NAIST-IS-MT0351052

修士論文

Web サービスにおける ソフトウェアメトリクスの提案と実験的評価

串戸 洋平

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。

串戸 洋平

審査委員：

松本 健一 教授　（主指導教員）
小山 正樹 教授　（副指導教員）
飯田 元 助教授　（副指導教員）

Web サービスにおける ソフトウェアメトリクスの提案と実験的評価*

串戸 洋平

内容梗概

近年，Web サービスを用いたシステム開発が注目を集めている．しかし，それらシステムの開発方法論について十分に議論されていない．そこで，我々は Web サービスの連携方式と品質との関係について研究を行ってきた．本論文では，Web サービスアプリケーションの品質特性を定量的に評価することを目的に，4 種類の新たなソフトウェアメトリクスを提案しその有効性について評価する．具体的には，まずサービス指向アーキテクチャとオブジェクト指向設計の違いに着目し，既存のオブジェクト指向メトリクスを Web サービスアプリケーションへ適用可能かを考察する．この考察結果を基に，新たに 4 種類の Web サービスメトリクス (RFWS, NOWS, EMWS, NHTWS) を提案する．そして，提案メトリクスと品質特性の関係について 3 種類の評価実験を行う．一つ目は我々が開発した Web サービスアプリケーションでの評価，二つ目は Web サービスアプリケーションのプロトタイプを短時間に構築できる WS-PROVE (Web Service Prototyping and Validation Environment) を用いた効率性評価，三つ目は Sum of Disjoint Products (SDP) という信頼性評価アルゴリズムを用いた信頼性評価である．評価実験の結果，RFWS, NOWS, NHTWS の 3 個のメトリクスに関して効率性と信頼性に関連が認められた．

キーワード

Web サービス, メトリクス, 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-IS-MT0351052, 2005 年 2 月 3 日.

Design and Evaluation of Software Metrics for Web Service*

Youhei Kushido

Abstract

Web service applications are one of the most emerging applications in the networked computing.However, there has been no systematic methodology to evaluate the quality of the Web service applications yet. To quantitatively evaluate the quality of the applications, this thesis presents four new software metrics and evaluates in experiments the relationship between the quality of Web Service applications. First, I see the difference between service-oriented architecture and object-oriented design, and validate the applicability of the conventional object-oriented metrics. Based on the validation, I propose four new metrics (RFWS, NOWS, EMWS and NHTWS) for Web service applications. And I evaluated in experiments the proposed metrics by three methods. First, I applied the proposed metrics to Web Service application that have been constructed by our group. Secondarily, I evaluated by WS-PROVE (Web Service Prototyping and Validation Environment). Finally, I applied SDP (Sum of Disjoint Products) algorithm to my metrics and evaluated it. The empirical result showed that the proposed metrics have a relevance to performance and reliability of the Web service application.

Keywords:

Web service, Metrics, Prototyping

* Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT0351052, February 3, 2005.

目 次

図 目 次

表 目 次

1. はじめに

ネットワークの進歩とソフトウェアの多様化により，多種多様なサービスが溢れている．このような中，より高度なサービスを効率的に実現するためにサービスの連携が必要になってきており，その一手段として Web サービスが注目されている [2]．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，分岐したサービス間の疎結合を標準化された手順 (XML, SOAP/HTTP, UDDI) により実現するものである．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 [4]．これにより，既存サービスの再利用性を向上し，標準化された通信手段によって，異なるシステム間の連携が効率よく実現できるとされている．

近年 Web サービスを用いたシステムがいくつか開発されつつある (例：Google Web APIs[5], Amazon Web Service[1]) ．しかし，サービス指向アーキテクチャに基づいた Web サービスは，実地運用が開始されてからまだ日が浅く，体系だった開発方法論は十分に議論されていない．また，我々の知る限りでは，Web サービスアプリケーションの品質特性を評価する手段も存在しない．

そこで本論文では，Web サービスアプリケーションの品質特性を定量的に評価することを目的に，4 種類の新たなソフトウェアメトリクスを提案する．具体的には，まずサービス指向アーキテクチャとオブジェクト指向設計の違いに着目し，既存のオブジェクト指向メトリクスである C&K メトリクスの Web サービスアプリケーションへの適用の可能性を考察する．この考察結果を基に，新たに 4 種類の Web サービスメトリクス (RFWS, NOWS, EMWS, NHTWS) を提案する．さらに，提案メトリクスと品質特性の関係について 3 種類の評価実験を行う．一つ目は，我々が開発した 3 種類の方法で実装された Web サービスアプリケーションに提案メトリクスを適用し，それらの性能計測値と提案メトリクスの関連性を考察する．二つ目は，WS-PROVE (Web Service Prototyping and Validation Environment) という，Web サービスアプリケーションのプロトタイプを短時間で構築できるシステムを用いて，様々なプロトタイプの性能計測値と提案メトリクスの関連性を考察する．三つ目は，Sum of Disjoint Products (SDP) という信

頼性アルゴリズムを用いて，様々な連携構成の Web サービスアプリケーションの信頼性を導出し，提案メトリクスとの関連性を考察する．

評価実験の結果，Web サービスアプリケーションが Web サービスを利用することで生じる非機能的な部分での効率性・信頼性 (Web サービス単体での効率性・信頼性ではなく，Web サービスの連携における効率性・信頼性) に関連が認められた．

2. ソフトウェア品質特性とソフトウェアメトリクス

本章では，一般的なソフトウェアの品質として定義されるソフトウェア品質特性 (ISO/IEC9126 - JIS X0129 [7]) と，それらソフトウェア品質特性を定量的に評価することを目的とするソフトウェアメトリクスについて紹介する．

2.1 ソフトウェア品質特性

ソフトウェア品質を分析するための定性的な枠組みとして，ソフトウェア品質特性が提案されている (ISO/IEC9126 - JIS X0129 [7]，図 1 参照)．

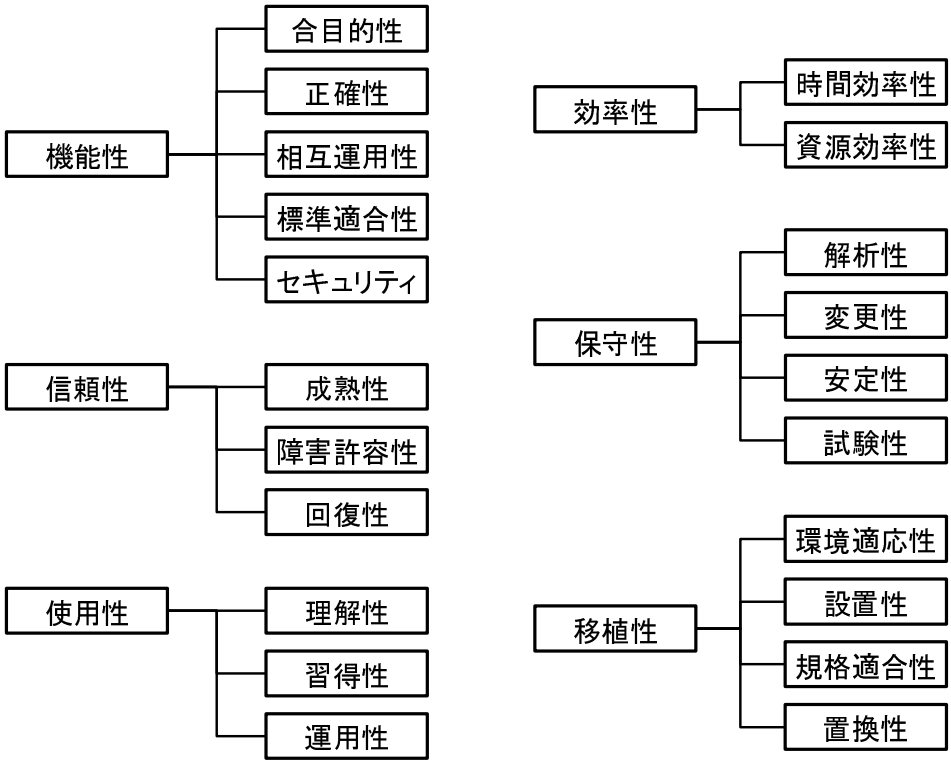


図 1 ソフトウェア品質特性

この規格では，ソフトウェアの品質の分類として，6 種類の品質特性とそれらをさらに詳細化した 21 種類の品質副特性を定めている．これらの品質特性は，ソ

フトウェアに要求される特徴的な品質特性を抽出する (定性化) ときに取捨選択される．抽出された品質特性は，数値化した基準値を当てはめる (定量化) ことにより，実際に作成されたソフトウェアの品質の評価 (定量的な評価) に役立つ．以降，代表的な 6 種類の品質特性について簡単に説明する．

一つ目は「機能性」である．機能性はソフトウェアがある目的をもって求められる必要な機能を実装している度合いである．例えば，ユーザの要求どおりのソフトウェアなのか，ソフトウェアが正しく動くかといったことや，相互運用性・標準適合性・セキュリティといったもの等が機能性にあたる．

二つ目は「信頼性」である．信頼性は実装している機能があらゆる条件の下で機能要件を満たして (必要な期間) 正常動作し続けることができる度合いである．例えば，ソフトウェアの障害の頻度や障害が起こっても表面上正しく動作する度合い，また障害が起こってから回復するまでの度合い等が信頼性にあたる．

三つ目は「使用性」である．使用性はソフトウェアの使いやすさや使用するのにかかる労力の度合いである．ソフトウェアの使い方の理解しやすさや習得のしやすさ，またソフトウェアを用いる際の導入のしやすさ等が使用性にあたる．

四つ目は「効率性」である．効率性はソフトウェアに要求される条件における，ソフトウェアがもつ目的達成の度合いと，使用する資源の量の関係である．例えば，ソフトウェアの機能を実行する際に時間がどのくらいかかるのか，資源をどれだけ消費するのかといったことが効率性にあたる．

五つ目は「保守性」である．保守性はソフトウェアの改訂や維持に関する労力の度合いである．例えば，保守を行うにあたって保守をする部分を特定する労力や保守全体の行程に関する労力，また他には保守作業によって発生する障害混入の度合いや保守に用いるテストに要する労力等が保守性にあたる．

六つ目は「移植性」である．移植性はソフトウェアをある環境から別の環境下に移した場合のソフトウェアの能力を推し量る度合いである．例えば，ソフトウェアが違う環境に移植され用いられても正常に動作するかといったことやその移植のためのコスト，他にはソフトウェアが規格に適合しているのかといったことや異なるソフトウェアと置き換えて使用する場合の労力等が移植性にあたる．

2.2 ソフトウェアメトリクス

ソフトウェアメトリクス (以降メトリクスと呼ぶ) とはソフトウェアに定義される様々な品質特性 (定性的なもの) を客観的な数学的尺度によって定量的に評価するものである．ソフトウェアの開発において早期に適切なメトリクスを用いて品質や進捗状況などを定量的に評価することは，後の行程での工数割り当てや品質管理のための指標として非常に有益なものとなる [13, 18]．

現在まで様々なメトリクスが提案されてきたが，ここではオブジェクト指向メトリクスとして有名な Chidamber らの複雑度メトリクス (C&K メトリクス) について説明する [13]．C&K メトリクスではソースコード中のクラスに注目して，主に保守性に関する以下の 6 つのメトリクスを提案している (表 1)．

表 1 C&K メトリクスと品質特性との関係

	計測対象	評価品質
WMC	メソッドの複雑さ	保守性
DIT	スーパクラスの数	保守性と使用性
NOC	サブクラスの数	保守性
CBO	関係するクラスの数	保守性
RFC	関係するメッセージの数	保守性
LCOM	クラスの凝集性	保守性

WMC(Weighted Methods per Class) あるクラスのメソッドをアルゴリズムの複雑さに基づき重み付けしその和を計測する．メソッドの重み付けにはそのメソッドの行数，変数の数等を用いるが，クラスのメソッドがどれも同等の複雑さだった場合は単純にクラスのメソッド数である．WMC が高いほど複雑であり保守性が下がる．

DIT(Depth of Inheritance Tree) あるクラスのスーパクラスの数を計測する．DIT が高いほど継承されている変数やメソッドが多いということであり，再利用性は向上するが，使用性や保守性が下がる．

NOC(Number Of Children) あるクラスのサブクラスの数を計測する．NOC が高いほどサブクラスへの影響が高いので保守性が下がる．

CBO(Coupling Between Objects) あるクラスに関係しているクラスの数を計測する．CBO が高いほど他のクラスに依存していることになり保守性が下がる．

RFC(Response For a Class) あるクラスに関係しているメッセージの数を計測する．RFC が高いほどメッセージの数が多いということであり，テスト・デバッグにコストがかかり保守性が下がる．

LCOM(Lack of Cohesion Of Methods) あるクラスの凝集性の欠如を計測する．LCOM が大きいほど変数を共有している部分が多いことを表しメンテナンスにコストがかかり保守性が下がる．

3. Web サービスとサービス指向アーキテクチャ

3.1 Web サービス

Web サービスとは，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．OS やプログラミング言語に依存することなく，異なるアプリケーション間の連携を可能にするサービス指向アーキテクチャの考えに基づいている [3]．図 2 に Web サービスの利用形態について示す．Web サービスは，UDDI レポジトリと呼ばれる Web サービスを管理するサーバに WSDL と呼ばれる規格に従い提供するサービスを登録する (図 2 (0))．UDDI レポジトリは，登録された Web サービスを利用するためのインターフェース (Web サービスメソッド) を公開し，アプリケーションが利用したいサービスを検索できるよう WSDL ファイルを管理する役目を持つ．クライアントアプリケーション (以降 CA と呼ぶ) は，UDDI レポジトリに登録された WSDL ファイルの情報をもとに Web サービスの場所を特定し (図 2 (1)(2))，公開された Web サービスメソッドを通じて Web サービスの機能を利用する (図 2 (3)(4))．この Web サービスメソッドの呼び出しは，リモートプロシージャコール (RPC) によって遠隔的に行われるが，RPC に伴うメッセージ授受は，.NET Framework や Apache Axis 等の Web サービスミドルウェアによって標準化された手続き (XML, SOAP) に自動的に変換され行われる．そのため，CA はオブジェクト指向プログラミングにおけるクラスのメソッド呼び出しとほぼ同じ方法で Web サービスを利用できる．つまり，CA の開発者は SOAP メッセージの書式や送受信の手順 (HTTP 等)，Web サービスの内部ロジック (Web サービスで提供される機能のアルゴリズム) 等を一切気にすることなく Web サービスの機能をアプリケーションに組み込むことが出来る (疎結合と呼ばれる)．

