(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6169081号
(P6169081)
(45)発行日　平成29年7月26日(2017.7.26)　(24)登録日　平成29年7月7日(2017.7.7)

(51)Int.Cl.　F I
G06F 9/54 (2006.01)　G06F 9/46 480D

請求項の数 10 (全 23 頁)

(21)出願番号　特願2014-529889(P2014-529889)
(86)(22)出願日　平成24年9月7日(2012.9.7)
(65)公表番号　特表2014-526740(P2014-526740A)
(43)公表日　平成26年10月6日(2014.10.6)
(86)国際出願番号　PCT/US2012/054131
(87)国際公開番号　WO2013/036750
(87)国際公開日　平成25年3月14日(2013.3.14)
審査請求日　平成27年6月19日(2015.6.19)
(31)優先権主張番号　61/533,068
(32)優先日　平成23年9月9日(2011.9.9)
(33)優先権主張国　米国(US)
(31)優先権主張番号　13/427,574
(32)優先日　平成24年3月22日(2012.3.22)
(33)優先権主張国　米国(US)

前置審査

(73)特許権者　502303739
オラクル・インターナショナル・コーポレイション
アメリカ合衆国カリフォルニア州９４０６５レッドウッド・シティー，オラクル・パークウェイ５００
(74)代理人　110001195
特許業務法人深見特許事務所
(72)発明者　カー，ハロルド
アメリカ合衆国、９４０６５　カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７

最終頁に続く

(54)【発明の名称】ミドルウェアまたはその他の環境において使用される動的呼出しおよびサービスインターフェイスを提供するためのシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
ミドルウェアまたはその他の環境において使用される動的呼出しおよびサービスインターフェイスを提供するためのシステムであって、
クライアントコンピュータ、クライアントコンテナ、およびクライアントアプリケーションを含むクライアントサイド環境と、
サービスプロバイダコンピュータ、サービスプロバイダコンテナ、およびサービスを含むサービスサイド環境と、
前記クライアントサイド環境および前記サービスサイド環境で動作可能な動的呼出しおよびサービスインターフェイスとを備え、
前記クライアントサイド環境において、
前記クライアントアプリケーションがメッセージをメッセージ処理のためのランタイムモジュールに与えることができるようにするディスパッチャリクエスト関数が与えられ、かつ、前記クライアントアプリケーションが前記メッセージを前記メッセージ処理後に前記ランタイムモジュールから受けることができるようにするディスパッチャレスポンス関数が与えられ、
メッセージの通信を提供するためのトランスポートモジュールから、前記クライアントサイド環境のメッセージ処理を切離すクライアントリクエストトランスポート関数およびクライアントレスポンストランスポート関数が与えられ、
前記サービスサイド環境において、

前記サービスサイド環境のメッセージ処理を前記トランスポートモジュールから切離すサービスリクエストトランスポート関数およびサービスレスポンストランスポート関数が与えられ、
前記メッセージが前記ランタイムモジュールを介して受けとられて前記サービスによって処理されるようにするプロバイダリクエスト関数と、レスポンスメッセージが処理のために前記サービスによって前記ランタイムモジュールに与えられるようにするプロバイダレスポンス関数とが与えられる、システム。
【請求項２】
各リクエスト関数は、前記クライアントサイド環境および前記サービスサイド環境において、対応するコールバック関数に関連付けられており、前記コールバック関数はメッセージ処理を非同期にする、請求項１に記載のシステム。
【請求項３】
前記サービスは、ウェブサービスであり、
前記サービスプロバイダコンテナは、ウェブサービスコンテナであり、
前記クライアントアプリケーションは、ウェブサービスクライアントである、請求項１または２に記載のシステム。
【請求項４】
前記クライアントサイド環境において、
前記クライアントアプリケーションは、前記ディスパッチャリクエスト関数をコールすることによって、ターゲットサービスに対するリクエストを行ない、
前記リクエストは、前記ランタイムモジュールによって処理され、
前記ランタイムモジュールはクライアントリクエストトランスポート関数をコールし、
前記クライアントコンテナにおける前記クライアントリクエストトランスポート関数は、前記リクエストをトランスポートモジュールに伝え、
前記サービスサイド環境において、
前記サービスリクエストトランスポート関数は前記リクエストを前記トランスポートモジュールから受け、
前記サービスリクエストトランスポート関数は前記リクエストを処理のために前記ランタイムモジュールに伝え、
前記ランタイムモジュールはプロバイダリクエスト関数を前記サービスによる処理のためにコールし、
前記サービスサイド環境において、
前記サービスがレスポンスを与えることができる状態になると、前記サービスはプロバイダレスポンス関数をコールして前記レスポンスを処理のために前記ランタイムモジュールに伝え、
前記ランタイムモジュールは前記サービスレスポンストランスポート関数をコールし、
前記サービスレスポンストランスポート関数は、前記レスポンスを前記トランスポートモジュールに伝え、
前記クライアントサイド環境において、
前記クライアントコンテナは前記レスポンスを前記トランスポートモジュールから受け、
前記クライアントコンテナはクライアントレスポンストランスポート関数をコールして前記レスポンスを前記ランタイムモジュールに伝え、
シンプルオブジェクトアクセスプロトコル（ＳＯＡＰ）に従う処理の後、前記ランタイムモジュールは、前記リクエストを前記クライアントアプリケーションに送るために前記ディスパッチャレスポンス関数に伝える、請求項１から３のいずれかに記載のシステム。
【請求項５】

リクエストヘッダに含まれるレスポンスアドレスに応じて前記レスポンスを前記クライアントアプリケーションに伝えるために、２つの異なるサービスレスポンストランスポート関数を与え選択的に呼出すことができ、
前記クライアントサイド環境において、
前記クライアントアプリケーションがメッセージを処理するために前記ランタイムモジュールに非同期で与えることができるようにするディスパッチャリクエスト関数が与えられ、
ディスパッチャレスポンス関数は、処理後にレスポンスを前記ランタイムモジュールから非同期で受けることができるようにするために与えられ、
メッセージ処理を前記トランスポートモジュールから切離し、かつメッセージを非同期でハンドリングできるようにする、クライアントリクエストトランスポート関数およびクライアントレスポンストランスポート関数が与えられ、
前記サービスサイド環境において、
メッセージ処理を前記トランスポートモジュールから非同期で切離し、かつ前記ランタイムモジュールがメッセージを処理できるようにする、サービスリクエストトランスポート関数およびサービスレスポンストランスポート関数が与えられ、
前記ランタイムモジュールによる処理後にリクエストメッセージを非同期で受けるプロバイダリクエスト関数が与えられ、
サービスレスポンスを前記ＳＯＡＰに従う処理のために前記ランタイムモジュールに非同期で与えることができるようにするプロバイダレスポンス関数が与えられる、請求項４に記載のシステム。
【請求項６】
ミドルウェアまたはその他の環境において使用されるメッセージングアプリケーションプログラミングインターフェイス（ＡＰＩ）を提供するための方法であって、
クライアントコンピュータを含むクライアントサイド環境において、クライアントコンテナおよびクライアントアプリケーションにアクセスするステップと、
サービスプロバイダコンピュータを含むサービスサイド環境において、サービスプロバイダコンテナおよびサービスにアクセスするステップと、
前記クライアントサイド環境および前記サービスサイド環境で動作可能な動的呼出しおよびサービスインターフェイスにアクセスするステップとを含み、
前記クライアントサイド環境において、
前記クライアントアプリケーションがメッセージを処理するためのランタイムモジュールに与えることができるようにするディスパッチャリクエスト関数が与えられ、かつ、処理後に、前記クライアントアプリケーションが前記メッセージを前記ランタイムモジュールから受けることができるようにするディスパッチャレスポンス関数が与えられ、
メッセージの通信を提供するためのトランスポートモジュールからメッセージ処理を切離すクライアントリクエストトランスポート関数およびクライアントレスポンストランスポート関数が与えられ、
前記サービスサイド環境において、
前記メッセージ処理を前記トランスポートモジュールから切離すサービスリクエストトランスポート関数およびサービスレスポンストランスポート関数が与えられ、
前記サービスが前記ランタイムモジュールを介してメッセージを受けて処理することができるようにするプロバイダリクエスト関数と、前記サービスが処理のためにレスポンスメッセージを前記ランタイムモジュールに与えることができるようにするプロバイダレスポンス関数とが与えられる、方法。
【請求項７】
各リクエスト関数は、前記クライアントサイド環境および前記サービスサイド環境において、対応するコールバック関数に関連付けられており、前記コールバック関数はメッセージ処理を非同期にする、請求項６に記載の方法。
【請求項８】

