Information Systems Frontiers manuscript No.
(will be inserted by the editor)

Quarry: A User-centered Big Data Integration Platform

Petar Jovanovic · Sergi Nadal · Oscar Romero · Alberto Abelló · Besim Bilalli

Received: / Accepted:

Abstract Obtaining valuable insights and actionable knowledge from data requires cross-analysis of domain data coming typically from various sources. Doing so, inevitably imposes burdensome processes of unifying different data formats, discovering integration paths, and all this given specific analytical needs of a data analyst. Along with large volumes of data, the variety of formats, data models, and semantics drastically contribute to the complexity of such processes. Although there have been many attempts to automate various processes along the Big Data pipeline, no unified platforms accessible by users without technical skills (like statisticians or business analysts) have been proposed.

In this paper, we present a Big Data integration platform (*Quarry*) that uses hypergraph-based metadata to facilitate (and largely automate) the integration of domain data coming from a variety of sources and provides an intuitive interface to assist end users both in: (1) data exploration with the goal of discovering potentially relevant analysis facets, and (2) consolidation and deployment of data flows which integrate the data, and prepare them for further analysis (descriptive or predictive), visualization, and/or publishing. We validate *Quarry*'s functionalities with the use case of World Health Organization (WHO) epidemiologists and data analysts in their fight against Neglected Tropical Diseases (NTDs).

Keywords Data Integration · Big Data · Data-Intensive Flows · Metadata

1 Introduction

The importance of data in today's society is unquestionable. Available data in almost any domain create an opportunity for users to obtain valuable in-

Universitat Politècnica de Catalunya (BarcelonaTech)
Campus Nord Omega-125, UPC - dept ESSI, C/Jordi Girona 1-3, E-08034 Barcelona, Spain,
E-mail: {petar,snadal,oromero,aabello,bbilalli}@essi.upc.edu

sights and contextualize their business processes, as well as to get actionable knowledge and competitive advantages, by performing advanced (descriptive or predictive) data analysis [37]. However, in modern data-intensive ecosystems the setting is moving from modeling well-defined domains with few structured data sources towards the scale of hundreds and thousands of evolving sources, each providing abundance of data, in a variety of forms. Consequently, in such Big Data settings, the data management requirements go beyond traditional data integration solutions [25], and integrating and preparing data for final exploitation require more complex data transformations, efficiently handling large quantities of data [20, 4].

Moreover, enabling an integrated, cross-organizational strategy for advanced analytics requires close collaboration between domain experts, data analysts (i.e., statisticians or machine learning experts), and data stewards (i.e., data management experts). To facilitate this, in Big Data settings, we need more complex processes than those typically encountered in the data integration lifecycle [22]. Namely, first, new or evolved data sources, many of them external, need to be *registered* by data stewards and made available for domain experts and data analysts. Such sources should then be jointly *explored* to assess their potential for the analysis, and to decide on the analytical facets of interest. This process, crucial for settings with high variability of likely unknown sources, includes the selection of relevant features (e.g., identifying predictors for model construction) and the identification of related descriptive statistics for those variables. Once data sources are explored, data stewards should prepare tailored portions of selected data for performing the analytical tasks. This process consists of *deploying* a data flow to, first extract the features of interest, and further perform data preparation tasks, including unifying formats and models, data pre-processing, and integrating different sources. Due to possible poor quality of external data sources and the need for more advanced (e.g., predictive) analysis, it may also be required *extending* the flows with various complex data transformations to improve data quality factors and prepare the data for their final exploitation by data analysts. Moreover, understanding the steps that undergo such data preparation opens the door to its optimization, an important step having in mind that these data flows in Big Data settings typically work with large quantities of data and require complex transformation over them. In addition to the classical flow optimization [43], given the dynamicity of different data flows and the fact that their parts are typically shared by different end-users (with high temporal locality - 80% of data being reused within minutes or hours [10]), more advanced techniques like multi-flow optimization or view materialization may also be required.

Currently, such tasks are manually handled, relying on data stewards to orchestrate numerous specialized technologies that cover specific processes in the data integration lifecycle [22], thus yielding the whole process burdensome and impractical for large scale settings. This clearly raises the need for automating different processes along the data preparation pipeline and assisting end users in carrying out their analytical tasks. At the same time, having well-defined and semantically rich metadata (i.e., data that describes the data

themselves, the platform under use, and the users of the platform) is identified as a cornerstone for boosting the automation of different parts of the data preparation pipeline [36,9]. In parallel, it has been advocated that integration of data sources must be led by "the users and their information seeking activities" [16], hence claiming for user-centric solutions, more accessible for data analysts to conduct richer (i.e., better contextualized) analysis, without requiring data management proficiency. However, current systems still do not completely cover the previously discussed data integration lifecycle for Big Data settings, but focusing typically only on some parts and each defining their own metadata, thus hindering their interoperability.

In this paper, we present *Quarry*, a comprehensive data integration platform, primarily dealing with, but not limited to the variety dimension of Big Data. *Quarry*, supported by semantically enriched metadata, provides a semi-automated solution for the data preparation tasks, with the main objective to assist non-technical users during their analytical efforts. *Quarry* represents a platform where *virtual* and *physical* data integration complement each other. In particular, the virtual approach, which avoids data materialization, can aid on the combined exploration of data sources with no cost for space or synchronization. Conversely, the physical approach, which stores data resulting from integration, can provide benefits on saving reexecution of complex data preparation pipelines, hence finally persisting analysis-ready data for the end user (e.g., data cubes, data matrices or graphs).

To the best of our knowledge, *Quarry* is the first platform to support collaborative, cross-organization analytics, by covering the complete data integration lifecycle for Big Data settings - from here onwards, we refer to it as *big data integration lifecycle*. In particular, *Quarry* defines four functional modules that facilitate the main tasks in the data preparation pipeline. During the data exploration phase, given the high-level user's information needs expressed in a domain-specific vocabulary, the *Query Manager* module automatically searches the possible integration paths (i.e., paths relating the concepts of different data sources) to resolve a user query. At the same time, it guarantees that the searched concepts and those found on the integration paths can be answered from the existing data sources, by means of mappings. From each resolved query, rewritten in terms of the data sources, the *Query Manager* defines a data flow that can bring analysis-ready data to the end user. This way, *Quarry* facilitates the analysts in discovering how the data sources at hand integrate. Moreover, by integrating the data, it also helps identifying relevant features for the analysis at hand, whose relevance in isolation might not be apparent. The data flow can be additionally extended by the *Data Quality Manager* module to resolve potential data quality issues and enable required analytical tasks. Finally, the *Flow Manager* module can deploy such data flow for periodic execution, providing a refreshed view of data (e.g., for periodical reporting). To boost the reuse of the data processing tasks and optimize resource utilization, *Flow Manager* can also propose a multi-flow execution for several data flows, and select the optimal materialization level and storage format for the resulting data.

To automate these tasks, *Quarry*, through its *Integration Management* module generates and stores a variety of metadata artifacts adopted from the foundations of data integration [31], as well as from recent works tackling data variety [47]. To represent such metadata artifacts, we propose a unified hypergraph-based metadata model, which enables applying different graph algorithms over the complete metadata structures, hence boosting the automation of the data preparation tasks throughout the platform.

Contributions. We propose a comprehensive architectural view of the platform to facilitate the complete data integration lifecycle, identifying the needed functional modules and the potential for their automation through efficient metadata exploitation. In particular, our main contributions are as follows:

- We present a unified Big Data integration platform, which enables end users without technical skills to explore data spanning various sources, integrate them, and prepare them for further specific analysis.
- We illustrate a compact, hypergraph-based model to flexibly and efficiently represent metadata artifacts.
- We describe a functional module to semi-automatically build the metadata repository in an incremental manner, as well as modules to facilitate the data preparation tasks by efficiently exploiting such metadata.
- Finally, we validate the usability of our approach by showcasing its application in an ambitious real world use case at the World Health Organization (WHO) to control and eliminate Neglected Tropical Diseases (NTD)[1].

Outline. The remainder of the paper is structured as follows. Section 2 discusses the related work. Next, Section 3 introduces the use case of applying *Quarry* to the project of control and elimination of NTDs at WHO. Section 4 describes *Quarry*'s functionality in a nutshell, presenting the main big data integration lifecycle workflows, and kinds of metadata used throughout the platform. Section 5, presents the core functional architecture of *Quarry*, providing detailed descriptions of its metadata, its functional modules and the interaction among them. Finally, in Section 6, we conclude the paper and discuss possible future directions.

2 Related Work

In this section, we review the literature on systems and architectures for Big Data integration to enable data analytics. To this end, we will distinguish among physical and virtual data integration systems. Nevertheless, we do not consider systems restricted to descriptive data analysis (i.e., those that perform traditional data warehousing and business intelligence tasks). These systems limit the availability to pre-defined data analysis pipelines since they load and manage data compliant with a pre-defined data model. In our use case, such assumption does not hold, and the goal of this paper is thus to present an architecture to support advanced and flexible data analysis.

[1] `https://www.who.int/neglected_diseases/disease_management/wiscentds`

2.1 Physical data integration systems

Data consolidation requires the physical data integration of the data at hand. While losing flexibility when dealing with source data, these systems exhibit good performance due to their unified consolidated view of data.

Data lakes. The data lake [49,39] is nowadays the most widespread reference architecture for physical integration of an heterogeneous set of data sources in their original format following the *load-first model-later* approach. In practice, a data lake is implemented via a distributed file system, which provides the foundations for distributed and parallel processing, together with fault tolerance thanks to data replication. Nonetheless, data lakes *per se* do not provide any means for data management or processing, hence there is a need to enrich them with additional features such as data homogenization, cleaning, deduplication or querying.

One approach to extend data lakes with richer semantic metadata is Constance [23]. Constance automatically extracts structural and semantical metadata from the data lake's contents in order to offer a unified query interface over it. Further, the extracted schema is linked via GAV mappings to the semi-structured data sources in order to provide query answering over them. Alternatively, Google's solution to manage their data lake Goods [24] crawls, indexes and integrates a plethora of datasets in order to provide an organized and unified view over them. One of its key features is the relationship graph, a data structure that encodes automatically extracted relationships between datasets like dataset containment, provenance or content similarity. A search engine makes use of these metadata to enable further exploration of the datasets content. Furthermore, the ADF architecture [21] builds on top of the Hadoop File System to provide several optimization techniques based on dynamically changing metadata. This includes features such as dynamic partition elimination, advanced indexing techniques or cost-based workload balancing.

Despite the many extensions to data lakes, they all purely focus on the management of data and do not on their transformations and flows.

Polystores. Polystores are the evolution of federated database management systems, which inherit their foundational principles but differ on the heterogeneity of the data sources, and the need to use and coordinate many specialized engines [46]. The BigDAWG polystore system [14] organizes datasets into islands of information. Each island represents a collection of engines that provides a single data model, and thus can be accessed using the same query language. Source data are connected to islands via shims, which are dedicated software modules that extract data from the specific source model into the island's target data model. Users query islands using their specific query language, which is translated into partial queries over the shims.

The Data Tamer system [47] pays special attention to the process of data curation and its scalability challenges. It adopts a wrapper-based architecture to extract data from the sources into *sites*, which are collections of key-value

pairs. Data Tamer contains components that mainly focus on functionalities for schema integration, entity consolidation and record linkage, which make heavy use of machine learning techniques. Alternatively, an evolution of Data Tamer, is the Data Civilizer system [12]. Besides including the previously described functionalities, this system also computes the linkage graph (i.e., a graph that represents relationships among tables or keys) as well as computing queries that discover join paths with data cleaning operators. Finally, Termite [15] follows an alternative approach for schema integration using word embeddings where entities, rows, columns and paragraphs are represented. An embedding is a data structures that capture the semantic similarity of terms over a corpus of documents. From these embeddings, Termite computes distances which determine how close or related different schema elements are. The Termite-Join operator returns, given an input entity, the top K closest elements in the embedding in terms of the distance function.

Overall, we can say that polystores focus on representing metadata for specific tasks, and thus are commonly limited on aspects such as the data flows optimization and their management for data consolidation.

2.2 Virtual data integration systems

Virtual data integration systems are ideal for data exploration tasks, since they provide higher degree of flexibility. However, the complexity of these systems makes them unfeasible for intensive computational tasks such as predictive analysis.

