Computer Science and Information Systems 21(4):1913–1961 https://doi.org/10.2298/CSIS240229065B

Automatic Conceptual Database Design based on Heterogeneous Source Artifacts*

Goran Banjac, Drazen Brdjanin, and Danijela Banjac

Faculty of Electrical Engineering, University of Banja Luka
Patre 5, 78000 Banja Luka, Bosnia and Herzegovina
{goran.banjac,drazen.brdjanin,danijela.banjac}@etf.unibl.org

Abstract. The article presents an approach to the automatic derivation of conceptual database models from heterogeneous source artifacts. The approach is based on the integration of conceptual database models that are derived from source artifacts of one single type by already existing tools, whereby those models possess limited certainty given their limited completeness and correctness. The uncertainty of the automatically derived models from specific source artifacts is expressed and managed through the effectiveness measure of the generation of specific concepts of the input conceptual database models. The approach is implemented by the DBomnia tool – the first online web-based tool enabling automatic derivation of conceptual database models from heterogeneous source artifacts (business process models and textual specifications). DBomnia employs other pre-existing tools to derive conceptual models from sources of the same type and then integrates those models. The case study-based evaluation proves that the implemented approach enables effective automatic derivation of the conceptual database model from a set of heterogeneous source artifacts. Moreover, the automatic derivation of the conceptual database model from a set of heterogeneous source artifacts is more effective than each independent automatic derivation of the conceptual database model from sources of one single type only.

Keywords: AMADEOS, DBomnia, Schema integration, Schema matching, Schema merging, TexToData, UML class diagram, Uncertain schema.

1. Introduction

The database design process undergoes several typical steps [20], whereby the first and the most important step is conceptual design. The result of the conceptual database design is the *conceptual database model* (CDM), which is platform-independent and describes the target database on a high level of abstraction. The result of all subsequent steps, including the database specification for the specific target database management system, can be straightforwardly derived starting from the CDM. This is the main reason why researchers have been significantly interested in the topic of automated CDM design.

Motivation. Automated CDM design has been a research topic since the 1980s [15], and since then a plethora of papers have been published in the field. Although different types

* This article constitutes an extended version of the conference paper entitled *"Towards Automatic Conceptual Database Design based on Heterogeneous Source Artifacts"* presented at the workshop *Modern Approaches in Data Engineering and Information System Design*, 4th of September 2023 Barcelona, Spain.

of artifacts (models, textual specifications, recorded speech, etc.) have been introduced as a source in the automated CDM design, the existing tools enable CDM synthesis based on sources of one single type only, whereby the generated CDMs are not 100% complete nor 100% correct.

Objectives. Since there is still no tool enabling the fully automatic generation of the complete target CDM from sources of one single type, we started to investigate the possibilities of increasing the completeness and correctness of automatically generated target CDMs by deriving them from heterogeneous source artifacts (compared to the CDMs derived from sources of one single type). Accordingly, we define our research objectives as follows:

1. *define an approach and implement a tool enabling automatic CDM derivation from a set of heterogeneous source artifacts,*
2. *evaluate the approach by assessing the CDM derived from a set of heterogeneous source artifacts against the CDMs that are derived from the source artifacts of one single type.*

Contributions. The main contribution of our research is the approach for the integration of incomplete and incorrect CDMs that are automatically derived from specific source artifacts. In our approach, we employ existing, already developed tools for CDM derivation from specific source artifacts, and then we integrate those CDMs. Uncertainty of CDMs automatically derived from specific source artifacts is expressed and managed through the effectiveness measure of generation of specific concepts (classes, attributes, associations, generalizations) of the input CDMs.

The second main contribution is the implemented tool named DBomnia. DBomnia is the first online web-based tool that enables automatic CDM derivation from a set of heterogeneous source artifacts. Currently, DBomnia supports two types of source artifacts: *business process models* (BPMs) and *textual specifications*. DBomnia is a full CASE tool for forward engineering of relational databases.

This article constitutes an extended version of the workshop paper [4], which is extended by: (*i*) formalization of the proposed approach, (*ii*) detailed presentation of the proposed approach, (*iii*) more detailed presentation of the related work, and (*iv*) case study-based evaluation of the proposed approach and implemented tool.

Article organization. This article is structured as follows. After this introduction, the second section presents the related work. The necessary definitions are provided in the third section. The proposed approach is presented in the next three sections. Firstly, an overview of the approach is presented in the fourth section, then the schema matching step is presented in the fifth section, while the sixth section presents the schema merging step. The implemented tool is presented in the seventh section. The eighth section presents the evaluation. The final section concludes the article.

2. Related Work

This section presents the related work. Firstly, we provide an overview of the existing approaches and tools to (semi-)automatic CDM design, followed by an overview of the existing approaches to schema matching and integration.

2.1. (Semi-)automatic CDM Design

The existing approaches and tools for (semi-)automatic CDM design derive the target model from the source artifacts of the same type and can be classified as: *Text-based*, *Model-based*, *Form-based*, and *Speech-based*.

Text-based approaches constitute the oldest and most dominant category. These approaches and tools derive CDMs from textual specifications that are typically unstructured and represented in some *natural language* (NL). They can be further classified (as suggested in [56]) as: (1) *Linguistics-based*, (2) *Pattern-based*, (3) *Case-based*, (4) *Ontology-based*, and (5) *Multiple approaches*.

Most text-based approaches and tools are *linguistics-based*. They use *natural language processing* (NLP) techniques to convert NL text into the CDM. The development of these approaches started with Chen's eleven rules [15] for the translation of English text into the corresponding CDM, which have been further enhanced and extended in [42, 40, 28]. The most important *linguistics-based* tools are: ER-Converter [40], CM-Builder [27], and LIDA [42]. The main representatives of other categories are: *pattern-based* APSARA [44], *case-based* CABSYDD [17], *ontology-based* OMDDE [54], and HBT [56] belonging to the *multiple approaches*.

The existing text-based tools typically support one single source NL (mainly English) and do not provide multilingual support. Only TexToData [13] enables automatic CDM derivation from textual specifications in different source NLs, even with very complex morphology (e.g. Slavic languages). In our approach presented in this article, we employ TexToData as one of the generators of uncertain CDMs in the multi-sourced automatic CDM synthesis.

Model-based approaches and tools emerged as an alternative to those that are text-based, in order to avoid their shortcomings mainly related to the modest effectiveness for languages with complex morphology. The existing approaches and tools take source models that can be represented by a number of different notations, which can be classified (according to [10]) as: (1) *Process-oriented* (e.g. BPMN), (2) *Function-oriented* (e.g. Data

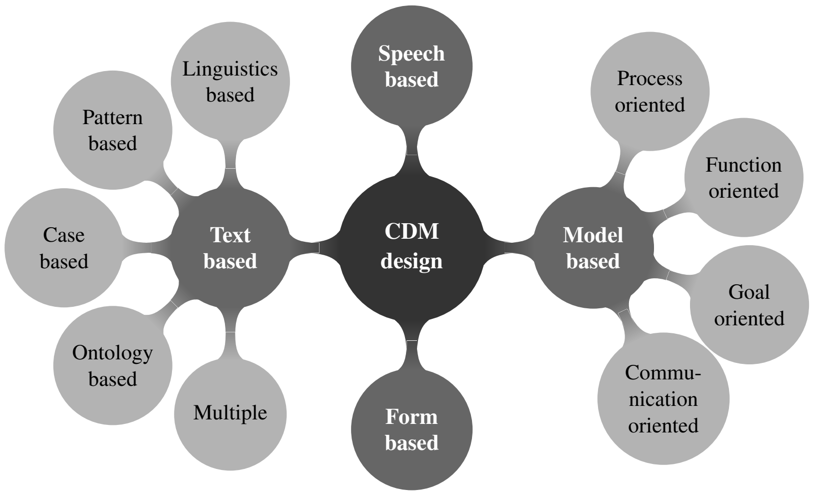

Fig. 1. Taxonomy of approaches to (semi-)automatic CDM design

Flow Diagram), (3) *Communication-oriented* (e.g. Sequence Diagram), and (4) *Goal-oriented* (e.g. TROPOS).

There is still no model-based approach nor tool enabling automatic derivation of the complete target CDM from a source model (regardless of the notation). Only a few papers present a set of formal rules for automatic CDM derivation (e.g. [55, 9]), while the majority give only guidelines and informal rules that do not enable automatic CDM derivation. Most of the proposed tools are actually transformation programs (e.g. [47, 31]) specified in some model-to-model transformation language (e.g. ATL [30]), while only a small number of papers present real CASE tools for automatic model-driven CDM synthesis (e.g. [39, 11]).

There is only one single online model-driven tool named AMADEOS [11], which enables the automatic derivation of an initial CDM from a set of BPMs. The most recent AMADEOS release [53] supports the whole BPM-driven forward database engineering process by using the standard UML notation in all stages. In our approach presented in this article, we employ AMADEOS as one of the generators of uncertain CDMs in the multi-sourced automatic CDM synthesis.

Form-based approaches take *collections of forms* as the source, whereby the most important tools are EDDS [16] and IIS*Case [33].

Speech-based approaches constitute the smallest category – in the existing literature, there is only one paper [12] presenting the SpeeD tool that is able to derive CDM from recorded English speech.

In comparison to the existing approaches that consider only sources of one single type, and in comparison to the existing tools that enable (semi-)automatic CDM derivation from sources of one single type, there is only our paper [4] that considers automatic CDM derivation from heterogeneous source artifacts and corresponding tool that enables automatic CDM derivation from two different types of source artifacts: textual specifications and BPMs.

2.2. Schema Matching and Integration

In the essence of our research objective is the problem of *uncertain schema* integration, i.e. integration of CDMs that are automatically generated by the tools that do not generate 100% correct nor 100% complete models.

Schema integration has been a research topic since the 1980s. It has been heavily investigated and is part of the research to this day. The term *schema integration*, introduced in [5], is defined as the activity of integrating database schemas. It is a generic term that is used in two contexts. In the database design context, it means to produce a global conceptual schema of the proposed database, and in the distributed database management context it means to produce a global schema that represents a virtual view of all databases in such an environment.

Schema integration implies the creation of a unified representation of the schemas from different sources. It consists of two core parts: *schema matching* and *schema merging*. Schema matching is the process of discovering mappings (correspondences) between different schemas, while schema merging is the process of the unification of different schemas based on the discovered mappings [36].

In the 2000s researchers started to investigate the topic of generic model management. In [8], the authors introduce the term *model management* and propose to generalize and integrate model management operations (such as matching and merging) to support generic model management. The term *model* is used to refer to any structured representation of the data (e.g. relational schema, XML, etc.). After that, numerous papers emerged focusing on generic model management operations, such as generic matching (e.g. [35]) and generic merging (e.g. [43]).

Although some papers deal with the uncertainty that is inherent in the schema matching step [36], the state of the art is the integration of the source schemas that can be described as *reliable*. Reliable source schemas are the product of reliable sources (e.g. when integrating schemas of the databases already in use). In this article, we deal with the integration of the models with reduced reliability, because tools for automated CDM derivation do not generate 100% correct nor 100% complete models.

Techniques. Over the years, many schema matching techniques have been introduced. Here we provide a basic taxonomy of matching techniques (according to [7]) with the focus on techniques adopted in our approach.

Basic categorization of schema matching techniques includes *instance-based* and *schema-based* techniques. Instance-based techniques use data instances to discover mappings among schema elements, while schema-based techniques rely only on schema information.

In our approach we deal with the conceptual (database) models, meaning there aren't any data instances that can be used during the schema matching process. Consequently, we are not able to use the instance-based techniques, which makes the schema-based techniques the cornerstone of our approach. Schema-based techniques can be categorized into *element-level* (they consider individual schema elements) and *structure-level* techniques (they consider groups of elements).

Another category of matching techniques is *constraint-based*. Constraint-based techniques use schema constraints such as data types, relationship cardinalities, participation constraints, etc.

Due to the unreliability of the input CDMs (in the sense of the CDM structure), which constitutes the main challenge in our approach, the usage of structure-based techniques is limited (but it is still possible to some extent). Therefore, we have to rely on *linguistic* matching techniques. Linguistic matching techniques use names and/or other textual descriptions of the schema elements to discover mappings between elements of different schemas. Linguistic matching (according to [57]) commonly includes string (e.g. element name) pre-processing (e.g. tokenization, elimination of special characters, stemming, lemmatization, elimination of stop words, etc.), identification of the *semantic similarity* (e.g. using NLP-based techniques) and/or identification of the *syntactic similarity*. Linguistic matching can include techniques of *auxiliary information usage* (e.g. thesaurus usage to find acronyms, synonyms, homonyms, heteronyms, etc.).

Our approach is not limited to single-language input artifacts and automatically generated CDMs, rather our approach limits the language in one execution to a single language (e.g. in one execution all artifacts can be in the English language, but it is possible that in another execution all artifacts are in the German language, etc.). Currently, a combination of multiple languages in one execution is not considered.

Due to the multilingual nature of the input CDMs (in different executions) and considering a large number of supported languages, the semantic analysis constitutes a great challenge and belongs to a separate research project that is out of the scope of this article. Here we are focused on the syntactic similarity measures.

Syntactic similarity measures for string comparison can be categorized (according to [24]) as *character-level* or *token-level*. Character-level measures compare strings on the level of a single character, while token-level measures compare string tokens. Typical representatives of character-level measures are *edit-distance measures* (Levenshtein [32], Damerau–Levenshtein [19], Needleman and Wunsch [38], Smith and Waterman [52], Smith–Waterman–Gotoh [25], Hamming [26], Jaro [29], Jaro–Winkler [58]) and *longest common string* [23]. Typical representatives of token-level measures are *set-based measures* (Jaccard [45], Dice [1]) and *bag-of-tokens measures* (Cosine [18], Euclidean [37], Manhattan [37]).

Some papers consider a combination of character-level and token-level measures. Such measures are called *soft measures* [24]. The basic principle of a soft measure is to apply the token-level measure, but when comparing tokens a character-level measure is used, i.e. two tokens are considered a match if their character-level measure is above some defined *threshold*. The typical usage of a soft measure is to eliminate possible typographical errors on a token level.

If different techniques are used to perform matching, then such combination of techniques is called *hybrid technique* (if different techniques are combined together to perform matching) or *composite technique* (if different techniques are performed independently and only combine the results). Our approach can be classified as a hybrid.

Usage of machine learning techniques. A number of schema matching approaches leverage of machine learning (ML) techniques. Some papers present approaches that apply ML techniques to the problem of combining different matchers (i.e. matching techniques). Examples of such approaches are LSD [21], YAM [22], and ALMa [46].

Another use case for ML techniques in the schema matching process is when such techniques are used as part of a similarity measure. The Automatch system [6] uses probabilistic methods and a knowledge base about schema attributes to calculate matching scores between the attributes of two schemas. Afterwards, an optimization process is carried out to find optimal schema matching (with respect to the sum of the individual attribute matching scores). Schema matching at the element level and structure level is proposed in [2], where ML techniques are used to generate matching results at the element level. Features for ML are constructed using different string similarity metrics and text processing techniques based on entity names. Element level and structure level mappings are combined to produce the final result. In [14], features are constructed using string similarity metrics and semantic information from entity names. These features are used as input to an ML model that predicts entity matchings. In [48], features (about attributes) for ML are constructed based on schema information (data type, not null specification, etc.). These features are used to create corresponding clusters, and attribute mappings are discovered from these clusters. SMAT [59] is based on NLP techniques and uses attribute names and descriptions to obtain schema mappings. Matching between two attributes is proportionate to the similarity of the corresponding sentence pair, where each sentence is constructed from attribute name and description.

Usage of ML techniques in the schema matching process is diverse. Corpus-based matching approach [34] uses a corpus of schemas and mappings in particular domain to augment information about input schemas. Authors argue that augmenting evidence about input schemas provide better matching results. The ADnEV approach [51] presents an additional step in the schema matching task. It uses deep learning to calibrate matching result of other (algorithmic) matchers. PoWareMatch [50] combines algorithmic matching and deep learning to improve human matching results.

Lately, researchers have tried to leverage the recent emergence of (large) language models. Paper [61] presents a novel linguistic schema matching approach that uses fine-tuned pre-trained language model for a more precise calculation of similarity between each pair of attributes in input schemas. The problem of similarity calculation is regarded as a binary text classification problem, where prediction of a correct match is based on a sentence that describes corresponding attributes (concatenation of the attributes' names and descriptions). The SMUTF approach [60] combines rule-based feature engineering, pre-trained language models, and generative large language models to predict the match score (the probability that two columns are matched). The ReMatch approach [49] uses retrieval-enhanced large language models in the process of selecting candidate matches. The approach is designed to aid human matchers throughout their work.

Schema matching approaches that use ML techniques also focus on matching complete and reliable schemas, with the dominant usage of attribute names and descriptions, and attribute values (where possible). However, in our future work we will focus on utilizing ML techniques to improve current results.

3. Definitions

In this section, we provide the necessary definitions.

3.1. CDM Representation

We use the *UML class diagram* [41] to represent the CDMs, thus the representation of the CDMs generated by different generators, as well as the target integrated CDM, is unified.

Each CDM (UML class diagram) contains two packages, ICM_PT and ICM_CD. The ICM_PT package contains definitions of the primitive types (e.g. *Integer*, *Text*, *Double*, etc.), while the ICM_CD package contains the CDM itself.

Definition 1: Let P, E, R, and G be sets of primitive types, classes, associations, and generalizations, respectively. The conceptual database model, $CDM(P, E, R, G)$, is a UML class diagram with the following properties:

1. Each entity type is represented by the corresponding class $e \in E$ of the same name.
2. Let $attrs(e)$ be a set of attributes (*ownedAttribute:Property*) of the class $e \in E$. Each attribute $a \in attrs(e)$ corresponds to an attribute of the respective entity type, whereby its type, denoted $type(a)$, is a primitive type ($type(a) \in P$).
3. A class may contain a single operation, named *PK*, which represents the primary key of the corresponding entity type [53]. The primary key attributes should be listed as the operation parameters, i.e. operation parameters should correspond to the primary key attributes by the name and by the type.

4. Each relationship type is represented by the corresponding binary association $r \in R$ of the same name, whose *memberEnd:Property* attribute is an ordered pair of association ends ($source(r)$ and $target(r)$). The type of each association end is an entity type ($type(source(r)) \in E \wedge type(target(r)) \in E$). The association end multiplicities correspond to the relationship cardinalities and participation constraints. The lower value of the association end multiplicity ($lower(source(r))$ and $lower(target(r))$) corresponds to the participation constraint of the entity type at the opposite end of the association. The upper value of the association end multiplicity ($upper(source(r))$ and $upper(target(r))$) corresponds to the mapping cardinality of the entity type at the opposite end of the association.
5. Each generalization/specialization relationship is represented by the corresponding generalization $g \in G$. It relates *specific* entity type ($specific(g) \in E$) to a more *general* entity type ($general(g) \in E, specific(g) \neq general(g)$).

