社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＤＳＭ－33　(10)
2004／5／13

属性およびレーティングにもとづくWebサービスの自動選択手法

阿多　信吾†　　松永　寿恵†　　岡　　育生†　　藤原値賀人††

† 大阪市立大学 大学院工学研究科　〒558-8585 大阪市住吉区杉本 3-3-138
†† 大阪成蹊大学 現代経営情報学部　〒533-0007 大阪市東淀川区相川 3-10-62
E-mail: †{ata,matsunaga,oka}@n.info.eng.osaka-cu.ac.jp, ††fujiwara-c@osaka-seikei.ac.jp

あらまし　Webサービスは、既存のwwwによる通信技術をもとに、分散アプリケーション間の情報交換を行うために提唱された共通のフレームワークである。Webサービスを用いることで、離れた端末で動作する異なるアプリケーション間で、情報の共有や交換が容易に行えるという利点がある。しかし、同一の機能を提供する複数のサービスが存在する場合、それらの中から適切なサービスを提供するためのメカニズムは、現在のWebサービスアーキテクチャには存在しない。したがって、このような場合サービスの選択はユーザ自身が行わなければならないという問題がある。そこで、本研究ではサービスを自動的に選択するための新たなメカニズムを提案する。具体的には、サービスごとに評価基準となる属性（アトリビュート）を与え、ユーザ側でこれらの属性を収集する。そして、サービスごとに収集した属性により評価値（レーティング）を計算し、最も高い評価値を持つサービスを選択して実行する。本研究では、以上の手法を実現するために、属性データベース（アトリビュートレポジトリ）を導入し、属性の登録および検索に必要となる通信プロトコルを新たに定義する。また、実験環境において提案したメカニズムを実装し、属性値に応じたサービスの自動選択が実現可能であることを示す。

キーワード　Webサービス, 自動選択, 属性, レーティング

A Rating-based Automatic Web Service Selection by Using Attributes

Shingo ATA†, Hisae MATSUNAGA†, Ikuo OKA†, and Chikato FUJIWARA††

† Graduate School of Engineering, Osaka City University
3-3-138, Sugimoto, Sumiyoshi-ku, Osaka 558-8585, Japan
†† Faculty of Modern Management Information, Osaka Seikei University
3-10-62, Aikawa, Higahiyodogawa-ku, Osaka 533-0007, Japan
E-mail: †{ata,matsunaga,oka}@n.info.eng.osaka-cu.ac.jp, ††fujiwara-c@osaka-seikei.ac.jp

Abstract　Web service is one of useful common frameworks in order to exchange information between different applications. In the current specification on the Web service there is no mechanism to select an appropriate service from candidate Web services when there are multiple candidates to provide a service which is requested by the end user. In this paper, we propose a new architecture to automate the selection of multiple Web services. To realize an automatic selection, we introduce an *attribute* which represents a capability of web services. We then propose the service selection method based on the value of *rating*, which is calculated from the values of *attributes*. To share the kinds of *atttributes* between end users and service providers, we also introduce an attribute repository. We have implemented our proposed architecture, and verify that the service is automatically and adaptively based on the condition (i.e., attribute) of web services.

Key words　Web Service, Dynamic Selection, Rating, Attribute

1. はじめに

近年、インターネットの普及に伴い、ビジネスツールとしてインターネットを利用したシステムの開発が進められている。企業内のシステムにおいては、例えば、受発注システムのような業務アプリケーションの分散化が進み、業務アプリケーションの統一化が行なわれつつある。さらに、インターネットを通じて、同じ企業の支店や関連企業なども含めた業務アプリケー

ションの連携が模索されている。

分散化された異なるシステム間の連携を考える場合、通信プロトコルの共通化および標準化が一つの重要な課題となる。このため、W3C、OASIS などが主体となり、分散オブジェクト間通信の標準として Web サービスが提唱されている。Web サービスは、通信プロトコルとして HTTP (Hyper Text Transport Protocol) を利用した RPC (リモートプロシージャコール) の一種である。HTTP を利用することで、今までに http で培われた技術等が容易に適用でき、SSL などでセキュリティも確保できる。また、広く浸透している技術であるため、Web サービスを利用することで新たな設定を必要としないといった利点が挙げられる。また、通信内容を XML (eXtensible Markup Language) [1] により記述することで、プラットフォームの壁を越えたデータ交換が容易になる。さらに、XML はテキスト形式であるため HTTP と親和性が高い。また、XML は自由に名前を決めることができるタグを用いているため柔軟性が高いという利点も挙げられる。

