Ontology Design Pattern Property Specialisation Strategies

Karl Hammar[1,2]

[1] Information Engineering Group, Jönköping University, Sweden
[2] Department of Computer and Information Science, Linköping University, Sweden
karl.hammar@jth.hj.se

Abstract. Ontology Design Patterns (ODPs) show potential in enabling simpler, faster, and more correct Ontology Engineering by laymen and experts. For ODP adoption to take off, improved tool support for ODP use in Ontology Engineering is required. This paper studies and evaluates the effects of strategies for object property specialisation in ODPs, and suggests tool improvements based on those strategies. Results indicate the existence of three previously unstudied strategies for ODP specialisation, the uses of which affect reasoning performance and integration complexity of resulting ontologies.

1 Introduction

Content Ontology Design Patterns (hereafter ODPs) were introduced by Gangemi [7] and Blomqvist & Sandkuhl [3] in 2005 (extending upon ideas by the W3C Semantic Web Best Practices and Deployment Working Group[3]), as a means of facilitating practical ontology development. These patterns are intended to help guide ontology engineering work by non-expert users, by packaging best practices into small reusable blocks of ontology functionality, to be adapted and specialised by those users in their individual ontology development use cases. Studies indicate that ODP usage can help lower the number of modelling errors and inconsistencies in ontologies, and that they are by the users perceived as useful and helpful [2, 5].

This idea has gained some traction within the academic community, as evidenced by the Workshop on Ontology Patterns series of workshops. However, the adoption of ODPs among practitioners is still quite limited. In order to support increased adoption and use of ODPs (and potentially, as a result thereof, increased adoption and use of Semantic Web technologies and ontologies in general), methods and tooling supporting their usage are required. While there are available tools for ODP use (e.g., the XD Tools for NeOn Toolkit), the functionality of these tools is limited; they are based on their authors' largely intuitive understanding of ODP usage and practices at the time when ODPs were first proposed. In order to improve tooling to support today's users, an empirically founded understanding of how ODPs are actually used today is needed.

[3] http://www.w3.org/2001/sw/BestPractices/

The author's present research concerns the development of such improved ODP tooling based on published ODPs and ontologies. In this paper, the focus is on the choices that users face when specialising an ODP, and the potential consequences those choices may give rise to. The work was initiated using the following research question:

– How are Content Ontology Design Patterns used or specialised in Ontology Engineering projects for the Semantic Web, and what are the effects of such usage?

In studying this rather generally expressed question, as further described in Section 3.1, several ODP application strategies were observed (listed and discussed in Section 3.2). The discovery of these strategies gave rise to more specific research questions on their usages and effects:

1. To what degree are the class-oriented, property-oriented, or hybrid ODP specialisation strategies used in published ODPs and ontologies?
2. What are the reasoning performance effects of specialising ODPs and ontologies in accordance with the class-oriented or property-oriented strategies?

The main contributions of this paper are a classification of three different strategies for ODP specialisation based on the semantics of object property domains and ranges, and a partial understanding of the consequences of employing two of these specialisation strategies. Additionally, the paper contributes suggestions on how to implement tooling supporting users in specialising ODPs based on a strategy that is consistent with their preferences and goals.

The paper is structured as follows: Section 2 introduces the background and some related work in the area. Section 3 describes the ODP specialisation strategies and the method by which they were discovered. Section 4 details a study on the use of these strategies in published ontologies, and the effects of such uses. Section 5 discusses the consequences of these findings for the development of ODP support tools. Section 6 concludes the paper by summarising the answers to the posed research questions.

2 Background and Related Work

Ontology Design Patterns were introduced at around the same time independently by Gangemi [7] and Blomqvist and Sandkuhl [3]. The former define such patterns by way of a number of characteristics that they display, including examples such as *"[an ODP] is a template to represent, and possibly solve, a modelling problem"* [7, p. 267] and *"[an ODP] can/should be used to describe a 'best practice' of modelling"* [7, p. 268]. The latter describes ODPs as generic descriptions of recurring constructs in ontologies, which can be used to construct components or modules of an ontology. Both approaches emphasise that patterns, in order to be easily reusable, need to include not only textual descriptions of the modelling issue or best practice, but also some formal encoding of the proposed solution.

The understanding of Ontology Design Patterns has been heavily influenced by the work taking place in the NeOn Project, the results of which include a pattern typology [13]. This typology is based on the uses to which patterns are put, whether they represent best practices in reasoning, naming, transformation, content modelling, etc. This paper focuses exclusively on Content patterns (the most common type of patterns). Content patterns represent some content in the domain of discourse, and their formal representations are typically packaged as reusable mini-ontologies in OWL format.

