Journal of Industrial Information Integration 41 (2024) 100657

Contents lists available at ScienceDirect

Journal of Industrial Information Integration

journal homepage: www.elsevier.com/locate/jii

Full length article

Domain-specific languages for the automated generation of datasets for industry 4.0 applications

Brian Sal, Diego García-Saiz, Alfonso de la Vega [*], Pablo Sánchez

Software Engineering and Real-Time Group, Dpto. Ingeniería Informática y Electrónica, Universidad de Cantabria, Facultad de Ciencias, Avda. Los Castros 48, 39005 Santander, Cantabria, Spain

ARTICLE INFO

Keywords:
Data selection
Industry 4.0
Fishbone diagrams
Ishikawa diagrams
Domain specific languages

ABSTRACT

Data collected in Industry 4.0 applications must be converted into tabular datasets before they can be processed by analysis algorithms, as in any data analysis system. To perform this transformation, data scientists have to write complex and long scripts, which can be a cumbersome process. To overcome this limitation, a language called Lavoisier was recently created to facilitate the creation of datasets. This language provides high-level primitives to select data from an object-oriented data model describing data available in a context. However, industrial engineers might not be used to deal with this kind of model. So, this work introduces a new set of languages that adapt Lavoisier to work with fishbone diagrams, which might be more suitable in industrial settings. These new languages keep the benefits of Lavoisier, reducing dataset creation complexity by 40% and up to 80%, and outperforming Lavoisier in some cases.

1. Introduction

Industry 4.0 [1–3] has emerged in recent years as a disruptive set of technologies that, when properly combined, can clearly improve both manufacturing processes and manufactured products. These technologies are clearly having a positive impact on the manufacturing industry and even transforming it, leading to Industry 4.0 being identified as the 4th Industrial Revolution.

Industry 4.0 relies on technologies like *Internet of Things (IoT)* [4–13], *CyberPhysical Systems* [14–16], *Cloud Computing* [17–19], *Big Data* [13,20–24], *Data Analysis* [21,25] or *Augmented Reality* [26–29], among others. There are different kinds of Industry 4.0 applications, which includes, for example, *Digital Twins* [30–32], *Augmented Reality Maintenance* [33,34] or *Smart Manufacturing* [35–37]. This article focuses on those Industry 4.0 applications that aim to take advantage of the large amounts of data collected by the different interconnected elements that comprise modern assembly lines [38]. For instance, robots in an automated assembly line have sensors that can measure the pressure in their pneumatic systems, among other issues. This information can be collected and complemented with data extracted from other systems, such as a *MES (Manufacturing Execution System)* [39] to know, for example, which worker was operating the assembly line at a particular time.

This vast amount of data can then be analyzed to improve both the manufacturing processes and their output products. For example, predictive models might be constructed to forecast when a piece of a production line should be replaced before it fails, which would contribute to reduce maintenance standstills [40–42]. Similarly, data mining techniques might help to find the causes that lead to products with defects, so that we can focus on avoiding them to achieve *Zero-Defect Manufacturing (ZDM)* [43–46].

Therefore, this kind of Industry 4.0 applications uses data science algorithms to process and analyze data. Despite the recent advances in this field, these algorithms often require their input data to be in a very specific format, known as a *dataset*. A dataset is a table-based format in which all data associated to each element under analysis must be placed in a single row of that table. Nevertheless, data is not commonly available in this format, as it is usually stored in relational databases, XML or JSON files, among other alternatives. In these representations, data are not tabular, but linked and often hierarchical. For instance, data about a manufactured product may be linked to data of its components. Similarly, data of each component might be connected to data of its subcomponents, and so on. Therefore, we would need to flatten this hierarchical data to arrange them as columns of a specific row of a table before using them as input to a data analysis algorithm.

To perform a flattening task, data scientists typically write intricate and lengthy scripts in languages like SQL [47], R [48], or Pandas [49]. To perform the necessary transformations, these scripts typically chain

* Corresponding author.
E-mail addresses: brian.sal@unican.es (B. Sal), diego.garcia@unican.es (D. García-Saiz), alfonso.delavega@unican.es (A. de la Vega), sanchezbp@unican.es (P. Sánchez).

https://doi.org/10.1016/j.jii.2024.100657
Received 16 March 2023; Received in revised form 14 June 2024; Accepted 19 June 2024
Available online 25 June 2024

several low-level operations, such as *joins* [50] and *pivots* [51]. So, data scientists need to work at a low abstraction level, which makes the dataset creation process labor-intensive and prone to errors. Moreover, domain experts, like industrial engineers, can hardly create or modify these scripts, so data scientists need to be hired to execute this task. Since data scientists are expensive and scarce, development time and costs of Industry 4.0 applications increase.

To alleviate these problems, a language called *Lavoisier* was recently developed [52]. This language provides high-level primitives to create datasets that focus on what data must be selected and skip details of how these data must be rearranged to make them fit in a dataset. These primitives are processed by the Lavoisier interpreter, which automatically converts them into chains of low-level operations that retrieve and transform the selected data. Using Lavoisier, complexity of scripts for dataset creation can be reduced by 40% and 60% on average, as compared to Pandas and SQL, respectively, and achieving 80% of complexity reductions in some scenarios.

Lavoisier relies on *object-oriented domain models* [53] to represent domain data. Nevertheless, this type of model is rarely found in manufacturing settings, which might be an obstacle to the adoption of Lavoisier in Industry 4.0 contexts. On the other hand, industrial engineers are used to deal with *fishbone diagrams* [54], which allow representing cause–effect relationships, typically used for quality control in industrial settings, and that might be used to specify influence relationships between domain data. Therefore, in an Industry 4.0 context, it could be more desirable that Lavoisier could work with fishbone diagrams instead of with object-oriented domain models.

This work presents a model-driven process that adapts Lavoisier to work with fishbone diagrams. To build this process, we first created a new kind of fishbone model, named *data-oriented fishbone model*, where causes can be connected to domain data. For this purpose, we augmented fishbone models with Lavoisier statements, so that each cause can be connected to a subset of domain data that characterizes such a cause. Next, we designed a new language called *Papin* to select which causes of a fishbone diagram should be used to build a concrete dataset. Finally, the Papin interpreter processes this selection and automatically generates the required dataset. Thus, industrial engineers can use Papin to create datasets directly from (data-oriented) fishbone diagrams instead of from object-oriented domain models.

Expressiveness of data-oriented fishbone models and Papin has been evaluated by applying these languages to five external case studies [55–58]. No major issues were found. In addition, it was expected that data-oriented fishbone models and Papin were able to achieve similar rates of accidental complexity reduction than Lavoisier, as these languages are built on it. To confirm this hypothesis, we created a set of dataset creation scenarios that covered all kind of inputs these scripts might need to deal with. Then, we created scripts for each one of these scenarios using our approach, Lavoisier, SQL and Pandas and we compared these scripts using several metrics. As a result of this comparison, we concluded that the complexity reduction rates of Lavoisier were not seriously affected in the newly designed languages. Moreover, as compared to Lavoisier, these new languages might provide extra benefits when a same fishbone diagram was used to generate different datasets.

This work extends a previous contribution presented at a conference.[1] Over this contribution, this article includes, as main difference, a more exhaustive evaluation of the expressiveness and effectiveness of our approach (Section 5) as well as a more detailed description of how the presented languages were implemented (Section 4). Moreover, we extended the related work section to include *propositionalization* [59] and *Multi-Relational Data Mining (MRDM)* [60] as well as modeling languages in Industry 4.0 and ETL processes. Finally, we added some background information, such as Section 2.3, to make this work accessible to a wider audience; and we revised the remaining sections to clarify some issues and provide some extra examples, taking advantage of not having space limits for this paper.

After this introduction, this work is structured as follows: Section 2 details the motivation behind this work, using a running example. Sections 3 and 4 describe our solution and how it was implemented, respectively. Section 5 explains how expressiveness and effectiveness were evaluated, discussing the obtained results. Finally, Section 6 comments on related work and Section 7 summarizes our achievements and outlines future work.

2. Background and motivation

2.1. Software language engineering via metamodelling

In this work, we have designed and implemented several Domain-Specific-Languages (DSLs) following a metamodel-based approach [61]. These languages aim to provide end users with a better experience over General Purpose Languages (GPLs) for achieving concrete tasks. First, DSLs offer a concise syntax, which is constrained to the task or tasks for which they were specified. Therefore, users do not have to worry about the presence of mandatory language constructs that are not needed for the task at hand (something that might happen with some GPLs such as Java), which helps reduce accidental complexity. Second, specifying DSLs for concrete contexts allows using custom syntaxes that contain terms coming from the application domain. For instance, a DSL in the context of a university subject might include terms such as *student*, *deliverable*, or test; or a DSL for clinicians might refer to *treatments*, *patients*, *medications* and so on. As a result, domain experts might find the syntax of DSLs more familiar and easier to understand than that of GPLs.

For specifying our DSLs, we have applied metamodelling techniques [61–63]. The main elements of a DSL with these characteristics can be found in Fig. 1. We describe them in the following.

The central component of a metamodel-based DSL is its abstract syntax (top left of Fig. 1) or metamodel. The abstract syntax specifies those concepts that are relevant for the language, as well as their properties and relationships between them. This syntax is defined with a metamodel, which is usually specified by means of a special type of class diagram such as the one of Fig. 4. For our implementation, we use Ecore, i.e., the notation provided by the Eclipse Modeling Framework (EMF) for specifying metamodels [64]. In the metamodelling perspective, any model must conform to the abstract syntax metamodel (Fig. 1, label 1). This means that the program must adhere to the rules defined in the abstract syntax (e.g. which concepts can or must appear, how they can inter-relate, among others).

The abstract syntax is an internal representation of the DSL concepts, and end users do not interact with it directly. Instead, this abstract syntax is *rendered* through one of more concrete syntaxes (Fig. 1, label 2), which can be either textual or graphical. In the case of textual concrete syntaxes, a grammar is defined to allow end users to read and write programs for that DSL in textual form (Fig. 1, label 3). Additionally, any concrete syntax has to be accompanied by a parser that converts programs written in that concrete syntax to models conforming to the abstract syntax (Fig. 1, label 4). The separation between abstract and concrete syntaxes allows for several concrete syntaxes for the same language to coexist. For instance, we could have (1) more than one textual or graphical syntax; (2) a graphical and a textual syntax; (3) hybrid (graphical and textual) syntaxes; or (4) any combination of the above. Moreover, new syntaxes could be added to a language at a later stage, such as adding a graphical syntax to a DSL that initially only had a textual one.

Finally, programs written with a DSL are given a meaning by specifying the semantics of the language. There are different approaches to specify semantics, and here we comment on two: *translational* and

[1] The conference name and any citation to this previous work have been omitted to try to ensure a double anonymized review.

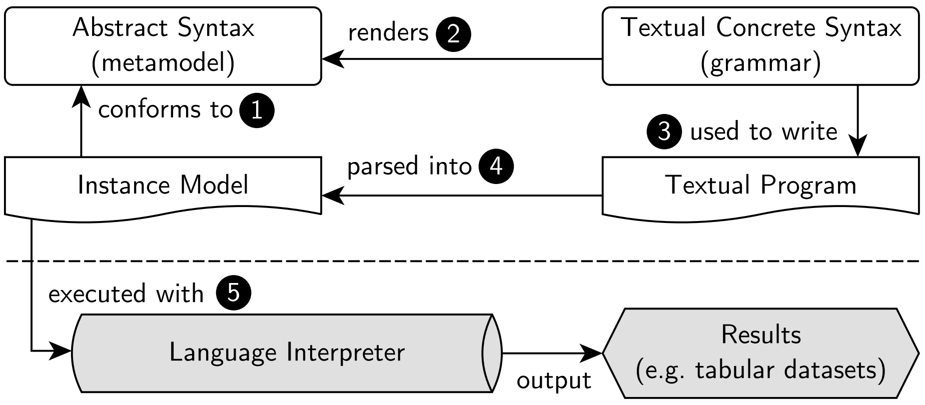

Fig. 1. Components of a textual domain-specific language with operational semantics.

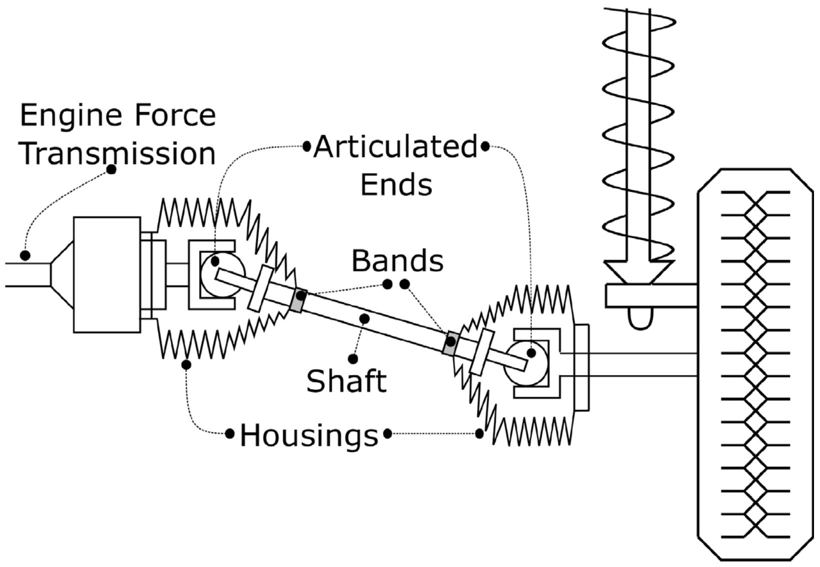

Fig. 2. Elements of a drive half-shaft.

operational. A DSL has *translational semantics* when its programs are converted to programs conforming to another language that has semantics, e.g., converting DSL expressions to Java or C expressions by means of code generation templates. In the case of *operational semantics*, which is the approach we follow for our languages, programs written with a DSL are processed by an interpreter (Fig. 1, label 5), which executes them step by step to obtain some kind of result (e.g., in our case, datasets).

The following sections present the running example and detail the challenges that we are trying to tackle in this work. Concrete examples of the language components introduced here are given in Section 4.

2.2. Running example: Falling band

As running example throughout this paper, we will use the production of *drive half-shafts* in a supplier company for the automotive sector. This example is taken from the literature [56] and based on a real case study. Fig. 2 provides a rough sketch of the components of a drive half-shaft. A drive half-shaft is a piece of the car transmission system that joins the engine with a wheel. It is composed of a rotating bar of metal, the *shaft*, which has two articulated ends that allow the wheel to move freely as required by the steering and suspension systems while at the same time receiving the engine force.

The articulated ends are protected by a flexible piece of rubber known as the *housing*. Housings are fixed to the shaft by means of two metal bands. To produce these drive half-shafts, an operator inserts a shaft in an assembly machine, mounts each housing on the shaft and tights the bands around the housings using a pneumatic pliers. Once finished, these pieces are sent to car manufacturers, who check compliance with their requirements. During these checks, car manufacturers detected that, sometimes, bands on the side of the wheel unfastened, originating different problems. The company wanted to analyze data gathered during the drive half-shafts production process in order to find the causes of this problem. To carry out such an analysis, the company may use data mining techniques, for which it would need to execute a data mining process, which is described in the next section.

2.3. Data mining processes

We will take the *KDD (Knowledge Discovery in Databases)* process [65] as basis to explain the different stages that are part of a data mining process. Some of these stages have been subdivided in smaller parts to highlight some issues that are particularly relevant for our work.

The final goal of all data mining processes is to answer one or more business questions (Fig. 3, step 1). For instance, in our running example, industrial engineers in the supplier company want to know what causes a band to fall. So, the very first step of a data mining process is to elicit the business questions to be answered.

After defining the business questions, data scientists must select those data sources that might have some influence on these questions

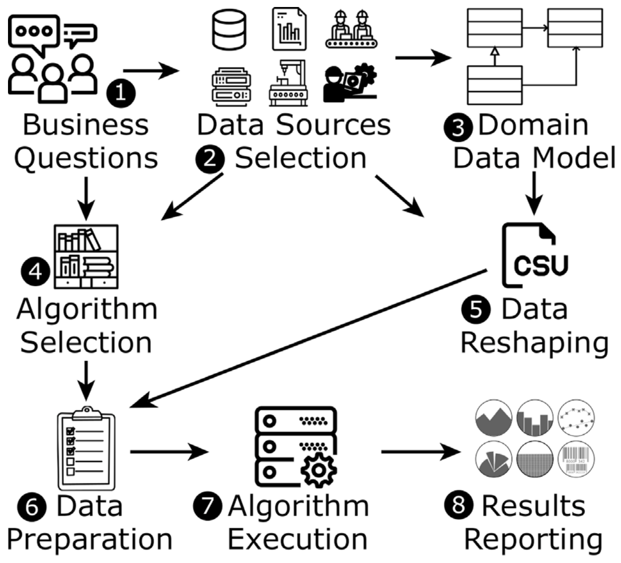

Fig. 3. Detailed data mining process.

and that, therefore, will be used to try to answer them (Fig. 3, step 2). In an Industry 4.0 context, data can come from very heterogeneous sources, such as production machine sensors, quality assurance reports, technical specifications of components used in the manufacturing process, or management data hosted in Manufacturing Execution Systems (MES), among other options. These sources must be selected with the help of domain experts, who know what data might be helpful to find answers to the business questions.

Data provided by these sources is initially unstructured and described in different notations despite having relationships among them. For instance, data of manufactured products might be initially retrieved from the MES relational database, which might be visualized using Entity-Relationship (ER) models [66]. These data might refer to raw materials used during the manufacturing process. Data about these raw materials might be available in technical data sheets, stored as pdf files. Additionally, some properties of these raw materials might have been checked during quality control processes. Results of these quality inspections might be registered in spreadsheets. In cases like this, it is worth constructing a *domain model* that unifies these data bundles and helps to visualize them as a whole in a common format [67] (Fig. 3, step 3).

Subsequently, data scientists need to select the data mining techniques that may be more appropriate for finding an answer to each business question (Fig. 3, step 4). For example, in our running example, *association rules* [68] might be employed to find patterns in data that are most likely to lead to a falling band. For each one of these data mining techniques, a vast amount of concrete algorithms exists. Obviously, each algorithm provides advantages, as compared to the others, in a specific context. Consequently, data scientists must select the most appropriate algorithm for each case. For instance, in our case study, we could use the *Apriori algorithm* [69] to discover the most frequent patterns in terms of *confidence* - number of times the pattern lead to a phenomenon as compared to the number of times the pattern does not lead to that phenomenon, and *support* - number of times the pattern is found in the dataset as compared to the number of items in the dataset. However, we may use other algorithms to get patterns with low support and high confidence [70]. This is, patterns that appear few times but that, when they appear, they lead to a falling band with a high probability.

To execute the selected data mining algorithms, data scientists need to provide them with input data. As already commented in the introduction, most data mining algorithms only accept as input data formatted as a *dataset*. Since data in the selected data sources is not initially in this format, data scientists need to transform them into datasets (Fig. 3, step 5). This work tackles this concrete problem, which will be detailed in the next section.

Moreover, depending on the selected algorithm, input data might need to satisfy some additional constraints. For example, data might need to be normalized into the range [0, 1] before being provided as input to some algorithms based on distances. Thus, in these cases, data scientists must carry out some extra data transformations to fulfill these extra requirements (Fig. 3, step 6).

Finally, data scientists run the data mining algorithms using the extracted and reshaped data as input (Fig. 3, step 7). Before passing the results of these executions to the domain experts, data scientist must analyze their quality and reliability. Then, those results that are considered as sound and reliable are presented in a way that can be easily visualized and understood by domain experts, so that they can interpret them correctly and use them to make better or more informed decisions (Fig. 3, Step 8). For instance, in our running example, the results of the analysis might report that most of drive-half shafts with falling bands contain bands from a specific provider and housings from another concrete provider, but on the other hand, these bands and housings, when mixed with pieces of other providers, do not lead to falling bands. Therefore, the problem would not be with these pieces themselves, but with their combination in the same half-shaft.

As previously commented, this work focuses on a specific stage of data mining processes, which is the dataset creation process (Fig. 3, Step 5). This stage is described in detail in the next section.

2.4. The dataset generation problem

As we have already said in previous sections, most data analysis algorithms can just process data provided in a concrete tabular format, known in the data science community as a *dataset*. In this format, all data belonging to a single entity to be analyzed must be place in a same table row, which we have named as the *one entity, one row* constraint.

Nevertheless, domain data is typically available as a graph of linked and nested elements. As an example, Fig. 4 shows a fragment of an object-oriented data model for our running example[2] and Listing 1 provides an example of data, in JSON format, for a concrete half-shaft, conforming to that model. The data model is in Ecore notation [64], which, roughly speaking, can be considered as a subset of UML 2.0 class diagrams.

```
[{‘id’:  ‘5209’,‘manufacturedTime’:‘2021—02—12  11:27:13’,
    ‘wheelBand’:{‘model’: ‘SNR390’,
      ‘parameters’:{‘batchId’:‘2020/2436832’,
                    ‘maxThickness’:1.67,‘minThickness’:1.64,
                    ‘avgThickness’:1.66,‘provider’:{‘name’: ‘ACME’’}}},
     ...,
     ‘assemblySession’:  {‘start’:‘2021—02—12  11:14:58’,
         ‘stop’:‘2021—02—12  11:17:23’,
         ‘parameters’:  {‘temperature’:20.4,‘humidity’:53.8,
             ‘pliersTightenings’:[
                 {‘number’:1,‘pressure’:7.9},
                 {‘number’:2,‘pressure’:7.9},
                 {‘number’:3,‘pressure’:7.8},
                 {‘number’:4,‘pressure’:7.9}]}}},
{...}]
```

Listing 1: Data for a drive half-shaft instance.

According to Fig. 4, each *DriveHalfShaft* has associated a *ComplianceReport*, created by the company customers, that indicates whether the piece fulfills the customer requirements. Each *DriveHalfShaft* is comprised of two *bands*, among other elements. Each band belongs to a *batch*. Band batches are acquired from different providers and, for each batch, some *parameters* are measured to verify its quality. On the other hand, each *DriveHalfShaft* is mounted in a *session* of an assembling machine. This machine has several sensors that monitor different parameters such as *temperature*, *humidity*, or the *pressure* in the pneumatic pliers system before tightening each band.

