R2V: Resource Shapeに基づくRDF文書の検証方法と評価

2009SE298 脇田 宏威　2011SE188 中島 啓貴　2011SE194 成田 貴大
指導教員：青山 幹雄

1　背景

RDF 文書の検証の仕組みが必要とされており，推論規則定義言語の解釈を拡張し制約定義言語として利用する検証方法 [1][4] が提案されている．しかし，推論規則定義言語は開世界仮説や唯一名仮説を前提とするため，型定義として用いることはできない．このような語彙の意味に沿わない運用は，語彙に対する複数の意味の混在という弊害を伴う．

意味の混在を伴わない検証を実現するため，制約定義言語を用いてシェイプと呼ばれる RDF モデルに対する制約文書を記述し，検証に利用する方法が提案されている．

2　研究課題

制約定義言語の 1 つとして Resource Shape がある．しかし，これを用いた RDF 文書検証の方法は現在確立されていない．そこで，本稿では Resource Shape に基づく RDF 文書検証の実現方法を提案するため，検証プロセスの定義と評価を研究課題とする．

3　関連技術

3.1　RDF[3]

RDF (Resource Description Framework) とは，Web 上でリソースに関する情報を記述するための枠組みである．RDF で意味づけられた情報は人間だけでなく計算機にも解釈可能で，収集や解析の自動化が可能である．

RDF では，任意の語彙を URI で一意に識別し，定義できる．

(1)　RDF の抽象構造

RDF ではリソース間の関係情報を記述することでリソースに関する情報を表現する．リソース間の関係情報は主語，述語（プロパティ），目的語（値）の 3 つのリソースによって構成され，この抽象構造を用いて表現された文をトリプルと呼ぶ．

トリプルを構成するリソースは URI，空白ノード，リテラルの 3 種類に分類できる．空白ノードは文書内のみでリソースを識別するための局所的な識別子である．リテラルは具体的な値を表し，値を表す文字列と，整数や日付などの型を識別するための情報の組である．

トリプルにおける主語は URI か空白ノードのいずれか，述語は URI，目的語は URI か空白ノード，リテラルのいずれかである．

RDF モデルを表現する具象構文としては，有向ラベル付きグラフや RDF/XML，Turtle などが用いられる．RDF は抽象構文を定義しているため，特定の具象構文に依存することなく同一の情報を記述できる．

3.2　SPARQL[5]

(1)　SPARQL とは

SPARQL (SPARQL Protocol And RDF Query Language) とは，RDF モデルの問い合わせ言語とプロトコルを定義した仕様である．本稿ではこの問い合わせ言語を SPARQL と呼ぶ．

(2)　具象構文への非依存性

リソースは RDF モデルとしてエンドポイントに登録されるため，RDF 文書の具象構文に依存することなく問い合わせできる．

(3)　クエリの種類

SPARQL は SELECT，ASK，CONSTRUCT，DESCRIBE の 4 種類のクエリを提供する．

SELECT クエリを実行すると，検索条件に一致する変数バインディングが得られる．

ASK クエリを実行すると，検索条件に一致する部分グラフが存在する場合は真，存在しない場合は偽が得られる．

CONSTRUCT クエリを実行すると，検索条件に一致する変数バインディングを利用して生成された任意の新たな RDF グラフが得られる．この際，エンドポイントに登録されている RDF グラフは変更されない．

DESCRIBE クエリを実行すると，検索条件に一致するリソースに関する有益な情報を表す RDF グラフが得られる．この情報は発行者が決定するため，必ずしも同じ形式の情報が提供されるわけではない．

3.3　OWL[2]

OWL (Web Ontology Language) とは，リソースの意味と関係の記述を行うためのオントロジ言語である．新たなクラスの定義に既存のクラス定義の和集合や積集合，補集合を用いることができる．

