Reasoner Performance on Ontologies for Operations

Scott Bell[1], Jim Carciofini[2], Mark Boddy[2] and Pete Bonasso[1]
[1]*TRACLabs Inc., Houston, TX, U.S.A.*
[2]*Adventium Labs, Minneapolis, MN, U.S.A.*

Keywords: Ontology, Operations, Reasoner, Performance.

Abstract: While creating a software suite of ontology tools for operations, we encountered several reasoner performance scaling issues. This paper describes the symptoms, the diagnosis, and the mitigation strategies used.

1 INTRODUCTION

In previous work, we described a set of software tools called the PRIDE ONTOlogy Editor (PRONTOE) and a methodology that allows system operators and domain experts to build and maintain ontologies of their systems with no explicit understanding of the underlying ontology representation (S. Bell et al., 2013). Represented in the Web Ontology Language or OWL (W3C OWL Working Group, 2009), these ontologies provide the knowledge necessary for software tools that assist operators in monitoring and controlling complex and dynamic systems. Ontological reasoners are used to automatically populate object data fields, derive relations between objects to improve search of system information, and maintain consistency across the model. The initial application modeled was International Space Station (ISS) operations, and among the issues that needed to be dealt with were: incomplete information, maintaining consistency in a model continually being updated by many different people, representing the physics of an operational domain (specifically that there are physical restrictions, e.g. one thing in exactly one place at one time), and implications of other changes made. These issues were addressed using four key features of OWL: functional properties, closed enumerated classes, existential property restrictions, and Semantic Web Rule Language (SWRL) (Horrocks et al., 2004) rules.

1.1 Functional Properties

In OWL we can specify functional relations, that is relations which, given values for the first n-1 arguments, have a single result for the nth argument at any point in time. For example, a tether can only be tethered to one agent at a time, so the relation *(tethered-to ?tether ?agent)* can only have one value for *?agent*, given a value for *?tether*. Thus if a user tries to assert that another agent is using that tether, PRONTOE will flag the inconsistency. However, the reasoner does not give us the inconsistency we would like. Using Pellet's Unique Name Assumption feature, GUID functional data properties, or *DifferentIndividuals* axioms allow inconsistencies to be detected instead of permitting undesired individual equality inferences.

1.2 Closed Enumerated Classes

Commonly with command and data handling systems, a given command or a given telemetry identifier can only have a certain set of values. For example, a blower for a carbon dioxide removal system can be commanded to be enabled or disabled, powered or unpowered, and an air inlet valve position can either be open or closed. To ensure that commands and telemetry are restricted in this manner, we use OWL's closed enumeration definitions for these classes.

1.3 Existential Property Restrictions

PRONTOE uses the existence of the existential property restriction to indicate properties in the GUI. The user is not required to fill in the value, but the fact that the value is required is simply highlighted for the user to eventually fill in. The model remains consistent with incomplete information, which is the desired behavior.

1.4 SWRL Rules

We use a number of logical rules, so that a reasoner

Bell S., Carciofini J., Boddy M. and Bonasso P..
Reasoner Performance on Ontologies for Operations.
DOI: 10.5220/0005082703040311
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2014), pages 304-311
ISBN: 978-989-758-049-9

can infer additional properties on behalf of the user. These rules manage bookkeeping during operations—e.g., moving a container changes the location of all the items in the container—and for domain physics, such as the loss of fluid flow when a pump loses power. The rules are given in the form of SWRL rules (Horrocks et al., 2004), which can be used to form rules whose left hand sides (the if portion or context) and right hand sides (the then portion or implication) are OWL relations. An example of a rule in our ontology is given below:

```
<DLSafeRule>
  <Body>
    <ClassAtom>
      <Class abbreviatedIRI="base:PhysicalEntity"/>
      <Variable IRI="urn:swrl#a"/>
    </ClassAtom>
    <ClassAtom>
      <Class abbreviatedIRI="base:PhysicalEntity"/>
      <Variable IRI="urn:swrl#b"/>
    </ClassAtom>
    <ClassAtom>
      <Class IRI="#ISSlocation"/>
      <Variable IRI="urn:swrl#l"/>
    </ClassAtom>
    <ObjectPropertyAtom>
      <ObjectProperty IRI="#isAttachedTo"/>
      <Variable IRI="urn:swrl#a"/>
      <Variable IRI="urn:swrl#b"/>
    </ObjectPropertyAtom>
    <ObjectPropertyAtom>
      <ObjectProperty IRI="#hasISSlocation"/>
      <Variable IRI="urn:swrl#b"/>
      <Variable IRI="urn:swrl#l"/>
    </ObjectPropertyAtom>
  </Body>
  <Head>
    <ObjectPropertyAtom>
      <ObjectProperty IRI="#hasISSlocation"/>
      <Variable IRI="urn:swrl#a"/>
      <Variable IRI="urn:swrl#l"/>
    </ObjectPropertyAtom>
  </Head>
</DLSafeRule>
```

