CHAMALEON: Framework to improve Data Wrangling with Complex Data

Completed Research Paper

Álvaro Valencia-Parra
Universidad de Sevilla
Avda. Reina Mercedes S/N 41012
Sevilla - SPAIN
avalencia@us.es

Ángel Jesús Varela-Vaca
Universidad de Sevilla
Avda. Reina Mercedes S/N 41012
Sevilla - SPAIN
ajvarela@us.es

María Teresa Gómez-López
Universidad de Sevilla
Avda. Reina Mercedes S/N 41012
Sevilla - SPAIN
maytegomez@us.es

Paolo Ceravolo
Università degli Studi di Milano
Via Giovanni Celoria, 18, 20133 Milano
– ITALY
Paolo.Ceravolo@unimi.it

Abstract

Data transformation and schema conciliation are relevant topics in Industry due to the incorporation of data-intensive business processes in organizations. As the amount of data sources increases, the complexity of such data increases as well, leading to complex and nested data schemata. Nowadays, novel approaches are being employed in academia and Industry to assist non-expert users in transforming, integrating, and improving the quality of datasets (i.e., data wrangling). However, there is a lack of support for transforming semi-structured complex data. This article makes a state-of-the-art by identifying and analyzing the most relevant solutions that can be found in academia and Industry to transform this type of data. In addition, we propose a Domain-Specific Language (DSL) to support the transformation of complex data as a first approach to enhance data wrangling processes. We also develop a framework to implement the DSL and evaluate it in a real-world case study.

Keywords: Data Wrangling, Complex Data, Data Transformation, Semi-structured Data, Data Preparation

Introduction

The continuous technological advances in Industry are leading to data-driven business processes. These changes (Gerbert et al. 2015) are motivated by the concept of Industry 4.0, whose objective is to improve their production processes by means of Cyber-Physical Systems. These systems enable companies to capture real-time data on any aspect related to these productive processes. On the other hand, Industry 4.0 promotes companies to a broader integration with their external environment. Therefore, the necessity of integration of internal data with data from external sources (e.g., data from other organizations, open data, social network data) arises (Obitko and Jirkovský 2015). The ultimate objective is to process this data in order to discover knowledge, improve the decision making, and optimize production processes. Big Data technologies are an essential part in this industrial context (Gerbert et al. 2015) since data to be processed fulfill the three Big Data dimensions (a.k.a., the three V's) (Lee 2017): volume (they are massively generated), velocity (creation rates increase as Cyber-Physical Systems and IoT are included in production processes), and variety (data become more heterogeneous as the amount of data sources increase). This new context has promoted the creation of new solutions adapted to the complexity of data.

Data complexity and heterogeneity are a prominent challenge. When the amount of data and the creation rate increase drastically, data heterogeneity tends to rise as well, resulting in non-structured data models with nested and complex schemata (hereinafter, complex data). Derived from the data complexity, data integration becomes also in a prominent challenge of Big Data management (Ceravolo et al. 2018; Jin et al. 2017; Stefanowski et al. 2017), since it involves combining diversified sources in a unified view supporting data analytic or reporting procedures (Dong and Srivastava 2015). Data integration requires, therefore, several transformations to be made, changing data values and structure but keeping at the same time the validity and consistency of data or event enhancing their value. For this reason, data transformation, as a part of the data preparation process, is considered the most time-consuming stage of data analytics (Guo et al. 2011).

In traditional data warehousing, this process has been extensively analyzed, proposing integration tools for data transformation by using the Extract-Transform-Load (ETL) approaches. Regarding the complex data, various query languages can be found in industry (Beyer et al. 2011; Florescu and Fourny 2013) to facilitate the transformation, integration and querying of data sources with complex data in ETL processes. However, with the emergence of the Big Data paradigm, actors focusing on the commoditization of Big Data technologies are addressing the fast roll-out of Big Data pipelines proposing visual data flow orchestrators (Milutinovic et al. 2017) and catalogs of congruent services (Ardagna et al. 2018). The aim is the introduction of the as-a-service approach in Big Data technologies, supporting the composition of services in an easy way and offering, at the same time, a guarantee about the consistency of the proposed compositions (Ardagna et al. 2018). In this sense, data wrangling has become one of the most employed techniques to facilitate the transformation and mapping of data from a raw format into the format required by data analysis processes in Big Data context. The current trend is to provide self-service data preparation (Hellerstein et al. 2018). It points at easing the data preparation process for non-expert users through data profiling and the automation of the tasks. Therefore, this assistance comes along with user-friendly Domain-Specific Languages, user interfaces, and features for data cleaning and data quality improvement.

Nowadays, several data wrangling solutions can be found in Industry. Although some of them are able to work with complex data, they are entirely focused on a table-oriented data model, flattening data into static structures avoiding nested data. It implies that operations must be directly applied over top-level attributes (i.e., columns). This operative inevitably difficulties the transformation of complex structures, requiring nested attributes to be shift to top-level positions. Consequently, (i) the number of operations needed to transform the format and/or schema of a dataset becomes significantly high depending on the depth of the nested attributes and the target schema, and (ii) the definition of the transformation operations becomes non-easy-to-use and anti-intuitive, being far away from the shape of the target schema, and hence, being more error-prone and hindering debugging operations. In this context, the identification of languages for data transformation that support complex operations by a concise syntax is of paramount importance for a flexible handling of data sources (Arputhamary and Arockiam 2015). Moreover, linking these operations to their effects on performances, in relation to the data structure, is crucial to increase the awareness of designers about the effects of their specifications.

Trying to reduce the existing gap in complex data transformation in data wrangling, this paper pursues to cover two main aspects: (i) discuss the approaches in the industry, and the academia to support the transformation of complex data in the data wrangling and self-service data preparation fields, and; (ii) the proposition of a framework, that includes a Domain-Specific Language (DSL), to support the transformation of data with complex schema. This language aims to provide a functional way to enable users to define the target schema along with the transformations needed to reach it from the source schema, minimizing the number of operations and the complexity of the language.

The rest of the paper is structured as follows. First, two real-world case studies are presented to understand the proposal better. Then, our proposal is described, depicting the solution, including a DSL that facilitates the complex data transformation in data wrangling context. The proposed implementation is introduced, before the most relevant references related to data wrangling and complex data are discussed. Next, the related work is discussed, and then we compare our proposal with other data wrangling tools. Finally, some conclusions are drawn.

