SSBasket: SuperSQL の SPARQL 対応拡張による LOD の入れ子表示

徳永あゆみ† 五嶋 研人† 根本 潤†† 遠山 元道†

† 慶應義塾大学 理工学部 情報工学科 〒 223–8522 神奈川県横浜市港北区日吉 3–14–1
†† 慶應義塾大学 大学院 政策・メディア研究科 〒 252-0882 神奈川県藤沢市遠藤 5322
E-mail: †{ayumi,goto}@db.ics.keio.ac.jp, ††nemoto@keio.jp , †††toyama@ics.keio.ac.jp

あらまし RDF 用クエリ言語 SPARQL を用いて Linked Open Data(LOD) を検索した結果は一般に表, あるいはグラフ形式で利用される. しかしながら表示する際のレイアウトを編集する場合, HTML や CSS の知識が必要となり手間がかかってしまう. 本論文では, SuperSQL を用いて連鎖的に関連するデータを入れ子表として表示するシステム (SSBasket) を提案した. SuperSQL とは SQL の拡張言語であり, 独自のクエリを記述して関係データベースの出力結果を構造化することで多彩なレイアウト表現の実現を可能とするものである. SSBasket は SPARQL に対応し, 動的に, 再帰的な問い合わせを行うことができる. 入れ子表示では, SuperSQL の記述力, 表現力を生かした表示を可能とする.

キーワード SPARQL, DBPedia, 入れ子構造, データベース, SuperSQL

1 はじめに

近年, Web を介してデータの公開・共有をオープンに行う技術を Linked Open Data(LOD) [1] の普及が進められている. 一般に LOD は Resource Description Framework(RDF) [2] として RDF ストアに格納され, RDF 問い合わせ言語 SPARQL [3] に対応する.

先行研究ではユーザーが記述した SPARQL クエリを SPARQL エンドポイントに発行し, 結果を表示したり, ファイルとしてダウンロードしたりするシステムが開発されてきた.

ここで, 結果の表示は表あるいは, グラフ形式で行われるが, 各システムの表示形式に依存している. そのため, ユーザーが自由にレイアウトを指定することが困難である. 任意のレイアウトで表示するためには, 取得した SPARQL 問い合わせ結果を表示するためのプログラムを独自に用意する必要があり, 専門的な知識や技術が必要となる.

この問題を解決するために, 本論文では SQL の拡張言語である SuperSQL [4] [5] を SPARQL に対応させ, SPARQL 問い合わせ結果をユーザーが自由にレイアウト指定して表示できるシステムを提案した.

本論文の構成を以下に示す. まず, 第 2 章では関連研究について述べる. 次に第 3 章で SuperSQL の概要について述べる. そして第 4 章で SuperSQL を SPARQL 対応するに当たり拡張した点について述べ, 第 5 章で提案手法である SSBasket について述べる. 第 6 章で評価について述べ, 最後に第 7 章で結論を述べる.

2 関連研究

本論文では SSBasket クエリ内に SPARQL を記述しているが, SPARQL クエリ自体の記述・視覚化を支援する研究は既に行われている. Apache Jena Fuseki [6] は GUI で RDF データをロードして SPARQL 問い合わせが行える RDF ストアである. 視覚的なネットワークグラフを用いて RDF データを探索できるものとしては, Boniati ら [7] による SPARQL クエリログを基にクエリの形状をグラフで表した SHARQL や, DBCLS [8] による Endpoint browser が挙げられる. Haag ら [9] は Filter/Flow グラフの概念を拡張し, SPARQL 問い合わせを直感的に行えるツールを開発した. また, Bottoni ら [10] は Google が提供する Blockly を用いるユーザーインターフェースを紹介し, Skjæveland [11] は SPARQL 問い合わせ結果を Google Chart Tools を利用して視覚化する JavaScript ラッパーである Sqvizler を, DBCLS は Data-Driven Documents(D3) を利用する d3sparql [12] を提案した.

関連研究らは視覚的に SPARQL 問い合わせを行い, 結果を閲覧できる. しかし, 各システムごとに結果を表示するレイアウトが定められている. ユーザーが自由なレイアウトを指定するためには独自にシステムを変更, 拡張する必要があり, 労力がかかる.

3 SuperSQL

この章では, SuperSQL [4] [5] について簡単に述べる. SuperSQL は関係データベースの出力結果を構造化し, 多様なレイアウト表現を可能とする SQL の拡張言語であり, 慶應義塾大学遠山研究室で開発されている. そのクエリは SQL の SELECT 部を GENERATE <*media*> <*TFE*> の構文を持つ GENERATE 句で置き換えたものである. ここで <*media*> は出力媒体を示し, HTML, Unity などの指定ができる. また <*TFE*> はターゲットリストの拡張である Target Form Expression を表し, 結合子, 反復子などのレイアウト指定演算子を持つ一種の式である.

3.1 結合子

結合子はデータベースから得られたデータをどの方向 (次元) に結合するかを指定する演算子であり, 以下の 3 種類がある.

