Computers in Industry 135 (2022) 103571

Contents lists available at ScienceDirect

Computers in Industry

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

A framework for merging ontologies in the context of smart factories

Felix Ocker[a,*], Birgit Vogel-Heuser[a,b], Christiaan J.J. Paredis[c]

[a] Institute of Automation and Information Systems, TUM School of Engineering and Design, Technical University of Munich, Boltzmannstr. 15, Garching, Germany
[b] Core Member of TUM's MDSI and Member of TUM's MIRMI
[c] BMW Chair in Systems Integration, Clemson University, 4 Research Drive, Greenville, USA

ARTICLE INFO

Article history:
Received 24 June 2021
Received in revised form 8 October 2021
Accepted 7 November 2021
Available online 9 December 2021

Keywords:
Ontology merging
Knowledge representation
Smart factories

ABSTRACT

Current trends, such as individualization, increasing complexity, and specialization, require digitization in engineering and production. However, digitization by itself often leads to so-called data silos, which cannot be leveraged effectively when designing and operating smart factories due to the heterogeneity of the information available. This paper presents a framework for (semi-)automatically merging the highly reusable terminological components of production ontologies in an a posteriori way. The framework combines translations, domain-specific vocabularies, and inconsistency checks with syntactic, terminological, and structural analyses to integrate knowledge representations formalized in the Web Ontology Language. Integrating heterogeneous knowledge representations in the production domain can improve support systems for engineers, increase awareness of interdependencies, and enable unambiguous communication in smart factories.

1. Motivation

Engineering and production have to cope with demands regarding individualization and short times to market. At the same time, the complexity of products and production resources increases, requiring a high degree of specialization and cooperation among domain specialists. Advances in digitization, computational power, and Knowledge Representation (KR) provide a sound basis for support systems. For both engineering and production, the applications of ontologies are manifold, ranging from inconsistency management (Feldmann et al., 2015) to Multi Agent Systems (MASs) initialization (Ocker et al., 2019a). Here, a common understanding of key notions, e.g., production processes, is crucial to avoid falsely identified inconsistencies and false negatives during the matchmaking of agents.

Even though the development of KR technologies is promising, these applications are mostly stand-alone endeavors. The semantic interoperability of information is still a major challenge involving several causes. The increasing complexity of production systems and advances in the technologies used require increasing specialization of the engineers. Their different backgrounds and ways of thinking in combination with time pressure lead to parallel development of heterogeneous KRs. In addition, companies may standardize how information is represented, but these standards are usually not shared across companies. However, all these KRs are still strongly interdependent because they describe the same systems.

There are two different approaches to coping with heterogeneous KRs. A priori approaches aim to standardize KRs, but require universal adoption. This may result in tremendous effort in the case of legacy systems. In contrast, a posteriori approaches aim to combine existing KRs. Hence, a posteriori approaches may also reduce the effort needed for legacy KRs to migrate. In the case of ontologies, Terminological Components (TBoxes) formalize the classes and properties used to describe the engineers' domains of interest. Merging the ontologies' reusable parts is the foundation for merging the Assertional Components (ABoxes), i.e., the instance level, and supports a more holistic design process and interoperability within smart factories.

A framework for ontology merging should fulfill the following aims, denoted *(Ax)*. The primary function is to merge heterogeneous ontologies *(A1)*. This comprises heterogeneity regarding the ontologies' domains of interest *(A1.1)* and their degrees of axiomatization *(A1.2)*. The matching quality of the automated process must be sufficient to provide a benefit to engineers *(A2)*. Additionally, the framework should be modularly designed *(A3)*, ensuring the reversibility of the merging process. Finally, the framework should be adaptable *(A4)*. This allows engineers to fine-tune the merging process according to their specific needs. The remainder of this paper is structured as follows. Section 2 gives an overview of

* Corresponding author.
E-mail address: felix.ocker@tum.de (F. Ocker).

https://doi.org/10.1016/j.compind.2021.103571
0166-3615/

selected production ontologies and ontology merging approaches. Next, we describe two application examples highlighting the challenges of merging ontologies. Section 4 presents Production Ontology Merging Framework (PrOM), while Section 5 describes implementation details and discusses results. The paper concludes with a summary and an outlook.

2. Related Work

2.1. Ontologies for Smart Factories

Knowledge extensive domains require methods to manage increasing complexity and support interdisciplinary cooperation. Ontologies provide an "explicit specification of a conceptualization" (Gruber, 1993) that is "formal and shared" across perspectives (Studer et al., 1998). Being formal, ontologies allow reasoning, and support interoperability as they build on the Open World Assumption (OWA) (Atkinson et al., 2006). Ontologies can be described using the Web Ontology Language (OWL). OWL 2 Description Logic (DL) corresponds to $\mathcal{SROIQ}(\mathcal{D})$ (Hitzler et al., 2010) and provides "maximum expressiveness without losing computational completeness" (Smith et al., 2004). While an ontology's ABox includes individuals, its TBox comprises the classes and properties.

Engineering and production are affected by increasing complexity and specialization. Accordingly, researchers have developed various ontology-based applications. For instance, inconsistency management approaches (Feldmann et al., 2015; Herzig et al., 2011) address interdependencies between different viewpoints involved in technical system designs. Similarly, feasibility feedback approaches (Ocker et al., 2019b) identify potential conflicts to reduce late, costly changes. Specifications can also be compared with capabilities to identify suitable suppliers (Ameri and McArthur, 2014). Further applications of production ontologies include skill descriptions (Köcher et al., 2020), service matchmaking (Zhao et al., 2017), management of production processes (Puttonen et al., 2013), and the initialization of MASs (Ocker et al., 2019a).

This variety of applications shows the usefulness of ontologies for smart factories. However, these ontologies are mostly stand-alone solutions despite their overlap. For instance, ontologies created for providing feasibility feedback could also be leveraged to create MASs and enable their automated decision making. Hence, ontology merging has the potential to facilitate ontology creation, support cooperation, and increase semantic interoperability.

2.2. Ontology Merging Approaches

Stakeholders with different viewpoints cooperating leads to heterogeneity in KRs (Euzenat and Shvaiko, 2013). The KRs may differ in syntax, terminology, concepts, and semiotics. A priori approaches, especially standardization and Top Level Ontologies (TLOs) such as Basic Formal Ontology (BFO) (Arp et al., 2015), aim to support KR combination from their creation on. Their impact is limited either by their complexity and overhead or their focus, though, reducing their applicability. This is indicated by the variety of TLOs available, which require alignments themselves (Schmidt et al., 2019), and the fact that aligning domain ontologies to TLOs is still challenging (Stevens et al., 2019). Hence, there is a need for a posteriori approaches, which merge ontologies after their creation. These range from taxonomy matching (Maedche and Staab, 2002) to the matching of full-fledged axiomatized ontologies (Euzenat and Shvaiko, 2013). Ontology merging is "the creation of a new ontology from two, possibly overlapping, input ontologies" (Euzenat and Shvaiko, 2013), whereby the original ontologies remain unchanged, and the resulting ontology contains the knowledge of both inputs (Euzenat and Shvaiko, 2013). The matching process is key to ontology merging and aims to align two input ontologies (Euzenat and Shvaiko, 2013), cp. Equation (1).

$$A^* = f(o_1, o_2, A, p, r) \tag{1}$$

where:

A^* = resulting alignment

o_i = input ontology i

A = input alignment

p = set of parameters

r = set of oracles and resources

An alignment is a set of correspondences c, each describing the relation between two entities (Euzenat and Shvaiko, 2013), cp. Equation (2).

$$\begin{aligned} &c = (e_1, e_2, r) \\ &s.\,t. \\ &e_1 \in Q_{L1}(o_1),\ \ e_2 \in Q_{L2}(o_2),\ \ r \in \Theta \end{aligned} \tag{2}$$

where:

e_i = element from ontology i

r = relation holding between e_1 and e_2

Q_{Li} = entity language of ontology i

Θ = set of alignment relations

To create alignments, engineers can apply several techniques. These include terminological analyses based on string and term comparisons, structural analyses that leverage relations between notions, extensional analysis, which compares the instances of classes, and semantic analyses based on external ontologies (Ardjani et al., 2015). So far, established tools like the ontology editor Protégé do not support automated ontology merging (Pawełoszek and Korczak, 2018).

