人工知能学会研究会資料
SIG-SWO-041-11

D'ownLOD: LOD のためのラピッドプロトタイピングプラットフォーム

D'ownLOD: a Rapid Prototyping Platform for LOD

的野 晃整 [1*]
Akiyoshi MATONO[1]

[1] 国立研究開発法人 産業技術総合研究所
[1] National Institute of Advanced Industrial Science and Technology (AIST)

Abstract: Linked Data is expected as a key technology of open data, web of data, and the semantic web. Linked data is used in a lot of fields and is written in an uniformed structured data format, and then we think that linked data itself must be a foundation infrastructure for rapid prototyping. In this paper, we propose an rapid prototyping platform called D'ownLOD. Using D'ownLOD, you can easily implement a web application using linked data that can be downloaded and immediately run standalone.

1 はじめに

Linked Open Data (LOD) は，分野を横断して相互にリンクした構造化データである．また，LOD を用いることで計算機が意味を理解できるデータのウェブを構築することを目指したプロジェクトそのものの名称でもある．LOD はオープンデータの要の技術として，また次世代 Web である Semantic Web への足掛りとして，大きな期待が寄せられており，現在多様で膨大な LOD が公開されている．しかしながら，その熱狂的な流行のピークを過ぎた現在でも，その認知度や利用が世間一般にまで広がっているという状況には至っていない．この最大の理由は，LOD のキラーアプリケーションが見つかっていないことにあると考えている．

LOD キラーアプリケーションについての厳密な定義はないが，LOD でなければ実現できず，LOD の重要性や魅力を広め，普及を飛躍的に促進するアプリケーションである．この LOD キラーアプリケーションがないという問題意識は，研究者の間で共有されている．ハッカソンやアイデアソンなど LOD に関連したイベントが頻繁に開催されている理由は，それらを通じてキラーアプリケーションの発掘を目指しているためであると考えている．しかしながら，今日までキラーアプリケーションはおろか，LOD でなければ実現できないアプリケーションでさえも見つかっていないと考えている．

なぜ LOD でなければ実現できないアプリケーションでさえ見つからないのか？その回答の一つとしては，LOD はそもそも既存するデータを用いて，外部リンクを作成し，フォーマットを Resource Description Framework (RDF) [1] に変換したものであり，LOD を使ったアプリケーションは既存データでも実現できるためである．また，既存データは関係データベースで管理されていることが主であり，発展途上の RDF データベースに比べると性能や信頼性，機能の面で有利であることも既存データを採用する動機となるのは自然である．さらに，既存データが一次データであるということは鮮度の面でも優位である．唯一，LOD が優位と思われる部分として，他のデータとリンクしている部分があるが，これもアプリケーション専用に効率化された連携機能を独自に構築したほうが効率的で無駄がない．そのため，LOD を使うメリットがほぼ皆無であるという状況であるため，LOD でなければ実現できないアプリケーションが存在していないのである．[*1]

LOD でなければ実現できないアプリケーションは存在しないが，一方で LOD でなければ実現できないアプリケーション開発方法は存在していると考えている．そ

* 連絡先： 産業技術総合研究所
〒305-8560 茨城県つくば市梅園 1-1-1
E-mail: a.matono@aist.go.jp

[*1] 現在は LOD でなければ実現できないアプリケーションは存在しないという主張であるが、将来一次データが LOD となり，効率的なプラットフォームが開発されれば，この状況は打開できると考えている．

れはラピッドプロトタイピングである．ラピッドプロトタイピングとは，製品やアプリケーションなどの開発において初期のデザインや設計が進んだ少し段階で，短時間で試作することで評価と設計を試行錯誤的に繰り返しながら実装する開発方法である．LOD は，異分野のデータが相互にリンクして，同一フォーマットで記述されて，公開されていることから，ラピッドプロトタイピングに適したデータである．本稿では，LOD を用いたラピッドプロトタイピングのためのプラットフォームシステム *D'ownLOD* を設計・開発したので，これを紹介する．