Ontology Design Patterns have also been studied within the CO-ODE project[1, 6], the results of which include a repository of patterns[4] and an Ontology Pre-Processing Language (OPPL)[5]. The patterns proposed and developed in these works are also Content ODPs, but they are generally more abstract or foundational in nature than those developed within the NeOn project. These patterns also double as transformation patterns, since they can be used in conjunction with the OPPL macro language to enable rapid transformation of large ontologies.

2.1 eXtreme Design

The eXtreme Design (XD) collaborative ontology development method, developed within the NeOn Project, is based on the use of Ontology Design Patterns [4]. XD is defined as *"a family of methods and associated tools, based on the application, exploitation, and definition of Ontology Design Patterns (ODPs) for solving ontology development issues"* [12, p. 83]. The method is influenced by the eXtreme Programming (XP) agile software development method, and like it, emphasises incremental development and continuous requirements management. Like XP it also recommends pair development, test driven development, refactoring, and a divide-and-conquer approach to problem-solving [11].

The XD method consists of a number of tasks, illustrated in Figure 1. The main development tasks are performed in iterating loops; these tasks are in the figure enclosed in a grey box. The "Reuse and integrate ODPs" task is where the developer specialises a selected ODP (representing a general reusable solution) to the specific modelling scenario. It is the choices that the developer need to make when performing this task, which are explored in the scope of this paper.

2.2 XD Tools

The XD method is supported by the XD Plugin for the Eclipse-based NeOn Toolkit[6]. The XD Plugin provides a number of components that simplify pattern browsing, selection, and specialisation. The specialisation component provides a wizard interface for specialising ODPs, consisting of three steps:

1. Specialising leaf classes of the class hierarchy by defining subclasses

[4] `http://odps.sourceforge.net/odp/html/index.html`
[5] `http://oppl2.sourceforge.net/`
[6] `http://neon-toolkit.org/`

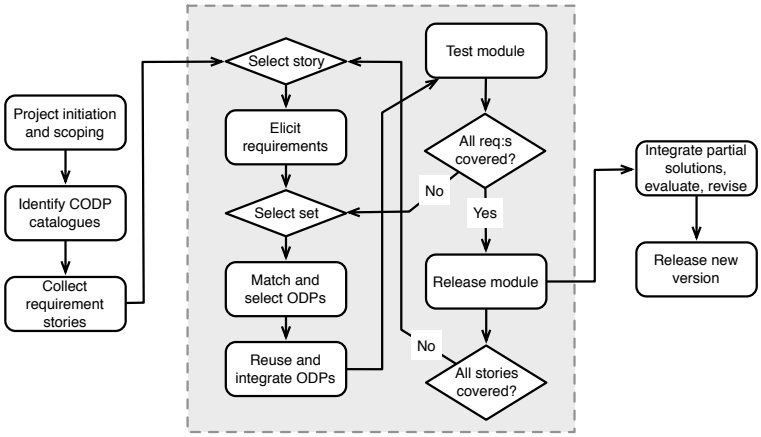

Fig. 1. XD Workflow (adapted from [12])

2. Specialising leaf properties of the property hierarchy by defining subproperties
3. Defining domains and ranges of newly defined subproperties to correspond with the new subclasses

The specialisation wizard also provides a certain degree of validation of the generated specialisations, by presenting the user with a list of generated axioms, expressed in natural language, for the user to reject or accept. Unfortunately, the XD Tools plugin's dependence on the NeOn Toolkit (which is no longer developed or maintained) means that ontology engineers who want to use newer tools and standards are unable to use XD Tools. Instead, they have to do their ODP-based ontology engineering without adequate tool support.

3 Understanding ODP Specialisation

In order to answer the initial research question, a two-part method was employed. Initially, a set of ODP-using ontologies were studied in order to extract commonalities or strategies regarding how ODPs were specialised. The following two subsections describe this process, and the ODP specialisation strategies discovered by it. Subsequently, the usage of those specialisation strategies among ontologies and the consequences of such use, were evaluated. These latter evaluations are described in section 4.

3.1 Study Setup

Ontologies making use of ODPs first had to be located and downloaded. For this purpose, a method combining several different sources of ontologies was

employed. The initial set of ODP-using patterns was retrieved using the Google Custom Search API[7]. This API was queried repeatedly, using all known ODP URIs; the results were downloaded, filtered based on type, and only such results that held both one or more instances of *owl:Ontology* and one or more references to known ODP namespaces were kept. Additionally, the LODStats[8] list of RDF dataset vocabularies, the Linked Open Vocabularies[9] dataset, and the known uses and instantiations of ODPs from OntologyDesignPatterns.org[10] were added to this set (the same criteria for filtering were employed). This resulted in 22 ODP-using OWL ontologies being found and downloaded. Additionally, a set of 19 such ontologies originating with the IKS[11] project were added to the set[12].

