社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－SE－148 (2)
2005／5／31

BPELを用いた複合 Web サービスのモデルベース開発支援

福永 遂重　高木 松雄　青山 幹雄

南山大学 数理情報学部 情報通信学科

Web サービスは Web 上の自律したサービスを連携する基盤技術として注目されている. Web サービスの基盤上で, 複数のサービスを組み合わせて, より高度な機能を提供する複合 Web サービスが実現できる. しかし, 機能に加えて非機能的要求も満たす複合 Web サービスの提供が要求されている. 本論文は, BPEL4WS を用いて, 共通のサービスモデルに基づき, 複数の Web サービスを動的に連携し, 非機能的要求も満たす複合 Web サービスの構築方法とその環境支援を提案する. さらに, 複合 Web サービス開発支援環境のプロトタイプを開発し, 例題を用いて評価した結果を示す. サービスの選択だけで BPEL プロセスと使用するサービスの WSDL ファイルの自動生成を実現した. また, DOM による実装との比較により XSLT を用いた BPEL プロセス自動生成の有用性を示す.

A Model-Based Development Support System for Composite Web Services with BPEL

Yukishige Fukunaga　Matsuo Takagi　Mikio Aoyama

Department of Information and Telecommunication Engineering,
Faculty of Mathematical Sciences and Information Engineering,　Nanzan University

Web services are attracting attentions for collaborating distributed services on the Web. On the Web service platform, multiple Web services are composed as a composite Web services for providing value-added services. For such services, non-functional requirements, such as performance and quality, need to be fulfilled across multiple services. This article proposes a model-based methodology to dynamically compose Web services, with BPEL4WS, of required non-functional requirements, and its support environment. Applying a prototype of the support environment to examples revealed effectiveness of the automated generation of BPEL process and WSDL interfaces through the XSLT transformation from common interface model.

1. はじめに

Web 上で分散されたサービスを連携する技術として Web サービスが注目されている[3, 6, 8, 9]. Web サービスでは Web 上に公開されている複数のサービスを組み合わせることにより, 1つの新しい機能を提供する複合 Web サービス(Composite Web Services)[1]を実現できる. しかし, 一般に, 同一の機能をもったサービスであっても品質, 性能などの非機能的特性が異なる. 従って, 機能だけではなく, 非機能的要求も満たす複合 Web サービスを提供することが要求される.

本論文は, BPEL4WS (Business Process Execution Language for Web Services: 以下 BPEL と略記)[2, 7]を用いて, 複数の同一の機能の Web サービスを動的に連携させて, リクエスタの非機能的要求も満たす複合 Web サービスの構築方法とその環境支援を提案し, 例題を用いて評価した結果を示す[5].

2. 複合 Web サービス開発へのアプローチ

2.1. 問題点と解決案

同一の機能であっても, 非機能的特性が異なる複数の Web サービスを複合 Web サービスとして動的に連携するための問題点として, サービス間のインタフェースの不整合性が挙げられる.

本稿では, サービス間のインタフェースの整合をとるために共通のインタフェースモデルを定義し, それに基づき WSDL で同一の機能を対応づけ, インタフェースの整合をとる方法を提案する.

また, BPEL による Web サービスの連携は, あらかじめ BPEL プロセスに記述しておいたサービスしか利用できない. 従って, 新しくサービスを追加する場合は, そのサービスのインタフェースを BPEL プロセスとして人手で記述しなければならない問題がある. 使用するサービスのインタフェース情報の理解と記述に人手を介することとなる.

本稿ではリポジトリに定義したサービスの共通のインタフェースモデルに基づき, 個々のサービスのインタフ

ェースへマッピングを行うことでサービス毎の WSDL から BPEL プロセスを自動生成する.

さらに, 非機能的要求が異なる複数のサービスを組み合わせることが可能となるので, 各サービスを選択的に実行できる複合 Web サービスの構築が可能となる.

2.2. モデルベース開発のプロセス

本稿で提案するモデルベース開発のプロセスを図 1 に示す.

