シームレスコンピューティングのための異種分散コンポーネントのPlug and Play環境

名倉正剛[†] 高田眞吾[††] 土居範久[†††]

近年，ユーザに意識させずにソフトウェアの動作する機器を自動的に連携させるために，“シームレスコンピューティング” という概念が注目されている．分散コンポーネント技術をシームレスコンピューティングに適応させるためには，コンポーネントが動作する機器をネットワークに接続するだけで，ネットワーク上のその他の機器と自動的に連携するという，いわゆる “Plug and Play” ができる必要がある．そのためには，満たさなければならない要件がいくつかある．本研究ではそれらのうち，コンポーネントを発見するための方法と，発見したコンポーネントを利用する際にコンポーネントの異種性を吸収するための方法に着目する．それらの要件を満たし，異種分散コンポーネントの存在するサーバやそれを利用するクライアントをネットワークに接続することによって，Plug and Play で動作させる環境を提案し，この環境を実現するシステムを実装する．

A Plug and Play Environment for Seamless Computing with Heterogeneous Distributed Components

Masataka Nagura,[†] Shingo Takada[††] and Norihisa Doi[†††]

Recently, “Seamless Computing”, which is a concept for automated integration of appliances executing many software, has attracted a great deal of attention. To accommodate technologies for distributed components to Seamless Computing, it is necessary to be able to “Plug and Play” appliances that execute component software through integrating automatically with other appliances attached to the network. Many requirements need to be satisfied to make this possible. In this work, we focus on the discovery of components and the integration of heterogeneous components. We propose an environment where servers with heterogeneous distributed components and clients using those components can be executed through Plug and Play. We implement a system to realize this environment.

1. はじめに

インターネットの発達や，端末の小型化，高機能化により，ユーザが端末をいろいろな場所に移動させて，その場所のネットワークに接続するような利用方法が増大している．従来は，ネットワークに接続される端末や利用されるアプリケーションの構成は，ある程度静的に決定されていた．しかし，ユーザの利用形態が多様化しており，動的に変更されうる環境での利用を考慮する必要が生じてきている．そのような中で，ユーザに意識させずにソフトウェアの動作する機器を自動的に連携させるために，“シームレスコンピューティング（Seamless Computing）” という概念が注目されている[1]．そのためには，それらの機器の Plug and Play を可能にしなければならない．

近年，EJB（Enterprise JavaBeans）[2]，CORBA（Common Object Request Broker Architecture）[3]，Web Service [4] などの分散コンポーネント技術が普及している．これらの分散コンポーネント技術は，従来のクライアントサーバシステム実現のための 1 つの手法として利用されている．そのため分散コンポーネント技術を利用して開発したシステムはクライアントとサーバによって構成され，コンポーネントはサーバ上に配置される．また EAI（Enterprise Application Integration）と呼ばれるアプリケーション開発アプローチにより，既存の分散コンポーネントを統合して新しいアプリケーションを開発することができる．そして，その際に開発を支援するための EAI システム[5)~7)] も

† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†† 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
††† 中央大学理工学部
Faculty of Science and Engineering, Chuo University

提案されている．これらは，サーバプログラムとして実行されるコンポーネントの処理を組み合わせることによってアプリケーションプログラムを統合する．

分散コンポーネントが動作する機器を利用してシームレスコンピューティングを実現するためには，分散コンポーネント技術を利用した Plug and Play の実現を考える必要がある．そしてそのためには，コンポーネントが動作する機器をネットワークに接続するだけで，ネットワーク上のその他の機器と自動的に連携できる必要がある．本研究では，分散コンポーネントを Plug and Play で動作させるために必要な要件のうち，以下の点に着目する．

- 必要なコンポーネントを発見するための方法
- 発見したコンポーネントを利用するための方法

Plug and Play で動作する環境を実現するためには，必要なコンポーネントが動作するコンピュータを自動的に発見できる必要がある．そして，いろいろなベンダによって提供されるコンポーネントを連携させるために，規格の異なるコンポーネント技術や異なるプラットフォームを利用して動作している複数の分散コンポーネント（本論文ではこれらを「異種分散コンポーネント」と呼ぶ）を連携させて利用できる必要がある．

本研究ではこれらの要件を満たし，コンポーネントの存在するサーバやそれを利用するクライアントを，Plug and Play で自動的に連携できる環境を提案し，それを実現するシステムを実装する．

まず 2 章で，分散コンポーネント技術を利用して Plug and Play 環境を実現する場合の問題点を分析し，3 章で関連研究を述べる．4 章で Plug and Play 環境を提案し，5 章でそれを実現するシステムを実装し，6 章で評価実験を行う．最後に 7 章で考察を行い，8 章でまとめる．

2. Plug and Play 環境実現のための問題点

本章では，まず従来の分散コンポーネントシステムの利用手順を述べる．その後，異種分散コンポーネントを利用した Plug and Play 環境を実現するための問題点として，分散して存在するコンポーネントを発見する際の問題と，利用する際の問題を分析する．

2.1 従来の分散コンポーネントの利用手順

EJB，CORBA，Web Service などの一般的な分散コンポーネント技術を利用したアプリケーションは，サーバプログラムとクライアントプログラムによって構成される．そして，コンポーネントが実行する処理をサーバプログラムとしてサーバコンピュータ上に配置する．

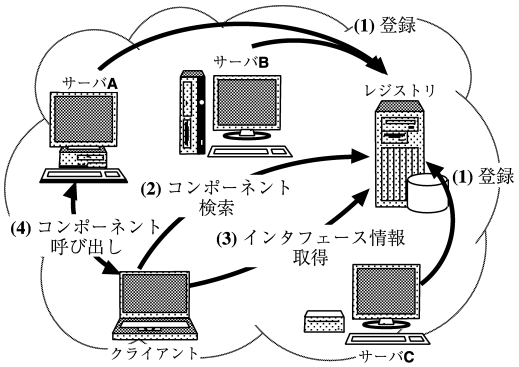

図 1 分散コンポーネントの呼び出し手順
Fig. 1 Calling sequences for distributed components.

分散コンポーネントを利用するためには，まずそのコンポーネントを提供するサーバを発見しなければならない．このために，コンポーネントの存在するサーバを登録しておくレジストリ[8),9)] を，組織や会社などの単位ごとに用意する．そして，それぞれのサーバで動作しているコンポーネントに関する情報を，あらかじめレジストリに登録しておく（**図 1**-(1)）．コンポーネントの利用者は，レジストリを検索し，利用したいコンポーネントと，それを提供するサーバを発見する（図 1-(2)）．

次に利用者は，外部からアクセスを行うために必要な，インタフェースに関する情報を入手する（図 1-(3)）．一般的な分散コンポーネントフレームワークでは，インタフェースに関する情報を記述するための言語が用意されている[10),11)]．そして，クライアントはインタフェースに関する情報を，レジストリやコンポーネントを提供するサーバから入手する．そして，入手した情報を利用し，利用者はコンポーネントを呼び出すプログラムを開発し，それを利用してコンポーネントを呼び出す（図 1-(4)）．

なお，本論文ではクライアントとサーバから構成される利用モデルを前提とするが，本研究で提案する Plug and Play 環境は利用モデルを限定しない．

2.2 Plug and Play 環境

現在，様々な分野にコンピュータが普及しており，自動車や家電などの新しい分野にもコンピュータを利用したシステムが展開している．さらにインターネットの発達により，それらの新しいタイプのコンピュータを簡単にネットワークに接続することができるようになってきている．したがって，ユーザが端末をいろいろな場所に移動させて，その場所のネットワークに接続するような利用方法が増大してきた．

ネットワークとそれに接続される端末の構成は，従来はある程度静的に決定されていたが，ユーザの利用形態が多様化したため，動的に変更されうる環境での利用を考慮する必要が生じてきている．

いろいろな端末が様々な場所のネットワークに動的に接続したり切断されたりする “Plug and Play” の環境を考えると，接続された端末は接続されたネットワークで利用可能な他の端末を探す必要がある．家電品を例にしてインターネットを利用してビデオを配信する環境を想定すると，ビデオプレイヤのクライアントを利用者が購入して自宅のネットワークに接続したときには，ネットワーク内に存在するビデオサーバを探す必要がある．一方で，映画配信サービスのストリーミングサービスのように，グローバルなネットワークのある特定の場所に存在するサーバについては，クライアントはどこのネットワークに接続しても同一のサーバを利用する．サービスの提供者も，グローバルで公開することを前提としているので，サーバの場所をそれほど頻繁に変更しないであろうし，グローバルなネットワークに存在するディレクトリサービスにその場所に関する情報を登録していることも十分に考えられる．したがってあらかじめサーバの場所を知っている場合は，クライアントにその場所を設定しておけばよく，存在を知らない場合でもグローバルなネットワークに存在するディレクトリサービスに問い合わせればよい．このため，ネットワークに接続されたクライアントが，ストリーミングサーバの存在を探す必要性は，ローカルネットワークのビデオサーバを探す必要性ほどは高くない．

