5(3):1–27, 2009.

[187] P. Vassiliadis, M. Bouzeghoub, and C. Quix. Towards quality-oriented data warehouse usage and evolution. Inf. Syst., 25(2):89–115, 2000.

[188] P. Vassiliadis and A. Simitsis. Near real time ETL. In New Trends in Data Warehousing and Data Analysis, pages 1–31. Springer, 2009.

[189] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. Inf. Syst., 30(7), 2005.

[190] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In DOLAP, pages 14–21, 2002.

[191] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop YARN: yet another resource negotiator. In SOCC, pages 5:1–5:16, 2013.

[192] M. Vrhovnik, H. Schwarz, O. Suhre, B. Mitschang, V. Markl, A. Maier, and T. Kraft. An approach to optimize data processing in business processes. In VLDB, pages 615–626, 2007.

[193] F. Waas, R. Wrembel, T. Freudenreich, M. Thiele, C. Koncilia, and P. Furtado. On-demand ELT architecture for right-time BI: extending the vision. IJDWM, 9(2):21–38, 2013.

[194] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang. Map task scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality. In INFOCOM, pages 1609–1617, 2013.

[195] K. Wilkinson, A. Simitsis, M. Castellanos, and U. Dayal. Leveraging Business Process Models for ETL Design. In ER, pages 15–30, 2010.

[196] R. Winter and B. Strauch. A Method for Demand-driven Information Requirements Analysis in Data Warehousing Projects. In In Proc. HICSS, pages 1359–1365, 2003.

[197] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In EuroSys, pages 265–278, 2010.

Appendices

Appendix A

Quarry: Digging Up the Gems of Your Data Treasury

The paper has been published in the
Proceedings of the 18th International Conference on Extending Database Technology (EDBT 2015), Brussels, Belgium, pp. 549-552, 2015. The layout of the paper has been revised.
DOI: http://dx.doi.org/10.5441/002/edbt.2015.55

Abstract

The design lifecycle of a data warehousing (DW) system is primarily led by requirements of its end-users and the complexity of underlying data sources. The process of designing a multidimensional (MD) schema and back-end extract-transform-load (ETL) processes, is a long-term and mostly manual task. As enterprises shift to more real-time and 'on-the-fly' decision making, business intelligence (BI) systems require automated means for efficiently adapting a physical DW design to frequent changes of business needs. To address this problem, we present Quarry, an end-to-end system for assisting users of various technical skills in managing the incremental design and deployment of MD schemata and ETL processes. Quarry automates the physical design of a DW system from high-level information requirements. Moreover, Quarry provides tools for efficiently accommodating MD schema and ETL process designs to new or changed information needs of its end-users. Finally, Quarry facilitates the deployment of the generated DW design over an extensible list of execution engines. On-site, we will use a variety of examples to show how Quarry facilitates the complexity of the DW design lifecycle.

1 Introduction

Traditionally, the process of designing a multidimensional (MD) schema and back-end extract-transform-load (ETL) flows, is a long-term and mostly manual task. It usually includes several rounds of collecting requirements from end-users, reconciliation, and redesigning until the business needs are finally satisfied. Moreover, in today's BI systems, deployed DW systems, satisfying the current set of requirements is subject to frequent changes as the business evolves. MD schema and ETL process, as other software artifacts, do not lend themselves nicely to evolution events and in general, maintaining them manually is hard. First, for each new, changed, or removed requirement, an updated DW design must go through a series of validation processes to guarantee the satisfaction of the current set of requirements, and the soundness of the updated design solutions (i.e., meeting MD integrity constraints [114]). Moreover, the proposed design solutions should be further optimized to meet different quality objectives (e.g., performance, fault tolerance, structural complexity). Lastly, complex BI systems may usually involve a plethora of execution platforms, each one specialized for efficiently performing a specific analytical processing. Thus the efficient deployment over different execution systems is an additional challenge.

Translating information requirements into MD schema and ETL process designs has been already studied, and various works propose either manual (e.g., [97]), guided (e.g., [13]) or automated [18, 131, 146] approaches for the design of a DW system. In addition, in [52] a tool (a.k.a. Clio) is proposed to automatically generate correspondences (i.e., schema mappings) among different existing schemas, while another tool (a.k.a. Orchid) [44] further provides interoperability between Clio and procedural ETL tools. However, Clio and Orchid do not tackle the problem of creating a target schema. Moreover, none of these approaches have dealt with automating the adaptation of a DW design to new information needs of its end-users, or the complete lifecyle of a DW design.

