特集論文

ユビキタス環境とWebサービスを仲介するミドルエージェントの開発

Development of Middle Agents Mediating Ubiquitous Environment and Web Services

寺崎 達也
Tatsuya TERASAKI
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
tat@maekawa.is.uec.ac.jp

中井戸 健至
Takeshi NAKAIDO
（同　　上）
nakaid-t@maekawa.is.uec.ac.jp

川村 隆浩
Takahiro KAWAMURA
(株) 東芝／電気通信大学　大学院情報システム学研究科
TOSHIBA Corporation / The Graduate School of Information Systems, University of Electro-Communications
kawamura@maekawa.is.uec.ac.jp

大須賀 昭彦
Akihiko OHSUGA
（同　　上）
ohsuga@maekawa.is.uec.ac.jp

前川 守
Mamoru MAEKAWA
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
maekawa@maekawa.is.uec.ac.jp

keywords: agent, planning, Web Services, BPEL

Summary

According to the widespread use of Web Services, composition of multiple services is becoming a main issue. BPEL was proposed as a Web Services workflow language in this context. Then, in many cases we need much interactions between a service requester and a BPEL process execution engine. However for the end user under the ubiquitous environment where the service is executed with a cellular phone or PDA, the interaction means simply a burden. The reason is: (1)it is hard to carry out the operation manually, (2)a network connection is unstable, (3)a connection charge is relatively high. In this paper, we propose middle agents between ubiquitous users and Web Services. The agents analyze BPEL, extract the interaction which the ubiquitous users would need, then produce action operators. It makes a plan in order to achieve a service request from the user. Here the planning agent can reduce operations of the user by collecting the necessary information at the beginning which will be inputted during the IP execution. We show the architecture of the agents and an example regarding ATM service, then evaluate the number of inputs, the performance, and the quantity of data transmitted on network. Finally, we consider adaptability of this approach to generic BPEL flow.

1. ま え が き

近年，Web サービスが注目を集めている．Web サービスとは SOAP, WSDL, UDDI などの技術を中心にしてソフトウェアの相互運用を可能とするものである．これは従来企業間のコンポーネントを対象としていたが，今後は企業間だけでなくエンドユーザに対しても Web ブラウザなどを通して Web サービスが提供されるようになるだろう．

このような Web サービスを複数利用する際に，それらを統合してフローとして記述する言語 Business Process Execution Language for Web Services(BPEL) [BPEL 03] が提案されている．この BPEL を使用することによりデータのコピーや簡単な計算などの処理，外部とのインタラクションなど複数のプロセスをまるで 1 つのサービスであるかのように見せることが可能になる．

しかし，ユーザごとにカスタマイズを行う場合や複雑なサービスを利用する場合などではユーザ側とのインタラクションが多く必要となる．例えば図 1 の場合は，ユーザがサービスのリクエストをして BPEL プロセス実行エンジン (BPEL エンジン) がサービスを提供する前に，数回ユーザに情報を要求している．このようにユーザとのインタラクションが増えるとその分ユーザの操作が煩雑になり実行時間も長くなってしまう．特にユビキタス環境下に置けるエンドユーザにとっては

- 携帯端末は操作がしにくい

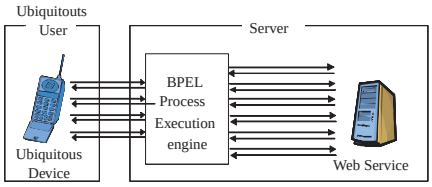

図 1 現状のユーザー Web サービス間モデル

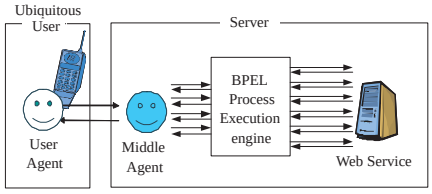

図 2 本研究でのユーザー Web サービス間モデル

- ネットワークが不安定
- 通信コストが高い

などの理由によりインタラクションは少ない方が望ましい．

この問題に対して，本論文ではエンドユーザと Web サービスの間にミドルエージェントを仲介させる方法をとる．エンドユーザの代わりにエージェントが複雑なインタラクションを行うことでユーザの関与を減らし，上記の問題の解決を試みる（図 2 参照）．

以下，2 章では実際にエージェントを生成する方法について記載する．3 章では ATM サービスを対象とした例題を記載する．4 章では ATM サービスを基にした評価と考察を行う．5 章で関連研究を記載する．6 章で本論文のまとめを行う．