[2] The complete domain model for this case study is available in Appendix A.

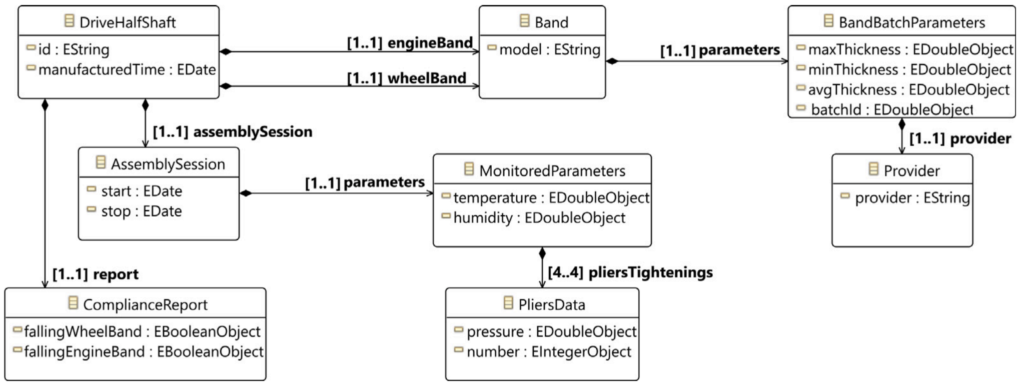

Fig. 4. A fragment of the domain model for the production of drive half-shafts.

id	...	humidity	temperature	1_pressure	2_pressure	3_pressure	4_pressure
5209	...	56.7	20.4	7.9	7.9	7.8	7.9
6556	...	58.0	19.8	7.2	7.1	7.1	7.1
...	...	...	...	...	...	...	...

Fig. 5. A tabular dataset for drive half-shaft analysis.

```
1   select a.id, m.temperature, m.humidity,
2     (select p.pressure from  PliersData p
3        where p.idMonitoredParameters = m.id
4          and p.number = 1),
5     (select p.pressure from  PliersData p
6        where p.idMonitoredParameters = m.id
7          and p.number = 2),
8     (select  p.pressure from  PliersData p
9        where p.idMonitoredParameters = m.id
10         and p.number = 3),
11    (select p.pressure from  PliersData p
12       where p.idMonitoredParameters = m.id
13         and p.number = 4)
14    from AssemblySession a join MonitoredParameters m
15       on a.idMonitoredParameters = m.id
```

Listing 2: SQL script to rearrange *Assembly* information as tabular data.

```
1   dataset driveHalfShaft_AssemblyParameters {
2     mainclass DriveHalfShaft [id] {
3       include report {
4         include assemblySession {
5           include parameters {
6             include pliersTightenings by number
7           }
8         }
9       }
10    }
11  }
```

Listing 3: An example of Lavoisier specification.

As it can be seen in Listing 1, these data are linked and nested. For example, the drive half-shaft with id *5209* (Listing 1, line 1) is linked to some *wheelBand* data (Listing 1, lines 2–6), as well as some *assembly session* data, among others. Moreover, each assembly session data contains four nested measures of the pressure of the pneumatic pliers system. Therefore, to create a dataset, we would have to transform these linked and nested information into sequence of values that can be place in cells of a table row, such as depicted in Fig. 5.

To create datasets, data scientists write complex and long scripts in which raw data is transformed using different chains of low-level operations, such as *filters*, *joins*, *pivots* or *aggregations*, to satisfy the *one entity, one row* constraint.

These scripts are written using data manipulation languages or libraries, like SQL [47] or Pandas [49]. Listing 2 shows an example of an SQL script for combining data from the *AssemblySession*, *MonitoredParameters* and *PliersData* classes to create a tabular representation like depicted in Fig. 5. This script needs to place each measure of the pressure in the pliers pneumatic system in a separated column, which is achieved by using a nested query for each specific tightening (Listing 2, lines 2–13).

As the reader can easily notice, this script is quite complex as compared to the example size and it has obvious scalability problems if the number of tightenings grows. This problem is not specific to the SQL language and similar situations occur when using other alternatives, such as R [48] or Pandas [49]. As a result, dataset creation becomes a labor-intensive and error-prone process. To overcome this problem, a language called *Lavoisier* [52,71] was recently developed. This language is described in the next section.

2.5. Lavoisier

Lavoisier [52,71] is a declarative language for dataset creation that provides a set of high-level primitives whose goal is to specify what data must be included in a dataset without having to detail how selected data must be transformed to be arranged as a dataset. We illustrate how Lavoisier works using the example of Listing 3. In this example, the dataset will contain just data from the compliance reports and the assembly session for each drive half-shaft.

In Lavoisier, we must first specify the name of the dataset that will be generated, which, in our example, is *driveHalfShaft_AssemblyParameters* (Listing 3, line 1). Then, we must specify, using the *mainclass* keyword, what are the main entities that will be analyzed. It is worth remembering that, according to the *one entity, one row* constraint, all the information about each instance of these entities must be placed in a single row of the output dataset. In our case, the *DriveHalfShaft* class will represent these main entities (Listing 3, line 2).

When a class is selected to be included in a dataset, Lavoisier adds all attributes of that class to the output dataset, whereas references to other classes are excluded by default. This behavior can be modified according to our needs and a concrete subset of attributes can be selected by listing them between square brackets after the class name. For example, Listing 3, line 2 specifies that just the *id* of each half-shaft has to be included in the output dataset.

Regarding references, we can add them using the *include* keyword. This primitive works differently depending on the reference multiplicity. For single-bounded references, i.e., references with upper bound lower than or equal to one, we just need to specify the name of the reference to be added. For example, Listing 3, line 3 incorporates the compliance report of each half-shaft, retrieved through the *report* reference, to the output dataset.

For each included reference, the default Lavoisier behavior is executed again. This is, attributes are included and references are excluded. As before, we can specify a list of concrete attributes of a reference to be included between square brackets as well as new *include* statements that add references of the reference to the output dataset. For instance, Listing 3, line 5 adds the *parameters* of the *assemblySession* reference to the resulting dataset. This recursive inclusion can continue as deep as we need.

To process reference inclusions, the Lavoisier interpreter relies on a set of transformation patterns [71,72] that automatically reduce a set of interconnected classes into a single class from which a tabular dataset can be directly generated. In the case of single-bounded references, the Lavoisier interpreter simply adds the attributes of the referenced class to the class holding the reference. For example, in Listing 3, line 3, the attributes of the *ComplianceReport* class are added to the *DriveHalfShaft* class, which would be equivalent to performing a *left outer join* between these classes. To handle nested references, the Lavoisier interpreter processes the deepest reference first and repeats the process until reaching the main class.

The case of multibounded references, i.e., references with upper bound greater than or equal to one, is more challenging since each instance of the referencing class is related to several instances of the referenced class. So, to satisfy the *one entity, one row* constraint, we need to find a mechanism to spread data of each one of these instances into a single row. For instance, in the case of the *pliersTightenings* reference, of the *MonitoredParameters* class, we would need to specify how data of each pliers tightening is exactly spread over concrete columns of the output dataset. To achieve this goal, the strategy that Lavoisier follows is to create a specific set of columns for each kind of instance that might appear in a multibounded reference. This way, each instance is able to find a well-defined set of columns where to place its data.

This strategy requires that Lavoisier can distinguish between different kinds of instances of the target class. Therefore, we must use one or more attributes of class as identifiers for each type of instance. In Lavoisier, these identifier attributes are specified after a *by* keyword, which is added to *include* statements involving multibounded references. For example, Listing 3, line 6 specifies that the *pliersTightenings* reference must be added to the output dataset using the tightening number as identifier for each *PliersData* instance (see Listing 1, lines 11–14).

With this information, the Lavoisier interpreter proceeds as follows: first, it computes all existing distinct tuples for the identifier attributes of the target reference. These values are *{1, 2, 3, 4}* for our running example, and they represent all existing types of instances in the multibounded reference. Then, the set of non-identifier attributes of the referenced class is calculated. In our case, this set is *{pressure}*. This set represents the information to be specified for each type of instance. Next, to place the information of each instance type, for each identifier value and element in the set of non-identifier attributes, a column is added to the output dataset. Each one of these columns will have as name the name of the corresponding non-identifier attribute prefixed with its respective identifier value. In our example, these generated columns are *{1_pressure, 2_pressure, 3_pressure, 4_pressure}*. This structure is illustrated in Fig. 5. For the sake of clarity, we would like to mention that, if the referenced class had contained an additional non-identifier attribute, such as *tighteningTime*, the resulting set of columns would have been the following: *{1_pressure*, *1_tighteningTime*, *2_pressure*, *2_tighteningTime*, *3_pressure*, *3_tighteningTime*, *4_pressure*, *4_tighteningTime}*.

Formally speaking, the processing of a multibounded reference would be equivalent to joining both classes first and then pivoting the result using the identifier attributes as pivoting elements. The processing of multibounded references contains a lot of picky details that we are omitting here for the sake of simplicity. We refer the interested reader to the original Lavoisier article [52].

Multibounded references can also be processed using *aggregation functions*. These functions calculate values that summarize the set of the instances of the multibounded references. Thus, we could use these values as representatives of the multibounded reference instead of including its elements individually in the output dataset. For instance, in the case of the *pliersTightenings*, we might use the *average pressure* in the pliers pneumatic system instead of the values of the pressure for each individual tightening.

By using Lavoisier, the total number of operations required to generate a dataset decreases by ∼40%, and script size reduces by ∼60% as compared to other languages used for this purpose, such as SQL or Pandas. Therefore, as an initial hypothesis, we considered Lavoisier might be helpful for data selection in Industry 4.0. However, we soon realized that most of the industrial engineers in the companies in our region were not familiar with object-oriented data models in particular, and with data models in general.

Therefore, we analyzed what kind of models were commonly used in these companies and whether these models could be used as a replacement for object-oriented data models in our approach. Among several alternatives, such as diagrams for describing assembly lines, we found a type of model, called *fishbone diagrams*, that offered three interesting elements for our goals: (1) it was known by a reasonable number of engineers; (2) it has a very easy to understand notation; and (3) it could be used to represent cause–effect relationships between domain data. Therefore, we decided to use these fishbone diagrams to replace the object-oriented data models of our original solution when working in Industry 4.0 contexts. Next section describes how these fishbone diagrams work.

2.6. Fishbone diagrams

Fishbone diagrams [54], also known as *Cause-effect*, *Ishikawa* or *Fishikawa diagrams*, aim to identify causes that might lead to a certain effect. They were formally proposed by Kaoru Ishikawa and they are acknowledged nowadays as one of the seven basic tools for quality control and process improvement in manufacturing settings [73].

Fig. 6 shows a fishbone diagram taken from the literature [56], for our running example. As it can be observed, these diagrams are called fishbone diagrams because of its shape, which resembles a fish skeleton. They are elaborated as follows. First, the effect to be analyzed is placed on the right, commonly inside a rectangle or a circle, representing the fish head. In our case, *falling bands on the wheel side* would be the problem, or effect, to be studied. Then, a horizontal line from right to left, representing the fish main bone, is added. This line is used to connect causes with the effect.

Then, we start to identify causes that might lead to the effect. For each identified cause, we would try to find subcauses that might lead to that cause, repeating recursively the process until reaching a decomposition level that can be considered as adequate. To help to start the process, a set of predefined main causes, known as *categories*, are used in each domain. These categories are attached as ribs to the main fishbone. In the case of the manufacturing domain, these predefined

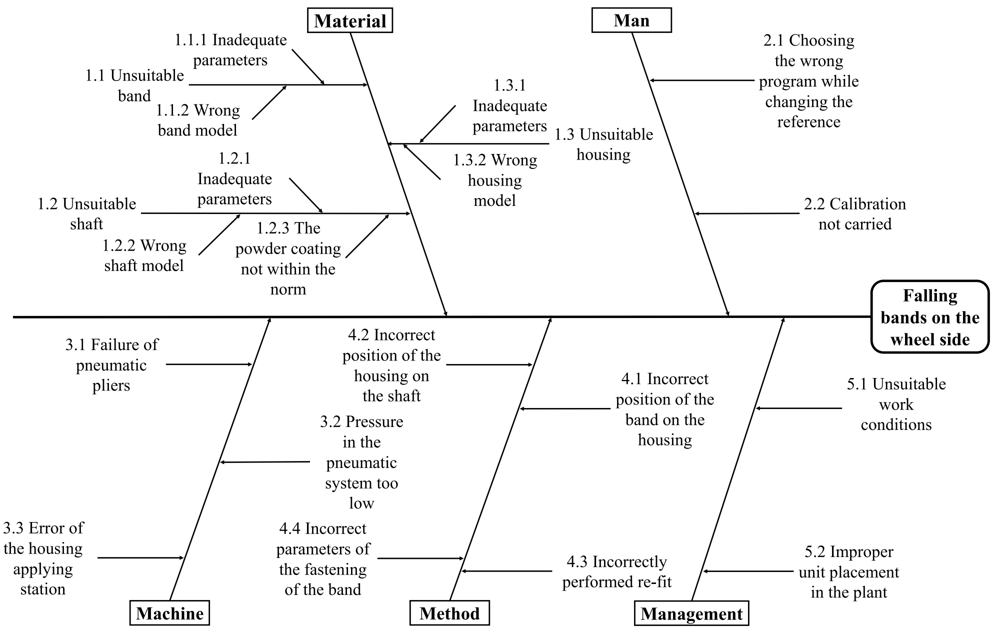
Material
1.1 Unsuitable band
1.1.1 Inadequate parameters
1.1.2 Wrong band model
1.2 Unsuitable shaft
1.2.1 Inadequate parameters
1.2.2 Wrong shaft model
1.2.3 The powder coating not within the norm
1.3 Unsuitable housing
1.3.1 Inadequate parameters
1.3.2 Wrong housing model
Man
2.1 Choosing the wrong program while changing the reference
2.2 Calibration not carried
Falling bands on the wheel side
Machine
3.1 Failure of pneumatic pliers
3.2 Pressure in the pneumatic system too low
3.3 Error of the housing applying station
Method
4.1 Incorrect position of the band on the housing
4.2 Incorrect position of the housing on the shaft
4.3 Incorrectly performed re-fit
4.4 Incorrect parameters of the fastening of the band
Management
5.1 Unsuitable work conditions
5.2 Improper unit placement in the plant

Fig. 6. A fishbone diagram for the drive half-shaft running example.
Source: Taken from [56].

categories, known as the *5 Ms model* [73], are *Material*, *Man*, *Machine*, *Method* and *Management*, as illustrated in Fig. 6.

Material specifies causes related to the source materials used during the manufacturing process. For instance, a wheeling band might fall because an unsuitable band has been used (Fig. 6, cause 1.1). A band might be unsuitable either because a wrong model is used or because its parameters are not adequate. For example, the band might be thicker than required. *Man* collects causes related to people participating in the process, such as the operator selecting a wrong assembly program (Fig. 6, cause 2.1). *Machine* analyzes elements of the machines and tools used in the process, such as, for instance, the pressure in the pliers pneumatic system (Fig. 6, cause 3.2). Finally, *Method* refers to the procedure used to manufacture an item, such as machine configurations; whereas *Management* considers general aspects of the process, as adequate working conditions or workers' motivation. It should be noted that more categories might be added if it is considered helpful. Indeed, in some manufacturing companies, these categories are complemented with an extra E that represents the *Environment*, which is known as the 5M + 1E model.

2.7. Problem statement

As commented, whereas object-oriented data models are rarely found in manufacturing settings, fishbone diagrams are frequently used for quality control in these contexts. These diagrams establish cause–effect relationships between domain elements, but they can also be used to identify influence relationships between domain data, helping to decide what concrete data might be useful to include in a dataset for analyzing a specific phenomenon. Therefore, it would be desirable that Lavoisier worked with fishbone diagrams instead of object-oriented models in industrial contexts. Nevertheless, fishbone diagrams were not designed to represent data. So, in a first step, we would need to adapt them for this purpose. The following section describes how we have addressed these challenges.

3. Dataset generation from fishbone diagrams

3.1. Solution overview

Fig. 7 shows the general scheme of our solution. The inputs are a fishbone diagram (Fig. 7, label 1) and an object-oriented model (Fig. 7, label 2), which describes data available in a domain. This fishbone diagram is a classical one created for quality control purposes, such as the one depicted in Fig. 6. Because of this, we will call it *Quality Control Fishbone diagram* (QCF). This fishbone diagram is created by industrial engineers, does not need to conform to any syntax, and it can be created using any tool, including sketching tools or whiteboards.

The object-oriented domain model is created by data scientists to fulfill two objectives: (1) to provide a well-formed description of data available in a concrete domain; and (2) to supply an access layer to retrieve domain data. This model must conform to the Ecore metamodel [64], so that it can be accessed and navigated using model management tools. Those relationships between data that cannot be modeled using this language, such as functional dependencies between attributes of different classes, can be specified using OCL (*Object Constraint Language*) [74–76].

We encourage industrial engineers to review domain models and data scientists to check fishbone diagrams, as it is indicated by the go and back arrows between these elements. By inspecting domain models, industrial engineers might discover new causes that could be added to the fishbone diagrams. On the other hand, data scientists could find elements in fishbone diagrams that might be included in the domain data model. However, we recommend that industrial engineers not be exposed to domain models until they are comfortable with this approach, to avoid overwhelming them with too many new elements.

Using these models as inputs, a new kind of fishbone diagram, which we have called *Data-Oriented Fishbone diagram* (DOF) (Fig. 7, label 3), is created by data scientists. A DOF reshapes a quality-control fishbone diagram and links causes with data that characterize these causes. The connections between causes and domain data are established using special code blocks that we have called *data feeders*. A data feeder specifies, using Lavoisier-like primitives, what data of a domain model must be used to represent a certain cause. Therefore, a DOF specifies influence relationships between domain data.

Finally, industrial engineers specify what concrete causes of a DOF must be included in a dataset using a new language called *Papin*. Papin specifications (Fig. 7, label 4) are automatically processed by the Papin interpreter, which, using the information in the *data feeders*, retrieves and transforms domain data to automatically create a dataset (Fig. 7, label 5).

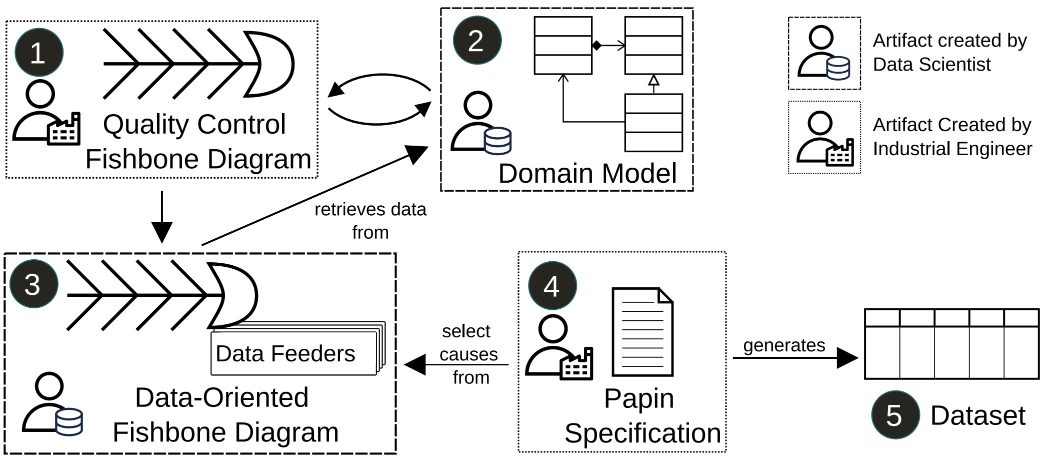
1
Quality Control Fishbone Diagram
2
Domain Model
Artifact created by Data Scientist
Artifact Created by Industrial Engineer
retrieves data from
3
Data Feeders
Data-Oriented Fishbone Diagram
select causes from
4
Papin Specification
generates
5
Dataset

Fig. 7. Overview of our solution.

3.2. Data-oriented fishbone models

A *data-oriented fishbone model* links causes in a fishbone diagram with data in an object-oriented domain model. For this purpose, we designed a special kind of code blocks called *data feeders*. A data feeder is comprised of two elements: a *path* and a *Lavoisier expression*. The path, starting always from a main class, traverses the domain data model to reach a class, an attribute or a collection that can be used to characterize a concrete cause. For instance, the data in the *BandBatchParameters* class might be employed to analyze the cause *Band Inadequate Parameters* (Fig. 6, cause 1.1.1).

As in Lavoisier, when a class is referenced by one of these paths, all its attributes are considered to be included in the output dataset, whereas all the references are excluded by default. This behavior can be modified by attaching a *Lavoisier expression* to the end of the path. This expression would specify, using an attribute list between square brackets and include statements, what concrete attributes and references of the selected class should be incorporated to the output dataset.

In the following, we describe how to construct a data-oriented fishbone diagram using Listing 4 as an example. This listing shows a fragment of a data-oriented fishbone diagram for the quality-oriented fishbone diagram depicted in Fig. 6.

A DOF must always start with the declaration of an effect. To associate domain data to the effect, we must take into account that, for analyzing a phenomenon, data analysis algorithms often require to be fed with data of entities that manifest that effect and entities that do not. For instance, to find patterns that lead to falling bands, these algorithms might need to compare drive half-shafts with and without falling bands. Therefore, in a DOF, the effect is associated to the domain entity whose instances might exhibit it.

Listing 4, lines 1–2 shows an example of an effect declaration. This declaration contains a name for the effect, *FallingBand*, the *is* keyword and a data feeder referencing a class from the domain data model. In our case, falling bands are analyzed by inspecting instances of *DriveHalfShaft*s. The class referenced by the effect is considered the main class for the output dataset. This means that this is the class for which the *one entity, one row* constraint must be satisfied.

In our example, since we need to know for each drive half-shaft whether its bands fall or not, we must include together with each half-shaft its compliance report, which is who contains the data about falling bands. This addition is performed using an *include* statement borrowed from the Lavoisier language (Listing 4, line 2).

