NAIST-IS-DD0261012

博士論文

CALS/ECにおける
Webサービス動的実行手法に関する研究

越田　高志

2005 年 2 月 3 日

奈良先端科学技術大学院大学
情報科学研究科 情報生命科学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士(工学) 授与の要件として提出した博士論文である。

越田　高志

審査委員：

植村 俊亮 教授　（主指導教員）
関 浩之 教授　（委員）
松本 健一 教授　（委員）
宮崎 純 助教授　（委員）

ＣＡＬＳ/ＥＣにおける
Ｗｅｂサービス動的実行手法に関する研究*

越田 高志

内容梗概

本研究は，ユーザサイドにおける Web サービス実行に関する問題を解決する手法についての研究である．これまでの CALS/EC システムは特定企業間に固定された電子商取引に代表される静的なシステムであった．しかし，今後は世界規模での市場開拓やビジネスチャンス獲得に向けて，ユーザニーズに応じて柔軟な商取引が可能である動的な CALS/EC が求められている．そして，その動的な CALS/EC を実現する新しい技術として SOAP，UDDI，WSDL を基盤技術とした Web サービスが開発され，その実用化研究が進められている．Web サービスとは，通信データとして XML を用い，SOAP プロトコルで通信する分散処理ソフトウェアの総称である．Web サービス実行に関する技術情報は WSDL で記述され，Web サービスの内容や提供者に関する情報は UDDI レジストリに登録される．ユーザは UDDI レジストリを検索し，必要な Web サービスを検出し実行するという利用形態である．Web サービスは，XML ベースの相互運用性やプラットフォーム独立などの柔軟性を備えているが，ビジネス分野における Web サービスの利用は拡大していない．その大きな原因は次の 3 点であると考える．

（１）Web サービス実行時に，ユーザ側でその Web サービスを駆動するスタブが必要であり，その開発負荷が大きい．

（２）ユーザが必要とする，または利用したい Web サービスの検出が難しい．

（３）ユーザが Web サービスを利用する際に，その機能や入出力インターフェースなど利用法について短時間で正確に知ることが難しい．

*奈良先端科学技術大学院大学 情報科学研究科 情報生命科学専攻博士論文,NAIST-IS-DD0261012, 2005年2月3日．

本研究では, これらの問題点を解決する新しい手法及び概念を提案する. 更に, その手法及び概念を具現化した Web サービスとその Web サービスの疎結合として構成される B2B システムを開発し，実際のビジネスプロセスに適用した実証実験を行い，その有用性を確認する.

最初に Web サービスの CALS/EC への適用を検証するために，UDDI レジストリを構築するとともに，実際のビジネスプロセスに対する Web サービスを開発した．そしてそれらを商品調達 B2B システムとして実装し，ユースケースを用いて評価し，適用可能性を確認した．次に，（１）の問題点を解決する手法として，複合型出力データに対する JavaBeans 動的生成手法，ならびに動的解析手法を提案し，UDDI レジストリから Web サービスを検出し，そのままスタブレスに実行する，出力データ型に依存しない Web サービス動的実行システムを実装した．続いて（２),（３）の問題点を解決する手法として，プリミティブ Web サービスという概念を提案し，それに基づき，プリミティブ Web サービスとそれらを連携・制御するエージェントを開発し，商品調達 B2B システムとして実装した．そのシステムを実際のユースケースに適用し，従来の Web サービスを用いたシステムと比較して，その有効性を確認した.

キーワード

CALS/EC, Webサービス, UDDIレジストリ，B2Bシステム，動的実行，プリミティブWebサービス

Studies on Web Service Dynamic Invocation for CALS/EC*

Takashi Koshida

Abstract

We studied the methods of resolving the issues on Web service execution on the user. The traditional CALS/EC system was a static system like the E-commerce fixed among specific companies. However, from now on, towards worldwide market cultivation or an extension of business opportunities, companies require a dynamic CALS/EC system that enables a flexible commercial transaction according to user request. As new technology of realizing the dynamic CALS/EC system, Web service based on SOAP, UDDI and WSDL technologies is proposed, and the utilization research is advanced. Web service is the general term of the distributed processing software that communicates by the SOAP protocol, using XML as communication data. The technical information on Web service is described in WSDL. The information on the contents and the provider of Web service is registered into a UDDI registry. A user searches a UDDI registry, and detects and performs required Web service. Although Web service is provided with flexibility, such as interoperability based on XML, and platform independence, use of the Web service is not extended in a business field. We believe that this has been due to three problems:

(1) Users cannot easily develop a stub for executing Web services on the user side.
(2) It is difficult for users to find useful and suitable Web services.
(3) Users cannot easily understand the functions of Web services and how to use them.

*Doctoral Dissertation, Department of Bioinformatics and Genomics, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0261012, Feburary 3, 2005.

In this study, we proposed new means and concepts that solve these problems. Furthermore, we developed the Web services that embodied the means and concepts, and implemented a B2B system that loosely coupled those Web services. And we conducted the experiment that applied the system to the actual business process, and verified the effectiveness.

First of all, to verify applicability of Web service in CALS/EC, we developed the Web service for an actual business process with construction of a UDDI registry. Then, we implemented a goods procurement B2B system, applied it to use-case, and confirmed its availability. Next, as the means of solving the problem of (1), for complex output data-type in Web service, we proposed the JavaBeans dynamic generation and analysis technique. Then we developed the Web service dynamic invocation system independent of an output data-type unifying these means. It is a stub-less system that detects Web service from a UDDI registry and is executed dynamically. Next, we proposed the concept of *primitive Web service* as the technique of solving (2) and (3) problems. Based on it, we developed primitive Web services, and agents that cooperate and control them, and implemented as a goods procurement B2B system. The system was applied to the actual use-case, and we confirmed the effectiveness as compared with the system using the conventional Web service.

Keywords:

CALS/EC, Web service, UDDI registry, Business-to-Business system, Dynamic invocation, Primitive Web service

目 次

第 1 章　序論 1
1.1　研究の背景 1
1.2　関連研究と本研究の位置付け 3
1.3　論文の構成 4

第 2 章　Web サービスとその基盤技術 7
2.1　序言 7
2.2　SOAP 8
2.2.1　メッセージ構造 9
2.2.2　実装 10
2.3　UDDI 12
2.3.1　データ構造 12
2.3.2　実装 15
2.4　WSDL 15
2.4.1　要素構造 15
2.4.2　実装 18
2.5　WSIF 18
2.6　結言 19

第 3 章　Web サービスによる B2B システムの試作 21
3.1　序言 21
3.2　商品調達 B２B システム 23
3.2.1　ビジネスモデル 23
3.2.2　システム構成 24

3.2.3 ビジネスプロセス.....25
3.2.4 システムの機能.....27
3.2.5 まとめと今後の課題.....37
3.3 結言.....39

第 4 章 Web サービスの動的実行に関する研究.....41
4.1 序言.....41
4.2 従来手法の問題点.....42
4.3 出力データ型に依存しない自動化された Web サービス動的実行システム.....44
4.3.1 システムの特徴.....44
4.3.2 WSDL 記述からのデータ抽出.....46
4.3.3 システムの機能.....49
4.3.4 実証実験.....53
4.3.5 評価.....60
4.3.6 関連研究.....62
4.4 結言.....64

第 5 章 Web サービスの検出と連携に関する研究.....67
5.1 序言.....67
5.2 従来手法の問題点.....69
5.3 プリミティブ Web サービスの提案.....70
5.3.1 プリミティブ Web サービスの定義.....70
5.3.2 予想される効果.....71
5.4 プリミティブ Web サービスの設計と実装.....72
5.4.1 ビジネスモデルへの適用.....72
5.4.2 ビジネスプロセス.....73

5.4.3　プリミティブ Web サービスの設計 75
5.4.4　エージェントによる制御と連携 78
5.5　実験と評価.....79
5.5.1　実証実験 80
5.5.2　評価 83
5.6　関連研究 84
5.7　結言 85

第 6 章　結論 87
6.1　本研究の成果 87
6.2　今後の課題.....88

謝　辞 **91**

参考文献 **93**

研究業績 **101**

図目次

図 2.1　SOAP メッセージの構造 9
図 2.2　Axis API による SOAP 通信 10
図 2.3　Axis API によるスタブ開発例 11
図 2.4　UDDI レジストリの登録例 14
図 2.5　WSDL 記述の例 17
図 3.1　システム全体図 24
図 3.2　システム構成と処理の流れ 27
図 3.3　発注入力画面 28
図 3.4　発注状況照会画面 29
図 3.5　メーカが提供する Web サービス 30
図 3.6　サービスブローカのメーカ検索画面 35
図 3.7　新規メーカ登録画面 36
図 3.8　UDDI レジストリのインターフェース画面 37
図 4.1　出力が基本型である場合の WSDL 記述の一部 47
図 4.2　出力が複合型である場合の WSDL 記述の一部 48
図 4.3　システム全体の処理の流れ 50
図 4.4　getter メソッドの抽出 51
図 4.5　getter メソッドの動的実行 52
図 4.6　UDDI レジストリに登録された Web サービス 54
図 4.7　システム A の実行結果（出力データ型：基本型） 55
図 4.8　システム B の実行結果（出力データ型：複合型） 56
図 4.9　本システムの実行結果：その 1（出力データ型：基本型） 57
図 4.10　本システムの実行結果：その 2（出力データ型：基本型） 58
図 4.11　本システムの実行結果：その 3（出力データ型：複合型） 59
図 4.12　本システムの実行結果：その 4（出力データ型：複合型） 60
図 5.1　ビジネスモデルにおける処理の流れ 74

図 5.2　メーカにおける Web サービス配備状況 77
図 5.3　信用調査会社における Web サービスの配備状況 78
図 5.4　卸売業者で稼動するエージェント 79
図 5.5　小売業者に対する信用照会実行状況 80
図 5.6　在庫管理 Web サービスの実行 81
図 5.7　メーカに対する在庫管理 Web サービスの実行結果 82
図 5.8　エージェントによる商品選択結果 83

表目次

表 3.1　システムの開発環境 25
表 3.2　受注データベース 30
表 3.3　取引メーカデータベース 30
表 3.4　在庫データベース 32
表 3.5　受発注データベース 32
表 3.6　メーカ情報データベース 34
表 4.1　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 61

第1章　序論

1.1　研究の背景

これまでの CALS/EC システムは，長期の取引関係に基づいた，限られた企業間における固定的なデータ通信が主流であった．これは，通信相手を特定した専用回線などの閉じたネットワーク上でデータ交換を実現するものであり，ある特定の製品受発注やデータ交換など，パターン化された情報交換には有効であった．しかし，インターネットの拡大とコンピュータ技術の進展とともに，多様な消費者ニーズへの対応，グローバルな生産・販売拠点の展開や異分野でのビジネスチャンスの獲得に向けて，より柔軟な企業間関係及び企業対消費者関係を実現するシステム構築の必要性が高まり，HTML データを利用した Java Servlet によるクライアントーサーバ・アーキテクチャを利用した様々な技術提案やシステムが提案された[35,36,37,63]．その Servlet システムの利用が拡大するにつれて，多種多様なビジネス文書，技術文書や技術データへの対応が要求され，タグが固定された HTML では対応できない事態となった．また，各企業間システムのインターネット上での連携も求められたが，Servlet 技術では，各サーバ上の Servlet システムを連携・実行することは不可能だった．

そこで，多様な文書やデータに対応するために，タグ拡張性をもち，柔軟な電子処理が可能な XML が開発され，この XML による通信をベースにした分散処理技術として，SOAP，UDDI，WSDL の標準化技術が開発された．そして，それらを基盤技術とした Web サービスが開発され，そのビジネス分野における実用化研究が進められている．Web サービスとは，通信データとして XML を用い，SOAP プロトコ

ルで通信する分散処理ソフトウェアの総称である. Web サービス実行に関する技術情報は WSDL で記述され，Web サービスの内容や提供者に関する情報は，UDDI レジストリに登録・管理される．ユーザは UDDI レジストリを検索し，必要な Web サービスを検出し実行する利用形態である．Web サービスは，XML ベースの相互運用性やプラットフォーム独立などの柔軟性を備えているが，現在のビジネス分野における Web サービスの利用はあまり広がりをみせていない．その大きな原因は,ユーザ側における Web サービス利用上の問題である，次の 3 点であると考える.

（１）Web サービス実行には，ユーザ側でその Web サービスを駆動するスタブプログラムが必要であり，その開発負荷が大きい.

ユーザが Web サービスを実行するためには，専門知識を必要とする難解かつ複雑なスタブプログラムを全てユーザ側で開発しなければならず，これが Web サービスの普及と動的実行実現を妨げる障害となっている.

（２）ユーザが必要とする，または利用したい Web サービスの検出が難しい.

Web サービスの登録・検索を一元化する技術として UDDI が開発されたが，その検索手段はキーワード検索に限られ，ユーザが必要に応じて短時間で最適な Web サービスを検出することが難しい状況である．更に，登録されている Web サービスが実際には存在していない，など登録情報や更新など信頼性に関する問題もみられる.

（３）ユーザが Web サービスを利用する際に，その機能や入出力インターフェースなどの利用法について短時間で正確に知ることが難しい．

Web サービスの名称や利用法に関する統一規約は存在しないので，その名称，機能や入出力インターフェースは提供企業毎に異なる状況である．従って，ユーザは利用する Web サービス毎に，事前にその利用情報を理解し，修得する必要があり，負荷が大きい．

これらの問題解決が Web サービスの利用拡大に直結する．

1.2　関連研究と本研究の位置付け

上記の 3 課題に関する関連研究について概観するとともに，本研究の位置付けについて述べる．まず，第 1 の課題を解決する手法として，2 章で説明する WSIF[16]が開発された．しかし，Web サービスの出力データ型が複合型の場合には，スタブレスに実行できないという制約があった．そして，その制約を解決するために WSIF に追加されるフレームワークとして JROM[24]が開発されたが，JROM API コードがサーバ側 Web サービスとユーザ側クライアントの両方に必要であるなど大幅なコード変更が要求される．また，第 2 の課題解決に対しては，DAML-S[15]などを利用したセマンテック Web が提案されている[22,33,51]．しかし，これらの手法はセマンテックデータを付加したテストデータに対しては有効であるが，それ以外のセマンテックデータを持たない任意のデータに対しては効果がない．また検出した Web サービスを実行する手法に関しても触れていない．第 3 の課題に対しては，RDF[52]を用いて Web サービスの入出力変数に対する範囲や制約事項を記述する試み[33]がある．

本研究では，Web サービスの標準化技術である UDDI と WSDL をベースにして，3 項目の問題点を解決する新しい手法及び概念を提案

する. 更に, その手法及び概念を具現化した Web サービスとその Web サービスの疎結合として構成される B2B システムを開発し, 実際のビジネスプロセスに適用した実証実験を行い, その有用性を確認している点で関連研究と異なる. まず, 第 1 の課題に対しては, WSIF を利用し, その上で複合型出力データをもつ Web サービスに対しても, スタブレスで実行可能な手法を考案し, 出力データ型に依存しない Web サービス動的実行手法をシステムとして実装した. この手法は, 全て標準の Web サービス開発・実行環境で実現でき, JROM など外部フレームワークを一切必要としない. 更に, 本手法では, 指定した Web サービスを UDDI レジストリから検出し, そのまま連続して動的実行が可能であり, Web サービスの検出とその動的実行をともに実現している. 第 2, 第 3 の課題に対しては, プリミティブ Web サービスという新しい概念を提案し, その概念に基づいた Web サービスを実装し, 具体的なユーズケースに基づいて実験を行い, その効果を検証している. Web サービスに対して一意に統一された規約を設定することで, Web サービスの検出や利用情報の理解を容易にするものである.

これらの新しい手法及び概念を実装することにより, ユーザサイドにおける Web サービス利用技術の向上に寄与することを本研究の目的とする.

1.3 論文の構成

本論文の構成は次の通りである.

第 2 章では, Web サービスの基礎である SOAP, UDDI, WSDL 及び WSIF の各標準化技術について詳述するとともに, 本研究で実装するシステムとの関連について説明する.

第 3 章では, Web サービスの試作とその Web サービスで構成される B2B システムについて述べる. Web サービスの有効性を評価する

ために，実際の商品調達の商取引に沿ったビジネスプロセスを想定して具体的に Web サービスを開発し，B2B システムとして実装した[32,38,39]．この開発・実験を通して，新しい技術である Web サービスの本質を理解するとともに，その問題点を明確にした．その問題点が前述の（１）～（３）である．

第 4 章では，Web サービスの動的実行手法とその実装について論述する．Web サービスの動的実行を実現するために WSIF 要素技術をベースにスタブレスなシステムの研究に取組んだ．これは UDDI レジストリから Web サービスを検出し，動的に実行する手法の提案であり，（１）の問題点を解決する手法である．Web サービスの実行結果をユーザに返す際には，その出力データの型に対応した処理が必要になる．出力データ型は基本型と複合型の 2 種類に大別される．まず基本型に対して，UDDI レジストリから Web サービスを検出し，動的に実行する方法を示した[40,41]．次に，実行時にその型を自動識別し，その型に応じた動的処理を実現し，その有効性を実験で確認した．特に，複合型に対して型名の認識，型オブジェクトの生成，型オブジェクトへのアクセスなどを全て自動かつ動的に実行する手法を考案・実装して，その有効性を検証した[42]．最終的に，出力データ型に依存しない自動化された動的実行システムとして開発した[43,45]．

第 5 章では，（２），（３）の問題点を解決する手法の提案とその実装について述べる．ユーザが利用したい，または必要とする Web サービスの検出容易化と，利用する Web サービスの機能や入出力インターフェースの理解に対するユーザ負荷を軽減する手法として，プリミティブ Web サービスという概念を提案している．プリミティブ Web サービスは，様々なビジネス分野で共通に利用しうる汎用的な Web サービスとして，一意の名称，機能，入出力インターフェースをもつとして定義される．これにより，現在の Web サービスが内包する，名称や機能に対する不特定性，任意性がなくなり，UDDI レジストリのキーワード検索でも，目的とする Web サービスの検出が容易にな

る．また，ユーザも一度，プリミティブ Web サービスの機能と入出力インターフェースを理解すれば良く，利用法の理解と修得に関する負荷が軽減される．更に，プリミティブ Web サービスを連携・制御するエージェントの利用を提案している．プリミティブ Web サービスとエージェントを組み合わせる方法を明らかにし，その有効性を具体的なビジネスプロセスに適用して確認している[44].

