

Towards Scalable Ontology Engineering Patterns: Lessons Learned from an Experiment based on W3C's Part-whole Guidelines

Laurent Lefort, Kerry Taylor and David Ratcliffe

CSIRO ICT Centre

GPO Box 664 Canberra ACT 2601, Australia

{laurent.lefort, kerry.taylor, david.ratcliffe}@csiro.au

Abstract

This paper presents an empirical evaluation of description logic reasoners to support the selection of scalable ontology engineering patterns for TBox reasoning. Our main objective is to define the rationale behind the design decisions required for the generation of large ontologies with XSLT-based tools. We discuss here the outcomes of an experiment focusing on aircraft components and parts for which we have implemented the ontology design guidelines for part-whole relationships published by W3C's Semantic Web best practices working group. We have worked with the following reasoners, being the best state-of-the-art currently available: FaCT++, RACER, Pellet and CEL. We found considerable variation in reasoner performance and have attempted to characterise the factors that distinguish the reasoners to enable a best-practice design style to be successfully applied for the generation of very large ontologies.

Keywords: Description logics, reasoner, classification, performance.

1 Introduction

1.1 Motivation

Experimental evaluations of advanced description logic (DL) reasoners have been performed, but there is a lack of systematic evaluation of modern reasoner performance where large-scale ontology reasoning (TBox-reasoning) is emphasised in preference to large-scale instance reasoning. As a general rule, based on our own experience to date as well as the literature, numbers of concepts in the low thousands are likely to be enough to create difficulty for OWL-DL reasoners. One particular challenge is to understand which reasoners perform better for each style of ontology, and this is difficult when the ontologies used for the evaluation are built on a large range of ontology engineering patterns (OEP).

Our objective is to provide a more systematic evaluation to support design decisions that trade off the choice of the more appropriate pattern and the scalability constraints imposed by the available DL reasoners. Now we can use ontologies based on guidelines produced by the W3C's Semantic Web Practice and Deployment¹ group (Schreiber 2006, KnowledgeWeb 2005) to better judge the performance of DL reasoners.

1.2 Related work

From our review of the literature, we have identified three main categories of performance evaluation studies: reasoner, benchmark and production studies.

- **Reasoner studies** are papers published by the authors of DL reasoners and are more likely to focus on specific DL features and the selection of the testing samples is often biased towards the newly available or improved features.
- **Benchmark studies** are papers published by end users trying to understand the difference between DL reasoner products and constructing specific benchmarks to support their analysis. The Lehigh University Benchmark (LUBM) is a popular benchmark designed against a model of ontology expressiveness with domain-realistic queries for ABox reasoning.
- **Production studies** are papers published by end users trying to use reasoners over their own large ontologies, and may or may not include comparative analyses of several DL products. The goal is to build ontologies for a specific purpose and then to use them, so the size, the complexity and the overall quality of the ontology are generally higher than what is evaluated elsewhere. The issue with such ontologies is that reasoners may or may not cope with the full content.

Table 1 is a summary of the existing literature based on these criteria.

Study / Group of studies	Study type
Haarslev et al., 2004, 2005 (RACER)	Reasoner + Benchmark
Sirin et al., 2005, 2006 (Pellet)	Reasoner + Benchmark
Tsarkov & Horrocks 2005a, 2005b (FaCT++)	Reasoner
Motik et al., 2002, Motik 2006 (Kaon 2)	Reasoner + Benchmark
Baader et al., 2005, 2006 (CEL)	Reasoner
Guo et al., 2003, 2004 (LUBM)	Benchmark
Gardiner et al., 2006	Benchmark
Dameron et al., 2005	Production

Table 1: Published studies

Our approach combines the merits of the benchmark and the production types of studies. We benefit here from the reuse of large domain-specific inputs. With the help of XO, our ontology generation tool (Lefort & Taylor 2005), we have generated an aircraft ontology describing components and parts from the Service Difficulty Reports (SDR) published by the Federal Aviation Administration (FAA 2005). The resulting ontology has 4000 classes. To enrich the performance evaluation, we have split the ontology into modules of various sizes accorded to the functional hierarchy of the industry-specific ATA/JASC coding system (FAA 2002).

1.3 Ontology normalisation and patterns

Rector (2003, 2005) defines ontology normalization as the application of a limited number of criteria to eventually “let the reasoner do multiple classification”. This is done through the construction of complex primitives as “a conjunction of one class and a boolean combination of zero or more restrictions”.

This experience in the creation of TBox-based ontologies has been disseminated through the W3C best practice group as shown in **Table 2**.

Normalisation (Rector)	Ontology Engineering Pattern documents (W3C)
Upper skeleton axioms	N-ary Relations
Complex primitives w/ property restrictions	Simple part-whole relations
Complex primitives w/ cardinality restrictions	Qualified cardinality restrictions
Refining primitives	Specified Values

Table 2: Relationships between W3C’s OEP and Rector’s Ontology Normalization

To enable a more scalable ontology engineering practice, we need to study how the reasoners can cope with the recommended ontology engineering patterns defined by the W3C OEP task force. The pattern selected for this study is the one described in the “Simple part-whole relations in OWL ontologies” document edited by Rector & Welty (2005).

1.4 Patterns for part-whole relations

Rector & Welty (2005) first provide useful advice on how to handle the *isPartOf* relation with two properties: one transitive property for the general case completed with a sub-property to handle “direct” relations. It also recommends focus on the *isPartOf* relation rather than on the inverse *hasPart* one, and to avoid using the two together because of scalability issues with cross-referenced existential restrictions.

