社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
NS2003-315, IN2003-270(2004-03)

異なる連携方式を用いたWebサービスアプリケーションの開発および評価

石井 健一　串戸 洋平　山内 寛己　井垣 宏　玉田 春昭　中村 匡秀　松本 健一

奈良先端科学技術大学院大学 情報科学研究科 〒630-0192 奈良県生駒市高山町 8916-5

E-mail: {keni-i, youhei-k, hiroki-y, hiro-iga, harua-t, masa-n, matumoto}@is.aist-nara.ac.jp

あらまし　Webサービスは，インターネット上に分散するサービスを統合し，付加価値を高めたサービス提供を実現するインフラとして注目されている．本稿では，複数のWebサービスが連携して新たなサービスを実現する際の連携方式に着目し，連携方式の違いが，Webサービスアプリケーションの性能や開発しやすさに如何に影響するかを評価する．具体的には，同一の仕様に基づくWebサービスアプリケーションを異なる連携方式（リダイレクト型，プロキシ型，スタンドアロン型）で実装し，実行時間，コード行数から評価を行う．その結果，プロキシ型連携方式が実行時間およびコード行数の両側面において，リダイレクト型連携方式より優れていることがわかった．また，スタンドアロン型と他の2方式の比較からWebサービスの数が性能面および保守性に大きな影響を与えることを示した．

キーワード　Webサービス，サービス連携，連携方式，Webサービスアプリケーション

Development and Evaluation of Web service applications with Different Integration Schemes

Ken-ichi ISHII　Youhei KUSHIDO　Hiroki YAMAUCHI　Hiroshi IGAKI　Haruaki TAMADA
Masahide NAKAMURA　and　Ken-ichi MATSUMOTO

Graduate School of Information Science, Nara Institute of Science and Technology

8916-5 Takayama, Ikoma, Nara, 630-0192 Japan

E-mail: {keni-i, youhei-k, hiroki-y, hiro-iga, harua-t, masa-n, matumoto}@is.aist-nara.ac.jp

Abstract　Web services are a powerful infrastructure to provide value-added services integrating distributed services over the Internet. This paper focuses the integration schemes to achieve the new service from multiple services. We evaluate how the integration schemes give impact on performance and development efforts of Web service applications. Specifically, based on the same specification, we developed three versions of the applications with different integration schemes (redirection type, proxy type, stand-alone type). Then, we evaluated the execution time, the lines of code of the Web applications. From the result of the experiment, the proxy type scheme is superior to the redirection type scheme in both performance and lines of codes. Also, the comparison with the stand-alone type and other two types showed that the number of Web services gave significant influence on performance and maintainability.

Keyword　Web Services, Service Integration, Integration Schemes, Web Service Applications

1. はじめに

近年，インターネット網の急速な整備と e-ビジネスの普及に伴い，Web の利用目的は，単なる情報流通基盤からサービス流通基盤へ進化をとげている．中でも，Web サービス（Web Service）は，次世代 Web の動的側面を支える分散計算環境の基盤として多方面で注目されている[5]．Web サービスは，インターネット上に分散するサービスを統合し，付加価値を高めたサービス提供を実現するインフラを提供する．また，Web サービスは，各サービスの独立性や再利用性を確保することで，変更や再構築の効率性を高めることが可能である [6]．

Web サービスは，サービス指向アーキテクチャ（Service Oriented Architecture）の考えに基づき，サービス提供者，サービス利用者，サービス仲介者の 3 者間の相互作用により構成され，標準化された規格（HTTP, XML, SOAP, UDDI）でサービスを実現するものである．Web サービスは，従来のオブジェクト指向設計のように，機能を一つのオブジェクトとして設計するのではなく，より粒度の大きいサービスを一つの部品（コンポーネント）として設計する．これにより，Web サービス内部の変更がシステム全体に波及することを防止でき，既存サービスの再利用性を向上できる．さらに，Web サービスでは，標準化された通信手段を用いるため，異なるシステム間の連携を効率よく実現できる．

