Merging data sources based on semantics, contexts and trust

Lovro Šubelj, David Jelenc, Eva Zupančič, Dejan Lavbič, Denis Trček, Marjan Krisper and Marko Bajec

Abstract—Matching and merging of data from heterogeneous sources is a common need in various scenarios. Despite numerous algorithms proposed in the recent literature, there is a lack of general and complete solutions combining different dimensions arising during the matching and merging execution. We propose a general framework, and accompanying algorithms, that allow joint control over various dimensions of matching and merging. To achieve superior performance, standard (relational) data representation is enriched with semantics and thus elevated towards the real world situation. Data sources are merged using collective entity resolution and redundancy elimination algorithms that are managed through the use of different contexts – user, data and also trust contexts. Introduction of trust allows for an adequate trust management and efficient security assurance which is, besides a general solution for matching and merging, the main novelty of the proposition.

Index Terms—merging data, semantic elevation, context, trust management, entity resolution, redundancy elimination.

1 Introduction

With the recent advent of *Semantic Web* and open (on-line) data sources, merging of data from heterogeneous sources is rapidly becoming a common need in various fields. Different scenarios of use include analyzing heterogeneous datasets collectively, enriching data with some on-line data source or reducing redundancy among datasets by merging them into one. Literature provides several state-of-the-art approaches for matching and merging, although there is a lack of general solutions combining different dimensions arising during the matching and merging execution. We propose a general and complete solution that allows a joint control over these dimensions.

Data sources commonly include not only relational data, but also semantically enriched data. Thus a state-of-the-art solution should employ semantically elevated algorithms, to fully exploit the data at hand. However, due to a vast diversity of data sources, also an adequate data architecture has to be employed. In particular, the architecture should support all types and formats of data, and provide appropriate data for each algorithm. As algorithms favor different representations and levels of semantics behind the data, architecture should be structured appropriately.

Due to different origin of (heterogeneous) data sources, the trustworthiness (or accuracy) of their data can often be questionable. Specially, when many such datasets are merged, the results are likely to be inexact. A common approach for dealing with data sources that provide untrustworthy or conflicting statements, is the use of trust management systems and techniques. Thus matching and merging should be advanced to a trust-aware level, to jointly optimize trustworthiness of data and accuracy of matching or merging. Such collective optimization can significantly improve over other approaches.

The article proposes a general framework for matching and merging execution. An adequate data architecture enables either pure relational data, in the form of networks, or semantically enriched data, in the form of ontologies. Different datasets are merged using collective entity resolution and redundancy elimination algorithms, enhanced with trust management techniques. Algorithms are managed through the use of different contexts that characterize each particular execution, and can be used to jointly control various dimensions of variability of matching and merging execution.

The rest of the article is structured as follows. The following section gives a brief overview of the related work, focusing mainly on trust-aware matching and merging. Next, section 3, presents employed data architecture and discusses semantic elevation of the proposition. Section 4 formalizes the notion of trust and introduces the proposed trust management techniques. General framework, and accompanying algorithms, for matching and merging are presented in section 5, and further discussed in section 6. Section 7 concludes the article.

2 Related work

Recent literature proposes several state-of-the-art solutions for matching and merging data sources. Relevant work and approaches exist in the field of data integration [1], [2], [3], [4], data deduplication [5],

• *L. Šubelj, D. Jelenc, E. Zupančič, D. Lavbič, D. Trček, M. Krisper and M. Bajec are with University of Ljubljana, Faculty of Computer and Information Science.*

[6], [7], information retrieval, schema and ontology matching [8], [9], [10], [11], and (relational) entity resolution [1], [12], [13]. However, the propositions mainly address only selected issues of more general matching and merging problem. In particular, approaches only partially support the variability of the execution; commonly only homogeneous sources, with predefined level of semantics, are employed; or the approaches discard the trustworthiness of data and sources of origin.

Literature also provides various trust-based, or trust-aware, approaches for matching and merging [14], [15]. Although they formally exploit trust in the data, they do not represent a general or complete solution. Mainly, they explore the idea of *Web of Trust*, to model trust or belief in different entities. Related work (on *Web of Trust*) exists in the fields of identity verification [16], information retrieval [17], [18], social network analysis [19], [20], data mining and pattern recognition [21], [22]. Our work also relates to more general research of trust management and techniques that provide formal means for computing with trust (e.g. [23]).

3 Data Architecture

An adequate data architecture is of vital importance for efficient matching and merging. Key issues arising are as follows: (1) architecture should allow for data from heterogeneous sources, commonly in various formats; (2) semantical component of data should be addressed properly; and (3) architecture should also deal with (partially) missing and uncertain data.

To achieve superior performance, we propose a three level architecture (Fig. 3). Standard relational data representation on the bottom level (*data level*) is enriched with semantics (*semantic level*) and thus elevated towards the topmost real world level (*abstract level*). Datasets on data level are represented with networks, when the semantics are employed through the use of ontologies.

Every dataset is (preferably) represented on data and semantic level. Although both describe the same set of entities on abstract level, the representation on each level is independent from the other. This separation resides from the fact that different algorithms of matching and merging execution privilege different representations of data – either pure relational or semantically elevated representation. Separation thus results in more accurate and efficient matching and merging, moreover, representations can complement each other in order to boost the performance.

The following section gives a brief introduction to networks, used for data level representation. Section 3.2 describes ontologies and semantic elevation of data level (i.e. semantic level). Proposed data architecture is formalized and further discussed in section 3.3.

3.1 Representation with networks

Most natural representation of any relational domain are *networks*. They are based upon mathematical objects called *graphs*. Informally speaking, graph consists of a collection of points, called *vertices*, and links between these points, called *edges* (Fig. 1). Let V_N, E_N be a set of vertices, edges for some graph N respectively. We define N as $N = (V_N, E_N)$ where

$$V_N = \{v_1, v_2 \dots v_n\}, \quad (1)$$

$$E_N \subseteq \{\{v_i, v_j\} |\ v_i, v_j \in V_N \wedge i < j\}. \quad (2)$$

Edges are sets of vertices, hence they are not directed (*undirected graph*). In the case of *directed graphs* equation (2) rewrites to

$$E_N \subseteq \{(v_i, v_j) |\ v_i, v_j \in V_N \wedge i \neq j\}, \quad (3)$$

where (v_i, v_j) is an edge from v_i to v_j. The definition can be further generalized by allowing multiple edges between two vertices and *loops* (edges that connect vertices with themselves). Such graphs are called *multigraphs* (Fig. 1 (b)).

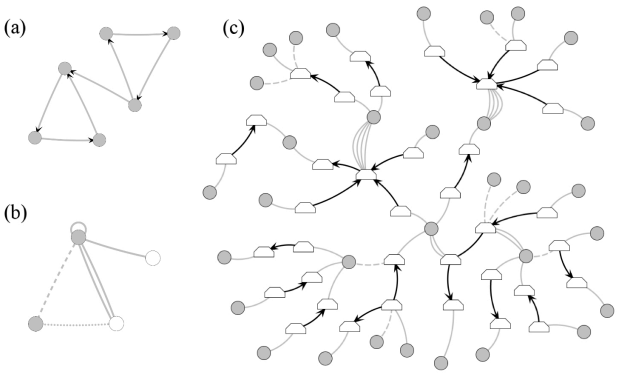

Fig. 1. (a) directed graph; (b) labeled undirected multigraph (labels are represented graphically); (c) network representing a group of related traffic accidents (round vertices correspond to participants and cornered correspond to vehicles).

