Designing a Business View of Enterprise Data: An approach based on a Decentralised Enterprise Knowledge Graph

Max Chevalier, Joan Marty, Franck Ravat, Bastien Vidé

Designing a Business View of Enterprise Data: An approach based on a Decentralised Enterprise Knowledge Graph

Max Chevalier, Joan Marty, Franck Ravat, Bastien Vidé

Designing a Business View of Enterprise Data

An approach based on a Decentralised Enterprise Knowledge Graph

Max Chevalier
Institut de la Recherche en Informatique de Toulouse
Toulouse, France
max.chevalier@irit.fr

Joan Marty
umlaut.
Blagnac, France
joan.marty@umlaut.com

Franck Ravat
Institut de la Recherche en Informatique de Toulouse
Toulouse, France
franck.ravat@irit.fr

Bastien Vidé
Institut de la Recherche en Informatique de Toulouse
Toulouse, France
umlaut
Blagnac, France
bastien.vide@irit.fr
bastien.vide@umlaut.com

ABSTRACT

Nowadays, companies manage a large volume of data usually organised in "silos". Each "data silo" contains data related to a specific Business Unit, or a project. This scattering of data does not facilitate decision-making requiring the use and cross-checking of data coming from different silos. So, a challenge remains: the construction of a Business View of all data in a company. In this paper, we introduce the concepts of Enterprise Knowledge Graph (EKG) and Decentralised EKG (DEKG). Our DEKG aims at generating a Business View corresponding to a synthetic view of data sources. We first define and model a DEKG with an original process to generate a Business View before presenting the possible implementation of a DEKG.

CCS CONCEPTS

• **Information systems → Enterprise applications**; *Information integration.*

KEYWORDS

Business View, Enterprise Knowledge Graph, Schema Matching

ACM Reference Format:
Max Chevalier, Joan Marty, Franck Ravat, and Bastien Vidé. 2021. Designing a Business View of Enterprise Data: An approach based on a Decentralised Enterprise Knowledge Graph. In *Proceedings of 25th International Database Engineering & Applications Symposium (IDEAS 2021).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3472163.3472276

1 INTRODUCTION

Today's organisations have large volumes of production data and documents scattered across multiple sources and heterogeneous environments. One of the most popular ways to store this production data is to organise it as "data silos". This type of storage allows companies to organise data according to different criteria specific to their activities (by project, by supplier, by customer, by Business Unit, etc.). Each silo allows local data management with an adapted storage system. However, the isolation of the data contained into silos is a major drawback: this can lead to redundancy or even strong inconsistencies between different silos. Also, the company management staff does not have a Unified View of the data due to the isolation of those silos. In addition, a lot of documents generated in the companies are also not intensively used for decision-making.

Different Unified Views construction strategies currently exist, such as Data Warehouses, or Data Lakes. Data Warehouses (DW) are Business Intelligence (BI) databases used to centralise useful data for the decision-makers of an organisation [21]. Data Warehouses contain only a part of production data, which is predetermined and modelled to match specific needs. The integration of this data is named Extract, Transform, Load (ETL). On the other hand, Data Lakes (DL) ingest raw data from multiple sources (structured or unstructured data) and store them in their native format [26]. Also, an advantage of DL is that the data is prepared only when they're used by a user. As opposed to the DW, the Data Lake ingest as much data as possible in the organisation. It also does not require a modelled schema, and can be then used to a greater number of users that are not decision-makers.

The Enterprise Knowledge Graph (EKG) is one of the newest approach that also can be used as a solution to the "data silos problem" in a company [18]. It is defined as a structure used to "represent relationships between the datasets" [8] but also as a "semantic network of concepts, properties, individuals and links representing and referencing foundational and domain knowledge relevant for an enterprise" [11]. An EKG particularly highlights the relationships existing, for instance, between the currently existing information in the company.
Unfortunately, organisations still have some difficulties to create their own EKG and to integrate in a single schema the large amount

of heterogeneous data, information and documents available in the numerous sources they own. Moreover, the obtained model do not really abstract the data to concepts, making its exploitation by decision-makers difficult.

In this paper, we propose an original process to generate a Business View of multiple data sources within an Enterprise Knowledge Graph. Thus, we first detail the different concepts related to our proposal and then describe the generation of such a view through different steps within a Decentralised Enterprise Knowledge Graph.

2 RELATED WORK

Initially, the concept of Knowledge Graph (KG) is defined in the Web Semantic domain. The goal of a KG in Web Semantic is to build an "Entity-centric view" of the data from multiple sources [13]. A KG links multiple resources from different websites together. A resource can be anything, from a webpage to an open data API endpoint that represents an entity. Back in 2012, Google was able to build a Knowledge Graph based on different sources, like Freebase or Wikidata [29].

Outside the Web Semantic domain, a Knowledge Graph may be defined as a graph of entities and relationships [24] or "a network of all kind of things which are relevant to a specific domain or to an organization [...]" [7]. Ehrlinger et al. definition is equivalent to Blumauer's one, except that the possible usage of inference in a Knowledge Graph is added: "A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge" [10]. Referring to these previous definitions, a Knowledge Graph may have technical characteristics but it is specific to a particular domain of research.

An Enterprise Knowledge Graph, a KG applied to an organisation, is a private Knowledge Graph containing private and domain-specific knowledge. The EKG can be used as a Unified View that is able to solve the "data silos problem" [18]. The goal of such an EKG is to help the users to represent, manage, exploit the knowledge of an organisation. It also allows machine-to-machine inter-operability using the stored knowledge. The Data Source of the EKG is usually centralized [32].

Some papers explain the implementation of Enterprise Knowledge Graphs [11, 30, 33]. Additionally, some tools allow to integrate database contents [1, 31] or documents [9, 14] into a Knowledge Graph or even directly from texts through Named Entity Resolution [12] and Thematic Scope Resolution [4, 12]. A few less researches have been done around the querying side of Enterprise Knowledge Graphs [30]. Based on the literature, the Enterprise Knowledge Graph (EKG) seems to be a great support for a Unified View, as it supports a lot of flexibility. Moreover, relationships may help to understand how the data is related for decision-making processes, and also help how to infer new knowledge from the existing one. However, neither a clear definition of what an Enterprise Knowledge Graph nor a global architecture are defined in the literature, as far as we know. We also identify that no process to build an EKG schema from the available data exists.

