第 1 章

はじめに

Web サービス[1]とはインターネットを利用して異なるシステムに接続、連携させる新たな標準技術であり、最も注目されている技術の一つである。今後 Web サービスの普及によって、サービスの利用者、提供者がますます増えていくことが予想される。膨大な数になると、サービスの発見・登録・管理する仕組みが必要となる。

今 Web の世界では、Google などの検索エンジンがもうたくさんの人にとって、欠かせない重要なツールになった。だが、文書ベースとしての Web サイトの検索と Web サービスの検索は本質的に違っているので、検索エンジン技術は Web サービスの検索に向いてないと考える。

Web サイトの検索の最終目的物は“知識”である。つまり人々はある知識を得るため、検索エンジンを利用して、この“知識”を乗せている Web サイトを探し出す。検索結果をゲットした時点で、目標が達成する。しかし Web サービスには違う、人々はサービスを利用するため、たくさんのサービスのなか自分に適切なものを探す。ですから、検索結果ではなく、サービスの利用することは最終目的である。例えば、私たちは旅行代理店で航空券を買うとき、目的地、出発日と帰り日、クラスなどを申して、代理店の人は各航空会社の空席情報をまとめて、どの航空会社を利用した方がいいかと教えてくれるだけではなく、直接航空券を買ってくれる。検索また選択はサービスの利用から離れるではなく、サービスの利用の重要な一環として、円滑的に利用者に提供することになる。

もちろん、利用者は事前にサービスを検索して、自分で提供者と接触しったり、第三者に意見を聞いたり、評価してから、利用するという形態もあるが、本研究の考察対象はサービスを利用する時点でつまり実行時動的にサービスを検索して、適切なサービスを選択する手法である。

以下の時点で、動的に検索することを求められると考えている。

- 利用者はサービスを利用することを決め、最適なサービスを求める場合
- サービスの内容、利用条件が動的に変わっている場合
- サービスは動的に組み合わせて、複合サービスとして提供できる場合

そして、検索結果は条件により毎回変わるかもしれない、利用者としてはやり取りの相手は不特定になる。しかし、今の Web サービスの仕組みより、利用する前に、利用者側はサービスのインターフェースより、接続用のクライアントプログラムを作らないと利用できない。また同じ内容のサービを提供しても、提供者が違うと、採用するインターフェースはそれぞれ違う可能性が高い。そのため、利用者は適正なサービスを探し出しでも、このサービスへの接続プログラムを作らないと、利用できない。全てのサービスへの接続プログラムを作るのは不可能なので、検索範囲は自分が接続を実装したサービスだけで、この数が少ないと、いい検索結果を得られないかもしれない、多くの接続プログラムをつくると、大変な負担になる。

本研究では、自動選択を実現するため、サービスブローカを導入する手法を提案する。サービスブローカは以下の機能を実現している

- サービス提供者への接続プログラムを実装する
- 統一したインターフェースで利用者にサービスを公開する
- サービスを検索する用のメソッドを提供する
- 利用者の希望より、適切なサービスを選択し、利用させる

それによって、利用者はブローカが提供したサービスを経由して、動的に適

切なサービスを選んで利用することができる。

本論文の構成を説明する。2 章では Web サービスに関連する技術仕様と自動選択についての既存研究を説明する。前者については Web サービスの基本要素 SOAP[2]、WSDL[3]、UDDI[4]についての説明し、Web サービスと深く関わっているセマンティック Web[5]技術について述べる。

第 2 章

研究背景

本章では Web サービスに関連する技術仕様と自動選択についての既存研究を説明する。

2.1 Web サービス

前章で説明したように Web サービスとはインターネットを利用して異なるシステムに接続、連携させる新たな標準技術である。Web サービスは以下の特徴がある：

- 標準のデータ形式である XML で送受信する
- 通信プロトコルは SOAP（下位レイヤーのトランスポート層は HTTP、SMTP、MQ 等）
- 標準仕様による通信なので、プラットフォームや言語に依存しない
- Web サービス提供側は、Web サーバが必要
- 利用側は XML/SOAP インターフェースを持つ Web クライアントであればよい
- Web サービスのインターフェースは、WSDL という文法で XML 形式で記述され、Web サービス提供サイトで公開される。利用側では、WSDL を参照してアプリケーションを作成する