最後に第 6 章では，本研究で得られた成果について総括し，今後の課題について述べる．

第2章　Webサービスとその基盤技術

本章では,Web サービスを実現する標準化技術である SOAP, UDDI, WSDL および WSIF に関して，その仕様や内容を説明する．また，それらの実装についても併せて説明する．これらの標準化技術をベースに，本研究のシステム開発が行われている．

2.1　序言

Web サービスとは, インターネット上で XML データと SOAP プロトコルを使って入出力データ交換を行う分散処理ソフトウェアの総称である．XML 言語は，従来の SGML や HTML に代わってユーザ拡張性と柔軟性をもつ言語として開発され，Web サービスはその XML を通信媒体として利用する．RMI や CORBA 規格を用いた従来の分散処理プログラムでは, 同じ規格同士の分散処理に限定されたが, Web サービスではテキストデータである XML によるメッセージ交換と Web 上の標準プロトコルである HTTP による通信が可能であり，また Web サービス自身も通常の Java オブジェクトとして開発できるので, プラットフォームに依存しない任意のシステムで分散処理が実行できる．また，開発された Web サービスをインターネット上で統一して登録・公開する UDDI と，Web サービスの技術情報を記述する WSDL により，ユーザが自由に Web サービスを検索し，単独にまたは相互に結合して実行可能な環境も整いつつある．今後, CALS/EC に代表されるグローバルなビジネス分野においても, Web サービスの活用が予想される．本研究では，従来の CALS/EC システムの中心であった Java Servlet に代わる技術として Web サービスを認識し，その Web サービスを用いた CALS/EC システムについて研究する．

2.2 SOAP

SOAP（Simple Object Access Protocol）[13]は，XML ベースのメッセージ交換により RPC（Remote Procedure Call）のようにリモートマシン上のサービス実行や，オブジェクトへのアクセスを可能にするプラットホーム独立なプロトコルである．SOAP を利用することで，インターネットに存在するサービスの結合や，システム間連携が可能になる．XML メッセージの通信プロトコルは任意であり，HTTP，SMTP，FTP などが利用できるが，通常 HTTP が用いられる．Web サービスを利用するためには，クライアントプログラムが必要であり，その

- クライアントプログラムから，Web サービスが稼動しているサーバに対してサービスの指定や入力パラメータを渡し，
- サーバからそのクライアントプログラムに，そのサービスの処理結果を返す必要がある．

一般にこのクライアントプログラムをスタブといい，Web サービス実行時にユーザが開発しなければならない必須のプログラムである．Web サービスサーバにおいて，Web サービスは，ある機能をもつオブジェクトとして存在している．通常，このオブジェクトは Java クラスとして，機能はメソッドとして提供される．SOAP では，このオブジェクトをスタブから呼び出して，実行する手順を SOAP RPC として決めている．また，Web サービスの処理結果はデータオブジェクトとして生成されるが，それをスタブに返送するために，ネットワーク送信が可能で，共通に認識可能な型に変換する必要がある．その変換規定を SOAP 符号化という．Integer,Float,String などの基本型や文字列型，それらを組み合わせた複合型などが規定されている．

2.2.1　メッセージ構造

クライアントーサーバ間で相互に交換される情報を SOAP メッセージと呼ぶ．SOAP メッセージは，エンベロープ，ヘッダ，ボディから構成される XML データである． 各々について説明する．

（1）エンベロープ

エンベロープにヘッダとボディ要素が包含される．このエンベロープ要素で,送受信される SOAP メッセージのフォーマットを規定する．つまり，利用するサービスとメソッドを規定し，送受信するデータのタイプを記述する．XML 文書を HTTP で送受信を行う場合，名前空間として，
http://schemas.XMLSOAP.org/SOAP/envelope/ が使われる．

（2）ヘッダ

メッセージの処理について記述する．通常は省略される．

（3）ボディ

メッセージの内容を記述する本体部分である．エラー発生時には，Body 要素中に Fault 要素を入れて返送する．

図 2.1 に SOAP メッセージの構造を示す．

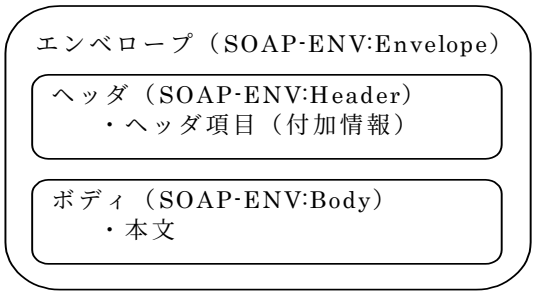

図 2.1　SOAP メッセージの構造

2.2.2　実装

SOAP を使う Web サービスを Java 言語で開発する際に必要になるのが Apache-Axis[3]である．この Apache-Axis は，スタブ開発に必要なライブラリや Web サービスの登録・運用管理ライブラリから構成されるフレームワークである．Apache-Axis は，名前空間を処理できる Xerces XML パーサ[9]と JSP/Servlet が動作する Tomcat[4]などの HTTP サーバを必要とする．Axis エンジンは Tomcat の Servlet として稼働し，Web サービス実行用 Servlet と管理用 Servlet が含まれる．また，開発した Web サービスを Apache-Axis に配備するためには，Web サービス名やメソッド名などを記述した配備ファイルが必要になる．Apache-Axis による SOAP 通信を図 2.2 に示す．

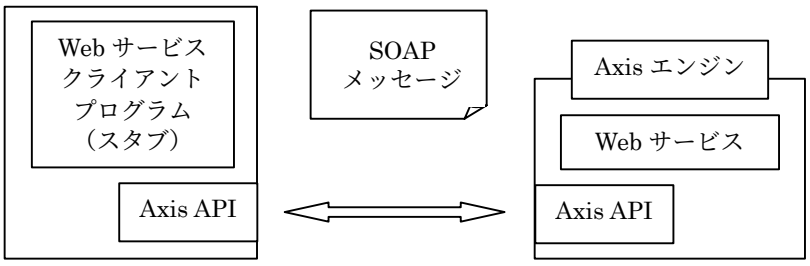

図 2.2　Axis API による SOAP 通信

スタブは，通常 Web サービスを利用する際にクライアント側で必ず開発しなければならないプログラムであり，その中で利用する Web サービス名，メソッド名，アクセスポイント，入力パラメータ値を設定する．本研究ではこの Apache-Axis1.0 を利用して Web サービスサーバを構築している．また，Web サービスを実行するスタブは Axis のライブラリ（API）を利用して開発している．そのスタブ開発例を図 2.3 に示す．

```
import org.apache.axis.client.Call ;
import org.apache.axis.client.Service ;
import org.apache.axis.encoding.ser.BeanDeserializerFactory ;
import org.apache.axis.encoding.ser.BeanSerializerFactory ;
import org.apache.axis.encoding.XMLType ;
import javax.xml.rpc.ParameterMode ;
import javax.xml.namespace.QName ;
import java.sql.* ;
public class GoodsClient7{
   public static void main(String[] args) throws Exception{
      String url ="http://192.168.1.185:8080/axis/services/GoodsService7";
      String qnws="http://192.168.1.185:8080/axis/services";
      String states = "";    SData6 resp       = null;    Object callArgs[];
       if( ( args == null ) || ( args.length != 2 ) ){
          System.err.println("Usage: java GoodsClient7 [beertype] [quatity]" );
          System.exit(1);
       }
      String beertype = args[0];
      int reqval = Integer.parseInt(args[1]);
      QName qName2 = new QName(qnws, "SData6");
      try{
         Service service = new Service();
         Call call = (Call)service.createCall();
         call.setTargetEndpointAddress(url);
         call.setOperationName(new QName("GoodsService7", "getStockdetails7"));
         call.addParameter("input", XMLType.XSD_STRING,    ParameterMode.IN);
         call.addParameter("num",     XMLType.XSD_INTEGER, ParameterMode.IN);
         call.setReturnType(qName2);
         call.registerTypeMapping(SData6.class, qName2,new BeanSerializerFactory(    SData6.class, qName2),
                  new BeanDeserializerFactory(SData6.class, qName2));
         callArgs = new Object[] {beertype,new Integer(args[1])};
         resp = (SData6)call.invoke(callArgs);
         System.out.println("Available maker      : " + resp.getMakername());
         System.out.println("Beer type           : " + beertype );
         System.out.println("Beer code           : " + resp.getBeercode());
         System.out.println("Beer name           : " + resp.getBeername());
         System.out.println("Price(bottle)       : " + resp.getUnitvalue());
         System.out.println("Order(case)         : " + resp.getCasevalue());
         System.out.println("Total Price         : " + resp.getTotalcost());
      }catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

図 2.3　Axis API によるスタブ開発例

2.3 UDDI

UDDI(Universal Description, Discovery and Integration)[62]は，インターネット上で Web サービスに関する様々な情報を登録･公開し，検索する方法を提供する仕様である．その仕様に基づいて，IBM や Microsoft などが実装した UDDI レジストリが一般ユーザに対して公開されている．この UDDI レジストリによって，インターネット上の Web サービスが一元的に登録・管理され，検索可能になる．Web サービスを公開したい企業は，企業情報や Web サービスの技術情報などを規約に従って UDDI レジストリに登録する．一方，Web サービスを利用したいユーザは，目的とする Web サービスの種類や内容をレジストリに対して問い合わせ，そのサービスを検索して利用する．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，全て XML データとして記述される．したがって，UDDI レジストリは，XML 形式である SOAP メッセージを利用してアクセスされる．また，UDDI レジストリ自身も Web サービスとして構成されており，UDDI レジストリへの登録，検索，参照なども SOAP メッセージを用いた Web サービスが利用できる[60].

2.3.1 データ構造

UDDI レジストリのデータ構造について，要素ごとに説明する．

（１）businessEntity

businessEntity 要素は最上位の要素であり，企業組織についての情報（職種や連絡先情報など）を示す．その中に複数の *businessService* 要素を記述できる．登録情報の識別のために，128 ビットのユニークなビジネスキーが付与される．

（２）businessService

企業が提供する Web サービスを登録するためのデータ構造が *businessService* である．ここでは，サービスの名前とその説明を記述する．サービスを識別するために，128 ビットのユニークなサービスキーが付与される．

（３）bindingTemplate

Web サービスに接続するための技術情報である，サービス型へのポインタや接続先 URL などを記述する．サービス型とは，Web サービスを利用するためのインターフェース（Web サービス名，引数の数とその型，戻り値の型など）情報のことである．そのサービス型は *tModelKey* というポインタで記述される．バインディングを識別するために，128 ビットのユニークなバインディングキーが付与される．

（４）tModel

Web サービスの型が記述される．そのサービス型は直接記述されるのではなく，その情報を記述した WSDL ファイルの URL を指定する形式であり，*overviewURL* 要素でその URL を記述する．また *tModel* 要素へのポインタが *tModelKey* というユニークな 128 ビットの数値で表される．

UDDI レジストリの登録例を図 2.4 に示す．

