情報処理学会論文誌 Vol. 51 No. 3 1019–1030 (Mar. 2010)

BPEL拡張を用いたユーザカスタマイズサービス合成機能の研究開発

山登庸次[†1] 中野雄介[†1] 宮城安敏[†1]
中村義和[†1] 大西浩行[†1]

ユーザカスタマイズサービスを容易に開発するため，BPELの拡張を用いたサービス合成機能を提案し，その有効性を示す．近年，Web2.0やSOAのキーワードが普及し，コンポーネントを組み合わせてサービスを実現することが増えている．このような連携サービスには，ユーザに応じてコンポーネントを変更するカスタマイズが必要と考えるが，BPELはユーザに応じたカスタマイズはできず，既存技術であるサービス合成技術は独自言語であるため普及が困難である．そこで，本研究では，ユーザカスタマイズサービスを容易に開発するため，標準技術であるBPELの拡張による手法を提案する．本論文は，①BPELを一部拡張し，ユーザに応じたカスタマイズ可能な拡張BPEL仕様を検討し，②拡張BPELを処理する拡張BPEL実行エンジンを，市中BPELエンジンを改変して実装し，③拡張BPELがBPELエディタを用いて容易に作成でき，拡張BPEL実行が性能上問題ないことを確認し，提案方式の有効性を示す．

Study of User Customize Sevice Composition Technology Based on BPEL Extension

Yoji Yamato,[†1] Yusuke Nakano,[†1]
Yasutoshi Miyagi,[†1] Yoshikazu Nakamura[†1]
and Hiroyuki Ohnishi[†1]

We propose the Service Composition Function based on BPEL extension for user customize services, and show the effectiveness of the proposal. Today, Web 2.0 and SOA become famous keywords, and many composition services are developed. We think user customizations are needed for these composition services. BPEL is a standard of Web Service orchestration, but BPEL cannot customize composition services for each user. The service composition technology which is our previous work needs original description, so developers cannot develope the description easily. Therefore in this paper, we propose the BPEL extension for developing user customize services easily. We study the BPEL extension, and implement a new BPEL engine which processes a BPEL extension. We evaluated descriptiveness of BPEL extension using common BPEL editors, and also measured the performance of executing BPEL extensions. The evaluation results show our method effectiveness.

1. はじめに

近年，Web2.0[1)]と呼ばれる新たなWWW利用法が注目されている．Web2.0は，集合知の側面があり，その代表例として，ユーザが複数のWebサービス（以下，WS）を組み合わせて作成するマッシュアップサービスがあげられる．また，SOA（Service Oriented Architecture）は，企業システムに多く適用されている思想で，システムを複数のサービスコンポーネントから構築し，コンポーネントを切り替えることで，変化に柔軟に対応する．SOAのコンポーネントは，主にWSを想定しており，WSの連携にはESB（Enterprise Service Bus）やBPEL（Web Services Business Process Execution Language）[2)]が用いられる．このように，コンポーネントを組み合わせて，連携サービスを開発する流れが進んできている．

マッシュアップサービスが登録されているProgrammable Web[3)]を見ると，GoogleとAmazonの利用が非常に多く，その2つと別コンポーネントを組み合わせたマッシュアップサービスが多い．たとえば，ホットペッパーで調べた店をGoogle Mapに配置するサービスと，じゃらんで調べた店をGoogle Mapに配置するサービス等があり，連携フローとしては類似だが，一部コンポーネントが異なるものが多い．これは，連携フローは類似でも，ユーザは自分に合わせたカスタマイズを望んでいるからと考える．

ユーザにカスタマイズしたサービスを合成する試みに，STONE[4)]や，サービス合成技術[5)]がある．著者らの以前の研究である文献5)は，サービスシナリオを抽象的に記述し，実行時にユーザに合わせてコンポーネントをバインドする方式を提案した．しかし，STONEも文献5)も，記述言語が独自であったため，一般の開発者には敷居が高いという問題があった．

これらの背景をふまえ，ユーザカスタマイズサービスを容易に開発するため，標準技術

†1 日本電信電話株式会社NTTネットワークサービスシステム研究所
NTT Network Service Systems Laboratories, NTT Corporation

である BPEL の拡張による連携手法を提案する．BPEL を拡張することで，数多くの市中 BPEL エディタが利用できるため，Java でマッシュアップサービスを開発するのに比べ，高い開発効率が期待できる．本論文では，①BPEL を一部拡張し，ユーザに応じたカスタマイズ可能な拡張 BPEL 仕様を検討し，②拡張 BPEL を処理する，拡張 BPEL 実行エンジンを，市中 BPEL エンジンを改変して実装し，③拡張 BPEL が BPEL エディタを用いて容易に作成でき，拡張 BPEL 実行が性能上問題ないことを確認し，提案方式の有効性を示す．

なお，連携サービスでは，主な登場人物に，コンポーネント提供者，連携サービス開発者，サービス利用者の 3 者があげられるが，本論文は，連携サービス開発者が容易に開発できることを主眼とする．一方，コンポーネント提供者には意味情報付与の負担を強いる．これは，Programmble Web[3] の登録数（2009 年 9 月 8 日に，コンポーネントは 1,440 の登録，連携サービスは 4,298 の登録）から分かるとおり，コンポーネント数よりその組合せである連携サービス数の方が大きいため，連携サービスの開発負担を減らすことが，コンポーネント提供者+連携サービス開発者のトータルの負担は減らせると考えるからである．

本論文の構成は以下のとおりである．2 章で，BPEL とサービス合成技術の課題を述べる．3 章で，BPEL 拡張を提案し，拡張 BPEL 実行エンジンを，オープンソースの Apache ODE（Orchestration Director Engine）[6] を用いて実装する．4 章で，市中 BPEL エディタを用いて作成した BPEL の一部変更により，容易に拡張 BPEL が作成できることを確認する．5 章で拡張 BPEL の実行性能を評価する．6 章で関連技術について言及する．7 章でまとめを行う．

2. BPEL とサービス合成技術の課題

既存技術として，WS を連携させる手法である，BPEL が標準化されている．しかし，BPEL は，企業間のシステムインテグレーションが目的であり，ユーザに応じてカスタマイズして連携することは困難である．具体的には，WS を連携するサービスシナリオである BPEL 文書には，利用する個々の WS の portType や operation が記述されるため，事前に BPEL エンジンに WSDL が設定された WS しか実行できず，ユーザに応じたカスタマイズ性はない．また，BPEL の switch 文で様々なパターンに応じて WS を指定することもできるが，ユーザ数が大きくなった際は，switch 文で記載するのは現実的でない．

文献 5) では，BPEL の課題を解決するため，サービスシナリオに，厳密なインタフェースを記述するのでなく，欲する WS の意味的情報を用いて抽象記述する，サービステンプレート（ST）を提案した．サービスシナリオを抽象記述することで，機能が同等な WS を，

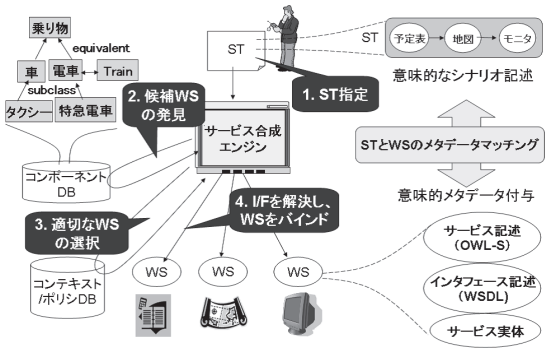


図 1　サービス合成技術の概要[5]
Fig. 1　Outline of our service composition technology [5].

