Effective RDF Resource Identifiers for Integration of Structured Data Sources

Ian Emmons
Raytheon BBN Technologies
Arlington, VA
iemmons@bbn.com

***Abstract*—Based upon extensive experience in the use of semantic technologies to integrate structured data from disparate systems, the author recommends a set of best practices for creating IRIs for RDF resources. Particular attention is paid to avoiding unnecessary coreferences in scenarios where data is drawn from a structured, non-semantic source of record, issues that commonly arise in Department of Defense (DoD), Intelligence Community (IC), and government contracting scenarios, as well as other common pitfalls.**

I. Background

At the very foundation of the Resource Description Framework (RDF), before we can ever write down a single triple, we encounter the notion of the International Resource Identifier (IRI) as a means to create identifiers with global validity.[1] The Web has proven that the IRI is in fact a good solution to this problem, and so the RDF standard has very little further to say about this topic [3]. However, creating sub-optimal IRIs is a common pitfall of the Semantic Web.

In the discussion that follows, an important concept is the *source of record* of a datum. This is the particular copy of the datum that is considered to be its authoritative source. When data is created directly in RDF format, so that the source of record is the RDF itself, then choosing good IRIs is relatively straightforward. However, the source of record for a data set is often a non-RDF database of some sort, such as a relational database, and the data is translated into RDF in order to enable Semantic Web processing techniques. In these cases, there are a number of additional considerations that come into play when choosing resource IRIs, which we address here. Note that a different set of issues arises when choosing IRIs for data whose source of record is unstructured. Such situations are not considered here.

In the sections below, we will first address the issues associated with forming IRIs in general, and then we will consider structured, non-RDF sources of record. We pay particular attention to situations that arise in DoD, IC, and government contracting.

II. Considerations for Any Source of Record

This section discusses issues that apply to data from any kind of source of record.

[1]The IRI [1] is a generalization of the Uniform Resource Identifier (URI) [2] to include international character sets.

A. Hierarchical Naming

The most important aspect of resource identifiers is this: A resource's IRI must be globally unique. In other words, no two resources may share the same identifier. Unlike the primary key of a database table, which need only be unique among the records of that table, a resource's IRI must be globally unique. There are two widely used systems for creating such identifiers: Globally Unique Identifiers (GUIDs), which are sometimes also called Universally Unique Identifiers (UUIDs), and hierarchical naming. The former gathers a number of relatively unique items from the local computing environment, such as the current time and network interface MAC addresses, and combines them algorithmically into a large, fixed-length number whose uniqueness is guaranteed with such high probability that we can assume absolute global uniqueness.

RDF uses hierarchical naming to achieve global uniqueness, as exemplified by the IRI system. This approach constructs an identifier as a variable-length character string consisting of a hierarchy of segments, each of which further narrows the scope until a unique identifier for a specific item is achieved. Each successive level of hierarchy carves out a subset of the namespace denoted by its predecessor and often corresponds to an organizational entity with jurisdiction over that subset. For instance, the segments might be arrayed as follows:

```
http://org/dept/project/class/item
```

The portion preceding the first colon designates the scheme. Most RDF IRIs use the `http:` scheme as shown here, but others are possible as discussed below in Section II-B. The "org" portion is a Domain Name System (DNS) name (see Section II-C). Using a DNS name leverages the domain registration process to reserve a namespace on behalf of an organization. From there, the IRI narrows the scope by appending a department name, a project name, the name of a class of entities, and then finally the identifier of one item within that class.

Naturally, there are many variations on this theme:

- In a very large organization, the "dept" segment may be replaced by several segments that descend through multiple layers of organizational structure. And some organizations may prefer to use a sub-domain of their primary DNS name for this purpose.

- The "project" portion of the IRI may also be subdivided into components.
- Including a date can help with versioning, and it can guard against the possibility that the remainder of the identifier is reproduced at a later time by a different organization that has the same name.
- Many practitioners include a segment immediately after the domain name that is either "id" or "ontology" to distinguish between instance data and its ontology.

There are numerous sources that offer guidelines for constructing such hierarchical names [4, 5, 6, 7], which the reader may wish to consult to gain a deeper understanding of best practices. However, there is no one-size-fits-all strategy, so you will need to adapt the given advice to your particular situation.

