Quest for Simplified Answers over Natural Language Queries

João Pedro V. Pinheiro - 1912709
Advisor: Marco A. Casanova

Department of Informatics, PUC-Rio, 22453-900 Rio de Janeiro, Brazil
{jpinheiro,casanova}@inf.puc-rio.br

Abstract. Question Answering (QA) systems combine techniques from multiple fields of computer science, among them: Natural Language Processing (NLP), Information Retrieval, Machine Learning (ML) and Semantic Web. These systems may be divided into two parts: Question, receives an user's input in natural language, transform it into a query and search in the knowledge base, and Answer, displays consistent results in understandable format to the user. The main focus of this work is to analyse the original answers and apply summarization heuristics. The original answer may be returned, or a new question is formulated to user in order to accurate results. The work assumes that the knowledge base is expressed in RDF.

Keywords: Question Answering (QA) · Natural Language Question (NLQ) · Summarization · Aggregation · RDF · Semantic Web.

1 Introduction

Question Answering (QA) systems combine techniques from multiple fields of computer science. In this work we focus on the quality of query answers using Information Retrieval and Semantic Web techniques. We analyse the set of triples returned from a query over a Knowledge Base (KB) and apply some predefined heuristics. These heuristics will help the system decide if the answer is ready to be displayed to the user, or if the answer must be improved.

Imagine yourself as user interacting with a voice virtual assistant and you ask an open-ended question about a specific subject, e.g. *What are the restaurants close to PUC-Rio?*. The query answer may have a long list of restaurants as showed in Table (1). Instead of listing the results, the virtual assistant is able to formulate questions to the user based on the prior result set. Still in this scenario, the following questions can be elaborated: *Do you prefer restaurants inside PUC-Rio, until 600m away, or far away?*, *Do you prefer Brazilian, Japanese, or Miscellaneous food?*, and *From one to five, how much do you want to spend?*.

In short, heuristics allow the QA system to decide which fields returned in the query answer are interesting to apply aggregations (**group by** operation). Also it defines a break point (threshold) to stop further attempts on improving results.

The rest of this paper is organized as follows. Section 2 highlights the key challenges. Section 3 describes technologies used. Section 4 presents software documentation with UML diagrams. Section 5 displays the followed schedule. Section 6 displays the results. Section 7 concludes our work and suggests future improvements.

Name	Address	Category	Rating	Price	Distance
Bandejão PUC	R. Marquês de São Vicente, 225	Miscellaneous	3.5	$	0.0m
Couve Flor	R. Marquês de São Vicente, 293	Misc. Buffet	4.3	$$$	0.0m
Villa 90 Gastro.	R. Marquês de São Vicente, 90	Brazilian Buffet	4.2	$$	500m
Casa da Táta	R. Professor Manuel Ferreira, 89	Coffee Shop	4.6	$$	550m
Origami	R. Marquês de São Vicente, 52/Lj. 144	Japanese	4.5	$$$$	600m
...	...	...	...	...	...
Guimas	R. José Roberto Macedo Soares, 5	Brazilian	4.6	$$$$	950m

Table 1: example of question answer for an open-ended question

2 Overview

Multiple studies target to solve the task of translating natural language queries into SPARQL or SQL queries. Usually, the proposed QA process has four steps: *Question Analysis*, *Phrase Mapping*, *Disambiguation*, and *Query Construction* - not necessarily in this order. If multiple KBs are used, an extra step called *Distributed Knowledge* may exist.

Our study starts just after the last step of the QA pipeline mentioned. As input it's expected the keywords detected and the result set. There are two possible scenarios: result set with **single column**, or result set with **multiple columns**.

2.1 Single Column

Looking for some examples in QA over triplestores, we decided to focus on Question Answering over Linked Data (QALD[1]). Multiple papers use this benchmark as base to measure quality metrics of system's answers. We noticed that most answers were single column.

Thus, it was necessary to enrich the result set. A naive approach is to fill the instances returned with their properties. Generally, the answers represent instances from the same *rdf:type*. So, with a SPARQL query it is possible to bring all related properties and values.

