IADIS International Journal on Computer Science and Information Systems
Vol. 20, No. 1, pp. 40-56
ISSN: 1646-3692

POST-HOC NATURAL-LANGUAGE EXPLANATIONS OF COMPONENT-BASED KNOWLEDGE GRAPH QUESTION ANSWERING SYSTEMS GENERATED BY LLMS

Dennis Schiese, Aleksandr Perevalov and Andreas Both
Leipzig University of Applied Sciences, Karl-Liebknecht-Straße 132, 04277 Leipzig, Germany

ABSTRACT

Modern software systems have become so complex that explaining their decisions poses significant challenges for both developers and users. This article addresses the explainability of component-based Knowledge Graph Question Answering (KGQA) systems, where components often rely on AI-driven processes. Such processes can be opaque, making it difficult even for KGQA experts to interpret the underlying behavior and outcomes. To tackle this issue, we propose an approach that leverages the input and output data flows of system components as a basis for representing their behavior and generating explanations. This enables users to better understand how decisions are made. In the KGQA framework considered here, component data flows are expressed as SPARQL queries (inputs) and RDF triples (outputs). Consequently, our work also provides insights into verbalizing these data types. Through experiments, we evaluate our approach, comparing template-based explanation generation (baseline) with automatic generation using Large Language Models (LLMs) configured in various ways. The results demonstrate that LLM-generated explanations are of high quality and generally outperform template-based methods based on user evaluations. This approach thus facilitates the automated natural-language explanation of KGQA components' behavior and decisions, contextualized within RDF and SPARQL representations.

KEYWORDS

Explainable AI, Large Language Models, Knowledge Graphs, RDF, SPARQL, Question Answering

1. INTRODUCTION

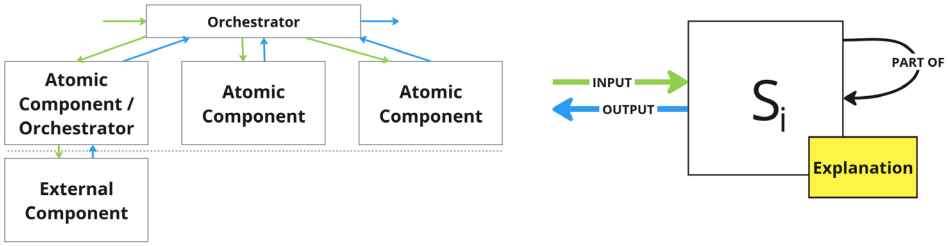
In times of rapid development of artificial intelligence (AI) applications, where new ones are created daily and the majority of them have become indispensable in everyday life. However, many users do not have a certain basic understanding of this technology's behavior. In a study

conducted in 2021, only 28% of respondents trusted AI systems (Gillespie et al. 2021). Nevertheless, more people are in favor of further development and research into AI than are against it (Gillespie et al. 2021). The ability to explain the decisions of AI-driven systems is an important step in addressing the lack of trust among its users. In addition, developers and researchers would also benefit from better traceability (Rosenfeld & Richardson 2019).

Nowadays, many developed software systems are component-based systems, i.e., they are composed of multiple blocks isolated from each other (e.g., web services, libraries, packages). A number of these systems follow the orchestration pattern (e.g., Figure 1a). Although the way how the components are called is transparent, their internal behavior is usually encapsulated. This lack of transparency is particularly increased by the use of AI models (e.g., language models) within the components. This limits the traceability of the behavior of such systems and increases the need for explainability. In particular, debugging (distributed) component-based systems is cumbersome and time-consuming. However, we follow here the hypothesis that due to their flexibility, component-based systems offer a decisive advantage in terms of explainability compared to monolithic systems, as different stages of a process can be considered separately, allowing for more detailed explanations. In general, the explanation of such systems typically involves providing a natural-language text, e.g., for clarifying the processes. Process explanation is central to AI methods such as machine learning and deep learning, with existing approaches such as (Adadi & Berrada 2018, Barredo Arrieta et al. 2020, Burkart & Huber 2021, Haar et al. 2023). Data explanation, on the other hand, aims to make system outputs understandable and transparent by explaining their individual components.

In this article, we present an approach for creating explainable component-based Question Answering (QA) systems, followed by a case study where we utilize the semantics of the internal data flows to generate explanations with a rule-based (template-based) and generative approach. Here, we consider explanations for end users (nevertheless experts in their field), i.e., they must be interpretable for humans. Due to the lack of reference works and datasets, we won't be able to compare our results with those. For the introduction of such a dataset, we will use the Qanary framework (Both et al. 2016) and derived QA systems. Here, we consider the data flows of components integrated into a component-based QA system. The used QA framework is well-prepared for this approach, as it already explicitly represents the components' input data as SPARQL[1] queries (i.e., the component requests data while using SPARQL from a central process memory) and the output data as RDF[2] triples. Hence, for all components, the data flow is transparent. To validate our approach, we implement a template-based setting and one based on Large Language Models (LLM-based) for verbalizing the data flow (i.e., converting to natural language). Both settings are compared with each other using quantitative (automatic) and qualitative (human experts) settings. We derive the following research questions: (RQ1) Can the data flow of a component be verbalized for the purpose of explaining the component's behavior?; (RQ2) What is the quantitative and qualitative difference between template-based and LLM-based prompts?

[1] https://www.w3.org/TR/sparql11-overview/
[2] Resource Description Framework, cf. https://www.w3.org/RDF/



(a) Exemplary orchestrated system. The green (input) and blue (output) lines represent the data flows. The external component below the dotted line, represents an external component, whereas the parent component acts as an atomic component and orchestrator at the same time.

(b) General, generalist component model with one input and one output stream. The Part-Of label represents the possible aggregation of components of this type. The yellow explanation box indicates the possibility of explaining this component.