To address these problems, we built Quarry, an end-to-end system for assisting users in managing the complexity of the DW design lifecycle.

Quarry starts from high-level information requirements expressed in terms of analytical queries that follow the well-known MD model. That is, having a subject of analysis and its analysis dimensions (e.g., Analyze the revenue from the last year's sales, per products that are ordered from Spain.). Quarry provides a graphical assistance tool for guiding non-expert users in defining such requirements using a domain-specific vocabulary. Moreover, Quarry automates the process of validating each requirement with regard to the MD integrity constraints and its translation into MD schema and ETL process designs (i.e., partial designs).

Independently of the way end-users translate their information requirements

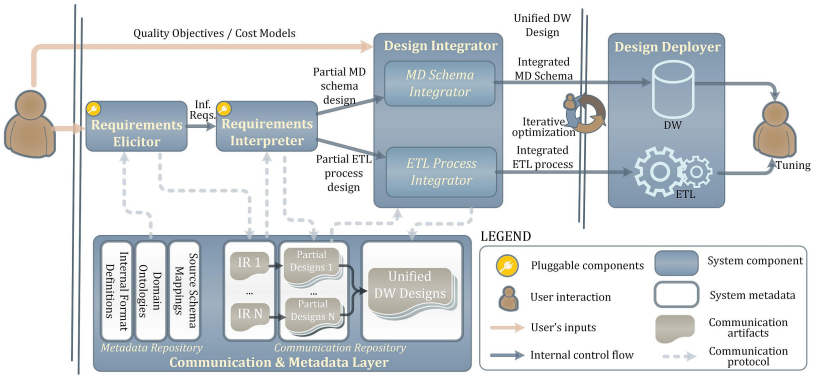


Fig. A.1: Quarry: system overview

into the corresponding partial designs, Quarry provides automated means for integrating these MD schema and ETL process designs into a unified DW design satisfying all requirements met so far.

Quarry automates the complex and time-consuming task of the incremental DW design. Moreover, while integrating partial designs, Quarry provides an automatic validation, both regarding the soundness (e.g., meeting MD integrity constraints) and the satisfiability of the current business needs.

Finally, for leading the automatic integration of MD schema and ETL process designs, and creating an optimal DW design solution, Quarry accounts for user-specified quality factors (e.g., structural design complexity of an MD schema, overall execution time of an ETL process).

Since Quarry assists both MD schema and ETL process designs, it also efficiently supports the additional iterative optimization steps of the complete DW design. For example, more complex ETL flows may be required to reduce the complexity of an MD schema and improve the performance of OLAP queries by pre-aggregating and joining source data.

Besides efficiently supporting the traditional DW design, the automation that Quarry provides, largely suits the needs of modern BI systems requiring rapid accommodation of a design to satisfy frequent changes.

Outline. We first provide an overview of Quarry and then, we present its core features to be demonstrated. Lastly, we outline our on-site presentation.

2 Demonstrable Features

Quarry presents an end-to-end system for managing the DW design lifecycle. Thus, it comprises four main components (see Figure A.1): Requirements Elicitor, Requirements Interpreter, Design Integrator, and Design Deployer.

