


## Article

# R2D2: A Dbpedia Chatbot Using Triple-Pattern Like Queries

Haridimos Kondylakis <sup>1,2,\*</sup> , Dimitrios Tsirigotakis <sup>1</sup>, Giorgos Fragkiadakis <sup>1</sup>, Emmanouela Panteri <sup>1</sup>, Alexandros Papadakis <sup>1</sup>, Alexandros Fragkakis <sup>1</sup>, Eleytherios Tzagkarakis <sup>1</sup>, Ioannis Rallis <sup>1</sup>, Zacharias Saridakis <sup>1</sup>, Apostolos Trampas <sup>1</sup>, Giorgos Pirounakis <sup>1</sup> and Nikolaos Papadakis <sup>1</sup>

<sup>1</sup> Department of Electrical and Computer Engineering, Hellenic Mediterranean University, 71004 Heraklion, Greece; tsiridim@yahoo.com (D.T.); atsipas@gmail.com (G.F.); emmanouela@panteri.gr (E.P.); lexpapadakis@gmail.com (A.P.); alexanderfragkakis@gmail.com (A.F.); eltzaster@gmail.com (E.T.); rallis@staff.teicrete.gr (I.R.); haris\_saridakis@hotmail.com (Z.S.); apostolis\_tr@yahoo.gr (A.T.); georgepirounakis@gmail.com (G.P.); npapadak@cs.hmu.gr (N.P.)

<sup>2</sup> Institute of Computer Science, FORTH, N. Plastira 100, 71500 Heraklion, Greece

\* Correspondence: kondylak@ics.forth.gr; Tel.: +30-2810-391449

Received: 21 June 2020; Accepted: 1 September 2020; Published: 3 September 2020



**Abstract:** Chatbots, also known as conversation agents, are programs that are able to simulate and reproduce an intelligent conversation with humans. Although this type of program is not new, the explosion of the available information and the rapid increase of the users seeking this information have renewed the interest in their development. In this paper, we present R2D2, an intelligent chatbot relying on semantic web technologies and offering an intelligent controlled natural language interface for accessing the information available in DBpedia. The chatbot accepts structured input, allowing users to enter triple-pattern like queries, which are answered by the underlying engine. While typing, an auto-complete service guides users on creating the triple patterns, suggesting resources available in the DBpedia. Based on user input (in the form of triple-pattern like queries), the corresponding SPARQL queries are automatically formulated. The queries are submitted to the corresponding DBpedia SPARQL endpoint, and then the result is received by R2D2 and augmented with maps and visuals and eventually presented to the user. The usability evaluation performed shows the advantages of our solution and its usefulness.

**Keywords:** chatbots; DBpedia; conversation agents

## 1. Introduction

During the last years, chatbots have played a prominent role as human–computer interfaces. Chatbots are computer programs that are capable of establishing a conversation with users, with either written messages or vocal commands. Such programs provide easy and pleasant conversations and answers; they can automatically manage large volumes of users, replacing human operators, which are not always available [1]. The first chatbot was ELIZA [2], trying to simulate a psychotherapist, whereas many chatbots are currently promoted and used by commercial vendors such as Apple’s Siri, Amazon Alexa, Microsoft Cortana, and Google Assistant [3], whereas it is foreseen that between 2017 and 2023, the spread of chatbots will be increased up to 37% [1].

In addition, although a vast amount of information is now available online, still, users find it difficult to identify the information of interest and navigate to the numerous and large information sources [4,5]. DBpedia [6] is a crowd-sourced community effort, including 38.4 million resources in multiple languages, trying to extract structured content from the information already available in various

Wikimedia projects. Data are available as Linked Open Data, and one can navigate through the available information through standard web browsers, automated crawlers, recommendation engines [7,8], or pose complex SPARQL queries. Although the information there is highly interconnected, trying to retrieve answers through browsing events to trivial queries for the interconnected information is really difficult. On the other hand, SPARQL endpoints are a no-go territory for the average web user.

On the other hand, traditional question answering systems enable users to issue queries in natural text, which is then usually transformed internally to a query over the knowledge graph, and the results are returned to the users. However, despite the progress in those systems, still, many of the user questions remain unanswered, and even simple questions in many cases return erroneous results. Toward this direction, in this paper, we present R2D2, a novel chatbot that enables real-time conversation with end users that is able to answer complex queries using an intuitive graphical user interface. The system trades expressivity for correctness, meaning that it imposes a triple-pattern like controlled language to the end users, heavily supported by auto-completion; however, it guarantees that the correct information will be retrieved from the knowledge base.

More specifically, the contributions of this work are the following:

We implement R2D2, an interactive chatbot answering complex queries using an intuitive graphical user interface, employing interactive cards, clickable images, and maps to enhance user experience.

- The chatbot accepts queries in a structured yet naïve triple-like language with the help of an auto-completion service that helps the users structure their query and explore the available information and in real time.
- The query is automatically transformed into the corresponding SPARQL query and sent to the DBpedia SPARQL endpoint to be further answered.
- Then, another module receives the results and formulates the answer to the end-user using nice cards, graphs, and maps.