前記サービスは、ウェブサービスであり、
前記サービスプロバイダコンテナは、ウェブサービスコンテナであり、
前記クライアントアプリケーションは、ウェブサービスクライアントである、請求項６または７に記載の方法。
【請求項９】
前記クライアントサイド環境において、
前記クライアントアプリケーションは、前記ディスパッチャリクエスト関数をコールすることによって、ターゲットサービスに対するリクエストを行ない、
前記リクエストは、前記ランタイムモジュールによって処理され、
前記ランタイムモジュールはクライアントリクエストトランスポート関数をコールし、
前記クライアントコンテナにおける前記クライアントリクエストトランスポート関数は、前記リクエストを前記トランスポートモジュールに伝え、
前記サービスサイド環境において、
前記サービスリクエストトランスポート関数は前記リクエストを前記トランスポートモジュールから受け、
前記サービスリクエストトランスポート関数は前記リクエストを処理のために前記ランタイムモジュールに伝え、
前記ランタイムモジュールはプロバイダリクエスト関数を前記サービスによる処理のためにコールし、
前記サービスサイド環境において、
前記サービスがレスポンスを与えることができる状態になると、前記サービスはプロバイダレスポンス関数をコールして前記レスポンスを処理のために前記ランタイムモジュールに伝え、
前記ランタイムモジュールは前記サービスレスポンストランスポート関数をコールし、
前記サービスレスポンストランスポート関数は、前記レスポンスを前記トランスポートモジュールに伝え、
前記クライアントサイド環境において、
前記クライアントコンテナは前記レスポンスを前記トランスポートモジュールから受け、
前記クライアントコンテナはクライアントレスポンストランスポート関数をコールして前記レスポンスを前記ランタイムモジュールに伝え、
シンプルオブジェクトアクセスプロトコル（ＳＯＡＰ）処理の後、前記ランタイムモジュールは、前記リクエストを前記クライアントアプリケーションに送るために前記ディスパッチャレスポンス関数に伝える、請求項６から８のいずれかに記載の方法。
【請求項１０】
リクエストヘッダに含まれるレスポンスアドレスに応じて前記レスポンスを前記クライアントアプリケーションに伝えるために、２つの異なるサービスレスポンストランスポート関数を与え選択的に呼出すことができ、
前記クライアントサイド環境において、
前記クライアントアプリケーションがメッセージを処理するために前記ランタイムモジュールに非同期で与えることができるようにするディスパッチャリクエスト関数が与えられ、
処理後にレスポンスを前記ランタイムモジュールから非同期で受けることができるようにするディスパッチャレスポンス関数が与えられ、
メッセージ処理を前記トランスポートモジュールから切離し、かつ前記メッセージを非同期でハンドリングできるようにする、クライアントリクエストトランスポート関数およびクライアントレスポンストランスポート関数が与えられ、
前記サービスサイド環境において、

メッセージ処理を前記トランスポートモジュールから非同期で切離し、かつ前記ランタイムモジュールが前記メッセージを処理できるようにする、サービスリクエストトランスポート関数およびサービスレスポンストランスポート関数が与えられ、

前記ランタイムモジュールによる前記ＳＯＡＰに従う処理後にリクエストメッセージを非同期で受けるプロバイダリクエスト関数が与えられ、

サービスレスポンスを前記ＳＯＡＰに従う処理のために前記ランタイムモジュールに非同期で与えることができるようにするプロバイダレスポンス関数が与えられる、請求項９に記載の方法。

【発明の詳細な説明】

【技術分野】

【０００１】

優先権主張

本願は、本明細書に引用により援用する、２０１１年９月９日に出願され「ミドルウェアまたはその他の環境において使用される動的呼出しおよびサービスインターフェイスを提供するためのシステムおよび方法（SYSTEM AND METHOD FOR PROVIDING A DYNAMIC INVOCATION AND SERVICE INTERFACE FOR USE IN A MIDDLEWARE OR OTHER ENVIRONMENT）」と題された米国仮特許出願第６１／５３３，０６８号に基づく優先権の利益を主張する、２０１２年３月２２日に出願され「ミドルウェアまたはその他の環境において使用される動的呼出しおよびサービスインターフェイスを提供するためのシステムおよび方法」と題された米国特許出願第１３／４２７，５７４号に基づく優先権の利益を主張する。

【０００２】

発明の分野

本発明は、概して、コンピュータシステムおよびミドルウェア等のソフトウェアに関し、具体的には、ミドルウェアまたはその他の環境において使用される動的呼出しおよびサービスインターフェイス（dynamic invocation and service interface）（ＤＩＳＩ）を提供するためのシステムおよび方法に関する。

【背景技術】

【０００３】

背景

一般的に、ウェブサービスは、ネットワークを介したマシン間の遣り取りをサポートするソフトウェアシステムである。ウェブサービスプロトコルスタックは、サービスの定義、サービスの場所の特定、サービスの実装、およびサービス間の遣り取りを可能にするために使用できる、ネットワーキングおよびその他のプロトコルのスタックである。このようなプロトコルの例にはシンプルオブジェクトアクセスプロトコル（Simple Object Access Protocol）（ＳＯＡＰ）が含まれる。ＳＯＡＰは、ウェブサービスで使用される構造化された情報の交換を規定し、メッセージフォーマットを拡張可能マークアップ言語（Extensible Markup Language）（ＸＭＬ）に依拠し、メッセージの送信をその他のプロトコル（たとえばハイパーテキスト転送プロトコル（Hypertext Transfer Protocol）（ＨＴＴＰ）または簡易メール転送プロトコル（Simple Mail Transfer Protocol）（ＳＭＴＰ））に依拠する。一般的に、各ウェブサービスには、ウェブサービス記述言語（Web Service Description Language）（ＷＳＤＬ）等の、マシンが理解可能なフォーマットで記述されたインターフェイスがある。その他のシステムは、ＳＯＡＰメッセージを用いることにより、そこに記述されたやり方でウェブサービスインターフェイスと遣り取りすることができる。

【０００４】

その他の種類のプロトコルスタックも同様に、メッセージ情報の何らかの処理を含み得る。このメッセージそのものは何らかのやり方で符号化される。例としてコモンオブジェクトリクエストブローカアーキテクチャ（Common Object Request Broker Architecture）（ＣＯＲＢＡ）スタックがある。

【０００５】

コンピュータ同士が比較的ハイレベルで通信できるようにする、ＳＯＡＰまたはＣＯＲＢＡスタック等のメッセージリモーティング（remoting）スタックという文脈では、メッセージを１つの単位として扱うことができる、すなわちメッセージを得てこのメッセージに必要なデコードを含む処理を施しその結果を提供することができる、という利点がある。ウェブサービスを構築する際に使用するＪａｖａ（登録商標）ＥＥプラットフォームの一部として提供される、Java API for XML Web Services（ＪＡＸ－ＷＳ）仕様は、クライアントおよびサービスサイドの動的メッセージ処理のいくつかの側面を含む。しかしながら、ＪＡＸ－ＷＳ仕様は、クライアントサイドおよびサービスサイドのトランスポートレベルでの動的メッセージ処理手段も、サービスサイドのプロバイダレベルでのメッセージの非同期ハンドリング手段も提供しない。これらが、本発明の実施の形態が対象としている一般的な分野である。

【発明の概要】

【課題を解決するための手段】

【０００６】

概要

本明細書において、ミドルウェアまたはその他の環境において使用される動的呼出しおよびサービスインターフェイス（ＤＩＳＩ）を提供するためのシステムおよび方法が開示される。ある実施の形態に従い、このシステムおよび／または方法は、クライアントサイドでもサービスサイドでも動作可能である。サービスサイドにおいて、ユーザは、サービスリクエストトランスポートを用いてメッセージをインバウンド処理チェーンに挿入することができる。サービスサイドでのインバウンド処理後、メッセージはプロバイダリクエスト関数を介してユーザに与えられる。ユーザは、メッセージをサービスサイドアウトバウンド処理チェーンに挿入するプロバイダレスポンス関数を用いてレスポンスを与える。サービスサイドでのアウトバウンド処理後、メッセージはユーザのサービスレスポンストランスポートに与えられる。サービスリクエストトランスポートおよびサービスレスポンストランスポートは、メッセージング処理をトランスポートから切離し、メッセージ処理を事実上非同期にする。プロバイダリクエストおよびプロバイダレスポンスも事実上非同期である。クライアントサイドでは、ユーザは、ディスパッチャリクエストを用いてメッセージをアウトバウンド処理チェーンに挿入することができる。クライアントサイドでのアウトバウンド処理後、メッセージはユーザのクライアントリクエストトランスポートに与えられる。これは、メッセージ処理をトランスポートから切離し、メッセージ処理を事実上非同期にする。レスポンスを受けると、ユーザは、クライアントレスポンストランスポート関数を用いてレスポンスをクライアントサイドインバウンド処理チェーンに挿入する。クライアントサイドでのインバウンド処理後、メッセージはユーザのディスパッチャレスポンス関数に与えられる。ディスパッチャリクエストおよびディスパッチャレスポンスも事実上非同期である。クライアントサイドでもサービスサイドでもＤＩＳＩは非同期なので、スレッドはバックアップされない、すなわち、クライアントはリクエストを送信することができレスポンスを待つ必要はない。このプロセスによって、たとえばＳＯＡＰ処理をメッセージトランスポートから切離すこともでき、これを事実上非同期にする。