This rule says if a object is attached to another object, and that object has an ISS location, then the first object also has that location. Thus if a user asserts that a piece of equipment is attached to a mount and that mount has a location, a reasoner will infer the location of the equipment. Figure 1 shows a piece of equipment on the ISS called a DC to DC Converter Unit (DDCU-E_3) with a few relations defined, the important one being the *isAttachedTo* linking DDCU-E_3 to the cold plate called DDCU-CP_1. This in turn is attached to a mount (S0_DDCUmount1), and finally ends with a specific ISS location (S0B01F01MS). After running the reasoner and applying the SWRL rules as shown in Figure 2, the reasoner can infer that DDCU-E_3 is located at S0B01F01MS.

2 REASONER PERFORMANCE ISSUES

As the PRONTOE project progressed, we noticed a significant performance degradation in two OWL reasoners: Pellet (Sirin et al., 2007) and HermiT (Shearer et al., 2008)—especially when we added telemetry data to the OWL ontology. Before we added the telemetry the reasoner would run over PRONTOE in 15–30 seconds; with the telemetry added it took from 15 to 20 minutes. Our ontology without the telemetry had 442 classes and 1386 individuals; with the telemetry these increased to 502 and 1539 respectively. It seemed strange that this slight increase in data would cause such severe degradation in performance. The telemetry data used all four of the OWL features mentioned above. We originally focused on the SWRL rules, but this was quickly dismissed since removing the SWRL rules had no significant effect on performance. After some experimentation, we discovered that removing all individuals of subclasses of Enumeration made the reasoner faster. Adding members for a single subclass back (e.g. *CDRSDayNightEnumeration* members day and night) made the reasoner slow again. *CDRSDayNightEnumeration* and other subclasses of Enumeration are so-called closed enumerated classes. They are specified with the following axioms (function-style syntax):

```
EquivalentClasses(:CDRSDayNightEnumeration
                  ObjectOneOf(:night :day))
```

We change this to an open class as follows:

```
ClassAssertion(:OpenOrClosedClass :a)
ClassAssertion(:OpenOrClosedClass :b)
```

The reasoner got much faster. There was initially concern about switching the closed classes to open classes. The intention of the closed classes was to restrict the operator to choosing among the specified individuals in a given class. However, OWL does not make any assumptions about the uniqueness of individuals. Thus, closed classes can lead to unexpected inference. For example, if you have a closed class with members (A, B, C) and you assert that D is also a member of the class, you will not get an inconsistency. You will instead infer that D is equivalent to A, B or C. We could get along with open classes by having PRONTOE do its own checking for closedness, but we decided to more thoroughly characterize the sources of the slow reasoner speeds.

2.1 Characterization

We created a scalable synthetic ontology to investigate the performance issue in more depth. A Python pro-