分散コンポーネントとしてこれらの機器の機能を提供することを考えると，上記の例と同様に，自宅のネットワークのようなローカルなネットワークに存在するコンポーネントについては，クライアントが探して利用できることに対する必要性が高い．したがって本論文では，ローカルネットワークにおいて Plug and Play 環境を実現することを考える．

次節以降では，分散コンポーネントシステムを利用して，このような Plug and Play 環境を実現する場合の問題点を考える．

2.3 分散コンポーネントを発見する際の問題

Plug and Play 環境を実現するためには，クライアントやサーバがネットワークにプラグインした後，他の機器と連携するために，利用したり利用されたりするための設定を自動的に行う必要がある．

クライアントがプラグインした場合は，プラグインしたネットワークで利用したいコンポーネントとそれを提供するサーバを，レジストリから探す必要がある．また分散コンポーネントを提供するサーバがプラグインした場合は，自分自身をクライアントから発見できるようにするために，レジストリに登録する必要がある．どちらの場合も自動的に他の機器と連携するためには，ネットワーク上のどこにレジストリが存在しているかを知らなければならない．また，プラグイン先のネットワークに存在するすべてのサーバが，同一のレジストリに登録されているとは限らない．このため，利用したいコンポーネントを，どのレジストリが登録しているかも知らなければならない．しかし，プラグインする端末は接続されるネットワークを特定できないので，それらを事前に知ることができない．したがって，連携させるための設定を自動化することは困難である．

さらに，ネットワークへの接続や切断が頻繁に発生することも想定する必要がある．仮にレジストリの存在を把握できても，クライアントはそのつどレジストリから必要なコンポーネントを検索し，サーバはそのつどレジストリへ登録や削除を行うことになる．したがって，ネットワークの構成が頻繁に変化する場合は，レジストリへの負荷が大きくなる．

2.4 分散コンポーネントを利用する際の問題

Plug and Play 環境では，どのようなサーバ上で動作するコンポーネントを利用するかを事前に特定できない．そのため，様々なコンポーネントを連携できる必要がある．それぞれのベンダはそれらを独自に開発しているため，必ずしも同一の分散コンポーネント技術やプラットフォーム実装を利用している保証がない．これらの異種分散コンポーネント間では，呼び出しの方法や，メッセージ，返戻値の形式が異なる可能性がある．このように分散コンポーネント間に異種性が存在する場合は，相互に連携できない場合がある．

3. 関 連 研 究

EAI システム[5)〜7)] を利用することによって，開発者はワークフローに関する情報を記述することで，複数のコンポーネントを統合して動作させるアプリケーションを開発する．開発者は組み合わせるコンポーネントがどれであるかを事前に想定し，それらを組み合わせるようにソフトウェアを開発する．したがってソフトウェアの利用者は，各コンポーネントやそれが動作するサーバの存在を意識することになる．さらに，EAI システムは一般的に分散コンポーネントのプラットフォーム実装ごとに用意されているため，異種分散コンポーネントを統合できない．本章ではこれらの問

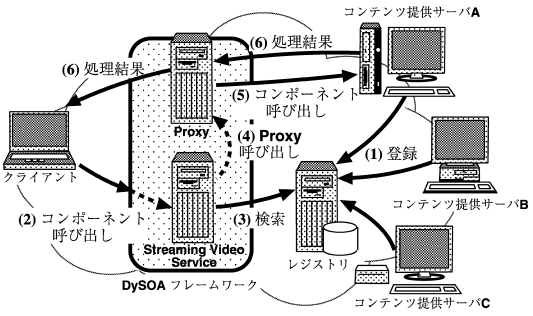

図 2 DySOA フレームワークを利用したアプリケーション
Fig. 2 An application using DySOA framework.

題を解決するための，既存の技術を取り上げる．

分散コンポーネントの動的発見

Web Service を利用するクライアントから，利用するコンポーネントの存在を意識させなくするための技術として，DySOA [12] が提案されている．これは，利用するコンポーネントを動的に発見し QoS に応じて切替えを行う．DySOA を利用して開発したビデオ配信システムの例を，**図 2** に示す．Streaming Video Service（以下，SVS と略記する）と Proxy は，DySOA フレームワークを利用して動作する．SVS はクライアントからのリクエストを受け取った際に，必要なコンテンツを配信するすべてのコンポーネントと，それを提供するサーバをレジストリから探す．そして，Proxy によって QoS に応じて一番適切なサーバへ処理を仲介する．

DySOA を利用することで，クライアントから呼び出しを行う段階で，適切なコンポーネントをレジストリから探して利用できる．DySOA プラットフォームがコンポーネントを探すので，クライアントはコンポーネントの存在を意識する必要がない．しかし，DySOA プラットフォーム上で動作するシステム（上の例における SVS）がコンポーネントの存在を管理しているので，クライアントはシステム（SVS）の存在を知らなければならない．またサーバを追加した場合に，追加されたサーバがレジストリを発見したり，登録したりする手段もない．Plug and Play を実現するためには，コンポーネントの存在を管理するための装置を設けずに，コンポーネントを利用時に探索できる必要がある．

汎用的なサービス探索プロトコル[13),14)] では，クライアントが IP マルチキャストを利用してサーバ群にリクエストを送信することで，必要なサーバを探索できる．Web Service にも，同様の方法でサーバを探索するために WS-Discovery [15] が提案されている．これは，Web Service を利用するためのプロトコルである SOAP（Simple Object Access Protocol）を用いて探索を行う．

異種分散コンポーネントの統合

異種分散コンポーネントを統合する技術としては，統合するアプリケーションの開発を支援するシステム[16)] や結合を行う部分を記述する言語 Lua [17)] が提案されている．これらは，呼び出そうとするコンポーネントごとに異種性を吸収するためのプログラムコードを事前に記述することで，コンポーネントを統合する．

Plug and Play 環境ではネットワークの構成が頻繁に変化するため，連携させるコンポーネントを事前に決定できない．異種性を吸収するためのプログラムコードを，連携させる可能性のあるすべてのコンポーネントに対して用意すれば，このような環境でもコンポーネントを統合できる．しかし組み合わせる対象となるコンポーネントが多いので，このような方法で異種性を吸収することは困難である．

4. 異種分散コンポーネントの Plug and Play 環境

本章では，本研究で提案する Plug and Play 環境について述べる．

4.1 概　　要

本研究では，分散コンポーネントの存在するサーバやそれを利用するクライアントをネットワークに接続することで，それらを Plug and Play で動作させることのできる環境を提案する．コンポーネントを利用するクライアントや，コンポーネントを統合することによって連携を行うサーバは，提案する環境を利用し，必要とするコンポーネントとそれを提供するサーバを利用時に自動的に発見する．そしてコンポーネントの種類によらずに統合する．したがって，それらのクライアントやサーバだけでなく，コンポーネントを提供するサーバも含め，分散コンポーネントシステムを構成するすべての端末を Plug and Play で動作させる．

提案する環境は，コンポーネントの存在を管理する特別な機構を設けずに，コンポーネントを利用時に探索することを可能にする．これにより，2.3 節で述べた分散コンポーネントを発見する際の問題を解決する．そして，どのようなソフトウェアが連携されるのかを事前に把握できなくても，探索して発見したコンポーネントを，それらの異種性によらずに統合して動作させることを可能にする．これにより，2.4 節で述べた分散コンポーネントを利用する際の問題を解決する．

本章では，まず利用シナリオを示すことで，提案する環境の概観について述べる．その後で，コンポーネ

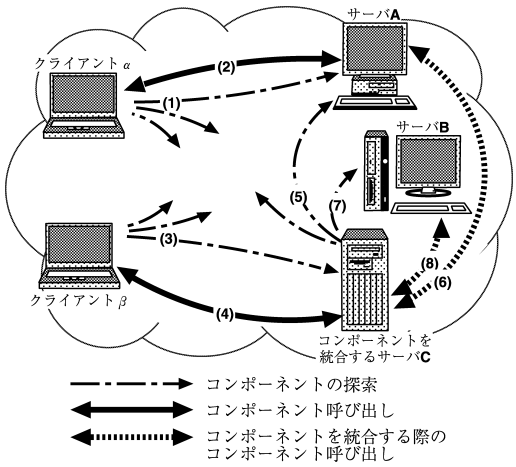

図 3 利用シナリオ
Fig. 3 Basic scenarios of our proposed environment.