As a point of terminology, the portion of the IRI in the example above that precedes "item", including the last slash, is called the *base IRI*. The base may end in a slash, as in this case, or with a hash '#'. For our purposes here, the distinction is immaterial, but a full discussion of the differences can be found in [4].

In order for hierarchical naming to properly achieve its goal of global uniqueness, it is crucial that each RDF author create new IRIs only within those hierarchical scopes in which he or she has the authority to do so. For instance, an author who works for Company A should not create IRIs using the domain name of Company B, unless Company B has given its permission to do so. Otherwise, there exists a very real possibility for different authors to use the same IRI to identify two different things. Likewise, an author within one department of a company should not create IRIs using the department identifier segment of a different department, unless the second department has given its permission to do so.

A different situation arises when one RDF author has already created an IRI for an entity and another author wants an identifier for the same item. In this case, the second author should, whenever possible, reuse the original author's IRI in order to avoid the confusion that arises from having multiple names for the same entity. (See Section III-A below.)

Note that although the RDF standard chose IRIs as its system for unique identification, RDF authors can still use GUIDs, and Section II-B shows how.

B. IRI Schemes

In Section III-A above, it was noted that the IRI scheme most commonly used in RDF is `http:`. While this is true, there are other schemes that work well. But first, why are `http:` IRIs so common? In part the answer is that this scheme possesses a mix of features that make unique identification easy, with extremely low cost. In addition, `http:` IRIs can be resolvable, which means that given appropriately configured infrastructure, the resource identifier can also be used to retrieve information about the resource it identifies. A detailed discussion of how to achieve this goal can be found in [4].

Another IRI scheme that is useful in RDF is the "tag" scheme. The syntax of `tag:` IRIs is given in [8], and an accessible discussion of how to create them can be found in [9]. The `tag:` and `http:` schemes are similar in many ways. The principle ways in which they differ are the following:

- `tag:` IRIs are explicitly non-resolvable.
- The root of a `tag:` IRI may be a domain name, as with `http:`, or an email address.
- The `tag:` scheme formalizes the use of dates in the IRI.

Our original IRI example, translated to the `tag:` scheme, looks like so:

```
tag:org,2014-10-01:dept/project/class/item
```

where "org" is either a DNS name as before or an email address.

IRI schemes other than `http:` and `tag:` are very rare in RDF. One other scheme that might seem useful is the Uniform Resource Name (URN) scheme [10]. However, URN IRIs require the registration of a namespace [11], which makes them far too cumbersome for use in RDF. However, there is a URN namespace already declared for GUIDs [12]. Using this, RDF authors can easily convert a GUID into a valid IRI simply by prepending the string `urn:uuid:`, like so:

```
urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

Of course, a GUID can also be converted into a valid IRI by prepending an `http:` or `tag:` base IRI.

C. DNS Names

This section discusses a number of issues involving the DNS name portion of an IRI so that developers can better avoid them.

Example Domains: Occasionally, an IRI author may not have a DNS name to which he or she can lay claim. Unless the work is truly intended as an example, it is best to avoid using the reserved names `example.org` or `example.com`. IRIs have a tendency to rapidly work their way into many nooks and crannies of a software system (such as queries, source code, or rules), so that the global search-and-replace operation to switch away from `example.org` that seems so easy at the beginning quickly becomes a large undertaking that inconveniences the whole development team. Instead, either acquire a proper domain name or use `tag:` IRIs containing an email address.

Contractors: Contractors who are working on behalf of another organization may be required to use the DNS name of their customer. In such cases, contractors should take care to be sure that the appropriate individuals in the customer organization know that this is happening so that naming collisions do not occur. One easy and effective way that the customer organization can solve such problems is to reserve to the contractor a subset of their IRI space by allocating an IRI segment beyond the DNS name to the contractor's project.

Resolvable IRIs: A popular way to expose semantically encoded data is via the Linked Open Data (LOD) methodology. One of its tenets is that RDF IRIs should always be resolvable, and that when resolved, an IRI should return some information about the concept it identifies. A discussion of best practices for implementing such systems can be found in [4]. A number

of issues can crop up in these cases that developers should be careful to consider:

- Resolvable IRIs require a DNS name that resolves to an actual Web server. In a large enterprise such as the DoD or IC, these tasks may require considerable administrative effort and entail non-trivial approval periods. Some cases may also require security authorizations. Thus, developers must establish DNS names and servers as early as possible in order to avoid changing IRIs mid-project.
- Be sure to check early in your project whether organizational security policies require that web traffic runs over secure connections. This is a frequent requirement on DoD and IC projects. In such cases, your IRIs will need to use the `https:` scheme.
- DoD and IC projects that result in an operational system typically must establish development and test deployments in addition to the operational system itself. These will necessarily have different DNS names, resulting in separate IRI spaces. Queries, source code, and rules, however, will by default contain a single DNS name, and will therefore break when moved from development to test unless a more sophisticated approach is devised. One partial solution is to segregate the ontologies on a separate server that is common to all of the deployed systems. Since the ontologies are typically read-only static content, this server can be an incredibly simple deployment that never changes. Since many of the IRIs that appear in queries, source code, and rules are taken from the ontologies, this will substantially decrease the number of IRIs that break when moving between deployments.
- Classified systems must sometimes run on multiple, isolated networks. However, the entities that are identified on the lower networks are typically also present on the upper ones. Thus, the IRIs that identify these entities should be the same in all cases. In an LOD scenario, this means that corresponding servers on different networks should have the same DNS name.

D. Allowed Characters

One confusing point about RDF IRIs is that the portion after the base IRI is allowed to contain a broad range of characters directly and nearly any character in an escaped (or percent-encoded) form. However, as a practical matter, the characters allowed in this portion of the IRI are limited to the following:

- The first character must be a letter or underscore, and
- Subsequent characters must be letters, underscores, hyphens, periods, or digits.

The reason for this is that in many contexts the base IRI is assigned a short *prefix,* which is used to abbreviate. For instance,

