# Webサービス連携のためのモバイルエージェント動作記述

石川冬樹[†] 田原康之[††]
吉岡信和[††] 本位田真一[†,††]

連携プロセスを手軽に，プラットフォームに依存しない形で記述するBPEL（Business Process Execution Language for Web Services）のように，分散コンポーネントの動的な連携のためのWebサービス技術への取り組みがさかんに行われている．Webサービス連携は今後，無線接続されたモバイルデバイスで構成されるようなパーベイシブネットワーク等様々な環境に適用されていくと考えられる．しかしそのような環境においては，比較的低速で不安定な無線通信路等の資源制約の問題に対処する必要がある．本研究ではこの問題に対しモバイルエージェント技術を適用し，Webサービス連携を行うモバイルエージェントの動作記述のための枠組みを提案する．この枠組みでは，連携ロジックをBPELを用いて記述し，それに対し移動およびクローニングというモバイルエージェントの物理的なビヘイビアをルール記述として付加する．この分離により，BPEL記述を変更することなしに環境条件に応じて物理的な振舞いを追加したり変更したりすることができる．本論文では特に，形式言語Mobile Ambientsを用いてこの枠組みの意味定義を行い，またBPELの意味論が保存されていることを示す．

# Behavior Descriptions of Mobile Agents for Web Services Integration

Fuyuki Ishikawa,[†] Yasuyuki Tahara,[††] Nobukazu Yoshioka[††]
and Shinichi Honiden[†,††]

Research on the Web Service technologies for dynamic integration of distributed components has recently commenced, including BPEL (Business Process Execution Language for Web Services) for specifying an integration process easily and platform-independently. Web Services integration is to be applied in various environments, for example, pervasive networks with wireless mobile devices. However, in such environments it is necessary to deal with constraints in resources, such as the relative narrowness and instability of wireless connections. This work adopts the mobile agent technology in response to this problem and presents a framework for description of agents' behaviors for integration. In this framework, the integration logic is described using BPEL, and physical behaviors of mobile agents, including migration and cloning, are added to the BPEL description as simple rules. This separation makes it possible to add or change physical behaviors according to environmental conditions without modification of the BPEL description. This paper especially concentrates on formal definition of the semantics of our framework using a formal language, Mobile Ambients, and proves preservation of the BPEL semantics.

## 1. まえがき

分散コンポーネントの動的な連携のためのWebサービス技術がさかんに研究されている[1]．Webサービスにおいては，XMLによる標準プロトコルを用いることによりサービス間の相互運用性を高め，実行時に互いを発見し連携するような疎結合システムの構成が目指されている．現在ではメッセージング等の基本プロトコルに加えて，Business Process Execution Language for Web Services（BPEL）[2]のような連携プロセス記述言語も提案されている．BPELを用いて他のサービスやクライアントとの相互作用の順序や条件等を記述することにより，それらを連携させて新たなサービスを構成するプロセスを実現することができる．

Webサービスは相互運用性およびサービスの動的な発見，連携のための基盤を提供するため，無線接続されたモバイルデバイスで構成されるようなパーベイシブネットワーク等，インターネット以外の様々な環境においても有効であると考えられる．また実際にモバイル端末上での実装の試みも始まっている[3]．しかし，

† 東京大学
The University of Tokyo
†† 国立情報学研究所
National Institute of Informatics

そのような環境においては比較的低速で不安定な無線通信路等の資源制約が存在し，連携の実現のためにはそれらに対処する枠組みが必要となる．BPEL のような従来の Web サービス技術は資源の豊富なサーバでの実行を仮定したもので，資源制約の存在する環境にはそのまま適用できない．典型的な例として，分散したサービスコンポーネント間でマルチメディアデータを交換することにより情報収集を実現するようなサービス連携は，比較的低速，不安定な無線通信路で構成されるパーベイシブネットワークにおいてはその実現が難しくなってしまう．また，インターネット上においても，科学技術計算等において大量のデータの交換や処理を必要とする場合には，同様の問題が生じる．

我々はこの問題に対し，モバイルエージェント技術[4),5)] の Web サービス連携への適用を考えている．モバイルエージェントは，実行状態を保持してのホスト間のコードの移動およびクローニングという物理的なビヘイビアを可能とする．これらを通常のリモート通信に組み合わせることにより，通信量削減等の要求に応じてタスクの様々な実行形態を実現することができる．モバイルエージェントによる連携を実現する際には，適用環境の変化に応じて連携ロジックを保持したまま物理的なビヘイビアのみを変更できることが望ましい．文献 6) において，我々はこの要件を考慮して Web サービス連携のためのモバイルエージェントシステムを提案している．このシステムでは BPEL の前身である WSFL[7)] を用いて連携フロー記述を記述し，フロー中のアクティビティ（タスク）の実行形態についてのルール記述として物理的な振舞いを付加する．

本論文においては，BPEL を対象とした現在のシステムにおける動作記述の枠組みを示し，特に Mobile Ambients[8)] による形式的な意味定義を行う．Mobile Ambient は，π-calculus[9)] と同等の並列プロセスの表現能力に加え，名前と境界を持つ ambient という概念によって内包関係を表現し移動性を表現することができる形式言語である．本論文では，BPEL における各タスク（アクティビティ）を，Mobile Ambients における通信プロセスとして表現し，エージェントとホストの内包関係，およびアクティビティとエージェントの内包関係を考えることにより，移動およびクローニングの意味を表現する．

本論文では，まず 2 章で本研究のモバイルエージェントによる連携のモデルについて述べる．その後に，ビヘイビア記述の構造を示し（3 章），その意味定義を行う（4，5 章）．さらにこの意味定義に基づき，本研究における物理的なビヘイビアの導入が BPEL の意味論を崩さないことを示す（6 章）．最後に関連研究を通して考察を行い，現在取り組んでいる課題について述べる（7，8 章）．

