

# **An Empirical Comparison of Scalable Part-Whole Ontology Engineering Patterns**

Laurent Lefort, Kerry Taylor and David Ratcliffe

CSIRO ICT Centre, GPO Box 664 Canberra ACT 2601, Australia

[{firstname.lastname}@csiro.au](mailto:{firstname.lastname}@csiro.au)

Correspondence should be sent to Laurent Lefort: [laurent.lefort@csiro.au](mailto:laurent.lefort@csiro.au)

## **Acknowledgements**

This research was supported jointly by the Boeing Company and CSIRO.

**Abstract:**

To enhance aerospace applications such as supply chain management or maintenance tracking and reliability assessment, aircraft manufacturers need to enrich their electronic documentation systems with better conceptualisations of the aerospace domain. Because of the number and diversity of products and parts in an aircraft, ontologies to model this domain can potentially be very large. Therefore, it is critical to have scalable ontology generation approaches, along with an understanding of how the design of these ontologies has an impact of the performance of description logic reasoners that can operate over them. In this paper, we investigate how to achieve this goal through the application of best practice ontology engineering patterns. In particular, we examine two types of ontology engineering patterns used to instantiate large-scale ontologies based on part-whole relations. The first approach is a direct implementation of the part-whole guidelines published by W3C. The second approach uses right-identity axioms supported by the EL+ description logic. The results of our empirical evaluation show the benefits of the latter approach, whereby the CEL reasoner is able to perform the task of classification over large ontologies significantly faster than the description logic reasoners FaCT++, RACER and Pellet that operate over comparable ontologies designed using the W3C pattern.

**Keywords:** Description logics, ontology engineering, part-whole, pattern, reasoner, classification, performance.

## 1 Introduction

Like components on an assembly line, Internet-ready resources can now be assembled into bigger, specialized information systems. Web services and semantic web technologies provide a powerful basis for application designers to find, relate and compose elementary databases or services into a purpose-built solution. To support this semantic integration process, large-scale ontologies are required to manage the complexity of resource-specific definitions on behalf of the end users of such systems of systems. The Web Ontology Language (OWL) (Dean and Schreiber, 2004) standardised by the World Wide Web Consortium (W3C) is the most popular ontology language for builders of such applications.

To enhance aerospace applications such as supply chain management or maintenance tracking and reliability assessment, it is valuable to upgrade the taxonomies already exploited by aircraft manufacturers in their electronic documentation systems into richer ontologies. Better conceptualisations of the aerospace domain are needed to develop future systems for flight operations (Reiss *et al.* 2006) and for Radio Frequency Identification (RFID) applications to aircraft maintenance (Nicolai *et al.* 2005). Because of the large size of existing taxonomies due to the diversity of products and parts in an aircraft, more scalable ontology generation approaches are needed for the creation of such ontologies. The advantages of a semi-automated approach are direct gains in terms of traceability, repeatability and scalability. The traceability to the original source of knowledge can be maintained and the whole generation process can be completed or amended later on. Finally, fine-tuned ontological patterns can be selected and applied repeatedly on structurally similar inputs. 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 (or TBox-reasoning) is emphasised in preference to large-scale instance reasoning (or ABox 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). In this paper, two types of ontology engineering patterns are examined to instantiate large scale ontologies based on part-whole relations for product modeling (Uschold and Callahan, 2004).

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 Deployment working group (W3C 2006) to better judge the performance of DL reasoners.

The group has published a set of design guidelines (Rector and Welty, 2005) to help users to develop OWL ontologies for the various types of part-whole relationships pointed out by Winston *et al.* (1987), and Artale *et al.* (1996). The Rector-Welty set of ontology engineering patterns can be expressed with the OWL-DL and OWL-Lite variants of the OWL language and is supported by existing OWL reasoners, like RACER (Haarslev and Möller, 2001), FaCT++ (Tsarkov and Horrocks, 2006), and Pellet (Sirin *et al.*, 2005).

All the OWL reasoners which support these patterns have architectures based on highly optimized tableau-calculus algorithms to compute the subsumption and satisfiability reasoning problems and their theoretical complexity depends on the specific variant of description logic which is supported. For example, reasoning tasks in OWL-Lite, a.k.a. SHIF(D+) have deterministic exponential time (EXPTIME) worst-case complexity (Zolin, 2006).

The usefulness of right identity rules to handle part-whole relations has first been demonstrated for medical ontologies like SNOMED (Spackman, 2000). This approach has not been included in the part-whole guidelines published by W3C because right-identity axioms are not yet included in the OWL standard. Horrocks and Sattler (2003) and Baader *et al.* (2005) have shown that adding this capability to some types of description logic reasoners can be done without impact on their decidability. CEL (Baader *et al.*, 2006), which is based on the EL+ description logic, is an interesting DL reasoner implementation which supports right-identity rules. In contrast with the OWL family, EL+ offers a trade-off between tractability of reasoning and expressivity of the language which favours the former, such that inference procedures such as ontology consistency checking, concept satisfiability, subsumption classification and instance checking have deterministic polynomial time complexity, even for cyclic TBoxes. EL+ users cannot use several basic OWL features such as *owl:allValuesFrom* and the domain and range assertions *rdfs:range* and *rdfs:domain* which depend on it. Concept disjunction (*owl:unionOf*) is also unavailable in the EL+ language.

EL+ supports a particular non-OWL feature called role inclusion, a form of role composition which enables the declaration of transitive roles (also in OWL) and right

identity (not in OWL). The medical example presented by Horrocks and Sattler (2003) shows how it is possible, with the help of a single right identity axiom, to remove the requirement to use disjunction as prescribed by the Rector-Welty pattern, and to allow the computation of a subsumption hierarchy over the EL+ ontology which is equivalent to the OWL-based one.

In this paper, the W3C-recommended approach based on OWL and an alternative approach based on EL+ are applied to instantiate large-scale ontologies describing part-whole relations between aircraft components and parts derived from the Service Difficulty Reports data collected through the Federal Aviation Administration (FAA) Flight Standard Service (2001). Sections 3 and 4 present the two types of ontology engineering patterns and how they are applied for this particular example.

Section 5 presents the methods used to generate the ontology sets and to evaluate reasoner performance. The results for FaCT++, RACER, Pellet and CEL and the supportive analysis are provided in Section 6 to illustrate the differences between the two approaches in terms of performance and scalability. Section 7 discusses the need for further work on ontology standards and tools to support the generation and use of large ontologies with an emphasis on the impact of the presence of dependency cycles on reasoner performance.

## **2 Background**

### **2.1 Related work**

There are three main categories of performance evaluation studies: reasoner, benchmark and production studies.

- **Reasoner studies** are papers usually 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 usually 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) (Guo *et al.*, 2004) is a popular benchmark designed against a model of ontology expressiveness with domain-realistic queries for ABox reasoning.
- **Production studies** are papers usually published by end users trying to use reasoners over their own large ontologies, and may or may not include comparative analyses of several 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 <i>et al.</i> , 2005 (RACER)	Reasoner + Benchmark
Sirin <i>et al.</i> , 2005, 2006 (Pellet)	Reasoner + Benchmark
Tsarkov & Horrocks 2005 (FaCT++)	Reasoner
Motik <i>et al.</i> , 2002, Motik 2006 (Kaon 2)	Reasoner + Benchmark
Baader <i>et al.</i> , 2006 (CEL)	Reasoner
Guo <i>et al.</i> , 2004 (LUBM)	Benchmark
Gardiner <i>et al.</i> , 2006	Benchmark
Dameron <i>et al.</i> , 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 and 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 2001). The resulting ontology has over 30,000 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 Air Transport Association (ATA) and Joint Aircraft System/Component (JASC) coding systems (FAA 2002).