ントを探索する方法と，統合する方法を示す．

4.2 提案する環境の利用シナリオ

図 3 に，提案する環境の利用シナリオを 2 つ示す．

まず，クライアント α がネットワークにプラグインしたときに，利用したいコンポーネントを，サーバ A が提供しているとする．クライアント α は，コンポーネントを呼び出す前に，必要とするコンポーネントを探索し，サーバ A を発見する（図 3-(1)）．そして，サーバ A を呼び出す（図 3-(2)）．4.3.1 項で，この発見と呼び出しの詳細について述べる．

また，クライアント β がネットワークにプラグインしたときに，利用したいコンポーネントを，サーバ C が提供しているものとする．ただし先ほどの例とは異なり，サーバ C にはコンポーネントを統合するアプリケーションが動作していて，サーバ C が統合したいコンポーネントをサーバ A とサーバ B が提供しているとする．コンポーネントを統合するサーバ C は，クライアントから見ればコンポーネントのサーバとして振る舞う．クライアント β は，クライアント α の場合と同様に，コンポーネントを呼び出す前に必要とするコンポーネントを探索し，サーバ C を発見して（図 3-(3)）それを呼び出す（図 3-(4)）．サーバ C は統合すべき各コンポーネントを探索して呼び出す．まずクライアントの場合と同様に，コンポーネントを探索する．その結果サーバ A を発見し（図 3-(5)），それを呼び出す（図 3-(6)）．次に同様に，サーバ B を発見し（図 3-(7)），それを呼び出す（図 3-(8)）．そしてその結果をクライアント β に返す．4.3.2 項で，この詳細を述べる．

4.3 コンポーネントの探索

提案する環境は，コンポーネントを呼び出す時点でコンポーネントを提供するサーバを発見する．そのために以下のメッセージを用意する．

(A) コンポーネント探索メッセージ（コンポーネント要求/応答メッセージ）
コンポーネントを探索し，発見するためのメッセージ．

(B) パラメータ探索メッセージ（パラメータ要求/応答メッセージ）
コンポーネントを呼び出すためのインタフェースに関する情報を入手するためのメッセージ．なお対象とする分散コンポーネントアーキテクチャによっては，インタフェースに関する情報の取得を独自の方法で行う場合がある．その場合は，インタフェースに関する情報の取得のために必要な情報を応答メッセージに含む．

(C) コンポーネント広告メッセージ
コンポーネントの存在を示すためのメッセージ．

コンポーネントを呼び出すためには，実装されているコンポーネントのアーキテクチャに依存した情報が必要になる．たとえば Web Service の場合は，サービスの名前や，WSDL [11] で記述されたインタフェースの情報を入手するための URL や，コンポーネントサーバを利用する際の，プログラムのエンドポイントのパスなどである．提案する環境では実装されているコンポーネント技術を特定せずに，いろいろな種類のサーバで動作するコンポーネントをクライアントから利用できるようにする．そのためクライアントは，同一の処理を提供するいろいろな種類のサーバを発見しうる．しかし，クライアントはすべての種類のコンポーネントに関する情報を必要としないかもしれない．そこで実装されているコンポーネントのアーキテクチャに依存した情報を，コンポーネント探索とは別個のパラメータ探索メッセージとして設計する．

まず 4.3.1 項で，クライアントがコンポーネントを探索して利用する際の手順を述べる．そして 4.3.2 項で，コンポーネントを統合するサーバから必要なコンポーネントを探索して利用する際の手順を述べる．

4.3.1 単一のコンポーネントを利用する場合

4.2 節に示したシナリオにおいて，クライアント α がコンポーネントを探索し，利用する場合のメッセージシーケンスを**図 4** に示す．

まずクライアント α は，利用したいコンポーネントが同一ネットワーク上に存在するかどうかを調べるために，コンポーネント要求メッセージを送信する．こ

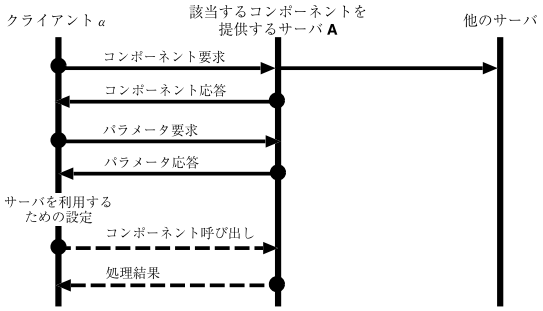

図 4 コンポーネントの探索と利用
Fig. 4 Component discovery and invocation.

のメッセージには，呼び出したいコンポーネントを指定するための文字列を含む．そして，コンポーネントを提供するすべてのサーバから必要なコンポーネントを探索するために，マルチキャストでネットワーク上に送信する．該当するコンポーネントを提供するサーバ A は，クライアント α にコンポーネント応答メッセージを返す．これにより，クライアント α は必要なコンポーネントを提供するサーバ A を発見する．呼び出したいコンポーネントを提供するサーバを複数発見した場合は，1 つを選ぶ．

次にクライアント α は，サーバ A にパラメータ要求メッセージを送信する．サーバ A はパラメータ要求メッセージを受信し，コンポーネントを呼び出すために必要なインタフェース情報を，パラメータ応答メッセージとしてクライアント α に返す．そしてクライアント α はその情報を利用して，コンポーネントを呼び出す．

なお，コンポーネントを提供するサーバがコンポーネント広告メッセージを定期的にマルチキャストで送信することによって，クライアントにコンポーネントの存在を通知することもできる．

4.3.2 コンポーネントを統合して利用する場合

コンポーネントを統合するアプリケーションを利用する際には，コンポーネントを統合するサーバが必要なコンポーネントを探索して統合する．これによって，その時点で利用できるコンポーネントを連携させる．4.2 節に示したシナリオにおいて，クライアント β がコンポーネントを統合するサーバ C を探索して利用する場合のメッセージシーケンスを**図 5** に示す．この際，サーバ C は必要なコンポーネントを提供するサーバ A とサーバ B を探索して利用する．

まずクライアント β は，4.3.1 項の場合と同様に，呼び出したいコンポーネントを探索する．そのために，コンポーネント要求メッセージをマルチキャストで送信する．コンポーネントを統合するサーバ C は，要

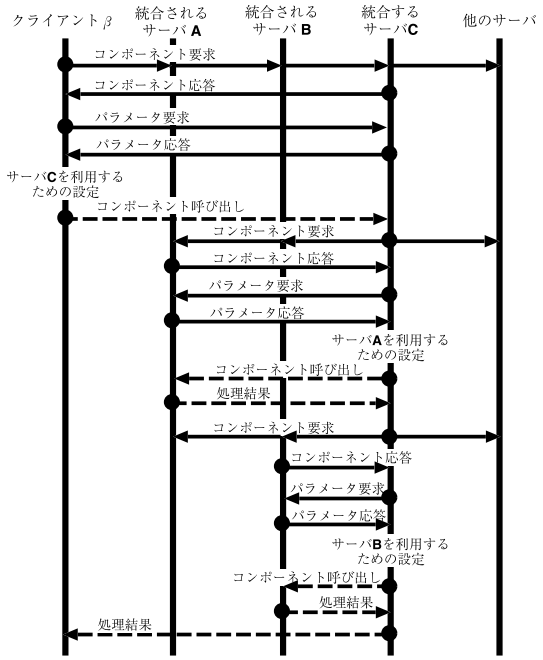

図 5 コンポーネントの統合
Fig. 5 Component integration.