From these 41 ontologies, 107 *specialisation mapping axioms*, that is, subsumption or equivalence axioms linking a class or property defined in the ontology to a class or property defined in a known ODP, were extracted. These mapping axioms were analysed for recurring patterns based on the features of the ODP class or property being specialised, and based on the type of mapping properties used. An excerpt of the set of extracted mapping axioms is displayed in Table 1.

Table 1. Excerpt from the set of extracted ODP specialisation mapping axioms.

ODP Class/Property	**Role in specialisation**	**Occurrences**
place.owl#Place	superclass	4
parameter.owl#Parameter	superclass	2
collectionentity.owl#hasMember	superproperty	1
bag.owl#hasItem	superproperty	1
bag.owl#hasItem	used in property restriction	4

Table 2 summarises the type of mappings used in the gathered data. As the table shows, simple mapping using *rdfs:subClassOf* and *rdfs:subPropertyOf* predicates against ODP named classes is the most common, together accounting for 85 % of all specialisation axioms. In all but a handful of these uses, ODP classes and properties act as superclasses and superproperties to specialised classes and properties. Equivalency mappings against named ODP concepts are used less often; no uses of *owl:equivalentProperty* are observed at all, and *owl:equivalentClass* is only used in a few cases.

The use of existential or universal quantification restrictions involving ODP classes and properties is worth noting. In the studied set of ontologies, such restrictions are used to constrain the uses of ODP object properties, locally em-

[7] `https://developers.google.com/custom-search/`
[8] `http://stats.lod2.eu/`
[9] `http://lov.okfn.org/`
[10] `http://ontologydesignpatterns.org/`
[11] `http://www.iks-project.eu/`
[12] Ontologies and scripts used in this paper are available via `http://goo.gl/jjU9OE`

Table 2. ODP Specialisation Mapping Axioms Summary

Mapping axiom type	Occurrences
rdfs:subClassOf against named ODP class	32
rdfs:subPropertyOf against named ODP property	59
owl:equivalentClass against named ODP class	3
owl:equivalentProperty against named ODP property	0
rdfs:subClassOf against value constraint over ODP property	7
owl:equivalentClass against value constraint over ODP property	2
rdfs:subClassOf against value constraint over ODP class	4
owl:equivalentClass against value constraint over ODP class	0

ulating domain or range axioms; for instance, a WeatherForecast is defined as being equivalent to the union of two restrictions, one using a project-defined vocabulary, and one on the WeatherForecast being an information realisation (i.e., EquivalentClass (WeatherForecast objectSomeValuesFrom(informationRealization:realizes WeatherInformation))).

Such a use of restrictions to constrain the local semantics of object properties can be seen as a form of specialisation of a more general model, or ODP, for a particular modelling case. This observation leads us to consider how this type of specialisation strategy differs from the more common strategy of specialising subproperties with defined domains and ranges, as supported by the existing XD Tools. In order to develop tool support for the use of property restrictions in ODP specialisation, the consequences of applying this type of modelling need be studied, such that users can be informed of the potential effects of applying either the traditional (hereafter denoted *"property-oriented"*, due to the use of subproperties) strategy for ODP specialisation, or the alternative restriction-based strategy (hereafter denoted *"class-oriented"*, due to the use of property restrictions on subclasses).

3.2 Study Results

The following section discusses and exemplifies the two initially discovered strategies of ODP specialisation. The possibilities of and the consequences of combining the two strategies in a third, hybrid strategy, are also discussed. Two things are important to note. Firstly, an *ODP specialisation* is here defined as the set of specialisation mapping axioms that together specialise a single ODP for use in a target ontology. In most cases each ODP specialisation will consist of several specialisation mapping axioms, each specialising different classes or properties of the ODP. Often this set of axioms will be held in an ontology module imported into the target ontology, an *ODP specialisation module*. Secondly, in discussing the relative usage frequency of the strategies, we here only compare those specialisations that modify the semantics of object properties defined in the original ODP (simple class taxonomies without any object property specialisation have been filtered out), and we also include specialisations of ODP specialisations; as

OWL imports are transitive, this type of layered structure is not uncommon. This gives a total of 20 ODP specialisations, the distributions of which over the specialisation strategies are summarised in Table 3.