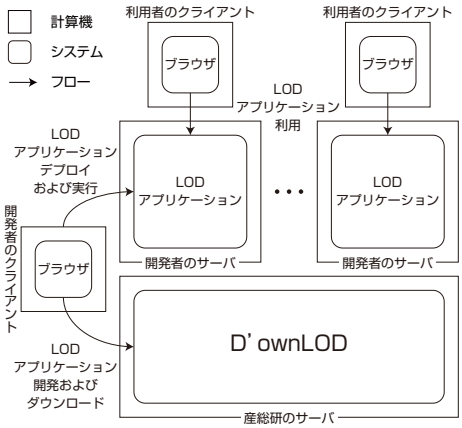

図 1: D'ownLOD と LOD アプリケーションの概要

2 D'ownLOD

2.1 概要

D'ownLOD は，LOD アプリケーションを開発できるプラットフォームである．図 1 に D'ownLOD の概要図を示す．想定ユーザとしては，LOD アプリケーションを開発する開発者と，LOD アプリケーションを利用する利用者の 2 種がある．D'ownLOD は産業技術総合研究所が提供するサーバ上で動作する Web アプリケーションで，開発者はブラウザから D'ownLOD にアクセスして，LOD アプリケーションを開発する．D'ownLOD を用いて開発された LOD アプリケーション（以下，D'ownLOD アプリケーションという．）は，ファイルとしてダウンロードできる．D'ownLOD アプリケーションは個別の Web アプリケーションとして動作する．開発者は動作確認等の目的であればローカルでも実行すれば動作するし，外部に公開する目的であれば，開発者が独自に用意するサーバ上で実行することで，単体の LOD アプリケーションとして動作する．外部公開したのであれば，アプリケーションの利用者は，通常の Web アプリケーションとして D'ownLOD アプリケーションにブラウザからアクセスできる．なお，D'ownLOD そのものを D'ownLOD アプリケーションと区別するため，D'ownLOD プラットフォームと呼ぶ．

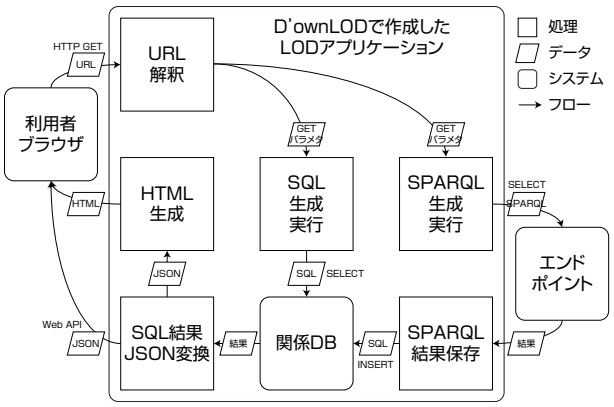

図 2: D'ownLOD アプリケーション

2.2 D'ownLOD アプリケーション

図 2 に D'ownLOD アプリケーションの処理やデータの流れを図示する．まず，利用者はブラウザから HTTP GET メソッドによって D'ownLOD アプリケーションにアクセスする．D'ownLOD アプリケーションは，URL を解釈し，GET パラメタに基づいて，SPARQL [2] クエリを生成し， SPARQL エンドポイント（以下 EP という）へ発行する．この SPARQL クエリは，プレースホルダを含む虫食い状態の SPARQL クエリをアプリケーション開発者が定義しておき，そのプレースホルダに GET パラメタの値が補完することで生成されたものである．EP から SPARQL クエリの結果が返ってくると，SPARQL クエリとその結果はローカルの関係データベース（以下 DB という） にキャッシュする．そのため，すでに実行済みの SPARQL クエリである場合，EP へは発行されない．その後，GET パラメタに応じて SQL クエリが生成され，ローカル DB に対して問合せを行う．この SQL クエリも先ほどの SPARQL クエリと同様にプレースホルダを含む SQL を開発者が定義しておき，そのプレースホルダを GET パラメタの値で補完して生成される．DB に問合せした結果は，JSON 形式に変換される．JSON は HTML を通じて可視化することで，利用者は動的な Web ページを閲覧できる．また，JSON は Web API としてアクセスすることもできる．