求の内容が提供するアプリケーションに該当する場合は，クライアント β にコンポーネント応答メッセージを返す．クライアント β は応答メッセージに従い，サーバ C にパラメータの要求を行う．そしてコンポーネントを呼び出すために必要な情報を，パラメータ応答メッセージとして受信する．

次にクライアント β は，受信したパラメータ応答メッセージに含まれる情報を利用して，コンポーネントを統合するアプリケーションを呼び出す．コンポーネントの統合は，外部の複数のコンポーネントへの呼び出しを行った結果を組み合わせることによって行う．サーバ C は，それぞれのコンポーネントへの呼び出しの際に，4.3.1 項と同じ手順でコンポーネントを探索して利用する．そして，必要なすべてのコンポーネントを呼び出し，処理を行った結果をクライアント β に返す．

なお 4.3.1 項の場合と同様に，コンポーネントを統合するサーバは，コンポーネント広告メッセージを定期的にマルチキャストで送信することもできる．さらに，各コンポーネントを提供するサーバからコンポーネント広告メッセージを定期的にマルチキャストで送信することによって，コンポーネントを統合するサーバが，必要なコンポーネントの存在をあらかじめ知ることもできる．

4.4 異種分散コンポーネントの統合

提案する環境では，探索したコンポーネントをその種類によらずに，呼び出せるようにする．

3 章で述べたように，従来の研究[16),17)] では，異種性を吸収するためのプログラムコードを，呼び出そうとするコンポーネントごとに事前に用意しなければならないことが問題であった．

そこで呼び出しの形式の相違を，コンポーネントを統合するアプリケーションやクライアントのプログラムを実行するプラットフォームで吸収する．そのために，種類の異なるコンポーネントへの呼び出しの際に，呼び出しの方法やメッセージや返戻値の形式を変換できるようにする．詳細については 5.3 節で述べる．

5. 実　　装

本章では，4 章で提案した環境を実現するシステムの実装について述べる．

5.1 システム概要

以下の 5 つの部分から構成するように実装した．

- コンポーネント要求部
  コンポーネント要求メッセージや，パラメータ要求メッセージを送信することによって，コンポーネントを提供するサーバを探索し，インタフェース情報を取得する部分
- コンポーネント応答部
  コンポーネント要求部からの要求メッセージに対し，コンポーネント応答メッセージやパラメータ応答メッセージを送信することによって応答し，インタフェース情報を提供する部分
- コンポーネント広告部
  コンポーネント広告メッセージによって，コンポーネントの存在を通知する部分
- メディエータ部
  探索した結果得られたコンポーネントに対して，コンポーネント技術や実装による異種性を吸収して，呼び出しを行う部分
- 統合アプリケーション運用部
  コンポーネントを統合するアプリケーションを運用する部分

実装したシステムは，EJB，Web Service，CORBA の 3 種類のコンポーネントを扱うことができる．他の種類のコンポーネント技術で実装されたコンポーネントに関しても，モジュールを追加することで組み合わせることができる．また，コンポーネントの統合には，BPEL4WS（Business Process Execution Language for Web Services）仕様[18)] を利用した．これは Web Service の連携を行う際の XML ベースの言語であり，ワークフローに基づいてコンポーネントを統合する．

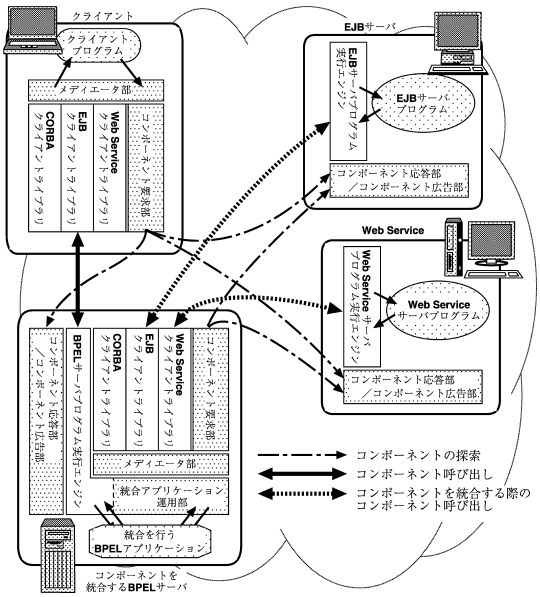

図 6 コンポーネントを統合するシステムの構成
Fig. 6 System structure for component integration.

実装したシステムを利用する際の構成を，**図 6** に示す．この図では，複数のコンポーネントを統合するサーバをクライアントが利用している．そしてコンポーネントを利用するクライアントと，コンポーネントを提供するサーバと，コンポーネントを統合するサーバから構成されている．

クライアントにはコンポーネントを探索して利用できるように，コンポーネント要求部とメディエータ部を用意する．そしてコンポーネントを提供するサーバやコンポーネントを統合するサーバには，クライアントから探索できるように，コンポーネント応答部を用意する．またコンポーネントを統合するサーバは，BPEL4WS に対応した既存のサーバを拡張し，統合アプリケーション運用部やメディエータ部を組み込むことによって用意する．

5.2 コンポーネント探索プロトコル

提案した環境は，それぞれのコンポーネントが利用する分散コンポーネント技術や実装によらずに，連携させることを可能にする．そのためコンポーネントを探索する仕組みも，コンポーネントが実装されているアーキテクチャに依存させずに実現する必要がある．そこで本研究では，4.3 節で述べたそれぞれのメッセージをやりとりする独自の探索プロトコルを設計し，それによって探索を行う機構を実装した．なおこのプロトコルの実装の際にはマルチキャストでメッセージを

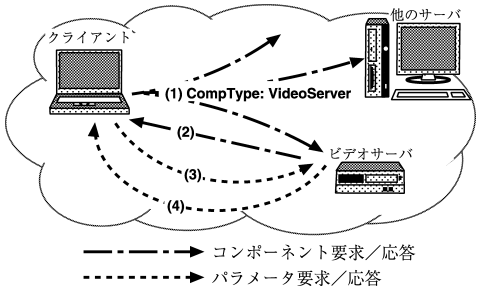

図 7 コンポーネントの探索の例
Fig. 7 An example of component discovery.

やりとりする必要があるが，この部分は IP マルチキャストを用いて実装した．

3 章であげた WS-Discovery は，利用するプロトコル自体に SOAP を用いており，Web Service が利用するプロトコルに依存している．また同様に 3 章であげた汎用的なサービス探索プロトコル[13),14)] は，情報を探索するためのメッセージのやりとりの一般的な方法を規定しており，これを拡張することによって実現することも可能である．しかし拡張する部分が多すぎ，結果として冗長性の高いものになってしまう．

提案機構でコンポーネントを探索する場合の例を，**図 7** に示す．

コンポーネントを利用したいクライアントはまず，コンポーネント要求部から，コンポーネント要求メッセージを IP マルチキャストを利用して送信する（図 7-(1)）．コンポーネント要求メッセージには，利用したいコンポーネントのタイプを示す文字列を含む．たとえば，2.2 節の例におけるビデオサーバを探索したい場合には，“CompType:VideoServer” のように，提供するサービスを指定する文字列を含む．また必要ならば，動作するプラットフォームや，サーバ実装の種類を指定することもできる．

コンポーネントを提供するサーバや，コンポーネントを統合するサーバは，コンポーネント応答部によってコンポーネント要求メッセージを受け取る．そしてコンポーネント要求メッセージに含まれるコンポーネントのタイプに一致する場合は，コンポーネント応答部によってコンポーネント応答メッセージを返す（図 7-(2)）．このメッセージには，動作するプラットフォームの種類や，サーバ実装の種類を含む．

コンポーネント応答メッセージを受け取ったクライアントは，コンポーネント要求部によって，パラメータ要求メッセージを送信する（図 7-(3)）．該当するサーバのコンポーネント応答部は，パラメータ要求メッセージを受け取り，実装されているコンポーネントのアーキテクチャに依存した情報を応答として返す（図 7-(4)）．したがってパラメータ応答メッセージの詳細はコンポーネントのアーキテクチャにより異なるが，それぞれのサーバ内でコンポーネントを識別するための名前と，メソッドのシグネチャを含む．

また，コンポーネントを提供するサーバや，コンポーネントを統合するサーバは，コンポーネント応答部によってコンポーネント広告メッセージを定期的に IP マルチキャストを利用して送信する．これによってクライアントにコンポーネントの存在を通知する．コンポーネント広告メッセージは，コンポーネント応答メッセージと同じ内容を含む．

5.3 メディエータ部によるコンポーネント呼び出し

実装したシステムは，EJB，Web Service，CORBA の 3 種類のコンポーネントを探索し，それらを呼び出すことによって連携させて動作させる．コンポーネントの利用者は，まずコンポーネントを呼び出すためのクライアントプログラムを用意する．コンポーネントの呼び出しの際は，メディエータ部がコンポーネントへの呼び出しを受け取り，コンポーネント要求部を利用して必要なコンポーネントを探索する．そして発見したコンポーネントの呼び出しの手順に従い，リクエストを変換し，コンポーネントを呼び出し，処理結果をクライアントに返す．これにより，コンポーネント技術や実装による異種性を吸収する．

