NAIST-IS-MT0351011

修士論文

WS-PROVE：Web サービスアプリケーションの プロトタイピングおよび性能評価のためのシステム

石井 健一

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 情報システム学専攻

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
修士(工学) 授与の要件として提出した修士論文である。

石井 健一

審査委員：

松本 健一 教授

小山 正樹 教授

飯田 元　 助教授

WS-PROVE：Webサービスアプリケーションの プロトタイピングおよび性能評価のためのシステム*

石井 健一

内容梗概

Webサービスの迅速なプロトタイピングを行うための汎用的なシステム WS-PROVE（Web Service PROtotyping Validation Environment）を提案する．WS-PROVEは，まだ実装が確定していない新規のWebサービスをダミーWebサービスとして抽象化し，別のダミーWebサービス，または，既存のWebサービスと任意のトポロジーで連携させることができる．また，応答時間やネットワーク遅延を設定すると，Webサービスを動的接続し，各サービスあるいは統合サービス全体の処理時間を計測できる．本論文では，Webサービスアプリケーションの性能プロトタイピングシステムの要件を整理し，WS-PROVEの設計，実装，評価を行う．また，WS-PROVEを用いて，いくつかのWebサービスアプリケーションのプロトタイピングおよび性能見積もりを行い，WS-PROVEの有効性を示す．

キーワード

Webサービス，Webサービスアプリケーション，プロトタイピング

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文，NAIST-ISMT0351011，2005 年2 月3 日．

WS-PROVE: A System for Rapid Prototyping and Performance Evaluation of Web Service Applications *

Ken-ichi Ishii

Abstract

This paper proposes a general framework, called WS-PROVE (Web Service PROtotyping Validation Environment), for rapid prototyping of Web service applications. In a requirement definition stage, WS-PROVE first abstracts a Web service under development as a dummy, and then integrate the dummy with other dummies or the existing ready-made Web services in arbitrary topologies. For a given configuration including the integration topology, processing time for each service and the network delay, WS-PROVE dynamically connects the Web services. Then, it measures the response time for each service and the whole integrated services. In this paper, we first clarify system requirements of the rapid prototyping system for Web service applications. Then, we present design and implementation of WS-PROVE. Additionally, we conduct a case study to prototype some Web service applications to demonstrate the effectiveness of WS-PROVE.

Keywords:

Web Services, Web Service Applications, Rapid Prototyping

.*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0351011, February 3, 2005.

目次

図目次

表目次

1. はじめに

Web サービスは，自己記述型の検索可能で汎用的なサービスを OS やアクセスデバイスの種類を問わずにネットワーク経由で接続・連携するための枠組みである[23]．異機種分散環境におけるソフトウェアシステムの各機能をサービスという単位で公開し，共通のインタフェースを用いて疎結合するサービス指向アーキテクチャ（SOA）の実現手段として注目を集めている．

近年，システムの大規模化とネットワークの発達に伴い，既存システムの再利用と分散処理が重要な課題となり，Web サービス技術が実際に使われ始めている．例えば，企業内の異機種の基幹システムを Web サービス化して連携・統合する[25]，レガシーな社内システムを付加価値サービスとして外部公開し企業間システムを実現する[5]，また，Web アプリケーションをサービス化して統合ポータルを構築する[26]といった事例がある．

しかし，Web サービスは比較的新しい技術であるため，Web サービスを用いたシステムの体系的な開発方法論やベストプラクティス，キラーアプリケーション等は，未だ十分明らかになっていない．また，従来のアプリケーションと異なり，Web サービスでは，サービス連携にかかるサーバのミドルウェア部（XML, WSDL, SOAP/HTTP の処理等）のオーバーヘッドとサーバ間のネットワークを陽に考慮しなければならない．従って，Web サービスアプリケーションでは，特に，**サービス連携部分の効率**や**信頼性**といった**非機能要求**が重要となる．しかし，現状，これらの非機能要求は，実装後のテストで初めて評価され，システムの要求定義段階では取得できない．結果として，開発の早期段階において，Web サービスを用いるべきかどうかの判断は，開発者の経験に基づいてアドホックに行わざるをえない．

この問題に対処するため，我々は，本論文において，Web サービスの迅速なプロトタイピング（Rapid Prototyping）[13]を行うための汎用的なシステム **WS-PROVE**（Web Service PROtotyping Validation Environment）を提案する．WS-PROVE の目的は，開発の要求段階において，Web サービス連携部分の非機能要求（現段階では主に性能）を評価可能にすることである ．

WS-PROVE は，まだ実装が確定していない新規の Web サービスを**ダミーWeb**

サービスとして抽象化し，別のダミーWeb サービス，または，既存の Web サービスと**任意のトポロジー**で連携させることができる. 全体のサービス連携トポロジーと各ダミーWeb サービスに対する応答時間や負荷を設定すると，WS-PROVE はダミー・既存 Web サービスを設定に従って動的接続し，各サービスあるいは統合サービス全体の応答時間を計測する．Web サービスの接続・連携は，実際のミドルウェアおよびネットワークを介して行われるため，シミュレーションによる計測見積もりより正確な性能評価が可能となる．本論文では，Web サービスアプリケーションのプロトタイピングシステムの要件を整理し，WS-PROVE の設計，実装，評価を行う．また，WS-PROVE を用いて，いくつかの Web サービスアプリケーションのプロトタイピングおよび性能見積もりを行い，WS-PROVE の有効性を示す.

2. 諸定義

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.その最大の特徴は，クライアント側からのアクセスに Web ブラウザを必要とせず，直接データを授受できる機構が提供されていることである．そして，サービスの利用は，クライアントアプリケーションから XML-RPC および SOAP といったリモートプロシージャコール（RPC）で行われる．このように，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.

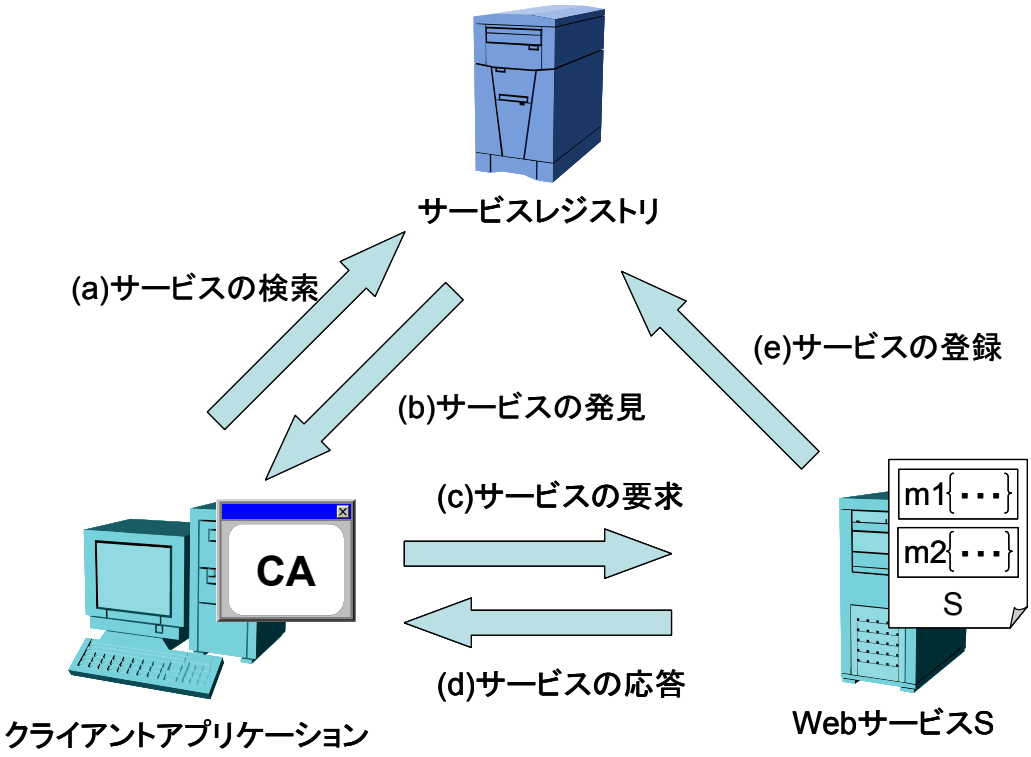


図 1　一般的な Web サービスの構成

本論文では，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 (a)）
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 (b)）
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(c)）
4. Web サービス S は，CA に応答する（図 1(d)）