Figure 1. Example of a component-based system and the generalized component model

The article is organized as follows. In Section 2, we give an overview of the previous work in this field. Section 3 defines the approach for explainable QA systems, and presents technical details on the implementation. The experimental setup and evaluation are presented in Section 4. We conclude the article in Section 5.

2. RELATED WORK

The concept of *explainability* is inherently abstract across any area due to the lack of a consistent definition (Chazette et al. 2021). In computer science, however, this term plays an increasingly important role as software systems become larger and more complex over time. Furthermore, the growing deployment of AI in these systems results in a certain degree of opacity, effectively transforming them into black boxes. Consequently, numerous approaches have been proposed to address this issue. Among these, the majority concentrate on justifying AI algorithms, given the growing utilization of artificial intelligence. This has led to the emergence of a separate research area, namely explainable AI (xAI). In this field, different approaches have emerged, each to enhance the explainability of machine- and deep-learning algorithms (Pope et al. 2019, Burkart & Huber 2021, Haar et al. 2023).

Verbalizing SPARQL queries, which can be seen as explaining them, has mostly been approached in an inverse way, e.g., by Ngonga Ngomo et al. (2013), Yin et al. (2021), and Sander (2014). This consideration is interesting in the context of answering questions over linked data, but not relevant to this work. Verbalizing or explaining SPARQL queries, on the other hand, has been the focus of limited research. Existing approaches can be broadly categorized as either rule-based (template-based) or generative AI-based. Notable rule-based approaches include SPARTIQULATION (Ell et al. (2015)) and SPARQL2NL (Ngonga Ngomo et al. (2013)). Both showed promising results, yet the authors note room for improvement, especially with complex queries where the effectiveness diminished. As the implementation effort was considered disproportionate to the potential impact, these methods were not used in this work. In contrast, the study presented by Perevalov & Both (2024) uses generative AI,

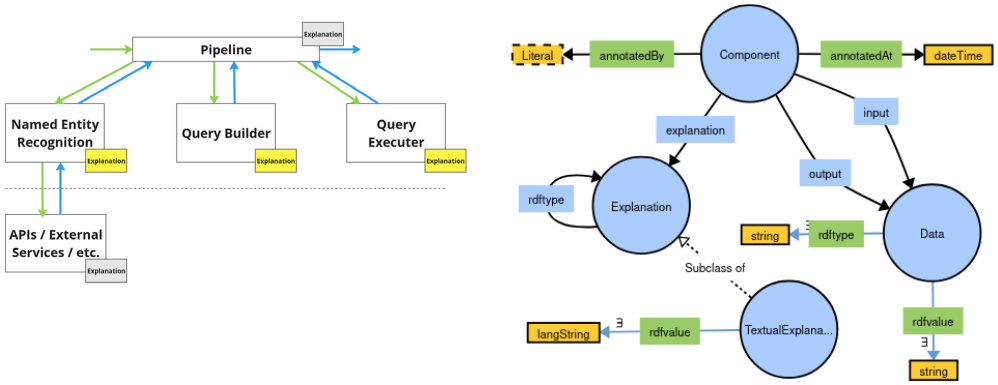
specifically large language models (LLMs). Since this study was published during the preparation of this article, a direct comparison of the results was not possible, although the approach in this article will use LLMs to translate SPARQL queries, too.

The second challenge in the scope of explanation generation is the *verbalization of grounded*[3] RDF triples. Although there is little research on this topic, a common approach aims to exploit the rich semantics that RDF offers by definition. In this context, Sun et al. found in 2006 that a large number of predicates (from natural language sentences) can be classified into RDF triples (Mellish & Sun 2006). In 2007, they leveraged this finding and presented an approach that uses this notion (Sun & Mellish 2007). In addition to these first steps towards verbalizing RDF triples, several other approaches have been developed, including several rule-based ones (Lapalme (2020), Moussallem, Gnaneshwar, Ferreira & Ngomo (2020), Moussallem, Speck & Ngonga Ngomo (2020)) and neural network (Moussallem, Speck & Ngonga Ngomo (2020)) approach(es). In our work, we first adopt the concept of templates in our template-based approach. At the same time, the generation of AI-based explanations is an experimental addition that aims to address the limitations of rule-based approaches in component-based systems outlined throughout the article.

As previously discussed in the context of verbalizing SPARQL queries and RDF triples, generative AI methods have also been employed in recent times for the verbalization of this data. In particular, *Large Language Models (LLMs)* have been utilized in this context. This is not surprising, given that these models have been demonstrated to be capable of generating (high-quality) text and thereby creating explanations (whose quality must be evaluated) from diverse data sources. Furthermore, the capacity to adapt such models, for instance through fine-tuning, has the potential to enhance their capabilities. In order to assess the potential quality of such a fine-tuned model, we rely on one of the most powerful and promising general-purpose (text-generation focused) LLMs, namely GPT-3.5 and GPT-4, which have been developed by OpenAI[4] and demonstrate consistently strong performance across a range of tasks (in the area of natural language generation) (Li et al. 2024, Ribeiro et al. 2021, Chen et al. 2020).

In addition to natural language generation, question-answering (QA) systems address the field of natural language understanding. Within this area, various *Question Answering frameworks* have been developed, which can be broadly categorized as component-based or monolithic. Concerning explainable, component-based QA systems, only the approach QA2Explanation has been found. It was proposed by Shekarpour et al. (Shekarpour et al. 2020). This approach utilizes the Frankenstein approach[5] (Singh et al. 2018a, Singh et al. 2018b) – an extension of the component-based Qanary framework (Both et al. 2016) – to provide valid explanations for each pipeline stage (i.e., processing steps).

