The TDDonto Tool for Test-Driven Development of DL Knowledge Bases

Agnieszka Ławrynowicz[1], C. Maria Keet[2]

[1] Institute of Computing Science, Poznan University of Technology, Poland
agnieszka.lawrynowicz@cs.put.poznan.pl
[2] Department of Computer Science, University of Cape Town, South Africa
mkeet@cs.uct.ac.za

Abstract. Adding knowledge to an ontology lacks a verification step by the modeller in most cases, other than 'try and see what the reasoner says about it'. This is due to the lack of a systematic testbed for ontology authoring. Reusing the notion of *Test-Driven Development* (TDD) from software engineering for ontology development resulted in the specification of 42 test types for the $\mathcal{SROIQ}$ language features, as TBox tests using its axioms and as ABox-driven tests with explicitly introduced individuals. We developed TDDOnto, which implements that subset of the TDD tests that could be done by leveraging extant technologies. We examined what the most efficient implementation strategy is with 82 ontologies. The TBox SPARQL queries with OWL-BGP were faster than the ABox-based approach except for disjointness, that effect is more pronounced with larger ontologies, and the OWL API approach is faster than the SPARQL queries for OWL 1 ontologies. A significant difference in performance between OWL and OWL 2 DL ontologies was observed. On average, the TDD tests are faster than classification reasoning, indicating that TDD tests are a promising alternative to the 'try and see' approach in ontology authoring.

1 Introduction

New methods are being proposed for the *ontology authoring* process that concerns writing the actual axioms to augment the overarching ontology engineering methodologies. They may be in the form of, among others, pitfall detection [16], foundational ontology-driven axiom suggestions [8], reasoner-driven suggestions with FORZA [9], or example-based ones with *advocatus diaboli* [3]. Such methods all constitute certain forms of small 'tests' on the knowledge represented in the ontology; e.g., FORZA, by using a description logic (DL) reasoner, analyses domain and range restrictions of roles and concept subsumption to propose axioms that will not lead to an inconsistency, and *advocatus diaboli* computes the logical consequences of the absence of a disjointness axiom between concepts. They are useful, yet limited in scope with respect to all possible axiom patterns one could add to an ontology. To address computing consequences of *all* possible types of axioms requires a different approach to testing an ontology. So-called 'test-last' approaches use either Competency Question-driven authoring [17] or unit tests

[20], or variations thereof (e.g., [4, 2]), where first a change is made and then examined on correctness. Test-driven development (TDD), on the other hand, is a *test-first* approach, as used in software engineering: one writes a test for some functionality, checks that it is not present (i.e., the test fails), writes the code, tests again and it should pass with the previous ones still passing as well [1]. In the setting for ontologies, the 'functionality' corresponds to the knowledge (vocabulary plus axioms) in the DL knowledge base, and the 'coding' to adding knowledge. For instance, when one wants to add $C \sqsubseteq D$, then first there would need to be a test on the ontology's vocabulary regarding presence of C and D, then it should hold that O does not model (or it is not present) $C \sqsubseteq D$ (return false in a TDD test), and one can add a test that its negation does not hold. Practically, one could use SPARQL-OWL and its implementation in the OWL-BGP system [11] and query for SubClassOf(?x D), or use the DL query tab in Protégé, or add a new individual a to the ABox, assert $C(a)$, classify the ontology, and check whether $D(a)$ is inferred. If it is not, then the test fails, i.e., $O \nvDash C \sqsubseteq D$, as intended, and the knowledge can be added and the test ran again. Given that DLs have only a limited set of language constructs, one can devise a test for each language feature, so that the modeller can populate it with whatever domain knowledge the user wants to add, and test it. There are multiple options to realise the TDD tests, once defined: 1) 'ABox tests' that focus on the instance-level, like TDD for UML Class Diagrams [19], so that instances of the assumed-to-be-added knowledge have to be added and behave in a certain way (classify into the right class, cause inconsistency, etc., as with $C(a)$, above), or 2) 'TBox tests' at the TBox-level, with either just SPARQL or DL queries or with the automated reasoner only.