```
http://org/dept/project/class/item
```

can be written more concisely as `p:item` in a document that declares a prefix like so:

```
prefix p: <http://org/dept/project/class/>
```

This prefix syntax is borrowed from eXtensible Markup Language (XML) namespaces, where the abbreviated form is called a *QName* (qualified name), and the portion following the prefix is called a *local name.* The XML syntax for a local name is much more restrictive than the IRI syntax, as noted above. Thus, for reasons of convenience, most RDF authors limit themselves to the more restrictive syntax so that they can use prefixes to abbreviate their IRIs. This is particularly helpful for making SPARQL Protocol and RDF Query Language (SPARQL)[2] queries readable [13], but it also helps when data is viewed in Turtle (`.ttl`) syntax [14].

E. IRIs as Content

While forming IRIs, it is natural and proper to put some thought into their information content. However, at query time, IRIs should be considered to be opaque identifiers devoid of content other than their ability to uniquely identify a resource. The reason for this is that semantic query languages and query processors are poorly adapted to efficiently parsing out pieces of IRIs. Therefore, if there is information embedded in your IRIs that you wish to access from queries, duplicate that information in properties designed for that purpose.

III. IRIs for Non-RDF Sources of Record

Over the last decade, Semantic Web technologies have proven to be an effective approach for solving data integration problems. Usually this involves creating an RDF representation of data from an existing, non-semantic source of record. The RDF may then be stored in, for example, a triple store such as Raytheon BBN Technologies's (BBN's) Parliament™ [15],[3] or it may be generated on demand and fed dynamically to the requesting activity, as with a federated query system like BBN's Asio™ [16].

Whether stored or generated on demand, the end result is that the same data is represented in multiple formats and places, and in such situations the proper creation and maintenance of identifiers is crucial. Our goal in such cases is always to maintain a one-to-one relationship between entities in the source of record (which may be any kind of identifier) and their identifiers in other representations (which are always IRIs in our case). In this section, we consider how best to accomplish this goal in general, as well as investigate a number of specific examples.

A. Uniqueness and Reproducibility

To achieve the goal given above, we put forth two guiding principles for the formation of IRIs:

- *Unique:* A resource's IRI must be globally unique, as discussed above in Section II-A.
- *Reproducible:* Every time we form the RDF representation of an entity, the IRI we create must be same one.

[2]The SPARQL acronym was chosen during a time when recursive acronyms were in vogue. As a result, the expansion of the acronym is confusing because it contains the acronym itself.

[3]http://parliament.semwebcentral.org/

These two tenets correspond to the two halves of the one-to-one constraint that is our goal. The uniqueness principle says that no two data items may share the same identifier, and the reproducibility principle says that no data item may have two distinct identifiers.

Most people understand the uniqueness principle easily, both the need for it and how to achieve it. The reproducibility principle, however, is more subtle: If we encounter the same entity twice, forming its RDF data representation each time, then we must form the same IRI for it both times. But why is this important?

Consider an example system that queries a Relational Database Management System (RDBMS) and then translates the result set into RDF. A straightforward way to do this is to create an IRI to represent each row (or, more precisely, to represent the real-world entity represented by the row), and then transform each column into a property of that resource. Foreign key columns become object properties, and other columns translate into datatype properties. So if we issue the following query:

```
select employee_id, first_name, last_name,
  ssn from Employee where employee_id < 40
```

and the result set looks like Table I, then the resulting RDF

employee_id	first_name	last_name	ssn
12	Robert	Smith	123-45-6789
37	Alice	Jones	987-65-4321

TABLE I
EXAMPLE RESULT SET

might look like so:

```
prefix ont: <http://example.org/ont#>
prefix id: <http://example.org/id#>
id:z138ce39f-0434-4d16-b307-82b9206142b5
  a ont:Employee ;
  ont:employeeId 12 ;
  ont:firstName "Robert" ;
  ont:lastName "Smith" ;
  ont:ssn "123-45-6789" .
id:z1e036a52-7e1e-4a33-a48f-03837634f776
  a ont:Employee ;
  ont:employeeId 37 ;
  ont:firstName "Alice" ;
  ont:lastName "Jones" ;
  ont:ssn "987-65-4321" .
```

Note that the IRIs for the two employee instances are based on GUIDs.

Now, suppose that later on in the execution of this same system, another query against the Employee table happens to return row 37 again. Then the same code will translate the new result set into RDF. When it returns to row 37, it will get a GUID just as it always does, generating something like this:

```
id:zf4139560-8c48-4b4c-a860-5d1bb9e02bdf
  a ont:Employee ;
  ont:employeeId 37 ;
  ont:firstName "Alice" ;
  ont:lastName "Jones" ;
  ont:ssn "987-65-4321" .
```

This is exactly the same RDF as before, except that by the nature of GUIDs, a different IRI is now representing employee 37. The end result is that in RDF, we now have two separate employees named Alice Jones with employee number 37. In other words, we have created a coreference where none existed before. This is exactly what the reproducibility principle seeks to avoid — unnecessary coreferences — and it also illustrates why GUIDs are best avoided in RDF resources under most circumstances. (For an exception, see Section III-B.)

One way to avoid the coreference created in this scenario is to identify employee 37 by the IRI `id:employee37`. Using the primary key from the RDBMS table ensures that every time we encounter row 37, we will form the same IRI. But what if we encounter Alice Jones in a different context, say in an RDBMS table at the Internal Revenue Service (IRS)? Now we cannot expect that Alice will be associated with the number 37, because that number is an internal implementation detail of her employer's database. In order to avoid creating a coreference in this case, we might turn to Alice's Social Security Number (SSN). This information will almost certainly be in both databases, because both the IRS and Alice's employer are concerned about her income tax. Thus if we identify Alice by the IRI `id:ssn-987-65-4321`, we can be sure that Alice will be a single, unified entity across these two separate organizations.

Generalizing from this particular example, we see that to comply with the reproducibility principle, the information used to form a resource's IRI should be semantically intrinsic to the thing being identified. Ideally, this should hold true whether we encounter the entity in a repeat of the original context (e.g., the same source of record) or in a different context altogether (such as a source of record in a different organization).

Clearly, if we encounter the entity in two *very* different contexts, we may discover that there is no identifying information held in common. The scenario above was cleverly constructed so that a solution satisfying the reproducibility principle was readily available. However, if we add a third database to this scenario from the immigration agency of a non-U.S. government, then the record of Alice's visit while on vacation will almost certainly not contain her SSN.

Thus, it is important to realize that while the uniqueness principle is of paramount importance, never to be violated, the reproducibility principle is really much more of a guideline to strive for, but which usually requires some carefully chosen compromises.

B. Example Scenarios

In this section, we seek to explore a variety of data integration scenarios to see how the uniqueness and reproducibility principles can best be achieved.

Derivation from a single source of record: In cases where the RDF will always be derived from a consistent source of record, we look to the identification system used in the source of record itself. One common case is a database table that uses a sequence number for its primary key. In this case, we can use the primary key itself to form the resource identifier, but as with the example given in Section III-A, this will cause a coreference whenever the same real-world entity is encountered in a different source of record. Thus, if the table contains a semantic key, i.e., a key that has semantic meaning intrinsic to the entity represented by the table, that may be preferable.

Note that a key, and particularly a semantic key, can span several columns. In such cases, the individual values from these columns must be combined to form the IRI. Section III-C shows one way to do this correctly and without producing long and unwieldy IRIs.

Occasionally, you may find a database table whose primary key is a GUID column. (This happens most often with Microsoft SQL Server.) This is one occasion when using a GUID in your IRIs is a reasonable thing to do, because the GUIDs come from the source of record, rather than being generated anew every time the IRI is created.