After the effect declaration, to construct a DOF, we specify *category blocks* (Listing 4, lines 3–13, 14–17 and 19–21). These blocks simply mimic the category structure of the quality-oriented fishbone diagram

```
 1  effect FallingBand is DriveHalfShaft[id]
 2    include report[fallingWheelBand]
 3  category Material
 4    cause Bands realizes ‘Unsuitable band’ contains {
 5      cause WheelBand contains {
 6        cause WB_Model realizes ‘Wrong band model’ is
 7          wheelBand.model
 8        cause WB_Parameters realizes ‘Inadequate
              parameters’ is
 9          wheelBand.parameters { include provider }
10      } -- WheelBand
11      cause EngineBand contains { ... }
12    } -- Bands
13    cause Shaft realizes ‘Unsuitable shaft’ contains
           {...}
14  category Machine
15    cause PneumaticPliersPressure realizes
16     ‘Pressure in the pneumatic system too low’ is
17     assemblySession.parameters.pliersFastenings by
          number
18     ...
19  category Management
20    cause WorkingConditions
21      realizes ‘Unsuitable Work Conditions’ notMapped
```

Listing 4: A data-oriented fishbone model for our running example.

associated to the data-oriented fishbone model. Next, for each category block, we attach causes to it.

A DOF always tries to replicate the same structure of a QCF, but it might need to modify this structure sometimes. For instance, in Fig. 6, we find a cause named *Unsuitable Band* (cause 1.1), but there are two bands tightened to a shaft, one on the engine side and the other one on the wheel side. Therefore, when creating the DOF, we need to decompose this cause into two subcauses: *Unsuitable Engine Band* and *Unsuitable Wheel Band*. To keep traceability between causes of a DOF and its corresponding QCF, causes in a DOF have an optional *realizes* statement, which serves to specify which cause of a QCF is associated to each cause in a DOF (see, for example, Listing 4, line 6).

In a DOF, three kinds of causes can be created: (1) compound causes; (2) data-linked causes; and, (3) not mapped causes. *Compound causes* represent causes containing one or more subcauses. Data for analyzing a compound causes is obtained by merging data associated to its subcauses. *Data-linked causes* are atomic causes that contain data feeders that specify what domain data must be retrieved to analyze them. Finally, *notMapped causes* represent atomic causes that are not associated to domain data. These causes serve to specify causes for which domain data is not available yet, but that we want to keep in a DOF to preserve traceability between that DOF and its associated QCF. We provide examples of each one of these causes below.

Listing 4, lines 4–12 show an example of a *compound cause*, whose name is *Bands* and that realizes the quality-oriented cause *Unsuitable Band* (see Fig. 6, cause 1.1). This cause is comprised of two subcauses: *WheelBand* (Listing 4, lines 5–10) and *EngineBand* (Listing 4, line 11), which are compound causes too. As previously commented, these causes are not in the quality-oriented fishbone model and they are added here to adapt the DOF to the domain model.

Concerning data-linked causes, Listing 4, lines 6–7 provide an example of one, which is named *WB_Model*. In these causes, following the name and the optional realizes statement, we must specify a data feeder after the *is* keyword. The data feeder, as previously commented, is composed of a path that traverses the domain data model plus an optional Lavoisier expression. This path is implicitly assumed to start in the main class and it can reference as target element: (1) a single attribute; (2) a reference to a class; or, (3) a collection of references to a class.

In Listing 4, lines 6–7, the pointed element is an attribute, more specifically the *model* attribute of the *Band* class. This means the *WB_Model* cause will be characterized by the value associated to that attribute. Listing 4, lines 8–9 show an example of data feeder path where the target element is a class. Moreover, in this case, we make use of a Lavoisier expression to specify that the *provider* reference of the target class must also be used to characterize this cause. Finally, Listing 4, lines 15–17 illustrate a case where the path of a data feeder points to a collection. Collections are handled as multibounded references, which means we need to provide identifiers for the instances in this collection so that data of each individual instance can be spread over a set of well-defined columns. In our concrete case, each measure of the pressure in the pliers pneumatic system is identified by its tightening number, as specified after the *by* keyword.

Finally, Listing 4, lines 19–21 contain an example of a not mapped cause. Obviously, working conditions, such as dusty or noisy environments, might be a cause for falling bands. However, we do not have data about these issues, so we cannot characterize these causes with domain data, so they must be managed as a not mapped cause. To keep these causes is useful to maintain traceability between fishbone models and to record hints about what new data might be gathered to improve this data analysis process.

3.3. Papin: Dataset specification by cause selection

After building a DOF, we use *Papin* to specify the causes of that model that should be included in a dataset for analyzing the DOF effect. Listing 5 provides an example of *Papin specification*. A Papin specification always starts defining a name for the output dataset (Listing 5, line 1), which is *FallingBand_MaterialsAndPressures* in our case. Then, we must provide the name of the DOF from which the causes will be selected (Listing 5, line 2). In our example, this fishbone model is the one depicted in Listing 4, which is identified as *FallingBand*. Then, categories to be included in the output dataset must be explicitly listed. In our case, two categories, *Material* and *Machine*, are selected (Listing 5, lines 3 and 4).

By default, all the causes included in a category or compound cause are added to the output dataset when these elements are selected. For example, in Listing 5, line 3, all causes inside the *Material* category are implicitly added to the target dataset. If we wanted to select just a subset of causes of a compound element, we must list the concrete subcauses to be selected inside an inclusion block after the element name. Listing 5, lines 4–6 show an example of inclusion block. In this case, just the *PneumaticPliersPressure* cause of the *Machine* category is added to the output dataset.

Specifications are processed by the Papin interpreter, which provides the semantics for the language. These semantics are summarized

```
1  dataset FallingBand_MaterialsAndPressures
2    using FallingBand {
3      include Material
4      include Machine {
5        include PneumaticPliersPressure
6      }
7    }
```

Listing 5: An example of Papin specification.

Table 1
Procedure for interpreting of Papin specifications.

1. For each category or cause selected by a Papin specification, we traverse it to extract its data-linked causes.
2. Each data-linked cause is evaluated using the Lavoisier interpreter, which returns a dataset for each one of them.
3. All these datasets are merged using the identifier of the main class as matching attribute between dataset rows.
4. The resulting dataset is written to a CSV file.

in Table 1, and work as follows: First of all, we traverse each category or cause selected in a Papin specification to obtain their leaves causes, which can be either data-linked causes or not mapped causes. Not mapped causes are simply ignored. For each data-linked cause, we evaluate its data feeder to retrieve the domain data associated to the cause. The data feeder is evaluated by the Lavoisier interpreter. As a result of these evaluations, for each data-linked cause, we get a tabular structure with one column per each attribute that characterizes that cause, plus one column for the identifier of the instances being analyzed.[3] For example, the *WB_Model* cause, contained in the *WheelBand* cause (Listing 4, lines 6–7), when selected, generates a tabular structure with *id* and *wb_model* as columns, where *id* is the identifier for a drive half-shaft and *wb_model* is the number of the model of the band in the wheel side.

Since these tabular structures are computed using the same transformation patterns executed by the *Lavoisier* interpreter, all of them satisfy the *one entity, one row* constraint. Each one of these tabular structures can be considered a projection of the output dataset that contains just the data for analyzing a concrete cause. Therefore, to generate the complete dataset, we would just need to combine these projections. This task can be easily executed by performing *joins* between these projections. The resulting table is then combined by a new *join* with the effect data, and the CSV file for the output dataset is finally generated from this resulting tabular structure.

4. Implementation

Here we comment on some relevant aspects of the implementation of the languages we designed to adapt Lavoisier to work with fishbone diagrams. This implementation is publicly available in an external repository.[4]

Fig. 8 provides a scheme of the different elements that comprise our model-driven infrastructure. This infrastructure is comprised of:

1. three domain-specific languages for which we have provided their corresponding editors and parsers;
2. an object-oriented domain model, specified using Ecore [64];
3. domain data, which is retrieved through the domain model and, therefore, must conform to it;
4. the Papin interpreter, which generates a CSV file containing the required output dataset;

[3] If entities being analyzed are identified by several values, a column per each one of these values would be added to this tabular structure.

[4] https://anonymous.4open.science/r/fishbones-872E.

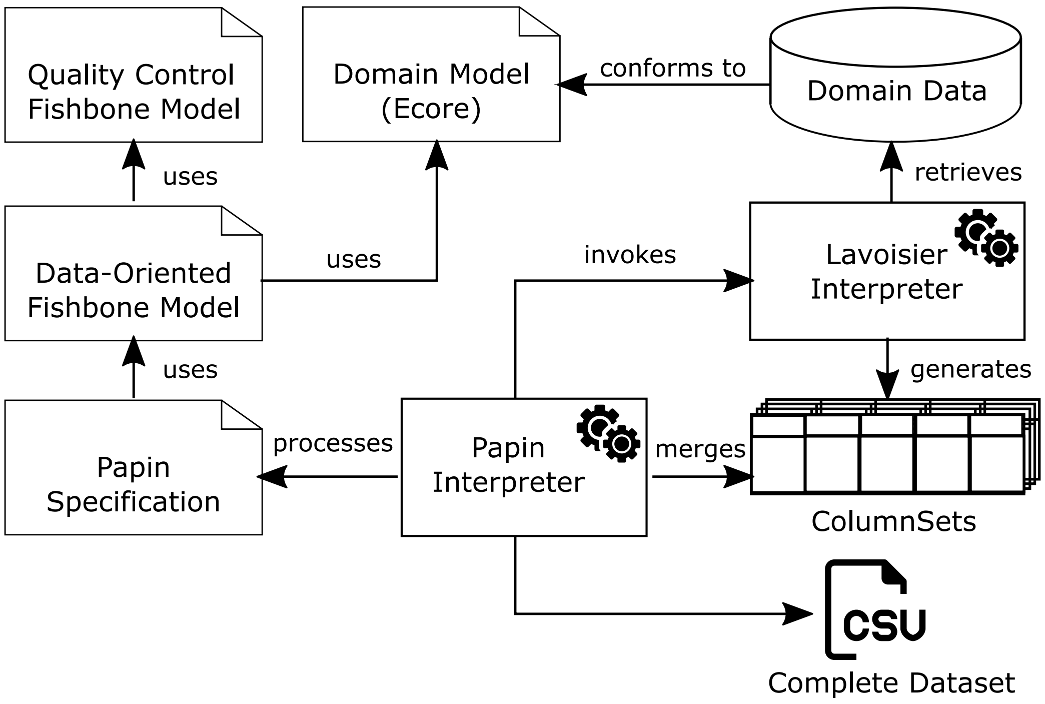

Fig. 8. Elements of our model-driven environment.

5. the Lavoisier interpreter, which is invoked by the Papin interpreter to process the Lavoisier expressions of the data feeders.

As it can be noticed, in addition to Data-Oriented Fishbone Models and Papin, we have also developed an editor for Quality-Control Fishbone models, so that these models can be stored inside our modeling infrastructure. The three languages were implemented following a metamodelling approach [61,62]. This approach starts with the definition of a metamodel that is, roughly speaking, a class diagram describing the abstract syntax of the language. The metamodels for our three languages were specified using Ecore [64], which can be considered the de-facto metamodeling standard.

Fig. 9 shows the metamodel for data-oriented fishbone models. This metamodel has been built atop the abstract syntax of Lavoisier, as expressions of the latter must appear in DOF specifications. A *DOF* model contains an *Effect*, which has associated a *DataFeeder* and is comprised of one or more *Categories*. Each *Category* contains one or more *Causes*. Each *Cause* can be a *NotMappedCause*, a *CompoundCause*, hosting several subcauses, or a *Data-Linked Cause*, which would have associated a *Data Feeder*.

The entry point of Lavoisier expressions into the DOF language is the *DataFeeder* element that, as described before, is used to relate elements of the domain model with effects or causes of the Fishbone Diagram. Therefore, the DOF elements that contain a data feeder are the *Effects* and the *Data-Linked Causes*. A data feeder has a *domainElementSelector*, which traverses the domain model from the class associated to the effect until reaching an attribute, a collection or a class. This target element is then processed using the *AttributeFilter* and *IncludeReference* elements, which are imported from the Lavoisier metamodel. Other less relevant Lavoisier syntax elements are used as part of a data feeder, such as the boolean expressions that can be specified to filter out some domain model instances. The interested reader can check the full abstract syntax of Lavoisier in its original article (see Figure 9 of de la Vega et al. [52]). The complete DOF metamodel is also available in the source code repository.[5] Similar metamodels were designed for QCFs and Papin. These metamodels are not shown here for the sake of brevity but they can also be found in the source code repository.[6][7]

After the creation of the metamodels for our languages, we defined a concrete syntax for each one of them. According to the metamodeling approach, this concrete syntax could be textual, graphical or even graphical-textual. In our case, we designed a textual syntax for all these languages because we considered that: (1) typing text is quicker than dragging and dropping graphical elements; (2) for DOF and Papin, textual specifications seem to be more natural than graphical ones; and, (3) textual models can be more easily versioned using configuration management tools than graphical ones. For these reasons, we opted for textual syntaxes even for quality-control fishbone models. Nevertheless, we must empirically check whether all the previous hypotheses are true as part of our future work.

The textual syntax for each one of our languages was defined using *Xtext* [77,78]. Xtext is a tool which allows language engineers to associate a grammar to a metamodel. As an example, Listing 6 shows the Xtext grammar of the DOF language, defined over the metamodel of Fig. 9. To illustrate briefly how Xtext works, we describe now the production rule for categories of the DOF language (Listing 6, lines 6–7). When executed (line 6), this rule creates an instance of the *Category* metaclass, contained in the DOF metamodel. A category definition begins with the *category* keyword, followed by an identifier that must fulfill a specific name convention, as indicated by the *ID* keyword. This string is assigned to the attribute *name* of the *Category* metaclass. Moreover, a *Category* must always contain a *Cause* (line 7), which must be specified according to the production rule for causes, not included in this example. This cause is added to the collection of *causes* of the *Category* metaclass. Finally, a *Category* might contain an unbounded number of causes, as specified at the end of line 7.

[5] https://gitlab.com/brian8sal/automated-generation-of-datasets-from-fishbone-diagrams/-/blob/master/es.unican.dof/model/generated/DOF.ecore?ref_type=heads.

[6] https://gitlab.com/brian8sal/automated-generation-of-datasets-from-fishbone-diagrams/-/blob/master/es.unican.qcf/model/generated/QCF.ecore?ref_type=heads.

[7] https://gitlab.com/brian8sal/automated-generation-of-datasets-from-fishbone-diagrams/-/blob/master/es.unican.papin/model/generated/Papin.ecore?ref_type=heads.

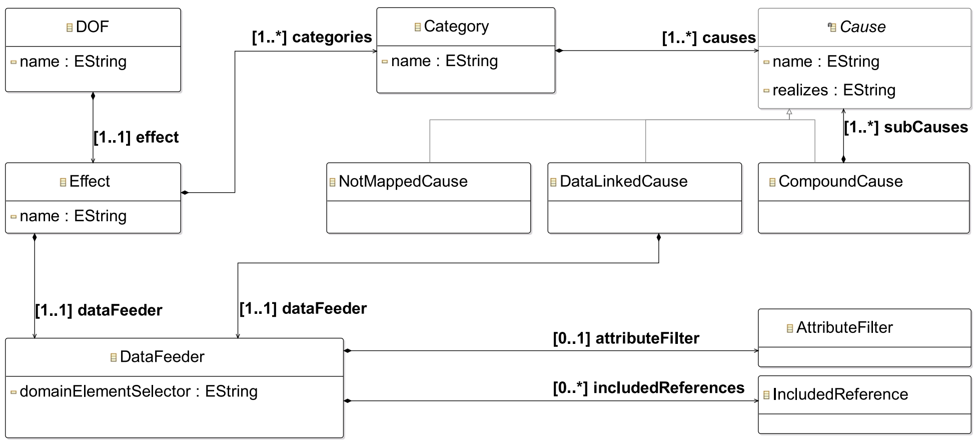

Fig. 9. Metamodel describing the abstract syntax of the DOF language.

```
DOF: 'dof' name=QualifiedName effect=Effect;

Effect: 'effect' name=ID 'is' dataFeeder=DataFeeder
    categories+=Category (categories+=Category)*;

Category: 'category' name=ID
    causes+=Cause (causes+=Cause)*;

Cause: CompoundCause | DataLinkedCause |
    NotMappedCause;

CompoundCause: 'cause' name=QualifiedName
    ('realizes' realizes=[qcf::Cause|EString])?
    'contains' '{' subCauses+=Cause (subCauses+=Cause)
        * '}';

DataLinkedCause: 'cause' name=QualifiedName
    ('realizes' realizes=[qcf::Cause|EString])?
    'is' dataFeeder=DataFeeder;

NotMappedCause: 'cause' name=QualifiedName
    ('realizes' realizes=[qcf::Cause|EString])?
    'notMapped';

DataFeeder: name=QualifiedNameWithWildcard
    attributeFilter=AttributeFilter?
    ('include' includedReferences+=IncludedReference)
        *;
```

Listing 6: Concrete textual syntax of the DOF language specified as an Xtext grammar.

As with the metamodel, the *DataFeeder* production rule is the entry point for the concrete syntax of the Lavoisier language. A full grammar including the integrated Lavoisier syntax is again available in the external repository.[8] The grammars for QCF and Papin can also be found in this external repository.[9][10]

The main advantage of using Xtext is that it is able to automatically generate an editor and a parser from a grammar specification. The generated editor provides some helpful features by default, like syntax highlighting, and can be easily extended to incorporate other common editor features, such as auto-completion or live-validations.

The data feeders of a DOF specification are interpreted on demand, this is, they are evaluated only and when their associated causes are selected as part of a Papin query. These queries are processed by the Papin interpreter, which with the help of the Lavoisier interpreter provides operational semantics to the language. The papin interpreter manages each selected cause as follows. If the cause is a data-linked cause, the Papin interpreter translates the data feeder expression to a Lavoisier expression. To do it, it replaces dots in the path to select a target element by *include* statements that do not select any attribute of the intermediate classes. Moreover, if the target element is an attribute, this is added to the attribute filter of the preceding class. Using this Lavoisier statement as input, the Papin interpreter invokes the Lavoisier interpreter, which retrieves domain data to generate a dataset corresponding to that statement. This dataset is provided as a *ColumnSet* [52], a data structure created for the Lavoisier implementation that is similar to a Pandas DataFrame.

If the cause to be processed is a compound cause, the Papin interpreter retrieves its subcauses and processes them individually. If the subcause is a not mapped cause, it is simply ignored. If the subcause is a data-linked cause, the process described in the previous paragraph is used. Finally, if the subcause is also a compound cause, the process is applied recursively.

5. Evaluation

This work has been developed to adapt Lavoisier to the Industry 4.0 context by allowing it to work with models that are more widely used in this environment, such as fishbone models. For this purpose, our solution provides two different elements: *Data-Oriented Fishbone Models* and *Papin*. These languages have two main objectives: (1) to increase the abstraction level at which work is performed when creating datasets; and, (2) to avoid people from the industry having to deal with model formats they are not familiar with to create a dataset.

We empirically validate here whether these objectives have been fulfilled by analyzing three different elements: (1) language expressiveness, this is, if the created languages can be effectively used in different Industry 4.0 problems; (2) whether industrial engineers can effectively deal with Papin specifications; and (3) how much accidental complexity can be saved by using *DOFs* and *Papin* for generating datasets.

In the following, we describe first the evaluation of the expressiveness of our languages. Then, we describe a small experiment we conducted to check whether industrial engineers were able to understand and create Papin specifications and we analyze its results. Next, we detail the procedure we have followed to evaluate how much accidental complexity can be removed thanks to *DOFs* and *Papin*, and we discuss the results gathered after executing this procedure. Finally, we comment and analyze potential threats to the validity of our results.

[8] https://gitlab.com/brian8sal/automated-generation-of-datasets-from-fishbone-diagrams/-/blob/master/es.unican.dof/src/es/unican/DOF.xtext.
[9] https://gitlab.com/brian8sal/automated-generation-of-datasets-from-fishbone-diagrams/-/blob/master/es.unican.qcf/src/es/unican/QCF.xtext.
[10] https://gitlab.com/brian8sal/automated-generation-of-datasets-from-fishbone-diagrams/-/blob/master/es.unican.papin/src/es/unican/Papin.xtext.

Table 2
Case study size and complexity.

Case study	Fishbone diagram			Domain model					
	#Ct	#Ca	Nest	#Cl	#Att	#1-Rfs	#*-Rfs	#Inh	Path
Nipples	6	24	1	16	51	10	7	0	2
Half shafts	5	33	2	17	41	21	3	0	5
Firewalls	4	16	0	9	30	2	2	1	1
Turbines	5	33	1	10	37	7	7	0	3
Alternator	4	22	0	8	29	2	2	1	1

#x: Number of *x*; Ct: Categories; Ca: Causes; Nest: Deepest Nesting for Causes; Cl: Classes; Att: Attributes; 1-Rfs: Single-bounded references; *-Rfs: Multibounded References; Inh: Inheritance Hierarchies; Path: Longest path between classes.

5.1. Expressiveness

To evaluate the expressiveness of *DOFs* and *Papin*, we used these languages to generate datasets for five external case studies. Four of these case studies were taken from the literature [55–58] and the remaining case study is from an industrial partner.

The first case study [57] is about finding causes in the production process of hydraulic nipples that might lead to defects in the final products. The second case study [56] is an extended version of the running example used throughout this work (see Section 2.2). The third case study [55] analyzes the production process of car firewalls, the piece that isolates the passenger compartment from the engine, to try to find causes for the appearance of rust in some of these firewalls. The fourth case study [58] aims to discover what was at the origin of the porosities found on the outlet nozzle of some manufactured turbines. Finally, the fifth case study, provided by an industrial partner of the automotive sector, analyzes data related to the tightening torque of an alternator pulley to reduce the number of these alternators that were rejected due to malfunctioning.

For the case studies taken from the literature, the articles describing them provided a quality-oriented fishbone diagram and a textual description of some of the data available in each domain. From these descriptions, we inferred the corresponding domain models, that were complemented with data from technical sheets of both the manufactured products and the materials used to manufacture them. For the industrial case study, the industrial partner has already specified a quality-oriented fishbone diagram and generated some datasets, provided as Excel sheets, from which we derived the corresponding domain model.