It is unclear whether 'TBox tests' or 'ABox tests' are better, whether to use the reasoner just for classification or for querying for the information, and to what extent the test and the ontology affect performance, and if one or the other technology has more or less limitations so that it has a higher/lower coverage of the TDD tests. As only one TDD tool exists that covers class subsumption only, uses Tawny-Owl notation, and relies on the not widely-used Clojure [21], we have defined TDD tests for $\mathcal{SROIQ}$ and aim to obtain answers with the proof-of-concept TDD plugin, named TDDonto, a testbed, and a set of 67 OWL and 15 OWL 2 DL ontologies of various sizes and with different expressiveness. The ABox approach with mock objects had the worst performance for most TDD tests, the query-based approach with OWL-BGP had better performance, and the OWL API the best. While generally the TDD tests with OWL were faster than with OWL 2 DL ontologies, the TDD test for class equivalence and for property domain/range were faster, on average, for OWL 2 DL ontologies than the OWL ontologies. Further, it is worthwhile to observe that the TDD tests are faster than classification reasoning, suggesting that TDD tests are a promising avenue for reducing ontology development time as a 'shortcut' to where otherwise classification reasoning may have been used by a modeller. The main contributions are thus: 1) implementation of the TDD tests; 2) Comprehensive performance evaluation of the three core approaches to realising TDD tests of

DL-based ontologies; and 3) The basic version of a TDD plugin in Protégé so that the early-adopter ontology engineer can start using it.

In the remainder of this paper, we first briefly describe some preliminaries of the TDD tests and present a selection of them in Section 2, and summarise the system design considerations in Section 3. Section 4 describes the experiments and reports on the results, and we discuss and conclude in Sections 5 and 6.

2 Preliminaries: TDD test specifications

The generalised TDD principle for ontologies, informed by TDD in software engineering, is as follows, in short, in the default case:

1. Require: domain axiom x of type X is to be added to the ontology; e.g., x may be $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}.\mathsf{Course}$, which has pattern $C \sqsubseteq \exists R.D$.
2. Check the vocabulary elements of x are in ontology O (itself a TDD test);
3. Run the TDD test:
   (a) The first execution should fail (check $O \nvDash x$ or not present),
   (b) Update the ontology (add x), and
   (c) Run the test again which then should pass (check that $O \models x$) and such that there is no new inconsistency or undesirable deduction
4. Run all previous successful tests, which still have to pass (i.e., regression testing); if not, resolve conflicting knowledge.

There are two options for the TDD tests: at the TBox-level with ontology's TBox axioms (where possible), and using individuals explicitly asserted in the ABox using mock objects similar to the TDD approach for UML class diagrams by [19] (but then in the DL setting and by availing of the automated reasoner). Tests for both approaches have been specified [7]. For the test specifications we use the usual DL notation, with $C, D, ...$ denoting DL concepts, $R, S, ...$ DL roles, and $a, b, ...$ individuals, as for $\mathcal{SROIQ}$ in [5].

SPARQL-OWL [11] is used for the TBox tests. Its notation uses OWL functional syntax-style notation [14] merged with SPARQL's queried objects (i.e., ?x) for the formulation of the query, and adding a variable for the query answer; e.g., $\alpha \leftarrow$ SubClassOf (?x D) returns all subclasses of class D. SPARQL-OWL's query rewriting is described in [11] and is implemented in OWL-BGP[3].

Some of the TBox and all ABox tests require additional elements for testing that have to be removed after the test terminates successfully; they are referred to as *mock class* for a temporary OWL class, *mock individual* for a temporary individual, and *mock axiom* for any auxiliary axiom, alike the "mock objects" in TDD for software development [13, 10].

Steps 2 and 3a in the sequence listed above may give an impression of epistemic queries. However, we only need to check whether an element is in the vocabulary of the TBox of the ontology (in V_C or V_{OP}); i.e., the epistemic-sounding 'not asserted in or inferred from the ontology' refers to whether an ontology has some entity in its vocabulary, not whether it is 'known to exist' in

[3] `https://github.com/iliannakollia/owl-bgp`

one's open or closed world. Note also the subtle difference between *logically* true or false versus a *test* evaluating to true or false.

To illustrate the TDD test specifications, let us take simple existential quantification, i.e., one needs to specify a test for an axiom of the form $C \sqsubseteq \exists R.D$; e.g., $\mathsf{Lion} \sqsubseteq \exists \mathsf{eats}.\mathsf{Impala}$ is to be added to the African Wildlife Ontology. Thus, a TDD test with $O \models C \sqsubseteq \exists R.D$ should return false, i.e., not be asserted nor entailed, before the ontology edit. In SPARQL-OWL, using [11]'s notation with at least one variable in an "axiom template", this can be combined into one query encapsulated in a TDD test:

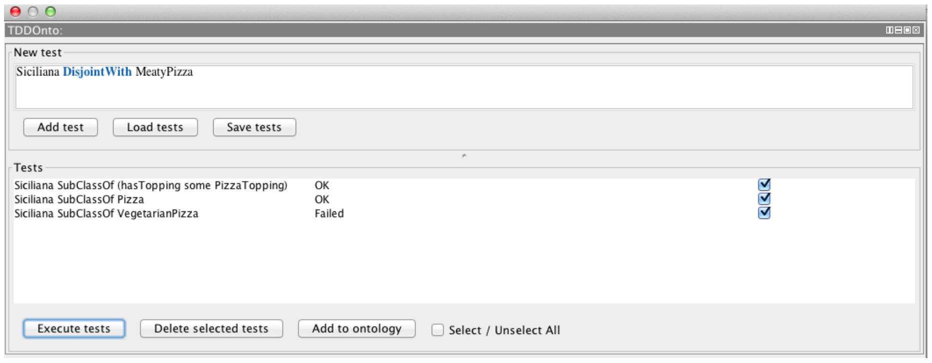
```
Require: Test $T(C \sqsubseteq \exists R.D)$                      ▷ i.e., test $T_{eq}$
 1: $\alpha \leftarrow$ SubClassOf(?x ObjectSomeValuesFrom(R D))
 2: if $C \notin \alpha$ then                                   ▷ thus, $O \not\models C \sqsubseteq \exists R.D$
 3:     return $T(C \sqsubseteq \exists R.D)$ is false
 4: else
 5:     return $T(C \sqsubseteq \exists R.D)$ is true
 6: end if
```

If the test fails, i.e., $C \notin \alpha$, then the axiom is to be added to the ontology, the TDD test run again, and if $C \in \alpha$, then the basic test cycle is completed for that axiom. The TDD test T'_{eq} with individuals can be carried out as follows, utilising De Morgan in that if the existential quantification were present and had an instance, then $C \sqcap \neg\exists R.D$ should result in an inconsistent ontology, or: in its absence, the ontology is consistent:

```
Require: Test $T(C \sqsubseteq \exists R.D)$                      ▷ i.e., test $T'_{eq}$
 1: Create a mock object, $a$
 2: Assert $(C \sqcap \neg\exists R.D)(a)$
 3: $ostate \leftarrow$ Run reasoner
 4: if $ostate ==$ consistent then                   ▷ thus, then $O \not\models C \sqsubseteq \exists R.D$
 5:     return T$(C \sqsubseteq \exists R.D)$ is false
 6: else
 7:     return T$(C \sqsubseteq \exists R.D)$ is true
 8: end if
 9: Delete $(C \sqcap \neg\exists R.D)(a)$ and $a$.
```

The TDD tests are more cumbersome for axioms that depend on others, such are property chaining, and for all object properties but transitivity and local reflexivity, only ABox-level TDD tests can be specified. Also, for some cases, testing for an inconsistent ontology is the easier route (e.g. for ($\mathsf{Func}(R)$), or checking the failure of the converse, as for asymmetry. In other tests, they are just simple classification reasoning, such as class membership, or property subsumption checking; e.g.,

```
Require: Test $T(R \sqsubseteq S)$                               ▷ i.e., test $T'_{ps}$
 1: Check $R, S \in V_{OP}$
 2: Add individuals $a, b$ to the ABox, add $R(a, b)$
 3: Run the reasoner
 4: if $O \not\models S(a, b)$ then                              ▷ thus $O \not\models R \sqsubseteq S$
 5:     return $T(R \sqsubseteq S)$ is false
```



Siciliana DisjointWith MeatyPizza

Siciliana SubClassOf (hasTopping some PizzaTopping)	OK	✓
Siciliana SubClassOf Pizza	OK	✓
Siciliana SubClassOf VegetarianPizza	Failed	✓

Fig. 1. A screenshot of the TDDOnto plugin.

```
6: else
7:     return $T(R \sqsubseteq S)$ is true
8: end if
9: Delete $R(a, b)$, and individuals $a$ and $b$
```

The complete set of TDD tests for $\mathcal{SROIQ}$ are specified in [7].

3 System design

Realising the execution of a TDD test faced the issue of choosing technologies and what to do with the trade-offs. They were not clear upfront, other than that during preliminary testing during manual specification of the TDD tests, both the DL query tab was used (where possible/applicable) and the automated reasoner to verify the intended effects. This made it clear that a tool had to be developed a technology would need to be used systematically.

