A Conceptual Approach for Supporting Traffic Data Wrangling Tasks

DOI:
10.1093/comjnl/bxy113

Document Version
Accepted author manuscript

A Conceptual Approach for Supporting Traffic Data Wrangling Tasks

Sandra Sampaio[1*], Mashael Aljubairah[1], Hapsoro Adi Permana[1] and Pedro Sampaio [21]

[1]*Information Management Group,School of Computer Science, University of Manchester, Manchester M13 9PL, UK*
[2]*Alliance Manchester Business School, University of Manchester, Manchester M1 3WE, UK*
Email: [*]Corresponding author: sandra.sampaio@manchester.ac.uk

Data Wrangling (DW) is the subject of growing interest given its potential to improve data quality. DW applies interactive and iterative data profiling, cleaning, transformation, integration and visualization operations to improve the quality of data. Several domain independent DW tools have been developed to tackle data quality issues across domains. Using generic data wrangling tools requires a time-consuming and costly DW process often involving advanced IT knowledge beyond the skills set of traffic analysts. In this paper, we propose a conceptual approach to data wrangling for traffic data by creating a domain-specific language for specifying traffic data wrangling tasks and an abstract set of wrangling operators that serve as the target conceptual construct for mapping domain-specific wrangling tasks. The conceptual approach discussed in this paper is tool-independent and platform agnostic and can be mapped into specific implementations of DW functions available in existing scripting languages and tools such as R, Python, Trifacta. Our aim is to enable a typical traffic analyst without expert Data Science knowledge to be able to perform basic DW tasks relevant to his domain.

Keywords: Data Wrangling, Data Transformation and Quality, Conceptual Wrangling Approaches

Received 30 September 2017; revised 30 September 2018

1. INTRODUCTION

Decision makers in different domains such as healthcare, education and transportation, can gain significant advantages from the enormous volume of available data obtained from various data collection methods such as site-based sensors, cell-phone tracking [1] and social media. However, data collected via these methods are prone to data quality problems, such as inaccuracy, incompleteness and heterogeneity [2] [3]. More recently, data management techniques for data profiling, cleaning and integration have been adapted to improve the quality of large amounts of raw data, in preparation for analysis. The combination of these data management tasks is often called Data Wrangling (DW), generally defined as "the process by which the data required by an application is identified, extracted, cleaned and integrated, to yield a data set that is suitable for exploration and analysis" [4]. According to IBM [5], data analysts spend around 70% of their time conducting DW activities. Being an interactive and iterative process that involves the application of a variety of data management methods, and that generally lacks a rigid methodology across application domains, DW is often regarded as a highly complex job requiring advanced skills and domain expertise.

Data analysts typically perform DW tasks by using one or a combination of the two following approaches: (i) programming their own DW applications, using languages such as Python, Java and R; and (ii) interacting with existing DW tools, which often provide access via a Graphical User Interface (GUI). While approaches (i) and (ii) provide benefits, they also have shortcomings. Approach (i), for example, is often associated with completeness of functionality for fulfilling the requirements of the application in consideration; however, it also involves complex application development, advanced programming skills and brittle solutions that cannot be easily applied over data from other sources than the ones for which the solution was originally designed. On the other hand, approach (ii) is often associated with ease of user interaction, limited need for programming skills,

provision of generic functionality that cannot be easily adapted to fulfill specific functional requirements, need for use of multiple tools to perform a single DW job, and limited opportunity for optimizing the DW strategy.

To mitigate the limitations identified in approaches (i) and (ii), we propose a conceptual approach and an architectural solution for DW that combines advantages from (i) and (ii), while offering user interaction via a high-level domain-specific declarative language for specifying DW tasks. The proposed architecture combines functionality from multiple DW tools and access to multiple data sources, by using Web Services technology [6]. The result is an extensible DW approach, able to take advantage of DW functionality implemented within a variety of existing DW tools, and that provides flexibility to allow data analysts to add functionality specific to the requirements of the application in consideration, creating a rich set of DW functions, that can be combined to accomplish simple and complex, general and domain-specific DW tasks. For that, high-level end-user DW requests are automatically mapped into a set of conceptual DW constructs, that are ultimately translated into an execution plan represented as a workflow [7] combining local as well as remote DW functions implemented across a multitude of tools. The proposed approach is tested with use cases from the Urban Traffic domain [8], in which DW tasks associated with common data analysis requests by traffic analysts are implemented using the Taverna platform [9].

An overview of the main DW tools is also provided, where state-of-the-art DW tools are compared and contrasted regarding functionality. This overview aids in the identification of suitable tools and includes information on the capabilities provided by DW tools to fulfill DW requests. Data wrangling technology is still in early stages of development, with a small set of tools supporting a limited range of functional capabilities, as seen in our review of existing tools.

After reviewing existing DW languages and tools we argue that (a) no single tool on its own is likely to cover the entire spectrum of functionality needed to implement DW tasks in the urban traffic domain; (b) there is a wide variety of visual notations and textual-based language constructs to express data wrangling functionality; (c) the learning curve to perform DW is steep mainly due to the need to learn several tools to complete the task; (d) there is limited opportunity for applying optimization techniques across tools.

We believe that a set of conceptual data wrangling operators to be used as an intermediate abstract DW representation language akin to SQL would address issues (b,c,d) and in the long run contribute to the development of comprehensive DW tools addressing issue (a). If DW tools are to attain levels of industrial uptake and commercial success comparable to what relational database languages achieved, it is imperative that research on DW operations and domain-specific languages also tackle DW tasks from a conceptual perspective, aimed at developing widely used DW constructs and languages. This process can ensure transferability of DW skills across different tools and application domains, as it was the case with the adoption of SQL as the mainstream language for relational data management [10]. The work reported in this paper pursues this direction.

Our contributions in this paper include (1) articulating the importance of developing conceptual DW languages with a particular emphasis on supporting complex traffic data wrangling tasks; (2) proposing constructs for conceptual DW operators and a high-level domain-specic language for traffic DW; (3) designing a system architecture to support the conceptual approach to DW, and (4) providing examples of how the conceptual DW approach could be implemented using the Taverna workflow management system.

The paper is organized as follows: Section 2 provides a literature review. Section 3 outlines two examples in a case study developed to illustrate the complexity of data wrangling and motivate the importance of the proposed approach discussed in this paper. Section 4 describes the conceptual DW approach, including the architecture, its conceptual and physical layers, and a discussion of the implementation aspects. Section 5 provides conclusions and future work.

2. LITERATURE REVIEW

2.1. Data Wrangling Activities

Data wrangling comprises a set of activities [11] towards preparing data to accomplish analytical goals. Each activity of the data wrangling process may require the application of several operations to reach the appropriate level of data preparation. Based on the investigation of the DW definitions described in the literature [12][13][14][11], the process of data wrangling involves seven core activities that are applied in an interactive and iterative fashion [12]. The activities are the following: *data profiling* to assess data by providing statistical descriptions of the dataset and identifying its quality issues, *data identification* to check data sources as to whether they can be used to achieve the analytical tasks, *data extraction* to retrieve data from one or multiple data sources towards providing the required information, *data cleaning* [15] to resolve erroneous values and address quality issues, *data re-structuring* to re-structure the data by changing the dataset schema or enriching it with additional information, *data integration* to aggregate data collected from multiple sources, *data visualization* to provide intermediary and final results of the wrangling process using different data representation methods.

2.2. Data Wrangling Tools

Due to the iterative and interactive nature of DW and the complexity involved in the DW process, there has been a body of research aimed at facilitating DW via Graphical User Interfaces (GUIs) and Domain-Specific Languages (DSLs) [16]. For example, in the work of *Kandel et al.* [12], the complexity of conducting DW was decreased, by associating DW functionality with data visualization constructs, allowing users to conduct DW via a visual interface. Tools offering DW functionality, such as Trifacta [17] and OpenRefine [18], provide concise DSLs combined with GUIs to isolate users from the complexities involved in the wrangling process. The DSLs supported by Trifacta and OpenRefine are limited in scope of operator functionality and designed for low-level data manipulations such as regular expressions [19] for text matching with users required to use the GUI for the majority of the DW tasks.

In addition, the level of completeness [20] of the functionality provided by these tools varies based on the DW requirements associated with the task at hand, as well as the characteristics of the target data set. For example, some tools may be suitable for cleaning and profiling certain types of data but may not perform data integration [21] of two or more data sets. An overview and assessment of the available functionality of some of the state-of-the-art DW tools is provided below. Table 1 summarizes the DW activities supported by each tool.

Tabula [22] is a web-based DW tool developed with a focus on data format transformations. It generates Excel, comma-separated values (CSV) or JavaScript Object Notation (JSON) data file formats via extracting data tables uploaded in Portable Document Format (PDF) data file formats.

Mr Data Converter [23] is similar to Tabula in terms of providing only data format transformation operations. Mr Data Converter differs from Tabula as it does not support extracting data from PDF files. Mr Data Converter provides transformation operations via uploading files in either Tab-Separated Values (TSV) or CSV format and can generate output files in a number of formats such as Actionscript, ASP/VB Script, MySQL, Ruby, Hypertext Markup Language (HTML), Extensible Markup Language (XML) or JSON.

Trifacta [17] incorporates operations relating to all DW activities outlined in Table 1 and supports a comprehensive set of data profiling and data re-structuring functionality. Compared to Tabula and Mr Data Converter, Trifacta does not extract and transform data from PDF files as Tabula does. However, there are some overlapping operations between Trifacta and Mr Data Converter as both can convert data files from CSV or TSV to JSON. Trifacta is one of the most popular tools in the data preparation market, allowing integration with a wide range of data science [24] and data ingestion [25] technologies.

