Vol.2017-IFAT-127 No.4
2017/7/22

番組連動型アプリ起動のためのデータ構造化について

浦川真[†1]　藤沢寛[†1]

概要：放送番組で紹介された情報を視聴者の行動につなげる取り組みを行っている．例えば、観光地が番組で紹介されたら、視聴者の地図アプリを起動し行き方を調べるという一連の流れをシームレスにつなげることで、テレビが映像を見るためだけのデバイスではなくユーザー行動を生むデバイスになると考えられる．そこで、著者らは、番組情報とアプリケーション情報を構造化し、番組とスマートフォンアプリを自動的に紐づけ起動できる仕組みを検討した．

キーワード：放送，行動連携，セマンティック Web，オントロジー，推論

Data Structure for Integration of Behavior at TV and Smartphone

MAKOTO URAKAWA[†1]　HIROSHI FUJISAWA[†1]

Abstract: A demand to connect users' behavior between TV viewing and other activities in their real lives increases at the era where broadcast and broadband are integrated with each other. For example, when a sightseeing spot are shown in a TV program, it is thought to be beneficial for users to seamlessly search how to get to the sightseeing spot. This idea redefines a TV device as a hub generating users' behavior. In order to connect TV viewing and other activities, it is necessary to make programs tractable and linkable. The authors applied the semantic web technology to this purpose, and structurized the information about TV programs and smartphone applications. The data structure generates linkage between TV and smartphone applications, and launch parameters for the applications by using reasoner functions. This paper explains the data model and its generated data by reasoning, and shows the demo applications.

Keywords: Ontology, Semantic Web, TV, Smartphone, Reasoning.

1. はじめに

近年、放送と同時にインターネットでライブ配信する、もしくは、放送直後にオンデマンド配信するといった、放送と通信を連携するサービス展開が行われている．また、演出面においても Twitter などのソーシャルネットワークサービスでのコメントを放送で表示する演出や、放送内容に関連したキーワードを表示し、Web で検索することを促すような演出も行われている．このように、放送とインターネットを連携させる取り組みが徐々に広がっている．その一方で、インターネット上では WEB サービスを連携させる仕組みとして、ifttt や zapier といった IFTTT（If-This-Then-That）というサービスが充実していきているab．例えば、ニュースが公開されたら自分あてにメールを送信するといった「レシピ」を作成することができる．このような、何かの条件に対して違うアプリケーションを連携させるようなサービス連携の考え方を、放送にも応用することで、例えば、「厳島神社」が放送されたら、「行く」アプリとして地図アプリを起動するといった、新たな放送と通信の連携が可能になると考える．

そこで、筆者らは、放送内の各シーンに映っている情報とスマートフォンアプリの情報を構造化し、それら情報のクラス定義とプロパティ属性に基づき、推論により連携させた．これにより、「厳島神社」が放送されたら、「行く」アプリとして地図アプリを起動するだけでなく、アプリ側に放送内容である「厳島神社」をセットして起動できる、よりシームレスな連携が可能となった．本稿では、これらデータ構造及び推論処理について報告する．

2. データ構造化の考え方

本稿で実現することは、放送内容に合わせてスマートフォンアプリケーションを起動できるようにすることであり、4 項に示す、放送とスマホアプリを連携する仲介アプリが存在する前提とする．具体的な利用場面としては、放送を視聴しているユーザーのスマートフォンにインストールされた仲介アプリが、本稿にて定義したデータをもとに、放送内容に合わせて視聴者のスマートフォンアプリを推薦することを想定する．