Case Studies

This section describes two case studies which demonstrate the applicability of our proposal in two different industries that require the transformation of datasets: (A) the transformation of datasets from several electricity wholesales to extract information about the consumption of electricity, and (B) the transformation of datasets taken from several IoT sensors to detect potential deficiencies in aircraft assembly processes.

Case Study A: Formatting datasets from several electricity wholesales

In order to introduce the necessity to facilitate the complex data transformation, we use a real case study based on the integration datasets provided by seven electricity companies that sell energy for private customers in Spain. The electricity wholesales describe consumption data in different formats and using different frequency of meter reading, depending on factors such as the distributor or the tariff hired for each customer. These various formats need to be uniform in order to be processed and analyzed later, such as to create patterns of behavior or to look for the best tariff for each customer (Parody et al. 2017). However, each electricity provider offers information using different nested schemata, depending on some factors, such as the number of months included in the meter reading, number of days, types and tariff. Therefore, all these heterogeneous schemata need to be transformed into a unified one being possible to integrate every dataset in a unified view. The quantity of information, the heterogeneity and the updating of the information, Big Data infrastructure must be used to facilitate the data analysis.

Figure 1 illustrates the scenario where several data sources must be conciliated into a unified format accessible to the final user by means of a set of transformation, each one applicable to a single data source.

As mentioned above, the provided information does not follow the same schema, but they generally share a customer ID, a tariff identifier, the contracted power for each daily billing period, and a list of consumption over a period (e.g., twelve months or more). Each consumption period keeps information on the start and end date for that period, and the power consumption for each daily billing period. Figure 2 shows a possible input schema for the data of the example and its relationships with the target schema.

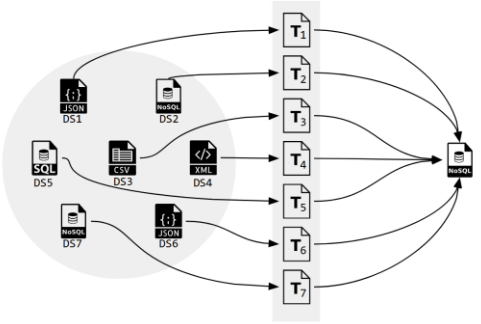


Figure 1. Case Study scenario

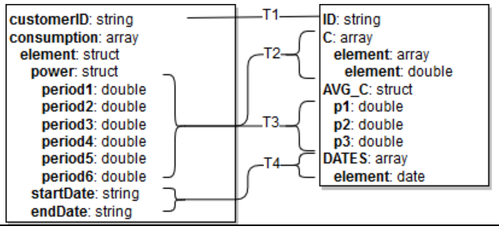


Figure 2. Transformations to convert the source schema (left) into the target schema (right)

Source Schema Description

The source schema is composed of basic and nested attributes. The description of the dataset attributes is given as follows:

- `customerID`. It is a string which identifies a unique customer supply point.
- `consumption`. It is the power consumption over a period, such as twelve months or more. It is an array of data structures. Each element represents a period, and it includes the following information:
    - `power`. It is the power consumption for each daily billing period. It is represented by a data structure with six decimal numeric attributes, each one representing the consumed power for a daily billing period.
    - `startDate`. It is the start date for the billing period represented by this element.
    - `endDate`. It is the end date for the billing period represented by this element.

Transformations and Target Schema

In order to reach the target schema, several transformations must be applied for each electricity supplier. For the example, six transformations are needed as depicted in Figure 2:

- **T1**. `customerID` must be renamed to `ID`. No further transformations are required.
- **T2**. `consumption` is transformed into a matrix `C`, whose rows have three elements that are calculated from the p_i attributes in `power` in accordance with the defined rules by the government.
- **T3**. In the target schema, `AVG_C` is a data structure with three elements: `p1`, `p2`, and `p3`. It represents the average consumed power for each daily billing period. The calculation is carried out by means of the matrix calculated in **T2**.
- **T4**. In the target schema, `DATES` is an array whose attributes are of type `date`. It is calculated by means of the `startDate` attribute in the source schema.

Case Study B: Detecting deficient aircraft

This case study is based on the aeronautic industry. It is a real case study from the aircraft factory of Airbus placed in Seville. The datasets represent the logs of an aircraft production plant. It is about the tests that are performed in the workstations where the aircraft are tested, and the incidents that occurred during these. Each workstation produces a dataset with a different schema and data format but they all have a set of attributes related to the aircraft, the workstation and the incidents occurred during the tests, as detailed in in (Valencia-Parra et al. 2019). It is required to wrangle this data in order to obtain a formatted dataset so that it helps experts to discover potential deficient aircraft.

Source Schema Description

The source schema is composed of the following attributes:

- `accode`. It is a string attribute representing the code of an aircraft.
- `workstation`. It is a string identifying the workstation where the tests have been executed.
- `incidents`. It is an array whose elements represent information about the incidents that have occurred during the test execution. It contains nested structures with the following attributes:
    - `start_date`. It is a string representing the date when the incidence started.
    - `resolution_date`. It is a string representing the date when the incidence was resolved.

Transformations and Target Schema

The following transformations are required in order to reach the target schema:

- **T1** and **T2**. Attributes `accode` and `workstation` are renamed as `aircraft` and `ws`, respectively.

- **T3**. `avg_incidents` is a numeric attribute created by calculating the average value of the array which results from iterating `incidents` and performing the following operation: `resolution_date - start_date`. It represents the average time that the incidents took to be solved.

The objective of these transformations is to produce a dataset that can be processed by an algorithm that detects abnormalities during the test of the aircrafts in the workstations, and thus, to detect potential deficient aircraft.

Our Proposal

This section depicts CHAMELEON: the proposal we have devised to improve the data wrangling processes when dealing with complex data. It consists of framework and a Domain-Specific Language (DSL), whose objective is to link the operations with their effects in the target schema. First, the necessary concepts to understand the proposal are presented. Then, Framework is introduced, following with the proposed DSL. Finally, the case studies are solved by using the proposal.

Related Concepts

Next, the concepts *Data Schema*, *data type*, and *Transformation Function* are defined. These concepts will facilitate the understanding of the DSL that is defined in this section.