Rector & Welty’s 2005 pattern “N.4” is for the propagation of properties along the part-whole hierarchy. It aims at an ontology combining a part inventory with a fault finding system to describe the devices made in a factory with a focus on the part-whole relations between parts and sub-parts. Its goal is to enable the inference that a fault in a part is a fault in the whole at different levels of the part-whole hierarchy. In this paper, we apply this pattern to manage the *hasFunction* relation with respect to the functional hierarchy provided by the ATA coding system.

1.5 Outline of the paper

This paper is structured in four parts. Section 2 outlines the goal of the experiment and section 3 describes the method used to evaluate the reasoner performance. The results and our supportive analysis are included in Section 4. Section 5 discusses what is required to enable a more scalable ontology engineering practice, both in terms of guidelines and tools. Section 6 concludes and identifies the opportunities to extend this work.

2 Experimental goals

The objective of this experiment is to understand the impact of the ontology design patterns on reasoner performance and the differences between reasoners and the effectiveness of their optimization tactics. We have created several variants of the part-whole ontologies to study specific DL reasoners and to strengthen the evaluation outcomes. Because the CEL reasoner (Baader et al. 2006) cannot represent concept disjunction (or *unionOf* in OWL), we have generated an alternative ontology without this part of the pattern. We have also evaluated the reasoner performance on ontologies using simultaneously roles and their inverses, e.g. *isPartOf* and *hasPart*, because this practice is flagged as a cause of problems in the Rector-Welty paper.

2.1 The part-whole ontology engineering pattern

The example used by Rector and Welty describes the OWL pattern for an ontology about:

- A part inventory for the devices made in a factory with the relation between each part and its sub-parts.
- A fault finding system for a device in which we want to progressively narrow down the functional region of the fault.

This pattern is based on the DL expression which uses the abstract syntax for OWL defined by W3C (Patel-Schneider et al. 2004).

```
Class(FaultInCar complete intersectionOf
(Fault
restriction (isLocusOf someValuesFrom
(unionOf (Car
restriction (isPartOf someValuesFrom (Car)))))))
```

(From Rector & Welty (2005))

A graphical view is given in **Figure 1**. CarPart is used to represent the anonymous class defined by the existential restriction (isPartOf someValuesFrom (Car)).

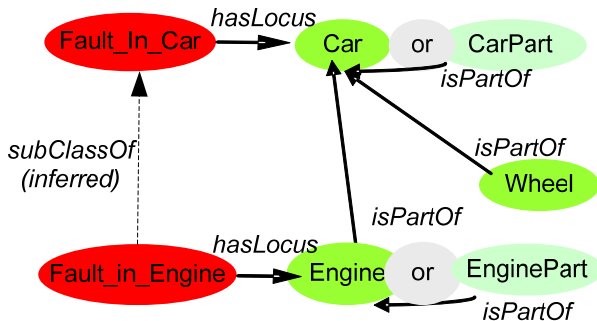


Figure 1: Rector-Welty pattern N.4 adapted from Rector and Welty (2005²)

This pattern can be repeated and thereby propagated at each level of the hierarchy, first for the Engine class as illustrated in **Figure 1** and then repeatedly. Let's suppose this pattern is applied again for sub-parts of Engine such as Crankcase. Once this is done, the reasoner will be able to infer subsumption relations between faults by going back up the part-whole hierarchy. A fault at the lowest level (a fault in a bolt in the crankcase) will be inferred as a fault in the crankcase and then as a fault in the engine and eventually as a fault in the car. The *isPartOf* relation

must be declared as transitive for the pattern to work repeatedly up the part-whole hierarchy.

2.2 Application to aircraft ontologies

The Rector-Welty recommendations are directly transposable to the aircraft domain we are interested in. Figure 2 shows an example of relation between part and whole derived from the SDR inputs.

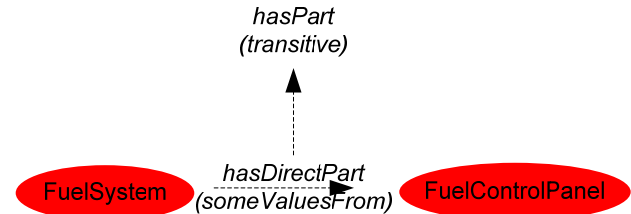


Figure 2: Basic Part-Whole relation

It can be applied to manage the relation between components/parts and function and exploit the functional hierarchy provided by the ATA coding system. In this case study, our approach is to apply the same pattern to propagate the functional hierarchy down the part-whole hierarchy. This is illustrated in Figure 3.

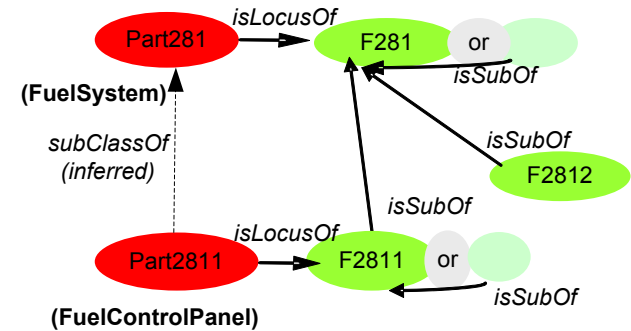


Figure 3: Functional hierarchy of physical parts

Here, we create supplementary classes for each function (e.g. Part281, Part2810) and then use the relation between the artefacts (e.g. Fuel System) and the functions to infer where each named component (e.g. Fuel Control panel) stands in the ATA-based hierarchy of classes.

