I18n of Semantic Web Applications

Sören Auer+, Matthias Weidl+, Jens Lehmann+, Amrapali J. Zaveri+, and Key-Sun Choi*

+ Universität Leipzig, Institut für Informatik, Johannisgasse 26,
D-04103 Leipzig, Germany,
{lastname}@informatik.uni-leipzig.de
http://aksw.org
* KAIST, Semantic Web Research Center
335 Gwahangno Yuseong, Daejeon 305-701, Korea
kschoi@cs.kaist.ac.kr
http://www.kaist.edu

Abstract. Recently, the use of semantic technologies has gained quite some traction. With increased use of these technologies, their maturation not only in terms of performance, robustness but also with regard to support of non-latin-based languages and regional differences is of paramount importance. In this paper, we provide a comprehensive review of the current state of the internationalization (I18n) of Semantic Web technologies. Since resource identifiers play a crucial role for the Semantic Web, the internatinalization of resource identifiers is of high importance. It turns out that the prevalent resource identification mechanism on the Semantic Web, i.e. URIs, are not sufficient for an efficient internationalization of knowledge bases. Fortunately, with IRIs a standard for international resource identifiers is available, but its support needs much more penetration and homogenization in various semantic web technology stacks. In addition, we review various RDF serializations with regard to their support for internationalized knowledge bases. The paper also contains an in-depth review of popular semantic web tools and APIs with regard to their support for internationalization.

1 Introduction

Recently, the use of semantic technologies has gained quite some traction. With the growing use of these technologies, their maturation not only in terms of performance, robustness but also with regard to support of non-latin-based languages and regional differences is of paramount importance. Internationalization and localization are means of adapting computer software to different languages and regional differences. *Internationalization* is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes. *Localization* is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text. For the localization of Semantic Web applications, existing software methodologies (such as GNU gettext for translation or

different locales for region-specific data formating) can be applied. Also, with the *datatype* and *language tags* for RDF literals, there is good support for localization of knowledge bases. With regard to internationalization of Semantic Web technologies the situation, however, is much more challenging as we experienced during the process of internationalizing the DBpedia extraction framework [4] for creating a Korean version of DBpedia.

We noticed in particular, that Asian languages and resources pose a special challenge for Semantic Web and Linked Data applications, tools and technologies. The (non-standard) generation of URIs (for Asian language resources) can have a substantial impact also with regard to classification, interlinking, fusing and information quality assessment. The importance of tackling a proper internationalization of the Semantic Web technology stack is stressed by the fact that Asia has compared to Europe and the USA the largest number of Internet users[1] and many Asian languages are based on fundamentally different linguistic paradigms. Hence, for the success of individual tools and the Web of Data as a whole it is crucial (a) to incorporate and outreach to user communities beyond the western world and (b) to consider the varying linguistic paradigms in order to achieve a wider applicability of the Semantic Web research and development results.

In this paper we want to contribute to a successful internationalization of Semantic Web technologies by summarizing our findings, providing a comprehensive review of the current state of the internationalization (I18n) of Semantic Web technologies and outlining some best-practices for Semantic Web tool and application developers as well as knowledge engineers.

We are starting to look at the situation with one of the uttermost important building blocks of the Semantic Web – resource identifiers. It turns out that the currently prevalent resource identification mechanism on the Semantic Web, i.e. *URIs*, are not sufficient for an effective internationalization of knowledge bases. Fortunately, with IRIs a standard for international resource identifiers is available, but its support needs much more penetration and homogenization in various semantic web technology stacks. Hence, one goal of this paper is to sensitize the Semantic Web community for the use of IRIs instead of URIs. We review various RDF serializations with regard to their support for internationalized ontologies and knowledge bases. Surprisingly, also here, the currently prevalent serialization technology - RDF/XML - is not adequate for serializing internationalized knowledge bases. The paper also contains an in-depth review of popular Semantic Web tools and APIs with regard to support for I18n.

The paper is structured as follows: We describe the internationalization issues with URIs and possible solutions in the Sections 2. We describe problems with regard to internationalization of the RDF/XML serialization in Section 3. We survey the other available RDF serialization techniques for their compatibility with internationalization in Section 4. We also provide a comprehensive evaluation of internationalization support in popular Semantic Web tools and APIs in Section 5 and conclude in Section 6.

[1] `http://www.internetworldstats.com/stats3.htm`

2 What's wrong with URIs?

Resource identifiers are one of the main building blocks of the Semantic Web. The concept of Universal Resource Identifiers (URIs) is the prevalent mechanism for identifying resources on the Semantic Web. URIs only use US-ASCII characters for names of the resources. However, from the standpoint of an internationalization, URIs are not suitable since characters from non-latin alphabets or special characters have to be encoded in a cumbersome way. The W3C suggests to use percent-encoding in such cases, where a special character is encoded using its two digit hexadecimal value prefixed with the percent character "%". For example, "%20" is the percent-encoding for the US-ASCII space character.