Knowledge-representation systems. Most virtual data integration systems have their origins in knowledge representation, and thus base their architecture on the orchestration of different reasoning services on the so-called ontology-based data access (OBDA) paradigm. An example is the VADA architecture [30], aimed to support the process of data wrangling (i.e., extracting, cleaning and collating datasets). VADA is composed by a set of transducers, which are software components with input and output dependencies defined as Datalog queries over the knowledge base or the state of a transducer. A reasoner acts upon such knowledge base using Vadalog, which is a language from the Datalog±, and provides services for schema matching, schema alignment, data fusion or data quality, among others. An alternative system is Ontop [8], an OBDA solution that supports answering SPARQL queries over OWL 2 QL ontologies mapped via R2RML mappings to Relational databases. Ontop adopts the *global-as-view* family of mappings, which guarantees that the complexity of query rewriting depends only on the ontology and mappings and not on the instances. A similar approach is that adopted by Morph [38], which also employs R2RML mappings for answering queries on Relational databases.

Knowledge-representation systems emphasize thus the provision of flexible reasoning mechanisms to directly query the sources. However, aspects such as the further deployment of these queries into data-intensive flows considering data quality aspects are not covered.

Dataspaces. A dataspace is a data integration system that aims to tackle the variety challenge by reducing the usual upfront and maintenance cost that such systems have [17]. Dataspaces claim for the adoption of a flexible and dynamic pay-as-you-go approach where different integration tasks are automated. One of their key features is the management of uncertainty, and thus their query mechanisms must support a certain degree of ambiguity [42].

Personal Information Systems (PIMs) are dataspaces that aim to capture the totality or part of the digital life of an individual across several platforms [1]. Their main goal is to offer user-friendly mechanisms so that users are in control of their personal data, hence their query language includes features such as NLP/KDD/search techniques for personal data, personalized answers and explainability of query results. PIMs also emphasize the dimension of ethical data management as such systems commonly interrelate policy makers, regulators, organizations with the end-user.

In the case of dataspaces, the focus is consequently on information retrieval and empowering users in the management of their data. This entails that data quality and transformation issues are typically less relevant.

	Purpose				Technique					Metadata				
										Type		Data struct.		
System	Integration management	Flow management	Data Quality mgt.	Query Management	Learning	Ad-hoc Technique	Crawling	Distance/ranking	Query Rewriting	Structural	Semantic	Graph/ontology	Relations	Embeddings
Constance [23]	✓		✓	✓		✓			✓	✓	✓	✓		
Goods [24]		✓	✓	✓			✓	✓		✓	✓	✓		
ADF [21]		✓				✓				✓			✓	
BigDAWG [14]	✓			✓		✓			✓	✓			✓	
Data Tamer [47]	✓		✓		✓			✓		✓	✓	NA	NA	NA
Data Civilizer [12]	✓		✓	✓	✓			✓	✓	✓	✓	✓		
Termite [15]	✓			✓	✓			✓		✓	✓		✓	✓
VADA [30]			✓	✓		✓			✓	✓	✓	✓		
Ontop [8]	✓			✓		✓			✓	✓	✓	✓		
Morph [38]	✓			✓					✓	✓	✓	✓		

Table 1: Characteristics of related systems

2.3 Summary

Table 1 categorizes the related work according to the purpose of each approach in the context of the data integration workflow we devise. We also show the presented techniques, kind of metadata adopted and the data structure that is used to encode such metadata. We conclude that currently there is no system spanning the whole data integration lifecycle: i.e., data exploration (through

virtual integration) and data consolidation (materializing integrated data for further analysis). Both approaches have well-known pros and cons but dealing with them as separate systems hinders the automation of the whole process. *Quarry* is a unique solution providing a unified view of metadata constructs that facilitates data consolidation after conducting data exploration tasks on the available data sources. As result, our level of automation is higher than using independent systems for each task since current systems are not prepared to interoperate and exchange their produced metadata.

3 Use Case: The Fight against NTDs at WHO

To validate the usability of *Quarry*, we present its application for building a system to control and eliminate Neglected Tropical Diseases (NTDs) at the World Health Organization (WHO). NTDs form a group of 21 diseases with different, sometimes very complex aspects, all having in common that they affect population typically from economically challenging, rural areas of the world. Depending on the disease (e.g., Chagas Disase, Leishmaniasis), transmission routes can vary (e.g., congenital from a mother to a child, blood transfusion, organ transplantation, insect bites), as well as the causes of its spreading worldwide (e.g., high presence and reproduction of insects in endemic areas, traveling to endemic areas, migration flows). All this makes the control and eventual elimination and/or eradication of NTDs very challenging.

In addition, having that NTDs are still very often overlooked by national health information systems, makes in most cases impossible to ensure good-quality data collection at country level, resulting that the data reported by health ministries are often incomplete. Nevertheless, historical or prospective data related to NTDs may also be collected by other actors (e.g., non-governmental organizations, researchers) or extracted from other existing systems (e.g., pharmacovigilance systems like Uppsala Monitoring Center[2] or outbreak alert systems like ProMED[3]). Still, these data can be largely fragmented covering different aspects of the diseases, and very heterogeneous in formats and data models used. Moving towards the target 3.3 of Sustainable Development Goals (SDGs), i.e., ending the NTD epidemics by 2030, WHO is on the need for a more complete and comprehensive epidemiological picture of NTDs' status worldwide (e.g., discovering epidemiological silence, studying coinfection/comorbidity of diseases with overlapping geographical prevalence). Intrinsically, they need to integrate NTDs data coming from a variety of data sources, and create a consolidated data platform, hence enabling complex analytics to be performed by epidemiologists and statisticians.

Therefore, in the rest of the paper, we exemplify how *Quarry* automates the integration of the heterogeneous NTDs data and assists WHO users in preparing data for further data analysis. We are focusing on a specific scenario for detecting possible lack of official disease reporting at the country

[2] `https://www.who-umc.org`

[3] `https://www.promedmail.org`

level (i.e., epidemiological silence), by querying alternative sources of information. In such a scenario, the official *diagnostic* and *treatment* data coming from ministries of health, can be crossed with information about the *distribution of medicine* to the same areas, and this can be analyzed in terms of the *country's main population* or additionally including *migrants*, which is especially important for the territories affected by the migration from disease endemic areas. For better understanding, we are focusing on the domain of Chagas disease[4].

We first here introduce the underlying data sources, and then in Section 5, we demonstrate the functionalities of *Quarry* by applying it in the given scenario. Importantly, notice that while *Quarry* is being applied in a real world project, for data privacy reasons, the data exemplifying the *Quarry*'s functionalities are either publicly available (i.e., published by United Nations) or synthetically generated to simulate real data internal to WHO.

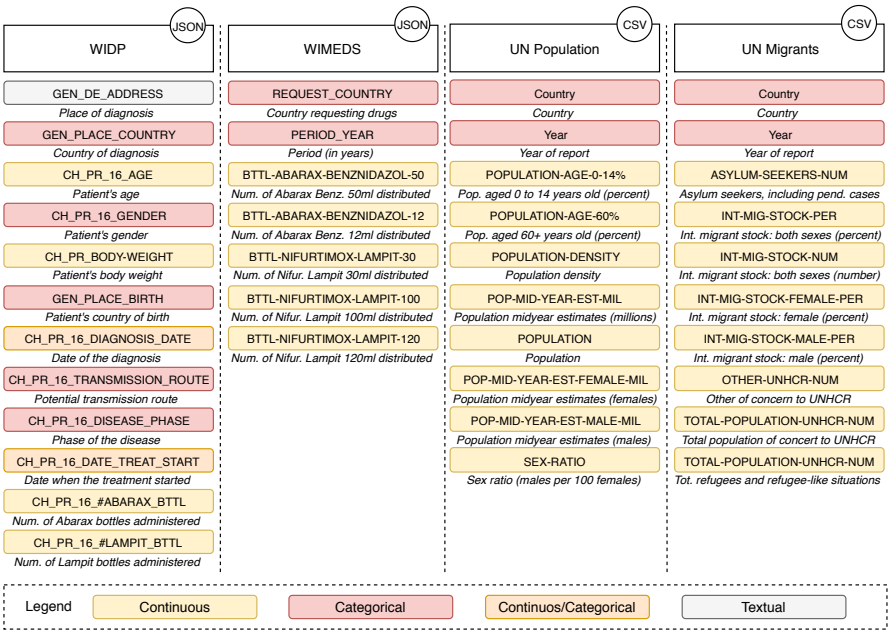


Fig. 1: Data sources and their corresponding variables

3.1 Data sources under consideration

Official country's surveillance data (WIDP). One of the main sources of NTD information is the country disease surveillance and monitoring system. To support countries in such data collection tasks, WHO has provided a

[4] `https://www.who.int/chagas/en`

global WHO Integrated Data Platform (WIDP)[5], powered by District Health Information System 2 (DHIS2)[6]. Data coming from WIDP register either individual events (e.g., patience diagnosis and treatment, dwellings inspection) or collective reports (e.g., total number of infected patients over a period of time and at a certain territory, total number of administered drugs). In our scenario, we focus on the individual diagnosis and treatment data of Chagas disease patients at a country. WIDP provides data in JSON format, through the variables reported in the first column of Figure 1.

Medicine distribution data (WIMEDS). In addition to WIDP, WHO has also developed a system for tracking, planning, and monitoring medicine distribution between WHO and countries, or manufacturers and countries, i.e., WHO Integrated Medical Supplies System (WIMEDS)[7], powered by BonitaSoft[8]. WIMEDS can provide data directly as a database extract (being an internal WHO tool) or by means of a public API in JSON key-value format (for external users), through the variables summarized in the second column of Figure 1 .

Population and immigration data (from United Nations). Lastly, to contextualize the analysis of the official data, we include part of United Nation's data[9] referring to countries' population and immigration. Notice that this is important for monitoring the main indicators about the disease, which are typically computed over the territory population or total number of potentially affected people. UN data can be accessed through a public API and are extracted as CSV/Excel files. They provide a set of variables yearly reported on country's population and country's immigration (see last two columns of Figure 1).

4 *Quarry* in Support of the Big Data Integration Lifecycle

In *Quarry*, we rely on a hypergraph-based metadata representation, which is flexible and extensible enough to store different levels of detail, while at the same time enabling automatic processing by means of graph traversals and pattern matching queries. In this section, we primarily introduce the metadata subsets used to automate the processing, as well as the high level workflows of the big data integration lifecycle that *Quarry* supports.

[5] http://mss4ntd.essi.upc.edu/wiki/index.php?title=WHO_Integrated_Data_Platform_(WIDP)

[6] https://www.dhis2.org

[7] http://mss4ntd.essi.upc.edu/wiki/index.php?title=WHO_Integrated_Medical_Supplies_System_(WIMEDS)

[8] https://www.bonitasoft.com

[9] http://data.un.org

4.1 Subsets of Metadata

To automate the processing of the data integration workflows, *Quarry* builds on well-known data integration artifacts [13,31], and takes advantage of additional ones needed for dealing with the complexity of Big Data systems [51]. Hence, to deal with the abundance of external data sources and advanced analytics pipelines, *Quarry* benefits from *data-intensive flows* that describe and record data provenance and traceability in the platform, and *service-level agreements* that store the preferred quality requirements set by end users. In this section, we present each metadata subset considering their specific adaptation for the *Quarry* platform. Note that the presented list is intentionally not exhaustive, as the metadata repository is easily extensible and could, for example, include additional metadata such as data provenance (i.e., information recording the steps used to derive any piece of data) among others.

Global schema. The global, or *mediated* schema encodes the domain of interest and the semantics of the concepts and relationships of the underlying data. Furthermore, *Quarry* uses the global schema (or parts of it) to expose the underlying data in an integrated manner to the end user for posing queries inside the platform (see Figure 2b). The format and complexity of the global schema representation can vary, from full semantic solutions in terms of ontology (e.g., OWL [45]), to lighter solutions that represent basic concepts of the conceptual schema for the domain at hand (e.g., UML [33]).