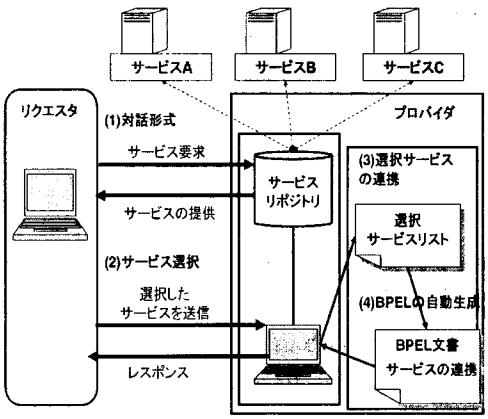

図 1 BPEL プロセス自動生成までの流れ

Web サービスを連携する場合, 機能の類似なサービスを選択および組み合わせる必要がある.

本論文では, 機能が類似するサービスを提供するために, BPEL を用いてサービスを動的に連携する方法を提案する.

この開発方法は図 1 に示す以下の4つのプロセスからなる.

(1) 対話形式によるサービス要求
(2) 類似機能のサービス選択
(3) 選択サービスの自動連携
(4) BPEL プロセスの自動生成

(1) 対話形式によるサービス要求

リクエスタは連携させる Web サービスをプロバイダと対話形式で要求する. プロバイダは提供可能なサービスのリストをリクエスタに送信する.

(2) 類似機能のサービスの選択

リクエスタはプロバイダが提供する類似サービスの中から利用したいサービスを要求する. 機能の類似なサービスを組み合わせることによって一度で複数のサービスの結果を得ることが可能になる.

機能の類似なサービスの例として, 異なる語彙を持つ辞書サービス, 異なる検索方法による検索サービスなどが挙げられる. これらのサービスは, 検索という同一の機能を提供する. しかし, サービス毎に検索時間などの性能, 得られる情報の質と量, 分野など非機能的要求が異なる. リクエスタが利用したいサービスの非機能的要求は, 状況によって変化する. リクエスタが一般的な辞書で単語を検索したい要求であれば IT 分野の辞書サービスを連携する必要がなく一般的な辞書の Web サービスだけを連携させればよい.

従って, プロバイダはリクエスタによるサービス要求を受信した時に, 非機能的要求に応じたサービスの連携を行う必要がある.

(3) 選択されたサービスの自動連携

リクエスタが選択したサービスだけを自動的に連携する. これを実現するため選択されたサービスのインタフェース情報を取得し, 選択サービスリストを生成しインタフェースの整合をとり, BPEL プロセスを生成する. リクエスタから要求された選択サービスのインタフェース情報を集めたものを選択サービスリストと定義する.

(4) サービスを利用する BPEL プロセスの自動生成

類似機能のサービスを BPEL の並行実行フローを用いて組み合わせる. 本論文では, 並行実行のフローを適用した BPEL プロセスの XSLT スタイルシートを作成する方法を提案する.

BPEL プロセスは XML で記述されているので, XSLT スタイルシートを用いて BPEL プロセスを自動生成する. しかし新しくサービスを追加するときに追加したいサービスをあらかじめ BPEL プロセスに記述しておかなければならない問題がある. このため, 必要なサービスを実行時に組み合わせて動的連携を行うために BPEL プロセスもサービスの選択時に動的に作成することとした.

3. 複合 Web サービスの開発方法

3.1. BPEL プロセス自動生成のプロセス

プロバイダがリクエスタの使用するサービス要求を受信してから BPEL プロセスを自動生成するまでのプロセスを図 2 に示す. この BPEL プロセス自動生成のための課題とその解決方法を示す.

本稿では, 同一機能のサービスを組み合わせて BPEL プロセスを自動生成する方法を提案した. 従って, 同一機能のサービスを組み合わせるためにサービス毎に機能の同一性の対応付けを定義しインタフェースの整合をとる. サービス間でモデルを定義し各 WSDL のインタフェースにモデルとの対応付けを定義するメタ情報を属性として付加する. このモデルより個々のインタフェースへのマッピングを行うためのマッピングオブジェクトを生成する. BPEL プロセスへの変換元となる選択サービスリストを XML インスタンスとして生成する. この XML インスタンスと XSLT スタイルシートより BPEL プロセスを生成する.