Definition. *A Data Schema is a set of attributes, {a_1: t_1, a_2: t_2, ..., a_n: t_n}, identified by a name (a_i) and a data type (t_i).*

Regarding the data type (t_i), two categories of data types have been identified:

- **Simple type**. It is a data type which represents a single value:
  - `Numeric`. It represents a numeric data type, i.e., Integer, Long, Float, and Double.
  - `String`. It is a sequence of characters.
  - `Boolean`. It is a two-valued data type which represents the truth values.
  - `Date`. It is a set of characters with a specific format that represents an instant of time.
- **Complex type**. It is a composite data type that can be:
  - `Array`. It is a collection of typed attributes identified with a unique numeric index.
  - `Struct`. It is a data type composed of a set of attributes, each one identified by a unique name.

Definition. *A Transformation Function, f_x, is a function that receives an attribute, a_{input}, and returns an attribute, a_{output}, resulting from applying an operation which modifies the value of a_{input}.*

$$f_x: a_{input} \rightarrow a_{output}$$

Framework Modeling

We have developed a framework[1] to implement the DSL so that we can solve the case studies in a real-world environment. The framework has been designed according to the composite design pattern (Riehle et al. 1997). In short, this pattern enables to build complex objects by using simpler ones. It means that an object could be composed of nested objects. Figure 3 depicts a schema of this pattern. As can be seen, the classes `Composite1` and `Composite2` are composed of a set of `Components`, which can be `Composite1`, `Composite2`, or `Leaf`. The latter is called `Leaf` because it is not compounded by any other `Component`. In this pattern, the instances of objects could be represented as a tree structure.

Figure 4 depicts the UML diagram of the transformation framework. As mentioned above, the instances of this model can be represented as a tree structure. In this structure, the leaves are operations that access the attributes, and internal nodes are intended to transform or create new structures. To better understand it,

[1] The implementation can be found in: http://www.idea.us.es/datatransformation/

Figure 5 shows an instance of the transformation T1 exposed in the case study A, and Figure 6 shows its tree representation.

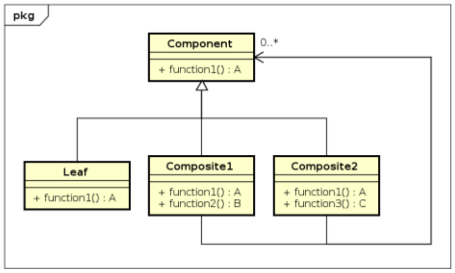


Figure 3. Composite design pattern

In this model, the `Component` is the `Evaluable` interface. An `Evaluable` represents an expression whose main objective is to perform transformation functions on attributes. Two methods can be applied over every `Evaluable` expression (hereinafter, expression): `getValue` and `getDataType`.

- `getValue`. It receives an attribute and returns another attribute as a result of applying a transformation to it.
- `getDataType`. It receives a data type and returns the data type as a result of applying a transformation to it.

These are intended to be the entry-point of the framework. The way these functions work depends on the `Leaf` or the `Composite` components. Next, the leaves of the transformation framework model are listed.

- `Select`. It is meant to select the attribute whose name matches the string `name` from an attribute of type `struct`.
- `Index`. It is meant to select the attribute whose position matches the integer `index` from an attribute of type `array`.

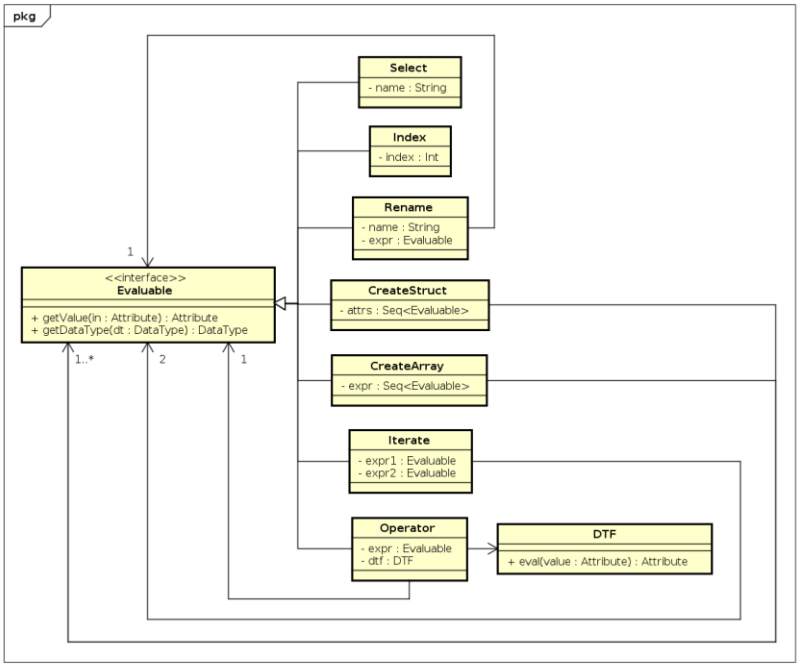
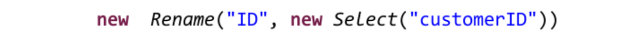


Figure 4. Framework modeling



```
new  Rename("ID", new Select("customerID"))
```

Figure 5. Instance of the model to perform the transformation T1. Code representation

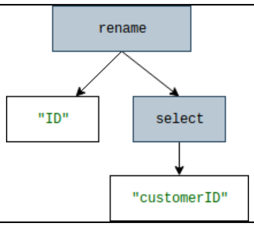


Figure 6. Instance of the model to perform the transformation T1. Tree representation.

Lastly, the `Composites` of the transformation framework model are listed.

- `Rename`. It is meant to transform an `Evaluable` expression (hereinafter, expression) which returns an attribute of any type by replacing its name by the string `name`.
- `CreateStruct`. It is meant to create an attribute of type `struct` from a set of expressions `attrs`.
- `CreateArray`. It is meant to create an attribute of type `array` from a set of expressions `attrs`.
- `Iterate`. It is meant to create an attribute of type `array` resulting from iterating over an expression which returns an attribute of type `array` (`expr1`). An expression (`expr2`) is applied to each element in that `array`.
- `Operator`. It is meant to transform an expression by applying a Data Transformation Function (hereinafter, `DTF`).
- `DTF`. It is meant to apply a transformation function to an expression. These enable users to perform advanced transformations on attributes of any data type.