1 つの ST で扱うことができるため，多くの switch 文を記述する必要がなく効率的である．WS の意味的情報には，Semantic Web Services 技術（たとえば，文献 7)）の OWL-S（Web Ontology Language for Services）[8] を用いている．OWL-S は，4 つの記述からなり，Profile 記述はどのようなサービスを提供するかを記述し，Process 記述はどのように動くかの情報である Atomic Process や入出力を記述し，Grounding 記述は実際にアクセスするための WSDL と Atomic Process のマッピングを記述し，Service 記述はその 3 つへのポインタを示す．

図 1 を参照して，サービス合成フローを説明する．ユーザは，サービスを合成する合成エンジンに，欲するサービスの ST を指定する．すると，合成エンジンは，ST に記述された意味的情報を用いて同等な WS 群を，コンポーネント DB から検索する．次に，合成エンジンは，発見された WS 群からユーザに適切な WS の選択を行う．選択には，ユーザポリシ（どの WS が優先か），ユーザコンテキスト（ユーザがどのような状況か），OWL-S Profile（WS がどのようなプロパティか）のマッチングで，点数を付け高得点の WS を選択することで実現する．利用する WS が決定された段階で，合成エンジンは，選択 WS の OWL-S Grounding を用いて WSDL を取得し，WS を実行する．

このようにすることで，ユーザポリシ，ユーザコンテキストに適切なコンポーネントを選択した連携サービス（ユーザカスタマイズサービス）を実現できる．また，連携サービス開発者は，個々のコンポーネントの WSDL を意識せず ST を記述できる．そのため，ST 作成時は存在していなかったコンポーネントも，ユーザ利用前にコンポーネント DB に登録

され，ST の意味的情報にマッチしていれば利用可能となる．

しかし，サービス合成技術は，ST という独自言語を用いていたため，連携サービスを作成する開発者にとって負担が大きいという問題があった．手動で作成する際は，ST と BPEL は同程度の時間で作成できるが，BPEL は GUI エディタで作成することが多く，その点で BPEL 作成の方が容易である．著者らは，以前に GUI でコンポーネントを接続することで ST を開発できるツールも開発した[9]．しかし，近年数多くの BPEL エディタがベンダやオープンソースから出てきているため，それらを用いることができるのが望ましいとの開発者意見が多かった．

そこで，本論文は，独自記述を用いるのでなく，標準記述である BPEL の文法に従い最小限の拡張をして，ユーザカスタマイズサービスを合成することを目的とする．これにより，市中の BPEL エディタを用いて開発できるため，連携サービス開発者の敷居を下げることができる．

3. 拡張 BPEL 検討と拡張 BPEL 実行エンジン実装

本章では，ユーザに応じたカスタマイズを実現する拡張 BPEL の検討と，それを実行する拡張 BPEL 実行エンジン実装について述べる．

3.1 拡張 BPEL 検討

サービス合成技術[5] は，OWL-S Process Model により WS を意味的に指定し，発見された同等機能の WS 群から Profile を用いて選択し，実行時に Grounding を用いて WSDL を取得し実行することで，ユーザに応じたカスタマイズを実現している．

そこで，この仕組みを，BPEL 文法に従った最小限の拡張で実現する方式を検討する．BPEL は，partnerLink でバインドする WS の portType を指定し，WSDL の operation，message をもとに，メソッド，パラメータ指定を行う．ユーザに応じたカスタマイズでは，WS は変更されるため，この 3 つの意味的指定（c，d，e）と準備（a，b）が必要である．以下，拡張 BPEL の変更部分と，記述例を示す．

a. 拡張 BPEL の名前空間指定

BPEL は拡張可能に設計されているので，BPEL の extensions 要素に，BPEL を拡張する名前空間を定義する．**図 2** (a) に記述例を示す．

b. 外部ファイルの import 宣言

外部ファイルで定義される内容のインポート宣言をする．import 要素の namespace 属性と location 属性で，外部ファイルの名前空間 URI と外部ファイルの URI を定義する．

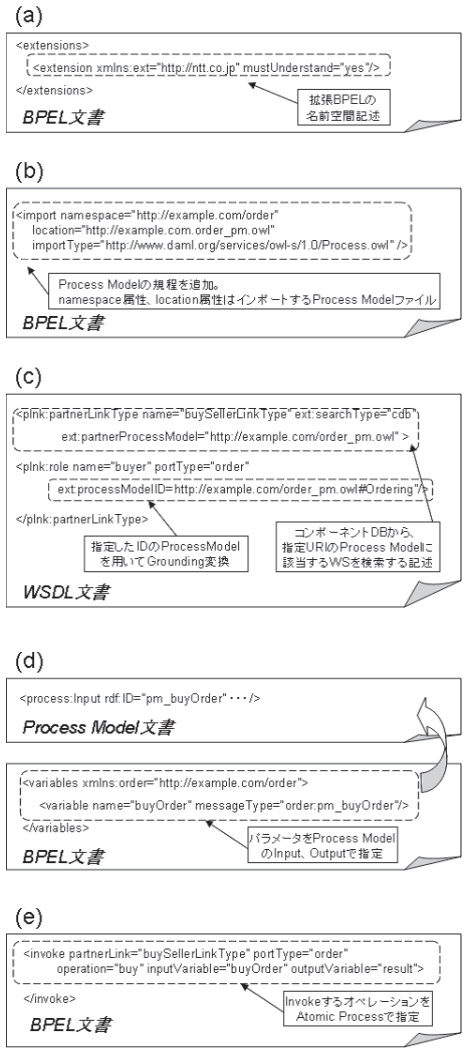


図 2　拡張 BPEL の記述例．(a) 拡張 BPEL 名前空間指定，(b) 外部ファイル import 宣言，(c) partnerLinkType 拡張，(d) variable 変更，(e) invoke 変更

Fig. 2　Example of BPEL extensiton. (a) declaration of extensiton namespace, (b) declaration of outer files “import”, (c) “partnerLinkType” extension, (d) “variable” change, (e) “invoke” change.

