UDDIとWSDL

高瀬 俊郎
日本アイ・ビー・エム（株） E30809@jp.ibm.com

UDDIとは

UDDI (Universal Description, Discovery and Integration) [1]は，Webサービスの動的な発見と統合を目指して，2000年9月にe-マーケットプレイスで有名なAriba，Microsoft，それにIBMが中心となって発足させたコンソーシアムである．発足時は36社がサポートを表明し，その中には日本からもNTTコミュニケーションズと富士通の2社が含まれていた．2001年5月現在で260社を超える企業がUDDIに参加するとしている．

ここで重要なのはIBM，Microsoft，Sun Microsystems，Hewlett-Packard (HP) などの本来ならば競合してもおかしくない大手ベンダがそろってこのコンソーシアムに参加していることである．標準化を行うことはその技術を普及させる上で重要であるが，さらに重要なことは標準規格の策定にそのコミュニティー全体が参加しサポートすることであろう．そういった意味でUDDIは順調なスタートをきったといえる．また現在公開されているUDDIの仕様はVersion 1であるが，今後Version 3まで拡張，発展されていくこととなっており，その後UDDIの仕様は国際的な標準化団体に委託される予定である．

UDDIはその名の通り，Webサービスの記述 (description)，発見 (discovery)，それに統合 (integration) が自由に行えるような枠組み作りを目指している．その第一段階として，グローバルなビジネスレジストリを構築しており，2000年11月16日からIBM，Microsoft，Aribaの3社によってレジストリの実装のβ版が公開された．そして2001年5月2日より正式版としてIBM，Microsoftのサイトで稼働している (Aribaによる正式版の実装は見送られ，代わりに今後Hewlett-PackardがUDDIレジストリのオペレータ［レジストリ実装の運営主体］となり，実装が行われることとなった)．

それではこのUDDIビジネスレジストリによってどのようなことが可能となるのであろうか．UDDIレジストリはインターネット上のサービスを分類し，公開するディレクトリサービスの機能を持っていることから，Yahoo!のようなWebページのディレクトリサイトによく例えられる．Webページのディレクトリはどのように使われるだろうか．たとえば，急な出張でホテルの予約をしたい場合，ホテルのページを検索し，場所や料金などの条件の合うところを探して，その場でWebページから予約を入れることができる．これはインターネットを用いたB2C (Business-to-Consumer，企業ー消費者間取引) のよい例であろう．

それではインターネットを介したB2B (Business-to-Business，企業間取引) の現状はどうだろうか．たとえば，ある企業が部品の調達を行う際にインターネットを介して注文書や請求書のやりとりを行うことを考える．手続きの自動化や合理化，また在庫管理なども容易になり双方の企業にとって大きなメリットがあるであろう．

しかし，実際にこのようなシステムを構築するには文書のデータフォーマットはどうするか，通信のプロトコルには何を使うのか，などあらかじめ双方の企業間でのすり合わせが必要である．また，もし元々社内で使っていた文書管理のシステムがあれば，そのシステムに統合するかたちでこのような企業間取引のシステムを作りたいかもしれない．このような場合，企業間でプラットフォームが違うと大変である．Windows上のVisualBasicで書かれたシステムとUNIX上のC言語で書かれたシステムの間で文書のやりとりを行うのは容易ではない．

さらに，何らかの理由で取引先を変更しなくてはならなくなったり，新しく現在の取引先では扱っていないような部品を調達する必要が出てきたりするかもしれない．この場合，まず何らかの形で取引先を探さなくてはならない．これには従来どおり足を使った営業活動を行うしかないであろう．そして，取引先が決まったとしてもその企業とは今までと同じシステムが使えるとは限らないのである．

UDDIではこれらの企業間取引における困難さを解消することを目指している．インターネットを介したサービスを持つ企業はそのサービスの内容と接続方法をUDDIレジストリに登録し，サービスを使いたい企業はUDDIレジストリを検索することで目的のサービスを探すことができる．さらに，条件の合ったサービスを見つけたなら，その接続方法を参照することによって，その日のうちにでもそのサービスを利用することができる．

