構造化データの自動抽出に向けた変換処理フレームワークの提案

數見　拓朗†　　白井　徳仁†　　善明　晃由†

† 株式会社サイバーエージェント　〒 150–0042 東京都渋谷区宇田川町
E-mail: †{kazumi_takuro,shirai_norihito,zenmyo_teruyoshi}@cyberagent.co.jp

あらまし　ウェブドキュメントから抽出される構造化データは，知識ベースの強化やファセット検索の支援など，幅広い目的のために活用されている．構造化データの抽出作業は，与えられた非構造データに対するドメイン知識を利用して行うことが多く，試行錯誤を繰り返すため，開発の手間が大きい．本稿では，構造化データの抽出作業を効率的に扱えるようなツールを提案する．提案ツールでは，構造と値の抽出処理を分離することで，開発にかかる作業を短縮することを目指す．また，提案手法を実装し，当社が保有するブログエントリに対して，様々な構造化データを抽出した事例について述べる．

キーワード　情報抽出，構造化データ

1　はじめに

ウェブドキュメントから抽出される構造化データは，様々な場面で活用されている．具体的には，大規模な知識ベースを構築することを目的として，ウェブドキュメントから構造化データを抽出する例がある [1] [2]．また，Google 検索では，ウェブドキュメントの内容を正しく理解することを目的として，構造化データを付与することを推奨している [1]．

構造化データは計算機によって利用されるため，用途に応じて構造と記法が統一されている必要がある．実際のウェブドキュメントは掲載されているサイトや書き手に応じて形式が異なるため，構造化データの抽出処理はウェブドキュメントの形式に応じて実装する必要がある．また，同じ形式のウェブドキュメントであっても，用途に応じて異なる構造化データを抽出する必要があり，別の抽出処理を実装する必要がある．

また，自然言語などの非構造化データから構造化データを抽出する場合，構造化データの項目には，その抽出に自然言語処理や統計的処理などの処理を必要とするものがある．例えば，キーワード抽出 [3] や TF-IDF などである．

例として，コード 1 やコード 2 のように，異なる HTML 構造を持つウェブドキュメントに対して，料理名を抽出することを考える．あるウェブドキュメントには，`title` タグのテキストノードに料理名が記載されているとする．一方で，別のウェブドキュメントには，`h1` タグのテキストノードに料理名が記載されているとする．XPath の指定などにより，それぞれのウェブドキュメントから料理名を抽出することは可能であるが，異なる DOM 構造ごとに抽出ルールを設定するのは現実的ではない．加えて，用途によっては料理名の他に料理画像や料理手順を抽出する場合 [9] があり，それぞれの属性に応じた抽出ルールを記述する必要がある．

また，抽出ルールにおいて，用途や抽出性能に応じて抽出方法を組合せ，変更する場合があり，複雑化することがある．例えば，レシピ検索を構築することを目的として，ウェブドキュメントからレシピの構造化データを抽出する場合，抽出された文字列が同義かどうかを判定する必要がある [10]．XPath の指定などにより，コード 1 とコード 2 から，材料名として「馬鈴薯」と「じゃがいも」を抽出することは可能である．しかし，検索アプリケーションに適用する場合には，文字列を抽出する処理に加えて，抽出された文字列が同義であるかどうかを判定する処理を組合せなければならない．さらに, [8] などの様々な手法により同義語の判定性能は大きく変わるため，適用する処理を柔軟に変更できることが望ましい．

情報抽出の分野では，wrapper induciton [4] に基づく手法がよく利用されるが，複雑な処理を含む構造化データの抽出処理の保守性を目的としたものではない．wrapper induction は，

```html
<!doctype html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <!--料理名-->
  <title>お好み焼き</title>
</head>
<body>
<div class="entry-text">
  <p>本文</p>
  <!--料理画像-->
  <img src="..." />
  <img src="..." />
  <ul>
    <!--料理手順-->
    <li>手順 1</li>
    <li>手順 2</li>
  </ul>
  <ul>
    <!--材料-->
    <li>馬鈴薯</li>
  </ul>
</div>
</body>
</html>
```