Table 3. ODP Specialisation Strategy Use

Specialisation strategy	Occurrences
Property-oriented	9
Class-oriented	6
Hybrid	5

Property-oriented Strategy The property-oriented strategy is the most common type of ODP specialisation seen in the originally studied set of ODP specialisations, being used in 9 out of 20 cases. This may be due to the fact that OWL tools and tutorials tend to emphasize properties as basic language features, and the construction of property subsumption hierarchies specialising those properties as fundamental modelling tasks.

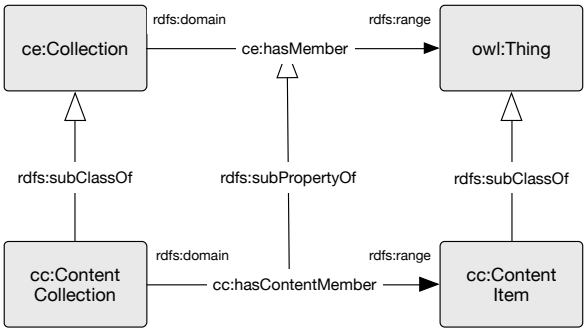

Fig. 2. Property-oriented ODP Specialisation Strategy

The process by which an ODP is specialised in accordance with the property-oriented strategy is illustrated and exemplified in Figure 2 (the example is taken from an ontology developed in the IKS project). In the figure, the *ce:* namespace prefix indicates that classes or properties are defined within the CollectionEntity ODP[13], whereas the *cc* namespace prefix indicates that classes or properties are defined within the ODP specialisation module ContentCollection. We see that

[13] `http://ontologydesignpatterns.org/wiki/Submissions:CollectionEntity`

the higher level class definitions defined in the ODP or OWL language itself (*ce:Collection*, *owl:Thing*) are specialised for the modelling problem at hand using newly defined subclasses (*cc:ContentCollection*, *cc:ContentItem*), and that the usage of *ce:hasMember* is specialised to apply to these new class definitions via a new subproperty *cc:hasContentMember* with corresponding domain and range declarations. As shown by [10], the *rdfs:subPropertyOf* definition implies that domains and ranges of subproperties must be subsets of the domains and ranges of their superproperties.

It should be noted that this specialisation strategy does not necessarily need to be fully instantiated in all parts; there are several cases where only one subclass is created, and where either the domain or range of the created subproperty are therefore defined over a more general term. Indeed, that is the way in which the CollectionEntity ODP itself is structured in Figure 2, with *ce:hasMember* having a range of *owl:Thing*. The important thing about this strategy, and the key differentiator against the class-oriented strategy, is the definition of a subproperty.

Class-oriented Strategy The class-oriented strategy is a little less common in the studied set of ODP specialisations, being seen in 6 of 20 cases. The fundamental idea of this strategy is to avoid creating specialised subproperties, by instead reusing the object properties of the original ODP, and locally constraining the usage of those properties by way of property restrictions on specialised classes. As shown by Horridge et al. [8] the definitions SubClassOf(A ObjectAllValuesFrom(someProperty B)) imposes a local range of B on the property *someProperty* for the class A. Using the same approach, we can represent a local domain of A over the property *someProperty* where the target of that property is B, by defining EquivalentClass(A ObjectSomeValuesFrom(someProperty B)).

The concept is illustrated in Figure 3. The namespaces used in the figure are *ir:*, denoting the Information Realization ODP[14], and *wf:*, denoting the Weather Forecast ODP specialisation module. In this example a *wf:WeatherForecast* class is defined in terms of how its member individuals are connected via the *ir:realizes* property to members of the *wf:WeatherInformation* class. In layman terms, the *owl:equivalentClass* property restriction imposes the condition that any individual which is connected via *ir:realizes* to some other individual that is a *wf:WeatherInformation*, must itself be a *wf:WeatherForecast*; this restriction corresponds to the use of a *rdfs:domain* definition over a specialised property in the property-oriented strategy. Similarly, the *rdfs:subClassOf* property restriction imposes the condition that only individuals that are *wf:WeatherInformation* may be linked via *ir:realizes* from an individual that is a *wf:WeatherForecast*; this corresponds to a rdfs:range axiom in the property-oriented strategy.

