Web API ドキュメントからの情報抽出による プログラムライブラリ作成支援

筑波大学

図書館情報メディア研究科

2013 年 3 月

高井　正成

目次

第 1 章 はじめに

近年，Webアプリケーションを取り巻く環境は大きく変化している．通信技術の発達により大規模データの高速通信が可能になり，Web上で大規模なデータを容易に扱えるようになった．Webブラウザも高機能化し，Java ScriptやAdobe Flash，HTML5などの技術が登場しWeb上での表現方法が多様になり，ユーザインタフェースが強化された．それに伴いWebアプリケーションも，様々な機能を持つものが登場するようになり，その数も増加した．またWebアプリケーションはその機能をユーザがブラウザ上で利用できるだけでなく，他のアプリケーションに対してその機能を提供するようにもなった．提供にはWeb APIという形式がとられ，他のアプリケーションはHTTPなどの通信プロトコルを利用し，そのWeb APIの機能を利用する．Web APIの公開により複数のWeb APIを組み合わせてWebアプリケーションを構築するマッシュアップも盛んに行われるようになった．Web APIリポジトリであるProgrammable Web [1]には現在8000件以上のWeb APIが登録されており，Web APIの登録数は年々増加している．

Webアプリケーションの開発者はWeb APIを利用する場合，各Web APIの仕様を提供されるオンラインドキュメントなどで確認する．プログラミングの際はWeb APIを利用する処理はプログラムライブラリとして他の処理と分けて記述する場合が多い．Web APIではHTTPなどの通信プロトコルを用いたリクエスト・レスポンスにより通信を行うという大きな枠組みは共通しているものの，細かな仕様は各Web API提供者によって定められる．そのため各Web APIに互換性がなく，Webアプリケーション開発者はWeb APIを利用する場合，それぞれのWeb APIの仕様に沿ったプログラムライブラリを作成する必要がある．新たなWeb APIが公開された場合複数の開発者がそのWeb APIを利用しようとして，同様の機能を持った複数のプログラムライブラリが作成されてしまう．Web APIの仕様が変更された場合は，プログラムライブラリを修正しなければならないが，修正漏れがあったプログラムライブラリは動作不能になったり，機能が不足してしまったりすることもある．以上のような理由からWeb APIのプログラムライブラリを開発者が個別に作成するのは開発において非効率であるといえる．

そこで本研究では開発者がプログラムライブラリを自分で作成する手間を省くため，Web APIのオンラインドキュメントからプログラムライブラリの作成に必要な情報を機械的に抽出する手法を提案する．また実際に提案手法を実装したシステムを作成し，既存のWeb APIプログラムライブラリの作成に比べ提案手法がどの程度有効であるのか示す．本研究でいう開発者とは，Web APIを利用してWebアプリケーションの開発を目的としているプログラマのことを指すこととする．

第2章では本研究で問題としている点とその問題に関係する諸技術について説明し，第3章ではその問題を解決するための手法を提案する．第4章では，その提案手法が実際に問題

を解決できるかどうか試験的に確認するため行った予備実験について述べる．第5章では予備実験の結果を元に開発した，提案手法が実装されたプロトタイプシステムについて述べる．第6章では提案手法およびそれを実装したプロトタイプシステムの評価とそれに基づく検討について述べる．第7章では本研究と関連・類似研究との違いについて示し，第8章で本研究の結論と今後の展望について述べる．

第 2 章 Web アプリケーションとその開発手法

2.1 Web アプリケーションと Web API

本研究における Web アプリケーションとは，ユーザが Web ブラウザを用いて Web サーバにアクセスして情報の閲覧，検索，編集などのサービスを利用することができる， Web 上で提供されるアプリケーションを指す．提示される情報は単純に文字列によって表現されるわけではなく，図を用いたりレイアウトが工夫されたりと多様な表現方法で表される．例としてはマイクロブログサービスを提供する Twitter [2]，SNS サービスを提供する Facebook [3]，地図情報を提供する Google Map [4]などがある．Web アプリケーションは自身の機能を Web ブラウザを通じてユーザに提供する他に，自身の機能を他のアプリケーションから利用できるようにしているものも存在している．このように他のアプリケーションが Web を通じて自身の機能を利用してもらうようにする場合 Web API という仕組みを利用して情報の受け渡しが行われる．Web API は HTTP などの通信プロトコルを利用して他のアプリケーションが Web アプリケーションサーバに対して任意の処理を依頼し，その結果を受取るための仕組みである．Web API の利用例としては，自身の Web サイトに地図を埋め込む例が挙げられる．例えば食べログ [5]という飲食店紹介サイトでは，飲食店の情報としてその飲食店の位置が地図と共に示されている(図 2.1).

図 2.1 で表示されている地図の部分は Google Maps の Web API を用いて店舗の位置を表示している．図 2.1 の地図が表示される仕組みとしては，以下のようになる．

(1) ページ内の Java Script や PHP などのスクリプト言語を用いて記述されたプログラムが Google Maps API を利用して Google Maps に情報を要求する(リクエスト)
(2) Google Maps は要求に即した情報を返す(レスポンス)
(3) ページ内のプログラムにより，レスポンスから地図情報を作成する

以上を図で表したものを図 2.2 に示す．この例のように，Web API を利用して作成される Web アプリケーションは，Web API の仕様にあったにリクエストを送り，またその返答結果を解釈するためのプログラムが必要である．

Web API の数は年々増加しており，Programmable Web における年ごとの Web API 登録数の推移(図 2.3)を見ると，2000 年から現在まで，Web API の登録数は指数関数的に増加していることがわかる．Programmable Web に登録されている Web API のうち，2013 年 1 月現在で REST 型として登録されている Web API は 5385 件で，SOAP 型として登録されている Web API は 1804 件である．

図 2.1 食べログの飲食店店舗位置表示画面(参考文献 [5]から引用)

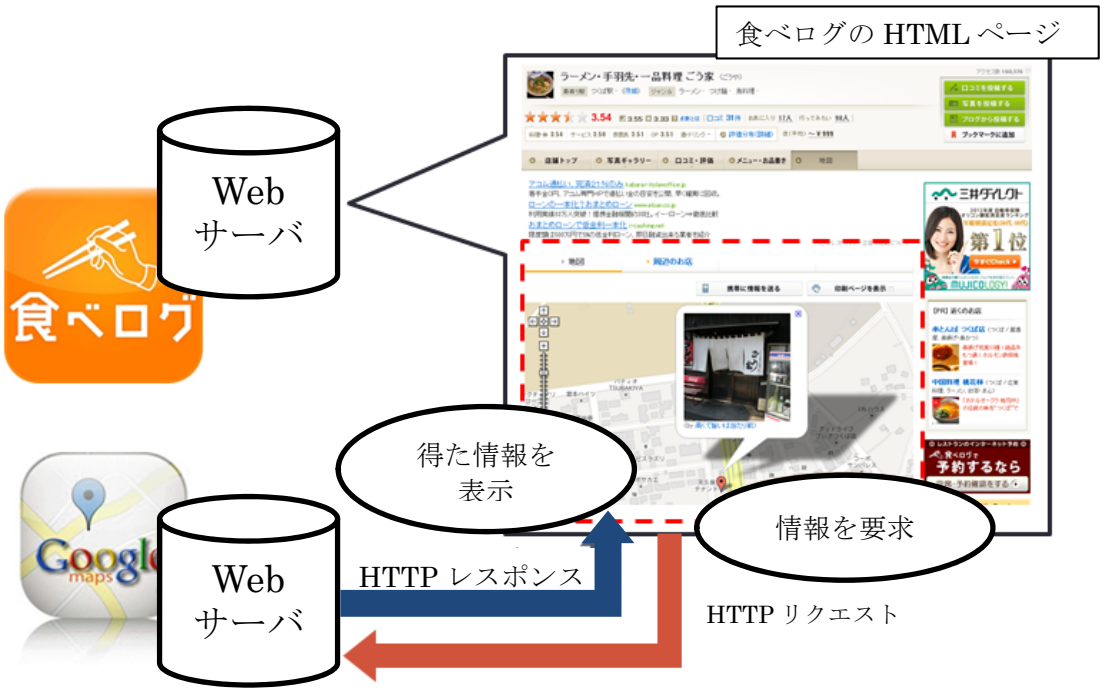


図 2.2 食べログの Web ページからの Google Maps API の利用

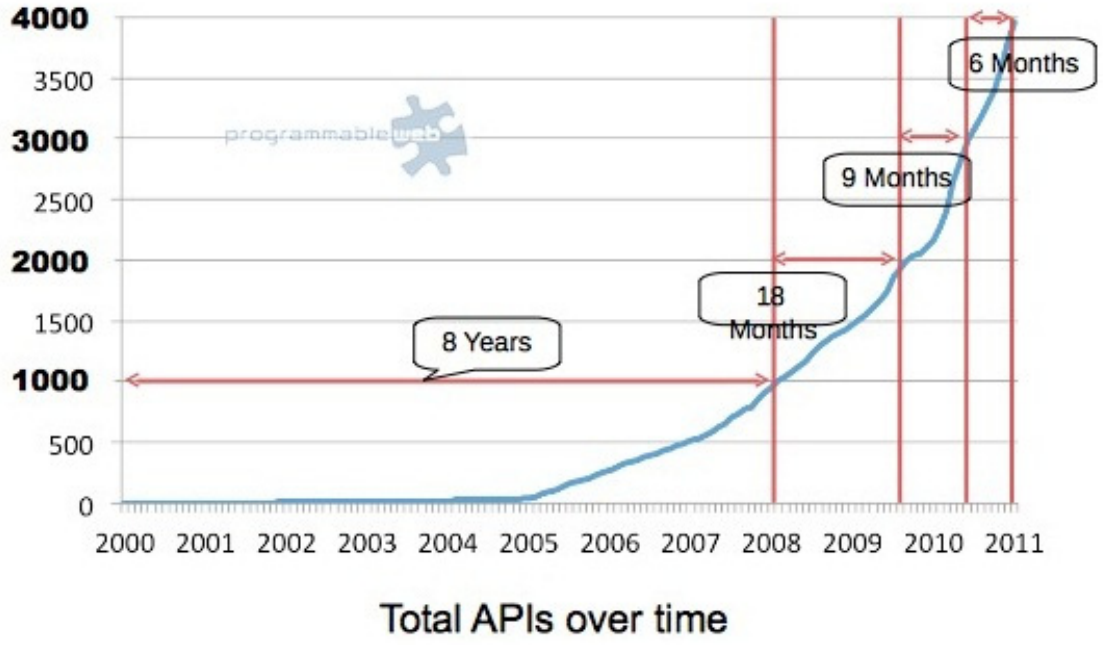


図2.3 Programmable Webにおける年ごとのWeb API登録数の推移(参考文献 [1]から引用)

2.2 Web API の種類

Web API には大別して SOAP(Simple Object Access Protocol) [6] 型と REST(Representational State Transfer) [7]型と呼ばれるものがある.

SOAP 型は XML 形式で示した「SOAP エンベロープ」を用いて，Web アプリケーションと他のアプリケーションが相互に通信を行う手法である．通信の際にはこの SOAP エンベロープを HTTP を利用して送受信を行う．SOAP エンベロープの内容は使用出来る属性やおおまかな構造(SOAP ヘッダと SOAP ボディを持つ，など)は決まっているが，詳細は各 Web API ごとに異なる.

REST 型は HTTP リクエストの URI によって欲しい情報を指定し，XML や JSON などの構造的なデータとしてレスポンスを受け取る手法である．特に HTTP の GET メソッドが使われる場合が多く，その場合情報の指定はリクエスト URI 中で行われるという特徴がある．本研究では，REST は Roy Fielding が提唱した REST の原義 [7]ではなく，以上で示した Web API のアーキテクチャを述べる際に一般的に言われる Web API の型を指すこととする.

SOAP 型は SMTP や HTTP などのプロトコルにより利用することが可能であるが，特に HTTP の POST メソッドが使われることが多い．SOAP 型におけるリクエスト・レスポンスの例を表 2.1 に示す.

表 2.1 SOAP 型におけるリクエスト・レスポンスの例([8]から引用)

リクエスト	POST /WebSite1/WebService.asmx HTTP/1.1 Host: localhost Content-Type: text/xml; charset=utf-8 Content-Length: **length** SOAPAction: "http://tempuri.org/getHello" <?xml version="1.0" encoding="utf-8"?> <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> <soap:Body> <getHello xmlns="http://tempuri.org/"> <str>**string**</str> </getHello> </soap:Body> </soap:Envelope>
レスポンス	HTTP/1.1 200 OK Content-Type: text/xml; charset=utf-8 Content-Length: **length** <?xml version="1.0" encoding="utf-8"?> <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> <soap:Body> <getHelloResponse xmlns="http://tempuri.org/"> <getHelloResult>**string**</getHelloResult> </getHelloResponse> </soap:Body> </soap:Envelope>

REST 型におけるリクエスト・レスポンスの例を表 2.2 に示す．GET メソッドにより指定される URL はベース URL を指定する部分と，パラメータを指定する部分の 2 つによって構成される．リクエスト URL は以下のような形式となる．

http://[ベース URL]?[パラメータ名 1]=[値 1]&[パラメータ名 2]=[値 2]

ベース URL とパラメータの指定の間には「?」を入力し，パラメータ名とその値の間には「=」を入力する．ベース URL は必ず入力する必要があるが，パラメータは指定できる個数が Web API によって異なる．REST 型ではレスポンスは XML や JSON [9]などによって情報が構造的に記述されたものが返却される．本論文ではレスポンスとして返ってくるこれらの情報が構造的に記述されたものを便宜上「構造データ」と呼ぶことにする．

表 2.2 REST 型におけるリクエスト・レスポンスの例([8]から引用)

リクエスト	GET /WebSite1/WebService.asmx/getHello?**str=string** HTTP/1.1 Host: localhost
レスポンス	HTTP/1.1 200 OK Content-Type: text/xml; charset=utf-8 Content-Length: **length** <?xml version="1.0" encoding="utf-8"?> <string xmlns="http://tempuri.org/">**string**</string>

REST 型は HTTP の GET メソッドを利用していることが多く，SOAP 型に比べてシンプルな構造となっている. Web ブラウザに URL を入力するだけで動作確認が行うことが可能なので利用や確認が容易である．SOAP 型は REST 型に比べて構造が複雑であるが，その分リクエストとして REST 型より多量の情報を送ることが可能であり，WS(Web Services)パッケージ [10]などの SOAP の機能拡張のための仕様も豊富に存在する．WS パッケージを用いることで実運用上の多様なケースに対応した機能を SOAP 型によって利用することができるようになったが，その分 WS パッケージに対応しているプログラムライブラリの導入が必要となる．Web API を利用する際，WS パッケージに対応しているプログラムライブラリが開発者の環境(OS やプログラミング言語)において存在しない場合もあるため，WS パッケージを用いた SOAP 型の Web API は多様な開発環境に対応しているとはいえない. 対して REST 型では HTTP のメソッドを用いたシンプルな構造となっており，SOAP 型のように特定のパッケージの仕様を解釈するためにライブラリを導入する必要はない.

2.3 Web API の仕様記述言語

Web API にはその仕様をコンピュータに入力として与え，仕様を解釈させるための仕様

記述言語が存在する．SOAP 型の Web API の仕様を記述する言語として WSDL(Web Services Description Language) [11]がある．REST 型の Web API の仕様を記述する言語としては WADL(Web Application Description Language) [12]がある．

2.3.1 WSDL

WSDL 文書は XML によって記述される．WSDL を構成する主要な要素を表 2.3 に示す．

表 2.3 WSDL を構成する主要な要素([13]から引用)

要素名	要素の説明
wsdl:definitions 要素	WSDL 文書の最上位要素
wsdl:types 要素	メッセージのフォーマットを定義する際に使用する型を，抽象的に定義する．wsdl:types 要素は省略可能だが，メッセージ定義にユーザ定義の型を使用する場合は，ここで型を定義しておく必要がある
wsdl:message 要素	Web サービスで使用するメッセージのフォーマットを抽象的に定義する．wsdl:types 要素で定義された型はここで使用する
wsdl:operation 要素	入出力メッセージや，エラー情報を通知するために使用するフォルトメッセージのフォーマットとして， wsdl:message 要素で定義されたフォーマットを割り当てる．入力と出力とフォルトメッセージ出力を行う処理の 1 単位である操作（operation）を抽象的に定義する
wsdl:portType 要素	関連する操作をひとまとめにした抽象的なポートである，ポートタイプ(portType)を定義する
wsdl:binding 要素	wsdl:portType 要素で定義されたポートタイプ内の個々の抽象的な操作に，具体的な通信プロトコルをバインドする．ポートタイプの定義に通信プロトコルをバインドした定義のことをバインディング(binding)と呼ぶ．具体的な通信プロトコルへのバインドは，拡張性要素を使用して記述する
wsdl:port 要素	wsdl:binding 要素で定義されたバインディングに，通信エンドポイントのネットワークアドレスをバインドして具体的なポートを定義する．ネットワークアドレスのバインドは拡張性要素を使用して記述する
wsdl:service 要素	wsdl:port 要素で定義したポートのうち，関連するポートをひとまとめにしたサービスを定義する

WSDL 文書の実際の記述例を示す．Web サービスとして，ショッピングサイトで，ある商品の値段を計算して返却する Web サービスが運用されていると考える．この Web サービスは商品コード(Code)と商品の個数(Value)を渡すと価格を返す機能を持つとする．この Web サービスのサービスを要求する場合，HTTP で SOAP メッセージを送る．図 2.4 にそのメッセージの例を示す．

