QROWD - Because Big Data Integration is Humanly Possible

Innovation action

D7.1 – Crowdsourced Streaming Data Quality Assessment

Author/s	Claus Stadler (InfAI) Gordian Dziwis (InfAI)
Due date	30.11.2017
Actual delivery date	01.12.2017
Version	1.0
Dissemination level	PU
Status	FINAL

Project co-funded under Horizon 2020 Research and Innovation EU programme, grant agreement no. 732194

History

Version	Date	Reason	Revised By
0.1	2017-08-21	Initial structure and notes	Claus Stadler
0.2	2017-09-29	Revision based on Leipzig meeting discussion	Claus Stadler
0.3	2017-10-06	Initial architecture	Claus Stadler
0.4	2017-10-18	Revision of the architecture	Gordian Dziwis
0.5	2017-10-24	Added technical description of the data flow	Gordian Dziwis
0.6	2017-11-03	Addition of related work	Claus Stadler
0.7	2017-11-07	Revision of the data flow	Gordian Dziwis
0.8	2017-11-16	Addressing internal review comments	Claus Stadler
0.9	2017-11-27	Proof reading	Gordian Dziwis
1.0	2017-11-29	Addressing internal review comments	Claus Stadler

Table of Contents

List of Figures

List of Listings

List of Abbreviations

CS	crowdsourcing
QA	quality assessment (a subtask of quality assurance)
DQ	data quality
RDF	Resource Description Framework
SPARQL	SPARQL protocol and RDF query language
OSM	OpenStreetMap

ABSTRACT

In this deliverable we investigate options for effective crowdsourcing of data quality assessment tasks and present a system prototype for the disambiguation of data records concerning bike racks.
Data quality is commonly conceived as *fitness for use* and comprise many aspects, such as timeliness, consistency, syntactic validity and factual accuracy. Certain aspects, especially the technical ones, e.g. accessibility, encoding and syntax, can often be validated automatically. However, other aspects, such as the factual accuracy - defined as (the degree of) accordance with reality - are significantly more difficult to assess. With the increasing availability of appropriate background knowledge this task can be automatized to a large extent, although there still needs to be human verification to ensure an appropriate level of quality. Many large datasets, such as generated by Wikipedia, Wikidata, GoogleMaps or OpenStreetMaps have in common, that curation is also collaboratively performed by a crowd.

In this work we first review (a) the state-of-the-art in data quality dimensions and metrics, and (b) crowdsourcing platforms and their use for data quality assessment. Based on the findings we devise an architecture for data quality assessment within the QROWD project. Present our realized infrastructure built from our own as well as third party software components and apply it to a use case of verifying bike rack data in Trento.

Executive Summary

This document is aimed at developers and audiences with an interest in topics related to data integration, crowdsourcing (CS), software engineering and data quality.

The main outcomes of this deliverable are as follows: First, we present a system design and conceptual considerations for on-demand data quality assessment on integrated (RDF) datasets using crowdsourcing platforms. Thereby, we put a particular focus on easing the process of making data records subject to quality assessment available in the CS platform. Second, we developed Java/Scala components which presently support the PyBossa CS and publish/subscribe communication patterns via the FIWARE Orion Context Broker. And third, we demonstrate our system using a bike racks use case.

The use case demonstrator is a joint work with D7.2 (Crowdsourcing-enabled unique URI), which features a similarity search service. In a general data quality assessment task, the work of D7.2 will first try to disambiguate entities automatically, but when the disambiguation confidence is below a required threshold, requests for human resolution are made to our system.

This work also forms the basis for the assessment of the interlinking and data fusion tasks of WP5 (Hybrid Link-Discovery and Fusion).

1. Introduction

Data quality assessment and repair is an ubiquitous topic in data intensive fields, such as data integration, machine learning, navigation systems, and distributed authoring scenarios. In general, classical crowdsourcing alone cannot compete with automatic approaches in terms of scale and performance. Even when highly parallelized, human computation processes yield delivery times that are generally orders of magnitude worse than the ones provided by machines, though their accuracy is typically higher. Yet, automatic approaches can be supported with human computation to improve the overall quality of a system's response.