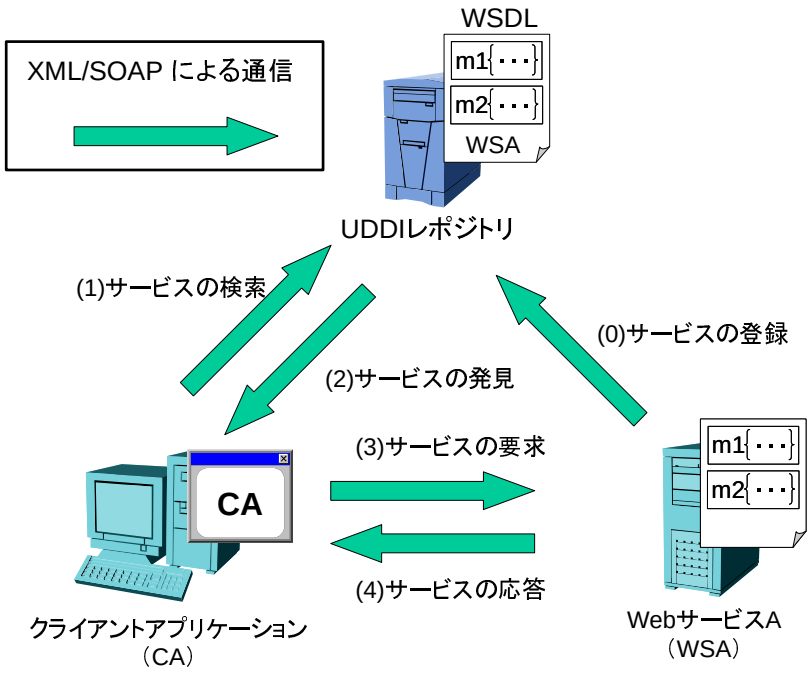


図 2　Web サービスの利用形態

3.2　サービス指向とオブジェクト指向の違い

サービス指向で，最も重要な概念は疎結合である．Web サービスはクライアントアプリケーション (CA) や他の Web サービスとの連携において疎な結合を行う．疎結合を実現するために，一旦公開された Web サービスはそのインターフェースを変更することは原則的に許されない． CA の開発者は Web サービスの実装内部を知ることが出来ないため，勝手なインターフェース仕様の変更はその Web サービスを利用する全ての CA のテストのやり直しにつながるからである．一方，疎結合の制約が特に規定されていない一般のオブジェクト指向開発では，オブジェクトが密に結合される場面が少なくない．このような場合では，あるオブジェクトに変更があった場合は開発者がその変更に対してアプリケーションが受ける影響等を考慮しつつプログラムしなおさなければならないという多大な労力が必要

となる．

また，サービス指向開発において「サービスの継承」という概念は存在しない．サービスの提供者の立場からは，公開するサービスがどのように実際に利用されるかはわからないし，利用する側からはそのサービスの全ての機能を継承して新たなサービスを作るというスタイルはとらない．

3.3 オブジェクト指向メトリクスの適用可能性

サービス指向においては，オブジェクト指向では規定されていなかった「サービス間の疎結合」という新たな概念が存在する．また,「サービス間での継承」と言うものもない．従って,「クラスとサービスの結合の違い」と「継承関係の有無」の点から従来のオブジェクト指向メトリクスを Web サービスアプリケーションにそのまま適用することは難しいと思われる．ここでは，2.2 節で述べた C&K メトリクスを取り上げて考察する．

まず，CBO メトリクスをそのまま Web サービスアプリケーションに適用することは問題があると思われる．Web サービスにおいては，疎結合を実現するために，Web サービスメソッドへのアクセスのためのインターフェースが変更されることは原則的に許されていない．そのため，インターフェースが変更されない限り Web サービスの機能を利用できるという点から，Web サービスの実装内部の修正によるアプリケーションへの影響は少ないと考えられる，従って，アプリケーションに関連する Web サービスの数が，必ずしも保守性の低下につながるとは言えない．3.2 節で述べたように，オブジェクト指向での結合とサービス指向での結合には違いがあるため，この事を考慮する必要がある．

次に，オブジェクト指向におけるクラスには継承関係があるが，Web サービスには継承関係はない．従って，DIT および NOC を Web サービスに適用することはできない．オブジェクト指向におけるクラスのメソッド呼び出しと Web サービスメソッドの呼び出しはコード内では同じ形でメソッド呼び出しが実行されるため一見似ているが，クラスのメソッド呼び出しは実行されているマシン上のコードを呼び出しにいくのに対し，Web サービスメソッドの呼び出しはミドルウェアによって SOAP 要求に変換され Web サービスに要求メッセージを送るといった

形になる．そのため，実際の動作においてはWebサービスの利用のためのオーバーヘッドを考慮する必要があり，慎重にソフトウェアメトリクスを考える必要がある．

既存のオブジェクト指向メトリクスは，CAまたはWebサービス単体の評価においてはそれらを構成するコードがオブジェクト指向で設計されていれば適用可能であるが，CAとWebサービスの連携，およびWebサービス間の連携の評価においては，新たなメトリクスが必要になると考える．

4. Webサービスメトリクスの提案

本章では，オブジェクト指向メトリクス (C&K メトリクス) での Web サービスへの適用の可能性を考慮し，我々は以下の 4 つのメトリクスを提案する [10].

- **RFWS**(Response For a Web Service)
- **NOWS**(Number Of Web Services)
- **EMWS**(Effective Methods per Web Service)
- **NHTWS**(Number of Hop to Terminal Web Service)

以降の各メトリクスの定義において，CA をメトリクスの計測対象とするクライアントアプリケーション，$W_i(1 \leqq i \leqq n, n$: 利用する Web サービスの数) を Web サービス (WS) とする．Web サービス W_A が別の Web サービス W_B を呼び出す場合には，W_A を CA とみなして各メトリクスを算出する．

4.1 RFWS(Response For a Web Service)

定義:

$W_1, W_2, \ldots, W_n$ を CA が呼び出す全ての WS とする．このとき，CA の RFWS メトリクスを，以下のように定義する．

RFWS = CA と $W_i(1 \leqq i \leqq n)$ との間での要求・応答メッセージの総和

説明:

RFWS メトリクスとは，CA とその CA が呼び出す全ての Web サービスについて，CA からのサービスの要求メッセージと Web サービスからの応答メッセージの数の総和を計測する (図 3)．ここで言う Web サービスとは，ある Web サーバ上に実装されている一つの Web サービスのことで，Web サービスにおいて複数の Web サービスメソッドが実装されていてもそれらは一つの Web サービスとみなす．また，同じ Web サーバ上に複数の Web サービスが実装されている場合で

はそれらの Web サービスは別の Web サービスとして扱う．RFWS メトリクスを計測することにより，CA の Web サービスの利用の度合いがわかり，効率性・信頼性を評価できると考える．

効率性に関して

RFWS メトリクスが大きければ，CA と Web サービスとのメッセージのやり取りが多いということになり，処理的にボトルネックとなりうるネットワークの利用の度合いが高い事が予想され，(特に遠隔地の Web サーバ等に実装されている Web サービス等を利用する場合) 効率性は悪くなると考えられ，効率性を評価できると考える．

信頼性に間して

RFWS メトリクスを計測することにより同時に CA の Web サービスへの依存の度合いがわかり，信頼性について評価できると考える．RFWS が高ければ，CA はネットワークを通して Web サービスとメッセージのやり取りをすることが多いということになり，CA と Web サービスが実装されている Web サーバでの間での通信状況によっては，要求・応答メッセージのタイムアウトや，予期せぬネットワーク障害にあい信頼性が下がると考えられ，信頼性を評価できると考える．

RFWS メトリクスが大きいほどコンポーネント間でのメッセージのやり取りの数が多いという点で既存の RFC メトリクスと類似しているが，RFC メトリクスで評価できるとされるテスト・デバッグにかかる労力については有用な評価はできないと考える．理由としては，CA と Web サービスとの関係は疎結合であるため，仮に Web サービス側で修正があったとしても，CA は Web サービス側が Web サービスメソッドのインターフェースを変更しないかぎりサービスの利用という点で影響を受けないからである (もちろん，Web サービス側の修正により効率性などで影響を受けるかもしれないが，Web サービス側の修正に合わせて CA 側を修正しなければ Web サービスを利用できないということは無い)．また，Web サービスは通常ネットワークを通して利用されるため，一般的に一つのマシン上

での実行が想定されるクラス間のメッセージ数を計測・評価する RFC メトリクスでは有用な評価をできないと考える．

具体例:

RFWS メトリクスの具体例について説明する．図 3 では，CA が呼び出す Web サービスは WS1 と WS2 の二つであり，それぞれの Web サービスとやり取りするメッセージの数は 2 であり，RFWS メトリクスは 4 となる．ここで，RFWS メトリクスを変化させるためには，CA と CA が呼び出す Web サービス群との間でやりとりされるメッセージ数を変化させる必要がある．具体的には，CA の RFWS メトリクスを増加したければ，CA が呼び出す Web サービスの数を増やせばよい (図 4)．図 4 では，図 3 における WS3 を CA が呼び出すことで，CA が呼び出す Web サービスの数を増やす事により，RFWS メトリクスは WS3 とのメッセージのやり取りの数だけ増加し 6 となる．

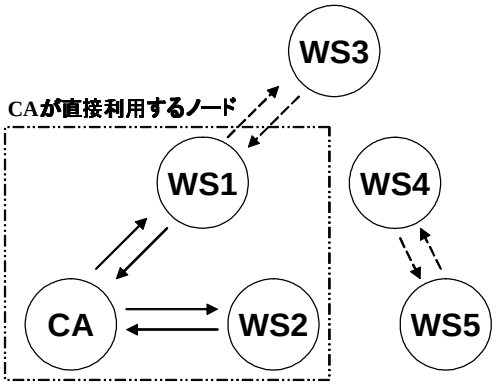


図 3 RFWS メトリクス

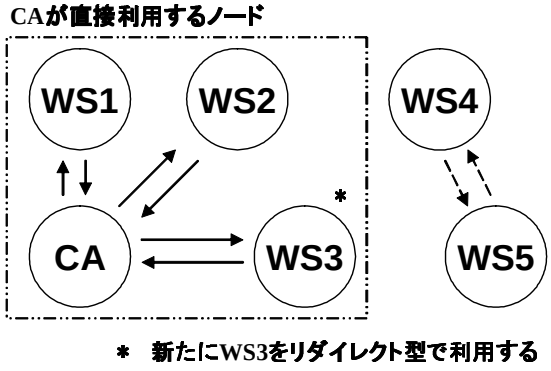


図 4 RFWS の値を増加させる方法

4.2 NOWS(Number Of Web Services)

定義:

$W_1, W_2, \ldots, W_n$ を CA が直接的・間接的に利用する全ての WS とする．このとき，CA の NOWS メトリクスを以下のように定義する．

$$\mathrm{NOWS} = n$$

説明:

NOWS メトリクスとは，CA が直接的または間接的に利用する全ての Web サービスの数を計測する (図 5)．ここで言う Web サービスとは，ある Web サーバ上に実装されている一つの Web サービスのことで，Web サービスにおいて複数の Web サービスメソッドが実装されていてもそれらは一つの Web サービスとみなす．また，同じ Web サーバ上に複数の Web サービスが実装されている場合ではそれらの Web サービスは別の Web サービスとして扱う．NOWS メトリクスを計測することにより，CA が Web サービスを直接的・間接的に利用している度合いがわかり，効率性・信頼性・保守性について評価できると考える．

効率性に関して

NOWS メトリクスが大きければ，CA が Web サービスを多く利用しているということになり，CA 内で Web サービスが提供する機能と同等の機能を実行するよりも，Web サービスとやり取りするためのオーバーヘッド (ミドルウェアによるサービス要求の作成や，ネットワークを利用することの通信遅延など) が大きくなるなるため効率性が悪くなることが考えられ，効率性を評価できると考える．

信頼性に関して

NOWS メトリクスが高ければ，CA が Web サービスを多く利用しているということになり，CA とそれらの Web サービスとの間の通信状況によっては，要求・応答メッセージのタイムアウトや，予期せぬネットワーク障害にあい信頼性が下がることが考えられ，信頼性を評価できると考える．

保守性に関して

NOWS メトリクスを計測する事により，Web サービスの疎結合の特徴から保守性を評価できると考える．NOWS メトリクスが高ければ，Web サービスの疎結合性より，Web サービスが提供する機能の保守に関して CA が考慮するべき労力が減少し保守性が良くなると考えられ，保守性を評価できると考える．

具体例:

NOWS メトリクスの具体例について説明する．図 5 では，CA が直接的に利用する Web サービスは WS1 と WS2 の二つであり，間接的に利用する Web サービスは WS3 の一つである．従って NOWS メトリクスは 3 となる．ここで，NOWS メトリクスを変化させるためには，CA が直接的に利用する Web サービスの数を変化させるか，間接的に利用する Web サービスの数を変化させればよい (図 6)．図 6 では，CA は新たに WS4 を間接的に利用し (*1)，また新たに直接的に WS5 を利用している (*2) ので NOWS メトリクスは 5 となる．

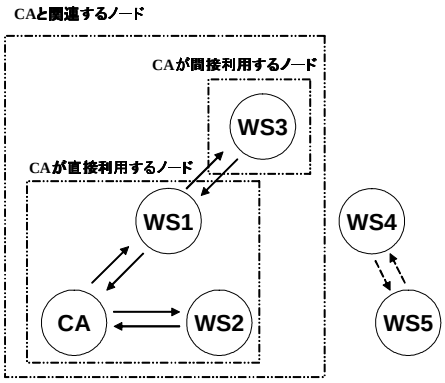


図 5 NOWS メトリクス

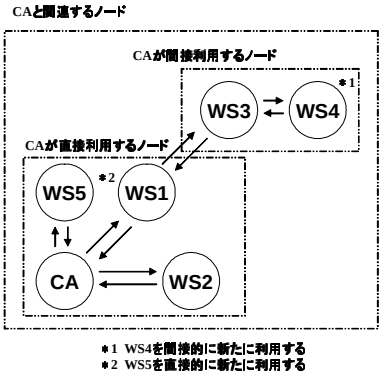


図 6 NOWS の値を増加させる方法

4.3 EMWS(Effective Methods per Web Service)

定義:

$W_1, W_2, \ldots, W_n$ を CA が呼び出す全ての WS とする．このとき，CA の EMWS メトリクスを以下のように定義する．

$$\begin{aligned} \text{EMWS} &= \text{EM} \div \text{PM} \\ \text{EM} &: W_1, W_2, \cdots, W_n\text{の利用メソッド数の総和} \\ \text{PM} &: W_1, W_2, \cdots, W_n\text{の公開メソッド数の総和} \end{aligned}$$

説明:

EMWS メトリクスとは，CA が呼び出す Web サービス群に関して，CA が利用している Web サービスメソッド数の総和を，CA が呼び出す Web サービスの公開 Web サービスメソッド数の総和で割った値を計測する (図 7)．ここで言う Web サービスとは，ある Web サーバ上に実装されている一つの Web サービスのことで，Web サービスにおいて複数の Web サービスメソッドが実装されていてもそれらは一つの Web サービスとみなす．また，同じ Web サーバ上に複数の Web サービスが実装されている場合ではそれらの Web サービスは別の Web サービスとして扱う．EMWS メトリクスを計測することにより，CA が呼び出す Web サービス群に関して，Web サービスが公開している Web サービスメソッドに対する CA の利用する Web サービスメソッドの利用の度合いがわかり，CA が利用している Web サービスがどれだけ CA の目的に合致している Web サービスであったのか，つまり機能性が評価できると考える．

機能性に関して

EMWS メトリクスが 1 に近ければ，CA が呼び出す Web サービス群が CA が必要としている機能に特化した Web サービスを公開していることになり機能性がよいことになる．

具体例:

EMWS メトリクスの具体例について説明する．図 7 では，CA が呼び出す Web サービスは WS1 と WS2 であり，WS1 では Web サービスメソッドを 10 個公開

しており，WS2 では Web サービスメソッドを 2 個公開している．また，図 7 中において，CA は WS1 の Web サービスメソッドを二つと WS2 の Web サービスメソッドを一つ利用している．よって，EMWS メトリクスは利用する Web サービスメソッド数の総和を公開されている Web サービスメソッド数の総和で割るので $(2+1) \div (10+2) = 0.25$ となり EMWS メトリクスは 0.25 となる．ここで，EMWS メトリクスを変化させるには，分子である利用する Web サービスメソッドの数を変化させるか，分母である公開されている Web サービスメソッド数が異なる Web サービスを利用すればよい (図 8)．図 8 では，図 7 において呼び出していた WS1 の代わりに，同じサービスを提供するが公開メソッド数が 4 個である WS3 を CA が呼び出すことにより EMWS メトリクスが $(2+1) \div (4+2) = 0.5$ となる．

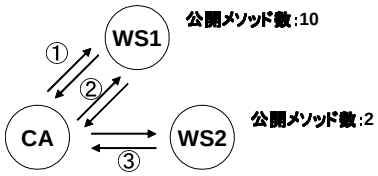


図 7 EMWS メトリクス

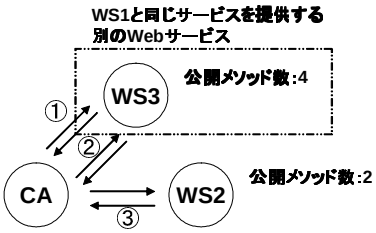


図 8 EMWS の値を増加させる方法

4.4 NHTWS(Number of Hop to Terminal Web Service)

定義:

$W_1, W_2, \ldots, W_k$ を WS とする．今，W_i が $W_{i+1}(0 \leqq i \leqq k)$ を要求するような系列

$$\rho = W_0, W_1, W_2, \cdots, W_k \quad (W_0 = \mathrm{CA})$$

を CA の WS 系列と定義する．又，ρ の長さ $(= k)$ をホップ数と定義し，$hop(\rho)$ と書く．CA が WS 系列 $\rho_1, \rho_2, \cdots, \rho_n$ を持つとき，CA の NHTWS メトリクスを，以下のように定義する．

$$\mathrm{NHTWS} = \mathrm{Max}\{hop(\rho_i)\}$$

説明:

NHTWS とは，CA が直接的または間接的に利用している Web サービス群に関して，Web サービス群の中で，ホップ数が最大となる Web サービスのホップ数の数値を NHTWS メトリクスの値とする (図 9)．ここで言う Web サービスとは，ある Web サーバ上に実装されている一つの Web サービスのことで，Web サービスにおいて複数の Web サービスメソッドが実装されていてもそれらは一つの Web サービスとみなす．また，同じ Web サーバ上に複数の Web サービスが実装されている場合ではそれらの Web サービスは別の Web サービスとして扱う．また，ホップ数とは，CA から Web サービス群のある Web サービス (WS_i) に至るまでにいくつの Web サービスを経由したかを表す数値である．NHTWS メトリクスを計測することにより CA が利用している Web サービスがどれだけ他の Web サービスに依存しているのかがわかり，効率性・信頼性を評価できると考える．

効率性に関して

NHTWS メトリクスが大きければ，CA が呼び出す Web サービスが潜在的に他の Web サービスを多く利用していることになり，呼び出す Web サービスが他の Web サービスとやり取りするためのオーバヘッド (ミドルウェアによるサービス要求の作成や，ネットワークを利用することの通信遅延など) の影響が大きくなるなるため効率性が悪くなることが考えられ，効率性を評価できると考える．

信頼性に関して

また，NHTWS が大きいほど CA が要求するサービスの実行に様々な Web サービスを経由することになり，Web サービス間での通信状況によっては，要求・応答メッセージのタイムアウトや，予期せぬネットワーク障害にあうことが考えられ信頼性が下がることが考えられ，信頼性を評価できると考える．

具体例:

NHTWS メトリクスの具体例について説明する．図 9 では，CA から WS1 までのホップ数は 1 であり，WS2 までのホップ数は 1 であり，WS3 までのホップ数は 2 である．よって，NHTWS メトリクスは利用している Web サービス群の中で最大のホップ数となる Web サービスのホップ数を NHTWS メトリクスとするので，図 9 においては WS3 までのホップ数が 2 で最大となり，NHTWS メトリクスは 2 となる．ここで，NHTWS メトリクスを変化させるためには最大となるホップ数を増やせばよい (図 10)．図 10 では，図 9 における WS3 と同じサービスを提供するが，WS5 を利用する WS4 を WS3 の代わりに利用することにより，最大のホップ数が CA から WS5 に至るホップ数の 3 となり，NHTWS メトリクスは 3 となる．

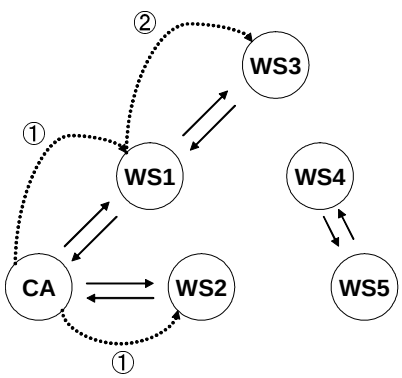


図 9 NHTWS メトリクス

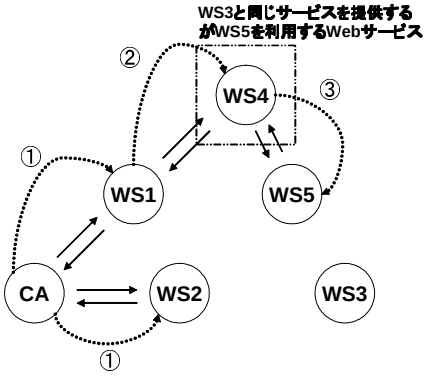


図 10 NHTWS の値を増加させる方法

4.5 Web サービスメトリクスと品質特性の関係

Web サービスメトリクスと品質において予想される関係について表 2 に示す．RFWS，NOWS，NHTWS の各メトリクスは，メトリクスが大きければそれだけネットワークを通して他の Web サービスを利用している (論理的にもしくは物理的にネットワークを利用する) ということを評価でき，効率性・信頼性を評価できると考えられる．また，EMWS については，CA が必要としているサービスにどれだけ Web サービスが特化しているかを評価でき，機能性を評価できると考えられる．

表 2 Web サービスメトリクスと品質特性の関係の予想

品質特性	メトリクス	評価内容
効率性	RFWS, NOWS, NHTWS	Web サービス利用の際の SOAP メッセージの変換といったオーバーヘッド
信頼性	RFWS, NOWS, NHTWS	ネットワークを経由することによる予期せぬネットワーク障害の混入等
保守性	NOWS	Web サービスの再利用性
機能性	EMWS	Web サービスが CA の必要としているサービスに特化しているか

5. Webサービスメトリクス評価実験

本章では，提案する4つのWebサービスメトリクス (以降，提案メトリクスと呼ぶ) の評価実験について述べる．提案メトリクスの評価を行うためには，まず各メトリクスの値が大きく異なるようなWebサービスアプリケーションを複数構築して応答時間などの品質を実際に計測する．次に，それらの品質と提案メトリクスの値とを比較する評価実験を行い，両者の関連性を評価する．本論文では，提案メトリクスについて3つの評価実験を行った [10, 11]．一つ目は，実際に構築したWebサービスアプリケーション (バス時刻表検索サービス [8]) の性能と提案メトリクスの評価実験である [10]．二つ目は，WS-PROVE(Web Service Prototyping and Validation Environment)[9] を用いてWebサービスアプリケーションプロトタイプを作成し，その性能計測結果と提案メトリクスの評価実験である [11]．三つ目は，Sum of Disjoint Products(SDP)[6, 14, 16] というネットワークにおけるノードの信頼性を評価することができるアルゴリズムを用いて，Webサービスアプリケーションを構成するWebサービスの信頼性と提案メトリクスの評価実験を行う [11]．以降，これらについて述べる．

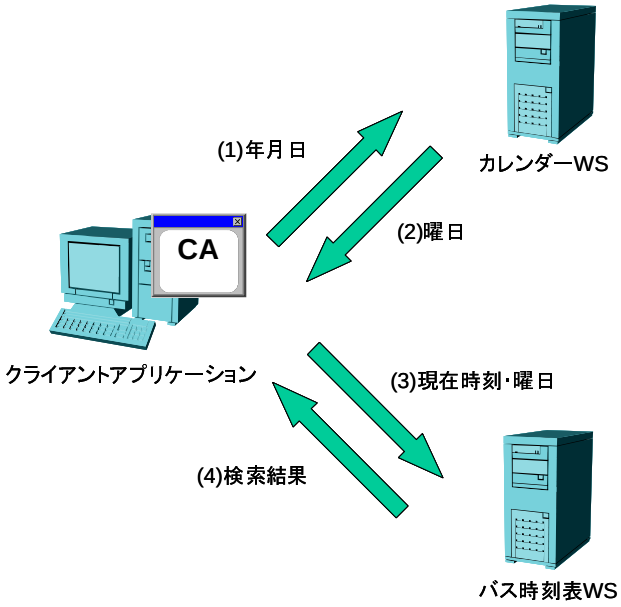
5.1 バス時刻表検索サービス

我々は文献 [8] において，Webサービスを用いた「バス時刻表検索サービス」を開発した．このWebサービスアプリケーションは，1つのクライアントアプリケーションCAと，2つのWebサービス (時刻表検索WS, カレンダーWS) から構成され，ユーザが現時刻から最も近いバスの到着時刻を1クリックで取得できる機能を提供する．我々はこの同一の仕様を基に，Webサービスの接続形態 (トポロジ) が与える影響を調べるために，図11に示す3種類の実装を行った．図11 (a) のリダイレクト型実装は，CAがカレンダーWSの結果を受け取ってから時刻表WSを呼び出す方式である．図11 (b) のプロキシ型実装は，時刻表WSがCAの代わりにカレンダーWSを呼び出す方式である．最後に図11 (c) のスタンドアロン型実装は，CAが時刻表WSの機能を取り込んだ実装となっている．これらのWebサービスアプリケーションの性能については，文献 [8] において，実行時

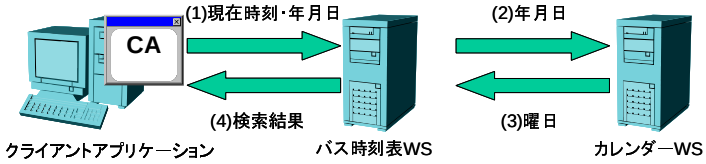
間の計測から効率性が，CA のソースコードの行数 (LOC) から保守性が実験的に評価されている．実験の結果については，表 3 の通りとなっており，1000 回の実行時間の計測からスタンドアロン型，プロキシ型，リダイレクト型の順に効率性が良く，LOC の値からプロキシ型，リダイレクト型が保守性の面においてスタンドアロン型より優れている結果となっている．また，表中に提案メトリクスを適用した結果も載せている．

表 3 バス時刻表サービスでの評価実験結果

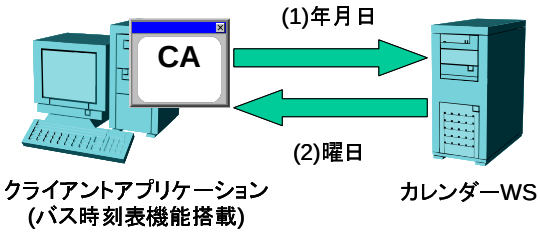
	プロキシ型	リダイレクト型	スタンドアロン型
実行時間 (秒)	16.98	21.65	11.39
LOC(行)	102	108	199
RFWS	2	4	2
NOWS	2	2	1
EMWS	1	1	1
NHTWS	2	1	1



(a)　リダイレクト型



(b)　プロキシ型



(c)　スタンドアロン型

図 11　バス時刻表検索サービスの 3 種類の実装方法 (文献 [8])

5.1.1 バス時刻表検索サービスによる評価実験結果・考察

提案する 4 つの Web サービスメトリクスのバス時刻表検索サービスへの適用結果とバス時刻表検索サービスの品質との関係について考察する (表 4).