2.3 Compatibility with DL languages

The Rector-Welty part-whole pattern N.4 uses a subset of the OWL Lite description logic language known as *SHF*, which is supported by all the reasoners belonging to the OWL-DL family such as FaCT++, RACER and Pellet. According to Zolin (2006), the theoretical complexity for SHF, is ExpTime-complete.

Our analysis of the literature suggests that reasoner performance problems occur with ontologies expressed in *ALC* using a large number of existential restrictions. In practice, these reasoners will differ mostly by the optimisations implemented in their tableau-based algorithms.

² (Figure 1., available from draft version 0.2 only <http://www.cs.man.ac.uk/~rektor/swbp/simple-part-whole/simple-part-whole-relations-v0-2.html>)

Theoretical complexity is not the same with *acyclic* and *cyclic* TBoxes³. A cyclic TBox is one that contains concept inclusion axioms that reference the same (or equivalent) classes on both side of the subsumption relation (known as a terminological cycle). In other words, a terminological cycle is one that defines a concept in terms of itself in some way. Generally speaking, complexity results for cyclic Tboxes are worse than for the same language in the presence of cycles (Nebel 1991, Baader 2003).

$\mathcal{EL}+$ is a description logic that does not belong to the OWL family. The reasoners implementing $\mathcal{EL}+$, such as CEL (Baader et al., 2005), can offer a different trade-off between tractability and expressivity. Inference procedures, such as checking for ontology consistency, concept satisfiability, subsumption and instance checking have known deterministic polynomial time complexity, even for cyclic TBoxes. On the other hand, $\mathcal{EL}+$ users lose several OWL features, most notably value restrictions, i.e. *allValuesFrom*, which is necessary for OWL domain and range assertions. Nor can they use concept disjunction or *unionOf* in OWL.

This is why we cannot use CEL to reason over ontologies applying the Rector-Welty pattern, but we study CEL for a substantially different perspective on reasoning.

3 Evaluation method

3.1 Generation of the test ontologies

XO (Lefort & Taylor 2005), is a tool to build transformations/conversions from XML to OWL. XO allows us to get the variability in ontology size and complexity required for this study. Because we import real world data from existing sources, this study has some elements which are specific to the aircraft maintenance domain. We process the Service Difficulty Reports to get information about parts, their physical and functional relations to other parts and their functional affiliation to the categories defined by ATA/JASC coding system (FAA 2002).

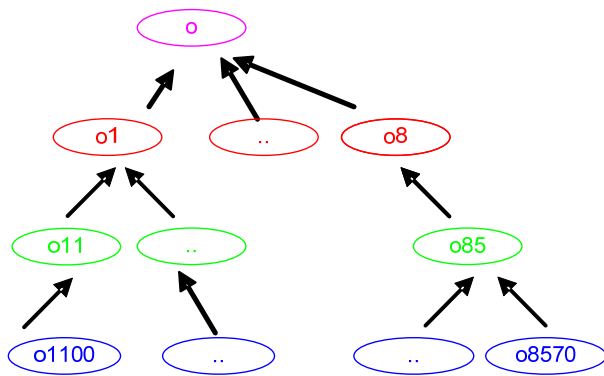


Figure 4: Multi-levels ontology test samples

Figure 4 illustrates how we work at multiple levels of this hierarchy to generate ontologies of variable size.

- The *aircraft ontology* [o] covers all ATA codes;
- each *one-digit ontology* [o1..o8] comprises between 0 and 1000 four-digit ATA codes;
- each *two-digit ontology* [o11..o85]) comprises between 0 and 100 four-digit ATA codes;
- And each *four-digit ontology* [o1100..o8570] comprises between 0 and 10 ATA codes.

This working environment is flexible enough to allow us to study some variants of the Rector-Welty patterns: for example, to evaluate CEL, we have generated another whole set of ontologies for which we have removed the constructs from the Rector-Welty pattern which are not allowed in $\mathcal{EL}+$.

3.2 Evaluation of reasoner performance

Classification can be defined as the computation of the subsumption hierarchy for classes and properties. In this experiment, we consider classification to be the crucial TBox reasoning task. Gardiner et al. (2006) describe a method to trigger the classification and to measure its time for any reasoner with a DIG interface. In this method, the classification time is obtained from the response time to a query on the satisfiability of the top concept, requested once the ontology has been loaded. We have used a comparable method for FaCT++, RACER and Pellet. For CEL, the classification time is measured internally and is provided as an output. We have also used a more direct measurement method for Pellet corresponding to the published benchmark results.

3.3 Extraction of supporting metrics

Additional size and complexity indicators are required to help us to study the impact of the Ontology Engineering Patterns (or OEP) on the reasoner performance. For each sample, we have recorded the number of classes, properties, and triples and the number of cycles flagged by RACER.

We have also defined two specific indicators for this study to help us to analyse how we apply the Rector-Welty pattern described above. The first one is for the top part of the Rector-Welty pattern (1) and the second one is for the bottom part (2).

