特別研究報告書

コンテキスト依存な通信による Web サービスの高速化

指導教員　　石田 亨 教授

京都大学工学部情報学科

松村 郁生

平成 18 年 2 月 10 日

コンテキスト依存な通信による Web サービスの高速化

松村 郁生

内容梗概

一般的な Web サービスでは，相互運用性を確保するために，データ交換のプロトコルとして SOAP を用いる．しかし，SOAP は XML ベースであるため，CORBA や DCOM など従来の分散オブジェクト技術で可能であった処理速度（パフォーマンス）が得られない．また，パフォーマンスの問題は，ユビキタスコンピューティング環境において限られた通信帯域・ハードウェア資源の下でサービス呼び出しを行う際に顕在化し，広範な分野における Web サービスの適用を妨げている．

SOAP のパフォーマンスを低下させる要因として，XML の高い処理コスト・転送コストの問題がある．また，事前にクライアントとサーバ間で合意が形成されない場合には，単一プロトコルの利用による機会損失の問題がある．さらに，連続してリクエストが行われる場合には，静的なパラメータの送信による冗長性の問題がある．

上記の問題を解決するために，本研究では，高速化に必要な情報がクライアントのコンテキストとして捉えられることに着目し，サーバがクライアントのコンテキストに適応することで高速な通信を実現することを目的とする．コンテキスト依存な Web サービスを実現する際には，以下の課題が生じる．

相互運用性の維持とプロトコルの多様性の両立

クライアントのコンテキストとして，クライアントが利用可能なプロトコルが考えられる．プロトコルが利用されるためにはサーバ・クライアント間で合意形成が必要であるが，従来この合意形成は人間が静的に行っているために，多様なプロトコルに対する実行時における相互運用性が保障されない．従来の Web サービスではこの相互運用性の問題を，SOAP を固定的に用いることで解決しているが，クライアントのコンテキストとして多様なプロトコルによる通信を許した場合にはこの問題が生じる．

静的なパラメータの獲得と動的なパラメータとの合成

クライアントのコンテキストとして，Web サービスに送信するパラメータのうち何が変化しないもの（静的パラメータ）で，何が頻繁に変化するもの（動的パラメータ）かについての情報（オペレーション用法）が考え

られる．この際，クライアントが静的パラメータをどのように獲得するのか，及び，サーバが，保持している静的パラメータと動的パラメータをどのように合成してサーバアプリケーションを実行するのかが問題となる．

本研究では，まず，オペレーション用法の獲得の方法として，静的なものと動的なものの２つを示した．また，静的パラメータと動的パラメータの合成が，RPC(Remote Procedure Call) スタイルにおいては動的パラメータの順序にもとづく穴埋めでできること，ドキュメントスタイルにおいてはXML のテンプレート方式が適用できることを示した．さらに本研究では，SOAP を用いて通信路を確保し，コンテキストに応じた動的なバイパスを作成することで，相互運用性を維持しながら多様なプロトコルを利用できるアーキテクチャを提案した．

また，本研究では提案したアーキテクチャを実装し，一般に考えられるクライアントのコンテキストの一例として，簡単な翻訳を対象にしたWeb サービスと Java RMI によるバイパスを用いた実験を行った．実験の結果，バイパス通信時には10 倍以上の高速化が達成され，総合的には，コンテキスト変化１回に対しおよそ 3 回以上のリクエストが行われる場合に高速化が達成されることが分かった．

本研究の貢献は次の通りである．

多様なプロトコルを汎用的に扱うアーキテクチャ

すでに広く用いられているSOAP を用いて通信路を確保し，コンテキストに応じて動的に作成したバイパスを用いて通信を行うことで，Web サービスの汎用性を維持しながらパフォーマンスの問題を改善するアーキテクチャを提案した．SOAP 通信路を用いてプロトコル利用の合意形成を自動的に行うことで，相互運用性とプロトコルの多様性の両立が実現できた．また，実験によりアーキテクチャのバイパス作成コストの評価を行った．

パラメータの獲得と合成手法

Web サービス実行のパラメータには静的なものと動的なものがある．このうち，静的パラメータの獲得方法について２通りの方法を示した．また，２種類のパラメータの合成方法を，RPC スタイル，ドキュメントスタイルの場合について示した．それぞれの手法を用いることで，コンテキスト依存なWeb サービスにとって必要な静的パラメータの獲得と，動的パラメータとの合成を行うことができる．

Performance Improvement of Web Services by Context-Aware Communication

Ikuo MATSUMURA

Abstract

Web Services generally use SOAP protocol for data exchange to ensure interoperability. But, since SOAP is an XML based protocol, Web Services cannot gain performance in comparison with former distributed object techniques such as CORBA and DCOM. Furthermore, the performance problem becomes obvious when service invocation is performed under limited bandwidth and hardware resource in ubiquitous computing environment, and the problem prevents Web Services from being applied to wide-ranging area.

There are some causes of Web Services' performance loss: (1) higher cost of processing and transmitting XML documents, (2) opportunity loss by using sole protocol when clients and servers cannot reach agreement and (3) redundancy of sending static parameters when repeated requests are performed.

In order to cope with these problems, this research proposes that information needed for performance improvement can be grasped as context of the client, and sets a goal of achieving high Web service performance by adapting to client contexts. In realizing context-aware Web Services, the following problems must be considered:

Coping with both interoperability and diversity of protocols

Available protocols can be grasped as client contexts. Although clients and servers need to reach an agreement to use protocols, run time interoperability of various protocols is not secured because these agreements have been made statically by human beings in the past. Though former Web Services cope with this interoperability problem by adhering to the SOAP protocol, this problem arises in situations where communication by various protocols as client context is permitted.

Acquisition of static parameters and composition with dynamic parameters

The information (operation usage) about which parameters are unchangeable (static parameters) and which parameters frequently change (dynamic parameters) can be grasped as client contexts. In this case, how

to acquire static parameters for clients and how to compose static parameters with dynamic parameters to execute application should be considered.

In this research, two methods (dynamic and static) for acquiring operation usage are presented. We can compose dynamic and static parameters by filling up by sequence of dynamic parameters in RPC (Remote Procedure Call) style, and we can compose them by applying XML template method.

Furthermore, an architecture which can dynamically use different protocols while retaining interoperability was proposed by ensuring a channel of widely-used SOAP and establishing bypass channels which are dynamically created to reflect client contexts.

The proposed architecture was implemented and a performance evaluation was conducted using the Java RMI bypass and the simple translation Web Service as an example of client context. The result showd that the speed increased more than ten times in bypass communication and when approximately three times or more requests were made for every single context change, performance improvement was to be achieved on the whole.

The contribution of this research is as follows:

General architecture handling various protocols

Architecture is proposed in which performance improvement is achieved while preserving generality of Web Service by ensuring a channel of widely-used SOAP and by communicating through bypass channels which are dynamically created. Coping with both interoperability and diversity of protocols is achieved by automatically reaching agreement of protocol use using the SOAP channel. Performance evaluation on bypass creation cost of the architecture was performed, and the effectiveness of the architecture was verified

The methods for acquisition and composition of parameters

Parameters of Web Services include static ones and dynamic ones. Two ways of acquiring static parameters were presented. Moreover, a way of composing two kinds of parameters is presented first in RPC style and second in document style. Using these methods, acquisition of static parameters and composition with dynamic parameters can be achieved.

コンテキスト依存な通信による Web サービスの高速化

目次

第１章　はじめに

SOA(Service Oriented Architecture) の実現形態として Web サービスが重要な技術であると考えられている．一般的な Web サービスでは，相互運用性を確保するために，データ交換のプロトコルとして SOAP[1] を用いる．しかし，SOAP は XML ベースであるため，CORBA や DCOM など従来の分散オブジェクト技術で可能であった処理速度（パフォーマンス）が得られない．また，パフォーマンスの問題はユビキタスコンピューティング環境において限られた通信帯域・ハードウェア資源の下でサービス実行を行う際に顕在化し，広範な分野における Web サービスの適用を妨げている．

パフォーマンスを低下させる要因として次の問題がある．

ＸＭＬの高い処理コスト・転送コスト

Web サービスで一般に用いられるプロトコルである SOAP において転送構文として用いられる XML を扱うことがパフォーマンス低下の要因となる．XML はキャラクタベースの符号化でありバイナリ表現に比べてデータサイズが大きくなるため，転送に時間がかかるのに加え XML の生成（シリアライズ）と解析（デシリアライズ）にも多くの時間を要する．また，タグ名や名前空間宣言の重複，属性空白の扱いなど XML にみられる各種の冗長性も XML の生成，解析時間を増大させる原因となっている．

単一プロトコルの利用による機会損失

Web サービスで一般的に行われるのは固定的な SOAP 通信であるが，場合によっては SOAP より高速なプロトコルによる通信が行える可能性がある．例えば，クライアントがファイアウォールを隔てた位置にあるときには SOAP を用いて HTTP による通信を行なわなければならないが，クライアントがファイアウォールの内側にあるときはより高速なプロトコルを用いることができる．このような状況で SOAP 通信を行うことで，他の高速な通信プロトコルを利用する機会が失われている．

静的なパラメータの送信による冗長性

あるサービスを繰り返し利用する場合，そのサービスに対する入力パラメータのうち変化のないもの（静的パラメータと呼ぶ）を毎回送信するのは非効率である．Web サービスは様々なクライアントが利用することを想定して提供されるためその論理的インタフェースも汎用的なものとなるが，

そのようなインタフェースを特定のクライアントが利用する際には利用状況に即さない静的なパラメータが生じてしまうことが十分に考えられる．

Web サービスの理念は，インターネットを介して言語やプラットフォームに依存しないアプリケーション同士の RPC(Remote Procedure Call) 及びデータ交換を実現することである．しかし，この目的のために Web サービスの仕様は本質的に汎用性を追及しなければならず，この汎用性の追及が上に述べた問題を生じさせている．

上記の問題を解決するために，本研究では，高速化に必要な情報がクライアントのコンテキストとして捉えられることに着目し，サーバがクライアントのコンテキストに適応することで高速な通信を実現することを目的とする．コンテキスト依存な Web サービスを実現する際には，以下の課題が生じる．

相互運用性の維持とプロトコルの多様性の両立

クライアントのコンテキストとして，クライアントが利用可能なプロトコルが考えられる．プロトコルが利用されるためにはサーバ・クライアント間で合意形成が必要であるが，従来この合意形成は人間が静的に行っているために，多様なプロトコルに対する実行時における相互運用性が保障されない．従来の Web サービスではこの相互運用性の問題を，SOAP を固定的に用いることで解決しているが，クライアントのコンテキストとして多様なプロトコルによる通信を許した場合にはこの問題が生じる．

静的なパラメータの獲得と動的なパラメータとの合成

クライアントのコンテキストとして，Web サービスに送信するパラメータのうち何が変化しないもの（静的パラメータ）で，何が頻繁に変化するもの（動的パラメータ）かについての情報（オペレーション用法）が考えられる．この際，クライアントが静的パラメータをどのように獲得するのか，及び，サーバが，保持している静的パラメータと動的パラメータをどのように合成してサーバアプリケーションを実行するのかが問題となる．

以降，本稿の構成は次のとおりである．まず，第 2 章で Web サービスの高速化の各種手法と，本稿のアプローチとその位置づけを述べる．次に，第 3 章でコンテキストの定義とそれをシステムが獲得・利用する方法について述べる．第 4 章では本研究のアプローチを実現するアーキテクチャについて考察する．第 5 章では，アーキテクチャの実装について述べ，第 6 章では実装したアーキテクチャのパフォーマンスの実験について述べ，評価を行う．そして第 7 章でまと

めを行い，本稿を締めくくる．

第２章　Webサービスの高速化

この章では，本稿の目的とするWebサービスの高速化に関する関連研究の紹介と，本稿で採用するアプローチの位置づけについて述べる．

2.1　関連研究

Webサービスのパフォーマンスの問題は以前から指摘されており，これまでに数多くの研究がなされてきた．Webサービスの高速化手法には，具体的に以下のものがある．

テキスト/SOAP/XMLに特化したバイナリ表現を用いる手法

本来テキスト形式であるXMLをバイナリで表現することにより，XMLの高い処理・転送コストを回避するアプローチがある．

まず，gzipなどの汎用のテキスト圧縮変換をOSI参照モデルにおけるプレゼンテーション層に付加することが考えられる．圧縮によりバイト効率が上がりトランスポート層以下の処理は高速になるが，プレゼンテーション層の処理速度は低下する．

XBIS[2]やFast Infoset[3]は，符号化の工夫によりXMLのシリアライズ・デシリアライズの時間を短縮し，バイナリベースで変換後のデータサイズも小さく抑えるデータ変換である．これらの変換技術では変換前後でテキストとしての同値性は保障されないが，XMLの論理的な同値性は保障される．例えば，これらの変換の前後では属性の順序が変わったり空白や改行の数が変わったりするが，論理的には同値である．しかし，送信されるメッセージの情報自体はXMLによるSOAPと同じであり，名前空間，タグなどの自己記述による冗長性が依然として残っている．

Fast Web Service[4]は，XML SchemaとASN.1[1)]とのマッピングを定義した仕様[2)]およびFast Infosetに基づき，メッセージスキーマ情報の共有を前提としたバイナリ通信を行うことで高速化を図るものである．XML用に設計された変換技術と異なり転送メッセージに名前空間やタグの情報

[1)] CCITT X.208/ISO 8824, CCITT X.209/ISO 8825

[2)] ITU-T X.694

が含まれないため，自己記述性は損なわれるが転送メッセージの冗長度はより減少する．

SOAP における XML の類似性に着目する手法

SOAP で交換される XML は複数回のやりとりにおいて共通する部分が大きく，変化する部分はパラメータ部分など一部に過ぎない．このような XML の特徴に注目し，以前にやり取りしたメッセージとの差分処理を行うことで XML の高い処理コストを回避する手法がある [5][6]．これらの手法では転送するメッセージは XML であるので，クライアント・サーバどちらかのみの変更も可能である．プレゼンテーション層での処理時間が小さくなるが，バイト効率は改善されずトランスポート層以下の処理時間は変わらない．

SOAP と他のプロトコルを切替利用する手法

単一プロトコル利用による機会損失の問題を回避するアプローチに，WSIF[7] がある．WSIF は WSDL のバインディングに SOAP 以外のプロトコルバインディングを記述し利用することで，より効率的なプロトコルによる通信を行うフレームワークである．しかし，静的パラメータに関する冗長性の問題には取り組んでいない．

2.2 コンテキストへの適応による高速化

バイナリ表現を用いる手法及び XML の類似性に着目する手法は，XML の高い処理・転送コストの問題を回避するものである．SOAP と他のプロトコルを切替利用する手法は，単一プロトコル利用による機会損失の問題を回避すると同時に，切り替えた結果のプロトコルがバイナリベースである場合など，XML の高い処理・転送コストの問題も同時に回避することができる．

本稿では，利用可能なプロトコル及び静的なパラメータをクライアントのコンテキストとして捉え，サーバがそのコンテキストに適応することで単一プロトコル利用による機会損失の問題と静的パラメータの送信による冗長性の問題を回避するアプローチをとる．

2.2.1 OSI における対象レイヤの対比

コンテキスト依存な Web サービスでは，送信するパラメータ自身すなわちアプリケーション層におけるデータの送信を限定することにより通信量を削減すると同時に，セッション層における処理を効率化する．このため，プレゼンテー

ション層およびトランスポート層以下を対象とした既存の高速化手法と独立に実現することが可能である．コンテキスト依存な Web サービスは，既存の Web サービスプロトコルである SOAP，Fast Infoset，Fast Web Service などに追加する形で適用できる．表 1 に，本手法を含む各種高速化手法が対象としている OSI 参照モデルにおけるレイヤを示す.「○」は手法の適用によって性能が向上することを示し,「×」は手法の適用によって性能が低下する可能性を示す.