図 2.4 HTTP による SOAP メッセージ例

```
POST /axis/services/Estimate HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1beta
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: …

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getPrice
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://unitec-denki.utj.co.jp/schema/2003/">
      <Code xsi:type="xsd:string">code-001</Code>
      <Value xsi:type="xsd:int">30</Value>
    </ns1:getPrice>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP メッセージは SOAP がバインディングされているプロトコル(ここでは HTTP)のヘッダを記述した後，SOAP の要求記述を行う．要求は SOAP エンベロープにまとめられており，その中に SOAP ヘッダと SOAP 本体(ボディ)を記述する．この例では，商品コー

ドが「code-001」の商品を 30 個購入する場合の価格を要求している．これに対するレスポンスは図 2.5 のようになる．価格として 1575 円が返されている．

図 2.5 SOAP メッセージのレスポンス例

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Wed, 12 Mar 2003 15:06:08 GMT
Server: Apache Coyote/1.0
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getPriceResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://unitec-denki.utj.co.jp/schema/2003/">
      <getPriceReturn xsi:type="xsd:int">1575</getPriceReturn>
    </ns1:getPriceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

これらの SOAP メッセージをやりとりする Web サービスの仕様は，図 2.6 の WSDL 文書で記述できる．

図 2.6 WSDL 文書例

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://unitec-denki.utj.co.jp/schema/2003/"
xmlns:impl="http://unitec-denki.utj.co.jp/schema/2003/"
xmlns:intf="http://unitec-denki.utj.co.jp/schema/2003/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<wsdl:message name="getPriceRequest">
<wsdl:part name="Code" type="xsd:string"/>
  <wsdl:part name="Value" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="getPriceResponse">
  <wsdl:part name="getPriceReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:portType name="Estimate">
  <wsdl:operation name="getPrice" parameterOrder="Code Value">
    <wsdl:input name="getPriceRequest" message="impl:getPriceRequest"/>
    <wsdl:output name="getPriceResponse" message="impl:getPriceResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="EstimateSoapBinding" type="impl:Estimate">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getPrice">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getPriceRequest">
      <wsdlsoap:body use="encoded"encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="http://unitec-denki.utj.co.jp/schema/2003/"/>
    </wsdl:input>
    <wsdl:output name="getPriceResponse">
      <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="http://unitec-denki.utj.co.jp/schema/2003/"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="EstimateService">
  <wsdl:port name="Estimate" binding="impl:EstimateSoapBinding">
    <wsdlsoap:address location="http://localhost:8080/axis/services/Estimate"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

この価格情報を返す Web サービスを利用したい場合は，Web アプリケーション開発時に WSDL を読み込ませることで，開発ツールが Web サービスの機能を利用するメソッドを自動生成できるようになるため，開発が容易になる．WSDL を読み込むことのできる開発ツールは多く存在する．具体例としては Enterprise Architect [14]や Microsoft Visual Studio [15]などがある．

2.3.2 WADL

WSDL と同様，WADL も XML で記述される．WADL を構成する主要な要素を表 2.4 に示す．

表 2.4 WADL を構成する主要な要素

要素名	要素の説明
application 要素	WADL のタグ構造の最上位要素. 以下で説明する要素は全てこの application 要素内で記述される
grammars 要素	HTTP レスポンスにより返却されるデータフォーマットのスキーマについて記述する．grammars 内で include 要素を用いることでスキーマ記述を読み込むことができる．読み込むスキーマ記述は RelaxNG [16]と W3C XML Schema [17]が利用可能となっている
resources 要素	Web API で利用可能な機能について記述する. 以下で説明する要素は全てこの resources 要素内で記述される．この要素の base 属性内でベースとなるリクエスト URI の指定を行う
resource 要素	Web API で利用可能な特定の 1 つの機能について記述する. つまり機能を複数持つ Web API は resource 要素を複数持つことになる．以下で説明する要素は全てこの resource 要素内で記述される
method 要素	HTTP プロトコルのどのメソッドが利用されるかを記述する．以下で説明する要素は全てこの method 要素内で記述される
request 要素	HTTP リクエストの仕様について記述する．主にパラメータの仕様を記述するために用いられ，以下で説明する param 要素を含む
param 要素	request 要素内で記述される．param 要素内で属性を指定することでパラメータの仕様を記述する．name 属性ではパラメータ名を, type 属性ではパラメータの型を指定することができる
response 要素	HTTP レスポンスの仕様について記述する．レスポンスのステータスごとにこの要素を定義することができる
representation 要素	この要素が記述されている親要素のインターネットメディアタイプ [18]やスキーマなどを記述することができる

図 2.7 に Yahoo News Search の WADL 記述例 [12]を示す．

図 2.7 Yahoo News Search の WADL

```
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
 xmlns:tns="urn:yahoo:yn"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:yn="urn:yahoo:yn"
 xmlns:ya="urn:yahoo:api"
 xmlns="http://wadl.dev.java.net/2009/02">

<grammars>
    <include href="NewsSearchResponse.xsd"/>
    <include href="Error.xsd"/>
</grammars>

<resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
        <method name="GET" id="search">
            <request>
                <param name="appid" type="xsd:string"
                 style="query" required="true"/>
                <param name="query" type="xsd:string"
                 style="query" required="true"/>
                <param name="type" style="query" default="all">
                    <option value="all"/>
                    <option value="any"/>
                    <option value="phrase"/>
                </param>
                <param name="results" style="query" type="xsd:int"
                 default="10"/>
                <param name="start" style="query" type="xsd:int"
                 default="1"/>
                <param name="sort" style="query" default="rank">
                    <option value="rank"/>
                    <option value="date"/>
                </param>
                <param name="language" style="query" type="xsd:string"/>
            </request>
            <response status="200">
                <representation mediaType="application/xml"
                 element="yn:ResultSet"/>
            </response>
            <response status="400">
                <representation mediaType="application/xml"
                     element="ya:Error"/>
            </response>
        </method>
    </resource>
</resources>

</application>
```

resources 要素中にアプリケーションが提供する情報を記述する．base 属性で Web API をリクエストする際のベース URL を記述する．resource 要素中の path 属性では URL に関して，各機能ごとに共通でない部分がある場合にベース URL に追加する部分 URL を指定することができる．例えば図 2.5 の例では resources 要素中でベース URL として「http://api.search.yahoo.com/NewsSearchService/V1/」が指定されており，resource 要素中では path 属性で「newsSearch」が指定されている．この例では，実際に HTTP リクエストを送る場合にはベース URL として「http://api.search.yahoo.com/NewsSearchService/V1/newsSearch」を利用することになる．例えばこの Web API に別の機能として論文検索が実装され，その機能のベース URL が「http://api.search.yahoo.com/NewsSearchService/V1/articleSearch」となった場合には，図 2.5 の WADL には resource 要素が追加され，path 属性として「articleSearch」が与えられることになる．

method 要素では http で用いるメソッドの名前を指定する．この例では GET メソッドが指定されている．id 属性にはその WADL 文書内の一意である任意の値を入力する．request 要素の各 param 要素が HTTP リクエスト内の各パラメータに対応している．request 要素の属性を以下に示す．詳細は param 要素の属性として記述する．

- name … パラメータの名前
- type … 文字列や数字などのパラメータの型
- default … パラメータのデフォルト値．クライアントから送られる HTTP リクエスト内で，パラメータに値が指定されなかった場合この値が使用される
- required … パラメータが必須か否か．クライアントから送られる HTTP リクエストで，必ず指定しなければならないパラメータならば，”required=true”を指定する
- style … スタイルを指定する．style 属性は他のタグの属性としても使われており，param タグ内では”style=query”を指定するように決められている．

応答の仕様は response タグ内の representation タグに記述する．status 属性でレスポンスの HTTP ステータスを指定し，タグ内で representation タグを用いてそのステータスが返された時の対応を記述する．representation タグの mediaType 属性ではレスポンスのインターネットメディアタイプを指定し，element 属性では grammar タグ内で宣言されているエレメント名を指定する．例では grammar タグ内で指定されている”NewsSearchResponse.xsd”内に ResultSet エレメントが宣言されている．このように grammar タグでインポートできるスキーマには W3C XML Schema などのスキーマ言語が利用される．例で用いられている NewsSearchResponse.xsd の内容を図 2.8 に示す．

図 2.8 NewsSearchResponse.xsd

```
<xs:schema targetNamespace="urn:yahoo:yn" elementFormDefault="qualified">
    <xs:element name="ResultSet">
        <xs:complexType>
            <xs:sequence>
                <xs:element  name="Result"  type="ResultType"  minOccurs="0"
            maxOccurs="50"/>
            </xs:sequence>
            <xs:attribute name="totalResultsAvailable" type="xs:integer"/>
            <xs:attribute name="totalResultsReturned" type="xs:integer"/>
            <xs:attribute name="firstResultPosition" type="xs:integer"/>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="ResultType">
        <xs:sequence>
            <xs:element name="Title" type="xs:string"/>
            <xs:element name="Summary" type="xs:string"/>
            <xs:element name="Url" type="xs:string"/>
            <xs:element name="ClickUrl" type="xs:string"/>
            <xs:element name="NewsSource" type="xs:string"/>
            <xs:element name="NewsSourceUrl" type="xs:string"/>
            <xs:element name="Language" type="xs:string"/>
            <xs:element name="PublishDate" type="xs:string"/>
            <xs:element       name="ModificationDate"       type="xs:string"
            minOccurs="0"/>
            <xs:element name="Thumbnail" type="ImageType" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ImageType">
        <xs:sequence>
            <xs:element name="Url" type="xs:string"/>
            <xs:element name="Height" type="xs:integer" minOccurs="0"/>
            <xs:element name="Width" type="xs:integer" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

SOAP 型の Web API は構造が複雑であるため Web API の開発者が WSDL も併せて提供する事例が多い．WADL は WSDL と違い提供されなくても REST 型の Web API の利用は難しくないため，併せて提供される事例が少ない．しかし利用は難しくないとしても，開発者はその利用にあたり Web API を利用するためのプログラムを作成する必要があり，また 2.1 で示したように Web API が急速に増加している状況で，開発者が各 Web API に対してプログラムを作成するのは非効率的である．以上の理由から本研究では REST 型の Web

API について考えることとする.

2.4 Web API を利用するためのプログラムライブラリ

現在, 開発者が Web API を利用してアプリケーションを開発する場合, 一般的には(1)Web API 提供者が公開しているオンラインドキュメントを読んで仕様を把握し，(2)仕様に則したプログラムを作成するというプロセスをたどる．オンラインドキュメントとは，各 Web API 提供者が Web 上で公開しているもので, Web API に関するリクエストやレスポンス等の仕様が書かれている．Web API のオンラインドキュメントの例を図 2.6 に示す．この例で送られるリクエスト URL は例えば以下のようになる.

http://search.yahooapis.com/WebSearchService/V1/webSearch?appid=test0001&query=Cameron&region=uk

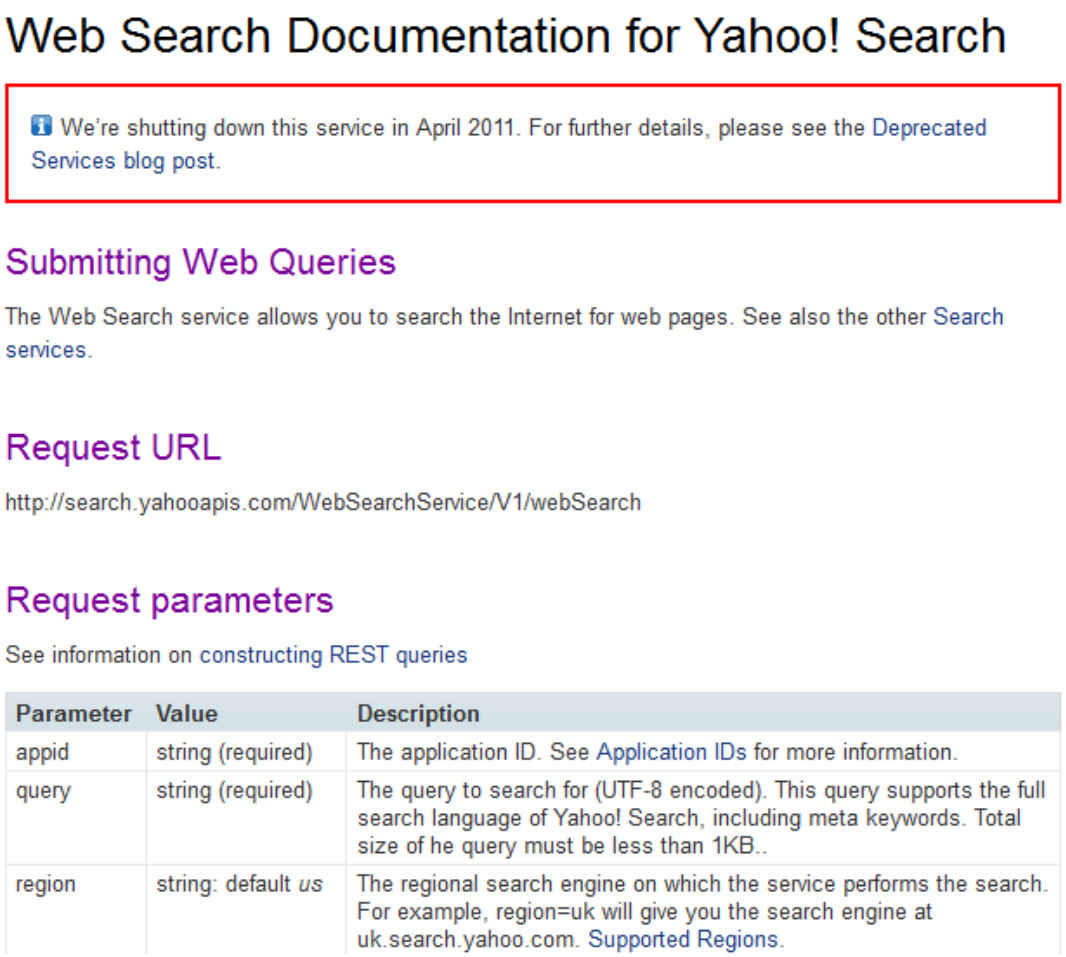
このリクエスト例では query パラメータとして「Cameron」を指定し，region パラメータで「uk」(イギリス)を指定している．Yahoo Web Search API は Yahoo の検索エンジンを利用して Web 検索をした結果を返す API であるので, この例では uk.search.yahoo.com のエンジンを利用してキーワード「Cameron」で検索した結果がレスポンスとして返される.

このように Web API 提供者によるオンラインドキュメントを元に，開発者は Web API の仕様に沿ったリクエストを生成するプログラム，またレスポンスを解釈できるプログラムなどを作成し, Web API を利用する. 一般にプログラムの中で Web API に関する処理は, プログラムライブラリとして汎用的にまとめられる事が多い．Web API に関する処理とは具体的には以下のことを指す.

- リクエストの生成
- リクエストの送信
- レスポンスの受信
- レスポンスとして受け取った構造データの解釈

実際に Twitter や Facebook など，比較的有名な Web API については作成されたプログラムライブラリが公開されている [19].

以上の他に多くの Web アプリケーションで導入されている Basic や OAuth などを利用したユーザ認証も Web API に関する処理といえるが, ほとんどの Web アプリケーションで共通する処理であり，認証に関するライブラリは Web API の種類に関係なく使いまわすことが可能であるため，ユーザ認証はそれに関する処理のみでライブラリとしてまとめられる場合が多い.



Web Search Documentation for Yahoo! Search

We're shutting down this service in April 2011. For further details, please see the Deprecated Services blog post.

Submitting Web Queries

The Web Search service allows you to search the Internet for web pages. See also the other Search services.

Request URL

http://search.yahooapis.com/WebSearchService/V1/webSearch

Request parameters

See information on constructing REST queries

Parameter	Value	Description
appid	string (required)	The application ID. See Application IDs for more information.
query	string (required)	The query to search for (UTF-8 encoded). This query supports the full search language of Yahoo! Search, including meta keywords. Total size of he query must be less than 1KB..
region	string: default *us*	The regional search engine on which the service performs the search. For example, region=uk will give you the search engine at uk.search.yahoo.com. Supported Regions.

図 2.6 Web API オンラインドキュメントの例(Yahoo Web Search [20]の一部)

2.5 プログラムライブラリ作成時の問題点

開発者が Web API のプログラムライブラリを作成する上で，2.2 で述べた現在の開発プロセスでは，以下のような問題がある．

（1） Web API ごとにプログラムライブラリを作成しなければならない

Web API は HTTP によるリクエスト・レスポンス通信など基本的な構造は共通しているが，指定可能なパラメータやレスポンスのスキーマなど細かな部分は共通しておらず，相互の互換性はない．例えば検索を行う Web API で Yahoo! Web Search と CiNii API [21]がある．検索クエリを指定するパラメータの名前として，Yahoo! Web Search は「query」，CiNii API は「q」を指定する．また検索の結果帰ってくるレスポンスの構造データのスキーマも違う．以上のことから，それぞれの Web API に対してプログラムライブラリを作成する必要がある．

（2） 各開発者がプログラムライブラリを作成しなければならない

Twitter や Facebook など，比較的有名な Web API に対しては，ライブラリが既に存在する場合が多いが，2.1 で述べたように Web API の数は急速に増加しており，ライブラリが存在しない Web API も多く存在する．このような Web API に対しては開発者自らが Web API のプログラムライブラリを作成しなければならない．また，プログラムライブラリを作成したとしてもそれを公開しない場合や，公開まで期間が空いた場合，他の開発者はまた自分でプログラムライブラリを作成する必要が出てくる．つまり同様の機能をもつコードを複数の開発者がコーディングしてしまうことになる．実際に twitter の Web API ではプログラムライブラリが複数作成，公開されており，公式サイトで紹介されているものだけでも 43 件のプログラムライブラリが存在し，Ruby という一つのプログラミング言語に限った場合でも 5 件のライブラリが存在する [19].