There exist different character encodings for different languages or language families, such as *ISO 8859-1* for Western Europe, *KS X 1001* for the Korean language or *Big-5* for traditional Chinese characters. If these encodings would be used for URIs, conflicts would arise when merging knowledge bases using different encodings for their URIs. For this purpose, the W3C suggested the use of UTF-8 for encoding in URIs[2] as it supports almost every language currently used in the world; it is widely supported, needs one to four bytes to encode the characters and also preserves US-ASCII characters[3]. First the characters not allowed in URIs are encoded according to UTF-8 and then each byte of the sequence is percent-encoded. The Korean word 베를린 (transcription of Berlin), for example, encoded in UTF-8 and percent-encoded looks as follows:

1. byte (베): EB B2 A0
2. byte (를): EB A5 BC
3. byte (린): EB A6 B0

http://ko.wikipedia.org/wiki/%EB%B2%A0%EB%A5%BC%EB%A6%B0

Even though the use of percent-encoding solves the problem of representing special characters, the URIs are (as the above example demonstrates) not easily readable by humans. In software applications it adds additional overhead, since URIs have to be encoded and decoded. It also has to be considered that different parts of the URI (such as the server name, the path and the local name) have different sets of allowed characters or that have to be encoded differently.

IRIs for the Semantic Web. To eliminate the disadvantages of URIs, the idea to use UTF-8 without percent-encoding in resource identifiers was raised by Francois Yergeau in 1996[4]. The W3C introduced IRIs (Internationalized Resource Identifier) in 2001 [3]. With the use of IRIs and UTF-8 it is possible to use all characters of the Unicode standard, which covers 107,000 characters and 90 different scripts[5]. With this technology users can easily use, read, alter, and create

[2] `http://www.w3.org/TR/REC-html40/appendix/notes.html`
[3] `http://unicode.org/faq/utf_bom.html`
[4] `http://www.w3.org/International/O-URL-and-ident.html`
[5] `http://www.unicode.org/standard/principles.html`

IRIs in their native language. The above mentioned URI for Berlin in Korean as an IRI would look as follows:

http://ko.wikipedia.org/wiki/베를린

Since non-US-ASCII characters do not have to be encoded, the "%" character does not have to be used in most cases. XML also does support IRIs and UTF-8 but does not allow certain characters in XML tags like %, (,), or & and some others. But since IRIs can contain all these characters, it can cause problems when serializing RDF in XML as we discuss in detail in the next section. Since most of the knowledge bases on the Semantic Web are currently represented using URIs, they have to be converted to IRIs. The challenge is to figure out whether a percent-encoded sequence was created from a legacy encoding or from UTF-8. But there exists a high chance of heuristic identification of UTF-8 [2] and rare coincidences with byte sequences from legacy encodings as is pointed out in [3]. Furthermore, URIs and IRIs are identical as long as no special characters are used [3].

The main security problem with IRIs is spoofing, because some characters are visually almost indistinguishable. This problem is an extension of those for URIs. However, because UTF-8 contains far more characters the chance for spoofing may increase. One example is the similarity of the Latin "A", the Greek "Alpha", and the Cyrillic "A" [5].

3 What's wrong with RDF/XML?

RDF/XML and the RDF embedding in XHTML (i.e. RDFa) are the only RDF serializations officially recommended by the W3C. In essence, RDF/XML is an XML dialect defined by an XML schema for representing data adhering to the RDF data model. In XML documents, it is possible to use different languages simultaneously even when they use different alphabets. To specify the language for content in the document the *xml:lang* attribute is used. XML markup, however, such as tag and attribute names, is not affected by this attribute. RDF resources are in RDF/XML represented as XML markup, i.e. XML tags and attributes of XML tags. XML tag names, for example, are limited to the following characters [7]:

```
Name ::= NameStartChar (NameChar)*
NameStartChar ::= ":" | [A-Z] | "_" | [a-z] | [#xC0-#xD6] |
[#xD8-#xF6] | [#xF8-#x2FF] | [#x370-#x37D] | [#x37F-#x1FFF] |
[#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] |
[#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] |
[#x10000-#xEFFFF]
NameChar ::= NameStartChar | "-" | "." | [0-9] | #xB7 |
[#x0300-#x036F] | [#x203F-#x2040]
```

Consequently, some IRIs, which use certain characters of the UTF8 encoding, cannot be used with RDF/XML and there are no simple workarounds or solutions to this dilemma. A solution to this problem would either require a change of the XML standard to allow UTF-8 encoded XML tag names, or a substantial change or extension of RDF/XML, which allows to represent IRIs as XML content.