importType 属性は，OWL-S 1.0 Process Model とする．図 2 (b) に記述例を示す．なお，文法上は BPEL 仕様からは変更ない．

c. partnerLinkType 拡張

c-1．partnerLinkType 要素に seacrhType 属性と partnerProcessModel 属性追加．

searchType 属性では，コンポーネント DB から WS 検索する際は，「cdb」を指定する．partnerProcessModel 属性は，WS を検索する際，WS の意味的情報である Process Model の URI を指定する．

c-2．role 要素に対し，processModelID 属性追加．

processModelID 属性は，拡張 BPEL プロセスから WS に対してのメッセージ交換時に Grounding 変換を行うため，利用する Process Model の RDF（Resource Description Framework）ID を指定する．

c-1，c-2 は BPEL サービスの WSDL（連携サービスを起動するための WSDL）に対する変更である．記述例を図 2 (c) に示す．なお，固定的に WS を利用する際はどちらの追加属性記述も不要である．

d. variable の変更

variable で変数定義を行う．変数の意味的情報である ProcessModel のパラメータを指定するため，名前空間とローカル名で，ProcessModel の Input または Output を指定する．記述例を図 2 (d) に示す．なお，文法上は BPEL 仕様からは変更ない．

e. invoke の変更

invoke で指定 partnerLink の起動を行う．オペレーションの意味的情報である Process-Model の Atomic Process やパラメータの意味的情報である Input や Output を指定する．記述例を図 2 (e) に示す．なお，文法上は BPEL 仕様からは変更ない．

c で WS，d でパラメータ，e でオペレーションの意味的指定をしている．このような拡張としたポイントを説明する．拡張 BPEL 開発は，市中 BPEL エディタを用いて BPEL を開発した後，拡張部分のみ変更することを想定している．市中 BPEL エディタは BPEL サービスの WSDL については，BPEL 製品依存であるため意識しない．そのため，Process Model の指定やコンポーネント DB からの検索を指定する部分については，BPEL サービス WSDL の partnerLinkType に拡張属性定義を記述する形とした．一方，その他の部分は，BPEL の文法に合わせることで，市中 BPEL エディタを用いて BPEL ファイルを作成することができるように設計した．

各記述の作成の流れを説明する．コンポーネント提供者は，WSDL2OWL-S [10] 等のツールを用いて OWL-S を作成し，コンポーネント DB に登録する．連携サービス開発者は，BPEL エディタで BPEL を開発後，コンポーネント DB や[5] のメタデータ管理 DB に登録されている Process 記述を参照して拡張 BPEL に変更する．さらに，拡張 BPEL を実行エンジンにデプロイする際，BPEL サービスの WSDL を変更する．連携サービスを起動するクライアントは，BPEL サービスの WSDL を取得してメッセージを送信するが，WSDL 拡張要素は通常のツール（Axis 等）であれば無視されるため，クライアント開発は通常の WS クライアント開発と変わらない．

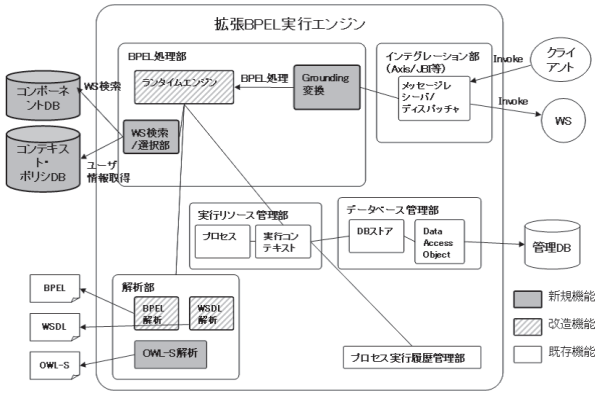


図 3　拡張 BPEL 実行エンジンの機能構成
Fig. 3　Function blocks of extended BPEL engine.

3.2　拡張 BPEL 実行エンジン実装

3.2.1　拡張 BPEL 実行エンジンの機能構成

拡張 BPEL を実行する拡張 BPEL 実行エンジンを Java 言語を用いて実装する．実装には，オープンソースの BPEL エンジンである Apache ODE 1.2 を用いた．Apache ODE を選択した理由は，Apache ライセンスでありコード改変しての頒布が可能で，BPEL に絞った機能実装のため高性能であるからである．

図 3 に，拡張 BPEL 実行エンジンの機能構成を示す．白ブロックは Apache ODE の既存機能，灰色ブロックが新規機能追加，斜線ブロックが改造機能である．Apache ODE は Web アプリケーションフレームワークである Jacob Framework により実装されており，改修が容易になっている．主な，機能追加点は，拡張 BPEL をデプロイする際の解析部と，WS 検索/選択を行う WS 検索/選択部，WS 実行時の Grounding 変換部である．

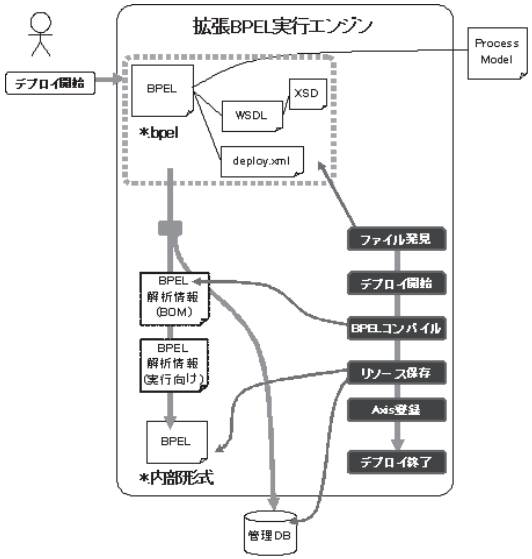


図 4　拡張 BPEL デプロイ時の処理フロー
Fig. 4　Process flow of BPEL extension deployment.

3.2.2　拡張 BPEL デプロイ処理

解析部は，拡張 BPEL がデプロイされる際に，拡張 BPEL や OWL-S Process Model 等を解析し，Apache ODE が処理可能な内部形式に変換して配置を行う．デプロイ時の動作は，**図 4** の形となる．Apache ODE では，BPEL のデプロイを開始すると，BPEL からインポートされた各種ファイルが取得され，BOM（BPEL Object Model）で，記述に沿ったオブジェクトモデルを作成する．次に，BOM をもとに，実行向けプロセス情報が作成され，内部形式に変換され配置される．ここで，デプロイされた日時等の一部情報は管理 DB に保存される．

最初の BPEL が拡張 BPEL で，Process Model をインポートしており，中間状態，最終状態も拡張 BPEL に対応する必要があるため，BPEL コンパイル部とリソース保存部で，拡張 BPEL と Process Model を解釈する処理を入れる必要がある．これらの機能は，BPEL の各記述要素ごとにクラスが存在するため，Apache ODE 解析部の partnerLink，import，variable 等クラスの改修を行うことで，実装した．また，OWL-S を解析する部分については，文献 5) のソフトウェアを流用している．

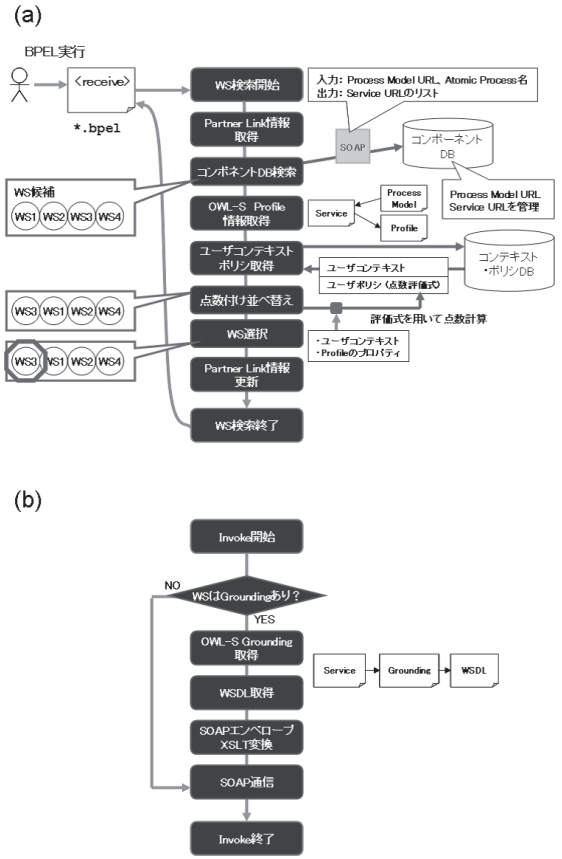


図 5　拡張 BPEL 実行時の処理フロー．(a) WS 検索・選択，(b) Grounding 変換
Fig. 5　Process flow of BPEL extension execution. (a) Web Service search and selection, (b) Grounding transformation.

