Welcome to ESWC Tutorial: Constructing Question Answering Systems over Knowledge Graphs
2021-06-07, online

Dennis Diefenbach
QA Company
dennis.diefenbach@the-qa-company.com

Pierre Maret
Université Jean Monnet
pierre.maret@univ-st-etienne.fr

Andreas Both
Anhalt University of Applied Sciences
andreas.both@hs-anhalt.de

Aleksandr Perevalov
Anhalt University of Applied Sciences
aleksandr.perevalov@hs-anhalt.de

Paul Heinze
Anhalt University of Applied Sciences
paul.heinze@student.hs-anhalt.de

All information will be available at https://qanswer.github.io/QA-ESWC2021/
Let’s get started

● Agenda of the tutorial
● Brief round of introductions of presenters
● Short survey about prior knowledge
  ○ [https://www.menti.com/edde7vswm2](https://www.menti.com/edde7vswm2)
  ○ Or go to menti.com and type 9383 4196
● Brief round of introductions of participants

Most of the information are available here:

https://qanswer.github.io/QA-ESWC2021/
Keynote and introduction into the field of Question Answering
general principles and variations
by Pierre Maret

https://docs.google.com/presentation/d/1S941Q4tzVJ1FZyMjH5jggX65cFKboAKrq32hvXn5K0s/edit?usp=sharing

All information will be available at https://qanswer.github.io/QA-ESWC2021/
Understanding the tasks of Question Answering (QA) over Knowledge Graphs (KGs) by Dennis Diefenbach
What is Question Answering over Knowledge Graphs?

ESWC 2021 (Q102560510)

computer science conference
Extended Semantic Web Conferences

* In more languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Label</th>
<th>Description</th>
<th>Also known as</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>ESWC 2021</td>
<td>computer science conference</td>
<td>Extended Semantic Web Confer...</td>
</tr>
<tr>
<td>Polish</td>
<td>No label defined</td>
<td>No description defined</td>
<td></td>
</tr>
<tr>
<td>French</td>
<td>ESWC 2021</td>
<td>No description defined</td>
<td></td>
</tr>
<tr>
<td>Greek</td>
<td>No label defined</td>
<td>No description defined</td>
<td></td>
</tr>
</tbody>
</table>

All entered languages

Statements

instance of:
- academic conference

0 references

add reference

add value

short name:
- ESWC 2021 (English)

0 references

add reference

add value

country:
- Greece

0 references

add reference

Oscar Corcho

Ruben Verborgh

Katja Hose

Heiko Paulheim
The Question Answering Process

- Question analysis
- Phrase mapping
- Disambiguation
- Query construction
Collect informations which can be deduced considering only the syntax of the question
- Type of the question
- NE recognition
- Identify the properties
- Identify dependencies

What is the population of Europe?
Collect informations which can be deduced considering only the syntax of the question
- Type of the question
- NE recognition
- Identify the properties
- Identify dependencies

What is the population of Europe?
Phrase Mapping

Mapping a phrase to possible resources in the underlying ontology

What is the population of Europe?

dbo:populationTotal

dbr:Europe (band)

dbr:Europe

dbr:Europe (dinghy)

dbr:Europe (anthem)
Mapping a phrase to possible resources in the underlying ontology

What is the population of Europe?

dbo:populationTotal

dbr:Europe (band)

dbr:Europe

dbr:Europe (dinghy)

dbr:Europe (anthem)
Query Construction

Use all informations collected in the steps before to construct a SPARQL query

What is the population of Europe?

Select * where {
  dbr:Europe  dbp:populationTotal  ?p
}
NE recognition

Who is the director of the Lord of the Ring?

- Use a NE recognition tool
  - Problem: Stanford NER tool could recognize only 51.5% of the NE in the QALD-3 training set
- Check all n-grams
  - Who is the brother of the CEO of the BBC?
use POS Tagging

When was the European Union founded?

General strategy: identify some reliable POS tags expressions

1. Hand made rules
2. Use ReVerb, based on the following regex

\[ V \mid VP \mid VW*P \]
\[ V = \text{verb particle? adverb?} \]
\[ W = (\text{noun | adjective | adverb | pronoun | determiner}) \]
\[ P = (\text{preposition | particle | inf. marker}) \]
Question Analysis

deep neural networks

Learn all this from embeddings
Summarizing

Works only for well formulated questions.
Is highly multilingual !!!!

Attention: Which countries are in the European Union?

dbr:Greece dbp:member dbr:European_Union .
dbr:France dbp:member dbr:European_Union .
Phrase Mapping

For a phrase „s“ find, in the underlying KG, a set of resources which correspond to s.

General strategy

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia: <http://dbpedia.org/resource/>

dbpedia:European_Union  rdfs:label  "European Union"@en

dbpedia:European_Union  rdfs:label  "Europäische Union"@de

dbpedia:European_Union  rdfs:label  "Union européenne"@fr
Phrase Mapping

Problems

• Phrase „s“ is only similar to the „label(r)“
  • „s“ is misspelled
  • order of words in „s“ is different
• Phrase „s“ is only similar on a semantic point of view to „label(r)“
  • „s“ is an abbreviation (e.g. EU)
  • „s“ is a nickname (e.g. „Mutti“ for „Angela Merkel“)
  • „s“ is a relational phrase (e.g. „is married with“ and „spouse“)
Dealing with string similarity

- use Levenstein distance, Jaccrad distance
- use a Lucene Index
  - build in ranking based on tf-idf
  - allows fuzzy searches (searches terms similar to a given metric)
- high performant
- all out of the box
Dealing with semantic similarity

- Database with lexicalizations
  - WordNet, Wiktionary
  - Expand phrase „s“ with synonyms (hypernyms/hyponyms)

Example: EU

{European Union, European Community, EC, European Economic Community, EU, Common Market, Europe}