Derivation from multiple sources of record: Sometimes there are multiple sources of record that contain overlapping data sets. If the sources of record were built within the same organization, they may have a common identifier system that can be leveraged to create consistent IRIs across the sources of record. Alternately, the entities in question may have a well-known standard system of identification, such as airplane tail numbers or merchant ship registration, that is included in all of the sources of record. However, in the general case of multiple sources of record that were built independently, the sources of record will not have a common system of unique identifiers. In such cases, it may be necessary to encode data from each source of record in isolation, resulting in potential coreferences, and then apply a coreference resolution algorithm to identify and merge the coreferences after the fact. Such algorithms are beyond the scope of this paper.

Flat file sources of record: When a flat file is nicely designed, it presents no issues that are not covered by the scenarios discussed above. Unfortunately, flat files are often more ad hoc than databases, with little thought given by their creators to identification of the entities contained within. As a result, it is common when translating a flat file into RDF to discover that there is no column that serves the purpose of identifying the row.

If possible, try to identify a subset of the columns in the file that uniquely identify the entity represented by the row. In particularly difficult cases, the author has resorted to regarding all of the columns as the key. This approach will tend to create coreferences, but it has the best chance of satisfying the uniqueness principle.

An approach that is not recommended is to use the flat file's name and/or path to form part of the IRI for each row. This is usually not a good idea because the file name and location can be changed without any change to the file content, and therefore without a change to the semantics of the entities represented therein. In other words, every time the file is moved or renamed, there is potential to create coreferences for all of the contained entities.

Derivation from a single source of record, with intermediate processing: Consider a case where there is a single source of record, like a database, that feeds data through some process (or set of processes), transforming or enriching the data on the way, and then we wish to render the final output as RDF. By far the best approach to forming IRIs in such a case is to find the identifying information from the original source of record and make sure that this information is carried throughout the processing chains. This enables the IRIs to be constructed independent of the particular processing steps, and it also allows a consistent IRI for an entity that passes through multiple processing chains, thereby avoiding unnecessary coreferences.

Sub-row entities: In simple cases, each row in a database table translates into one RDF entity, as outlined in Section III-A. However, ontologies are often more structured than database schemas, and so what appears as just more columns in a database table may well be a separate entity in your ontology. Thus, it is often the case that one database row translates into multiple related resources, each with its own properties, in RDF.

Some sub-row entities are logically part of the entity represented by the row in which they occur. In such cases, a handy way to form the IRI is to use the IRI for the row entity and then append additional key fields to distinguish it from the row entity.

Other sub-row entities are logically independent of the entity represented by the row in which they occur. An easy way to distinguish whether a sub-row entity falls into this category is to ask yourself the following question: "If the columns containing the sub-row entity in two rows contain the same values, should the end result be two row entities related to one sub-row entity?" If the answer is yes, then the sub-row entity is logically independent. In this case, you will want to form the IRI for the sub-row entity from only those columns containing the sub-row entity much as if they were a row in a separate table.

C. Avoiding Overly Long IRIs

As indicated in the above example scenarios, sometimes many individual pieces of information must be combined into a single IRI. This can result in an enormously long IRI, and it can also result in arbitrary characters that must be escaped. A handy way to avoid this situation is to concatenate the individual strings and then run the result through a cryptographic message digest (or hash) algorithm. Sample Java code for this procedure is shown in Figure 1.

There are a couple of subtleties to this procedure that require some explanation. First, the use of a hash weakens the uniqueness guarantee. However, cryptographic hash algorithms are designed specifically to avoid collisions, and so the probability

