LINQを用いたLinked Open Dataに対する問合せ

井上　寛之†　　天笠　俊之††　　北川　博之††

† 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
〒 305–8577 茨城県つくば市天王台 1-1-1
†† 筑波大学システム情報系情報工学域　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †inohiro@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし　コンピュータによる処理に適したデータの公開，共有，および利用を Web 上で行う方法の一つとして，Linked Open Data（LOD）が注目を集めている．近年，様々な種類のデータが LOD として公開され始めており，誰でも自由にアクセスすることができるため，LOD を利用するアプリケーションの開発が強く期待されている．一方，LOD の利用には，一般的に SPARQL と呼ばれる問合せ言語を用いるが，SPARQL の学習，記述にコストを要する．また，LOD は巨大で複雑なグラフ構造を構成し，必要なデータへのアクセスは容易とは言えない．本研究では，アプリケーションやサービス等から目的の LOD へのアクセスを容易にするために，JSON 形式のビューを提供するシステムを提案する．具体的には，データベース設計者等が予め LOD に対して JSON 形式のビューを定義する，これに対して，利用者は C#言語などから利用可能な LINQ 構文を用いて問合せを行う事を可能にする．

キーワード　Linked Open Data，ビュー，LINQ，JSON

1. はじめに

政府や行政などの公共機関が保有する様々なデータを，再利用可能な形式で外部へ公開する取り組みをオープンデータと呼び，近年先進国を中心に積極的に推進されている (注1)．無償でのアクセス，再頒布，再利用可能なデータをオープンデータと定義し (注2)，多くの人が自由に計算機などから利用できるように，機械可読性を有する標準化された形式で公開されることが推奨されている．このような構造化されたデータを Web 上で公開，共有する一つの方法として LOD [5] が注目されている．

LOD では，W3C によって標準化された Resource Description Framework（RDF） [7] を用いて Web 上のリソース自身と，他のリソースとの関係を記述する．リソース間のリンクは，HTML 文書間におけるハイパーリンクと同様であり，LOD では異なる領域のデータが互いにリンクすることで，様々なデータの活用を促進させる狙いがある．

オープンデータのカタログサイト Datahub においては，2014 年 1 月の時点で 263 の団体から，およそ 8,900 件のデータセットが LOD として公開されており，そのうち 882 個が LOD とされている (注3)．これらのデータセットはだれでも無償で利用することができるが，RDF を用いて記述された LOD は，SPARQL [14] と呼ばれる RDF 問合せ言語を利用して目的のデータを得るのが一般的である．ただし SPARQL の記述に際しては，問合せ言語の習得，LOD および RDF の関連技術についての知識が必要になる．また，RDF は本質的にはグラフ構造であり，LOD は複雑かつ冗長な構造になるため，これらの知識を持ち得ないデータ利用者が，LOD として公開されたオープンデータを利用するのは容易ではない．

そこで，本稿ではビューを用いた LINQ による LOD に対する問合せを提案する．LOD に対するビュー定義に関する研究についてもいくつか存在するが [1, 15]，本研究は LOD，RDF の知識を持たないユーザーによる LOD の利用を促進させる事を目的とする．そこで本手法は，予め LOD や RDF に知識のあるデータベース設計者らが定義した LOD に対する JSON ビューに問合せを行うことで，SPARQL を直接記述すること無く LOD への問合せを実現する．本論文では米マイクロソフト社が提供する.NET Framework の機能の一つである LINQ を用いて，C#等のプログラミング言語から LOD に対する問合せを実現する．

本研究で提案する手法は，大きく二つに分けられる．1）

（1） LOD や RDF に関する知識を有するデータベース設計者等が，LOD の特定の箇所を便利に利用することができるビューを定義する

（2） LOD や RDF に関する知識の無いデータ利用者が，定義されたビューを利用して，目的のデータに対して問合せ等の操作を行う

本研究では，ビューを用いた問合せを行うための Web API を検討する．またコレクションへの問合せ，集約処理に長ける LINQ を用いてこの Web API へリクエストすることで，既存のプログラミング環境から LOD への問合せを，特別な知識を必要とせずに実現する．

本稿の構成は以下のとおりである．2. 節では本論文に必要な基本的事項について解説する．3. 節では，ビューを用いた問合せ書き換えの手法について述べる．4. 節では，本手法の評価のための実験計画について説明する．5. 節で，本手法の関連研究を説明し，6. 節でまとめと今後の課題について述べる．

(注1)：http://www.mofa.go.jp/mofaj/gaiko/page4_000099.html
(注2)：http://opendefinition.org/
(注3)：http://datahub.io/dataset

2. 基本的事項

2.1 RDF

RDF [7] は，Web 上のリソース自身と，他のリソースとの関係を記述する枠組みである．RDF では，Universal Resource Identifier（URI，統一資源識別子）によって識別されるものすべてをリソースとして扱い，文書や画像だけでなく，人や施設，リソース間の意味までもリソースとして扱われる．あるリソースの一つのメタデータは，主語（Subject），述語（Predicate），および目的語（Object）によって記述される．主語はメタデータを記述する対象のリソース，述語は主語に関する情報のプロパティもしくは特徴を定義し，目的語は述語の対象となる値を格納する，これはら三つ組でトリプルと呼ばれる．主語及び述語は URI で，目的語は URI もしくはリテラルで記述し，そのデータ型や言語に関する情報を付与することも可能である．RDF グラフを図示する場合は，リソースを楕円で，リテラルは句形で表す．