ここでは、放送内容とスマホアプリを連携するためのデータ構造について、概要を図 1 に示すとともに、その考え方を紹介する．

†1 NHK 放送技術研究所
NHK Science & Technology Research Laboratories

a https://ifttt.com/discover

b https://zapier.com/

2.1 放送内容のデータ構造化

放送番組に関する情報として、番組名から放送日時などがあり、電子番組表（EPG）等を通じて、テレビや Web で閲覧できるcとともに、API 等から利用できるサービスも出てきているd．しかし、番組表のための番組情報は、あくまで番組視聴のための案内であり、番組全体に関する情報である．そのため、シーン毎の放送内容に関する情報がないため、本稿で対象とするスマホアプリとの連携に用いるには不十分である．そこで、まず、番組シーン毎に紹介されている情報をエンティティとしてデータ構造化を行った．ただし、「宮島」といった詳細情報だけでは、どのスマホアプリと連携させるかといった処理が複雑になるため、例えば「場所」といった抽象化したクラス定義が必要と考えてデータ化した．なお、放送内容のデータ構造の詳細については 3 項で後述する．

2.2 スマホアプリのデータ構造化

放送内容とスマホアプリを連携させるためには、スマホアプリの情報についてもデータ構造が必要となる．具体的には、「場所」に「行く」場合は地図系のアプリ、「食材」を「買う」場合はネットスーパーアプリというような、動作とその対象を記述する．この記述により、「食材」といったカテゴリー情報をもとに、放送内容とアプリをマッチングできるため、放送内容と連動したアプリ起動が実現できる．ただし、放送内容とアプリを紐づけるだけでは、起動したアプリ上で、ユーザーが放送内容を手入力することになり、放送内容とユーザー行動がシームレスに繋がらない．そのため、アプリを起動する際に必要なパラメータを得るためには、放送内容から、動的にパラメータとしてセットできる仕組みが必要である．

スマホアプリの情報は、アプリストアで確認できるeが、動作とその対象が統一的に記述されないことや、放送内容を起動パラメータに格納する必要があることから、本稿では、3 項に示すオントロジーに基づき、手動でデータ構造化を行った．

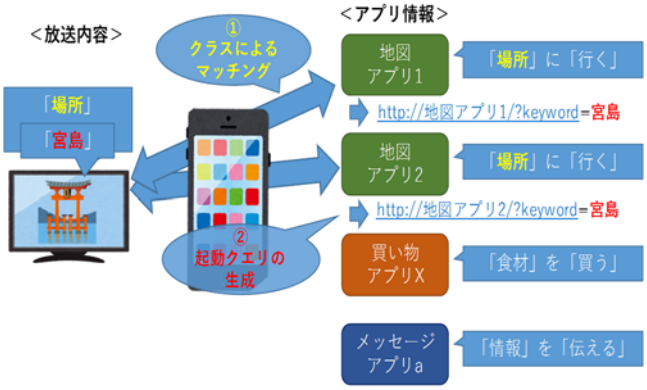

図 1　放送とアプリを連携するためのデータ構造化

放送内容とスマホアプリの情報を動的に、かつ効率的に連携させるため、情報の概念構造を記述するオントロジーによりデータ構造を定義し、情報を三つ組で表現できる Resource Description Framework（RDF）[1]やオントロジー記述言語である Web Ontology Language（OWL）[2]により放送内容とアプリ情報を記述した．さらに、推論によりそれらのデータ連携や生成処理を行った．

3. オントロジーの構築

3.1 オントロジーによるデータ定義

オントロジーとは、対象世界に「存在している」ことをどのように捉えてモデルとして構築したかを明示的に示すものであり、「知識」そのものではなく、骨格となる「概念」を明示したものである．オントロジーにより、語彙の共有だけでなく、知識の体系化や再利用が可能となる[3]．本稿では、放送内容とアプリ情報の概念関係をオントロジーにより定義する（図 2）．図 2 に示す名前空間である tv 及び app は本検討用に一時的に構築したものである．