Web サービスの利点は、一連のサービスをシステムが透過的に一括処理できるところにある。旅行の予約を例にすると、従来は、ユーザが Web サイトでホテルの予約、飛行機の予約などを、別々に手動で行なっていた。これに対して Web サービスを用いると、システムにおいてホテルの予約、飛行機の予約などが一括で処理できるようになる。

さらに、実際に Web サービスで動作している基盤的技術は、XML や HTTP など、インターネットで広く使用されている標準の技術である。そのため、Web サービスを利用するユーザは、既存のシステムを大きく変更する必要がなく接続したいサービスと通信することができ従来の資源を有効活用することができる。また、Web サービスは、Java などのオープンソースを用いて実装されている。システムの開発者は、その作成されたクラスを用いることで、容易にシステムを実装することができるという利点がある。

通常、Web サービスはそれぞれ固有のプロトコルあるいはインターフェース (API, WSDL) によって提供されているが、複数のサービスが同じインターフェースを提供することも可能である。複数の Web サービスが同じインターフェースを提供する目的として、以下の 2 つが考えられる。

（1） **複数のサービスプロバイダの設置による負荷分散:** 特に人気のある Web サービスでは、多数のクライアントからのアクセスによる負荷の集中が問題となるが、これらは複数の同一機能を提供する Web サービスプロバイダの設置によって軽減することができる。この場合、効率よく負荷分散を行うためには、各プロバイダの負荷状況や接続されているネットワーク性能値（遅延、利用可能帯域など）をもとに適切なプロバイダを選択することが望ましい。また、クライアント側からはこれらのプロバイダの中からもっとも近いプロバイダに接続されるのがよいと考えられる。

（2） **同種サービスを提供する異なる企業間での API の統一によるユーザ利便性の向上:** 同種のサービスの場合、従来は異なるプロバイダは異なるインターフェースを用いて提供していたが、これらを統一化することによって、ユーザは同じクライアントを用いるだけで、自分の好みに応じたプロバイダ選択を行なうことが可能となる。これによって、ユーザに対する利便性が向上する。

同一インターフェースを持つ Web サービスが複数存在する場合、ユーザはこれらの候補の中から自分の目的に応じたサービスを選択し、実行する必要がある。特に、Web サービスはシステムの自動化を目標としているので、これらのサービス選択についても自動化されることが望ましい。しかしながら、複数の Web サービスを目的に応じて自動的に選択するメカニズムは現在のところ標準化されていない。

自動的な Web サービスの選択を行うメカニズムとしては、ユーザが実行する複数の Web サービスの流れを自動化するために、あらかじめシナリオテンプレートを作成し、それにもとづく Web サービスの検索、および実行を行う手法が提案されている。しかしながらこれらの方法は、複数の Web サービスの連携に重点が置かれており、本稿が対象とする複数 Web サービスの自動選択を目標としたものではない。このため現在では、サービスのミラーリングなどの場合、DNS など別の負荷分散メカニズムを利用してサービスプロバイダの負荷を軽減している。これらの方法でも、ある程度はサービスプロバイダの負荷を分散させることは可能である。しかし、その分散方法は自動選択の際に用いられる負荷分散メカニズムのポリシーに強く依存したものであり、必ずしも Web サービスの自動選択にとって最適な手法とは限らない。また、異なるプロバイダのインターフェース統一による Web サービスについては、それらを自動的に選択する手法は存在しない。このような場合、サービスの実行は自動で行われるものの、サービス選択自体は自動化されていないため、最終的なサービスの選択は依然としてユーザ自身が行わなければならない。すなわち、目的の機能を提供する Web サービスが複数存在する場合、ユーザは従来の検索エンジンと同様、得られた結果を基に手動で目的のサービスを決定しなければならない。さらに、それら複数のサービスを順序づけする手段も提供されていないため、選択基準はすべてユーザに委ねられることになる。