以上に加えて，既存のライブラリを利用することを考えても以下の様な問題がある．

（3） 利用目的にあったライブラリを探す必要がある

基本的にプログラムライブラリは各開発者が，自分が利用する Web API の機能に関してのみ実装したものであるため，Web API のすべての機能が利用可能となっているプログラムライブラリでない場合もある．そのため既存のライブラリを利用する場合，自分の利用したい機能を備えたプログラムライブラリを探す手間が発生する．

例えば前述した twitter の Web API ライブラリとして twitter4r [22]というものがある．2013 年 1 月現在バージョン 0.1.1 であるこのライブラリではフォロワーの管理やメッセージの送信などタイムラインの検索以外の機能が実装されているが，タイムラインの検索機能は実装されていない．

また開発者はライブラリ自体に継続的サポートがあるかどうかも確認する必要がある．継続的サポートとは，ライブラリ公開後もそのライブラリがデバッグや機能追加が行われることである．継続的サポートがないライブラリの場合，安全性は低く，利用の際のリスクは高まる．

（4） Web API の仕様変更に対応する必要がある

Web API の仕様が変更された場合，そのプログラムライブラリもその仕様変更に対応し更新する必要がある．更新ではライブラリの開発者がコーディングをし直す必要がある．もし開発者が更新を行わなかった場合，ライブラリの一部もしくは全てが利用不可能になってしまう可能性がある．例を挙げると，実際に twitter API では過去に大規模な仕様変更が行われており [23] [24] [25]，以前使用できていた機能が利用不可になるケースが起きている．この仕様変更によって，多くのライブラリの一部機能が利用できなくなる問題が起きた．このように Web API の仕様変更が行われた場合，ライブラリの修正によって開発者に負担がかかる．この問題は，(3)で述べた継続的サポートがあるライブラリでも発生する．

ライブラリが仕様変更に対応するまで期間が空いてしまう場合，ライブラリがその期間において一部，もしくは全部利用できなくなる可能性がある．Web API の仕様変更によってライブラリの仕様も抜本的に変わってしまう場合にも開発者に負担がかかる．

これらの問題を解決するためには，いくつかの方法が存在する．1つ目は，1つのWeb API に対するライブラリを共通化し，公開する方法である．複数のライブラリが存在することが問題となっているため，1つのWeb APIに対して対応するライブラリを1つだけ作成し，それを公開することで，今後新たなライブラリを作らないようにする．この方法では，(1)から(4)で挙げた問題点のうち(1)から(3)の改善につながる．しかし，この方法を実現するためには重複してライブラリを作成しないよう全ての開発者に呼びかけ，管理する必要がある上，すべてのプラットフォーム，すべてのプログラミング言語に対応した統一的なライブラリを作ることも困難であるため，現実的であるとはいえない．

2つ目はライブラリの作成コストを下げる方法である．開発者が自分でコーディングを行い，ライブラリを作成することが時間がかかり，手間である点が問題であるので，ライブラリの作成が短時間でできるようになれば，(1)から(4)で挙げた問題全ての解決につながる．機械的に自動でライブラリを生成できるようになることが最も望ましいが，そうでなくてもライブラリ作成においてコーディングのような時間のかかる作業をなくすことでも，問題の解決につながる．

ライブラリ作成のコストが下がれば，開発者ではなくWeb API提供者がライブラリを提供する事例も増加すると考えられる．Web API提供者によって提供されるライブラリは継続サポートされる可能性が高いため，結果として公開されるライブラリの種類は減少し，ライブラリの質も高まると考えられる．

本研究ではWeb APIのプログラムライブラリ作成に必要な情報を自動で抽出することで，人手によるコーディングを行わず，ライブラリが作成できるよう開発者の支援を行う手法を提案する．

第 3 章 Web API のプログラムライブラリ作成支援

3.1 概要

本章では実際にどのようにしてWeb APIのプログラムライブラリ作成に必要な情報を抽出するのか，その手法について述べる．

本研究ではWeb APIのオンラインドキュメントを利用する．オンラインドキュメントからプログラムライブラリの生成に必要な情報を機械的に抽出する．その抽出した情報からプログラムライブラリを生成する(図3.1)．これまで開発者が自分で行なっていたライブラリの作成を機械的に行うことにより，開発者のライブラリ作成にかかる手間を省く．Web APIのプログラムライブラリの生成を機械的に行うことにより，開発者は2.5で述べたライブラリを作成する際の問題から開放される．またWeb APIの利用者がライブラリを作成するのではなく，提供者がライブラリを作成することも簡単に行えるようになるので，Web APIとライブラリが同時に提供されることも増加する可能性がある．開発者がライブラリを作成することがなくなるため，1つのWeb APIに対して各プログラミング言語で1つのライブラリのみが存在する状況となる．これにより2.5で述べた既存のライブラリを利用する際の問題も解消される．

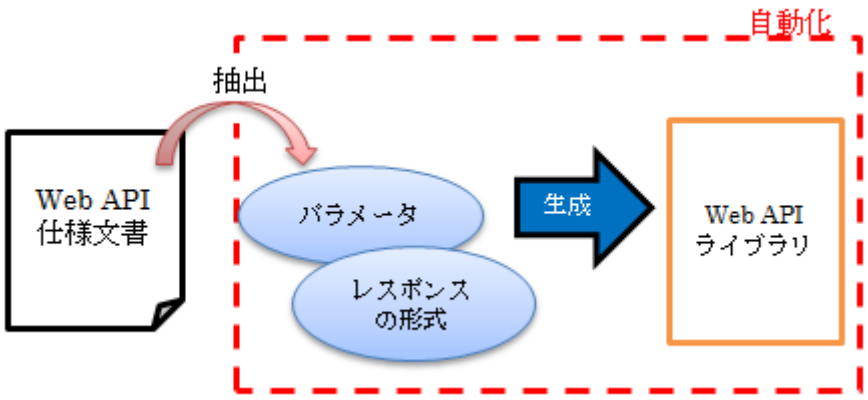


図3.1. Web APIプログラムライブラリ生成手法概要図

3.2 Web API オンラインドキュメントからの情報抽出

プログラムライブラリの生成に必要な情報は各Web APIのオンラインドキュメントから機械的に抽出する．抽出にはWeb APIのオンラインドキュメントに見られる規則性を利用する．オンラインドキュメントではWeb APIの情報が提供元によって様々な記述形式で記されているがその中には幾つかの共通点がある．例としてtwitter APIのオンラインドキュメントとYahoo! Answers APIのオンラインドキュメントをそれぞれ図3.2，図3.3に示す．

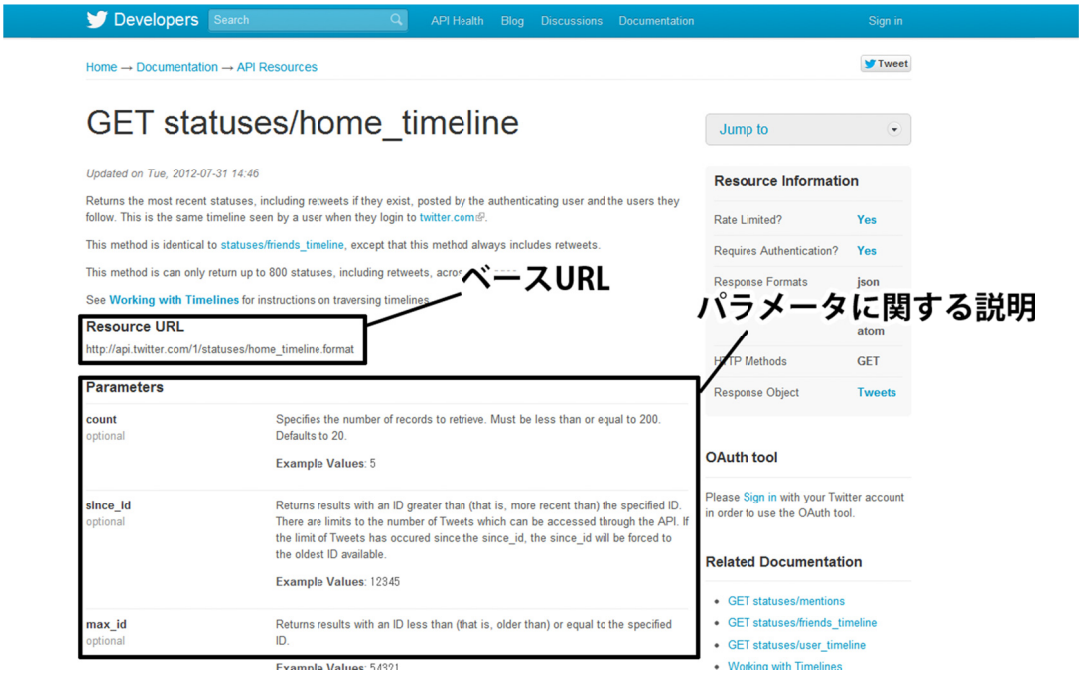


図3.2. Twitter API [26]のオンラインドキュメント(一部)

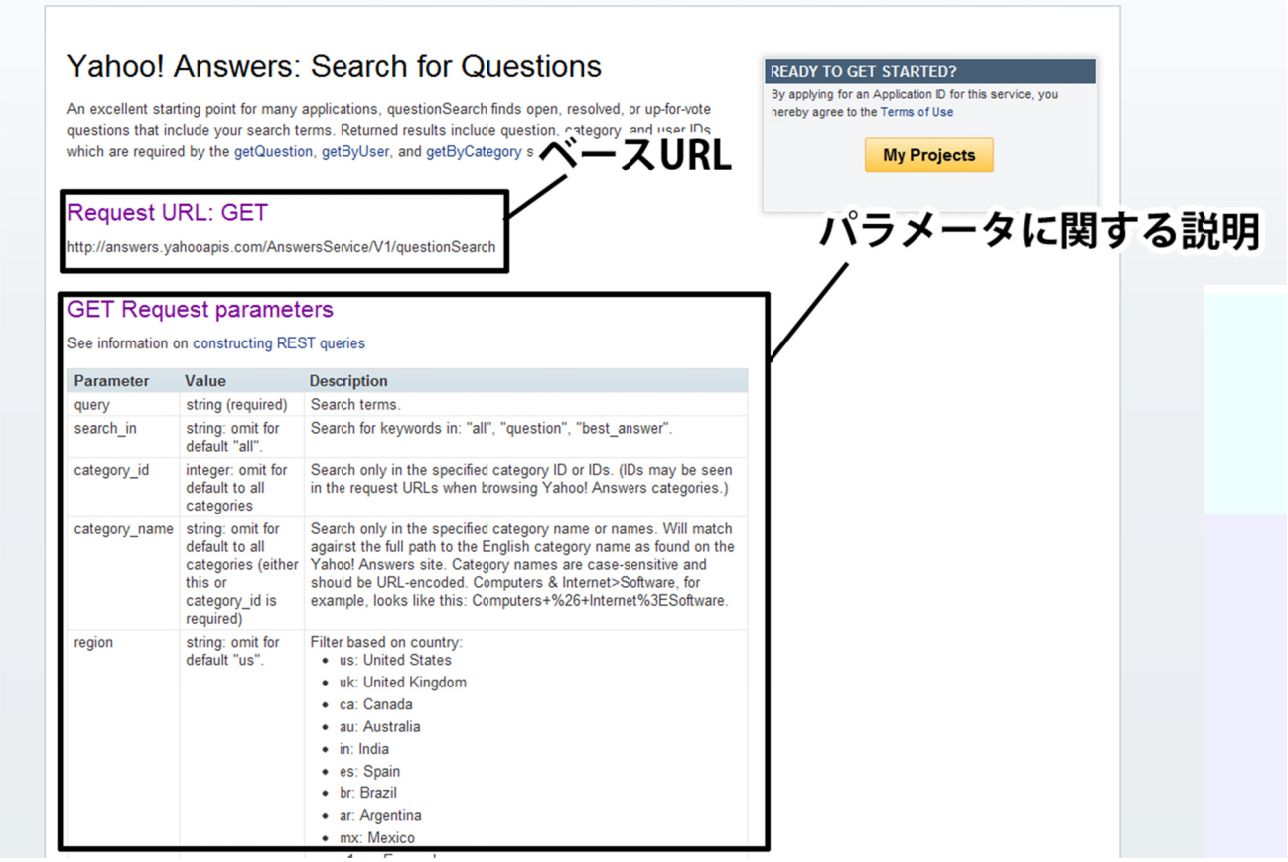


図3.3. Yahoo! Answers API [27]のオンラインドキュメント(一部)

REST 型の Web API では，HTTP のリクエストを送りそのリクエストに応じたレスポンスが返される．図 3.2，図 3.3 ではドキュメント中に Resource URL もしくは Request URL という表記でベース URL が記述されている．その下にベース URL と一緒に送るリクエストパラメータについて，そのパラメータ名とパラメータの型や説明が記述されている．このように多くのオンラインドキュメントではベース URL の記述からパラメータに関する説明など共通の項目が説明されており，各情報は箇条書きや表によって整理されていたり，大文字によるタイトルの後に記述されていたりと記述の方法に特定のパターンがある．本手法ではオンラインドキュメントの HTML のタグ構造とキーワードに見られるパターンを利用し，情報の抽出を行う．

実際の抽出では，オンラインドキュメントの規則性を利用した複数の「抽出ルール」をプログラム上に実装し，そのプログラムにオンラインドキュメントを入力として与えた(図 3.4)．抽出ルールはヒューリスティクスを用いて作成する．抽出ルールの例を以下に擬似コードとして示す．

例)

```
if（HTML ファイルの 1 行 == 「http://」ではじまり，「?」で終わる）
    ベース URL とみなす
end
```

この例のように，抽出ルールはオンラインドキュメント内の HTML の文字列に対してパターンマッチを行い，抽出すべき情報を取得するものである．

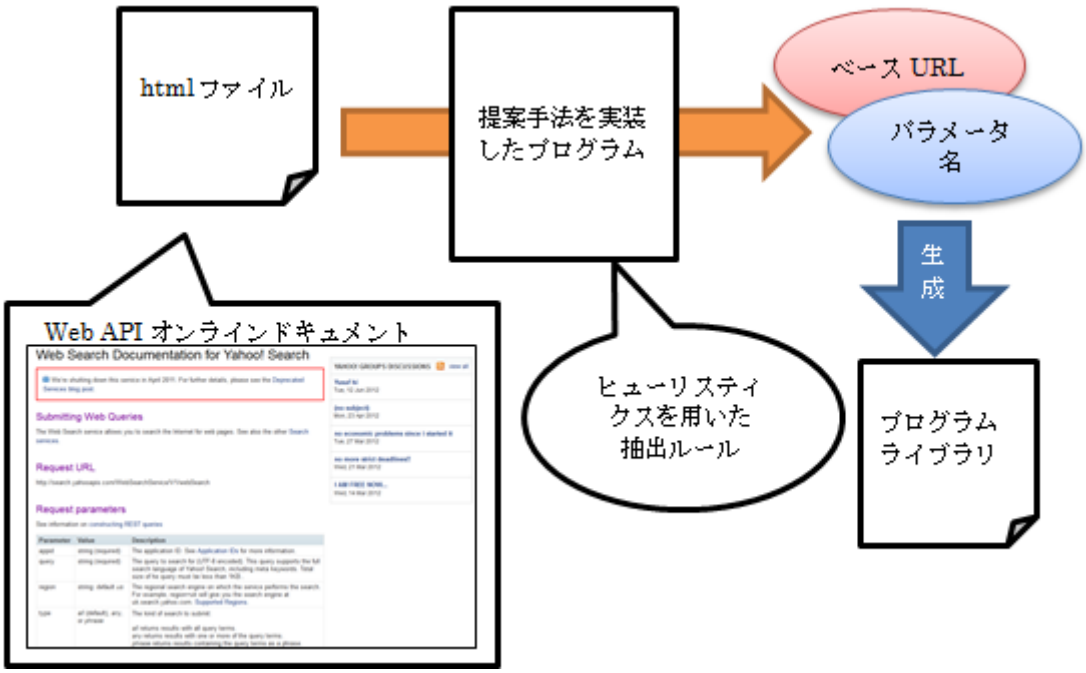


図 3.4．オンラインドキュメントからの情報抽出手順

3.3 抽出した情報に基づくプログラムライブラリの作成

本研究では抽出した情報の記述にはWADLを用いる．WADLを利用することで抽出した情報を機械的に解釈しやすい形で記述できる．現在WADLを利用することで既存の開発環境においてプログラムライブラリを生成することが可能となっている．例えば統合開発環境のひとつであるNetBeans [28]ではWADLを読み込み，プログラムライブラリを生成することができる．

WADLの生成にあたり，本研究では生成に必要とする情報を以下の7要素とした．以下の項目は，(1)WADL内に必須の要素である，(2)サーバに対してリクエストを送る際に不正なリクエストかどうかの判定に必要である，という2点のうちいずれかにあてはまるものである．本手法ではオンラインドキュメントからこれらの情報を抽出し，WADLを作成する．

- ベースURL　…　HTTPリクエストを送る際に指定するURL
- パラメータ名　…　リクエストで指定できるパラメータの名前
- パラメータの型　…　文字列や数字など各パラメータで指定できる型
- パラメータのデフォルト値　…　リクエストで指定しなかった場合の省略時値
- 必須パラメータか否か　…　必ず指定しなければならないパラメータであるか否か
- リクエストの形式　…　リクエストの送信方法に関する規定(GETメソッドかPOSTメソッドか，など)
- レスポンスの形式　…　レスポンス自体の形式に関する規定(XML か JSON か，など)

第 4 章 予備実験

4.1 実験の目的

3 章で述べた提案手法によってどの程度正確に情報が抽出できるかを確認するため予備実験を行った．本予備実験では(1)提案手法によって情報抽出が可能かどうか確かめる，(2)提案手法の問題点を明確にする，という 2 点を目的とした．

4.2 実験方法

4.2.1 概要

実験では実際のオンラインドキュメントに対し，本手法を試験的に実装したプログラムを適用し, 情報が抽出できるかどうか試行した. 抽出する情報は, 3 章で示した 7 項目のうち，ベース URL とパラメータ名のみとした．オンラインドキュメントからはあらかじめ人手でベース URL とパラメータ名を抽出しておき，それを正解とする．またオンラインドキュメントは HTML ファイルとして保存しておき，そのドキュメントを実験用プログラムに入力として与えた．実験用プログラムは入力されたファイルから抽出したベース URL とパラメータ名を出力する．この出力結果とあらかじめ人手で抽出しておいたベース URL，パラメータ名を比較した(図 4.1).

