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

名倉正剛[†] 河野泰隆[†]
高田眞吾[††] 土居範久[†††]

近年，EAI（Enterprise Application Integration）と呼ばれるアプリケーション開発アプローチが注目されている．これにより，既存のアプリケーションプログラムを構成するコンポーネントを組み合わせることによって，新しいアプリケーションを開発することができる．コンポーネントの組合せは，コンポーネントを実行した結果を別のコンポーネントの入力に利用したり，複数のコンポーネントを実行した結果をまとめたりするプログラムコードを生成することで行われる．そして利用者は生成されたプログラムコードを介して，それぞれの分散コンポーネントを利用する．しかし，いろいろなベンダによって開発された様々なコンポーネントを組み合わせてアプリケーションを開発する場合には，それぞれのコンポーネントが独自に開発されていることに起因して，組み合わせるコンポーネント間の異種性の問題が発生する．また，組み合わされるコンポーネントが分散していることに起因して，開発したアプリケーションプログラムを運用する際の信頼性の問題が発生する．本研究では，これらの問題を解決し，ワークフロー情報を用いて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を設計し，実装する．

Application Development Using Heterogeneous Distributed Components

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

EAI (Enterprise Application Integration) is widely used for developing applications. In this approach, existing application programs are integrated by combining their constituent components. Components are combined by generating program code which takes the output of one component and uses it as input to another component, and program code that combines the result of several components. Current EAI support focuses more on components that are developed from the same base technology from the same vendor. This however limits the components that can be used. Furthermore, components are normally dispersed among the Internet and managed by different personnel. This raises questions as to the reliability of those components. We thus propose a software development tool that supports combining heterogeneous distributed components resulting in an application of high reliability.

1. はじめに

インターネットの発達にともない，ネットワークを介して通信を行うソフトウェアに対してコンポーネント技術[1)~3)]を適用した，分散コンポーネント技術が普及してきている．EJB（Enterprise JavaBeans）[4)]，CORBA（Common Object Request Broker Architecture）[5)]，Web Service[6)]などの分散コンポーネント技術を利用して構築されたアプリケーションプログラムは，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．コンポーネントが実行する処理はサーバプログラムとして外部から呼び出すことができるように実装され，サーバコンピュータ上に配置される．クライアントプログラムはクライアントコンピュータに配置され，ネットワークを介してサーバプログラムを呼び出しコンポーネントの処理を実行する．

近年，EAI（Enterprise Application Integration）と呼ばれるアプリケーション開発アプローチが注目されている．このアプローチでは，既存のアプリケー

† 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 Technology, Chuo University

ションプログラムを統合することによって新しいアプリケーションを開発する．この際，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て統合することによって，アプリケーションプログラムを統合する．これは，コンポーネントの処理結果を別のコンポーネントに入力したり，複数のコンポーネントの処理結果をまとめたりすることで実現される．このようにコンポーネントを統合することで新たなサーバプログラムを作成し，それを呼び出すクライアントプログラムを作成する．

組合せに利用する分散コンポーネントは，それぞれが 1 つのビジネスロジックを実現していることが多く，比較的粒度が大きい．このため，それらを利用してアプリケーション開発を行う際には，組み合わせる各コンポーネント間に強い関連性がない．したがって，コンポーネントを組み合わせてアプリケーション開発を行う際には，コンポーネントをどのような順番で組み合わせるのかという処理やデータの流れ（ワークフロー）に関する情報が重要になる．具体的には，以下の点を明確にする必要がある．

- どのコンポーネントとどのコンポーネントが対応づけられるのか，呼び出す順番はどうなるのか．
- 引数や返戻値はどのように対応するのか．

アプリケーション開発は，開発者がこれらを指定するためのプログラムコードを記述することによって行われる．利用者は外部からそのプログラムコードを直接呼び出したり，ユーザインタフェースを介して呼び出したりすることで，利用する（**図 1**）．また，分散コンポーネント技術の種類によっては，統合を行う際に必要な情報を開発者があらかじめ XML で記述しておき，実行時にその記述に従ってコンポーネントを呼び出す仕組みも提案されている[7)~9)]．

分散コンポーネントを利用したアプリケーションを開発するために，いろいろなベンダによって様々なコンポーネントが提供されている．それらはそれぞれのベンダで独自に開発されたものであり，必ずしも同一の分散コンポーネント技術を用いて開発されているという保証はない．そのため，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．そのような規格の異なるコンポーネント技術を利用していたり，異なるプラットフォームの上で動作していたりする複数の分散コンポーネントを本稿では「異種分散コンポーネント」と呼ぶ．

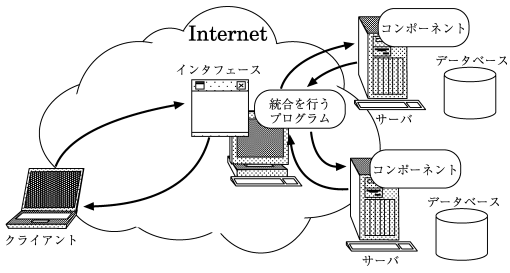

図 1 分散コンポーネントの統合
Fig. 1 Integration of distributed components.

