February 2024

Ontology Pattern Substitution: Toward their use for domain ontologies

William Leighton DAWSON and C. Maria KEET
[a] *Department of Computer Science, University of Cape Town, South Africa,*
dwswil002@myuct.ac.za, mkeet@cs.uct.ac.za
ORCiD ID: William Leighton Dawson https://orcid.org/0000-0002-2799-3570, C. Maria Keet https://orcid.org/0000-0002-8281-0853

Abstract. As ontologies find an ever-larger number of applications, the diversity of domain ontologies and the requirements for their intended uses increases as well, creating challenges for interoperability and tooling. There are often multiple ways of modelling the same knowledge, which have coalesced into ontology patterns and modelling styles, and pattern alignments for perceived to semantically the same domain knowledge have been identified. To facilitate interoperability and applicability of foundational ontology-based modelling choices with domain ontologies and so-called application ontologies or conceptual data models, we propose a general framework for the substitution of one pattern for another. This can be applied by various methods, including purely syntactic comparisons. A proof-of-concept tool that implements such a syntax-based approach for FOL ontologies encoded in CLIF is demonstrated and evaluated against a set of DOLCE-aligned ontologies.

Keywords. Ontology Design Patterns, Foundational ontologies, Application ontologies, Ontology Development, Modelling Style

1. Introduction

Ontologies have found increasing use in information systems, in part due to the popularisation of the decidable Description Logic-based languages with their standardised serialisations, such as the Web Ontology Language (OWL) family [1], and as ontologies such as the Gene Ontology [2,3] as well as recent developments with the adoption of FIBO in the banking industry, ontologies for cultural heritage, among many others, and their connection with popular knowledge graphs [4,5]. The more expressive but undecidable first-order logic (FOL) is often used as a common superset within which to describe other logics, but it is also used for its expressiveness to describe foundational ontologies and for computational knowledge representation such as in Colore [6]. Further, also conceptual data models have been given logic-based reconstructions in various logics, including Description Logics (DLs) (e.g., [7,8]), OWL (e.g., [9]), and three flavours of FOL, being common logic interchange format (CLIF) [10], Alloy [11], and Z [12].

Since much like conceptual data modelling for software engineering, ontology engineering is a human practice, design patterns and modelling styles have arisen. These are patterns of expression which reflect either their users' preferred ontological view of the semantic relationships they must describe or pragmatics for the prospective application.

For instance, representing 'marriage' as a class Marriage or as a relationship married to and the colour of a car as Colour as a subtype of Quality or as an attribute hasColour.

In many cases, the same knowledge may be represented in multiple, nearly equivalent, ways, and which design pattern is used depends on factors such as preference or customs, affordance of the modelling language or modelling tool, of model quality, and/or intended purpose of the artefact. For instance, biological and biomedical ontologies typically follow the modelling style as in BFO [13] and ontology-based data access systems [14] require a so-called "applied" style [15] for more efficient computational processing. This causes problems both in ontology development and in ontology-driven conceptual modelling, notably when developing multiple conceptual data models from a single ontology to ensure system interoperability upfront [16,17] or, vice versa, align multiple conceptual data models to a domain or foundational ontology [18], they will likely mismatch. Aligning ontologies that overlap but have different modelling styles—and thus use different design patterns in their representations—requires identifying alignments of such patterns, so that mappings may be described from one onto the other.

Several patterns and pattern alignments have been identified [19] which are de facto special cases of carefully curated complex alignments [20]. Instead of finding alignments, as in [19,20], we are interested in transforming one axiom pattern into another. Pattern-based transformations are useful in a number of applications, including 1) simplification, by transforming foundational ontology-inspired elaborate representations of high complexity that also make ontologies harder to verbalise into compacter patterns that need fewer language features and are easier to implement; 2) harmonisation of modelling style to facilitate current or prospective ontology alignment; or 3) substitution of one foundational ontology for another [21]. This requires direct substitution of one design pattern for another within an ontology, rather than aligning two disparate ontologies. Pat-O-Mat [22] proposes this for OWL ontologies by exploiting SPARQL queries. Since such queries are constrained by the logic on which they operate (OWL), such an approach does not generalise to full FOL or above to cover more and more expressive ontologies and conceptual data models, whose expressiveness is at least popular for several ontologies, such as those in Colore [6] and conceptual data modelling languages, such as ORM [23]. While DLs are used due to their decidability, limitations are well-known and there is still interest in knowledge representation in FOL (e.g., [24,6]). Additionally, most new logics are presented in the literature with mappings into FOL (e.g., recently in [25]), and reasoners such as Vampire [26] continue to receive attention for their relevance to theorem proving and software/hardware verification [27].