In this deliverable, we assume a setting where a semi-automatic system can request human input on cases where the confidence in the computed solutions islow. As a concrete use case, we demonstrate the solution to the problem of disambiguating/deduplicating bike racks collected via crowdsourcing tasks, with those available in other data sources (e.g. Open Street Maps, data from municipalities), in connection with BC2-UC#3 - "Completing information about mobility infrastructure through spatial crowdsourcing", described in deliverable D2.2.

This document also relates to the following deliverables:

- D3.1 (Participatory framework) introduces the general framework for including participation of crowdworkers and citizens for tackling the computationally complex problems that appear in the context of the QROWD project. The design and implementation of this component was informed by it.
- D7.2 (Crowdsourcing-enabled unique URI) presents the *Entity Name Service* (ENS), which is a similarity search service that for a given data record determines whether or not it describes a known entity. In a general data quality assessment task, ENS will first try to disambiguate entities automatically, but when the disambiguation confidence is below a required threshold, ENS invokes a crowdsourcing disambiguation task, that is performed by the component described in this deliverable.
- D5.1 (Link discovery and data fusion algorithms) will need quality assessment of the proposed algorithms for which we plan to use and enhance the work presented in this document.

In summary, we performed the following work:

- We devised a general system architecture for realizing distributed crowdsourced streaming data quality assessment workflows using a publish/subscribe infrastructure.
- We provide implementations of essential components to effectively and efficiently implement dataflows from an (integrated) RDF dataset to a crowdsourcing (CS) system.
- A concrete implementation of a data quality assessment system on the example of bike racks disambiguation is presented.

- All our resources are publicly available as open source on GitHub[1].

Our system uses the following key technologies:

- The FIWARE[2] Orion Context Broker[3], an entity store and publish/subscribe system at the core of SmartCities infrastructures.
- A Java/RDF Object Mapper[SL17][4] to bridge the gap between the RDF format - with its advantages in regard to data integration - and the JSON world - with its omnipresence in (Web) application development, including CS systems, due to its ease-of-use.
- The popular open source Pybossa[5] crowdsourcing platform, of which Crowdflower is the hosted version. Hence, our system design and implementation offer the flexibility of taking advantage of the global Crowdflower community, as well as performing e.g. self-hosted deployments within an organization.

The remainder of this document is structured as follows:
In Section 2 we give an introduction to data quality & assessment concepts from a scientific perspective. In Section 3 we present a generic distributable system design for enabling quality assessment using crowdsourcing. Subsequently, in Section 4, we present a prototype implementation of this system adapted for a bike racks use case. Finally, in Section 5 we conclude this report and give pointers for future work.

2. An Overview of Data Quality & Assessment

Data quality (DQ) is commonly conceived as *fitness for use* and conversely, *freedom from deficiencies* [JG99]. While low quality data is often sufficient for use in prototypes, which demonstrate that a system can *in principle* be effective. However, the same data could be insufficient in critical applications, like medical ones, where such data may lead to severe consequences to a patient's health. In general, low quality data is likely to negatively impact the effectiveness of an organisation.
A widely accepted approach to the understanding of data quality is, to introduce *data quality dimensions* and define respective metrics for their quantification.
Frequently mentioned DQ dimensions are accuracy, completeness, consistency, and timeliness [WW96]. While some dimensions are generally applicable, others only apply to certain application domains. For example, substantial work on identifying further ones has been done in the Linked Data domain [ZRMPLA15].
Literature categorizes the approaches by which dimensions can be derived into: theoretical (based on a formal model), empirical (based on e.g. questionnaires) and intuitive.[SCC02][BCS10]