【図面の簡単な説明】

【０００７】

【図１】ある実施の形態に従う、動的呼出しおよびサービスインターフェイス（ＤＩＳＩ）を利用できるシステムを示す。

【図２】ある実施の形態に従う、標準およびＤＩＳＩクライアントサイド呼出しフローを示す。

【図３】ある実施の形態に従う、標準およびＤＩＳＩサービスサイド呼出しフローを示す。

【図４】ある実施の形態に従う、クライアントサイドの動的呼出しおよびサービスインターフェイス（ＤＩＳＩ）を提供する方法のフローチャートである。

【図５】ある実施の形態に従う、サービスサイドの動的呼出しおよびサービスインターフ

ェイス（ＤＩＳＩ）を提供する方法のフローチャートである。
【発明を実施するための形態】
【０００８】
詳細な説明
上記のように、ウェブサービスはネットワークを介したマシン間の遣り取りをサポートするソフトウェアシステムである。ウェブサービスプロトコルスタックは、サービスの定義、サービスの場所の特定、サービスの実装、およびサービス間の遣り取りを可能にするために使用できる、ネットワーキングおよびその他のプロトコルのスタックである。このようなプロトコルの例にはシンプルオブジェクトアクセスプロトコル（ＳＯＡＰ）が含まれる。ＳＯＡＰは、ウェブサービスで使用される構造化された情報の交換を規定し、メッセージフォーマットを拡張可能マークアップ言語（ＸＭＬ）に依拠し、メッセージの送信をその他のプロトコル（たとえばハイパーテキスト転送プロトコル（ＨＴＴＰ）または簡易メール転送プロトコル（ＳＭＴＰ））に依拠する。一般的に、各ウェブサービスには、ウェブサービス記述言語（ＷＳＤＬ）等の、マシンが理解可能なフォーマットで記述されたインターフェイスがある。その他のシステムは、ＳＯＡＰメッセージを用いることにより、そこに記述されたやり方でウェブサービスインターフェイスと遣り取りすることができる。
【０００９】
その他の種類のプロトコルスタックも同様に、メッセージ情報の何らかの処理を含み得る。このメッセージそのものは何らかのやり方で符号化される。例としてコモンオブジェクトリクエストブローカアーキテクチャ（ＣＯＲＢＡ）スタックがある。
【００１０】
コンピュータ同士が比較的ハイレベルで通信できるようにする、ＳＯＡＰまたはＣＯＲＢＡスタック等のメッセージリモーティングスタックという文脈では、メッセージを１つの単位として扱うことができる、すなわちメッセージを得てこのメッセージに必要なデコードを含む処理を施しその結果を提供することができる、という利点がある。ウェブサービスを構築する際に使用するＪａｖａ（登録商標）ＥＥプラットフォームの一部として提供される、Java API for XML Web Services（ＪＡＸ－ＷＳ）仕様は、クライアントサイドおよびサービスサイドの動的メッセージ処理のいくつかの側面を含む。しかしながら、ＪＡＸ－ＷＳ仕様は、クライアントサイドおよびサービスサイドのトランスポートレベルでの動的メッセージ処理手段も、サービスサイドのプロバイダレベルでのメッセージの非同期ハンドリング手段も提供しない。
【００１１】
ある実施の形態に従い、本明細書において、ミドルウェアまたはその他の環境において使用される動的呼出しおよびサービスインターフェイス（ＤＩＳＩ）を提供するためのシステムおよび方法が開示される。このシステムおよび／または方法は、クライアントサイドおよびサービスサイド双方で動作する。
【００１２】
サービスサイドにおいて、メッセージング処理をトランスポートから切離しメッセージ処理を事実上非同期にするサービスリクエストトランスポートおよびサービスレスポンストランスポートを用いて、メッセージを挿入することができる。サービスサイドでは、ユーザが、サービスリクエストトランスポートを用いてメッセージをインバウンド処理チェーンに挿入することができる。サービスサイドでのインバウンド処理後、メッセージはプロバイダリクエスト関数を介してユーザに与えられる。ユーザは、メッセージをサービスサイドアウトバウンド処理チェーンに挿入するプロバイダレスポンス関数を用いてレスポンスを与える。サービスサイドでのアウトバウンド処理後、メッセージはユーザのサービスレスポンストランスポートに与えられる。サービスリクエストトランスポートおよびサービスレスポンストランスポートは、メッセージング処理をトランスポートから切離し、メッセージ処理を事実上非同期にする。プロバイダリクエストおよびプロバイダレスポンスも事実上非同期である。（ＪＡＸ－ＷＳにプロバイダはないが、これは非同期ではない

。ＪＡＸ－ＷＳには、サービスリクエストトランスポートおよびサービスレスポンストランスポートに相当するものはない。）

クライアントサイドでは、アウトバウンド処理チェーンの初めにメッセージを置くディスパッチャリクエスト関数が与えられ、インバウンド処理チェーンの最後でメッセージを受けるディスパッチャレスポンス関数が与えられる。このプロセスは非同期なので、スレッドはバックアップされない、すなわち、クライアントはリクエストを送信することができレスポンスを待つ必要はない。また、このプロセスによって、たとえばＳＯＡＰ処理をメッセージトランスポートから切離すことができ、これを事実上非同期にする。クライアントサイドでは、ディスパッチャリクエスト関数がメッセージをクライアントサイドアウトバウンド処理チェーンに挿入する。クライアントサイドにおいて、ユーザは、メッセージを、ディスパッチャリクエストを用いてアウトバウンド処理チェーンに挿入することができる。クライアントサイドでのアウトバウンド処理後、メッセージはユーザのクライアントリクエストトランスポートに与えられる。これは、メッセージ処理をトランスポートから切離し、メッセージ処理を事実上非同期にする。レスポンスを受けると、ユーザは、クライアントレスポンストランスポート関数を用いてレスポンスをクライアントサイドインバウンド処理チェーンに挿入する。クライアントサイドのインバウンド処理後、メッセージはユーザのディスパッチャレスポンス関数に与えられる。ディスパッチャリクエストおよびディスパッチャレスポンスも事実上非同期である。（ＪＡＸ－ＷＳには、非同期機能がないディスパッチ関数がなく、ＪＡＸ－ＷＳには、クライアントリクエストトランスポートおよびクライアントレスポンストランスポートに相当するものはない。）クライアントサイドおよびサービスサイドいずれにおいても、ＤＩＳＩは非同期であるため、スレッドはバックアップされない、すなわち、クライアントはリクエストを送信することができレスポンスを待つ必要はない。また、このプロセスによって、たとえばＳＯＡＰ処理をメッセージトランスポートから切離すことができ、これを事実上非同期にする。

【００１３】

ある実施の形態に従い、ＤＩＳＩインターフェイスを、標準ＪＡＸ－ＷＳクライアントおよびサービスエンドポイントインターフェイスに倣ってモデル化することができるが、特に非同期性要件およびトランスポート中立性要件の分野において、オラクルサービスバス（Oracle Service Bus）（ＯＳＢ）等の環境または製品を含むという要件を満たすのに必要な相違も含み得る。たとえば、ＪＡＸ－ＷＳはクライアントサイド非同期プログラミングモデルを含むが、非同期サービスのためのモデルはない。したがって、ある実施の形態に従い、ＤＩＳＩは、それ自身の非同期クライアントサイドプログラミングモデルを定義することにより、クライアント、サービスエンドポイント、およびトランスポートレベルインターフェイスが整合するようにすることができる。

【００１４】

ある実施の形態に従い、このＤＩＳＩインターフェイスを用いることにより、たとえば、構成、管理性、データバインディング、および一般的なランタイムを含めてウェブサービスをＯＳＢが如何にして統合すべきかに関して、Oracle WebLogic、JRF Web Services、およびＯＳＢ等の、異なる環境または異なる製品間のコントラクトを形にすることができる。ＤＩＳＩインターフェイスの利点の一部には、明示的なＪａｖａ（登録商標）ＥＥまたはＪＲＦスタイルのデプロイメントなしでサービスエンドポイントを動的かつ自発的に初期化できること、コーラー（caller）（すなわち製品を含む）がフックポイントを通してインバウンドおよびアウトバウンドのトランスポートを完全に制御できること、コーラー（すなわち製品を含む）がWebServiceFeatureインスタンスを通してサービスまたはクライアント構成を完全に制御でき明示的にデプロイメント記述子が不要である、ならびに、完全に非同期であることが可能であり、リクエストおよび特定のリクエストに対するレスポンス処理を異なるスレッドで実行できることが、含まれる。

