Rapid Engineering of Question Answering Systems using the lightweight Qanary Approach

Tutorial at JIST 2017

Andreas Both

Head of Architecture, Web Technology and IT Research
at DATEV eG

2017-11-10, 7th Joint International Semantic Technology Conference (JIST 2017)

http://wdaqua.eu, https://github.com/WDAqua/

Tutorial Plan

Introduction and Motivation
- Question Answering
- Question Answering Systems

Qanary Methodology and Technical Framework
- Idea
- Knowledge Representation using the qa Vocabulary
- Qanary Methodology

Technical Part
- Interactive Session: Solution Definition
- Coding Session: Implement your first QA system from existing components
- Validate the quality of your QA system
- Improve and revalidate your QA system
- Solve new QA tasks

Final Remarks

Introduction and Motivation

Something about me

Dr. Andreas Both

– 2005 Studies of Computer Science, University Halle (Germany)

– 2010 PhD in Software Engineering and Programming Languages, University Halle (Germany)

– 2012 Project Lead of "Semantic Web Project" (R & D), Unister GmbH (Germany)

– 2015 Head of Research and Development Department, Unister GmbH (Germany)

2016 Research and Development Lead Mercateo AG (Germany)

11/2016 – Head of Architecture, Web Technology and IT Research, DATEV eG (Germany)

DATEV eG: https://www.datev.com/

- software company and IT service provider
- turnover: $>$ 900 million euros
- age: $>$ 50 years old
- core market: Germany
- fields: accounting, business consulting, taxation, enterprise resource planning (ERP) as well as organization and planning
- members: $>$ 40.000
- customers: $>$ 2.6 million companies

Today's Goals

You will ...

- receive a compact overview about Question Answering (QA) and its challenges
- understand the $Qanary$ methodology, the RDF vocabulary qa and the component-oriented $Qanary$ framework
- learn to iteratively build, validate and improve your own QA system using the $Qanary$ framework

Thereafter, you will ...

- be enabled to implement you own QA system
- take advantage of the Qanary ecosystem for rapid research results
- contribute to the research community to improve the state-of-the-art

Schedule

30 min Introduction

30 min Question Answering (QA) using $Qanary$

20 min RDF-based knowledge design of a QA problem using the qa vocabulary

15 min coffee break

30 min exercise: model QA ontology using the qa vocabulary, write SPARQL queries for answering exemplary questions

40 min exercise: implement your own QA system using $Qanary$

10 min conclusions and outlook

Question Answering

Introduction on Question Answering

Overview

- aim: answer users questions using given data
- importance: enables user to actually work with Big Data
- challenges: ambiguity of language, large data sets,
- technologies: information retrieval (IR), natural language processing (NLP), Linked Data & Semantic Web, artificial intelligence (AI), ...

Attributes of QA

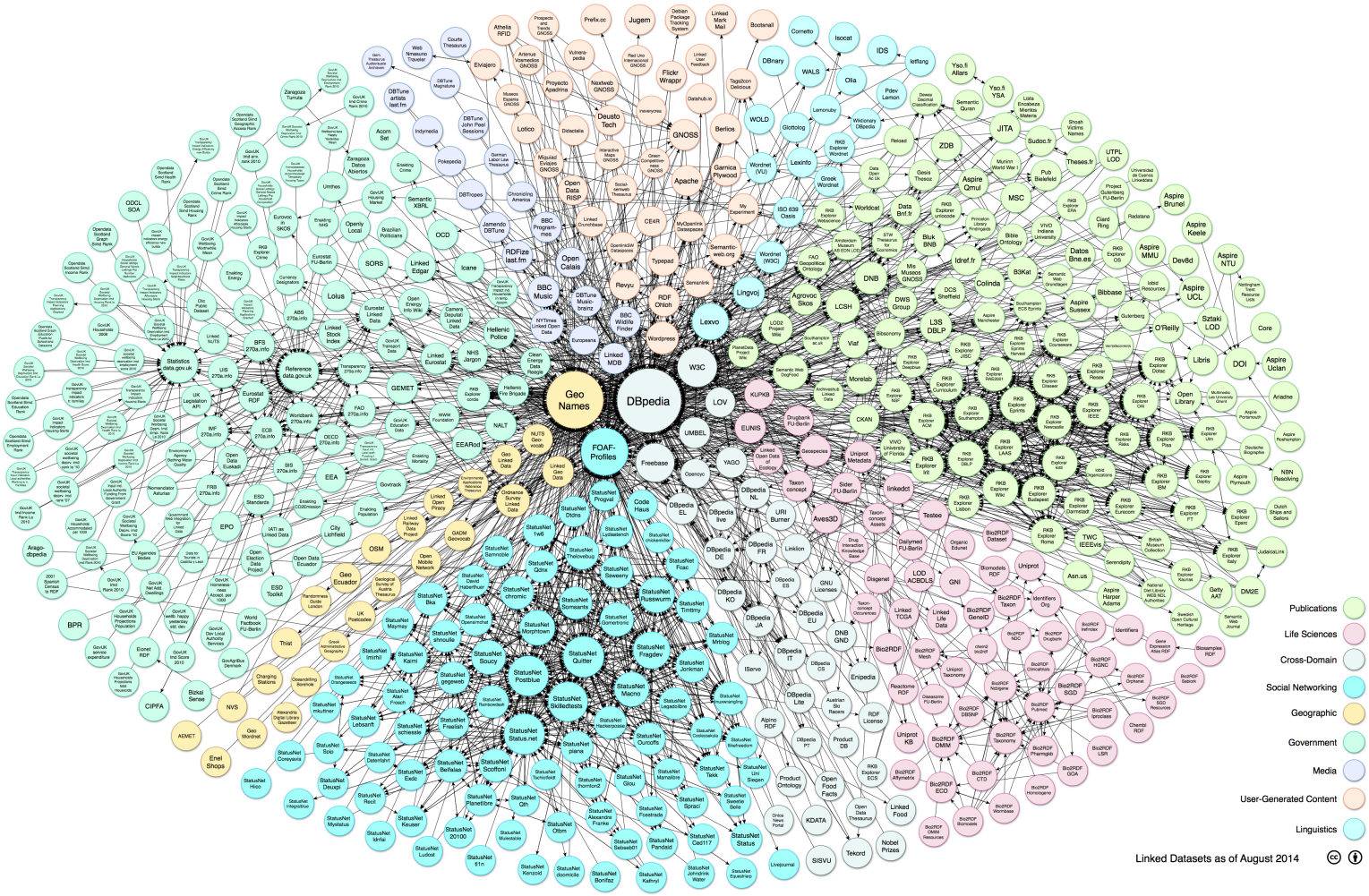
- fact-based
- text-based
- statistical
- multilingual
- community-based
- closed/open domain
- hybrid
- visual
- ...