ただし，CA が Web サービス S の存在を知っている場合，上記の手順 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(e)）．

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．さらに，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．Web サービスの連携については，第 3.2 節 Web サービス連携トポロジーで説明する．

2.2　ソフトウェアプロトタイピング

ソフトウェア工学では，ソフトウェアの開発や運用・保守段階がどのように進んでいくかモデル化したものを**ソフトウェアライフサイクルモデル**（software life cycle model）と呼ぶ[13]．伝統的なソフトウェアライフサイクルモデルには，**ウォータフォールモデル**（waterfall model）がある．ウォータフォールモデルは，ソフトウェア開発工程を要求，設計，テスト，導入，運用，保守といった諸段階に分け，重複や反復が無いように各工程を実行するモデルである（図 2）．

ウォータフォールモデルには，各工程を分業化できる利点がある．しかし，工程の後戻りは，一般的に莫大な工数がかかるため，各工程は，出来る限り完全に完了しておく必要がある．特に，ライフサイクルの一番目にあたる要求定義は，完全に行わないと後続の工程からの後戻りが頻繁に発生するため，慎重を期する必要がある．設計開始前に利用者の要求定義を完全に行わなければならない．これらの問題点を解決するため，新たなソフトウェアライフサイクルモデルがいくつか提案されている[20]．その 1 つに**ラピッドプロトタイピング**（rapid prototyping）がある．

ラピッドプロトタイピングは，利用者の要求を正確に定義することを目的として，開発の初期段階で**ソフトウェアプロトタイプ**（software prototype）を開発し，利用者に試用させることによって要求を正確に定義するモデルである（図 3）．

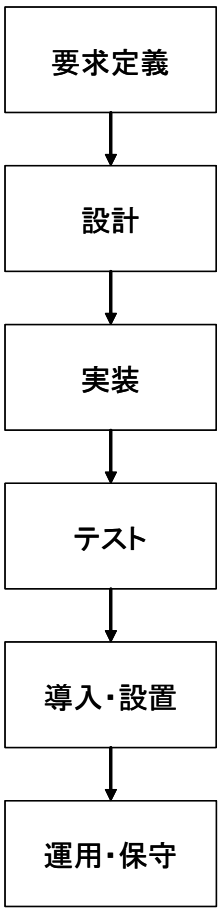


図 2　ウォータフォールモデル

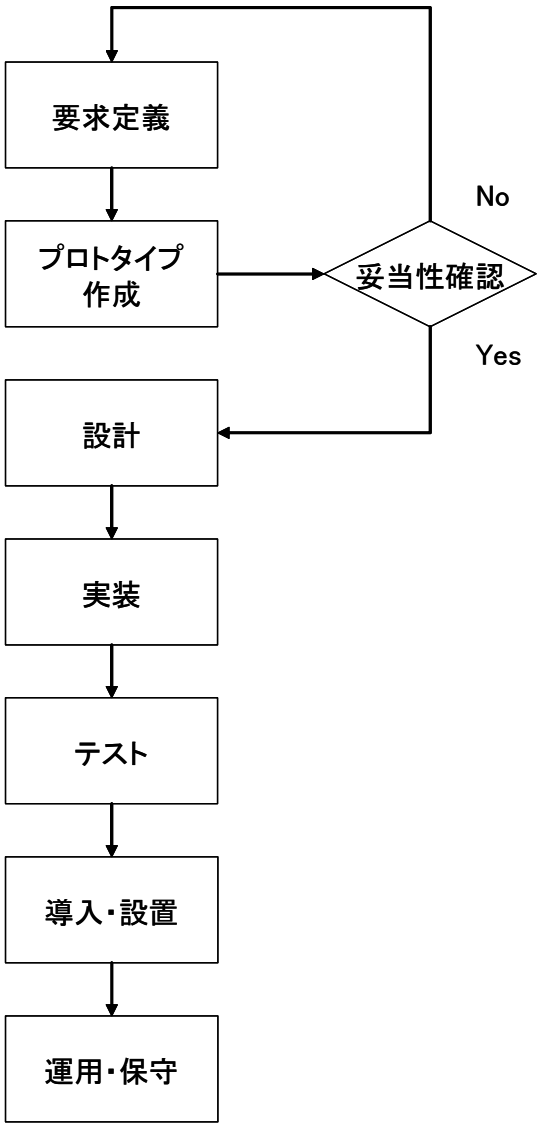


図 3　ラピッドプロトタイピング

ソフトウェアプロトタイプでは，最終製品が備えるべき機能の詳細まで実装せず，ユーザインタフェースや主な機能だけが実現される．また，ソフトウェアプロトタイプでは，開発の迅速さが要求されるため，一般に，非機能要求である安全性や信頼性，効率，性能などが無視される．本論文では，ソフトウェアプロトタイプを単にプロトタイプ，プロトタイプを開発することをプロトタイピングと呼ぶ．

プロトタイピングは，プロトタイプの対象，プロトタイピングのタイミング，プロトタイプと最終製品の関係によって分類される．以降では，それぞれの分類について詳細を解説する．

まず，プロトタイピングは，プロトタイプの対象によって，**ユーザインタフェースプロトタイピング**と**機能型プロトタイピング**（functional prototyping）に分類される．対象がユーザインタフェースである場合，ユーザインタフェースプロトタイピング，対象がユーザインタフェースと機能（計算機能・処理機能）である場合，機能型プロトタイピングに分類される．

次に，プロトタイピングのタイミングによって，**発見型プロトタイピング**（exploratory prototyping）と**経験型プロトタイピング**（experimental prototyping）に分類される．ユーザによる要求の妥当性確認を目標とし，要求定義段階で行われる場合，発見型プロトタイピングに分類される．設計段階で採用した技法によって，目標とする性能が達成できるか，ハードウェアは適切かどうか開発者が確認する段階で行われる場合，経験型プロトタイピングに分類される．また，経験型プロトタイピングは，性能を確認する**性能型プロトタイピング**（performance prototyping）とハードウェアを確認する**ハードウェアプロトタイピング**（hardware prototyping）に分類される．

最後に，ソフトウェアプロトタイプと最終製品の関係によって，**使い捨て型プロトタイピング**（throw-away prototyping）と**進化型（成長型）プロトタイピング**（evolutional prototyping）に分類される．この分類は，ユーザによる要求の妥当性確認後のプロトタイプがどのように使われるかによって行われる．要求の妥当性確認後のプロトタイプを破棄する場合，使い捨て型プロトタイピング，要求の妥当性確認後のプロトタイプを破棄せず，ベースとして開発を進める場合，進化型（成長型）プロトタイピングに分類される．プロトタイピングの分類を表 1 に示す．