3.2.3　拡張 BPEL 実行処理

WS 検索/選択部は，拡張 BPEL プロセスを起動する際に利用される．処理動作としては，**図 5** (a) の形となる．まず，クライアントからリクエストを receive で受信すると，partnerLinkType にコンポーネント DB から検索とあれば，WS 検索が開始される．WS の意味的

情報である Process Model を用いて，コンポーネント DB から WS 検索が行われ，OWL-S の Service 記述が取得され，Service 記述から Profile 情報を取得する．次に，ユーザコンテキスト/ポリシがコンテキスト・ポリシ DB から取得され，ユーザに応じた点数付けがされる．最後に，高得点の WS が選択される（検索，点数付けの詳細は文献 5) を参照）．この処理をコンポーネント DB から検索と指定された WS の数だけ行う．なお，複数検索する際は，ユーザコンテキスト/ポリシ情報は，最初の 1 回目の取得時のキャッシュを用いる．

WS 検索/選択部を実装する際は，ア）receive アクティビティクラス改修と，イ）Message Exchange Interceptors クラス利用の方法がある．イ）は，Apache ODE の拡張機能で，SOAP リクエストを受け BPEL プロセス起動前にフックして追加処理を記述できるクラスである．ア）は今後 Apache ODE がバージョンアップした際に再度変更になるため，バージョンアップに対応できるよう，イ）の手法で実装した．

Grounding 変換部は，invoke を行う際に，拡張 BPEL では意味的に指定されているオペレーションを，実際の WS のオペレーションに変換して実行する．処理動作としては，図 5 (b) の形となる．invoke 時は，partnerLink から，Grounding 変換が必要かどうかを判断し，必要な場合は，Grounding 記述を取得し，Process Model と WSDL の間のマッピングに従い，SOAP メッセージを XSLT 変換した後，SOAP 通信を行う．

Grounding 変換部を実装する際は，ウ）Apache ODE の ExternalService クラス改修と，エ）SOAP エンジンである Axis2 ハンドラの改修の方法がある．ウ）は外部サービス呼び出しを行うクラスの改修であり，エ）は実際に SOAP を実行するインテグレーション層である Axis2 の拡張機能の利用である．しかし，BPEL エンジンは，インテグレーション層は Axis や JBI（Java Business Integration）[11] 等多様な手段を選択できる必要があるため，特定の SOAP エンジンに依存するのは，今後 SOAP 以外への対応時に問題となるため，ウ）の手法で実装した．

改修，新規の Java コード数は，12 kline であり，Apache ODE の 200 kline 程度の全コード数に比べて十分小さい規模で，ユーザに応じたカスタマイズが可能な機能を実現した．実装では，拡張 BPEL かどうかで分岐する形としているため，拡張 BPEL 実行エンジンは，通常の BPEL も動作させることができる．

4. 拡張 BPEL の記述性評価

4.1 評 価 方 法

拡張 BPEL の記述性を評価するため，オープンソースの BPEL エディタ Eclipse BPEL Designer [12] を用いて BPEL プロセスを開発した後，BPEL ソースを表示し，ユーザに応じた WS 選択を行うための記述変更を行い，変更行数，変更作業時間を測定する．作成は BPEL 仕様を理解している 1 人の技術者が行った．

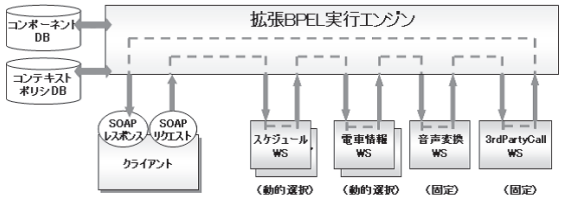


図 6　一部コンポーネントをユーザに応じて選択する拡張 BPEL の処理フロー
Fig. 6　Process flow of BPEL extension which selects two components based on user context.

なお，個々の BPEL エディタのプラグインを開発することで，変更作業も GUI で行うようにすることもできるが，本論文は，市中 BPEL エディタでユーザカスタマイズサービスの大部分の開発ができることをターゲットとしており，個々の BPEL エディタのプラグイン開発は対象外とした．

開発する BPEL および拡張 BPEL

通常の BPEL（固定 BPEL と呼ぶ）を開発し，固定 BPEL からすべての WS をユーザに応じて選択する拡張 BPEL（動的 BPEL と呼ぶ）と，固定 BPEL から一部 WS をユーザに応じて選択する拡張 BPEL（複合 BPEL と呼ぶ）に変更する．マッシュアップサービスでは，一部がユーザに応じてカスタマイズされていることが多いため，複合 BPEL がその場合に対応する．動的 BPEL は，すべて変更する際の変更比較のために作成を行う．

固定 BPEL は，4 つの WS を呼び出す BPEL であり，スケジュール WS からユーザの出張予定を取得し，出張予定に応じて電車情報を電車情報 WS から取得し，電車情報を音声変換 WS で音声ファイルに変換した後，Parlay-X [13] の 3rd パーティ Call Control WS を用いて電話で電車情報を音声通知するサービスである．

動的 BPEL はすべての WS をユーザに応じて選択し，複合 BPEL は，スケジュール WS と電車情報 WS をユーザに応じて選択する（例：outlook とサイボウズから選択等）．複合 BPEL の動作フローを**図 6** に示す．SOAP クライアントからのリクエストには，拡張 BPEL のオペレーション指定，パラメータ，ユーザ ID が含まれている．拡張 BPEL エンジンは，ユーザ ID からそのユーザのコンテキスト/ポリシ情報を取得する．また，意味的に指定されているスケジュール WS と電車情報 WS の候補をコンポーネント DB から取得し，ユー

	BPEL				WSDL			
	書換ライン数/総ライン数		文字数/比率(固定:1)		書換ライン数/総ライン数		文字数/比率(固定:1)	
動的	17		9278	1.02	9		4205	1.27
複合	9		9213	1.01	5		3777	1.14
固定		/194	9138	1		/83	3306	1

図 7　BPEL と拡張 BPEL の XML 行数比較
Fig. 7　Comparison of XML lines between BPEL and BPEL extension.