Using these quality-oriented fishbone diagrams and the domain models as input, we created the corresponding data-oriented fishbone diagrams and produced several datasets using Papin. Table 2 provides some numbers about the size and complexity of each case study, so that the reader can get a quick glimpse of them. Moreover, these models are available in the supplementary material of this article.[11]

We were able to specify a DOF and several Papin specifications for each one of these case studies without major problems. Therefore, we can conclude expressiveness of data-oriented fishbone models is adequate. Nevertheless, several small issues, most of them not related to language expressiveness, were detected during this evaluation process. We detail them below.

First, we detected that the same piece of data might be attached to two or more causes in some cases. For example, in the case of the car firewalls [55], the machine used to manufacture these firewalls has a liquid that is in contact with both the pieces of the firewall and the machine itself. Therefore, the pH level of this liquid might affect both these pieces, which would be in the *Material* category, and the machine, which would be, as obvious, in the *Machine* category. Consequently, if we select these two categories when generating a dataset, the pH level of this liquid would be included twice in such a dataset, which might lead to some problems with some data mining algorithms. To avoid this, we added an extra step to the Papin interpreter to detect and remove duplicate columns.

Second, we realized that, in all these case studies, we copied first the corresponding QCF in the DOF editor and then we started to modify it adding data feeders and *realizes* statements. It would be interesting if this process could be automated and an initial skeleton of a DOF, with even the *realizes* statements automatically filled up, could be generated from a QCF. We will try to address this issue as part of our future work.

Third, we found the need, in some cases, of adding custom columns to the output dataset containing the result of a calculation on domain data. For example, instead of adding information about each training course of a worker individually, it might be better to add a single column containing all the hours of training this worker has received, for which we would need to sum the hours of all his training courses. To do it, we need both a mechanism to define custom columns and primitives and aggregation functions to perform calculations on domain data. As our languages rely on Lavoisier and this language has support for this calculated values, it was easier to incorporate this kind of column to our languages. However, the current catalogue of operators and aggregated functions provided by Lavoisier is somewhat limited, so it would be advisable to extend it. For instance, it would be interesting to have operators to perform calculations with dates. We will address this issue as part of our future work.

Finally, we found several causes in the fishbone diagrams of these case studies for which it might be really hard to find data in a domain model, such as, for instance, worker motivation or machine positioning in production plants. Curiously, most of these causes are included in the *Management* category. DOFs provide a *not mapped* statement for these cases. However, a DOF full of *not mapped* statements would be useless. So, fishbone diagrams should be designed having into account that causes should be measurable or characterized somehow. This is aligned with the original thoughts of Kaoru Ishikawa, the creator of these diagrams, who stated that causes in a fishbone diagram are only useful if we can intervene on them and analyze the influence of such interventions on the effect.

5.2. Experiments with industrial engineers

In order to conduct a sound and conclusive analysis of whether our languages can be used effectively by industrial engineers, we should have conducted some empirical experiments in which some carefully selected industrial engineers performed some dataset generation tasks using our languages in a controlled environment under our supervision. According to the literature [79,80], these empirical experiments are one of the most effective ways to evaluate the benefits of using a DSL [79,80]. Unfortunately, however, these types of experiments are very rare in both the DSL and Industry 4.0 communities [81,82]. The main problem with these experiments, and probably the reason why they are not very common in the community, is that their design, organization, and execution require a huge effort in terms of both time and resources, which usually exceeds the limits of a regular research project. Moreover, in the case of Industry 4.0, the participants are industrial engineers, who often have complex schedules, very little time to participate in these experiments, and serious problems to leave their workplaces during their working hours. Furthermore, at the time of writing this work, the Covid-19 health crisis made it more difficult to organize empirical experiments that require bringing together people with different profiles and from different organizations in the same room.

For these reasons, we opted for a more simple procedure based on distributing an online questionnaire to a set of industrial engineers, who had to solve some exercises related to Papin. We focused exclusively on Papin since this is the language that industrial engineers would use. In this regard, it should be remembered that data-oriented fishbone models are created mainly by data scientists. This questionnaire was divided into six sections:

[11] https://doi.org/10.6084/m9.figshare.24581361.v1.

Table 3
Questions 5.1 and 5.2 of the questionnaire.

Q5.1 Given the Ishikawa diagram and the Papin specification attached to this question, would the cause *1.3.2 Wrong Housing Model* be included in the output dataset? Briefly reason your answer.

```
dataset Question5.1
  using FallingBand {
    include Material {
          include UnsuitableShaft
        }
        include Machine
  }
```

Q5.2 Given the Ishikawa diagram and the Papin specification attached to this question, would the cause *1.2.3 The powder coating not within the norm* be included in the output dataset? Briefly reason your answer.

```
dataset Question5.2
  using FallingBand {
        include Material {
          include UnsuitableShaft
        }
        include Machine
 }
```

Note: The Ishikawa diagram mentioned in all these questions is the diagram of Fig. 6.

Table 4
Questions 5.3 and 5.4 of the questionnaire.

Q5.3 Given the Ishikawa diagram and the Papin specification attached to this question, indicate the numbers of all the causes that would be selected by that specification. Briefly reason your answer.

```
dataset Question5.3
  using FallingBand {
    include Man
        include Machine
  }
```

Q5.4 Given the Ishikawa diagram and the Papin specification attached to this question, indicate the numbers of all the causes that would be selected by that specification. Briefly reason your answer.

```
dataset Question5.4
  using FallingBand {
    include Man
        include Material {
          include UnsuitableShaft {
            include InadequateParameters
                include WrongShaftModel
          }
        }
  }
```

Note: The Ishikawa diagram mentioned in all these questions is the diagram of Fig. 6.

Table 5
Questions 6.1 and 6.2 of the questionnaire.

Q6.1 Given the Ishikawa diagram attached to this question, write a Papin specification to analyze the *Method* and *Management* dimensions. You can use the following Papin specification as an example of the language syntax.

```
dataset FallingBand_MaterialsAndPressures
  using FallingBand {
    include Material
        include Machine {
          include PneumaticPliersPressure
        }
  }
```

Q6.2 Given the Ishikawa diagram attached to this question, write a Papin specification to analyze only the causes *2.2 Calibration not carried* and *3.1 Failure of pnematic pliers*. You can use the following Papin specification as an example of the language syntax:

```
dataset FallingBand_MaterialsAndPressures
  using FallingBand {
    include Material
        include Machine {
          include PneumaticPliersPressure
        }
  }
```

Note: The Ishikawa diagram mentioned in all these questions is the diagram of Fig. 6.

1. General information about the questionnaire, data protection issues and asking for agreement to participate in the experiment.
2. Demographic data, e.g. academic degrees, job positions, experience.
3. Respondents' background on data modeling and fishbone diagrams.
4. A training section with a brief video tutorial about Papin, plus two control questions to check if the respondent understood the tutorial.
5. Three questions to verify if respondents were able to understand Papin specifications.
6. Two small exercises where the respondents had to write a Papin specification.

The questions and exercises of sections 5 and 6 are contained in Tables 3, 4 and 5. The Ishikawa diagram mentioned in all these questions is the diagram of Fig. 6. The complete questionnaire can be found in Appendix B.

Questions of section 5 aim to verify whether industrial engineers are able to interpret correctly Papin specifications, and to reason about them to reach a conclusion. In these questions, we requested a brief rationale together with each answer to verify that the answers were actually consequence of a correct deduction and not simply a lucky bid. Exercises of section 6 try to check whether industrial engineers are able to create Papin specifications by themselves. The first exercise asks for the writing of a very simple specification and it can be considered as a warn-up exercise whereas, for the second one, respondents must write a more complex specification that requires a deeper knowledge of Papin.

Before disseminating the questionnaire, we carried out a pilot experiment with a naval engineer who is close to us. Based on his feedback, we adjusted the questionnaire and circulated it between different groups of industrial engineers. These groups included engineers from the industries surrounding our university, engineers we knew from other Spanish regions, and members of the departments of mechanical and chemical engineering of our university. The questionnaire was opened during three weeks and it took around 30 min to be completed. We gathered 16 answers during these three weeks after an intensive activity to publicize the questionnaire. From these 16 answers, one was discarded because the respondent failed the control questions about Papin, which indicated that something went wrong during the training section and the respondent had not the required skills to complete the questionnaire. Another answer was discarded because the respondent did not fill sections 5 and 6 arguing he did not have time.

The respondents included industrial engineers (10), telecomunications engineers (1), chemical engineers (1), chemists (1), and naval engineers (1). In addition, 7 of them had more than 20 years of engineering experience, 2 between 10 and 20 years, 1 between 2 and 5 years, and 4 less than 2 years. The current job positions included project managers, maintenance manager of a set of logistics centers,

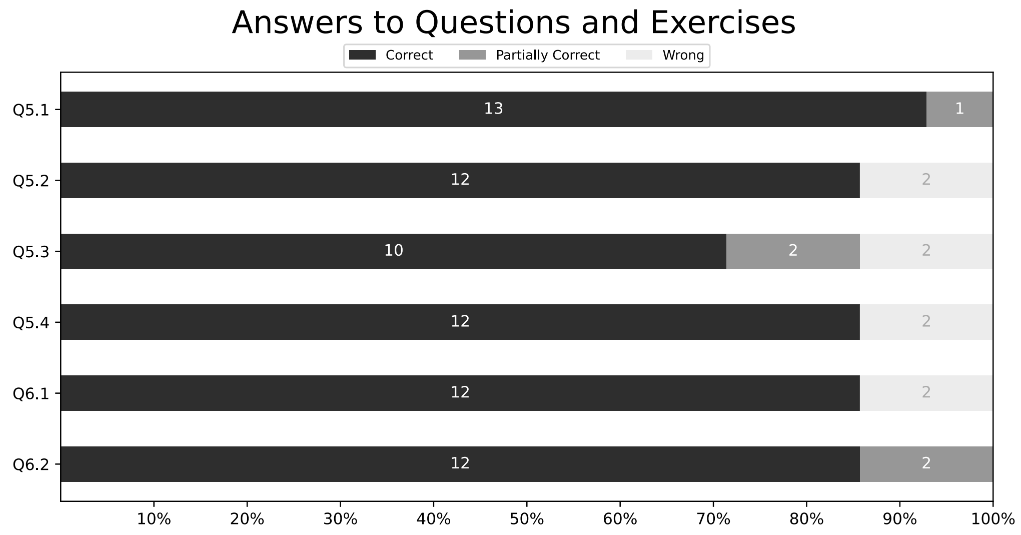

Fig. 10. Answers to the questions and exercises of the questionnaire.

Chief Technical Officer of a ship building company, head of technical office, some professors and one Ph.D. student. Only two respondents had some previous experience with data modeling notations. These notations were UML [83] and Entity-Relationship (ER) [66] models.

After gathering the answers, we marked them as *correct*, *partially correct* and *wrong*. Correct answers have both a right answer and a right rationale. Partially correct answers have a right rationale, but there is a small incorrect detail in the answer. For instance, in question 5.3, there is a missing cause. Fig. 10 shows the results after the marking process.

As it can be seen, most engineers were able to answer to these questions and exercises correctly. With the exception of Q5.3, all questions have more than 80% of correct answers, and 3 out of 6 questions were answered correctly by all participants, which is quite positive. Concerning Q5.3, the respondents had to list five causes, corresponding to the *Man* and *Machine* categories. Two respondents provided a partially correct answer. While they forgot to include the cause 3.3 from the Machine category, the other four causes were correctly included. This issue may be related to the layout of the fishbone diagram or to the rush to finish. Regarding the wrong answers of Q5.3, one respondent forgot completely a category and the other one answered that no cause was selected. The first wrong answer may be due to the rush to finish, whereas the second one may be related to the fact that we are actually selecting categories in this case and not causes. Nevertheless, we marked both answers as wrong. Other minor problems appeared in other questions. For example, in question 6.1, a respondent provided a list of selected causes as answer, instead of a Papin specification, which indicates he did not read the question carefully.

So, considering the number of correct answers and the lack of wrong answers in half of the questions, we may conclude that there is some preliminar evidence indicating that industrial engineers are able to use Papin effectively. However, to state this conclusion firmly, we would need to conduct more complex and better monitored experiments, since there are some obvious threats to the validity of our conclusion that are detailed in Section 5.4.

5.3. *Accidental complexity reduction*

5.3.1. *Evaluation procedure*

Here we show our efforts to measure how much accidental complexity can be saved by using *Data-Oriented Fishbone Models* and *Papin* for data selection and preparation. For this, we applied the same procedure that was executed to evaluate Lavoisier [52]. This procedure was based on defining a set of dataset extraction scenarios that represented all potential situations a data scientist might face during the creation of a dataset. In our case, these scenarios will be based on the falling bands case study, which have also been used as running example throughout this paper. Then, we write scripts for each scenario using different technologies and our languages, we gather a set of metrics for each script and we compare the results. So, to execute the procedure, we need three elements: (1) the technologies to be included in the comparison; (2) the set of metric to be calculated; and, (3) the scenarios for which scripts would be written in each language. We comment on each one of these elements below.

Candidate technologies. As it was mentioned in previous sections, and stated by de la Vega et al. [52], the technologies typically used for dataset creation can be classified in two different groups: (1) data-management dedicated languages; and, (2) data-management libraries for general-purpose programming languages. So, we replicated de la Vega et al.'s [52] decisions and we chose one representative from each group, selecting the SQL language [47] for the first group, and Python Pandas [49] for the second one, as de la Vega et al. did [52]. We also included Lavoisier as a third language for the comparison as we wanted to evaluate the advantages and disadvantages of our new approach as compared to this cutting-edge language.

As the reader can easily notice, neither SQL or Pandas are able to work with object-oriented models. Therefore, for these cases, we built a relational model that were equivalent to the domain model of the falling bands case study and we wrote the SQL and Pandas scripts using it. To carry out this transformation, we follow the classical object-relational mapping patterns [84,85]. The SQL scripts for creating the relational model resulting from this transformation process can be found in the supplementary material of this article.

Metrics. We used the same set of metrics that was used for the Lavoisier evaluation [52] and we adopted the same decisions as them. So, we use script size as an indirect and rough measure of accidental complexity: the more accidental complexity a script has, the more boilerplate code it may contain and the larger its size will be. We measured script size in characters, to focus in the amount of code a data scientist would need to write, and we ignored tabs, spaces and line breaks for this count, to avoid the results were affected by differences in the coding style. We also excluded from this metric the code required to transform the tabular structures resulting of the SQL and Pandas operations into a CSV file, to focus just in the accidental complexity of the data transformations that are required to create a dataset.

To measure script size of DOFs plus Papin, we needed to decide how DOF specifications were exactly computed in this case. With these technologies, we always need to write a Papin specification to generate a dataset, but DOFs may be reused, so they do not need to be written in all cases. Therefore, we considered three different options. The first one was to include the whole DOF, with all its causes, in the script size, independently on whether a cause was useful for generating a concrete dataset or not. This option would correspond to the case where a complete DOF is initially created before generating any dataset and, once the DOF is written, a Papin specification is then created. We will refer to this option as *Full DOF*. The second option was to consider just the causes of a DOF that are selected from a Papin specification. This option matches the situation where the DOFs is completed on demand, writing just those causes that are required by each Papin specification. We will refer to this option as *Data Feeders*. The third option was to completely exclude the DOF from the script size. This option would happen when a DOF is already available in a domain and we simply need to write a new Papin specification to create a dataset. We will refer to this option as *Just Papin*. As we wanted to get better insights into strengths and weaknesses for our approach, we took measures for the three options.

As previously mentioned, script size is a rough estimator for the presence of accidental complexity in the scripts for dataset generation. However, script size might be affected by different factors in addition to accidental complexity. A lower script size might be a consequence of less verbose syntax, which, on the other hand, might be harder to understand in some cases. This argument can be illustrated with the use of the conditional ternary operator (i.e. *condition? trueBlock : falseBlock*) versus regular if statements (i.e. *if (condition) trueBlock else falseBlock*). The former is less verbose, more compact and quicker to write, but it does not seem to reduce the accidental complexity of an if statement. Both cases can be said to be the same old song and dance, just with a different tune.

Therefore, script size was complemented with an additional set of metrics to gain a better understanding of how accidental complexity works in each case. These metrics are summarized in Table 6 and are taken from the Lavoisier evaluation [52]. *NumOps* and *NumDiffOps* aims to assess the capabilities of the operators of a language. If a language A has to use less operations than a language B to achieve the same goal, i.e. A has a lower *NumOps* than B, the operators of language A might be considered more powerful than the ones of B, since they do more work per operation.

This observation is complemented with the *NumDiffOps* metrics. A higher *NumOps* together with a greater *NumDiffOps* in a language would mean that such a language needs to use more operators to complete the same tasks, so these operators probably work at a lower level of abstraction and we need to concatenate several of them to execute a task that in other languages could be made just with a single one. A higher *NumOps* and similar *NumDiffOps* in a language might imply that the language has to repeat some operations several times to complete a task, which is a typical symptom of repetitive boilerplate code.

NumPar and *AvgParOp* are used to analyze the complexity of invoking an operator in a language. In general, the fewer parameters an operator needs, the less accidental complexity it may contain. For example, when joining two tables, the columns of each table on which this join is performed can be inferred in most cases. If the operators of a language A have capabilities like this, this will result in a fewer number of parameters.

Finally, *NumKw* analyzes accidental complexity in a similar way to *NumOps* and *NumPar*. A lower number of keywords in the scripts of a language A may be a consequence of these keywords being more powerful, as fewer of them are required to fulfill the same objectives. Complementary, *NumDiffKw* aims to measure how many elements we would need to understand and remember to use a language proficiently.

Table 6
Script complexity metrics, taken from de la Vega et al. [52].

Name	Description
NumOps	Number of operations
NumDiffOps	Number of distinct operations
NumPar	Number of parameters
AvgParOp	Average number of parameters per operation
NumKw	Number of keywords
NumDiffKw	Number of distinct keywords

Scenarios. For the scenarios, we also followed the procedure of Lavoisier [52] and first identified all possible atomic scenarios a data scientist might need to deal with during dataset creation. This atomic scenarios, as its name says, cannot be decomposed into simpler ones. We considered that there is an atomic scenario for each pair of type of cause and kind of domain element that might be selected. For example, the selection of a data-linked cause that points to a class would be an atomic scenario, and the selection of a data-linked cause that references a collection would be another scenario. Once these atomic scenarios were identified, we built a concrete scenario, based on our the drive half-shaft case study, for each one of them. The complete domain model and DOF for this case study can be found in Appendix A.

The rationale behind this technique is that more complex scenarios can always be decomposed into a set of atomic ones. Therefore, if our languages are able to reduce accidental complexity of the atomic scenarios, they will also be able to reduce accidental complexity of the more complex ones.

Table 7 shows the atomic scenarios we identified. For each scenario, we provide: (1) a character that identifies it; (2) the kind of DOF cause being selected; (3) the type of domain elements referenced by the DOF causes; (4) the concrete cause being selected; and, (5) the specific domain element included in the output dataset. To reduce the set of cause and domain element combinations, we considered categories and compound causes as equivalent, since categories are a just special cases of compound causes. Moreover, *notMapped* causes were left out as they cannot be used to select data.

All datasets generated in these scenarios contain as base information the identifier of each drive half-shaft and a boolean value indicating whether its wheel band falls or not. In addition, each scenario adds the following data:

Scenario a Wheel band model.

Scenario b Assembly program number.

Scenario c A column for the pressure of each one of the four pliers tightenings.

Scenario d All wheel band batch parameters, and the provider.

Scenario e Coordinate values for the position of the wheel and engine bands.

Scenario f Wheel and engine bands model, all band batch parameters and the providers.

Scenario g Shaft model, all shaft batch parameters, the provider and thickness of the powder coating in the middle and both extremes of the shaft.

As can be seen, some combinations of cause type and model element kind are missing. This is due to the fact we could not found concrete examples for these combinations in any of the five case studies we used to evaluate our work. Concretely, we lack scenarios in which:

1. a data-linked cause references a class and includes a multi-bounded reference of this class;

Table 7
Evaluation scenarios.

Id	Cause kind	Domain element	Selected cause	Selected elements
a	Data-linked	Single attribute	WB_Model	Wheel band model.
b	Data-linked	1-ref (class)	Program	Program.
c	Data-linked	*-ref (collection)	PliersPressure	Pneumatic pliers pressure.
d	Data-linked	Class, 1-refs	WB_Parameters	Wheel band parameters & provider.
e	Compound	Class	BandsPosition	Bands coordinates.
f	Compound	Class, 1-refs	Bands	Band models, parameters & provider.
g	Compound	Class, 1 & *-refs	Shaft	Shaft model, parameters, provider & coating inspections.

2. a data-linked cause references a class and includes both a single bounded reference and a multibounded reference,
3. a compound cause references just a single attribute;
4. a compound cause references just a collection;
5. a compound cause references a class including just a multibounded reference.

We do not include these scenarios because either they are covered somehow by other scenarios or they can be rarely found in practice. For instance, compound cases selecting just a single attribute o just a collection - lacking scenarios listed as (3) and (4), are very rarely found, since compound cases are most likely to select several elements of different kinds at the same time. On the other hand, scenario (1) in the previous list would be equivalent to scenario *c* in Table 7, as the same effort is practically required to navigate to a class and select a multibounded reference of that class than to navigate to a class containing a multibounded reference and select both the class and the reference together.

This way, lacking scenario 1 might be considered equivalent to scenarios *c*, 2 to *g*, 3 to *a*, 4 to *c*, and, finally, 5 to *c*. Moreover, scenarios 4 and 5 are very rare cases that can be hardly found in practice.

Once all the elements for our evaluation were defined, we wrote scripts for each designed scenario using the chosen technologies and we compared them using the set of metrics. These scripts can be found in the supplementary material to this article. Next section discusses the results of this comparison.

5.3.2. Evaluation results

Here we analyze the results obtained after executing the evaluation procedure. First, we will explain why we will discard the *Full DOF* option for the global analysis of the results. Next, we will compare the script size of our approach first with SQL and Pandas, and then with Lavoisier. Finally, we will analyze the collected accidental complexity metrics to try to get better insight into the conclusions drawn from the script size.

Reasons for discarding the Full DOF option. Fig. 11 shows the script size for scenario *a*. As it can be easily noticed, the worst case corresponds clearly to the *Full DOF* option. This was expected as the DOF, as a whole, contains data feeders for selecting a wide range of domain model elements, independently of whether they are required or not to generate a dataset. For example, in the case of scenario *a*, just one data feeder is used, but all of them count for the script size.