Intending to propose a new way of building Enterprise Knowledge Graphs, we also studied the Schema Matching. The Schema Matching is a a set of methods allowing to "determine which concepts of one schema match those of another" [23] in distributed data sources. Schema Matching has been well studied in the literature, both Database Matching [16, 25] aiming at defining methodologies to match different relational databases, and Ontology Matching [3] which aims more at matching web semantic ontologies. New approaches based on similarity have been developed to allow schema matching on graph data structures [22], and improve current Schema Matching techniques [20, 27].

In the next section, we detail our own definition of the Enterprise Knowledge Graph as a Business View of company data. We also define the concept of Decentralised EKG (DEKG). We propose a process to build a DEKG Business View schema from existing data using schema matching, before discussing its architecture and its implementation.

3 ENTERPRISE KNOWLEDGE GRAPH

3.1 From EKG to DEKG

In our context, an Enterprise Knowledge Graph (EKG) represents all the source data of interest for the company in order to offer end users a Unified View of this data. Moreover, an EKG should emphasise the relationships that exist between them. Such a Unified View allows end-users to identify, locate and access and finally analyse these data. An EKG also helps them in decision-making tasks thanks to the knowledge they can extract from the Unified View. Such Unified View mostly corresponds to the result of the integration of the different source's schemata. An EKG should be easily **extensible** in terms of data (i.e. should ingest new data sources) and user needs (i.e. queries, exploration capabilities..), without the need of human intervention.

In this paper we define an extension of such a concept that is named a *Decentralised Enterprise Knowledge Graph* (DEKG). The decentralised dimension of a DEKG aims at offering a better scalability of the underlying system since data are not integrated in a centralised system (i.e. only data source schemata are centralised). We based our approach around a global-as-view approach [19].

Furthermore, in contrast with the above definition, a DEKG proposes to non-technical end users a Business View that is a **synthetic view** of source data through a unified schema. It is a schema that corresponds to a **"end-user view"** of the global schema content. It "erases" the specific implementation particularities present in the different data sources. Such a view allows the end-user to understand what information is available, how it's linked and in which database/repository it is stored.

To obtain such a Business View, a DEKG implementation is based on a 3-steps process (see Figure 1):

- the first step aims at generating data sources schemata from the different data sources, one independently to others;
- the second step aims at generating a global schema that is the result of the schema matching of the different data source schemata;
- the third step is more original since it aims at synthesising the global schema into a synthetic and understandable schema that will be proposed as the Business View to the end-user.

Please note that the DEKG always keeps all the data sources while generating all the different schemata in each step.

Since a DEKG should emphasise the relationships between data and data schema, we propose in this paper to model a DEKG as a set of

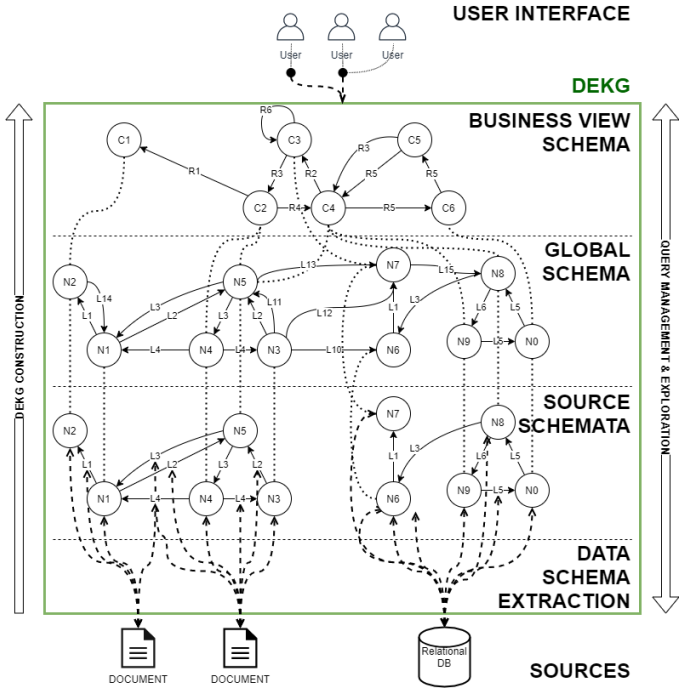

Figure 1: Decentralised Enterprise Knowledge Graph workflow

graphs. These relationships present in the different DEKG schemata, will help decision-making by showing the "context" of the available data. These relationships also support graph exploration or new knowledge discovery.

3.2 Our DEKG construction process

Our process generates different schemata. According to the Figure 1, we define the concepts of source schema, global schema and Business View and explain how to generate them. Moreover, in order to explain every schema generation, we propose an illustrative example.

Example. Three data sources are available (one relational database containing 2 tables and 2 CSV files). The content of each data source is presented in the table 1.

3.2.1 General schema graph model. All the schemata generated by the DEKG is modelled as an heterogeneous graph, based on the Property Graphs [5, 17]. We define a graph g as follows: g = (V,E); $V = \{\vartheta_1, ..., \vartheta_v\}$ is the set of nodes and $E = \{e_1, ..., e_e\}$ the set of edges, with $E \subseteq V \times V$. Each node has a label τ that belongs to $T = \{\tau_1, ..., \tau_t\}$. The function $w : \vartheta \rightarrow \tau$ is used to return the label τ of a node. Every edge also has a label μ belonging to $M = \{\mu_1, ..., \mu_u\}$ and the function $n : e \rightarrow \mu$ returns the label μ of an edge. Node and edge labels can be characterised by a set of attributes belonging to $X = \{\chi_1, ..., \chi_x\}$. Those attributes, in the context of an Enterprise Knowledge Graphs, are also called *properties*. To simplify, we ignore in this definition all companion functions (modifying an attribute, get the list of attribute of an edge or a node...).

Thanks to these definitions, we define every DEKG schemata in the next sections. Due to space limitation, we limit our discussion to structured data sources only.