{europium, Eu, atomic number 63}
Dealing with semantic similarity

- Using large texts
- wordToVec/ESA
  - Associate to each word a real n-dimensional vector
  - The vector "contains" semantic information!!
  - ex1. vec(France) near to vec(spania), vec(belgium).
  - ex2. vec(queen) is near to vec(king)-vec(man)+vec(woman)
- Compare how similar words are by comparing their vectors
Disambiguation

Mostly the graph structure is used

What is the population of Europe?

dbo:populationTotal

<table>
<thead>
<tr>
<th>dbo:Europe (band)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo:Europe</td>
</tr>
<tr>
<td>dbo:Europe (dinghy)</td>
</tr>
<tr>
<td>dbo:Europe (anthem)</td>
</tr>
</tbody>
</table>
Take all triples

What is the population of Europe?

dbr:Europe (band)
dbr:Europe
dbr:Europe (dinghy)
dbr:Europe (anthem)
Query Construction

What is the population of Europe?
Query Construction

Based on the Graph Structure

What is the population of Europe?
Challenges

- Multilinguality
- Portability
- Scalability
- Robustness
- Multiple Knowledge Graphs
- Dialogues
Build a Question Answering system using QAnswer and the Qanary framework
by Dennis Diefenbach and Andreas Both

Dennis Diefenbach
QA Company
dennis.diefenbach@the-qa-company.com

Pierre Maret
Université Jean Monnet
pierre.maret@univ-st-etienne.fr

Andreas Both
Anhalt University of Applied Sciences
andreas.both@hs-anhalt.de

Aleksandr Perevalov
Anhalt University of Applied Sciences
aleksandr.perevalov@hs-anhalt.de

Paul Heinze
Anhalt University of Applied Sciences
paul.heinze@student.hs-anhalt.de

All information will be available at https://qanswer.github.io/QA-ESWC2021/
A promising industrial deployment
Main idea: We do not perform natural language processing (NLP) in the traditional sense, but we rely on the graph structure of the KB!
Running example: Give me actors born in Strasbourg.
QAnswer: how it works?

Step 1: Expansion

- actors http://dbpedia.org/ontology/starring dbpedia actor 0 PR
- actors http://dbpedia.org/property/actor dbpedia actor 0 PR
- actors http://dbpedia.org/ontology/Actor dbpedia actors 0 CL
- actors http://dbpedia.org/property/actors dbpedia actors 0 PR
- ...
- actors http://dbpedia.org/resource/Actor_(2016_film) dbpedia actor 65 IN
- actors http://dbpedia.org/resource/Actor_(UML) dbpedia actor 21 IN
- actors http://dbpedia.org/resource/Actor_(album) dbpedia actor 111 IN
- ...
- born http://dbpedia.org/ontology/hometown dbpedia born 0 PR
- born http://dbpedia.org/property/birthPlace dbpedia born 0 PR
-  
- born http://dbpedia.org/resource/Lucien-Hubert_Borne dbpedia borne 62 IN
- born http://dbpedia.org/resource/Milestone dbpedia borne 69 IN
- born http://dbpedia.org/resource/Max_Born dbpedia born 252 IN
-  
- strasbourg http://dbpedia.org/resource/Category:Strasbourg dbpedia strasbourg 33 CL
- strasbourg http://dbpedia.org/resource/Strasbourg dbpedia strasbourg 1480 IN

127 Possible Meanings
Step 2: Query Generation

PREFIX dbp: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>

• SELECT DISTINCT ?s0 where {
  VALUES ?s0 { db:Strasbourg }
} limit 1000
• SELECT DISTINCT ?s1 where {
  ?s1 dbo:birthPlace db:Strasbourg .
  ?s1 ?p1 db:Actor .
} limit 1000
• SELECT DISTINCT ?o2 where {
  ?o1 dbp:birthPlace ?o2 .
} limit 1000
• SELECT DISTINCT ?s1 where {
  ?s1 dbo:hometown db:Strasbourg .
} limit 1000
QAnswer: how it works?

Step 3: Query Ranking

```sql
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

1. SELECT DISTINCT ?s1 where {
   ?s1 dbo:birthPlace dbr:Strasbourg .
} limit 1000

2. SELECT DISTINCT ?o2 where {
   ?o1 dbp:birthPlace ?o2 .
} limit 1000

3. SELECT DISTINCT ?s1 where {
   ?s1 dbo:hometown dbr:Strasbourg .
} limit 1000

4. SELECT DISTINCT ?s0 where {
   VALUES ?s0 { dbr:Strasbourg }
} limit 1000
```
QAnswer: how it works?

Step 4: Answer Decision

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT DISTINCT ?s1 where {
    ?s1 dbo:birthPlace dbr:Strasbourg .
} limit 1000

Is this query matching the intended meaning of the user?
Step 1: Expansion

Example: Give me actors born in Strasbourg.
• Index all labels in the dataset
• Analyze every n-gram in the question and try to find a corresponding URI
• We rely on a Lucene index: fast and low memory footprint
• Stemming is important! (adapted to the language)
Step 2: Query Generation

From step 1 we have a list of resources r1,...,rn (e.g. Max_Born, Actor)

Idea:

- Travers the RDF graph starting from the identified resources in a breadth-search manner
- Find the distance between the resources
- Use the distances to build triple patterns
Step 3: Query Ranking

It's a learning to rank problem

Idea:

- Construct some features: the number of words covered in the question, the edit distance to the label, relevance score
- Construct a training dataset
- Learn

We use linear models, Coordinate Ascent
Step 4: Answer Decision

It's a binary classification problem. Fire or don't fire.