To complement this observation, Fig. 12 shows the script size of the *Full DOF* option for all scenarios. As can be seen, the script size is practically constant for each scenario. In all these cases, the DOF is the same and the Papin specification is what changes. However, the differences in the Papin specifications are quite small as compared to the DOF size and the final result remains essentially the same.

Therefore, a first conclusion is that creating DOF specifications is costly. This is due in part to the verbosity of DOF specifications, which include a lot of text dedicated to traceability issues. To measure the weight of the traceability information in a DOF, Fig. 13 shows the size of the DOF for our case study with *realizes* statements for all causes and without any *realize* statement. It is observed that removing the *realize* statements reduces script size by 33%.

So, it can be concluded that adding traceability information is clearly costly, but, on the other hand, it can help to improve DOF comprehension, as the mapping between a QCF and its corresponding DOF is preserved. This information can also be helpful for DOF evolution, since the impact of changes in the QCF may be more easily identified. Therefore, it seems that traceability information creates a trade-off between the cost of creating DOF and the understanding and evolution of the DOF. Since this information is optional, it is up to each team to decide whether to provide it or not.

A solution for reducing the cost of DOF specifications might be to create the data feeders incrementally, as they are required for building datasets. This way, we ensure that all data feeders are used to generate one dataset at least. This situation corresponds precisely to the *Data Feeders* option.

Considering all these arguments, the *Full DOF* is discarded for the rest of this section, as it is clearly worse than the others, and is also removed from all figures, to make the visual comparison of the other options easier.

Script size comparison. Fig. 14 shows the script size for all scenarios. The following issues can be observed:

1. Our approach performs better than SQL and Pandas, except for the scenarios *a* and *b* in the *Data Feeders* option.
2. The *Data Feeders* option is always worse than the *Lavoisier* alternative.
3. The *Just Papin* option is clearly better than the Lavoisier option for all scenarios, except *a*.
4. The size of the *Just Papin* option does not seem to depend on the complexity of the scenarios.

To understand these findings, we must consider that data feeders in DOFs are basically Lavoisier statements embedded in a structure for representing causes which, in addition, might contain traceability information. This scaffolding introduces a noticeable overhead compared to Lavoisier. So, the *Data Feeder* option can be considered like Lavoisier plus something else. This extra element makes this option worse than SQL and Pandas for very trivial scenarios, such as *a* and *b*, and, obviously, worse than Lavoisier for all cases. This explains findings 1 and 2, respectively.

Concerning finding 3, it should be noticed that each Papin clause might potentially invoke several data feeders, which would correspond to several Lavoisier statements. Therefore, these clauses can help to reduce script size as compared to Lavoisier specifications. However, as stated in finding 4, this potential is not related with the complexity of the scenarios. This is because the size of a Papin specification depends on two main elements: (1) how many fine-grained causes need to be selected; and, (2) how deeply these causes are nested in the fishbone diagram. For instance, in Papin, selecting two subsubcauses that belong to different categories is more complex than selecting a category containing dozens of causes.

This explains, for example, why the script size of the scenario *g* is smaller than that of the scenario *a*, even thought *g* is much more complex than *a*. In scenario *g*, we only have to select a compound cause that is nested at level 1, whereas in scenario *a*, we have to navigate to a data-linked cause that is nested at level 2. This also explains why,

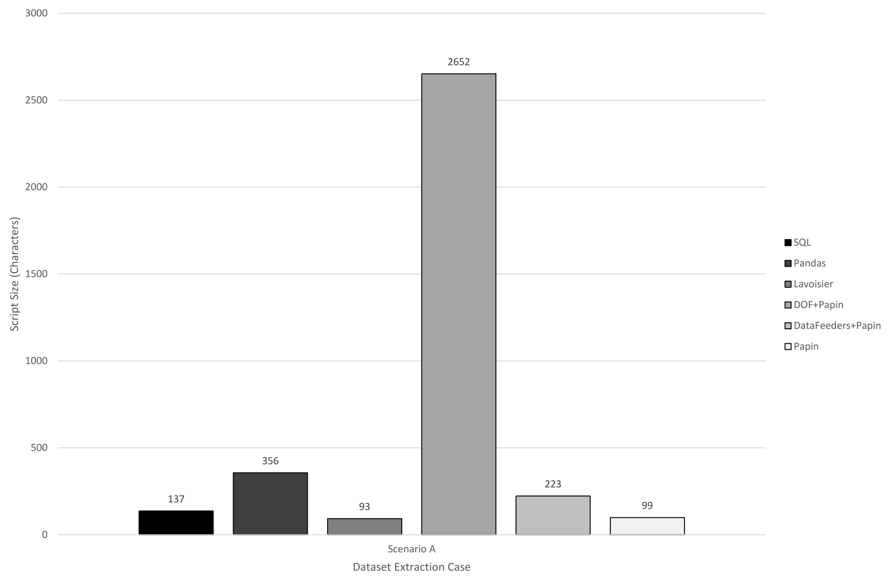

Fig. 11. Scripts size for scenario a.

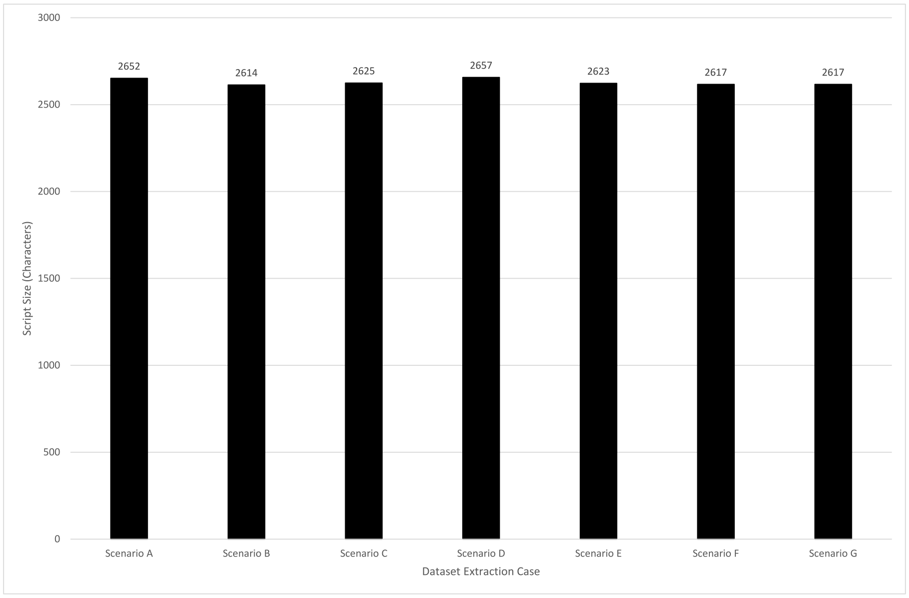

Fig. 12. Script sizes, for all scenarios, of the option that includes the whole DOF.

in scenario *a*, Lavoisier performs better than the *Just Papin* option. Scenario *a* is a very small scenario, rarely found in practice, where only one attribute is added to the output dataset. This dataset can be generated with a very simple Lavoisier specification, whereas in Papin we need to navigate until a level 2 cause.

In general terms, the *Data Feeders* option reduces script size by 16% and 70% as compared to SQL and Pandas, respectively, achieving reductions up to 54% and 82%. The *Just Papin* option achieves reductions by 72% and 87% on average, and up to 90% and 93%, as compared to SQL and Pandas, respectively. Therefore, it can be stated than our approach keeps the benefits of Lavoisier concerning accidental complexity reduction when compared with SQL or Pandas.

Compared with Lavoisier, the *Data Feeders* option is on average by 50% higher than the Lavoisier alternative, with variations ranging from 35% to 65%. So, it can be concluded that this option offers no advantage over Lavoisier in terms of accidental complexity reduction. On the other hand, the *Just Papin* option performs better than Lavoisier, reducing script size by 50% on average and up to 65% in some cases. So, the *Data Feeders* option performs worse than Lavoisier, but the *Just Papin* option performs better. This means that if we want to generate a dataset in a domain where there is no DOF yet, whether it would be worthwhile to create one depends on how much we expect to use it. Creating a DOF for generating just one dataset would be more costly than using Lavoisier directly, as can be concluded from our data. However, as we use this DOF as the basis for more and more Papin specifications, i.e. to generate more datasets in the same domain, our approach will start to pay off and provide benefits. It should be noted that the latter case is very common in data analysis, where an initial dataset is typically fine-tuned several times to obtain better results.

Finally, we would like to point out that we have obtained script size reduction rates for Lavoisier similar to those of de la Vega et al. [52], which could be taken as an indicator of the soundness of these analyses.

Accidental complexity metrics. Tables 8, 9 and 10 show the results of the accidental complexity metrics. As before, the *Full DOF* option is discarded.

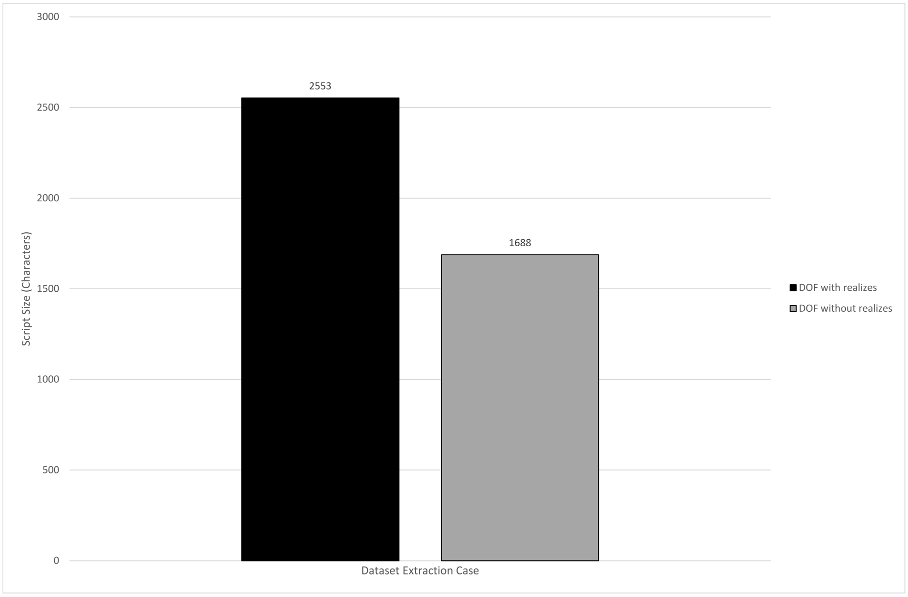

Fig. 13. DOF size including and excluding traceability information.

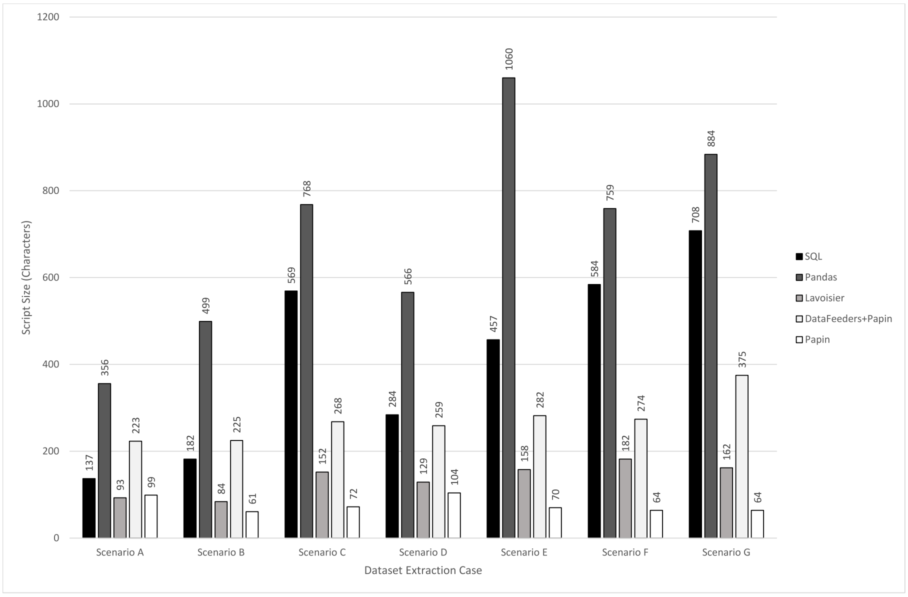

Fig. 14. Script sizes for all scenarios.

Table 8
Measures for the complexity metrics *NumOps* and *NumDiffOps*.

Sc.	NumOps					NumDiffOps				
	SQL	Pan	Lv	DOF	Pap	SQL	Pan	Lv	DOF	Pap
a	3	4	3	7	5	2	2	2	4	2
b	5	4	3	5	3	2	2	2	4	2
c	12	7	5	5	3	3	3	2	4	2
d	5	6	5	7	5	2	2	2	4	2
e	7	8	6	6	3	2	2	2	4	2
f	8	9	8	11	3	2	2	2	4	2
g	12	9	6	9	3	3	3	2	4	2

Table 9
Measures for the complexity metrics *NumPar* and *AvgPars*.

Sc.	NumPar					AvgParOp				
	SQL	Pan	Lv	DF	Pap	SQL	Pan	Lv	DF	Pap
a	7	11	5	12	5	2.3	2.7	1.7	1.7	1
b	11	16	4	10	3	2.2	4.0	1.3	2.0	1
c	24	22	8	11	3	2.0	3.1	1.6	2.2	1
d	15	21	7	12	5	3.0	3.5	1.4	1.7	1
e	20	31	7	11	5	2.9	3.9	1.2	1.8	1
f	16	28	10	12	5	2.0	3.1	1.2	1.1	1
g	40	34	9	15	3	3.6	3.8	2.5	1.7	1

Regarding the number of the operations (*NumOps* in Table 8), the *Data Feeder* option has the higher number of operations in most of cases, with the exception of scenarios *c*, *e* and *g*. On the other hand, the *Just Papin* option generates the lowest number of operations, but scenario *a*, and, as with the script size, this number does not seem to depend on the scenario complexity.

The increase in the number of operations for the *Data Feeder* option is due to the scaffolding for expressing causes, with the exceptions

Table 10
Measures for the complexity metrics *NumKw* and *NumDiffKw*.

Sc.	NumKw					NumDiffKws				
	SQL	Pan	Lv	DF	Pap	SQL	Pan	Lv	DF	Pap
a	6	12	4	9	6	4	7	3	6	3
b	10	17	4	7	4	4	7	3	6	3
c	24	25	7	8	4	6	11	4	7	3
d	10	22	6	10	6	4	7	3	6	3
e	20	31	7	11	4	5	7	3	7	3
f	21	29	9	16	4	5	11	3	7	3
g	22	36	8	18	4	5	11	4	8	3

are scenarios *c*, *e* and *g*. In the case of *c* and *g*, the reason is that these scenarios deal with collections. In SQL and Pandas, collections are processed by concatenating a join operator with a pivot, as described in Section 2.5. On the other hand, this concatenation is abstracted into a single operator both in Lavoisier and in our approach, which helps to reduce the number of operations. Moreover, SQL does not provide direct support for pivot operations, so they need to be indirectly performed by means of other strategies, like concatenating nesting queries (see Listing 2 for an example). This is why the number of operations is so high for the scenarios *c* and *g* in the SQL case.

Concerning the *Just Papin* option, the number of operations again depends mainly on how many fine-grained causes nested at deep levels we need to select, rather than on the number of elements to include in the dataset. This can be easily observed in scenarios *a* and *d*, where only one cause nested at level 2 is selected. Therefore, we can state that the benefits of the *Papin* option can be related to a reduction on the number of operations, as compared to the other alternatives. Moreover, this reduction depends mainly on the deep of the causes to be selected instead of other elements.

On the other hand, the *Data Feeder* option inherits some benefits of Lavoisier regarding the management of collections, but this is not enough to explain the script size reductions of this option as compared to SQL and Pandas, since the number of operations is similar for the three alternatives in most of the cases.

The analysis of the metric about the number of different operations (*NumDiffOps* in Table 8) shows that the advantages of Lavoisier and our approach do not seem to be generally related to the need to use lower number of different operations in each script. The number of different operations is similar for all the options, but the one for *Data Feeder*, due to the additional operations for managing causes. There is only a slight increase in this metric for the scenarios *c* and *g* in the SQL and Pandas options. This increase is due to the need to use pivots to process collections.

Regarding the number of parameters (Table 9), we can observe several issues. First, the *Data Feeders* option requires a lower number of parameters than SQL and Pandas, excepting for the trivial scenario *a*. Second, the *Data Feeders* option requires more parameters than Lavoisier; and, third, the *Just Papin* option performs better than the other alternatives.

The first finding seems to indicate that the lower script size of the *Data Feeders* option, as compared to SQL and Pandas, seems to be mainly related to a reduction in the number of parameters of each data transformation operation. For example, to include the *ComplianceReport* class (see Fig. 4) in a dataset, we would have to write something like *LEFT JOIN ComplianceReport cr ON dhs.idComplicanceReport = cr.id* in SQL, and *merge(reports, left_on = "idComplianceReport", right_on = "id")* in Pandas. In addition, both in Pandas and in SQL, we may need to rename some attributes of *ComplianceReport* to avoid name collisions after the join; and then filter the results to remove the identifiers of the joined table (e.g., *ComplianceReport.id*) as well as foreign keys used for the join (e.g., *DriveHalfShaft.idComplianceReport*) from the output dataset. All this information is provided as parameters to the join and merge operators in SQL and Pandas. On the other hand, to perform the same operation in both Lavoisier and our approach, we only need to write *include ComplianceReport*, and the language itself is able to infer the parameters required for joining the classes, as well as to perform the attribute renaming and filtering automatically. This helps to reduce the number of parameters in Lavoisier and the *Data Feeder* option.

Concerning the second observation, the increase in the number of parameters for the *Data Feeders* option is due to the scaffolding for associating causes with data elements. The third finding, the reduction in the number of parameters for the *Just Papin* option, complements the observation of the number of operations. It shows that the *Just Papin* option is simpler not only because it requires fewer operations, but also because those operations also need fewer parameters.

Finally, concerning the number of keywords metrics (Table 9), we can observe that the previous phenomena repeat themselves: the *Data Feeder* option requires fewer keywords than SQL and Pandas, but more than Lavoisier; and the *Papin* option is the one with the lowest numbers. However, in terms of the number of different keywords, SQL has values close to Lavoisier and the *Just Papin* option, and lower than the *Data Feeder* one. This means that SQL is a concise language where a few keywords are used over and over again.

The increase in the number of keywords in SQL and Pandas is connected to the additional parameters that we need to provide when using these technologies. For instance, to perform a join in Pandas, we need to use the *left_on* and *right_on* keywords to specify the columns over which the join will be performed. The increase in the number of keywords of *Data Feeder* option as compared to Lavoisier and the *Just Papin* option is due to the scaffolding for dealing with causes. Finally, the *Just Papin* option outperforms the others because Papin is a very simple language with only three keywords. Therefore, we expect it to be easily adopted by industrial engineers, although we need to verify this empirically as part of our future work.

In summary, as a result of this evaluation process, we can conclude that our approach preserves the benefits of Lavoisier and performs better than SQL and Pandas in both the *Data Feeder* and *Just Papin* options. This is due to a better management of collections and to the capability of automatically inferring certain parameters. On the other hand, the *Full DOF* option is worse because we have to write Lavoisier statements that may never be used. Compared to Lavoisier, our approach is worse when we must build a new DOF or a new data feeder (the *Data Feeder* option), and it is better when we can reuse these elements (the *Just Papin* option). Moreover, Papin is particularly good at generating dataset composed of coarse-grained causes, such as categories, and performs worse when these datasets contain fine-grained causes nested at deep levels. So, we can state that our approach retains the advantages of Lavoisier in terms of accidental complexity reduction and it outperforms Lavoisier in some contexts.

5.4. Threats to validity

Here we analyze different threats to the validity of our conclusions. We do it separately by each one of the elements evaluated in this section.

5.4.1. Expressiveness

We did not find major expressiveness problems for the five case studies we used in the evaluation, but it may be that other case studies exhibit some issues. This is a classic problem in software language engineering, where the expressiveness of a language is very difficult to assess in a universal way. Indeed, research on software languages can be seen, roughly speaking, as the process of finding expressiveness problems in languages and then providing solutions to these problems. For example, aspect-orientation [86] can be seen as a solution to adequately express crosscutting concerns in object-oriented languages.

However, while this threat cannot be completely eliminated, we have tried to mitigate it by using external and heterogeneous case studies that correspond to real industrial problems, so that these case studies: (1) cover a wide range of possible situations; (2) are not biased by us; and (3) represent real industrial problems and not toy examples artificially fabricated in research labs. On this last point, we would like to point out that although the four case studies are drawn from the literature, they correspond to problems faced by different real companies. On the other hand, it could be perfectly argued that the fishbone diagrams were provided by the case studies, but the domain models were created by us, and, consequently, these domain models might be biased. As it was commented in Section 2.7, domain models are rarely used in industrial settings, so finding case studies corresponding to real industrial problems that provided a fishbone diagram and a domain model at the same time is an unfeasible task in practice. Therefore, we decided to create the domain models for each case study using only the description of the data reported as available in the corresponding case study. In addition, to reduce potential bias, we asked several experts and practitioners to review the domain model we created. None of them reported any problems with these models.

5.4.2. Experiments with industrial engineers

Regarding the experiments to test whether industrial engineers were able to use Papin, we used a single case study for these experiments. Therefore, in order to be scientifically rigorous, we could not generalize the conclusions of these experiments to other case studies, as we have no evidence that industrial engineers can use Papin with other case studies. Nevertheless, this case study is taken from a real case study, it is not a trivial one, and it has the degree of complexity expected in real scenarios. So, there are no peculiarities that might make us think that engineers could have problems using Papin in other case studies. However, to be rigorous, we should repeat these experiments with other case studies, and we will address this as part of our future work.

Second, we had no control over how respondents answered the questions. Therefore, these respondents could have cheated on the questionnaire. However, the respondents did not receive any reward for answering the test correctly, and they were informed about the importance of being honest and reliable in their answers, so there is no reason to believe that they cheated. On the other hand, respondents may have answered the questions under poor personal conditions or in an inappropriate environment. For example, we know that some respondents filled out the questionnaire late at night, just before going to bed. Therefore, they may have been exhausted after a long day at work and not in the best conditions to learn something new and then do some exercises. Other respondents completed the questionnaire on a cell phone while commuting to work. Thus, they had to read the diagrams on a small screen and elaborate the answers in a noisy and uncomfortable environment. Therefore, some respondents may have answered some questions incorrectly due to these conditions. However, this means that our results could be even better than they are if we had been able to control for these external elements.