### **2.2 Ontology normalisation and patterns**

Rector (2003) defines ontology normalization as the application of a limited number of criteria to eventually let the reasoner do the 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.

The pattern selected for this study is the one described in the “Simple part-whole relations in OWL ontologies” document edited by Rector and Welty (2005). This paper provides an estimate of the size of the ontologies which can be built with the proposed part-whole pattern.

## **3 Part-Whole ontology engineering**

### **3.1 The part-whole ontology engineering problem**

The Rector-Welty guidelines identify a range of common issues for the representation of nested part-whole relations in an ontology. One of the more interesting problems combines the requirements of a part inventory system and of a fault finding system to describe the devices made in a factory with a specific emphasis on the localization of faults. The objective of this work 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. The intention of the authors is to get the reasoner to infer the subsumption links between faults throughout a part-whole hierarchy, e.g. bolt-crankcase-engine-car in Figure 1. A lower-level fault, e.g. a fault in a bolt in the crankcase must be represented so that it can be successfully inferred to be a fault in the crankcase, then to be a fault in the engine and eventually to be a fault in the car.

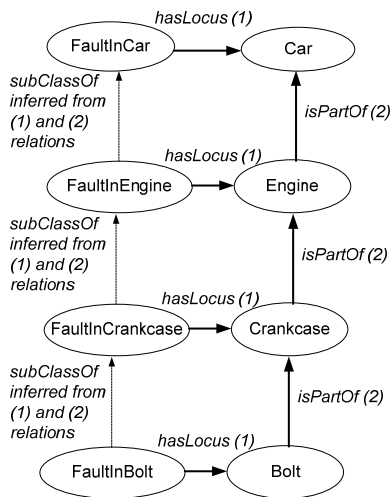


Figure 1: The part-whole ontology problem (adapted from the Rector-Welty guidelines)

### 3.2 The W3C part-whole ontology engineering pattern

The ontology engineering pattern Number 4 specified in the Rector-Welty guidelines is described in Figure 2 with the classic description logic notation recommended by Baader (2003).

```

isPartOf ◦ isPartOf   ⊆ isPartOf
Engine   ⊆ ∃ isPartOf . Car
Crankcase ⊆ ∃ isPartOf . Engine
Bolt     ⊆ ∃ isPartOf . Crankcase
FaultInCar ≡ Fault ⊓ ∃ hasLocus . ( Car ⊔ ( ∃ isPartOf . Car ) )
FaultInEngine ≡ Fault ⊓
    ∃ hasLocus . ( Engine ⊔ ( ∃ isPartOf . Engine ) )
FaultInCrankcase ≡ Fault ⊓
    ∃ hasLocus . ( Crankcase ⊔ ( ∃ isPartOf . Crankcase ) )
FaultInBolt ≡ Fault ⊓ ∃ hasLocus . ( Bolt ⊔ ( ∃ isPartOf . Bolt ) )

```

Figure 2: Ontology Engineering Pattern N. 4 (OWL)

This pattern is based on the following elements. First, the *isPartOf* relation is declared as transitive: this is necessary for the pattern to work repeatedly up the part-whole hierarchy. Then, the *isPartOf* relations between instances of concepts representing parts and sub-parts are declared. Finally, concepts describing Faults are defined as fault located at each level of the hierarchy with the existential restriction on the *hasLocus* role: here, the disjunction operator is used because it is more appropriate to exploit the *isPartOf* relation to identify all the relevant sub-parts.

The description logic language required for this pattern is ALCR+ (a.k.a. S). It is a subset of the languages used for OWL-Lite and OWL-DL.

### 3.3 EL+ alternative based on right-identity axiom

This alternative is derived from the approach first used for medical ontologies described in Horrocks and Sattler (2003). The two differences are the addition of the right-identity axiom and the simplification of the fault definitions which no longer require disjunctions.

$isPartOf \circ isPartOf \sqsubseteq isPartOf$   
 $hasLocus \circ isPartOf \sqsubseteq hasLocus$   
 $Engine \sqsubseteq \exists isPartOf . Car$   
 $Crankcase \sqsubseteq \exists isPartOf . Engine$   
 $Bolt \sqsubseteq \exists isPartOf . Crankcase$   
 $FaultInCar \equiv Fault \sqcap \exists hasLocus . Car$   
 $FaultInEngine \equiv Fault \sqcap \exists hasLocus . Engine$   
 $FaultInCrankcase \equiv Fault \sqcap \exists hasLocus . Crankcase$   
 $FaultInBolt \equiv Fault \sqcap \exists hasLocus . Bolt$

Figure 3: Alternative solution to Ontology Engineering Pattern N. 4 (EL+)

Figure 3 presents how this alternative approach works. The same declarations are required for the *isPartOf* relations and the Faults definitions are simplified at all levels of the part-whole hierarchy. The right-identity axiom defines *hasLocus*  $\circ$  *isPartOf* as a sub-property of *hasLocus* to signify that something that is located in a part P is also located in the places that P is a part of.

In practice, a fault which is located in the crankcase is inferred to be located in the places that crankcase is a part of such as Engine, and is then inferred to be a fault of the engine itself. Furthermore, because the *isPartOf* role is declared to be transitive and the appropriate part and component taxonomy is defined, a fault located in the car as a whole is also inferred.

## 4 From patterns to macros

### 4.1 Ontology engineering macros

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.

The ontology engineering problem described by Rector and Welty is a good example of a case where ontology engineering patterns can be repeatedly applied. XO (Lefort and Taylor 2005) is an ontology generation tool based on eXtensible Stylesheet Language (XSL) Transformations (Kay 2007) which can facilitate the development of semi-automatic conversions from XML to OWL and allow the creation of large ontologies. XO provides a programmatic syntax to specify where to put information sourced from relevant inputs in the resulting ontology. This syntax allows users to program what can be called ontology engineering macros. Macros in our ontology generation environment are an efficient method to apply the advice embedded in the best practice examples which are applicable. The Semantic Web community has not yet produced any standard language to describe ontology generation macros. The objective of the graphical notation proposed below is to provide a high level view of what is in practice implemented with the help of XO.

#### 4.2 Macros for Part-whole ontologies

To handle the example presented above, two types of macros are required, Exists and Exists\*.