このような背景から、Web サービスにおいてユーザが判断することなくサービスの自動選択を行うための研究がこれまでにもなされている [2], [3]。

文献 [2] では、第三者機関がサービスの評価を与えることにより、Web サービスの順序づけを行う手法が提案されている。これにより、複数の該当する Web サービスがあった場合に、ユーザはもっとも評価の高い Web サービスを自動的に受けることができる。しかしながら、このモデルではサービス選択が第三者機関の評価基準にのみ依存しており、その評価基準がユーザの要求に適合したものであるかはユーザ自身が判断しなければならない。また、このモデルが適切に動作するためには、評価を行う第三者機関の信頼性、適切な評価基準が必要であるが、これらを整備するためには業界全体の取り組みが必要不可欠であるため、実現には多くの時間を要する。また、文献 [3] では、独自のプロトコルの上で自動選択が行なわれるため、ユー

ザ自身もそのプロトコルに対応しなければならない。また、従来の Web サービスの形である、まずユーザが、UDDI に問い合わせ、サービスに関する情報を得る、という順序をとっておらず、現在の Web サービスに対する仕様を大きく変更しなくてはならない。

そこで本稿では、Web サービスをより幅広く運用するために、複数の Web サービスの候補が存在する場合にサービスを自動的に選択するための新たなメカニズムを提案する。具体的には、サービスごとに評価基準となる属性（アトリビュート）を与え、ユーザ側でこれらの属性を収集する。そして、サービスごとに収集した属性により評価値（レーティング）を計算し、最も高い評価値を持つサービスを選択して実行する。本稿では、以上の手法を実現するために、属性データベース（アトリビュートレポジトリ）を導入し、属性の登録および検索に必要となる通信プロトコルを新たに定義する。さらに、アトリビュートレポジトリを用いた自動選択メカニズムへの移行シナリオについても説明する。また、実験環境において提案したメカニズムを実装し、属性値に応じたサービスの自動選択が実現可能であることを示す。

以下、2. で本稿で提案するサービスの属性について説明し、3. で属性を用いたレーティング計算法および提案するサーバ選択手法を説明する。次に 4. でアトリビュートレポジトリの機能と目的、およびその利用手順について示す。さらに 5. で自動選択メカニズムの運用シナリオについて、その一例を示す。そして提案方式の実装および実験環境における動作検証について 6. で説明する。最後に 7. でまとめと今後の課題について述べる。

2. Web サービスの属性

本章では、提案するサービス自動選択機構を実現するために必要となるサービスの属性（アトリビュート）について定義し、その目的と役割、および利用法について説明する。

2.1 属性の定義

本稿で用いる属性とは、Web サービスに付加的につけられた情報であり、属性の名称を表す「属性キー」とその属性に対応する値「属性値」のペアにより構成される。属性キーはたとえばそのサービスの価格、所要時間、利用可能クライアント数など、サービスごとに異なる値を持つサービスの性質を示す要素である。属性値はレーティング計算に用いるために数値で表現する。各プロバイダはあらかじめ自身の Web サービスが提供できる属性キーを決定し、問い合わせに応じてその属性キーに対応する属性値を返す処理を実装する。1 つの Web サービスに対して、複数の属性を保持することが可能である。属性キーにはそれぞれ serviceKey の uuid を割り当てる。

2.2 属性の問い合わせ

図 1 に、各サービスに対する属性の問い合わせ手順を示す。ユーザは Web サービスの自動選択をする際に、各プロバイダに対して Web サービスの属性を問い合わせる。問い合わせは、各サービスに対してユーザが問い合わせたい属性キーの serviceKey を引数として queryAttribute を実行する。queryAttribute をコールされた Web サービスはそこに指定された