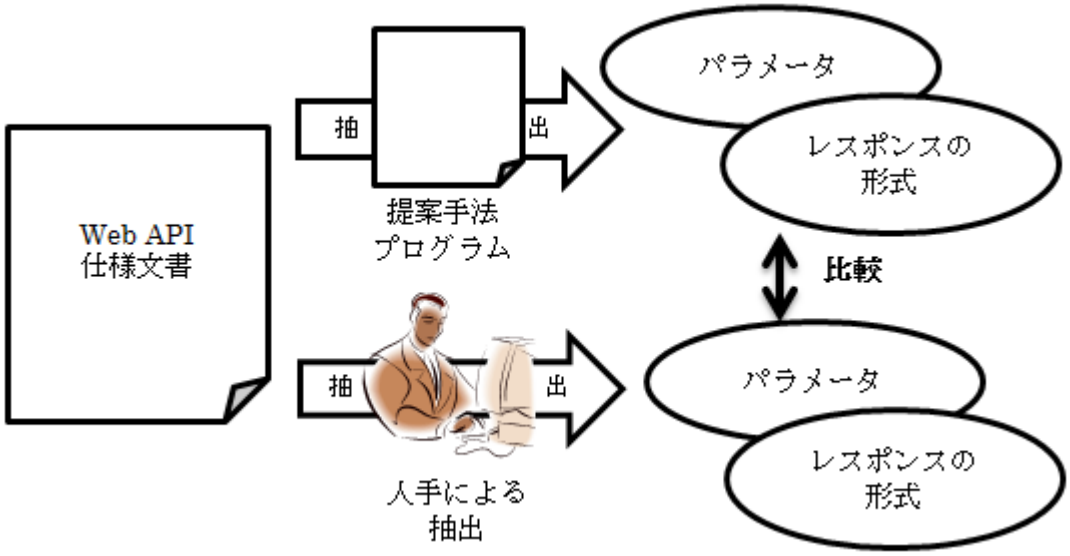


図 4.1 予備実験概要

4.2.2 実験環境

実験時の環境を以下に示す.

実験用プログラムのプログラミング言語：Ruby 1.9.2 (標準添付ライブラリのみ使用)
OS：Windows 7 Ultimate SP1

4.2.3 対象ドキュメント

予備実験で用いたオンラインドキュメントを以下に示す．これらを対象ドキュメントとした理由は，日本語で記述されたオンラインドキュメントである点と，被マッシュアップされているWeb APIである点の2点である．日本語で記述されたオンラインドキュメントであれば，本文は日本語であるのに対し，ベースURLやパラメータ名はアルファベットで記されるので，英語のオンラインドキュメントに比べ抽出が容易なはずであり，これらのドキュメントから十分に情報が抽出できなければ，手法に問題があることがわかると考えた．また，被マッシュアップされているWeb APIであれば，Web APIとして活用されていることが保証されていることになると考えた.

(1) はてなキーワードAPI(旧文書)
http://d.hatena.ne.jp/keyword/%A4%CF%A4%C6%A4%CA%A5%AD%A1%BC%A5%EF%A1%BC%A5%C9API(参照 2012-08-20)
(2) はてなキーワードAPI(新文書)
http://developer.hatena.ne.jp/ja/documents/keyword/apis/rest(参照 2012-08-20)
(3) CiNii API(論文検索)
http://ci.nii.ac.jp/info/ja/api/a_opensearch.html(参照 2012-08-20)
(4) 郵便番号検索API
http://groovetechnology.jp/webservice/zipsearch/index.html(参照 2012-08-20)
(5) 顔検出 API
http://kaolabo.com/webapi/spec(参照 2012-08-20)

予備実験で使用したオンラインドキュメントは全て日本語で記述されているものである．はてなキーワードAPIに関しては，旧文書と新文書の2件が存在したので，その両方を利用した.

4.2.3 実験プログラム内の抽出ルール一覧

今回適用したルールの擬似コードを以下に示す．抽出ルールは，予備実験に用いたプログラムではRubyによるハードコーディングにより実装されている．

```
if（キーワードリストにあるワードが URL より前に出てくる）
    URL のスコアを加算する
end

if（HTML ファイルの 1 行 == 「http://」ではじまり，「?」で終わる）
    if（「http://」から「?」の間に URL として不適切な文字が含まれていない）
        ベース URL とみなす
        高いスコアを基礎スコアとして与える
    end
else if（HTML ファイルの 1 行 == 「http://」ではじまる）
    if（「http://」から始まる文字列に URL として不適切な文字が含まれていない）
        URL とみなす
        低いスコアを基礎スコアとして与える
    end
end

if（文字列 ==（ある表の前の数行もしくは表の要素名にキーワードリストのキーワードがある && 表の第 1 列目の要素である && 基本英数字で構成されており，文字数も少ない））
        パラメータ名とみなす
    end
end

if（定義リストがある）
    定義リスト内のワードをパラメータ名とみなす
end
```

ベース URL に関する抽出ルール中にあるスコアとは，あるオンラインドキュメントから抽出される複数のベース URL 候補を順位付けするために設定した数値である．

キーワードリストとはオンラインドキュメント中に出てくる単語のうち情報抽出で役に立つと思われるキーワードをあらかじめ手作業でまとめておいたリストのことである．予備実験で使用したキーワードリスト内に登録されているキーワードを以下に示す．

- パラメータ
- parameter
- Parameter

定義リスト(Definition List)とは HTML の dl 要素で囲まれたものを指す．dl 要素内の dt 要素により単語が定義され，dd 要素により単語の説明が定義される．

4.3 実験結果

今回の予備実験では，4.2.2で示したすべてのオンラインドキュメントから正解と等しい情報を抽出することができた．実験におけるベースURLの抽出に関する結果を表4.1に示す．表4.1では抽出候補の上位3件をスコアの昇順で示している．表中の背景を灰色で示しているものが正解と一致している抽出候補である．

表4.1 予備実験結果(ベースURLの抽出)

抽出対象	人手による抽出(正解)	提案手法による抽出
(1)	http://search.hatena.ne.jp/keyword	http://search.hatena.ne.jp/keyword
		http://red3.hatena.ne.jp/adframe
		http://red.st-hatena.com/adframe
(2)	http://search.hatena.ne.jp/keyword	http://search.hatena.ne.jp/keyword
		http://d.hatena.ne.jp/keyword
		http://hatenadeveloper.g.hatena.ne.jp/keyword/ja/keyword/apis/rest
(3)	http://ci.nii.ac.jp/opensearch/search	http://ci.nii.ac.jp/opensearch/search
		https://register-ci.nii.ac.jp/auth/action/login
		https://register-ci.nii.ac.jp/userregist/userTypeSel.do
(4)	http://api.postalcode.jp/v1/zipsearch	http://api.postalcode.jp/v1/zipsearch
		http://groovetechnology.jp/sitemap.html
		http://uchiai.com/
(5)	https://kaolabo.com/api/detect	https://kaolabo.com/xmlrpc.php
		https://kaolabo.com/api/detect
		http://kaolabo.com/feed

4.4 考察

今回の予備実験によりベースURLとパラメータ名が本手法によって取得可能であることがわかった．比較的簡単な抽出ルールしか用いていないにもかかわらず，高精度で情報抽出が行うことができた．今回は情報を2項目しか抽出していないが，パラメータ名に関しては表形式で記述されているもの(HTMLのtableタグを用いているもの)はその表の中の要素を抽出することに成功しているので，パラメータに関する他の情報もパラメータ名と同じように抽出できる可能性が高い．今後の課題として，ベースURLに関して正解のURL以外のURLも誤って優先度が高いとみなされて抽出されたものもあったため，ベースURL候補の優先順位の付け方を検討する必要がある．また今回は対象としたオンラインドキュメントは件数が少なかったが，より対象件数を増やした場合には表記のゆれなど今回出なかった問題が発生する可能性がある．

第 5 章 Web API オンラインドキュメントからの情報抽出の実現

5.1 情報抽出の方法

本節では予備実験の結果を元に，実際のオンラインドキュメントから情報を抽出するために開発したプロトタイプシステムについて詳述する．

5.1.1 プロトタイプシステム概要

プロトタイプシステムの概要図を図 5.1 に示す．オンラインドキュメントは Web 上にあるものを HTML としてローカルディスク上に保存する．プロトタイプシステムはこの HTML を入力として受け取り，あらかじめ実装しておいた抽出ルールに基づいて情報抽出を行う．抽出された情報にはその情報がプログラムライブラリの生成において抽出されるべき正しい情報であるかどうかのスコアを付与した．このスコアは予備実験ではベース URL にのみ適用されていたが，プロトタイプシステムでは英語のオンラインドキュメントを対象とすることを考え，予備実験時より不必要な情報を多く抽出してしまう状況が想定できるため，ベース URL 以外の情報もスコアにより順位付けを行い，一定スコア以下のものをフィルタリングする機能を備えることとした．

抽出された情報はユーザに一覧として提示される．また，オンラインドキュメントのどの位置から抽出したものなのか，実際に抽出した情報(文字列)をハイライトしたオンラインドキュメントを出力し，提示する機能も持つ．ユーザはそこから不要な情報の削除や，取りこぼされてしまった必要な情報のを自分で入力することが可能である．最終的に抽出すべき情報が確定すると，プロトタイプシステムは抽出した情報を WADL 形式で出力する．

プロトタイプシステムの開発環境を以下に示す．

OS：Windows 7 Ultimate SP1
プログラミング言語：Ruby 1.9.2 (標準添付ライブラリのみ使用)

プロトタイプシステムはコンソールアプリケーションとして作成した．プロトタイプシステムの主要な機能を以下に示す．

- HTML からの 3.3 で示した要素の抽出
- 抽出した情報を元にした WADL の生成
- 抽出した情報の部分をハイライトしたオンラインドキュメント(HTML)の生成
- 抽出した情報のユーザによる追加・削除・編集

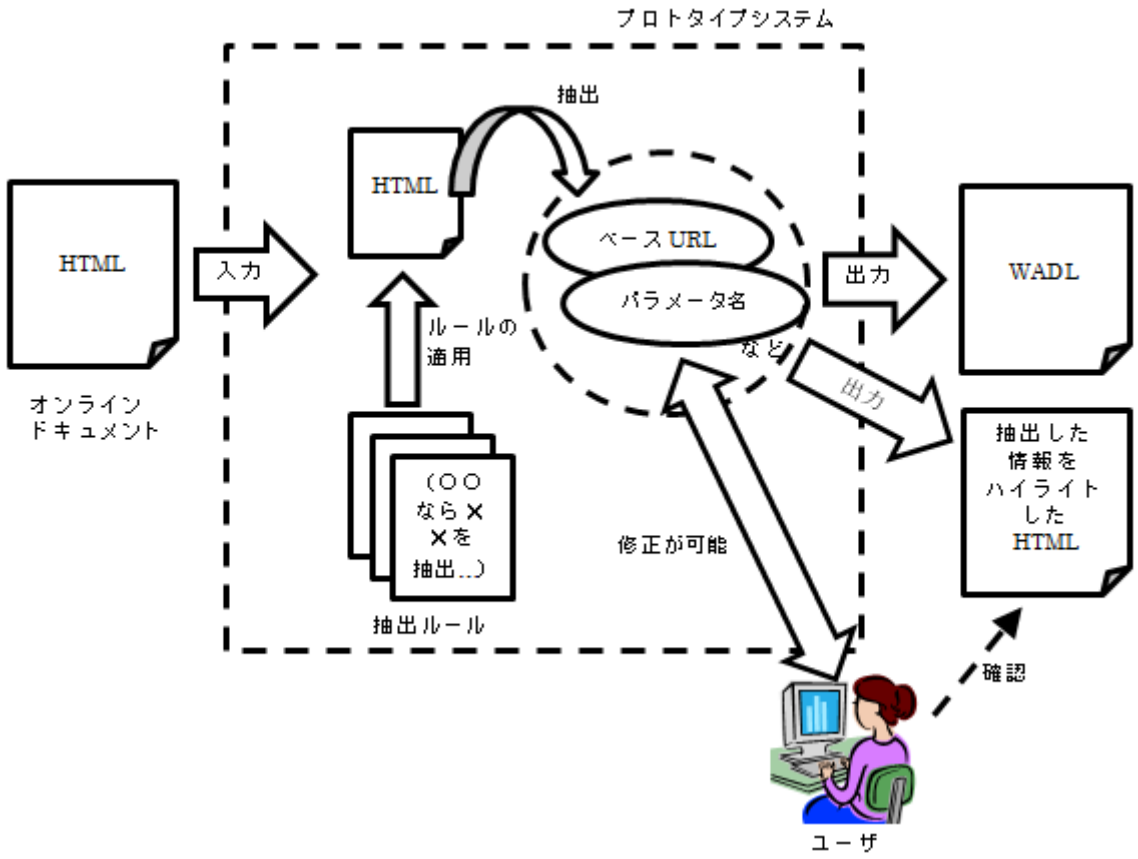


図 5.1 プロトタイプシステム概要

5.1.2 抽出ルールに基づいた情報の抽出

プロトタイプシステムでは入力された html から，各抽出ルールに一致する情報を判定し抽出を行う．各抽出ルールは Ruby プログラムによって記述されている．

抽出された情報はシステム内でオブジェクトとして，以下の情報を持って管理される．

- 情報の文字列
  抽出した情報そのもの．ベース URL ならば，「http://...」，といったものが入る
- 情報の種類
  3.3 で示した抽出する 7 要素のうち，どれにあたるか
- 抽出された元の html での行番号
  オンラインドキュメントの html の何行目から抽出されたものであるか
- スコア
  抽出された情報がプログラムライブラリの生成において抽出されるべき正しい情報であるかどうかの度合い

スコアは，まず抽出された情報に基礎点を与え，スコアの点数を上下させるための抽出ルールによって基礎点に対し加減算を行うことで決定される．

5.1.3 ユーザによる情報の修正

抽出された情報はユーザに一覧として提示される．また，オンラインドキュメントのどの位置から抽出したものなのか，実際に抽出した情報の文字列もしくはその行内の文章をハイライトしたオンラインドキュメントを出力し，提示することもできる．実際にプロトタイプシステムが情報を一覧として提示している画面例を図 5.2 に，ハイライトして出力されたオンラインドキュメントの例を図 5.3 に示す．図 5.2 と図 5.3 では共にプロトタイプシステムに twitter API(図 3.2)のオンラインドキュメントを入力として与えた場合の例である．

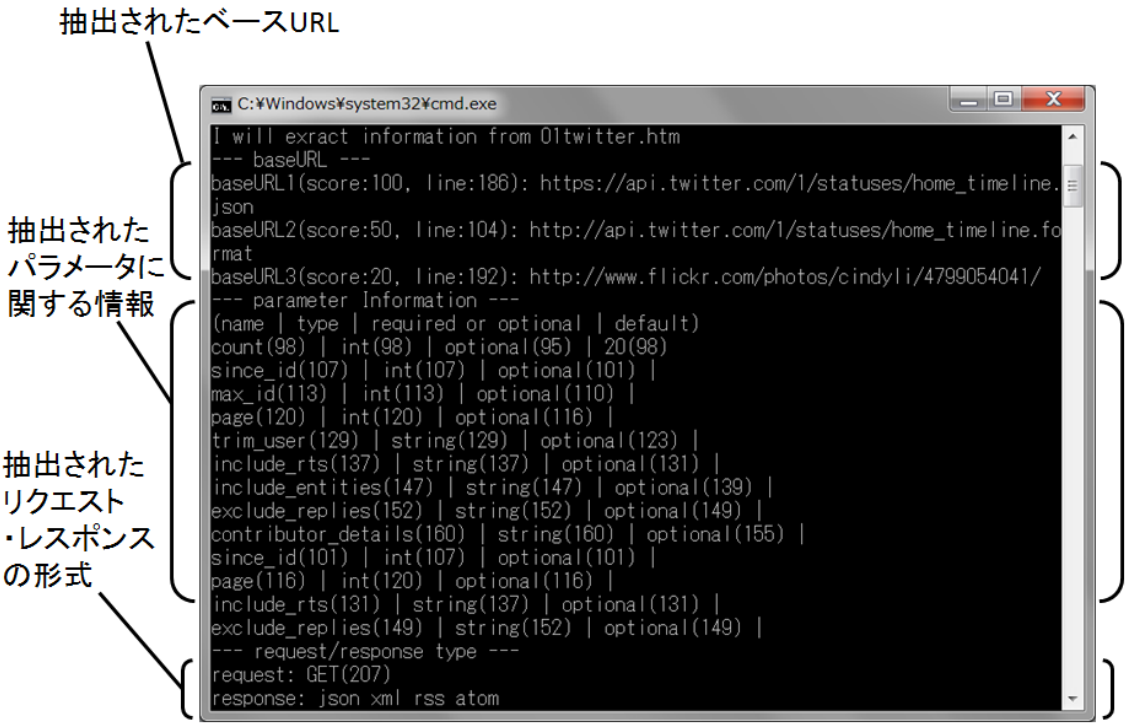
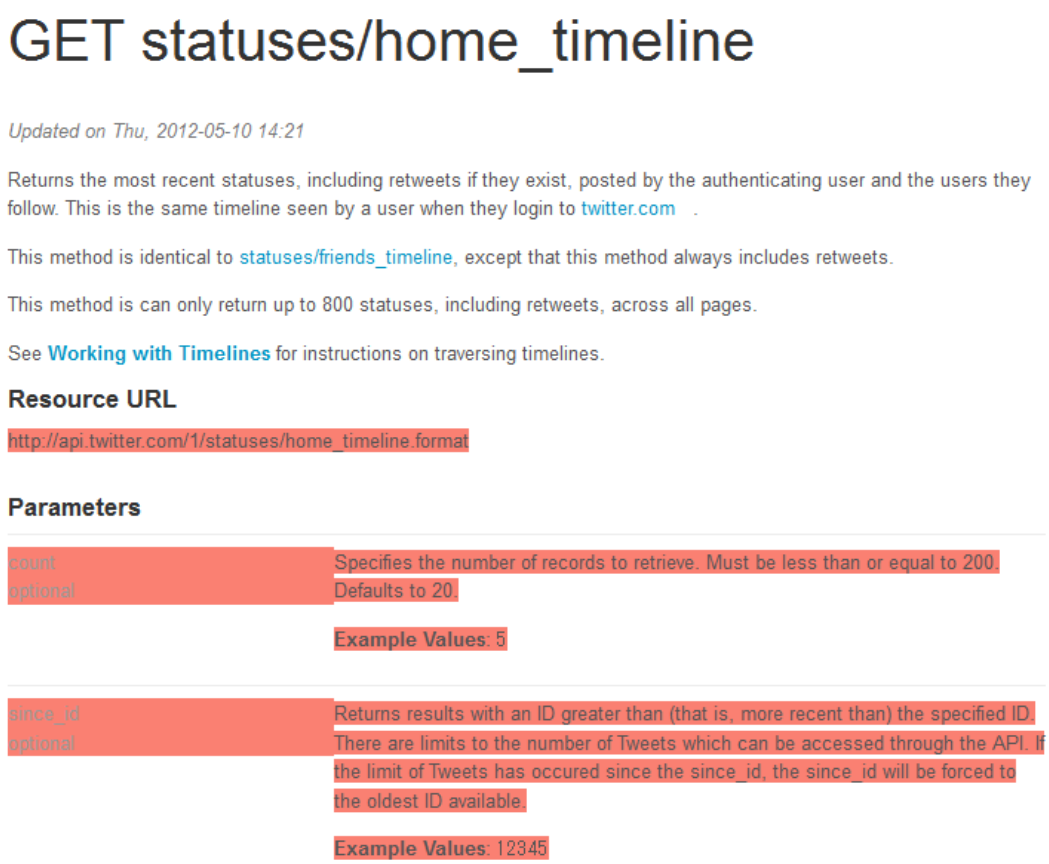


図 5.2 プロトタイプシステム情報抽出画面例



GET statuses/home_timeline

Updated on Thu, 2012-05-10 14:21

Returns the most recent statuses, including retweets if they exist, posted by the authenticating user and the users they follow. This is the same timeline seen by a user when they login to twitter.com .