Introduction on Question Answering

Our Focus

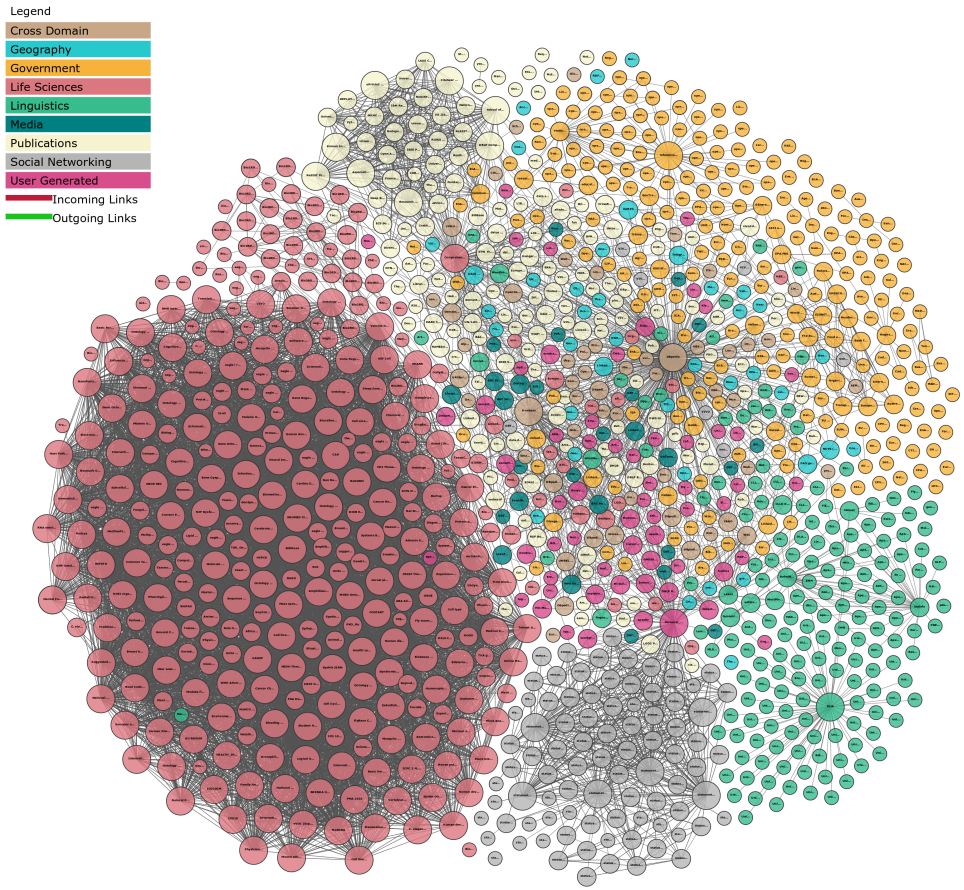
- natural language input
  - general: (multilingual) natural language, factoid questions
  - today: English questions
  - examples:
    - "What is the real name of Batman?"
    - "Is Bruce Wayne the real name of Batman?"
    - "How many partners had Batman?"
  - possible sources to answer the questions: en.wikipedia.org/wiki/Batman, dbpedia.org/resource/Batman, wikidata.org/wiki/Q2695156
- structured data sets as knowledge base
  - DBpedia, Wikidata, Freebase, ...
  - today: DBpedia