As an example, we are going to analyse the query answer for the question *Give me all songs by O Rappa*. The result set has *mo:Track* instances - see Table

[1] http://qald.aksw.org

2. Then, modifying the original SPARQL query is possible to add properties and values in result set.

Notice in Table 3 the column *song* has repeated values. This is happening because the generic structure of (*subject, predicate, object*) is maintained in the result set. Instead of normalizing the returned table, we decided to keep this three-column format.

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?song
WHERE {
    ?song a mo:Track .
    ?song foaf:maker ?artist .
    ?artist foaf:name 'O Rappa'
}
```

song
db:track/fad758d8-9416-45e1-ab2b-9910f54a8353
db:track/5cfbac0b-1d40-485f-ad48-06849ee6b2af
db:track/0131ea07-f370-4560-8b40-3b17d68869ff
db:track/60ed8a2c-a694-4f3b-806b-6747e3a1bf69
db:track/0b329600-5b55-4b2b-a052-314d833cd5c1
db:track/d355cc55-1e3d-4649-9cde-9cbb309ee767
db:track/ccc63309-8f01-4db9-89ce-e3238aeda352
db:track/40665a38-cb8a-4f1d-87b8-7e8dd65e894a
db:track/8517f5b7-b74f-422a-9770-86a6603853a9
db:track/2924b5bc-d88c-4bfe-bf7f-6e54e9826902

Table 2: SPARQL query and single column return set

2.2 Multiple Columns

As mentioned before, when looking at the QALD benchmark we noticed few query answers were multiple columns. Even filtering for aggregation queries - train and test benchmark questions have a *boolean* property aggregation - the result set in most cases is single column.

Thus, it seems that this scenario is strongly related with relational approach. So, we decided not to consider these cases in this study.

2.3 Graph Analysis

A key characteristic of RDF models is the presence of metadata and data together. The triple form (*s, p, o*), where *s* is the subject, *p* is the predicate and *o*

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?song ?prop ?obj
WHERE {
    ?song a mo:Track .
    ?song foaf:maker ?artist .
    ?artist foaf:name 'O Rappa' .
    ?song ?prop ?obj
}
ORDER BY ?song
```

song	prop	obj
db:track/007e1f7f-54ad...	foaf:maker	db:artist/00034ede-a1f1...
db:track/007e1f7f-54ad...	mo:musicbrainz	mo:track/007e1f7f-54ad...
db:track/007e1f7f-54ad...	rdf:type	mo:Track
db:track/007e1f7f-54ad...	dc:date	0
db:track/007e1f7f-54ad...	mo:track_number	11
db:track/007e1f7f-54ad...	mo:length	291106
db:track/007e1f7f-54ad...	dc:title	"Candidato Caô Caô"
db:track/007e1f7f-54ad...	rdfs:label	"Candidato Caô Caô"
db:track/01130e65-04fc...	foaf:maker	db:artist/00034ede-a1f1...
db:track/01130e65-04fc...	mo:musicbrainz	mo:track/01130e65-04fc...
db:track/01130e65-04fc...	rdf:type	mo:Track
db:track/01130e65-04fc...	dc:date	0
db:track/01130e65-04fc...	mo:length	184600
db:track/01130e65-04fc...	mo:track_number	4
db:track/01130e65-04fc...	dc:title	"Miséria S.A"
db:track/01130e65-04fc...	rdfs:label	"Miséria S.A"
db:track/0131ea07-f370...	foaf:maker	db:artist/00034ede-a1f1...
db:track/0131ea07-f370...	mo:musicbrainz	mo:track/0131ea07-f370...
db:track/0131ea07-f370...	rdf:type	mo:Track
db:track/0131ea07-f370...	dc:date	0
db:track/0131ea07-f370...	mo:track_number	12
db:track/0131ea07-f370...	mo:length	299560
db:track/0131ea07-f370...	dc:title	"Papo de Surdo e Mudo"
db:track/0131ea07-f370...	rdfs:label	"Papo de Surdo e Mudo"

Table 3: SPARQL query and multiple columns return set

is the object of the triple, allows the triplestores to be interpreted as a *directed labeled graph*.

A set of RDF queries were used to generate graph statistics which helps the algorithm to decide what to do next. These statistics are related to unique classes, frequency of instances by classes, frequency of predicates and relationships between instances of classes.

