Instance Data Evaluation for Semantic Web-Based Knowledge Management Systems

Jiao Tao, Li Ding, Jie Bao, and Deborah L. McGuinness
Tetherless World Constellation, Rensselaer Polytechnic Institute, Troy, NY, USA
{taoj2, dingl, baojie, dlm}@cs.rpi.edu

Abstract

As semantic web technologies are increasingly used to empower knowledge management systems (KMSs), there is a growing need for mechanisms and automated tools for checking content generated by semantic-web tools. The content in a KMS includes both the knowledge management (KM) schema and the data contained within. KM schemas can be viewed as ontologies and the data contained within can be viewed as instance data. Thus we can apply semantic web ontology and instance data processing techniques and tools in KM settings.

There are many semantic web tools aimed at ontology evaluation, however there is little, if any, research focusing on instance data evaluation. Although instance data evaluation has many issues in common with ontology evaluation, there are some issues that are either more prominent in or unique to instance data evaluation. Instance data often accounts for orders of magnitude more data than ontology data in organization intranets, thus our work focuses on evaluation techniques that help users of KMSs to determine when certain instance data is ready for use.

We present our work on semantic web instance data evaluation for KMSs. We define the instance data evaluation research problem and design a general evaluation process GEP. We identify three categories of issues that may occur in instance data: syntax errors, logical inconsistencies, and potential issues. For each category of issues, we provide illustrative examples, describe the symptoms, analyze the causes, and present our detection solution. We implement our design in TW OIE which is an online instance data evaluation service. We perform experiments that show that the TW OIE is more comprehensive than most existing online semantic web data evaluators.

1. Introduction

Knowledge Management Systems have been widely used in various organizations to help users create, capture, store, share, and retrieve information. Emerging semantic web technologies [3], which combine the power of conventional semantic technologies and the fast growing World Wide Web, are being used to create next generation KMSs [6]. The Semantic Web offers a stack of knowledge representation languages, such as RDF[1], RDFS[2], and OWL [3], at different levels of expressivity; and associated computing tools that can be used to build collaborative online KMSs. The benefits have been demonstrated by various research efforts such as the On-to-Knowledge Project [9] and the SkiM system of the Swiss life insurance group [5].

Semantic web based KMSs leverage the Web to publish and share knowledge collaboratively. They organize information into two categories: ontologies (i.e. descriptions of classes and properties) and instance data (i.e. instantiations of classes). For example, the TW wine agent [21] , a simple semantic web based KMS, stores wine and food classifications as well as wine recommendation knowledge. The agent can provide aggregated recommendations with explanations. A typical scenario works as follows: a customer Joe wants a wine recommendation for his lobster meal. He consults the TW wine agent and finds the answer "a lobster meal pairs well with dry white varieties and full-bodied wines match especially well." This recommendation may come from the food/wine parings which are encoded in the wine ontology[4], or recommendations from other users. He can also contribute his wine recommendations for certain meals to the TW wine agent. In this scenario, wine recommendations from different sources are aggregated, and it is important to ensure each wine recommendation is compliant with the restrictions encoded in the wine ontology. This instance-ontology-conformance problem does not just exist in

[1] http://www.w3.org/TR/rdf-concepts/
[2] http://www.w3.org/TR/rdf-schema/
[3] http://www.w3.org/TR/owl-ref/
[4] www.w3.org/TR/owl-guide/wine

above scenario; it exists in any semantic web based KMS that is expected to conform to the ontology.

Our earlier survey on online semantic web data [7] reported the following observations: (i) instance data is usually published separately in many online RDF documents by different users, and these RDF documents are typically published after the publication of the corresponding ontologies; (ii) most RDF documents (>99%) contain only instance data; (iii) many ontologies, especially popular ones, are typically carefully designed and free of errors. These observations indicate that, on the semantic web, instance statements dwarf ontology (class and property) statements. Further, instance data is more error-prone since often it is generated by different authors at differing times, potentially leading to semantic mismatches.

This work investigates a novel research problem – semantic web instance data evaluation in KM environments, which complements the work on ontology evaluation in terms of evaluation scope: the latter focuses on consistency and completeness of ontologies; but the former focuses on checking the semantic expectation mismatches between instance data and ontologies, i.e. whether the instance data meets the (explicit and implicit) expectations of the corresponding ontologies. Consider the following example mismatch: in the wine ontology, two subclasses of wine:Wine, namely wine:EarlyHarvest and wine:LaterHarvest, are defined to be disjoint. Although the wine ontology is consistently defined, users may still publish an RDF document introducing inconsistent instance data using the wine ontology, e.g. a bottle of wine is typed as an instance of wine:EarlyHarvest and wine:LateHarvest. The same wine ontology also contains the restriction that, all wines have at least one maker. Users may publish instance data on wines that do not include a maker.