Excursus: Linked Open Data Cloud



http://lod-cloud.net/

Excursus: Linked Open Data Cloud



Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. `http://lod-cloud.net/`

Question Answering Benchmarks

Challenge to Measure the Quality of QA systems

- high variety of questions
- training requires data
- comparability requires gold standards

QA Benchmarks

- Question Answering over Linked Data (QALD)
  - hundreds of questions
  - tasks: Multilingual QA over DBpedia, Hybrid Question Answering, English question answering over Wikidata
  - website: `http://www.sc.cit-ec.uni-bielefeld.de/qald`
  - e.g., QALD-8 challenge at ISWC 2017
- Largescale Complex Question Answering Dataset (LC-QuAD)
  - thousands of English questions
  - `https://iswc2017.semanticweb.org/paper-152/`
  - website: `http://lc-quad.sda.tech/`

Question Answering Systems

Introduction on Question Answering Systems

- BASEBALL[1]
  - very early QA system (1963)
  - using baseball database
  - answers questions w.r.t. dates, locations, ...
- START Natural Language Question Answering System[2]
  - open-domain QA system
  - uses particular knowledge bases
  - demo: `http://start.csail.mit.edu/`

Introduction on Question Answering Systems

- WATSON[3]
  - well known from the Jeopardy show
  - industrial applicability in several domains
  - website: `https://www.ibm.com/watson/`
- Siri[4]
  - answering of (spoken) user questions targeting predefined domains
  - knowledge base representing the iOS functionality
  - common knowledge
- many more: LUNAR (1977), PHLIQA 1 (1978), AquaLog (2004), YodaQA (demo, 2015), ...

State-of-the-Art of QA Systems

Qanary-based QA system: WDAqua QA

- on-top of *Qanary* framework
- targets: DBpedia, Wikidata, MusicBrainz (open music encyclopedia) and DBLP (computer science bibliography)
- custom implementation of answer computation
- *Qanary*-compatible front-end "Trill"
- demo: www.wdaqua.eu/qa

Existing QA systems

Observations

- state of the art not as advanced as expected
- see also QALD challenge

Reasons: How are question answering systems created?

- in general: hard and complex task
- cumbersome and inefficient
  - lack of *methodology* for creating question answering systems

Processing Steps within QA systems

- Query Analysis and Classification
  - Named Entity Recognition
  - Entity Linking, Named Entity Disambiguation
  - Relation Detection
  - Query Type Detection
- Query (Candidate) Building (e.g., SPARQL, SQL, Query DSL, . . .)
- Query (Candidate) Ranking (e.g., learning to rank using a gold standard)
- Answer Generation (e.g., Natural Language Generation, data visualization, . . .)
- Answer Validation (Feedback)

→ **many similar tasks and distinguished technology**

Note: Sometimes steps are not needed or need to be executed several times (loops) to take advantage of the available knowledge. A good QA framework should not request limitations here (*Qanary* has no such limitations).

Motivation for using a QA framework

Observations and Requirements

Observations

- limited compatibility
- use predefined QA process
- limited semantics

Derived demands

+ interoperable infrastructure
+ exchangeable components
+ flexible granularity
+ isolation of components

Goals

1. easy-to-build QA systems on-top of reusable components
2. establish an ecosystem of components for QA systems

→ efficient research steps → enabling of synergies between PhD topics

→ **best-of-breed QA system & components for use cases and research topics**

Qanary Methodology and Technical Framework

Idea: Knowledge-driven QA system representation

Requirements of knowledge perspective

1. abstract knowledge representation: qa vocabulary
   - represent all the available knowledge about a question
     - + representation of knowledge about question separated from process
     - + includes trust & provenance
     - + self-describing, reusable and extensible
     - + enables efficient collaboration on a data-level
     - + agnostic to question format (text, structured, audio, . . .)
     - + agnostic to question answering processing steps and implementation
2. align the input/output of the each component in a QA process
   - required input mapped from KB
   - computed output mapped into KB
   - mapping on a logical and sound level

→ *Qanary* methodology for creating question answering systems

Knowledge Representation using the qa Vocabulary

Abstract Knowledge Representation

Idea

Represent all the knowledge about a question using a RDF vocabulary

requirements for knowledge representation

- self-describing, sound knowledge representation
- represent provenance for (all) information
- represent trust for (all) information

derived technology stack

- Resource Description Framework (RDF)
- Web Annotation Data Model (WADM)
- question answering vocabulary (qa)