OpenRefine [18] is a tool designed with a focus on data profiling [26] and cleaning tasks. It supports the following DW activities: data profiling, data cleaning and data re-structuring. Compared to Trifacta, OpenRefine offers more operations for data cleaning and string manipulation; but lacks advanced statistical and re-structuring operations. There are some overlapping operations between OpenRefine and Trifacta such as moving and dropping columns, date/time operations and array operations. 46 OpenRefine is also similar to Trifacta in its limited support for extracting data from PDF files. In terms of data format transformation, the overlapping operations between OpenRefine and Trifacta relate to converting data formats from either JSON, CSV or TSV to CSV. OpenRefine's big data capabilities have shown limitations to scale to the Gigabyte order of magnitude as indicated in [27].

Talend Data Preparation [28] focuses on importing, structuring and transforming data via a web-based visual interface that enables users to develop data preparation workflows. Although, Talend provides data visualization operations, these operations are limited in support for visual analytics charts for profiling data. Compared to Trifacta and OpenRefine, there are some overlapping data cleaning and re-structuring functionality such as column manipulation operators like rename, create or drop, and fill in missing data. Talend is also similar to OpenRefine in terms of providing functionality for string manipulation, however, Talend Data Preparation supports fewer operations for Date/Time manipulation compared to Trifacta and OpenRefine.

R [29] is a data analysis and statistics language with several DW packages available. R supports all DW activities outlined in Table 1. R packages such as `dplyr` and `tidyr` include functionality to support data profiling, data re-structuring and data integration. R data visualization packages such as `ggplot2` provide powerful data visualization functions. R DW packages also support extensive functionality for data re-structuring and data integration. For example, there are many operations to allow data analysts to create and populate new columns and operations to combine data sets. However, R DW packages lack native support for Date/Time manipulation operations typically used when wrangling spatial-temporal data [30] such as traffic data. The main limitation for R users relates to the need for using low-level programming constructs to customize the code for specific purposes and the steep learning curve associated with learning the R language and finding the required functions scattered across several R DW packages in order to wrangle data.

2.3. Traffic Data Wrangling Challenges

Table 2 illustrates an example of a traffic DW task that was used to assess the capability of generic DW tools using the traffic data set depicted in Figure 1. Note

that a description of the task is provided in the text below the table. The example describes how the DW tools vary in the support for different DW requirements. In addition, it shows that, with the exception of R, all are unable to support Step 3 which complements the time and date information provided in column *Date* with day, month, year and hour information, and using a specific Date/Time format. Although it was possible to implement this step using R, the used function is not natively supported in R, and so data analysts are required to write customized code in order to complete the task.

The difficulties in finding a tool that offers all the functionality required to perform DW tasks forces data analysts to face a steep learning curve to familiarizing themselves with multiple tools and experience a rather laborious and complex process in which data often needs to be transformed/reformatted to be transferred between different tools. In addition, data analysts may still have to use low-level programming constructs implemented in languages such as R, Python or Java to be able to customize code and solve specific data quality issues, despite using the DW tools. A typical example of this case is depicted in Table 2, which illustrates the capabilities and limitations of six of the most popular data wrangling tools towards fulfilling the following traffic data wrangling request: *"Retrieve the value of the average speed of vehicles passing Chester Road on a Friday between 17:00 and 18:00"*. The relevant dataset is illustrated in Figure 1. Note that *OR* is the abbreviation for *OpenRefine*, as *Mr DC* is for *Mr Data Converter*. The version of the Trifacta tool used in this comparison is the free desktop one obtained from [17] and the version of *Talend Preparation Tool* is the free desktop one obtained from [28]. The comparison work is organized in steps, where *Step 1* checks dataset sizes; *Step 2* assesses dataset formats, e.g., CSV; *Step 3* complements and reformats the dataset's time and date properties; *Step 4* extracts the last four digits of the *Site ID* column values, given that only these four digits out of the 16 are relevant; *Step 5* extracts weekday out of the *Date* column; *Step 6* joins the dataset of Figure 1 with another dataset containing the site's description, using *Site ID* as joining key; *Step 7* filters the data based on weekday, road name and time; *Step 8* calculates the average speed of vehicles, using the *Speed (mph)* column.

Particular to traffic data sets, data generated by Bluetooth-based road sensors often include duplicate records [15] relating to the same moving object due to multiple passengers carrying switched-on Bluetooth devices in a vehicle. For removing duplicates, multiple attributes need to be considered, such as vehicle identifier, time/location of detection, and device MAC address. Attributes such as device MAC address and vehicle license plate numbers may be removed from the data set to comply with privacy regulations complicating the task of identifying and deleting duplicates from a data set. Tools offering domain-independent DW functionality (virtually all tools covered in the literature review) would require a complex combination of operators to automate the task of removing duplicates. Missing data is another common issue found in traffic data sets due to extreme weather impact on sensors or malfunction. This can be partially addressed by replacing missing data with data with similar temporal characteristics; e.g., same day of the week and hour for the particular road segment or similar spatial characteristics (nearby roads with identical traffic volume, speed-limit and road topology characteristics) [31]. Spatial Joins [32] using latitude and longitude information are often required to address missing data problems and are often not supported by general DW tools. Outliers [33] are also difficult to address in generic DW tools due to the need to include domain-specific information regarding parameter ranges to distinguish between outliers and noise. Although there are outlier detection operations supported across several generic DW tools, semantic data assistance (for example using domain ontologies [34]) for correlating data with other attributes that are important to decide whether the value is an outlier is not supported.

TABLE 1: Assessing data wrangling tool activity support.

DWA	T	Mr DC	OR	TW	R	TL
Data Identification Activity				Yes	Yes	Yes
Data Extraction Activity	Yes	Yes	Yes	Yes	Yes	Yes
Data Profiling Activity			Yes	Yes	Yes	Yes
Data Re-structuring Activity	Yes	Yes	Yes	Yes	Yes	Yes
Data Cleaning Activity			Yes	Yes	Yes	Yes
Data Integration Activity			Yes	Yes	Yes	Yes
Data Visualization Activity			Yes	Yes	Yes	Yes

Notation: DWA is the Data Wrangling Activity, T is Tabula, Mr DC is Mr Data Converter, OR is OpenRefine, TW is Trifacta Wrangler and TL is Talend.

2.4. Conceptual Data Manipulation Languages

Conceptual data manipulation languages are used to represent or describe data in a high level and abstract form [35], playing the role of Intermediate Representation Languages (IRLs) in a language processing framework [36]. IRLs facilitate the mapping between source and target languages by providing language constructs that are platform agnostic, and that can be easily translated into several target computational platforms and application programming interfaces. Consider the building of a compiler for n source languages (e.g. DW languages for three different domains: education, health and traffic) and for m target platforms/languages (e.g. Java, Python or R). In this

TABLE 2: Implementing a DW task using DW tools.

Step No.	OR	Trifacta	Talend	Mr DC	Tabula	R
1	No.	Yes.	No.	No.	No.	Yes.
2	Yes.	Yes.	Yes.	Yes.	No.	Yes.
3	No.	No.	No.	No.	No.	Yes.
4	Yes.	Yes.	Yes.	No.	No.	Yes.
5	Yes.	Yes.	Yes.	No.	No.	Yes.
6	Yes.	Yes.	Yes.	No.	No.	Yes.
7	Yes.	Yes.	Yes.	No.	No.	Yes.
8	No.	Yes.	No.	No.	No.	Yes.

Capabilities and limitations of six of the most popular data wrangling tools towards fulfilling the following traffic data wrangling request: *"Retrieve the value of the average speed of vehicles passing Chester Road on a Friday between 17:00 and 18:00".*

scenario and without the use of an IRL, *n*m* is the total number of language mappings required. With the use of IRLs placed in the middle between the high-level languages and the implementation layers, the mapping of a source language to multiple platforms using a compiler that maps from the IRL to each target language is possible. Thus, IRLs provide flexibility and extensibility when there is a need to map a multitude of source languages to several target platforms and APIs. The use of conceptual data manipulation languages is particularly relevant in today's fast, dynamic and continuously evolving big data platforms [37]. Conceptual data manipulation languages enable data transformations to be expressed as high-level abstractions [38], separating the problem of data transformation and querying from the platform-specific data format and application programming interfaces.

Developing a conceptual data manipulation language to implement DW tasks can benefit both developers and users of DW tools, by isolating operator functionality design from platform specific features, simplifying development and enabling portability to a variety of target platforms. The search for a minimal set of operators that can address the majority of data manipulation requirements is also another important aspect of language engineering and is based in the parsimony principle in language design [39].

An early work in conceptual languages for information management is the Relational Algebra (RA)[40], serving as foundation for database query languages, and in particular for the Structured Query Language (SQL). Later, with the proposal of new data models, RA has been extended to address new functionality, for example object-orientation giving rise to object-relational models, deductive object-oriented models [41], data quality extensions [42] and algebras to exploit parallelism [43].

Our research has also been influenced by the work on the Papyrus Interface Language (PIL) [44] which serves as an intermediate notation between high-level database languages such as SQL and QSQL and an implementation layer notation. The main motivation and novelty behind PIL was the development of conceptual operators for representing functionality related to some forms of exploitation of parallelism [45] and optimization [46], such as `merge` operator for performing simple iteration, aggregate operators such as `sum` and `average` for performing reduction and `group by` operator for performing grouping functionality.

2.5. Conceptual Data Wrangling Approaches

An important direction relating to conceptual data wrangling involves the development of organisational repositories including a multitude of data sets where end users are provided with self-service and agile data management tools. These data repositories are known as data lakes [5]. In data lakes, schemas (metadata) are not created in advance of the creation of data instances and are produced only when data is retrieved. Query and information retrieval operations support both SQL and NoSQL approaches. Key to the success of a data lake is the development of data wrangling processes and tools made available to end users, providing data cleaning, integration and format conversion of raw data sources into structured data readily available for ingestion by analytical and business intelligence tools. This approach often tries to anticipate the majority of end-user requests to data and therefore provides extensive development of pre-defined data wrangling recipes implemented in the data lake. Compared to other data wrangling approaches discussed in this section and the approach proposed in this paper, data lakes perform the process of data procurement before the data wrangling process in contrast to procuring data on the fly when requested. Data lakes also encode several data wrangling recipes in advance based on the most common data wrangling requests posed by end-user communities.

