Multilingual RDF and OWL

Jeremy J. Carroll[1] and Addison Phillips[2]

[1]HP Labs, Bristol, UK
jjc@hpl.hp.com
[2]webMethods, Sunnyvale, CA, US
aphillips@webmethods.com

Abstract. RDF uses the RFC3066 standard for language tags for literals in natural languages. The revision RFC3066bis includes productive use of language, country and script codes. These form an implicit ontology of natural languages for marking-up texts. Relating each language tag with classes of appropriately tagged literals allows this implicit ontology to be made explicit as an ontology in OWL in which every class in the ontology is a datarange. The treatment extends to XML Literals, which may have multiple embedded language tags. Further features of RFC3066bis such as the relationship with deprecated codes, language ranges and language tag fallback can be expressed in OWL. A small change to the RDF model theory is suggested to permit access to the language tag in the formal semantics, giving this ontology a precise formal meaning. Illustrative use cases refer to use of English, Japanese, Chinese and Klingon texts.

1 Introduction

RDF, the foundation of the Semantic Web standards, has some provision for the use of natural language text from multiple languages, distinguished by use of language tags. This is a minimal requirement, made explicit in [2], for a knowledge representation language which is intended for interoperable use on a World Wide scale. This provision depends on XML, which in turn depends on "*RFC 1766 or its successors*": the IETF is in the process of updating that track with RFC 3066bis [3].

RFC 3066bis contains the significant advance of a generative ontology of language identification, and this paper explores the natural step of expressing that ontology using the Web Ontology Language (OWL). In addition, RFC 3066bis (and the earlier versions) connect various registries etc. Correct use depends on some detailed expert knowledge. We show how that expert knowledge can be expressed within OWL, making it easier for non-experts to correctly formulate OWL expressions for text with specific linguistic properties.

We use three example use cases throughout the paper, and demonstrate how the advances we propose help construct better Semantic Web applications, in which linguistic knowledge is captured in OWL, rather than application code.

This paper introduces a few properties and a large number of classes, using a variety of namespaces. We use the following namespace prefixes: `rdfl:` `rdfl-dflt:` `core-lang:` `xmllit-all-lang:` `xmllit-some-lang:`

`xmllit-main-lang: presentable-lang: lang-range:`, but omit their binding to mutually distinct URIs. Such a binding is assumed.

2 Three Use Cases

2.1 Appropriate Display of Labels

This use case is described in the OWL Requirements [2]: a Semantic Web application has data to be displayed to an end user. Many of the resources in the knowledge base are to be displayed using one of the values of `rdfs:label`, selected to make a good match between the linguistic capabilities of the end-user and the language tag associated with that particular text string. A simplified example of such labels taken from the OWL Test Cases [6] is:

```
 <owl:Class rdf:ID="ShakespearePlay">
  <rdfs:label xml:lang="it">Opere di Shakespeare</rdfs:label>
   <rdfs:label rdf:parseType="Literal"><span
     xml:lang="ja">□□□□□□□□□<ruby>
      <rbc><rb>□</rb><rb>□</rb></rbc>
      <rtc><rt>□□</rt><rt>□□</rt></rtc>
     </ruby></span></rdfs:label>
 </owl:Class>
```

We consider a specific end user: Brian who is a mother tongue English speaker, with a good knowledge of Japanese, can read Kanji, and hence can make some sense of any language written in traditional Chinese characters; he still remembers some of his schoolboy French.

2.2 Finding all Klingon text in a knowledge base

A Star trek fan wishes to search an RDF knowledge base for all the Klingon text in it, and then to explore the knowledge base from these resources.