Exploiting our approach for a different knowledge base only requires the availability of a SPARQL endpoint and an update in the auto-complete mechanism. As such, it has the potential to be linked to other knowledge bases as well. To the best of our knowledge, there is not currently available an alternative simple yet powerful platform enabling access to SPARQL endpoints through an intelligent chatbot offering a controlled natural language interface.

The remainder of this paper is structured as follows. In Section 2, we present related work. Then, in Section 3, we present the architecture of the proposed system and describe the various components implemented. Section 4 presents our user-based evaluation, whereas Section 5 concludes this paper and presents directions for future work.

## 2. Related Work

As chatbots are supposed to provide answers to user queries, developing query-answering systems has been the target of several research works [9]. For an overview in the area, the interested reader is forwarded to [10].

As there have been several chatbots already developed, we could distinguish them into two categories: the retrieval-based and the generative-model ones [11]. The retrieval-based chatbots come with a set of written responses to minimize grammatical errors, improve coherence, and avoid situations where the system can be hacked to respond in less appropriate ways. Retrieval-based chatbots are best suited for closed-domain systems, which are built to specifically solve simple recurring problems. However, the set of data available usually cannot foresee responses for all possible scenarios. Trying to mitigate the aforementioned problem, the generative-model conversational agents are more intelligent but more difficult to implement than retrieval-based chat bots. With generative chatbots, a program without any active human assistance generates responses. Generative-model chatbots best suit open-domain systems that are not constrained to a particular type of conversation domain and they can easily be adapted or trained to be used in diverse domains. R2D2 is an open domain system

limited only by the knowledge available in the linked SPARQL endpoint. As there have been many approaches falling in these two categories, we only focus on the ones using semantic technologies.

Among these approaches, we can further distinguish ones that exploit templates and ones that exploit semantic-like search algorithms. One system in the former category is proposed by Unger et al. [12] for RDF data. The authors transform a method that parses user questions to produce a SPARQL template that directly mirrors the internal structure of the question. Other template-based approaches employ graph learning to map user queries to SPARQL queries [13]. In the second category, we can find approaches such as those of [14,15] that focus on semantic search-like algorithms, performing entity linking on properties and entities and eventually retrieving the top k-ranked entities representing the intention behind a natural language query. However, in our case, we do not match the input queries with template queries but directly transform them into fully fledged queries that can be directly issued to the triple store. Of course this is possible, since the user input is really close to triple patterns.

In another approach by Vegesna et al. [16], the authors propose an ontology-based chatbot for the e-commerce domain. Available e-commerce APIs (e.g., eBay API) are used for retrieving external data, whereas ontology templates are used to retrieve data from the data source using Jape rules. When data have been extracted, users can ask questions which are parsed using NLP techniques and the results are being returned to the users. However, in R2D2 no time-consuming data transformation is happening, whereas the agent can also provide figures and maps to better inform the end user.

Another approach based on semantics is proposed within the research project FRASI [17] trying to develop a question-answering system supporting consumers to find information regarding products of interest. The choice was to implement a Question-Answering (QA) system using a pattern-matching chatbot technology. However, their design required the definition of all possible interactions with the user, leading to a cumbersome Knowledge Base (KB) and a time-consuming development process. A common sense ontology was used trying to recognize the kind of sentence and to consequently reduce it to a simpler structure. However, in our case, we do not rely on recording all possible user interactions, nor do we try to reduce input to simpler structures.

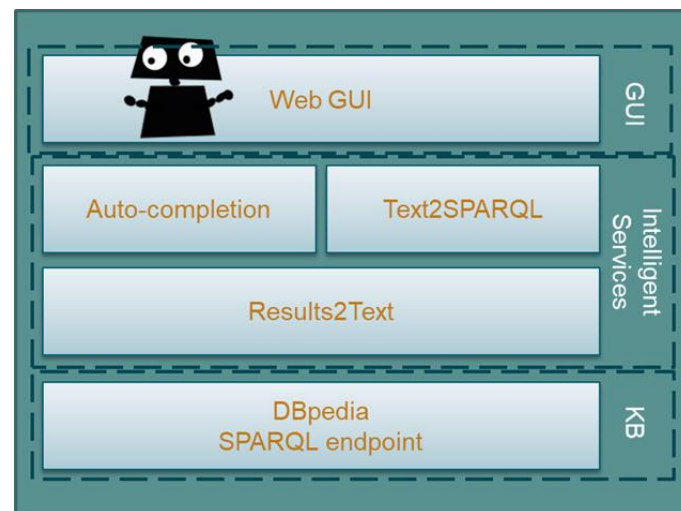
In addition, there are works such as those that focus solely on the generation of textual descriptions of sets of triples [18,19], which generate textual summaries from knowledge base triples [20] or exploit neural networks to generate explanations in natural language from knowledge graphs [21]. However, although relevant to one component of our work (the one transforming the results into text), those approaches could be used off the shelf to our purpose, and we target to a more generic solution that includes also services such as auto-completion of user query, graphical visualization of the results, chatbot functionality, etc.