A seminal work relating to the development of conceptual DW language constructs is Potter's Wheel [47], which incorporates data transformation operations designed based on data manipulation constructs introduced in [48], [49]. A subset of Potter's Wheel's operations are specified at an abstract level and defined using formal methods, for example schema manipulation operations such as `Add`, `Drop` and `Copy` for creating a new column, removing a column and copying a column, respectively. In addition, more advanced operations are also provided such as `Merge` and `Split` operations to concatenate data in two columns and splitting a column data into two columns, respectively. These operations provide core

functionality, however advanced data manipulation operations are still required to wrangle traffic data.

DataCommandr [50] is a data processing engine based on a conceptual model [51], in which data transformations are defined as functions applied over data columns. The approach does not assume the generation of new tables from existing ones, as in relational algebra. In DataCommandr, set-oriented operators such as joins and group-by are explicitly avoided. However, a significant number of traffic data wrangling requests rely on set-oriented transformations, and so, it is not obvious how traffic data wrangling requirements could benefit from this approach, in that numerous iterations over simple column-oriented operations would be necessary to fulfill such requirements.

The VADA data wrangling system [52] focuses on pay as you go data wrangling, proposing an innovative architecture capable of incorporating user feedback/priorities into the data wrangling process. The functionalities involved in the data wrangling process are represented as transducers (software components with input/output dependencies defined using Datalog [53]). The VADA architecture uses a knowledge base to represent user requirements, application domain and metadata. The VADA approach also supports the high-level specification of data wrangling tasks with minimal input from end users.

In the next section, we show data wrangling examples to articulate the importance of developing conceptual DW languages with a particular emphasis on supporting complex traffic data wrangling tasks.

3. TRAFFIC DATA WRANGLING EXAMPLES

Example 1 shows a high-level description of a concise and frequent traffic data analysis request that requires a number of data wrangling steps when departing from raw, sensor-collected data before the request can be fulfilled. For the simplicity and relevance of this request in traffic analysis, it is used throughout the paper as a case study.

```
Example 1: What is the typical Friday Journey
Time (JT) for the fragment of Chester Road
stretching from the Poplar Road to the Hulme
area between 17:00 and 18:00?
```

Chester Road is an arterial road in the city of Manchester (United Kingdom) that links the city center to other popular areas, for example, where the Old Trafford Stadium is located as well as the Trafford Centre, one of the largest shopping centers in the UK. The fragment of Chester Road used in this example has just over three miles in length and links residential areas to the city center and the Old Trafford stadium. As the specified time of the day is within rush hour periods, heavy traffic is expected. The time of the year when the data was collected is February 2018, when no holidays or special scheduled events take place.

Site ID	Date	Lane Name	Direction Name	Class Name	Headway (s)	Gap (s)	Speed (mph)
'000000001083	00:01.2	SB_NS	South	Car			31.691
'000000001083	00:08.0	SB_MID	South	Car			40.39
'000000001083	00:13.0	SB_NS	South	Car	12.6	12.304	47.846
'000000001083	00:14.1	NB_NS	North	Car			31.691
'000000001083	00:26.0	NB_NS	North	Rigid			41.632

FIGURE 1: Excerpt of Traffic Data File 1.

Site ID	Date	Lane Name	Direction Name	Class Name	Headway (s)	Gap (s)	Speed (mph)
'000000001415	00:00.1	SW	SouthWest	Car			46.602
'000000001415	00:03.1	NE_NS	NorthEast	Car			36.039
'000000001415	00:06.2	NE_NS	NorthEast	Car	3.906	2.839	32.932
'000000001415	00:08.0	SW	SouthWest	Car			34.176
'000000001415	00:13.1	SW	SouthWest	Car	3.676	4.066	44.117

FIGURE 2: Excerpt of Traffic Data File 2.

To prepare or "wrangle" the data to answer the question asked in *Example 1*, three data files need to be integrated, all of which were obtained from the Manchester Traffic Authority, TfGM, information systems. Two of them contain records describing vehicles detected in February 2018, each generated from a fixed location on Chester Road where Inductive Loops constantly collect data. Between these two sites Journey Time (JT) is to be estimated. Excerpts of the files are shown in Figures 1 and 2. The third file contains static data about each site of the city with fixed sensors and provides information about the length of the road fragment between two consecutive sites, such as the ones considered in this section.

In the following paragraphs, a description of the wrangling process that was carried out to get the data files ready to answer the case study question is provided. The wrangling process or "recipe" was devised step-by-step and interactively using the open source Trifacta data wrangling tool [17]. This process is illustrated in Table 3, showing a rather long sequence of data wrangling steps leading to the answer to the question. It is worth pointing out that, while 11 steps are described in the table, each step is composed of multiple sub-steps, suggesting that a significant amount of user interaction is required to answer a simple traffic analysis question departing from the raw data files provided. In addition, prior to Step 1, complex processing involving the `Datetime` column, *Date*, in the data files shown in Figures 1 and 2 had to be performed because a selection operation based on day of the week and time of the day is required in Example 1, but the date and time information available from the files includes only minutes and seconds when a vehicle was detected. Therefore, iteration over all records of the file identifying the start of a new hour and a new day had to be performed. As this functionality is absent in Trifacta, it was implemented as a separate piece of Python code, and its result was input into Trifacta.

The rationale behind each data wrangling step in Table 3 is explained as follows. While no order of user actions is imposed by the tool, a view of possible

dependencies between actions and some background in relational databases are required from a user to successfully wrangle the files.

- Step 1 is required to bring the input files into context and make them accessible from the same location. This step is in fact composed of three sub-steps, each associated with the loading of one of the files. While it was not necessary to sequentially upload the files into Trifacta in any particular order, as suggested in Table 3, it is believed that the decision to upload them before any other action is performed does not incur any errors.
- Steps 2, 3 and 4 are associated with the disposal of irrelevant columns. The decision about which columns to discard from each of the files should be based on knowledge of which columns should be used to answer the question as well as on how the files are to be integrated in later steps. In this particular example, a union operation between the files in Figures 1 and 2 is carried out in Step 5 to generate a single file from the two, and so, following the removal of columns, the files should present the same structure. Note that background in relational databases is useful at this point as the user should know that the union of two csv files requires the files to present the same structure. Because columns are removed one at a time in Trifacta, each of Steps 2 and 3 is composed of 13 sub-steps since, for each file, 13 columns are removed.
- Step 5 requires the user to work with a pop-up window to input parameters for the union operation involving the files in Figures 1 and 2. The integrated file is called $\text{file}_{1,2}$.
- Step 6 is associated with the extraction of information about day of the week from the `Datetime` column, and was carried out with the use of the Trifacta function `WEEKDAY`. As a result, a new column, *Weekday* is generated.
- In Step 7, the `Datetime` column was split into two separate columns, one of type `Date` and another of type `Time`, so that the selection predicate `'17:00:00'<= Time < '18:00:00'` could be later applied over the `Time` column. As in the previous steps, this requires multiple actions from the user to input parameters to the operation via a pop-up window.
- Step 8 encompasses all the actions required to allow the application of a number of selection predicates over the file, such as `Direction Name = 'North' or 'NorthEast'`, depending on the site, `Weekday = 'Friday'`, `'17:00:00'<= Time < '18:00:00'`. Note that each selection requires multiple actions from the user and each action is typically performed one at a time.
- Step 9 is associated with the join between the currently active file, $\text{file}_{1,2}$, and the third file uploaded in Step 1, which contains the length of the road fragment in consideration, *LinkLength*. As in Step 5, a pop-up window is offered to the user to allow parameterization of the operation. This join is performed as a left join on the *site IDs* of the files.
- Finally, Steps 10 and 11 are associated with the calculation of Journey Time (JT) between the two road sites, which requires obtaining the average speed of vehicles going from one site to the other during the specified time period, *Average Speed*, as well as the length of the road fragment, *LinkLength*.

TABLE 3: Data Wrangling with Trifacta for Example 1.

Step	Description
1	upload the three input files (file_1, file_2 and file_3)
2	remove all unwanted columns from file_1
3	remove all unwanted columns from file_2
4	remove all unwanted columns from file_3
5	apply a union operation involving file_1 and file_2
6	extract 'Weekday' from 'Date'
7	split the 'Date' column into 'Date' and 'Time' columns
8	apply filters
9	apply a join operation between the current file and file_3
10	apply average on 'Speed'
11	apply division between 'Average Speed' and 'LinkLength'

Despite the number of steps required to prepare the data to answer the question in Example 1, Trifacta is perhaps the most popular data wrangling tool, particularly for its friendly user interface and flexibility in allowing users to decide what step to do next. Its generality, though, makes it not ideal for traffic data wrangling, and the level of flexibility it offers leaves complex decisions to be made by the traffic analyst.

A second and more complex traffic data wrangling example is provided in *Example 2*, where the average and hourly speed of vehicles on rainy days is compared against the average and hourly speed of vehicles on dry days. For simplicity, the availability of the average hourly speed of vehicles on dry days is assumed, and the estimation of the average hourly speed of vehicles on rainy days is made considering only one day of the week, Friday. To prepare or "wrangle" the data to answer the question asked in this example, the traffic data collected for Chester Road (shown in Figure 1) needs to be integrated with two other files. One of the files contains static data about each site of the city with fixed sensors, which not only provides information about the length of the road fragment between two consecutive sites, used in the previous example, but also latitude and longitude information relating to each site, which is needed in this example. The other file contains weather information provided by the MET office, the United Kingdom's national

```
"i": "3238",
"lat": "55.02",
"lon": "-1.88",
"name": "ALBEMARLE",
"country": "ENGLAND",
"continent": "EUROPE",
"elevation": "146.0",
"Period": [
    {
        "type": "Day",
        "value": "2018-02-02Z",
        "Rep": [
            {
                "D": "W",
                "H": "69.0",
                "P": "1008",
                "S": "15",
                "T": "17.1",
                "V": "40000",
                "W": "3",
                "Pt": "R",
                "Dp": "11.4",
                "$": "960"
            },
            {
                "D": "W",
```


FIGURE 3: Excerpt of the Weather Data File in JSON Format.

weather service. An excerpt of this file is shown in Figure 3. The file contains information about weather measurement units, observation location, date, time, and weather related details, such as temperature, wind direction, and weather conditions. The observation location is encoded as latitude and longitude; and date and time are separated into two attributes, where date is formatted in ISO 8601 standard and time is represented as minutes calculated after midnight, i.e. 00:00. The weather condition is encoded in numbers and different ranges of numbers represent different weather conditions. As there are numerous possible values representing the actual weather condition, these values were generalized into several broader weather conditions.