[3] Grounded or explicit triples are triples without variables, cf. https://www.w3.org/TR/rdf-sparql-query/
[4] cf. https://platform.openai.com/docs/models
[5] https://github.com/WDAqua/Frankenstein



(a) Exemplary Qanary QA system following Figure 1a. Grayed-out explanation boxes indicate that the explanation of this component is not within the scope of this work. The green and blue lines indicate the data flows (input/output data stream) controlled by the parent component, the pipeline.

(b) Ontology that represents the explanation for components within a component-based system. The primary focus is on the data that is separated into output and input as well as the explanation. Thus, the explanation comprises the type and the actual natural language text (as we defined the type TextualExplanation).

Figure 2. An exemplary Qanary Question Answering system and a foundational ontology that serves as the foundation for representing the most crucial aspects for explaining components within component-based systems

By augmenting the framework with classifiers, they return the expected success for components to fulfill the question-answering task and select appropriate templates. Qualitative analysis of these explanations revealed that interviewees accepted them when provided, enhancing confidence in the QA process and underscoring the value of data-driven explanations for traceability. However, the authors also emphasize that the explanations depend on various factors from other disciplines, such as social science and cognitive decision-making. This reiterates the previously mentioned challenge associated with the term *explainability*. Similar results can also be found in (Hoffman et al. 2017). From the existing QA frameworks, only the Qanary framework offers the functionality required to answer our research question, as it is a component-based QA framework that manages the components as web services and explicitly represents the input and output data.

3. APPROACH AND IMPLEMENTATION

As a foundation for explainable QA systems, we present a concept that is aimed at systems following the orchestration pattern (cf. Figure 1a). Therefore, we define a model as well as an ontology capable of representing the intended information as general and extensible as possible. In examining component-based systems, it may be beneficial to consider the individual components for further analysis. Then, by focusing on the most relevant aspects in terms of

components' explainability (i.e., component input and output) in an orchestration-based system, we derived the component model depicted in Figure 1b, where each component might integrate other components (expressed by the part-of relation). In addition, like the orchestrated system in Figure 1a, the general component model contains two data streams – *INPUT* and *OUTPUT* – and an explanation of the component's behavior for the considered specific execution (i.e., a particular process).

```
SELECT *
FROM ?graph
WHERE {
    ?annotationId rdf:type qa:AnnotationOfInstance .
    ?annotationId oa:hasTarget [
        a oa:SpecificResource;
          oa:hasSource ?hasSource;
          oa:hasSelector [
            a oa:TextPositionSelector;
              oa:start ?start;
              oa:end ?end
          ]
    ].
    ?annotationId oa:hasBody ?hasBody ;
        oa:annotatedBy ?annotatedBy ;
        oa:annotatedAt ?annotatedAt .
        OPTIONAL {
            ?annotationId qa:score ?score .
        }
}
```

(a) This SPARQL query is an exemplary INPUT data stream (I_n) used by a component C_n. There is a template for almost every distinct SPARQL query in the aforementioned Qanary repository. For generative explanations, the complete SPARQL query is passed to the used Large Language Model. Note: The prefix definitions are excluded from the SPARQL query.

```
All Annotations of the
type AnnotationOfInstance
have been requested,
which inherits the date
of its creation, the URI
for the origin question,
a knowledge-graph
resource as well as the
correlating start and end
position for the found
entity. If
provided, a score is
fetched too.
```

(b) Exemplary explanation text for a specific input data type I_n. The templates for the input data types are predefined and thus, they do not contain any placeholders. The rationale behind this decision is as follows: The requested data doesn't vary much (only the requested graph).

Figure 3. Exemplary template for an explanation and input data for a component C_n

We apply this concept to concrete QA systems that are built using the Qanary framework[6] (cf. Section 2), as these are orchestrated systems (for example as shown in Figure 2a), comprising a pipeline (orchestrator) and (atomic) components. All of these components are web services following a particular scheme to transfer data autonomously (after being activated in a process):

- They fetch the required data from a centralized knowledge base (a triplestore) via SPARQL query. Hence, the input is represented by a SPARQL SELECT query.
- After computation, each Qanary component stores data in process memory, making it accessible to follow-up components. This data, represented as grounded RDF triples, is traceable to its source via Qanary-specific metadata.

[6] https://github.com/WDAqua/Qanary

```
"The component
${component} has
added
${numberOfAnnotations}
annotation(s) to the
graph: "
```

(a) Explanation template for the prefix that represents the 1st part of the output data explanation. There always exists exactly one prefix per explanation.

```
"i. on ${annotatedAt}
&{with a confidence of
${score}}& and the
resource ${hasBody}
[...]"
```

(b) Explanation template for the output data item that represents the 2nd part of the output data explanation. An explanation can contain multiple occurrences of this, depending on the number of data items existing for the component (see Figure 4c. There, each float-like identifier represents one of those *data items*). The "&{}" placeholder indicates a conditional placeholder that is only resolved whenever the inner placeholder contains a valid value.

```
<0.4310899118106299> rdf:type qa:AnnotationOfInstance .
<0.4310899118106299> oa:hasBody
<http://dbpedia.org/resource/Civilization> .
<0.4310899118106299> oa:hasTarget
<0d5aae10afd67914bf9f788f01e17632> .
<0.4310899118106299> oa:annotatedAt
"2023-10-20T09:32:08.322985Z"^^xs:dateTime .
<0.4310899118106299> oa:annotatedBy
<urn:qanary:NED-DBpediaSpotlight> .
<0.4310899118106299> qa:score 0.8546948727066047 .
<0d5aae10afd67914bf9f788f01e17632> rdf:type oa:SpecificResource .
<0d5aae10afd67914bf9f788f01e17632> oa:hasSelector
<a14ca4a20af83dbd2d4d988b5145066f> .
<a14ca4a20af83dbd2d4d988b5145066f> rdf:type oa:TextPositionSelector
.
<a14ca4a20af83dbd2d4d988b5145066f> oa:start 9 .
<a14ca4a20af83dbd2d4d988b5145066f> oa:end 30 .
```

(c) Exemplary output data for a component C_n. The structure is very similar to the turtle format and represents grounded triples. To make it sentence-like, for each triple, a dot was appended.

```
The component urn:qanary:NED-
DBpediaSpotlight has added 1 annotation(s)
to the graph:
1. on 2023-10-20T09:32:08.322985Z with a
confidence of 0.8546948727066047 and the
resource
http://dbpedia.org/resource/Civilization
```

(d) Explanation following the templates for this data type O_n and the concrete data (exemplary from Figure 4c). As a score is provided in the referenced data set, the conditional placeholder is replaced with the actual value.