異なったベンダの提供する様々な異種分散コンポーネントを組み合わせてアプリケーションを開発するためには，まだ多くの問題が残されている．本稿ではそのうち以下に着目する．

問題 1：組み合わせるコンポーネント間に異種性が存在することにより，組み合わせて運用することが容易ではない．

問題 2：開発したアプリケーションプログラムによって組み合わされる各コンポーネントが分散していることにより，信頼性が低下する．

本研究では，これらの問題を解決し，ワークフロー情報を用いて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を設計し，実装する．

まず 2 章で本研究で着目する問題点を分析し，3 章で関連研究を述べる．そして 4 章で本研究で提案するシステムについて述べ，5 章で評価実験を行い，6 章で考察を行った後，7 章でまとめる．

2. コンポーネント統合における問題点

本章では，異種分散コンポーネントを統合する際の問題点，および分散コンポーネントを統合することによって開発したアプリケーションプログラムを運用する際の問題点を分析する．

2.1 異種分散コンポーネントを統合する際の問題

異種分散コンポーネントは，呼び出し方法やメッセージや返戻値の形式が異なるため，相互運用性が低い．そのため，異種分散コンポーネントを組み合わせてアプリケーションを開発することは容易ではない．本節では，この問題を分析する．

2.1.1 異種分散コンポーネント間の結合の記述

異種分散コンポーネントを利用する場合，コンポーネント間の実装形式の異種性を吸収する必要がある．さらに，コンポーネントの実装形式の異種性を吸収できたとしても，呼び出されるコンポーネント間でパラメータや返戻値の形式や意味が異なる．そのため，異種分散コンポーネントを利用して新しいアプリケーションを開発する場合は，コンポーネント間の異種性を吸収し，パラメータや返戻値の対応関係を考慮して，

必要なプログラムコードを開発者が記述する必要がある．このことは，結合を行う部分を記述するための言語と，コンポーネントに対して呼び出しを行ったり他のコンポーネントから呼び出されたりするための形式を，開発者が熟知していなければならないことを意味する．

EAI の本質的な目的は異なる業務アプリケーションを相互に連携させることなので，アプリケーションプログラムの統合を容易に実現できる必要がある．したがって開発者は，どのようなコンポーネントをどのような順番で組み合わせてアプリケーションを開発するのかのみを考えることが望ましい．そしてどのような形式でコードを記述して，どのような手順で呼び出しを行うのかは考えなくて済ませられるべきである．

2.1.2 アプリケーションプログラムの設計方法

コンポーネントを組み合わせることによって新しいアプリケーションプログラムを開発しようとする開発者は，どのコンポーネントをどの順番で呼び出すかというワークフローに関する情報を記述する必要がある．

アプリケーションプログラムの統合を行うための EAI システムには様々な製品が存在する．既存の EAI システムは，EAI ツールと呼ばれるアプリケーション開発ツールを，それぞれ独自に提供している．開発者は，開発されるアプリケーションプログラムが処理する業務（アクティビティ）を組み合わせ，EAI ツールを利用してその結果をワークフローとして記述する．そして記述したワークフローに基づき，それぞれのアクティビティにコンポーネントを対応づけることによって，アプリケーションの開発を行う．

この際，ワークフローの記述に関してツール間で統一したアプローチをとっておらず，逐次文書として表現することもあれば，フローチャートなどの図を用いて表現することもある．しかし EAI ツールの操作に慣れていない開発者にとっては，独自の表現方法を学習するより，汎用的な表現方法を利用できる方が望ましい．また EAI ツールの操作に慣れている開発者も，異なる EAI ツールを利用してシステムを開発する際には新たに独自の表現方法を学習しなければならない．したがってこの場合も，汎用的な表現方法を利用できる方が望ましい．

また，EAI ツールによって記述したワークフロー情報などの成果物は，EAI ツール独自の形式で表現され保存されることが多い．したがって別の EAI システムで利用しようとすると，同一の形式ではないために利用できない場合がある．EAI システムによって生成したアプリケーションプログラムをいろいろな EAI システム間で再利用できるようにするために，コンポーネント間のワークフロー情報を保存する際の記述形式を統一することが望ましい．少なくとも，利用する EAI システムで解釈できなくとも開発者がメインテナンスできるように，可読な文書として表されるべきである．

2.2 開発したアプリケーションプログラムを運用する際の問題

第三者によって公開されている分散コンポーネントを組み合わせて新しいアプリケーションを開発する場合，組み合わせられる各コンポーネントを実行するコンピュータは一般的に異なっている．特に異なるベンダが提供していることを考えると，通常はそれらのコンピュータが接続されるネットワークも異なっていることが多い．したがって，コンポーネントを統合してアプリケーションを開発した場合は，アプリケーションプログラムを実行するコンピュータの管理者が対処できない以下の障害が発生する可能性がある．

- コンポーネントを実行するコンピュータの障害
- コンポーネントを実行するコンピュータと，アプリケーションプログラムを実行するコンピュータとの間のネットワークの障害

前者については，それぞれのコンポーネントを実行するコンピュータを多重化することで軽減できる．しかし通常は，コンポーネントを実行するコンピュータの管理者は，コンポーネントを組み合わせようとする開発者や，アプリケーションプログラムを実行するコンピュータの管理者とは異なっている場合が多い．したがって多重化を期待することは，現実的ではない．もし仮に多重化できたとしても，コンポーネントを提供するベンダが多重化されたコンピュータ群を設置することになる．同じベンダによって提供されるため，それらのコンピュータ群は，通常は多重化する対象のコンピュータからネットワーク的に近接した場所に設置されると考えられる．したがって，後者の解決にはならない．

