Using Signifiers for Data Integration in Rail Automation

Alexander Wurl[1], Andreas Falkner[1], Alois Haselböck[1] and Alexandra Mazak[2,*]
[1]*Siemens AG Österreich, Corporate Technology, Vienna, Austria*
[2]*TU Wien, Business Informatics Group, Austria*

Keywords: Data Integration, Signifier, Data Quality.

Abstract: In Rail Automation, planning future projects requires the integration of business-critical data from heterogeneous data sources. As a consequence, data quality of integrated data is crucial for the optimal utilization of the production capacity. Unfortunately, current integration approaches mostly neglect uncertainties and inconsistencies in the integration process in terms of railway specific data. To tackle these restrictions, we propose a semi-automatic process for data import, where the user resolves ambiguous data classifications. The task of finding the correct data warehouse classification of source values in a proprietary, often semi-structured format is supported by the notion of a *signifier*, which is a natural extension of composite primary keys. In a case study from the domain of asset management in Rail Automation we evaluate that this approach facilitates high-quality data integration while minimizing user interaction.

1 INTRODUCTION

In order to properly plan the utilization of production capacity, e.g., in a Rail Automation factory, information from all business processes and project phases must be taken into account. Sales people scan the market and derive rough estimations of the number of assets (i.e. producible units) of various types (e.g. control units for main signals, shunting signals, distant signals, etc.) which may be ordered in the next few years. The numbers of assets get refined phase by phase, such as bid preparation or order fulfillment. Since these phases are often executed by different departments with different requirements and interests (e.g. rough numbers such as 100 signals for cost estimations in an early planning phase, vs. detailed bill-of-material with sub-components such as different lamps for different signal types for a final installation phase), the same assets are described by different properties (i.e. with - perhaps slightly - different contents) and in different proprietary formats (e.g. spreadsheets or XML files). Apart from the technical challenges of extracting data from such proprietary structures, heterogeneous feature and asset representations hinder the process of mapping and merging information which is crucial for a smooth overall process and for efficient data analytics which aims at optimizing future projects based upon experiences from all phases of previous projects. One solution approach is to use a data warehouse and to map all heterogeneous data sets of the different departments to its unified data schema.

To achieve high data quality in this process, it is important to avoid uncertainties and inconsistencies while integrating data into the data warehouse. Especially if data includes information concerning costs, it is essential to avoid storing duplicate or contradicting information because this may have business-critical effects. Part of the information can be used to identify corresponding data in some way (i.e. used as key), part of it can be seen as relevant values (such as quantities and costs). Only if keys of existing information objects in the data warehouse are comparable to that one of newly added information from heterogeneous data sets, that information can be stored unambiguously and its values are referenced correctly.

Keys are formed from one or many components of the information object and are significant for comparing information of heterogeneous data sets with information stored in the data warehouse. If two of such keys do not match, this is caused by one of two significantly different causes: (i) two objects should have the same key but they slightly differ from each other, and (ii) two objects really have different keys. Using solely heuristic lexicographical algorithms (Cohen et al., 2003) to automatically find proper matches

[*] Alexandra Mazak is affiliated with the CDL-MINT at TU Wien.

Wurl, A., Falkner, A., Haselböck, A. and Mazak, A.
Using Signifiers for Data Integration in Rail Automation.
DOI: 10.5220/0006416401720179
In *Proceedings of the 6th International Conference on Data Science, Technology and Applications (DATA 2017)*, pages 172-179
ISBN: 978-989-758-255-4

does not necessarily succeed to reliably distinguish those two cases because each character of the key might be important and have a deeper meaning, or not. Inappropriate heuristics lead to wrong matching results, which may have major consequences to business. Therefore it is critical to rely on semantics for matching algorithms.

To support this proposition, we adapt the concept of *signifiers* (Langer et al., 2012). A human actor - an expert of the domain - defines a useful combination of properties during the design phase. In the integration process of a new data set to the data warehouse, we use this definition for calculating each information object's key. In case of mismatches between the data warehouse and the new data set during data integration, a manual control either confirms the mismatch or interacts aiming for a match. The signifier serves as a key to uniquely identify an object in the course of normalization of information in the data warehouse.