Most ontology merging approaches rely on terminological analyses, often leveraging the lexical database WordNet (Miller, 1995). WordNet can also be combined with string analyses (Chatterjee et al., 2018). Various frameworks combine terminological and structural analyses, e.g., COMA++ (Aumueller et al., 2005), PROMPT (Noy and Musen, 2000), and AgreementMaker (Cruz et al., 2009). FALCON-AO (Hu and Qu, 2008) relies on linguistic analyses via virtual documents and a structure matcher (Hu et al., 2005) which iteratively increases thresholds. Risk Minimization based Ontology Mapping (RiMOM) (Tang et al., 2006) employs Bayesian reasoning, while S-Match (Giunchiglia et al., 2012) uses semantic matching and rephrases ontology matching as a satisfiability problem. Stumme and Maedche (2001) presented an example for extensional matching, relying on linguistic processing of domain-specific documents. Further frameworks use fuzzy Formal Concept Analysis (FCA) (Chen et al., 2011), heuristic functions (Robin and Uma, 2010), or non-deterministic approaches (Lv and Peng, 2020). Furthermore,

Table 1
Comparison of selected ontology merging frameworks (+ *supported*; o *partially supported*; - *not supported*).

	Translations	Syntactic and terminological analysis	Structural analysis	Extensional analysis	Semantic analysis	Consistency checks	Domain-specific information
Agreement Maker (Cruz et al., 2009)	–	+	+	–	–	o	–
FCA-MERGE (Stumme and Maedche, 2001)	–	+	–	+	–	–	+
OECM (Ibrahim et al., 2019)	+	+	–	–	–	+	–
Ontology mapping method (Kumar and Harding, 2013)	–	+	o	–	o	+	–
PROMPT (Noy and Musen, 2000)	–	+	+	–	o	+	o
S-Match (Giunchiglia et al., 2012)	+	+	–	–	+	o	–
PrOM	+	+	+	–	o	+	+

there is research regarding multilingual ontology matching (Fu et al., 2012; Ibrahim et al., 2019). Since coverage of automated translations is lower for domain-specific notions, manually mapped wordnets are helpful (Helou et al., 2016). Detailed comparisons of generic ontology merging approaches are provided by Ardjani et al. (2015), Shvaiko and Euzenat (2013), and by the Ontology Alignment Evaluation Initiative (OAEI) (Algergawy et al., 2019). Note that the OAEI does not include ontologies from the engineering domain.

2.3. Semantic Interoperability in the Context of Smart Factories

The multitude of heterogeneous metamodels in production systems engineering shows the need for KR, but interoperability is still a challenge (Cha et al., 2020). Domain-specific a priori approaches include information models, e.g., the one provided by the OPC Unified Architecture (OPC Foundation, 2017). Furthermore, there are metamodels for Digital Twins (Plattform Industrie 4.0, 2020) and data exchange, e.g., AutomationML (AutomationML Consortium, 2014), and ontologies, e.g., for integrating design and manufacturing in an a priori way (Chungoora et al., 2013). A set of well-formed production ontologies based on BFO is provided by the Industrial Ontologies Foundry (IOF) (Industrial Ontologies Foundry, 2020). Also pursuing a modular approach, Hildebrandt et al. (2020) formalized standards and aligned them manually. Similarly, Witherell et al. (2013) derived an "upper-tier ontology" focusing on product knowledge. These standardization approaches are valuable as a reference, but their success requires adoption by all stakeholders. Thus, some authors have suggested a posteriori approaches for ensuring semantic interoperability. Kumar and Harding (2013) presented an approach for ontology mapping that leverages synonyms and axioms. Even though the lexical analysis relies on WordNet, they do not mention advanced Natural Language Processing (NLP) techniques such as Part of Speech (POS) tagging or lemmatization. Anjum et al. (2012) used a TLO to combine domain ontologies from design and production, but this requires aligning the input ontologies with a common TLO. Adamczyk et al. (2020) presented a process for ensuring semantic interoperability using reference ontologies. However, they rely on manually created rules for semantic reconciliation. Creating a virtual single underlying metamodel via synchronization transformations (Kramer et al., 2013) seems promising, but has not yet been realized for ontologies. TRAILS (Wolfenstetter et al., 2018) leverages customized model mappers to support traceability, integration, and visualization of knowledge in Product Service Systems.

2.4. Research Gap

Although valuable for designing and operating smart factories, available ontologies are very heterogeneous despite their interdependecies. Engineers may leverage partial overlaps to reduce the effort required to create ontologies and improve their semantic interoperability. A priori approaches for ontology merging are helpful, but not easily applicable if there are legacy systems to be combined. Most approaches to improving semantic interoperability in the context of smart factories fall into this category. In contrast, a posteriori approaches merge ontologies after their creation. There exist advanced generic frameworks and a few domain-specific approaches, but ontology merging is still an open issue for the production domain. Here, the heterogeneity in content and axiomatization, varying overlaps between ontologies, domain-specific terminology, and different languages in use are especially challenging. The production domain can benefit from combining syntactic, terminological, and structural analysis with domain-specific information. Table 1 gives an overview of selected frameworks regarding matching techniques (Euzenat and Shvaiko, 2013), translations, consistency checks, and the use of domain-specific information. It is based on the original papers as well as a comprehensive survey (Ardjani et al., 2015), and indicates the need for an integrated ontology merging pipeline tailored to the production domain. PrOM aims to address this need by combining advanced preprocessing, specifically debugging, translations, and spell-checking, with terminological and structural analyses into an iterative approach that ensures consistency of the merged ontology. To increase matching quality, PrOM leverages domain-specific information, namely translations scraped from the web and predefined vocabularies, which enable low-level semantic analysis. Compared to existing ontology merging approaches, PrOM also leverages recent developments in NLP.

3. Application Example

This paper uses two application examples. The first is a minimal example created to illustrate the framework's potential. The second uses excerpts of two ontologies from the production domain to demonstrate the framework's practical applicability. Fig. 1 shows excerpts of the minimal example's TBoxes.

The ontology depicted on the left includes elements with English-language Internationalized Resource Identifiers (IRIs), while the one on the right uses random IRIs, but includes French labels. The labels can be leveraged for terminological analyses. For instance,

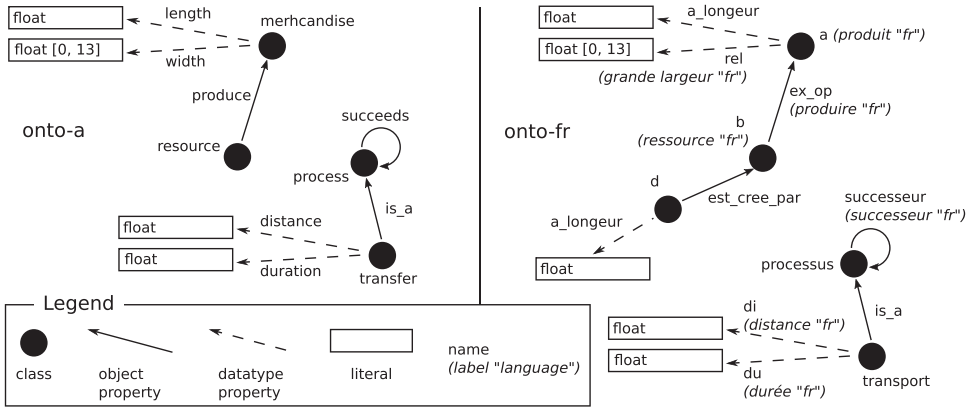

Fig. 1. Excerpt of the minimal example.

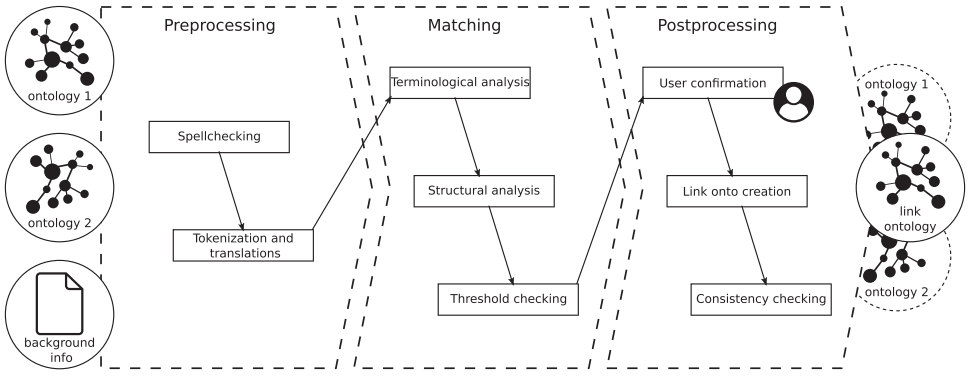

Fig. 2. Process for merging the TBoxes of two ontologies.

synonymous labels such as *transfer* and *transport* indicate equivalence. Structural analyses on the other hand leverage the ontologies' axioms. For example, *transfer* and *transport* are both axiomatized using properties for distance and duration, also indicating equivalence. Interoperability of such process descriptions is essential, e.g., to avoid false negatives during matchmaking between intelligent agents. If a product agent requests a *transfer* process but the resource agents available only offer *transport* processes, valuable resource capabilities remain unused and the product may even seem impossible to produce. However, the example also shows that structural heterogeneity, as in the case of the resource classes, impedes the merging. We added further notions, including domain-specific terms such as *engrenage à vis sans fin*, with and without axiomatization to test features of the merging framework. The typo in *merhcandise* was induced to demonstrate the spellchecker.