3.2. Quantitative CDM Assessment

The input in the schema integration process is a set of automatically generated CDMs, which are not 100% correct nor 100% complete. In order to integrate such unreliable CDMs, it is necessary to know their correctness and completeness. The only way to precisely determine such measures for some concrete automatically generated CDM is to manually evaluate it. Considering that we are dealing with automated schema integration, manual evaluation is not an option, but rather approximate or estimated values of those measures should be known in advance. Since all our generators are already evaluated, we propose *a posteriori* approach – use the calculated measures from the evaluation of the CDM generators as the estimated measures of the automatically generated CDMs in the schema integration process.

During the evaluation of generators, we used *recall*, *precision*, and *F-score* as measures for the evaluation of the automatically generated CDM.

Definition 2: If N_c represents the number of correctly generated concepts in the generated CDM and N_m represents the number of missing concepts in the generated CDM, then **recall** (R) constitutes a measure of completeness of the generated CDM, and it is defined as:

$$R = \frac{N_c}{N_c + N_m}\,. \tag{1}$$

Definition 3: If N_c represents the number of correctly generated concepts in the generated CDM and N_w represents the number of incorrectly generated concepts in the generated CDM, then **precision** (P) constitutes a measure of correctness of the generated CDM, and it is defined as:

$$P = \frac{N_c}{N_c + N_w}\,. \tag{2}$$

Definition 4: The **effectiveness**, named *F-score* or *F-measure* (F), is defined as the harmonic mean of precision and recall, i.e.

$$F = \frac{2PR}{P + R}\,. \tag{3}$$

3.3. Input and Output

Previously described measures are calculated separately for each concept of the CDM, i.e. classes, attributes, associations, and generalizations. The F-score, as the effectiveness measure, will be used in the integration approach. Therefore, input in the schema integration process is a set of automatically generated CDMs, as well as estimated measures for each input CDM.

Definition 5: Let F_E be the estimated effectiveness for classes, and F_A be the estimated effectiveness for attributes in the classes, and F_R be the estimated effectiveness for associations, and F_G be the estimated effectiveness for generalizations, then the estimated effectiveness F of the CDM generation process is represented by the $<F_E, F_A, F_R, F_G>$ tuple, i.e. $F =<F_E, F_A, F_R, F_G>$.

Definition 6: The input I in the schema integration process, is a set of n ordered pairs (CDM_i, F_i), where CDM_i is the *i*-th CDM, while F_i represents its estimated F-measures, i.e.

$$\begin{aligned} I = \{(CDM_i, F_i) : CDM_i = CDM(P_i, E_i, R_i, G_i) \wedge \\ F_i =<F_{E_i}, F_{A_i}, F_{R_i}, F_{G_i}> \wedge \\ i = 1, \ldots, n\}\,. \end{aligned} \tag{4}$$

Definition 7: The output of the schema integration process is the integrated CDM $CDM_O = CDM(P_O, E_O, R_O, G_O)$.

3.4. Syntactic Similarity Assessment

The names of the concepts in input (automatically generated by our generators) CDMs typically consist of one or a few words, usually without stop words and possibly with special characters such as underscore ('_'). Therefore, tokenization of names and a set-based measure stand out as the candidates for use. Also, to eliminate possible typographical errors, a set-based measure should be reinforced with the character-level measure – thus making it a soft measure.

There are (too) many possible combinations of the token-level and character-level measures and, possibly, some combinations would prove to be a little better than others for this concrete purpose. However, by defining empirically determined *threshold* for the character-level measure, we can achieve acceptable results despite the chosen measures in the concrete implementation of the tool. Also, when a new type of source artifacts is introduced and expected measures are determined for the CDM derived from those source artifacts, again, by (if necessary) modified threshold, chosen (matching) measures should also run smoothly in that case.

We tried some combinations of measures, and, at the moment, decided to go with the combination of Jaccard similarity as the token-level measure, and Levenshtein distance as the character-level measure. Therefore, further in the article, a syntactic similarity between two strings A and B, denoted $synt\_s(A, B)$, is calculated as follows.

First, we tokenize the strings on special characters and uppercase letters (e.g. $tokens(BookEdition) = \{book, edition\}$), and we compare two sets of tokens $tokens(A)$ and $tokens(B)$ to determine the intersection. To determine the intersection of two sets of tokens, we use Levenshtein distance – if the value of Levenshtein distance between tokens $t_a \in tokens(A)$ and $t_b \in tokens(B)$ is less than defined *threshold* (LD), then tokens t_a and t_b are added to intersection (as a single element) of token sets. Such intersection of token sets is denoted $tokens(A) \cap_{LD} tokens(B)$.

Definition 8: Considering that Jaccard similarity [45] of two sets P and Q (in general) is calculated as a ratio between the number of elements in the intersection of the sets $(|P \cap Q|)$ and the number of elements in the union of the sets $(|P \cup Q|)$, i.e.

$$Jaccard(P, Q) = \frac{|P \cap Q|}{|P \cup Q|} = \frac{|P \cap Q|}{|P| + |Q| - |P \cap Q|}\,, \tag{5}$$

we calculate the syntactic similarity of two strings A and B as follows:

$$synt\_s(A, B) = \frac{|tokens(A) \cap_{LD} tokens(B)|}{|tokens(A)| + |tokens(B)| - |tokens(A) \cap_{LD} tokens(B)|}\,. \tag{6}$$

Note that when determining the number of elements in the union of token sets, we only use Levenshtein distance when computing the intersection of token sets.

4. Approach Overview

Our task is to integrate multiple (unreliable) CDMs into a single unified CDM. The schema integration process consists of two core parts: *schema matching* and *schema merging*. Schema matching is the process of discovering mappings (correspondences) between different schemas, while schema merging is the process of unification of different schemas based on the discovered mappings [36]. Therefore, the schema matching process precedes the schema merging process. Figure 2 shows an overview of the schema integration process. The input in the schema integration process are (unreliable) CDMs automatically generated by tools for automatic derivation of CDMs from specific source artifacts (e.g. CDM derived from a source set of BPMs by AMADEOS, CDM derived from a textual specification by TexToData).

Based on the type of CDM concepts, we divided the schema matching process into four steps: (i) *classes matching step*, (ii) *attributes matching step*, (iii) *associations matching step*, and (iv) *generalizations matching step*. The top elements of the CDM are classes, so the first step in the schema matching process is the classes matching step. Other matching steps (attributes matching step, associations matching step, and generalizations matching step) are directly dependent on the result of the classes matching step. Therefore, the classes matching step must precede other steps in the schema matching process. However, the attributes matching step, associations matching step, and generalizations matching step are not mutually dependent, therefore these steps can be executed in an arbitrary order.

The schema merging process is, also, divided into four steps: (i) *classes merging step*, (ii) *attributes merging step*, (iii) *associations merging step*, and (iv) *generalizations merging step*. The classes merging step is dependent only on the classes matching step, while other merging steps are dependent on the classes merging step and corresponding matching step. Therefore, the classes merging step must precede other merging steps. Other merging steps are not mutually dependent, therefore these steps can be executed in an arbitrary order.

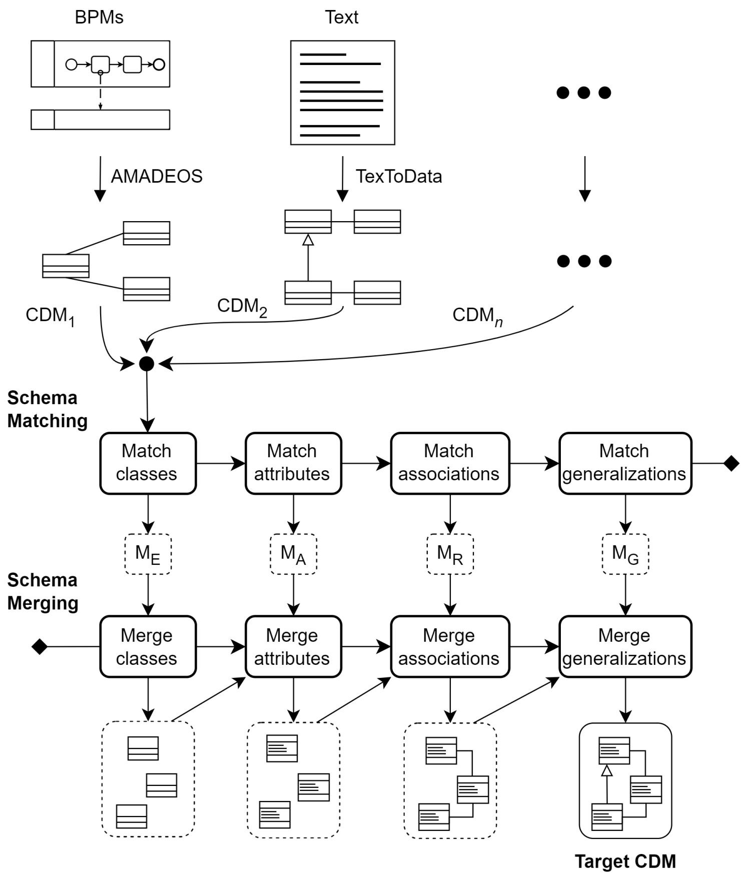

Fig. 2. Overview of the schema integration process

The schema matching and schema merging processes are described in detail in the following two sections.

Note that the default input in the algorithms is I (equation (4)), i.e. a set of n ordered pairs (CDM_i, F_i), so we purposely omitted the explicit definition of such input (in the *require* section) and explicitly stated only necessary inputs that are produced as intermediate results of other algorithms.

5. Schema Matching

The schema matching process in the task of automated schema integration is very important because the result of the schema merging process (i.e. the final result of the schema integration task) directly depends on the result of the schema matching process. Inadequate schema matching may lead to a lower value of the effectiveness of the integrated CDM than the value of the effectiveness of each input CDM (CDMs that are being integrated).

As already stated, the cornerstone of our approach are schema-based matching techniques, which consider only schema information. In our case, schema information includes names, properties (e.g. the properties of classes are attributes and connections to other classes such as associations and generalizations), and constraints (e.g. cardinalities and participation constraints for associations). The usage of structure-based techniques depends directly on the reliability of generated CDM concepts which is, in our approach, directly expressed with the effectiveness measures.

Due to the unreliability of the input CDMs, the usage of structure-based techniques is limited. Therefore, we need to use the names of the concepts and rely on linguistic matching. In cases where we cannot more rely on the structure and constraints, we have to mostly rely on the names. On the other hand, if the reliability of the structure is good, then we can combine linguistic matching and structure-based matching and/or constraint-based matching.

In our approach, we expect that all input CDMs in one execution are in the same language, but in different executions, the input CDMs can be in different languages. This brings us to the fact that our approach is not a single-language approach (focusing on only one language). Due to the multilingual nature of the input CDMs (in different executions), we are focused on the syntactic similarity measure for linguistic matching. The semantic analysis constitutes a great challenge and it is not considered in this article.

The aforementioned combination of different matching techniques classifies our approach in the hybrid matching category. Concerning *matching cardinality*, we produce 1:1:…:1 mapping between schema elements, which means that, for example, a class from CDM_1 can be matched with at most one class from CDM_2 and at most one class from CDM_3 and, so on, at most one class from CDM_n.

5.1. Classes Matching

In the classes matching step, it is necessary to discover class mappings based on the sets of classes from input CDMs $(E_1, \ldots, E_n)$. The result of the classes matching step is a *list*[1] of sets of matched classes from different CDMs, $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$, where:

- m is the number of sets of matched classes (each element $M_E^{(i)} \in M_E$ is the set of classes),
- each set $M_E^{(i)}$ must be a non-empty set,
- each class from each input CDM ($\forall e \in E_i, i = 1, \ldots, n$) belongs to exactly one set $M_E^{(j)}$, and
- not a single set $M_E^{(i)}$ contains multiple classes from the same CDM.

The minimal number of elements in any set $M_E^{(i)}$ is 1, which means that the corresponding class $e \in M_E^{(i)}$ is not matched with any other class from any other CDM. The maximal number of elements in any set $M_E^{(i)}$ is equal to the number of input CDMs (n).

[1] We use the term *list* (according to [3]) to denote an ordered collection of elements (the elements of the list will usually be sets, but in general may be other entities) separated by commas and surrounded by square brackets (e.g. $X = [X^{(1)}, X^{(2)}, X^{(3)}]$ is the list of length 3).

The classes matching step is executed in two phases. In the first phase (Algorithm 1), we compare every pair of classes (e_a, e_b) from different CDMs ($e_a \in E_i$, $e_b \in E_j$, $i \neq j$), which results in the set of triplets $(e_a, e_b, sim_E(e_a, e_b))$. The result of the comparison of two classes e_a and e_b, is the *similarity coefficient*, denoted $sim_E(e_a, e_b) \in [0, 1]$. A higher value of the similarity coefficient means that the corresponding classes are more likely to be the same class, while the lower value means that the corresponding classes are less likely to be the same class. If the similarity coefficient is greater than or equal to the defined threshold (th_{S_E}), then the triplet is accepted and added to the output set Y_E.

The value of the similarity coefficient for classes from different CDMs, $e_a \in E_i$, $e_b \in E_j$, $i \neq j$, is determined from classes' names and attributes, i.e.

$$\begin{aligned} sim_E(e_a, e_b) = {} & synt\_s(name(e_a), name(e_b)) \cdot (1 - attr\_w) \\ & + attr\_s(attrs(e_a), attrs(e_b)) \cdot attr\_w \,, \end{aligned} \tag{7}$$

where $attr\_w = min(F_{A_i}, F_{A_j}) \cdot W_A$ represents the attributes' similarity weight[2] (which depends on the effectiveness measure for attributes of the corresponding CDMs and empirically determined coefficient $W_A \in [0, 1]$), and $attr\_s(attrs(e_a), attrs(e_b)) \in [0, 1]$ is the similarity measure for the sets of attributes of the corresponding classes. The similarity measure for two sets of attributes, $A = attrs(e_a)$ and $B = attrs(e_b)$, similar to the calculation of two sets of string tokens, is calculated using Jaccard similarity, i.e.

$$attr\_s(A, B) = \frac{|A \cap_{sim_A} B|}{|A| + |B| - |A \cap_{sim_A} B|} \,. \tag{8}$$

where the intersection of two sets of attributes is determined using the similarity measure for comparison of two attributes. The calculation of the similarity of two attributes is described in the next subsection. The combination of linguistic matching (names) and structure-based matching (attributes) makes classes matching a hybrid technique.

Algorithm 1 The first phase of the classes matching step

```
Ensure: $Y_E$
 1: $Y_E \leftarrow \emptyset$
 2: for all $E_i : i = 1, \ldots, n-1$ do
 3:     for all $e_a \in E_i$ do
 4:         for all $E_j : j = i+1, \ldots, n$ do
 5:             for all $e_b \in E_j$ do
 6:                 $sim_{ab} \leftarrow sim_E(e_a, e_b)$
 7:                 if $sim_{ab} \geq th_{S_E}$ then
 8:                     $Y_E \leftarrow Y_E \cup \{(e_a, e_b, sim_{ab})\}$
 9:                 end if
10:             end for
11:         end for
12:     end for
13: end for
```

[2] Values $attr\_w$ and $(1 - attr\_w)$ allow us to normalize the $sim_E(e_a, e_b)$ value, as well as to favor more reliable similarity measure ($synt\_s(name(e_a), name(e_b))$ or $attr\_s(attrs(e_a), attrs(e_b))$).

Based on the results from the first phase (i.e. the Y_E set), in the second phase (Algorithm 2), we try to produce sets of best matching classes from different CDMs. In the beginning, the result set M_E is empty. We also create a helper set T_E containing classes that have not yet been matched. In the beginning, this set contains all classes from all input CDMs. Later, when we match some class, then we remove it from this set. Thus, this set enables easy check (whenever necessary) whether some class is already matched or not.

Until the input set does not become an empty set, steps are repeated as follows. First, we find the triplet $(e_a, e_b, sim_E(e_a, e_b))$ in the input set with the highest value of similarity, i.e. the triplet with the most compatible pair of classes. Such triplet is then removed from the input set (in the next iteration of the *while* loop, the next triplet should be considered). The next step is to check if class e_a or class e_b are already matched with

Algorithm 2 The second phase of the classes matching step

```
Require: $Y_E$
Ensure: $M_E = [M_E^{(1)}, \dots, M_E^{(m)}]$
 1: $M_E \leftarrow \emptyset$ ;   $T_E \leftarrow \cup_{i=1}^{n} E_i$
 2: while $Y_E \neq \emptyset$ do
 3:     $(e_a, e_b, sim_E(e_a, e_b)) \leftarrow max_{sim}(Y_E)$
 4:     $Y_E \leftarrow Y_E \setminus \{(e_a, e_b, sim_E(e_a, e_b))\}$
 5:     if $e_a \in T_E \wedge e_b \in T_E$ then
 6:         $me \leftarrow \{e_a, e_b\}$ ;   $M_E \leftarrow M_E \cup \{me\}$ ;   $T_E \leftarrow T_E \setminus me$
 7:     else if $e_a \in T_E$ then
 8:         $me \leftarrow me_t : me_t \in M_E \wedge e_b \in me_t$ ;   $E_t \leftarrow E_i : e_a \in E_i$
 9:         if $me \cap E_t = \emptyset$ then
10:             $me \leftarrow me \cup \{e_a\}$ ;   $T_E \leftarrow T_E \setminus \{e_a\}$
11:         end if
12:     else if $e_b \in T_E$ then
13:         $me \leftarrow me_t : me_t \in M_E \wedge e_a \in me_t$ ;   $E_t \leftarrow E_i : e_b \in E_i$
14:         if $me \cap E_t = \emptyset$ then
15:             $me \leftarrow me \cup \{e_b\}$ ;   $T_E \leftarrow T_E \setminus \{e_b\}$
16:         end if
17:     else
18:         $me_a \leftarrow me_t : me_t \in M_E \wedge e_a \in me_t$ ;   $me_b \leftarrow me_t : me_t \in M_E \wedge e_b \in me_t$
19:         $found \leftarrow false$
20:         for all $E_i : i = 1, \dots, n$ do
21:             if $E_i \cap me_a \neq \emptyset \wedge E_i \cap me_b \neq \emptyset$ then
22:                 $found \leftarrow true$
23:             end if
24:         end for
25:         if $\neg found$ then
26:             $me_a \leftarrow me_a \cup me_b$ ;   $M_E \leftarrow M_E \setminus \{me_b\}$
27:         end if
28:     end if
29: end while
30: for all $e \in T_E$ do
31:     $me \leftarrow \{e\}$ ;   $M_E \leftarrow M_E \cup \{me\}$
32: end for
```

some other classes. If classes e_a and e_b are not already matched with some other classes ($e_a \in T_E \wedge e_b \in T_E$), then classes e_a and e_b are the elements of the new set that needs to be added to the output set. If only class e_a is already matched with some other classes, then it is necessary to add class e_b to that set of classes that are matched with class e_a (if that does not break the rule that set must not contain multiple classes from same input CDM). Analogously, if class e_b is already matched with some other classes, then it is necessary to add class e_a to that set of classes that are matched with class e_b (if that does not break the rule that set must not contain multiple classes from same input CDM). Otherwise, if both classes are already matched, but not together (they are in different sets), then those sets can be joined (if that does not break the rule that set must not contain multiple classes from the same input CDM). Finally, all unmatched classes from all CDMs are added to separate (single element) new sets. In the following steps, the order of the elements in the result is important so the final result (M_E) can be considered to be the list of sets.

