自動化・統合化されたWebサービス動的実行システム

越田高志[†,††] 波多野賢治[††] 天笠俊之[††]
宮崎純[††] 植村俊亮[††]

ユーザがWebサービスを利用する際には，そのWebサービスを実行するためのクライアントプログラム（スタブ）を開発する必要がある．そして，その開発にはWebサービスに関する高度な専門知識と利用技術が必要であり，Webサービスの容易な利用が難しい状況にある．Webサービスが広く実用化され普及するためには，ユーザが必要とする最適なWebサービスを容易に検出し，動的に実行できる仕組みが必要である．本論文では，そのWebサービスの動的実行を容易に実現するシステムについて述べる．WSIFによりユーザ側のスタブ開発の負担は軽減したが，依然として利用するWebサービスごとに人手による難解かつ多様なプログラミング作業が必要である．特にWebサービスの出力データが複合型の場合，クライアント側におけるそのデータオブジェクトの動的自動生成が難しく，任意のWebサービスに対してそれらを自動化した動的実行システムはいまだ実現されていない．我々は，UDDIレジストリを基にそこに登録されているWebサービスを出力データ型に依存することなく，動的に実行する自動化および統合化システムを開発し，これらの問題を解決した．

An Automated and Unified Dynamic Invocation System for Web Services

Takashi Koshida,[†,††] Kenji Hatano,[††] Toshiyuki Amagasa,[††]
Jun Miyazaki[††] and Shunsuke Uemura[††]

A user has to develop a client program (stub) for invoking a Web Service. Advanced special knowledge and technology are required for the stub development, and it is difficult for user to use the Web Service easily. To expand in the use of Web Service, we have to construct the architecture that detects easily the optimal Web Service a user needs, and that executes it dynamically. This paper focuses on the realization of easy dynamic invocation for Web Service. The burden of stub development by the user was mitigated by WSIF. Still however, for every Web Service, various difficult programming works is required. For a complex output data type, it is difficult to generate dynamically the data object in the client, and a dynamic invocation system that automates them to arbitrary Web Services is not yet proposed. We developed an automated, unified system that can execute dynamically Web Service registered in the UDDI registry, and that is not dependent on the output data type. Then we solved these problems.

1. はじめに

Webサービスとは，データアクセスプロトコルとしてSOAP（Simple Object Access Protocol）[1]を用い，XML形式でデータ交換を行うインターネット上の分散処理プログラムの総称である．そのWebサービスに関する情報をインターネット上で一元的に管理・運用する仕様がUDDI（Universal Description, Discovery and Integration）[2]であり，その実装がUDDIレジストリとしてIBMやMicrosoftなどによって運用されている[3)~5)]．さらに，Webサービスを利用するための技術情報がWSDL（Web Services Description Language）[6]で記述され，そのWSDLファイルのURLがUDDIレジストリに登録される．したがって，ユーザはUDDIレジストリを検索して必要なWebサービスとその実行に必要な情報を入手することができる．一般的に，ユーザがWebサービスを利用する際には，そのWebサービスを実行するためのクライアントプログラム（以下，スタブという）を開発する必要がある．しかし，その開発にはSOAPを実装す

† 松江工業高等専門学校情報工学科
Department of Information Engineering, Matsue National College of Technology
†† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

るための Axis API[7] や XML プログラミング技術，および XML に関する知識などの高度な専門技術や知識が必要なため，Web サービスの容易な実行が難しく，Web サービスの利用普及を妨げる大きな要因になっていた．

我々は UDDI と WSDL をインターネット上で Web サービスを一元的に管理し，運用する最も基本的なものとしてとらえ，その UDDI を基にユーザが必要とする最適な Web サービスを検索し，動的に実行するシステムの研究開発に取り組んでいる．その実現に対する問題点は大きく以下の 3 点であると考える．

(1) 実行する`Web` サービスごとにスタブ開発が必要であり，その開発には`Axis` や`XML` に関する知識と高度なプログラム専門技術が必要なため，その容易な開発が難しい．
(2) ユーザが必要とする最適な`Web` サービスの検出が現状のキーワード検索だけでは難しい．
(3) `Web` サービスを利用するうえで，そのサービスの機能，実行形式および入出力パラメータに関する正確な理解と把握に時間がかる．

今回，我々はこの中の（1）の問題点を解決するために，WSIF（Web Services Invocation Framework）[8] を利用して Web サービスの出力データ型に依存しない，自動化・統合化された Web サービス動的実行システムの研究開発に取り組んだ．WSIF とは，プロトコルに依存せずに，スタブ開発が不要な Web サービスの動的実行を可能にするフレームワークである．WSIF は WSDL ファイルを直接入力データとして読む API を提供し，プロトコルバインディングを自動的に行う．また WSIF や Axis では，JavaBeans に対するシリアライズ/デシリアライズ API が標準で用意されており，複合型出力データを JavaBeans クラスにマッピングすることでそのオブジェクトとして容易に操作できる．この WSIF を利用したシステム A[9] は WSIF への入力データとして WSDL URL を直接記述し，かつ出力データ型が基本型 Web サービスにのみ限定された動的実行システムである．同じくシステム B[10] は出力データ型が複合型である Zip2Gio Web サービス[11] に限定された動的実行システムである．この場合も入力データとして，WSDL URL を直接記述し，さらにその出力データ型に対応した JavaBeans クラスがユーザ側にあらかじめ存在している場合にのみ実行可能なシステムである（以下，システム A，B をまとめて従来システムという）．これらのシステムは Web サービスの出力データ型に応じて使い分ける必要があるとともに，WSIF への入力データである Web サービスの WSDL URL をつねに事前に把握し設定する必要があること，さらに出力が複合型の Web サービスに対してはその出力データに対応した JavaBeans クラスをつねに事前に用意する必要があること，など実行に関する制限・制約事項があるため，ユーザにとって利用しにくいシステムである．また，Web サービスやそのメソッドに対する名前空間指定や入出力データ名とその型指定，および出力オブジェクトから出力データを抽出するための getter メソッドの設定など人手を介した多くの複雑な設定処理が必要であり，そのためには WSDL に関する詳細な知識も要求される，などの点でも容易に利用できないのが現状である．