The second application example relies on two sound ontologies, namely MAnufacturing's Semantics ONtology (MASON)[1] (Lemaignan et al., 2006) and the IOF process planning ontology[2] (Sarkar and Šormaz, 2019). They include partially overlapping descriptions of production processes. Both ontologies include approximately 200 classes. While MASON comprises 37 object properties and 18 datatype properties, the process planning ontology uses 15 different object properties, but no datatype properties.

[1] https://sourceforge.net/projects/mason-onto/, last accessed: November 19, 2021
[2] https://github.com/kbserm/ProcessPlanningOntology-IOF, last accessed: November 19, 2021

4. Framework for Ontology Merging

This section presents PrOM, a framework for ontology merging targeted to the production domain. PrOM combines syntactic and terminological analyses with a structural analysis of OWL 2 DL axioms and interactive ontology debugging.

4.1. Overview of the Framework

The merging process, cp. Fig. 2, consists of the preprocessing phase, the matching phase, and the postprocessing phase.

During the preprocessing, the input ontologies are checked for spelling errors and are translated. The matching phase combines the syntactic and terminological analysis with the structural analysis. The former extracts the essential parts of the elements' names and compares them. Since the framework's terminological analysis builds on the syntactic one, we refer to their combination as terminological analysis. The structural analysis aims to find graph isomorphisms, which indicate correspondences. The analysis results are combined using weightings and then compared to thresholds. Both the weightings and thresholds may be adapted by the user. Leveraging terminological and structural analyses allows us to match ontologies that differ in language, domain of interest, and degree of axiomatization *(A1)*.

4.2. Assumptions and Available Information

As inputs, the framework takes OWL 2 DL ontologies such as the ones presented in Section 3. OWL allows us to make explicit which

Table 2
Alignment relations by element type.

	Classes	Object properties	Datatype properties
Equivalence	owl:equivalentClass	owl:equivalentProperty	
Subsumption	rdfs:subClassOf	rdfs:subPropertyOf	
Disjointness	owl:disjointWith	owl:propertyDisjointWith	
Inversion	–	owl:inverseOf	–

Table 3
Parts Of Speech usually available in labels by type (* *optional*; + *mandatory*; ! *not applicable*; || *alternative*).

	Adverb	Adjective	Verb	Noun	Adposition
Class	*	*	(*)	+	!
Object property	*	!	+	||	*
Datatype property	*	*	!	+	!

information is available in the ontologies to be merged. Terminology is the meaning of the notions intended by the ontology's author, which is indicated by the elements' names. We assume that the ontologies include meaningful names, e.g., *transfer* for a process, which are either encoded into the IRIs or made explicit via labels. For IRIs, we assume that CamelCase, dromedaryCase, or snake_case are used as notations. Words in labels should be separated using spaces. As we focus on the ontologies' TBoxes, we standardize all names to lowercase to facilitate the comparison. To enable terminological analysis, we translate all names to English. We assume that similar elements can also be identified via similar structures expressed as axioms. These OWL restrictions are expected to be defined in the Disjunctive Normal Form (DNF) for easier analysis. In DNF, the amount of nesting levels in OWL restrictions to be parsed is limited to two. For instance, the definition of a product via relations to a machine, a length, and a volume is assumed to be expressed in DNF as indicated in Equation (3), not in the shorter form presented in Equation (4). This assumption is reasonable, as a transformation to DNF could be automated.

$$\begin{aligned} product \equiv\ & \exists\ Inverse(produces)\ machine \sqcap \exists\ length\ float \sqcup \\ & \exists\ Inverse(produces)\ machine \sqcap \exists\ volume\ float \end{aligned} \tag{3}$$

$$\begin{aligned} product \equiv\ & \exists\ Inverse(produces)\ machine \sqcap \\ & (\exists\ length\ float \sqcup \exists\ volume\ float) \end{aligned} \tag{4}$$

Even though reification is useful for expressing complex relations, matching reified and non-reified properties is a challenge that is out of the scope of this contribution. We also assume that roles and classes are used appropriately by the engineers who initially created the ontologies.

4.3. Cases of Correspondence

We consider four alignment relations that may hold between the two ontologies' elements. These are equivalence, subsumption, disjointness, and, specifically for object properties, inversion. The relations can be represented using standardized object properties, cp. Table 2.

In order to represent a correspondence c, the triple described in Equation (2) can be extended by a confidence rating $cr \in [0,1]$ (Lv and Peng, 2020), cp. Equation (5). The ontologies' elements e_i are unambiguously identifiable via their IRIs.

$$c = (e_1, e_2, r, cr) \tag{5}$$

For instance, this allows us to describe the correspondence between the notions *resource* and *b* from the minimal example, cp. Fig. 1, as shown in Equation (6).

$$(\text{“}a\text{: }resource\text{”}, \text{“}fr\text{: }b\text{”}, \text{“}equivalence\text{”}, 0.7) \tag{6}$$

where the prefixes are defined as follows:

a=http://example.org/onto-a.owl#
fr=http://example.org/onto-fr.owl#

4.4. Preprocessing

In a first step, PrOM ensures that each input ontology is consistent and provides interactive debugging support. OWL DL reasoners can identify inconsistent classes and provide respective explanations. If the reasoner detects inconsistencies in the link ontology, PrOM extracts potentially problematic notions and their axioms. PrOM iterates through these axioms and prompts the engineer in an interactive way which of them should be removed. This process is repeated until the reasoner does not detect any more inconsistencies.

To enable a terminological analysis, all the elements' names must be available in a common language. By default, we choose English because of its prevalence and availability of translators. Engineers may override this choice though. If no labels are available, we extract the elements' names from the IRIs by tokenizing them. In case the language of a name is unknown, a language detector is used. This is relevant if the names are encoded in the IRIs or if the label does not specify a language. Additionally, the engineers may specify a language for each input ontology, e.g., French for the ontology “onto-fr” in the minimal example. This yields better results because language detectors do not work reliably for short text snippets. For the translation, we try a domain-specific vocabulary first, e.g., the International Electrotechnical Vocabulary provided by the International Electrotechnical Commission (IEC) (International Electrotechnical Commission, 2020). If the expression to be translated is not listed in this vocabulary, we rely on a generic translator. The translation is then added to the original element as a label, including a language tag. If the language of an input ontology is specified to be English, a spell checker is used to correct typing errors, such as *merhcandise*, cp. Fig. 1. Since we assume that all axioms are expressed using the DNF, the ontology does not have to be preprocessed for structural analyses.

4.5. Terminological Matching

The terminological matching revolves around the linguistic analysis of the labels. We speak of an explicit match if two labels use the exact same words, called tokens. Otherwise, we compare the tokens' abstractions, possibly resulting in implicit matches. All labels consist of mandatory and optional parts, cp. Table 3, which can be retrieved via a POS tagger. Comparing only words with the same POS tags increases the matching quality and efficiency. To standardize the tokens to their base form, we use a lemmatizer that leverages the POS tags. If an object property's label is reified, i.e., a noun is used instead of the verb, we extract its derivational related form from a lexical database. Note that *precedes* is equivalent to *is predecessor of*, which indicates the role and can be identified by the adposition, but

inverse to *has predecessor*. Also, in the case of classes, there may be adjectives or parts of compound nouns that are tagged as verbs as in the case of *boring tool*.