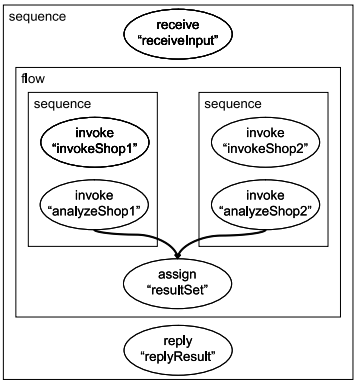


**図 1** BPEL による連携プロセスの例
Fig. 1 Example of BPEL integration process.

## 2. モバイルエージェントによる Web サービス連携モデル

本章では我々の提案している，モバイルエージェントによる Web サービス連携のモデル[6)] の概要を示す．

### 2.1 BPEL

BPEL（Business Process Exectuion Language for Web Services）は，他のサービスを連携させて新たなサービスを提供するための実行可能なプロセスを記述する XML 言語である．**図 1** はそのようなプロセスの例である．このプロセスでは，呼び出し元からのリクエストを受け取り（receive），2 つの店舗情報提供サービスからデータを取得するとともに，それぞれのデータに対して検索/解析サービスを起動した後（invoke），結果を変数にセットして（assign）呼び出し元に結果を返している（reply）．これらの個々のタスクの単位はアクティビティと呼ばれ，たとえば invoke アクティビティは次のような形で記述される．

```
<invoke partner="shopService1"
  portType="shopPT" operation="getItemData"
  inputVariable="query" outputVariable="result1">
```

partner は WSDL☆（の拡張）により記述される他のサービスやクライアントであり，invoke ではそのサービスオペレーションの呼び出しを記述する．また，その入出力等のデータは変数（variable）を通して管理される．

☆ WSDL: Web Services Description Language[10)] は Web サービスのインタフェースを記述する言語である．

```
<process>
  <!-- パートナーや変数，例外処理等の宣言... -->
  <sequence>
    <receive name="receiveInput" />
    <flow>
      <links>
        <link name="l1">
        <link name="l2">
      </links>
      <sequence>
        <invoke name="invokeShop1" />
        <invoke name="analyzeShop1">
          <source linkName="l1">
        </invoke>
      </sequence>
      <sequence>
        <invoke name="invokeShop2" />
        <invoke name="analyzeShop2">
          <source linkName="l2">
        </invoke>
      </sequence>
      <assign name="resultSet"
          joinCondition="l1 or l2">
        <target linkName="l1">
        <target linkName="l2">
      </assign>
    </flow>
    <reply name="replyResult"/>
  </sequence>
</process>
```

**図 2** BPEL の制御構造
Fig. 2 BPEL control structure.

さらに sequence 等の構造アクティビティを用いてこれらの制御構造を記述することによりプロセスを記述する．**図 2** はこのプロセスの制御構造の記述の全体像を示したものである．ここでは順序実行を表す sequence と，グラフ指向の記述を表す flow を組み合わせた実行制御が可能である．flow 内では，実行グラフの有向辺を表す link を用いて実行順序制約を記述し，joinCondition により並列パスのジョイン条件を記述している．このように，sequence，while，switch 等の構造的な制御と，link とその遷移条件やジョイン条件によるグラフ指向の制御とを組み合わせて実行制御を記述することができる．また，例外処理や完了した処理の明示的な取り消し処理の記述も可能である．

### 2.2 モバイルエージェントの適用

前節で述べたように BPEL を用いて連携プロセスを記述することにより，サービスを動的に連携させ新たなサービスを実現することができる．しかし，今後様々な環境における様々な連携の実現を考えた場合，計算/通信資源の制約に対処する必要が生じることが考えられる．たとえば分散したサービス間でマルチメディアデータを交換し情報収集するような連携を考えてみると，比較的低速，不安定な無線通信路を用いるパーベイシブネットワークにそれをそのまま適用することはできない．

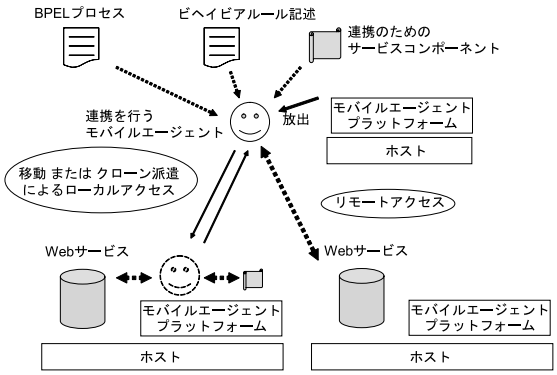


**図 3** モバイルエージェントによる Web サービス連携モデル
Fig. 3 Web services integration model with mobile agent.

我々はこの問題に対しモバイルエージェント技術の適用を考え，そのための取り組みを行っている[6]．モバイルエージェントは，実行状態を保持してのホスト間のコードの移動およびクローニングという物理的なビヘイビアを可能とする．これらを通常のリモート通信に組み合わせることにより，通信量削減等の要求に応じてタスクの様々な実行形態を実現することができる．モバイルエージェントによる連携を実現する際には，適用環境の変化に応じて連携ロジックを保持したまま物理的なビヘイビアのみを変更できることが望ましい．そこで本研究においては，連携ロジックの記述に関して最も有力な仕様である BPEL をそのまま利用し，それに対して物理的なビヘイビアのみを付加，変更可能な枠組みを目指している．

**図 3** に，我々の提案しているモバイルエージェントによる Web サービス連携のモデルを示す．このモデルにおいてはモバイルエージェントは以下の 3 つの記述により構成される．

**BPEL による連携プロセス記述** エージェントはこの記述に従い，他のサービスやクライアントと相互作用する．

**持ち運ぶサービス** エージェントは BPEL によりアクセスされるサービスコンポーネントを持ち運び，ローカルサービスとしてアクセスする☆．

**ビヘイビアルール記述** エージェントはこの記述に従い，移動もしくはクローニングという物理的なビヘイビアをとる．

☆ 現在この機能は WSDL によりインタフェースを記述した Java クラスに限られている．

```
 1 <behavior>
 2   <rules>
 3     <migrate block="invokeShop1 analyzeShop1">
 4       <target> <local /> </target>
 5       <retry time="30" />
 6     </migrate>
 7     <clone block="invokeShop2 analyzeShop2">
 8       <migrate block="invokeShop2 analyzeShop2">
 9         <target> <local /> </target>
10       </migrate>
11     </clone>
12     <handle>
13       <bpws:throw faultName="taskFailed" />
14     </handle>
15   </rules>
16 </behavior>
```

**図 4** ビヘイビアルール記述の例
Fig. 4 Example of behavior rule descriptions.

モバイルエージェントは BPEL プロセスを解釈実行するが，その際にビヘイビアルール記述に従い，実行ホストを変えたり，クローンを生成して並列タスクを違うホスト上で実行させたりする．本論文においては，このような物理的なビヘイビアの記述方法を示しその意味定義を考える．また，BPEL プロセスを媒介として相互作用させるサービスコンポーネントを持ち運ぶことにより，サービスのローカルでの相互作用を実現する．たとえば，データ提供サービスのローカル（稼動ホスト）に移動し，そこで得たデータを持ち込んだ検索/解析サービスに与え，必要な結果のみ持ち帰るという実行形態が考えられる．これはモバイルエージェントの典型的な利用例である．なお本研究においては，同じ機能を提供するサービスプロバイダのうちどれをパートナとするかという問題は扱わず，検索サービス等の結果として，利用可能なサービスの候補リストが与えられているものとする．