We define the *number of times we use the Rector-Welty pattern* as the number of times we use constructs like:

```
Class(Entity281 complete intersectionOf
(Entity
restriction (hasFunction someValuesFrom
(unionOf (Function 281
restriction (isSubOf someValuesFrom (Function281)))))))))
```

(1) *Class counted as one use of the Rector-Welty pattern*

³ The DL Complexity navigator web page from Zolin (2006) can give the theoretical complexity for acyclic and for “general” TBoxes for the relevant range of DLs.

We define the *number of classes involved in the Rector-Welty pattern* as the number of times we use constructs like:

```
Class(Entity2810 complete
intersectionOf (Entity
restriction hasFunction someValuesFrom Function2810))
```

(2) *Class involved in the Rector-Welty pattern*

We define the *number of isLocusOf existential restrictions* as the number of times we use constructs like the one present in (3). A given class may have one or more of such existential restrictions:

```
Class(FuelControlPanel partial intersectionOf
(Entity
restriction isLocusOf someValuesFrom Function2810))
```

(3) *Class with one isLocusOf existential restriction*

For the Rector-Welty based ontologies, we also use the number of *isPartOf* existential restrictions to measure the number of constructs of the form presented in **Figure 2**.

To monitor the presence of “cyclic axioms” (Haarslev et al. 2005), we also extract the number of cycles from RACER’s warning messages.

Table 3 provides a summary of the indicators which are used for this study:

Indicators
Number of classes
Number of properties
Number of triples
Number of cycles from RACER (option -v)
Number of times we use the Rector-Welty pattern
Number of classes involved in the Rector-Welty pattern
Number of <i>isLocusOf</i> existential restrictions (relations between entities and functions)
Number of <i>hasDirectPart</i> existential restrictions (relations between components and parts)
Total number of existential restrictions

Table 3: Complexity indicators

3.4 Experimental setup

Our XO tool is available through an ANT-based working environment which can be used to create the ontologies, to evaluate the performance of reasoners and to provide the supporting complexity indicators of **Table 3**. **Figure 5** gives an overview of this working environment.

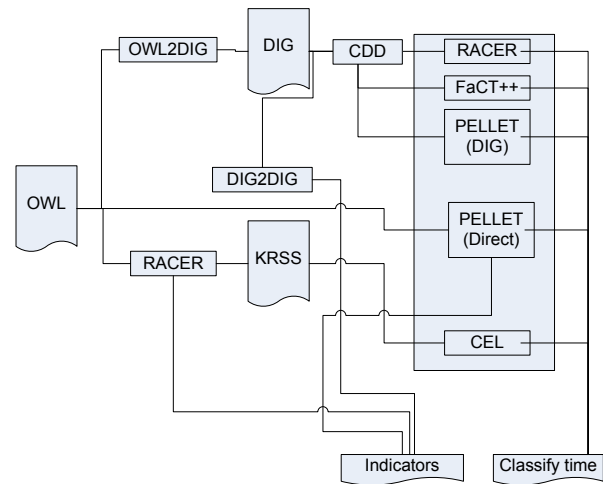


Figure 5: Experiment setup

In the figure, OWL2DIG is the tool developed by Zhang and Zhou available from SemWebCentral⁴. CDD, the tool used to interact with the DIG reasoners is derived from a subset of the Context-Driven Development Toolkit⁵ developed by Wagelaar to provide a service equivalent to the approach described by Gardiner et al (2006). The authors’ own XSL transformation Dig2Dig is used to tidy the DIG file and to generate some of the size and complexity indicators specifically defined for this study. RACER is also used directly to convert ontologies from OWL to KRSS, the format required by the version of CEL used in this experiment.

We can use this setup to compare the results for Pellet in two execution modes, when it is used directly and when it is used as a DIG server. The results presented here correspond only to the direct execution mode.

Table 4 lists the tool versions and the more critical configuration parameters we have used.

Tool	Version	Parameters
FaCT++	1.1.3	N/A
RACER	1.7.24	Stack size 45000000
Pellet	1.3	Max heap size (jvm) 800m
CEL	0.8	N/A

Table 4: Tools versions and parameters

Our classification time results are measured in CPU seconds on a PC workstation with a 3 GHz CPU and 2 Gb

⁴ SemWebCentral: <http://projects.semwebcentral.org/>

⁵ available from <http://ssel.vub.ac.be/viewcvs/>

of RAM. After 300 CPU seconds, we terminate the classification task and consider it failed.

We also used Protégé⁶ and Swoop⁷ to browse the ontologies and check the classification results.

4 Results

We provide here three separate analyses. The first one will show that, on ontologies based on the Rector-Welty pattern, FaCT++ and RACER behave differently to Pellet with respect to the number of classes, the number of existential restrictions and the number of cycles. The second one illustrates the relatively superior tractability of CEL and its ability to scale for the full aircraft configuration. CEL is much faster than the other reasoners, but can only be used when it is possible to re-factor the ontologies to be $\mathcal{EL}+$ compatible. The third analysis focuses on the changes in reasoner performance.

4.1 Performance on Rector-Welty ontologies

A detailed comparative reasoner performance analysis over the full set of ontologies derived from the Service Difficulty Reports is not presented here in detail. Instead, we provide an illustrative sample of the results we have obtained, as shown below in **Figure 6** which illustrates this analysis for the top 10 four-digit ontologies. To get the top 10 four-digit ontologies, we have ordered the 253 four-digit ontologies (see 3.1 for the definition of one, two and four-digit ontologies) according to the three following indicators in decreasing order of significance: the number of classes and the number of existential restrictions for *isLocusOf* and *hasDirectPart*.