Finally, we have to mention that there are approaches already that are specifically designed for DBpedia such as the DBNQA dataset [22] that provides a comprehensive dataset of questions and SPARQL queries. However, in our case, the construction of the corresponding SPARQL queries is trivial from the user input. In addition, there is already a chatbot available for the DBpedia [23]. The bot is capable of responding to users in the form of simple short text messages or through more elaborate interactive messages. Users can communicate or respond to the bot through text and through interactions (such as clicking on buttons/links). However, in order for the engine to work, the intent of the questions should be appropriately classified and used for query answering, with many rules being defined and exploited by the system. Our approach moves to the same direction as the existing DBpedia chatbot using interactive cards, clickable images, and maps. However, our system can work as is, without requiring any additional preprocessing, cleaning, or rule definition, which makes it really simple to be exploited and used.

### 3. Architecture

The architecture of the R2D2 chatbot is composed of three layers as shown in Figure 1: (a) the graphical user interface (GUI), (b) the intelligent services including a recommendation service,

a text-to-SPARQL service, and a results-to-text and images service, and finally, (c) the DBpedia knowledge base. In the following, we will analyze each one of those components in detail.



**Figure 1.** The high-level architecture of the R2D2 chatbot.

### 3.1. The Graphical User Interface

The graphical user interface has been developed using HTML, CSS, and JavaScript. In addition, Bootstrap was used, which is a free and open-source front-end framework for enabling a responsive website. The generated website includes relevant information and the main chatbot where the users can interactively ask for information from the DBpedia. A screenshot of the chatbot is shown in Figure 2. The R2D2 chatbot accepts queries in a structured language and consists of one or more of triple patterns of the following form:

$$\textit{Entity Property Value} \quad (1)$$

where *Entity* is a resource from the DBpedia knowledge base linked with a *Property* to some *Value*. Entities, properties, and values might be variables, whereas many triple patterns can be used for a single query. As a running example, consider the following query asking for Albert Einstein's birth date and birthplace.

$$\textit{Albert\_Einstein birth\_date? x Albert\_Einstein birth\_place?y.} \quad (2)$$

As soon as the user starts typing, the system suggests possible auto-completions from the knowledge already available to the DBpedia KB. As soon the entity part is typed, then the available connections to that entity are also proposed to the user, making really trivial the formulation of the whole query—without requiring any knowledge about triple patterns and structured query languages (it only needs 5 min explanation to the untrained end users, as our evaluation results showed; refer to Section 4).

Then, the query is issued to the system, and the result is visualized back to the end user, which is usually accompanied by a related image, map, or information card. We have to mention that we have added trivial answers to trivial user questions (e.g., who am I, hello, good morning, etc.) in order to reply to those questions without even going to the knowledge base.

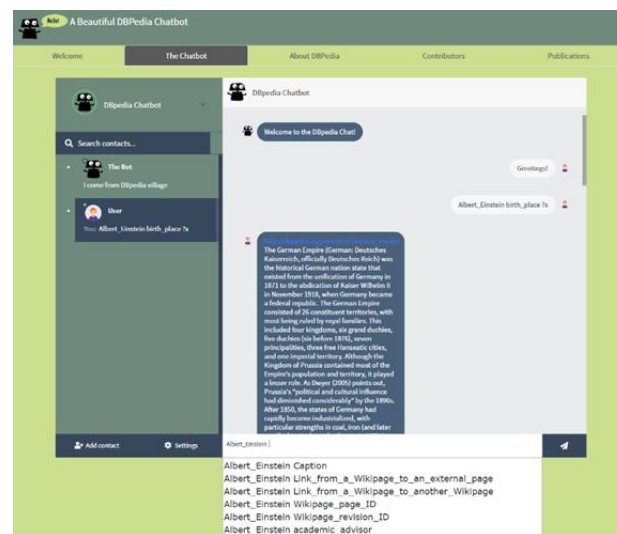


Figure 2. A screenshot of the graphical user interface (GUI).

### 3.2. Query Completion

As already mentioned, a service is constantly used to auto-complete end-user input and to guide query formulation. When the user starts typing in the input field, as soon as more than three characters are typed, the plugin starts to search for entries that match and displays a list of entries to choose from. As soon as the first part of the triple pattern is retrieved (i.e., the entity part), the same service is called to retrieve properties that are linked with the entities that might have been inserted in the entity part of the triple pattern. The properties are retrieved and presented to the user as a drop-down list, as well, allowing him to select the one of interest. If a variable is inserted at the entity part of the triple pattern, then all properties are retrieved matching the first three characters that the user has entered. In any case, the user can ignore the recommendations and continue writing; the auto-completion suggestions are constantly updated as more characters are typed. However, in case of incomplete labels, the query will return no answers.

This is implemented using a JQuery autocomplete widget. For searching, the DBpedia lookup service has been used and extended. The available service is used to lookup DBpedia URIs using as input related keywords. Those keywords can either be the label of a resource or frequently used anchor text in Wikipedia. The results are ranked by the number of links from other Wikipedia pages and automatically sorted and presented to the user as s/he types. The implemented service offers a web API for prefix search and for given partial keyword, it returns the URIs of the related DBpedia resources. All data can be returned in either XML or JSON.