Resource Description Framework (RDF)

Introduction to RDF (slides by Manolis Koubarakis)
`http://cgi.di.uoa.gr/~pms509/past_lectures/introduction-to-rdf.pdf`

Web Annotation Data Model (WADM)

Web Annotation Data Model (WADM)

(W3C Working Draft 15 October 2015, http://www.w3.org/TR/annotation-model)

- oa:Annotation
- oa:hasTarget
- oa:hasBody
- oa:annotatedAt
- oa:annotatedBy

```
<myIRI> a oa:Annotation;
   oa:hasTarget <questionIRI> ;
   oa:hasBody <TextSelector> ;
   oa:annotatedBy <DBpediaSpotlight> ;
   oa:annotatedAt "..."^^xsd:date ;
```

qa Vocabulary

introducing new QA-related concepts on-top of WADM:

```
qa:Question
    rdfs:subClassOf oa:Annotation.
qa:Answer, ...
qa:Dataset, ...
qa:AnnotationQuestion, ...
...
```

- K. Singh, A. Both, D. Diefenbach, and S. Shekarpour. "Towards a message-driven vocabulary for promoting the interoperability of question answering systems." In Proc. of the 10th IEEE Int. Conf. on Semantic Computing (ICSC), 2016
- website: `https://github.com/WDAqua/QAOntology`

From knowledge representation to methodology

Conclusion: Advantages of using an ontology

- agnostic to question format (text, structured, audio, . . .)
- agnostic to question answering processing steps
- agnostic to implementation
  - programming language
  - component granularity

From knowledge representation to methodology

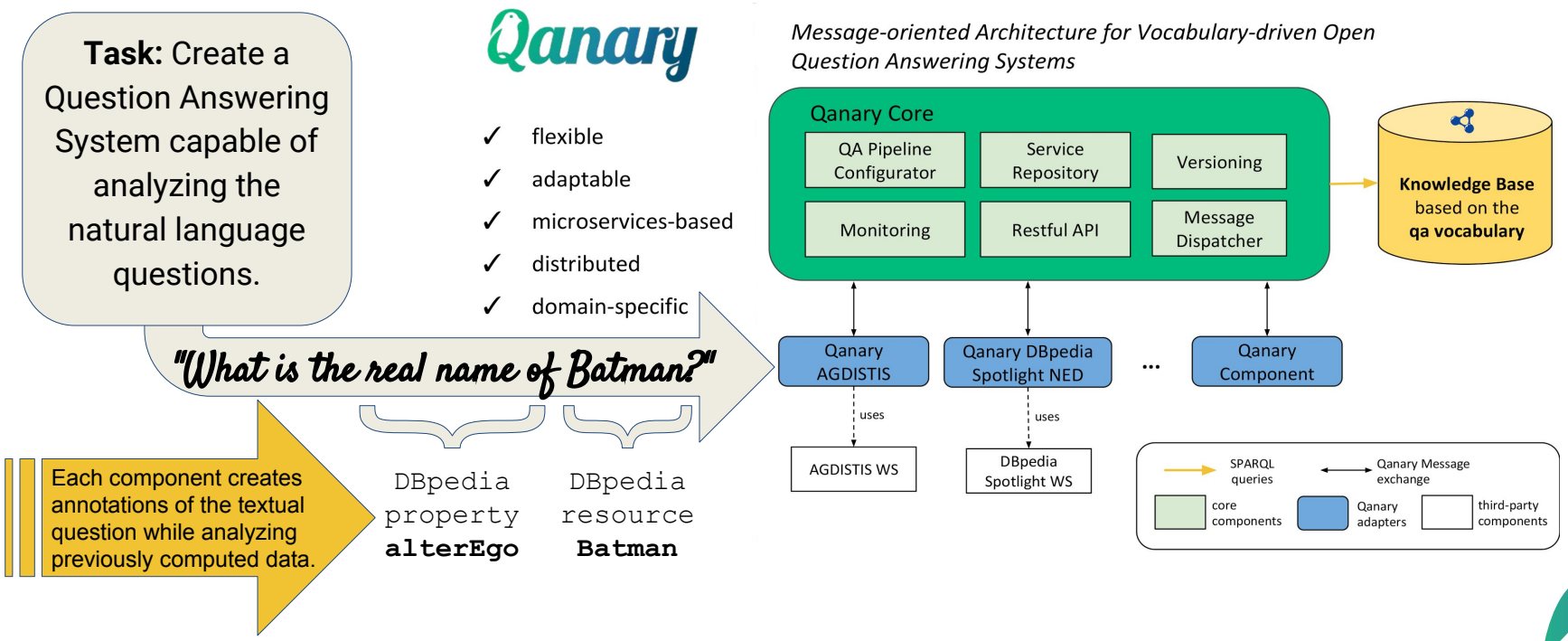
Methodology

1. abstract knowledge representation
   - advantage: independent representation
2. align the input/output of the each component
   - on a logical and sound level

Qanary Methodology

Overview

A trivial Question Answering system



It's about the components, stupid.