図中の放送内容オントロジーにあるように、tv:Program クラスのインスタンスに紐づく tv:Scene クラスのインスタンスには、各シーンで紹介される情報を tv:hasEntity プロパティにより関連付けている．紐づけられるエンティティは、所属するクラス情報を保持しているため、放送内容で紹介されているクラス情報を取得することができる．一方、アプリ情報オントロジーでは、放送内容に合わせて検知するための情報として、app:detectedBy プロパティにより、検知できるクラス情報を設定できる．さらに、アプリ情報は、アプリを起動するための情報として、app:launchedBy プロパティで紐づくインスタンスに app:queryURL を保持できる．ただし、放送内容に合わせて動的に変更するため、app:queryURL から推論した結果を app:inferredquery プロパティに保持できるデータ構造とした．なお、放送内容オントロジーの tv:hasEntity やアプリ情報オントロジーの app:detectedBy の値域（rdfs:range）であるクラス群は、運用に応じて追加可能であるため、図 2 はその一例を示している．また、推論ルールの記述を効率化するため、放送内容のエンティティが属するクラスや、アプリ情報を検知する際のクラス情報は、外部クラスを rdfs:subClassOf により参照する．そのため、DBpedia オントロジーのクラスを参照している．推論ルールの記述等の詳細は、3.3 項で紹介する．

c http://www2.nhk.or.jp/hensei/program/index.cgi?area=001&f=top
d http://api-portal.nhk.or.jp/
e https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=ja

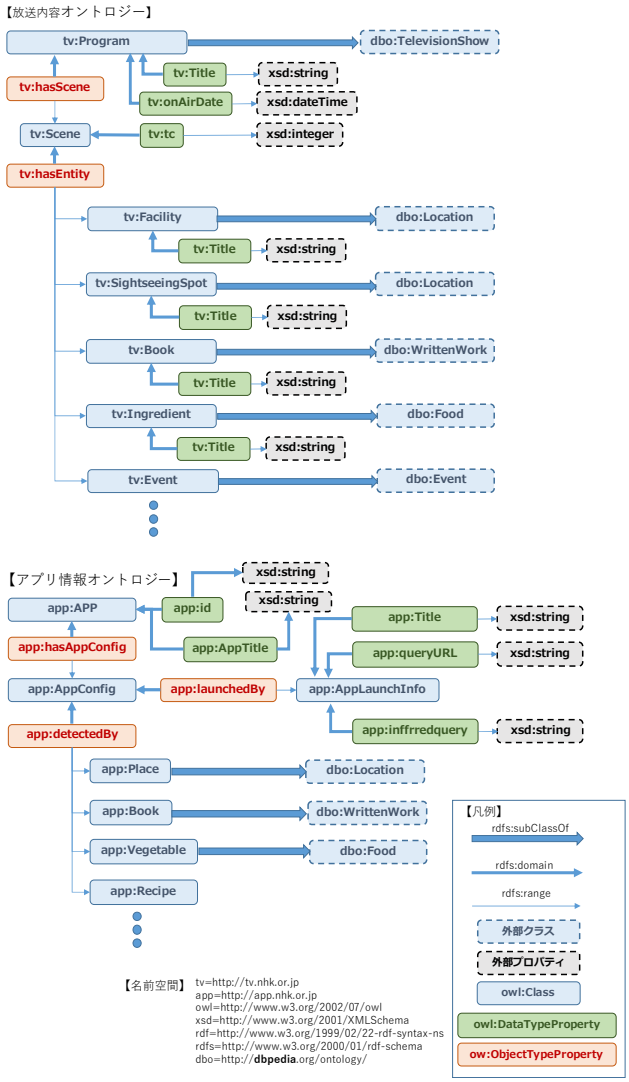

図 2　放送内容とアプリ情報オントロジー

3.2　RDF データの生成

図 2 に示すオントロジーに基づき、オントロジーエディタである Protégé[4]を利用し RDF 形式でデータを作成した．そのインスタンス例を図 3 に示す．図 3 の例では、「小さな旅」という番組の番組冒頭から 45 秒目に S1 というシーンがあり、そのシーンには tv:SightseeingSpot クラスのインスタンスである「厳島神社」インスタンスが、368 秒目に tv:Ingredient クラスの「くわい」インスタンスが紹介されていることを示す．一方、アプリ情報には、app:Place クラスのインスタンスで検知される「GoogleMap」アプリと、app:Vegetable クラスのインスタンスで検知される「きょうの料理」アプリが設定されていることを示す．さらには、アプリ起動情報として、「GoogleMap」が「geo:0,0?q={%X%}」により起動でき、「%X%」は app:Title プロパティの値であることをデータとして表現している．この Title プロパティを、連携した放送内容にもある Title プロパティから取得することで、放送内容に合わせたアプリ起動が可能となる．この推論処理詳細については次項で紹介する．