Some of the options are: i) SPARQL-OWL's implementation OWL-BGP and its SPARQL SELECT queries (as in our specification of the tests) that uses a SPARQL query answering engine and HermiT v1.3.8, ii) SPARQL-DL's [18] implementation[4], specifically its ASK queries, that uses the OWL API and an OWL 2 DL reasoner, iii) using just the OWL API with one of the OWL 2 DL reasoners implementing the API. SPARQL-DL's implementation by Derivo with SPARQL ASK queries is an interesting option for having a simple mechanism for checking existence/non-existence of particular axioms, but we have omitted it as for now, since it does not entirely follow the OWL 2 entailment regimes. We have also found out that the OWL-BGP does not implement the SPARQL ASK queries. Although we extended the code to incorporate ASK queries (it was rather easy to extend it, because all necessary other components of OWL-BGP were already in place), we ultimately decided to use OWL-BGP as it

[4] http://www.derivo.de/en/resources/sparql-dl-api.html

constitutes a clearer baseline (and postpone deciding on extending the existing tools implemented by others after performing this initial analysis).

In order to support ontology engineers in performing TDD, we have implemented the Protégé plugin named TDDOnto. The plugin provides a view where the user may specify the set of tests to be run. After their execution, the status of the tests is displayed. It is also possible to add a selected axiom to the ontology (and re-run the test). Fig. 1 presents a screenshot of the TDDOnto plugin.

During implementation of the tests, it appeared that blank nodes/unnamed classes (complex class expressions) are not supported by OWL-BGP [11] when on the right-hand-side of the inclusion axiom, and neither of the query-based options works well with object properties[5]. Also, it may be the case that overall performance may be different when a different reasoner is used, as reasoners do exhibit performance differences [15]. This forced us to adjust the tool into one of implementing and testing what can be done now. From its results, tendencies emerge that are based on a solid, experimentally motivated, which then is the basis for deciding which technique likely will have the best chance of success, hence, the best candidate for extending the corresponding tool.

Thus, overall, we implemented the tests as per specification in [7], use existing tools only—OWL-BGP and OWL API with HermiT v1.3.8—for those components of the TDD test that require query answering or classification, and added a 'wrapper' for creation/deletion of mock entities, the true/false, and a basic user interface accessible as plugin to Protégé.

4 Evaluation

The aim of the evaluation is to answer the question *Which TDD approach is better: SPARQL queries with OWL-BGP, mock objects in the ABox, or using the reasoner with the OWL API?*. We focus on the performance component of these question rather than usability by a modeller. The set-up of the experiment is described first, and then the results and discussion.

4.1 Set-up of the experiments

To assess the question quantitatively, we pose the following general hypotheses:
H1: Query-based TDD is faster than ABox object-based TDD tests.
H2: Classification time of the ontology contributes the most to overall performance (time) of a TDD test.
H3: The TDD tests on OWL ontologies are faster than on OWL 2 DL ontologies.
We expect H1 to hold because once classified, one can query multiple times without having to classify the ontology again, and for some mock object-driven TDD test, the ontology should be inconsistent, which is a more cumbersome step to deal with than checking membership of a class or individual in a query

[5] it is still possible to carry out the sequence of each of the ABox test 'manually' by adding the individuals, relations, run the reasoner and check the instance classification results, in the sequence as described by the TDD tests.

answer. The reason for expecting H2 to hold is that the other operations—adding and removing entities, testing for membership—can be executed in linear time, whereas there are not many ontologies in a language that is linear or less in data complexity. For H3: one may expect a difference in performance between OWL and OWL 2 DL ontologies, because their respective complexity increased from Exptime to N2ExpTime. These general hypotheses can be refined to suit statistical tests for each hypotheses:

$H1_0$: There is no difference between query-based and mock object-based TDD tests.

$H1_a$: There is a difference, with query-based faster execution times of the tests.

and likewise for H2 and H3. This requires measuring the difference in executions time among the query-based tests and among the mock object-based test, compute average, mean, median, stdev overall and differences per TDD test. Finally, the performance is expected to depend on the ontology's content that is being revised, as reasoning time does. While we do not aim to assess the internals of the reasoner, we would like to obtain an indication whether there might be interference regarding this aspect.

As test data, we downloaded all ontologies from the TONES repository mirror at OntoHub[6], removed those that were either not in OWL (but in OBO format) or were having datatypes incompatible with OWL 2. The remaining 67 OWL ontologies were divided into 4 groups, depending on the overall number of their axioms: up to 100 (n =20), 100–1000 axioms (n =35), 1000–10,000 axioms (n =10), and over 10,000 (n =2). A second set of 20 OWL 2 DL ontologies were collected from the authors and from non-TONES OntoHub OWL ontologies that had at least either an $\mathcal{R}$ or $\mathcal{Q}$ in the DL expressiveness. This was a manual process, as there is no up-to-date ontology repository that lists such characteristics. Five had to be removed from testing due to datatype issues, leaving 15, and thus 82 in the full set of ontologies.