- an *agile* QA framework can only provide common features
  - central data access, logging, ...
- any particular problem solving/algorithm needs to be separated from the pipeline

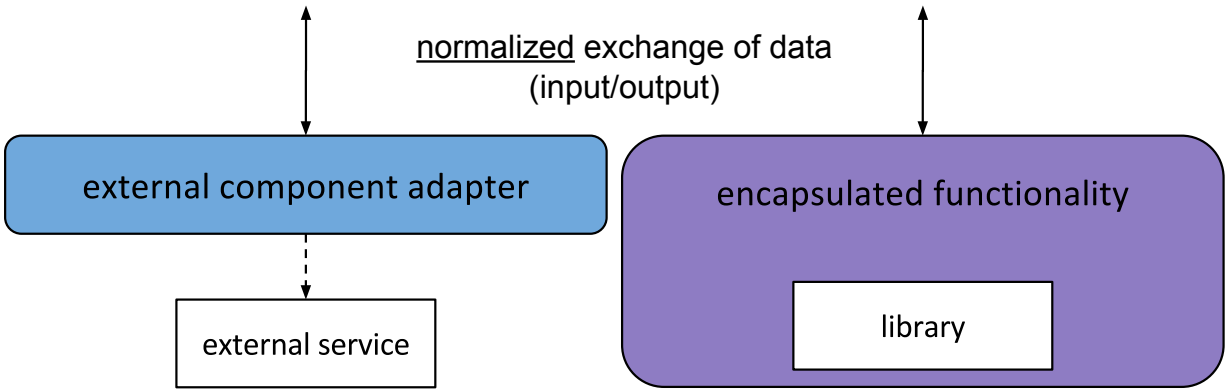
→ create exchangeable, isolated components only communicating via data

→ component data needs to be mapped/aligned to the data of the QA process

Component data alignment

Goal: Establish common ground for the research community

Two options:



alignment of input/output of each component with qa

- input represented using qa (RDF)
  → input required for the component C
- output from the component C
  → output represented using qa (RDF)

Component data alignment

alignment of input/output of each component with qa

if component provides output using a presentation as

. . . semantic data (RDF)
- logical representation of alignment
  - ontology alignment (OWL, DOL)
  - SPARQL query

. . . non-semantic data (API, JSON, XML, CSV, . . .)
- SPARQL query

Note: many options for alignment

- NER/NED
  - DBpedia Spotlight (NIF)
    P. N. Mendes, M. Jakob, A. Garca-Silva, and Ch. Bizer: "DBpedia Spotlight:shedding light on the web of documents." In I-SEMANTICS, 2011
- relation detection
  - PATTY
    N. Nakashole, G. Weikum, and F. M. Suchanek. PATTY: "A taxonomy of relational patterns with semantic types." In EMNLP-CoNLL, 2012
- query construction
  - SINA
    S. Shekarpour, E. Marx, A.-C.N. Ngomo, and S. Auer. SINA: "Semantic interpretation of user queries for question answering on interlinked data." Web Semantics: Science, Services and Agents on the WWW, 2015

Component data alignment: NED

- create component's input:
  - fetch question URI (from Qanary triplestore)
- processing:
  - retrieve textual question representation from URI
  - compute named entities within the text
- store component's output:
  - for each named entity:
    - create a `oa:TextSelector` within the Qanary triplestore containing the positions of the particular Named Entity

→ benefit: easily replace the NED component

→ benefit: measure quality against exchangeable relation detection and query construction components

Component data alignment: Relation Detection

Relation Detection Example

- create component's input:
  - fetch question URI (from Qanary triplestore)
  - fetch Named Entities which are already available
- processing:
  - retrieve textual question representation from URI
  - compute relations within the text
- store component's output:
  - for each detected relation:
    - create a relation resource within the Qanary triplestore (using a `oa:TextSelector` to mark the positions)

→ benefit: any improvement on the NED component (i.e., replace) will improve the quality here

→ benefit: measure quality against exchangeable query construction components

Component data alignment: Query Construction

SPARQL Query Construction

- create component's input:
  - fetch Named Entities (which are already available)
  - fetch Relations (which are already available)
- processing:
  - compute SPARQL
- store component's output:
  - for each created SPARQL:
    - store a resource/SPARQL in the Qanary knowledge base

→ benefit: any improvement on the NED component (i.e., replace) will improve the quality here

→ benefit: any improvement on the Relation detection component (i.e., replace) will improve the quality here

Component data alignment: Relation Detection

Relation Detection Example

- create component's input:
  - fetch question URI (from Qanary triplestore)
  - fetch Named Entities which are already available
- processing:
  - retrieve textual question representation from URI
  - compute relations within the text
- store component's output:
  - for each detected relation:
    - create a relation resource within the Qanary triplestore (using a `oa:TextSelector` to mark the positions)