This method is identical to statuses/friends_timeline, except that this method always includes retweets.

This method is can only return up to 800 statuses, including retweets, across all pages.

See **Working with Timelines** for instructions on traversing timelines.

Resource URL

http://api.twitter.com/1/statuses/home_timeline.format

Parameters

count optional	Specifies the number of records to retrieve. Must be less than or equal to 200. Defaults to 20. **Example Values**: 5
since_id optional	Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available. **Example Values**: 12345

図 5.3 情報がハイライトされたオンラインドキュメント例

ユーザは提示された情報を見て，不要な情報の削除や，必要な情報の追加などその情報の編集をすることができる．修正はプロトタイプシステムに対して特定のコマンドを入力することで可能となる．コマンドを以下に示す．

- select
  - select [information type]
  - 現在の各情報の一覧を表示
- insert
  - insert into [value] values [information type]
  - 新たな要素を追加
- update
  - update parameter set [paramName]=[value], [paramType]=[value], [paramRequired]=[value], [paramDefault]=[value]
  - パラメータに関する情報を更新

- delete
  - delete from [information type] where [value]
  - 要素の削除

文中の[information type]には以下の要素のうちどれかを入力する．()内はコマンド入力の際のショートカットである．

- ベース URL：baseURL(b)
- パラメータに関する情報：parameter(p)
- リクエストの形式：requestType(req)
- レスポンスの形式：responseType(res)

文中の[value]は実際にユーザが入力したい任意の値を指す．

update コマンド中の「param」で始まる[]内の要素は以下の要素を指す．

- paramName(pn)：パラメータ名
- paramType(pt)：パラメータの型
- paramRequired(pr)：パラメータが必須か否か
- paramDefault(pd)：パラメータのデフォルト値

5.1.4 WADL の出力

プロトタイプシステムで抽出した情報は WADL 形式で出力される．出力された WADL は NetBeans を用いてプログラムライブラリに変換することが可能である．

5.2 オンラインドキュメントの分類

Web API のオンラインドキュメントは Web API と同じ数だけ存在する．記される情報は共通しているが，表による情報の整理や具体例の提示などその記述方法にはそれぞれ差がある．抽出ルールを検討する上で，これらの各オンラインドキュメントの性質から分類をすることが望ましいと考えた．本研究ではオンラインドキュメントをタイプ 1 とタイプ 2 に分けて考える．以下にその 2 種類の特徴を示す．

5.2.1 タイプ 1 のオンラインドキュメント

以下の特徴を持つオンラインドキュメントをタイプ 1 とした．

(1) Web API の 1 つの機能に関する情報が 1 つの Web ページ(html ページ)に示されている．その 1 つの機能の情報を得るのにリンクから他のページへを閲覧する必要がない

(2) 情報が提示される順番が決まっている

タイプ 1 では，情報が提示される順番が決まっている．その順番は以下の通りである．

- ベース URL
- パラメータに関する情報
- レスポンスの形式
- 具体例

(3) パラメータに関する情報は表，箇条書き，定義リストのいずれかの形式で整形されて示される

基本的にタイプ 1 は(1)～(3)の要素すべてを満たすが，いずれかの要素が欠けている場合もある．タイプ 1 の例としては，Yahoo! Answers API や twitter API が挙げられる．いずれも(1)～(3)の要素を満たしている．

5.2.2 タイプ 2 のオンラインドキュメント

以下の特徴を持つオンラインドキュメントをタイプ 2 とした．

(1) Web API の 1 つの機能に関する情報がリンクによって分割され，複数の Web ページを閲覧する必要がある．または，1 つの機能に関する情報だけでなく，他の機能の情報も併せて示されている

これは，例えば SNS に関する Web API があり，その Web API は SNS 内の投稿のキーワード検索の機能と自分が SNS に投稿する機能の 2 つを備えているとする．この Web API がタイプ 2 のオンラインドキュメントである場合は，ベース URL，パラメータに関する情報，レスポンスの形式などが別々のページに示されていたり，またはキーワード検索に利用可能なパラメータと SNS に投稿する際に利用可能なパラメータが同じ Web ページ内で示されていたりする，ということである．

(2) 各情報は表，箇条書き，定義リストの形式は用いられず，整形されていない文章内で説明される

タイプ 2 では，3.3 で示した本研究で抽出される情報が表や箇条書きを用いず，文章の一部として出てくる事が多い．

タイプ 2 の例としては Bing API や Shopping.com API などがある．

5.3 実装した抽出ルール

5.3.1 抽出ルールの実装方法

抽出ルールは予備実験時と同様にハードコーディングにより実装されているもののほか，正規表現を拡張した独自スクリプトを利用して実装されているものもある．

独自スクリプトでは正規表現を用いた文字列パターンの一致判定に加え，HTML コードの複数行の文字列に対して，どの行でどのパターンが一致するか，複数行間のパターン一致の関係を記述することができる．独自スクリプトでは基本的に Ruby で利用可能な正規表現のパターン [29]が利用可能で，それに加えて「 /」(半角スペースとスラッシュ)を用いて文字列パターンを区切ることで，パターンの一致する行の位置関係が記述できる．「¥/」は例外として区切りとはみなさない．「 /」で区切られた区画を便宜上「ブロック」と呼ぶこととする．独自スクリプトの基本的な構文は図 5.4 のようになる．

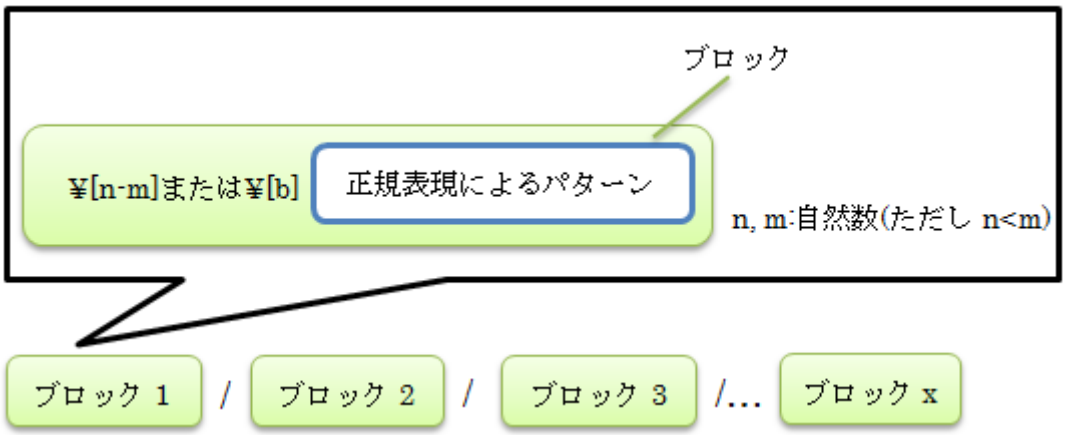


図 5.4 独自スクリプトの構文

例えば HTML コードのある 1 行 で「http://～?」というパターンと一致し，かつその直前の 1 行～3 行の間で「Base URL」もしくは「Resource URL」と一致し，かつ「http://～?」というパターンと一致した行より下にある行で「Method」というパターンと一致し，更にその「Method」というパターンと一致した行より下 1～5 行の中で「GET」というパターンと一致する部分を探したい場合，独自スクリプトでは以下のように記述する．

¥[1-3](Base|Resource) URL /¥[b]http:¥/¥/.+? /¥[1-2]Method /¥[1-5]GET

上の例ではブロックが 4 つできていることになる．各ブロックは先頭に必ず「¥[n-m]」(n と m は自然数で n<m)，もしくは「¥[b]」という記述をする．これは HTML コード内におけるブロック内の文字列パターンの適用範囲を示す．「¥[n-m]」で表す適用範囲は「¥[b]」

の記述があるブロックの左側に位置するか右側に位置するかによって異なる.

基準点より左に記述されているブロックであれば，自分の右隣のブロックのパターンが一致した HTML コード中の行より上の n 行から m 行の範囲，基準点となったブロックより右に記述されているブロックであれば，自分の左隣のブロックのパターンが一致した HTML コード中の行より下の n 行から m 行の範囲を表す.

キーワードリストは予備実験時とは異なり，抽出する要素に応じて複数のキーワードリストを作成した．作成したキーワードリストは,

- ベース URL に関するキーワードリスト(リスト b)
- パラメータ名に関するキーワードリスト(リスト pn)
- パラメータの型に関するキーワードリスト(リスト pt)
- パラメータのデフォルト値に関するキーワードリスト(リスト pd)
- パラメータが必須か否かに関するキーワードリスト(リスト pr)
- リクエストの形式が GET メソッドの場合のキーワードリスト(リスト rg)
- リクエストの形式が POST メソッドの場合のに関するキーワードリスト(リスト rps)
- リクエストの形式が PUT メソッドの場合のに関するキーワードリスト(リスト rpu)
- レスポンスの形式に関するキーワードリスト(リスト res)

の 9 種類である. 以下に示す抽出ルールでは, これらのキーワードリストを使用している. なお以下に示す抽出ルール内では，これらのキーワードリストを以上に示した[リスト b]などの省略形で記述する.

5.3.2 抽出ルール一覧

以下にプロトタイプシステムで実装されている抽出ルールを示す.

(1) ベース URL に関するルール

if (HTML ファイルの 1 行の一部 == “http://”ではじまり, ”?”で終わる)
    if (“http://”から”?”の間に URL で不適切な文字が含まれていない)
        ベース URL とみなす
        高い基礎スコアを与える
    end
else if (HTML ファイルの 1 行の一部 == “http://”ではじまる)
    if (“http://”から始まる文字列に URL で不適切な文字が含まれていない)
        ベース URL とみなす
        低い基礎スコアを与える
    end
end

if (リスト b にある文字列が抽出した URL より前に出てくる)
    その URL のスコアを加算する
end

if (抽出した URL の文字数 < 10)
    その URL のスコアを減算する
end

if (“/”と英数字, ”_”, ”-”, ”.”, ”{“, ”}”で構成された文字列がある && 今
    まで抽出した URL の末尾 n 文字 == その文字列の先頭 n 文字)
    一致した 2 つの文字列を結合させ, それをベース URL とみなす
end

(2) パラメータに関するルール

if (HTML ファイルの 1 行の一部 == “http://” ではじまり, ”?” が記述されている
　&& “?” 以降は英数字または”.”, “-”, “_”,”=“, ”英数字または”.”, “-”,
　“_” “” &” のパターンを持つ文字列が存在する)
　　“英数字または”.”, “-”, “_” “”=“” 英数字または”.”, “-”,
　　“_” “の組み合わせのうち左辺の文字列をパラメータ名とみなす
end

if (HTML ファイルの 1 行の一部 ==“&”, “英数字または”.”, “-”, “_” “,”=“,
　“英数字または”.”, “-”, “_” “ のパターンを持つ文字列が存在する)
　　“英数字または”.”, “-”, “_” “”=“” 英数字または”.”, “-”,
　　“_” “の組み合わせのうち左辺の文字列をパラメータ名とみなす
end

if (定義リストが存在する)
　定義リスト内の文字列をパラメータ名とみなす
end

if (パラメータ名を抽出した定義リストの要素内の文字列一部 ==リスト pr の文字列)
　パラメータが必須か否かの情報であるとみなす
end

if (div 要素の class 属性の値 == リスト pn の文字列)
　その で定義されている語をパラメータ名とみなす
end

if (パラメータ名を抽出した div の要素内の文字列一部 ==リスト pd の文字列)
　パラメータのデフォルト値とみなす
end

```
if（パラメータ名を抽出したdivの要素の文字列の一部 == “example”,”values”
    といった例を示しているような文字列）
    if（例として数字が示されている）
        そのパラメータの型を整数型とみなす
    else if（例として文字列が示されている）
        そのパラメータの型を文字列とみなす
    else if（例として”true”や”false”が示されている）
        そのパラメータの型をbool型とみなす
    end
end

if（span要素のclass属性の値 == リストpnの文字列）
    その<div>で定義されている語をパラメータ名とみなす
end

if（パラメータ名を抽出したspanの要素内の文字列一部 ==リストpdの文字列）
    パラメータのデフォルト値とみなす
end

if（パラメータ名を抽出したspanの要素の文字列の一部 == “example”,”values”といった例を示しているような文字列）
    if（例として数字が示されている）
        そのパラメータの型を整数型とみなす
    else if（例として文字列が示されている）
        そのパラメータの型を文字列とみなす
    else if（例として”true”や”false”が示されている）
        そのパラメータの型をbool型とみなす
    end
end
```

```
if（ある表の前の数行もしくは表の要素内の文字列の一部 ==リスト pn の文字列）
    if（上記の if 文に一致した文字列 == 表の第 1 列目の要素である &&上記の if 文に一
        致した文字列 ==“英数字または”.”, “-”, “_” “）
            パラメータ名とみなす
end

if（パラメータ名を抽出した表の要素内の文字列一部 ==リスト pt の文字列
    && それ以外の文字列の比が少ない）
        パラメータの型とみなす
end

if（パラメータ名を抽出した表の要素内の文字列一部 ==リスト pr の文字列
    && それ以外の文字列の比が少ない）
        パラメータが必須か否かの情報であるとみなす
end

if（パラメータ名を抽出した HTML の一行の一部 == リスト pr の文字列）
    パラメータが必須か否かの情報であるとみなす
end

if（パラメータ名を抽出した表の要素内の文字列一部 ==リスト pd の文字列）
    パラメータのデフォルト値とみなす
end

if（パラメータ名として抽出した文字列の一部 == “(スペース)”
  || パラメータ名として抽出した文字列の一部 == “{”
  || パラメータ名として抽出した文字列の一部 == “}”）
    そのパラメータに関する情報(パラメータ名，パラメータの型，パラメータが必須か否
    か，パラメータのデフォルト値)のスコアを減算する
end

if（パラメータ名として抽出した文字列の一部 == “,”）
    そのパラメータ名を”,”で分割しそれぞれを個々のパラメータ名とみなす
end
```

(3) リクエスト・レスポンスの形式に関するルール

if (HTML ファイルの 1 行 == キーワードリストにある”GET”を含む文字列と一致する
    || HTML ファイルのまとまった複数行 ==リスト rg にある文字列と一致する)
        リクエストの形式を GET とみなす
else if (HTML ファイルの 1 行 == リスト rps にある文字列と一致する
    || HTML ファイルのまとまった複数行 == リスト rps にある文字列と一致する)
        リクエストの形式を POST とみなす
else if (HTML ファイルの一行 == リスト rpu にある文字列と一致する
    || HTML ファイルのまとまった複数行 == リスト rpu にある文字列と一致する)
        リクエストの形式を PUT とみなす
end

if (HTML ファイルの一行 == リスト res にある文字列と一致する)
        それらをレスポンスの形式とみなす
end

第 6 章 提案手法の評価

6.1 評価方法

本章では．本研究の手法およびそれを実装したプロトタイプシステムの評価とそれに基づく検討について述べる．

6.1.1 評価実験概要

プロトタイプシステムのオンラインドキュメントの情報抽出を評価するため，評価実験を行った．実験対象は Programmable Web に登録されている Web API のオンラインドキュメントのうち，被マッシュアップ件数の多いもの上位 30 件とした．この実験対象を選択した理由は，これらの Web API のオンラインドキュメントは Web API のオンラインドキュメントの基準となっているといえるからである．被マッシュアップ件数が多いということは，よく使われる人気の高い Web API であるということであり，これらのオンラインドキュメントは後に作成される Web API のオンラインドキュメントの参考となる可能性が高く，今後も同様の記述形式のオンラインドキュメントが多数を占める可能性が高い．

評価実験は以下の手順で行った．評価実験では，5.1.3 で述べたユーザによる抽出結果の修正機能は利用しないこととした．

(1) 実験対象 30 件のうち，上位 10 件のオンラインドキュメントを見てそこから情報が抽出できるよう抽出ルールを作成する．

　この上位 10 件のオンラインドキュメントをグループ A とする．またそれ以外の 20 件をグループ B とする．

(2) グループ A に対して抽出精度の算出を行い，ある一定以上の抽出精度が得られるまで抽出ルールの追加を行う．

　抽出精度の算出は 4 章の予備実験時と同様に，オンラインドキュメントからあらかじめ人手で抽出しておいた情報と，プロトタイプシステムが抽出した情報を比較することによって行った．

(3) (2)の結果得られたルールで，グループ B に対しての抽出精度も算出する．

　新たな抽出ルールの作成を行わない理由は，未知のオンラインドキュメントに対する精度を算出するためである．新たな抽出ルールの作成を継続した場合，その抽出ルールがグループ B のオンラインドキュメントのうちの一部のオンラインドキュメントの精度のみに作用するものとなってしまう可能性もあり，その場合抽出精度に意図的な偏りが生まれてしまう．

この抽出精度の結果と 30 件のオンラインドキュメントそれぞれのタイプ，実装した抽出

ルールから，どのようにして抽出ルールを作成するのが良いか，オンラインドキュメントのタイプと抽出ルールの関係，プロトタイプシステムのプログラムライブラリ生成における有効性について検討した.