表 1　プロトタイピングの分類

分類方法	分類		説明	
対象	ユーザインタフェース		ユーザインタフェースが対象	
	機能型		ユーザインタフェースと機能が対象	
タイミング	発見型		要求定義段階	
	経験型	性能型	実装段階	性能を確認
		ハードウェア		ハードウェアを確認
最終製品との関係	使い捨て型		プロトタイプを破棄する	
	進化型（成長型）		プロトタイプを破棄しない	

プロトタイピングの利点は，以下のような事柄が挙げられる．

- 開発者と利用者の誤解を発見できる
- 機能の欠陥を検出できる
- 使い勝手の悪い操作や問題のある機能を発見できる
- 動作可能なシステムの実現により，最終システムの実現性や有用性が部分的ではあるが明らかになる

3. Web サービスプロトタイピングシステム設計

3.1 システム要件

本項では，プロトタイピングシステムの要件を明確にする．

(R1)Web サービスの選択

Web サービスアプリケーションには，ネットワーク上に分散する Web サービスとの柔軟な連携が望まれる．そのため，システムは，既存の Web サービス，新規に構築する Web サービスに関わらず，プロトタイプ対象のサービスと容易に連携できる必要がある．これは，Web サービスの特徴である疎結合性によるものである．システムには，サービスの変更や再構築の効率性を高めることが要求される．

(R2)トポロジーの設定

複数の Web サービスを利用し，Web サービスアプリケーションを実現する際，Web サービスの連携方法は，複数存在する．そのため，システムには，Web サービスの連携トポロジーを容易に変更できる必要がある．

(R3)性能の計測

Web サービスアプリケーションの性能評価には，Web サービスのミドルウェア部およびネットワーク部のオーバーヘッドが無視できない．そのため，システムでは，性能面を見積もれることが必要である．具体的には，応答時間を計測できること，ネットワークの遅延を容易に設定できることが必要である．

(R4)システムの使いやすさ

Web サービスアプリケーションの開発には，経験や知識が必要とされる．そのため，システムは，設定を簡単にでき，最低限の知識で利用できることが重要である．また，プロトタイピングが開発の初期段階で行われるために，システムには，プロトタイプを短時間で構築できることが望まれる．

3.2　Web サービス連携トポロジー

Web サービスは，複数のサービスの連携，再利用を強く意識した**サービス指向アーキテクチャ**（SOA: Service Oriented Architecture）の考えに基づいている．サービス指向アーキテクチャとは，互いに協調動作するソフトウェアが，自由に連結しやすいような仕組みや方法である．Web サービスアプリケーションにおいて，1 つの CA が複数の Web サービス（以降 WS とする）と連携する場合，CA-WS 間，または WS-WS 間において，様々なサービス呼び出し形態（連携方式）が考えられる．この連携方式を **Web サービス連携トポロジー**（以降 **WS トポロジー**とする）と呼ぶ．

複数 WS の連携における最小単位として，1 つの CA と 2 つの WS の連携を考える．まず，CA が，WS1 にサービスを要求し，その応答を一旦受け取ってから，CA が，WS2 にサービスを要求する方式が考えられる．本論文では，このような方式を**リダイレクト型**と呼ぶ．図 4 は，リダイレクト型のトポロジーを UML（Unified Modeling Language）のコラボレーション図に基づいて表現したものである．

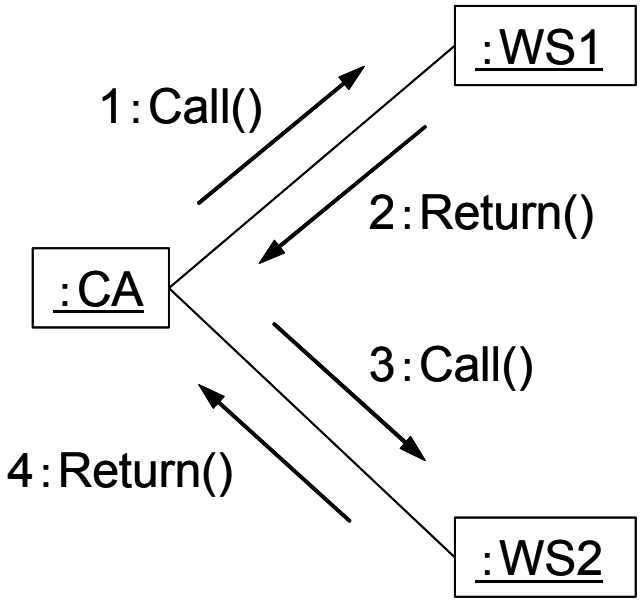


図 4　リダイレクト型

図 4 において，CA が，WS1，WS2 を呼び出す手順は，以下の通りである．

1. CA は，WS1 へサービスを要求する
2. WS1 は，CA へ応答する
3. CA は，WS2 へサービスを要求する
4. WS2 は，CA へ応答する

もう 1 つの最小単位の連携方式として，CA が，WS1 にサービスを要求すると，その WS1 が，CA の代わりに WS2 にサービスを要求する方式が考えられる．本論文では，このような方式を**プロキシ型**と呼ぶ（図 5）．

図 5 において，CA が，WS1 を呼び出す手順は，以下の通りである．

1. CA は，WS1 へサービスを要求する
2. WS1 は，WS2 へサービスを要求する
3. WS2 は，WS1 へ応答する
4. WS1 は，CA へ応答する

次節では，WS トポロジーの記法について述べる．

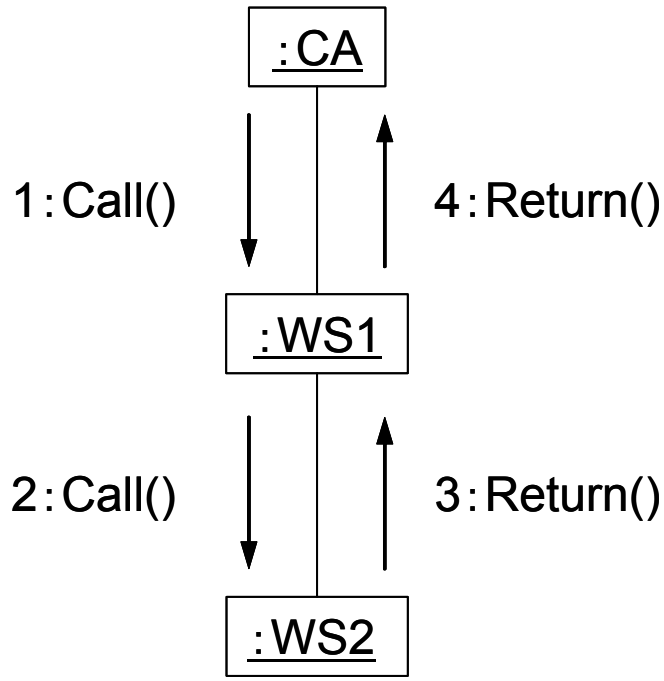


図 5　プロキシ型

3.3 Web サービス連携トポロジー式

ここで，任意の WS トポロジーを表現するための記法，**WS トポロジー式**（以降トポロジー式とする）を導入する．図 6 に BNF（Backus Naur Form）による文法定義を示す．

トポロジー式の意味を簡単に述べる．全てのトポロジー式は CA，すなわちクライアントアプリケーションから始まる．これはサービスの開始が CA によって引き起こされることを表す．式中の「;」は 2 つの WS の直列的な逐次呼び出しを表し，プロキシ型の連携を形成する．また，「+」は同一オブジェクトが 2 つの WS を順番に呼び出すことを意味し，リダイレクト型の連携を表す．「()」は式における優先度を制御するために用いられる．例として，図 4 および図 5 のトポロジーT1, T2 は，それぞれ，T1 = CA; (WS1 + WS2), T2 = CA; WS1; WS2 で表現される．

