QROWD - Because Big Data Integration is Humanly Possible
Innovation Action

Grant agreement no.: 732194

D7.2 – Data storage and access component

<table>
<thead>
<tr>
<th>Due Date</th>
<th>30 Nov 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Delivery Date</td>
<td>30 Nov 2017</td>
</tr>
</tbody>
</table>
| Document Author/s                          | Semih Yumuşak (AI4BD)  
|                                 | Martin Voigt (AI4BD)   
|                                 | Daniel Hladky (AI4BD)  |
| Version          | 0.5         |
| Dissemination level | PU         |
| Status           | FINAL       |
| Document approved by | Luis-Daniel Ibáñez |
# TABLE OF CONTENT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTIVE SUMMARY</td>
<td>4</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>5</td>
</tr>
<tr>
<td>2 TECHNICAL DESCRIPTION</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Entity Name Service:</td>
<td>5</td>
</tr>
<tr>
<td>2.2Crowdsourcing-enabled Entity Name Service</td>
<td>8</td>
</tr>
<tr>
<td>2.3 API definition</td>
<td>9</td>
</tr>
<tr>
<td>2.3.1 GET URLs for Entities</td>
<td>9</td>
</tr>
<tr>
<td>2.3.2 GET Disambiguation Tasks</td>
<td>10</td>
</tr>
<tr>
<td>2.3.3 Resolve Disambiguation Tasks</td>
<td>11</td>
</tr>
<tr>
<td>2.3.4 GET Entity Details</td>
<td>12</td>
</tr>
<tr>
<td>2.3.5 Management API</td>
<td>14</td>
</tr>
<tr>
<td>3 BIKE RACK USE-CASE FOR ENTITY DISAMBIGUATION...</td>
<td>15</td>
</tr>
</tbody>
</table>
D7.2 Data storage and access component

### History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>23.10.2017</td>
<td>Initial version</td>
<td>Semih Yumuşak (AI4BD)</td>
</tr>
<tr>
<td>0.1</td>
<td>31.10.2017</td>
<td>ENS service descriptions and API, Use-case description</td>
<td>Semih Yumuşak (AI4BD)</td>
</tr>
<tr>
<td>0.2</td>
<td>02.11.2017</td>
<td>Enhancements</td>
<td>Semih Yumuşak (AI4BD)</td>
</tr>
<tr>
<td>0.3</td>
<td>20.11.2017</td>
<td>Corrections based on internal review</td>
<td>Semih Yumuşak (AI4BD)</td>
</tr>
<tr>
<td>0.4</td>
<td>21.11.2017</td>
<td>Corrections based on internal review</td>
<td>Semih Yumuşak (AI4BD)</td>
</tr>
<tr>
<td>0.5</td>
<td>29.11.2017</td>
<td>Executive summary</td>
<td>Semih Yumuşak (AI4BD)</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

This demonstration document includes the application and services created to enable a crowdsourced curation for disambiguating named entities and validation of uniqueness in the storage. One of the main challenge addressed in this document is to solve the problem of unique URI within T7.2 by enabling new services using state of the art machine learning methods and enhancing those with HIT. Due to the complexity of unique URI, an algorithm can’t always automate such process therefore HIT involvements is needed. Therefore, the service is enhanced with various interfaces. The main output of this deliverable is the crowdsourcing enabled unique URI service (namely crowdsourcing-enabled ENS), which provides a disambiguated and validated URI for an entity (which may include spatial information as well as textual and visual properties). The sample service requests listed in this document are provided for developers who will be developing applications that needs a unique URI for named entities. This deliverable relates to future developments within WP7 and the developments within the datasets/use-cases provided by D1.1 and D2.2. The result of T7.2 is that all foreseen use cases in relation to unique URI are fully satisfied.
1 INTRODUCTION

This demonstration document includes the application and services created to enable a crowdsourced curation for disambiguating named entities and validation of uniqueness in the storage.

Based on “T7.2 Crowdsourcing-enabled unique URI”, this deliverable aims at providing a unique URI for named entities by integrating the crowdsourcing functionalities for disambiguation. Users of the QROWD platform will be able to identify the uniqueness of a named entity, and in case of no uniqueness, pass it to a stage of merging, which also has to be crowd-validated.

Based on the data sets provided in “D1.1 Datasets”, the QROWD platform will be working on the parking & traffic data provided by TomTom. Additionally, there will be data collected from the use-cases provided by “D2.2 Business case requirements and design” (e.g. Modal Split, Parking Fees). Although all data provided from TomTom and the municipality have unique records for roads, parking spaces, parking fees, POIs etc., named entities related to those records do not have URIs to be published in the Linked Data Cloud. By the service explained in this document, unique named entity URIs will be created for Linked Data integration purposes with the Entitypedia knowledge base and the QROWDDDB to provide a unified open access to entities.