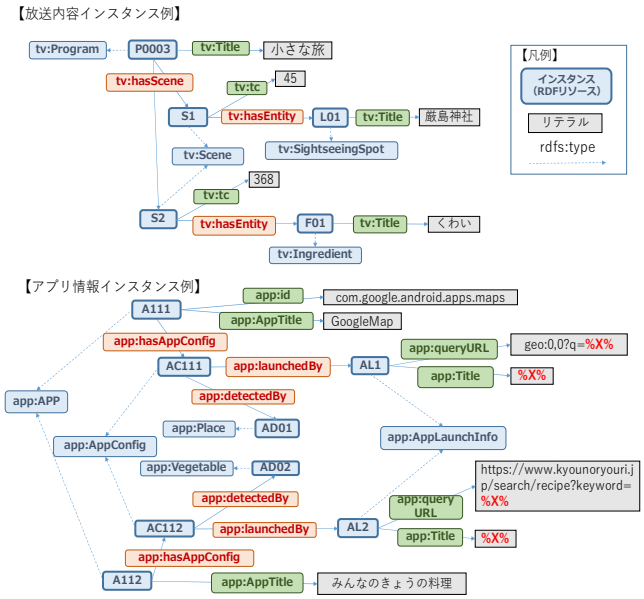

図 3 インスタンス例

3.3　推論による連携データの生成

図 3 の状態では、放送内容とスマホアプリ情報が連携されていない状態にある．本項では、これらの情報をつなげることで、放送内容とスマホアプリを連携し、さらに、スマホアプリを起動するために必要なパラメータを連携した放送内容から継承する方法について述べる．なお、Protégé では、RDF スキーマや OWL による推論だけでなく、Semantic Web Rule Language（SWRL）[5]によるデータ生成が可能であるため、Protégé の推論エンジンを利用して検証した．なお、使用した Protégé のバージョンは 5.2.0、推論エンジンは Pallet のバージョン 2.2.0 を使用した．

(1)　放送内容とアプリ連携のための推論処理

図 3 のインスタンス例の場合、放送内容とアプリを連携することは、tv:Scene クラスのインスタンスである S1 と、app:APP クラスのインスタンスである A111 を紐づけることを意味する．本稿では、この関係を tv:hasRelation とし、rdfs:domain に tv:Scene を、rdfs:range に app:APP となる関係とした．この tv:hasRelation を、SWRL 推論により生成した．図 4 は、番組開始 45 秒目にある場所に関するシーンと、地図系のアプリを連携させるためのルールと生成されるインスタンス例を示す．

図 4　SWRL ルールと生成されるトリプル

図 4 の例では、アプリ情報側のクラス定義である app:Place と、放送内容側のクラス定義である tv:SightseeingSpot のインスタンス同士を、tv:hasRelation 関係でトリプル化するルールとなる．つまり、双方のクラス定義を把握しておく必要がある上に、クラスの組み合わせ

毎にルールが必要となる．そこで、tv 及び app 名前空間のクラスを、DBpedia といった外部ソースのクラスに対して、rdfs:SubClassOf により関係づけることで、上記課題を解決できる．図 1 に示す通り、app:Place や tv:SightseeingSpot は、dbo:Loaction のサブクラスとして定義しているため、図 4 の推論ルールは図 5 のように変更することができる．各々の名前空間上のクラス定義を記述する必要があった推論ルールが、外部オントロジーのクラス定義に共通化できていることが分かる．これは、RDF スキーマ推論により、rdfs:SubClassOf 関係にもとづき、tv:SightseeingSpot のインスタンス S1 や app:Place のインスタンス A111 が、同時に dbo:Location のインスタンスであることを示すデータが生成されているため、図 5 の SWRL ルールが成立する．

```
app:App(?app)^app:hasAppConfig(?app,?ac)^app:detectedBy(?ac,?ad)^dbo:Location(?ad)
^tv:Scene(?s) ^tv:hasEntity(?s, ?e)^ dbo:Location(?e)
 -> tv:hasRelation(?app, ?s)
```