In practical applications we commonly strive to store some additional information along with the vertices and edges. Formally, we define *labels* or *weights* for each node and edge in the graph – they represent a set of properties that can also be described using two attribute functions

$$A_{V_N}: \quad V_N \to \Sigma_1^{V_N} \times \Sigma_2^{V_N} \times \dots, \quad (4)$$

$$A_{E_N}: \quad E_N \to \Sigma_1^{E_N} \times \Sigma_2^{E_N} \times \dots, \quad (5)$$

$A_N = (A_{V_N}, A_{E_N})$, where $\Sigma_i^{V_G}$, $\Sigma_i^{E_G}$ are sets of all possible vertex, edge attribute values respectively.

Networks are most commonly seen as labeled, or weighted, multigraphs with both directed and undirected edges (Fig. 1 (c)). Vertices of a network represent some entities, and edges represent relations between them. A (relational) dataset, represented with a network on the data level, is thus defined as (N, A_N).

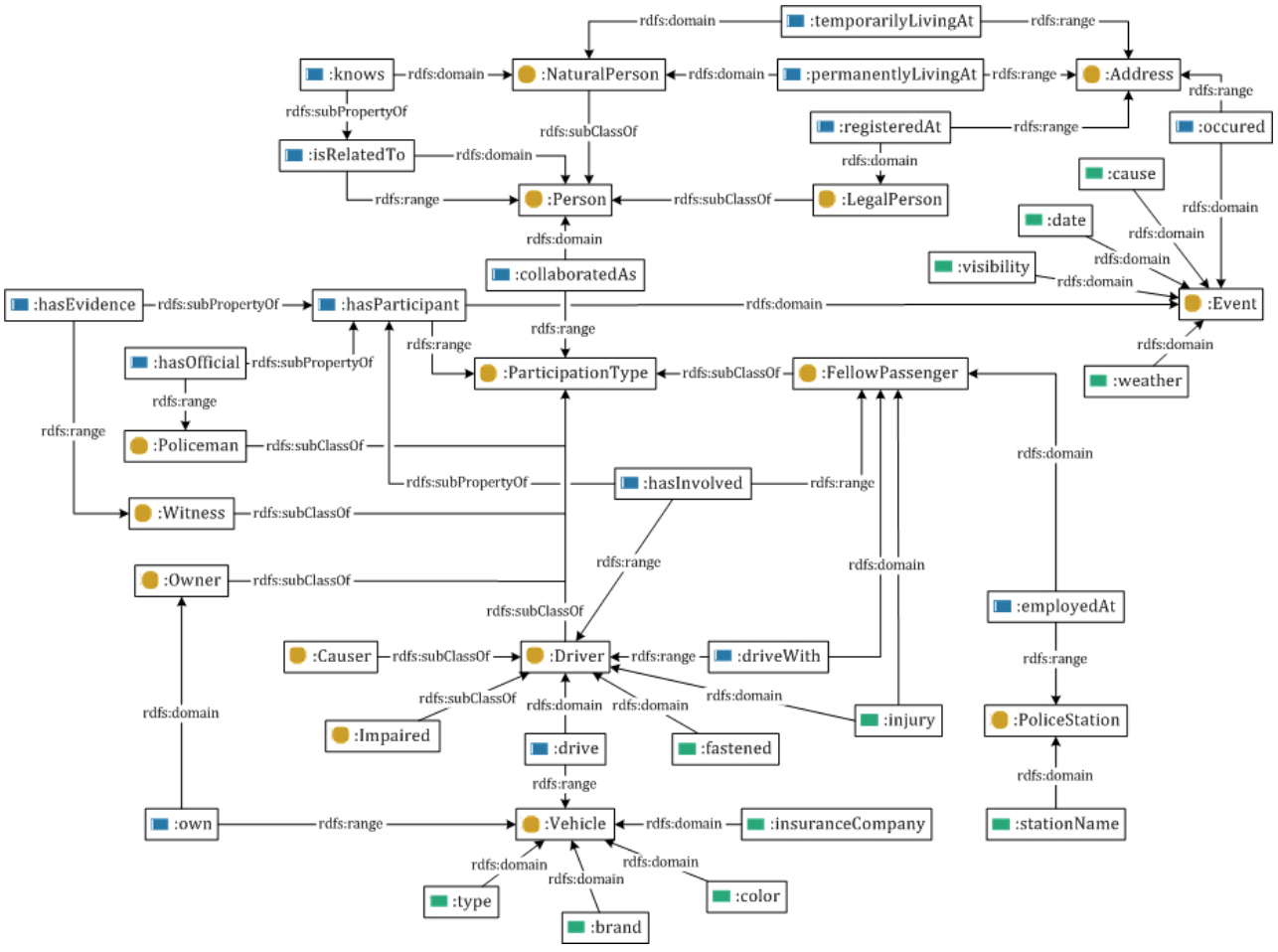

Fig. 2. Ontology representing various classes, relations and attributes related to traffic accidents and automobile insurance domain. Classes are colored with orange, relations with blue and attributes with green. Key concepts of the ontology are *Event*, *Person*, *Driver*, *Witness*, *Owner* and *Vehicle*.

3.2 Semantic elevation using ontologies

Ontologies are a tool for specifying the semantics of terminology systems in a well defined and unambiguous manner [24] (Fig. 2). They can simply be defined as a network of *entities*, restricted and annotated with a set of *axioms*. Let E_O, A_O be the sets of entities, axioms for some ontology O respectively. Dataset, represented with an ontology on semantic level, is defined as $O = (E_O, A_O)$ where

$$E_O \subseteq E^C \cup E^I \cup E^R \cup E^A, \quad (6)$$

$$A_O \subseteq \{a |\ E_O^a \subseteq E_O \wedge a \text{ axiom on } E_O^a\}. \quad (7)$$

Entities E_O consist of *classes* E^C (concepts), *individuals* E^I (instances), *relations* E^R (among classes and individuals) and *attributes* E^A (properties of classes); and axioms A_O are assertions (over entities) in a logical form that together comprise the overall theory described by ontology O.

This article focuses on ontologies based on *descriptive logic* that, besides assigning meaning to axioms, enable also reasoning capabilities. The latter can be used to compute consequences of the previously made assumptions (queries), or to discover non-intended consequences and inconsistencies within the ontology.

With the advent of *Semantic Web*, ontologies are rapidly gaining importance. One of the most prominent applications of ontologies is in the domain of semantic interoperability (among heterogeneous software systems). While pure semantics concerns the study of meanings, *semantic elevation* means to achieve semantic interoperability and can be considered as a subset of information integration (including data access, aggregation, correlation and transformation).

Thus one of the key aspects of semantic elevation is to derive a common representation of classes, individuals, relations and attributes within some ontology. We employ a concept of *knowledge chunks* [9], where each entity is represented with its *name* and a set of semantic *relations* (or attributes), their *values* and (ontology) *identifiers*. All of the data about a certain entity is thus transformed into *attribute-value* format, with an identifier of the data source of origin appended to each value. Knowledge chunks, denoted $k \in K$, thus provide a (common) synthetic representation of an ontology that is used during the matching and

merging execution. For more details on knowledge chunks, and their construction from a *RDF(S)* (*Resource Description Framework Schema*) repository or an *OWL* (*Web Ontology Language*) ontology, see [9], [25].

Notion of knowledge chunks is introduced also on data level. Hence, each network is represented in the same, easily maintainable, form, allowing for common matching and merging algorithms. Exact description of the transformation between networked data and knowledge chunks is not given, although it is very similar to the definition of inferred axioms in equation (12).