固定的に書換え必要な部分	行数	変更時間(分)
BPEL:拡張用の名前空間追加	1	1
WSDL:拡張用の名前空間追加	1	1
ユーザに応じて選択するWS数だけ書換え必要な部分	行数	変更時間(分)
BPEL:import定義部分(importTypeの指定)	1	2
BPEL:variable定義部分(message変更)	2	3
BPEL:シーケンス定義部分の記述(invoke変更)	1	1
WSDL:partnerLinkType定義部分(拡張用新属性追加)	2	4

図 8　拡張 BPEL の変更部分の行数と作業時間
Fig. 8　Lines and work time for developing each activity of BPEL extension.

ザコンテキスト/ポリシに適切な WS を選択する．最後に，選択された WS および固定指定の WS を実行し，連携サービスを提供する．

4.2 評価結果

4.2.1 記述量

記述量に関する結果を**図 7** に示す．図 7 のとおり，動的 BPEL，複合 BPEL，固定 BPEL で作成する XML 行数は変更がないが，動的 BPEL の方が新たな定義が増えているため文字数が多くなることが分かる．特に，BPEL サービスの WSDL は partnerLinkType 部分に検索方法を指定する searchType や意味的指定にあたる processModelID の記述等が増えたため，固定 BPEL に比べ文字数が約 1.2 倍に増えている．

図 8 は，固定 BPEL から変更する際に，書き換えが必要な部分と行数，平均変更時間を示している．名前空間追加等の固定的処理には固定的に 2 行の変更が必要である．ユーザに応じて選択する WS の数だけ変更が必要な部分は，BPEL の importType で利用する Process Model を指定する部分，variable や invoke で指定するパラメータやオペレーションを Process Model の Input，Output や Atomic Process 等に変更する部分，WSDL で partnerLinkType に，検索方法の searchType，検索する Process Model ID を追加する部分である．これにより，1 つの WS につき 6 行の記述変更が必要である．

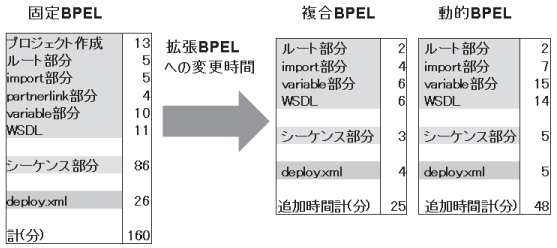


図 9　拡張 BPEL 作成時の変更作業時間
Fig. 9　Work time for developing BPEL extension.

しかし，図 7 から分かるとおり，変更部分は，固定の場合の BPEL や WSDL の記述量に比べれば小さい．BPEL は，BPEL エディタによりグラフィカルに記述することで容易にコードが作成されるため，BPEL エディタを用いることができるメリットは大きいと考える．

4.2.2 変更時間

開発者の変更時間を**図 9** に示す．図 9 のとおり，固定 BPEL 記述には，Eclipse BPEL Designer を用いて 160 分かかっており，その後複合 BPEL への変更に 25 分，動的 BPEL への変更に 48 分の作業時間がかかっている．deploy.xml は，拡張 BPEL を拡張 BPEL 実行エンジンにデプロイするための設定ファイルである．また，図 8 から，名前空間追加等の固定的な作業に約 2 分，partnerLinkType や variable 等のユーザに応じて選択する WS の数に応じて，1 つの WS あたり約 10 分の時間がかかることが分かった．

多くのマッシュアップサービスの例では，カスタマイズは 1 つ，2 つ程度の WS の変更が多いため，20 分程度の追加作業で，ユーザに応じた選択が可能である．また，固定 BPEL の記述時間に比べて，追加作業は小さく，開発者が変更に慣れることで追加作業の短縮が期待できる．BPEL サービスで，1 つの WS を変更する処理は，今回の固定 BPEL の例では，4 つの WS 利用のシーケンス作成に 86 分かかっているため，25〜30 分程度と想定される．つまり，通常の BPEL でカスタマイズサービスを行うには，ユーザ数 × 30 分程度の作業が必要である．一方提案方式は，10 分の追加作業で，ユーザに応じたカスタマイズが可能であり，効率良い開発ができるといえる．

4.2.3 連携サービス開発者以外の設定作業

1 章のとおり，本論文は連携サービス開発者が容易にユーザカスタマイズサービスを開発できることを主眼としており，前節までで容易に変更できることを確認した．本項では，参

考のため，コンポーネント提供者やサービス利用者等の設定作業を補足説明する．

コンポーネント提供者は，コンポーネント DB に OWL-S を登録する．記述量は，1 つの WS あたり OWL-S の 4 ファイルが必要なため，今回のサンプル WS では 4 ファイルで 500 行程度となる．しかし，大部分は WSDL2OWL-S [10] 等のツールにより生成するため，1 つのサンプル WS あたりの作成時間は 20 分程度で，登録時間も 5 分程度である．

サービス利用者は，ポリシ DB へ選択ポリシ設定を行う．設定作業は，サービス利用者が設定する選択ポリシの数によるが（安さ重視等），5 つ程度の選択ポリシであれば，1 サービス利用者あたり 10 行程度，5 分程度で登録ができる．また，デフォルトポリシとすることもできる．

コンテキスト DB に登録されるコンテキスト情報（プレゼンス，位置等）は，通信事業者や利用者端末等が登録するが，これは，プレゼンスサーバやロケーションサーバ等を利用することを想定している．

5. 性 能 評 価

5.1 性能測定目的

拡張 BPEL 実行エンジンの性能測定を通じて，以下の 2 つの評価を行う．

①ユーザに応じた WS 選択にともなう性能変化

固定的な WS 実行に比べ，ユーザに応じた WS 選択により，どの程度性能が劣化するかを検証する．比較には，4 章で作成した固定 BPEL，複合 BPEL，動的 BPEL を用いる．この検証により，ユーザに応じた WS 選択の実用性を評価する．

②市中 BPEL エンジンと比べた性能検証

実装した拡張 BPEL エンジンの性能を，市中 BPEL エンジンと比較する．比較対象は，母体の Apache ODE，別のオープンソース製品，市販ベンダ製品である．この検証により，提案方式が十分な性能であるかを確認する．

5.2 性能測定条件

性能測定では，拡張 BPEL 実行エンジンまたは市中 BPEL エンジンに一定の負荷をかけた際の，CPU 使用率，応答時間を測定した．測定は 3 回行われ，グラフは 3 回の平均値である．なお，市販製品の性能は今回の条件での性能であり，条件により大きく変わる可能性があることは注意されたい．性能測定条件を以下に示す．

測定パターン：

・スループット測定（1 時間あたりのリクエスト数を 1,000〜2,500,000 と変化）

・複数同時実行（同時処理スレッド数を 1〜350 と変化）

測定項目：

・CPU 使用率（Linux の top コマンドで取得）

・応答時間（リクエスタ（soapUI）のログで取得）

測定設定：

・拡張 BPEL 実行エンジンは，4 章の固定 BPEL，複合 BPEL，動的 BPEL を実行する．市中 BPEL エンジンは，固定 BPEL を実行する．

・クライアントは，100 種類のユーザ ID を順番に選択してリクエスト．

・ユーザに応じて選択される WS 候補は，コンポーネント DB から各 WS ごとに 10 個発見．

・拡張 BPEL 実行エンジンは，発見した WS の，OWL-S Profile の値段プロパティとユーザポリシをマッチングし，最も値段が安い WS を選択．

・利用する 4 つの WS は，メッセージを受け取ると 100 ms スリープした後に固定のメッセージを返答する，document/literal 型の WS.