メディエータ部には，Web Service での呼び出しの形式に類似した，Java 言語による共通の API を用意した．Web Service の呼び出しとの形式的な相違は，コンポーネントの場所を示すエンドポイントの URL を指定するための API の代わりにコンポーネントのタイプを示す文字列を指定するための API を用意した点である．クライアントプログラムにはこの共通 API を利用し，通常のコンポーネント呼び出しと同様に，メディエータ部を呼び出すためのコードを記述する．このようにすることで，クライアントプログラムからは意識せずに，メディエータ部によってコンポーネントを探索して呼び出すことを可能にしている．

メディエータ部によるコンポーネント呼び出しの手順を，**図 8** に示す．なおこの図では，Web Service に対しての呼び出しの手順を示す．

まずメディエータ部は，コンポーネント要求部を利用し，必要なコンポーネントを探索する（図 8-(1)，図 8-(2)）．この際，すでにコンポーネントの存在を知っている場合には，探索を行わない．なお同一の内容を提供する複数のコンポーネントが探索された場合にどのような基準で選択するかを，メディエータ部に

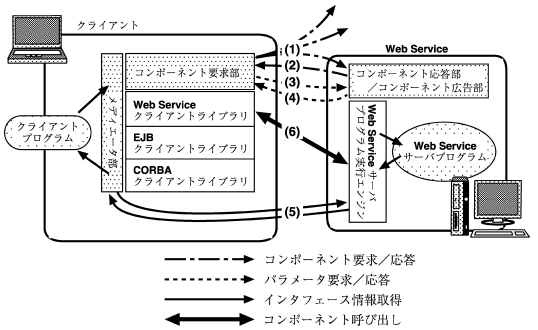

図 8 メディエータ部からのコンポーネント呼び出し
Fig. 8 Component invocation from the mediator.

あらかじめ設定しておくこともできる．この基準としては，コンポーネントの提供者によって設定された生存時間や，コンポーネントプラットフォームの種類，コンポーネントの処理に要する時間により選択できる．またメディエータ部にモジュールを追加することで，基準を追加することもできる．

次にコンポーネント要求部は，探索したコンポーネントに対してパラメータ情報を要求する（図 8-(3)）．そしてパラメータ応答メッセージによって，コンポーネントが実装されているアーキテクチャに依存した情報を得る（図 8-(4)）．メディエータ部はこの情報に従って，コンポーネントを呼び出す．それぞれのアーキテクチャによって，具体的には以下のようになる．

- Web Service
  パラメータ応答メッセージによって，WSDL で記述されたインタフェース情報を入手するための URL を受け取る．そしてその URL からインタフェース情報を入手する（図 8-(5)）．メディエータ部は，クライアントプログラムから受け取ったリクエストを該当するメソッドのシグネチャに変換し，探索した Web Service を呼び出す（図 8-(6)）．
- CORBA
  パラメータ応答メッセージによって，CORBA-IDL [10] で記述されたインタフェース情報を入手するための URL を受け取る．そしてその URL からインタフェース情報を入手する．メディエータ部は入手したインタフェース情報に従い，CORBA のコンポーネントを呼び出すための，スタブプログラムを生成する．メディエータ部は，クライアントプログラムから受け取ったリクエストを，スタブの該当するメソッドのシグネチャに変換し，スタブを介して探索した CORBA コンポーネントを呼び出す．
- EJB
  パラメータ応答メッセージによって，該当するメソッドを利用するためのインタフェースに関する情報を受け取る．この情報は，他の 2 つのアーキテクチャにおいて WSDL や CORBA-IDL によって記述されるコンポーネントのメソッドのシグネチャに関する情報に相当する．メディエータ部は，クライアントプログラムから受け取ったリクエストを該当するメソッドのシグネチャに変換し，探索した EJB コンポーネントを呼び出す．

これらの呼び出しが行われコンポーネントのメソッドの処理結果を受け取ると，返戻値としてクライアントプログラムの受け取れる形式に変換して返す．

クライアントからのリクエストの変換，およびコンポーネントからの返戻値の変換のためには，マッピングテーブルを用意しておく．たとえば，CORBA-IDL では，文字列は string や wstring という型で表現するが，これを Java の String クラスに変換する必要がある．提案システムの実装では，それらの対応をあらかじめマッピングテーブルとして用意している．

上記において，パラメータ応答メッセージを受け取りその解析を行った後，コンポーネントを呼び出す部分までの手順は，それぞれのコンポーネントアーキテクチャによって異なる．そしてそれぞれのコンポーネントを呼び出すためのライブラリが相互に影響を与えないように，各コンポーネントへの呼び出しを個別のスレッドで行っている．新たなコンポーネントアーキテクチャを利用するコンポーネントを統合する場合は，コンポーネントアーキテクチャに依存した，パラメータ応答メッセージの解析と，コンポーネントの呼び出しを行うモジュールをメディエータ部に追加するだけで対応できる．

なお現在の実装では，単純な同期呼び出しの変換のみを対象としている．しかし，たとえば CORBA の保留同期や，Web Service での片方向呼び出しのような，一般の同期呼び出しと異なる呼び出しを行いたい場合にも，それらのセマンティックスを持つモジュールをメディエータ部に追加することで，対応できる．

5.4 コンポーネントを統合するサーバの実装

コンポーネントを統合するサーバでは，複数のコンポーネントを統合するために，BPEL4WS に対応した既存のサーバを拡張した．

まず，コンポーネントを探索して呼び出すことができるように，BPEL4WS の言語を拡張した．具体的には 5.3 節で述べた Java API の場合と同様に，コンポーネントのタイプを示す文字列を指定するための

タグを用意し，エンドポイントの URL を指定するタグ要素の代わりとして利用できるようにした．そして BPEL4WS の実行時に統合アプリケーション運用部がこの拡張部分を解釈し，メディエータ部を呼び出す．これによって BPEL4WS を利用してコンポーネントを探索して呼び出すアプリケーションを実現する．また，BPEL4WS の実装では Web Service しか統合できないが，メディエータ部を呼び出すことによって，その他の種類のコンポーネントも呼び出せるようにできる．複数の異種コンポーネントを統合する場合は，異種性をメディエータ部が吸収する．

コンポーネントを統合するアプリケーションの提供者は，アプリケーションを BPEL4WS によって記述する．その際に拡張した記述を利用して，コンポーネントのタイプを指定して呼び出すためのタグ要素を記述する．そして，コンポーネントを統合するサーバに配置することにより，通常の BPEL アプリケーションと同様に，Web Service として呼び出すことができるようになる．外部から Web Service のインタフェースで呼び出すことができるので，作成したアプリケーションを組み合わせて，BPEL4WS に対応した他のサーバによって，さらに新しいアプリケーションを作ることも可能である．また同様に，実装したシステムを利用することによって，作成したアプリケーションを新しい別のアプリケーションが自動的に発見して，組み込むこともできる．

6. 評価実験

実装したシステムの性能を評価するため実験を行った．実験は，100 BASE-TX の同一ネットワークに接続された端末で行った．各端末には，CPU に Pentium4 3.0 GHz を搭載した同一の構成の PC を用いた．

6.1 オーバヘッドの測定

4.2 節で述べたように，本研究で実装したシステムでは，コンポーネントを利用するクライアントや，コンポーネントを統合するアプリケーションは，分散コンポーネントを探索してから呼び出す．またその際に，異種性を吸収するためにリクエストや返戻値を変換する．これらの処理によって，コンポーネントの呼び出しの処理を行うために必要な時間が増加する．この影響を調べるため，呼び出しの際に発生するオーバヘッドを測定した．

6.1.1 単一コンポーネントの呼び出し

Web Service，EJB，CORBA のそれぞれについて，同一の機能を提供するサーバプログラムと，それを利用するように記述したクライアントプログラムを作成した．また実装したシステムで動作するクライアントプログラムを作成し，各サーバにコンポーネント応答部を配置した．3 種類のコンポーネントそれぞれを利用する 3 つのクライアントプログラムと，実装したシステムで動作するクライアントプログラムの，計 4 種類のクライアントプログラムで，それぞれのコンポーネントを 30 回ずつ呼び出し，それに要する時間を測定した．測定結果の平均を**表 1** に示す．

表 1 単一コンポーネントの呼び出しに要する時間（単位：ms）
Table 1 Invocation time for single component (unit: ms).