2.3 Multilingual Knowledge Base Construction

An open source Semantic Web knowledge base is developed. The project started in the US, and all the natural language text strings in it have been tagged as `"en-US"` (following RDF Concepts [1] and RFC3066bis [3]). Other plain literals, with text that is not intended as natural language are marked up with the empty language tag. Gradually groups of Chinese developers (some from the mainland and other groups from Taiwan) become involved. Their typical interest is in using some subset of the knowledge base, possibly with some additional axioms. Moreover, each group has a specific application in mind, involving specific queries which return literal values for end-user presentation. Also, depending on the intended users of the application, the presented text must be available in English or traditional Chinese or simplified Chinese or some combination. Clearly, some of the developers will need to add

traditional Chinese literals corresponding to the original US English literals; others will need to add simplified Chinese; but precisely when it is *necessary* to translate which literals can only really be determined by asking an OWL reasoner the relevant queries and comparing the results for the various natural languages.

3 Language and Text in the Semantic Web Recommendations

RDF and OWL use literal nodes for natural language text. Literal nodes are either plain literals or typed literals.

3.1 Plain Literals in RDF

A plain literal is a Unicode string paired with an optional language tag [1]. For natural language text, the tag should be used in accordance with RFC 3066 (or its successors). A plain literal without a tag is simply a string and, like data of type `xsd:string`, it is not appropriate for natural language text, which may mean different things in different languages. There is no support within RDF or OWL semantics for language tags, other than the ability to distinguish literals that differ in language tag (case insensitively).

3.2 XML Literals in RDF

For some natural language text it is beneficial to use additional markup, for example for indicating bi-directional text or Ruby [7] markup (seen in the earlier Japanese example). These are typed literals and hence do not have an explicit language tag, however language information may be embedded within the literal using `xml:lang`. However, there may be more than one such `xml:lang`, each of which will have different parts of the XML literal in its scope; or even if there is exactly one `xml:lang`, there may be some text that is outside its scope.

3.3 OWL DataRanges

OWL does provide support for describing classes consisting entirely of literals. These are classes with type `owl:DataRange` . A significant difference from RDF datatypes in that plain literals as well as typed literals may belong to a datarange.

There is only explicit support for finite sets of literals, and no encouragement to use class expressions. In OWL Full, however, it is possible to construct class expressions from dataranges using `owl:unionOf`, `owl:intersectionOf` and `owl:complementOf` and dataranges can be infinite and can be named. We will make extensive use of these capabilities.

4 Language Tags and their Generative Capacity

RFC 3066bis [3] is (currently) an Internet-Draft that expands and redefines the language tags used by `xml:lang` and other applications. The basic goal of a language tag is to identify the natural language of content in a machine accessible manner and the design of RFC 3066bis improves this capability.

The structure of all RFC 1766 and successor language tags is a series of subtags with an assigned meaning for each type of subtag. The combination of subtags forms a very rough ontology that approximates the historical, geographical, and linguistic distinction of various natural languages and their dialects.

Prior to RFC 3066bis, all language tags fell into two categories: 1) generative tags made up of a language subtag and optional country code; 2) registered tags that must be considered as a singular unit.

RFC 3066bis changes the structure of the registry so that the generative mechanism is always applicable and greatly restricts the ability to register tags. The highly generative structure makes language tags into a much more robust ontological structure on which to base applications (such as the ones described here).

Tags under the new scheme are comprised of five subtag types, plus user-defined extensions. Each subtag must appear in a specific position in a tag and has unique length and content restrictions. Thus it is always possible to identify each subtag and assign it meaning (even without access to the underlying standards). The four subtag types are: 1) language subtags defined by ISO 639-1 or ISO 639-2; 2) extended language subtags which are reserved for possible use with ISO 639-3 in the future; 3) script subtags defined by ISO 15924; 4) region subtags defined by either ISO 3166 or by the UN M.49 region codes; and 5) variant subtags which are defined either by registration with IANA or by private use subtags for specific expert use.

Thus an example RFC 3066bis tag looks something like:

```
en-Latn-US-boont-x-anExtension
zh-s-min-s-nan-Hant-CN
```

The structure of a tag is described by this BNF notation:

```
Lang*[-s-extlang][-script][-region][-variant][-x*[-extension]]
```

5 The Use Cases with Current Technology

To motivate the rest of the paper, we show the limitations of the current recommendations when faced with these simple tests.