6.1.2 実験対象

実験対象とした Programmable Web に登録されているオンラインドキュメントを以下に示す.

(1) Twitter API　https://dev.twitter.com/docs (参照 2012-12-13)
(2) Flickr API　http://code.flickr.com/ (参照 2012-12-13)
(3) Amazon eCommerce API
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html (参照 2012-12-13)
(4) Facebook API　http://developers.facebook.com/ (参照 2012-12-13)
(5) Twilio API　http://www.twilio.com/ (参照 2012-12-13)
(6) eBay API　https://www.x.com/developers/ebay (参照 2012-12-13)
(7) Last.fm API　http://www.lastfm.jp/api (参照 2012-12-13)
(8) Twilio SMS API　http://www.twilio.com/api/sms (参照 2012-12-13)
(9) del.icio.us API　http://delicious.com/developers (参照 2012-12-13)
(10) Yahoo Web Search API　http://developer.yahoo.com/search/web/webSearch.html (参照 2012-12-13)
(11) Yahoo Geocoding API　http://developer.yahoo.com/maps/rest/V1/geocode.html (参照 2012-12-13)
(12) foursquare API　https://developer.foursquare.com/ (参照 2012-12-13)
(13) GeoNames API　http://www.geonames.org/export/ (参照 2012-12-13)
(14) Box.net API　http://developers.box.com/docs/ (参照 2012-12-13)
(15) Wikipedia API　http://www.mediawiki.org/wiki/API (参照 2012-12-13)
(16) Amazon EC2 API
http://aws.amazon.com/archives/3215?_encoding=UTF8&jiveRedirect=1 (参照 2012-12-13)
(17) Bing API　http://www.bing.com/developers/ (参照 2012-12-13)
(18) Yahoo Image Search API
http://developer.yahoo.com/search/image/V1/imageSearch.html (参照 2012-12-13)
(19) Shopping.com API
https://partners.shopping.com/apiHome.action?apiSuiteSelection=sdc101 (参照 2012-12-13)

(20) Yahoo BOSS API　http://developer.yahoo.com/boss/search/ (参照 2012-12-13)
(21) Yahoo Local Search API　http://developer.yahoo.com/search/local/localSearch.html (参照 2012-12-13)
(22) Freebase API　http://wiki.freebase.com/wiki/Freebase_API (参照 2012-12-13)
(23) Yelp API　http://www.yelp.com/developers (参照 2012-12-13)
(24) LinkedIn API https://developer.linkedin.com/ (参照 2012-12-13)
(25) Eventful API http://api.eventful.com/ (参照 2012-12-13)
(26) Salesforce.com API http://developer.force.com/ (参照 2012-12-13)
(27) Instagram API http://instagram.com/developer/ (参照 2012-12-13)
(28) geocoder API http://geocoder.us/help/ (参照 2012-12-13)
(29) Upcoming API http://upcoming.yahoo.com/services/api/ (参照 2012-12-13)
(30) Commission Junction API http://www.cj.com/webservices (参照 2012-12-13)

6.1.3 抽出精度の算出

抽出結果は適合率と再現率で表現し，最終的な抽出精度は f 値によって表現することとした．適合率と再現率，f 値の式はそれぞれ以下のようになる [30].

A：抽出された情報のうち，誤って抽出したもの
B：抽出された情報のうち，正解だったもの
C：抽出されなかった情報のうち，正解だったもの

$$再現率 = \frac{B}{B+C}$$

$$適合率 = \frac{B}{A+B}$$

$$f値 = \frac{2 \times 再現率 \times 適合率}{再現率 + 適合率}$$

適合率が低い場合，正解ではない情報が含まれることになるので，ライブラリ生成の際に有効でないパラメータやベース URL が指定できるライブラリが生成されてしまう可能性がある．再現率が低い場合，正解の情報全てを漏れ無く取得することに失敗していることになるので，ライブラリ生成の際に Web API の一部機能が利用不可能なライブラリが生成されてしまう可能性がある．ライブラリ生成において全ての機能を備えた利用可能な Web API ライブラリを生成するためには，適合率，再現率は共に 1 である必要があるといえる．今回の実験では，グループ A の全てのオンラインドキュメントの f 値が 0.8 以上となるまで抽出ルールの作成を行った．f 値を 0.8 を超えて 1 に近づける場合，抽出ルールの条件が，各オンラインドキュメントに対して限定的なものばかりになり，他のオンラインドキュメ

ントに対して有効な抽出ルールとなるケースが少ない．実験では未知のオンラインドキュメントに対しての抽出精度を見ることと，多くのオンラインドキュメントに対して有効な抽出ルールは何であるかを見ることが目的であるため，抽出ルール作成の条件となる f 値を 0.8 とした．

6.2 実際のオンラインドキュメントに対する抽出精度

6.1.2 で示した実験対象に対する再現率，適合率，f 値を表 6.1 と表 6.2 に示す．表 6.1 と表 6.2 では，3.3 で示した 7 要素それぞれの再現率，適合率，f 値の平均を示している．表 6.1 がグループ A の結果，表 6.2 がグループ B の結果である．7 要素それぞれでの再現率，適合率，f 値は表 6.3 から表 6.16 に示す．また各オンラインドキュメントが 5.2 で述べるグループのうちどちらに属するかも表中に示した．

表 6.1 グループ A のオンラインドキュメントに対する抽出精度(7 要素平均)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	0.90	1.00	0.95
Flickr	1	0.73	0.94	0.82
Amazon eCommerce	1	0.77	0.79	0.78
Facebook	2	0.57	0.83	0.67
Twilio	1	1.00	0.88	0.93
eBay	2	0.53	0.80	0.64
Last.fm	1	0.50	1.00	0.67
Twilio SMS	1	0.84	0.81	0.83
del.icio.us	2	0.82	0.89	0.86
Yahoo Search	1	0.78	0.82	0.80
タイプ 1 の平均		0.79	0.89	0.83
タイプ 2 の平均		0.64	0.84	0.72

表 6.2 グループ B のオンラインドキュメントに対する抽出精度(7 要素平均)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	0.53	0.63	0.58
foursquare API	1	0.20	0.20	0.20
GeoNames API	2	0.01	0.17	0.01
Box.net API	2	0.06	0.04	0.05
Wikipeda API	2	0.23	0.26	0.24
Amazon EC2 API	1	0.31	0.60	0.41
Bing API	2	0.08	0.25	0.13
Yahoo Image Search API	1	0.42	0.75	0.53
Shopping.com API	2	0.68	0.69	0.69
Yahoo BOSS API	2	0.17	0.35	0.23
Yahoo Local Search API	1	0.28	0.53	0.37
Freebase API	1	0.41	0.47	0.44
Yelp API	1	0.55	0.59	0.57
LinkedIn API	1	0.44	0.57	0.50
Eventful API	1	0.34	0.63	0.44
Salesforce.com API	2	0.00	0.00	0.00
Instagram API	2	0.50	0.50	0.50
geocoder API	2	0.19	0.25	0.21
Upcoming API	2	0.27	0.30	0.28
Commission Junction API	1	0.00	0.00	0.00
タイプ 1 の平均		0.35	0.50	0.40
タイプ 2 の平均		0.22	0.28	0.23

表 6.3 グループ A のオンラインドキュメントに対する抽出精度(ベース URL)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	0.50	1.00	0.67
Flickr	1	0.50	1.00	0.67
Amazon eCommerce	1	1.00	1.00	1.00
Facebook	2	0.03	1.00	0.57
Twilio	1	1.00	1.00	1.00
eBay	2	0.08	1.00	0.14
Last.fm	1	1.00	1.00	1.00
Twilio SMS	1	1.00	1.00	1.00
del.icio.us	2	0.52	0.73	0.61
Yahoo Search	1	1.00	1.00	1.00
タイプ 1 の平均		0.86	1.00	0.91
タイプ 2 の平均		0.21	0.91	0.44

表 6.4 グループ B のオンラインドキュメントに対する抽出精度(ベース URL)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	0.33	1.00	0.50
foursquare API	1	0.00	0.00	0.00
GeoNames API	2	0.05	1.00	0.09
Box.net API	2	0.00	0.00	0.00
Wikipeda API	2	0.50	0.67	0.57
Amazon EC2 API	1	0.33	1.00	0.50
Bing API	2	0.33	1.00	0.50
Yahoo Image Search API	1	0.50	1.00	0.67
Shopping.com API	2	0.50	1.00	0.67
Yahoo BOSS API	2	0.20	0.17	0.18
Yahoo Local Search API	1	0.17	1.00	0.29
Freebase API	1	0.50	1.00	0.67
Yelp API	1	1.00	1.00	1.00
LinkedIn API	1	0.33	1.00	0.50
Eventful API	1	0.33	1.00	0.50
Salesforce.com API	2	0.00	0.00	0.00
Instagram API	2	1.00	1.00	1.00
geocoder API	2	0.75	1.00	0.86
Upcoming API	2	0.33	1.00	0.50
Commission Junction API	1	0.00	0.00	0.00
タイプ 1 の平均		0.35	0.80	0.46
タイプ 2 の平均		0.37	0.68	0.44

表 6.5 グループ A のオンラインドキュメントに対する抽出精度(パラメータ名)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	0.90	1.00	0.95
Flickr	1	0.67	1.00	0.80
Amazon eCommerce	1	0.78	0.97	0.87
Facebook	2	0.67	0.50	0.57
Twilio	1	1.00	0.50	0.67
eBay	2	0.07	0.23	0.10
Last.fm	1	0.25	1.00	0.40
Twilio SMS	1	0.38	0.23	0.29
del.icio.us	2	1.00	1.00	1.00
Yahoo Search	1	0.64	1.00	0.78
タイプ 1 の平均		0.66	0.81	0.68
タイプ 2 の平均		0.58	0.58	0.56

表 6.6 グループ B のオンラインドキュメントに対する抽出精度(パラメータ名)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	0.88	1.00	0.93
foursquare API	1	0.00	0.00	0.00
GeoNames API	2	0.00	0.00	0.00
Box.net API	2	0.33	0.25	0.29
Wikipeda API	2	0.20	0.10	0.13
Amazon EC2 API	1	0.20	1.00	0.33
Bing API	2	0.00	0.00	0.00
Yahoo Image Search API	1	0.42	1.00	0.59
Shopping.com API	2	0.86	0.39	0.54
Yahoo BOSS API	2	0.30	0.89	0.44
Yahoo Local Search API	1	0.43	1.00	0.60
Freebase API	1	0.40	0.29	0.33
Yelp API	1	0.55	0.79	0.65
LinkedIn API	1	0.74	1.00	0.85
Eventful API	1	0.02	0.50	0.04
Salesforce.com API	2	0.00	0.00	0.00
Instagram API	2	0.00	0.00	0.00
geocoder API	2	0.00	0.00	0.00
Upcoming API	2	1.00	0.50	0.67
Commission Junction API	1	0.00	0.00	0.00
タイプ 1 の平均		0.36	0.66	0.43
タイプ 2 の平均		0.27	0.21	0.21

表 6.7 グループ A のオンラインドキュメントに対する抽出精度(パラメータの型)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	1.00	1.00	1.00
Flickr	1	NA	NA	NA
Amazon eCommerce	1	0.76	0.73	0.75
Facebook	2	NA	NA	NA
Twilio	1	NA	NA	NA
eBay	2	NA	NA	NA
Last.fm	1	NA	NA	NA
Twilio SMS	1	NA	NA	NA
del.icio.us	2	NA	NA	NA
Yahoo Search	1	0.65	1.00	0.79
タイプ 1 の平均		0.80	0.91	0.85
タイプ 2 の平均		NA	NA	NA

表 6.8 グループ B のオンラインドキュメントに対する抽出精度(パラメータの型)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	0.75	1.00	0.86
foursquare API	1	NA	NA	NA
GeoNames API	2	0.00	0.00	0.00
Box.net API	2	0.00	0.00	0.00
Wikipeda API	2	NA	NA	NA
Amazon EC2 API	1	0.00	0.00	0.00
Bing API	2	NA	NA	NA
Yahoo Image Search API	1	0.58	1.00	0.74
Shopping.com API	2	NA	NA	NA
Yahoo BOSS API	2	NA	NA	NA
Yahoo Local Search API	1	0.71	0.83	0.77
Freebase API	1	0.00	0.00	0.00
Yelp API	1	0.75	0.27	0.40
LinkedIn API	1	0.00	0.00	0.00
Eventful API	1	0.00	0.00	0.00
Salesforce.com API	2	NA	NA	NA
Instagram API	2	NA	NA	NA
geocoder API	2	NA	NA	NA
Upcoming API	2	NA	NA	NA
Commission Junction API	1	NA	NA	NA
タイプ 1 の平均		0.47	0.52	0.46
タイプ 2 の平均		0.00	0.00	0.00

表 6.9 グループ A のオンラインドキュメントに対する抽出精度(パラメータのデフォルト値)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	1.00	1.00	1.00
Flickr	1	NA	NA	NA
Amazon eCommerce	1	1.00	1.00	1.00
Facebook	2	NA	NA	NA
Twilio	1	NA	NA	NA
eBay	2	NA	NA	NA
Last.fm	1	0.50	1.00	0.67
Twilio SMS	1	NA	NA	NA
del.icio.us	2	NA	NA	NA
Yahoo Search	1	0.30	0.60	0.40
タイプ 1 の平均		0.70	0.90	0.77
タイプ 2 の平均		NA	NA	NA

表 6.10 グループ B のオンラインドキュメントに対する抽出精度(パラメータのデフォルト値)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	0.00	0.00	0.00
foursquare API	1	0.00	0.00	0.00
GeoNames API	2	0.00	0.00	0.00
Box.net API	2	0.00	0.00	0.00
Wikipeda API	2	NA	NA	NA
Amazon EC2 API	1	NA	NA	NA
Bing API	2	NA	NA	NA
Yahoo Image Search API	1	0.60	0.50	0.55
Shopping.com API	2	NA	NA	NA
Yahoo BOSS API	2	NA	NA	NA
Yahoo Local Search API	1	0.33	0.25	0.29
Freebase API	1	0.00	0.00	0.00
Yelp API	1	NA	NA	NA
LinkedIn API	1	0.00	0.00	0.00
Eventful API	1	NA	NA	NA
Salesforce.com API	2	NA	NA	NA
Instagram API	2	NA	NA	NA
geocoder API	2	NA	NA	NA
Upcoming API	2	NA	NA	NA
Commission Junction API	1	NA	NA	NA
タイプ 1 の平均		0.16	0.13	0.14
タイプ 2 の平均		0.00	0.00	0.00

表 6.11 グループ A のオンラインドキュメントに対する抽出精度(パラメータが必須か否か)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	0.90	1.00	0.95
Flickr	1	0.47	0.70	0.56
Amazon eCommerce	1	0.05	0.06	0.06
Facebook	2	NA	NA	NA
Twilio	1	NA	NA	NA
eBay	2	NA	NA	NA
Last.fm	1	0.25	1.00	0.40
Twilio SMS	1	NA	NA	NA
del.icio.us	2	0.94	0.94	0.94
Yahoo Search	1	0.64	1.00	0.78
タイプ 1 の平均		0.46	0.75	0.55
タイプ 2 の平均		0.94	0.94	0.94

表 6.12 グループ B のオンラインドキュメントに対する抽出精度(パラメータが必須か否か)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	0.25	0.29	0.27
foursquare API	1	0.00	0.00	0.00
GeoNames API	2	0.00	0.00	0.00
Box.net API	2	0.00	0.00	0.00
Wikipeda API	2	NA	NA	NA
Amazon EC2 API	1	0.00	0.00	0.00
Bing API	2	NA	NA	NA
Yahoo Image Search API	1	0.42	1.00	0.59
Shopping.com API	2	NA	NA	NA
Yahoo BOSS API	2	NA	NA	NA
Yahoo Local Search API	1	0.04	0.10	0.06
Freebase API	1	0.00	0.00	0.00
Yelp API	1	0.45	0.90	0.60
LinkedIn API	1	0.00	0.00	0.00
Eventful API	1	NA	NA	NA
Salesforce.com API	2	NA	NA	NA
Instagram API	2	NA	NA	NA
geocoder API	2	NA	NA	NA
Upcoming API	2	0.00	0.00	0.00
Commission Junction API	1	NA	NA	NA
タイプ 1 の平均		0.15	0.29	0.19
タイプ 2 の平均		0.00	0.00	0.00

表 6.13 グループ A のオンラインドキュメントに対する抽出精度(リクエストの形式)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	1.00	1.00	1.00
Flickr	1	1.00	1.00	1.00
Amazon eCommerce	1	NA	NA	NA
Facebook	2	NA	NA	NA
Twilio	1	1.00	1.00	1.00
eBay	2	1.00	1.00	1.00
Last.fm	1	NA	NA	NA
Twilio SMS	1	1.00	1.00	1.00
del.icio.us	2	NA	NA	NA
Yahoo Search	1	NA	NA	NA
タイプ 1 の平均		1.00	1.00	1.00
タイプ 2 の平均		1.00	1.00	1.00

表 6.14 グループ B のオンラインドキュメントに対する抽出精度(リクエストの形式)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	NA	NA	NA
foursquare API	1	1.00	1.00	1.00
GeoNames API	2	NA	NA	NA
Box.net API	2	NA	NA	NA
Wikipeda API	2	NA	NA	NA
Amazon EC2 API	1	1.00	1.00	1.00
Bing API	2	0.00	0.00	0.00
Yahoo Image Search API	1	NA	NA	NA
Shopping.com API	2	NA	NA	NA
Yahoo BOSS API	2	0.00	0.00	0.00
Yahoo Local Search API	1	NA	NA	NA
Freebase API	1	1.00	1.00	1.00
Yelp API	1	0.00	0.00	0.00
LinkedIn API	1	1.00	1.00	1.00
Eventful API	1	NA	NA	NA
Salesforce.com API	2	0.00	0.00	0.00
Instagram API	2	NA	NA	NA
geocoder API	2	0.00	0.00	0.00
Upcoming API	2	0.00	0.00	0.00
Commission Junction API	1	NA	NA	NA
タイプ 1 の平均		0.80	0.80	0.80
タイプ 2 の平均		0.00	0.00	0.00