また，B2CではWebページを利用するのは人間であり，ディレクトリを検索するのも人間が行うことであった．しかしB2Bのサービスではサービスを利用するのは機械であり，UDDIレジストリを検索するのも機械が行うことを想定している．そのため，サービスや接続方法の記述は機械が利用できるよう形式的に定められており，UDDIレジストリ自体のサービスも機械から使われることを考慮して，プラットフォームに依存しないWebサービスとして提供されている．

UDDIはこのような機械による動的な企業間取引を行うことのできる世界を導こうとしているのである．

UDDIの構成要素

UDDIが定めている仕様はWebサービスの動的な発見，統合を目指して作成されたものであるが，その中心になるのは，UDDIビジネスレジストリである．それらは主に，UDDIレジストリに格納されるデータ構造の定義と，それらのデータの検索などを行うためのUDDIプログラマーAPIからなる．それぞれUDDI Data Structure Reference[2]およびUDDI Programmer's API[3]によって定義されている．

■UDDIレジストリのデータ構造

UDDIレジストリのデータ構造ではWebサービスを検索して利用するときに必要となる情報をビジネス情報，サービス情報，バインド情報，そしてサービス型の4つのデータに分けて考え，それぞれをXMLの要素として定義している．これら4つのデータ間の関係を図-1に示す．これらの情報が構造化されてレジストリに登録され，検索に使われることになる．

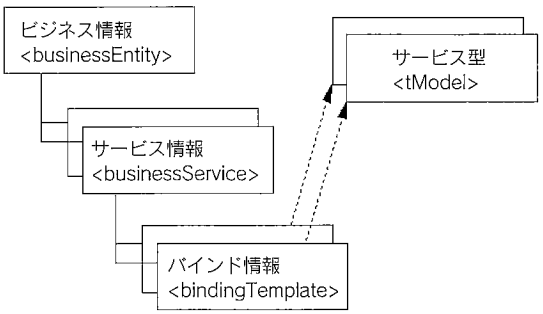

図-1　UDDIのデータ構造

[ビジネス情報 <businessEntity>]

このデータがUDDIレジストリの最上位の要素となる．<businessEntity>要素にはビジネスの主体，つまりサービスを提供する企業または団体の情報が格納される．1つの企業に対してこの<businessEntity>が1つあると考えればよい．<businessEntity>には，企業名，人が読んで理解できるようなその企業の簡単な説明，連絡先，URL，カテゴリ（業種），企業を特定するための何らかのIDなど，その企業とやりとりする場合に必要となる情報を記述することができる．多くの場合，1つの企業は複数のサービスを提供しているだろう．たとえばIBMはパソコンの販売もしているし，企業向けにシステムの構築などのサービスも行っている．そこで<businessEntity>には，複数のサービス情報<businessService>を持つことができるようになっている．

図-2に<businessEntity>の例を示す．ここでは，実際にUDDIレジストリに登録されていたものを一部省略して掲載している．

[サービス情報 <businessService>]

<businessService>要素には企業がどのようなサービスを提供しているかを記述する．これは必ずしもインターネットでアクセス可能なサービスでなくても構わず，電話やFAXによるサービスを登録してもよい．またそれぞれのサービスには複数のバインド情報<bindingTemplate>が含まれてもよい．つまり，たとえばパソコンを注文するのに電話で注文することもできるし，インターネットのWebページで注文することもできる，といった具合である．図-3に<businessService>の例を示す．

[バインド情報 <bindingTemplate>]