## 3. ビヘイビアルール記述の構造

本章ではビヘイビアルール記述の構造を示す．

### 3.1 概　　要

ビヘイビアルール記述では，連続して実行される一連のアクティビティの集合（実行ブロック）に対し，移動またはクローニングという物理的なビヘイビアを関連づける．以下**図 4** の例を用いてビヘイビアルール記述の概要を説明する．

**移動**　3〜6 行目のように，migrate ルールにより実行ブロックに対し移動を関連づける．これは（途中で他の移動をはさむことなく）移動先でそれらのアクティビティを実行することを指定する．1 つの移動ルールが 1 回の移動に対応し，移動は開始アクティビティの実行直前のみに行われる．ここでは移動先の指定（target 要素）において利用サービスと同一のホストに移動するよう指定する（local 指定）とともに，移動失敗時にタイムアウト時間をおいての再試行を指定している（retry 指定）．

**クローニング**　7〜11 行目のように，clone ルールにより一連のアクティビティに対しクローンの生成，実行割当てを関連づける．BPEL においては flow 要素によりジョイン条件等も含めて並列性を記述可能であるため，このルールではクローンが実行すべき一連のアクティビティのみを指定する．クローンルールには特にオプション指定はなく，8〜10 行目のようにクローンのビヘイビアを入れ子となるルールで指定する．

**例外処理**　移動等が失敗した場合，通常の BPEL エンジンと同様にリモート通信によって実行を継続することが考えられる．しかし，やりとりするデータが非常に大きいとき等ビヘイビアが重要となる場合，ビヘイビアの失敗を関連づけられた実行ブロックの失敗に結び付けることも考えられる．この例では，rules タグによって囲まれたルールすべてに対し，BPEL プロセス中に関連づけられた例外処理を起動するための fault の通知（throw）を関連づけている（12〜14 行目）．

特筆すべき点として，移動ルールにおいては単に移動を行う地点を指定するのではなく，他の移動をはさまずに移動先で実行する実行ブロックを指定している．これは 2.2 節で述べたサービス間のローカルでの一連の相互作用の実現を保証するために，並列に起動されうる移動処理を実行前の前処理において検出するためである．そのような衝突する移動を実現したい場合には，クローンを生成しての派遣を行えばよい．現時点では，平易な実現手段としてこのように明示的に同一ホストで実行する一連の作業を指定しているが，将来的にはこれを最適化手法により自動化することも考えられる．

ビヘイビアルール記述の擬似 BNF による文法の詳細を付録 A.1 に示す．以下では実行ブロックの定義を示し，移動ルールと例外処理に関して特徴的な記述能力について述べる．

### 3.2 実行ブロックの定義

3.1 節の例において block 属性として記述されている，ビヘイビアを関連づける実行ブロックは以下のいずれかとする．

**activity**　アクティビティ単体 $A_1$ を指定する．

**subsequence**　sequence の直の入れ子のアクティ

ビティ $B_1, B_2, \cdots, B_m$ は順序実行される．このときその部分列をなすアクティビティの集合 $A_1 = B_i, A_2 = B_{i+1}, \cdots, A_n = B_{i+n-1}$ を指定する．

**subgraph** flow の直の入れ子のアクティビティの集合は，link 要素により記述された制御グラフに従い実行される．このときその部分集合 $A_1, A_2, \cdots, A_n$ のうち，それらがなす部分グラフにおいて $A_1$ から $A_2, \cdots, A_n$ のすべてに到達可能であるものを指定する☆．

この定義は，連続して実行される一連のアクティビティの集合を BPEL の制御構造にまたがらないように抜き出すものである，BPEL において複数のアクティビティを入れ子として持つものは sequence と flow のみであり，subsequence，subgraph はこのそれぞれに対応している．どの場合においても，実行ブロック内で最初に実行されるアクティビティ $A_1$ を開始アクティビティと呼ぶ．また，2 つの実行ブロックが一部分のみ同じアクティビティを共有することは許されず，実行ブロックの入れ子構造のみが許される．

### 3.3 記 述 能 力

ここでは移動ルールと例外処理に関して特徴的な記述能力について述べる．

#### 3.3.1 移動先の決定方法

移動先の決定方法（target 要素）に関しては以下のような指定が可能である．

- BPEL においてサービスのバインディング情報を参照するのに用いられる serviceReference 要素により，移動先のプラットフォームサービスを直接指定する．そのリストを指定した場合，移動失敗時に記述された順にホストを変更しながら移動可能なところへ移動する．
- home キーワード指定により最初に生成されたホストに移動する．
- 3.1 節の例で示したように，local キーワード指定により，開始アクティビティで相互作用するパートナのローカルに移動する☆☆．これは利用サービスが実行時に決定する場合でも，システムがアドレス管理をサポートすることによりローカルへの移動を容易に指定可能にするためものである．
- 外部または持ち運んでいるホスト割当てサービスを呼び出す．入力は assign アクティビティによりセットし，出力は上で述べた serviceReference またはそのリスト型である．

#### 3.3.2 移動の再試行

無線通信を用いる場合等においては，一時的な断線等により移動が失敗する状況が考えられるため，移動の再試行に関する指定が必要となる．この指定に関しては，3.1 節の例であげたようにタイムアウト時間を指定しての再試行指定（retry）が可能である．これに加えて，移動先を変更して再試行することが考えられる．条件を満たす別の移動先への移動再試行は，先に述べた移動先のリスト指定により可能である．さらに，移動先の指定が local 指定である場合の移動失敗時に，利用可能なサービスの候補リストに基づいて利用サービスを変更して移動を再試行するように指定することもできる（change キーワード指定）．

#### 3.3.3 例 外 処 理

handle 要素による例外処理はルール自体または rules 要素に対し関連づける．rules 要素に関連づけた場合，その内側にあるすべてのルールに対してこの関連づけが有効となるが，内側の handle 要素によりそれをオーバライドすることもできる．3.1 節では fault を throw する例を示したが，この際に assign アクティビティにより fault 情報を表す変数の代入を行うことや，ビヘイビアの失敗を無視する（例外処理として何もしない empty アクティビティをセットする）ことができる．特に指定をしなかった場合，デフォルトではビヘイビアの失敗を無視して実行を継続する．

## 4. 意味定義の準備

本章ではまず本研究の枠組みの意味定義に用いる Mobile Ambients の概要について述べ，さらに BPEL プロセスを Mobile Ambients により表現する．