While some tools can find inconsistent instance data such as above wine instances of two disjoint classes, few tools identify problems such as missing maker values in previous example. Our work is aimed to be evaluate instance data suitability for application use and thus. We investigate not only logical inconsistency issues but also some potential issues that may cause application processing challenges. Besides the above missing value for a required property problem (maker in the example), we have also identified a number of potential issues related to data sharing applications. For instance, when describing a wine instance W, a user may mistakenly assert that the maker of W is wine:Red using the property wine:hasMaker. In the wine ontology, wine:Red is known to have type wine:WineColor. Given the description for W, a reasoner, for example, Pellet, which is a Java based OWL DL reasoner supporting species validation, consistency checking and taxonomy classification, may infer that the individual, wine:Red also has type wine:Winery. Since the two classes, wine:WineColor and wine:Winery, neither have rdfs:subClassOf nor owl:disjointWith relation, no logical inconsistency will be reported by Pellet. Our work aims to include these potential issues that are not reported by conventional reasoners such as OWL DL reasoners. We then report potential issues as warning messages, so that users can notice the mismatches and take appropriate actions.

The rest of this paper is organized as follows: section two reviews related work on knowledge source evaluation and data quality control, in particular, semantic web data evaluation; section three defines semantic web instance data evaluation as a research problem and outlines a generic solution; section four elaborates on the issues, especially potential issues, found in instance data; section five discusses the highlights of our implementation, TW OIE, and evaluates it by comparing it with other evaluation tools; section six summarizes our contributions and identifies future work.

2. Related work

Knowledge source evaluation and quality control is a critical issue in KMSs. In [16] the authors show the value of data quality and highlight its importance in traditional enterprise KMSs, then present a methodology to improve data quality, from technical and managerial perspectives. Our work addresses the knowledge sources quality issue of KMSs from a technical perspective and focuses on the semantic web setting. In [24] the authors present their work on the evaluation of a chest radiology semantic data model which represents the chest radiology findings from textual documents and x-ray reports. However this evaluation is conducted from the aspect of similarity between the semantic model and original documents, and does not address the issue of conformance between the abstract model/schema and actual data. Our work on knowledge sources evaluation uses a semantic web data evaluation approach which aims at checking whether the semantic expectations encoded in ontologies are really embodied in the instance data.

There is a long history of semantic web data evaluation including both theoretical and practical approaches. However, most work focuses on ontology evaluation emphasizing syntax validation and logical consistency analysis.

The research on semantic web ontology evaluation has been conducted in multiple directions. One direction is syntax validation. The W3C RDF validation service [5] is an online RDF parsing and validation service. It checks whether an RDF document conforms to RDF/XML syntax [6]. The validation is backed by the ARP parser of Jena [17]. This tool contributes a critical step in instance data evaluation - parsing RDF data to RDF triples.

Another direction is ontology diagnosis. The merging and diagnosis of large knowledge bases has been explored in [18]. The computational environment, Chimaera, has a number of tests that includes not only logical consistency checks but also checks for issues such as cycles in ontologies and style issues such as classes with a single subclass (which is usually indicative of an incomplete subclass hierarchy). Chimaera focuses more on ontologies rather than instance data and the released version does not include a full integration with a full expressivity reasoner, thus the widely used version does not provide full logical consistency checking. (An internal version uses a full first order logic reasoner to check more logical issues).

The third direction is semantic constraint validation. After the release of RDFS and OWL, many tools were proposed to check semantic constraints encoded in OWL and RDFS class descriptions. Early semantic web data evaluation work extends syntax validation with some simple semantic evaluation of ontology structure, such as detecting cycles in class taxonomy and domain/range mismatch issues. This line of work can be used to perform initial instance data evaluation. The following work is notable:

- The ICS-FORTH validating RDF parser (VRP)[7] parses RDF triples from RDF/XML documents and then evaluates the triples using the semantic constraints, such as class hierarchy loops, derived from RDFS inferences.
- In [4], a tool ODEval[8] was designed to detect some of the three categories of errors, namely inconsistency, incompleteness and redundancy, in concept taxonomies represented in RDFS, DAML and OWL.
- The BBN validator [9] leveraged the DAML validator [10] to use limited reasoning to check OWL markup for problems beyond simple syntax errors.
- The work in [10] provides an axiomatic accounting for RDFS and DAML+OIL reasoning. When encoded in a reasoner that provides declarative encodings of its reasoning, Inference Web [19] can be used to explain logical issues.

More recent work on semantic constraints validation has focused on OWL logical consistency issues (i.e. unsatisfiable classes) and OWL species (i.e. OWL-Lite, OWL-DL, and OWL-Full) identification related issues;

- While all OWL reasoners can detect inconsistencies, not all can pinpoint the causes of inconsistencies, which is typically needed in explaining evaluation results. Pellet [11] offers OWL reasoner-based approaches [22,15] to detect unsatisfiable concepts, inconsistent ontologies and a visualization of the evaluation results using an ontology editor Swoop. In [25], more recent work on debugging OWL-DL ontologies was covered. In [23,14], debugging consistency in ontology evolution was investigated.
- The WonderWeb OWL ontology validator [12] identifies the OWL species of a given ontology. In [2], the authors further analyzed the causes of OWL-Full vs. OWL-DL classifications. Two main causes are syntax errors and unintended mistakes in OWL usage, such as missing type or incorrect namespace. Their OWL ontology patcher[13] may even suggest strategies to fix the detected issues.