We resolve these issues in a two-step approach. First, we propose a broadly applicable framework for pattern substitution in ontology transformation that can be implemented not only by querying an ontology for axiom satisfaction, but also by other methods, including purely syntactic comparisons. Second, we apply this framework, along with devising algorithms for, the substitution of the five ontology pattern (OP) alignments in [19] that list pairs of patterns of an applied style and foundational ontology-inspired modelling style. The framework and algorithms were implemented in a proof-of-concept tool [1] named Humusha (from Zulu: to translate/interpret), availing of ontologies in FOL encoded in CLIF or converted into it, and using the parsing facilities of the Heterogeneous Tool Set (HETS) [28]. Since this is, to the best of our knowledge, the first

[1] Available at: `https://github.com/wleightond/humusha`

solution to this problem, we use a preliminary use case-based evaluation to demonstrate its usefulness and usability. It showed correctness of encoding, but also that alignments of domain ontologies to foundational ontologies are partial, which limits the number of pattern substitutions.

The remainder of the paper is structured as follows. After introducing the running example in Section 2, we present the framework and algorithms in Section 3. The implementation and evaluation is presented in Section 4. We discuss in Section 5 and conclude in Section 6.

2. Running Example: The Representing Roles Pattern

Besides the pattern pairs collected and formalised in [19], we also consider those in [29]. One of the patterns of [19] will be used as a running example, being the Representing Roles pattern, and, in the case of patterns that reference classes and relationships from a foundational ontology, DOLCE [30] will be used. The other patterns are included in the supplementary material (see details at the end of paper).

The Representing Roles (RR) pattern concerns the modelling choice to represent a role as either a subclass or a separate class with their relationship described by classes and relationships from a foundational ontology. In the latter case, One-sided (Generic) Constant Dependence (*OD* and *OGD*, respectively) relates two classes that are subclasses of Physical object (*POB*) and Social Object (*SOB*), respectively, which also share the *OD* or *OGD* relation. This is illustrated in Fig. 1.

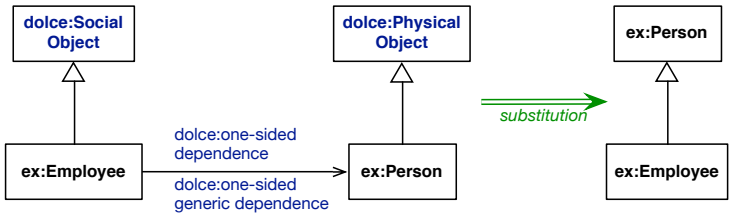

Figure 1. An example of the RR pattern substitution. Left: part of a DOLCE-aligned conceptual data model or domain ontology where either one-sided dependence or one-sided generic dependence is declared. Right: the resultant compacter representation after the substitution.

After formalising each pair of complex and simplified ontology patterns, the question is thus of how to achieve that algorithmically, and in such a way that the system is adaptable to addition of new pattern pairs.

3. Framework and Architecture

OP substitution in general consists of a notion of patterns, substitutions, and the processes for pattern instance identification and substitution. An OP is comprised of a templated set of axioms (also called axiom types) that must hold over an ontology. For OWL, identification of the satisfaction of axioms may be done using SPARQL or SPARQL-OWL queries, but for FOL and other logics, other approaches are needed. A general framework for substitution therefore requires first the specification of its key components, such

as the logic being operated on, its encoding, and the criterion for satisfaction and how such satisfaction is assessed. Taking these considerations into account, we declare the components of the framework to be as listed in Definition 1.

Definition 1 *An ontology pattern substitution framework consists of:*

- *The logic $\mathscr{L}$ and encoding (its particular syntax or serialisation) being used, L_s;*
- *A logical theory represented in language L_s, which is an ontology, O;*
- *A representation for axioms in the given logic, $\mathscr{A}$;*
- *A representation for patterns as constrained templated combinations of axioms, $\mathscr{P}$;*
- *A representation for substitutions $\mathscr{S}$ as pairs of valid pattern instances and their replacements;*
- *A set of pattern substitution specifications $\mathscr{U}$ such that for each pattern k there exists an element $u_k = \langle P_k, S_k \rangle$, a pair of pattern specifications for the pattern $P_k = \langle C_k, R_k, F_k \rangle$ and its substitution $S_k = \langle C'_k, R'_k, F'_k \rangle$, where:*
  - *each pattern specification $p \in \mathscr{P}$ is a tuple $\langle C, R, F \rangle$ of sets of elements (classes, relations, and axioms) of the ontology O, where:*
    - *$C \subseteq O_C$, a subset of classes in the ontology,*
    - *$R \subseteq O_R$, a subset of relationships in the ontology, which are used in a set of logical formulae F such that:*
    - *$F \subseteq O_A$, a subset of the axioms in the ontology using only the classes and relationships in C and R, respectively.*
- *An architecture for identifying pattern instances in $\mathscr{P}$ and substituting them using specifications in $\mathscr{U}$. Such an architecture must necessarily consist of, at least:*
  - *A manner of identifying valid axiom instances;*
  - *A manner of identifying valid pattern instances and generating substitutions from them;*
  - *A manner of applying those substitution transformations to an ontology.*

  where these manners are assumed to be algorithms. ♢

While this *framework* is fixed, certain *components* can be instantiated in many different ways for a particular application, in particular regarding the logic one chooses, which patterns to include for substitutions, the pattern-finding algorithms, and the 'manner', or ordering of, applying the substitutions. One could also conceive of an optional add-on to introduce a human-in-the-loop during the application of the substitutions, but this is strictly not necessary to solve the task at hand.