### 4.1 Mobile Ambients

Mobile Ambients[8] は $\pi$-calculus[9] と同様の，並列プロセスを表現する形式的言語であり，名前と境界を持つ ambient という概念により入れ子構造を表現し移動性を表現することができる．Mobile Ambients の文法および基本的な動作を**図 5** に示す．このうち特徴的であるのは最後の ambient の概念であり，これを操作する 3 つの capability（$in, out, open$）である．$in$，$out$ はある ambient が他の ambient に出入りするという動作を表現し，これにより移動性を表現することができる．並列する ambient を開く動作を行う $open$ は同期動作を表現する際によく用いられる．本

☆ BPEL の link 要素によるコントロールフローグラフは acyclic なものに制限されているため，この条件が満たされれば $A_1$ は実行ブロック内で最初に実行されることになる（while により実行ブロック全体が繰り返し実行される場合もある）．

☆☆ この際の開始アクティビティはパートナとのメッセージ通信をともなうものに限られる．

| 文法定義 | | | | 基本的な動作 |
|---|---|---|---|---|
| $P, Q \triangleq$ | プロセス | $M \triangleq$ | メッセージ | $(x).P\|M \rightarrow P\{x \leftarrow M\}$ |
| $(\nu n)P$ | 名前制限 | $x$ | 変数 | $n[in\ m.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$ |
| $0$ | 不活動 | $n$ | 名前 | $m[n[out\ m.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$ |
| $P\|Q$ | 並列構成 | $in\ M$ | $M$ に入る | $open\ n.P\|n[Q] \rightarrow P\|Q$ |
| $!P$ | 複製 | $out\ M$ | $M$ から出る | |
| $n[P]$ | ambient | $open\ M$ | $M$ を開く | |
| $M.P$ | 動作 | $\epsilon$ | null | |
| $(x).P$ | 入力動作 | $M.M'$ | パス | |
| $M$ | 出力動作 | | | |

**図 5** Mobile Ambients の文法と基本的な動作
Fig. 5 Syntax and basic actions of Mobile Ambients.

論文ではその詳細を引用しないが，文献 8) においてはこれらの組合せにより条件分岐や反復等の表現がなされている．通常の手続き的プログラミングの制御構造は $\pi$-calculus にマッピング可能であり[11]，さらに Mobile Ambient は $\pi$-calculus を表現可能である[8] ので，Mobile Ambient は手続き的なプログラミングの制御構造を表現する能力を十分に持っている．

### 4.2 BPEL プロセスの表現の方針

通常 BPEL プロセスは 1 つのホスト内の 1 つのエンジンによって実行される．本研究では BPEL プロセス内のアクティビティの一部を移動先で実行したり，クローンと分担して実行したりすることを考えている．そこで本論文においては，個々のアクティビティをそれぞれ Mobile Ambients における 1 つのプロセスとして表現し，それらの制御構造を起動メッセージや完了メッセージ等の交換により表現する．さらに，アクティビティを表現したプロセスは，他のアクティビティやサービスとはリモート通信のみを用いて相互作用するものとする．つまり，すべての相互作用はホスト内で行われることはない．すると直感的には，独立したプロセスとして表現された各アクティビティの所属するエージェントの存在するホストが変わったり（移動），アクティビティの所属するエージェントが変わったり（クローニング）した場合にも，アクティビティ自体の意味は変わらないことになる．

BPEL の解釈実行において各アクティビティは，まず sequence 等の構造アクティビティの意味に基づきそのアクティビティに実行が到達し，さらにフローグラフの有向辺（link）の記述による実行順序制約が存在する場合それが満たされた後に実行される．すると，アクティビティ $A$ は次の順序で実行解釈を行うプロセスとして表現できる．

（$A = A_{PRE1}.A_{PRE2}.A_{ACTION}.A_{POST1}.A_{POST2}$）

(1) 親となる構造アクティビティによる実行制御のためのメッセージ受信（$A_{PRE1}$）
(2) link 記述による実行制御のためのメッセージ受信（$A_{PRE2}$）
(3) アクティビティ自体の実行（$A_{ACTION}$）
(4) 構造アクティビティによる実行制御のためのメッセージ送信（$A_{POST1}$）
(5) link 記述による実行制御のためのメッセージ送信（$A_{POST2}$）

このうち $PRE1$，$POST1$ は親アクティビティの記述によりマッピングされ，その他の 3 つはそのアクティビティ自身の記述によりマッピングされる．

以後混同がない場合においては，「BPEL におけるアクティビティを表現した Mobile Ambients のプロセス」のことを単に「アクティビティ」と呼ぶ．個々のアクティビティの表現 $P, Q, \cdots$ に対し，それらを内包する ambient としてエージェント $a$ を，さらに $a$ を内包する ambient としてホスト $h$ を表現することができる（$h[a[P|Q|\cdots]\cdots]$）．

### 4.3 リモート動作の定義

ここではアクティビティを通信プロセスとして表現するために必要な動作を定義する．

#### 4.3.1 メモリ表現

BPEL プロセスはデータを変数の形で保持する．この際の変数名と値とのペア情報は，文献 8) にある以下のレコードの表現を用いて表現できる．

- $rec\ r(l_1 = v_1, \cdots, l_n = v_n)$：ラベル $l_i$ と値 $v_i$ の組を持つレコード $r$.
- $getr\ r\ l\ (x).P$：レコード $r$ のラベル $l$ の値を $x$ に受け取り $P$ を継続する動作.
- $setr\ r\ l\ x\ .P$：レコード $r$ のラベル $l$ の値を $x$ に置き換え，$P$ を継続する動作.