As we described in Section 2, non US-ASCII characters can be represented in URIs by using the percent-encoding. The "%" character in UTF-8 is defined as #x0025 and not allowed for XML tag names (cf. XML tag name definition above). Thus, a RDF graph with non US-ASCII characters cannot be serialized in RDF/XML. The possible workarounds to this problem are:

1. Use of a different character or sequence of characters instead of the "%" character.
2. Add an underscore to the end of an URI with encoded characters.
3. Use of a different RDF serialization, possibly with IRIs instead of URIs.

Encoding of the % character. The solution used by DBpedia for the English DBpedia edition was to replace the "%" character by "_percent_". With this solution the default Wikipedia encoding (i.e. percent-encoding) could be maintained. But this solution produced errors during the DBpedia extraction process for languages with many special characters like Korean, for example. This is due to the fact that after replacing the "%" character with e.g. "_percent_" URIs get very long and although the URI length is not a constraint, some tools (such as the Internet Explorer[6]) have trouble processing very long URIs. This solution is also problematic since RFC3986 states: "URI producers should ... limit these names (URIs) ... to no more than 255 characters in length" [6]. If we would limit URIs to 255 characters, Korean URIs could only encode 7 Korean characters. To increase this number, it would be possible to use a shorter replacement, for example, "_p_" instead of "_percent_" for the "%" character. Thus, a Korean URI could reach up to 17 characters. Such a solution may be sufficient for some applications but renders URIs with non-Latin characters unreadable by humans. Also exchanging data using these URIs with other applications not being aware of the non-standard % encoding (or making already otherwise use of the encoding sequence) will render the encoding irreversible.

Underscore workaround. During the search for different solutions it was discovered that adding an underscore at the end of every URI with special characters makes it possible that this URI can be serialized in XML with certain tools e.g. Jena [7] even though this did not adhere to the XML standard as mentioned above. This approach was used for the Korean DBpedia, because it enabled to fully maintain the Wikipedia encoding. Furthermore, this workaround did not require dramatic changes, since only the underscore at the end of an URI had to be considered.

[6] `http://support.microsoft.com/kb/208427`

[7] `http://lists.w3.org/Archives/Public/semantic-web/2009Nov/0116.html`

Use of a different RDF serialization. The last mentioned possibility is to use a different RDF serialization with IRIs instead of URIs. This solution is discussed more in detail in the following Section and different serializations are evaluated wrt. their support of IRIs.

Furthermore, the idea was raised to use XML entities for special characters instead of percent-encoding. First, the special characters have to be encoded according to UTF-8. The encoded character can be represented in two formats: &#nnnn; for the decimal form and &#xhhhh; for the hexadecimal form. Unfortunately, the & and the # character are both reserved characters in URIs and, thus, XML entities cannot be used for replacing percent-encoding.

4 Looking at other RDF serializations

In addition to RDF/XML, there exist a number of other RDF serialization formats, which partially go beyond the RDF data model in their expressivity and differently accentuate the balance between human readability and simple machine processability. In this section, we assess these different RDF serializations with regard to their support for internationalized resource identifiers. In our comparison, we include JSON, N-Triples, Notation 3, Turtle and RDFa. In order to demonstrate the support for international resource identifiers in the different formats, we showcase the RDF triples from Listing 1 serialized in each of the different formats. These example triples are an exerpt from the Korean DBpedia describing the Korean Advanced Institute of Science and Technology (KAIST).

```
1  http://ko.dbpedia.org/resource/KAIST
2  http://ko.dbpedia.org/property/ 이름
3  "KAIST 한국과학기술원"
4
5  http://ko.dbpedia.org/resource/KAIST
6  http://ko.dbpedia.org/property/ 설립
7  "1971"
8
9  http://ko.dbpedia.org/resource/KAIST
10 http://ko.dbpedia.org/property/ 종류
11 http://ko.dbpedia.org/resource/ 국립대학
```

Listing 1. RDF triples of KAIST.

JSON JSON (JavaScript Object Notation) was developed for easy data interchange between human beings and applications as well. JSON, allthough carrying JavaScript in its name and being a subset of JavaScript, meanwhile became a language independent format which can be used for exchanging all kinds of data structures and is widely supported in different programming languages. Compared to XML, JSON does require less overhead wrt. parsing and serializing. There is a non-standardized specification[8] for RDF serialization in JSON.