The set of pattern substitution specifications ($\mathscr{U}$) is extensible and contains formal specifications such as for the RR pattern of the running example. In this paper, we focus on foundational-to-applied pattern substitutions. Others are, among others, the following ones, illustrated with an example each (see supplementary material for the formal specifications):

- Class vs Object Property (COP): Marriage can be represented either as a class which a *Person participates in*, or directly as a relationship between two people.

- Perdurant Class vs Object Property (PCOP): One could model a marathon as something which a *Runner runs*, or reify running into a class *Running* that is *involved in* (part of) a *Marathon*.
- Class vs Data Property (CDP): In OWL a person having a skill could be modelled with an object property the range of which is a class *Skill*, or a data property *hasSkill* with data type *String*.
- Qualities vs Data Properties (QDP): an apple having a colour could be modelled with a data property with range *RGBvalue*, or could make use of a foundational ontology's classes and relationships to model the *Apple* as an *Endurant* which *has quality* some *Colour* which *has quale* some *physical region*.
- Class-Relationship-Attribute (CRA): a person could be modelled as having a name either by a relationship *has name* some class which in turn has a data property *name*, or by associating such a data property directly with the person class.
- Intrinsic descriptive property (IDP) truth-making pattern [29]: a rose having a colour could be modelled as a *Rose* which *participates in* some *ColorOccurence* which *has focus* some *Color* which *inheres in* the *Rose*, or as a *Rose* with a data property *color*.

Adapting the OP to FOL from the original formalisation in Description Logics [19], the pattern specification for the running example RR pattern is the tuple $\langle C, R, F\rangle$—where C denotes the unary predicates, R the n-ary predicates where $n \geq 2$, and F the set of axioms—that constitute the pattern, where, for the RR pattern:

$$
\begin{aligned}
C_r &= \{D, E, POB, SOB\} \text{ where } POB, SOB \in O_{DOLCE} \text{ and} D, E \in O_S \\
R_r &= \{OD, OGD\} \text{ where } OD, OGD \in O_{DOLCE} \\
F_r &= \{\forall x(D(x) \rightarrow POB(x)), \forall x(E(x) \rightarrow SOB(x)), \\
&\qquad \forall x(E(x) \rightarrow \exists y(D(y) \wedge OGD(x,y)))\}
\end{aligned}
$$

where D and E are unary relations in the source ontology O_S, such as Employee, and O_{DOLCE} the DOLCE ontology specifically. A similar pattern could be specified for other foundational ontologies. To demonstrate that the three axioms in F can be encoded into CLIF:

```
(forall (x) (if (D x) (POB x)))
(forall (x) (if (E x) (SOB x)))
(forall (x) (if (E x) (exists (y) (and (D y) (OGD x y)))))
```

The candidate substitution for the RR pattern is accordingly $\langle C', R', F'\rangle$ where:

$$
\begin{aligned}
C'_r &= \{D, E\} \\
R'_r &= \{\} \\
F'_r &= \{\forall x(D(x) \rightarrow E(x))\}
\end{aligned}
$$

which is encoded into CLIF as: `(forall (x) (if (D x) (E x)))`.

Another well-known OP substitution is n-nary into n binary relations, which could occur when converting an expressive ontology in FOL into OWL. Another one at the

expressiveness vs implementation pareto frontier includes simplifying property chains, such as from contain ∘ hasPart ⊑ contains to asserting contains directly, i.e., materialising the deductions and removing the original chain. They may be motivated also by language feature usage conflicts when integrating ontologies or converting an expressive ontology into a simpler one in a language that does not have property chains, not only from a foundational ontology-inspired style to an applied style.

4. Implementation and Evaluation

To test the framework, we designed an architecture, implemented it, tested it, and evaluated it, which is described in this section.

4.1. Design and implementation

For our instantiation of the framework and a proof-of-concept implementation, the logic used is FOL as encoded in CLIF. The architecture for the proof-of-concept application consists of the following components:

- A CLIF parser;
- A generator for axiom instance identifiers which takes a set of pattern specifications and generates binaries which take in an ontology and output axioms which are instances of the given axiom, with the templated variables extracted;
- A substitution generator which takes the set of valid axiom instances, identifies combinations of those which form valid pattern instances, and outputs the appropriate substitution for each in the form of ADD/DELETE commands;
- Encodings of the patterns [19] and substitutions generated for them ;
- A substitution program which executes ADD/DELETE commands in series on the original input ontology.

These components fit best together in a pipeline architecture to execute a sequence of steps in a set of modules, which is depicted in Fig 2.