<bindingTemplate>要素にはバインド情報，つまりそ

のサービスに接続する際の技術的な情報が記述される．サービス型へのポインタや接続先のURLなどである．ここで気をつけておくべきなのはサービスの型はそれを指すポインタ（tModelKey）として記述されているということである．<bindingTemplate>には実際のサービスの接続先，つまり実装のありかが記述されるが，そのサービスが提供する機能，つまりサービス型は複数の実装によって共通であることが考えられる．よってサービスの型<tModel>は別のデータとして記述され，ここにはそのサービス型への参照が記述される．**図-4**に<bindingTemplate>の例を示す．

[サービス型 <tModel>]

<tModel>要素にはサービスの型が記述される．サービスの型とはどのようなプロトコルでどのような形式のデータを送るとどのようにレスポンスが返ってくるかといった種類の情報である．**図-5**に<tModel>の例を示す．

この<tModel>の例はHTTP GETすなわち一般的にはWebブラウザを介したサービスの型を示している．よってこのサービス型を持つサービスはWebブラウザを使える環境にある人ならばすぐに利用可能であることが分かる．

もう1つ例を挙げると，たとえばUDDIのディレクトリサービス自体もUDDIレジストリにすでに登録されているが，UDDIのサービス型はIBMによるものもMicrosoftによるものも同じ<tModel>を指している．IBMとMicrosoftではおそらく実装方法などは違うであろう．しかし同じ方法でアクセスできるという意味で同じサービス型を持つのである．

これによって，提供されているサービスと自分の知っているサービス型が同じ<tModel>を持っていればすぐにでも利用可能であるということが分かる．逆に，自分が呼び出し可能なサービスのリストを得るには，自分がサポートするtModelKeyを検索キーにしてサービスを探せばよい．

また，<tModel>にはWebサービス記述言語であるWSDL（Web Services Description Language）のドキュメントへのポインタを入れておくことも想定されている．WSDLはWebサービスのインタフェースを形式的に記述する言語であるので，WSDLを扱うツールを用いることで，WSDLの情報から自動的にそのWebサービスへアクセスするためのコードを生成することも可能になる．

```
<businessEntity
   authorizedName="0100000MDJ"
   businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
   operator="www.ibm.com/services/uddi">
  <discoveryURLs>
   <discoveryURL useType="businessEntity">http://...</discoveryURL>
  </discoveryURLs>
  <name>IBM Corporation</name>
  <description xml:lang="en">At IBM, we ....</description>
  <contacts>
   <contact useType="US general">
    <personName>IBM Corporation</personName>
    <phone>1 800 IBM 4YOU</phone>
    <email>askibm@vnet.ibm.com</email>
    <address>...</address>
   </contact>
  </contacts>
  <businessServices>
   (ここにいくつかのサービス情報 <businessService> が入る)
  </businessServices>
  <identifierBag>
   <keyedReference
    keyName="D-U-N-S" keyValue="00-136-8083"
    tModelKey="UUID:8609C81E-EE1F-4D5A-B202-3EB13AD01823"/>
  </identifierBag>
  <categoryBag>
   <keyedReference
    keyName="UNSPSC: Operating system enhancement software" keyValue="43162603"
    tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"/>
  </categoryBag>
</businessEntity>
```

図-2　<businessEntity>の例

```
<businessService
   businessKey="D2033110-3AAF-11D5-80DC-002035229C64"
   serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
  <name>Buy from IBM</name>
  <description xml:lang="en">This service enables direct purchasing....</description>
  <bindingTemplates>
   (ここにいくつかのバインド情報 <bindingTemplate> が入る)
  </bindingTemplates>
  <categoryBag>
   <keyedReference
    keyName="UNSPSC: Database software" keyValue="43161501"
    tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"/>
   ....
  </categoryBag>
</businessService>
```

図-3　<businessService>の例

```
<bindingTemplate
  bindingKey="6D8F8DF0-3AAF-11D5-80DC-002035229C64"
  serviceKey="894B5100-3AAF-11D5-80DC-002035229C64">
 <description xml:lang="en">Register to ShopIBM</description>
 <accessPoint URLType="https">https://....</accessPoint>
 <tModelInstanceDetails>
  <tModelInstanceInfo tModelKey="UUID:68DE9E80-AD09-469D-8A37-088422BFBC36"/>
 </tModelInstanceDetails>
</bindingTemplate>
```