For example, for DBpedia, the following SPARQL query is issued to the corresponding SPARQL endpoint to get the available labels of the various properties that are used to link an already typed <ENTITY> in the query with other information:

```
SELECT DISTINCT ?label
WHERE {
  <ENTITY> ?p ?o.
  ?p <http://www.w3.org/2000/01/rdf-schema#label> ?label.
  FILTER(LANG(?label) = "" || LANGMATCHES(LANG(?label), "en"))
}
```

The *FILTER* for the language is used as the current implementation is focusing only in the English language; nevertheless, this is quite easy to be updated for other languages as well. Then, the output is parsed, and the list of properties is formulated and presented to the user. All the properties, which consist of more than one word, are converted to snake\_case format, enabling a better segmentation of the three parts of the triple patterns.

We have to note that this auto-completion service can be directly connected to other SPARQL endpoints as well, but we foresee that minor adjustments might be needed according to the peculiarities of the individual data sources. However, this configuration should be implemented only once per SPARQL endpoint.

### 3.3. Text2SPARQL Service

This service receives a query string as formulated by the end user and constructs the SPARQL query to be issued to the SPARQL endpoint for retrieving the information of interest. Before formulating the SPARQL query, the web service checks if there is a malformed user input and returns appropriate messages to the end user. Then, it first adds the prefixes of the SPARQL query and identifies the triple patterns in the query string. The variables in the triple patterns are added in the select clause of the SPARQL query, and the necessary triple patterns for retrieving the related images, the primary topics, and the correct language are added. In our running example, Equation (2) is transformed to the following SPARQL query:

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE {
    dbr:Albert_Einstein dbo:birth_date ?x.
    dbr:Albert_Einstein dbo:birth_place ?y.
    OPTIONAL {
        ?x dbo:abstract ?x1;
        foaf:isPrimaryTopicOf ?x2;
        foaf:depiction ?x3
    }.
    FILTER (lang(?x) = " || lang(?x) = 'en')
    OPTIONAL {
        ?y dbo:abstract ?y1;
        foaf:isPrimaryTopicOf ?y2;
        foaf:depiction ?y3
    }.
    FILTER (lang(?y) = " || lang(?y) = 'en')
}
```

### 3.4. Results2Text

This service retrieves as input (a) the user query and (b) the SPARQL query formulated by the Text2SPARQL and returns the results to be visualized by the web GUI.

There are two modes in the result formulation: (a) natural language answers and (b) references to card boxes, maps, and figures.

Using the first option, the triple patterns are seen as sentences that can be reused to answer user's queries trivially. Although there are many works for optimizing the presentation of the results (see [18] for an example where the authors use machine learning for sentence creation out of triples), we leave this as future work and we present a simple yet effective algorithm. This algorithm that is used for transforming the triple answers into text is presented in Algorithm 1. As shown, first, the SPARQL query is forwarded to the Virtuoso triple store for execution. The results are retrieved, and the mapping of the variables in the user triple patterns and the retrieved results is recorded. Then, based on this mapping between the user triple patterns and the returned results, for each user, triple pattern, and returned result, the corresponding sentence is added to the text that will be presented to



the user (lines 2–4). When all individual sentences have been constructed, the final result is returned to the user.

---

**Algorithm 1.** Results2Text.
 

---

**Input:** TQ: The user triple patterns, Q: The SPARQL query

**Output:** result\_text: The text to be presented to the user.

1. result:=execute(Q)
  2. result\_text:=""
  2. for each (entity property value) in TQ
  3. for each binding of the (entity property value) in the results
  4. result\_text:=result\_text + entity + ' ' + property + ' is ' + value ' '.
  - 5.
  6. return result\_text
- 

For example, the triple pattern “Albert\_Einstein birth\_date ?x.” and the result of the query “14 March 1879” can be combine to return the following sentence:

*Albert Einstein birth date is 14 March 1879*

Using the second option, the corresponding summary is retrieved by DBpedia and presented as a card to the user, visualizing also interesting images or maps. Although a combination of those might be seen as useful, preliminary experiments showed that users preferred the card/image/map approach only, and we configured the chatbot to use the second option.

We have to note that besides star-shaped queries, queries of arbitrary shape are also possible, which are similar to the way SPARQL queries are being formulated. For example, searching Barack Obama’s spouse’s religion would require the formulation of the following query:

*Barack\_Obama wife ?x. ?x religion ?y.*

The result on the aforementioned question would be:

*Barack Obama’s wife is Michelle Obama. Michele Obama’s religion is United Methodist*

#### 4. Evaluation

For evaluating the chatbot, we recruited 70 students of the Department of Electrical and Computer Engineering at the Hellenic Mediterranean University. The requirements for using the system are minimal, and only 5 min were adequate to explain how the chatbot works to the untrained students. After a short demonstration of the system, we asked them to use the system for 5 min searching for information through R2D2. We instructed them to perform at will at least 3 queries with at least one of them including more than one triple pattern. Although we did not recorded individual user queries, we provide online a set of examples provided to the students to better understand query formulation (note that the last three of those queries are the three example queries proposed from the DBpedia chatbot). As most of the students were already familiar with chatbots such as Apple Siri and Cortana, they quickly grasped the idea behind the chatbot and were able to use it effectively.

Then, we asked them to fill in a questionnaire (Appendix A). The questionnaire was constructed using the quality features from the product quality model of the ISO/IEC 25.000 series [24] together with the System Usability Scale (SUS) [25], which we transformed to simple questions. The results are reported in Table 1 for the various evaluation categories.

**Table 1.** The results for the various evaluation categories.

Functionality	Suitability	4.18	4.15
	Accurateness	4.09	
	Compliance	4.18	
Efficiency	Time Behavior	4.2	4.25
	Resource Utilization	4.3	
Compatibility	Co-Existence	4.09	4.27
	Interoperability	4.45	
Usability	Understandability	4.36	4.33
	Learnability	4.54	
	Operability	4	
	Attractiveness	4.45	
Reliability	Fault tolerance	3.4	3.47
	Recoverability	3.54	
Maintainability	Analyzability	3.7	3.94
	Changeability	4	
	Stability	3.9	
	Testability	4.18	
Portability	Adaptability	4.18	4.04
	Installability	3.90	
	Conformance	3.94	
	Replaceability	4.14	
Quality of Use	Effectiveness	4	3.96
	Efficiency	4	
	Satisfaction	3.90	
SUS	78/100 (Kappa 0.8)		

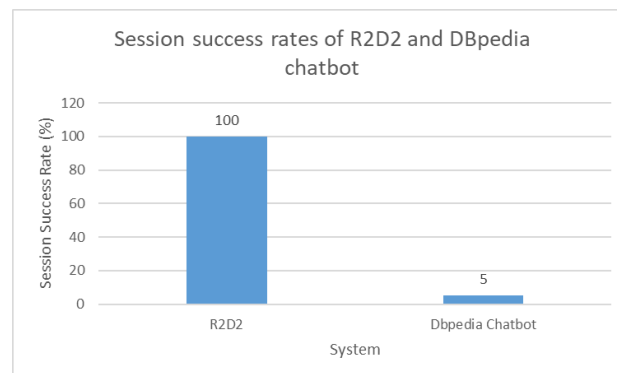
Values greater than three represent the high level of the specific software feature; values between 2.5 and 3 are at low risk, whereas values below 2.5 are considered high risk. In our case, all were graded with a satisfactory average above 3.47, showing the high quality of the software. The lowest average score was for the reliability (3.7/5), since in some cases, errors appeared (e.g., images not shown) without allowing a proper recovery or adequate explanation. Reliability of the system is also affected by the external services that are used (directly from DBpedia), as in several cases, the services were not functioning—however, not during the user evaluation.