Figure 4 presents the Exists(**A**, **B**, p) macro which corresponds to the standard existential restriction based on classes A and B and the property p.

$$A \sqsubseteq \exists p . B$$

Figure 4: Exists(**A**, **B**, p)

Exists\*(**SomethingAboutA**, A, Something, p, r) is a slightly more complex macro with two possible implementations. Figure 5 is the representation to be used if OWL is the selected description logic. Figure 6 presents the EL+ alternative.

$$p \circ p \sqsubseteq p$$

$$\text{SomethingAboutA} \sqsubseteq \text{Something} \sqcap \exists r (A \sqcup (\exists p A))$$

Figure 5: Exists\*(**SomethingAboutA**, A, Something, p, r) for OWL

$$p \circ p \sqsubseteq p$$

$$r \circ p \sqsubseteq r$$

$$\text{SomethingAboutA} \sqsubseteq \text{Something} \sqcap \exists r A$$

Figure 6: Exists\*(**SomethingAboutA**, A, Something, p, r) for EL+

Figure 7 presents how the Exists and Exists\* macros could be used to generate the description logic expressions listed in Figure 2 and Figure 3 respectively.

Exists(Engine, Car, isPartOf)  
 Exists(Crankcase, Engine, isPartOf)  
 Exists(Bolt, Crankcase, isPartOf)  
 Exists\*(Fault, Car, isPartOf, hasLocus)  
 Exists\*(Fault, Engine, isPartOf, hasLocus)  
 Exists\*(Fault, Crankcase, isPartOf, hasLocus)  
 Exists\*(Fault, Bolt, isPartOf, hasLocus)

Figure 7: Macros answering the part-whole problem

A simplified graphical notation is introduced in this paper to represent how these two ontology engineering macros are combined to allow the generation of an ontology out of any resource providing information about parts and the related part-whole relations.

In this notation, the calls to macros are represented by a single arrow which links the two principal classes.  $\text{Macro}(\mathbf{A}, \mathbf{B}, \text{parameters})$  is represented by an arrow from the class A to the class B. The label of this arrow is subsequently shortened to  $\text{Macro}(\text{parameters})$  to improve readability. The direction of the arrow shows which class definition depends on the other. An arrow from A to B with a dotted line is used to indicate that A is a subclass of a potentially complex description logic expression in which B is involved in some way. A full arrow from A to B is used for the case where B is defined as an equivalent class with respect to the expression implied by the macro. The text in the arrow specifies the other parameters required to execute the macro which can be class names (e.g. Fault) or property names (e.g. *hasLocus*, *isPartOf*). A *subClassOf* arrow linking A to B corresponds simply to the generation of the two classes and of their *subClassOf* relation. And a *subClassOf\** arrow is used to mark the results which are expected after the reasoner has re-built the class hierarchy.

The seven macro calls listed in Figure 7 are drawn in Figure 8 with this notation.

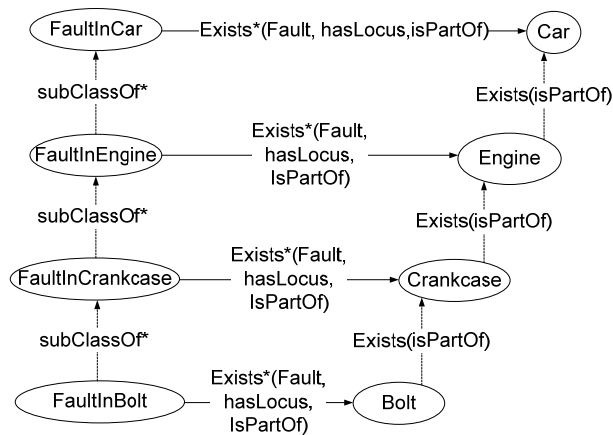


Figure 8: Graphical notation for part-whole macros

It should be noted that this notation is designed to help the programmers of the transformation supporting the ontology generation to document it. Figure 9 illustrates

what would be required to describe a solution based on an automatic generation approach for a case expanding this first example.

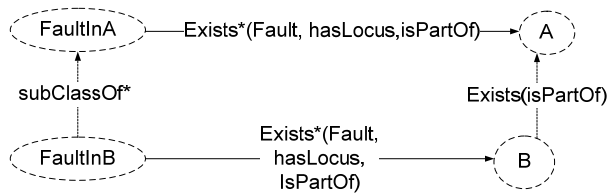


Figure 9: Generic part-whole macro

This graphical notation is used below to explain how the defined macros are applied for the real world example used to support this empirical evaluation.

### 4.3 Application to Service Difficulty Reports

The case study presented here is based on the Service Difficulty Reports online service (FAA 2001) collected by the Federal Aviation Administration. This aviation safety information collection programs has been in operation since 1966 (GAIN 2004) and is a valuable source of data for safety performance analysis (Luxhøj 1999). This research exploits the recorded information about parts, their physical and functional relationships to other parts and their functional affiliation to the categories defined by the ATA/JASC coding system (FAA 2002). In this context, the detection of repetitive incidents would benefit from the availability of an ontology combining the two types of relations between part and wholes: the physical part-whole hierarchy, which is sometimes available as the location of the service difficulty and the functional part-whole hierarchy which is currently managed with the help of the ATA/JASC coding system. The ATA/JASC coding system is a FAA-managed variant of ATA Specification 100 coding system, an international industry numbering standard developed to identify aircraft systems, sub-systems and components and support part procurement and maintenance operations.

Service difficulties are comparable to the faults present in the Rector-Welty example. However, in this case study, it is possible to apply the *Exists\** macro in two different ways to handle both the physical and the functional part-whole hierarchy of an aircraft. Figure 10 shows the application of the *Exists\*(X, Y, Difficulty, hasFunctionalLocus, isSubOf)* macro to manage the functional part-whole hierarchy. The *isSubOf* relations are derived from the JASC specification. The three levels at which the macros are exploited corresponds to different levels of the JASC coding system hierarchy. In practice, when the whole ontology is generated, three different passes over the input are done, for all system, sub-system and component categories.

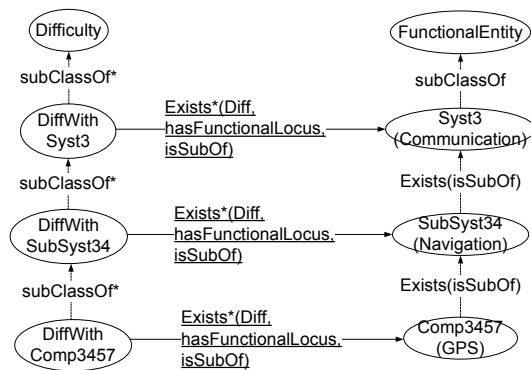


Figure 10: *Exists\** macros for the functional hierarchy

Figure 11 shows the application of the *Exists\*(X, Y, Difficulty, hasLocus, isPartOf)* macro to manage the physical part-whole hierarchy.

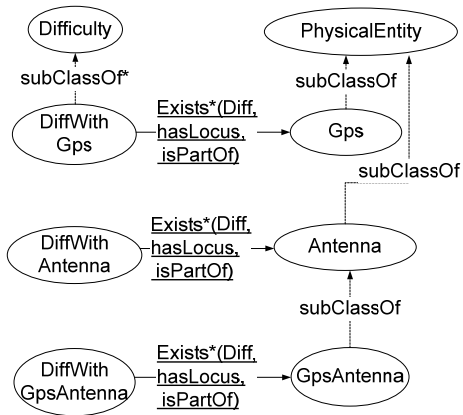


Figure 11: Exists\* macros for the physical hierarchy

The relation between the part and the whole is extracted with the help of the location field present for each SDR record. The Exists\*(**X**, **Y**, Difficulty, *hasLocus*, *isPartOf*) macro is applied to create the Difficulty classes for the Physical parts. The Exists(**X**, **Y**, *isPartOf*) macro is not applied here.