コンポーネントを組み合わせたアプリケーションの開発では，一般的にどのコンポーネントを組み合わせるか，アプリケーションプログラムの設計時にすでに決定している．開発者が設計を行った時点では利用できたコンポーネントが，利用者がアプリケーションプログラムを実行しようとする時点では利用できないことがある．その際の障害を，開発者や，アプリケーションプログラムを実行するコンピュータの管理者，利用者のだれもが解決できない可能性があり，アプリケーションプログラムの運用の信頼性を低下させる．

3. 関連研究

異種コンポーネントの相互運用を行う既存のシステムとして，Java プログラムから C++ のコンポーネントを呼び出すことを可能にするためのインタフェースを生成するシステム[10] が提案されている．これは，C++ コンポーネントのインタフェースを解析し，Java から呼び出すためのインタフェースを自動生成する．

異種コンポーネント間の結合を行う部分の記述のためのインタプリタ言語として，結合言語 Lua[11] が提案されている．Lua システムは，CORBA，COM，Java のコンポーネントを結合する（**図 2**）．結合言語 Lua を用いることにより，Lua システムを利用して外部のコンポーネントへ呼び出しを行うプログラム（ブリッジ）を作成する．ブリッジを各コンポーネントが呼び出すことで，コンポーネントを結合する．

また，Java コンポーネントにおいて，インタフェースやコンポーネント間の結合の制約を記述するための言語として，ArchJava[12] が提案されている．これは，Java コンポーネントのコード中に，外部へアクセスしたり外部からアクセスされたりするメソッドのシグネチャを定義できるように拡張している．コンポーネントどうしの結合は，シグネチャが同じメソッドどうしを結合することによって表現できる．

上記 3 つの技術により，異種コンポーネントを相互に呼び出すことで新しいアプリケーションを開発したり，その際のコンポーネントのインタフェースや結合を定義したりできるようになる．しかしコンポーネントの実装形式の異種性が吸収できても，コンポーネント間の呼び出しの関係を記述したプログラムコードを記述する必要がある．同様に，コンポーネントの結合を定義することでコンポーネントの呼び出しの関係を記述できても，コンポーネント間の結合に合わせて異なるシグネチャの形式を調整するプログラムコードを記述する必要がある．それらはすべてコンポーネントを組み合わせようとする開発者が行わなければならず，それを支援するための仕組みが提供されていない．

アプリケーションプログラムの統合を行う既存の EAI システムとしては，BEA Systems の eLink[13]，Sybase の e-Biz Integrator[14]，webMethods の webMethods Glue[15] などがある．これらを利用することで，複数のコンポーネントを統合して動作させることができる．これらのシステムでは，開発者がワークフローに関する情報を記述することによって，コンポーネントを呼び出して統合するようにサーバを動作させる．しかしワークフローの表現は，それぞれのシステムで独自の形式で行っている．また，コンポーネントを提供するベンダによってアプリケーションプログラムを統合することを前提としているため，ネットワークを介して分散コンポーネントを呼び出すことによる信頼性の低下を考慮していない．

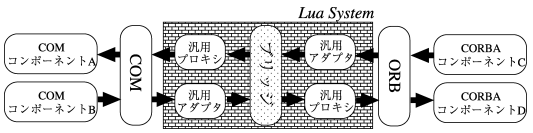

図 2 Lua を用いた異種分散コンポーネントの結合
Fig. 2 Binding of heterogeneous distributed components using Lua.

4. ワークフロー情報を用いて異種分散コンポーネントを統合するシステム

本章では，本研究で提案するシステムの設計と実装について述べる．なお本システムは，既存の研究[16] に拡張を行うことにより実装した．

4.1 概　　要

本研究で提案するシステムは，開発者がワークフロー情報を記述することで，異種分散コンポーネントを 1 つのプログラムとして連携動作させるためのプログラムコード（アダプタ）を生成する．このアダプタがそれぞれのコンポーネントを呼び出すことによって，全体として開発対象のアプリケーションプログラムとして動作する．アダプタを分散コンポーネントとして動作するように生成し，利用者には他の分散コンポーネントと同様に利用させる．組み合わせる分散コンポーネントとしては，EJB コンポーネントと，SOAP インタフェースを備えた Web Service コンポーネントと，CORBA コンポーネントの 3 種類を対象とする．

提案システムは，以下の 4 つの部分からなる．

- ワークフローエディタ
  開発者がワークフローの記述を行うためのインタフェース
- アダプタ生成部
  ワークフローエディタで記述されたワークフローに従って，アダプタを生成する部分
- アプリケーションプログラム運用部
  生成したアダプタを配備し，運用を行う部分
- アダプタリポジトリ
  生成したアダプタを登録しておくリポジトリ

3 章であげた EAI システムも，同様に開発者がワークフローを記述することで分散コンポーネントの統合を行うが，提案システムは以下の点で異なる．

- ワークフローに関する情報の表現方法

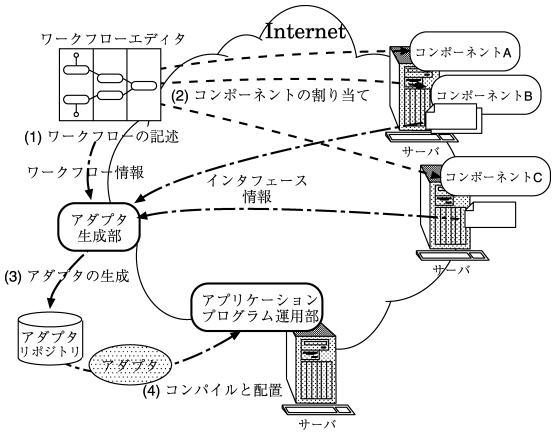

図 3 アダプタ生成
Fig. 3 Adapter generation.