Example. To illustrate the classes matching step, let us consider Fig. 3 which shows an illustrative example of three input CDMs ($n = 3$): $CDM_1 = CDM(P_1, E_1, R_1, G_1)$, $CDM_2 = CDM(P_2, E_2, R_2, G_2)$, and $CDM_3 = CDM(P_3, E_3, R_3, G_3)$, where:

$P_1 = \{Text\}, E_1 = \{A1, B1\}, R_1 = \{Z1\}, G_1 = \emptyset\,,$
$P_2 = \{Text\}, E_2 = \{A2, B2, C2\}, R_2 = \{Z2, Y2, X2\}, G_2 = \emptyset\,,$
$P_3 = \{Text\}, E_3 = \{A3, B3, D3\}, R_3 = \{Z3\}, G_3 = \emptyset\,.$

In the first phase of the classes matching step, we compare classes from different CDMs, i.e. $A1$ and $A2$, $A1$ and $B2$, $A1$ and $C2$, $A1$ and $A3$, etc. Let us assume that the result of the comparison, i.e. the result of the first phase is the following set of triplets (for simplicity, the set is shown in a supposed descending order by the value of the similarity coefficient):

$$\begin{aligned}Y_E = \{&(A1, A2, sim_E(A1, A2)), (A1, A3, sim_E(A1, A3)), (A2, A3, sim_E(A2, A3)),\\ &(B1, B2, sim_E(B1, B2)), (B1, B3, sim_E(B1, B3)), (B2, B3, sim_E(B2, B3))\\ &(A1, B2, sim_E(A1, B2))\}\,.\end{aligned}$$

Note that the Y_E set contains only combinations where similarity is greater than or equal to the defined threshold.

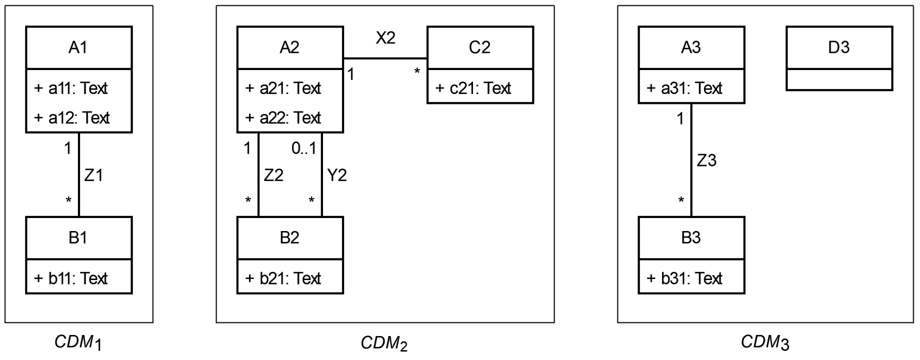

Fig. 3. Illustrative generic example of three input CDMs

The second phase of the classes matching step (i.e. the population of the M_E set, based on the Y_E set) is illustrated in the following lines.

$$\begin{aligned}M_E &\Rightarrow \{\emptyset\} \Rightarrow \{\{A1, A2\}\} \Rightarrow \{\{A1, A2, A3\}\} \Rightarrow \{\{A1, A2, A3\}, \{B1, B2\}\}\\ &\Rightarrow \{\{A1, A2, A3\}, \{B1, B2, B3\}\} \Rightarrow \{\{A1, A2, A3\}, \{B1, B2, B3\}, \{C2\}\}\\ &\Rightarrow \{\{A1, A2, A3\}, \{B1, B2, B3\}, \{C2\}, \{D3\}\}\\ &\Rightarrow [\{A1, A2, A3\}, \{B1, B2, B3\}, \{C2\}, \{D3\}]\,.\end{aligned}$$

At the beginning, classes $A1$ and $A2$ form a new set of matched classes, and then the $A3$ class is added to the same set (containing the $A1$ class). The $(A2, A3)$ pair is ignored, because both classes are already matched (together). The $\{B1, B2, B3\}$ set is formed in a similar manner. The $(A1, B2)$ pair is ignored because joining the sets containing these classes would result in a set that contains multiple classes from a single input CDM. At the end, sets $\{C2\}$ and $\{D3\}$ are formed from the unmatched classes.

5.2. Attributes Matching

In the attributes matching step, it is necessary to match attributes of already matched classes. The input in this step is the output of the classes matching step, i.e. the list of sets of matched classes from different CDMs ($M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$).

The algorithm for attributes matching is similar to the classes matching step – first, it is necessary to compare attributes of matched classes, and then create sets of matched attributes. The output of the attributes matching step is the list of sets of sets of matched attributes $M_A = [M_A^{(1)}, \ldots, M_A^{(m)}]$, where each set $M_A^{(i)}$ is the set of sets of matched attributes of classes from the $M_E^{(i)}$ set.

The input in the first part of the algorithm (Algorithm 3) is the M_E list of sets of matched classes from different CDMs, while the output is the Y_A list of sets of triplets $(at_a, at_b, sim_A(at_a, at_b))$. Each set $Y_A^{(i)} \in Y$ corresponds to the set of matched classes $M_E^{(i)} \in M_E$. Also, each set $Y_A^{(i)}$ is empty in the beginning. Since it is necessary to match attributes only for matched classes, then the following steps are repeated for each input set of matched classes $M_E^{(i)}$. Each attribute ($at_a \in attrs(e_j)$) of each class ($e_j \in M_E^{(i)}$) is compared with every attribute ($at_b \in attrs(e_k)$) of every other class from the same set of matched classes ($e_k \in M_E^{(i)}$). The result of the comparison of two attributes is the similarity coefficient $sim_A(at_a, at_b)$. Attributes at_a and at_b, and the similarity coefficient $sim_A(at_a, at_b)$ form a triplet $(at_a, at_b, sim_A(at_a, at_b))$ which is added to the output set $Y_A^{(i)}$ if the value of the similarity coefficient is greater than or equal to the defined threshold (th_{S_A}).

The value of the similarity coefficient for two attributes (of classes from different CDMs) is determined based on attributes' names, i.e.

$$sim_A(at_a, at_b) = synt\_s(name(at_a), name(at_b))\,. \tag{9}$$

Here we completely rely on the attribute names (linguistic matching) as they are, currently, the most useful information about attributes. If proved to be necessary, it will be easy to include attribute types in the equation.

Algorithm 3 The first phase of the attributes matching step

```
Require: $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$
Ensure: $Y_A = [Y_A^{(1)}, \ldots, Y_A^{(m)}]$
 1: for all $M_E^{(i)} \in M_E$ do
 2:     $Y_A^{(i)} \leftarrow \emptyset$ ;    $n_c \leftarrow |M_E^{(i)}|$
 3:     for all $e_j \in M_E^{(i)} : j = 1, \ldots, n_c - 1$ do
 4:         for all $at_a \in attrs(e_j)$ do
 5:             for all $e_k \in M_E^{(i)} : k = j+1, \ldots, n_c$ do
 6:                 for all $at_b \in attrs(e_k)$ do
 7:                     $sim_{ab} \leftarrow sim_A(at_a, at_b)$
 8:                     if $sim_{ab} \geq th_{S_A}$ then
 9:                         $Y_A^{(i)} \leftarrow Y_A^{(i)} \cup \{(at_a, at_b, sim_{ab})\}$
10:                     end if
11:                 end for
12:             end for
13:         end for
14:     end for
15: end for
```

In the second phase (Algorithm 4) of the attributes matching step, it is necessary to create sets of matched attributes. The input is the list of sets of matched classes and the list of sets of similarity triplets. The output of the attributes matching step is the list of sets of sets of matched attributes, where each set $M_A^{(i)} \in M_A$ is the set of sets of matched attributes that correspond to the set of matched classes $M_E^{(i)} \in M_E$.

In the beginning, each output set $M_A^{(i)}$ is initialized to an empty set, and the T_A set contains attributes from all classes from the $M_E^{(i)}$ set. When an attribute is matched, we remove it from the T_A set (which makes it easy to check whether some attribute is already matched or not). For each input set $Y_A^{(i)} \in Y_A$ of the similarity triplets it is necessary to create sets of matched attributes as follows. In each iteration of the inner *while* loop, we find the $(at_a, at_b, sim_A(at_a, at_b))$ triplet with the highest value of similarity. That triplet is removed from the input set (in the next iteration, another triplet should be considered). Then, it is necessary to check if any of the attributes, at_a or at_b, has already been matched with other attributes. If neither of the current attributes is already matched with other attributes ($at_a \in T_A \wedge at_b \in T_A$), then the given attributes at_a and at_b form a new set of matched attributes (ma) that is added to the set of sets of matched attributes. If one of the attributes is already matched with other attributes, i.e. if there exists set $ma_t \in M_A^{(i)}$ such that $at_a \in ma_t$ or $at_b \in ma_t$, then it is necessary to add other attribute to the same set (if this does not break the rule that the set must not contain multiple attributes from the same class). Otherwise, if both attributes are already matched, but not together (they are in different sets), then those sets can be joined (if this does not break the rule that the set must not contain multiple attributes from the same class). Finally, all unmatched attributes are added to separate (single element) new sets – unmatched attributes of classes from the $M_E^{(i)}$ set are added to separate sets of the output set $M_A^{(i)}$.

Algorithm 4 The second phase of the attributes matching step

```
Require: $M_E = [M_E^{(1)}, \dots, M_E^{(m)}]$   $Y_A = [Y_A^{(1)}, \dots, Y_A^{(m)}]$
Ensure: $M_A = [M_A^{(1)}, \dots, M_A^{(m)}]$
 1: for all $Y_A^{(i)} \in Y_A$ do
 2:     $M_A^{(i)} \leftarrow \emptyset$ ;   $T_A \leftarrow \cup_{\forall e \in M_E^{(i)}} attrs(e)$
 3:     while $Y_A^{(i)} \neq \emptyset$ do
 4:         $(at_a, at_b, sim_A(at_a, at_b)) \leftarrow max_{sim}(Y_A^{(i)})$
 5:         $Y_A^{(i)} \leftarrow Y_A^{(i)} \setminus \{(at_a, at_b, sim_A(at_a, at_b))\}$
 6:         if $at_a \in T_A \wedge at_b \in T_A$ then
 7:             $ma \leftarrow \{at_a, at_b\}$ ;   $M_A^{(i)} \leftarrow M_A^{(i)} \cup \{ma\}$ ;   $T_A \leftarrow T_A \setminus ma$
 8:         else if $at_a \in T_A$ then
 9:             $ma \leftarrow ma_t : ma_t \in M_A^{(i)} \wedge at_b \in ma_t$ ;   $e \leftarrow e_t : e_t \in M_E^{(i)} \wedge at_a \in attrs(e_t)$
10:             if $ma \cap attrs(e) = \emptyset$ then
11:                 $ma \leftarrow ma \cup \{at_a\}$ ;   $T_A \leftarrow T_A \setminus \{at_a\}$
12:             end if
13:         else if $at_b \in T_A$ then
14:             $ma \leftarrow ma_t : ma_t \in M_A^{(i)} \wedge at_a \in ma_t$
15:             $e \leftarrow e_t : e_t \in M_E^{(i)} \wedge at_b \in attrs(e_t)$
16:             if $ma \cap attrs(e) = \emptyset$ then
17:                 $ma \leftarrow ma \cup \{at_b\}$ ;   $T_A \leftarrow T_A \setminus \{at_b\}$
18:             end if
19:         else
20:             $ma_a \leftarrow ma_t : ma_t \in M_A^{(i)} \wedge at_a \in ma_t$
21:             $ma_b \leftarrow ma_t : ma_t \in M_A^{(i)} \wedge at_b \in ma_t$
22:             $found \leftarrow false$
23:             for all $e \in M_E^{(i)}$ do
24:                 if $attrs(e) \cap ma_a \neq \emptyset \wedge attrs(e) \cap ma_b \neq \emptyset$ then
25:                     $found \leftarrow true$
26:                 end if
27:             end for
28:             if $\neg found$ then
29:                 $ma_a \leftarrow ma_a \cup ma_b$ ;   $M_A^{(i)} \leftarrow M_A^{(i)} \setminus \{ma_b\}$
30:             end if
31:         end if
32:     end while
33:     for all $at \in T_A$ do
34:         $ma \leftarrow \{at\}$ ;   $M_A^{(i)} \leftarrow M_A^{(i)} \cup \{ma\}$
35:     end for
36: end for
```

Example. Now we will illustrate the attributes matching step using the example from the previous subsection. Here we repeat the output, i.e. the M_E list of sets of matched classes:

$$M_E = [M_E^{(1)}, M_E^{(2)}, M_E^{(3)}, M_E^{(4)}]\,,$$

where

$$M_E^{(1)} = \{A1, A2, A3\},\ M_E^{(2)} = \{B1, B2, B3\},\ M_E^{(3)} = \{C2\},\ M_E^{(4)} = \{D3\}.$$

In the first phase, we need to create the output list Y_A of sets of similarity triplets for attributes. The number of elements in the output list corresponds to the number of elements in the list M_E ($m = 4$), and element/set $Y_A^{(i)}$ corresponds to the element/set $M_E^{(i)}$. Note that set $Y_A^{(i)}$ contains similarity triplets for attributes of only those classes that are in the $M_E^{(i)}$ set. Finally, let us assume the following output of the first phase (elements in the sets are shown in a supposed descending order by the value of the similarity coefficient):

$$\begin{aligned} Y_A =& \{Y_A^{(1)}, Y_A^{(2)}, Y_A^{(3)}, Y_A^{(4)}\}, \\ Y_A^{(1)} =& \{(a11, a21, sim_A(a11, a21)), (a21, a31, sim_A(a21, a31)), \\ & (a12, a22, sim_A(a12, a22))\}, \\ Y_A^{(2)} =& \{(b11, b21, sim_A(b11, b21)), (b11, b31, sim_A(b11, b31)), \\ & (b21, b31, sim_A(b21, b31))\}, \\ Y_A^{(3)} =& Y_A^{(4)} = \{\emptyset\}\,. \end{aligned}$$

In the second phase of the algorithm, we create the output list of sets of sets of matched attributes $M_A = [M_A^{(1)}, M_A^{(2)}, M_A^{(3)}, M_A^{(4)}]$, where each $M_A^{(i)}$ is created from the set of similarity triplets $Y_A^{(i)}$. The following lines describe the population of each set $M_A^{(i)}$.

$$\begin{aligned} & M_A^{(1)} \Rightarrow \{\emptyset\} \Rightarrow \{\{a11, a21\}\} \Rightarrow \{\{a11, a21, a31\}\} \Rightarrow \{\{a11, a21, a31\}, \{a12, a22\}\} \\ & M_A^{(2)} \Rightarrow \{\emptyset\} \Rightarrow \{\{b11, b21\}\} \Rightarrow \{\{b11, b21, b31\}\} \\ & M_A^{(3)} \Rightarrow \{\emptyset\} \Rightarrow \{\{c21\}\} \\ & M_A^{(4)} \Rightarrow \{\emptyset\}\,. \end{aligned}$$

The $M_A^{(3)}$ set is empty in the beginning. Since $Y_A^{(3)}$ is empty, only a set containing (unmatched) attribute $c21$ is added to $M_A^{(3)}$. The $M_A^{(4)}$ set remains empty because $Y^{(4)}$ is empty and the only class in $M_E^{(4)}$ ($D3 \in M_E^{(4)}$) has no attributes.

5.3. Associations Matching

The next step in the matching process is the associations matching step. In this step, it is necessary to match associations between different input CDMs with respect to already matched classes. For example, let us consider the result in the classes matching step example, i.e. the list of sets of matched classes

$$M_E = [\{A1, A2, A3\}, \{B1, B2, B3\}, \{C2\}, \{D3\}]\,.$$

In the associations matching step, it is necessary to create the sets of matched associations for all combinations of sets of matched classes ($M_E^{(i)}$ and $M_E^{(j)}$ for $i = 1, \ldots, m$ and $i \leq j \leq m$). If the list of sets of matched classes has m elements, then the list of matched associations has to have $m(m+1)/2$ elements. In this example, since $m = 4$ and $m(m+1)/2 = 10$, it is necessary to create the list with ten elements, i.e.

$$M_R = [M_R^{(1)(1)}, M_R^{(1)(2)}, M_R^{(1)(3)}, M_R^{(1)(4)},$$
$$M_R^{(2)(2)}, M_R^{(2)(3)}, M_R^{(2)(4)}, M_R^{(3)(3)}, M_R^{(3)(4)}, M_R^{(4)(4)}] .$$

The elements of the sets $M_R^{(i)(j)}$, where $i \neq j$, should be the sets of matched associations between the classes from sets $M_E^{(i)}$ and $M_E^{(j)}$. For example, the $M_R^{(1)(2)}$ element should be the set of sets of matched associations between the classes from sets $M_E^{(1)}$ and $M_E^{(2)}$. Since the $M_E^{(1)}$ set contains classes $A1$, $A2$, and $A3$, and the $M_E^{(2)}$ set contains classes $B1$, $B2$, and $B3$, then we must compare the associations between the pair of classes $(A1, B1)$ and the associations between the pair of classes $(A2, B2)$ and the associations between the pair of classes $(A3, B3)$.

The elements of the sets $M_R^{(i)(i)}$ should be sets of matched reflexive associations of the classes from the $M_E^{(i)}$ set. For example, the $M_R^{(1)(1)}$ element should be the set of sets of matched reflexive associations between the classes from set $M_E^{(1)}$. Since the $M_E^{(1)}$ set contains classes $A1$, $A2$, and $A3$, then we must compare reflexive associations of the $A1$ class and reflexive associations of the $A2$ class and reflexive associations of the $A3$ class.