Example 2: On rainy weekdays, is the average hourly speed of vehicles on Chester Road, near Poplar Road, lower than that typically observed on dry days?

The steps taken to wrangle the data files to answer the question in Example 2 are described in Table 4. File_1 denotes the traffic data file in Figure 1, file_2 denotes the static site data file, and file_3 denotes the weather data file in Figure 3.

Using Trifacta to answer the question in Example 2 proved to be challenging for the reasons enumerated below. In particular, only data wrangling functions used to, for example, join two data files, group records based on hour of the day, and calculate the average speed of vehicles on rainy days could be performed using Trifacta in a straightforward way. More complex traffic data wrangling functions, such as the ones enumerated below, had to be encoded in a programming language and carried out outside Trifacta.

1. As in Example 1, a separate piece of Python code implementing the complex processing involving the `Datetime` column, *Date*, in the data files shown in Figures 1 and 2 was used prior to loading the files into Trifacta, as a later selection of records based on day of the week had to be performed.
2. The integration between the traffic data files and the weather data file required a complex join operation involving time, latitude and longitude, which could not be performed using Trifacta. The reason is the fact that the latitude and longitude for the traffic observation site were not identical to the ones for the weather observation site. This was also true for the date and time properties for the sites. And so, a spatial-temporal join was implemented in Python to enable matches between nearby geographical locations and temporal properties. This operation was used after the weather data file was converted from its original json format into csv format, also outside Trifacta.
3. Some of the available functionality within Trifacta could not be successfully used in this example without additional data manipulation outside Trifacta, described as follows: (i) the facility to join a json file with a csv file, because of the complex nesting of elements in the json file causing the need to manually remove some of the elements; and (ii) the facility to visually display results using bar charts, mainly because the histograms that are automatically provided at the top of each column in a Trifacta csv file were not detailed enough for the purposes of the application and could not be customized.

TABLE 4: Data Wrangling with Trifacta for Example 2.

Step	Description
1	upload the three input files
2	remove all unwanted columns from file_1
3	remove all unwanted columns from file_2
4	remove all unwanted columns from file_3
5	apply a join between the file_1 and file_2 (generating $\text{file}_{1,2}$)
7	derive 'Weekday' from 'Date'
6	split the 'Date' column into 'Date' and 'Time'
8	apply filter (`Weekday = 'Friday'`)
9	join current file with file_3
10	apply filters associated with weather conditions
11	group records by hour of the day applying average on 'Speed'

The examples discussed above show the considerable semantic gap between the high-level data wrangling requirements expressed in *Example 1* and *Example 2*, the steps used in the data wrangling recipes as outlined in tables 3 and 4, respectively, and the execution of the recipes using a combination of GUI-based manipulations, available in DW tools, combined with coding of tasks using programming languages to address

steps that GUI-based tools are unable to perform. And so, in this paper, we argue for the principle of separation of concerns [54] to be applied to the overall problem of data wrangling, with data wrangling requirements expressed in a conceptual high-level notation (D^2WL), the DW recipes that need to be performed towards addressing the high-level requirements to be expressed in an intermediate conceptual DW operator language (DWL), and for the intermediate operator language to be mapped into target application programming interface functionality of existing DW tools and data science programming languages. This will support the separation of three key aspects of the DW process: (1) describe what needs to be wrangled; (2) organize the DW sequencing into steps involving simple DW functions that together form the DW recipe; (3) executing the recipe in the target data/computational environment. The following section illustrates the conceptual data wrangling approach proposed in this paper.

4. A CONCEPTUAL APPROACH TO TRAFFIC DATA WRANGLING

4.1. Architecture Overview

Figure 6 illustrates the proposed Data Wrangling (DW) architecture. The architecture is composed of three main layers: (1) an external User Interface (UI) layer, (2) a conceptual operator layer, and (3) a physical layer. The UI layer enables the submission of expressions in a domain-specific language (D^2WL) to be input to the system. As a result of on-going work, the user will be able to write D^2WL expressions with the help of a graphical user interface (GUI), in addition to writing the expressions directly using the syntax of the language. This GUI should also allow the user to visualize the desired remote or local documents before inputting them into the system.

Transformation of a user's D^2WL expression into an executable plan is gradual and takes place within a component in the UI layer, the parser, as well as a number of other components in the conceptual operator and physical layers. The parser is responsible for validating the submitted expression by checking the correctness of the syntax, the URLs to input data sources and any available schema information. For the examples described in Section 3, no schema information is used, therefore understanding of the structure of the documents specified in the input expression is left to the user, who not only needs to know where the documents are located, but also have a understanding of their schema prior to writing an expression. Once the input expression is validated, it is mapped into an initial DWL expression, which is then input to the conceptual operator layer.

In the conceptual operator layer, a number of DWL expressions are generated from the initial DWL expression generated during parsing, from which one is selected to be input to the physical layer. Currently, the number of expressions considered in this process is limited and the expressions are generated via the application of a few heuristics. Examples of heuristics include application of data filters as early as possible in the process, possibly before the execution of any data integration operation. The selection of the best expression is also heuristic-based.

In the physical layer, the selected DWL expression is mapped into a workflow composed of calls to web services. By using descriptions of the services and other information encoded in the input DWL expression, such as the order in which services are to be called and the format of the input data, a physical execution plan is generated. Considering all servers from which similar DW functionality is available, the server that offers all (or most) of the functions required to fulfill the DW expression is chosen as the main source of functionality and the location where the expression is to be executed. This simple heuristic helps avoiding overhead costs associated with the transmission of intermediate results through the network, by allowing all or most of a DW job to be carried out in a single location. However, it is not always the case that one server will contain all the necessary functionality to fulfill a DW request and, so, transmission of input and output between servers is sometimes inevitable.

This stage of the input transformation is associated with a number of challenges that are to be addressed in future work. For example, investigation of the possibility of using machine learning techniques in the generation of equivalent DWL expressions from a single one, and of methods for mapping DWL functions into appropriate distributed services (services composition) considering a cost model and service descriptions.

4.2. The Declarative Language Used in the External Layer

The declarative language, D^2WL, is designed for data analysts with limited or no programming skills and so it is based on a small number of clauses. The main clauses are used to define the location and format of input data sources (the `FROM` clause), the location and format of results (the `TO` clause), the main data wrangling activities to be carried out (the `WRANGLE BY` clause), the data elements to be present in the result data set (the `SELECT` clause), and other data wrangling activities using clauses such as `GROUPBY`.

Figures 4 and 5 show D^2WL expressions for the traffic data wrangling examples 1 and 2, respectively, described in Section 3. The expression in Figure 4 uses all the main clauses of the language. In the `SELECT` clause shown in line 1, the structure of the desired result is specified as a single column and single row table, containing the calculated JT (*JTFri17to18oclock_1083_to_1415*) for the road fragment. The `TO` clause in line 5 specifies the location

and the format of the output. Note that the output is specified as a csv file that should contain the table specified in the SELECT clause in line 1. In the FROM clause, line 2, the URL and format of the files to be input to the wrangling process are described, in this case, the two csv traffic data files and the csv file describing static information about all the fixed sensor collection sites in the city of Manchester. Similarly to SQL, in the WHERE clause, line 6, row filters are described, such as the ones constraining the days of the week to only Friday, the time of the day to be between 17:00 and 18:00, and the lanes of the road, by selecting the ones leading in the North direction. Finally, in the WRANGLE BY clause, line 15, the data integration operations of join and union, used to generate a single file from the input ones, are defined. Note that, because all input files to be integrated are in csv format, which represent the canonical data manipulation format used within the system, there is no need to convert the files into csv prior to integration, which is done using the TRANSFORM function (shown in Figure 5, line 13). Also in line 15 of Figure 4, the data wrangling function ENRICH_TIMESTAMP is used to format and add missing information to an existing Timestamp or Datetime column of the input file. This function is able to iterate over the entire column, adding information such as calendar dates and hour of the day, departing from an initial date (provided as an input parameter) and time (typically midnight). Note the use of another wrangling function specific to traffic data, JT in the SELECT clause (line 1), which takes two numerical parameters as input (road length and speed) to calculate journey time. Examples of other functions specific to traffic data wrangling that are available to be used in other examples include functions to convert speed, road length and time units, e.g., from kilometers/hour to miles/hour, etc.

```
1  SELECT JT(LinkLength, AVG(Speed_in_MPH))
2  FROM http://www.informationtraffic.com/rawpvr_2018-02-01_28d_1083.csv as f1,
3       http://www.informationtraffic.com/rawpvr_2018-02-01_28d_1415.csv as f2,
4       http://www.informationtraffic.com/StaticSitesInfo.csv as f3
5  TO http://www.informationtraffic.com/example_1_results.csv
6  WHERE f1.Direction_Name = 'North' AND
7        DAYOFWEEK(DATE(f1.Date)) = '6' AND
8        TIME(f1.Date) >= '17:00:00' AND TIME(f1.Date) < '18:00:00' AND
9        f2.Direction_Name = 'NorthEast' AND
10       DAYOFWEEK(DATE(f2.Date)) = '6' AND
11       TIME(f2.Date) >= '17:00:00' AND TIME(f2.Date) < '18:00:00' AND
12       (f1.Site_ID = f3.StartSite_ID OR f2.Site_ID = f3.EndSite_ID) AND
13       f3.StartSite_ID = '000000001083' AND f3.EndSite_ID = '000000001415'
14 WRANGLE BY ENRICH_TIMESTAMP(f1.Date,'%d/%c/%Y %H:%i:%s','01/02/2018'),
15            ENRICH_TIMESTAMP(f2.Date,'%d/%c/%Y %H:%i:%s','01/02/2018'),
16            (f1 UNION f2);
```