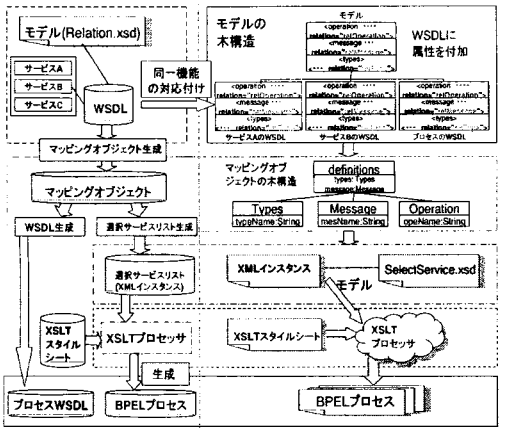

図 2 BPEL プロセスの自動生成までのプロセス

3.2. 属性による類似サービスのモデル定義

機能的に類似したサービス間のインタフェースの整合をとるメタ情報として WSDL にインタフェース定義の type, message, operation に新たに属性を付加する. 属性を付加することでサービスの機能的に同一なインタフェースを抽出することができ, 連携が可能となる. また WSDL に準拠しているので, 全ての WSDL によるインタフェース定義に適応可能である. 提供するサービス毎に同一の機能に対応して, type 毎に同一の属性, message 毎に同一の属性, operation 毎に同一の属性を付加する. 一方, BPEL プロセス側の WSDL にも属性を付加する.

従って, 各サービスの機能ごとに対応付けることができる. 付加する属性は XML スキーマで定義する[11]. 本稿では Relation.xsd で定義した. Operation に対応する属性名を"relOperation", message に対応する属性名を"relMessage", part に対応する属性名を"relPart", type に対応する属性名を"relType"とする.

例として, サービス A, サービス B, BPEL プロセスの対応関係を図 3 に示す.

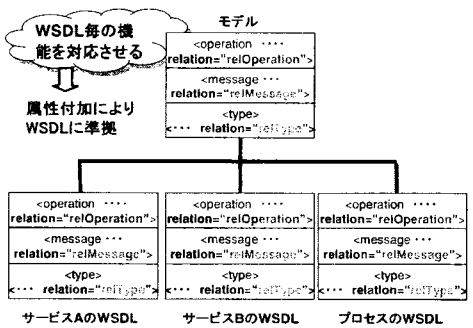

図 3 属性付加による WSDL の対応関係

3.3. インタフェースのマッピング方法

選択サービスの各 WSDL の構造は, 同じ木構造であるので同型な木構造のマッピングで実装する. 一つの方法として, ある木構造のノードから子ノードを走査して message の name 属性を探索するために DOM 木を適用できる[8]. しかし, 実際に必要なのは DOM 木ではなく, 各要素クラスのオブジェクトである. ここでは, message クラスや, types クラスのオブジェクトにあたる.

図 4 に, モデルで類似機能を対応させたインタフェースからマッピングしたクラス図を表す. 構造も WSDL と同じ木構造で表現する. WSDL の要素<types>, <message>, <operation>で類似機能を対応させた属性を読み込みマッピングしマッピングオブジェクトを生成する. 各要素に対して選択サービス毎のマッピングオブジェクトを保持する.

このようにクラスのインスタンス変数にそのクラスの保持する値が自動的に設定できれば, 選択サービスリスト作成の際に DOM 木を走査する処理を省略できる.

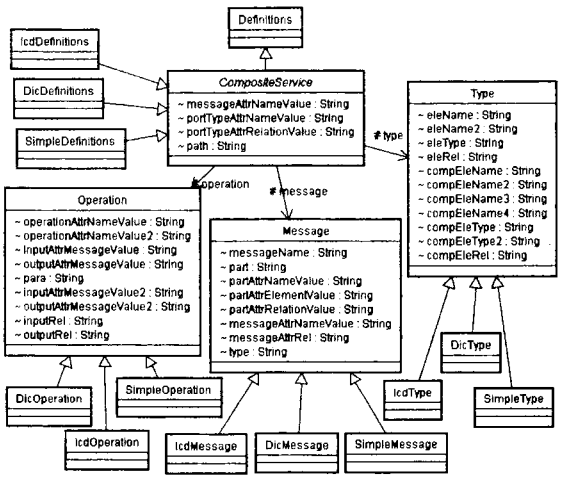

図 4 マッピングしたクラス図

図 5 に木構造へのマッピングの構造を示す.