3.3 Three level architecture

As previously stated, every dataset is (independently) represented on three levels – *data*, *semantic* and *abstract level* (Fig. 3). Bottommost *data level* holds data in a pure relational format (i.e. networks), mainly to facilitate state-of-the-art relational algorithms for matching. Next level, *semantic level*, enriches data with semantics (i.e. ontologies), to further enhance matching and to promote semantic merging execution. Data on both levels represent entities of topmost *abstract level*, which serves merely as an abstract (artificial) representation of all the entities, used during matching and merging execution.

The information captured by data level is a subset of that of semantic level. Similarly, the information captured by semantic level is a subset of that of abstract level. This *information-based* view of the architecture is seen in Fig. 3 (a). However, representation on each level is completely independent from the others, due to absolute separation of data. This provides an alternative *data-based* view, seen in Fig. 3 (b).

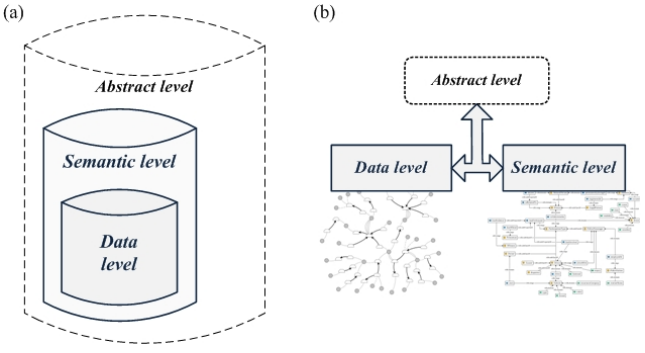

Fig. 3. (a) information-based view of the data architecture; (b) data-based view of the data architecture.

To manage data and semantic level independently (or jointly), a mapping between the levels is required. In practice, data source could provide datasets on both, data and semantic level. The mapping is in that case trivial (i.e. given). However, more commonly, data source would only provide datasets on one of the levels, and the other has to be *inferred*.

Let (N, A_N) be a dataset, represented as a network on data level. Without loss for generality, we assume that N is an undirected network. Inferred ontology $(\tilde{E}_{\tilde{O}}, \tilde{A}_{\tilde{O}})$ on semantic level is defined with

$$\tilde{E}_C = \{vertex, edge\}, \quad (8)$$
$$\tilde{E}_I = V_N \cup E_N, \quad (9)$$
$$\tilde{E}_R = \{isOf, isIn\}, \quad (10)$$
$$\tilde{E}_A = \{A_{V_N}, A_{E_N}\} \quad (11)$$

and

$$\begin{aligned}\tilde{A}_{\tilde{O}} = &\{v\ isOf\ vertex |\ v \in V_N\} \\ &\cup \{e\ isOf\ edge |\ e \in E_N\} \\ &\cup \{v\ isIn\ e |\ v \in V_N \wedge e \in E_N \wedge v \in e\} \\ &\cup \{v.A_{V_N} = a |\ v \in V_N \wedge A_{V_N}(v) = a\} \\ &\cup \{e.A_{E_N} = a |\ e \in E_N \wedge A_{E_N}(e) = a\}.\end{aligned} \quad (12)$$

We denote $\mathcal{I}_N : (N, A_N) \mapsto (\tilde{E}_{\tilde{O}}, \tilde{A}_{\tilde{O}})$. One can easily see that $\mathcal{I}_N^{-1} \circ \mathcal{I}_N$ is an identity (transformation preserves all the information).

On the other hand, given a dataset (E_O, A_O), represented with an ontology on semantic level, inferred (undirected) network $(\tilde{N}, \tilde{A}_{\tilde{N}})$ on data level is defined with

$$\tilde{V}_{\tilde{N}} = E_O \cap E^I, \quad (13)$$
$$\tilde{E}_{\tilde{N}} = \{E_O^a \cap E^I |\ a \in A_O \wedge E_O^a \subseteq E_O\} \quad (14)$$

and

$$\tilde{A}_{\tilde{V}_{\tilde{N}}} : \ \tilde{V}_{\tilde{N}} \to E^C \times E^A, \quad (15)$$
$$\tilde{A}_{\tilde{E}_{\tilde{N}}} : \ \tilde{E}_{\tilde{N}} \to E^R. \quad (16)$$

Instances of ontology are represented with the vertices of the network, and axioms with its edges. Classes and relations are, together with the attributes, expressed through vertex, edge attribute functions.

We denote $\mathcal{I}_O : (E_O, A_O) \mapsto (\tilde{N}, \tilde{A}_{\tilde{N}})$. Transformation $\mathcal{I}_O$ discards purely semantic information (e.g. relations between classes), as it cannot be represented on the data level. Thus $\mathcal{I}_O$ cannot be inverted as $\mathcal{I}_N$. However, all the data, and data related information, is preserved (e.g. relations among individuals, and individuals and classes).

Due to limitations of networks, only axioms, relating at most two individuals in E_O, can be represented with the set of edges $\tilde{E}_{\tilde{N}}$ (equation (14)). When this is not sufficient, *hypernetworks* (or *hypergraphs*[1]) should be employed instead. Nevertheless, networks should suffice in most cases.

One more issue has to be stressed. Although $\mathcal{I}_N$ and $\mathcal{I}_O$ give a "common" representation of every dataset, the transformations are completely different. For instance, presume (N, A_N) and (E_O, A_O) are (given) representations of the same dataset. Then

1. Hypergraphs are similar to ordinary graphs only that the edges can connect multiple vertices.

$\mathcal{I}_N(N, A_N) \neq (E_O, A_O)$ and $\mathcal{I}_O(E_O, A_O) \neq (N, A_N)$ in general – inferred ontology, network does not equal given ontology, network respectively. The former non-equation resides in the fact that network (N, A_N) contains no knowledge of the (pure) semantics within ontology (E_O, A_O); and the latter resides in the fact that $\mathcal{I}_O$ has no information of the exact representation used for (N, A_N). Still, transformations $\mathcal{I}_N$ and $\mathcal{I}_O$ can be used to manage data on a common basis.

Last, we discuss three key issues regarding an adequate data architecture, presented in section 3. Firstly, due to variety of different data formats, a mutual representation must be employed. As the data on both data and semantic level is represented in the form of knowledge chunks (section 3.2), every piece of data is stored in exactly the same way. This allows for common algorithms of matching and merging and makes the data easily manageable.

Furthermore, introduction of knowledge chunks naturally deals also with missing data. As each chunk is actually a set of attribute-value pairs, missing data only results in smaller chunks. Alternatively, missing data could be randomly inputted from the rest and treated as extremely uncertain or mistrustful (section 4).

Secondly, semantical component of data should be addressed properly. Proposed architecture allows for simple (relational) data and also semantically enriched data. Hence no information is discarded. Moreover, appropriate transformations make all data accessible on both data and semantic level, providing for specific needs of each algorithm.

Thirdly, architecture should deal with (partially) missing and uncertain or mistrustful data, which is thoroughly discussed in the following section.

4 Trust and Trust Management

When merging data from different sources, these are often of different origin and thus their trustworthiness (or accuracy) can be questionable. For instance, personal data of participants in a traffic accident is usually more accurate in the police record of the accident, then inside participants' social network profiles. Nevertheless, an attribute from less trusted data source can still be more accurate than an attribute from more trusted one – a relationship status (e.g. single or married) in the record may be outdated, while such type of information is inside the social network profiles quite often up-to-date.

