差分更新を実現する分散オブジェクト再構成ミドルウェアの実装と検証

寺島美昭[†] 別所雄三[†] 宮内直人[†]
中川路哲男[†] 鹿間敏弘[†] 福岡久雄[††]
佐藤文明[†††] 水野忠則[†††]

状況に応じて多様なサービスを提供するユビキタス環境では，単一の携帯機アーキテクチャ上で複数の無線方式を切り替えながら利用する基盤が重要である．本論文では IDL（Interface Definition Language）を用いたオブジェクト部品により構成される上位レイヤ機能を対象に，動的な再構成を実現する Configuration ミドルウェアを提案する．このミドルウェアは差分となる限定された更新情報を用いて，オブジェクトの再起動手順を実行する差分更新管理方式を備える．またサーバを特定することなく IDL Call を確立する間接接続方式を用いて，再構成により変化するクライアント/サーバ関係に対応する．実際に Configuraiton ミドルウェアの実装と評価を行い，差分更新の効果を確認した．

An Implementation and Evaluation Related to Configuration Middleware Enabling Difference Update of Distributed Object Structures

Yoshiaki Terashima,[†] Yuzo Bessho,[†] Naoto Miyauchi,[†]
Tetsuo Nakakawaji,[†] Toshihiro Shikama,[†] Hisao Fukuoka,[††]
Fumiaki Sato[†††] and Tadanori Mizuno[†††]

In this paper, we propose the configuration middleware for dynamic reconfiguration of wireless communication software. The software consists of distributed objects whose interface is specified by IDL (Interface Definition Language). Analysis of the difference between complex object structures and execution of reconfiguration process tend to suffer from low performance and reliability of terminal in runtime. Our middleware provides two functions for updating the difference between the new object structure to be downloaded and the existing one in the terminal. The configuration management function can change object configuration by minimum reconfiguration process. The indirect connection function enables objects to invoke IDL calls without specifying a server in dynamically changing object structure. Finally, we show the effectiveness of our middleware by the experimental evaluation of an overhead characteristic for actual reconfiguration of wireless communication modes.

1. はじめに

ネットワーク（以降ではコアネットワークと呼ぶ）からダウンロードしたソフトウェアを用いて，運用中に多様なサービスを提供するユビキタス環境が期待されている．ここでは単一の携帯機上で複数の無線方式を切り替えながら利用する環境が重要である．ソフトウェア無線は，DSP（Digital Signal Processor）やFPGA（Field Programmable Gate Array）など汎用回路を用いた単一アーキテクチャ上で，マルチモードやマルチバンドなど複数の無線方式を提供する[1)]．一方，無線方式の上位レイヤ機能を実現する GPP（General Purpose Processor）上の開発では，OS が提供する API（Application Programming Interface）仕様の公開により，プロトコル制御などの論理的な処理を，可搬性のあるソフトウェアを用いて動的に再構成する取り組みがある．代表的な TOPPERS プロジェクト（Toyohashi OPen Platform for Embedded Real-time Systems）[2)] では，ITRON が提供するメモリや

† 三菱電機株式会社情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation
†† 松江工業高等専門学校情報工学科
Department of Information Engineering, Matsue National College of Technology
††† 静岡大学情報学部
Faculty of Information, Shizuoka University

タスク管理による動作環境に対して，運用中の交換が可能なソフトウェア単位を拡張する研究を行っている．これにより，アプリケーションやデバイスドライバの動的な入換え（更新と呼ぶ）を実現する．この方法はITRON に最適化した効率的なメモリ制御により，高い更新性能や安全性を備えた再構成を実現できる．また OSGi Alliance[3] は，無線方式を実現するデバイスやアプリケーションの再利用性，軽量化の実現を目的に，JVM（Java Virtual Machine）上のオブジェクトフレームワークを提案している．ここでは，Java オブジェクトのライフサイクルを管理する共通機能や，Java API による利用方法が規定されている．

このように，今後は DSP/FPGA 技術の発展や API の共通化により，無線方式を実現するソフトウェアの部品化が進むと予想される．このため携帯機の動作環境とは独立に，ソフトウェア部品間の多様な組合せが可能な，流通性のある開発が重要である．我々の研究は，無線方式を実現するソフトウェアが，CORBA（Common Object Request Broker Architecture）IDL（Interface Definition Language)[4] 仕様を介して相互接続する，分散オブジェクト（オブジェクトと呼ぶ）構成であることを前提としている．IDL メソッド呼び出し（IDL Call と呼ぶ）を用いる開発では，IDL Call を呼び出すオブジェクトをクライアント，呼び出される側をサーバと呼ぶ部品として取り扱う[5]．この部品間を相互接続可能な関係として明確に分離することにより，たとえば複数のベンダが個々のオブジェクトを，Java や C++など異なるオブジェクトモデル（実装モデルと呼ぶ）を用いて独立に開発できる．このように IDL Call の採用は，動作環境や実装モデルを限定しないソフトウェア部品の流通性を実現する．また同時に，ユビキタス環境上で無線方式を実現するソフトウェア構成に対して，組合せの自由度を与えることができる．

本論文では，IDL Call に基くオブジェクト構成を差分更新することにより，運用中に動的な無線方式の切替えを実現する Configuration ミドルウェアを提案する．差分更新とは，携帯機で動作しているオブジェクト構成に対して，異なるレイヤ機能やアルゴリズムを実現するオブジェクトのみの更新により，無線方式を変更する方法である．ダウンロード量や再構成の手順を削減できるため，更新性能の向上に効果がある．このため，Configuration ミドルウェアは，オブジェクト部品一式（パッケージと呼ぶ）と，携帯機に搭載されている構成との違いを分析し，差分となる部品のみを適用してオブジェクト構成を更新する．しかし，IDL Call を用いた差分更新では，流通性を実現する代償として，2 つの負荷の発生が問題となる．

第 1 はオブジェクト構成を再起動する手順分析の負荷である．再起動とは新たなオブジェクト構成を構築するために，差分となる個々の新旧オブジェクトが実行する手順である．ここで 1 つのオブジェクトを更新する影響は，クライアントやサーバとして関連付けられる他のオブジェクトにも及ぶ．このため複数のオブジェクト更新の影響が，タイミングによる遅延や，必要な IDL Call を起動するオブジェクトを発見できない未検出の問題を発生させる．この問題を回避するためには，変更の有無にかかわらずすべてのオブジェクトの IDL Call 関係を分析する必要がある．第 2 は新たに連携するクライアント/サーバ関係を確立（IDL Call 確立と呼ぶ）する負荷である．IDL Call 確立は，クライアントとサーバの間で IDL Call を行う通信路を確保する処理である．差分更新の過程では，クライアントは開発時に想定していないサーバの特定が必要なため，検索のために複雑な手順による負荷が発生する．以上，ここで述べた課題は，差分更新による手順削減の効果を損なう恐れがある．このため，Configuration ミドルウェアは 2 つの方式を提供する．差分更新管理方式は，オブジェクトと IDL Call 仕様の差分に限定した情報を用いて，タイミングと未検出の問題を回避した再起動手順を実行する．また間接接続方式は，クライアントがサーバを特定することなく IDL Call を確立する．実際に Configuration ミドルウェアの実装を行い，再構成時間に対する差分更新の効果を確認した．2 章ではシステム構成と課題，3 章では Configuration ミドルウェアによる解決，4 章では実装と検証を述べる．5 章では関連研究を概観し，6 章でまとめを行う．