```
<?xml version="1.0" encoding="utf-8" ?>
 <businessDetail generic="2.0" xmlns="urn:uddi-org:api_v2"
       operator="www.ibm.com/services/uddi" truncated="false">
    <businessEntity businessKey="FDFDBBA0-A7D3-11D5-A30A-002035229C64"
          operator="www.ibm.com/services/uddi" authorizedName="1000000FKU">
       <discoveryURLs>
          <discoveryURL useType="businessEntity">
            http://www-3.ibm.com/services/uddi/uddiget?businessKey=FDFDBBA0-A7D3-11D5-A30A-002035229C64
          </discoveryURL>
       </discoveryURLs>
       <name xml:lang="en">IBM WSTK Tutorial</name>
       <description xml:lang="en">IBM WSTK Tutorial</description>
       <businessServices>
          <businessService serviceKey="12F63B90-A7D4-11D5-A30A-002035229C64"
                businessKey="FDFDBBA0-A7D3-11D5-A30A-002035229C64">
            <name xml:lang="en">NasdaqQuotes</name>
            <bindingTemplates>
              <bindingTemplate bindingKey="12FF1530-A7D4-11D5-A30A-002035229C64"
                    serviceKey="12F63B90-A7D4-11D5-A30A-002035229C64">
                <description xml:lang="en" />
                <accessPoint URLType="http">http://localhost:8080/soap/servlet/rpcrouter/
                </accessPoint>
                <tModelInstanceDetails>
                   <tModelInstanceInfo tModelKey="UUID:A0735690-9732-11D5-BC97-002035229C64">
                     <instanceDetails>
                       <overviewDoc>
                         <overviewURL>./ibm_nasdaqquotesclient-impl.wsdl</overviewURL>
                       </overviewDoc>
                       <instanceParms>
                         <port name="NasdaqQuotesPort" binding="NasdaqQuotesBinding"/>
                       </instanceParms>
                     </instanceDetails>
                   </tModelInstanceInfo>
                </tModelInstanceDetails>
              </bindingTemplate>
            </bindingTemplates>
            <categoryBag>
              <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
                    keyName="Education and Training Services" keyValue="86" />
            </categoryBag>
          </businessService>
       </businessServices>
       <categoryBag>
          <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
                keyName="Education and Training Services" keyValue="86" />
       </categoryBag>
    </businessEntity>
 </businessDetail>
```

図 2.4　UDDI レジストリの登録例

2.3.2　実装

この UDDI レジストリに対するアクセスプログラムを開発する際には，UDDI4J[27]を利用する．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り，本研究でも，この UDDI4J を利用してプログラムを開発している．

2.4　WSDL

WSDL(Web Services Description Language)[14]は Web サービスのインターフェースや接続情報を記述する言語仕様であり，その具体的な記述内容として，Web サービス実行に必要な技術情報である 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[60].

2.4.1　要素構造[12]

WSDL では *definitions* 要素をルート要素として文書内容を記述する．*definitions* 要素の属性にはこの文書の名前とこの文書中で使用する名前空間を定義する．以下にこの *definitions* 要素中で指定される５種類のデータ要素について説明する．

（１）types

types 要素はメッセージ交換に使う Web サービスの入出力データ型を定義する．通常，基本型に対しては，XML Schema の定義が自動的に適用されるので，ここで定義する必要があるのは，複合型だけである．

（２）message

message 要素は入出力データの抽象定義を行う．各 *message* 要素

の *part* 要素で, Web サービスメソッドへの入力データ名とその型, または出力データ名とその型を定義する．各 *message* 要素には, *name* 属性を使って，WSDL 文書内で一意の名前を宣言する．

（３） portType

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．子要素の *operation* 要素で公開する Web サービスメソッドを定義する. Java で Web サービスを構築する場合, *portType* 要素には１つの Java クラスかインターフェースを定義し，*operation* 要素にはそのメソッドとそれに対する入力引数や戻り値を定義する．

（４） binding

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．*binding* 要素の type 属性値に *portType* 要素の *name* 属性値が指定される．*input* 要素と *output* 要素では，メッセージ転送に SOAP body を使うことを宣言する．

（５） service

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．

図 2.5 に WSDL 記述の具体例を示す．

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://192.168.1.185:8080/axis/services/GoodsService7"
   xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://192.168.1.185:8080/axis/services/GoodsService7"
    xmlns:intf="http://192.168.1.185:8080/axis/services/GoodsService7"
    xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
   xmlns:tns1="http://192.168.1.185:8080/axis/services"
   xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema   targetNamespace=http://192.168.1.185:8080/axis/services
              xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <complexType name="SData6">
        <sequence>
           <element name="beercode" nillable="true" type="xsd:string"/>
           <element name="beername" nillable="true" type="xsd:string"/>
         </sequence>
      </complexType> <element name="SData6" nillable="true" type="tns1:SData6"/>
    </schema>  </wsdl:types>
  <wsdl:message name="getStockdetails7Request">
    <wsdl:part name="in0" type="xsd:string"/> <wsdl:part name="in1" type="xsd:int"/>
  </wsdl:message>
  <wsdl:message name="getStockdetails7Response">
    <wsdl:part name="getStockdetails7Return" type="tns1:SData6"/>
  </wsdl:message>
  <wsdl:portType name="GoodsService7">
    <wsdl:operation name="getStockdetails7" parameterOrder="in0 in1">
      <wsdl:input  message="intf:getStockdetails7Request"  name="getStockdetails7Request" />
      <wsdl:output message="intf:getStockdetails7Response" name="getStockdetails7Response"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="GoodsService7SoapBinding" type="intf:GoodsService7">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getStockdetails7"> <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getStockdetails7Request">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="http://192.168.1.185:8080/axis/services/GoodsService7" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getStockdetails7Response">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="http://192.168.1.185:8080/axis/services/GoodsService7" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="GoodsService7Service">
    <wsdl:port binding="intf:GoodsService7SoapBinding" name="GoodsService7">
      <wsdlsoap:address location="http://192.168.1.185:8080/axis/services/GoodsService7"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

図 2.5　WSDL 記述の例

2.4.2　実装

WSDL 記述内容をソフトウェアで操作するためには，Java のクラスライブラリである WSDL4J[29]を利用する．本研究においても，Web サービスの Dynamic Invocation を実現するために，WSDL 記述における *types*，*message*，*portType* 要素データを解析し，必要データを自動抽出するシステム開発に WSDL4J を用いている．

2.5　WSIF

WSIF(Web Services Invocation Framework)[16]は，

・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，及び

・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．従って WSIF を利用することで，これまで Apache-Axis API 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 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ので，開発者はバインディングを意識しなくても良い[47]．また，JavaBeans に対するシリアライズ／デシリアライズの API も標準で用意されているので，複合型（複数の基本型から構成される型）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操作することができる．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なわれている[6].

2.6 結言

本章では,Web サービスを実現する基礎技術である SOAP, UDDI, WSDL と 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 について説明した.

従来の Java Servlet 技術によるクライアント・サーバ型の CALS/EC システムでは，サービスに関する情報を一元化してユーザに公開する仕組みが存在しなかったため，ユーザはサーチエンジンなどを利用して，個々にサービス提供元の HTML データを確認して，システムを利用する以外に手段がなかった．また，インターネット上の個々の CALS/EC システムを相互に連携することも不可能であった.しかし，Web サービスと UDDI レジストリの出現により，グローバルに Web サービスが検索できるようになり，それらの相互連携も可能になっている.Web サービスから構成される CALS/EC システムは，Servlet 技術では不可能であった XML による任意のデータ交換とサービス選択の自由度を可能にする.

実際のビジネス分野における Web サービスの適用性を確認するためには，Web サービスを実際に開発し，それらを CALS/EC システムとして実装して，ビジネスプロセスに適用する実証実験が必要であると考える．第 3 章では，その Web サービスを用いた CALS/EC システムに関する研究と試作について述べる.

第3章　WebサービスによるB2Bシステムの試作

3.1　序言

これまでの企業間システムは，長期の取引関係に基づいた，限られた企業間における固定的なデータ通信が主流であった．これは，通信相手を特定した専用回線などの閉じたネットワーク上でデータ交換を実現するものであり，ある特定の製品受発注やデータ交換など，パターン化された情報交換には有効であった．しかし，インターネットの拡大とコンピュータ技術の進展とともに，多様な消費者ニーズへの対応，グローバルな生産・販売拠点の展開や異分野でのビジネスチャンスの獲得に向けて，より柔軟な企業間関係及び企業対消費者関係を実現するシステム構築の必要性が高まり，様々な標準化提案やシステム提案がなされてきた．

例えば,データの標準化に関しては CAD/CAM データに対するデータ標準化の提案[35]や電子部品や半導体に対するデータ標準化の提案[61]があり，情報共有化に関してはデータに対するアプリケーション開発事例[36]や標準データの情報共有化の試み[10,37]などがなされてきた．また，商品調達と受発注の業務サイクルに対する企業間 E コマースシステムも提案されている[23,34].

これらのシステムの多くは，データとして HTML を利用し，Java Servlet 技術で開発された各企業独自のシステムとして提供されていた．そのため，システムの利用が増えるに従って，多種多様なビジネス文書・技術文書の電子交換への対応に対する限界が明らかになってきた．また，各企業間システムのインターネット上での相互連携も求められたが，各サーバの Servlet システムに対するクライアント・サ

ーバアーキテクチャによる密結合では，その相互連携と実行は技術的に不可能であり，さらに柔軟な分散処理システム技術が求められていた．そこで多様な文書やデータに対応するために，タグの拡張性をもち，柔軟な電子処理が可能な XML が開発され，この XML による通信をベースにした分散処理技術として，SOAP，UDDI，WSDL の標準化技術が開発された．それらを基にした Web 上の分散処理アプリケーションが Web サービスである．

第 2 章で述べたように，これらの標準化技術を用いて Web サービスが開発可能な環境が実現されつつある．これらの標準化技術は，ひとつひとつがかなり大きな仕様であり，その全体を明確に理解し捉えることは，単に仕様文書を読むだけでは困難である．また，それらの技術は相互に関連し，それが Web サービスとして実際に開発され，ユーザに公開されて結実するものであり，実行可能なシステムとしての開発と実装が必要である．第 3 章では，この Web サービスをベースにした CALS/EC システムについての研究と試作について述べる．

Web サービスに対する理解やその課題，及びビジネス分野での適用性などを判断するためには，Web サービスを実際に開発し，それらを実際のビジネスプロセスに沿って実験検証することが必要不可欠である．しかし，実際のビジネスプロセスに対して，Web サービスを設計・実装し，UDDI レジストリと連携して処理を行う，まとまったシステムを構築し，実証実験を通して検証した報告は少ない．その観点から，第 3 章では，CALS/EC システムとして，具体的に商品調達のビジネスプロセスを設定し，実際に Web サービスを開発して，UDDI レジストリと連携して処理を行う商品調達 B2B システムを構築し，そのビジネスプロセスに沿った実証実験を行い，Web サービスの有効性を検証する．さらに，問題点や解決すべき研究課題を明確にする．

3.2 商品調達B２Bシステム

Web サービスの実用化に対する有効性と UDDI レジストリへのアクセス処理を検証するために，具体的なビジネスモデルを想定し，そのビジネスプロセスに対する全ての処理を Web サービスと UDDI レジストリとの連携で実行する商品調達 B2B システムを開発する．

以下，その内容について詳述する．

3.2.1 ビジネスモデル

商品調達に関する動的な電子商取引を想定したビジネスモデルを考える．このビジネスモデルは，食料品を扱う卸売業者を中心として，小売業者，その卸売業者と日常的に取引を行っている通常メーカ，さらに小売業者からの受注に対する欠品発生時に卸売業者と新規の取引を行うテンポラリメーカで構成される．具体的には，卸売業者は小売業者を顧客として，複数のメーカの様々な商品に対する注文を一元的に受付け，顧客の指定する日時，指定する店舗に納品する業務を行う．特に，受注商品に対する納品率をできるだけ 100%に近づけることが重要である．小売業者からの新規商品の受注や既存商品の在庫切れに対して，UDDI レジストリを利用して，同じ仕様の商品を提供するメーカを動的に検索し，発注することが要求される[48]．小売業者への納品回答の高速化と納品処理状況がいつでも確認できることが必要である．このモデルでは，UDDI レジストリ検索を補助する機構として，メーカに関する情報を提供するサービスブローカを設定する．Web サービスのアクセスポイントは UDDI レジストリで管理され，メーカに関する情報は UDDI レジストリとそれを補完するサービスブローカで提供される．そのシステム全体図を図 3.1 に示す．

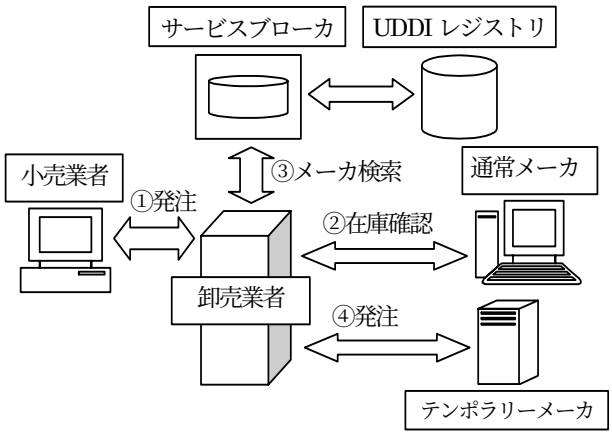

図 3.1　システム全体図

3.2.2　システム構成

この商品調達 B2B システムは，Web サービスを提供する 4 サーバ（卸売業者，通常メーカ，テンポラリメーカ，サービスブローカ）と UDDI レジストリ，及びクライアントである小売業者から構成される．そのシステム構成の詳細を図 3.2 に示す．

卸売業者は小売業者に対する Web サービスとして，商品受注サービスとその注文に対するメーカへの商品発注状況を知らせる発注状況照会サービスを提供する．また，メーカは卸売業者と日常的に取引がある通常メーカ，ならびに欠品商品を供給するテンポラリメーカの 2 種類のメーカを想定する．ともに，同じ在庫・価格見積り Web サービスと商品受注 Web サービスを提供すると想定する．卸売業者において欠品が発生した場合は，UDDI レジストリからその欠品商品を提供できるメーカを探し，そのメーカに在庫・価格見積りを行い，その後欠品商品を発注する．その UDDI レジストリからのメーカ検索を補助する機構としてサービスブローカを想定する．そのサービスブローカは，卸売業者に対してメーカ検索 Web サービスとメーカ情報 Web サービスを提供し，その情報を基に卸売業者は UDDI レジストリから

テンポラリメーカを検索する．本システムでは，パブリックな UDDI レジストリに対する利用技術の修得と確立のために，ローカルな環境で動作するプライベートな UDDI レジストリとして SOAPUDDI-0.3.1[57]を使用している．また，Web サービスを実行するためにはスタブが必要であるが，本システムではそのスタブは全て JavaServerPage(JSP)で作成し，GUI にカプセル化している．これら Web サービスサーバ，ならびに Web サービスクライアントの開発環境をまとめて表 3.1 に示す．

表 3.1　システムの開発環境

フレームワーク	バージョン	内　容
Apache-Axis	1.0	SOAP エンジン
Tomcat	4.0.6	Java Servlet/JSP エンジン
SOAPUDDI	0.3.1	UDDI レジストリ
UDDI4J	2.0	UDDI アクセスライブラリ
Java2 SDK	1.4.0_01	Java パッケージ

3.2.3　ビジネスプロセス

この商品調査の商取引全体のビジネスプロセスについて説明する．その処理の流れを図 3.2 に示す．

（１）商品納品要求：

