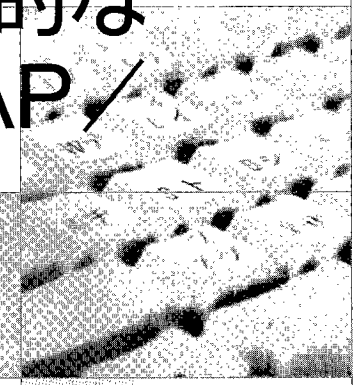
2 Web Servicesによる動的な電子商取引の実現－SOAP／WSDL／UDDI－



日本アイ・ビー・エム（株）東京基礎研究所
丸山　宏[*1]　小坂一也[*2]
日本アイ・ビー・エム（株）東京基礎研究所・国立情報学研究所
浦本直彦[*3]

[*1] E-mail:maruyama@jp.ibm.com　[*2] E-mail:kosaka@jp.ibm.com　[*3] E-mail:uramoto@jp.ibm.com

はじめに

インターネット，特にWeb技術の普及は，アプリケーション開発に大きな変革をもたらした．e-Businessという造語は「インターネットの利用によるビジネス革新」というような意味で用いられる．e-Businessという概念が提唱された初期の段階では，主に従来企業内部の特定アプリケーションだけで使われてきた企業の基幹システムを，Webを通してインターネット上から利用できるようにすることによって，既存の情報資産に新たな価値を与えるものとして捉えられてきた．この概念は既存の情報資産とインターネット上の人間ユーザを結びつけるという意味で，Business-to-Consumer（B2C）と呼ばれる．B2Cを可能にした主要なテクノロジは，TCP/IP，HTTP/HTMLという通信プロトコルの標準化と，3層モデルに基づく分散プログラミングであった．

現在，多くの企業は第2段階のe-Businessとして，企業間のシステム連携に乗り出している．企業間のデータ連携の例としてサプライチェーンがある．具体的には，企業が必要な物品・サービスを他企業から購入する場合，あるいは他企業に納入する場合，従来，電話やFAX，紙の伝票で行われてきた注文書／納品書／請求書／領収書などの処理を電子的に行うものである．電子的に行うことでジャスト・イン・タイム方式の調達が可能となり，在庫を減らし効率的な企業経営が可能となる．この段階のe-Businessは既存の企業間取引をインターネット技術で置き換えるもので，Business-to-Business（B2B）という言葉で捉えられる．

B2Bを加速した技術的な最大の要因は，データ表現の世界標準としてのXML[1]であった．多くの企業がインターネットによるB2Bに注目した結果，XMLやその周辺技術が急速に発展し，さまざまな技術や標準規格が提案されることとなった．特に，ebXML[2]や，SOAP（Simple Object Access Protocol）[3] UDDI（Universal Description, Discovery, and Integration）[4]に代表される，企業間連携をより動的にする技術は現在多くの注目を集めている．同時に，これらの技術は新たなソフトウェアコンポーネント技術や異なる環境にあるコンポーネントを結合する分散プログラミングとして捉え直すこともできる．

本稿では，まずXMLに基づく企業間取引において必要性が認識されつつあるメッセージレイヤと，それを実現するSOAPについて簡単に説明する．次にXML/SOAPによって可能になるソフトウェア部品としてのWebサービスについて述べる．最後に，より動的な企業間連携を可能にするグローバルディレクトリとしてのUDDIについて解説する．

XMLによる企業間取引（B2B）

■EDIからXMLへ

企業間商取引における代表的なデータ交換方式は今まで，EDI（Electronic Data Interchange）であった．EDIは，企業間のビジネスオートメーションを進めるために，1970年代から金融，製造などの分野で広く使われてきた．EDIは長い実績があり，企業間取引の電子化に関して大変功績があったのは事実だが，インターネットの時代になり，ほぼすべての企業がネットワークに接続された現在でもその恩恵を享受している企業の数はそれほど多くない．その理