図-4　<bindingTemplate>の例

```
<tModel authorizedName="0100000M99" operator="www.ibm.com/services/uddi"
  tModelKey="UUID:68DE9E80-AD09-469D-8A37-088422BFBC36">
 <name>uddi-org:http</name>
 <description xml:lang="en">An http or web browser based web service</description>
 <overviewDoc>
  <description xml:lang="en">This tModel is used to describe a web service
    that is invoked through a web browser and/or the http protocol.</description>
  <overviewURL>http://www.uddi.org/specification.html</overviewURL>
 </overviewDoc>
 <categoryBag>
  <keyedReference keyName="tModelType" keyValue="transport"
   tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"/>
 </categoryBag>
</tModel>
```

図-5　<tModel>の例

■UDDIレジストリによってできること

UDDIレジストリによってできることは大きく分けて2つある．1つはすでにUDDIレジストリ登録されている情報を検索することであり，もう1つは自分が提供するサービスをUDDIレジストリに登録することで広く一般に公開することである．

UDDIレジストリは現在IBMとMicrosoftによって別々にサービスが提供されているが，これらのUDDIレジストリは同期処理が行われ，どのレジストリの内容も同じになるようになっている．よって，サービスを検索する場合はどのレジストリで検索をかけても同じ結果を得ることができる．また，サービスを公開する場合もどちらのレジストリに登録してもすべてのUDDIレジストリに登録される．

UDDIレジストリへの検索や登録は人間がWebページからGUIを用いても行うことができるが，プログラムから自動的に検索，登録するための仕組みがあり，これは<find_business>などのタグを使ったXML文書を，SOAPで包み，そのSOAPメッセージをHTTPに乗せてUDDIレジストリに送るものである．このときのXMLの書式がUDDIプログラマーAPIによって定められている．APIといってもUDDIに投げるXMLの書式を定義しているわけだが，UDDIのインタフェースがSOAPになっていることで呼び出し側のOSや言語などのプラットフォームに依存せずにサービスを提供できるようになっている．

[Webサービスの検索]

UDDIビジネスレジストリはWebサービスを，(1) 名前，(2) サービスのカテゴリ（業種）で検索できるだけでなく，(3) サービスの型でも検索できるところに特徴がある．UDDIではこれらをそれぞれ電話帳にたとえてホワイトページ，イエローページ，グリーンページと呼んでいる．

ホワイトページではサービスを提供する企業の名前や何らかの企業のID情報によって検索をすることができる．

イエローページでは企業が提供するサービスを分類した情報（カテゴリ）によってサービスを検索することができる．これは，既存の（非電子的な）標準に基づく．Version 1では，次の3つの分類標準をサポートしている．

- NAICS (North American Industry Classification System)
- UNSPSC (Universal Standard Products and Services Classification)
- Geographical taxonomy (地理情報)

NAICS (http://www.naics.com/) は，カナダ政府，メキシコ政府，アメリカ政府の共同プロジェクトであり，UNSPSC (http://www.unspsc.org/) は，United Nations Development ProgramとDun & Bradstreetが，成果を1つの分類システムに併合して作成したものである．また，UDDIでは将来UDDI独自の分類を導入する予定である．

グリーンページでは提供されるサービスのサービス型による検索が可能である．サービス型とは前述のようにどのようなプロトコルでどのような形式のデータを送るとどのようにレスポンスが返ってくるかということである．このサービス型による検索が行えることでプログラムは自分が接続可能なサービスを見つけることが可能になる．