【００１５】

図１は、ある実施の形態に従う、動的呼出しおよびサービスインターフェイス（ＤＩＳＩ）を利用できるシステム１００を示す。

【００１６】
図１に示されるように、クライアントコンピュータ１１３、クライアントコンテナ１１４、およびクライアントアプリケーション１１６（たとえばウェブサービスクライアント）を含むクライアントサイド環境１１２は、トランスポート１２８を介して、サービスプロバイダコンピュータ１２３、サービスコンテナ１２４（たとえばウェブサービスコンテナ）、およびサービス１２６（たとえばウェブサービス）を含むサービスサイド環境１２２と、通信する。
【００１７】
クライアントサイドにおいて、クライアントアプリケーションは、ディスパッチャリクエスト（DispatcherRequest）１３２を呼出す（レスポンスが戻されたとき／戻されるならばスタックによって呼出されるディスパッチャレスポンス（DispatcherResponse）インスタンスも与える）ことによって、アウトバウンドコールを開始する。ランタイムスタック１４０は、アウトバウンドＳＯＡＰ処理（たとえばＷＳアドレス指定、ＭＴＯＭ、文字符号化、WS-ReliableMessaging、WS-Security等）を行なう。アウトバウンドＳＯＡＰ処理が完了すると、ランタイムスタックは、送信するメッセージ（およびレスポンスが戻されたときに呼出すクライアントレスポンストランスポートインスタンス）で、クライアントリクエストトランスポート１４２をコールする。次に、クライアントはトランスポート（たとえば共有メモリ、ＪＭＳ、ＦＴＰ等）でメッセージを送信１４４する役割がある。また、クライアントは、レスポンス（たとえばＪＭＳキュー、ソケットリスナ）を受けるクライアントレスポンスハンドリングコード１４６をセットアップする役割がある。このトランスポートでレスポンスが戻る１４７と、クライアントコードは、クライアントレスポンストランスポート（ClientRequestTransport）１４８をメッセージでコールする。ランタイムスタックは、インバウンドＳＯＡＰ処理を行なう。インバウンドＳＯＡＰ処理が完了すると、ランタイムスタックは、ディスパッチャレスポンス１５０をレスポンスでコールし、このレスポンスは次にクライアントアプリケーションに与えられる。
【００１８】
サービスサイドはクライアントサイドと同様である。サービスサイドは、トランスポート１２８からリクエストを受けるサービスリクエストハンドリングコード１６２をセットアップしなければならない。メッセージは、トランスポートに到着する１６０と、サービスリクエストハンドリングコード１６２によって扱われる。サービスリクエストハンドリングコード１６２は、サービスリクエストトランスポート（ServiceRequestTransport）１６３をこのメッセージでコールする（また、レスポンスが戻されたとき／戻されるならば呼出される２つのサービスレスポンストランスポートインスタンスを与える）。ランタイムスタック１４０はインバウンドＳＯＡＰ処理を行なう。インバウンドＳＯＡＰ処理が完了すると、ランタイムスタックは、サービス１２６に対するメッセージ（およびレスポンスを戻すために呼出すプロバイダレスポンスインスタンス）でプロバイダリクエスト１６４をコールする。サービス１２６は、このレスポンスを、プロバイダレスポンス１６６をメッセージ（メッセージは、呼出しのデータおよび処理に必要な任意のメタデータ（たとえばＭＩＭＥタイプ）双方を意味する）で呼出すことにより、伝える。ランタイムスタック１４０はアウトバウンドＳＯＡＰ処理を行なう。アウトバウンドＳＯＡＰ処理が完了すると、ランタイムスタックは、レスポンスをトランスポートに置く１７２という役割があるサービスレスポンストランスポート（ServiceResponseTransport）１６８をコールする。また、図１に示されるように、ボックス（すなわちオブジェクト）が、クライアント、ランタイム、またはサービスコンテナの内部に示されている場合、オブジェクトを作成することがコンテナの役割であることを示す。
【００１９】
クライアントサイド呼出しフロー
図２は、ある実施の形態に従う、それぞれ収容環境２１０、２５０と、ランタイムスタック２１２、２５２と、物理トランスポート２１６、２５６との間の、標準クライアントサイド呼出しフロー２０２およびＤＩＳＩクライアントサイド呼出しフロー２４２を示す

。

【００２０】

アウトバウンドトランスポートプロセッサ２２８およびインバウンドトランスポートプロセッサ２３２を使用する標準モデルでは、トランスポートハンドリングコードおよび物理トランスポートはたいてい不透明である。これに対し、ＤＩＳＩモデルでは、トランスポートハンドリングコードおよび物理トランスポートは、クライアントが利用できる／クライアントから見えるものである。（図２に示されるように、ボックス（すなわちオブジェクト）がコンテナの内部に示されているとき、これは、オブジェクトがコンテナ側から見えることを示す）。このため、標準モデルではクライアントコンテナから見えるのがディスパッチ２２６と非同期ハンドラ２３４のみであるのに対し、ＤＩＳＩモデルではクライアントリクエストハンドリング（すなわちクライアントリクエストおよびレスポンストランスポート）および物理トランスポート双方が見える／利用できる。

【００２１】

図２に示されるように、標準クライアントサイド呼出しフロー２０２は、ディスパッチインスタンスで始まり、ウェブサービスランタイムを通して実行され、トランスポートで終わる。ディスパッチは、使用し易い非同期プログラミングモデルを提供する。図２はモデルの一変形を示しており、この場合のコーラーは非同期ハンドラを与えてレスポンスを受ける：

ディスパッチ→…ランタイムスタック…→…トランスポート

非同期ハンドラ←…ランタイムスタック…←トランスポート

上記ディスパッチは収容環境によってコールされるのに対し、上記非同期ハンドラは収容環境によって実装される。上記手法の場合、非常に低レベルのソケットまたはＵＲＬ接続のファクトリ構成を使用する以外に、アプリケーションがトランスポート実装をオーバライドまたは制御する標準的な方法はない。ディスパッチは、永続性またはクラスタリングに対する標準的なサポートを提供せず、非同期ハンドラはシリアライズ可能である必要はない。

【００２２】

ある実施の形態に従い、ＤＩＳＩクライアントサイド呼出しフローは、それぞれディスパッチおよび非同期ハンドラに代えてディスパッチャリクエスト２６６およびディスパッチャレスポンス２７４を定め、それぞれアウトバウンド（リクエスト）トランスポートおよびインバウンド（レスポンス）トランスポートに代えてクライアントリクエストトランスポート２６８およびクライアントレスポンストランスポート２７０を定める：

ディスパッチャリクエスト→…ランタイムスタック…→クライアントリクエストトランスポート

ディスパッチャレスポンス←…ランタイムスタック…←クライアントレスポンストランスポート

上記ディスパッチャリクエストおよびクライアントレスポンストランスポートは収容環境によってコールされるのに対し、上記ディスパッチャレスポンスおよびクライアントリクエストトランスポートは収容環境によって実装される。

【００２３】

ある実施の形態に従い、収容環境（たとえばＯＳＢ）は、ディスパッチャリクエストのインスタンスを呼出すことによってリクエストを発行することができる。このリクエストは、クライアントリクエストトランスポートのインスタンスに対するコールで終わる。クライアントリクエストトランスポートインスタンスには、物理トランスポートと遣り取りする役割がある。物理トランスポートがレスポンスを受けると、収容環境は、クライアントレスポンストランスポートのインスタンスに対するレスポンス処理を呼出す。このレスポンス処理は、ランタイムスタック２４６を通して進行し、ディスパッチャレスポンスのインスタンスに対するコールで終わる。

【００２４】

Oracle WebLogic、Oracle Web Services、およびOracle Service Busを使用する実装例

は以下に示す構成を含み得る。異なる製品を利用する他の環境および実装は異なる構成を使用し得る。
・ディスパッチャリクエスト：Web Servicesによって実装される。
・クライアントリクエストトランスポート：収容環境（たとえばＯＳＢ）によって実装される。
・クライアントレスポンストランスポート：Web Servicesによって実装され、アウトバウンドコールがなされたときにクライアントリクエストトランスポートに送られる。
・ディスパッチャレスポンス：収容環境によって実装され、最初のリクエストがなされたときにディスパッチャリクエストに送られる。
【００２５】
ある実施の形態に従い、クライアントは、ディスパッチャリクエスト／ディスパッチャレスポンスの使用と、クライアントリクエストトランスポート／クライアントレスポンストランスポートの使用および標準パターンの使用とを、組合せてもよい。すなわち、クライアントは、標準ディスパッチ（同期、ポーリング、非同期ハンドラ）を、クライアントリクエストトランスポート／クライアントレスポンストランスポートとともに使用してもよい。また、クライアントは、ディスパッチャリクエスト／ディスパッチャレスポンスを、ビルトイントランスポートとともに使用してもよい。
【００２６】
クライアントライフサイクル
ある実施の形態に従い、ＤＩＳＩは、ディスパッチャリクエストインスタンスに対するファクトリとして機能するサービスクラスを提供する。収容環境（たとえばＯＳＢ）は、サービスまたはディスパッチ／ディスパッチャリクエストいずれかの初期化時に、ClientTransportFeatureという特徴を用いて、クライアントリクエストトランスポートの実装（implementation）を送る。
【００２７】

```
// Initialize instance of customer-implemented ClientRequestTransport
ClientRequestTransport clientRequestTransport = new OSBClientRequestTransport();
// Create DispatcherRequest
ClientTransportFeature ctf = new ClientTransportFeature(clientRequestTransport);
Service s = ServiceFactory.factory().create(..., ctf);
DispatcherRequest dispatcherRequest = s.createDispatch(...);
// Making a call
DispatcherResponse callback = new MyDispatcherResponse();
dispatcherRequest.request(..., callback);
```