The main contributions of this paper are: (i) we reveal challenges of integrating heterogeneous data sets into a data warehouse in an industrial context; (ii) we introduce a technique using a signifier to avoid inconsistencies between assets in the data warehouse; (iii) we show how some minimal human interaction can significantly improve the matching result; (iv) we evaluate in a case study how those techniques improve data quality.

2 PROBLEM DEFINITION

We address data integration scenarios where values from highly heterogeneous data sources are merged into a conjoint data warehouse. In more detail, merged features and assets of different data sources inherently share the same content but reveal various overlapping compositions which are caused by alternating amounts of data generated by manual estimations or obtained from the installed base. In order to avoid duplicate and contradicting data in the data warehouse, corresponding information from different data sources needs to be identified. Unfortunately, there are no global IDs available to identify information objects over different data sources. Instead, computing keys from the different data representations in the sources allows to match them with keys which are already stored in the data warehouse. Figure 1 shows the setting and the issues in more detail.

One typical format is spreadsheets (such as Microsoft Excel) where each row represents an information object, e.g., the expected quantity from a planning phase. "Source1" in Figure 1 has five columns where the first three are used to compute a key (by

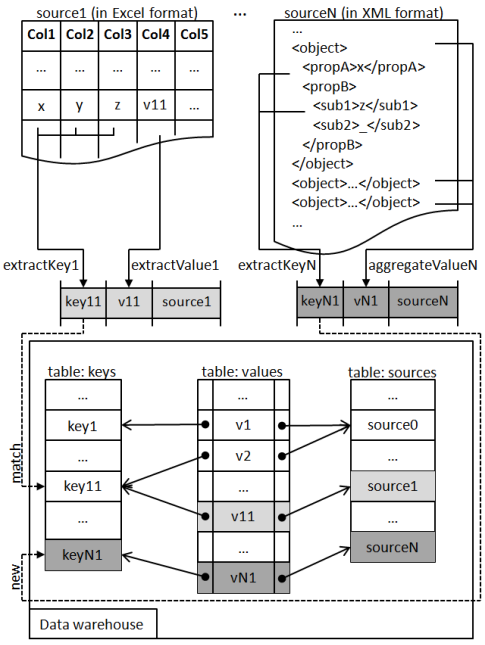

Figure 1: Integration scenario of heterogeneous data sets.

function "extractKey1") and the fourth column (selected by function "extractValue1") contains a relevant value, e.g., the quantity in form of a numerical value.

Another format is XML which contains structured objects (such as representations of physical objects), e.g., a bill of materials from an installation phase. "SourceN" in Figure 1 shows some objects. Each object consists of sub-elements in which all information of the corresponding object is included - either for use in keys (accessed by function "extractKeyN") or for values. In this example, the value is not selected directly, but aggregated from all objects with the same key (using aggregation function "aggregateValueN"), e.g., by counting all those object instances to derive their quantity as a numerical value.

The essence of each data source can be seen as a set of triples, containing a key, a value, and the source identifier. As an intermediate representation, those triples are merged into the data warehouse which comprises the history of all values with references to data sources and keys. The keys are necessary to map different data representations of the same information from different data sources onto each other. If a key from a triple matches an existing key, the latter is reused, e.g., "key11" for the triple from "source1" in Figure 1. Else, a new key is added to the data warehouse, e.g., "keyN1" for the triple from "sourceN". This means that the new information does not correspond to other information in the data warehouse.

Such a scenario poses the following questions:

- Can a simple approach, such as to assemble some properties as components for a unique identification key, cover many use cases?
- High heterogeneity in technical aspects, data model, and semantics requires an advanced approach. How can the extraction functions (e.g., "extractKey1", "extractValue1", "extractKeyN") be defined in a systematic way?
- How shall a match be defined in detail? Perfect match vs. near match (case-sensitivity, lexicographical distance)? How to avoid wrong matches?
- How to decide whether (syntactically) not matching keys refer to the same information? Are synonyms used?
- The process of comparing keys needs some user interaction (expert knowledge). What is the best process? How to minimize efforts?