括弧内はクエリ中の演算子を示している.

- 水平結合子 (,)

データを横に結合して出力する.

例：name, place

name	place

- 垂直結合子 (!)

データを縦に結合して出力する.

例：name! place

name
place

3.2 反 復 子

反復子は指定する方向に, データベースの値があるだけ繰り返して表示する. また反復子はただ構造を指定するだけではなく, そのネストの関係によって属性間の関連を指定できる. 例えば

[部署名]! , [雇用者名]! , [給料]!

とした場合には各属性間に関係はなく, 単に各々の一覧が表示されるだけである. 一方, ネストを利用して

[部署名! [雇用者名, 給料]!]!

とした場合には, その部署毎に雇用者名と給料の一覧が表示されるといったように, 属性間の関連が指定される. 以下, その種類について述べる.

- 水平反復子 ([ ],)

データインスタンスがある限り, その属性のデータを横に繰り返し表示する.

例：[Name],

name1	name2	・・・	name10

- 垂直反復子 ([ ]!)

データインスタンスがある限り, その属性のデータを縦に繰り返し表示する.

例：[Name]!

name1
name2
・・・
name10

3.3 装 飾 子

SuperSQL では関係データベースより抽出された情報に, 文字サイズ, 文字スタイル, 横幅, 文字色, 背景, 高さ, 位置などの情報を付加できる. これらは装飾演算子 (@) によって指定する.

< 属性名 >@{< 装飾指定 >}

装飾指定は “装飾子の名称 = その内容” として指定する. 複数指定するときは各々を “,” で区切る.

[name@{width=100, color='red'}]!

3.4 FROM 句と WHERE 句

SuperSQL 質問文の FROM 句と WHERE 句の記述方法は SQL と同じで, 下記の様になる.

```
FROM dept d, store s
WHERE d.store = s.id
```

3.5 関　　数

SuperSQL にはいくつかの関数が用意されている. 画像を表示することのできる image 関数やリンクを生成することのできる link 関数, anchor 関数, クロス表を生成することのできる cross_tab 関数など様々なものがある. また, 通常の関数とは異なる集約関数というものもある. これは, 複数の入力行から 1 つの結果となる値を生成する関数である. 例としては最大値を返す max, 最小値を返す min, 平均値を返す avg などが挙げられる. 集約関数では引数を丸括弧ではなく角括弧で囲う形式で記述する.

4 SuperSQL の SPARQL 対応

この章では, SuperSQL で SPARQL 問い合わせを可能とするために, SuperSQL を拡張した点について述べる.

4.1 FROM 句拡張

従来の SuperSQL クエリの FROM 句に記述できるのは, 関係データベースのテーブル名と対応するエイリアスであった。そこで, SuperSQL の FROM 句に CSV ファイル名と SPARQL クエリを記述できるように拡張した.

関係データベースに格納されたデータと区別するため, FROM 句に関係データベースのテーブル名以外を用いる際は,
“ ”# ” と “ ” ” で囲んで記述する.

FROM ”#< 入力 >”

1 つの SuperSQL クエリに対して, CSV ファイル名は複数, SPARQL クエリは 1 つ記述できる. 以下に FROM 句に CSV ファイル名, SPARQL クエリを記述する例をそれぞれ示す.

- CSV ファイル名

```
FROM ''#meibo1.csv'' m1, ''#meibo2.csv'' m2
```

更に,

FROM ”#<CSV ファイル名 >(< カラム名 カラム型, ...>)”

と記述することで, CSV ファイルを関係データベースに登録する際のカラム名, カラム型を指定できる. 尚, 1CSV ファイルが 1 テーブル, CSV の各行が 1 タプルとなる.

```
FROM ’’#meibo1.csv(name TEXT, age INTEGER)’’ m1
```

meibo1.csv

Sato, 30
Suzuki, 25
Tanaka, 40

name	age
Hanako	30
Suzuki	25
Tanaka	40

- SPARQL クエリ

```
FROM ’’#SPARQL(SELECT ?o WHERE
{ ?コンピュータ ?v ?o. LIMIT 100 } )’’ s
```

SPARQL クエリを記述する際は, その旨を明示するため,

”#SPARQL(<SPARQL クエリ >)”

と記述する.

4.2 SPARQL 問い合わせ処理

RDF 用のライブラリである Jena [13] を利用し, DBPedia Japanese [14] の SPARQL エンドポイントに対し, SPARQL1.1 の問い合わせを可能とした. 図 1 に SPARQL 問い合わせ処理の流れを示す.