本論文ではこれらの表現の定義は引用しないが，これらの表現を用いて次のようにレコードの読み書き動作が表現される．

$$\begin{aligned} & rec\ r(l_1 = v_1, \cdots, l_n = v_n)|getr\ r\ l_i\ (x).P \\ \rightarrow\ & rec\ r(l_1 = v_1, \cdots, l_n = v_n)|P\{x \leftarrow v_i\} \\ & rec\ r(l_1 = v_1, \cdots, l_n = v_n)|setr\ r\ l_i\ x\ .P \\ \rightarrow\ & rec\ r(l_1 = v_1, \cdots, l_i = x, \cdots, l_n = v_n)|P \end{aligned}$$

また，本論文においては，ラベルの名前や値の詳細が問題でない場合，$l_1 = v_1, \cdots, l_n = v_n$ を $\bar{l} = \bar{v}$ と

書くこととする．

本研究の表現において各エージェントは，自身のメモリ領域として $mem$ というレコードを含むとする（$h[a[\cdots|mem[\cdots]]\cdots]$）．$mem$ には変数の値の情報に加えて，そのエージェントの名前と現在所属するホストの名前をそれぞれ管理する特別なエントリ $l_a$，$l_h$ を含むものとする．このとき，この $mem$ からエージェント名と所属するホスト名を取得する動作は次のように表現される．

**定義 4.1（エージェント名とホスト名の取得）**

$$getloc(a_c, h_c).P \triangleq getr\ mem\ l_a\ (a_c).getr\ mem\ l_h\ (h_c).P$$

#### 4.3.2 リモート入出力

外部サービスや他のアクティビティとの相互作用に用いるメッセージ通信の表現としては，文献 8) にあるリモート入出力（非同期送信および同期受信）の表現を用いる．

**定義 4.2（リモート入出力）**

$$\begin{aligned}
cmp_{send}(M, x) &\triangleq pkt_{io}[M.x\,] \\
send(M, x).P &\triangleq cmp_{send}(M, x)|P \\
cmp_{recv}(M, x, n) &\triangleq pkt_{io}[M.(x).n[M^{-1}.x\,]] \\
recv(M, x).P &\triangleq \\
&\quad (\nu n)(cmp_{recv}(M, x, n)|open\ n)|(x).P
\end{aligned}$$

これらは移動動作 $M$ によりある地点に移動してから入出力を行うコンポーネントの放出を表現したものである．入出力を行う地点においては，$pkt_{io}$ 内の入出力動作を起動するために，$!open\ pkt_{io}$ という動作が必要となる．$recv$ の場合，このコンポーネントは帰還動作 $M^{-1}$ を行い，ブロックしているプロセスと同期を行う☆．以下にこれらの動作の例をあげる．

$$\begin{aligned}
&a_1[send(out\ a_1.\ in\ ch, n).P] \\
&\quad |a_2[recv(out\ a_2,\ in\ ch, x).Q]|ch[!open\ pkt_{io}] \\
&\rightarrow * a_1[P]|a_2[Q\{x \leftarrow n\}]|ch[!open\ pkt_{io}]
\end{aligned}$$

本研究のモデルにおいては，リモート I/O はアクティビティにより実行される．このため，入出力のための移動動作（定義 4.2 の $M$）は，それが属するエージェントおよびホストから出た後通信のためのチャネルに相当する ambient $ch$ に入るものとなる（$M = getloc(a_c, h_c).out\ a_c.out\ h_c.in\ ch$）．この $ch$ は通信を区別する役割を持ち，その名前を知っている相手のコンポーネントとのみ相互作用できることを保証するものである．以降，このリモート I/O の動作を以下のように略記する．

☆ この帰還動作 $M^{-1}$ は，M 中の動作列を逆順にし，in/out を入れ替えたものである．

**定義 4.3（アクティビティによるリモート I/O）**

$$\begin{aligned}
send'(ch, x) &\triangleq \\
&\quad getloc(a_c, h_c).send(out\ a_c.out\ h_c.in\ ch, x) \\
recv'(ch, x) &\triangleq \\
&\quad getloc(a_c, h_c).recv(out\ a_c.out\ h_c.in\ ch, x)
\end{aligned}$$

#### 4.3.3 データ共有

本研究では BPEL プロセスの実行を複数のエージェントが分担することを考えるため，エージェント間のデータ共有のための機構が必要となる．大域共有メモリのような機構を仮定するとモデルは単純になるが，効率が悪くなる．このため本研究では，各エージェントが自身のメモリ領域を持ち，必要なデータのみを直接エージェント間でやりとりするモデルを考える．データの流れを正確に表現することは，将来的にセキュリティ等の課題に取り組む際に重要となる．

ここでは，ある変数の最新の値をどのエージェントが持っているか，という情報を管理するレコード $loc$，およびそれを提供するメモリ情報管理サービス $sv$ の存在を仮定する．このとき，ある変数に値を代入する動作 $setv$ および最新の値を取得する動作 $getv$ は次のように表現できる．

**定義 4.4（変数アクセス）**

$$\begin{aligned}
&cmp_{setv}(M, l, M', n) \triangleq \\
&\qquad pkt_{loc}[M.setr\ loc\ lM'\ .n[M^{-1}]] \\
&setv(M, l, x, M').P \triangleq \\
&\quad (\nu n)(cmp_{setv}(M, l, M', n) \\
&\qquad |open\ n.setr\ mem\ lx\ .P) \\
&cmp_{getv}(M, l, x, n) \triangleq \\
&\quad pkt_{loc}[M.getr\ loc\ l(M'). \\
&\quad pkt_{mem}[M'.getr\ mem\ l(x).n[(M')^{-1}.M^{-1}.x\,]]] \\
&getv(M, l, x).P \triangleq \\
&\quad (\nu n)(cmp_{getv}(M, l, x, n)|open\ n.(x).P)
\end{aligned}$$

定義 4.2 におけるリモート I/O と同様に，ここでも放出するコンポーネントを考えている．移動動作 $M$ は今回はレコード $loc$ の存在する場所への移動動作である．$setv$ はローカルにある $mem$ 中の $l$ の値を $x$ とし，$loc$ 中の $l$ の値を $M'$ に更新する．この $M'$ は $loc$ のある位置から，$l$ の値を更新したエージェント内までのパスを表す．$getv$ はまず $loc$ 中の $l$ に含まれているパスを取得し，その移動動作を行い $l$ の最新の値を持つエージェントのメモリにアクセスする．リモート I/O のときと同様に，今回もメモリ情報管理サービスは $!open\ pkt_{loc}$ を，各エージェントは $!open\ pkt_{mem}$ を含む必要がある．

この動作は本研究のモデルにおいては，アクティビ

ティによって行われる．そのため移動動作 $M$ は，それが属するエージェントおよびホストから出た後，メモリ管理サービス $sv$ に入るものとなる．また，$M' = M^{-1}$ とすることができる．リモート I/O のときと同様に，これらの動作については以下の略記を用いる．

**定義 4.5（アクティビティによる変数アクセス）**

$$setv'(sv, l, x) \triangleq getloc(a_c, h_c).setv(out\ a_c.out\ h_c.in\ sv, l, x, out\ sv.in\ h_c.in\ a_c)$$

$$getv'(sv, l, x) \triangleq getloc(a_c, h_c).getv(out\ a_c.out\ h_c.in\ sv, l, x)$$

#### 4.3.4 リモート動作

以降，リモート I/O および変数アクセスを合わせて，リモート動作と呼ぶ．次節では BPEL プロセス中の各アクティビティを，リモート動作のみにより外界と相互作用するプロセスとして表現する．