処理	WS	CORBA	EJB
各コンポーネント技術を利用するクライアントプログラム			
コンポーネント呼び出し	382.2	119.4	574.1
実装したシステムで動作するクライアントプログラム			
コンポーネント探索	327.2	326.5	326.3
パラメータ探索	0.9	0.8	0.9
インタフェース情報取得	219.8	20.0	-
スタブ生成	-	1,062.2	-
コンポーネント呼び出し	382.9	125.4	697.1
パラメータ変換	0.0	0.0	0.0
合計	930.8	1,535.0	1,024.3

6.1.2 複数コンポーネントを統合するアプリケーションの呼び出し

まず，コンポーネントを組み合わせるプログラムを，BPEL4WS によって記述した．そしてそのプログラムを実行する際に必要とする Web Service のコンポーネント W_1，W_2 を用意しそれぞれ別のサーバに配置した．BPEL4WS で記述したプログラムを，BPEL4WS に対応したサーバ S_1 と，実装したシステムを組み込んだサーバ S_2 で，それぞれ実行した．なお，サーバ S_1 で動作させるプログラムコードには，サーバ W_1，W_2 を利用するように記述した．またサーバ W_1，W_2 には，コンポーネント応答部を配置した．なお比較対象のサーバ S_1 は Web Service のコンポーネントのみを統合するため，ここでは Web Service のコンポーネントのみを用意して行った．

クライアントから呼び出したときに，それぞれのサーバが各コンポーネントを呼び出すために要する時間を測定した．30 回測定した結果の平均を**表 2** に示す．

6.2 利用するコンポーネントの切替えに要する時間の測定

実装したシステムでは，利用していたコンポーネントが利用不可能になった場合は，別のコンポーネントを探索して利用する．このような状況が発生した際に，クライアントからの呼び出しにどれだけの影響を与えるかを調べるために，利用するコンポーネントの切替えに要する時間を測定する．

表 2 コンポーネント統合の際の呼び出しに要する時間（単位：ms）
Table 2 Invocation time for integrated components (unit: ms).

	処理		1 回目	2 回目以降
S_1	BPEL インタプリタ実行		1,062.5	426.1
	W_1 呼び出し		290.2	48.5
	W_2 呼び出し		38.2	47.7
	合計		1,391.0	522.2
S_2	BPEL インタプリタ実行		1,068.9	456.7
	W_1 呼び出し			
		コンポーネント探索	331.0	-
		パラメータ探索	1.2	-
		インタフェース情報取得	212.1	-
		コンポーネント呼び出し	190.3	43.1
		パラメータ変換	0.0	0.0
	W_2 呼び出し			
		コンポーネント探索	330.1	-
		パラメータ探索	0.9	-
		インタフェース情報取得	156.7	-
		コンポーネント呼び出し	191.2	45.2
		パラメータ変換	0.0	0.0
	合計		2,482.4	545.0

表 3 呼び出すコンポーネントの切替えに要する時間（単位：ms）
Table 3 Process time for component switching (unit: ms).

処理		時間
W_1 呼び出し		
	W_1 を利用できないことの検知	75,024.0
E_1 呼び出し		
	コンポーネント探索	328.2
	パラメータ探索	0.6
	コンポーネント呼び出し	522.4
	パラメータ変換	0.0
合計		75,875.2

まず実装したシステムを組み込んだサーバが，2 つのコンポーネント W_1 と W_2 を統合するアプリケーションを実行している．そして W_1 を提供しているサーバをネットワークから切断し，利用不能にする．この場合，W_1 を呼び出す時点で，その呼び出しに失敗する．その結果，利用できないことを検知し，別のコンポーネントを探索する．そして同じ機能を提供する EJB のコンポーネント E_1 を発見して利用する．

このときに，W_1 を利用できないことを検知し，E_1 を呼び出して応答を得るまでの時間を測定した．30 回測定した結果の平均を**表 3** に示す．

6.3 スケーラビリティに関する測定

実装したシステムでは，IP マルチキャストを利用してコンポーネントの探索を行う．つまりコンポーネントを探索するクライアントによって，多数のサーバへ同時にリクエストが送信される．またクライアントは複数のサーバから応答を得る可能性がある．そこで，メッセージ送受信の処理の影響を調べるために，コンポーネント探索とパラメータ探索に要するメッセージのサイズと，その処理に要する時間を測定した．

表 4 送受信されるメッセージのサイズと処理に要する時間
Table 4 Size and process time of passing messages.

	コンポーネント探索		パラメータ探索	
	要求	応答	要求	応答
文字列の長さ：12 byte				
サイズ（byte）	21	23	19	97
処理時間（ms）	0.349	0.097	0.066	0.268
文字列の長さ：24 byte				
サイズ（byte）	33	23	19	97
処理時間（ms）	0.343	0.096	0.064	0.262
文字列の長さ：48 byte				
サイズ（byte）	57	23	19	97
処理時間（ms）	0.337	0.098	0.063	0.262

6.1.1 項でオーバヘッドを測定した際に利用した EJB のコンポーネントを，実装したシステムで動作するクライアントプログラムから呼び出した．ここでは同一のコンポーネントに対して，要求するタイプを指定するための文字列を 12 byte，24 byte，48 byte の 3 種類で設定し，それぞれについて探索を行った．

まず各メッセージの送受信を行う際のパケットを取得し，IP ヘッダ部分を除いたペイロード部分のサイズを測定した．そしてこれらのメッセージの送受信を行う際に，クライアントの処理に要する時間を 30 回測定した．これらの結果を，**表 4** に示す．

6.4 異種分散コンポーネントを統合するアプリケーションの呼び出しに要する時間の測定

実装したシステムでは，異種分散コンポーネントを自動的に発見し，連携させる．それぞれのコンポーネントを統合するための手順に要する処理時間を調べるために，実装を行ったシステムで対象とした EJB と Web Service と CORBA の 3 種類のコンポーネントを連携させるアプリケーションを作成し，それらを実行した際の処理時間を測定した．

まず，3 つのコンポーネントを逐次実行し，実行したコンポーネントの出力を次に実行するコンポーネントに入力することによって連携させる BPEL アプリケーションを作成した．このアプリケーションには本実装により拡張を行った記述を含んでおり，拡張を行った BPEL サーバ上で動作させることで，必要なコンポーネントを探索してから呼び出す．また，各サーバのコンポーネント応答部を設定し，6.1.1 項でオーバヘッドを測定した際に利用した各コンポーネントを，作成したアプリケーションが利用し統合できるようにした．ここでは探索の結果，まず最初に CORBA コンポーネントが実行され，次に Web Service が，最後に EJB のコンポーネントが実行されるように，該当するサーバのコンポーネント応答部を設定した．

表 5 異種分散コンポーネント統合の際の呼び出しに要する時間（単位：ms）
Table 5 Invocation time for integrated heterogeneous distributed components (unit: ms).

処理			1 回目	2 回目以降
コンポーネント探索			326.9	327.0
パラメータ探索			0.6	0.6
インタフェース情報取得			497.9	222.3
BPEL アプリケーション呼び出し			3,772.1	529.2
（内訳）				
	BPEL インタプリタ実行		745.9	418.8
	CORBA コンポーネント呼び出し			
		コンポーネント探索	326.3	-
		パラメータ探索	0.8	-
		インタフェース情報取得	21.5	-
		スタブ生成	909.7	-
		コンポーネント呼び出し	241.9	29.0
		パラメータ変換	0.0	0.0
	Web Service 呼び出し			
		コンポーネント探索	328.2	-
		パラメータ探索	0.8	-
		インタフェース情報取得	155.5	-
		コンポーネント呼び出し	196.8	46.9
		パラメータ変換	0.0	0.0
	EJB コンポーネント呼び出し			
		コンポーネント探索	326.8	-
		パラメータ探索	0.8	-
		コンポーネント呼び出し	517.1	34.5
		パラメータ変換	0.0	0.0
合計			4,597.5	1,079.1

そしてこの BPEL アプリケーションを本提案により拡張を行った BPEL サーバに配置し，クライアントから呼び出した．なおクライアントからの呼び出しの際にも，実装したシステムを利用し，必要なコンポーネントを探索してから実行する．30 回呼び出しを行った結果の平均を，**表 5** に示す．