3.4　Resource Shape[6]

RDF モデルに対する文書型と要素型を RDF モデルとして定義する制約定義言語である．

シェイプとその制約対象となったリソースを照合し正当性を確認する方法は示唆の段階にとどまっている．

(1)　リソースへのシェイプの適用

シェイプは特定の型のリソースにのみ適用可能なシェイプと任意のリソースに適用可能なシェイプに分類できる．

特定の型のリソースにのみ適用可能なシェイプは `oslc:describes` プロパティによる型指定を含み，指定された型のリソースにのみ適用可能である．

任意のリソースに適用可能なシェイプは `oslc:describes` プロパティによる型指定を含まず，

任意の型のリソースに適用可能である．

シェイプは，Resource Shape の定義する語彙を用いてリソースに関連付けられ，その中のすべての適用可能なシェイプがリソースに適用される．リソースは，適用されたシェイプに定義されたすべての制約を満たさなければならない．

リソースにシェイプを 1 つも関連付けていない場合は，リソースを制約するシェイプが存在しないためリソースは正当である．リソースに関連付けたすべてのシェイプが適用できない場合はエラーとして扱うべきである．

(2) リソースへのシェイプの関連付け

以下に述べる 3 種類の方法のいずれかを用いることで，シェイプとリソースを関連付けることができる．

`oslc:instanceShape` **プロパティによる関連付け**は，リソースからシェイプへリンクすることで，リソースとリンク先のシェイプを関連付ける．

`oslc:resourceShape` **プロパティによる関連付け**は，サービス記述からシェイプへリンクすることで，サービスで扱うリソースとそのシェイプを関連付ける．サービス記述とは，ソフトウェアが提供するサービスに関する情報を定義するリソースである．

`oslc:valueShape` **プロパティによる関連付け**では図 1 に示すように，シェイプ S_1 がリソース R_1 に適用されていることを前提とし，S_1 から特定のプロパティ P に対する制約として `oslc:valueShape` プロパティを用いてシェイプ S_2 へとリンクすることで，R_1 から P によってリンクされたリソース R_2 と S_2 を関連付ける．

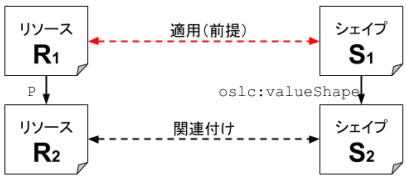

図 1 `oslc:valueShape` プロパティによる関連付け

3.5 RDFUnit[1]

SPARQL クエリで構成されたテストスイートを実行するフレームワークである．RDF モデルの品質評価のために提案されている．

3.6 Pellet Integrity Constraint Validator[4]

OWL を制約定義言語として用い RDF モデルの正当性を確認する検証器である．制約定義から SPARQL クエリを生成することで検証を実現している．

4 RDF 文書検証の問題点

4.1 従来の RDF 文書の検証方法

OWL は推論規則定義言語であり，開世界仮説と非唯一名仮説を前提としている．しかし，これらの前提を無視することで RDF モデルに対する制約定義として用いられることがある．Resource Shape は制約定義言語であるため，語彙の本来の運用方法で RDF モデルに対する制約定義を実現する．

4.2 ValueShape 制約の循環定義

検証を有限時間内に完了するためには，他の検証結果に依存しない組み合わせに到達する必要がある．しかし ValueShape 制約の循環定義が発生した場合，そのような組み合わせに到達しない経路が作られるため，検証を有限時間内に完了できない場合がある．

4.3 RDF の性質

RDF は開世界仮説を前提としており，閉世界仮説を前提とする文書検証の定義を適用できない．このような RDF の性質を前提とした検証を定義する必要がある．

5 RDF 文書検証の定義

5.1 検証範囲

RDF 文書検証における検証範囲を，検証対象 RDF 文書の閲覧者が参照可能な RDF グラフとする．本稿では以降，ある RDF 文書の閲覧者が参照可能な RDF グラフを，ある RDF 文書に基づく参照可能グラフと呼ぶ．