2.2 LOD

LOD は構造化されたデータを Web 上で公開，共有する一つの方法であり，データ同士をリンクさせることで「データのウェブ」を構築する運動そのものを指す．また，LOD は標準化された RDF や HTTP などから構成される，データセットの集合である．LOD として公開されるデータは原則として，以下の 4 原則が定義されている (注4)．

（1） URI をあらゆるデータの識別子として利用する

（2） 識別子には HTTP スキームの URI を利用し，アクセス可能にする

（3） URI にアクセスされた際には有用な情報を，RDF などの標準化された形式で提供する

（4） データに他の情報源における URI を含め，より多くの情報発見を促進する．

2.3 SPARQL

RDF の集合である LOD に対する問合せ言語として，*SPARQL* [14] が W3C によって標準化されており，近年デファクトスタンダードとして幅広く利用されている．SPARQL 処理系はクエリからグラフパタン（Basic Graph Pattern，BGP）を構築し，そのパタンに一致するリソースを RDF 集合の中から探し出す．最新版である SPARQL 1.1 では，単純な選択にとどまらず，BGP に適合する問合せ結果から新たな RDF を生成する機能（`CONSTRUCT` 句）や，集約演算などがサポートされている．

SPARQL クエリの基本的な構造を表 1 に示す．SPARQL クエリにおいて，名前空間宣言が複数用いられる場合は `PREFIX` 句で複数宣言する．SPARQL において，変数は “?” 文字から始めることで宣言できる．この変数を使って見つけ出すグラフパタンを Turtle 形式で `WHERE` 節に記述し，選択条件を `SELECT` 句で記述する．`WHERE` 句ではそれぞれの変数について `FILTER` 句を用いて条件を記述することができる．条件の種類としては，

(注4)：http://www.w3.org/DesignIssues/LinkedData.html

表 1: SPARQL クエリの基本構造

節	キーワード	説明
接頭辞宣言	PREFIX	名前空間接頭辞宣言
データセット定義	FROM	探索対象のグラフ IRI（URI）
結果節	SELECT	指定した変数の値を返す
	CONSTRUCT	指定した構造で RDF を作り，返す
	ASK	指定したパタンの存在を有無を返す
	DESCRIBE	指定したパタンを含む DB 内のグラフを返す
クエリパタン	WHERE	グラフパタンの指定
	FILTER	条件式
クエリ修飾子	ORDER BY (DESC)	順序指定
	LIMIT	取得件数指定
	OFFSET	オフセット指定
	DISTINCT	重複の除去

1）存在（`EXISTS`，`NOT EXIST`），2）比較演算子（“=”，“>=” など）が利用可能である．

2.3.1 SPARQL エンドポイント

LOD に対して HTTP リクエストによって直接 SPARQL を実行するフロントエンドを *SPARQL* **エンドポイント**と呼ぶ．SPARQL エンドポイントはトリプルストア（RDF ストア）が提供する機能であり，Virtuoso (注5) や Fuseki (注6) などが挙げられる．Web 上のパブリックな SPARQL エンドポイントについて，その可用性をチェックしている “SPARQL Endpoints Status” (注7) によれば，全世界に 457 の SPARQL エンドポイントが存在する．

多くの SPARQL エンドポイントが，問合せの結果として様々な形式に対応している．主に，XML（XHTML），CSV，TSV，JSON などが挙げられる．また問合せの種類によって利用可能な形式が限られる場合がある．これは結果節でのキーワードの指定に依存し， `SELECT` 句は RDF グラフではなく指定された変数の値について結果を返すが， `CONSTRUCT` 句を指定した場合は指定されたグラフ構造に結果を RDF として整形して返すためである．

統合言語クエリ（Language-Integrated Query, LINQ） (注8) はマイクロソフト社の.NET Framework が提供する，様々な情報源から得られる集合に対して統一的な方法で問合せ等の操作を行う，プログラミング言語に統合された機能である．LINQ を用いた構文では，SQL や SPARQL のように宣言的に集合に対して操作を記述することができる．

LINQ の利点として，以下が挙げられている (注9)．

（1） 簡潔で読みやすい（特に複数の条件をフィルター処理する場合）

（2） 強力なフィルター処理，並び替え，およびグループ化機能を最小限のアプリケーションコードで実現できる

（3） ほとんど変更せずに，他のデータソースに移植できる

LINQ は主に `C#`，Visual Basic .NET などのプログラミング言語で利用することができるが，Java，JavaScript，PHP，Pytho などのプログラミング言語で LINQ の構文を実現する試みは数多く存在する．.NET Framework 標準の機能では，文字

(注5)：http://virtuoso.openlinksw.com/

(注6)：http://jena.apache.org/documentation/serving_data/

(注7)：http://sparqles.okfn.org/

(注8)：http://msdn.microsoft.com/en-us/library/bb308959.aspx

(注9)：http://msdn.microsoft.com/ja-jp/library/bb397919.aspx

列型や数値型のコレクション，RDB のレコードや XML 文書の要素集合などに対して SQL に似た構文で問合せ，集約などの操作を行うことができる．また，独自にプロバイダを実装することで，任意の情報源に LINQ の構文でアクセスすることができる．

2.4 JSON (JavaScript Object Notation)

JavaScript Object Notation（JSON）はデータ記述言語の一つであり，JavaScript 言語におけるオブジェクト表記方法として定義され，標準化された [10]．近年では，複数のフトウェアや他のプログラミング言語間でのデータのやり取りに，広く利用されている．JSON はデータ型として，数値，文字列，真偽値，配列，オブジェクト（連想配列），null を利用することができる．オブジェクトのキーの値は文字列である必要がある．