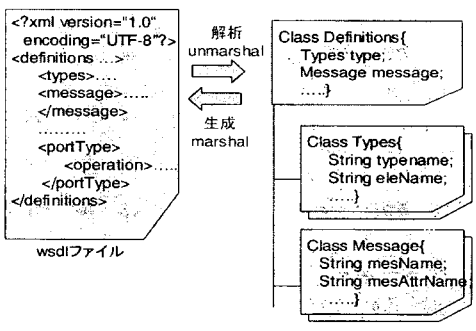

図 5 木構造へのマッピング

3.4. 選択サービスリストの生成

BPEL プロセスを生成するためのサービスリストを生成するために利用するサービスの WSDL からインタフェース定義を抽出する. このリストはインタフェース間の整合をとるために属性によって対応付けしたリストである.

XSLT の変換元になる XML インスタンスにあたる.

3.4.1. 選択サービスリストのモデルの設計

選択サービスリストの XML インスタンスを生成するために, XML インスタンスのモデルを設計する. 本稿では SelectService.xsd で定義した. BPEL プロセスに組み込む Web サービスの要素を木構造のクラス図として図 6 に示す.

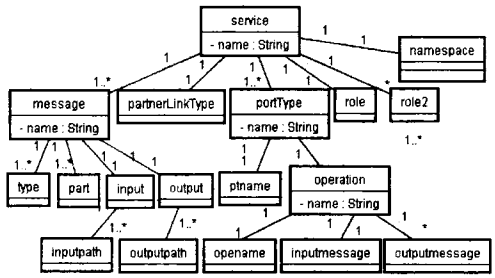

図 6 選択サービスリストのモデル

3.4.2. 選択サービスリスト要素による XSLT スタイルシート作成

BPEL プロセスのタグを生成する XSLT スタイルシートを呼び出す current ノードを決めるために, 選択サービスリストから XML インスタンスの要素でマッチングさせる XSLT スタイルシートを作成する. 図 7 に XSLT スタイルシートのデータ構造を示す.

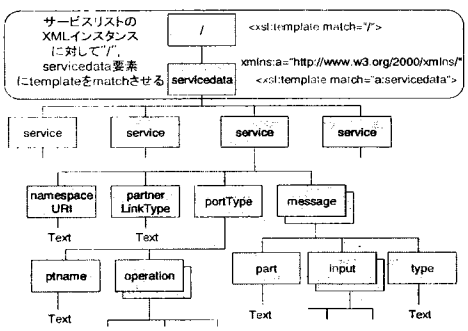

図 7 選択サービスリストの XSLT スタイルシート構造

図 7 に示すように選択サービスリストは木構造になる. まず選択サービスリストの全ての要素のデータにアクセスできる currect ノードの位置を設定する. XML インスタンスの root ノードが "/"であるので, XSLT スタイルシートをマッチングさせる必要がある. " / "の XSLT スタイルシート内では唯一の子ノードである servicedata にマッチングするので servicedata に対する XSLT スタイルシートを作成する. それ以下の子要素に対してのマッチングには XSLT スタイルシートを作成しないことにする. この理由としては, 単純にどのようなサービスの組み合わせでも全ての要素をアクセスできるノードは servicedata ノードにあたるからである.

従って, current ノードを servicedata にすることで順次あてはめていく. 次に servicedata の XSLT スタイルシート内で BPEL プロセスを構成する要素を生成する call-template を作成する.

3.4.3. XSLT スタイルシートの作成

選択サービスから動的に WSDL インタフェースを組み込むために表 1 に示すように, BPEL プロセスの生成するアクティビティに対応する XSLT スタイルシートの名前を定義する. アクティビティ毎にあてはめる XSLT スタイルシートを用いることで動的に値を組み込めるようにする.

表 1 生成アクティビティと XSLT スタイルシート名

BPEL プロセスアクティビティ	XSLT スタイルシート名
<partnerLink>	Partner
<variable>(グローバル変数)	Variable
<variable>(初期値設定)	Init
<flow>	Flow

表 1 で示す<process>の各アクティビティに対して自動生成する XSLT スタイルシートを作成し, アクティビティを生成する際に XSLT スタイルシートを呼び出す.

XSLT スタイルシートの呼び出し方法を図 8 に示す.