表 1: 各高速化手法と OSI における対象レイヤ

	XML 差分 [5][6]	XMil, gzip	XBIS, Fast Infoset	Fast Web-Service	本手法
アプリケーション層					○
プレゼンテーション層	○	×	○	○	
セッション層					○
トランスポート層以下		○	○	○	

2.2.2 呼び出しモデルの対比

サービスを呼び出すモデルは，従来の手法では図 1 のようにパラメータの送信と応答からなる単純なものである．これに対し本稿によるアプローチでは,

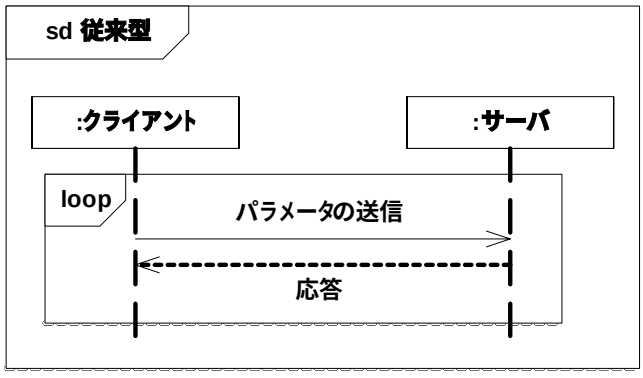

図 1: 従来の呼び出しモデル

サーバがコンテキストに適応する必要があるために，呼び出しモデルは図 2 のようになる.

コンテキスト依存な呼び出しモデルでは，まずクライアントがコンテキスト

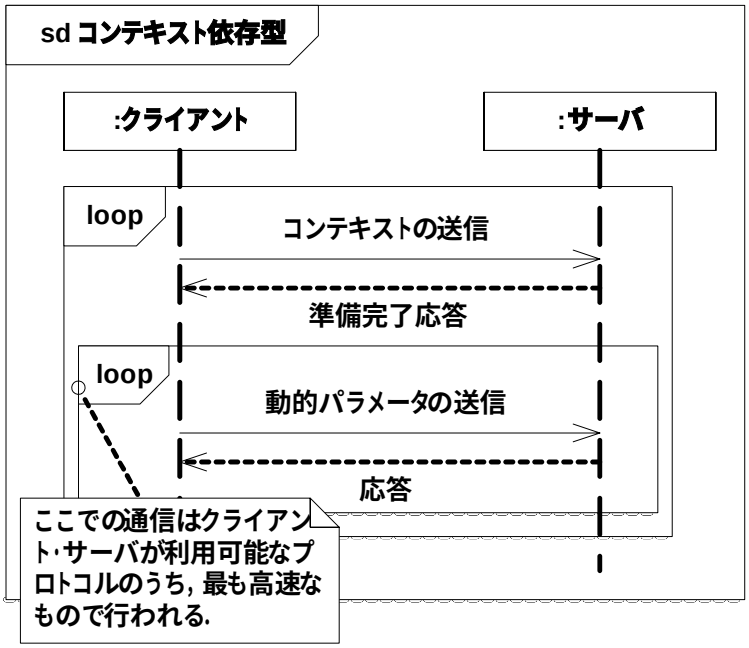

図 2: コンテキスト依存な呼び出しモデル

をサーバに送信し，サーバは受け取ったコンテキストに対する準備を行う．その後にコンテキストの変化が生じない限り，クライアントは以前に送信したコンテキストを前提として動的パラメータのみを送信することができる．またこの際には，利用可能なプロトコルのうち最も高速なものが用いられる．

第 3 章　Web サービス実行のためのコンテキスト

この章では，第 2 章で述べたアプローチにおいて重要な役割を果たすコンテキストについて，その定義と扱い方を述べる．

3.1　コンテキストの定義

クライアントのコンテキスト (Context) とは，まとまった回数の連続するリクエストの間一定であるような情報のことである．一般にクライアントのコンテキストは幅広く捉えることができるが，ここでは高速化の視点に限定し，コンテキストを以下の 2 つから構成されるものとする．

1. 利用可能プロトコル
2. オペレーション用法

以下それぞれについて説明する．

3.1.1　利用可能プロトコル

クライアントが利用可能なプロトコルは頻繁に変化することはないため，コンテキストとして捉えることができる．利用可能プロトコルが変化する場合には，次の 2 つが考えられる．

1. クライアントとサーバの置かれたネットワーク環境が変化した場合
2. プロトコルを扱うモジュール群に変更があった場合

前者は，例えばあるクライアントがサーバのファイアウォール外にあり HTTP による通信が必要であったが，その後サーバのファイアウォール内に移動したため HTTP 以外のより高速なプロトコルによる通信が可能になったという場合である．後者は，例えば Java RMI のプロトコルを解釈するためのモジュールが追加されたり，無効化されたりする場合である．いずれの場合も頻繁に生じることはないため，利用可能プロトコルはコンテキストと捉えることができる．

利用可能プロトコルをコンテキストとして捉えることは，単一プロトコルの利用による機会損失の問題を回避することとなる．

3.1.2　オペレーション用法

連続する複数のリクエストの間で一定であることが分かっている Web サービスの入力パラメータを，静的パラメータと呼ぶ．これに対し常に複数の連続するリクエストの間で変化するようなパラメータを動的パラメータと呼ぶ．入力パラメータのうち何が静的パラメータで何が動的パラメータとなるかは，Web サービスインタフェースのオペレーションの利用法に依存し，この情報をオペレーション用法と呼ぶ．オペレーション用法はまとまった回数の連続するリクエストの間で変化することはないため，コンテキストと捉えることができる．

オペレーション用法をコンテキストとして捉えることは，静的パラメータの送信による冗長性の問題を回避することとなる．

3.2　コンテキストの獲得

クライアントアプリケーションが Web サービスを呼び出す際には，呼び出すサービスに対するプロキシを用いることになる．サーバがクライアントのコンテキストに適応するためには，このプロキシがまずクライアントのコンテキストを獲得しなければならない．プロキシがコンテキストを獲得する方法には，次の 2 通りが考えられる．

静的に獲得する方法

クライアントアプリケーションが，プロキシに静的にコンテキストを通知する方法．静的にコンテキストを指定するのでプロキシに負荷が生じないが，クライアントアプリケーション側でコンテキストを指定するためのコーディングが必要となり，柔軟性が損なわれる．

動的に獲得する方法

プロキシ自体がクライアントアプリケーションから受けたリクエストをもとに，動的にコンテキストを獲得する方法．プロキシが従来の Web サービスプロキシと同様のインタフェースをもつため汎用性が高いが，リクエストのたびにコンテキストを獲得するための計算が生じるためにプロキシに負荷が生じる．

いずれの方法も一長一短がある．本稿は Web サービスのパフォーマンスの問題に対処することを目的とするので，以降では，より高速な通信が可能な前者の方法を採用し，詳細に検討する．

3.3 コンテキストの利用

3.2 節ではどのようにコンテキストを獲得するかについて述べた．次に，このようにして獲得されたクライアントのコンテキストがサーバに伝えられた後，サーバがそのコンテキストをどのように高速化に利用するかについて述べる．

本稿では，図 3 のようにコンテキストに応じたバイパスを作成する手法を提案する．

まず何もない状態からの初回の呼び出しは，プロキシからサーバ側のリスナを通じてサーバアプリケーションへと到達する．この際，クライアントのコンテキストを同時に送信し，サーバ側では受け取ったコンテキストに適応したバイパスを作成する．以後の呼び出しは，コンテキストの変化がない限り初回に用いた通信路ではなくバイパスを用いて通信を行う．

3.4 パラメータの合成

コンテキストが伝えられる際に静的なパラメータがサーバに送られると，その値はサーバに保持される．以後の呼び出しで動的パラメータが送られてくると，サーバは保持された静的パラメータに動的パラメータを合成し，サーバアプリケーションへの完全なパラメータを作成しなければならない．このパラメータの合成は，RPC スタイルの Web サービスとドキュメントスタイルの Web サー

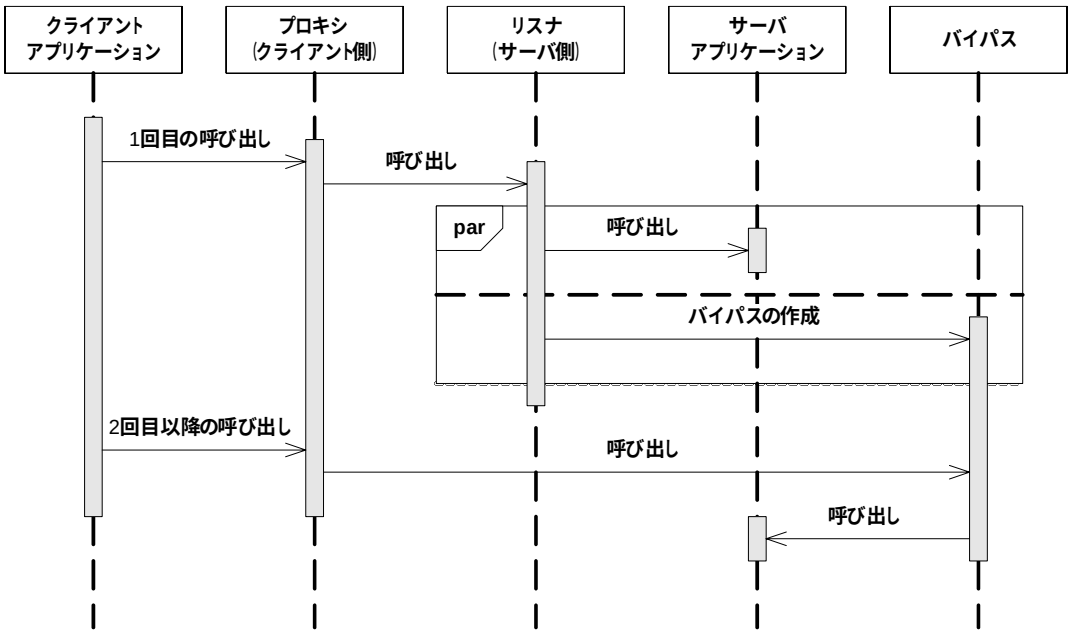

図 3: バイパスを用いたサービス呼び出しのシーケンス図

ビスで異なったものとなる．

RPC スタイルにおいては，動的パラメータの集合を受け取ったときに，引数のうち静的パラメータで占められていないインデックスに，受け取った順番に動的パラメータを対応させるという方法が考えられる．例えば 5 つの引数 $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$ をもつサーバアプリケーションに対して $(\alpha_1, \alpha_2, \alpha_4) = (a_1, a_2, a_4)$ という静的パラメータがあるとする．このとき動的パラメータが (b_1, b_2) の順に送られてきた場合には $\{a_1, a_2, b_1, a_4, b_2\}$ という引数列を作成し，サーバアプリケーションに適用する．

ドキュメントスタイルにおいては，引数の概念がないため上記の方法は使えない．代わりに [6] で示されている XML のテンプレートのオートマトンによる表現が利用できる．すなわち，静的パラメータを XML テンプレートのオートマトンとして表現し，動的パラメータをオートマトンの入力として与えることで，サーバアプリケーションに対する完全なパラメータを復元することができる．

第4章　コンテキスト依存なWebサービスのアーキテクチャ

第3章では，コンテキストの定義と扱い方について述べた．本章ではより具体的に，コンテキストに依存したWebサービスのアーキテクチャについて述べる．

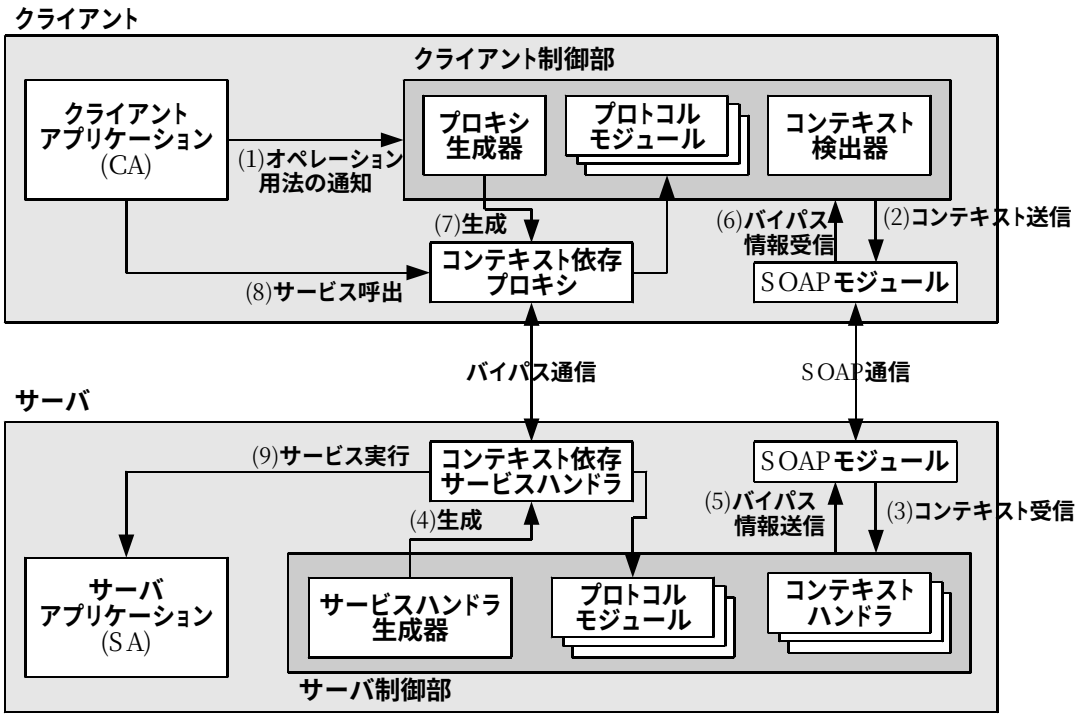

図4: コンテキスト依存なWebサービスのアーキテクチャ

4.1　アーキテクチャの設計

コンテキスト依存なWebサービスのアーキテクチャは，図4のようになる．

クライアントとサーバ間の通信にはSOAPによる通信と，コンテキストに応じて動的に生成されるバイパスによる通信の2通りがある．クライアント，サーバはそれぞれCORBAやRMIなど，SOAPより高速な代替プロトコルモジュールを持っている．

何もない状態からの初回の呼び出し及びその次の呼び出しは，次のような流れになる．但し，括弧内の数字は図中の注釈に対応する．

1. (1) クライアントアプリケーションが，オペレーション用法Sをクライアン

ト制御部に通知する．

2. クライアント制御部は利用可能プロトコル P を検出し，S と P からサーバに送信するコンテキスト C を得る．
3. (2) クライアントが，自身のコンテキストを SOAP 通信によりサーバに送信する．
4. (3) サーバがコンテキスト C を受信すると，サーバは P のうちサーバ側でも対応しているプロトコルのうち，バイパス通信で利用するもの p を選択する．p はサーバ側で最も高速なプロトコルと判断できるものを選ぶ．
5. その後 (4) サービスハンドラ生成器を用いて C と p に基づいたコンテキスト依存なサービスハンドラを生成する．
6. SOAP 通信の応答として (5) 作成したバイパス情報をクライアントに送信する．バイパス情報には p が含まれる．
7. (6) クライアントがバイパスの情報をサーバから受け取る．
8. (7) プロキシ生成器を用いて C と p に基づいたコンテキスト依存なプロキシを生成する．
9. クライアントアプリケーションは，このように作成されたプロキシを用いて (8) サービス呼び出しを行う．
10. (9) 生成されたサービスハンドラが，リクエストをサーバアプリケーションに対して実行する．

一旦バイパスが作成された後の呼び出しは，コンテキストに変化がない限り (8)(9) の経路を通じて行われる．コンテキストに変化が生じると，上記の手順をもう一度繰り返す．

初回に通信を行うプロトコルとして既に広く利用の合意形成がなされている SOAP を用い，バイパスによる通信時に採用されるプロトコルの選択を自動的に行うことで，Web サービスとしての汎用性を維持することができ，多様なプロトコルの汎用性の欠如の問題が解決される．

4.2 バイパス作成のためのメッセージ

上述のアーキテクチャにおいて，クライアントのコンテキスト及び作成したバイパス情報を SOAP 通信路において伝達しなければならない．これらの情報の格納は，SOAP のヘッダを用いて行われる．このためのスキーマを，XML Schema を用いて定める．以降で，定義したスキーマについて詳述する．