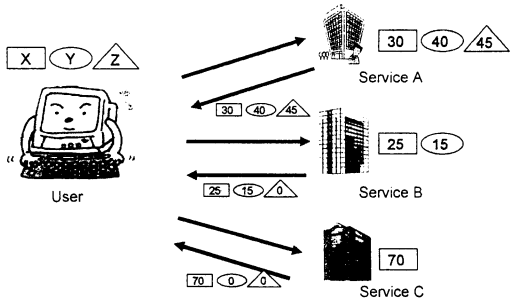

図 1 属性の問い合わせ

serviceKey が、自分が提供できる属性キーに含まれているかどうかをチェックし、もし属性値を返すことが可能であればその属性値を返す。そうでなければ属性値として null 値を返す。ただし、レーティングは数値計算により行われることから、null 値はゼロで代用できる。すなわち、queryAttribute をコールされた Web サービスにおいて指定された属性キーに対する属性値を提供できない場合は、属性値として 0 を返すこととする。

queryAttribute では、複数の属性キーを指定することが可能である。この場合、Web サービスを提供するプロバイダは指定された複数の属性キーに対する属性値のセットを返さなければならない。ただし、先ほどと同様に自身が提供できない属性キーに対する属性値は 0 として返す。また、queryAttribute 自体が実装されていないサービスに対する属性問い合わせの結果はすべて 0 が返されたものとして扱う。

2.3 属性回答の委譲

原則的にはユーザが各サービスに対して直接属性の問い合わせを行うが、属性キーの種類によってはサービスプロバイダが回答することが適さない場合もある。たとえば、サーバの負荷やネットワーク利用帯域などの定量的な客観的指標であればサービス自身が属性値を返すことが適当であるが、そのサービスの人気度や外部評価結果など、本来第 3 者が回答すべき属性キーも存在しうる。そのような場合は、プロバイダはコールされた queryAttribute に対して、属性値を null としたうえで delegatePoint 情報を付加してユーザに回答する。delegatePoint とは、そのプロバイダに代わって問い合わせたサービスに対する属性値を回答できる委譲機関のアクセスポイントを示している。ユーザは queryAttribute に対する回答に delegatePoint が含まれていた場合、そこに記述されたアクセスポイントに対してそのサービスの serviceKey と属性キーのセットを引数として queryAttribute を再度コールする。属性値の回答を委譲された機関は、queryAttribute で指定された Web サービスに対する属性値を、そのサービスプロバイダに代わって回答する。もし、委譲機関においても指定された属性キーに対する属性値が回答できない場合は属性値として null を返すものとし、再度 delegatePoint を指定することはできないものとする。

3. レーティングの計算とサービスの自動選択

ユーザは、Web サービスプロバイダそれぞれに対して、queryAttribute による属性値の問い合わせを行った後、サービスの選択基準となるレーティング (rating) の計算を行う。レーティングは、属性値をもとに計算される単一の数値指標であり、値が高いほどサービス選択において有利となる。したがって、サービスのレーティング値が高いほどそのユーザにとって適切なサービスであることを意味する。

問い合わせた属性値が単一の場合、それらの属性値をそのままサービスに対するレーティングとすることが可能である。しかしながら、収集した属性値がそれぞれ複数の場合、レーティング計算式にもとづき、単一指標への変換を行う必要がある。以下、属性値からのレーティング導出方法について説明する。

3.1 レーティングの計算

各サービス i に対するレーティング R_i は、以下の式により求められる。

$$R_i = \sum_{a \in \boldsymbol{A}} w_a \times N_a(V_i(a)) \tag{1}$$

ここで、$\boldsymbol{A}$ は queryAttribute で問い合わせた属性キーの集合、$V_i(a)$ は、サービス i に対して属性キー a による問い合わせの結果回答された属性値を表す。

複数の属性値から単一レーティングを計算する場合、それぞれの属性値が公平に扱われなければならない。したがって、レーティングの計算時にこれらの値を正規化する必要がある。式 (1) 中の $N_a(\cdot)$ がサービス a に関する正規化関数を示す。本稿では簡単化のため、以下の式を用いて正規化することとする。

$$N_a(v) = \frac{v}{\max v_i} \times 100 \tag{2}$$

ここで、$\max v_i$ は各サービスから得られた属性キー a に対する属性値のうち、最大である属性値を表す。すなわち上式は、もっとも属性値が大きいサービスを 100 とした場合の相対的な属性値を表している。