Figure 4. Exemplary Template, data, and explanation for a component C_n. The explanation consists of two parts, namely the prefix and the sum of all annotation explanations. The placeholders are replaced with the values from the data set. Following the standard RDF definitions, triples can be separated into a subject, predicate, and object. Thus, more precisely, the objects are being taken as values

Hence, the task is to verbalize the specific input data (SPARQL INSERT query) and output data (grounded RDF triples) to reflect the components' behavior regarding a user-demanded QA process identified by the question that is the starting point of any specific QA process (i.e., one that was previously executed) in Qanary. To accomplish the objective, the explanations are to be generated in two ways: a) rule-/template-based and b) through the utilization of generative AI. Moreover, to represent the required data, we employed an ontology[7] that is visualized in Figure 2.

[7] The ontology is published unter DOI: 10.6084/m9.figshare.27979427

Template-based generation of component explanations. For the template-based approach, an external Java service[8] has been implemented to generate explanations based on the defined datasets. To do so, it queries the QA-process-related knowledge graph (that contains all the required information) and utilizes this fetched data to explain any component in the prior executed QA process. For both types of data (input or output) different approaches were used: The *input data* is explained through fixed templates without placeholders. These templates are defined once for all centrally[9] available SPARQL queries. In a concrete case, as depicted in Figure 3, the appropriate template for the executed SELECT query is chosen based on the requested data item (called annotation in the Qanary terminology). While this method provides a straightforward foundation, it is constrained in the long term as new queries necessitate a new manual explanation. The explanation of the *output data*, namely the RDF triples, also employs a template-based approach. In contrast to the input data, where only the graph within the SPARQL query differs, the data here is highly variable. Depending on the component, the question, or the existing data in the triplestore, the data changes accordingly. Consequently, the templates use placeholders that utilize the values from the queried data, as illustrated in Figure 4. Although the data set is highly variable, it still encompasses both shared and distinct variables. Accordingly, this necessitates a two-part explanation, which is structured as follows: (1) The prefix: Consists of the executed component and the number of data items (recognizable by different IDs) that were sent from the component to the QA system (stored in a knowledge graph, see Figure 4a); (2) The actual output data items: Explanation of each data item. The template for each item is selected based on the corresponding data type. For each item, additional pieces of information are provided, e.g., time and confidence score (see Figure 4b).

Following this structure, an explanation is shown in Figure 4d. The explanations generated by this method represent a systematic approach that demonstrates the applicability of the concept, as well as providing a foundation for further considerations.

LLM-based generation of component explanations. The second approach, which is the focus of the following experiments, investigates the ability of LLMs to verbalize data flows with the assistance of LLMs. When examining the potential benefits of this approach in comparison to the template-based approach, it becomes evident that the template-based approach has several limitations. These include the high costs associated with maintaining or extending templates and the inability to generalize, which in turn limits the automation of explanations for new annotation types. In order to create LLM-based explanations, the prompt templates shown in Figure 5a (output data) and Figure 5b (input data) were utilized.

4. EXPERIMENTS AND EVALUATION

As a prerequisite to the experiments section, we unite concrete Qanary components into "component sets", which serve to generalize the different existing data types[10]. Therefore, the notation (1) C_1, ..., C_n represents the used components, and (2) O_1, ..., O_n stands for the different (output) data types the components produce. The same applies to 2-shot experiments. For each metric, the best rating is highlighted (per row). In contrast, the input data (SPARQL SELECT

[8] Software released under DOI: 10.6084/m9.figshare.27979319

[9] The Qanary commons package defined all used SPARQL queries in a central place: https://github.com/WDAqua/Qanary/tree/master/qanary_commons

[10] In Qanary, these different data types are represented by different annotation types

queries) are referenced as I_1, ..., I_6[11]. In our experiments, the following components were used to produce the following data types:

1. O1: C1 = NED-DBpediaSpotlight,
   C2 = NED Dandelion,
   C3 = NED-Ontotext,
   C4 = NED-TagMe;

2. O2: C5 = NER-DBpediaSpotlight,
   C6 = TagMeNER,
   C7 = TextRazor,
   C8 = DandelionNER;

3. O3: C9 = REL Python Falcon,
   C10 = DiambiguationProperty-OKBQA;

4. O4: C11 = SINA,
   C12 = QAnswerQueryBuilderAndQueryCandidateFetcher,
   C13 = PlatypusQueryBuilder.

Subsequently, we intend to evaluate the experiments in qualitative and quantitative terms. It should be noted that the quantitative evaluation will be based on a specifically derived formula and will pertain only to the latter output-data explanation experiments, not the template-based explanations, which serve as the baseline in this context.

With regard to the qualitative evaluation, we recognize the problem of a generally applicable evaluation scheme for explanations (Adadi & Berrada 2018, Anjomshoae et al. 2019). That is why we concentrated on two aspects that aim to encompass the two dimensions of explanation evaluation as described by (Gilpin et al. 2018), namely interpretability and completeness. Thus, we use ‘correctness’ to assess completeness and ’usefulness’ to assess interpretability.