A complete solution for matching and merging execution should address such problems as well. A common approach for dealing with data sources that provide untrustworthy or conflicting statements, is the use of *trust management (systems)*. These are, alongside the concept of trust, both further discussed in sections 4.1 and 4.2.

4.1 Definition of trust

Trust is a complex psychological-sociological phenomenon. Despite of, people use term trust in everyday life widely, and with very different meanings. Most common definition states that trust is an *assured reliance on the character, ability, strength, or truth of someone or something.*

In the context of computer networks, trust is modeled as a relationship between entities. Formally, we define a *trust relationship* as

$$\omega_E : \quad E \times E \to \Sigma^E \tag{17}$$

where E is a set of entities and Σ^E a set of all possible, numerical or descriptive, trust values. ω_E thus represents one entity's attitude towards another and is used to model trust(worthiness) T_E of all entities in E. To this end, different trust modeling methodologies and systems can be employed, from qualitative to quantitative (e.g. [14], [15], [23]).

We introduce trust on three different levels. First, we define trust on the level of data source, in order to represent trustworthiness of the source in general. Let S be the set of all data sources. Their trust is defined as $T_S : S \to [0, 1]$, where higher values of T_S represent more trustworthy source.

Second, we define trust on the level of attributes (or semantic relations) within the knowledge chunks. The trust in attributes is naturally dependent on the data source of origin, and is defined as $T_{A_s} : A_s \to [0, 1]$, where A_s is the set of attributes for data source $s \in S$. As before, higher values of T_{A_s} represent more trustworthy attribute.

Last, we define trust on the level of knowledge chunks. Despite the trustworthiness of data source and attributes within some knowledge chunk, its data can be (semantically) corrupted, missing or otherwise unreliable. This information is captured using trustworthiness of knowledge chunks, and again defined as $T_K : K \to [0, 1]$, where K is a set of all knowledge chunks. Although the trust relationships (equation (17)), needed for the evaluation of trustworthiness of data sources and attributes, are (mainly) defined by the user, computation of trust in knowledge chunks can be fully automated using proper evaluation function (section 4.2).

Three levels of trust provide high flexibility during matching and merging. For instance, attributes from more trusted data sources are generally favored over those from less trusted ones. However, by properly assigning trust in attributes, certain attributes from else less trusted data sources can prevail. Moreover, trust in knowledge chunks can also assist in revealing corrupted, and thus questionable, chunks that should be excluded from further execution.

Finally, we define trust in some particular value within a knowledge chunk, denoted *trust value* T. This is the value in fact used during merging and matching

execution and is computed from corresponding trusts on all three levels. In general, T can be an arbitrary function of T_S, T_{A_s} and T_K. Assuming independence, we calculate trust value by concatenating corresponding trusts,

$$T = T_S \circ T_{A_s} \circ T_K. \quad (18)$$

Concatenation function $\circ$ could be a simple multiplication or some fuzzy logic operation (trusts should in this case be defined as fuzzy sets).

4.2 Trust management

During merging and matching execution, trust values are computed using trust management algorithm based on [15]. We begin by assigning trust values T_S, T_{A_s} for each data source, attribute respectively (we actually assign trust relationships). Commonly, only a subset of values must necessarily be assigned, as others can be inferred or estimated from the first. Next, trust values for each knowledge chunk are not defined by the user, but are calculated using the *chunk evaluation function* f_{eval} (i.e. $T_K = f_{eval}$).

An example of such function is a *density of inconsistencies* within some knowledge chunk. For instance, when attributes *Birth* and *Age* of some particular knowledge chunk mismatch, this can be seen as an inconsistency. However, one must also consider the trust of the corresponding attributes (and data sources), as only inconsistencies among trustworthy attributes should be considered. Formally, density of inconsistencies is defined as

$$f_{eval}(k) = \frac{\hat{N}_{inc}(k) - N_{inc}(k)}{\hat{N}_{inc}(k)}, \quad (19)$$

where k is a knowledge chunk, $k \in K$, $N_{inc}(k)$ the number of inconsistencies within k and $\hat{N}_{inc}(k)$ the number of all possible inconsistencies.

Finally, after all individual trusts T_S, T_{A_s} and T_K have been assigned, trust values T are computed using equation (18). When merging takes place and two or more data sources (or knowledge chunks) provide conflicting attribute values, corresponding to the same (resolved) entity, trust values T are used to determine actual attribute value in the resulting data source (or knowledge chunk). For further discussion on trust management during matching and merging see section 5.

5 Matching and Merging Data Sources

Merging data from heterogeneous sources can be seen as a two-step process. The first step resolves the real world entities of abstract level, described by the data on lower levels, and constructs a mapping between the levels. This mapping is used in the second step that actually merges the datasets at hand. We denote these subsequent steps as *entity resolution* (i.e. matching) and *redundancy elimination* (i.e. merging).

Matching and merging is employed in various scenarios. As the specific needs of each scenario vary, different dimensions of variability characterize every matching and merging execution. These dimensions are managed through the use of *contexts* [9], [26]. Contexts allow a formal definition of specific needs arising in diverse scenarios and a joint control over various dimensions of matching and merging execution.

The following section discusses the notion of contexts more throughly and introduces different types of contexts used. Next, sections 5.2, 5.3 describe employed *entity resolution* and *redundancy elimination* algorithms respectively. The general framework for matching and merging is presented and formalized in section 5.4, and discussed in section 6.

5.1 Contexts

Every matching and merging execution is characterized by different dimensions of variability of the data, and mappings between. *Contexts* are a formal representation of all possible operations in these dimensions, providing for specific needs of each scenario. Every execution is thus characterized with the contexts it defines (Fig. 4), and can be managed and controlled through their use.

The idea of contexts originates in the field of requirements engineering, where it has been applied to model domain variability [26]. It has just recently been proposed to model also variability of the matching execution [9]. Our work goes one step further as it introduces contexts, not bounded only to user or scenario specific dimensions, but also data related and trust contexts.

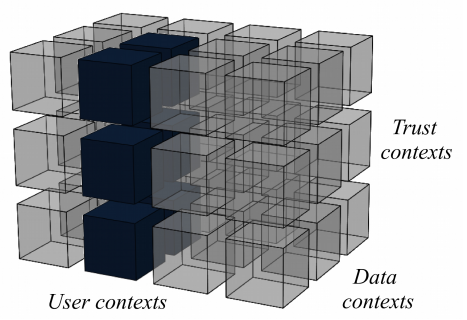

Fig. 4. Characterization of merging and matching execution defining one context in user dimension, two contexts in data dimension and all contexts in trust dimension.

Formally, we define a context C as

$$C: \quad D \rightarrow \{true, false\}, \quad (20)$$

where D can be any simple or composite domain. A context simply limits all possible values, attributes, relations, knowledge chunks, datasets, sources or other, that are considered in different parts of matching and

merging execution. Despite its simple definition, a context can be a complex function. It is defined on any of the architecture levels, preferably on all. Let C_A, C_S and C_D represent the same context on abstract, semantic and data level respectively. The joint context is defined as

$$C_J = C_A \wedge C_S \wedge C_D. \quad (21)$$

In the case of missing data (or contexts), only appropriate contexts are considered. Alternatively, contexts could be defined as fuzzy sets, to address also the noisiness of data. In that case, a fuzzy AND operation should be used to derive joint context C_J.

We distinguish between three types of contexts due to different dimensions characterized (Fig. 4).