Note that the use of the term *corresponds to* above does not imply that the described uses of property restriction axioms are logically equivalent to the use of domain or range axioms; merely that both modelling strategies allow

[14] `http://ontologydesignpatterns.org/wiki/Submissions:Information_realization`

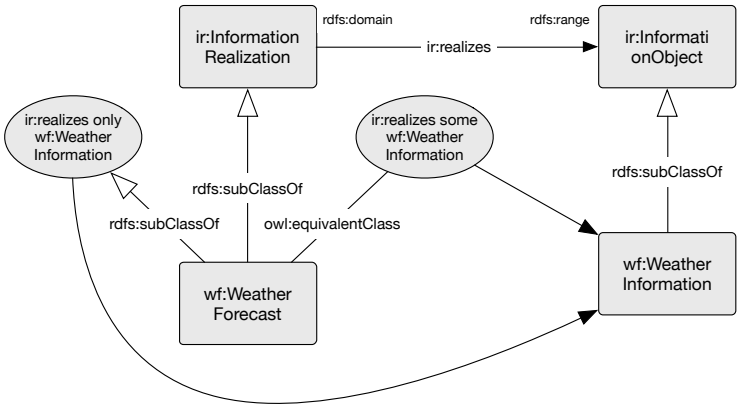

Fig. 3. Class-oriented ODP Specialisation Strategy (rounded rectangles denote named classes, ovals denote property restriction axioms)

for describing similar phenomena, via expressing constraints on which types of entities that object properties can connect.

Hybrid Strategy In addition to the pure property-oriented or pure class-oriented strategies for ODP specialisation, the combination of the two in a hybrid strategy approach also occurs in the set of ODP specialisations: 5 of the 20 ODP specialisations use such a hybrid strategy. In these specialisations, subproperties with domain and range declarations are defined, and the classes involved are also defined using universal and/or existential quantification restrictions ranging over the newly created subproperties. The latter restriction axioms are possibly redundant if they are in this manner defined over properties that have themselves got domains and ranges. They could however be helpful from a usability perspective, in a large ontology of taxonomic nature, i.e. with a large class subsumption hierarchy, where one wants readers of said ontology to be able to easily grasp how classes are intended to be interconnected without having to study the property subsumption hierarchy.

4 Strategy Usage and Effects

As seen in Section 3.1, the number of published ODP specialisations in which object property specialisation takes place is limited; only 20 such cases were found. This is too small a dataset to base conclusions on. This section broadens the scope from just ODP specialisations to ontologies in general. If the same strategies are also observed in a larger set of ontologies, this strengthens the notion that ontology engineers need tool support for these strategies. Similarly,

if the effects of applying these strategies can be observed on ontologies in general, ODP specialisation tooling need take this into account, and guide the user accordingly when constructing ontologies using ODPs. Section 4.1 describes a study on to what extent the strategies are employed in ontologies published on the web. Section 4.2 discusses the effects of using the property-oriented or class-oriented strategies, and details a benchmark-based evaluation of the reasoning performance effects of such use.

4.1 Strategy Use

The ontologies studied were gathered in the same manner as described in Section 3.1. While we previously selected only those downloaded RDF files that held instances of *owl:Ontology* and that had references to known ODP namespaces, in this case the latter selection filter was dropped, and all downloaded graphs containing OWL ontologies were kept. This resulted in 423 ontologies for study.

```
   Input: graph = An RDF graph containing an owl:Ontology
 1 restrictionProperties = List();
 2 hasDomainOrRange = List();
 3 for subProperty defined in graph do
 4 |   if hasDomain(subProperty) then
 5 |   |   Append(hasDomainOrRange,subProperty);
 6 |   if hasRange(subProperty) then
 7 |   |   Append(hasDomainOrRange,subProperty);
 8 for class defined in graph do
 9 |   for superClass in getSuperClass(class) do
10 |   |   if hasPredicate(superClass, owl:allValuesFrom) then
11 |   |   |   restrictionProperty = getObject(superClass,owl:onProperty);
12 |   |   |   Append(restrictionProperties,restrictionProperty);
13 |   for equivalentClass in getEquivalentClass(class) do
14 |   |   if hasPredicate(equivalentClass, owl:someValuesFrom) then
15 |   |   |   restrictionProperty = getObject(equivalentClass,owl:onProperty);
16 |   |   |   Append(restrictionProperties,restrictionProperty);
17 if Overlap(Set(hasDomainOrRange),Set(restrictionProperties)) > 0 then
18 |   Return(Hybrid strategy);
19 if Size(hasDomainOrRange) == 0 AND Size(restrictionProperties) == 0 then
20 |   Return(No property specialisation occurring);
21 if Size(hasDomainOrRange) > Size(restrictionProperties) then
22 |   Return(Property-oriented strategy);
23 else
24 |   Return(Class-oriented strategy);
```

Algorithm 1: Detects ontology property specialisation strategy.