4.2.1 コンテキスト情報のスキーマ

コンテキスト情報のスキーマ定義を，図 5 に示す．

clientContext 要素が，コンテキスト情報のルートを表す．clientContext の子要素に operationUsage と availableProviders があり，それぞれオペレーション用法と利用可能プロトコルを表す．

operationUsage 要素には，オペレーション用法として静的パラメータと動的パラメータの区別を記述する．RPC スタイルの Web サービスにおいては，staticIndex 要素に当該オペレーションの引数のうち何番目の引数が静的パラメータとなるかを 0 以上の整数で指定する．指定は，1 からではなく 0 から始まる．ドキュメントスタイルの Web サービスにおいては引数の概念が存在しないため，staticRange 要素に XPointer[8] などの XML の範囲を記述する言語を用いて，SOAP のボディにおける静的な範囲を指定する．

availableProviders 要素には，利用可能プロトコルを表す識別子を文字列として指定する．

リクエストの SOAP ヘッダに記述されるコンテキスト情報の例を，以下に示す．

```
<clientContext xmlns="http://i.kyoto-u.ac.jp/siws/soap/">
  <operationUsage>
    <staticIndex>1</staticIndex>
    <staticIndex>0</staticIndex>
  </operationUsage>
  <availableProviders>
    <provider>rmi</provider>
    <provider>fastinfoset</provider>
  </availableProviders>
</clientContext>
```

この例では，operationUsage 要素内の staticIndex に 1 と 0 を指定することで，RPC スタイルにおける 1 番目と 0 番目 (最初の 2 つ) の引数が静的パラメータであることを示している．また，availableProviders 要素内の provider に識別子 rmi 及び fastinfoset を指定することで，それぞれに対応するプロトコルがクライアントにおいて利用可能であることを示している．

4.2.2 バイパス情報のスキーマ

バイパス情報のスキーマ定義を，図 6 に示す．

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://i.kyoto-u.ac.jp/siws/soap/"
  xmlns:tns="http://i.kyoto-u.ac.jp/siws/soap/" xmlns:Q1="xs">

  <element name="clientContext">
    <complexType>
      <all minOccurs="1" maxOccurs="1">
        <element name="operationUsage"
          type="tns:OperUsage" maxOccurs="1" minOccurs="1">
        </element>
        <element name="availableProviders"
          type="tns:AvailableProviders" maxOccurs="1" minOccurs="1">
        </element>
      </all>
    </complexType>
  </element>

  <complexType name="OperUsage">
    <sequence>
      <choice>
        <element name="staticIndex" type="nonNegativeInteger"
          maxOccurs="unbounded" minOccurs="0">
        </element>
        <element name="staticRange" type="string"
          maxOccurs="unbounded" minOccurs="0">
        </element>
      </choice>
    </sequence>
  </complexType>

  <complexType name="AvailableProviders">
    <sequence>
      <element name="provider" type="string" maxOccurs="unbounded"
        minOccurs="1">
      </element>
    </sequence>
  </complexType>
</schema>
```

図 5: コンテキスト情報のスキーマ

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://i.kyoto-u.ac.jp/siws/soap/"
  xmlns:tns="http://i.kyoto-u.ac.jp/siws/soap/" xmlns:Q1="xs">

  <element name="bypassInfo">
    <complexType>
      <sequence>
        <element name="provider" type="string"></element>
        <element name="param" type="tns:Param"
          maxOccurs="unbounded" minOccurs="0">
        </element>
      </sequence>
    </complexType>
  </element>

  <complexType name="Param">
    <simpleContent>
      <extension base="string">
        <attribute name="key" type="string" form="unqualified">
        </attribute>
      </extension>
    </simpleContent>
  </complexType>
</schema>
```

図 6: バイパス情報のスキーマ

bypassInfo 要素が，バイパス情報のルートを表す．bypassInfo の子要素に provider 要素と param 要素があり，それぞれサーバ側でバイパス作成に採用されたプロトコルとバイパスに関する付加情報を表す．

provider 要素には，作成されたバイパスに用いられるプロトコルの識別子を文字列で指定する．

param 要素には，作成したバイパスにアクセスするのに必要な情報をキーと文字列値のペアで任意個数記述する．key 属性にキーの名前を，要素のテキストノードに文字列値を記述する．

レスポンスの SOAP ヘッダに記述されるバイパス情報の例を，以下に示す．

```
<bypassInfo xmlns="http://i.kyoto-u.ac.jp/siws/soap/">
  <provider>rmi</provider>
  <param key="rmi_name">rmi://localhost:11099/siwsObj0</param>
</bypassInfo>
```

この例では，bypassInfo 要素内の provider にプロバイダの識別子 rmi を指定することで，対応するプロトコルによるバイパスが作成されたことを示す（この場合は Java RMI）．また param 要素には，rmi_name という名のキーに rmi://localhost:11099/siwsObj0 という値をもつ付加情報を示している．param 要素に記述される情報の意味解釈は各プロトコルを扱うモジュールに依存し，ここではキーの値は Java RMI のリモートオブジェクトのアドレスを表している．

第5章　実装

以上のようなコンテキスト依存な Web サービスのアーキテクチャを，Java 上に実装した．実装に当たっては，Apache Axis[1] 1.2.1 及び JAXB[2] を用いた．

Apache Axis は Java と C++による SOAP の実装であり，今回は Java の実装を利用した．JAXB(The Java Architecture for XML Binding) は XML と Java をマップするデータバインディング仕様とその実装であり，今回は図 5 のコンテキスト情報のスキーマと図 6 のバイパス情報のスキーマの取り扱いのために利用した．

実装の詳細を，クライアント側，サーバ側に分けて以下に述べる．なお，これらのプログラムのソースコードは本稿の付録に掲載した．

5.1　クライアント

クライアント実装において重要な構成要素の静的構造を，図 7 に示す．

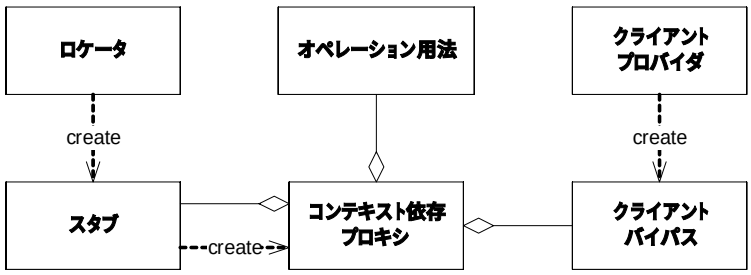

図 7: クライアントの構成要素の静的構造

[1] http://ws.apache.org/axis/

[2] https://jaxb.dev.java.net/

スタブ (Stub) は，特定の Web サービスを呼び出す際にクライアントアプリケーションがやり取りを行うコンポーネントである．ロケータ (Locator) はクライアントにおいて特定の Web サービスを表す構成要素で，特定の Web サービスに対するスタブを生成する役割を持つ．

コンテキスト依存プロキシ (SitutatedProxy) は，スタブにオペレーション用法 (OperationUsage) を与えることにより生成されるプロキシである．通常の Web サービスではスタブに対してオペレーション呼び出しのリクエストを行うが，コンテキスト依存な Web サービスではこのように生成されたプロキシに対してオペレーション呼び出しのリクエストを行う．

クライアントプロバイダ (ClientProvider) は，クライアントにおいて Java RMI, Fast Infoset などの様々なプロトコルを扱うための機能を提供するコンポーネントである．クライアントバイパス (ClientBypass) はサーバ側で用意されたコンテキスト依存なバイパスにアクセスするためのコンポーネントで，サーバから受け取ったバイパス情報に基づいてクライアントプロバイダにより作成される．

クライアントアプリケーションからこれらの構成要素がどのように利用されるかを，図 8 に示す．

処理の流れは次の通りである．

1. クライアントアプリケーションはまずロケータを生成し，そのロケータを用いてスタブを取得する．
2. ロケータはスタブを生成しクライアントアプリケーションにスタブを返す．
3. 平行して，クライアントアプリケーションはオペレーション用法インスタンスを生成し，静的パラメータをセットする．
4. 作成したオペレーション用法を引数に，作成したスタブに対してコンテキスト依存プロキシの取得を行う．
5. スタブがコンテキスト依存プロキシを作成してクライアントアプリケーションに返す．
6. クライアントアプリケーションは取得したコンテキスト依存プロキシに対してオペレーション呼び出しを行う．

ここでコンテキストのうち，オペレーション用法はクライアントアプリケーションが明示的に与えたが，利用可能プロトコルについては自動的に検出される．

ここで，クライアントアプリケーションが行わなければならないことが従来の

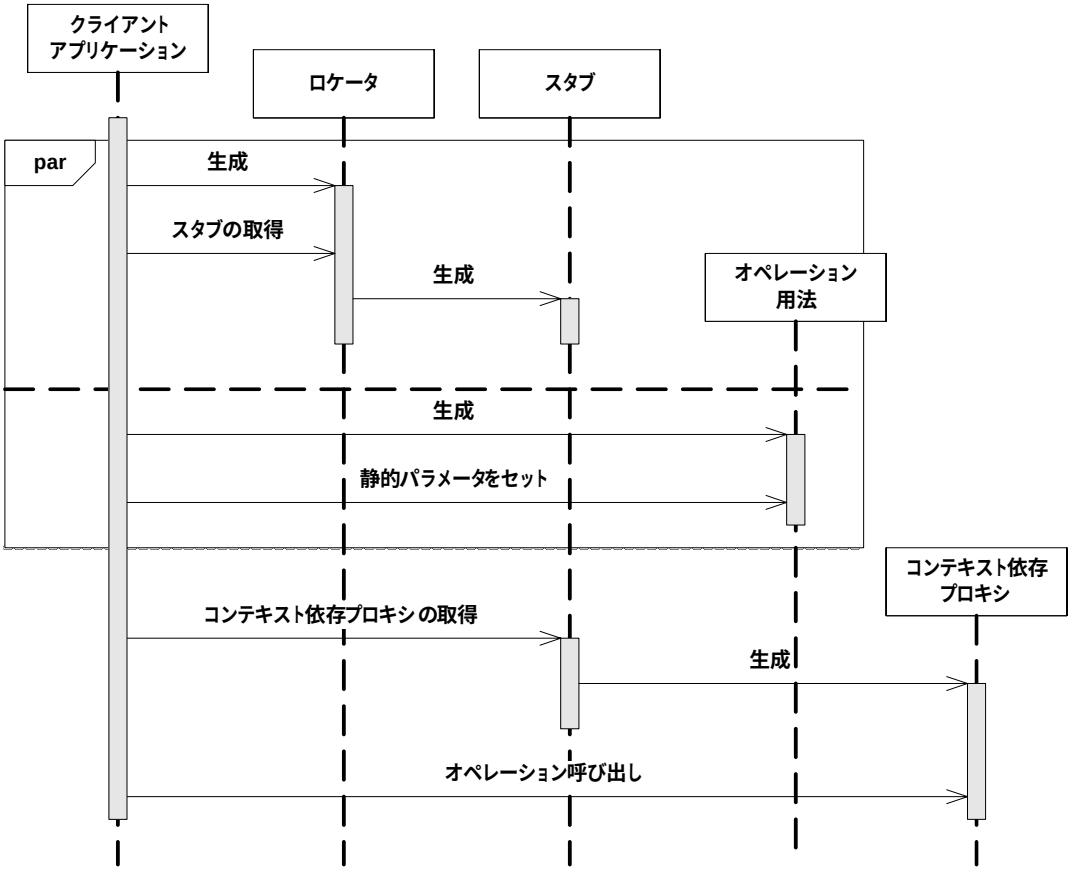

図 8: オペレーション呼び出しまでのシーケンス図

Web サービスと異なることに注意しなければならない．すなわち，従来の Web サービス呼び出しではスタブを生成した後はそのスタブに対してオペレーション呼び出しを行うのに対し，コンテキスト依存な Web サービスではオペレーション用法をスタブに与えてコンテキスト依存プロキシを取得した後でプロキシに対しオペレーション呼び出しを行うことになる．これは 3.2 節で述べたように，本稿で静的にコンテキストを獲得するモデルを採用したためである．動的にコンテキストを獲得するモデルでは従来どおりの呼び出し方が可能である．

クライアントアプリケーションのコーディング例を，以下に示す．

```
1:    // ロケータとスタブの取得
2:    SimpleTranslationService service =
       new SimpleTranslationServiceLocator();
3:    SimpleTranslation trans = service.getSimpleTranslation();
4:
5:    // オペレーション用法の作成
6:    OperationUsage usage = new OperationUsage(String.class);
```

```
 7:    usage.setStaticParam(0, "en_US");
 8:    usage.setStaticParam(1, "ja_JP");
 9:
10:    // コンテキスト依存プロキシの取得
11:    SituatedProxy proxy = trans.getProxyTranslate(usage);
12:    System.out.println(proxy.invoke(new Object{"Hello!"}));
```

この例では，まず2行目で SimpleTranslationServiceLocator というロケータのインスタンスを作成し，3行目で作成したロケータから SimpleTranslation という型のスタブを取得する．一方，6 行目で usage というオペレーション用法のインスタンスを作成し，7，8 行目で setStaticParam というメソッドで静的パラメータをセットしている．ここでは 0 番目の引数が “en_US”, 1 番目の引数が “ja_JP” という値であることを指定している．次に 11 行目で，3 行目で作成したスタブのインスタンス trans から usage を指定してコンテキスト依存プロキシ proxy を作成し，12 行目で”Hello!”という引数でオペレーションを呼び出している．このようなコードでサーバアプリケーションで呼び出されるのは translate("en_US", "ja_JP", "Hello!") となる．動的パラメータが複数ある場合には，配列に指定された順番どおりに，静的パラメータで占められていない場所に格納される．

5.2 サーバ

サーバの実装は，Apache Axis のアーキテクチャにおいてサービスのリクエスト処理の前及びレスポンス処理の後に，コンテキスト依存な Web サービスに必要なコンポーネントによる処理を追加する形で行った．

Apache Axis のサーバ側のアーキテクチャは大まかに，ハンドラ (Handler) と呼ばれるコンポーネントと，チェイン (Chain) と呼ばれる順序付けられたハンドラの集合から構成される．メッセージを処理する為にハンドラが Axis のフレームワークから順番に呼び出される．メッセージが HTTP サーブレットなどのトランスポートリスナに到着すると MessageContext と呼ばれるオブジェクトが作成され，パースされた SOAP のメッセージや様々なプロパティが MessageContext に取り込まれる．その後 MessageContext をパラメータとして後続のハンドラ及びチェインが次々と呼び出され，SOAP リクエストメッセージの解析と対象サービスの呼び出し，SOAP レスポンスメッセージの生成が行われる．

上述のハンドラには様々なものが存在し，新たに追加することが可能である．

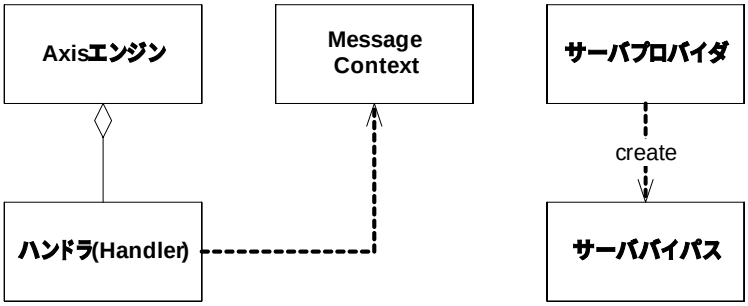

図 9: サーバの構成要素の静的構造

そこで，本稿では新たにコンテキスト依存な Web サービスのための処理を行うハンドラを作成し，Axis に組み込むという形で実装を行った．

新たに作成したハンドラなどを含むサーバ側の静的構造を，図 9 に示す．

Axis エンジン (AxisEngine) は SOAP プロトコルを処理するフレームワークであり，HTTP サーブレットなどのトランスポートリスナに届いたリクエストをもとに SOAP のメッセージなどを格納する MessageContext を作成する．ハンドラ (SiWSContextHandler) は，前述の Axis を構成している MessageContext を処理するコンポーネントのうち新たに作成したコンテキスト依存な Web サービス用のものである．

サーバプロバイダ (ServerProvider) は，サーバにおいて Java RMI，Fast Infoset などの様々なプロトコルを扱うための機能を提供するコンポーネントである．サーババイパス (ServerBypass) はコンテキスト依存なリスナであり，クライアントから受け取ったコンテキスト情報に基づいてサーバプロバイダにより作成される．

クライアントからのリクエストをこれらのサーバ側の構成要素がどのように処理するのかを，図 10 に示す．

初回の呼び出しにおける処理の流れは次の通りである．

1. クライアントが Axis エンジンに対して呼び出しを行う（初回）．
2. Axis エンジンがリクエストを受け付けると MessageContext を生成する．
3. 登録されたハンドラ (今回新たに作成したものを示す) を呼び出す．
4. ハンドラがリクエストの処理依頼を受けると，MessageContext からクライアントのコンテキスト（オペレーション用法と利用可能プロトコル）を取得する．
5. Axis エンジンが，対象となるサーバアプリケーションを実行する．