ある RDF 文書に基づく参照可能グラフは以下のように再帰的に定義する．

1) ある RDF 文書に含まれる RDF グラフ
2) サービス記述に含まれる RDF グラフ
3) 参照可能グラフに含まれる URI から参照可能な RDF 文書に含まれる RDF グラフ

5.2 入力

RDF 文書検証における入力は，検証対象 RDF 文書およびサービス記述とする．ただし，サービス記述は存在しない場合与えなくても良い．

5.3 出力

RDF 文書検証における出力は，検証結果を表すリソースの URI とし，このリソースを結果リソースと呼ぶ．結果リソースには検証結果の他に，検証対象となった RDF 文書の URI，作成日時，正当でないと判断される原因となった箇所を特定可能な診断情報が含まれる必要がある．

6 アプローチ

我々は以下に述べる 4 つのアイデアを用いて，RDF 文書検証を実現する．

6.1 部分グラフの検索による制約充足性の判断

Resource Shape による制約の充足性は，制約充足性に影響する部分グラフの有無によって判断できる [6]．

このことから，上記の部分グラフの検索条件を SPARQL クエリとして表現することで，制約の充足条件を構造化し，検証を実現する．

6.2 SPARQL による制約の有効化条件の一意表現

Resource Shape による制約のリソースに対する有効性は，特定の部分グラフの有無によって判断できる．

このことから，上記の部分グラフの検索条件を SPARQL クエリとして表現することで，制約の有効化条件を構造化し，制約の有効性の確認を実現する．

6.3 依存関係グラフを用いた制約の循環定義の検出

依存関係グラフとは，リソースとシェイプの組を表すノードから成る有向グラフであり，ValueShape 制約によって形成される依存関係を表現する．制約に循環定義が存在する場合，依存関係グラフは閉路を持つ．

このことより，依存関係を解決する過程で依存関係グラフの閉路の有無を確認することで，制約の循環定義を検出する．

6.4 検証の局所的中断による有限時間内での検証遂行

検証結果は分割統治法により部分的な検証結果の集約で決定できる．

このことから，制約の循環定義を検出した場合に部分的な検証結果を循環を表す結果に決定することで，依存関係の解決を局所的に中断し，以降の検証を継続する．

この検証の局所的中断の導入により有限時間内での検証が可能となる．

7 提案方法

7.1 前提条件

本研究は，RDF 文書の構文的誤りの検出や，シェイプの構成的誤りの検出を対象としない．そのため RDF 文書は構文的に正しく，かつシェイプは構成的に正しいと仮定する．

また，既存のリソースに含まれる空白ノードは参照できない，これにより空白ノードが ValueShape 制約の対象となった場合，結果リソースとして正当でないと判断される原因となった箇所を特定可能な診断情報を表現することができない．そのため，空白ノードが ValueShape 制約の対象とならない場合のみを想定する．

7.2 検証結果の定義

検証結果は**正当 (Valid)**，**正当でない (Invalid)**，**失敗 (Failed)** のいずれかをとる．検証結果は図 2 に示す関係に基づき決定する．

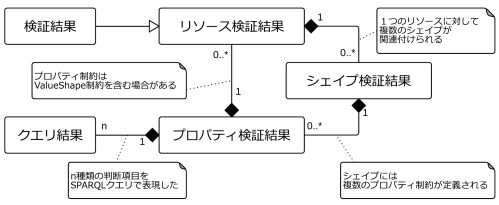

図 2 検証結果のメタモデル

7.3 クエリテンプレートの作成

検証では，制約充足性判断と診断情報生成のために SPARQL クエリを用いる．制約充足性の判断には，制約充足に必要な部分グラフを検索する ASK クエリ（判定クエリ）と，制約充足を妨げる部分グラフを検索する ASK クエリ（例外クエリ）を用いる．診断情報の生成には，制約充足を妨げる部分グラフから診断情報を抽出し，RDF グラフを生成する CONSTRUCT クエリ（診断クエリ）を用いる [6]．