1 つの CA と連携する WS の総数を n とする場合，可能な WS トポロジーの総数を T(n)とすると，$T(n) = 2^{(n-1)}$ となる．例として，n=2, 3, 4 の全ての WS トポロジーを図 7 に示す．n=2 の場合，図 7(1)(2)，n=3 の場合，図 7(3)～(6)，n=4 の場合，図 7(7)～(14)の WS トポロジーとなる．すなわち，リダイレクト型は，図 7(1)，プロキシ型は，図 7(2)であり，すべての WS トポロジーは，リダイレクト型とプロキシ型の組み合わせで表現できる．

```
WS トポロジー:
    CA ';' 式
式:
    プロキシ式
    リダイレクト式
    '(' 式 ')'
プロキシ式:
    Web サービス
    Web サービス ';' 式
リダイレクト式:
    Web サービス '+' 式
Web サービス:
    "WS[0-9]+"
```

図 6　WS トポロジー式

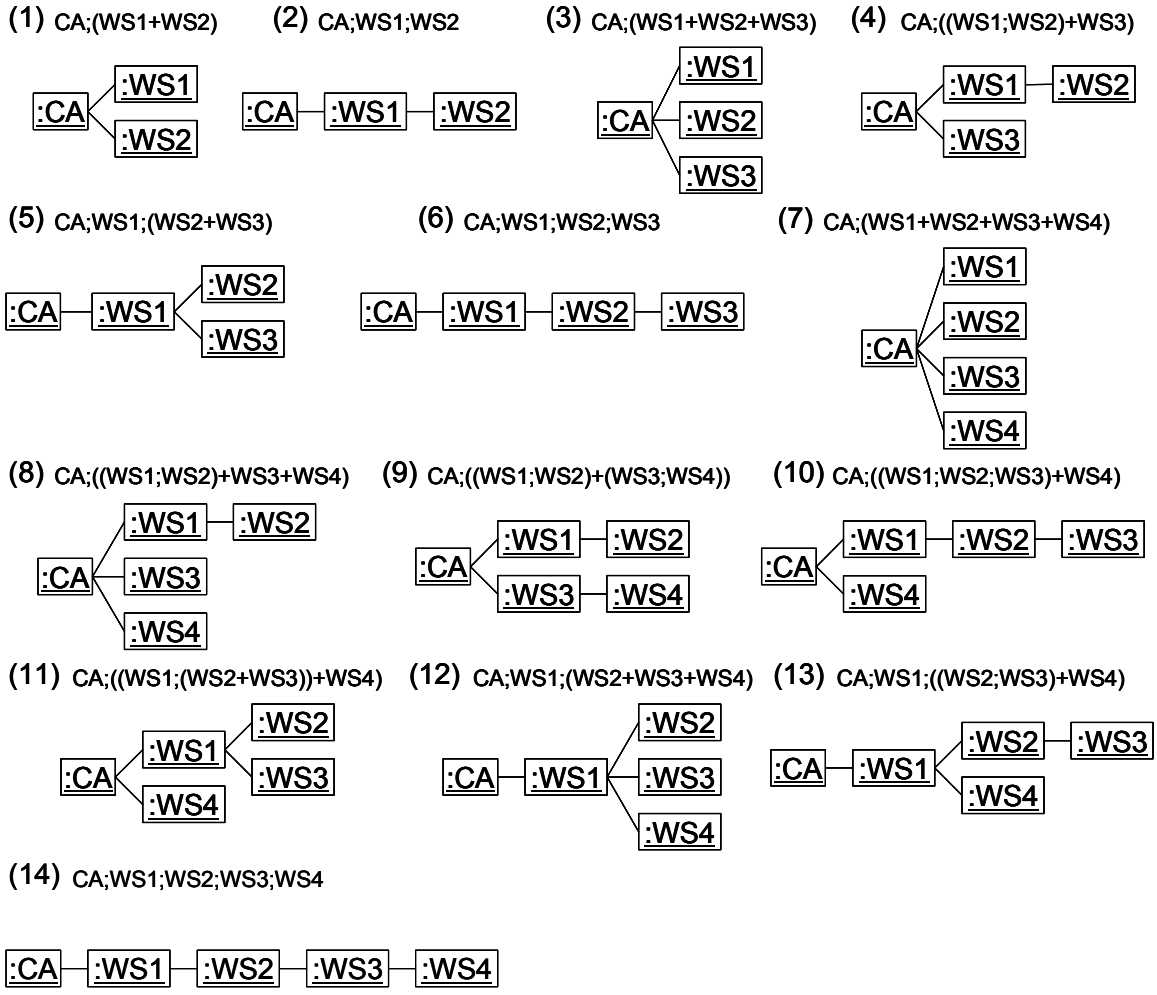


図 7　WS トポロジー(n=2, 3, 4)

3.4　Web サービス定義ファイル

WS トポロジー（式） は，複数の WS 間の大域的な呼び出し関係を表したものである．しかし，実際にトポロジーに従い，WS の統合を実現するには，各 WS が局所的にどの WS を呼び出せばよいのかを知る必要がある． **Web サービス定義ファイル**（以降 **WS 定義ファイル**とする）は，各 WS および CA の局所的な動作を定義する．具体的に，WS 定義ファイルは，次段の WS 呼び出しに必要となる「呼び出す WS 名」,「呼び出す WS のメソッド名」,「メソッドに渡すパラメータ情報」を格納している．

WS 定義ファイルは，XML（Extensible Markup Language）で記述する．XML は，文書の論理や意味構造を記述するための言語である．WS 定義ファイルの拡張子は，「.xml」とする．WS 定義ファイルの例を図 8 に示す．

WS 定義ファイルの文書型は，DTD（Document Type Definition）によって定義する．DTD では，XML の構成要素（element），指定できる属性（attribute），要素同士の関係などについて定義する．WS 定義ファイルの DTD（**wsprove.dtd** と呼ぶ）を図 9 に示す．

WS 定義ファイルは，13 の要素から構成される．そのうち，WS_Method 要素，CallMethod 要素，Parameter 要素は，要素の付属情報として属性を持つ．属性は，属性名と属性値から構成される．例えば，図 8 の WS_Method 要素は，属性名 Name，属性値 Add の属性を持つ．

各要素には，要素同士の親子関係が定義されている．これは，ファイルの操作や解析の際に役立つ．wsprove.dtd で定義される各要素の関係を木構造で図 10 に示す．例えば，Parameter 要素は，Name，Type，Value の 3 つの子要素を持ち，CallMethod を親要素に持つ．各要素の詳細については，以下の通りである．また，要素の一覧を表 2 に示す．

- WS_CONFIG 要素

WS 定義ファイルのルート要素である．ルート要素とは，最上位の要素で，必ず記述されなければならない．

- ServiceSimulationID 要素

プロトタイピング番号を記述する．プロトタイピング番号は，開発者がプロトタイピングに付与した固有の番号である．

- WS_Name 要素

定義する WS の名前を記述する．

- WS_ProcessingTime 要素

定義する WS 固有の処理時間（ミリ秒）を記述する．

- WS_Method 要素

定義する WS のメソッドを記述する．

- WS_Method 要素

属性 Name の属性値にメソッド名を指定する．

- CallMethod 要素

連携 WS の呼び出しメソッドを記述する．

- CallMethod 要素

属性 SequenceNumber の属性値に連携 WS の実行順序，属性 Name の属性値に連携 WS のメソッド名を指定する．

- ServiceProviderURI 要素

連携 WS の WSDL（Web Services Description Language）ファイルへの URL を記述する．WSDL ファイルとは，Web サービスの持つ機能や利用するための方法などが記述されているファイルである．