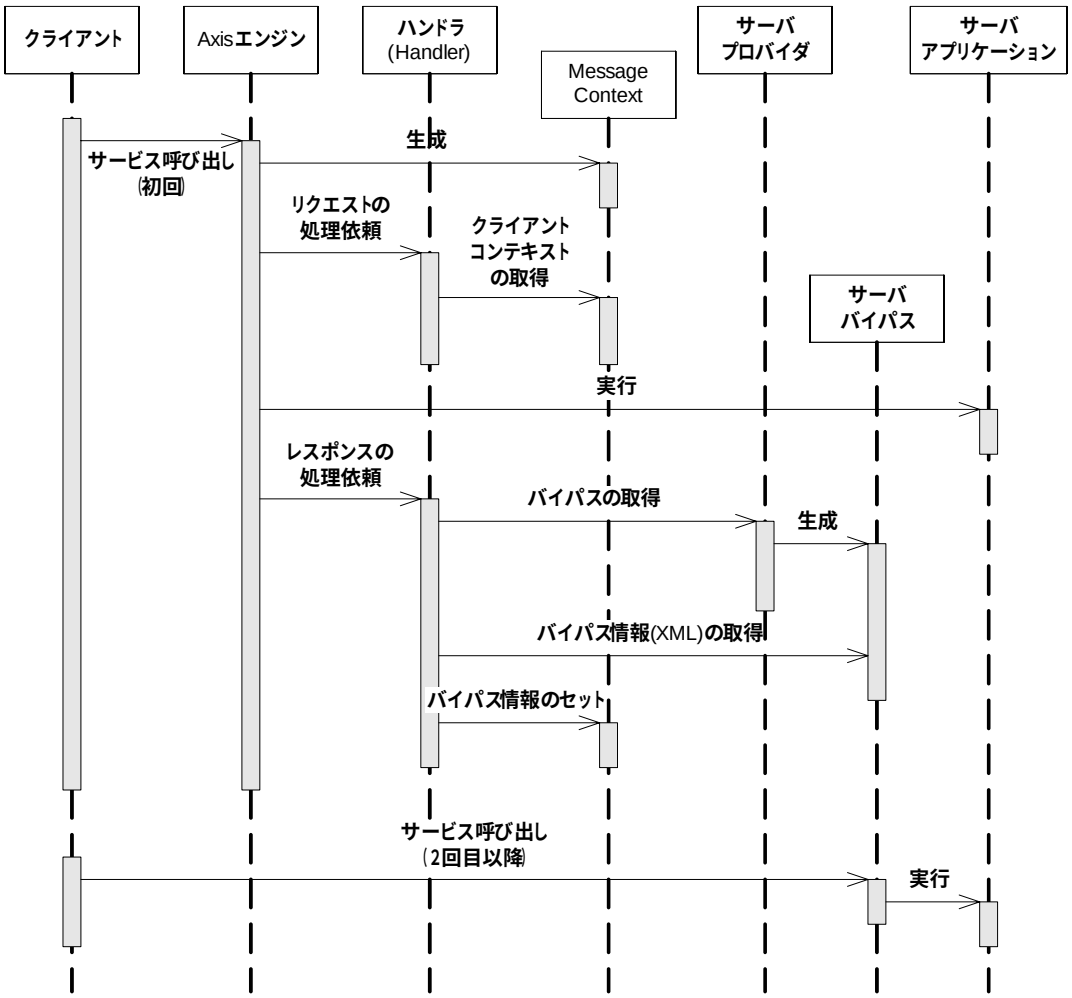

図 10: サーバ側構成要素のシーケンス図

6. レスポンスの処理の途中で再びハンドラが呼び出される.
7. ハンドラがサーバプロバイダに対してバイパスの作成を依頼する.
8. サーバプロバイダが，そのプロトコルに応じたサーババイパスを作成する.
9. ハンドラが，作成されたサーババイパスからそのバイパスにアクセスするための情報を取得し，MessageContext にセットする.
10. バイパス情報がセットされた MessageContext を Axis エンジンは SOAP のレスポンスメッセージへとシリアライズし，クライアントへのレスポンスとする.

クライアントの 2 回目以降の呼び出しは，コンテキストの変化が生じない限り初回に作成されたサーババイパスに対して行われる．コンテキストに変化が生じた場合には，初回と同様の手順でサーババイパスを作成しなおす.

第6章　バイパス作成コストの評価

第4章で述べたコンテキスト依存なWebサービスのアーキテクチャにおいて，バイパス通信時にはSOAPより高速なプロトコルを用いるので高速化が実現できるのは自明である．しかし，コンテキストに依存したバイパスを作成するためにはそれなりのコストがかかり，頻繁にコンテキストが変化する場合において高速化が達成できない．この章では第5章で述べた実装を用いた実験について述べ，バイパス作成コストを評価する．

6.1　実験

実験には翻訳サービスを想定した図11に示すWSDLのポートタイプをもつWebサービスSimpleTranslationを作成し，これを用いた．全てのインタフェース記述 (WSDL) を，付録のA.2.3に示す．

```
<wsdl:message name="translateResponse">
   <wsdl:part name="translateReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="translateRequest">
   <wsdl:part name="sourceLang" type="xsd:string"/>
   <wsdl:part name="targetLang" type="xsd:string"/>
   <wsdl:part name="text" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="SimpleTranslation">
   <wsdl:operation name="translate"
         parameterOrder="sourceLang targetLang text">
      <wsdl:input message="impl:translateRequest"
         name="translateRequest"/>
      <wsdl:output message="impl:translateResponse"
         name="translateResponse"/>
   </wsdl:operation>
</wsdl:portType>
```

図11: 翻訳サービスSimpleTranslationのポートタイプ（WSDLの一部）

ポートタイプ`SimpleTranslation`は，`translate`という1つのオペレーションをもつ．`translate`オペレーションは`sourceLang`, `targetLang`, `text` とい

う３つの引数をもち，それぞれ翻訳元言語，翻訳先言語，翻訳対象文を示す．なお，通信にかかる時間のみを測定するため，SimpleTranslation は実処理を何も行わないサービスとして実装してある．

このような Web サービスを対象に，以下の３つの場合についてサービス呼び出しにかかった時間の平均を調べた．

(a) 従来型の SOAP 通信時

(b) コンテキスト依存アーキテクチャにおける SOAP 通信時（バイパス準備時）

(c) コンテキスト依存アーキテクチャにおけるバイパス通信時

(b),(c) の場合において設定したクライアントコンテキストのうち，オペレーション用法については静的パラメータが `sourceLang`, `targetLang` の２つでありその値は

$$(\texttt{sourceLang}, \texttt{targetLang}) = (\text{“}\texttt{en\_US}\text{”}, \text{“}\texttt{ja\_JP}\text{”})$$

である．クライアントコンテキストのうち利用可能プロトコルは Java RMI のみである．すなわち，(c) におけるバイパス通信時においてやり取りされる情報（動的パラメータ）は引数 `text` に相当するデータであり，そのプロトコルは Java RMI となる．

100 回平均による実験の結果は，図 12 のようになった．左から順に，(a)(b)(c) の時間を表す．

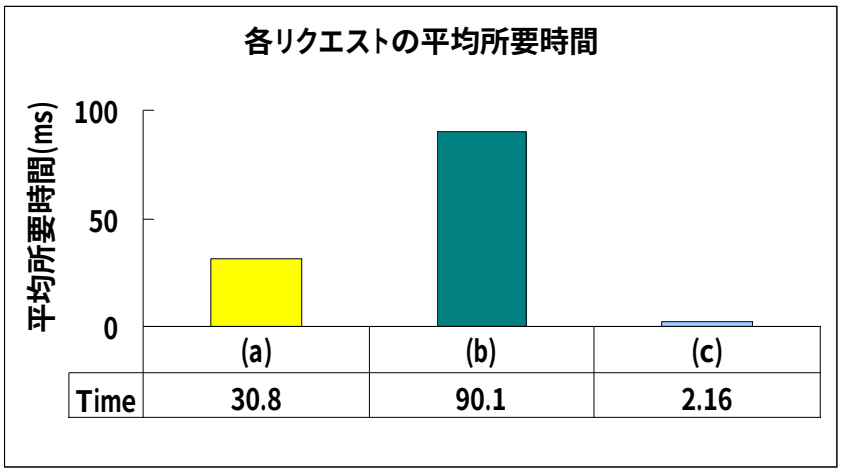

図 12: サービス呼び出しにかかった時間の平均

6.2 考察

上記の実験結果における (a)(b)(c) の実行時間の値を，それぞれ s, $r^{\emptyset}$, r とおく．実験の結果から，

$$s/r = 14.3$$

となり，バイパス通信時には 10 倍以上の高速化が達成されていることが分かる．

以下では，コンテキスト変化の頻度がどの程度以下だと，全てにかかる時間を含めて従来型の SOAP より高速であるかを評価する．

ここで，n 回のリクエストの間に k 回コンテキストが変化するとき，処理時間の総和の従来型 SOAP との比 f は，次式で表される．

$$f = \frac{kr^{\emptyset} + (n-k)r}{ns}$$

この式に実験の結果 $(s, r^{\emptyset}, r) = (30.8, 90.1, 2.16)$ を適用すると，次式が得られる．

$$f = 2.86\alpha + 0.07 \quad (\alpha = k/n)$$

高速化が達成できる限界 $f = 1$ のときの α の値は，

$$\alpha = 0.33$$

となる．これより，コンテキスト変化 1 回に対しおよそ 3 回以上のリクエストが行われる場合に，高速化が達成されることが分かる．

第 7 章　おわりに

本稿では，Web サービスの高速化に関する以下の問題を指摘した．SOAP のパフォーマンスを低下させる要因として，XML の高い処理コスト・転送コストの問題がある．また，事前にクライアントとサーバ間で合意が形成されない場合には，単一プロトコルの利用による機会損失の問題がある．さらに，連続してリクエストが行われる場合には，静的なパラメータの送信による冗長性の問題がある．

上記の問題を解決するために，本研究では，高速化に必要な情報がクライアントのコンテキストとして捉えられることに着目し，サーバがクライアントのコンテキストに適応することで高速な通信を実現した．

具体的には，まず，オペレーション用法の獲得の方法として，静的なものと

動的なものの２つを示した．また，静的パラメータと動的パラメータの合成が，RPC(Remote Procedure Call) スタイルにおいては動的パラメータの順序にもとづく穴埋めでできること，ドキュメントスタイルにおいてはXML のテンプレート方式が適用できることを示した．さらに本研究では，SOAP を用いて通信路を確保し，コンテキストに応じた動的なバイパスを作成することで，相互運用性を維持しながら多様なプロトコルを利用できるアーキテクチャを提案した．

また，本研究では提案したアーキテクチャを実装し，一般に考えられるクライアントのコンテキストの一例として，簡単な翻訳を対象にしたWeb サービスと Java RMI によるバイパスを用いた実験を行った．実験の結果，バイパス通信時には10 倍以上の高速化が達成され，総合的には，コンテキスト変化１回に対しおよそ 3 回以上のリクエストが行われる場合に高速化が達成されることが分かった．

本研究の貢献は次の通りである．

多様なプロトコルを汎用的に扱うアーキテクチャ

すでに広く用いられているSOAP を用いて通信路を確保し，コンテキストに応じて動的に作成したバイパスを用いて通信を行うことで，Web サービスの汎用性を維持しながらパフォーマンスの問題を改善するアーキテクチャを提案した．SOAP 通信路を用いてプロトコル利用の合意形成を自動的に行うことで，相互運用性とプロトコルの多様性の両立が実現できた．また，実験によりアーキテクチャのバイパス作成コストの評価を行った．

パラメータの獲得と合成手法

Web サービス実行のパラメータには静的なものと動的なものがある．このうち，静的パラメータの獲得方法について２通りの方法を示した．また，２種類のパラメータの合成方法を，RPC スタイル，ドキュメントスタイルの場合について示した．それぞれの手法を用いることで，コンテキスト依存なWeb サービスにとって必要な静的パラメータの獲得と，動的パラメータとの合成を行うことができる．

今後の課題としては，コンテキストをリクエストの時系列パターンに拡張することが挙げられる．リクエストの時系列パターンとは，例えば図 11 において (`sourceLang`, `targetLang`) の値が (“en_US”, “ja_JP”) → (“ja_JP”, “en_US”) → (“en_US”, “ja_JP”) → (“ja_JP”, “en_US”) → · · · と周期的に変化するようなパターンである．複数のリクエストにおける引数の規則的なパターンをコンテキ

ストとして伝達・利用することで，送信するパラメータの冗長性をさらに低減させることができる．

また，本稿で行ったバイパス作成コストの評価はSimpleTranslationという１つのWebサービスに対して１種類のオペレーション用法と１種類のバイパス用プロトコルを用いたものであるが，一般にWebサービス及びクライアントのコンテキストはこれに限らず多様なものが存在する．このため，より一般的な評価を行うことが望まれる．

謝辞

本研究を行う機会と環境を与えてくださり，熱意あるご指導を頂いた石田亨教授に深く感謝致します．また，日頃より助言や議論をしてくださった村上陽平氏，藤代祥之氏をはじめとする石田研究室の皆様に感謝致します．

参考文献

[1] Mitra, N. et al.: SOAP Version 1.2, W3C Recommendation (2003). http://www.w3.org/TR/soap12.

[2] Sosnoski, D. M.: XBIS XML infoset encoding, W3C Workshop on Binary Interchange of XML Information Item Sets (position paper) (2003).

[3] Sandoz, P. et al.: FastInfoset, Technical report, Sun Microsystems (2004). http://java.sun.com/developer/technicalArticles/xml/fastinfoset/.

[4] Sandoz, P. et al.: Fast Web Services, Technical report, Sun Microsystems (2003). http://developer.java.sun.com/developer/technicalArticles/WebServices/fastWS/.

[5] Takase, T. et al.: An Adaptive, Fast, and Safe XML Parser Based on Byte Sequences Memorization, Proceedings of the 14th International World Wide Web Conference, pp. 692–701 (2005).

[6] Takeuchi, Y., Okamoto, T. et al.: A Differential-analysis Approach for Improving SOAP Processing Performance, 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05), pp. 472–479 (2005).

[7] Matthew J. Duftler, Nirmal K. Mukhi, A. S. and Weerawarana, S.: Web Services Invocation Framework (WSIF), Proceedings of the OOPSLA Workshop on Object-Oriented Web Services (2001).

[8] Grosso, P. et al.: XPointer Framework, W3C Recommendation (2003). http://www.w3.org/TR/xptr-framework/.

[9] Suryanarayana, L. et al.: Profiles for the Situated Web, Proceedings of the 11th International World Wide Web Conference (WWW 2002), pp. 200–209 (2002).

[10] Markus Keidl, A. K.: Towards Context-Aware Adaptable Web Services, Proceedings of the 13th International World Wide Web Conference (WWW 2004), pp. 55–65 (2004).

付録

A.1 メインプログラム (SiWS) のソースコード

コンテキスト依存な Web サービスのアーキテクチャを SiWS と名づけて実装した.