2. エージェントの生成手順

現状では本の購入を行える Web サービスを利用したい時，Web 上の本屋の名前をユーザが選択や入力をする．すると，欲しい本のタイトルや著者名を尋ねられ，これらを入力し，送信すると名前やクレジット番号などの個人情報入力を要求され，これらを送信すると結果が送られてくるというような手順となる．ここでは 3 回の入力及びデータ送信を行わなければならない．しかし，本論文で提案するエージェントを用いた場合は，最初に本屋の名前をユーザが送信した後，BPEL の解析が行われることでミドルエージェントが生成され，タイトル，著者名，個人情報をまとめてユーザに問い合わせ，ユーザもまとめて答える．そして，これらのデータを受け取ったミドルエージェントがユーザの代わりにタイトル及び著者名を Web サービスへ送信し，個人情報を要求されたら先に受信した個人情報を送信することによって，同じ Web サービスの利用に対しユーザは 2 回の入力だけでサービスを受けることができる．

このようなフローが書かれている BPEL ファイルの中には Web サービス呼び出しの実行，データの操作，switch や while などの制御文，障害（fault）の通知，プロセスの終了などのさまざまなアクティビティがある．これらの組み合わせによって同じ BPEL ファイルでも実行時にフローが異なる場合がある．例えば，Amazon のようなショッピングサービスにて，本を購入する際にカード決済と着払いの 2 種類がある場合，本を購入するという目的には 2 種類のプロセスが存在するということである．また，本以外に DVD の販売も行っている場合，本の購入に DVD 購入のためのプロセスは関係ないため，DVD 購入のための入力をユーザに求めることは意味がない．このようにユーザのリクエストを達成するためのプロセスが 2 つ以上存在する場合，必要なプロセスだけの抽出，すなわちプラニングが必要になる．

つまり，本論文では “本を購入したい” という動機で端末の “本の購入” というボタンをクリックすることや “地図を見たい” という動機で端末の “地図” というボタンを押すなど，インタラクションのきっかけになるユーザのメッセージ送信によってサービスリクエストが送られ，適切な引数情報（引数名やデータ型）のみをユーザへ問い合わせるエージェントを自動生成するために以下のような処理を行う．まず，ユーザからのサービスリクエストをトリガとして BPEL ファイルを入手，解析し，アクションオペレータを生成する．プラニングでサービスを提供するために必要なアクションオペレータの列（プラン）を生成し，ユーザの入力する引数情報を抽出する．一方でインタラクションプロトコル (IP) を生成し，エージェントを生成する．エージェントは IP で定められた挙動の通りにユーザへ引数情報を問い合わせ，処理を代行する．

なお，本システムにおけるプラニングドメインは次のタプルとして定義される．

$$D =< S, Act, Out, Tr >$$

ここで，S はプラニングによって構成される状態遷移システムにおける状態の集合であり，具体的には BPEL において定義されているメッセージ系アクティビティ invoke，receive，reply，プロセス制御系アクティビティ while，switch，pick，terminate，およびデータ制御系アクティビティ assign からなる．また，Act は個々のアクティビティに含まれる状態遷移を引き起こすアクションの集合であり，Out はアクションを実行した結果，出力されうるメッセージ型の集合である．遷移関数は $Tr : S \times Act \times Out \to S$ として定義され，$Tr(s_i, a, o_a) = s_j$ は出力 o_a を持つアクション a によって状態 s_i から s_j へ遷移することを表す．なお，ここでアクション a は決定的 (deterministic) であり，$\exists\ s' \quad \forall\ o \in Out \quad Tr(s, a, o) = s'$ と表される．

次節以降では以下の手順で詳細を説明をする．

(1) ターゲットとなる Web サービスの BPEL ファイルを入手する．

(2) BPEL からユーザ側以外とのインタラクションな

ど余計な処理を除く．そして，ユーザ側と BPEL エンジン間の通信や BPEL エンジンの内部処理に関わるアクティビティやイベントを中心にプラニングのためのアクションオペレータを生成する（2･1 節参照）．

(3) ユーザの意図を達成できるようにプラナーを用いて行うべきアクティビティの列であるプランを生成する．そして，IP にユーザとのインタラクション部分を追加してユーザとエージェント間の IP を生成する（2･2 節参照）．

2･1 BPEL の解析からアクションオペレータの生成

BPEL にはユーザが行う事象が記述されているため，これを解析してプラニングの材料であるアクションオペレータを生成することでユーザの代理エージェントを生成することが可能である．アクションオペレータを生成する手順は以下の通りである．

(1) BPEL で登場する “partner” という概念の理解とそれに応じた処理 (2･1･1 節参照)
(2) エージェント用の処理へ置き換え (2･1･2 節参照)
(3) アクションオペレータの生成 (2･1･3 節参照)