[1] https://github.com/QROWD/rdf-crowdsourced-quality-checker
[2] https://www.fiware.org/
[3] https://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
[4] https://github.com/SmartDataAnalytics/jena-sparql-api/tree/develop/jena-sparql-api-mapper
[5] http://pybossa.com/

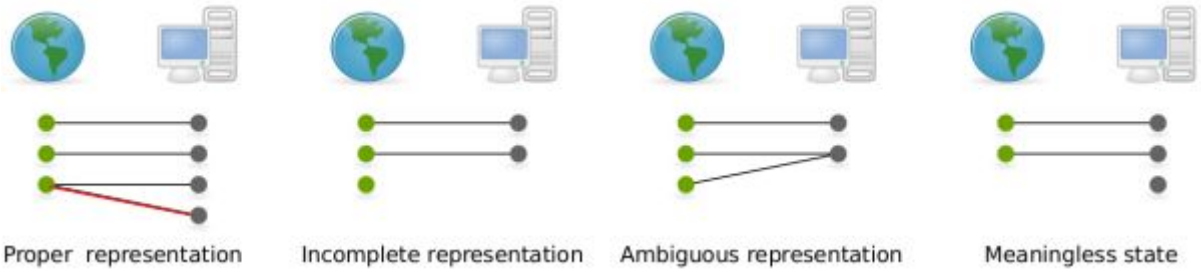

Figure 1: Quality Model of Wand & Wang

Figure 1 shows an example of a theoretical approach based on the formal notion of an *information system* capturing a representation of states in the real world. In regard to proper representations, the authors state, that two conditions must hold: First, every lawful state of the real world system should be mapped to at least one lawful state of the information system (a real-world state can be mapped into multiple information system states). Second, it should be possible, in principle, to map an information system state back to the "correct" real-world state.
The work of [PLYW02] distinguishes between subjective and objective DQ metrics and concludes that “experience suggests a ‘one size fits all’ set of metrics is not a solution”. In this regard, with this deliverable we aim providing a flexible core infrastructure to which can be easily adopted to future data quality assessment tasks within the QROWD project.

3. Streaming Data Quality Assessment System Prototype

In this section, we describe our architecture for facilitating streaming data quality assessment using a crowdsourcing system. *Streaming* hereby refers to the support of on-demand task creation and sending out respective solutions as soon as they are available using a message broker infrastructure.

3.1 System Architecture

Figure 2 presents the architecture of the quality assessment system in a disambiguation task scenario.