また Web サービスに関する情報の登録、公開、及び検索を実現するレジストリを利用可能とする定義も存在する。

Web サービスの利用の手順は図 2.1 に示す。

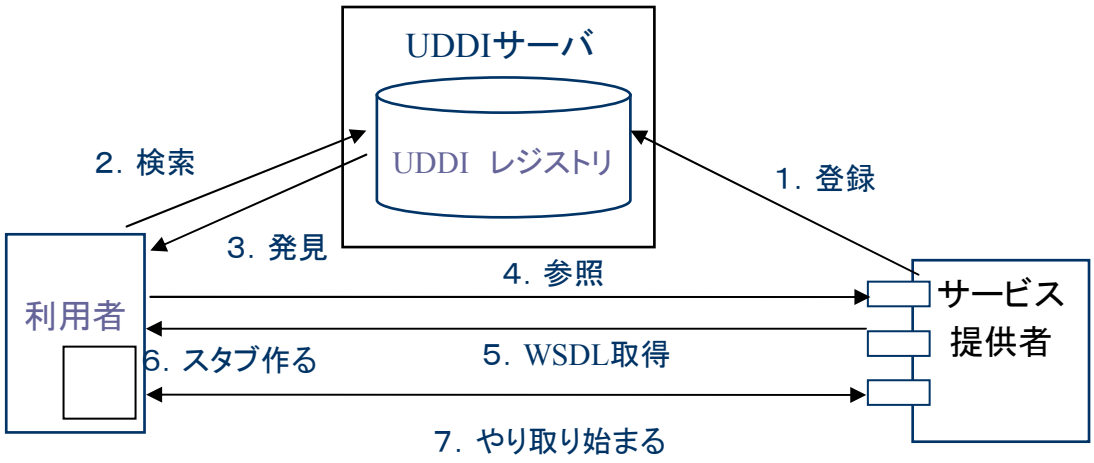

図 2.1 Web サービスの利用する手順

サービス提供者は、サービスへ送信するデータの送り先とインターフェースなどのサービスに関する情報に関する情報を UDDI レジストリに登録すし、公開する。

サービス利用者は UDDI レジストリからサービスを検索、発見する。サービスの提供者からサービスのインターフェースを記述した WSDL 文書を取得し、接続用のスタブプログラムを作って、所望のサービスを利用し始まる。

2.1.1　Web サービスの課題

Web サービスの課題には技術的な課題とビジネスの課題がある。技術的な課題はサービスの品質、性能の確保方法、セキュリティ対策といった問題である。また複合サービスの連携、トランザクションの管理について競合仕様の林立、仕様の標準化しづらい、また仕様についてベンダー間で解釈の違う問題もたくさんある。

ビジネスの課題は信頼性を保証するため、企業与信情報の公開、サービスについて認証の問題、相互接続性を向上させるため、業界の常用用語の標準化の課題。オンラインでのダイナミックな契約、決済方法についての問題などが取り上げられる。

2.2 Web サービスにおける技術仕様

Web サービスは基本的に、WSDL、SOAP、UDDI といった仕様で構成されている。SOAP は Web サービスが送受信するメッセージのフォーマットに関する仕様である。WSDL は Web サービスのインターフェースを記述するための XML 形式の言語である。UDDI は Web サービスに関する情報の登録、公開、発見を実現するレジストリの仕様である。

現在では、実際のビジネスで Web サービスを利用するためのさまざまな要求にこたえるため、これらの基本仕様の上に、セキュリティ、トランザクション、コレオグラフィといった幅広い拡張仕様が存在し、または仕様策定のための議論が行われています。これらの仕様は多くの企業や W3C や OASIS などの標準化団体によって策定されている。

以下は SOAP、WSDL、UDDI に関する説明を行う。

2.2.1 SOAP

SOAP で規定しているのは、メッセージのフォーマットつまり、郵便でいうところの封筒の仕様である。封筒には、宛名や差出人を書く、複数の書類を束ねる、中身が見えないようにする、中身が改ざんされるのを防ぐ、中身が汚れるのを防ぐ、といった具合である。

SOAP メッセージの構造は以下のようになっている。

- SOAP エンベロープ:SOAP メッセージの一番外側の要素となる。SOAP ヘッダと SOAP ボディで構成される
- SOAP ヘッダ：メッセージの受信者（処理する人）に対して、どう扱う

べきか、ということを知らせるために使う

- SOAP ボディ : 送受信するメッセージの中身とメッセージの取り扱い中に発生したエラーを記述する

この構造は以下図 2.2 のようになっている。

また、SOAP エンベロープでは、ヘッダ部に付加情報を格納することができる。WS-Security や WS-Transaction では、この仕組みを利用してセキュリティやトランザクションの情報を伝搬している。

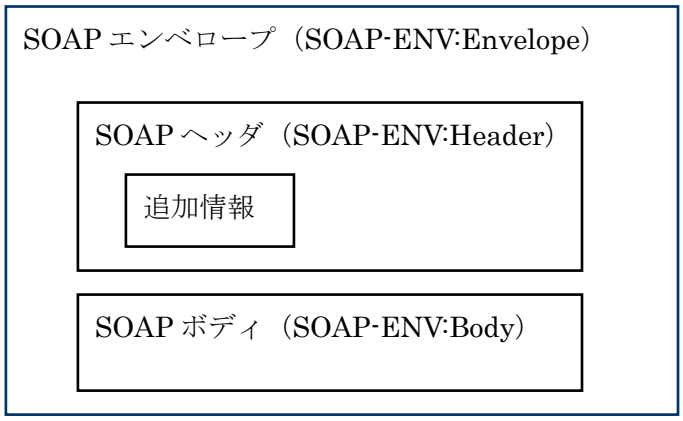

図 2.2 SOAP エンベロープ構造

2.2.2　WSDL

WSDL は Web サービスを利用するため、必要な情報を記述する XML ベースの言語である。WSDL に記述された具体的な内容は、そのサービスがどのようなインターフェースで、どのようなプロトコルで、どこで提供されているのかなどの情報である。

