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

2003－UBI－2 (28)
2003／11／19

ユビキタスデバイスからのWebサービス利用を支援するWebサービスプロキシエージェント

川村隆浩[†,††] 寺崎達也[††]
大須賀昭彦[†,††] 前川守[††]

近年，Webサービスがe-Businessのインフラといて注目を集めている．一方で，一般ユーザの間では携帯電話やPDAに代表されるユビキタスデバイスが広く普及している．そこで，本論ではユビキタスデバイスからWebサービスを利用する際の問題点として，インタラクションの多さと処理時間の長さを挙げ，Webサービスによるビジネスプロセスをユーザの要求に応じて代理実行するプロキシエージェントを提案する．また，Webサービスによる ATM サービスのビジネスプロセスを例にプロキシエージェントの生成方法を説明し，インタラクションの削減効果，処理時間，Webサービスフロー言語 BPEL への適用性について評価し，その有効性を確認する．

Web Services Proxy Agent managing the interaction between Web Services and Ubiquitous devices

TAKAHIRO KAWAMURA,[†,††] TATSUYA TERASAKI,[††]
AKIHIKO OHSUGA[†,††] and MAMORU MAEKAWA[††]

Web Services is becoming an infrastructure of e-Business these days. On the other hand, ubiquitous devices such as cell phones and PDAs are already popular gadgets for the end users. In this paper, we focus on the interaction between Web Services and the ubiquitous devices, which reduces the usability of services for those users and increases the total processing time. Thus, we propose the proxy agent who runs business services on the users' demand, then illustrate how to automatically create the agent through a ATM service example. Finally, we examine the reduction of the interaction, the processting time, and applicability to the standard Web Services flow language, BPEL.

1. はじめに

近年，Webサービスがe-Businessの今後のインフラといて注目を集めている．一方，一般ユーザの間では携帯電話やPDAに代表されるユビキタスデバイスが広く普及している．しかし，Webサービスはまだバックエンドで試行的に導入されているに過ぎず，フロントエンドのユビキタスデバイスとをつなぐ明確なパスは存在しない．しかし，今後数年の間にIT設備投資が一巡すれば，多くのサーバ間連携システムがWebサービスをベースにしたものになることは一般に予想されている．そこで，本論ではユビキタスデバイスからのWebサービス利用時に想定される問題点に焦点をあてる．

現状のWebサービスは(多くのWebアプリケーションも同様に)，ブロードバンド環境を利用可能なPCユーザを主なターゲットに設計，開発されている．結果的に，Webサービスによるビジネスプロセスはユーザとのインタラクション(入出力情報のやり取り)が多く，ユビキタスデバイスのユーザにとっては以下の点から利用しにくいものといえる．

- 多くのユビキタスデバイスは操作がしにくい
- 移動しながらの利用の場合，ネットワークが不安定
- 通信コストが高い

加えて，PCのブロードバンド環境に比べれば3G携帯といえども通信回線が細く，全体としての処理時間(ユーザの待ち時間)が長くなる．

そこで，我々はユビキタスユーザのために，Webサービスを代理実行するWebサービスプロキシエージェントを開発した．以下，2章においてWebサービスプロキシエージェントの概要について述べ，3章に

† (株) 東芝 研究開発センター
Research & Development Center, Toshiba Corp.
†† 電気通信大学大学院 情報システム学研究科
The Graduate School of Information Systems, University of Electro-Communications

てATMサービスを例にとりWebサービスプロキシエージェントの具体的な処理を示す．更に，4章において本システムにおけるインタラクション削減効果，処理時間および本手法の適用性に関する評価と考察を行い，5章にて関連研究について述べる．最後に，6章においてまとめとする．