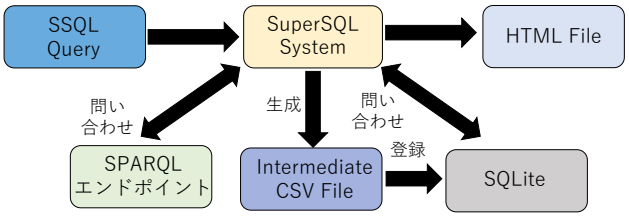

図 1 SPARQL Process

SSQL クエリの FROM 句に記述された SPARQL クエリを抽出し, SPARQL エンドポイントに問い合わせを行う. 問い合わせ結果から中間 CSV ファイルを生成し, 中間データベースである, SQLite に登録する. 最後に SQLite の出力結果を構造化し, HTML を出力する.

4.3 中間データベースの導入

中間データベースとして SQLite3 を導入した. SQLite3 は, FROM 句に CSV ファイル名, あるいは SPARQL クエリが記述されている際に, 中間テーブルを生成するために利用される.

SQLite [15] はオープンソースのリレーショナルデータベース管理システム (RDBMS) の 1 つで, サーバーとしてではなく, アプリケーションに組み込んで利用する. また, 簡易的なデータベースであり, データ型の指定を強制しないといった特徴がある. そのため, RDBMS ごとに決まるデータ型の差異を許容可能であり, 中間テーブルの生成に適していると考えた. 中間テーブルを導入した理由は, 従来の SuperSQL が関係データベースの出力結果を構造化するものであるため, 関係データベース以外の入力を直に扱えなかったからである. また, 中間データベースを導入した理由は, ユーザーが SuperSQL に参照先として設定したデータベースに INSERT, UPDATE, DELETE 等の権限があるとは限らないからである.

5 SSBasket

本章では提案手法である SSBasket について述べる.

5.1 概　　要

本節では, SSBasket クエリの定義とシステム概要を示す.

5.1.1 SSBasket クエリ

SSBasket クエリは SuperSQL 部 (SSQL 部), RECURSIVE_SPARQL 部 (RE_SPARQL 部), RECURSIVE_EXPRESSION 部 (RE_EXPRESSION 部) から構成される.

```
SSBASKET(<Hop_Count>) {
        SSQL{
                <SSQL query>
        }
        RECURSIVE_SPARQL{
                <SPARQL query>
        }
        RECURSIVE_EXPRESSION{
                <TFE>
        }
}
```

```
SSBASKET(Hop_Count) {
        SSQL{
                GENERATE <media> <TFE>
                FROM <from clause>
                WHERE <where clause>
        }
        RECURSIVE_SPARQL{
                SELECT select_clause
                WHERE{ where_expression }
                other_expression
        }
        RECURSIVE_EXPRESSION{
                <TFE>
        }
}
```

図 2 SSBasket クエリの定義

SSQL 部には初回問い合わせ時に用いる SuperSQL クエリを記述する. SuperSQL クエリの FROM 句には従来データベースのテーブル名を記述していたが, ここでは初回問い合わせに使用する SPARQL クエリを “’’#SPARQL()”’ の () の中に記述する. RE_SPARQL 部では再問い合わせ時に用いる SPARQL クエリを記述し, RE_EXPRESSION 部では再問い合わせ時に生成される SuperSQL クエリの構造を指定する. また, *Hop_Count* には, 1 回の再問い合わせで何ホップ目まで SPARQL クエリを実行するかを指定する.

5.1.2 システム概要

システムのユーザーは, 記述した SSBasket クエリを SSBasket システムに通して HTML ファイルを生成する. SSBasket は SuperSQL System, SPARQL Process, Re-Execution から構成される.

SPARQL Process は, 4 章で述べた SuperSQL を拡張に該当する. またシステムの実行の流れは初回問い合わせと再問い合わせに分かれる. 各問い合わせの流れを図 3, 図 4 に示す.

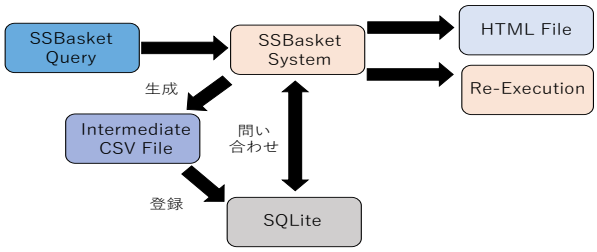

図 3　初回問い合わせ処理

初回問い合わせでは, SSBasket クエリに記述された SSQL 部の WHERE 句を基に SPARQL 問い合わせを行う. 問い合わせ結果から中間 CSV ファイルを生成し, この中間 CSV ファイルを SQLite に登録する. そして SuperSQL システムが SQLite を参照し, SSQL 部の GENERATE 句の装飾子を基に構造を指定し, HTML ファイルを生成する.