The following queries can be performed in any SPARQL end-point. There is an execution order to be respected since relationships between instances of classes depend on available classes.

```
select distinct ?class {
    [] a ?class
}
```

```
select ?class (count(*) as ?frequency) {
    [] a ?class
}
group by ?class
order by desc(?frequency)
```

```
select ?predicate (count(*) as ?frequency) {
    [] ?predicate []
}
group by ?predicate
order by desc(?frequency)
```

```
select distinct ("$class_1" as ?class_1) ?r ("$class_2" as ?class_2) {
    {
        {
            select distinct ?v1 {
                ?v1 a <$class_1>
            }
            limit $limit_c1
        }
        union
        {
            select distinct ?v2 {
                ?v2 a <$class_2>
            }
            limit $limit_c2
        }
    }
    ?v1 ?r ?v2
}
```

3 Technologies

3.1 RDF

Resource Description Framework[2] (RDF) is part of World Wide Web Consortium (W3C) specifications and is a standard model for data interchange on the web. A key characteristic of the RDF is the presence of data and metadata combined which makes it flexible to support the evolution of schemas over time without breaking the way data is consumed.

The data representation is known as *triples* and the linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by graph nodes - (*subject, predicate, object*). This simple structure is very powerful enabling application in multiple scenarios.

3.2 Ontologies

An ontology is a formal description of knowledge as a set of concepts within a domain and the relationships that hold between them. The idea is to enhance the power of RDF adding new knowledge about a specific domain. Web Ontology Language[3] (OWL) is an example of ontology and is also part of the W3C Semantic Web technology stack. Its use is related to verify consistency of knowledge and to make implicit knowledge explicit.

Knowledge Bases usually refer to multiple ontologies. The idea with that is to reuse known terms to express statements in RDF graph and also to facilitate future comparisons between knowledge bases.

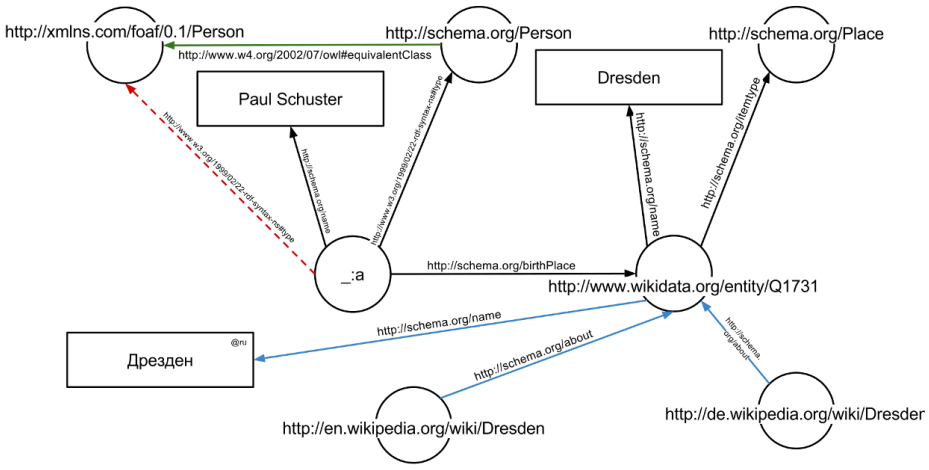

Fig. 1: RDF example

[2] https://www.w3.org/RDF/
[3] https://www.w3.org/OWL/

3.3 MusicBrainz / LinkedBrainz

MusicBrainz[4] is an open music encyclopedia that collects music metadata and makes it available to the public. Its data is available in relational database format. A project named LinkedBrainz[5] has the goal to publish MusicBrainz's data in Linked Data format.

