Vol.2016-DBS-163 No.20
Vol.2016-IFAT-123 No.20
2016/9/15

スキーマ定義が不明な Linked Data 検索のためのクエリ構築支援

閻家楠[†1]　阪口哲男[†2]

概要：近年，いくつかの Linked Data (LD) のデータセットを組み合わせてシステムを作り上げることが増えてきた．その際，開発者は LD のスキーマを確認しながら SPARQL クエリを組立て検索するが，スキーマ定義が記述されていない場合，開発者はスキーマを推定しながら SPARQL クエリを組立てるため，手間がかかる．そこで，本研究はその手間を減らすため，LD からクエリ組立に役立つ情報を抽出し，ヒントとして開発者に提示するための手法の確立を目的とする．入力された情報要求から検索クエリのグラフの生成と，そのグラフに基づく LD からの情報抽出の試みについて報告する．

キーワード：Linked Data，SPARQL，RDF

Support for Building Queries of Retrieving Linked Data whose Schemas are not Described

Jianan Yan[†1]　Tetsuo Sakaguchi[†2]

Abstract: Recently, the number of systems which use combinations of several datasets of Linked Data (LD) has been increased. In that case, to retrieve LD, developers usually build SPARQL queries based on the schemas of the LD datasets. But if the schema definitions are not described, it will take much effort of estimating the schemas for building SPARQL queries. Therefore, to reduce the effort of estimating schemas, the authors propose an approach to extract useful information for building queries from LD and show them as hints to the developers to build queries. This paper also discusses an experiment of a sample of extracting information.

Keywords: Linked Data, SPARQL, RDF

1. はじめに

近年，Linked Data (LD) のデータセットをいくつか集め，それらを組み合わせて応用システムを作ることが増えてきた. LD 応用システムの開発者が LD のデータセットから欲しい情報を探すには，Linked Data のスキーマを確認しながら SPARQL クエリを組み立て検索する．しかしながら，スキーマ定義が記述されていない LD が多いため，開発者はそのデータセットに対して様々なパターンの検索を試行錯誤し，スキーマを推定しながらクエリを組み立てる．しかし，それは手間がかかるため，その検索の手間を減らすことを目指す手法が提案されてきた．例えば，キーワード指定クエリから段階的に詳細化していく検索支援手法 MorphingAssist[1]は，構造を指定するクエリを書く知識が欠けているユーザ向けである．また，メタデータインスタンスと SPARQL クエリを用いたスキーマを抽出する Honma らの手法[2]は，rdf:type に依存するため，それが用いられていない場合は適用できない．SPARQL クエリの自動生成ツールとしては，SPARQL Builder[3]がある．これは入力と出力の 2 つのクラスを指定すれば，クラス間関係リストがユーザに提示され, ユーザがその中から 1 つ選ぶと，そのパスのクラス間関係に対応する SPARQL クエリが自動的に生成されるツールである．しかし，SPARQL Builder は生命科学分野の RDF データに限定されている．一方，リレーショナル・データベースのための対話型自然言語インタフェースの構築[4]は，ユーザが入力した質問文を木構造に変換し, 木構造の各ノードを SQL クエリの構成要素に対応してクエリを生成する手法であるが，その構成要素に対応するには，メタデータが必要である．

本研究の目的は，LD の構造を指定する検索クエリの書き方を知っている開発者向けに, スキーマ定義が不明な LD のデータセットからクエリの組み立てに役立つ情報を抽出し，必要な項目を補うことでクエリの組み立てを容易に可能にすることである．本研究では LD のデータセットに RDF モデルを, クエリ言語に SPARQL を使用しているものを対象とする．

2. Linked Data とその応用システムの開発

Linked Data はウェブ技術でデータを公開・共有するためのベストプラクティスであり，これまでウェブ上でデータ公開や共有について議論されてきたことの集大成である[5]. Linked Data 応用システムは，開発者がいくつかのデータセットを組み合わせて作り上げたシステムのことである. 例えば，「where does my money go?」[6]はイギリス政府によ

†1 筑波大学図書館情報メディア研究科
Graduate School of Library, Information and Media Studies, University of Tsukuba
†2 筑波大学知的コミュニティ基盤研究センター
Research Center for Knowledge Communities, University of Tsukuba

って公開された公共支出統計データを用いて開発されたシステムで,これをベースにして開発された日本語版もある.また,「さばえぶらり」[7]という鯖江市関連のデータセットと他 7 種類の地図画像データが用いられて開発された地図システムもある.