コード 1: HTML スニペット 1

1：https://www.youtube.com/watch?v=bUwmHX_EPmw（最終アクセス日：2020 年 12 月 23 日）

```html
<!doctype html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>タイトル</title>
</head>
<body>
<div class="entry-text">
  <!--料理名-->
  <h1>肉じゃが</h1>
  <!--料理手順-->
  <div>
    1. 手順 1。2. 手順 2。
  </div>
  <div>
    3. 手順 3。4. 手順 4。
  </div>
  <ul>
    <!--材料-->
    <li>じゃがいも</li>
  </ul>
</div>
</body>
</html>
```

コード 2: HTML スニペット 2

ウェブドキュメントからどのように情報を抽出するかについて，独自の抽出言語を提供し [5] [6]，抽出作業を効率化することが目的である．

本稿では，構造化データの抽出タスクにおいて保守性を高く維持できるツールを提案する．具体的には，構造と値の抽出方法を分けて管理し，構造に応じて抽出処理を柔軟に変更できるようにする．また，抽出処理における基本的なインターフェースを定義し，用途に応じて実装を切り替えることができるようにする．これらにより，構造と処理の独立性と各処理の独立性を保ち，再利用や保守を容易にする．

我々は，提案ツールを開発し社内での導入を進めている．本稿では，当社が保有する様々なブログエントリに対して，構造化データを抽出した事例を通して，提案ツールの適用可能性を示す．

本稿の貢献について述べる．第一に，構造と値の抽出方法を分離し，抽出方法の基本的なインターフェースを定義することで，高い保守性を維持するツールを提案した．第二に，構造と値の抽出方法を宣言的に記述し，構造化データを抽出するツールとして実装した．第三に，実際のデータに対して，提案ツールの適用可能性を示した．

本稿の構成は以下の通りである．第 2 節において，関連研究を紹介をする．第 3 節では，提案ツールの詳細を述べる．第 4 節では，提案ツールの実装を説明する．第 5 節では，提案ツールを当社の保有するデータに適用した例を紹介する．第 6 節で，本稿のまとめを述べる．

2 関 連 研 究

ウェブドキュメントから情報を抽出するタスクは，wrapper induction に基づく手法によって行われることが多い．wrapper induction に関連する先行研究は，抽出対象となる文書に対して高い抽出正解率が期待できる方法の提案と，少ない人的労力で抽出を行う方法の提案が主である．例えば，WHISK [5] は，抽出対象となる文書の言語情報を利用して，抽出方法を提案している．また，STALKER [6] では，DOM 構造に代表される構造的な情報を利用した抽出方法を提案している．こうした方法を，対象となる文書ごとに人手で作成するのは手間のかかる作業である．そこで, [7] は少量の教師データを用いて，未知のウェブドキュメントに対しても頑健な抽出手法を提案している．一方で，本稿では，用途に応じて抽出方法を柔軟に変更し，組合せることのできるツールを提案している．

また，wrapper induction は，ウェブドキュメントから構造化データを抽出することを対象としていない．例えば，ウェブドキュメントから料理名・料理画像・料理手順を含む構造化データを抽出したい場合，それぞれの属性が，どのような方法によって抽出するかを定義しない．そこで，本稿で提案するツールは，構造と抽出方法との対応関係について明確な定義を与える．