RFWS メトリクスについて

RFWS メトリクスについては，信頼性と効率性について関連があると考えていたが，評価実験の結果，リダイレクト型が大きな値を示し，プロキシ型とスタンドアロン型が小さな値を示した．この事は，表 3 に示される「リダイレクト型はプロキシ型とスタンドアロン型に比べて実行時間が遅かった」という結果に一致し，RFWS メトリクスは効率性に関連があると思われる結果となった．信頼性についてはバス時刻表サービスで信頼性の計測がなされていなかったため評価できなかった．

NOWS メトリクスについて

NOWS メトリクスについては，信頼性と効率性と保守性について関連があると考えていたが，評価実験の結果，リダイレクト型とプロキシ型が大きな値を示し，スタンドアロン型が小さな値を示した．この事は，表 3 に示される「リダイレクト型とプロキシ型はスタンドアロン型に比べて実行時間が遅かった」という結果に一致し，NOWS メトリクスは効率性に関連がある思われる結果となった．また，表 3 に示される「リダイレクト型とプロキシ型はスタンドアロン型に比べて保守性の面で優れている」という結果にも一致し，NOWS メトリクスは保守性にも関連があると思われる結果となった．信頼性との関連については前述の通り評価を行う事はできなかった．

EMWS メトリクスについて

EMWS メトリクスについては，機能性について関連があると考えていたが，バス時刻表サービスでは，それぞれの Web サービスがそれぞれ目的の機能だけを果たすように開発されていたために，EMWS メトリクスとしては全てが同じ値となる結果となった．そのため，MWS メトリクスの機能性との関連について評価できなかった．

NHTWS メトリクスについて

NHTWS については，信頼性と効率性について関連があると考えていたが，評価実験の結果，プロキシ型が大きな値を示し，リダイレクト型とスタンドアロン型が小さな値を示した．この事は，表 3 に示される「プロキシ型はスタンドアロン型に比べて実行時間が遅かった」という結果に一致したが,「プロキシ型はリダイレクト型とくらべて実行時間が早い」という結果には一致しなかった．よって，NHTWS メトリクスは効率性に関して関連があるとは言えない結果となった．これは，CA からの NHTWS が，リダイレクト型とプロキシ型で両方ともに 1 でありそれらの連携方式の違いを評価できなかったためだと思われる．信頼性との関連については前述の通り評価を行う事はできなかった．

表 4 バス時刻表検索サービスによる評価実験結果

提案メトリクス	予想関連品質	実験で関連が見られた項目
RFWS	信頼性，効率性	効率性:実行時間
NOWS	信頼性，効率性，保守性	効率性:実行時間，保守性:LOC
EMWS	機能性	関連が見られる項目は無し
NHTWS	信頼性，効率性	関連が見られる項目は無し

5.2 WS-PROVE による効率性の評価

前節 5.1 で述べたバス時刻表検索サービスは小規模な Web サービスアプリケーションであったため，3 つの実装の性能差が大きく表れなかった．そのため提案メトリクスとの相関を完全に評価するにはいたらなかった．

そこで我々は，文献 [9] において自由に Web サービスアプリケーションのプロトタイプを構築できる WS-PROVE (Web Service Prototyping and Validation Environment) を開発した (図 12)．WS-PROVE は，Web サービスアプリケーションの開発において，初期段階で必要とされるプロトタイプを短時間で構築できる．プロトタイプは，実際の Web サービスに類似した仮想の Web サービスから構成され，プロトタイプを実際に動作させることにより，Web サービスを利用することによる SOAP 変換等のオーバーヘッドやネットワークでの遅延といった非機能的な要求を評価できる．具体的には，Web サービスの連携方式・個々の Web サービスの処理時間・Web サービス間での遅延を設定することにより，様々な条件のプロトタイプを構築でき，プロトタイプ全体と個々の Web サービスでの動作時間と実際にかかった Web サービス間での遅延を計測できる．WS-PROVE を用いた評価実験は，まず，各メトリクスの値が顕著に出るようにプロトタイプを構築し，個々の Web サービスの動作時間やシステム全体での動作時間を計測する．そして，計測された動作時間と提案メトリクスの比較を行い評価を行う．以降，効率性の評価ができると考えた 3 つの提案メトリクス (RFWS, NOWS, NHTWS) の評価実験用プロトタイプと評価実験方法について述べる．

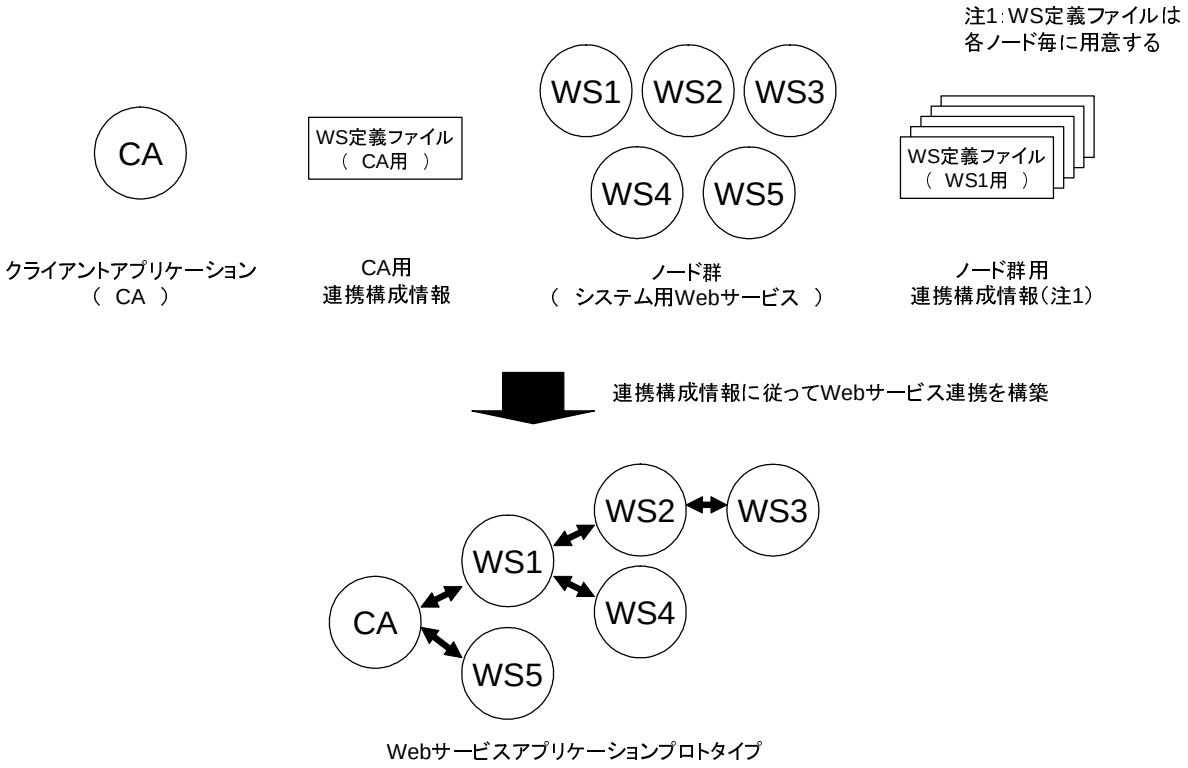

注1：WS定義ファイルは
各ノード毎に用意する
CA
WS定義ファイル
（ CA用 ）
WS1
WS2
WS3
WS4
WS5
WS定義ファイル
（ WS1用 ）
クライアントアプリケーション
（ CA ）
CA用
連携構成情報
ノード群
（ システム用Webサービス ）
ノード群用
連携構成情報（注1）
連携構成情報に従ってWebサービス連携を構築
WS2
WS3
WS1
CA
WS4
WS5
Webサービスアプリケーションプロトタイプ

図 12 WS-PROVE

5.2.1 評価実験用プロトタイプと評価実験方法

Web サービスの連携方式については，大きく分けて 3 つの場合が考えられる [8]．一つ目は，プロキシ型と呼ばれるもので，図 13 のように Web サービスアプリケーションを構成する Web サービスが直列な連携方式をとっているものである．二つ目は，リダイレクト型と呼ばれるもので，図 14 のように Web サービスが並列な連携方式をとっているものである．三つ目は，図 15 のように Web サービスが直列・並行の混合な連携方式をとっているものである．一般に，Amazon Web Services[1] や Google Web APIs[5] といった外部公開されている Web サービスを利用して Web サービスアプリケーションを作成する場合にはリダイレクト型での連携構成となる．また，Web サービスを利用して新たな Web サービスを構築する場合には，プロキシ型もしくは混合型の形となる．

図 13 プロキシ型での連携方式

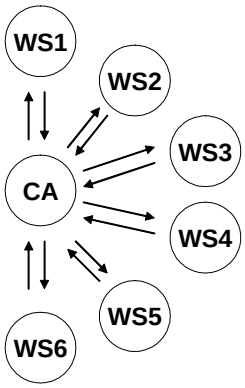


図 14 リダイレクト型での連携方式

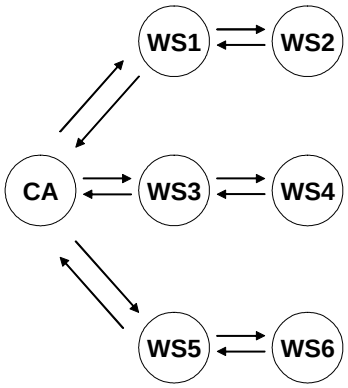


図 15 混合型での連携方式

評価実験用プロトタイプ

本論文では，評価実験用のプロトタイプを，様々な条件で提案メトリクスの値が特徴的に現れるように構築する．具体的には，CalcWS という Web サービスのプロトタイプを，Web サービスの連携方式・Web サービスの数・Web サービスの処理時間・Web サービス間でのネットワーク遅延の 4 つの項目を変化させプロトタイプを構築する．CalcWS という Web サービスは，二つの数字を CalcWS に渡すとそれらを加算してくれる Web サービスである．今回の評価実験においては，CalcWS という Web サービスと同じアルゴリズムの CalcWS01，CalcWS02，. . .，CalcWS09 の 9 個の Web サービスを作成し，WS-PROVE によって Web サービスアプリケーションのプロトタイプを構築し評価実験を行う．以降，評価実験では図 16 に示す 8 つの Web サービス連携方式のうち，いずれかの連携方式を用いてプロトタイプを構築し実験を行う．また，評価実験において各用語の意味は表 5 に示す通りである．

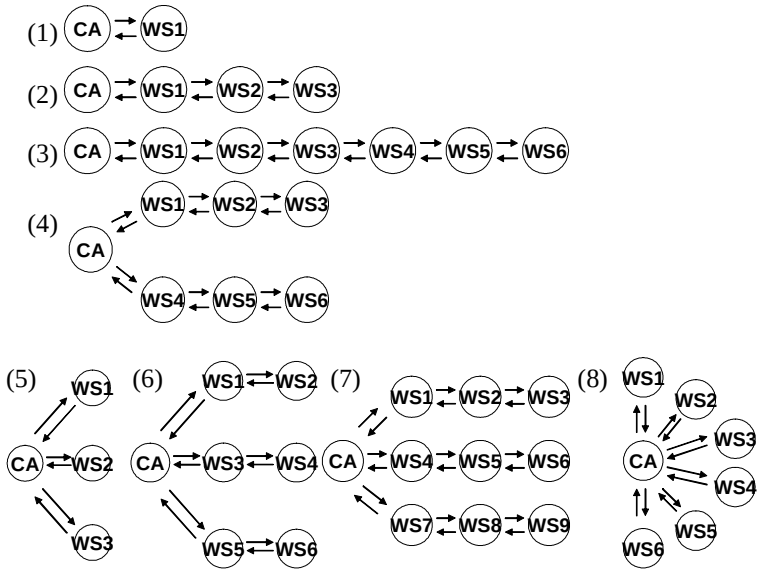


図 16 評価実験用 Web サービス連携方式

表 5 評価実験における用語の意味

用語	意味
CA	クライアントアプリケーション
WS	Web サービス
WS_X	X 番目の Web サービス，CalcWS0X
連携 WS	ノードが呼び出す次の Web サービス
ノード	CA もしくは WS_X
リンク	ノード間のネットワーク
連携方式	Web サービスアプリケーションの Web サービスの利用の形式
トポロジ	意味は連携方式に同じ，プロトタイプの識別で使う
処理時間	ノード単体の処理にかかる固有の時間
動作時間	ノードが処理を終えるまでの実際の計測時間
遅延	リンクでの遅延時間
待ち時間	連携 WS の処理を待つ時間

評価実験 1:連携方式の違いによる効率性評価

評価実験 1 では，Web サービスの基本的な 3 つの連携方式であるプロキシ型・リダイレクト型・混合型の連携方式として図 17 に示す 3 つのプロトタイプを以下の 4 つの条件で構築し計測を行う．そして，計測値と Web サービスメトリクスとの関係についての評価を行う．

1. 各 Web サービスの処理時間は等しく，ネットワークでの遅延は無し (表 6)
2. 各 Web サービスの処理時間は等しく，ネットワークでの遅延を考慮 (表 7)
3. 各 Web サービスの処理時間は異なり，ネットワークでの遅延は無し (表 8)
4. 各 Web サービスの処理時間が異なり，ネットワークでの遅延を考慮 (表 9)

ただし，表 6〜表 9 において $CA(= WS_0), WS_1, WS_2, WS_3, WS_4, WS_5, WS_6$ は，図 17 中の CA および個々の Web サービスに対応し，WS_i^{Time} は WS_i の処理時間 (単位:msec) を表し，$WS_i - WS_k$ は WS_i と WS_k との間の遅延 (単位:msec) とする．また，WS-PROVE での動作時間の計測は 100 回行い，その平均を動作時間の計測結果とする．

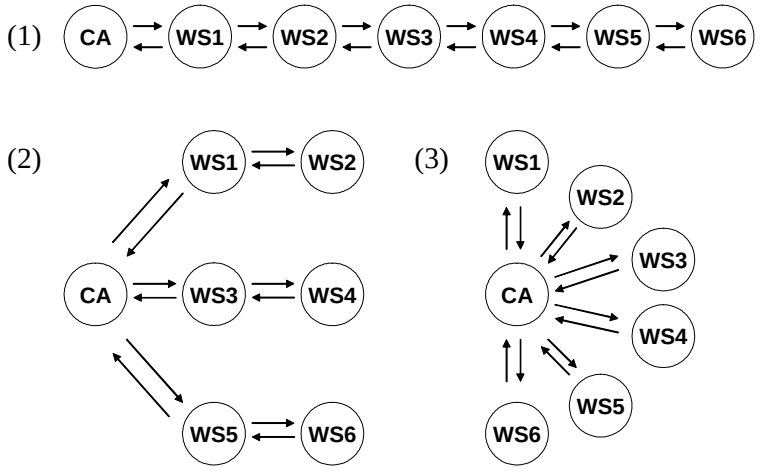


図 17 評価実験 1 の連携方式

1. 各 Web サービスの処理時間は等しく，ネットワークでの遅延は無し

表 6 評価実験 1(条件 1)