図 5　更新した SWRL ルール

上記推論により更新されたインスタンスを図 6 に示す．「厳島神社」が紹介されたシーンが、地図系アプリとして定義された GoogleMap が紐づき、レシピを探すことができる「みんなのきょうの料理」が紐づいていないことが分かる．

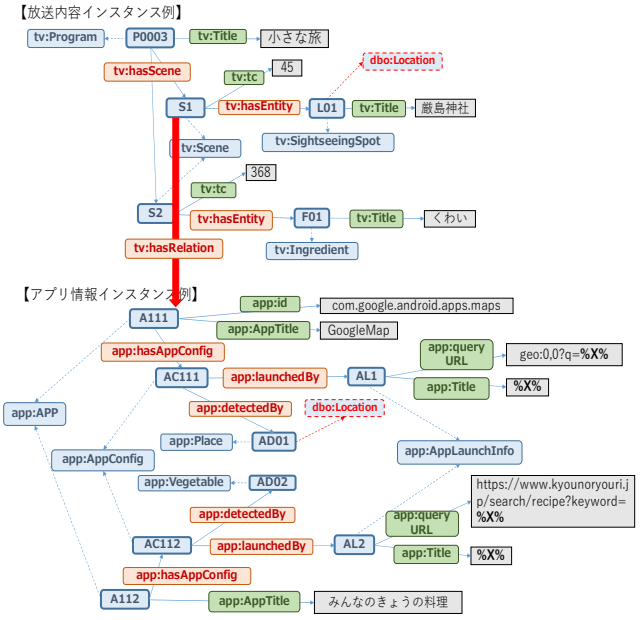

図 6　生成されたインスタンス

(2)　連携されたアプリを起動するための推論処理

上記の放送内容とアプリの連携だけでは、連携されたアプリを起動することが可能となるだけであり、起動後にユーザーによる手動入力が残る．そこで、推論により起動パラメータを放送内容から生成する．具体的には、app:LaunchInfo クラスのインスタンスに、app:inferredQuery プロパティとして起動情報を生成する．例えば、厳島神社が紹介されたシーンで、GoogleMap を起動する場合は、「geo:0,0?q=厳島神社」が生成される．図 7 に SWRL による推論ルールと生成されるトリプルと Protégé 上での結果画面を示す．SWRL にはサブセットとして、文字列操作も実装されているため、図 7 に示すように、swrlb:repalce によりパラメータ変数を放送内容で入れ替えることが可能となった．

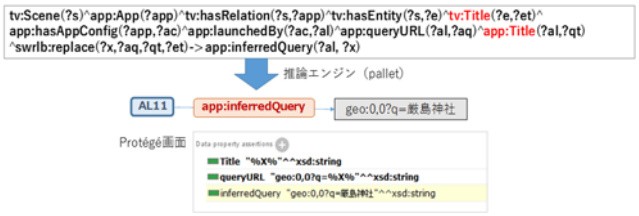

図 7 SWRL ルールと生成されるトリプル

放送内容とアプリを連携する際に生じた課題は起動情報設定推論においても同様に発生する．ここでも、tv:Title や app:Title を外部語彙に対して rdfs:SubPropertyOf 関係を事前に設定しておくことにより、RDF スキーマ推論を活用し図 8 のように共通化できる．図 8 で参照する外部語彙を Dublin Core(dc)とした．

```
tv:Scene(?s)^app:App(?app)^tv:hasRelation(?s,?app)^tv:hasEntity(?s,?e)^dc:title(?e,?et)^
app:hasAppConfig(?app,?ac)^app:launchedBy(?ac,?al)^app:queryURL(?al,?aq)^dc:title(?al,?qt)
^swrlb:replace(?x,?aq,?qt,?et)-> app:inferredQuery(?al, ?x)
```