[8] `http://n2.talis.com/wiki/RDF_JSON_Specification`

Text in JSON and, thus, also RDF resource identifiers are encoded in Unicode and hence can contain IRIs. Also, there is no problem to use the "%" character for URIs in JSON. The only characters that must be escaped are quotation marks, reverse solidus, and the control characters (U+0000 through U+001F)[9]. Thus, JSON can be considered to be an excellent solution for the serialization of internationalized RDF.

```
{
  "http://ko.dbpedia.org/resource/KAIST" : {
    "http://ko.dbpedia.org/property/ 이름" : [ { "type" : " literal ",
        "lang" : "ko", "value" : "KAIST 한국과학기술원" } ],
    "http://ko.dbpedia.org/property/ 설립" : [ { "type" : " literal ",
        "value" : "1971" } ],
    "http://ko.dbpedia.org/property/ 종류" : [ { "type" : "uri", "lang" :
        "ko", "value" : "http://ko.dbpedia.org/resource/ 국립대학" } ]
  }
}
```

Listing 2. RDF triples in RDF/JSON.

N-Triples. This serialization format was developed specificially for RDF graphs. The goal was to create a serialization format which is very simple. N-Triples are easy to parse and generate by software. They are a subset of *Notation 3* and *Turtle* but lack, for example, shortcuts such as CURIEs. This makes them less readable and more difficult to create manually. Another disadvantage is that N-triples use only the 7-bit US-ASCII character encoding instead of UTF-8. Thus, it does not support IRIs but can handle the "%" character.

```
<http://ko.dbpedia.org/resource/KAIST> <http://ko.dbpedia.org/property/%EC
    %9D%B4%EB%A6%84> "KAIST\n\uD55C\uAD6D\uACFC\uD559\uAE30\
    uC220\uC6D0"@ko .
<http://ko.dbpedia.org/resource/KAIST> <http://ko.dbpedia.org/property/%EC
    %84%A4%EB%A6%BD> "1971"^^<http://www.w3.org/2001/XMLSchema#
    gYear> .
<http://ko.dbpedia.org/resource/KAIST> <http://ko.dbpedia.org/property/%EC%
    A2%85%EB%A5%98> <http://ko.dbpedia.org/resource/%EA%B5%AD%EB%
    A6%BD%EB%8C%80%ED%95%99> .
```

Listing 3. RDF triples in N-Triple.

Notation 3. N3 (Notation 3) was devised by Tim Berners-Lee and developed just for the purpose of serializing RDF. The main aim was to create a very human-readable serialization. That's why a RDF model serialized in N3 is much more compact than the same model in RDF/XML but still allows a great deal of expressiveness. The encoding for N3 files is UTF-8. Thus, the use of IRIs does not pose a problem. The "%" character can be used at any place that is allowed in RDF.

[9] `http://www.ietf.org/rfc/rfc4627.txt`

```
@base <http://ko.dbpedia.org/resource/> .
@prefix koprop: <http://ko.dbpedia.org/property/> .

<KAIST> koprop: 이름"KAIST 한국과학기술원"@ko .
<KAIST> koprop: 설립"1971"^^<http://www.w3.org/2001/XMLSchema#gYear> .
<KAIST> koprop: 종류< 국립대학> .
```

Listing 4. RDF triples in Notation 3 (or Turtle).

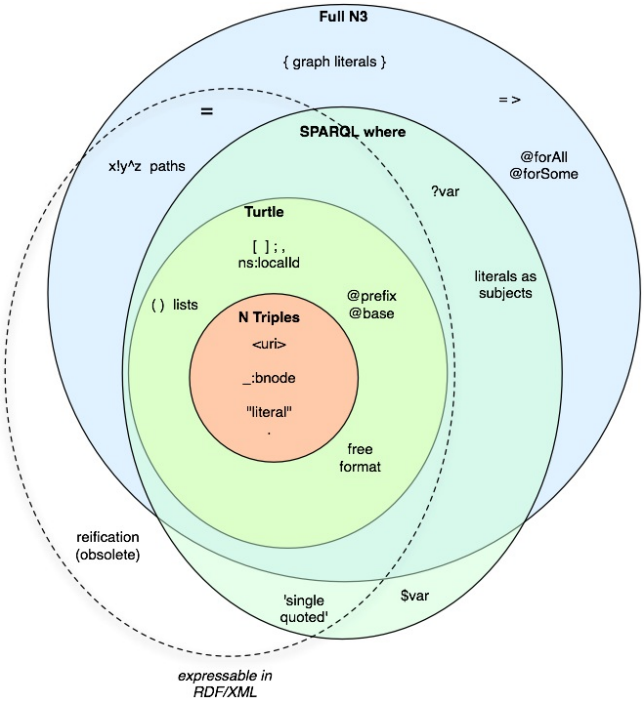

Fig. 1. N3 subsets [1]

Turtle. Turtle (Terse RDF Triple Language) is a subset of, and compatible with, Notation 3 and a superset of the minimal N-Triples format (cf. Figure 1). The goal was to use the essential parts of Notation 3 for the serialization of RDF models and omit everything else. Turtle became part of the SPARQL query language for expressing graph patterns. Turtle, just like Notation 3, is human-readable, can handle the "%" character in URIs as well as IRIs due to its UTF-8 encoding. Our example in Turtle format is exactly the same as in Notation 3, since Notation 3 specific syntax is required.