For pattern instance detection, we devised one generic algorithm with an extensible set of *instancePatterns*, which goes through the ontology to find axiom matches (Algorithm 1). Each supported pattern has its own tailored algorithm; the one for the RR OP is included in Algorithm 2. Then, with the set of patterns found in the input ontology, it applies the pattern-specific substitutions.

Tooling for Common Logic is not as widespread as the software ecosystem of OWL. We resorted to the CLIF parser from HETS, the Heterogeneous ToolSet [28]. The proof-of-concept was implemented mainly in Python; however, since the only available CLIF parser useful for the task was implemented in Haskell, the generator creates and compiles Haskell programs for identifying axiom instances.

The correctness of the implementation has been evaluated against a series of synthetic test ontologies containing individual instances of the currently supported patterns. Here, we illustrate the algorithms for the running example.

4.2. Walking fully through an example

Let us take as example an ontology containing the axioms as listed at position ① in Fig. 3. Running Algorithm 1, it calls Algorithm 2, which finds this to be a formula in-

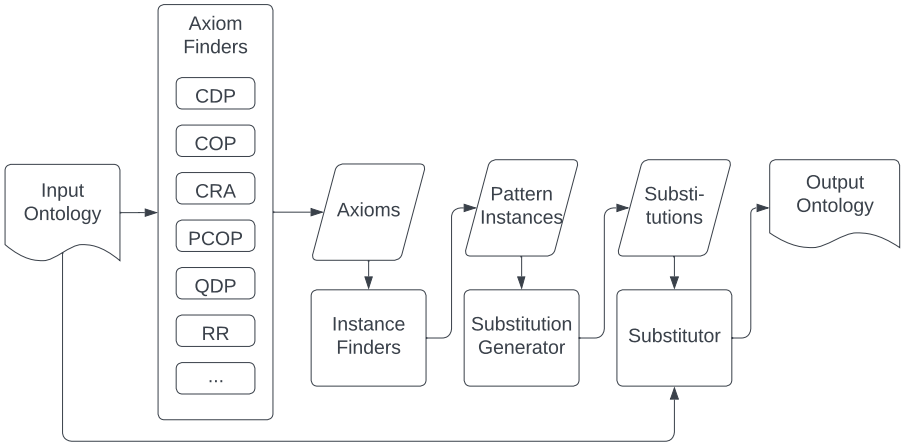

Figure 2. Proof-of-Concept pipeline architecture for the ontology pattern substitution. CDP, COP, CRA, PCOP, QDP, and RR are the abbreviations of the pattern substitutions currently included in the architectures (see text for details), which is, by design, extensible to additional substitution patterns.

Algorithm 1 Axiom instance detection

Require: $sentences, instancePatterns$

procedure FINDAXIOMINSTANCES($axioms, instancePatterns$)
  $m \leftarrow \emptyset$
  for $s \in axioms$ **do**
    for $(name, p) \in instancePatterns$ **do**
      if s matches p **then**
        $vars \leftarrow$ variables extracted from s ▷ by pattern matching
        $m \leftarrow m \bigcup \{(name, vars)\}$
      end if
    end for
  end for
end procedure

stance for RR and isolates the variables, after which the pattern instance checker finds that it is a pattern instance, and generates the substitution for it, and the substitution can be performed in turn.

The relevant function in the axiom checker matches on the syntactic structure of the axiom and ensures that the role of the second component of the `and` inside the first `exists` has the name `PC`, and returns the formula name and list of variables; see ② in Fig. 3. The output from the axiom checking step is as follows:

```
INSTANCE
Just ("RR_f1",["Person","POB"])
(forall (x) (if (Person x) (POB x)))
INSTANCE
Just ("RR_f2",["Employee","SOB"])
(forall (x) (if (Employee x) (SOB x)))
```

Algorithm 2 Get RR pattern instances

Require: $matches$

procedure GETRR($matches$)
  $instances \leftarrow \emptyset$
  $S_{f3} \leftarrow filter(matches, name = \text{"}RR\_f3\text{"})$
  for $f_3 \in S_{f3}$ **do**
    $f_1, f_2 = \langle\rangle, \langle\rangle$
    for $\langle name, objs, ax\rangle \in matches$ **do**
      if $name = \text{"}RR\_f1\text{"} \wedge objs[0] = f_3.objs[1]$ **then**
        $f_1 \leftarrow \langle name, objs, ax\rangle$
      end if
      if $name = \text{"}RR\_f2\text{"} \wedge objs[0] = f_3.objs[0]$ **then**
        $f_2 \leftarrow \langle name, objs, ax\rangle$
      end if
      if $f_1 \neq \langle\rangle \wedge f_2 \neq \langle\rangle$ **then**
        $instance \leftarrow \langle \text{"}RR\text{"}, f_3.objs, \{f_1.ax, f_2.ax, f_3.ax\}\rangle$
        $instances \leftarrow instances \cup \{instance\}$
      end if
    end for
  end for
  return $instances$
end procedure