また、式 (1) において w_a は属性キー a に対する重み (weight) を表している。通常、各属性値は同じ重みによりレーティングが計算されるが、ユーザが特定の属性値に重みを置いた評価を行いたい場合、属性値ごとに異なる重み付けを行うこともできる。ただし、$\sum w_a = 1$ である。特にユーザが重みを明示しない場合、以下の式により w_a が計算される。

$$w_a = \frac{1}{|\boldsymbol{P}|} \tag{3}$$

ここで、集合 $\boldsymbol{P}$ はすべてのプロバイダが回答できた属性キーの集合、すなわち各プロバイダにおいて属性値が 0 でない属性キーのセットに対して積集合 (AND) をとったものになる。したがって、

$$\boldsymbol{P} = \cup_i P_i, \qquad P_i = \{a | a \in \boldsymbol{A}, V_i(a) > 0\} \tag{4}$$

である。

ただし、各プロバイダが提供できる属性キーに共通部分がない場合、すなわち $\boldsymbol{P} = \phi$ である場合には式 (3) を求めることができない。この場合は、レーティングの計算を行わないものし、すべての w_a を 0 として扱う。

3.2 レーティングを用いたサービスの選択

各サービスに対してレーティングが決定されると、その値にももとづきサービスの自動選択が行われる。すでに述べたとおり、レーティングの値が大きいほどユーザに適していると考えることができるため、ユーザはもっともレーティングの高いサービスを選択するだけで、ユーザに適したサービスを実行することができる。ただしレーティングが同じサービスが複数存在する場合は、その中からランダムで選択する。

4. アトリビュートレポジトリ

4.1 目　　的

提案手法をもちいたサービスの自動選択が効果的に行われるためには、サービスごとにレーティングが異なることが必要となる。そのためには、P が空集合とならないようプロバイダが保持する属性キーを可能な限り共通化する必要がある。属性キーを決定するにあたって、ユーザおよびプロバイダ側において以下の問題がある。

- サービスプロバイダからみた場合、より高い頻度でサービスが選択されるためには、どのような属性を提供すればいいのかがわからない。いくら多くの属性を提供できたとしても、それが属性問い合わせの際に指定されなければ、サービスが選択されることはない。したがって、提供する属性を決定する場合には、他のサービスがどのような属性を提供しているかを考慮することが必要となる。また、ユーザがどのような属性を好んで用いるかを調べることで、より多くのユーザから選択されることも考えられる。
- エンドユーザからみた場合、効果的なサービスの自動選択を実現するためには、プロバイダがどのような属性を提供しているのかを把握する必要がある。ユーザが指定した属性キーに対する値を書くプロバイダが提供できない場合は、返されるレーティングはすべて 0 となるため、サービスの自動選択がなされない（サービスがランダムに選択されるのみである）。したがってユーザはプロバイダが提供している属性キーをあらかじめ把握した上で、レーティングによる差別化が行われるよう適切な属性キーを指定して属性値を問い合わせる必要がある。

このように、効果的なサービス選択を実現するためには、ユーザ側およびサービスプロバイダ側の両方が属性キーに関する情報を共有することが必要不可欠であると考えられる。これらの背景から、本稿では属性キーの管理を行うデータベース（アトリビュートレポジトリ）を新たに導入する。アトリビュートレポジトリでは属性キーに関する情報を収集、管理する。したがって、各プロバイダが提供する属性値は保持しない。

4.2 アトリビュートレポジトリの役割

アトリビュートレポジトリは以下の機能を提供する。

- サービスの設計者に対して、サービスがどのような属性を提供すべきかの決定を支援するために、既存のサービスが実装している属性キーのリストを提供する。また、利用頻度にも