FIGURE 4: D^{2}WL expression for Example 1.

```
1  SELECT PLOT(bar, AVG(f1.Speed_in_MPH), m="Hourly Avg Speeds",h, Ynames=(
2         "00:00", "01:00", "02:00", "03:00", "04:00", "05:00","06:00", "07:00", "08:00","09:00",
3         "10:00", "11:00","12:00", "13:00", "14:00","15:00", "16:00", "17:00","18:00", "19:00",
4         "20:00","21:00", "22:00", "23:00"))
5  FROM http://www.informationtraffic.com/rawpvr_2018-02-01_28d_1083.csv as f1,
6       http://www.informationtraffic.com/StaticSitesInfo.csv as f2,
7       http://www.informationweather.com/weatherUK.JSON as f3
8  TO http://www.informationtraffic.com/example_2_results.pdf
9  WHERE f1.Direction_Name = 'North' AND DAYOFWEEK(DATE(f1.Date)) = '6' AND
10       f1.Site_ID = f2.StartSite_ID AND f3.W = 'wet'
11 WRANGLE BY TRANSFORM(f3, json, csv),
12            ENRICH_TIMESTAMP(f1.Date,'%d/%c/%Y %H',"01/02/2018"),
13            ENRICH_TIMESTAMP(f3.period,'%d/%c/%Y %H'),
14            (f1 JOIN f3 ON (f1.Latitude ~ f3.lat AND f1.Longitude ~ f3.lon AND
15                            DATE(f1.Date) = DATE(f3.period) AND
16                            TIME(f1.Date) ~ TIME(f3.period))) as f4
17 GROUP BY TIME(f1.Date);
```

FIGURE 5: D^{2}WL expression for Example 2.

Additional features of the language are illustrated in the expression in Figure 5. For example, in lines 16, 17 and 18 a spatial temporal join operation is used to integrate traffic with weather data where latitude, longitude and time property values need to be approximated rather than exact, where symbol ~ is used. Note that the conversion of the weather data file from json format into csv is requested by using the TRANSFORM function in line 13. It is worth pointing out that csv is the canonical data format, thus all input files that are not in csv need to be converted into csv format before being integrated and further processed. Also note in the SELECT clause (line 1) the request to display the results (24 speed averages for each hour of the day considering all Fridays of February 2018) as a bar plot, to be stored in a pdf file, as specified in the TO clause in line 8.

While data integration operations can be specified in the WRANGLE BY, WHERE and FROM clauses (e.g., left joins), the data wrangling operations that represent extensions to the classic SQL are mostly requested in the WRANGLE BY clause.

Compared to the data wrangling processes described in Section 3 for both examples, the expressions in Figures 4 and 5 present the user with the advantage of avoiding the construction of platform-specific data wrangling strategies step-by-step and, instead, writing a single expression, or a number of them, to articulate in a declarative way *what* information management and data wrangling tasks are required. Also, the user is given the advantage of not having to specify DW tasks in the order they need to be executed, as the system orders them according to their dependencies. The choice of the number of expressions to write for obtaining a result is left to the user and may depend on the user's need to visualize intermediate results. Future work includes the proposal of additional syntax for expressing the need for visualization of multiple intermediate results within a single expression. Another

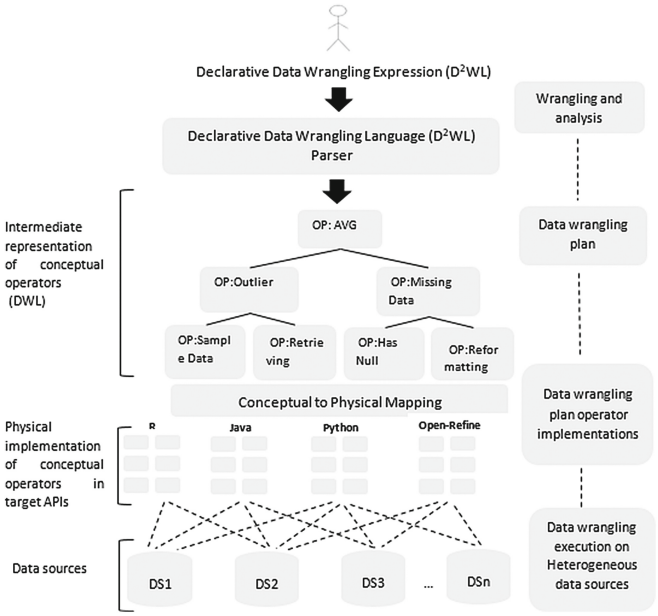

FIGURE 6: Data wrangling system architecture.

advantage relates to the provision of combinations of data wrangling functionalities that would not typically be found in any single tool, including functions that are specifically designed to facilitate traffic data wrangling. It is also worth pointing out that the approach provides more extensibility and flexibility in the visualization of results, allowing the user to customize visualization facilities via a number of parameters. One potential future direction to explore is the extension of the notation to incorporate comprehensive graphical primitives such as the constructs proposed in the Protovis toolkit [55], and an evaluation of the complexity of data wrangling tasks achieved and DW effort savings obtained with the conceptual approach to data wrangling.

4.3. The Conceptual Operator Layer

Figures 7 and 8 show expressions in DWL into which the D²WL expressions in Figures 4 and 5, respectively, are mapped following parsing. A high-level description of each operator is provided as follows:

- Although it is omitted, for simplicity, in Figure 7, lines 1, 6 and 11, and in Figure 8 in lines 2, 8 and 12, the `read` operator takes an URL as input parameter identifying the name, format and location of the input file, possibly on a remote server, and brings the file into context. In example 1 (Figure 7), the input files are csv files obtained from TfGM Information Systems. In example 2 (Figure 8), a json file obtained from the MET Office data point is also used, in addition to the TfGM files.
- Operator `transform` takes a file, a from-format, and a to-format as input parameters and converts

```
1  write(project(join(union(project(select(enrich_timestamp(read(f1),
2                                                  Date,'%d/%c/%Y %H',"01/02/2018")
3                                  Direction_Name = 'North' AND
4                                  DAYOFWEEK(DATE(Date)) = '6' AND
5                                  TIME(Date) >= '17:00:00' AND TIME(Date) < '18:00:00'>)
6                           <Site_ID, Speed_in_MPH>)
7                    project(select(enrich_timestamp(read(f2),
8                                                  Date,
9                                                  '%d/%c/%Y %H',
10                                                 "01/02/2018")
11                                 <Direction_Name = 'North' AND
12                                 DAYOFWEEK(DATE(Date)) = '6' AND
13                                 TIME(Date) >= '17:00:00' AND
14                                 TIME(Date) < '18:00:00'>)
15                          <Site_ID, Speed_in_MPH>))
16             project(select(read(f3),
17                                 StartSite_ID='000000001083' AND EndSite_ID='000000001415'),
18                          <StartSite_ID, EndSite_ID, LinkLength>),
19                    f3.StarSite_ID=f1.Site_ID OR f3.EndSite_id=f2.Site_ID),
20              <LinkLength, Speed_in_MPH>),
21              JT(LinkLength.AVG(Speed_in_MPH)),
22       http://www.informationtraffic.com/example_1_results.csv)
```

FIGURE 7: DWL expression for Example 1.

```
1  write(
2  barplot(
3   group_by(
4    project(spatialtemporal_join(join(project(select(enrich_timestamp(read(f1),
5                                                                  Date, '%d/%c/%Y %H',
6                                                                  "01/02/2018")
7                                                     Direction_Name = 'North' AND
8                                                     DAYOFWEEK(DATE(Date)) = '6')
9                                                  <Site_ID, Date, Speed_in_MPH>)
10                                       project(select(read(f2),
11                                                      StartSite_ID = '000000001083')
12                                                  <StartSite_ID,Latitude,Longitude>),
13                                       f1.Site_ID = f2.StartSite_ID)
14                                    project(enrich_timestamp(select(transform(read(f3),
15                                                                              json, csv)
16                                                                    W='wet'),
17                                                                 f3.period,'%d/%c/%Y  %H')
18                                                <period, lat, long>),
19                                    f2.Latitude ~ f3.lat AND f2.Longitude ~ f3.lon
20                                    AND DATE(f1.Date) = DATE(f3.period)
21                                    AND TIME(f1.Date) ~ TIME(f3.period)),
22              <Date, Speed_in_MPH>),
23     AVG(f1.Speed_in_MPH), TIME(f1.Date)),
24  Avg_Speed_in_MPH, m="Hourly Avg Speeds", h,
25  Ynames=("00:00","01:00","02:00","03:00","04:00","05:00","06:00","07:00","08:00"
26          "09:00","10:00","11:00","12:00","13:00","14:00","15:00","16:00","17:00",
27          "18:00","19:00","20:00","21:00","22:00","23:00")),
28  http://www.informationtraffic.com/example_2_results.pdf)
```

FIGURE 8: DWL expression for Example 2.

the file from its original format (from-format) into a csv format (to-format). A `transform` operator is shown in lines 12 and 13 of Figure 8, where the conversion is from json format.