The fourth direction related to identifying best practice. However such work focused on ontologies rather than instance data. In [12, 11], three categories of issues in concept taxonomies were classified: inconsistency, incompleteness and redundancy. These issues are further refined to eight sub-categories, including logical inconsistency, cycles in class hierarchies, and redundant individual type.

- In [1], a symptom ontology is proposed to expose evaluation results for an OWL ontology consistency checking tool, ConsVISor [14]. The symptom ontology reports not only syntactic and semantic issues, but also some potential issues in concept-taxonomy.
- The BioPAX validator[15] performs specific best practices validation. Before that, it also checks if the given input is a valid OWL document by performing OWL validation.

[5] http://www.w3.org/RDF/Validator/
[6] http://www.w3.org/TR/rdf-syntax-grammar/
[7] http://139.91.183.30:9090/RDF/VRP/
[8] http://minsky.dia.fi.upm.es:8080/odeval/ODEval.html
[9] http://projects.semwebcentral.org/projects/vowlidator/
[10] http://www.daml.org/validator/
[11] http://pellet.owldl.com/
[12] http://www.mygrid.org.uk/OWL/Validator
[13] http://www.mygrid.org.uk/OWL/Patcher
[14] http://68.162.250.6:8080/consvisor/
[15] http://biopaxwiki.org/cgi-bin/moin.cgi/tentative_validator

3. Semantic web instance data evaluation

In this section, we define ontologies, instance data, and describe the relation between them. We then define the instance data evaluation and present the *g*eneric instance data *e*valuation *p*rocess GEP.

3.1. Ontology and instance data

An ontology is an explicit specification of a conceptualization [13]. In the semantic web, an ontology is used to describe "(i) Classes (general things) in the many domains of interest, (ii) The relationships that can exist among things, (iii) The properties (or attributes) those things may have"[16].

RDFS and OWL are the two primary ontology languages on the semantic web. RDFS allows users to define a concept taxonomy and some simple relations; and OWL offers additional expressive power that allows descriptions of relations between classes, (e.g., equality, disjointness), number restrictions (e.g., exactly one value), and other descriptions including enumerated class descriptions and property relations. RDFS and OWL are based on the RDF data model which uses URI references to identify resources and represents the information in the form of subject-predicate-object triples.

Definition 1. Semantic web ontology. A semantic web ontology refers to a set of triples that is used to describe classes, properties and their relations. Using recommended ontology languages, a class or property is an RDF resource that has been explicitly defined or referenced as a general thing or attribute. For example, Wine is a PotableLiquid that is made of grapes and has a maker.

Definition 2. Semantic web instance data. Semantic web instance data refers to a set of triples that is used to describe individuals. Using recommended ontology languages, an individual is an RDF resource that has been explicitly described as an instance of a class. For example, wine:Red is an instance of wine:WineColor.

Ontology and instance data play different roles in applications. Ontology is used to describe application domain vocabularies and their interrelationships while instance data is used to describe the "ground level" data particular to the application.

Two extreme cases are notable: (i) given a set of RDF triples, an RDF resource may be defined as both a class and an individual; (ii) given a set of RDF triples, we don't have enough information to tell if an RDF resource is a class or individual. The first case can be deemed as an OWL Full example, where the resource can be both an individual and a class. In the second case, we simply have to gather more information concerning the status of the resource.

[16] http://www.w3.org/TR/webont-req/#onto-def

3.2. Instance data evaluation

While both ontology and instance evaluation processes inspect the syntax issues and semantic issues of the data being evaluated, instance data evaluation focuses on issues caused by instance data and ontology evaluation focuses on the ontology side. Before proceeding, we first introduce the notion of a dataset and its notations. A dataset refers to a set of triples. Its notations are as follows:

- D: the instance data to be evaluated
- O: all referenced ontologies
- O': the set of consistent referenced ontologies
- DC: the deductive closure of D and O'

Definition 3. Instance data evaluation. Instance data evaluation refers to a process that inspects some input instance data D and its referenced ontologies $O = \{O_1, O_2, \ldots\}$. The referenced ontologies are determined by a function ref (D) that recursively loads ontologies that contribute definitions to the terms used in D. This process reports only issues caused by D and ignores the issues that would arise from the ontology in the absence of the instance data.

Intuitively, instance data evaluation checks the quality of instance data with respect to its referenced ontologies. Since ontologies are typically used to describe the relations among different groups of RDF resources and their common properties, they convey certain expectations on the corresponding instances. For example, in the wine ontology, the rdfs:range of property wine:hasColor is wine:WineColor which indicates the expectation that the color type of a wine instance should be wine:WineColor or a subclass of it. Using this expectation, we may detect mistakes in instance data such as using wine:Sweet as the value of wine:hasColor.

3.3. Generic evaluation process

We approach the instance data evaluation problem with a generic evaluation process GEP. Given some instance data as input, GEP will first load and parse the input, then load the referenced ontologies, and evaluate the input, then output a list of evaluation reports including syntactic issues, semantic inconsistency issues, and optional potential issues. We now describe the GEP evaluation steps as follows.

Step 1: Load instance data D. Unless instance data D is supplied as a text string, GEP needs to load D from either the web or a local file system. Any loading failure will be reported as fatal error and cause termination of GEP.