Figure 6 uses a logarithmic scale to facilitate the comparison between the classification time (in seconds) and the numbers used to characterise the complexity of the test samples described in **Table 3**.

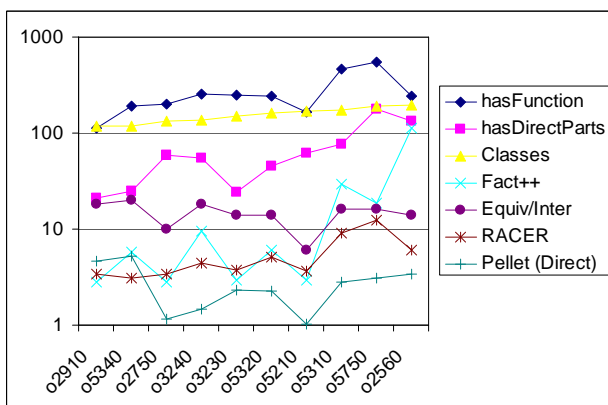


Figure 6: Results for the top 10 four-digit Rector-Welty ontologies (117 to 196 classes)

Our empirical finding is that to avoid reasoner timeouts, it is recommended to limit the generated ontology to use the Rector-Welty pattern less than 6 times, with less than 40

sub-classes involved in it, and to keep the overall size down to less than 150 classes and less than 300 to 400 existential restrictions tied to the same property. We believe that further cleaning and tuning of the generation process is possible to push these empirical limits upward so that we can work over larger subset of the aircraft domain at a time.

We believe that the performance of the three evaluated reasoners is tied to the number of classes involved in the pattern (Classes in **Figure 6**). We believe that the performance of Pellet varies more significantly with respect to this indicator and to the number of times the pattern is exploited; this value is constant for the test sample, so it is not shown in **Figure 6**. RACER and FaCT++ performance is also dependent on this indicator because all these classes are also flagged as cycles, which degrade the performance of these reasoners. In our test samples, FaCT++ and RACER are also penalised by the presence of large numbers of existential restrictions tied to small numbers of properties (*hasFunction* and *hasDirectParts* in **Figure 6**).

We have also noticed that FaCT++ performance is significantly degraded for ontologies containing cycles corresponding to modelling mistakes inherited from the original source: o2560 (on the far right in **Figure 6**) presents such a case with a classifying time over 100s.

4.2 Performance on re-factored ontologies

The re-factored ontologies correspond to a different generation method which suppresses the representation pattern N.4 from Rector & Welty and introduces a greater number of properties to manage relations between components and parts.

Rector & Welty (2005) note that “the inability of existing classifiers to cope with ontologies mixing *isPartOf* and *hasPart* is a significant limitation”. The re-factored ontologies are designed to check if this statement is valid for CEL or not: for each type of relations defined in the ontology, we have added the statements required to represent the inverse relation, but we have not used the *owl:inverseOf* construct for this case because CEL cannot handle it. With this approach, the measured number of classes involved in cyclic axioms signalled by RACER is almost equal to the total number of classes.

In this configuration, CEL is roughly 10 times better than Pellet, and Pellet is roughly 10 times better than RACER. One of the main finding of this experiment is that Pellet is less sensitive to the presence of cyclic TBox inclusion axioms than RACER and FaCT++. This also confirms our previous analysis on the sensitivity of RACER and FaCT++ to cycles. The full results are shown in **Figure 7** for the full set of 51 two-digit ontologies.

⁶ Protégé <http://protege.stanford.edu>

⁷ Swoop <http://www.mindswap.org/2004/SWOOP/>

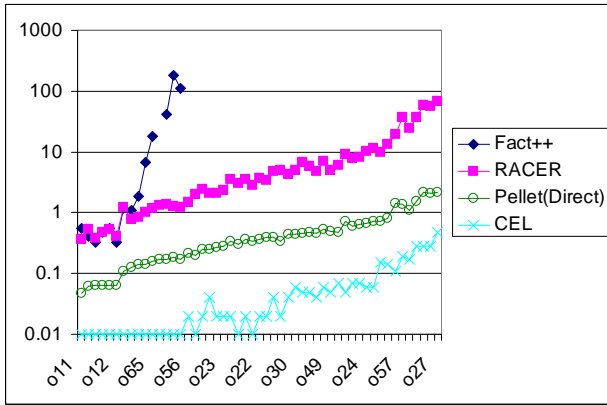


Figure 7: Results for the 51 two-digit re-factored ontologies (10 to 410 classes)

4.3 Performance on ontologies without cycles

To complete this analysis, we have modified the ontology generation instructions to create another set of *mono-directional* re-factored ontologies, simply by removing the statements corresponding to one of the inverse relation pairs (*isPartOf* and *hasPart*, *isFunctionOf* and *hasFunction*, etc.). This allows us to investigate the sensitivity of CEL to cyclic axioms and to check how the performance of FaCT++ and Pellet varies for ontologies without cycles.

For CEL, we can use the totality of the ontology. **Table 5** provides the classification time in seconds, the number of classes and the number of cycles for the mono-directional and bi-directional re-factored ontologies comprising all ATA codes. The remaining cycle for the mono-directional re-factored ontologies (60 classes in total signalled by RACER) are inherited from modelling errors in the source data which are not suppressed by the generation process.

