# Question Answering and Chatbots

## 5th Practical exercise – Qanary Framework

Aleksandr Perevalov

aleksandr.perevalov@hs-anhalt.de

November 10, 2021

Hochschule Anhalt
Anhalt University of Applied Sciences

# Plan for today

# Plan for today

- Demo Session;

# Plan for today

- Demo Session;
- Task for the exercise 5;

# Plan for today

- Demo Session;
- Task for the exercise 5;
- Hands-on with Qanary Framework.

Let's start the demo.

# Frameworks for Quesiton Answering & Chatbots – Low Code

# Frameworks for Quesiton Answering & Chatbots – Low Code

Dialogflow

# Frameworks for Quesiton Answering & Chatbots – Low Code

Dialogflow

Microsoft Bot Framework

# Frameworks for Quesiton Answering & Chatbots – Traditional

# Frameworks for Quesiton Answering & Chatbots – Traditional

# Frameworks for Quesiton Answering & Chatbots – Traditional

# Frameworks for Quesiton Answering & Chatbots – Traditional

# Qanary Framework

# Qanary Framework

- the framework helps you to facilitate KGQA system with multiple independent **components**;

# Qanary Framework

- the framework helps you to facilitate KGQA system with multiple independent **components**;
- each **component** is implemented as a micro-service and agnostic to the used programming language;

# Qanary Framework

- the framework helps you to facilitate KGQA system with multiple independent **components**;
- each **component** is implemented as a micro-service and agnostic to the used programming language;
- the **components** interact with each other by writing and reading standardized RDF annotations from a common triplestore;

# Qanary Framework

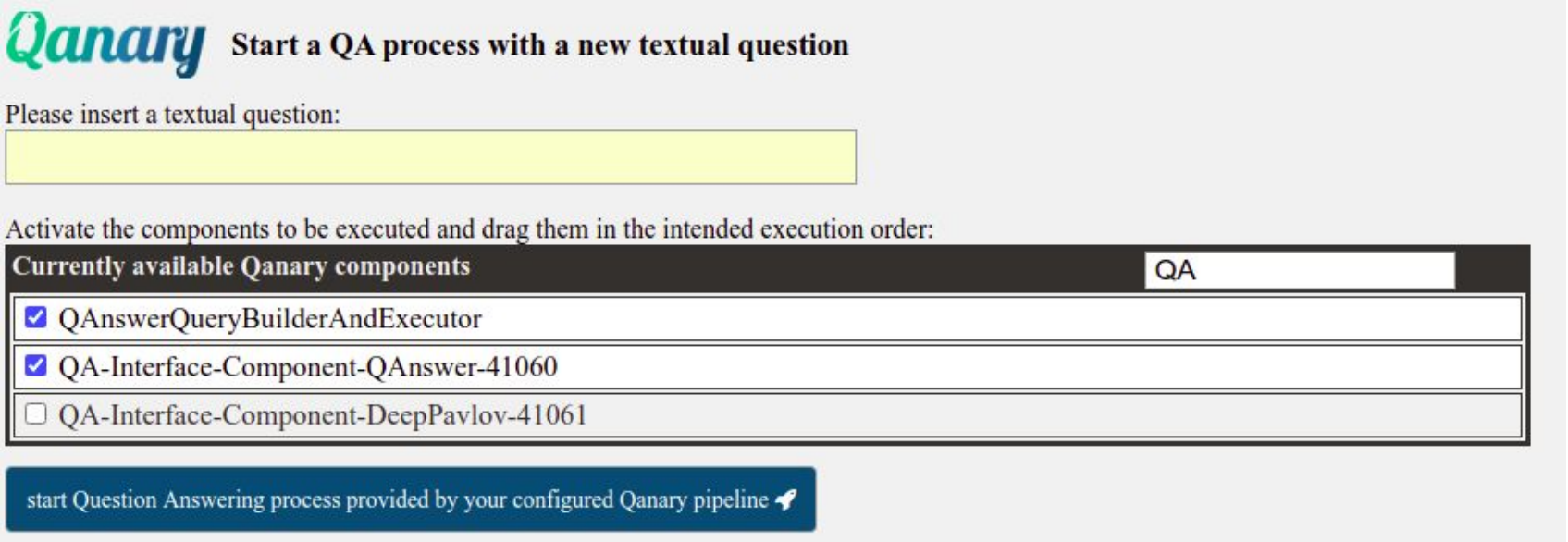
- the framework helps you to facilitate KGQA system with multiple independent **components**;
- each **component** is implemented as a micro-service and agnostic to the used programming language;
- the **components** interact with each other by writing and reading standardized RDF annotations from a common triplestore;
- a user of the framework may reuse **components**, developed and deployed by other users without having access the source code;