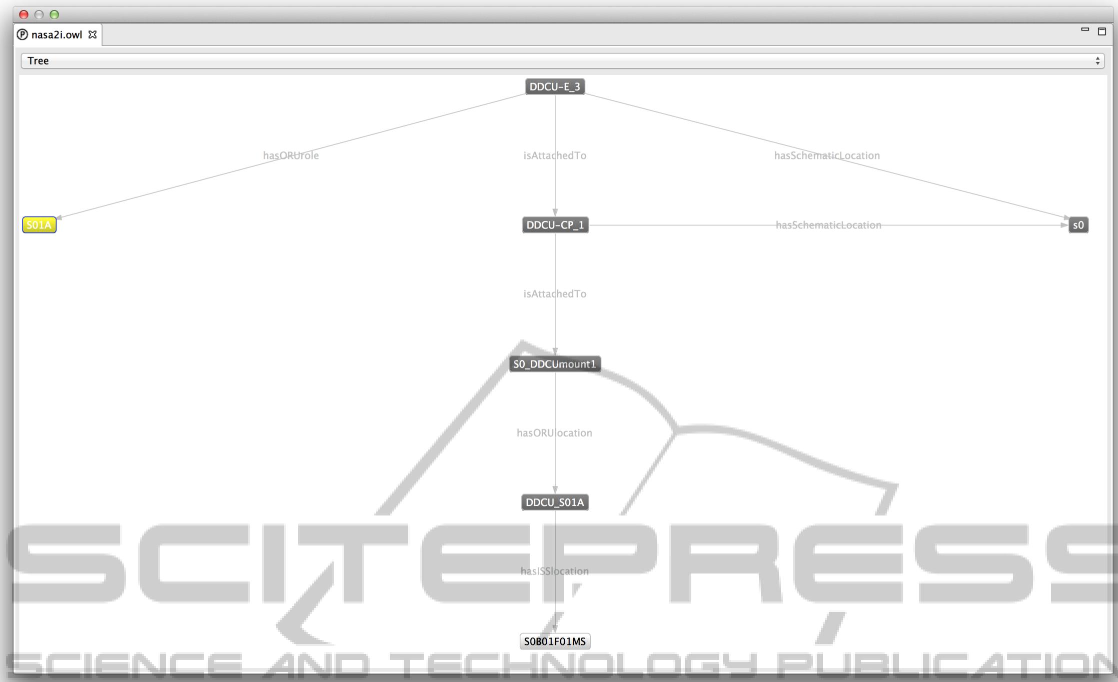


Figure 1: A piece of equipment modeled in PRONTOE before SWRL rules are applied.

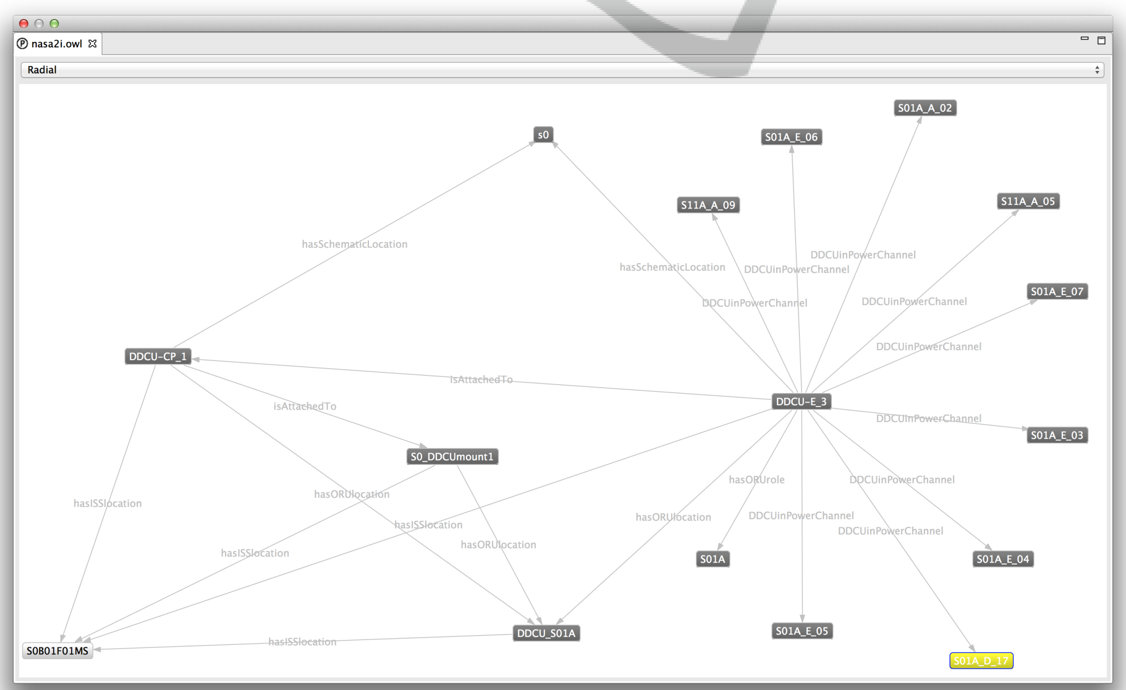


Figure 2: After reasoning, the equipment can be precisely located.

gram was written to create instances of this synthetic model with the following parameters:

I Number of individuals.

S Number of subclasses.