- CWS_Name 要素

連携 WS の名前を記述する．

- NetworkDelay 要素

  ネットワーク遅延時間（ミリ秒）を記述する．

- Parameter 要素

  連携 WS のメソッドに渡すパラメータを記述する．Parameter 要素では，属性 Index の属性値に引数の順番を指定する．

- Name 要素

  連携 WS の WSDL ファイルに定義されている引数名を記述する．

- Type 要素

  連携 WS の WSDL ファイルに定義されている引数型を記述する．

- Value 要素

  連携 WS のメソッドに渡すパラメータの実引数を記述する．

```
<?xml version="1.0" encoding="utf-8" ?>
<WS_CONFIG>
        <ServiceSimulationID>1</ServiceSimulationID>
        <WS_Name>CalcWS01</WS_Name>
        <WS_ProcessingTime>100</WS_ProcessingTime>
        <WS_Method Name="Add">
                <CallMethod SequenceNumber="1" Name="Add">
                        <ServiceProviderURI>
                                http://metrics/Simulation/CalcWS02/
                                CalcWS02.asmx?wsdl
                        </ServiceProviderURI>
                        <CWS_Name>CalcWS02</CWS_Name>
                        <NetworkDelay>20</NetworkDelay>
                        <Parameter Index="1">
                                <Name>augend</Name>
                                <Type>int</Type>
                                <Value>2</Value>
                        </Parameter>
                </CallMethod>
        </WS_Method>
</WS_CONFIG>
```

図 8　WS 定義ファイルの例

```
<!DOCTYPE WS_CONFIG [
<!ELEMENT
WS_CONFIG
(ServiceSimulationID,WS_Name,WS_ProcessingTime,WS_Method+)>
<!ELEMENT ServiceSimulationID (#PCDATA)>
<!ELEMENT WS_Name (#PCDATA)>
<!ELEMENT WS_ProcessingTime (#PCDATA)>
<!ELEMENT WS_Method (CallMethod*)>
<!ELEMENT
CallMethod (ServiceProviderURI,CWS_Name,NetworkDelay,Parameter*)>
<!ELEMENT ServiceProviderURI (#PCDATA)>
<!ELEMENT CWS_Name (#PCDATA)>
<!ELEMENT NetworkDelay (#PCDATA)>
<!ELEMENT Parameter (Name,Type,Value)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
<!ATTLIST WS_Method Name CDATA #REQUIRED>
<!ATTLIST CallMethod
        SequenceNumber      ID      #REQUIRED
        Name                CDATA #REQUIRED
>
<!ATTLIST Parameter Index ID #REQUIRED>
]>
```

図 9　wsprove.dtd

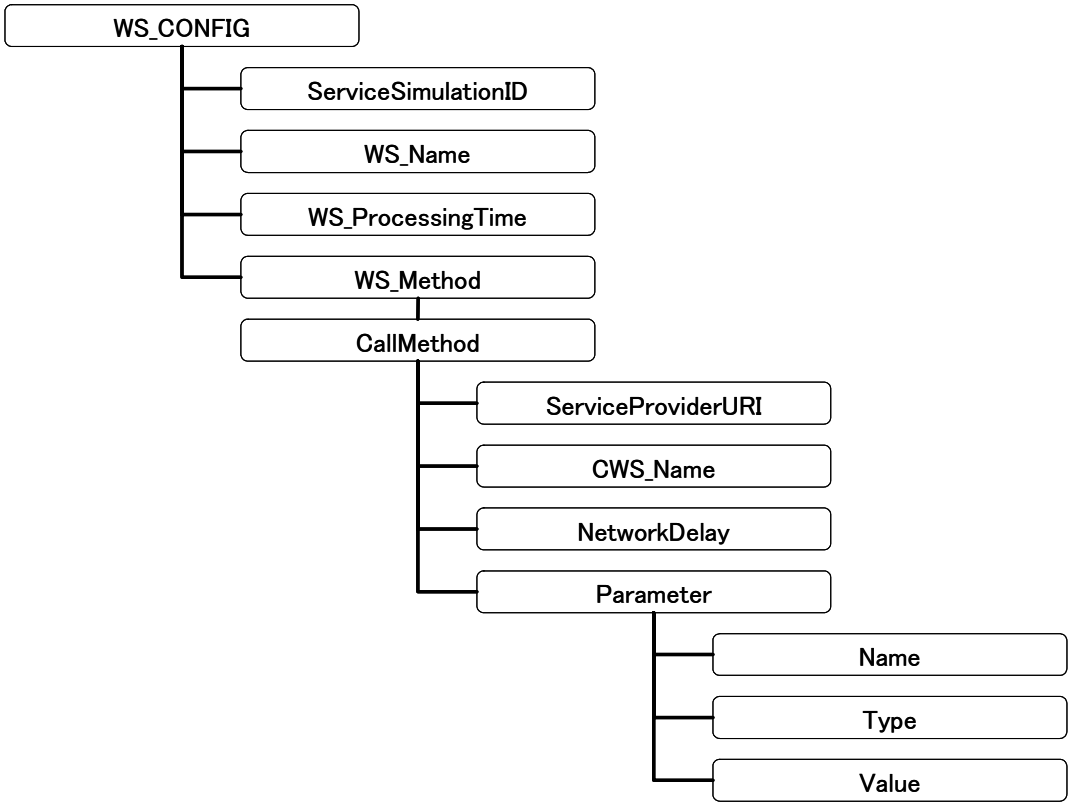

WS_CONFIG
ServiceSimulationID
WS_Name
WS_ProcessingTime
WS_Method
CallMethod
ServiceProviderURI
CWS_Name
NetworkDelay
Parameter
Name
Type
Value

図 10　要素の木構造

表 2　WS 定義ファイルの要素一覧

要素名	説明	属性名	説明
WS_CONFIG	WS定義ファイルのルート要素	−	−
ServiceSimulationID	プロトタイピング番号	−	−
WS_Name	定義するWSの名前	−	−
WS_ProcessingTime	定義するWSの固有処理時間(ミリ秒)	−	−
WS_Method	定義するWSのメソッド	Name	定義するWSのメソッド名
CallMethod	連携WSのメソッド呼び出し	SequenceNumber	連携WSの実行順序
		Name	連携WSのメソッド名
ServiceProviderURI	連携WSのWSDLファイルへのURL	−	−
CWS_Name	連携WSの名前	−	−
NetworkDelay	ネットワーク遅延時間(ミリ秒)	−	−
Parameter	連携WSのメソッドに渡すパラメータ	Index	引数の順番
Name	連携WSのWSDLファイルに定義されている引数名	−	−
Type	連携WSのWSDLファイルに定義されている引数型	−	−
Value	連携WSのメソッドに渡すパラメータの実引数	−	−

4. Web サービスプロトタイピングシステム実装

4.1 システム概要

我々は,Web サービスアプリケーションにおけるプロトタイピングを実現するため，**Web サービスプロトタイピングシステム**（以降 **WS-PROVE: Web Service PROtotyping and Validation Environment**）を開発した.