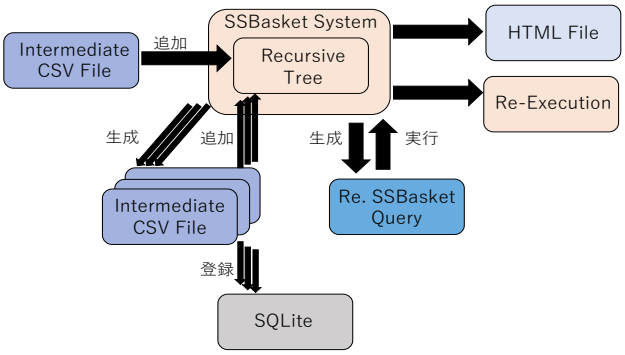

図 4　再問い合わせ処理

再問い合わせでは, *Hop_Count* 目まで, RE_SPARQL 部の SPARQL クエリを用いて問い合わせを行い, *Recursive Tree* と RE_EXPRESSION 部を基に新たな SSBasket クエリを生成し, HTML ファイルを出力する. 1 回目の再問い合わせは自動的に実行され, 2 回目以降は動的に行うことができる.

5.2　Recursive Tree

Recursive Tree は SPARQL 問い合わせ結果を格納する木構造である. *Recursive Tree* により, 各中間 CSV ファイルの親子関係を表現できる. *Recursive Tree* の定義は

$$Leaf[<csvfilename>, <word>, <url>, <child\_Leaf>...]$$

である. *csvfilename* は該当する中間 CSV のファイル名, *child_Leaf* には子要素の *Leaf* が追加されていく. *url* は対象の DBPedia Japanese のページの URL, *word* は *url* から 'http://.../' を取り除いたものである. 5.5 節で示す例において, 神奈川県の隣接都道府県を求めた際は, *url* が 'http://ja.dbpedia.org/resource/神奈川県', *word* が '神奈川県' となる.

アルゴリズム 1 は *Recursive Tree* に <*child_Leaf*> を追加するアルゴリズム, アルゴリズム 2 は, 新たな SSBasket クエリを生成する際, *Recursive Tree* を元に SSQL 部の GENERATE 句に SPARQL 問い合わせ結果を追加するアルゴリズムである.

Algorithm 1

Recursive Tree への <*child_Leaf*> の追加

```
1: function addLeaf(Leaf, parent_name, input_child_Leaf)
2:     if Leaf.csvfilename == parent_name then
3:         Leaf に input_child_Leaf を追加
4:         return
5:     else
6:         for all Leaf.child_Leaf do
7:             addLeaf(Leaf.child_Leaf, parent_name, input_child_Leaf
8:         end for
9:     end if
10: end function
```

Algorithm 2

Recursive Tree を基にした GENERATE 句への追加

```
1: function writeLeaef(Leaf)
2:     String result = ""
3:     if Leaf のホップ数 < Hop_Count then
4:         if Leaf.url が URL である then
5:             result+ =
6:                 "a('" + Leaf.word + "','" + Leaf.url + "')"
7:         else
8:             result+ = "'" + Leaf.word + "'"
9:         end if
10:    else if Leaf のホップ数 == Hop_Count then
11:        result+ =
12:            "[a(" + Leaf.csvfilename の alias + ", "
13:            + Leaf.csvfilename の alias + ")]"
14:            + "! " OR ", "
15:    ▷ 最下層のホップのみ, CSV を登録した SQLite のテーブルを
16:    参照し, その出力結果を SuperSQL の反復子を用いて表示する.
17:    end if
18:    return result
19: end function
```

5.3　SPARQL 問い合わせ

SSBasket では, Java パッケージの "org.apache.jena" を用いることで, SPARQL エンドポイントに SPARQL クエリを実行した結果を CSV として取得している.

初めに, 生成される中間 CSV の構成を以下に示す.

title は SPARQL クエリの SELECT 句の変数から "?" を除いた値であり, 中間 CSV を SQLite に登録する際にカラム名となる. value_1, value_2, ... value_n はクエリを実行した結果の各値で, 1 行が 1 タプルとして扱われる.

中間 CSV

```
title
value_1
value_2
...
value_n
```

次に, アルゴリズム 3 を示す. このアルゴリズムでは SSBasket クエリの SSQL 部の FROM 句に記述された SPARQL クエリを入力とし, 再帰的に SPARQL 問い合わせを行い, 中間 CSV を生成する. そして, 中間 CSV の各行の値と RE_SPARQL 部を基に新たな SPARL クエリを生成し, 再帰的に子要素の問い合わせを実行する.

Algorithm 3 再帰的な *SPARQL* 問い合わせ

1: Integer[Hop_Count+1] $nums$ ▷ 0 に初期化
2: Integer $now\_hop = 0$, $remember\_hop = 0$
3: **function** $executeQueries(input\_query)$
4: String $csvfilename = sparql\_result + now\_hop + nums[now\_hop] + .csv$
5: $input\_query$ を実行し, 中間 CSV ファイル: $csvfilename$ を生成
6: $Leafleaf$ ▷ $Leaf$ を新規生成
7: $leaf$ の最初の要素に $csvfilename$ を追加
8: **if** $now\_hop == 0$ **then**
9: $Recursive\_Tree$ を生成し,$leaf$ を追加 ▷ 最も親に位置する $first_Leaf$ となる
10: **else**
11: String $parent\_csvfilename = sparql\_result+(now\_hop-1) + nums[(now\_hop - 1)] + .csv$
12: $addLeaf(first\_leaf, parent\_csvfilename, leaf)$
13: **end if**
14: $nums[now\_hop] + +$
15: $remember\_hop = now\_hop$
16: **for all** $value$: $csvfilename.values$ **do** ▷ 中間 CSV の各行に対して処理を行う
17: $value$ と RE_SPARQL 部を基に new_query を生成
18: $executeQueries(new\_query)$
19: **end for**
20: $now\_hop = remember\_hop$
21: **end function**