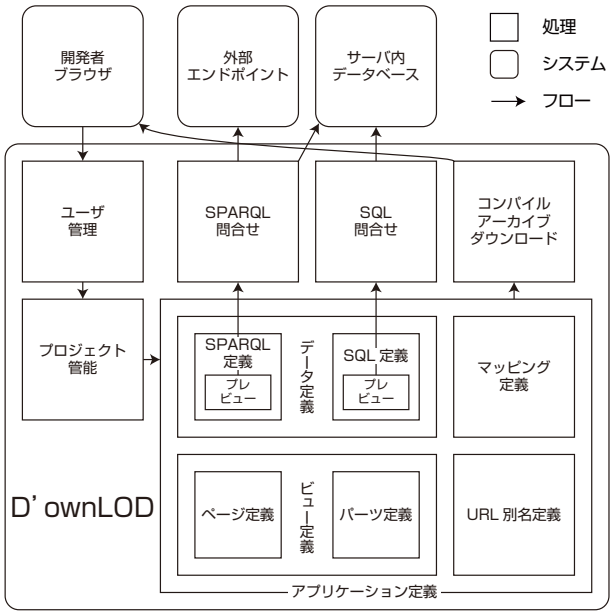

図 3: D'ownLOD プラットフォーム

2.3 D'ownLOD プラットフォーム

D'ownLOD プラットフォームの概要図を図 3 に図示する．D'ownLOD は, ユーザ管理やプロジェクト管理の機能を有しており，個人のアプリケーションを作成したり，開発中のアプリケーションをプロジェクトとして保存したりする管理機能を提供する．その後，D'ownLOD アプリケーション定義として，データ定義，ビュー定義，マッピング定義，および URL 別名定義の 4 つの定義を行う．この 4 つの定義を通じて D'ownLOD アプリケーションは作成される．

データ定義は SPARQL 定義と SQL 定義で構成される．SPARQL 定義ではプレースホルダを含む SPARQL クエリを定義する．定義した SPARQL クエリが正しく動作するかどうかを確認するために，プレビュー機能も有しており，D'ownLOD プラットフォームから外部の EP に SPARQL クエリを発行する機能がある．SQL 定義ではプレースホルダを含む SQL クエリを定義する．SQL 定義では，SPARQL 定義と同様にプレビュー機能を有する．

ビュー定義は，データ定義を通じて作成されたデータをどのように表示するかを定義する．ビュー定義はページ定義とパーツ定義の二つで構成される．ページは複数のパーツから構成された Web ページに対応している．ページ定義では，使うパーツを指定し，それらの順序を定義する．パーツ定義では，部分 HTML を記述し，その過程でデータを記載する．データは DB から検索した結果が JSON として渡されるため， JSON を可視化する Javascript を含む HTML を記述する．

マッピング定義は，データ定義とビュー定義のマッピングを定義する．つまり，どのデータをどのビューで表示するかを定義するもので，この機能によってデータとビューを分離して定義できる．例えば，どんなデータであっても可視化できる，表形式の表示などの汎用のビューは，多様な場面やアプリケーションで使いまわすことができる．

URL 別名定義は，別名 URL とフォワード先の URL を定義できる機能で，トップページや短縮 URL などが定義できる．D'ownLOD では， GET メソッドによって SPARQL や SQL を動的に変更するため，URL が自然と長くなりがちで，きれいな URL とはいいがたいものにならざるを得ない．そのため，URL 別名機能を通じて別名 URL を定義することができる．