2. Webサービスプロキシエージェント

2.1 プロキシエージェントの概要

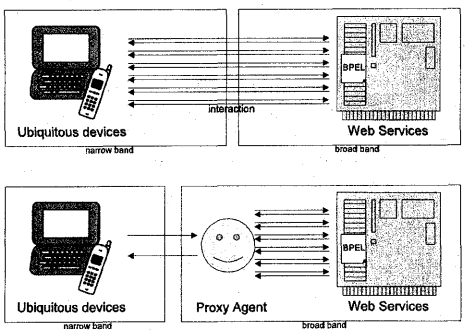

図1 Webサービスプロシキエージェントの位置づけ
Fig. 1 Relationship to Web Services Proxy Agent

図1上段は，モバイルユーザがWebサービスを利用する場合を表したものである．ここで，ユーザとWebサービス間のインタラクションを減らすため，その中間にエージェントを導入する(図1下段)．エージェントは，ユーザとWebサービスとのインタラクションを減らし，サービス利用時のトータルな処理時間を減らすために，ユーザの要求に応じて一連のWebサービスからなるビジネスプロセスを自動的に実行する．なお，Webサービスによるビジネスプロセスは現在OASISにおいて標準化が進められているWebサービスフロー言語BPEL(Web services Business Process Execution Language)[1]にて記述されるものとする．エージェントは，あらかじめBPELファイルを解析し，プロセス実行時に必要とされる入力情報をインタラクション開始時にユーザに一括入力させる．これにより，エージェントは後に続く処理をユーザとのインタラクションなしに自動実行することが可能となり，ユーザはサービス利用全体を通してのインタラクションの総数を減らすことができる．エージェントは，Webサービスによるビジネスプロセス(つまりBPEL)単位で自動的に生成される．次節において，エージェント生成方法を示す．

2.2 プロキシエージェントの生成方法

エージェント生成は，「クライアントプロセスの生成」と「入力情報の抽出」の2つのフェーズから成る．

A. クライアントプロセスの生成

クライアントプロセスとは，BPELエンジン上で実行されるWebサービスによるビジネスプロセス(BPEL)に対応するクライアントサイドの一連の処理を指す．BPELには，Webサービスの呼び出し，データの操作，障害の通知，プロセスの終了などアクティビティと呼ばれる処理単位が存在する．クライアントプロセスは，これらのアクティビティの構造や属性値，イベントなどを解析することで生成される(なお，BPELファイルは対象とするWebサービスの提供元よりダウンロード可能であるとする)．クライアントプロセスの生成は，対象とするビジネスプロセス(BPELファイル)に以下のルールを適用することで実行される．

A1. 属性partnerの処理 BPELのアクティビティやイベントにはpartnerと呼ばれる属性が存在し，この値によって通信相手を特定できる．Webサービスは基本的にリクエストに応じてサービスを実行するという受動的なものであるため，最初にリクエストを送ってきたpartnerをユーザと判断する．

A2. 内部処理の削除 ユーザ以外のpartnerとのインタラクションを行っているアクティビティや，BPELエンジンが内部で行うデータ処理などのアクティビティは，クライアントプロセス側には不要のため削除する．例えば，変数のコピーを行うアクティビティassignは無視することができる．

A3. 処理の送受反転 BPELにおけるメッセージ受信を送信に，送信を受信に変換する．具体的には，メッセージ送信，返信を行うときに使われるinvoke, replyは，receiveに変換する．反対に，メッセージ受信を行うreceiveはinvokeへ変換する．また，受信と送信を行うときに使われる，

`<invoke inputVariable="Req" outputVariable="Res">`

は，`<invoke variable="Req">`と`<receive variable="Res">`に変換する．なお，メッセージの内容による分岐を表すpickとonMessageの組は，個々のonMessageに対応するinvokeに変換する．while, swithなどの制御フローはそのまま再現する．

図2に，簡単な変換例を示す．

B. 入力情報の抽出

クライアントプロセスは，様々な処理内容を含むビジネスプロセス全体をクライアントサイドに写し取ったものといえる．ここでは，それを基にユーザの行いたい処理(ゴール)に応じて必要とされる入力情報を

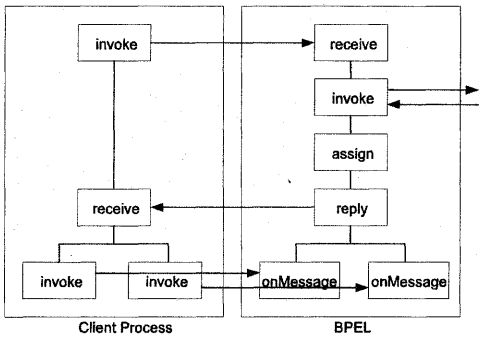

図 2 BPEL からクライアントプロセスへの変換例
Fig. 2 Translation from BPEL to Client Processes

洗い出す処理を行う．例えば，後述する ATM の例では，ビジネスプロセスには預金照会，振込み，引き出しなど複数のゴールが存在する．それに対して，ユーザが預金照会を行いたい場合は，預金照会に至るプロセスにおいて入力が必要とされる情報のみを洗い出す (つまり，振込みに必要な振込み金額などの情報は必要ない)．