これらの SPARQL クエリは Resource Shape の仕様をもとにテンプレートとして作成した．テンプレートには制約の充足条件に加え制約の有効化条件も含まれる，そのためシェイプの内容に関わらず，テンプレートから生成したすべてのクエリを実行することで，Resource Shape によって定義されるすべての制約の充足性を判断できる．

7.4 検証プロセス

検証は図 3 に示すアクティビティ図に従って行う．

(1) 再帰的な手順による検証範囲の再現

検証範囲構築では，検証範囲の再帰的な定義に従い SPARQL クエリを再帰的に実行することで，検証範囲をエンドポイントに再現し，探索可能な状態にする．

(2) 循環の検出方法

検証のアクティビティでは，ValueShape 制約が形成する循環定義を検出するために依存関係グラフを用いる．ValueShape 制約によって形成される依存関係はリソースとシェイプの組み合わせによって決定されるため，ノードはリソースとシェイプの組とし，図 3 (e) シェイプ検証結果取得で依存関係グラフの更新をする．図 3 (e) シェイプ検証結果取得で入力として渡されたリソースとシェイプの組を依存関係グラフのノードに追加し閉路の有無を確認することで，制約の循環定義の発生を検出する．

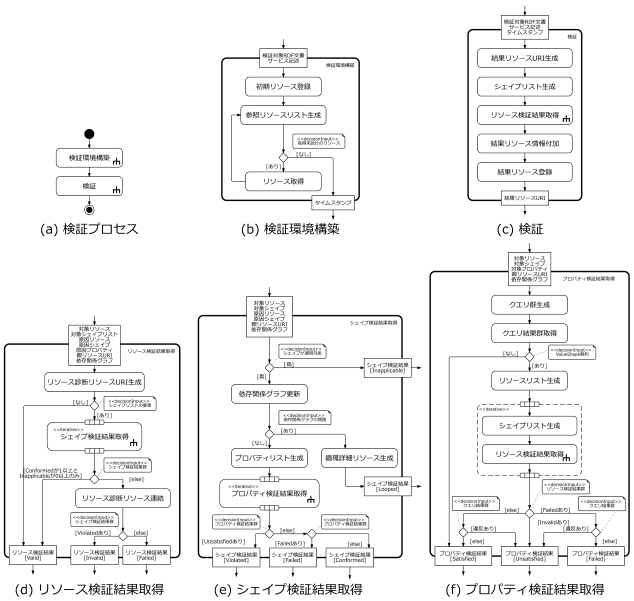

図 3 検証プロセスのアクティビティ図

8 妥当性の確認

8.1 確認方法

本稿では，各制約を表現する SPARQL クエリ群に対するブラックボックス型の単体テストと，検証プロセスを

構成する各アクティビティに対するブラックボックス型の単体テスト，検証アクティビティの例題への適用を通した結合テストによって提案方法の妥当性を確認する．

8.2 単体テスト

それぞれの単体テストの禁則を除外したテストパターン数の一覧を表 1 に示す．

表 1 単体テスト一覧

テストパターンテーブル名	パターン数
AllowedValues 制約	11
MaxSize 制約	5
Occurs 制約	13
Range 制約	8
Representation 制約	16
ValueType 制約	133
検証アクティビティ	14
リソース検証結果取得アクティビティ	65
シェイプ検証結果取得アクティビティ	48
プロパティ検証結果取得アクティビティ	16

8.3 結合テスト

検証アクティビティの結合テストでは，例題に対して検証アクティビティの適用を行った．

例題となるリソース `fooLibrary` は，AllowedValues 制約に違反したリソース `barBook` へのリンクによって ValueShape 制約に違反している．よって，リソース `fooLibrary` が正当でない (Invalid) ことを表す結果リソースが生成され，結果リソースを識別する URI が出力されることが想定される．