2.4.1 JSON スキーマ

JSON スキーマ (注10) は JSON で任意のデータを記述するためのスキーマ定義を行うものである．要素の名称，データ型，および中身の構造などについて JSON を用いて定義する．JSON スキーマは，現在 IETF による標準化プロセスにある (注11)．

3. ビューを用いた LINQ による LOD に対する問合せ

本節では，ビューを用いた LINQ による LOD への問合せについて説明する．

3.1 概　　要

本研究で提案するビューを用いた LINQ による LOD に対する問合せについて，概要図を図 1 に示す．本研究は，図中のクエリトランスレータを導入することで，アプリケーションから LOD に対して直接 SPARQL を発行する必要を無くす．代わりに，予め LOD や RDF に関する知識をもつデータベース設計者等によって定義されたビューを利用し，LINQ による LOD への問合せを可能にする．このため，予め JSON 形式のビューを与えておき，LINQ 問合せの変換に用いる．本手法を実現するために，以下の二つの項目について検討する必要がある．

（1） LOD に対する JSON ビューの定義

（2） 定義されたビューを用いた問合せ

これはそれぞれ，図 1 内の（1），(2）に対応する．(1）LOD に対する JSON ビュー定義は 3.2 節で，(2）ビューを用いた LOD への問合せについては 3.3 節でそれぞれ述べる．また実際に LINQ を用いて問合せを行う例を 3.4 節に示す．本手法の実装については 3.5 節で説明する．

3.2 LOD に対する JSON ビューの定義

LOD に対する JSON ビューは，LOD や RDF に関する知識を持つデータベース設計者等によって，特別な知識を持たないアプリケーション開発者のために予め定義される．ビューの定義には，SPARQL クエリおよび JSON スキーマを基に行う．

SPARQL によるビュー定義の例を示す．図 2 は RDF グラフの例である．このグラフには，ある人（“Person_1”）の姓

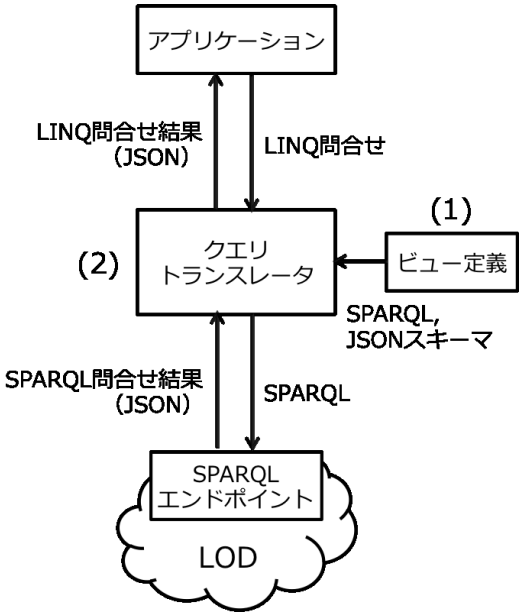

図 1: 提案モデル

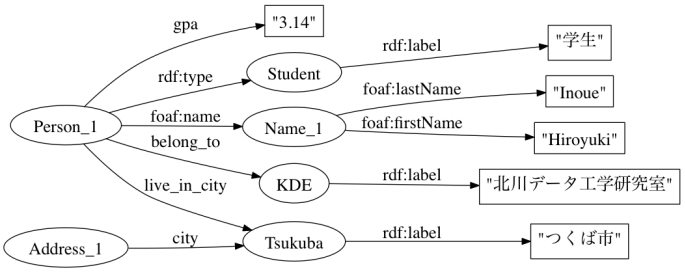

図 2: RDF グラフ例

表 2: ビュー定義クエリの構造

節	キーワード	説明	必須項目
接頭辞宣言	PREFIX	名前空間接頭辞宣言	いいえ
データセット定義	FROM	探索対象のグラフ IRI（URI）	いいえ
結果節	SELECT	指定した変数の値を返す	はい
クエリパタン	WHERE	グラフパタンの指定	はい
	FILTER	条件式	いいえ
クエリ修飾子	ORDER BY (DESC)	順序指定	いいえ
	LIMIT	取得件数指定	いいえ
	OFFSET	オフセット指定	いいえ
	DISTINCT	重複の除去	いいえ
JSON スキーマ	AS JSON	結果の JSON スキーマを指定	はい

（“last”），名（“first”），身分（“rdf:type”），所属研究室（“belong_to”），住んでいる街（“live_in_cit”）が含まれている．このグラフから，実アプリケーションで利用するような文字列型，数値型の値を得るにはグラフを辿る必要がある．そこで，ここではこのグラフが持つ文字列型，数値型の値が RDB のタプルのように取得可能なビューを検討する．データベース設計者が目的の値を選択する SPARQL クエリ，および結果のテンプレートとなる JSON スキーマを記述し，ビューとして登録する．本研究で提案する，ビュー定義に用いる拡張された SPARQL クエリの構造を表 2 に示す．

3.2.1 JSON スキーマによる結果フォーマットの定義

ビュー定義にはデータ利用者が最終的に受け取る JSON の構造が指定される．具体的には選択を行う SPARQL クエリに続けて， AS JSON 句に JSON スキーマを記述する．SPARQL に