DSL Definition

The DSL has been defined with two main goals: (i) provide versatility so that a wide range of transformations can be carried out, and (ii) reduce the gap between the definition of the transformations and their effects in the target schemas. The syntax of the grammar is given bellow by means of Extended Backus-Naur form notation (Reilly et al. 2003).

`Syntax` is the entry-point to the language. It is given by an `Expression`, which might be one of the following: `Select`, `Index`, `Rename`, `CreateStruct`, `CreateArray`, `Iterate`, or `Transform`.

```
Syntax ::= Expression
Expression ::= Select|Index|Rename|CreateStruct|CreateArray|Iterate|Transform
```

`Select` is intended to be the syntax for selecting either an attribute in a `struct` or a position in an `array`. For instance, regarding the case study A, `t"customerID"` selects the attribute `customerID`, and `t"consumption.[0]"`, selects the position `0` of the `consumption` array.

```
Select ::= 't' '"' ( StringLiteral | '[' Digit+ ']' )
            ( '.' ( StringLiteral | '[' Digit+ ']' ) )* '"'
```

`Rename` is meant to modify the name of an attribute. For example, `"ID" << t"customerID"` changes the name of the attribute `customerID` to `ID`.

```
Rename ::= StringLiteral '<<' Expression
```

`CreateStruct` and `CreateArray` are intended to create a `struct` or an, respectively. For example, `struct("ID" << t"customerID")` creates a `struct` composed of an attribute, `ID`, with the value of the `customerID` attribute. On the other hand, `array(t"customerID")` creates an `array`, with just one position, which contains the value of the `customerID` attribute.

```
CreateStruct ::= 'struct' ' (' Expression ( ',' Expression )* ')'
CreateArray ::= 'array' '(' Expression ( ',' Expression )* ')'
```

`Iterate` enables to perform an operation over an `array` attribute. For instance, `t"consumption" iterate t"startDate"` creates an `array` whose elements are `string` resulting from iterating over the `consumption` attribute and selecting the attribute `startDate`.

```
Iterate ::= Expression 'iterate' Expression
```

`Transform` enables to apply a transformation function to an expression. We have defined nine transformation functions which fit our case study A, but more functions might be added. Regarding the ones defined in `DTF`, they can be classified in two groups: (i) `max`, `min`, `avg`, `sum`, and `subtract`, which are intended to return the maximum, minimum, the average, sum or the subtraction of the values of an attribute, `array` respectively; (ii) `toInt`, `toDouble`, `toString`, and `toDate`, which are intended to cast an attribute to `integer`, `double`, `string`, or `date`, respectively.

```
Transform ::= Expression '->' DTF
DTF ::= 'max'|'min'|'avg'|'sum'|'subtract'|'toInt’|'toDouble'|'toString'|
        'toDate(' StringLiteral')'
```

Transformationss for the Case Study A

Next, the transformations shown in the case study A are performed by means of the DSL we have proposed.

- **T1**. `customerID` is renamed to `ID` by using the `'<<'` operator.

```
"ID" << t"customerID"
```

- **T2**. The matrix `C` is created by iterating over each position in the `consumption array`.

```
"C" << ( t"consumption" iterate array(
            array(t"power.period1", t"power.period4") -> max,
            array(t"power.period2", t"power.period5") -> max,
            array(t"power.period3", t"power.period6") -> max,
        ))
```

- **T3**. Each attribute of `AVG_C` is created by means of the `C` attribute previously created. Each `iterate` operation will result in an `array` with all the values in one of the columns. Then, the average value of each `array` is calculated.

```
"AVG_C" << struct(
             "p1" << (t"C" iterate t"[0]") -> avg,
             "p2" << (t"C" iterate t"[1]") -> avg,
             "p3" << (t"C" iterate t"[2]") -> avg,
            ))
```

- **T4**. `DATES` is created by employing the operator `iterate` and by applying the `toDate` transformation function over the `t"startDate"` attribute.

```
"DATES" << t"consumption" iterate (t"startDate" -> toDate("mm/dd/yyyy"))
```

Transformations for the Case Study B

The transformations for the case study B performed by means of CHAMELEON are as follows.

- **T1** and **T2**. The attributes `accode` and `workstation` are included in the target dataset.

```
                                   "accode" << t"aircraft"
                                    "workstation" << t"ws"
```

- **T3**. `avg_incidents` is created by iterating over each structure of it. Then, we subtract the resolution date and the start date. The date is previously transformed by using the `toDate` function. Finally, the average of all the subtractions is calculated.

```
    "avg_incidents" << (t"incidents" iterate substract(
              t"resolution_date" -> toDate("MM/dd/yyyy HH:mm:ss"),
              t"start_date" -> toDate("MM/dd/yyyy HH:mm:ss")
    )) -> avg
```

Benchmark

A set of tests has been devised to evaluate and check the performance of the framework that we have implemented in a Big Data environment. First, the Big Data architecture used to perform the tests is presented. Afterward, we describe the evaluation design to test the performance of our proposal. Finally, the results are drawn and discussed.

Architecture and Implementation

The architecture employed to perform the benchmark is based on a cluster managed by Mesosphere DC/OS (hereinafter DC/OS). DC/OS is an operating system based on Apache Mesos, which enables the execution of technologies for simultaneous data processing. In this case, an Apache Spark cluster has been deployed together with Spark History Server, that enables to extract execution metrics of the Apache Spark applications.

Regarding the infrastructure, it consists of a DC/OS `master` node, responsible for managing the cluster resources and assign them to services, and nine `agents`, responsible for managing the services. The instance of Spark includes a `driver` and nine `executors`. The architecture also includes a node with HDFS to store the datasets and a MongoDB database for storing the execution results. Regarding the computational characteristics, the cluster can reach fifty-two cores between 2 and 2,6 GHz for each and 136 gigabytes of RAM in global. Figure 7 depicts the infrastructure as well as the computational characteristics of the cluster. Summing up, the cluster can reach fifty-two cores and 136 gigabytes of principal memory in global.