例題に対する検証アクティビティの適用によってリスト 1 に示す結果リソースが生成された．

リスト 1 結果リソース

```
1  @base <http://example.org/bookSampleViolation/> .
2  @prefix validation: <http://example.org/validation#> .
3  @prefix dcterms: <http://purl.org/dc/terms/> .
4
5  validation:detail-4a60623e a validation:Detail, validation:
       InvalidResourceError ;
6      validation:detail validation:detail-b6cb8a28 ;
7      validation:basedOn <libraryShape> ;
8      validation:property <collect> ;
9      validation:subject <fooLibrary> ;
10     validation:value <barBook> .
11
12 validation:detail-b6cb8a28 a validation:Detail, validation:
       UnallowedValueError ;
13     validation:basedOn <bookShape> ;
14     validation:property <genre> ;
15     validation:subject <barBook> ;
16     validation:value <Comic> .
17
18 validation:detail-ab38c3f1 a validation:Detail, validation:
       InvalidResourceError ;
19     validation:detail validation:detail-4a60623e .
20
21 validation:result-0e6cc892 a validation:Result ;
22     validation:detail validation:detail-ab38c3f1 ;
23     validation:document <fooLibrary> ;
24     validation:status validation:Invalid ;
25     dcterms:created "2015-02-15T09:56:02.44+09:00" .
```

結果リソースは，リソース `barBook` が AllowedValues 制約に違反したことで，リソース `fooLibrary` が ValueShape 制約に違反したことを表す診断リソースを含む．

8.4 結果

単体テストと結合テストから，提案方法によって Resource Shape の仕様と，本稿で定義した RDF 文書検証の定義の両方を満たす結果が得られることを確認できた．

9 評価

全てのテストパターンにおいて，想定通りの結果が得られたため，各制約を表現するクエリ群と検証アクティビティ，リソース検証結果取得アクティビティ，シェイプ検証結果取得アクティビティ，プロパティ検証結果取得アクティビティは，Resource Shape の仕様と，本稿における RDF 文書検証の定義に対して妥当といえる．

このことから，提案方法により Resource Shape を用いた RDF 文書検証が可能となった．既存のアプローチ [4][1] では不可能だった意味の混在を伴わない RDF 文書検証を実現できたといえる．

10 今後の課題

10.1 計算効率の向上

本研究は，提案方法の計算効率を対象としないが，以下に挙げる方法によって計算効率の向上が期待できる．

1) 診断クエリのみの利用
2) シェイプ検証結果のメモ化
3) 展開領域の並行処理

10.2 空白ノードで表現されるリソースの扱い

空白ノードは SPARQL 1.1 クエリ言語の仕様により，そのままでは参照不可能である [3]．そのため，空白ノードは検証できず，また診断リソースにおいて空白ノードを指示できない．

この問題は空白ノードのスコーレム化によって解決可能だが，それに対応した検証プロセスの定義が必要である．

11 まとめ

Resource Shape に基づく RDF 文書検証を SPARQL を利用して実現し，単体テストと結合テストによって検証方法の妥当性を示した．提案方法は，Linked Open Data[7] などの検証への活用が期待できる．

参考文献

[1] S. A. R. Cornelissen et al. RDFUnit, 2014. `http://aksw.org/Projects/RDFUnit.html` (accessed 2015.1.13).

[2] P. Hitzler, et al. OWL 2 Web Ontology Language Primer (Second Edition), 2012. `http://www.w3.org/TR/2012/REC-owl2-primer-20121211/` (accessed 2015.1.13).

[3] G. Klyne et al. RDF 1.1 Concepts and Abstract Syntax, 2014. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/` (accessed 2015.1.13).

[4] H. Prez-Urbina et al. Validating RDF with OWL Integrity Constraints, 2008. `http://docs.stardog.com/icv/icv-specification.html` (accessed 2015.1.13).

[5] E. Prud'hommeaux. SPARQL Query Language for RDF, 2008. `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/` (accessed 2015.1.13).

[6] A. G. Ryman. Resource Shape 2.0, 2014. `http://www.w3.org/Submission/2014/SUBM-shapes-20140211/` (accessed 2015.1.13).

[7] W3C. Linking Open Data. `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData` (accessed 2015.1.13).