検索用のAPI (Inquiry API) には次のような2種類がある．1つは<find_xxx>でもう1つは<get_xxx>である．xxxの部分にはUDDIの4つの要素 (businessEntity, businessService, bindingTemplate, tModel) に対応して，business, service, binding, tModelといった文字列が入る．たとえば<find_business>という要素の中に

検索のキーの情報が入っているXMLをつくり，このXMLをSOAPで包んでHTTPでUDDIに送ればよい．図-6にUDDIレジストリへ送るメッセージの例を示す．

この場合，名前がIBMで始まる<businessEntity>を検索する．結果は<businessEntity>のUDDI内部でのキー（businessKey）のリストがXMLで返される．他の要素（businessService，bindingTemplate，tModel）を検索する場合も同様である．UDDIでは名前で検索する以外にもカテゴリやサービス型での検索が可能であった．その場合は<name>のところを<categoryBag>や<tModelBag>で置き換えることになる．このようにして各要素のキーが得られたら，そのキーを使って<get_xxx>を呼ぶことで各要素の実体を得ることができる．

[Webサービスの公開]

サービスの提供者はUDDIにサービスを公開することができる．公開用のAPI（Publish API）には次のような2種類がある．<save_xxx>と<delete_xxx>である．これらのタグにUDDIの要素を入れてUDDIレジストリに送ることによってUDDIに登録される．サービスの公開の際にはbusinessEntity，businessService，bindingTemplateはサービスを提供する企業が登録することになるが，tModelに関しては，一般に多くのサービスに共通に使われるサービス型の場合その仕様を策定した団体が登録することになる．ある企業の独自仕様の場合はそのサービスを提供する企業自身で登録する．また，他人に勝手に内容を変更されないよう，パスワードを入手する必要がある．公開用APIを用いる場合そのユーザIDとパスワードを用いて<get_authToken>メッセージを作成し，UDDIに送ることでトークンを取得して，そのトークンで登録者の認証を行う．

[Webサービスの接続]

興味のあるbusinessEntityと，それが提供するbusinessServiceが見つかったら，それのプログラミングインタフェースに相当する，bindingTemplateとtModelを手に入れる．bindingTemplateには，どのようなプロトコルでどのURLに接続すればよいかがXMLで記述されている．また，tModelにはサービスの型が記述されているので，これに基づいて，サービスへ接続する．この際には，UDDIを経由することはない．すなわち，いったん適切なサービスが見つかれば，そのサービスの情報を保存しておけばUDDIを毎回検索する必要はない．これは普通のWebページを検索し，その結果をブックマークしておくのと同じことである．

■UDDI Version 2

2001年6月18日にUDDI Version 2が発表された．Version 2ではOperator's SpecificationとReplication Specificationという文書が追加されているが，これらはUDDIレジストリそのものを運営する者のための仕様であり，それぞれUDDIレジストリのオペレータの要件，UDDIレジストリ間の同期に関する仕様が示されている．Version 2での変更点としては多言語対応や，検索条件の拡張，分類体系の検査の仕組みなどが挙げられるが，一番大きな変更は<publisherAssertion>というデータ構造の追加である．このデータはビジネス情報とビジネス情報の間の関係を表すものである．大企業の個々の子会社や，マーケットプレイスの個々の参加者はそれぞれ別々の<businessEntity>を持っていると考えられるが，これらの<businessEntity>は互いに関連しているであろう．このような企業間の関係を<publisherAssertion>を用いて表すことができる．もちろんこの<publisherAssertion>を扱うためのAPIも追加されている．また，Version 2では仕様書のいくつかの日本語版がUDDI公式ページ（http://www.uddi.org/）から入手可能である．