# Qanary Framework

- the framework helps you to facilitate KGQA system with multiple independent **components**;
- each **component** is implemented as a micro-service and agnostic to the used programming language;
- the **components** interact with each other by writing and reading standardized RDF annotations from a common triplestore;
- a user of the framework may reuse **components**, developed and deployed by other users without having access the source code;
- the question answering process becomes transparent, interpretable, and explainable because all the data generated by **components** is stored in a triplestore (i.e., the process can be traced).

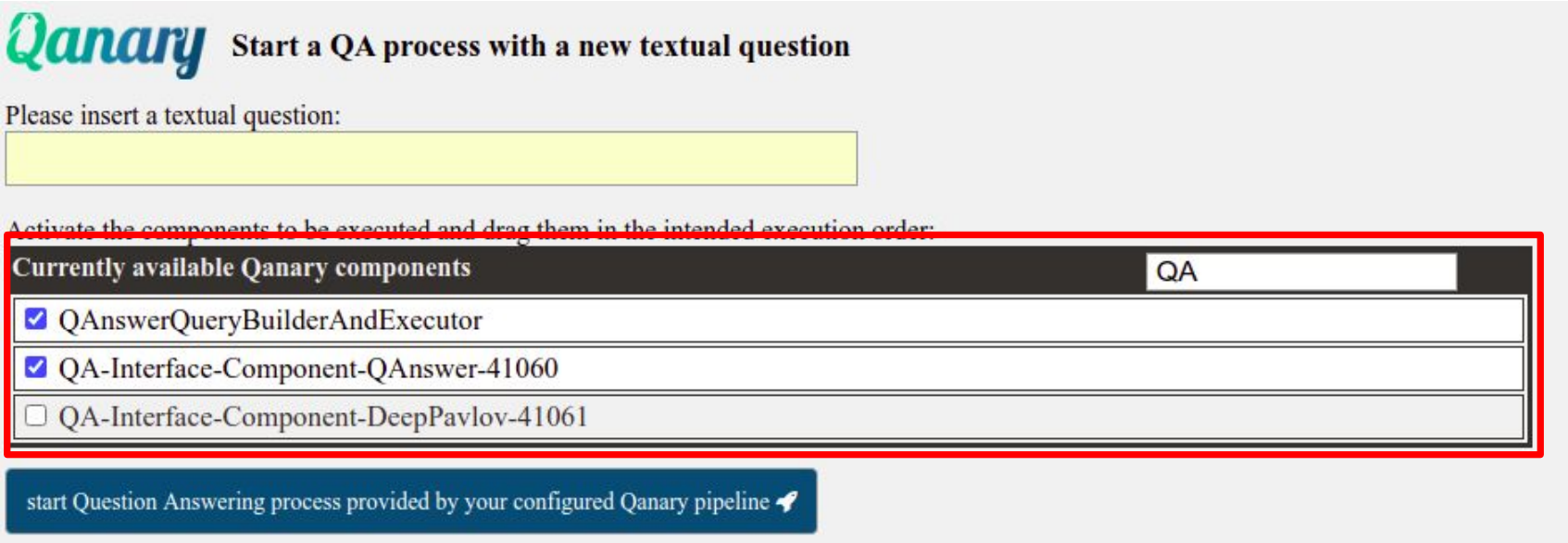
# Qanary Pipeline

Serves a similar to API gateway function which is to execute components with certain parameters in a certain order.