[11] The corresponding SPARQL queries (mappings) are part of the online appendix for all experimental data, DOI: https://doi.org/10.6084/m9.figshare.27927015.v1

```
Given the following context:
Here, we consider the data of a
Question Answering system. The data
describes the outcome of this
system.
As a user I'd like to understand
what happened inside that particular
component. For this purpose a (text-
based) explanation has to be
computed. For example, the following
explanation was created for the
question "<QUESTION_ID_EXAMPLE>"
from the given raw data.

The example explanation:
<EXAMPLE_EXPLANATION>
Given raw data:
<EXAMPLE_RDF_DATA>
Now, create an explanation for the
following RDF data:
<TASK_RDF_DATA_TEST>
Don't introduce your answer and only
return the result.
```

(a) Prompt for output data explanations

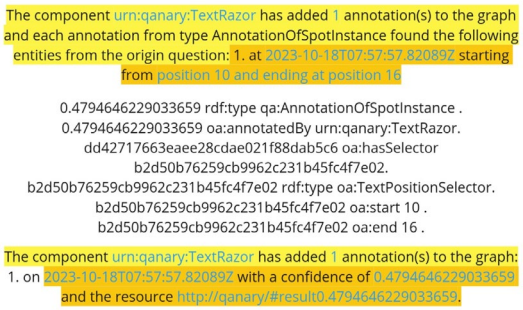
```
Given the following context:
Here, we consider the data of
a Question Answering system.
The data describes a SPARQL
query. As a user I'd like to
understand what the query
means and does. For this
purpose a (text-based)
explanation has to be
computed.

Here's an example explanation:
The query:
${EXAMPLE_QUERY}
The example explanation:
"${EXAMPLE_EXPLANATION}"
Now explain the following
query, used by the component
"${COMPONENT}":
${TEST_QUERY}
Don't use more than 3
sentences.
```

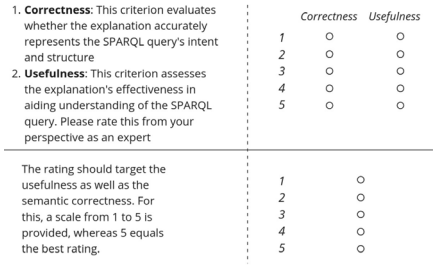
(b) Prompt for input data explanations

Figure 5. Prompt templates for input and output data

The evaluation of the **baseline explanations** and all subsequent experimental explanations has been carried out by three experts, each with a research background in the field of Question Answering and Linked Data. However, in order to get feedback from a wider audience and gain further insights, we've also implemented and published a demo with feedback functionality (Schiese et al. 2024). The result shows the quality of the template-based explanations for both types: input-data explanations and output-data explanations. Furthermore, the quality was rated on a 5-point Likert scale (with "5" being the best rating). The evaluation of the *input-data explanations* was based on the separated metrics' *correctness* and *usefulness*. For both metrics, the average was 3.67, whereas the majority was rated with 4. This in turn means that the explanations seem to be 73.4% correct. In terms of their usefulness, the experts consider them to be rather useful. The evaluation for the *output-data explanations* is based on the combination of the correctness and usefulness metrics. This is due to the fact that the values are based on factual information and therefore, they're less semantically influenced by the templates. Here, the majority of ratings was 4, too. On average, these were rated at 3.704. Since we can assume that the accuracy of the factual data is probably 100%, it is reasonable to conclude that these explanations are just as useful to the experts as the explanations of the input data.



(a) Exemplary comparison and quantitative evaluation of a template-based and generative explanation. The yellow part represents the prefix of the explanations, while everything after (orange) represents the explanation of the annotations. As can be seen, the prefixes consist of the component and the number of annotations in both explanations. However, the generative annotation explanation refers to the annotationId (0.479...) as a score and hallucinates a resource. Furthermore, the generative explanation does not mention the position of the entity. Finally, the prefix isn't depreciated, but the annotation is, because (1) it added a non-existent score and resource, and (2) it misses the position. This leads to QE = 3 + 1 = 4.



(b) Definitions of the evaluation tasks and metrics employed in the qualitative expert evaluation. These encompass both the explanations of the input data (presented above) and the explanations of the output data (presented below). With regard to the latter, the metrics were combined, given that the evaluation of the factual information (RDF triples) lacked relevance, as this has already been addressed in the quantitative analysis.

Figure 6. Evaluation: Example of output-data explanation and rating scheme

Concerning the LLM-based experiments, they were performed with the OpenAI[12] models GPT-3.5 and GPT-4[13]. Moreover, the employed settings comprised vanilla LLM configurations (i.e., no fine-tuning, due to the absence of a training dataset) and the Chat Completions API without modified parameters[14]. The question set consisted of the QALD-10[15] dataset with 394 questions. Finally, the used prompts are shown in Figures 5a and 5b.

Evaluation approach. As with the template-based explanation evaluation, these experiments were also evaluated qualitatively using the metrics and the questionnaire shown in Fig 6b. In the case of the output-data explanations, and as pointed out earlier, we carried out an additional quantitative analysis.