Step 2: Parse instance data D into RDF triples. D is loaded in text format; thus, GEP uses an appropriate RDF parser to obtain RDF triples. Since parsing issues may not be recoverable, this step reports errors as well as warnings. GEP only proceeds when some RDF triples can be parsed.

Step 3: Load referenced ontologies O = {O_1, O_2, …}, where each O_i is an ontology on the Web directly (using owl:import) or indirectly(using namespaces) referenced by RDF triples in D. By loading O, semantic web data users may get additional knowledge, including expected constraints about D. GEP will try to load all referenced ontologies and report issues when it cannot load any referenced ontology. GEP will proceed even if no referenced ontologies can be loaded.

Step 4: Inspect logical inconsistencies. GEP first inspects logical inconsistencies of each O_i in O, then drops and reports the inconsistent ones. Then GEP will try to merge the consistent referenced ontologies into one, i.e. O'. If O' is consistent, GEP will further inspect if D and O' together are consistent; otherwise, GEP will terminate.

Step 5: Inspect potential issues in D. GEP first computes the deductive closure of D, i.e. DC = INF(D,O'), which includes all original triples in D and O', and all inferred sub-class among classes and sub-property relations among properties after performing OWL and/or RDFS inference. It is used to (i)materialize the class hierarchy built using ontology language constructs and (ii)eliminate recursive queries on class and property hierarchies. Then GEP inspects different potential issues using corresponding SPARQL query solutions.

4. Issues in instance data

Issues in instance data can be classified into three categories: syntax errors, logical inconsistencies, and potential issues. The first two categories are well defined by RDF, RDFS, and OWL specifications and can be inspected by existing tools; therefore we will briefly illustrate examples of these issues. For the third category, the potential issues, we will detail the symptom and causes of the frequently encountered issues. We assume that all referenced ontologies are encoded using OWL-DL, and we will use the wine ontology and instance data to compose examples.

4.1. Syntax errors

Semantic web data may be serialized using certain syntaxes, such as RDF/XML, N3, N-Triples, Turtle, and RDFa. GEP should be able to choose an appropriate RDF parser for the input instance data, and report the syntax errors encountered. Figure 1 shows an example text fragment encoded with RDF/XML syntax. GEP may use the W3C RDF validator to report syntax issues: "'/' is missing in wine:hasMaker on line4".

```
1. <wine:Zinfandel rdf:about="#W3">
2.   <wine:hasMaker>
3.     <wine:Winery  rdf:about="#Elyse" />
4.   <wine:hasMaker>
5. </wine:Zinfandel>
 (missing "/" in the last markup on line 4)
```

Figure 1. Syntax error example

4.2. Logical inconsistencies

Logical inconsistencies depend on the choice of ontology language. They can be automatically detected by an appropriate reasoner. As we assume all referenced ontologies are in OWL-DL, we may use any OWL DL reasoner, such as Pellet, to detect OWL-DL inconsistencies. In Figure 2, individual W is described to have type EarlyHarvest and LateHarvest; however these two types have an owl:disjointWith relation in the wine ontology. Logical inconsistency occurs here because the description of W does not conform to the wine ontology, i.e. an individual should not belong to two or more disjoint classes. To resolve this issue, we can either delete W's membership in EarlyHarvest (or LateHarvest), or revise the wine ontology by deleting the disjoint relation between EarlyHarvest and LateHarvest when it is safe and reasonable to do so.

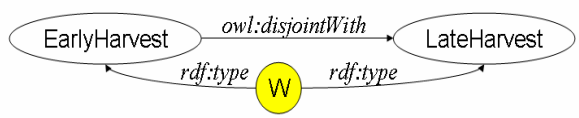

Figure 2. Logical inconsistency example

4.3. Potential issues

When checking the issues in data, OWL-DL reasoners use the open world assumption [8] and assume that missing information will be added later or will be available somewhere else. Therefore they ignore the potential issues because they do not know whether the incompleteness is caused by users'

mistakes or because the information is temporarily incomplete. As data consumers may impose different constraints on incoming instance data, there is no common agreement on whether a potential issue is harmful or not. In this work, we collect potential issues from frequently encountered practices in instance data that can not be detected by logical consistency checking, provide corresponding repair instructions, and leave the end-users the option of further inspection.

We focus on potential issues caused by instance data, and identify three sub-categories: (i) potential issues related to an individual type, (ii) potential issues related to a property value, and (iii) other application-specific potential issues. Note that our current list of issues is driven by practical application needs although it is not claimed to be exhaustive.

Some symptoms of potential issue can be captured by a conjunctive combination of the presence and absence of certain triple patterns in certain dataset. The triple patterns are described by SPARQL [17] query syntax. Our instance data evaluation uses SPARQL as the query language because SPARQL is the W3C recommendation query language for RDF and it fits very well with our problem of search for potential issue triple patterns. We use the following variables in triple patterns:

- ?c, ? c_1, ? c_2,…: a class
- ?p: a property
- ?r: an owl restriction
- ?n: integer value for owl cardinality
- ?i: an individual
- ?x: an RDF resource

4.3.1 Issues related to the types of individuals

PI-1. Unexpected individual type (UIT)

Symptom: Given an individual i, and all of its types {t_1, t_2, …} in DC, including the declared types and inferred types. Therefore, (i) a logical inconsistency will occur if there is a declared owl:disjointWith relation between a pair of t_i and t_j; (ii) no issue will occur if there is an rdfs:subClassOf relation between any pair of t_i and t_j; (iii) UIT potential issue will occur otherwise.