Idea:
- Reuse the same features as in the previous step
- Construct a training dataset
- Learn

We use logistic regression to have a number between 0 and 1, like a confidence.
QAnswer advantages: Multilinguality

Italia (Stato dell'Europa centro-me...) / capo di Stato

Sergio Mattarella


Summary
Hardware

We can run it on this Laptop!
16 Gb of RAM, 2,8 GHz Intel Core i7, 500 Gb
QAnswer advantages: Portability

European Union / has part

/ stores, brands / Lidl
/ categories / Muffins
QAnswer advantages: Robustness
# QAnswer Evaluation: QALD

<table>
<thead>
<tr>
<th>QA system</th>
<th>Lang</th>
<th>Type</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>full</td>
<td>0.58</td>
<td>0.46</td>
<td>0.51</td>
<td>1.08s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>key</td>
<td>0.65</td>
<td>0.44</td>
<td>0.52</td>
<td>0.81s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>full</td>
<td>0.88</td>
<td>0.29</td>
<td>0.44</td>
<td>0.22s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>key</td>
<td>0.92</td>
<td>0.29</td>
<td>0.44</td>
<td>0.18s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>full</td>
<td>0.86</td>
<td>0.28</td>
<td>0.43</td>
<td>0.31s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>key</td>
<td>0.92</td>
<td>0.29</td>
<td>0.44</td>
<td>0.20s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>full</td>
<td>0.89</td>
<td>0.29</td>
<td>0.44</td>
<td>0.20s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>key</td>
<td>0.92</td>
<td>0.29</td>
<td>0.44</td>
<td>0.16s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>full</td>
<td>0.90</td>
<td>0.29</td>
<td>0.44</td>
<td>0.18s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>key</td>
<td>0.91</td>
<td>0.29</td>
<td>0.44</td>
<td>0.17s</td>
</tr>
<tr>
<td>gAnswer [28]*</td>
<td>en</td>
<td>full</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>≈ 1 s</td>
</tr>
<tr>
<td>RTV [14]</td>
<td>en</td>
<td>full</td>
<td>0.32</td>
<td>0.34</td>
<td>0.33</td>
<td>-</td>
</tr>
<tr>
<td>Intui2 [11]</td>
<td>en</td>
<td>full</td>
<td>0.32</td>
<td>0.32</td>
<td>0.32</td>
<td>-</td>
</tr>
<tr>
<td>SINA [21]*</td>
<td>en</td>
<td>full</td>
<td>0.32</td>
<td>0.32</td>
<td>0.32</td>
<td>≈10-20s</td>
</tr>
<tr>
<td>DEANNA [25]*</td>
<td>en</td>
<td>full</td>
<td>0.21</td>
<td>0.21</td>
<td>0.21</td>
<td>≈1-50s</td>
</tr>
<tr>
<td>SWIP [19]</td>
<td>en</td>
<td>full</td>
<td>0.16</td>
<td>0.17</td>
<td>0.17</td>
<td>-</td>
</tr>
<tr>
<td>Zhu et al. [27]*</td>
<td>en</td>
<td>full</td>
<td>0.38</td>
<td>0.42</td>
<td>0.38</td>
<td>-</td>
</tr>
</tbody>
</table>
# QAnswer Evaluation: QALD

<table>
<thead>
<tr>
<th>QA system</th>
<th>Lang</th>
<th>Type</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xser [24]</td>
<td>en</td>
<td>full</td>
<td>0.72</td>
<td>0.71</td>
<td>0.72</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>key</td>
<td>0.76</td>
<td>0.40</td>
<td>0.52</td>
<td>0.32s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>full</td>
<td>0.56</td>
<td>0.30</td>
<td>0.39</td>
<td>0.46s</td>
</tr>
<tr>
<td>gAnswer [28]</td>
<td>en</td>
<td>full</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.973 s</td>
</tr>
<tr>
<td>CASIA [16]</td>
<td>en</td>
<td>full</td>
<td>0.32</td>
<td>0.40</td>
<td>0.36</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>key</td>
<td>0.92</td>
<td>0.20</td>
<td>0.33</td>
<td>0.04s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>key</td>
<td>0.92</td>
<td>0.20</td>
<td>0.33</td>
<td>0.06s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>key</td>
<td>0.92</td>
<td>0.20</td>
<td>0.33</td>
<td>0.04s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>key</td>
<td>0.92</td>
<td>0.20</td>
<td>0.33</td>
<td>0.05s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>full</td>
<td>0.90</td>
<td>0.20</td>
<td>0.32</td>
<td>0.06s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>full</td>
<td>0.92</td>
<td>0.20</td>
<td>0.32</td>
<td>0.16s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>full</td>
<td>0.90</td>
<td>0.20</td>
<td>0.32</td>
<td>0.06s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>full</td>
<td>0.86</td>
<td>0.18</td>
<td>0.29</td>
<td>0.09s</td>
</tr>
<tr>
<td>Intui3 [12]</td>
<td>en</td>
<td>full</td>
<td>0.23</td>
<td>0.25</td>
<td>0.24</td>
<td>-</td>
</tr>
<tr>
<td>ISOFT [18]</td>
<td>en</td>
<td>full</td>
<td>0.21</td>
<td>0.26</td>
<td>0.23</td>
<td>-</td>
</tr>
<tr>
<td>Hakimov [15]*</td>
<td>en</td>
<td>full</td>
<td>0.52</td>
<td>0.13</td>
<td>0.21</td>
<td>-</td>
</tr>
</tbody>
</table>
## QAnswer Evaluation: QALD