For supporting non-expert users in providing their information requirements

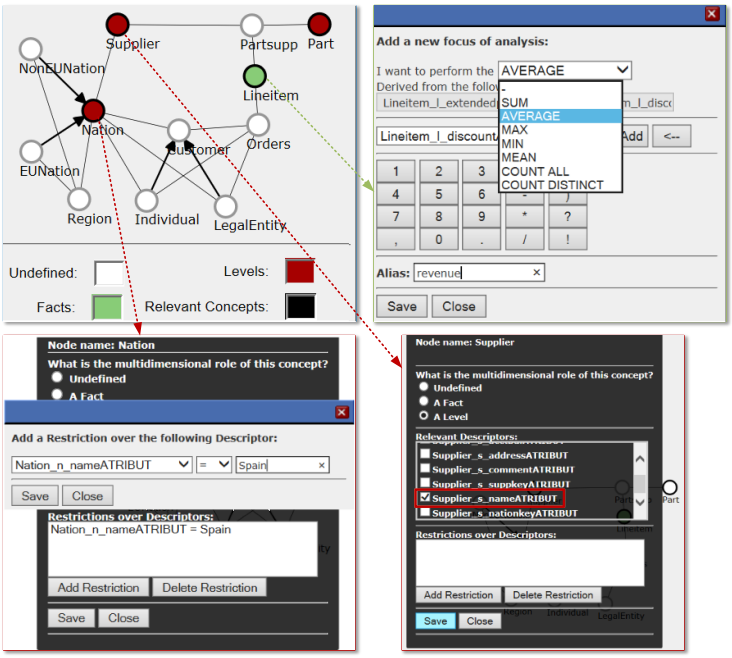


Fig. A.2: Requirements Elicitor

at input, Quarry provides a graphical component, namely Requirements Elicitor (see Figure A.2). Requirements Elicitor then connects to a component (i.e., Requirements Interpreter), which for each information requirement at input semi-automatically generates validated MD schema and ETL process designs (i.e., partial designs). Quarry further offers a component (i.e., Design Integrator) comprising two modules for integrating partial MD schema and ETL process designs processed so far, and generating unified design solutions satisfying a complete set of requirements. At each step, after integrating partial designs of a new requirement, Quarry guarantees the soundness of the unified design solutions and the satisfiability of all requirements processed so far. The produced DW design solutions are further sent to the Design Deployer component for the initial deployment of a DW schema and an ETL process that populates it. The deployed design solutions are then available for further user-preferred tunings and use.

To support intra and cross-platform communication, Quarry uses the communication & metadata layer (see Figure A.1).

2.1 Requirements Elicitor

Requirements Elicitor uses a graphical representation of a domain ontology capturing the underlying data sources. A domain ontology can be additionally enriched with the business level vocabulary, to enable non-expert users to express their analytical needs. Notice for example a graphical representation of an ontology capturing the TPC-H[1] data sources in top-left part of Figure A.2. Apart from manually defining requirements from scratch, Requirements Elicitor also offers assistance to end-users' data exploration tasks by analyzing the relationships in the domain ontology, and automatically suggesting potentially interesting analytical perspectives. For example, a user may choose the focus of an analysis (e.g., `Lineitem`), while the system then automatically suggests useful dimensions (e.g., `Supplier`, `Nation`, `Part`). The user can further accept or discard the suggestions and supply her information requirement.

2.2 Requirements Interpreter

Each information requirement defined by a user, is then translated by the Requirements Interpreter to a partial DW design. In particular, Requirements Interpreter maps an input information requirement to underlying data sources (i.e., by means of a domain ontology that captures them and corresponding source schema mappings; see Section 2.5), and semi-automatically generates MD schema and ETL process designs that satisfy such requirement. For more details and a discussion on correctness we refer the reader to [146].

In addition, Quarry allows plugging in other external design tools, with the assumption that the provided partial designs are sound (i.e., meet MD integrity constraints) and that they satisfy an end-user requirement. To enable such cross-platform interoperability, Quarry provides logical, platform-independent representations (see Section 2.5). Generated designs are stored to the Communication & Metadata layer using corresponding formats and related to the information requirements they satisfy.

2.3 Design Integrator

Starting from each information requirement, translated to corresponding partial MD schema and ETL process designs, Quarry takes care of incrementally consolidating these designs and generating unified design solutions satisfying all current requirements (see Figure A.3).

MD Schema Integrator. This module semi-automatically integrates partial MD schemas. MD Schema Integrator, comprises four stages, namely matching facts, matching dimensions, complementing the MD schema design, and integration. The first three stages gradually match different MD concepts and