2. ソフトウェア更新環境

2.1 システム構成

本論文で対象とするユビキタス環境は，無線方式を実現するソフトウェアを，IDL 仕様によりブラックボックス化したオブジェクト部品の集合として構成する．無線方式に対する分散オブジェクト技術の導入の代表的な研究は，JTRS（The Joint Tactical Radio System）が提供する SCA（Software Communications Architecture)[6] である．SCA は GPP 上に配置される，論理的なプロトコル制御を実現する Waveform ソフトウェアを，オブジェクトのビルディングブロック構成としてモデル化している．このモデルは無線方式を実現するソフトウェアの国際的な流通や，新たなビジネス展開の基盤として期待されている．以降では

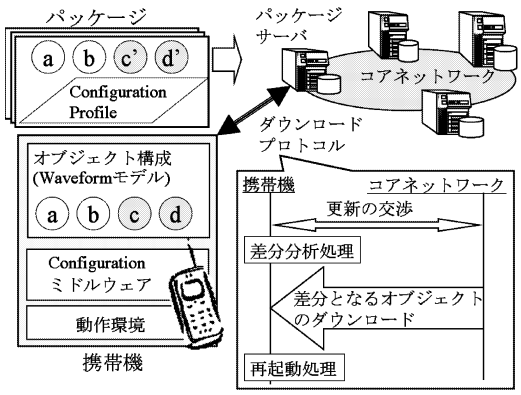


図 1 再構成を実現するシステム構成
Fig. 1 A ubiquitous system structure.

SCA が規定する Waveform モデルに基づくオブジェクト構成を対象に，ネットワークダウンロードを用いて差分更新を実現する方法を述べる．

機能更新を行う携帯機の種類は，携帯電話や PDA（Personal Digital Assistance）などリソースが限られる機器だけでなく，携帯型コンピュータなど比較的リソースが豊富な機器も存在する．これらに搭載される動作環境も様々である．また今後のソフトウェアやハードウェアの動的な再構成技術の革新により，無線方式を実現する機能の共通化による機能粒度の細分化が進展し，ソフトウェアの部品化が進むと予想される．この携帯機の動作環境と無線方式の多様性を考慮すると，あらかじめすべての状況や環境を想定したパッケージ開発は不可能である．このため携帯機に搭載されるソフトウェア部品間が，動作環境やパッケージの違いに依存せず連携できることが重要となる．また，更新される新旧オブジェクトの組合せ状況に応じて再起動手順は様々である．これに対して，手順の違いをオブジェクトから分離することにより，独立性の高い開発によるソフトウェア部品の流通性を実現する．

この流通性のあるオブジェクト開発が可能なユビキタス環境として，**図 1** に示すシステム構成を規定した．Waveform モデルに基くオブジェクト構成は，それぞれの無線方式のためのアプリケーションロジック（実装アルゴリズムと呼ぶ）を隠蔽し関係を単純化する．これにより，動作環境や実装モデルとは独立に，自由度の高い連携を実現できる．ここで携帯機に搭載する Configuration ミドルウェアは，コアネットワークからパッケージを取得し，オブジェクト構成を更新する．このパッケージは，以下の 2 種類のソフトウェアからなる．

- Configuration Profile
  無線方式を実現するオブジェクト構成を，XML（eXtensible Markup Language）を用いて定義したファイル．
- オブジェクト
  無線方式を実現するソフトウェア部品．IDL 仕様によるブラックボックスとして開発する．

利用者の要求や，無線エリア間の移動検出による無線方式の切替え要求の発生により，Configuration ミドルウェアは再構成を開始する．更新の交渉では，コアネットワーク上のパッケージ検索や，ターゲット携帯機へのパッケージ適用に関する認証を行う．更新の交渉が完了すると，検索されたパッケージの Configuration Profile のみをダウンロードする．差分分析処理は，この Profile と現在の携帯機のオブジェクト構成の比較から差分を抽出し，実行する再起動手順を決定する．再起動処理は，決定した手順に基づき，実際に差分となる新旧のオブジェクトに対して，起動や停止，および IDL Call 確立の指示を与える．この例では，Configuration ミドルウェアが，差分となるオブジェクト c' と d' のダウンロード，および旧構成のオブジェクトである c と d の停止，さらに新メンバである c' と d' の起動と関連する IDL Call の確立を行うことにより，無線方式を更新する．

Configuration ミドルウェアは，流通性のあるオブジェクト開発を実現するために，相互接続のための共通仕様である IDL Call の情報を用いて構成を分析する．この方法により，特定の実装モデルに依存せずに，差分となるオブジェクト構成を抽出できる．また差分更新の状況に応じた再起動手順の違いを Configuration ミドルウェア内に隠蔽することにより，これらの手順とは独立にオブジェクトを開発する．以降では，流通性のあるオブジェクトによる構成を対象に，差分更新を実現する差分分析処理と再起動処理を議論する．

このオブジェクト構成の問題を明確にするために，本論文では無線方式の切替え時に，新旧の無線通信間で連続性がない場合を想定する．このため，切り替えられる新旧オブジェクト間では状態を引き継がず，再構成の完了時に各オブジェクトが初期起動する場合が対象である．携帯機の初期起動は，連携するすべてのオブジェクト間の IDL Call が確立され，受信や送信待ちの状態になることを意味する．

2.2 再構成の課題

SCA が規定する Waveform モデルに基づくオブジェクト関係を**図 2** に示す．RF（Radio frequency）から Speaker 方向の PushData_a() と PushData_b() を用いた音声やデータの伝播が受信に相当し，Push-

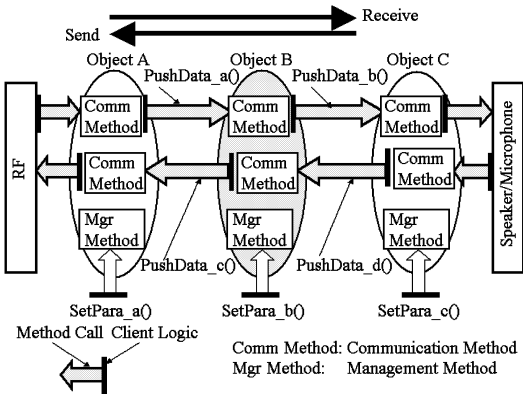


図 2 SCA Waveform モデルを実現するオブジェクトの関係
Fig. 2 Relationship between distributed objects based on SCA waveform model.