Algorithm 3 Apply substitutions

Require: $substitutions, ontology$

procedure SUBPATT($substitutions, ontology$)
  $O_c \leftarrow copy(ontology)$
  for $\langle command, axiom\rangle \in substitutions$ **do**
    if $command = DELETE$ **then**
      Replace $axiom$ in O_c with $\emptyset$
    else if $command = ADD$ **then**
      Append $axiom$ to O_c
    end if
  end for
  return O_c
end procedure

```
INSTANCE
Just ("RR_f3",["Employee","Person","OGD"])
(forall (x) (if (Employee x)
            (exists (y) (and (Person y) (OGD x y)))))
```

The `INSTANCE` line marks the start of a new instance, followed by the particular formula identified and the variables extracted from it, and thereafter the exact text of the axiom identified (interrupted only by the next instance). When this is processed in the next stage, the function to identify RR instances checks for matches on `RR_f1`, `RR_f2`, and `RR_f3` and adds to the list of valid pattern instances if the extracted variables match, after

```
$ ./humusha build-instance-finders
Configured patterns: ['cdp', 'cop', 'cop_b', 'cra_a', 'cra_e', 'pcop', 'qdp', 'rr']
Instance finders built to: ./bin
$ ./humusha normalise-ontology  ./test_files/test_rr.clif
Normalised ontology written to: ./output/test_rr/test_rr.formatted.clif
$ ./humusha find-axiom-instances rr ./test_files/test_rr.clif                    2
Axiom instances written to: ./output/test_rr/test_rr.rr.instances
$ ./humusha make-substitutions rr ./test_files/test_rr.clif
Logs written to: ./output/test_rr/test_rr.rr.make_subs.log
Substitutions written to: ./output/test_rr/test_rr.rr.subs
$ ./humusha apply-substitutions rr ./test_files/test_rr.clif
Output ontology written to: ./output/test_rr/test_rr.sub_rr.clif
$ cat ./test_files/test_rr.clif
(forall (x) (if (Person x) (POB x)))                                             1
(forall (x) (if (Employee x) (SOB x)))
(forall (x) (if (Employee x) (exists (y) (and (Person y) (OGD x y)))))
$ cat ./output/test_rr/test_rr.rr.subs
COMMAND: DELETE
(forall (x) (if (Person x) (POB x)))

COMMAND: DELETE
(forall (x) (if (Employee x) (SOB x)))                                           3

COMMAND: DELETE
(forall (x) (if (Employee x) (exists (y) (and (Person y)
                                              (OGD x y)))))
COMMAND: ADD
(forall (x) (if (Employee x) (Person x)))

$ cat ./output/test_rr/test_rr.sub_rr.clif                                       4

(forall (x) (if (Employee x) (Person x)))
$
```

Figure 3. Proof-of-Concept tool applied to the running example, with key steps indicated (see text for details).

which the substitutions are generated (see ③ in Fig. 3) These are applied in sequence to the input ontology, rendering the output ontology (see ④ in Fig. 3).

4.3. Evaluation

The aim of the evaluation is to ascertain the correct functioning of the OP substitution architecture.

Our evaluation is carried out in two parts. We evaluate the full pipeline against a set of synthetic ontologies containing valid instances of each pattern and against a set of DOLCE-aligned ontologies converted from OWL to CLIF.

4.3.1. Materials

To ensure that each pattern is tested from detection through substitution, ontologies which are both guaranteed to contain the patterns and for which a correct substitution is easily predictable are necessary. To this end a set of test ontologies was created manually in CLIF, one per pattern containing one instance of that pattern, and one larger ontology combining them. The former contain minimal axioms necessary to constitute a pattern instance, while the latter contains the concatenation of all the individual patterns. The combined synthetic ontology was carefully constructed to ensure that various phenomena do not affect pattern detection or substitution, notably: interlacing axioms from various patterns to avoid grouping, the use of additional spurious axioms with similar structure to those expected to match, and use of the same class name in axioms of multiple patterns.

For the second part of the evaluation, a collection of DOLCE-aligned ontologies was sourced from the corpus[2] used in testing SUGOI [21], which was used to:

1. assess the occurrence of these patterns in real-world ontologies (and thus the immediate applicability of these pattern alignments);
2. ensure that the conversion pipeline conserves patterns present in OWL ontologies, allowing them to be detected.

Three of the 12 ontologies in the SUGOI DOLCE-aligned corpus failed to parse in HETS and thus could not be converted into CLIF ontologies, leaving nine with which to evaluate. The largest of them is a version of the data mining optimization ontology (DMOP).

4.3.2. Methodology

The test procedure for each synthetic test ontology is as follows:

1. Normalise the ontology text
2. Apply axiom instance finder binary for target pattern
3. Apply substitution generator to valid axiom instances
4. Run substitution commands over normalised input ontology
5. Compare output ontology with predicted result

The DOLCE-aligned ontologies are encoded in OWL, and so the above procedure is preceded by a step to first '0. convert the ontology from OWL to CLIF using HETS' and step 5 is replaced by '5. Inspect output and discuss'. Conversion with HETS was done with manual assistance as its output format is incompatible with its input parser, and as such some portions of the output required manual removal. The remainder of the process from normalisation of the CLIF representation through substitution was timed end-to-end using the bash `time` command, with the real time recorded to include both processing and I/O.