As OWL reasoner, we used the same reasoner that is built-in into OWL-BGP, namely HermiT 1.3.8, to ensure fair comparison. In addition, we compared the SPARQL-OWL to the OWL API and the reasoner. The tests were carried out on a Mac Book Air with 1.3 GHz Intel Core i5 CPU and 4 GB RAM.

Regarding test generation, for each axiom type (assuming the most basic form where C and D are primitive concepts), there is a fixed number of 'slots' that can be replaced with URIs. For each test, these slots were randomly filled from the set of URIs existing in the ontology, taking into account whether an URI represents a class or a property. The tests, being tested axioms that were randomly generated, are available in the online material. Each test kind was repeated 3 times and averaged to obtain more reliable results.

4.2 Results and discussion

In developing the implementation of the TDD tests and executing them, we discovered that not all the features of OWL 2 are covered by OWL-BGP, and

[6] `https://ontohub.org/repositories`

in particular, RBox axioms (e.g., `subPropertyOf`) and property characteristics were not handled. Therefore, we only present the comparative results of the tests that could be run in both settings: ABox tests and TBox SPARQL-OWL tests, and compared to the OWL API. All the experimental results are available at `https://semantic.cs.put.poznan.pl/wiki/aristoteles/doku.php`.

Let us consider first H1. On the set of all ontologies, a mock object (ABox) test took 5.191s on average, with a standard deviation (sd) of 71.491s and median of 0.014s. The SPARQL-OWL query-based (TBox) test took 6.244s on average, with sd 113.605s and median value 0.005s. Separating this into OWL and OWL 2 DL ontologies, it becomes more interesting, as summarised in Table 1: TBox with SPARQL-OWL performs, on average as well as by median, but much better with OWL 1, whereas the ABox approach is faster with the OWL 2 DL ontologies. Using the reasoner directly ("TBox(reasoner only)") shows intermediate average for OWL and the worst results for OWL 2 DL ontologies, yet their median is better for both, or: there are a few outliers that skew the results. We have computed a statistical significance t-test with $H1_0$ of identical average scores and the threshold of 5%. In case of the set of all ontologies, t=-0.322 and p=0.748, therefore we cannot reject the null hypothesis. In case of a set of OWL 1 ontologies, it has t=2.959 and p=0.003, therefore we can reject the null hypothesis and conclude that the query-based tests are significantly faster.

Table 1. Statistics for running times of TDD tests on OWL and OWL 2 ontologies.

	AVG		MEDIAN	
	OWL	OWL 2	OWL	OWL 2
ABox	2.561±25.376	19.876±172.620	0.012	0.025
TBox (SPARQL-OWL)	0.544±3.107	30.116±257.303	0.005	0.005
TBox (reasoner only)	1.672±26.254	57.722±417.574	6.8e-05	7.75e-05

Second, H2 on TDD and classification execution times. The statistics are summarised in Fig. 2 and Fig. 4, where the X axis presents the groups of the ontologies (the ranges of the minimum and maximum number of the axioms each ontology in the group has). Note that the Y axis is scaled logarithmically. In the figures, there is a box plot presenting the results (time taken to execute the TDD test) for every group of ontologies, being: the median m (horizontal line within the box); the first and third quartile (bottom and top line of the box); the lowest value above $m - 1.5 \cdot IQR$ (short horizontal line below the box), the highest value below $m + 1.5 \cdot IQR$ (short horizontal line above the box), where IQR (interquartile range) is represented with the height of the box; and outliers (points above and below of the short lines).

In the figures it is apparent that the ontology classification times are large—in fact, higher on average—in comparison to the times of running the test. The overall results for the ABox-based tests were as follows: the average ontology classification time was 15.990s (sd 128.264s) and median 0.040s, and the average test time was 5.191s (sd 71.491s) and median 0.013s. The overall results for