```
static String encode(List<String> keys)
{
  StringBuilder buf = new StringBuilder();
  boolean isFirstKey = true;
  for (String key : keys)
  {
    if (!isFirstKey)
    {
      buf.append(',');
    }
    if (key.contains("\""))
    {
      buf.append('\"');
      buf.append(
        key.replace("\"", "\"\""));
      buf.append('\"');
    }
    else if (key.contains(",")
      || key.contains("\r")
      || key.contains("\n"))
    {
      buf.append('\"');
      buf.append(key);
      buf.append('\"');
    }
    else
    {
      buf.append(key);
    }
    isFirstKey = false;
  }
  try
  {
    MessageDigest md = MessageDigest
      .getInstance("SHA-256");
    byte[] input = buf.toString()
      .getBytes("UTF-8");
    HexBinaryAdapter hba
      = new HexBinaryAdapter();
    return hba.marshal(md.digest(input));
  }
  catch (NoSuchAlgorithmException
    | UnsupportedEncodingException e)
  {
    // This should never happen, because
    // all JVMs must support the SHA-256
    // hash and UTF-8 char encoding.
    throw new RuntimeException(e);
  }
}
```

Fig. 1. Hashing a List of Strings for Inclusion in an IRI

of this procedure causing a collision of IRIs is vanishingly small when a strong algorithm such as SHA-256 is used.

Second, a quick examination of the code in Figure 1 reveals that the individual pieces of key material are not simply concatenated, but rather encoded as if they were a row within a Comma-Separated Values (CSV) file. The reason for this is that string concatenation is not an invertible operation. For instance, if we concatenate "their reversible", we get exactly the same result as if we concatenate "the irreversible". However, when we CSV-encode these two pairs of strings, we get "their,reversible" and "the,irreversible", which are distinct. Thus, the use of CSV encoding upholds the uniqueness principle by assuring that two distinct keys are not mapped to a single IRI.

IV. Conclusion

Though the IRI lies at the heart of the RDF standard, creating IRIs for RDF resources is a topic that is often glossed over in the literature. In this treatment, we hope to have given the reader a solid understanding of the issues underlying the creation of effective IRIs, as well as specific advice for a range of scenarios relating to structured, non-RDF sources of record as well as situations that arise in DoD, IC, and government contracting.

V. Glossary

Asio	Asio™ is BBN's semantic federated query framework. This is not an acronym. It is simply a name derived from a genus of owls. (p. 3)
BBN	Raytheon BBN Technologies, Inc. (p. 3)
CSV	Comma-Separated Values (p. 6)
DNS	Domain Name System (pp. 1–3)
DoD	Department of Defense (pp. 1, 3, 6)
GUID	Globally Unique Identifier (pp. 1, 2, 4, 5)
IC	Intelligence Community (pp. 1, 3, 6)
IRI	International Resource Identifier (pp. 1–6)
IRS	Internal Revenue Service (p. 4)
LOD	Linked Open Data (pp. 2, 3)
Parliament	Parliament™ is BBN's triple store, so named because "parliament" is the collective noun for a group of owls. A triple store is a specialized database tuned to the unique needs of the Semantic Web data representation. (p. 3)
RDBMS	Relational Database Management System (p. 4)
RDF	Resource Description Framework (pp. 1–6)
SPARQL	SPARQL Protocol and RDF Query Language. This acronym is a bit confusing, because it was conceived when recursive acronyms were popular. (p. 3)
SSN	Social Security Number (p. 4)
URI	Uniform Resource Identifier (p. 1)