5.1 Appropriate Display of Labels

Brian has clear, but complex linguistic preferences: 1) English, from whatever geographical region, with a preference for British English. 2) Japanese, in whatever script. 3) Any language in traditional Chinese script 4) French.

Each of these corresponds to a number of possible language tags; all but the third correspond to what RFC 3066bis calls a language range. However, there is no support

for language ranges in RDF or OWL. The third case, a ‘script range’ is not covered by RFC 3066bis, either. So, despite the ontological capabilities of both RFC 3066bis and OWL, someone, (either Brian or the application developer) needs to turn each of these preferences into either a list of explicit alternatives, or a wildcard matching; and then the application developer needs to write custom code in the display routine that selects all the known labels of a resource and find the best match.

This best match needs to be able to cope with both plain literals, in which the language tag is an explicit component, and XML Literals, in which language mark-up is embedded within the literal value (see the Japanese in the example). It is unlikely that Brian will be able to reuse his expression of preference for one Semantic Web application in a second application.

5.2 Finding all Klingon text in a knowledge base

Searching for Klingon text has an additional complexity: there are two possible tags for Klingon, the preferred primary tag of `“tli”` a recent addition to the ISO 639-2 registry, or the older IANA tag of `“i-klingon”`, grandfathered in RFC 3066bis.

Even if the Semantic Web knowledge base provides support for searching by language tag, or better by language range, and even if that support covers both plain literals and XML Literals, the Star Trek fan still needs to know that there are two tags, and has to perform two searches in place of one.

5.3 Multilingual Knowledge Base Construction

One approach is for each group to check every literal in the subset of the knowledge base that they are using and ensuring that an appropriate translation is available. This involves unnecessary translation work in that some of the literals may never be relevant to the queries being asked, and others may already have a translation available, but only through the application of an OWL reasoner. A second approach would be to ask each relevant query of the knowledge base, and then have custom code to examine the language tags in the literals returned, and to have further custom code to determine whether one such literal meets the linguistic requirements of the application. This custom code should be able to look inside XML Literals for embedded language information. It may also need to support the concepts of language range and language tag fallback from RFC 3066bis.

6 The Formal Machinery

OWL provides a language for describing ontologies: much of this paper shows how OWL can be used to describe the ontology of RFC 3066bis. In this case, it is attractive to integrate this description of an ontology in with the lower level machinery of RDF, and RDF’s use of language tags. This section defines a few properties and classes that, with suitable extensions to RDF semantics, serve to

expose the language information in RDF literals in a way that can then be integrated with OWL's class definition mechanisms.

6.1 Properties

The minimal requirement to use OWL's ontological capabilities to capture the relationships between language tags and literals using these tags, is that it should be possible to identify within RDF and OWL the tag of a literal. The current recommendations provide no relationship between the language tag of a literal and any other feature of the language. The approach we take is to define a new property `rdfl:lang` (where `rdfl:` is a new namespace used in this paper).

This property is such that the triple (*x* `rdfl:lang` *y*) is true if and only if one of the following:

1. *x* is a plain literal with language tag *z* (normalized to lowercase), and *y* is *z*`^^xsd:language`
2. *x* is an XML literal containing non-white character data (a text node or attribute value) in-scope of an xml:lang= '*z*' (*z* normalized to lowercase), and *y* is *z*`^^xsd:language`

To express this formally as a semantic extension, the above can be read as a definition of the property extension *Iext*(*I*(`rdfl:lang`)) using the notation of RDF Semantics [4]. This property contravenes the purely syntactic constraint that literals cannot be subjects, but that presents no intrinsic difficulties[1] (other than that a few of the examples in this paper are written with a non-standard notation).

The second part of the definition treats all languages actually used within an XML literal equally. For some applications it may be more appropriate to, when possible, identify the principle language of an XML literal. For example, consider

```
<span xml:lang="en">
  This is <span xml:lang="la">ad hoc</span>.</span>
```

The text is essentially English, even though it contains a Latin phrase. This can be identified by noting that the outermost language tag for all the text is "en". Thus we define a further property `rdfl:mainLang` which relates a XML literal to its outermost language tag (if unique). This similarly constitutes a semantic extension to RDF.