※dc=http://purl.org/dc/elements/1.1/

図 8　更新された SWRL ルール

上記推論により更新されたインスタンスを図 9 に示す．

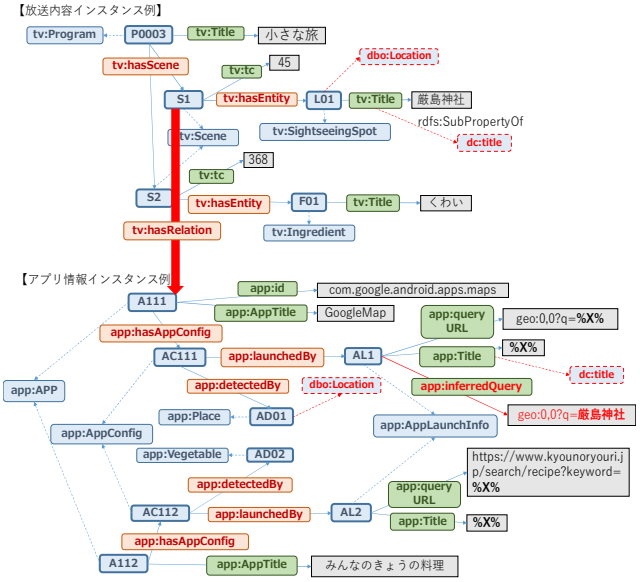

図 9　生成されたインスタンス例

同様に、その他のクラスから、放送内容とアプリを連携させ、アプリ起動情報を生成することができる．例えば、図 10 のように、dbo:Food クラスにもとづく推論ルールを追加することで、番組のシーン S2 とアプリ A112 を tv:hasRelation で連携させ、アプリ起動情報である Al2 に、「https://www.kyounoryouri.jp/search/recipe?keyword=くわい」を app:inferredQuery として生成することができる．これにより、「くわい」が紹介されたシーンにおいて、食材を扱うアプリである「みんなのきょうの料理」を「くわい」をセットした状態で起動することができる．

```
app:App(?app)^app:hasAppConfig(?app,?ac)^app:detectedBy(?ac,?ad)^dbo:Food(?ad)
^tv:Scene(?s) ^tv:hasEntity(?s, ?e)^ dbo:Food(?e)
 -> tv:hasRelation(?app, ?s)
```

図 10　Food クラスのための SWRL ルール

このように、クラス定義数だけ SWRL ルールを作成することで、サービス間の連携が可能となる.

4. アプリケーション試作

推論処理により生成された RDF データをもとに、番組連動型アプリ起動ソフトウェアを試作した. Protégé で生成した OWL ファイルを、RDF ストア（Fuseki）fにて登録し SPARQL[6]クエリにより取得可能とした. システム構成を図 11 に示す. なお、放送を疑似的に再現するため、動画再生クライアントが WEB サーバ内にある番組動画を再生し、その再生位置を番組連動型アプリが取得することで、放送中のシーンを取得することにした.

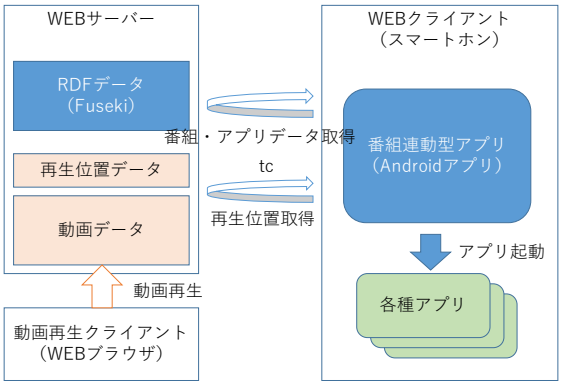

図 11　システム構成

図 12 に、番組連動型アプリの画面及び、アスパラガスが紹介されたシーンにおけるアプリ起動のイメージを示す. アスパラガスが dbo:Food クラスのインスタンスであることを示すため、画面内に「食材」と記され、食材を扱えるアプリが候補として表示される. なお、「買う」や「レシピを探す」というデータもあらかじめ RDF 化としておくことで、「食材」に対して「買う」アプリや、「レシピを探す」アプリをグループ化して表示している. 例えば、図 9 中にある AD01 に、app:hasAction プロパティとして「買う」というリテラル型データをトリプルとして紐づけることで可能となる.