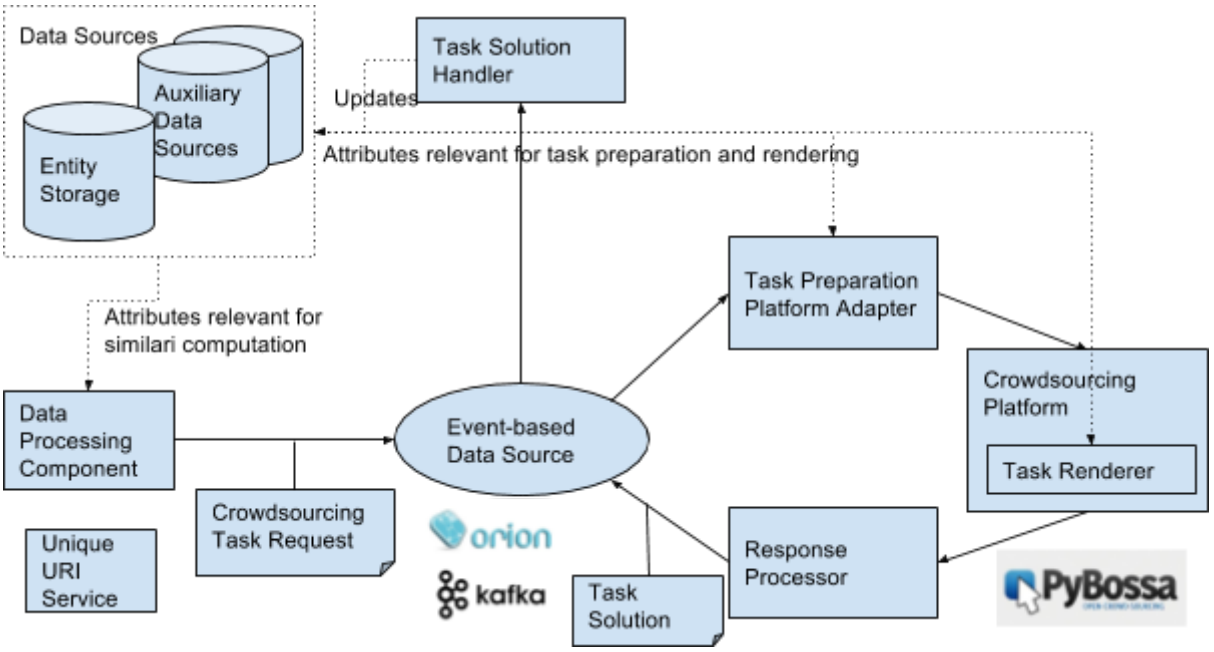

Figure 2: Architecture of the Streaming Data Quality Assessment System

- *Entity Storage & Auxiliary Data Sources:* In general, we assume the existence of a primary database which contains the data records about entities of interest. In addition, information about these entities may be distributed across multiple (possibly heterogeneous) data sources. Entity attributes can conceptually play a role in three stages throughout the quality assessment process: First, in the initial processing which may introduce data quality problems. Second, when preparing the static task description to be handed to the crowd sourcing platform. And third, when presenting the task to a contributor.
- *Data processor:* This component sends out appropriate task creation requests when encountering potential data quality issues.
- Task Preparation Platform Adapter: This component acts as a connector between the task creation request and a specific crowd-sourcing platform. Its purpose is to take the necessary steps for the crowd-sourcing platform to be capable of presenting the task to its users in an appealing way. It may re-use

task renderers available by the platform or register its own.

- *Task Renderer:* This component encapsulates the presentation logic, and handles yielding the appropriate data record for a user's answer to the system.
- *Crowd-sourcing platform:* The platform takes care of user, task and answer management. Advanced platforms provide out-of-the box analysis facilities, such as agreement coefficient and minimum required answers for a task considered to be answered.
- *Response processor:* Answers to tasks in the crowd-sourcing platform are picked up by the processor and transformed into messages that are distributed via the message channel for pick up by the appropriate recipients. Ideally, the platform pushes messages of tasks that met the defined criteria for being considered answered. Additional analysis outside of the scope of crowdsourcing platforms can also be performed at this stage.

3.2 Message Exchange via Publish/Subscribe

In order to support processing of tasks in a streaming fashion, i.e. posing tasks to the crowd on-demand as well as retrieving their aggregated responses as soon as they become ready, our system is designed to support arbitrary publish/subscribe infrastructures. Several infrastructures, comprising both protocols and server/client implementations, exist for this purpose, such as the FIWARE Orion Context Broker, Apache Kafka[6], and all Advanced Message Queuing Protocol (AMQP) compliant systems, such as RabbitMQ[7]. Libraries that help implementing Reactive Programming paradigms[8], a.k.a Flows, in conjunction with Dependency Injection concepts can often be applied to appropriately abstract away the technical details of the messaging infrastructure, thus allowing the development of components that can be readily re-used in a multitude of computing environments, regardless of whether central or decentral.

3.3 Provisioning of Task Descriptions

Using the publish/subscribe infrastructure, one component can express its need for human input, whereas another component can accept the request and take the necessary actions to satisfy the need. However, this requires agreed-upon protocols to achieve successful communication between the components as well as data models that adequately capture the necessary information.