それぞれの処理について説明する．

§1 BPEL に現れる属性 “partner” に対する処理

外部とのインタラクションを行う BPEL のアクティビティやイベントには “partner” という属性があり，この値によってどの通信相手に対するアクティビティなのかを特定できる．そして，Web サービスの基本的な考え方はリクエストに対してサービスを提供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信する partner がユーザ側であることがわかる．

ユーザ側以外と BPEL エンジンのインタラクションはユーザにはどうすることもできない．そこで，partner 属性がユーザでない invoke，receive，reply アクティビティと onMessage イベントはデータの関連性のみを考慮し，アクションオペレータを生成する際には扱わない．

§2 エージェント用の処理への置き換え

2･1･1 節で紹介した方法で余分なアクティビティやイベントを消去した上で，BPEL エンジンが行うインタラクションからエージェントが行うインタラクションを生成する．具体的には，ユーザ－ BPEL エンジン間のインタラクションはエージェント－ BPEL エンジン間のインタラクションになるため，BPEL エンジンが受信／送信するメッセージはエージェントが送信／受信するメッセージとなる．assign についてはデータ操作のタイミングが大事なため，BPEL エンジンのままエージェントでも利用する．pick はメッセージの種類によって分岐する switch のようなアクティビティである． switch, case に対して pick, onMessage というアクティビティおよびエレメント／イベントが割り当てられている．onMessage の

```
<onMessage operation="a"> ... </onMessage>
```

```
<onMessage operation="b"> ... </onMessage>
```

という形は受け取り側の問題でメッセージを送るエージェント側では

```
変数 = a or b
<invoke operation="変数">
```

というように onMessage を意識する必要はないため，実際には invoke を行えばよいが，BPEL エンジン内部で行われる処理を解釈するためにこのままの形で実現をする．

§3 アクションオペレータの生成

アクションオペレータの基本的な生成方法として，BPEL の 1 つの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等を 1 つのアクションオペレータとして定義する．BPEL はアクティビティ単位で形成されているため，粒度がちょうど良く，コードが見やすくなり，どのような順番でアクションオペレータを実行しているのかがわかりやすくなる．さらに複数のアクションオペレータから選択するような場面では同じ名前のアクションオペレータを複数用意する．一般的なプログラミング言語では同名を使うことはエラーの原因となるが，ここでは複数用意することで，プラニングを行う際，すべてのパターンを生成させることができる．もし複数の結果が得られた場合，その中から選択することができ，本研究ではインタラクションが最小回数で済むプランを選択する．以下ではどのようにアクションオペレータを生成するかを紹介する．

基本的にアクティビティはアクティビティ名をアクションオペレータ名とする．onMessage は並列に同じ名前が並んでしまうため，一意である operation 名を用い，図 3 はその例である．内部プロセスの制御を行う sequence, flow, while，switch，pick，wait，terminate はそれぞれ次のようにしてアクションオペレータを生成する．sequence は前後条件を繋げてシーケンスとなるようにする．flow はサーバ側が複数のメッセージタイプを同時に受け取りたいなどの理由で設けるもので，ユーザ側が使うものではない．while は BPEL エンジンのプロセス内容を知るために必要なため，ループになるようにアクションオペレータを構成する．同じく terminate も処理が終了するように構成する．wait は BPEL エンジンが待つ必要があっても，エージェントは待つ必要はないため無視してよい．図 4 は while の中に invoke と receive が入った例である． なお，同じアクティビティが複数現れたときにアクションオペレータ名だけは一意になるようにしなければならない．そのため，アクションオペレータ名の最後に数字を付けて一意にする．

2･2 ユーザとエージェント間の IP の生成

ここではユーザからリクエストされた処理をするために必要な手順をプラニングし，最短経路となる部分フローを抽出する．また，その抽出過程において，変数間の単一化 (unification) を行い事前に入力が必要な情報を洗い

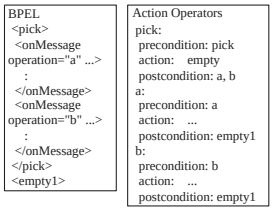

図 3 アクションオペレータ生成の例 pick

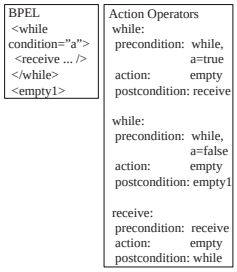

図 4 アクションオペレータ生成の例 while

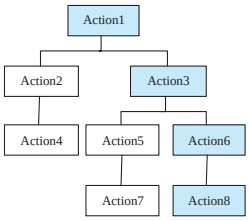

図 5 アクションオペレータの構造

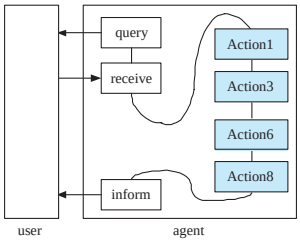

図 6 IP の構造

出す．その手順は以下の通りである．

(1) プラナーによりリクエストに応じられる最短経路となるプランを生成する．
(2) プランからユーザに問い合わせるべき引数情報を洗い出す．
(3) プランの最初に引数情報を問い合わせる query を追加する．
(4) query の直後にユーザ側から引数情報を受ける受信処理を追加する．
(5) プランの最後にユーザ側へ返信する inform を追加する．