- Operator `enrich_timestamp` complements missing information from a `Timestamp` or `Datetime` column while formatting it according to a desired format and, possibly, a start date. Its input parameters are the name of the `Datetime` column, the desired format and, in some cases, a start date. In Figure 7 two `enrich_timestamp` operators appear in lines 1 (with input parameters extending to line 2) and 6 (with input parameters extending to line 7). In Figure 8 two `enrich_timestamp` operators appear in lines 2 (with input parameters extending to lines 3 and 4) and 12 (with input parameters extending to line 15).
- Operator `select` filters out the input's rows for

which the specified condition evaluates to false. Its input parameters include an input table and a boolean expression. In Figure 7, three `select` operators appear in lines 1 (with input parameters extending to lines 3 and 4), 6 (with input parameters extending to lines 8 and 9) and 11 (with input parameters extending to line 12). In Figure 8, three `select` operators appear in lines 2 (with input parameters extending to lines 5 and 6), 8 (with input parameters extending to line 9) and 12 (with input parameters extending to line 14).

- Operator `union` merges two input tables, passed as input parameters, with similar structure into a single one. In Figure 7, one `union` operator appears in line 1 (with input parameters extending to line 10).
- Operator `join` has the same semantics as SQL's left_join, i.e., it integrates two datasets, *d1* and *d2* on given key columns to which they are joined, resulting in a third dataset, which contains all observations from *d1* coupled with matching observations from *d2*. Its input parameters are the two tables to be joined and a boolean expression representing the join condition. In Figure 7, a `join` operator appears in line 1 (with input parameters extending to line 14). In Figure 8, a `join` operator appears in line 2 (with input parameters extending to line 11).
- Operator `spatialtemporal_join` is associated with spatial temporal functionality to calculate approximated matches of geographical locations using latitude and longitude, as well as time related approximations, as described in Section 3. Its input parameters are the two tables to be joined and a boolean expression representing the join condition, which typically involves latitude and longitude valued columns and, possibly, time-related columns. In Figure 8, a `spatialtemporal_join` operator appears in line 2 (with input parameters extending to line 19).
- Operator `project` selects columns from the input table to be kept for the next steps of the processing, while the ones which are not present in the operator's input list of column names are discarded. Both the input table and the list of column names are passed to the operator as input parameters. In Figure 7, two `project` operators appear in lines 1 (with input parameters extending to line 5, and to line 15, each), one in line 6 (with input parameters extending to line 10) and another one in line 11 (with input parameters extending to line 13). In Figure8, two `project` operators appear in line 2 (with input parameters extending to line 7 and line 20, each), one in line 8 (with input parameters extending to line 10) and another one in line 12 (with input parameters extending to line 16).
- Operator `group_by` groups rows of an input dataset according to the values in a list of columns, while applying an aggregate function over one of the columns. The input dataset, list of columns and aggregate function are passed to the operator as input parameters. In Figure 8, a `group_by` operator appears in line 1 (with input parameters extending to line 21).
- Operator `write` writes its input into a file whose location and type are specified in an URL. Its input parameters typically include an input table and an URL, but can also include the output of a visualization operator that can be written to a pdf file. In Figure 7, a `write` operator appears in line 1 (with input parameters extending to line 22). In Figure 8, a `write` operator appears in line 1 (with input parameters extending to line 28).
- Operator `barplot` is a visualisation operator that draws a barplot from its input parameters, which include an input table, from which values for the x and y axis are provided, display orientation, legend, and labels. In Figure 8a barplot operator appears in line 2 (with input parameters extending to line 27).

The mapping from a D^{2}WL to a DWL expression is performed during parsing, where a canonical DWL expression is generated and input to the conceptual layer for improvement via the application of a few heuristics. A few of the most basic mapping rules are described as follows:

- A `SELECT` clause is mapped into a `project` operator, which selects a number of columns from its input dataset to be displayed as results.
- A `FROM` clause is mapped into one or more `read` operators, each associated with an input dataset.
- A `TO` clause has is mapped into the `write` operator.
- A `WHERE` clause is mapped into one or more `select` and/or `join` operators, where the expressions in this clause are interpreted as selection and join predicates.
- A `WRANGLE BY` clause can be mapped into a variety of conceptual operators, depending on which DW tasks are specified in this clause, including `transform`, `enrich_timestamp`, `union` and `spatialtemporal_join`.
- A `GROUP BY` clause is mapped into a `group_by` operator.
- D^{2}WL functions such as `JT`, `AVG`, `DATE`, `DAYOFWEEK`, `TIME`, `PLOT`, etc. are mapped into their conceptual counterparts, which have the same name (with the exception of `PLOT`, which has a number of counterparts) and list of input parameters.

Note that one of the main heuristics applied to the canonical DWL expression generated during parsing relates to the insertion of `project` operators into the expression to discard columns that are irrelevant to the remaining processing, decreasing the size of the intermediate results.

The output produced in the conceptual operator layer is a DWL expression that is input to the physical layer to be ultimately mapped into a sequence of workflows, where each workflow is composed of input and output ports and a set of services orchestrated with the support of a workflow management tool. In Section 4.4 the functionality associated with the physical layer is described.

4.4. The Physical Implementation Layer

In the physical layer, each operator in the input DWL expression is mapped into a workflow that encompasses one of more calls to web services. To manage these calls, the Taverna workflow management system [56] is used to orchestrate and execute DW operations implemented within different target tools. The choice of each service composing the implementation of a DWL expression is currently made using heuristics based on the availability of functionality from a number of service providers and on minimizing communication costs. For example, once the input data files to a DWL expression are retrieved from their sources, the intermediate results derived from them are forced to be transferred from one service to another without being brought to the client. In other words, due to the large sizes of data files, all processing takes place on the remote servers, and the servers of choice are the ones that offer the majority of the required functionality. It is the responsibility of the `write` operator to transfer the final result to a data server to be accessed by the client.

Figure 9 illustrates the workflow representation of the DWL `read` operator used in the expression shown in Figure 7. Each argument of an operator is mapped into an input port in a Taverna workflow and its results, mapped into output ports of the workflow. A physical layer workflow typically includes three processors, each associated with either a Beanshell service or a REST service. For example, as Figure 9 suggests, a RESTful component was used to call the `read` operator, a function in the R package `Utils`, offered as a RESTful (OpenCPU framework) web service. The input argument to R's `read` operation is an URL, which is sent to the server as an HTTP request body, encoded as a Taverna Beanshell component. This component offers the advantage of allowing parameters with special characters to be encoded into a format that is accepted by HTTP. R's `read` operator then imports a dataset from a remote server and outputs an URI that directs towards the same data set, which may be stored locally or on a remote server. The result data of a wrangling workflow is referred to by the data session key, also encoded as a Taverna Beanshell component, as illustrated in Figure 9, and this key is passed from one workflow to the next.

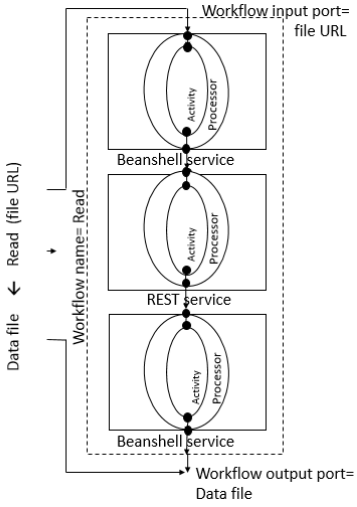

FIGURE 9: Details of the Taverna workflow for the conceptual read operation.

4.4.1. Data Wrangling Tools and Functions

Table 5 shows associations between each DW tool that has been considered as possible service provider for examples 1 and 2, and the DW functionality available from its API. Note that Trifacta in not in the table because its API is not available. Also, note the limitations of OpenRefine compared to the DW functionality offered by R's packages such as `tidyr`, `dplyr` and `ggplot2`. Using the operations from these packages, the majority of the wrangling requirements from the examples are satisfied. The main exceptions include specific traffic DW functionality, which had to be implemented in Python, e.g., enrich/complement timestamp and the spatial and temporal join and OpenRefine's facilities to transform a json file into a csv file, because R's facilities could not handle lists of pairs of values, present in the json input file to example 2.

The developed algorithm for the function responsible for enriching/complementing a timestamp is based on the following assumptions: (i) observations are pre-sorted in ascending order, so that the oldest observation is placed first in the file; and (ii) vehicles are detected at each hour within the observed period. More specifically, the algorithm iterates through the observations in the dataset and, as it progresses, performs checks to compare if the minute of the current observation, n, is less than that of the previous observation, $n-1$. If it is, the hour is incremented. Furthermore, the hour is checked to find if it has surpassed the day limit of 24 hours. If so, then the day is incremented and hour restarted to zero. The enriched timestamp is then concatenated to the original traffic data.

The spatial and temporal join algorithm solves geospatial (i.e., latitude and longitude) entity matching problems by implementing coordinate matching [57]. The distance function used in the implementation of

Algorithm 1 Spatial and Temporal Join Pseudocode

```
1: function SpaceTimeJoin(x, y, threshold)
2:     x̂ = ReadFromUrl(x)
3:     ŷ = ReadFromUrl(y)
4:     d = Array()
5:     x_rows = SizeOf(x̂)
6:     for it = 0 to x_rows do
7:         δ = CalculateDistances(x̂_it, ŷ)
8:         SortAscending(δ)
9:         if δ_0 ≤ threshold then
10:            d_it = δ_0
11:        else
12:            d_it = NA
13:        end if
14:    end for
15:    result = Merge(x̂, d)
       return result
16: end function
17: function CalculateDistances(x, y)
18:    δ_haversine = HaversineDistance(x, y)
19:    δ_temporal = TimeDistance(x, y)
20:    δ = Merge(δ_haversine, δ_temporal)
       return δ
21: end function
```

coordinate matching is the Cosine-Haversine formula described in [58], which calculates the distance between two geospatial locations. As both traffic and weather datasets are spatial and temporal, and coordinate matching solves only the geospatial aspect of the problem, the temporal aspect was solved using simple temporal matching based on hour of the day.

Due to the complexity involved in joining approximate latitude, longitude and time information, the spatial and temporal join algorithm is shown in Algorithm 1. More specifically, the algorithm takes datasets x and y as parameters, and configurable *thresholds* for the time and space distances, so that if the nearest weather observation is greater than the accepted threshold, then it is neglected. The function yields a dataset which represents the merging of the two input datasets. Similar to the timestamp enrichment function, the resulting dataset is then given a name and stored.