In this regard, the JavaScript Object Notation (JSON) format provides significant advantages:

- As a text-based format, it is human readable in contrast to binary formats. This simplifies the investigation of problems in the messaging infrastructure.
- It is the most popular data exchange format due to its simplicity while still

[6] https://kafka.apache.org/
[7] https://www.rabbitmq.com/
[8] https://www.reactivemanifesto.org/

being extensible.

- If desired, data can be represented in accordance with the JSON-LD specification, which makes such JSON documents compatible with RDF

Static vs Dynamic Data Exchange

An essential question when designing a QA pipeline is, which attributes need to be part of the static task description registered at a crowdsourcing system, and which ones can be retrieved during task-presentation or solution-handling dynamically on-demand.

As crowdsourcing systems usually keep a history of task solutions, in order to present historical, possibly solved, tasks in the same way as they were originally posed, a static snapshot of all related information has to be made accessible to the crowdsourcing system. However, a task may include links to auxiliary sources, such as definition pages on OpenStreetMap, articles on Wikipedia, etc. While it would be possible to create snapshots of auxiliary sources, this is only worthwhile if a use case demands it.

4 Prototype Implementation

Here we describe a concrete use case together with its realization for disambiguation of bike racks in Trento, based on the design considerations of the prior section.

4.1 The Bike Racks Use Case

In accordance with BC2-UC#3 - "Completing information about mobility infrastructure through spatial crowdsourcing", described in deliverable D2.2, the municipality of Trento is in the process of assembling an official dataset of their bike rack locations for public release. There are already third party datasets that are publicly available, in-use and highly likely to contain overlapping information. This use case is about facilitating the assessment of the quality of the official data in regard to reference data and thus enable the identification of potential data quality problems and their resolution.

The goal of *D7.2 Crowdsourcing-enabled unique URI* is to offer the *entity name service* (ENS) service that features similarity search based on given data records (in JSON format), and which is capable of assigning existing identifiers (URIs) or allocating new ones.

The goal of this deliverable is to facilitate the automatic creation of human interaction tasks for resolving cases where the similarity between two entities is such that the accuracy of an automatic equivalent/distinct decision is expected to be low and thus likely to introduce data quality problems.

In this work, we use the OpenStreetMap[9] (OSM) project, whose goal is to create a free map of the world, as the source for reference data. OSM uses three primitive entities for modeling maps, which are nodes, ways and relations, corresponding to point, line/area and complex geometries, respectively. All entities can be annotated with key-value pairs, called tags. While tags can be chosen freely, the community is aware of and concerned with semantics, and thus works out guidelines on how to appropriately tag entities. Effectively, this is a devising of data models and has been done for bicycle parking[10].
The LinkedGeoData community project[SHLA12][11] provides tooling for converting OSM data to RDF, thus making it generally easier to consume in data integration pipelines.

4.2 Interactions between QROWD Components

Figure 3 depicts how the quality assessment service interacts with the Unique Entity Name Service of D7.2 and the Fiware Orion Context Broker. Essentially, the ENS can request crowd workers to assess the accuracy of detected candidates matches via our QA system.

[9] http://openstreetmap.org
[10] http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbicycle_parking
[11] http://linkedgeodata.org