<table>
<thead>
<tr>
<th>QA system</th>
<th>Lang</th>
<th>Type</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QALD-5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xser [24]</td>
<td>en</td>
<td>full</td>
<td>0.74</td>
<td>0.72</td>
<td>0.73</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>full</td>
<td>0.56</td>
<td>0.41</td>
<td>0.47</td>
<td>0.62s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>key</td>
<td>0.60</td>
<td>0.27</td>
<td>0.37</td>
<td>0.50s</td>
</tr>
<tr>
<td>AskNow [13]</td>
<td>en</td>
<td>full</td>
<td>0.32</td>
<td>0.34</td>
<td>0.33</td>
<td>-</td>
</tr>
<tr>
<td>QAAnswer [20]</td>
<td>en</td>
<td>full</td>
<td>0.34</td>
<td>0.26</td>
<td>0.29</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>full</td>
<td>0.92</td>
<td>0.16</td>
<td>0.28</td>
<td>0.20s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>key</td>
<td>0.90</td>
<td>0.16</td>
<td>0.28</td>
<td>0.19s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>full</td>
<td>0.90</td>
<td>0.16</td>
<td>0.28</td>
<td>0.19s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>key</td>
<td>0.90</td>
<td>0.16</td>
<td>0.28</td>
<td>0.18s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>full</td>
<td>0.88</td>
<td>0.18</td>
<td>0.30</td>
<td>0.20s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>key</td>
<td>0.90</td>
<td>0.16</td>
<td>0.28</td>
<td>0.18s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>full</td>
<td>0.88</td>
<td>0.14</td>
<td>0.25</td>
<td>0.20s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>key</td>
<td>0.90</td>
<td>0.14</td>
<td>0.25</td>
<td>0.20s</td>
</tr>
<tr>
<td>SemGraphQA [2]</td>
<td>en</td>
<td>full</td>
<td>0.19</td>
<td>0.20</td>
<td>0.20</td>
<td>-</td>
</tr>
<tr>
<td>YodaQA [1]</td>
<td>en</td>
<td>full</td>
<td>0.18</td>
<td>0.17</td>
<td>0.18</td>
<td>-</td>
</tr>
<tr>
<td>QuerioDali [17]</td>
<td>en</td>
<td>full</td>
<td>0.48</td>
<td></td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>QA system</td>
<td>Lang</td>
<td>Type</td>
<td>P</td>
<td>R</td>
<td>F</td>
<td>Time</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>UTQA [23]</td>
<td>en</td>
<td>full</td>
<td>0.82</td>
<td>0.69</td>
<td>0.75</td>
<td>-</td>
</tr>
<tr>
<td>UTQA [23]</td>
<td>es</td>
<td>full</td>
<td>0.76</td>
<td>0.62</td>
<td>0.68</td>
<td>-</td>
</tr>
<tr>
<td>UTQA [23]</td>
<td>fs</td>
<td>full</td>
<td>0.70</td>
<td>0.61</td>
<td>0.65</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>full</td>
<td>0.62</td>
<td>0.40</td>
<td>0.49</td>
<td>0.93s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>key</td>
<td>0.52</td>
<td>0.32</td>
<td>0.40</td>
<td>0.68s</td>
</tr>
<tr>
<td>SemGraphQA [2]</td>
<td>en</td>
<td>full</td>
<td>0.70</td>
<td>0.25</td>
<td>0.37</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>full</td>
<td>0.95</td>
<td>0.17</td>
<td>0.29</td>
<td>0.12s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>key</td>
<td>0.96</td>
<td>0.17</td>
<td>0.29</td>
<td>0.08s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>full</td>
<td>0.91</td>
<td>0.16</td>
<td>0.27</td>
<td>0.37s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>key</td>
<td>0.96</td>
<td>0.17</td>
<td>0.29</td>
<td>0.12s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>full</td>
<td>0.96</td>
<td>0.17</td>
<td>0.29</td>
<td>0.16s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>key</td>
<td>0.96</td>
<td>0.17</td>
<td>0.29</td>
<td>0.07s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>full</td>
<td>0.96</td>
<td>0.17</td>
<td>0.29</td>
<td>0.19s</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>key</td>
<td>0.96</td>
<td>0.17</td>
<td>0.29</td>
<td>0.19s</td>
</tr>
</tbody>
</table>
## QAnswer Evaluation: QALD

<table>
<thead>
<tr>
<th>QA System</th>
<th>Lang</th>
<th>Type</th>
<th>Total</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>Runtime</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>full</td>
<td>100</td>
<td>0.37</td>
<td>0.39</td>
<td>0.37</td>
<td>1.68s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>en</td>
<td>key</td>
<td>100</td>
<td>0.35</td>
<td>0.38</td>
<td>0.35</td>
<td>0.80s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>key</td>
<td>100</td>
<td>0.31</td>
<td>0.32</td>
<td>0.31</td>
<td>0.45s</td>
<td>-</td>
</tr>
<tr>
<td>Sorokin et al. [39]</td>
<td>en</td>
<td>full</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>0.29</td>
<td>-</td>
<td>[39]</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>key</td>
<td>100</td>
<td>0.27</td>
<td>0.28</td>
<td>0.27</td>
<td>1.13s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>key</td>
<td>100</td>
<td>0.27</td>
<td>0.30</td>
<td>0.27</td>
<td>1.14s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>fr</td>
<td>full</td>
<td>100</td>
<td>0.27</td>
<td>0.31</td>
<td>0.27</td>
<td>1.05s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>es</td>
<td>full</td>
<td>100</td>
<td>0.24</td>
<td>0.26</td>
<td>0.24</td>
<td>0.65s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>de</td>
<td>full</td>
<td>100</td>
<td>0.18</td>
<td>0.20</td>
<td>0.18</td>
<td>0.82s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>full</td>
<td>100</td>
<td>0.19</td>
<td>0.20</td>
<td>0.18</td>
<td>1.00s</td>
<td>-</td>
</tr>
<tr>
<td>WDAqua-core1</td>
<td>it</td>
<td>key</td>
<td>100</td>
<td>0.17</td>
<td>0.18</td>
<td>0.16</td>
<td>0.44s</td>
<td>-</td>
</tr>
</tbody>
</table>