6.2 Classes

An omission from the RDF Vocabulary [8] is that there is no class of plain literals, we rectify that with `rdfl:PlainLiteral`.

A further consideration is that the typical use case involves presenting text to an end-user. An arbitrary piece of XML does not include any presentational guidelines, hence it is useful to sometimes restrict consideration only to XML using the XHTML,

[1] This is not breaking new ground, for example: `"foo" owl:differentFrom "bar" .`

Ruby, SVG, MathML namespaces. Thus we also define a class [2] `rdfl:XHTMLLiteral`, which is a subclass of the datatype `rdf:XMLLiteral.`

Neither of these classes can be defined using the built-in formal machinery of RDF or OWL semantics, and so, as for the properties, they are suggested as a semantic extension.

6.3 Internationalized Interpretations

To follow the language of RDF Semantics [4], we define an internationalized interpretation *I*, as an RDFS Interpretation (as defined in [4]), such that:

- for each plain literal with language tag *lg* in the vocabulary *V* of *I*, the typed literal *lg*`^^xsd:language` is in *V*.
- for each XMLLiteral *x* in *V* and every language tag *lg* occurring in *x* as the value of an `xml:lang` attribute, *lg*`^^xsd:language` is in *V*
- the property extensions of `rdfl:lang` and `rdfl:main-lang` are as above
- the class extension of `rdfl:PlainLiteral` and `rdfl:XHTMLLiteral` are as above

7 Defaults in RFC 3066bis

RFC 3066bis has specific advice suggesting defaults for script codes and geographical codes, viz:

> *Use as precise a tag as possible, but no more specific than is justified. For example, 'de' might suffice for tagging an email written in German, while 'de-CH-1996' is probably unnecessarily precise for such a task.*
>
> *Avoid using subtags that add no distinguishing information about the content. For example, the script subtag in 'en-Latn-US' is generally unnecessary, since nearly all English texts are written in the Latin script.*

Unfortunately, defaults are known to be problematic in the Semantic Web, for example OWL contains no provision for them, despite them being an objective of the OWL requirements [2].

There are two possible implications of the language in RFC 3066bis. One is that there are defaults implied by various combinations of subtags, such as the "default" of Latin script for a language tag "`en-US`".

The other possible implication is that an omitted subtag is an *implied range*, suggesting that a user will accept any value in that position. Thus the tag "`en-US`" really implies a language tag of "`en-*-US-*`".

We will address these implications in this paper by assuming the former for tags marking up data and further, that the rules for these defaults are shared public knowledge. Ideally these rules should be maintained by the IETF and kept at IANA. We return to implied ranges, for use when querying data, in section 9.2.

---

[2] This suffers from being non-extensible, different application environments may be able to support only XHTML and SVG, or XHTML and MathML etc.

The rules only address script codes and geographical codes, and each rule is a pair. The first item in the pair is a language tag missing either or both of a script subtag or a geographical subtag, the second item include the default values (if any) for the missing items. Some sample rules are as follows:

```
en en-latn
fr fr-latn-FR
zh-TW zh-hant-TW
```

The first rule says that English text defaults to the Latin script, and applies uniformly to any geographical (or private variant) of English, without some other explicit script code. The second rule says that French defaults to being in Latin script and from the geography France. These two defaults apply independently so that lacking a geographical code `fr-arab` defaults to `fr-arab-FR` (the French of France written in Arabic) rather than `fr-arab-015` (North African French written in Arabic)[3].

With these rules we note that it becomes impossible to mark up some texts in a very general way. For example, while it is possible to use a language code of `zh` for Chinese in a variety of scripts, it is not possible to use `en` for English in a variety of scripts. This is not an additional constraint on top of RFC 3066bis, but merely an articulation of a limitation that is already implicit within it.