For explicit terminological matching, we compare the tokens of two labels. To be considered equivalent, the set of tokens T_i must be the same for both labels, cp. Equation (7).

$$T_1 = T_2 \tag{7}$$

Explicit hyponyms can be identified by checking whether the first label's token set, T_1, is subsumed by the second, T_2, cp. Equation (8). For instance, the notion *width* can be assumed to be a generalization of *large width* as (*width*) ⊂ (*large*, *width*). In order to identify hypernyms, Equation (8) can be inverted.

$$T_1 \subset T_2 \tag{8}$$

Disjoint and inverse elements cannot be identified by simply comparing the labels' tokens. Also, independently created ontologies may differ in vocabulary. Such implicit relations between tokens are analyzed using a lexical database. Synonyms indicate equivalent elements, antonyms disjoints, and hyponyms and hypernyms taxonomical relations. In line with the WordNet vocabulary, a set of cognitive synonyms is denoted *synset*. Leveraging the synonyms for all tokens in the labels, we assess their implicit equivalence, cp. Equation (9). That way, PrOM can infer that, e.g., the notions *product* and *merchandise* are equivalent.

$$\begin{aligned} & t_1 \in \bigcup_{t_2 \in T_2} synset(t_2) \quad \forall \quad t_1 \in T_1 \\ \wedge \quad & |T_1| = |T_2| \end{aligned} \tag{9}$$

We check subsumption of classes based on their labels analogously, cp. Equation (10). That way, the framework can identify specializations.

$$\begin{aligned} & t_1 \in \bigcup_{t_2 \in T_2} synset(t_2) \quad \forall \quad t_1 \in T_1 \\ \wedge \quad & |T_1| < |T_2| \end{aligned} \tag{10}$$

Leveraging a corpus, we also extract antonyms for each token. If two elements' labels include antonyms, e.g., *lower* and *souleve* (French for *raise*), we assume that they are disjoint, cp. Equation (11) for object properties.

$$\begin{aligned} & v_1 \in \bigcup_{v_2 \in VERB_2} antonyms(v_2) \quad \forall \quad v_1 \in VERB_1 \\ \wedge \quad & a_1 \in \bigcup_{a_2 \in A_2} synset(a_2) \quad \forall \quad a_1 \in A_1 \\ & s.t. \quad a \in \{adv, adp\}, A \in \{ADV, ADP\} \end{aligned} \tag{11}$$

In order to detect inverse object properties, the algorithm identifies passive constructs, such as *is created by*, instead of *creates*. For this, it searches for two patterns using a rule-based matcher.[3] Such patterns may include dependency labels (*DEP*), quantifiers (*OP*), and simple (*POS*) or extended (*TAG*) POS tags. The first pattern accepts all constructs that include an arbitrary number of auxiliary tokens (*aux*), an auxiliary token indicating a passive structure (*auxpass*), and a past participle verb (*VBN*):

['DEP': 'aux', 'OP': '*', 'DEP': 'auxpass', 'TAG': 'VBN']

The second pattern catches exceptions that only consist of a verb (*VERB*) and an adposition (*ADP*), e.g., *created by*:

['POS': 'VERB', 'POS': 'ADP']

To improve reliability, the framework leverages domain-specific vocabularies. If various notions have been matched for a specific domain, engineers can use them as domain-specific synsets. For instance, analyses for the production domain identified synonyms such as *process*, *operation*, and *activity* (Ocker et al., 2019b). Non-overlapping synsets, e.g., for *process* and *resource* imply disjoints.

Note that all equations in this section can be adapted for object properties and datatype properties according to Table 3.

4.6. Structural Matching

Structural ontology matching distinguishes the types of elements, namely classes, object properties, and datatype properties. Depending on the type, the information available differs greatly, as does the matching algorithm. Since classes are axiomatized using properties, equivalent properties should be identified before classes are matched. Properties, though, are formalized via their domains and ranges. Hence, the framework relies on terminological matches for structurally comparing properties.

The axiomatization of datatype properties includes a class as a domain and a literal as a range. Numeric datatypes may be restricted using lower and upper bounds. Additionally, datatype properties can be specified as functional. For example, the range of the notion *width* may be restricted to exactly one value larger than zero. For two datatype properties, our confidence in an equivalence or subsumption correspondence increases, if any of these three pieces of information overlap. The similarity rating can thus be calculated as shown in Equation (12). The boolean variables *DM* and *FM* indicate whether the two properties have the same domain and are functional. If the range is not restricted, *RM* is also boolean. If one of the ranges is half-bounded, a dedicated rating can be assigned. Otherwise *RM* is calculated as the relative overlap of the two properties' ranges, cp. Equation (13). Here, *ub* denotes the upper bound and *lb* the lower bound.

$$rating_{dp} = \frac{w_d{}^*DM + w_r{}^*RM + w_f{}^*FM}{w_d + w_r + w_f} \tag{12}$$

$$RM = \frac{max(0,\, min(ub_1,\, ub_2) - max(lb_1,\, lb_2))}{max(ub_1 - lb_1,\, ub_2 - lb_2)} \tag{13}$$

If their domains or ranges are disjoint, two datatype properties are also disjoint.

For structurally comparing object properties, the framework relies on their domains, ranges, and attributes. The domains and ranges of object properties can be compared analogously to the domains of datatype properties. Other than datatype properties, object properties are characterized by various axioms, i.e., they can be functional, inverse functional, symmetric, asymmetric, transitive, reflexive, or irreflexive. If one of two object properties is symmetric or reflexive, while the other is asymmetric or irreflexive, respectively, these two properties are disjoint. Also, if a property is functional or inverse functional, it is disjoint with all transitive properties. This is reflected by the disjoint indicator *di*. Otherwise, we can assess their similarity based on their attributes using cosine similarity, cp. Equation (14).

$$\begin{aligned} & cosine\text{-}similarity = cos(\theta) = \frac{\boldsymbol{vec_1} \cdot \boldsymbol{vec_2}}{|\boldsymbol{vec_1}| \cdot |\boldsymbol{vec_2}|} \\ & with \quad \boldsymbol{vec_i}^{\mathrm{T}} = [attribute_1, ..., attribute_n],\ attribute_i \in \{0, 1\} \end{aligned} \tag{14}$$

Equation (15) shows the accumulated similarity rating for equivalent object properties.

$$\begin{aligned} & rating_{op} = di \cdot \frac{w_d{}^*DM + w_r{}^*RM + w_a{}^*cosine\text{-}similarity_{axioms}}{w_d + w_r + w_a} \\ & with \quad di \in \{0, 1\} \end{aligned} \tag{15}$$

Equation (15) can be applied analogously to assess the similarity of hyponyms, while domain and range must be inverted for inverse relations. Relations are said to be disjoint for disjoint domains or ranges, or for contradictions in the properties' axioms, as indicated by the parameter *di*.

[3] https://spacy.io/usage/rule-based-matching, last accessed: November 19, 2021

Classes are axiomatized using object and datatype properties. Hence, a class can be represented using a property vector ***pv***, with each entry representing a property, cp. Equation (16). The property vector includes only the properties identified as equivalent or hyponyms in the terminological matching phase. For the example in Fig. 1, ***pv*** would include the properties *length* and *width*. This allows PrOM to infer that the notions *merchandise* and *produit* are likely to be equivalent as they are both axiomatized using these properties.

$$\begin{aligned} \boldsymbol{pv}^T \;=\; & [op1_a \vee op2_b, ...,op1_m \vee op2_n, \\ & dp1_a \vee dp2_b, ...,dp1_m \vee dp2_n] \end{aligned} \tag{16}$$

We use binary values for a property vector's entries, i.e., whenever a property is used to formalize a class, the respective entry is 1; otherwise 0. Analogously, we encode into the property vector that classes are used as objects in other classes' axiomatizations.

For comparing vectors, *cosine similarity* and *context similarity* are established measures. However, with many properties being shared between ontologies, the similarity ratings usually converge to 1 for cosine similarity. This is because most classes are likely to be axiomatized with only a small subset of all properties, yielding property vectors consisting primarily of zeros. Due to the OWA, this does not mean that the properties are not used, but simply that there is no information regarding their relevance for a class, resulting in a high false positive rate. In contrast, context similarity considers the nodes to which a class is connected. This results in a more accurate assessment of the node's neighborhood, but requires information about the surrounding classes. To cope, we use a simplified similarity measure based on cosine similarity. This measure assesses how many properties two classes have in common, compared to the number of all properties used in the classes' axioms, cp. Equation (17).

$$sim = \frac{\sum_i a_i \wedge b_i}{\sum_i a_i \vee b_i} \tag{17}$$

4.7. Weightings and Thresholds

Engineers may configure the framework via various parameters. Providing the files and the IRIs of the ontologies to be matched is mandatory. Optionally, engineers may specify the languages used, the default language, a domain-specific dictionary, spellchecker activation, and two similarity thresholds. Additionally, the evaluation mode can be activated to assess the matching quality, requiring a reference alignment. Matches that have ratings above the acceptance threshold are automatically accepted, while those below the rejection threshold are rejected. All other matches require confirmation. If both thresholds are set to the same value no user interaction is required.