(注10)：http://json-schema.org/

(注11)：http://json-schema.org/latest/json-schema-core.html

```
SELECT ?last ?first ?position ?labname ?cityname ?gpa
WHERE {
    ?person a               Student ;
            foaf:name       ?name ;
            live_in_city    ?city ;
            belong_to       ?lab ;
            gpa             ?gpa .
    Student rdf:label       ?position .
    ?name   foaf:firstName ?first ;
            foaf:lastName   ?last .
    ?city   rdf:label       ?cityname .
    ?lab    foaf:name       ?labname . }
AS JSON {
{  "type":"object", "properties":{
    "results": {
      "type":"array", "items":{
        "type":"object", "properties":{
          ?last:     { "type":"string" },
          ?first:    { "type":"string" },
          ?position: { "type":"string" },
          ?labname:  { "type":"string" },
          ?cityname: { "type":"string" },
          ?gpa:      { "type":"number" }
}}}}}}
```

リスト 1: ビュー定義クエリの例（PREFIX は省略）

```
{ [ { "last": "Inoue", "first": "Hiroyuki",
      "position": "M2","labname": "北川データ工学研究室",
      "cityname": "Tsukuba", "gpa": "3.14" },
    { "last": "Tsukuba", "first": "Taro",
      "position": "M1", ... }
] }
```

リスト 2: リスト 1 から得られる JSON の例

おいて変数は，“?position” といったような “?” 付き文字から始まるが，SPARQL エンドポイントから得た値は JSON スキーマにおける同一の変数名に展開される．JSON の特徴を活かした，任意の階層構造，多値を配列で表現するといった結果フォーマットの定義が可能である．リスト 1 に図 2 の RDF グラフに対するビュー定義の例を示す．また，リスト 2 にビュー定義 1 から得られる JSON の例を示す．

3.3 ビューを用いた問合せ

提案手法における LINQ 問合せの処理方法を説明する．

データベース設計者によって定義されたビューに対して，アプリケーション開発者（データ利用者）が LINQ 問合せを行う．与えられた問合せを構文解析することで，ノードがクエリ演算子，エッジが演算子間の依存関係を表す式木を得ることができる．LINQ 問合せから式木への変換については，本研究の対象から外れるためここでは詳しく議論しないが，.NET の構文解析木を利用することで式木を得ることができるため，これを利用している．

表 3: 本手法が対応する.NET Framework 標準クエリ演算子

キーワード	説明
Select	射影演算子
Where	選択演算子
GroupBy	グループ化演算子
OrderBy	順序付け演算子
Min	最小値演算子
Max	最大値演算子
Sum	合計値演算子
Count	集計演算子
Average	平均値演算子

得られた式木を元に，SPARQL エンドポイントに対する問合せを生成する．基本的には，式木のリーフに当たるデータソースが対象とする JSON ビューである場合，対応する問合せをそのまま SPARQL エンドポイントで処理することで LINQ 問合せが必要とする情報が得られる．これを定義された JSON ビューに従わせ，LINQ 処理系に返すことで結果を得ることが可能である．ただし以下のような場合，対応する処理を SPARQL エンドポイント上で実行することにより，処理の効率化が図れる場合がある．

データソースに対して，絞り込み（選択），射影，グルーピング等の条件が存在する場合，これらをデータソースに対する SPARQL 問合せへプッシュダウンすることで，SPARQL エンドポイント上で処理が可能である．この場合，結果としてネットワーク上を転送されるデータ量が減るため，その分のコストも抑えることができる．一方，SPARQL エンドポイントの処理性能やレスポンスが遅い場合もあり，そのときは多くの処理をなるべく LINQ 処理系の側で実行した方が効率的な場合もある．この観点から，適切なコストモデルを設定して，コストベースの問合せ最適化を行うことが考えられるが，これは今後の研究課題である．

具体的に，式木から SPARQL エンドポイントに対する問合せの生成について説明する．LINQ 問合せにおける演算子について，本手法が対応する演算子を表 3 に示した．これらは式木においてそれぞれの式の演算子として利用されるため，式の解析が必要となる．式は種類ごとに異なるクラスで表現されるため，本手法では以下の 4 つのクラスに注目して情報を得る．

MethodCallExpression **クラス**
静的メソッド，もしくはインスタンスメソッドの呼び出しを行う式．クエリ演算子はこのクラスの式に含まれる． `Method` プロパティが演算子（メソッド），`Arguments` がその式の被演算子（引数）である．

NewExpression **クラス**
コンストラクタメソッドの呼び出しを行う式であるが，LINQ においては射影の結果を格納する無名クラスを作成するときに作成される．複数変数の射影として扱う．

```
var context = new Context( "http://example.com/view/1/" );
var labnames = from person in context
               where person["first"] == "Hiroyuki"
               select person["labname"];
```