とづいた属性キーの順位を提供することで、サービス設計者に対して属性提供の優先度に関する情報を与える。

- ユーザに対してサービスプロバイダが提供している属性キーのリストを提供する。これにより、どの属性キーを使えばサービス選択が容易に行えるかを判断することができる。
- ユーザが問い合わせに用いた属性キーの情報を管理する。すなわち、どの属性キーがどのような頻度で問い合わせに利用されたかの統計情報を保持し、これらをユーザならびにサービスプロバイダに提供する。これにより、ユーザはどのような属性キーが好んで利用されるかを知ることが可能となり、たとえば問い合わせ頻度の高い属性キーを用いてサービス選択を行えば、属性キーに関して予備知識を持たないユーザでもサービスの自動選択によるメリットを受けることができる。また、サービスプロバイダにとっては問い合わせ頻度の高い属性キーを積極的に提供することで、より多くのユーザからサービスが選択されることになる。

4.3 アトリビュートレポジトリを利用したサービス選択の手順

サービスプロバイダがサービスを設計し、登録を行う際の手順を以下に示す。

（1） アトリビュートレポジトリに対して listAttributeKeys をコールし、他のプロバイダが提供している属性キーのリストを取得する。listAttributeKeys はオプションとして、サービスの種類を指定することが可能であり、属性キーのリストを関連サービスに限定して取得することも可能である。一覧は、属性キーと提供サービス数のリストで与えられる。

（2） アトリビュートレポジトリに対し、listAccessAttributes をコールし、ユーザが利用した属性キーを問い合わせ回数とともに一覧で取得する。listAttributeKeys と同様にオプションとしてサービスの種類を指定でき、関連サービス間における属性キーの問い合わせ回数のリストを取得できる。

（3） サービスプロバイダは、1. および 2. で取得したリストをもとに、サービスに対してどのような属性を提供するかを決定し、それらを実装する。

（4） サービスを公開する前に、アトリビュートレポジトリに対して registAttributeKeys をコールし、サービスが提供する属性キーをアトリビュートレポジトリに登録する。

また、ユーザがサービスの検索をする際の動作は以下の手順である。

（1） あらかじめアトリビュートレポジトリに対して listAttributeKeys をコールし、プロバイダが提供している属性キーのリストを取得する。このリストをもとにユーザは自分が優先度を上げたい属性キーに対して重みを指定する。また、特にユーザが重みを指定しない場合は、たとえば提供プロバイダ数による順序づけで上位１０種類の属性キーを用いて問い合わせることも考えられる。

（2） サービスを実行する場合、まずユーザは目的のサービスを UDDI により検索する。

（3） 検索の結果、複数のサービスが存在する場合は、それぞれのサービスに対して 1. で指定した属性キーを用いて queryAttribute をコールするし、属性値の収集を行う。

（4） 得られた属性値と 1. で指定した重みをもとに、それぞれのサービスに対するレーティングを計算する。

（5） レーティング計算の結果、もっともレーティングが高いサービスを実行する。

5. 段階的な移行シナリオ

本提案方式によるサービスの自動選択が効果的になされるためには、多くのプロバイダが属性を提供するメカニズムを実装し、さらにユーザがそれらの属性を積極的に利用できることが必要不可欠である。本章では、提案方式を段階的に普及させるための導入シナリオについて説明する。

すでに述べた通り、提案手法では一つのサービスに対して複数の属性を指定できる。また、問い合わせた属性をプロバイダが提供できない場合、その属性値は 0 となるため、サービスの選択に不利に働くことになる。したがって、プロバイダにとってはより多くのユーザからサービスを選択してもらうためには、より多くの属性キーの問い合わせに対して属性値を提供できることが必要となる。さらに、提案手法を実装していないサービスは、提案方式を実装したクライアントから選択される可能性が非常に低くなる。したがって、クライアントにおいて提案方式が十分に普及した状態になれば、おのずとサービスプロバイダは自身のサービスが優位に選択されるよう属性値の提供を行うことになる。その結果、より多くのプロバイダが提案方式による属性提供をサポートすることとなり、ユーザにとっての利便性が向上する。このように、クライアントあるいはサーバそれぞれにおいて提案方式がある程度普及することになれば、ユーザとプロバイダがお互いに影響を及ぼしあうことによって普及の速度が加速するということが期待される。これを移行スパイラルと呼ぶ。

したがって、本提案方式が広く利用されるようになるためには、ある程度のユーザとクライアントにおいてサービス自動選択が実現することが必要となる。その後は移行スパイラルが有効に働くことで広い普及が期待できる。