WSDL 文書は決められたスキーマで、これらの情報を記述する。このスキーマは以下の要素を含まれている。

- types 要素：下位のデータタイプを定義する。具体的にはスキーマ定義

が入れられる

- message 要素：送信データフォーマットを定義する
- portType 要素：message 要素で定義された送信データフォーマットをいくつか組み合わせて、1 つの操作単位を論理的に定義する
- binding 要素：types、message、portType などで定義されたインターフェースの論理的なモデルと、物理的なモデルとを結び付ける定義をする
- port 要素：port 要素では、どの binding 要素のインターフェースを、どの URL で提供するかを定義する
- service 要素：port 要素で定義したポートのうち、関連するポートをひとまとめにしたサービスを定義する
- operation 要素：入力と出力とフォルトメッセージ出力を行う処理の 1 単位である操作（operation）を抽象的に定義する

2.2.3　UDDI

UDDI（Universal Description, Discovery and Integration）はサービスの情報を登録・検索するためのレジストリ仕様です。UDDI が定めている仕様の中心になるのは、UDDI ビジネスレジストリである。その中では Web サービスを検索して利用するときに必要となる情報を以下の 4 つのカテゴリに分けている。

- ビジネス情報（businessEntity）：企業体などを表すためのものだ。企業名、連絡先、企業を特定（確認）するための何らかの ID 番号などを記述する。

- 業務情報（businessService）: 企業がどのような業務を提供しているかを記述する。例えば業種、製品/サービス名、場所などの情報を挙げられる
- サービス記述（tModel）: Web サービスの仕様を記述するためのメタデータである。名前、仕様を策定した団体、仕様への URL などが記述される
- バインド情報（bindingTemplate）: Web サービスを提供している URL と<tModel>要素への参照によって、どの Web サービスを使うかとの情報を記述される

2.3　セマンティック Web

セマンティック Web はページに「メタデータ」を付加することによって、コンピュータがページ上の情報を理解できるように、Web を再構築しようという構想です。メタデータとは文書に記載されている内容そのものではなく、その構造や属性に関する情報のことです。

図 2.3 のようにセマンティック Web は 9 階層の技術階層に分けて、順次開発が進められている。

2.3.1　セマンティック Web と Web サービス

動的適切な Web サービスを探し出して、処理を自動的に遂行することを実現するためには、HTML や WSDL、UDDI に、ソフトウェアが人間の介在なしに理解できるセマンティックス（意味）を付加していく必要があります。

例えば、A 社と B 社が同じサービス提供しているが、住所を記述する項目名は A 社が「住所」、B 社が「宛先」をしかってしまう。人間はこの二つの項目は同じ意味をしているのを理解できるが、コンピュータはこれを理解できなく、二つ違う項目として処理せざるを得なかった。

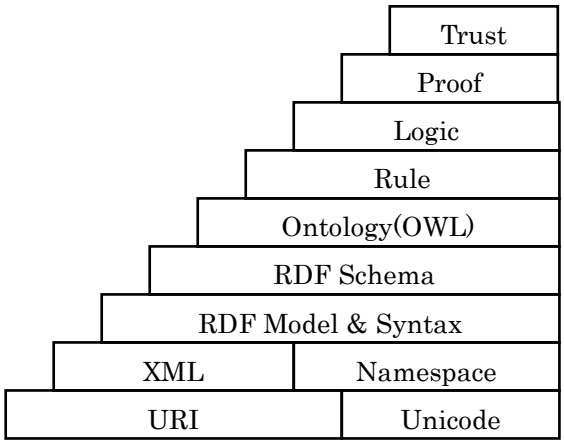

図 2.3 セマンティック Web 技術階層

人の手を使わずにデータを機械的に処理できる空間を創出するため、セマンティックスを定義するためのメタデータ記述言語 RDF（Resource Description Framework）と意味付けのための語彙を定義するための言語 OWL（Web Ontology Language）の策定作業に取り組んでいる。さらに、Web サービスのための OWL オントロジ記述言語の上位オントロジとして OWL-S の策定も挙げられた。

2.3.2 RDF/ RDF スキーマ

RDF は文書に付加された構造を記述するメタデータ仕様である。単純な SVO（主語，動詞，目的語）の三つ組（Resource，Property，Value）で記述され，意味ネットワークとして定義することができる。

RDF だけでは，複雑な表現をつくるのは難しいため，RDF をベースに，クラス概念を持った階層的表現が可能なように拡張したものが RDF スキーマである。これによりオブジェクト指向的に意味情報の関係を記述できる。

2.3.3 OWL

Web で相互運用するために、各言語における言葉の意味を関連付ける「オントロジー （ontology）」が必要になる。Web における「オントロジー」とは、

言葉がどのように使用されているか、どのような知識を表現しているか定義するものである。OWL (Web Ontology Language)は、RDF によって定義された言語間の関係を定義する言語である。OWL によって、複数のメタデータが統一化され、Web 全体が 1 つの知識データベースとして取り扱えるようになる。

2.3.4 OWL-S

OWL-S による Web サービス記述には，プロフィール、プロセス、グラウンディングの三つの側面がある。Web サービスを利用するエージェントは、Web サービスの発見、実行、合成、相互作用を遂行する。ここで、発見にはサービスプロフィール、実行にはサービスプロセスとグラウンディング、合成と相互作用にはサービスプロセスが関係する。