Data_d() と PushData_c() による Microphone からの逆の流れが送信を意味する．また SetPara_a()〜SetPara_c() によるパラメータ設定により，変調方式などを変更する．各オブジェクトは，PushData_a() などプロトコル制御のための通信メソッド（Communication Method），および SetPara_a() などの管理メソッド（Management Method）を IDL Call 仕様として公開することにより，動作環境や実装モデルの違いに関係なく相互接続を実現できる．

ネットワークダウンロードを用いて，このオブジェクト構成を差分更新する処理時間（再構成時間と呼ぶ）は，以下から計算できる．

再構成時間 = 差分分析時間 + (単位ダウンロード時間 + 単位再起動時間) × 更新オブジェクト数

単位再起動時間とは，1 つのオブジェクトの再起動時間の平均である．また単位ダウンロード時間は，コアネットワークから 1 つのオブジェクトを取得する平均時間である．これら 2 つの時間は，利用する環境や状況により決まることから，再構成時間の短縮には，差分分析時間と単位再起動時間の圧縮が必要なことが分かる．しかし，IDL Call に基づくオブジェクト構成では，特定の動作環境からの分離や，実装モデルに依存しない相互接続を実現するための 2 つの負荷が，これらの圧縮を妨げるという課題がある．

第 1 の課題は差分分析処理の負荷である．動作環境依存のソフトウェア再構成技術では，関数を呼び出すメモリ空間内のアドレスに制約を与えることにより，新旧のソフトウェア単位間の違いを吸収している．この方法では更新するソフトウェア部品を，メモリ上で再配置することにより再起動を実現できる．しかし，オブジェクト構成は，動作環境として OS 管理下にあるメモリやプロセス制御とは独立に実現されるため，ソフトウェア単位間の関係は IDL Call のみで決定される．ここでは差分として抽出される構成の変化が，クライアントやサーバとして他のオブジェクト（隣接オブジェクトと呼ぶ）へ与える影響も考慮する必要がある．この IDL Call の確立は必ずクライアントから実行するため，再起動は差分に関連するクライアントの検出，および IDL Call 確立の指示が基本的な手順である．しかし，複数のオブジェクトが実行する再起動手順の関係により，無駄な遅延が発生する可能性がある．たとえば図 2 の構成において Object B は，Object A と Object C に対して PushData_c() と PushData_b() を呼び出すクライアントとして動作する．ここでは Object B による IDL Call の確立時に，Object A や Object C の起動が完了していないなければ，待ち時間による遅延が発生する．また PushData_b() と PushData_d() により連携する Object B と Object C は，双方がクライアントであり，かつサーバとして動作する．この場合は Object B が PushData_b()，また Object C が PushData_d() を，それぞれクライアントとして確立するため，双方のオブジェクトの起動待ちが競合し，タイムアウトを待つ遅延が発生する可能性がある（これら遅延の発生をタイミング問題と呼ぶ）．サーバ起動や競合の発生による遅延を回避するためには，クライアントだけでなく，サーバにもタイミングを計るための指示が必要である．さらに Object_B がサーバとして動作する場合は，Object A と Object C がクライアントとして，PushData_a() と PushData_d() を確立する．しかし，構成がつねに変化する環境では新旧のオブジェクト関係が混在するため，必ずしも Object B は自身を呼び出すすべてのオブジェクトを特定できない．この結果，必要な再起動の指示を与えることができない場合が発生する（未検出問題と呼ぶ）．これらタイミングと未検出の問題を回避するためには，変更の有無にかかわらず新旧すべてのオブジェクトの IDL Call 関係の解析が必要である．しかし，この解析を再構成時に行うことは，再構成時間に対する手順削減の効果を損う恐れがある．この課題に対して，オブジェクト構成の単純な比較から再起動手順を決定する，差分更新管理方式による解決を図る．

第 2 の課題は，再起動処理における IDL Call 関係確立の負荷である．クライアントはサーバとなるオブジェクトの実体を特定することにより IDL Call を行う．しかし，差分更新では開発時には想定していないオブジェクトへの IDL Call が必要となる．このため，

新たにサーバとなるオブジェクトが，自身を特定する識別子 IOR（Interoperable Object Reference）を，ネーミングサービスを提供するオブジェクトや特定のファイルなど，Waveform モデルを形成するオブジェクト以外の第三者機能に登録する．クライアントは，この IOR を検索することにより実体を特定し，IDL Call を確立する．しかし，この方法は 1 つの IDL Call を確立するごとに第三者機能へアクセスするため，この機能との間での新たな IDL Call や，ファイル I/O の負荷が発生する．またサーバとなるオブジェクトが変更されていない場合，IDL Call を発行するクライアントは，この変更の有無をあらかじめ判断できない．このため，不必要な検索や複雑な登録手順が発生する可能性がある．この課題に対して，第三者機能を介することなく，新たな構成に対応した複雑な手順を必要とせずに IDL Call を確立する，間接接続方式による解決を図る．

3. Configuration ミドルウェア

3.1 ミドルウェアの構成と役割

図 3 に示すように，Configuration ミドルウェアは CORBA 準拠の ORB（Object Request Broker）が提供する IDL Call 確立機能に対して，差分更新を実現する機能を拡張している．Configuration Repository は，ターゲット携帯機のオブジェクト構成情報を保持するために，ORB に追加するリポジトリ機能である．最新のオブジェクト構成として，再構成であるか否か，また，その再構成状況の違いにかかわらず，ORB が実行するすべての IDL Call 確立に関する履歴を機械的に記録する．この ORB が内部のリポジトリを用いる方法より，オブジェクトの実装アルゴリズムとは独立に情報を蓄積できる．また Configuration Manager は，実際に差分分析処理を行うとともに，再起動手順を実行するスケジューラ機能である．差分更新の状況に応じた再起動手順を決定するために，差分分析の結果を再起動手順分析ルールに基づいて分析する．これらの機能を用いて差分更新管理方式を実現するとともに，IDL Call 接続機能に対して間接接続方式を拡張する．

図 4 は，SCA が提供する Waveform モデルに基づくオブジェクト構成である．OSI レイヤごとに Application，Resource，Device の各クラスにより階層化している．Application はプロトコルの論理的なロジックの実現，Resource は共通アルゴリズムの管理，Device は DSP や FPGA などの汎用回路とのインタフェースを実現する．SCA は無線方式のマルチベンダ開発を目的に，これらのオブジェクト間が連携するための仕様を IDL Call として標準化している．

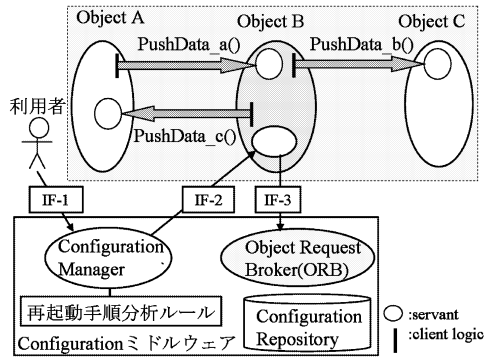


図 3 Configuration ミドルウェアによる再構成の実現
Fig. 3 Configuration of distributed objects structure by the configuration middleware.