[1] `http://www.tpc.org/tpch/`

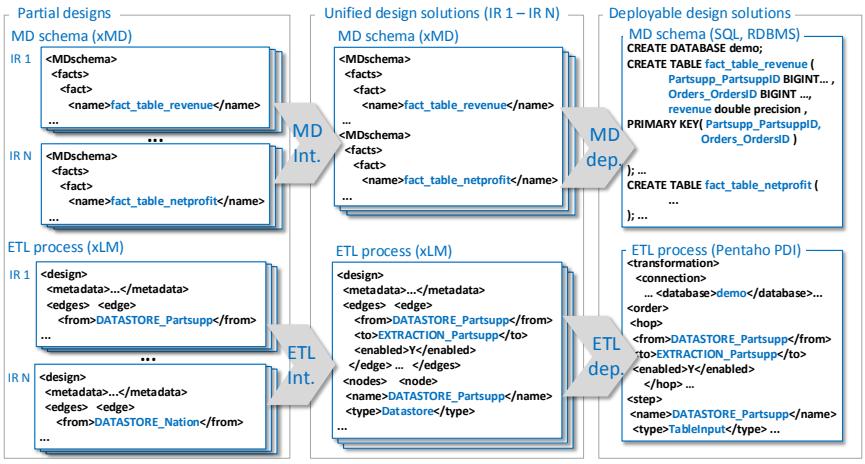


Fig. A.3: Design integration & deployment example

explore new DW design alternatives. The last stage considers these matchings and end-user's feedback to generate the final MD schema that accommodates new information requirements. To boost the integration of new information requirements spanning diverse data sources into the final MD schema design, we capture the semantics (e.g., concepts, properties) of the available data sources in terms of a domain ontology and corresponding source schema mappings (see Section 2.5). MD Schema Integrator automatically guarantees MD-compliant results and produces the optimal solution by applying cost models that capture different quality factors (e.g., structural complexity).

ETL Process Integrator. This module processes partial ETL designs and incrementally consolidates them into a unified ETL design. ETL Process Integrator, for each new requirement maximizes the reuse by looking for the largest overlapping of data and operations in the existing ETL process. To boost the reuse of the existing data flow elements when answering new information requirements, ETL Process Integrator aligns the order of ETL operations by applying generic equivalence rules. ETL Process Integrator also accounts for the cost of produced ETL flows when integrating information requirements, by applying configurable cost models that may consider different quality factors of an ETL process (e.g., overall execution time).

More details, as well as the underlying algorithms of MD Schema Integrator can be found in [91] and of ETL Process Integrator in [89].

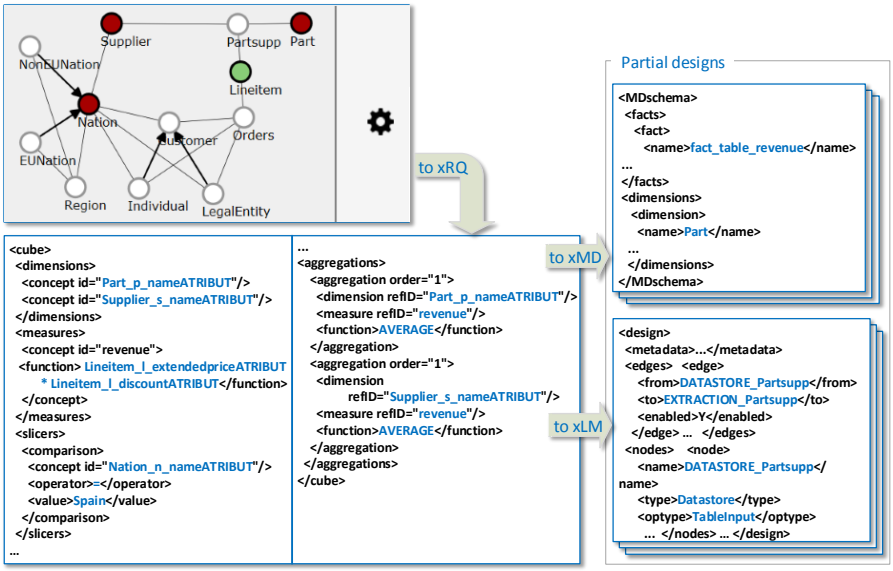


Fig. A.4: Example design process

2.4 Design Deployer

Finally, Quarry supports the deployment of the unified design solutions over the supported storage repositories and execution platforms (see example in Figure A.3). By using platform-independent representations of a DW design (see Section 2.5), Quarry is extensible in that it can link to a variety of execution platforms. At the same time, the validated DW designs are available for additional tunings by an expert user (e.g., indexes, materialization level).