A.1.1 Commons.java

```
/**
 * $Id: Commons.java 187 2005-12-04 02:44:37Z ikuom $
 */
package jp.ac.kyoto_u.i.siws;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.URL;
import java.util.Properties;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import jp.ac.kyoto_u.i.siws.client.SiWSClientException;
import jp.ac.kyoto_u.i.siws.server.SiWSServerException;
import jp.ac.kyoto_u.i.siws.xml.conf.Config;

import org.apache.axis.AxisFault;
import org.apache.axis.components.logger.LogFactory;
import org.apache.commons.logging.Log;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

/**
 * 共通のプロパティなどを提供するクラス
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class Commons {
  protected static Log log =
        LogFactory.getLog(Commons.class.getName());
```

```
private static Properties prop = null;
private static Config conf = null;
private static boolean isPropertiesLoaded = false;

/**
 * プロパティファイルからロードした設定を取得します.
 * null を返すことはありません.
 * @return ファイルからロードした設定
 */
public static Properties getProp() {
  if(isPropertiesLoaded) {
    // すでにファイルがロードされている場合
    return prop;
  }

  // まだファイルがロードされていない場合
  final String FN = "siws.properties";
  File prop_file = new File("../" + FN);

  prop = new Properties();
  try {
    // 親ディレクトリにファイルがなければ子ディレクトリを参照
    if(!prop_file.exists())  prop_file = new File(FN);
    prop.load(new FileInputStream(prop_file));
  } catch (FileNotFoundException e) {
    // プロパティファイルが存在しなかった場合
    log.error(Messages.getMessage("err-no-propfile", prop_file.toString()));
  } catch (IOException e) {
    throw new SiWSServerException(
        Messages.getMessage("err-propfile", prop_file.toString()),
        e);
  }
  isPropertiesLoaded = true;
  return prop;
}

/**
 * siws-config.xml に基づく設定オブジェクトを取得します.
 * 初回の取得時にのみ XML ファイルからロードを行います.
 * @return 設定オブジェクト
 */
public static Config getConfig() throws AxisFault {
  final String filename = "siws-config.xml";
  if(conf == null) conf = loadXML(filename, Constants.PKGNAME_CONF);
  return conf;
```

```
}

/**
 * 指定した XML ファイルから指定されたパッケージのオブジェクトをロードします.
 * @param <T> ロードされた結果の型
 * @param filename ロードする XML ファイル
 * @param pkgname ロード先のオブジェクトのパッケージ名
 * @return XML からロードされたオブジェクト
 * @throws SiWSClientException ロード中に何らかの例外が発生した場合
 */
@SuppressWarnings("unchecked")
public static <T> T loadXML(String filename, String pkgname)
  throws AxisFault {
  T ret = null;
  try {
    Unmarshaller um = getUnmarshaller(pkgname);
    //URL url = Commons.class.getResource(".");
    URL url = Commons.class.getClassLoader().getResource(".");
    //String pkg_dir = "../../../../../../";
    //String path = url.getFile() + pkg_dir + "../../";
    String path = url.getFile() + "../../";
    File file = new File(path + filename);
    ret = (T) um.unmarshal(file);
  } catch (Exception e) {
    throw new AxisFault(Messages.getMessage("err-0002"), e);
  }
  return ret;
}

/**
 * 指定した XML ノードから指定されたパッケージのオブジェクトをロードします.
 * @param <T> ロードされた結果の型
 * @param node ロードする XML ノード
 * @param pkgname ロード先のオブジェクトのパッケージ名
 * @return ロードされたオブジェクト
 * @throws SiWSClientException ロード中に何らかの例外が発生した場合
 */
@SuppressWarnings("unchecked")
public static <T> T loadXML(Node node, String pkgname)
  throws SiWSClientException {
  T ret = null;
  try {
    ret = (T) getUnmarshaller(pkgname).unmarshal(node);
  } catch (Exception e) {
    throw new SiWSClientException(Messages.getMessage("err-0002"), e);
```

```
  }
  return ret;
}

private static Unmarshaller getUnmarshaller(String pkgname) throws JAXBException {
  JAXBContext context = JAXBContext.newInstance(pkgname);
  Unmarshaller um = context.createUnmarshaller();
  return um;
}

/**
 * 指定したオブジェクトをXMLの{@link Node}オブジェクトとしてマーシャリングします.

 * @param obj マーシャリング元のオブジェクト
 * @param pkgname JAXBによって作成されたパッケージ名
 * @return マーシャリングされたノード
 * @throws AxisFault マーシャリングが失敗した場合
 */
public static Node getXMLAsNode(Object obj, String pkgname) throws AxisFault {
  // ノードオブジェクトの取得
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  Node node = null;
  try {
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.newDocument();
    node = doc.createElement("_temp");

    // XMLのシリアライズ
    Marshaller m = getMarshaller(pkgname);
    m.marshal(obj, node);
    node = node.getFirstChild();
  } catch (Exception e) {
    throw new AxisFault(Messages.getMessage("err-0001"), e);
  }

  return node;
}

private static Marshaller getMarshaller(String pkgname) throws JAXBException {
  JAXBContext context = JAXBContext.newInstance(pkgname);
  Marshaller m = context.createMarshaller();
  return m;
}

/**
```

```
   * デバグ用
   * @param args
   */
  public static void main(String[] args) throws Exception {
    log.info("info");
    log.debug("debug");
    System.out.println(getConfig().getClient().getProvider());
  }
}
```

A.1.2 Constants.java

```
/**
 * $Id: Constants.java 155 2005-11-26 12:12:40Z ikuom $
 */
package jp.ac.kyoto_u.i.siws;

/**
 * 定数を格納するクラス
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class Constants {
    /**
     * JAXB で生成された SOAP 用パッケージの名前
     */
    public static final String PKGNAME_SOAP = "jp.ac.kyoto_u.i.siws.xml.soap";

    /**
     * JAXB で生成された Config 用パッケージの名前
     */
    public static final String PKGNAME_CONF = "jp.ac.kyoto_u.i.siws.xml.conf";

    /**
     * SOAP ヘッダで処理するタグの名前空間
     */
    public static final String HEADER_NS = "http://i.kyoto-u.ac.jp/siws/soap/";

    /**
     * SOAP ヘッダで処理するリクエストのタグ
     */
    public static final String HEADER_REQ = "clientContext";

    /**
     * SOAP ヘッダで処理するレスポンスのタグ
     */
```

```
    public static final String HEADER_RES = "bypassInfo";
}
```

A.1.3 OperationUsage.java

```
/**
 * $Id: OperationUsage.java 185 2005-12-02 04:54:57Z ikuom $
 */
package jp.ac.kyoto_u.i.siws;

import java.math.BigInteger;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.xml.bind.JAXBException;

import jp.ac.kyoto_u.i.siws.xml.soap.ObjectFactory;
import jp.ac.kyoto_u.i.siws.xml.soap.OperUsage;

import org.apache.axis.AxisFault;
import org.apache.axis.components.logger.LogFactory;
import org.apache.commons.logging.Log;

/**
 * オペレーション用法を表すクラス
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class OperationUsage {
    protected static Log log = LogFactory.getLog(OperationUsage.class.getName());
    private final Map<Integer, Object> staticParams = new HashMap<Integer, Object>();
    private Class returnClass = null;
    private OperUsage _core = null;

    /**
     * 返り値のクラスを指定してインスタンスを作成します.
     * パラメータの個数は静的パラメータと動的パラメータをあわせた数です.
     * @param returnClass 返り値のクラス
     */
    public OperationUsage(Class returnClass) {
        this.returnClass = returnClass;
    }
```

```
/**
 * XML からアンマーシャリングしたオブジェクト (core)
 * とパラメータリストを元にインスタンスを作成します.
 * @param core XML からアンマーシャリングしたオブジェクト
 * @param params パラメータリスト
 */
public OperationUsage(OperUsage core, Object[] params) throws AxisFault {
    try {
        this._core = core;

        for(Object obj: core.getStaticIndex()) {
            int i = ((BigInteger)obj).intValue();
            staticParams.put(i, params[i]);
        }
    } catch(Exception e) {
        throw AxisFault.makeFault(e);
    }
}

/**
 * 指定したインデックスの静的パラメータをセットします.
 * @param index オペレーション内のパラメータインデックス
 * @param param パラメータ
 */
public void setStaticParam(int index, Object param) {
    staticParams.put(index, param);
}

/**
 * 指定したインデックスの静的パラメータのセットを解除します.
 * @param index オペレーション内のパラメータインデックス
 */
public void unsetStaticParam(int index) {
    staticParams.remove(index);
}

/**
 * 静的パラメータの配列に，指定された動的パラメータをマージしたものを返します.
 * @param dynParams 動的パラメータ配列
 * @return マージされたパラメータ配列
 * @throws AxisFault
 */
public Object[] mergeParams(Object[] dynParams) throws AxisFault {
    Object[] ret = new Object[staticParams.size() + dynParams.length];
    Arrays.fill(ret, null);
```

```
    boolean[] isStatic = new boolean[ret.length];
    Arrays.fill(isStatic, false);

    // 静的パラメータのセット
    for(Map.Entry<Integer, Object> entry: staticParams.entrySet()) {
        isStatic[entry.getKey()] = true;
        ret[entry.getKey()] = entry.getValue();
    }

    // 動的パラメータのセット
    int dynIndex = 0;
    for (int i = 0; i < isStatic.length; i++) {
        try {
            if(!isStatic[i]) {
                ret[i] = dynParams[dynIndex];
                dynIndex++;
            }
        } catch(ArrayIndexOutOfBoundsException e) {
            log.error(Messages.getMessage("err-0004"), e);
            throw new AxisFault(Messages.getMessage("err-0004"), e);
        }
    }
    if(dynIndex < dynParams.length)
        log.warn(Messages.getMessage(
            "warn-0003",
            Integer.toString(dynIndex),
            Integer.toString(dynParams.length)));

    return ret;
}

/**
 * @return Returns the returnClass.
 */
public Class getReturnClass() {
    return returnClass;
}

/**
 * XML バインディング用のオブジェクトを取得します.
 * @return XML バインディング用のオブジェクト
 * @throws JAXBException 作成中に JAXB で例外が発生した場合
 */
@SuppressWarnings("unchecked")
public OperUsage getCore() throws JAXBException {
```

```
        if(_core != null) return _core;

        ObjectFactory factory = new ObjectFactory();
        OperUsage usage = factory.createOperUsage();
        List<BigInteger> list = usage.getStaticIndex();
        for(Map.Entry<Integer, Object> entry: staticParams.entrySet()) {
            list.add(BigInteger.valueOf(entry.getKey()));
        }

        return usage;
    }
}
```

A.1.4 SituatedProxy.java

```
/**
 * $Id: SituatedProxy.java 189 2005-12-04 07:07:43Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.client;

import java.rmi.RemoteException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

import org.apache.axis.AxisFault;
import org.apache.axis.Message;
import org.apache.axis.client.Call;
import org.apache.axis.client.Stub;
import org.apache.axis.components.logger.LogFactory;
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.message.SOAPHeaderElement;
import org.apache.commons.logging.Log;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import jp.ac.kyoto_u.i.siws.*;
import jp.ac.kyoto_u.i.siws.client.provider.ClientBypass;
import jp.ac.kyoto_u.i.siws.client.provider.ClientProvider;
import jp.ac.kyoto_u.i.siws.xml.soap.*;

/**
```

```
 * コンテキスト依存プロキシクラス.
 * <br />
 * ハンドシェイク時には{@link org.apache.axis.client.Call}へのプロキシとして動作
します.
 * バイパス通信時には{@link jp.ac.kyoto_u.i.siws.OperationUsage}への
 * プロキシとして動作します.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class SituatedProxy<T> {
    protected static Log log =
        LogFactory.getLog(SituatedProxy.class.getName());
    private Stub _stub = null;
    private Call _call = null;
    private OperationUsage _usage = null;
    private ClientBypass<T> bypass = null;
    private boolean noBypass = false;

    /**
     * Axis の{@link Stub}オブジェクト，Axis の{@link Call}オブジェクト，
     * オペレーションの使い方を指定してインスタンスを作成します.
     * @param stub スタブオブジェクト
     * @param call 呼び出しオブジェクト
     * @param usage オペレーションの使い方
     */
    public SituatedProxy(Stub stub, Call call, OperationUsage usage) {
        this._stub = stub;
        this._call = call;
        this._usage = usage;
    }

    /**
     * パラメータを指定してサービスを呼び出します．<br />
     * バイパスが作成されていない場合は，コンテキストを送信しサーバからバイパス情報
を受信した後バイパスを作成します.
     * バイパスがすでに作成されている場合はバイパスを用いてサービスを呼び出します.
     *
     * @param params 呼び出しのパラメータ
     * @return 呼び出しの結果
     * @throws RemoteException 内部にもつ{@link org.apache.axis.client.Call}イ
ンスタンスの{@link org.apache.axis.client.Call#invoke()}が失敗した場合
     */
    public T invoke(Object[] params) throws RemoteException {
        if(bypass != null && !noBypass) {
            // バイパスがすでに作成されている場合
```

```
        log.debug(Messages.getMessage("log-0002", bypass.getClass().getName()));
        return bypass.invoke(params);
    }

    // バイパスが存在しない場合
    List<ClientProvider> list = ClientProvider.getAvailables();
    if(list.size() == 0) {
        // 利用できるプロバイダが一つも存在しない場合
        log.debug(Messages.getMessage("log-0000"));
        return invokeCore(params);
    } else {
        // 利用できるプロバイダが一つ以上存在する場合
        return invokeWithHandShake(params);
    }
}

/**
 * _call オブジェクトに対する呼び出しを行います.
 * @param params
 * @return
 * @throws RemoteException
 */
@SuppressWarnings("unchecked")
private T invokeCore(Object[] params) throws RemoteException {
    log.debug(Messages.getMessage("log-enter"));

    /*
     * Axis 1.2.1 の WSDL2Java で生成されたスタブのメソッドから
     * _call を操作するソースコードを移動
     *
     * ==begin
     */
    Object _resp = null;
    T res = null;
    try {
        _resp = _call.invoke(_usage.mergeParams(params));
        if (_resp instanceof java.rmi.RemoteException) {
            throw (java.rmi.RemoteException)_resp;
        }
        else {
            _stub.extractAttachments(_call);
            try {
                //return (java.lang.String) _resp;
                res = (T) _resp;
            } catch (java.lang.Exception _exception) {
```

```
                // return (java.lang.String) org.apache.axis.utils.JavaUtils.convert(_resp,
                res = (T) org.apache.axis.utils.JavaUtils.convert(
                        _resp, _usage.getReturnClass());
            }
        }
    } catch (org.apache.axis.AxisFault axisFaultException) {
        throw axisFaultException;
    }
      /*
       * ==end
       */

    log.debug(Messages.getMessage("log-exit"));
    return res;
}

/**
 * クライアントコンテキストとバイパス情報のやり取りを含む呼び出しを行います.
 *
 * @param params 呼び出しのパラメータ
 * @return 呼び出しの結果
 * @throws RemoteException {@link Call#invoke()}が失敗した場合
 */
@SuppressWarnings("unchecked")
private T invokeWithHandShake(Object[] params) throws RemoteException {
    log.debug(Messages.getMessage("log-enter"));
    setSoapHeader(_call);

    T res = invokeCore(params);

    // _call からバイパス情報を取得して処理する
    /* エンベロープからエレメントの取得 */
    Message msg = _call.getResponseMessage();
    if (msg == null) throw AxisFault.makeFault(new RuntimeException("msg is null."));
    SOAPEnvelope env = msg.getSOAPEnvelope();
    SOAPHeaderElement header = env.getHeaderByName(
            Constants.HEADER_NS, Constants.HEADER_RES, true);

    if(null != header) {
        // バイパス情報のエレメントが見つかった場合
        log.debug(Messages.getMessage("log-0007", Constants.HEADER_RES));
        try {
            /* バイパス情報のアンマーシャリング */
            BypassInfo info = Commons.loadXML(header, Constants.PKGNAME_SOAP);
            if(info == null) return noBypass(res, 0);
```

```
            String pName = info.getProvider();
            if(pName == null) return noBypass(res, 1);
            ClientProvider p = ClientProvider.getByName(pName);
            if(p == null) return noBypass(res, 2);

            /* パラメータマップの作成: map */
            List<Param> list = info.getParam();
            Map<String, String> map = new HashMap<String, String>();
            for(Param param: list) {
                map.put(param.getKey(), param.getValue());
            }

            ClientBypass bypass =  p.createBypass(map);
            if(bypass == null) return noBypass(res, 3);
            this.bypass = bypass;
        } catch(Exception e) {
            throw AxisFault.makeFault(e);
        }
    }

    log.debug(Messages.getMessage("log-exit"));
    return res;
}

private T noBypass(T res, int code) {
    log.info(Messages.getMessage("info-0000", Integer.toString(code)));
    noBypass = true;
    return res;
}

/**
 * Call オブジェクトに SiWS 用のヘッダを追加する.
 * @param _call 追加対象の Call オブジェクト
 * @throws AxisFault
 */
private void setSoapHeader(Call _call) throws AxisFault {
    try {
        /* マーシャル */
        Node node = Commons.getXMLAsNode(getClientContext(), Constants.PKGNAME_SOAP);

        SOAPHeaderElement elem =
            new SOAPHeaderElement(
                    (Element) node
            );
        _call.addHeader(elem);
```

```
    }catch(Exception e) {
        throw AxisFault.makeFault(e);
    }
}


/**
 * デバグ用
 * @param args
 * @throws Exception
 */
public static void main(String[] args) throws Exception {
    /* マーシャル */
    JAXBContext context = JAXBContext.newInstance(Constants.PKGNAME_SOAP);
    Marshaller m = context.createMarshaller();
    //m.marshal(getClientContext(), System.out);

    System.out.println("hoge");
}

@SuppressWarnings("unchecked")
private ClientContext getClientContext() throws AxisFault {
    // モデルの作成
    ObjectFactory factory = new ObjectFactory();
    ClientContext cc = null;
    try {
        /* AvailableProviders の作成 */
        AvailableProviders providers = factory.createAvailableProviders();
        ClientProvider.storeAvailables(providers.getProvider());

        /* StaticParameters の作成 */
        OperUsage usage = _usage.getCore();

        /* ClientContext の作成 */
        cc = factory.createClientContext();
        cc.setAvailableProviders(providers);
        cc.setOperationUsage(usage);
    } catch(JAXBException e) {
        throw AxisFault.makeFault(e);
    }

    return cc;
}

/**
```

```
     * @return bypass を戻します。
     */
    public ClientBypass<T> getBypass() {
        return bypass;
    }

}
```