5.4 動的な再問い合わせ処理

提案システムでは, 再問い合わせを 1 回実行する度に, 図 5 に示すパネルが表示される. このパネルは Java Swing で実装されており, Hop_Count, RE_SPARQL 部, RE_EXPRESSION 部を変更することができる. パネル表示時の初期値としては直前の問い合わせ時の各値が与えられる. これにより, 次回再問い合わせ時に異なる SPARQL クエリを発行したり, 新たなレイアウトを指定したりすることが可能となる.

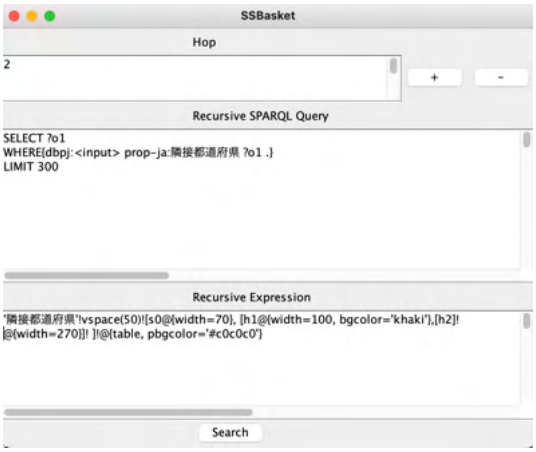

図 5 再問い合わせ用パネル

5.5 使 用 例

SSBasket クエリ例を次に示し, 実行結果を図 6 に示す.

SSBasket サンプルクエリ

```
SSBASKET(2){
SSQL{
  GENERATE HTML
  {
    ' 隣接都道府県'!  vspace(50)!
    [a.(s.related)]!@{table}
  }@{pbgcolor = 'beige'}
  FROM "#SPARQL(select distinct * where {
          dbpj:東京都 prop-ja:隣接都道府県 ?related . })" s
}
RECURSIVE_SPARQL{
  SELECT ?o1
  WHERE{dbpj:<input> prop-ja:隣接都道府県 ?o1 . }
  LIMIT 300
}
RECURSIVE_EXPRESSION{
  ' 隣接都道府県'!  vspace(50)!
  [s0@{width=70}, [h1@{width=100, bgcolor='khaki'},
  [h2]!@{width=270}]! ]!
  @{table, pbgcolor='aliceblue'}
}
}
```

本クエリ例では東京都を始点に隣接都道府県を表示している. 初回問い合わせでは, SSQL 部の WHERE 句に東京都の隣接都道府県を "?related" として求める SPARQL クエリを記述し, そのエイリアスを "s" とした. SuperSQL の仕様に従い, SSQL 部の GENERATE 句に "[s.related]! " と記述することで, 隣接都道府県が縦結合で表示される. このとき "anchor(s.related)", あるいは "a(s.related)" とすると "s.related" がリンクとして扱われる. また, "table" 装飾子を指定することで表の罫線が表示される.

図 6　実 行 結 果

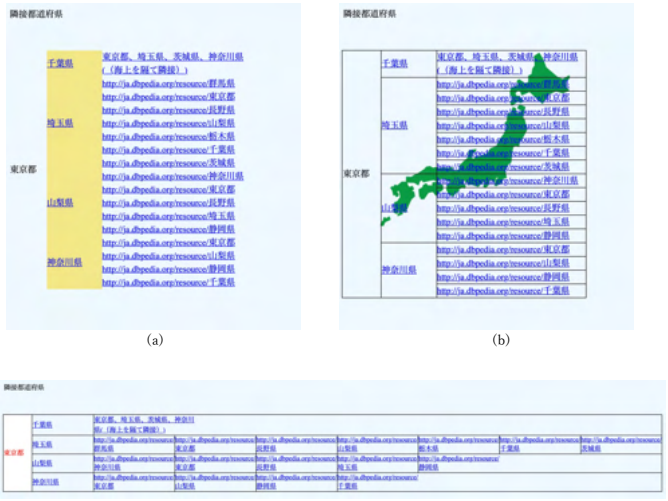

図 7　SSBASKET によるレイアウト変更

再問い合わせでは RE_SPARQL 部の “<input>”を初回問い合わせ時の “s.related” の各値に置き換え SPARQL をそれぞれ実行する．これを “SSBasket(*Hop_Count*)” で指定した *Hop_Count* ホップ目の値を得るまで繰り返し，RE_EXPRESSION 部で指定したレイアウトを用いて新たに SSBasket クエリを生成・実行し，HTML を出力する．RE_EXPRESSION 部の “s0, h1, h2” はそれぞれ各ホップ目の値を表し，“s0” は 0 ホップ目の主語，すなわち SSQL 部の WHERE 句に記述した SPARQL クエリの主語に該当する ‘東京都’ を表示することを指定する．