3 RELATED WORK

Data cleansing, also known as data cleaning, is an inevitable prerequisite to achieve data quality in the course of an ETL-process (Bleiholder and Naumann, 2009). Naumann describes data cleansing as use case of data profiling to detect and monitor inconsistencies (Naumann, 2014). Resolving inconsistencies as part of the transformation phase has been a topic for the last two decades (Leser and Naumann, 2007; Sharma and Jain, 2014).

In the work of (Rahm and Do, 2000; Naumann, 2014) tasks for ensuring data quality are classified; various data cleaning techniques have been proposed such as rule-based (Dallachiesa et al., 2013; Fan and Geerts, 2012), outlier detection (Dasu and Johnson, 2003; Hellerstein, 2008), missing values (Liu et al., 2015), and duplicate detection (Bilenko and Mooney, 2003; Wang et al., 2012). Most of these techniques require human involvement.

The work of (Müller and Freytag, 2005; Krishnan et al., 2016) points out that integrating data is an iterative process with user interaction. Various approaches take this into consideration. Frameworks proposed in (Fan et al., 2010; Khayyat et al., 2015) enable the user to edit rules, master data, and to confirm the calculations leading to correct cleaning results. A higher detection accuracy in duplicate detection by a hybrid human-machine approach is achieved in the work of (Wang et al., 2012). As presented in (Liu et al., 2015), numerous techniques are used to associate the data to get useful knowledge for data repairing, e.g., calculating similarities of contextual and linguistic matches being able to determine relationships. In (Volkovs et al., 2014) a logistic regression classifier learns from past repair preferences and predicts the type of repair needed to resolve an inconsistency.

As (Dai et al., 2016) reports that - although there are various data profiling tools to improve data quality - if people use them without having in mind a clear quality measurement method according to their needs, they are challenged by limited performance and by unexpectedly weak robustness. (Gill and Singh, 2014) claims that various frameworks offer the integration of heterogeneous data sets but a framework for quality issues such as naming conflicts, structural conflicts, missing values, changing dimensions has not been implemented in a tool at one place yet.

The work of (Gottesheim et al., 2011) analyzes the representation of real-world objects in the context of ontology-driven situations. Similar to how real-world objects are characterized by attributes, in (Langer et al., 2012) the characteristics of models are described by a *signifier*. Basically, the concept of a signifier has its origin in the domain of model-driven engineering (MDE) where a signifier enhances the versioning system by describing the combination of features of model element types that convey the superior meaning of its instances. A signifier improves versioning phases in comparing and merging models leading to a higher quality of finally merged models. As we integrate objects from different sources, a signifier structures the combination of properties and finally improves data integration when objects to be integrated are compared and merged with objects in the data warehouse. Similarly, the work of (Papadakis et al., 2015) addresses real-world entities with blocking approaches based on schema-agnostic and schema-based configurations. A schema-based approach may be an alternative to signifiers but with precision limitations in terms of a superior number of detected duplicates when comparing properties. On the other hand, a schema-agnostic approach may skip differing important information of real-world objects when clustering similar properties. In the integration process of objects, signifiers (1) provide a careful and flexible identification structure for properties, and (2) support normalization of information in the data warehouse.

4 USING SIGNIFIERS FOR DATA INTEGRATION

We propose a strategic technique in the ETL-process to instantly react on potential inconsistencies, e.g.,

when there is not a perfect match of a source object with an object in the data warehouse. Instead of names or IDs, we use and extend the concept of *signifiers*, as introduced in (Langer et al., 2012), for mapping an object of a data source to the right object type in the data target (the data warehouse). In simple terms, a signifier is an object consisting of different components. To check if two objects match, the components of their signifiers are checked pairwise.

To cope with the usage of different wordings or words in different languages in the data source, we need two versions of signifiers: a source signifier and a target signifier which is extended by aliases.