- *user* User or scenario specific contexts are used mainly to limit the data and control the execution. This type coincides with dimensions identified in [9]. An example of user context is a simple selection or projection of the data.
- *data* Data related contexts arise from dealing with relational or semantic data, and various formats of data. Missing or corrupted data can also be managed through the use of these contexts.
- *trust* Trust and data uncertainty contexts provide for an adequate trust management and efficient security assurance between and during different phases of execution. An example of trust context is a definition of required level of trustworthiness of data or sources.

Detailed description of each context is out of scope of this article. For more details on (user) contexts see [9].

5.2 Entity resolution

First step of matching and merging execution is to resolve the real world entities on abstract level, described by the data on lower levels. Thus a mapping between the levels (entities) is constructed and used in consequent merging execution. Recent literature proposes several state-of-the-art approaches for entity resolution (e.g. [5], [1], [12], [13], [6]). A naive approach is a simple pairwise comparison of attribute values among different entities. Although, such an approach could already be sufficient for flat data, this is not the case for relational data, as the approach completely discards relations between the entities. For instance, when two entities are related to similar entities, they are more likely to represent the same entity. However, only the attributes of the related entities are compared, thus the approach still discards the information if related entities resolve to the same entities – entities are even more likely to represent the same entities when their related entities resolve to, not only similar, but the same entities. An approach that uses this information, and thus resolves entities altogether (in a collective fashion), is denoted *collective* (relational) entity resolution algorithm.

We employ a state-of-the-art *(collective) relational clustering* algorithm proposed in [12]. To further enhance the performance, algorithm is semantically elevated and adapted to allow for proper and efficient trust management.

The algorithm is actually a greedy agglomerative clustering approach. Entities (on lower levels) are represented as a group of clusters C, where each cluster represents a set of entities that resolve to the same entity on abstract level. At the beginning, each (lower level) entity resides in a separate cluster. Then, at each step, the algorithm merges two clusters in C that are most likely to represent the same entity (most *similar* clusters). When the algorithm unfolds, C holds a mapping between the entities on each level (i.e. maps entities on lower levels through the entities on abstract level).

During the algorithm, similarity of clusters is computed using a *joint similarity measure* (equation (28)), combining *attribute*, *relational* and *semantic similarity*. First is a basic pairwise comparison of attribute values; second introduces relational information into the computation of similarity (in a collective fashion); and third represents semantic elevation of the algorithm.

Let $c_i, c_j \in C$ be two clusters of entities. Using knowledge chunk representation, attribute cluster similarity is defined as

$$sim_A(c_i, c_j) = \sum_{k_{i,j} \in c_{i,j} \wedge a \in k_{i,j}} trust(k_i.a, k_j.a) sim_A(k_i.a, k_j.a), \quad (22)$$

where $k_{i,j} \in K$ are knowledge chunks, $a \in A_s$ is an attribute and $sim_A(k_i.a, k_j.a)$ similarity between two attribute values. (Attribute) similarity between two clusters is thus defined as a weighted sum of similarities between each pair of values in each knowledge chunk. Weights are assigned due to trustworthiness of values – trust in values $k_i.a$ and $k_j.a$ is computed using

$$trust(k_i.a, k_j.a) = \min\{T(k_i.a), T(k_j.a)\}. \quad (23)$$

Hence, when even one of the values is uncertain or mistrustful, similarity is penalized appropriately, to prevent matching based on (likely) incorrect information.

For computation of similarity between actual attribute values $sim_A(k_i.a, k_j.a)$ (equation (22)), different measures have been proposed. *Levenshtein distance* [27] measures edit distance between two strings – number of insertions, deletions and replacements that traverse one string into the other. Another class of similarity measures are *TF-IDF*[2]-based measures (e.g. *Cos TF-IDF* and *Soft TF-IDF* [28], [29]). They treat

2. *Term Frequency-Inverse Document Frequency.*

attribute values as a bag of words, thus the order of words in the attribute has no impact on the similarity. Other attribute measures are also *Jaro* [30] and *Jaro-Winkler* [31] that count number of matching characters between the attributes.

Different similarity measures prefer different types of attributes. *TF-IDF*-based measures work best with longer strings (e.g. descriptions), when other prefer shorter strings (e.g. names). For numerical attributes, an alternative measure has to be employed (e.g. simple evaluation, followed by a numerical comparison). Therefore, when computing attribute similarity for a pair of clusters, different attribute measures are used with different attributes (equation (22)).

Using data level representation, we define a *neighborhood* for vertex $v \in V_N$ as

$$nbr(v) \quad = \quad \{v_n |\ v_n \in V_N \wedge \{v, v_n\} \in E_N\} \quad (24)$$

and cluster $c \in C$ as

$$nbr(c) = \{c_n |\ c_n \in C \wedge v \in c \wedge c_n \cap nbr(v) \neq \emptyset\}. \quad (25)$$

Neighborhood of a vertex is defined as a set of connected vertices. Similarly, neighborhood of a cluster is defined as a set of clusters, connected through the vertices within.

For a (collective) relational similarity measure, we adapt a *Jaccard coefficient* [12] measure for trust-aware (relational) data. *Jaccard coefficient* is based on *Jaccard index* and measures the number of common neighbors of two clusters, considering also the size of the clusters' neighborhoods – when the size of neighborhoods is large, the probability of common neighbors increases. We define

$$sim_R(c_i, c_j) = \frac{\sum_{c_n \in nbr(c_i) \cap nbr(c_j)} trust(e^T_{in}, e^T_{jn})}{|nbr(c_i) \cup nbr(c_j)|} \quad (26)$$

where e^T_{in}, e^T_{jn} is the most trustworthy edge connecting vertices in c_n and c_i, c_j respectively (for the computation of $trust(e^T_{in}, e^T_{jn})$, a knowledge chunk representation of e^T_{in}, e^T_{jn} is used). (Relational) similarity between two clusters is defined as the size of a common neighborhood (considering also the trustworthiness of connecting relations), decreased due to the size of clusters' neighborhoods. Entities related to a relatively large set of entities that resolve to the same entities on abstract level, are thus considered to be similar.

Alternatively, one could use some other similarity measure like *Adar-Adamic similarity* [32], random walk measures, or measures considering also the ambiguity of attributes or higher order neighborhoods [12].

For the computation of the last, semantic, similarity, we propose a random walk like approach. Using a semantic level representation of clusters $c_i, c_j \in C$, we do a number of random assumptions (queries) over underlying ontologies. Let N_{ass} be the number of times the consequences (results) of the assumptions made matched, $\tilde{N}_{ass}$ number of times the consequences were undefined (for at least one ontology) and $\hat{N}_{ass}$ the number of all assumptions made. Furthermore, let N^T_{ass} be the trustworthiness of ontology elements used for reasoning in assumptions that matched (computed as a sum of products of trusts on the paths of reasoning, similar as in equation (23)). Semantic similarity is then defined as

$$sim_S(c_i, c_j) \quad = \quad \frac{N^T_{ass}(c_i, c_j)}{\hat{N}_{ass}(c_i, c_j) - \tilde{N}_{ass}(c_i, c_j)}. \quad (27)$$

Similarity represents the trust in the number of times ontologies produced the same consequences, not considering assumptions that were undefined for some ontology. As the expressiveness of different ontologies vary, and some of them are even inferred from relational data, many of the assumptions could be undefined for some ontology. Still, for $\hat{N}_{ass}(c_i, c_j) - \tilde{N}_{ass}(c_i, c_j)$ large enough, equation (27) gives a good approximation of semantic similarity.