7. 考察

本章では，評価実験の結果を分析した後，提案システム利用の有効性について考察する．

7.1 オーバヘッドについて

表 1 より，Web Service，CORBA，EJB のいずれの場合も，コンポーネント探索により 330 ms 程度のオーバヘッドが発生していることが分かる．コンポーネント探索は，IP マルチキャストにより行っており，該当する複数のサーバから応答を得る可能性がある．そのため，コンポーネント応答メッセージを受信しても，他のサーバからの応答を待ち続ける．本実装では，コンポーネント要求メッセージを送信してから応答を待ち続ける時間を，300 ms に設定した．これによりコンポーネント探索にオーバヘッドが生じた．

コンポーネント探索を除いたオーバヘッドについては，利用するコンポーネントのそれぞれのアーキテクチャによって異なる．これは 5.3 節で述べたように，メディエータ部がそれぞれのアーキテクチャによって，異なった方法でインタフェース情報を取得して，呼び出しを行うためである．Web Service のサーバ実装では，クライアントが HTTP を経由してインタフェース情報を記述した WSDL の文書を取得する．サーバはその時点で WSDL の文書を動的に生成し，クライアントに返す．また CORBA コンポーネントでは，インタフェース情報をあらかじめ CORBA-IDL で記述して公開しておき，クライアントはこれを HTTP で取得している．そして取得したインタフェース情報に従ってスタブを生成する．どちらの場合も，これらの処理のために発生するオーバヘッドが大きくなっている．なお EJB の場合は，これらの処理を行わないため，オーバヘッドが発生しない．またどの場合でも，パラメータ探索や，メソッドのシグネチャやパラメータの変換のためには，ほとんどオーバヘッドを発生しない．

また表 2 や表 5 に示すように，複数のコンポーネントを統合するアプリケーションを実行する場合も，それぞれのコンポーネントについて同様にオーバヘッドを発生している．

このように通常のそれぞれのコンポーネント呼び出しに対して，合計で約 330 ms（EJB の場合）〜約 1,400 ms（CORBA の場合）の長さのオーバヘッドが生じている．その結果，コンポーネントの呼び出しに要する時間が，約 2 倍〜約 13 倍に増加している．

発生したオーバヘッドのうち，コンポーネント探索に要する時間は，接続するネットワークのターンアラウンドタイムを参考に調節できる．また複数のサーバからの応答を待ち続けずに，最初に応答が到着したものを利用するようにすれば，300 ms の待ち時間が短縮できる．EJB のようにパラメータ応答メッセージに含めてサーバから送信すれば，インタフェース情報の取得によって発生するオーバヘッドをなくすことができる．CORBA では通常はスタブのコードを生成して呼び出すが，メディエータ部が直接 CORBA のコンポーネントを呼び出せば，スタブの生成によって発生するオーバヘッドもなくすことができる．このように，実装を改良することによって，オーバヘッドをかなり減らすことができる．

さらに表 2 や表 5 に示したように，コンポーネントを統合するサーバでは，探索を行った結果をキャッシュするため，2 回目以降の呼び出しではオーバヘッドを生じない．利用状況にもよるが，これによりある程度のオーバヘッドを軽減できる．

また 2.1 節で述べたように，分散コンポーネントを利用するためには，通常はまずレジストリから必要なコンポーネントを発見し，レジストリや発見したコンポーネントを提供するサーバからインタフェースに関する情報を入手する．そして入手したインタフェースに関する情報に従って，コンポーネントの利用者がクライアントプログラムを作成する．一方提案システムでは，コンポーネントを発見する前にクライアントプログラムが作成されており，コンポーネントを発見した後には，発見したコンポーネントを呼び出せるように自動的に調整している．すなわち通常の分散コンポーネントの利用方法では，仮にコンポーネントが発見できたとしても，利用者は発見したコンポーネントを呼び出すようにプログラムコードを開発する必要があり，発見したコンポーネントを自動的に利用することはできない．提案システムでは長いオーバヘッドが生じたが，通常の分散コンポーネントの利用方法で同様のことを利用者が行うとすると，それよりもさらに長い時間を要する．そのうえ，コンポーネントの利用者にアプリケーションの開発知識を求めることになる．

7.2 コンポーネントが利用不可能な場合の切替え処理について

コンポーネントを統合するアプリケーションを実行している際に，呼び出そうとするコンポーネントの実行に失敗すると，他のコンポーネントを探索して処理を切り替える．

呼び出されるサーバをネットワークから切断した場合は，BPEL4WS の実行エンジンが HTTP リクエストによるコネクションがタイムアウトするまで失敗したことを検知できない．そのため表 3 のように，非常に長い時間を要した．しかし検知すると，すぐに他のコンポーネント E_1 を発見して処理を切り替えることができた．

提案システムでは，クライアントはコンポーネント探索を行ってからサーバを呼び出す．サーバがネットワークから切断されている場合にはコンポーネント探索に対する応答が得られないので，HTTP によるコンポーネント呼び出しを行わない．またサーバがネットワークに接続した状態で，何らかの要因によってコンポーネントが利用不可能になっている場合には，HTTP によるコネクションが切断されるか，HTTP によるコネクションが成功して利用不可能である応答を得るかのいずれかになる．このようにどちらの場合でも，HTTP コネクションのタイムアウトによってコンポーネント実行の失敗を検知するような状況は発生しない．そのような状況は，探索を行っている結果をクライアントによってキャッシュしているときに，該当するサーバが切断された場合にのみ発生する．よって，探索結果をキャッシュとして保持している場合に，該当するサーバのネットワークへの接続性をクライアントから定期的に確認することで，この状況をある程度は回避できる．

7.3 メッセージ処理のスケーラビリティについて

コンポーネント探索は，IP マルチキャストを利用して行われる．したがって，多数のサーバとコンポーネント要求やコンポーネント応答のメッセージのやりとりをするため，どれだけの大きさのメッセージがネットワーク上に送出されるかが問題になる．

表 4 によると，コンポーネント要求メッセージとコンポーネント応答メッセージは，非常に小さいことが分かる．ネットワーク上に多数のサーバが存在し，多数の要求や応答がやりとりされるとしても，非常に小さなメッセージであるので，ネットワークの負荷を高めるとは考えにくい．なおコンポーネント要求メッセージは必要な機能を指定する文字列を含むため，文字列の長さによってメッセージサイズが線形に変化する．

また，コンポーネント応答メッセージが多数のサーバから送信された場合は，クライアントがコンポーネント応答メッセージを処理するのにどれだけの時間を要するかが問題になる．しかしこれについても，非常に短い時間で終わっているため，クライアントの負荷にそれほどの影響を与えないといえる．なお，実装したシステムのコンポーネント探索機構では，クライアントが多数のサーバからの応答を得ることを考慮して，なるべくメッセージサイズを小さくするために，パラメータ応答メッセージに含まれるようなコンポーネントの呼び出しに関する情報を含まない．そしてそれらの情報を，ユニキャストで行われるパラメータ探索によって得るようにしている．また，クライアントの処理を単純にするために，コンポーネント応答メッセージについては，サーバ内での固定長の番号によってコンポーネントを識別し，メッセージの長さは変化しないようにしている．

7.4 提案システムの有効性について

本節では，提案システムの有効性を考察するために，提案システムの適用先と，コンポーネント探索のために IP マルチキャストを利用したことに対する有効性と，異種性を吸収する方法についての有効性について述べる．

提案システムの有効な適用先

提案システムではコンポーネントを発見する前に，それを利用するアプリケーションを開発することにな

る．このため，どのような機能のコンポーネントが存在していてどれを利用するかという見当があらかじめある程度ついている場合には有効に利用できる．逆に，どのような機能のコンポーネントが存在するかがまったく想定できず，発見したコンポーネントの機能によってプログラムコードの構造が決定するような場合には利用できない．したがってソフトウェア部品市場が発達し，機能ごとに交換可能なソフトウェア部品群が流通するようになると，提案システムを有効に利用できる．たとえば，ホームアプライアンスのような組み込み製品に展開し，ハードウェアベンダが提供する製品の機能ごとにコンポーネントを組み込むことを考えてみる．この場合，利用者は必要な機能を提供する製品を購入しネットワークに接続することで，自動的に組み合わせることができるようになる．このような場合にネットワークに設置した装置に対して機能追加が簡単に行えるようになり，提案システムを有効に利用できる．

IP マルチキャスト利用の有効性

提案システムの実装では，コンポーネント探索に IP マルチキャストを利用しており，これによって同一ネットワーク上に存在する未知のコンポーネントサーバを発見できるようにしている．提案システムでは，2.2 節に示したような比較的狭い環境での Plug and Play を想定しており，グローバルネットワークへのコンポーネント探索を対象にしていない．そのため，ローカルネットワークで IP マルチキャストを利用することによって，コンポーネント探索を実現している．