2.4 Web サービスの自動選択

Web サービスの普及により、サービスの利用者、提供者がますます増えていくことが予想される。同一機能を持つ複数のサービスが提供される可能性が高くなってくる。また利用者はそれぞれの要求を持って、自分の目的を応じたサービスを選択し、利用する要求が出てくると考えられる。

前述したように、Web サービスを利用する一般な手順は利用者が UDDI でサービスの検索を行い、得られた情報を基づいて、手動で目的のサービスを決定しなければならない。目的サービスを決めたら、サービスのインターフェースなどの情報を提供者からもらって、接続用のプログラムを作って、利用し始まり。

ここで、二つの問題点がある。

- オープン UDDI レジストには誰でもサービスの登録ができるので、UDDI に公開された情報を基づいて、利用者は本当に信用してサービスを利用できるかが分からない
- UDDI で公開されたのは企業名、業務名、業種などの静的情報だけで、サービスに関する動的な情報を反映されない、そして、実行時検索を行

って、動的に適切なサービスを選択して、利用するのは不可能だ

利用者に対して、理想的なサービスを選択、利用するとして以下の要求を認められると考えられる。

- リアルタイムで検索を行う。最新状況を応じて、選択を行って、適切なサービスを利用する
- サービスに対し、過去の利用情報、人気度など、第三者からの評価を参考できる
- 一つのクライアントプログラムですべてのサービスを利用できる
- 新しくできたサービスを自動で発見できる

2.4.1 既存研究

これらの問題を解決するため、[1]では属性及びレーティングに基づく Web サービスの自動選択手法を提案された。この手法によって、利用者が同一インターフェースを持つ Web サービスを利用候補とする。また評価基準となるサービスの属性と評価するためのレーティングアルゴリズムを定義しておく。サービスを利用するとき、候補としたサービスの属性のデータをリアルタイムで収集し、評価する。評価結果により、適切なサービスを選択して利用する。利用する手順は図 2.4 に示す。

しかしこの手法は以下の不足点がある。

- この手法は各提供者が統一したインターフェースでサービスを提供することを前提条件としている。違うインターフェースで提供されたサービスがあったら、適用できなくなる。また、この業界について Web サービスの標準化作業を行ってない場合、各提供者は完全に統一したインターフェースを利用することは難しいである。

- 新しくできたサービスを利用するため、常に UDDI へ最新サービスを検索し、インターフェースの照合、サービスを候補リストに追加するなどの作業を行わなければならない。
- この手法では、評価用の属性がサービスの品質、人気度など第三者が回答すべき属性の場合、提供者は委譲機関へのリンクを利用者に提供して、利用者はこのリンク先をアクセスして、属性値が回答される。しかし、それぞれの提供者はそれぞれの委譲機関を利用すると、統一した基準がないので、属性値を回答されても、参考にならない。

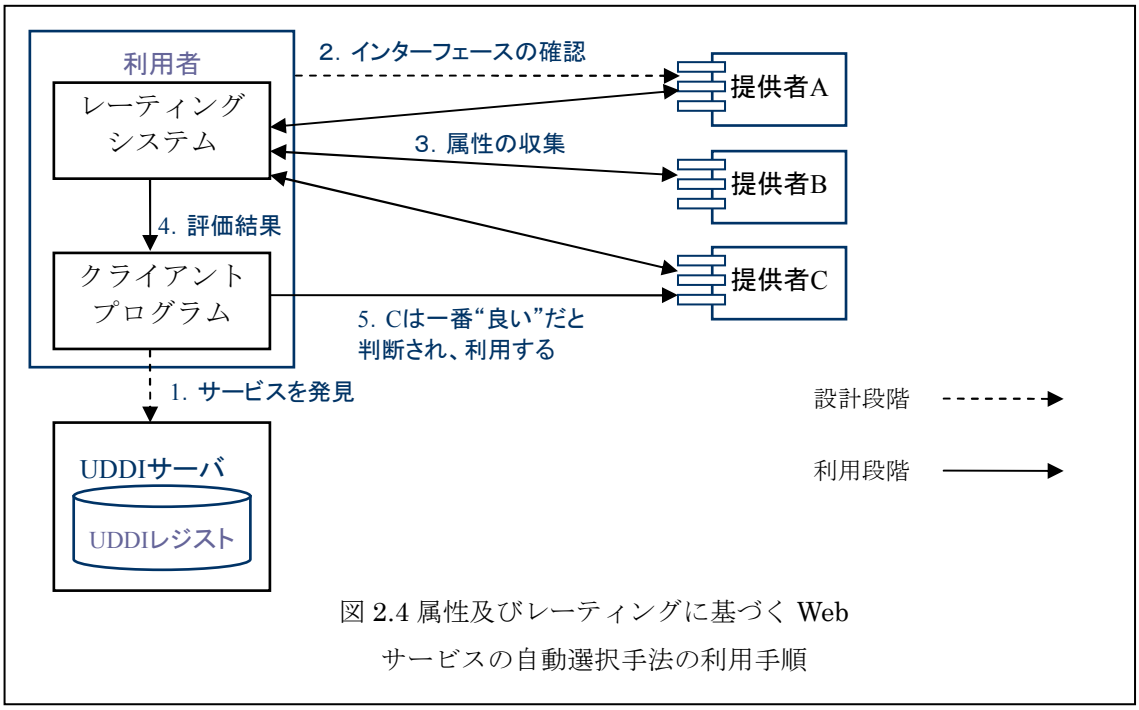

図 2.4 属性及びレーティングに基づく Web サービスの自動選択手法の利用手順

第 3 章

サービスブローカにおける自動選択

本章ではサービスブローカを導入するよりサービス自動選択を実現するの手法について説明する。

3.1 サービスブローカの導入

本手法では、サービスブローカが同じ機能を提供しているサービスを探し出し、各サービスへの接続プログラムを作って、それぞれのインターフェースをまとめて、統一したインターフェースで、実行サービスとして利用者へ提供する。利用者は直接サービスへの接続を持たず、サービスブローカの経由でサービスを利用する。