これらのアプリケーション定義によって作成されたアプリケーションは，サーバ上で自動的にコンパイルされ，Web サーバや Spring Boot 等のライブラリを同梱した jar ファイルとしてアーカイブされる．開発者はコンパイルが終了した後に，すぐに実行できるアプリケーションとしてダウンロードできる．

2.4 アプリケーション定義の詳細

D'ownLOD アプリケーションを作成するために開発者は，1) データ定義，2) ビュー定義，3) マッピング定義，4) URL 別名定義の 4 つを定義する．

1) データ定義は，アプリケーションで使うデータを定義する．データ定義は, 1-1) SPARQL 定義と 1-2) SQL 定義から構成される．SAPRQL 定義 では EP に問合せる SPARQL クエリを，SQL 定義では DB に問合せる SQL クエリを定義する．SPARQL クエリと SQL クエリは，任意数のプレースホルダを設定できる．それぞれのプレースホルダを区別するため，前者を SPARQL プレースホルダ，後者を SQL プレースホルダと呼ぶ．

1-1) SPARQL 定義では，任意の EP に対して 任意の SPARQL SELECT クエリを発行できる．SPARQL 定義では，SPARQL クエリの名称，および SPARQL プレースホルダの数と，各 SPARQL プレースホルダの名称，それらの SPARQL プレースホルダをすべて含む SPARQL クエリと SPARQL EP を定義する．SPARQL クエリの例を Listing 1 に示す．このクエリは，あるチームに所属する野球選手の一覧を取得するもので，1 つの SPARQL プレースホルダ (`team`) を含む．チームに SPARQL プレースホルダが配置されてお

り，この部分が GET パラメタに応じて後で補完される．なお，SPARQL プレースホルダは，###<SPARQL クエリ名>.<SPARQL プレースホルダ名>### の形式で記述される．

Listing 1: SPARQL の例

```
SELECT ?label ?abstract ?birth ?now
WHERE {
?s dcterms:subject <http://.../Category:野球選手> ;
   rdfs:label ?label ;
   dbp-owl:birthDate ?birth ;
   dbp-owl:abstract ?abstract ;
   dbp-owl:team <http://.../###sparql1.team###> ;
   prop-ja:所属球団 ?now .
}
```

D'ownLOD プラットフォーム上では，定義した SPARQL プレースホルダを含む SPARQL クエリをテストして，プレビューによって確認することができる．プレビューでは，各 SPARQL プレースホルダに対して，置換する値を明記することで，完全な SPARQL クエリを生成して，発行する．SQL のプレビューで利用するために，SPARQL クエリの結果は，クエリと共に DB 内に保存する．

1-2) SQL 定義では，前述の SPARQL クエリで取得した LOD を保存しているローカル DB に対して，データの絞り込み等の加工を行うために SQL クエリを定義する．SQL 定義では，SQL クエリの名称，および SPARQL クエリの選択，SQL プレースホルダの数と，各 SQL プレースホルダの名称，それらの SQL プレースホルダを含む SQL クエリを定義する．SQL クエリは SPARQL クエリの結果を保存したテーブルから検索するため，対象の SPARQL クエリを選択し，それを FROM 句に記述する．したがって， テーブル名は SPARQL 名および SPARQL プレースホルダ名を含み，<SPARQL 名称>(###<SPARQL 名称>.<SPARQL プレースホルダ名>###) の形式となる．また，SQL プレースホルダの記述方法は，###<SQL クエリ名>.<SQL プレースホルダ名>### となる．

SQL クエリの例を Listing 2 に示す．この例は SQL プレースホルダは利用しておらず，SPARQL プレースホルダを含むテーブルに対して，label と abstract に射影している．SPARQL プレースホルダを含むテーブル名はこの SQL クエリの結果が動的に JSON に変換される．

Listing 2: SQL の例

```
SELECT label, abstract
FROM sparql1(###sparql1.team###)
```