→ benefit: any improvement on the NED component (i.e., replace) will improve the quality here

→ benefit: measure quality against exchangeable query construction components

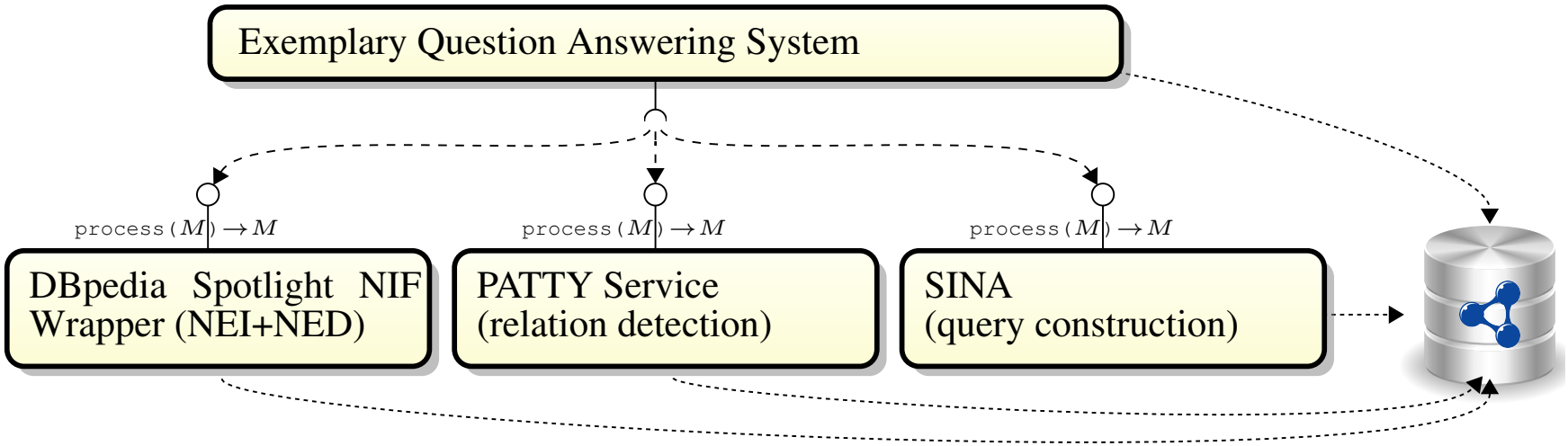
Component data alignment: NED

- create component's input:
  - fetch question URI (from Qanary triplestore)
- processing:
  - retrieve textual question representation from URI
  - compute named entities within the text
- store component's output:
  - for each named entity:
    - create a `oa:TextSelector` within the Qanary triplestore containing the positions of the particular Named Entity

→ benefit: easily replace the NED component

→ benefit: measure quality against exchangeable relation detection and query construction components