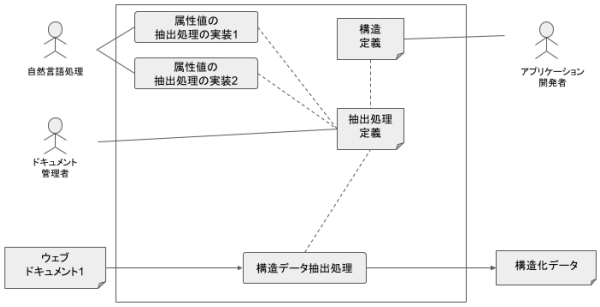

図 1　提案ツールの概要

3 提案ツール

提案ツールでは，構造化データの抽出タスクにおいて高い保守性を維持することを企図して，構造と抽出方法を分離し，抽出方法の基本的なインターフェースを定義する．ここで，構造とは属性と属性値の複数のペアであり，抽出方法とは属性値をどのように抽出するかを示す処理の工程である．構造と抽出方法を分離することで，構造と値の抽出方法の独立性が増す．また，処理の基本的なインターフェースを準備することで，抽出方法の各処理の独立性が増すように工夫している．

図 1 に提案ツールの概要を示す．提案ツールでは，構造化データを応用したアプリケーションを開発する技術者（以下，アプリケーション開発者）と自然言語処理技術を活用し属性値の抽出に必要な処理を開発する技術者（以下，自然言語処理開発者）の責務が分離される [2]．アプリケーション開発者は求める構造定義を記述したもの（図 1 の構造定義に対応）を本ツールに登録する．その際に，属性値の抽出に必要となる処理を定義するためのインタフェースを設定する．自然言語処理開発者は，自然言語処理の汎用的なパッケージや機械学習モデルを用いて，構造化データ定義に指定されたインタフェースに準拠し

2：画像処理などその他の分野の技術者により属性値の抽出処理が開発される場合もある．

た属性値の抽出処理を実装（図 1 の属性値の抽出処理の実装 1・2 に対応）し，ツールに登録する．

具体的な構造化データ抽出処理の定義は，対象のウェブドキュメントに関するドメイン知識をもった担当者（以下，ドキュメント管理者）によって設定（図 1 の抽出処理定義に対応）される．必要な構造やそのための処理が事前に登録されているため，ドキュメント管理者は，処理を組合わせるのみで構造化データの抽出を実現できる．

3.1 構造の定義

提案ツールで記述する構造の定義を述べる．レシピのように構造を持つものをジャンル g とし，構造化データの抽出対象となるウェブドキュメントを p とする．p に対して抽出する構造化データを $S_{p,g} \equiv \{(a_k, f_k(p)); k = 1, \ldots, K_g\}$ と定義する．ここで，a_k は料理名などの属性を表し，f_k は，対象となるウェブドキュメント p を入力とし，属性 a_k に対応した属性値を出力する関数とする．また K_g はジャンル g で定義される属性 a と属性の抽出方法 f の組の数である．

参考として，あるウェブドキュメント p に対して，料理名・料理画像・料理手順の 3 つの属性を含むレシピの構造化データを抽出する場合を考える．このとき，$g =$ レシピ であり，$K_g = 3$ である．料理名・料理画像・料理手順の 3 つの属性に対して，それぞれ抽出する方法 f を紐付ける．つまり，$S_{p,\ \text{レシピ}}$ は，

$$
\begin{aligned}
S_{p,\ \text{レシピ}} \equiv \{&(\text{料理名}, \text{料理名の抽出方法}),\\
&(\text{料理画像}, \text{料理画像の抽出方法}),\\
&(\text{料理手順}, \text{料理手順の抽出方法})\}
\end{aligned}
$$

となる．

3.2 処理の定義

構造化データの抽出には，構造の定義で説明をした関数 f に対して具体的な実装を与えることによって定義をする．そこで，値の抽出方法に関わる処理の定義について述べる．まず，h を型 A から型 B への写像 $h : \mathrm{A} \to \mathrm{B}$ とする．加えて，型 A から型 B へ写像 h の具体的な実装を $impl$ と表す．i 番目の処理を，$b_i \equiv (h, impl)$ と書く．属性 a_k の値の抽出方法は，処理の最小単位の列として，$f_k \equiv \{b_{n_k}\}$ と表現する．ここで，n_k は属性 a_k を抽出するのに必要な処理のステップ数を表す．

同じ写像 h である場合，抽出方法を切り替えるには h に対する実装 $impl$ を変更する．このような定義を用いることで，抽出方法の再利用を促し，柔軟に変更できるようにする．