[12] https://openai.com/
[13] GPT-3.5 models: gpt-3.5-turbo / gpt-3.5-turbo-16k, GPT-4 model: gpt-4-0613
[14] The parameters are listed in the reference documentation for the Chat Completions API: https://platform.openai.com/docs/api-reference/chat/create. Only the message and the model were passed.
[15] https://github.com/WSE-research/QADO-datasets/releases/tag/v0.6.0

Table 1. Qualitative expert evaluation for both data types Ii and Oi. The displayed ratings represent the average quality across all experiments for the given setting. For the output data explanations, no zero-shot examples were considered. The same applies to 2-shot experiments for GPT-4. For each metric, the best rating is highlighted (per row)

		Template	Zero-shot prompt (LLM)			One-shot prompt (LLM)			Two-shot prompt (LLM)		
		-	GPT-3.5	GPT-4	Δ	GPT-3.5	GPT-4	Δ	GPT-3.5	GPT-4	Δ
I_i explanations	Correctness	3.67	4.50	4.50	±0.00	4.50	4.56	+0.06	4.50	**4.72**	+0.22
	Usefulness	3.67	3.83	3.78	−0.05	3.89	3.78	−0.11	4.06	**4.11**	+0.05
O_i explanations	Quality	3.70	∅	∅	∅	3.59	**3.71**	+0.11	∅	∅	∅

Table 2. Quantitative output-data explanation evaluation: 1-shot test series with quantitatively determined average values for each experimental setting (Exemplary output, and output to be explained). One experimental setting consisted of 50 experiments. The highlighted values represent the best result for each column and both used models

Evaluation		Tested data														
		O_1			O_2			O_3			O_4			dy		
		GPT-3.5	GPT-4	Δ	GPT-3.5	GPT-4	Δ	GPT-3.5	GPT-4	Δ	GPT-3.5	GPT-4	Δ	GPT-3.5	GPT-4	Δ
Example for prompt	O_1	5.29	5.60	+0.31	4.98	5.02	+0.04	5.84	**6.00**	+0.16	5.41	5.84	+0.43	5.38	5.62	+0.24
	O_2	5.33	5.47	+0.14	**5.83**	**6.00**	+0.17	5.58	5.94	+0.36	5.36	5.85	+0.49	**5.53**	5.82	+0.29
	O_3	**5.62**	5.76	+0.14	5.02	5.79	+0.77	**5.99**	**6.00**	+0.01	4.97	5.66	+0.69	5.40	5.80	+0.40
	O_4	5.36	**5.90**	+0.54	5.18	5.92	+0.74	5.59	5.98	+0.39	**5.88**	**5.98**	+0.10	5.50	**5.95**	+0.45

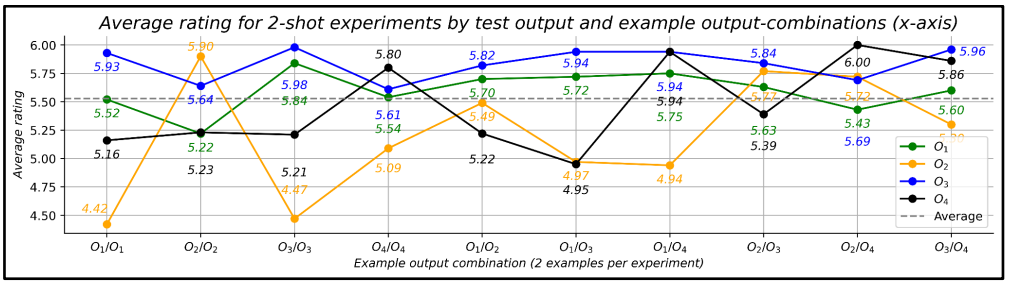


Figure 7. Quantitative output-data explanation evaluation: 2-shot test series with quantitatively determined average values for each test setup (test output and sample output combinations, 2 samples per prompt). One test setup consisted of 50 trials. The x-axis displays the output-data combinations passed as examples to the prompt. The dashed line represents the average across all experiments and is 5.53

Due to the variability of the RDF triples, which depended on factors such as the question, the component used, and the data available in the knowledge graph, we were faced with a significantly greater variety of data. This diversity is also reflected in the variance of the explanations as they utilize this data. Therefore, generating these explanations using LLMs required more adaptation, as more than just NLG capabilities were needed. In a preliminary study, we then found that the data referenced in the explanations were misused to varying degrees, which led us to decide on a quantitative evaluation. As a basis for this evaluation, we derived the following formula:

$Q_E = Rating_{Prefix} + ((\sum_{i=1}^{A_{Ann}} Q_{Ann_i})/A_{Ann})$. There, the quality of an experiment (Q_E) is a composition of the prefixes'[16] rating ($Rating_{Prefix}$) and the average rating of all annotations ($(\sum_{i=1}^{A_{Ann}} Q_{Ann_i})/A_{Ann}$), whereas A_{Ann} represents the number of explainable annotations. A depreciation of a rating took place: (1) For the prefix in the case of incorrect recognition: (a) of the component or, (b) of the number of annotations; (2) For the annotation on (a) missing values (multiple occurrences only evaluated once), (b) incorrect values (multiple occurrences only evaluated once), An exemplary evaluation following this formula is shown in Figure 6a.

4.1 Results

In the following, all results are displayed and briefly discussed. The raw data for each experiment, as well as the evaluation results, are accessible in the online appendix[17].

Input data explanations For the input data, explanations for 6 (I_1 to I_6) different SPARQL queries were generated. Zero-, one-, and two-shot experiments were carried out two times for each of these queries (one each with GPT-3.5 and GPT-4). Together with the template explanations, there were 42 explanations. Firstly, all generatively computed explanations achieved better results than the template-based ones. Furthermore, the differences between zero-, one-, and few-shot approaches seem to be small, but more examples never worsen the results. Regarding the used GPT models, it could not clearly be determined whether one was better than the other. In particular, GPT-3.5 produced better results for usefulness in zero- and one-shot experiments while GPT-4 did so for correctness in one- and two-shot examples as well as for usefulness in the two-shot experiments (cf. Table 1).

Quantitative (output data explanations). The results of the quantitative evaluation were analyzed as part of the study according to the 1- and few-shot experiment series (Table 2). In addition to anomalies, possible correlations (according to the Pearson correlation coefficient) between the overall quality and the number of annotations were also examined. For the 1-shot experiment series (see Table 2), the test datatype O_3 turned out to be the best with the highest average ratings. It was followed by O_1, O_4, and O_2. For all except O_1, experiments performed best when the example and test data types matched. In general, this is no surprise, because generative AI usually works best when the model is provided by concrete examples that only need to be repeated.