リスト 3: LINQ による問合せの例

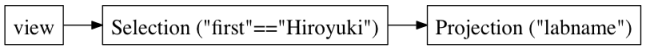

図 3: リスト 3 の処理木

BinaryExpression **クラス**
二項演算子をもつ式．条件式として用いられる．演算子と，左右の被演算子を持つ．

ConstantExpression **クラス**
定数を表現する式．条件式で用いられる． `Value` プロパティが定数である．

式木の解析及び問合せ書き換えアルゴリズムを Algorithm 1 に示す．問合せ対象のビューは LINQ 問合せ時にビューごとの URI にリクエストされるものとし，式木の解析時には既に書き換え対象のビューは認識されているとする．まず， *ExtractExpressions* は入力された式から再帰的に演算子や条件を抽出する． *ExtractMethodCallExpression* は MethodCallExpression クラスの式から演算の種類や対象となる変数，条件式等を探し出し，射影の場合は変数が揃い次第 `SELECT` 句の書き換えを行う．選択の場合は条件式を二項演算の演算子と変数を取得する *ExtractBinaryExpression* を用いて，揃い次第 `FILTER` 句を `WHERE` 句に書き加える．集合演算の場合は， `SELECT` 句，`FILTER` 句， `ORDER BY` 句で利用される可能性があるため，書き換えは行わず，演算子の種類と変数名を返す．グループ化演算子，もしくは順序演算子の場合は変数が揃い次第 `WHERE` 句の後に書き加える． *ExtractConstantExpression* は与えられた式から定数を見つけ出す． 3.4 節で具体的な問合せの例を示す．

3.4 LINQ を用いた問合せの例

アプリケーション開発者（データ利用者）は LINQ クエリを記述する際に利用するビュー URI を入力として，コンテキストを作成する．作成されたコンテキストは指定されたビュー URI に対して LINQ 問合せを行う．リスト 3 は ViewURI（`http://example.com/views/1/`）で，リスト 1 で定義されたビューが利用可能であるとして，“first” 属性が “Hiroyuki” であるようなリソースの研究室名（“labname”）を取得する LINQ 問合せである．この問合せは，選択および射影を行うので，LINQ コンテキストでは図 3 に示す式木が組み立てられる．これに基づいた SPARQL クエリ生成がクエリトランスレータによって行われ，最終的にリスト 4 に示すクエリを SPARQL エンドポイントに問合せる．また結果の JSON としてリスト 5 がアプリケーションに返される．

3.5 実　　装

本研究では提案手法である LOD に対するビュー定義とクエリ書き換えを行うクエリトランスレータ，及びアプリケーションからビューに対する条件を組み立て，クエリトランスレータへリクエストする LINQ プロバイダの実装を行った．それぞれ説明する．

3.5.1 クエリトランスレータ

クエリトランスレータの役割は，1）ビューの管理，2）ビューと条件に基づいたクエリ書き換え，および LOD へのリクエストである．

（1）ビューの管理について．クエリトランスレータは，ビュー

Algorithm 1 式木の解析および問合せ書き換え

```
procedure ExtractExpression(expression)
    if expression.type is MethodCallExpression || NewExpression then
        method ← expression.method.name
        for all ( doargument ∈ expression.arguments)
            ExtractMethodCallExpression(argument)
        end for
    end if
    if expression.type is BinaryExpression then
        ExtractBinaryExpression(expression.arguments)
    end if
    if expression.type is ConstantExpression then
        ExtractConstantExpression(expression.arguments)
    end if
end procedure
procedure ExtractMethodCallExpression(expression)
    condition ← ∅
    if expression.method is 'Select' then
        condition.method ← 'Projection'
        condition.variable ← ExtractExpression(expression.argument)
        ReWriteSelectStatement(condition)
    end if
    if expression.method is 'Where' then
        condition.method ← 'Selection'
        condition.conditions ← ExtractBinaryExpression(expression.argument)
        AddFilterCondition(condition)
    end if
    if expression.method is 'Min' || 'Max' || 'Sum' || 'Count' || 'Average'
then
        condition.method ← the method which was detected
        condition.variable ← argument.variable
        Return condition
    end if
    if expression.method is 'GroupBy' || 'OrderBy' then
        condition.method ← 'GroupBy' || 'OrderBy'
        condition.variable ← ExtractExpression(expression.argument)
        AddGroupOrOrderCondition(condition)
    end if
     return ∅
end procedure
procedure ExtractBinaryExpression(expression)
    conditions ← ∅
    condition.operator ← expression.operator
    condition.left ← EcstractExpression(expression.left)
    condition.right ← ExtractExpression(expression.right)
    Return condition
end procedure
procedure ExtractConstantExpression(expression)
    Return Cleansing(expression.value)
end procedure
```

```
SELECT ?labname
WHERE {
    <-- グラフパタン -->
    FILTER (str(?first) = "Hiroyuki")
}
```

リスト 4: LINQ 問合せ 3 から生成された SPARQL

```
{ [ { "?labname": "北川データ工学研究室" } ] }
```

リスト 5: リスト 4 の結果（JSON）

URI，タイトル，概要，ビューを用いることで得られる属性とそのデータ型を保存する．

（2）クエリトランスレータの実装について．クエリトランスレータは Ruby 言語およびその Web フレームワークである Sinatra (注12) を用いて実装した．LINQ プロバイダからのアクセスは HTTP で発行され，URL パラメータとしてパーセントエンコーディング (注13) された LINQ 問合せ条件を受け取る．

クエリトランスレータは “LodViewRewrite” (注14)．として，LINQ 問合せを受け付ける Web アプリケーションは “LodViewWebApp” (注15)．として，それぞれオープンソースで開発されている

3.5.2 LINQ プロバイダ

LINQ プロバイダの実装は一般的に行われている `System.Linq.IQueryable<T>` インタフェースおよび，`System.Linq.IQueryProvider` プロバイダの実装を行うことで実現した．

実装した機能としては，射影，選択，集約演算の一部である．複雑な `GroupBy`， `Having`（グループ化後の選択）や複数情報源にまたがった結合演算（`JOIN`）はサポートしていない．これらのサポートは今後の課題である．

実装においては，“LINQ: Building an IQueryable provider series” (注16) や，“LINQ to Twitter” (注17) の実装を参考にした．