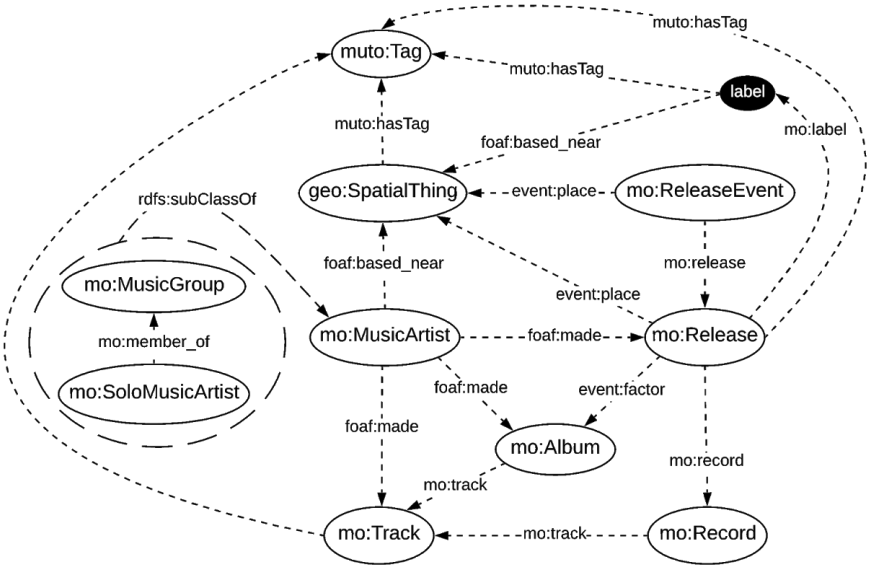

Fig. 2: LinkedBrainz's schema

3.4 Virtuoso Universal Server

Virtuoso Universal Server[6] is a middleware and database engine hybrid that combines multiple functionalities:

- Relational database (RDBMS)
- Object-relational database (ORDBMS)
- Virtual database
- Triplestore (RDF)
- Web Application Server
- File Server

[4] https://musicbrainz.org
[5] https://linkedbrainz.org
[6] https://virtuoso.openlinksw.com

In this project we installed Virtuoso as a docker container and used it as triplestore.

3.5 SPARQL

SPARQL[7] is a query language for RDF and can be used to express queries across multiple data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. As mentioned before, we have loaded LinkedBrainz's data into Virtuoso and then access its SPARQL's endpoint to retrieve information.

3.6 Python

Python[8] is an interpreted programming language used in many fields of computer science. It's a flexible language supporting object oriented, functional programming, and also imperative programming. Many developer surveys shows Python as fastest-growing major programming language. Hence, many interesting libraries are available and maintained by its engaged community.

3.7 Pandas

Pandas[9] is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. We decided to use its data structure called *DataFrame* to handle returned data queried from SPARQL's endpoint.

4 Documentation

4.1 Functional Requirements

1. The system is going to work only with questions in English language
2. The system is going to work only with triplestores
3. Questions, keywords, and SPARQL queries are loaded as INI (*.ini*) format
4. Graph statistics is persisted as JSON Lines (*.jsonl*) format
5. Graph statistics must be reused instead of recalculated on every run

4.2 Class Diagram

Our Class Diagram is represented in Figure 3. The system is composed by the following classes:

– **Dataset**: Responsible for loading *.ini* files which contains information about desired knowledge base

[7] https://www.w3.org/TR/rdf-sparql-query/
[8] https://www.python.org
[9] https://pandas.pydata.org

– **Question**: Responsible for wrapping system's input - question in natural language, keywords detected, sparql query and prefixes
– **SparqlQuery**: Responsible for performing queries over specified endpoint - all results are returned as *pandas.DataFrame*
– **GraphStatistics**: Responsible for calculating RDF graph statistics based on predefined SPARQL queries over resources and predicates
– **Enrich**: Responsible for applying heuristics over the result set of initial question - as the name suggests, it tries to enrich results with extra information
– **Main**: Responsible for starting application