- コンポーネントを結合する方法
- 運用を行う際に発生した障害の回避

本章では，まず異種分散コンポーネントを統合する際の問題（2.1 節）を解決するために，アダプタを生成する方法を示す．次に運用に関する問題（2.2 節）を解決するための方法を示す．

4.2 アダプタ生成

提案システムでは，以下の手順でアダプタを生成する（**図 3**）．

(1) ワークフローの記述
(2) コンポーネントの割当て
(3) アダプタの生成
(4) アダプタのコンパイルと配備

4.2.1 ワークフローの記述

開発者はワークフローエディタ上で，異種分散コンポーネントを呼び出すアプリケーションプログラムのワークフローを記述する．2.1.2 項で述べたように，ワークフローの記述は，コンポーネント技術や実装の種類によらず，統一されたアプローチであることが望ましい．そこで，UML アクティビティ図[17] を用いて記述できるインタフェースを提供する（**図 4**）．アクティビティ図は，ワークフローを記述するための記法として一般的に普及しているため，開発者は本システムを比較的容易に利用できることが期待できる．

4.2.2 コンポーネントの割当て

ワークフローを記述した後，開発者はそれぞれのアクティビティに，実際のコンポーネントを対応づける．それらの各コンポーネントは，ネットワークを介して呼び出すことができれば，離れた場所のサーバに存在してもよい．本実装では，コンポーネントを検索するための機構[18] を利用することで，開発者が入力したクエリを基にコンポーネントをリポジトリから検索できるようにしている．

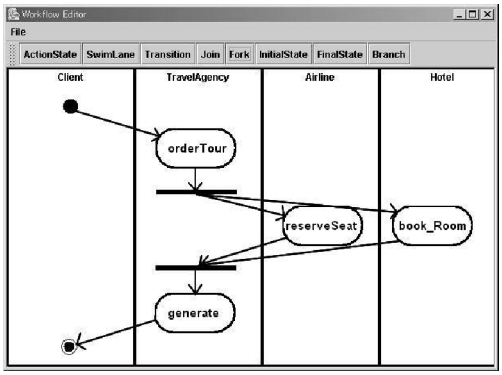

図 4 ワークフローエディタ
Fig. 4 Workflow Editor.

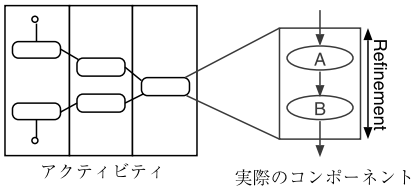

図 5 Refinement
Fig. 5 Refinement.

また，各アクティビティはそれぞれ 1 つのコンポーネントによって実現されるかもしれないし，複数のコンポーネントを連続して呼び出すことによって実現されるかもしれない．そこで 1 つのアクティビティに複数のコンポーネントの組合せを指定できるようにした（**図 5**）．本稿ではこれを Refinement と呼ぶ．

コンポーネントの割当てが完了すると，ワークフローに関する情報を文書として出力する．これは 2.1.2 項で述べたように，汎用的で統一された記述形式で可読文書として表すことが望ましい．本システムでは，開発者によって記述されたワークフローに関する情報を，開発者の選択により，Web Service 間の連携を行う際の XML ベースの言語である WSFL（Web Services Flow Language）仕様[8]，または BPEL4WS（Business Process Execution Language for Web Services）仕様[9] のどちらかに従った文書として出力する．

4.2.3 アダプタの生成

次に，コンポーネント間の異種性を吸収し，ワークフローに従って各コンポーネントを呼び出すためのアダプタを生成する．既存の EAI システムでは独自のサーバを用意し，ワークフローに関する情報を指定することで，複数のコンポーネントを呼び出せるようにしている．提案システムでは，統合を行う処理自体を，

分散コンポーネントとして動作するプログラムコードの形式で生成する．これによって，汎用的なサーバで動作させることができる．

生成するアダプタを，その機能によって以下の 2 つの部分に分ける．

- Proxy
  2.1.1 項で指摘したコンポーネント間の異種性に起因する問題に対処するため，各分散コンポーネントに対応する Proxy を生成する．それぞれの Proxy はコンポーネントが利用する分散コンポーネント技術に合わせた呼び出し方法でアクセスを行い，各プロセスを実行し，その結果を得る．これにより，異種分散コンポーネントを利用する際に，呼び出しごとに異なるアクセス方法を吸収する．統合する分散コンポーネントの種類を増やす場合は，Proxy の種類を増やすことで対応できる．なお，コンポーネント間の実装形式の異種性を吸収するため，Proxy を単一の形式の分散コンポーネントとして呼び出せるように生成する．
- Manager
  与えられたワークフロー情報に基づき，各分散コンポーネントの実行順序やアプリケーションプログラムの実行の遷移を管理する．そして適切なコンポーネントに Proxy を介してアクセスする．異なるコンポーネント間で連携を行う場合は，対応する複数の Proxy を呼び出す．
  統合して作り出そうとするアプリケーションプログラムごとに，外部から呼び出すことのできる分散コンポーネントとして，Manager を生成する．利用者は Manager を呼び出すことで，複数の異種分散コンポーネントを呼び出した結果を得る．