そこで我々は，ユーザがこれらの制限・制約事項にとらわれずに容易に Web サービスの動的実行が可能なシステムの実現を目標にして，前述の手動による設定が必要な処理をすべて自動化した．さらに複合型出力データを持つ Web サービスに対しても，JavaBeans クラスを Web サービス実行時に動的に生成する手法を考案し基本型に対する処理と統合して，出力データ型に依存しない自動化・統合化された Web サービス動的実行システムとして開発した．

以下，2 章では本システムで利用した Web サービスの基盤技術について述べ，3 章で開発した本システムの特徴や機能について従来システムと比較して詳細に説明する．4 章では本システムの評価実験について従来システムと比較して説明する．5 章で関連研究について述べ，最後に 6 章でまとめを述べる．

2. Web サービスの基盤技術

Web サービスおよび本システムの基盤となる要素技術について，その概要を説明する．

2.1 UDDI[12]

UDDI は Web サービスに関する様々な情報を公開し，検索する方法を提供する仕様である．その実装が UDDI レジストリであり，IBM や Microsoft などが実際に運用している．UDDI レジストリは *businessEntity*，*businessService*，*bindingTemplate*，*tModel* の 4 種類の階層型データ要素から構成される．*businessEntity* はその最上位の要素であり，サービス提供者に関する情報を提供し，その中に複数の *businessService* 要素を記述できる．*businessService* 要素は提供するサービスに関する情報を示し，その中に複数の *bindingTemplate* 要素を記述できる．*bindingTemplate* 要素は Web サービスに接続するための技術情報（サービス型へのポインタや接続先 URL など）を提供する．*tModel* 要素は Web サービスの技術仕様を表すデータ

構造のルート要素であり，その中の *overviewURL* 要素で Web サービスの技術情報を記述した WSDL ファイルへのポインタを記述する．また *tModel* 要素へのポインタが *tModelKey* という 128 ビットの数値で表される．

本システムは *tModel* 要素を扱い，Web サービスを特定し，その WSDL URL を検出する．これらの処理を UDDI4J API[13] を用いて実装した．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．

2.2 WSDL[12]

WSDL は Web サービスのインタフェースや接続情報を記述する言語仕様であり，その WSDL ファイルには Web サービス実行に必要な技術情報（Web サービス名，メソッド名，アクセスポイントや入出力データ情報など）が記述される．これらの情報を抽出し，WSIF API に設定することで Web サービスの動的実行が可能になる．この WSDL は 5 つの独立したデータ要素から構成される．*types* 要素は Web サービスへの入出力データ型の定義である．*message* 要素は入出力データの抽象定義であり，この中の *port* 要素で入出力データ名とその型が定義される．*portType* 要素は Web サービスの抽象定義であり，入出力のメッセージとメソッドを定義する．*binding* 要素は Web サービスの実装定義であり，*portType* 要素で定義された操作の具体的なプロトコルとメッセージのデータ形式を定義する．*service* 要素は Web サービスの実装とアクセスポイントを定義する．このように WSDL ではサービスに対する操作やメッセージを抽象的に記述し，具体的なプロトコルと分離して定義されるので，サービスを特定のプロトコルではなく，複数のプロトコルにバインディングできる．

本システムは *types*，*message*，*portType* 要素を扱い，その解析と必要データの抽出，WSIF API への設定処理を XML パーサである Xerces API[14] と WSDL4J API[15] を用いて実装した．WSDL4J は WSDL 記述内容をオブジェクトとして操作できる Java クラスライブラリである．

2.3 WSIF[16]

WSIF は，

(1) `SOAP` プロトコルなどのある特定のプロトコルに依存しない形式，および
(2) スタブを必要としない完全に動的な形式，

で Web サービスを呼び出すことを目的に開発されたフレームワークである．この WSIF は WSDL ファイルを直接読む API を提供し，また WSDL ファイルに記述されているポートタイプ名やそれの名前空間，オペレーションなどを各 API に設定することでポートタイプに対応したプロトコルとポートに自動的にバインディングする機能を持つので，ユーザは入出力メッセージと特定のプロトコルとの対応を意識する必要がない．したがって WSIF を利用することで，これまで Axis を用いて直接記述していた Web サービスアクセスコードが不要になり，WSDL 記述内容を基にした抽象度の高い動的な Web サービス実行システムが開発可能になる．WSIF は IBM によって開発され，現在は Apache Software Foundation に移管され研究開発が行われている．

3. 自動化・統合化された Web サービス動的実行システム

この章では，まず従来システムの問題点について説明し，次にそれらの問題点を解決した本システムの特徴を述べる．続いて本システムの機能について，WSDL 記述内容とともに詳細に説明する．

3.1 従来システムの問題点

Web サービスを実行するための技術情報は WSDL ファイルに記述され，出力データに関する情報もそこに記述される．そして，Web サービスの出力データ型は次の 2 種類