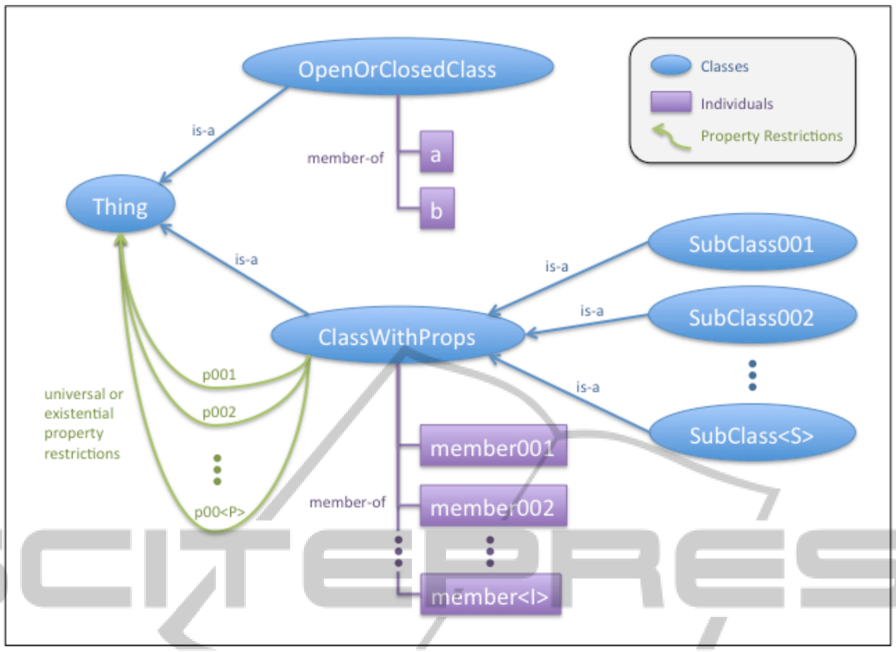


Figure 3: A diagram showing the overall structure of the scalable synthetic ontology.

P Number of property restrictions.

T Type of property restriction, existential or universal.

C Additional closed or open class with two members.

Figure 3 shows the overall structure of the scalable synthetic ontology. The ontology fragments below are in functional-style syntax for readability. All generated models have these declarations:

```
Declaration(Class(:ClassWithProps))
Declaration(Class(:OpenOrClosedClass))
Declaration(NamedIndividual(:a))
Declaration(NamedIndividual(:b))
```

A number of individuals in class ClassWithProps are generated as indicated by parameter *I*:

```
Declaration(Class(:ClassWithProps))
Declaration(Class(:OpenOrClosedClass))
Declaration(NamedIndividual(:a))
Declaration(NamedIndividual(:b))
Declaration(NamedIndividual(:member000))
ClassAssertion(:ClassWithProps :member000)
Declaration(NamedIndividual(:member001))
ClassAssertion(:ClassWithProps :member001)
Declaration(NamedIndividual(:member002))
ClassAssertion(:ClassWithProps :member002)
...
Declaration(NamedIndividuael(:member<I>))
ClassAssertion(:ClassWithProps :member<I>)
```

A number of subclasses of ClassWithProps are generated per parameter *S*:

```
Declaration(Class(:SubClass000))
SubClassOf(:SubClass000 :ClassWithProps)
Declaration(Class(:SubClass001))
SubClassOf(:SubClass001 :ClassWithProps)
Declaration(Class(:SubClass002))
SubClassOf(:SubClass002 :ClassWithProps)
...
Declaration(Class(:SubClass<S>))
SubClassOf(:SubClass<S> :ClassWithProps)
```

A set of *P* properties are generated with range and domain ClassWithProps:

```
Declaration(ObjectProperty(:p000))
ObjectPropertyDomain(:p000 :ClassWithProps)
ObjectPropertyRange(:p000 :ClassWithProps)
Declaration(ObjectProperty(:p001))
ObjectPropertyDomain(:p001 :ClassWithProps)
ObjectPropertyRange(:p001 :ClassWithProps)
Declaration(ObjectProperty(:p002))
ObjectPropertyDomain(:p002 :ClassWithProps)
ObjectPropertyRange(:p002 :ClassWithProps)
...
Declaration(ObjectProperty(:p<P>))
ObjectPropertyDomain(:p<P> :ClassWithProps)
ObjectPropertyRange(:p<P> :ClassWithProps)
```