アプリケーションプログラムごとに，1 つの Manager と呼び出そうとするコンポーネントに対応する複数の Proxy を生成することによって，複数の異種分散コンポーネントを統合する新しいアプリケーションを開発する．生成されたアプリケーションプログラムにおける Proxy と Manager との関係を**図 6** に示す．

Proxy は，各コンポーネントのインタフェース情報を入手することによって生成される．インタフェース情報は，Web Service で一般的に利用されている WSDL（Web Service Definition Language）[19] を利用して記述する．そして Proxy そのものを Web Service として生成する．

Manager は，ワークフローの記述に従い生成される．ワークフローに関する情報が WSFL で記述されている場合は，記述された順番に従って各分散コンポーネントを呼び出すプログラムコードを，外部から呼び出すことができるように Web Service として生成する．この際に，実際に各コンポーネントを呼び出すのではなく，対応する Proxy を呼び出すようにする．BPEL4WS で記述されている場合は，BPEL 実行エンジン[20] を用い BPEL4WS の記述をそのまま解釈・実行することで Web Service として動作させることができる．したがって，BPEL4WS の記述を各コンポーネントに対応する Proxy を呼び出すように変更することで，そのまま Manager として利用する．

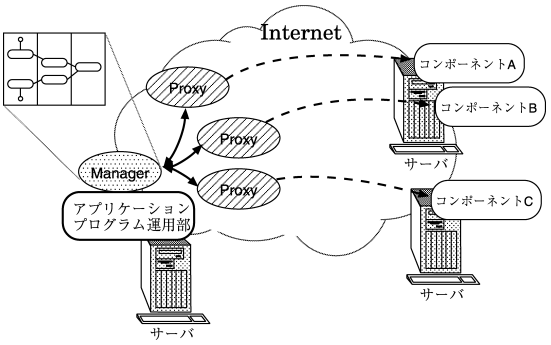

図 6 Proxy と Manager の関係
Fig. 6 Relation between Proxies and Manager.

4.2.4 アダプタのコンパイルと配備

コンポーネント間の連携では，ある処理の結果を別の処理の入力として単純に結び付けることはできず，値自体を加工することが必要になる場合がある．そこで，コンポーネント間で値を変換する必要がある場合は，データのマッピングを行うコードを Manager に記述する．同時に必要なビジネスロジックも記述する．そしてアダプタをアダプタリポジトリに登録する．

アプリケーションプログラム運用部は，アダプタ生成部で生成した Proxy や Manager を運用する Web Service のサーバや BPEL エンジンと，それらに配備するモジュールと，サーバの実行状況を監視するモジュールから構成する．まず生成されたアダプタをアダプタリポジトリから取得する．そして Proxy をコンパイルして Web Service のサーバに配備する．Manager については，Web Service として実装されたコードが生成されている場合は，Proxy と同様にコンパイルして Web Service のサーバに配備する．BPEL4WS で記述されている場合は，BPEL エンジンによって解釈，実行するため，そのまま BPEL エンジンに渡す．BPEL エンジンは渡された文書を解釈して，Web Service のコンポーネントを生成し，自動的に Web Service のサーバに配備する．これにより，利用者がクライアントから利用できるようになる．

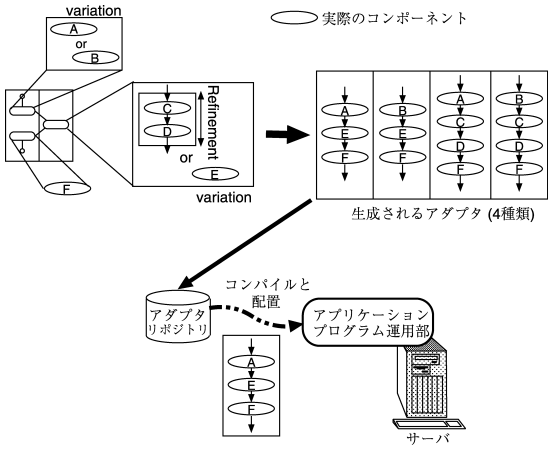

図 7 Variation を考慮したアダプタ生成
Fig. 7 Adapter generation with Variation.

4.3 アプリケーションプログラムの運用の問題を解決する方法

本システムでは，呼び出そうとするコンポーネントの処理に障害が発生したときに，同じ機能を持つ他のコンポーネントに自動的に処理を切り替えることにより処理を続行させる方法を提供する．そのために，まずアプリケーションプログラムの設計時に，開発者がワークフローエディタを用いて各アクティビティを実現するコンポーネント（または Refinement によるコンポーネントの組合せ）を複数指定する．そしてワークフローの開始から終了まで，考えられるコンポーネントのすべての組合せについてアダプタを生成する．アプリケーションプログラムの実行時には，このうちから任意のアダプタを実行する．アダプタから呼び出すコンポーネントが利用できなくなったときには，同じワークフローを実現する別のアダプタに実行を切り替える（**図 7**）．本稿ではこれを Variation と呼ぶ．このように，実行不可能な場合に代替として動作するアダプタをあらかじめ用意し，全体として実行不可能になる確率を低くすることで，信頼性の低下を回避する．

なお，アプリケーションプログラムを設計する時点で利用できるコンポーネントが，代替として実行される時点では利用できなくなっている可能性がある．代替として動作するアダプタの生成は，本来はアダプタの実行に失敗した段階で，その時点で利用できるコンポーネントを呼び出すように行うべきである．しかし，アダプタ生成の際に開発者がコンポーネントの割当てを行うため，アダプタの実行に失敗した段階でアダプタの生成を行うと，切替えに長い時間を要してしまう．そのため，代替として利用できなくなる可能性があるものの，設計を行う段階で開発者に利用できるすべてのコンポーネントを指定させることによって，代替として動作するアダプタをあらかじめ生成している．