In addition, as the different services have been developed using many different technologies, users identified as a relatively hard task the maintainability of the system (average maintainability 3.94/5). The questions regarding the quality of use also score relatively high, demonstrating again the quality of the system. The evaluation provided us with valuable feedback on the current state and was used to correct the reported errors. The highest average score for efficiency (4.25/5) shows that the system can quickly respond to the queries issued and uninterruptedly provide answers from such a large source of information. In essence, the execution time required for query answering is the time required to return the result through DBpedia SPARQL endpoint, with only milliseconds spent in the trivial SPARQL formulation and result presentation.

Furthermore, the SUS usability score was 78 on a scale of 0 to 100, whereas the inter-rater agreement was rather good (kappa 0.8). The average SUS score from published studies has been measured by Sauro et al. [26] as 62.1, but 68 is often used as a gold standard. Therefore, SUS scores exceeding 68 are considered above average, while SUS scores below 68 are below average. Our SUS score of 79.18 exceeds by far the reference point of 68, yet future improvements can be made to provide even higher levels of reliability, perhaps by installing a triple store within the R2D2 so that the whole system does not have to rely on external SPARQL endpoints.



Finally, we used the 20 examples that we provided to the students for comparing R2D2 with the DBpedia chatbot [23]. We have to note that three of those examples were proposed in the corresponding DBpedia chatbot site. In addition, as the DBpedia chatbot accepts textual input and R2D2 accepts triple-pattern like input, in the online Excel sheet, we provide both the textual and the triple-pattern like queries. We measured the session success rate of using the two systems and the results are shown in Figure 3: the session success rate measures the ratio of user sessions that lead to a success.



**Figure 3.** Session success rates of R2D2 and DBpedia chatbot.

As shown, all queries were able to be appropriately formulated and answered using the R2D2 system, whereas for the DBpedia chatbot, only one query returned an actual answer (despite the fact that three of the queries were examples presented in the DBpedia chatbot site). This corresponds to only 5% of the cases in our evaluation.

Overall, what we observe is that 5 min were enough for the untrained students to use the chatbot, enabling students to formulate simple triple-like queries and effectively returning the corresponding answers. As triple pattern formulation is guided by an auto-complete service, the engine guarantees that if there is a corresponding result available in DBpedia, the information will always be retrieved and presented to the user. This is not true for the DBpedia chatbot as previously shown, which struggles to answer textual user input.

## 5. Discussion and Conclusions

This paper presents a novel chatbot that is able to respond to user queries regarding information stored in the DBpedia. User input is received in the form of triple-pattern like queries. The chatbot accepts structured input and guides users on query formulation. Then, those queries are transformed to SPARQL queries and are sent to the DBpedia SPARQL endpoint to be answered. The answers are collected by another service, which enriches the results with natural text, graphical images, maps, and cards, and it sends the corresponding information to the GUI to be finally visualized and presented to the user. Our system differs from a traditional question-answering system in the fact that it employs a cleaner, user-friendly interface, it provides multimedia answers, and it also provides answers to trivial queries without even going to the DBpedia. The user evaluation performed shows the high usability of our solution and the advantages gained when R2D2 is used for searching information available through DBpedia. An advantage of the system is that it requires no complex, costly mapping of natural language input to templates, SPARQL queries, or parts of the graph, as the triples patterns are provided as input by the user through a triple-pattern like language. The simplicity of the triple-pattern like queries makes the system usable by non-trained experts, trading expressivity with usability.

**Limitations.** As SPARQL cannot handle queries with missing label parts, to mitigate partially provided labels by the users, the auto-complete service is used/enforced. As such, the end user has to select and complete the missing part of each label. However, a more natural approach would be to allow partially provided labels and do a search and importance ranking (e.g., [27]) for automatically solving the problem without relying on user input. In addition, another problem is that when the

returned results are many, then the text returned by our algorithms can have a considerable length with obvious information repetition. Further, solution modifiers (for example queries including how long, how many, average, count, etc.) performing calculation over the retrieved data are not currently supported. Although SPARQL queries are supporting this functionality, more research is required on ways that the user would prefer to optimally formulate these requests in our triple-pattern like queries. Finally, in several cases, we experienced unavailability of the external services used from DBpedia, which impacts the reliability of the system.