Because of the nature of the inputs, the *isPartOf* relation can only be considered in more restrictive conditions for cases where the part and the whole also share the same JASC code. This approach is illustrated in Figure 12. New classes are created to represent the intersection of the functional concepts identified in Figure 10 with the physical concepts identified in Figure 11, and named by a combination of the part name and the JASC code. For these classes, the two Exists\* macros are used jointly and the Exists(**X**, **Y**, *isPartOf*) can be declared.

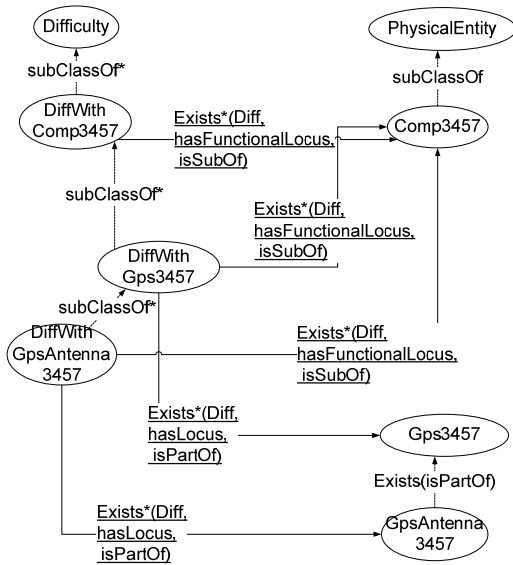


Figure 12: Joint use of functional and physical Exists\* macros

In practice, two different passes over the inputs are done to handle simultaneously the macros presented in Figure 11 and Figure 12. The first pass is required to create the simple classes like Gps, Antenna, Gps3457 and Antenna3457. The second pass, based on a join operation through the location field, creates the supplementary classes like GpsAntenna and GpsAntenna3457.

In the difficulty branch of the generated ontology, the hierarchy inferred from the physical part-whole relations is positioned below the hierarchy inferred from the functional relations. Figure 13 illustrates how the two hierarchies unfold together after the inferred hierarchy is computed by a reasoner.

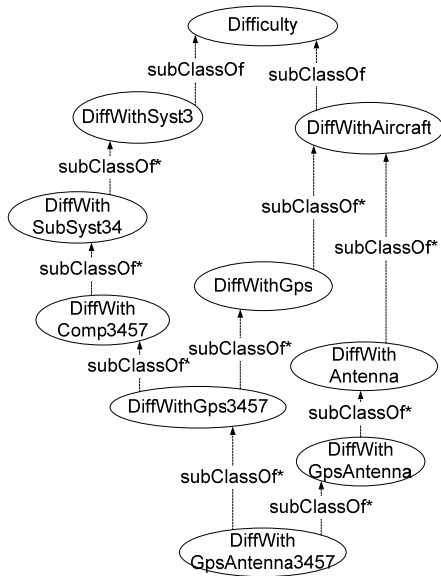


Figure 13: Combined physical / functional hierarchies

## 5 Experiment

### 5.1 Generation of the test ontologies

The XO tool can be used to generate ontology of various sizes to strengthen the evaluation outcomes.

The JASC coding system can be used to isolate definitions closely related in the domains in groups of various sizes. Figure 14 illustrates the selected groupings as follows:

The global aircraft ontology “o” handling all JASC codes in a single file;

- 8 one-digit ontologies “o1” to “o8” handling between 1 and 1000 four-digit JASC codes;
- 51 two-digit ontologies “o11” to “o85” handling between 1 and 100 four-digit JASC codes;

- And 301 four-digit ontologies “o1100” to “o8597” handling between 1 and 10 JASC codes.

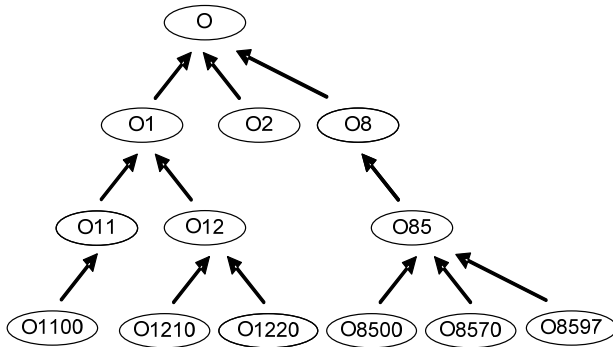


Figure 14: Multi-level ontology test samples

In this experiment, the test samples are generated for the OWL pattern at each of these levels. Because the current ontology engineering environment exploits versions of the OWL and the Description Logic Implementation Group’s Description Logic Interface format (DIG) standards which cannot support the role inclusion axioms required for the EL+ ontology, the easiest method to get the EL+ ontologies is to execute a conversion from the OWL ontology which replaces the outputs of the Exists\* macro described in Figure 5 by the corresponding content which would have been generated out of the action of the Exists\* macro described in Figure 6. This is done with the help of three specific tools: the first tool removes the disjunctions specific to the Rector-Welty pattern, the second tool translates the result into the Knowledge Representation System Specification (KRSS) format, and the third tool adds the two right identity axioms.

## 5.2 Evaluation of reasoner performance

Classification can be defined as the computation of the subsumption hierarchy for classes and properties. It is a minimum value which is representative of the initial work which has to be done before the reasoner can answer any specific request. 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 v. 1.1 interface (Bechhofer 2003). In this method, the classification time is obtained from the response time to a query which asks information for all concepts, triggered once the ontology has been loaded. All the OWL reasoners, FaCT++, RACER Pro, RACER and Pellet can be evaluated through their DIG interface with this method. This approach is implemented in the Ontology Reasoner Benchmark publicly available from the University of Manchester (see Gardiner *et al.* (2006) for more information), an implementation which supersedes the authors' adaptation of CDD, the Context-Driven Development Toolkit developed by Wagelaar (2004). The Ontology Reasoner Benchmark allows the user to supply a list of ontologies to evaluate and compiles the results automatically to facilitate the analysis. It re-starts the reasoner under evaluation every time a new ontology is submitted and triggers a timed abortion of the reasoning process for cases where the reasoner execution takes longer than a user-specified duration. The tool also measures the time for the reasoner response to a set of requests designed to retrieve the information required to rebuild the subsumption hierarchy. The DIG client embedded in Protégé (Knublauch *et al.* 2004) can also be used to evaluate classification times. The inconvenience of this approach is that it is based on a manual process which is impractical on large test samples, but it is a useful alternative for users willing to check the results produced by Ontology Reasoner Benchmark. To measure times, the user has to load an ontology and click the classify button of this tool. Four measures are then produced: the loading time, the check consistency time, and the times required to compute the inferred sub-classes and equivalent classes. The comparison

between the two methods is also important to detect cases where Protégé gives better results than the Ontology Reasoner Benchmark because its pre-processing of the loaded ontology is more efficient to suppress redundant *rdfs:subClassOf* statements which are implicitly contained by *owl:equivalentClass* statements.

