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

QA Tutorial @ESWC 2021

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
  - 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/

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

Keynote and introduction into the field of Question Answering

general principles and variations

by Pierre Maret

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

QA Tutorial @ESWC 2021

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

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

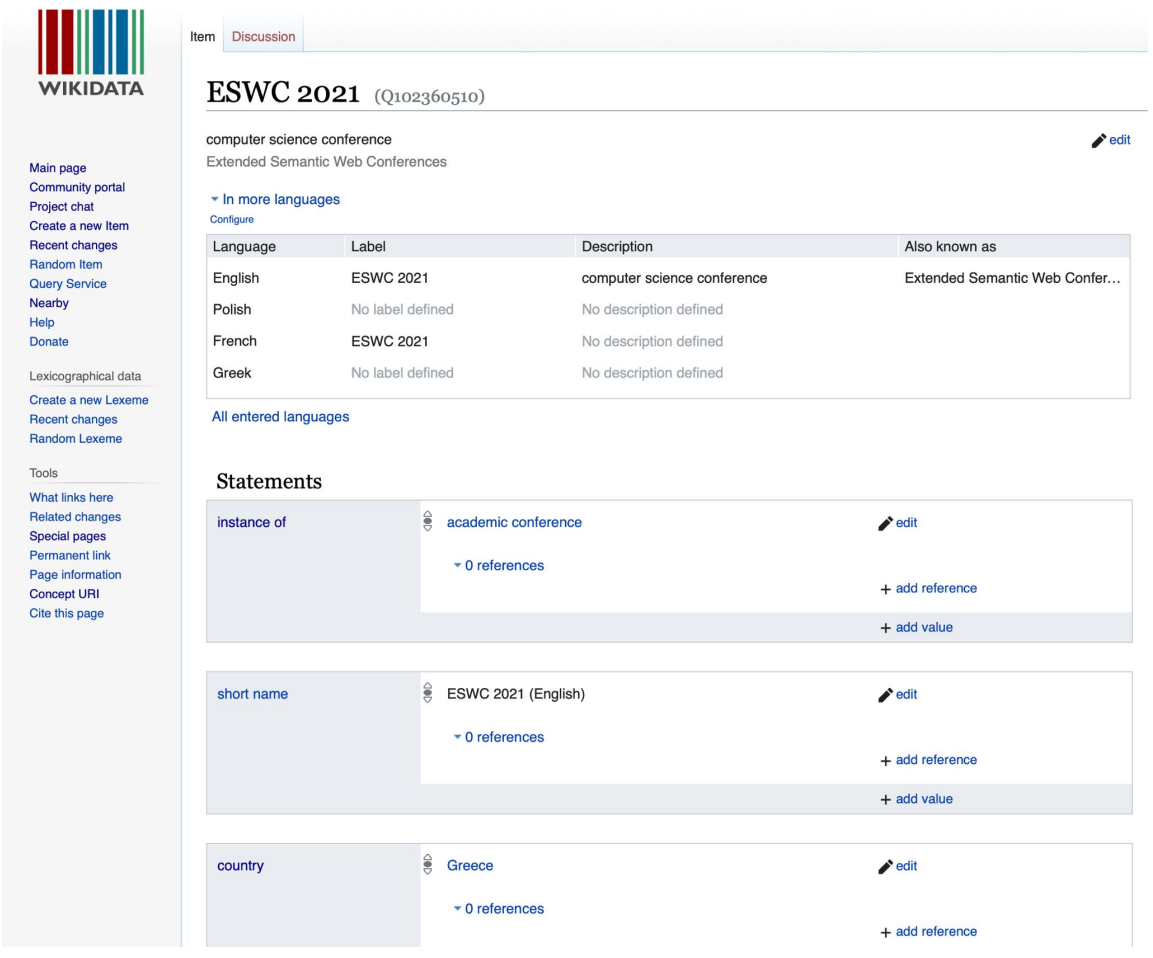
Aleksandr Perevalov
Anhalt University of Applied Sciences
aleksandr.perevalov@hs-anhalt.de

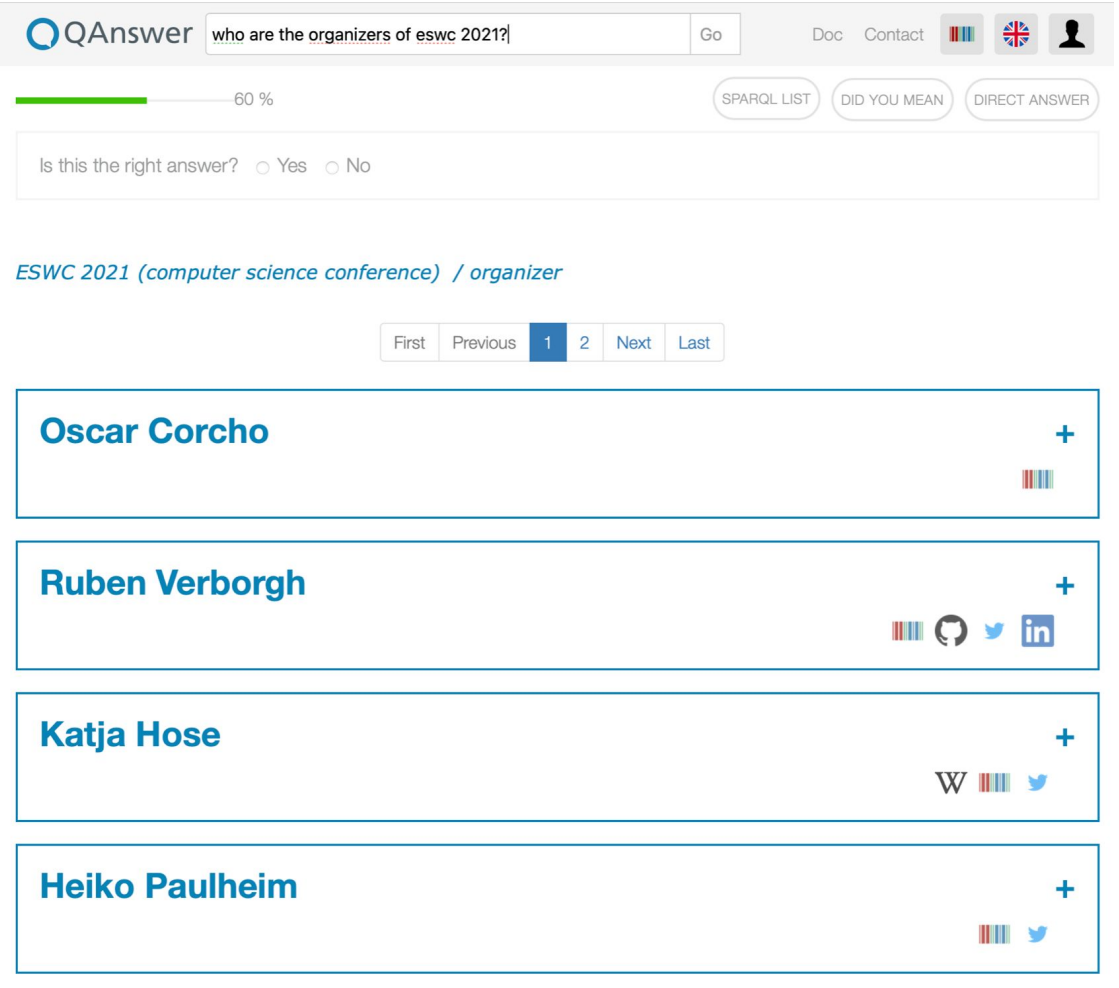
Paul Heinze
Anhalt University of Applied Sciences
paul.heinze@student.hs-anhalt.de

QA Tutorial @ESWC 2021

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

What is Question Answering over Knowledge Graphs?





The Question Answering Process

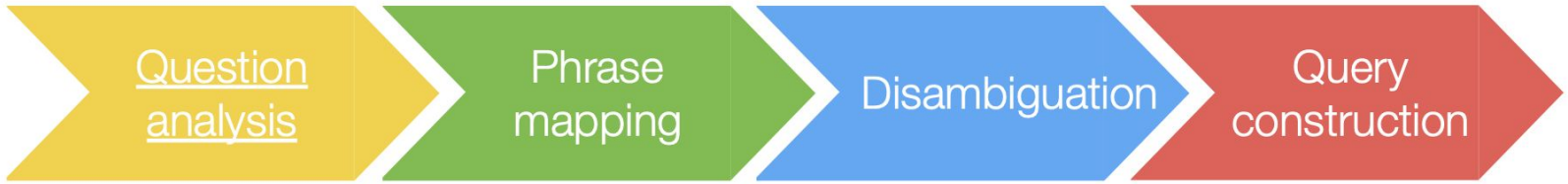
Question Analysis

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?