LINQ プロバイダは，“LodViewProvider” という名称で，オープンソースで開発されている (注18)．

4. 評価実験

提案手法性能を調査するための実験による評価を行ったので，その結果を報告する．

実験の目的は，提案手法が実際に実用可能な性能で動作することを実環境で確認することと，ビューを扱うためのオーバーヘッドを実機で確認することにある．このため，提案手法に加えて，LINQ 問合せと等価な SPARQL 問合せを SPARQL エンドポイントに投入し，両者の実行時間を比較する．

提案手法は，Windows 7 に実装され，ビューに対する LINQ 問合せを受け付ける．この問合せは，Mac OS X 上のクエリトランスレータに Web API 経由で転送され，ビュー定義に基いて SPARQL クエリが生成される．生成された SPARQL 問合せは，SPARQL エンドポイントへのリクエストとして送信される．SPARQL エンドポイントには，Fuseki (注19) を用いた．Fuseki は，Java で RDF を扱うライブラリである Apache Jena の成果物の一つであり，既存の RDF ファイルを用いて SPARQL エンドポイントを立ち上げることができる．

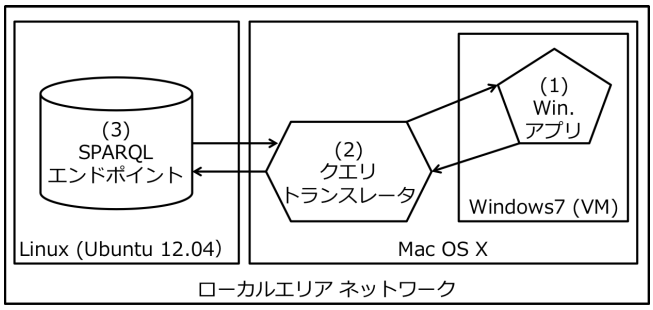

図 4: 実験環境

```
SELECT DISTINCT ?label ?value
WHERE {
  ?product a bsbm-inst:ProductType105 ;
           rdfs:label ?label ;
           bsbm:productPropertyNumeric1 ?value .
}
AS JSON {
{  "type":"object", "properties":{
    "results": {
      "type":"array", "items":{
        "type":"object", "properties":{
          ?label: { "type":"string" },
          ?value: { "type":"number" }
}}}}}
```

リスト 6: ビュー定義（PREFIX は省略）

実験環境を図 4 に示す．図中の（1）-（3）はそれぞれ，(1) LINQ 問合せを行う Windows アプリケーション (注20)，(2) クエリトランスレータ (注21)，(3) Fuseki (注22) である．Windows アプリケーションとクエリトランスレータは同一のコンピュータで，Fuseki は別のコンピュータにそれぞれ設置され，全ての通信は HTTP で行われる．

本実験で利用したデータセットは，SPARQL エンドポイント（およびトリプルストア）のベンチマークを行うプロジェクトである Berlin SPARQL Benchmark [6] が開発する，“BSBM Tools” (注23) を利用して作成した．このツールは製品のメタデータや製造者に関するテストデータを生成する．生成するトリプル数や出力形式を柔軟に指定することが可能である．この RDF データに対して，リスト 6 に示す定義に従って JSON ビューを構築した．

LINQ による JSON ビューに対する問合せ，およびそれと等価な SPARQL 問合せの実行時間を比較する．LINQ 問合せにおいては，それぞれ 1）式木の分析（Analyze），2）SPARQL エンドポイントへのリクエストおよび結果の受取り（Request），3）受取った結果の整形（Format）について時間を計測した．一方，SPARQL 問合せの場合は，SPARQL エンドポイントへ

(注12)： http://www.sinatrarb.com/
(注13)：http://tools.ietf.org/html/rfc3986\#section-2.1
(注14)： https://github.com/inohiro/LodViewRewrite/
(注15)： https://github.com/inohiro/LodViewWebApp/
(注16)：http://blogs.msdn.com/b/mattwar/archive/2008/11/18/linq-links.aspx
(注17)：http://linqtotwitter.codeplex.com/
(注18)： https://github.com/inohiro/LodViewProvider/
(注19)：http://jena.apache.org/documentation/serving_data/

(注20)：Windows 7 (32bit, VM), 2GB RAM, .NET Framework 4.5
(注21)：Mac OS X, Core i7 2.4 GHz, 8GB RAM, ruby 2.0.0
(注22)：Ubuntu Server 12.04, Core i5 2.8GHz, 6GB RAM, java 1.7
(注23)：http://sourceforge.net/projects/bsbmtools/

```
var query = from resource in context
            where Int32.Parse( resource["value"] ) < 400
            orderby resource["label"]
            select resource;
```

リスト 7: LINQ 問合せ

```
SELECT DISTINCT ?label ?value
WHERE {
  ?product a bsbm-inst:ProductType29 ;
           rdfs:label ?label ;
           bsbm:productPropertyNumeric1 ?value1 .
  FILTER (?value < 400 )
}
ORDER BY ?label
```