Similarly, we could not interact with the respondents while they were completing the questionnaire. Thus, we could not resolve any questions they might have about Papin or clarify any doubts they might have about the exercises in the questionnaire. As before, this may have caused some respondents to incorrectly solve some exercises that they might have done correctly if they had been able to interact with us. So, again, our results could be even better than they are.

A more serious threat is that we used the same case study for the training session and the exercises. As a result, some respondents may have correctly solved some exercises by simply imitating a similar one we used in the training session to illustrate a concept. In this case, the answer would be correct, but the respondent may not understand how Papin actually works. We decided to use the same case study for the training and the exercises to reduce the time needed to complete the questionnaire and thus encourage participation. It was a real challenge to find industrial engineers willing to spend 30 min to complete the questionnaire. If we had extended this time to include a different case study for solving the exercises, we are quite sure that the number of people who completed the questionnaire would have decreased significantly. To try to mitigate this threat, we carefully designed the exercises so that they could not be solved by simply copying, pasting, and adjusting an existing example.

5.4.3. Accidental complexity reduction

It could be alleged that the results of accidental complexity reduction are a consequence of the scenarios we designed for the comparison, but that other sets of scenarios might have yielded different results. As described in Section 5.3.1, these scenarios were systematically designed to capture all situations a data scientist encounter when building a dataset. It is also important to note that we used the simplest version of each scenario, rather than opting for more complex versions that would have provided much better results for our approach, given the conclusions we reached. For example, we did not use any scenario that required processing dozens of collections, which are cases where our approach and Lavoisier could have obtained excellent results.

In addition, it may be thought that the evaluation results are influenced by the use of the drive half-shaft case study and that other case studies may yield different values and conclusions. In this regard, we would like to clarify that the complexity of data selection scripts depends on the syntactic structure of the elements to be selected, rather than on what exactly these elements represent. Thus, we used the case study about falling bands to make the data extraction scenarios easier to understand, but we can say that it has no influence on the collected results.

It could be said that the evaluation results are biased by the strategies used to write the SQL or Pandas scripts and that, if other strategies had been adopted, the results might have been different. In this respect, we would like to point out that these scripts were reviewed by a data scientist with sufficient expertise in these languages, and that, when several strategies were available, we always chose the most compact one, i.e., the one with less accidental complexity.

Finally, the comparison of the four selected languages may not be seen as fair because they work against different conceptual data models: SQL and Pandas work at the table level, while Lavoisier and our approach use object-oriented models. Consequently, both Lavoisier and our approach take advantage of working with a data model where some concepts, such as many-to-many relationships, are easier to express. However, this is not a consequence of an badly designed evaluation procedure, but of an explicit decision. First Lavoisier, and then our approach, decided to face the challenges of working with a more abstract kind of conceptual data model in order to take advantage of its benefits. Therefore, the results should be seen as an expected consequence of a conscious design decision, rather than a side effect of poorly designed evaluation process.

6. Related work

This section is structured as follows: firstly, we comment on work that deals with fishbone diagrams and data models. Secondly, we discuss on the different modeling languages that are currently used in manufacturing settings and their relationship with the languages selected for our solution. Next, we analyze works on dataset generation from nested and linked data, and modeling languages for datasets. Finally, we comment on the relationship between dataset generation processes and ETL processes.

6.1. Fishbone diagrams and data models

To the best of our knowledge, this is the first work that enriches fishbone diagrams with data so that datasets can be automatically generated from them. Nevertheless, there exists in the literature other work that connects fishbone diagrams with data models.

For example, Xu and Dang [87] developed an approach to generate fishbone diagrams from databases of problem reports. These reports identify the causes that generate each problem. These reports are automatically analyzed using natural language processing and machine learning techniques to generate as output sets of causes that could lead to each reported problem. Using this information, fishbone diagrams are automatically constructed. So, this approach works in the opposite direction of ours. It starts with a set of well-known cause–effect relationships, expressed in natural language, and builds a fishbone model. In our case, we build a fishbone model to specify causes that could hypothetically lead to an effect. From this fishbone model, we build datasets and use data analysis techniques to try to confirm or reject these hypotheses in order to find the concrete set of causes that actually lead to the effect.

Shigemitsu and Shinkawa [88] propose the use of fishbone diagrams as part of the requirements engineering process for developing systems where the problem to be solved is known, but the causes of that problem are unknown, and therefore the solutions to be implemented for those causes are also unknown. Fishbone diagrams are used to decompose the problem into multiple subproblems. Then, each atomic subproblem is associated with a software function and, following some guidelines, a UML class diagram is generated. However, these diagrams contain only functions and no data, so they are not useful for specifying data available in a domain.

In Yurin et al. [89], a model-driven process for the automated development of rule-based knowledge bases from fishbone diagrams is presented. In this process, rules of a knowledge base are first modeled as cause–effect relationships of a fishbone model. By using fishbone diagrams, domain experts can get a system-wide view of these rules. As in Xu and Dang [87], the cause–effect relationships are well known from the beginning, since they are expert knowledge. In contrast, in our approach, we build datasets to try to find cause–effect relationships that, once discovered, can be used as expert knowledge. On the other hand, similar to Yurin et al. [89], we use fishbone diagrams to provide a global view of the influences between data, so that decisions about what data to include in a dataset can be made more easily.

The inclusion of quantitative information in fishbone diagrams is addressed in Gwiazda [90]. That is, these authors study how to include numerical values in causes using *weighted Ishikawa diagrams*. The weights of the categories are first determined using a form of *Saaty Matrix*. Then, the weights of the causes and sub-causes are calculated by distributing the category weight among them. However, these authors do not associate fishbone models with domain data.

Yun et al. [91] explore the use fishbone diagrams to control the quality of data mining processes. In this case, ribs and bones of fishbone diagrams represent steps of a data mining process. Issues that may affect the quality of the results are identified as causes and attached to the appropriate step. These fishbone diagrams help to visualize a data analysis process, but they do not link causes and data and they cannot be used to generate datasets.

Azzoni et al. [92] have released a tool for transforming CSV files into Ecore models so that the relationships between these elements can be more easily visualized. This tool takes as input a bundle of CSV files and a domain model specified in Ecore, and provides a mechanism for mapping CSV columns to domain model elements. Using this mapping, the tool is able to generate the code needed to load the CSV data into an instance of the Ecore domain model. This work goes in the opposite direction to ours, from existing datasets to object-oriented domain models. Also, this domain model has to be created manually, since this tool is not designed to create it, only to populate it with CSV data.

Finally, *DescribeML* [93,94] is a domain-specific language to describe the contents of a dataset. In this case, the dataset already exists and this tool aims to provide a common language to describe its contents, so that the documentation of datasets becomes more uniform and they can be more easily inspected and compared. In addition, the language provides some features to alert data scientists to issues that may lead to problems with machine learning algorithms, such as datasets with a gender imbalance. This language is used on datasets that have already been created, while we are interested in generating these datasets automatically.

Table 11
Summary of related work.

Work	Inputs	Outputs	Automation degree
[87]	Problem reports	Fishbone model	Fully automated
[88]	Fishbone model	UML class diagram	Manual
[89]	Fishbone model	Knowledge base	Fully automated
[90]	Fishbone model	Weighted fishbone model	Manual
[91]	Fishbone model	KDD process problems	Manual
[92]	CSV files	Ecore model instance	Fully automated
[93]	Dataset	Dataset documentation	Manual
Papin	Fishbone model	Dataset	Fully automated

Table 11 summarizes the contents of this section and compares each previous work with our solution. For each approach we have previously described, Table 11 shows the inputs it accepts, the outputs that it generates and its degree of automation. As it can be easily seen, there is no work, excepting ours, that automatically transforms fishbone diagrams into datasets.

6.2. Models in Industry 4.0

Our solution uses two kind of models: (1) object-oriented models, conforming to the *Ecore* [64] notation, to specify data available in a domain; and (2) fishbone diagrams, as a basis for data selection. However, there is a wide range of alternative models that are also used in Industry 4.0 and that could also have been used for our solution. We comment on these in the following.

Wortmman et al. [82] conducted a comprehensive analysis of the modeling languages used in Industry 4.0 in the form of a systematic mapping study. In this analysis, they identified the following categories of modeling languages in Industry 4.0:

1. UML (*Unified Modeling Language*) [83,95] and its variants (e.g. UML4IOT [96]);
2. SysML (*System Modeling Language*) [97–99] and its variants (e.g., SysML4Modelica [100]);
3. Ontology specification languages, such as OWL (*Web Ontology Language*) [101,102] or SWRL (*Semantic Web Rule Language*) [103];
4. Specific languages for Industry 4.0, such as *AutomationML* [104,105].
5. Ad hoc DSLs (e.g. [106–108]),
6. Other notations, such as EXPRESS [109,110] for product data modeling, process modeling languages [111,112], or value networks [113].

These languages are used for different purposes, ranging from specifying manufacturing processes [114] to describing exchange formats for CAD (*Computer-Assisted Design*) applications [115,116]. In our case, we are interested in two types of languages: (1) those ones that can be used to specify conceptual data models; and (2) languages that are familiar to industrial engineers, not directly focused for data representation but that can be somehow associated with data, and thus used as a basis for data selection. We selected Ecore for the first type, and fishbone diagrams for the second one. In the following, we comment on the alternatives to each one of our selections, and justify the reasons for our concrete choice.

6.2.1. Languages for data modeling

Alternatives for conceptual data modeling include languages such as UML, ER (Entity-Relationship) [66], OWL, and EXPRESS, as well as less popular alternatives. UML [83] is a software modeling standard widely used in the software engineering community and maintained by the *Object Management Group (OMG)*. It covers a wide range of modeling perspectives, including object-oriented structure and behavior modeling, process specification, or IT infrastructure modeling. UML has been widely used for conceptual data modeling [117] as a modern alternative to ER. In our work, we use Ecore, which can be considered as a subset of UML 2.x class diagrams. We chose Ecore simply because it integrates seamlessly with the other tools we selected for building our DSLs, and because its notation can be understood by most software engineers, since it is inspired by UML class diagrams. In fact, many software engineers may do not notice the subtle differences between UML 2.x class diagrams and Ecore.

ER [66] could also be perfectly used to represent data available in a domain, but this notation, oppositely to Ecore, lacks mature tools that can be easily integrated into model-driven environments. OWL [101] is also used for conceptual data modeling in the form of ontologies. OWL is mainly used when we are interested in specifying semantic relationships between data, which is helpful when we need to integrate data from different domains [118], perform inference between data [119] or execute queries at a more semantic level [120]. In our case, ontologies might help in the construction of the object-oriented conceptual data model that represents all the data available in a domain, since this model integrates data coming from different sources in a unified view. In addition, ontologies could also be used to enable more flexible queries in Lavoisier, so that Lavoisier statements do not have to adhere strictly to the syntax of the conceptual data model. We will explore this issue as part of our future work.

EXPRESS [109,110] is a standard for modeling product data in manufacturing environments. EXPRESS is one of the parts of the STEP (*Standard for the Exchange of Product model data*) standard, which covers the full lifecycle of product data. EXPRESS, as UML, ER and OWL, provides elements to specify entities, attributes and relationships between entities, so it can be considered very similar to these languages. Following this reasoning, some publications have designed transformation processes of EXPRESS models into UML [121,122], and of EXPRESS models into OWL [123,124]. Since EXPRESS was designed for manufacturing environments and is similar to other conceptual data modeling languages, it could be argued that the goal of this work could be achieved by simply adapting Lavoisier to work with EXPRESS. The reasons for using fishbone models instead of EXPRESS in this work are twofold.

First, although EXPRESS has been around for a long time, is standardized, and has a large community using and supporting it, there are still some niches where it is not very popular. For example, as part of the questionnaire we used to validate our work (see Section 5.1), we asked industrial engineers in our region if they knew the EXPRESS language. The responses showed that none of them knew EXPRESS, while a noticeable number of them were working or had worked with fishbone models. Of course, we surveyed a small sample, and this problem may be specific to our region, but it can be taken as preliminary evidence that EXPRESS is not known in some places.

Second, to the best of our knowledge, there is no tool that supports the integration of EXPRESS models into model-driven environments, while Ecore can be considered the de facto standard in this context.

Third, we would like to highlight that, as pointed out in our evaluation section, when multiple datasets are constructed from the same conceptual data model, the use of fishbone models provides advantages even for those people who are able to use Lavoisier directly against data models.

Despite the above, as part of our future work, we will study the existing interest and required effort to adapt our work to other conceptual data modeling languages, such as OWL or EXPRESS, so that everyone can choose the option that best suits their needs. As mentioned before, these languages are more or less similar to Ecore, and different transformations have been defined between them. Therefore, adapting Lavoisier to these new data modeling languages can be done in two steps. First, we would need to change the Lavoisier syntax to be able to select data from these alternative data modeling languages. Then, we could either define new interpreters for these new Lavoisier flavors or, alternatively, transform the corresponding conceptual data models into Ecore and, using the traces of these model transformations, transform the new Lavoisier queries into Ecore-based queries and interpret them.

6.2.2. Well-known modeling languages in industry

There is a considerable variety of modeling languages, not related to data modeling, which are often used in industrial environments. For example, SysML [97,98] is widely used in automotive engineering [125], avionics [126] or robotics [127], among others. For the sake of brevity, we will not comment on all these languages here and refer the interested reader to the systematic mapping study of Wolny et al. [99].

Thus, we could have selected one of these languages and applied on it the same strategy we used for fishbone model. For example, we could have designed a technique for associating SysML blocks with domain data and then created a language for selecting the blocks that we want to include in a dataset. However, we chose fishbone diagrams for two reasons. First, in our experience, they are familiar to a reasonable number of engineers. Second, they are easy to learn and understand. As evidence for this statement, we would like to point out that as part of the video we used to train engineers in Papin during the evaluation of our work, we dedicated 2 min to explaining how fishbone models work to those people who could not know them. Some of the engineers who participated in this experiment had no experience with fishbone diagrams, and yet they were able to correctly solve the exercises we proposed. This could be taken as an indicator that a 2-min tutorial may be enough to understand how fishbone diagrams work.

We will study the interest of supporting other languages, such as SysML [97,98] or process modeling languages [107,111,112] as part of our future work.

6.3. Dataset generation

The problem of generating datasets from hierarchical and nested data has been studied by the *propositionalization* [59,128–130] community. Their goal is to convert complex graphs of data into tabular structures that can be used as input to data analysis algorithms. When using these approaches, we typically start by selecting an element to analyze, e.g., *DriveHalfShaft*. Then, the *propositionalization* algorithm randomly creates columns in the output dataset that summarize the relationships between the main element and other elements. To this purpose, these algorithms use functions like *max*, *min* or *average*.

This random process can help to discover values that might have been initially discarded for being considered as not relevant. However, data scientists lose control over exactly what features are included in a dataset because they are selected at random. In addition, elements of collections cannot be analyzed individually because their information is always included using aggregation functions. Therefore, problems related to specific problems in one of these individual instances, such as falling bands due to a too low pressure during the third pliers tightening, may remain hidden in the data. Furthermore, these algorithms have a poor scalability and performance because they need to deal with a huge search space of candidate features.

Finally, there are researchers who have tried to modify data mining algorithms so that they can process nested and hierarchical data. This idea is usually known as *Multi-Relational Data Mining (MRDM)* [60]. Although there are some success stories of *Multi-Relational Data Mining (MRDM)* in some fields, such as time-sequence analysis [131,132], medicine [133] or finance [134], these techniques are still young and, although promising, not as powerful as the classical ones, those that work exclusively with tabular data satisfying the *one entity, one row* constraint.

6.4. Datasets and ETL processes

Dataset creation could be considered a particular case of an ETL (*Extract, Transform, Load*) process [135]. In these processes, data are first extracted from one or more sources, which may be in different formats, then they are transformed as required, and finally loaded into a specific structure that, in our case, would be a tabular structure satisfying the *one entity, one row* constraint. However, this concept has been mainly applied to *Data Warehouses* [136] construction and, to the best of our knowledge, there are no ETL process specifically defined for dataset generation.

Nevertheless, there are some works that, like us, aim to specify ETL processes at a higher level of abstraction. Trujillo and Luján Mora [137] define a UML Profile for some of the most common ETL operations, like the integration and transformation of data which come from different sources. Similarly, in Vassiliadis and Alkis [135], a model is presented to specify those operations that are typically included in ETL processes for DataWarehouses, so that a concrete ETL process can be specified as an instance of this model. Therefore, this work defines a metamodel for ETL processes. These approach inspired the development of further research work, which is analyzed and compared by Munoz et al. [138] in a systematic literature review.

Model-driven approaches have been also used to develop ETL processes, such as in Zineb et al. [139], where the authors introduce a platform-independent model to design ETL processes and describe how it can be transformed into a specific platform.

Some authors have studied how to optimize these ETL processes for specific domains. For example, Wang and Liu [140] created a metadata-based ETL model for managing ETL processes in a mobile sales service company context. Similarly, Przysucha et al. [141] designed and implemented an ETL process for the medical context to retrieve data and load in the Standardized Common Data Model defined by the Observational Medical Outcomes Partnership.[12] Related to the industrial context, in Suleykin and Panfilov [142] a metamodel is proposed to define and implement ETL processes that supported their optimization when working with big amounts of data.

Finally, Hira and Deshpande [143] define a methodology to extract data from tabular structures contained in multiple and heterogeneous sources, like spreadsheets or websites, and load it in multidimensional datawarehouse models. This is the opposite direction of our work, as we go from linked and nested data to tabular structures.

7. Summary and future work

This article has described a set of languages to automate the generation of datasets for data analysis systems in industrial contexts, such as Industry 4.0 applications. This process adapts Lavoisier [52], a language for the automated generation of datasets, to work with fishbone models rather than object-oriented data models. Object-oriented data models are rarely found, but industrial engineers are used to deal with fishbone models, so we considered these models were more suitable than object-oriented models in manufacturing settings.

To adapt Lavoisier to fishbone models, we designed two new languages. First, we created a variant of fishbone models named *Data-Oriented Fishbone* models, which can be used to represent influence relationships between domain data. In these models, causes are connected to domain data through special code blocks called *data feeders*. In this way, domain data is used to characterize causes in a fishbone model that now specifies influence relationships between domain data. The code of the data feeders is based on Lavoisier, which has been designed to be used by people without expertise in data science [71]. Therefore, the data feeders could even be written by industrial engineers themselves. Nevertheless, this hypothesis has yet to be empirically verified.

[12] https://www.ohdsi.org/data-standardization/.

Second, we designed a second language, called *Papin*, to select causes to be included in a dataset. This is a very concise language that just references causes and prevents industrial engineers from having to deal with object-oriented data models. The Papin interpreter processes these specifications, invoking the Lavoisier interpreter to execute the data feeders, and automatically generates the required dataset.

Third, we evaluated the expressiveness of these languages by applying them to five case studies, four of them coming from the literature and the remaining one from an industrial partner. No major issues were reported. In addition, we evaluated whether our approach preserves the advantages of Lavoisier in terms of accidental complexity reduction. It was concluded that our approach outperforms SQL and Pandas in terms of accidental complexity reduction. Compared to Lavoisier, our approach introduces an initial overhead associated with the need to create a DOF. On the other hand, Papin is a very simple language and is able to select a large number of features with very few keywords. Thus, the initial effort associated to the creation of the DOF may not pay off when we generate a small number of datasets from a domain model, but as this number increases, our approach begins to provide benefits.

In summary, it can be stated that our approach helps to reduce the accidental complexity required to create a dataset, which would help to decrease development times, and thus the costs of Industry 4.0 applications. Furthermore, it could help to alleviate the dependency on data scientists, whose fees are often expensive and whose availability may be scarce. Data scientists would still be needed to build object-oriented domain models, but they might not be required to build DOFs and Papin specifications.

As future work, we plan to perform more complex controlled experiments that will allow us to test better whether industrial engineers are actually able to use our languages. We will also investigate if alternative concrete syntaxes might be more suitable than the current textual one. For instance, a graphical concrete syntax could be created using a language workbench such as Sirius, using as abstract syntaxes the existing metamodels of the proposed languages. Alternatively, the current textual syntax can be complemented with graphical views (i.e. no editing capabilities) generated automatically using a tool like *Picto* [144]. Additionally, we will develop facilities to generate the skeleton of a DOF from its corresponding QCF model, to help to reduce the effort associated with building DOFs. We will investigate how to warn users of some problems that may appear when creating a dataset, such as incorporating two columns with a functional dependency between them. We will also try to adapt Lavoisier to data modeling languages beyond Ecore, such as EXPRESS [109,110] or OWL [101,102]. We will explore whether other modeling languages very popular in industrial contexts, such as SysML [97,98] can be used instead of fishbone diagrams, as a basis for data selection. Finally, we will add support to our languages for specifying aggregated values in *data feeders*, so that columns representing the average value of a collection, for example, can be included in a dataset.

CRediT authorship contribution statement

Brian Sal: Writing – original draft, Visualization, Validation, Software, Investigation, Conceptualization. **Diego García-Saiz:** Writing – review & editing, Validation, Resources. **Alfonso de la Vega:** Writing – review & editing, Visualization, Validation. **Pablo Sánchez:** Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

A link to the source code implementation is present in the manuscript.

Acknowledgments

We would like to thank Pablo Víquez and José Luis Lebrero for participating in the pilot study of the empirical experiment with industrial engineers and providing valuable feedback. We would also like to thank Manuel de la Rosa and Gregorio Galán, as well as all the anonymous people who donated some of their time to fill out the questionnaires. Their help was key to a proper evaluation of our approach.

Icons of *library book*, *assembly line*, *industry worker with cap protection and a laptop*, *industrial robot*, *server* in Fig. 3, as well as the *gear wheels* icon in Fig. 8 were created by *SVG Repo*. The person male, the database icon, and the factory building icon were created by Theforgesmith, Solar Icons, and Neuicons of SVG Repo.