Table 1. Overview of RDF serialization techniques.

Technique	Percent-encoding	UTF-8/IRI support	Expressivity	Readability	Overhead
RDF/XML	n	r	+	◯	◯
JSON	s	s	+	+	+
N-Triples	s	n	-	+	+
Notation 3	s	s	+	++	++
Turtle	s	s	+	++	++
RDFa	s	s	+	+	◯

++: Very good +: Good ◯ : Moderate -: Poor
s:supported r: supported with some restrictions n: not supported

RDFa. RDFa[10] (RDF in Attributes) was developed for embedding RDF into XHTML pages. Since it is an extension to the XML based XHTML, UTF-8 and UTF-16 are used for encoding. The "%" character for URIs in triples can be used because in RDFa tags are not used for a part of a RDF statement. Thus IRIs are usable, too. Because RDFa is embedded in XHTML, the overhead is bigger compared to other serialization technologies and also reduces the readability.

```
<!DOCTYPE html PUBLIC "−//W3C//DTD XHTML+RDFa 1.0//EN" "http://
    www.w3.org/MarkUp/DTD/xhtml−rdfa−1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:kores="http://ko.dbpedia.org/resource"
      xmlns:koprop="http://ko.dbpedia.org/property">
  <head>
    <title></title>
    <meta http−equiv="Content−Type" content="text/html; charset=utf−8"/>
  </head>
  <body>
    <div about="kores:KAIST">
      <span property="koprop: 이름">KAIST 한국과학기술원</span>
      <span property="koprop: 설립">1971</span>
      <span property="koprop: 종류">kores: 국립대학</span>
    </div>
  </body>
</html>
```

Listing 5. RDF triples in RDFa.

[10] `http://www.w3.org/2001/sw/wiki/RDFa`

5 Tool Evaluation

In this section, we review some popular Semantic Web applications and APIs with regard to their support for internationalization. For testing purposes we use the Korean DBpedia edition in two different versions: firstly percent-encoded and secondly with IRIs.

5.1 OntoWiki and Erfurt API

OntoWiki[11] is a tool for agile, distributed knowledge engineering scenarios. It follows a Wiki like approach for browsing and authoring of RDF knowledge bases. It offers different views on the stored information and an inline editing mode for RDF data. Social collaboration aspects are added as well. It can be used with relational databases and triple stores and is based on the Erfurt API, an API for developing Semantic Web applications, which is written in PHP. Besides RDF/XML, OntoWiki also supports Turtle, RDF/JSON, and Notation 3 for exporting RDF data.

We tested OntoWiki 0.9 with a MySQL database with UTF-8 encoding and collation for storing the RDF data. OntoWiki supports percent-encoded URIs as well as IRIs. It was possible to load both Korean DBpedia editions in OntoWiki. Unfortunately, when accessing triples with IRIs, the error message "Illegal mix of collations" appeared (but this seems to be fixed in version 0.9.5 alpha). Exporting percent-encoded triples did not cause problems. The serialization of RDF/XML with percent-encoded properties was performed, although this is not allowed as outlined above. When exporting triples with IRIs, the properties only contained question marks, independently of the serialization format used.

5.2 Protégé

Protégé[12] is a Java desktop application for developing ontologies and knowledge bases. Protégé 3 supports OWL 1. The first version of OWL does not support IRIs and accordingly Protégé 3 does not support IRIs, too. Protégé 4 supports OWL 2 and IRIs. Protégé serialized IRIs without errors but IRIs were encoded using HTML entities:

```
http://ko.dbpedia.org/Ontology1276166885490.owl#CA_&#50724;
&#49324;&#49688;&#45208;/
```

Working with percent-encoded URIs in Protégé does not pose a problem. But when serializing such URIs in RDF/XML, which should not work, Protégé often serialized the triples without any error. The supported serialization formats of Protégé include Notation 3 and Turtle.