Ontology	CEL	Number of classes	Number of cycles
o w/o/ cycles	40.640	3957	60
o w/ cycles	50.530	3957	3687

Table 5: CEL results for the mono-directional and bi-directional re-factored ontologies

The classification time for the ontology “with cycles” (i.e. the ones including both the *is[...]*Of and the *has[...]* properties) is bigger than the result for the one “without cycles”. This increase is easily explained by the doubling in the number of existential restrictions in the second ontology and suggests that cycles do not pose a problem for performance.

For FaCT++, we focus on the four one-digit ontologies (o1, o4, o6, and o8) for which no cycles are signalled by RACER. **Figure 8** shows the classification times for the Rector-Welty ontologies (FaCT++(1) and Pellet(1)) and for the mono-directional re-factored ontologies (FaCT++(4) and Pellet(4)).

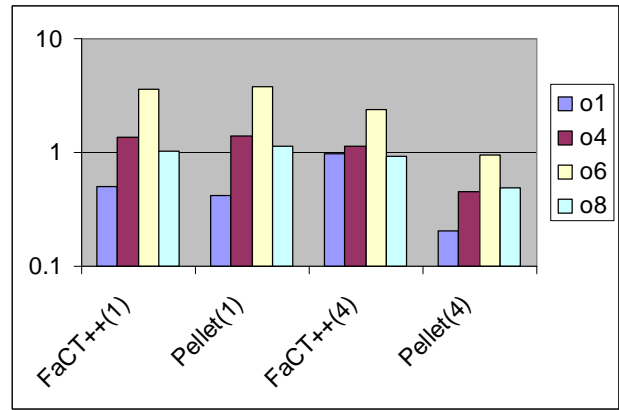


Figure 8: Results for 4 one-digit R-W ontologies and mono-dir. re-factored ontologies (43 to 327 classes)

Figure 8 shows that the performance figures for FaCT++ and Pellet are a lot closer to what has been observed previously. For the Rector-Welty ontologies, the minor difference between FaCT++(1) and Pellet(1) is much smaller than the gap visible in **Figure 6**. The slightly larger difference between FaCT++(4) and Pellet(4) contrasts with the large gap visible in **Figure 7**.

4.4 Other lessons learned

We summarise here the lessons learned during this experiment with respect to how to get the most out of the DL reasoners.

Avoid duplicate superclass declarations.

With our method, it is not always easy to prevent the occurrence of duplicates, especially those linked to existential restrictions which we do not handle well. The capability to remove all types of duplicates, now available in Protégé 3.2, should be added to our generation environment. To give an idea of the size of the problem, more than 7 hours have been necessary for Protégé to remove the duplicate definitions out of the largest ontology of this experiment, with a resulting ontology one tenth of the size. Further analysis is required to understand how the presence of duplicates may influence reasoner performance in their various execution modes. This gain in simplicity is now available for the users of Protégé thanks to the *duplicate superclasses post processor* developed by H. Knublauch⁸.

Beware of oversized content and of DIG server behaviour.

Both execution modes for Pellet have been used for evaluation to compare the direct execution of the reasoning core to the one triggered through the DIG interface. Our results show a significant gap between the two modes and memory errors that are more likely to occur through the DIG interface. FaCT++ also has trouble with oversized input because it does not manage interrupted sessions adequately. RACER is probably the

⁸ DuplicateSuperclassesPostProcessor.java is accessible through the Protégé source code repository hosted at <http://smi-protege.stanford.edu/> Accessed 25 Aug. 2006.

most robust and stable DIG implementation we have used. The DIG capability for CEL has not yet been evaluated.

5 Discussion

5.1 Scalable ontology engineering guidelines

It is challenging to develop a best-practice design style which can be successfully applied for the generation of very large ontologies. To reach this objective, we need more precise guidelines and modularisation approaches which acknowledge the practical limitations of reasoners.

There is a lack of reference to *scalable modelling style* in the currently available guidelines from W3C listed in **Table 2**. **Table 6** proposes four categories to better scope the presently available advice with respect to the orientation of the users towards scalable TBox reasoning or ABox reasoning and to their reasoner preferences. Further refinement of these categories is likely to be needed once the work on new tractable fragments to extend OWL from the OWL community (2006) is finalised.

Scalable modelling style	TBox / ABox	Reasoner
SNOMED-like ontologies	TBox	CEL (FaCT++)
GALEN-like ontologies	TBox	FaCT++, (RACER), (Pellet)
DOLCE and Wine-like ontologies	TBox / ABox	RACER, Pellet, (KAON2)
LUBM-like ontologies	ABox	(RACER), (Pellet), KAON2

Table 6: Benchmark categories and reasoners

End users are likely to be annoyed by classification times greater than 5 minutes for non repetitive tasks and 30 seconds or less for repetitive tasks. This is why the present OEP guidelines also need to be augmented with more empirical advice on the number of times a specific pattern can be used and the number of classes and existential restrictions which can be planned so that the generated ontologies do not lead to excessive reasoning delays. In the case of the Rector-Welty pattern, an important requirement for potential users is to know how many times it can be applied recursively. Further work is required because in the results presented here, the Rector-Welty pattern is acted upon over the ATA/JASC code hierarchy and not over the part-whole hierarchy inherited from the input.

5.2 The Rector-Welty guideline and CEL

We have discussed above that CEL cannot handle the *unionOf* construct present in the Rector-Welty pattern. For this case, an alternative approach should be defined.