Evaluation Design

We have selected the case study A to perform the benchmark, since its schema and transformations are more complex than those of the case study B, so the results will be more reliable. This dataset is composed of approximately more than five million tuples and a size of 2,1 GB. In order to test the scalability of the proposal, nine additional datasets have been created based on two different criteria: (A) four new datasets by increasing the number of tuples; and (B) five new datasets by increasing the size of each tuple (i.e., by increasing the size of the columns), for instance, by duplicating the number of elements in the `consumption` array attribute. Table 1 summarizes the datasets which have been synthetically created by using these two criteria.

Ten test cases have been defined, each of them being executed one hundred times. These tests cases have been classified into two groups of benchmarks: (i) Benchmark 1, where these test cases are intended to check the performance when the dataset size increases by the criteria A; and (ii) the Benchmark 2, where these test cases are intended to check the performance when the dataset size increases by the criteria B. In each test case, all transformations described in the case study A have been applied for each tuple of the dataset.

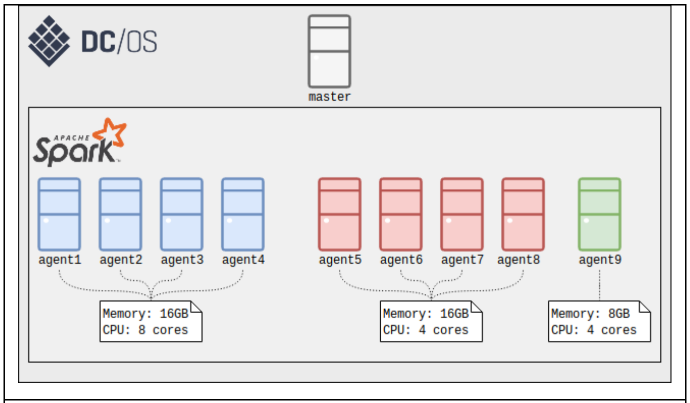


Figure 7. Architecture of the cluster

Table 1. Evaluation Design

Dataset ID	Criteria	Size (MB)	Benchmark
D1	A	4,119.4	1
D2	A	6,178.4	1
D3	A	8,239.9	1
D4	A	10,299.9	1
D5	B	3,659.8	2
D6	B	5,258.9	2
D7	B	6,859.4	2
D8	B	8,459.2	2
D9	B	10,060.3	2

Table 1. Dataset and Benchmarks for the evaluation

Both benchmarks have been developed by using an Apache Spark application. The application consists of two main stages. The first reads the dataset from HDFS and infers its schema, and the latter distributes the tuples across the cluster, applies the transformations and finally stores the results in MongoDB. As for performance metrics, both the Elapsed Real Time ERT and the CPU Time of the second stage have been measured in each test case. The ERT is the execution time since the stage corresponding to the application of the transformations is launched until it ends. On the other hand, the CPU Time is a time accumulator that includes the time the tasks related to the transformations spent on the CPU. For each test case, the average value of one hundred executions will be considered.

Evaluation Results

The results for both benchmarks have been depicted in Figure 8. A trend line has been included in the charts in order to highlight the tendency of the results. The least-squares fitting method has been employed to calculate the trend lines.

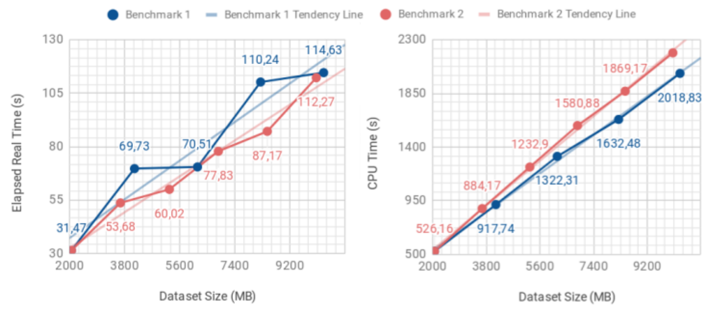


Figure 8. Comparison between Elapsed Real Time (left) and CPU Time (right)

The chart on the left side shows the `ERT` comparison between both benchmarks. The `ERT` tends to be higher in the case of the Benchmark 1. It means that for datasets with the same size, the `ERT` is greater for datasets with more tuples to process than for datasets with less but more complex tuples. This is because the distribution cost is higher when processing a higher number of tuples.

The right-side chart shows the `CPU Time` comparison between both benchmarks. Unlike in the case of the `ERT`, the trend line of the Benchmark 2 tends to be higher than the Benchmark 1.

The trend line equations are (1) for Benchmark 1 whilst for Benchmark 2 it is (2). In fact, the complexity of the dataset used for the Benchmark 2 is higher as its tuples are larger than in Benchmark 1, consuming each one more CPU time. Despite this fact, there is only a 15% of the difference between the slope of the trend lines, being both lines under linear.

$$y = 0.18x + 174 \quad (1) \qquad\qquad y = 0.21x + 122 \quad (2)$$

As a conclusion, it is possible to confirm that the proposal scales regarding the dataset complexity because the increment on the complexity on processing nested structures and hence the transformations to apply, only suppose a 15% regarding processing a dataset with less-complex structures.

Related Work

The transformation and querying of complex data have been studied in the fields of database, data warehousing and Big Data. This section describes the state-of-the-art of the transformation of complex data in the ETL and data wrangling fields.

Traditional Approach

Traditionally, the transformation and combination of data sources have been faced up by means of ETL techniques (Arputhamary and Arockiam 2015) in the databases and data warehousing fields. Then, there exist query languages which enable the extraction of complex data. Nevertheless, these transformations are normally meant to give support to a query, and hence, the target schema is not the focus, harvesting the task of defining a specific schema due to the kind of syntax which are typically employed in query languages. The objective of querying is to answer a question on a dataset, and therefore, they usually do not support operators for renaming, restructuring, or transforming complex structures into another complex structure.

Following, four popular query languages with complex data support (Ong et al. 2014) are analyzed. We assess the versatility of the languages, i.e., the operation and transformation capabilities with nested attributes, and the degree in which the query language is aligned with the target schema (i.e., the degree in which the syntax eases the possibility of linking the operations to their effects on performances in relation to the data structure). Both criteria are focused on evaluating the suitability of these languages in a self-service data preparation context.