Case Study



Component

1. DBpedia Spotlight
2. PATTY
3. SINA query execution

Process within components

1. retrieve data from KB
2. process data
3. extend KB

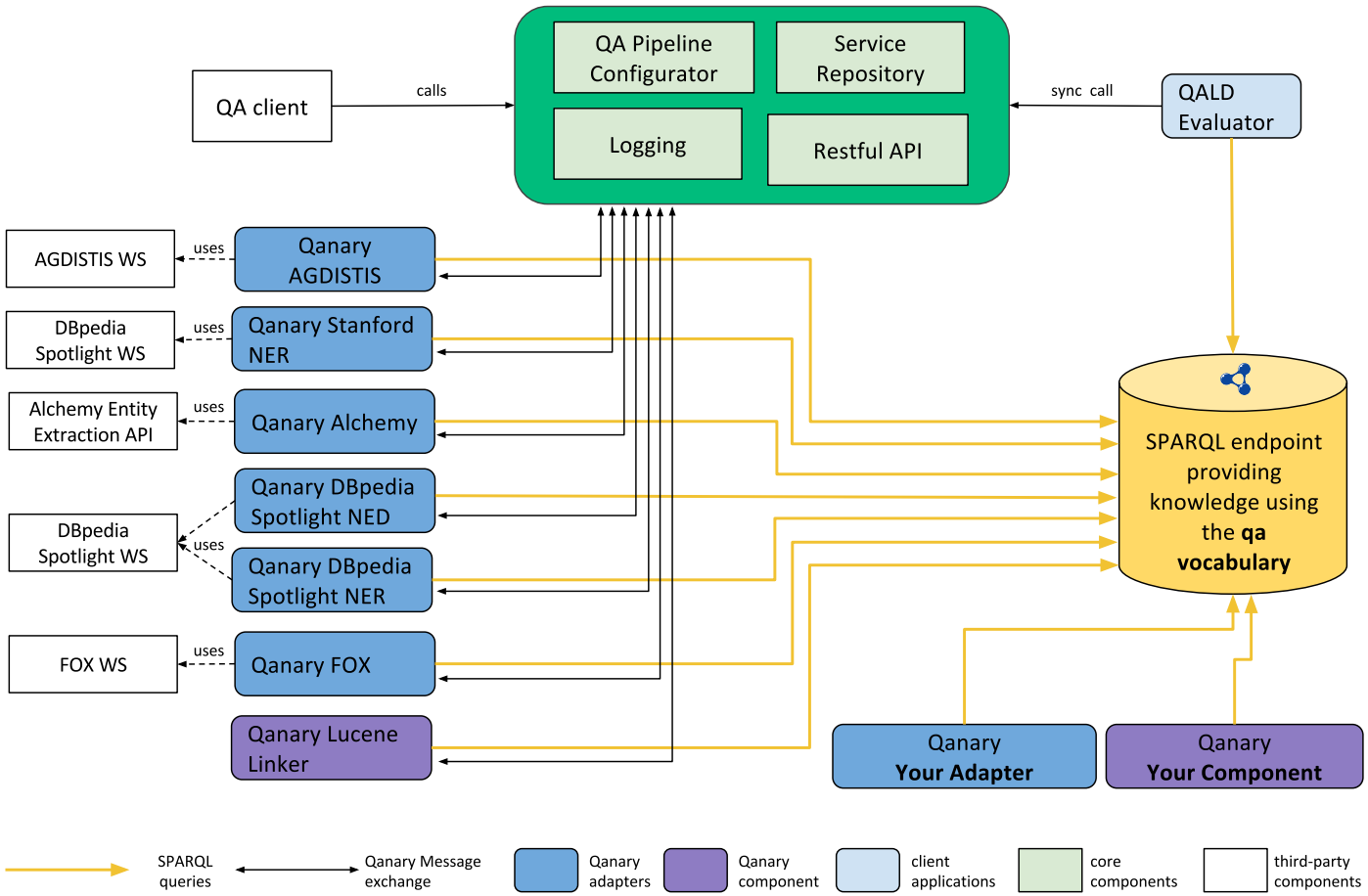
→ vocabulary-driven, component-oriented QA system possible

Available Architecture

- goal: easy-to-use framework for creating QA systems

Qanary

Message-oriented Architecture for vocabulary-driven open Question Answering systems



Outlook/Roadmap

- goal: enable infrastructure for optimizing/training of data interpretation
  - establish a methodology for representing goal standards within the qa vocabulary
  - provide a component for training on-top of ontology

  → best-of-breed QA system for your scope of application
- goal: reduce integration efforts (beyond RDF)
  - provide RESTful service interfaces for read/write access

  → even easier integration in external systems
- goal: automatic QA process creation
  - express/analyze data requirements for components

  → you define only your component, Qanary fulfills requirements
- goal: *provide benefits for your work*

Take Away: Qanary methodology

- Qanary: knowledge-driven methodology for QA systems
  - *and* reference implementation of methodology, too
- build on-top of the qa vocabulary
- agile approach for creating QA systems
  - interoperable infrastructure, exchangeable components, flexible granularity, isolation of components,
  - collects data in a sound way (provides support for AI components, particularly ensemble learning), does not fix the QA process to a template, allows concurrent executions, enables multi-path execution, freedom of candidate/option filtering (your/developers choice)
- *ecosystem of QA components* enabling best-of-breed approaches for your research topics

Join Qanary at Github!