QALD-7 task 4, training dataset
## QAnswer Evaluation: Simple Questions

<table>
<thead>
<tr>
<th>QA System</th>
<th>Lang</th>
<th>Type</th>
<th>Total</th>
<th>Accuracy</th>
<th>Runtime</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lukovnikov et al.</td>
<td>en</td>
<td>full</td>
<td>21687</td>
<td>0.712</td>
<td>-</td>
<td>[29]</td>
</tr>
<tr>
<td>Golub and He</td>
<td>en</td>
<td>full</td>
<td>21687</td>
<td>0.709</td>
<td>-</td>
<td>[21]</td>
</tr>
<tr>
<td>Yin et al.</td>
<td>en</td>
<td>full</td>
<td>21687</td>
<td>0.683</td>
<td>-</td>
<td>[48]</td>
</tr>
<tr>
<td>Bordes et al.</td>
<td>en</td>
<td>full</td>
<td>21687</td>
<td>0.627</td>
<td>-</td>
<td>[4]</td>
</tr>
<tr>
<td>Dai et al.*</td>
<td>en</td>
<td>full</td>
<td>21687</td>
<td>0.626</td>
<td>-</td>
<td>[7]</td>
</tr>
<tr>
<td><strong>WD Aqua-core1</strong></td>
<td>en</td>
<td>full</td>
<td>21687</td>
<td><strong>0.571</strong></td>
<td><strong>2.1 s</strong></td>
<td></td>
</tr>
</tbody>
</table>
### QAnswer Evaluation: LcQuad

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Lang</th>
<th>Type</th>
<th>Total</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC-QuAD</td>
<td>en</td>
<td>full</td>
<td>5000</td>
<td>0.59</td>
<td>0.38</td>
<td>0.46</td>
<td>1.5 s</td>
</tr>
<tr>
<td>WDAquaCore0Questions</td>
<td>mixed</td>
<td>mixed</td>
<td>689</td>
<td>0.79</td>
<td>0.46</td>
<td>0.59</td>
<td>1.3 s</td>
</tr>
</tbody>
</table>
From QAnswer to Qanary

- not everything is solvable inside of the query builder QAnswer
  - Remark: the same is true for any QA system and any QA component
  - Our challenge as scientist:
    - implement a high-quality QA system for the considered knowledge domain

- Example: adding additional functionality
  - User’s question: “Where and when was Johann Sebastian Bach born?”
    - Possible solution: transformation into two interpretations
      - When was Johann Sebastian Bach born? → 31 March 1685 (Gregorian Calendar)
      - Where was Johann Sebastian Bach born? → Eisenach
  - Follow-up: Give me the corresponding state. → Saxe-Eisenach

(https://www.wikidata.org/wiki/Q1339)

(https://www.wikidata.org/wiki/Q696651)
Motivation of the Qanary approach

- Observations:
  - Implementing a QA system is cumbersome and time-consuming
    → As researchers, we would like to invest the time into novel methods (and not into engineering)

- Vision:
  - establish an infrastructure in which the state-of-the-art QA components can be easily integrated, run, and evaluated.
    → As researchers, we can focus on novel methods and also reuse typical component to complete a QA system

Motivation Qanary Approach

- **Goal of the Qanary methodology:** Establish a component-oriented framework
  - Each sub-task of a QA system is considered to be a QA component

- **Features:**
  - Flexible w.r.t. the size and purpose of the QA components
  - Plug & play behavior of the QA components
  - Knowledge-driven approach (RDF-based information architecture)

- **Your advantages:**
  - Open source implementation
  - Add additional functionality
  - Reuse previously implemented components
    - [https://github.com/WDAqua/Qanary-question-answering-components](https://github.com/WDAqua/Qanary-question-answering-components)
    - Remark: QAnswer is an example of a Query Builder component

→ Make building Question Answering systems easier
Example 1 (Wikidata KG)

- QA system to answer questions like: *When and where was <Person> born?*
- Implement 3 components:
  - Recognize person entity in Wikidata (NED/NER)
  - Create SPARQL query using the QAnswer API
  - Query Wikidata using compute SPARQL query to fetch results

- Example: *Where and when was Ada Lovelace born?*

The mentioned components are available at: [https://github.com/WDAqua/Qanary-question-answering-components](https://github.com/WDAqua/Qanary-question-answering-components)
The Qanary Approach: Adding additional components to the QA system for custom functionality

Task: Create a Question Answering System capable of analyzing natural-language questions

"What is the real name of Batman?"

Qanary triplestore
stores all relevant data known about the current question

all component analyze the given question
- fetch all available information
- store created annotations of a question

Input

Output

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
  dbr:Batman foaf:name ?anser .
  ...
}

Qanary framework

Framework is managing the interaction between the Qanary tasks (i.e., orchestrating the microservices) following a message-driven architecture

Input

Output

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
  dbr:Batman foaf:name ?anser .
  ...
}

Qanary triplestore
stores all relevant data known about the current question

Input

Output

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
  dbr:Batman foaf:name ?anser .
  ...
}
Actually computed question for "What is the real name of Batman?"

SELECT * WHERE {

    ?resource foaf:name ?answer . # real name of superhero

    ?resource rdfs:label ?label . # get the character name of the superhero

    FILTER(LANG(?label) = "en") . # only English names

    ?resource dct:subject dbr:Category:Superhero_film_characters . # only superheros

    FILTER(! strStarts(LCASE(?label), LCASE(?answer))). # filter starting with the same name

    VALUES ?resource { dbr:Batman } . # only for this specific resource

}
Example 2 (DBpedia KG)

- QA system to answer questions like: \textit{What is the real name of <superhero character>?}
- Implement 3 components:
  - Recognize Name Entity in DBpedia
  - Create SPARQL query by a custom component
  - Query DBpedia using compute SPARQL query to fetch results

- Example: \textit{What is the real name of Batman?}

Qanary triplestore stores all relevant data known about the current question

chatbot-like UI (demo installation)
https://webengineering.ins.hs-anhalt.de:43712/

The mentioned components are available at:
https://github.com/WDAqua/Qanary-question-answering-components
Quality measurement

- Qanary triplestore is global memory processes
  - Each request is stored into a graph of the triplestore
  - Each component stores the computed information into the same graph
    - Vocabulary: Web Annotation Data Model
      - W3C Recommandation: https://www.w3.org/TR/annotation-model/
- Enabled quality measurement via SPARQL queries:
Vision: Let’s build QA systems together

- Long-term agenda:
  - Establish automatically a QA system for any domain
  - Find the best possible QA component for a given task
  - Optimize the QA quality automatically