A.1.5 SiWSClientException.java

```
/**
 * $Id: SiWSClientException.java 156 2005-11-26 12:51:23Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.client;

/**
 * SiWS のクライアント側で発生する例外.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class SiWSClientException extends RuntimeException {
    /**
     *
     */
    private static final long serialVersionUID = -8131347898334134219L;

    /**
     * 詳細メッセージを指定してインスタンスを作成します.
     * @param message 詳細メッセージ
     */
    public SiWSClientException(String message) {
        super(message);
    }

    /**
     * 原因を指定してインスタンスを作成します.
     * @param cause 原因
     */
    public SiWSClientException(Throwable cause) {
        super(cause);
    }

    /**
     * 詳細メッセージと原因を指定してインスタンスを作成します.
     * @param message 詳細メッセージ
     * @param cause 原因
```

```
     */
    public SiWSClientException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

A.1.6 ClientBypass.java

```
/**
 * $Id: ClientBypass.java 188 2005-12-04 04:24:06Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.client.provider;

import java.rmi.RemoteException;

/**
 * クライアント側のバイパスを表すクラス.
 * {@link jp.ac.kyoto_u.i.siws.client.provider.ClientProvider}
 * が{@link ClientBypass}のファクトリとして機能します.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 * @param <T> {@link #invoke(Object[])}メソッドの戻り値の型
 */
public abstract class ClientBypass <T> {

    /**
     * このバイパスを通してオペレーションを実行します.
     * @param params オペレーションの引数
     * @return 実行結果
     * @throws RemoteException
     */
    public abstract T invoke(Object[] params) throws RemoteException;

    /**
     * このバイパスを破棄します.
     * @throws RemoteException
     */
    public abstract void tearDown() throws RemoteException;

    /**
     * このバイパスを作成したクライアントプロバイダを取得します.
     * @throws RemoteException
     */
    public abstract ClientProvider getClientProvider() throws RemoteException;
}
```

A.1.7 ClientProvider.java

```
/**
 * $Id:$
 */
package jp.ac.kyoto_u.i.siws.client.provider;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.axis.AxisFault;
import org.apache.axis.components.logger.LogFactory;
import org.apache.commons.logging.Log;

import jp.ac.kyoto_u.i.siws.Commons;
import jp.ac.kyoto_u.i.siws.Messages;
import jp.ac.kyoto_u.i.siws.client.SiWSClientException;
import jp.ac.kyoto_u.i.siws.client.provider.ClientBypass;
import jp.ac.kyoto_u.i.siws.xml.conf.*;

/**
 * クライアントのプロバイダ．シングルトン．
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public abstract class ClientProvider {
    private static final Map<String, ClientProvider> providerMap =
        new HashMap<String, ClientProvider>();
    private static boolean isLoaded = false;
    protected static Log log = LogFactory.getLog(ClientProvider.class.getName());

    protected boolean isEnabled = true;

    /**
     * バイパスオブジェクトを作成します．{@link ClientProvider}を Creator，
     * {@link ClientBypass}を Product とするファクトリメソッドです．
     * @return バイパス
     * @param paramMap バイパス作成に必要なパラメータのマップ
     */
    public abstract ClientBypass createBypass(Map<String, String> paramMap)
        throws AxisFault;

    /**
```

```
 * クライアント環境を検査してプロバイダが利用可能かどうかを返します.
 *
 * @return 利用可能か否か.
 */
public boolean isAvailable() {
    return isEnabled;
}

/**
 * プロバイダの名前を取得します.
 *
 */
public abstract String getName();

/**
 * インストールされたプロバイダのマップを返します.
 * プロバイダは siws.properties ファイルの
 * client.providers プロパティに基づいてインストールされます.
 *
 * @return インストールされたプロバイダのマップ
 */
public static Map<String, ClientProvider> getAll() throws AxisFault {
    if(!isLoaded) load();
    return providerMap;
}

/**
 * インストールされたプロバイダのうち,
 * 現在のクライアント環境を検査した上で利用可能なもののリストを返します.
 *
 * @return 利用可能なプロバイダのリスト
 */
public static List<ClientProvider> getAvailables() throws AxisFault {
    final List<ClientProvider> list =
        new ArrayList<ClientProvider>(providerMap.size());
    for (Map.Entry<String, ClientProvider> entry: getAll().entrySet()) {
        ClientProvider provider = entry.getValue();
        if(provider.isAvailable()) list.add(provider);
    }
    return list;
}

public static ClientProvider getByName(String name) throws AxisFault {
    if(!isLoaded) load();
    return providerMap.get(name);
```

```
}

/**
 * 指定されたリストに利用可能なプロバイダ名を追加します.
 * @param list    プロバイダ名を追加するリスト
 * @throws AxisFault
 */
public static void storeAvailables(List<String> list) throws AxisFault {
    for(ClientProvider provider: getAvailables()) {
        list.add(provider.getName());
    }
}

/**
 * プロバイダのクラスをロードする
 *
 */
@SuppressWarnings("unchecked")
private static void load() throws AxisFault {
    log.debug(Messages.getMessage("log-enter"));
    List<ConfigType.ClientType.ProviderType> providers =
        Commons.getConfig().getClient().getProvider();

    String errMessage = Messages.getMessage("err-client-provider-load");

    // オブジェクトの生成
    for (ConfigType.ClientType.ProviderType provider : providers) {
        String name = provider.getClassName();
        errMessage = Messages.getMessage("err-client-provider-load", name);
        try {
            Class c = Class.forName(name);
            Method m = c.getMethod("instance", new Class[]{});
            ClientProvider p = (ClientProvider)m.invoke(c, new Object[]{});
            if(p == null) throw new NullPointerException(errMessage);
            providerMap.put(p.getName(), p);
            log.info(Messages.getMessage("info-0001", name));
        } catch (Exception e) {
            throw new SiWSClientException(
                errMessage, e);
        }
    }

    isLoaded = true;
    log.debug(Messages.getMessage("log-exit"));
}
```

```
    /**
     * @return isEnabled を戻します。
     */
    public boolean isEnabled() {
        return isEnabled;
    }

    /**
     * @param isEnabled 設定する isEnabled。
     */
    public void setEnabled(boolean isEnabled) {
        this.isEnabled = isEnabled;
    }
}
```

A.1.8 RMIClientBypass.java

```
/**
 * $Id: RMIClientBypass.java 254 2006-01-24 05:17:10Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.client.provider;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.util.Map;

import org.apache.axis.AxisFault;
import org.apache.axis.components.logger.LogFactory;
import org.apache.commons.logging.Log;

import jp.ac.kyoto_u.i.siws.Messages;
import jp.ac.kyoto_u.i.siws.server.provider.rmi.Invoker;
import jp.ac.kyoto_u.i.siws.server.provider.rmi.RMIServerBypass;

/**
 * Java RMI のバイパス.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 * @param <T>
 */
public class RMIClientBypass <T> extends ClientBypass {
    protected static Log log = LogFactory.getLog(RMIClientBypass.class.getName());
    private String rmiName = null;
    private Invoker<T> invoker = null;
```

```
    /**
     * パラメータのマップを指定してインスタンスを作成します.
     * @param params パラメータのマップ
     */
    @SuppressWarnings("unchecked")
    public RMIClientBypass(Map<String, String> params) throws AxisFault {
        final String key = RMIServerBypass.KEY_RMINAME;
        rmiName = params.get(key);
        if(rmiName == null) {
            log.info(Messages.getMessage("info-0003", key));
            throw new AxisFault();
        }
        try {
            invoker = (Invoker<T>)Naming.lookup(rmiName);
        } catch (Exception e) {
            throw AxisFault.makeFault(e);
        }
    }

    @Override
    public T invoke(Object[] params) throws RemoteException {
        log.debug(Messages.getMessage("log-enter"));
        T res = null;
        res = invoker.invoke(params);
        log.debug(Messages.getMessage("log-exit"));
        return res;
    }

    @Override
    public void tearDown() throws RemoteException {
        // TODO 自動生成されたメソッド・スタブ

    }

    @Override
    public ClientProvider getClientProvider() throws RemoteException {
        return RMIClientProvider.instance();
    }

}
```

A.1.9 RMIClientProvider.java

```
/**
 * $Id: RMIClientProvider.java 254 2006-01-24 05:17:10Z ikuom $
 */
```

```
package jp.ac.kyoto_u.i.siws.client.provider;

import java.util.Map;

import org.apache.axis.AxisFault;

import jp.ac.kyoto_u.i.siws.client.provider.ClientBypass;

/**
 * Java RMI のクライアントプロバイダ.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class RMIClientProvider extends ClientProvider {
    private static RMIClientProvider instance = null;

    /**
     * シングルトン. コンストラクタは private.
     *
     */
    private RMIClientProvider() {
        super();
    }

    /**
     * シングルトンのインスタンスを取得する.
     * @return インスタンス.
     */
    public static RMIClientProvider instance() {
        if(null == instance) instance = new RMIClientProvider();
        return instance;
    }


    @Override
    public boolean isAvailable() {
        return super.isAvailable();
    }

    @Override
    public String getName() {
        return "rmi";
    }

    @SuppressWarnings("unchecked")
```

```
    @Override
    public ClientBypass createBypass(Map<String, String> paramMap) throws AxisFault {
        return new RMIClientBypass(paramMap);
    }

}
```

A.1.10 ServantInvoker.java

```
/**
 * $Id: ServantInvoker.java 189 2005-12-04 07:07:43Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server;

import java.lang.reflect.Method;

import org.apache.axis.AxisFault;

import jp.ac.kyoto_u.i.siws.OperationUsage;

/**
 * サーバント（サービスオブジェクト）の実行を表すクラス
 */
public class ServantInvoker {
    private Object servant = null;
    private Method method = null;
    private OperationUsage usage = null;

    /**
     * サーバントとメソッド，オペレーション用法を指定してインスタンスを作成します.
     * @param servant サーバント（サービスオブジェクト）
     * @param method 呼び出すメソッド
     * @param usage 静的パラメータを含むオペレーション用法
     */
    public ServantInvoker(Object servant, Method method, OperationUsage usage) {
        this.servant = servant;
        this.method = method;
        this.usage = usage;
    }

    /**
     * 指定した差分パラメータを用いてサーバントを呼び出します.
     * @param params 差分パラメータ
     * @return 呼び出し結果
     * @throws AxisFault
     */
```

```
    public Object invoke(Object[] params) throws AxisFault {
        Object res = null;
        try {
            res = method.invoke(servant, usage.mergeParams(params));
        } catch(Exception e) {
            throw AxisFault.makeFault(e);
        }
        return res;
    }
}
```

A.1.11 SiWSContextHandler.java

```
/**
 * $Id: SiWSContextHandler.java 188 2005-12-04 04:24:06Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

import javax.xml.namespace.QName;
import javax.xml.rpc.holders.Holder;
import javax.xml.rpc.holders.IntHolder;

import jp.ac.kyoto_u.i.siws.*;
import jp.ac.kyoto_u.i.siws.Constants;
import jp.ac.kyoto_u.i.siws.server.provider.ServerBypass;
import jp.ac.kyoto_u.i.siws.server.provider.ServerProvider;
import jp.ac.kyoto_u.i.siws.xml.soap.ClientContext;
import jp.ac.kyoto_u.i.siws.xml.soap.OperUsage;

import org.apache.axis.*;
import org.apache.axis.components.logger.LogFactory;
import org.apache.axis.constants.Style;
import org.apache.axis.description.OperationDesc;
import org.apache.axis.description.ParameterDesc;
import org.apache.axis.description.ServiceDesc;
import org.apache.axis.handlers.BasicHandler;
import org.apache.axis.handlers.soap.SOAPService;
import org.apache.axis.message.*;
import org.apache.axis.providers.java.JavaProvider;
import org.apache.axis.soap.SOAPConstants;
```

```
import org.apache.axis.utils.JavaUtils;
import org.apache.commons.logging.Log;
import org.xml.sax.SAXException;

/**
 * SiWS 特有の SOAP ヘッダメッセージを処理する Axis のハンドラ
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class SiWSContextHandler extends BasicHandler {
    /**
     *
     */
    private static final long serialVersionUID = 6137313478820523415L;

    private static Log log =
        LogFactory.getLog(SiWSContextHandler.class.getName());

    /**
     * Axis のメッセージコンテキストに登録するバイパスプロバイダのキー
     */
    public static final String SIWS_BPPROVIDER = "SiWS.BypassProvider";

    /**
     * Axis のメッセージコンテキストに登録するオペレーション用法のキー
     */
    public static final String SIWS_OPERUSAGE = "SiWS.OperUsage";


    /**
     * メッセージコンテキストがハンドラを通過する際に呼ばれるメソッド.
     * @param context Axis のメッセージコンテキスト
     */
    public void invoke(MessageContext context) throws AxisFault {
        log.debug("enter-method: invoke");
        /* Pivot を通過しているか否か */
        if(context.getPastPivot()) handleResponse(context);
        else handleRequest(context);
        log.debug("exit-method: invoke");
    }

    /**
     * リクエスト時の処理
     * @param context SOAP のメッセージコンテキスト
     * @throws AxisFault
```

```
 */
@SuppressWarnings("unchecked")
private void handleRequest(MessageContext context) throws AxisFault {
    log.debug("enter-method: handleRequest");
    Message msg = context.getRequestMessage();
    if (msg == null) throw new AxisFault(Messages.getMessage("noRequest00"));

    /* ヘッダからコンテキストの取得 */
    SOAPEnvelope env = msg.getSOAPEnvelope();
    SOAPHeaderElement header = env.getHeaderByName(
            Constants.HEADER_NS, Constants.HEADER_REQ, true);

    if (header != null) {
        log.debug(Messages.getMessage("log-0008", Constants.HEADER_REQ));
        // 処理すべき SiWS のヘッダを発見した場合

        // コンテキストのアンマーシャリング
        ClientContext cc = Commons.loadXML(header, Constants.PKGNAME_SOAP);

        // 利用可能プロトコルの取得とバイパス用プロトコルの選択
        List<String> pNames = cc.getAvailableProviders().getProvider();
        ServerProvider sp =    ServerProvider.selectForBypass(pNames);
        if(sp == null) log.warn(Messages.getMessage("warn-0000", concatString(pNames)));
        else log.debug(Messages.getMessage("log-0006", sp.getName()));

        // オペレーション用法の取得
        log.debug("To get operation usage.");
        OperUsage core = cc.getOperationUsage();
        if(core == null) throw AxisFault.makeFault(
                new NullPointerException(Messages.getMessage("err-0003")));

        context.setProperty(SIWS_OPERUSAGE, core);
        context.setProperty(SIWS_BPPROVIDER, sp);
        header.setProcessed(true);
    }
    log.debug("exit-method: handleRequest");
}

/**
 * レスポンス時の処理
 * @param context SOAP のメッセージコンテキスト
 * @throws AxisFault
 */
private void handleResponse(MessageContext context) throws AxisFault {
    log.debug("enter-method: handleResponse");
```

```
// メッセージコンテキストからバイパスに用いるサーバプロバイダを取得
ServerProvider sp = (ServerProvider)context.getProperty(SIWS_BPPROVIDER);
OperUsage core = (OperUsage)context.getProperty(SIWS_OPERUSAGE);

if(sp == null || core == null) return;