コーラーがDispatcherRequest.request(...)を呼出すと、リクエスト処理が始まり、リクエスト処理は、エラーでまたはクライアントリクエストトランスポートに対するコールで終わる。交換がなされたとき、クライアントリクエストトランスポートは、呼出すクライアントレスポンストランスポートのインスタンスを受ける。アプリケーションコードがクライアントレスポンストランスポートインスタンスを呼出すと、レスポンス処理が始まりそのフローはディスパッチャレスポンスインスタンスに対するコールで終わる。
【００２８】
サービスサイド呼出しフロー
図３は、ある実施の形態に従う、それぞれサービスサイドコンテナ３１０、３５０と、ランタイムスタック２１２、２５２と、物理トランスポート２１６、２５６との間の、標準サービスサイド呼出しフロー３０２およびＤＩＳＩサービスサイド呼出しフロー３４２を示す。
【００２９】
図３に示されるように、標準サービスサイド呼出しフロー３０２は、トランスポートで

始まり、インバウンドトランスポートプロセッサ３１４およびアウトバウンドトランスポートプロセッサ３２０を介してウェブサービスランタイムを通して進行し、プロバイダ（Provider）インスタンス（またはＳＥＩ）３１６で終わり、次にリターンする。ＪＡＸ－ＷＳリファレンス実装は非同期プロバイダを提供しないが、標準非同期サービスサイドプログラミングモデルはない：

トランスポート→…ランタイムスタック…→プロバイダ

トランスポート←…ランタイムスタック…←プロバイダ（リターン）

上記プロバイダおよびプロバイダ（リターン）は、収容環境によって実装される。クライアントサイドと同じく、非常に低レベルのまたはアプリケーション－サーバ専用のインテグレーションを使用する以外に、アプリケーションがトランスポート実装をオーバライドまたは制御する標準的な方法はない。プロバイダモデルは、持続性またはクラスタリングに対する標準的なサポートを提供せず、プロバイダはシリアライズ可能である必要はなく、標準プロバイダモデルは非同期ではない。

【００３０】

ある実施の形態に従い、ＤＩＳＩサービスサイド呼出しフロー３４２は、それぞれインバウンドおよびレスポンストランスポートに代えてサービスリクエストトランスポート３５４およびサービスレスポンストランスポート３６４、３６８、ならびにプロバイダに代えてプロバイダリクエスト（ProviderRequest）３５６およびプロバイダレスポンス（ProviderResponse）３６０を定める：

サービスリクエストトランスポート→…ランタイムスタック…→プロバイダリクエスト

サービスレスポンストランスポート（匿名）←…ランタイムスタック…←プロバイダレスポンス

サービスレスポンストランスポート（非匿名）←…ｏｒ…

上記サービスリクエストトランスポートおよびプロバイダレスポンスは収容環境によってコールされ、サービスレスポンストランスポート（匿名）、サービスレスポンストランスポート（非匿名）およびプロバイダリクエストは収容環境によって実装される。

【００３１】

ある実施の形態に従い、収容環境（たとえばＯＳＢ）は、サービスリクエストトランスポートのインスタンスを呼出すことによってリクエストを発行することができる。このリクエストは、アプリケーションが何を初期化したかに応じて、Java Web Service（ＪＷＳ）、プロバイダインスタンス、またはプロバイダリクエストのインスタンスに対するコールで終わる。

【００３２】

サービスリクエストトランスポートのコーラーおよびサービスレスポンストランスポートの実装には、物理トランスポートと遣り取りするという役割がある。たとえば、収容環境は、サーブレット（servlet）、ＭＤＢにおけるリクエストを受けるかまたはこのリクエストをファイルから読出してサービスリクエストトランスポートを通してウェブサービスランタイムに送ることができる。

【００３３】

プロバイダリクエスト実装にはアプリケーションリクエストを実行するという役割があり、これは、ディスパッチャリクエスト（ＯＳＢの場合と同様媒介手段として機能する）を用いてクライアントフローにコールすることを含む。

【００３４】

アプリケーションレスポンスを利用できるとき、アプリケーションには、プロバイダリクエストに対する元のリクエストがなされたときにアプリケーションに送られたプロバイダレスポンスを呼出すという役割がある。

【００３５】

プロバイダレスポンスの呼出しは、ウェブサービスランタイムを通して行なわれ、サービスレスポンストランスポートインスタンスのうちの１つに対するコールで終わる。ある実施の形態に従うと、このモデルはクライアントサイドとサービスサイドでわずかに異な

る。非匿名のアドレス指定をサポートするために、ウェブサービスランタイムは、２つの異なるサービスレスポンストランスポート（ここでは３６４および３６８として示される）を呼出すことができる。これらトランスポートのうちの第１のトランスポートは、リクエストトランスポートの匿名のサービスレスポンストランスポートを表わす（すなわち「バックチャネル」としても知られている「匿名（anon）」）。ある実施の形態に従い、リクエストのReplyToまたはFaultToヘッダが匿名に設定されると、アウトバウンドのサービスサイドＳＯＡＰ処理後に匿名のサービスレスポンストランスポートがコールされる。この場合、非匿名のサービスレスポンストランスポートは決してコールされない。リクエストのReplyToまたはFaultToヘッダが非匿名に設定されると、アウトバウンド処理後に、匿名のサービスレスポンストランスポートが最初にメタデータのみで（すなわちメッセージなしで）コールされる。これは、接続を閉じ「ＯＫ」（たとえばＨＴＴＰ２０２）メッセージを送信者に送信する機能を与える。次に、非匿名のサービスレスポンストランスポートが、レスポンスメッセージでコールされる。レスポンスメッセージのための配信アドレスはアドレス指定ヘッダによって決まるので、非匿名のメッセージは、サービスリクエストトランスポートの前にあるトランスポートとは異なる種類のトランスポートで配信できる。非匿名のレスポンスを必要としないサービスコンテナについては、収容環境が、非匿名のサービスレスポンストランスポートに対してnullを送ってもよい。

【００３６】

Oracle WebLogic、Oracle Web Services、およびOracle Service Busを使用する実装例は、以下に示す構成を含み得る。異なる製品を利用するその他の環境および実装は異なる構成を使用し得る：

・サービスリクエストトランスポート：ウェブサービスによって実装される。

【００３７】

・プロバイダリクエスト：収容環境（たとえばＯＳＢ）によって実装される。

・プロバイダレスポンス：ウェブサービスによって実装されインバウンドコールがなされたときにプロバイダリクエストに送られる。

【００３８】

・サービスリクエストトランスポート（バックチャネル）：収容環境によって実装され、元のリクエストがなされたときにサービスリクエストトランスポートに送られる。

【００３９】

・サービスリクエストトランスポート（非匿名）：収容環境によって実装され、元のリクエストがなされたときにサービスリクエストトランスポートに送られる。

【００４０】

ある実施の形態に従い、サービスは、サービスリクエストトランスポート／サービスレスポンストランスポートの使用と、プロバイダリクエスト／プロバイダレスポンスの使用および標準パターンの使用とを、組合せてもよい。すなわち、サービスは、標準プロバイダ（またはＳＥＩ）を、サービスリクエストトランスポート／サービスレスポンストランスポートとともに使用してもよい。また、サービスは、プロバイダリクエスト／プロバイダレスポンスを、ビルトイントランスポートとともに使用してもよい。しかしながら、これには既存のデプロイメントモデルの使用が必要である。

【００４１】

サービスライフサイクル

ある実施の形態に従い、ＤＩＳＩは、サービスリクエストトランスポートインスタンスに対するファクトリとして機能するエンドポイントクラスを提供する。たとえば、ＯＳＢは、エンドポイント初期化中にプロバイダリクエストの実装を送ることができる。

【００４２】

```
// Initialize customer-implemented ProviderRequest and create Endpoint
ProviderRequest providerRequest = new OSBProviderRequest();
Endpoint e = EndpointFactory.factory().create(providerRequest, …);
```

```
// Create ServiceRequestTransport
ServiceRequestTransport serviceRequestTransport = e.createServiceRequestTranspor
t(...);
// Making a call
ServiceResponseTransport backchannel = new OSBBackchannelSRT();
ServiceResponseTransport nonanonchannel = new OSBNonAnonSRT();
serviceRequestTransport.request(..., backchannel, nonanonchannel);
```