In order to produce the list of sets of sets of matched associations, we split the associations matching step into two phases (similar to previous matching steps). In the first phase (Algorithm 5), we only compare the associations and create a list of sets of triplets $(r_a, r_b, sim_R(r_a, r_b))$, where $sim_R(r_a, r_b)$ is the result of the comparison (similarity coefficient) of the associations r_a and r_b. As mentioned earlier, if the list of sets of matched classes has m elements, the Y_R list of sets of triplets $(r_a, r_b, sim_R(r_a, r_b))$ has $m(m + 1)/2$ elements. In the beginning, each set $Y_R^{(i)(j)}$ is initialized to an empty set, and then it is populated as follows. First, it is necessary to find the pair of classes (e_{k_a}, e_{k_b}) from the *k*-th input CDM such that $e_{k_a} \in M_E^{(i)}$ and $e_{k_b} \in M_E^{(j)}$. Then, it is necessary to find all associations (R'_k) between that pair of classes. The next step is to find the pair of classes (e_{p_a}, e_{p_b}) from the *p*-th input CDM such that $e_{p_a} \in M_E^{(i)}$ and $e_{p_b} \in M_E^{(j)}$. Here is also necessary to find all associations (R'_p) between that pair of classes. Now, we can compare all associations from R'_k with all associations from R'_p. If the value of the similarity coefficient is greater than or equal to the defined threshold (th_{S_R}), then the $(r_a, r_b, sim_R(r_a, r_b))$ triplet is added to the $Y_R^{(i)(j)}$ set. These steps are repeated for every possible combination of k and p, i.e. for each input set of associations. All these steps represent the steps for the population of the output set $Y_R^{(i)(j)}$.

As described above, we compare only associations between already matched classes. Let there be two associations from different CDMs, $r_a \in R_k$ and $r_b \in R_p$. Let r_a be an association between the classes $e_{k_a} \in E_k \wedge e_{k_a} \in M_E^{(i)}$ and $e_{k_b} \in E_k \wedge e_{k_b} \in M_E^{(j)}$, and let r_b be an association between the classes $e_{p_a} \in E_p \wedge e_{p_a} \in M_E^{(i)}$ and $e_{p_b} \in E_p \wedge e_{p_b} \in M_E^{(j)}$. This means that one end of the first association is a class matched with a class that is one end of the second association (both classes are in the $M_E^{(i)}$ set). Also, the other end of the first association is a class matched with a class that is the other end of the second association (both classes are in the $M_E^{(j)}$ set). The value of the similarity coefficient for such two associations (and we only compare associations between the pairs of matched classes) is determined based on associations' names and constraints (cardinalities and participation constraints), i.e.

$$\begin{aligned} sim_R(r_a, r_b) = {} & synt\_s(name(r_a), name(r_b)) \cdot (1 - W_{card} - W_{pc}) \\ & + ceq(r_a, r_b) \cdot W_{card} + peq(r_a, r_b) \cdot W_{pc}\,, \end{aligned} \tag{10}$$

where $ceq(r_a, r_b) \in \{0, 1\}$ checks whether the cardinalities of associations are equal, i.e.

$$ceq(r_a, r_b) = \begin{cases} 1, & (type(source(r_a)) \in M_E^{(i)} \wedge type(source(r_b)) \in M_E^{(i)} \\ & \vee type(source(r_a)) \in M_E^{(j)} \wedge type(source(r_b)) \in M_E^{(j)}) \\ & \wedge upper(source(r_a)) = upper(source(r_b)) \\ & \wedge upper(target(r_a)) = upper(target(r_b)) \\ & \vee \\ & (type(source(r_a)) \in M_E^{(i)} \wedge type(source(r_b)) \in M_E^{(j)} \\ & \vee type(source(r_a)) \in M_E^{(j)} \wedge type(source(r_b)) \in M_E^{(i)}) \\ & \wedge upper(source(r_a)) = upper(target(r_b)) \\ & \wedge upper(target(r_a)) = upper(source(r_b)) \\ 0, & \text{otherwise} \end{cases}, \tag{11}$$

Algorithm 5 The first phase of the associations matching step

```
Require: $M_E = [M_E^{(1)}, \dots, M_E^{(m)}]$
Ensure: $Y_R = [Y_R^{(1)(1)}, \dots, Y_R^{(1)(m)}, Y_R^{(2)(2)}, \dots, Y_R^{(2)(m)}, \dots Y_R^{(m)(m)}]$
 1: for $i = 1, \dots, m$ do
 2:     for $j = i, \dots, m$ do
 3:         $Y_R^{(i)(j)} \leftarrow \emptyset$
 4:         for $k = 1, \dots, n-1$ do
 5:             $e_{k_a} \leftarrow e : e \in E_k \wedge e \in M_E^{(i)}$ ;   $e_{k_b} \leftarrow e : e \in E_k \wedge e \in M_E^{(j)}$
 6:             $R'_k \leftarrow \{r : r \in R_k \wedge (source(r) = e_{k_a} \wedge target(r) = e_{k_b} \vee$
 7:                             $source(r) = e_{k_b} \wedge target(r) = e_{k_a})\}$
 8:             for all $r_a \in R'_k$ do
 9:                 for $p = k+1, \dots, n$ do
10:                     $e_{p_a} \leftarrow e : e \in E_p \wedge e \in M_E^{(i)}$ ;   $e_{p_b} \leftarrow e : e \in E_p \wedge e \in M_E^{(j)}$
11:                     $R'_p \leftarrow \{r : r \in R_p \wedge (source(r) = e_{p_a} \wedge target(r) = e_{p_b} \vee$
12:                                     $source(r) = e_{p_b} \wedge target(r) = e_{p_a})\}$
13:                     for all $r_b \in R'_p$ do
14:                         $sim_{ab} \leftarrow sim_R(r_a, r_b)$
15:                         if $sim_{ab} \geq th_{S_R}$ then
16:                             $Y_R^{(i)(j)} \leftarrow Y_R^{(i)(j)} \cup \{(r_a, r_b, sim_{ab})\}$
17:                         end if
18:                     end for
19:                 end for
20:             end for
21:         end for
22:     end for
23: end for
```

$peq(r_a, r_b) \in \{0, 1\}$ checks whether the participation constraints are equal, i.e.

$$peq(r_a, r_b) = \begin{cases} 1, & (type(source(r_a)) \in M_E^{(i)} \wedge type(source(r_b)) \in M_E^{(i)} \\ & \vee type(source(r_a)) \in M_E^{(j)} \wedge type(source(r_b)) \in M_E^{(j)}) \\ & \wedge lower(source(r_a)) = lower(source(r_b)) \\ & \wedge lower(target(r_a)) = lower(target(r_b)) \\ & \vee \\ & (type(source(r_a)) \in M_E^{(i)} \wedge type(source(r_b)) \in M_E^{(j)} \\ & \vee type(source(r_a)) \in M_E^{(j)} \wedge type(source(r_b)) \in M_E^{(i)}) \\ & \wedge lower(source(r_a)) = lower(target(r_b)) \\ & \wedge lower(target(r_a)) = lower(source(r_b)) \\ 0, & \text{otherwise} \end{cases}, \quad (12)$$

and W_{card} and W_{pc} are the empirically determined coefficients ($0 \leq W_{card} + W_{pc} \leq 1$).

The combination of linguistic matching (names) and constraint-based matching (cardinalities and participation constraints) makes the associations matching, also, a hybrid technique.

In the second phase (Algorithm 6), it is necessary to produce the list of sets of sets of matched associations based on the results from the first phase. The output list M_R has the same number of elements as the input list Y_R, i.e. $m(m+1)/2$ (this corresponds to every combination of sets of the list M_E). The T_R set is a utility set of unmatched associations between the corresponding pairs of classes from the sets $M_E^{(i)}$ and $M_E^{(j)}$ (when an association is matched it is then removed from this set). Each element $M_R^{(i)(j)}$ of the output list M_R is created from the corresponding set $Y_R^{(i)(j)}$ of triplets $(r_a, r_b, sim_R(r_a, r_b))$ as follows. First, it is necessary to find the triplet $(r_a, r_b, sim_R(r_a, r_b))$ in the $Y_R^{(i)(j)}$ set with the highest value of similarity. The found triplet is removed from the $Y_R^{(i)(j)}$ set (in the next iteration of the *while* loop, the next triplet must be considered). Now, it is necessary to check if any of the associations, r_a or r_b, has already been matched with other associations. If neither of the current associations is already matched with other associations ($r_a \in T_R \wedge r_b \in T_R$), then the given associations r_a and r_b form a new set of matched associations that is added to the $M_R^{(i)(j)}$ set. If one of the associations is already matched with other associations, i.e. if there exists a set $mr_t \in M_R^{(i)(j)}$ such that $r_a \in mr_t$ or $r_b \in mr_t$, then it is necessary to add other association to the same set (if this does not break the rule that the set must not contain multiple associations from the same input CDM). Otherwise, if both associations are already matched, but not together (they are in different sets), then those sets can be joined (if this does not break the rule that the set must not contain multiple associations from the same input CDM). Finally, all unmatched associations are added to separate (single element) new sets – unmatched associations from each input CDM between the corresponding classes in sets $M_E^{(i)}$ and $M_E^{(j)}$ are added to separate new sets of the output set $M_R^{(i)(j)}$.

Algorithm 6 The second phase of the associations matching step

```
Require: $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$   $Y_R = [Y_R^{(1)(1)}, \ldots, Y_R^{(1)(m)}, Y_R^{(2)(2)}, \ldots, Y_R^{(m)(m)}]$
Ensure: $M_R = [M_R^{(1)(1)}, \ldots, M_R^{(1)(m)}, M_R^{(2)(2)}, \ldots, M_R^{(2)(m)}, \ldots, M_R^{(m)(m)}]$
 1: for all $Y_R^{(i)(j)} \in Y_R$ do
 2:     $M_R^{(i)(j)} \leftarrow \emptyset$ ;   $T_R \leftarrow \emptyset$
 3:     for $k = 1, \ldots, n$ do
 4:         $e_{k_a} \leftarrow e : e \in E_k \wedge e \in M_E^{(i)}$ ;   $e_{k_b} \leftarrow e : e \in E_k \wedge e \in M_E^{(j)}$
 5:         $T_R \leftarrow T_R \cup \{r : r \in R_k \wedge (source(r) = e_{k_a} \wedge target(r) = e_{k_b} \vee$
 6:                                 $source(r) = e_{k_b} \wedge target(r) = e_{k_a})\}$
 7:     end for
 8:     while $Y_R^{(i)(j)} \neq \emptyset$ do
 9:         $(r_a, r_b, sim_R(r_a, r_b)) \leftarrow max_{sim}(Y_R^{(i)(j)})$
10:         $Y_R^{(i)(j)} \leftarrow Y_R^{(i)(j)} \setminus \{(r_a, r_b, sim_R(r_a, r_b))\}$
11:         if $r_a \in T_R \wedge r_b \in T_R$ then
12:             $M_R^{(i)(j)} \leftarrow M_R^{(i)(j)} \cup \{\{r_a, r_b\}\}$ ;   $T_R \leftarrow T_R \setminus \{r_a, r_b\}$
13:         else if $r_a \in T_R$ then
14:             $mr \leftarrow mr_t : mr_t \in M_R^{(i)(j)} \wedge r_b \in mr_t$ ;   $R_t \leftarrow R_k : r_a \in R_k$
15:             if $mr \cap R_t = \emptyset$ then
16:                 $mr \leftarrow mr \cup \{r_a\}$ ;   $T_R \leftarrow T_R \setminus \{r_a\}$
17:             end if
18:         else if $r_b \in T_R$ then
19:             $mr \leftarrow mr_t : mr_t \in M_R^{(i)(j)} \wedge r_a \in mr_t$ ;   $R_t \leftarrow R_k : r_b \in R_k$
20:             if $mr \cap R_t = \emptyset$ then
21:                 $mr \leftarrow mr \cup \{r_b\}$ ;   $T_R \leftarrow T_R \setminus \{r_b\}$
22:             end if
23:         else
24:             $mr_a \leftarrow mr_t : mr_t \in M_R^{(i)(j)} \wedge r_a \in mr_t$
25:             $mr_b \leftarrow mr_t : mr_t \in M_R^{(i)(j)} \wedge r_b \in mr_t$
26:             $found \leftarrow false$
27:             for all $R_k : k = 1, \ldots, n$ do
28:                 if $R_k \cap mr_a \neq \emptyset \wedge R_k \cap mr_b \neq \emptyset$ then
29:                     $found \leftarrow true$
30:                 end if
31:             end for
32:             if $\neg found$ then
33:                 $mr_a \leftarrow mr_a \cup mr_b$ ;   $M_R^{(i)(j)} \leftarrow M_R^{(i)(j)} \setminus \{mr_b\}$
34:             end if
35:         end if
36:     end while
37:     for all $r \in T_R$ do
38:         $M_R^{(i)(j)} \leftarrow M_R^{(i)(j)} \cup \{\{r\}\}$
39:     end for
40: end for
```

Example. Continuing the example from previous subsections, here we need to produce the list of sets of sets of matched associations. In order to do that, first we need to compare the associations and create a list of ten ($m = |M_E| = 4 \Rightarrow m(m+1)/2 = 10$) sets of similarity triplets, i.e.

$$\begin{aligned} Y_R = [&Y_R^{(1)(1)}, Y_R^{(1)(2)}, Y_R^{(1)(3)}, Y_R^{(1)(4)}, \\ &Y_R^{(2)(2)}, Y_R^{(2)(3)}, Y_R^{(2)(4)}, Y_R^{(3)(3)}, Y_R^{(3)(4)}, Y_R^{(4)(4)}] \,. \end{aligned}$$

To populate the $Y_R^{(1)(2)}$ set we need to compare the associations between the pairs of classes $(A1, B1)$, $(A2, B2)$, and $(A3, B3)$, i.e. associations $Z1$, $Z2$, $Y2$, and $Z3$. Other sets of similarity triplets will remain empty because there are no reflexive associations and sets $M_E^{(3)}$ and $M_E^{(4)}$ contain only a single class.

Let us assume the following output of the first phase (elements in the sets are shown in a supposed descending order by the value of the similarity coefficient):

$$\begin{aligned} Y_R^{(1)(1)} =& Y_R^{(1)(3)} = Y_R^{(1)(4)} = \{\emptyset\}, \\ Y_R^{(1)(2)} =& \{(Z1, Z2, sim_R(Z1, Z2)), (Z1, Z3, sim_R(Z1, Z3)), \\ & (Z2, Z3, sim_R(Z2, Z3)), (Z1, Y2, sim_R(Z1, Y2))\}, \\ Y_R^{(2)(2)} =& Y_R^{(2)(3)} = Y_R^{(2)(4)} = Y_R^{(3)(3)} = Y_R^{(3)(4)} = Y_R^{(4)(4)} = \{\emptyset\} \,. \end{aligned}$$

In the second phase of the algorithm, we create the output list of sets of sets of matched associations M_R, where each $M_R^{(i)(j)}$ is created from the $Y_R^{(i)(j)}$ set of similarity triplets. The following lines describe the population of each set $M_R^{(i)(j)} \in M_R$.

$$\begin{aligned} & M_R^{(1)(2)} \Rightarrow \{\emptyset\} \Rightarrow \{\{Z1, Z2\}\} \Rightarrow \{\{Z1, Z2, Z3\}\} \Rightarrow \{\{Z1, Z2, Z3\}, \{Y2\}\}, \\ & M_R^{(1)(3)} \Rightarrow \{\emptyset\} \Rightarrow \{\{X2\}\}, \\ & M_R^{(1)(1)}, M_R^{(1)(4)}, M_R^{(2)(2)}, M_R^{(2)(3)}, M_R^{(2)(4)}, M_R^{(3)(3)}, M_R^{(3)(4)}, M_R^{(4)(4)} \Rightarrow \{\emptyset\} \,. \end{aligned}$$

The $M_R^{(1)(3)}$ set is empty in the beginning. Since $Y_R^{(1)(3)}$ is an empty set, only a set containing (unmatched) association $X2$ is added to the $M_R^{(1)(3)}$ set. The sets $M_R^{(1)(1)}$, $M_R^{(1)(4)}$, $M_R^{(2)(2)}$, $M_R^{(2)(3)}$, $M_R^{(2)(4)}$, $M_R^{(3)(3)}$, $M_R^{(3)(4)}$, and $M_R^{(4)(4)}$ remain empty because there are no respective associations in the input CDMs.

5.4. Generalizations Matching

The generalizations matching step is similar to the associations matching step, with the exception that there can be only one generalization between two classes. The generalizations matching step is, also, split into two phases.

In the first phase (Algorithm 7), we create a list of sets of triplets $(g_a, g_b, sim_G(g_a, g_b))$, where $sim_G(g_a, g_b)$ is the result of the comparison (similarity coefficient) of the generalizations g_a and g_b. If the list of sets of matched classes has m elements, then the Y_G list of sets of triplets $(g_a, g_b, sim_G(g_a, g_b))$ has $m(m-1)/2$ elements because it is necessary to create the sets of matched generalizations for all combinations of sets of matched classes ($M_E^{(i)}$ and $M_E^{(j)}$ for $i = 1, \ldots, m-1$ and $i < j \leq m$). The number of elements of the output list for generalizations is less than the number of elements of the output list for associations because we had to take into account reflexive associations.

Algorithm 7 The first phase of the generalizations matching step

```
Require: $M_E = [M_E^{(1)}, \dots, M_E^{(m)}]$
Ensure: $Y_G = [Y_G^{(1)(2)}, \dots, Y_G^{(1)(m)}, Y_G^{(2)(3)}, \dots, Y_G^{(2)(m)}, \dots, Y_G^{(m-1)(m)}]$
 1: for $i = 1, \dots, m-1$ do
 2:     for $j : i+1, \dots, m$ do
 3:         $Y_G^{(i)(j)} \leftarrow \emptyset$
 4:         for $k = 1, \dots, n-1$ do
 5:             $e_{k_a} \leftarrow e : e \in E_k \wedge e \in M_E^{(i)}$ ;   $e_{k_b} \leftarrow e : e \in E_k \wedge e \in M_E^{(j)}$
 6:             $g_a \leftarrow g : g \in G_k \wedge (specific(g) = e_{k_a} \wedge general(g) = e_{k_b} \vee$
 7:                         $specific(g) = e_{k_b} \wedge general(g) = e_{k_a})$
 8:             for $p = k+1, \dots, n$ do
 9:                 $e_{p_a} \leftarrow e : e \in E_p \wedge e \in M_E^{(i)}$ ;   $e_{p_b} \leftarrow e : e \in E_p \wedge e \in M_E^{(j)}$
10:                 $g_b \leftarrow g : g \in G_p \wedge (specific(g) = e_{p_a} \wedge general(g) = e_{p_b} \vee$
11:                             $specific(g) = e_{p_b} \wedge general(g) = e_{p_a})$
12:                 $sim_{ab} \leftarrow sim_G(g_a, g_b)$
13:                 if $sim_{ab} \geq th_{S_G}$ then
14:                     $Y_G^{(i)(j)} \leftarrow Y_G^{(i)(j)} \cup \{(g_a, g_b, sim_{ab})\}$
15:                 end if
16:             end for
17:         end for
18:     end for
19: end for
```

In the beginning, each set $Y_G^{(i)(j)}$ is initialized to an empty set, and then it is populated as follows. First, we need to find a pair of classes (e_{k_a}, e_{k_b}) from CDM_k such that $e_{k_a} \in M_E^{(i)}$ and $e_{k_b} \in M_E^{(j)}$, and then we need to find the generalization (g_a) between that pair of classes. Then, we need to find a pair of classes (e_{p_a}, e_{p_b}) from CDM_p such that $e_{p_a} \in M_E^{(i)}$ and $e_{p_b} \in M_E^{(j)}$, and then we, also, need to find the generalization (g_b) between that pair of classes. Now, we can compare generalization g_a with the generalization g_b, and add the $(g_a, g_b, sim_G(g_a, g_b))$ triplet to the $Y_G^{(i)(j)}$ set if the value of the similarity coefficient is greater than or equal to the defined threshold (th_{S_G}). These steps are repeated for every combination of k and p, i.e. for each input set of generalizations. All these steps represent the steps for the population of the output set $Y_G^{(i)(j)}$.