Algorithm 1 was then executed over the ontology set. Per the algorithm, the number of specialised object properties that have domain or range definitions are

compared against the number of properties that occur in a restriction emulating a local domain or range, and the ontology as a whole is classified based on the most commonly occurring type of structure. In the case that an overlap exists between these two sets, that is, that there are individual object properties that are specialised in both ways, the ontology is classified as employing the hybrid strategy. The results of this classification are summarised in Table 4 (simple taxonomies and alignment ontologies not defining any own OWL constructs have been filtered out of the results).

Table 4. Ontology Specialisation Strategy Use

Specialisation strategy	**Occurrences**
Property-oriented	193
Class-oriented	33
Hybrid	23
No property specialisation occurring	98

These results indicate that all three object property specialisation strategies discovered in Section 3.2 to some extent also occur in ontology engineering where ODPs are not used. The results also indicate that the property-oriented strategy is by a significant margin the most commonly used strategy. Comparing against the previously studied ODP specialisations (Table 3), we see that the class-oriented and hybrid strategies are used less frequently in ontologies (13 % vs 30 % of cases for the former, 9 % vs 25 % of cases for the latter). This suggests there may be a difference in how ontology engineering is performed when using ODPs as compared to in the general case.

4.2 Strategy Effects

An advantage of the property-oriented strategy is that it creates new subproperties, which can themselves be dereferenced and annotated or typed as needed. For instance, such a specialised subproperty could be defined to be transitive or functional, without this definition affecting the parent property. Another advantage is that this type of modelling is accessible from a usability perspective; the simple tree view of the property subsumption hierarchy as used in many tools enables the ontology engineer or end-user to get an at-a-glance understanding of how the properties are organised and intended to be used. Yet another advantage is that, given that domains and ranges are defined, inferring the type of individuals connected via the property is a relatively fast operation, when compared to the class-oriented strategy.

The main advantage of the class-oriented specialisation strategy is that no subproperties are created, but rather that the original parent properties are reused. This allows RDF datasets that are expressed in accordance with an ontology using this strategy to be natively interoperable with other datasets

using the same property, without the need for reasoning. This is particularly relevant in an ODP context, where the ODP and its properties are intended be reused extensively. Such interoperability can have many advantages, including in querying, where SPARQL triple patterns will often define only the predicate used and leave subject and object variables unbound. Further, this strategy allows for modelling typing based on property links, much like *duck typing* in programming; such an approach can have advantages in situations where the ontology engineer does not control the predicates used in data creation or extraction, but rather has to deal with what they are given.

There are also downsides to this strategy, most noticeably in terms of reasoning performance. As pointed out by Horridge et al. [8], universal quantification axioms, used in this strategy to emulate *rdfs:range* definitions, are disallowed in the computationally friendly OWL 2 EL profile. As illustrated by Urbani et al. [15], using property restrictions to infer typing requires multiple joins between large sets of candidate entities, greatly complicating reasoning, particularly when dealing with large datasets. The results of Kang et al. [9] also indicate that there may be performance penalties associated with this type of reasoning. Their predictive model for reasoning performance includes eight ontology metrics categorised as impacting or strongly impacting reasoning performance; of these, three (the number of existential quantifications, average class out-degree, and subsumption tree impurity) are increased by employing the class-oriented strategy, as opposed to the property-oriented one. The strongest impact factor of any metric in their model is the number of existential quantification axioms, which are heavily used in this modelling strategy.

Reasoning Performance Evaluation In order to evaluate the reasoning performance effects of the class-oriented and property-oriented specialisation strategies, an experiment was set up using the well known LUBM[15] and BSBM benchmarks[16]. The hypothesis was that, due to the above mentioned characteristics of the two strategies, the execution of reasoning tasks on datasets using ontologies adhering to the property-oriented strategy would be faster than on the same datasets using ontologies adhering to the class-oriented strategy.

Each of the two benchmark suite ontologies were adapted to both the property-oriented and class-oriented strategies, via replacing domain and range axioms by universal and existential quantification restrictions or vice versa (in the case of BSBM, as an OWL ontology is not provided, said ontology first had to be created from scratch using the BSBM Dataset Specification). Datasets of non-trivial size (LUBM: 1053084 triples, BSBM: 334479 triples) were then generated using each benchmark suite's data generator. In order to make the performance evaluation tasks which include inferring typing axioms non-trivial, RDF typing axioms generated by the benchmark data generators were removed.