現在，Web サービスを用いたシステムがいくつか開発され，公開されている[1] [3] [7] [8]．しかし，Web サービスを用いたシステムは，まだ，試験段階のものや実地運用が開始されてから日の浅いものが多く，体系だった開発方法論は未だに提案されていない．また，Web サービスアプリケーションの性能を評価する体系的な手段も存在しない．

本稿の目的は，複数の Web サービスが連携して新たなサービスを実現する場合の連携方式に着目し，連携方式の違いが，Web サービスアプリケーションの性能や開発しやすさに如何に影響するかを評価することである．

具体的には，クライアントアプリケーションが 2 つの Web サービス A,B を利用する場合を考え，以下の 2 種類の連携方式に着目する：クライアントアプリケーションが直接 A,B を逐次的に利用するリダイレクト型連携方式，クライアントアプリケーションが A を利用し，A がクライアントアプリケーションの代わりに B を利用するプロキシ型連携方式．

性能評価においては，バス時刻表検索サービスを 2 通りの連携方式を用いて開発した．実験においては，サービスの実行時間およびソースコード行数を計測した．その結果，プロキシ型連携方式が実行時間およびクライアントアプリケーションのコード行数の両側面において，リダイレクト型より優れていることがわかった．また，Web サービスの数が性能面に大きな影響を与えることを示した．

2. Web サービスと 2 つの異なる連携方式

本稿では，同じ目的を達成する Web サービスアプリケーションを 2 つの異なる連携方式（リダイレクト型とプロキシ型）を用いて設計・実装を行い，その評価を行う．本章では，Web サービスと 2 つの異なる連携方式について定義，解説する．

2.1. Web サービス

Web サービスは，ソフトウェアをサービスという単位でコンポーネント化し，Web サーバ上でその機能を提供するための標準的な枠組みである．クライアントアプリケーションは，サービスへのアクセスに Web ブラウザを必要とせず，直接データの授受が行える．サービスの利用は，クライアントアプリケーションからリモートプロシージャコール（RPC）で行われる．XML-RPC および SOAP といった Web サービスで標準化された手段を用いることで，アプリケーションの開発者は，送受信するメッセージの書式やプロトコルを意識することなく，通常のメソッド呼び出しとほぼ同じ方法で，Web サーバ上のサービスをアプリケーションに組み込むことができる．図 1 に一般的な Web サービスの構成を示す．

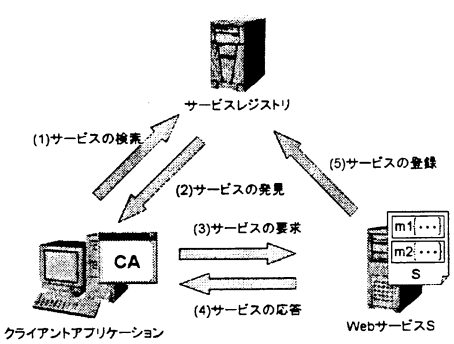

図 1　一般的な Web サービスの構成

本稿では，Web サービスを利用するアプリケーションをクライアントアプリケーション(以降 **CA** とする)，Web サービスが登録されているサーバをサービスレジストリと呼ぶ．図 1 において，CA が，Web サービス S を呼び出す手順は，以下の通りである．

1. CA は，サービスレジストリを利用して，Web サービス S の検索を行う（図 1(1)）
2. サービスレジストリは，Web サービス S が提供

されるサーバとWebサービスSが公開しているメソッドのインタフェース情報をCAへ返す（図1(2)）

3. CAは，メソッドを介してサービスをWebサービスSへ要求する（図1(3)）
4. WebサービスSは，CAに応答する（図1(4)）

ただし，CAがWebサービスSの存在を知っている場合，上記の1，2は省略される．また，WebサービスSは，存在を公開するために，サービスレジストリへサービスの登録を行う（図1(5)）．