github.com/**WDAqua/Qanary**

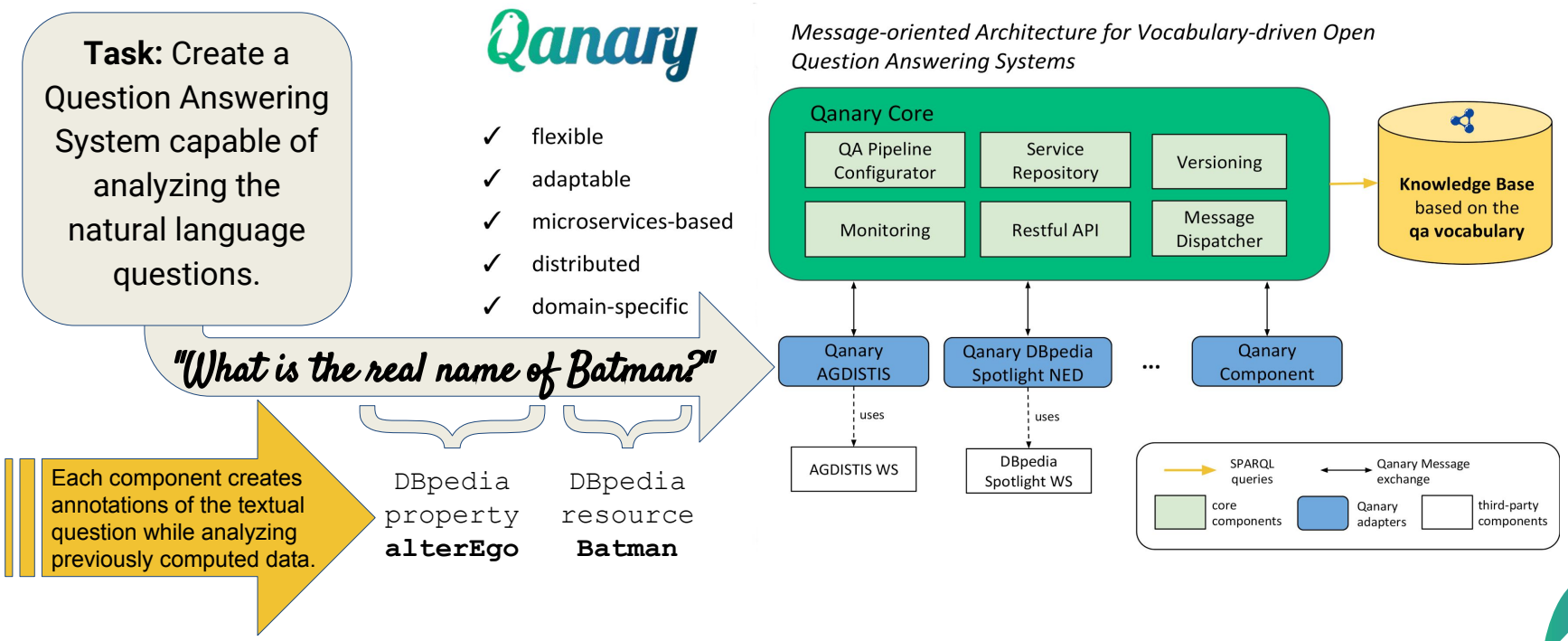
Coffee Break

Have break and prepare your notebook.
In the following practical session you will need:

- Internet connection
- text editor or any Ontology Designer
- Git client
- Java and Maven
- Stardog triplestore (free version)

Our Goal

Implementation of a trivial Question Answering system using *Qanary*



Let's define pairs/groups using the ranking ...

Interactive Session: Solution Definition

Preparation (15 min interactive session)

Given questions:

- "What is the real name of Batman?"
- "Is Bruce Wayne the real name of Batman?"
- "How many partners had Batman?"

Your Tasks

- model the required annotations for answering these questions
- write the SPARQL query to retrieve the answers for these queries

Note: Typically, the result of a QA process is not a SPARQL query. Due to time constraints, we exclude the mostly following Answer Generation (e.g., using Natural Language Generation or visualizations) from this exercise. See wolframalpha.com from inspiration.

Coding Session: Implement your first QA system from existing components

1. Step: Implement your first QA system

We follow the description on github.com/WDAqua/Qanary/wiki/Demo:-How-to-Create-a-Question-Answering-System-capable-of-Analyzing-the-Question-%22What-is-the-real-name-of-Batman%3F%22

- git checkout Qanary ecosystem's components
- run components
- run Qanary QA system template
- configure your pipeline
- run the pipeline
- test your QA system with some questions on DBpedia
- done

Validate the quality of your QA system

2. Step: Validate the quality of your QA system

- interactive validation using TRILL front-end from Qanary ecosystem
- use Qanary QALD validator to compute precision, recall and f-measure

Improve and revalidate your QA system

3. Step: Improve and revalidate your QA system

- solve questions not implemented before ...
  - pick from prepared list
  - define test cases
  - extend functionality
  - validate results in triplestore
- ...

4. Step: Solve new QA tasks

- extend the qa vocabulary
- choose existing QA components supporting your task
- implement new QA component for your new use case
- extend test cases and validate your work

Final Remarks

Summary

- compact overview about Question Answering (QA) and its challenges
- $Qanary$ methodology, the RDF vocabulary qa and the corresponding component-oriented $Qanary$ framework (reference implementation)
- advantage of the Qanary ecosystem for rapid research results
- learn to iteratively build, validate and improve your own QA system using the $Qanary$ framework
- you built a QA system capable of answering generic question in a specific domain (not only the exemplary questions)

→ I am looking forward to your contribution to the research community to improve the state-of-the-art

Take Away: Qanary methodology

- Qanary: knowledge-driven methodology for QA systems
- build on-top of the qa vocabulary (i.e., knowledge-driven approach)
- agile approach for creating QA systems
  - interoperable infrastructure, exchangeable components, flexible granularity, isolation of components, supports AI-approach
- today, was your first step towards participating in the $Qanary\ ecosystem\ of\ QA\ components$ enabling best-of-breed approaches for future QA systems

Join Qanary at GitHub!

github.com/**WDAqua/Qanary**

Andreas Both
contact@andreasboth.de
xing.com/profile/Andreas_Both6
linkedin.com/in/andreas-both-9426722