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Header>
  <t:Transaction xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
0005721
</t:Transaction>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
  <m:GetLastTradePrice xmlns:m="Some-URI">
    <m:symbol>DIS</m:symbol>
  </m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図-1　SOAPメッセージの例

由の1つはEDIシステムの構築コストが高いことである．これに対して，XML[1]は，データ表現が自己記述的であり柔軟にデータ構造が記述できるため変化するシステムやビジネス環境に，低コストで対応することが容易である．また，標準化団体による厳密な定義がなされており，マイクロソフトやIBMなど主要なベンダの製品群が相互運用可能であることも大きな利点の1つで，特にB2Bの分野で標準のデータフォーマット言語として地位を確立した．

現在多くの企業が，XMLとHTTPに基づいたB2Bシステムの構築を急いでいる．しかし，XML/HTTPを用いた企業間のアプリケーション統合（application integration）は，単なるEDIの置き換えではなく，実は大きな分散プログラミングであることが認識されるにつれ，XML/HTTPだけでは分散プログラミングのベースとしては十分でないことも分かってきた．その1つとして挙げられるのが，現在のXMLによるB2Bベースのシステムアーキテクチャにおけるメッセージレイヤの重要性である．

多くのシステムでは，XML/HTTPに基づくデータ交換は1つのXML文書を単位として行われる．また，1つのXML文書の送信をメッセージの送信として捉えてもよい．メッセージの内容の構文は，xxML（xxにはさまざまな文字，単語が入る）のように業界ごとに決められたマークアップ言語あるいは，企業間で取り決めたマークアップによって決定される．このメッセージをHTTPに載せて送信することはよいとしても，これだけで複雑な電子商取引を行う企業間の通信ができるだろうか？　実際に企業間のデータ交換を考えるといくつか問題があり，HTTPに代表されるトランスポートとXMLで記述されるコンテンツの間に，データのやりとり方式を規定するメッセージレイヤが必要であることが認識されつつある．これらの要件をふまえ，XMLによるメッセージ交換のための仕様であるSOAP[2]が注目を集めている．

■ Simple Object Access Protocol (SOAP) 1.1

XMLに基づくトランスポート独立なプロトコルおよびメッセージ・エンベロープのための仕様として現在注目されているのが，2000年5月にマイクロソフト，ロータス，IBMなど5社が中心となってW3Cに提案した，Simple Object Access Protocol（SOAP）1.1である．SOAP 1.1はXMLの構文を用いたきわめて単純なメッセージ・エンベロープ構造と，同じくXMLの構文を使った汎用のデータ直列化を定義した仕様である．エンベロープの名前が示すとおり，SOAPは，メッセージ本体をくるむ「封筒」の役目を果たす．手紙を便箋のまま送るより封筒に入れて送る方が何かと便利である（中身が見られない，宛先が書けるなど）のと同様に，メッセージ・エンベロープを用いることで，XMLメッセージのやりとりをより柔軟かつ効率よく行うことができる．

SOAP 1.1のメッセージエンベロープをHTTPトランスポートの上に載せた例を図-1に示す．SOAP1.1のエンベロープはHeaderとBodyという要素からなり，Bodyの中身がこのメッセージの保管場所（ペイロード）となる．Headerは XMLの特徴を活かして拡張可能になっており，この例ではトランザクションIDをヘッダに記述する例を示している．

この例で分かるように，SOAPのメッセージ・エンベロープ（Envelope）はきわめて単純であり，HeaderとBodyという2つの構成要素しか定義していない．これに基づき必要な拡張を加えて実際のプロトコルを構成することを想定している．このようにSOAPは，必要最小限でかつ合意が得られるものからインクリメンタルに標準策定を行っていくアプローチをとっており，プロトコルからコンテンツまでの一式を同時に策定しようとするebXML[7]のアプローチとは対照的である．なおSOAP 1.1は2000年5月にW3Cに提出された技術ノート（Note）であり，これを受けてW3CはXML Protocol Working Groupを同年9月に発足させた．SOAPやebXMLの議論に基づきXML上のメッセージングの標準がここで決まることが期待されている．すでにいくつかの実装が利用可能である．たとえば，オープンソース団体のApacheではSOAPの実装を公開している[5]．