入力情報の抽出では，まずクライアントプロセスをプラニングにおける知識ベースとなるアクションオペレータ (AO) に変換する．次に AO を基にプラニングによってユーザが与えたゴールに至るプロセスのリスト (プラン) を生成する．最後に，生成されたプロセスリスト内に含まれるサービス利用時に必要な引数情報を収集する．

B1. AO への変換 BPEL における 1 アクティビティやアクティビティに似た役割をするイベントを 1 つの AO(自状態を表す事前条件，次状態を表す事後条件，処理内容を表すアクションの組みからなる) に変換する．基本的にアクティビティ名を AO 名として使用するが，複数のアクティビティから選択するような場合は同じ名前を持つ AO を複数生成する．そうすることで，プラニング時にプラナーが全ての組み合わせを試行することができる．それ以外の場合は，AO 名はユニークになるように数字と組み合わせる．図 3 は，BPEL から AO への変換を模式的に表したものである．

B2. プラニング 指定されたゴールに至る AO のリスト (プラン) を作成する．なお，条件分岐に対しては複数のプランを生成し，実行時に最短プランから優先的に実行するものとする．図 4 に概念図を示す．

B3. 引数情報の取得 抽出された AO 内に含まれている，BPEL エンジンとのインタラクションに必要な引数 (入力情報) を，WSDL のメッセージタイプを

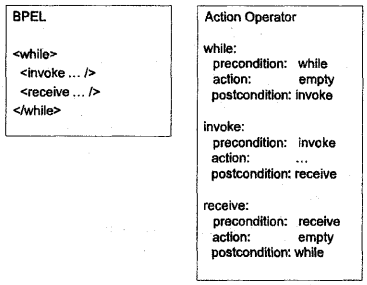

図 3 クライアントプロセスから AO への変換例
Fig. 3 Translation from Client Processes to AOs

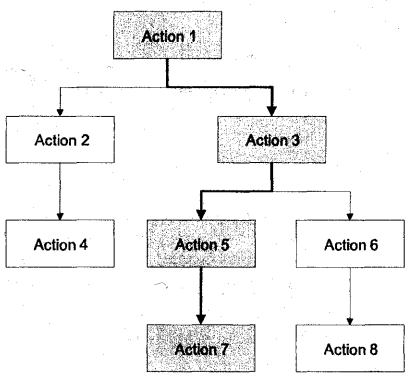

図 4 プラニングによるアクション抽出
Fig. 4 Action extraction fro AOs

チェックすることで洗い出す，一種のユニフィケーション (単一化) を行う．

なお，クライアントプロセスの生成と入力情報の抽出はエージェント作成時 (つまり，BPEL ファイルが与えられた時点) に行われる．そして，特定のゴールに応じたメソッドがエージェントに実装され，WSDL として公開される．クライアントは WSDL を基に SOAP によるアクセスを行うことで，エージェントを呼び出すことができる．

3. 応用事例 – ATM サービスの利用

以下，ユビキタスユーザが ATM Web サービスを利用する場合を例を挙げてプロキシエージェントの生成を行う．なお，ここで使用している ATM サービスの BPEL ファイルは，IBM の alphaWorks[2] より提供されている BPEL エンジン BPEL4J にてサンプルとして紹介されているものである．図 5 に BPEL のアクティビティを示す．