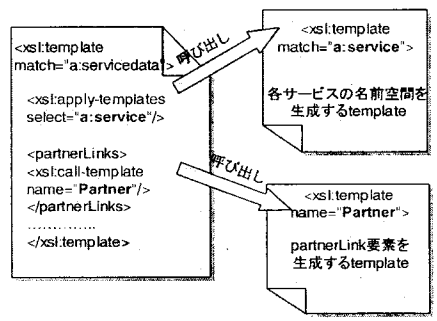

図 8 XSLT スタイルシート呼び出しの流れ

3.5. BPEL プロセスを自動生成するプロセスコントローラ

3.5.1. プロセスコントローラの役割

BPEL プロセスを自動生成するアクティビティ図を図 9 に示す.

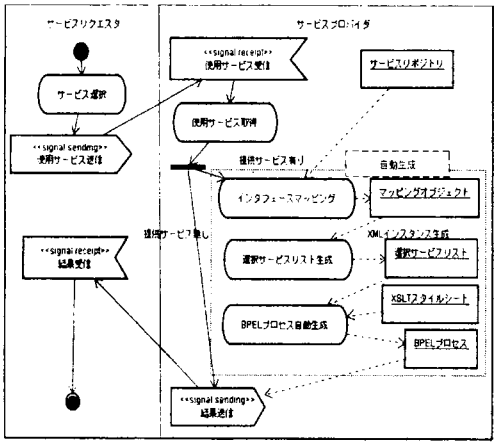

図 9 BPEL プロセス自動生成

動的に BPEL プロセスを生成するプロセスコントローラを設計する. プロセスコントローラがサービスの情報の入力を受け付けたときに, インタフェースのマッピングを行い, 選択サービスリストを生成し, BPEL プロセスを生成までを自動で行う.

3.5.2. プロセスコントローラのアーキテクチャ

データ変換を行うプロセスコントローラでは, アプリケーションのモデルをハブとして, 各種のデータフォーマットに変換するアーキテクチャを考える.

2 つのアーキテクチャのパターンを図 10 に示す.

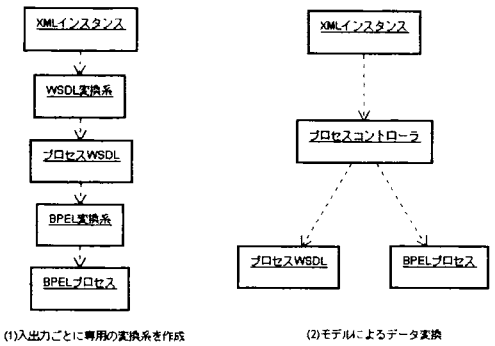

図 10 動作モデル

図 10 の(1)は逐次実行するアーキテクチャである. 入出力毎に専用の変換系を作成する必要がある. プロセスコントローラの対象となるフォーマットが少ない場合は効率が良いが, フォーマットが増えると対応が難しい.

(2)は並行実行するアーキテクチャである. モデルを作成することで新しいフォーマットを追加する場合でも, 追加するフォーマットとプロセスコントローラ間の変換処理を行うだけで対応できる.

本稿では(2)の手法で開発する. 選択サービスの情報を取得し, BPEL プロセス生成まで行うプロセスコントローラのクラス図を図 11 に示す.

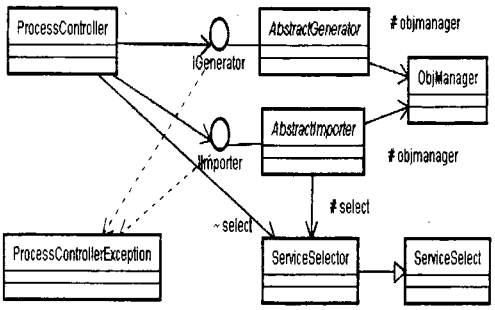

図 11 プロセスコントローラのクラス図

プロセスコントローラを構成する大きな 4 つのクラスとして ServiceSelect クラス, Importer クラス, ObjManager クラス, Generator クラスを作成した. プロセスコントローラの制御である BPEL プロセス生成までを図 12 のシーケンス図に示す.