4 実　　装

本節では，前節で説明した構造と処理の定義が，提案ツール上で設定する定義 DSL とどのように対応するか述べる．また，2 つの架空のウェブドキュメントに対して，同一の構造化データを抽出する場合において，いかに保守性を高めることができるかを説明する．ここで，コード 1 のような構造を持つエントリが投稿されているブログをブログ A と呼び，コード 2 のような構造を持つエントリが投稿されているブログをブログ B と呼ぶことにする．それぞれのブログに投稿されるエントリについて，料理名と料理手順の属性を持つ構造化データを抽出する例を用いて，抽出処理を再利用する場合と抽出処理を変更する場合について述べる．

```yaml
source:
  url: ...
  body: /html

structures:
    - genre:
      - name: recipe
      - attributes:
        - name: recipe_name
          derivations:
            - action:
              - func_id: parser
              - args:
                  - $source.body
            - action:
              - func_id: xpath_extractor
```

コード 3: データソースと構造に関する定義 DSL

```yaml
elements:
  - name: document
    fields:
      - name: body
        type: org.jsoup.nodes.Document
  - name: recipe_name
    fields:
      - name: name
        type: string

functions:
  - id: parser
    arg_type: string
    return_type: document
    impl: j.c.c.m.HTMLParser.java
  - id: xpath_extractor
    arg_type: document
    return_type: recipe_name
    impl: j.c.c.m.XPathExtractor.java
    config:
      - path: /html/head/title
```

コード 4: 型と処理に関する定義 DSL

4.1 定義 DSL

提案ツールでは，ウェブドキュメントに対して，どのような構造化データを，どのような処理工程で抽出するかを，定義 DSL に記述する．定義 DSL は，コード 3・4 から構成される．定義 DSL の設定項目は，図 1 のアプリケーション開発者・自然言語処理開発者・ドキュメント管理者の設定する項目に対応している．

コード 3 の `source` 項目には，対象となるウェブドキュメントを記述する．`source` 項目には，対象となるウェブドキュメントの URL を記載し，テキスト本文の位置を XPath などで指定する．この作業は，ドキュメント管理者が，どのウェブドキュメントを対象にするかを選択することに対応する．

コード 4 の `elements` 項目には，関数の入力と出力で必要な型を記載する．これは，写像 h の定義で必要な型に対応する．コード 4 では，料理名を抽出するにあたって必要な型を，

document 型と recipe_name 型の 2 つを設定している．この作業は，自然言語処理開発者が属性値抽出に必要なインターフェースを定義することに対応する．

コード 4 の functions 項目には関数名・引数の型・返り値の型・実装パスを指定する．これは，抽出処理 f を構成する b に対応する．この作業は，自然言語処理開発者が属性値を抽出するためのインターフェースと，その実装を開発することに対応する．

コード 3 の structures 項目には，ジャンル名，属性と値の抽出方法を記載する．structures 項目配下の derivations 項目には，functions 項目で登録をした関数を用いて抽出手順を記載する．これは f に対応する．structures 項目に設定する内容は，$S_{p,g}$ に対応する．この作業は，アプリケーション開発者が，抽出したい構造を定義することに対応する．

コード 3・4 では，source 項目に設定をしたテキスト本文を表す body が示す位置の DOM 構造を解析し，XPath である /html/head/title にあるテキストノードを取り出す処理を記述している．構造が同一で抽出処理が異なる場合，コード 4 の functions 項目を変更することで実現できる．

4.2 処理の再利用

形式の異なるブログエントリからの料理名の抽出を例に，提案ツールにおいて処理の再利用を容易に行えることを説明する．ブログ A のエントリでは，料理名が title タグのテキストノードにある一方で，ブログ B のエントリでは，料理名が h1 タグのテキストノードにある．したがって，ブログ A とブログ B に投稿されるエントリで目的となる文字列を抽出するには，対象となる HTML 文書を解析し，XPath などによりテキストノードを指定することで実現できる．