Source schemata. Also referred to as source descriptions, they define the core properties of the sources that the system needs to be aware of. These properties are, for instance, a list of variables that the source exposes, or annotations on whether data can be considered as complete or not. In practice, the source schema models and exposes the structure of the sources, and is used by the data integration system to answer the individual queries over each data source. Similarly to the global schema, source schemata can be represented in either Relational or other more flexible non-Relational forms (e.g., graphs).

Schema mappings. Schema mappings connect the global schema with the source schemata in a declarative way. These are traditionally categorized as *global-as-view* (GAV), *local-as-view* (LAV), or the more general *tuple-generating dependencies* (GLAV); which directly determine how queries are processed [18]. As discussed in more detail in Section 5.1, in *Quarry*, we use LAV mappings to represent correspondences among source schemata and the global schema, given their inherent flexibility in front of evolutionary changes at data sources.

Wrappers. A wrapper implements an extraction program using the source-specific query language, receives the answers and potentially applies some basic transformations to them [19]. Automatic wrapper construction is a difficult task (e.g., a web crawler that parses HTML pages and returns sets of tuples), which is commonly tackled via machine learning techniques [32]. Hence, one needs to usually account for incompleteness or approximate answers. In

Quarry, the wrapper construction can be partially automated by means of applying transformation rules [26] to generate extraction queries for the data source at hand.

Data-intensive flows. Data-intensive flows (DIFs) are processes that deliver data in user-preferred and analysis-ready formats, from a multitude of sources. For DIFs, *Quarry* as well uses a hypergraph-based solution, where a DIF at a logical level is represented as a directed acyclic graph (DAG), with its nodes being either data stores or operations, and its edges representing a directed data flow among the nodes of the graph [27]. Furthermore, *Quarry* relies on the flow translation functionalities, like those presented in [29], to generate an executable version of a DIF (i.e., eDIF) that can be run on the preferred execution engine.

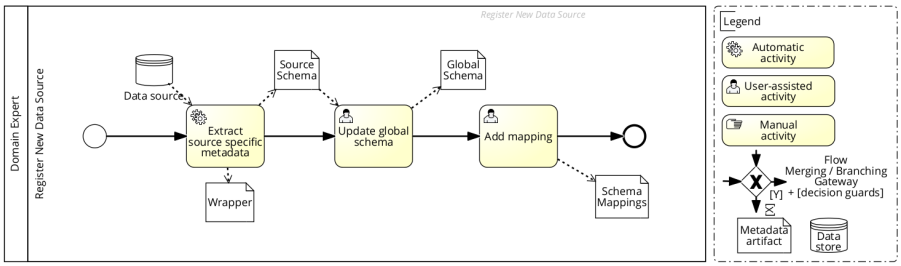
Service-level agreements. These are quality criteria, or non-functional requirements, that specify the degree to which a certain property must be fulfilled. In *Quarry*, we will focus on data management-oriented service-level agreements (SLAs) [34] and use them in terms of maximization of a multi-objective utility function with different weights for the components.

Rules. When considered in isolation, different data sources may have different business rules to whom they must adhere. Some of these rules may seize to hold, when different data get integrated with each other. However, some others could span to hold even on the final integrated data. To this end, we aim for the latter and those rules include: Funcional dependencies (FDs), Multi-valued dependencies (MDs), or any kind of business rule that can be used in the validation of data. *Quarry* uses these metadata to facilitate the automation of data repairing/cleaning when violations occur. A typical case is for instance when duplicates or inconsistencies with respect to some master data appear.

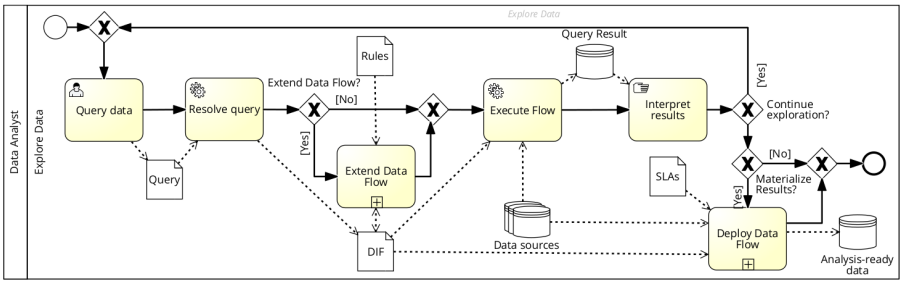
4.2 Big Data Integration Workflows

Importantly, *Quarry* considers different types of users interacting with the platform during the data integration lifecycle, namely: (1) *Domain Expert*, a user with a profound knowledge of the domain at hand (but usually no technical skills), who typically interacts with the platform by defining necessary domain-specific metadata and resolving semantic ambiguities among considered data sources; (2) *Data Steward*, a technical IT user, who, assisted by automated processes in the platform, creates the necessary infrastructure for analysis specific data preparation (e.g., data flow implementation, execution engine selection and configuration); and (3) *Data Analyst*, finally, is a user with somewhat domain knowledge, expert statistical knowledge to decide the specific analysis of the data, but no technological skills to integrate and prepare such data for the analysis.

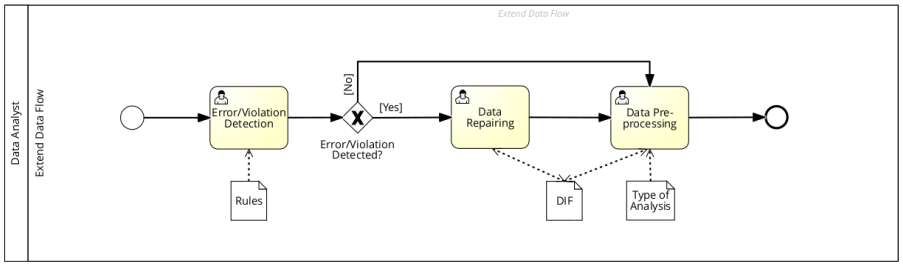
Furthermore, we consider four core workflows that occur within the big data integration lifecycle (see Figure 2), namely: (a) *register new data source*, (b) *explore data*, (c) *extend data flow*, and (d) *deploy data flow*.



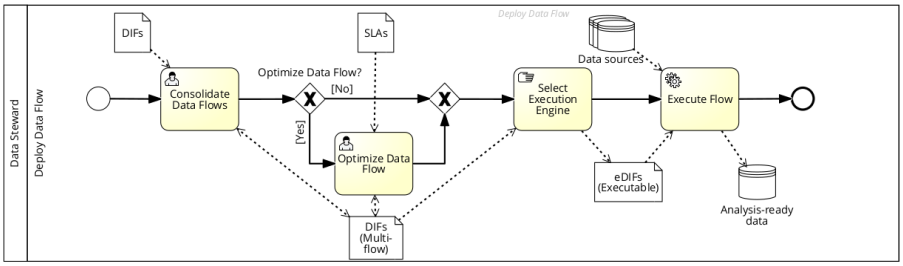
(a) *Register New Data Source*



(b) *Explore Data*



(c) *Extend Data Flow*



(d) *Deploy Data Flow*

Fig. 2: Big Data Integration Lifecycle Workflows

Register new data source (Figure 2a). The data integration workflow begins by gathering and registering the data sources that will be considered. To do so, the source-specific metadata, like schema of the data source, or format and model conversion, need to be extracted. Furthermore, to guarantee that all the information exposed from a new data source is available for querying, we need to potentially update the global schema with new information. Besides the updated global schema, in this step, we also discover how the data from a new data source relate to our global schema, in terms of schema mappings. These activities, are largely automated in our system, but may still require the assistance of *Domain Experts* and *Data Stewards* to resolve semantic ambiguities occurred when matching a new source schema with the existing global one for certain wrappers [35].

Explore data (Figure 2b). Once data sources are added to the platform, and the necessary metadata are generated, the *Data Analysts* may start exploring them in order to scrutinize their characteristics and relationships with other data inside the platform. This will also allow to decide on his/her analysis of interest. In a typical scenario, the *Data Analyst* poses a query in a domain-specific language over the global schema, from which the execution plan (in terms of a data flow) is returned. The query is then automatically resolved and the corresponding data flow is executed over the sources to produce the query results, which are then manually interpreted by the analyst and the exploration may continue or be stopped if the analyst has found the analysis facet of his/her interest. In addition, the *Data Analyst*, assisted by the tool may extend the data flow with more complex data transformations and deploy it for periodical execution with other data flows inside the platform. These subprocesses are further explained in more detail.

Extend Data Flow (Figure 2c). While launching queries for exploring the data, the *Data Analyst* may immediately want to apply some transformations either with the aim of enabling the analysis, or improving it. The former applies when due to some violations the analysis cannot be performed (e.g., inconsistent data) — *Quarry* facilitates the analyst by recommending repairing transformations [11], if violations are detected. The latter however, is required when the *Data Analyst* already wants to apply some transformations for the sake of improving the results of his analysis. To this end, in the *Data preprocessing* component, by learning from historical knowledge and knowing the type of analysis (e.g., supervised learning problem), *Quarry* is able to recommend additional complex transformations (e.g., feature selection) that have potentially positive impact on the final analysis [5] (e.g., increase the predictive accuracy of the supervised learning).

Deploy Data Flow (Figure 2d). After the users select their analysis and trigger the materialization of the data, *Quarry* needs to deploy the materialization flows over executable engines to periodically obtain analysis-ready data for the end user. First, given that a variety of users with similar analytical needs may interact with the platform, data flow deployment starts by consolidating data flows coming from different users, finding shared data processing

parts among the flows, and proposing a multi-flow execution. Such consolidation is done automatically starting from the matching sources of data flows and iteratively, in each step attempting to match more data flow operations, possibly having to reorder their positions inside the flows. Consequently, each found match extends the shared data flow part and guarantees its coincidence among all the consolidated flows. The flows may be then further optimized, by means of selecting the possible materialization of intermediate results, and selecting specific formats in which data will be stored. Lastly, an execution engine is selected for the input flow and it is translated into an executable language ready for deployment. Throughout the complete workflow, the *Data Steward* may be prompted to give technical support for the flow managing tasks (e.g., selecting optimal materialization level or storage format), until the flow is finally ready for the automatic execution over the selected engine.

In the following section, we present the functional architecture of *Quarry*. We also report the application of each of *Quarry*'s core modules to the NTD use case introduced in Section 3, by referring to the data integration workflows that the module implements.

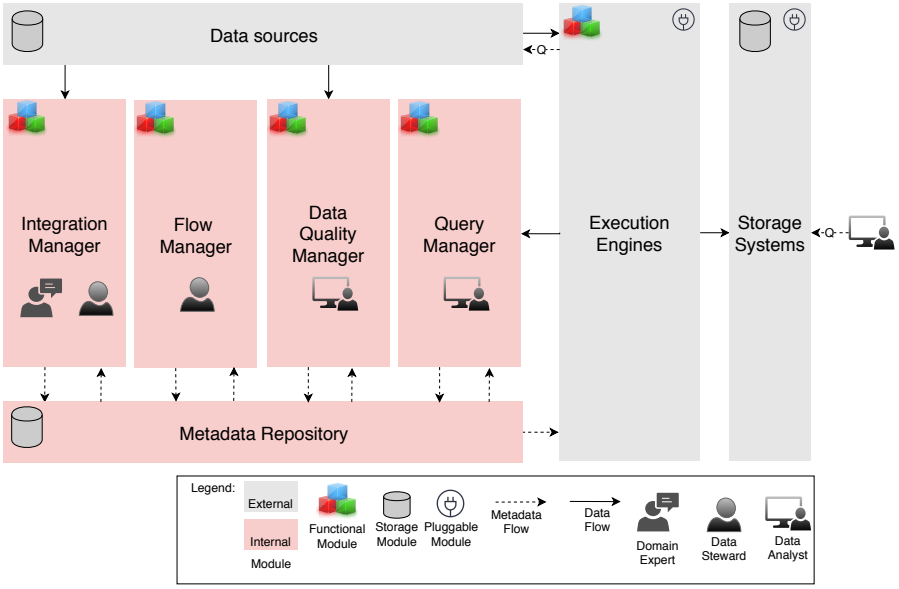


Fig. 3: *Quarry* Functional Architecture

5 *Quarry*'s Functional Architecture

To support and automate the big data integration lifecycle, *Quarry* relies on four core functional modules (i.e., *Integration Manager*, *Query Manager*,

Data Quality Manager, and *Flow Manager*), which generate and use a set of metadata artifacts stored inside the *Metadata Repository* (see Figure 3). In addition, *Quarry* complements its functionality by connecting to a group of external *Execution Engines* (e.g., *Apache Spark*[10], or *Apache Flink*[11]) that can be plugged into the platform to run previously generated data flows. Such data flows produce the *analysis-ready* data that can be then stored in a variety of external *Storage Systems* (e.g., *Hadoop Distributed File System-HDFS*[12], a graph analytics system such as Neo4J[13] or a Relational DBMS like *PostgreSQL*[14], among others), and be ready for further exploration by the data analysts (e.g., OLAP analysis, graph analytics, classification and predictive analysis, visualization, or publishing).