### 4.4 BPEL プロセスとエージェントの表現

Mobile Ambients による本研究の枠組みの表現のため，BPEL プロセスの Mobile Ambient での表現を考える．以下では

⟦ BPEL 記述 ⟧ ::=Mobile Ambients 記述

という表記によってこのマッピングを表す．

#### 4.4.1 エージェントの生成

ホスト $h$ においてインスタンス化された BPEL プロセスは，それを実行するエージェントに含まれるものとして次のように表現される．

⟦ <process> $activities$ </process> ⟧ ::=
$(\nu\ a\ l_1\ \cdots\ l_n\ ch_1\ \cdots\ ch_n\ sv)$
$a[\ [\![activities]\!]$
$|rec\ mem(l_1 = \epsilon, \cdots, l_n = \epsilon, l_a = a, l_h = h)$
$|!open\ pkt_{mem}$
$|ch_1[out\ a.out\ h.!open\ pkt_{io}]$
$|\cdots|ch_n[out\ a.out\ h.!open\ pkt_{io}]$
$|sv[rec\ loc(l_1 = \epsilon, \cdots, l_n = \epsilon)]$
$]$

生成されたエージェントは自身のメモリ $mem$ を持ち，各変数に相当するエントリ $l_1, \cdots, l_n$ と特別なエントリ $l_a, l_h$（4.3.1 項）を含む．また，アクティビティ間の通信に用いるチャネル $ch_1, \cdots, ch_n$ およびメモリ情報管理サービス $sv$ を放出する．すべての名前は制限されており，明示的に受け渡さない限り外部から操作されることはない．このエージェントは，既存の他のエージェントとともにホスト $h$ に含まれる（$h[P|a[\cdots]]$）．

#### 4.4.2 各アクティビティの表現

以下では様々な種類のアクティビティについて，通信プロセスとして表現する際の方針を述べる．**図 6** にいくつかの代表的なアクティビティのマッピング例を示す．

##### 4.4.2.1 メッセージ送受信の表現

基本アクティビティのうちメッセージ送信を行う invoke，receive，reply については，エージェント同様にホスト内のサービスを表現すればリモート I/O を用いて表現可能である．たとえば request-response 型の invoke のマッピングは図 6 (***invoke***) のようになる．ただし $ch$ は，$ptn$，$pt$，$opr$ によって定められる，パートナサービスとの通信チャネルである．

##### 4.4.2.2 構造アクティビティの表現

1 つまたは複数のアクティビティを入れ子に持ち，その実行を制御するアクティビティについて考える．そのようなアクティビティは $ACTION$ 部分において子アクティビティへの制御メッセージの送受信を行い，また子アクティビティは $PRE1$ 部分において開始メッセージ受信，$POST1$ 部分において終了メッセージ送信を行う．たとえば sequence のマッピングは図 6 (***sequence***) のようになる．これは子アクティビティに対し，起動メッセージの送信と終了メッセージの受信を順次行っていくプロセスである．これらは単に同期をとることが目的であるため，空のメッセージとなっている．while，switch アクティビティは文献 8) にある反復，条件分岐の表現により同様の形で記述できる．外部からのイベントに基づいた処理を行う pick も，switch と同様にフラグに基づいた条件分岐により記述すればよい．flow は次に述べる link による制御を行うため，子アクティビティすべてを同時に起動しそれらの終了を待つのみとなる．

##### 4.4.2.3 link の表現

BPEL では flow 要素内に含まれるアクティビティに対し link を関連づけることによりグラフ指向の実行制御が可能である．4.2 節で述べたように，これは構造アクティビティによる実行順序の制御に加えての制御メッセージ送受信として表現できる．link による制御を含むアクティビティのマッピングは図 6 (***link***) のようになる．link の source となる側では $POST2$ 部分で link が遷移したことを表すメッセージを送信し，target となる側では $PRE2$ 部分でそのメッセージを受け取った後アクティビティ自体の処理（$ACTION$ 部分）を開始する．

ただし，この link による実行制御では，遷移条件とジョイン条件を記述することもある．遷移条件の記述が存在する場合に関しては，source 側において条件分岐を行い送るメッセージを $true/false$ から選択

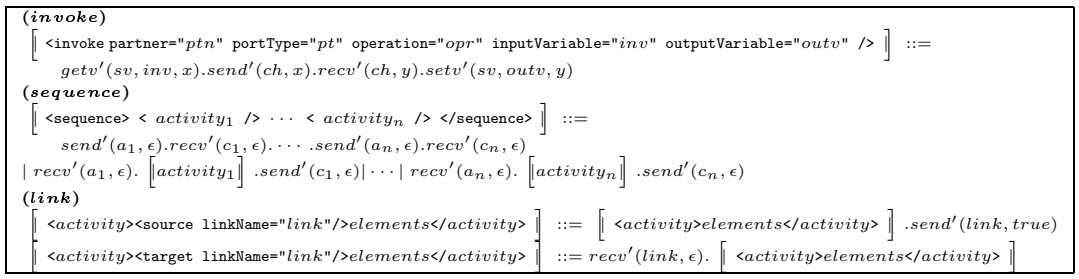


**図 6** アクティビティのプロセスへのマッピング例
Fig. 6 Examples of mapping from activity to process.

するものとして表現できる．また，それを受け取る側では，ジョイン条件を評価しアクティビティの実行を行うかどうかという条件分岐を行う．ジョイン条件が false となった場合は target 側は $ACTION$ 部分の動作には移らず，自身を source とする全 link において $false$ を送り選択の結果を伝播することになる（dead-path-elimination）．

#### 4.4.2.4 scope の表現

BPEL においてはあるアクティビティを含む scope に対して，イベントハンドラや例外処理，ロールバック処理を記述することができる．

- イベントハンドラについては，scope の実行が行われている最中のイベントのみを捕捉するため，scope に対応するプロセスにおいて scope の開始時にフラグを立て，scope の実行終了後にそのフラグを降ろす（$open$）．イベントハンドラはフラグの存在を確認し，かつメッセージとしてのイベントを受信したときに処理を行う．
- 例外の throw と catch については，throw の $ACTION$ 部分において起動メッセージを送信し catch の $PRE1$ 部分においてそれを受信する．invoke 等の実行結果として例外が発生する場合，$POST1$ 部分に通常の完了メッセージ送信を行うか例外処理起動のメッセージ送信を行うか，という条件分岐を記述することになる．これはグラフ指向の記述と似た表現となる．
- BPEL では scope に対して実行完了後のロールバック処理を関連づけ，compensate により起動することができる．これは対応する compensationHandler 内のアクティビティへの起動メッセージの送信として表現できる．
- scope におけるローカル変数の概念は，Mobile Ambients における名前の制限により直接表現できる．また，スコープ中で変数へのアクセスを直列化する serializable scope の概念は，ロックを表現する ambient の導入により表現できる．