	WS_0	WS_1	WS_2	WS_3	WS_4	WS_5	WS_6
WS_i^{Time}	100.0	100.0	100.0	100.0	100.0	100.0	100.0
$WS_0 - WS_i$	-	0.0	0.0	0.0	0.0	0.0	0.0
$WS_1 - WS_i$	0.0	-	0.0	0.0	0.0	0.0	0.0
$WS_2 - WS_i$	0.0	0.0	-	0.0	0.0	0.0	0.0
$WS_3 - WS_i$	0.0	0.0	0.0	-	0.0	0.0	0.0
$WS_4 - WS_i$	0.0	0.0	0.0	0.0	-	0.0	0.0
$WS_5 - WS_i$	0.0	0.0	0.0	0.0	0.0	-	0.0
$WS_6 - WS_i$	0.0	0.0	0.0	0.0	0.0	0.0	-

設定項目: 連携方式，処理時間，遅延 (=0.0msec)

計測項目: 動作時間，待ち時間

目的: 遅延を考慮せず，各 Web サービスの処理時間が等しい条件で実験を行い，連携方式による性能の違いを計測

説明: 評価実験 1(条件 1) においては，表 6 に示す各値を各プロトタイプの対応する個々の Web サービスの WS 定義ファイルに設定し実験を行う．条件 1 では実験の対象となるプロトタイプの各 Web サービスの処理時間を等しく設定し，100.0msec とした．また，CA と各 Web サービスおよび Web サービスと Web サービスとの間のデータのやり取りにおいてネットワークでの遅延を考慮しないために $WS_i - WS_k$ の各値を 0.0msec とした．

2. 各 Web サービスの処理時間は等しく，ネットワークでの遅延を考慮

表 7 評価実験 1(条件 2)

	WS_0	WS_1	WS_2	WS_3	WS_4	WS_5	WS_6
WS_i^{Time}	100.0	100.0	100.0	100.0	100.0	100.0	100.0
$WS_0 - WS_i$	-	50.0	100.0	150.0	200.0	250.0	300.0
$WS_1 - WS_i$	50.0	-	50.0	100.0	150.0	200.0	250.0
$WS_2 - WS_i$	100.0	50.0	-	50.0	100.0	150.0	200.0
$WS_3 - WS_i$	150.0	100.0	50.0	-	50.0	100.0	150.0
$WS_4 - WS_i$	200.0	150.0	100.0	50.0	-	50.0	100.0
$WS_5 - WS_i$	250.0	200.0	150.0	100.0	50.0	-	50.0
$WS_6 - WS_i$	300.0	250.0	200.0	150.0	100.0	50.0	-

設定項目: 連携方式，処理時間，遅延

計測項目: 動作時間，待ち時間，実際の遅延

目的: 遅延を考慮し，各 Web サービスの処理時間が等しい条件で実験を行い，遅延の影響がある場合の連携方式の性能の違いを計測

説明: 評価実験 1(条件 2) においては，表 7 に示す各値を各プロトタイプの対応する個々の Web サービスの WS 定義ファイルに設定し実験を行う．条件 2 では条件 1 と同じく実験の対象となるプロトタイプの各 Web サービスの処理時間を等しく設定し，100.0msec とした．しかし条件 1 とは異なり，ネットワークでの遅延を考慮し，CA と各 Web サービスおよび Web サービスと Web サービスとの間のデータのやり取りにかかる時間 (ネットワーク遅延，$WS_i - WS_k$) を表 7 に示すとおりに設定した．

3. 各 Web サービスの処理時間は異なり，ネットワークでの遅延は無し

表 8 評価実験 1(条件 3)

	WS_0	WS_1	WS_2	WS_3	WS_4	WS_5	WS_6
WS_i^{Time}	100.0	200.0	300.0	400.0	500.0	600.0	700.0
$WS_0 - WS_i$	-	0.0	0.0	0.0	0.0	0.0	0.0
$WS_1 - WS_i$	0.0	-	0.0	0.0	0.0	0.0	0.0
$WS_2 - WS_i$	0.0	0.0	-	0.0	0.0	0.0	0.0
$WS_3 - WS_i$	0.0	0.0	0.0	-	0.0	0.0	0.0
$WS_4 - WS_i$	0.0	0.0	0.0	0.0	-	0.0	0.0
$WS_5 - WS_i$	0.0	0.0	0.0	0.0	0.0	-	0.0
$WS_6 - WS_i$	0.0	0.0	0.0	0.0	0.0	0.0	-

設定項目: 連携方式，処理時間，遅延 (=0.0msec)

計測項目: 動作時間，待ち時間

目的: 遅延を考慮せず，各 Web サービスの処理時間が異なる条件で実験を行い，処理時間が異なる場合の連携方式の性能の違いを計測

説明: 評価実験 1(条件 3) においては，表 8 に示す各値を各プロトタイプの対応する個々の Web サービスの WS 定義ファイルに設定し実験を行う．条件 3 では実験の対象となるプロトタイプにおいて，CA および各 Web サービスの処理時間が異なるように，表 8 が示すとおりに設定した．また，CA と各 Web サービスおよび Web サービスと Web サービスとの間のデータのやり取りにおいてネットワークでの遅延を考慮しないために $WS_i - WS_k$ の各値を 0.0msec とした．

4. 各 Web サービスの処理時間が異なり，ネットワークでの遅延を考慮

表 9 評価実験 1(条件 4)

	WS_0	WS_1	WS_2	WS_3	WS_4	WS_5	WS_6
WS_i^{Time}	100.0	200.0	300.0	400.0	500.0	600.0	700.0
$WS_0 - WS_i$	-	50.0	100.0	150.0	200.0	250.0	300.0
$WS_1 - WS_i$	50.0	-	50.0	100.0	150.0	200.0	250.0
$WS_2 - WS_i$	100.0	50.0	-	50.0	100.0	150.0	200.0
$WS_3 - WS_i$	150.0	100.0	50.0	-	50.0	100.0	150.0
$WS_4 - WS_i$	200.0	150.0	100.0	50.0	-	50.0	100.0
$WS_5 - WS_i$	250.0	200.0	150.0	100.0	50.0	-	50.0
$WS_6 - WS_i$	300.0	250.0	200.0	150.0	100.0	50.0	-

設定項目: 連携方式，処理時間，遅延

計測項目: 動作時間，待ち時間，実際の遅延

目的: 遅延を考慮し，各 Web サービスの処理時間が異なる条件で実験を行い，現実的な状況での連携方式の性能の違いを計測

説明: 評価実験 1(条件 4) においては，表 9 に示す各値を各プロトタイプの対応する個々の Web サービスの WS 定義ファイルに設定し実験を行う．条件 4 では条件 3 と同じく実験の対象となるプロトタイプにおいて，CA および各 Web サービス処理時間が異なるように，表 9 が示すとおりに設定した．また，条件 2 と同じくネットワークでの遅延を考慮し，CA と各 Web サービスおよび Web サービスと Web サービスとの間のデータのやり取りにかかる時間 (ネットワーク遅延，$WS_i - WS_k$) を表 9 に示すとおりに設定した．

評価実験 2:Web サービス利用数の違いによる効率性評価

評価実験 2 では，図 18 における連携方式 (1)〜(8) のプロトタイプを構築する．各プロトタイプにおいては，各トポロジ間で利用している Web サービスの処理時間の総和が等しくなるように個々の Web サービスの処理時間を WS 定義ファイルに設定し ((1)〜(8) の $\sum WS_i^{Time}$ を等しくする)，各 Web サービスの動作時間およびシステムでの動作時間の計測を行い，Web サービスメトリクスとの関係についての評価を行う．

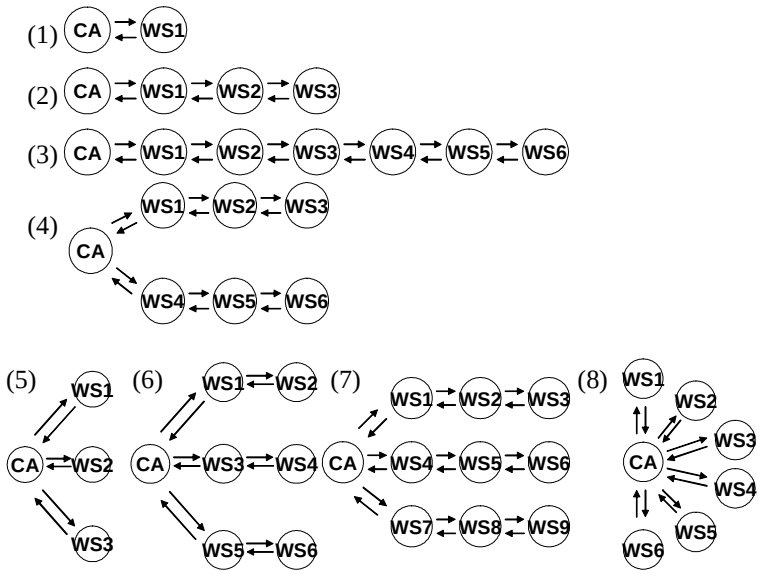


図 18　評価実験 2 用プロトタイプ

図 18 に示す各プロトタイプにたいして表 10 に示す設定値で評価実験を行う．ただし，$CA(= WS_0), WS_1, \ldots, WS_9$ は，図 18 中の CA および個々の Web サービスに対応し，$WS_i^{Time}(N)$ はプロトタイプ N(図 18 中の (1)〜(8) に対応) における WS_i の処理時間を表し，$WS_i - WS_k$ は WS_i と WS_k との間の遅延とする．また，WS-PROVE での動作時間の計測は 100 回行い，その平均を動作時間の計測結果とする．

表 10 評価実験 2 における設定値

	WS_0	WS_1	WS_2	WS_3	WS_4	WS_5	WS_6	WS_7	WS_8	WS_9
$WS_i^{Time}(1)$	100.0	900.0	-	-	-	-	-	-	-	-
$WS_i^{Time}(2)$	100.0	300.0	300.0	300.0	-	-	-	-	-	-
$WS_i^{Time}(3)$	100.0	150.0	150.0	150.0	150.0	150.0	150.0	-	-	-
$WS_i^{Time}(4)$	100.0	150.0	150.0	150.0	150.0	150.0	150.0	-	-	-
$WS_i^{Time}(5)$	100.0	300.0	300.0	300.0	-	-	-	-	-	-
$WS_i^{Time}(6)$	100.0	150.0	150.0	150.0	150.0	150.0	150.0	-	-	-
$WS_i^{Time}(7)$	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
$WS_i^{Time}(8)$	100.0	150.0	150.0	150.0	150.0	150.0	150.0	-	-	-
$WS_i - WS_k$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

設定項目: 連携方式，処理時間，遅延 (=0.0msec)

計測項目: 動作時間，待ち時間，実際の遅延

目的: 遅延を考慮せず，各プロトタイプでの Web サービスの処理時間の総和が等しい状態で実験を行い Web サービスの利用数によるオーバーヘッドを計測する

説明: 評価実験 2 においては，表 10 に示す各値を各プロトタイプの対応する個々の Web サービスの WS 定義ファイルに設定し実験を行う．実験では，ネットワークでの遅延を考慮しないこととし，$WS_i - WS_k$ の各値を 0.0msec とした．そして，図 18 中の (1)〜(8) の各プロトタイプにおいて，プロトタイプが利用している各 Web サービスの処理時間の総和が 900msec となるように，表 10 に示すとおりに WS_i^{Time} を設定した．

5.2.2 WS-PROVE による評価実験結果

評価実験 1:連携方式の違いによる効率性評価

評価実験 1 についての結果を表 11〜表 14 に示す．以下，評価実験 1 の各条件ごとに結果をまとめる．表 11〜表 14 中において，試行回数とは対象となる連携方式 (図 16 中の番号に一致) で何回試行したかを表し，処理時間とは WS 定義ファイルに設定した Client 及び CalcWSXX の処理時間である．動作時間 (処理を開始してから処理が終わるまでの時間) とはその試行回数におけるプロトタイプ全体 (System) またはクライアントアプリケーション (Client，WS_0) または Web サービス (CalcWSXX，$WS_1, WS_2, \ldots, WS_i$) の動作時間の平均値である．また，遅延とは試行回数における Client または CalcWSXX におけるネットワーク遅延時間の平均値であり，連携 WS とは表中で同じ行に示される Client もしくは CalcWSXX が呼び出す Web サービスであり，待ち時間とは試行回数におけるそれら連携 WS の処理が終わるまでの待ち時間の平均値である．

1. 各 Web サービスの処理時間は等しく，ネットワークでの遅延は無し

   条件 1 における実験の結果を表 11 に示す．条件 1 ではネットワークでの遅延を考慮しないということで実験を行ったので，遅延は 0.0msec である．また，CalcWSXX の処理時間を等しく 100msec と WS 定義ファイルに設定し実験を行った．各プロトタイプにおいて全体の動作時間の合計は 803，812，812msec となっており，顕著な差は見られなかった．しかし，プロトタイプの連携方式の違いにより CalcWSXX の動作時間には顕著な差が見られた．具体的には，(1) の連携方式 3 の形 (プロキシ型) でのプロトタイプでは，個々の CalcWSXX の処理時間は 100msec で等しくても，呼び出す連携 WS の処理が終わるまでは CalcWSXX の処理を終えることができず CalcWSXX の動作時間は処理時間に対して長くなるという結果になった．一方で，(3) の連携方式 8 の形 (リダイレクト型) でのプロトタイプでは，CalcWSXX の動作時間は呼び出す連携 WS が無いため，個々の CalcWSXX の処理時間である 100msec に近い値となる結果となった．

