Umaka Score

March 31, 2016

Abstract

Umaka Scores represents how valuable endpoints are. Umaka Score is calculated on the basis of the evaluation from the 6 aspects, Availability, Freshness, Operation, Usefulness, Validity and Performance. We also rank the endpoints on a scale of A to E according to the Umaka score.

In this document, we show how to calculate Umaka score.

1 Umaka Score

Umaka Score represents how valuable endpoints are. We believe there are six aspects, Availability, Freshness, Operation, Usefulness, Validity and Performance, for valuable endpoints. We evaluate and score endpoints from these aspects. Then Umaka Score is average of these score:

$$\text{Umaka Score} = \frac{\sum_{aspects} score}{6}$$

where

$$\text{aspects} = [\ \text{Availability}, \ \text{Freshness}, \ \text{Operation}, \ \text{Usefulness}, \ \text{Validity}, \ \text{Performance}]$$

We rank the endpoint as shown in Table 1

In Section 2, we show how to score endpoints from each aspect.

Umaka Score	Umaka rank
81 - 100	A
61 - 80	B
41 - 60	C
21 - 40	D
0 - 20	E

Table 1: Umaka Rank

2 Metrics of Umaka Score

2.1 Availability

Availability represents the degree of ready for use. High availability value means we can access the endpoint most of all the time. Low availability value means the endpoint is often down. We measure the following metrics for availability:

- Alive

  We send HTTP request to the endpoint URI daily. If the endpoint return 200 HTTP response, alive is true, otherwise false. Note that we assume the endpoint as dead when the endpoint returns 302 Found.

- Alive Rate

  Alive rate is the percentage of alive in 30 days.

The Availability score is calculated as:

Availability = Alive Rate

2.2 Freshness

Freshness represents how often data in the endpoint is updated. We measure the following metrics for freshness:

- Last Updated

  We retrieve Service Description and VoID and get the literals specified by dcterms:modified. Then we assume the last updated as the latest date among those literals.

  Unfortunately, most of all endpoints provide Service Description and VoID without dcterms:modified statement.

  Thus, we determine the data modification thought the following adhoc procedure:

  First, we retrieve a number of statements using a query described in Listing 1 and compare the number with the previous one. If the number of statements has changed, we assume the endpoint seems to be updated today.

Listing 1: A query for retrieving a number of statements

```
SELECT COUNT(*) AS ?c
WHERE {?s ?p ?o}
```

When the number of statements is not changed, we retrieve the first and the last statements using queries described in Listing 2 and 3. Then we compare them with those of a previous day. If one of them is different, we assume the endpoint seems to be updated today.

Listing 2: A query for the first statement

```
SELECT *
WHERE {?s ?p ?o}
OFFSET 0 LIMIT 1
```

Listing 3: A query for the last statement

```
SELECT *
WHERE {?s ?p ?o}
OFFSET ($COUNT - 1) LIMIT 1
```

Note that $COUNT represents the number of total statements retrieved by Listing 1.

Note that Virtuoso has trouble to count the number of statements. So this adhoc procedure does not work well for the endpoint based on Virtuoso.

- Update Interval

  Update Interval is average of the interval between last updated. Update Interval is N/A if there are less than two last updated dates for the endpoint.

Even though we would like to score freshness on the basis of the Update Interval, we give up to score and set 50 for all endpoints since the adhoc approach does not work well for Virtuoso.

Freshness = 50

2.3 Operation

Operation represents the degree of the maintenance. We send HTTP request to the endpoint URI with using the accept request-header field to specify both Turtle and RDF/XML, and validate the format of its response. We measure the two metrics:

- Service Description

  true if Service Description can be retrieved in Turtle format or RDF/XML format, otherwise false. We access the endpoint URI withe following HTTP Request Header:

```
Accept: text/turtle,application/rdf+xml
```

- VoID

  true if VoID can be retrieved from well-known URI[1] in Turtle format or RDFXML format, otherwise false. We access the endpoint URI withe following HTTP Request Header:

```
Accept: text/turtle,application/rdf+xml
```

We calculate Operation score as follows:

$$\text{Operation} = \begin{cases} 0 & \text{if both of them are false} \\ 50 & \text{if one of them is false} \\ 100 & \text{if both of them are true} \end{cases}$$

2.4 Usefulness

Usefulness represents the degree how easily we can link data in the endpoint. We measure the three metrics:

- Metadata Score

  Metadata Score represents how much the endpoint contains metadata defined in [3].

  First we retrieve a list of graphs in the endpoint using a query described in Listing 4.

Listing 4: Obtain graph URIs on a SPARQL endpoint

```
SELECT DISTINCT ?g
WHERE{
 GRAPH ?g{ ?s ?p ?o.}
}
```

  Then we try to retrieve the metadata for each graph except for Table 2 as follows:

Graph URI
`http://www.openlinksw.com/schemas/virtrdf#`

Table 2: List of Ignore Graphs

1. Classes

   We retrieve a list of classes using a query described in Listing 5 and 6.