Causes: No subset or disjoint relation can be found between two expected types of an individual. The types come from: (i) the types declared in D and O', (ii) inferred from properties rdfs:domain, rdfs:range and owl:allValuesFrom in DC.

Detection: The following three SPARQL queries.

Case 1 (UIT- domain): Figure 3 illustrates an UIT-domain issue: in instance data D, W is declared to have a value for property madeFromGrape which has domain Wine; however, after importing all referenced ontologies O and performing OWL-DL inference, we find that W is a member of Winery, which is not related by subclass or disjoint relation with Wine. The SPARQL query solution to the general UIT-domain issue is in Table 1.

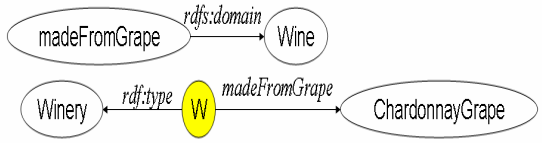

Figure 3. UIT-domain issue example

Table 1. SPARQL solution for UIT-domain issue

```
SELECT  ?i ?p ?x ?c1 ?c2
FROM …
WHERE {
  GRAPH D { ?i ?p ?x .}
  GRAPH DC {
   ?i rdf:type ?c1.
   ?p rdfs:domain ?c2.
   OPTIONAL {?c1 rdfs:subClassOf ?c3 .
                    ?c3 rdfs:subClassOf ?c2 .}
   FILTER( !BOUND(?c3) )
   OPTIONAL {?c2 rdfs:subClassOf ?c4 .
                    ?c4 rdfs:subClassOf ?c1. }
   FILTER( !BOUND(?c4) ) }}
//Note: type c1(i.e. Winery) is not known to be
compatible with expected type c2 (i.e. Wine)
```

Case 2 (UIT- range): Figure 4 shows an UIT-range issue: W has value Sweet for property hasColor; after loading O and OWL-DL inference, we know that all wine colors should be instances of WineColor; however Sweet is typed as WineSugar which has no subclass and disjoint relation with WineColor. The SPARQL solution is in Table 2.

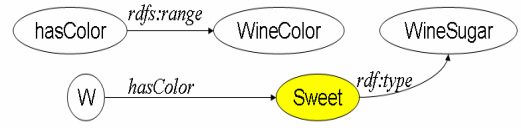

Figure 4. UIT-range issue example

[17] http://www.w3.org/TR/rdf-sparql-query/

Table 2. SPARQL solution for UIT-range issue

```
SELECT  ?i ?p ?x ?c1 ?c2
FROM …
WHERE {
  GRAPH D { ?i ?p ?x .}
  GRAPH DC {
    ?x rdf:type ?c1.
    ?p rdfs:range ?c2.
    OPTIONAL {?c1 rdfs:subClassOf ?c3 .
                  ?c3 rdfs:subClassOf ?c2 .}
    FILTER( !BOUND(?c3) )
    OPTIONAL {?c2 rdfs:subClassOf ?c4 .
                  ?c4 rdfs:subClassOf ?c1. }
    FILTER( !BOUND(?c4) ) }}
//Note: type c1 (i.e. WineSugar) is not known to be
compatible with expected type c2 (i.e. WineColor)
```

Case 3 (UIT- allValuesFrom): Figure 5 demonstrates an UIT- allValuesFrom issue: W has value Rose, which is typed as WineColor, for property hasMaker; however type Winery is expected because of the owl:allValuesFrom restriction on class Wine. The SPARQL query solution is in Table 3.

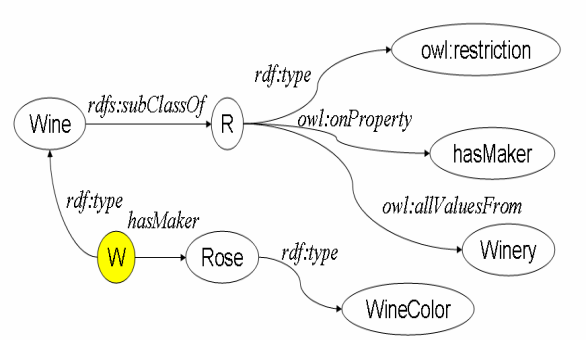

Figure 5. UIT- allValuesFrom issue example

Table 3. SPARQL solution for UIT-allValuesFrom issue

```
SELECT  ?i ?p ?x ?c1 ?c2
FROM …
WHERE {
  GRAPH D { ?i ?p ?x .}
  GRAPH DC {
     ?x rdf:type ?c1.
     ?i rdf:type ?c5.
     ?c5 rdfs:subClassOf ?r
     ?r rdf:type owl:Restriction.
     ?r owl:onProperty ?p.
     ?r owl:allValuesFrom ?c2.
     OPTIONAL {?c1 rdfs:subClassOf ?c3 .
                   ?c3 rdfs:subClassOf ?c2 .}
     FILTER( !BOUND(?c3) )
     OPTIONAL {?c2 rdfs:subClassOf ?c4 .
                   ?c4 rdfs:subClassOf ?c1. }
     FILTER( !BOUND(?c4) ) }}
//Note: type c1 (i.e. WineColor) is not known to be
compatible with expected type c2 (i.e. Winery)
```

PI-2. Redundant individual type (RIT)

Symptom: Given an individual i, assume its types declared in the instance data D are { td_1, td_2, …}, it may also have some types declared in instance data D or the referenced ontologies O' { tdo_1, tdo_2, …}, if any two types td_i and tdoj from these two group of types having a subset relation (not same) in DC, RIT potential issue will occur.