表 11　評価実験 1(条件 1) の結果

評価実験1　条件1　処理時間は等しく、ネットワーク遅延なし						
(1)		動作時間	試行回数	連携方式	−	−
	System	812.42	100	3	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	805.55	CalcWS01	702.16	0.00
	CalcWS01	100.00	692.66	CalcWS02	585.10	0.00
	CalcWS02	100.00	575.67	CalcWS03	467.95	0.00
	CalcWS03	100.00	458.46	CalcWS04	350.84	0.00
	CalcWS04	100.00	341.14	CalcWS05	233.72	0.00
	CalcWS05	100.00	224.31	CalcWS06	116.64	0.00
	CalcWS06	100.00	107.35	−	−	−
(2)		動作時間	試行回数	連携方式	−	−
	System	812.73	100	6	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	805.87	CalcWS01	233.71	0.00
	〃	−	−	CalcWS03	234.14	0.00
	〃	−	−	CalcWS05	234.42	0.00
	CalcWS01	100.00	224.21	CalcWS02	116.83	0.00
	CalcWS02	100.00	107.56	−	−	−
	CalcWS03	100.00	224.62	CalcWS04	116.59	0.00
	CalcWS04	100.00	107.31	−	−	−
	CalcWS05	100.00	224.77	CalcWS06	116.59	0.00
	CalcWS06	100.00	107.31	−	−	−
(3)		動作時間	試行回数	連携方式	−	−
	System	803.26	100	8	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	796.70	CalcWS01	115.94	0.00
	〃	−	−	CalcWS02	112.94	0.00
	〃	−	−	CalcWS03	113.78	0.00
	〃	−	−	CalcWS04	113.28	0.00
	〃	−	−	CalcWS05	114.43	0.00
	〃	−	−	CalcWS06	114.43	0.00
	CalcWS01	100.00	106.65	−	−	−
	CalcWS02	100.00	103.75	−	−	−
	CalcWS03	100.00	104.53	−	−	−
	CalcWS04	100.00	103.97	−	−	−
	CalcWS05	100.00	105.09	−	−	−
	CalcWS06	100.00	105.10	−	−	−

（単位：　msec）

2. 各 Web サービスの処理時間は等しく，ネットワークでの遅延を考慮

   条件 2 における実験の結果を表 12 に示す．条件 2 ではネットワークでの遅延を考慮し，$WS_i - WS_{i+1}$ 間のネットワークの遅延は小さいが，$WS_i - WS_{i+2}$ といったように WS_i と WS_k の k の値が i から離れるほどネットワーク遅延が大きくなるようにネットワーク遅延を設定した．また，CalcWSXX の処理時間は等しく 100msec と WS 定義ファイルに設定し実験を行った．各プロトタイプにおいて全体の動作時間の合計は 1188，1440，1875msec となっており，連携方式の差によって顕著な差が見られた．具体的には，(1) の連携方式 3 の形 (プロキシ型) でのプロトタイプでは最もネットワーク遅延が少なくなるような形で CalcWSXX の連携が行われ，System での動作時間が 1188msec と小さくなったのに対し，(3) の連携方式 8 の形 (リダイレクト型) のプロトタイプでは Client が CalcWSXX を呼び出すのでネットワーク遅延が (1) のプロトタイプより余分にかかり，System での動作時間が 1875msec と大きくなるという結果になった．個々の CalcWSXX の動作時間に関しては条件 1 の時と同様の結果となった．

表 12　評価実験 1(条件 2) の結果

評価実験1　条件2　処理時間は等しく、ネットワーク遅延あり						
(1)		動作時間	試行回数	連携方式	−	−
	System	1,188.10	100	3	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	1,181.22	CalcWS01	1,015.12	62.23
	CalcWS01	100.00	1,005.65	CalcWS02	835.58	62.16
	CalcWS02	100.00	826.05	CalcWS03	655.90	62.21
	CalcWS03	100.00	646.48	CalcWS04	476.27	62.23
	CalcWS04	100.00	466.50	CalcWS05	296.59	62.21
	CalcWS05	100.00	287.07	CalcWS06	117.06	62.26
	CalcWS06	100.00	107.56	−	−	−
(2)		動作時間	試行回数	連携方式	−	−
	System	1,440.31	100	6	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	1,433.49	CalcWS01	296.34	62.03
	〃	−	−	CalcWS03	296.48	142.55
	〃	−	−	CalcWS05	296.50	236.13
	CalcWS01	100.00	286.87	CalcWS02	116.75	62.34
	CalcWS02	100.00	107.47	−	−	−
	CalcWS03	100.00	287.02	CalcWS04	117.03	62.24
	CalcWS04	100.00	107.73	−	−	−
	CalcWS05	100.00	287.00	CalcWS06	116.83	62.26
	CalcWS06	100.00	107.47	−	−	−
(3)		動作時間	試行回数	連携方式	−	−
	System	1,874.83	100	8	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	1,868.20	CalcWS01	117.00	61.70
	〃	−	−	CalcWS02	116.86	101.75
	〃	−	−	CalcWS03	116.88	148.73
	〃	−	−	CalcWS04	117.08	195.56
	〃	−	−	CalcWS05	117.23	242.27
	〃	−	−	CalcWS06	116.99	304.59
	CalcWS01	100.00	107.55	−	−	−
	CalcWS02	100.00	107.55	−	−	−
	CalcWS03	100.00	107.62	−	−	−
	CalcWS04	100.00	107.51	−	−	−
	CalcWS05	100.00	107.53	−	−	−
	CalcWS06	100.00	107.53	−	−	−

（単位：　msec）

3. 各 Web サービスの処理時間は異なり，ネットワークでの遅延は無し

   条件 3 における実験の結果を表 13 に示す．条件 3 ではネットワークでの遅延を考慮しないということで実験を行ったので，遅延は 0.0msec である．しかし，CalcWSXX の処理時間を WS_i の i が大きくなるにつれ，処理時間が 100msec ずつ大きくなるように WS 定義ファイルに設定し実験を行った．各プロトタイプにおいて，System と Client と CalcWSXX について設定した処理時間に比例して動作時間が増加したものの，基本的には条件 1 と似たような結果となった．しかし，条件 1 に比べて，処理時間が大きい CalcWSXk を連携 WS として利用している CalcWSXi は，その利用している CalcWSXk の処理時間に比例して待ち時間が増加し，CalcWSXi そのものの処理時間に対して待ち時間がさらに大きくなるという結果になった．

表 13　評価実験 1(条件 3) の結果

評価実験1　条件3　処理時間は異なり、ネットワーク遅延なし						
(1)		動作時間	試行回数	連携方式	−	−
	System	2,891.61	100	3	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	2,884.73	CalcWS01	2,780.78	0.00
	CalcWS01	200.00	2,771.18	CalcWS02	2,569.93	0.00
	CalcWS02	300.00	2,560.32	CalcWS03	2,249.62	0.00
	CalcWS03	400.00	2,240.13	CalcWS04	1,835.66	0.00
	CalcWS04	500.00	1,826.11	CalcWS05	1,327.66	0.00
	CalcWS05	600.00	1,318.22	CalcWS06	710.55	0.00
	CalcWS06	700.00	701.09	−	−	−
(2)		動作時間	試行回数	連携方式	−	−
	System	2,890.89	100	6	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	2,884.18	CalcWS01	530.36	0.00
	〃	−	−	CalcWS03	922.14	0.00
	〃	−	−	CalcWS05	1,328.10	0.00
	CalcWS01	200.00	520.88	CalcWS02	319.77	0.00
	CalcWS02	300.00	310.42	−	−	−
	CalcWS03	400.00	912.41	CalcWS04	507.28	0.00
	CalcWS04	500.00	497.95	−	−	−
	CalcWS05	600.00	1,318.43	CalcWS06	710.41	0.00
	CalcWS06	700.00	701.11	−	−	−
(3)		動作時間	試行回数	連携方式	−	−
	System	2,879.38	100	8	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	2,872.81	CalcWS01	209.77	0.00
	〃	−	−	CalcWS02	316.29	0.00
	〃	−	−	CalcWS03	409.19	0.00
	〃	−	−	CalcWS04	504.00	0.00
	〃	−	−	CalcWS05	613.89	0.00
	〃	−	−	CalcWS06	707.87	0.00
	CalcWS01	200.00	200.45	−	−	−
	CalcWS02	300.00	307.02	−	−	−
	CalcWS03	400.00	400.00	−	−	−
	CalcWS04	500.00	494.62	−	−	−
	CalcWS05	600.00	604.57	−	−	−
	CalcWS06	700.00	698.56	−	−	−

（単位：　msec）

4. 各 Web サービスの処理時間が異なり，ネットワークでの遅延を考慮

   条件 4 における実験の結果を表 14 に示す．条件 4 では条件 2 と同様にネットワークでの遅延を考慮し，$WS_i - WS_{i+1}$ 間のネットワークの遅延は小さいが，$WS_i - WS_{i+2}$ といったように WS_i と WS_k の k の値が i から離れるほどネットワーク遅延が大きくなるようにネットワーク遅延を設定した．また，条件 3 と同様に各 CalcWSXX の処理時間を WS_i の i が大きくなるにつれ，処理時間が 100msec ずつ大きくなるように WS 定義ファイルに設定し実験を行った．結果としては，条件 2 と条件 3 の結果を組み合わせたような結果となり，各プロトタイプにおいて，(1) の連携方式 3 であるプロキシ型ではネットワーク遅延の合計が小さくなり System としての動作時間が一番小さい結果 (3266msec) となるのに対し，(3) の連携方式 8 であるリダイレクト型ではネットワーク遅延の合計が大きくなり System としての動作時間が大きくなるという結果 (3952msec) になった．しかし，個々の CalcWSXX の動作時間ということになると (1) のプロキシ型では最初に利用される CalcWSXi ほど後の CalcWSXk の処理を待つ時間が多くなり，処理時間に対して待ち時間の割合がとても大きくなる (例:CalcWS01 の処理時間の 200msec に対し待ち時間は 2820msec となる) のに対し，(3) のリダイレクト型では個々の処理時間に近い値で処理を終えることができているという結果になった．

表 14　評価実験 1(条件 4) の結果

評価実験1　条件4　処理時間は異なり、ネットワーク遅延あり						
(1)		動作時間	試行回数	連携方式	−	−
	System	3,266.18	100	3	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	3,259.40	CalcWS01	3,093.51	62.23
	CalcWS01	200.00	3,083.96	CalcWS02	2,820.00	62.24
	CalcWS02	300.00	2,810.42	CalcWS03	2,437.19	62.19
	CalcWS03	400.00	2,427.75	CalcWS04	1,960.81	62.21
	CalcWS04	500.00	1,951.10	CalcWS05	1,390.53	62.26
	CalcWS05	600.00	1,380.93	CalcWS06	710.84	62.27
	CalcWS06	700.00	701.34	−	−	−
(2)		動作時間	試行回数	連携方式	−	−
	System	3,521.07	100	6	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	3,514.26	CalcWS01	593.36	61.99
	〃	−	−	CalcWS03	984.01	143.50
	〃	−	−	CalcWS05	1,390.25	237.36
	CalcWS01	200.00	583.83	CalcWS02	320.13	62.16
	CalcWS02	300.00	310.75	−	−	−
	CalcWS03	400.00	974.40	CalcWS04	507.58	62.29
	CalcWS04	500.00	498.24	−	−	−
	CalcWS05	600.00	1,380.78	CalcWS06	710.76	62.20
	CalcWS06	700.00	701.45	−	−	−
(3)		動作時間	試行回数	連携方式	−	−
	System	3,952.93	100	8	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延
	Client	100.00	3,946.32	CalcWS01	210.78	61.69
	〃	−	−	CalcWS02	320.03	101.72
	〃	−	−	CalcWS03	413.76	148.68
	〃	−	−	CalcWS04	507.74	195.55
	〃	−	−	CalcWS05	617.05	242.40
	〃	−	−	CalcWS06	710.73	304.78
	CalcWS01	200.00	201.37	−	−	−
	CalcWS02	300.00	310.62	−	−	−
	CalcWS03	400.00	404.46	−	−	−
	CalcWS04	500.00	498.20	−	−	−
	CalcWS05	600.00	607.61	−	−	−
	CalcWS06	700.00	701.25	−	−	−

（単位：　msec）

評価実験 2:Web サービス利用数の違いによる効率性評価

評価実験 2 の結果を表 15 に示す．評価実験 2 では，ネットワーク遅延を考慮せず，各プロトタイプ間で個々の Web サービスの処理時間の総和が等しくなるような設定で実験を行った．その結果，各プロトタイプ間での動作時間は，評価実験 1 と同様にプロキシ型での連携 WS の利用の場合に，連携 WS もしくは CA の動作時間は増加する結果となった ((2)(3))．しかし，ネットワークでの遅延を考慮しなかったため，リダイレクト型での連携 WS の利用において動作時間に差は見られず，それら連携 WS の処理待ち時間はその連携 WS の処理時間+オーバーヘッドとなる結果になった (トポロジ (5)(8))．また，Web サービスの利用数が異なると System での動作時間が異なる結果になった ((1)，(2)(5)，(3)(4)(6)(8)，(7))．これは，Web サービスの利用数が増えると，その WS を利用するために標準化された手続きを行わなければならず，オーバーヘッドが増加したためだと思われる．

表 15 評価実験 2 の結果

(1)		動作時間	試行回数	連携方式	−
	System	1,031.03	100	1	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,024.39	CalcWS01	913.54
	CalcWS01	900.00	904.18	−	−
(2)		動作時間	試行回数	連携方式	−
	System	1,077.90	100	2	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,071.21	CalcWS01	960.36
	CalcWS01	300.00	950.69	CalcWS02	640.00
	CalcWS02	300.00	630.64	CalcWS03	319.85
	CalcWS03	300.00	310.44	−	−
(3)		動作時間	試行回数	連携方式	−
	System	1,093.98	100	3	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,087.17	CalcWS01	983.55
	CalcWS01	150.00	974.00	CalcWS02	819.51
	CalcWS02	150.00	809.86	CalcWS03	655.51
	CalcWS03	150.00	646.15	CalcWS04	491.64
	CalcWS04	150.00	482.09	CalcWS05	327.50
	CalcWS05	150.00	318.10	CalcWS06	163.53
	CalcWS06	150.00	154.17	−	−
(4)		動作時間	試行回数	連携方式	−
	System	1,096.83	100	4	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,090.18	CalcWS01	491.47
	〃	−	−	CalcWS04	487.78
	CalcWS01	150.00	481.92	CalcWS02	327.43
	CalcWS02	150.00	318.07	CalcWS03	163.57
	CalcWS03	150.00	154.14	−	−
	CalcWS04	150.00	478.11	CalcWS05	327.46
	CalcWS05	150.00	318.09	CalcWS06	163.58
	CalcWS06	150.00	154.18	−	−
(5)		動作時間	試行回数	連携方式	−
	System	1,070.45	100	5	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,063.83	CalcWS01	319.69
	〃	−	−	CalcWS02	316.89
	〃	−	−	CalcWS03	316.25
	CalcWS01	300.00	310.31	−	−
	CalcWS02	300.00	307.59	−	−
	CalcWS03	300.00	306.81	−	−