In the following subsections, we introduce in more detail the functional architecture of *Quarry*, including metadata artifacts considered inside the *Metadata Repository*, as well as the core modules to support data management workflows described in Section 4.2.

5.1 Metadata Repository

The *Metadata Repository* module is responsible of organizing the catalog of metadata artifacts in *Quarry*. Such catalog sits at the core of the platform and enables the large degree of automation of both, data exploration and consolidation tasks, as well as efficiently bridging between them by reusing the same concepts.

Graphs have been extensively used in scenarios where the relationships topology of data is as important as the data themselves [3]. Hence, *Quarry* advocates for the use of hypergraphs to represent all metadata artifacts to automate the end-to-end integration process. Unlike other approaches that, for example, combine data structures with logical rules for mappings, by using graphs, we are capable of encoding all metadata artifacts in a single construct. This yields significant advantages on simplicity, as the algorithms using this constructs need only to be concerned with a single formalism.

Representing metadata using graphs also brings benefits in terms of expressiveness (i.e., the ability to express different representations, regardless of their complexity), semantic relativism (i.e., the ability to accommodate different representations of the same data) [26,41], as well as conciseness (because of the possibility of reusing pieces of metadata information in different places by just using edges connecting from/to them). Finally, graphs offer great flexibility (i.e., the ability to easily add new metadata artifacts without modifying the underlying data model), as well as query performance (since queries over

[10] `https://spark.apache.org`
[11] `https://flink.apache.org`
[12] `https://hadoop.apache.org`
[13] `https://neo4j.com`
[14] `https://www.postgresql.org`

metadata are commonly dominated by traversals, i.e., joins with low cardinality, which are particularly efficient in graph databases).

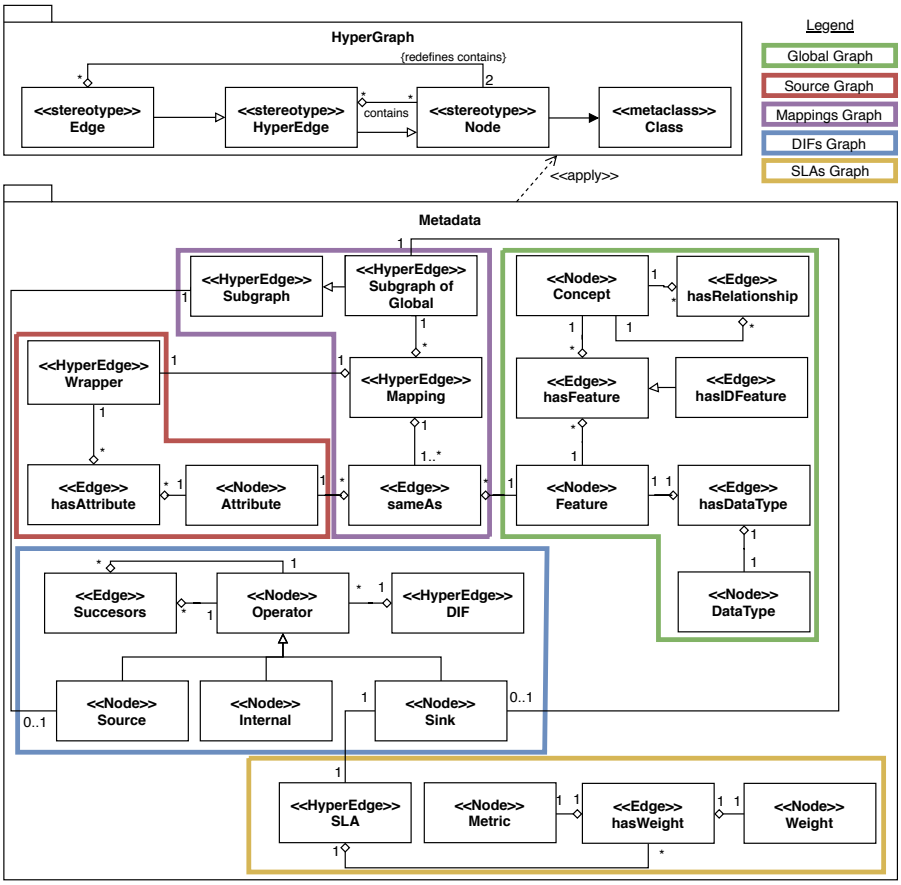


Fig. 4: UML class diagram for the hypergraph-based metadata artifacts

Figure 4, depicts a UML conceptual schema showing the relationships between the different artifacts discussed in Section 4.1. We distinguish nodes, hyperedges (i.e., nodes that define a generic node grouping) and edges (i.e., groups of only two nodes), in a generalized hypergraph structure. Next, we discuss the details of each artifact.

Global graph. The global graph $\mathcal{G}$ represents the domain of interest of analysts in terms of *concepts* (i.e., classes) and *features* (i.e., class attributes having a data type), which are linked with an edge labeled `hasFeature`. Then, an edge labeled `hasRelationship` connects concepts with each other, which can correspond to associations, specializations or aggregations. $\mathcal{G}$ is the unified in-

terface for analysts to pose queries $Q_{\mathcal{G}}$, which are represented as edge-patterns without variables [2].

Source graph. The source graph $\mathcal{S}$ represents the schema of wrappers and their attributes. We restrict wrappers to expose a flat structure of the sources or a tree structure without arrays, hence the source graph $\mathcal{S}$ contains nodes of type attribute, and others representing wrappers, that following the hypergraph model in [26], allow to encode inside the physical data structure (e.g., JSON document). Details of that encoding are omitted in the UML class diagram for simplicity.

Mappings graph. The mappings graph $\mathcal{M}$ represents LAV schema mappings linking $\mathcal{G}$ and $\mathcal{S}$. In the proposed hypergraph-based representation, LAV mappings are formalized as subgraphs of $\mathcal{G}$ (i.e., hyperedges) at the wrapper level. Note that, for the sake of simplicity, the associations between *Subgraph of Global* and elements in $\mathcal{G}$ have been suppressed in Figure 4. Then, for each attribute, $\mathcal{M}$ contains an edge labeled `sameAs`, which creates a connection between the local attribute and its corresponding feature in $\mathcal{G}$. This is particularly relevant in heterogeneous integration settings where attribute names (externally defined in the wrappers) might differ from feature names (defined in the global graph).

Data-intensive flows graph. A DIF is composed by a set of operators linked via `successor` edges. A DIF operators can either be: (1) *source nodes*, which, using the source-specific metadata (i.e., source graphs and wrappers) extract data from data sources, (2) *internal nodes*, which besides the schema also encode the operator's semantics, and (3) *sink nodes*, which output data to a target data store, corresponding to a set of global features, modeled to suit the end user needs. Thus, we encode the DAG $\mathcal{D}$ corresponding to all DIFs.

SLAs graph. The graph of SLAs $\mathcal{T}$ encodes the different kinds of objective functions in the form of a subgraph, too. These are further linked to DIF elements, in order to characterize what are the characteristics to optimize when executing DIFs or consolidating results. Given that a DIF can have several sink nodes, an SLA is represented as a hypernode associated to a sink, and composed of a set of metrics to optimize.

5.1.1 Use case: NTDs metadata

Here, we report on the metadata needed in the application of the NTD use case.

Integration graph of the NTDs use case. In the top part of Figure 5, we depict the global graph corresponding to the NTDs use case, specifically focusing on disease diagnosis (per time and country), the medicines used in their treatment, and when they have been distributed in the different countries, including the information about the countries population and migrants. The bottom depicts an example extract of a source graph, which for clarity of representation includes only the WIDP diagnosis data. Here, we can see that

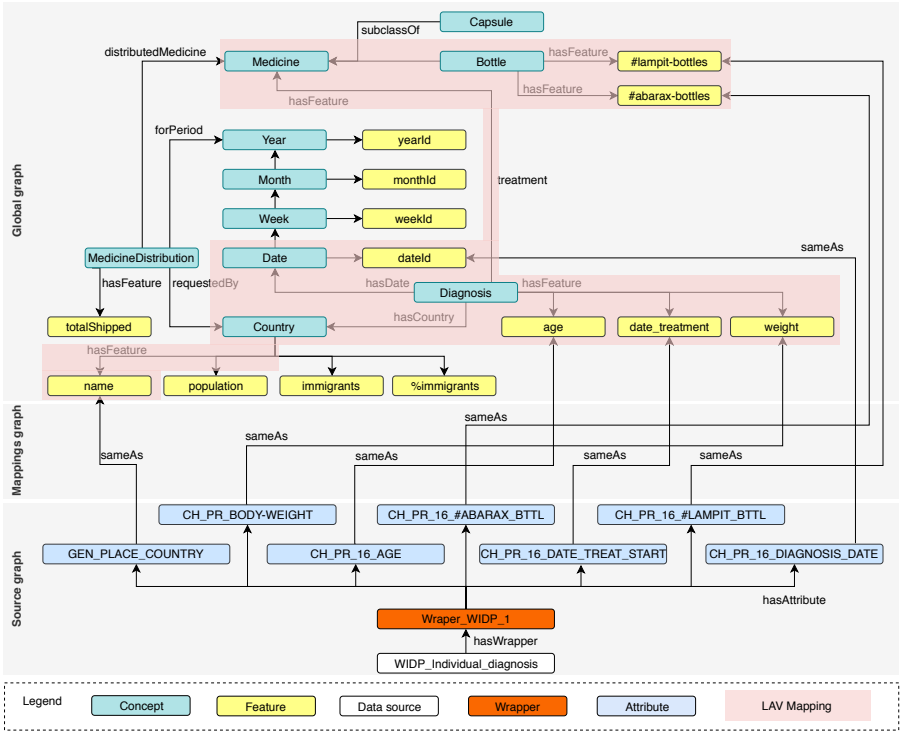


Fig. 5: Example of metadata representation including global graph, source graph for the diagnosis source, and corresponding mappings graph

the source exposed by the wrapper has a given set of attributes. Lastly, the middle part depicts the mappings for the WIDP diagnosis data, consisting of the `sameAs` edges connecting source graph attributes with the global graph.

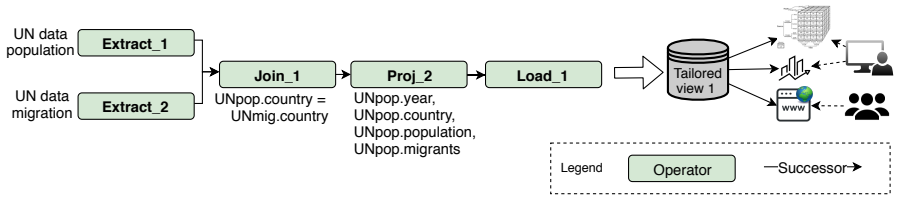


Fig. 6: DIF retrieving UN data

DIF graph to retrieve UN data. In Figure 6, we depict a DIF graph that answers the user query for jointly retrieving UN data (e.g., for evaluating the migration prevalence ratios in the country). We have two source nodes, namely `Extract_1` and `Extract_2`, using specific wrappers to read the data from the

population and immigration datasets of UN, respectively. These data are then joined and the needed fields are projected by the internal nodes, `Join_1` and `Proj_2`. Lastly, the DIF uses a sink node `Load_1` to store the integrated data into a materialized view tailored for the user needs.

SLA graph for querying UN data. Finally, for the DIF above, we define an SLA graph (see Figure 7) to express that the user favors execution time (i.e., efficiency of retrieving the results) over data freshness, which in the case of UN data is not critical, being that the population and migration data are updated annually.