Funding

This work was partially supported by the Spanish Government and FEDER funds (AEI/FEDER, UE) under grant TIN2017-86520-C3-3-R (PRECON-I4), and by MCIN/ AEI /10.13039/501100011033/ FEDER "Una manera de hacer Europa" under grant PID2021-124502OB-C42 (PRESECREL).

Appendix A. Drive-half shaft case study

A.1. Domain model

See Fig. 15.

A.2. Data-oriented fishbone model

```
dof FallingBand
  effect FallingBand is
       DriveHalfShaft[id] include report[fallingWheelBand]
  category Material
    cause Bands realizes "Unsuitable band" contains {
      cause WheelBand contains {
        cause WB_Parameters realizes "Inadequate parameter" is
          wheelBand.parameters {include provider}
        cause WB_Model realizes "Wrong band model" is
          wheelBand.model
      }
      cause EngineBand contains {
        cause EB_Parameters realizes "Inadequate parameter" is
          engineBand.parameters{include provider}
        cause EB_Model realizes "Inadequate parameter" is
          engineBand.model
      }
    }
    cause Shaft realizes "Unsuitable shaft" contains {
      cause S_Parameters realizes "Inadequate parameters" is
        shaft.parameters {include provider}
      cause SModel realizes "Wrong shaft model" is
        shaft.model
      cause SPowderCoating realize
        "The powder coating not within the norm" is
        shaft.parameters.powderCoatingInspections by zone
    }
    cause Housing realizes "Unsuitable housing" contains {
      cause WheelHousing contains {
        cause WH_Parameters realizes "Inadequate parameters" is
          wheelHousing.parameters {include provider}
        cause WH_Model realizes "Wrong housing model" is
          wheelHousing.model
      }
    }
    cause EngineHousing contains {
      cause WH_Parameters realizes "Inadequate parameters" is
        engineHousing.parameters {include provider}
      cause WH_Model realizes "Wrong housing model" is
        engineHousing.model
    }
  category Man
    cause Program
      realizes "Choosing the wrong program while changing the reference" is
      assemblySession.program
    cause Calibration realizes "Calibration not carried" is
      calculate daysWithoutCalibration as today —
        assemblySession.machine.lastCalibration
  category Machine
    cause PliersPressure realizes
      "Pressure in the pneumatic system too low" is
      assemblySession.parameters.pliersTightenings[pressure] by number
    cause PneumaticPliers realizes "Failure of pneumatic pliers" is
      assemblySession.ReportedErrors.pliersError
    cause Housing realizes "Error of the housing applying station"
      assemblySession.ReportedErrors.housingError
  category Method
    cause BandPosition realizes
      "Incorrect position of the band on the housing" contains {
      cause WheelBandPosition is
        assemblySession.program.parameters.wheelBandPos
      cause EngineBandPosition is
        assemblySession.program.parameters.engineBandPos
    }
    cause HousePosition realizes
      "Incorrect position of the housing on the shaft" contains {
      cause WheelBandPosition is
        assemblySession.program.parameters.wheelHousingPos
      cause EngineBandPosition is
        assemblySession.program.parameters.engineHousingPos
    }
    cause ReFit realizes "Incorrectly performed re—fit" notMapped
    cause ParamFastening realizes
      "Incorrect parameters of the fastening of the band" is
      assemblySession.program.parameters[tighteningForce, tighteningTime]
  category Management
    cause WorkingConditions realizes "Unsuitable work conditions" notMapped
    cause UnitPlacement
          realizes "Improper unit placement in the plant" notMapped
```

Appendix B. Experiments questionnaire

This section contains the full questionnaire that was answered by the industrial engineers who participated in the empirical experiment around Papin (see Section 5.2). The original questionnaire was in Spanish, which was the respondent's mother tongue. So, here we provide an English translation of that questionnaire. The original Spanish questionnaire can be found in the supplementary material of this work. All questions that make reference to a fishbone diagram are using the one present in Fig. 6.

B.1. General information and participation agreement

The first section of the questionnaire gave participants an introduction to the experiments context. This section also described the confidentiality and data protection regulations in place for the results of the questionnaire and, lastly, asked for consent to the candidates willing to participate in the study.

B.2. Demographic information

Q2.1 Indicate your academic qualifications.

Q2.2 Indicate your years of experience in an engineering field. (0–2, 2–5, 5–10, 10–20, more than 20).

Q2.3 Indicate your current job position.

Q2.4 Indicate other past job positions of relevance.

B.3. Data modeling knowledge

Q3.1 Did you know of the existence of fishbone diagrams? (Yes or No)

Q3.1.1 (If answer to Q3.1 is yes.) In a scale from 0 to 10, indicate your ability to read and understand a fishbone diagram.

Q3.1.2 (If answer to Q3.1 is yes.) Using the same scale, indicate whether fishbone diagrams are used in your current job environment.

Q3.2 Do you know of the existence of the EXPRESS Modeling notation? (Yes or No)

Q3.2.1 (If answer to Q3.2 is yes.) Using a scale from 0 to 10, indicate whether EXPRESS models are used in your current job environment.

Q3.3 Do you know of any other data modeling notations?

Q3.3.1 (If answer to Q3.3 is yes.) Indicate those known notations.

Q3.3.2 (If answer to Q3.3 is yes.) Indicate which notations you have used in a job environment, now or in the past.

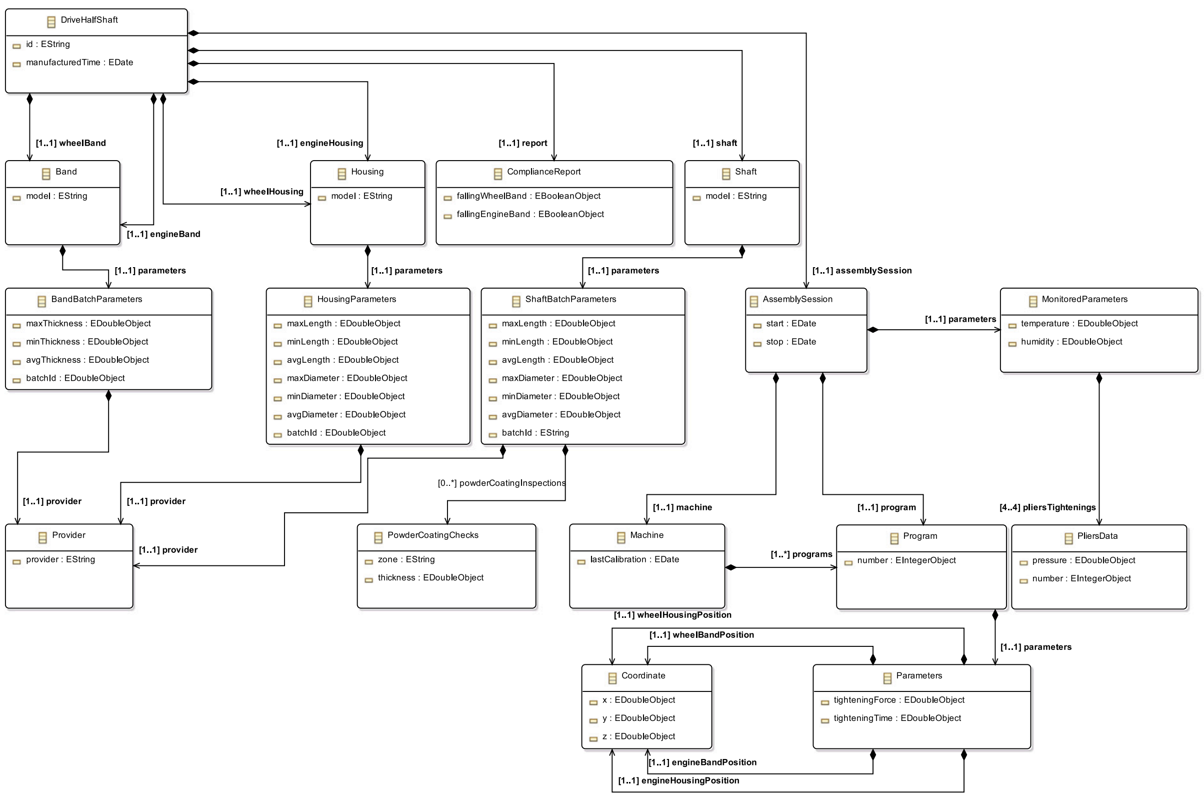

Fig. 15. Domain model of the Drive-Half Shaft case study.

B.4. Training session on Papin

Here we inserted a video explaining some general aspects of our approach and how Papin works. This video can be found at *Omitted for double blind review*. After the video, we make two questions to check that participants understood its contents.

Q4.1 For the generation of datasets, the Papin language allows selecting (a: Classes of an object-oriented diagram; b: fishbones or causes of an Ishikawa diagram; c: table columns).

Q4.2 In an Ishikawa diagram, the Papin language allows selecting (a: Categories and causes of any level and sublevel; b: Only categories, such as Man or Machine; c: Categories and causes, but only of the first level).

B.5. Understanding Papin specifications

Q5.1 Given the Ishikawa diagram and the Papin specification attached to this question, would the cause *1.3.2 Wrong Housing Model* be included in the output dataset? Briefly reason your answer.

```
dataset Question5.1
  using FallingBand {
    include Material {
          include UnsuitableShaft
        }
        include Machine
  }
```

Q5.2 Given the Ishikawa diagram and the Papin specification attached to this question, would the cause *1.2.3 The powder coating not within the norm* be included in the output dataset? Briefly reason your answer.

```
dataset Question5.2
  using FallingBand {
        include Material {
          include UnsuitableShaft
        }
        include Machine
  }
```

Q5.3 Given the Ishikawa diagram and the Papin specification attached to this question, indicate the numbers of all the causes that would be selected by that specification. Briefly reason your answer.

```
dataset Question5.3
  using FallingBand {
    include Man
        include Machine
  }
```

Q5.4 Given the Ishikawa diagram and the Papin specification attached to this question, indicate the numbers of all the causes that would be selected by that specification. Briefly reason your answer.

```
dataset Question5.4
  using FallingBand {
    include Man
        include Material {
          include UnsuitableShaft {
            include InadequateParameters
                include WrongShaftModel
          }
        }
  }
```

B.6. Writing Papin specifications

Q6.1 Given the Ishikawa diagram attached to this question, write a Papin specification to analyze the *Method* and *Management* dimensions. You can use the following Papin specification as an example of the language syntax.

```
dataset FallingBand_MaterialsAndPressures
  using FallingBand {
    include Material
        include Machine {
          include PneumaticPliersPressure
        }
  }
```

Q6.2 Given the Ishikawa diagram attached to this question, write a Papin specification to analyze only the causes *2.2 Calibration not carried* and *3.1 Failure of pnematic pliers*. You can use the following Papin specification as an example of the language syntax:

```
dataset FallingBand_MaterialsAndPressures
  using FallingBand {
    include Material
        include Machine {
          include PneumaticPliersPressure
        }
  }
```

References

[1] Y. Lu, Industry 4.0: A survey on technologies, applications and open research issues, J. Ind. Inf. Integr. 6 (2017) 1–10, http://dx.doi.org/10.1016/j.jii.2017.04.005.

[2] Y. Liao, F. Deschamps, E. de Freitas Rocha Loures, L.F.P. Ramos, Past, present and future of industry 4.0 - a systematic literature review and research agenda proposal, Int. J. Prod. Res. 55 (2017) 3609–3629, http://dx.doi.org/10.1080/00207543.2017.1308576.

[3] N. Karnik, U. Bora, K. Bhadri, P. Kadambi, P. Dhatrak, A comprehensive study on current and future trends towards the characteristics and enablers of Industry 4.0, J. Ind. Inf. Integr. 27 (2022) 100294, http://dx.doi.org/10.1016/j.jii.2021.100294.

[4] K. Ashton, That 'internet of things' thing, RFID J. (2009).

[5] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Comput. Netw. 54 (2010) 2787–2805, http://dx.doi.org/10.1016/J.COMNET.2010.05.010.

[6] F. Mattern, C. Floerkemeier, From the internet of computers to the internet of things, in: K. Sachs, I. Petrov, P.E. Guerrero (Eds.), From Active Data Management to Event-Based Systems and more, in: Lecture Notes in Computer Science (LNCS), Vol. 6462, 2010, pp. 242–259, http://dx.doi.org/10.1007/978-3-642-17226-7_15.

[7] D. Miorandi, S. Sicari, F.D. Pellegrini, I. Chlamtac, Internet of things: Vision, applications and research challenges, Ad Hoc Netw. 10 (2012) 1497–1516, http://dx.doi.org/10.1016/J.ADHOC.2012.02.016.

[8] L.D. Xu, W. He, S. Li, Internet of things in industries: A survey, IEEE Trans. Ind. Inform. 10 (2014) 2233–2243, http://dx.doi.org/10.1109/TII.2014.2300753.

[9] S. Li, L.D. Xu, S. Zhao, 5G Internet of Things: A survey, J. Ind. Inf. Integr. 10 (2018) 1–9, http://dx.doi.org/10.1016/j.jii.2018.01.005.

[10] J. Cheng, W. Chen, F. Tao, C.-L. Lin, Industrial IoT in 5G environment towards smart manufacturing, J. Ind. Inf. Integr. 10 (2018) 10–19, http://dx.doi.org/10.1016/j.jii.2018.04.001.

[11] H. Boyes, B. Hallaq, J. Cunningham, T. Watson, The industrial internet of things (iiot): An analysis framework, Comput. Ind. 101 (2018) 1–12, http://dx.doi.org/10.1016/J.COMPIND.2018.04.015.

[12] J.H. Nord, A. Koohang, J. Paliszkiewicz, The internet of things: Review and theoretical framework, Expert Syst. Appl. 133 (2019) 97–108, http://dx.doi.org/10.1016/J.ESWA.2019.05.014.

[13] G. Aceto, V. Persico, A. Pescapé, Industry 4.0 and Health: Internet of things, big data, and cloud computing for healthcare 4.0, J. Ind. Inf. Integr. 18 (2020) 100129, http://dx.doi.org/10.1016/j.jii.2020.100129.

[14] A.W. Colombo, S. Karnouskos, O. Kaynak, Y. Shi, S. Yin, Industrial cyberphysical systems: A backbone of the fourth industrial revolution, IEEE Ind. Electr. 11 (1) (2017) 6–16, http://dx.doi.org/10.1109/MIE.2017.2648857.

[15] H. Chen, Applications of cyber-physical system: A literature review, J. Ind. Integr. Manage. 02 (03) (2017) 1750012, http://dx.doi.org/10.1142/S2424862217500129.

[16] A. Napoleone, M. Macchi, A. Pozzetti, A review on the characteristics of cyber–physical systems for the future smart factories, J. Manuf. Syst. 54 (2020) 305–335, http://dx.doi.org/10.1016/j.jmsy.2020.01.007.

[17] Y. Wei, M.B. Blake, Service-oriented computing and cloud computing: Challenges and opportunities, IEEE Internet Comput. 14 (6) (2010) 72–75, http://dx.doi.org/10.1109/MIC.2010.147.

[18] T. Erl, R. Puttini, Z. Mahmood, Cloud Computing: Concepts, Technology, & Architecture, Prentice Hall, 2013.

[19] L. Arockiam, S. Monikandan, G. Parthasarathy, Cloud Computing: A Survey, Int. J. Comput. Commun. Technol. 8 (2017) 21–28, http://dx.doi.org/10.47893/IJCCT.2017.1393.

[20] S.-C. Huang, S. McIntosh, S. Sobolevsky, P.C.K. Hung, Big data analytics and business intelligence in industry, Inf. Syst. Front. 19 (6) (2017) 1229–1232, http://dx.doi.org/10.1007/s10796-017-9804-9.

[21] Y. Cheng, K. Chen, H. Sun, Y. Zhang, F. Tao, Data and knowledge mining with big data towards smart production, J. Ind. Inf. Integr. 9 (2018) 1–13, http://dx.doi.org/10.1016/j.jii.2017.08.001.

[22] E. Hämäläinen, T. Inkinen, Industrial applications of big data in disruptive innovations supporting environmental reporting, J. Ind. Inf. Integr. 16 (2019) 100105, http://dx.doi.org/10.1016/j.jii.2019.100105.

[23] L.D. Xu, L. Duan, Big data for cyber physical systems in industry 4.0: a survey, Enterp. Inf. Syst. 13 (2) (2019) 148–169, http://dx.doi.org/10.1080/17517575.2018.1442934.

[24] M. Javaid, A. Haleem, R.P. Singh, R. Suman, Significant applications of big data in industry 4.0, J. Ind. Integr. Manage. 06 (04) (2021) 429–447, http://dx.doi.org/10.1142/S2424862221500135.

[25] P.M. Seeger, Z. Yahouni, G. Alpan, Literature review on using data mining in production planning and scheduling within the context of cyber physical systems, J. Ind. Inf. Integr. 28 (2022) 100371, http://dx.doi.org/10.1016/j.jii.2022.100371.

[26] F. De Pace, F. Manuri, A. Sanna, Augmented reality in industry 4.0, Am. J. Comput. Sci. Inf. Technol. 06 (01) (2018) http://dx.doi.org/10.21767/2349-3917.100017.

[27] R. Palmarini, J.A. Erkoyuncu, R. Roy, H. Torabmostaedi, A systematic review of augmented reality applications in maintenance, Robot. Comput.-Integr. Manuf. 49 (2018) 215–228, http://dx.doi.org/10.1016/j.rcim.2017.06.002.

[28] T. Masood, J. Egger, Augmented reality in support of Industry 4.0—Implementation challenges and success factors, Robot. Comput.-Integr. Manuf. 58 (2019) 181–195, http://dx.doi.org/10.1016/j.rcim.2019.02.003.

[29] O. Danielsson, M. Holm, A. Syberfeldt, Augmented reality smart glasses in industrial assembly: Current status and future challenges, J. Ind. Inf. Integr. 20 (2020) 100175, http://dx.doi.org/10.1016/j.jii.2020.100175.

[30] S.S. Kamble, A. Gunasekaran, H. Parekh, V. Mani, A. Belhadi, R. Sharma, Digital twin for sustainable manufacturing supply chains: Current trends, future perspectives, and an implementation framework, Technol. Forecast. Soc. Change 176 (2022) 121448, http://dx.doi.org/10.1016/j.techfore.2021.121448.

[31] A. Sharma, E. Kosasih, J. Zhang, A. Brintrup, A. Calinescu, Digital Twins: State of the art theory and practice, challenges, and open research questions, J. Ind. Inf. Integr. 30 (2022) 100383, http://dx.doi.org/10.1016/j.jii.2022.100383.

[32] L. Li, B. Lei, C. Mao, Digital twin in smart manufacturing, J. Ind. Inf. Integr. 26 (2022) 100289, http://dx.doi.org/10.1016/j.jii.2021.100289.

[33] I. Fernández del Amo, J.A. Erkoyuncu, R. Roy, R. Palmarini, D. Onoufriou, A systematic review of augmented reality content-related techniques for knowledge transfer in maintenance applications, Comput. Ind. 103 (2018) 47–71, http://dx.doi.org/10.1016/j.compind.2018.08.007.

[34] A. Ceruti, P. Marzocca, A. Liverani, C. Bil, Maintenance in aeronautics in an industry 4.0 context: The role of augmented reality and additive manufacturing, J. Comput. Des. Eng. 6 (4) (2019) 516–526, http://dx.doi.org/10.1016/j.jcde.2019.02.001.

[35] M. Ghobakhloo, N.T. Ching, Adoption of digital technologies of smart manufacturing in SMEs, J. Ind. Inf. Integr. 16 (2019) 100107, http://dx.doi.org/10.1016/j.jii.2019.100107.

[36] S. Sahoo, C.-Y. Lo, Smart manufacturing powered by recent technological advancements: A review, J. Manuf. Syst. 64 (2022) 236–250, http://dx.doi.org/10.1016/j.jmsy.2022.06.008.

[37] M.P. Uysal, A.E. Mergen, Smart manufacturing in intelligent digital mesh: Integration of enterprise architecture and software product line engineering, J. Ind. Inf. Integr. 22 (2021) 100202, http://dx.doi.org/10.1016/j.jii.2021.100202.

[38] B.R. Haverkort, A. Zimmermann, Smart Industry: How ICT will change the game!, IEEE Internet Comput. 21 (1) (2017) 8–10.

[39] J. Kletti, Manufacturing Execution Systems, MES, Springer, 2007.

[40] F. Civerchia, S. Bocchino, C. Salvadori, E. Rossi, L. Maggiani, M. Petracca, Industrial Internet of Things monitoring solution for advanced predictive maintenance applications, J. Ind. Inf. Integr. 7 (2017) 4–12, http://dx.doi.org/10.1016/j.jii.2017.02.003.

[41] S.M. Lee, D. Lee, Y.S. Kim, The quality management ecosystem for predictive maintenance in the Industry 4.0 era, Int. J. Qual. Innov. 5 (1) (2019).

[42] M. Compare, P. Baraldi, E. Zio, Challenges to IoT-enabled predictive maintenance for Industry 4.0, IEEE Internet Things J. 7 (5) (2020) 4585–4597, http://dx.doi.org/10.1109/JIOT.2019.2957029.

[43] G. May, D. Kiritsis, Zero defect manufacturing strategies and platform for smart factories of industry 4.0, in: L. Monostori, V. Majstorovic, S.J.D. Hu, D. Djurdjanovic (Eds.), Proc. of the 4th International Conference on the Industry 4.0 Model for Advanced Manufacturing, AMP, in: Lecture Notes in Mechanical Engineering, 2019, pp. 142–152, http://dx.doi.org/10.1007/978-3-030-18180-2_11.

[44] C. Caccamo, R. Eleftheriadis, M.C. Magnanini, D. Powell, O. Myklebust, A hybrid architecture for the deployment of a data quality management (dqm) system for zero-defect manufacturing in industry 4.0, in: A. Dolgui, A. Bernard, D. Lemoine, G. von Cieminski, D. Romero (Eds.), Proc. of the Advances in Information and Communication Technology, APMS, in: IFIP Advances in Information and Communication Technology, Vol. 632, 2021, pp. 71–77, http://dx.doi.org/10.1007/978-3-030-85906-0_8.

[45] A.A. Nazarenko, J. Sarraipa, L.M. Camarinha-Matos, C. Grunewald, M. Dorchain, R. Jardim-Goncalves, Analysis of relevant standards for industrial systems to support zero defects manufacturing process, J. Ind. Inf. Integr. 23 (2021) 100214, http://dx.doi.org/10.1016/j.jii.2021.100214.