When *T*=existential, the following existential property restrictions are defined:

```
SubClassOf(:ClassWithProps ObjectSomeValuesFrom(:p000 :Thing))
SubClassOf(:ClassWithProps ObjectSomeValuesFrom(:p001 :Thing))
SubClassOf(:ClassWithProps ObjectSomeValuesFrom(:p002 :Thing))
...
SubClassOf(:ClassWithProps ObjectSomeValuesFrom(:p<P> :Thing))
```

When *T*=universal, the following universal property restrictions are defined:

```
SubClassOf(:ClassWithProps ObjectAllValuesFrom(:p000 :Thing))
SubClassOf(:ClassWithProps ObjectAllValuesFrom(:p001 :Thing))
```

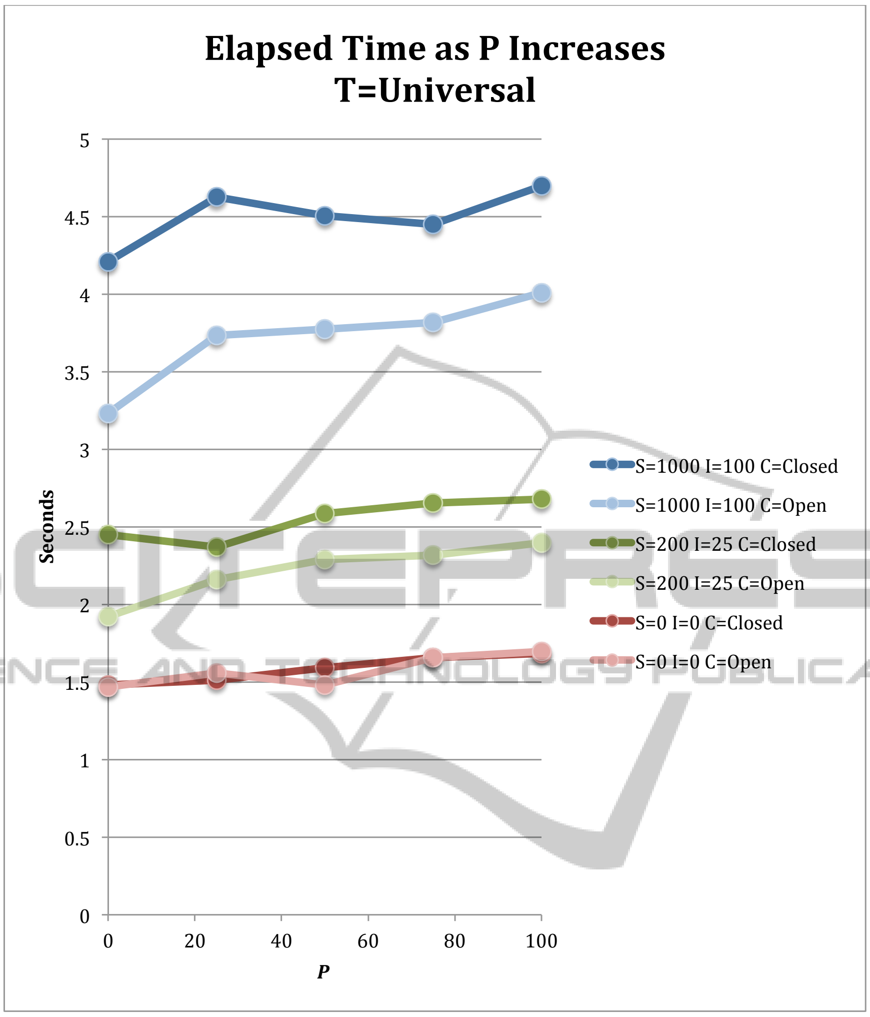


Figure 4: No evident performance problem for the size problems we benchmarked.

```
SubClassOf(:ClassWithProps ObjectAllValuesFrom(:p002 :Thing))
...
SubClassOf(:ClassWithProps ObjectAllValuesFrom(:p<P> :Thing))
```

When *C*=closed, OpenOrClosedClass is defined as a closed class:

```
EquivalentClasses(:OpenOrClosedClass ObjectOneOf(:b :a))
```

When *C*=open, OpenOrClosedClass is defined as an open class:

```
ClassAssertion(:OpenOrClosedClass :a)
ClassAssertion(:Thing :a)
ClassAssertion(:OpenOrClosedClass :b)
ClassAssertion(:Thing :b)
```