例として図 5 を考える．この図は前節の処理を行った結果であるアクションオペレータの構造を木構造で表現したものである．ここで，ユーザから Action8 の機能（サービス）を求められた場合，プラナーにより Action8 を実行するためには Action1 → 3 → 6 → 8 という順番で実行すれば良いというプランが生成される．このプランの前後に上で述べた (3),(4),(5) の処理を追加すると図 6 のようになる．

このようにしてエージェントが行うべきインタラクションを決めることができる．また，BPEL エンジンとインタラクションを行うにあたって必要な入力情報を WSDL のメッセージタイプから洗い出し，最初に query としてユーザへ送ることでユーザはまとめて引数入力を行うことができ，少ない関与で処理を行うことができる．しかし，引数が多くて一度に送るのが面倒である場合や，引数が必要となる直前まで値は決めかねる場合などがある．そのために，最初の query で処理に必要な値はすべて尋ねるが，ユーザは必要に応じて空欄のままリクエストを行えるようにする．空欄の部分は実際に値が必要になる直前に再び聞き返す．その結果，ユーザは必要な情報を一度にまとめて送るか，少ない量を複数回に分けて送るかのトレードオフを選択できるようになる．

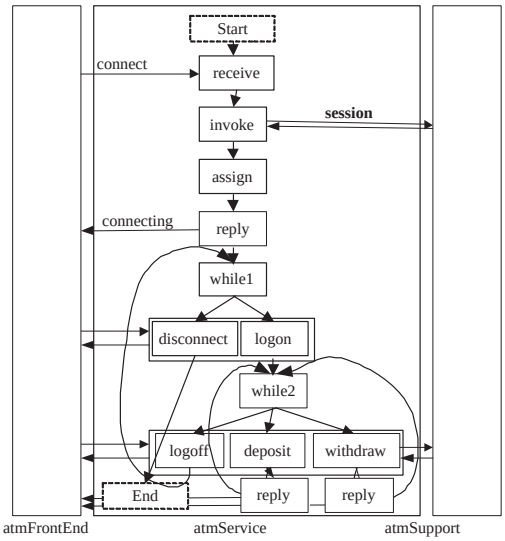

図 7 ATM サービスのアクティビティ構造

3. ATM サービスの例

IBM の alphaWorks より BPEL エンジン BPWS4J[?] が提供されている．その BPWS4J のサンプルとして紹介されている ATM(Automatic Teller Machine) サービスを具体的な例として取り上げる．まずこのサンプルの構造について説明する．

これは図 7 のような構造をしている．ATM サービスを行う BPEL プロセスが真中に位置し，振込みや引出しなどをリクエストするエンドユーザが左の atmFrontEnd にあたる．サービスをサポートする ATM サポートが右の atmSupport である．

このサービスは，ユーザが ATM サービスの利用をリクエストすることによってプロセスが atmSupport に対してコネクションを繋ぎ，その結果を atmFrontEnd にいるユーザへ返す．コネクションがうまくいけば while1 へ入る．ここでユーザが disconnect メッセージを送らない限り正常な状態で while1 から抜けることはない．ユーザが logon メッセージを送ると while2 へ入る．ここでユーザが logoff メッセージを送らない限り正常な状態で while2 から抜けることはない．ユーザが deposit メッセージを送る場合，引数として入力した金額分を預金する．一方，withdraw の場合は引数の金額分を引き落とす．これらの目的が果たされたら，logoff メッセージを送り while2 を抜け，while1 へ戻る．次に disconnect メッセージを送ることで while1 を抜け処理を正常終了することができる．