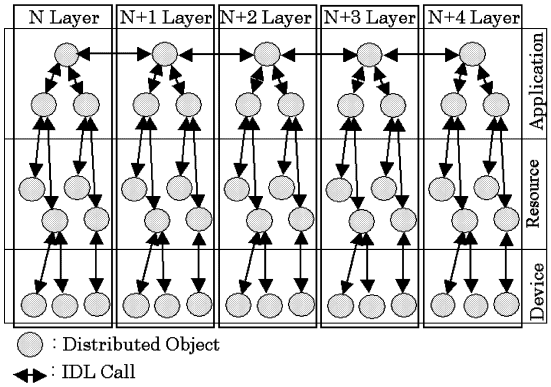


図 4 SCA Waveform モデルに基づくオブジェクト構成
Fig. 4 Distributed object structure based on SCA waveform model.

Configuration ミドルウェアは，この Waveform モデルを構成する個々のオブジェクトの実体を，IDL Call に基づいて抽象化したオブジェクト名と IDL Call 名を用いて管理する．たとえばベンダの開発者は，名前を用いて Configuration Profile の構成を設計する．Configuraion Manager は，これらの設計情報から差分分析処理を行い，さらに再起動処理では，名前からオブジェクトの実体を検索して再起動手順を指示する．この指示は 3.2 節に説明する再構成インタフェースを用いて行う．

これらのオブジェクト名と IDL Call 名には，バージョン番号を付与している．Configuration Manager が行う差分分析処理では，Configuration Profile と Configuration Repository に記録されている双方のバージョン番号を比較する単純な分析から，オブジェクトの実装アルゴリズムと IDL Call の変更（差分種別と呼ぶ）を検出する．この検出結果は，再起動時にお

表 1 再構成インタフェース
Table 1 Interface specification for enabling difference update.

分類	型	名称	パラメータ	役割
IF-1	boolean	configure	なし	configuration の要求
	boolean	configConfirm	なし	configuration 完了の確認
	ConfigInfoList	getStatus	なし	Configuration Repository 情報の取得
IF-2	boolean	connect	IDL Call 名，オブジェクト名	IDL Call の確立指示
	boolean	disconnect	IDL Call 名	IDL Call の解放指示
IF-3	Object	bind	IDL Call 名，オブジェクト名	IDL Call の確立
	boolean	registIOR	IDL Call 名，IOR	IOR を Configuration Repository に登録
	boolean	close	IDL Call 名	IDL Call の解放

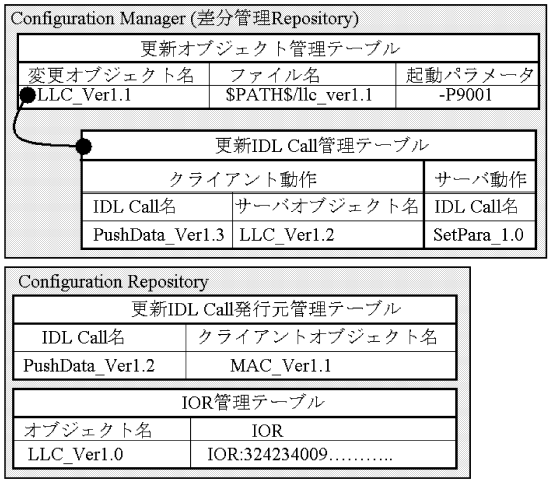


図 5 差分更新のためのテーブル構成
Fig. 5 Table structure for enabling difference update.

ける差分分析処理と再起動処理間での受け渡しのために，**図 5** に示す差分管理 Repository に記録しておく．これは差分更新の実行時に，Configuration Manager が一時的に情報を保持するリポジトリである．ここで更新される各オブジェクトごとに，自身がクライアントとして呼び出す動作情報（クライアント動作）である IDL Call とサーバ，およびサーバとして提供する情報（サーバ動作）である IDL Call のセットを 1 つの差分情報として管理する．このためオブジェクト，あるいは IDL Call のいずれかのバージョンに違いがある場合は，更新オブジェクト管理テーブルと，リンクされる更新 IDL Call 管理テーブルに 1 セット分の差分情報を登録する．

以降では，この Configuration ミドルウェア構成を用いて，差分更新を実現するための 2 つの方式を説明する．

3.2 差分更新管理方式

再起動手順は，Configuration Manager が**表 1** に示す再構成インタフェースを用いて，新旧の差分となるオブジェクトへ指示を与えることにより実現する．分類の IF-1〜IF-3 は，図 3 に示す位置付けである．IF-1 は Configuration ミドルウェアに対する再構成の指示や現在の構成確認を行う．IF-2 は Configuration Manager がオブジェクトへ与える再構成のための指示である．対象となるオブジェクトが採用している実装モデルの違いを IDL Call が吸収することにより，実体の入換えに対応した指示が可能となる．また IF-3 はオブジェクトが Configuration ミドルウェアに対して，IDL Call の確立や解放を要求するために用いる．これらのインタフェースにより，再起動するオブジェクトは，IF-2 の connect()，disconnect() による指示を受けて，クライアントとして F-3 の bind()，close() を用いた実際の IDL Call 確立，解放の要求を行う．この機械的な実装により，各オブジェクトを特定の再起動手順には依存しないブラックボックスとして開発できる．

再起動手順の決定は，Configuration Manager が差分管理 Repository に記録されている差分種別を，**表 2** に示す再起動手順分析ルールに基づいて分析することにより行う．このルールは，差分種別が IDL Call によるオブジェクト間の関係により及ぼす影響を分析し，対応する再起動手順を整理したものである．ここで 1 つのオブジェクトに必要となる再起動手順は，差分種別に対するクライアントとサーバの双方の依存関係により異なる．このため，クライアントとして果たす役割から再起動手順を決定する分析を，クライアント分析において行う．同じくサーバとしての役割からの決定を，サーバ分析で行う．決定される再起動手順は，タイミング問題と再起動問題の発生を○，発生しない場合を×で表している．さらに再起動手順の実行/未実行により，更新オブジェクトに再起動指示を与えるか否かが決まる．

決定された再起動手順を実行するために，差分管理 Repository に不足する情報は，Configuration Repository の更新 IDL Call 発行元管理テーブルが補う．このテーブルはターゲット携帯機の ORB 機能が，つねに記録している IDL Call 名とサーバとなるオブジェ

表 2 再起動手順分析ルール
Table 2 Decision rule for configuration process.