リスト 8: SPARQL 問合せ（PREFIX は省略）

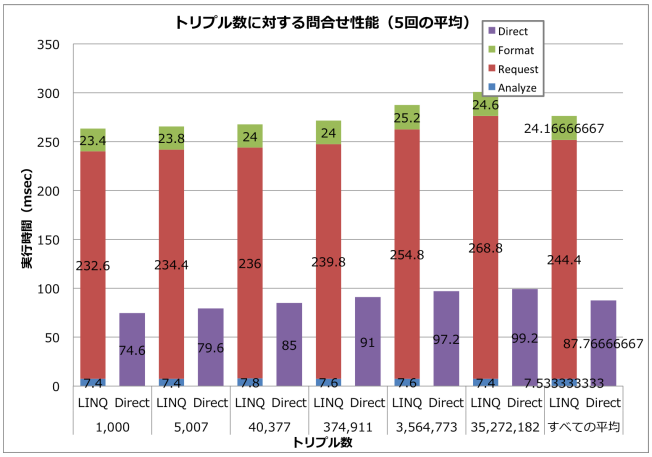

図 5: トリプル数に対する問合せ性能（それぞれ 5 回の平均）

のリクエストを開始時点から結果を受け取るまでの時間を計測した．

4.1 トリプル数に対する問合せ性能

使用した LINQ 問合せおよび対応する SPARQL 問合せはそれぞれリスト 7，リスト 8 の通りである．データセットはそれぞれ “5,007”，“40,377”，“374,911”，“3,564,773”，“35,272,182” 個のトリプルを含むデータを前述の RDF ジェネレータで生成した．

図 5 に，トリプル数ごとに計測を 5 回ずつ行って得た結果の平均値を示す．横軸，縦軸はそれぞれ，トリプル数の変化，実行時間（msec）を表している．提案手法であるビューを用いた LINQ による問合せは平均的に，SPARQL を直接実行した場合のおよそ 3 倍程度の実行時間で問合せを行えることがわかった．具体的には平均して，式木の分析（Analyze）に 7.53 msec，SPARQL エンドポイントへの問合せおよび結果の取得（Request）に 244.4 msec，結果の整形（Format）に 24.1 msec かかり，合計すると 276.03 msec である．一方，SPARQL を直接リクエストした場合は， 87.7 msec かかっている．トリプル数が変化しても，実行時間に大幅な変化は見られなかった．

表 4: 実験を行った問合せの種類と条件

種類	条件
射影	名称（“label”）だけ得る
最大値	価格（“value”）の最大値を得る
最小値	価格（“value”）の最小値を得る
平均値	価格（“value”）の平均値を得る
並び替え	価格（“value”）について昇順に並び替えた結果を得る
選択と射影	価格（“value”）が “400” 以下の製品の名称（“label”）だけ得る
選択と平均値	価格（“value”）が “400” 以上の製品の価格の平均値を得る

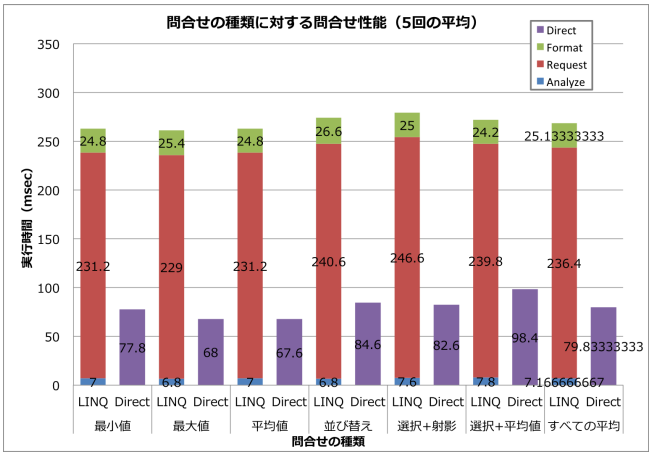

図 6: 問合せの種類に対する問合せ性能（それぞれ 5 回の平均値）

4.2 問合せの種類に対する問合せ性能

問合せに含まれる関係演算や問合せの複雑度が問合せ性能に与える影響を調査した．

本実験では，SPARQL エンドポイントが持つトリプル数を “3,564,773” 個に固定して行った．具体的に実験を行った演算とその条件について表 4 に示す．

図 6 に演算ごとに 5 回計測を行って得た結果の平均値を示す．横軸，縦軸はそれぞれ演算の種類，実行時間（msec）を表す．いずれの演算子においても，本研究が提案する LINQ 問合せから SPARQL 問合せを生成する方法は，直接 SPARQL 問合せを行う場合のおよそ 3 倍程度の実行時間になることが示された．具体的には平均して，式木の分析（Analyze）に 7.16 msec，SPARQL エンドポイントへの問合せおよび結果の取得（Request）に 236.4 msec，結果の整形（Format）に 25.1 msec，合計すると 268.66 msec かかっている．一方，SPARQL を直接リクエストした場合は，79.83 msec である．

5. 関 連 研 究

5.1 LOD，RDF に対するビューの導入

RDF の集合に対するビューの導入に関する研究は LOD の概念が登場する前から行われている [8, 11, 13, 15, 16]． [8] および [11] は双方とも 2005 年の論文であり，現在の仕様とは異なる SPARLQL を対象に，前者はスキーマレベルのビュー定義を提案している．後者は RDB に格納された RDF に対して集約演算を独自に開発した． [16] はセンサ等から得た時系列 RDF データに高速にアクセスするための一時的なビューの設計に関する研究である．本手法が LOD を利用しようとするアプリケーションに対して，複雑性を除去して利用させる点で類似する．ただし SPARQL による問合せを想定している．

Le ら [15] や，Etcheverry ら [12] の研究は，LOD に対して SPARQL クエリを用いてビュー定義を行う点で本研究と類似するが，彼らの定義するビューは CONSTRUCT 句を用いた，RDF を生成するビューである．一方，本研究が提案するビューは LOD への問合せを簡単化する目的の JSON ビューであり，ビューの定義と研究の目的が異なる．