Summary

- Qanary framework enables composition of components
  - Remark: QAnswer an example of a Query Builder component
- You can control/trace the behavior in the NLU / query construction process.
- You can combine different implementations.
- You can exchange implementations to optimize the quality.
- The Qanary ecosystem provides several tools to make your life as as QA researcher easier:
  - Easy-to-use framework
  - Plug & play of typical components
  - SPARQL access to process information
  - Options to microbenchmark components / your implementations
  - Ready-to-use chatbot-like Web user interface
Wrap-up

- QAnswer is powerful Question Answering Query Engine
  - evaluates graph structure of KG
  - Matches entities and corresponding graph patterns
  - Fast and scalable
- Qanary provides a methodology
  - easy-to-use framework for building QA systems
  - already reusable set of QA components
  - Easy quality measurement

After the lunch break 🍽️:

1. Learn to use QAnswer
2. Learn to take advantage of the Qanary ecosystem
QAnswer: Examples & Hands on
by Dennis Diefenbach and Pierre Maret

After the lunch break

Dennis Diefenbach
QA Company
dennis.diefenbach@the-qa-company.com

Pierre Maret
Université Jean Monnet
pierre.maret@univ-st-etienne.fr

Andreas Both
Anhalt University of Applied Sciences
andreas.both@hs-anhalt.de

Aleksandr Perevalov
Anhalt University of Applied Sciences
aleksandr.perevalov@hs-anhalt.de

Paul Heinze
Anhalt University of Applied Sciences
paul.heinze@student.hs-anhalt.de

All information will be available at https://qanswer.github.io/QA-ESWC2021/
Example 1: HR

Querying a Knowledge Graph containing information about Employees.
Example 2: EU Knowledge Graph

Querying a Knowledge Graph of the European Commission that contains data about:
- countries / capitals
- head of states
- European institutions
- buildings by the commission
- more than 700.000 projects financed by the European Commission
Example 3: Product Catalogue

Querying a Knowledge Graph of products coming from openFoodFacts 1.7 M products with information about:

- brands and stores
- countries
- ingredients
- nutrients
Create our own Digital Twin using QAnswer!
The Question Answering Framework

Qanary

A short hands-on introduction by Andreas Both, Aleksandr Perevalov, Paul Heinze

Please use the Slack channel to ask questions: https://app.slack.com/client/T023597R6E4/C024B8B03EC

Dennis Diefenbach
QA Company
dennis.diefenbach@the-qa-company.com

Pierre Maret
Université Jean Monnet
pierre.maret@univ-st-etienne.fr

Andreas Both
Anhalt University of Applied Sciences
andreas.both@hs-anhalt.de

Aleksandr Perevalov
Anhalt University of Applied Sciences
aleksandr.perevalov@hs-anhalt.de

Paul Heinze
Anhalt University of Applied Sciences
paul.heinze@student.hs-anhalt.de

All information will be available at https://qanswer.github.io/QA-ESWC2021/
Overview

- core was developed in the Horizon 2020 ITN WDAqua (2015-2018)
- available as open-source
- consists of two main perspectives
  - **Qanary** framework:
    - reference implementation of central components (Java, Spring Framework)
    - manifests the Qanary approach to develop a knowledge-driven Question Answering system
    - available at [https://github.com/WDAqua/Qanary](https://github.com/WDAqua/Qanary)
  - **Qanary** components:
    - all of them follow the Qanary implementation concept
    - solve specific tasks (e.g., language classification, intent detection, NED, query building)
    - can be implemented using any programming language (mainly Java, Python)
    - some are available at [https://github.com/WDAqua/Qanary-question-answering-components](https://github.com/WDAqua/Qanary-question-answering-components)
The Qanary Approach: Adding additional components to the QA system for custom functionality

Task: Create a Question Answering System capable of analyzing natural-language questions

"What is the real name of Batman?"

Qanary triplestore
stores all relevant data known about the current question

Qanary framework
managing the interaction between the Qanary tasks (i.e., orchestrating the microservices) following a message-driven architecture

Input

Query Builder

Entity Recognition

Relation Detection

Intent Classification

Output

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT * WHERE {
  dbr:Batman foaf:givenName ?label .
}

all components analyze the given question
- fetch all available information
- store created annotations of a question
## Contributions of the Qanary ecosystem

### Qanary approach
- exchangeable, reusable components
  - real plug & play micro services
  - supports microbenchmarking
- flexible orchestration
  - global quality improvement / self-optimizing
  - auto-wiring of Question Answering components
- knowledge-driven approach (Qanary triplestore)
  - descriptive data access
  - traceable information
  - reasoning possible

### Impact
- rapid and efficient system development
- supports agile software development
- easy to measure systems quality
- native support for AI-driven development
- flexible usage:
  - can be used to develop new sub-systems and
  - to develop complete Question Answering systems

### Research Vision
- integrate all best-of-breed approaches
- establish domain-agnostic automatic optimization of Question Answering quality
QA system to answer questions like: *What is the real name of a superhero character?*

Implement 3 components:

- Recognize Name Entity in DBpedia
- Create SPARQL query by a custom component
- Query DBpedia using compute SPARQL query to fetch results

Example: *What is the real name of Batman?*

Qanary triplestore stores all relevant data known about the current question.
Full process: What is the real name of a superhero character?