3.2.2 Source Schema - Step #1. The source schema is composed of all the schemata of all sources handled by the DEKG. This schema results from the extraction of the structure (e.g. list of attributes) of every source independently from others. In order to facilitate schema matching done at step #3 we decided to not store attributes into nodes and choose an "Exploded" Graph. Thus, in the source schema, any node ϑ in V corresponds to either an *entity* (e.g. the name of a table in a relational database), either a *relationship* between two entities (e.g. a foreign key in a relational database), either an *attribute* (property) characterising an entity. The set of node types of V is defined as $T = \{REL, ENTITY, PROP\}$ where *"REL"* corresponds to a relationship, *"ENTITY"* to an entity and *"PROP"* to a property.

In order to properly connect the different nodes, we define the different types of any edge e in E as : $M = \{hasProp, hasRel\}$ where *"hasProp"* connects a node of type ENTITY or REL to a node of type PROP and *"hasRel"* connects a node of type ENTITY to another node of type ENTITY.

Moreover, to keep the location of data within the data source, every node or edge are characterised by a minimal set of attributes $X = \{NAME, ID, URIS\}$ where *NAME* corresponds the the name of the entity/property/relationship, *ID* corresponds the identifier of the node/edge and *URIS* corresponds to a set of URIs. Every URI corresponds to the data source URI where the data is located (entity/relationship).

Table 1: Data Source structures and content

DATABASE

Students Table			
id	Last Name	First Name	Class (FK)
1	Doe	John	A
2	Doe	Jane	B

Class Table	
id	Teacher (FK)
A	Alice Machin
B	Bob Lambda
C	John Doe

Teachers Table	
id	Full name
1	Alice Machin
2	Bob Lambda
3	John Doe

People.csv	
LastName	FirstName
Machin	Alice
Lambda	Bob
Doe	John
Doe	Jane

Grades.csv		
FULL NAME	GRADE	COURSE
John Doe	15	M3xxx
Jane Doe	16	M1xxx

In the following sections and figures, we will represent Exploded Graphs, such as Figure 2, as Properties Graph where our T types ($< entity >$, $< rel >$ and $< prop >$) are displayed as labels, and the node attributes are stored as properties. for clarity sake, all the properties, except the name, are not depicted on the figures.

schema-2.png

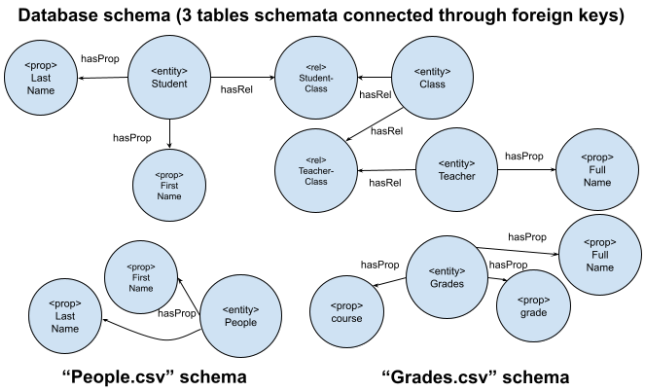

Figure 2: Data source schemata extraction result

So, as a result the Source schema contains all the data source schemata that are not yet connected. The objective of the next step is to construct the global schema. Figure 2 shows the Source schema extracted from our example data sources (Table 1).

3.2.3 Global Schema - Step #2. The global schema corresponds to the result of schema matching of the all data source schemata available in the Source schema.

Inspired from previous work in schema matching, in this paper, we define five additional edge types, meaning that the edge types set M in the global schema are defined as:

$$M = \{hasProp, hasRel, identical, similar, extends, includes, aggregation\}$$

To go deeper in every type, we defined them as following:

- **identical** L_i: can be applied to an edge between two entities, that highlights that they are textually identical and should be treated as exactly the same entity;
- **similar**: can be applied to an edge between two entities, two relationships or two properties, that highlights that they are related and eventually could be treated as the same entity, relationship or property;
- **extends**: can be applied to an edge between two entities, or two relations, that shows that one entity/relationship is a "superclass" of another entity/relationship. The superclass ontologically represents a more broad entity/relationship;
- **includes**: can be applied to an edge between two properties that highlights that **values** of a property class are included in another property class at a certain rate p;
- **aggregation**: can be applied to an edge between three or more properties that highlights that a property is a combination of two or more other properties. Somehow, those properties could be treated as similar in a low-granularity view of the schema. Introducing aggregation inside a graph schema makes it an n-uniform hypergraph, as we are linking more than two nodes together.

To infer new edges of such types in the global schema, we also define eight rules that exploit graph structure and data from data sources to create new relationships. These rules and global schema mapping algorithm are detailed in section 3.3.

3.2.4 Business View schema - Step #3. The Business View schema is a schema that abstracts the schema matching process that was run in the previous steps. At the step, the schema returns to a Property Graph as defined in Section 3.2.1.

The way this Business View Schema is generated is detailed in section 3.4.

3.3 Constructing the DEKG Global Schema

In our process, the main step is the definition of the global schema since it is the result of schema matching of all the source schemata.

In order to connect nodes of the different graphs of source data with new edges, we define ten rules that aim at creating edges which of M type.

Such rules will allow to consider multiple entities, properties, or

relationships and related data. This section details these rules and the way they are applied through a specific algorithm.

3.3.1 Matching schemata rules. To facilitate decision-making on a global view, it's necessary to define new relationships between components of graphs representing source schemata. In our approach, we propose a schema matching based on rules specifying automatically new relationships and/or nodes. We based them on both Database Matching [25] and Ontology Matching [3] ideas to propose both a relational and a semantic approach. Those rules cover multiple approaches of schema matching defined by Rahm et al: Both schema-based and instance-based, and using linguistic and constraints.

These rules are designed to be generic and automatically ran, but they also can be completed by multiple sets of domain-specific rules depending on the information the company wants to include into the Business View.