6. 実装および実験評価

本章では、提案方式を実装し実験環境において評価した結果について説明する。

6.1 実装環境

UUDDI にはオープンソースのプログラム juddi [4] および uddi4j [5] を用いる。アトリビュートレポジトリには、オープンソースのナチュラル XML データベース xindice [6] を用いて構築した。また、ユーザ、サービスプロバイダ、およびアトリビュートレポジトリ間の通信はすべて SOAP を用いることとし、それらの実装は Java を用いた。

6.2 実験結果

実装した提案方式を図 2 に示した環境で実験を行った。そのときの動作を以下に示す。

（1） ユーザが travel というキーワードでユーザプロキシに検索を依頼し、ユーザプロキシは travel のキーワードで UDDI

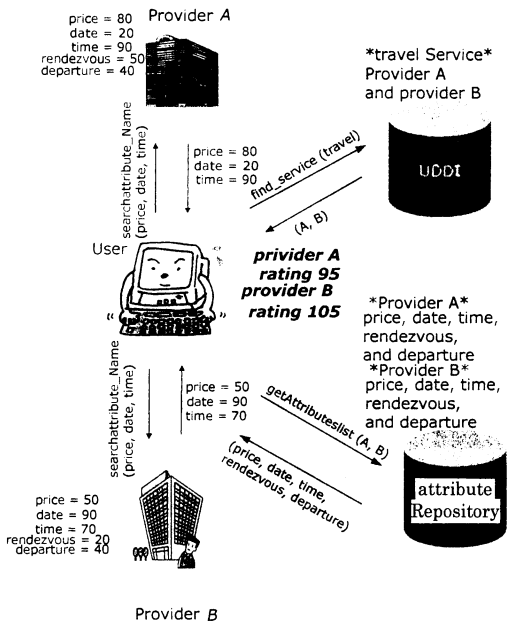

図 2 サービスの自動選択の実行結果

を検索する

（2） UDDI からプロバイダ A とプロバイダ B が発見され、ユーザプロキシがアトリビュートレポジトリに プロバイダ A とプロバイダ B についてのアトリビュートリストを問い合わせる

（3） プロバイダ A とプロバイダ B が提供しているアトリビュート price, date, time, rendezvous, departure と記述されたリストが得られ、その中から price, date, time を選択しそれらについてのアトリビュートの値をプロバイダ A とプロバイダ B に問い合わせる

（4） プロバイダから返されたアトリビュートの値とアトリビュートに付けた重みを用いてそれぞれ プロバイダ A とプロバイダ B についてレーティングを行なう

（5） レーティングの値が高いプロバイダ B に対してサービスを実行する

以上の実装内容から、実際にレーティングを行ないサービスを自動的に選択しサービスを提供しているプロバイダと通信することを確認した。

7. まとめと今後の課題

本論文では、Web サービスにおいてサービスの自動選択をする手法を提案した。サービスの自動選択を実現するために、各サービスに付加する情報として属性を定義し、属性の問い合わせ手法を示した。さらに、それらの属性値をもとにレーティングを計算し、その結果を用いたサービスの自動選択手法を提案した。また、提案方式の導入および普及に必要となる、アトリビュートレポジトリを提案し、提案方式への移行シナリオについてその一例を示した。

今後の課題としては、さらなる動作検証、サービス選択に適した属性キーの定義などが挙げられる。

文 献

[1] “XML.” `http://www.w3.org/XML/`.
[2] E. M. Maximilien and M. P. Singh, “Conceptual model of web service reputation,” *SIGMOD Rec.*, vol. 31, no. 4, pp. 36–41, 2002.
[3] M. Keidl, S. Seltzsam, and A. Kemper, “Flexible and reliable web service execution,” in *Technische Universitat Darmstadt, Germany*, pp. 17–30, Entwicklung von Anwendungen auf der Basis der XML Web-Service Technologie, July 2002.
[4] “juddi.” `http://ws.apache.org/juddi/`.
[5] “uddi4j.” `http://www.uddi4j.org/`.
[6] “xindice.” `http://xml.apache.org/xindice/index.html`.