# Qanary Components

Set of components that were developed and deployed by the Qanary community (note: be aware that this is just an open collection, there might be many more components out there).



# Qanary Annotations

```
INSERT {
GRAPH <uuid> {
    ?newAnnotation rdf:type qa:AnnotationOfAnswerSPARQL .
    ?newAnnotation oa:hasBody "sparqlQuery"^^xsd:string .
    ?newAnnotation oa:annotatedAt ?time .
    ?newAnnotation oa:annotatedBy <urn:qanary:componentName> .
    }
}
WHERE {
    BIND (IRI(str(RAND())) AS ?newAnnotation) .
    BIND (now() as ?time)
}
```

[1] http://www.w3.org/TR/annotation-model

# Qanary Annotations

Each component executes its task, creates new information about the given question, and stores it in the **Qanary triplestore** – global memory of the RDF information computed while analyzing a question...

```
INSERT {
GRAPH <uuid> {
    ?newAnnotation rdf:type qa:AnnotationOfAnswerSPARQL .
    ?newAnnotation oa:hasBody "sparqlQuery"^^xsd:string .
    ?newAnnotation oa:annotatedAt ?time .
    ?newAnnotation oa:annotatedBy <urn:qanary:componentName> .
    }
}
WHERE {
    BIND (IRI(str(RAND())) AS ?newAnnotation) .
    BIND (now() as ?time)
}
```

[1] http://www.w3.org/TR/annotation-model

# Qanary Annotations

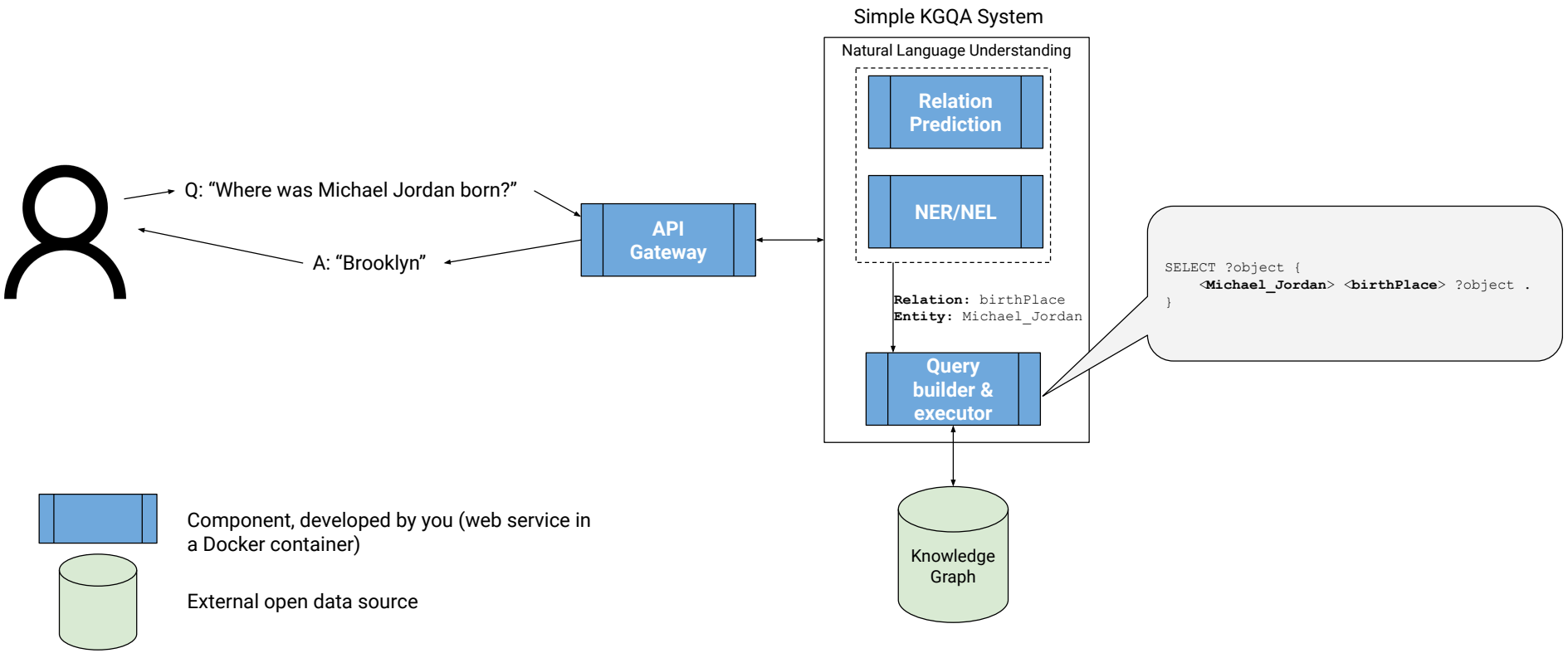
Each component executes its task, creates new information about the given question, and stores it in the **Qanary triplestore** – global memory of the RDF information computed while analyzing a question...

```
INSERT {
GRAPH <uuid> {
    ?newAnnotation rdf:type qa:AnnotationOfAnswerSPARQL .
    ?newAnnotation oa:hasBody "sparqlQuery"^^xsd:string .
    ?newAnnotation oa:annotatedAt ?time .
    ?newAnnotation oa:annotatedBy <urn:qanary:componentName> .
    }
}
WHERE {
    BIND (IRI(str(RAND())) AS ?newAnnotation) .
    BIND (now() as ?time)
}
```