3. クエリの組み立てに関する検討

前節で述べたように LD 応用システムを開発するためには LD データセットのスキーマを知る必要がある.スキーマ定義が明示的に与えられていない場合,それを知るために,SPARQL クエリを用いて様々なパターンの検索を繰り返し,スキーマを推定しながらクエリを組み立てるという方法が多く使われている.そこで,クエリ組み立てのヒントを開発者に提示すれば,検索の手間も軽減される.スキーマ定義が不明であるため,LD のデータセットからクエリ組み立てに必要とする情報を抽出し,ヒントとして開発者に提示することが考えられる.

また,開発者に適切なヒントを提示するには,開発者の情報要求に基づいたクエリのパターンを生成し,そのパターン中の不確定な箇所とデータセットに照合した結果を用いることが考えられる.その際,複数の不確定な箇所があるパターンで問い合わせると検索結果が膨大になり,そのままではヒントとして提示することが難しくなる.そこで,パターンをいくつかの部分クエリに分け,その部分クエリで問い合わせし,その部分クエリ毎に結果を提示すれば,絞り込みやすくなると考えられる.

4. データセットより抽出可能な情報

クエリを組立てるためには,データセット中での RDF トリプルにどのような語彙が存在し,どのように組み合わさった構造になるかを知る必要がある.まずデータセット中から語彙を単純に抽出する予備実験を行い,どのような語彙が抽出可能かについて確認する.下記のような環境を用い,Dbpedia 2015-4 のデータセットを使い,その中から主語・述語・目的語ごとに出現頻度順に 1000 件を取り出した.

実験環境：
RDF ストア：Virtuoso 07.20.3212
OS：Ubuntu 14.04 LTS
投入データ：Dbpedia 2015-4（669,045,743 件）

まず主語 1000 件を調べてみると,リソースを表していて,基本的に URI で記述されている識別子であるため,それをヒントとして開発者に提示しても,そのリソースの識別情報や所在情報が分かるだけで,主語と目的語の関係を知ることができない.また,その URI 自体は意味を表すとは限らないため,欲しい情報を見つけるのは難しい.主語 1000 件の一部とその出現頻度を表 1 に示す.表 1 に抜粋した範囲では URI の末尾が単語の組み合わせであり,これだけを見ると意味の類推が可能に見えるが、中には末尾が単なる数字というのもある.

表 1　主語 1000 件の一部とその出現頻度

主語	出現頻度
http://de.dbpedia.org/resource/Liste_von_Vornamen	8802
http://dbpedia.org/resource/List_of_places_in_Afghanistan	7112
http://de.dbpedia.org/resource/Liste_der_Kulturdenkmäler_in_Bremen-Mitte	6946
http://de.dbpedia.org/resource/Liste_der_Stolpersteine_in_Hamburg-Rotherbaum	6399
http://de.dbpedia.org/resource/Liste_in_der_europäischen_Expansion_entdeckter_Inseln	6103
http://dbpedia.org/resource/Index_of_Andhra_Pradesh-related_articles	5959
http://dbpedia.org/resource/List_of_populated_places_in_Bosnia_and_Herzegovina	5878
http://dbpedia.org/resource/List_of_cities,_towns_and_villages_in_Kerman_Province	5848
http://dbpedia.org/resource/Alphabetical_list_of_communes_of_Italy	5615
http://dbpedia.org/resource/List_of_cities,_towns_and_villages_in_Sistan_and_Baluchestan_Province	5599
...	...

一方,述語 1000 件を調べてみると,述語の URI の末尾が意味を表す単語が多いため,開発者がそれを見れば主語と目的語の関係を推定しやすいことが多い.述語 1000 件の一部とその出現頻度を表 2 に示す.

表 2　述語 1000 件の一部とその出現頻度

述語	出現頻度
http://dbpedia.org/ontology/wikiPageWikiLink	218501067
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	74394795
http://www.w3.org/2002/07/owl#sameAs	50221229
http://dbpedia.org/property/wikiPageUsesTemplate	30174599
http://purl.org/dc/terms/subject	27177158
http://www.w3.org/2000/01/rdf-schema#label	23224850
http://dbpedia.org/ontology/wikiPageWikiLinkText	22321882
http://dbpedia.org/ontology/wikiPageLength	14542119
http://dbpedia.org/ontology/wikiPageOutDegree	14542119
http://dbpedia.org/ontology/abstract	12176785
...	...