Content authors should use a shorter form of any language tag, if available (following RFC 3066bis). We then apply the default rules by introducing two new properties `rdfl-dflt:lang` and `rdfl-dflt:mainLang` that are defined by axioms derived from the default rules, such as (in the OWL abstract syntax[4] [5]):

```
EquivalentClass(
  restriction( rdfl-dflt:lang value(“en-latn”) )
  unionOf( restriction( rdfl:lang value(“en-latn”) )
        restriction( rdfl:lang value(“en”) ) ) )

EquivalentClass(
  restriction( rdfl-dflt:lang value(“en-latn-us”) )
  unionOf( restriction( rdfl:lang value(“en-latn-us”) )
        restriction( rdfl:lang value(“en-us”) ) ) )
```

The first axiom says that for any resource the property `rdfl-dflt:lang` has the value `en-latn` if and only if the property rdfl:lang has the value `en` or `en-latn`.

Note that the second axiom is implicit in the first rule, when combined with the IS0 3166 country code `US`. Since there are many such codes there are very many of these axioms generated. In fact, when we consider private extensions such as `en-US-x-newyorkcity`, there are infinitely many axioms. We return to this issue in section 11. The axioms above, like all the axioms with sample language tags in this paper, should be seen as prototypical, invoking an infinite pattern.

For any language tag that is not (explicitly or implicitly) in the defaults table, we equate the `rdfl:lang` and `rdfl:mainLang` properties, e.g.:

---

[3] If this was thought inappropriate a more specific rule for `fr-arab` could be included in the table of rules.

[4] In all the examples we omit the datatype `^^xsd:language` inside the value construct. E.g. `value(“en-latn”)` abbreviates `value(“en-latn” ^^xsd:language)`

```
EquivalentClass(
  restriction( rdfl-dflt:lang value(“en-arabic”) )
  restriction( rdfl:lang value(“en-arabic”) ) )
```

We generate an equivalent infinite set of axioms defining `rdfl-dflt:mainLang` in terms of `rdfl:mainLang`.

In combination these axioms apply the defaults table to an RDF or OWL knowledge base, and from hereon we use the properties `rdfl-dflt:lang` and `rdfl-dflt:mainLang` in preference to `rdfl:lang` and `rdfl:mainLang`.

8 The Core DataRanges

In the previous section, we introduced two properties `rdfl-dflt:lang` and `rdfl-dflt:mainLang`, bound into an extended RDF model theory. OWL's ontological modeling capability is class focused, so we move from this simple property view into a class view. Since all these classes are classes of literals, they are dataranges. We use the namespace prefix "`core-lang:`" for these dataranges, with corresponding definitions[5] such as:

```
DataRange( core-lang:en-latn-us
 intersectionOf( rdfl:PlainLiteral
  restriction( rdfl-dflt:lang, value( “en-latn-us” ) )))
```

i.e. the datarange `core-lang:en-latn-us` is those plain literals which have a value for `rdfl-dflt:lang` of "`en-latn-us`". We have an infinite number of these definitions, one for each possible language tag (including private extension tags). Note that the class name is the language tag normalized to lower case. These classes are pairwise disjoint so that:

```
“hello world”@en-US rdf:type core-lang:en-latn-us .
```

But none of the following are true:

```
“hello world”@en-US rdf:type core-lang:en-latn .
“hello world”@en rdf:type core-lang:en-latn-us .
“hello world” rdf:type core-lang:en-latn .
```

The operation of the defaulting rules for the script code is apparent in these facts. Further, we see that `core-lang:en-us` is empty, since any literal explicitly tagged as `en-US` is subject to the default rule, and treated as `en-latn-US`.

In addition we define `core-lang:None` as the datarange of plain literals without a language tag.

```
DataRange( core-lang:None
 intersectionOf( rdfl:PlainLiteral
        restriction( rdfl:lang, cardinality=0 )))
```

[5] We extend the abstract syntax with Datarange axioms modeled on the class axiom of OWL DL. These are mapped to triples similarly to the class axiom.