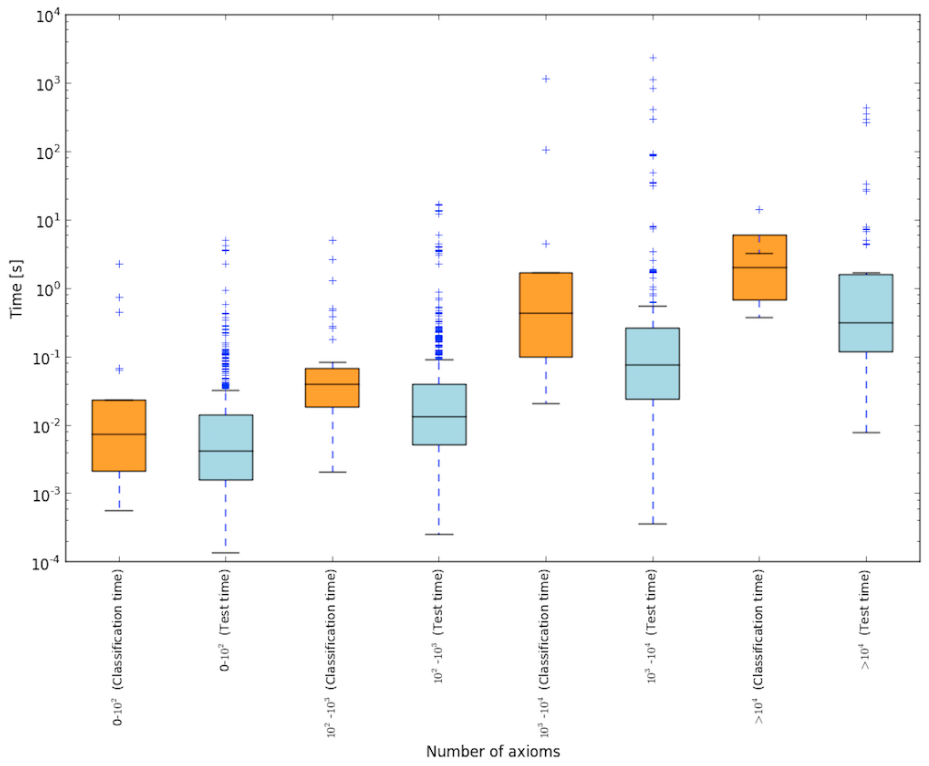


Fig. 2. Ontology classification and test computation times per ontology axiom count (ABox), for the combined set of 82 OWL 1 and OWL 2 ontologies.

OWL-BGP-based tests were, respectively, as follows: 15.954s (sd 28.267s) and median 0.040s, and 6.244s (sd 113.606s) and median 0.005s.

Hypothesis H3 on OWL vs OWL 2 differences overall already suggests $H3_a$ based on H1. More precisely, the average computing time of all three types of tests—ABox, SPARQL-OWL, and OWL API+reasoner—was 1.592s for OWL 1 ontologies and 37.361s for OWL 2 ontologies, with a standard deviation, respectively, 21.173s and 309.615s, and median values 0.005s and 0.006s, respectively. The t-test values are t=-7.425 and p=1.309e-13; thus, we can conclude that the tests on OWL ontologies were significantly faster. This is as expected purely on theoretical grounds based on the complexity results of $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$, and these results show that it also does significantly in praxis.

More results are included online at the aforementioned URL.

5 Discussion

It is, perhaps, surprising that running a TDD test is, on average, faster than the 'add, classify, and see' scenario. It does provide interesting new prospects

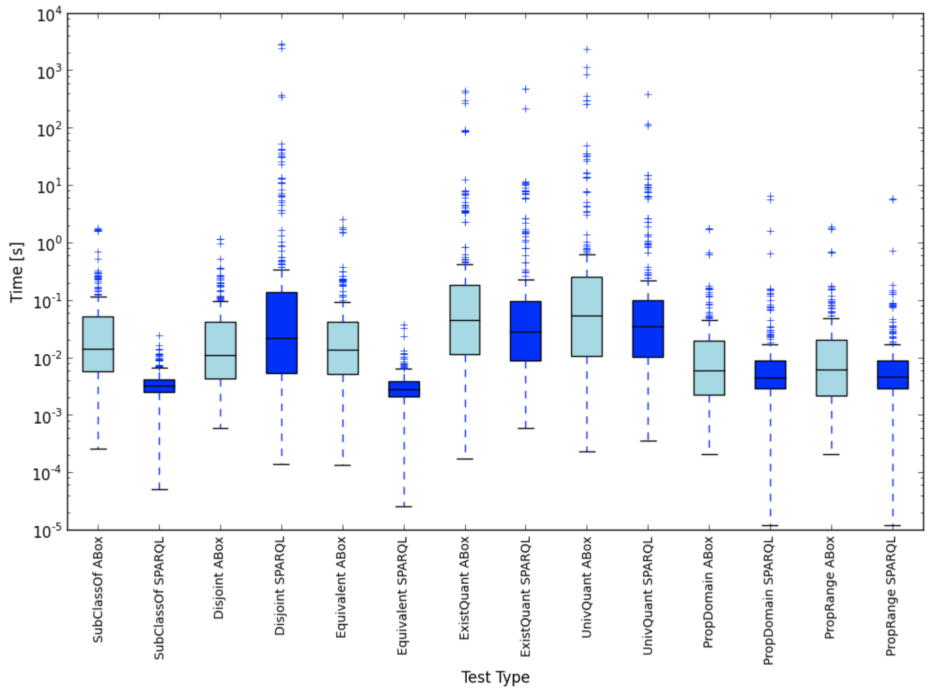


Fig. 3. Test computation times per ontology axiom count (mock objects vs. SPARQL-OWL), for the combined set of 82 ontologies.