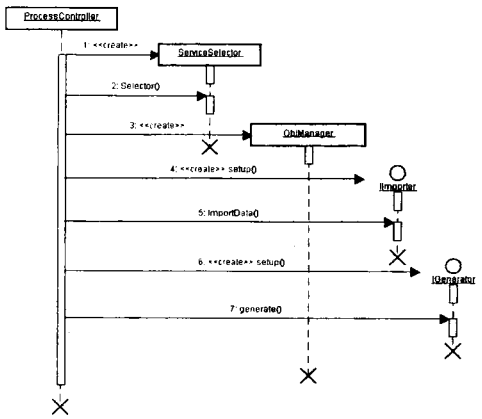

図 12 BPEL プロセス生成のシーケンス図

3.6. BPEL プロセス生成プロセスコントローラの実装

BPEL プロセスを実行可能な状態で提供するために必要な生成物をプロセスコントローラで生成するように実装する. 必要な生成物とクラス名を表 2 に示す. これらのクラス全ては図 11 に示す IGenerator クラスにアドオンする.

表 2 必要な生成物に対するクラス

必要な生成物	クラス名
選択サービスリスト	ServiceListGenerate
BPEL プロセス	BpelGenerator
使用する WSDL	WSDLGenerator

4. 例題による実験

4.1. 辞書サービス

(1) 辞書サービス

本論文では BPEL プロセスで扱う類似サービスの例として辞書サービスを利用する[10]. 使用するサービスを表 3 に示す.

表 3 使用サービス概要

サービス名	特徴 / URL
ICD	コンピュータ用語事典
http://www.iwebmethod.net/icd1.0/icd.asmx	
NetDic	英和・和英・国語辞典サービス
http://btonic.est.co.jp/NetDic/NetDicv08.asmx	
SimpleSearch	本論文で作成したサービス
http://app.nise.org/axis/services/SampleSearchService	

(2) 各サービスのインタフェースの対応

3 つの Web サービスのサービスとプロセスのインタフェースの整合性をとるために, 見出し検索で部分検索にあたるメソッドの対応関係, メソッドの引数, 返り値にあたるデータの対応, また複合 Web サービスのインタフェースとの対応付けを行う.

4.2. サービスリストスキーマの作成

属性を付加して同一の機能間で対応付けたインタフェースの一覧を表 4 に示す.

WSDL の記述の新たに追加する属性は XML スキーマにより定義する. 作成するスキーマは, 属性の定義とその属性名を定義する.

本論文では BPEL プロセスを生成する方法として XSLT を採用したので, その変換元となる XML インスタンスである選択サービスリストを生成する.

(1) 複合 Web サービスの WSDL ファイルの生成

提供する全てのサービス情報を保持した BPEL プロセスの WSDL を生成する. マッピングオブジェクトを生成して, その中で使用しないサービスのインタフェース情報を削除する.

(2) BPEL プロセスの自動生成

選択されたサービスによって生成されるファイル, またファイルの内部構造が変化する. 表 5 に示すように, 共通で生成されるファイルと, 選択サービスによって異なるファイルに分ける. 実験ではこれらのファイルを, ディレクトリ ComplexWebService 内に生成する.

表 5 共通する生成ファイル

	ファイル名	インタフェース
共通	ComplexWebService.bpel	選択サービスにより変化
	ComplexWebService.wsdl	
選択	選択サービス名.wsdl	各サービスの WSDL

4.3. 実験方法

プロセスコントローラを使用し, BPEL プロセスを自動生成する実験を行った. まずリクエスタが要求したサービスリストを受信しインタフェースのマッピングを行い, その変換元となる XML インスタンスである選択サービスリストを生成する. XSLT スタイルシートと選択サービスリストにより BPEL プロセスを生成する. ここまでの処理を自動で行う.

表 6 に本稿での開発, 実行環境を示す.

実行エンジンは BPEL Process Manager[12]を使用した. BPEL Process Manager は既存のミドルウェアやプラットフォームに依存せずに BPEL エンジンを提供できるサーバである. BPEL Process Manager は Web サービスをデプロイするファイルを自動生成する. また, デプロイした BPEL プロセスのリクエスタプログラムが実験用として提供されている.

表 6 開発, 実行環境

OS	WindowsXP
CPU/メモリ	Pentium 4(2.66GHz), 1GB RAM
Web サーバ	Tomcat 5.0.28, Apache Axis 1.1
実行エンジン	BPEL Process Manager
Java 開発環境	J2SDK 1.4.2
XML パーサ	xerces-1_4_4