4.3.3. Results

The extraction of axioms and pattern instances, and computation and execution of substitutions are successful over all input ontologies. Fig. 3 displays the process in action for the RR pattern used in the running example. Furthermore, the execution times for both the synthetic and real-world ontologies are included in Table 2, demonstrating the tractability of the syntactic approach with FOL ontologies.

Of the real-world ontologies, MDMOP matched the CDP pattern in three places, generating three substitutions. The CDP pattern's original formulation in Description Logics refers to data properties, and thus is less applicable in FOL. The structure is nonetheless preserved upon conversion to FOL and is both detected and substituted.

To test that other patterns would also be found in these real-world ontologies, an instance of the RR pattern was introduced to MSceneOntology and the steps above repeated. The axioms added to the OWL ontology prior to conversion:

$$\texttt{Plant} \sqsubseteq \texttt{agentive-physical-object}$$

$$\texttt{PetPlant} \sqsubseteq \texttt{agentive-social-object}$$

$$\texttt{PetPlant} \sqsubseteq \exists \texttt{generically-dependent-on.Plant}$$

[2] Available at: `http://www.thezfiles.co.za/ROMULUS/ontologyInterchange.html`

The pattern was detected and correctly substituted:

```
COMMAND: DELETE
(forall (a) (if (Plant a) (agentive-physical-object a)))
COMMAND: DELETE
(forall (a) (if (PetPlant a)  (agentive-social-object a)))
COMMAND: DELETE
(forall (a) (if (PetPlant a) (exists (b)
        (and (generically-dependent-on a b) (Plant b)))))
COMMAND: ADD
(forall (x) (if (PetPlant x) (Plant x)))
```

To account for this apparent lack of pattern instances, we investigate the usage of the key classes and relationships from DOLCE in the ontologies by counting usage as assessed by Protégé for the original OWL files. As this metric includes imports, imported axioms from DOLCE that do not contain references to the domain ontology nonetheless contribute. As such, the presence of axioms containing the same classes as a pattern confirms only that the pattern may possibly be present. This data is presented in Table 1.

Inspecting the alignments, we observe several interesting cases. One concerns improper DOLCE reuse. For instance, and besides that the Naive Animal Ontology is multilingual and so most IRIs use a UUID and encode names in labels, DOLCE terms must have been manually added as one was spelled incorrectly (as Non-Agentive_Phisical-Object), and thus not picked up by our algorithm. Another issue is that several ontologies appear not to import DOLCE correctly, or have incorrect representations of the DOLCE taxonomy, such as physical object not having been declared a subclass of physical endurant in the Spatial Action ontology, but which it is in DOLCE. Also, in some cases, only a very small fragment of DOLCE turned out to have been reused, such as only physical object, physical quality and non-physical endurant out of the 37 classes in DOLCE, neither of which participates directly in either of the six patterns.

In addition, since the tool is using syntactic comparison, a granularity mismatch can cause a pattern to fail to be identified. As such, pattern specifications would require all relevant subclasses (here, all subclasses of the class specified within the pattern which are within the DOLCE taxonomy) to be included in the constraint specification for the pattern. Our tool can be configured to do this, but additional tooling is required to streamline the pattern specification process to make pattern specification manageable when the list of subclasses is long.

5. Discussion

The current algorithms cover the Class vs Object Property (COP), Perdurant Class vs Object Property (PCOP), Class vs Data Property (CDP), Qualities vs Data Properties (QDP), Representing Roles (RR), and Class-Relationship-Attribute (CRA) pattern pairs, and can execute the substitutions for these correctly. They work in isolation, but axiom patterns may overlap in real world ontologies. The current insights in ontology patterns, including ontology design patterns, do not provide sufficient insights into what to do in those cases. This may require a human-in-the-loop during the substitution process, or a ranking of priority on the pattern pairs. In addition, variability in axiom encoding limits simple comparison, requiring more sophisticated pattern matching.

Table 1. SUGOI corpus (DOLCE-aligned ontologies only) with the usage of relevant classes and relationships in the ontology. ED: endurant, PD: perdurant, APO: agentive-physical-object, ASO: agentive-social-object, pc: participates-in, gd: generically-dependent-on.