- **R1**: We create an **"identical"** type edge between two entity nodes if they have a strictly equal value for attribute NAME;
- **R2**: We create a **"similar"** type edge between two entity nodes if their NAME are semantically close [15];
- **R3**: We create create a **"similar"** type edge between two relationships A and B if A is linked by "*hasRel*" to an entity C, B linked by "*hasRel*" to another distinct entity D, where C and D are linked by a "similar" edge. A and B must also be linked to a third common entity E;
- **R4**: We create an **"aggregation"** type edge between a set of properties from one entity A and one property of another entity B if concatenated values of the properties of A equal the values of the property of B;
- **R5**: We create an **"includes"**type edge from a property A to a second one B if the values of the property A partly equal property B. We store the rate of equality in the "include" edge;
- **R6**: We create an **"hasProp"** type edge between an entity E linked to a property A by "hasProp", and a property B of another entity if the property A entirely includes B;
- **R7**: We create an **"extends"** type edge from one entity (nodes of type ENTITY) A to another B if *all* the properties (nodes of type PROP) of A includes entirely or aggregates B property classes;
- **R8**: We create a new **REL** node A, an "*hasRel*" edge between A and an entity B, another "*hasRel*" edge between A and a different entity C, if all properties of B are linked by an "*extends*" or "*aggregates*" edge to the properties of C, and B and C are not linked by any direct edges (*extends*, *similar*, *identical*, ...).

Some of those rules can even create nodes in our Global Schema, meaning that it is possible to create new Entity, Relationship and Property (i.e. a node with one of these type). $R8$, for instance, is creating a new relationship (node of type REL) between two entities (two nodes of type ENTITY).

The rules are ordered to be executed from $R1$ to $R8$ and must be re-executed each time one of the original data source schema changes. The rules have to be executed once the source schema is constructed to start schema matching. An algorithm to construct the global schema is proposed in Algorithms 1, 2 and 3.

Example. After applying the proposed algorithms on Source schema (see Figure 2) we obtain the Global Schema shown on the Figure 3. With only four sources and a few properties per source,

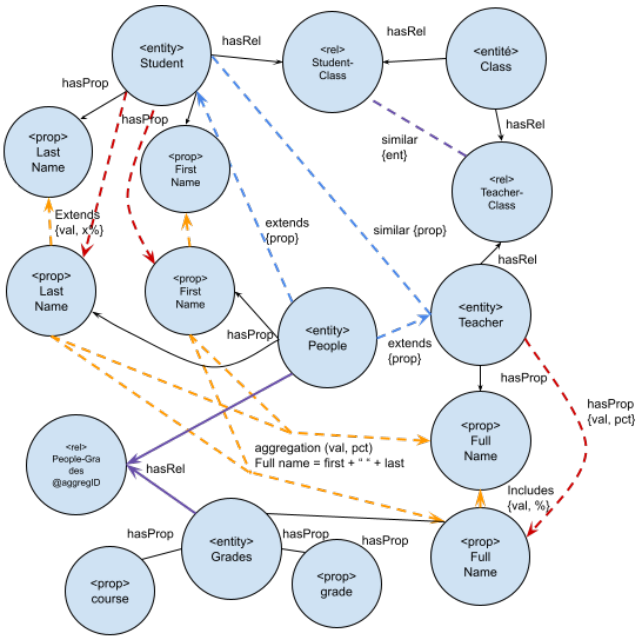

Figure 3: Full matched Global Schema example

Algorithm 1: Global Schema Construction: linguistic schema rules

```
Input: S = s_1, s_2, ...s_n  #Set of all the data source schemata (= Source Schema)
GS = ∅
for each data source schema s of S do
    copy the data source schema s into the global schema (GS);
for each couple of "ENTITY" nodes (e1, e2) in GS do
    #R1
    if e1.name = e2.name then
        Create "identical" between e1 and e2 (name) in GS
    #R2
    if wordSimilarity(e1.name, e2.name) > simThreshold then
        Create "similar" if not exists between e1 and e2 ('name', 'wordsim', word-similarity(e1.name, e2.name)) in GS
    if thesaurusSynonym(e1.name, e2.name) > synThreshold then
        Create "similar" between e1 and e2 ('name', 'thesaurus', thesaurus.synonym(e1.name, e2.name)) in GS
return GS;
```

we can observe that a lot of new edges (i.e. dashed edges) have been created between nodes from different data source schemata. For instance "red coloured" edges are new hasProp type edges, "purple coloured" edges are new hasRel type edges, whereas as "blue coloured" edges are similar or extends type edges. All those new links will be exploited when constructing the Business View Schema (see section 3.4). The sources URIs of every source node, whatever type they are (*entity*, *prop* or *rel*), are stored as an attribute in the

Algorithm 2: Global Schema Construction: instance based rules

```
1  Input: S = s1, s2, ...sn  #Set of all the data source schemata (= Source Schema)
2  GS = CurrentGlobalSchema #Result of the previous Algorithm
3  for each couple of "REL" (r1, r2) nodes in GS do
4      #R3
5      # if r1 and r2 have a common entity through REL nodes
6      if hasRels(r1).some(hasRels(r2)) then
7          if links(nodes(r1)).filter(links(nodes(r2))).containsType(['similar', 'identical']) then
8              Create "similar" between r1 and r2 in GS
9  for each set of 2 properties or more (p1, p2, ..., pn) of the same entity and a property p0 of another entity in GS do
10     #R4
11     sameNum := 0
12     for each line of values(p1) as v1, values(p2) as v2, ..., values(pn) as vn do
13         if value(p0) != v1.concat(v2, v3, ..., vn) then
14             sameNum++
15     if sameNum > 0 then
16         Create "aggregation" between p1, p2, ..., and pn with argument (sameNum/length(values(p1, p2,...), sameNum/length(values(p0))) in GS
17 for each couple of propery nodes (p1, p2) in GS do
18     #R5
19     includePct = values(p1).some(values(p2))
20     if includePct > 0 then
21         Create "includes" from p1 to p2, with (includePct/length(values(p1), includePct/length(p2)) in GS
22     #R6
23     if links(p1).filter(links(p2).filter(type = 'extends').size > 0 then
24         A := nodes(links(p1).filter(type = 'hasProp')).filter(type = 'entity')[0] B := nodes(links(p2).filter(type = 'hasProp')).filter(type = 'entity')[0] if A != B then
25             Create new "hasProp" between A and p2 in GS
26 return GS;
```