The datasets were then used together with the property- and class-oriented benchmark ontologies as input for two leading OWL reasoners, Pellet (version

[15] `http://swat.cse.lehigh.edu/projects/lubm/`
[16] `http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/`

2.3.1) and HermIT (version 1.3.8). The operations performed were consistency checking (ensuring that no contradictory axioms exist in the datasets and ontologies), and realization (finding the most specific class that a named individual belongs to). As the HermiT reasoner does not support performing realization from the command line, it was only used to perform consistency checking over the two datasets. The experiments were executed on a quad-core 2.6 GHz Intel Core i7 machine with an 8 GB Java heap size, running Mac OS X 10.9.3.

Table 5. Specialisation strategy realisation performance effects.

Reasoning task	**Benchmark**	**Reasoner**	**PO time**	**CO time**
Consistency checking	BSBM	Pellet	1.274 s	1.897 s
Consistency checking	BSBM	HermiT	1.984 s	27.193 s
Consistency checking	LUBM	Pellet	8.230 s	42.887 s
Consistency checking	LUBM	HermiT	10.097 s	46 min, 17 s
Realising individuals	BSBM	Pellet	2.389 s	9.482 s
Realising individuals	LUBM	Pellet	1.801 s	4+ hours

As illustrated in Table 5, the hypothesis holds for the generated datasets. In all of the reasoning tasks performed, the use of class-oriented ontologies resulted in slower execution than the use of property-oriented ontologies. In most cases the effects were severe; in one case execution of the reasoning task was halted when no results were reported after 4 hours of continuous execution. It should be noted that the inferred axioms of the reasoning tasks were equivalent, regardless of which strategy the ontology in question used.

5 Discussion

As shown above, the different strategies for object property specialisation are used both in the specialisation of ODPs and in specialisation of ontologies. The consequences of selecting one or the other of the strategies can be significant, and there is a trade-off to make: the class-oriented strategy can reduce the complexity of RDF data integration via the use of shared RDF predicates, but is very slow to reason with, while on the other hand the property-oriented strategy is far more efficient for reasoning but requires the use of new properties. This tradeoff is not common knowledge in the ontology engineering community, and consequently, methods and tools that aid engineers in understanding it, would be beneficial. Furthermore, it seems that the usages of the two strategies differ such that the class-oriented strategy is more commonly used in ODP specialisation than in ontologies in general. If this is the case, and if this more common use of the class-oriented strategy is a consequence of the design or use of ODPs, then tooling for ODP specialisation needs to support ontology engineers in applying this to them unfamiliar type of modelling.

Such improved ODP specialisation tooling needs to support at least three different tasks: applying a strategy when specialising an ODP, visualising strategy use in an ODP-based ontology in a user-friendly manner, and refactoring an ODP-based ontology from using one strategy to another.

The first task requires updating the ODP specialisation wizard from the existing XD Tools to be strategy-aware. Upon initiating said wizard, the user needs to be given the choice of by which strategy the ODP is to be specialised. This choice needs to be supported by both information on known effects of strategy use (e.g., guidance texts regarding the trade-offs) and by information on existing strategy use in the ontology project under development and/or the ODP itself. In the case that the user selects the class-oriented or hybrid strategies, the second and third steps of the existing XD tools specialisation wizard (specialising leaf properties, and defining domains and ranges) would need be either replaced or modified, to support the implementation of said strategy.

The second task concerns how to display strategy use in an accessible manner, such that an ontology engineer can quickly ascertain the suitability of an ODP-based ontology for different purposes. For this purpose, the existing Protégé ontology metrics view could be extended with metrics indicating specialisation strategy. In order to simplify the use of ontologies built using a class-oriented strategy, displaying the "emulated" domains and ranges of specialised properties in proximity to those properties' definitions would be helpful.

The third task concerns refactoring an ontology from using one strategy to another, and also harmonising strategy use in the case that different strategies are employed in different parts of an ontology. This appears to be a good use case for the OPPL (Ontology Pre-Processing Language) macro language for ontology transformation discussed in Section 2.

In addition to improved ODP tools, the discovery of these specialisation strategies may also warrant updates to ODP repositories; these repositories would need to provide examples of ODPs specialised per the different strategies. This may necessitate new visual representations for representing universal and existential restrictions in an accessible and user-friendly manner; the work of Stapleton et al.[14] in this direction appears very promising.

5.1 Delimitations and Future Work

This is, to the author's best knowledge, the first work on specialisation strategies for object properties. Consequently, there are many limitations in place, and many opportunities for further work. To begin with, the numbers of ODP specialisations initially studied, benchmarks used for reasoning performance evaluation, and reasoners used for that testing, is limited. Validation of these results on larger datasets and other benchmarks and reasoners would be welcome.