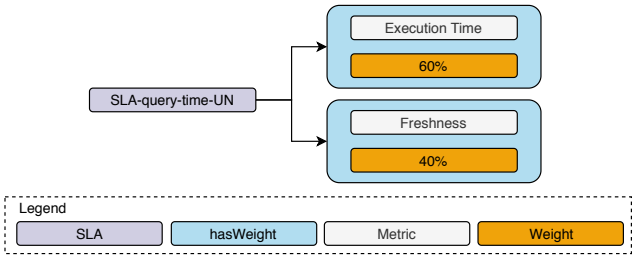


Fig. 7: SLA for querying UN data

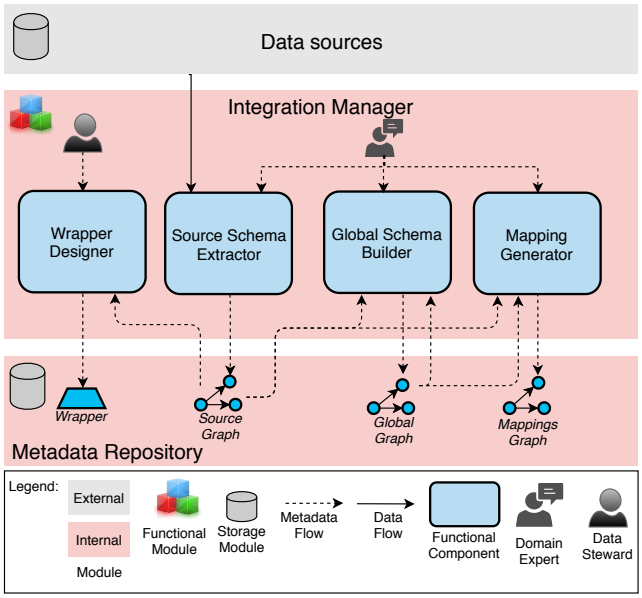


Fig. 8: Integration Manager

5.2 Integration Manager

This module is primarily in charge of supporting automatic data integration by generating source-specific metadata when *registering a new data source* in the platform (see Figure 2a). For data sources entering into the platform, many of which come without any kind of schema information, *Quarry*, through its *Integration Manager* (see Figure 8), provides support for incorporating them in further analytical pipelines.

Given a source data model (e.g., relational, key/value, documents) and its specific format (e.g., JSON, XML, CSV), the *Wrapper Designer* proposes wrappers that deal with variety in the sources and convert their data into a common unified model (see Section 5.1). Such wrappers allow the other components in the data integration platform to access the data in a unified manner [26]. Next, *Quarry* largely supports extraction of the wrapper's schema via the *Source Schema Extractor* component. This process can benefit from existing information (e.g., database catalog, XSD schema), but it primarily works over available data instances to incrementally extract schema information using model driven transformations [48]. Such source schema information is expressed in the source graph.

Quarry further supports an incremental, bottom-up construction of the global graph for the domain at hand, directly from data sources and the extracted source graphs. For such task, it provides the *Global Schema Builder* component which relies on semi-automatic schema alignment and merging process [50, 40], which, backed by domain experts produces a knowledge base with the vocabulary oriented to end users of the system. The alignments are found by means of applying state of the art techniques for ontology alignment, which besides lexical matching in terms of Jaccard index, also uses Wordnet as a synonym base. The candidate matchings are then ranked based on the confidence level, representing the degree of similarity between the concepts, and can be accepted or rejected by the user. *Global Schema Builder* as well offers an intuitive interface allowing manual definition of additional matchings. Lastly, in order to rewrite queries expressed over the global graph, *Quarry* requires the correspondences between data sources and the global graph. As previously mentioned *Quarry* uses mappings to relate each variable of the source graph retrieved by the wrapper to the global graph, based on LAV approach. Building of such mappings graph is done by the *Mapping Generator* and it naturally comes as a side-product of finding the alignments between the source and the global graph [50, 35]. That is, whenever a matching of the source concept with the global schema is found and accepted by the user, *Mapping Generator* first defines `sameAs` edges to all the mapped concepts in the global graph. Then, starting from the mapped concepts in the global graph, it defines the LAV mapping by discovering the possible integration paths among the global graph concept. Such process may require user intervention for resolving ambiguities (e.g., multiple paths between the concepts), and as well allows the user to manually record such LAV mappings in the metadata repository.

As a result, the *Integration Manager* module builds and maintains the integration graph (see example in Figure 5), which resolves possible semantic conflicts among a variety of data sources, thus enabling the automated data integration and query processing.

5.2.1 Use Case: Registering New Data Source

Here, we report on the application of the NTD use case over the Integration Manager module for registering a new source (see Figure 2a).


```
{"events":[
{"storedBy":"chagas.countryOfficer", "dueDate":"2018-11-09T15:39:27.202",
"program":"H_D_PR_HMO16_INDIVIDUAL-DIAGNOSIS","status":"COMPLETED",
"orgUnitName":"Kingdom of Spain", "eventDate":"2010-02-09T00:00:00.000",
"coordinate":{"latitude":0.0,"longitude":0.0},
"dataValues":[
{"CH_PR_16_#LAMPIT-BOTTLES":"3"},
{"CH_PR_16_LAMPIT-NIFURTIMOX":"Yes"},
{"CH_PR_16_#ABARAX-BOTTLES":"0"},
{"CH_PR_16_DATE-TREATMENT-START":"2010-01-01"},
{"CH_PR_16_MEDICINE-REQUEST-REFERENCE":"100010"},
{"CH_PR_16_NON-ANTIPARASITIC-TREATMENT":"Yes"},
{"CH_PR_16_DISEASE-PHASE":"CHRONIC"},
{"GEN_PLACE_Infection_OU_T":"cpmmPDsQ3uG"},
{"CH_PR_16_TRANSMISSION-ROUTE":"VECTORIAL"},
{"GEN_PLACE_Birth_OU_T":"Bolivia"},
{"CH_PR_16_BODY-WEIGHT":"74"},
{"CH_PR_16_AGE":"37"},
{"GEO_DE_ADDRESS":"Av.Drassanes, 17-21, 08001"},
{"GEN_PLACE_OU_T":"DUgfII2clbi"},
{"CH_PR_16_HEALTHCARE-CENTER-NAME":"Medicina Tropical y Salud Internacional
    Drassanes"}]},
...]}
```


Fig. 9: Individual diagnosis data from WIDP in original JSON format

Extract source specific metadata. An example of a source graph, extracted from WIDP's individual diagnosis dataset, is depicted in the bottom of Figure 5. The WIDP source graph is extracted by parsing the sample of input JSON files obtained from WIDP API. In addition, the *Wrapper Designer* component generates a source-specific wrapper (`Wrapper_WIDP1`) in order to convert the JSON format of WIDP (see Figure 9) into a common tabular format. Obviously, depending on the data source, the complexity of such conversion can vary and certain transformation may require user involvement.

Create/update global graph. Starting from the WIDP and WIMEDS source graphs, whose variables are depicted in Figure 1, *Global Schema Builder* searches for the alignments between the extracted source concepts and the global graph. Notice that while we start here with an existing (potentially incomplete) global graph for NTDs, *Global Schema Builder* also allows to start creating a global graph from scratch. In that case, a source graph of the first data source (e.g., WIDP) will directly constitute the initial global graph. In Figure 10, we respectively depict the resulting matchings for WIDP

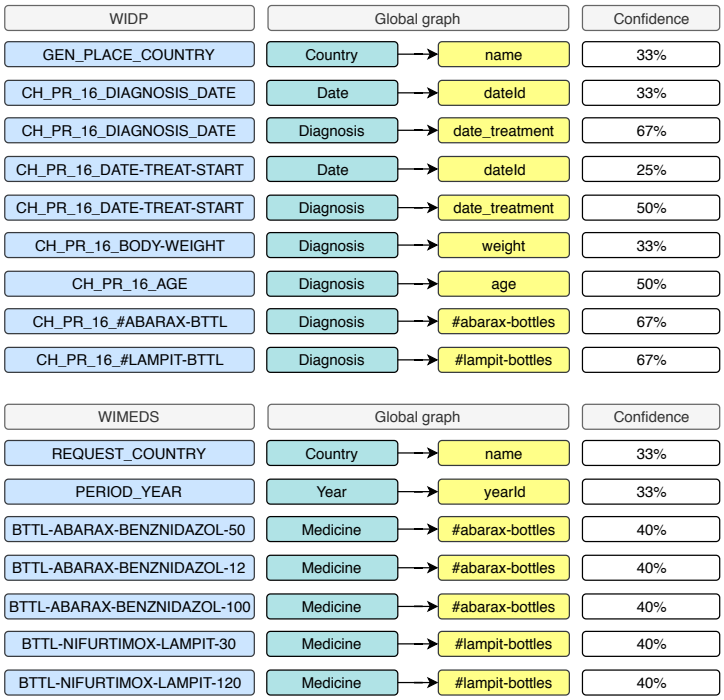


Fig. 10: Resulting alignments for WIDP and WIMEDS source graphs

and WIMEDS data sources, which are found by *Global Schema Builder*, together with the found confidence levels. Notice that although very efficient, the automatic alignment may not always yield the correct matching (e.g., see in Figure 10 that `CH_PR_16_DIAGNOSIS_DATE` matches `Diagnosis->date_treatment` with confidence of 67% while the correct one `CH_PR_16_DATE-TREAT-START` matches it with lower confidence of 50%). Nevertheless, *Global Schema Builder* prompts the user all the matched alternatives for the approval before it continues processing. Notice that *Global Schema Builder* could automatically find 16 different matchings for WIDP and WIMEDS data sources (see Figure 10), with different confidence levels, while the user had to intervene and reject 2 matchings (i.e., date mismatches: `CH_PR_16_DIAGNOSIS_DATE` $\neq$ `Diagnosis -> date_treatment` and `CH_PR_16_DATE-TREAT-START` $\neq$ `Date -> dateId`).

Add mappings. Finally, as a result of each matching between the WIDP source graph and the global NTD graph, either automatically found or manually defined by the user in *Global Schema Builder*, the *Mapping Generator* component further adds a `sameAs` edge to connect the WIDP source concept to the global graph, allowing then recording of the LAV mappings for those concepts (see Figure 5).

5.2.2 Evaluation

Following from the examples reported above, we can see that *Quarry* highly automates the processes essential to facilitate registering of a new data source in the data integration system, namely the extraction of the source metadata and creation of the global graph. A more detailed evaluation of the *Global Schema Builder* module, using the identical set of data sources as introduced in Section 3.1, is reported in [40], showing an average precision of 58.5% and 100% recall achieved when finding the alignments for the WIDP, WIMEDS, and the two UN data sources. Furthermore, while for 79% of all source concepts *Quarry* found the matching automatically, the user had to intervene and manually define four matchings in total and reject other eight, through an intuitive graphical interface. Indeed, as reported also in [40], user satisfaction with the usability of the provided graphical assistance for manually defining the metadata objects is considerably high. More than half of the users (56.3%) qualified the process as very easy, while the rest found it to be moderately hard. Regarding the individual tasks, only 12.5% had to invest more significant effort for extracting source metadata and 6.3% faced some difficulties when creating the global graph. Nevertheless, notice that such manual efforts are typically done only once when registering the data source, while the resulted metadata are later continuously reused for automating query processing in the rest of the platform.