Figure 10 shows an activity flow for example 2, associating the functions used in the execution of example 2 with the tool from which the function was called. Note that, as Taverna calls a service from one of the tools, the server responds by executing one step in the workflow, which can be associated with the retrieval of a relevant file from one of the data sources, denoted as DSi_{j} (i.e., file j retrieved from Data Source i). Also, all operators shown in this figure represent implementations associated with one or more conceptual operators offered as services called from Taverna. The mappings from the DWL operators into the services shown in Figure 10 are described in Table 6.

TABLE 5: Tools and Respective Functions Considered in Examples 1 and 2

General Funct.	OpenRefine	R	Python
import tabular file format		X	
import JSON file format	X		
export to tabular file	X	X	
select columns	X	X	
filter		X	
create column	X	X	
rename column		X	
enrich timestamp			X
group by		X	
summarize		X	
join		X	
spatial temporal join			X
bar chart		X	

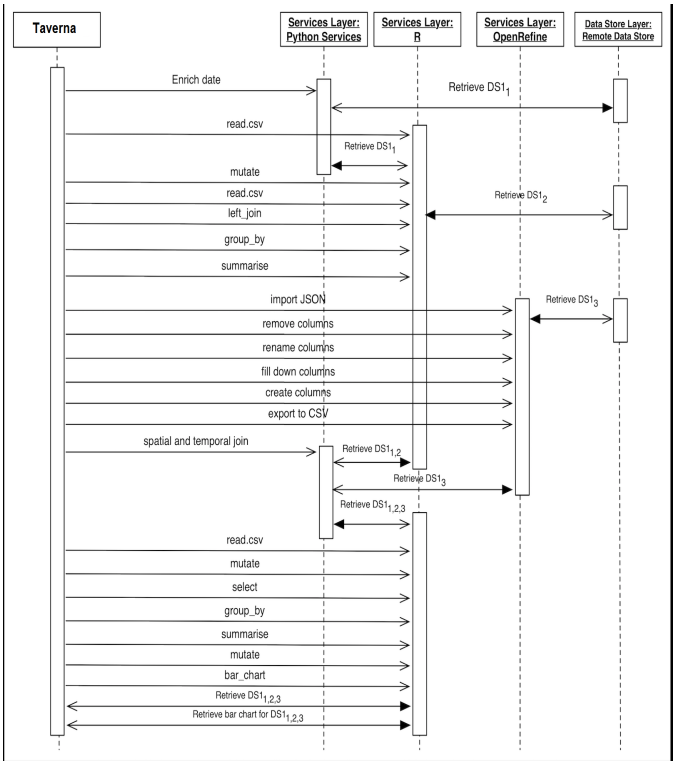

FIGURE 10: Activity Flow for Example 2.

4.4.2. Interactions between Taverna and the Tools

R, OpenRefine and Python services were each assigned to a designated server in the current prototype. While OpenCPU served DW operations from R packages, the OpenRefine server hosted a list of its own functionalities, and a separate Python server provided operations that were specific for traffic data wrangling. For communication between services to be possible, data formats had to be translated into one which the destination service could work with. For example, in example 2, OpenRefine's json to csv service was used so that the data could become readable to all other services. As all services commonly work with tabular

TABLE 6: Mappings from the DWL Operators to the Functions Implemented within the Tools for Example 2.

Conceptual Op.	Physical Op.
read	import JSON *or* read.csv
transform	remove, rename, fill down and create column, and export to CSV
enrich_timestamp	enrich date
select	filter
project	remove column *or* select
join	left_join
spatialtemporal_join	spatial and temporal join
group_by	group_by and mutate
barplot	bar_chart
write	export to CSV

data format, i.e., CSV, this became the preferred data format.

As OpenRefine generally forces its clients to download any datasets it exports, a special download service was implemented to prevent the CSV file output by OpenRefine to be transmitted to Taverna, in the middle of the execution of the task. This service interacted with the OpenRefine server to enable the exported OpenRefine project to be stored rather than sent to Taverna.

The interactions between Taverna with the three OpenRefine's functions, `fill down`, `remove column`, `rename column`, and `create column` (described in Section 4.4.1) were implemented using nested workflows configured to handle a list of parameters. As such, filling down, dropping, and renaming of multiple columns could be performed in a single interaction.

Once the CSV file exported by OpenRefine was imported by R's `read` service via the file's session key, other R services could process the file, such as R's `select`, `filter`, `groupby`, `summarize`, `mutate` and `left_join`. Each of these services produced a data session key which pointed to its respective result. Thus, a sequential interaction to each DW function in R could be performed.

The interactions between Taverna and the Python services were similar to the previously described interactions with OpenRefine and R, where parameters were firstly encoded to a HTTP-accepted format using the Beanshell component; the encoded parameters were then mapped as the request body of a REST service component; and the output of both interactions was an URL pointing to the result datasets.

While the Taverna interactions with the OpenCPU and Python servers were straightforward, the interactions with the OpenRefine server posed challenges, mainly for the following reasons: (i) OpenRefine's server specifically demanded any service request to be encoded in multiple parts; (ii) the process by which OpenRefine imports files is asynchronous and so, OpenRefine does not send a response immediately. Thus, the status of the process has to be repeatedly checked until the file is ready to be imported by the OpenRefine server. This repeated interaction was implemented using a nested workflow; (iii) similar to the file import step of the interaction, project creation in OpenRefine is also an asynchronous process, and so a similar solution had to be applied. In conclusion, extensions of the current system towards integrating new DW tools presents challenging API integration issues due to the specific execution models of each target tool's API.

5. CONCLUSIONS

In this paper we argue for the development of a set of conceptual data wrangling constructs and a flexible and extensible DW architecture, to shield end users from the complexity of low-level data wrangling APIs.

The main advantages derived from the proposed approach relate to the possibility of reusing knowledge and skills gained from the use of DW operations across a wide range of tools and the potential for creating automated optimization strategies to address data wrangling requirements. We also believe that high-level domain-specific data wrangling languages can have a positive impact in improving the productivity of data wranglers. Due to the complexity associated with the DW process, there is often the need to apply several DW tools, presenting significant challenges to the data analyst. To address these challenges, the proposed architecture combines advantages from existing DW approaches by providing abstract and domain-specific DW constructs minimizing the need for end users to learn low-level programming APIs. The proposed approach allows data analysts to take advantage of the functionalities available in existing tools whilst focusing at the conceptual aspects of the data handling task.

5.1. Limitations and Future Work

The approach discussed in this paper needs to be underpinned by the development of a comprehensive set of operators at the conceptual operator layer and only a small range of examples were included in the case study to show the viability of the approach. Both the conceptual operator language (DWL) and the high-level DSL (D^2WL) need to be extended to incorporate additional conceptual data wrangling operators, high-level constructs to capture traffic domain requirements and, potentially, constructs that demonstrate the viability of the approach in other domains.

A comprehensive language design specification (extending the snapshot provided in Figure 11) will be included in future work and also a formal description of DWL and D^2WL and their semantics. Optimization of the target data wrangling execution recipes (data wrangling execution plans) is also a challenging problem to be investigated as part of future work. To be able to

generate optimization strategies, we also need to develop formalizations of the properties of the conceptual operators to ensure soundness of the translation into equivalent optimized data wrangling recipes.

Finally, we also plan to perform extensive end-user evaluation comparing and contrasting the proposed constructs in relation to the wrangling tools covered in the literature review. From our experience working with traffic data analysts and data scientists, we have realized that some data wrangling operations may be more effectively performed via direct manipulation of visual constructs supported by user interfaces, and that the inherent complexity involved in applying the operations may require expert system support incorporated into DW tools. These are two important future research directions requiring further exploration and empirical studies involving data scientists and casual users of data wrangling tools.

For a more comprehensive assessment of the proposed approach, we also plan to perform a number of end-user experiments towards measuring productivity and end-user ability to express requests using (D^2WL). User productivity will be measured based on ability to tackle DW tasks of varying levels of complexity, time, number of steps required to complete the tasks, and overall user acceptance of the proposed approach.

ACKNOWLEDGEMENT

The second author was supported by the Government of the Kingdom of Saudi Arabia.

REFERENCES

[1] Lopes, J., Bento, J., Huang, E., Antoniou, C., and Ben-Akiva, M. (2010) Traffic and mobility data collection for real-time applications. *13th International IEEE Conference on Intelligent Transportation Systems, Funchal, Portugal*, 19-22 Sept, pp. 216–223. IEEE, New York.

[2] Hutchins, J., Ihler, A., and Smyth, P. (2008) Probabilistic analysis of a large-scale urban traffic sensor data set. In Gaber, M. M., Vatsavai, R. R., Omitaomu, O. A., Gama, J., Chawla, N. V., and Ganguly, A. R. (eds.), *Knowledge Discovery from Sensor Data*, Las Vegas, NV, USA, August 24-27, pp. 94–114. Springer, Berlin Heidelberg.

[3] Jagadish, H., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. (2014) Big data and its technical challenges. *Communications of the ACM*, **57**, 86–94.

[4] Furche, T., Gottlob, G., Libkin, L., Orsi, G., and Paton, N. (2016) Data wrangling for big data: Challenges and opportunities. *Advances in Database Technology EDBT 2016*, Bordeaux, France, March 15-18 Advances in Database Technology, pp. 473–478. University of Konstanz, Germany.

[5] Terrizzano, I., Schwarz, P. M., Roth, M., and Colino, J. E. (2015) Data wrangling: The challenging journey from the wild to the lake. *Proceedings of the CIDR Conference*, Asilomar, California, 4-7 January. www.cidrdb.org.

[6] Papazoglou, M. P. and van den Heuvel, W.-J. (2007) Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, **16**, 389–415.

[7] Weske, M. (2007) *Business Process Management - Concepts, Languages, Architectures*. Springer-Verlag Berlin Heidelberg.