Using attribute, relational and semantic similarity (equations (22), (26) and (27)) we define a joint similarity for two clusters as

$$sim(c_i, c_j) = \frac{1}{\delta_A + \delta_R + \delta_S} \quad (28)$$
$$(\delta_A sim_A(c_i, c_j) + \delta_R sim_R(c_i, c_j) + \delta_S sim_S(c_i, c_j)),$$

where δ_A, δ_R and δ_S are weights, set due to the scale of relational and semantical information within the data. For instance, setting $\delta_R = \delta_S = 0$ reduces the algorithm to a naive pairwise comparison of attribute values, which should be used when no relational or semantic information is present.

Finally, we present the collective clustering algorithm employed for entity resolution (algorithm 1).

First, the algorithm initializes clusters C and priority queue of similarities Q, considering the current set of clusters (lines 1-5). Each cluster represents at most one entity as it is composed out of a single knowledge chunk. Algorithm then, at each iteration, retrieves currently the most similar clusters and merges them (i.e. matching of resolved entities), when their similarity is greater than threshold θ_S (lines 7-11). As clusters are stored in the form of knowledge chunks, matching in line 11 results in a simple concatenation of chunks. Next, lines 12-17 update similarities in the priority queue Q, and lines 18-22 insert (or update) also neighbors' similarities (required due to relational similarity measure). When the algorithm terminates, clusters C represent chunks of data resolved to the same entity on abstract level. This mapping between the entities (i.e. their knowledge chunk representations) is used to merge the data in the next step.

Threshold θ_S represents minimum similarity for two clusters that are considered to represent the same entities. Optimal value should be estimated from the data.

Algorithm 1 Collective entity resolution

Initialize clusters as $C = \{\{k\} | \ k \in K\}$
Initialize priority queue as $Q = \emptyset$
for $c_i, c_j \in C$ **and** $sim(c_i, c_j) \geq \theta_S$ **do**
  $Q.insert(sim(c_i, c_j), c_i, c_j)$
end for
while $Q \neq \emptyset$ **do**
  $(sim(c_i, c_j), c_i, c_j) \leftarrow Q.pop()$ {Most similar.}
  if $sim(c_i, c_j) < \theta_S$ **then**
    return C
  end if
  $C \leftarrow C - \{c_i, c_j\} \cup \{c_i \cup c_j\}$ {Matching.}
  for $(sim(c_x, c_k), c_x, c_k) \in Q$ **and** $x \in \{i, j\}$ **do**
    $Q.remove(sim(c_x, c_k), c_x, c_k)$
  end for
  for $c_k \in C$ **and** $sim(c_i \cup c_j, c_k) \geq \theta_S$ **do**
    $Q.insert(sim(c_i \cup c_j, c_k), c_i \cup c_j, c_k)$
  end for
  for $c_n \in nbr(c_i \cup c_j)$ **do**
    for $c_k \in C$ **and** $sim(c_n, c_k) \geq \theta_S$ **do**
      $Q.insert(sim(c_n, c_k), c_n, c_k)$ {Or update.}
    end for
  end for
end while
return C

Three more aspects of the algorithm ought to be discussed. Firstly, pairwise comparison of all clusters during the execution of the algorithm is computationally expensive, specially in early staged of the algorithm. Authors in [12] propose an approach in which they initially find groups of chunks that could possibly resolve to the same entity. In this way, the number of comparisons can be significantly decreased.

Secondly, due to the nature of (collective) relational similarity measures, they are ineffective when none of the entities has already been resolved (e.g. in early stages of the algorithm). As the measure in equation (26) counts the number of common neighbors, this always evaluates to 0 in early stages (in general). Thus relative similarity measures should be used after the algorithm has already resolved some of the entities, using only attribute and semantic similarities.

Thirdly, in the algorithm we implicitly assumed that all attributes, (semantic) relations and other, have the same names or identifiers in every dataset (or knowledge chunk). Although, we can probably assume that all attributes within datasets, produced by the same source, have same and unique names, this cannot be generalized.

We propose a simple, yet effective, solution. The problem at hand could be denoted *attribute resolution*, as we merely wish to map attributes between the datasets. Thus we can use the approach proposed for entity resolution. Entities are in this case attributes that are compared due to their names, and also due to different values they hold; and relations between entities (attributes) represent co-occurrence in the knowledge chunks. As certain attributes commonly occur with some other attributes, this would further improve the resolution.

Another possible improvement is to address also the attribute values in a similar manner. As different values can represent the same underlying value, *value resolution*, done prior to attribute resolution, can even further improve the performance.

5.3 Redundancy elimination

After the entities, residing in the data, have been resolved (section 5.2), the next step is to eliminate the redundancy and merge the datasets at hand. This process is somewhat straightforward as all data is represented in the form of knowledge chunks. Thus we merely need to merge the knowledge chunks, resolved to the same entity on abstract level. Redundancy elimination is done entirely on semantic level, to preserve all the knowledge inside the data.

When knowledge chunks hold disjoint data (i.e. attributes), they can simply be concatenated together. However, commonly various chunks would provide values for the same attribute and, when these values are inconsistent, they need to be handled appropriately. A naive approach would count only the number of occurrences of some value, when we consider also their trustworthiness, to determine the most probable value for each attribute.

Let $c \in C$ be a cluster representing some entity on abstract level (resolved in the previous step), let $k_1, k_2 \dots k_n \in c$ be its knowledge chunks and let k^c be the *merged knowledge chunk*, we wish to obtain. Furthermore, for some attribute $a \in A.$, let X^a be a random variable measuring the true value of a and let X^a_i be the random variables for a in each knowledge chunk it occurs (i.e. $k_i.a$). Value of attribute a for the merged knowledge chunk k^c is then defined as

$$\arg\max_v P(X^a = v | \bigwedge_i X^a_i = k_i.a). \tag{29}$$

Each attribute is thus assigned the most probable value, given the evidence observed (i.e. values $k_i.a$). By assuming pair-wise independence among X^a_i (conditional on X^a) and uniform distribution of X^a equation (29) simplifies to

$$\arg\max_v \prod_i P(X^a_i = k_i.a | X^a = v). \tag{30}$$

Finally, conditional probabilities in equation (30) are approximated with trustworthiness of values,

$$P(X^a_i | X^a) \approx \begin{cases} T(k_i.a) & \text{for } k_i.a = v \text{ (31a)} \\ 1 - T(k_i.a) & \text{for } k_i.a \neq v \text{ (31b)} \end{cases}$$