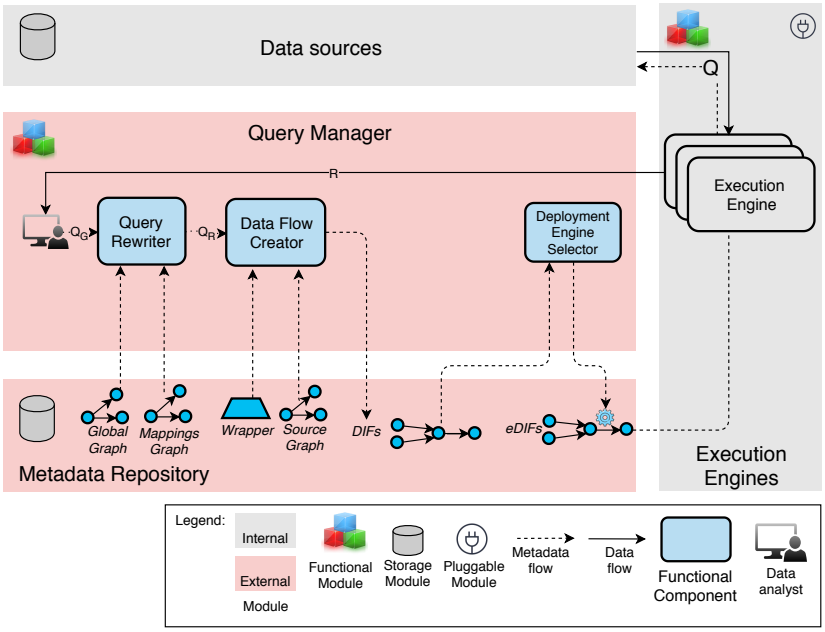


Fig. 11: Query Manager

5.3 Query Manager

Quarry's *Query Manager* module supports data analysts by providing an intuitive graphical interface with functionalities to visually guide the user into exploring the domain at hand and searching for particular insights. For that purpose, the *Query Manager* conveniently exploits previously created hypergraph-based metadata from the *Metadata Repository* for enabling end user's interaction with the system. Having a variety of external and evolving data sources at hand, we require an efficient query mechanism so that end users and domain experts can, on-demand, query and scrutinize the underlying data. This exploration allows a manual evaluation to decide their usefulness and relevance for the analytical tasks, and once the analysis of interest is clear, decide if and how to materialize the results of such query considering the SLAs (see Figure 2b).

Query Manager supports the data exploration workflow via query answering techniques over the global schema. Hence, the task here consists of dealing with the automatic translation (i.e., rewriting) of a query over the *global schema* into queries over the data sources. To this end, the *Query Rewriter* component takes as input a query $Q_{\mathcal{G}}$ posed by the data analyst over the global graph. Recall that *Quarry* adopts LAV mappings, which are well known to result in a complex reasoning task for query rewritting. Yet, given that *Quarry* works under Closed World Assumption (CWA) dismissing the need to deal with incompleteness on the sources, we guarantee that the problem is tractable in practice [36]. Intuitively, the *Query Rewriter* component identifies integration paths between the queried concepts (i.e., paths that connect two queried concepts of the global graph through the mappings and source graphs concepts), and outputs a set of rewritten queries Q_R (i.e., a union of conjunctive queries). For each rewritten query, the *Data Flow Creator* component generates a DIF being able to answer the user query and retrieve the needed data to the *Data Analyst*.

http://who.int/name	http://who.int/population	http://who.int/numberimmigrants
Afganistan	36370000	87119
Albania	2930000	113
Algeria	42010000	94248
American Samoa	60000	null
Andorra	80000	null
Angola	30770000	48232
Anguilla	20000	null
Antigua and Barbuda	100000	1
Argentina	44690000	3322
Armenia	2930000	17922
Aruba	110000	1

Fig. 12: Excerpt of the `Query 1` results

Notice that the *Query Manager* can return the resulting data in a tabular form, like that in Figure 12, which is suitable for quick visualization during the exploration phase, as well as for importing into an external data analysis and visualization tool for further exploitation. However, *Quarry* also provides means to adapt the target data model suitably to the type of the analysis set by the user, and in case that for example the user chooses to perform OLAP analysis, the DIF could output data in a multidimensional model (e.g., star schema) suitable for loading into an OLAP engine [28].

Lastly, through the *Deployment Engine Selector* component, the *Data Steward* assists the *Data Analyst* in choosing the optimal execution engine and preparing logical DIFs for execution in a selected engine and returning the results. Part of such process is automated using engine-specific translations to produce executable DIFs (i.e., eDIFs) [29].

Notice that *Query Manager* stays at the schema level when it comes to integration, meaning that it does not resolve possible value level conflicts or other data quality issues. Nevertheless, *Quarry* allows users to further extend the resulting DIFs using its *Data Quality Manager* module (Section 5.4), and address such problems.

5.3.1 Use Case: Data Exploration

Here, we report on the application of the NTD use case over the Query Manager module for data exploration (see Figure 2b).

Query data. In this use case, *Query Manager* adopts the existing WebVOWL tool[15], which, consuming the previously created global graph from the *Metadata Repository*, visualizes the NTD domain and enables end users to navigate and pose queries. In particular, the *Data Analyst* may first explore the NTD domain, and zoom in to particular part of the domain of interest (e.g., *Diagnosis* data). In addition, and especially in the case of large graphs which are expected in many domains, the user can also do a keyword search provided by the WebVOWL tool to easily refocus the view to the part of the graph of interest and start exploring from there. Using the same interface, the user can then select the concepts he/she is interested in and start querying.

`Query 1:` A user may be interested to first integrate and explore the two datasets of UN about countries' population and migration to those countries, in order to analyze which portion of the population in a country is made by migrants. This is an important information in the analysis of NTDs, as for many of them (e.g., Chagas) prevalence depends on countries. Given that both sources have LAV mapping to the `Country` concept of the global graph, the *Query Manager* integrates these sources on the `Country->name` feature and jointly retrieves for each country the population and migration counts. The excerpt of the results of `Query` 1 is presented in Figure 12. Notice that source data may not always come with a good quality (e.g., see *null* values).

[15] WebVOWL:Web-based Visualization of Ontologies - `http://vowl.visualdataweb.org/webvowl.html`

Nevertheless, *Quarry* through its *Data Quality Manager* module (see Section 5.4) allows users to address such issues.

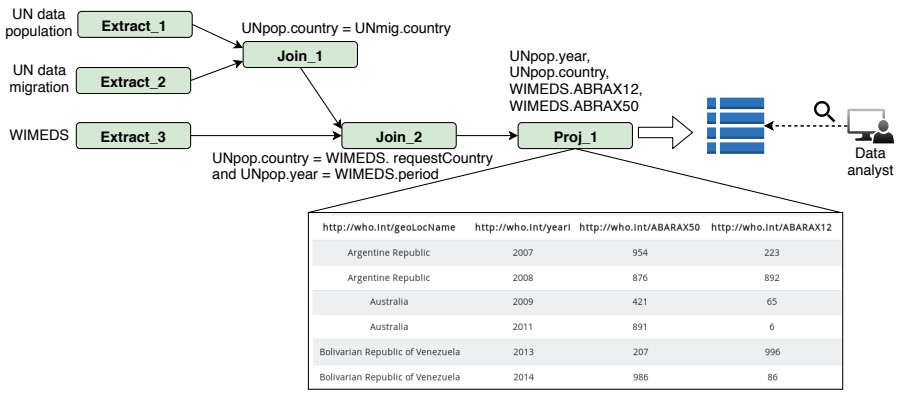


Fig. 13: DIF to answer *Query 2*

`Query 2:` Continuing the exploration, the same or another *Data Analyst* may now be interested in crossing these demographic data with medicine distribution data extracted from WIMEDS (see Figure 1). When searching for integration paths among these concepts (i.e., `MedicineDistribution` and `Country`), *Query Rewriter*, using previously defined LAV mappings, encounters that the integration (i.e., join) can be done over `Country->name` and `Year->yearID` features (i.e., the two data sources should be matched on time and space dimensions). In Figure 13, we depict the DIF generated entirely automatically to answer *Query* 2.

5.3.2 Evaluation

Following from the examples above, we can observe that once having the metadata objects prepared, *Quarry* can exploit the Global and Source graphs together with their corresponding mappings, and created data source wrappers, to automatically resolve user queries and generate DIFs that return analysis-ready data. Although it follows the LAV mappings approach, *Query Manager* stays tractable in integrating data when answering user queries, due to the CWA consideration discussed above. In fact, as [36] reports, the cost of the algorithm behind the *Query Rewriter* component derives from the complexity of wrapper mechanism and yields exponential growth with the number of wrappers. However, for realistic scenarios with tens of wrappers it stays within tractable bounds.

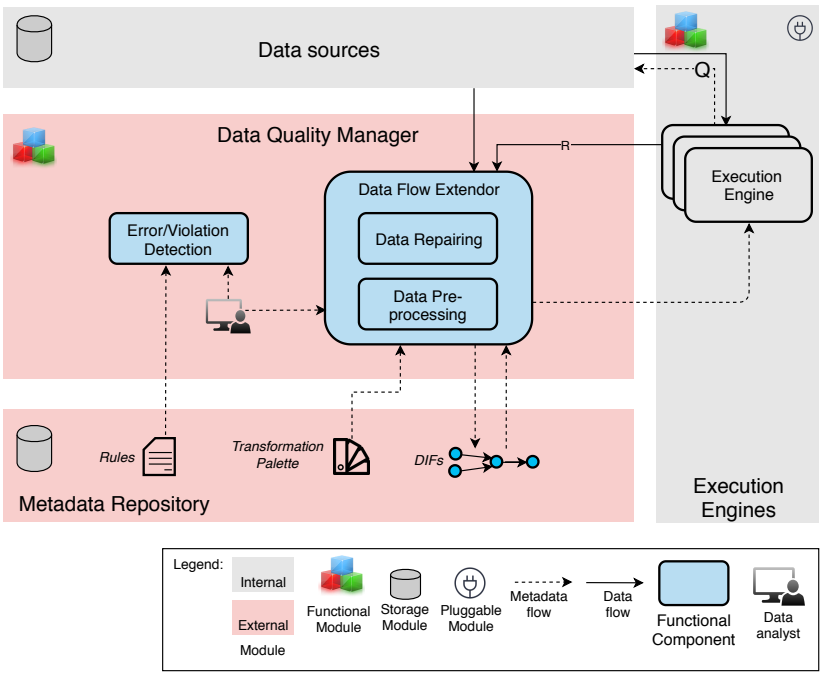


Fig. 14: Data Quality Manager

5.4 Data Quality Manager

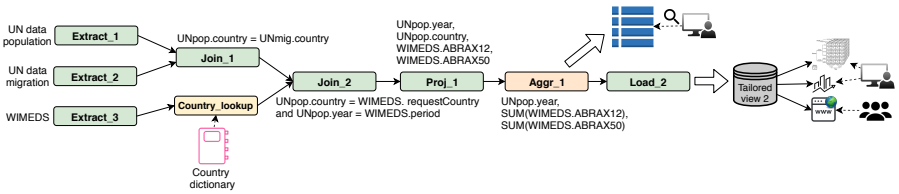
Given that user queries in *Quarry* result in DIFs that automatically extract data from different data sources and integrate them into a unified view, additional (instance-based) transformations may need to be added to the initial DIF to support a variety of data processing tasks; some related to improving the overall quality of resulting data and thus enabling the analysis (e.g., deduplication, inconsistency detection), and others related to data preparation, specifically tailored to user analysis facets (e.g., discretization, outlier detection). In addition, especially during the data exploration phase, the *Data Analyst* may as well choose to sample the resulting data and obtain a briefer overview of the data before choosing the actual analytical facet.

In the *Data Quality Manager* module, the *Data Analyst* may simply select such transformations from the existing palette, or whenever applicable he/she may get assistance, in which case the recommended transformations are ranked according to their relevance for the problem at hand.

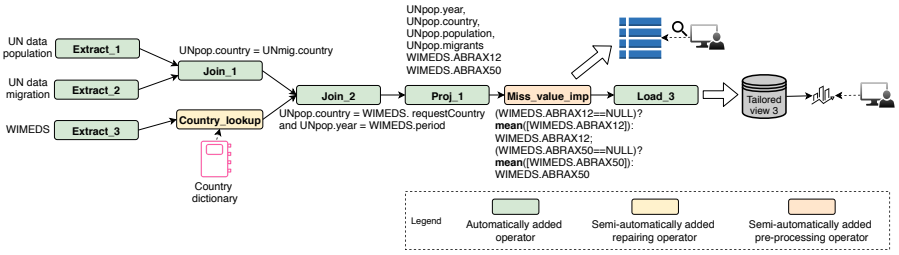
For instance, in the *Data Repairing* component, given some rules and a master data to refer to, the *Data Analyst* may receive support through recommendations which suggest him/her to complement the missing data, remove the data that has been identified as duplicate, or given some integrity con-

straints he/she may be asked how to handle the detected inconsistent rows. Note that this is agnostic to the type of analysis to be performed.