### 5.3 Computation of size and complexity indicators

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. Because real world data is used to generate the test samples, the size and complexity of the generated ontologies can change in many ways. This means that several size and complexity indicators are required to capture the differences between the samples and support the analysis of the reasoner performance. The complexity indicators used for this study are listed in Table 2.

Indicators/Numbers	Macros/Patterns
Classes	
Maximum and average role depth	
Existential restrictions with disjunctions	Exists*
Existential restrictions without disjunctions	Exists
Classes containing exist. restrictions with disjunctions	(Difficulty classes)

Table 2: Complexity indicators

The number of classes is generally a good indicator of the sample size but is not sufficient to provide an idea of the complexity of the ontology. Some authors (e.g. Baader, personal communication) recommend the use of role depth, as the main secondary indicator for this purpose. Figure 15 shows the subset of the formal definition

from Baader and Sattler (1999) which is applicable for ALCR+ ontologies, and is used to define role depth in the context of this study.

$$\text{Depth}(C1 \sqcap C2) = \text{Max}(\text{Depth}(C1), \text{Depth}(C2))$$

$$\text{Depth}(C1 \sqcup C2) = \text{Max}(\text{Depth}(C1), \text{Depth}(C2))$$

$$\text{Depth}(\exists r.C) = 1 + \text{Depth}(C)$$

Figure 15: Role depth definition

Counting the number of existential restrictions can help to understand the complexity of each sample with respect to the different types of relations: *isPartOf*, *isSubOf*, *hasLocus*, *hasFunctionalLocus*. Counting the number of existential restrictions with disjunctions is an indicator of the number of occurrences of Exists\* patterns. The number of classes which contain existential restrictions with disjunctions is also a useful indicator because it is the size of the Difficulty branch of the ontology.

A customized XSL transformation processing the DIG 2.0-formatted version (Turhan *et al.* 2006) of the OWL ontologies is used to calculate the role depths, the number of occurrences of Exists\* and Exists patterns of various kinds and the number of Difficulty classes.

#### 5.4 Experimental setup

Our XO tool is available through an Apache 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. This environment includes several utilities to convert the generated OWL file in the formats (KRSS, DIG 2.0) which are useful to perform the test and to compute the indicators supporting the analysis of the results.

The Ontology Reasoner Benchmark tool was used to measure the results, except where specified otherwise, in CPU seconds on a PC workstation (Windows) with a 3 GHz CPU and 2 Gb of RAM. After 300 CPU seconds, the classification task is terminated and considered to have taken too long for practical purposes. The CEL results are based on a Linux workstation running over comparable hardware.

Figure 16 gives an overview of this working environment

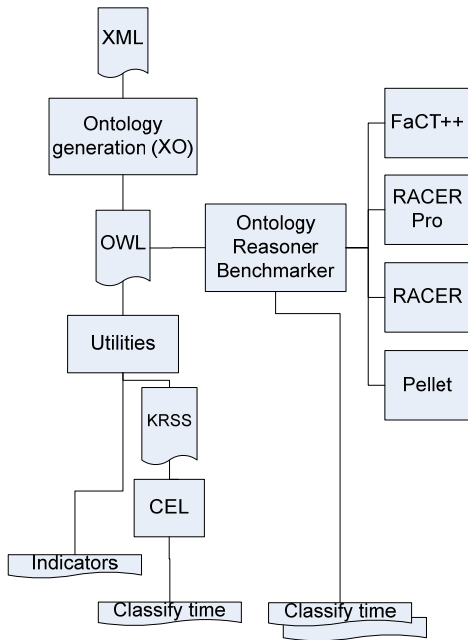


Figure 16: Experiment setup

Table 3 lists the used versions of the evaluated description logics reasoners and their critical configuration parameters.

Tool	Param.	OS	Hardware
Fact++ 1.1.4	N/A	Win XP	PC 3Ghz 2 Gb RAM
RACER Pro 1.9.0	9551	Win XP	idem
Racer 1.17.24	Stack size 45M	Win XP	idem

Pellet 1.3	Heap size 1024m	Win XP	idem
CEL 0.94	N/A	Linux Debian(sid )	idem

Table 3: Tools versions and configurations

## 6 Results

The main outcome of this evaluation is an estimate of the maximum size of OWL and EL+ ontologies which can be processed in a given time with OWL and EL+ for a range of description logics reasoners. This upper limit is very important to users in practice, especially in the context of manual tasks and can also be used to set the timeout threshold for experiments such as this one.

These results can be used to evaluate the relative performance of several OWL reasoners on the ontologies based on the Rector-Welty pattern and to compare it to the performance of CEL on the EL+ ontologies. This illustrates the relatively superior tractability of CEL and its ability to scale for the full aircraft configuration.

### 6.1 Characteristics of the OWL and EL+ ontologies

Table 4 provides an overview of the size of generated ontologies for some of the complexity indicators listed in Table 2. To give an idea of the progression of these values over the totality of the sample, the maximal and average values are given for each of the four levels used for the generation (see Figure 14).

Ontologies	Nb of classes	Avg role depth	Exists* (physical)	Exists* (functional)	Exists
All-in-one	30281	1.98	14233	10636	5595
1-digit o. (Max)	9551	2.49	4543	3141	1888

1-digit o. (Avg)	3918	1.98	1844	1298	789
2-digits o. (Max)	2641	2.56	1285	855	520
2-digits o. (Avg)	661	1.88	333	194	179
4-digits o. (Max)	851	2.03	415	254	181
4-digits o. (Avg)	126	1.82	56	35	31

Table 4: Measured complexity

A consequence of the generation process which has been defined is that all the ontologies are built on the same mould. The number of Difficulty classes is always almost equal to half of the number of classes. This ratio is representative of what users of the Rector-Welty pattern should obtain because one Fault class is needed for each of the classes belonging to the part-whole hierarchy.

Table 4 also shows that there is little variation amongst the indicators that do not depend on size such as the average role depth. This homogeneity of the tested samples, which is also apparent in Figure 17 shows the control over the test samples which is enabled by the benchmarking method used here.

## 6.2 Performance on OWL ontologies

The 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 17 which illustrates this analysis for the selection of the top 20 most challenging ontologies – that is the 20 ontologies with the greatest number of classes. Figure 17 uses a logarithmic scale to facilitate the comparison between the classification time (in

milliseconds) and the numbers used to characterize the complexity of the test samples described in Table 2.

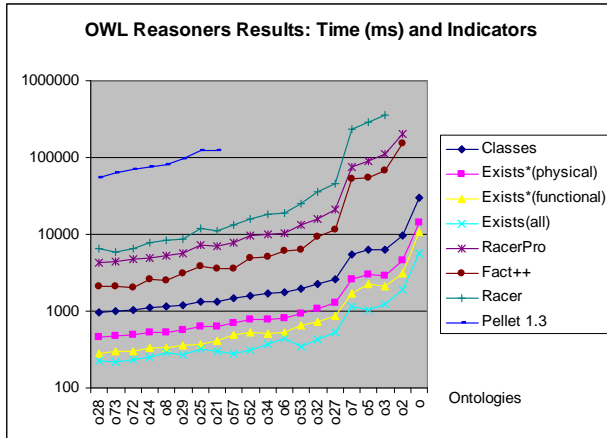


Figure 17: Reasoners results on top 20 OWL ontologies

Figure 18 presents a compilation of the results generated at all levels for the four OWL reasoners. The chart is a scatter plot with the number of classes on the X axis and the classification time as the Y axis in milliseconds. A polynomial trend line by Excel is also given.