本節ではまず，Variation を考慮したアダプタの生成方法を述べ，その後でアプリケーションプログラムの実行とアダプタの切替えの方法について述べる．

4.3.1 Variation を考慮したアダプタ生成

開発者は，まずワークフローを構成する各アクティビティについて，それぞれ複数のコンポーネントを割り当てる．そして，割り当てたすべてのコンポーネントについて Proxy を生成する．次に，ワークフローの開始から終了まで，考えられるコンポーネントのすべての組合せについて，Manager を生成する．これによってワークフローを実現するアダプタは，Manager の個数分だけ生成される．そして，生成した Manager と Proxy をアダプタリポジトリに登録する．

4.3.2 アプリケーションプログラムの実行

アプリケーションプログラム運用部は，実行するアプリケーションプログラムに対応するアダプタをアダプタリポジトリから入手する．この際に複数のアダプタから 1 つを選択する必要があるが，どのような優先順位で選択するかという基準を開発者があらかじめ指定できるようにする．これにより，たとえば組み合わせて実行するコンポーネントの実行時間や利用するための価格情報などを基に，複数のアダプタから適切なものを順番に選択できる．

そして選択したアダプタについては，4.2.4 項に示したようにコンパイルと配備を行い，クライアントから利用できるようにする．

4.3.3 アダプタの切替え

アプリケーションプログラム運用部はアダプタの実行時に，呼び出そうとする各コンポーネントを実行できるかどうか確認する．実行できないことを検知した場合（**図 8** (1)）は，実行するアダプタを切り替える．

まず，代替として動作させることのできるアダプタのうち，実行できなかったコンポーネントを利用しないアダプタをアダプタリポジトリから取得し，開発者があらかじめ指定した優先順位に基づいて適切なものを選択する．そして選択したアダプタをコンパイルし，今までのアダプタの代わりに配備する（図 8 (2)）．

また，クライアントからアダプタが呼び出されてアプリケーションプログラムが実行中だった場合は，実行中の処理を適切に継続させなければならない．このとき，実行できなかったコンポーネントの実行の直前までは正常に実行が行われている．したがって，実行開始から直前のコンポーネントの実行まで，実行中のアダプタとすべて同一のコンポーネントを呼び出す

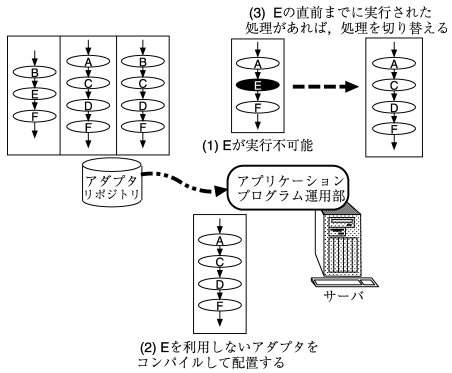

図 8 アダプタの切替え
Fig. 8 Adapter switching.

ものを優先してアダプタリポジトリから入手する．そのようなアダプタが見つからない場合は，直前のコンポーネントの実行結果を取り消し，さらに 1 つ前のコンポーネントの実行までが同一であるアダプタを入手する．アダプタが見つかるか，ワークフローの最初のアクティビティとして実行されたコンポーネントの実行結果が取り消されるまで，この処理を繰り返す．そのようなアダプタが見つかった場合は，残りのコンポーネントを実行して結果をクライアントに返す（図 8 (3)）．

5. 評価実験

5.1 アダプタによるオーバヘッド

提案システムでは，運用するアプリケーションプログラムはアダプタを介して分散コンポーネントを呼び出す．この際に要するオーバヘッドを計測する．

統合の対象とした Web Service，CORBA，EJB の各コンポーネントについて，同一の処理を行うメソッドを作成し，直接呼び出すときに要する時間と，アダプタ経由で呼び出すときに要する時間を計測した．

なお，アプリケーションプログラムを実行するコンピュータと各コンポーネントが動作するコンピュータを同一ネットワーク上に設置し，ネットワークトポロジによって呼び出し時間へ影響を与えないようにする．また，アダプタからコンポーネントを呼び出す前には，コンポーネントを実行できるかどうかを確認する必要がある．これは，コンポーネントに確認のためのメソッドを用意して呼び出すことによって実現できる．しかし事前にメソッドの呼び出しを行うことで，コンポーネントの名前解決が行われてしまううえに，コンポーネントを実行するサーバの実装によってはサーバ側でコンポーネントがキャッシュされてしまう．これによって呼び出し時間にばらつきが生じてしまい，測定結果の比較が困難になるため，ここではコンポーネントを実行できるかどうかを，ネットワーク的な到達性があるかどうかのみによって確認した．

表 1 メソッドの呼び出しに要する時間（単位：ms）
Table 1 Process time for method call (unit: ms).