8.1 XML Literals

XML Literals provide additionally complexity in that they may have more than one language tag. We model this complexity with three dataranges for each language tag. Using the prefix `xmllit-all-lang:` we create dataranges for XML Literals all of whose non-white character data content is tagged with the appropriate language tag; using the prefix `xmllit-some-lang:` we create dataranges for XML Literals some of whose non-white character data content is appropriately tagged; using the prefix `xmllit-main-lang:` we create dataranges for XML Literals all of whose non-white character data content is contained within ancestor XML elements with the appropriate `xml:lang` tag even if overridden on a closer ancestor element.

```
# The class of XMLLiterals wholly in language “it”
DataRange(xmllit-all-lang:it
 intersectionOf( rdf:XMLLiteral
    restriction( rdfl-dflt:lang, cardinality = 1 )
    restriction( rdfl-dflt:lang, value(“it”) )))
# The class of XMLLiterals partially in language “it”
DataRange(xmllit-some-lang:it
 intersectionOf( rdf:XMLLiteral
    restriction( rdfl-dflt:lang, value(“it”) )))
# The class of XMLLiterals partially in language “it”
DataRange(xmllit-main-lang:it
 intersectionOf( rdf:XMLLiteral
    restriction( rdfl-dflt:mainLang, value(“it”) )))
```

9 Approximate Matching

The core dataranges of the previous section do not provide any additional utility for our use cases. While they map the language tag into some base classes, the key issue in the use cases was the additional linguistic knowledge needed to make the best use of the language tags. In this section, we should how the ontology implicit in RFC 3066bis can be made explicit with further OWL datarange definitions.

9.1 Presentable Literals

For most use cases, the application wishes to display some natural language text.
In practice this text will be in the RDF graph either as a plain literal or an XML Literal which is XHTML. Other XML Literals are, in general, not useful, since they lack presentational information. Thus for each language tag we define a further datarange, using the namespace `presentable-lang:` defined in terms of the earlier definitions:

```
# Literals with tag “zh-hant”, (traditional Chinese)
# suitable for display or other presentation.
DataRange(presentable-lang:zh-hant
 unionOf( core-lang:zh-hant
```

```
    intersectionOf( rdfl:XHTMLLiteral
       restriction( rdfl:mainLang, value("zh-hant") ))))
```

9.2 Language Ranges

A language range is a way of matching a specific language tag and all extensions of it. Thus for every tag we can create a further datarange (with namespace prefix `lang-range:`) corresponding to the concept of language range, and the `presentable-lang:` datarange of all extension tags are subclasses:

```
# Language ranges corresponding of "zh-hant-CN",
# (traditional Chinese in mainland China)
SubClass(presentable-lang:zh-hant-cn lang-range:zh-hant-cn )
SubClass(presentable-lang:zh-hant-cn lang-range:zh-hant )
SubClass(presentable-lang:zh-hant-cn lang-range:zh )
```

Since we have already applied defaulting rules (in section 7), we can extend the concept of language range beyond that in RFC 3066bis, to permit the insertion of a script code when none is given. This corresponds to an additional axiom:

```
SubClass(presentable-lang:zh-hant-cn lang-range:zh-cn )
```

The language tag `zh-CN` is usually understood as being in simplified Chinese (script code `hans`), however that is using the default rules. For the classes we are defining on top of `rdfl-dflt:lang`, it is necessary to use the fully expanded form `zh-hans-cn` of the language tag (e.g. `lang-range:zh-hans-cn`) if that is what is desired. `lang-range:zh-cn` is understood as with a genuinely unknown script code, i.e. an implied range (cf. section 7).

It would be more accurate to define a language range as an infinite union formed with all extension tags. We return to this in section 11.