- **Qanary framework** receives question
  a. question is stored in the Qanary triplestore
  b. Calls 1st Qanary component: **NED-DBpediaSpotlight**
     1. Fetch the question from the Qanary triplestore
     2. Call the DBpedia Spotlight service for the given question
     3. Store Named Entity annotations into the Qanary triplestore
  c. Calls 2nd Qanary component: **QueryBuilderSimpleRealNameOfSuperHero**
     1. Fetch the Named Entity annotations from the Qanary triplestore
     2. Create DBpedia-related SPARQL query
     3. Store SPARQL query into the Qanary triplestore
  d. Calls 3rd Qanary component: **SparqlExecuter**
     1. Fetch the SPARQL query from the Qanary triplestore
     2. Fetch results of SPARQL query from the DBpedia endpoint
     3. Store query results (JSON) into the Qanary triplestore

Please use the Slack channel to ask questions: [https://app.slack.com/client/T023597R6E4/C024B8B03EC](https://app.slack.com/client/T023597R6E4/C024B8B03EC)
Generalized Process

- Every component follows a **3-step process**

  1. Fetch required data from the Qanary triplestore using SPARQL:
     
     ```sparql
     SELECT * FROM <inGraph> WHERE { ... }
     ```

  2. Compute new information about the current question

  3. Store new information into the Qanary triplestore using SPARQL:
     
     ```sparql
     INSERT { GRAPH <outGraph> {
     ...
     } }
     ```

Each component is called by the Qanary framework via a predefined RESTful interface.

**3 parameters are provided to all components:**

- **endpoint**
  - URI of the Qanary triplestore

- **URI inGraph**
  - URI of the graph with information from the previous components

- **URI outGraph**
  - URI of the graph where the newly computed data has to be stored

Please use the Slack channel to ask questions: [https://app.slack.com/client/T023597R6E4/C024B8B03EC](https://app.slack.com/client/T023597R6E4/C024B8B03EC)
Your turn (I): Implement a simple Question Answering system using pre-existing component

Example: What is the real name of Batman?

1. goto: https://webengineering.ins.hs-anhalt.de:43740/startquestionansweringwithtextquestion
2. insert the question
3. activate the 3 components and drag & drop them into the correct order:
   ○ NED-DBpediaSpotlight
   ○ QueryBuilderSimpleRealNameOfSuperHero
   ○ SparqlExecutor
4. click on the blue/green button
5. as a result you see the access information of the current question answering process
6. goto the Stardog studio: https://stardog.studio/
   ○ click “connect” and insert https://webengineering.ins.hs-anhalt.de:40159
     ■ use the credentials “admin” and “admin”
   ○ select the database “qanary”
   ○ insert a SPARQL query you see on the right-hand side
   ○ this will show you the annotations that were computed by Qanary process
7. use the SPARQL studio (see following slides) to search for other oa:Annotations

Now, you know how information are stored inside of the Qanary triplestore

# get the answer as JSON
PREFIX oa: <http://www.w3.org/ns/openannotation/core/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX qa: <http://www.wdaqua.eu/qa#>
SELECT * FROM <ADD HERE THE URI SHOWN AS Qanary question analysis outgraph>
WHERE {
  ?s rdf:type qa:AnnotationOfAnswerJson.
  ?s oa:hasBody ?body.
  ?body rdf:value ?json .}

# find all annotations computed in your process
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX oa: <http://www.w3.org/ns/openannotation/core/>
PREFIX qa: <http://www.wdaqua.eu/qa#>
SELECT * FROM <ADD HERE THE URI SHOWN AS Qanary question analysis outgraph>
WHERE {
  ?annotationId rdf:type ?type.
  ?annotationId oa:hasBody ?body.
  ?annotationId oa:hasTarget ?target.
Your turn (II): Implement a simple Question Answering system using a self-developed component

- You need to decide for yourself what programming language you would like to use.
- Suggestion to create a very easy QA system:
  - Implement a QA system computing the “date of death” for a person using the Wikipedia KG
  - Reuse the OpenTapiocaNED component
  - Reuse the WikidataQueryExecutor component

Hence, you just need to implement the Query Builder component:
Your turn (II): Implement a simple Question Answering system using a self-developed component

- Process:
  - prepare the workspace:
    - you have three options (see following slides)
  - implement your Qanary component (preferred: in Java or Python)
    - see following slides for a description
    - you need to follow 3 steps:
      1. fetch the question from the Qanary triplestore
      2. create your own SPARQL query that should be used for requesting the answer from Wikidata
      3. store the created annotation into the Qanary triplestore
  - you might re-use the following examples as templates (they follow the described 3-step process):
    - Java:
    - Python:

- deploy/start your Qanary component and QA process
  - you need to register to the Qanary framework instance you would like to use (see the linked tutorials on the following slides)
  - open the Qanary framework URI (the one you would like to use, see following slides)
    - check if the components are available
  - goto https://webengineering.ins.hs-anhalt.de:43712/ open configuration dialog (right-upper corner)
    - add your Qanary framework endpoint
    - add your list of names of your components

Please use the Slack channel to ask questions: https://app.slack.com/client/T023597R6E4/C024B8B03EC
Let’s build a QA system using the Qanary ecosystem

Preparation:

- a triplestore: we prepared everything for the Stardog triplestore (hence, suggest to use it)
  - https://www.stardog.com/get-started/
- the Qanary framework component
  - https://github.com/WDAqua/Qanary
- the Qanary components that you would like to reuse
  - https://github.com/WDAqua/Qanary-question-answering-components

Implement your own Qanary component:

- Implement you own Qanary component, we provide 3 easy-to-use options described on GitHub:
  - Java using Maven archetypes:
  - Python using a prepared library:
  - Python using plain Flask:
three options are available to prepare your workspace:

- **Option 1:**
  - your PC is accessible from the Internet
  - just reuse our demo installation of the Qanary framework

- **Option 2:**
  - you would like to run everything on your own machine
  - **Option 2.1**
    - you start required components using Docker
  - **Option 2.2**
    - you install the required components locally

Any programming language might be used to implement Qanary components.