Algorithm 3: Global Schema Construction: instance and schema based rules

```
1  Input: S = s1, s2, ...sn  #Set of all the data source schemata (= Source Schema)
2  GS = CurrentGlobalSchema #Result of the previous Algorithm
3  for each couple of "ENTITY" nodes (e1, e2) in GS do
4      #R7
5      if links(nodes(links(e1).filter(type = 'hasProp'))).filter(type in 'includes', 'aggregate').includes(links(nodes(links(e2).filter(type = 'hasProp'))).filter(type in 'includes', 'aggregate')) AND links(nodes(links(e2).filter(type = 'hasProp'))).filter(type in 'includes', 'aggregate').includes(links(nodes(links(e1).filter(type = 'hasProp'))).filter(type in 'includes', 'aggregate')) AND "all 2nd arg is 1" then
6          Create "extends" between e1 and e2 in GS
7      #R8
8      if links(nodes(links(e1).filter(type = 'hasProp'))).filter(type in 'extends', 'aggregate').includes(links(nodes(links(e2).filter(type = 'hasProp'))).filter(type in 'extends', 'aggregate')) AND nodes(links(e1)).some(e2) == 0 then
9          Create new REL node (A) in GS Create new hasRel between e1 and A in GS Create new hasRel between e2 and A in GS
10 return GS;
```

matched nodes of the Global Schema. Please note that in Figure 3, we also stored, as an attribute in every element we created during this step, the rule name it has been generated with. That allows the system to differentiate all graph nodes/edges even if they have the same type.

3.4 Constructing the Business View Schema

As explained in section 3.2.4, the Business View Schema is one of the most important aspect of our proposal. That view must be built from the Global Schema, and allow user to see all available data at a glance. The construction of the Business View schema relies on 2 phases.

3.4.1 First Phase. The first phase is quite usual, since it aims to gather all similar nodes/edges available in the global schema into a single node/edge. The result is called "Synthetic Global Schema". It is important to note that the NAME attribute of every gathered nodes is stored in a new attribute (a set of NAME) called *CONTAINS* in the final node. Such new attributes allow us to keep the path to the aggregated nodes within the global schema. Also, every source URIs from the sources nodes are still stored in a new attribute containing the set of URIs corresponding to the sources of those source nodes.

To do so, we exploit the new edges created during schema matching in the Global Schema ; meaning that we gather all nodes connected

by edges of type "identical", "similar", "extends", "aggregation" and "includes", as specified in the Algorithm 4.

Algorithm 4: Business View: node combining

```
Input: GS
for each link R in GS do
    if type(R.subject) = prop and type(R.object) = prop then
        if R.predicate = includes(x,y) and x = 1 then
            Combine R.object into R.subject
        if R.predicate = aggregate(x,y) and x = 1 then
            Combine R.object into R.subject
    if type(R.subject) = entity and type(R.object) = entity
     then
        if R.predicate = extends then
            Combine R.object into R.subject Add R.object
              into "contains" attribute in R.subject
        if R.predicate = similar OR R.predicate = identical
         then
            Create a node E in GS Set E.name =
              R.subject.name + R.object.name Combine
              R.subject into E Combine R.object into E
    if type(R.subject) = rel and type(R.object) = rel then
        if R.predicate = extends then
            Combine R.object into R.subject Add R.object
              into "contains" attribute in R.subject
        if R.predicate = identical then
            Create a node E in GE Set E.name =
              R.subject.name + R.object.name Combine
              R.subject into E Combine R.object into E
        if R.predicate = similar then then
            if links(R.subject).filter(type = "hasRel").subject =
              links(R.object).filter(type = "hasRel").subject
              AND links(R.subject).filter(type =
              "hasRel").object = links(R.object).filter(type =
              "hasRel").object then
                Combine R.object into R.subject Set
                  R.subject = links(R.subject).filter(type =
                  "hasRel").subject.name + '-' +
                  links(R.subject).filter(type =
                  "hasRel").object.name
```

3.4.2 Example. When applying such process on the Global Schema (Figure 3) we obtain the resulting Synthetic Global Schema as shown in Figure 4. We can see that the nodes with *name "Teacher"* and *"Student"* have been gathered in node with *name "People"* since *extends* type edges have been created between these nodes in the Global Schema. You can also see that a new attribute named *"contains"* have been added to node *People* with the values "Teacher" and "Student" to keep a link to the corresponding nodes in the Global Schema.

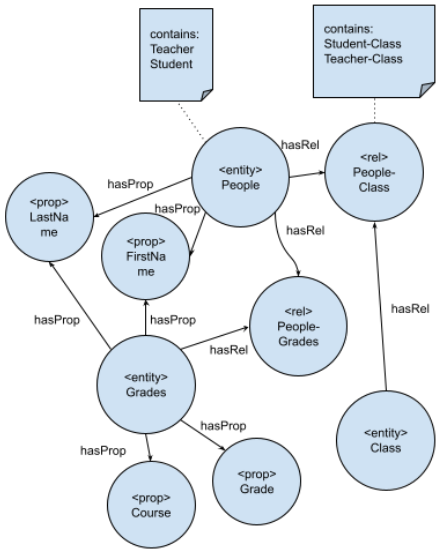

Figure 4: Synthetic global schema

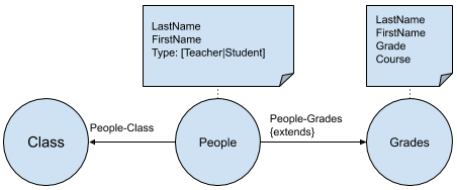

Figure 5: Generated Business View

3.4.3 Second Phase. While the first phase goal was to combine similar/identical nodes of the global schema, the second phase aims at transforming the "Synthetic Global Schema" into the final Business View in which *PROP* nodes are re-integrated in corresponding nodes and relationships as attributes. The Business View is a non-technical view, computed on the fly, that represents all the sources data in a single endpoint. It is aimed at end-users to help them understand and query the available organisations data.
The types of the nodes and relationships in the Business View correspond to the NAME attribute value in the Synthetic Global Schema. That phase makes the Business View look like an understandable and regular entity-relationship meta-graph as graph databases like *Neo4J* present their schema. The algorithm 5 describes this nodes re-integration.