ソフトウェア部品としてのWebサービス

SOAPやXMLプロトコルの議論は，B2Bから始まったインターネット上でのアプリケーション統合の動きを分散プログラミングの枠組みとして捉え直したもの，と見ることもできる．ここでは，その動きの1つとしてWebサービス（Web Services）と呼ばれる考え方を紹介する．

■ソフトウェア部品の疎結合

ソフトウェアの部品化とその再利用は，ソフトウェアの生産性向上のための1つの柱として研究・開発されてきたし，これからもそうであろう．古くはサブルーチン

から始まり，オブジェクト指向やクラスライブラリ，あるいはJava Beansに代表されるようなコンポーネント・モデルは，いずれもソフトウェアを部品化し再利用する試みである．

ソフトウェア部品を再利用する際に重要になるのが，「その部品同士がどのくらい深く依存関係を持っているか」ということである．あるいは，「2つの部品AとBを組み合わせる場合に必要とされる前提条件はどれほどあるか」と読み替えてもよい．あるソフトウェア部品が，それと組み合わせられる部品に対して特定の環境，たとえば特定のオペレーティングシステムと特定のプログラミング言語の組を要求する場合，それらの部品は密に結合される（tightly coupled）という．典型的な場合はC言語などの手続き呼び出しのライブラリである．部品の密結合においては，多くの場合プログラミング言語のレベルで細かいデータの受け渡しが効率よく可能であり，したがって実行効率の高いアプリケーションを構築することができる．

一方，複数のホストで動く既存のアプリケーションをソフトウェア部品とみなして統合する場合，特に企業間をまたがって統合するB2Bのような場合には，特定のオペレーティングシステムやプログラミング言語を仮定することはできない．したがって，より少ない前提条件で結合，すなわちより疎に結合する（loosely coupled）ことが望まれる．MQ（Message Queue）やCORBAなどのアプリケーション統合ミドルウェアはこのような疎結合を可能にするための道具立てと考えることもできる．これらのミドルウェアは多種の環境に対応しているので，それぞれのホストのオペレーティングシステムやプログラミング言語に対応したミドルウェアを導入しておけば，統合にかかる前提条件を大幅に緩和することができる．このミドルウェアのAPIが他方の環境を完全に隠してくれるからである．

この疎結合の考え方をさらに進めると，プログラミング言語／APIレベルでの整合性をとるのではなく，部品間で実際に流れるデータの形式のみを取り決め，その処理の仕方はそれぞれの自由にしようという考え方が可能になる．すなわち，部品の双方がアプリケーション統合ミドルウェアに依存することなく交信できることを目指す．このためには，データ交換形式がきわめてシンプルで，かつ誤解のない分かりやすい形式でなければならない．

HTTP，XML，SOAPといった標準によって，このような特定のミドルウェアに依存しないデータ交換が可能になった．あるソフトウェア部品が，あるスキーマに基づいたXMLメッセージをHTTP/SOAPを使って，あるURL上で受け付ける，という仕様を公開したとしよう．このソフトウェア部品を使うためには，高価なアプリケーション統合ミドルウェアの特定のバージョンを購入しそのAPIを呼ぶ必要はない．XMLパーサなどの公開ライブラリを使用する必要さえないかもしれない．たとえば，上記のSOAPメッセージのようなきわめて簡単なメッセージならば，telnetセッション上で直接端末のキーボードをたたいてもこの部品を呼ぶことができる．