As described above, we compare only generalizations between already matched classes. Let there be two generalizations from different CDMs, $g_a \in G_k$ and $g_b \in G_p$. Let g_a be a generalization between the classes $e_{k_a} \in E_k \wedge e_{k_a} \in M_E^{(i)}$ and $e_{k_b} \in E_k \wedge e_{k_b} \in M_E^{(j)}$, and let g_b be a generalization between the classes $e_{p_a} \in E_p \wedge e_{p_a} \in M_E^{(i)}$ and $e_{p_b} \in E_p \wedge e_{p_b} \in M_E^{(j)}$. The value of the similarity coefficient for such two generalizations is determined only based on the direction, i.e.

$$sim_G(g_a, g_b) = \begin{cases} 1, & specific(g_a) \in M_E^{(i)} \wedge specific(g_b) \in M_E^{(i)} \\ & \vee \\ & specific(g_a) \in M_E^{(j)} \wedge specific(g_b) \in M_E^{(j)} \\ 0, & \text{otherwise} \end{cases} . \tag{13}$$

In the second phase (Algorithm 8), it is necessary to produce a list of sets of matched generalizations based on the results from the first phase. The output list M_G has the same number of elements as the input list Y_G, i.e. $m(m-1)/2$.

Algorithm 8 The second phase of the generalizations matching step

```
Require: $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$   $Y_G = [Y_G^{(1)(2)}, \ldots, Y_G^{(1)(m)}, Y_G^{(2)(3)}, \ldots, Y_G^{(m-1)(m)}]$
Ensure: $M_G = [M_G^{(1)(2)}, \ldots, M_G^{(1)(m)}, M_G^{(2)(3)}, \ldots, M_G^{(2)(m)}, \ldots, M_G^{(m-1)(m)}]$
 1: for all $Y_G^{(i)(j)} \in Y_G$ do
 2:     $M_G^{(i)(j)} \leftarrow \emptyset$ ;   $T_G \leftarrow \emptyset$
 3:     for $k = 1, \ldots, n$ do
 4:         $e_{k_a} \leftarrow e : e \in E_k \wedge e \in M_E^{(i)}$ ;   $e_{k_b} \leftarrow e : e \in E_k \wedge e \in M_E^{(j)}$
 5:         $T_G \leftarrow T_G \cup \{g : g \in G_k \wedge (specific(g) = e_{k_a} \wedge general(g) = e_{k_b} \vee$
 6:                                     $specific(g) = e_{k_b} \wedge general(g) = e_{k_a})\}$
 7:     end for
 8:     while $Y_G^{(i)(j)} \neq \emptyset$ do
 9:         $(g_a, g_b, sim_G(g_a, g_b)) \leftarrow max_{sim}(Y_G^{(i)(j)})$
10:         $Y_G^{(i)(j)} \leftarrow Y_G^{(i)(j)} \setminus \{(g_a, g_b, sim_G(g_a, g_b))\}$
11:         if $g_a \in T_G \wedge g_b \in T_G$ then
12:             $M_G^{(i)(j)} \leftarrow M_G^{(i)(j)} \cup \{\{g_a, g_b\}\}$ ;   $T_G \leftarrow T_G \setminus \{g_a, g_b\}$
13:         else if $g_a \in T_G$ then
14:             $mg \leftarrow mg_t : mg_t \in M_G^{(i)(j)} \wedge g_b \in mg_t$ ;   $G_t \leftarrow G_k : g_a \in G_k$
15:             if $mg \cap G_t = \emptyset$ then
16:                 $mg \leftarrow mg \cup \{g_a\}$ ;   $T_G \leftarrow T_G \setminus \{g_a\}$
17:             end if
18:         else if $g_b \in T_G$ then
19:             $mg \leftarrow mg_t : mg_t \in M_G^{(i)(j)} \wedge g_a \in mg_t$ ;   $G_t \leftarrow G_k : g_b \in G_k$
20:             if $mg \cap G_t = \emptyset$ then
21:                 $mg \leftarrow mg \cup \{g_b\}$ ;   $T_G \leftarrow T_G \setminus \{g_b\}$
22:             end if
23:         else
24:             $mg_a \leftarrow mg_t : mg_t \in M_G^{(i)(j)} \wedge g_a \in mg_t$
25:             $mg_b \leftarrow mg_t : mg_t \in M_G^{(i)(j)} \wedge g_b \in mg_t$
26:             $found \leftarrow false$
27:             for all $G_k : k = 1, \ldots, n$ do
28:                 if $G_k \cap mg_a \neq \emptyset \wedge G_k \cap mg_b \neq \emptyset$ then
29:                     $found \leftarrow true$
30:                 end if
31:             end for
32:             if $\neg found$ then
33:                 $mg_a \leftarrow mg_a \cup mg_b$ ;   $M_G^{(i)(j)} \leftarrow M_G^{(i)(j)} \setminus \{mg_b\}$
34:             end if
35:         end if
36:     end while
37:     for all $g \in T_G$ do
38:         $M_G^{(i)(j)} \leftarrow M_G^{(i)(j)} \cup \{\{g\}\}$
39:     end for
40: end for
```

Each element $M_G^{(i)(j)}$ of the output list M_G is created from the corresponding set $Y_G^{(i)(j)}$ of triplets $(g_a, g_b, sim_G(g_a, g_b))$ as follows. First, it is necessary to find the $(g_a, g_b, sim_G(g_a, g_b))$ triplet in the $Y_G^{(i)(j)}$ set with the highest value of similarity. The found triplet is removed from the $Y_G^{(i)(j)}$ set (in the next iteration of the *while* loop, the next triplet must be considered). Then, it is necessary to check if any of the generalizations, g_a or g_b, has already been matched with other generalizations. If neither of the current generalizations is already matched with other generalizations ($g_a \in T_G \wedge g_b \in T_G$), then given generalizations g_a and g_b form a new set of matched generalizations that is added to the $M_G^{(i)(j)}$ set. If one of the generalizations is already matched with other generalizations, i.e. if there exists set $mg_t \in M_G^{(i)(j)}$ such that $g_a \in mg_t$ or $g_b \in mg_t$, then it is necessary to add other generalization to the same set (if this does not break the rule that the set must not contain multiple generalizations from the same input CDM). Otherwise, if both generalizations are already matched, but not together (they are in different sets), then those sets can be joined (if this does not break the rule that the set must not contain multiple generalizations from the same input CDM). Finally, all unmatched generalizations are added to separate (single element) new sets – unmatched generalizations from each input CDM between the corresponding classes in sets $M_E^{(i)}$ and $M_E^{(j)}$ are added to separate new sets of the output set $M_G^{(i)(j)}$.

Due to the article length and the similarity of the generalizations matching step to the associations matching step (there can be only one generalization between two classes, while there can be more than one association between two classes) we do not provide an example for the generalizations matching step.

6. Schema Merging

The merging process is the process of creating a new integrated CDM ($CDM_O = CDM(P_O, E_O, R_O, G_O)$) based on the mappings discovered in the matching process.

6.1. Primitive types

The creation of primitive types is the pre-step in the merging process, and it is a rather simple one. The P_O set of primitive types of the integrated CDM is a union of primitive types of the input CDMs, i.e.

$$P_O = \cup_{i=1}^{n} P_i \,. \tag{14}$$

For example, the set of primitive types of the integrated CDM in the sample used in the previous section is

$$P_O = \cup_{i=1}^{3} P_i = P_1 \cup P_2 \cup P_3 = \{Text\} \,.$$

6.2. Classes

The merging process starts with the classes (Algorithm 9) as the root elements of the CDM. The input in the classes merging step is the output of the classes matching step, i.e.

the list of sets of matched classes $M_E = [M_E^{(1)}, \dots, M_E^{(m)}]$. Each set $M_E^{(i)}$ of matched classes is a candidate to be mapped into the class in the integrated CDM. The $M_E^{(i)}$ set is mapped into the $e_o \in E_O$ class if at least one of the following criteria is met:

– the number of elements in the $M_E^{(i)}$ set is equal to the number of input CDMs ($|M_E^{(i)}| = n$), meaning this set contains a class from each input CDM, or
– the $M_E^{(i)}$ set contains a class from an input CDM with the effectiveness for classes greater than or equal to the defined threshold ($e \in M_E^{(i)} \wedge e \in E_j \wedge F_{E_j} \geq th_{F_E}$), meaning the set contains a class from an input CDM with the acceptable effectiveness for classes, or
– the $M_E^{(i)}$ set contains a relevant class. A class is said to be relevant if its *relevance coefficient* is greater than or equal to the defined threshold (th_{R_E}).

The relevance coefficient of the $e \in E_j$ class is determined based on its attributes, associations, and generalizations, i.e.

$$\begin{aligned} rel_E(e) = {} & |attrs(e)| \cdot RW_A \\ & + (|\{r : r \in R_j \wedge (source(r) = e \vee target(r) = e)\}| \\ & + |\{g : g \in G_j \wedge (specific(g) = e \vee general(g) = e)\}|) \cdot RW_R \,, \end{aligned} \tag{15}$$

where RW_A and RW_R represent empirically determined weight coefficients for attributes and relationships (associations and generalizations), respectively. The value of the relevance coefficient is not normalized.

Algorithm 9 The classes merging step

```
Require: $M_E = [M_E^{(1)}, \dots, M_E^{(m)}]$
Ensure: $I_E = [I_E^{(1)}, \dots, I_E^{(m)}]$
Ensure: $E_O$
 1: $E_O \leftarrow \emptyset$
 2: for all $M_E^{(i)} \in M_E$ do
 3:     $cr\_met \leftarrow false$
 4:     if $|M_E^{(i)}| = n$ then
 5:         $cr\_met \leftarrow true$
 6:     else
 7:         for all $e \in M_E^{(i)}$ do
 8:             if $e \in E_j \wedge F_{E_j} \geq th_{F_E} \vee rel_E(e) \geq th_{R_E}$ then
 9:                 $cr\_met \leftarrow true$
10:             end if
11:         end for
12:     end if
13:     if $cr\_met$ then
14:         $e_o \leftarrow new\_class()$ ;   $e' \leftarrow max_{F_E}(M_E^{(i)})$
15:         $name(e_o) \leftarrow name(e')$ ;   $E_O \leftarrow E_O \cup \{e_o\}$ ;   $I_E^{(i)} \leftarrow e_o$
16:     else
17:         $I_E^{(i)} \leftarrow \emptyset$
18:     end if
19: end for
```

The name of the class (if created) is inherited from a class from CDM with the highest effectiveness for classes. Besides the set of classes of integrated CDM (E_O), the output of the algorithm is a list of indicators of created classes, where $I_E^{(i)}$ is the class created from the $M_E^{(i)}$ set or $I_E^{(i)} = \emptyset$ if the $M_E^{(i)}$ set is not mapped into a class.

Example. The classes merging step will be illustrated using the result from the example of the classes matching step, i.e., the following list of sets of matched classes

$$M_E = [\{A1, A2, A3\}, \{B1, B2, B3\}, \{C2\}, \{D3\}] \,.$$

The output of the classes merging step, is the list of indicators of created classes

$$I_E = [I_E^{(1)}, I_E^{(2)}, I_E^{(3)}, I_E^{(4)}] \,.$$

First, we consider the set $M_E^{(1)} = \{A1, A2, A3\}$. The number of elements in this set is equal to the number of input CDMs ($|\{A1, A2, A3\}| = 3$), and we map this set into a new class (AI) in the integrated CDM, and the indicator $I_E^{(1)}$ is set to this class ($I_E^{(1)} = AI$). If we assume that CDM_1 has the highest effectiveness for classes, then a new class in the integrated CDM inherits the name from the class $A1$ ($A1 \in E_1$), i.e. $name(AI) = name(A1)$.

Similarly, the next set $M_E^{(2)}$ is also mapped into a new class (BI) in the integrated CDM and the $I_E^{(2)}$ indicator is set to this class ($I_E^{(2)} = BI$). However, the number of elements in the third set is less than the number of input CDMs ($|\{C2\}| = 1 < 3$). If we assume that the effectiveness for classes of the second CDM is greater than or equal to the threshold (th_{F_E}), or if the $C2$ class is relevant (it does have an attribute and an association), then this set is also mapped into a new class (CI) in the integrated CDM, and the $I_E^{(3)}$ indicator is set to this class ($I_E^{(3)} = CI$). Of course, the CI class inherits the name from the $C2$ class, i.e. $name(CI) = name(C2)$.

The number of elements in the fourth set is also less than the number of input CDMs ($|\{D3\}| = 1 < 3$). If we assume that the effectiveness for classes of the third CDM is less than the threshold (th_{F_E}), and if the $D3$ class is not relevant (it does not have any attributes nor associations), then this set is not mapped into a new class and the $I_E^{(4)}$ indicator is unset ($I_E^{(4)} = \emptyset$).

Considering all of the above, we have $I_E = [AI, BI, CI, \emptyset]$ and $E_O = \{AI, BI, CI\}$. Figure 4 shows the partial result of the integration task after the classes merging step.

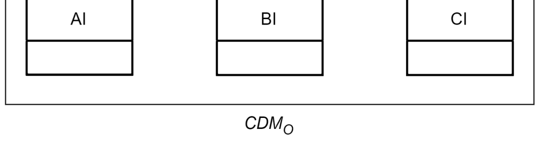

Fig. 4. Partial result of the integration task after the classes merging step

6.3. Attributes

The input in the attributes merging step (Algorithm 10) is the result of the attributes matching step (i.e. the list of sets of sets of matched attributes $M_A = [M_A^{(1)}, \ldots, M_A^{(m)}]$) and the result of the classes merging step (i.e. the list of class indicators $I_E = [I_E^{(1)}, \ldots, I_E^{(m)}]$). The input is also the list of sets of matched classes $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$.

In this step, it is necessary to create attributes for the classes that are already created in the integrated CDM, i.e. if $I_E^{(i)}$ is a class ($I_E^{(i)} \neq \emptyset$) then it is necessary to create attributes for this class based on the set of sets of matched attributes $M_A^{(i)}$.

Each set of matched attributes $ma_j \in M_A^{(i)}$ is a candidate to be mapped into the attribute of the class $I_E^{(i)} \neq \emptyset$. The ma_j set is mapped into the attribute $at_o \in attrs(I_E^{(i)})$ if at least one of the following criteria is met:

- the number of elements in the ma_j set is equal to the number of input CDMs ($|ma_j| = n$), meaning this set contains an attribute from a class from each input CDM (this is possible only if $|M_E^{(i)}| = n$), or
- the ma_j set contains an attribute from a class from CDM with the effectiveness for attributes greater than or equal to the defined threshold ($at \in ma_j \wedge at \in attrs(e) \wedge e \in E_k \wedge F_{A_k} \geq th_{F_A}$), meaning the set contains an attribute of the class from input CDM with the acceptable effectiveness for attributes.

Algorithm 10 The attributes merging step

```
Require: $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$
Require: $I_E = [I_E^{(1)}, \ldots, I_E^{(m)}]$
Require: $M_A = [M_A^{(1)}, \ldots, M_A^{(m)}]$
 1: for all $I_E^{(i)} \in I_E$ do
 2:     if $I_E^{(i)} \neq \emptyset$ then
 3:         for all $ma_j \in M_A^{(i)}$ do
 4:             $cr\_met \leftarrow false$
 5:             if $|ma_j| = n$ then
 6:                 $cr\_met \leftarrow true$
 7:             else
 8:                 for all $at \in ma_j$ do
 9:                     if $at \in attrs(e) \wedge e \in E_k \wedge F_{A_k} \geq th_{F_A}$ then
10:                         $cr\_met \leftarrow true$
11:                     end if
12:                 end for
13:             end if
14:             if $cr\_met$ then
15:                 $at_o \leftarrow new\_attribute()$ ;   $at' \leftarrow max_{F_A}(ma_j)$
16:                 $name(at_o) \leftarrow name(at')$ ;   $type(at_o) \leftarrow type(at')$
17:                 $attrs(I_E^{(i)}) \leftarrow attrs(I_E^{(i)}) \cup \{at_o\}$
18:             end if
19:         end for
20:     end if
21: end for
```

The name and the type of the attribute (if created) are inherited from the attribute of the class from the CDM with the highest effectiveness for attributes.

Example. In this example we need to create attributes for classes in the integrated CDM. The attributes of the class $I_E^{(i)} \neq \emptyset$ are created from the set of sets of matched attributes $M_A^{(i)}$. Each set in the $M_A^{(i)}$ set is a candidate to be mapped into an attribute (if the criteria is met).

First, we consider the class $I_E^{(1)} = AI$ and the set of sets of matched attributes $M_A^{(1)} = \{\{a11, a21, a31\}, \{a12, a22\}\}$. The first set in $M_A^{(1)}$ contains three elements, which is equal to the number of the input CDMs ($|\{a11, a21, a31\}| = 3$), thus this set is mapped into the $a1$ attribute of the AI class. If we assume that CDM_1 has the highest effectiveness for attributes, then the $a1$ attribute of the AI class inherits the name and type from the $a11$ attribute which is an attribute of the $A1$ class from CDM_1, i.e. $name(a1) = name(a11)$ and $type(a1) = type(a11)$.

The second set in $M_A^{(1)}$ contains two elements ($|\{a12, a22\}| = 2$). We already assumed that CDM_1 has the highest effectiveness for attributes. If we further assume that the effectiveness for attributes of CDM_1 is greater than or equal to the threshold (th_{F_A}), then this set is also mapped into an attribute ($a2$) of the AI class. The $a2$ attribute of the AI class inherits the name and type from the $a12$ attribute which is an attributes of the $A1$ class from CDM_1, i.e. $name(a2) = name(a12)$ and $type(a2) = type(a12)$.

Now, we come to the class $I_E^{(2)} = BI$ and the set of sets of matched attributes $M_A^{(2)} = \{\{b11, b21, b31\}\}$. The only set of matched attributes in the $M_A^{(2)}$ set is mapped into an attribute ($b1$) of the BI class because $|\{b11, b21, b31\}| = 3$.