**Future Work.** Our immediate plans include extending the functionality of the chatbot for tackling the aforementioned limitations. We are already in the process of installing locally the services of the DBpedia so as not to rely in external services. However, as ontologies and KBs are constantly evolving [28,29], this might lead to access to outdated information, trading effectiveness for the always more recent information. A solution there would be to rely on the online DBpedia services and only in case of failures to use locally deployed services. In addition, we intend to extend our approach to include non-expected text and by asking questions for refining the result (for example, when multiple results are available). To this direction, a memory will be added to the chatbot being able to expand over previous results, not repeating the same information and providing more information about a certain entity based on user feedback.

**Author Contributions:** Conceptualization, H.K.; Methodology, H.K. and N.P.; Software, D.T., G.F., E.P., A.P., A.F., E.T., I.R., Z.S., A.T. and G.P.; Supervision, H.K.; Writing—original draft, H.K., D.T., G.F., E.P., A.P., A.F., E.T., I.R., Z.S., A.T., G.P. and N.P.; Writing—review & editing, H.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** This work has been implemented as a project of the course “Semantic Web” at the Department of Electrical and Computer Engineering, Hellenic Mediterranean University. This research received no external funding.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A Questionnaire Answered by the Users

Functionality	Suitability	Can software perform the tasks required? <i>Explanation: Were the users able to perform queries easily for the information of interest?</i>
	Accurateness	Is the result as expected? <i>Explanation: Do the results provided have appropriate information content?</i>
	Compliance	Is the system compliant with standards? <i>Explanation: Is the system compliant with the standards you have in mind for the web (e.g., HTTP/CSS/HTML/RDF)?</i>
Efficiency	Time Behavior	How quickly does the system respond? <i>Explanation: Do you receive answers in reasonable time?</i>
	Resource Utilization	Does the system utilize resources efficiently? <i>Explanation: Does the system have too high computational resources to be used by end users?</i>
Compatibility	Co-existence	Can the system share resources without loss of its functionality? <i>Explanation: Based on the high-level architecture presented, do you believe that the system can be run in parallel with other services, provided by the server?</i>
	Interoperability	Can the system share information/data with other components? <i>Explanation: Is it able to communicate and exchange information with other components?</i>
Usability	Understandability	Does the user comprehend how to use the system easily? <i>Explanation: Is it easy to understand how to use the system?</i>
	Learnability	Can the user learn to use the system easily? <i>Explanation: Is it easy to learn how to use the system?</i>
	Operability	Can the user use the system without much effort? <i>Explanation: Is it easy to use the system without devoting much effort?</i>
	Attractiveness	Does the interface look good? <i>Explanation: Are you satisfied with the graphical user interface?</i>
Reliability	Fault tolerance	Is the software capable of handling errors? <i>Explanation: Have you experienced errors from the system? Was it easy for the system to handle them?</i>
	Recoverability	Can the software resume working & restore lost data after failure? <i>Explanation: Have you experienced errors from the system? Was it easy for the system to resume working?</i>

Maintainability	Analyzability	Can faults be easily diagnosed? <i>Explanation: If you experienced strange behavior by the system, was it easy to understand why?</i>
	Changeability	Can the software be easily modified? <i>Explanation: Based on the high-level presentation of the architecture, do you believe that the system can be easily modified?</i>
	Stability	Can the software continue functioning if changes are made? <i>Explanation: Based on the high-level presentation of the architecture, do you believe that some services can be changed transparently to the user?</i>
	Testability	Can the software be tested easily? <i>Explanation: Can you easily test the system?</i>
Portability	Adaptability	Can the software be moved to other environments? <i>Explanation: Based on the high-level presentation of the architecture, do you believe that the software can be easily migrated to other environments?</i>
	Installability	Can the software be installed easily? <i>Explanation: Based on the high-level presentation of the architecture, do you believe that the software can be easily installed?</i>
	Conformance	Does the software comply with portability standards? <i>Explanation: Based on the high-level presentation of the architecture, do you believe that the software complies with portability standards?</i>
	Replaceability	Can the software easily replace other software? <i>Explanation: Do you believe that the software can be used as a service for other systems to call it?</i>
Quality of Use	Effectiveness	How accurate and complete is the software for the intended use? <i>Explanation: Based on the suggested scenarios described, do you believe the software is complete for its intended use?</i>
	Efficiency	Does the software improve the time or reduce resources for the intended goal? <i>Explanation: Based on the suggested scenarios described, do you believe that the system help you save time and resources for getting the desired information?</i>
	Satisfaction	Does the software satisfy the perceived achievements of pragmatic goals? <i>Explanation: Do you believe that the target of the system is pragmatic and realistic?</i>
SUS	1. I think that I would like to use this system frequently. 2. I found the system unnecessarily complex. 3. I thought the system was easy to use. 4. I think that I would need the support of a technical person to be able to use this system. 5. I found the various functions in this system were well integrated. 6. I thought there was too much inconsistency in this system. 7. I would imagine that most people would learn to use this system very quickly. 8. I found the system very cumbersome to use. 9. I felt very confident using the system. 10. I needed to learn a lot of things before I could get going with this system.	

## References

1. Valtolina, S.; Barricelli, B.R.; Gaetano, S.D.; Diliberto, P. Chatbots and Conversational Interfaces: Three Domains of Use. In Proceedings of the Fifth International Workshop on Cultures of Participation in the Digital Age—CoPDA, Castiglione della Pescaia, Italy, 29 May 2018; pp. 62–70.
2. Weizenbaum, J. ELIZA—A computer program for the study of natural language communication between man and machine. *Commun. ACM* **1996**, *9*, 36–45. [\[CrossRef\]](#)