Ontology	ED	PD	APO	ASO	pc	gd	hasDataValue	has-quality	has-quale
DMOP	44	28	-	-	-	18	14	310	166
MDMOP	44	28	-	-	-	18	14	310	166
MNaive_animal_ ontology2	86	60	10	-	18	14	-	40	34
MOntoDerm	86	60	-	-	18	14	-	36	12
MSEGO	86	62	-	4	20	14	-	38	12
SceneOntology	-	-	-	-	12	44	-	10	9
MSceneOntology	-	-	-	-	18	18	-	36	12
PhysicalEntity	-	-	-	-	-	6	-	-	-
SpatialAction	-	-	-	-	-	-	-	-	-

Table 2. CLIF corpus of ontologies: axiom count and the time taken per pattern (ms).

ontology	# axioms	CDP	COP	CRA	PCOP	QDP	RR
Synthetic CDP	4	238					
Synthetic COP	1		229				
Synthetic CRA	1			214			
Synthetic PCOP	4				219		
Synthetic QDP	1					215	
Synthetic RR	3						213
Synthetic combined	16	196	193	190	215	231	232
DMOP	1174	469	448	462	461	469	467
MDMOP	3934	994	978	1003	1029	1010	1025
MNaive_animal_ontology2	1767	1099	1084	1183	1137	1129	1133
MOntoDerm_5.3	1318	517	448	482	458	461	431
MSEGOv3	971	405	400	398	431	395	447
MSceneOntology	1318	503	458	505	499	487	457
PhysicalEntity	320	271	281	261	280	289	286
SceneOntology	374	297	309	287	279	290	284
SpatialAction	85	237	225	220	234	218	214

Other considerations include the technologies used. Ontologies already in CLIF exhibited version mismatch. The change in encodings between the first [31] and second [32] edition practically means that being a `.clif` file is not enough to ensure that it can be parsed successfully. HETS' CLIF parser applies to first edition CLIF files, and thus so does our tool. More generally, while the expressiveness is a distinct advantage regarding pattern principles and precision of substitutions, tooling for CLIF (and FOL more generally) is limited. In addition, the representations are sufficiently heterogeneous that operating on native CLIF ontologies (cf. the output of OWL-to-FOL tools) requires different treatment depending on which CLIF version is being used and whether the ontology is CLIF-native or is part of a DOL ontology.

Finally, it is noteworthy that a corpus of DOLCE-aligned ontologies contains so few instances of the core ontology design patterns expected. In some cases technical obstacles such as the use of UUID IRIs or import failures prevented the assessment of the

presence of these patterns, but more research is necessary to determine why they are not present in the rest and, subsequently, how they can be aligned more systematically and comprehensively. In in broader view, this then also may facilitate ontology quality, promoting general, application-independent ontologies for the sought-after interoperability in a principled theory-based "foundational ontology style", and making it easier for ontology-driven information system developers through an automatic conversion into an 'applied style" of modelling that more closely matches the needs of a particular application whilst maintaining the links for interoperability and reuse.

6. Conclusion

We have introduced an ontology pattern substitution framework that enables alternative approaches to OP substitution, extensible to multiple logics and substitution patterns. This was realised in a proof-of-concept tool that applies syntactic pattern matching algorithms to successfully perform substitutions on FOL ontologies for six pattern alignments. Evaluation showed correctness of encoding and it revealed limited practical use of well-known foundational ontology-inspired modelling patterns for those ontologies aligned to DOLCE.

Future work includes permutations, such as devising similar foundational ontology-inspired patterns for BFO and UFO, and evaluation with a corresponding set of real-world ontologies. The approach may also be extended to work in different scenarios, such as expanding the algorithms to include identification of syntactically equivalent axioms ($a \wedge b$ vs $b \wedge a$, de Morgan, etc.) and embedding in other applications, such as the swapping an entire foundational ontology for another as in SUGOI [21] and developing a GUI with feedback for domain experts.

Acknowledgements This work was financially supported in part by the National Research Foundation (NRF) of South Africa (Grant Number 120852).

References

[1] Motik B, Patel-Schneider PF, Parsia B. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. W3C; 2009. `http://www.w3.org/TR/owl2-syntax/`.

[2] Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, et al. Gene Ontology: Tool for the Unification of Biology. Nature genetics. 2000 May;25(1):25-9.

[3] The Gene Ontology Consortium, Carbon S, et al. The Gene Ontology Resource: Enriching a GOld Mine. Nucleic Acids Research. 2021 Jan;49(D1):D325-34.

[4] Tudorache T. Ontology Engineering: Current State, Challenges, and Future Directions. Semantic Web. 2020 Jan;11(1):125-38.

[5] Keet CM. An introduction to ontology engineering. vol. 20 of Computing. UK: College Publications; 2018. 334p.

[6] Gruninger M, Chui C, Katsumi M. Upper Ontologies in COLORE. In: Borgo S, et al., editors. Proceedings of the Joint Ontology Workshops 2017. vol. 2050 of CEUR-WS; 2017. .

[7] Artale A, Calvanese D, Kontchakov R, Ryzhikov V, Zakharyaschev M. Reasoning over Extended ER Models. In: Parent C, et al., editors. Proc. of ER'07. vol. 4801 of LNCS. Springer; 2007. p. 277-92. Auckland, New Zealand, Nov 5-9, 2007.

[8] Berardi D, Calvanese D, De Giacomo G. Reasoning on UML class diagrams. Artificial Intelligence. 2005;168(1-2):70-118.