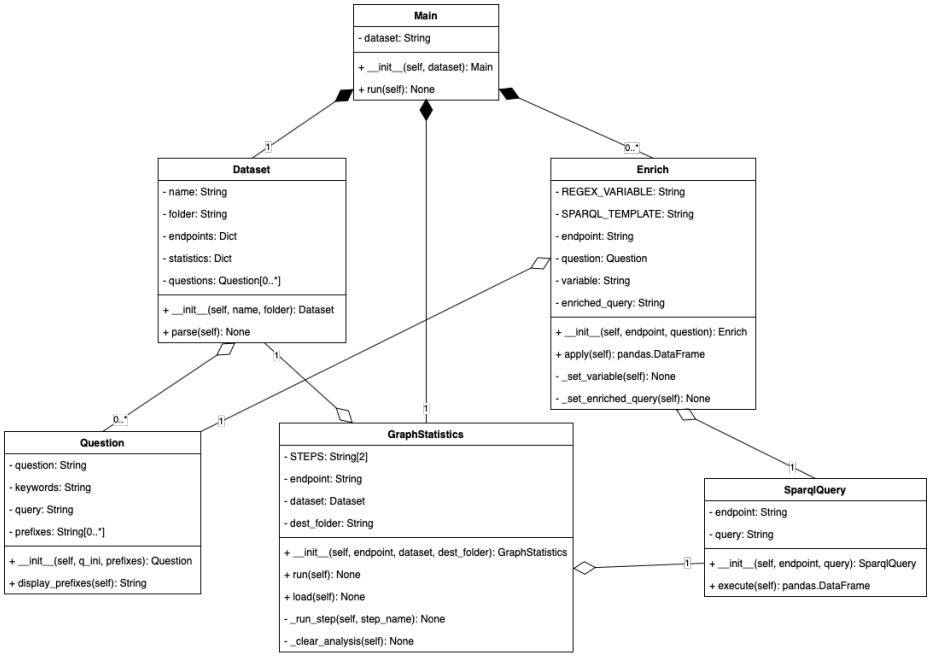

Fig. 3: Class Diagram

4.3 Sequence Diagram

Our Sequence Diagram is represented by Figure 4 and describes the whole process from *end-to-end*. It starts reading the input information - *question, keyword, query* - and finishes with the final result set.

4.4 Unit Testing

Automatic unit testing was applied in project to guarantee compliance even with future modifications. All methods of our six classes were tested using two

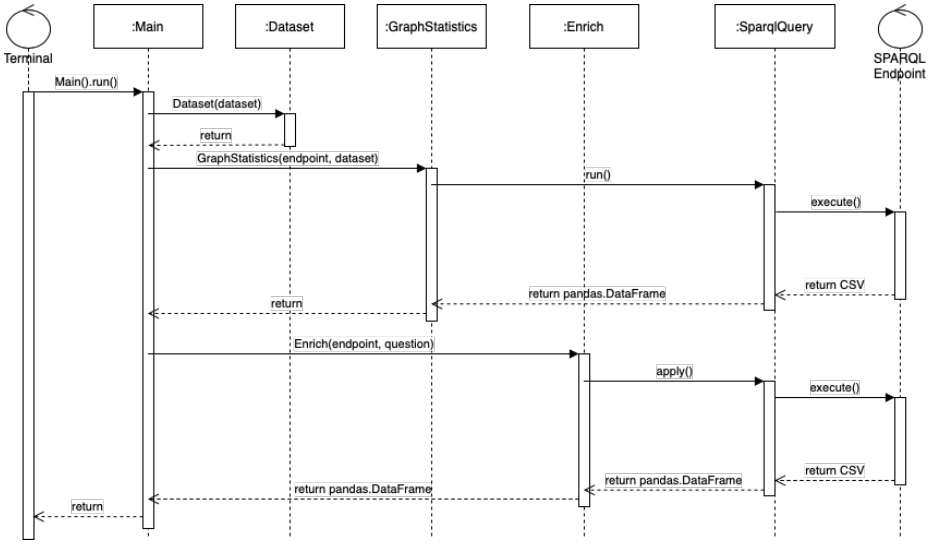

Fig. 4: Sequence Diagram

Python's library: *unittest* and *pytest*. In the following Figure 5 shows an example of class test.

5 Schedule

Table describes our development's schedule with start and end dates for each accomplished tasks.

Task	Start date	End date
Reading related papers	Mar 24	May 4
Requirements specification	May 5	May 19
Development	May 20	Jul 1
Unit Testing	Jul 2	Jul 6
Documentation	Jul 7	Jul 12

Table 4: Schedule of activities

6 Conclusion

In this paper we outline a preliminary study and propose an initial set of steps - which we called heuristics - to generate new questions based on the prior query