表 6.15 グループ A のオンラインドキュメントに対する抽出精度(レスポンスの形式)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Twitter	1	1.00	1.00	1.00
Flickr	1	1.00	1.00	1.00
Amazon eCommerce	1	1.00	1.00	1.00
Facebook	2	1.00	1.00	1.00
Twilio	1	1.00	1.00	1.00
eBay	2	1.00	1.00	1.00
Last.fm	1	NA	NA	NA
Twilio SMS	1	1.00	1.00	1.00
del.icio.us	2	NA	NA	NA
Yahoo Search	1	1.00	1.00	1.00
タイプ 1 の平均		1.00	1.00	1.00
タイプ 2 の平均		1.00	1.00	1.00

表 6.16 グループ B のオンラインドキュメントに対する抽出精度(レスポンスの形式)

オンラインドキュメント名	タイプ	適合率	再現率	f 値
Yahoo Geocoding API	1	1.00	0.20	0.67
foursquare API	1	NA	NA	NA
GeoNames API	2	0.00	0.00	0.00
Box.net API	2	0.00	0.00	0.00
Wikipeda API	2	0.00	0.00	0.00
Amazon EC2 API	1	NA	NA	NA
Bing API	2	0.00	0.00	0.00
Yahoo Image Search API	1	0.00	0.00	0.00
Shopping.com API	2	NA	NA	NA
Yahoo BOSS API	2	NA	NA	NA
Yahoo Local Search API	1	0.00	0.00	0.00
Freebase API	1	1.00	1.00	1.00
Yelp API	1	NA	NA	NA
LinkedIn API	1	1.00	1.00	1.00
Eventful API	1	1.00	1.00	1.00
Salesforce.com API	2	0.00	0.00	0.00
Instagram API	2	NA	NA	NA
geocoder API	2	0.00	0.00	0.00
Upcoming API	2	0.00	0.00	0.00
Commission Junction API	1	NA	NA	NA
タイプ 1 の平均		0.67	0.53	0.61
タイプ 2 の平均		0.00	0.00	0.00

グループ A に，より対応する抽出ルールを実装していたためグループ B の抽出精度はグループ A の抽出精度より劣る結果となった．抽出する要素の中でも，リクエストとレスポンスの形式に関しては，表現形式が表内，文中，独立した div 要素内など記述形式が多様であるため，抽出精度が低くなっている．しかしリクエストとレスポンスの形式に関する抽出ルール自体は他の要素に比べ単純なものであるため，キーワードリストへの単語の追加により，抽出精度を他の要素に比べ容易に上昇させることができると考えられる．

評価実験において，Commission Junction API のオンラインドキュメントに関してはどの情報も取得できていない．これは Commission Junction API のオンラインドキュメントでは，抽出すべき情報は全て JavaScript 内に記述されており，HTML は単純なレイアウトの指定のみに用いられていたのが原因である．プロトタイプシステムでは入力として与えているのはオンラインドキュメントの HTML ファイルのみであるため Commission Junction API からは情報抽出が行えなかった．

6.3 抽出ルールに関する考察

6.2 の結果より，タイプ 1 のドキュメントはグループ B でも f 値が高く，逆にタイプ 2 のドキュメントはグループ B だと f 値が低い．これより，タイプ 1 のドキュメントは，各ドキュメント内で共通する抽出ルールが多いことがわかる．特にタイプ 1 は表を用いて情報を記述する場面が多かったため，表中から要素を抽出する抽出ルールが有効に機能した．また，http リクエストの記入例が示されるケースも同様に多く，http リクエストの文字列は Web API のオンラインドキュメント特有の記述であることもあり，例からベース URL とパラメータ名を抽出する抽出ルールも有効であった．

Shopping.com API の平均 f 値が比較的高いのは，Shopping.com のオンラインドキュメントから得られる要素がベース URL とパラメータ名のみであり，それらの抽出精度が高いためである．これらの抽出精度が高い理由として，Shopping.com にはリクエスト URL の例が多く示されており，以下の抽出ルールによってパラメータ名が正確に抽出できたためであると考えられる．

```
if（HTML ファイルの 1 行の一部 == “http://”ではじまり，”?”が記述されている
    && “?”以降は”英数字または”.”，“-”，“_” “,” = “，”英数字または”.”，
     “-”，“_” “” &”のパターンを持つ文字列が存在する）
        “英数字または”.”，“-”，“_” “” = “” 英数字または”.”，“-”，
        “_”“の組み合わせのうち左辺の文字列をパラメータ名とみなす
end
```

タイプ 2 のドキュメントはタイプ 1 に比べ f 値が低い．タイプ 2 のドキュメントの f 値を上げるためには，そのドキュメントに適した抽出ルールを実装する必要がある．例えば，GeoNames API の f 値を理論上 0.8 以上にするには以下のような抽出ルールの実装が必要である．

```
if（パラメータ名を抽出した表の要素内の文字列一部 == ”Default is”）
    “Default is”の後の文字列をパラメータのデフォルト値とみなす
end

if（パラメータ名を抽出した表の要素内の文字列一部 == ”DEFAULT =”）
    ”DEFAULT =”の後の文字列をパラメータのデフォルト値とみなす
end
```

```
if (HTML ファイルの 1 行の一部 == ” default =”
    && ” default = names in local language”が現れる直前の英単語がパラメータ名と
    して既に抽出されている)
        ” default =”の後の文字列をデフォルト値とみなす

if (HTML ファイルの 1 行の一部 == ” default”
    && HTML ファイルの 1 行の一部 != ” default =”
    && ” default = names in local language”が現れる直前の英単語がパラメータ名と
    して既に抽出されている)
        ” default ”の後の文字列をデフォルト値とみなす
end

if (HTML ファイルの 1 行の一部 == ” default = names in local language”
    && ” default = names in local language”が現れる直前の英単語がパラメータ名と
    して既に抽出されている)
        このプログラムが実行されている地域の言語を ISO-639-1 language code に従う
        形式でパラメータ名に対応するデフォルト値とみなす
end

if (HTML ファイルの 1 行の一部 == ” request method:” )
        ” request method:”の後の文字列をリクエストの形式とみなす
end
```

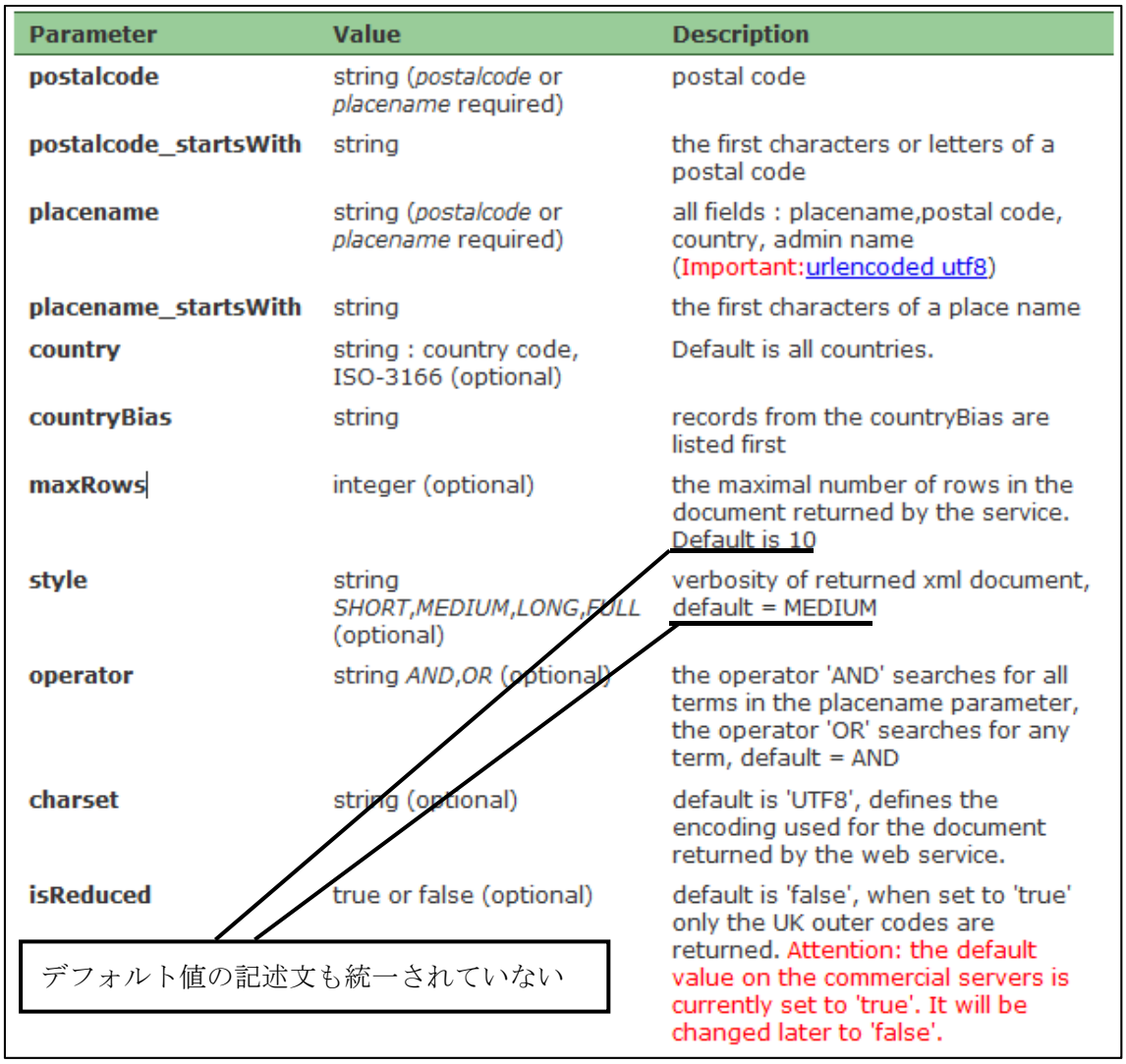
GeoNames API のオンラインドキュメントの抜粋を図 6.1 と図 6.2 に示す．この GeoNames API の例では，デフォルト値の記述方式が表内，英語の文章内と統一した形式で記述されておらず，記述される際の単語や記号の並びも一意でない．また，デフォルト値の値そのものを記述する場合だけでなく，意味を英文で表現するケースも存在する．

この他に, Bing API ではパラメータが表や箇条書き等の整形された形での記述ではなく，文章中に記述される(図 6.3)．またパラメータ名は英単語による単純な命名により定義されているものの他に，クラスとして構造的に定義されており，「.」を用いて階層的に指定するものも存在しており，詳細なパラメータの情報は他のページにて示されている．文章中からパラメータに関する記述を特定するためには自然言語処理的な手法が必要となる場合もあると考える．また，複数ページ間に情報が分散している場合，どのページの情報が何の情報とどう対応するのか，ページ間の関係性を考慮する必要もあると考える．

以上のように，タイプ 2 は抽出ルールの作成が難しく，限定的な多数のルールを作成する必要がある．一方でタイプ 1 は比較的抽出ルールの作成が容易で，1 つの抽出ルールが複

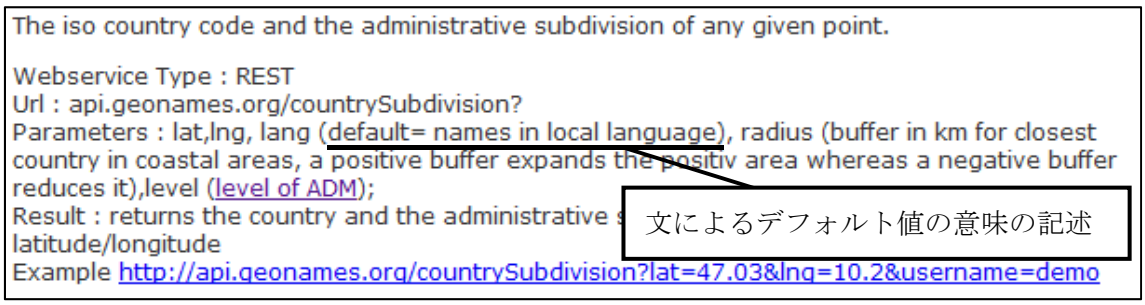
数のオンラインドキュメントに対応していることが多い.

図 6.1 GeoNames API オンラインドキュメント抜粋(表内におけるデフォルト値の記述)



Parameter	Value	Description
postalcode	string (*postalcode* or *placename* required)	postal code
postalcode_startsWith	string	the first characters or letters of a postal code
placename	string (*postalcode* or *placename* required)	all fields : placename,postal code, country, admin name (Important:urlencoded utf8)
placename_startsWith	string	the first characters of a place name
country	string : country code, ISO-3166 (optional)	Default is all countries.
countryBias	string	records from the countryBias are listed first
maxRows	integer (optional)	the maximal number of rows in the document returned by the service. Default is 10
style	string *SHORT,MEDIUM,LONG,FULL* (optional)	verbosity of returned xml document, default = MEDIUM
operator	string *AND,OR* (optional)	the operator 'AND' searches for all terms in the placename parameter, the operator 'OR' searches for any term, default = AND
charset	string (optional)	default is 'UTF8', defines the encoding used for the document returned by the web service.
isReduced	true or false (optional)	default is 'false', when set to 'true' only the UK outer codes are returned. Attention: the default value on the commercial servers is currently set to 'true'. It will be changed later to 'false'.

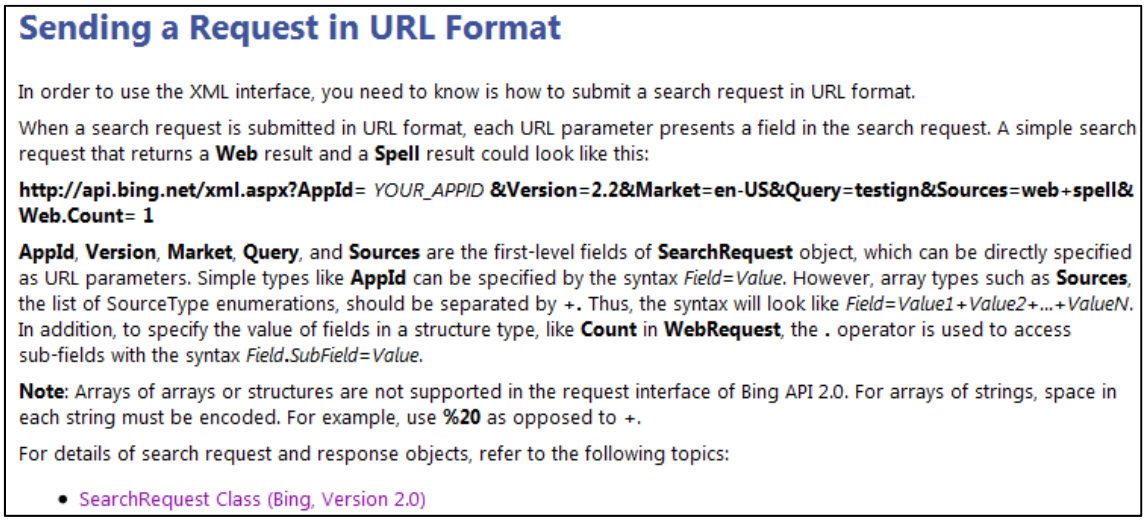
図 6.2 GeoNames API オンラインドキュメント抜粋(文中におけるデフォルト値の記述)



The iso country code and the administrative subdivision of any given point.

Webservice Type : REST
Url : api.geonames.org/countrySubdivision?
Parameters : lat,lng, lang (default= names in local language), radius (buffer in km for closest country in coastal areas, a positive buffer expands the positiv area whereas a negative buffer reduces it),level (level of ADM);
Result : returns the country and the administrative s
latitude/longitude
Example http://api.geonames.org/countrySubdivision?lat=47.03&lng=10.2&username=demo

図 6.3 Bing API オンラインドキュメント抜粋



Sending a Request in URL Format

In order to use the XML interface, you need to know is how to submit a search request in URL format.

When a search request is submitted in URL format, each URL parameter presents a field in the search request. A simple search request that returns a **Web** result and a **Spell** result could look like this:

http://api.bing.net/xml.aspx?AppId= *YOUR_APPID* **&Version=2.2&Market=en-US&Query=testign&Sources=web+spell&Web.Count= 1**

AppId, **Version**, **Market**, **Query**, and **Sources** are the first-level fields of **SearchRequest** object, which can be directly specified as URL parameters. Simple types like **AppId** can be specified by the syntax *Field=Value*. However, array types such as **Sources**, the list of SourceType enumerations, should be separated by **+**. Thus, the syntax will look like *Field=Value1+Value2+...+ValueN*. In addition, to specify the value of fields in a structure type, like **Count** in **WebRequest**, the **.** operator is used to access sub-fields with the syntax *Field.SubField=Value*.

Note: Arrays of arrays or structures are not supported in the request interface of Bing API 2.0. For arrays of strings, space in each string must be encoded. For example, use **%20** as opposed to +.