この構造からアクションオペレータを生成する．まず 2･1 節で述べたようにユーザ側以外とのインタラクショ

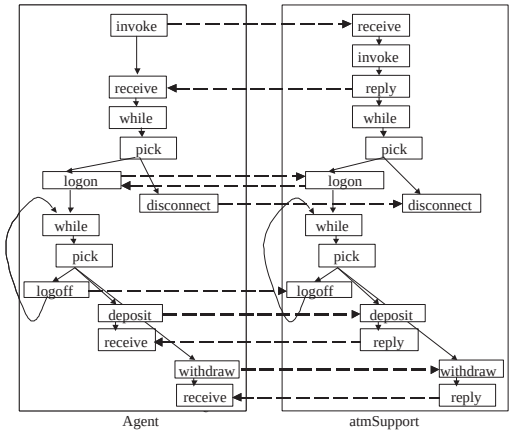

図 8 エージェント− BPEL エンジン間のインタラクション

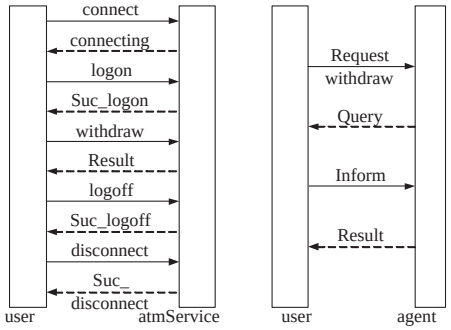

図 9 本手法適用によるインタラクションの減少

ンは考えない．具体的には最初の receive の次の処理である invoke は atmSupport とのインタラクションであるため省略することができる．そして，ユーザの代わりをするエージェントのアクションオペレータとして receive と invoke, reply を入れ替える．その結果，図 8 の左図ようなエージェントの処理を生成できる．

ユーザが “引き出し (withdraw)”をリクエストしている場合，プラニングによりリクエストを満たすプランが生成される．そして，その実行に必要な引数情報を WSDL より収集し，必要な引数であるカスタマー名，パスワード，引き出す金額をユーザにあらかじめ入力してもらうために query を代入し，その結果を受け取る receive を query の直後に代入する．そして，最後には結果をユーザへ返すための inform を追加する．このようにしてユーザとエージェント間の IP を生成することができる．

提供されるまま利用する場合には，図 9 の左図のように 5 回の入力を行わなければならないが，エージェントを使うことによって右図のように 2 回の入力で処理を実行することができる．

4. 評 価 と 考 察

本研究ではユビキタスユーザのために入力回数の軽減やネットワーク使用度の軽減を目的としたため，これらに関わる以下の 3 点について ATM の例を使用して評価を行った．

- ユーザの入力回数
- サービスの実行時間
- データ転送量

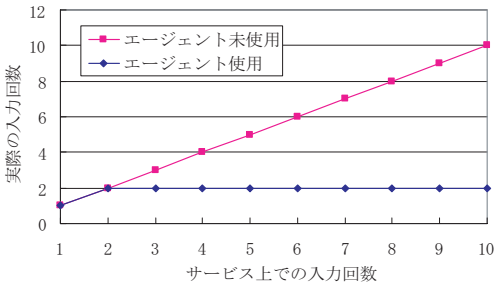

図 10 エージェントによる入力回数の違い

4·1 実 験 環 境

サーバ側は図 2 のようにミドルエージェント，BPEL エンジン，Web サービスを持ち，スペックは次のようである．OS は Linux version 2.4.20-24.9，CPU は Intel Xeon 2.80Ghz，メモリ 1GB，ネットワーク環境は FTTH，J2SE はバージョン 1.4.0_03，J2EE は 1.3.1，TOMCAT は 4.1，BPWS4J は 2.0，エージェントシステムは Bee-gent β 2.0[Bee-gent].

本研究では，ユーザ側はエージェントだけを持つ携帯電話のようなリソースが極めて少ない端末で行うことを前提としているが，実験の都合上，ノートパソコンに PHS を繋げて代用した．スペックは以下のようである．OS は WindowsXP，CPU は Intel Celeron 1.33GHz，メモリ 256MB，ネットワーク環境は AirH” 32kbps，J2SE は 1.4.0_03，エージェントシステムは Bee-gent β 2.0 を用いた．

4·2 ユーザの入力回数

入力の手間は内容や文字数などによって異なるが，入力する引数の内容はエージェントの使用不使用に関わらず同じであるため，ここでは入力回数を計測とする．エージェントを使用しない場合は当然のことながらサービスの入力回数に比例して実際に行う入力回数は増える．しかし，エージェントを使用する場合，引数が必要ない場合はサービス名のみの入力 1 回で済み，引数が必要な場合はサービス名とすべての引数情報という 2 回で済む．これを図にしたものが図 10 である．