Definition. *A **Source Signifier** is an n-tuple of strings. The term S_i refers to the i^{th} component of a source signifier S.*

The meaning of each element of the signifier is determined by its position in the tuple. In the railway asset management example described in section 1, we use signifiers of length 3, representing category, subcategory and subsubcategory, respectively. Example: (”Signal”, ”Main Signal”, ”8 Lamps”).

Definition. *A **Target Signifier** is an n-tuple of sets of strings. The term T_i refers to the i^{th} component of a target signifier T.*

Target signifiers allow to specify more than one string per component. These strings represent aliases. Example: ({”Signal”, ”S”}, {”Main Signal”, ”MainSig”, ”Hauptsignal”, ”HS”}, {”8”, ”8 Lamps”, ”8lamps”})

The main task of integrating a new object from a data source into a target data warehouse is to match source and target signifiers. To be able to deal with approximate matches, too, we use a distance function $dist(s,t)$, returning a value from $[0,1]$, where 0 means that the two strings s and t are equal. There are many well-studied string metrics that can be used here – see, e.g., (Cohen et al., 2003). Given a string distance function $dist(.,.)$, we define the minimum distance of a string s and a set of strings ts by: $dist_{min}(s,ts) = \min_{i=1..|ts|} dist(s,ts_i)$.

In order to express different significances of different components of a signifier, we use weight factors w_i for each component i. The total sum of all weighted components of a signifier is 1. In Section 5, the weighting of the components is demonstrated.

Definition. *Let S be a source signifier and T be a target signifier with n components. Let $dist(.,.)$ be a string distance function. Let w_i be component weights. The function **Dist(S,T)** returns a value from $[0,1]$ and is defined in the following way:*

$$Dist(S,T) = \sum_{i=1..n} dist_{min}(S_i,T_i) * w_i$$

Now we are in the position to formally define perfect and approximate matches.

Definition. *Let S be a source signifier and T be a target signifier. S and T **perfectly match**, if and only if $Dist(S,T) = 0$. For a given threshold value τ ($0 < \tau < 1$), S and T **approximately match**, if and only if $0 < Dist(S,T) \leq \tau$.*

The algorithm of integrating a source data object into the target database (data warehouse) is a semi-interactive task based on the previously defined perfect and approximate matches. If a perfect match is found, the new object is automatically assigned to the target data warehouse. In all other cases, the user is asked to decide what to do. The following steps summarize this algorithm for adding a source object with signifier S to a target database with existing target signifiers TS:

1. If we find a $T \in TS$ that is a perfect match to S, add the new object to the target database using T. Done.
2. Let TS_{approx} be the (possibly empty) set of target signifiers that are approximate matches to S. Ask the user for a decision with the following options:
   (a) Accept one $T \in TS_{approx}$ as match. The new object will be added to the target database using T. The aliases of T are updated in the target database by adding all components of S, that do not fully match, to the aliases in T.
   (b) Accept S as a new target signifier. The new object will be added to the database and S will be added to the target signifiers.
   (c) Abort import process and fix the source database.

Each import of a data source potentially increases and completes the number of target signifiers or accordingly the aliases of target signifiers. In that, user interaction will decrease over time and the import of source data will get closer and closer to run fully automatically.

5 CASE STUDY

In this section, we perform an empirical case study based on the guidelines introduced in (Runeson and Höst, 2009). The main goal is to evaluate if the approach using signifiers for the integration of data from heterogeneous data sets into the data warehouse significantly improves data quality, i.e., reduces incorrect classifications of objects imported into a given data warehouse. Especially in the domain of rail automation with business critical data it is crucial to

avoid incorrect classification of data. We conducted the case study for the business unit Rail Automation, in particular for importing planning, order calculation and configuration data of railway systems into an asset management repository.

5.1 Research Questions

Q1: Can signifiers, as described in Section 4, minimize the incorrect classification of objects at data import?
Q2: Can our import process based on signifiers minimize user interactions?
Q3: From an implementation point of view, how easily can conventional (composite) primary keys of objects in a data warehouse be extended to signifiers?