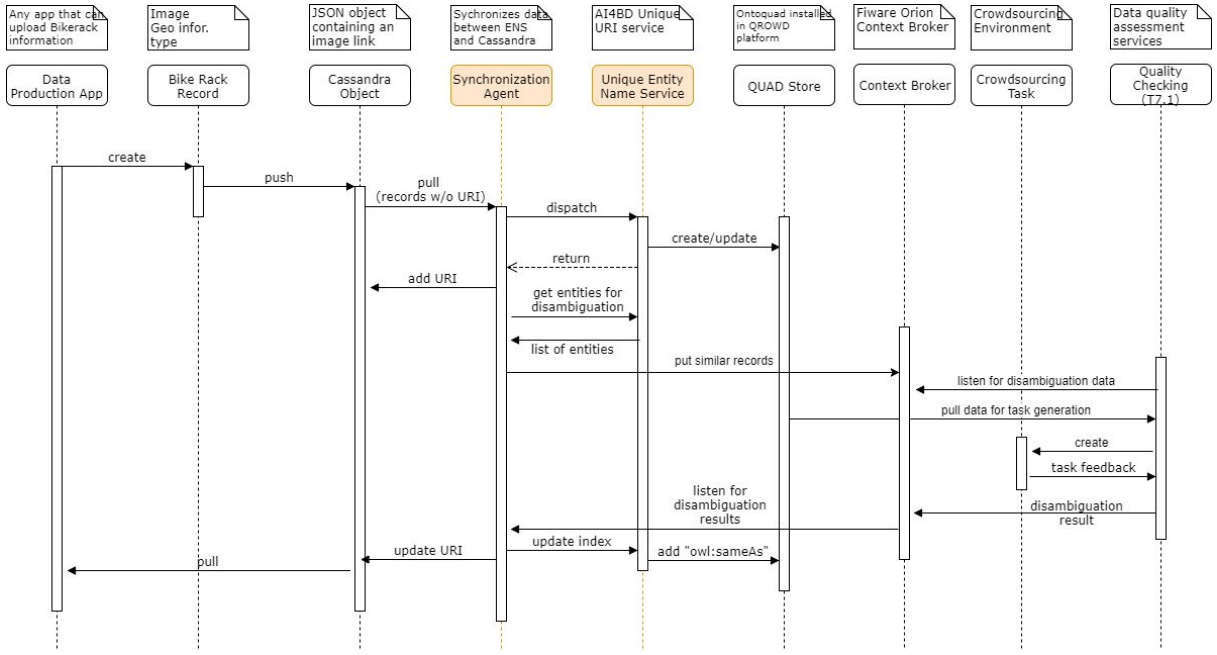

Figure 3: Sequence diagram of the interactions between the data production service and data quality assessment service

4.3 Transferring Data from a SPARQL Endpoint into Pybossa

In this section, we describe the technical details about how disambiguation requests are handled by our QA system. Listing 1 shows an example JSON document describing a task request emitted by the ENS.

```
{
  "sparqlEndpoint": "http://ontoquad.qrowd.aksw.org/sparql",
  "graph": "http://aistudio.ai4db.com/resource/ens/2",
  "tasks": [{
    "task": "http://aistudio.ai4db.com/resource/ens/bike+rack+abc+street+6",
    "entities": [
      "http://aistudio.ai4db.com/resource/ens/bike+rack+abc+street+1",
      "http://aistudio.ai4db.com/resource/ens/bike+rack+abc+street+2"
    ]
  }]
}
```

Listing 1: A JSON document describing a disambiguation task

The *entities* fields holds the resource identifiers which were sufficiently similar - yet different enough - to qualify as candidate matches for the identifier in the *task* attribute. The *sparqlEndpoint* and *graph* attributes refer to the SPARQL service where additional information about the resources can be retrieved.
Note, that the task description does not contain any attributes. Yet, there is a reference to the SPARQL endpoint from which any desired available information can be retrieved.
Listing 2 shows how Java classes can be annotated in order to map RDF data directly to a Plain Old Java Object (POJO). This approach gives three advantages:

(1) No SPARQL is needed to fetch the attributes, (2) objects can be serialized as JSON in a single line of code, and (3) it enables schema-based validation, such as checking for nulls and the consistency of data types.

```
@RdfType("lgdo:BicycleParking")
public class BikeRack {
	@Iri("lgdo:bicycleParking") public BikeRackType bikeRackType;
	@Iri("geo:wgs84_pos#lat") public Double xlat;
	@Iri("geo:wgs84_pos#long") public Double xlong;
	@Iri("lgdo:capacity") public Integer capacity;
	@Iri("dbo:thumbnail") String thumbnail;
}

@RdfType("lgdo:BicycleParkingType")
public class BikeRackType {
	@Iri("rdfs:label") public String label;
	@Iri("foaf:depiction") public String depiction;
}
```