ここで，SSBasket クエリの装飾子や結合子，反復子の指定で自由にレイアウトを変更である．その例を図 7 に挙げ，図 6 の SSBasket クエリとの相違点となる RECURSIVE_EXPRESSION 部を示す．

図 7(a) は装飾子 “table” を指定しないことで，表の罫線を表示しない設定をした．図 7(b) は装飾子 “pbgimage” を用いることで背景に画像を挿入した．図 7(c) は “h2” の反復子を “[ ]!” から “[ ],” に変更したことで，2 ホップ目の値を水平に結合した例である．

6　評　　価

本章では，コード量の比較と，既存システムとの比較を行なった結果を示す．

6.1　コ ー ド 量

本節では，コード量の観点から、 SSBasket クエリと単体の SPARQL クエリ，SPARQL クエリ実行のためのプログラムの比較を行う．比較のために既存システムの d3sparql [12] を取り挙げる．ユーザーは D3.js [16] と，d3sparql ライブラリ内の d3sparql.js をインポートし，SPARQL クエリ実行結果を任意の形式の表やグラフで表示するプログラムを作成可能である．本評価では，d3sparql ライブラリに予め用意されている，表出力をする d3sparql-htmltable.html のコード量を比較対象とした．各比較対象の文字数，及び行数を数えた結果を以下に示す．

表 1　文字数と行数の比較

	文字数 (文字)	行数 (行)
SSBasket クエリ	390	20
SPARQL クエリ	73	5
d3sparql-htmltable.html	1738	52

表 2　SSBasket クエリのコード量

該当する生成 HTML の図	文字数 (文字)	行数 (行)
図 6	388	19
図 7(a)	382	21
図 7(b)	400	21
図 7(c)	399	19
平均	390	20

表 1 の SSBasket クエリの文字数と行数は，5.5 章で示した 4 つの SSBasket 実行例で使用した SSBasket クエリの平均である．各クエリのコード量の詳細は表 2 に記す．SPARQL クエリは，6.2 節の SPARQL 比較クエリである．表 1, SSBasket クエリ，に対し，SSBasket クエリの文字数は 22.4%，行数は 38.5% 削減された．また，単体の SPARQL クエリに対して，SSBasket クエリの文字数は 5.3 倍，行数は 4.0 倍となった．

上記が可能となったのは，SSBasket が SuperSQL を拡張したシステムだからである．SuperSQL は非手続き的であり，ユーザーは数行から数十行程度のクエリを記述することで，複雑な HTML を生成できる．単体の SPARQL クエリと比較すると，SSBasket クエリは数倍のコード量となった．しかしながら，SPARQL クエリ自体が数行であることを鑑みると，単体の SPARQL クエリと比較して増える記述量以上に，SSBasket クエリのみで HTML を生成できる労力の削減の方が遥かに大きいと考えられる．

6.2 既存システムとの比較

本節では, SSBasket と Apache Jena Fuseki(Fuseki), 及び, Endpoint browser との比較を行う. いずれのシステムも SPARQL 問い合わせ結果を基に HTML が生成される. 尚, 結果の表示は, Fuseki は表形式, Endpoint browser はグラフ形式で行われる.

6.2.1 表出力システムとの比較

下記の SPARQL クエリを Fuseki を用いてブラウザ上で実行した結果を図 8 に示す. このクエリは 5.5 で示した SSBasket のクエリと同様に, 東京都を起点に隣接等道府県を 2 ホップ分出力するものである.

SPARQL 比較クエリ

```
SELECT ?o1 ?o2
WHERE {
  dbpj:東京都 prop-ja:隣接都道府県 ?o1 .
  ?o1 prop-ja:隣接都道府県 ?o2 .
}
```

(a) (b)

図 8 Fuseki による表示

図 8 の (a) は Fuseki4.1.0 をダウンロードし, DBPedia Japanese から該当データをロードし, クエリを実行した結果, (b) は DBPedia Japanese の SPARQL エンドポイントにて, 直接クエリを実行した結果である. DBPedia Japanese の SPARQL エンドポイントは Fuseki を用いてデータを公開している. Fuseki の結果表示のフォーマットは図 8 に見られるように, 各システムで定まっている. ユーザーが任意のデザインで結果を表示させたければ, Fuseki のコードをカスタマイズする, あるいは, 結果をダウンロードし, 自作のプログラムにロードし表示させる必要がある. これにはプログラミングや CSS の知識が求められる. また, 入れ子形式で表示することができず, 対応する “?o2” の数分だけ, “?o1” が重複して出力される. 一方, SSBasket を用いれば, 図 7 に示したように, SSBasket クエリの装飾子や結合子, 反復子によって自由にレイアウトを指定することができる. 従って, SSBasket は従来の SPARQL 問い合わせ結果を表形式で表示するシステムと比べ, 出力レイアウトの表現力が高いことが示された.