SQL クエリも，SPARQL クエリと同様にプレビューすることができる．ただし，SPARQL 定義でのプレビューのために実行した SPARQL クエリの結果のみが DB に保存されているため，他の SPARQL プレースホルダの値でのプレビューはできない．SQL 定義では，指定した SPARQL をそのまま使う場合，つまり SPARQL クエリの結果に対して特に加工が必要ないのであれば，SPARQL の指定をするだけで，デフォルトの SQL クエリが自動生成される．そのため，SQL クエリの記述が必要が不要であれば，SQL 定義は SQL の名称と SPARQL クエリを選択するのみで，SQL プレースホルダや SQL クエリの記述も不要である．

2) ビュー定義はアプリケーションのビューを定義するもので，2-1) ページ定義と 2-2) パーツの 2 種類の定義がある．ページとはアプリケーションの Web ページの対応しており，ページは複数のパーツを含むことができる．パーツとはページを構成する要素で，HTML の div タグに挿入される HTML の部品なる．

2-1) ページ定義は，ページ内に表示するパーツを選択することで，一つの Web ページを定義する．ページ定義では，ページの名称と，複数パーツおよびそれらの出現順を定義する．現時点では単にパーツの順序のみを定義するが，将来的にはパーツのレイアウトを定義できるよう拡張したい．

2-2) パーツ定義は， 部分 HTML を記述し，その中に SQL クエリの結果である JSON を表示するように記述する．パーツ定義では，パーツの名称，および 部分 HTML を定義する．HTML 内で JSON を表示するには javascript を使う必要があるが，JSON から DOM を生成することで可視化は可能だが，MVVM ライブラリ，例えば JsViews[*2] や Knockout.js[*3] ，AngularJS[*4]など，を利用することでより簡単に JSON を可視化できる．JsViews を用いてパーツを定義する例を Listing 3 に示す．この例では JSON データを table 形式で表示している．JSON データは results オブジェクトとして渡され，その中の rows 配列に SQL クエリの結果が束縛されている．このパーツは任意の JSON データを表形式で表示するため，汎用で使うことができる．

Listing 3: パーツ定義の例

```
<div id="table"></div>
<script id="jsviews" type="text/x-jsrender">
{{for results}}
  <table>
```

[*2] https://www.jsviews.com/
[*3] http://knockoutjs.com/
[*4] https://angularjs.org/

```
    <thead>
      <tr>
      {{props rows[0]}}
        <th>{{:key}}</th>
      {{/props}}
      </tr>
    </thead>
    <tbody>
    {{for rows}}
      <tr>
      {{props #data}}
        <td>
        {{:prop}}
        </td>
      {{/props}}
      </tr>
    {{/for}}
    </tbody>
  </table>
{{/for}}
</script>
<script>
$(function() {
  $('#table').append($('#jsviews').render(jsondata))
})
</script>
```

3) マッピング定義では，データとビューの関連付け，つまり，どのビューにどのデータを適用するかを定義する．具体的には，マッピングの名称と，一つの SQL クエリと 一つのパーツを定義することで，データとビューの関連付けを行う．これによって，どの SQL の結果からどのパーツに，JSON データが渡されるかを定義する．

4) URL 別名定義では，URL の別名を定義することで短い URL やトップページなどの特殊ページを定義することができる．URL 別名定義では，URL の別名とフォワード先 URL のペアを定義する．別名 URL にアクセスすると，フォワード先 URL にリダイレクトされる．無名の別名を定義することで，トップページを定義できる．フォワード先 URL として定義できるのは，ページあるいは SQL，SPARQL の 3 種類で，ページの場合は通常の HTML を返すが，SQL や SPARQL では JSON を返すため，Web API として利用できる．