入力回数が 1，2 回の時は同じだが，3 回以上になると差が出てくる．もちろん入力回数が少なくなる代わりに 1 回辺りの入力量は増える．

表 1 エージェント使用／未使用時の実行時間（単位：msec）

	エージェント使用	エージェント未使用
実行時間	7304	13898

表 2 主な内部処理にかかる時間（単位：msec）

	BPEL 解析	スレッド生成
実行時間	201	304

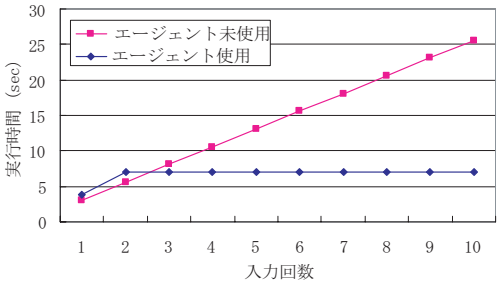

図 11 本来の入力回数と実行時間の関係

表 3 ACL（単位：byte）

送信	受信
Request 255	Query 198
Inform1 222	Inform2 228
合計 477	合計 426

表 4 SOAP（単位：byte）

	送信	受信
connect	418	477
logon	513	430
deposit	513	481
logoff	472	432
dis-connect	467	440
合計	2383	2260

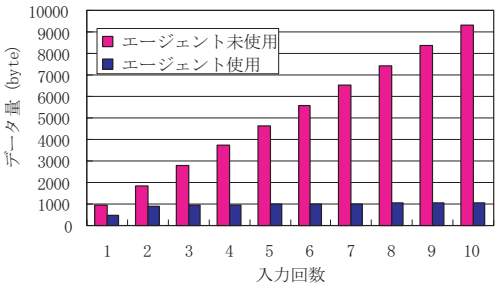

図 12 入力回数とデータ量の関係

4·3 サービスの実行時間

ATM の例を用いてエージェント使用時と未使用時で，ユーザがリクエストを入力してから実際にサービスが提供されるまでにかかった時間を計測した．

§ 1 実験結果

実行時間を 6 回計測して平均を取った結果が表 1 である．この結果は通信状況による影響が大きいため，通信時間を除いた主な内部処理の実行時間だけを計測して表 2 とした．表 1 と表 2 より Web サービスで提供されている本来の入力回数と実行時間の関係をグラフにしたものが図 11 である．

§ 2 実験の考察

直に SOAP を送信する場合と異なり，本研究の手法では BPEL の解析や，エージェントのためのスレッド生成という処理が加えられる．しかし，表 2 より BPEL の解析とスレッドの生成時間の合計は高々0.5 秒である．全体の実行時間が 7.3 秒であることを考えると無視できるほどの時間である．

また，エージェント使用時と未使用時の実行時間を比べると 7.3 秒と 13.9 秒でエージェント使用時では約 52%になっていることがわかる．ここではユーザによる入力時間を 0 秒としているが，入力回数が 2 回と 5 回ということを考慮するとさらに差は広がる．例えば入力 1 回に 5 秒掛かる場合はエージェント使用時が 17.3 秒，未使用時は 38.1 秒となり，約 45%の実行時間で済む．

そして，それぞれの内外部処理に掛かった時間を計測した結果，一般的な入力回数と実行時間の関係を求めたものが図 11 である．入力回数が 1，2 回の場合は通信以外の処理をしていない分，エージェントを使用しない方がわずかに高速である．しかし，入力回数が 3 回以上になるとエージェントを使用した方が早くなり，その後は入力回数に比例して差は広がる．

4·4 データ転送量

本節ではサービスを利用するためにネットワーク上を流れるデータ量がエージェント使用未使用でどのように変わるかを計測した．

エージェントを用いる場合，ネットワーク上を流れるデータは Bee-gent で用いられている ACL に HTTP が付加されたものである．一方，エージェントを用いない場合は SOAP ボディに SOAP ヘッダと HTTP が付加されたものである．HTTP は本研究で扱っているレイヤではないため，ATM の例においてネットワーク上を流れる ACL と SOAP ボディ＋ヘッダの容量を比較した．

§ 1 実験結果

ACL のデータ量が表 3，SOAP のデータ量が表 4 であり，

この ATM の例から一般的な場合を考える．まず ACL で，入力が 1 回で済む場合は Request を送って結果を含んだ Inform(表 3 では Inform2) を受けるだけなので 225+228=483 バイトとなる．入力が 2 回（つまり引数がある場合）となると表 3 のような 2 回入力の形になる．入力が 3 回以上になっても 2 回の時と変わらないが，入力するデータ量が増える分，1 回増えるごとに 30 バイト増やした．

一方 SOAP では，入力回数だけデータ量も増えるため，表 4 で 1 インタラクションに必要な平均データ量 929 バイトを基準としてインタラクションの回数だけ倍にした．このことで得られた結果を図にしたものが図 12 である．

§2 実験の考察

ACL も SOAP も XML 形式であるが，SOAP にはヘッダが付加されるため 1 メッセージ辺りのデータ量が増え，ACL は SOAP の約 50%になっている（表 3 と表 4）．さらに通信回数が少ないことにより送受信それぞれのメッセージ総量は，

$$477/2383 = 0.20 \qquad 426/2260 = 0.19$$

と 20%以下になっている．形式の違いにより 50%になった点を除いてもインタラクションの減少により約 40%のデータ量になっていることがわかる．この例から一般的なインタラクションの回数とデータ量の関係を求めた図 12 を見てもデータ量の差は一目瞭然である．