/*
 * サービスハンドラを作成し，それを元にバイパスを作成.
 */
ServerBypass bypass = null;
try {
    /*
     * メッセージコンテキストからサーバントと引数の情報を取得.
     * 取得した情報を元にサービスハンドラを作成.
     */
    ServantInvoker invoker = null;
    {
        // サービスオブジェクトの取得: serviceObject
        Handler service = context.getService();
        String clsName = (String)service.getOption(JavaProvider.OPTION_CLASSNAME);
        IntHolder scope   = new IntHolder();
        JavaProvider jp = (JavaProvider)((TargetedChain)service).getPivotHandler();
        Object serviceObject = jp.getServiceObject(context, service, clsName, scope);

        // メソッドの取得: method
        Method method = context.getOperation().getMethod();

        // 引数の取得: argValues
        Object[] argValues = getParams(context);

        // サービスハンドラの作成
        OperationUsage usage = new OperationUsage(core, argValues);
        invoker = new ServantInvoker(serviceObject, method, usage);
    }
    bypass = sp.createBypass(invoker);
} catch (Exception e) {
    throw AxisFault.makeFault(e);
}

// ヘッダ要素の作成
// TODO 各プロバイダの構造を反映したバイパス情報をセットするように変更
//SOAPHeaderElement header = new SOAPHeaderElement(
//        Constants.HEADER_NS, Constants.HEADER_RES, sp.getName());
```

```
        SOAPHeaderElement header = new SOAPHeaderElement(bypass.infoAxXML());

        // レスポンスメッセージにヘッダをセット
        Message msg = context.getResponseMessage();
        if (msg == null)
            throw AxisFault.makeFault(new RuntimeException("msg is null."));
        SOAPEnvelope env = msg.getSOAPEnvelope();
        env.addHeader(header);

        log.debug("exit-method: handleResponse");
    }

    private Object[] getParams(MessageContext msgContext) throws Exception {
        /*
         * 以下のコードは Axis 1.2.1 の RPCProvider.java からのコピーを含む
         * == begin
         */
        SOAPService service = msgContext.getService();
        ServiceDesc serviceDesc = service.getServiceDescription();
        OperationDesc operation = msgContext.getOperation();
        Vector bodies = msgContext.getRequestMessage().getSOAPEnvelope().getBodyElements();

        RPCElement body = null;

        // Find the first "root" body element, which is the RPC call.
        for (int bNum = 0; body == null && bNum < bodies.size(); bNum++) {
            // If this is a regular old SOAPBodyElement, and it's a root,
            // we're probably a non-wrapped doc/lit service.  In this case,
            // we deserialize the element, and create an RPCElement "wrapper"
            // around it which points to the correct method.
            // FIXME : There should be a cleaner way to do this...
            if (!(bodies.get(bNum) instanceof RPCElement)) {
                SOAPBodyElement bodyEl = (SOAPBodyElement) bodies.get(bNum);
                // igors: better check if bodyEl.getID() != null
                // to make sure this loop does not step on SOAP-ENC objects
                // that follow the parameters! FIXME?
                if (bodyEl.isRoot() && operation != null && bodyEl.getID() == null) {
                    ParameterDesc param = operation.getParameter(bNum);
                    // at least do not step on non-existent parameters!
                    if (param != null) {
                        Object val = bodyEl.getValueAsType(param.getTypeQName());
                        body = new RPCElement("",
                                              operation.getName(),
                                              new Object[]{val});
                    }
```

```
        }
    } else {
        body = (RPCElement) bodies.get(bNum);
    }
}

// special case code for a document style operation with no
// arguments (which is a strange thing to have, but whatever)
if (body == null) {
    // throw an error if this isn't a document style service
    if (!(serviceDesc.getStyle().equals(Style.DOCUMENT))) {
        throw new Exception(Messages.getMessage("noBody00"));
    }

    // look for a method in the service that has no arguments,
    // use the first one we find.
    ArrayList ops = serviceDesc.getOperations();
    for (Iterator iterator = ops.iterator(); iterator.hasNext();) {
        OperationDesc desc = (OperationDesc) iterator.next();
        if (desc.getNumInParams() == 0) {
            // found one with no parameters, use it
            /* msgContext.setOperation(desc); */
            // create an empty element
            body = new RPCElement(desc.getName());
            // stop looking
            break;
        }
    }

    // If we still didn't find anything, report no body error.
    if (body == null) {
        throw new Exception(Messages.getMessage("noBody00"));
    }
}

String methodName = body.getMethodName();
Vector args = null;
try {
    args = body.getParams();
} catch (SAXException e) {
    if(e.getException() != null)
        throw e.getException();
    throw e;
}
int numArgs = args.size();
```

```
// This may have changed, so get it again...
// FIXME (there should be a cleaner way to do this)
//operation = msgContext.getOperation();

if (operation == null) {
    QName qname = new QName(body.getNamespaceURI(),
            body.getName());
    operation = serviceDesc.getOperationByElementQName(qname);

if (operation == null) {
    SOAPConstants soapConstants = msgContext == null ?
            SOAPConstants.SOAP11_CONSTANTS :
            msgContext.getSOAPConstants();
    if (soapConstants == SOAPConstants.SOAP12_CONSTANTS) {
        AxisFault fault =
                new AxisFault(org.apache.axis.Constants.FAULT_SOAP12_SENDER,
                              Messages.getMessage("noSuchOperation",
                                                  methodName),
                              null,
                              null);
        fault.addFaultSubCode(org.apache.axis.Constants.FAULT_SUBCODE_PROC_NOT_PRESENT);
        throw new SAXException(fault);
    } else {
        throw new AxisFault(org.apache.axis.Constants.FAULT_CLIENT, Messages.getMessage(
                null, null);
    }
    } else {
         msgContext.setOperation(operation);
    }
}

// Create the array we'll use to hold the actual parameter
// values.  We know how big to make it from the metadata.
Object[] argValues = new Object[operation.getNumParams()];

// A place to keep track of the out params (INOUTs and OUTs)
ArrayList outs = new ArrayList();

// Put the values contained in the RPCParams into an array
// suitable for passing to java.lang.reflect.Method.invoke()
// Make sure we respect parameter ordering if we know about it
// from metadata, and handle whatever conversions are necessary
// (values -> Holders, etc)
for (int i = 0; i < numArgs; i++) {
```

```
    RPCParam rpcParam = (RPCParam) args.get(i);
    Object value = rpcParam.getObjectValue();

    // first check the type on the paramter
    ParameterDesc paramDesc = rpcParam.getParamDesc();

    // if we found some type info try to make sure the value type is
    // correct.  For instance, if we deserialized a xsd:dateTime in
    // to a Calendar and the service takes a Date, we need to convert
    if (paramDesc != null && paramDesc.getJavaType() != null) {

        // Get the type in the signature (java type or its holder)
        Class sigType = paramDesc.getJavaType();

        // Convert the value into the expected type in the signature
        value = JavaUtils.convert(value, sigType);

        rpcParam.setObjectValue(value);
        if (paramDesc.getMode() == ParameterDesc.INOUT) {
            outs.add(rpcParam);
        }
    }

    // Put the value (possibly converted) in the argument array
    // make sure to use the parameter order if we have it
    if (paramDesc == null || paramDesc.getOrder() == -1) {
        argValues[i] = value;
    } else {
        argValues[paramDesc.getOrder()] = value;
    }

    if (log.isDebugEnabled()) {
        log.debug("  " + Messages.getMessage("value00",
                "" + argValues[i]));
    }
}

// See if any subclasses want a crack at faulting on a bad operation
// FIXME : Does this make sense here???
/*String allowedMethods = (String) service.getOption("allowedMethods");
checkMethodName(msgContext, allowedMethods, operation.getName());*/

// Now create any out holders we need to pass in

int count = numArgs;
```

```
    for (int i = 0; i < argValues.length; i++) {

        // We are interested only in OUT/INOUT
        ParameterDesc param = operation.getParameter(i);
        if(param.getMode() == ParameterDesc.IN)
            continue;

        Class holderClass = param.getJavaType();
        if (holderClass != null &&
                Holder.class.isAssignableFrom(holderClass)) {
            int index = count;
            // Use the parameter order if specified or just stick them to the end.
            if (param.getOrder() != -1) {
                index = param.getOrder();
            } else {
                count++;
            }
            // If it's already filled, don't muck with it
            if (argValues[index] != null) {
                continue;
            }
            argValues[index] = holderClass.newInstance();
            // Store an RPCParam in the outs collection so we
            // have an easy and consistent way to write these
            // back to the client below
            RPCParam p = new RPCParam(param.getQName(),
                    argValues[index]);
            p.setParamDesc(param);
            outs.add(p);
        } else {
            throw new AxisFault(Messages.getMessage("badOutParameter00",
                    "" + param.getQName(),
                    operation.getName()));
        }
    }
    /*
     * == end
     */

    return argValues;
}

private String concatString(List<String> list) {
    StringBuffer sb = new StringBuffer();
    String sep = ", ";
```

```
        boolean isFirst = true;
        for (String str: list) {
            if(isFirst) isFirst = false;
            else sb.append(sep);
            sb.append(str);
        }
        return sb.toString();
    }
}
```

A.1.12 SiWSServerException.java

```
/**
 * $Id: SiWSServerException.java 156 2005-11-26 12:51:23Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server;

/**
 * SiWS のサーバ側で発生する例外.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class SiWSServerException extends RuntimeException {
    /**
     *
     */
    private static final long serialVersionUID = 7738298771069991739L;

    /**
     * 詳細メッセージを指定してインスタンスを作成します.
     * @param message 詳細メッセージ
     */
    public SiWSServerException(String message) {
        super(message);
    }

    /**
     * 原因を指定してインスタンスを作成します.
     * @param cause 原因
     */
    public SiWSServerException(Throwable cause) {
        super(cause);
    }

    /**
     * 詳細メッセージと原因を指定してインスタンスを作成します.
```

```
     * @param message 詳細メッセージ
     * @param cause 原因
     */
    public SiWSServerException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

A.1.13 ServerBypass.java

```
/**
 * $Id: ServerBypass.java 189 2005-12-04 07:07:43Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server.provider;

import jp.ac.kyoto_u.i.siws.server.ServantInvoker;

import org.apache.axis.AxisFault;
import org.w3c.dom.Element;;

/**
 * サーバ側のバイパス.
 * サーバプロバイダがバイパスのファクトリとして機能します.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public abstract class ServerBypass {
    protected ServantInvoker invoker = null;

    /**
     * サービスハンドラを指定してインスタンスを作成します.
     * @param invoker サービスハンドラ
     */
    public ServerBypass(ServantInvoker invoker) {
        this.invoker = invoker;
    }

    /**
     * クライアントに伝えるバイパスの情報 XML ノードとして取得します.
     * @return バイパス情報
     */
    public abstract Element infoAxXML() throws AxisFault;

    /**
     * このバイパスを破棄します.
     *
```

```
     */
    public abstract void tearDown();

    /**
     * @return invoker を戻します。
     */
    public ServantInvoker getInvoker() {
        return invoker;
    }
}
```

A.1.14 ServerProvider.java

```
/**
 * $Id: ServerProvider.java 189 2005-12-04 07:07:43Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server.provider;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.axis.AxisFault;
import org.apache.axis.components.logger.LogFactory;
import org.apache.commons.logging.Log;

import jp.ac.kyoto_u.i.siws.Commons;
import jp.ac.kyoto_u.i.siws.Messages;
import jp.ac.kyoto_u.i.siws.client.SiWSClientException;
import jp.ac.kyoto_u.i.siws.server.ServantInvoker;
import jp.ac.kyoto_u.i.siws.server.provider.rmi.RMIServerProvider;
import jp.ac.kyoto_u.i.siws.xml.conf.ConfigType;
import jp.ac.kyoto_u.i.siws.xml.conf.Param;

/**
 * サーバのプロバイダ
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public abstract class ServerProvider implements Comparable<ServerProvider> {
    private static boolean isLoaded = false;
    private static final Map<String, ServerProvider> providerMap =
        new HashMap<String, ServerProvider>();
```

```
protected static final Log log = LogFactory.getLog(ServerProvider.class.getName());

protected final Map<String, String> params = new HashMap<String, String>();

private int priority = 0;

/**
 * インストールされたサーバプロバイダを
 * 優先度を元に降順ソートしたリストを取得します.
 * @return 降順ソートされたサーバプロバイダ
 */
public static List<ServerProvider> getAllSorted() throws AxisFault {
    log.debug("enter-method: getAllSorted");

    if(!isLoaded) load();
    List<ServerProvider> list = new ArrayList<ServerProvider>(providerMap.values());
    Collections.sort(list);
    Collections.reverse(list);

    log.debug("exit-method: getAllSorted");
    return list;
}

/**
 * プロバイダの名前でサーバプロバイダのインスタンスを取得します.
 * @param name プロバイダの名前
 * @return 名前に対応するサーバプロバイダ
 * @throws AxisFault
 */
public static ServerProvider getByName(String name) throws AxisFault {
    if(!isLoaded) load();
    return providerMap.get(name);
}

/**
 * 指定されたプロバイダ名のリストから,
 * 対応するサーバプロバイダのうち優先度の最も高いものを返します.
 * list の中に対応するサーバプロバイダ一つもない場合 null を返します.
 * @param list プロバイダ名のリスト
 * @return 対応するサーバプロバイダ. 見つからない場合は null.
 */
public static ServerProvider selectForBypass(List<String> list) throws AxisFault {
    Collections.sort(list);

    for (ServerProvider sp: getAllSorted()) {
```

```
        if(Collections.binarySearch(list, sp.getName()) >= 0)
            return sp;
    }
    return null;
}

/**
 * 設定ファイルから providerMap にサーバプロバイダをロードします.
 *
 */
@SuppressWarnings("unchecked")
private static void load() throws AxisFault {
    log.debug(Messages.getMessage("log-enter"));

    // 設定の取得
    ConfigType config = Commons.getConfig();

    List<ConfigType.ServerType.ProviderType> providers =
        config.getServer().getProvider();

    String errMessage = Messages.getMessage("err-server-provider-load");

    // オブジェクトの生成
    for (ConfigType.ServerType.ProviderType provider : providers) {
        String name = provider.getClassName();
        errMessage = Messages.getMessage("err-server-provider-load", name);
        try {
            // サーバプロバイダ p の生成
            Class c = Class.forName(name);
            Method m = c.getMethod("instance", new Class[] {});
            //ServerProvider p = (ServerProvider)c.newInstance();
            ServerProvider p = (ServerProvider)m.invoke(c, new Object[]{});
            if(p == null) throw new NullPointerException(errMessage);

            /* 優先度の設定 */
            p.setPriority(provider.getPriority());

            /* オプションパラメータの設定 */
            Map<String, String> map = p.getParams();
            List<Param> params = provider.getParam();
            for(Param param: params) {
                map.put(param.getKey(), param.getValue());
            }

            /* 初期化 */
```

```
            p.init();

            providerMap.put(p.getName(), p);
            log.info(Messages.getMessage("info-0002", name));
        } catch (Exception e) {
            throw new SiWSClientException(
                errMessage, e);
        }
    }

    isLoaded = true;

    log.debug(Messages.getMessage("log-exit"));
}

public int compareTo(ServerProvider provider) {
    return this.getPriority() - provider.getPriority();
}

/**
 * プロバイダの名前を取得します.
 * @return プロバイダの名前
 */
public abstract String getName();

/**
 * 初期化します.
 * 一般にはパラメータのマップを元にフィールドを設定します.
 *
 */
public void init() {}

/**
 * プロバイダの優先度を取得します.
 * 数値が高いほど優先度が高く,
 * バイパス用のプロバイダとして利用されやすいことを意味します.
 * @return プロバイダの優先度
 */
public int getPriority() {
    return priority;
}

/**
 * クライアントと通信を行うバイパスを作成します.
 * @param invoker サーバントの呼び出しを行うサービスハンドラ
```

```
     * @return 作成したバイパス
     */
    public abstract ServerBypass createBypass(ServantInvoker invoker) throws AxisFault;

    /**
     * デバグ用
     * @param args
     */
    public static void main(String[] args) throws Exception {
        for (ServerProvider sp : getAllSorted()) {
            System.out.println(sp.getName());
        }
        /*List<String> list = new ArrayList<String>();
        list.add("http");
        System.out.println(selectForBypass(list).getName());*/
    }


    /**
     * @param priority 設定する priority。
     */
    public void setPriority(int priority) {
        this.priority = priority;
    }

    /**
     * @return params を戻します。
     */
    public Map<String, String> getParams() {
        return params;
    }
}
```