小売業者は卸売業者提供の受注 Web サービスを利用して，卸売業者に商品を発注する．

（２）在庫確認と価格見積り：

卸売業者は，小売業者からの商品発注を受けて，通常の取扱商品ならば，通常取引メーカの提供する在庫･価格見積り Web サービスを利用して，在庫確認・価格見積りを行う．通常メーカは在庫と商品価格を卸売業者に回答する．

（３）通常メーカへの発注：

必要な在庫数と適正価格が確認されれば，卸売業者は通常取引メーカが提供する受注 Web サービスを利用して発注処理を行い，メーカからの発注回答を受信後に小売業者に対して商品受注回答を行う．

（４）欠品もしくは新規商品受注の場合：

もし，小売業者からの商品発注が新規商品の場合，または通常取引メーカに在庫がなく，欠品が発生した場合には，その商品が供給可能なメーカを Web サービスブローカの提供するメーカ検索 Web サービスを利用して検索し，メーカ情報サービスでそのメーカに関する詳細情報を取得する．このサービスブローカは UDDI レジストリと連携して処理を行う．

（５）テンポラリメーカへの在庫確認と発注：

検索されたメーカに対して，在庫確認と価格見積りを行い，その確認後，メーカに対して商品発注を行う．全ての欠品商品に対して，この処理を繰り返し，小売業者から受注した全商品に対してメーカへの発注を完了する．この後，小売業者に対して，受注完了を知らせる．

（６）発注状況の確認：

小売業者は，卸売業者が提供する発注状況照会 Web サービスを

利用して適宜，商品の発注状況を確認することができる．

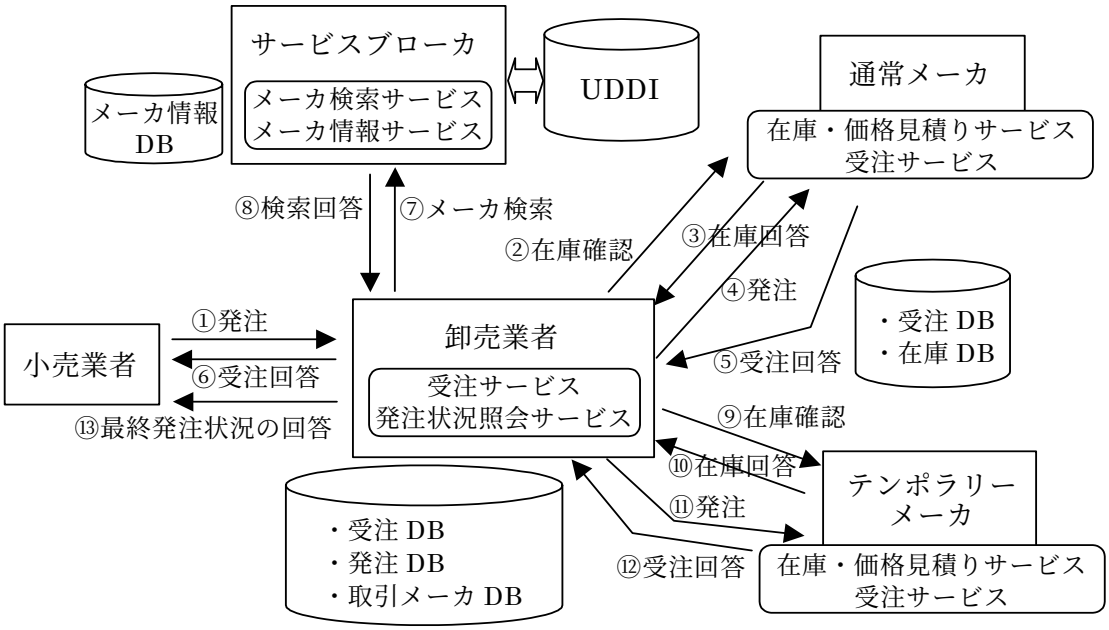

図 3.2　システム構成と処理の流れ

3.2.4　　システムの機能

ここでは，各サーバにおける Web サービスの機能について詳述する．

（１）卸売業者が提供する Web サービス

卸売業者は本商品調達システムの中心であり，小売業者に対して受注 Web サービスと発注状況照会 Web サービスを提供する．

a) 受注 Web サービス

小売業者に対する商品受注サービスを提供する.小売業者は商品 ID, 商品名，納品希望日，数量を入力する．ここで指定された商品情報は，そのままメーカに対する在庫・価格見積り Web サービスの入力データとして利用される．この入力画面を図 3.3 に示す.

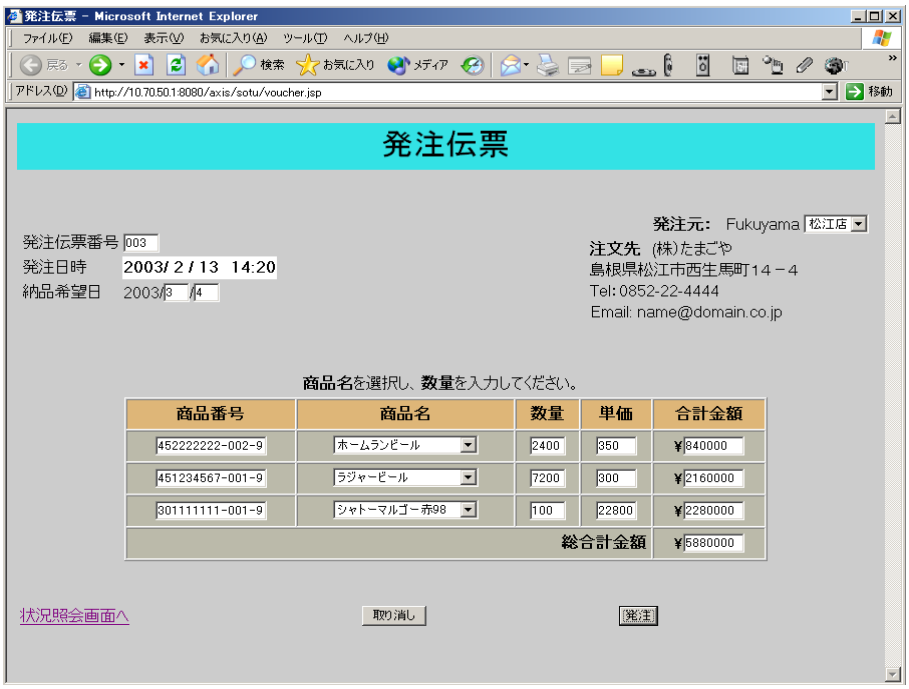

図 3.3　発注入力画面

b) 発注状況照会 Web サービス

小売業者に対して，発注後の商品状況知らせる Web サービスである．卸売業者における商品発注状況，商品欠品状況などを小売業者に知らせる．商品発注伝票 ID を入力すれば，アクセスした時点での商品発注状況が，商品毎に「発注完了」,「在庫分のみ発注」,「未発注」の 3 区分で各々「○」,「△」,「×」記号で表示される．この表示画面を図 3.4 に示す.

図 3.4　発注状況照会画面

c)　データベースについて

・ 受注データベース

小売業者からの商品発注情報を管理する Retailer テーブルと，商品毎の在庫を管理する Order テーブルをもつ．それら各テーブルの内容を表 3.2 に示す．

・ 取引メーカデータベース

商品とそのメーカに関するデータベースであり，Maker テーブルを持つ．日常的に商取引があるメーカに対して，その商品に対する在庫・価格見積りを効率良く実施するために，各 Web サービスのアクセス情報を格納している．その Maker テーブルの内容を表 3.3 に示す．

表 3.2　受注データベース

テーブル名：Retailer				
voucherNo	deliveryDay	deliveryPlace	orderedTime	retailer
伝票番号	納品希望日	納品場所	発注日時	小売業者名

テーブル名：Order				
voucherNo	JAN	demand	maker	stocks
伝票番号	商品コード	要求数量	メーカ名	メーカの在庫数

表 3.3　取引メーカデータベース

テーブル名：Maker			
JAN	product	maker	accessPoint
商品コード	商品名	メーカ名	アクセスポイント

（２）メーカが提供する Web サービス

メーカは卸売業者に対して在庫・価格見積り Web サービス(PriceAndAvailable)と受注 Web サービス(RequestOrder)を提供する．それらサービスの配備状況を図 3.5 に示す．

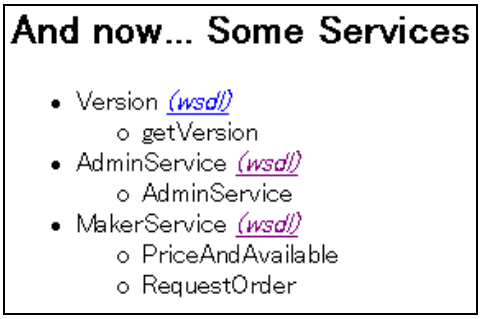

図 3.5　メーカが提供する Web サービス

a) 商品在庫・価格見積り Web サービス：PriceAndAvailable

メーカは商品の在庫状況と商品価格を卸売業者に回答する．卸売業者は，自身の卸売業者 ID，商品名，発注数量，希望納品日などを入力し，商品価格，発注数量に対する対応数量などを出力する．この入力データは小売業者が卸売業者に対して発注したデータを用いて，卸売業者によって入力される．

b) 商品受注 Web サービス：RequestOrder

卸売業者がメーカに対して発注を行う．商品名，発注数量，希望納品日など入力する．メーカにて受注が完了すると，メーカの在庫データベース(product テーブル),受発注データベース(TimeAndPlace, order テーブル)が更新されるとともに，受注完了メッセージを出力データとして卸売業者に返答する．この入力データは a)の Web サービスの出力データを用いる．

c) データベースについて

・在庫データベース

在庫データベースは Product テーブルをもち，卸売業者からの商品在庫・価格見積りに対して利用される．また，商品受注 Web サービス:RequestOrder によって卸売業者から受注を受けると，Product テーブルの在庫数が更新される．在庫データベースの内容を表 3.4 に示す．

・受発注データベース

受発注データベースは Order テーブルと TimeAndPlace テーブルをもつ．商品受注 Web サービス(RequestOrder)によって卸売業者から商品を受注すると，TimeAndPlace テーブルに卸売業者からの商品発注の詳細を記録するとともに， Order テーブルに伝票番号を主キーとして発注した全ての商品内容を記録する．受発注

データベースの内容を表 3.5 に示す.

表 3.4　在庫データベース

テーブル名：Product			
JAN	product	stocks	price
商品コード	商品名	在庫数	単価

表 3.5　受発注データベース

テーブル名：TimeAndPlace						
voucher No	receiveTime	appointedDay	retailer	place	ID	wholesaler
伝票番号	受注日時	納品予定期日	小売業者名	納品店名	卸売業者 ID	卸売業者名

テーブル名：Order				
voucherNo	JAN	product	request	totalPrice
伝票番号	商品コード	商品名	要求数量	合計金額

（3）サービスブローカが提供する Web サービス

サービスブローカは UDDI レジストリと連携して,卸売業者に対してメーカ情報を提供し，メーカ検索を補助する．提供する Web サービスは，メーカ検索 Web サービス(findMakerByProductCode)とメーカ情報提供 Web サービス(getMakerDetail)の 2 種類である.

a) メーカ検索 Web サービス：findMakerByProductCode

卸売業者に対する Web サービスであり, 4 桁数字で指定される商品分類コードを入力として, そのコードに属する商品取引メーカ名, UDDI レジストリに登録されているビジネスキー, メーカの Web サービスへのアクセスポイントを回答する. 入力された商品分類コードを使用して, メーカ情報データベースからビジネスキーとそのメーカ名を検索する. その検索されたビジネスキーを基に, UDDI レジストリからメーカの Web サービスアクセスポイントを取得する. 検索された全てのメーカ名を出力する.

b) メーカ情報 Web サービス：getMakerDetail

卸売業者に対する Web サービスであり, ビジネスキーを入力として, UDDI レジストリと連携してメーカの詳細情報を出力する. 指定されたビジネスキーを使用して, メーカ情報データベースからメーカ名, 資本金, 主要株主, 事業内容, 主要取引銀行, 主要取引先, 主要仕入先, 取扱商品などの詳細情報を抽出する. また, UDDI レジストリからはメーカの電話番号, E-mail アドレス, 住所を抽出して, 卸売業者に回答する.

c) データベースについて

サービスブローカは商品供給メーカの詳細情報を格納するメーカ情報データベースをもつ. MakerInfo テーブルは, UDDI レジストリに格納することできない財務情報や詳細業務情報などのメーカ詳細情報を管理する. また, product テーブルは商品分類コードと商品分類名との対応表である. メーカ情報データベースを表 3.6 に示す.

表 3.6　メーカ情報データベース

テーブル名：Product	
productCode	productName
商品分類コード	商品分類名

テーブル名：MakerInfo						
businessKey	makerName	capital	stockHolder	contents	mainBank	customer
ビジネスキー	メーカ名	資本金	主要株主	業務内容	主要取引銀行	主要取引先

テーブル名：MakerInfo		
supplier	productCode	products
主要仕入先	商品分類コード	取扱い商品名

d)　サービスブローカのホームページ

サービスブローカのメーカ検索画面を図 3.6 に示す．ここでは，メーカの検索を商品分類，メーカ名，ビジネスキーのいずれかで行うことができる．検索が完了するとメーカ名が表示され，そのメーカ名をクリックすると詳細情報が表示される．

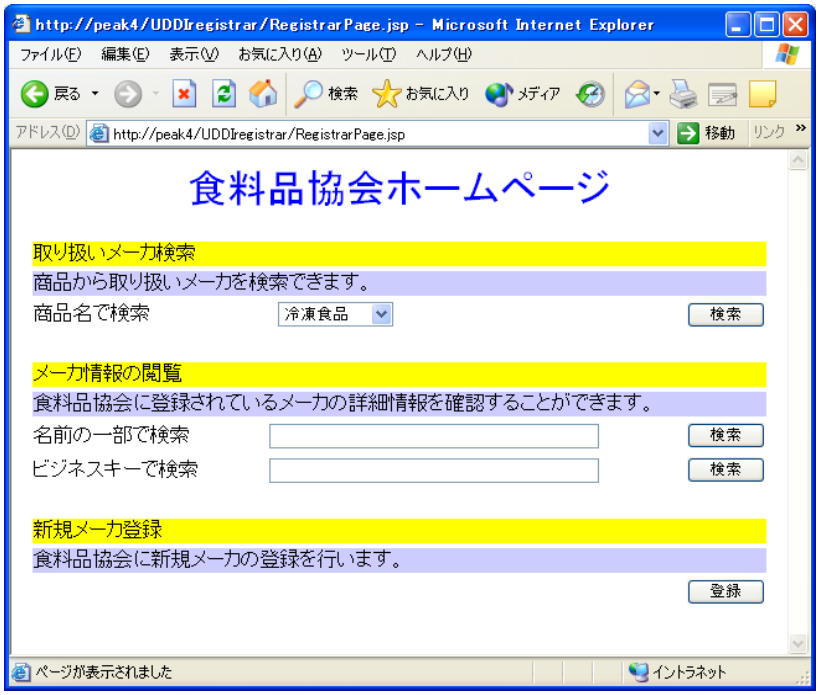

図 3.6　サービスブローカのメーカ検索画面

また，UDDI レジストリに Web サービスを登録したメーカのみが，サービスブローカに登録できるように登録データの一貫性を保証するシステムとなっている．UDDI レジストリ登録時に付与されるビジネスキーを入力して登録する仕組みであり，指定したビジネスキーが UDDI に存在しなければ登録はできない．その登録画面を図 3.7 に示す．

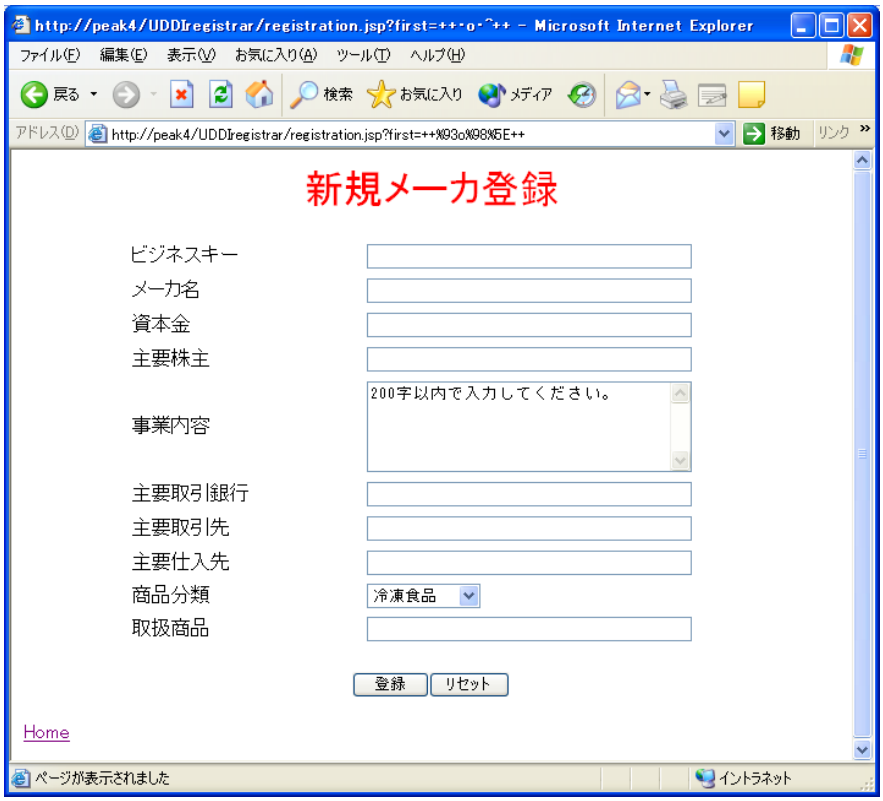

図 3.7　新規メーカ登録画面

（4）UDDI レジストリ

UDDI レジストリのインターフェース画面を図 3.8 に示す．SOAPUDDI へのユーザ登録や内容確認が行える．オリジナルの SOAPUDDI に対する修正や日本語対応などの改良を行った．その修正内容を以下にまとめて記す．

- 全てのインターフェースにおいて，英語表示から日本語表示に修正した．
- ビジネスリストを選択し，表示されたビジネス名のリンクからそのビジネスに関連した Web サービスが表示されるように修正した．
- *bindingTemplate* 情報を表示するように新規にプログラムを開

発した．

- *serviceKey* を与えると *bindingTemplate* を表示するように修正した．
- *tModel* リストを表示するように修正した．

図 3.8 は，上記修正後のインターフェース画面である．

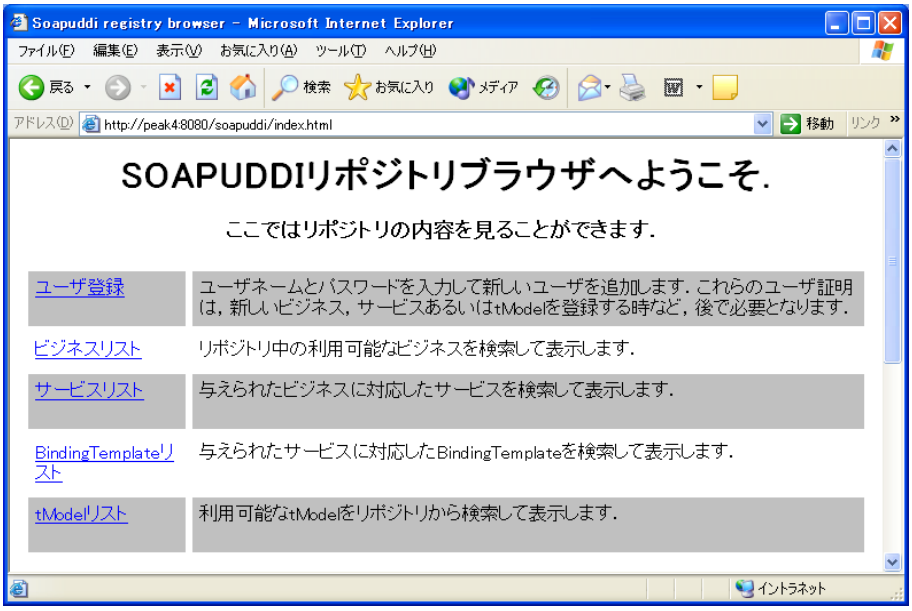

図 3.8　UDDI レジストリのインターフェース画面

3.2.5　まとめと今後の課題

3.2.3 節で想定したビジネスプロセス（１）～（６）に沿って，本システムの動作検証を行い，開発した全ての Web サービスの動作を検証した．その結果，全て設計通り動作し，Web サービスサーバとクライアント間の SOAP メッセージ交換も正常に実行した．このビジネスプロセスの中でのポイントである商品欠品時のメーカ検索に際してのサービスブローカと UDDI レジストリ間の連携動作，ならびに UDDI レジストリの検索処理も正常に動作した．

本研究におけるシステムは，研究室内に SOAPUDDI を用いたローカルな UDDI レジストリを構築し，Web サービスを配備した仮想的な B2B システムであったが，そのビジネスプロセスはセキュリティ面を除けば，ほぼ実際の商取引のビジネスプロセスである．そして，そのビジネスプロセスに対して，想定した小売業者－卸売業者－サービスブローカ－UDDI レジストリ－メーカ間の Web サービスを通しての通信と処理は全て正常に動作した．従って，この研究開発を通して，SOAP，UDDI，WSDL を基本とする Web サービス技術は，これからのインターネット上の分散処理技術として，ビジネス分野に適用可能であることを確認した．

今後の課題として，今回使用した UDDI レジストリは SOAPUDDI によるローカルなテスト用レジストリであり，一部バクもあるなど機能的に不十分なものであった．既に，IBM や Microsoft が公開しているパブリックな UDDI レジストリが存在するので，今後はビジネス分野で利用される Web サービスや B2B システムに対して，このパブリック UDDI レジストリをベースにしたシステムとして研究開発する必要がある．また今回のシステムでは，実行する Web サービスのアクセスポイントやその利用法について，予め既知であるとした．しかし，Web サービス実行に関する情報はパブリックな UDDI レジストリと WSDL 記述に全て一元的に格納されているので，必要になった時点でそれらから Web サービスを検出し，実行する動的なシステムとして実現することが求められる．そのためには，ユーザは UDDI レジストリにアクセスして，必要とする Web サービスを検出し，その WSDL 記述を読み，必要な情報を的確に抽出し，処理しなければならない．これらの

（１）UDDI レジストリからの Web サービス検出と連動した Web サービスの動的実行，

（２）その Web サービスの的確な検出とその機能や入出力インターフェースを正確に理解する手法の実現

に対する解決手法を第 4 章，第 5 章で議論する．

3.3 結言

本章では，Web サービスを用いた CALS/EC システムを研究開発し，実証実験を通して，Web サービスの有効性を確認した．

実際のビジネスモデルを想定し，そのビジネスプロセスを処理する Web サービスを開発するとともに，UDDI レジストリと連携した商品調達 B2B システムとして構築・適用して，Web サービスならびに UDDI の有効性を確認した．これまでは，CALS/EC システムやその提供企業に関する情報をインターネット上で一元的に管理・提供する仕組みが存在しなかったため，Web 上の HTML で提供される情報を目視で確認し，利用するしかなかった．そのため，Servlet 技術を中心とした，従来の CALS/EC システムは単独のクライアントーサーバシステムとして提供され，システム間の相互連携は不可能であった．それに対して，XML 技術を元にした SOAP や UDDI が提案され，その仕様を実装した UDDI レジストリにより，インターネット上でソフトウェアとその提供元情報を一元的に管理・運用する仕組みが実現され，ソフトウェアが Web サービスという単位で自由に実行可能になった．その UDDI レジストリからの Web サービスの検出と実行に関して，前節で述べた課題に関する解決手法を第 4 章と第 5 章で議論する．

第4章　Webサービスの動的実行に関する研究

4.1　序言

第 3 章で説明した商品調達 B2B システムは，予め利用する Web サービスのアクセスポイントや利用法が指定され，かつスタブが既に用意されている状況における，いわゆる静的なシステムである．しかし，多様化し，かつ変化が激しい消費者ニーズへの対応や新規の Web サービスに対応するためには，その変化に追随できる柔軟なシステムが要求される．インターネット上の各サーバに分散されている Web サービスに関する情報やその実行に関する技術情報を記述した WSDL データも，UDDI レジストリによって一元的に登録され，管理されている．従って，必要かつ最新の Web サービスは，その都度 UDDI レジストリから動的に検出され，実行できる柔軟なシステム形態が望まれる．Web サービスの本来の目的は，ユーザが必要な Web サービスを必要な時点で UDDI レジストリから検索し，そのまま実行できるサービス形態の実現であり，これは一般的に動的な実行形態といわれている．本章ではこの動的実行を実現する手法について提案する．

第 2 章 2.2 節に述べたように，Web サービスを実行する際には，ユーザ側で実行する Web サービス毎にスタブを開発する必要がある．このスタブ開発には Apache-Axis クラスライブラリに関する専門知識や Java 言語及び XML の専門知識が必要である．そして，利用する Web サービス毎にスタブ開発が必要な状況は，ユーザにとって負荷が大きく，Web サービスの利用を妨げる大きな要因の一つでもある．Web サービスの動的実行（Dynamic Invocation）とは，このスタブを用いることなく,Web サービスを実行する形態である．この動的実

行が実現されれば, スタブ開発が不要になり, Web サービス実行に要する労力やユーザ負荷が大幅に削減される. B2B システムに代表される CALS/EC システムにおいても, このスタブ開発が不要な Web サービスの動的実行の実現は, ビジネス処理の高速化と使いやすさを確立するために必要不可欠であると考える. このスタブを利用せずに Web サービス実行が可能なフレームワークとして, WSDL 記述を利用する WSIF が開発されたが, 機能的に不十分であり, Web サービスの出力データ型によってその動的実行が制限されている. UDDI レジストリからユーザ要求に合致する Web サービス検出に関する研究は, オントロジを利用した研究［33,51,59］があり, Web サービスの動的実行に関する研究は JROM[24]があるが, Web サービスの検出と連携した動的実行に関してはほとんど研究されていない.

本章では, この WSIF をベースにして, その機能面を改善した Web サービスの動的実行手法について提案する. そして, その手法に基づき UDDI レジストリから Web サービスとその WSDL 記述を検出・解析して, スタブレスに Web サービスを実行するシステムを実装し, 実証実験を通して, その手法の効果と有効性を論ずる. 以下, 4.2 節で WSIF を用いた従来手法の問題点について説明する. 次に, 4.3 節でその問題を解決する自動化手法と動的実行手法示し, その手法を実装したシステムについて詳述する. そして, 本システムと従来システムと比較した実証実験について説明し, 本システムの有効性を示す. 最後に, 4.4 節でまとめと考察を述べる.

4.2 従来手法の問題点

ここで, WSIF を用いた従来手法の問題点について説明する.
Web サービスの出力データ型は, 次の 2 種類,

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

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，例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．従来の方法では，以下の理由のため，これら 2 種類の型を統合して処理することができず，型ごとに別々の実行方式が存在していた．例えば，システム A[7]は WSIF への入力データとして WSDL ファイルの URL を直接記述し，かつ出力データ型が文字型などの基本型 Web サービスにのみ限定された動的実行システムである．システム B[8]は出力データ型が複合型である Zip2Gio[64] Web サービス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，更にその出力データ型に対応した JavaBeans クラスがユーザ側に予め存在している場合にのみ実行可能なシステムである．これらのシステムは WSIF への入力データである Web サービスの WSDL URL を常に事前に把握し設定する必要があること，更に出力が複合型の Web サービスに対してはその Web サービス毎に複合型データに対応した JavaBeans クラスを常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．

基本型データの場合は，その XML スキーマと対応する Java の基本型が存在するために，そ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．従って，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，実行前に出力データ型を識別し，その型別に 2 種類の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（例えば, 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，ユーザは WSDL の構造と WSIF API に関する専門知識を要求される，など負荷が大きく，利用しにくい状況であった.

4.3 出力データ型に依存しない自動化されたWebサービス動的実行システム

前述の問題を解決し，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 サービスが利用できるための自動化手法と動的実行手法を提案する．まず基本型に対する自動化を実現し[40,41], それを基に複合型に対する自動化と JavaBeans の動的生成を実現した[42]. 最終的に，それらを統合し，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を提案している[43,45].

本システムは, UDDI レジストリの *tModel* 要素から Web サービスを検索し，その WSDL 記述を入手・解析して，Web サービスの出力データ型に依存せずにスタブレスに Web サービスを実行するシステムである.

4.3.1 システムの特徴

本システムの特徴は，次の 2 点である.

（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

以下，それらについて説明する．

（１）設定処理の自動化

これまで実行する Web サービス毎に以下の 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全て手動で行っていた．本研究ではこの 3 工程を全て自動化した．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，全て自動化した．まず，基本型に対する自動化システムを開発し[40,41], 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）出力データ型に依存しない Web サービスの動的実行

従来, Web サービスの出力データ型毎に動的実行システムを使い分けており, 常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考案し, 実装する[42]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[43,45]と

して実現した.

4.3.2 WSDL 記述からのデータ抽出

Web サービスの技術情報を記述する WSDL の出力データ記述部分について,基本型と複合型の WSDL 記述内容を具体的に示すとともに,WSIF API に設定するデータ項目の抽出について説明する.

（1）出力データ：基本型

出力データが基本型である場合の WSDL 記述の一部を図 4.1 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.