5.2 Case Study Design

Requirements. We perform our empirical tests in an asset management scenario for Rail Automation. A data warehouse should collect the amount of all assets (i.e., hardware modules and devices) of all projects and stations that are installed, currently engineered, or planned for future projects. To populate this data warehouse, available documents comprising planned and installed systems should be imported. Planning and proposal data, available in Excel format, can be classified as semi-structured input: While the column structuring of the tables is quite stable, the names of the different assets often differ because of the usage of different wording, languages, and formatting. The tables also contain typing errors, because they are filled out manually as plain text. Configuration data for already installed systems is available in XML format and is therefore well-structured. As the underlying XML schema changed over time due to different engineering tool versions, also XML data contain structural variability.

The exemplary import of a couple of different Excel files, including files from projects of different countries, and one XML file should be performed. The test should start with an empty data warehouse, i.e., no target signifiers are known at the beginning; the set of appropriate target signifiers should be built up during import of the different files.
Setup. The software architecture for our implementation of the ETL process for the case study is sketched in Fig. 2. We use a standard ETL process as, e.g., described in (Naumann, 2014). The extraction phase is implemented with KNIME[2].

The result of the extraction phase is a dataset containing source signifiers and corresponding data

[2]https://www.knime.org/

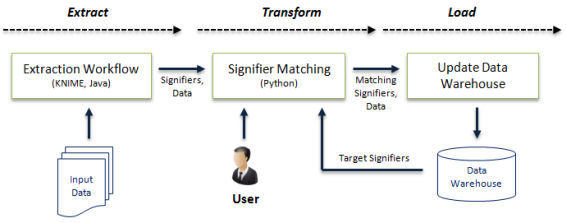

Figure 2: Case study ETL process.

values. Our Signifier Matching component, implemented in Python, tries to find for each entry in that dataset a matching target signifier already stored in the data warehouse. Ambiguities are resolved by asking the user. The load phase consists of updates to the data warehouse: input data are added with their references to a target signifier and a source descriptor. New target signifiers and new aliases of existing target signifiers are added, as well.

Using our method, the following application-specific parameters must be adjusted: the number of signifier components, the string distance metrics, the weights of the signifier components, and the threshold value for approximate matches. In our tests, we used signifiers consisting of 3 components, representing category, subcategory and subsubcategory of a data object. For string comparison we used a case-insensitive Jaro-Winkler distance (Cohen et al., 2003). Weights and threshold are determined as shown in the next section, specifically cf. Table 4.

5.3 Results

In this section, we present the results of our case study from data and behavioral perspectives. Our main goal was to analyze the applicability of signifiers in the transform phase of the ETL process with respect to data quality and amount of necessary user decisions.
Data Integration with Signifiers. We demonstrate our approach by describing the key steps of data integration of an Excel source and an XML source on a concrete example from the Rail Automation domain. We are sketching the main database tables according to Figure 1.

Target signifiers are represented in a table as depicted in Table 1. Each row represents an asset type with a primary key, ID, for being referenced from data tables. The other columns represent the components of the target signifiers. Please note that such components typically contain not only a single string but a set of strings (i.e., aliases) for different wordings or different languages.

Figure 3 shows a small part of an input table (in German language). The extraction of the first row results in the source signifier `[Signale,`

Table 1: Target signifier table of the data warehouse.

ID	Category	SubCat.	SubSubCat.
1	Signal	Main Signal, Hauptsignal, HS	4 lamps, 4
2	Signal	Shunting Signal, Rangiersignal, RS	2 lamps, 2
...	...	...	...

`Hauptsignal, 4 Lampen]` and value 10 representing the amount of main signals with 4 lamps in the railway station. Obviously, this signifier has no perfect match in the target signifiers as shown in Table 1, because German language and some extended wording is used in the source Excel file. The matching distances according to our settings are 0.05 for the first and 0.182 for the second target signifier. The first one has a clearly lower distance value. After the user has confirmed that this is the right match, firstly, the target signifier table is updated by adding aliases "Signale" (for category) and "4 Lampen" (for subsubcategory) to signifier 1, and secondly, the object value 10 is added to the data table (see first row in Table 2) with links to the right target signifier and its source. The other rows are extracted in the same way (not shown in the table).