for chunking up the reasoning tasks so that the overall process of ontology development may become a bit more efficient compared to the hurdles of dealing with long classification times each time a minor change is made. It is an avenue of future work to investigate this in more detail.

While we are not in the position to explain in detail why the performance is as it is—slower with the ABox-based approach, faster with queries and even faster by directly using the reasoner, in most cases—extending the implementation using the reasoner directly seems most promising. This, in turn, asks for further investigation with other DL-based reasoners. This could be refined even further by considering reasoners for the profiles, in the light that there is a significant difference between OWL and OWL 2 DL ontologies in TDD performance.

An observation unrelated to performance are the ontologies themselves. The main problem were the datatypes used in several ontologies. There is a mismatch between XML's data types, OWL and OWL 2's data types and datatype map, and what appears as possible option in some ontology development environments. For instance, one can select `anyType` in Protégé 4.3, but this is not supported by OWL (one ought to use `Literal`), and likewise for the datatype `ENTITY`. In a few other cases, imports were not where they were supposed to

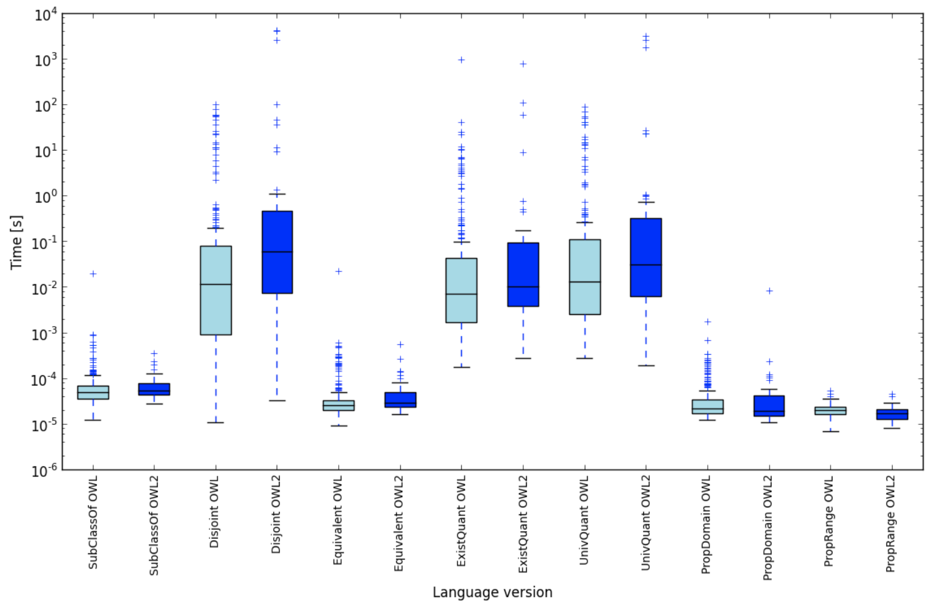


Fig. 4. TDD results with the reasoner and OWL API, separated by TONES's OWL ontologies and the OWL2 ontologies.

be (SWEET ontologies), or a SWRL rule was using a built-in atom that is not supported yet. Similar issues have been observed with the ORE test set.

Once all tests are implemented and a multi-modal interface is developed to cater for various use cases, user evaluations are planned to be conducted to evaluate whether also for ontology engineering the TDD benefits can be reaped, as observed for conceptual modelling [19] and software development [6, 12].

6 Conclusions

The paper presented TDDonto, a Protégé plugin for Test-Driven Development tests. Performance evaluation showed that TDD tests outperformed classification reasoning, and as such may aid in reducing the overall ontology authoring time. Among the two strategies—ABox-based with mock objects and TBox-based tests—the TBox tests were generally faster. Among the SPARQL-OWL (OWL-BGP) and OWL API+reasoner options for TBox TDD tests, the latter had better median performance. It was also clear from the results that TDD tests on OWL ontologies are faster overall than on OWL 2 DL ontologies.