またサービスの評価、選択の作業は利用者自分自身で行うとサービスブローカに委託二つの選択肢がある。

3.1.1 利用者による選択

評価、選択作業は利用者自分自身で行う場合、まず、サービスブローカは利用者が関心を持つサービスの属性を定義し、これらの属性の問い合わせに応じての回答処理を属性収集 Web サービスとして利用者に提供する。さらに、属性値の回答処理を実装するとき、この属性はサービス提供者元に問い合わせを行うか、サービスブローカが第三者の立場で属性値を回答するかを分けて、それぞれの実装を行う。

利用者はブローカが提供した、属性収集サービスと実行サービスへの接続プログラムを作って、また、属性評価、選択するアルゴリズムを実装する。

利用するとき、利用者はブローカが提供した属性収集サービスを利用して、各サービスの属性値を取得し、また評価、選択プログラムを行って、どのサービスを利用することを決めて、実行サービスを利用してサービスを利用する。

最後は利用状況をデータベースに保存する。これらのデータは前述した第三者の立場で属性について値を回答する場面で用いられる。さらに、各サービスの人気度のランキング、利用者の好みの分析などのデータマイニングを行って、新しいサービスやビジネスを起こすことが可能である。

利用する手順は図 3.1.1 に示す。

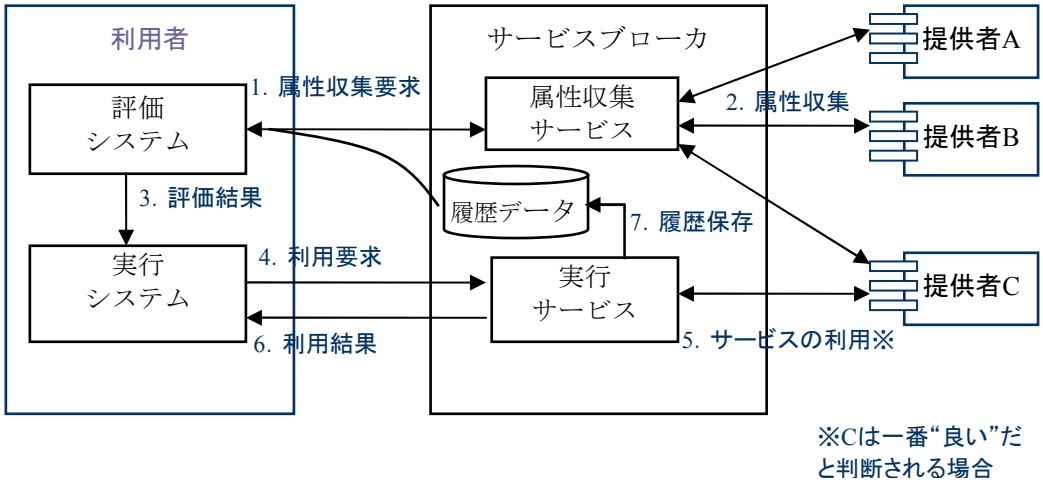

図 3.1.1 利用者が選択を行う手法

3.1.2　サービスブローカによる選択

評価、選択作業はサービスブローカに委託する場合、まずサービスブローカが前述したように属性を定義し、属性収集プログラムを実装し、これらの属性をパラメーターとして Web サービスを構築する。利用者はこの Web サービス

を経由して、実際の各サービスを利用する。また、サービスブローカは利用者を代わりに評価、選択システムを構築しなければならない。

利用するとき、利用者はサービスブローカが提供した Web サービスのインターフェースに従って、自分の要求をサービスブローカに渡す。サービスブローカは各サービスの属性値の収集を行い、受けた利用者の要求と照らし合わせて、評価、選択システムを利用して、適切なサービスを選び出して、サービスを実行する。実行した結果を利用者に渡す。