画面内のアプリアイコンを押下すると、紹介された「アスパラガス」がパラメータとしてセットされた状態で、レシピを探すことができる「みんなのきょうの料理」が起動される.

なお、新しいアプリは、app:APP、apphasAppConfig、app:detectedBy、app:launchedBy クラスのインスタンス設定作業のみで追加することが可能である.

図 12　試作アプリケーション画面

5. 考察

5.1 クラス定義による連携処理

本稿では、放送内容とアプリを、放送で紹介されている内容そのものであるエンティティではなく、エンティティが属するクラスに抽象化して連携した. これは、例えば、「厳島神社」を調べる場合は、ある特定のアプリといった個別具体的な利用ではなく、「場所」を調べる時には地図系アプリ、「書籍」を買うときには E コマースアプリ、「食材」を買うときにはネットスーパーアプリを起動する、といった行動時の思考を反映することができる. この考えは、データ管理という観点でも効率的な運用を可能とする. 例えば、エンティティベースでサービス連携をする場合、DBpedia リソースへのエンティティリンキングを行った上で、双方のサービスが同じ DBpedia リソース経由で連携することが一例としてあげられるが、一意に特定されない場合が多くエンティティリンキング処理自体に多くのコストがかかる. その一方で、本稿で提案するクラスベースの連携は、エンティティのクラスを定義するのみで良い. また、概念構造を持つクラスによる推論処理が可能であるため、複数のデータ空間内のデータの連携が容易である.

5.2 推論によるデータ生成

本稿では、rdfs:SubClassOf や rdfs:SubPropertyOf といった RDF スキーマや、SWRL による推論を行い RDF データの生成を行った. 複数事業者のデータを連携する場合、それぞれのクラス定義やプロパティ定義を把握しておき、全ての組み合わせを推論処理で記述する必要がある. その点、RDF スキーマにより外部のクラスやプロパティ定義と連携しておくことにより、推論処理が共通化される. さらに、連携したくないデータがある場合は、そのエンティティのクラスを外部クラスに連携しないことで制御可能となる.

f https://jena.apache.org/index.html

例えば、放送内容の tv:Ingredient クラスをアプリに連携したくない場合は、当該クラスを dbo:Food といった外部クラスと rdfs:SubClassOf 関係を作らなければ、推論処理により連携されることはない.

放送内容とアプリを、クラス定義をもとに連携する処理と、連携したアプリを起動するためのクエリをプロパティ定義から生成する処理がない場合、図 12 に示したアプリケーションを開発する事業者が、放送内容を記述したデータストアと、アプリ情報を記述したデータストアそれぞれに、SPARQL クエリを作成して連携する処理を実装する必要があり、開発コストがかかる. 図 13 にクエリコードの比較を示す. 推論処理によりデータが生成されている場合は、app:inferredQuery プロパティにアプリ起動情報がセットされていることさえ把握しておけばよいが、推論されていない場合は、図 13 の下部に示す通り、データ構造を把握した上で、さらにはクラスやプロパティをもとに、起動情報を自ら生成するための文字列処理まで必要となる.

＜推論結果に対するSPARQLクエリ＞

```
select ?app ?new_query
{
?app app:inferredQuery ?new_query
}
```

＜推論結果がない場合のSPARQLクエリ＞

```
select ?app ?new_query
where{
 {
  select ?e ?et
  where
 {
  ?scene rdf:type tv:Scene.
  ?scene app:tc ?tc.
  FILTER(?tc=3422).
  ?scene tv:hasEntity ?e.
  ?e rdf:type dbo:Location.
  ?e tv:Title ?et.
 }
}
?app rdf:type app:App.
?app app:hasAppConfig ?ac.
?app app:detectedBy ?ad.
?ad rdf:type dbo:Location.
?ac app:launchedBy ?al.
?al app:hasQuery ?aq.
?aq app:Title ?qt.
?aq app:queryURL ?qu.
BIND(REPLACE(?qu,?qt,?et) as ?new_query)
}
```