最後に,目的語 1000 件を調べてみると,URI で記述されているものはあるが,リテラルで記述されているものも多いため,文字列パターンや値の範囲指定など検索条件に

使うことが考えられる. 目的語 1000 件の一部とその出現頻度を表 3 に示す.

表 3　目的語 1000 件の一部とその出現頻度

目的語	出現頻度
1	8428124
http://www.w3.org/2002/07/owl#Thing	5550962
http://dbpedia.org/ontology/Image	3844944
http://dbpedia.org/ontology/Person	3205565
http://dbpedia.org/resource/Template:Reflist	3199902
http://xmlns.com/foaf/0.1/Document	3043085
"de"^^<http://www.w3.org/2001/XMLSchema#string>	3022972
http://dbpedia.org/ontology/Agent	2992151
http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Agent	2984759
http://xmlns.com/foaf/0.1/Person	2715692
...	...

以上より本研究では，目的語に対する文字列パターンや値の範囲などの条件指定を行うと共に，クエリパターンで表される構造を用いてデータセットを検索し，述語の候補をヒントとして開発者に提示し，検索クエリの組み立てを支援する.

5. クエリの構築支援

本研究では, 情報要求から検索クエリのグラフを生成し，そのグラフに基づいて未知の述語を探し，クエリの作成を支援する．また，述語の候補をクエリ組立のヒントとして開発者に提示する.

5.1 クエリ構築の過程

開発者が情報要求を入力すると，それに基づいた検索クエリのグラフが生成され，その部分クエリの検索を通じて述語候補が提示され，開発者が選択する．全ての述語候補が確定すれば，完成したクエリとして出力する対話型システムの構築を目指す.

全体の流れは以下のようになる：

- 開発者が情報要求を入力すると, システムがその情報要求をクエリのグラフに変換し，出力する.
- システムがクエリのグラフに基づいて部分クエリを生成して検索を行い, 提示された述語の候補とその出現頻度から選ぶ.
- 前のステップを繰り返してクエリを完成する.
- 途中で述語の候補が提示されない場合, 開発者がクエリのグラフの別候補を選択したり, グラフの一部を修正した上で, 任意のステップに戻ることができる.

以上の過程を情報要求例に適用した事例を 5.2 に示し，それに基づいて検討したグラフ生成に用いる規則を 5.3 で述べる.

5.2 クエリ構築の過程例

本節では，そこに必要な技術要素を明確にするために，クエリ構築の過程を人手で行った事例を示す.

Dbpedia 2015-4 のデータセットを使用し,「1960 年 1 月 1 日以前に生まれた芸術家の名前を知りたい」を情報要求例として検索クエリ完成までの過程を追った.

設計したクエリ構築の過程を例で説明すると，以下の通りになる：

(1)　「1960 年 1 月 1 日以前に生まれた芸術家の名前を知りたい」という情報要求を文節に分割する（図 1）.

（1960年1月1日以前に）（生まれた）（芸術家の）（名前を）（知りたい）
A　p1　X　p3, Y

図 1　情報要求の分割結果

(2)　「芸術家」は「芸術する人」なので,「芸術家の」を更に「芸術する」と「人」に分割する（図 2）.

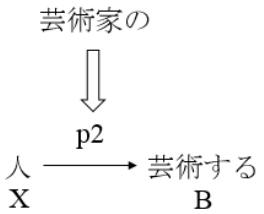

図 2 「芸術家の」の分割結果

(3)　図 1 の分割結果により，1 つの係り受け関係を 1 つの RDF トリプルとする.「芸術家の」に対する RDF トリプルは図 2 に示す．また，検索したい部分を主語とし，動詞を含む部分を述語とし，具体値を示す名詞や形容詞を含む部分を目的語とする. それらのトリプルの組み合わせが検索クエリのグラフとなる（図 3）.

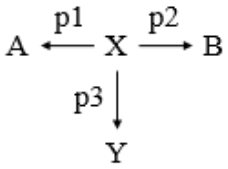

図 3　検索クエリのグラフ

(4)　クエリのグラフに基づいて，X と A を p1 で繋ぐ部分グラフに対する部分クエリ I で検索を行い, 述語の候補 p1 とその出現頻度を提示する．部分クエリ I は図 4 のようになり, その検索結果を表 4 に示す.