[11] http://ontowiki.net/
[12] http://protege.stanford.edu/

5.3 Virtuoso Universal Server

Virtuoso Universal Server is a middleware and database engine which contains a triple store to save and query RDF graphs. There exists an open source and a commercial edition of Virtuoso, which vary with regard to support and some functionality (e.g. clustering). Virtuoso handles RDF graphs with percent-encoding in URIs very well. When loading the Korean DBpedia data set, which was serialized in N-Triples format, no problems have been discovered and the data could be easily accessed using the integrated SPARQL endpoint. When uploading data sets with IRIs, however, Virtuoso did not accept all triples. 92 triples of the dataset, consisting of 5.21 million triples, were not processed correctly. The reason is that some triples contain the ”>” character, which in Notation 3 format represents the end of the subject, predicate, or object, respectively. If this character occurs, the Virtuoso parser probably assumes that the end of the N-Triple representation of the triple has been reached and expects the ”.” character. We will analyze this issue in more detail, since it leads to a part of the IRI specification, which is problematic in our opinion. The following is an example triple from the Korean DBpedia extraction:

```
<http://ko.dbpedia.org/resource/더 _ 로드>
<http://ko.dbpedia.org/property/wikilink>
<http://ko.dbpedia.org/resource/< 로드> >  .
```

In Listing 6 we look at the ABNF of the IRI specification (RFC 3987) [5] in order to determine whether this kind of IRI is allowed and should be accepted by Virtuoso.

```
1 IRI = scheme ":" ihier-part [ "?" iquery ] [ "#" ifragment ]
2 ihier-part = "//" iauthority ipath-abempty / ipath-absolute
3              / ipath-rootless / ipath-empty
4 iauthority = [ iuserinfo "@" ] ihost [ ":" port ]
5 ipath-abempty = *( "/" isegment )
6 isegment = *ipchar
7 ipchar = iunreserved / pct-encoded / sub-delims / ":" / "@"
```

Listing 6. Excerpt of the IRI specification ABNF.

The observed issue is related to the *ihier-part*. The *iauthority* part starts with a double slash (”//”) and is terminated by another slash (”/”), a question (”?”) mark or number sign (”#”), or the end of the IRI. In RDF *iuserinfo* and *port* are not needed and *ihost specifies* the server address, in this case `http://ko.dbpedia.org`. The second part of the *ihier-part* is *ipath-abempty* which is formed by a slash (”/”) followed by *isegment*: The first time *ipath-abempty* is used for ”resource” and the second time for ”/< 로드 >”. The first part (”resource”) only uses US-ASCII characters and, thus, is correct. *ipchar* are formed as follows: *iunreserved* contains all unreserved characters from URIs (see Table 2) as well as *ucschars*:

Table 2. Unreserved characters in URIs [6]

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	-	_	.	~												

```
ucschar = %xA0-D7FF / %xF900-FDCF / %xFDF0-FFEF
          / %x10000-1FFFD / %x20000-2FFFD / %x30000-3FFFD
          / %x40000-4FFFD / %x50000-5FFFD / %x60000-6FFFD
          / %x70000-7FFFD / %x80000-8FFFD / %x90000-9FFFD
          / %xA0000-AFFFD / %xB0000-BFFFD / %xC0000-CFFFD
          / %xD0000-DFFFD / %xE1000-EFFFD
```

pct-encoded refers to percent encoding ("%" HEXDIG HEXDIG) and *sub-delims* are a part of the reserved characters of URIs (see Table 3).

The *isegment* part of the object of our example IRI uses Korean characters (로드), which are allowed in IRIs (these are contained in *ucschar*). The characters "<" and ">" that caused the problems (003C and 003E) are not included in the unreserved characters but are also not mentioned in the reserved characters for IRIs. Furthermore, RFC 3987 [5] states "Systems accepting IRIs MAY also deal with the printable characters in US-ASCII that are not allowed in URIs, namely "<", ">", ' " ', space, "{", "}", "|", "\", "^", and " ‘ " ". In our opinion, this part of the specification is problematic, because it allows different tools and APIs to handle such IRIs differently, since it is not defined exactly how to handle these characters. This conflicts with the goals of Semantic Web technologies, which aim at a clear and unambiguous transfer of knowledge between different systems.

Table 3. sub-delim characters in IRIs [5]

!	*	'	(	)	;	&	=	+	$	,

Due to these reasons and because "<" and ">" are used for the start and end of the subject, predicate, or object in Notation 3, these characters should be percent-encoded to avoid misunderstandings. In this case Virtuoso accepted all triples. Thus, Virtuoso appears to be a very good solution for storing RDF graphs with URIs as well as IRIs.

5.4 Jena

Jena[13] is an open-source Java framework for developing Semantic Web applications. It includes a RDF API, an OWL API, a SPARQL query engine, an inference engine, and it can read and write RDF in RDF/XML, Notation 3 and N-Triples.