9.3 Script Ranges

A further extension to the concept of language range is to allow language ranges that contain only a script code, and omit the primary language tag. For example `lang-range:latn` is for all literals in Latin script; `lang-range:hant` for all literals in traditional Chinese script. A sample axiom defining such ranges is:

```
SubClass(presentable-lang:zh-hant-cn lang-range:hant )
```

9.4 Language Tag Fallback

RFC 3066bis suggests a process of language tag fallback to exploit "a [putative] semantic relationship between two tags that share common prefixes". This relationship is weaker than that exhibited with language ranges. This process excludes private use extensions, which are explicitly ignored. We support this in the ontology with the use of three annotation properties on language ranges. `lang-`

`range:fallback1` gives an alternative datarange that ignores extension tags, `lang-range:fallback2` also ignores the geographical code (if any), `lang-range:fallback3` also ignores the script code. Since we have expanded the default script information, it is generally unhelpful to use `lang-range:fallback3` except in specific cases where scripts have some degree of mutual intelligibility (e.g. simplified Chinese and traditional Chinese). An example axiom:

```
DataRange(lang-range:zh-s-min-s-nan-hant-cn
  annotation( lang-range:fallback1 lang-range:zh-hant-cn )
  annotation( lang-range:fallback2 lang-range:zh-hant )
  annotation( lang-range:fallback3 lang-range:zh ) )
```

9.5 Grandfathered Tags

When languages that have registered tags or subtags in the IANA registry are added to the ISO 639 registry, the old IANA entry is updated to show that it has been deprecated in favour of the ISO 639 entry. An example is Klingon, where the IANA tag `i-klingon` has been grandfathered in favour of `tli`. This can be expressed using `owl:DeprecatedClass` by expressing the identify between the corresponding language ranges:

```
DataRange( lang-range:i-klingon Deprecated lang-range:tli )
```

10 The Use Cases Revisited

10.1 Appropriate Display of Labels

Brian (or a tool he is using) can construct a sequence of dataranges from those given in this paper expressing his linguistic preferences. He needs to understand about language tag fallback, and when it is appropriate to use it. He needs to use fully specified language tags, expanded from their default form (e.g. `en-latn-GB` rather than `en-GB`), to form the language range expressions of interest.

10.2 Finding all Klingon text in a knowledge base

`lang-range:tli` includes plain and XML Literals with both the `tli` and `i-klingon` language tags. Using this an OWL reasoner can be used to find all instances of these literals in a knowledge base.

10.3 Multilingual Knowledge Base Construction

Each group can use the language ranges defined above to construct datarange expressions corresponding to the linguistic requirements of their subproject. These expressions can be combined with the queries over the knowledge base, as further

OWL expressions, which can then be passed to an OWL reasoner which can simply identify the work that needs to be done, taking into account all knowledge already implicit in the axioms.

11 Finiteness of Necessary Knowledge

This paper has presented an infinite ontology representing both the (large and finite) generativity of RFC 3066bis through the registered ISO and IANA registered codes and tags, and the infinite potential extensibility of RFC 3066bis through private extension tags. This raises the question of how a semantic web application can load all this knowledge, or does it require special purpose code to implement?

However, we saw in section 6 that we only added two properties and two classes to the core of the theoretical model. Since the actual vocabulary in any knowledge base is finite, only a finite subset of the infinite axioms presented here are relevant to any specific piece of reasoning. This subset can be determined from the language tags used in the vocabulary of the knowledge base. Moreover, the infinite unions mentioned at the end of section 9.2 can be replaced with finite unions determined by the vocabulary actually used. The formal machinery of section 6.3 is expressed only in term of such a finite vocabulary. Thus we structure the ontology described in this paper in such a way that a Semantic Web knowledge base should import a subontology generated by the language tags that occurs in the knowledge base. Since the KB is finite, this is a finite number of tags. The amount of knowledge for each tag is fairly small, and moreover, in a typical knowledge base the number of language tags is small, hence this is only a small overhead.

An appropriate implementation might be with a servlet that accepts a set of language tags in an HTTP POST action, or as a query on a URI, and returns the relevant subontology generated from the prototypes in this paper together with the tags in the request.