Listing 2: Annotated Java Class for specification of relevant entities and their attributes in a (SPARQL-accessible) RDF dataset

Listing 3 demonstrates the use of the RDF/Java mapper: First, one has to configure a SparqlEntityManagerFactory with (1) the appropriate prefixes, (2) the package names about where to look for annotated classes, and (3) the reference to the SPARQL service & graph name. This enables querying the RDF dataset indirectly via the classes and attributes of the Java domain model. Queries can be posed using the criteria query feature of the standard Java Persistence API[12] (JPA).

```
SparqlEntityManagerFactory emFactory = new SparqlEntityManagerFactory()

emFactory.getPrefixMapping()
    .setNsPrefix("lgdo", "http://linkedgeodata.org/ontology/")
    .setNsPrefix(/* ... */);

emFactory.setSparqlService(FluentSparqlService
    .http("http://ontoquad.qrowd.aksw.org/sparql", "http://aistudio.../ens/2")
    .create());

emFactory.addScanPackageName(BikeRack.class.getPackage().getName());

EntityManager em = emFactory.getObject()

/* Example queries: */

BikeRack anIndividualBikeRack = em.find(BikeRack.class,
"http://aistudio.ai4db.com/resource/ens/bike+rack+abc+street+6");

List<BikeRack> allBikeRacks = em.list(BikeRack.class);

List<BikeRack> specificBikeRacks =
    JpaUtils.getResultList(em, BikeRack.class, (cb, cq) -> {
              Root<BikeRack> r = cq.from(BikeRack.class);
              cq.select(r)
                     .where(cb.greaterThanOrEqualTo(r.get("capacity"), 20))
    });
```

[12] https://www.jcp.org/en/jsr/detail?id=338

Listing 3: Example code for querying the mapped RDF dataset

Note, that a developer only has to cope with Java domain model; the RDF specifics are abstracted away by the mapper engine.

The Pybossa server provides an RESTful API for the interaction with its domain-objects.[13] By posting a JSON document shown in Listing 4, to the tasks endpoint (http://{pybossa-site-url}/api/tasks[?api_key=API-KEY]) a new task for the crowd will be created.
This JSON document contains in the field *info* the information from the SPARQL queries concerning the entities. This payload is produced directly by writing the Java domain model to a JSON Document. Additional there are parameters for the Pybossa platform: With the *calibration* field this task can be assigned as an gold standard for evaluating crowd workers. The field *n_answers* determines how many times the task must be answered, until it is considered as completed. While the *quorum* field specifies the number of times this task must be answered by a *different* user.

```
{  "project_id": 2,
  "info": [{
        "bikeRackType": {
        "label": "A special kind...",
        "depiction": "http://localhost:9000/assets/images/image-2.png"
        },
        "lat": 45.4069416,
        "long": 11.874385700000001,
        "capacity": 80,
        "thumbnail": "http://www.landscapeforms.com/..."
  }, {
        "bikeRackType": {
        "label": "A bent piece...",
        "depiction": "http://localhost:9000/assets/images/image-1.png"
        },
        "lat": 45.4081078,
        "long": 11.8714984,
        "capacity": 30,
        "thumbnail": "https://cyclesafe.com/wp-content/..."
  }],
  "calibration": false,
  "priority_0": 0,
  "n_answers": 30,
  "quorum": 0
}
```

Listing 4: JSON document for posting to the Pybossa API in order to create a task

Collecting Results from Pybossa

We provided a custom template for the Pybossa server to present a task to a worker. Figure 4 is a screenshot of the view presented to the worker.
Every time a worker answers a task the Pybossa server creates a task_run object.