ここで，図中央に位置しているのが ATM サービスを行う BPEL プロセスであり，図左にあるのが振り込みや引き出しなどをリクエストするクライアント

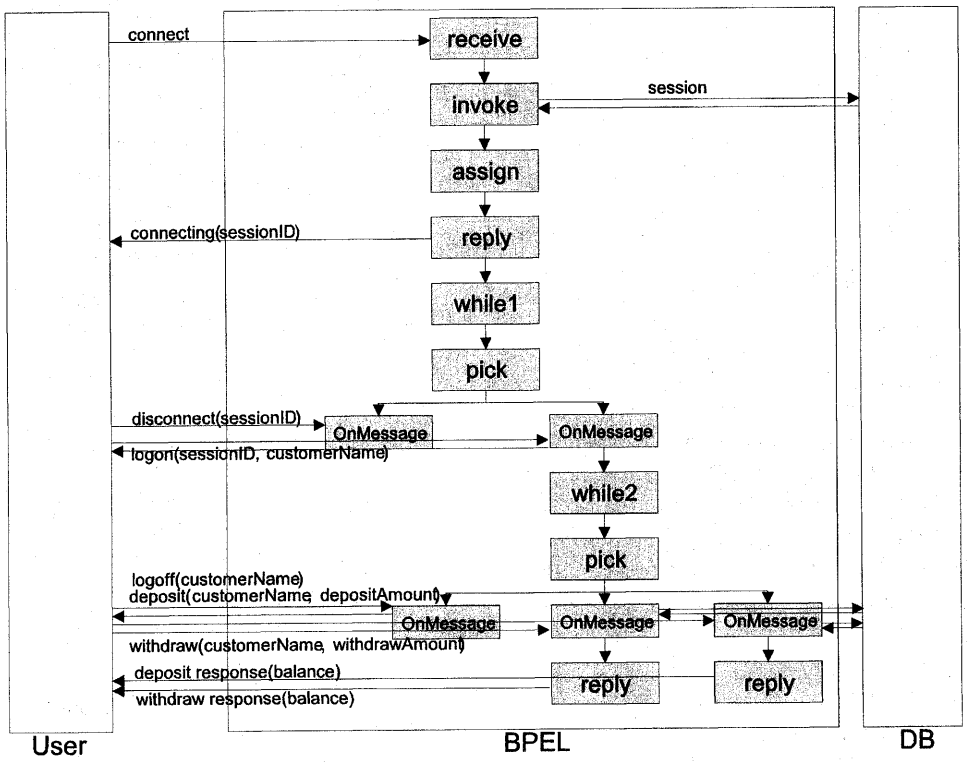

図 5 ATM Web サービスの BPEL
Fig. 5 BPEL for ATM Web Services

(ユーザ)，図右にあるのが DB などのバックエンドである．このビジネスプロセスの流れをおおまかに説明すると，まずクライアントが ATM サービスの利用をリクエストすると，BPEL プロセスがバックエンドにコネクションを繋ぎ，その結果をクライアントに返す．コネクションが成功であった場合には，while1 に入り，ユーザは disconnect または logon メッセージを送る．ユーザが logon メッセージを送った場合は，while2 に入り，ユーザは deposit か withdraw かを選択する．deposit であった場合は預金金額を，withdraw であった場合は引き落とし金額が入力として求められる．

この BPEL プロセスを基に生成されたクライアントプロセスを図 6 に示す．

次に，これを AO に変換した例を図 7 に示す．なお，本論文ではプラナーの実装系としては SHOP(Simple Hierarchical Ordered Planner)[3] をベースに，Java に移植した JSHOP を利用している．そのため，AO は S 式で記述されている．