Running the prompts with the GPT-4 model led to a significant improvement in the results regarding the average values in each series of experiments. In this case, type O_3 achieved the highest possible average score in two cases and thus performed best, again. Consequently, an optimized recognition and processing of grounded RDF triples can be stated. Based on these results, the two-shot test series was carried out (i.e., two examples are provided in the prompt). Each of these comprised 10 example type combinations, which represented the combination of all existing types without taking the order into account. The overall results are presented in Figure 7.

Starting with the consideration of the test data type O_1, the average rating is between $5.22 \leq \text{Rating} \leq 5.84$. It was noticeable that the problem regarding missing scores and the C_1 component occurred here, too. The correlation coefficients were found to be in the range of

[16] The proposed output-data explanations can be divided into a prefix and the list of annotation explanations. The prefix includes information about the number of annotations and the component. This data was part of every dataset for the output-data explanations

[17] Raw data and evaluation results under DOI: 10.6084/m9.figshare.27927015

$0.249 \leq |r| \leq 0.595$. For the test type O_2, the range of average ratings is considerably broader, with a minimum of 4.42 and a maximum of 5.90. Furthermore, four experiment series exhibit ratings below 5. It was observed that the combination of data types with O_1 resulted in sub-optimal performance. In contrast, the combinations with O_2 exhibited the most favorable performance. The primary reason for the inferior outcomes can be attributed to the presence of missing and wrong values within the annotation's explanation. With regards to the component exhibiting the highest performance, C_6 achieved a score of 6 in 66.7% of cases (i.e., an excellent result). Finally, the calculated correlation coefficients were found to lie between $|r| < 0.30$ for five cases and $|r| > 0.30$ for another five cases.

Thirdly, the test series for O_3 was evaluated and, as with the corresponding one-shot test series, no anomalies were found. Nevertheless, the average ratings were below those of the one-shot test series. However, the observed discrepancy precludes any conclusion regarding the relative performance of the two experiment series. It was also found that the distribution of ratings across both components used was almost identical. The correlation coefficients were in the range of $0.015 \leq |r| \leq 0.431$. Finally, the type O_4 was considered. It should be mentioned that the vast majority of experiments were carried out with the component C_{11}[18]. The result should therefore be considered in the context of this circumstance. However, it was noticeable that the series of experiments with at least one component of the type O_1 performed significantly better and ranged between 5.8 and 6.0 (i.e., very good results). Different combinations varied between $4.95 \leq \text{Rating} \leq 5.39$. The consideration of possible correlations and the component score distribution was omitted due to the aforementioned bias.

Qualitative (output data explanations). For the qualitative evaluation (cf. last row in Table 1) a total of 54 experiments were selected. Since this qualitative analysis of the explanations served primarily as a trend check for the results of the quantitative evaluation (see above paragraph), we limited the experiments used, resulting in a subset of one-shot explanations of GPT-3.5 and GPT-4 as well as the template explanations. Thus, each experiment type contained 18 experiments. The metric was the same as for the template quality analysis. It was notable that both experiment types exhibit a comparable level of performance, with 3.59 for GPT-3.5 and 3.71 for GPT-4. Therefore, even with only this subset of evaluated experiments, the tendency for good generative-created explanations is amplified.

5. CONCLUSION

In this article[19], we investigated the potential of generating explanations for Question Answering systems which are typically built as a component-based system following the orchestration pattern. These systems integrate processing steps that include AI-driven implementations, namely, natural language processing. For the sake of a generalized solution and as many AI approaches cannot be explained properly, we decided to focus in this article on creating explanations of the observable data flows (input and output) of the components. The created explanations enable experts to receive a stepwise (component-wise) understanding of the processing of concrete component behavior and therefore the whole process. Here we used

[18] Over the duration of the experimentation, other components ceased to yield solutions

[19] The corresponding publication "Towards LLM-generated explanations for Component- based Knowledge Graph Question Answering Systems" received the outstanding paper award at the 23rd WWW/Internet 2024 Conference (ICWI 2024)

Qanary-based KGQA systems (i.e., natural-language question processing) as a case study. In this study, the components data flows are represented as SPARQL queries (I_1, ..., I_6) and grounded RDF triples (O_1, ..., O_4) that constituted the basis for these explanations. Therefore, we initially introduced a concept to represent the explanations for components and their input as well as output streams. In this second step, we demonstrated a template-based approach to generate explanations, which served as a foundation for subsequent experimental considerations (where LLMs were used). The results of these experiments clearly showed that these explanations stay within the former ones. When translating SPARQL queries, we found that LLMs produce almost correct natural-language representations of these queries. However, in our case study, there was no clear improvement regarding different GPT models. Regarding the explanation of the output data, here RDF triples, we focused on the adaptability between the different data types. While our quantitative evaluation showed several different results and problems, such as counting or mismatching, it also showed potential, namely the relevance of predicates. The human expert evaluation, on the other hand, clearly showed at least the same quality in comparison.

The presented approach provides a huge potential as it is not limited to the field of Question Answering. Instead, it provides a feasible method to explain the behavior of (component-based) systems while establishing a semantic layer for the input and output data of each observable component in a system. As this recording can be minimally invasive, there is a wide scope of applicability. Moreover, our experiments have shown that LLMs are well-suited for automatically generating human-readable explanations which are highly valued by the experts who typically need to understand (often growing and changing) systems while aiming for additional features, improved result quality, or must have their system audited. From this, we derive a great potential for the explanation of complex systems by using additional semantics for the data flows.

Further research into the field of explainability in component-based (QA) systems, utilizing a range of datasets or alternative methodologies, could prove beneficial. Additionally, following this first step, it might be worth investigating the possibility of explaining systems and their hierarchical aggregations of components to help researchers and developers understand the behavior of component-based systems.