Question Analysis

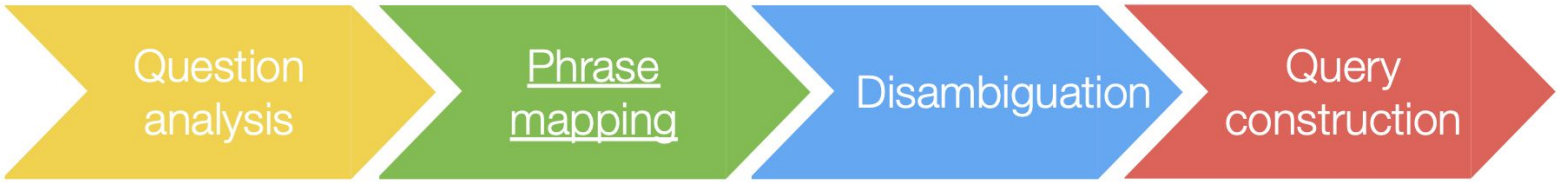


Collect informations which can be deduced considering only the syntax of the question

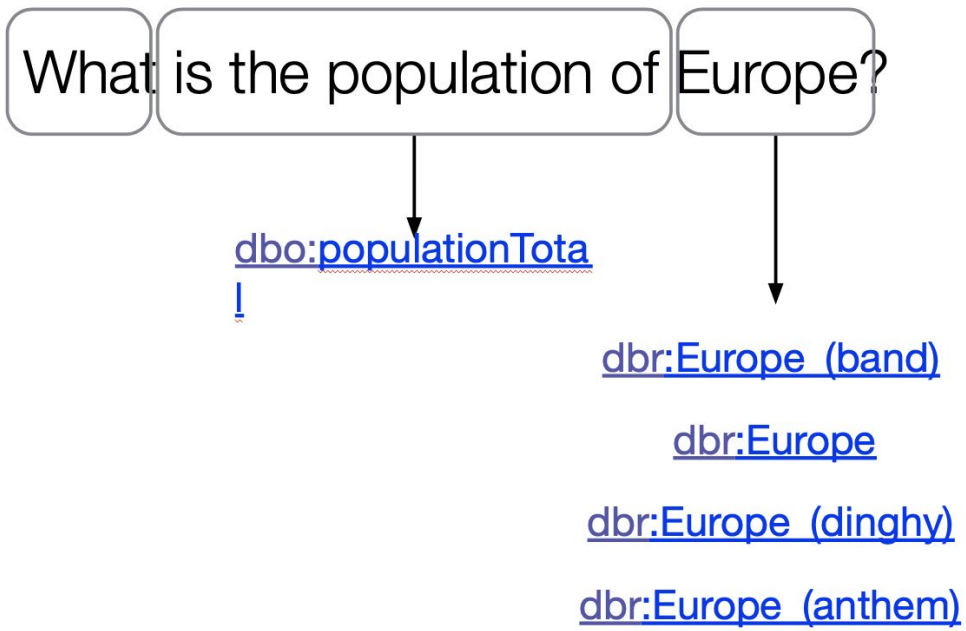
- Type of the question
- NE recognition
- Identify the properties
- Identify dependencies

What | is the population of | Europe?

Phrase Mapping

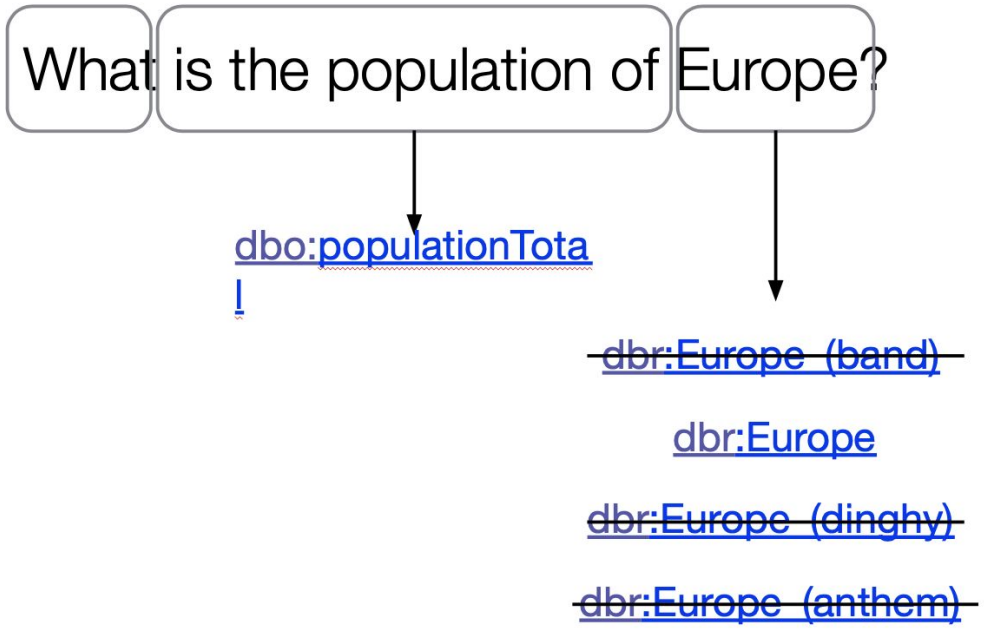


Mapping a phrase to possible resources in the underling ontology

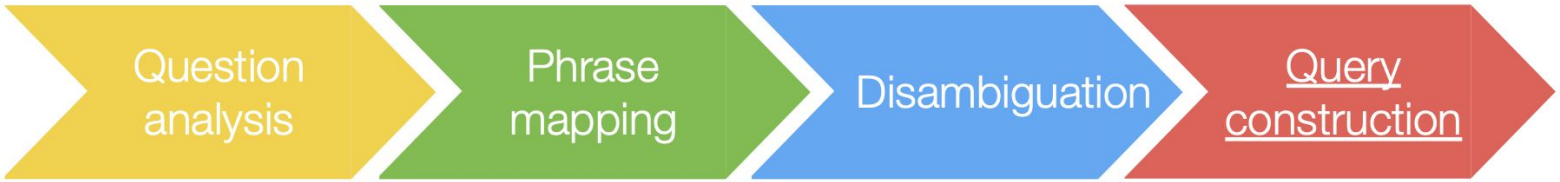


Disambiguation

Mapping a phrase to possible resources in the underling ontology



Query Construction



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

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

Question Analysis

NE recognition

Who is the director of the Lord of the Ring?

- Use a NE recognition tool
  - Problem: Standford 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?

Question Analysis

use POS Tagging

WRB	VBD	DT	NNP	NNP	VBN	.
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 | VP | VW*P
V = verb particle? adverb?
W = (noun | adjective | adverb | pronoun | determiner)
P = (preposition | particle | inf. marker)