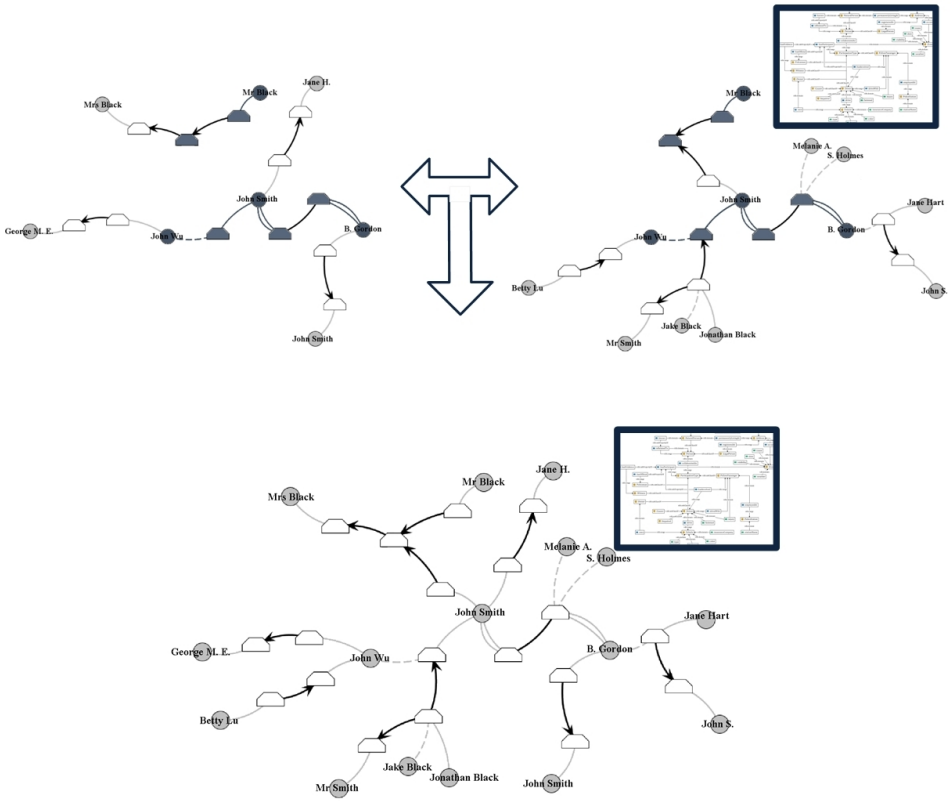

Fig. 5. Entity resolution and redundancy elimination for two relational datasets, representing a group of traffic accidents (above). One dataset is also annotated with ontology in Fig. 2.

hence

$$k^c.a = \arg\max_v \prod_{k_i.a=v} T(k_i.a) \prod_{k_i.a\neq v} 1 - T(k_i.a). \quad (32)$$

Only knowledge chunks containing attribute a are considered.

We present the proposed redundancy elimination algorithm (algorithm 2).

Algorithm 2 Redundancy elimination

1: Initialize knowledge chunks K^C
2: **for** $c \in C$ **and** $a \in A$. **do**
3: $k^c.a = \arg\max_v \prod_{k\in c \wedge k.a=v} T(k.a) \prod_{k\in c \wedge k.a\neq v} 1 - T(k.a)$
4: **end for**
5: **return** K^C

The algorithm uses knowledge chunk representation of semantic level. First, it initializes merged knowledge chunks $k^c \in K^C$. Then, for each attribute $k^c.a$, it finds the most probable value among all given knowledge chunks (line 3). When the algorithm unfolds, knowledge chunks K^C represent a merged dataset, with resolved entities and eliminated redundancy. Each knowledge chunk k^c corresponds to unique entity on abstract level, and each attribute holds the most trustworthy value.

At the end, only the data that was actually provided by some data source, should be preserved. Thus all inferred data (through $\mathcal{I}_N$ or $\mathcal{I}_O$; section 3.3) is discarded, as it is merely an artificial representation needed for (common) entity resolution and redundancy elimination. Still, all provided data and semantical information is preserved and properly merged with the rest. Hence, although redundancy elimination is done on semantic level, resulting dataset is given on both data and semantic level (that complement each other).

Last, we discuss the assumptions of independence among X_i^a and uniform distribution of X^a. Clearly, both assumptions are violated, still the former must be made in order for the computation of most probable value to be feasible. However, the latter can be eliminated when distribution of X^a can be approximated from some large-enough dataset.

5.4 General framework

Proposed entity resolution and redundancy elimination algorithms (sections 5.2, 5.3) are integrated into a general *framework* for matching and merging (Fig. 6). Framework represents a complete solution, allowing a joint control over various dimensions of matching and merging execution. Each component of the framework is briefly presented in the following, and further discussed in section 6.

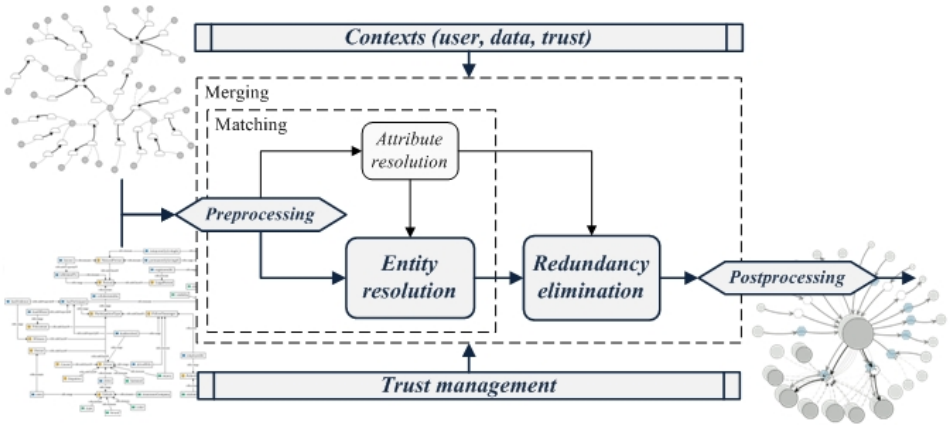

Fig. 6. General framework for matching and merging data from heterogeneous sources.

Initially, data from various sources is preprocessed appropriately. Every network or ontology is transformed into a knowledge chunk representation and, when needed, also inferred on an absent architecture level (section 3.3). After preprocessing is done, all data is represented in the same, easily manageable, form, allowing for common, semantically elevated, subsequent analyses.

Prior to entity resolution, attribute resolution is done (section 5.2). The process resolves and matches attributes in the heterogeneous datasets, using the same algorithm as for entity resolution. As all data is represented in the form of knowledge chunks, this actually unifies all the underlying networks and ontologies.

Next, proposed entity resolution and redundancy elimination algorithms are employed (sections 5.2, 5.3). The process thus first resolves entities in the data, and then uses this information to eliminate the redundancy and to merge the datasets at hand. Algorithms explore not only the relations in the data, but also the semantics behind it, to further improve the performance.

Last, postprocessing is done, in order to discard all artificially inferred data and to translate knowledge chunks back to the original network or ontology representation (section 3). Throughout the entire execution, components are jointly controlled through (defined) user, data and trust contexts (section 5.1). Furthermore, contexts also manage the results of the algorithms, to account for specific needs of each scenario.

Every component of the framework is further enhanced, to allow for proper trust management, and thus also for efficient security assurance. In particular, all the similarity measures for entity resolution are trust-aware, moreover, trust is even used as a primary evidence in the redundancy elimination algorithm. The introduction of trust-aware and security-aware algorithms represents the main novelty of the proposition.

6 Discussion

The following section discusses key aspects of the proposition.

Proposed framework for matching and merging represents a general and complete solution, applicable in all diverse areas of use. Introduction of contexts allows a joint control over various dimensions of matching and merging variability, providing for specific needs of each scenario. Furthermore, data architecture combines simple (relational) data with semantically enriched data, which makes the proposition applicable for any data source. Framework can thus be used as a general solution for merging data from heterogeneous sources, and also merely for matching.

The fundamental difference between matching, including only attribute and entity resolution, and merging, including also redundancy elimination, is, besides the obvious, in the fact that merged data is read-only. Since datasets, obtained after merging, do not necessarily resemble the original datasets, the data cannot be altered thus the changes would apply also in the original datasets. Alternative approach is to merely match the given datasets and to merge them only on demand. When altering matched data, user can change the original datasets (that are in this phase still represented independently) or change the merged dataset (that was previously demanded for), in which case he must also provide an appropriate strategy, how the changes should be applied in the original datasets.