Vol.2016-DBS-163 No.20
Vol.2016-IFAT-123 No.20
2016/9/15

```
select distinct ?p1, (count(?p1) as ?cp)
from <http://dbpedia.org> ... where
{ ?X ?p1 ?A.
FILTER(?A<=
"1960-01-01"^^xsd:dateTime)
}
group by ?p1
order by DESC(?cp)
```

図 4　部分クエリ I

表 4　部分クエリ I の検索結果

p1	cp
http://dbpedia.org/property/dateOfBirth	471407
http://dbpedia.org/ontology/birthYear	326395
http://dbpedia.org/ontology/birthDate	245681
http://dbpedia.org/property/date	195289
...	...

(5) 表 4 より人物の生年月日の意味を表している「http://dbpedia.org/ontology/birthDate」を選び，部分クエリ I を完成する．次の X と B を p2 で繋ぐ部分グラフに対する部分クエリ II と繋いで検索を行い，述語の候補 p2 とその出現頻度を提示する．部分クエリ II は図 5 のようになり，その検索結果を表 5 に示す．

```
prefix a: <http://dbpedia.org/ontology/>
select distinct ?p2, (count(?p2) as ?cp)
from <http://dbpedia.org> ... where
{ ?X a:birthDate ?A;
      ?p2 ?B.
FILTER(?A<=
"1960-01-01"^^xsd:dateTime)
FILTER regex(?B,“artist","i")
}
group by ?p2
```

図 5　部分クエリ II

表 5　部分クエリ II の検索結果

p2	cp
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	59636
http://dbpedia.org/ontology/wikiPageWikiLink	23498
http://purl.org/dc/terms/subject	17091
http://dbpedia.org/ontology/abstract	15915
http://dbpedia.org/property/wikiPageUsesTemplate	13069
...	...

(6) 表 5 より人物の紹介文の意味を表している「http://dbpedia.org/ontology/abstract」を選び，部分クエリ II を完成する．次の X と Y を p3 で繋ぐ部分グラフに対する部分クエリ III と繋いで検索を行い，述語の候補 p3 とその出現頻度を提示する．部分クエリ III を図 6 のようになり，その検索結果を表 6 に示す．

```
prefix a: <http://dbpedia.org/ontology/>
select distinct ?p3, (count(?p3) as ?cp)
from <http://dbpedia.org> ... where
{ ?X a:birthDate ?A;
      a:abstract ?B;
      ?p3 ?Y.
FILTER(?A<=
"1960-01-01"^^xsd:dateTime)
FILTER regex(?B,"artist","i")
}
group by ?p3
order by DESC(?cp)
```

図 6　部分クエリ III

表 6　部分クエリ III の検索結果

p3	cp
http://dbpedia.org/ontology/wikiPageWikiLink	11280
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	3533
http://purl.org/dc/terms/subject	1733
http://dbpedia.org/property/wikiPageUsesTemplate	1362
http://www.w3.org/2002/07/owl#sameAs	1202
http://www.w3.org/2000/01/rdf-schema#label	795
...	...

(7) 表 6 よりラベルを意味している「http://www.w3.org/2000/01/rdf-schema#label」を選び，部分クエリ III を完成すれば，クエリも完成する．完成されたクエリを図 7 に示し，最後の検索結果の一部を表 7 に示す．

```
prefix a: <http://dbpedia.org/ontology/>
prefix b: <http://www.w3.org/2000/01/rdf-schema#>
select distinct ?Y
from <http://dbpedia.org> ... where
{ ?X a:birthDate ?A;
      a:abstract ?B;
      b:label ?Y.
FILTER(?A<= "1960-01-01"^^xsd:dateTime)
FILTER regex(?B,"artist","i")
}
```

図 7　完成されたクエリ

Vol.2016-DBS-163 No.20
Vol.2016-IFAT-123 No.20
2016/9/15

表 7　最後の検索結果（575 件）の一部

Y
"Franz von Bayros"@de
"Franz von Bayros"@en
"Franz von Bayros"@es
"Franz von Bayros"@fr
"Franz von Bayros"@it
"Franz von Bayros"@nl
"Franz von Bayros"@pt
"Байрос, Франц фон"@ru
…

5.3 検索クエリのグラフの生成規則

本節では，構文解析ツールで情報要求を解析し，その結果に基づいてクエリのグラフを生成する規則について述べる．情報要求から検索クエリのグラフへの変換ルールによっていくつかのグラフが出力される可能性があるが，今回は 1 つだけを示す．

日本語係り受け解析器 Cabocha[8]を利用し，その出力結果をクエリのグラフに変換する規則を定め，自動変換を試みる．規則の検討においては，まず 5.2 で取り上げた情報要求例について変換できるものを定義する．そして，それを他の情報要求例にも適用し，不十分なところを補うことでより汎用性の高い規則とする．