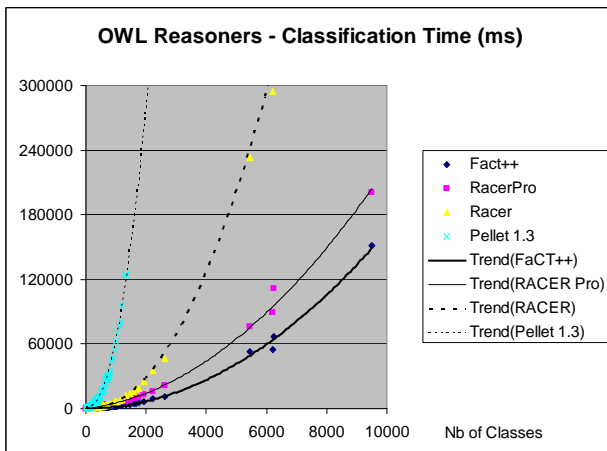


Figure 18: Compiled results for OWL reasoners

A rough estimate of the maximal number of classes which can be attained for a set classification time threshold of 3 minutes (180000 ms) can be extracted from Figure 18 for each reasoner. A summary of these results is provided in Table 5.

Reasoner	Fact++	RACE R Pro	Racer	Pellet 1.3
Classes	~10,000	~9000	~4500	~1500

Table 5: Empirical bounds for a time limit of 3 min

### 6.3 Performance on EL+ ontologies

Figure 19 provides the results collected for CEL with the top 20 ontologies, ordered according to CEL's classification time. CEL takes 2 minutes and 8 seconds to classify the whole Aircraft Part ontology, O, which contains more than 30000 classes (see Table 3 for more information on it). This value corresponds to the point on the far right in Figure 19.

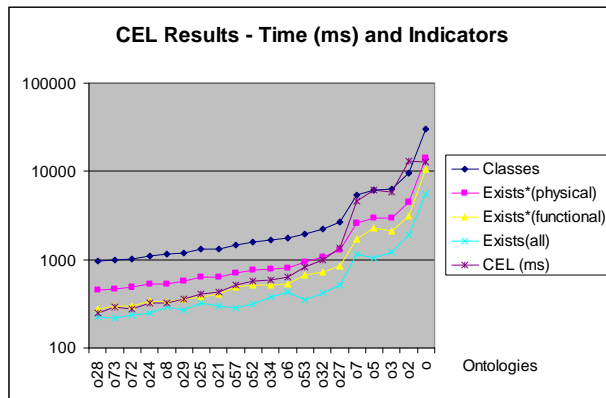


Figure 19: CEL results on top 20 EL+ ontologies

To compute the empirical bounds from the results collected for CEL, we can apply the same method used before, also with a polynomial trend line. This is shown in Figure 20. The extrapolation of the trend line would lead to a value greater than 35000 classes for a classification time of 3 minutes.

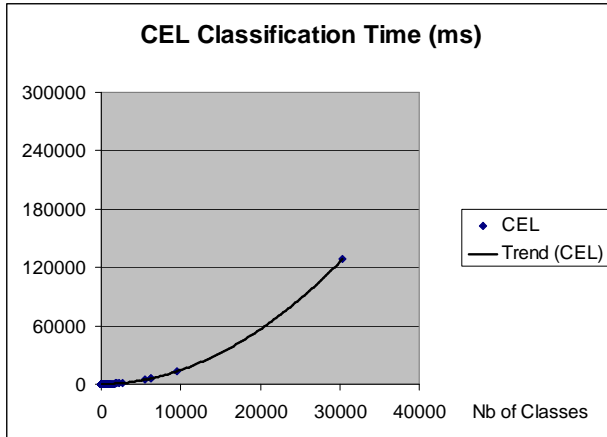


Figure 20: Compiled results for CEL

#### 6.4 Comparison of OWL and EL+ patterns

The result of this experiment shows the superior tractability of EL+ reasoners over OWL reasoners. For the ontologies with a size between 1000 and 10000 classes, a direct comparison is possible: Figure 21 shows that CEL is roughly ten times faster than the best of the OWL reasoners, FaCT++, for this size interval.

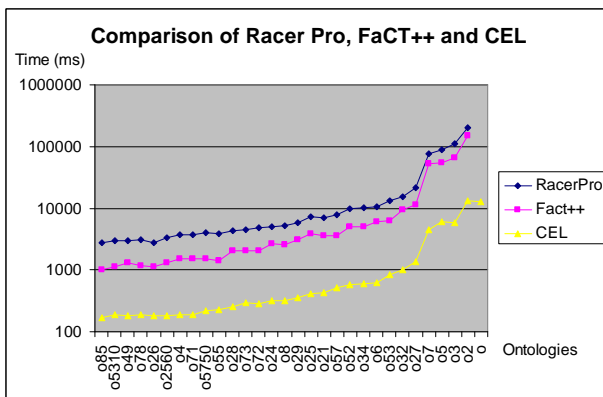


Figure 21: Comparison on top 20 ontologies

For FaCT++ and RACER Pro, results from Protégé (version 3.2.1) to estimate the classification time are partially available for on “o”, the largest ontology. In both cases, the interaction between Protégé and the reasoner ends in an error situation: FaCT++ performs the classification in a time of 1318 seconds, almost exactly 10 times more than

CEL, before an “Out of Memory” error occurs on the Protégé side, probably because the reasoner output triggers changes for more than half of the total number of classes. For RACER Pro, the exchange protocol ends by an error on the reasoner according to Protégé, so no conclusion can be drawn from the times spent in the earlier phases of the interaction.

## 7 Discussion

### 7.1 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 scalability limits in the currently available guidelines from W3C. 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 (Cuenca Grau 2006) is finalised.

The main objective of this research is to develop methods to automatically generate ontologies which belong to the first two categories listed in Table 6.

Scalable modelling style	TBox / Abox	Reasoners
SNOMED-like ontologies	TBox	CEL FaCT++
GALEN-like	TBox	FaCT++,

ontologies		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

The EL+ version of the Exists\* macro is applicable to get SNOMED-like ontologies. The Rector-Welty macro is relevant for the generation of GALEN-like ontologies. This is why it is interesting to compare the results presented above to the ones obtained over the OWL version of GALEN (Rector and Rogers 2004) in the same evaluation environment. The closest ontology to GALEN in the test samples in terms of size and complexity is O27. Table 7 illustrates some of the differences between the two ontologies.

Ontologies	Classes	Avg role depth/ Max role depth	Exists
O27	2641	1.95/5	520
GALEN	2749	0.75/9	1145

Table 7: Comparison with GALEN - complexity

Table 8 results shows that while RACER Pro performance on GALEN is roughly comparable to the one from our evaluation on O27, the FaCT++ results are far worse. And the results are different when the Ontology Reasoner Benchmark is used (230 seconds) and when the measure is estimated directly with Protégé (59 seconds).