RDB に格納されたデータに対して RDF としてアクセスするためのビュー定義については数多くの研究，およびソフトウェアとしての実装が存在する [2–4, 9]．これらの手法は仮想的な RDF 集合をビューとして作り出し，SPARQL クエリを SQL に書き換えて問合せを行う．対象がリレーショナルデータである点が，本研究とは異なる．

5.2 LINQ を用いた LOD へのアクセス

[1] は.NET 言語向けのライブラリ “LINQ to RDF” (注24) を用いて，Web 上の LOD に対して LINQ 問合せによる SPARQL クエリの発行について説明している．この手法では，取得対象となるリソースがどのような属性を持つグラフ構造であるかをアプリケーション開発者が把握して，これに合わせた問合せ結果を格納するクラスを実装することで LINQ を用いた問合せを実現している．一方，本研究ではビューを定義することで LOD のリソース集合を抽象化し，アプリケーション開発者は特定のビューに対する LINQ 問合せを記述するだけで，LOD のグラフ構造を知ること無く，LOD に対する問合せを行うことができる点が異なる．

6. ま と め

本稿では，ビューを用いた LINQ による LOD に対する問合せ手法の提案を行った．本手法では，RDF，LOD および SPARQL の知識を持たないアプリケーション開発者が，データベース設計者等によって定義された JSON ビューを利用することで，直接 SPARQL を記述すること無く，LOD への問合せを行うことを実現した．

本手法が実用的な速度で動作することを確認する実験では，直接 SPARQL 問合せを行うのと比べ，サポートする全ての演算でおよそ 3 倍前後の処理時間で問合せを行えることがわかった．また，トリプル数の変化に影響されない事を示した．

今後の課題としては以下が挙げられる．

（1） クエリトランスレータから SPARQL エンドポイントに対する問合せ（Request）の高速化

（2） 射影条件が変わった場合の，結果となる JSON のコンパクション

- 例えば，LINQ 問合せの射影条件が一つの変数である場合，JSON スキーマによって定義された結果テンプレートに従うのではなく，配列等を用いたより効率的な構造へのコンパクションなどが考えられる

（3） 複雑な射影演算（HAVING 句）の実現

（4） コストモデルに基づいた問合せ最適化

- 具体的には，問合せの複雑性，選択率，および問合せ先 SPARQL エンドポイントの性能を考慮する

（5） 複数情報源を跨いだ結合演算（JOIN 句）の実現

- 結合演算は，1）LINQ 側で行う場合，2）クエリトランスレータで行う場合，および 3）SPARQL 問合せとしてエンドポイントで行う場合の三つの方法が考えられるため，3 に示したコストモデルに基づいて最適な処理系で演算を行う

(注24)：http://code.google.com/p/linqtordf/

謝　　辞

本研究の一部は，共同研究（富士通研究所 CPE25149），JSPS 科研費（25240014），および文部科学省委託業務「ビッグデータ利活用のためのデータ連携技術に関するフィージビリティスタディ及び予備研究」による．

文　　献

[1] Exploiting the RDF-based Linked Data Web using .NET via LINQ. http://virtuoso.openlinksw.com/whitepapers/rdf%20linked%20data%20dotNET%20LINQ.html.

[2] RDF Support in the Virtuoso DBMS. volume 113 of *LNI*, pages 59–68. GI, 2007.

[3] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: Light-weight linked data publication from relational databases. WWW ’09, pages 621–630, New York, NY, USA, 2009. ACM.

[4] Christian Bizer. D2rq - treating non-rdf databases as virtual RDF graphs. In *In Proceedings of the 3rd International Semantic Web Conference (ISWC2004*, 2004.

[5] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[6] Christian Bizer and Andreas Schultz. The Berlin SPARQL Benchmark. *International Journal On Semantic Web and Information Systems*, 2009.

[7] Jeremy J. Carroll and Graham Klyne. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[8] Huajun Chen, Zhaohui Wu, and Yuxin Mao. Rdf-based ontology view for relational schema mediation in semantic web. KES’05, pages 873–879, Berlin, Heidelberg, 2005. Springer-Verlag.

[9] Richard Cyganiak, Souripriya Das, and Seema Sundara. R2RML: RDB to RDF Mapping Language. W3C recommendation, W3C, September 2012. http://www.w3.org/TR/r2rml/.

[10] ECMA International. *Standard ECMA-262 - ECMAScript Language Specification.* 5.1 edition, June 2011.

[11] Yu Deng Edward Hung and V.S. Subrahmanian. Rdf aggregate queries and views. ICDE’05, 2005.

[12] Lorena Etcheverry and Alejandro A. Vaisman. Views over rdf datasets: A state-of-the-art and open challenges. *CoRR*, abs/1211.0224, 2012.

[13] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *Proc. VLDB Endow.*, 5(2):97–108, October 2011.

[14] Steven Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, W3C, March 2013. http://www.w3.org/TR/sparql11-query/.

[15] Wangchao Le, Songyun Duan, Anastasios Kementsietsidis, Feifei Li, and Min Wang. Rewriting queries on sparql views. WWW ’11, pages 655–664, New York, NY, USA, 2011. ACM.

[16] Geetha Manjunath, R Badrinath, Craig Sayers, and Venugopal K. S. Temporal views over rdf data. WWW ’08, pages 1131–1132, New York, NY, USA, 2008. ACM.