3.4.4 Example. After converting the Synthetic Global Schema (Figure 4) into a Business View Schema as explained in above section, we obtain the Business View Schema (Figure 5). As we can see, this figure is quite "simple" and is synthetic enough to be displayed to end-users. Such visualisation will support exploration, navigation or querying operators.

4 IMPLEMENTATION OF A DEKG

Despite Distributed solutions for Knowledge Graphs already exist (like Akutan by Ebay [2]), it has not been clearly defined in academic

Algorithm 5: Business View: schema reducing

```
for each "prop" P node do
    if links(P).length = 1 then
        N = links(P).nodes[0]
        for each property Pr of P do
            N.properties[Pr.name] = Pr
        Delete links(P)
        Delete P
for each "rel" Rn node do
    Create a link L
    L.type = Rn.name
    for each property P of Rn do
        L.P = Rn.P
    Delete links(Rn)
    Delete Rn
```

papers as far as we know, especially for the Enterprise Knowledge Graph. The current well known decentralised one is a federated approach of EKGs allow small subsets of information or knowledge to be linked together, despite being in separated Knowledge Bases [11].

In order to improve maintenability, scalability and high availability of the DEKG we propose a specific architecture (see Figure 6) based on specific DEKG components that can be implemented as services. This proposal is inspired from Federated Databases [28] and its adaptation to the Data Warehouses [6].

The figure 6 follows the workflow of figure 1, from the sources (top of the figure) to the users (bottom). The different components were designed to follow the different steps of the DEKG construction process described at Section 3.2: the Data Component objective to produce the source schema (Step #1; Section 3.2.2) which creates the Sources Schemata; the DEKG Management System is in charge of Step #2 (Section 3.2.3), building and storing the Global Schema; endly the EKG App objective is to build the Business View for the User described in Step #3 (Section 3.2.4), and send queries to the DEKGMS. The following sections introduce those main components of this architecture.

4.1 Data Components.

Our proposition of architecture contains numerous components named "Data Component". Their aim is to interpret the data inside one or multiple data sources which are all related by its location (for instance, in a Business Unit). As we're working in a "Knowledge Graph" environment, the Data Component will have to create a graph schema representing the data of the source. It is also responsible for the link between the schema sent to the "DEKG Management System" and the data contained in the source.

All those Data Components act as "bridges" between the Business View managed by the DEKG Schema component and the data sources. Indeed, when the EKG will be queried, Data Component will be also queried in order to obtain the corresponding data. So, they have to translate the query coming from the Business View to match the queried source (querying a SQL, a document, a CSV...). They all are implemented differently depending on the information the organisation wants to expose for each source. Furthermore, the Data Components ensures the availability to the information. As they're quite small, they can be scaled both horizontally and vertically to ensure both a great scalability and availability to the users. Finally, those components also insure the performance of the overall system. They can for instance cache either data or information, or even queries to answer faster to the queries made by the Decentralised Enterprise Knowledge Graph (DEKG) Management System.

4.2 DEKG Management System.

As shown in Figure 6, a bigger component named "DEKG Management System" is proposed. Its aim is to manage the User and

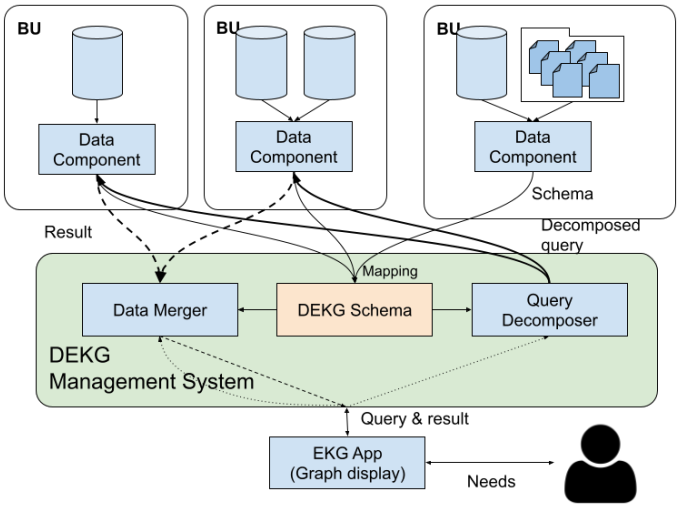

Figure 6: Decentralised EKG architecture components

Applications queries and communicate with every Data Component to answer to the user queries. It builds and manages all DEKG schemata and generates at the end the Business View. The answers, the schema, and queries must be transparent to the user as if it was querying a centralised system. The Management System is divided in three essential components : the "Query Decomposer", the "DEKG Schema", and the "Data Merger".

The most important and most complex component of the DEKG Management System is the **"DEKG Schema"** component. It integrates, stores, and maps the different schemata - not the actual data - coming from all the Data Components included in the Decentralised Enterprise Knowledge Graph. It is in that specific component that the Global Schema of the section 3.3 is constructed and stored.

The **Query Decomposer** is the component receiving the queries from the User/App. Its goal is to break down the user query into sub-queries, which will be sent to the corresponding Data Components using the DEKG schema. The **Data Merger** goal is to get the different responses from the Data Components and merge them back as a single response using the original user query and the DEKG Schema. That unified response is sent to the user.
To do so, the Query Decomposer must use the previous Global Schema to expand the user query into subsets of queries for each source, concurrently run them and all subset responses. Those are received by the Data Merger, which will need to aggregate them, and manage the inconsistencies between all sources [19, 23].

4.3 Querying the DEKG Synthetic View

Finally, the component named **"EKG App"** in the Figure 6 represents the Human Machine Interface between the end-user and the Decentralised Enterprise Knowledge Graph. This component allows the user to visualise the Business View as specified in the Section 3.4 and shown on Figure 5. The Application goal is also to query the whole DEKG from a unified endpoint.