WS-PROVE は，WS を任意の WS トポロジーに従って動的に統合し，統合された WS 全体（以降**統合 WS** とする）の性能を計測する．ユーザが WS トポロジー式を与えると，各 WS について，それが自ら呼び出す次の WS（以降**次段 WS** とする）の集合とその呼び出し順序が決定する.プロトタイピングシステムでは,この次段 WS 集合と呼び出し順に関する情報を,各 WS が参照する WS 定義ファイルとして保持する．各 WS は，実行時に WS 定義ファイルを参照し，次段 WS を動的に決定，呼び出しを行う．こうして，与えられた WS トポロジーに従った統合 WS が実行される．WS-PROVE は，その実行中に，統合 WS と各 WS の性能を計測する．WS-PROVE は，**WS 定義ファイル**（第 3.4 節参照），**連携 WS** および**連携 CA** の 3 つから構成される.

連携 WS は，プロトタイプの対象となる開発中の Web サービスをシミュレートする．その Web サービスが次段 WS の情報を WS 定義ファイルから実行時に取得し，動的呼び出しを行う．また，WS 定義ファイルから処理時間を取得し，一定時間スリープすることで完成版の処理のシミュレートを行う．複数の次段 WS を呼び出す場合は，現在呼び出し中の次段 WS の応答を受け取ってから，次の次段 WS を呼び出す方式（同期的呼び出し）を採用する．連携 WS は，処理を依頼された時刻，各次段 WS を呼び出した時刻，処理が終了した時刻にログを書き出す.

連携 CA は，CA をシミュレートするアプリケーションである．連携 WS との本質的な違いは，WS トポロジーにおいて，最初に実行されることだけで，連携 WS の機能を内包する．つまり，連携 CA は，連携 WS と同様に，与えられた WS 定義ファイルを元に動的に次段 WS を呼び出す．連携 CA は，実行時刻に関するログを書き出す．評価実験用に，統合 WS のシミュレーション回数を指定で

きる機能も備えている．システム概要図を図 11 に示す．

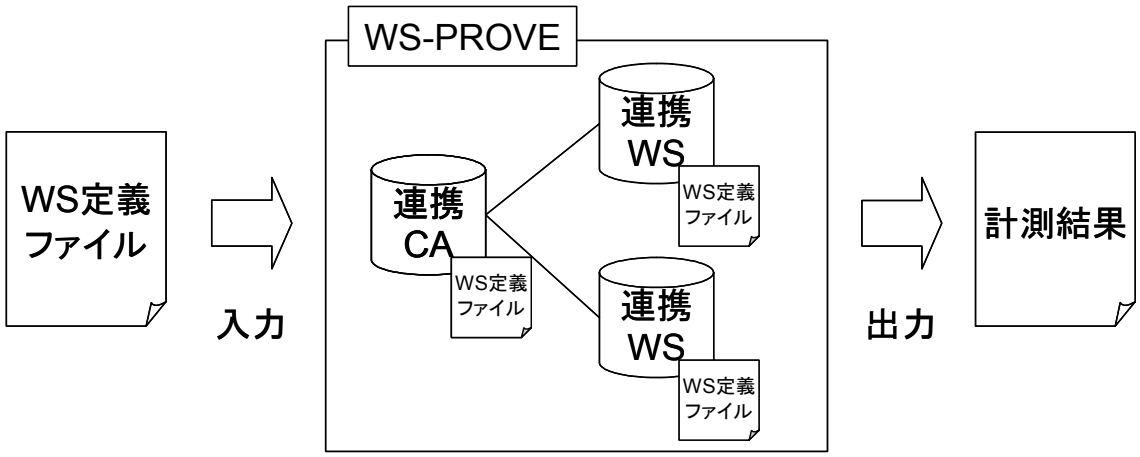

WS-PROVE
WS定義
ファイル
入力
連携
CA
WS定義
ファイル
連携
WS
WS定義
ファイル
連携
WS
WS定義
ファイル
出力
計測結果

図 11　システム概要図

4.2　システム実装

第 3 章でのシステム設計を基に，WS-PROVE を実装した．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

リダイレクト型の統合 WS を例にしたシステム詳細図を図 12 に示す. これは，2 つの Web サービス WS1，WS2 および 1 つの CA をプロトタイプするシナリオである．連携 CA は，コンソールアプリケーションとして実装した．また，連携 WS の動的呼び出し部の実装においては，実行時バインドに基づくアプローチを採った．これは，呼び出す可能性のある全ての WS を起動する DLL（Dynamic Link Library）とダミーの WS をあらかじめ作成しておく．DLL は，プロトタイプの対象となる WS の WSDL（Web Services Description Language）ファイルから生成する．次に，ダミーWS は，実行時に WS 定義ファイルを参照して，適切な DLL を動的リンクし（図 12(2)(7)）目的の連携 WS を呼び出す（図 12(3)(8)）というアプローチである．

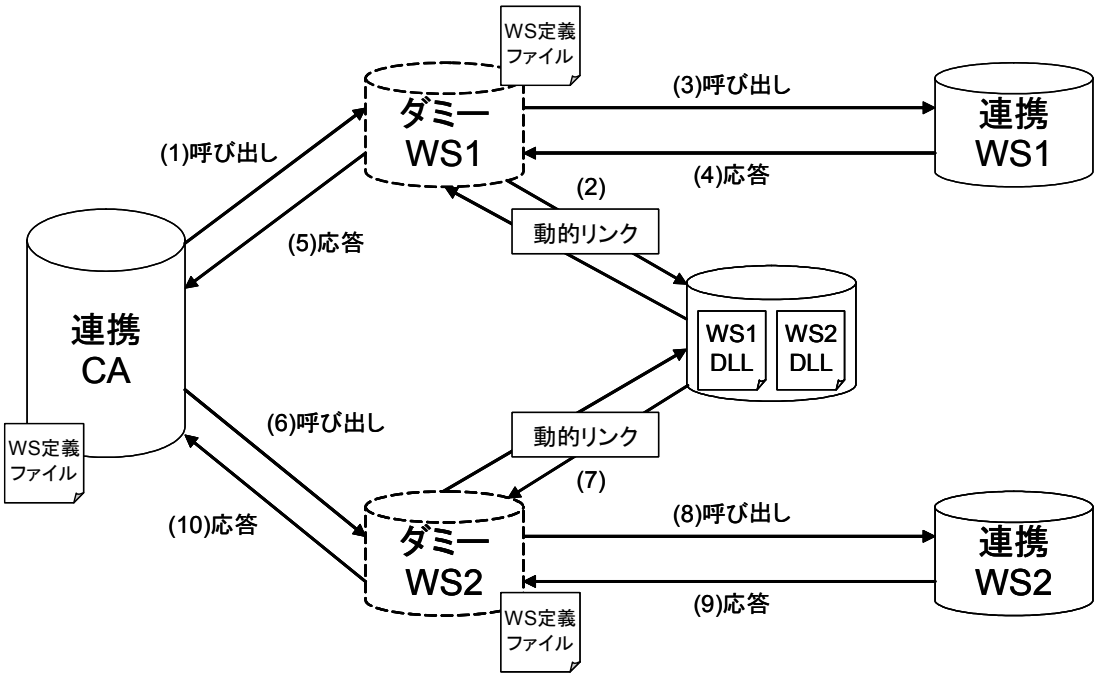

WS定義
ファイル
ダミー
WS1
(3)呼び出し
連携
WS1
(1)呼び出し
(4)応答
(2)
動的リンク
(5)応答
連携
CA
WS1
DLL
WS2
DLL
(6)呼び出し
動的リンク
WS定義
ファイル
(7)
(10)応答
(8)呼び出し
ダミー
WS2
連携
WS2
(9)応答
WS定義
ファイル