まず，引数の型が string 型であり，返り値が document 型であるような写像を h_1 とし，引数の型が document 型であり，返り値が recipe_name 型であるような写像を h_2 とする．次に，h_1 に実装 j.c.c.m.HTMLParser.java を紐付け，$impl_1$ とする．同様に，h_2 に実装 j.c.c.m.XPathExtractor.java を紐付け，$impl_2$ とする．b_i は，

$$b_1 = (h_1, impl_1)$$
$$b_2 = (h_2, impl_2)$$

と表現できる．料理名を抽出する処理は，$f_1 = \{b_1, b_2\}$ の列として記述ができる．また，j.c.c.m.XPathExtractor.java は，指定された XPath から，任意のノードを抽出する実装であるとする．

コード 5 は，ブログ A のエントリからレシピの構造化データを抽出するための定義 DSL であり，コード 6 はブログ B のエントリからレシピの構造化データを抽出するための定義 DSL である．structures 項目の料理名を抽出する処理を記述する derivations 項目は両方の DSL で共通である．derivations 項目には，f_1 が記載されている．functions 項目には，f_1 の具体的な要素である b_1 と b_2 が記載されている．b_1 と b_2 には共通の実装である j.c.c.m.HTMLParser.java と j.c.c.m.XPathExtractor.java がそれぞれ紐付いている．このように，共通の処理ルールを見出すことができれば，対象となるウェブドキュメントが異なるときでも，処理の再利用を容易に行うことができる．

4.3 処理の変更

形式の異なるブログエントリからの料理手順の抽出を例に，提案ツールにおいて処理の変更を容易に行えることを説明する．ブログ A のエントリでは，料理手順が ul タグを用いて箇条書きで記述されるのに対して，ブログ B のエントリでは，div タグの複数のテキストノードに記述されている．

両方のブログエントリともに，料理手順を抽出するインターフェースは共通であるが，抽出方法は異なる．つまり，写像 h の入出力の型は共通であるが，どのような実装 $impl$ を紐付けるかが異なる．ここで，引数の型が document 型であり，返り値が recipe_instruction 型のリスト型であるような写像を h_3 とする．ブログ A のエントリから料理手順を抽出する実装を j.c.c.m.RecipeInstructionExtractorA.java として，これを $impl_{3,A}$ とする．同様に，ブログ B のエントリから料理手順を抽出する実装を j.c.c.m.RecipeInstructionExtractorB.java として，これを $impl_{3,B}$ とする．ブログ A のエントリから料理手順を抽出する処理 $b_{3,A}$ と，ブログ B のエントリから料理手順を抽出する処理 $b_{3,B}$ は，それぞれ

$$b_{3,A} = (h_3, impl_{3,A})$$
$$b_{3,B} = (h_3, impl_{3,B})$$

と表現できる．コード 5 とコード 6 の functions 配下の impl 項目に紐付ける実装を記載することで処理の切り替えを行う．このように，処理が異なる場合でも，共通の処理のインターフェースに基づいた実装をそれぞれ紐付けることで，柔軟に処理を変更できるようにする．

5 適 用 例

本節では，提案ツールを当社ブログサービスの SEO 対策に適用した例に基づき，いかに保守性を高めることができるかを議論する．

これまでは，単純な HTML 文書を用いて提案ツールをどのように適用しているかを説明した．本節では，提案ツールを用いて当社が保有するブログのうち，4 つのブログを対象として，それぞれのブログエントリから構造化データを抽出する．

表 1 は抽出対象とした属性を表している．レシピの構造化データは，料理名・料理画像・分量・料理手順の 4 つの属性を含むものとした．ただし，料理名と料理画像は必ず含むべき属性とした．エントリによっては，複数のレシピを掲載しているものがあるが，それらは抽出の対象外とした．