```
<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 4.1　出力が基本型である場合の WSDL 記述の一部

（2）出力データ：複合型

出力データが複合型の WSDL 記述の一部を図 4.2 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．

```
<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 4.2　出力が複合型である場合の WSDL 記述の一部

4.3.3　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 4.3 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 ”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.

・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. 何故なら, 複数の Web サービスが同じ *tModel* 名を持つ場合があるので（図 4.9), その特定のためには Web サービス毎に固有な値をもつ *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, 本研究では *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.

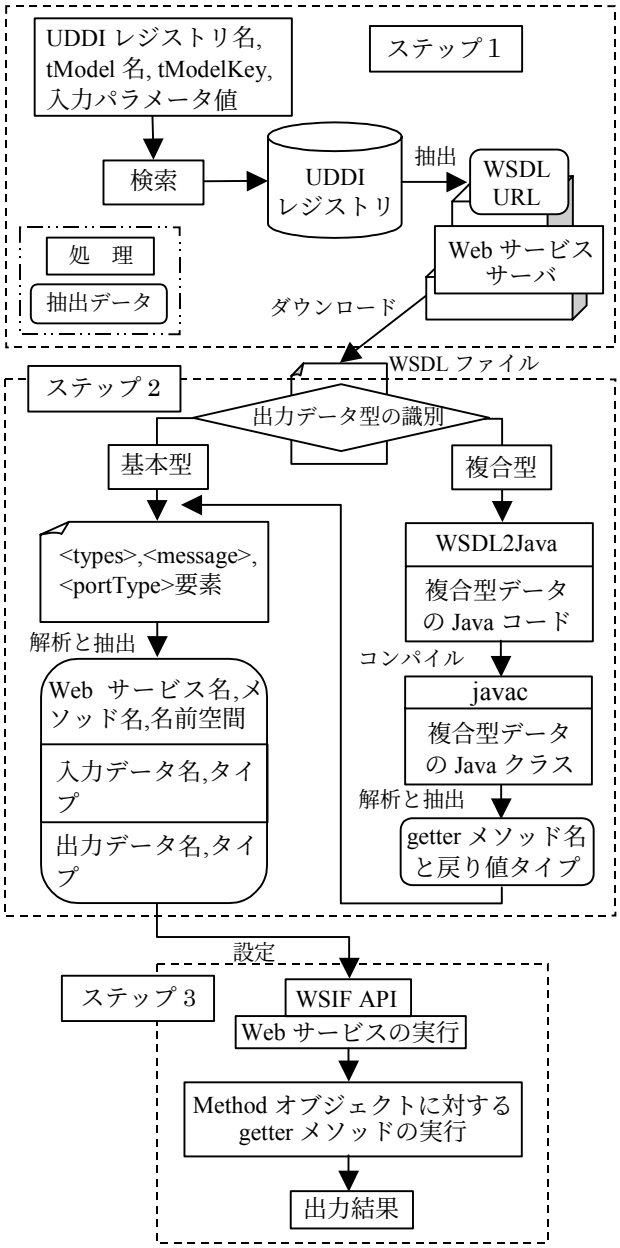

図 4.3　システム全体の処理の流れ

・**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*[5]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，Java2 SE ライブラリ中の *Class, Method* 及び *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 4.4 に示す．

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 4.4　getter メソッドの抽出

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* 要素及び *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，更に *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.

・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及び getter メソッドの動的実行など図 4.5 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, 本システムで完全に自動化された.

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 4.5 getter メソッドの動的実行

4.3.4 実証実験

提案した自動化手法と動的実行手法の有効性を確認するため，基本型 Web サービス 4 種類，複合型 Web サービス 4 種類に対して実行実験を行った．実際のビジネスでの使用を想定し，これらの Web サービスを 3 種類のパブリックな UDDI レジストリ：IBM UDDI ビジネスレジストリ[25]，IBM UDDI テストレジストリ[26]，Microsoft UDDI レジストリ[46] に登録し，そのレジストリから個々の Web サービスを検出し，実行を評価した．UDDI レジストリへの登録データを図 4.6 に示す．

試作したシステムは，Windows 2000 上で Java 言語（Java2 SDK 1.4.1_02），WSIF 2.0，Apache-Axis 1.0 で実装したものである．使用したハードウェアは SOTEC G380AV (Intel Pentium Ⅲ 800MHz) である．

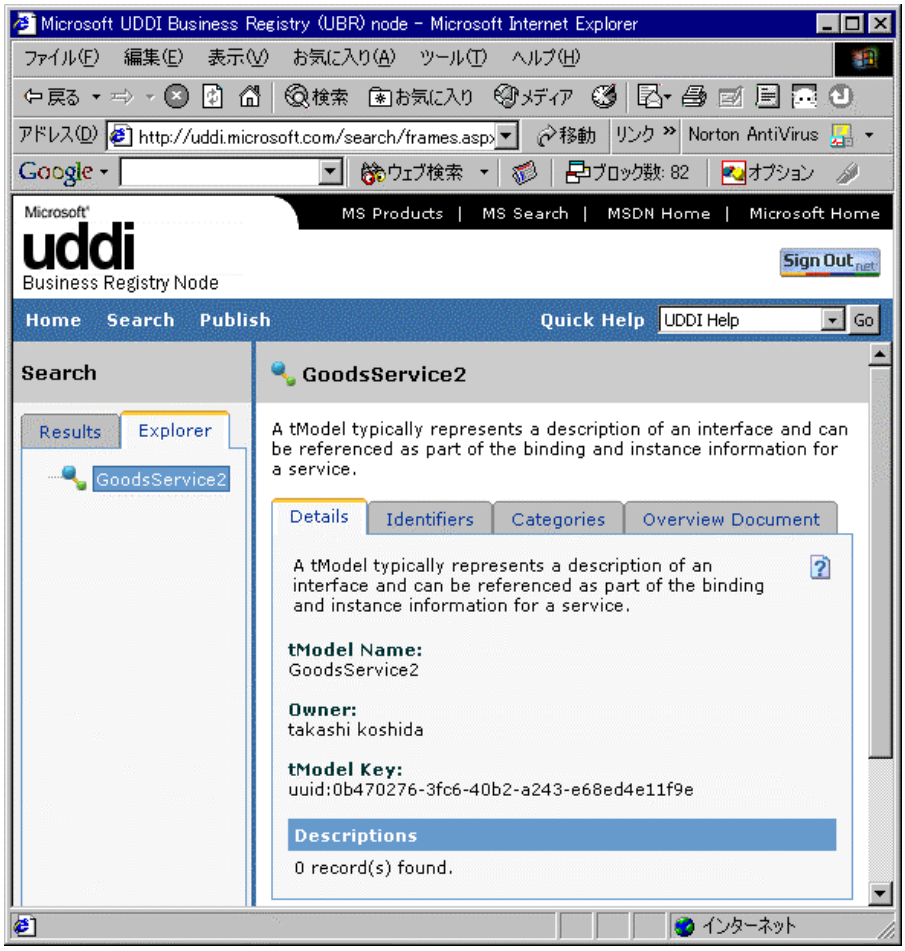

図 4.6　UDDI レジストリに登録された Web サービス

（1）従来手法との比較

基本型 Web サービスに対するシステム A の実行結果を図 4.7 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，それらが特定されないと実行できない．

図 4.7　システム A の実行結果（出力データ型：基本型）

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 4.8 に示す．このシステム B は，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[64]Web サービス専用の実行システムである．この場合も入力パラメータ値として，Web サービスの WSDL URL を指定する必要がる．このシステムには，Zip2Gio の出力データに対応した JavaBeans クラスと WSDL ファイルも予め添付されている．従って，汎用性がない専用システムである．

図 4.8　システム B の実行結果（出力データ型：複合型）

（2）本システムの実行結果

本システムの実行結果を図 4.9 から図 4.12 に示す. 図 4.9 と図 4.10 は出力データが基本型の例であり, 図 4.9 は図 4.7 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 4.10 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.

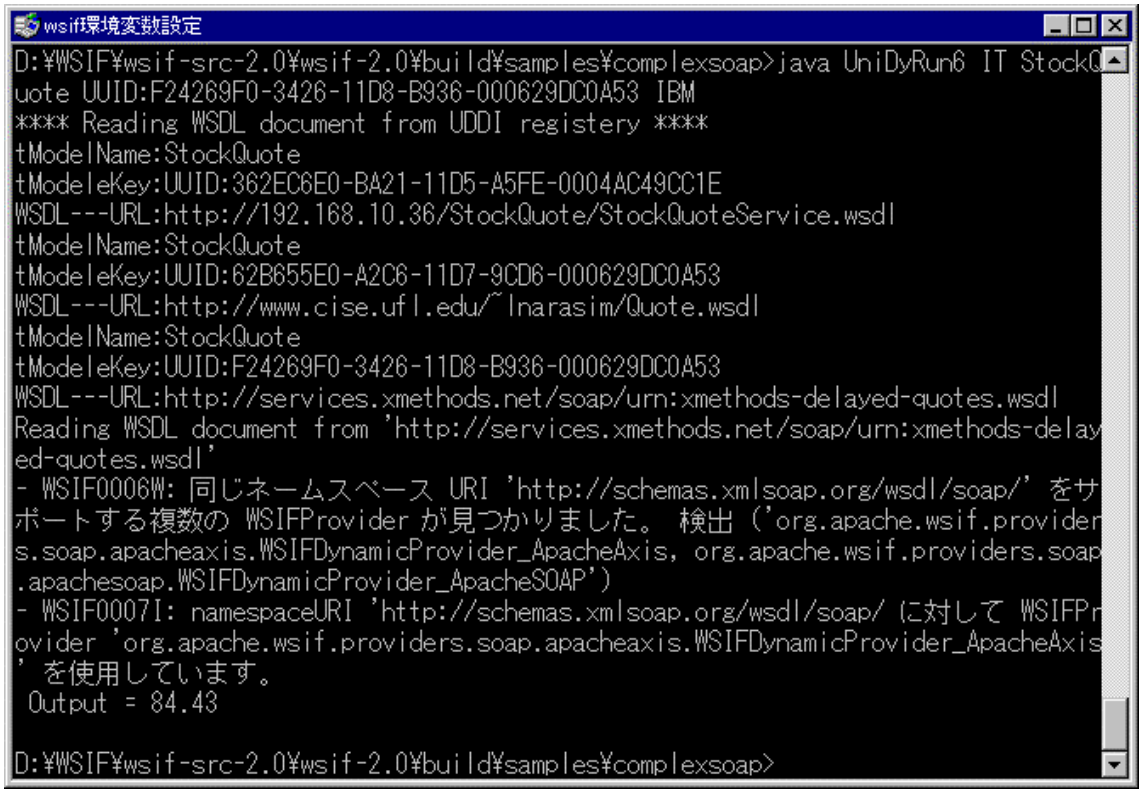

図 4.9　本システムの実行結果：その 1（出力データ型：基本型）

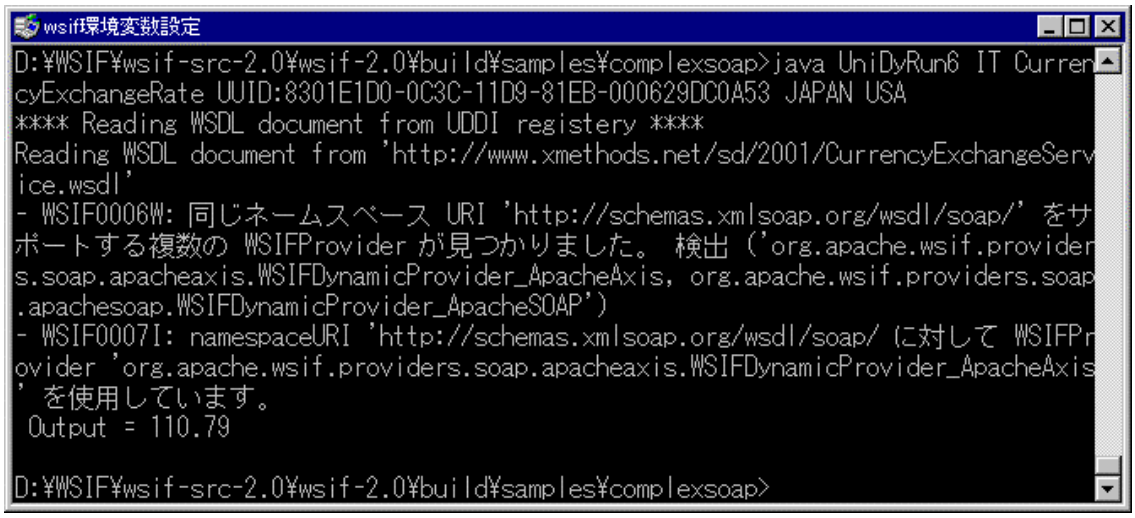

図 4.10　本システムの実行結果：その 2（出力データ型：基本型）

図 4.11 と図 4.12 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 4.11 は図 4.8 と同じ Web サービスの実行結果である．図 4.12 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 4.11 の“I”は IBM UDDI ビジネスレジストリの指定であり，図 4.12 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．

表 4.1　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

	従来のシステム		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 / 工程	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
WSIF 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「抽出するデータ数，または設定されるデータ数」を示している．
（*数値）は「生成するクラス数」を示している．

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 ヵ所だけであり，他は本システムと同様に全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[64]Web サービス実行用に限定されたシステム（図 4.8）である．他の複合型 Web サービスを実行するためには，表 1 で示した全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

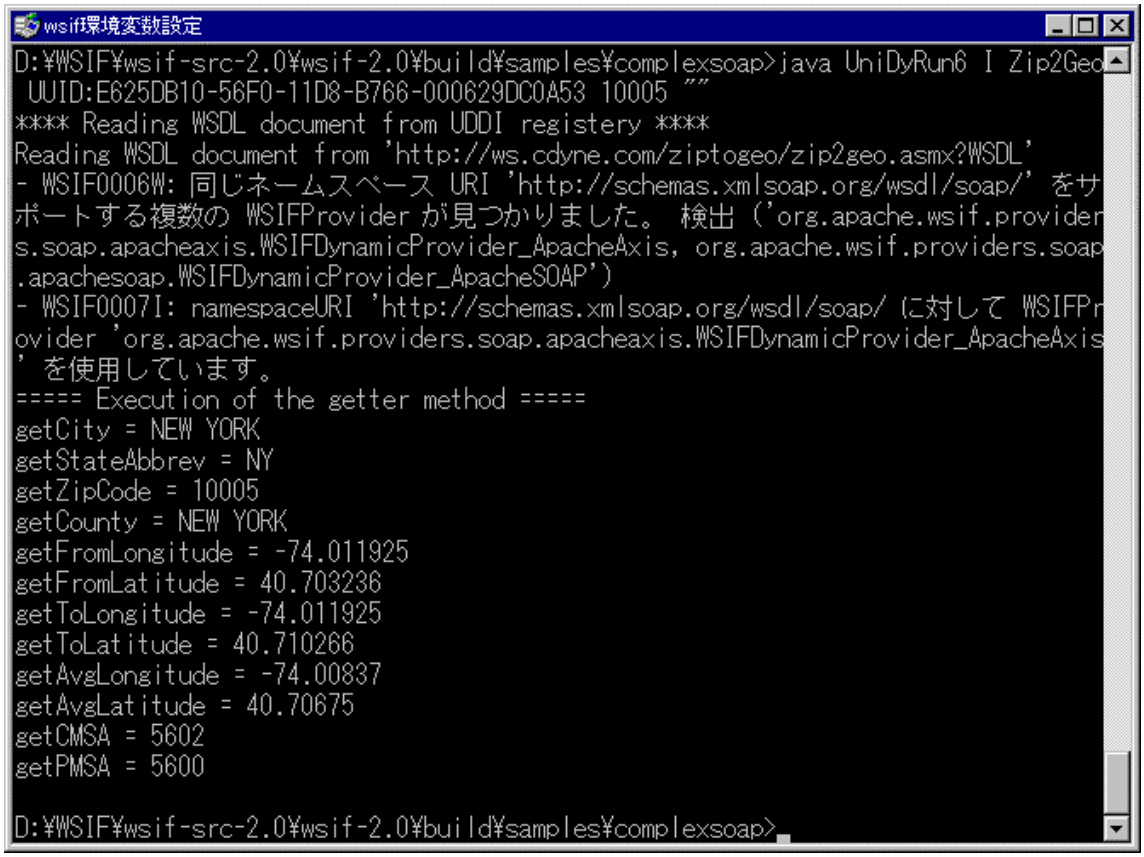

図 4.11　本システムの実行結果：その 3（出力データ型：複合型）

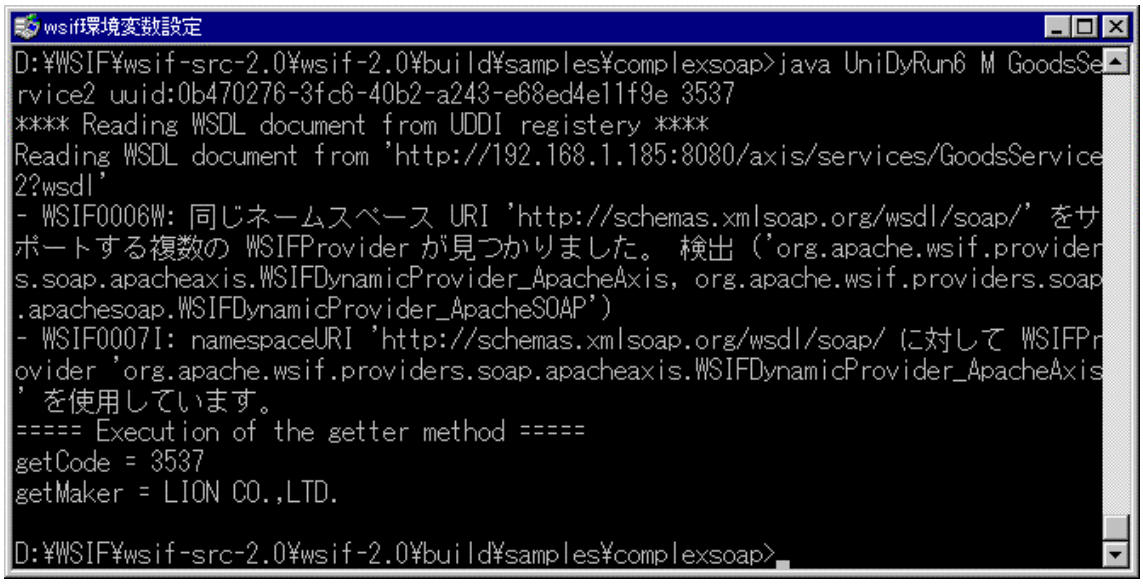

図 4.12　本システムの実行結果：その 4（出力データ型：複合型）

4.3.5　　評価

実験の目的は，提案手法を実装したシステムの動作検証と本手法の有効性の検証である．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，全て本システムで正常に実行できた．有効性は，本システムと従来システムの Web サービス実行効率を比較して検証した．表 4.1 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．

分であった．しかし，本システムでは全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．従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ることがわかる．

4.3.6 関連研究

本システムの場合と同様に，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[33,51]がある．このシステムはオントロジ記述言語として DAML-S[15]を利用して，それのプロファイル記述データを UDDI レジストリの *businessEntity* 及び *businessService* 要素にマッピングするとともに，更に *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全ての 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．従来通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易く，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．更に将来的には，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，Apache-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．従って，Apache-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 のインストール及び Java 総合開発環境 Eclipse [20]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 を使うための開発及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．更に，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．それに対して本研究では，Apache-Axis ライブラリの *WSDL2Java* 及び Java2 SE の標準ライブラリである *Class, Method, Object, Runtime* クラスなど全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勿論のこと，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，全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．

4.4 結言

本章で提案した手法とその手法を実装したシステムにより，任意の Web サービスに対して，UDDI レジストリからその Web サービスを特定し，スタブを用いることなく動的に Web サービスを実行することが可能になった．これは，第 1 章で述べた本研究の研究課題 3 項目中の第 1 課題，

（1）Web サービス実行には，ユーザ側でその Web サービスを駆動するスタブが必要であり，その開発負荷が大きい．

を解決する手法である．この動的実行手法を CALS/EC システムに組込むことによって，常に必要に応じた最新の Web サービスを UDDI レジストリから選択して，実行することが可能となる．従来の Servlet 技術によるクライアントーサーバ型 CALS/EC システムでは，システムに関するディレクトリ情報は存在しなかったため，限られた範囲内のシステムしか活用できなかったが，この UDDI レジストリによるインターネット上の Web サービスディレクトリ情報により，ユーザ要求に合う Web サービスを検出し，実行することが可能になった．この UDDI レジストリの活用は，変化の激しいビジネス分野において，それに対応した柔軟な CALS/EC システム構築に極めて有効である．本章で，Web サービスをインターネット上で一元的に登録・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 レジストリと WSDL 記述を利用し，更に 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 Web サービスの出力データ型に依存しない統合化システムについて詳述した．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もつ Web サービスに対して，そのデータ型に対応した JavaBeans クラスの実行時動的生成と getter メソッドの自動抽出，及びその動的実行手法について説明した．また，Web サービス実行時における，各種のパラメータ設定を自動化する手法についても説明した．これらの手法によって，Web サービス実行時におけるユーザの開発負荷が格段に軽減されるとともに，利便性が向上し，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 サービスについて更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，実際の CALS/EC ビジネス分野での活用に繋げること，

- 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，選別と連携をユーザ側の視点で研究し，システムとしての実用化をめざすこと，

が挙げられる．

第5章　Webサービスの検出と連携に関する研究

5.1　序言

第4章では,本研究の第1の研究課題を解決する手法を提案し,その手法に基づいたシステムを実装,実証実験を通して,提案手法の有効性を確認した.本章では,残りの二つの課題,

第2の課題：ユーザが必要とする,または利用したいWebサービスの検出が難しい.

第3の課題：ユーザがWebサービスを利用する際に,その機能や入出力インターフェースなど利用法について短時間で正確に知ることが難しい.

を解決する手法として,*プリミティブWebサービス*という概念を提案する.そして,その概念に基づいて実際にプリミティブWebサービスを開発し,実証実験を通してその有効性を示す.

多様なビジネス分野において,Webサービス,ならびにそのWebサービスから構成されるCALS/ECシステムが広く活発に利用されるためには,ユーザが必要とするWebサービスが,必要になった時点で的確に検出され,かつ容易に実行できることが必要である.つまり,Webサービス提供企業が,ユーザ要望に合致したサービスを如何に使いやすく提供できるかが重要である.ユーザにとって重要なことは,Webサービスによって得られる情報

であり，Web サービスそのものではない．インターネット上で Web サービスを一元的に登録し，管理・運営する仕組みとして UDDI レジストリがあり，様々な Web サービスに関する情報が格納されている．そして，その UDDI レジストリは，柔軟でかつグローバルな CALS/EC システムを実現する基盤である．

しかし，その UDDI レジストリに対する Web サービス登録に際しては，Web サービスの名称についての命名法やその機能説明に対する規約，入出力インターフェースに対する規約などに関しての標準化機構はなく，全て Web サービス提供側の任意性に委ねられている．従って，その機能や入出力インターフェースなどの利用法についても，利用する Web サービス毎に，UDDI レジストリ登録情報や WSDL 記述を目視で確認して理解するほかなく，ユーザの負荷・労力が大きい．また，UDDI レジストリの検索手段もキーワード検索とコード検索（業種コードと製品コード）に限られ，多彩なユーザ要望を反映した検索手段がなく，要望に合致する最適な Web サービスを検出することが難しい状況である．ユーザニーズに合致する Web サービスの検出に対する研究として，オントロジ記述言語 DAML-S データを利用した方法が提案されているが[51]，その効果は，DAML-S データを付与された Web サービスに限定されるなどの問題点がある．