12 Additional Expert Knowledge

The ontology presented could be augmented with additional expert knowledge. In particular, there are relationships amongst the script codes, for example “Hrkt” (Hiragana + Katakana) is a plausible fall back for “Hira” (Hiragana) but not for “Brai” (Braille). There are also relationships amongst the geographical codes, particularly with the new addition in RFC 3066bis of the numerical UN geographical codes, for example the geography US is part of the geography 021 (northern America) but not of 419 (Latin America and the Caribbean). This results in plausible corresponding subclass relationships between `lang-range:es-latn-US` (US Spanish) and `lang-range:es-latn-021`, (north American Spanish) but not with `lang-range:es-latn-419` (Latin American Spanish). Fleshing this idea out is left as future work.

13 Suggestions for the Recommendations

We believe that appropriate support for multilingual applications is vital to the Semantic Web. The work presented here provides some significant forward steps. In this section we suggest changes that could be incorporated in future revisions of the three standards we have considered.

13.1 RDF

The model theoretic changes to support the two new properties and two new classes defined in section 6 should be added to the core RDF Recommendations.

13.2 OWL DL

With the changes to RDF, OWL Full would already support all the functionality we have described, using the class expressions we have shown. However, OWL DL excludes the possibility of infinite DataRanges, does not permit named DataRanges, nor does it permit class expressions in the definition of DataRanges. All of these could be added without compromising the design integrity of OWL DL datatyping, which depends on the separation of a datatype oracle as in [9] from the tableau reasoner. The core dataranges of section 8 can be treated as primitive datatypes, and the additional expressivity of forming (finite) unions and intersections of datatypes does not compromise the functioning of the datatype oracle. I.e. the datatype oracle can be extended to include the ability to handle the datarange expressions we have used.

13.3 RFC 3066bis

The generative capabilities of RFC 3066bis fit fairly naturally into OWL, and are distinctly easier to use than the somewhat *ad hoc* list of tags maintained by IANA under RFC 3066. The systematic inclusion of script codes into RFC 3066bis enables new functionality that recognizes that script is sometimes more important than language when trying to (partially) understand some natural language.

However, the continuation of the defaulting rules from RFC 1766 through to RFC 3066bis creates difficulties for true interoperability. These could be addressed by making the default rules more explicit as described in section 7.

The language range concept we have used is more powerful than the simple mechanism used in RFC 3066bis (for example, taking into account grandfathered codes). It may be appropriate for further revisions of that standard to incorporate some of the ideas.

14 Conclusions

We have shown that relatively small changes to RDF and OWL make it significantly easier to build and use multilingual knowledge bases. The generative capacity of RFC 3066bis can be modeled and exploited in OWL; while this results in an infinite ontology, this is usable in practice, because for any knowledge base a finite subset suffices.

15 References

[1] Klyne, G., Carroll, J. J. (eds.): RDF Concepts and Abstract Syntax. W3C Rec. 2004
[2] Heflin, J. (ed) OWL Use Cases and Requirements W3C Rec. 2004.
[3] Phillips, A., Davis, M.: Tags for Identifying Languages. draft-phillips-langtags-08, 2004 (also known as RFC 3066bis).
[4] Hayes, P. (ed.): RDF Semantics. W3C Rec 2004
[5] Patel-Schneider, P.F., Hayes, P., Horrocks, I. (eds.): OWL Semantics and Abstract Syntax. W3C Rec. 2004.
[6] Carroll, J.J., de Roo, J. (eds.): OWL Test Cases, W3C Rec. 2004.
[7] Sawicki, M., Suignard, M., Ishikawa, M., Dürst, M.,Texin, T., (eds) Ruby Annotation, W3C Rec. 2001.
[8] Brickley, D. Guha, R.V. (eds) RDF Vocabulary, W3C Rec 2004.
[9] Pan, J. and Horrocks, I. Extending Datatype Support in Web Ontology Reasoning, CoopIS/DOA/ODBASE 2002 pp 1067-1081 2002