ここで提案システムの対象外ではあるが，より広域に探索を行いたい場合の拡張を考えてみる．

広域網を対象に探索を行うためには，ディレクトリサービスのような階層的に情報を管理できる方式が必要になる．その場合には，ディレクトリサービスからコンポーネントを発見したり，ディレクトリサービスにコンポーネントの情報を登録したりすることになる．この際に，ディレクトリサービスを自動発見するために，提案システムを利用できる．また，それほど広域ではない複数ネットワーク間での Plug and Play については，提案システムに加えて，各ネットワークに単純にリクエストを転送するための装置を設置することによっても実現できる．いずれの場合も提案システムを発展させることにより実現できる．

異種性を吸収する方法の有効性

従来の研究[16]では，異種コンポーネントを呼び出す際に各サーバに Web Service から特定の方式の分散コンポーネントにマッピングするための Proxy と呼ぶ機構を用意し，外部からはすべて Web Service として呼び出せるようにしている．この場合は，それぞれの異種コンポーネントを，単純な Web Service のセマンティックスで利用できる．

このように，異種性を吸収するだけならば，コンポーネントを提供するサーバに Proxy のような機構を用意することで実現できる．しかし提案システムのように探索を行うためには，クライアントも変更する必要がある．また，Proxy の方法では，呼び出しが Web Service の提供する機能に制限される．たとえば，SOAP のプロトコルにない機能を持った新しいプロトコルを利用するサーバを呼び出す場合，SOAP でしかクライアントと Proxy の間でのやりとりができないことによって，その機能が失われる．またさらにそれに加えて，サーバに Proxy のような変換機構を用意する場合は，サーバは Web Service のサーバとしても動作する必要があり，このために多くのリソースを要求することになる．

これらの理由により，提案システムでは呼び出し側で異種性を吸収する方法を用いた．そして呼び出し元に配置したメディエータ部が，呼び出し先の分散コンポーネントの呼び出しの方法に変換している．これにより呼び出しが Web Service に制限されないため，新しいプロトコルのサーバを呼び出す場合でも，メディエータを変更するだけで対応できる．そして，サーバにはコンポーネント応答部のみを用意すればよく，これは表 4 に測定したように，非常にサイズの小さいメッセージのやりとりしか行わない．そのため，サーバに対するリソースの要求を抑えることができる．

8. ま と め

本研究では，分散コンポーネントの存在するサーバやそれを利用するクライアントをネットワークに接続することによって，Plug and Play で動作させる環境を提案した．この環境を利用すると，コンポーネントとそれを提供するサーバを利用時に自動的に探索して発見し，コンポーネントの種類によらずに統合できる．そしてこの環境を実現するシステムを実装し，実験を行い，オーバヘッドが発生する要因を分析した．

提案システムを利用することにより，ネットワークに端末を接続するだけで煩雑な設定操作を行わずにコンポーネントを利用できるようになる．このため，ユーザインタフェースが十分ではなく設定操作が容易ではない組み込み機器のような，今まで分散コンポーネントを利用しなかった分野にも，分散コンポーネントを基盤技術として適用し，新しいシステムを構築で

きるようになる．

今後はまず，コンポーネントの探索の際のメッセージの送受信による負荷を軽減するための方法を検討する予定である．また，コンポーネントの探索に対してセキュリティを考慮する必要もある．

謝辞 本研究は，文部科学省科学技術振興調整費「環境情報獲得のための高信頼性ソフトウェアに関する研究」の支援による．

参 考 文 献

1) Microsoft Corp.: Speech Transcript: Seamless Computing: Hardware Advances for a New Generation of Software; WinHec 2004 (2004). http://www.microsoft.com/billgates/speeches/2004/05-04winhec.asp
2) Sun microsystems: Enterprise JavaBeans, Specification Version 2.1 (2003). http://java.sun.com/products/ejb/
3) Object Management Group: Common Object Request Broker Architecture: Core Specification (3.0.3), OMG Document (2004). http://www.omg.org/technology/documents/formal/corba_2.htm
4) World Wide Web Consortium: Web Services Architecture, W3C Working Group Note 11 February 2004 (2004). http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
5) BEA Systems: BEA eLink Adapter for Mainframe (2001). http://e-docs.bea.com/elink/eam/v41/
6) Sybase: e-Biz Integrator 4.1 Product Documentation (2005). http://sybooks.sybase.com/nav/detail.do?docset=690
7) webMethods: webMethods Glue 6.0 (2005). http://www.webmethods.com/meta/default/folder/0000008413
8) Object Management Group: Naming Service Specification (1.3), OMG Document (2004). http://www.omg.org/docs/formal/04-10-03.pdf
9) OASIS UDDI Specification Technical Committee: UDDI Version 3.0.2, UDDI Spec Technical Committee (2004). http://uddi.org/pubs/uddiv3.0.220041019.htm
10) Object Management Group: OMG IDL Syntax and Semantics (chapter 3 of CORBA v3.0), OMG Document (2002). http://www.omg.org/docs/formal/02-06-39.pdf
11) World Wide Web Consortium: Web Service Definition Language (WSDL) 1.1, W3C Note (2001). http://www.w3.org/TR/2001/NOTE-wsdl-20010315
12) Siljee, J., Bosloper, I., Nijhuis, J. and Hammer, D.: DySOA: Making Service Systems Self-adaptive, *3rd International Conference on Service Oriented Computing* (*ICSOC 2005*), Lecture Notes in Computer Science, Vol.3826, pp.255–268 (2005).
13) Goland, Y.Y., Cai, T., et al.: Simple Service Discovery Protocol/1.0, IETF Internet-draft (expired), The Internet Engineering Task Force (1999). http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt
14) Guttman, E., Perkins, C., et al.: Service Location Protocol, Version 2, RFC 2608, The Internet Engineering Task Force (1999). http://www.ietf.org/rfc/rfc2608.txt
15) Beatty, J., Kakivaya, G., et al.: Web Services Dynamic Discovery (WS-Discovery), Microsoft Corporation (2004). http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-discovery1004.pdf
16) 名倉正剛，河野泰隆，高田眞吾，土居範久：異種分散コンポーネントを利用するアプリケーションの開発を支援するシステム，情報処理学会論文誌，Vol.47, No.2, pp.484–494 (2006).
17) Cerqueira, R., Cassino, C. and Ierusalimschy, R.: Dynamic Component Gluing Across Different Componentware System, *International Symposium on Distributed Objects and Applications* (*DOA '99*), pp.362–371 (1999).
18) BEA Systems, IBM Corporation, et al.: Business Process Execution Language for Web Services Version 1.1, WSBPEL TC, OASIS (2003). ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

(平成 18 年 5 月 11 日受付)
(平成 18 年 11 月 2 日採録)

名倉 正剛（正会員）

1999 年慶應義塾大学理工学部卒業．2001 年同大学大学院理工学研究科修士課程修了．同年日本電気株式会社入社．ネットワークス開発研究所にて，インターネットプロトコルに関する研究開発に従事．2003 年日本電気株式会社退社．同年慶應義塾大学大学院理工学研究科博士課程入学．2006 年同博士課程単位取得退学．現在，同大学大学院理工学研究科特別研究助手．分散コンポーネントシステムに関する研究に従事．

高田 眞吾（正会員）

1990 年慶應義塾大学理工学部卒業．1992 年同大学大学院理工学研究科修士課程修了．1995 年同博士課程修了．博士（工学）．同年奈良先端科学技術大学院大学情報科学研究科助手．1999 年より慶應義塾大学理工学部情報工学科専任講師，現在，同大学助教授．ソフトウェア工学，情報検索等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE CS 各会員．

土居 範久（名誉会員）

1969 年慶應義塾大学大学院博士課程単位取得退学．慶應義塾大学理工学部教授を経て，2003 年より中央大学理工学部教授，慶應義塾大学名誉教授．工学博士．現在，日本学術会議会員・副会長，文部科学省科学技術・学術審議会委員，総務省情報通信審議会委員，経済産業省産業構造審議会専門委員，総合科学技術会議評価専門調査会委員，科学技術振興機構（JST）社会技術研究開発センター「情報と社会」領域総括，特定非営利活動法人日本セキュリティ監査協会会長，国際計算機学会（ACM）日本支部長，等．専門はソフトウェアを中心とした計算機科学．情報処理学会名誉会員．