We gathered data on runtimes for the Pellet reasoner on ontologies generated for the cross product of the following parameter values:

I Number of individuals in (0, 25, 50, 75, 100)

S Number of subclasses in (0, 200, 400, 600, 800, 1000)

P Number of property restrictions in (0, 25, 50, 75, 100)

T Type of property restriction in (existential, universal)

C Additional class in (closed, open)

The machine used to gather these runtimes is shown in Table 1. We also recorded runtimes for a limited number of instances with larger parameter values to use in the regression analysis described below. Figure 4 shows that when using universal property restrictions, there is no evident performance problem for the size problems we benchmarked. The graph shows only a

Table 1: Test machine characteristics.

CPU	Intel Xenon 2.83GHz
Number of CPUs	2
Memory	8GB

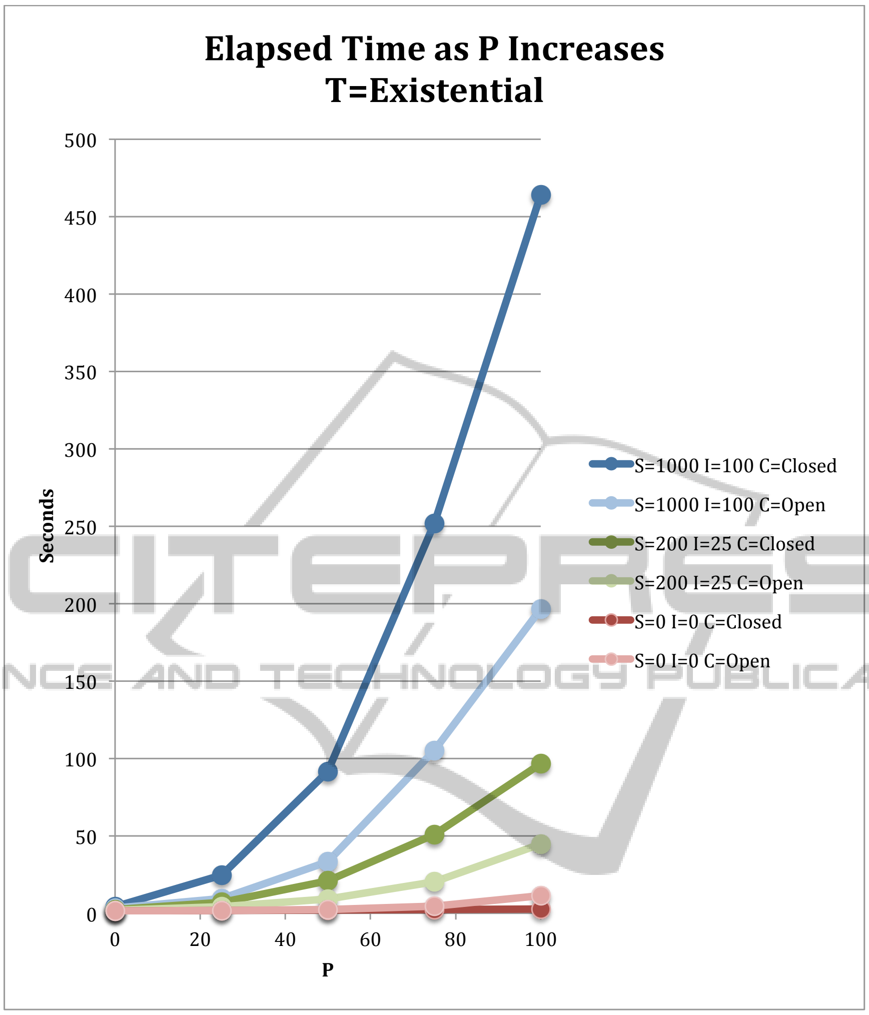


Figure 5: Runtimes increase non-linearly with problem size.

subset of the benchmarks run, but all runs completed in less than 5 seconds.

For the remainder of this discussion we will focus on cases with existential property restrictions (T=Existential). In those cases runtimes increase non-linearly with problem size. The dominant parameter appears to be P. Figure 5 shows this behavior for several slices of the data. Figure 6 and Figure 7 show the linear regression analysis on the data and found SP^2 and IP^2 to be significant parameters. The dominant parameter is IP^2, but SP^2 is significant. Figure 8 shows more detail on how runtime increases with I and S increasing and P and S increasing. All use the same scale for run time.