Question Analysis

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
```

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“)

Phrase mapping

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

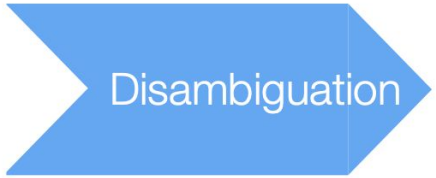
{European Union, European Community, EC, European Economic Community, EU, Common Market, Europe}

{europium, Eu, atomic number 63}

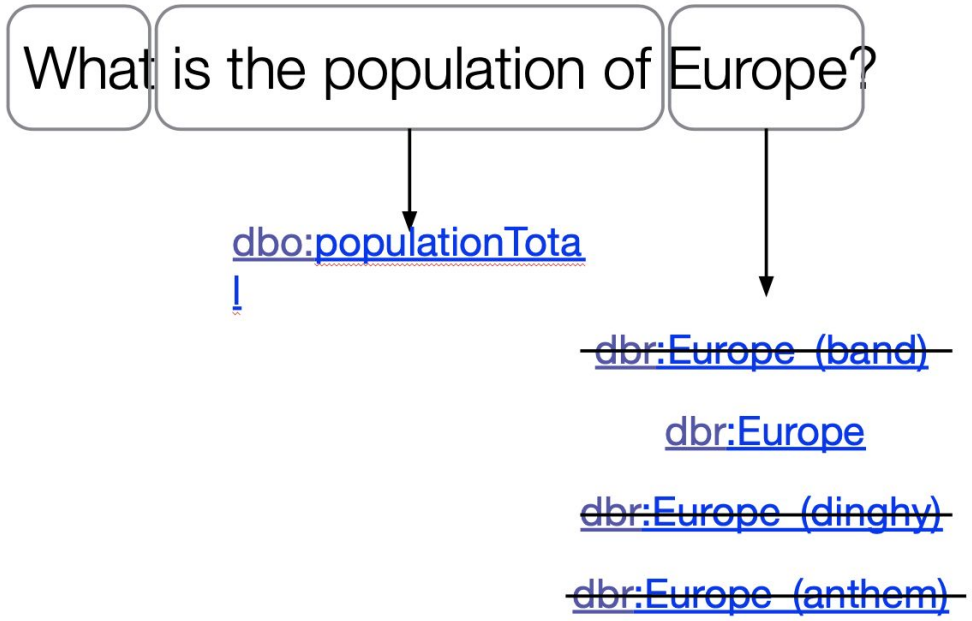
Dealing with semantic similarity

- 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(spain),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



Query Construction

Take all triples

What | is the population of | Europe?

dbr:Europe_(band)

dbr:Europe

dbr:Europe_(dinghy)

dbr:Europe_(anthem)

?p ?
o

Query Construction

Templates

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

After the coffee break ☕, continue at 11:25h

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

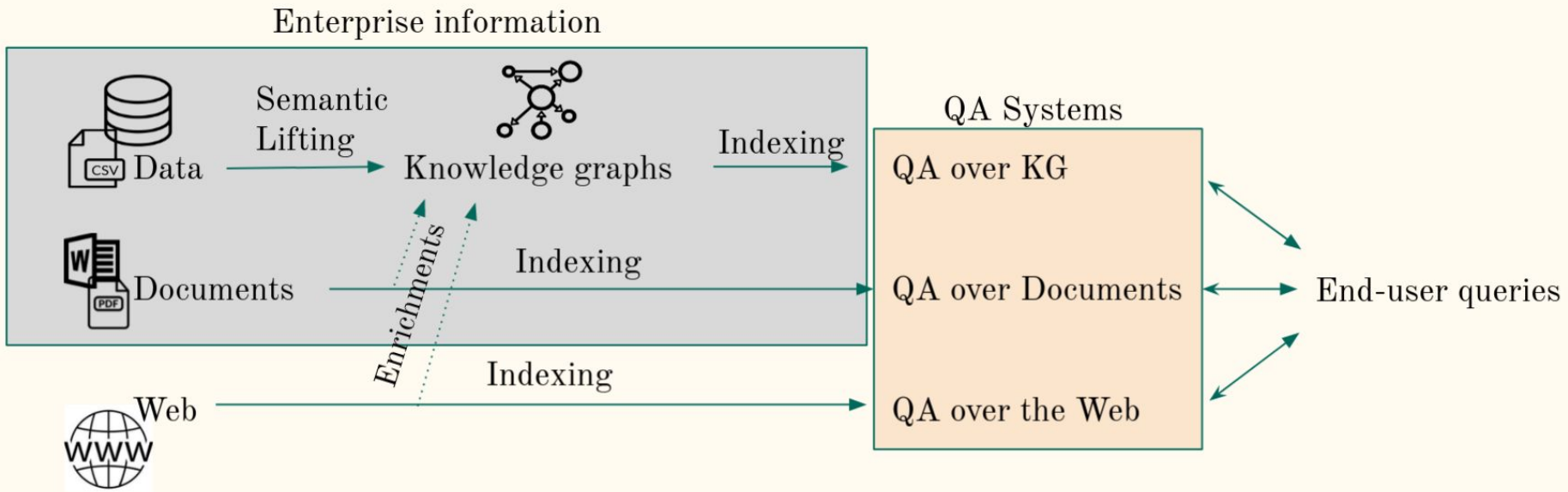
Aleksandr Perevalov
Anhalt University of Applied Sciences
aleksandr.perevalov@hs-anhalt.de

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

QA Tutorial @ESWC 2021

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

A promising industrial deployment



QAnswer: how it works?

Main idea: We do not perform natural language processing (NLP) in the traditional sense, but we rely on the graph structure of the KB!

QAnswer: how it works?

Running example: Give me actors born in Strasbourg.

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

- SELECT DISTINCT ?s0 where {
  VALUES ?s0 { dbr:Strasbourg }
  } limit 1000
- SELECT DISTINCT ?s1 where {
  ?s1 dbo:birthPlace dbr:Strasbourg .
  ?s1 ?p1 dbr:Actor .
  } limit 1000
- SELECT DISTINCT ?o2 where {
  dbr:Actor_(1993_film) dbo:starring ?o1 .
  ?o1 dbp:birthPlace ?o2 .
  } limit 1000
- SELECT DISTINCT ?s1 where {
  ?s1 dbo:hometown dbr:Strasbourg .
  } limit 1000

500 Possible Interpretations

Step 3: Query Ranking

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

1. SELECT DISTINCT ?s1 where {
   ?s1 dbo:birthPlace dbr:Strasbourg .
   ?s1 ?p1 dbr:Actor .
   } limit 1000
2. SELECT DISTINCT ?o2 where {
   dbr:Actor_(1993_film) dbo:starring ?o1 .
   ?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

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 .
        ?s1 ?p1 dbr:Actor .
} 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

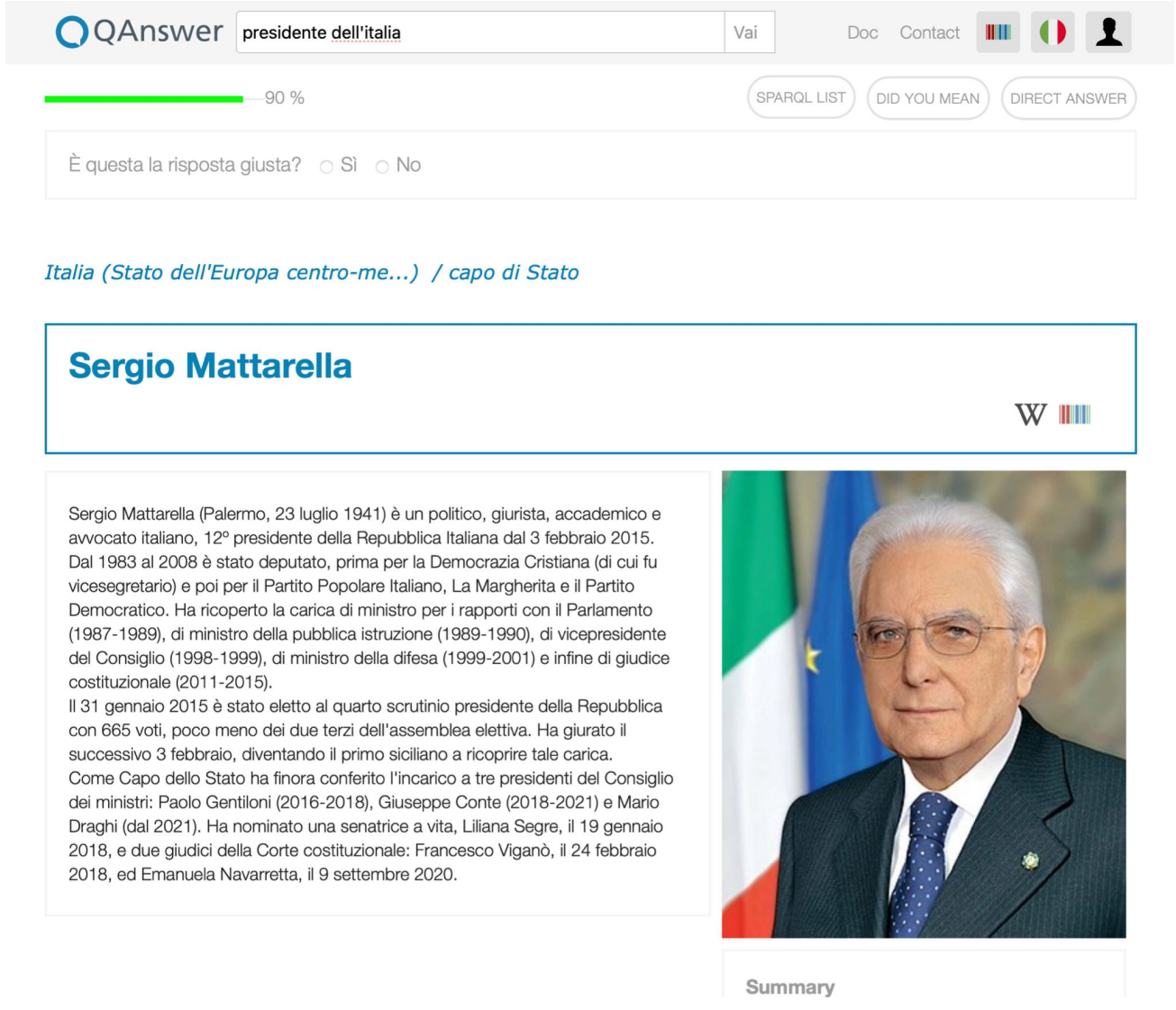
Step 4: Answer Decision

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

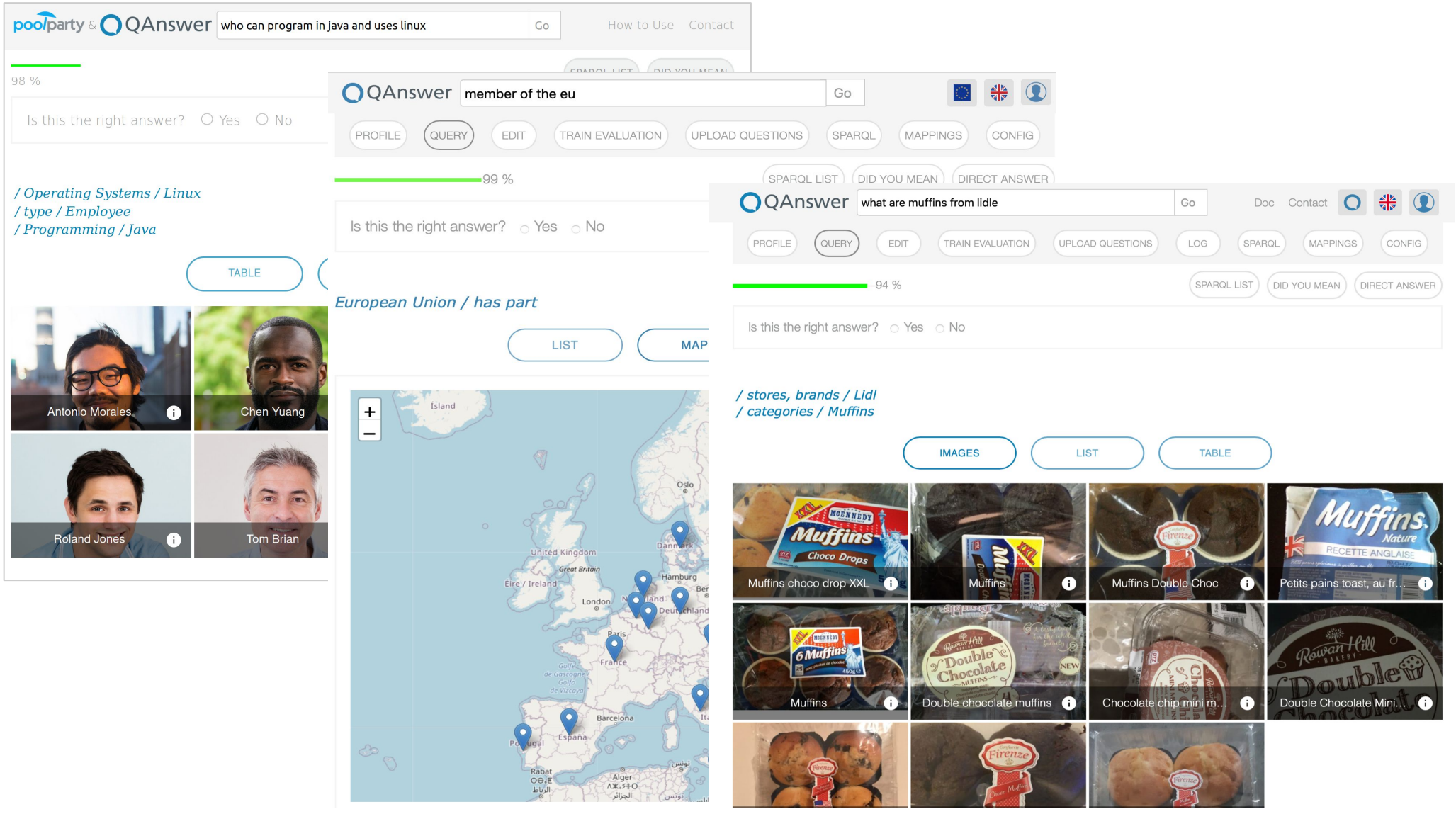


QAnswer advantages: Scalability

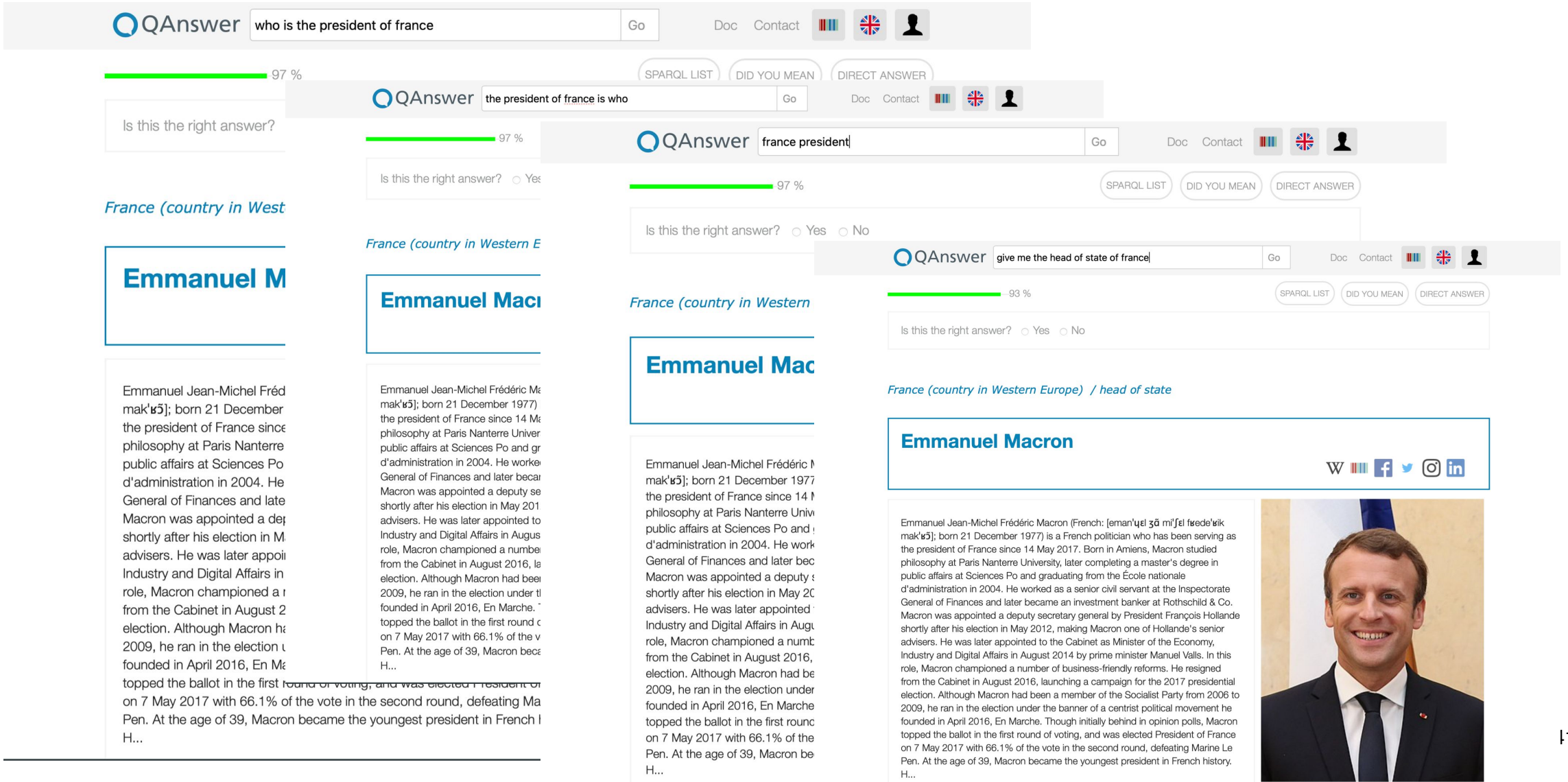
Hardware

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

QAnswer advantages: Portability



QAnswer advantages: Robustness



QAnswer Evaluation: QALD

QA system	Lang	Type	P	R	F	Time
QALD-3						
WDAqua-core1	**en**	**full**	**0.58**	**0.46**	**0.51**	**1.08s**
WDAqua-core1	**en**	**key**	**0.65**	**0.44**	**0.52**	**0.81s**
WDAqua-core1	**de**	**full**	**0.88**	**0.29**	**0.44**	**0.22s**
WDAqua-core1	**de**	**key**	**0.92**	**0.29**	**0.44**	**0.18s**
WDAqua-core1	**fr**	**full**	**0.86**	**0.28**	**0.43**	**0.31s**
WDAqua-core1	**fr**	**key**	**0.92**	**0.29**	**0.44**	**0.20s**
WDAqua-core1	**it**	**full**	**0.89**	**0.29**	**0.44**	**0.20s**
WDAqua-core1	**it**	**key**	**0.92**	**0.29**	**0.44**	**0.16s**
WDAqua-core1	**es**	**full**	**0.90**	**0.29**	**0.44**	**0.18s**