[13] `http://jena.sourceforge.net/`

http://ko.dbpedia.org/resource/도봉_분기점/고속도로_나들목3	http://ko.dbpedia.org/property/형태	IC X.svg
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/wikiPageUsesTemplate	http://ko.dbpedia.org/resource/Template:coor_dms
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	E
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	N
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	3
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	35
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	47.2164
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	47.2428
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	53
http://ko.dbpedia.org/resource/도리데_역/좌표/coor_dms	http://ko.dbpedia.org/property/coorDmsProperty	140
http://ko.dbpedia.org/resource/도리데_역/인접정차역1	http://ko.dbpedia.org/property/wikiPageUsesTemplate	http://ko.dbpedia.org/resource/Template:인접정차역

Fig. 2. Virtuoso with the Korean DBpedia data set and IRIs.

When testing the Korean DBpedia with the Jena framework and the output format RDF/XML, Jena stopped and threw an `InvalidPropertyURIException`. This behaviour was expected, because percent-encoded URIs cannot be serialized in XML. When adding an underscore to properties with such URIs, Jena magically could serialize the triples. In this workaround the namespace is (due to a ”misimplementation” of the URI segmentation algorithm in Jena) extended with a part of the property name. Only the underscore is written in the start-tag and end-tag. Thus, the tags do not contain a percent character and the XML file becomes valid. An example of Jena's RDF/XML serialization of triples with the underscore appended to URIs is shown in Listing 7.

```
1 <?xml version=”1.0”?>
2 <rdf:RDF
3     xmlns:rdf=”http://www.w3.org/1999/02/22−rdf−syntax−ns#”
4     xmlns:j.0=”http://ko.dbpedia.org/property/%EC%A0%9C%EB%AA%A9”>
5   <rdf:Description rdf:about=”http://ko.dbpedia.org/resource/Rock_U_%281
        st_Mini_Album%29”>
6     <j.0:_ xml:lang=”ko”>Good Day</j.0:_>
7   </rdf:Description>
8 </rdf:RDF>
```

Listing 7. Korean DBpedia in RDF/XML by Jena framework.

The Jena framework handles IRIs generally well. However, there exist some triples which were not accepted, such as the following:

`<http://ko.dbpedia.org/resource/2006 년 _ 태풍/태풍 _ 정보 _(소)1>`

```
<http://ko.dbpedia.org/property/최대강풍반경 (kma)>
"550"^^<http://www.w3.org/2001/XMLSchema#integer> .
```

This triple was not accepted, because of the ”(” and ”)” in the property part. During the test with Jena no triples with such brackets have been accepted. This could be caused by a code migration problem, because usually brackets are not allowed at this position in URIs. However, as discussed in Section 5.3, in IRIs additional characters can be used at this position. These characters, which are reserved in a URI but allowed in an IRI, are summarised in Table 3.

5.5 Sesame

Sesame[14] is an open-source triple store implemented in Java. It also supports RDFS inference.

Sesame did fine in the most tests. There were no errors during the tests with percent-encoded triples. When loading the Korean DBpedia data set with IRIs we observed an anomaly when processing IRIs containing the bracket ”{” character. This is not a reserved character in IRIs but also not part of the unreserved characters. This issue has been discussed in Section 5.3 which points out that the specification does not define exactly how to handle these characters.

Table 4. Reserved characters in URIs [6]

!	*	’	(	)	;	:	@	&	=	+	$	,	/	?	#	[	]

Subject	Predicate	Object
res:가위벌과	prop:이름	"가위벌과"@ko
res:가위벌과	prop:색	"동물"@ko
res:가위벌과	prop:그림	"Anthidium manicatum male.jpg"@ko
res:가위벌과	prop:그림설명	"Anthidium manicatum"@ko
res:가위벌과	prop:계	res:동물
res:가위벌과	prop:문	res:절지동물
res:가위벌과	prop:강	res:곤충
res:가위벌과	prop:목	res:벌목 (곤충)
res:가위벌과	prop:아목	res:벌아목

Fig. 3. Sesame with Korean DBpedia data set and IRIs.

5.6 OWL API

The OWL API is an open-source API written in Java for creating, manipulating and serializing OWL ontologies. It includes a parser and writer for RDF/XML,

[14] http://www.openrdf.org/

OWL/XML, and Turtle[15]. Since version 3.0, the OWL API uses the OWL 2 specification. This is a requirement for using IRIs in OWL.

Percent-encoded triples were used for the first test. As expected, the OWL API did not accept these triples resulting in the error message "Illegal Element Name (Element Is Not A QName)". When adding an underscore at problematic triples, the OWL API could serialize the triples. It uses the same strategy as Jena, i.e. usage of an extra namespace for such properties.