4·5 BPEL 仕様への適応性と正当性

定義されている全てのアクティビティとイベントについて考察した結果を以下にまとめる．

- 適応可能

  先の章で挙げたように以下のアクティビティは適応可能である．

  - メッセージ送受信系アクティビティ invoke, receive, reply

    エージェント用に送受信を入れ替える．
  - データ処理系アクティビティ assign

    BPEL エンジンでの処理と同じようにデータ処理を行う．
  - プロセス制御系アクティビティ sequence, flow, while, switch, pick, wait, terminate

    wait は BPEL エンジンだけでエージェントも wait する必要はなく，他のアクティビティに関しては本来の処理と同じようにアクションオペレータで実現する．但し，while, switch, pick は以下の場合を除く．
- 現状では適応できていないもの
  - エラー処理等のイベント compensation handlers, fault handlers, event handlers

    これらのイベントはいつ起こるか分からないため，BPEL を解析する時点で予想することができない．そのため，現状では適応できていないが，イベントが発生した時に再プラニングを行ったり，別スレッドを用意するなどして適応することは可能である．また，これらで対処できない状況では BPEL エンジンからのメッセージに手を加えずユーザへ転送し，逆向きのメッセージも同様に転送することで実行が強制終了するようなことにはならない．
  - 実行するまでループの回数がわからない while 文や実行するまで分からない変数の値による switch や pick 文

    while の中にユーザへのメッセージ送受信があった場合に，何回送受信を行うのかがわからなくなってしまう．また，Web サービス側の意図によって挙動が変わる switch や pick では実行するまで挙動がわからなく，事前にすべての引数情報をユーザから収集することは困難であり現状では適応できていない．この場合も再プラニングをすることで適応することは可能である．また，この方法で対処できない状況では上記と同様に転送することで強制終了するようなことにはならない．

なお，この方法は BPEL ファイルから必要なアクティビティを抽出することにより，エージェントの挙動を決定するが，BPEL エンジン内で行われる処理を変えるわけではないため，Web サービス全体の振る舞いが変わるようなことはない．

4·6 ユーザインタフェース

サービス名を最初にリクエストし，サービスに必要な引数を後から入力するという手順はどのサービスでも同じである．そのため，一度開発したインターフェース用のソフトウェア（図 2 のユーザエージェント）を他のサービスにも再利用でき，サービスごとに開発する必要はない．また，Web サービスにおいて外部に公開されているアクティビティやイベントにはサービス内容についての自然言語によるコメントが記述されているのが通例である．そのため，ユーザのインタフェースに大きく関係する “withdraw” や “deposit” などはそのコメントを利用し，ユーザへ意味を伝えることができるだろう．

5. 関 連 研 究

[Sashima 03] はユビキタス環境を考慮した Web サービスの提供を目的とし，その手段として柔軟なエージェントを仲介させるアーキテクチャを提案している．ユーザが持つユビキタス端末の位置や時間などの情報をエージェントが随時取得し，条件があった時に自律的に適した Web サービスを利用するというものである．この研究ではリクエストするとすぐに結果が返ってくるような簡単な Web サービスを対象にしている．一方，本研究ではインタラクションが多い複雑な Web サービスを対象にして，そのインタラクションの回数を減らしているという点で異なる．

[WSDLTool 03] はエージェントと Web サービスの連携を目的とし，その手段としてプロキシの設置を提案し，実装している．WSDL を解析することにより ACL と SOAP を相互変換できるエージェントを半自動で生成するというものである．しかし，この研究はメッセージを 1 対 1 で変換するものであり，本研究のようにインタラクションの回数を減らすようなことはしていない．

[Petrone 03] はユーザとサーバの円滑なインタラクションを目的としている．一般的にインタラクションを行う際，相手側のプロセスの内部で行われている処理はわからないが，この研究ではすべての処理内容をサーバ側が

管理し，ユーザ側は送られてくるメッセージに答えるだけにするという方法を提案している．この研究では，ユーザ側に BPEL エンジンを置くことを回避しているが，本研究のようにユーザが行うインタラクションの回数自体を減らすようなことは考慮されていない．

[Kim 03] は事実上の標準となりつつある BPEL を使って企業間の交渉システムを実現することを目的としている．これまで特別なアプリケーションを導入することで実現してきた交渉システムを，標準の BPEL で可能とした為に，実装や導入コストを大きく抑えることができている．この研究では，交渉システムを作る際のコストを減らすものであり，本研究のように既に作られた BPEL を利用する際に通信回数を減らすということとは異なる．