2.5 Communication & Metadata Layer

To enable communication inside Quarry, the Communication & Metadata layer uses logical (XML-based) formats for representing elements that are exchanged among the components. Information requirements are represented in the form of analytical queries using a format called xRQ[2] (see bottom-left snippet in Figure A.4). An MD schema is represented using the xMD format[3] (see top-right snippet in Figure A.4), and an ETL process design using the xLM format [160] (see bottom-right snippet in Figure A.4). Moreover, the Communication & Metadata layer offers plug-in capabilities for adding import and

[2]xRQ's DTD at: `www.essi.upc.edu/~petar/xrq.html`
[3]xMD's DTD at: `www.essi.upc.edu/~petar/xmd.html`

export parsers, for supporting various external notations (e.g., SQL, Apache PigLatin, ETL Metadata; see more details in [93]).

Besides providing the communication among different components of the system, the Communication & Metadata layer
also serves as a repository for the metadata that are produced and used during the DW design lifecycle. The metadata used to boost the semantic-aware integration of DW designs inside the Quarry platform, are domain ontologies capturing the semantics of underlying data sources, and source schema mappings that define the mappings of the ontological concepts in terms of underlying data sources.

2.6 Implementation details

Quarry has been developed at UPC, BarcelonaTech in the last three years, using a service-oriented architecture.

On the client side, Quarry provides a web-based component for assisting end-users during the DW lifecycle (i.e., Requirements Elicitor). This component is implemented in JavaScript, using the specialized D3 library for visualizing domain ontologies in form of graphs. The rest of modules (i.e., Requirements Interpreter, MD Schema Integrator, and ETL Process Integrator) are deployed on Apache Tomcat 7.0.34, with their functionalities offered via HTTP-based RESTful APIs. Such architecture provides the extensibility to Quarry for easily plugging and offering new components in the future (e.g., design self-tuning). Currently, all module components are implemented in Java 1.7, whilst new modules can internally use different technologies. For generating internal XML formats (i.e., xRQ, xMD, xLM) we created a set of Apache Velocity 1.7 templates, while for their parsing we rely on the Java SAX parser. For representing domain ontology inside Quarry, we used Web Ontology Language (OWL), and for internally handling the ontology objects inside Java, we used the Apache Jena libraries. Lastly, the Communication & Metadata layer, which implements communication protocols among different components in Quarry, uses a MongoDB instance, and a generic XML-JSON-XML parser for reading from and writing to the repository.

3 Demonstration

In the on-site demonstration, we will present the functionality of Quarry, using our end-to-end system for assisting users in managing the DW design lifecycle (see Figure A.1). We will use different examples of synthetic and real-world domains, covering a variety of underlying data sources, and a set of representative information requirements from these domains depicting typical scenarios of the DW design lifecycle. Demo participants will be especially encouraged to

provide example analytical needs using Requirements Elicitor, and play the role of Quarry's end-users. The following scenarios will be covered by our on-site demonstration.

DW design. Business users are not expected to have deep knowledge of the underlying data sources, thus they may choose to pose their information requirements using the domain vocabulary. To this end, business users may use the graphical component of Quarry (i.e., Requirements Elicitor), and its graphical representation of a domain ontology. This scenario shows how Quarry supports non-expert users in the early phases of the DW design lifecycle, to express their analytical needs (i.e., through assisted data exploration of Requirements Elicitor), and to easily obtain the initial DW design solutions.

Accommodating a DW design to changes. Due to possible changes in a business environment, a new information requirement could be posed or existing requirements might be changed or even removed from the analysis. Designers thus must reconsider the complete DW design to take into account the incurred changes. This scenario demonstrates how Quarry efficiently accommodates these changes and integrate them by producing an optimal DW design solution. We will consider structural design complexity as an example quality factor for output MD schemata, and overall execution time for ETL processes. The participants will see the benefits of integrated DW design solutions (e.g., reduced overall execution time for integrated ETL processes, executed in Pentaho PDI).

Design deployment. Finally, after the involved parties agree upon the provided solution, the chosen design is deployed on the available execution platforms. In this scenario, we will show how Quarry facilitates this part of the design lifecycle and generates corresponding executables for the chosen platforms. We use PostgreSQL for deploying our MD schema solutions, while for running the corresponding ETL flows, we use Pentaho PDI.

4 Acknowledgements

This work has been partly supported by the Spanish Ministerio de Ciencia e Innovación under project TIN2011-24747.