Ontologies	FaCT++	RACER Pro	CEL
O27	11.5 s	21.3 s	1.34 s
GALEN	59-230 s	21.6 s	N/A

Table 8: Comparison with GALEN - performance

This variation is caused by the presence of dependency cycles in this version of GALEN. The FaCT++ trace lists 59 cycles (or cyclic dependences) in total.

This confirms that the presence or absence of cycles is a major factor in the reasoner performance. All the results presented in Section 6 are valid only in cases where a specific effort has been made to suppress all cycles to avoid the significant degradation in reasoner performance which occurs in their presence.

## 7.2 Handling cyclic axioms and complex GCI chains

The working environment we have defined allows us to monitor the cyclic axiom warnings (Haarslev *et al.*, 2005) reported by some reasoners. In its standard output, FaCT++ identifies the root class of the detected cyclic dependences. RACER, version 1.7.24, when used directly in the verbose mode (option `-v`) traces all the classes involved in cycles. In practice, we need to get this type of information more seamlessly and more homogeneously amongst reasoners and extend their services to identify the classes involved in cycles individually, with as much detail as available. A possible approach would be to extend the DIG interface with such services to help users of tools such as Protégé to better understand how the reasoner performance is affected by the choice of the ontology engineering pattern.

In this experiment, the repeated application of the part-whole pattern leads to relatively long chains of generalized concept inclusion axioms (GCIs) where *owl:equivalentClass* and *rdfs:subClassOf* constructs are used simultaneously. Here, the first challenge for the ontology designer is to eliminate as many redundant GCI statements to avoid potentially costly GCI elimination effort on the reasoner side such as role absorption (Tsarkov and Horrocks 2005). The second challenge is to prevent the apparition of possible modelling

errors, especially those corresponding to chained GCIs where a concept is inappropriately defined in terms of itself.

Currently, these challenges are difficult to manage, even for experienced users; to avoid cases where the reasoner response time for classification becomes unacceptable due to these cases, the one technique may be to replace the use of *owl:equivalentClass* with *rdfs:subClassOf* where it may be done so without significantly impacting on the semantics of the terminological model, and such that the ontology is not invalidated. We have also observed that the use of Protégé to systematically remove redundant GCIs is beneficial to the performance of some reasoners.

Using CEL is a very effective work around for this specific issue. CEL does not incur any reasoning time penalty for the presence of such cycles thanks to the simplification of the algorithm which is allowed by the choice of the EL+ description logic.

It is too early to conclude whether RACER Pro is or is not significantly affected by the presence of cycles. This analysis would require the reinstatement of the traces available in RACER 1.17.24 or the implementation of the extension of the DIG interface suggested above.

### **7.3 Scalability issues for Aircraft ontologies**

The ontologies which are used for this evaluation have grown in size and complexity w.r.t. the examples used in our previous work (Lefort *et al.* 2006) because of the ontology design choice to represent the functional and physical part-whole hierarchy and to introduce the supplementary classes where the two macros are jointly used presented in Figure 12.

It should be possible to downsize and re-factor the ontology created for this experiment to a more manageable size, probably superior to the core 4,000 classes found in this previous iteration of our work. The applicability of such a refactored ontology to practical operational problems would still be limited because of the focus of Service Difficulty Reports on maintenance events.

The work presented here also anticipates the transformation of more detailed sources such as aircraft maintenance manuals into even larger and richer ontologies.

## **8 Conclusion**

Reasoner performance over large ontologies with large TBox components is highly variable and dependent on the choice of the ontology engineering patterns. Previous experience in the creation and normalization of TBox-based ontologies (Rector 2003) has been disseminated by W3C through the publication of a set of recommendations on simple part-whole relations in OWL Ontologies (Rector and Welty 2005). This research is a continuation of this work and the evaluation work presented in Lefort *et al.* (2006) centred on the OWL-based ontology engineering pattern defined by the W3C contributors. The main contribution of this case study is the implementation and evaluation of an alternative approach taking advantage of features which are specific to the EL+ description logic.

This experiment confirms the difference in tractability between the OWL and the EL+ classes of reasoners. In the conditions of this experiment, CEL classification times are routinely an order of magnitude better than the fastest of the OWL reasoners, FaCT++. One useful result of this evaluation is an estimate of the maximal size of an ontology generated with the two types of Exists\* macros which can be classified in less than three

minutes. For this threshold, the CEL limit is well over 30,000 classes, the maximum size of the ontologies used for this evaluation, while the best performing OWL reasoner is FaCT++, followed closely by RACER Pro, with ontologies of up to 10,000 classes in the set response time threshold.

The other lessons learned from this experience are that OWL users are likely to get trapped into cases where the reasoner response time becomes unacceptable just because of the susceptibility of OWL reasoners to cyclic dependencies which are not easy to manage, even for experienced users. The results presented here show that it is possible to apply the Rector-Welty pattern and avoid the presence of such cycles. The GALEN example illustrates the steep degradation in performance which can be experienced when this is not the case.

The efficient representation of part-whole hierarchies is a critical requirement for real world applications in the aerospace domain and also for very large medical ontologies such as SNOMED. In this context, the time and size gains which are enabled by the switch to an ontology engineering pattern based on EL+ are quite significant. This is why the integration of the non-standard features of EL+ in the future versions of the OWL standard and the update of the DIG protocol to enable tools such as Protégé to interact with CEL should be accelerated. This effort will also support the introduction of right-identity axioms for description logics of the OWL family, with potential performance improvement on this side too.

## **9 References**

- ARTALE, A., FRANCONI, E. GUARINO, N. and PAZZI, L. (1996) Part-whole relations in object-centered systems: An overview, *Data and Knowledge Engineering*, **20**(3) 347-383.
- BAADER, F. (2003) Appendix 1: Description Logics Terminology, in *The Description Logic Handbook: Theory, Implementation, and Applications*, BAADER, F., CALVANESE, D., MCGGUINNESS, D. L., NARDI, D., and PATEL-SCHNEIDER, P. F. (eds), Cambridge University Press, New York, NY, 485-495.
- BAADER, F., BRANDT, S., and LUTZ, C. (2005) Pushing the EL Envelope, in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 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 third International Joint Conference on Automated Reasoning*, Seattle, WA, Lecture Notes in Artificial Intelligence, **4130**, 287–291.
- BAADER, F. and SATTLER, U. (1999) Expressive number restrictions in description logics, *Journal for Logic and Computation*, **9**(3) 319-350.
- BECHHOFER, S. (2003) The DIG Description Logic Interface: DIG/1.1, in *Proceedings of Description Logics 2003 Workshop*, Rome, Italy, CEUR Workshop Proceedings, **81**, ceur-ws.org.
- CUENCA GRAU, B. (ed) (2006) *Tractable Fragments of the OWL 1.1 Extension to the W3C OWL Web Ontology Language*, Editor's Draft, June, University of Manchester. Available online at <http://owl1-1.cs.manchester.ac.uk/Tractable.html>.

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*, Washington DC, 181-185.