TDD being novel for ontology engineering, there are multiple avenues for future work, including implementing the remaining tests.

Acknowledgments This research has been supported by the National Science Centre, Poland, within the grant number 2014/13/D/ST6/02076.

References

1. Beck, K.: Test-Driven Development: by example. Addison-Wesley, Boston, MA (2004)
2. Blomqvist, E., Sepour, A.S., Presutti, V.: Ontology testing – methodology and tool. In: Proc. of EKAW'12. LNAI, vol. 7603, pp. 216–226. Springer (2012)
3. Ferré, S., Rudolph, S.: Advocatus diaboli exploratory enrichment of ontologies with negative constraints. In: Proc. of EKAW'12. LNAI, vol. 7603, pp. 42–56. Springer (2012), oct 8-12, Galway, Ireland
4. Garca-Ramos, S., Otero, A., Fernández-López, M.: OntologyTest: A tool to evaluate ontologies through tests defined by the user. In: Proc. of IWANN 2009 Workshops, Part II. LNCS, vol. 5518, pp. 91–98. Springer (2009)
5. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. Proceedings of KR-2006 pp. 452–457 (2006)
6. Janzen, D.S.: Software architecture improvement through test-driven development. In: Companion to ACM SIGPLAN'05. pp. 240–241. ACM Proceedings (2005)
7. Keet, C.M., Ławrynowicz, A.: Test-driven development of ontologies (extended version). Technical Report 1512.06211 (December 2015), `http://arxiv.org/abs/1512.06211`, arxiv.org
8. Keet, C.M., Fernández-Reyes, F.C., Morales-González, A.: Representing mereotopological relations in OWL ontologies with ONTOPARTS. In: Simperl, E., et al. (eds.) Proc.of ESWC'12. LNCS, vol. 7295, pp. 240–254. Springer (2012), 29-31 May 2012, Heraklion, Crete, Greece
9. Keet, C.M., Khan, M.T., Ghidini, C.: Ontology authoring with FORZA. In: Proc. of CIKM'13. pp. 569–578. ACM proceedings (2013)
10. Kim, T., Park, C., Wu, C.: Mock object models for test driven development. In: Proc. of SERA06. IEEE Computer Society (2006)
11. Kollia, I., Glimm, B., Horrocks, I.: SPARQL Query Answering over OWL Ontologies. In: Proc, of ESWC'11. LNCS, vol. 6643, pp. 382–396. Springer (2011), 29 May-2 June, 2011, Heraklion, Crete, Greece
12. Kumar, S., Bansal, S.: Comparative study of test driven development with traditional techniques. Int. J. Soft Comp. & Eng. 3(1), 352–360 (2013)
13. Mackinnon, T., Freeman, S., Craig, P.: Extreme Programming Examined, chap. Endo-testing: unit testing with mock objects, pp. 287–301. Addison-Wesley, Boston, MA (2001)
14. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 web ontology language structural specification and functional-style syntax. W3c recommendation, W3C (27 Oct 2009), http://www.w3.org/TR/owl2-syntax/
15. Parsia, B., Matentzoglu, N., Goncalves, R., Glimm, B., Steigmiller, A.: The OWL Reasoner Evaluation (ORE) 2015 competition report. In: Proc. of SSWS'15. CEUR-WS, vol. 1457 (2015), bethlehem, USA, Oct 11, 2015.
16. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A.: Validating ontologies with OOPS! In: ten Teije, A., et al. (eds.) Proc. of EKAW'12. LNAI, vol. 7603, pp. 267–281. Springer (2012), oct 8-12, Galway, Ireland
17. Ren, Y., Parvizi, A., Mellish, C., Pan, J.Z., van Deemter, K., Stevens, R.: Towards competency question-driven ontology authoring. In: Proc. of ESWC'14. LNCS, vol. 8465, p. 752767. Springer (2014)
18. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: Proceedings of the Third International Workshop OWL: Experiences and Directions (OWLED'07). CEUR-WS (2007), 6-7 June 2007, Innsbruck, Austria

19. Tort, A., Olivé, A., Sancho, M.R.: An approach to test-driven development of conceptual schemas. Data & Knowledge Engineering 70, 1088–1111 (2011)
20. Vrandečić, D., Gangemi, A.: Unit tests for ontologies. In: OTM workshops 2006. LNCS, vol. 4278, pp. 1012–1020. Springer (2006)
21. Warrender, J.D., Lord, P.: How, What and Why to test an ontology. Technical Report 1505.04112, Newcastle University (2015), http://arxiv.org/abs/1505.04112