- **MongoDB**. It is a well-known NoSQL database specialized in semi-structured data. Since its data model enables to deal with complex data, its query language (Botoeva et al. 2018) also supports querying such data, providing operators and function to access nested attributes and operate between them. However, this query language has significant limitations in the definition of custom nested structures, and hence, this language is not aligned with the target schema.
- **N1QL**. It is the query language of Couchbase (Ostrovsky et al. 2015), a NoSQL database. This is a SQL-like language, adding operators and functions to operate with columns with nested structures. Due to its orientation toward SQL, the syntax of the language is not aligned with the definition of the target schema.
- **JSONiq**. It is a JSON-oriented scripting query language (Florescu and Fourny 2013), based on a JSON-like data model. Although this language offers great versatility, the definition of the queries is not aligned with the target schema and the input data type is restricted to JSON.
- **Jaql**. This query language (Beyer et al. 2011) is intended to query semi-structured data in Hadoop. As JSONiq, it is also a scripting language, offering good versatility and hence supporting operators to perform a wide range of transformations. In addition, the language enables to align its queries to the target schema.

While the most versatile languages are JSONiq and Jaql, the one with the best syntax for our purpose is Jaql. However, we find areas of improvement in relation to the complexity and the syntax. First, the great versatility of this language leads to a complexity that, in our opinion, can be decreased. Second, although the flexibility of the syntax enables users to align the transformations with the effects in the target schema, we think that it is possible to achieve a better alignment between them.

Data Wrangling

In recent years, Academia and Industry have used the term data wrangling to refer to the transformation, combination and cleaning of data in an exploratory way (Furche et al. 2016). The current trend is to make this task easier so that it can be carried out by non-expert users. Consequently, the latest proposals that can be found in the literature are focused on assisting users in this process (a.k.a. self-service), offering features such as (i) data profiling, so that the user can explore the data and obtain a holistic view of them (e.g., see the quality of them at first glance); (ii) suggest transformations based on a knowledge base, or based on criteria that may improve the quality of the data; and (iii) automatically infer transformations by means of example process results.

Although there exist data wrangling solutions in Industry, just a few of them support obtaining data from semi-structured data sources, transforming them, and exporting the dataset with complex data structures. In the study we have carried out, we have considered two of the most influential tools in academia and in the industry that enable to deal with complex data: Trifacta (a.k.a. Google Cloud Dataprep) and OpenRefine.

- **Trifacta**. It was originated in Academia, conceived as a visual data wrangling tool with a transformation language known as data-wrangler (Kandel et al. 2011). Now, it is a commercial tool, and one of the references in the self-service data preparation field. In relation to the transformation of complex data, it supports nesting and unnesting operators.
- **OpenRefine**. It was originally maintained by Google (Kusumasari and Fitria 2016). Now, it is an open source tool that has been employed in multiple research works. It provides a query language known as GREL (Google Refine Expression Language). It is a Java Script-based language which enables the transformation of complex data.

First, the Trifacta data model is table-oriented. For this reason, transformation operations are carried out on columns. This implies that in data with nested structures, only those that are in the top-level can be operated. This has two major drawbacks: (i) Data profiling functionalities, data quality analysis, and transformation suggestions do not reach those attributes that are nested; and (ii) in order to perform any type of operation between attributes that are within the same nested structure, it is required to perform as many unnesting operations as how deep is the attribute to be employed. This inevitably increases the complexity of the transformations and the ease of making mistakes. OpenRefine poses similar problems. In this case, users must employ a Java Script-based language to perform the operations in a programmatic way. Although it offers a good versatility, it does not fulfill the criteria we defined above, being far from the objectives of the self-service data preparation. In addition to these drawbacks, there is a lack of support for data profiling, data quality, and transformation suggestions and inferences based on complex data.

To the best of our knowledge, our proposal is the first attempt to contribute to the inclusion of semi-structured complex data in the data wrangling and self-service data preparation fields.

Comparison with Data Wrangling Tools

The objective of this section is to evidence the drawbacks of the current data wrangling tools when dealing with complex data. We also point at how our proposal could improve the transformation of complex data from the point of view of the user.

In the Related Work section, we considered two of the most influential data wrangling tools in the Industry and Academy (Trifacta and OpenRefine). However, Trifacta includes and improves the functionalities offered by the other proposals. Therefore, the analysis of Trifacta implies the analysis of the functionalities of their competitors, since Trifacta is the more complete tool in data wrangling context. On the one hand, Trifacta's data model is table-oriented, being unable to represent data with more than one dimensionality (i.e., nested structures). Despite this, it is possible to transform nested attributes as well as nesting other attributes, but it will require the user to flatten arrays, or unnest attributes by creating new columns. On the other hand, OpenRefine is also a table-oriented solution, but it is unable to nest attributes, making it impossible to create columns of data type struct. To work around it, OpenRefine allows users to employ an imperative Java Script-based language, as explained in the Related Work section. Since it is an imperative language, it is out of our scope. For this reason, and since it is one of the most complete data wrangling tools in Industry, we have selected Trifacta to show how our proposal could improve the transformation of complex data.

In order to show the comparison, we have resolved the cases studies presented in this paper by using Trifacta. Then, for each case study we have created a table which depicts the number of operations which are necessary to complete each single transformation, classified by type. The types of operations that we have considered are: (i) **unnesting**, which consists of extracting nested attributes in columns of type struct, resulting in an augmentation in the number of columns; (ii) **nesting**, which joins several columns in a single struct column; (iii) **flattening**, which is applied to columns of type array, resulting in an augmentation in the number of rows; (iv) **grouping**, which reduces the number of rows by grouping them according to a criteria; (v) **dropping**, which deletes a set of columns specified by the user; (vi) **renaming**, that includes those renaming operations that must be performed due to column name changes that might occur during intermediate operations, and (vii) **non-intermediate**, that includes simple operations such as column renaming, data formatting, arithmetic operations and operations with lists of numbers.