(6)		動作時間	試行回数	連携方式	−
	System	1,093.70	100	6	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,086.96	CalcWS01	327.29
	〃	−	−	CalcWS03	328.11
	〃	−	−	CalcWS05	328.14
	CalcWS01	150.00	317.84	CalcWS02	163.59
	CalcWS02	150.00	154.26	−	−
	CalcWS03	150.00	318.60	CalcWS04	163.68
	CalcWS04	150.00	154.21	−	−
	CalcWS05	150.00	318.49	CalcWS06	163.70
	CalcWS06	150.00	154.17	−	−
(7)		動作時間	試行回数	連携方式	−
	System	1,165.65	100	7	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,159.00	CalcWS01	350.65
	〃	−	−	CalcWS04	347.97
	〃	−	−	CalcWS07	349.13
	CalcWS01	100.00	341.08	CalcWS02	233.70
	CalcWS02	100.00	224.14	CalcWS03	116.68
	CalcWS03	100.00	107.36	−	−
	CalcWS04	100.00	338.43	CalcWS05	233.87
	CalcWS05	100.00	224.45	CalcWS06	116.77
	CalcWS06	100.00	107.31	−	−
	CalcWS07	100.00	339.53	CalcWS08	234.38
	CalcWS08	100.00	224.84	CalcWS09	117.20
	CalcWS09	100.00	107.35	−	−
(8)		動作時間	試行回数	連携方式	−
	System	1,089.05	100	8	−
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,082.39	CalcWS01	163.01
	〃	−	−	CalcWS02	160.97
	〃	−	−	CalcWS03	161.14
	〃	−	−	CalcWS04	161.30
	〃	−	−	CalcWS05	162.26
	〃	−	−	CalcWS06	162.24
	CalcWS01	150.00	153.58	−	−
	CalcWS02	150.00	151.66	−	−
	CalcWS03	150.00	151.68	−	−
	CalcWS04	150.00	151.76	−	−
	CalcWS05	150.00	152.70	−	−
	CalcWS06	150.00	152.79	−	−

（単位： msec）

5.2.3 Web サービスメトリクスと効率性についての考察

Web サービスメトリクスと WS-PROVE による各プロトタイプの性能との関係について考察を行う (表 16)．考察においては，Web サービスの連携における効率性を評価するために，評価実験 1(条件 2) の結果と評価実験 2 の結果を用いて考察を行う．考察を行うにあたって，評価実験の結果と各プロトタイプに対して Web サービスメトリクスを適用した結果を表 17，表 18 に示す．

NOWS メトリクスについて

NOWS メトリクスについては，表 18 より Web サービスアプリケーション全体のオーバーヘッドの量と関連が見られる結果となった．NOWS メトリクスが 1 の場合には System の動作時間が 1031msec なのに対し，NOWS メトリクスが 3，6，9 の場合にはそれぞれ 1070msec，1090msec，1165msec に近い値になっている．よって，NOWS メトリクスが大きければ System の動作時間が大きくなる傾向があるということが言える．一方で，表 17 より NOWS が等しくても，ネットワークの遅延と連携構成によっては，System の動作時間が大きく変化してしまうことが示された．従って，NOWS メトリクスは Web サービスアプリケーション全体のオーバーヘッドの量の評価には適するが，動作時間の評価には NOWS メトリクスは適さないと言える．

RFWS メトリクスについて

RFWS メトリクスについては，CA もしくは Web サービスが利用する連携 WS の処理を待つ時間 (自身の処理時間以外に時間を割かれるか) の数に関して関連がみられる結果となった．表 17 の (2) の RFWS メトリクスが 2 である CalcWS01，03，05 では処理時間が 100msec，動作時間が 287msec となっているのに対し，RFWS メトリクスが 0 である CalcWS02，04，06 では処理時間が 100msec，動作時間が 107msec になっている．従って，RFWS メトリクスが大きい WS は処理時間に対して動作時間が大きくなる傾向があると言える．しかし，表 17 の (1) の様なプロキシ型の連携 WS の利用では RFWS での動作時間の評価は適さない結果となった．これは，RFWS メト

リクスでは連携 WS の処理を待つ時間の数を評価すると考えられるが，それら連携 WS 単体の処理待ち時間の量の評価には適さないためと考えられる．

NHTWS メトリクスについて

NHTWS メトリクスについては，CA もしくは WS が利用する連携 WS の処理を待つ時間 (自身の処理時間以外に時間を割かれるか) の量に関して，関連がみられる結果となった．表 17 の (1) の CalcWSXX では，NHTWS メトリクスが 1 の CalcWS05 は処理待ち時間が 117msec であり，NHTWS メトリクスが 2 の CalcWS04 では処理待ち時間が 296msec となっている．以降 NHTWS メトリクスが大きくなるにつれ処理待ち時間は増加し，NHTWS メトリクスが 5 の CalcWS01 では処理待ち時間が 835msec となっている．従って，NHTWS メトリクスが大きい WS は処理時間に対して処理待ち時間が大きくなり動作時間も大きくなる傾向があると言える．しかし，RFWS メトリクスとは逆に表 17 の (3) のようなリダイレクト型の連携 WS の利用では NHTWS メトリクスでの動作時間の評価は適さない結果となった．これは，NHTWS メトリクスでは Web サービスが利用する連携 WS 単体の処理待ち時間の量を評価すると考えられるが，連携 WS の処理を待つ時間の数の評価には適さないためと考えられる．

表 16 提案メトリクスと効率性との関係

提案メトリクス	評価項目	効率性との関係
NOWS	全体でのオーバーヘッド量	値が大きいと効率性が悪くなる
RFWS	連携 WS の処理を待つ時間の数	値が大きいと効率性が悪くなる
NHTWS	連携 WS の処理を待つ時間の量	値が大きいと効率性が悪くなる

表 17 評価実験 1(条件 2) による Web サービスメトリクスの評価

(1)		動作時間	試行回数	連携方式	NOWS	−	−	−
	System	1,188.10	100	3	6	−	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延	RFWS	NHTWS
	Client	100.00	1,181.22	CalcWS01	1,015.12	62.23	2	6
	CalcWS01	100.00	1,005.65	CalcWS02	835.58	62.16	2	5
	CalcWS02	100.00	826.05	CalcWS03	655.90	62.21	2	4
	CalcWS03	100.00	646.48	CalcWS04	476.27	62.23	2	3
	CalcWS04	100.00	466.50	CalcWS05	296.59	62.21	2	2
	CalcWS05	100.00	287.07	CalcWS06	117.06	62.26	2	1
	CalcWS06	100.00	107.56	−	−	−	0	0
(2)		動作時間	試行回数	連携方式	NOWS	−	−	−
	System	1,440.31	100	6	6	−	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延	RFWS	NHTWS
	Client	100.00	1,433.49	CalcWS01	296.34	62.03	6	2
	〃	−	−	CalcWS03	296.48	142.55		
	〃	−	−	CalcWS05	296.50	236.13		
	CalcWS01	100.00	286.87	CalcWS02	116.75	62.34	2	1
	CalcWS02	100.00	107.47	−	−	−	0	0
	CalcWS03	100.00	287.02	CalcWS04	117.03	62.24	2	1
	CalcWS04	100.00	107.73	−	−	−	0	0
	CalcWS05	100.00	287.00	CalcWS06	116.83	62.26	2	1
	CalcWS06	100.00	107.47	−	−	−	0	0
(3)		動作時間	試行回数	連携方式	NOWS	−	−	−
	System	1,874.83	100	8	6	−	−	−
		処理時間	動作時間	連携WS	待ち時間	遅延	RFWS	NHTWS
	Client	100.00	1,868.20	CalcWS01	117.00	61.70	12	1
	〃	−	−	CalcWS02	116.86	101.75		
	〃	−	−	CalcWS03	116.88	148.73		
	〃	−	−	CalcWS04	117.08	195.56		
	〃	−	−	CalcWS05	117.23	242.27		
	〃	−	−	CalcWS06	116.99	304.59		
	CalcWS01	100.00	107.55	−	−	−	0	0
	CalcWS02	100.00	107.55	−	−	−	0	0
	CalcWS03	100.00	107.62	−	−	−	0	0
	CalcWS04	100.00	107.51	−	−	−	0	0
	CalcWS05	100.00	107.53	−	−	−	0	0
	CalcWS06	100.00	107.53	−	−	−	0	0

(単位:msec)

表 18 評価実験 2 による Web サービスメトリクスの評価

		動作時間	試行回数	連携方式	NOWS
(1)	System	1,031.03	100	1	1
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,024.39	CalcWS01	913.54
	CalcWS01	900.00	904.18	−	−

		動作時間	試行回数	連携方式	NOWS
(2)	System	1,077.90	100	2	3
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,071.21	CalcWS01	960.36
	CalcWS01	300.00	950.69	CalcWS02	640.00
	CalcWS02	300.00	630.64	CalcWS03	319.85
	CalcWS03	300.00	310.44	−	−

		動作時間	試行回数	連携方式	NOWS
(3)	System	1,093.98	100	3	6
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,087.17	CalcWS01	983.55
	CalcWS01	150.00	974.00	CalcWS02	819.51
	CalcWS02	150.00	809.86	CalcWS03	655.51
	CalcWS03	150.00	646.15	CalcWS04	491.64
	CalcWS04	150.00	482.09	CalcWS05	327.50
	CalcWS05	150.00	318.10	CalcWS06	163.53
	CalcWS06	150.00	154.17	−	−

		動作時間	試行回数	連携方式	NOWS
(4)	System	1,096.83	100	4	6
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,090.18	CalcWS01	491.47
	〃	−	−	CalcWS04	487.78
	CalcWS01	150.00	481.92	CalcWS02	327.43
	CalcWS02	150.00	318.07	CalcWS03	163.57
	CalcWS03	150.00	154.14	−	−
	CalcWS04	150.00	478.11	CalcWS05	327.46
	CalcWS05	150.00	318.09	CalcWS06	163.58
	CalcWS06	150.00	154.18	−	−

		動作時間	試行回数	連携方式	NOWS
(5)	System	1,070.45	100	5	3
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,063.83	CalcWS01	319.69
	〃	−	−	CalcWS02	316.89
	〃	−	−	CalcWS03	316.25
	CalcWS01	300.00	310.31	−	−
	CalcWS02	300.00	307.59	−	−
	CalcWS03	300.00	306.81	−	−

		動作時間	試行回数	連携方式	NOWS
(6)	System	1,093.70	100	6	6
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,086.96	CalcWS01	327.29
	〃	−	−	CalcWS03	328.11
	〃	−	−	CalcWS05	328.14
	CalcWS01	150.00	317.84	CalcWS02	163.59
	CalcWS02	150.00	154.26	−	−
	CalcWS03	150.00	318.60	CalcWS04	163.68
	CalcWS04	150.00	154.21	−	−
	CalcWS05	150.00	318.49	CalcWS06	163.70
	CalcWS06	150.00	154.17	−	−

		動作時間	試行回数	連携方式	NOWS
(7)	System	1,165.65	100	7	9
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,159.00	CalcWS01	350.65
	〃	−	−	CalcWS04	347.97
	〃	−	−	CalcWS07	349.13
	CalcWS01	100.00	341.08	CalcWS02	233.70
	CalcWS02	100.00	224.14	CalcWS03	116.68
	CalcWS03	100.00	107.36	−	−
	CalcWS04	100.00	338.43	CalcWS05	233.87
	CalcWS05	100.00	224.45	CalcWS06	116.77
	CalcWS06	100.00	107.31	−	−
	CalcWS07	100.00	339.53	CalcWS08	234.38
	CalcWS08	100.00	224.84	CalcWS09	117.20
	CalcWS09	100.00	107.35	−	−

		動作時間	試行回数	連携方式	NOWS
(8)	System	1,089.05	100	8	6
		処理時間	動作時間	連携WS	待ち時間
	Client	100.00	1,082.39	CalcWS01	163.01
	〃	−	−	CalcWS02	160.97
	〃	−	−	CalcWS03	161.14
	〃	−	−	CalcWS04	161.30
	〃	−	−	CalcWS05	162.26
	〃	−	−	CalcWS06	162.24
	CalcWS01	150.00	153.58	−	−
	CalcWS02	150.00	151.66	−	−
	CalcWS03	150.00	151.68	−	−
	CalcWS04	150.00	151.76	−	−
	CalcWS05	150.00	152.70	−	−
	CalcWS06	150.00	152.79	−	−

（単位： msec）

5.3 SDP アルゴリズムを用いた信頼性の評価

本節では，Sum of Disjoint Products(SDP)[6, 14, 16] という信頼性評価アルゴリズムを用いた信頼性の評価方法について述べる．SDP アルゴリズムは，グラフの各ノードと各辺にそれぞれ信頼性を与えると，グラフ内の特定の部分グラフ (の集合) が稼動する確率を，ノードや辺の重なりを考慮して導出できる．本論文では Web サービスアプリケーションのトポロジを論理的なネットワークとみなしグラフで表現することにより，RFWS，NOWS，NHTWS について SDP による信頼性の評価実験を行う．

5.3.1 SDP アルゴリズムを用いた評価実験方法

SDP アルゴリズムに対して，トポロジにおいて CA もしくは WS 単体 (ノード) が正常に動作する確率 (ノード信頼性:NR) とノード間のネットワーク (リンク) が正常に動作する確率 (リンク信頼性:LR) を与えると，トポロジ内の特定のノード群 (あるノード A が他のノード $N_i(i = 0, 1, 2, \ldots)$ を利用する場合，ノード A とノード N_i とそれらの間でのリンク) がサービス連携して正常に動作する確率 (サービス信頼性:SR) を導出できる．本論文では， 図 19 に示される各トポロジを対象に，トポロジを構成するノードのノード信頼性は一律に 1.00 とし，ノード間のリンク信頼性は 0.99 として SDP を適用し，評価実験を行った．