本章では，これらの問題を解決する手法として，プリミティブ Web サービス(PWS)という概念を提案する．PWS は様々なビジネス分野で共通に利用可能な一意に統一された名称，機能，入出力インターフェースをもつ基本的な Web サービスとして定義される．これにより，任意性や曖昧さが解消され，必要とする Web サービスに対する的確な検索と正確な理解が期待される．さらに，PWS を連携・制御するエージェント群を新規に開発し，具体的なビジネスプロセスに適用して，その有効性を確認する．

以下，5.2 節で従来手法の問題点について述べ，5.3 節で PWS

の提案について詳述する．5.4 節で具体的な PWS の機能と実装について説明し，適用するビジネスモデルについて詳述する．5.5 節でそのモデルに沿った実証実験と評価を述べる．5.6 節で関連研究について紹介し，最後に 5.7 節で本章のまとめを述べる．

5.2 従来手法の問題点

近年，Web サービスの連携など Web サービス実用化に関する研究が活発に行なわれているが，しかし，実際のビジネス分野における Web サービスの利用は依然として，あまり広がりを見せていない．その原因として，次の 2 項目が考えられる．

（１）ユーザが必要とする，または利用したい Web サービスの検出が難しい．

（２）ユーザが Web サービスを利用する際に，その機能や入出力インターフェースなど，利用法の正確な理解と把握に時間がかかる．

（１）の解決のために，インターネット上で Web サービスの登録と検索を一元化する UDDI レジストリが導入されたが，Web サービス検索手段はキーワード検索とコード検索のみであり検索機能が不十分であること，また UDDI レジストリに登録されていても実際には Web サービスが存在しない，など登録情報の信頼性に欠けるなどの問題が生じている[56]．（２）については，Web サービスのインターフェースを記述する WSDL とともに RDF(Resource Description Framework)[52] や DAML-S[15]などを用いて，Web サービスの機能や入力データに関する制約を記述したセマンテックサポートが提案されている．しかし，全ての Web サービスに果たして矛盾なくセマ

ンテックデータが付与できるかなどの問題がある．今後，様々なビジネス分野においてWebサービスの利用が拡大するためには,上記(１),（２）の問題解決と複数の Web サービスの効果的な連携が必要不可欠であると考える．

これらの問題を総合的に解決し,Webサービスの効果的な連携を実現する手法として,「プリミティブWebサービス(PWS)」という概念を提案する．

5.3 プリミティブWebサービスの提案

ここでは，プリミティブ Web サービス(PWS)の具体的な定義について説明するとともに，それを利用した場合の効果やその利点についてユーザ側とサービス提供者側の両方の視点から説明する．

5.3.1 プリミティブ Web サービスの定義

現在，Webサービスを利用するユーザにとっては，予め，その Web サービスの機能や入出力インターフェースを充分理解していることが必要最低条件である．そのため，事前に利用する Web サービスについて調査し，理解するための充分な時間が必要になる．その労力と負荷を軽減するためには，ビジネスプロセスの最小構成単位毎に，名称，機能，入出力インターフェースが一意に統一された「基本的な」Webサービスが用意されていれば良い.ユーザは一度その機能と入出力インターフェースについて理解すれば，以降はユーザがそれら「基本的な」Webサービスを組み合わせて任意の業務処理を構築・実行できるようになる．

この考え方は，従来の任意性のある Web サービス開発を見直し，できるだけ汎用性をもつ「基本的な」Webサービスを設計し，それらの組合せによってビジネスプロセスを処理する方向へと導く．以降，この汎用性をもつ「基本的な」Web サービスを「プリミティブ Web

サービス(PWS)」と呼び，様々なビジネス分野で共通に利用可能な一意に統一された名称，機能，入出力インターフェースをもつ基本的なWebサービスとして定義する．例えば，物品購入等の商取引に関しては，物品検索，信用確認，受発注，在庫確認，価格見積り，納期確認，配送，代金決済などの各処理がPWSとして考えられる．また，これらPWSの一覧表を作成し，インターネット上で公開し，UDDIレジストリ検索を補助する．更に，PWSを組み合わせて，ビジネスプロセス単位毎の標準Webサービスも開発できる．そして，このPWSと標準Webサービスを実行するためのユーザ側のインターフェース・システムを開発することで目的に応じたWebサービスが容易に利用可能となる．

5.3.2　予想される効果

複数の企業が同じ仕様をもつ製品を生産している場合，その製品を購入しようとするユーザは，各企業の製品を比較し，希望条件に合致する製品を購入するプロセスをとる．この際，ユーザは全ての競合企業に対して，同じ条件で効率良く製品を比較し，選別したい．例えば，商品価格見積りにおいては，ユーザ指定の商品に対して，競合他社より如何にユーザ側にメリットがある見積りを提供できるかが重要であり，そのWebサービス自体は他社と同一で何ら問題ない．そのWebサービスで提供できる「サービスの内容と質」が重要であり，それ以外は問題ではない．

以下で，PWS導入の利点をユーザ側，企業側に分けて述べる．

（１）ユーザ側の利点

従来のWebサービスが標準化され，一意に統一された名称，機能，入出力インターフェースをもつPWSとして提供されるので，従来のWebサービスがもつ不統一性，曖昧さがなくなり，従来のUDDIレジストリのキーワード検索でも目的のWebサービスが容易に検

出できるようになる．また，Web サービスの機能・利用方法修得の負荷が格段に減り，ユーザの使い易さが大きく向上する．更に，同一入力条件で，企業別に複数の同じ Web サービスが一度に実行でき，処理効率が向上するとともに実行結果の比較とそれに基づく選別処理が容易になる．

（２）Web サービス提供企業側の利点

PWS を提供・利用することにより，各企業の Web サービス開発コストが削減され，その削減された経営資源をその企業のコア・コンピタンスに投資できる．また，様々なビジネス分野で PWS として標準化されるので，UDDI レジストリからの検出や連携が容易になり，Web サービスが使い易くなる．それにより，Web サービスの利用機会が増え，ビジネスチャンス拡大が期待できる．

5.4　プリミティブWebサービスの設計と実装

前節で提案した概念に基づき，具体的に PWS を開発・配備すると共に，その実行制御と連携を行うエージェントも新規開発し，これらを統合した商品調達 B2B システムとして実装する．

5.4.1　　ビジネスモデルへの適用

本システムを適用する商品調達のビジネスモデルについて説明する．このモデルは，卸売業者を中心として，小売業者，商品の製造メーカ，企業の信用を調査する信用調査会社から構成される．具体的な商品として，ビールを想定し，ビールメーカ，酒類卸売業者，小売業者間の商取引を想定している．小売業者から商品納品を受注した小売業者は，メーカ 3 社に対して商品の在庫と価格見積りを行い，その中から最も低価格の商品製造メーカを選別し，発注するシナリオである．

ビジネスプロセスは複数のサブプロセスから構成されので，そのサブプロセス毎に独立した PWS を用意し，それらを組み合わせて一連のビジネスプロセスを処理する．ここでは，指定された商品在庫数と商品価格を見積る“ 在庫管理 ”，商品の受注処理を行う“ 商品受注 ”，そして，企業の信用を照会する“ 信用照会 ”のサブプロセスに対して，各々PWS「在庫管理 Web サービス」,「商品受注 Web サービス」,「信用確認 Web サービス」を用意した．

商品を製造する複数のメーカが同じ PWS である「在庫管理 Web サービス」,「商品受注 Web サービス」を提供している．更に，企業の信用を調査する複数の信用調査会社も同一の PWS である「信用確認 Web サービス」を提供しているとする．

5.4.2 ビジネスプロセス

このビジネスモデルおいて，PWS が配備されているサーバは，3 メーカと 1 信用調査会社の計 4 サーバである．商品調達 B2B システムは，これら PWS の実行制御と連携を行う卸売業者サーバ上の 3 エージェントを中心にして，各 PWS が疎結合されているシステムである．小売業者-酒類卸売業者-ビールメーカ間の商品調達に関する一連の処理が，以下に示す（１）～（８）のビジネスプロセスとして実行される．また，その処理の流れを図 5.1 に示す．

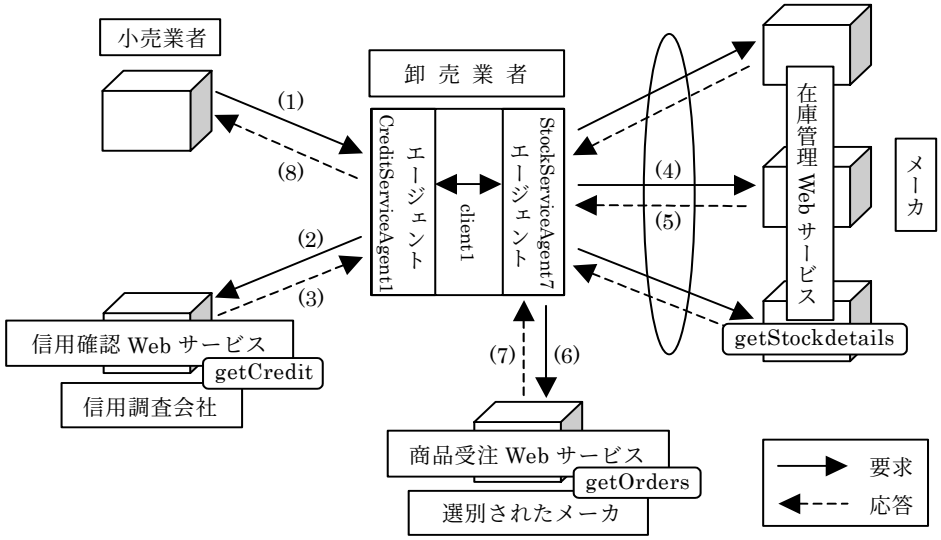

図 5.1　ビジネスモデルにおける処理の流れ

（１）商品納品要求：

ある小売業者が新規の取引を求めて，卸売業者に対して商品納品を要求する．

（２）信用確認要求：

卸売業者は企業調査会社に対して，その小売業者の信用調査を依頼する．

（３）信用確認回答：

信用調査会社は，「信用確認 Web サービス」を利用してその小売業者の信用確認を行い，返答する．

（４）在庫確認と価格見積り要求：

信用調査結果に問題がなければ，卸売業者は要求があった商品を製造するメーカに対して，在庫確認と価格見積りを依頼する．こ

の際，同じ仕様の商品製造メーカが複数存在すると仮定し，その中の 3 メーカに対して同時に在庫確認と価格見積りを依頼する．

（５）在庫確認と価格見積り回答：

メーカは「在庫管理 Web サービス」を利用して，商品の在庫確認と価格見積もりを同時に行い，卸売業者に返答する．

（６）商品選別と発注：

卸売業者は，3 メーカからの価格見積り結果をエージェント(StockServiceAgent7)によって比較し，希望条件（様々な条件が考えられるが，今回は最も価格が安い条件に設定する）に合致した商品製造メーカを選別し，商品を発注する．

（７）メーカ商品受注完了：

発注を受けたメーカは,「商品受注 Web サービス」を利用して受注処理を行い，その完了後に卸売業者に対して受注完了メッセージを送信する．

（８）商品受注完了：

それを受けて卸売業者は小売業者に納品要求受注完了メッセージを送信する．

5.4.3　プリミティブ Web サービスの設計

各サーバがもつ PWS の機能について説明する．いずれの Web サービスも Apache-Axis 1.0 で開発し，Tomcat 4.1.27 上に配備している．また，DBMS は Microsoft Access 2000 を使用し JDBC(Java DataBase Connectivity)を用いて各 PWS とアクセスしている．

（１）メーカのプリミティブ Web サービスの機能

各メーカのサーバには，「GoodsService6」Web サービスに属する２つの PWS「在庫管理 Web サービス」(getStockdetails)と「商品受注 Web サービス」(getOrders)が配備されている．

・**「在庫管理 Web サービス」(getStockdetails)：**
入力データとして卸売業者 ID 番号，卸売業者名，希望ビールタイプ，希望ケース数，希望納品日を指定し，出力データとして回答メーカ名，指定ビールタイプ，指定ビールタイプに合致する中で最も低価格の商品名，商品コード，単価，指定ケース数，全体価格を返す．

・**「商品受注 Web サービス」(getOrders)：**
入力データとして卸売業者 ID 番号，卸売業者名，希望ビールタイプ，希望ケース数，希望納品日を指定し，出力データとして受注完了メッセージを返す．

これら 2 種類の PWS を開発し，Web サービスサーバ上に実際に配備した状況を図 5.2 に示す．

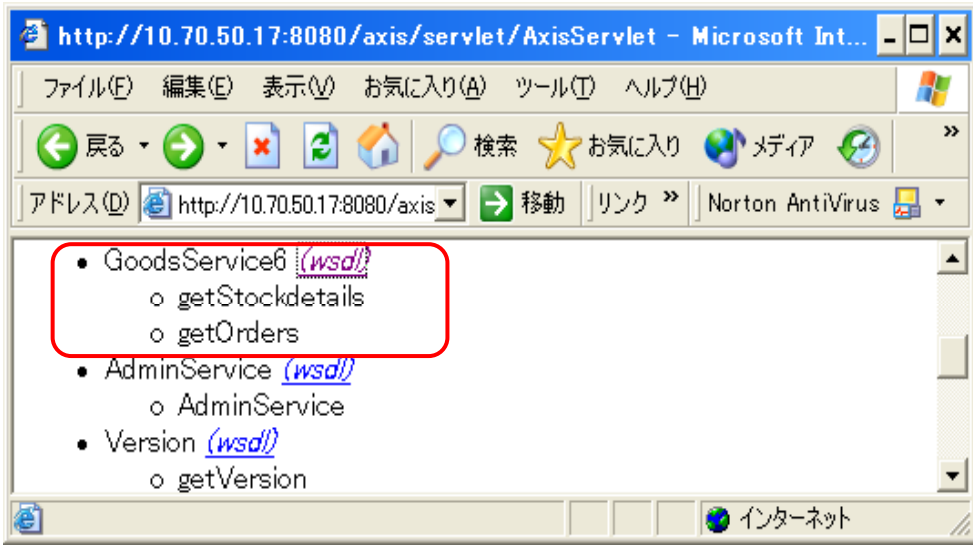

図 5.2　メーカにおける Web サービス配備状況

（２）信用調査会社のプリミティブ Web サービスの機能

信用調査会社のサーバには「CreditCheck1」Web サービスに属する PWS「信用確認 Web サービス」(getCredit)が配備されている.

・**「信用確認 Web サービス」(getCredit)：**
入力データとして調査対象企業名とその電話番号を指定し，出力データとして調査対象企業名，社長名，住所，電話番号，信用結果（OK または NG）を返す.

その PWS を開発し，Web サービスサーバ上に，実際に配備した状況を図 5.3 に示す.

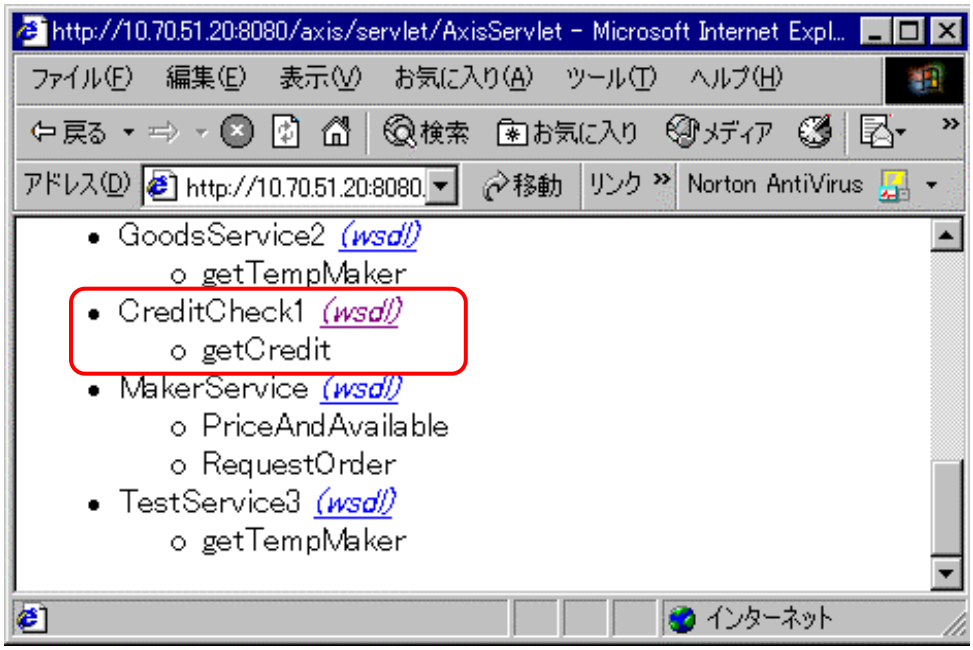

図 5.3　信用調査会社における Web サービスの配備状況

5.4.4　エージェントによる制御と連携

上記の各 PWS の実行を制御し，連携する手段として，卸売業者のサーバ上で稼動する 3 種類のエージェントを開発した．そのエージェントの機能について説明する．卸売業者のサーバには，次に示す 3 種類のエージェント，

- メーカの PWS を実行するエージェント(StockServiceAgent7),
- 信用調査会社の PWS を実行するエージェント(CreditServiceAgent1),
- それらの 2 エージェントを連携し，制御するクライアントエージェント(client1),

が配備されている．クライアントエージェントは，3 メーカからの価格見積り結果を比較し，最も低価格の商品を選別する．エージェントの稼動状況を図 5.4 に示す．これらの各エージェントは JADE-3.0[31] を利用して開発した．JADE[11]とは FIPA(The Foundation for

Intelligent Physical Agents)[58]仕様に準拠したエージェントを Java 言語で開発するためのフレームワークである.

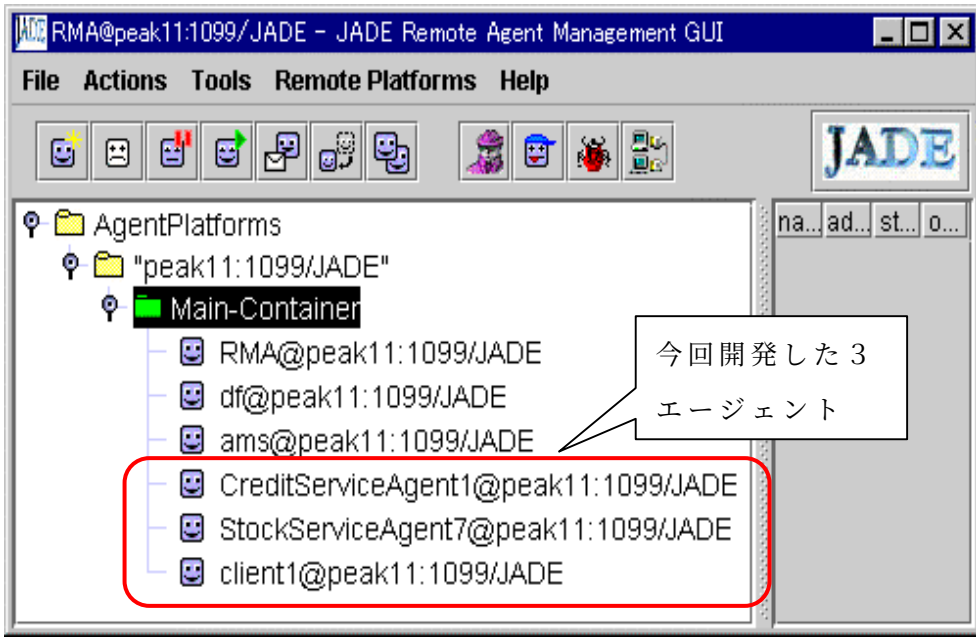

図 5.4　卸売業者で稼動するエージェント

5.5　実験と評価

この実証実験の目的は, 商品調達 B2B システムを構成する PWS とエージェントの動作・連携を確認すること, 及び従来の Web サービスを用いた場合の処理工数と本システムの処理工数を比較し, PWS の有効性を示すことである. 以下で, 前節で示したビジネスプロセスに沿って, 本システムの実証実験結果を具体的に説明し, 従来の Web サービスを利用した場合と処理工数を比較する.

5.5.1　実証実験

ビジネスプロセスに沿った実験結果を示す.

（１）商品受注と信用確認

最初に，卸売業者は新規の小売業者から商品納品依頼を受けるとクライアントエージェント(client1)の制御の元で，その小売業者の信用照会をエージェント(CreditServiceAgent1)を使って実行する．このエージェントによって，信用調査会社の PWS「信用確認 Web サービス」getCredit が実行され，その調査結果が卸売業者に回答される．その処理結果を図 5.5 に示す.

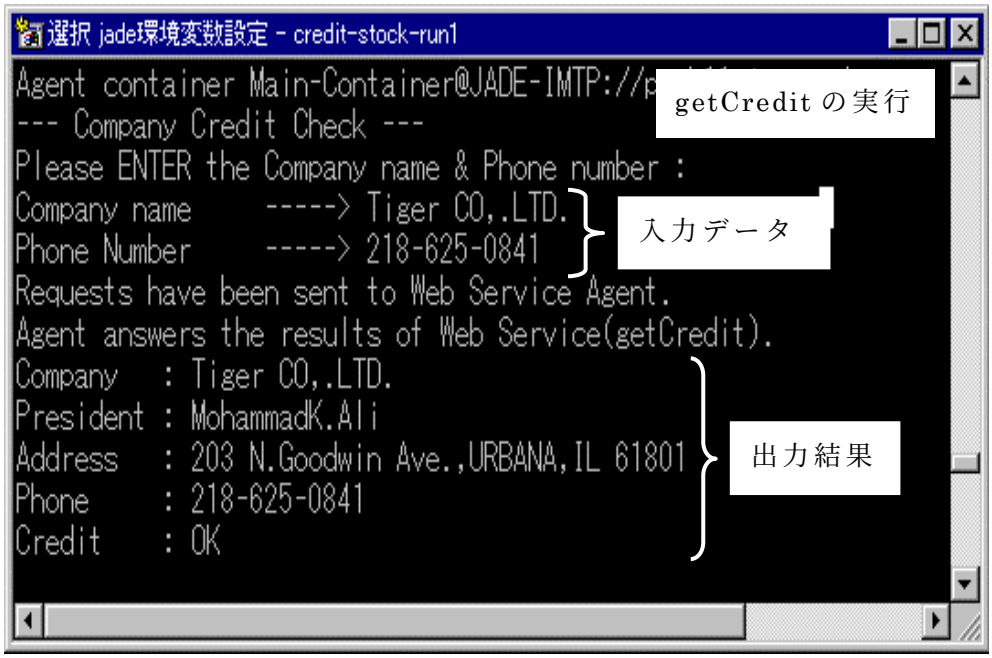

図 5.5　小売業者に対する信用照会実行状況

（２）在庫確認と価格見積り

小売業者の信用が確認されれば，次にエージェント(StockServiceAgent7)を使って，3 社のビールメーカの PWS「在

庫管理 Web サービス」(getStockdetails)を順次実行し，各メーカから在庫と商品価格見積り結果をこのエージェントがまとめて表示する．まず，メーカ 3 社の「在庫管理 Web サービス」に対する入力データを図 5.6 に示す．この入力データは一度指定するだけで，3 メーカの Web サービスに自動的に設定される．次に図 5.7 に，3 メーカからの「在庫管理 Web サービス」からの出力結果を各々maker1, maker2, maker3 の部分で示す．

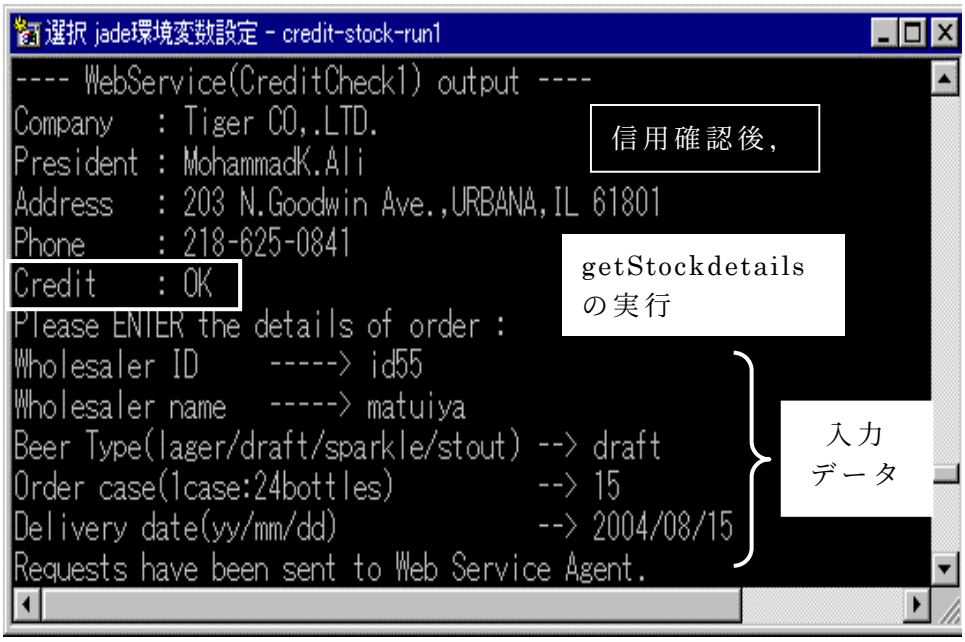

図 5.6　在庫管理 Web サービスの実行

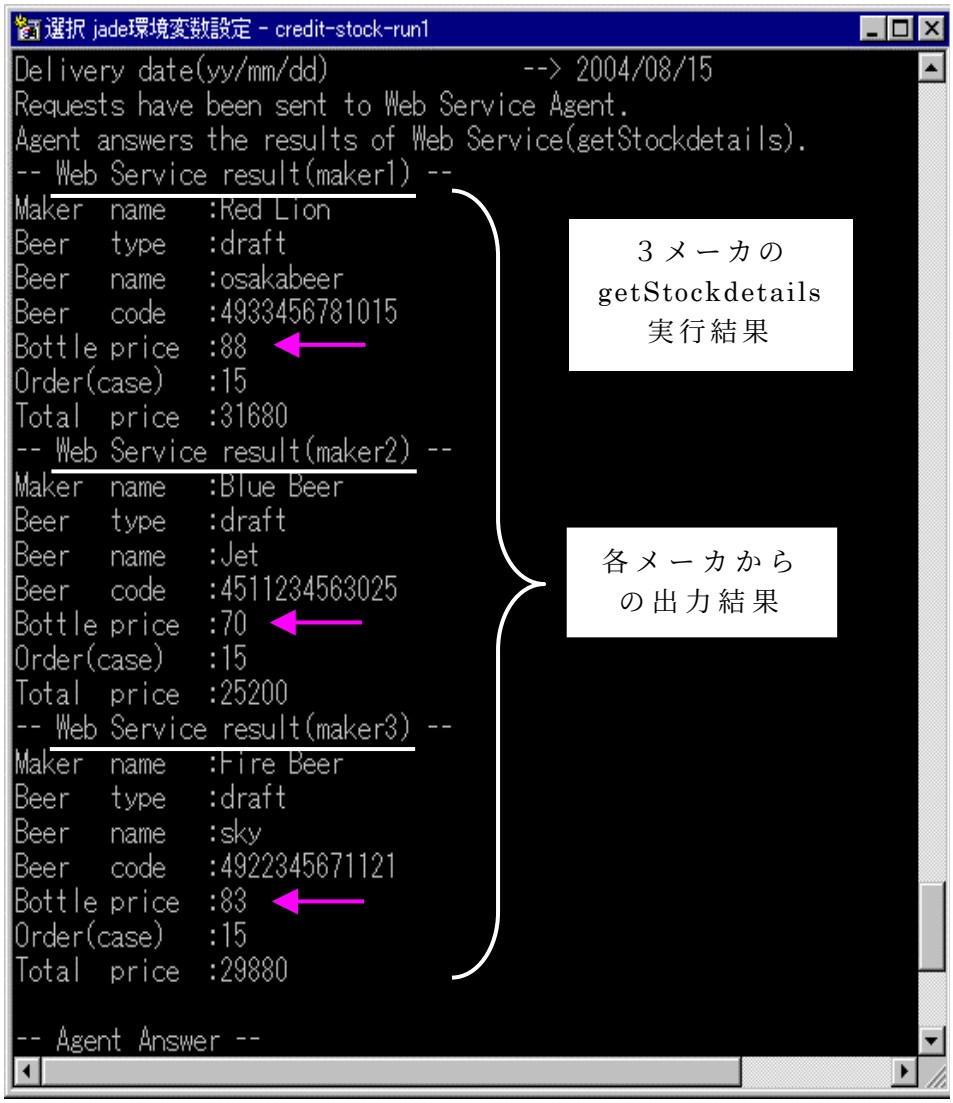

図 5.7　メーカに対する在庫管理 Web サービスの実行結果

（3）商品選別と発注

次に，クライアントエージェント(client1)がこれら 3 メーカからの商品価格見積り結果を比較して，最も低価格の商品を選別し，卸売業者に知らせる．図 5.8 の前半部分にその選別結果を示す．最も低価格な商品である maker2 の Blue Beer が選択されていることが分かる．その選択結果で良ければ，卸売業者はエージェン

ト(StockServiceAgent7)を使って，選択されたメーカ 1 社に対して，その商品を発注する．受注したメーカは PWS「商品受注 Web サービス」(getOrders)を使って，その商品の受注処理を行い，処理完了後に受注完了メッセージを卸売業者に返す．図 5.8 の後半部分にその受注処理結果を示す．

図 5.8　エージェントによる商品選択結果

5.5.2　　評価

今回開発した PWS とエージェントを用いた商品調達 B2B システムは，設定したユースケースに基づいて，全ての処理を正常に実行した．「在庫管理 Web サービス」(getStockdetails)を，PWS として開発・配備したので，3 メーカに対する在庫確認と商品価格見積りを一度に実行することが可能になり，その実行結果を比較しての商品選別処理もエージェントを利用することによって自動化された．各企業個別に「在庫管理」，「価格見積り」，「受注」の各々のプロセスに対応した