(1) 文字列型，整数型などの基本型
(2) 基本型データを複数個組み合わせた複合型

に大別され，その型に従って出力データが Web サービスからクライアントに返される．ここで複合型データとは，たとえば企業の従業員データ（ID コード，氏名，年齢，役職，住所などの文字列型，整数型データから構成される）などである．基本型データの値はクライアント側でそのまま抽出することができる．一方，複合型データの場合はそのデータ型に対応した JavaBeans クラスにマッピングされ，そのオブジェクトとしてクライアントに返されるので，ユーザはそのオブジェクトに対して getter メソッドを適用して個々の基本型データの値を抽出しなければならない．したがって，クライアント側にも Web サービス実行前に JavaBeans クラスが必要であり，そして WSIF を用いてもクライアント側で JavaBeans クラスを実行時に動的に生成することができず，人手による事前の開発が必要であった．そのため，従来のシステムでは出力データ型別に 2 種類のシステムがあり，実行前に出力データ型を識別し，システムを使い分ける必要があった．

また，上記の問題以外にも実行する Web サービス

の WSDL URL の検出や WSIF API に設定する様々なデータ（Web サービス名，メソッド名，名前空間，入出力パラメータ名とその型など）を WSDL ファイルから読み取り，手動で記述するという煩雑な処理が必要であった．そのためには WSDL の構造と WSIF API に関する専門知識が必要なため，ユーザにとって負荷が大きく，利用しにくい状況であった．

3.2 システムの特徴

上記問題点を解決した本システムの特徴である，

(1) 設定処理の自動化，
(2) 出力データ型に依存しない Web サービスの動的実行，

について説明する．

（1）に対しては，これまで実行する Web サービスごとに以下の 3 工程，

a. 指定された Web サービスの WSDL URL の検出
b. その WSDL ファイルの解読と WSIF API への設定データの抽出
c. その抽出データの WSIF API への設定

を UDDI や WSDL，WSIF に関する専門知識を有するユーザがすべて手動で行っていた．我々はこの 3 工程をすべて自動化した．まず a. の処理に対しては，UDDI4J API を利用して UDDI レジストリから実行する Web サービスの WSDL URL を自動検出してダウンロードするプログラムを新規に開発し，自動化した．また b., c. の処理に対しては，Xerces と WSDL4J API を用いて WSDL 記述内容を自動解析し，WSIF API に対する設定データの抽出とそれを自動設定するプログラムを開発し，すべて自動化した．まず，基本型に対する自動化システムを開発し[17),18)]，それを複合型に応用した．これらの自動化処理により，Web サービス実行工数が削減されるとともに，特別な専門知識も不要となり，誰もが容易に実行できるようになった．

（2）に対しては，従来，Web サービスの出力データ型ごとに動的実行システムを使い分けており，つねに事前に出力データ型を識別する作業と前述の煩雑な手動工程が必要であった．しかし，その手動工程の自動化とこれまで実現できなかったクライアント側における複合型に対する JavaBeans クラスの動的生成手法を新規に考案し，実装する[19)] とともに，基本型に対する自動化システムと統合し，出力データ型に依存しない自動化・統合化された Web サービス動的実行システムとして実現した．

```
<definitions name="TemperatureService" targetNamespace=
 http://www.xmethods.net/sd/TemperatureService.wsdl
 xmlns:tns="http://www.xmethods.net/sd/TemperatureService.wsdl"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    ……..
 <message name="getTempRequest">
  <part name="zipcode" type="xsd:string"/>
 </message>
 <message name="getTempResponse">
  <part name="return" type="xsd:float"/>
 </message>
 <portType name="TemperaturePortType">
  <operation name="getTemp">
   <input message="tns:getTempRequest"/>
   <output message="tns:getTempResponse"/>
  </operation>
 </portType>
```

図 1 出力が基本型である場合の WSDL 記述の一部
Fig. 1 WSDL description (part) that the output is the primitive data type.

3.3 WSDL 記述内容と WSIF API 設定データの抽出

出力データが基本型と複合型の場合の WSDL 記述内容を具体的に示し，WSIF API に設定するデータ項目の抽出について説明する．

3.3.1 出力データが基本型の場合

出力データが基本型である場合の WSDL 記述の一部を **図 1** に示す．*portType* 要素の *name* 属性値が Web サービス名を表し，この例では “TemperaturePortType” である．*operation* 要素の *name* 属性値がメソッド名を表し，この例では “getTemp” である．*part* 要素の *name* 属性値と *type* 属性値が入出力データ名とそのデータ型を表し，この例では入力データ名が “zipcode” で string 型であり，出力データ名が “return” で float 型である．これらのデータを Xerces と WSDL4J API を用いて抽出し，WSIF API に自動設定している．

3.3.2 出力データが複合型の場合

出力データが複合型の WSDL 記述の一部を **図 2** に示す．この場合は，基本型に比べて複雑な構造となる．この例では出力データが名前空間 tns1 に属する “GData” 型であり，その “GData” 型のデータ構成が *types* 要素の *element* 要素で示されている．この例では “GData” 型は，両方とも string 型である “code” と “maker” の 2 変数から構成される．これらを Xerces と WSDL4J API を用いて抽出し，WSIF API に自動設定している．

3.4 システムの機能

本システムは 3 ステップから構成される．その全体の処理の流れを **図 3** に示す．ステップ 1 は “UDDI レジストリ検索と WSDL URL 抽出” を行う．ステップ 2 は “WSDL 解析と WSIF API 設定データの抽出”

```
<wsdl:definitions targetNamespace=
    "http://192.168.1.185:8080/axis/services/GoodsService2"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
        ……..
    xmlns:intf=http://192.168.1.185:8080/axis/services/GoodsService2
    xmlns:tns1="http://10.70.51.20:8080/axis/services"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
   <schema targetNamespace=http://10.70.51.20:8080/axis/services
                    xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
     <complexType name="GData">
       <sequence>
         <element name="code" nillable="true" type="xsd:string"/>
         <element name="maker" nillable="true" type="xsd:string"/>
       </sequence>
     </complexType>
       <element name="GData" nillable="true" type="tns1:GData"/>
  </schema>
 </wsdl:types>
 <wsdl:message name="getTempMakerRequest">
   <wsdl:part name="in0" type="xsd:string"/>
 </wsdl:message>
 <wsdl:message name="getTempMakerResponse">
   <wsdl:part name="getTempMakerReturn" type="tns1:GData"/>
 </wsdl:message>
 <wsdl:portType name="GoodsService2">
   <wsdl:operation name="getTempMaker" parameterOrder="in0">
     <wsdl:input message="intf:getTempMakerRequest"
                    name="getTempMakerRequest"/>
     <wsdl:output message="intf:getTempMakerResponse"
                    name="getTempMakerResponse"/>
   </wsdl:operation>
 </wsdl:portType>
```

図 2 出力が複合型である場合の WSDL 記述の一部
Fig. 2 WSDL description (part) that the output is the complex data type.