図 12　システム詳細図

本システムの実行手順は，以下の通りである．

1. 統合 WS 開発に必要な WS を準備する．新規に WS を作成，あるいは，既存の WS を検索し，発見する．
2. 各 WS の WSDL からプロキシクラスのソースを生成する．
3. 各プロキシクラスのソースコードから DLL を生成する．
4. ダミーWS を必要な数だけ作成する．
5. WS 定義ファイルを作成する．
6. WS 定義ファイルを入力し，連携 CA を実行する．

5. ケーススタディ

本章では，実装したシステムを用いて，4 つの WS トポロジーに対してプロトタイピングを行い，システムの有効性を実証する．実験用 Web 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

実験では，連携 CA，すべての連携 WS を 1 台のサーバ内に配置する．

5.1 プロトタイピングのシナリオ

実験では，図 13 に示した 4 つの WS トポロジーのプロトタイピングを行う．連携 CA・各連携 WS には，WS 定義ファイルによって処理時間を設定する．処理時間とは，連携 CA および連携 WS が処理を開始してから処理を終えるまでの時間である．また，連携 CA と連携 WS，連携 WS と連携 WS の間には，ネットワーク遅延を設定する．処理時間，ネットワーク遅延の設定は，以下のように行う．

E1　連携 CA・各連携 WS の処理時間は等しく，ネットワーク遅延が無い
　処理時間を 100 ミリ秒，ネットワーク遅延を 0 秒に設定する．

E2　連携 CA・各連携 WS の処理時間は等しく，ネットワーク遅延がある
　処理時間を 100 ミリ秒，ネットワーク遅延を表 4 に基づき設定する．例えば，CA と WS6 の間のネットワーク遅延は，300 ミリ秒になる．

E3　連携 CA・各連携 WS の処理時間は異なり，ネットワーク遅延が無い
処理時間を表 3 に基づき設定する．ネットワーク遅延を 0 秒に設定する．

E4　連携 CA・各連携 WS の処理時間は異なり，ネットワーク遅延がある
処理時間を表 3 に基づき設定する．ネットワーク遅延を表 4 に基づき設定する．

各 WS トポロジーに条件 1 から 4 までをそれぞれ適用し，1 つの WS トポロジーにつき 4 種類の実験を行う．サービス 1 回の処理時間は，極めて短いため，連携 CA で繰り返し回数を 100 回に設定して処理時間を計測した．この結果から，連携 CA および連携 WS それぞれの処理時間の合計と平均を得る．

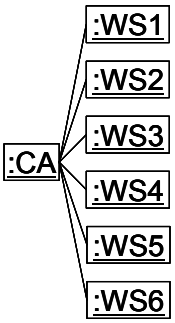
表 3　処理時間の設定（ミリ秒）

	CA	WS1	WS2	WS3	WS4	WS5	WS6
処理時間	100	200	300	400	500	600	700

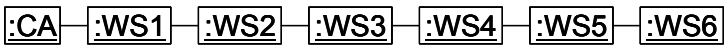
表 4　ネットワーク遅延の設定（ミリ秒）

	WS1	WS2	WS3	WS4	WS5	WS6
CA	50	100	150	200	250	300
WS1		50	100	150	200	250
	WS2		50	100	150	200
		WS3		50	100	150
			WS4		50	100
				WS5		50

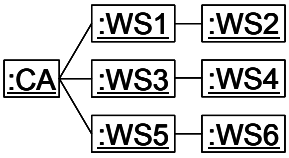
(1) CA;(WS1+WS2+WS3+WS4+WS5+WS6)



(2) CA;WS1;WS2;WS3;WS4;WS5;WS6



(3) CA;((WS1;WS2)+(WS3;WS4)+(WS5;WS6))



(4) CA;((WS1;(WS2+WS3))+(WS4;(WS5+WS6))

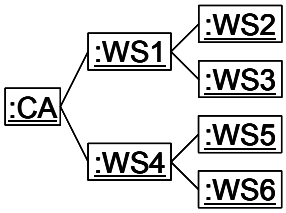


図 13　評価する WS トポロジー

5.2　プロトタイプの評価

システムは，すべての実験用トポロジーの性能を計測することができた．我々は，先行研究[9]において，1 つの連携 CA と 2，3，4 個の連携 WS で構成されるトポロジーでもシステムの動作を確認している．この結果からシステムは，様々なトポロジーに対応できる．

E1，E3 の動作時間を表 5，表 6 に示す．動作時間とは，連携 CA および連携 WS の個別の処理時間であり，繰り返し回数 100 回の平均値となっている．各トポロジーの終端に位置する連携 WS は，設定した処理時間の-6～+11 ミリ秒の範囲に収まっている．また，E1 において，同一トポロジー内で同じ形のノードに位置する連携 WS では，動作時間がほぼ同じである．この結果から CA と各連携 WS は，設定した処理時間通りに動作していることがわかる．

次に，E2，E4 におけるトポロジー(3)のネットワーク遅延を表 7 に示す．ネットワーク遅延は，設定の-8～+12 ミリ秒の範囲に収まっている．他のトポロジーの遅延についても同様の結果が得られた．これらの結果から設定したネットワーク遅延通りに動作していることがわかる．

次に，E2，E4 の動作時間を表 8，表 9 に示す．遅延を考慮しても，各トポロジーの終端に位置する連携 WS は，設定した処理時間の-3～+11 ミリ秒の範囲に収まっている．また，E2 においても，同一トポロジー内で同じ形のノードに位置する連携 WS では，動作時間がほぼ同じである．これらの結果からも CA と各連携 WS は，設定した処理時間通りに動作していることがわかる．

表 5　E1 動作時間（ミリ秒）

	(1)	(2)	(3)	(4)
CA	796.70	805.55	805.87	799.10
WS1	106.65	692.66	224.21	337.14
WS2	103.75	575.67	107.56	107.25
WS3	104.53	458.46	224.62	103.58
WS4	103.97	341.14	107.31	337.76
WS5	105.09	224.31	224.77	107.17
WS6	105.10	107.35	107.31	103.68

表 6　E3 動作時間（ミリ秒）

	(1)	(2)	(3)	(4)
CA	2872.81	2884.73	2884.18	2874.22
WS1	200.45	2771.18	520.88	930.40
WS2	307.02	2560.32	310.42	310.28
WS3	400.00	2240.13	912.41	400.11
WS4	494.62	1826.11	497.95	1821.35
WS5	604.57	1318.22	1318.43	607.19
WS6	698.56	701.09	701.11	696.37

表 7　E2・E4 遅延（ミリ秒）

	(3)
WS1	61.70
WS2	101.75
WS3	148.73
WS4	195.56
WS5	242.27
WS6	304.59

	(3)
WS1	61.69
WS2	101.72
WS3	148.68
WS4	195.55
WS5	242.40
WS6	304.78

表 8　E2 動作時間（ミリ秒）

	(1)	(2)	(3)	(4)
CA	1868.20	1181.22	1433.49	1385.17
WS1	107.55	1005.65	286.87	505.84
WS2	107.55	826.05	107.47	107.60
WS3	107.62	646.48	287.02	107.60
WS4	107.51	466.50	107.73	505.76
WS5	107.53	287.07	287.00	107.68
WS6	107.53	107.56	107.47	107.59

表 9　E4 動作時間（ミリ秒）

	(1)	(2)	(3)	(4)
CA	3946.32	3259.40	3514.26	3463.70
WS1	201.37	3083.96	583.83	1099.52
WS2	310.62	2810.42	310.75	310.75
WS3	404.46	2427.75	974.40	404.48
WS4	498.20	1951.10	498.24	1990.09
WS5	607.61	1380.93	1380.78	607.63
WS6	701.25	701.34	701.45	701.42