例えば，4 つのブログに投稿されたエントリから料理画像を抽出することを考える．どのブログでも料理画像は，ブロガー自身がアップロードした画像の中で，エントリ中のトップに配置され

```yaml
functions:
  - id: parser
    arg_type: string
    return_type: document
    impl: j.c.c.m.HTMLParser.java
  - id: recipe_name_extractor
    arg_type: document
    return_type: recipe_name
    impl: j.c.c.m.XPathExtactor.java
    config:
      - path: /html/head/title/text()
  - id: recipe_instruction_extractor
    arg_type: document
    return_type: list
      - param: recipe_instruction
    impl: j.c.c.m.RecipeInstructionExtractorA.java

structures:
    - genre:
      - name: recipe
      - attributes:
        - name: recipe_name
          derivations:
            - action:
              - func_id: parser
              - args:
                  - $source.body
            - action:
              - func_id: recipe_name_extractor
        - name: recipe_instructions
          derivations:
            - action:
              - func_id: parser
              - args:
                  - $source.body
            - action:
              - func_id: recipe_instruction_extractor
```

コード 5: ブログ A のエントリに対応する定義 DSL

```yaml
functions:
  - id: parser
    arg_type: string
    return_type: document
    impl: j.c.c.m.HTMLParser.java
  - id: recipe_name_extractor
    arg_type: document
    return_type: recipe_name
    impl: j.c.c.m.XPathExtactor.java
    config:
      - path: /html/div/h1/text()
  - id: recipe_instruction_extractor
    arg_type: document
    return_type: list
      - param: recipe_instruction
    impl: j.c.c.m.RecipeInstructionExtractorB.java

structures:
    - genre:
      - name: recipe
      - attributes:
        - name: recipe_name
          derivations:
            - action:
              - func_id: parser
              - args:
                  - $source.body
            - action:
              - func_id: recipe_name_extractor
        - name: recipe_instructions
          derivations:
            - action:
              - func_id: parser
              - args:
                  - $source.body
            - action:
              - func_id: recipe_instruction_extractor
```

コード 6: ブログ B のエントリに対応する定義 DSL

属性	必須	型	例
料理名	○	文字列型	おいしいピザ！
料理画像	○	URL	https://...
分量		文字列型	4 人分
料理手順		文字列のリスト	[“ボウルに...”, ...]

表 1　レシピ構造化データの属性

ることが多い．そこで，エントリ中の `img` タグの `src` 属性の値の中で，ブロガーのユーザ名を含む先頭の値を取得する．4 つブログで処理を再利用できるように，XPath により，DOM のノードを抽出する処理（以下，`xpath_extractor`），正規表現により，任意の文字列を抽出する処理（以下，`regexp_extractor`），リストの要素から，先頭の要素を選択する処理（以下，`first`）を実装した．これらの処理を組合せることで，料理画像を抽出する．これを定義 DSL で表現するとコード 7 のようになる．ブログによって，`xpath_extractor` や `regexp_extractor` に設定をする XPath や正規表現パターンを対象ブログごとに変更することで，異なるブログであっても料理画像を抽出できるようにする．

料理名・料理手順・分量の抽出についても同様の処理を利用した．具体的には，料理名は `xpath_extractor` と `first` の処理を組合せ抽出した．料理手順は `regexp_extractor` を利用し，分量は `regexp_extractor` と `first` を組合せて抽出をした．

```yaml
structures:
    - genre:
      - name: recipe
      - attributes:
        - name: recipe_image
          derivations:
            - action:
              - func_id: parser
              - args:
                  - $source.body
            - action:
              - func_id: xpath_extractor
            - action:
              - func_id: regexp_extractor
            - action:
              - func_id: first
```

コード 7: 料理画像を抽出する例

表 2 は，対象としたブログとそれぞれの抽出結果を記載している．対象エントリ数とは，対象期間内にそれぞれのブログで投稿されたエントリの数である．レシピ構造化データの抽出数とは，対象エントリのうち，提案ツールで定義するルールによって抽出された構造化データの数である．正解数は，抽出された構造化データが，意図通りレシピの構造化データとなっていた数を表す．例えば，属性値が意図通りの値になっていないものは正解には数えない．対象としたブログでは，レシピエントリを記載する際に決まった DOM 構造を用いることが多く，