The framework also relies on various relative weighting factors. By default, we consider terminological and structural matching equally important and set both weights to 1. Engineers may change these weights. For instance, if an ontology does not include any axioms, the relevance of structural similarity is 0. The rating for a terminological correspondence can take one of four values. For an explicit match, the rating is set to .9, matches identified via the domain-specific vocabulary are rated .8, implicit synonyms .7, and implicit antonyms .6. These ratings represent the reliability of the terminological match, which decreases, as the match becomes less explicit. For structural matching, we distinguish the weights for classes, object properties, and datatype properties. The structural similarity rating for datatype properties is influenced by the domain, the range datatype, the exact range specification (i.e., the interval for numerical values), and the functional attribute. The first three are equally rated .3, while the functional attribute is rated .1. Since the OWA holds, the absence of the functional attribute does not mean the attribute is not functional. Hence, a discrepancy does not indicate disjoint properties, but our belief in the equivalence of two datatype properties increases if both are specified as functional. The rating of object property matches is influenced by the domain, the range, and several attributes, such as the property being functional. Due to a lack of statistical insights, we assign equal relative ratings of 1 to these three factors. The structural analysis of classes leverages property vectors, which depend on the property matches. To avoid faulty property vectors, we only include correspondences with a rating higher than .6.

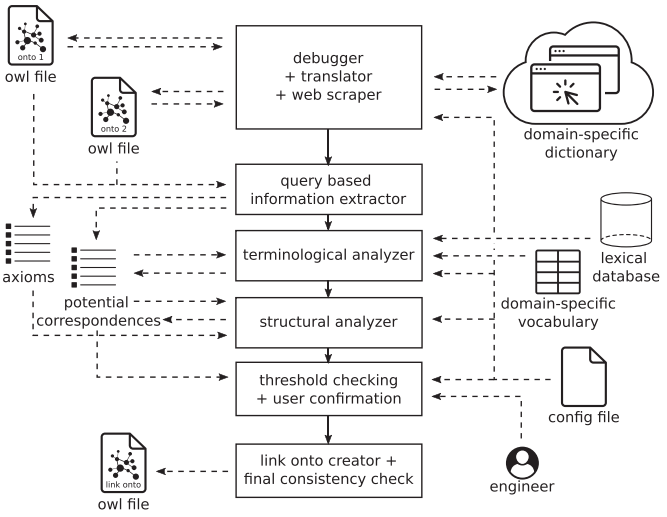

Fig. 3. Overview of the framework's implementation.

4.8. Link Ontology Creation and Consistency Checks

To merge two ontologies, a link ontology is created, including only the correspondences. For this, an algorithm iterates over all the correspondences identified and checks whether their similarity rating is sufficiently high to accept them automatically or low enough to reject them. Every element in between must be reviewed manually. To ensure consistency of the alignment, PrOM only allows one correspondence of the types equivalence, hyponym, and hypernym per notion. In addition to one equivalent, subordinate, or superordinate notion, object properties may also have one inverse property, while the number of disjoints is unlimited for all notions. If an input ontology includes equivalent notions, the respective correspondences can be inferred using an OWL DL reasoner. PrOM provides two algorithms for selecting correspondences. The greedy one selects the correspondences with the highest confidence ratings first, while limiting the number of correspondences a notion appears in. Additionally, PrOM provides an optimal selection algorithm, which maximizes the sum of the confidence ratings of the correspondences included in the alignment. By default, PrOM uses the greedy selection for two reasons. It provides better performance, as the worst case computational complexity of the optimal selection is $\mathcal{O}(n!)$ as opposed to a worst case of $\mathcal{O}(n^2)$ for the greedy selection. Additionally,

optimizing for the highest overall similarity score is not necessarily beneficial in case of partially overlapping ontologies. Instead, the correspondences with the highest scores may be rejected in favor of several correspondences with lower ratings, possibly resulting in a flawed alignment.

For all correspondences accepted, two elements are created in the link ontology. These two elements are linked via the correspondence's relation, and also to the respective elements in the original ontologies. The input ontologies are referenced as imports from the link ontology.

Lastly, the framework checks the link ontology and the two input ontologies for inconsistencies. PrOM uses an OWL DL reasoner and provides interactive debugging support analogously to the preprocessing phase, cp. Section 4.4. Note that the input ontologies are assumed to be consistent after the preprocessing and handled as ontology imports. Hence, only axioms from the link ontology can be removed in this phase. The explanations provided by the reasoner may help finding inconsistencies in the input ontologies that can be identified only when combining them, though.

5. Implementation and Discussion

5.1. Implementation Details

Fig. 3 shows an overview of the framework's architecture. The prototype is implemented in Python and available online.[4]

The framework's modules were built to be exchangeable and adaptable *(A4)*. This enables engineers to adapt the weightings and thresholds and provide additional resources. As inputs, the framework takes two OWL ontologies, a domain-specific vocabulary, and a configuration file. The paths to the ontology files and additional resources are specified in the configuration file, which also sets all parameters, cp. Section 4.7. For managing the ontologies, we rely on Owlready2 (Lamy, 2017). Algorithm 1 shows the recursive function developed for debugging ontologies. In case of simple inconsistencies, Owlready2 returns a list of inconsistent classes. However, more complex inconsistencies may lead to an exception. Then, the *analyzeExplanation()* function uses regular expressions to extract the explanation generated by the reasoner Pellet from the traceback and identifies potentially inconsistent notions. PrOM iterates through all axioms associated to potentially inconsistent notions and prompts the user whether the axiom should be deleted via the *getUserDecision()* function.

Algorithm 1. Interactive ontology debugging.

```
1:  procedure debugOnto(onto)
2:      inconsistentElements ← runReasoner()
3:      if exception then
4:          inconsistentElements ← analyzeExplanation(traceback)
5:      if inconsistentElements = ∅ then
6:          onto.save()
7:          break
8:      for ie ∈ inconsistentElements do
9:          axioms ← extractAxioms(ie)
10:         for ax ∈ axioms do
11:             if getUserDecision() = ⊤ then
12:                 deleteAxiom(ax)
13:     debugOnto(onto)
```

The information extractor uses Owlready2 to extract the classes, object properties, and datatype properties, including their labels and the labels' languages, from the source ontologies. To extract axioms, we use SPARQL Protocol And RDF Query Language (SPARQL) queries. For domain-specific translations, we scrape the IEC's International Electrotechnical Vocabulary (International Electrotechnical Commission, 2020). If this fails, we use huggingface transformers (Wolf et al., 2020), but also support Google Translate. Note that using the domain-specific dictionary increases translation quality but decreases performance due to the scraping process. If a label is English and English is the default language, we use a spellchecker. The terminological analysis, cp. Algorithm 2, including lemmatization, uses spaCy (Honnibal et al., 2020) and NLTK (Bird et al., 2009). Here, t_i refer to individual tokens of a label, while $|T_i|$ refer to a label's token set. The semantic matching is based on WordNet and domain-specific vocabularies stored in a CSV file. spaCy is also used to identify passive object properties via patterns.

Algorithm 2. Terminological matching functions.

```
1:  function explicitLabelMatching(label_1, label_2)
2:      if t_1.lemma ∈ [t_2.lemma ∀ t_2 ∈ label_2] ∀ t_1 ∈ label_1 then
3:          if |T_1| = |T_2| then
4:              return equivalence(label_1, label_2)
5:          else if |T_1| < |T_2| then
6:              return hypernym(label_1, label_2)
7:      else if t_2.lemma ∈ [t_1.lemma ∀ t_1 ∈ label_1] ∀ t_2 ∈ label_2 ∧ |T_2| < |T_1|
    then
8:          return hyponym(label_1, label_2)
9:  function implicitLabelMatching(label_1, label_2, type_elem)
10:     for i ∈ (1, 2) do
11:         syns_i ← getSynonyms(label_i)
12:         ants_i ← getAntonyms(label_i)
13:     if t_2.lemma ∈ syns_1 ∀ t_2 ∈ label_2 and |T_1| = |T_2| then
14:         return equivalence(label_1, label_2)
15:     else if t_2.lemma ∈ syns_1 ∀ t_2 ∈ label_2 then
16:         return hyponym(label_1, label_2)
17:     else if t_1 ∈ syns_2 ∀ t_1 ∈ label_1 then
18:         return hypernym(label_1, label_2)
19:     else if ∃ t_2 ∈ label_2 ∋ t_2 ∈ ants_1 then
20:         return disjoint(label_1, label_2)
```

Regarding structural matching, cp. Algorithm 3, engineers may choose to limit the search space to matches identified via terminological matching, or to analyze all possible combinations of classes for structural similarity. Note that we use the abbreviations *op*, *dp*, *di*, and *pv* for object properties, datatype properties, the disjoint indicator, and the property vector analogously to Section 4. The formulas for the individual similarity ratings are implemented according to Section 4, and each matching function is run for all combinations of respective elements.