In order to provide a unique named entity storage, the Entity Name Service (ENS) was developed by AI4BD, which provides a Unique URI Knowledge Base as required in T7.2 and explained in Section 2.1. The crowdsourcing enhancement to the ENS service is explained in Section 2.2. The API definition for the complete service is explained in Section 2.3. The running service is demonstrated and explained with sample use cases in Section 3.

2 TECHNICAL DESCRIPTION

2.1 Entity Name Service:

The Entity Name Service (ENS) is a tool, developed by AI4BD, which is acting like a DNS\(^1\) in order to provide a unique URL for a given entity based on a list of properties. In the case of the QROWD platform, an entity refers to a named entity, which will be generated for a record that needs to be uniquely stored and served in Entitypedia knowledge base and the QROWDDDB. However, the usage could be easily expanded to other use cases and domains, e.g., creating a unique named entity URI for information extracted from news texts.

In Figure 1, the architecture of the service is shown. In this architecture, the following features and sub-modules are present:

---
\(^1\) https://en.wikipedia.org/wiki/Domain_Name_System
The service provides an API to input attributes and return a unique URI based on the inputs.

- There is a queue for parallel processing of the requests.
- The service is extendable regarding data model and strategies for querying, matching and ranking. Actually, the whole dataset of unique entities with their attributes is stored as RDF. To allow for a fast querying, the data is also kept in an in-memory Lucene index.
- Caching reduces the load of service significantly.

**Figure 1: Classical ENS Architecture**
The unique entity name service requires a JSON request with a list of attributes. A sample input/output request is given below:

```json
[1. {
2.   "id": "1234",
3.   "params": [
4.     {
5.       "name": "http://www.w3.org/2000/01/rdf-schema#label",
6.       "value": "bike rack abc street 9"
7.     },
8.     {
10. },
11.     {
14.     },
15.     {
16.       "name": "http://www.w3.org/2003/01/geo/wgs84_pos#lat",
17.       "value": "37.864000"
18.     },
19.     {
20.       "name": "http://www.w3.org/2003/01/geo/wgs84_pos#long",
21.       "value": "32.538300"
22.     }
23.   ]
24. }
25. ]
26. ]
```

**Figure 2. Input: simple key-value pairs for each entity**

The service returns a unique URI together with an id and a confidence score, which is shown in Figure 3.

```json
{1.   "id": "1871646331",
2.   "url": "http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+6",
3.   "confidence": 1
4. }
5. ]
```

**Figure 3: Output: URI and confidence for each entity**

The confidence score of the similarity of two entities are calculated by comparing the properties of an entity as a text. In this manner, the simmetrics\(^2\) Java library is used to get a similarity score. In addition to text similarity, numerical similarity calculation for spatial data is added. If an entity record includes a latitude and longitude value, these values are compared with other entities based on spatial distance calculations (with a given distance threshold) by using the spatial4j\(^3\) library.

Specifically, the calculation is done for the entities in English as follows;

---

\(^2\) [https://github.com/Simmetrics/simmetrics](https://github.com/Simmetrics/simmetrics)

\(^3\) [https://github.com/locationtech/spatial4j](https://github.com/locationtech/spatial4j)
1. Three pre-processing tasks are performed on the candidate entities: Diacritics removal, converting to lowercase, replace non-word phrases.
2. Two different tokenizers were used: whitespace tokenizer, n-gram tokenizer (3-gram for this service).
3. Simmetrics CosineSimilarity class is used to calculate the similarity of the strings extracted from the entities.
4. Spatial4j within function is used to calculate the distance between 2 spatial points.

The confidence score of a unique URI is calculated as an average value out of all entity comparisons defined above. Although this score is mainly used for matching a request with a previously created URI, it is also used for creating the disambiguation tasks. In the RDF data store, the unique URIs are stored as triples as shown in Figure 4.

```
1. { "s": { "type": "uri", "value": "http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+1" }, "p": { "type": "uri", "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" }, "o": { "type": "uri", "value": "http://ens.ontos.com/resource/Entity" }},
2. { "s": { "type": "uri", "value": "http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+1" }, "p": { "type": "uri", "value": "http://www.w3.org/2000/01/rdf-schema#label" }, "o": { "type": "literal", "value": "bike rack abc street 1" }},
3. { "s": { "type": "uri", "value": "http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+1" }, "p": { "type": "uri", "value": "http://www.w3.org/2003/01/geo/wgs84_pos#lat" }, "o": { "type": "literal", "value": "37.866400" }},
4. { "s": { "type": "uri", "value": "http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+1" }, "p": { "type": "uri", "value": "http://www.w3.org/2003/01/geo/wgs84_pos#long" }, "o": { "type": "literal", "value": "32.538500" }},
```