Proposed algorithms employ relational data, semantically enriched with ontologies. With the advent of *Semantic Web*, ontologies are gaining importance mainly due to availability of formal ontology languages. These standardization efforts promote several notable uses of ontologies like assisting in communication between people, achieving interoperability (communication) among heterogeneous software systems and improving the design and quality of software systems. One of the most prominent applications is in the domain of semantic interoperability.

While pure semantics concerns the study of meanings, semantic elevation means to achieve semantic interoperability and can be considered as a subset of information integration (including data access, aggregation, correlation and transformation). Semantic elevation of proposed matching and merging framework represents one major step towards this end.

Use of trust-aware techniques and algorithms introduces several key properties. Firstly, an adequate trust management provides means to deal with uncertain or questionable data sources, by modeling trustworthiness of each provided value appropriately. Secondly, algorithms jointly optimize not only entity resolution or redundancy elimination of provided datasets, but also the trustworthiness of the resulting datasets. The latter can substantially increase the accuracy. Thirdly, trustworthiness of data can be used also for security reasons, by seeing trustworthy values as more secure. Optimizing the trustworthiness of matching and merging thus also results in an efficient security assurance.

Next, we discuss the main rationale behind the introduction of contexts. Although, contexts are merely a way to guide the execution of some algorithm, their definition is relatively different from that of any simple parameter. The execution is controlled with mere definition of the contexts, when in the case of parameters, it is controlled by assigning different values. For instance, when default behavior is desired, the parameters still need to be assigned, when in the case of contexts, the algorithm is used as it is. For any general solution, working with heterogeneous clients, such behavior can significantly reduce the complexity.

As different contexts are used jointly throughout matching and merging execution, they allow a collective control over various dimensions of variability. Furthermore, each execution is controlled and also characterized with the context it defines, which can be used to compare and analyze different executions or matching and merging algorithms.

Last, we briefly discuss a possible disadvantage of the proposed framework. As the framework represents a general solution, applicable in all diverse domains, the performance of some domain-specific approach or algorithm can still be superior. However, such approaches commonly cannot be generalized and are thus inappropriate for practical (general) use.

7 Conclusion

Article proposes a general framework, and accompanying algorithms, for matching and merging data from heterogeneous sources. All the proposed algorithms are trust-aware, which enables the use of appropriate trust management and security assurance techniques. An adequate data architecture supports not only (pure) relational data, but also semantically enriched data, to promote semantically elevated analyses that thoroughly explore the data at hand. Matching and merging is done using state-of-the-art collective entity resolution and redundancy elimination algorithms that are managed and controlled through the use of different contexts. Framework thus allows a joint control over various dimensions of variability of matching and merging execution.

Further work will include empirical evaluation of the proposition on some large testbeds. Next, soft computing and fuzzy logic will be introduced for contexts manipulation and trust management, to provide for inexactness of contexts and ambiguity of trust phenomena. Moreover, trust management will be advanced to a collective approach, resulting also in a collective redundancy elimination algorithm. Last, all proposed algorithms will be adapted to hypernetworks (or hypergraphs), to further generalize the framework.

Acknowledgment

This work has been supported by the Slovene Research Agency *ARRS* within the research program P2-0359.

References

[1] I. Bhattacharya and L. Getoor, "Iterative record linkage for cleaning and integration," in *Proceedings of the ACM SIGKDD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2004, pp. 11–18.

[2] W. W. Cohen, "Data integration using similarity joins and a word-based information representation language," *ACM Transactions on Information Systems*, vol. 18, no. 3, pp. 288–321, 2000.

[3] M. Hernandez and S. Stolfo, "The merge/purge problem for large databases," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 127–138, 1995.

[4] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the ACM SIGMOD Symposium on Principles of Database Systems*, 2002, pp. 233–246.

[5] R. Ananthakrishna, S. Chaudhuri, and V. Ganti, "Eliminating fuzzy duplicates in data warehouses," in *Proceedings of the International Conference on Very Large Data Bases*, 2002, pp. 586–597.

[6] D. Kalashnikov and S. Mehrotra, "Domain-independent data cleaning via anlysis of entity-relationship graph," *ACM Transactions on Database Systems*, vol. 31, no. 2, pp. 716–767, 2006.

[7] A. Monge and C. Elkan, "The field matching problem: Algorithms and applications," *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pp. 267–270, 1996.

[8] S. Castano, A. Ferrara, and S. Montanelli, "Matching ontologies in open networked systems: Techniques and applications," *Journal on Data Semantics*, pp. 25–63, 2006.

[9] ——, "Dealing with matching variability of semantic web data using contexts," in *Proceedings of the International Conference on Advanced Information Systems Engineering*, 2010, to be presented.

[10] J. Euzenat and P. Shvaiko, *Ontology matching*. Springer-Verlag, 2007.

[11] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *Journal on Very Large Data Bases*, vol. 10, no. 4, pp. 334–350, 2001.

[12] I. Bhattacharya and L. Getoor, "Collective entity resolution in relational data," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, p. 5, 2007.

[13] X. Dong, A. Halevy, and J. Madhavan, "Reference reconciliation in complex information spaces," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005, pp. 85–96.
[14] M. Nagy, M. Vargas-Vera, and E. Motta, "Managing conflicting beliefs with fuzzy trust on the semantic web," in *Proceedings of the Mexican International Conference on Advances in Artificial Intelligence*, 2008, pp. 827–837.
[15] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *Proceedings of the International Semantic Web Conference*, 2003, pp. 351–368.
[16] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings of the IEEE Symposium on Security and Privacy*, 1996, pp. 164–173.
[17] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Keinberg, "Automatic resource compilation by analyzing hyperlink structure and associated text," *Proceedings of the International World Wide Web Conference*, pp. 65–74, 1998.
[18] T. Joachims, "A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization," in *Proceedings of the International Conference on Machine Learning*, 1997, pp. 143–151.
[19] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 57–66.
[20] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604–632, 1999.
[21] H. Kautz, B. Selman, and M. Shah, "Referral web: combining social networks and collaborative filtering," *Communications of the ACM*, vol. 40, no. 3, pp. 63–65, 1997.
[22] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: an open architecture for collaborative filtering of netnews," in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, 1994, pp. 175–186.
[23] D. Trcek, "A formal apparatus for modeling trust in computing environments," *Mathematical and Computer Modelling*, vol. 49, no. 1-2, pp. 226–233, 2009.
[24] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.
[25] S. Castano, A. Ferrara, and S. Montanelli, "The iCoord knowledge model for P2P semantic coordination," in *Proceedings of the Conference on Italian Chapter of AIS*, 2009.
[26] A. Lapouchnian and J. Mylopoulos, "Modeling domain variability in requirements engineering with contexts," in *Proceedings of the International Conference on Conceptual Modeling*. Gramado, Brazil: Springer-Verlag, 2009, pp. 115–130.
[27] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.
[28] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *Proceedings of the IJCAI Workshop on Information Integration on the Web*, 2003, pp. 73–78.
[29] E. Moreau, F. Yvon, and O. Capp, "Robust similarity measures for named entities matching," in *Proceedings of the International Conference on Computational Linguistics*, 2008, pp. 593–600.
[30] M. A. Jaro, "Advances in record linking methodolg as applied to the 1985 census of tampa florida," *Journal of the American Statistical Society*, vol. 84, no. 406, pp. 414–420, 1989.
[31] W. E. Winkler, "String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage." in *Proceedings of the Section on Survey Research Methods*, 1990, pp. 354–359.
[32] L. Adamic and E. Adar, "Friends and neighbors on the web," *Social Networks*, vol. 25, pp. 211–230, 2001.