Web サービスが存在する従来形式ならば，企業毎に別々の Web サービス実行が必要となる．更に，各企業の「価格見積り」結果を比較しての商品選別処理も別途行わねばならず，ユーザの煩雑さと負荷は比較する企業数に比例して増加する．今回のユースケースでは，3 メーカを仮定して，その 3 メーカ全てに対する在庫確認と商品の比較・選別処理を，PWS とエージェントを利用して一括実行するので，その処理工数は従来の 3 社毎に別々の Web サービスを個別に実行する場合の 1/3 以下に削減されると考える．

今回のビジネスプロセスでは，「信用確認」のプロセスは，卸売業者が小売業者に対して行う「信用確認」のみに限定したが，この「信用確認」Web サービスは，どのビジネス分野でも必要な PWS の典型的な例であると考える．このような汎用的な PWS が多く開発され配備されることで，ユーザの Web サービス利用や連携が容易になり，企業側のビジネスチャンス拡大につながる．

5.6 関連研究

Web サービスの検出や連携に関する研究として，近年オントロジやエージェントを利用した論文が発表されている[21,22,51]．福田ら[21]の DAML-S と JADE を用いた Web サービス連携に関する論文は，DAML-S のプロセス記述に基づき，複数の Web サービスを連携するための JADE プログラムコード自動生成システムを提案している．しかし，生成されるコードはひな形コードのみであり，エージェントが Web サービスを実行する処理，及びその実行結果に対するエージェントの処理については全て人手で個別に開発しなければならず，その具体的な手法や内容ついては一切触れていない．複数の Web サービスを実際に連携して実行するには，必要な Web サービスをどのように検出し，実行するのか，更にその実行結果に対して，次の Web サービスをどう連携させるのかが最も重要であるが，それについて具体的

に述べていない．それに対して我々は，JADE エージェントを利用して実際に Web サービスを実行する手法とその実行結果に基づいて Web サービスを連携させる手法を実装し，システムとして具体的に示している点が異なる．

また，Paolucci ら[51] は UDDI レジストリに DAML-S 記述によるオントロジデータを追加することで，UDDI レジストリから Web サービスを検出する精度を向上させる手法を示している．しかし，オントロジデータを付与したサンプルデータに対しては有効であるが，オントロジデータを持たない任意の Web サービス検出に対しては効果がない．実用化するためには，全ての Web サービスに対して，効果的なオントロジデータを付与することが必要であるが，その方策について触れていない．更に，検出した Web サービスの具体的な実行方法についても述べていない．今回のシステムでは URL を指定して Web サービスを実行しているが，その Web サービスは，統一された一意の名称，機能，入出力インターフェースをもつ「プリミティブ Web サービス」として UDDI レジストリに登録される．更に，これら PWS の一覧表を作成し，インターネット上で公開することによって，従来の Web サービスがもつ不統一性，曖昧さがなくなり，従来の UDDI レジストリのキーワード検索でも容易に目的とする Web サービスが検出できるようになる．

5.7 結言

本章では，必要とする Web サービスを UDDI レジストリから的確に検出し，スムーズに実行するための手法として，PWS という概念を提案した．そして，その概念に基づき，具体的に PWS を開発・配備するとともに，それらを連携・制御するエージェントを新規に開発・統合化し，商品調達 B2B システムとして実装した．PWS を導入することにより，Web サービスの名称，機能，入出力インターフェー

スに対する不統一性や曖昧さが解消され，必要な Web サービスが UDDI レジストリのキーワード検索で的確に検出可能となり，その実行も容易になる．

PWS を用いた具体例は，今回の商品調達の例のみであり，今後は様々なビジネスプロセスを分析して，汎用的な PWS を設計・開発することが必要である．その際には，

・ビジネスプロセス単位での入出力データの種類とその内容，

・PWS の粒度，

・高レベルの汎用化と再利用化を実現する設計手法，

などに関して更なる検討が必要である．セキュリティの確立も実用化に向けての重要な条件である．また，任意の PWS の動的実行と連携を制御できる汎用的なエージェントシステムについてもセマンテックスを利用して検討したい．Web サービスの連携に関しては，BPEL4WS(Business Process Execution Language for Web Services)[2]が注目を浴びているが，連携を記述する BPEL ファイルの作成が難解であること，連携を実行する BPEL エンジンが新たに必要なこと，更に静的な連携に限定されること，などの制約がある．我々はできるだけ標準化技術をベースに Web サービス連携を実現することを目標にしている．

第6章　結論

6.1　本研究の成果

今後の CALS/EC の中心となる分散処理技術である Web サービスを利用した B2B システムを開発し，実際の商品調達ビジネスモデルに適用して，その有効性を確認するとともに Web サービスの動的実行に関する問題点や Web サービスの検索，利用法についての問題点を明確化した．そして，それらの問題点を解決する研究に取組み，その研究成果として，これまで実現できなかった，Web サービスの出力データ型に依存しない動的実行手法を提案し，システムとして実装を行い本手法の有効性を確認した．これにより，UDDI レジストリから Web サービスを検出し，そのまま連続して Web サービス実行が可能になった．また，従来，複合型 Web サービス実行時に必要であったデータクラスの開発や WSDL 記述アドレスの指定などを全て動的に自動化し，完全な動的システムとして実装することができた．本システムにより，特に Web サービスに関する専門知識を有しないユーザも，容易に Web サービスが実行できるようになった．

また，必要とする Web サービスの容易な検出や特定，および Web サービスの機能や入出力インターフェースに対する理解しやすさなど，利用法に関する問題を解決する手法として，プリミティブ Web サービス（PWS）という概念を提案した．PWS とは，様々なビジネス分野で共通に利用可能な，一意に統一された名称，機能，入出力インターフェースをもつ Web サービスとして定義される．それにより，従来の Web サービスがもつ，名称や機能に対する曖昧性や重複性，および入出力インターフェースの不統一性が解消され，的確な検索な

らびに利用法修得も容易になると予想される．その概念に基づき，実際に共通なビジネスプロセスに対応する PWS を開発するとともに，それらの PWS の実行を制御し，連携する手法として，新たにエージェントプログラムを開発し，それらをまとめてシステムとして実装した．そして，そのシステムを商品調達のビジネスプロセスに適用して，システム動作検証と提案手法の有効性を確認した．

これらの新しい手法や概念を提案しシステムとして具現化することで，Web サービスを利用するユーザサイドにおける，Web サービス利用技術の向上に寄与した．Web サービスに関する専門知識を必要としなくとも，容易に Web サービスが利用できる環境に近づきつつあると考える．

6.2 今後の課題

最後に，本研究に対する今後の課題として以下が考えられる．

（１）Web サービス動的実行に関して

まず，Web サービスの動的実行に関しては，複合型出力データを表す WSDL 記述データに関して，更なる実証実験が必要である．本研究では，複数の標準的な複合型出力データ記述に対しては正常処理を確認しているが，未確認の記述パターンについても漏れなく検証を行う必要がある．また，現在のシステムはコマンドライン入力であるが，実行する Web サービス毎に動的な GUI を提供することで更に使いやすくなると考える．

（２）プリミティブ Web サービスに関して

プリミティブ Web サービス（PWS）に対して更なる研究が必要であると考える．今回は，ある特定の商品調達ビジネスモデルを想定して，PWS を設計したが，多くの多種多様なビジネス分野

のビジネスプロセスを分析して，共通な基本設計単位と粒度を明確にする必要がある．共通に利用できる汎用性が，プリミティブ Web サービスにとって最も重要である．

現在，広範囲なビジネス分野を対象とした CALS/EC の規格として ebXML[17]があり，また電子部品・半導体・情報機器分野を対象とした CALS/EC の規格として RosettaNet[53]がある．また，Web サービス関連技術の標準化をめざす様々な技術仕様が提案されている．今後はこれらの仕様を検討し，その仕様に沿って PWS の研究開発を推進する予定である．以下で項目に分けて，具体的に説明する．

- **ビジネスプロセスについて**

  RosettaNet では，ビジネスプロセスを 7 クラスタに分類して，そのクラスタ毎に細分化して全部で 83 種類の PIPs(Partner Interface Process) [55]を定義している．例えば，受発注のビジネスプロセスでは，Buyer と Seller 間での情報交換の手順とその内容が定義されている．また，ebXML では，BPSS(Business Process Specification Schema) [18]を使って定義される．PWS もこれらで規定されたビジネスプロセスに対応して設計する方向で研究を進める．

- **トランザクションについて**

  ビジネスプロセスのトランザクション管理についても，現在 OASIS が提唱する仕様である BTP(Business Transaction Protocol) [50] や IBM などが提唱する仕様である WS-Coordination[28]などがある．どの仕様を利用するかを各々比較・評価して検討したい．

・ **セキュリティについて**

Web サービスの SOAP メッセージに対するセキュリティ仕様として,IBM などが提唱する WS-Security[30]がある. これは, XML 暗号（PKI 技術を利用して，XML メッセージ全体もしくはその一部を暗号化する）と XML 署名（XML 要素を利用してデジタル署名を実現する）から構成される[1]. この組込みを検討する.

・ **ボキャブラリについて**

Web サービス設計における変数名の定義などについては，RosettaNet の Business Dictionary[54]や ebXML の Core Component[19]などを利用し，それらに登録されている標準用語を利用して，用語や変数名の統一化を行う.

また, Web サービスの合成や連携も重要な研究分野である. 本研究でのエージェント処理は，予め指定された PWS に対する指定された制御であったが, さらに動的な合成などの柔軟性と判断処理などの自律性をもたせることが必要である. そのための RDF，RDF スキーマや OWL-S などのオントロジ技術を利用したセマンテック Web の実用化に向けても，本研究をベースにして進めて行きたい.

謝辞

本稿を終えるにあたり，研究の機会を与えて頂くとともに，多くの面で有益な御指導と御配慮，激励を頂きました，主指導教員である情報科学研究科情報生命科学専攻（データベース学分野）植村俊亮教授に深く感謝致します．

さらに，本論文をまとめるにあたり，種々の御指導と御配慮を頂きました，情報処理学専攻（情報基礎学講座）関浩之教授，情報システム学専攻（ソフトウェア工学講座）松本健一教授，情報生命科学専攻（データベース学分野）宮崎純助教授に深く感謝致します．また，ミーティングを通し，数々の有益な御助言を頂きました，情報生命科学専攻（データベース学分野）天笠俊之助手，波多野賢治助手に深く御礼申し上げます．

さらに，研究初期に種々の御指導と御配慮を頂きました，名古屋大学 吉川正俊教授（前・奈良先端科学技術大学院大学助教授）に深く御礼申し上げます．また，本研究に対する有益な御助言，御協力を頂いたデータベース学分野 植村研究室の皆様に感謝致します．

さらに，本研究の機会を与えて頂くとともに，本研究の遂行に際して有益な御助言，数々の御配慮と激励を頂きました，筆者の所属する松江工業高等専門学校 宮本武明校長に心から感謝致します．さらに，いつも貴重な御助言と励ましを頂いた松江工業高等専門学校 門脇健氏に心から感謝致します．また，松江工業高等専門学校教職員の皆様には多くの面で御協力頂いたことに感謝致します．

最後に，常に有益な御助言と激励を頂いた学生時代からの友人である田辺政博氏，ならびに松下電器産業株式会社 芹川光彦氏，西脇青児博士，平野幹雄氏に心から感謝致します．

参考文献

[1] 株式会社アークシステム：6 章 注目されるアプリケーション統合技術-Web サービス（5.どのような課題があるのか），
http://www.arksystems.co.jp/closeupit/integration/0605.htm.

[2] Andrews,T. *et al.*,: BPEL4WS(Business Process Execution Language for Web Services) Version 1.1,
http://www-106.ibm.com/developerworks/library/ws-bpel/. (2003).

[3] Apache Software Foundation: Apache-Axis.
http://ws.apache.org/axis/. (2001).

[4] Apache Software Foundation: Tomcat.
http://jakarta.apache.org/tomcat/index.html. (2003).

[5] Apache Software Foundation: WSDL2Java.
http://ws.apache.org/axis/. (2001).

[6] Apache Software Foundation: WSIF (Web Service Invocation Framework). http://ws.apache.org/wsif/. (2002).

[7] Apache Software Foundation: Web services invocation framework.wsif-2.0.zip
(wsif-2.0¥build¥samples¥clients¥dynamicInvoker.class).
http://ws.apache.org/wsif/. (2002).

[8] Apache Software Foundation: Web services invocation framework.wsif-2.0.zip
(wsif-2.0¥build¥samples¥complexsoap¥client¥dynamic¥Run.class). http://ws.apache.org/wsif/. (2002).

[9] Apache Software Foundation: Xerces.
http://xml.apache.org/xerces2-j/. (2000).

[10] 荒井勝, 川野邉 衛：部品流通基盤の構築, CALS/EC Japan 1998 論文集, pp.225-232 (1998).

[11] Bellifemine, F., Poggi, A. and Rimassa, G.:
JADE - A FIPA-compliant agent framework, *Proceedings of PAAM'99*, pp.97-108 (1999).

[12] Borgden, Bill 著, 沖林正紀 監訳: Java による SOAP プログラミングパーフェクトガイド. 株式会社 技術評論社, 平成 14 年.

[13] 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/. (2000).

[14] Christensen,E., Curbera,F., Meredith,G.and Weerawarana, S.: Web services description language (WSDL) 1.1, http://www.w3.org/TR/WSDL. (2001).

[15] DAML-S (The DARPA Agent Markup Language Services). http://www.daml.org/services/. (2001).

[16] Duftler,M.J., Mukhi,N.K., Solminski,A. and Weerawarana,S.: Web Services Invocation Framework(WSIF), *Proceedings of Workshop on Object-Oriented Web Services (OOPSLA 2001)*, (2001).

[17] ebXML. http://www.ebxml.org/.

[18] ebXML: BPSS. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-bp.