[9] Franconi E, Mosca A, Solomakhin D. The formalisation of ORM2 and its encoding in OWL2. Faculty of Computer Science, Free University of Bozen-Bolzano, Italy; 2012. KRDB12-2. Available from: `http://www.inf.unibz.it/krdb/pub/TR/KRDB12-2.pdf`.

[10] Pan WL, Liu Dx. Mapping Object Role Modeling into Common Logic Interchange Format. In: Proc. of ICACTE'10. vol. 2. IEEE Computer Society; 2010. p. 104-9.

[11] Braga BFB, Almeida JPA, Guizzardi G, Benevides AB. Transforming OntoUML into Alloy: towards conceptual model validation using a lightweight formal methods. Innovations in Systems and Software Engineering. 2010;6(1-2):55-63.

[12] Jahangard Rafsanjani A, Mirian-Hosseinabadi SH. A Z Approach to Formalization and Validation of ORM Models. In: Ariwa E, El-Qawasmeh E, editors. Digital Enterprise and Information Systems. vol. 194 of CCIS. Springer; 2011. p. 513-26.

[13] Arp R, Smith B, Spear AD. Building Ontologies with Basic Formal Ontology. USA: The MIT Press; 2015.

[14] Calvanese D, Cogrel B, Komla-Ebri S, Kontchakov R, Lanti D, Rezk M, et al. Ontop: Answering SPARQL queries over relational databases. Semantic Web Journal. 2017;8(3):471-87.

[15] Fillottrani PR, Keet CM. Dimensions Affecting Representation Styles in Ontologies. In: Villazón-Terrazas B, Hidalgo-Delgado Y, editors. Knowledge Graphs and Semantic Web. vol. 1029 of CCIS. Springer; 2019. p. 186-200.

[16] Jarrar M, Demy J, Meersman R. On Using Conceptual Data Modeling for Ontology Engineering. Journal on Data Semantics. 2003;1(1):185-207.

[17] Sugumaran V, Storey VC. The Role of Domain Ontologies in Database Design: An Ontology Management and Conceptual Modeling Environment. ACM Transactions on Database Systems. 2006;31(3):1064-94.

[18] Khan ZC, Keet CM. ROMULUS: a Repository of Ontologies for MULtiple USes populated with foundational ontologies. Journal on Data Semantics. 2016;5(1):19-36.

[19] Fillottrani PR, Keet CM. Patterns for Heterogeneous TBox Mappings to Bridge Different Modelling Decisions. In: Blomqvist E, et al., editors. Proc. of ESWC'17. vol. 10249 of LNCS. Springer; 2017. p. 371-86. 30 May - 1 June 2017, Portoroz, Slovenia.

[20] Thiéblin E, Haemmerlé O, Hernandez N, Trojahn C, Sabou M. Survey on complex ontology matching. Semantic web. 2020;11(4):689-727.

[21] Khan ZC, Keet CM. Feasibility of automated foundational ontology interchangeability. In: Janowicz K, Schlobach S, editors. Proc. of EKAW'14. vol. 8876 of LNAI. Springer; 2014. p. 225-37. 24-28 Nov, 2014, Linkoping, Sweden.

[22] Zamazal O, Svátek V. Patomat: Versatile Framework for Pattern-Based Ontology Transformation. Computing and Informatics. 2015;34(2):305-36.

[23] Halpin T, Morgan T. Information modeling and relational databases. 2nd ed. Morgan Kaufmann; 2008.

[24] Freitas F, Lins F. The Limitations of Description Logic for Mathematical Ontologies: An Example on Neural Networks. In: ONTOBRAS-MOST; 2012. p. 84-95.

[25] Gómez Álvarez L, Rudolph S. Standpoint Logic: Multi-Perspective Knowledge Representation. In: Neuhaus F, Brodaric B, editors. Proc. of FOIS'21. vol. 344 of FAIA. IOS Press; 2021. p. 3-17.

[26] Kovács L, Voronkov A. First-Order Theorem Proving and Vampire. In: Hutchison D, et al, editors. Computer Aided Verification. vol. 8044. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 1-35.

[27] Sutcliffe G. The TPTP Problem Library and Associated Infrastructure. Journal of Automated Reasoning. 2017 Dec;59(4):483-502.

[28] Mossakowski T, Maeder C, Lüttich K. The Heterogeneous Tool Set, Hets. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer; 2007. p. 519-22.

[29] Guarino N, Sales TP, Guizzardi G. Reification and Truthmaking Patterns. In: Proc. of ER'18. Springer; 2018. p. 151-65.

[30] Masolo C, Borgo S, Gangemi A, Guarino N, Oltramari A. WonderWeb Deliverable D18 Ontology Library (Final). ICT project. 2003;33052:31.

[31] International Organization for Standardization. Common Logic (CL): A Framework for a Family of Logic-Based Languages; 2007. Available from: `https://www.iso.org/standard/39175.html`.

[32] International Organization for Standardization. Common Logic (CL): A Framework for a Family of Logic-Based Languages; 2018. Available from: `https://www.iso.org/standard/66249.html`.