**Figure 4: Example of the RDF data for an entity in the ENS**

### 2.2 Crowdsourcing-enabled Entity Name Service

In order to enhance the ENS service as a crowdsourcing-enabled component, further developments have been performed. Whenever a new entity name record is created, the entity is checked for similarities among other entities, if the confidence score (i.e. cosine similarity) of the entity properties is higher than a predefined threshold, then the entity is stored in a different disambiguation graph temporarily until the disambiguation is resolved. The enhanced architecture is shown in Figure 5 including Disambiguation Task Generator, Disambiguation Resolver and related API enhancements.
2.3 API definition

This section explains the API definitions available in the Crowdsourcing-enabled unique URI component.

2.3.1 GET URLs for Entities

The main endpoint is available under /entities, which can be queried by the following options:

- **HTTP-Method:** POST
- **Header:** Content-Type: application/json; charset=utf-8
- body need to contain an array with at least one JSON object with an ID to refer back in the response and an array of parameter - value - pairs that are used to gather the correct entity URI upon.
- The parameter name need to be a URI so that it could be used as predicate in a RDF graph.
Based on the API definitions, an example call is shown in Figure 7.

```json
{
  "id": "1871646331",
  "params": [
    {
      "name": "http://www.w3.org/2000/01/rdf-schema#label",
      "value": "IPhone"
    },
    {
      "name": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
      "value": "https://schema.org/Product"
    }
  ]
}
```

**Figure 7: Example URI Request**

The response to the request in Figure 7 is shown in Figure 8, which comprises

```json
[
  {
    "id": "1871646331",
    "url": "http://ens.ontos.com/resource/iphone",
    "confidence": 1
  }
]
```

**Figure 8: Unique URI Result with a Confidence Score**

the ID, the unique URL and the confidence value between 0 and 1.

### 2.3.2 GET Disambiguation Tasks

The main endpoint is available under **disambiguationtasks**

- **HTTP-Method:** GET
- **Header:** Content-Type: application/json; charset=utf-8
D7.2 Data storage and access component

- the service should call the `DisambiguationTaskManager` which queries all tasks that are currently available from the graph
- the output should be a JSON in the following format like below

  ✓ **sparqlEndpoint**: the URI how to reach the endpoint
  ✓ **graph**: the URI of the ENS graph, not of the disambiguation graph. It should be accessible in read-only mode without authentication for the current demo scenario
  ✓ **task**: the URI of the disambiguation task

```json
1. {
2.   "sparqlEndpoint": "http://ontoquad.qrowd.aksw.org/sparql",
4.   "tasks": [
5.     {
7.       "entities": [
11.      ]
12.    }
13.   ]
14. }
```

**Figure 9: Response List of Disambiguation Tasks**

### 2.3.3 Resolve Disambiguation Tasks

The main endpoint is available under `/disambiguationstaskResolve`

- **HTTP-Method**: POST
- **Header**: `Content-Type: application/json;charset=utf-8`
- The input should be a JSON in the format shown in Figure 10
  - **task**: the URI of the disambiguation task
  - **identicalEntities**: the URIs of the entities which are marked as identical to the task entity.

```json
1. {
2.   "tasks": [
3.     {
5.       "identicalEntities": ["http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+4"]
6.     }
7.   ]
8. }
```

**Figure 10: Sample Resolve Request**


2.3.4 GET Entity Details

The main endpoint is available under /getEntities, which can be queried by the following options:

- **http-Method**: POST
- **Header**: Content-Type: application/json; charset=utf-8
- body need to contain an array with at least one JSON object.

A sample request and response for this function is listed below.

```json
{
  "entities": ["http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+5",
               "http://aistudio.ai4bd.com/resource/ens/bike+rack+abc+street+6"]
}
```

**Figure 11: Request Sample for the Get Entity Details**
Figure 12: Response Sample for the Get Entity Details
2.3.5 Management API

To manage the ENS, the API under `/manager` is used. Figure 10 gives a sample call to clear the ENS.

```
1. curl -X POST \\
```

Figure 13: Clear ENS API Call

In order to submit a task to load data of a given type from DBpedia into the ENS, a sample command illustrated in Figure 14 can be used. The 2 parameters are `typeUri` (String [URI], mandatory) and `clear` (boolean, optional)

```
1. curl -X POST \\
3. -H 'cache-control: no-cache' \\
4. -H 'content-type: application/x-www-form-urlencoded' \\
5. -d 'typeUri=http%3A%2F%2Fschema.org%2FProduct&clear=false'
```