WDAqua-core1	**es**	**key**	**0.91**	**0.29**	**0.44**	**0.17s**
gAnswer [28]*	en	full	0.40	0.40	0.40	≈ 1 s
RTV [14]	en	full	0.32	0.34	0.33	-
Intui2 [11]	en	full	0.32	0.32	0.32	-
SINA [21]*	en	full	0.32	0.32	0.32	≈10-20s
DEANNA [25]*	en	full	0.21	0.21	0.21	≈1-50s
SWIP [19]	en	full	0.16	0.17	0.17	-
Zhu et al. [27]*	en	full	0.38	0.42	0.38	-

QAnswer Evaluation: QALD

QA system	Lang	Type	P	R	F	Time
QALD-4						
Xser [24]	en	full	0.72	0.71	0.72	-
WDAqua-core1	**en**	**key**	**0.76**	**0.40**	**0.52**	**0.32s**
WDAqua-core1	**en**	**full**	**0.56**	**0.30**	**0.39**	**0.46s**
gAnswer [28]	en	full	0.37	0.37	0.37	0.973 s
CASIA [16]	en	full	0.32	0.40	0.36	-
WDAqua-core1	**de**	**key**	**0.92**	**0.20**	**0.33**	**0.04s**
WDAqua-core1	**fr**	**key**	**0.92**	**0.20**	**0.33**	**0.06s**
WDAqua-core1	**it**	**key**	**0.92**	**0.20**	**0.33**	**0.04s**
WDAqua-core1	**es**	**key**	**0.92**	**0.20**	**0.33**	**0.05s**
WDAqua-core1	**de**	**full**	**0.90**	**0.20**	**0.32**	**0.06s**
WDAqua-core1	**it**	**full**	**0.92**	**0.20**	**0.32**	**0.16s**
WDAqua-core1	**es**	**full**	**0.90**	**0.20**	**0.32**	**0.06s**
WDAqua-core1	**fr**	**full**	**0.86**	**0.18**	**0.29**	**0.09s**
Intui3 [12]	en	full	0.23	0.25	0.24	-
ISOFT [18]	en	full	0.21	0.26	0.23	-
Hakimov [15]*	en	full	0.52	0.13	0.21	-

QAnswer Evaluation: QALD

QA system	Lang	Type	P	R	F	Time
QALD-5						
Xser [24]	en	full	0.74	0.72	0.73	-
WDAqua-core1	**en**	**full**	**0.56**	**0.41**	**0.47**	**0.62s**
WDAqua-core1	**en**	**key**	**0.60**	**0.27**	**0.37**	**0.50s**
AskNow[13]	en	full	0.32	0.34	0.33	-
QAnswer[20]	en	full	0.34	0.26	0.29	-
WDAqua-core1	**de**	**full**	**0.92**	**0.16**	**0.28**	**0.20s**
WDAqua-core1	**de**	**key**	**0.90**	**0.16**	**0.28**	**0.19s**
WDAqua-core1	**fr**	**full**	**0.90**	**0.16**	**0.28**	**0.19s**
WDAqua-core1	**fr**	**key**	**0.90**	**0.16**	**0.28**	**0.18s**
WDAqua-core1	**it**	**full**	**0.88**	**0.18**	**0.30**	**0.20s**
WDAqua-core1	**it**	**key**	**0.90**	**0.16**	**0.28**	**0.18s**
WDAqua-core1	**es**	**full**	**0.88**	**0.14**	**0.25**	**0.20s**
WDAqua-core1	**es**	**key**	**0.90**	**0.14**	**0.25**	**0.20s**
SemGraphQA[2]	en	full	0.19	0.20	0.20	-
YodaQA[1]	en	full	0.18	0.17	0.18	-
QuerioDali[17]	en	full		0.48	?	?

QAnswer Evaluation: QALD

QA system	Lang	Type	P	R	F	Time
QALD-6						
UTQA [23]	en	full	0.82	0.69	0.75	-
UTQA [23]	es	full	0.76	0.62	0.68	-
UTQA [23]	fs	full	0.70	0.61	0.65	-
WDAqua-core1	**en**	**full**	**0.62**	**0.40**	**0.49**	**0.93s**
WDAqua-core1	**en**	**key**	**0.52**	**0.32**	**0.40**	**0.68s**
SemGraphQA [2]	en	full	0.70	0.25	0.37	-
WDAqua-core1	**de**	**full**	**0.95**	**0.17**	**0.29**	**0.12s**
WDAqua-core1	**de**	**key**	**0.96**	**0.17**	**0.29**	**0.08s**
WDAqua-core1	**fr**	**full**	**0.91**	**0.16**	**0.27**	**0.37s**
WDAqua-core1	**fr**	**key**	**0.96**	**0.17**	**0.29**	**0.12s**
WDAqua-core1	**it**	**full**	**0.96**	**0.17**	**0.29**	**0.16s**
WDAqua-core1	**it**	**key**	**0.96**	**0.17**	**0.29**	**0.07s**
WDAqua-core1	**es**	**full**	**0.96**	**0.17**	**0.29**	**0.19s**
WDAqua-core1	**es**	**key**	**0.96**	**0.17**	**0.29**	**0.19s**

QAnswer Evaluation: QALD

QA System	Lang	Type	Total	P	R	F	Runtime	Ref
QALD-7 task 4, training dataset								
WDAqua-core1	**en**	**full**	**100**	**0.37**	**0.39**	**0.37**	**1.68s**	-
WDAqua-core1	**en**	**key**	**100**	**0.35**	**0.38**	**0.35**	**0.80s**	-
WDAqua-core1	**es**	**key**	**100**	**0.31**	**0.32**	**0.31**	**0.45s**	-