を行う．ステップ 3 は “Web サービス実行と出力データ抽出” を行う．以下，各ステップの機能について説明する．

3.4.1 ステップ 1：UDDI 検索と WSDL 抽出

ここでは，まず *tModel* 名と *tModelKey* 値を入力データとして UDDI レジストリを検索し，Web サービスを特定する．IBM や Microsoft など計 3 種類の UDDI レジストリから Web サービスが検出可能である．本システムは Web サービスを特定するために *tModel* 名と *tModelKey* 値を必要とする．なぜなら，複数の Web サービスが同じ *tModel* 名を持つ場合があるので（図 9），その特定のためには Web サービスごとに固有な値を持つ *tModelKey* 値を必要とする．その *tModelKey* 値は各 UDDI レジストリ付属の GUI を利用して確認することができるが，我々は *tModel* 名と *tModelKey* 値を対で出力する検索プログラムを用意して簡単に *tModelKey* 値が入手できるようにした．これらのパラメータを用いて *tModel* 要素を検索し，入力条件に合致する *overviewURL* 要素中の WSDL URL を自動検出し，その URL から WSDL ファイルをダウンロードする．

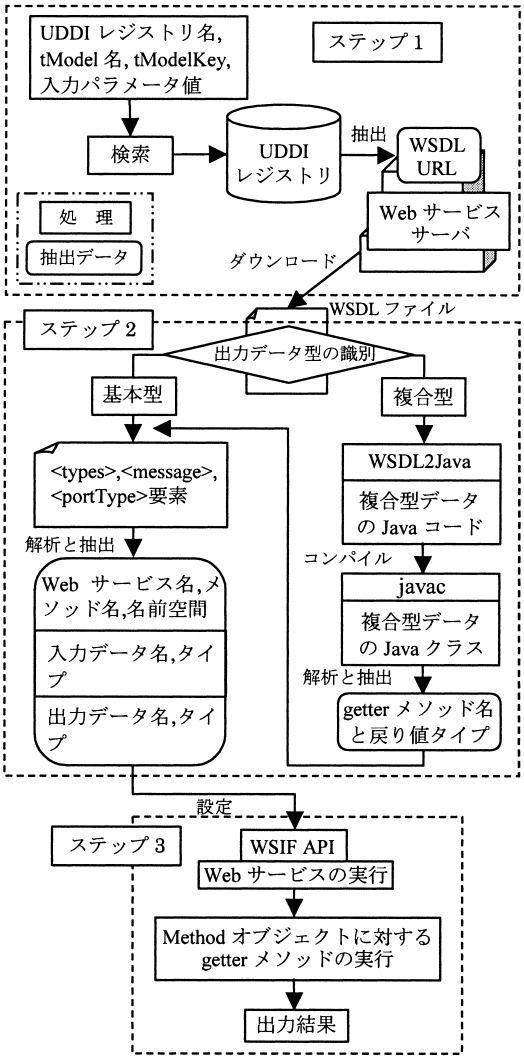

図 3 システム全体の処理の流れ
Fig. 3 System workflow.

3.4.2 ステップ 2：WSDL 解析と設定データ抽出

ここでは，ステップ 1 で入手した WSDL ファイルから WSIF API に設定するデータを自動抽出する．最初に Web サービスの出力データ型を識別し，複合型の場合は WSDL2Java [20] を用いてそのデータ型に対応した JavaBeans コードを動的に生成する．そして，Java2 SE ライブラリ Runtime の Runtime.getRuntime().exec() メソッドを利用してコンパイルし，クラスを生成する．Web サービス実行後の出力データはこの複合型のオブジェクトとしてクライアントに返されるので，個々の出力値を得るためにはそのオブジェクトに対して getter メソッドを実行しなければならない．その getter メソッドを抽出するために，我々は Java2 SE ライブラリ中の Class，Method お

```
1:  複合型データに対する JavaBeans クラスのオブジェクトを
    生成する.
2:  そのオブジェクトの全メソッドを配列に格納する.
3:  for ループ（配列中の全メソッドをチェックする）: 開始
4:      メソッドの戻り値を示すクラスを抽出する.
5:      getter メソッドのみを抽出する.
6:      その getter メソッドの戻り値タイプを抽出する.
7:  for ループ : 終了
```

図 4 getter メソッドの抽出
Fig. 4 Extraction of the getter method.

```
1:  出力メッセージのオブジェクトを生成する.
2:  複合型データに対する JavaBeans クラスのオブジェクトを
    生成する.
3:  出力メッセージ中の複合型データの出力オブジェクトを得る.
4:  その出力オブジェクトへの参照を得る.
5:  for ループ（全ての getter メソッドをチェックする）: 開始
6:      getter メソッドの戻り値が String タイプか？
7:          その getter メソッドへの参照を得る.
8:          その getter メソッドを実行する.
9:          実行結果を出力する.
10:     残りの基本型タイプに対して, 6:から 9:を繰り返す.
11: for ループ : 終了
```

図 5 getter メソッドの動的実行
Fig. 5 Dynamic invocation of the getter method.