OWL API also supports Turtle as output format. When using Turtle, serializing of percent-encoded triples works well. The OWL API failed to load RDF triples with IRIs when non-ASCII character occurred in a resource or property (with exeption "org.xml.sax.SAXParseException: Element or attribute do not match, QName production: QName::=(NCName':')?NCName."). It seems that OWL API was not able to find an appropriate QName even though such a QName does exist and only allowed characters have been used. However, the OWL API was able to serialize triples into XML with IRIs. It uses non-ASCII characters in tags as intended by the XML specification. Unfortunately, XML entities where used in XML tag attributes, as the following example shows:

```
1 <?xml version="1.0"?>
2 ...
3 <owl:NamedIndividual rdf:about="http://ko.dbpedia.org/#M2_
      (&#52380;&#52404;)">
4       <rdf:type rdf:resource="http://ko.dbpedia.org/#&#51060;&#47492;"/>
5       <ko: 별자리 rdf:resource="http://ko.dbpedia.org
            /#&#47932;&#48337;&#51088;&#47532;"/>
6 </owl:NamedIndividual>
7 ...
```

Listing 8. Korean DBpedia in RDF/XML with IRIs by the OWL API.

Table 5. Overview of I18n support in popular tools and APIs.

Tools	Percent-encoding	Underscore workaround	UTF-8/IRI support	Problematic Characters with IRIs	Output formats
OntoWiki 0.9	+	-	◯		1, 2, 3, 5
Protégé 4.1	+	-	◯		1, 2, 3
Jena 2.6.2	+	+	◯	(	1, 2, 4
Virtuoso 6	+	-	+	>	1, 4, 5, 6
Sesame 2.3.1	+	+	+	{	1, 2, 3, 4, 6
OWL API 3	+	+	◯		1, 3

+: Available; ◯ : Available with some restrictions; -: Not available
1: RDF/XML 2: Notation 3: Turtle 4: N-Triples 5: RDF/JSON 6: HTML

[15] `http://owlapi.sourceforge.net/index.html`

6 Conclusions

With the maturing of Semantic Web technologies proper support for internationalization is a crucial issue. This particularly involves the internationalization of resource identifiers, RDF serializations and corresponding tool support. As it was, for example, noted by Richard Cyganiak "the relationships between URIs, IRIs, RDF, XML, UTF-8 etc are incredibly complex"[16]. With this work we aimed at shedding some light on this intricate matter and providing some guidelines for knowledge engineers and tool developers alike. It turned out, that there are serious issues with the two most prevalent building blocks of the Semantic Web, namely URIs and the RDF/XML serialization. With IRIs, some workarounds for RDF/XML or the use of alternative serialization techniques these issues, however, can be circumnavigated. Unfortunately, the semantic web tools landscape is very diverse with regard to support for IRIs, RDF/XML workarounds and alternative serializations. As a result of our in-depth evaluation of prominent tools, it turns out, that an internationalization is possible, when one limits oneself to IRIs, which do not contain a relatively small set of characters commonly causing problems with certain tools or RDF serializations. For the future, it would be desirable, if all RDF serializations could cope with IRIs in a natural way and the tool support would be more homogeneous. However, besides some engineering effort, this will also require some modifications to better align the different standards and specifications.

References

1. Tim Berners-Lee. Notation 3, 1998. See `http://www.w3.org/DesignIssues/Notation3.html`.
2. M. J. Dürst. The properties and promises of UTF-8. In Unicode Consortium, editor, *11th International Unicode Conference*. The Unicode Consortium, 1997. http://www.ifi.unizh.ch/mml/mduerst/papers/PDF/IUC11-UTF-8.pdf.
3. Martin J. Dürst. Internationalized resource identifiers: From specification to testing. In *Proc. of the 19th Internationalization and Unicode Conference*, San Jose, California, September 2001.
4. Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia -a crystallization point for the web of data. *Journal of Web Semantics*, (3):154–165, 2009.
5. M. Duerst and M. Suignard. *Internationalized Resource Identifiers (IRIs)*. Network Working Group, 2005. ftp://ftp.rfc-editor.org/in-notes/rfc3987.txt.
6. T. Berners-Lee and R. Fielding and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. Network Working Group, 2005. ftp://ftp.rfc-editor.org/in-notes/rfc3986.txt.
7. Tim Bray and Jean Paoli and C. M. Sperberg-McQueen and Eve Maler and François Yergeau. Extensible markup language (XML) 1.0 (fifth edition), 2008. http://www.w3.org/TR/REC-xml/.

[16] `http://lists.w3.org/Archives/Public/semantic-web/2009Nov/0122.html`