※PREFIXは図2と同じ

図 13　推論結果の有無による SPARQL クエリの比較

ただし、SWRL による推論ルールでは連携できない場合がある. 例えば、緯度と経度が必要な場合など、アプリ起動パラメータが複数ある場合、swrlb:repalce を多段に組む必要がある. さらに、何段階必要かはパラメータ数によるため、SWRL ルール自体の生成処理を別途行う必要がある.

6. 関連研究

番組情報に関連する取り組みとして、番組画像の関連付けを SWRL 推論により生成している[7]が、単一事業者内のデータの拡張という観点での推論処理となっている. OWL や SWRL を使い、申請情報の妥当性検証システムの提案もなされている事例もある[8]. ここでは、直接的もしくや間接的にクラスに帰結するかを推論ルールにより判定している. 本論文同様に、OWL や SWRL といった標準化された推論記述では網羅できないため拡張ルールを設けている. ただし、クラスに属するかどうかで妥当性を判定するシステムであり、本論文が対象とするデータの連携とは異なる. ルールベースでオンライン上のコミュニケーションを連携させる研究もおこなわれている[9]. ここでは、例えば、News クラスの記事が Facebook に公開されたら、Twitter に公開するといったルールによる連携を可能としている. クラスにもとづき次の行動につなげる点では本研究と類似するが、次の行動に受け渡す情報の定義等は考慮されていない. ルールベースの他の研究として、ドアが開いたら電気をつけるといったオフィス空間における連携処理をオントロジーにより実現するための研究[10]でも同様に、次の行動にどのような情報を連携するかまでは考慮されていない.

7. おわりに

放送とインターネットを連携させる取り組みが注目を集めている中、放送内容に合わせて視聴者の行動を促す仕組みを実現するため、オントロジーによるデータ構造化及び推論によるデータ連携方法について提案した. エンティティベースではなくクラスベースでの連携により、推論処理を利用できるだけでなく、連携したくないデータの制御も可能となる. また、SWRL によりアプリ起動パラメータの動的な生成等も実現したが、その一方で複数パラメータの処理に関する課題も明らかになった. 今後は、推論ルールの生成処理の自動化にも取り組んでいく.

参考文献

[1] “RDF Primer”. https://www.w3.org/TR/2004/REC-rdf-primer-20040210/, (参照 2017-06-19).
[2] “OWL Web Ontology Language Reference “. https://www.w3.org/TR/owl-ref/, (参照 2017-06-19).
[3] “オントロジーと知識処理”. http://www.ei.sanken.osaka-u.ac.jp/pub/miz/bit99.pdf, (参照 2017-06-19).
[4] “オントロジーエディタ Protege”. http://protege.stanford.edu/, (参照 2017-06-19).
[5] “A Semantic Web Rule Language Combining OWL and RuleML “. https://www.w3.org/Submission/SWRL/, (参照 2017-06-19).
[6] ” SPARQL 1.1 Query Language”. https://www.w3.org/TR/sparql11-query/,（参照 2017-06-19）
[7] ” 番組情報データベースの Linked Open Data 化の検討” . https://www.nhk.or.jp/strl/publica/rd/rd156/PDF/P48-54.pdf, (参照 2017-06-19).
[8] 乗松 真二,村上 研二,” OWL と SWRL を用いた不動産登記申請 妥当性検証システムの提案” .情報システム学会誌 Vol.

10, No. 1
[9] “On Using Semantically-Aware Rules for Efficient Online Communication “. http://oc.sti2.at/sites/default/files/Rules.pdf, (参照 2017-06-19).
[10] Sergio Muñoz, Antonio F. Llamas, Miguel Coronado, Carlos A. Iglesias,” Smart Office Automation Based on Semantic Event-Driven Rules”. Volume 21: Intelligent Environments 2016