Causes: An individual is stated to be an instance of a class and its super-class at same time.

Detection: The solution is shown in Table 4.

Figure 6 depicts an RIT issue: W is stated to have types Zinfandel and Wine; however, Zinfandel is found to be a sub-Class of Wine in O; therefore, the type Wine of W is redundant.

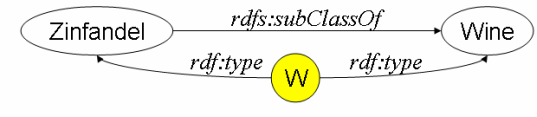

Figure 6. RIT issue example

Table 4. SPARQL solution for RIT issue

```
SELECT  i p x c1 c2
 FROM …
 WHERE {
    GRAPH D{  ?i rdf:type ?c1 . }
    GRAPH D+ O' { ?i rdf:type ?c2 . }
    GRAPH DC {
       ?c1 rdfs:subClassOf ?c2 .
        UNION
       ?c2 rdfs:subClassOf ?c1 .} }}
//Note: one type in c1 and c2 (i.e. Wine) is redundant
```

PI-3. Non-specific individual type (NSIT)

Symptom: Given an individual i, and its declared types { t_1, t_2, …} in D and O'. If there is a type t_i which has some sub-classes { t_{i1}, t_{i2} …} in DC, but i is not known to be an instance of any of the sub-classes, NSIT potential issue will occur.

Causes: An individual is stated to be an instance of a class which is a non-leaf node in the class hierarchy. Usually this is because the instance data author forgets to provide more specific type after assigning the individual a general type, or the author does not know the instance's specific type.

Detection: This issue could be detected using multiple queries, checking for membership relations between W and all of the leaf nodes of type t_i.

Figure 7 shows a NSIT issue: W is a Wine; however no more specific type is known even though

there are multiples wine subclasses provided in wine ontology, such as Zinfandel, Merlot, etc.

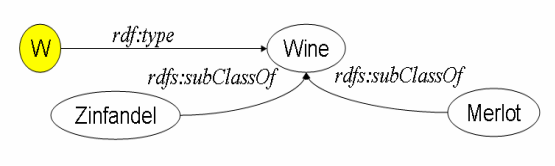

Figure 7. NSIT issue example

4.3.2 Issues related to property values

PI-4. Missing property value (MPV)

Symptom: Given an individual i that has type c, let m be the number of triples (i p x) in D, where p is a property. If there is a cardinality restriction in DC requiring all instances of c to have at least n values for property p, MPV issue will occur if m<n.

Causes: Violation of the owl:cardinality or owl:minCardinality restriction.

Detection: We can use two SPARQL queries to detect the MPV issues in OWL-Lite caused by owl:cardinality and owl:minCardinality respectively.

Case 1 (MPV- owl:cardinality): As shown in figure 8, W has no maker. However, W is expected to have one maker by the referenced ontologies. The SPARQL solution is in Table 5.

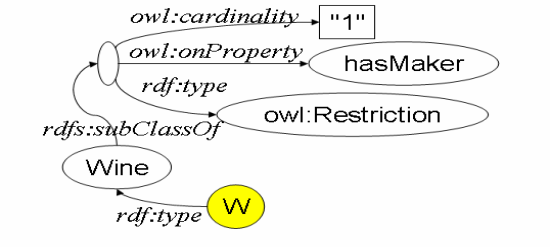

Figure 8. MPV- owl:cardinality issue example

Table 5. SPARQL solution for MPV-owl:cardinality issue

```
SELECT  ?i ?c ?p ?n
FROM …
WHERE {
   GRAPH D{  ?i rdf:type ?c .}
   GRAPH DC {
     c rdfs:subClassOf r;
     r rdf:type owl:Restriction;
     r owl:onProperty p;
     r owl:cardinality n.
     FILTER (?n > 0).
     OPTIONAL {?i ?p ?x.}
     FILTER (!BOUND(?x))}}
//Note: individual i (i.e. W) missing value for property p (i.e. hasMaker)
```

Case 2 (MPV- owl:minCardinality): It is similar to case 1. We do not give the details here.

PI-5. Excessive property value (EPV)

Symptom: Similarly, MPV issue would occur if there are more (i p x) triples in D than expected by referenced ontologies.

Causes: Violation of the owl:cardinality or owl:maxCardinality restriction.

Detection: This issue can not be resolved by a single SPARQL query. We have to find out all (i p x) triples in DC for given i and p, use owl:sameAs assertions to eliminate duplicate x values, then compute the number of distinct x values and compare it with the number expected by O.