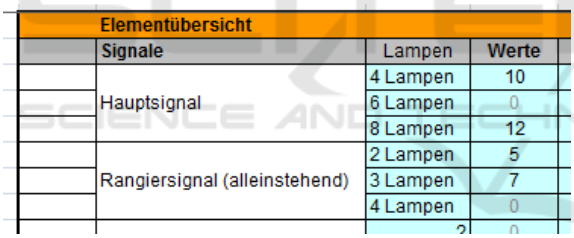

Figure 3: An excerpt of objects in an Excel spreadsheet.

For the XML source document depicted in Figure 4, the signifier components for the first object are built in the following way: The first component is the tag name of the XML element, `signal`, the second one its signal type, `HS`, and the third one is created by counting the lamp sub-elements. Now we get the source signifier `[signal, HS, 4]`. Since our distant metrics are case-insensitive, we have a perfect match with target signifier 1 in the data warehouse. The data value for this kind of object is computed by counting the number of objects of this type in the XML file. After processing the second XML element in a similar way, the data table is updated. Table 2 shows the resulting two entries at the end.

Results in Data Quality and User Interactions. We performed our tests on several data sources in Excel and XML format, starting with an empty data warehouse without any target signifiers. In total about 90

```
<DATA-FORMAT schema-version="23">
  <signal id="4463">
    <Name type="string" state="30">A</Name>
    <signaltype type="string" state="40">HS</signaltype>
    ...
    <lamps type="assoc" state="30">
      <_REF>4362</_REF>
      <_REF>4361</_REF>
      <_REF>4363</_REF>
      <_REF>4364</_REF>
    </lamps>
  </signal>
  <signal id="4462">
    <Name type="string" state="30">R</Name>
    <signaltype type="string" state="40">RS</signaltype>
    ...
    <lamps type="assoc" state="30">
      <_REF>4112</_REF>
      <_REF>4113</_REF>
    </lamps>
  </signal>
</DATA-FORMAT>
```

Figure 4: An XML data source containing object elements.

Table 2: Data objects table after import of values from an Excel and an XML source.

Signifier	Value	Source
1	10	Source 1 (Excel)
...	...	...
1	1	Source 2 (XML)
2	1	Source 2 (XML)

signifiers were created; 10% of these signifiers got aliases. The different sources partly contained different wordings, languages, and abbreviations for object designation, and contained typing errors. Qualitatively speaking, for the most of these different wordings, our signifier matcher found approximate matches and could very specifically ask the user for a match decision. In case of no perfect match could be found, the number of provided options was mostly in the range of 3. A few signifiers had name variations which fell out of the class of approximate matches, and the user had to match them manually. Provided that incorrect object classifications should be avoided, the number of user decisions was minimal.

Table 3: The correlation of match candidates and expert reference (correct match).

	correct match	*no*
match candidate	tp	fp
no	fn	tn

For a quantitative assessment of the quality of matches we use the F-measure (more specifically, the F_2-measure) from the field of information retrieval (Salton and Harman, 2003; Wimmer and Langer, 2013). This measure is based on the notion

of *precision* and *recall*, which are defined in terms of true/false positives/negatives. Table 3 shows how true/false positives/negatives are defined by comparing the correct value as defined by an expert with the match candidates computed by signifier matching (perfect or approximate matches). Precision, recall and the F_2-measure are defined as follows: $P = |tp|/(|tp|+|fp|)$, $R = |tp|/(|tp|+|fn|)$, $F_2 = 5 \times P \times R/(4 \times P + R)$. We use the F_2-measure here to emphasize the recall value; it is important in our application that we do not miss any correct matches.