WebService		CORBA		EJB	
直接	アダプタ	直接	アダプタ	直接	アダプタ
3991.60	4023.43	9.81	44.23	132.60	165.20
+31.83		+34.42		+32.60	

それぞれ 30 回測定し，その平均を**表 1** に示す．

5.2 アダプタの切替えによるオーバヘッド

アプリケーションプログラム運用部は，アダプタが呼び出すコンポーネントが実行不能であることを検知した場合に，同じワークフローを実現する別のアダプタに処理を切り替える．実行不能であることを検知してから切替えが終了するまではリクエストを受け付けられない．その間に要するオーバヘッドを測定した．

測定のため，同一ネットワークに存在する別々のコンピュータに 4 つのコンポーネントを配置した．内訳は，EJB のコンポーネントが 1 つ（E_1）と CORBA コンポーネントが 3 つ（C_1，C_2，C_3）である．ここで E_1 と C_1，および C_2 と C_3 を組み合わせることで，同一のワークフローをそれぞれ実現できる．そのようにコンポーネントを組み合わせるアダプタを，それぞれ AD_1，AD_2 として用意する．

これらのコンポーネントとは別のコンポーネントを利用して，同一のワークフローを実現するアダプタ AD_0 を配備し，AD_0 が利用するコンポーネントを実行不能な状態にした．クライアントから AD_0 を呼び出すと，AD_0 を運用するアプリケーションプログラム運用部は実行不能になったことを検知し，適切なアダプタを選択し配備する．この処理に要する時間を測定した．なおここでは，あらかじめコストを各コンポーネントに割り当て，それに基づいて最もコストの合計値が低いコンポーネントの組合せを呼び出すアダプタを選択する．そして，各コンポーネントにはコストの問合せのためのメソッドを用意する．このメソッドはリクエストに対して，あらかじめ設定されたコストの値を返す．また，簡単化のため AD_0 が利用するコンポーネントすべてを実行不能な状態にし，処理結果の一部を AD_1 や AD_2 で継続できないようにした．

アプリケーションプログラム実行部は，実行不能になったことを検知すると，まずアダプタリポジトリから同じワークフローを実現するアダプタ AD_1，AD_2 を読み込む（A）．次に各アダプタについて，呼び出すコンポーネントのコストを問い合わせる（B）．なお，

表 2 アダプタの切替えに要する時間（単位：ms）
Table 2 Process time for adapter switching (unit: ms).

(A) 読み込み		(B) コスト問合せ		(C) 配備	合計
AD_1	AD_2	AD_1	AD_2		
1.06	1.06	261.14	112.50	1,619.54	1,995.30

コストの問合せは呼び出す対象とするコンポーネントに対して行われるため，コストを問い合わせることによって各コンポーネントを呼び出せることも同時に確認する．そして，コストの合計値が低い方のアダプタを配備する（C）．この処理に要する時間を 30 回測定し，その平均を**表 2** に示す．

6. 考　　察

本章では，評価実験の結果を分析した後，アプリケーションプログラムの開発者が提案システムを利用して効率的に開発するための条件を示す．

6.1 オーバヘッドの分析

表 1 によると，生成したアダプタを経由して呼び出すことによって，呼び出しに要する時間が約 30 ms 増加することが分かった．

アダプタを経由して呼び出す際，コンポーネントを実行できるかどうかを確認するための時間と，Manager から Proxy のコンポーネントの名前解決を行うための時間と，実際に Proxy を呼び出すための時間が発生する．これらは呼び出そうとするコンポーネントの処理内容には依存しない．しかしそれぞれのコンポーネントの利用に必要な処理なので，アダプタから呼び出すコンポーネントの数に従い単純に増加する．

すでに呼び出したコンポーネントと同一のコンポーネントに存在するメソッドを呼び出す場合には名前解決の処理は不要なので，その処理のための時間は発生しない．したがって，アダプタから多くのメソッド呼び出しを行っても，少数のコンポーネントに対して行う場合には，ある程度はオーバヘッドが小さくなる．逆に同数のメソッド呼び出しであっても，呼び出されるメソッドがそれぞれまったく別のコンポーネントに存在する場合は，オーバヘッドが大きくなる．本質的にはオーバヘッドの多くは，Web Service として実装された Proxy を呼び出す際に，XML で記述された SOAP メッセージのエンコーディングやデコーディングの処理を行う時間である．したがって，Proxy と Manager 間の通信に CORBA のような呼び出しに要する時間の短い分散コンポーネント技術を利用するようにアダプタを生成することによって，さらにオーバヘッドを小さくできる．

また表 2 によると，アプリケーションプログラム運用部が利用不能となったことを検知してから，2 秒程度でアダプタの切替えが終了している．今回の実装では，コストの問合せをすべてのコンポーネントに対して逐次行っている．ここでは，AD_1 が呼び出す E_1，C_1 と，AD_2 が呼び出す C_2，C_3 にコストの問合せを行う．表 1 で示したように，EJB のコンポーネントの方が，CORBA のコンポーネントよりも呼び出しに要する時間が長いため，コストの問合せに関しても AD_1 に要する時間の方が長くなっている．このようにコストの問合せをすべてのコンポーネントに対して行っているため，候補となっているアダプタの数や，アダプタから呼び出すコンポーネントが増加すると，それにともないアダプタの切替えに要する時間も増加する．したがって，このようなコンポーネントの選択のために必要な処理を，すべてのコンポーネントに対して並行に行うことで，ある程度の一定時間でアダプタの切替えを終わらせることが可能である．

6.2 効率的に開発するための条件

提案システムを利用することで開発を効率的に行うためには，提案システムの利用者であるアプリケーションプログラムの開発者がある程度の知識や経験を保持していることを前提とする．それらを以下に示す．