Figure 14: Sample URI Load

In order to get a list of tasks executed to load data from DBpedia, a sample command illustrated in Figure 15 can be used.

```
1. curl -X GET \\
2. http://localhost:8080/ens/manager/list \\
3. -H 'cache-control: no-cache'
```

Figure 15: Sample Data Load Trace
3 BIKE RACK USE-CASE FOR ENTITY DISAMBIGUATION

In this use case, the existence of bike racks is to be saved, disambiguated, and shared.

A simple bike rack may have the following attributes:

- label
- URL to an image : String → URL
- lat : float
- long : float
- type : String
- capacity : int
- availability

The idea is to extend existing bike rack repositories such as the ones listed in Open Street Map\(^4\) (OSM). For Trento, there are some bike racks already stored in OSM, which are shown in Figure 16. By using pre-existing sources as a starting point will allow this service to be cross-validated in terms of uniqueness.

A crowdsourcing task can be created to solve a disambiguation problem by using the images of similar/closer bike racks, which cannot easily be solved by image similarity algorithms.

\(^4\) http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbicycle_parking
A node shown in Figure 16 is defined as an XML record shown in Figure 17.

1. `<xml version="1.0" encoding="UTF-8">`
2. `<osm version="0.6" generator="CGIgenerator 0.6.0 (32189 thorn-02.openstreetmap.org)" copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright" license="http://opendatacommons.org/licenses/odbl/1-0/">`
3. `<node id="1366801259" visible="true" version="1" changeset="8770327" timestamp="2011-07-19T16:50:34Z" user="pikappa79" uid="330007" lat="46.0704828" lon="11.1267200">`
4. `<tag k="amenity" v="bicycle_parking"/>
5. `<tag k="capacity" v="4"/>
6. `</node>
7. `</osm>`

Figure 16: Trento Bike Racks Map

Figure 17: An Example Bike Rack Record in Open Street Map

As well as pre-existing sources, bike rack information may also be collected from different sources; such as iLog app or MT official records. Independent from where the data comes from, all bike racks can be defined as a named entity and be associated with a unique URI. In order to define a unique URI for
each bike rack, the ENS service explained in this document will be used. Although the OSM dataset includes information about bike racks, those are not marked as a named entity which is defined by a unique URI. Therefore, independent from the source of the data, all bike racks will be associated with a unique URI and defined as a named entity.

The URI generation stage defined in this document allows similar bike rack entities to be disambiguated by creating HIT tasks for the crowdsourcing component. In that sense, the named entity text and spatial similarity scores will be used to create a disambiguation tasks which cannot be solved by the service.

The sequence diagram in Figure 18 summarizes the proposed data flow starting from an app to a crowdsourcing-enabled curation.
In terms of the data access and storage component, the use case requires two main parts.

1. Synchronization Agent:

   This agent is designed to create periodical checks over databases, which are explained as follows;
• If new bike racks are stored in a Cassandra database, so receive a unique URI for them as identifier.
• If there are some entities in the ENS that are nearly similar so that a manual disambiguation task is required, so push these entities for the disambiguation to the Context Broker
• If there are updates of disambiguation tasks, e.g., finished, so request an update in the ENS

2. Unique URI Service (Entity Name Service, ENS)

   The crowdsourcing enabled unique URI service to ensure the uniqueness of the named entities supported by HITs generated by the ENS.

Other than the data access and storage component, the following components are used to support the generation and curation of the bike rack entity data.

1. Data Production App:

   For this use-case, the bike rack information is assumed to be created by an app/software. This app creates and updates a bike rack record to the cassandra database or the QROWDDB. The storage and the bike rack records will be synchronized with the named entities (URIs) created for each entity (bike rack).

2. Quad Store:

   The entity related information will be stored in this RDF storage to identify unique URIs for new requests. This storage is the backend for the ENS service and can be replaced by any W3C standards compliant RDF storage.

3. Context Broker:

   A Fiware Orion Context Broker will be used to exchange bike rack information in real time for the quality checking and entity disambiguation tasks.

4. Crowdsourcing Task:

   A crowdsourcing task (HIT job) will be generated by using similar entities (identified by the ENS) or from the quality checking results.

5. Quality Checking:

   This is to ensure the quality of the bike rack data, which will work together with the ENS component. This component is a part of “D7.1 Data quality assessment services”, which is explained in D7.1 outcomes.