WebサービスSは，CAに対して，メソッドmを通してのみサービスを提供する．CAは，WebサービスSの仕組みや内部構造を意識せずに，サービスを利用できる．

CAは，複数のWebサービスを利用することができる．また，WebサービスSは，他のWebサービスと連携し，複合的なサービスを提供することもできる．

Webサービスのこのような性質から，CAとWebサービスとの連携においては，複数の方式が考えられる．以降では，考えの基礎となる1つのCAと2つのWebサービスとの異なる連携方式について述べる．

2.2. リダイレクト型

1つのCAと2つのWebサービスが連携するとき，まず，CAが一方のサービスを要求し，その応答を一旦受け取ってから，CAが他方のサービスを要求する方式が考えられる．本稿では，このような方式をリダイレクト型と呼ぶ（図2）．

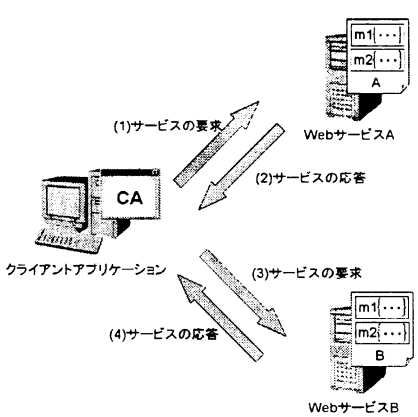

図2　リダイレクト型

図2において，CAが，2つのWebサービスA，Bを呼び出す手順は，以下の通りである．

1. CAは，WebサービスAへサービスを要求する（図2(1)）
2. WebサービスAは，CAへ応答する（図2(2)）
3. CAは，WebサービスBへサービスを要求する（図2(3)）
4. WebサービスBは，CAへ応答する（図2(4)）

2.3. プロキシ型

もう1つの連携方式として，CAが一方のサービスを要求すると，そのサービスがCAの代わりにもう一方のサービスを要求する方式が考えられる．本稿では，このような方式をプロキシ型と呼ぶ（図3）．プロキシ型において，CAは，目的とする処理を行うために，1つのWebサービスのみと連携する．

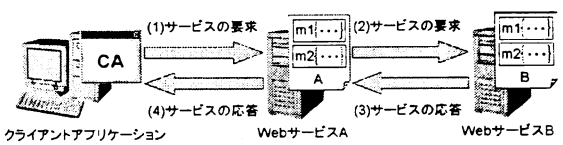

図3　プロキシ型

図3において，CAが，WebサービスAを呼び出す手順は，以下の通りである．

1. CAは，WebサービスAへサービスを要求する（図3(1)）
2. WebサービスAは，WebサービスBへサービスを要求する（図3(2)）
3. WebサービスBは，WebサービスAへ応答する（図3(3)）
4. WebサービスAは，CAへ応答する（図3(4)）

3. バス時刻表検索サービスの開発

本研究では，ケーススタディとして，Webサービスを用いたバス時刻表検索サービスの開発を行った．

3.1. シナリオ

本システムは，ユーザがCAを操作し，現在時刻以降に最も早く乗車可能なバスの時刻を知ることを目的とするバス時刻表検索サービスである．バスの時刻は，平日，土曜日，日曜・祝日により運行時間が異なる．従って，バスの時刻を検索するためには，曜日を調べなければならず，カレンダーサービスが必要となる．しかし，カレンダーサービスは，比較的汎用性が高いため，CA本体に組み込まず，我々が別目的に作成した既存のカレンダーWebサービスを再利用することにした．本システムの特徴は，以下の通りである．

1. 時刻などの入力を省略し，CAを1クリックするとバスの時刻がわかる
2. バスの時刻の検索対象を1つのバス停に限定

3.2. 設計

本サービスを開発するにあたっては，以下の3つの機能が必要となる．

F1　時刻表検索機能：与えられた曜日と現在時刻から最も早く乗車可能なバスの時刻を検索する機能
F2　曜日取得機能：与えられた年月日が平日，土曜日，日曜日，祝日かを検索する機能
F3　表示機能：ユーザに結果を表示する機能