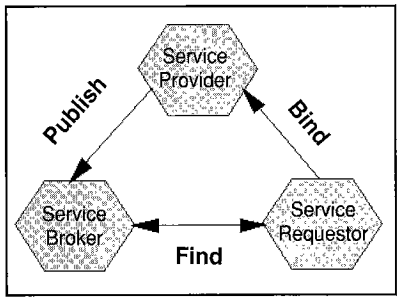


図-2　Webサービス・アーキテクチャ

疎結合という概念はまた，部品の統合の決定がいつなされるか，という観点で考えることもできる．密結合のシステムでは，「どの部品とどの部品を結合するか」という決定は，システムの設計時，あるいはコンパイル時に行わなければならない．OCXやJava Beansのようなコンポーネントモデルの場合でも，システムのインストール時には結合され得る部品が判明していなければならない．このように，密結合システムにおいては，結合は早期に起きている（early binding）といえる．一方，疎結合システムの場合は，結合の決定は実行時にまで延期することができる．部品結合の前提条件が非常に緩いので，システムがある機能を果たす部品を必要としたときに初めて，そのような部品を探しにいき，発見し，それを動的に結合するというようなことが現実味を帯びてくるのである．また，XMLでインタフェースをとることによって，異なるコンポーネント実装の相互運用といった実装上の問題も解決することができる．このように，XML/HTTP/SOAPなどの標準は，緩い，そしてきわめて遅い結合（late binding）を可能にすると考えられている．

■Webサービスの世界

このような，疎結合／遅結合の仕組みを活かしたソフトウェア構築のモデルはどのようなものであろうか？その1つの答えは「Webサービス・アーキテクチャ」という概念に見ることができる．Webサービスは，インターネット上でアクセス可能なWebアプリケーションをソフトウェア部品とみなしたものである．Webサービスは多くの場合，HTTP上である仕様（DTD）に基づいたXML文書で要求を受け取り，応答もXML文書で返す．

IBMがそのWeb Services Toolkit[6]の中で提唱しているWebサービス・アーキテクチャは図-2に示すものである．Webサービス・アーキテクチャにおいては，Webサービスであるソフトウェア部品はサービス・プロバイダと呼ばれる．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（publish）しておく．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を問い合わせる（find）．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，すなわち入出力のデータ型（たとえばDTD），ト

ランスポート・プロトコル（たとえばHTTP/SOAP），およびアクセスポイント（たとえばURL）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（bind）しサービスを受ける．

動的なサービスの発見には，ここでいうブローカ，あるいはディレクトリが必要であり，また，サービスのインタフェースを記述するための言語が必要である．これらの標準として期待されているのが次に述べるUDDIである．

Universal Description, Discovery, and Integration (UDDI)

UDDIはWebサービスとそれによる動的な統合を目指して，2000年9月にAriba，マイクロソフト，それにIBMが中心となって発足させたコンソーシアム（http://www.uddi.org/）である．発足時は日本からもNTTデータと富士通を含む36社がサポートを表明していたが，12月1日現在，100社を超える企業がUDDIに参加している．

■UDDIとは何か？

UDDIはその名の通り，サービスの記述（description），発見（discovery），それに統合（integration）が自由に行える枠組みである．その第1段階として公開ビジネスディレクトリを構築し2000年11月16日からサービスを提供している．このビジネスディレクトリはWebサービスをビジネス名，ビジネスの業種（カテゴリ）で検索できるだけでなく，サービスの型でも検索できるところに特徴がある．

あるアプリケーションが，ある機能が必要になったのでUDDIディレクトリから適当なWebサービスを検索するとしよう．その機能はあるWebサービスではAというDTDで通信することを要求するかもしれず，また別のWebサービスではBというDTDで通信することを要求するかもしれない．しかし，リクエスタ側のアプリケーションがDTD Aのみをサポートしていたとしたら，DTD Bを使うWebアプリケーションに接続することができない．したがって，リクエスタは，「DTD Aをサポートし，かつこの機能をサービスするWebサービスは何か？」という質問をUDDIディレクトリに投げればよいわけである．