The most important aspect here is that in order to access nested attributes in a data schema, it is required to isolate those attributes in single columns, and once they have been transformed, they must be sent back to their corresponding nested structure. We consider that these intermediate operations complicate the transformation process of complex data. To illustrate this problem, Figure 9 represents the sequence of operations that are required in order to carry out the transformation T2 of case study A with Trifacta. As can be seen, in order to access to the attributes inside `consumption.power`, two intermediate operations are required (flattening and unnesting). Then, three additional operations (three nesting) are required before calculating the maximum value for each period, and finally, in order to create the `C` matrix, two additional intermediate operations are required (nesting and grouping). In addition to these intermediate operations, three operations (two dropping and one renaming) must be performed in order to deal with temporary columns and column names that are generated during the transformation process. In total, T2 requires 13 operations, where 10 of them are intermediate operations, which means that the 77% of the operations that the user must perform to carry out this transformation are intermediate operations needed to access the attributes and to give them the right structure. The objective of our proposal is to abstract users from these intermediate operations so that they just have to navigate through the structure in order to operate with the desired attributes, and at the same time they can change the structure of the dataset. We believe that, in this way, the transformations are better aligned with their results in the target schema, making the transformation process more intuitive and in concordance with the objectives of data wrangling.

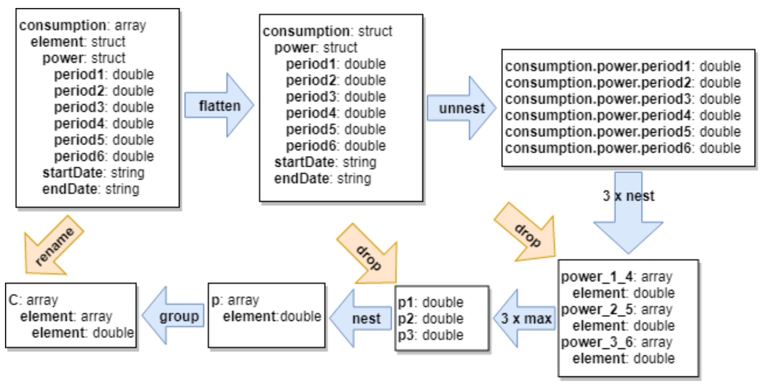


Figure 9. Representation of the intermediate steps needed for the transformation T2 in case study A

Case Study A

Table 2 shows the operations performed to complete the case study A with Trifacta. This case study requires 34 operations, where 26 of them are intermediate operations and 8 of them are not. Approximately the 77% of the operations written by the user are intermediate. Transformations T2 and T3 have a similar proportion of intermediate operations, while in T4 this percentage increases to the 85%. Our proposal abstracts users from these intermediate operations, so we can state that, except for T1 (that does not require to transform nested attributes), it improves the way the transformations are carried out.

Table 2. Operations to complete Case Study A with Trifacta

	Unnesting	Nesting	Flattening	Grouping	Dropping	Renaming	Non-intermediate	Total
T1	0	0	0	0	0	0	1	1
T2	1	4	1	1	2	1	3	13
T3	1	4	1	1	2	1	3	13
T4	1	1	1	1	1	1	1	7
TOTAL	3	9	3	3	5	3	8	34

Table 2. Operations to complete Case Study A with Trifacta

Case Study B

Table 3 shows the operations performed to complete case study B with Trifacta. The most complex transformation here is T3, which is the only one that requires access to nested attributes. It requires a total of 5 operations, being 4 of them intermediate operations. Hence, the 80% of the operations for T3 are intermediate operations. Ultimately, the 57% of the operations of the case study are intermediate. Our proposal also improves the transformation process for this case study.

Table 3. Operations to complete Case Study B with Trifacta

	Unnesting	Nesting	Flattening	Grouping	Dropping	Renaming	Non-intermediate	Total
T1	0	0	0	0	0	0	1	1
T2	0	0	0	0	0	0	1	1
T3	1	0	1	1	0	1	1	5
TOTAL	1	0	1	1	0	1	3	7

Table 3. Operations to complete Case Study B with Trifacta

Conclusions

One of the most important challenges that last advances in Industry poses is the transformation of complex data and the conciliation of complex data schemata. There is an emerging field in Big Data which intends to ease these tasks for non-expert users: data wrangling and self-service data preparation. This study has highlighted the lack of support for complex data in these fields.

Our proposal is intended to contribute to the improvement of data wrangling techniques by means of a framework that includes a Domain-Specific Language. The goal of it is to link the operations carried out by users to their effects in the target schema. It represents an improvement in relation to the data wrangling solutions that can be found in Industry since they are not focused on dealing with complex data. Several well-known query languages for semi-structured data have been studied to enhance our proposal.

Future Work

The limitations of our proposal define the actions that we want to face up for the future. First, our proposal lacks support for data profiling, data quality assessment, and automatic assistance to users. These shortcomings are hence a great opportunity for the future, since as proven in this study, data wrangling and self-service data preparation are of paramount relevance in both academia and Industry.

In particular, we identify prior opportunities (Furche et al. 2016) in the automation of error-detection and feedback giving in the definition of transformation rules by users. It would be a pioneering proposal in the field of semi-structured complex data.

Acknowledgments

This work has been partially funded by the Ministry of Science and Technology of Spain by ECLIPSE project (RTI2018-094283-B-C33) and the European Regional Development Fund (ERDF/FEDER) via METAMORFOSIS project.

References

Ardagna, C. A., Bellandi, V., Bezzi, M., Ceravolo, P., Damiani, E., and Hebert, C. 2018. "Model-Based Big Data Analytics-as-a-Service: Take Big Data to the Next Level," *IEEE Transactions on Services Computing*, pp. 1–1. (https://doi.org/10.1109/TSC.2018.2816941).

Ardagna, C. A., Bellandi, V., Ceravolo, P., Damiani, E., Di Martino, B., D'Angelo, S., and Esposito, A. 2018. "A Fast and Incremental Development Life Cycle for Data Analytics as a Service," in *2018 IEEE International Congress on Big Data (BigData Congress)*, IEEE, July, pp. 174–181. (https://doi.org/10.1109/BigDataCongress.2018.00030).

Arputhamary, B., and Arockiam, L. 2015. "Data Integration in Big Data Environment," *Bonfring International Journal of Data Mining* (5:1), Bonfring, pp. 01–05. (https://doi.org/10.9756/BIJDM.8001).

Beyer, K. S., Ercegovac, V., Gemulla, R., Balmin, A., Eltabakh, M. Y., Kanne, C.-C., Özcan, F., and Shekita,

E. J. 2011. "Jaql: A Scripting Language for Large Scale Semistructured Data Analysis," in *Proceedings of VLDB Conference*. (https://www.semanticscholar.org/paper/Jaql%3A-A-Scripting-Language-for-Large-Scale-Data-Beyer-Ercegovac/28d0280e2b972155c203b96bd6eb9f826aa73850).