測定環境：

測定環境を**図 10** に示す．測定環境は，拡張 BPEL 実行エンジン，コンポーネント DB およびユーザコンテキスト・ポリシ DB，クライアント，コンポーネントサーバを同一 LAN で接続した．WS は 4 つとも同一のコンポーネントサーバ上で動作している．また，市中 BPEL エンジン測定の際は，拡張 BPEL 実行エンジンを外し，市中 BPEL エンジンを接続した．使用したハードウェア，ソフトウェアは図に記載のとおりである．拡張 BPEL 実行エンジンに使われている，Derby は BPEL 処理状態を保持するための管理 DB であり，コンポーネント DB とは異なる．

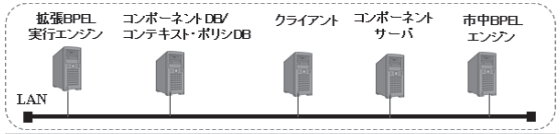
処理フロー：

測定時の処理フローは，4 章の図 6 のとおりである．リクエスタは soapUI を用いて，SOAP メッセージで，利用する拡張 BPEL のオペレーションと入力パラメータ，ユーザ ID 情報を送信する．拡張 BPEL 実行エンジンは，ユーザ ID 情報を用いて認証し，ユーザコンテキスト情報を取得し，それを用いて WS の選択を行い，4 つの WS を実行して，レスポンスをリクエスタに返す．

5.3 評 価 結 果

5.3.1 ①ユーザに応じた WS 選択にともなう性能変化

図 11 (a) は，横軸に拡張 BPEL 実行エンジンの CPU 使用率，縦軸スループット（サン



	ハードウェア	OS	ソフトウェア
拡張BPEL実行エンジン	Dell PowerEdge1950 CPU: Xeon5160 3.0GHz×2 RAM: 4.0GB NIC: Broadcom Gigabit Ethernet	RedHat Enterprise Linux ES4	JDK 1.6.0_07 Tomcat 6.0.18 Axis 1.4 Derby 10.4.1.3 拡張BPEL実行エンジン
コンポーネントDB/コンテキスト・ポリシDB	Dell PowerEdge1950 CPU: Xeon5420 2.5GHz×2 RAM: 4.0GB NIC: Broadcom Gigabit Ethernet	RedHat Enterprise Linux ES4	PostgreSQL 8.3.3
クライアント	Dell PowerEdge1950 CPU: Xeon5160 3.0GHz×2 RAM: 4.0GB NIC: Broadcom Gigabit Ethernet	RedHat Enterprise Linux ES4	JDK 1.6.0_07 soapUI 1.7.6
コンポーネントサーバ	Dell PowerEdge1950 CPU: Xeon5160 3.0GHz×2 RAM: 4.0GB NIC: Broadcom Gigabit Ethernet	RedHat Enterprise Linux ES4	JDK 1.6.0_07 Tomcat 6.0.18 サンプルWS×4
市中BPELエンジン	Dell PowerEdge1950 CPU: Xeon5160 3.0GHz×2 RAM: 4.0GB NIC: Broadcom Gigabit Ethernet	RedHat Enterprise Linux ES4	市中BPELエンジン - Apache ODE - オープンソース製品 - 市販製品

図 10　性能測定環境
Fig. 10　Performance measurement environment.

プルシナリオの処理数/h）をとったグラフ，図 11 (b) は，横軸に拡張 BPEL 実行エンジンの同時処理スレッド数，縦軸平均応答時間をとったグラフである．

図 11 (a) より，固定 BPEL に比べて，複合 BPEL，動的 BPEL でスループットが低下していることが分かる．動的 BPEL の場合で，最大スループットは固定 BPEL のほぼ 2/3 となっている．複合 BPEL，動的 BPEL でスループットが低下する理由としては，ユーザに応じた WS 選択関連の処理として，WS の検索回数，WS の選択回数，Grounding によるインタフェース解決回数が，ユーザに応じた WS 選択数に応じて増えるからである．

図 11 (b) より，固定 BPEL に比べて，複合 BPEL，動的 BPEL で平均応答時間が増加していることが分かる．固定 BPEL は 250 同時スレッド時も，520 ms 程度とオーバヘッドは 120 ms 程度だが，動的 BPEL は 100 スレッドを超えたあたりから増大し，200 スレッドで 600 ms 程度と遅延が大きくなることが分かる．拡張 BPEL 実行エンジンの新規機能追加した箇所のログ出力から，複合 BPEL で 90 スレッド時に，WS 検索/選択処理に 28 ms かかっていることが分かっており，遅延の大きな割合を，WS 検索と選択が占めていることが分かる．

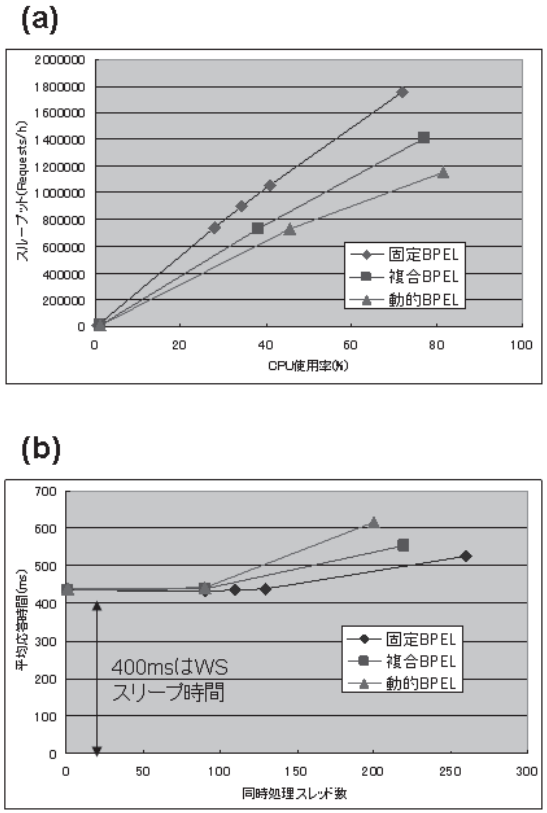


図 11　(a) 拡張 BPEL 実行スループットの CPU 使用率変化，(b) 拡張 BPEL 平均応答時間の同時処理スレッド数変化
Fig. 11　(a) Throughput result of BPEL extension execution, (b) Average response time of BPEL extension execution.