No.	差分種別			再起動手順		
	処理	IDL Call 仕様	隣接オブジェクトの実装アルゴリズム	タイミング問題	未検出問題	再起動手順の実行
1	クライアント分析	変更あり	変更あり/変更なし	○	×	実行
2		変更なし	変更あり	○	×	実行
3			変更なし	×	×	実行
4	サーバ分析	変更あり	変更あり/変更なし	○	×	未実行
5		変更なし	変更あり	○	×	未実行
6			変更なし	×	○	実行

クト名の関係情報である．再構成実行前からつねに Configuration Manager が関係情報を単純に記録するため，軽量の処理であると同時に，オブジェクトの実装アルゴリズムには特定の手順が必要ない．これは旧構成となる関係情報の一部を利用して，差分情報から判断できない IDL Call 関係を分析する方法である．この結果，再構成時には，新旧すべてのオブジェクト構成を分析する負荷を排除できる．

以降では図 3 に示す，PushData_a() と PushData_b() により連携するオブジェクト構成に注目し，Object B が更新オブジェクト管理テーブルに記録された場合を例に，再起動手順分析ルールを用いた再起動手順の決定を説明する．まず，No.1，No.2，No.4，No.5 のルールに該当する差分情報は，隣接オブジェクトを含めて差分として抽出できる場合である．この中で No.1 と No.4 は，PushData_a() と PushData_b() に変更がある差分種別である．これらは隣接オブジェクトとなる Object A と Object C でも，この IDL Call に対するクライアント，あるいはサーバとしての処理が変更される．このため 3 つのオブジェクトは，すべてバージョンが変更されることにより差分として検出される．これによりすべてのオブジェクトに対する指示が可能となるため，未検出問題は発生しない．しかし，クライアントとサーバ双方が変更されるため，タイミング問題を回避する必要がある．また隣接オブジェクトの実装アルゴリズムに変更がある，No.2 と No.5 の場合も同様である．これらの場合は，各更新オブジェクトの更新 IDL Call 管理テーブルに記録されている，クライアント動作とサーバ動作を IDL Call 名をキーとして比較することにより Object A，Object B，Object C のクライアント/サーバ関係が判断できる．この結果，クライアント分析となる No.1 と No.2 の場合には，Configuration Manager が Object C の起動を確認した後に，Object B に対して connect() 命令による指示を行う．またサーバ分析となる No.4 と No.5 の場合は，再起動処理は行わない．これは差分となる Object A に対して行われるクライアント分析において，No.1 と No.2 の適用により PushData_a() の確立が実行されるためである．これらの分析により，サーバ起動のタイミング問題を回避した再起動手順を実行できる．

また，No.3 と No.6 のルールは，それぞれ隣接オブジェクトである Object A と Object C に変更がないため，これらのオブジェクトが差分情報として検出されない場合である．クライアント分析となる No.3 のルールでは，Object C が更新されず起動状態が継続しているため問題は発生せず，Object B に対して，いつでも PushData_b() の確立を指示できる．しかしサーバ分析となる No.6 のルールでは，Object A を差分として検出できないため未検出問題が発生する．このとき，Configuration Manager は，IDL Call 名をキーとして更新 IDL Call 発行元管理テーブルからクライアントとなる Object A を検索して connect() を発行する．この結果，未検出となる Object A は，PushData_a() の再起動を実行できる．

次に同じく図 3 に示す，PushData_a() と PushData_c() により連携するオブジェクト構成に注目し，Object A と Object B が更新オブジェクト管理テーブルに記録される場合を説明する．これは双方がクライアントであり，かつサーバとして動作する状況に該当する．それぞれのオブジェクト内に，クライアント分析である No.1 と No.2，およびサーバ分析である No.4 と No.5 の，いずれか 1 つ以上の組合せが存在することになる．このため Object A による PushData_a() と，Object B による PushData_c() の確立に関して，起動待ちの競合回避が必要である．この場合，Configuration Manager は，1 つのオブジェクトの起動時に，サーバ分析をクライアント分析に優先して実行する．先に実行する IDL Call 確立を，つねにサーバ側となるオブジェクトが IDL Call 確立を要求する前に処理する方法により，競合の発生によるタイミング問題を回避できる．

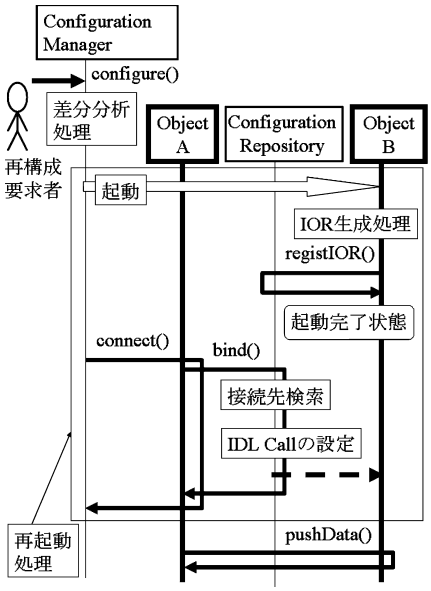


図 6 再起動シーケンス
Fig. 6 Bind sequence by the indirect IDL Call.

以上の決定により，すべてのオブジェクト構成を分析することなく，タイミング問題と未検出問題を回避した再起動手順を実行できる．この単純化した比較による差分分析の実行，および再起動手順分析ルールに基づく差分情報に限定した分析により，第 1 の課題である差分分析処理の負荷を軽減できる．

3.3 間接接続方式

間接接続方式は，一般の IDL Call が実現するオブジェクトと動作環境の分離に加え，クライアントとサーバの関係も透過とした IDL Call を実現する方法である．つまり ORB が適切なサーバへ IDL Call をディスパッチすることにより，クライアントはサーバとなるオブジェクトの実体を特定することなく関係を確立する．これは Configuration Repository の IOR 管理テーブルに記録する IOR 情報の利用により実現できる．**図 6** は，この方式による IDL Call の動作である．configure() の受信により再起動を開始した Configuration Manager は，差分更新管理方式により決定した再起動手順に基づき，Object B を新たに起動し，Object A に対して pushData() の確立を指示する．ここで起動された Object B は，registIOR() を介して IOR 管理テーブルに，自身のオブジェクト名と IOR を登録する．この後，Object A は，connect() により取得した IDL Call 名とサーバオブジェクト名を用いて bind() を要求する．この要求を受けた ORB は，IOR 管理テーブルからオブジェクト名に対応する IOR を検索して IDL Call を確立する．この結果，Object A は，サーバの実体が Object B であることを意識することなく IDL Call を確立できる．