DEAN, M. and SCHREIBER G. (2004) *OWL Web Ontology Language Reference*, W3C Recommendation, February. Available online at <http://www.w3.org/TR/owl-ref/>.

FAA (2001) *FAA Flight Standard Service: FAA Service Difficulty Reporting*, Version 2.0. Available online at <http://av-info.faa.gov/isdr/>.

FAA (2002) *Joint Aircraft System/Component Code Table and Definitions*, Federal Aviation Administration, Flight Data Services, Regulatory Support Division, Aviation Data Systems Branch AFS-620, Oklahoma City, OK. Available online at <http://av-info.faa.gov/isdr/>.

GAIN (2004) *Updated List of Major Current or Planned Government Aviation Safety Information Collection Programs*, Global Aviation Information Network. Available online at [http://204.108.6.79/products/products\\_GST.cfm](http://204.108.6.79/products/products_GST.cfm).

GARDINER, T., TSARKOV, D. and HORROCKS, I. (2006) Framework For an Automated Comparison of Description Logic Reasoners, in *Proceedings of the 2006 International Semantic Web Conference*, Athens, GA, Lecture Notes in Computer Science, **4273**, 654-667.

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, Lecture Notes in Computer Science, **3298**, 274-288.

HAARSLEV, V. and MÖLLER, R. (2001) RACER System Description, in *Proceedings of the First International Joint Conference on Automated Reasoning*, Siena, Italy, Lecture Notes in Computer Science, **2083**, 701-706.

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*, Edinburgh, Scotland, UK, CEUR Workshop Proceedings, **147**, ceur-ws.org.

HORROCKS, I. and SATTLER, U. (2003) Decidability of SHIQ with complex role inclusion axioms, *Artificial Intelligence*, **160**(1-2) 79-104.

KAY, M. (2007) *XSL Transformations (XSLT) Version 2.0*, W3C Recommendation January 2007. Available online at <http://www.w3.org/TR/xslt20/>.

KNUBLAUCH, H., FERGERSON, R. W., NOY, N. F. and MUSEN, M. A. (2004) The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications, in *The Semantic Web - Proceedings of the Third International Semantic Web Conference*, Hiroshima, Japan, Lecture Notes in Computer Science, **3298**, 229-243.

LEFORT L. and TAYLOR, K. (2005) Large scale colour ontology generation with XO, in *Proceedings of the Australasian Ontology Workshop (AOW 2005)*, Sydney, Australia, Conferences in Research and Practice in Information Technology, **58**, 47-52.

LEFORT, L., TAYLOR, K. and RATCLIFFE D. (2006) Towards Scalable Ontology Engineering Patterns: Lessons Learned from an Experiment based on W3C's Part-whole Guidelines, in *Proceedings of the Australasian Ontology Workshop (AOW 2006)*, Hobart, Australia. Conferences in Research and Practice in Information Technology, **72**, 31-40.

- LUXHØJ, J. T. (1999) Trending of equipment inoperability for commercial aircraft *Reliability Engineering & System Safety*, **64**(3), 365-381.
- MOTIK, B. (2006) *Reasoning in Description Logics using Resolution and Deductive Databases*, PhD thesis, University of Karlsruhe, Germany.
- 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, CEUR Workshop Proceedings, **79**, 39–50, ceur-ws.org.
- NICOLAI, T., SINDT, T., KENN H., and WITT H. (2005): Case Study of Wearable computing for Aircraft Maintenance. In *Proceedings of the Second International Forum on Applied Wearable Computing (IFAWC)*, Zürich, Switzerland, March 17-18, 2005.
- RECTOR, A. (2003) Modularisation of domain ontologies implemented in description logics and related formalisms including OWL, in *Proceedings of the second international Conference on Knowledge Capture*, Sanibel Island, FL, ACM Press, New York, NY, USA, 121-128.
- RECTOR, A. and ROGERS J. (2004) Patterns, Properties and Minimizing Commitment: Reconstruction of the GALEN Upper Ontology in OWL, in *Proceedings of the EKAW\*04 Workshop on Core Ontologies in Ontology Engineering*, Northampton, UK, CEUR Workshop Proceedings, **118** , ceur-ws.org.
- RECTOR, A. and WELTY, C. (eds) (2005) *Simple part-whole relations in OWL Ontologies*, W3C Editor's Draft, Version 1.5, August. Available online at <http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/index.html>.

REISS, M., MOAL, M., BARNARD, Y., RAMU, J.-Ph., and FROGER, A. (2006) Using Ontologies to Conceptualize the Aeronautical Domain. In *Proceedings of the International Conference on Human-Computer Interaction in Aeronautics (HCI-AERO 2006)*, Seattle, WA, September 20-22, Cépaduès-Editions. Toulouse, France, 56-63.

SIRIN, E., PARSIA, B., CUENCA GRAU, B., KALYANPUR, A. and KATZ, Y. (2005) *Pellet: A Practical OWL-DL reasoner*, University of Maryland Institute for Advanced Computer Studies, Technical Report 2005-68.

SPACKMAN, K. A. (2000) Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT, *Journal of the American Medical Informatics Association*, Fall Symposium 2000, Special Issue.

TURHAN, A.-Y., BECHHOFFER, S., KAPLUNOVA, A., LIEBIG, T., LUTHER, M., MÖLLER, R., NOPPENS, O., PATEL-SCHNEIDER, P., SUNTISRIVARAPORN, B. and WEITHÖNER, T. (2006) DIG 2.0 Towards a Flexible Interface for Description Logic Reasoners, in *Proceedings of the second international workshop OWL: Experiences and Directions*, Athens, GA, CEUR Workshop Proceedings, **216**, ceur-ws.org.

TSARKOV, D. and HORROCKS, I. (2005) Optimised classification for taxonomic knowledge bases. In *Proceedings of the 2005 Description Logic Workshop (DL 2005)*, Edinburgh, Scotland, UK, CEUR Workshop Proceedings **147**, ceur-ws.org

TSARKOV, D. and HORROCKS, I. (2006) FaCT++ description logic reasoner: System description, in *Proceedings of the third International Joint Conference on Automated Reasoning*, Seattle, WA, Lecture Notes In Artificial Intelligence, **4130**, 292-297.

USCHOLD, M. and CALLAHAN, S. (2004) Unifying Knowledge and Product Data, in *Proceedings of the NASA Workshop on the Knowledge Integrating Virtual Iron Bird*, Monterey, CA. Available online at <http://ic.arc.nasa.gov/vib/>.

W3C (2006) *Semantic Web Best Practice and Deployment Working Group*, October 2006. Available online at <http://www.w3.org/2001/sw/BestPractices/>.

WAGELAAR, D. (2004) Context-Driven Model Refinement, in *Proceedings of the Model Driven Architecture®: Foundations and Applications workshop*, Linköping, Sweden, Lecture Notes in Computer Science, **3599**, 189-203.

WINSTON, M. E., CHAFFIN, R. and HERRMANN D. (1987) A Taxonomy of Part-Whole Relations, *Cognitive Science*, **11**, 417-444.

ZOLIN, E. (2006) Complexity of reasoning in Description Logics, Available online at <http://www.cs.man.ac.uk/~ezolin/logic/complexity.html>.