For details of search request and response objects, refer to the following topics:

- SearchRequest Class (Bing, Version 2.0)

6.4 現在のライブラリ開発プロセスとの比較

現在 Web API のライブラリ開発は，開発者が Web API のオンラインドキュメントを元に人手でコーディングを行なっている．ただ，特定の Web API には，ライブラリ生成を手助けするツールやサービスが用意されている場合もある．例えば Google Maps では，任意の地点を指した状態の地図表示機能を Web サイトに埋め込めるよう html コードとして提供するためのツールが用意されている．しかし多くの Web API にはこのようなツールは用意されていない．

本研究で作成したプロトタイプシステムを用いれば，今まで人出で行なっていた Web API のライブラリ作成を機械的に行うことができる．プロトタイプシステムを用いてライブラリを生成する手順は以下のようになる．

(1) プロトタイプシステムにオンラインドキュメントを入力として与える
(2) プロトタイプシステムの抽出結果から，必要であればユーザが抽出された情報を修正する
(3) WADL 形式でプロトタイプシステムから情報を出力する
(4) WADL をライブラリが生成できるツール(Net Beans など)に読み込ませる
(5) そのツールからライブラリを出力する

Net Beans の例では，WADL から Java のライブラリを生成することが可能である．その際生成されるコードの行数の総計は 204 行であり，単純に考えればその分だけコーディング量が減少できたことになる．

Web API のライブラリ開発においては，開発者が行う作業は以下のとおりである．

- コーディング(Web API に関する処理以外)
- コーディング(Web API に関する処理)
- オンラインドキュメントからの仕様の確認

オンラインドキュメントからの仕様の確認では，Web API を利用するにあたり，ベース URL やパラメータ名, レスポンスの形式など Web API 提供者から指定されている各要素の仕様を確認する．

プロトタイプシステムを用いる場合は，コーディングの Web API に関する処理はライブラリとして生成可能であるので必要ないが，代わりにユーザによる抽出された情報の修正が必要となる．よって，このユーザによる抽出された情報の修正が，プロトタイプシステムを用いない場合のコーディングよりも作業量が少ない場合，開発者による Web API のライブラリ開発において，プロトタイプシステムを用いたライブラリの作成が有効であるといえる．

本研究ではグループ A，B 共に，情報抽出の精度はまだ十分であるとは言えないが，5.1.3 で述べたユーザによる情報の修正機能により Web API の全ての機能を備えたライブラリを生成することが可能となっている．表 6.17 にグループ B のオンラインドキュメントにおける抽出すべき情報の数とプロトタイプシステムで実際に抽出できた数を示す．ここで記す情報の数とはオンラインドキュメント中に出現した, 3.3 で示した 7 要素全ての合計である．表 6.17 より，プロトタイプシステムの現在の抽出精度では，平均で約 14 の情報の追加が必要であることがわかる．また，表 6.2 より，適合率の平均は約 0.28 である．これと 6.1.3 の式から，誤って抽出された情報は平均で約 28 あることがわかる．ユーザが 14 の追加と 28 の削除をプロトタイプシステムが出力したハイライトされたオンラインドキュメントを見ながら行う際の作業量と，200 行程度のコーディングの際の作業量を比較した場合，前者の作業量の方が少ないといえる．またコーディングの場合は各プログラミング言語を用いてプログラムが可能な開発者でないとライブラリ作成が行えないが，プロトタイプシステムでは WADL から各プログラミング言語のライブラリを生成するツールや統合開発環境があれば，生成したいライブラリのプログラミング言語に詳しくない開発者でもライブラリを生成することが可能である．

以上のことから，Web API のライブラリ開発はプロトタイプシステムを用いる方が，プロトタイプシステムを用いない場合に比べて作業量が少ないといえる．

表 6.17 グループ B のオンラインドキュメントにおける抽出した情報の数

オンラインドキュメント名	タイプ	抽出できた数	抽出すべき情報の数
Yahoo Geocoding API	1	17	24
foursquare API	1	1	33
GeoNames API	2	26	36
Box.net API	2	3	27
Wikipeda API	2	3	18
Amazon EC2 API	1	4	8
Bing API	2	1	10
Yahoo Image Search API	1	33	39
Shopping.com API	2	20	50
Yahoo BOSS API	2	9	16
Yahoo Local Search API	1	39	66
Freebase API	1	7	51
Yelp API	1	24	37
LinkedIn API	1	20	40
Eventful API	1	3	5
Salesforce.com API	2	0	5
Instagram API	2	1	3
geocoder API	2	3	7
Upcoming API	2	2	8
Commission Junction API	1	0	11
平均		10.8	24.7
タイプ 1 の平均		14.8	31.4
タイプ 2 の平均		6.8	18.0

6.5 提案手法における問題点

本研究の問題点としてはタイプ 2 のオンラインドキュメントへの対応が挙げられる．タイプ 2 のオンラインドキュメントは特に抽出精度が低い結果となっていた. 対策としては, 抽出ルールの追加作成がある．抽出ルールを追加で作成することでタイプ 2 のオンラインドキュメントに対する f 値は上昇させることが可能である. しかしどの程度の数の抽出ルールを作成すれば f 値が高い値になるか現時点で不明である.

現在はプロトタイプシステム内で html のパターンマッチのみを行なっているが，CSS

やJavaScriptなどWebページのレイアウトを考慮した抽出ルールも作成可能なシステムとすれば，背景色や文字サイズ，レイアウト的に隠されているテキストなど今まで取得していなかった情報も取得できるようになるため，タイプ 2 に有効な抽出ルールが作成できるようになるかもしれない．特に JavaScript を用いたオンラインドキュメントの場合，JavaScript 内で Web API の仕様に関する情報が記述され，HTML 中に情報がほとんど記述されない場合もあるため，JavaScript の記述も入力として与え，そこから情報を抽出できるような抽出ルールを作成する必要がある．

Web API を用いたアプリケーションの開発効率を上げることを考える．開発者にとって，オンラインドキュメントからの仕様の確認は，各 Web API によってレイアウトの異なるドキュメントを見なければならず，情報を探す手間もかかる．本研究では Web API 中のライブラリ生成に必要な情報を取得できるので，その情報を利用して，本手法を実装したシステム中に情報の記述形式を統一したオンラインドキュメントを出力する機能も実装すれば，開発者が仕様の確認のために，記述方法に違いがある複数のオンラインドキュメントを読む手間を省くことができると考える．

第 7 章 関連研究

Web API を用いたアプリケーション開発効率向上に関しては井上らの研究 [31]がある．この研究では REST 型と呼ばれる Web API においてのサーバ内でのデータ管理について，アプリケーションに求められる共通機能をそなえたデータベース・システムを開発している．この研究は Web アプリケーションの提供者側のサーバ内の処理機能の効率化であり，開発者側のプログラムライブラリについては言及していない．この他に開発者側の開発効率化として Riabov らの研究 [32]や Lu ら [33]の研究がある．彼らの研究では複数の Web API を利用したマッシュアップの効率化について述べられており，Web API の仕様は既にプログラムとして実装されている前提での研究となっている．以上の研究とは異なり，本研究では Web API の機能を利用するためのプログラムライブラリを生成する．

HTML からの情報抽出という観点では片山らの研究 [34]や池田ら [35]の研究がある．これらの研究では HTML のタグ構造や類似度を利用し HTML の解析を行なっているが，対象となる HTML は Web API のオンラインドキュメントではなく，広範囲の HTML を対象としている．また，単語などの特定の情報を抽出するわけではなく，DOM ツリーからのブロックの抽出や HTML 内に現れるパターンの抽出をしている．本研究の手法は Web API のオンラインドキュメントのみを対象とし，それに見られる規則性を利用して，特定の情報の抽出を行なうものである．

Web コンテンツのラッピング技術という観点では石井ら [36]や Liu ら [37]の HTML のラッピング言語の研究がある．これらは HTML 中から情報を抽出し，XML データ等の他の形式にマッピングすることが可能である．本研究とは HTML 中の情報抽出という点で共通している．石井らや Liu らの研究では主に HTML のタグ構造に注目し情報抽出が行えるよう言語設計がなされているが，本研究では正規表現を元にした独自スクリプトにより HTML のタグ構造だけでなく，前後の特定の単語の関係や URL 等の特殊な文章記述にも対応し，情報抽出が行えるようになっている．

また，研究ではないが本研究の目的と似たものとして Object Network [38]というプロジェクトが存在する．これは Web API で利用されるデータの定義とアクセスフォーマットを共通化するプロジェクトである．HTTP と JSON に関する基本的な規約が決められており，連絡先やカレンダーイベント，ニュース，記事，メッセージ，メディアメタデータ，コメント，レビューなどの共通データのフォーマットも用意されている．しかしこのプロジェクトは現在の Web API に対してフォーマットを統一するよう呼びかけるもので，実際にフォーマットを Object Network が提案しているフォーマットに変換するような仕組みは提供されていない．またフォーマットを共通化しなければならないため，各 Web API 特有の機能が作成しにくく，提案されたフォーマットでは表現できない Web API が登場した場合には対応できない．

第 8 章 おわりに

本研究では開発者がプログラムライブラリを自分で作成する手間を省くため，Web APIのオンラインドキュメントから情報を抽出し，プログラムライブラリの作成に必要な情報を機械的に抽出する手法を提案した．また本手法を実装したプロトタイプシステムを作成した．結果として，5.2で述べたタイプ1のオンラインドキュメントを持つWeb APIに対して，プロトタイプシステムを用いないプログラムライブラリの開発よりもプロトタイプシステムを用いた方が作業量が少なくなることを示し，2.5で述べたプログラムライブラリ作成時の諸問題を解消するのに有効であることを示した．

現在REST型のWeb APIではWADLがWeb APIと併せて提供される事例は少ないが，本研究によってWADLの生成が容易になれば，2章で述べた諸問題の解決につながるだけでなく，WADLを利用したWeb API同士の新たな連携も展望として考えられる．これには例えば複数のWeb APIに対応する共通ライブラリの作成がある．

共通ライブラリの作成は，全く異なる機能を提供しているWeb API同士では難しいが，データベースからの検索など，似ている機能を提供しているWeb API同士であれば，作成が可能なケースがあると考える．例えばCiNii APIとWikipedia API，Yahoo! Web Search APIはデータベース上のリソースに対するキーワード検索機能を提供しており，これらのAPIすべてに検索キーワードを指定するパラメータが存在している．パラメータ名はそれぞれ「q」，「search」，「query」と差はあるが，パラメータとしての役割は同じである．このように似た機能を提供するWeb APIは似た役割のパラメータを持つ．このような似た役割のパラメータをライブラリで共通化し単一のメソッドで複数のWeb APIを利用できるようにすることで，Web APIを用いたマッシュアップ開発の効率化が可能となる．これを実現するためには複数のWADLを読み込み，ライブラリを生成する仕組みが必要となる．

今回はHTMLの記述のみを利用して情報抽出を行ったが，前述したようにCSSやJava ScriptなどWebページ作成に使われる他の記述も利用することで，より良い抽出精度が得られると考える．

謝辞

本論文の作成にあたり，終始適切な助言を賜り，また丁寧に指導して下さった指導教員である阪口哲男先生に深く感謝致します．また，ゼミの場や発表会にて多くのご指摘をして下さった杉本重雄先生，森嶋厚行先生，永森光晴先生，品川徳秀先生，鈴木伸崇先生にも感謝しております．そして私と同じ阪口研究室の後輩である瀬尾崇一郎君を始め，杉本・永森研，森嶋研，鈴木研のゼミ生の皆様にも研究を通じて活発な議論にお付き合い頂きました．ここに感謝の意を表します．

参考文献

[1] ProgrammableWeb - Mashups, APIs, and the Web as Platform. Programmable Web.com.（オンライン）（引用日：2012年6月27日.）http://www.programmableweb.com/.
[2] Twitter.（オンライン）Twitter, Inc.（引用日：2012年12月5日.）https://twitter.com/.
[3] Facebook.（オンライン）Facebook, Inc.（引用日：2012年12月5日.）http://www.facebook.com/.
[4] Google マップ - 地図検索.（オンライン）Google.（引用日：2012年12月5日.）https://maps.google.co.jp/.
[5] グルメ・レストランガイド [食べログ].（オンライン）Kakaku.com, Inc.（引用日：2012年12月5日.）http://tabelog.com/.
[6] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).（オンライン）（引用日：2012年12月13日.）http://www.w3.org/TR/soap12-part1/.
[7] FieldingThomasRoy. Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST).（オンライン）（引用日：2012年12月13日.）http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
[8] 荒本道隆. 3.8 Web 2.0時代のWebServices ～SOAP/REST使い分けの指針. XMLコンソーシアム コミュニティ.（オンライン）2007年06月07日.（引用日：2012年12月31日.）http://xmlconsortium.org/wg/web2.0/teigensho/4--REST-SOAP.html.
[9] Introducing JSON.（オンライン）（引用日：2012年12月31日.）http://www.json.org/.
[10] DardaillerDaniel. W3C WS Package PAS Explanatory Report. W3C.（オンライン）2011年09月20日.（引用日：2012年12月31日.）http://www.w3.org/2010/08/ws-pas.html.
[11] Web Services Description Language (WSDL) 1.1. World Wide Web Consortium.（オンライン）2001年03月15日.（引用日：2012年12月31日.）http://www.w3.org/TR/wsdl.
[12] Sun Microsystems. Web Application Description Language.（オンライン）（引用日：2012年7月23日.）http://www.w3.org/Submission/wadl/.
[13] 吉田稔, 青木秀起. 連載 Webサービスのキホン（4） WSDL : Webサービスのインターフェイス情報. @IT.（オンライン）（引用日：2012年03月12日.）http://www.atmarkit.co.jp/fxml/tanpatsu/21websvc/websvc04.html.
[14] Enterprise Architect - UML Design Tools and UML CASE tools for software development.（オンライン）（引用日：2012年12月31日.）http://www.sparxsystems.com/products/ea/index.html.
[15] Microsoft. Microsoft Visual Studio ホームページ.（オンライン）（引用日：2012年12月31日.）http://www.microsoft.com/ja-jp/dev/.

[16] James Clark, MURATA Makoto. RELAX NG Specification. OASIS.（オンライン）2001 年 12 月 03 日.（引用日: 2013 年 01 月 11 日.）http://relaxng.org/spec-20011203.html.
[17] David C. Fallside, Priscilla Walmsley. XML Schema Part 0: Primer Second Edition. World Wide Web Consortium (W3C).（オンライン）2004 年 10 月 28 日.（引用日: 2013 年 01 月 11 日.）http://www.w3.org/TR/xmlschema-0/.
[18] Register an Internet Media Type for a W3C Spec. World Wide Web Consortium (W3C).（オンライン）2012 年 11 月 06 日.（引用日: 2013 年 01 月 11 日.）http://www.w3.org/2002/06/registering-mediatype.
[19] Twitter Libraries | Twitter Developers.（オンライン）Twitter, Inc.（引用日: 2012 年 8 月 8 日.）https://dev.twitter.com/docs/twitter-libraries.
[20] Web Search Web Service - YDN.（オンライン）Yahoo! Inc.（引用日: 2012 年 12 月 5 日.）http://developer.yahoo.com/search/web/V1/webSearch.html.
[21] CiNii - メタデータ・API - CiNii Articles 論文検索の OpenSearch.（オンライン）（引用日: 2012 年 12 月 31 日.）http://ci.nii.ac.jp/info/ja/api/a_opensearch.html.
[22] twitter4r/twitter4r-core · GitHub.（オンライン）（引用日: 2012 年 12 月 31 日.）https://github.com/twitter4r/twitter4r-core.
[23] SippeyMichael. Changes coming in Version 1.1 of the Twitter API. Twitter Developers.（オンライン）Twitter Inc, 2012 年 08 月 16 日.（引用日: 2013 年 01 月 12 日.）https://dev.twitter.com/blog/changes-coming-to-twitter-api.
[24] Roomann-KurrikArne. A timeline of API announcements. Twitter Developers.（オンライン）Twitter Inc, 2012 年 12 月 03 日.（引用日: 2013 年 01 月 12 日.）https://dev.twitter.com/blog/timeline-api-announcements.
[25] Calendar of API changes. Twitter Developers.（オンライン）Twitter Inc, 2012 年 12 月 06 日.（引用日: 2013 年 01 月 12 日.）https://dev.twitter.com/calendar.
[26] GET statuses/home_timeline | Twitter Developers.（オンライン）Twitter, Inc.（引用日: 2012 年 8 月 8 日.）https://dev.twitter.com/docs/api/1.1/get/statuses/home_timeline.
[27] Yahoo! Answers: Search for Questions - YDN.（オンライン）Yahoo! Inc.（引用日: 2012 年 8 月 8 日.）http://developer.yahoo.com/answers/V1/questionSearch.html.
[28] Welcome to NetBeans.（オンライン）Oracle Corporation.（引用日: 2012 年 12 月 13 日.）http://netbeans.org/.
[29] まつもとゆきひろ. 正規表現. オブジェクト指向スクリプト言語 Ruby リファレンスマニュアル.（オンライン）（引用日: 2013 年 01 月 16 日.）http://doc.ruby-lang.org/ja/1.9.3/doc/spec=2fregexp.html.

[30] 徳永健伸. 情報検索と言語処理.東京大学出版会, 1999. ページ: xi, 234p. 4130654055.
[31] 井上武, 朝倉浩志, 植松幸生, 佐藤浩史, 高橋紀之. 迅速なアプリケーション開発のための Web API データベースシステム.電子情報通信学会, 2010. ページ: p. 97-102, 電子情報通信学会技術研究報告. IN, 情報ネットワーク. 109(411).
[32] Anton V. Riabov ほか. Wishful Search: Interactive Composition of Data Mashups. World Wide Web Conferences : World Wide Web Conferences, 2008. ページ: 775-784.
[33] Bin Lu ほか. sMash: Semantic-based Mashup Navigationfor Data API Network. World Wide Web Conferences : World Wide Web Conferences, 2009. ページ: 1133-1134.
[34] 片山太一ほか. スプログ検出における HTML 構造の類似性の有効性の評価.情報処理学会, 2009. ページ: 1-8, 情報処理学会研究報告. データベース・システム研究会報告. 2009-DBS-149(19).
[35] 池田 彰吾ほか. 繰り返し構造を考慮した Web ページの見出しの階層構造の解析.情報処理学会, 2008. ページ: 31-38, 情報処理学会研究報告. 情報学基礎研究会報告. 2008(34).
[36] 石井悠太ほか. ラッピング言語を用いた Web サイトの再構築手法の提案. 情報処理学会, 2009. ページ: 1-8, 情報処理学会研究報告. データベース・システム研究会報告. 2009-DBS-149(13).
[37] L. Liu ほか. XWRAP: An XML-enabled wrapper construction system for web information sources. 出版地不明 : International Conference on Data Engineering (ICDE), 2000. ページ: 611-621.
[38] GraggDuncan. The Basics | The Object Network. (オンライン) (引用日: 2012 年 02 月 20 日.) http://duncan-cragg.org/blog/.