CEL does support a particular non-OWL feature called role inclusion, a form of role composition that supports declarations of transitive roles (also in OWL) and right identity (not in OWL). Right identity is likely to be important for applications of aircraft configuration ontologies: indeed it obviates the Rector-Welty pattern for explicit transmission of faults up the *isPartOf* hierarchy. We illustrate by example, following the syntax and style of the medical example of Horrocks and Sattler (2003).

We assert the right-identity axiom,

$$hasLocus \circ isPartOf \leq hasLocus$$

to mean that something that is located in a part P is also located in the places that P is a part of. This means that an expression declaring a fault of the crankcase of the aircraft engine:

$$Fault \cap \exists hasLocus \bullet (Crankcase \cap \exists isPartOf \bullet Engine)$$

is inferred to be a fault of the engine itself. That is, the following can be inferred to hold, from the previous two declarations:

$$Fault \cap \exists hasLocus \bullet Engine$$

Furthermore, assuming the *isPartOf* role is declared to be transitive and the appropriate part and component taxonomy is defined, a fault located in the aircraft as a whole is also inferred. Presently, we can apply the transitive *isPartOf* structure in the ontologies we have generated (see section 6), but not the right-identity feature because it is not (yet) handled by tools based on the present versions of OWL and on DIG.

5.3 Scalability and modularisation

The development of modularisation approaches to managing the scalability of ontologies is in a very early stage; and is typically applied from the point of view of the need for independent development of modules by knowledge engineers (Rector & Pan 2005).

There is certainly an opportunity to consider what can be done in a semi-automated environment such as that which is offered by XO, where a combination of techniques related to modularisation and restrictions to the language may be achievable.

5.4 Handling cyclic axioms through DIG

The working environment we have defined allows us to retrieve the cyclic axiom warnings reported by RACER. We have also manually monitored a similar type of warning as reported by FaCT++. We would be interested to get this type of information more seamlessly. A possible approach would be to extend the DIG interface with specific services providing the number of cycles and the identity of the classes involved with as much detail as

available. This would help users of tools such as Protégé to better understand how the reasoner performance is affected by the choice of the ontology engineering pattern and also to help them to fix possible modelling errors.

6 Conclusion

Reasoner performance over large ontologies with large TBox components is highly variable and dependent on the choice of the ontology engineering patterns. Starting from the published advice on part-whole ontologies has been critical for this experiment, guiding us towards a much needed simplification of the generated ontologies.

The main contribution of this paper is our experimental analysis on the representation pattern recommended for the propagation of properties along the part-whole relation. Our experience shows that this pattern can be used on sub-modules of a size of less than around 150 to 200 classes.

We have also confirmed that avoiding or removing inadvertent cyclic axioms is critical for the performance of some reasoners, especially for FaCT++. This reinforces the W3C advice to avoid combining *isPartOf* relations and their inverses for part-whole ontologies.

CEL is less sensitive to terminological cycles, and we are confident it can be used to handle an ontology corresponding to the whole aircraft configuration with 4000 classes or more such as the one we have created for this experiment.

We plan to continue this work in two directions. Our first priority is to expand the results of this study for the other ontology engineering patterns authored by the OEP task force listed in **Table 2**. Our second priority is to investigate the scalability and modularity challenges presented by the large medical ontology, SNOMED. Further work is required to validate and evaluate the adaptation proposed to the present W3C part-whole recommendation to match CEL capabilities.

Finally, this study illustrates the effectiveness of modularisation approaches combined with the selection of the appropriate modelling style to support the generation of large ontologies out of existing resources. More support from the existing tools is required to better align the published guidelines to the specificities of each reasoner and to help the user to eliminate the present causes of problems such as cycles or duplicate superclass definitions which are more likely to occur in this context.

Acknowledgements

The authors gratefully acknowledge the support of Boeing in this work, particularly of Robb Graham, Steve Uczekaj and Craig Battles.

Many thanks also to John Colton of CSIRO.

7 References

Baader, F. (2003): *Terminological Cycles in a Description Logic with Existential Restrictions*. LTCS-Report 02-02, Institute for Theoretical Computer

Science, Dresden University of Technology, Germany, 2002.

Baader, F., Brandt, S., and C. Lutz, C. (2005): Pushing the EL Envelope, In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, Edinburgh, UK, Morgan-Kaufmann .

Baader, F., Lutz, C. and Suntisrivaraporn, B. (2006): CEL-A Polynomial-time Reasoner for Life Science Ontologies (System Description). In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, Seattle, WA, USA, Lecture Notes in Artificial Intelligence 4130. Springer, pp. 287–291.

Dameron, O., Rubin, D., and Musen, M. (2005): Challenges in converting frame-based ontology into OWL: the Foundational Model of Anatomy case-study. In *Proceedings of the American Medical Informatics Association Annual Symposium (AMIA05)*, Washington DC, pp. 181-185.

FAA (2002): *Joint Aircraft System/Component Code* Federal Aviation Administration, Flight Data Services http://av-info.faa.gov/isdr/documents/JASC_Code.pdf Accessed 25 Aug 2006.

FAA (2005): *FAA Service Difficulty Reporting Service* <http://av-info.faa.gov/isdr/> Accessed 25 Aug. 2006.