Table 4 shows the results of our tests by precision, recall and F_2 values. We varied the threshold and weight values to find an optimal combination. We used threshold values $\tau1 = 0.01$, $\tau2 = 0.025$, $\tau3 = 0.05$, $\tau4 = 0.1$; we used weight values $w1 = (\frac{1}{14}, \frac{4}{14}, \frac{9}{14})$, $w2 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, $w3 = (\frac{9}{14}, \frac{4}{14}, \frac{1}{14})$. It was observed that:

- As expected, small threshold values lead to high precision values, large threshold values lead to high recall values. Precision and recall are negatively correlated.
- For the set of used test data a value combination $\tau = 0.025$ and $w = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ achieves the best matching results, i.e., has the highest F_2-measure value.
- The F_2 value of many threshold/weight combinations are - in absolute terms - quite high, indicating that the proposed method achieves high-quality matches and is quite robust against small changes of input parameters.

Table 4: Precision, recall and F_2 values for signifier matching tests varying approximate match threshold (τ) and signifier component weights (w).

	$\tau1 = 0.00$	$\tau2 = 0.025$	$\tau3 = 0.05$	$\tau4 = 0.1$
P	$w1$: 1.000 $w2$: 1.000 $w3$: 1.000	$w1$: 0.891 $w2$: 0.930 $w3$: 0.901	$w1$: 0.779 $w2$: 0.887 $w3$: 0.793	$w1$: 0.710 $w2$: 0.835 $w3$: 0.432
R	$w1$: 0.895 $w2$: 0.895 $w3$: 0.895	$w1$: 0.950 $w2$: 0.950 $w3$: 0.950	$w1$: 0.956 $w2$: 0.950 $w3$: 0.950	$w1$: 0.961 $w2$: 0.950 $w3$: 0.956
F_2	$w1$: 0.914 $w2$: 0.914 $w3$: 0.914	$w1$: 0.938 $w2$: **0.946** $w3$: 0.940	$w1$: 0.914 $w2$: 0.937 $w3$: 0.914	$w1$: 0.898 $w2$: 0.925 $w3$: 0.770

5.4 Interpretation of Results

We analyze the results with regard to our research questions.

Q1: *Can signifiers minimize the incorrect classification of objects?* Yes, according to the test results shown in Table 4, our method is capable to achieve high values of precision and recall values. Compared to the "perfect match only" scenario ($\tau = 0$) with its perfect precision, approximate matches showed a weaker precision but a much better recall and therefore a better F-measure.

Q2: *Can our import process based on signifiers minimize user interactions?* Yes, provided that automated matches should only be made based on a perfect match, the number of user interactions were minimal in the sense that the user was not asked for a similar match twice. Furthermore, the list presented to the user for a manual match was sorted by match distance; in most cases the user found his/her match on the first or second place in that list.

Q3: *From an implementation point of view, how easily can conventional (composite) primary keys of objects in a data warehouse be extended to signifiers?* Instead of using data tables with composite keys in the data warehouse, signifiers are stored in a separate table with a generated key for referencing from data tables. Design and implementation of this separate signifier table is straight-forward.

5.5 Threats to Validity

The first import of a data source where all signifier components were translated to another language produces a lot of user interaction. This could be avoided by generating and adding translations as aliases to the signifiers in the data warehouse.

In situations where a signifier component represents a number (e.g., number of lamps of a railway signal), plain string distance is not an optimal choice. E.g., a human would consider "2" closer to "3" than to "5", which is usually not the case in a string distance function. Our definition of signifiers as a tuple of string components could be extended to components of different types. This would allow the implementation of type-specific distance functions and solve that problem.

6 CONCLUSION AND FUTURE WORK

In this paper, we identified various issues in the integration process of business-critical data from heterogeneous data sources. To address these issues, we proposed a semi-interactive approach. We introduced a technique using the notion of a *signifier* which is a natural extension of composite primary keys to support the user resolving ambiguous data classification. In a case study, we validated the applicability of our approach in the industrial environment of Rail Automation. The results show a significant improvement of data quality.

There are several ideas for future work. One relates to extending the textual representation of com-

ponent types with numerical values. This affects the storage of values in the data warehouse as well as the algorithm that compares values. Another direction is to strive for a joint dictionary bridging language-specific component terms. This accelerates the integration process especially in companies with international respect.