Algorithm 3. Structural matching functions.

```
1:  function opStructureMatching(op_1, op_2, alignment_classes, relation)
2:      rating_domain ← compareDomain(op_1, op_2, alignment_classes, relation)
3:      rating_range ← compareRange(op_1, op_2, alignment_classes, relation)
4:      di ← checkIfDisjoint(op_1.attributes, op_2.attributes)
5:      rating_attributes ← cosSim(op_1.attributes, op_2.attributes, relation)
6:      rating_op1,op2 ← calculateRating(rating_domain, rating_range, di, rating_attributes)
7:      return rating_op1,op2
8:  function dpStructureMatching(dp_1, dp_2, alignment_classes, relation)
9:      rating_domain ← compareDomain(dp_1, dp_2, alignment_classes, relation)
10:     rating_range ← compareRange(dp_1, dp_2, relation)
11:     rating_attribute ← cosSim(dp_1.functional, dp_2.functional, relation)
12:     rating_dp1,dp2 ← calculateRating(rating_domain, rating_range, rating_attribute)
13:     return rating_dp1,dp2
14: function classStructureMatching(axioms_class1, axioms_class2, alignment_properties)
15:     pv ← checkThresholdAndCorrespondenceType(alignment_properties)
16:     for i ∈ (1, 2) do
17:         pv_i ← 0
18:         for counter, p ∈ enumerate(pv) do
19:             for a ∈ axioms_class_i do
20:                 if p ∈ a.properties then
21:                     pv_i[counter] ← 1
22:     rating_class1,class2 ← relSim(pv_1, pv_2)
23:     return rating_class1,class2
```

PrOM combines terminological and structural analysis results in a preliminary alignment, which is then assessed using the ratings.

[4] https://github.com/felixocker/prom, last accessed: November 19, 2021

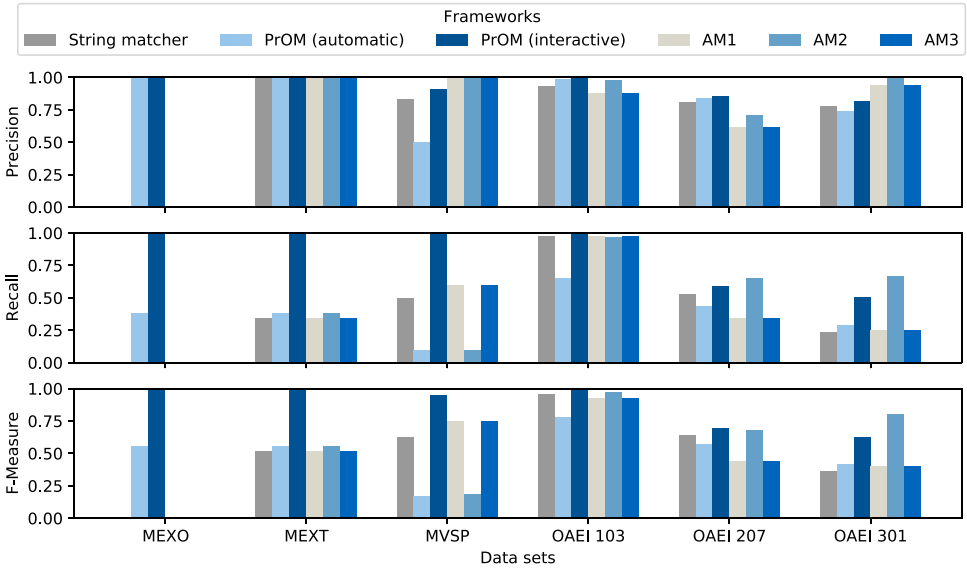

Fig. 4. Benchmark using several frameworks (*String matcher* implemented as a baseline; *PrOM (automatic)* without any user interaction, *acceptanceThreshold* = .6; *PrOM (interactive)* with user interaction, *interactiveCorridor* = [.3, .6]; *AM1* - AgreementMaker with the "Base Similarity Matcher"; *AM2* - AgreementMaker with the "Vector Based Multi-Words Matcher", and *AM3* - AgreementMaker with the "Basic Structural Selector Matcher" that also leverages *AM1*) and several data sets (minimal example without (*MEXO*) and with (*MEXT*) English translations; MASON vs. IOF process planning ontology (*MVSP*); and three examples from the 2012 OAEI benchmark set, namely the combinations 101 vs. 103 (*OAEI 103*), 101 vs. 207 (*OAEI 207*), and 101 vs. 301 (*OAEI 301*).

Table 4
Number of user interactions PrOM requires in interactive mode by data set compared to the number of notions per input ontology. Notions include classes, object properties, and datatype properties.

Data set	MEXO	MEXT	MVSP	OAEI 103	OAEI 207	OAEI 301
# interactions	13	13	8	33	19	14
# notions ontology 1	17	17	135	122	122	122
# notions ontology 2	20	20	100	122	121	55

For creating the final alignment from the correspondences identified, cp. Section 4.8, the engineer may choose from a greedy, cp. Algorithm 4, and an optimal selection algorithm via the configuration file. The optimal selection algorithm, including a multi-processing implementation for improved performance, is available as a part of the implementation. However, PrOM uses the greedy selection by default, cp. Section 4.8.

Algorithm 4. Greedy correspondence selection.

```
1: function greedySelection(correspondences)
2:     selection ← ∅
3:     correspondences.sortByRating(reverse = ⊤)
4:     for c ∈ correspondences do
5:         overlap ← set(e ∀ e ∈ selection if ∃ e.notion = c.notion)
6:         if overlap = ∅ then
7:             alignment. append(c)
8: return alignment
```

Depending on the thresholds specified, the engineer is prompted to confirm matches via a Command Line Interface (CLI). Owlready2 is used to create the link ontology. The framework generates a random IRI and adds a description that refers to the original ontologies, respective imports, and all correspondences from the alignment.

5.2. Evaluation

For evaluating PrOM's automatic and interactive mode, we implemented a string matcher based on Levenshtein distance as a naive but sound baseline. Additionally, we used AgreementMaker[5] as a representative ontology merging framework, cp. Section 2.4. The matching algorithms used with AgreementMaker are the "Base Similarity Matcher" (AM1), the "Vector Based Multi-Words Matcher" (AM2), and the "Basic Structural Selector Matcher" (AM3), which is a second layer matcher we operated on the results of AM1. All matchers worked out of the box and we did not tune them in any way. We used several data sets for the benchmark. The minimal example introduced in Section 3 was designed to test PrOM's features, including translations. Since AgreementMaker does not provide translations, we used this data set without (*MEXO*) and with translations (*MEXT*). The alignment of excerpts of MASON and the IOF process planning ontology (*MVSP*) is a larger example also representing the production domain. We created reference alignments for these data sets manually and provide them as a part of the implementation. For better comparability, we also used three data sets including reference alignments from the OAEI 2012 campaign[6] referenced in the AgreementMaker GitHub repository. The data sets OAEI 1, 2, and 3 correspond to the matching tasks "101 vs. 103″, "101 vs. 207″, and "101 vs. 301″, respectively. The benchmark results regarding precision, recall, and F-measure (Ochieng and Kyanda, 2018) are presented in Fig. 4. Table 4 shows the number of user

[5] https://github.com/agreementmaker/agreementmaker, last accessed: November 19, 2021, run using Java 11.0.11

[6] http://oaei.ontologymatching.org/2012/benchmarks/index.html#datasets, last accessed: November 19, 2021

interactions required by PrOM's interactive mode compared to the number of notions included in the respective input ontologies.

MEXO showed that PrOM is capable of handling the spelling errors induced and correctly translating domain-specific terms. In interactive mode, critical matches with ratings between the default thresholds .3 and .6 were presented to the user for assessment. When fully automating the process using a threshold of .6, PrOM rejects correspondences that were found solely based on a terminological analysis, thus significantly reducing recall. Note that PrOM identified a disjoint between the notions *first name* and *last name* in the "101 vs. 103″ scenario. Even though correct, this correspondence was not included in the official reference alignment. In the case of *MSVP*, PrOM erronously identified the correspondence *(mason:uses_tool, ppo:uses_equation_type, equivalence)*, which can be explained by the tokenization process during terminological matching, which focused on the verb "use".