- 分散コンポーネントに関する知識
  個々の分散コンポーネント技術については深く知る必要がないが，ワークフローエディタを利用して設計を行う際にどの程度の詳細度でアクティビティ図を記述すればよいかを判断する必要がある．そのため，分散コンポーネントを利用することによって一般的にどの程度の規模のアプリケーションプログラムが開発されているのかを知っている必要がある．
- UML を利用したソフトウェアの開発経験
  ワークフローエディタ上で描画するアクティビティ図の文法を理解している必要があり，さらにアクティビティ図を利用したソフトウェア開発経験があることが望ましい．
- アクティビティとコンポーネントの対応を判断するための知識
  Variation を実現するために，1 つのアクティビティに複数のコンポーネントを割り当てる．そのため，あるアクティビティを実現できるのはどのコンポーネントなのか，アクティビティの入出力は各コンポーネントのどの変数に相当するのかなど，アクティビティとコンポーネントの対応を判断する必要がある．したがって，抽象化されて記述したアクティビティによって，実際にはどのよ

うな処理を行うのかを判断できなければならず，開発しているアプリケーションプログラムの設計を十分理解している必要がある．

- アダプタの選択基準に関するドメイン知識
  アダプタを切り替える際の選択基準は開発するアプリケーションプログラムの利用目的によって異なるため，開発したアプリケーションプログラムが利用されるドメインに関する知識を十分に持っていることが望ましい．

7. まとめ

本研究では，異種分散コンポーネントを統合してアプリケーションを開発する際の問題を分析し，ワークフロー情報を用いて異種分散コンポーネントを統合するシステムの設計と実装を行った．このシステムを利用すると，異なるベンダが提供する異なる形式の分散コンポーネントを統合して動作させるためのプログラム（アダプタ）を分散コンポーネントとして自動的に生成することによって，アプリケーションを開発することが可能になる．さらに，開発したアプリケーションプログラムを運用する際に，アダプタが利用するコンポーネントが利用不能になった場合，他のコンポーネントを利用するアダプタに処理を切り替えることで，開発したアプリケーションプログラムの信頼性も向上させることができる．そして実験を行い，生成したアダプタを実行する際のオーバヘッドやアダプタの切替えの際のオーバヘッドが，どのような状況で大きくなるかを分析した．

今後はまず，オーバヘッドを短縮するための方法を検討する予定である．また，提案システムではアダプタの切替えを行っている間はリクエストを受け付けることができないが，これを回避するための方法も検討する予定である．

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) Brown, A.: *Component-Based Software Engineering*, IEEE Computer Society Press (1996).
2) Brown, A.: *Large-Scale, Component-Based Development*, Prentice Hall PTR (2000).
3) Orso, A., Harrold, M.J. and Rosenblum, D.: Component Metadata for Software Engineering Tasks, *Proc. 2nd International Workshop in Engineering Distributed Objects* (*EDO 2000*), pp.126–140 (2000).
4) Sun microsystems: Enterprise JavaBeans, Specification Version 2.1 (2003). http://java.sun.com/products/ejb/
5) Object Management Group: Common Object Request Broker Architecture: Core Specification (3.0.2), OMG Document (2002). http://www.omg.org/technology/documents/formal/corba_2.htm
6) The 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/
7) Microsoft Corp.: XLANG — Web Services for Business Process Design (2001). http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
8) IBM Software Group: Web Services Flow Language (WSFL 1.0) (2001). http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
9) 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
10) Kaplan, A., Schmerl, B. and Wileden, J.C.: Automating Interoperability For Heterogeneous Software Components, *International Workshop on Component-based Software Engineering*, pp.111–114 (1999).
11) 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).
12) Aldrich, J., Sazawal, V., et al.: Language Support for Connector Abstractions, *17th European Conference on Object-Oriented Programming* (*ECOOP 2003*), pp.74–102 (2003).
13) BEA Systems: BEA eLink Documentation (2002). http://e-docs.bea.com/elink/
14) Sybase: e-Biz Integrator 3.6.2 Product Documentation (2002). http://sybooks.sybase.com/bzr0362e.html
15) webMethods: webMethods Glue 6.0 (2005). http://www.webmethods.com/Products/ESP/Glue/Glue60
16) Nagura, M., Iijima, T., Takada, S. and Doi, N.: Automated Adapter Generation for Gluing Heterogeneous External Components, *Proc. 14th International Conference on Software & Systems Engineering and their Applications* (*ICSSEA 2001*), pp.1–8 (2001).

17) Dumas, M. and ter Hofstede, A.: UML Activity Diagrams as a Workflow Specification Language, *Fourteenth International Conference on Advanced Information Systems Engineering*, pp.76–90 (2002).
18) Usanavasin, S., Nakamori, T., Takada, S. and Doi, N.: A Multi-faceted Approach for Searching Web Applications, 情報処理学会論文誌, Vol.46, No.5, pp.1256–1265 (2005).
19) World Wide Web Consortium: Web Service Definition Language (WSDL) 1.1, W3C Note (2001). http://www.w3.org/TR/2001/NOTE-wsdl-20010315
20) ActiveBPEL, LLC: ActiveBPEL engine v1.0.11 (2005). http://www.activebpel.org/download/index.html

(平成 17 年 5 月 9 日受付)
(平成 17 年 11 月 1 日採録)

名倉 正剛（学生会員）

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．現在は分散コンポーネントシステムに関する研究に従事．

河野 泰隆

2004 年慶應義塾大学理工学部卒業．同年同大学大学院理工学研究科修士課程入学．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．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 各会会員．

土居 範久（正会員）

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・第 3 部副部長，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，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」領域統括，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．