6. 考察

開発した WS-PROVE を第 3.1 節の要件(R1)～(R4)の実現という観点から評価する．(R1)については，システムが特定の WS に限らず利用できることや WS 定義ファイルによって変更が可能なことから実現されている．(R2)については，プロトタイプのトポロジーを WS 定義ファイルによって自由に変更できることで実現されている．(R3)については，第 4.2 節の結果からもわかるように，満足な性能を確認できていることから十分である．(R4)については，システムに GUI が実装されていないことやダミーの WS を手動で作成しなければならないなどシステムの一部が自動化されていないことから不十分である．

従来のプロトタイピングでは，開発の迅速さが要求されるために性能が無視される．しかし，Web サービスアプリケーションにおけるプロトタイピングでは，性能を重視しなければならない．これは，従来のプロトタイピングが Web サービスアプリケーションに適用できないことを示唆している．今後，Web サービスアプリケーションへ適用可能なプロトタイピングモデルの提案が望まれる．現状では，性能を考慮してプロトタイピングを行うならば，WS-PROVE が機能型プロトタイピングに当てはまることになる．

7. まとめ

本論文では, Web サービスアプリケーションにおけるプロトタイピングシステムの要件を整理し, WS-PROVE の設計, 実装, 評価を行った. また, WS-PROVE を用いて, いくつかの Web サービスアプリケーションのプロトタイピングおよび性能見積もりを行い, WS-PROVE の有効性を示した.

今後の課題としては, 実現されていない要件についてシステムへの実装を進める. 具体的には, 作業の自動化を含め, 機能の充実を図り, 使いやすいシステムにすることが挙げられる. 将来的には, Web サービスアプリケーション開発方法論についての研究へとつなげていきたい.

謝辞

本研究を進めるに当たり多くの方々に御指導，御協力，御支援戴きました．ここに謝意を添えて御名前を記させて戴きます．本当にありがとうございました．

奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授には，本研究の全過程において多大な御指導を賜りました．研究への取り組み方，研究に対する姿勢といった研究者としての基本を学びました．また，日常生活においても多大な御助力を賜りました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 小山 正樹 教授には，ソフトウェア開発技術において御指導を賜りました．新しい研究分野でしたが，臆することなく邁進できました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 飯田 元 助教授には，ソフトウェアプロセス改善技術において御指導を賜りました．実践的なソフトウェア開発プロセスの知識を得ることができました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 Mike Barker 客員教授には，プロジェクトマネジメント技術において御指導を賜りました．毎日の昼食では，楽しいひと時を過ごせました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 助教授には，ソフトウェアプロテクション技術において御指導を賜りました．ソフトウェアの直面している複雑な現実問題を理解することができました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 中村 匡秀 助手には，本研究の全過程において御指導を賜りました．研究の小さな積み重ねを無駄にせず，形にすることが大切であると痛感しました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 大平 雅雄 助手には，ソフトウェアの要求分析において御指導を賜りました．研究の新たな興味分野を広げることができました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座 Web サービス班の皆様には，日々の研究，論文執筆，システム開発において御指導，御協力を賜りました．毎回のミーティングから生まれるアイデアには，驚きや感動

がありました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座の皆様には，研究発表の準備，毎日の生活において多大な御助言，御協力を賜りました．素晴しい仲間たちと出会い，2年間の大学院生活は，楽しく充実したものでした．心より厚く御礼申し上げます．

最後に，研究に専念できる環境を支えてくれた家族に心から感謝します．

参考文献

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

[2] 有澤誠，ソフトウェアプロトタイピング，近代科学社，1986.

[3] 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.

[4] 福田直樹，肥塚八尋，和泉憲明，山口高平，“オントロジーに基づく Web サービスの自動連携”, 信学技報, KBSE2004-3, pp.13-18, May.2004.

[5] GeOap，http://www.geoap.jp

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

[7] 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.

[8] 本多泰理，“Web サービスのトランザクション処理の性能設計技術に関する一検討”, 信学会総合大会講演論文集, 2004-3, pp.235, Mar.2004.

[9] 石井健一，串戸洋平，井垣宏，中村匡秀，松本健一，“複数 Web サービス連携手法の実験的評価”, 信学技報, IN2004-58, pp.75-80, Sep.2004.

[10] 石井健一，大葉直史，尾島陽子，勝靖宏，福嶋一史，古沢亜利耶，ステップアップ XML 活用法，池辺正典，北川悦司，吉田博哉（編），田中成典（監修），工学社，2002.

[11] 石川冬樹，田原康之，吉岡信和，本位田真一，“Web サービス連携のためのモバイルエージェント動作記述”, 情報処理学会論文誌, Vol.45, No.6，pp.1614-1629，June.2004.

[12] 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.

[13] 片山卓也，土居範久，鳥居宏次，ソフトウェア工学大辞典，朝倉書店，1998.

[14] 近藤秀和，村岡洋一，“Web サービスを利用した次世代型オンラインブックマークシステム”, 信学技報，DE2004-4，pp.19-24，June.2004.

[15] 来間啓伸，本位田真一，“ポリシに基づく Web サービス・コミュニティ連合のモデル”, 情報処理学会論文誌, Vol.45, No.6, pp.1593-1602, June.2004.

[16] 串戸洋平，石井健一，山内寛己，井垣宏，玉田春昭，中村匡秀，松本健一，“Web サービスアプリケーションのソフトウェアメトリクスに関する考察”, 信学技報，ネットワークシステム研究会，NS2003-316，pp.113-118，March 2004.

[17] 村山隆彦，由良俊介，“Web サービスのビジネス運用に向けて”，信学技報，IN2003-166，pp.37-42，Jan.2004.

[18] 中島毅，田村直樹，岡田和久，和田雄次，“顧客対話時の要求獲得と確認を支援するプロトタイピングツール：ROAD/EE”，電子情報通信学会論文誌，Vol.J85-D-I，No.8，pp.725-740，Aug.2002.

[19] 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.

[20] 大西淳，郷健太郎，要求工学，共立出版，2002.

[21] 大野邦夫，“オフィス文書と XML”, 信学技報，IN2003-167, pp.43-50, Jan.2004.

[22] 山登康次，武本充治，島本憲夫，“ユビキタス環境に適したサービス連携フレームワークにおけるサービステンプレート生成手法”，信学技報，IN2003-169，pp.1-6，Jan.2004.

[23] 寺口正義，山口裕美，西海聡子，伊藤貴之，“ストリーミング処理による Web サービスおよび Web サービスセキュリティの軽量実装”，情報処理学会論文誌，Vol.45，No.6，pp.1603-1613，June.2004.

[24] 寺井公一，和泉憲明，山口高平，“オントロジーとパターンに基づく

Web サービス連携”, 信学技報, KBSE2003-39, pp.35-40, Jan.2004.

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

[26] 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/

[27] 山本晃治, 上原忠弘, 松尾明彦, 大橋恭子, 猪股順二, 松田友隆, 山本里枝子, “XML アプリケーション開発のためのパターン体系の開発”, 情報処理学会論文誌, Vol.45, No.6, pp.1584-1592, June.2004.

[28] 山岡和雄, 佐藤厚, 山本茂樹, 瀧口修, 岡庭祐, 小原利光, 宮下慎一, “障害管理用統合 NMS 構築における Web サービスの適用事例”, 信学技報, IN2003-160, pp.1-6, Jan.2004.

[29] .NET XML Web Services Repertory,

“Dynamically invoke XML Web Services (improved version)”,

http://xmlwebservices.cc/index_Samples.htm