URN	Uniform Resource Name (p. 2)
UUID	Universally Unique Identifier (p. 1)
XML	eXtensible Markup Language (p. 3)

VI. References

[1] M. Düerst and M. Suignard, "Internationalized Resource Identifiers (IRIs)," IETF, Request for Comments 3987, Jan. 2005. [Online]. Available: http://tools.ietf.org/html/rfc3987 (cit. on p. 1).

[2] T. Berners-Lee, R. T. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," IETF, Request for Comments 3986, Jan. 2005. [Online]. Available: http://tools.ietf.org/html/rfc3986 (cit. on p. 1).

[3] G. Schreiber, Y. Raimond, F. Manola, E. Miller, and B. McBride, "RDF 1.1 Primer," W3C, Working Group Note, version 1.1, Jun. 24, 2014. [Online]. Available: http://www.w3.org/TR/rdf-primer/ (cit. on p. 1).

[4] L. Sauermann, R. Cyganiak, D. Ayers, and M. Völkel, "Cool URIs for the Semantic Web," W3C, Interest Group Note, Mar. 31, 2008. [Online]. Available: http://www.w3.org/TR/cooluris/ (cit. on p. 2).

[5] "223 Best Practices URI Construction," W3C, Wiki, Mar. 14, 2012. [Online]. Available: http://www.w3.org/2011/gld/wiki/223_Best_Practices_URI_Construction (cit. on p. 2).

[6] P. Bryant, "REST-ful URI design," 2PartsMagic Blog, May 30, 2012. [Online]. Available: http://blog.2partsmagic.com/restful-uri-design/ (cit. on p. 2).

[7] M. T. C. Benitez, "Best Practice for Web Data URI," W3C, Editor's Draft, Jun. 10, 2014. [Online]. Available: http://dragoman.org/duri/ed-1.html (cit. on p. 2).

[8] T. Kindberg and S. Hawke, "The 'tag' URI Scheme," World Wide Web Consortium, Request for Comments 4151, Oct. 2005. [Online]. Available: http://tools.ietf.org/html/rfc4151 (cit. on p. 2).

[9] ——, (Jul. 9, 2008). Tag URI, [Online]. Available: http://www.taguri.org (cit. on p. 2).

[10] R. Moats, "URN Syntax," IETF, Request for Comments 2141, May 1997. [Online]. Available: http://tools.ietf.org/html/rfc2141 (cit. on p. 2).

[11] L. L. Daigle, D.-W. van Gulik, R. Iannella, and P. Faltstrom, "URN Namespace Definition Mechanisms," IETF, Request for Comments 2611, Jun. 1999. [Online]. Available: http://tools.ietf.org/html/rfc2611 (cit. on p. 2).

[12] P. J. Leach, M. Mealling, and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," IETF, Request for Comments 4122, Jul. 2005. [Online]. Available: http://tools.ietf.org/html/rfc4122 (cit. on p. 2).

[13] S. Harris, A. Seaborne, and E. Prud'hommeaux, "SPARQL 1.1 Query Language," W3C, Recommendation, version 1.1, Mar. 21, 2013. [Online]. Available: http://www.w3.org/TR/sparql11-query/ (cit. on p. 3).

[14] E. Prud'hommeaux, G. Carothers, D. Beckett, and T. Berners-Lee, "RDF 1.1 Turtle, Terse RDF Triple Language," W3C, Recommendation, version 1.1, Feb. 25, 2014. [Online]. Available: http://www.w3.org/TR/turtle/ (cit. on p. 3).

[15] D. Kolas, I. Emmons, and M. Dean, "Efficient Linked-List RDF Indexing in Parliament," in *Proceedings of the Fifth International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2009)*, ser. Lecture Notes in Computer Science, vol. 5823, Washington, DC: Springer, Oct. 2009, pp. 17–32. [Online]. Available: http://ceur-ws.org/Vol-517/ (cit. on p. 3).

[16] D. Kolas, "Query Rewriting for Semantic Web Information Integration," in *Proceedings of the Sixth International Workshop on Information Integration on the Web (IIWeb-07), at the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, Vancouver, Canada, Jul. 2007. [Online]. Available: http://www.aaai.org/Papers/Workshops/2007/WS-07-14/WS07-14-008.pdf (cit. on p. 3).