F2に関しては，前述のカレンダーWebサービスを再利用するため，今回新しく開発を行う必要がない．残りのF1，F3の機能の実装法として，以下の2通りが考えられる．

I.　F1をWebサービス，F3をCAとして実装
II.　F1，F3共にCAとして実装

Iは，F3のみをCAとして実装し，サービスの核であるF1をWebサービス（バス時刻表 **Web** サービスと呼ぶ）化する方法である．この場合，CAは，バス時刻表Webサービスおよびカレンダー**Web**サービスの2つと連携が必要となる．従って，2章で述べたリダイレクト型とプロキシ型の2通りの実装が考えられる．

まず，リダイレクト型（図4）では，最初に，ユーザの操作によりCAがカレンダーWebサービスへ年月日を送信する（図4(1)）．次に，カレンダーWebサービスは，受信した年月日に基づいて曜日をCAへ返答する（図4(2)）．次に，CAは，バス時刻表Webサービスへ現在時刻と曜日を送信する（図4(3)）．そして，バス時刻表Webサービスは，受信した現在時刻と曜日に基づいて時刻を検索し，CAへ検索結果を返答する（図4(4)）．最後に，CAは，結果を表示する．

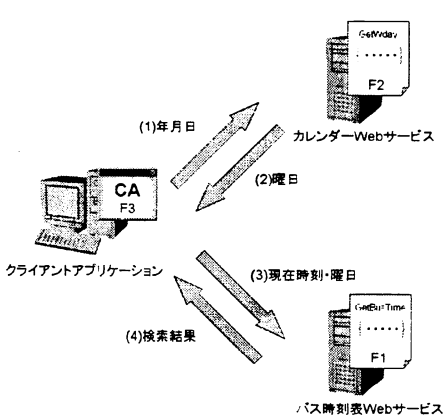

図4　リダイレクト型バス時刻表検索サービス

一方，プロキシ型（図5）では，まず，ユーザの操作によりCAがバス時刻表Webサービスへ現在時刻と年月日を送信する（図5(1)）．次に，バス時刻表Webサービスは，カレンダーWebサービスへ年月日を送信する（図5(2)）．次に，カレンダーWebサービスは，受信した年月日に基づきバス時刻表Webサービスへ曜日を返答する（図5(3)）．そして，バス時刻表Webサービスは，受信した現在時刻と曜日に基づいて時刻を検索し，検索結果をCAへ返答する（図5(4)）．最後に，CAは，結果を表示する．

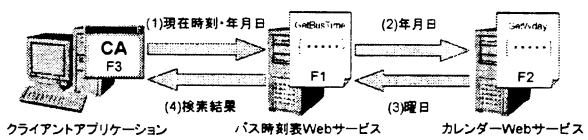

図5　プロキシ型バス時刻表検索サービス

また，上記IIのように，F1とF3を共にCAの中へ実装する方法が考えられる．ここでは，この実装方法をスタンドアロン型と呼ぶことにする（図6）．スタンドアロン型では，まず，ユーザの操作によりCAがカレンダーWebサービスへ年月日を送信する（図6(1)）．次に，カレンダーWebサービスは，受信した年月日に基づいて曜日をCAへ返答する（図6(2)）．そして，CAは，現在時刻と受信した曜日に基づいて時刻を検索し，検索結果を表示する．

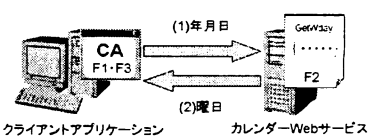

図6　スタンドアロン型バス時刻表検索サービス

3.3. 実装

3.2節での設計を基に，リダイレクト型バス時刻表検索サービス，プロキシ型バス時刻表検索サービス，スタンドアロン型バス時刻表検索サービスの3種類の実装を行った．実装環境は，以下の通りである．

言語：Microsoft Visual C# .NET
プラットフォーム：Microsoft .NET Framework 1.1
Webサーバ：IIS(Internet Information Services)5.1