よび Object API を利用して，正確にそれらのメソッド名と戻り値タイプを抽出する方法を考案し，実装した．その手順を **図 4** に示す．

また，ここでは出力データ型には依存しない共通動的実行データも抽出する．*portType* 要素から Web サービス名を抽出し，*operation* 要素からメソッド名を抽出する．*input/output* 要素および *part* 要素から入出力データ名とそれらのデータ型を抽出する．複合型に対しては，さらに *types* 要素を解析して出力データ名と複合型名，名前空間を抽出する．

3.4.3 ステップ 3：サービス実行と出力データ抽出

ここでは，ステップ 2 で抽出されたデータを WSIF API に自動設定して Web サービスを実行する．複合型出力データの場合には，クライアント側における JavaBeans クラスの動的生成と参照および getter メソッドの動的実行など **図 5** で示す処理が別途必要になる．出力データはこの JavaBeans クラスのオブジェクトとしてクライアントに返される．そのオブジェクトに対して，ステップ 2 で求めた getter メソッドを適用して出力値を得る．基本型出力データの場合には，ステップ 2 で求めた出力データ名とデータ型を WSIF API の getObjectPart() メソッドに設定し実行するだけでよい．任意の複合型データを持つ Web サービスに対して，このクライアント側における動的処理の自動化が難しく今まで実現されなかったが，我々は本システムで完全に自動化した．

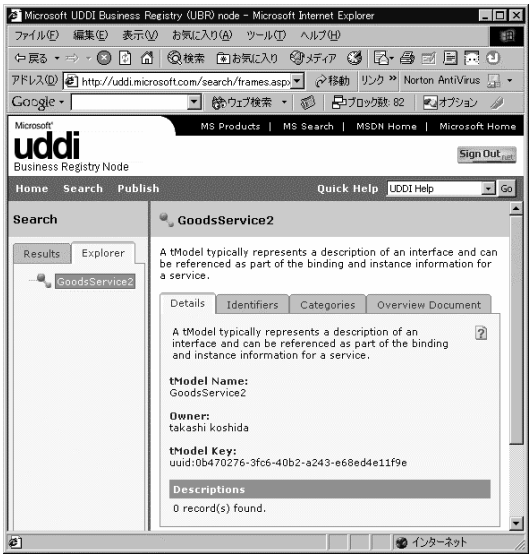

図 6 UDDI レジストリに登録された Web サービス
Fig. 6 Web Service registered into the UDDI registry.

4. 評価実験

本システムの評価実験を基本型 4 種類，複合型 4 種類の Web サービスに対して行い，実行に要する工数を従来システムと比較し，本システムの有用性を確認した．従来システムの実行結果と本システムの実行結果を比較して説明する．

4.1 システムの評価

我々は実際に動作している Web サービスを 3 種類の UDDI レジストリ：IBM UDDI ビジネスレジストリ[3]，IBM UDDI テストレジストリ[4]，Microsoft UDDI レジストリ[5] に登録し，それらのレジストリを使ってシステムの評価実験を行った．その実際の登録データを **図 6** に示す．

実験の目的はシステムの動作検証と本システムによる Web サービス実行効率の検証である．実験に使用した Web サービスは基本型 4 種類，複合型 4 種類であり，すべて本システムで正常に実行できた．**表 1** に，Web サービスの動的実行に必要な工程とその手動と自動の区別を従来システムと本システムに分けてまとめた．実際の実験においては，この手動と自動の違いが処理時間の差として大きく反映される．

基本型 Web サービスに対しては，従来システムにおいても手動で抽出・設定する部分は 1 カ所だけであり，他は本システムと同様にすべて自動化されている．しかし，最初に Web サービスの出力データ型を判別し，使用するシステムを選択する処理が必要であり，その工数が余分にかかる．本システムでは出力データ

表 1 従来システムと本システムにおける Web サービス実行前工程の比較
Table 1 Comparison with the conventional systems and this system in a Web Service preparation process.

	従来システム		本システム	
出力データ型	基本型	複合型	基本型	複合型
システム	A	B	UniDyRun6	
実行 Web サービス数	4	3	4	4
メソッド名の抽出と設定	手動（1）	手動（1）	自動	自動
WSIF API に設定するデータの抽出	自動	手動（10）	自動	自動
WSDL API へのデータ設定	自動	手動（10）	自動	自動
JavaBeans クラスの生成	—	手動（*1）	—	自動
その getter メソッド記述	—	手動（平均 5）	—	自動

（数値）は抽出または設定するデータ数を示す.（*数値）は生成するクラス数を示す.

図 7 システム A の実行結果（基本型）
Fig. 7 Execution result of system A (primitive type).

型に依存せず 1 つのシステムで実行できるので，本システムを使用することにより，その判別工数も含めて従来システムの約 1/2〜1/3 に工数削減される．

複合型 Web サービスの場合は，従来の方式ではシステム B があるが，これは Zip2Gio Web サービス実行用に限定されたシステム（図 8）である．他の複合型 Web サービスを実行するためには，表 1 で示したすべての工程を毎回手動で解析・設定し，コンパイルする作業が必要である．今回，Zip2Geo 以外の 3 種類の複合型 Web サービスに対する作業時間は専門知識を有する技術者が行って約 25 分から 30 分であった．しかし，本システムではすべての工程がソフトウェアにより自動化されているので，実行までに要する作業時間は Web サービスの *tModel* 名と *tModelKey* 値の確認時間の約 2 分のみである．したがって本システムの使用により約 1/10 またはそれ以上の工数削減となり，Web サービス実行作業効率が大幅に改善された．