Finally, we come to the class $I_E^{(3)} = CI$ and the set of sets of matched attributes $M_A^{(3)} = \{\{c21\}\}$. The $M_A^{(3)}$ set contains only one set of matched attributes ($\{c21\}$). This set has only one attribute ($|\{c21\}| = 1$). If we assume that the effectiveness for attributes of CDM_2 is also greater than or equal to the threshold (th_{F_A}), then the $\{c21\}$ set is also mapped into an attribute ($c1$) of the CI class. Also, the $c1$ attribute inherits the name and type from the $c21$ attribute, i.e. $name(c1) = name(c21)$ and $type(c1) = type(c21)$.

Considering all of the above, we have $attrs(AI) = \{a1, a2\}$, $attrs(BI) = \{b1\}$, and $attrs(CI) = \{c1\}$. The partial result of the integration task (after the attributes merging step) is shown in Fig. 5.

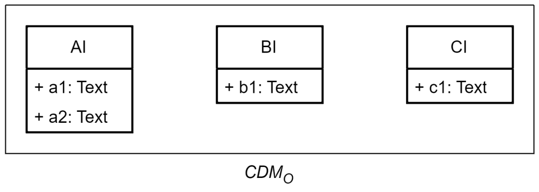

Fig. 5. Partial result of the integration task after the attributes merging step

6.4. Associations

The input in the associations merging step (Algorithm 11) is the result of the associations matching step (i.e. the list of sets of sets of matched associations M_R) and the result of the classes merging step (i.e. the list of class indicators I_E). The input is also the list of sets of matched classes M_E.

In this step, it is necessary to create associations between the classes that are already created in the integrated CDM, i.e. if $I_E^{(i)}$ and $I_E^{(j)}$ are classes ($I_E^{(i)} \neq \emptyset \wedge I_E^{(j)} \neq \emptyset$) then it is necessary to create associations between those classes based on the set of sets of matched associations $M_R^{(i)(j)}$.

Algorithm 11 The associations merging step

```
Require: $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$
Require: $I_E = [I_E^{(1)}, \ldots, I_E^{(m)}]$
Require: $M_R = [M_R^{(1)(1)}, \ldots, M_R^{(1)(m)}, M_R^{(2)(2)}, \ldots, M_R^{(2)(m)}, \ldots, M_R^{(m)(m)}]$
Ensure: $R_O$
 1: for all $M_R^{(i)(j)} \in M_R$ do
 2:     if $I_E^{(i)} \neq \emptyset \wedge I_E^{(j)} \neq \emptyset$ then
 3:         for all $mr_k \in M_R^{(i)(j)}$ do
 4:             $cr\_met \leftarrow false$
 5:             if $|mr_k| = n$ then
 6:                 $cr\_met \leftarrow true$
 7:             else
 8:                 for all $r \in mr_k$ do
 9:                     if $r \in R_p \wedge F_{R_p} \geq th_{F_R}$ then
10:                         $cr\_met \leftarrow true$
11:                     end if
12:                 end for
13:             end if
14:             if $cr\_met$ then
15:                 $r_o \leftarrow new\_association()$ ;    $r' \leftarrow max_{F_R}(mr_k)$
16:                 $name(r_o) \leftarrow name(r')$
17:                 $lower(source(r_o)) \leftarrow lower(source(r'))$
18:                 $upper(source(r_o)) \leftarrow upper(source(r'))$
19:                 $lower(target(r_o)) \leftarrow lower(target(r'))$
20:                 $upper(target(r_o)) \leftarrow upper(target(r'))$
21:                 if $type(source(r')) \in M_E^{(i)} \wedge type(target(r')) \in M_E^{(j)}$ then
22:                     $type(source(r_o)) \leftarrow I_E^{(i)}$ ;    $type(target(r_o)) \leftarrow I_E^{(j)}$
23:                 else if $type(source(r')) \in M_E^{(j)} \wedge type(target(r')) \in M_E^{(i)}$ then
24:                     $type(source(r_o)) \leftarrow I_E^{(j)}$ ;    $type(target(r_o)) \leftarrow I_E^{(i)}$
25:                 end if
26:                 $R_O \leftarrow R_O \cup \{r_o\}$
27:             end if
28:         end for
29:     end if
30: end for
```

Each set of matched associations $mr_k \in M_R^{(i)(j)}$ is a candidate to be mapped into the association between the classes $I_E^{(i)} \neq \emptyset$ and $I_E^{(j)} \neq \emptyset$. The set $mr_k \in M_R^{(i)(j)}$ is mapped into the association $r_o \in R_O$ if at least one of the following criteria is met:

- the number of elements in the set $mr_k \in M_R^{(i)(j)}$ is equal to the number of input CDMs ($|mr_k| = n$), meaning this set contains an association from each input CDM (this is possible only if $|M_E^{(i)}| = |M_E^{(j)}| = n$), or
- the set $mr_k \in M_R^{(i)(j)}$ contains an association from the CDM with the effectiveness for associations greater than or equal to the defined threshold ($r \in mr_k \wedge r \in R_p \wedge F_{R_p} \geq th_{F_R}$), meaning the set contains an association from input CDM with the acceptable effectiveness for associations.

The name and the association end multiplicities (if created) are inherited from the association from CDM with the highest effectiveness for associations.

Example. Using the result of the associations matching step, in the associations merging step we need to create associations in the integrated CDM.

Let us recall the result of the classes merging step

$$I_E = [AI, BI, CI, \emptyset] .$$

Since $I_E^{(4)} = \emptyset$, then sets $M_R^{(1)(4)}$, $M_R^{(2)(4)}$, $M_R^{(3)(4)}$, and $M_R^{(4)(4)}$ are automatically eliminated. The sets $M_R^{(1)(1)}$, $M_R^{(2)(2)}$, $M_R^{(2)(3)}$, and $M_R^{(3)(3)}$ are considered, but those sets are empty (meaning there are no candidate sets to be mapped into associations).

Non-empty sets are $M_R^{(1)(2)}$ and $M_R^{(1)(3)}$. The $M_R^{(1)(2)}$ set contains two sets of matched associations between the corresponding pairs of classes from $M_E^{(1)}$ and $M_E^{(2)}$. The number of the elements in the first set is equal to the number of input CDMs ($|\{Z1, Z2, Z3\}| = 3$), thus this set is mapped into an association (ZI) between the classes AI and BI in the integrated CDM. Assuming that CDM_1 has the highest effectiveness for associations, the ZI association inherits the name and end multiplicities from the $Z1$ association. Further, if we assume that the $A1$ class is the *source* end of $Z1$ and the $B1$ class is the *target* end of $Z1$, then we have

$$\begin{gathered} name(ZI) = name(Z1), \quad source(ZI) = AI, \quad target(ZI) = BI, \\ lower(source(ZI)) = 1, \quad upper(source(ZI)) = 1, \\ lower(target(ZI)) = 0, \quad upper(target(ZI)) = \infty . \end{gathered}$$

The second set of the $M_R^{(1)(2)}$ set contains only one element ($|\{Y2\}| = 1$). If we assume that CDM_2 has the effectiveness for associations greater than or equal to the threshold (th_{F_R}), then this set is also mapped into an association (YI) between the classes AI and BI in the integrated CDM. The YI association inherits the name and end multiplicities from the $Y2$ association. If we assume that the $B2$ class is the *source* end of $Y2$ and the $A2$ class is the *target* end of $Y2$ (opposite from the case with $Z1$), then we have

$$\begin{gathered} name(YI) = name(Y2), \quad source(YI) = BI, \quad target(YI) = AI, \\ lower(source(YI)) = 0, \quad upper(source(YI)) = \infty, \\ lower(target(YI)) = 0, \quad upper(target(YI)) = 1 . \end{gathered}$$

Finally, we come to the set $M_R^{(1)(3)} = \{\{X2\}\}$, which contains only one set of matched associations ($\{X2\}$). This set has only one association ($|\{X2\}| = 1$). We already assumed that the effectiveness for associations for CDM_2 is greater than or equal to the threshold (th_{F_R}), meaning that the $\{X2\}$ set is also mapped into an association (XI) in the integrated CDM. If we assume that the $A2$ class is the *source* end of $X2$ and the $C2$ class is the *target* end of $X2$, then we have

$$name(XI) = name(X2), \quad source(XI) = AI, \quad target(XI) = CI,$$
$$lower(source(XI)) = 1, \quad upper(source(XI)) = 1,$$
$$lower(target(XI)) = 0, \quad upper(target(XI)) = \infty\,.$$

Considering all of the above, we have $R_O = \{ZI, YI, XI\}$. The result of the integration task (after the associations merging step) is shown in Fig. 6.

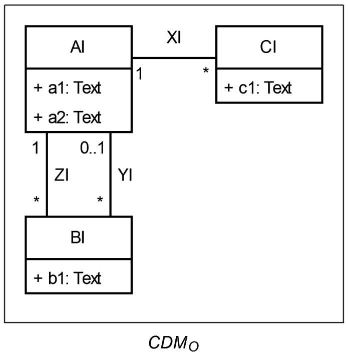

Fig. 6. Result of the integration task after the associations merging step

6.5. Generalizations

In this step it is necessary to create generalizations between the classes that are already created in the integrated CDM. The input in the generalizations merging step (Algorithm 12) is the result of the generalizations matching step (i.e. the list of sets of sets of matched generalizations M_G), the result of the classes merging step (i.e. the list of class indicators I_E), as well as the list of sets of matched classes M_E.

Each set of matched generalizations $mg_k \in M_G^{(i)(j)}$ is a candidate to be mapped into the generalization between the classes $I_E^{(i)} \neq \emptyset$ and $I_E^{(j)} \neq \emptyset$. The mg_k set is mapped into the generalization $g_o \in G_O$ if at least one of the following criteria is met:

- the number of elements in the mg_k set is equal to the number of input CDMs ($|mg_k| = n$), meaning this set contains a generalization from each input CDM (this is possible only if $|M_E^{(i)}| = |M_E^{(j)}| = n$), or
- the mg_k set contains a generalization from CDM with the effectiveness for generalizations greater than or equal to the defined threshold ($g \in mg_k \land g \in G_p \land F_{G_p} \geq th_{F_G}$), meaning the set contains a generalization from input CDM with the acceptable effectiveness for generalizations.

Algorithm 12 The generalizations merging step

```
Require: $M_E = [M_E^{(1)}, \ldots, M_E^{(m)}]$
Require: $I_E = [I_E^{(1)}, \ldots, I_E^{(m)}]$
Ensure: $M_G = [M_G^{(1)(2)}, \ldots, M_G^{(1)(m)}, M_G^{(2)(3)}, \ldots, M_G^{(2)(m)}, \ldots, M_G^{(m-1)(m)}]$
Ensure: $G_O$
 1: for all $M_G^{(i)(j)} \in M_G$ do
 2:     if $I_E^{(i)} \neq \emptyset \wedge I_E^{(j)} \neq \emptyset$ then
 3:         for all $mg_k \in M_G^{(i)(j)}$ do
 4:             $cr\_met \leftarrow false$
 5:             if $|mg_k| = n$ then
 6:                 $cr\_met \leftarrow true$
 7:             else
 8:                 for all $g \in mg_k$ do
 9:                     if $g \in G_p \wedge F_{G_p} \geq th_{F_G}$ then
10:                         $cr\_met \leftarrow true$
11:                     end if
12:                 end for
13:             end if
14:             if $cr\_met$ then
15:                 $g_o \leftarrow new\_generalization()$ ;    $g' \leftarrow max_{F_G}(mg_k)$
16:                 if $specific(g') \in M_E^{(i)} \wedge general(g') \in M_E^{(j)}$ then
17:                     $specific(g_o) \leftarrow I_E^{(i)}$ ;    $general(g_o) \leftarrow I_E^{(j)}$
18:                 else if $specific(g') \in M_E^{(j)} \wedge general(g') \in M_E^{(i)}$ then
19:                     $specific(g_o) \leftarrow I_E^{(j)}$ ;    $general(g_o) \leftarrow I_E^{(i)}$
20:                 end if
21:                 $G_O \leftarrow G_O \cup \{g_o\}$
22:             end if
23:         end for
24:     end if
25: end for
```

Due to the article length and the similarity of the generalizations merging step to the associations merging step (there can be only one generalization between two classes, while there can be more than one association between two classes) we do not provide an example for the generalization merging step.

6.6. Tidying up

In the final step of the schema integration process, we create surrogate attributes, i.e. attributes named *id*, for classes that do not already have an attribute named *id* and do not have generalizations. If a class already has an attribute named *id* and it has generalizations, then such attribute is removed from the class. Also, an operation named *PK* which represents the *primary key* specification is added to classes that do not have generalizations. Each *PK* operation has a single parameter named *id* (which corresponds to a surrogate attribute of the class).

7. Implemented Tool

This section presents the implemented tool named DBomnia[3]. DBomnia is the first online web-based tool providing the functionality of CDM derivation from heterogeneous source artifacts. Currently, DBomnia enables automatic CDM derivation from two different types of source artifacts: textual specifications and collections of BPMs. Figure 7 shows the architecture of the DBomnia tool, while Fig. 8 shows a screenshot of the tool in action.

The process of the CDM synthesis consists of four steps: (1) generation of the CDM from the source collection of BPMs, (2) generation of the CDM from the input textual specification, (3) integration of the generated CDMs, and (4) generation of the diagram layout for the integrated CDM.

In the first step, the source collection of BPMs is sent to AMADEOS. AMADEOS generates the corresponding CDM and responds with a JSON object containing the generated CDM, execution status, etc. In the second step, the input textual specification is sent to TexToData, which generates the corresponding CDM. TexToData also responds with a JSON object containing generated CDM, error messages (if any), etc.

In the third step, DBomnia integrates the CDMs generated in the first two steps, by applying the approach described in the previous sections. Then, in the fourth step, it sends the integrated CDM to the Layouter service, which generates the corresponding diagram layout. Layouter is the pre-existing service that provides the functionality of generating a diagram layout for the input UML class diagram (or in our case, CDM represented by the UML class diagram). Layouter responds with a file containing the generated layout.

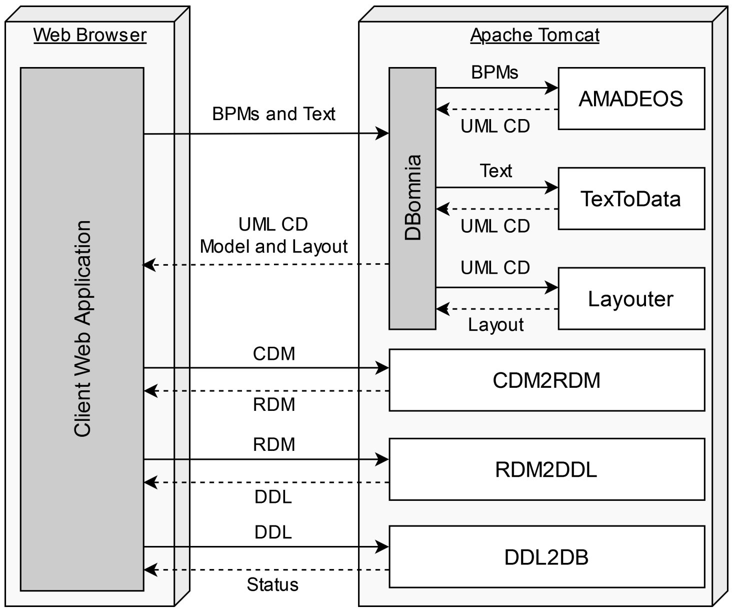

Fig. 7. DBomnia architecture

[3] http://m-lab.etf.unibl.org:8080/dbomnia

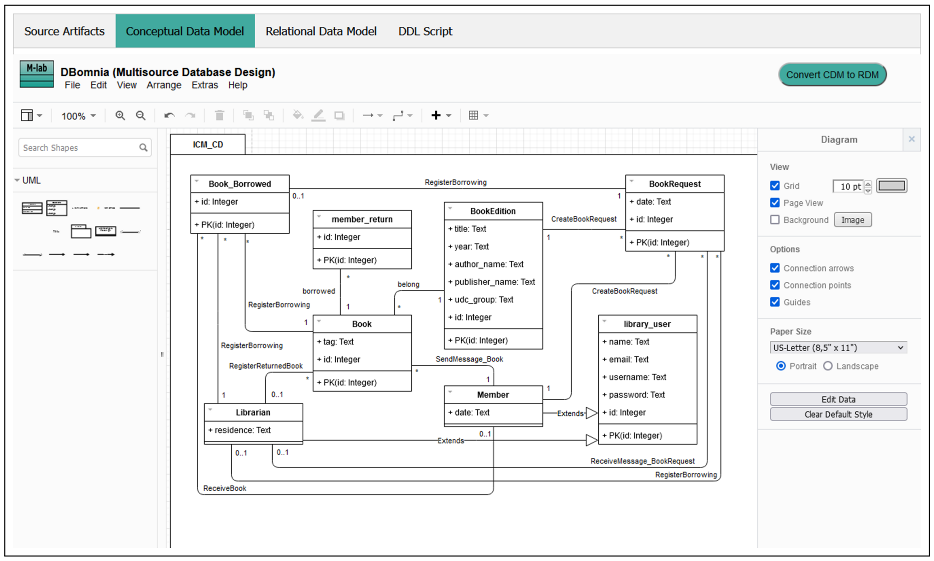

Fig. 8. Screenshot of DBomnia in action

Finally, DBomnia prepares and returns the response – a JSON object containing the integrated CDM, diagram layout, and status information.

The user interface of the client web application is separated into four tabs: *Source Artifacts*, *Conceptual Data Model*, *Relational Data Model*, and *DDL Script*. Tab *Source Artifacts* contains fields that allow users to input a textual specification and upload a collection of source BPMs. Upon the user's request (click on the *Generate CDM* button), all source artifacts are sent to DBomnia. When the client web application receives the JSON response, it visualizes[4] the class diagram in the browser (in the *Conceptual Data Model* tab). The visualized diagram is editable, so users can additionally improve it.

DBomnia also supports all the subsequent steps of forward database engineering, from CDM to the target physical database, by employing CDM2RDM, RDM2DDL, and DDL2DB services, as described in [53].

8. Evaluation

This section presents the evaluation of the approach and the tool on three case studies. One case study-based evaluation is described in detail, while only the most significant results are provided for the other two.

8.1. Environment Setup

This subsection provides concrete values of the estimated measures for input CDMs, as well as empirically determined thresholds and weights used by the tool.

Table 1 shows estimated measures for input CDMs generated by AMADEOS and estimated measures for input CDMs generated by TexToData. The measures for input

[4] Implementation is based on the mxGraph library (https://jgraph.github.io/mxgraph/)

Table 1. Values of estimated measures for input CDMs

Tool	F_E	F_A	F_R	F_G
AMADEOS	0.76	0.00	0.41	0.00
TexToData	0.70	0.72	0.30	1.00

CDMs generated by AMADEOS are experimentally determined [11], while measures for input CDMs generated by TexToData are determined from case study-based evaluation. AMADEOS is able to generate a highly complete CDM structure, but generates only *id* attribute which represents the surrogate key in each entity type. TexToData is able to generate CDMs of less complete and less correct structure but with a more complete set of attributes in each entity type. Also, AMADEOS does not generate generalizations.