バス時刻表Webサービスは，int型の現在時刻（時・分）とint型の曜日コード（平日：0，土曜日：1，日曜日：2，祝日：3）を入力すると内部のバス時刻表（XMLファイル）を検索し，int型で最も早く乗車可能なバスの時刻を出力する．CAは，ユーザが操作しやすいよ

うに，ボタンを1クリックするだけで最も早く乗車可能なバスの時刻が表示される（図7）．

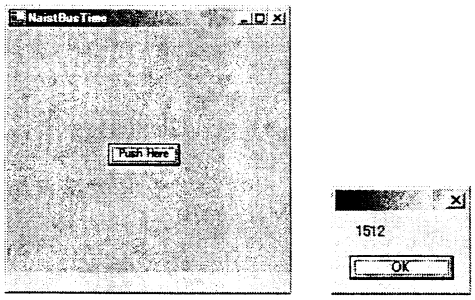

図7　CA実行画面

4. 評価実験

本章では，実装した3種類のバス時刻表サービスを用いて実験を行い，性能の比較評価を行う．実験には，1台のサーバを用いた．実験用サーバのスペックは，以下の通りである．

CPU：Pentium 4 2.4GHz
メモリ：512MB
OS：Windows XP Professional SP1
プラットフォーム：Microsoft .NET Framework 1.1
Webサーバ：IIS(Internet Information Services)5.1

今回の実験では，CAとWebサービス，Webサービス同士のネットワークにおける通信時間を無視するため，1台のサーバ内にCAおよびWebサービスを全て配置した．

4.1. 比較項目の検討

実験では，各連携方式におけるCAの実行時間を計測する．実行時間とは，ユーザがCAのボタンをクリックしてからバスの時刻が表示されるまでの時間である．実行時間の計測は，それぞれ各連携方式で10回ずつ行い，その平均を計算する．また，高負荷時での連携方式の違いを評価する目的で，Webサービスの呼び出し回数を100回，1000回と変化させた場合の実行時間も測定した．さらに，連携方式と保守性との関連を考察するために，各連携方式のCAおよびバス時刻表WebサービスのLOCを計測した．

4.2. 比較評価

実行時間の計測結果を表1に示す．

表1　実行時間の計測結果（秒）

ループ回数	プロキシ型	リダイレクト型	スタンドアロン型
1	3.80	4.06	2.48
100	5.11	5.83	3.39
1000	16.98	21.65	11.39

CAの実行時間に関しては，3つの連携方式のうち，スタンドアロン型が最も優れた結果となった．これは，スタンドアロン型でCAが利用するWebサービスの数が他の2方式と比べて少ないからであり，アプリケーションが利用するWebサービスの数が性能に大きな影響を与えることがわかる．

プロキシ型とリダイレクト型の比較では，プロキシ型の性能が優れていることがわかる．特に，ループ回数が多い高負荷時には，顕著な差が見られる．また，いずれの連携方式においても，CAの初回の実行には，時間がかかり，ループ回数の増加に応じて1回あたりの実行時間が減少していることがみとめられた．これは，Webサービスが実行されるサーバ上で，サービスのキャッシングが発生していると思われる．詳細な分析については，今後の課題とする．

次に，LOCの計測結果を表2に示す．

表2　LOCの計測結果（行）

	プロキシ型		リダイレクト型		スタンドアロン型	
LOC	CA	WS	CA	WS	CA	WS
	102(8)	145(35)	108(15)	140(30)	199(106)	NA

各連携方式において，CAの行数およびバス時刻表Webサービスのソースコード行数を計測した．3種類の実装は，異なる連携方式であるが，同一のシステム機能を有するため，実装間で共通のコードが存在する．従って，連携方式独自のコード行数を括弧中に示す．