A.1.15 Invoker.java

```
/**
 * $Id: Invoker.java 188 2005-12-04 04:24:06Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server.provider.rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * サービスハンドラを呼び出すための RMI 用インタフェース.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
```

```
 *
 */
public interface Invoker <T> extends Remote {
    /**
     * サービスハンドラに対して呼び出しを行います.
     * @param params 呼び出しのパラメータ
     * @return 呼び出し結果
     * @throws RemoteException
     */
    public T invoke(Object[] params) throws RemoteException;
}
```

A.1.16 InvokerImpl.java

```
/**
 * $Id: InvokerImpl.java 185 2005-12-02 04:54:57Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server.provider.rmi;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import jp.ac.kyoto_u.i.siws.server.ServantInvoker;

/**
 * サービスハンドラを呼び出すRMI用のクラスです.
 * {@link jp.ac.kyoto_u.i.siws.server.ServantInvoker}
 * へのプロキシとして動作します.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class InvokerImpl extends UnicastRemoteObject implements Invoker {
    private static final long serialVersionUID = 1L;
    private ServantInvoker _invoker = null;

    /**
     * サービスハンドラを指定してインスタンスを作成します.
     * @throws RemoteException
     * @param invoker サービスハンドラ
     */
    public InvokerImpl(ServantInvoker invoker) throws RemoteException {
        this._invoker = invoker;
    }

    public Object invoke(Object[] params) throws RemoteException {
        try {
```

```
            return _invoker.invoke(params);
        } catch(Exception e) {
            throw new RemoteException("Server-side RMI exception: ", e);
        }
    }

}
```

A.1.17 RMIServerBypass.java

```
/**
 * $Id: RMIServerBypass.java 189 2005-12-04 07:07:43Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server.provider.rmi;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.rmi.Naming;
import java.util.List;

import jp.ac.kyoto_u.i.siws.Commons;
import jp.ac.kyoto_u.i.siws.Constants;
import jp.ac.kyoto_u.i.siws.Messages;
import jp.ac.kyoto_u.i.siws.server.ServantInvoker;
import jp.ac.kyoto_u.i.siws.server.provider.ServerBypass;
import jp.ac.kyoto_u.i.siws.xml.soap.BypassInfo;
import jp.ac.kyoto_u.i.siws.xml.soap.ObjectFactory;
import jp.ac.kyoto_u.i.siws.xml.soap.Param;

import org.apache.axis.AxisFault;
import org.apache.axis.components.logger.LogFactory;
import org.apache.commons.logging.Log;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

/**
 * RMI のバイパスクラスです.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class RMIServerBypass extends ServerBypass {

    /**
     * {@link #infoAxXML()}において{@link #getRMIName()}の値を格納するキーの名前
```

```
 */
public static final String KEY_RMINAME = "rmi_name";

public static final String OBJ_PREFIX = "siwsObj";

private static long max_id = -1;

protected static Log log = LogFactory.getLog(RMIServerBypass.class.getName());
private long id = -1;

/**
 * サービスハンドラを指定してインスタンスを作成します.
 * @param invoker サービスハンドラ
 */
public RMIServerBypass(ServantInvoker si) throws AxisFault {
    super(si);
    prepareRegistry();
    id = fetchId();

    try {
        Invoker invoker = new InvokerImpl(si);
        Naming.rebind(getRMIName(), invoker);
    } catch (Exception e) {
        throw AxisFault.makeFault(e);
    }
}

/**
 * 新しい ID を取得し，カウンタをインクリメントする.
 * @return 新しい ID
 */
private static long fetchId() {
    max_id ++;
    return max_id;
}

/**
 * RMIRegistry を起動します.
 * すでに起動されている場合はログに例外が出力されますが,
 * そのまま実行が継続されます.
 * @throws AxisFault
 *
 */
public void prepareRegistry() throws AxisFault {
    log.debug(Messages.getMessage("log-enter"));
```

```
    RMIServerProvider p = RMIServerProvider.instance();
    if(p.isRegistryPrepared()) {
        log.debug(Messages.getMessage("log-0004"));
        return;
    }

    try {
        String command = "rmiregistry " + p.getPort();
        log.info(Messages.getMessage("info-0004", command));
        Process process = Runtime.getRuntime().exec(command);
        log.debug("Process returned.");

        BufferedReader reader = new BufferedReader(new InputStreamReader(
                process.getErrorStream()));

        /* エラーの捕捉 */
        String str = null;
        StringBuffer sb = new StringBuffer("");
        while(null != (str = reader.readLine())) {
            sb.append(str);
            sb.append("\n");
        }
        str = sb.toString();
        if(str.length() > 0) log.info(sb.toString());
        else log.debug("No error found in exec.");
    } catch (IOException e) {
        throw AxisFault.makeFault(e);
    }

    p.setRegistryPrepared(true);

    log.debug(Messages.getMessage("log-exit"));
}

/**
 * RMI のネーミングサービスに登録されたオブジェクト名を取得します.
 * "rmi://localhost:11099/siwsObj0"などのような名前です.
 * @return RMI のオブジェクト名
 * @throws AxisFault
 */
public String getRMIName() throws AxisFault {
    RMIServerProvider p = RMIServerProvider.instance();

    return "rmi://" + p.getFHost() + ":" + p.getPort() + "/" + OBJ_PREFIX + id;
}
```

```
@SuppressWarnings("unchecked")
@Override
public Element infoAxXML() throws AxisFault {
    ObjectFactory factory = new ObjectFactory();
    Element elem = null;
    try {
        BypassInfo info = factory.createBypassInfo();

        info.setProvider(RMIServerProvider.NAME);
        List<Param> list = info.getParam();

        /* RMIName プロパティの作成 */
        Param p = factory.createParam();
        p.setKey(KEY_RMINAME);
        p.setValue(getRMIName());
        list.add(p);
        Node node = Commons.getXMLAsNode(info, Constants.PKGNAME_SOAP);
        elem = (Element)node;
    } catch (Exception e) {
        throw AxisFault.makeFault(e);
    }

    return elem;
}

@Override
public void tearDown() {
    // TODO rmiregistry から unbind するコード

}

/**
 * デバグ用
 * @param args
 * @throws Exception
 */
public static void main(String[] args) throws Exception {
    log.debug(Messages.getMessage("log-enter"));
    Process p = Runtime.getRuntime().exec("rmiregistry 11099");
    BufferedReader reader = new BufferedReader(new InputStreamReader(
            new BufferedInputStream(p.getErrorStream())));
    System.out.println(reader.readLine());
    //p.destroy();
}
```

```
}
```

A.1.18 RMIServerProvider.java

```
/**
 * $Id: RMIServerProvider.java 255 2006-01-24 05:30:24Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.server.provider.rmi;

import java.net.InetAddress;

import jp.ac.kyoto_u.i.siws.Messages;
import jp.ac.kyoto_u.i.siws.server.ServantInvoker;
import jp.ac.kyoto_u.i.siws.server.provider.ServerBypass;
import jp.ac.kyoto_u.i.siws.server.provider.ServerProvider;

import org.apache.axis.AxisFault;
import org.apache.axis.components.logger.LogFactory;
import org.apache.commons.logging.Log;

/**
 * Java RMI のサーバプロバイダ.
 * @author Ikuo Matsumura (matumura@lab7.kuis.kyoto-u.ac.jp)
 *
 */
public class RMIServerProvider extends ServerProvider {
    private static RMIServerProvider instance = null;

    public static final String NAME = "rmi";

    /**
     * ホスト名を表す設定ファイルでのキー名
     */
    public static final String CN_HOST = "host";

    /**
     * クライアントが接続するホスト名を表す設定ファイルのキー名
     */
    public static final String CN_FHOST = "foreign_host";

    /**
     * ポート番号を表す設定ファイルのキー名
     */
    public static final String CN_PORT = "port";

    /**
```

```
 * RMI レジストリがすでに起動されているか否かを表す設定ファイルのキー名
 */
public static final String CN_IRP = "isRegistryPrepared";

protected static Log log = LogFactory.getLog(RMIServerProvider.class.getName());

private boolean isRegistryPrepared = false;

private String host = "localhost";
private String fHost = "localhost";
private int port = 11099;

/**
 * インスタンスを作成します.
 */
private RMIServerProvider() {
}

/**
 * フィールドの初期化を行います.
 * 設定ファイルから読み込んだ情報を元に
 * ホスト名とポートの設定を行います.
 * デフォルトのホスト名は"localhost", ポート番号は 11099 です.
 */
@Override
public void init() {
    super.init();
    String v = null;
    if(null != (v = params.get(CN_HOST))) host = v;
    if(null != (v = params.get(CN_FHOST))) fHost = v;
    if(null != (v = params.get(CN_PORT))) {
        try {
            port = Integer.parseInt(v);
        } catch(NumberFormatException e) {
            log.warn(Messages.getMessage("warn-0001", v, Integer.toString(port)));
        }
    }
    if(null != (v = params.get(CN_IRP))) {
        try {
            isRegistryPrepared = Boolean.parseBoolean(v);
        } catch(Exception e) {
            log.info(e.toString());
            log.info(Messages.getMessage("warn-0002"));
            isRegistryPrepared = false;
        }
```

```
    }
}

/**
 * インスタンスを取得する.
 * シングルトンのコンストラクタの役割を果たす.
 * @return インスタンス
 */
public static RMIServerProvider instance() {
    if(instance == null)
        instance = new RMIServerProvider();
    return instance;
}

@Override
public String getName() {
    return NAME;
}

@Override
public ServerBypass createBypass(ServantInvoker invoker) throws AxisFault {
    RMIServerBypass bypass = new RMIServerBypass(invoker);
    return bypass;
}

/**
 * デバグ用
 */
public static void main(String[] args) throws Exception {
    InetAddress addr = InetAddress.getLocalHost();
    System.out.println(addr.getHostAddress());
}

/**
 * @return fHost を戻します。
 */
public String getFHost() {
    return fHost;
}

/**
 * @return host を戻します。
 */
public String getHost() {
    return host;
```

```
    }

    /**
     * @return port を戻します。
     */
    public int getPort() {
        return port;
    }

    /**
     * @return isRegistryPrepared を戻します。
     */
    public boolean isRegistryPrepared() {
        return isRegistryPrepared;
    }

    /**
     * @param isRegistryPrepared 設定する isRegistryPrepared。
     */
    public void setRegistryPrepared(boolean isRegistryPrepared) {
        this.isRegistryPrepared = isRegistryPrepared;
    }

}
```

A.2 SimpleTranslation のソースコードと WSDL

SimpleTranslation は SiWS の動作テスト及びバイパス作成コストの評価の目的のために作られたサンプル Web サービスである.

A.2.1 Translation.java

```
/**
 * $Id: Translation.java 127 2005-11-18 06:43:08Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.samples.server;

/**
 * 翻訳インタフェース
 * @author Matsumura
 *
 */
public interface Translation {
    public String translate(String srcLang, String tgtLang, String source);
}
```

A.2.2 SimpleTranslation.java

```
/**
 * $Id: SimpleTranslation.java 127 2005-11-18 06:43:08Z ikuom $
 */
package jp.ac.kyoto_u.i.siws.samples.server;

/**
 * サンプルのサーバアプリケーション
 * @author Matsumura
 *
 */
public class SimpleTranslation implements Translation {
    /**
     * 翻訳対象文 source を srcLang で指定された言語から tgtLang で指定された
     * 言語に翻訳する.
     * 現在の実装はパラメータにかかわらず「source + ": こんにちは"」を返す.
     *
     * @param srcLang 翻訳元言語 ID
     * @param tgtLang 翻訳先言語 ID
     * @param source 翻訳対象文
     * @return source + ": こんにちは"
     */
    public String translate(String srcLang, String tgtLang, String source) {
        return source + ":こんにちは";
    }
}
```

A.2.3 WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://i.kyoto-u.ac.jp/siws/samples/sts" xmlns:apachesoap="ht
<!--WSDL は Apache Axis version: 1.2.1
Built on Jun 14, 2005 (09:15:57 EDT) によって生成されました / [en]-(WSDL created by Apache Axis ver
Built on Jun 14, 2005 (09:15:57 EDT))-->

   <wsdl:message name="translateResponse">
      <wsdl:part name="translateReturn" type="xsd:string"/>
   </wsdl:message>

   <wsdl:message name="translateRequest">
      <wsdl:part name="in0" type="xsd:string"/>
      <wsdl:part name="in1" type="xsd:string"/>
      <wsdl:part name="in2" type="xsd:string"/>
   </wsdl:message>

   <wsdl:portType name="SimpleTranslation">
```

```
      <wsdl:operation name="translate" parameterOrder="in0 in1 in2">
         <wsdl:input message="impl:translateRequest" name="translateRequest"/>
         <wsdl:output message="impl:translateResponse" name="translateResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="SimpleTranslationSoapBinding" type="impl:SimpleTranslation">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="translate">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="translateRequest">
             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://i.kyoto-u.ac.jp/siws/samples/sts" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="translateResponse">
             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://i.kyoto-u.ac.jp/siws/samples/sts" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="SimpleTranslationService">
      <wsdl:port binding="impl:SimpleTranslationSoapBinding" name="SimpleTranslation">
         <wsdlsoap:address location="http://localhost:8080/axis/services/SimpleTranslation"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

A.3　ＸＭＬスキーマ定義

A.3.1　siws-soap.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Id: siws-soap.xsd 254 2006-01-24 05:17:10Z ikuom $ -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://i.kyoto-u.ac.jp/siws/soap/"
    xmlns:tns="http://i.kyoto-u.ac.jp/siws/soap/" xmlns:Q1="xs">

    <complexType name="OperUsage">
        <sequence>
            <choice>
                <element name="staticIndex" type="nonNegativeInteger"
                    maxOccurs="unbounded" minOccurs="0">
                </element>
                <element name="staticRange" type="string"
                    maxOccurs="unbounded" minOccurs="0">
```

```
            </element>
        </choice>
    </sequence>
</complexType>

<complexType name="AvailableProviders">
    <sequence>
        <element name="provider" type="string" maxOccurs="unbounded"
            minOccurs="1">
        </element>
    </sequence>
</complexType>

<complexType name="Param">
    <simpleContent>
        <extension base="string">
            <attribute name="key" type="string"
                form="unqualified">
            </attribute>
        </extension>
    </simpleContent>
</complexType>

<element name="clientContext">
    <complexType>
        <all minOccurs="1" maxOccurs="1">
            <element name="operationUsage"
                type="tns:OperUsage" maxOccurs="1" minOccurs="1">
            </element>
            <element name="availableProviders"
                type="tns:AvailableProviders" maxOccurs="1" minOccurs="1">
            </element>
        </all>
    </complexType>
</element>

<element name="bypassInfo">
    <complexType>
        <sequence>
            <element name="provider" type="string"></element>
            <element name="param" type="tns:Param"
                maxOccurs="unbounded" minOccurs="0">
            </element>
        </sequence>
    </complexType>
```

```
    </element>

</schema>
```

A.3.2 siws-config.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="NewDataSet"
    targetNamespace="http://i.kyoto-u.ac.jp/siws/config"
    xmlns:mstns="http://i.kyoto-u.ac.jp/siws/config"
    xmlns="http://i.kyoto-u.ac.jp/siws/config"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified" elementFormDefault="qualified">
    <xs:element name="config">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="client" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="provider" minOccurs="0"
                                maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:attribute name="className"
                                        form="unqualified" type="xs:string" />
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="server" minOccurs="0" maxOccurs="1">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="provider" minOccurs="0"
                                maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="param" type="Param"
                                            maxOccurs="unbounded" minOccurs="0">
                                        </xs:element>
                                    </xs:sequence>
                                    <xs:attribute name="priority"
                                        form="unqualified" type="xs:int" />
                                    <xs:attribute name="className"
                                        form="unqualified" type="xs:string" />
                                </xs:complexType>
```

```
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="Param">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="key" type="xs:string"
                    form="unqualified">
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

    <xs:element name="NewDataSet" msdata:IsDataSet="true"
        msdata:Locale="ja-JP">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element ref="config" />
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```