Sorokin et al. [39]	en	full	100	-	-	0.29	-	[39]
WDAqua-core1	**de**	**key**	**100**	**0.27**	**0.28**	**0.27**	**1.13s**	-
WDAqua-core1	**fr**	**key**	**100**	**0.27**	**0.30**	**0.27**	**1.14s**	-
WDAqua-core1	**fr**	**full**	**100**	**0.27**	**0.31**	**0.27**	**1.05s**	-
WDAqua-core1	**es**	**full**	**100**	**0.24**	**0.26**	**0.24**	**0.65s**	-
WDAqua-core1	**de**	**full**	**100**	**0.18**	**0.20**	**0.18**	**0.82s**	-
WDAqua-core1	**it**	**full**	**100**	**0.19**	**0.20**	**0.18**	**1.00s**	-
WDAqua-core1	**it**	**key**	**100**	**0.17**	**0.18**	**0.16**	**0.44s**	-

QAnswer Evaluation: Simple Questions

QA System	Lang	Type	Total	Accuracy	Runtime	Ref
Lukovnikov et al.	en	full	21687	0.712	-	[29]
Golub and He	en	full	21687	0.709	-	[21]
Yin et al.	en	full	21687	0.683	-	[48]
Bordes et al.	en	full	21687	0.627	-	[4]
Dai et al.*	en	full	21687	0.626	-	[7]
WDAqua-core1*	**en**	**full**	**21687**	**0.571**	**2.1 s**	**-**

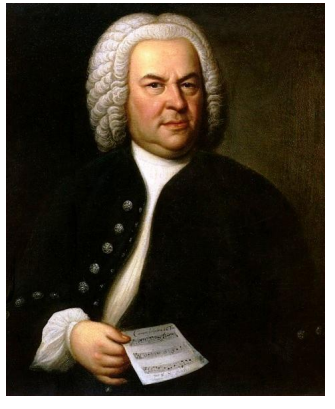
QAnswer Evaluation: LcQuad

Benchmark	Lang	Type	Total	P	R	F	Runtime
LC-QuAD	en	full	5000	0.59	0.38	0.46	1.5 s
WDAquaCore0Questions	mixed	mixed	689	0.79	0.46	0.59	1.3 s

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

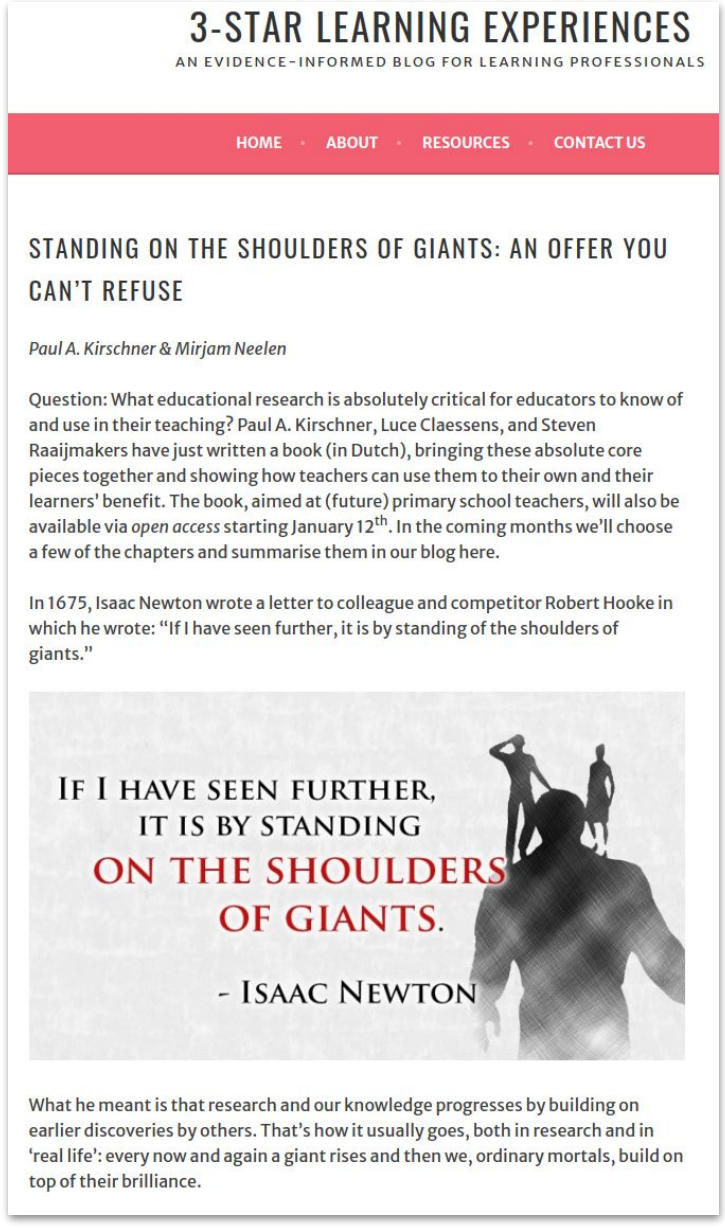
- Example: adding additional functionality
  - User's question: *"Where and when was Johann Sebastian Bach born?"*
    - Possible solution: transformation into two interpretations
      - When was Johann Sebastian Bach born? → 31 March 1685 (Gregorian Calendar)
      - Where was Johann Sebastian Bach born? → Eisenach
  - Follow-up: *Give me the corresponding state.* → Saxe-Eisenach (https://www.wikidata.org/wiki/Q696651)



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

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



Both, Andreas, Dennis Diefenbach, Kuldeep Singh, Saeedeh Shekarpour, Didier Cherix, and Christoph Lange. ***Qanary - a methodology for vocabulary-driven open question answering systems.*** In 13th Extended Semantic Web Conference, 2016.

Motivation Qanary Approach

https://github.com/WDAqua/Qanary

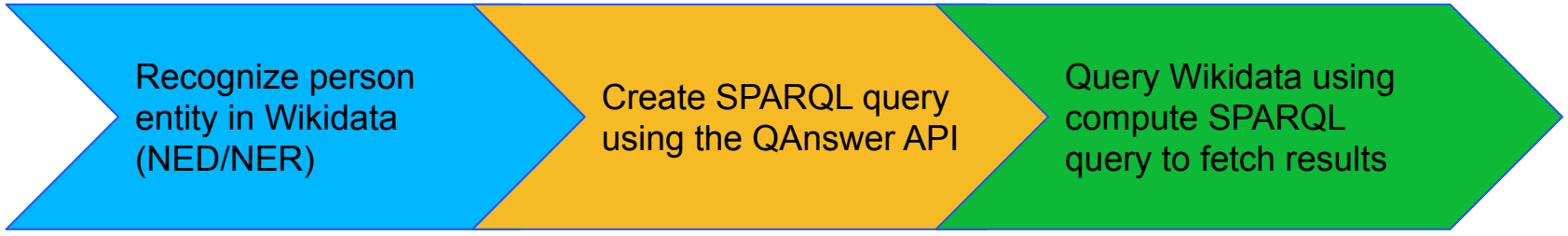
- Goal of the Qanary methodology: Establish a component-oriented framework
  - Each sub-task of a QA system is considered to be a QA component
- Features:
  - Flexible w.r.t. the size and purpose of the QA components
  - Plug & play behavior of the QA components
  - Knowledge-driven approach (RDF-based information architecture)
- Your advantages:
  - Open source implementation
  - Add additional functionality
  - Reuse previously implemented components
    - https://github.com/WDAqua/Qanary-question-answering-components
    - Remark: QAnswer is an example of a Query Builder component
      - https://github.com/WDAqua/Qanary-question-answering-components/tree/master/qanary_component-QBE-QAnswer

  → Make building Question Answering systems easier

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

Example 1 (Wikidata KG)

- QA system to answer questions like: *When and where was <Person> born ?*
- Implement 3 components:

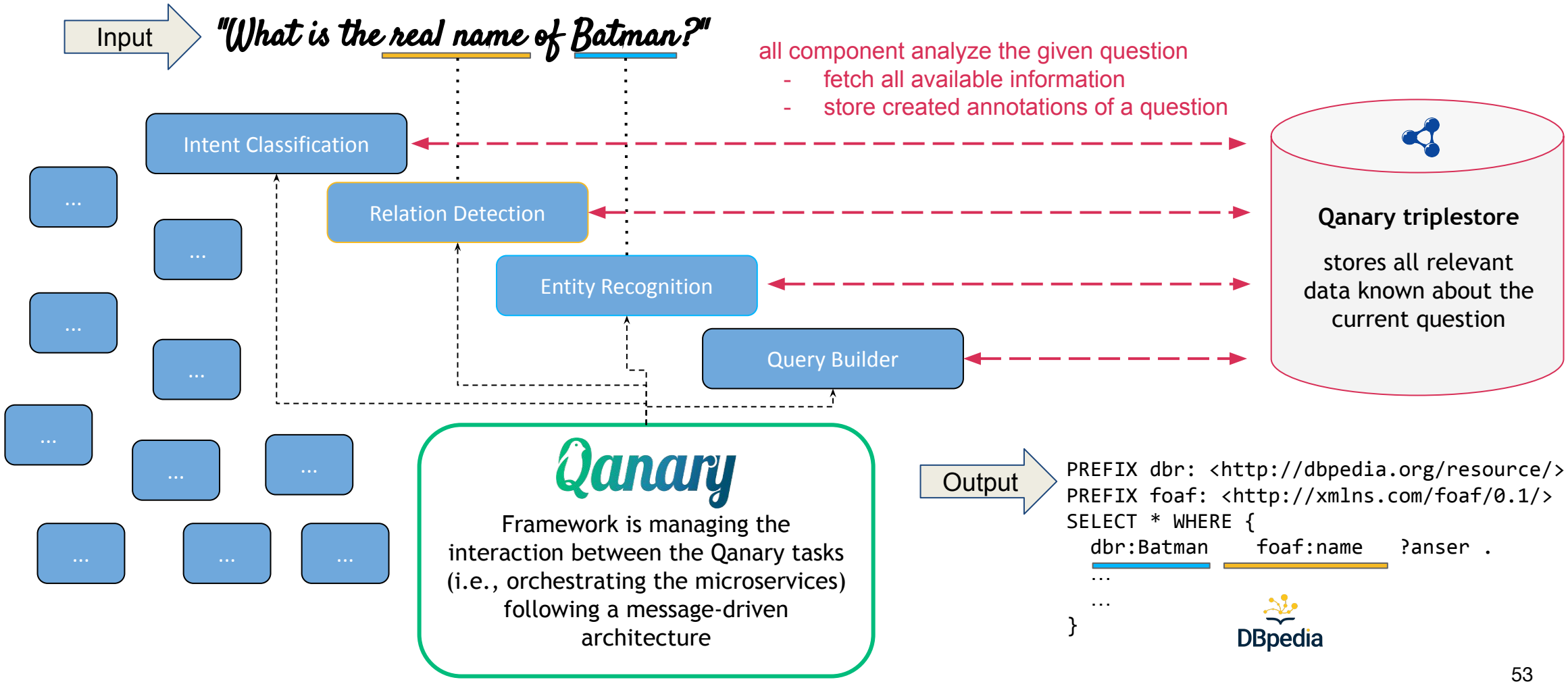


The mentioned components are available at: https://github.com/WDAqua/Qanary-question-answering-components

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

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



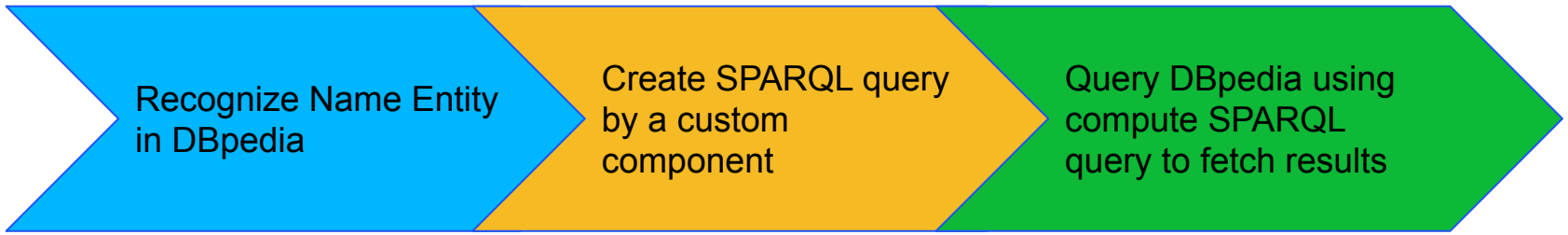
Actually computed question for *"What is the real name of Batman?"*