表 4 類似機能における対応付け

サービス名 / 属性	ICD サービス [対応部分名]	NetDic サービス [対応部分名]	Simple サービス [対応部分名]	複合 Web サービス インタフェース[対応部分名]
RelationOperation (operation)	SearchWord	SearchDiCItem	Search	initiate, onResult
SearchInfo (message)	SearchWordSoapIn SearchWordSoapOut	SearchDicItemSoapIn SearchDicItemSoapOut	SearchRequest getItemResponse	ComplexWSRequestMessage ComplexWSResponseMessage
InputWord (message/part)	parameters (SearchWord)	parameters (SerachDicItem)	in0	payload (ComplexWSRequest)
OutputWord (message/part)	parameters (SearchWordResponse)	parameters (SearchDicItemResponse)	getItemReturn	payload (ComplexWSResponse)
inputRelation (element)	Query	Words	なし	in0
outputRelation (element)	japanese	body	mean, keyword	keyword, icdmean, dicmean, sammean

5. 評価

5.1. BPELプロセスの自動生成

各サービスに同一な機能のデータを決定し，各WSDLにおいて対応付けることによって各サービス間に対応関係を持たせた．これにより類似したサービスの同一機能インタフェースを抽出し，そのインタフェースを組み合わせることによって類似機能を提供する複合Webサービスを自動生成できた．

5.2. BPELプロセス生成評価

本論文でBPELプロセス生成に用いたプロセスコントローラを評価する．新しいサービスを追加する拡張性や変更の柔軟性の視点で評価する．

プロセスコントローラでは，WSDLのインタフェースを一度解析しオブジェクト化することで解析処理を自動化できる．

サービスの追加を行う時には，本稿で作成したServiceInfoクラスにサービス情報を記述し，マッピングのモデルとして作成したObjManagerクラスにマッピングオブジェクトを登録することで，サービスの追加が容易にできる点で拡張性があるといえる．

次に，BPELプロセス生成に選択サービスリストの生成とXSLTを用いたのでこれを評価する．

マッピングしたマッピングオブジェクトより直接DOMを用いてBPELプロセスを生成した場合と，選択サービスリストを作成してXSLTにより生成した場合の2つを比較する．ICD, NetDic, Simpleの3つのサービスを使用して実験を行った．

DOMで直接BPELプロセスを生成した場合，実行系のコード数は553行となった．また，選択サービスリストとXSLTによってBPELプロセスを生成した場合では1163行となった．

1つのサービスのマッピングオブジェクトから1つのサービスのBPELプロセスのアクティビティを生成するために約130行のコード数が必要であった．従って，サービスを追加する毎に約130行増える．

選択サービスリストの生成ではサービスの追加に対してマッピングオブジェクトの数だけ繰り返して要素を生成するので，追加コード数は約30行となる．

また，XSLTのスタイルシートは1つのサービスに対して約30行で追加できることが分かった．従って，1つのサービスの追加に約60行が必要となる．

この結果をまとめるとサービスを追加するためのコードの増加は表7に示すものとなる．

表7 サービス追加のコード数の変化

生成方法	コード行数
DOMによるBPELプロセス生成	553行+130*サービス数
選択サービスリスト+XSLT	1163行+60*サービス数

追加サービス毎に記述される総コード数を図13に示す．

BPELプロセスの自動生成は，DOMが選択サービスリストとXSLTを用いた場合の約半分のコードで済む．しかし，サービスの追加を行うのに必要なコード数は選択サービスリストとXSLTを用いた方がDOMの半分以下である．

従って，追加するサービス数が一定数を越えると，DOMで生成するよりも選択サービスリストとXSLTを用いた生成の方が効率のよいサービスの追加が可能であるといえる．

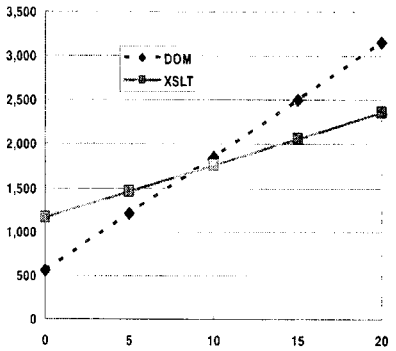

図13 追加サービスにおけるコード数の変化

本論文では並行実行のプロセスのみ生成するので，DOMで生成するBPELプロセスでも対応できる．しかし，サービスの実行順序が動的に変更される場合，DOMを用いた方法で生成すると構造の変更が困難であり，XSLTを用いて構造を変更し直す必要がある．構造を柔軟に変更できるXSLTはDOMを用いた方法より優れていると言える．このような結果からBPELプロセスの生成方法は柔軟性と拡張性があると評価できる．