3 RELATED WORK

There have been other examinations of analyzing performance of OWL reasoners. Dentler et al. (2011) compared a variety of reasoners on the SNOMED CT ontology, but this was done using OWL 2 EL. Riboni and Bettini (2011) proposed and architecture for modeling human activities with ontologies. Kang et al. (2014) and Kang et al. (2012) were able to predict reasoner performance based on metrics of the ontology. Our work is distinct in that we studied the performance of specific OWL semantics designed to assist the end user, i.e., a domain expert rather than an ontologist.

4 CONCLUSIONS

The primary source of the performance problem is the existential property restrictions. While they are semantically correct, they provide no inferences of use. Removing the existential property restrictions resulted in faster reasoner times and much better scaling behav-

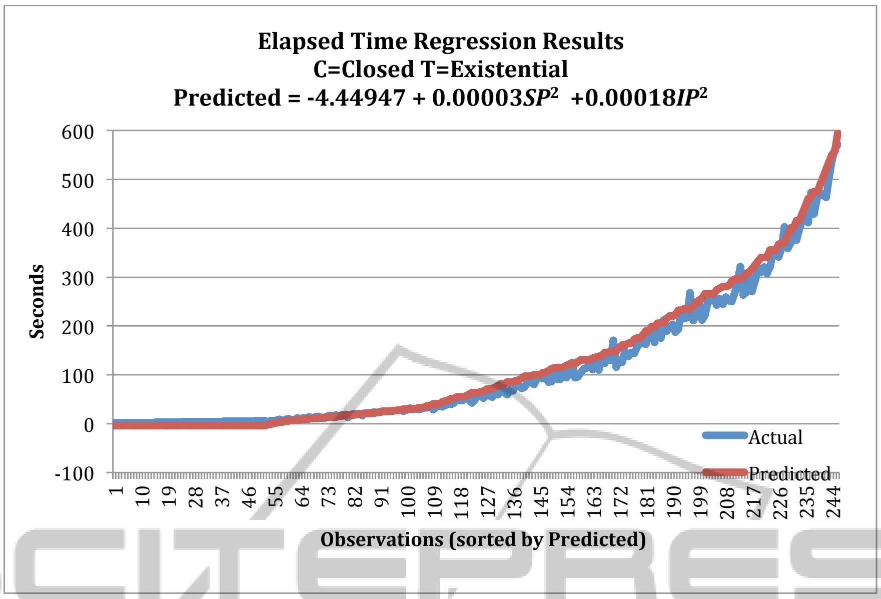


Figure 6: A linear regression with closed enumerated classes.

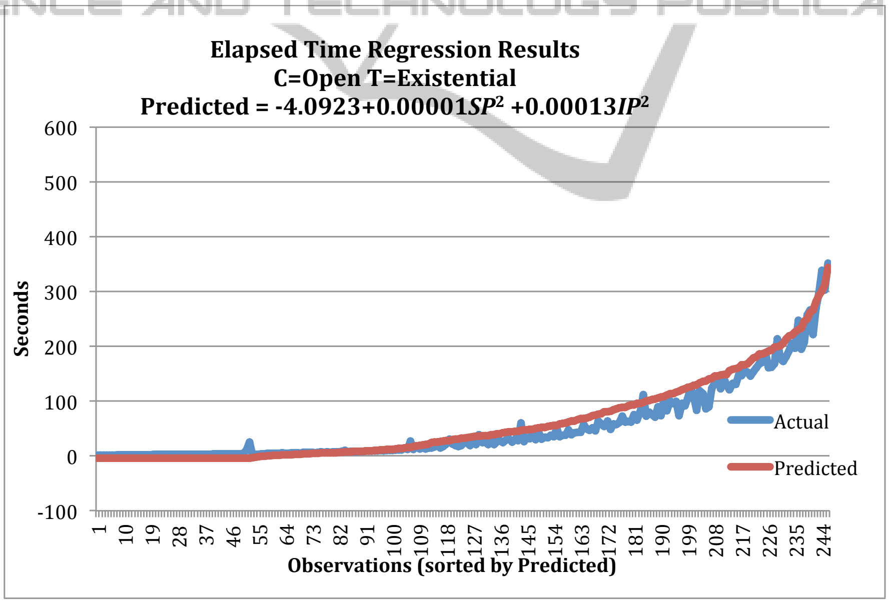
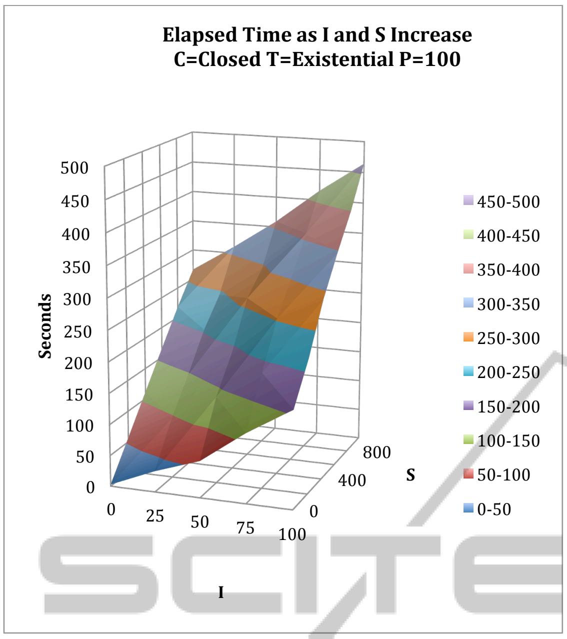