この構成では Configuration Repository によるオブジェクト名と IOR の関係を，再構成インタフェース IF-3 が Configuration ミドルウェア内に隠蔽して管理する．このため Object A には，Object B に対して行う registIOR() や bind() による要求以外に，オブジェクトの実体を検索する手順を必要としない．この結果，運用中に更新されるオブジェクトをあらかじめ特定する必要のないオブジェクト開発が可能となる．また bind() を受けて実際に IDL Call を確立する ORB は，再起動手順の実行時に，該当するオブジェクトが記録されている Configuration Repository が，必ず携帯機の同一メモリ空間内に存在することに限定して処理を行う．このため，IDL Call 確立時に，第三者機能との間の IDL Call やファイル I/O アクセスによる無駄な検索負荷を発生させることがない．このように間接接続方式は Naming 方式とは異なり，第三者機能を用いて他のメモリ空間にオブジェクトを配置する自由度を排除することにより，携帯機の無線方式更新に最適化して手順を削減している．この結果，第 2 の課題である再起動処理の負荷を軽減できる．

4. 実装と評価

4.1 実　　装

差分分析処理と再起動処理の処理時間圧縮に対して，それぞれ差分更新管理方式と間接接続方式による効果を評価するために，**図 7** に示す構成で Configuration ミドルウェアを実装した．PentiumM プロセッサ 1 GHz，メモリ 512 MByte，Red Hat Linux 9.0 の環境上の CORBA 2.3 仕様，C++を採用した ORB に，提案方式を拡張している．ORB 機能は，Configuration ミドルウェアを利用するための Configuration ライブラリ（Config-Lib）に実装している．また Configuration Manager は，1 つの管理用のオブジェクトとして常駐させる．ここには差分更新時に一時的に生成される，差分管理 Repository が用意される．つねに構成情報を記録する Configuration Repository の更新 IDL Call 発行元管理テーブルと IOR 管理テーブルは，各オブジェクトの Config-Lib に搭載される ORB 機能が共通に利用するため，共有メモリに格納している．再構成インタフェースの実装では，オブジェクト間の関係となる IF-1 と IF-2 を，IDL Call 仕様として提供する．また Configuration ミドルウェアとオブジェクト間で行う IF-3 は，Config-Lib に実装することにより，メモリ呼び出しによる性能向上を図る．

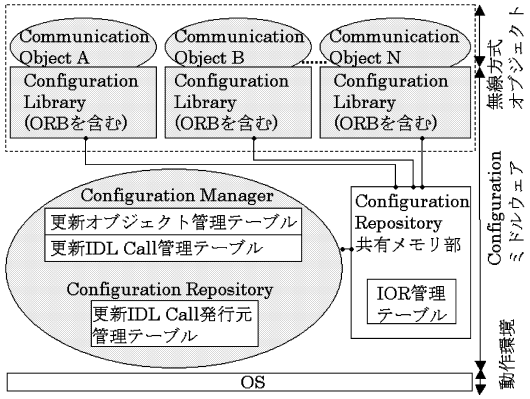


図 7 Configuration ミドルウェアの実装
Fig. 7 Implementation of the configuration middleware.

Configuration ミドルウェアは IDL Call により携帯機の動作環境に依存しない相互接続性を実現するが，さらにリソースの違いに対応するために，Dual モードと Single モードを用意する．Dual モードは運用中の無線方式を実行しながら，並行して差分となるオブジェクト構成を構築する方法である．差分更新が完了する直前まで旧無線方式の提供が可能であるとともに，再構成における障害発生が利用者に与える影響が少ない利点もある．しかし，Configuration Manager が新旧双方の構成情報を同時に保持するため，差分管理 Repository のメモリ使用量が大きく，かつ無線方式の切替えが終了するまで破棄できない．また，同時に運用中のオブジェクト構成一式が動作しているため，つねに CPU 負荷が大きい状況において更新を実行する．このため比較的リソースの大きな携帯機に適用することにより，安全性や利便性の高い無線方式の更新を実現できる．これに対して Single モードは，更新されるオブジェクトの置換えを順次行う方式である．サービス展開の中断が発生するものの，旧構成の情報はオブジェクトの停止と同時に破棄できるため，同時に利用するメモリ使用量が少なく，また他の CPU 負荷も存在しない．このため，リソースの小さな携帯機に適している．

4.2 結果と考察

表 3 は Single モードと Dual モードの双方について，オブジェクトが registIOR() による IOR の登録，および bind() による IDL Call の確立を繰り返す動作を 1,000 回実行した平均時間である．この時間をネーミングサービスを利用する方式（Naming 方式）と，間接接続方式の場合（提案方式）について測定した．この測定環境は，双方の方式とも同一の IOR 登録と，IDL Call 確立のためのロジックを用いている．このため，測定される時間の差は，両方式の実装方法の違いである，第三者機能へのアクセスの有無により生じる．この結果，間接接続方式は Naming 方式に対して，Single モードの場合で 30.2%，Dual モードの場合で 20.6%の性能向上が確認できた．

表 3 IDL Call 再起動時間の比較
Table 3 Comparison of performance times for IDL Call.

Single モード		Dual モード	
Naming 方式	提案方式	Naming 方式	提案方式
0.1628 msec	0.1136 msec	0.1630 msec	0.1294 msec

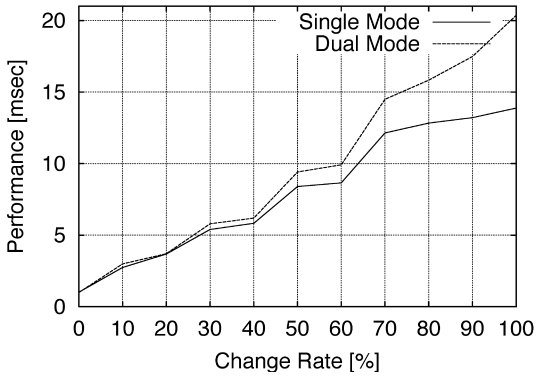


図 8 差分分析処理時間
Fig. 8 Performance time for analyzing difference between object structures.