図４および図５は、ある実施の形態に従う、動的呼出しおよびサービスインターフェイス（ＤＩＳＩ）を提供するための方法のフローチャートである。

【００４３】

図４に示されるように、クライアントサイドでは、ステップ３７０で、収容環境が、ディスパッチャリクエストのインスタンスを呼出すことによって、リクエストを発行する。ステップ３７２で、ランタイムスタックは、アウトバウンドＳＯＡＰ処理を行なう。ステップ３７４で、リクエストは、物理トランスポートと遣り取りする役割を有するクライアントリクエストトランスポートのインスタンスに対するコールで終了する。ステップ３７６で、物理トランスポートがレスポンスを受けると、収容環境が、クライアントレスポンストランスポートのインスタンスに対するレスポンス処理を呼出す。ステップ３７８で、ランタイムスタックはインバウンドＳＯＡＰ処理を行なう。ステップ３８０で、レスポンス処理がランタイムスタックを通して進行し、ディスパッチャレスポンスのインスタンスに対するコールで終了する。

【００４４】

図５に示されるように、サービスサイドでは、ステップ３８２で、リクエストがトランスポートに到着する。ステップ３８４で、収容環境が、サービスリクエストトランスポートを呼出すことによって、リクエストを発行する。ステップ３８６で、ランタイムは、サービスサイドのインバウンドＳＯＡＰ処理を行なう。ステップ３８８で、リクエストは、プロバイダリクエストに対するコールで終了する。ステップ３９０で、プロバイダリクエストが、アプリケーションリクエストの実行を開始する。ステップ３９２で、アプリケーションが、プロバイダレスポンスを、レスポンスで呼出す。ステップ３９４で、ランタイムは、サービスサイドのアウトバウンドＳＯＡＰ処理を行なう。ステップ３９６で、リクエスト処理が、サービスレスポンストランスポートに対するコールで終了する。ステップ３９８で、サービスレスポンストランスポートが物理トランスポートと遣り取りする。

【００４５】

スレッディング（threading）

ある実施の形態に従い、ユースケース（use-case）がＳＯＡＰランタイム内部でのバッファリングを必要としないとき、以下の特性が有効である。

【００４６】

・サービスリクエストトランスポートを呼出すスレッドは、プロバイダリクエストを呼出すスレッドと同一である。

【００４７】

・プロバイダレスポンスを呼出すスレッドは、レスポンスがある場合、非匿名のサービスレスポンストランスポートを呼出すスレッドと同一である。

【００４８】

・バックチャネルサービスレスポンストランスポートは、ReplyToまたはFaultToヘッダが匿名か非匿名かに応じて、サービスリクエストトランスポートを呼出したスレッドかまたはプロバイダレスポンスを呼出したスレッドによってコールされる。

【００４９】

・ＷＳ－ＡＴを用いる一方向のコールは、レスポンスが利用できるようになるまでリクエスト（たとえばＨＴＴＰ２０２）を承認しない。

【００５０】

・さもなければ、一方向のまたは非匿名のReplyToまたはFaultToに対し、ウェブサービスランタイムはできるだけ早くバックチャネルを呼出す。

【００５１】

・ディスパッチャリクエストを呼出すスレッドは、クライアントリクエストトランスポートを呼出すスレッドと同一である。

【００５２】

・クライアントレスポンストランスポートを呼出すスレッドは、ディスパッチャレスポンスを呼出すスレッドと同一である。

【００５３】

・クライアントリクエストトランスポートを呼出すスレッドはクライアントレスポンストランスポートを呼出し得る、または、異なるスレッドが、クライアントリクエストトランスポートを呼出すスレッドのリターン前または後に、クライアントレスポンストランスポートを呼出し得る。

【００５４】

・プロバイダリクエストを呼出すスレッドはプロバイダレスポンスを呼出し得る、または、異なるスレッドが、プロバイダリクエストを呼出すスレッドのリターン前または後に、プロバイダレスポンスを呼出し得る。

【００５５】

ある実施の形態に従い、バッファリングがイネーブルされた状態で、バッファリングされねばならないリクエストまたはレスポンスフローは、バッファリングサブシステムにおいて終了し、その後、バッファリングサブシステムのスレッド（たとえばＭＤＢのためのワークマネージャ）がこのフローを完了する。言い換えると、バッファリングポイントを追加するために修正された上記ルールはすべて、その他の追加なしで引続き有効である。

【００５６】

メタデータアクセス

ある実施の形態に従い、エンドポイントは、メッセージの一部である、リクエスト毎のメタデータ（「リクエストコンテキスト」として知られている）におけるメタデータリクエストを示すことによって、メタデータ（すなわちＷＳＤＬドキュメントおよびＸＳＤ双方）をコーラーが利用できるようにすることが可能である。リクエストコンテキストは以下のような特性を有し得る。

【００５７】

・TransportPropertySet.TRANSPORT_REQUEST_PRESENT_PROPERTY = 偽（false）

・TransportPropertySet.TRANSPORT_QUERY_PROPERTY = <何らかの値>

・TransportPropertySet.TRANSPORT_METADATA_BASEADDRESS_PROPERTY = <メタデータドキュメントのＵＲＬベースアドレス>

永続性およびクラスタリング

ある実施の形態に従い、ＤＩＳＩ永続性およびクラスタリングは、リクエスト／レスポンスコンテキストと、クライアントリクエストトランスポート、クライアントレスポンストランスポート、ディスパッチャレスポンス、プロバイダリクエスト、プロバイダレスポンス、およびサービスレスポンストランスポートオブジェクトのシリアライズに基づき得る（クライアントリクエストトランスポートおよびプロバイダリクエストのシリアライズは、ＳＯＡＰ処理中にバッファリングが生じるように構成されている場合に限り必要である）。

【００５８】

ユースケース

ある実施の形態に従い、プロバイダリクエストの実装は、プロバイダレスポンスオブジェクトと、リクエストからの必要なアーギュメント（arguments）を、シリアライズすることができる。その後、バッチプロセス等の別のプロセスが完了したときに、プロバイダレスポンスをデシリアライズし呼出すことができる。元のリクエストが発生したマシンと同じクラスタ内の任意のマシンから、またはサーバの再スタート後に、プロバイダレスポ

ンスをデシリアライズし呼出してもよい。

【００５９】

Web Servicesを使用する実装において、Web Servicesランタイムは、非匿名のサービスレスポンストランスポートオブジェクトをシリアライズして、後にこのオブジェクトを用いて非匿名のレスポンスを送信することができる。これが起こり得る理由は、バッファリングポイント（たとえば非同期のWeb Services Reliable Messaging、ＷＳ－ＲＭの使用）、または、プロバイダレスポンスオブジェクトがシリアライズされたことである。バックチャネルサービスレスポンストランスポートオブジェクトがシリアライズされるとは予測されない。なぜなら、バックチャネルをサポートするトランスポートは、バックチャネルレスポンスの永続性またはクラスタリングをサポートしないからである。

【００６０】

ある実施の形態に従い、クライアントリクエストトランスポートの実装は、クライアントレスポンストランスポートオブジェクトをシリアライズして、クライアントレスポンストランスポートオブジェクトをサーバの再スタート後にまたはクラスタ内の別のマシンで呼出すことができるようにしてもよい。これは、非匿名のアドレス指定を用いる非同期のレスポンスハンドリングの１つの可能な実装である。

【００６１】

Web Servicesを使用する実装において、Web Servicesランタイムは、ディスパッチャレスポンスオブジェクトをシリアライズして、後にこのオブジェクトを呼出してアプリケーションレスポンスを配信することができる。これが起こり得る理由は、バッファリングポイント、または、クライアントレスポンストランスポートオブジェクトがシリアライズされたことである。ある実施の形態に従うと、ＤＩＳＩは、シリアライズされたコールバックオブジェクトのライフサイクルを管理するためのモデルを定義しない。Web Servicesバッファリングの実装は、格納するオブジェクトに対しこれらの機能を提供することができる（たとえば、ＷＳ－ＲＭは、失効または終了した遣り取りに関連するシリアライズされたオブジェクトを含む永続データを削除することができる）。

【００６２】

構成

ある実施の形態に従い、ＤＩＳＩサービスエンドポイントおよびクライアントのすべての構成は、ＪＡＸ－ＷＳ標準ＡＰＩ、ＤＩＳＩ　ＡＰＩ、およびＤＩＳＩ固有のWebServiceFeatureクラス（すなわち構成ビーンズ（beans））を用いてプログラムできる。これらＡＰＩは、それぞれ標準サービスおよびエンドポイントクラスから、ならびに標準WebServiceFeatureクラスたとえばMTOMFeatureおよびAddressingFeatureから得られる、ＤＩＳＩのサービスおよびエンドポイントクラス上で利用できるビーン特性を含み得る。ほとんどのＷＳ－＊特徴について標準WebServiceFeatureクラスはない。所有WebServiceFeatureクラスを、下にあるＳＯＡＰスタックがこれらクラスを理解するのであれば、ＤＩＳＩに与えてもよい。ある実施の形態に従うと、ＤＩＳＩは、ＤＩＳＩベースのエンドポイント（すなわちサービスサイド）を動的に再構成するために収容環境が使用できるEndpoint.update　ＡＰＩを有する。たとえば、ＯＳＢはＯＷＳＭを用いてウェブサービスを管理してもよい。ＯＷＳＭから変更の通知が届いたときに、Endpoint.updateを新たな構成で呼出す。