ACKNOWLEDGEMENTS

This work is funded by the Austrian Research Promotion Agency (FFG) under grant 852658 (CODA). We thank Walter Obenaus (Siemens Rail Automation) for supplying us with test data.

REFERENCES

Bilenko, M. and Mooney, R. J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD*, pages 39–48. ACM.

Bleiholder, J. and Naumann, F. (2009). Data fusion. *ACM Computing Surveys (CSUR)*, 41(1):1.

Cohen, W. W., Ravikumar, P., and Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03, August 9-10, 2003, Acapulco, Mexico*, pages 73–78.

Dai, W., Wardlaw, I., Cui, Y., Mehdi, K., Li, Y., and Long, J. (2016). Data profiling technology of data governance regarding big data: Review and rethinking. In *Information Technology: New Generations*, pages 439–450. Springer.

Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I. F., Ouzzani, M., and Tang, N. (2013). Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD*, pages 541–552. ACM.

Dasu, T. and Johnson, T. (2003). Exploratory data mining and data cleaning: An overview. *Exploratory data mining and data cleaning*, pages 1–16.

Fan, W. and Geerts, F. (2012). Foundations of data quality management. *Synthesis Lectures on Data Management*, 4(5):1–217.

Fan, W., Li, J., Ma, S., Tang, N., and Yu, W. (2010). Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment*, 3(1-2):173–184.

Gill, R. and Singh, J. (2014). A review of contemporary data quality issues in data warehouse etl environment. *Journal on Today's Ideas - Tomorrow's Technologies*.

Gottesheim, W., Mitsch, S., Retschitzegger, W., Schwinger, W., and Baumgartner, N. (2011). Semgentowards a semantic data generator for benchmarking duplicate detectors. In *DASFAA*, pages 490–501. Springer.

Hellerstein, J. M. (2008). Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*.

Khayyat, Z., Ilyas, I. F., Jindal, A., Madden, S., Ouzzani, M., Papotti, P., Quiané-Ruiz, J.-A., Tang, N., and Yin, S. (2015). Bigdansing: A system for big data cleansing. In *Proceedings of the 2015 ACM SIGMOD*, pages 1215–1230. ACM.

Krishnan, S., Haas, D., Franklin, M. J., and Wu, E. (2016). Towards reliable interactive data cleaning: a user survey and recommendations. In *HILDA@ SIGMOD*, page 9.

Langer, P., Wimmer, M., Gray, J., Kappel, G., and Vallecillo, A. (2012). Language-specific model versioning based on signifiers. *Journal of Object Technology*, 11(3):4–1.

Leser, U. and Naumann, F. (2007). *Informationsintegration - Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. dpunkt.verlag.

Liu, H., Kumar, T. A., and Thomas, J. P. (2015). Cleaning framework for big data-object identification and linkage. In *2015 IEEE International Congress on Big Data*, pages 215–221. IEEE.

Müller, H. and Freytag, J.-C. (2005). *Problems, methods, and challenges in comprehensive data cleansing*. Professoren des Inst. für Informatik.

Naumann, F. (2014). Data profiling revisited. *ACM SIGMOD Record*, 42(4):40–49.

Papadakis, G., Alexiou, G., Papastefanatos, G., and Koutrika, G. (2015). Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *Proceedings of the VLDB Endowment*, 9(4):312–323.

Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131.

Salton, G. and Harman, D. (2003). *Information retrieval*. John Wiley and Sons Ltd.

Sharma, S. and Jain, R. (2014). Modeling etl process for data warehouse: an exploratory study. In *In ACCT, 2014 Fourth International Conference on*, pages 271–276. IEEE.

Volkovs, M., Chiang, F., Szlichta, J., and Miller, R. J. (2014). Continuous data cleaning. In *2014 IEEE 30th ICDE*, pages 244–255. IEEE.

Wang, J., Kraska, T., Franklin, M. J., and Feng, J. (2012). Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494.

Wimmer, M. and Langer, P. (2013). A benchmark for model matching systems: The heterogeneous metamodel case. *Softwaretechnik-Trends*, 33(2).