4.2 従来システムの実行結果

基本型 Web サービスに対するシステム A の実行結果を **図 7** に示す．企業名（ここでは IBM）を指定するとその株価を出力する Web サービスの実行例である．このシステムでは，入力パラメータ値として Web サービスの WSDL URL とメソッド名を指定する必要があり，それらが分からないと実行できない．

次に，複合型 Web サービスに対するシステム B の実行結果を **図 8** に示す．指定した zip コードに対応した都市の地理情報を出力する Zip2Gio Web サービスの実行例である．ここでも入力パラメータ値として Web サービスの WSDL URL を指定する．このシステム B には，複合型データに対応した JavaBeans クラスと WSDL ファイルがあらかじめ添付されている．したがって，このシステムでは JavaBeans クラス名や名前空間，Web サービス名，メソッド名，さらには getter メソッドなどがすべて直接記述されているため汎用性がまったくなく，実行する Web サービスごとに再プログラミングと再コンパイルが必要である．

4.3 本システムの実行結果

本システムの実行結果を図 9 から図 12 に示す．図 9

図 8 システム B の実行結果（複合型）
Fig. 8 Execution result of system B (complex type).

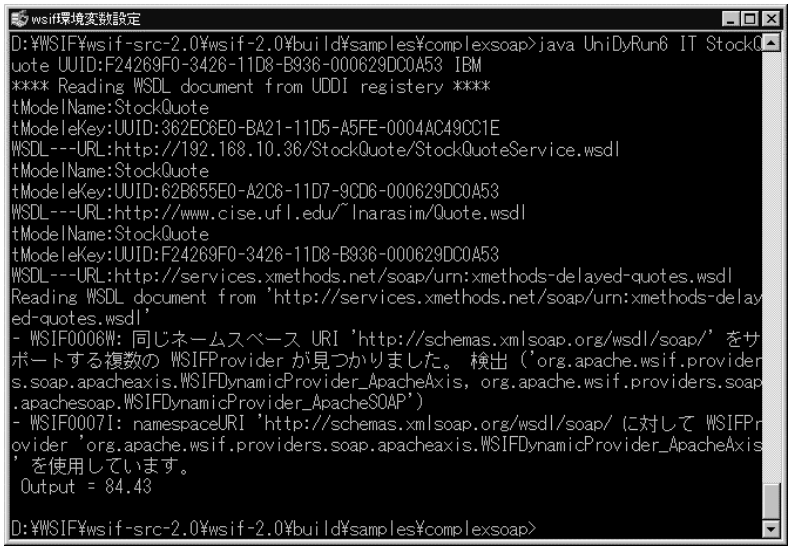

図 9 本システムの実行結果：その 1（基本型）
Fig. 9 Execution result of this system: 1 (primitive type).

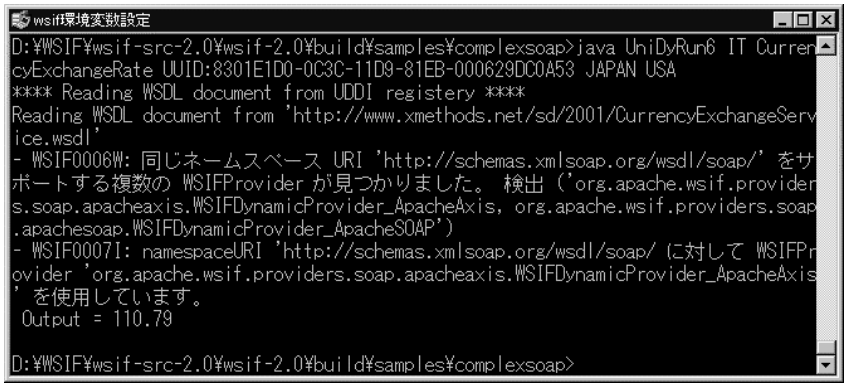

図 10 本システムの実行結果：その 2（基本型）
Fig. 10 Execution result of this system: 2 (primitive type).

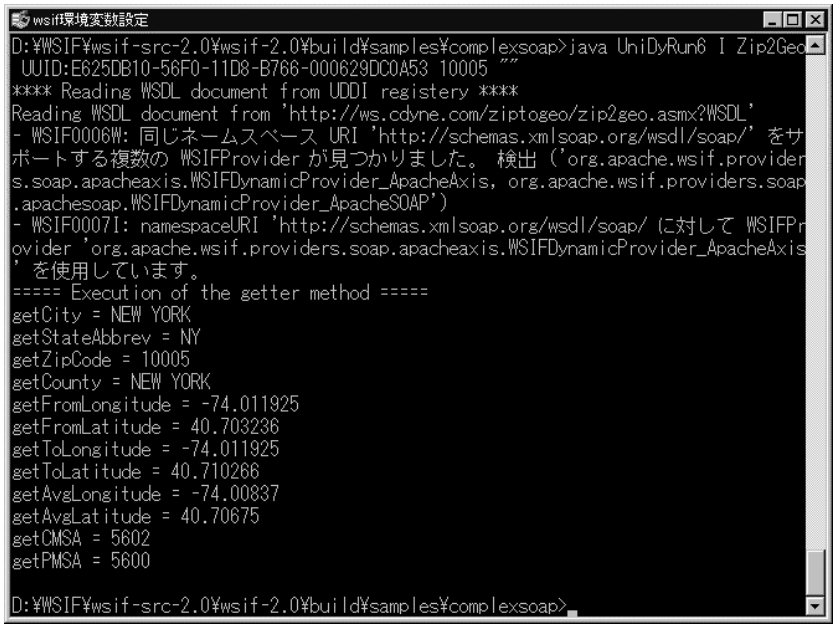

図 11 本システムの実行結果：その 3（複合型）
Fig. 11 Execution result of this system: 3 (complex type).