【００６３】

リクエストおよびレスポンスコンテキスト

ある実施の形態に従い、ＤＩＳＩリクエストまたはレスポンスに関するコンテキスト（メッセージ以外のデータ）は、Map<String, Object>コンテキストオブジェクトのインスタンスを用いて送ることができる。使い易くするために、ＤＩＳＩは、それぞれ標準サーブレットリクエストおよびレスポンスオブジェクトからのリクエストおよびレスポンスコンテキストを構成するためのアダプタクラス、ServletContextAdapterを与える。

【００６４】

サービスリクエストトランスポート

表１は、ある実施の形態に従う、ServiceRequestTransport.request()についての、リクエストコンテキストキー／値の対を説明する。

【００６５】

【表１】

キー	データタイプ	デフォルト	説明
RequestHeadersPropertySet.REQUEST_HEADERS_PROPERTY	Map<String, List<String>>	null	トランスポートレベルリクエストヘッダ。マップはリクエストヘッダ名から値の順序付きリストまで。
JavaEESecurityPropertySet.USER_PRINCIPAL_PROPERTY	Accessor<Principal>	null	ユーザプリンシパル。多くの技術はリクエスト処理中まではユーザプリンシパルを与えることができないので、値はアクセサ（Accessor）でラッピングされる。
JavaEESecurityPropertySet.ROLE_PROPERTY	RoleAccessor	null	ユーザロール
TransportPropertySet.TRANSPORT_SECURE_PROPERTY	Boolean	false	トランスポートが安全、たとえば暗号化されているまたは機密であることを示す。
TransportPropertySet.TRANSPORT_CLIENTCERTS_PROPERTY	X509Certificate	null	トランスポートによって配信されるクライアント証明書
TransportPropertySet .TRANSPORT_REQUEST_PRESENT_PROPERTY	Boolean	true	トランスポートレイヤにリクエストがあるか否かを示す。たとえばHTTP GETメソッドの使用はこの特性がfalseである一例であろう。
TransportPropertySet .TRANSPORT_RESPONSE_EXPECTED_PROPERTY	Boolean	true	トランスポートレイヤによりレスポンスが予測されるか否かを示す。たとえばHTTP HEADメソッドの使用はこの特性がfalseである一例であろう。
TransportPropertySet .TRANSPORT_BASEADDRESS_PROPERTY	String	null	エンドポイントマッピングが関係するベースアドレス。この値は、所与のエンドポイントのマッピングと組合されるときは有効なURL、またはWSDLもしくはスキーマのクエリアドレスでなければならない。この値は現在のクライアントリクエストのコンテキストの中になければならない。
TransportPropertySet .TRANSPORT_METADATA_BASEADDRESS_PROPERTY	String	TRANSPORT_BASEADDRESS_PROPERTY propertyの値	エンドポイントマッピングが関係するメタデータのベースアドレス。この値は、所与のエンドポイントのマッピングと組合されるときは有効なURL、または、WSDLもしくはスキーマのクエリアドレスでなければならない。この値は現在のクライアントリクエストのコンテキストの中になければならない。この特性の１つのユースケースは、トランスポートポリシーが機密性を要求するときであるが、WSDLはクリアなチャネルでアクセスされてもよい。
TransportPropertySet.TRANSPORT_PATH_PROPERTY	String	null	クライアントがこのリクエストを行なったときに送ったURLに関連する他の経路情報。これはエンドポイントマッピングアドレスに続く経路情報であるがクエリストリングに先行する。
TransportPropertySet.TRANSPORT_QUERY_PROPERTY	String	null	クライアントがこのリクエストを行なったときに使用したURLのクエリストリング部分。

Table 1

【００６６】

サービスレスポンストランスポート

表２は、ある実施の形態に従う、ServiceResponseTransport.response()およびServiceResponseTransport.fail()についての、レスポンスコンテキストキー／値の対を説明する。

【００６７】
【表２】

キー	データタイプ	デフォルト	説明
ResponseHeadersPropertySet .RESPONSE_HEADERS_PROPERTY	Map<String, List<String>>	null	トランスポートレベルレスポンスヘッダ。マップはレスポンスヘッダ名から値の順序付きリストまで。
ResponseMetadataPropertySet .CONTENTTYPE_PROPERTY	String	null	レスポンスのコンテンツタイプ
ResponseMetadataPropertySet .RESPONSE_AVAILABLE_PROPERTY	Boolean	false	レスポンスメッセージまたはfaultを送信できることを示す。レスポンスを使用できない場合、これはトランスポートが特定のメッセージなしでステータスをコーラーに提供しなければならないことを示す（たとえばHTTPの場合はHTTP202、401、403、404、または 415 等）。
ResponseMetadataPropertySet .RESPONSE_ISFAULT_PROPERTY	Boolean	false	レスポンスがfaultまたはエラー状態であることを示す。
ResponseMetadataPropertySet .RESPONSE_ERRORSTATUS_PROPERTY	ResponseMetadataPropertySet.ErrorStatus	NOT_FOUND	レスポンスを使用できないときにのみ使用され、レスポンスはfault（すなわちエラー状態）である。エラーのタイプを示す。
ResponseMetadataPropertySet .RESPONSE_TARGETENDPOINT_PROPERTY	String	*匿名*	非匿名のレスポンスに対するターゲットエンドポイントアドレス

Table 2

【００６８】
プロバイダリクエスト
表３は、ある実施の形態に従う、ProviderRequest.request()についての、リクエストコンテキストキー／値の対を説明する。
【００６９】
【表３】

キー	データタイプ	デフォルト	説明
RequestMetadataPropertySet .CONTENTTYPE_PROPERTY	String	null	リクエストのコンテンツタイプ
RequestMetadataPropertySet .REQUEST_SOAPACTION_PROPERTY	String	null	リクエストに対するＳＯＡＰアクション

Table 3

【００７０】
プロバイダレスポンス
表４は、ある実施の形態に従う、ProviderResponse.response()についての、リクエストコンテキストキー／値の対を説明する。
【００７１】

【表４】

キー	データタイプ	デフォルト	説明
ResponseMetadataPropertySet .CONTENTTYPE_PROPERTY	String	null	レスポンスのコンテンツタイプ
ResponseMetadataPropertySet .RESPONSE_SOAPACTION_PROPERTY	String	null	レスポンスに対するＳＯＡＰアクション

Table 4

【００７２】

ディスパッチャリクエスト

表５は、ある実施の形態に従う、DispatcherRequest.request()についての、リクエストコンテキストキー／値の対を説明する。

【００７３】

【表５】

キー	データタイプ	デフォルト	説明
RequestMetadataPropertySet .CONTENTTYPE_PROPERTY	String	null	リクエストのコンテンツタイプ
RequestMetadataPropertySet .REQUEST_SOAPACTION_PROPERTY	String	null	リクエストに対するＳＯＡＰアクション

Table 5

【００７４】

ディスパッチャレスポンス

表６は、ある実施の形態に従う、DispatcherResponse.response()についての、リクエストコンテキストキー／値の対を説明する。

【００７５】

【表６】

キー	データタイプ	デフォルト	説明
ResponseMetadataPropertySet .CONTENTTYPE_PROPERTY	String	null	レスポンスのコンテンツタイプ
ResponseMetadataPropertySet .RESPONSE_SOAPACTION_PROPERTY	String	null	レスポンスに対するＳＯＡＰアクション

Table 6

【００７６】

クライアントリクエストトランスポート

表７は、ある実施の形態に従う、ClientRequestTransport.request()についての、リクエストコンテキストキー／値の対を説明する。

【００７７】

【表７】

キー	データタイプ	デフォルト	説明
RequestMetadataPropertySet .CONTENTTYPE_PROPERTY	String	null	リクエストのコンテンツタイプ
RequestMetadataPropertySet .REQUEST_SOAPACTION_PROPERTY	String	null	リクエストに対するＳＯＡＰアクション

Table 7

【００７８】

クライアントレスポンストランスポート

表８は、ある実施の形態に従う、ClientResonseTransport.response()についての、リ

クエストコンテキストキー／値の対を説明する。

【００７９】

【表８】

キー	データタイプ	デフォルト	説明
ResponseMetadataPropertySet .CONTENTTYPE_PROPERTY	String	null	レスポンスのコンテンツタイプ
ResponseMetadataPropertySet .RESPONSE_SOAPACTION_PROPERTY	String	null	レスポンスに対するＳＯＡＰアクション

Table 8

【００８０】

本発明は、１台以上のプロセッサ、メモリ、および／または本開示の教示に従いプログラムされたコンピュータ読取可能な記憶媒体を含む、従来の汎用または専用デジタルコンピュータ、コンピューティングデバイス、マシン、またはマイクロプロセッサを１台以上用いて、適宜実現し得る。適切なソフトウェアコーディングは、熟練したプログラマが本開示の教示に基づいて容易に準備できる。これはソフトウェア技術の当業者には明らかであろう。