6. 考察と今後の課題

6.1. 選択サービスリストにおける対応

現在の共通する機能の対応関係だけでは，そのサービスが持つ特有の機能を使用することはできない．従って，類似な機能の対応だけでなく，サービス特有の機能も使用可能となる，対応関係の構築が必要となる．

6.2. BPELプロセスの自動生成

BPELプロセス生成プログラムは，提供できるサービス内では実行時にプロセスを生成することが可能ではあるが，プロバイダがあらかじめ発見しておく必要がある．従って，サービスを動的に発見することが今後の課題として挙げられる．

また，BPELプロセスにおけるクライアントによるリク

エストとレスポンスに対するportTypeを指定する際に，本システムはリクエストとレスポンスを異なるportTypeで分けて考えた．しかし，辞書サービス以外の複合Webサービスを提供したときに1つの複合Webサービスによってリクエストとレスポンスのインタフェースは異なるので，利用者側はどのportTypeを使用すればサービスを利用できるかが困難になると考えられる，従って，1つの複合Webサービスに対してリクエストとレスポンスを1つのportTypeにまとめて指定する方法が望ましいと考えられる．

6.3. モデル駆動Webサービス開発

Webサービスの合成(composition)にモデル駆動アーキテクチャ(MDA: Model-Driven Architecture)を適用する方法が提案されている[4, 13]．しかし，従来の方法は，サービスの非機能的特性や内容に基づく動的選択，結合には対応していない．今後，サービスの実行コンテキストやサービスの品質などに応じてサービスの動的合成を支援するモデル駆動型Webサービス開発方法論の検討が必要である．

7. まとめ

本論文では，ビジネスプロセスの自動化を実現するため，BPELを用いた複合Webサービスの構築を提案した．特に類似サービスの連携に注目し，モデルに基づくBPELプロセスの自動生成方法を提案した．サービス毎のWSDLにより類似サービスの対応を行い，インタフェース間で類似の機能，要素を対応させた．

プロトタイプによる実験で行った結果よりサービスの選択だけでBPELプロセスと，使用するサービスのWSDLファイルの自動生成を実現した．またBPELプロセスの構造を変更の柔軟性を考えXSLTスタイルシートを用いて生成することを提案し，実装して実験結果，実現方法の比較によりBPELプロセス自動生成プログラムの有用性が評価できた．

参考文献

[1] G. Alonso, et al., *Web Services*, Springer, 2004.

[2] T. Andrews, et al., *Business Process Execution Language for Web Services*, Ver. 1.1, May. 2003, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[3] 青山 幹雄，ソフトウェアサービス技術へのいざない，情報処理，Vol. 42, No. 9, Sep. 2001, pp. 857-862.

[4] K. Baina, et al., Model-Driven Web Service Development, *Proc. of CAiSE 2004*, Jun. 2004, pp. 290-306.

[5] 福永 遂重，高木 松雄， BPEL4WSを用いた複合Webサービスの研究， 2004年度南山大学数理情報学部情報通信学科卒業論文，Jan. 2005.

[6] IBM jstartチーム，まるごと図解 最新Webサービスがわかる，技術評論社，2003.

[7] IBM Web Services ビジネスプロセスBPEL4WSについて，2002, http://www-6.ibm.com/jp/developerworks/webservices/021025/j_ws-bpelcol1.html#1.

[8] 丸山 宏ほか， XMLとJavaによるWebアプリケーション開発， 第2版， ピアソン・エデュケーション，1999.

[9] 本 俊也，最新Webサービスマスタリングハンドブック，秀和システム，2004.

[10] 中村 一仁，柘植 亮人，青山 幹雄，価値を用いたWebサービスの動的連携ブローカとその評価， 情報処理学会ソフトウェア工学研究会，Vol. 2003-SE-144, No.17, Mar. 2004, pp. 123-130.

[11] 屋内 恭輔，安陪 隆明，XMLスキーマ書法，毎日コミュニケーションズ，2003.

[12] Oracle BPEL Process Manager，2004，http://www.oracle.com/technology/products/ias/bpel/index.html.

[13] B. Orriens, et al., Model Driven Service Composition, *Proc. of ICSOC 2003*, LNCS, Vol. 2910, Springer, Dec. 2003, pp. 75-90.