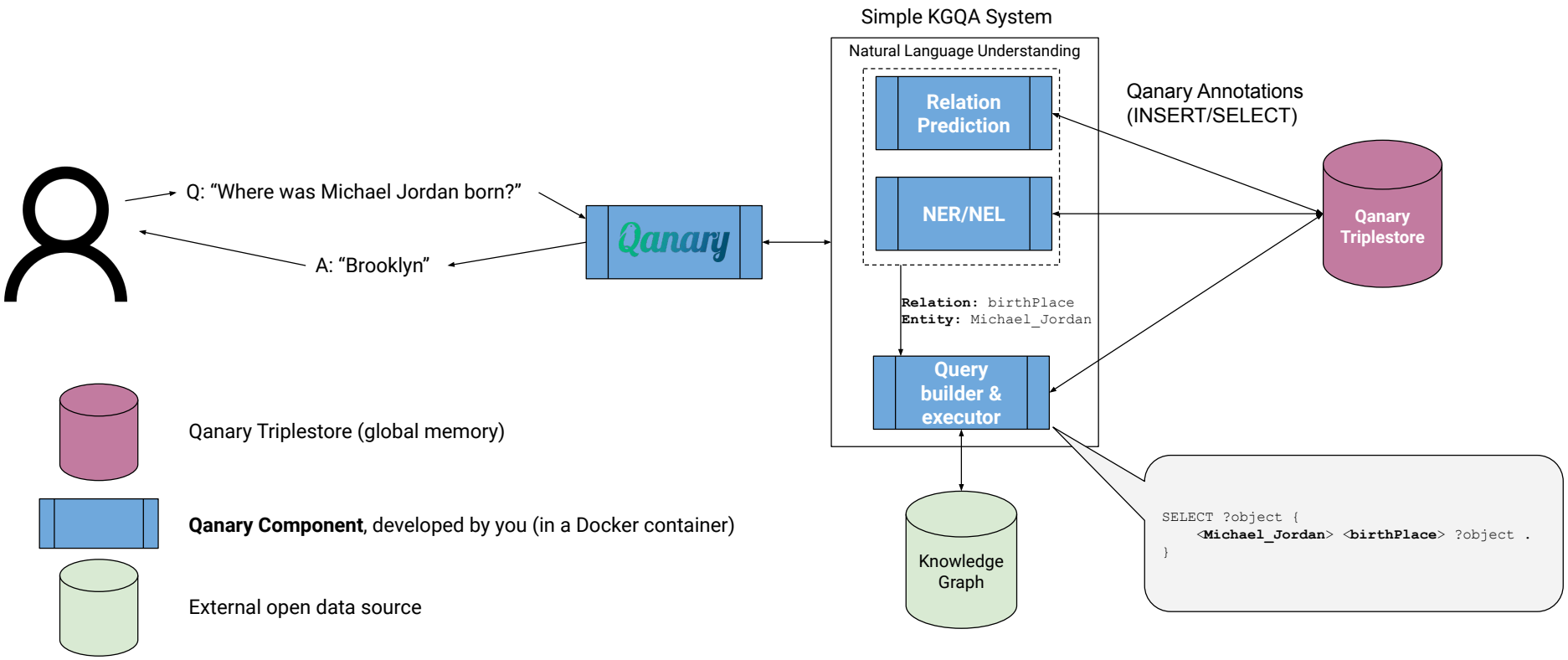
...this RDF data is called *semantic annotations of the given question*[1]