利用する手順は図 3.1.2 に示す。

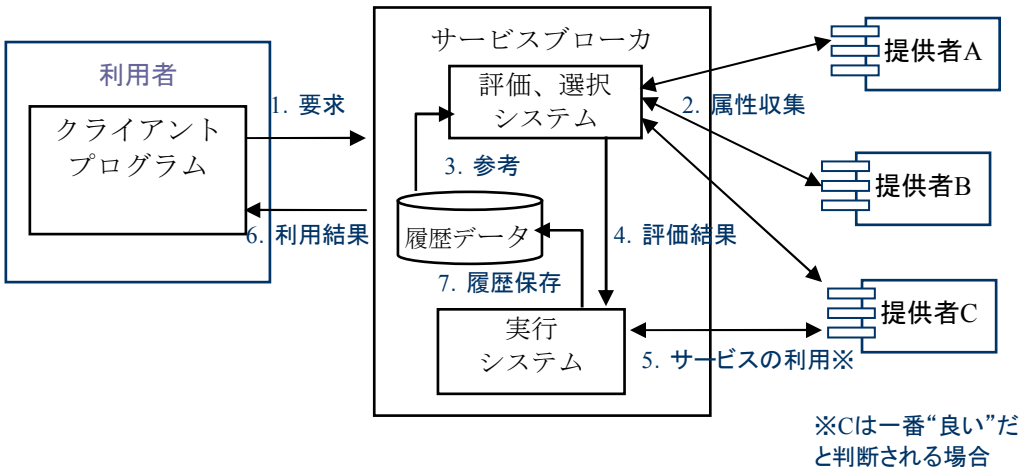

図 3.1.2 サービスブローカが選択を行う手法

3.1.3　適用場面

前述二つの選択肢はそれぞれの適用場面がある。どのモデルを利用するのか、該当サービス或いは業界の特性と深く関係ある。

あるサービスに対して、それぞれの利用者はそれぞれの要求を持つ場合、また同じサービスの属性値について、利用者はそれぞれの価値観を持って、まったく違う評価結果が出る場面が多い場合、サービスブローカは統一した評価、選択ロジックを使ってサービスを選び出すのは難しい。この場合、各利用者は自分の評価ロジックを実装して、選択を行った方が良いと考えられる。

しかし、このモデルでは、利用者自分で評価ロジックを実装するだけではなく、サービスブローカへの属性収集サービスと実行サービスを二回呼び出す必要になる。使い勝手やコストの面では良くない。

そして、サービスに対して、属性が単純で、また殆どの属性に対し、利用者が同じ価値観を持つ場合は（例：値段ついて、殆どの利用者は低い方が良いと判断すると考えられる）、サービスブローカ側で選択を行った方が良いと考えられる。

3.2　本手法の利点

サービスブローカを導入するより、以下の利点がある。

- 利用者はサービスブローカだけへの接続プログラムを実装すれば、複数のサービス提供者とのやり取りが可能になる。
- 手間もコストも取らず取引先を拡大できる。
- ブローカは検索、取引の履歴データ蓄積し、第三者の立場でサービスに対して評価することが可能になる。

さらに、第三者であることで、サービスブローカは一方的に利用者の立場でサービスを仲介することに限られるではなく、サービス提供者の視点から、仲介することもできる。これにより、両者の利益を考慮して、市場全体の状況より、最適化にしたやり取りを結びつくことができるようになる。

3.3　インターフェースの統合

この手法を利用する前提条件として、各サービスのインターフェースを統合できないといけない。理想的な状況は同じ機能を持つサービスが統一したインターフェースを持っているが、現実的には難しい、また同じインターフェースを持っても、全ての項目の意味が必ず一致しているのも限らない。例えば、まったく同じ機能を提供しているサービス二つがあって、また同じ float 型の price という項目を持っている、しかし、一つは日本円、一つは US ドルを単位とし

ている場合、同じの使い方でこの二つのサービスを利用すると、大変な差が出る。そして、サービスブローカとして、各サービスを統合するため、やはり人の介在が必要である。

第 4 章 応用実例

本章では花卸売市場システムを作って、実例として本手法を説明する。

4.1　システム概要説明

このシステムの提供者は卸売会社のような仲介者つまり本手法のブローカである。このシステムの利用者は、花屋さんなどの小売店（買い手）と花の産地の人（売り手）である。花セリと同じく、このシステムは毎日定刻で行い、買い手から受けた注文情報（値段、数量）と売り手の提供情報を照らし合わせて、取引をまとめる。

4.2　原則

ブローカが買い手に対し、注文の Web サービスを公開する。買い手はこれを利用して買いたい花の種類、支払い可能な最高の値段、数量を提出する。売り手も提供可能な最低値段、数量を Web サービスで公開する。決めた時間になったら、ブローカは当日の注文サービスを一旦止めて、受けた注文を花ごとにグループ化する。同じ花を注文する買い手を一つのグループにして、提出値段でソートする。また花の種類をパラメータとして、売り手に問い合わせを行う。取得したのは売り手の値段、数量と注文データをあわせて、以下の原則に従って引き取りを結びつく。

- より高い値段で注文した買い手は買う優先権がある。
- 低く売る売り手は売る優先権がある。
- （1）と抵触しない限り、高い値段で注文した買い手はより高い値段で

買う。

- （2）と抵触しない限り、低く売る人は低い値段で売る。

4.3 ソースコード

4.3.1 Broker

ブローカを代表するクラス。

```
public int checkBuyerIn()
{
  int i,j = 0;
  int notEnough = 0;

  for(i = 0; i < buyers.length; i++)
  {
    notEnough = checkQuantity(i);
    if (notEnough > 0){
      while (notEnough > 0){
        buyers[i].quantity -= notEnough;

        if (buyers[i].quantity <=0 )
          break;
        notEnough = checkQuantity(i);
      }
      break;
    }
  }
  return i;
}
```