まず，簡単な場合について規則が定義可能であることを確かめるため，次のような制約がある場合について考える．

- 情報要求は「～を知りたい」という形式に限定するものとする．
- 生成するグラフにおいて，RDF の主語は 1 つであると仮定する．
- 最後に得るのはリテラルである場合に限る．

情報要求例の構文解析結果を図 8 に示す．図中で，「*」で始まる行から次の「*」の前までは 1 つの文節を表し，「*」の後に続く数値は文節番号である．文節番号の後に続く数値は順次に係り先の文節番号，主辞の形態素番号/機能語の形態素番号，係り関係のスコアとなる．係り先なしの場合は，係り先の文節番号は-1 となる．

以下，文節番号を i，節 i を C_i，主語 j を S_j，目的語 i を O_i，S_j と O_i を繋ぐ述語を P_{ji} として表す．ここで主語，述語，目的語は RDF モデルにおけるものである．文節の係り受け関係を図 9 に示す．変換ルールを文節結合ルールとトリプル生成ルールに分け，まず文節結合ルールを適用することで，細かく分かれ過ぎた文節を結合し，その後トリプル生成ルールを適用する．文節結合ルールは以下のように定義した．

- もしある節 C_i の機能語と主辞が名詞であり，係る先の節 C_j の主辞が名詞であれば，C_i と C_j を 1 つの節 C_k と見なし，C_k の機能語と主辞を C_j と同じにする．

トリプル生成ルールは以下のように定義した．

- 「知りたい」に係る節 C_i の主辞が目的語に相当し，これを O_i として select に指定する．
- 節 C_i の主辞が名詞であり，且つそれに係る節 C_j の機能語が助詞であれば，節 C_j の主辞が主語に相当し，これを S_j とし，節 C_i の主辞が目的語に相当するので，S_j と O_i を繋ぐ述語を P_{ji} としたトリプル{S_j, P_{ji}, O_i}とする．
- 節 C_j に係る節 C_k の主辞が動詞であり，節 C_l が節 C_k に係るのであれば，節 C_k の主辞を S_j と O_l を繋ぐ述語 P_k としたトリプル{S_j, P_k, O_l}とする．
- 前述の全ての節の中にある節の主辞が名詞であれば，その名詞の意味を単語として FILTER に指定し，条件は開発者に書いてもらう．

```
1960年1月1日以前に生まれた芸術家の名前を知りたい
* 0 1D 4/4 1.482436
1       名詞,数,*,*,*,*,1,イチ,イチ,,
9       名詞,数,*,*,*,*,9,キュウ,キュー,,
6       名詞,数,*,*,*,*,6,ロク,ロク,,
0       名詞,数,*,*,*,*,0,ゼロ,ゼロ,,
年      名詞,接尾,助数詞,*,*,*,年,ネン,ネン,,
* 1 2D 0/0 0.740106
1月     名詞,副詞可能,*,*,*,*,1月,イチガツ,イチガツ,,
* 2 3D 2/3 1.416432
1       名詞,数,*,*,*,*,1,イチ,イチ,,
日      名詞,接尾,助数詞,*,*,*,日,ニチ,ニチ,,
以前    名詞,副詞可能,*,*,*,*,以前,イゼン,イゼン,,
に      助詞,格助詞,一般,*,*,*,に,ニ,ニ,,
* 3 4D 0/1 0.874276
生まれ  動詞,自立,*,*,一段,連用形,生まれる,ウマレ,ウマレ,うまれ/生まれ/生れ,
た      助動詞,*,*,*,特殊・タ,基本形,た,タ,タ,,
* 4 5D 1/2 1.609654
芸術    名詞,一般,*,*,*,*,芸術,ゲイジュツ,ゲイジュツ,,
家      名詞,接尾,一般,*,*,*,家,カ,カ,,
の      助詞,連体化,*,*,*,*,の,ノ,ノ,,
* 5 6D 0/1 1.609654
名前    名詞,一般,*,*,*,*,名前,ナマエ,ナマエ,,
を      助詞,格助詞,一般,*,*,*,を,ヲ,ヲ,,
* 6 -1D 0/1 0.000000
知り    動詞,自立,*,*,五段・ラ行,連用形,知る,シリ,シリ,しり/知り,
たい    助動詞,*,*,*,特殊・タイ,基本形,たい,タイ,タイ,,
EOS
```