[13] http://docs.pybossa.com/api/intro/

```
[
  {
    "info": "No",
    "external_uid": null,
    "user_id": 1,
    "links": [
      "<link rel='parent' title='project' href='http://localhost:5000/api/project/2'/>",
      "<link rel='parent' title='task' href='http://localhost:5000/api/task/18'/>"
    ],
    "task_id": 18,
    "created": "2017-11-28T13:39:53.585998",
    "finish_time": "2017-11-28T13:40:16.596225",
    "calibration": null,
    "user_ip": null,
    "link": "<link rel='self' title='taskrun'
href='http://localhost:5000/api/taskrun/6'/>",
    "timeout": null,
    "project_id": 2,
    "id": 6
  }
]
```


Listing 5: A Pybossa task_run object

The task_run object stores who answered a task and stores the corresponding answer.
When enough people answered a task, Pybossa marks the task as completed and creates a result object. The prototype is notified via a webhook configured in the Pybossa project.


```
{
  "id": 2,
  "created": 1511864591000,
  "project_id": 1,
  "task_id": 2,
  "task_run_ids": [10, 9, 2, 3],
  "info": null
}
```


Listing 6: A Pybossa result object

The result object gives the Ids of all the task_run objects belonging to a finished tasks. The answers are requested from the pybossa server, aggregated and the task resolution is broadcasted via the message broker.


```
{
  "tasks": [{
    task: "http://aistudio.ai4db.com/resource/ens/bike+rack+abc+street+6",
    "identicalEntities": ["http://aistudio.ai4db.com/resource/ens/bike+rack+abc+street+1"]
  }]
}
```


Listing 7: The final task resolution JSON document

Figure 4: Screenshot of a bike rack disambiguation task in Pybossa rendered using our custom template.

5. Conclusions and Future Work

In this deliverable we (a) summarized aspects of data quality assessment, namely data quality models, dimensions and metrics, (b) presented the design and implementation of our highly customizable streaming data quality assessment system on the example of disambiguating data records about bike racks in the municipality Trento.

Notably, this work complements the efforts of D7.2 (*Crowdsourcing-enabled unique URI*).

The broader goal of this work is to establish the technical foundations to ease the implementation of future data quality assessment workflows within and outside of QROWD. For example, we plan to improve the system in order to validate the outputs of the interlinking and fusion systems to be delivered in WP5 by M24. By providing initial support for using the FIWARE Orion Context Broker as a means to relay quality assessment requests and responses, we investigated the technical foundations for integrating our system in SmartCities infrastructures.

REFERENCES

[JG99] Juran's quality handbook, J. Juran, and A. Godfrey. Juran's quality handbook, 5e McGraw Hill, (1999)

[ZRMPLA15] *Quality Assessment for Linked Data: A Survey*, A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. In: Semantic WebJournal (2015)

[SL17] *JPA Criteria Queries over RDF Data*, C. Stadler, and J. Lehmann. In: Workshop on Querying the Web of Data co-located with the Extended Semantic Web Conference, (2017)

[WW96] Wand, Yair and Richard Y. Wang: Anchoring Data Quality Dimensions in Ontological Foundations. Communications of the ACM, 39(11):86–95, November 1996.

[AZSKAL13] Crowdsourcing Linked Data quality assessment by Maribel Acosta, Amrapali J. Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer,and Jens Lehmann in 12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia

[PLYW02] Data quality assessment.
Pipino, Leo L., Yang W. Lee, and Richard Y. Wang.
In: *Communications of the ACM* 45.4 (2002): 211-218.

[SCC02] Scannapieco, Monica and Tiziana Catarci: Data Quality under the Computer Science Perspective. Archivi & Computer, 2:1–15, 2002.

[BCS10] Batini, Carlo and Monica Scannapieco: Data Quality: Concepts. Methodologies and Techniques. Data-Centric Systems and Applications.
Springer-Verlag, Berlin Heidelberg, 2010.

[SHLA12] Stadler, C., Lehmann, J., Höffner, K., & Auer, S. (2012). Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, *3*(4), 333-354.