Finally, once errors are detected and repaired, and the data is ready to be analyzed, in the *Data Pre-processing* component the *Data Analyst* receives recommendations of complex transformations that intend to positively impact the final analysis. To this end, *Quarry* takes into account the type of analysis to be performed and uses a meta-learning approach [6] to recommend transformations that are ranked according to their impact on the final analysis. Hence, transformations that are expected to improve the analysis (e.g., increase the predictive accuracy of a classification algorithm) are ranked higher in the list.



(a) Extension for descriptive analysis



(b) Extension for predictive analysis

Fig. 15: Extended DIF for *Query* 2

5.4.1 Use Case: Data Flow Extension

Here, we report on the application of the NTD use case over the Data Quality Manager module for data flow extension (see Figure 2c).

Extend data flow. Given that the underlying data for answering `Query 2` are coming from two independent data sources (i.e., UN data and WIMEDS), the country names, which are on one of the proposed integration paths, may differ (e.g., *Kingdom of Spain* as official UN name and *Spain* as a standard short name, or similarly, *Bolivarian Republic of Venezuela* and *Venezuela*). To resolve these mismatches, the analyst may add some data repairing transformation from the palette (e.g., a dictionary lookup for unifying country names - `Country_Lookup` in Figure 15). In addition, analysts may as well decide to push some of the transformations from their analytics flow to the DIF. For example,

a user may be interested in performing descriptive (OLAP) analysis, and group resulting data per year, aggregating the medicine quantities to report on the total quantity of shipped medicines in particular years, irrespectively of the country. Note that such aggregation addition may be done after producing the initial flow like here or even from the very beginning by expressing this need in the user information requirement. The resulting, extended DIF for *Query 2* is depicted in Figure 15a. The same or another user may be interested in making predictive analysis on the quantity of specific medicines needed for a country in the following years. Based on such a requirement, *Quarry* may suggest to the user to extend the flow with a data pre-processing transformation that could positively impact the final analysis (e.g., `Miss_value_imp`, using the attribute mean to complete the dataset; see Figure 15b). Notice that *Quarry* may as well propose other data pre-processing transformations (e.g., *discretization*, *outlier detection*) specific to a final data analysis [6].

5.4.2 Evaluation

We can observe from our use case above that *Quarry* provides flexibility to end users to fine tune the resulting DIFs and extend them for improving the data quality. While on the one hand, *Data Quality Manager* provides a predefined palette of possible data reparation transformations to be manually added to the flow, *Quarry* can also assist users to more effectively identify the preprocessing operators appropriate to their analytical applications, and improve the predictive power of the analytical algorithms. Specifically, based on more detailed evaluation of the meta-learning algorithm behind the *Data Quality Manager* module reported in [6,7], we were able to observe that: (a) even if a user randomly picks a transformation from the entire list of transformations ranked by *Data Quality Manager*, our meta-learning algorithm results in a good transformation with an average accuracy of 61%, (b) recommending only the top-1 transformation, increased the accuracy to 68%, (c) measuring the gain obtained from our ranking for all transformations using Discounted Cumulative Gain (DCG), we were as close as 73% on average to the gain obtained from the best possible ranking of transformations, (d) measuring the gain from the top-1 recommendations using DCG, we were as close as 79% on average to the gain obtained from the best possible ranking , and (e) in a set of randomly selected classification problems, *Quarry* performed 2.5 times better than humans.

5.5 Flow Manager

Having DIFs resulting from several analysts' queries, previously processed by the *Query Manager*, *Quarry* provides the *Flow Manager* module, in charge of organizing the deployment of such DIFs and their (periodical) execution over the available engines (see Figure 16).

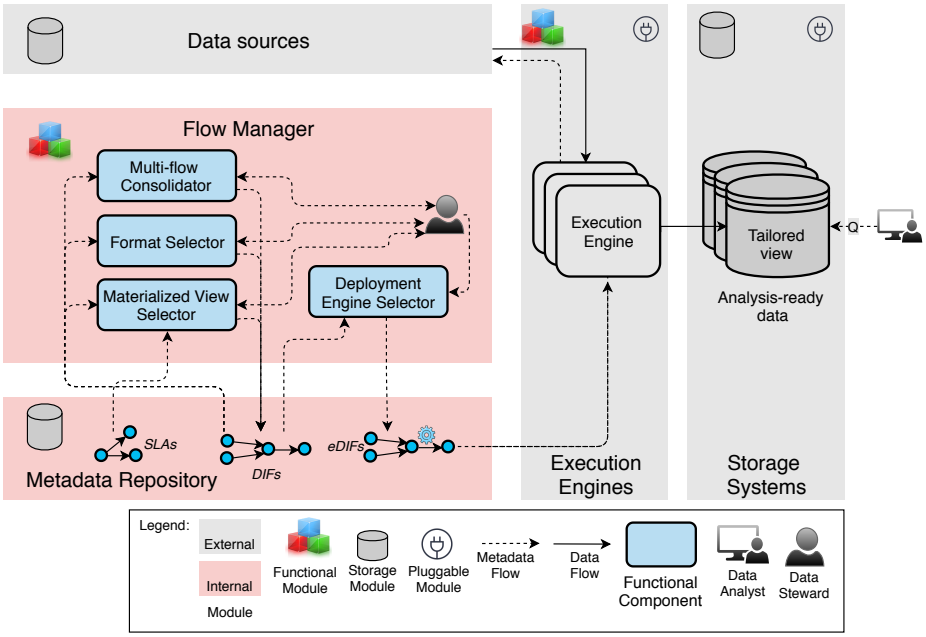


Fig. 16: Flow Manager

In particular, *Flow Manager*, through the *Multi-flow Consolidator* component, searches for possible shared data processing operators among DIFs coming from different analysts in the system. To maximize the potential overlapping among the flows, it applies various flow transformations (e.g., operation reordering, merging or splitting) [27]. Once the overlapping data processing parts are identified, the DIFs are consolidated and a multi-flow execution is proposed for them. *Multi-flow Consolidator* is generic in terms of the cost model that is used for creating a multi-flow. For example, we can maximize resource usage by finding maximal overlaps, but this may result in penalizations for some DIFs, because their filtering operations with high selectivity may be pulled towards the end of the DIF. Furthermore, to address other quality factors, typically described as SLAs (e.g., *data freshness*, *storage space*, *response time*), some of which may be conflicting, the *Flow Manager* employs a component called *Materialized View Selector*. Given a specific SLA, *Materialized View Selector* finds the optimal materialization of intermediate results [34]. Intuitively, for input DIFs, *Materialized View Selector* decides where to "cut" the flow, i.e., for which part of the flow to materialize the data before hand and which part to leave for on-demand querying.

Lastly, once the logical data flow is built and the data materialization level is selected, the *Flow Manager* prepares the flow for execution by focusing on the physical characteristics. Namely, *Flow Manager* provides two components dealing with physical, engine-specific, characteristics of DIFs, i.e., *Format Selector* and *Deployment Engine Selector*. For the data selected for material-

ization, the *Format Selector* component selects the optimal layout in which the data will be stored on the disk, which can significantly reduce the I/O costs [34], while similarly to the *Query Manager* module, through *Deployment Engine Selector*, the *Data Steward* assists in choosing the optimal execution engine for executing the DIFs. As a result, a set of eDIFs are prepared, by means of data flow translations [29], to be sent to a selected execution engine, finally resulting in data prepared for exploitation (i.e., analysis-ready data).

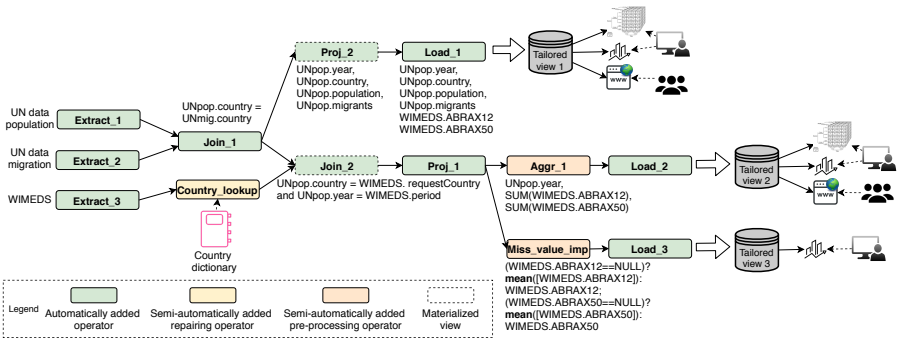


Fig. 17: Multi-flow DIF answering both *Query* 1 and *Query* 2

5.5.1 Use Case: Data Flow Deployment

Here, we report on the application of the NTD use case over the Flow Manager module for deploying data flows (see Figure 2a).

Consolidate data flow. Once DIFs to answer user queries are defined by *Query Manager* (see Figures 6 and 13), and extended by *Data Quality Manager* (see Figure 15), a user can decide to deploy the DIFs for future view materialization. *Flow Manager* then first searches for consolidation of the flows to propose their multi-flow execution. An example of such multi-flow for `Query 1` and `Query 2` in our NTD use case is depicted in Figure 17. Notice that part of the multi-flow that integrates two datasets of the UN data source is common for answering both queries, thus its results can be reused. Furthermore, both DIFs extended by *Data Quality Manager* from the DIF that answers `Query 2` (see Figures 15a and 15b) reuse the same base data flow to integrate UN data and WIMEDS. The iterative search for consolidating DIFs would then go from the common data sources (i.e., extraction operations `Extract_1` and `Extract_2`) following the overlapping operations and stop when no further overlap between DIFs can be found, in which case the multi-flow would create a split (i.e., see operations `Join_1` and `Proj_1` in Figure 17). Now, let us consider a slight modification of `Query 1`, i.e., `Query 1′`, where user is interested in analyzing migration from UN data, but only for a subset of European countries. In such case, the DIF for answering `Query 1′` would in addition require

filtering operator to retrieve only European countries' population and then join them with migration data (see Figure 18). Consequently, we would not be able to find the straightforward overlap as in the previous case, having that DIF for `Query 2` does not require such country filtering. Nonetheless, *Multi-flow Consolidator* would still proceed to find the maximal possible overlap. In particular, it would first apply DIF transformation and reorder the filter after `Join_1`, such that the DIFs could then be matched as in the previous case, creating the split after `Join_1`.

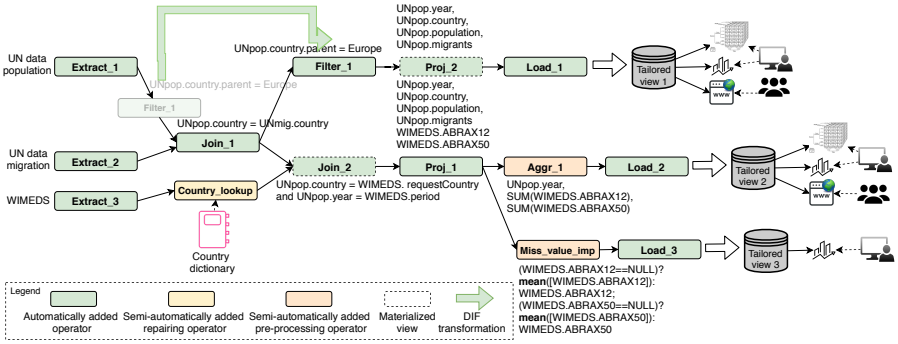


Fig. 18: Multi-flow DIF answering *Query* 1′ and *Query* 2

Optimize data flow. Next, given the frequency of the reports (e.g., weekly, monthly or yearly) and the need for timely answering such reports (represented as SLAs; see Section 5.1), *Flow Manager* decides to materialize the results of some of the operations of the multi-flow (see dashed-lined rectangles in Figure 17). In particular, given that for `Query 1` a user is interested in improving the query cost while the freshness is not critical, and having that UN only yearly updates such information, the *Materialized View Selector* component chooses to materialize the result of `Proj_2` operator. However, in addition to query cost, SLA for `Query 2` also requires improving the freshness of the results, and thus, the costly `Join_2` operator is chosen for materialization, while setting up the frequency of flow execution to *weekly*, in order to timely bring potentially new WIMEDS data. Lastly, the *Format Selector* component chooses to store such materializations following the optimal storage layout in *HDFS*, in this case *Parquet* format.