#### 4.4.3 アクティビティの表現の性質

BPEL は様々なアクティビティを含みここではすべてのマッピングを示すことはできないが，以上のように，各アクティビティはリモート動作と，外界（他のアクティビティや外部サービス）に影響を与えない内部動作のみを用いて表現ことができる．この性質は形式的には以下のように表現することができる．

**定義 4.6（ACT）** 以下を満たすプロセスの集合のうち最小のものを $ACT$ とする．

- $0 \in ACT$.
- if $P' \xrightarrow{\tau} P$ and $P \in ACT$ then $P' \in ACT$.
- if $P \in ACT$ then $ract.P \in ACT$
  where $ract \in \{send'(ch, x), recv'(ch, x), setv'(sv, l, x), getv'(sv, l, x)\}$.
- if $P, Q \in ACT$ then $P|Q \in ACT$.

**定理 4.1（アクティビティの表現）** BPEL における任意のアクティビティ，およびその集合としての実行ブロックは $ACT$ に属するプロセスとして Mobile Ambient により表現可能である．

$ACT$ の定義においては，$ACT$ に属するプロセスのの並列構成もまた $ACT$ に属するため，実行ブロックもまた $ACT$ に属するプロセスとして表現できる．この性質により，本研究の枠組みの性質を議論する際にその議論の対象を $ACT$ に属するプロセスに制限することができる．

## 5. ビヘイビアルールの意味定義

本章では本研究の記述の枠組みの意味定義を行う．

### 5.1 意味定義の方針

ビヘイビアルール記述は，BPEL で記述された連携プロセスに対して移動やクローン生成といったビヘイビア制御，および必要ならば fault の throw 等を挿入するものとして説明することができる．ここで移動動作等は BPEL により記述できるものではなく，プラットフォームがそれをサポートする動作を行う必要がある．また移動やクローニングといった物理的なビヘイ

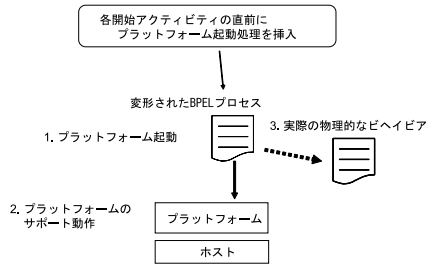


**図 7** 3 段階による物理的なビヘイビア
Fig. 7 3-phase physical behavior.

ビアそのものは Mobile Ambient を用いて表現することができる．そこで本章では，3 段階の処理としてビヘイビアルールの意味定義を行う（**図 7** は移動の場合の処理を表す図である）．

(1) ビヘイビア処理の起動処理等を BPEL プロセスに挿入し BPEL プロセスを変形する方法を示す（5.2 節）．
(2) 起動されたプラットフォームの，移動先の決定や移動の再試行等を含めたビヘイビア処理を手続き的な擬似プログラムとして示す（5.3 節）．
(3) 移動およびクローニングというビヘイビア自体の意味を Mobile Ambients を用いて示す（5.4 節）．

4 章において Mobile Ambients を用いて BPEL を表現したこと，また Mobile Ambients は手続き的なプログラム構造を表現可能であることより，最終的に連携のための全動作を Mobile Ambients によって表現することができる．以下ではこの 3 段階の動作のうち特に，(3) の BPEL プロセスに物理的なビヘイビアを関連づけること自体の意味に重点をおいて説明する．

### 5.2 BPEL プロセスの変換

移動およびクローン生成という実際のビヘイビア動作は，実行ブロックの開始アクティビティが実行可能になり，実際に実行される直前に行われる．これは BPEL プロセスに対し，以下のようにビヘイビア動作を挿入することとして説明できる．

各ブロックの開始アクティビティ $A_1$ を以下のように置き換える．

```
<sequence>
<!--- behavior action --->
<A1 .../>
</sequence>
```

さらに，A1 の実行開始を制御する（link に対する）target 要素や joinCondition 属性があれば，それらをすべて外側の sequence 要素内に移動する．

指定された実行ブロックが存在しない場合，そのルールは無効となる．ここで挿入されるビヘイビア動作は，ビヘイビア処理のプラットフォームへの依頼に相当する．すなわち上の behavior action の部分には，その実行ブロックに関連づけられたルールの内容を入力とした，プラットフォームに対する invoke アクティビティが含まれる．さらに，必要に応じて以下の処理が挿入される．

- ビヘイビアが失敗した場合にプラットフォームから返される fault に対応する処理を挿入する．もしもそのビヘイビアに関連づけられた handle 要素がある場合，その要素内の処理を sequence に入れ子として，invoke に対する faultHandler として付加する．そのような例外処理がない場合は，faultHandler として empty アクティビティを挿入する．
- クローン生成に対応する faultHandler においては，クローン生成に失敗したことを表す変数をセットしておく．これは並列する移動を禁止するため，クローンの生成に失敗した場合にそのクローンの行う移動を無効にするためである．ビヘイビアルールにおいて clone 要素の入れ子となっている migrate 要素に対しては，switch アクティビティ内でその変数の値を参照し，クローン生成が成功したときのみ移動を起動するようにする．

### 5.3 プラットフォームの動作

前節の変換後の BPEL プロセスから呼び出されたプラットフォームの動作を**図 8** に手続き的に示す．移動先の複数指定と再試行指定が併用された場合，タイムアウト時間をおいての再試行と移動先の切替えが交互に行われる．実際の移動（`migrate`）およびクローニング（`clone`）自体の意味については次節で示す．

### 5.4 ビヘイビアの定義

ここでは 4 章で述べた BPEL とエージェントの表現を用い，移動およびクローニングという物理的なビヘイビア自体の意味定義を行う．

#### 5.4.1 migrate 動作

ある実行ブロックを表現したプロセス $P$ に対して移動動作を関連づけたものは，プロセス $migrate\ ch.P$ として表現される．ここで移動動作 $migrate\ ch$ は次のように定義される．