たとえば，あるとき思い立って脱サラし，自分の町で小さな靴屋を始めようとしたとしよう．仕入れや在庫管理や決算処理のために靴屋用のソフトウェアパッケージを買ってきたとする．このパッケージは受注や発注のための靴業界特有のDTDがあらかじめ組み込まれている．このパッケージソフトを自分のコンピュータにインストールすると，このソフトはまずインターネット上のUDDIディレクトリに接続し自分が知っているDTD（すなわち靴業界DTD）をサポートするWebサービスを探してくる．そうすると，スペインの靴メーカから発注し韓国の顧客に売る，などということがその日からできるようになるわけである．

■UDDIの構成要素

- サービスの型とtModel

サービスの型はアプリケーションAがサービスBを呼び出すことができるために必要な概念である．オブジェクト指向言語においてメソッドの型が合わないとオブジェクトの呼出しができないように，Webサービスにおいても型が合わないと呼出しができない，あるいは呼出しに意味がなくなってしまう．しかしながら，サービスの型を一般的に記述するのは非常に難しい．XMLでデータをやりとりするサービスにおいては，サービスの型は基本的には入出力のDTD（スキーマ）で指定できるかもしれないが，しかし，同じDTDをサポートするからといって常に呼出し可能とは限らない（特定のフィールドに特定の値を要求するかもしれないし，呼出し／応答の順序が異なるかもしれない）．

現在のところサービス型を万能に記述する方法はないので，UDDIでは登録されたサービス型をtModelと呼び，tModelにつけられたグローバルにユニークな識別子（tModelKey）によってサービス型を識別する．サービス型が互換であるかどうかは，このtModelKeyが一致するかどうかだけで判断する．tModelKeyの実体はネットワークデバイスIDや時刻から生成される16バイトのUUIDである．サービス型はこのtModelKeyとともにUDDIディレクトリに登録される．登録の際にサービス型，すなわちこのサービスを呼び出すための規約を詳細に記述した仕様書へのリンクを含める．たとえば業界ごとに定められたXML文書の仕様と，呼出しのシーケンスなどを記述した仕様書がこれにあたる．あるいは，「HTTPのGETメソッドで呼出し，HTMLが返される」というtModelを登録してもよい．これは通常のWebサイトのサービス型にあたる．

- Webサービスの記述

WebサービスをUDDIに登録するには，まずそのサービスを記述しなければならない．UDDIでは，以下の3種類の情報を記述する必要がある．

1. ビジネスエンティティの記述（businessEntity）
2. サービスの記述（businessService）
3. 接続情報の記述（bindingTemplate）

businessEntityはビジネスを行う主体，たとえば上述の靴屋の例ではその会社である．businessEntityには，名前，カテゴリ（業種），コンタクト先などの情報が付加される．ビジネスカテゴリの分類体系としては，最初は米国政府が決めた業界コード，あるいは国連が決めた業界コードなどの体系が使われる（UDDIは将来異なる業種カテゴリ体系を導入する意図を表明している）．

1つのbusinessEntityは，複数のbusinessServiceを持つことができる．たとえば「Goodness Shoes, Inc.」は，男性用，女性用の靴販売，メーカからの仕入れ，カスタマーサービス，など複数のbusinessService を持つことが考えられる．businessServiceは必ずしもインターネット上でアクセスされるものでなくてもかまわない．たとえば，電話やFaxに

よるサービスの窓口を登録してもよい.

あるbusinessServiceにプログラム的に接続するためには, そのbusinessServiceがサポートするサービス型と接続先のURLなどの技術的な情報が必要である. これらの情報はbindingTemplateとして記述される.

• Webサービスの登録

UDDIグローバルディレクトリは, 当面マイクロソフト, Ariba, IBMの3社が個別に運営する. これら3つのディレクトリは内容が完全に同じになるように同期される. 上記のbusinessEntity, businessService, bindingTemplateによって記述されたWebサービスは3社のディレクトリのどれかに登録するが, 同期処理によって同じ内容は他の2つのディレクトリからも得ることができる. 詳細については参考文献5) を参照してほしい.