UDDI4Jを使ったサンプルプログラム

UDDIは機械どうしが対話するための仕組みであった．また，APIにSOAPのメッセージを用いることでプラットフォームに依存しないWebサービスとして提供されている．しかし実際このようなSOAPのメッセージ

```
POST /services/uddi/inquiryapi HTTP/1.0
Host: www-3.ibm.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <find_business generic="1.0" xmlns="urn:uddi-org:api">
   <name>IBM</name>
  </find_business>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図-6　UDDIレジストリへ送るメッセージ

を扱うことは少々面倒なようにも思える．このような負担を軽減するためにプログラム中から簡単にUDDIに接続できるようなツールが現れ始めている．ここではUDDI4J[4]を簡単に紹介しよう．

UDDI4JはIBMのdeveloperWorksのサイトにおけるOpen Source projectによってオープンソースとして公開されている．UDDI4JはUDDIレジストリと対話するクライアント用APIを提供する，Javaクラス・ライブラリである．SOAPの実装はApache-SOAPを前提にしている．**図-7**に簡単な使用例のコードの一部分を示す．

細かい説明は省略するが，UDDIProxyというクラスが中心となり，そのクラスのメソッドがUDDIのSOAPによるAPIの名前に対応する．これを実行することによって上記のSOAPメッセージとまったく同じ物がUDDIレジストリに投げられ，レスポンスの情報がBusinessListクラスに格納される．この例ではビジネスの検索を行っているが，もちろん公開をはじめUDDIレジストリのすべてのAPIを呼ぶことができる．

このようにJavaのコードの中から簡単にUDDIレジストリの機能を呼び出すことができる．ちょっとプログラムを書けば単にUDDIレジストリに対して検索や公開を行うだけでなく，動的にもっと複雑な操作をプログラム中から行うことが簡単にできるだろう．たとえばカテゴリごとにいくつのサービスが登録されているかカウントし，ファイルに書き出すなどということも可能であろう．人間がWebページを見ながら手作業で行ったのでは気の遠くなる作業である．また，見つけてきたサービスの型が知っているものか否かを判別し，知っているものならば，そのまま自動的にそのサービスに接続するようなプログラムを組むことが可能となる．

UDDI4JのようなUDDIをサポートするツールが多くのプラットフォームやプログラミング言語に対してサポートされることで，UDDIの重要性はますます大きくなっていくであろう．

```
UDDIProxy proxy = new UDDIProxy();
proxy.setInquiryURL("http://www-3.ibm.com/services/uddi/inquiryapi");
BusinessList bl = proxy.find_business("IBM", null, 0);
```

図-7　UDDI4Jのコード

サービス型記述言語WSDL

WSDL[5]はそれまでそれぞれ独自に開発が進められていたIBMのNASSL (Network Accessible Services Specification Language) とMicrosoftのSCL (SOAP Contract Language) を1つに統合したものである．

2000年9月，UDDIの発表の数週間後にWSDL1.0がIBMとMicrosoftによってリリースされ，2001年3月15日にはWSDL1.1がW3CへNoteとしてIBMとMicrosoftから共同提案されている．

■WSDLの概要

WSDLはWebサービスのインタフェースや接続のための情報を記述する言語である．CORBAにおけるインタフェース定義言語 (IDL) に相当し，UDDIにとってはサービス型である．どのプロトコルで，どのようなデータを受け取り，どのようなデータを返すのかを記述する．実際にはWSDLはいくつかの定義の集合として記述される．<definitions>要素の中に<types>，<message>などの定義が入る．**図-8**にWSDLの例を示す．

サービスは，5つの主要な要素を使用して定義される．

<types>は交換されるメッセージに使用されるデータ型の定義である．この中にはデータのXML Schemaを入れることができる．この例では"TradePriceRequest"と"TradePrice"の2つのデータ型を定義している．

<message>には送られるデータの抽象定義を表す．この要素は，いくつかの<part>要素で構成され，それぞれの<part>は何らかのデータ型を示す．これはメソッドの引数の組や戻り値を表すと考えればよいであろう．この例では"GetLastTradePriceInput"は"TradePriceRequest"のデータ型を持ち，"GetLastTradePriceOutput"は"TradePrice"のデータ型を持つことになる．

<portType>は抽象化された操作の定義を記述する．この要素には<operation>要素が複数含まれ，これらがそれぞれの操作を表す．<operation>では入力messageと出力messageを指定する．Javaでいうところのインタフェースが<portType>，メソッドシグネチャが<operation>である．この例では"StockQuotePortType"というインタフェースがあり，"GetLastTradePrice"メソッドの引数が"GetLastTradePriceInput"，戻り値が"GetLastTradePriceOutput"ということになる．

<binding>は<portType>の実装の情報である．<portType>によって定義された操作の具体的なプロトコルとデータ形式仕様を指定する．"StockQuoteSoap-

Binding"は"StockQuotePortType"を実装したものであり，SOAPのHTTPで接続する．

<service>は<port>の集合として記述され，<port>にはそのサービスの実装のありか，つまりアクセスポイントのアドレスが指定される．"StockQuoteService"には，"StockQuoteSoapBinding"の実装があり，そのアクセスポイントは"http://example.com/stockquote"である．

このようにWSDLはいくつかの定義の集合として記述されるが，これらの5つの主要な要素はそれぞれネストすることなく独立に記述され，お互いは名前によって参照されているだけである．このことによって，ある1つの定義は複数の定義から参照されることが可能である．また，以前書いた定義や他の人によって記述された定義の再利用ができる．さらに，この定義の再利用を容易にするために<import>というタグを用いることで，他のWSDLドキュメントの定義を取り込むことができるようになっている．