5 EXPERIMENTAL RESULTS OF THE DEKG

To evaluate our DEKG, we decided to implement real-world open-data on a real use case. Our EKG would allow a user of Toulouse or its surrounding cities to know which vaccination centre they can go, depending either on their city name, or postal code. In France, a Postal Code can cover multiple cities or a city can have multiple postal codes: thus the importance to be able to search both by "City" or "Postal Code".

To do so, we integrated the french Toulouse city Opendata (Communes Toulouse[1]; Code postaux Toulouse[2]) containing data on all Cities of Toulouse and ZIP Codes; and a COVID-19 dataset from France (Centres de Vaccination[3]) representing all available vaccination centres in France.

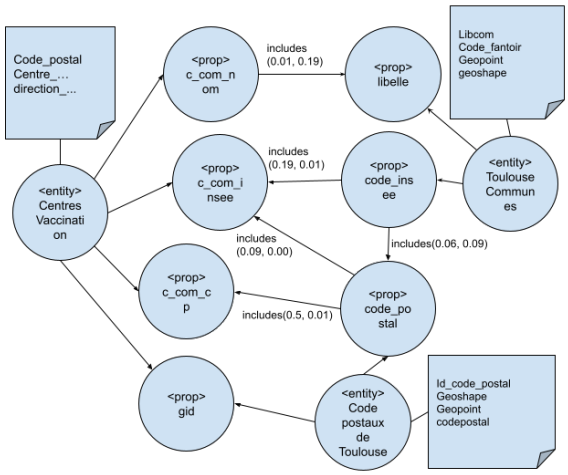

Figure 7: Open Data Schema Matching

As those open-data are all flat files, there were not any relationships in their initial schemata. The only applicable rule in that specific case was the Rule 5 (creating includes between properties). After Global Schema reducing in Figure 7, we can see that we are able to get indirect relationships between entities. That example has shown us than even with if not all rules are applicable on data sources, our approach is able to highlight and create relationships between data sources.

To continue our tests, we ran the algorithm onto our first School example, but also on enterprise employees skills data, acquired with internal survey. All the results of the different steps execution times, and the number of nodes and links contained in both the Global Schema and the Business View are presented on the Table 2. We can see that when the amount of data grows, the Schema Matching is the step taking the most time. Also, the Business View has a much more reduced number of nodes and links compared to the Global Schema; making it much more understandable by the end-user, especially when the data is clean, and therefore successfully matched.

6 CONCLUSION

Organisations really need a Unified Views of their data in order to strengthen their data management. We presented in this paper

[1]https://data.toulouse-metropole.fr/explore/dataset/communes/information/, accessed 2021-05-11
[2]https://data.toulouse-metropole.fr/explore/dataset/codes-postaux-de-toulouse/information/, accessed 2021-05-11
[3]https://www.data.gouv.fr/fr/datasets/relations-commune-cms/, accessed 2021-05-11

Table 2: Algorithm runs on different datasets

Dataset	Execution time (ms)					Global Schema		Business View	
	Source Load	Schema Load	Schema Matching	Schema Reducing	Total	Nodes	Links	Nodes	Links
School Example	6.141	2.75	3.992	1.224	35.221	22	31	5	6
France OpenData	96.283	29.796	424.383	3.237	563.481	52	56	10	13
Company Team of Teams	12.362	36.748	1:02.683 (m:ss)	65.805	1:02.807 (m:ss)	281	7766	214	7681

the Decentralised Enterprise Knowledge Graph as one solution to build a Unified View of the whole organisation data, through our Business View. Thus, we offered a more organisation-oriented definition of the Enterprise Knowledge Graph and a decentralised architecture that can be implemented in an enterprise. Using the sources schemata and schema matching, our Decentralised Knowledge Graph is able to generate a Business View of all the data and data-sources. This approach has been tested against sample data, but also on real-life data from different public sources of multiple providers.

This Decentralised Enterprise Knowledge Graph can be used by organisation to build Enterprise Knowledge Graph which are not copying the original sources, while still allowing its users to query the source data from a single unified point. Our Business View method allow to show the stored Global Schema to non-technical end-users in an easy and understandable way. This DEKG can be used in a lot of applications the Enterprise Knowledge Graph currently used today, for instance building Data Catalogues of organisations.

Despite our architecture being scalable in terms of sources integration, the Global Schema building might grow in computational complexity as the sources grow. Thus, we plan on working on more specific processes to help the Global Schema update when the sources schemata are updated, while keeping the integrity of the schema. Using graph embeddings to enhance our current rule-set is also planned. Also, we'll need to work and integrate existing methods allowing the queries decomposition and responses merging, that we did not describe and evaluate in this current paper. Finally, another future work is to handle data duplication and inconsistency when the same data can be queried from multiple sources. We've worked until now on non-duplicated and consistent data that showed us the feasibility of our DEKG, but we might face difficulties and challenges when working with low-quality data.

REFERENCES

[1] [n.d.]. Neo4j Data Import: Moving Data from RDBMS to Graph.

[2] 2019. Akutan: A Distributed Knowledge Graph Store.

[3] Sarawat Anam, Yang Sok Kim, Byeong Ho Kang, and Qing Liu. 2016. Adapting a knowledge-based schema matching system for ontology mapping. In *Proceedings of the Australasian Computer Science Week Multiconference*. ACM, Canberra Australia, 1–10. https://doi.org/10.1145/2843043.2843048

[4] Mithun Balakrishna, Munirathnam Srikanth, and Lymba Corporation. 2008. Automatic Ontology Creation from Text for National Intelligence Priorities Framework (NIPF). *OIC 2008* (2008), 5.

[5] Giacomo Bergami, Matteo Magnani, and Danilo Montesi. [n.d.]. A Join Operator for Property Graphs. ([n. d.]), 9.

[6] Stefan Berger and Michael Schrefl. 2008. From Federated Databases to a Federated Data Warehouse System. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. IEEE, Waikoloa, HI, 394–394. https://doi.org/10.1109/HICSS.2008.178

[7] Andreas Blumauer. 2014. From Taxonomies over Ontologies to Knowledge Graphs.