Table 2 shows threshold and weight values. To eliminate only minor typographical errors, the Levenshtein distance threshold is set to 1. The weight of attributes in similarity calculation for classes in the current environment (with AMADEOS and TexToData) does not play any role, because AMADEOS does not generate attributes (except for the surrogate key attribute *id* ignored in the integration task). Currently, for the same reason, the value of the similarity threshold for attributes is irrelevant. To keep the attributes generated by TexToData, the effectiveness threshold for attributes is lower than the effectiveness measure for attributes for TexToData.

Although AMADEOS generates associations with lower precision than classes, the CDM generated by AMADEOS is expected to be more precise and more complete than the CDM generated by TexToData regarding associations. To increase the completeness of the associations in the integrated CDM, the environment is set to keep unmatched associations from both input CDMs. We consider two associations a match if their cardinalities (upper values of the respective member ends) are equal or if their name similarity is very high. We only compare the cardinalities because we expect that the CDM generated by AMADEOS will be more precise than the CDM generated by TexToData regarding the participation constraints.

On the other hand, in order to reduce the number of excessive classes, the environment is set up to keep only relevant unmatched classes, i.e. the effectiveness threshold for classes is greater than the effectiveness measure for classes for AMADEOS and TexToData. Currently, an unmatched class is considered to be relevant if it has at least one attribute or at least one association or if it participates in at least one generalization (the relevance threshold for classes is set to one, while the weight of attributes and weight of associations and generalizations is also set to one).

Since AMADEOS does not generate generalizations, we want to keep generalizations generated by TexToData (the effectiveness threshold for generalizations is less than the effectiveness measure for generalizations for TexToData).

Note that we provided concrete values for thresholds and weights. Since currently there can be only two input CDMs (CDM generated by AMADEOS and CDM generated by TexToData), other values can be set up to achieve the same result. For example, if we want to keep only relevant unmatched classes from both CDMs, then the value of the effectiveness threshold for classes can be any value in the segment $(0.76, 1]$, i.e. any value greater than the maximum of the effectiveness measure for classes for AMADEOS and TexToData.

Table 2. Values for thresholds and weights

Symbol	Description	Value
LD	Levenshtein distance threshold	1
W_A	Weight of attributes in similarity calculation for classes	0.00
W_{card}	Weight of cardinalities in similarity calculation for associations	0.75
W_{pc}	Weight of participation constraints in similarity calculation for associations	0.00
th_{S_E}	Similarity threshold for classes	0.25
th_{S_A}	Similarity threshold for attributes	1.00
th_{S_R}	Similarity threshold for associations	0.20
th_{S_G}	Similarity threshold for generalizations	1.00
th_{F_E}	Effectiveness threshold for classes	0.80
th_{F_A}	Effectiveness threshold for attributes	0.50
th_{F_R}	Effectiveness threshold for associations	0.10
th_{F_G}	Effectiveness threshold for generalizations	0.10
th_{R_E}	Relevance threshold for classes	1.00
RW_A	Weight of attributes in relevance calculation for classes	1.00
RW_R	Weight of associations and generalizations in relevance calculation for classes	1.00

8.2. Case Study: *Online Library*

We evaluated the approach through the assessment of the CDM derived from a set of heterogeneous source artifacts against the CDMs that are derived from sources of one single type. Firstly, we prepared a sample source set of BPMs and a textual description of the *Online Library*, as well as the reference (target) CDM (Fig. 9). Secondly, we evaluated generated CDMs (CDM generated by AMADEOS from source BPMs, CDM generated by TexToData from textual specification, and CDM generated by DBomnia) against the reference CDM. Finally, we compared the results obtained by DBomnia against those obtained by AMADEOS and TexToData.

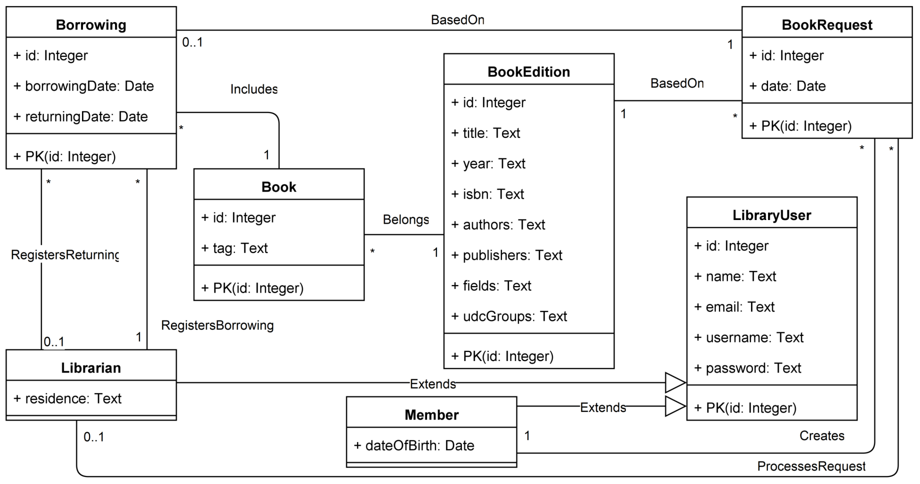

Fig. 9. Reference CDM of the *Online Library*

AMADEOS. The sample source set of BPMs (Fig. 10) represents two main processes in the *Online Library – Book Borrowing* and *Book Returning*. In the first process: a member creates a borrowing request for a copy of some book edition; a librarian registers the borrowing and issues a book. In the second process: the member returns the borrowed book, and the librarian registers the returned book.

Figure 11 shows the CDM derived from the source set of BPMs. When compared against the reference CDM (Fig. 9):

1. all classes, four *id* attributes, and six associations (*BookEdition-CreateBookRequest-BookRequest*, *BookRequest-RegisterBorrowing-Book_Borrowed*, *Member-CreateBookRequest-BookRequest*, *Librarian-RegisterBorrowing-BookRequest*, *Librarian-RegisteBorrowing-Book_Borrowed*, *Book-RegisterBorrowing-Book_Borrowed*) could be evaluated as correctly generated,
2. two *id* attributes and four associations (*Librarian-RegisterReturnedBook-Book*, *Librarian-ReceiveMessage_BookRequest-BookRequest*, *Member-SendMessage_Book-Book*, *Member-ReceiveBook-Book_Borrowed*) could be evaluated as surplus, and

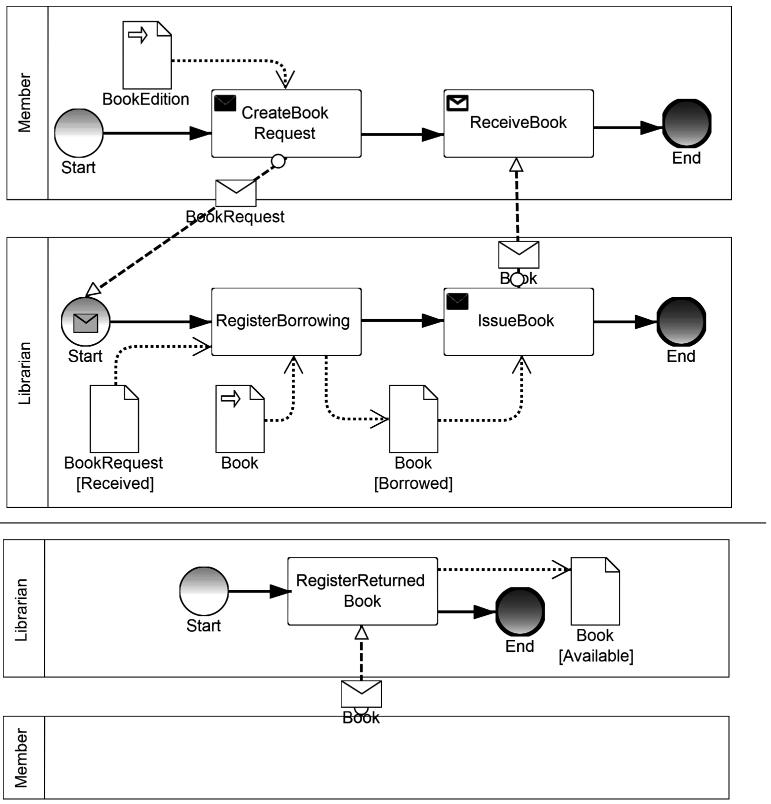

Fig. 10. Sample set of BPMs: *Book Borrowing* (top), *Book Returning* (bottom)

3. one class (*LibraryUser*), 18 attributes, both generalizations, and two associations (*BookEdition*↔*Book*, and *Librarian*↔*Book_Borrowed* that represents the returning of the book) are missing in the generated CDM.

Table 3 shows the results of quantitative evaluation based on the previous discussion.

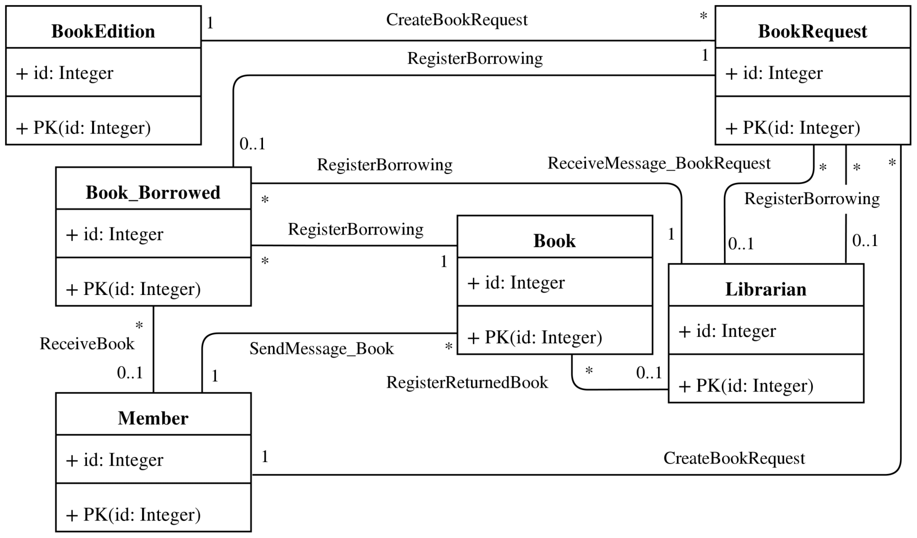

Fig. 11. CDM generated by AMADEOS based on the source set of BPMs

Table 3. Quantitative evaluation of the CDM generated by AMADEOS

Concept	N_c	N_m	N_w	R	P	F
Classes	6	1	0	0.857	1	0.923
Attributes	4	18	2	0.182	0.667	0.286
Associations	6	2	4	0.75	0.6	0.667
Generalizations	0	2	0	0	–	–

TexToData. Figure 12 shows the textual description of the *Online Library*, while Fig. 13 shows the CDM derived from the given textual description. The automatically generated CDM has ten classes, 13 attributes, two generalizations, and only four associations. When compared against the reference CDM (Fig. 9):

1. seven classes (*library_user*, *librarian*, *member*, *borrowing_request*, *book*, *book_edition*, and *borrowing*) and 11 attributes, both generalizations, and three associations (*book-belong-book_edition*, *book_edition-belongs-borrowing_request*, *member-creates-borrowing_request*) could be evaluated as correctly generated,
2. one association (*book-borrowed-member_return*) and three classes (*issue*, *register*, *member_return*) could be evaluated as surplus,
3. two attributes (*date* in the *member* class and *date* in the *borrowing_request* class) could be evaluated as incorrectly generated (incorrect type), and
4. five associations, and nine attributes are missing in the generated CDM.

Table 4 shows the results of quantitative evaluation based on the previous discussion.

Library users are librarians or members. Library user has name, email, username, and password. Librarian has residence. Member has date of birth. Book edition has title, year, ISBN, authors names, publishers names, fields, and UDC groups. Book has tag. Books belong to book edition. Member creates borrowing requests. Borrowing request has date. Borrowing requests belongs to book edition. Librarian registers borrowings and issues books. Member returns borrowed book. Librarian registers returned book.

Fig. 12. Sample textual description of the *Online Library*

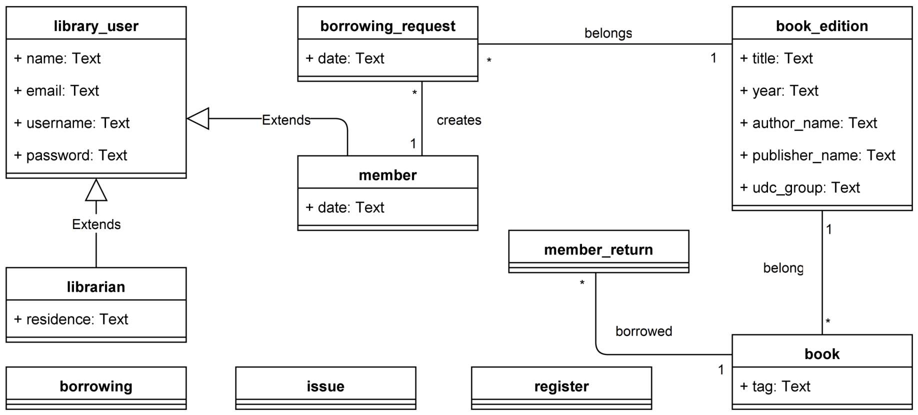

Fig. 13. CDM generated by TexToData based on the sample textual specification

Table 4. Quantitative evaluation of the CDM generated by TexToData

Concept	N_c	N_m	N_w	R	P	F
Classes	7	0	3	1	0.7	0.824
Attributes	11	9	2	0.55	0.846	0.667
Associations	3	5	1	0.375	0.75	0.5
Generalizations	2	0	0	1	1	1

DBomnia. When we use the sample set of BPMs (Fig. 10) and sample textual description (Fig. 12) with DBomnia, we obtain the CDM (Fig. 14) containing eight classes, 19 attributes, 12 associations, and two generalizations. When compared against the reference CDM (Fig. 9):

1. seven classes, 16 attributes, both generalizations, and seven associations could be evaluated as correct,
2. one class (*member_return*) and corresponding *id* attribute, and five associations (*Librarian-RegisterReturnedBook-Book*, *Member-SendMessage_Book-Book*, *Book-borrowed-member_return*, *Librarian-ReceiveMessage_BookRequest-BookRequest*, *Member-ReceiveBook-Book_Borrowed*) could be evaluated as surplus,
3. two attributes (*date* in the *Member* class and *date* in the *BookRequest* class) could be evaluated as incorrectly generated (incorrect type), and

4. only one association (*Librarian↔Book_Borrowed*) and only four attributes (*ISBN* and *fields* in the *BookEdition* class, and *borrowing_date* and *returning_date* in the *Book_Borrowed* class) are missing in the generated CDM.

Table 5 shows the results of quantitative evaluation based on the previous discussion.

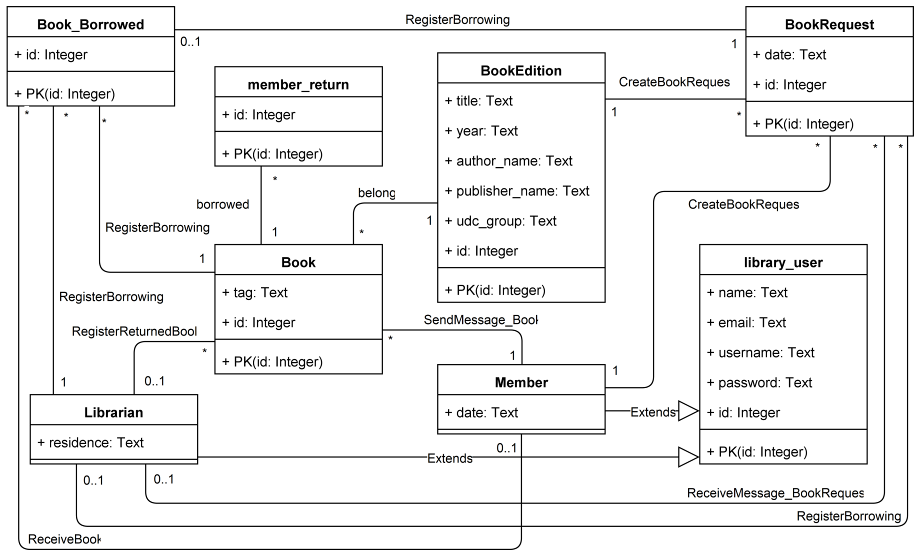

Fig. 14. CDM of the *Online Library* generated by DBomnia

Table 5. Quantitative evaluation of the CDM generated by DBomnia

Concept	N_c	N_m	N_w	R	P	F
Classes	7	0	1	1	0.875	0.933
Attributes	16	4	3	0.8	0.842	0.821
Associations	7	1	5	0.875	0.583	0.7
Generalizations	2	0	0	1	1	1

Results discussion. Table 6 shows the overview of the quantitative evaluation (i.e. the effectiveness measure) for AMADEOS, TexToData, and DBomnia. The presented results show that DBomnia still does not generate 100% complete nor 100% correct target model. However, the effectiveness measure for each CDM concept generated by DBomnia is greater than or equal to the higher value of the effectiveness measure for the corresponding concept generated by AMADEOS and TexToData. Although the automatically generated CDM has one surplus class, it is easy to spot and simply delete such (surplus) classes – unlike the correctly generated classes, such classes (probably) will not contain attributes or will not have associations with other classes. We believe that it is easier to delete a surplus class than to add a new class when it's missing in the generated CDM.

The results show that the automatic derivation of the CDM from a set of heterogeneous source artifacts is more effective than each independent automatic derivation of the CDM based on sources of one single type only. Hence, this case study-based evaluation of the approach and implemented tool confirm the research goal.

Table 6. Overview of the quantitative evaluation for the *Online Library*

Effectiveness measure	AMADEOS	TexToData	DBomnia
F_E	0.923	0.824	0.933
F_A	0.286	0.667	0.821
F_R	0.667	0.5	0.7
F_G	–	1	1

8.3. Case Studies: *Admission Exam* and *Student Jobs*

In order to conduct a more extensive evaluation, compared to the illustrative example from the previous subsection, we run two more case study-based evaluations as follows. Firstly, two authors of the paper had to choose an arbitrary domain and create a set of at least ten BPMs and a textual specification (at least 100 words) for the chosen domain. Secondly, they had to (i) exchange created artifacts, (ii) manually design a reference CDM based on received source artifacts, and (iii) evaluate CDMs generated by AMADEOS, TexToData, and DBomnia against the manually designed reference CDM. Finally, the third author had to inspect created artifacts and reference CDMs, as well as the entire evaluation process.