Case 1 (EPV- owl:cardinality): Figure 9 depicts an EPV- owl:cardinality issue: V has two vintage years Year1990 and Year1998; however V is expected to have exactly one vintage year by O.

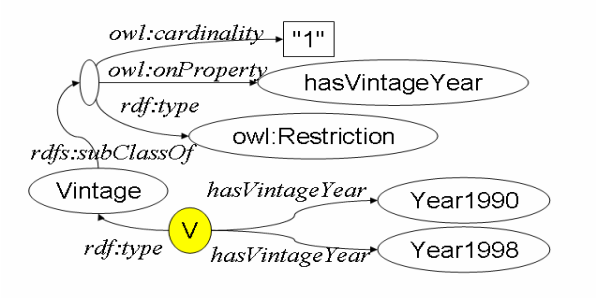

Figure 9. EPV-owl:cardinality issue example

Case 2 (EPV- owl:maxCardinality): It's similar to case 1 except owl:maxCardinality instead of owl:cardinaliy is used.

4.3.3 Application-specific and other issues

There are still some other issues caused by the application-specific expectation failures. These expectations require a flexible and extensible evaluation approach, so that users can customize their own evaluation criteria to check issues beyond syntax, logical consistency, and those frequently encountered potential issues. For example, some applications may require instances of some classes to always be named, i.e. they must be a URI instead of a blank RDF node. In other situations, users may require all instance descriptions to have an annotation property rdfs:label.

5. Implementation and evaluation

We have implemented our approach in TW OIE[18]. We integrated Jena and Pellet at the JAVA API level for syntax and logical inconsistency issues detection, and Pellet and SPARQL for potential issues detection.

[18] http://onto.rpi.edu/demo/oie/

In this section we will compare the TW OIE with five related tools discussed in section two: Pellet, WonderWeb species validator (WW), BBN validator (BBN), ODEval, and ConsVisor.

5.1 Experiment setting

Reference ontology loading: Many evaluation tools only load the directly referenced ontologies (using owl:import), and neglect those indirectly referenced ontologies (using namespace). Incomplete reference ontologies would make some tools incapable of detecting some issues in instance data. Our approach loads all of the referenced ontologies first before actually evaluating the instance data.

Test files: Instance data test files are generated such that each file contains one kind of issue, and all categories of issues (syntactic issues, semantic issues, and potential issues) are covered by the test files. To make up the effect of incomplete referenced ontologies loading of other tools, for each test file D_i, we preload all of its referenced ontologies, then merge them with the test file into a new file D_i' which is used as the actual input data for test. Since syntax error detection does not require loading reference ontologies, we do not change the original test files. The ten test files and associated issues are as follows:

- D_1 - No RDF triples
- D_2 - RDF triples with syntax errors
- D_3' - Plain literal missing data type
- D_4' - Cardinality violation
- D_5' - Inconsistent individual type
- D_6' - Unexpected individual type
- D_7' - Redundant individual type
- D_8' - Non-specific individual type
- D_9' - Missing property value
- D_{10}' - Excessive property value

Test procedures: Each tool is tested with all ten test files and we record whether the tool can successfully detect the issues in the file.

5.2 Experiment results

The results are shown in Table 6 where each result entry is denoted by the following notations:

- Y: the tool can correctly detect the error/issue.
- N: the tool can not detect the error/issue.
- PE: exceptions happened in evaluation.

5.3 Analysis

Table 6 shows that TW OIE successfully detects all of the issues in the test files. All of the other tools can detect the syntax issues, however, only Pellet, can find the semantic inconsistencies. We also determined that if Pellet had used its own referenced ontology loading strategy, it would not have been able to detect the cardinality violation and inconsistent individual type issues. This is because (i) some example instances appear to be in OWL-Full due to the lack of information required to classify an RDF resource into a class, a property or an individual; and (ii) Pellet's DL-reasoning is not engaged for these examples. Therefore, we observed that TW OIE outperforms other tools in loading referenced ontologies.

Table 6. Comparative experiment results

	Test Files	TW OIE	Pellet	WW	BBN	OD -Eval	Cons -Visor
Syntax Issues	D1	Y	Y	Y	Y	Y	Y
	D2	Y	Y	Y	Y	Y	Y
Semantic Issues	D3'	Y	Y	N	N	N	PE
	D4'	Y	Y	N	N	N	PE
	D5'	Y	Y	N	N	N	PE
Potential Issues	D6'	Y	N	N	N	N	PE
	D7'	Y	N	N	N	N	PE
	D8'	Y	N	N	N	N	PE
	D9'	Y	N	N	N	N	PE
	D10'	Y	N	N	N	N	PE

Among all of the other five tools, no tools can handle the potential issues encoded in the test files. Although ConsVisor claimed to check a lot of issues by associating the symptoms with underlying axiom violations, our experimental results show it is not able to detect the semantic inconsistency errors and potential issues in the given test files.

Through the initial comparative experiment, we have provided evidence that the TW OIE is a more comprehensive evaluation tool which can find not only syntax errors and logical inconsistencies, but also all of the listed potential issues. The other tools may do some error/issue checking, but no one is as comprehensive and powerful as TW OIE on our test cases.