【００８１】

実施の形態によっては、本発明は、本発明のプロセスのうちいずれかを実行するためにコンピュータをプログラムするのに使用できる命令が格納された非一時的な記憶媒体または（１つまたは複数の）コンピュータ読取可能な媒体であるコンピュータプログラムプロダクトを含む。この記憶媒体は、フロッピー（登録商標）ディスク、光ディスク、ＤＶＤ、ＣＤ－ＲＯＭ、マイクロドライブ、および光磁気ディスクを含む、任意の種類のディスク、ＲＯＭ、ＲＡＭ、ＥＰＲＯＭ、ＥＥＰＲＯＭ、ＤＲＡＭ、ＶＲＡＭ、フラッシュメモリデバイス、磁気もしくは光カード、ナノシステム（分子メモリＩＣを含む）、または、命令および／またはデータを格納するのに適した任意の種類の媒体もしくはデバイスを含み得るものの、これらに限定されない。

【００８２】

本発明に関するこれまでの記載は例示および説明を目的として提供されている。すべてを網羅するまたは本発明を開示された形態そのものに限定することは意図されていない。当業者には数多くの変更および変形が明らかであろう。実施の形態は、本発明の原理およびその実際の応用を最もうまく説明することによって当業者が本発明のさまざまな実施の形態および意図している特定の用途に適したさまざまな変形を理解できるようにするために、選択され説明されている。本発明の範囲は、以下の特許請求の範囲およびその均等物によって定められることが意図されている。

【図１】

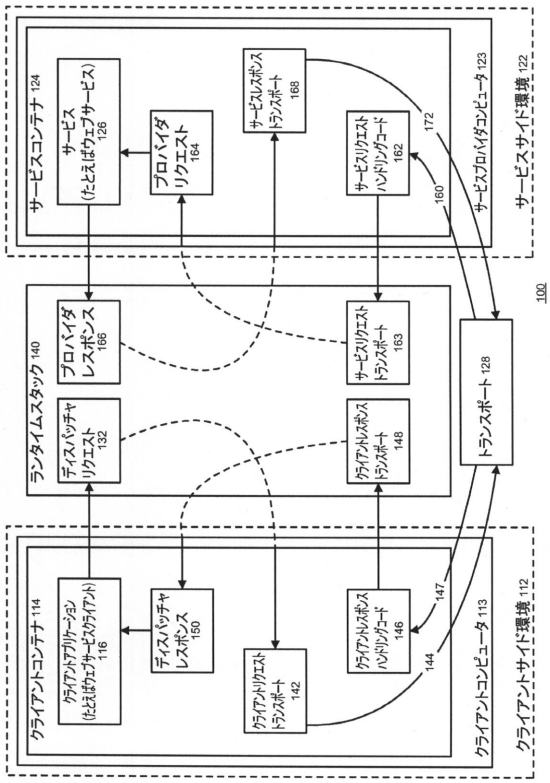

FIGURE 1

【図２】

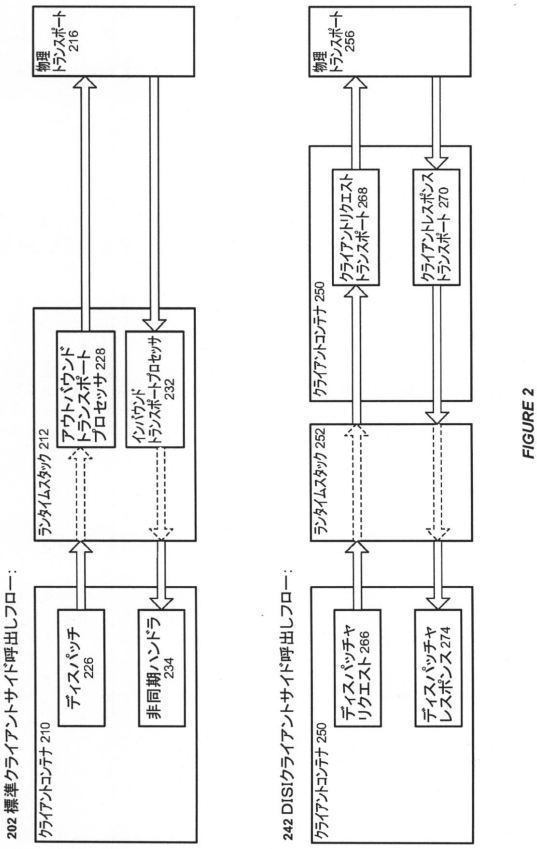

FIGURE 2

【図３】

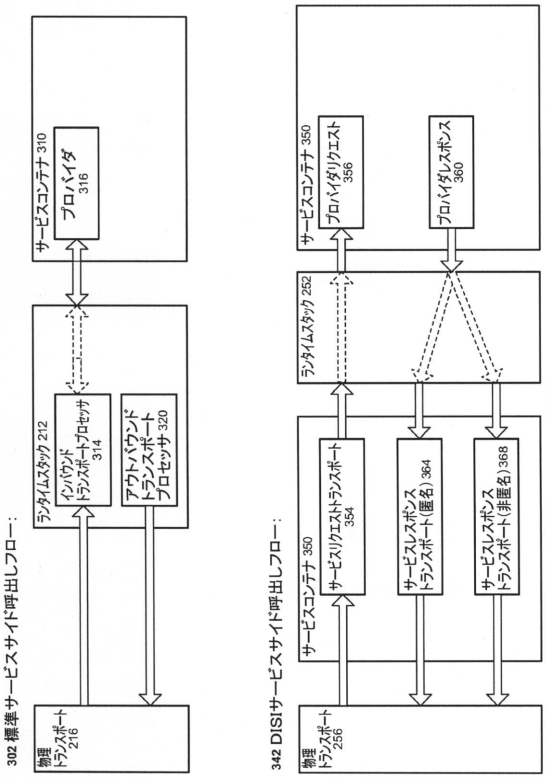

FIGURE 3

【図４】

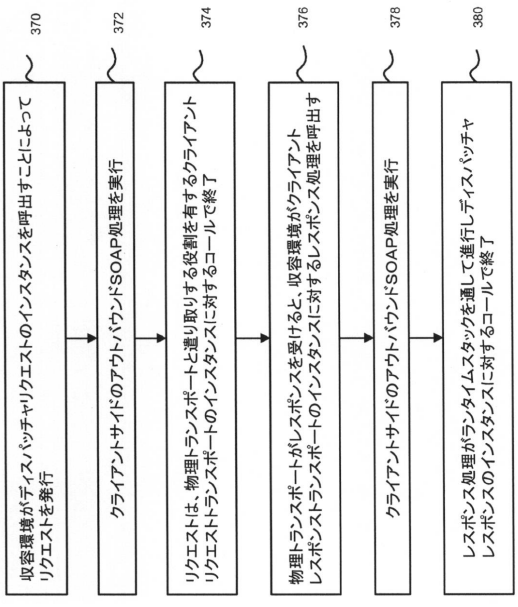

FIGURE 4

【図５】

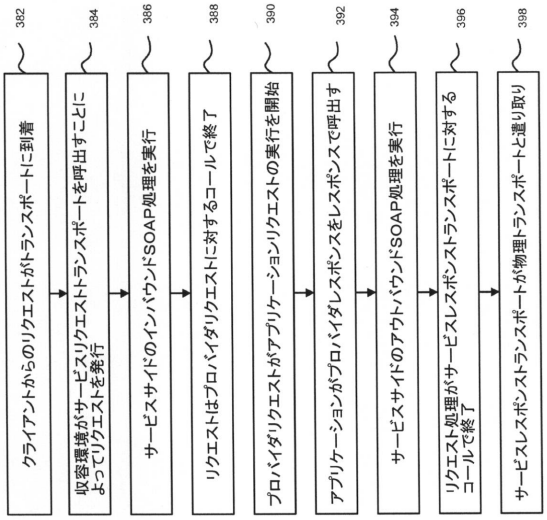
382
クライアントからのリクエストがトランスポートに到着
384
収容環境がサービスリクエストトランスポートを呼出すことによってリクエストを発行
386
サービスサイドのインバウンドSOAP処理を実行
388
リクエストはプロバイダリクエストに対するコールで終了
390
プロバイダリクエストがアプリケーションリクエストの実行を開始
392
アプリケーションがプロバイダレスポンスをレスポンスで呼出す
394
サービスサイドのアウトバウンドSOAP処理を実行
396
リクエスト処理がサービスレスポンストランスポートに対するコールで終了
398
サービスレスポンストランスポートが物理トランスポートと遣り取り

FIGURE 5

フロントページの続き

(72)発明者　エーベルハルト，リャン
アメリカ合衆国、１８４３１　ペンシルベニア州、ホーンズデール、トップ・オブ・ザ・ヒル・ドライブ、１１

審査官　田中　幸雄

(56)参考文献　特開２００３－１１４８０５（ＪＰ，Ａ）
特開２００６－８５３６５（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／５４