The chosen domains were *Admission Exam* and *Student Jobs*. Source artifacts[5] consist of 12 BPMs and 112-word textual specification for *Admission Exam*, and ten BPMs and 117-word textual specification for *Student Jobs*. Reference CDM[6] for *Admission Exam* contains 16 classes, 41 attributes, 25 associations, and three generalizations, while reference CDM for *Student Jobs* contains 15 classes, 37 attributes, 18 associations, and three generalizations.

Table 7 shows the results of quantitative evaluation[7] for both domains and all three tools. Each row of the table contains values for one concept type (classes – *E*, attributes – *A*, associations – *R*, or generalizations – *G*) of the generated CDM.

Results discussion. Table 8 shows the overview (only F-scores) of the quantitative evaluation, while Fig. 15 shows the comparison of the F-score values. The obtained results confirm the results obtained for the illustrative example, i.e. the effectiveness measure for each CDM concept generated by DBomnia is greater than or equal to the higher value of the effectiveness measure for the corresponding concept generated by AMADEOS and TexToData. The proposed approach and implemented tool (DBomnia) exploit the best features of other tools (as already mentioned, AMADEOS is able to generate a highly complete CDM structure, while TexToData is able to generate CDMs of less complete and less correct structure but with a more complete set of attributes in each entity type) to achieve more effective derivation of the CDMs. Therefore, the evaluation of the approach and implemented tool on three case-studies confirm the research goal. Since the results of the case study-based evaluation are positive, we are now directed toward extensive evaluation of the approach and implemented tool.

[5] Source artifacts for both domains are available at:
https://gitlab.com/m-lab-research/ComSIS-2024/-/tree/main/01-SourceArtifacts

[6] Reference CDMs for both domains are available at:
https://gitlab.com/m-lab-research/ComSIS-2024/-/tree/main/02-ReferenceCDMs

[7] CDMs generated by all three tools for both domains are available at:
https://gitlab.com/m-lab-research/ComSIS-2024/-/tree/main/03-GeneratedCDMs

Table 7. Quantitative evaluation for *Admission Exam* and *Student Jobs*

Tool		*Admission Exam*						*Student Jobs*					
		N_c	N_m	N_w	R	P	F	N_c	N_m	N_w	R	P	F
AMADEOS	*E*	16	0	3	1	0.842	0.914	15	0	0	1	1	1
	A	13	28	6	0.317	0.684	0.433	12	25	3	0.324	0.8	0.462
	R	24	1	13	0.96	0.649	0.774	15	1	14	0.938	0.517	0.667
	G	0	3	0	0	–	–	0	3	0	0	–	–
TexToData	*E*	14	2	2	0.875	0.875	0.875	11	4	2	0.733	0.846	0.786
	A	25	16	0	0.61	1	0.758	16	21	1	0.432	0.941	0.593
	R	7	18	1	0.28	0.875	0.424	7	11	5	0.389	0.583	0.467
	G	3	0	0	1	1	1	3	0	0	1	1	1
DBomnia	*E*	16	0	3	1	0.842	0.914	15	0	0	1	1	1
	A	38	3	3	0.927	0.927	0.927	28	9	1	0.757	0.966	0.848
	R	25	0	14	1	0.641	0.781	16	0	15	1	0.516	0.681
	G	3	0	0	1	1	1	3	0	0	1	1	1

Table 8. Summary of the evaluation results

Measure	*Admission Exam*			*Student Jobs*		
	AMADEOS	TexToData	DBomnia	AMADEOS	TexToData	DBomnia
F_E	0.914	0.875	**0.914**	1	0.786	**1**
F_A	0.433	0.758	**0.927**	0.462	0.593	**0.848**
F_R	0.774	0.424	**0.781**	0.667	0.467	**0.681**
F_G	–	1	**1**	–	1	**1**

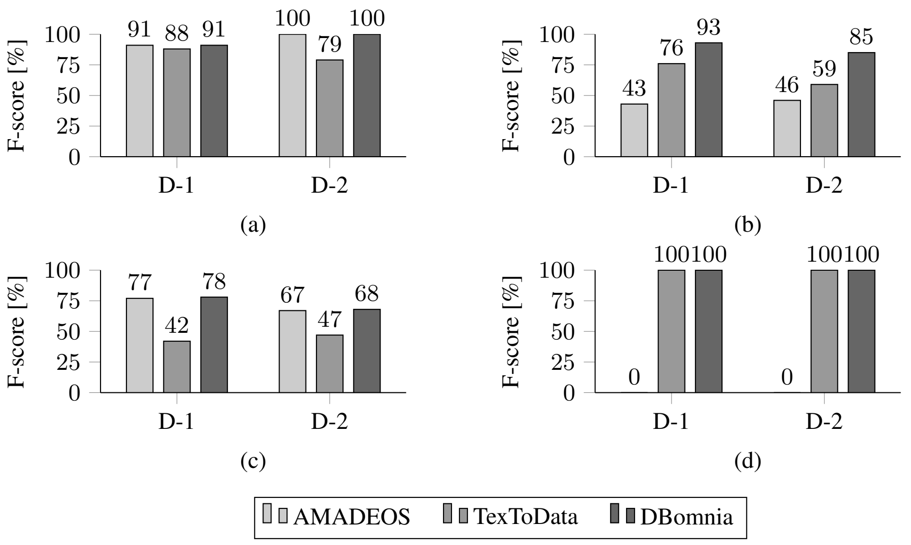

Fig. 15. Comparison of (a) F_E, (b) F_A, (c) F_R, and (d) F_G measures by tool per domain (D-1 – *Admission Exam*, D-2 – *Student Jobs*)

9. Conclusion

In this article we proposed an approach to the automatic CDM derivation from heterogeneous source artifacts. The approach is based on the integration of CDMs that are derived from source artifacts of one single type by already existing tools, whereby the main characteristic of these CDMs is uncertainty. Uncertainty of CDMs automatically derived from specific source artifacts is expressed and managed through the effectiveness measure of generation of specific concepts of the input CDMs.

The approach is implemented by the DBomnia tool – the first online web-based tool enabling automatic CDM derivation from a heterogeneous set of source artifacts, whereby the currently supported source artifacts are BPMs and textual specifications. DBomnia employs other tools (AMADEOS and TexToData) to generate CDMs from specific source artifacts (AMADEOS derives CDM from BPMs, while TexToData derives CDM from textual specifications) and then integrates the generated CDMs into a single unified CDM.

The approach and implemented tool were evaluated in three case studies. As expected, evaluation proves that the implemented approach enables effective automatic derivation of the conceptual database model from a set of heterogeneous source artifacts. Automatic derivation of the conceptual database model from a set of heterogeneous source artifacts is more effective than each independent automatic CDM derivation from sources of one single type only.

In line with the long-term research goals, our future work will focus on the approach and tool improvements and will include: work on semantic matching and combination of multiple languages in one execution of the automatic CDM derivation, further improvements of the specific CDM generators, inclusion of other types of source artifacts, and thorough validation and verification. Our intention is also to evaluate the approach with more complex real collections of BPMs and textual specifications.

Although the great advantage of the implemented multilingual tool is a large number of supported languages, the semantic analysis (e.g. analysis of the acronyms, synonyms, etc.) constitutes a great challenge and it is not considered in this paper. Taking that into consideration, as well as the effort of increasing the effectiveness of the overall CDM derivation process, we will focus on exploring the possibility of leveraging the ML techniques in our approach.

References

1. Adamson, G.W., Boreham, J.: The use of an association measure based on character structure to identify semantically related pairs of words and document titles. Information Storage and Retrieval 10(7), 253–260 (1974)
2. Anam, S., Kim, Y.S., Kang, B., Liu, Q.: Designing a knowledge-based schema matching system for schema mapping. Australasian Data Mining Conference (1 2015), https://figshare.utas.edu.au/articles/conference_contribution/Designing_a_knowledge-based_schema_matching_system_for_schema_mapping/23095379
3. Axler, S.: Linear Algebra Done Right. Springer Cham (2023)
4. Banjac, G., Brdjanin, D., Banjac, D.: Towards automatic conceptual database design based on heterogeneous source artifacts. In: Abelló, A. et al. (ed.) New Trends in Database and Information Systems. pp. 487–498. Springer Nature Switzerland, Cham (2023)
5. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Comput. Surv. 18(4), 323–364 (1986)

6. Berlin, J., Motro, A.: Database schema matching using machine learning with feature selection. In: Pidduck, A.B., Ozsu, M.T., Mylopoulos, J., Woo, C.C. (eds.) Advanced Information Systems Engineering. pp. 452–466. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
7. Bernstein, P., Madhavan, J., Rahm, E.: Generic schema matching, ten years later. Proc. VLDB Endow. 4(11), 695–701 (2011)
8. Bernstein, P.A., Halevy, A.Y., Pottinger, R.A.: A vision for management of complex models. SIGMOD Rec. 29(4), 55–63 (dec 2000), https://doi.org/10.1145/369275.369289
9. Brdjanin, D., Maric, S.: An Approach to Automated Conceptual Database Design Based on the UML Activity Diagram. Computer Science and Information Systems 9(1), 249–283 (2012)
10. Brdjanin, D., Maric, S.: Model-driven Techniques for Data Model Synthesis. Electronics 17(2), 130–136 (2013)
11. Brdjanin, D., Vukotic, A., Banjac, D., Banjac, G., Maric, S.: Automatic derivation of the initial conceptual database model from a set of business process models. Computer Science and Information Systems 19(1), 455–493 (2022)
12. Brdjanin, D., Banjac, G., Babic, N., Golubovic, N.: Towards the speech-driven database design. In: Proc. of TELFOR 2022. pp. 1–4. IEEE (2022)
13. Brdjanin, D., Grumic, M., Banjac, G., Miscevic, M., Dujlovic, I., Kelec, A., Obradovic, N., Banjac, D., Volas, D., Maric, S.: Towards an online multilingual tool for automated conceptual database design. In: Braubach, L., et al. (eds.) Intelligent Distributed Computing XV. pp. 144–153. Springer (2023)
14. Bulygin, L.: Combining lexical and semantic similarity measures with machine learning approach for ontology and schema matching problem. In: Proceedings of the XX International Conference “Data Analytics and Management in Data Intensive Domains”(DAMDID/RCDL’2018). pp. 245–249 (2018)
15. Chen, P.: English sentence structure and entity-relationship diagrams. Information Sciences 29(2-3), 127–149 (1983)
16. Choobineh, J., Mannino, M., Nunamaker, J., Konsynsky, B.: An expert database design system based on analysis of forms. IEEE Transaction on Software Engineering 14(2), 242–253 (1988)
17. Choobineh, J., Lo, A.W.: CABSYDD: Case-based system for database design. Journal of Management Information Systems 21(3), 281–314 (2004)
18. Cohen, W.W., Ravikumar, P., Fienberg, S.: A comparison of string metrics for matching names and records. In: Proc. of KDD 2003, Workshop on Data Cleaning, Record Linkage, and Object Consolidation (2003)
19. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Commun. ACM 7(3), 171–176 (1964)
20. Date, C.: An Introduction to Database Systems, 8th edn. Addison-Wesley (2003)
21. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling schemas of disparate data sources: a machine-learning approach. SIGMOD Rec. 30(2), 509–520 (may 2001), https://doi.org/10.1145/376284.375731
22. Duchateau, F., Coletta, R., Bellahsene, Z., Miller, R.J.: (not) yet another matcher. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. p. 1537–1540. CIKM ’09, Association for Computing Machinery, New York, NY, USA (2009), https://doi.org/10.1145/1645953.1646165
23. Friedman, C., Sideli, R.: Tolerating spelling errors during patient validation. Computers and Biomedical Research 25(5), 486–509 (1992)
24. Gali, N., Mariescu-Istodor, R., Hostettler, D., Fränti, P.: Framework for syntactic string similarity measures. Expert Systems with Applications 129, 169–185 (2019)
25. Gotoh, O.: An improved algorithm for matching biological sequences. Journal of Molecular Biology 162(3), 705–708 (1982)
26. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal 29(2), 147–160 (1950)

27. Harmain, H., Gaizauskas, R.: CM-Builder: A Natural Language-Based CASE Tool for Object-Oriented Analysis. Automated Software Eng. 10(2), 157–181 (2003)
28. Hartmann, S., Link, S.: English sentence structures and EER modeling. In: Proc. of the 4th Asia-Pacific conf. on conceptual modelling – Vol. 67. pp. 27–35 (2007)
29. Jaro, M.A.: Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. Journal of the American Statistical Association 84(406), 414–420 (1989)
30. Jouault, F., Allilaire, F., Bezivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)
31. Kriouile, A., Addamssiri, N., Gadi, T.: An MDA Method for Automatic Transformation of Models from CIM to PIM. American J. of Software Eng. and Applications 4(1), 1–14 (2015)
32. Levenshtein, I.V.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
33. Lukovic, I., Mogin, P., Pavicevic, J., Ristic, S.: An approach to developing complex database schemas using form types. Software: Practice & Experience 37(15), 1621–1656 (2007)
34. Madhavan, J., Bernstein, P., Doan, A., Halevy, A.: Corpus-based schema matching. In: 21st International Conference on Data Engineering (ICDE'05). pp. 57–68 (2005)
35. Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with cupid. In: Proc. of VLDB 2001. pp. 49–58. Morgan Kaufmann (2001)
36. Magnani, M., Rizopoulos, N., Mc.Brien, P., Montesi, D.: Schema integration based on uncertain semantic mappings. In: Conceptual Modeling – ER 2005. pp. 31–46. Springer (2005)
37. Malakasiotis, P., Androutsopoulos, I.: Learning textual entailment using svms and string similarity measures. In: Proc. of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. p. 42–47. Association for Computational Linguistics, USA (2007)
38. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48(3), 443–453 (1970)
39. Nikiforova, O., Gusarovs, K., Gorbiks, O., Pavlova, N.: BrainTool: A tool for generation of the UML class diagrams. In: Proc. of ICSEA 2012. pp. 60–69. IARIA (2012)
40. Omar, N., Hanna, P., McKevitt, P.: Heuristics-based entity-relationship modelling through natural language processing. In: Proc. of AICS 2004. pp. 302–313 (2004)
41. OMG: Unified Modeling Language (OMG UML), v2.5. OMG (2015)
42. Overmyer, S.P., Benoit, L., Owen, R.: Conceptual modeling through linguistic analysis using LIDA. In: Proc. of ICSE 2001. pp. 401–410. IEEE (2001)
43. Pottinger, R.A., Bernstein, P.A.: Merging models based on given correspondences. In: Proceedings 2003 VLDB Conference, pp. 862–873. Morgan Kaufmann, San Francisco (2003), https://www.sciencedirect.com/science/article/pii/B9780127224428500811
44. Purao, S.: APSARA: A tool to automate system design via intelligent pattern retrieval and synthesis. SIGMIS Database 29(4), 45–57 (1998)
45. Rezaei, M., Fränti, P.: Matching similarity for keyword-based clustering. In: Fränti, P. et al. (ed.) Structural, Syntactic, and Statistical Pattern Recognition. pp. 193–202. Springer (2014)
46. Rodrigues, D., da Silva, A., Rodrigues, R., dos Santos, E.: Using active learning techniques for improving database schema matching methods. In: 2015 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2015)
47. Rodriguez, A., Garcia-Rodriguez de Guzman, I., Fernandez-Medina, E., Piattini, M.: Semi-formal transformation of secure business processes into analysis class and use case models: An MDA approach. Information and Software Technology 52(9), 945–971 (2010)
48. Sahay, T., Mehta, A., Jadon, S.: Schema matching using machine learning. In: 2020 7th International Conference on Signal Processing and Integrated Networks (SPIN). pp. 359–366 (2020)
49. Sheetrit, E., Brief, M., Mishaeli, M., Elisha, O.: Rematch: Retrieval enhanced schema matching with llms (2024), https://arxiv.org/abs/2403.01567

50. Shraga, R., Gal, A.: Powarematch: A quality-aware deep learning approach to improve human schema matching. J. Data and Information Quality 14(3) (may 2022), https://doi.org/10.1145/3483423
51. Shraga, R., Gal, A., Roitman, H.: Adnev: cross-domain schema matching using deep similarity matrix adjustment and evaluation. Proc. VLDB Endow. 13(9), 1401–1415 (may 2020), https://doi.org/10.14778/3397230.3397237
52. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
53. Spasic, Z., Vukotic, A., Brdjanin, D., Banjac, D., Banjac, G.: UML-based forward database engineering. In: Proc. of INFOTEH 2023. pp. 1–6. IEEE (2023)
54. Sugumaran, V., Storey, V.C.: Ontologies for conceptual modeling: their creation, use, and management. Data & Knowledge Engineering 42(3), 251–271 (2002)
55. Tan, H.B.K., Yang, Y., Blan, L.: Systematic Transformation of functional analysis model in Object Oriented design and Implementation. IEEE Trans. on Soft. Eng. 32(2), 111–135 (2006)
56. Thonggoom, O.: Semi-automatic conceptual data modelling using entity and relationship instance repositories. PhD Thesis, Drexel University (2011)
57. Unal, O., Afsarmanesh, H.: Using linguistic techniques for schema matching. In: Filipe, J. et al. (ed.) Proc. of ICSOFT 2006. pp. 115–120. INSTICC Press (2006)
58. Winkler, W.E.: String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. (1990), https://eric.ed.gov/?id=ED325505
59. Zhang, J., Shin, B., Choi, J.D., Ho, J.C.: Smat: An attention-based deep learning solution to the automation of schema matching. Symposium on Advances in Databases and Information Systems (2021)
60. Zhang, Y., Di, M., Luo, H., Xu, C., Tsai, R.T.H.: Smutf: Schema matching using generative tags and hybrid features (2024), https://arxiv.org/abs/2402.01685
61. Zhang, Y., Floratou, A., Cahoon, J., Krishnan, S., Müller, A.C., Banda, D., Psallidas, F., Patel, J.M.: Schema matching using pre-trained language models. In: 2023 IEEE 39th International Conference on Data Engineering (ICDE). pp. 1558–1571 (2023)

Goran Banjac is a Senior Teaching Assistant and PhD student at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina). He is a member of the M-lab Research Group. His research interests include model-driven software development, business process modeling, databases, and UML. He has published several research papers and articles.

Drazen Brdjanin is an Associate Professor at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina), where he heads the M-lab Research Group. His research interests focus on information systems and software engineering. He has participated in several national and international R&D projects and also authored a number of research papers and articles in the field of model-driven development.

Danijela Banjac is a Senior Teaching Assistant and PhD student at the Faculty of Electrical Engineering, University of Banja Luka (Bosnia and Herzegovina). She is a member of the M-lab Research Group. Her research interests include model-driven software development, business process modeling, object-oriented information systems, and UML. She has published several research papers and articles.

Received: March 01, 2024; Accepted: August 06, 2024.