図 12 本システムの実行結果：その 4（複合型）
Fig. 12 Execution result of this system: 4 (complex type).

と図 10 は出力データが基本型の例であり，**図 9** は図 7 と同じ Web サービスの実行結果である．ここでは UDDI レジストリの指定として，IBM UDDI テストレジストリを示す “IT” を指定している．同じ *tModel* 名 “StockQuote” をもつ複数の Web サービスが登録されているため，*tModelKey* 値で Web サービスを特定している．**図 10** は 2 国間の為替レートを出力する Web サービスの実行例である．

図 11 と **図 12** は出力データが複合型の例である．2 種類の異なる Web サービスに対しても，本システムで汎用的に処理できることが分かる．図 11 は図 8 と同じ Web サービスの実行結果である．図 12 は 4 桁の商品コードに対応した商品製造メーカをそのコードとともに出力する Web サービスの実行例である．図 11 の “I” は IBM UDDI ビジネスレジストリの指定であり，図 12 の “M” は Microsoft UDDI レジストリの指定である．どちらの場合もクライアント側で JavaBeans クラスを動的に生成し，Web サービスを実行している．

5. 関 連 研 究

本システムと同様に，UDDI と WSDL を Web サー

ビスの核と見なし，研究を進めている点で共通するシステムとしてWebサービス・マッチメイキング[21),22)]がある．このシステムはオントロジ記述言語としてDAML-S[23)]を利用して，それのプロファイル記述データをUDDIの*businessEntity*および*businessService*要素にマッピングするとともに，さらに*categoryBag*要素の*keyedReference*要素で入出力データに対する条件や制約を新たに記述・登録し，オントロジ技術を用いて，ユーザが必要とするWebサービスの検出を支援するシステムである．しかし，これはDAML-Sプロファイル記述データを付与したサンプルWebサービスに対しては有効であるが，プロファイル記述データを持たないWebサービスに対しては効果がない．実用化するためには，UDDIレジストリに登録されているすべてのWebサービスに対して，効果的なプロファイル記述データを付与することが必要であり，その実現には多大な工数と負荷が予想されるが，その実現手法について触れていない．また，検出されたWebサービスは実行されなければ意味がないと考えるが，その検出と連携した効果的な実行手法については述べられていない．従来どおりのユーザ負荷が大きいスタブを利用する方法での実行が予想される．Webサービスを利用するユーザにとっては，Webサービスの検出とその実行を別々に分離して操作するシステムより，連続して処理できるシステムの方がより使いやすく，効率的であると考えられる．その観点から考えると，検出したWebサービスと連動して，その効果的な実行方法を確立する意義は大きいと考える．本システムでは，UDDIレジストリからのWebサービスの検出とその実行を連続した一連の自動化処理として実現し，かつその動的な実行方法を新規に考案して，Webサービスの出力データ型に依存しない，自動化・統合化されたWebサービス動的システムとして実現した点で，よりユーザ利便性に優れていると考える．さらに将来的には，マッチメイキングと本システムを融合することで，ユーザにとって最適なWebサービスの自動検出とその動的実行の自動化・統合化が実現できると考える．

また，出力データ型が複合型であるWebサービスの動的実行に関する関連研究として，IBMのJROM（Java Record Object Model）[24)]がある．このJROMは複合型データをメモリ上に木構造として保持するJROMComplexValueやそれをシリアライズ/デシリアライズするためのAPIなどから構成され，AxisやWSIFなど既存のWebサービス開発環境に追加する形で動的実行を可能にするものである．したがって，AxisやWSIFコード中に複合型データを操作するためのJROM APIプログラミングが必要であり，加えてJROMコードのコンパイルと実行のために，クライアント側でJROM APIのインストールおよびJava総合開発環境Eclipse[25)]に含まれる5種類のAPIのインストールも別途必要となり，JROMを使うための開発および実行環境設定に対する負荷やJROMプログラミングに対する負荷が増大する．さらに，Webサービス自体もJROM対応のためのコード修正が必要であり，そのサーバ側にもJROM APIのインストールが必要となり，標準のWebサービス運用環境から大幅な設定変更が必要である．それに対して我々は，Axisライブラリの WSDL2JavaおよびJava2 SEの標準ライブラリであるClass，Method，Object，Runtimeクラスなどすべて標準のライブラリを利用して，複合型データに対する動的実行を実現するために必要なJavaBeansクラスの動的処理を実現した．Webサービスのサーバ側はもちろんのこと，クライアント側においても他のフレームワークの導入や設定変更なども必要なく，すべて既存の標準環境で動的実行が実現できる点で有利である．

6. おわりに

我々はWebサービスを一元的に管理・運用し，記述する最も基本的なアーキテクチャとしてUDDIとWSDLを考えた．さらにWebサービス動的実行のフレームワークとしてWSIFを導入し，それらを基に，UDDIレジストリからWebサービスを検出し，その動的実行までを自動化した出力データ型に依存しない統合化システムを研究し，開発した．特に，これまで動的実行が不可能であった複合型出力データを持つWebサービスに対して，そのデータに対応したJavaBeansクラスのクライアント側での実行時動的生成とgetterメソッドの自動抽出，動的実行を実現し，Webサービス実行時におけるユーザの開発負荷を格段に軽減し，利便性を向上させた．本システムを用いて，出力データ型が基本型であるWebサービス4種類，複合型であるWebサービス4種類について評価実験を行った．その結果，本システムを用いた方が，従来システムよりWebサービス実行工数が約1/2〜1/10に削減され，本システムの有用性が確認された．

今後の課題として，より多くの複合型Webサービスについてさらなる評価実験を行い，本システムの実用性と信頼性を確立したい．また，オントロジ技術とエージェント技術を用いて，効果的なWebサービスの検出，選別と合成などをユーザ側の視点で研究し，