Figure 7: A linear regression with open classes.

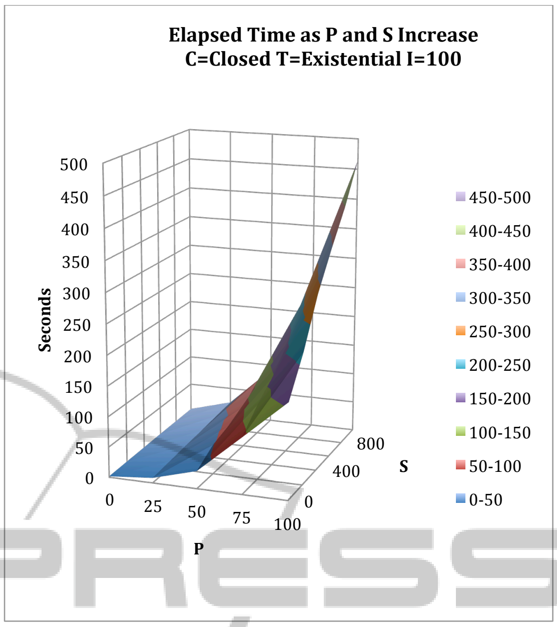
ior. PRONTOE can use the existence of the universal property restrictions to guide the GUI in eliciting properties from the user instead of the existential property restriction. Enumerated closed classes are being used for their original intended purpose: restricting the user to a set of values. With existential properties removed, we have no performance issues.

ACKNOWLEDGEMENTS

This work is funded by a NASA Small Business Innovation Research (SBIR) grant. The authors grateful to Dr. Jeremy Frank of NASA Ames Research Center for his help with this project.



(a)



(b)

Figure 8: (a) When increasing the *I* and *S* parameters and using closed enumerated classes, time to finish reasoning increases dramatically. (b) *P* and *S* also affect the result.

REFERENCES

Dentler, K., Cornet, R., ten Teije, A., and de Keizer, N. (2011). Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web*, 2(2):71–87.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3c member submission, World Wide Web Consortium.

Kang, Y.-B., Li, Y.-F., and Krishnaswamy, S. (2012). Predicting reasoning performance using ontology metrics. In *The Semantic Web–ISWC 2012*, pages 198–214. Springer.

Kang, Y.-B., Pan, J. Z., Krishnaswamy, S., Sawangphol, W., and Li, Y.-F. (2014). How long will it take? accurate prediction of ontology reasoning performance.

Riboni, D. and Bettini, C. (2011). OWL 2 modeling and reasoning with complex human activities. *Pervasive and Mobile Computing*, 7(3):379–395.

S. Bell, R. B., Boddy, M., Kortenkamp, D., and Schreckenghost, D. (2013). PRONTOE: A case study for developing ontologies for operations. In *International Conference on Knowledge Engineering and Ontology Development*, Vilamoura, Portugal.

Shearer, R., Motik, B., and Horrocks, I. (2008). HermiT: A highly-efficient OWL reasoner. In Ruttenberg, A., Sattler, U., and Dolbear, C., editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, Karlsruhe, Germany.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53.

W3C OWL Working Group (27 October 2009). *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. http://www.w3.org/TR/owl2-overview.