However, completely prepared are tools for Java and Python.
Option 1: Running everything using pre-installed Web services

- **A Stardog triplestore** is already available at [http://webengineering.ins.hs-anhalt.de:40159/](http://webengineering.ins.hs-anhalt.de:40159/)
  - see [http://webengineering.ins.hs-anhalt.de:40159/qanary](http://webengineering.ins.hs-anhalt.de:40159/qanary) for predefined vocabularies
  - credentials: admin / admin
  - optional: SPARQL editor for Stardog: [https://stardog.studio](https://stardog.studio)
    - sign in for free to use the cloud-based UI
- **A Qanary framework** component is already available at [https://webengineering.ins.hs-anhalt.de:43740](https://webengineering.ins.hs-anhalt.de:43740)
  - currently available components: [https://webengineering.ins.hs-anhalt.de:43740/#/applications](https://webengineering.ins.hs-anhalt.de:43740/#/applications)
- **A Qanary chatbot-like Web UI** is already available at [https://webengineering.ins.hs-anhalt.de:43712/](https://webengineering.ins.hs-anhalt.de:43712/)
- **A Qanary developer’s view** is available at [https://webengineering.ins.hs-anhalt.de:43740/startquestionansweringwithtextquestion](https://webengineering.ins.hs-anhalt.de:43740/startquestionansweringwithtextquestion) providing information about the current graph where the annotations of your question are stored
- **Remark:** If your machine is not running on a public IP address, then your self-developed locally installed Qanary components would not be reachable. The easiest option to overcome this issue is to use `ngrok`:
  - Expose your port to public: `./ngrok http PORT_NUMBER`
  - Copy the public link provided by ngrok to the component’s configuration
- **start** your QA component
- goto you Qanary framework URI: **check** is your component shown and marked as running (green)
- start your QA process (see previous slides)
Option 2.1: Locally run the required component using Docker

- You might reuse the Stardog service as described in Option 1 or you start your local
  - `docker run -it -v ~/stardog-home/:/var/opt/stardog -p 5820:5820 stardog/stardog`
  - you will be asked for an email address to receive a free license automatically
- The Qanary pipeline Docker image is available on Docker Hub: [https://hub.docker.com/repository/docker/qanary/qanary-pipeline](https://hub.docker.com/repository/docker/qanary/qanary-pipeline)
  - `docker pull qanary/qanary-pipeline:2.4.0`
  - `docker run --net host qanary/qanary-pipeline:2.4.0`
  - The pipeline will be available on [http://localhost:8080](http://localhost:8080) (using the default configuration)
  - You can set a custom port with an environment variable
    - `docker run -e SERVER_PORT=<port> --net qanary/qanary-pipeline:2.4.0`
- Start and connect one or many Qanary components
  - all components are available as Docker images: [https://hub.docker.com/search?q=qanary&type=image](https://hub.docker.com/search?q=qanary&type=image)
    - `docker run --net host <component>`
  - Again, you can specify the port using the same approach as for the pipeline
  - If you set a custom port for the pipeline you need to tell the component
    - `docker run -e SPRING_BOOT_ADMIN_URL=http://localhost:<port> --net host <component>`
- implement your own Qanary component (see the description on the previous slides)
- start your QA component
- goto your Qanary framework URI: check is your component shown and marked as running (green)
- start your QA process (see previous slides)
Option 2.2: Running the required components locally

- You might reuse the **Stardog service** as described in Option 1 or Option 2.1 or you install Stardog locally from scratch:
  - [https://www.stardog.com/get-started/](https://www.stardog.com/get-started/)
- the whole process is described at [https://github.com/WDAqua/Qanary/wiki/Qanary-tutorial:-How-to-build-a-trivial-Question-Answering-pipeline](https://github.com/WDAqua/Qanary/wiki/Qanary-tutorial:-How-to-build-a-trivial-Question-Answering-pipeline)

  - in a nutshell: You need to set up the **Qanary framework**
    - Requirements:
      - Java JDK 8 (or newer)
      - Maven 3.5+
    - Clone the repository at [https://github.com/WDAqua/Qanary](https://github.com/WDAqua/Qanary)
      - `git clone git@github.com:WDAqua/Qanary.git`
    - Build the project with:
      - `mvn clean install`
    - Start the pipeline with:
      - `java -jar target/qa.pipeline-2.4.0.jar`
    - Build and run the Qanary components following the same approach: [https://github.com/WDAqua/Qanary-question-answering-components](https://github.com/WDAqua/Qanary-question-answering-components)

- implement your Qanary component
- goto your Qanary framework URI: **check** is your component shown and marked as running (green)
- start your QA process (see previous slides)
You have learned

● how a Linked Data driven Question Answering framework works
● implemented a simple QA component
  ○ many pre-implemented components are available. They all follow the same process: https://github.com/WDAqua/Qanary-question-answering-components

● your next steps might be:
  ○ easy: implement some more components
  ○ easy: replace the rule-based approach of your QA component by an ML-based implementation
  ○ medium: add additional functionality to the Qanary framework
    ■ for example, dialogue management for a chatbot behavior
  ○ advanced: optimize your Question Answering system while integrating additional components (for example, several NED components)
  ○ advanced: optimize your complete QA system by analyzing the data stored in the Qanary triplestore and determine optimized combinations of your QA components
    ■ thereafter, implement new components improving the quality

Please use the Slack channel to ask questions: https://app.slack.com/client/T023597R6E4/C024B8B03EC
References

- 2014 – Both, Andreas, Viet Nguyen, Mandy Keck, Dietrich Kammer, Rainer Groh, and Dana Henkens. Get inspired: A visual divide and conquer approach for motive-based search scenarios. In 13th International Conference WWW/INTERNET (ICWI). Received Best Paper Award. → long version (journal)
Wrap-up of our tutorial

You have learned:

- how to configure and use the QAnswer Query Builder
- how to use the Qanary methodology to build a QA system

Thanks for your participation.

All information will be available at https://qanswer.github.io/QA-ESWC2021/
Open Discussion
Last words

Dennis Diefenbach
QA Company
dennis.diefenbach@the-qa-company.com

Pierre Maret
Université Jean Monnet
pierre.maret@univ-st-etienne.fr

Andreas Both
Anhalt University of Applied Sciences
andreas.both@hs-anhalt.de

Aleksandr Perevalov
Anhalt University of Applied Sciences
aleksandr.perevalov@hs-anhalt.de

Paul Heinze
Anhalt University of Applied Sciences
paul.heinze@student.hs-anhalt.de

All information will be available at https://qanswer.github.io/QA-ESWC2021/