Select execution engine. At the end, to execute such flows, we need to select an optimal execution engine(s) using the *Deployment Engine Selector* component. In this case, *Apache Spark* was selected, in order to provide fast, in-memory, distributed data processing. Thus, the resulting multi-flow DIF needs to be translated into an executable format suitable for the selected engine (i.e., the *Java* version of Apache Spark). Alternatively, we could as well choose *Scala* for executing the same DIF.

5.5.2 Evaluation

Observing the examples above, we can see how *Quarry* closes the big data integration lifecycle by generating and deploying data flows to efficiently answer user queries and prepare the data for further exploitation. In addition, the algorithm for consolidating DIFs has been evaluated in terms of the number of queries that a multi-flow needs to answer [27]. Given the limited number of alternative DIF transformations, having that they all must produce equivalent results for all DIFs, *Multi-flow Consolidator* yields linear execution times depending mainly on the complexity of the input DIFs. At the same time, performance gains of such multi-flow execution are estimated to be 17.8-31.9%, advocating strongly for the benefits of such deployment of DIFs. *Materialized View Selector* also estimates the materialization decision to be tractable in practice, and besides the size of a DIF, also depends on the number of involved metrics and how conflictive they are [34] (e.g., freshness vs. query time).

6 Conclusions and Future Work

In this paper, we have presented *Quarry*, a Big Data integration platform for managing data integration and preparation tasks, offering at the same time a user-friendly query interface for expressing analytical needs. Indeed, *Quarry* advocates a comprehensive architectural view and proposes the four core functional modules for facilitating different steps in the big data integration lifecycle, including: (1) *registering* a new data source and making it available for the end user, (2) *exploring* the available data and selecting the analytical facet of interest, (3) *deploying* data flows to efficiently integrate and prepare data for further analysis, and (4) *extending* such data flows to address potential data quality issues or enable user-preferred analytical tasks. Moreover, we also discuss the potential for automating each of the proposed modules by means of exploiting metadata artifacts, for which we propose a compact hypergraph-based metadata model encompassing the entire platform.

We validate *Quarry* over a real world project, covering the domain of fighting Neglected Tropical Diseases at the World Health Organization. We thus apply the core functional modules of *Quarry* to facilitate different steps of the data integration lifecycle, and observe the high level of automation that the state of the art approaches can provide for them, while also pinpointing the needed user involvement along the way.

Our final goal is to provide an end-to-end user-centered platform for managing the complete data integration lifecycle in the context of complex Big Data settings, specifically (but not only) focusing on the variety of data coming from numerous external data sources. To this end, our future directions are primarily focused on: (1) providing higher automation of the flow deployment process by facilitating dynamic flow reparation mechanisms [52] and engine selection and optimization for the resulting multi-flow [44, 29], (2) enabling

users to effectively incorporate parts of their data analysis pipeline into the platform for boosting data preparation specific to their analytical needs [7], and (3) performing a unified end-to-end evaluation of the platform, both for scrutinizing the core functional modules and for assessing *Quarry*'s usability for non-technical users (e.g., statisticians or epidemiologists).

Acknowledgements We thank Dr. Lise Grout and Dr. Pedro Albajar-Viñas from the Neglected Tropical Diseases (NTD) department at WHO, for providing the use case. This work is partially supported by GENESIS project, funded by the Spanish Ministerio de Ciencia, Innovación y Universidades under project TIN2016-79269-R.

References

1. Abiteboul, S., André, B., Kaplan, D.: Managing your digital life. Commun. ACM **58**(5), 32–35 (2015)
2. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**(5), 68:1–68:40 (2017)
3. Angles, R., Gutiérrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1), 1:1–1:39 (2008)
4. Bean, R.: Variety, not volume, is driving big data initiatives (2016). URL `https://sloanreview.mit.edu/article/variety-not-volume-is-driving-big-data-initiatives`
5. Bilalli, B., Abelló, A., Aluja-Banet, T., Munir, R.F., Wrembel, R.: PRESISTANT: data pre-processing assistant. In: CAiSE Forum, pp. 57–65 (2018)
6. Bilalli, B., Abelló, A., Aluja-Banet, T., Wrembel, R.: Intelligent assistance for data pre-processing. Computer Standards & Interfaces **57**, 101–109 (2018)
7. Bilalli, B., Abelló, A., Aluja-Banet, T., Wrembel, R.: PRESISTANT: Learning based assistant for data pre-processing. Data & Knowledge Engineering **123**, 100–122 (2019)
8. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web **8**(3), 471–487 (2017)
9. Ceravolo, P., Azzini, A., Angelini, M., Catarci, T., Cudré-Mauroux, P., Damiani, E., Mazak, A., van Keulen, M., Jarrar, M., Santucci, G., Sattler, K., Scannapieco, M., Wimmer, M., Wrembel, R., Zaraket, F.A.: Big data semantics. J. Data Semantics **7**(2), 65–85 (2018)
10. Chen, Y., Alspaugh, S., Katz, R.H.: Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. PVLDB **5**(12), 1802–1813 (2012)
11. Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I.F., Ouzzani, M., Tang, N.: Nadeef: A commodity data cleaning system. In: SIGMOD, pp. 541–552 (2013)
12. Deng, D., Fernandez, R.C., Abedjan, Z., Wang, S., Stonebraker, M., Elmagarmid, A.K., Ilyas, I.F., Madden, S., Ouzzani, M., Tang, N.: The data civilizer system. In: CIDR (2017)
13. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann (2012)
14. Duggan, J., Elmore, A.J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T., Zdonik, S.B.: The BigDAWG Polystore System. SIGMOD Record **44**(2), 11–16 (2015)
15. Fernandez, R.C., Madden, S.: Termite: a system for tunneling through heterogeneous data. In: aiDM@SIGMOD, pp. 7:1–7:8 (2019)
16. Fletcher, G.H.L., Mandreoli, F.: No users no dataspaces! query-driven dataspace orchestration? In: SEBD, pp. 150–157 (2016)
17. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record **34**(4), 27–33 (2005)

18. Friedman, M., Levy, A.Y., Millstein, T.D.: Navigational plans for data integration. In: IJCAI (1999)
19. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.D., Vassalos, V., Widom, J.: The TSIMMIS approach to mediation: Data models and languages. J. Intell. Inf. Syst. **8**(2), 117–132 (1997)
20. Golshan, B., Halevy, A.Y., Mihaila, G.A., Tan, W.: Data integration: After the teenage years. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017, pp. 101–106 (2017)
21. Gorawski, M., Lorek, M.: Efficient storage, retrieval and analysis of poker hands: An adaptive data framework. Applied Mathematics and Computer Science **27**(4), 713–726 (2017)
22. Gorton, I., Klein, J.: Distribution, data, deployment: Software architecture convergence in big data systems. IEEE Software **32**(3), 78–85 (2015)
23. Hai, R., Geisler, S., Quix, C.: Constance: An intelligent data lake system. In: SIGMOD, pp. 2097–2100 (2016)
24. Halevy, A.Y., Korn, F., Noy, N.F., Olston, C., Polyzotis, N., Roy, S., Whang, S.E.: Managing google's data lake: an overview of the goods system. IEEE Data Eng. Bull. **39**(3), 5–14 (2016)
25. Halevy, A.Y., Rajaraman, A., Ordille, J.J.: Data integration: The teenage years. In: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006, pp. 9–16 (2006)
26. Hewasinghage, M., Varga, J., Abelló, A., Zimányi, E.: Managing polyglot systems metadata with hypergraphs. In: ER, pp. 463–478 (2018)
27. Jovanovic, P., Romero, O., Simitsis, A., Abelló, A.: Incremental consolidation of data-intensive multi-flows. IEEE Trans. Knowl. Data Eng. **28**(5), 1203–1216 (2016)
28. Jovanovic, P., Romero, O., Simitsis, A., Abelló, A., Mayorova, D.: A requirement-driven approach to the design and evolution of data warehouses. Inf. Syst. **44**, 94–119 (2014)
29. Jovanovic, P., Simitsis, A., Wilkinson, K.: Engine independence for logical analytic flows. In: ICDE, pp. 1060–1071 (2014)
30. Konstantinou, N., Koehler, M., Abel, E., Civili, C., Neumayr, B., Sallinger, E., Fernandes, A.A.A., Gottlob, G., Keane, J.A., Libkin, L., Paton, N.W.: The VADA architecture for cost-effective data wrangling. In: SIGMOD, pp. 1599–1602 (2017)
31. Lenzerini, M.: Data integration: A theoretical perspective. In: PODS, pp. 233–246 (2002)
32. Lerman, K., Minton, S., Knoblock, C.A.: Wrapper maintenance: A machine learning approach. J. Artif. Intell. Res. **18**, 149–181 (2003)
33. Luján-Mora, S., Trujillo, J.: Applying the UML and the unified process to the design of data warehouses. JCIS **46**(5), 30–58 (2006)
34. Munir, R.F., Nadal, S., Romero, O., Abelló, A., Jovanovic, P., Thiele, M., Lehner, W.: Intermediate results materialization selection and format for data-intensive flows. Fundam. Inform. **163**(2), 111–138 (2018)
35. Nadal, S., Rabbani, K., Romero, O., Tadesse, S.: ODIN: A dataspace management system. In: ISWC, pp. 185–188 (2019)
36. Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., Vansummeren, S.: An integration-oriented ontology to govern evolution in big data ecosystems. Inf. Syst. **79**, 3–19 (2019)
37. Popovic, A., Hackney, R., Tassabehji, R., Castelli, M.: The impact of big data analytics on firms' high value business performance. Information Systems Frontiers **20**(2), 209–222 (2018)
38. Priyatna, F., Corcho, Ó., Sequeda, J.F.: Formalisation and experiences of r2rml-based SPARQL to SQL query translation using morph. In: WWW, pp. 479–490 (2014)
39. Quix, C., Hai, R.: Data lake. In: Encyclopedia of Big Data Technologies. (2019)
40. Rabbani, K.: Supporting the Semi-Automatic Creation of the Target Schema in Data Integration Systems. Master's thesis, Technische Univesitat Berlin - Universitat Politècnica de Catalunya, BarcelonaTech (2019)
41. Saltor, F., Castellanos, M., García-Solaco, M.: Suitability of data models as canonical models for federated databases. SIGMOD Record **20**(4), 44–48 (1991)

42. Sarma, A.D., Dong, X.L., Halevy, A.Y.: Uncertainty in data integration and dataspace support platforms. In: Schema Matching and Mapping, pp. 75–108 (2011)
43. Simitsis, A., Vassiliadis, P., Sellis, T.K.: State-space optimization of ETL workflows. IEEE Trans. Knowl. Data Eng. **17**(10), 1404–1419 (2005)
44. Simitsis, A., Wilkinson, K., Dayal, U., Hsu, M.: HFMS: managing the lifecycle and complexity of hybrid analytic data flows. In: ICDE, pp. 1174–1185 (2013)
45. Skoutas, D., Simitsis, A.: 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)
46. Stonebraker, M.: The Case for Polystores – ACM SIGMOD Blog (2019). [Online; accessed 27. Jun. 2019]
47. Stonebraker, M., Bruckner, D., Ilyas, I.F., Beskales, G., Cherniack, M., Zdonik, S.B., Pagan, A., Xu, S.: Data Curation at Scale: The Data Tamer System. In: CIDR (2013)
48. Tadesse, S., Gómez, C., Romero, O., Hose, K., Rabbani, K.: ARDI: Automatic Generation of RDFS Models from Heterogeneous Data Sources. In: EDOC (2019)
49. Terrizzano, I.G., Schwarz, P.M., Roth, M., Colino, J.E.: Data wrangling: The challenging yourney from the wild to the lake. In: CIDR (2015)
50. Touma, R., Romero, O., Jovanovic, P.: Supporting data integration tasks with semi-automatic ontology construction. In: DOLAP, pp. 89–98 (2015)
51. Varga, J., Romero, O., Pedersen, T.B., Thomsen, C.: Towards next generation BI systems: The analytical metadata challenge. In: DaWaK, pp. 89–101 (2014)
52. Wojciechowski, A.: ETL workflow reparation by means of case-based reasoning. Information Systems Frontiers **20**(1), 21–43 (2018)