```
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
}
```

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

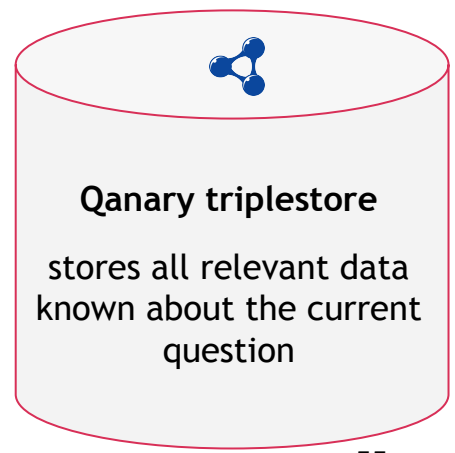
Example 2 (DBpedia KG)

- QA system to answer questions like: *What is the real name of <superhero character> ?*
- Implement 3 components:



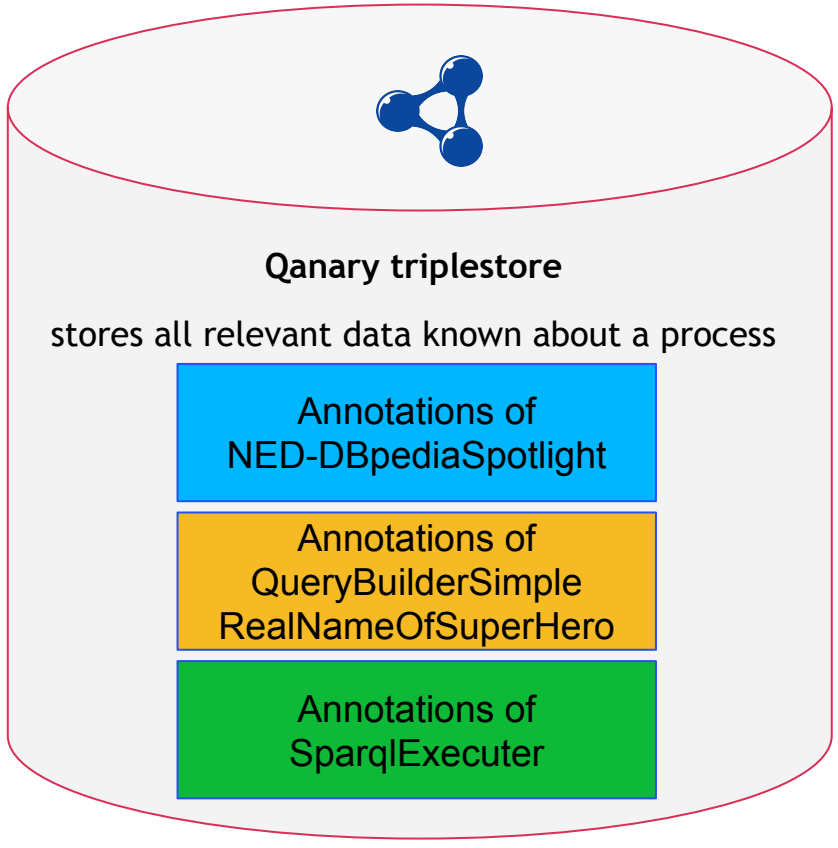
The mentioned components are available at: https://github.com/WDAqua/Qanary-question-answering-components

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



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

Kuldeep Singh, Arun Sethupat Radhakrishna, Andreas Both, Saeedeh Shekarpour, Ioanna Lytra, Ricardo Usbeck, Akhilesh Vyas, Akmal Khikmatullaev, Dharmen Punjani, Christoph Lange, Maria-Esther Vidal, Jens Lehmann, and Sören Auer. ***Why reinvent the wheel: Let’s build question answering systems together. The World Wide Web Conference***, WWW 2018.

Summary

https://github.com/WDAqua/Qanary

- 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

After the lunch break 🍴

QAnswer: Examples & Hands on

by Dennis Diefenbach and Pierre Maret

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

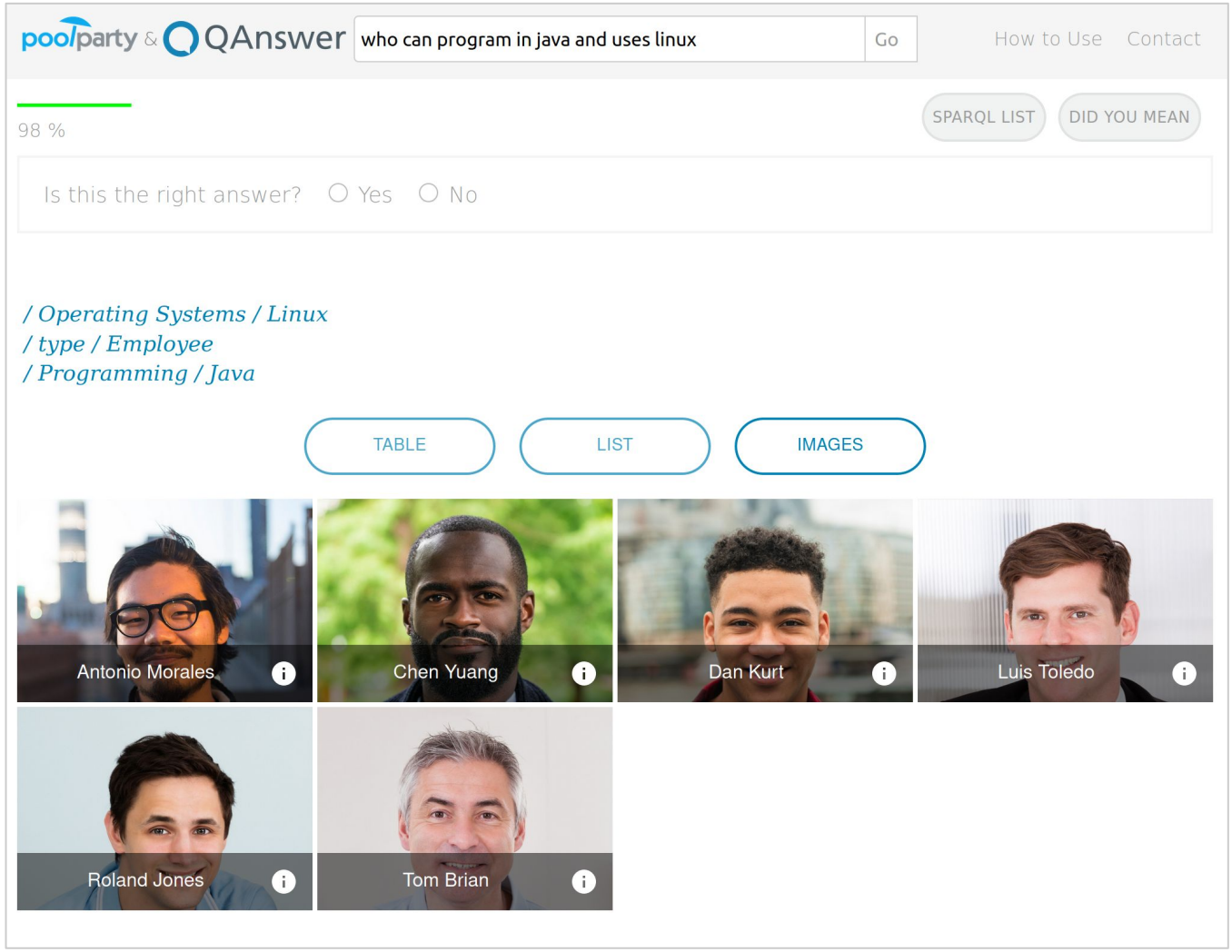
Paul Heinze
Anhalt University of Applied Sciences
paul.heinze@student.hs-anhalt.de

QA Tutorial @ESWC 2021

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

Example 1: HR

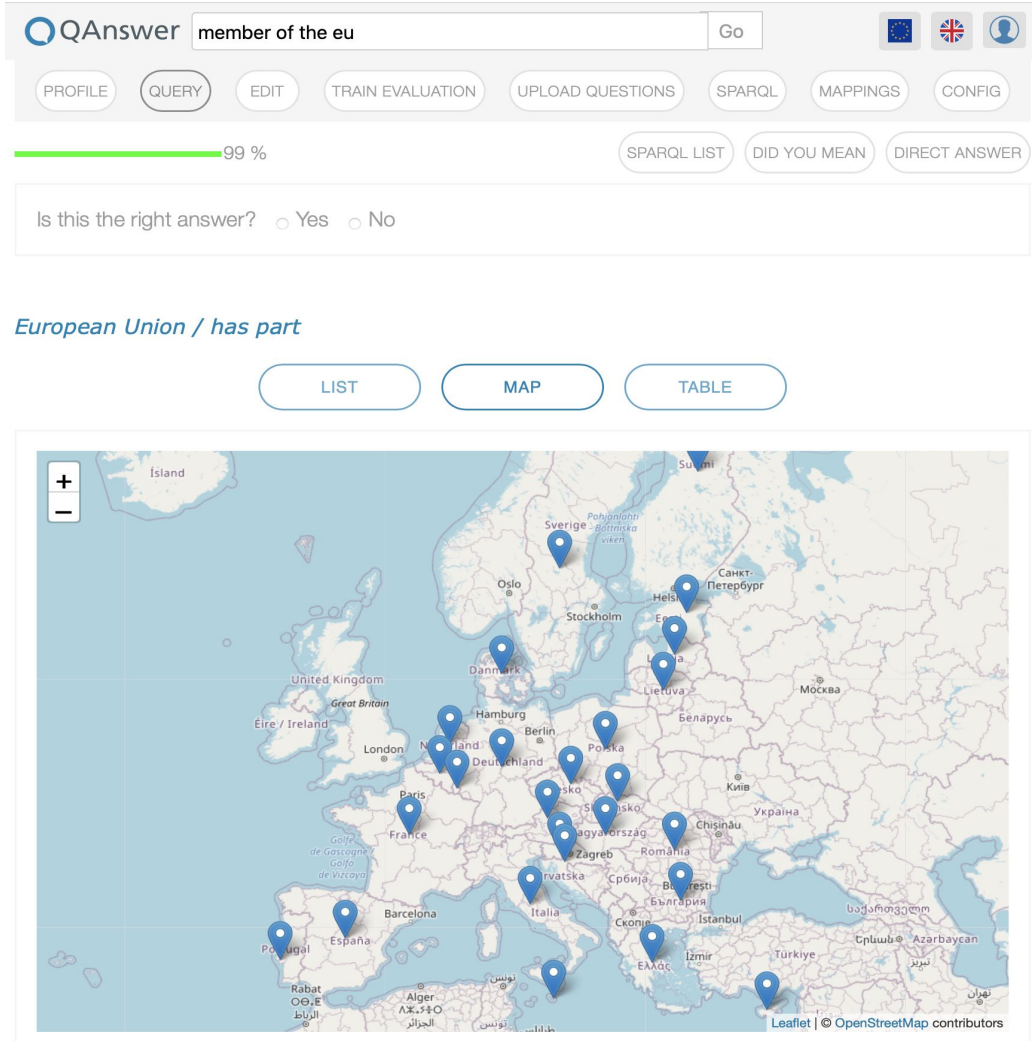
Querying a Knowledge Graph containing information about Employees.



Example 2: EU Knowledge Graph

Querying a Knowledge Graph of the European Commission that contains data about:

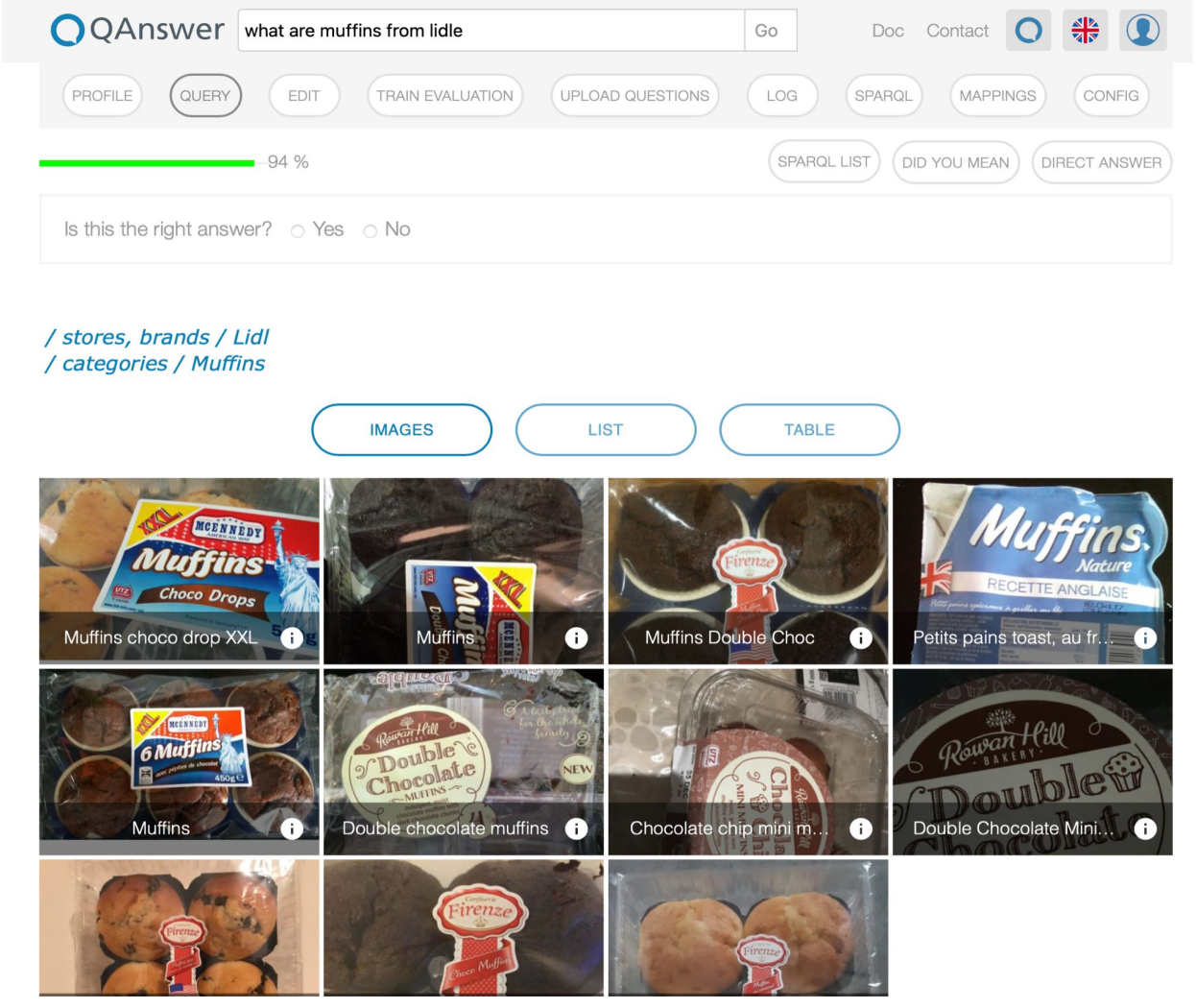
- countries / capitals
- head of states
- european institutions
- buildings by the commission
- more than 700.000 projects financed by the European Commission



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



Hands-on session

Create our own Digital Twin using QAnswer!

After the coffee break ☕

The Question Answering Framework

Qanary

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