まず，プロキシ型とリダイレクト型のLOCの比較を行う．CAの行数に関しては，プロキシ型の方が少ない行数で済む．独自のコード行数の比較では，プロキシ型のコード行数がリダイレクト型の約半分である．これは，プロキシ型では，CAが直接呼び出すWebサービスの数が1つ（バス時刻表Webサービスのみ）であるのに対して，リダイレクト型では，2つ（バス時刻表WebサービスおよびカレンダーWebサービス）になるからである．分離できる処理は，なるべくサービス側に閉じ込め，クライアント側の呼び出しインタフェースをなるべくシンプルにするという目的においては，プロキシ型がリダイレクト型より優れているといえる．

スタンドアロン型では，CAがバス時刻表Webサービス部分を全て含むため，他の2方式に比べてCAのコード行数が多くなっている．従って，CAの実装の容易性および信頼性の観点からは，他の2方式に劣るといえる（LOCが大きいほどバグが含まれやすい）．また，F1とF3が密に結合しているため，保守性に関する問題も生じる．もし，バスの時刻表が更新された場合，CAの全てのユーザは，CAを最新版にする必要

がある．一方，他の2方式では，バス時刻表Webサービスの時刻表のみを更新すれば良く，CAの再インストールは不要である．この利点は，Webサービスの理念である疎結合によるものである．

4.3. 考察

実験では，プロキシ型連携方式が，実行時間およびCAのコード行数の両側面において，リダイレクト型より優れていることがわかった．本実験で開発した小規模なWebサービスアプリケーションにおいて，この2方式の差は，それほど大きなものではなかった．しかし，2つのWebサービスの連携において，一方のWebサービスの出力を他方がそのまま入力としてとるような場合，性能面でプロキシ型の利点がより大きくなる．なぜならば，リダイレクト型では，2つのWebサービスのデータ授受をCAが中継する手間が入るからである．従って，このような場合，Webサービスの設計段階において，連携方式が選択可能ならば，プロキシ型を選択する方が望ましいと思われる．

今回の開発においては，新たにWebサービスを設計・開発したため，連携方式の選択が可能であった．しかしながら，独立した既存のWebサービスを組み合わせてCAから利用する場合には，リダイレクト型を取らざるを得ない．これは，公開されているWebサービスを外部から変更できないためである．この連携の汎用性という意味においては，リダイレクト型が優れているといえる．

スタンドアロン型との比較により，利用するWebサービスの数が性能面に大きな影響を与えることもわかった．しかし，これは，疎結合による保守性の向上とトレードオフの関係にあるため，目的に応じた連携方式の設計が望まれる．

5. おわりに

本稿では，同じ目的を達成するWebサービスアプリケーションを2つの異なる連携方式を用いて設計・実装を行い，その性能評価を行った．

今後の課題としては，Webサービスが実行されるサーバ上のサービスのキャッシングについて詳細な分析を行うことや各連携方式のサービスを実際の運用環境に近い形で性能評価することである．また，将来的には，Webサービスにおけるサービス競合問題の研究につなげていきたい．

文　献

[1] Amazon Web Services, http://www.amazon.com/gp/browse.html/104-0877510-2922306?node=3435361

[2] 青山幹雄，“Webサービス技術とWebサービスネットワーク”，信学技報，IN2002-163，pp.47-52，Jan.2003.

[3] Google Web APIs, http://www.google.com/apis/

[4] 本多泰理，矢田健，山田博司，“トラヒックフローを考慮したWebサービスのネットワーク構成法の一検討”，信学会総合大会講演論文集，2003-3，pp.354，Mar.2003.

[5] 和泉憲明，幸島明夫，車谷浩一，福田直樹，山口高平，“多粒度リポジトリに基づくWebサービスのビジネスモデル駆動連携”，信学技報，KBSE2002-32，pp.43-48，Jan.2003.

[6] 西村徹，堀川桂太郎，“Webサービスにおける境界条件の検査”，信学会総合大会講演論文集，2003-6，pp.234，Mar.2003.

[7] XML Webサービス対応Business Travel Systemパイロット版，http://net.est.co.jp/jtb/about/

[8] XML Webサービス対応三省堂デイリーコンサイス体験版，http://www.btonic.com/ws/