```
<?xml version="1.0"?>
<definitions name="StockQuote"
   targetNamespace="http://example.com/stockquote.wsdl"
   xmlns:tns="http://example.com/stockquote.wsdl"
   xmlns:xsd1="http://example.com/stockquote.xsd"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
     <schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="TradePriceRequest">
          <complexType>
             <all>
                <element name="tickerSymbol" type="string"/>
             </all>
          </complexType>
        </element>
        <element name="TradePrice">
          <complexType>
             <all>
                <element name="price" type="float"/>
             </all>
          </complexType>
        </element>
     </schema>
   </types>
   <message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
   </message>
   <message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
   </message>
   <portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
      </operation>
   </portType>
   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetLastTradePrice">
        <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
      </operation>
   </binding>
   <service name="StockQuoteService">
      <documentation>My first service</documentation>
      <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote"/>
      </port>
   </service>
</definitions>
```

図-8 WSDLの例

■WSDLの目指すもの

WSDLはサービスを記述する言語であるが，何のためにWSDLドキュメントを記述しなければならないのであろうか．それは，Webサービスの動的な統合という大きな目標に関係している．ある公開されているWebサービスに対してWSDLが記述され，誰にでも手に入る状態になっていれば，そのサービスを使いたいと思っている人にとっては単純にそのサービスのインタフェースや接続法が分かるという利点がある．WSDLはXMLで記述されているため自己記述的であるので人間が読んでも大体の意味は理解できる．しかしもっと有益なことはWSDLが形式的に記述されているためにWSDLを扱うツールを用いることによって，そのサービスに接続するためのクライアントコード（スタブ）を自動的に生成することが可能になることである．

このようなツールを使うことで次のようなシナリオが考えられる．ある人が使いたいサービスをUDDIから

見つけてきたとしよう．しかし，そのサービス型は自分のよく知らないものであったとする．それでも，もし<tModel>にWSDLが登録されていれば，そのWSDLを入手しツールにかけることで，自動的にそのサービスに接続するための環境を作ることができる．UDDIは元々実行時のサービスの発見と統合を目指したものであるが，WSDLによって，たとえインタフェースを知らないサービスに対しても実行時に自動的に接続することが可能になる．