4.3.2 Buyer

購買者を代表するクラス。

```
public int minusQuantity(int quantity)
{
  if (this.quantity > quantity)
  {
    this.quantity -= quantity;
  }
  else{
    this.quantity = 0;
  }
  return this.quantity;
}

public int buyFrom(Saler saler, int quantity) throws CalcException
{
  if (saler.getQuantity() < quantity)
    throw new CalcException("不正な取引：購入量＞売却量");

  if (quantity > this.quantity)
    throw new CalcException("不正な取引：購入量＞所要量");
      if (Broker.priceFlg == 1)
    saler.sale(this.price,quantity);
  else
    saler.sale(saler.price,quantity);
```

```
    this.quantity -= quantity;
    return this.quantity;
  }
```

4.3.3 Saler

販売者を代表するクラス。

```
/**
 * @return price を戻します。
 */
public double getPrice() {
	return price;
}
/**
 * @param price price を設定。
 */
public void setPrice(double price) {
	this.price = price;
}
/**
 * @return quantity を戻します。
 */
public int getQuantity() {
	return quantity;
}
```

```
/**
 * @param quantity quantity を設定。
 */
public void sale(double price,int quantity)
{
        this.quantity = this.quantity - quantity;
}
```

4.4 サービス

4.4.1 ServiceA

サービス A

```
public class ServiceProviderA {
  static String srcFile = “”;

  public ServiceProviderA() {
  }

  public boolean buy(String itemID, int num){
    boolean rtn = true;
    return rtn;
  }

  public Object[] searchByItem(String itemID){
    String[] rtn = null;
    Vector search = new Vector();
    String tmp = null;
    try{
      File file = new File(srcFile);
      FileReader fd = new FileReader(file);
      BufferedReader brd = new BufferedReader( fd );
      while ((tmp = brd.readLine()) != null){
        //System.out.println(tmp);
        if (tmp.indexOf(itemID) == 0){
```

```
            search.add(tmp.substring(tmp.indexOf(",") + 1));
            System.out.println(tmp.substring(tmp.indexOf(",") + 1));
          }
        }
        return (Object[])(search.toArray());
      }
      catch(IOException e){
        e.printStackTrace();
        return null;
      }
    }
    public static void main(String args[]){
      ServiceProviderA sp = new ServiceProviderA();
      Object[] rtn = sp.searchByItem("a");
      System.out.println(rtn[1]);
    }
}
```

4.4.2 ServiceB

サービス B

```
public class ServiceProviderB {
  static String srcFile =
"D:¥¥chou¥¥project¥¥ws¥¥serviceProvider¥¥src¥¥serviceprovider¥¥itemsB.
csv";

  public ServiceProviderB() {
  }

  public boolean buy(String itemID, int num){
    boolean rtn = true;
    return rtn;
  }


  public Object[] searchByItem(String itemID){
    String[] rtn = null;
    Vector search = new Vector();
    String tmp = null;
    try{
      File file = new File(srcFile);
      FileReader fd = new FileReader(file);
      BufferedReader brd = new BufferedReader( fd );
      while ((tmp = brd.readLine()) != null){
        //System.out.println(tmp);
```

```
          if (tmp.indexOf(itemID) == 0){
            search.add(tmp.substring(tmp.indexOf(",") + 1));
            System.out.println(tmp.substring(tmp.indexOf(",") + 1));
          }
        }
        return (Object[])(search.toArray());
      }
      catch(IOException e){
        e.printStackTrace();
        return null;
      }
    }
    public static void main(String args[]){
      ServiceProviderB sp = new ServiceProviderB();
      Object[] rtn = sp.searchByItem("a");
      System.out.println(rtn[0]);
    }
}
```

4.4.3 ServiceC

サービス C

```
public class ServiceProviderC {
 static String srcFile = ¥¥itemsC.csv″;

 public ServiceProviderC() {
 }

 public boolean buy(String itemID, int num){
   boolean rtn = true;
   return rtn;
 }


 public Object[] searchByItem(String itemID){
   String[] rtn = null;
   Vector search = new Vector();
   String tmp = null;
   try{
     File file = new File(srcFile);
     FileReader fd = new FileReader(file);
     BufferedReader brd = new BufferedReader( fd );
     while ((tmp = brd.readLine()) != null){
       //System.out.println(tmp);
       if (tmp.indexOf(itemID) == 0){
         search.add(tmp.substring(tmp.indexOf(″,″) + 1));
```

```
            System.out.println(tmp.substring(tmp.indexOf(",") + 1));
          }
        }
        return (Object[])(search.toArray());
      }
      catch(IOException e){
        e.printStackTrace();
        return null;
      }
    }
    public static void main(String args[]){
      ServiceProviderC sp = new ServiceProviderC();
      Object[] rtn = sp.searchByItem("a");
      System.out.println(rtn[0]);
    }
}
```

4.4.4 Stub

```
package stubA;

public interface ServiceProviderA extends java.rmi.Remote {
    public boolean buy(java.lang.String in0, int in1) throws java.rmi.RemoteException;
    public java.lang.Object[] searchByItem(java.lang.String in0) throws java.rmi.RemoteException;
}
```

4.5 サービス設定

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
  <display-name>Apache-Axis</display-name>
  <servlet>
    <servlet-name>AxisServlet</servlet-name>
    <display-name>Apache-Axis Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AxisServlet
    </servlet-class>
  </servlet>