次に Configuration ミドルウェア環境上に構築した，図 4 に示す 50 オブジェクト構成に対する差分更新の処理時間を測定した．SCA 実装実験[7] では，Waveform ソフトウェアの 3 レイヤ分にオブジェクトを適用し，それぞれを 6 個程度で構成している．このため合計 18 個のオブジェクト構成を標準と考え，これに対して今後のソフトウェア部品粒度の細分化を考慮して，約 2 倍の 50 オブジェクト構成を設定した．ここで更新するオブジェクト数を 10%ずつ増加させた場合（更新比率と呼ぶ）の処理時間を測定した．この結果，得られた差分分析処理時間を**図 8** に，この時間を含み差分となるオブジェクトを再起動する処理時間（更新時間と呼ぶ）を**図 9** に示す．また，差分分析処理時間が更新時間に占める割合を，**図 10** に計算した．これらの結果では，差分分析時間と更新時間は，更新比率に対して一定の増加を示しており，最大となる 50 オブジェクトすべてを再構成する差分分析処理時間は，Single モードで 13.9 msec，Dual モードで 20.4 msec である．また更新時間は Single モードで 1,591.8 msec，Dual モードで 2,152.6 msec である．差分分析時間の占める割合は更新比率が小さいほど大きく，最大値となる 10%のオブジェクト更新は Single モードが 1.2%，Dual モー

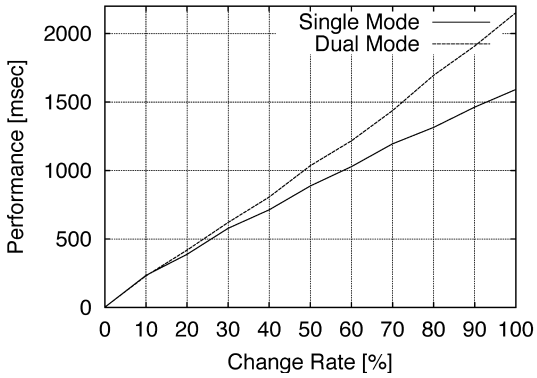


図 9 更新時間
Fig. 9 Performance time for configuration process.

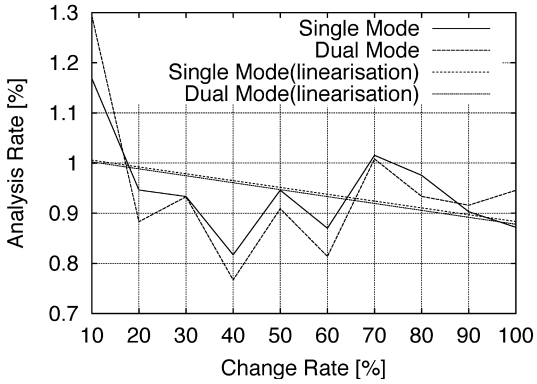


図 10 差分分析処理時間の比率
Fig. 10 Comparison between analyzing difference and configuration process.

ドが 1.3%である．

ここで表 3 の結果を用いて，最も更新時間が大きな Dual モードについて，IOR 管理テーブルへの登録と検索の処理時間を計算した．想定を更新比率 50%とした結果は，3.235 msec である．このときの差分分析処理時間の 9.414 msec と合わせた処理時間が，更新時間である 1,035.396 msec に対して占める割合は約 1.2%である．このため残りの処理時間に close() による IDL Call の解放などが含まれることを考慮しても，動作環境が管理するオブジェクトの起動や停止の処理時間と比較して，十分な性能の実現が確認できた．なお図 10 に示す割合の変動は非常に大きいため，ここでは線形近似による割合の傾向も計算している．この計算値では，オブジェクトの起動や停止数が増える更新比率の増加に対して，差分分析処理時間の比率が減少傾向にあるという結果が得られた．これは差分分析処理時間が十分に小さいとする考察と一致する．

最後に評価結果を用いて，ネットワークダウンロードを考慮した再構成時間を計算した．ここでは標準的なスループット 60%，1 つのオブジェクトサイズを 100 Kbyte，Dual モードを更新比率 50%で実行する場合を仮定した．ダウンロード伝送速度は，比較的低速な 128 kb/s，および次世代の無線通信で想定されている 100 Mb/s の 2 種類である．この結果は，128 kb/s の場合に 50 オブジェクトすべてを更新する時間が 2,693.8 msec であるのに対して，差分更新は 1,305.2 msec である．また 100 Mb/s の場合は，すべてのオブジェクト更新の時間が 2,173.6 msec であるのに対して，差分更新は 1,045.1 msec である．これは 128 kb/s の場合で約 51.5%，100 Mb/s の場合で約 51.9%の性能向上を示している．当然，伝送時間の違いによる再構成時間の絶対値には違いがあるが，50%の更新比率に対して実現される負荷は，それぞれ無視できる，1〜2%の増加にとどまる結果が得られた．

以上から，提案する Configuration ミドルウェアは，差分更新による無線方式の再構成時間の圧縮を実現することを確認した．その効果は単位ダウンロード時間が再構成時間に与える影響が大きな 128 kb/s だけではなく，理想的な 100 Mb/s の場合にも十分に発揮する．

5. 関 連 研 究

これまでに IDL Call によるオブジェクト実装時の流通性を考慮した条件下で，無線方式を再構成する方式の提案はない．本論文では将来の現実的なユビキタス環境の条件を設定し，新たな課題と解決を提案した．我々はまず，DSP/FPGA とオブジェクトの最適配置や開発手法など，SCA の実用化に向けた提案[8] に注目した．また流通性を重視した再構成時間の課題設定は，文献 9) の研究を参考にしている．ここでは更新条件と実現形態の分析から，携帯機とソフトウェアの分離を提案している．

ソフトウェア再構成の研究には，文献 10) がある．これは動的にリンクする共有ライブラリを用いて，運用中の差分更新をプロセス単位に行う方式である．この一連の研究では，差分更新のタイミング，ソフトウェア部品間の連携の考慮，再起動時間の定式化など幅広い提案がある．しかし，これはプロセス制御を用いる方法であるため，IDL Call を用いる方法と課題設定が異なる．我々は動作環境に手を加えることなく，差分分析処理と再起動処理の負荷を削減する方法を提案した．オブジェクト指向を用いた再構成の研究として JavaBeans [11] がある．シリアライズによる動作状態の保持や，コンテナによる連携クラス検索など，オブジェクト自身が構成を実行する豊富な機能が用意されている．しかし，利用状況により差分が変化する環境

では，各オブジェクト開発や実行の負荷が大きい．我々が提案する再起動手順は，Configuration ミドルウェアが一括して構成を管理することにより，オブジェクト開発の負担を軽減し，流通性を実現できる．また文献 12) ではオブジェクトの生成，消滅，移動が他へ与える影響を排除するために，ネーミングサービスや構成管理を用いた方式を提案している．しかし，これはオブジェクト間を 1 つのリンク関係に単純化した理想的なモデルであり，再起動手順や負荷の影響の考慮はない．我々は IDL Call による実装に対して，2 つの方式による再構成の負荷軽減を提案した．