Listing 5: Obtain the classes on a graph g

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
    ns#>
SELECT DISTINCT ?c
FROM <g>
WHERE {
  { ?c rdf:type rdfs:Class. }
  UNION
  { [] rdf:type ?c. }
  UNION
  { [] rdfs:domain ?c. }
  UNION
  { [] rdfs:range ?c. }
  UNION
  { ?c rdfs:subclassOf []. }
  UNION
  { [] rdfs:subclassOf ?c. }
}
LIMIT 100
```

Listing 6: Obtain the classes having instances on a graph g

```
PREFIX rdf:
SELECT DISTINCT ?c
        FROM <g>
WHERE{
        [] rdf:type ?c.
}
```

2. Labels

   We retrieve a list of labels using a query described in Listing 7.

Listing 7: Obtain labels of the classes c1 c2 ... cn from a graph g

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?c ?label
WHERE {
    graph <g> {
      ?c rdfs:label ?label.
      filter (
        ?c IN (<c1>, <c2>, ..., <cn>)
      )
    }
}
```

3. Datatypes

   We retrieve a list of datatypes using a query described in Listing 8.

Listing 8: Obtain the datatypes on a graph g

```
SELECT DISTINCT (datatype(?o) AS ?ldt)
```

```
FROM <g>
WHERE{
  [] ?p ?o.
  FILTER(isLiteral(?o))
}
```

4. Properties

   We retrieve a list of properties using a query described in Listing 9.

   Listing 9: Obtain the properties on a graph g

```
SELECT DISTINCT ?p
        FROM <g>
WHERE{
        ?s ?p ?o.
}
```

We evaluate Metadata score as follows:

$$\text{Metadata Score} = \frac{\sum_{graphs}^{g} (c(g) + l(g) + p(g) + d(g))}{N}$$

where

N = Number of Graphs

$$c(g) = \begin{cases} 0 & \text{if g does not contains any classes} \\ 25 & \text{if g contains more than zero classes} \end{cases}$$

$$l(g) = \begin{cases} 0 & \text{if g does not contains any labels} \\ 25 & \text{if g contains more than zero labels} \end{cases}$$

$$p(g) = \begin{cases} 0 & \text{if g does not contains any properties} \\ 25 & \text{if g contains more than zero properties} \end{cases}$$

$$d(g) = \begin{cases} 0 & \text{if g does not contains any datatypes} \\ 25 & \text{if g contains more than zero datatypes} \end{cases}$$

- Vocabulary Score

  Vocabulary Score, which is calculated based on metadata, represents how many vocabularies data in the endpoint use.

  Vocabulary Score is calculated as follows:

$$\text{Vocabulary Score} = \frac{\sum_{graphs}^{g} v(g)}{N}$$

where

$$N = \text{Number of Graphs}$$
$$v(g) = \text{Number of Properties in Graph g}$$

- Ontology Score

  Ontology Score, which is calculated based on metadata, represents how much common ontologies data in the endpoint use.

  Ontology Score is calculated as follows:

$$\text{Vocabulary Score} = \frac{\sum_{graphs}^{g} o(g)}{N}$$
where
$$N = \text{Number of Graphs}$$
$$o(g) = \frac{NCO}{NO}$$
$$NO = \text{Number of Ontologies used for Properties}$$
$$NCO = \text{Number of Ontologies used for Properties in Table 3}$$

Ontology URI
http://www.w3.org/2000/01/rdf-schema
http://www.w3.org/1999/02/22-rdf-syntax-ns
http://www.socrata.com/rdf/terms
http://www.w3.org/2003/01/geo/wgs84_pos
http://xmlns.com/foaf/0.1/
http://www.w3.org/2002/07/owl
http://purl.org/dc/elements/1.1/
http://purl.org/dc/terms/
http://www.w3.org/2000/10/swap/pim/usps
http://dublincore.org/documents/dcmi-box/
http://www.territorio.provincia.tn.it/geodati/ontology/
http://www.w3.org/2004/02/skos/core

Table 3: List of Common Ontologies

At last, we evaluate Usefulness Score as follows:

$$\begin{aligned} \text{Usefulness} &= 30.0 * \text{Metadata Score} \\ &+ 40.0 * f10(\text{Vocabulary Score}) \\ &+ 30.0 * \text{Ontology Score} \end{aligned}$$
where

$$f10(x) = \begin{cases} 10 & \text{if } x > 10 \\ x & \text{Otherwise} \end{cases}$$

2.5 Validity

Validity represents how endpoint and data in it obey the rules. We measure the two metrics:

- Cool URI

  The URI of endpoints is preferred to be Cool URI[5], [4].