ここで，method とは前提条件が成り立つならば実行される処理リストを表し，(:method メソッド名＋

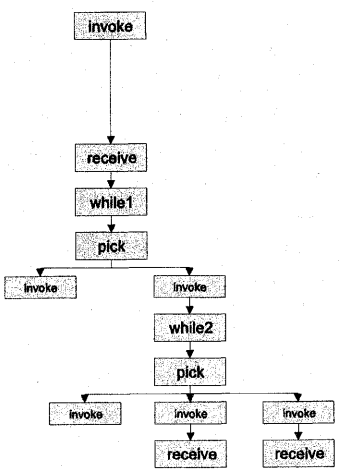

図 6 ATM Web サービスのクライアントプロセス
Fig. 6 Client Processes for ATM Web Services

引数 前提条件 処理リスト) という文法を持つ．また，operator とは前提条件が成り立つならば削除リストを削除し，追加リストを追加するという意味を表し，

```
(defdomain atm
  (
;;;;;;;;;;;;;;;;;;;;;;method;;;;;;;;;;;;;;;;;;;;;;;;
    (:method (invoke1)
      ((state invoke1))
      ((!change-state invoke1 receive1)(receive1)))

    (:method (receive1)
      ((state receive1))
      ((!change-state receive1 while1)(while1)))

    (:method (while1)
      ((state while1)(intention disconnect))
      ((!change-state while1 disconnect)(disconnect)))

    (:method (while1)
      ((state while1))
      ((!change-state while1 logon)(logon)))

    (:method (disconnect)()())

    (:method (logon)
      ((state logon))
      ((!change-state logon while2)(while2)))

    (:method (while2)
      ((state while2)(intention deposit))
       ((!change-state while2 deposit)(deposit)))

    (:method (while2)
      ((state while2)(intention withdraw))
       ((!change-state while2 withdraw)(withdraw)))

    (:method (while2)
      ((state while2)(intention logoff))
       ((!change-state while2 logoff)(logoff)))

    (:method (deposit)
      ((state deposit))
      ((!deposit)(!change-state deposit while2)
       (!change-intention deposit logoff)(while2)))

    (:method (withdraw)
      ((state withdraw))
      ((!withdraw)(!change-state withdraw while2)
       (!change-intention withdraw logoff)(while2)))

    (:method (logoff)
      ((state logoff))
      ((!change-state logoff while1)
       (!change-intention logoff disconnect)(while1)))

;;;;;;;;;;;;;;;;;;;;;;operator;;;;;;;;;;;;;;;;;;;;;;;;

    (:operator (!change-state ?pre ?post)
       ((state ?pre))((state ?pre))((state ?post)))

    (:operator (!change-intention ?pre ?post)
       ((intention ?pre))((intention ?pre))
       ((intention ?post)))

    (:operator (!deposit)()()())
    (:operator (!withdraw)()()())
))
```

図 7 ATM Web サービスのアクションオペレータ
Fig. 7 AOs for ATM Web Services

(:operator オペレータ名＋引数 前提条件 削除リスト 追加リスト) という文法で記述される.

更に，withdraw をゴールとしてプラニングを行い，抽出された AO のリストを図 8 に示す.

ここで，エージェントは change-state で表示された順番に AO を実行することで，ユーザの与えたゴール withdraw を自動実行できる．なお，change-intention はゴールの変更を意味しており，ここでは withdraw というゴールを達成したために，終了処理である logoff と disconnect にゴールを移行している.

```
( (!change-state invoke1 receive1  )
  (!change-state receive1 while1  )
  (!change-state while1 logon  )
  (!change-state logon while2  )
  (!change-state while2 withdraw  )
  (!withdraw  )
  (!change-state withdraw while2  )
     (!change-intention withdraw logoff  )
  (!change-state while2 logoff  )
  (!change-state logoff while1  )
     (!change-intention logoff disconnect  )
  (!change-state while1 disconnect  ))
```

図 8 withdraw プラン
Fig. 8 Plan for withdraw

最後に，withdraw というゴールを達成するために抽出された入力情報は customerName，withdrawAmount の 2 つであった．この結果は，図 5 を目で追うことによって確認できる.

4. 評価と考察

前章の事例を基に，ユーザが直接 Web サービスを実行する場合とエージェントを介して実行した場合のユーザのインタラクションを比較したものを図 9 に示す．この図より，エージェントを介することでユーザの手間を大きく削減できたことが確認できる．なお，SOAP メッセージサイズをベースにユビキタスデバイス上での通信量を計測した結果，約 34%になっていることが確認できた.