2.5 D'ownLOD の特徴

D'ownLOD の特徴を以下に列挙する．

- **モビリティ**: 一つのファイルにすべての機能を同梱しているため，移動が容易で，D'ownLOD アプリケーションそのものをダウンロードすることができる．
- **即時実行可能**: D'ownLOD アプリケーションは，Java Spring Boot フレームワークおよびデータベース H2，Web サーバ Jetty 等を同梱しており，さらにサーバサイドでコンパイルしてアーカイブするため，Java さえインストールされていれば，ダウンロードした後，すぐ実行できる．
- **SPARQL クエリ発行機能**: 任意の SPARQL EP に対して任意の SPARQL クエリを発行できる．SPARQL クエリの一部にプレースホルダを設置でき，後から値を補完できる．
- **DB キャッシュ機能**: SPARQL クエリの結果はローカル DB にキャッシュされるため，すでにキャッシュ済みのクエリについては高速に結果を取得できる．
- **SQL 結果の JSON 変換**: DB へ格納されたデータを SQL で問合せした結果は，動的に JSON に変換される．JSON は HTML から読み込むことができ，テンプレートに流し込むことで，ビューを生成する．
- **HTTP GET パラメタによる動的ページ**: HTTP GET メソッドの GET パラメタに応じて，表示される情報が動的に変わる．
- **ソースコードの同梱**: ダウンロードしたファイルには，ソースコードも同梱しており，修正や機能追加などはローカルで実装することもできる．
- **Web API 機能**: JSON を直接ダウンロードできる URL も提供しており，Web API として機能する．
- **テーブル定義の自動生成**: DB のテーブル生成等のスキーマ定義は SPARQL クエリに応じて自動に生成される．テーブルの定義をする必要がなく，また INSERT も自動で実行されるため，開発者は SELECT 文のみを定義することになる．
- **データとビューの分離**: データとビューを分離して定義できることで，データやビューの再利用性が高い．
- **別名 URL**: 別名の URL を独自に定義できるため，トップページなどの特殊ページの作成や URL を短縮したり，きれいにすることができる．
- **プレビュー機能**: SPARQL や SQL を定義する際，正しく意図通りに動作するかどうかをプレビューして確認することができる．

3 考察

D'ownLOD を用いたアプリケーション定義では，開発者が最低限実装する必要があるものは SPARQL と（必要があれば）SQL，それらを表示する HTML と Javascript のみで，あとは LOD を使ったアプリケーションを試作できる．D'ownLOD アプリケーションは GET パラメタに応じて動的に生成される LOD を可視化するだけの単純なアプリケーションであるため，コントローラの定義を省略することで，単純化を図っている．逆に言えば，データの入力や更新はできないが，ソースコードを同梱しているため，必要な機能は後から追加できる．

LOD がラピッドプロトタイピングに適しているという点に着目した関連研究としては，LinkData.org [3] がある．LinkData.org では独自のデータを LOD として公開し，そのデータを用いたアプリケーションを作成できるという特徴がある．D'ownLOD との最大の違いは，D'ownLOD アプリケーションはそれ自体が単体の Web アプリケーションとして個別に動作するのに対し，LinkData.org のアプリケーションは LinkData.org のサーバ上で動作する．

4 まとめ

本稿では，LOD アプリケーションのためのラピッドプロトタイピングプラットフォーム D'ownLOD を提案した．D'ownLOD の名前の由来は，自身 (own) のアプリケーションをダウンロードできるという本システムの最大の特徴を表している．本システムは，アルファ版として `http://downlod.linkedopendata.net` にて運用している．

今後の課題としては，D'ownLOD の機能拡張は多くが考えられるが，最大の問題点として評価が十分ではないと考えているため，今後は関連研究との比較を含めた評価方法について検討したい．

謝辞

本研究の一部は，JSPS 科研費 24680010 および 15H02781 の助成を受けたものです。

参考文献

[1] W3C. Resource Description Framework (RDF). `https://www.w3.org/RDF/`.

[2] W3C. SPARQL 1.1 Overview. `http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/`. W3C Recommendation 21 March 2013.

[3] 一般社団法人リンクデータ. LinkData.org. `http://ja.linkdata.org/`.