このようにWSDLを公開しておくことによってサービスを使う側の接続するための負担は軽減する．サービスを提供する側としてもWSDLを公開しておくことで多くの人に使ってもらうことができる．また，WSDLを扱うツールによって，サービスのコンポーネントから自動的にWSDLを生成することも可能である．これによってサービスを提供する側としても，サービスを構築した時点でWSDLを自動的に生成することが可能であり，それをそのままUDDIで公開しておけばよい．

たとえば，IBMのalphaWorksからリリースされているWeb Services Toolkit[6]にはは WSDLからApache-SOAP用のJavaのクライアント・スタブを自動的に生成するツールや，公開するサービスのJavaクラスやEJBのJarファイルからWSDLドキュメントを生成するツールが含まれている．

Webサービス・アーキテクチャ

UDDIは図-9に示すようなWebサービス・アーキテクチャのモデルを背景としている．Webサービス・アーキテクチャはサービス・リクエスタ，サービス・プロバイダ，サービス・ブローカの3つの役割からなる．サービス・リクエスタはサービスの要求者，サービス・プロバイダはサービスの提供者である．固定された企業間のサービスの接続ならばこの2者間の関係だけでよいことになる．しかし，Webサービス・アーキテクチャではサービスの実行時の動的な結合を目指している．この場合，サービスを実行時に発見し，接続するための情報を提供する仕組みが必要となる．この役割を果たすのがサービス・ブローカである．UDDIはそれ自体がサービス・ブローカであると考えることもできるし，UDDIをバックグラウンドに持っている特定の業界のe-マーケットプレイスやポータルがサービス・ブローカの役割を果たすことも考えられるだろう．

Webサービス・アーキテクチャにおいては次のようなシナリオが考えられている．サービス・プロバイダは，自分自身の機能を，サービス・ブローカに対して公開しておく．そこで，あるプログラム（サービス・リクエスタ）がある機能のサービスを必要としたとすると，サービス・リクエスタはサービス・ブローカに対して，その機能を持つサービスが存在するか検索する．サービス・ブローカはそのようなサービス・プロバイダがあればそれのインタフェース情報を返す．この情報に基づいて，リクエスタはプロバイダに接続し，サービスを受ける．

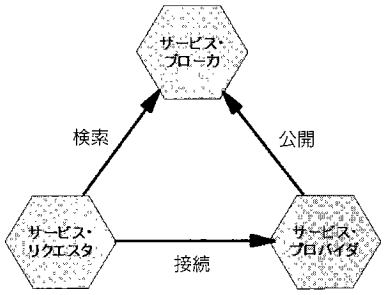

図-9　Webサービス・アーキテクチャ

もちろん，このような動的な統合をうまく行うためにはまだまだ解決しなければならないことはたくさんある．たとえば，サービス型であるtModelを検索キーにすれば，プログラム的にコンパチブルなWebサービスは正しく見つかるが，そのサービスの意味的な内容については，サービスのカテゴリと自然言語に基づく記述に頼らざるを得ない．また，UDDIに登録されたWebサービスが本当に主張するとおりのサービスを提供できるのか，あるいはそのサービス品質についてはどの程度のものを保証するのか，なども問題となるであろうし，リクエスタとプロバイダとの間の契約の締結はどのようにするかなどについてもこれからの課題である．

参考文献
1) Universal Description, Discovery and Integration（UDDI）, http://www.uddi.org/
2) UDDI Data Structure Reference V1.0, http://www.uddi.org/pubs/DataStructure-V1.00-Open-20000930_2.pdf
3) UDDI Programmer's API 1.0, http://www.uddi.org/pubs/ProgrammersAPI-V1.01-Open-20010327_2.pdf
4) UDDI4J Project, http://oss.software.ibm.com/developerworks/projects/uddi4j
5) Web Services Description Language（WSDL）1.1, http://www.w3.org/TR/wsdl, 日本語訳, http://www.microsoft.com/JAPAN/developer/workshop/xml/general/wsdl.asp
6) IBM, Web Services Toolkit, http://www.alphaworks.ibm.com/tech/webservicestoolkit

（平成13年6月25日受付）