[46] F. Psarommatis, D. Kiritsis, A hybrid decision support system for automating decision making in the event of defects in the era of zero defect manufacturing, J. Ind. Inf. Integr. 26 (2022) 100263, http://dx.doi.org/10.1016/j.jii.2021.100263.

[47] L. Beighley, Head First {SQL}, 0'Reilly, 2007.

[48] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna (Austria), 2020, URL https://www.R-project.org/.

[49] W. McKinney, Data structures for statistical computing in python, in: Proceedings of the 9th Python in Science Conference (SciPy), Austin (Texas, USA), 2010, pp. 56–61, http://dx.doi.org/10.25080/Majora-92bf1922-00a.

[50] E.F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (1970) 377–387, http://dx.doi.org/10.1145/362384.362685.

[51] C.M. Wyss, E.L. Robertson, A formal characterization of pivot/unpivot, in: Proc. of the 14th International Conference on Information and Knowledge Management, CIKM, 2005, pp. 602–608, http://dx.doi.org/10.1145/1099554.1099709.

[52] A. de la Vega, D. García-Saiz, M. Zorrilla, P. Sánchez, Lavoisier: A dsl for increasing the level of abstraction of data selection and formatting in data mining, J. Comput. Lang. 60 (2020) 100987, http://dx.doi.org/10.1016/j.cola.2020.100987.

[53] E. Evans, Domain-Driven Design, Addison Wesley, 2003.

[54] K. Ishikawa, Guide to Quality Control, Asian Productivity Organization, 1976.

[55] N. Dave, R. Kannan, T. Suresh, S.K. Chaudhury, Analysis and prevention of rust issue in automobile industry, Int. J. Eng. Res. Technol. 4 (10) (2018) 1–10.

[56] S.T. Dziuba, M.A. Jarossová, N. Gołębiecka, Applying the Ishikawa diagram in the process of improving the production of drive half-shafts, in: S. Borkowski, M. Ingaldi (Eds.), Toyotarity. Evaluation and Processes/Products Improvement, Aeternitas, 2013, pp. 20–23, Ch. 2.

[57] A. Piekara, S. Dziuba, B. Kopeć, The use of Ishikawa diagram as means of improving the quality of hydraulic nipple, in: S. Borkowski, J. Selejdak (Eds.), Toyotarity. Quality and Machines Operating Conditions, Aeternitas, 2012, pp. 162–175, Ch. 15.

[58] D. Siwiec, A. Pacana, The use of quality management techniques to analyse the cluster of porosities on the turbine outlet nozzle, Prod. Eng. Arch. 24 (24) (2020) 33–36.

[59] A.J. Knobbe, M. de Haas, A. Siebes, Propositionalisation and aggregates, in: L.D. Raedt, A. Siebes (Eds.), Proc. of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD, in: Lecture Notes in Computer Science, Vol. 2168, 2001, pp. 277–288, http://dx.doi.org/10.1007/3-540-44794-6_23.

[60] S. Džeroski, Relational Data Mining, Springer, 2009, pp. 887–911, http://dx.doi.org/10.1007/978-0-387-09823-4_46, Ch. 46.

[61] R.F. Paige, D.S. Kolovos, F.A. Polack, A tutorial on metamodelling for grammar researchers, Sci. Comput. Program. 96 (P4) (2014) 396–416, http://dx.doi.org/10.1016/j.scico.2014.05.007.

[62] A. Kleppe, Software Language Engineering: Creating Domain-Specific Languages using Metamodels, Addison-Wesley, 2008.

[63] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Morgan & Claypool Publishers, 2012, http://dx.doi.org/10.2200/S00441ED1V01Y201208SWE001.

[64] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework, second ed., Addison-Wesley Professional, 2008.

[65] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery in databases, AI Mag. 17 (1996) 37, http://dx.doi.org/10.1609/AIMAG.V17I3.1230.

[66] P.P. Chen, The entity-relationship model - toward a unified view of data, ACM Trans. Database Syst. 1 (1976) 9–36, http://dx.doi.org/10.1145/320434.320440.

[67] T. Hartmann, A. Moawad, F. Fouquet, Y.L. Traon, The next evolution of MDE: a seamless integration of machine learning into domain modeling, Softw. Syst. Model. 18 (2) (2019) 1285–1304, http://dx.doi.org/10.1007/s10270-017-0600-2.

[68] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, Data Mining: Practical Machine Learning Tools and Techniques, fourth ed., Morgan Kaufmann, 2016.

[69] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: J.B. Bocca, M. Jarke, C. Zaniolo (Eds.), Proceedings of the 20th International Conference on Very Large Databases, Morgan Kaufmann, 1994, pp. 487–499, URL http://www.vldb.org/conf/1994/P487.PDF.

[70] Y.S. Koh, N. Rountree, Rare Association Rule Mining: An Overview, IGI Global, 2010, pp. 1–14, http://dx.doi.org/10.4018/978-1-60566-754-6.ch001, Ch. 1.

[71] A. de la Vega, Domain-Specific Languages for Data Mining Democratisation (Ph.D. thesis), Universidad de Cantabria, 2019, URL http://hdl.handle.net/10902/16728.

[72] A. de la Vega, D. García-Saiz, M. Zorrilla, P. Sánchez, On the automated transformation of domain models into tabular datasets, in: Proc. of the ER Forum, in: CEUR Workshop Proceedings, Vol. 1979, 2017, pp. 100–113.

[73] N.R. Tague, The Quality Toolbox, second ed., Rittenhouse, 2005.

[74] Object constraint language (OCL) v2.2, 2010.

[75] A. Kleppe, The Object Constraint Language, Addison-Wesley, 2003.

[76] R. Baena, R. Aragón, M. Enciso, C. Rossi, P. Cordero, Ángel Mora, Quality improvement in data models with SLFD-based OCL constraints, in: Proceedings of the 8th International Joint Conference on Software Technologies (ICSOFT), Reykjavík (Iceland), 2013, pp. 563–569, http://dx.doi.org/10.5220/0004593405630569.

[77] M. Eysholdt, H. Behrens, Xtext: Implement your language faster than the quick and dirty way, in: Companion to the 25th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (SPLASH/OOPSLA), 2010, pp. 307–309, http://dx.doi.org/10.1145/1869542.1869625.

[78] L. Bettini, Implementing Domain Specific Languages with Xtext and Xtend, Packt Publishing, 2016.

[79] T. Kosar, S. Gaberc, J.C. Carver, M. Mernik, Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments, Empir. Softw. Eng. 23 (5) (2018) 2734–2763, http://dx.doi.org/10.1007/s10664-017-9593-2.

[80] A. Barisic, V. Amaral, M. Goulão, Usability driven DSL development with USE-ME, Comput. Lang. Syst. Struct. 51 (2018) 118–157, http://dx.doi.org/10.1016/j.cl.2017.06.005.

[81] T. Kosar, S. Bohra, M. Mernik, Domain-specific languages: A systematic mapping study, Inf. Softw. Technol. 71 (2016) 77–91, http://dx.doi.org/10.1016/j.infsof.2015.11.001.

[82] A. Wortmann, O. Barais, B. Combemale, M. Wimmer, Modeling languages in Industry 4.0: an extended systematic mapping study, Softw. Syst. Model. 19 (1) (2020) 67–94, http://dx.doi.org/10.1007/s10270-019-00757-6.

[83] Object Management Group, Unified modeling language (OMG UML), 2017, URL https://www.omg.org/spec/UML/2.5.1/PDF.

[84] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002.

[85] C. Bauer, G. King, G. Gregory, Java Persistence with Hibernate, Manning, 2015.

[86] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, in: M. Akşit, S. Matsuoka (Eds.), Proceedings of the 11th European Conference on Object-Oriented Programming, ECOOP, Vol. 1241, 1997, pp. 220–242, http://doi.acm.org/10.1145/503209.503260.

[87] Z. Xu, Y. Dang, Automated digital cause-and-effect diagrams to assist causal analysis in problem-solving: a data-driven approach, Int. J. Prod. Res. 58 (17) (2020) 5359–5379.

[88] M. Shigemitsu, Y. Shinkawa, Extracting class structure based on fishbone diagrams, in: Proc. of the 10th Int. Conference on Enterprise Information Systems, ICEIS, Vol. 2, 2008, pp. 460–465.

[89] A. Yurin, A. Berman, N. Dorodnykh, O. Nikolaychuk, N. Pavlov, Fishbone diagrams for the development of knowledge bases, in: Proc. of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2018, pp. 967–972.

[90] A. Gwiazda, Quality tools in a process of technical project management, J. Achiev. Mater. Manuf. Eng. 18 (1–2) (2006) 439–442.

[91] Z. Yun, L. Weihua, C. Yang, The study of multidimensional-data flow of fishbone applied for data mining, in: Proc. of the 7th Int. Conference on Software Engineering Research, Management and Applications, SERA, 2009, pp. 86–91.

[92] I. Al-Azzoni, N. Petrovic, A. Alqahtani, A utility to transform CSV data into EMF, in: Proceedings of the 8th International Conference on Software Defined Systems (SDS), Gandía (Spain), 2021, pp. 1–6, http://dx.doi.org/10.1109/SDS54264.2021.9732143.

[93] J. Giner-Miguelez, A. Gómez, J. Cabot, A domain-specific language for describing machine learning datasets, J. Comput. Lang. 76 (2023) 101209, http://dx.doi.org/10.1016/j.cola.2023.101209.

[94] J. Giner-Miguelez, A. Gómez, J. Cabot, DescribeML: A dataset description tool for machine learning, Sci. Comput. Program. 231 (2024) 103030, http://dx.doi.org/10.1016/j.scico.2023.103030.

[95] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, second ed., Addison-Wesley, 2005.

[96] K. Thramboulidis, F. Christoulakis, UML4IoT—A UML-based approach to exploit IoT in cyber–physical manufacturing systems, Comput. Ind. 82 (2016) 259–272, http://dx.doi.org/10.1016/j.compind.2016.05.010.

[97] O.M.G. (OMG), OMG systems modeling language (OMG SysML) (formal/19-11-01), 2019, URL https://www.omg.org/spec/SysML/1.6/.

[98] J. Holt, S. Perry, SysML for Systems Engineering: A Model-Based Approach, third ed., The Institution of Engineering and Technology (IET), 2018.

[99] S. Wolny, A. Mazak, C. Carpella, V. Geist, M. Wimmer, Thirteen years of SysML: a systematic mapping study, Softw. Syst. Model. 19 (1) (2020) 111–169, http://dx.doi.org/10.1007/s10270-019-00735-y.

[100] O. Berndt, U. Freiherr Von Lukas, A. Kuijper, Functional modelling and simulation of overall system ship – virtual methods for engineering and commissioning in shipbuilding, in: Proc. of the 29th Conference on Modeling and Simulation, ECMS, Albena (Varna, Bulgaria), 2015, pp. 347–353, http://dx.doi.org/10.7148/2015-0347.

[101] W.O.W. Group, OWL 2 web ontology language document overview, 2012, URL https://www.w3.org/TR/owl2-overview/.

[102] M. Uschold, Demystifying OWL for the Enterprise, No. 17 in Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, 2018.

[103] I. Horrocks, P. f. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and ruleml, 2004, URL https://www.w3.org/submissions/SWRL/.

[104] R. Drath (Ed.), AutomationML: A Practical Guide, De Gruyter Oldenbourg, 2021.

[105] N. Schmidt, A. Lüder, Automationml in a nutshell, 2015, URL https://www.automationml.org/wp-content/uploads/2021/06/AutomationML-in-a-Nutshell_151104.pdf.

[106] M. Schneider, T. Mittag, J. Gausemeier, Modeling language for value networks, in: Proceedings of the 25th Conference of the International Association for Management of Technology, IAMOT, Orlando (Florida, USA), 2016, pp. 94–110.

[107] M. Lütjen, D. Rippel, GRAMOSA framework for graphical modelling and simulation-based analysis of complex production processes, Int. J. Adv. Manuf. Technol. 81 (1–4) (2015) 171–181, http://dx.doi.org/10.1007/s00170-015-7037-y.

[108] D. Chen, D.V. Panfilenko, M.R. Khabbazi, D. Sonntag, A model-based approach to qualified process automation for anomaly detection and treatment, in: Proceedings of the 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, Germany, 2016, pp. 1–8, http://dx.doi.org/10.1109/ETFA.2016.7733731.

[109] Industrial automation systems and integration. Product data representation and exchange. Part 11: Description methods: The EXPRESS language reference manual, 2004, URL https://www.iso.org/standard/38047.html.

[110] D.A. Schenck, P.R. Wilson, Information Modeling: The EXPRESS Way, Oxford University Press, 1994.

[111] J. Mosser, R. Pellerin, M. Bourgault, C. Danjou, N. Perrier, GRMI4.0: a guide for representing and modeling Industry 4.0 business processes, Bus. Process Manage. J. 28 (4) (2022) 1047–1070, http://dx.doi.org/10.1108/BPMJ-12-2021-0758.

[112] I. Compagnucci, F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A systematic literature review on IoT-aware business process modeling views, requirements and notations, Softw. Syst. Model. 22 (3) (2023) 969–1004, http://dx.doi.org/10.1007/s10270-022-01049-2.

[113] B. Roelens, G. Poels, Towards a strategy-oriented value modeling language: Identifying strategic elements of the VDML meta-model, in: W. Ng, V.C. Storey, J.C. Trujillo (Eds.), Proceedings of the 32nd International Conference on Conceptual Modeling, in: Lecture Notes in Computer Science (LNCS), Vol. 8217, Hong-Kong (China), 2013, pp. 454–462, http://dx.doi.org/10.1007/978-3-642-41924-9_38.

[114] M. Schleipen, R. Drath, Three-view-concept for modeling process or manufacturing plants with automationml, in: Proceedings of the Conference on Emerging Technologies & Factory Automation, EFTA, Palma de Mallorca (Spain), 2009, pp. 1–4, http://dx.doi.org/10.1109/ETFA.2009.5347260.

[115] G.P. Gujarathi, Y. Ma, Parametric CAD/CAE integration using a common data model, J. Manuf. Syst. 30 (3) (2011) 118–132, http://dx.doi.org/10.1016/j.jmsy.2011.01.002.

[116] A. Perzylo, N. Somani, M. Rickert, A. Knoll, An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions, in: Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Hamburg (Germany), 2015, pp. 4197–4203, http://dx.doi.org/10.1109/IROS.2015.7353971.

[117] A. Olivé, Conceptual Modeling of Information Systems, Springer, 2007.

[118] O. Givehchi, K. Landsdorf, P. Simoens, A.W. Colombo, Interoperability for industrial cyber–physical systems: An approach for legacy systems, IEEE Trans. Ind. Inform. 13 (6) (2017) 3370–3378, http://dx.doi.org/10.1109/TII.2017.2740434.

[119] B.L. Sadigh, H.O. Unver, S. Nikghadam, E. Dogdu, A.M. Ozbayoglu, S.E. Kilic, An ontology-based multi-agent virtual enterprise system (OMAVE): part 1: domain modeling and rule management, Int. J. Comput. Integr. Manuf. 30 (2–3) (2017) 320–343, http://dx.doi.org/10.1080/0951192X.2016.1145811.

[120] A. Patel, N.C. Debnath (Eds.), Semantic Technologies for Intelligent Industry 4.0 Applications, River Publishers, 2023.

[121] F. Arnold, G. Podehl, Best of both worlds – a mapping from EXPRESS-g to UML, in: J. Bézivin, P.-A. Muller (Eds.), Proceedings of the 1st International Workshop on the Unified Modeling Language (UML), in: Lecture Notes in Computer Science (LNCS), Vol. 1618, Mulhouse (France), 1998, pp. 49–63, http://dx.doi.org/10.1007/978-3-540-48480-6_5.

[122] J. Lubell, R.S. Peak, V. Srinivasan, S.C. Waterbury, STEP, XML, and UML: Complementary technologies, in: Proceedings of the 24th International Conference on Computers and Information in Engineering, Salt Lake City (Utah, USA), 2004, pp. 915–923, http://dx.doi.org/10.1115/DETC2004-57743.

[123] P. Pauwels, W. Terkaj, EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology, Autom. Constr. 63 (2016) 100–133, http://dx.doi.org/10.1016/j.autcon.2015.12.003.

[124] J. Beetz, J. Van Leeuwen, B. De Vries, IfcOWL: A case of transforming EXPRESS schemas into ontologies, Artif. Intell. Eng. Des. Anal. Manuf. 23 (1) (2009) 89–101, http://dx.doi.org/10.1017/S0890060409000122.

[125] F.G.C. Ribeiro, A. Reuberg, C.E. Pereira, M.S. Soares, An approach for architectural design of automotive systems using MARTE and sysml, in: Proc. of the 14th International Conference on Automation Science and Engineering (CASE), Munich (Germany), 2018, pp. 1574–1580, http://dx.doi.org/10.1109/COASE.2018.8560415.

[126] S. Zhu, J. Tang, J.-M. Gauthier, R. Faudou, A formal approach using SysML for capturing functional requirements in avionics domain, Chin. J. Aeronaut. 32 (12) (2019) 2717–2726, http://dx.doi.org/10.1016/j.cja.2019.03.037.

[127] M. Morelli, Automated generation of robotics applications from simulink and SysML models, in: Proceedings of the 30th Annual Symposium on Applied Computing (SAC), Salamanca (Spain), 2015, pp. 1948–1954, http://dx.doi.org/10.1145/2695664.2695882.

[128] M. Boullé, C. Charnay, N. Lachiche, A scalable robust and automatic propositionalization approach for bayesian classification of large mixed numerical and categorical data, Mach. Learn. 108 (2) (2019) 229–266, http://dx.doi.org/10.1007/s10994-018-5746-9.

[129] J.M. Kanter, K. Veeramachaneni, Deep feature synthesis: Towards automating data science endeavors, in: Proceedings of the 2nd International Conference on Data Science and Advanced Analytics, DSAA, Paris (France), 2015, pp. 1–10, http://dx.doi.org/10.1109/DSAA.2015.7344858.

[130] M. Samorani, Automatically generate a flat mining table with dataconda, in: Proceedings of the International Conference on Data Mining Workshop, ICDMW, Atlantic City (New Jersey, USA), 2015, pp. 1644–1647, http://dx.doi.org/10.1109/ICDMW.2015.100.

[131] C. Nica, A. Braud, F.L. Ber, Exploring heterogeneous sequential data on river networks with relational concept analysis, in: Proceedings of the 23rd International Conference on Conceptual Structures ICCS, in: Lecture Notes in Computer Science, Vol. 10872, Edinburgh (Scotland, United Kingdom), 2018, pp. 152–166, http://dx.doi.org/10.1007/978-3-319-91379-7_12.

[132] C. Abreu Ferreira, J. Gama, V. Santos Costa, Contrasting logical sequences in multi-relational learning, Prog. Artif. Intell. 8 (4) (2019) 487–503, http://dx.doi.org/10.1007/s13748-019-00188-w.

[133] E. Cilia, N. Landwehr, A. Passerini, Relational feature mining with hierarchical multitask kfoil, Fund. Inform. 113 (2) (2011) 151–177, http://dx.doi.org/10.3233/FI-2011-604.

[134] G. Manjunath, M.N. Murty, D. Sitaram, Combining heterogeneous classifiers for relational databases, Pattern Recognit. 46 (1) (2013) 317–324, http://dx.doi.org/10.1016/j.patcog.2012.06.015.

[135] P. Vassiliadis, A. Simitsis, Extraction, transformation, and loading, in: L. Liu, M.T. Özsu (Eds.), Encyclopedia of Database Systems, Springer, 2009, pp. 1095–1101, http://dx.doi.org/10.1007/978-0-387-39940-9_158.

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

[137] J. Trujillo, S. Luján-Mora, A UML based approach for modeling ETL processes in data warehouses, in: I.-Y. Song, S. W. Liddle, T.-W. Ling, P. Scheuermann (Eds.), Proc. of 22nd International Conference on Conceptual Modeling, ER, in: Lecture Notes in Computer Science (LNCS), Vol. 2813, Chicago (Illinois, USA), 2003, pp. 307–320, http://dx.doi.org/10.1007/978-3-540-39648-2_25.

[138] L. Munoz, J.-N. Mazon, J. Trujillo, Etl process modeling conceptual for data warehouses: A systematic mapping study, IEEE Lat. Am. Trans. 9 (3) (2011) 358–363, http://dx.doi.org/10.1109/TLA.2011.5893784.

[139] Z. El Akkaoui, E. Zimànyi, J.-N. Mazón, J. Trujillo, A model-driven framework for ETL process development, in: Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, Glasgow (Scotland, United Kingdom), 2011, pp. 45–52, http://dx.doi.org/10.1145/2064676.2064685.

[140] J. Wang, B. Liu, Design of etl tool for structured data based on data warehouse, in: Proceedings of the 4th International Conference on Computer Science and Application Engineering, CSAE, Sanya, China, 2020, pp. 1–5, http://dx.doi.org/10.1145/3424978.3425101.

[141] M. Przysucha, J. Hüsers, D. Liberman, O. Kersten, A. Schlüter, S. Fraas, D. Busch, M. Moelleken, C. Erfurt-Berge, J. Dissemond, U. Hübner, Design and implementation of an etl-process to transfer wound-related data into a standardized common data model, Stud. Health Technol. Inform. 307 (2023) 258–266, http://dx.doi.org/10.3233/SHTI230723.

[142] A. Suleykin, P. Panfilov, Metadata-driven industrial-grade etl system, in: 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 2433–2442, http://dx.doi.org/10.1109/BigData50022.2020.9378367.

[143] S. Hira, P.S. Deshpande, Automated heuristic based context dependent etl process to generate multi-dimensional model for tabular data, Concurr. Comput.: Pract. Exper. 35 (2) (2023) e7459, http://dx.doi.org/10.1002/cpe.7459.

[144] D. Kolovos, A. de la Vega, J. Cooper, Efficient generation of graphical model views via lazy model-to-text transformation, in: Proceedings of the 23rd International Conference on Model Driven Engineering Languages and Systems (MoDELS), 2020, pp. 12–23, http://dx.doi.org/10.1145/3365438.3410943.