  We check four criteria:

  1. A host of URI of endpoints should not be specified by IP address
  2. A port of URI of endpoints should be 80
  3. A URI of endpoints should not contain query parameters
  4. A length of URI of endpoints should be less than 30 characters

  Cool URI Score is a percentage of the satisfied rules.

- Linked Data Rule

  The endpoints are preferred to be satisfied with the four rules of linked data[2].

  We check four criteria:

  1. Use URIs as names for things

     We assume all subjects of statements are things. We search invalid statement using a query described in Listing 10, and if nothing is found the endpoint satisfied this rule.

     Note that we ignore Virtuoso specific graphs since Virtuoso contains a graph which contains invalid statements.

     Listing 10: A Query for searching non-URI subjects

```
SELECT
  *
WHERE {
GRAPH ?g { ?s ?p ?o } .
  filter (!isURI(?s) && !isBLANK(?s) && ?g NOT IN (
    <http://www.openlinksw.com/schemas/virtrdf#>
  ))
}
LIMIT 1
```

2. Use HTTP URIs so that people can look up those names

   We assume all subjects of statements are things. We search invalid statement using a query described in Listing 11, and if nothing is found the endpoint satisfied this rule.

   Note that we ignore Virtuoso specific graphs since Virtuoso contains a graph which contains invalid statements.

Listing 11: A Query for searching non-HTTP-URI subjects

```
SELECT
  *
WHERE {
  GRAPH ?g { ?s ?p ?o } .
  filter (!regex(?s, "http://", "i") && !isBLANK(?s)
      && ?g NOT IN (
    <http://www.openlinksw.com/schemas/virtrdf#>
  ))
}
LIMIT 1
```

3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)

   We assess this rule by obtaining a subject (URI) using a query described in Listing 12 and accessing the URI via HTTP protocol. We assume that the endpoint is satisfied with the rule if the URI returns any data.

   Note that we ignore Virtuoso specific graphs since Virtuoso contains a graph which contains invalid statements.

Listing 12: A Query for a Subject

```
SELECT
  ?s
WHERE {
  GRAPH ?g { ?s ?p ?o } .
  filter (isURI(?s) && ?g NOT IN (
    <http://www.openlinksw.com/schemas/virtrdf#>
  ))
}
LIMIT 1
OFFSET 100
```

4. Include links to other URIs. so that they can discover more things

   We assume the statement representing the link to other URI uses the vocabularies owl:sameAs or rdfs:seeAlso. We think if there are any statement of which property is owl:sameAs or rdfs:seeAlso, the endpoint is satisfied with the rule. Thus we check the feasibility of the rule by using queries described in Listing 13, 14.

Listing 13: A Query for a Same AS Statement

```
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT
  *
WHERE {
  GRAPH ?g { ?s owl:sameAs ?o } .
}
LIMIT 1
```

Listing 14: A Query for a See Also Statement

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT
  *
WHERE {
  GRAPH ?g { ?s rdfs:seeAlso ?o } .
}
LIMIT 1
```

Linked Data Score is a percentage of the satisfied rules.

We evaluate Validity as follows:

Validity = 40 * Cool URI Score + 60.0 * Linked Data Rule Score

2.6 Performance

Performace suggests how powerful the endpoint is.

We measure the response times of the two queries, Listing 15, 16. The former query is a most simple query and we use this query to estimate the transfer time. The latter query requires a little computations for endpoints. We believe the execution cost of this query does not differ very much according to the size of data.

Listing 15: A Most Simple Query

```
ASK {}
```

Listing 16: A Query for Listing Graphs

```
SELECT DISTINCT
  ?g
WHERE {
  GRAPH ?g { ?s ?p ?o }
}
```

We assume the execution time as:

Execution Time = Differences of the response time for those queries.

After that, we evaluate Performance as:

$$\text{Performance} = \begin{cases} 100.0 * (1.0 - \text{Execution Time}) \\ \quad \text{if Execution Time is less than 1 second} \\ 0 \quad \text{Otherwise} \end{cases}$$

References

[1] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets with the void vocabulary. https://www.w3.org/TR/void/, March 2011.

[2] Tim Berners-Lee. Linked data - design issues. https://www.w3.org/DesignIssues/LinkedData.html, 2006.

[3] DBCLS. Sparql queries for sparql builder metadata. http://www.sparqlbuilder.org/doc/sparql-queries-for-sparql-builder-metadata/.

[4] Leigh Dodds and Ian Davis. Linked data patterns - a pattern catalogue for modelling, publishing, and consuming linked data. http://patterns.dataincubator.org, 2012.

[5] Leo Sauermann and Richard Cyganiak. Cool uris for the semantic web. https://www.w3.org/TR/cooluris/, Descember 2008.