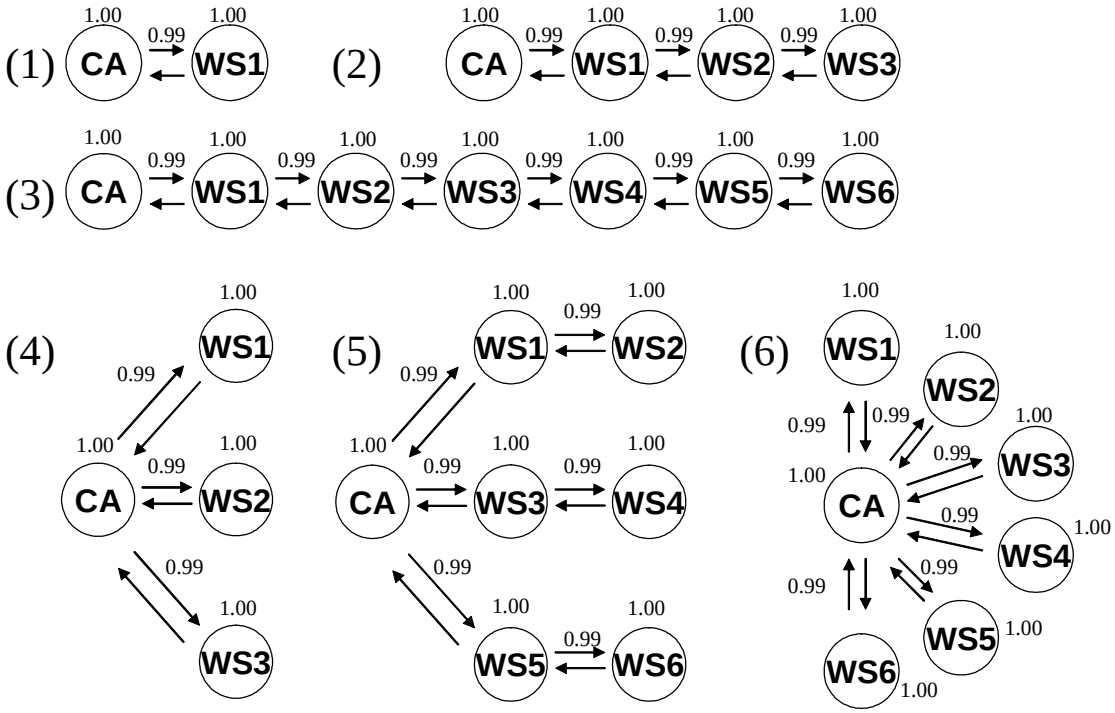


図 19 SDP アルゴリズムによる信頼性評価実験

5.3.2 SDP アルゴリズムを用いた評価実験結果

SDP による評価実験結果を表 19 に示す．表中に各トポロジに対する Web サービスメトリクス (RFWS，NOWS，NHTWS) も計算している．表 19 の信頼性とは，各トポロジを構成するノードのサービス信頼性を SDP で求めた結果を示している．また，表 19 の T1〜T6 はそれぞれ図 19 のトポロジ (1)〜(6) に対応している．CA の信頼性に関しては，(T1)，(T2T4)，(T3T5T6) の間で違いが見られた．これらのトポロジではそれぞれ Web サービスを 1，3，6 個利用しており，信頼性については 0.9801，0.9415，0.8864 という結果になった．個々の Web サービスの信頼性に関しては，連携 WS を利用しない Web サービスでは，計測した信頼性 (サービス信頼性) がノード信頼性に等しくなるという結果になった．これは (T4T6) の各 Web サービスや (T1T2T3) で末端となる Web サービスや T5 の WS2WS4WS6 の信頼性から見て取れる．逆に，利用する連携 WS がさらに他の Web サービスを利用するといった連携 WS を利用している Web サービスは，連携 WS が他の Web サービスを利用している分だけ信頼性が下がるという結果になった．T3 を例に挙げると，CA が利用する WS1 は WS2 を利用し，WS2 は WS3 を利用し，WS3 は…というような連携方式であるが，WS6 から WS1 の順番で信頼性が下がっている．

表 19 SDP アルゴリズムによる信頼性評価

トポロジ	ノード	信頼性	NOWS	RFWS	NHTWS	トポロジ	ノード	信頼性	NOWS	RFWS	NHTWS
T1	CA	0.9801	1	2	1	T5	CA	0.8864	6	6	2
	WS1	1.0000	–	0	0		WS1	0.9801	–	2	1
T2	CA	0.9415	3	2	3		WS2	1.0000	–	0	0
	WS1	0.9606	–	2	2		WS3	0.9801	–	2	1
	WS2	0.9801	–	2	1		WS4	1.0000	–	0	0
	WS3	1.0000	–	0	0		WS5	0.9801	–	2	1
T3	CA	0.8864	6	2	6		WS6	1.0000	–	0	0
	WS1	0.9044	–	2	5	T6	CA	0.8864	6	12	1
	WS2	0.9227	–	2	4		WS1	1.0000	–	0	0
	WS3	0.9415	–	2	3		WS2	1.0000	–	0	0
	WS4	0.9606	–	2	2		WS3	1.0000	–	0	0
	WS5	0.9801	–	2	1		WS4	1.0000	–	0	0
	WS6	1.0000	–	0	0		WS5	1.0000	–	0	0
T4	CA	0.9415	3	6	1		WS6	1.0000	–	0	0
	WS1	1.0000	–	0	0						
	WS2	1.0000	–	0	0						
	WS3	1.0000	–	0	0						

5.3.3 Web サービスメトリクスと信頼性についての考察

NOWS メトリクスについては，表 19 に示されるように NOWS が 1，3，6 となる各トポロジにおいて CA の信頼性が 0.9801，0.9415，0.8864 となっており，NOWS メトリクスと信頼性に関連がみられる結果となった．これは，NOWS メトリクスでは Web サービスの利用数を計測するが，Web サービスの利用数が多くなれば信頼性が下がる要因となるネットワークを通したデータのやり取りが多くなるため，NOWS メトリクスが大きくなれば信頼性が下がるという結果になったと考えられる．

RFWS メトリクスと NHTWS メトリクスについては，RFWS メトリクスかつ NHTWS メトリクスの値が大きくなれば信頼性が下がる傾向が見られた．しかし，表 19 の T3 の CA，WS1〜WS5 の RFWS メトリクスは 2 であるが信頼性がそれぞれ異なっていることや，NHTWS メトリクスが 2 である T2 の WS1 および T3 の WS4 と T5 の CA において信頼性が異なっていることから，RFWS メトリクス単体や NHTWS メトリクス単体では，信頼性を評価できない場合がある結果となった．これは，RFWS メトリクス単体で信頼性を評価しようとしても，連携 WS の信頼性がトポロジによって変化するためだと考えられる．逆に，NHTWS メトリクス単体では，トポロジによって変化する連携 WS の信頼性を評価できるが，利用する連携 WS の数を評価できないため，信頼性を評価できないと考えられる．

表 20　提案メトリクスと信頼性との関係

提案メトリクス	評価対象	信頼性との関係
NOWS	ネットワークの利用量	値が大きくなると信頼性が悪くなる
RFWS	連携 WS の WS 利用数	値が大きくなると信頼性が悪くなる
NHTWS	連携 WS の連携 WS 利用数	値が大きくなると信頼性が悪くなる

5.4 Web サービスメトリクスに関するまとめ

本論文の評価実験で明らかになった，NOWS メトリクス，RFWS メトリクス，NHTWS メトリクスと効率性・信頼性との間の関連について表 21 にまとめる．3 つのメトリクスは効率性・信頼性について表 21 に示す項目において効率性・信頼性に関連がある結果となった．効率性においては，NOWS メトリクスでは Web サービスアプリケーション全体でのオーバーヘッドを，RFWS メトリクスでは連携 WS の処理を待つ数を，NHTWS メトリクスでは他の WS を待つ時間の量を評価できると思われる．信頼性においては，NOWS メトリクスは Web サービスアプリケーション全体でのネットワークの利用数から信頼性を，RFWS メトリクスと NHTWS メトリクスは両方のメトリクスを同時に用いることにより対象のサービス信頼性を評価できると思われる．EMWS メトリクスについては，本論文では機能性に関する評価実験を行えなかったために評価できなかった．

表 21 Web サービスメトリクスと品質の関係 (まとめ)

	評価できる項目	効率性	信頼性
NOWS	全体のオーバーヘッド	値が大きいと悪くなる	値が大きいと悪くなる
RFWS	連携 WS を待つ数	値が大きいと悪くなる	値が大きいと悪くなる
NHTWS	連携 WS を待つ時間の量	値が大きいと悪くなる	値が大きいと悪くなる
EMWS	本論文では評価を行えなかった		

6. 終わりに

本論文では，Web サービスアプリケーションの品質特性を評価することを目的に 4 種類の Web サービスメトリクスを提案し，その提案メトリクスと品質特性の関連を評価する実験を行った．まずは，既存のオブジェクト指向ソフトウェアメトリクス (C&K メトリクス) を Web サービスに適用可能か考察した．考察の結果，Web サービスの疎結合性と Web サービスでは継承関係が存在しない事からそのまま適用する事に問題があることを指摘し，その問題を考慮して新たな４つのメトリクス (RFWS, NOWS, EMWS, NHTWS) を提案した．そして，提案メトリクスの評価実験として，実際に構築した Web サービスアプリケーションであるバス時刻表検索サービスの品質との評価実験と，WS-PROVE を用いた Web サービスアプリケーションのプロトタイプによる効率性との評価実験，そして，SDP という信頼性評価アルゴリズムを用いて信頼性との評価実験を行った．評価実験の結果，提案メトリクスのうち RFWS, NOWS, NHTWS メトリクスについて Web サービスを用いて連携を行うことで生じる非機能的な部分での効率性・信頼性と関連があることが示された．今後の課題としては，EMWS に関する機能性の評価実験と，プロトタイプによる評価ではなく実際の Web サービスを多く利用した Web サービスアプリケーションで効率性や信頼性の評価を行う必要があると思われる．

謝辞

本研究を進めるにあたり多くの方々に，御指導，御協力頂きました．お世話になった方々に感謝の意を表したいと思います．

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，本ソフトウェア工学講座において新たな研究分野である本研究について研究を行う機会を与えて頂き，さらには本研究について主指導教官を担当していただき，一歩引いた立場から鋭い御指摘，御指導を頂きました．また本研究についてのみならず，大学院での活動と生活について多くの御指導と御助言を頂きました．心より感謝致します．

同　門田　暁人　助教授におかれましては，本研究の提案の定義において曖昧な部分を御指摘頂いたり，本研究の目指すべき所等を御助言して頂きました．心より感謝致します．

本研究を行う際に，直接指導して頂いた　同　中村　匡秀　助手におかれましては，本研究に興味を持つきっかけを頂いたり，数限りない御指導，御助言，御協力を頂きました．また，本研究の国内研究会の発表に際しては，論文の書き方の作法から発表の仕方まで，懇切丁寧にご教授して頂きました．また，本論文の執筆についても的確な御助言を頂きました．深く心より感謝致します．

同　大平　雅雄　助手におかれましては，短い間ながらも研究生活を共にし，広い見識を教えて頂きました．心より感謝致します．

副指導教官を担当して頂いた　同　小山 正樹　教授におかれましては，本研究の発表において，御意見，御指摘を頂きました．心より感謝致します．

副指導教官を担当して頂いた　同　飯田　元　助教授におかれましては，本研究の発表において，本研究が目指すべき所，本研究で明らかにしなければならない問題点など鋭い御指摘，御助言を頂きました．心より感謝致します．

同　ソフトウェア工学講座　博士後期課程　井垣　宏　様におかれましては，本研究を進めるにあたり，研究に行き詰まった時に貴重な意見を頂いたり，共に解決法について模索して頂きました．また，本論文の執筆においても，多大な御指導，御助言，御協力をして頂きました．深く心より感謝致します．

最後に，ソフトウェア工学講座の皆様には，本研究について多くの御助力と御

協力を頂きました．また，本研究についてのみならず，大学院での活動と生活，さらには大学院外での生活に関しても数え切れないくらいの御助言，ご協力をして頂きました．深く心より感謝いたします．

参考文献

[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] David A.Chappel and Tyler Jewell, Java Web サービス, 長瀬嘉秀, オライリー・ジャパン, 東京, 2002.

[4] Ethan Cerami, Web サービス, 長瀬嘉秀, オライリー・ジャパン, 東京, 2002.

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

[6] Hariri, S. and Raghavendra, C. S., ”SYREL: A symbolic reliability algorithm based on path and cutset methods”, IEEE Trans. Computers, 36, 1224-1232, 1987.

[7] 東基衛, ソフトウェア品質評価ガイドブック, 東基衛他編, 日本規格協会, 1994.

[8] 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, ”異なる設計・実装法を用いた Web サービスアプリケーションの開発および比較評価,” 信学技報, NS2003-315, pp.107-112, March 2004.

[9] 石井健一, 串戸洋平, 井垣宏, 中村匡秀, 松本健一, ”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開発,” 信学技報, NS2004-318, pp.361-366, March. 2005.

[10] 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, NS2003-316, pp.113-118, March 2004.

[11] 串戸洋平, 石井健一, 井垣宏, 中村匡秀, 松本健一, ”WS-PROVE を用いた Web サービスメトリクスの実験的評価,” 信学技報, NS2004-319, pp.367-372, March 2005.

[12] 大西 淳, 郷 健太郎, 要求工学, 共立出版, 2004.

[13] Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol.20, No.6, pp.476-493, June. 1994.

[14] Soh, S. and Rai, S., "CAREL: Computer aided reliability evaluator for distributed computing networks", IEEE Trans. Parallel and Distributed Systems, 2, 199-213, 1991.

[15] Tatsuhiro Tsuchiya, Tohru Kikuno, "Availability Evaluation of Quorum-Based Mutual Exclusion Schemes in General Topology Networks", The Computer Journal, Vol.42, No. 7, 1999.

[16] Tatsuhiro Tsuchiya, Tomoya Kajikawa, and Tohru Kikuno, "Parallelizing SDP (Sum of Disjoint Products) Algorithms for Fast Reliability Analysis", IEICE Transactions on Information and Systems, Vol.E83-D, No.5, pp.1183-1186, May 2000.

[17] Vonk,R.: Prototyping - The effective use of CASE technology, Prentice Hall Int., 1990 (黒田純一郎訳: 「プロトタイピング - CASE テクノロジーの有効利用」共立出版, 1992.)

[18] Yacoub S., Ammar H. and Robinson T., "Dynamic Metrics for Object Oriented Designs", Proc. of the Sixth International Symposium on Software Metrics, pp50-60, Boca Raton, Florida, November, 1999.