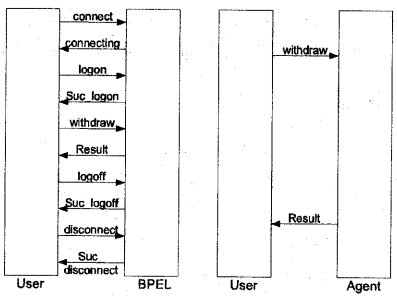

図 9 インタラクションの比較図
Fig. 9 Comparison of usr interaction

また，現在開発中のシステムを基に本事例を走らせた場合の処理時間を算出したところ，処理時間の合計は約 74%になることが確認できた (むろん，この結果は通信環境や実行プラットフォームによって大きく異なるため，ひとつの目安に過ぎない)．なお，エージェントと BPEL エンジンとの間は LAN で結ばれており，ユーザー Web サービス間，ユーザーエージェン

ト間の PHS64k と比較して通信に要する時間は無視できる程度である．また，エージェント生成にかかる時間，およびユビキタスデバイス側で WSDL や BPEL ファイルから SOAP クライアントオブジェクトを生成する時間は前処理であるため考慮していない．今回の実験からは，通信時間よりもサービス処理にかかる時間の方がオーダーが大きく，かつ両者において BPEL エンジン側の処理速度には変わりはないため，トータルな処理時間はユビキタスデバイス (今回は NotePC で代用) とエージェント側サーバの処理速度に大きく依存することが確認された．

最後に，本システムの Web サービスフロー言語 BPEL への適用性について，今回の事例適用から以下の考察が得られた．

- 個々のアクティビティに関しては変換可能
  例：invoke, reply, receive, pick, while, switch, scope, faultHandlers など
- イベントドリブンな処理には対応できない
  例：常に実行できる処理を表す eventHandler アクティビティなど
- Web サービス側で変数を返し，その値をアクティビティの制御に利用する場合には，条件付きプランを生成する，またはインターリーブを導入するなどの工夫が必要，今回実装に使用した JSHOP では対応していない．
  例：while で何回ループするか不明，swith で分岐先が確定できない，など

今後も実用的な BPEL ファイルを対象とし，インタラクション削減効果や性能面の検証を行っていく．

5. 関連研究

Petrone et al[4] では，ビジネスプロセスをサーバ側が管理し，コンテキストに応じて，ユーザに選択可能なオペレーションを随時提供することで，ユーザがどのようなメッセージを送信すればよいかをガイドするシステムについて述べられている．これは，ユーザのインタラクションに注目した点において本研究と類似しているといえるが，ユビキタスデバイス利用時のインタラクション削減と言った観点は考慮されていない．

また，WSDL2JADE[5] では，Web サービスとエージェントの連携を目指し，SOAP とエージェント間通信言語 ACL を相互変換するツールを提供している．但し，変換は 1 対 1 で行われており，上と同じくインタラクション削減と言った観点は考慮されていない．

6. まとめ

本論では，ユビキタスデバイスから Web サービスを利用する際の問題点として，インタラクションの多さと処理時間の長さを挙げ，Web サービスによるビジネスプロセスをユーザの要求に応じて代理実行するプロキシエージェントの開発について述べた．また，事例を用いてインタラクションの削減効果，処理時間，Web サービスフロー言語 BPEL への適用性について評価し，その有効性を確認した．

今後の課題としては，ユーザが一度に全ての入力情報を与えたくない場合 (あるいは，与えられない場合) に，該当項目を未入力 (空文字) にしたままにすることを可能としたい．エージェントはプラン実行時に未入力情報に出会った場合，プラン実行をインターリーブしその時点でユーザに不足情報の入力を促す．それによって，ユーザは必要な情報を一度に送るか，少ない情報を複数回に分けて送るか (通信量と通信回数のトレードオフ) を選択できるようになるだろう．また，ユーザ固有のプロファイルや，前回実行時のヒストリ情報を利用することも検討している．

なお，本論ではプレゼンテーションレイヤーでの実現方法については触れていないが，ユビキタスデバイス上のクライアントが kSOAP などを用いて SOAP メッセージを直接送信するという方法以外にも，プロキシエージェントが Web サーバーを介して必要な情報の入力を促す HTML のフォームをユビキタスデバイス上に表示するなどの方法も考えられる．

参考文献

1) BPEL, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
2) IBM alphaWorks, http://www.alphaworks.ibm.com/
3) Nau, D., SHOP:Simple Hierarchical Ordered Planner, http://www.cs.umd.edu/projects/shop/
4) Petrone, G., Managing flexible interaction with Web Services, Proceedings of Workshop on Web Services and Agent based Engineering (WSABE 2003), July, 2003
5) WSDL2JADE, http://sas.ilab.sztaki.hu:8080/wsdl2jade/index.html.