[19] ebXML: Core Component. http://www.ebxml.org/specs/ccOVER.pdf.

[20] eclipse.org: Eclipse. http://www.eclipse.org/. (2001).

[21] 福田直樹, 和泉憲明, 山口高平: JADE を用いた Web Service 連携エージェントの設計と実装, *The 17th Annual conference of the Japanese Society for Artificial Intelligence* (2003).

[22] Hausmann, J.H., Heckel, Reiko. and Lohmann, Marc.: Model-based Discovery of Web Services, *Proceedings of 2004 IEEE International Conference on Web Services(ICWS 2004),* pp.324-331 (2004).

[23] 細井和宏：インターネット技術を利用した企業間 EC システム, CALS/EC Japan 1998 論文集, pp.475-480 (1998).

[24] IBM alphaWorks: JROM. http://www.alphaworks.ibm.com/tech/jrom. (2002).

[25] IBM corporation: IBM UDDI Business Registry Version2. http://uddi.ibm.com/ubr/.

[26] IBM corporation: IBM Test Registry. https://uddi.ibm.com/testregistry/.

[27] IBM developerWorks: UDDI4J. http://oss.software.ibm.com/developerworks/projects/uddi4j. (2001).

[28] IBM developerWorks: WS-Coordination. http://www-106.ibm.com/developerworks/library/specification/ws-tx/.

[29] IBM developerWorks: WSDL4J. http://oss.software.ibm.com/developerworks/projects/wsdl4j. (2003).

[30] IBM developerWorks: Web Services Security (WS-Security)Version 1.0. http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.

[31] JADE (Java Agent DEvelopment Framework). http://jade.cselt.it/.(2000).

[32] 金山隆志, 中村典浩, サナウォンサイプッタソン, 越田高志：SOAP/UDDI/WSDL による B2B システム構築の一事例, 情報処

理学会第 65 回全国大会論文集，1ZB-3 (2003).
[33] Kawamura, T., Hasegawa, T., Ohsuga, A. and Yamamoto, J.: Proposal of Semantics-based Web Service Matchmaking, *Proceedings of International Conference of Computational Intelligence and MultiMedia Applications 2001(ICCIMA' 2001)*, IEEE, pp. 87-92 (2001).
[34] 黒岩博孝：NEC の提案する企業間 EC ソリューション，CALS/EC Japan 1998 論文集，pp.485-490 (1998).
[35] 越田高志：VRML の CALS への適用，CALS Expo INTERNATIONAL 1997 Proceedings Track8, pp.91-95 (1997).
[36] 越田高志：VRML と Java による 3 次元動作シミュレーションの実現,CALS/EC Japan 1998 論文集，pp.503-508 (1998).
[37] 越田高志：VRML ライブラリ・サーバの構築と運用, CALS/EC Japan 1999 論文集 (CD-ROM CEJ-9911)，A3-4 (1999).
[38] 越田高志，金山隆志，中村典浩，サナウォンサイプッタソン：SOAP/UDDI/WSDL による B2B システムの開発，松江工業高等専門学校　研究紀要，第 38 号，pp.25-30 (2003).
[39] Koshida,T., Kaneyama, T., Hatano, K. and Uemura,S.: A Realization of the Dynamic Procurement Web Service in Practical Use Case, *Proceedings of the Second Workshop on e-Business (WeB 2003)*, pp.153-159 (2003).
[40] 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 (CD-ROM：ISSN 1349-1377)，D-9-5 (2004).
[41] Koshida,T. and Uemura,S.：An Effective Dynamic Invocation for Web Service, *Proceedings of the International workshop on Business and Information 2004 (BAI2004),* ISSN 1729-9322, Session3-3 (2004).

[42] Koshida,T. and Uemura,S.: Automated Dynamic Invocation System for Web Service with a User-defined Data Type, *Proceedings of the 2nd European Workshop on Object Orientation and Web Service (EOOWS 2004)*, pp.1-7 (2004).

[43] Koshida,T. and Uemura,S.: Unified Dynamic Invocation System for Web Services, *Proceedings of the Second International Conference on Computer Science and its Applications (ICCSA-2004),* pp.317-323 (2004).

[44] 越田高志, 植村俊亮: プリミティブ Web サービスとエージェントによる商品調達システムの設計と実装, *Proccedings of DBWeb 2004*, pp.9-16 (2004).

[45] 越田高志, 波多野賢治, 天笠俊之, 宮崎純, 植村俊亮：自動化・統合化された Web サービス動的実行システム, 情報処理学会論文誌, Vol.46, No.3, pp.671-682 (2005).

[46] Microsoft Corporation: Microsoft uddi Business Registry Node, http://uddi.microsoft.com/default.aspx.

[47] Mukhi, N.K. and Slominski, A.: The architecture of Web Service Invocation Framework, http://www-106.ibm.com/developerworks/webservices/library/ws- wsif2/. (2001).

[48] 日本ユニシス株式会社：XML Web サービス実験室. http://www.unisys.co.jp/net/index.html.

[49] OASIS. http://www.oasis-open.org/home/index.php.

[50] OASIS: BTP (Business Ttansaction Protocol). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=business-transaction.

[51] Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K.: Importing the Semantic Web in UDDI, *Proceedings of the Workshop on Web Services, E-Business and the Semantic Web*

(WES). Springer-Verlag Lecture Notes in Computer Science, No. 2512 (2002).

[52] Resource Description Framework (RDF). http://www.w3.org/RDF/. (1999)

[53] RosettaNet. http://www.rosettanet.org/RosettaNet/Rooms/DisplayPages/LayoutInitial.

[54] RosettaNet: Business Dictionary. http://www.rosettanet.org/rosettanet/Rooms/DisplayPages/LayoutInitial?Container=com.webridge.entity.Entity[OID[6EDB5FE87F69D411BD84009027E33DD8]]

[55] RosettaNet: PIP (Partner Interface Process). http://www.rosettanet.gr.jp/koukai/pip.html.

[56] ShaikhAli, A., Rana, O.F., Al-Ali, R. and Walker, D.W.: UDDIe -An Extended Registry for Web Services-, *Proceedings of UK e-Science All Hands Meeting 2003*, pp.731-735 (2003).

[57] SourceForge.net: SOAPUDDI-0.3.1. http://soapuddi.sourceforge.net/. (2002).

[58] The Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org/. (1996).

[59] 株式会社 東芝：Web サービス マッチメーカー. http://www.agent-net.com/.

[60] 高瀬俊郎：UDDI と WSDL, 情報処理, Vol.42, No.9, pp.870-887 (2001).

[61] 田中文雄：複合電子部品情報における用語・EDA ライブラリの標準化, CALS Expo INTERNATIONAL 1997 Proceedings (CD-ROM ACD-8591) , TS10-6 (1997).

[62] UDDI: The UDDI Technical White Paper, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, (2000).

[63] 渡辺南：情報共有型流通業企業間取引システムの実現，CALS Expo INTERNATIONAL 1997 Proceedings Track4, pp.75-80 (1997).
[64] Zip2Geo. http://ws.cdyne.com/ziptogeo/zip2geo.asmx? WSDL.

研究業績

学術論文誌

1. 越田高志, 波多野賢治, 天笠俊之, 宮崎純, 植村俊亮 :“ 自動化・統合化された Web サービス動的実行システム ”, 情報処理学会論文誌, Vol.46, No.3, pp.671-682 (2005).

国際会議発表（査読有）

1. Takashi Koshida, Takashi Kaneyama, Kenji Hatano and Shunsuke Uemura: “A Realization of the Dynamic Procurement Web Service in Practical Use Case”, Proceedings of the Second Workshop on e-Business (WeB 2003), pp.153-159, December 2003.

2. Takashi Koshida and Shunsuke Uemura: “An Effective Dynamic Invocation System for Web Service”, Proceedings of International Workshop on Business and Information (BAI 2004), Session III-3, CD-ROM (ISSN 1729-9322), March 2004.

3. Takashi Koshida and Shunsuke Uemura: “Automated Dynamic Invocation System for Web Service with a User-defined Data Type”, Proceedings of the 2nd European Workshop on Object Orientation and Web Service (EOOWS 2004), pp.1-8, June 2004.

4. Takashi Koshida and Shunsuke Uemura: “Unified Dynamic Invocation System for Web Services”, Proceedings of the 2nd International Conference on Computer Science and its Applications (ICCSA-2004), pp.301-307, June 2004.

国内発表（査読有）

1. 越田高志, 植村俊亮:“プリミティブ Web サービスとエージェントによる商品調達システムの設計と実装”, データベースと Web 情報システムに関するシンポジウム (DBWeb2004), Proceedings of DBWeb 2004, pp.9-16, 情報処理学会, 2004 年 11 月.

全国大会口頭発表

1. 金山隆志, 中村典浩, サナウォンサイ・プッタソン, 越田高志:“SOAP/UDDI/WSDL による B2B システム構築の一事例”, 情報処理学会第 65 回全国大会講演論文集 CD-ROM, 1ZB-3, 2003 年 3 月.

2. 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, CD-ROM (ISSN: 1349-1377), 電子情報通信学会, 2004 年 3 月.

その他

(著書：分担執筆)

1. 越田高志, 北村友博, 柏原秀明, 木本善朗, 安福千尋:“CIM における実践的データベースシステム構築法”,（財団法人）大阪科学技術センター出版書籍, 1995.

(学会発表：査読有)

1. 越田高志:“VRML の CALS への適用”, CALS Expo INTERNATIONAL 1997 Proceedings (Track8), pp.91-95 (1997).

2. 越田高志:“VRML と Java による 3 次元動作シミュレーションの実現”, CALS/EC Japan 1998 論文集, pp.503-508 (1998).

3. 越田高志：“ VRML ライブラリ・サーバの構築と運用 ”, CALS/EC Japan 1999 論文集（CD-ROM CEJ-9911), A3-4 (1999).

（研究紀要）

1. 越田高志, 金山隆志, 中村典浩, サナウォンサイ・プッタソン：“ SOAP/UDDI/WSDL による B2B システムの開発 ”, 松江工業高等専門学校研究紀要, No.38, CD-ROM(ISSN:1347-037X), pp.25-30, 2003 年 2 月.

（特許出願）

1.「回路シミュレーション装置」, 共同出願・筆頭者, H01207836, 1989 年 8 月.（連名者： 八木田秀樹, 南部修太郎）

2.「半導体増幅回路装置」, 共同出願・筆頭者, H01259216, 1989 年 10 月.（連名者：八木田秀樹, 中塚忠良, 反保敏治, 南部修太郎）

3.「半導体素子のマスクデータ生成装置」, 単独出願, H04022297, 1992 年 2 月.

4.「バイポーラ IC 抵抗識別装置」, 単独出願, H04027741, 1992 年 2 月.

5.「半導体集積回路論理接続データ変換装置」, 単独出願, H04027742, 1992 年 2 月.