性能劣化度合いについて考察する．文献 14) は，リクエストするユーザに応じて BPEL を動的に作成，デプロイする手法について検討しているが，その場合は，通常の BPEL 実行に比べて，スループットは 1/10 程度に落ちている．一方，提案方式は，デプロイされる拡張 BPEL は 1 つで，リクエストを receive した際に，拡張 BPEL の場合は，WS 検索，WS 選択を行う処理に分岐する形としている．そのため，BPEL をデプロイする処理が不要

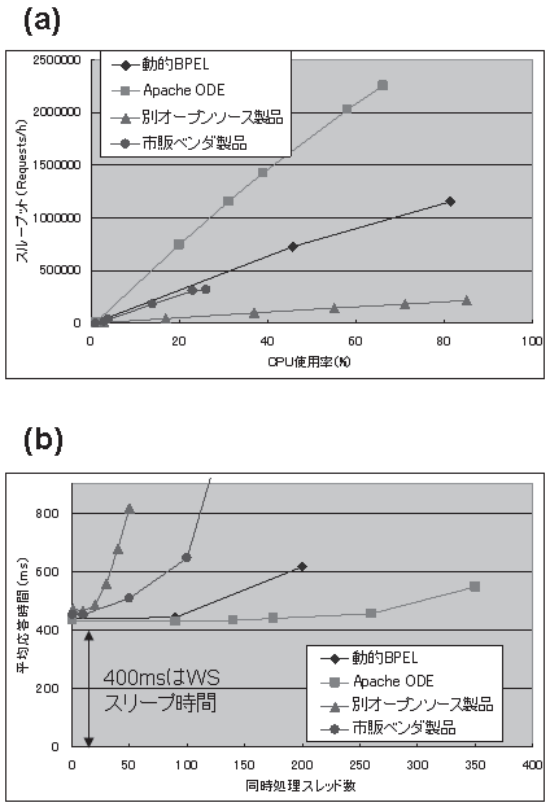


図 12　(a) BPEL 実行スループットの CPU 使用率変化，(b) BPEL 平均応答時間の同時処理スレッド数変化
Fig. 12　(a) Throughput result of BPEL execution, (b) Average response time of BPEL execution.

となるため，スループットの低下も 2/3 まで抑えており，性能劣化が少ない方式と考える．

5.3.2　②市中 BPEL エンジンと比べた性能検証

図 12 (a) は，横軸に拡張 BPEL 実行エンジンまたは BPEL エンジンの CPU 使用率，縦軸スループットをとったグラフ，図 12 (b) は，横軸に拡張 BPEL 実行エンジンまたは BPEL エンジンの同時処理スレッド数，縦軸平均応答時間をとったグラフである．図 12 (a) より，拡張 BPEL 実行エンジンで動的 BPEL の場合は，Apache ODE に比べると同 CPU 使用率でのスループットが 1/2 程度であるが，別オープンソース製品に比べると 6 倍程度，市販ベンダ製品の 1.1 倍程度であることが分かる．また，図 12 (b) より，平均応答時間は，拡張 BPEL 実行エンジンで動的 BPEL の場合は，同時処理スレッド数が 100 を超えると増大するため，Apache ODE に比べると遅延が大きいが，別オープンソース製品，市販ベンダ製品に比べると小さいことが分かる．

この結果の理由としては，まず，母体として利用している Apache ODE が，BPEL 処理機能に絞って実装しているため，性能が高いことがあげられる．なお，図 11 (a) と図 12 (a) から固定 BPEL でも Apache ODE の 8 割強しかスループットが出ていないが，これは，固定 BPEL でもユーザ認証や分岐判断等の処理が入るためである．次に，①で考察したように，今回の実装方式が，文献 14) のように毎回 BPEL をデプロイする方式に比べるとオーバヘッドが少なかったからと考える．

このように，動的 BPEL でも 120 万リクエスト/h 処理でき，市販製品と同等以上の性能を示すため，WS を連携するエンジンとして十分実用的であるといえる．

6. 関 連 技 術

ユーザに応じたサービス連携手法として，STONE [4]，Ninja [15] 等があげられる．STONE，Ninja は，それぞれサービスグラフ，Path という，コンポーネントの連携記述に従ってサービスを合成する．STONE，Ninja とも，独自記述言語であることから普及が困難である．本研究は，BPEL を一部拡張しており，BPEL に慣れた開発者であれば，BPEL エディタで開発後，10 行程度変更するだけでよいので，効率的に開発ができる．

WS 連携の仲介バスとして，ESB がある．ESB は，JBI 標準で一部機能の標準化が進められている．ESB の機能の 1 つとして，SOAP メッセージ内のパラメータ値に応じて，呼び出す WS を切り換える機能がある．しかし，この切換えは，呼び出される WS は事前に switch 文で条件を設定されていることが前提である．そのため，多くのユーザに応じて switch 文を記述するのは現実的でない．

本研究と同様，Semantic Web 技術を用いてサービス合成する試みに，TaskComputing [16] や Ubiquitous Service Finder [17] がある．TaskComputing では，サービスを OWL-S で記述する．TaskComputing は，ユーザがいるエリアで利用可能なサービス群の中で，あるサービスの出力が次のサービスの入力になる 2 つのサービスの組合せ候補が提示され，手動で組合せを選択し合成サービスを実行する．TaskComputing と本研究の違いは，TaskComputing が個々のサービスに注目してボトムアップ的に合成するのに対し，本研究は欲する連携サービスを拡張 BPEL としてユーザが指定し，それに合う WS を発見し合成する形で，トップ

ダウン的な合成を目指している．TaskComputing ではエリア内サービス数が増えた場合に組合せ候補が増えすぎユーザの選択が困難，分岐やループ等の複雑なサービス実現が困難，という課題がある．本研究は，拡張 BPEL で指定した意味的に同等な WS から選択すればよいため選択が容易であり，また，BPEL 拡張であるため複雑なサービスの実現も可能である．

本論文は連携サービス開発を容易にするが，コンポーネントへの意味的情報（メタデータ）付与の容易化は解決していない．しかし，市中や標準動向から，メタデータ付与の流れは進んでいくと考える．WS にメタデータを付与する SAWSDL（Semantic Annotations for WSDL）が W3C で標準化が進んでいる．また，OWL-S 作成を支援する，WSDL2OWL-S [10] というツールがある．また，旅行業界のメタデータ標準である travelXML や，文書関係のメタデータである Dublin Core 等，メタデータ標準を定める動きも出てきている．そのため，カスタマイズサービスのニーズが高まれば，効率的にそれを実現するための，メタデータが WS に付与されていくと考える．

著者らの以前の研究である，文献 5)，14) と本論文の比較を行う．文献 5)，14) とも ST を用いたサービス合成であるが，文献 14) はユーザリクエストに応じて ST から BPEL を生成した後，市中の BPEL エンジンにデプロイして実行させている．これらのサービス合成エンジンを，性能，標準記述，製品依存で比較する．

文献 5)：性能中，独自記述，全機能内製

文献 14)：性能低，独自記述，他 BPEL エンジンに変更可

本論文：性能高，標準記述，Apache ODE に依存

文献 5) は，ほとんどの機能は本論文と同等であるため，現在は適用領域はない．

文献 14) は，個々の BPEL エンジンの BPEL 処理以外の特徴機能（種々の暗号方式や認証方式対応等）を利用可能なユーザカスタマイズエンジンを狙いとしている．試作では ActiveBPEL を用いたが，他の BPEL エンジンにも変更可能であり，市中 BPEL エンジンの特徴機能を用いるサービスが適用領域である．