REFERENCES

Adadi, A. & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). In *IEEE Access*, Vol. 6, pp. 52138-52160.

Anjomshoae, S., Najjar, A., Calvaresi, D. and Främling, K. (2019). Explainable agents and robots: Results from a systematic literature review. *'18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019)', International Foundation for Autonomous Agents and Multiagent Systems*, pp. 1078-1088.

Barredo Arrieta, A. et al. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. In *Information Fusion*, Vol. 58, pp. 82-115.

Both, A. et al. (2016). Qanary – a methodology for vocabulary-driven open question answering systems. *Proceedings of the 13th International Conference on The Semantic Web. Latest Advances and New Domains.* Springer, Vol. 9678, pp. 625-641.

Burkart, N. and Huber, M. F. (2021), A survey on the explainability of supervised machine learning. In *Journal of Artificial Intelligence Research,* Vol. 70, pp. 245-317.

Chazette, L., Brunotte, W. and Speith, T. (2021). *Exploring explainability: A definition, a model, and a knowledge catalogue*. https://doi.org/10.48550/arXiv.2108.03012

Chen, Z. et al. (2020). Few-shot NLG with pre-trained language model. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL, pp. 183-190.

Ell, B., Vrandečić, D. and Simperl, E. (2015). SPARTIQULATION: Verbalizing SPARQL queries. In E. Simperl (ed.) *The Semantic Web: ESWC 2012 Satellite Events, Lecture Notes in Computer Science*, Springer, pp. 117-131.

Gillespie, N., Lockey, S. and Curtis, C. (2021). *Trust in artificial Intelligence: a five country study* (Report). The University of Queensland.

Gilpin, L. H. et al. (2018). Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 80-89.

Haar, L. V., Elvira, T. and Ochoa, O. (2023). An analysis of explainability methods for convolutional neural networks. In *Engineering Applications of Artificial Intelligence*, Vol. 117, 105606.

Hoffman, R. R., Mueller, S. T. and Klein, G. (2017). Explaining explanation, part 2: Empirical foundations. In *IEEE Intelligent Systems*, Vol. 32, No. 4, pp. 78-86.

Lapalme, G. (2020), Rdfjsrealb: a symbolic approach for generating text from rdf triples. *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pp. 144-153.

Li, J. et al. (2024). Pre-trained language models for text generation: A survey. In *ACM Computing Surveys*, Vol. 56, No. 9.

Mellish, C. and Sun, X. (2006). The semantic web as a linguistic resource: Opportunities for natural language generation. In *Knowledge-Based Systems,* Vol. 19, No. 5, pp 298-303.

Moussallem, D., Gnaneshwar, D., Ferreira, T. C. and Ngomo, A.-C. N. (2020). *Nabu − multilingual graph-based neural rdf verbalizer*. https://doi.org/10.48550/arXiv.2009.07728

Moussallem, D., Speck, R. and Ngonga Ngomo, A.-C. (2020). Generating explanations in natural language from knowledge graphs. In *Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges*. IOS Press, Vol. 47, pp. 213-241.

Ngonga Ngomo, A.-C. et al. (2013). Sorry, I don't speak SPARQL: translating SPARQL queries into natural language. *Proceedings of the 22nd International Conference on World Wide Web', WWW '13*. Association for Computing Machinery, New York, NY, USA, pp. 977-988.

Perevalov, A. and Both, A. (2024). Towards LLM-driven natural language generation based on SPARQL queries and RDF knowledge graphs. In *3rd International Workshop on Knowledge Graph Generation From Text (TEXT2KG) Co-located with the Extended Semantic Web Conference (ESWC 2024)*.

Pope, P. E. et al. (2019). Explainability methods for graph convolutional neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10772-10781.

Ribeiro, L. F. R., Schmitt, M., Schütze, H. and Gurevych, I. (2021). Investigating pretrained language models for graph-to-text generation. *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*. Association for Computational Linguistics, Online, pp. 211-227.

Rosenfeld, A. and Richardson, A. (2019). Explainability in human–agent systems. In *Autonomous Agents and Multi-Agent Systems*, Vol. 33, No. 6, pp. 673-705.

Sander, M. (2014). Ontology-based translation of natural language queries to SPARQL. *Natural Language Access to Big Data: Papers from the 2014 AAAI Fall Symposium*.

Schiese, D., Perevalov, A. and Both, A. (2024). Post-hoc insights: Natural-language explanations for AI-enhanced/-integrated software systems. In *20th International Conference on Semantic Systems*.

Shekarpour, S., Nadgeri, A. and Singh, K. (2020). Qa2explanation: Generating and evaluating explanations for question answering systems over knowledge graph. *Proceedings of the First Workshop on Interactive and Executable Semantic Parsing*. Online. Association for Computational Linguistics.

Singh, K., Both, A., Sethupat, A. and Shekarpour, S. (2018a). Frankenstein: A platform enabling reuse of question answering components. In A. Gangemi et al. (ed.) *The Semantic Web. ESWC 2018. Lecture Notes in Computer Science*. Springer, Cham, Vol 10843, pp. 624-638.

Singh, K. et al. (2018b). Why reinvent the wheel: Let's build question answering systems together. *Proceedings of the 2018 World Wide Web Conference, WWW '18*, pp. 1247-1256.

Sun, X. and Mellish, C. (2007). An experiment on "free generation" from single RDF triples. *Proceedings of the Eleventh European Workshop on Natural Language Generation', ENLG '07*, ACL, pp. 105-108.

Yin, X., Gromann, D. and Rudolph, S. (2021). Neural machine translating from natural language to SPARQL. In *Future Generation Computer Systems*, Vol. 117, pp. 510-519.