  <servlet>
    <servlet-name>AdminServlet</servlet-name>
    <display-name>Axis Admin Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AdminServlet
    </servlet-class>
    <load-on-startup>100</load-on-startup>
  </servlet>
```

```
<servlet>
   <servlet-name>SOAPMonitorService</servlet-name>
   <display-name>SOAPMonitorService</display-name>
   <servlet-class>
         org.apache.axis.monitor.SOAPMonitorService
   </servlet-class>
   <init-param>
      <param-name>SOAPMonitorPort</param-name>
      <param-value>5001</param-value>
   </init-param>
   <load-on-startup>100</load-on-startup>
</servlet>

<servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>/servlet/AxisServlet</url-pattern>
</servlet-mapping>

<servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>*.jws</url-pattern>
</servlet-mapping>

<servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>/services/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
```

```
    <servlet-name>SOAPMonitorService</servlet-name>
    <url-pattern>/SOAPMonitor</url-pattern>
  </servlet-mapping>

 <!-- uncomment this if you want the admin servlet -->
 <!--
  <servlet-mapping>
    <servlet-name>AdminServlet</servlet-name>
    <url-pattern>/servlet/AdminServlet</url-pattern>
  </servlet-mapping>
 -->

    <!-- currently the W3C havent settled on a media type for WSDL;
    http://www.w3.org/TR/2003/WD-wsdl12-20030303/#ietf-draft
    for now we go with the basic 'it's XML' response -->
  <mime-mapping>
    <extension>wsdl</extension>
      <mime-type>text/xml</mime-type>
  </mime-mapping>


  <mime-mapping>
    <extension>xsd</extension>
    <mime-type>text/xml</mime-type>
  </mime-mapping>

  <welcome-file-list id="WelcomeFileList">
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.jws</welcome-file>
```

```
  </welcome-file-list>

</web-app>


<deployment xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="urn:ServiceProviderA" provider="java:RPC">
    <parameter                                    name="className"
value="serviceprovider.ServiceProviderA" />
    <parameter name="allowedMethods" value="*" />
    <parameter name="scope" value="session" />
  </service>

</deployment>
```

4.6 結果

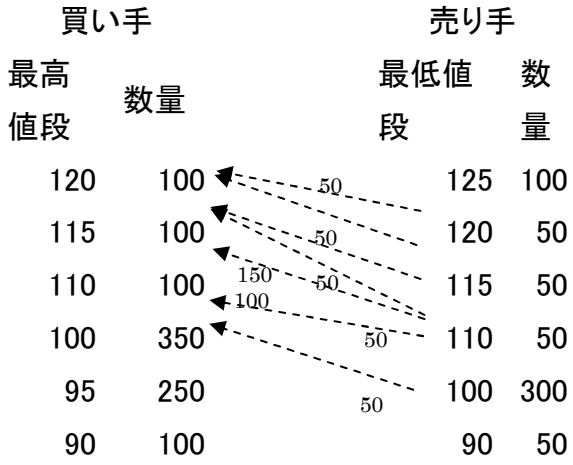

図 1 自動検索結果

図 1 は実際のデータを使って実行した結果である。その結果に基づき、ブローカは買い手の名義で各売り手の注文のメソッドを呼び出し、自動的に取引を成立させる。

第 5 章

おわりに

本論文では Web サービス自動選択の背景を紹介して、Web サービスの検索また選択の技術の現状、未解決問題点について説明した。サービス自動選択を実現するための既存研究を紹介し、この研究の不足点について述べて、サービスブローカを導入する手法を提案した。次には該当手法について二つの応用モデルが存在することとそれぞれの利用手順、適用場面を説明した。また該当手法の利点を述べた。さらに該当手法の前提条件としてサービスのインターフェースを統合する問題について説明した。最後は実例として、シミュレーションでこの手法の利点を示す。

謝辞

本研究を進めるにあたり、数々のご指導を頂いた深澤良彰教授、修士 1 年目の研究指導者であり去年卒業した山根一樹さんに深く感謝致します。また、多大なる御助言、御助力を下さった深澤研究室の皆様に感謝致します。

参考文献

[1] Web サービス
http://www.w3.org/2002/ws/

[2] SOAP 1.2
http://www.w3.org/TR/soap/

[3] Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

[4] Universal Description, Discovery, and Integration
http://www.uddi.org/whitepapers.html

[5] Semantic web
http://www.semanticweb.org

[6] 神崎正英
メタ情報とセマンティック・ウェブ
http://www.kanzaki.com/docs/sw/

[7] RDF
http://web.resource.org/rss/1.0/spec

[8] OWL-S
http://www.daml.org/services/owl-s/1.0/

[10] 阿多信吾、松永寿恵、岡育生、藤原値賀人、
属性およびレーティングに基づくWebサービスの自動選択手法
情報処理学会研究報告（DSM）2004年5月

[11] TravelXML 仕様公開ページ
http://www.xmlconsortium.org/wg/TravelXML/TravelXML_index.html

研究業績

◎Web サービス自動選択のためのサービスブローカの導入とその一実装
張 晨, 山根 一樹, 深澤 良彰
情報処理学会　第 67 回全国大会（発表予定）