6.2.2 グラフ出力システムとの比較

図 9 は Endpoint browser で以下の設定の基, “dbpj:東京都” を起点に表示させた結果である.

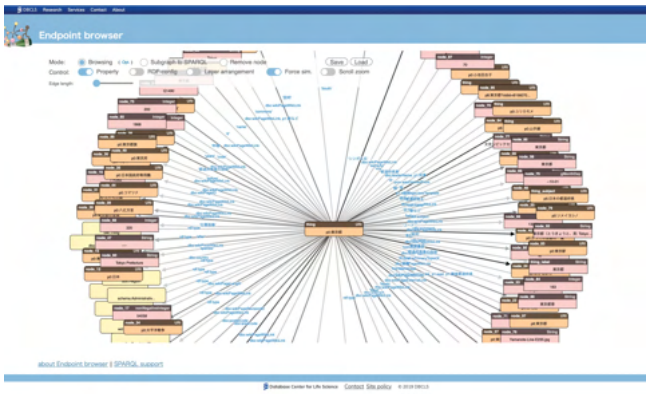

図 9 Endpoint browser による表示 A

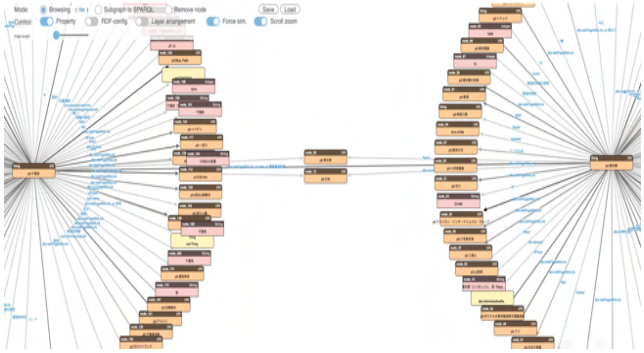

図 10 Endpoint browser による表 B

Endpoint browser の設定

```
Endpoint: https://ja.dbpedia.org/sparql
Start node: http://ja.dbpedia.org/resource/東京都
<Options> Limit of edge type: 1000
```

Endpoint browser による表示では, 視覚的に RDF データを探索することができる. しかしながら, Endpoint browser の出力グラフのノードやエッジ, ハイライトの色等はシステム依存である. この色を変更するためには, Endpoint browser のソースコードを編集する必要がある. また, ユーザーは GUI でノードの位置の変更, 不要なノードの削除, ノードのクリックによる, 図 10 に見られるような, RDF グラフの展開などが可能である. 更に, RDF グラフから対応する SPARQL クエリを取得することも可能である.

SSBasket では, 再問い合わせ用パネルを用いて, 任意の SPARQL クエリを再発行したり, レイアウトを変更したりすることができる. しかし, 意図する HTML 出力を得るために, SSBasket クエリをどのように変更するか考える必要があるため, Endpoint browser と比較すると, 直感的, 視覚的, 且つインタラクティブな操作性という点では劣ると言える.

ここで, Endpoint browser では起点となるノード (subject, あるいは object) の prediction が無数にある際, 利用するデータを抽出する作業が発生することがあり得る. これはデータ探索を斬新な関係を見つける意図で行う際には役に立つが, 予めデータを利用する方向性が定まっている場合には煩雑な過程と

なる．他方, SSBasket では SPARQL クエリを記述するため, クエリでトリプルパターンの各要素を変数にするか, 定数にするかによって, 目的に応じた探索が可能である．

6.2.3 比較結果

表 3　SSBasket, Fuseki, Endpoint browser の比較

	SSBasket	Fuseki	Endpoint browser
出力形式	表	表	グラフ
表現力	◎	△	○
多目的性	○	○	△
直感的操作性	△	△	◎
インタラクティブ性	○	△	◎

表 3 に示されるように, SSBasket は従来のシステムと比較し, 出力 HTML のレイアウトを自由に指定することができ, 出力レイアウトの表現力が優れている．また, SPARQL クエリを記述するため, グラフ出力のシステムに対して直感的操作性は劣るが, ユーザーは, LOD を探索の過程で利用したいデータを見つけることも, 予め意図していた検索をすることも容易であり, 多目的性を備えている．そして, 1 度の実行の中で再問い合わせパネルを活用することで, SPARQL クエリやレイアウトを変更することができ, インタラクティブ性がある．

総じて, SSBasket は SuperSQL を踏襲しており, コード量を削減して, LOD に対して SPARQL クエリを実行した結果を自由にレイアウトして HTML 表示可能であることが示された．