• Webサービスの検索

検索には, HTML経由のものと, プログラムからSOAP経由で呼ぶものとがある. この場合, <find_business>のようなタグをSOAPメッセージの中に入れて検索機能を呼び出す.

• Webサービスへの接続

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を手に入れる. bindingTemplateには, そのWebサービスがどのような入出力を要求しているか, また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で記述されているので, これに基づいてWeb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. いったん適切なWebサービスが見つかれば, そのWebサービスの情報をキャッシュしておけば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.

• Webサービス記述言語WSDL

発見されたWebサービスのtModelが自分の知っているtModelと一致しなかったとしても, そのインタフェース仕様が形式的に記述されていれば自分の持つインタフェースを適合させることができるかもしれない. そうでなくても, サービス型がプログラム開発ツールに理解できる形になっていれば, そのインタフェース仕様に合わせたプログラムの開発が楽になる. これは, CORBAなどの分散オブジェクト指向プログラミングにおけるIDLの考え方と同様である. WebサービスにおけるIDLに相当するものとして, Web Service Description Language (WSDL) [7] が提案されている. WSDLでは, サービスの入出力のデータ型をXML Schemaで指定したりすることができる. また, IDLの場合と同様, WSDLに基づいてスタブコードを生成するツールも提供されている.

■動的なアプリケーションの構築

UDDIを使うと, 他のWebサービスを実行時に動的に発見し, 接続することができる. これは何を意味するだろうか? まず, プログラム開発時には, 「どのソフトウェア部品 (Webサービス) を使うか」を指定する必要がない. これは, 前述した「きわめて遅い結合」を実現する. この結合を, アプリケーションの統合あるいは構築と考えれば, アプリケーション統合におけるカンバン方式 (Just-in-Time Integration) と呼べるのではないか. たとえば, カリブ海のクルーズに行きたかったとしよう. このためには, 航空会社のWebサービス, フロリダのレンタカーのWebサービス, ホテルのWebサービス, そしてもちろん, クルーズ会社のWebサービスを発見し, 統合し, それらをアクセスしながら, 最適な組合せを見つける必要があるだろう. UDDIとWebサービスに基づくJust-in-Time Integrationはこのようなシナリオを可能にすると考えられている. このために, UDDIでは将来のワーク・アイテムとして, Webサービスを動的に統合するための枠組みを挙げている. 最初の一歩としては, Webサービスを発見し, 呼び出すための仕組みを持ったスクリプト言語のような形態になるだろう. 将来的には, アプリケーションの仕様を入力すると, 自動的にWebサービスの発見と組合せを行い, 新たなWebサービスとして登録するようなことが可能になる.

もちろん, このような動的な統合をうまくやる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, などについてはこれからの課題である.

おわりに

本稿ではXML技術をベースとした動的なB2Bのための基本的な枠組みと, それらを実現するためのいくつかのXML規格 (SOAP, UDDI, WSDL) について解説した. 誌面の都合であまり具体例を挙げられなかったが, それらの詳細については参考文献を参照していただきたい.

参考文献

1) Maruyama, H., Tamura, K. and Uramoto, N.: XML and Java: Developing Web Applications, Addison Wesley (日本語訳「XMLとJavaによるWebアプリケーション開発」, ピアソンエジュケーション) (May 1999).
2) ebXML, http://www.ebxml.org
3) SOAP 1.1 Specification, http://www.w3.org/TR/SOAP/, 日本語の仕様書は, http://www.ibm.com/jp/developerworks/link/soap.html (May 2000).
4) UDDI Programmer's API Specification.
5) Apache SOAP, http://xml.apache.org/soap/
6) IBM, Web Services Toolkit, http://www.alphaworks.ibm.com/tech/webservicestoolkit, http://www.uddi.org/pubs/UDDI_Programmers_API_Specification.pdf
7) Web Services Description Language(WSDL) 1.1, http://www-4.ibm.com/software/developer/library/w-wsdl.html

(平成13年5月31日受付)