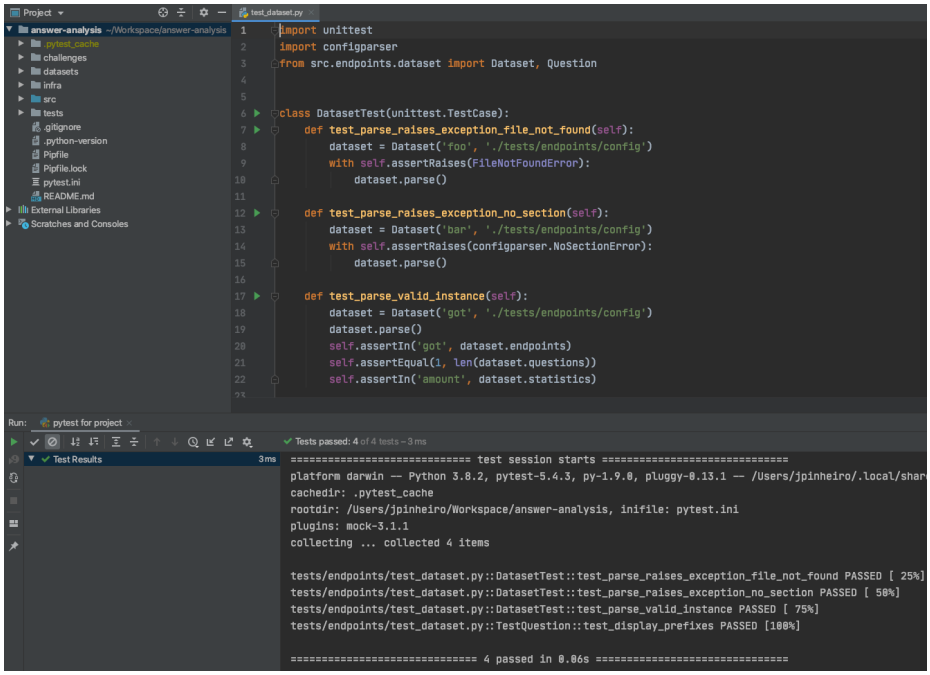

Fig. 5: Unit tests - class Dataset

answer result set. A refinement must still be done in order to apply these steps on different scenarios. Our strategy is still very attached with MusicBrainz dataset.

As future work, the Graph Analysis phase can be extended to contemplate other classical graph problems approach. Some algorithms like PageRank or HITS may be very useful to sort results and also to discard irrelevant information. Specifically with HITS, we could use hubs nodes to help graph navigation and use authority nodes to rank results.

In summary, we still have a way to go. But this study helped to build a clearer understanding of the problem and made easier to set goals for future research addressing the problem.

References

1. Dalianis H., Hovy E. (1996) Aggregation in natural language generation. In: Adorni G., Zock M. (eds) Trends in Natural Language Generation An Artificial Intelligence Perspective. EWNLG 1993. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), vol 1036. Springer, Berlin, Heidelberg https://doi.org/10.1007/3-540-60800-1_25
2. Deutch, Daniel & Frost, Nave & Gilad, Amir. (2017). Provenance for natural language queries. Proceedings of the VLDB Endowment. 10. 577-588. https://doi.org/10.14778/3055540.3055550.

3. Xin Hu, Depeng Dang, Yingting Yao, Luting Ye, Natural Language Aggregate Query over RDF Data, Information Sciences (2018)
https://doi.org/10.1016/j.ins.2018.04.042
4. Diefenbach, D., Lopez, V., Singh, K. et al. Core techniques of question answering systems over knowledge bases: a survey. Knowl Inf Syst 55, 529–569 (2018).
https://doi.org/10.1007/s10115-017-1100-y
5. Juan L. Reutter, Adrián Soto, and Domagoj Vrgoč. 2015. Recursion in SPARQL. In Proceedings of the 14th International Conference on The Semantic Web - ISWC 2015 - Volume 9366. Springer-Verlag, Berlin, Heidelberg, 19–35.
https://doi.org/10.1007/978-3-319-25007-6_2
6. Jeff Z. Pan, Guido Vetere, Jose Manuel Gomez-Perez, and Honghan Wu. 2017. Exploiting Linked Data and Knowledge Graphs in Large Organisations (1st. ed.). Springer Publishing Company, Incorporated.
https://doi.org/10.1007/978-3-319-45654-6
7. Isotani, Seiji: Dados abertos conectados
São Paulo, Novatec Editora, 2015
ISBN 978-85-7522-449-6