It is likely that the strategies discovered here have more effects and tradeoffs associated with them than just those discussed in this paper. For instance, we note that using the class-oriented strategy, typing is inferred from the typing of related individuals; from this we see that the degree of interconnectedness of individuals in a knowledge base expressed according to the class-oriented

strategy will have an impact on reasoning performance over that knowledge base; the strategy used may give rise to different effects across different datasets.

Two other areas that the author aims to explore further in the near future are the effects of applying the hybrid strategy, and if, and in that case how, these strategies can be applied to datatype properties.

6 Conclusions

In this paper we have studied how Content Ontology Design Patterns (ODPs) are used and specialised in practical Ontology Engineering tasks, with an eye towards improving tooling for such engineering work. The main contributions of this work are a classification of three different strategies for ODP specialisation, and an initial understanding of some consequences of employing these strategies. We have shown that these strategies are being used in modelling of publicly released ontologies, both ODP-based and non ODP-based ones. We have noted the trade-off that an ontology engineer may need to make between reasoning performance efficiency on the one hand, and data integration simplicity on the other. Finally, we have suggested and are in the process of implementing tooling improvements to better support Ontology Engineers in understanding and using these strategies.

References

1. Aranguren, M.E., Antezana, E., Kuiper, M., Stevens, R.: Ontology Design Patterns for Bio-ontologies: A Case Study on the Cell Cycle Ontology. BMC Bioinformatics 9(Suppl 5) (2008)
2. Blomqvist, E., Gangemi, A., Presutti, V.: Experiments on Pattern-based Ontology Design. In: Proceedings of the Fifth International Conference on Knowledge Capture. pp. 41–48. ACM (2009)
3. Blomqvist, E., Sandkuhl, K.: Patterns in Ontology Engineering: Classification of Ontology Patterns. In: Proceedings of the 7th International Conference on Enterprise Information Systems. pp. 413–416 (2005)
4. Daga, E., Blomqvist, E., Gangemi, A., Montiel, E., Nikitina, N., Presutti, V., Villazon-Terrazas, B.: D2.5.2: Pattern Based Ontology Design: Methodology and Software Support. Tech. rep., NeOn Project (2007)
5. Dzbor, M., Suárez-Figueroa, M.C., Blomqvist, E., Lewen, H., Espinoza, M., Gómez-Pérez, A., Palma, R.: D5.6.2 Experimentation and Evaluation of the NeOn Methodology. Tech. rep., NeOn Project (2007)
6. Egaña, M., Rector, A., Stevens, R., Antezana, E.: Applying Ontology Design Patterns in Bio-Ontologies. In: Knowledge Engineering: Practice and Patterns, pp. 7–16. Springer (2008)
7. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: The Semantic Web–ISWC 2005, pp. 262–276. Springer (2005)
8. Horridge, M., Aranguren, M.E., Mortensen, J., Musen, M.A., Noy, N.F.: Ontology Design Pattern Language Expressivity Requirements. In: Proceedings of the 3rd Workshop on Ontology Patterns (2012)

9. Kang, Y.B., Li, Y.F., Krishnaswamy, S.: Predicting reasoning performance using ontology metrics. In: The Semantic Web–ISWC 2012, pp. 198–214. Springer (2012)
10. Keet, C.M.: Detecting and revising flaws in owl object property expressions. In: Knowledge Engineering and Knowledge Management, pp. 252–266. Springer (2012)
11. Presutti, V., Blomqvist, E., Daga, E., Gangemi, A.: Pattern-Based Ontology Design. In: Ontology Engineering in a Networked World, pp. 35–64. Springer (2012)
12. Presutti, V., Daga, E., Gangemi, A., Blomqvist, E.: eXtreme Design with Content Ontology Design Patterns. In: Proceedings of the Workshop on Ontology Patterns (WOP). p. 83 (2009)
13. Presutti, V., Gangemi, A., David, S., Aguado de Cea, G., Suárez-Figueroa, M.C., Montiel-Ponsoda, E., Poveda, M.: D2.5.1: A Library of Ontology Design Patterns: Reusable Solutions for Collaborative Design of Networked Ontologies. Tech. rep., NeOn Project (2007)
14. Stapleton, G., Howse, J., Taylor, K., Delaney, A., Burton, J., Chapman, P.: Towards diagrammatic ontology patterns. In: Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns (2014)
15. Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., Bal, H.: OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In: The Semantic Web: Research and Applications, pp. 213–227. Springer (2010)