Botoeva, E., Calvanese, D., Cogrel, B., and Xiao, G. 2018. "Expressivity and Complexity of MongoDB Queries," *DROPS-IDN/8607*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. (https://doi.org/10.4230/LIPICS.ICDT.2018.9).

Ceravolo, P., Azzini, A., Angelini, M., Catarci, T., Cudré-Mauroux, P., Damiani, E., Mazak, A., Van Keulen, M., Jarrar, M., Santucci, G., Sattler, K. U., Scannapieco, M., Wimmer, M., Wrembel, R., and Zaraket, F. 2018. "Big Data Semantics," *Journal on Data Semantics* (7:2), Springer Berlin Heidelberg, pp. 65–85. (https://doi.org/10.1007/s13740-018-0086-2).

Dong, X. L., and Srivastava, D. 2015. "Big Data Integration," *Synthesis Lectures on Data Management* (7:1), Morgan & Claypool Publishers, pp. 1–198. (https://doi.org/10.2200/S00578ED1V01Y201404DTM040).

Florescu, D., and Fourny, G. 2013. "JSONiq: The History of a Query Language," *IEEE Internet Computing* (17:5), pp. 86–90. (https://doi.org/10.1109/MIC.2013.97).

Furche, T., Gottlob, G., Libkin, L., Orsi, G., and Paton, N. W. 2016. "Data Wrangling for Big Data: Challenges and Opportunities," in *19th International Conference on Extending Database Technology (EDBT)*, pp. 473–478. (https://doi.org/10.5441/002/EDBT.2016.44).

Gerbert, P., Lorenz, M., Rüßmann, M., Waldner, M., Justus, J., Engel, P., and Harnisch, M. 2015. "Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries," *Boston Consulting Group*. (https://www.bcg.com/publications/2015/engineered_products_project_business_industry_4_future_productivity_growth_manufacturing_industries.aspx, accessed December 19, 2018).

Guo, P. J., Kandel, S., Hellerstein, J. M., and Heer, J. 2011. "Proactive Wrangling: Mixed-Initiative End-User Programming of Data Transformation Scripts," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology - UIST '11*, New York, New York, USA: ACM Press, p. 65. (https://doi.org/10.1145/2047196.2047205).

Hellerstein, J. M., Heer, J., and Kandel, S. 2018. "Self-Service Data Preparation: Research to Practice," *Undefined*. (https://www.semanticscholar.org/paper/Self-Service-Data-Preparation%3A-Research-to-Practice-Hellerstein-Heer/715cba311d4e5ad6b5f8cba7694ccc03ef7583b7).

Jin, Z., Anderson, M. R., Cafarella, M., and Jagadish, H. V. 2017. "Foofah: Transforming Data By Example," in *Proceedings of the 2017 ACM International Conference on Management of Data - SIGMOD '17*, New York, New York, USA: ACM Press, pp. 683–698. (https://doi.org/10.1145/3035918.3064034).

Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. 2011. "Wrangler: Interactive Visual Specification of Data Transformation Scripts," in *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems - CHI '11*, New York, New York, USA: ACM Press, p. 3363. (https://doi.org/10.1145/1978942.1979444).

Kusumasari, T. F., and Fitria. 2016. "Data Profiling for Data Quality Improvement with OpenRefine," in *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, IEEE, October, pp. 1–6. (https://doi.org/10.1109/ICITSI.2016.7858197).

Lee, I. 2017. "Big Data: Dimensions, Evolution, Impacts, and Challenges," *Business Horizons* (60:3), Elsevier, pp. 293–303. (https://doi.org/10.1016/J.BUSHOR.2017.01.004).

Milutinovic, V., Kotlar, M., Stojanovic, M., Dundic, I., Trifunovic, N., and Babovic, Z. 2017. *DataFlow Systems: From Their Origins to Future Applications in Data Analytics, Deep Learning, and the Internet of Things*, Springer, Cham, pp. 127–148. (https://doi.org/10.1007/978-3-319-66125-4_5).

Obitko, M., and Jirkovský, V. 2015. *Big Data Semantics in Industry 4.0*, Springer, Cham, pp. 217–229. (https://doi.org/10.1007/978-3-319-22867-9_19).

Ong, K. W., Papakonstantinou, Y., and Vernoux, R. 2014. "The SQL++ Semi-Structured Data Model and Query Language: A Capabilities Survey of Sql-on-Hadoop, Nosql and Newsql Databases," *CoRR, Abs/1405.3631*.

Ostrovsky, D., Haji, M., and Rodenski, Y. 2015. "The N1QL Query Language," in *Pro Couchbase Server*, Berkeley, CA: Apress, pp. 107–133. (https://doi.org/10.1007/978-1-4842-1185-4_6).

Parody, L., Vaca, Á. V., Gómez-López, M., and Gasca, R. 2017. "FABIOLA: Defining the Components for Constraint Optimization Problems in Big Data Environment," in *International Conference on Information Systems Development (ISD) 2017*, , September 26. (https://aisel.aisnet.org/isd2014/proceedings2017/CogScience/3).

Reilly, E. D., Ralston, A., and Hemmendinger, D. 2003. "Backus-Naur Form (BNF)," in *Encyclopedia of Computer Science*, Wiley, pp. 129–131. (https://dl.acm.org/citation.cfm?id=1074155).

Riehle, D., Dirk, Riehle, and Dirk. 1997. "Composite Design Patterns," *ACM SIGPLAN Notices* (32:10), ACM, pp. 218–228. (https://doi.org/10.1145/263700.263739).

Stefanowski, J., Krawiec, K., and Wrembel, R. 2017. "Exploring Complex and Big Data," *International Journal of Applied Mathematics and Computer Science* (27:4), Walter de Gruyter & Co., pp. 669–679. (https://doi.org/10.1515/amcs-2017-0046).

Valencia-Parra, Á., Ramos-Gutiérrez, B., Varela-Vaca, Á. J., Gómez-López, M. T., and Gracía Bernal, A. 2019. "Enabling Process Mining in Aircraft Manufactures: Extracting Event Logs and Discovering Processes from Complex Data," in *Proceedings of the Industry Forum at BPM 2019*, Vienna, pp. 166–177.