[8] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, Paris, 1001–1012. https://doi.org/10.1109/ICDE.2018.00094

[9] Andreia Dal and José Maria. 2012. Simple Method for Ontology Automatic Extraction from Documents. *IJACSA* 3, 12 (2012). https://doi.org/10.14569/IJACSA.2012.031206

[10] Lisa Ehrlinger and Wolfram Wöß. 2016. Towards a Definition of Knowledge Graphs. In *Joint Proceedings of the Posters and Demos Track of 12th International Conference on Semantic Systems*. 4.

[11] Mikhail Galkin, Soren Auer, Haklae Kim, and Simon Scerri. 2016. Integration Strategies for Enterprise Knowledge Graphs. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. IEEE, Laguna Hills, CA, 242–245. https://doi.org/10.1109/ICSC.2016.24

[12] Aldo Gangemi, Valentina Presutti, Diego Reforgiato Recupero, Andrea Giovanni Nuzzolese, Francesco Draicchio, and Misael Mongiovì. 2017. Semantic Web Machine Reading with FRED. *SW* 8, 6 (Aug. 2017), 873–893. https://doi.org/10.3233/SW-160240

[13] Jose Manuel Gomez-Perez, Jeff Z. Pan, Guido Vetere, and Honghan Wu. 2017. Enterprise Knowledge Graph: An Introduction. In *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, Jeff Z. Pan, Guido Vetere, Jose Manuel Gomez-Perez, and Honghan Wu (Eds.). Springer International Publishing, Cham, 1–14. https://doi.org/10.1007/978-3-319-45654-6_1

[14] Lushan Han, Tim Finin, Cynthia Parr, Joel Sachs, and Anupam Joshi. 2008. RDF123: From Spreadsheets to RDF. In *The Semantic Web - ISWC 2008*, Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan (Eds.). Vol. 5318. Springer Berlin Heidelberg, Berlin, Heidelberg, 451–466. https://doi.org/10.1007/978-3-540-88564-1_29 Series Title: Lecture Notes in Computer Science.

[15] Wael H.Gomaa and Aly A. Fahmy. 2013. A Survey of Text Similarity Approaches. *IJCA* 68, 13 (April 2013), 13–18. https://doi.org/10.5120/11638-7118

[16] Lan Jiang and Felix Naumann. 2020. Holistic primary key and foreign key detection. *J Intell Inf Syst* 54, 3 (June 2020), 439–461. https://doi.org/10.1007/s10844-019-00562-z

[17] Martin Junghanns, André Petermann, Niklas Teichmann, Kevin Gómez, and Erhard Rahm. [n.d.]. Analyzing Extended Property Graphs with Apache Flink. ([n. d.]), 9.

[18] Kendall Clark. 2017. What is a Knowledge Graph.

[19] Maurizio Lenzerini. [n.d.]. Data Integration: A Theoretical Perspective. ([n. d.]), 15.

[20] Anan Marie and Avigdor Gal. [n.d.]. Managing Uncertainty in Schema Matcher Ensembles. ([n. d.]), 15.

[21] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, Panos Vassiliadis. 2000. *Fundamentals of Data Warehouses*. Springer Berlin Heidelberg.

[22] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. [n.d.]. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. ([n. d.]), 13.

[23] M. Tamer Özsu and Patrick Valduriez. 2011. *Principles of Distributed Database Systems, Third Edition*. Springer New York, New York, NY. https://doi.org/10.1007/978-1-4419-8834-8

[24] Heiko Paulheim. 2016. Knowledge graph refinement: A survey of approaches and evaluation methods. *SW* 8, 3 (Dec. 2016), 489–508. https://doi.org/10.3233/SW-160218

[25] Erhard Rahm and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. *The VLDB Journal* 10, 4 (Dec. 2001), 334–350. https://doi.org/10.1007/s007780100057

[26] Franck Ravat and Yan Zhao. 2019. Data Lakes: Trends and Perspectives. In *Database and Expert Systems Applications*, Sven Hartmann, Josef Küng, Sharma Chakravarthy, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil (Eds.). Vol. 11706. Springer International Publishing, Cham, 304–313. https://doi.org/10.1007/978-3-030-27615-7_23

[27] Tomer Sagi. [n.d.]. Non-binary evaluation measures for big data integration. ([n. d.]), 22.

[28] Amit P. Sheth and James A. Larson. 1990. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.* 22, 3 (Sept. 1990), 183–236. https://doi.org/10.1145/96602.96604

[29] Amit Singhal. 2012. Introducing the Knowledge Graph: things, not strings.

[30] Dezhao Song, Frank Schilder, Shai Hertz, Giuseppe Saltini, Charese Smiley, Phani Nivarthi, Oren Hazai, Dudi Landau, Mike Zaharkin, Tom Zielund, Hugo Molina-Salgado, Chris Brew, and Dan Bennett. 2019. Building and Querying an Enterprise Knowledge Graph. *IEEE Trans. Serv. Comput.* 12, 3 (May 2019), 356–369. https://doi.org/10.1109/TSC.2017.2711600

[31] Varish Mulwad. 2010. *T2LD - An automatic framework for extracting, interpreting and representing tables as Linked Data*. Ph.D. Dissertation. Faculty of the Graduate School of the University of Maryland.

[32] Boris Villazon-Terrazas, Nuria Garcia-Santa, Yuan Ren, Alessandro Faraotti, Honghan Wu, Yuting Zhao, Guido Vetere, and Jeff Z. Pan. 2017. Knowledge Graph Foundations. In *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, Jeff Z. Pan, Guido Vetere, Jose Manuel Gomez-Perez, and Honghan Wu (Eds.). Springer International Publishing, Cham, 17–55. https://doi.org/10.1007/978-3-319-45654-6_2

[33] Boris Villazon-Terrazas, Nuria Garcia-Santa, Yuan Ren, Kavitha Srinivas, Mariano Rodriguez-Muro, Panos Alexopoulos, and Jeff Z. Pan. 2017. Construction of Enterprise Knowledge Graphs (I). In *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, Jeff Z. Pan, Guido Vetere, Jose Manuel Gomez-Perez, and Honghan Wu (Eds.). Springer International Publishing, Cham, 87–116. https://doi.org/10.1007/978-3-319-45654-6_4