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

QA Tutorial @ESWC 2021

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

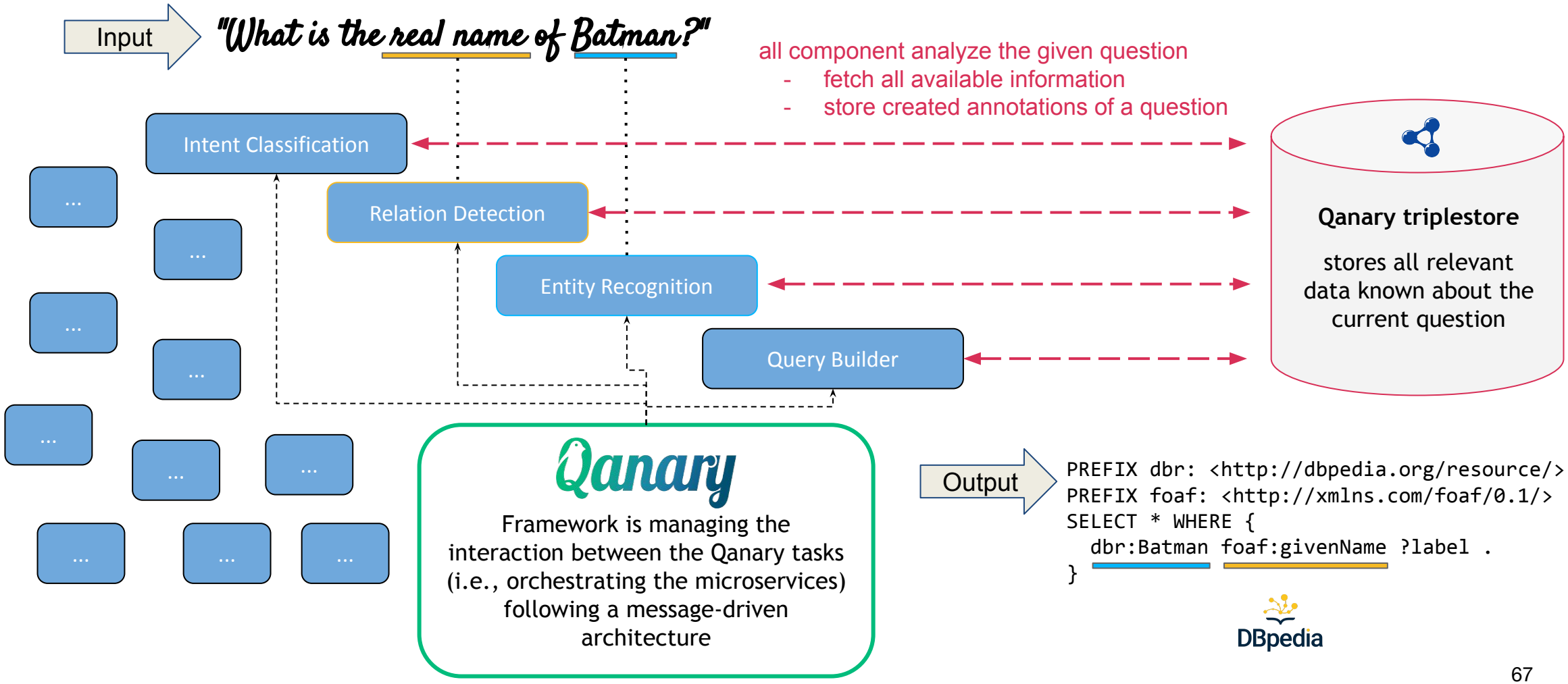
All information will be available at https://qanswer.github.io/QA-ESWC2021/

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
  - 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

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



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

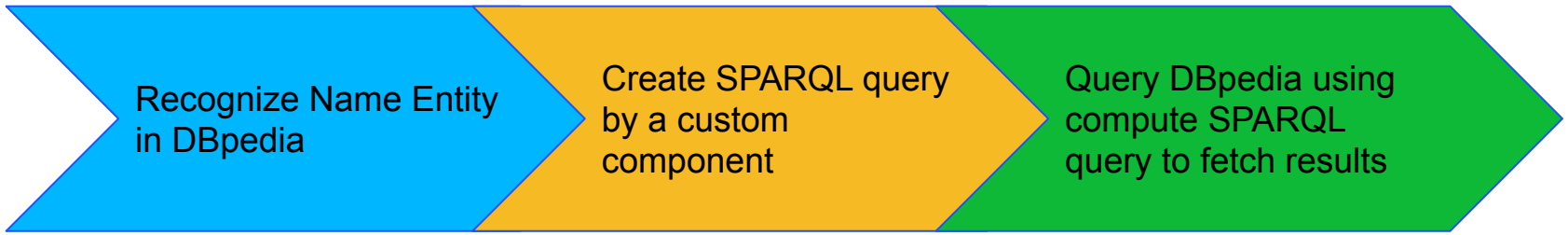
- rapid and efficient system development
- supports agile software development
- easy to measure systems quality
- native support for AI-driven development
- flexible usage:
  - can be used to develop new sub-systems and
  - to develop complete Question Answering systems

Research Vision

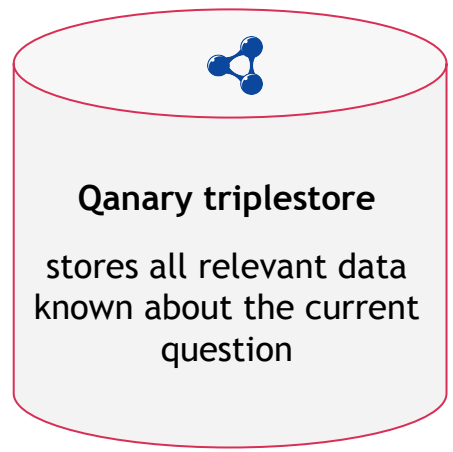
- integrate all best-of-breed approaches
- establish domain-agnostic automatic optimization of Question Answering quality

Example (DBpedia KG)

- QA system to answer questions like: *What is the real name of <superhero character> ?*
- Implement 3 components:



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



Full process: *What is the real name of <superhero character> ?*

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

- **Qanary framework** receives question
  a. question is stored in the Qanary triplestore
  b. Calls 1st Qanary component: **NED-DBpediaSpotlight**

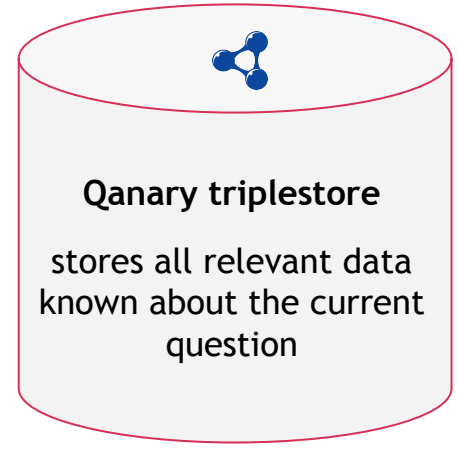
code@GitHub
  1. Fetch the question from the Qanary triplestore
  2. Call the DBpedia Spotlight service for the given question
  3. Store Named Entity annotations into the Qanary triplestore

  c. Calls 2nd Qanary component: **QueryBuilderSimpleRealNameOfSuperHero**

code@GitHub
  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**

code@GitHub
  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



Generalized Process

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

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

1. Fetch required data from the Qanary triplestore using SPARQL:

```
SELECT * FROM <inGraph> WHERE { ... }
```

2. Compute new information about the current question

3. Store new information into the Qanary triplestore using SPARQL:

```
INSERT { GRAPH <outGraph> {
    ...
}}
```

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

3 parameters are provided to all components:

1. `endpoint`
   - URI of the Qanary triplestore
2. `URI inGraph`
   - URI of the graph with information from the previous components
3. `URI outGraph`
   - URI of the graph where the newly computed data has to be stored

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
   - SparqlExecuter
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 .
}
```

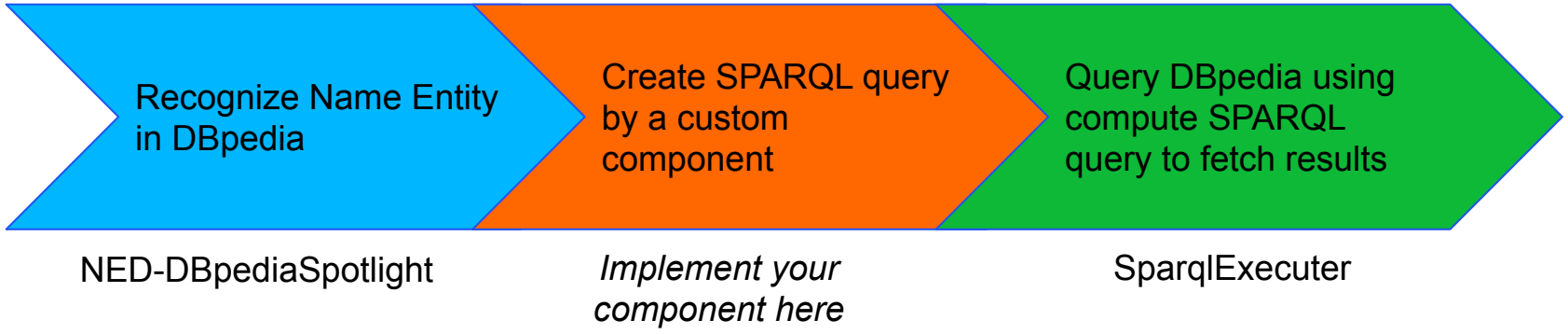
```
# find all annotations computed in your process
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX oa: <http://www.w3.org/ns/openannotation/core/>
PREFIX qa: <http://www.wdaqua.eu/qa#>
SELECT *
FROM <ADD HERE THE URI SHOWN AS Qanary question analysis outgraph>
WHERE {
    ?annotationId 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

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

- 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 WikidataQueryExecuter 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

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

- **Process:**
  - prepare the workspace:
    - you have three options (see following slides)
  - implement your Qanary component (prefered: 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: https://github.com/WDAqua/Qanary-question-answering-components/blob/master/qanary_component-QB-SimpleRealNameOfSuperHero/src/main/java/eu/wdaqua/qanary/component/querybuilder/QueryBuilderSimpleRealNameOfSuperHero.java#L44
  - Python: https://github.com/WDAqua/Qanary-question-answering-components/blob/master/qanary_component-Python-QC-EAT-classifier/app/answer_type_classifier.py
- **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

Let’s build a QA system using the Qanary ecosystem

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

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:
    https://github.com/WDAqua/Qanary/wiki/How-do-I-implement-a-new-Qanary-component-using-Java%3F
  - Python using a prepared library:
    https://github.com/WDAqua/Qanary/wiki/How-to-Implement-a-Qanary-Component-using-Python-Qanary-Helpers
  - Python using plain Flask:
    https://github.com/WDAqua/Qanary/wiki/How-to-Implement-a-Qanary-Component-using-Python

Required components to run your own Qanary system

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

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

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

- A **Stardog triplestore** is already available at http://webengineering.ins.hs-anhalt.de:40159/
  - see http://webengineering.ins.hs-anhalt.de:40159/qanary for predefined vocabularies
  - credentials: admin / admin
  - optional: SPARQL editor for Stardog: 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
  - currently available components: 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/
- A **Qanary developer's view** is available at 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

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

- 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
    - `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 (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
    - `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 you 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

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

- 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/
- the whole process is described at 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
    - `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
- 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

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

- 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

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)
- 2014 – Usbeck, R., Ngomo, A. C. N., Röder, M., Gerber, D., Coelho, S. A., Auer, S., & Both, A. AGDISTIS-graph-based disambiguation of named entities using linked data. In International semantic web conference. ISWC 2014. Springer, Cham. Received Best Research Paper Award
- 2016 – Both, Andreas, Dennis Diefenbach, Kuldeep Singh, Saeedeh Shekarpour, Didier Cherix, and Christoph Lange. Qanary - a methodology for vocabulary-driven open question answering systems. In 13th Extended Semantic Web Conference, 2016.
- 2018 – Kuldeep Singh, Arun Sethupat Radhakrishna, Andreas Both, Saeedeh Shekarpour, Ioanna Lytra, Ricardo Usbeck, Akhilesh Vyas, Akmal Khikmatullaev, Dharmen Punjani, Christoph Lange, Maria-Esther Vidal, Jens Lehmann, and Sören Auer. Why reinvent the wheel: Let's build question answering systems together. The World Wide Web Conference, WWW 2018.
- 2019 – Diefenbach, D., Migliatti, P. H., Qawasmeh, O., Lully, V., Singh, K., & Maret, P. QAnswer: a question answering prototype bridging the gap between a considerable part of the LOD cloud and end-users. In The World Wide Web Conference, WWW 2019.
- 2020 – Diefenbach, D., Both, A., Singh, K., & Maret, P. Towards a question answering system over the semantic web. Semantic Web Journal, iOS Press.
- 2020 – Perevalov, A, Both, A. (2020) Augmentation-based Answer Type Classification of the SMART dataset, SeMantic AnsweR Type prediction task (SMART) at ISWC 2020 Semantic Web Challenge.

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/

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

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

QA Tutorial @ESWC 2021

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