3. Reis, A.; Paulino, D.; Paredes, H.; Barroso, J. Using Intelligent Personal Assistants to Strengthen the Elderlies' Social Bonds—A Preliminary Evaluation of Amazon Alexa, Google Assistant, Microsoft Cortana, and Apple Siri. In Proceedings of the 2018 2nd International Conference on Technology and Innovation in Sports, Health and Wellbeing (TISHW), Thessaloniki, Greece, 20–22 June 2018; pp. 1–5.
4. Troullinou, G.; Kondylakis, H.; Stefanidis, K.; Plexousakis, D. Exploring RDFS kbs using summaries. In Proceedings of the International Semantic Web Conference, Monterey, CA, USA, 8–12 October 2018.
5. Cebiric, S.; Goasdoué, F.; Kondylakis, H.; Kotzinos, D.; Manolescu, I.; Troullinou, G.; Zneika, M. Summarizing Semantic Graphs: A Survey. *Vldb J.* **2019**, *28*, 295–327. [\[CrossRef\]](#)
6. Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P.N.; Hellmann, S.; Morsey, M.; van Kleef, P.; Auer, S.; et al. DBpedia—A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Soc. Work* **2015**, *6*, 167–195. [\[CrossRef\]](#)

7. Stratigi, M.; Kondylakis, H.; Stefanidis, K. Fairness in Group Recommendations in the Health Domain. In Proceedings of the International Conference on Data Engineering, San Diego, CA, USA, 19–22 April 2017; pp. 1481–1488.
8. Stratigi, M.; Kondylakis, H.; Stefanidis, K. Multidimensional Group Recommendations in the Health Domain. *Algorithms* **2020**, *13*, 54. [\[CrossRef\]](#)
9. Marakakis, E.; Kondylakis, H.; Papakonstantinou, A. APANTISIS: A Greek Question-Answering System for Knowledge-Base Exploration. In *Strategic Innovative Marketing*; Springer: Cham, Switzerland, 2017; pp. 501–510.
10. Diefenbach, D.; Lopez, V.; Singh, K.; Maret, P. Core techniques of question answering systems over knowledge bases: A survey. *Knowl. Inf. Syst.* **2018**, *55*, 529–569. [\[CrossRef\]](#)
11. Mutiwokuziva, M.T.; Chanda, M.W.; Kadebu, P.; Mukwazvure, A.; Gotor, T.T. A Neural-network based Chat Bot. In Proceedings of the International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 19–20 October 2017.
12. Unger, C.; Bühmann, L.; Lehmann, J.; Ngonga Ngomo, A.C.; Gerber, D.; Cimiano, P. Template-based question answering over RDF data. In Proceedings of the 21st International Conference on World Wide Web, Lyon, France, 16–20 April 2012; pp. 639–648.
13. Soru, T.; Marx, E.; Valdestilhas, A.; Esteves, D.; Moussallem, D.; Publio, G. Neural Machine Translation for Query Construction and Composition. *arXiv* **2018**, arXiv:1806.10478.
14. Marx, E.; Höffner, K.; Shekarpour, S.; Ngomo, A.C.N.; Lehmann, J.; Auer, S. Exploring Term Networks for Semantic Search over RDF Knowledge Graphs. In Proceedings of the Metadata and Semantics Research, MTSR 2016, Communications in Computer and Information Science, Göttingen, Germany, 22–25 November 2016; Volume 672.
15. Dubey, M.; Banerjee, D.; Chaudhuri, D.; Lehmann, J. EARL: Joint Entity and Relation Linking for Question Answering over Knowledge Graphs. In Proceedings of the International Semantic Web Conference, Monterey, CA, USA, 8–12 October 2018.
16. Vegesna, A.; Jain, P.; Porwal, D. Ontology based Chatbot (For E-commerce Website). *Int. J. Comput. Appl.* **2018**, *179*, 51–55. [\[CrossRef\]](#)
17. Augello, A.; Pilato, G.; Machi, A.; Gaglio, S. An Approach to Enhance Chatbot Semantic Power and Maintainability: Experiences within the FRASI Project. In Proceedings of the IEEE Sixth International Conference on Semantic Computing, Palermo, Italy, 19–21 September 2012; pp. 186–193.
18. Zhu, Y.; Wan, J.; Zhou, Z.; Chen, L.; Qiu, L.; Zhang, W.; Jiang, X.; Yu, Y. Triple-to-Text: Converting RDF Triples into High-Quality Natural Languages via Optimizing an Inverse KL Divergence. In Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, Paris, France, 21–25 July 2019; pp. 455–464.
19. Li, Z.; Lin, Z.; Ding, N.; Zheng, H.T.; Shen, Y. Triple-to-Text Generation with an Anchor-to-Prototype Framework. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, Yokohama, Japan, 11–17 July 2020; pp. 3780–3786.
20. Vougiouklis, P.; Elshahar, H.; Kaffee, L.; Gravier, C.; Laforest, F.; Hare, J.S.; Simperl, E. Neural Wikipedian: Generating Textual Summaries from Knowledge Base Triples. *J. Web Semant.* **2018**, *52*, 1–15. [\[CrossRef\]](#)
21. Moussallem, D.; Speck, R.; Ngonga Ngomo, A.C. Generating Explanations in Natural Language from Knowledge Graphs. In *Knowledge Graphs for eXplainable Artificial Intelligence*; IOS Press: Amsterdam, NH, USA, 2020; pp. 213–241.
22. Hartmann, A.K.; Tommaso, M.E.; Moussallem, D.; Publio, G.; Valdestilhas, A.; Esteves, D.; Neto, C.B. Generating a large dataset for neural question answering over the dbpedia knowledge base. In Proceedings of the Workshop on Linked Data Management, WEBDB, Vienna, Austria, 17–18 April 2018.
23. Athreya, R.G.; Ngomo, A.C.N.; Usbeck, R. Enhancing Community Interactions with Data-Driven Chatbots-The DBpedia Chatbot. In Proceedings of the WWW' 18: The Web Conference 2018, Lyon, France, 23–27 April 2018; pp. 143–146.
24. ISO/IEC DIS 25023. *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Measurement of System and Software Product Quality*; ISO: Geneva, Switzerland, 2016.
25. Brooke, J. SUS-A quick and dirty usability scale. In *Usability Evaluation in Industry*; CRC Press: Boca Raton, FL, USA, 1996; Volume 189, pp. 4–7.

26. Sauro, J.; Lewis, J.R. Correlations among prototypical usability metrics: Evidence for the construct of usability. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Boston, MA, USA, 4–9 April 2009; pp. 1609–1618.
27. Pappas, A.; Troullinou, G.; Roussakis, G.; Kondylakis, H.; Plexousakis, D. Exploring Importance Measures for Summarizing RDF/S KBs. In Proceedings of the European Semantic Web Conference, Portorož, Slovenia, 28 May–1 June 2017; pp. 387–403.
28. Kondylakis, H.; Plexousakis, D. Ontology evolution: Assisting query migration. In Proceedings of the International Conference on Conceptual Modeling, Florence, Italy, 15–18 October 2012; pp. 331–344.
29. Kondylakis, H.; Plexousakis, D. Ontology evolution in data integration: Query rewriting to the rescue. In Proceedings of the International Conference on Conceptual Modeling, Brussels, Belgium, 31 October–3 November 2011; pp. 393–401.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).