6. ま と め

本論文では，IDL 仕様を用いたオブジェクト構成において，差分更新による動的な無線方式の切替えを実現する，Configuration ミドルウェアを提案した．IDL Call による構成情報に基づいて差分分析処理と再起動処理を行うため，1 つの無線方式を構成する個々のオブジェクトに対して，異なる実装モデルを適用できる．また，更新の状況に応じて異なる再起動手順をミドルウェア内に隠蔽することにより，独立性の高いオブジェクト開発が可能である．これらは，今後のユビキタス環境の展開に重要な，マルチベンダ開発に適した特徴である．

Configuration ミドルウェアが提供する差分更新管理方式は，オブジェクトと IDL Call の変更に限定した情報の分析により，タイミングと未検出の問題を回避した差分分析処理を実行する．また間接接続方式は，オブジェクトの実装アルゴリズムに影響を与えることなく，再構成によるサーバの入換えに対応して IDL Call を確立する．Configuration ミドルウェアの実装実験による性能測定では，間接接続方式による 20%〜30%の性能改善を確認した．また差分更新管理方式による差分分析処理の負荷が再構成時間に対して占める割合は，十分に無視できる約 1%である．これらの評価結果を用いてネットワークダウンロードを考慮した計算では，伝送速度が比較的低速な 128 kb/s だけでなく，理想的な 100 Mb/s の場合でも，更新比率に対して数%以下の負荷増加にとどまる．この結果から，無線方式を実現するオブジェクト構成に対して，再構成時間の短縮に対する提案方式の十分な効果を確認した．

本論文ではオブジェクト構成に注目して議論したが，次の段階では無線方式のシームレスな提供を目的に，新旧のオブジェクト間における状態保持の課題に取り組む．ここではオブジェクト構成だけでなく，DSP や FPGA などのハードウェアの状態や，音声や動画などメディアの状態など，複数の状態を取り扱う解決が必要である．また実際のユビキタス環境では，利便性のために様々なアプリケーションの連携が予想される．このため，今後はオブジェクト状態の保持の実現方法の検討，および多様なオブジェクト部品による再構成を想定し，伝送時間に対してパッケージ検索などが加わる，新たな単位ダウンロード時間を考慮した検証を行う．

謝辞 本研究の一部は，情報通信研究機構による委託研究「第 4 世代移動体通信システム実現のための研究開発」[13] に基づき実施した．

参 考 文 献

1) SDR Forum. http//www.sdrforum.org/
2) TOPPERS Project. http://www.toppers.jp/
3) Open Servcie Platform Alliance. http://www.osgi.org/
4) Object Management Group. http://www.omg.org/
5) 田野 哲，渋谷 彰：ソフトウェア無線技術，NTT DoCoMo テクニカルジャーナル，Vol.10, No.1, pp.53–60 (2002).
6) Joint Tactical Radio Systems: Software communications architecture specification JTRS-5000 SCA V3.0, JTRS Joint Program Office (2004).
7) SCA Core Framework Projects. http://www.crc.ca/en/html/crc/home/research/satcom/rars/rars
8) Pucker, L. and Holt, G.: Extending the SCA core framework inside the modem architecture of a software defined radio, *IEEE Communications Magazine* (2004).
9) 笠井裕之，菊田洋子，川崎紀宏，山崎憲一：端末間サービス移動のためのシームレスサービス環境プラットフォーム，電子情報通信学会論文誌 B, Vol.J86-B, No.8, pp.1389–1403 (2003).
10) 中島雷太，谷口秀夫：実行中プログラム部分入替え法における入替え時間の短縮，情報処理学会論文誌，Vol.41, No.6, pp.1734–1744 (2000).
11) JavaSoft, The only component architecture for Java (1997). http://java.sun.com/beans/index.html
12) Rosa, F.A. and Silva, A.R.: Component Configurer: A Design Pattern for Component-Based Configuration, *The European Conference on Pattern Languages of Programming, EuroPLoP '97*, pp.13–32 (1997).
13) http://www2.nict.go.jp/ns/s802/s1_seika.htm

(平成 17 年 2 月 3 日受付)
(平成 17 年 7 月 4 日採録)

寺島 美昭（正会員）

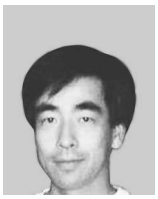
1984年埼玉大学工学部電子工学科卒業．同年三菱電機（株）入社．現在，同社情報技術総合研究所に勤務．分散処理システム構築と試験に関する研究・開発に従事．1992年本学第44回全国大会奨励賞受賞．電子情報通信学会会員．

別所 雄三（正会員）

1991年東海大学工学部電子工学科卒業．同年三菱電機（株）入社．現在，同社情報技術総合研究所に勤務．ネットワーク運用管理と分散システム構築技術の研究・開発に従事．

宮内 直人（正会員）

1987年中央大学理工学部物理学科卒業．同年三菱電機（株）入社．現在，同社情報技術総合研究所に勤務．ネットワーク運用管理と分散システム構築技術の研究・開発に従事．電子情報通信学会会員．



中川路哲男（正会員）

1983年東京大学大学院電気系工学科修士課程修了．同年三菱電機（株）入社．現在，同社情報技術総合研究所に勤務．情報ネットワーク技術と情報セキュリティ技術の研究・開発に従事．工学博士．電子情報通信学会会員．

鹿間 敏弘

1976年東京工業大学大学院総合理工学研究科電子システム専攻修了．同年三菱電機（株）入社．現在，同社情報技術総合研究所に勤務．衛星利用コンピュータネットワーク，高速リング型LAN，ATM，ネットワークセキュリティ，高速PLC等に関する研究・開発に従事．電子情報通信学会，IEEE各会員．

福岡 久雄（正会員）

1978年神戸大学大学院工学研究科電気工学専攻修士課程修了．同年三菱電機（株）入社．分散処理システムの研究開発に従事．1999年静岡大学理工学研究科博士後期課程修了．工学博士．現在，松江工業高等専門学校情報工学科教授．電子情報通信学会，IEEE各会員．

佐藤 文明（正会員）

1986年東北大学大学院工学研究科電気および通信工学専攻博士前期課程修了．同年三菱電機（株）入社．通信ソフトウェアの研究開発に従事．1995年1月より静岡大学工学部助教授．現在，静岡大学情報学部教授．工学博士．通信ソフトウェア，形式的記述，分散処理システムに関する研究に興味を持つ．電子情報通信学会，IEEE Computer Society各会員．

水野 忠則（フェロー）

1968年名古屋工業大学経営工学科卒業．同年三菱電機（株）入社．1993年静岡大学工学部情報知識工学科教授．現在，静岡大学情報学部情報科学科教授．工学博士．情報ネットワーク，プロトコル工学，モバイルコンピューティングに関する研究に従事．著書としては『プロトコル言語』（カットシステム），『分散システム入門』（近代科学社）等がある．電子情報通信学会，IEEE，ACM各会員．本学会フェロー．