システムとしての実用化を目標に研究開発を進めたい．

参 考 文 献

1) Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S. and Winer, D.: Simple object access protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/
2) OASIS: UDDI. http://www.uddi.org/
3) IBM corporation: IBM UDDI Business Registry Version2. http://uddi.ibm.com/ubr/
4) IBM corporation: IBM Test Registry. https://uddi.ibm.com/testregistry/
5) Microsoft Corporation: Microsoft uddi Business Registry Node. http://uddi.microsoft.com/default.aspx
6) Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S.: Web services description language (WSDL) 1.1. http://www.w3.org/TR/WSDL
7) Apache Software Foundation: Axis. http://ws.apache.org/axis/
8) Apache Software Foundation: Web services invocation framework (WSIF). http://ws.apache.org/wsif/
9) Apache Software Foundation: Web services invocation framework (WSIF). wsif-2.0.zip(wsif-2.0\ build\ samples\ clients\ dynamicInvoker.class). http://ws.apache.org/wsif/
10) Apache Software Foundation: Web services invocation framework (WSIF). wsif-2.0.zip(wsif-2.0\build\samples\complexsoap\client\dynamic\Run.class). http://ws.apache.org/wsif/
11) Zip2Geo. http://ws.cdyne.com/ziptogeo/zip2geo.asmx?WSDL
12) 高瀬俊郎：UDDI と WSDL，情報処理，Vol.42, No.9, pp.870–887 (2001).
13) IBM developerWorks: UDDI4J. http://oss.software.ibm.com/developerworks/projects/uddi4j
14) Apache Software Foundation: Xerces. http://xml.apache.org/xerces2-j/
15) IBM developerWorks: WSDL4J. http://oss.software.ibm.com/developerworks/projects/wsdl4j
16) Fremantle, P.: Applying the Web services invocation framework, IBM developerWorks. http://www-106.ibm.com/developerworks/webservices/library/ws-appwsif.html (2002).
17) 越田高志，植村俊亮：WSIF における Web サービス・メソッドの自動設定，電子情報通信学会 2004 年総合大会講演論文集，D-9-5 (2004).
18) Koshida, T. and Uemura, S.: An Effective Dynamic Invocation System for Web Service, *Proc. BAI'2004*, 3-3, CD-ROM (2004).
19) Koshida, T. and Uemura, S.: Automated Dynamic Invocation System for Web Service with a User-defined Data Type, *Proc.EOOWS'2004*, pp.1–8 (2004).
20) Apache Software Foundation: WSDL2Java. http://ws.apache.org/axis/
21) Kawamura, T., Hasegawa, T., Ohsuga, A. and Yamamoto, J.: Proposal of Semantics-based Web Service Matchmaking, *Proc. ICCIMA'2001*, pp.87–92, IEEE (2001).
22) Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K.: Importing the Semantic Web in UDDI, *Proc. WES'2002*, Lecture Notes in Computer Science, No.2512, Springer-Verlag (2002).
23) DAML: The DARPA Agent Markup Language Services (DAML-S). http://www.daml.org/services/
24) IBM alphaWorks: JROM. http://www.alphaworks.ibm.com/tech/jrom
25) eclipse.org: Eclipse. http://www.eclipse.org/

(平成 16 年 6 月 17 日受付)
(平成 17 年 1 月 7 日採録)

越田 高志（学生会員）

1980 年福井大学大学院工学研究科修士課程応用物理学専攻修了．同年松下電器産業株式会社入社．エンジニアリング分野の各種ソフトウェアシステムの研究開発に従事．1996 年より松江工業高等専門学校情報工学科勤務．2002 年より奈良先端科学技術大学院大学情報科学研究科博士後期課程在学中．技術士（情報工学部門）．ACM，電子情報通信学会，日本データベース学会各会員．

波多野賢治（正会員）

1995 年神戸大学工学部計測工学科卒業．1999 年同大学大学院自然科学研究科博士後期課程修了．博士（工学）．同年から奈良先端科学技術大学院大学情報科学研究科助手，現在に至る．2005 年 AT&T Labs-Research 訪問研究員．情報検索システム，データベースシステムに関する研究に従事．電子情報通信学会，ACM，IEEE Computer Society，日本データベース学会各会員．

天笠　俊之（正会員）

1994年群馬大学工学部情報工学科卒業．1999年同大学大学院工学研究科修了．博士（工学）．同年から奈良先端科学技術大学院大学情報科学研究科助手．XMLデータベース，装着型コンピュータにおけるデータベース応用等の研究に従事．電子情報通信学会，ACM，IEEE Computer Society，日本データベース学会各会員．

宮崎　　純（正会員）

1992年東京工業大学工学部情報工学科卒業．1997年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（情報科学）．同大学助手を経て，2003年より奈良先端科学技術大学院大学助教授．2000〜2001年テキサス大学アーリントン校客員研究員．2003年より科学技術振興機構さきがけ研究員．高性能・高機能データベースシステムの研究に従事．電子情報通信学会，日本データベース学会，IEEE CS，ACM SIGMOD各会員．

植村　俊亮（フェロー）

1964年京都大学工学部電子工学科卒業．1966年同大学大学院工学研究科修士課程修了．同年電気試験所（産業技術総合研究所）．1970年マサチューセッツ工科大学電子システム研究所客員研究員，1981年電総研ソフトウェア部プログラム研究室長，1988年東京農工大学教授を経て，1993年から奈良先端科学技術大学院大学情報科学研究科教授．データ工学，データベースシステムの研究に従事．工学博士．IEEE Fellow，電子情報通信学会フェロー．現在，情報処理学会理事，日本情報考古学会理事，データベース振興センター評議員等．