対象ブログ	対象期間	対象エントリ数	構造化データ抽出数（割合）	正解数（正解率）
ブログ 1	2020/7/1 – 2020/10/28	187 件	63 (33.689%)	63 (100%)
ブログ 2	2020/7/1 – 2020/10/28	273 件	30 (10.989%)	19 (63.333%)
ブログ 3	2020/11/1 – 2020/11/23	20 件	17 (85.000%)	17 (100%)
ブログ 4	2020/11/1 – 2020/11/23	15 件	4 (26.666%)	4 (100%)

表 2　対象データと抽出結果

ブログごとの単純なルールで抽出でき，正解率が高い．

今回の実験で対象とした実データは，単純な DOM 構造を持つ HTML 文書である．そのため，比較的単純な抽出処理の組合せによって，構造化データを抽出することができた．

提案ツールでは，ある属性値を抽出する際に，処理のインターフェースに実装を紐付けることによって，処理の再利用や変更を容易にしている．しかし，インターフェースの入出力の型さえ同じであれば，モノリシックな実装を紐付けることが可能である．処理を再利用可能で変更容易なものとするには，条件分岐や繰り返しなどの制御フローを整えるなど，各処理の独立性を高めるような工夫が必要である．

6 ま と め

本稿では，構造化データの抽出タスクにおいて保守性を高く維持するツールを提案した．提案ツールでは，構造と値の抽出方法を分離し，抽出処理における基本的なインターフェースを定義した．これにより，構造と抽出処理の独立性と各処理の独立性が増し，再利用性や保守性を高く維持できるよう試みた．また，当社の保有するブログエントリに対して提案ツールを適用した．構造化データを抽出し，どのように活用しているかを紹介した．今回は，XPath や正規表現による抽出例のみであったが，今後は，様々な用途に対応できるように機械学習を用いた抽出処理などを組合せることを検討している．

文　献

[1] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang, “From Data Fusion to Knowledge Fusion,” Proceedings of the VLDB Endowment, Vol. 7, No. 10, pp. 881–892, 2014.

[2] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang, “Knowledge vault: a web-scale approach to probabilistic knowledge fusion,” Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD), pp. 601–610 2014.

[3] Peter D. Turney, “Learning algorithms for keyphrase extraction,” Information retrieval, Vol. 2, Issue. 4, pp. 303–336 2000.

[4] Sergio Flesca, Giuseppe Manco, Elio Masciari, Eugenio Rende, and Andrea Tagarelli, “Web wrapper induction: a brief survey,” AI Commun, Vol. 17, No. 2, pp. 57–61, 2004.

[5] Stephen Soderland, “Learning Information Extraction Rules for Semi-Structured and Free Text,” Machine Learning, Vol. 34, pp. 233–272, 1999.

[6] Ion Muslea, Steven Minton and Craig A. Knoblock, “Hierarchical Wrapper Induction for Semistructured Information Sources,” Autonomous Agents and Multi-Agent Systems, Vol. 4, pp. 93–114, 2001.

[7] Bill Yuchen Lin, Ying Sheng, Nguyen Vo, and Sandeep Tata, “FreeDOM: A Transferable Neural Architecture for Structured Information Extraction on Web Documents,” Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 1092–1102, 2020.

[8] Derek Tam, Nicholas Monath, Ari Kobren, Aaron Traylor, Rajarshi Das, and Andrew McCallum, “Optimal Transport-based Alignment of Learned Character Representations for String Similarity,” Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 5907—5917, 2019.

[9] 苅米志帆乃，藤井敦，“料理どうしの類似と組合せに基づく関連レシピ検索システム，” 言語処理学会第 14 回年次大会発表論文集, pp. 959–962, 2008.

[10] 安川美智子，“料理レシピに含まれる類似文字列の検索，” 自然言語処理研究会（NL）, 2012.