The benchmark shows that PrOM works especially well with axiomatized ontologies from the production domain, such as *MEXO*, and performs decent on other data sets such as the ones included in OAEI, even when defaulting to the terminological analysis. The low recall in PrOM's automatic mode compared to the interactive one is due to the high acceptance threshold of .6 in combination with a lack of axiomatization. This also highlights the benefit of the interactive mode regarding alignment quality, while keeping the effort for engineers at a reasonable level. Performance for these application examples was acceptable as ontology merging is not a time critical task and the merging processes were completed within seconds using a regular computer.[7]

5.3. Discussion

Even though production ontologies are intrinsically heterogeneous *(A1)*, the framework copes via a combination of preprocessing, including spellchecks and translations, and terminological and structural analyses as well as consistency checks. Domain-specific knowledge in the form of dictionaries and vocabularies is integrated, and engineers may confirm or reject arguable matches. Due to the combination of terminological and structural analyses with domain-specific knowledge, the framework is fit to merge ontologies with partially overlapping domains of interest *(A1.1)*. Additionally, ontologies that differ in their degrees of axiomatization can also be analyzed *(A1.2)*, e.g., if one ontology includes no axioms. This is a case for purely terminological matching. However, the matching quality greatly increases if an axiomatization can be leveraged. This was confirmed by the benchmark, cp. Section 5.2, which also showed that the matching quality *(A2)* is promising. The framework was implemented modularly *(A3)* regarding the implementation and the reversibility of the merging process, which is ensured by creating a link ontology. Lastly, we designed the framework to be adaptable *(A4)*. While engineers may adapt relevant weightings and thresholds, software engineers can adapt the framework due to its modular architecture. Overall, PrOM's performance seemed reasonable, even when run on a regular personal computer.

To further automate the merging of two ontologies, reification as a design choice should be considered. As another source of information, the comments may be analyzed and alignments with common TLOs could be leveraged (Anjum et al., 2012). Additionally, more advanced algorithms for structural matching and combinatorial optimization (Euzenat and Shvaiko, 2013) should be included, and the framework's computational performance improved, which is beneficial for merging large ontologies. Also, an n-ary matching approach (Babalou and König-Ries, 2020) may be beneficial if large numbers of ontologies need to be merged.

[7] Intel Core i7–10510 U CPU, 16 GB RAM

Even though we included debugging support using an OWL DL reasoner, the automated resolution of inconsistencies remains challenging. Explanations provided by reasoners may be hard to interpret, complicating both a manual and an automated resolution.

The quality of the resulting alignments strongly depends on the choice of weightings and thresholds. Since choosing these is a major challenge for engineers due to the high number of dimensions to be considered, an automated parameterization is desirable. If a large amount of labeled data were available, a model with the weighting factors could be trained. Also, there is related work regarding automated parameterization (Ritze and Paulheim, 2011) that may be adopted in future work. In addition, a rigorous statistical analysis of the use of OWL axioms, cp. Section 4.7, could help with identifying appropriate weighting factors.

6. Summary and Outlook

We presented PrOM for merging the TBoxes of production ontologies. The framework leverages spellchecks, translations, and analyses regarding syntax, terminology, and structure. The matching's reliability is increased by using domain-specific dictionaries and vocabularies and the results are validated using consistency checks. The resulting alignment is stored in a link ontology that imports the source ontologies, thus realizing a modular architecture. We provide an adaptable implementation regarding the software and input data such as thresholds and domain-specific information. As discussed in Section 5.3, the framework merges heterogeneous ontologies, while addressing matching quality, modularity, and adaptability. Its applicability was shown in a benchmark. A dedicated minimal example demonstrates the capabilities of the framework, while excerpts from two established production ontologies were merged to show real-life applicability.

Even though PrOM has been shown to be applicable as is, there is room for further improvements. First, ontology merging requires formalized knowledge representations. These have been investigated for a long time (Sowa, 1999), but formal knowledge representation should be pursued further and engineers should be enabled to apply these technologies. Second, we propose to investigate two approaches for choosing weightings and thresholds. Engineers can be expected to benefit from an easy-to-use graphical user interface, rather than the currently implemented CLI, which would allow them to navigate and possibly adapt the weightings and thresholds. Additionally, the parameters could be continuously improved as the framework is used. For creating labeled data sets, engineers would need to assess whether a specific alignment was correct. To further improve the overall quality, the inconsistency check would benefit from an automated analysis. Hereby, the explanations provided by the reasoner are promising and approaches for OWL restriction conciliation (Grygorova et al., 2020) may be helpful. Third, the framework should be extended to be able to merge ABoxes. This aspect was intentionally neglected in this paper, as the boundary conditions differ from the merging process of TBoxes.

The framework presented is a step towards automatically merging production ontologies. Together with advances in digitization and ontology creation, this has the potential to integrate the knowledge of various viewpoints, which can be expected to shorten development times and facilitate integration of production resources across vendors and countries.

CRediT authorship contribution statement

Felix Ocker, M.Sc.: Conceptualization; Data curation; Formal analysis; Investigation; Methodology; Project administration; Software; Validation; Visualization; Roles/Writing - original draft; Writing - review & editing. Prof. Birgit Vogel-Heuser: Conceptualization; Funding acquisition; Investigation; Methodology; Project administration; Resources; Supervision; Validation; Writing - review & editing. Prof.

Christiaan J. J. Paredis: Conceptualization; Investigation; Methodology; Supervision; Validation; Writing - review & editing.

Declaration of Competing Interest

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

Acknowledgment

We kindly thank the German Research Foundation for funding the project SFB 768 T5 and the Bavarian Ministry of Economic Affairs, Regional Development, and Energy for funding the project MEvoDiP (grant no. DIK0246/04).