7 まとめ

7.1 結論

本論文では, SuperSQL を拡張し, LOD 問い合わせ言語 SPARQL に対応させたシステムである, SSBasket を提案した．

SSBasket の実装は, Java, Apache Jena を用いて行った．初めに SuperSQL の新たな入力として CSV, SPARQL を扱えるようにし, 次に SPARQL を用いた再帰的な問い合わせを実行できるよう対応させた．

評価としては, コード量の比較, 及び既存システムとの比較を行なった．コード量の観点では, SSBasket クエリと単独の SPARQL クエリを比較すると記述量は 4〜5 倍に増え, SSBasket クエリと SPARQL 問い合わせを行うためのプログラムと比較すると記述量は約 30%削減された．また, 既存システムとの比較では, SSBasket は, グラフ出力システムと比べて直感的操作性, インタラクティブ性は劣るが, 既存システムより出力レイアウトの表現力が優れていることが示された．ここで, 単体の SPARQL クエリは数行であるため, 単体の SPARQL クエリに対して増える記述量より, SSBasket を用いることで HTML を生成できる労力の削減の方が遥かに大きいと考えられた．

SSBasket を用いることでユーザーは HTML, CSS, その他プログラミングの知識がなくとも, 少ない記述量で SPARQL 問い合わせ結果を自由にレイアウト指定し, HTML で表示することが可能となった．

7.2 課題・展望

本論文の提案する SSBasket では DBPedia Japanese の SPARQL エンドポイントに対して SPARQL 問い合わせを行った．しかし SPARQL エンドポイントは WikiData [17], 英国図書館 [18], e-Stat [19] をはじめ, 国内外に数多く存在する．そのため, 問い合わせ可能な SPARQL エンドポイントを拡充する必要がある．また, 現在の提案システムでは表出力は可能だが, グラフ出力はできない．SuperSQL でデータから二次元グラフを生成する機能の研究 [20] [21] は行われており, これらを拡張し SPARQL に対応させることを検討する．他にも, ローカルデータベースと SPARQL 問い合わせ結果を統合して表示することや, 再問い合わせ処理の高速化等, システム改善の余地がある．

文　献

[1] C. Bizer, T. Heath, T. Berners-Lee: “Linked data the story so far. special issue on linked data”. IJSWIS’09: International Journal on Semantic Web and Information Systems, Vol.5, No.3, pp. 1-22, 2009.

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

[3] SPARQL: https://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/

[4] M. Toyama: “SuperSQL: An Extended SQL for Database Publishing and Presentation”. Proceedings of ACM SIGMOD’98 International Conference on Management of Data, pp. 584-586, 1998.

[5] SuperSQL: http://ssql.db.ics.keio.ac.jp

[6] Apache Jena Fuseki: https://jena.apache.org/documentation/fuseki2/

[7] A. Bonifati, W. Martens, T. Timm: “SHARQL: Shape Analysis of Recursive SPARQL Queries”. Proceedings of ACM SIGMOD’20 International Conference on Management of Data, pp. 2701–2704, 2020.

[8] Endpoint browser: https://sparql-support.dbcls.jp/sparql-support_j.html

[9] F. Haag, S. Lohmann, S. Bold, T. Ertl: “Visual SPARQL Querying Based on Extended Filter/Flow Graphs”. Proceedings of ACM AVI’14 International Working Conference on Advanced Visual Interfaces, pp. 305–312, 2014.

[10] P. Bottoni, M. Ceriani: “Linked Data Queries as Jigsaw Puzzles: a Visual Interface for SPARQL Based on Blockly Library”. Proceedings of ACM CHItaly’15: the 11th Biannual Conference on Italian SIGCHI Chapter, pp. 86–89, 2015.

[11] M. G. Skjæveland: “Sgvizler: A JavaScript Wrapper for Easy Visualization of SPARQL Result Sets”. The Semantic Web ESWC’12: Extended Semantic Web Conference on Satellite Events, pp. 361-365, 2012.

[12] d3sparql: https://biohackathon.org/d3sparql/

[13] http://openjena.org/

[14] DBPedia Japanese: https://ja.dbpedia.org

[15] SQLite: https://www.sqlite.org/index.html

[16] Data Driven Document: http://d3js.org

[17] Wikidata: https://www.wikidata.org/wiki/Wikidata:Main_Page

[18] British National Bibliography: http://search.bl.uk/primo_library/libweb/action/search.do?vid=BLBNB

[19] e-Stat: http://data.e-stat.go.jp/lodw/

[20] 大多和俊介, 五嶋研人, 遠山元道: “SuperSQL による構造化データの二次元可視化”. DEIM’19: データ工学と情報マネジメントに関するフォーラム, 2019.

[21] 大山 直輝, 五嶋研人, 遠山元道: “SuperSQL を用いた HTML 生成における特定構造生成関数の実装”. DEIM’20: データ工学と情報マネジメントに関するフォーラム, 2020.