6. むすび

本論文ではユビキタスユーザが Web サービスを利用する際にネックとなる入力のしにくさ，ネットワークの不安定さ，通信コストの高さを軽減するためのエージェントを提案した．ユーザへの複雑なインタラクションが存在する Web サービスに対して，エージェントを仲介させ，事前にユーザからすべての引数情報を受け取ることで，ユーザの代わりにインタラクションを行い，ユーザの負担を軽減するというものである．

そのエージェントを作るために BPEL を解析してアクションオペレータを生成し，ユーザからサービスリクエストを達成するために必要なアクションオペレータの列（プラン）をプラナーにより生成し，それに従って，入力情報を洗い出し，ユーザとエージェント間の IP 及びエージェントを生成する．

このエージェントによってインタラクションに対するユーザの関与を減らすことができ，煩雑さを減少させるだけでなく付加価値として実行時間の現象や通信量の削減なども得ることができた．そして，将来的には，今回適応できなかったイベントドリブン処理や実行するまで挙動がわからない部分への適応性，コストなどを用いた最適プランの抽出，他のエージェントシステムとの協調による負荷分散，モバイルユーザのためにユビキタス情報やプロファイルなどによるサポートを強化していきたい．

◇ 参 考 文 献 ◇

[Bee-gent] Bee-gent, : Bonding and Encapsulation Enhancement aGENT, http://www2.toshiba.co.jp/beegent/

[BPEL 03] BPEL, : BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems.: Business Process Execution Language for Web Services, Version 1.1 (5 May 2003)

[Kim 03] Kim, J. B., Segev, A., Patankar, A., and Cho, M. G.: Web Services and BPEL4WS for Dynamic eBusiness Negotiation Processes, *Proceedings of the 1st International Conference on Web Services (ICWS'03)* (2003)

[Petrone 03] Petrone, G.: Managing flexible interaction with Web Services, *Proc. Workshop on Web Services and Agent-based Engineering (WSABE 2003)*, pp. 41–47 (July 2003)

[Sashima 03] Sashima, A., Kurumatani, K., and Izumi, N.: Location-mediated service coordination in ubiquitous computing, *In Proc. of the Third International Workshop on Ontologies in Agent Systems (OAS-03) AAMAS2003*, pp. 39–46 (2003)

[WSDLTool 03] WSDLTool, : System Development Department, Computer and Automation Research Institute(SZTAKI).: Tool for Deploying Web Service Wrapper Agents, *In Proc. of 2003 Agentcities Aent Technology Exhibition*, pp. 44–46 (2003)

〔担当委員：北村泰彦〕

2003 年 11 月 4 日　受理

著 者 紹 介

寺崎　達也

2002 年電気通信大学電気通信学部情報工学科卒業．2004 年電気通信大学大学院情報システム研究科情報システム設計学専攻修士課程了．同年 (株) 日立システムアンドサービス入社，現在に至る．主として Web サービス/XML の研究，開発に従事．

中井戸　健至

2002 年電気通信大学電気通信学部情報工学科卒業．2004 年電気通信大学大学院情報システム研究科情報システム設計学専攻修士課程了．同年 (株) 東芝入社，現在に至る．主としてエージェント，Web サービス，J2EE の研究に従事．

川村　隆浩(正会員)

1992 年早稲田大学理工学部電気工学科卒業．1994 年同大大学院理工学研究科電気工学専攻修士課程了．同年 (株) 東芝入社．現在同社研究開発センター知識メディアラボラトリー研究主務，工学博士．2001-2002 年米国カーネギ・メロン大学ロボット工学研究所客員研究員，2003 年より現在電気通信大学大学院情報システム学研究科客員助教授．主として分散 AI，マルチエージェントシステムの研究・開発に従事．情報処理学会，人工知能学会各会員．

大須賀　昭彦

1981 年上智大学理工学部数学科卒業．同年 (株) 東芝入社．1985〜1989 年 (財) 新世代コンピュータ技術開発機構 (ICOT) に出向．現在 (株) 東芝研究開発センター知識メディアラボラトリー主任研究員．工学博士 (早稲田大学)．電気通信大学大学院客員教授ならびに大阪大学大学院非常勤講師兼任．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．情報処理学会，電子情報通信学会，日本ソフトウェア科学会，IEEE CS 各会員．

前川　　守

1965 年京都大学工学部数理工学科卒.Ph.D.．同年 (株) 東芝入社．アイオワ大学，東大理学部情報科学科助教授等を経て，1993 年電通大大学院情報システム学研究科情報システム設計学専攻教授，現在に至る．主としてオペレーティングシステム，分散システム，ソフトウェア開発環境，マルチメディア，GIS 等の研究に従事．