REFERENCES

Feldmann, S., Herzig, S.J., Kernschmidt, K., Wolfenstetter, T., Kammerl, D., Qamar, A., Lindemann, U., Krcmar, H., Paredis, C.J., Vogel-Heuser, B., 2015. Towards effective management of inconsistencies in model-based engineering of automated production systems. IFAC-Pap. 28 (3), 916–923.
Ocker, F., Kovalenko, I., Barton, K., Tilbury, D., Vogel-Heuser, B., 2019a. A framework for automatic initialization of multi-agent production systems using semantic web technologies. IEEE RA-L 4 (4), 4330–4337.
Gruber, T.R., 1993. A translation approach to portable ontology specifications. Knowl. Acquis. 5 (2), 199–220.
Studer, R., Benjamins, V., Fensel, D., 1998. Knowledge engineering: Principles and methods. DKE 25 (1–2), 161–197.
Atkinson, C., Gutheil, M., Kiko, K., 2006. On the relationship of Ontologies and Models. Int. Workshop Meta-Model., Vol. 96, Ges. für Inform., Karlsr., Ger. 47–60.
Hitzler, P., Krötzsch, M., Rudolph, S., 2010. Foundations of Semantic Web Technologies. Chapman & Hall/CRC,, Boca Raton, USA.
Smith, M.K., Welty, C., McGuinness, D.L., 2004. W3C OWL Web Ontology Language Guide. Tech. rep.
Herzig, S.J.I., Qamar, A., Reichwein, A., Paredis, C.J.J., 2011. A conceptual framework for consistency management in model-based systems engineering. IDETC/CIE 1329–1339.
Ocker, F., Vogel-Heuser, B., Paredis, C.J.J., 2019b. Applying semantic web technologies to provide feasibility feedback in early design phases. JCISE 19 (4), 041016.
Ameri, F., McArthur, C., 2014. Semantic rule modelling for intelligent supplier discovery. IJCIM 27 (6), 570–590.
Köcher, A., Hildebrandt, C., VieiradaSilva, L.M., Fay, A., 2020. A Formal Capability and Skill Model for use in Plug and Produce Scenarios. In: ETFA. IEEE, pp. 1663–1670.
Zhao, Y., Liu, Q., Xu, W., 2017. Open Industrial Knowledge Graph Development for Intelligent Manufacturing Service Matchmaking. In: INDIN. IEEE, pp. 194–198.
Puttonen, J., Lobov, A., Lastra, J.L., 2013. Semantics-based composition of factory automation processes encapsulated by web services. TII 9 (4), 2349–2359.
Euzenat, J., Shvaiko, P., 2013. Ontology Matching. Springer,, Heidelberg, New York.
Arp, R., Smith, B., Spear, A.D., 2015. Building Ontologies with Basic Formal Ontology. MIT Press,.
Schmidt, D., Trojahn, C., Viera, R., 2019. Matching BFO, DOLCE, GFO and SUMO: an evaluation of OAEI 2018 matching systems. Semin. De. Pesqui. em Ontol. no Bras., CEUR.
Stevens, R., Lord, P., Malone, J., Matentzoglu, N., 2019. Measuring expert performance at manually classifying domain entities under upper ontology classes. JoWS 57, 100469.
Maedche, A., Staab, S., 2002. Measuring similarity between ontologies. In: EKAW 2473. Springer, pp. 251–263.
Ardjani, F., Bouchiha, D., Malki, M., 2015. Ontology-Alignment Techniques: Survey and Analysis. IJMECS 7 (11), 67–78.
Pawełoszek, I., Korczak, J., 2018. Merging of ontologies – Conceptual design issues. In: ACM International Conference Proceeding Series. Association for Computing Machinery, pp. 59–63.
Miller, G.A., 1995. WordNet: a lexical database for English. Commun. ACM 38 (11), 39–41.
Chatterjee, N., Kaushik, N., Gupta, D., Bhatia, R., 2018. Ontology merging: a practical perspective. In: Smart Innovation, Systems and Technologies 84. Springer, pp. 136–145.
Aumueller, D., Do, H.-H., Massmann, S., Rahm, E., 2005. Schema and ontology matching with COMA. ACM SIGMOD Int. Conf. Manag. Data.
Noy, N.F., Musen, M.A., 2000. Algorithm and Tool for Automated Ontology Merging and Alignment. In: IAAI-00. AAAI Press, pp. 450–455.
Cruz, I.F., Antonelli, F.P., Stroe, C., 2009. AgreementMaker: efficient matching for large real-world schemas and ontologies. Proc. VLDB Endow. 2 (2), 1586–1589.
Hu, W., Qu, Y., 2008. Falcon-AO: a practical ontology matching system. JWS 6 (3), 237–239.
Hu, W., Jian, N., Qu, Y., Wang, Y., 2005. GMO: a graph matching for ontologies. K. -CAP Workshop Integr. Ontol., CEUR.
Tang, J., Li, J., Liang, B., Huang, X., Li, Y., Wang, K., 2006. Using Bayesian decision for ontology mapping. JoWS 4 (4), 243–262.
Giunchiglia, F., Autayeu, A., Pane, J., 2012. S-match: an open source framework for matching lightweight ontologies. Semant. Web 3 (3), 307–317.
Stumme, G., Maedche, A., 2001. FCA-MERGE: bottom-up merging of ontologies. IJCAI 225–230.
Chen, R.C., Bau, C.T., Yeh, C.J., 2011. Merging domain ontologies based on the WordNet system and fuzzy formal concept analysis techniques. Appl. Soft Comput. 11 (2), 1908–1923.
Robin, C.R.R., Uma, G.V., 2010. A novel algorithm for fully automated ontology merging using hybrid strategy. Eur 47 (1), 74–081.
Lv, Z., Peng, R., 2020. A novel meta-matching approach for ontology alignment using grasshopper optimization. KBS, 106050.
Fu, B., Brennan, R., O'Sullivan, D., 2012. A configurable translation-based cross-lingual ontology mapping system to adjust mapping outcomes. JWS 15, 15–36.
Ibrahim, S., Fathalla, S., ShariatYazdi, H., Lehmann, J., Jabeen, H., 2019. From Monolingual to Multilingual Ontologies: The Role of Cross-Lingual Ontology Enrichment. In: International Conference on Semantic Systems. Springer, pp. 215–230.
Helou, M.A., Palmonari, M., Jarrar, M., 2016. Effectiveness of automatic translations for cross-lingual ontology mapping. JAIR 55, 165–208.
Shvaiko, P., Euzenat, J., 2013. Ontology matching: State of the art and future challenges. TKDE 25 (1), 158–176.
Algergawy, A., Faria, D., Ferrara, A., Fundulaki, I., Harrow, I., Hertling, S., Jimenez-Ruiz, E., Orcid-Karam, N., Khiat, A., Lambrix, P., Li, H., Montanelli, S., Paulheim, H., Pesquita, C., Saveta, T., Shvaiko, P., Splendiani, A., Thiéblin, E., Trojahn, C., Vataščinová, J., Zamazal, O., Zhou, L., 2019. Results of the Ontology Alignment Evaluation Initiative 2019. Tech. rep.
Cha, S., Vogel-Heuser, B., Fischer, J., 2020. Analysis Of metamodels for model-based production automation system engineering, collaborative Intell. Manuf 2 (2), 45–55.
OPC Foundation. OPC Unified Architecture - Part 5: Information Model, Tech. rep.
Plattform Industrie 4.0, Details of the Asset Administration Shell - Part 1, Tech. rep., German Federal Ministry for Economic Affairs and Energy (2020).
AutomationML Consortium, 2014. AutomationML Whitepaper Part 1- Architecture and general requirements. Tech. rep.
Chungoora, N., Young, R.I., Gunendran, G., Palmer, C., Usman, Z., Anjum, N.A., Cutting-Decelle, A.F., Harding, J.A., Case, K., 2013. A model-driven ontology approach for manufacturing system interoperability and knowledge sharing. CII 64 (4), 392–401.
Industrial Ontologies Foundry, IOF Website (2020). ⟨https://www.industrialontologies.org/⟩.
Hildebrandt, C., Kocher, A., Kustner, C., Lopez-Enriquez, C.M., Muller, A.W., Caesar, B., Gundlach, C.S., Fay, A., 2020. Ontology building for cyber-physical systems: application in the manufacturing domain. TASE 17 (3), 1266–1282.
Witherell, P., Kulvatunyou, B., Rachuri, S., 2013. Towards the synthesis of product knowledge across the lifecycle. In: IMECE 12 ASME.
Kumar, S.K., Harding, J.A., 2013. Ontology mapping using description logic and bridging axioms. CII 64 (1), 19–28.
Anjum, N.A., Harding, J.A., Young, R.I., Case, K., 2012. Mediation of foundation ontology based knowledge sources. CII 63 (5), 433–442.
Adamczyk, B.S., Szejka, A.L., Canciglieri, O., 2020. Knowledge-based expert system to support the semantic interoperability in smart manufacturing. CII 115, 103161.
Kramer, M.E., Burger, E., Langhammer, M., 2013. View-centric engineering with synchronized heterogeneous models. In: VAO. ACM Press, New York, USA.
Wolfenstetter, T., Basirati, M.R., Böhm, M., Krcmar, H., 2018. Introducing TRAILS: A tool supporting traceability, integration and visualisation of engineering knowledge for product service systems development. JSS 144, 342–355.
Lemaignan, S., Siadat, A., Dantan, J.-Y., Semenenko, A., 2006. MASON: A Proposal for an Ontology of Manufacturing Domain. In: DIS. IEEE, Prague, Czech Republic, pp. 195–200.
Sarkar, A., Šormaz, D., 2019. Ontology model for process level capabilities of manufacturing resources. Procedia Manuf. 39, 1889–1898.
International Electrotechnical Commission, IEC 60050 - International Electrotechnical Vocabulary (2020). ⟨http://www.electropedia.org/⟩.
Lamy, J.B., 2017. Owlready: ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. Artif. Intell. Med. 80, 11–28.
Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., vonPlaten, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M., 2020. Transformers: State-of-the-art Natural Language Processing. EMNLP, Assoc. Comput. Linguist.

Babalou, S., König-Ries, B., 2020. Towards building knowledge by merging multiple ontologies with CoMerger: A partitioning-based approach. Tech. rep.
Bird, S., Klein, E., Loper, E., 2009. Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit, 1st edition,,. O'Reilly Media,.
Ochieng, P., Kyanda, S., 2018. Large-scale ontology matching: State-of-the-art analysis. ACM Comput. Surv. 51 (4), 1–35.
Honnibal, M., Montani, I., Van Landeghem, S., Boyd, A., 2020. spaCy: Industrial-strength Natural Language Processing in Python. DOI is: 10.5281/zenodo.1212303.
Ritze, D., Paulheim, H., 2011. Towards an automatic parameterization of ontology matching tools based on example mappings. Int. Workshop Ontol. Matching, CEUR.
Sowa, J.F., 1999. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing Co.,, Pacific Grove, USA.
Grygorova, E., Id, S.B., König, B., Id, R., 2020. Toward OWL Restriction Reconciliation in Merging Knowledge. In: ESWC. Springer.