[1] http://www.w3.org/TR/annotation-model

# Your system architecture so far



# System architecture with Qanary Framework



# The task

[2] guide will be provided in the Moodle

# The task

1. Sketch a component-oriented solution + your data flow for annotations (what information your component produces) – use material from previous exercise;

[2] guide will be provided in the Moodle

# The task

1. Sketch a component-oriented solution + your data flow for annotations (what information your component produces) – use material from previous exercise;
2. Transform components to Qanary framework – wrap your functionality according to the example;

[2] guide will be provided in the Moodle

# The task

1. Sketch a component-oriented solution + your data flow for annotations (what information your component produces) – use material from previous exercise;
2. Transform components to Qanary framework – wrap your functionality according to the example;
3. Ask questions with Qanary pipeline – you can debug it by executing SPARQL queries on the triplestore[2];

[2] guide will be provided in the Moodle

# The task

1. Sketch a component-oriented solution + your data flow for annotations (what information your component produces) – use material from previous exercise;
2. Transform components to Qanary framework – wrap your functionality according to the example;
3. Ask questions with Qanary pipeline – you can debug it by executing SPARQL queries on the triplestore[2];
4. Use general chatbot frontend to connect your system with the user interface with no coding.

[2] guide will be provided in the Moodle

Hands-on with Qanary Framework

Let's do the exercise!

# Plan for the Exercise 6: Evaluation of QA systems

# Plan for the Exercise 6: Evaluation of QA systems

- Metrics for unordered output: Precision, Recall, etc.

# Plan for the Exercise 6: Evaluation of QA systems

- Metrics for unordered output: Precision, Recall, etc.
- Metrics for ordered output: Precision@k, Mean Reciprocal Rank, NDCG@k, etc.

# Plan for the Exercise 6: Evaluation of QA systems

- Metrics for unordered output: Precision, Recall, etc.
- Metrics for ordered output: Precision@k, Mean Reciprocal Rank, NDCG@k, etc.
- Qualitative metrics – rate your experience.

# Plan for the Exercise 6: Evaluation of QA systems

- Metrics for unordered output: Precision, Recall, etc.
- Metrics for ordered output: Precision@k, Mean Reciprocal Rank, NDCG@k, etc.
- Qualitative metrics – rate your experience.
- Component-level metrics.

0 Introduction;

1 NER & NEL;

2 Question classification & Web service/API;

3 SPARQL queries over Knowledge Graphs;

4 Simple KGQA system – based on exercises 0, 1, 2, 3;

5 **Qanary Framework – component oriented approach**;

6 ~~Simple ODQA system;~~

6 Evaluation of QA systems.