本論文は，文献 5)，14) の独自記述を解決するための検討であり，BPEL 文法に準拠した拡張 BPEL を定義しており，BPEL エディタを用いた開発が可能である．Apache ODE への依存が懸念点であるため，改修時は他の BPEL エンジンでも流用できる機能が多いように設計した．市中 BPEL エンジンの特徴機能が不要なカスタマイズサービスが適用領域である．

7. ま と め

本論文では，ユーザカスタマイズサービスを容易に開発することを目的として，コンポーネントを意味的に指定する BPEL 拡張を提案した．さらに，拡張 BPEL を解釈実行する拡張 BPEL 実行エンジンを，Apache ODE を用いて実装した．提案手法の評価として，記述性と性能評価を行った．記述性に関しては，市中の BPEL エディタを用いてサンプル BPEL を作成後，10 行程度の変更，20 分程度の作業で容易に拡張 BPEL を作成できることを確認した．性能に関しては，コンポーネントを固定実行するのに比べユーザに応じた選択時は，スループットが 2/3 程度になるが，サンプル拡張 BPEL を 120 万リクエスト/h 処理できることを確認した．これは，市販 BPEL エンジンに比べ十分高い性能で，ユーザに応じたコンポーネント選択が方式として性能に問題ないことを示した．

今後は，Apache ODE のバージョンアップや他 BPEL エンジンへのポーティング等，母体となる BPEL エンジンが変更された際も，早期に対応できるための実装方式を検討する．また，通信キャリアが持つ，位置，プレゼンス，加入者情報等を用いた，カスタマイズサービスの実現のため，SDP（Service Delivery Platform）[18] への適用を検討していく．

参 考 文 献

1) What is Web 2.0 web site. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
2) Business Process Execution Language web site. http://www.oasis-open.org/committees/download.php/18714/wsbpel-specification-draft-May17.htm
3) Programmable Web web site. http://www.programmableweb.com/
4) 南　正輝，森川博之，青山友紀：動的でアドホックなネットワークサービスフレームワークの検討，マルチメディア，分散，協調とモバイルシンポジウム（DICOMO2000）(June 2000).
5) 山登庸次，中辻　真，須永　宏：ユビキタス環境にて動的にサービス実現するためのサービス合成技術，情報処理学会論文誌，Vol.48, No.2, pp.562–577 (2007).
6) Apache ODE web site. http://ode.apache.org/
7) Paolucci, M. and Sycara, K.: Autonomous Semantic Web Services, *IEEE Internet Computing*, pp34–41 (Oct. 2003).
8) OWL-S web site. http://www.daml.org/services
9) Moriya, T., Ohnishi, H., Yoshida, M. and Hirano, M.: A Support System for Designing Ubiquitous Service Composition Scenarios, *IEEE International Conference on Communications 2007* (*ICC2007*), pp.1594–1599 (June 2007).

10) WSDL2OWL-S web site. http://www.daml.ri.cmu.edu/wsdl2owls/
11) JSR (Java Specification Request) 312: Java Business Integration 2.0 web site. http://jcp.org/en/jsr/detail?id=312
12) Eclipse BPEL Designer web site. http://www.eclipse.org/bpel/
13) Parlay-X web site. http://www.parlay.org/en/specifications/pxws.asp
14) 山登庸次，中野雄介，須永　宏：BPEL エンジンを用いたユーザカスタマイズサービス合成・切換技術の研究開発，電子情報通信学会論文誌，Vol.J91-B, No.11, pp.1428–1439 (2008).
15) Gribble, S., Welsh, M., Behren, R., Brewer, E., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A., Katz, R., Mao, Z., Ross, S. and Zhao, B.: The Ninja Architecture for Robust Internet-Scale Systems and Services, *Special Issue of Computer Networks on Pervasive Computing* (2000).
16) Masuoka, R., Parsia, B., Labrou, Y. and Sirin, E.: Ontology-Enabled Pervasive Computing Applications, *IEEE Inteligent Systems*, Vol.18, No.5, pp.68–72 (2003).
17) Kawamura, T., Ueno, K., Nagano, S., Hasegawa, T. and Ohsuga, A.: Ubiquitous Service Finder - Discovery of Services semantically derived from metadata in Ubiquitous Computing, *4th International Semantic Web Conference* (*ISWC 2005*), pp.902–915 (Nov. 2005).
18) Ohnishi, H., Yamato, Y., Kaneko, M., Moriya, T., Hirano, M. and Sunaga, H.: Service Delivery Platform for Telecom-Enterprise-Internet Combined Services, *IEEE Global Telecommunications Conference 2007* (*GLOBECOM2007*), pp.108–112 (Nov. 2007).

(平成 21 年 5 月 21 日受付)
(平成 21 年 12 月 17 日採録)

山登　庸次

2000 年東京大学理学部卒業．2002 年同大学大学院理学系研究科修士課程修了．2009 年同大学院総合文化研究科にて博士（学術）取得．2002 年日本電信電話株式会社入社．同社ネットワークサービスシステム研究所にて，ユビキタスコンピューティング，Service Delivery Platform 研究開発に従事．現在，米 Verio 社にて SaaS，ホスティングサービス開発に従事．2005〜2007 年，電子情報通信学会次世代ネットワークソフトウェア時限研究専門委員会幹事補佐．2009 年電子情報通信学会学術奨励賞受賞．電子情報通信学会会員．

中野　雄介（正会員）

2005 年和歌山大学大学院システム工学研究科修了．同年日本電信電話株式会社入社．同社ネットワークサービスシステム研究所にて Web マイニング，ユビキタスコンピューティング，グループウェア研究に従事．2004 年情報処理学会第 66 回全国大会学生奨励賞受賞．平成 19 年度電子情報通信学会学術奨励賞受賞．電子情報通信学会会員．

宮城　安敏（正会員）

2005 年立命館大学理工学部卒業．2007 年奈良先端科学技術大学院大学情報科学研究科修士課程修了．2007 年日本電信電話株式会社入社．同社ネットワークサービスシステム研究所にて，Service Delivery Platform，サービス開発手法の研究開発に従事．

中村　義和

2000 年九州大学工学部卒業．2003 年同大学大学院工学府材料物性工学専攻修了．2003 年日本電信電話（株）入社．同社ネットワークサービスシステム研究所にて，移動体向けセキュリティ技術，センサーネットワーク構築技術，Service Delivery Platform の研究開発に従事．電子情報通信学会会員．

大西　浩行

現在，東日本電信電話（株）ネットワーク事業推進本部担当課長．1996 年早稲田大学にて機械工学学士，1998 年早稲田大学大学院にて機械工学修士取得．1998 年に日本電信電話（株）入社．同社ネットワークサービスシステム研究所にて次世代モバイル IP ネットワークアーキテクチャ，サービスアプリケーション開発プラットフォームの研究開発に従事．1999 年に電子情報通信学会交換システム研究賞，2005 年に電子情報通信学会研究奨励賞．電子情報通信学会会員．