[8] Guo, C., Jensen, C. S., and Yang, B. (2014) Towards total traffic awareness. *SIGMOD Record*, **43**, 18–23.

[9] Oinn, T. M., Greenwood, R. M., Addis, M., Alpdemir, M. N., Ferris, J., Glover, K. R., Goble, C. A., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P. W., Pocock, M. R., Senger, M., Stevens, R., Wipat, A., and Wroe, C. (2006) Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, **18**, 1067–1100.

[10] Abiteboul, S., Hull, R., and Vianu, V. (eds.) (1995) *Foundations of Databases: The Logical Level*, 1st edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[11] Services, E. E. (2015) *Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data*. John Wiley Sons, New Jersey, United States.

[12] Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. (2011) Wrangler: Interactive visual specification of data transformation scripts. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, BC, Canada, May 7-12 CHI '11, pp. 3363–3372. ACM, New York, NY, USA.

[13] Tye Rattenbury, J. H., Joseph M. Hellerstein and Kandel, S. (2016). Data wrangling techniques and concept for agile analytics. https://www.trifacta.com. [Online; accessed 03-09-2017].

[14] Endel, F. and Piringer, H. (2015) Data wrangling: Making data useful again. *IFAC-PapersOnLine*, **48**, 111 – 112. 8th Vienna International Conference on Mathematical Modelling.

[15] Rahm, E. and Do, H. H. (2000) Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, **23**, 3–13.

[16] Fowler, M. and Parsons, R. (2011) *Domain-specific Languages* Addison-Wesley signature series. Addison-Wesley, Boston, United States.

[17] Trifacta (2017). Trifacta wrangler. https://www.trifacta.com/. [Online; accessed 03-09-2017].

[18] OpenRefine-Google (2017). Openrefine tool. https://github.com/OpenRefine/OpenRefine/wiki/General-Refine-Expression-Language. [Online; accessed 03-09-2017].

[19] Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006) *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[20] Garey, M. R. and Johnson, D. S. (1990) *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[21] Batini, C., Lenzerini, M., and Navathe, S. B. (1986) A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, **18**, 323–364.

[22] Aristaran, M., Tigas, M., and Merrill, J. B. (2016). Tabula extract tables from pdfs. http://tabula.technology/. [Online; accessed 01-09-2017].

[23] Github (2018). Mr. data converter. https://thdoan.github.io/mr-data-converter/. [Online; accessed 28-09-2018].

[24] Cao, L. (2017) Data science: A comprehensive overview. *ACM Computing Surveys*, **50**, 1–42.

[25] Meehan, J., Aslantas, C., Zdonik, S., Tatbul, N., and Du, J. (2017) Data ingestion for the connected world. *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*.

[26] Abedjan, Z., Golab, L., and Naumann, F. (2015) Profiling relational data: A survey. *The VLDB Journal*, **24**, 557–581.

[27] Larsson, P. (2013). Evaluation of open source data cleaning tools: Open refine and data wrangler. https://courses.cs.washington.edu/courses/cse544/13-sp/final-projects/p12-plarsson.pdf. [Online; accessed 28-09-2018].

[28] Mike Tuchen, C. (2005). Talend. https://www.talend.com/why-talend. [Online; accessed 03-09-2017].

[29] Ihaka, R. and Gentleman, R. (1993). R programming language. https://www.r-project.org/. [Online; accessed 03-09-2017].

[30] Revesz, P. (2010) *Introduction to Databases: From Biological to Spatio-Temporal*, 1st edition. Springer-Verlag, London.

[31] Guo, C., Jensen, C. S., and Yang, B. (2014) Towards total traffic awareness. *ACM SIGMOD Record*, **43**, 18–23.

[32] Jacox, E. H. and Samet, H. (2007) Spatial join techniques. *ACM Transactions on Database Systems*, **32**.

[33] Aggarwal, C. C. (2013) *Outlier Analysis*. Springer-Verlag, New York.

[34] Guarino, N. (1998) *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*, 1st edition. IOS Press, Amsterdam, The Netherlands.

[35] ter Hofstede, A. H., Proper, H. A., and Van Der Weide, T. P. (1993) Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, **18**, 489–523.

[36] Aho, A. V. and Ullman, J. D. (1977) *Principles of compiler design*. Addison-Wesley, Boston, United States.

[37] Kune, R., Konugurthi, P. K., Agarwal, A., Chillarige, R. R., and Buyya, R. (2016) The anatomy of big data computing. *Software Practice and Experience*, **46**, 79–105.

[38] Abelson, H. and Sussman, G. J. (1996) *Structure and Interpretation of Computer Programs*, 2nd edition. MIT Press, Cambridge, MA, USA.

[39] Beekhuizen, B., Bod, R., and Zuidema, W. (2013) Three design principles of language: The search for parsimony in redundancy. *Language and speech*, **56**, 265–290.

[40] Codd, E. F. (1970) A relational model of data for large shared data banks. *Communications of the ACM*, **13**, 377–387.

[41] Sampaio, P. R. F. and Paton, N. W. (2000) Query processing in doql: A deductive database language for the odmg model. *Data & Knowledge Engineering*, **35**, 1–38.

[42] Mendes Sampaio, S. d. F., Dong, C., and Sampaio, P. (2015) Dq2s - a framework for data quality-aware information management. *Expert Systems with Applications*, **42**, 8304–8326.

[43] Sampaio, S. d. F. M., Paton, N. W., Smith, J., and Watson, P. (2006) Measuring and modelling the performance of a parallel odmg compliant object database server. *Concurrency and Computation: Practice and Experience*, **18**, 63–109.

[44] Hasan, W. and Krishnamurthy, R. (1993) Pil: An optimizable functional language for data intensive applications. In Albano, A. and Morrison, R. (eds.), *Persistent Object Systems*, San Miniato (Pisa), Italy, September 1-4 1992, pp. 262–279. Springer London.

[45] Graefe, G. (1996) Iterators, schedulers, and distributed-memory parallelism. *Software Practice and Experience*, **26**, 427–452.

[46] Jarke, M. and Koch, J. (1984) Query optimization in database systems. *ACM Computing Surveys*, **16**, 111–152.

[47] Raman, V. and Hellerstein, J. M. (2001) Potter's wheel: An interactive data cleaning system. *Proceedings of the 27th International Conference on Very Large Data Bases*, San Francisco, CA, USA, September 11 - 14, pp. 381–390. Morgan Kaufmann Publishers Inc., Massachusetts, United States.

[48] Lakshmanan, L. V. S., Sadri, F., and Subramanian, I. N. (1996) Schemasql - a language for interoperability in relational multi-database systems. *Proceedings of the 22th International Conference on Very Large Data Bases*, San Francisco, CA, USA, September 3-6, pp. 239–250. Morgan Kaufmann Publishers Inc., Massachusetts, United States.

[49] Lee, M.-L., Ling, T. W., Lu, H., and Ko, Y. T. (1999) Cleansing data for mining and warehousing. *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, Florence, Italy, August 30 - September 3 DEXA '99, pp. 751–760. Springer-Verlag, London, UK.

[50] Savinov, A. (2017). Data commandr – integrate, transform, analyze. http://dc.conceptoriented.com. [Online; accessed 03-09-2017].

[51] Savinov, A. (2014) Concept-oriented model. In Wang, J. (ed.), *Encyclopedia of Business Analytics and Optimization*, pp. 502–511. IGI Global, Pennsylvania, United States.

[52] Konstantinou, N., Koehler, M., Abel, E., Civili, C., Neumayr, B., Sallinger, E., Fernandes, A. A., Gottlob, G., Keane, J. A., Libkin, L., and Paton, N. W. (2017) The vada architecture for cost-effective data wrangling. *Proceedings of the 2017 ACM International Conference*

on Management of Data, Chicago, Illinois, USA, May 14-19 SIGMOD '17, pp. 1599–1602. ACM, New York, NY, USA.

[53] Ceri, S., Gottlob, G., and Tanca, L. (2012) *Logic Programming and Databases*, 1st edition. Springer-Verlag, Berlin Heidelberg.

[54] Dijkstra, E. W. (1982) On the role of scientific thought (EWD447). *Selected Writings on Computing: A Personal Perspective*, pp. 60–66. Springer-Verlag, New York, NY.

[55] Bostock, M. and Heer, J. (2009) Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, **15**, 1121–1128.

[56] Apache (2014). Taverna. https://taverna.incubator.apache.org. [Online; accessed 28-09-2018].

[57] Sehgal, V., Getoor, L., and Viechnicki, P. D. (2006) Entity resolution in geospatial data integration. *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, Arlington, Virginia, USA, November 10-11 GIS '06, pp. 83–90. ACM, New York, NY, USA.

[58] Robusto, C. C. (1957) The cosine-haversine formula. *The American Mathematical Monthly*, **64**, 38–40.

6. APPENDIX: SNAPSHOT OF D²WL'S BNF

<query specification> ::=
SELECT [<set quantifier>] <select list> <table expression>
<select list> ::=
<asterisk>
| <select sublist> [{ <comma> <select sublist> }...]
<select sublist> ::= <derived column> | <qualifier> <period> <asterisk>
<derived column> ::= <value expression> [<as clause>]
<table expression> ::=
<from clause>
<to clause>
[<where clause>]
[<wrangle by clause>]
[<group by clause>]
[<having clause>]
<from clause> ::= FROM <from sublist> [<comma> <from sublist>]
<from sublist> ::= <URL expression> [<as clause>]
<URL expression> ::= <URL syntax rules>
<as clause> ::= [AS] <column name>
<content expression> ::= <content syntax rules>
<to clause> ::= TO <URL expression>
<wrangle by clause> ::= WRANGLE BY <wrangle by subclause> [{<comma> <wrangle by subclause>}...]

<wrangle subclause> ::=

<transform expression>
|<enrichtimestamp expression>
|<join expression>
|<union expression>
|<fillmissingvalue expression>

FIGURE 11: Snapshot of D²WL's BNF.