Gardiner, T., Horrocks, I., and Tsarkov, D. (2006): Automated benchmarking of description logic reasoners. In *Proceedings of the 2006 Description Logic Workshop (DL 2006)*. Windermere, Lake District, UK, Parsia, B., Sattler, U. and Toman, D. Eds. CEUR Workshop Proceedings 189, CEUR-WS.org

Guo, Y.; Heflin, J; and Pan, Z. (2003): Benchmarking DAML+OIL Repositories In *The Semantic Web - Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, Sanibel Island, FL, USA, Springer, pp, 613-627.

Guo, Y., Pan, Z., and J. Heflin, J., (2004): An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *Proceedings of Third International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, LNCS 3298, Springer, 2004, pp. 274-288.

Haarslev, V., Möller, R. and Wessel, M. (2004): Querying the Semantic Web with Racer + nRQL In *Proceedings of the Workshop on Description Logics 2004 (ADL 2004)*, Ulm, Germany, 2004

Haarslev, V., Möller, R. and Wessel, M. (2005): Description Logic Inference Technology: Lessons Learned in the Trenches in *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, Edinburgh, Scotland, UK, July 26-28, 2005,

Horrocks, I. and Sattler, U. (2003): Decidability of SHIQ with complex role inclusion axioms. *Artificial Intelligence*, Volume 160, Issues 1-2:79-104,

KnowledgeWeb (2005): *Success Stories and Best Practices, KnowledgeWeb Deliverable 1.4.2* Lancieri,

- L., Maynard, D. and Gandon, F. Eds. KWEB/2004/D1.4.2/v3 KnowledgeWeb consortium
- Lefort L. and Taylor, K. (2005): Large scale colour ontology generation with XO In *Proceedings of the Australasian Ontology Workshop (AOW 2005)*, Conferences in Research and Practice in Information Technology (CRPIT), Vol. 58. Meyer T. and, Orgun, M. Eds. Sydney, Australia.
- Motik, B., Volz, R. and Maedche, A. (2002): Optimizing query answering in description logics using disjunctive deductive databases. In *Proceedings of the 10th International Workshop on Knowledge Representation Meets Databases (KRDB-2003)*, Hamburg, Germany, pp 39–50.
- Motik, B. (2006): *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, University of Karlsruhe, Germany.
- Nebel, B. (1991): Terminological cycles: Semantics and computational properties. In *Principles of Semantic Networks*, Sowa, J., F. Ed., Morgan Kaufmann, Los Altos, pp 331-361.
- OWL community (2006): *Tractable Fragments of the OWL 1.1 Extension to the W3C OWL Web Ontology Language*, Editor's Draft of 14 June 2006, Cuenca Grau, B. Ed., University of Manchester <http://owl1-1.cs.manchester.ac.uk/Tractable.html> Accessed 25 Aug. 2006.
- Patel-Schneider, P., Hayes, P. and Horrocks, I. (2004): OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation 10 February 2004 <http://www.w3.org/TR/owl-semantics/> Accessed 25 Aug. 2006
- Rector, A. (2003): Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *Proceedings of the 2nd international Conference on Knowledge Capture (K-CAP '03)* Sanibel Island, FL, USA, ACM Press, New York, NY, pp 121-128.
- Rector, A. (2005): *Ontology Normalisation, Pre- and Post- Coordination* Advanced Ontology Tutorial <http://www.cs.man.ac.uk/~rektor/tutorials/feb/Presentations/Normalisation.ppt> Accessed 25 Aug. 2006.
- Rector, A. and Pan, J. (2005): Chapter 12 Engineering Robust Modules In *Report on Modularization of Ontologies*, Spaccapietro S. Ed. KnowledgeWeb Deliverable D2.1.3.1, KWEB/2004/D2.1.3.1/v1.1 KnowledgeWeb consortium
- Rector, A. and Welty, C. (2005): *Simple part-whole relations in OWL Ontologies*, W3C Editor's Draft 11 Aug 2005, Version 1.5, <http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/index.html> Accessed 25 Aug. 2006.
- Schreiber G. (2006): *Ontology patterns (Lecture notes), Knowledge modeling course*, Dutch Research School for Information and Knowledge Systems, Vught, Netherlands <http://hcs.science.uva.nl/SIKS/> Accessed 25 Aug. 2006.
- Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A. and Katz, Y. (2005): Pellet: A Practical OWL-DL reasoner. *UMIACS Technical Report 2005-68*. Submitted for Publication to Journal of Web Semantics
- Sirin, E., Cuenca Grau, B. and Parsia, B. (2006): From wine to water: Optimizing description logic reasoning for nominals. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Tenth International Conference (KR2006)*, Lake district, UK. Doherty, P. Mylopoulos, J. and Welty, C. Eds, AAAI Press, pp 90-99.
- Tsarkov, D. and Horrocks, I. (2005a): Ordering heuristics for description logic reasoning. In *Proceedings of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pp. 609-614.
- Tsarkov, D. and Horrocks, I. (2005b): Optimised classification for taxonomic knowledge bases. In *Proceedings of the 2005 Description Logic Workshop (DL 2005)*, Edinburgh, Scotland, UK, Ian Horrocks, I. Sattler, U. and Wolter, F. Eds, CEUR Workshop Proceedings 147, CEUR-WS.org
- Zolin, E. (2006): Complexity of reasoning in description logics <http://www.cs.man.ac.uk/~ezolin/logic/complexity.htm> Accessed 25 Aug 2006