6. Conclusions and future work

Effective instance data evaluation is critical to the success of semantic web based KMSs that integrate information from different sources. In this paper we have defined the instance data evaluation problem and suggested a practical general purpose solution GEP. Some frequently encountered potential issues in instance data are identified and then addressed by the corresponding SPARQL solutions. The solution is

also implemented in the TW OIE. Our initial evaluation demonstrates advantages over existing approaches and tools. We have also run the evaluation processes on other semantic web instance data sets such as those used in the Virtual Solar Terrestrial Observatory [20].

In future work, we will capture more potential issues, characterize them with formal logic languages, and improve system performance with emphasis on scalability, response time, and friendly user interface.

Acknowledgements: This work is partially supported by NSF #0524481(TAMI), DARPA #55-300000680(CALO), #55-002001(CALO), RPI contract number A71249 (GILA), and RPI's ITA project.

7. References

[1] K. Baclawski, C. J. Matheus, M. M. Kokar, J. Letkowski, and P. A. Kogut, Towards a Symptom Ontology for Semantic Web Applications, ISWC, pp. 650-667, 2004

[2] S. Bechhofer, and R. Volz, Patching Syntax in OWL Ontologies, ISWC, pp. 668-682, 2004.

[3] T. Berners-Lee, J. Hendler, and O. Lassila, The Semantic Web, Scientific American, 2001.

[4] ó. Corcho, A. Gómez-Pérez, R. González-Cabero, and M. del Carmen Suárez-Figueroa, ODEVAL: A Tool for Evaluating RDF(S), DAML+OIL and OWL Concept Taxonomies, AIAI, pp. 369-382, 2004.

[5] J. Davies, D. Fensel, and F. van Harmelen, Towards the Semantic Web – Ontology-driven Knowledge Management, John Wiley & Sons, West Sussex, 2003.

[6] J. Davies, M. Lytras, and A. Sheth, Semantic-Web-Based Knowledge Management, IEEE Internet Computing, Vol. 11, No. 5, pp. 14-16, 2007.

[7] L. Ding, and T. Finin, Characterizing the Semantic Web on the Web, ISWC, pp. 242-257, 2006.

[8] N. Drummond, and R. Shearer, The Open World Assumption, Presentation, The Univ. of Manchester, 2006.

[9] D. Fensel, Ontology-Based Knowledge Management, Computer, 2002.

[10] R. Fikes, and D. L. McGuinness, An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL, KSL Technical Report KSL-01-01, 2001.

[11] A. Gomez-Perez, Evaluation of Ontologies, International Journal of Intelligent Systems, Vol. 16, No. 3, pp. 391-409, 2001

[12] A. Gomez-Perez, Some Ideas and Examples to Evaluate Ontologies, AIA, pp. 299, 1995.

[13] T. R. Gruber, A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, Vol. 5, No. 2, pp.199-220, 1993.

[14] P. Haase, and L. Stojanovic, Consistent Evolution of OWL Ontologies, ESWC, pp. 182-197, 2005.

[15] A. Kalyanpur, B. Parsia, E. Sirin, and J. A. Hendler, Debugging Unsatisfiable Classes in OWL Ontologies, Journal of Web Semantics, Vol. 3 No. 4, pp. 268-293, 2005.

[16] D. Loshin, Enterprise Knowledge Management: The Data Quality Approach, Morgan Kaufmann, 2001.

[17] B. McBride, Jena: A Semantic Web Toolkit, Internet Computing, IEEE, Vol. 6, No. 6, pp. 55-59, 2002.

[18] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder, An Environment for Merging and Testing Large Ontologies, KR, pp. 483-493, 2000.

[19] D. L. McGuinness, and P. P. da Silva, Explaining Answers from the Semantic Web: the Inference Web Approach, Journal of Web Semantics, Vol. 1, No. 4, pp. 397-413, 2004.

[20] D. L. McGuinness, P. Fox, L. Cinquini, P. West, J. Garcia, J. L. Benedict, and D. Middleton, The Virtual Solar-Terrestrial Observatory: A Deployed Semantic Web Application Case Study for Scientific Research, IAAI, pp. 1730-1737, 2007.

[21] James Michaelis, Li Ding, and Deborah L. McGuinness, The TW Wine Agent: A Social Semantic Web Demo, In ISWC Poster and Demo Track (to appear), 2008.

[22] B. Parsia, E. Sirin, and A. Kalyanpur, Debugging OWL Ontologies, WWW, pp. 633-640, 2005.

[23] P. Plessers, and O. D. Troyer, Resolving Inconsistencies in Evolving Ontologies, ESWC, pp. 200-214, 2006.

[24] R. A. Rocha, S. M. Huff, P. J. Haug, D. A. Evans, and B. E. Bray, Evaluation of a Semantic Data Model for Chest Radiology: Application of a New Methodology, Methods of Information in Medicine, Vol. 37, No.4-5, pp. 477-490, 1998.

[25] H. Wang, M. Horridge, A. Rector, N. Drummond, and J. Seidenberg, Debugging OWL-DL Ontologies: A Heuristic Approach, ISWC, pp. 745-757, 2005.