図 8　情報要求例の構文解析結果

（1960年1月1日以前に）→（生まれた）→（芸術家の）→（名前を）→（知りたい）
C_4　　C_3　　C_2　　C_1

図 9　情報要求例の係り受け関係

5.4　クエリ変換ルールの適用例

本節では，5.3 で定義した変換ルールを図 8 の情報要求例に適用し，結果の妥当性を確認する．

まず解析結果に対して文節結合ルールを適用し，細かく分割された文節のうち，以降の処理で一つの文節としたいものを結合する．

- 節「1960 年」の機能語と主辞が名詞であり，係る先の節「1 月」の主辞が名詞であるため，その 2 つの節を 1 つの節「1960 年 1 月」と見なし，機能語と主辞が名詞になる．
- また，節「1960 年 1 月」の機能語と主辞が名詞であり，係る先の節「1 日以前に」の主辞が名詞であるため，それらを 1 つの節「1960 年 1 月 1 日以前に」と見なし，主辞が名詞になり，機能語が助詞になる．

次に，結合した文節と他の文節に対してトリプル生成ル

ールを適用し，以降の処理でトリプルを生成する．

- 「知りたい」に係る節「名前を」を C_1 とし，C_1 の主辞が目的語に相当し，これを O_1 として select に指定する．
- 節 C_1 の主辞が名詞であり，且つそれに係る節「芸術家の」の機能語が助詞であるため，節「芸術家の」を C_2 とし，C_2 の主辞が主語に相当し，これを S_2 とし，節 C_1 の主辞が目的語に相当するので，S_2 と O_1 を繋ぐ述語を P_{21} としたトリプル{S_2, P_{21}, O_1}とする．
- 節 C_2 に係る節「生まれた」を節 C_3 とし，節「1960年1月1日以前に」を C_4 とし，C_3 の主辞が動詞であり，節 C_4 が節 C_3 に係るため，節 C_3 の主辞を S_2 と O_4 を繋ぐ述語 P_3 としたトリプル{S_2, P_3, O_4}とする．完成したクエリグラフを図 10 に示す．

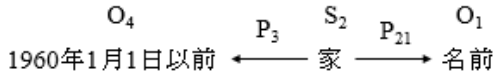

図 10　完成したクエリグラフ

図 10 を図 3 と比較してみると，予想より S_2 を「芸術する人」と指定するトリプルが欠けている．文節として分解できてないため，「芸術家の」は FILTER に指定する条件として何かグラフパターンに補う形式にすると考えられる．また，他の様々な情報要求例にも適用し，不十分なところを修正していく．

6. おわりに

本稿では，情報要求の構文解析結果によって検索クエリのグラフを生成し，そのグラフに基づいて未知の述語を探し，クエリの作成支援手法を提案した．今のところで情報要求は「～を知りたい」という形で，主語は 1 つの場合にしか適用しないが，今後は例を増やし，任意の情報要求と主語 1 つ以上の場合に適用できるかどうかについて検討を進めていく．

参考文献

[1] Yui Yasunaga, Atsuyuki Morishima. "MorphingAssist: seamless transformation from keywords to structured queries." Proceedings of the 31st Annual ACM Symposium on Applied Computing. 2016, p. 808-811.
[2] Tsunagu Honma, et al. "Extracting description set profiles from RDF datasets using metadata instances and SPARQL queries." Proceedings of the 2014 International Conference on Dublin Core and Metadata Applications. Dublin Core Metadata Initiative, 2014, p. 109-118.
[3] Atsuko Yamaguchi, et al. "An intelligent SPARQL query builder for exploration of various life-science databases." Proceedings of the 3rd International Conference on Intelligent Exploration of Semantic Data, 2014, Vol. 1279, p. 83-94.
[4] Fei Li, Jagadish H V. "Constructing an interactive natural language interface for relational databases." Proceedings of the VLDB Endowment, 2014, vol. 8, no. 1, p. 73-84.
[5] 加藤文彦，川島秀一，岡別府陽子，山本泰智，片山俊明．オープンデータ時代の標準 Web API SPARQL．株式会社インプレス R&D, 2015, p. 14.
[6] “where does my money go?”. http://app.wheredoesmymoneygo.org/, (参照 2016-08-14).
[7] “さばえぶらり”. http://atr-c.jp/burari/product/oldmap/sabae.html, (参照 2016-08-14).
[8] Taku Kudo. "CaboCha: Yet Another Japanese Dependency Structure Analyzer." Technical report, Nara Institute of Science and Technology, 2004.