**定義 5.1（migrate 動作）**

$$migrate\ ch.P \triangleq (\nu\ b)(recv'(ch, th).getloc(a_c, h_c).$$

```
if rule is migrate-rule goto migrate
if rule is clone-rule goto clone
migrate:
  if target is local or invocation
    hostlist = result of
                 service invocation;
  i=0;
  while (i<hostlist.length){
    host = hostlist[i];
    if able to migrate to host{
      migrate(agent);
      return ‘‘migration-succeeded’’;
    }
    else{
      i++;
      if retry with timeout is set{
        while (not timeout){
          if able to migrate to host{
            migrate(agent);
            return ‘‘migration-succeeded’’;
          }
        }
      }
      if target is local
           and retry with change is set{
        if another service is available
          set the service as the current
          goto migrate
      }
    }
  }
  return fault;
}
clone:
  if able to clone{
    clone(block);
    return ‘‘clone-succeeded’’;
  }
  else
    return fault;
```

**図 8** プラットフォームのビヘイビア制御
Fig. 8 Behavior control by platform.

$$out\ h_c.in\ th.setr\ mem\ l_h\ th.open\ b|b[P])$$

この動作はまず，起動されたプラットフォームから，前もって定められたチャネル $ch$ を通して移動先のホスト名（$th$）を受け取り，現在のホストを出て移動先のホストに入り，メモリ中の $l_h$ エントリを更新し，最終的に $P$ の実行を継続する．この定義は以下のように動作する（この導出の詳細は付録 A.2 に示す）．

$$\begin{aligned} &h_1[a[migrate\ ch.P|Q|rec\ mem(l_h = h_1, \bar{l} = \bar{v})]|R] \\ &\quad |h_2[S]|ch[!open\ pkt_{io}]||send(in\ ch, h_2) \\ &\rightarrow^* h_1[R]|h_2[S|a[P|Q|rec\ mem(l_h = h_2, \bar{l} = \bar{v})]] \\ &\quad |ch[!open\ pkt_{io}] \end{aligned}$$

### 5.4.2 clone 動作

ある実行ブロックを表現したプロセス $P$ に対してクローニング動作を関連づけたものは，プロセス（$clone\ ch.P$）として表現される．ここでクローニング動作 $clone\ ch$ は次のように定義される．

**定義 5.2（clone 動作）**

$$\begin{aligned} clone.P \quad \triangleq \\ &(\nu\ a)recv'(ch, \epsilon).getloc(a_c, h_c).a[out\ a_c.P| \\ &\quad rec\ mem(l_a = a, l_h = h_c, \bar{l} = \bar{\epsilon})|!open\ pkt_{mem}] \end{aligned}$$

この動作はまず，起動されたプラットフォームから，前もって定められたチャネル $ch$ を通して起動され，$P$ を含む新たなエージェントを生成する．この定義は以下のように動作する（この導出の詳細は付録 A.2 に示す）．

$$\begin{aligned} &h[a_1[clone.P|Q|rec\ mem(l_a = a_1, l_h = h, \bar{l} = \bar{v})]] \\ &\quad |ch[!open\ pkt_{io}]||send(in\ ch, \epsilon) \\ &\rightarrow^* (\nu\ a)(h[a_1[Q|rec\ mem(l_a = a_1, l_h = h, \bar{l} = \bar{v})] \\ &\quad |a[P|rec\ mem(l_a = a, l_h = h, \bar{l} = \bar{\epsilon})|!open\ pkt_{mem}]] \\ &\quad |ch[!open\ pkt_{io}]) \end{aligned}$$

## 6. BPEL との整合性

4 章では，BPEL のアクティビティをリモート動作を通してのみ外界と相互作用するプロセスとして表した．アクティビティ間の実行順序の制御等の相互作用はエージェント内やホスト内で行われることはないため，直感的には，移動やクローニングによってアクティビティが異なるエージェント，ホストに分散した場合でもその意味は変わらない．本章ではこのことを形式的に示す．

### 6.1 振舞いの等価性の定義

本研究では，アクティビティの属するエージェント，もしくはその属するホストを変更するという物理的なビヘイビアを考えている．その際，BPEL プロセス自体に影響を与える handle 要素による例外処理を除き，アクティビティ間の実行順序等 BPEL プロセス自体のロジックは保存されるべきである．定理 4.1 においては，通信プロセスとして表現されたアクティビティ間の相互作用はリモート動作のみであることを述べた．そこで以下では，ホストの外から検出できるリモート動作のみに注目した振舞いの等価性を定義する．

このように特定の動作に注目して等価性を定義する方法として，barbed bisimulation（双模倣性）の概念がある．たとえば文献 12) では，in，out，open の動作☆についての等価性を議論している．以下ではリモート動作のみに注目した双模倣性を定義し，それに基づいた等価関係 $\approx$ を定義する．

☆ このように観測の対象とする動作を barbs という．

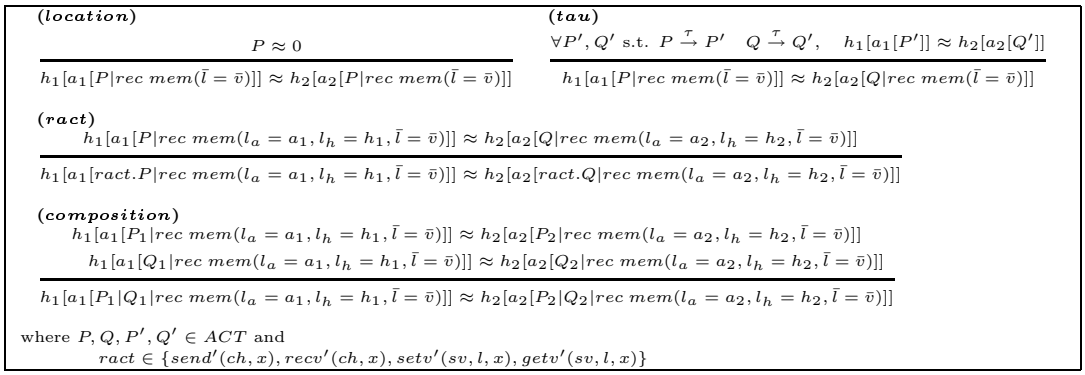


**図 9** 等価性に関する補題
Fig. 9 Lemmas about equivalence.

**定義 6.1（Barbs）** 以下では $ch$ および $sv$ が $m_1 \ldots m_n$ に含まれないとする．

$$P \downarrow send'(ch,x) \triangleq P \equiv (\nu m_1 \ldots m_n)(cmp_{send}(in\ ch, x)|P')$$

$$P \downarrow recv'(ch,x) \triangleq \exists n \text{ s.t. } P \equiv (\nu m_1 \ldots m_n)(cmp_{recv}(in\ ch, x, n)|P')$$

$$P \downarrow setv'(sv,l,x) \triangleq \exists M', n \text{ s.t. } P \equiv (\nu m_1 \ldots m_n)(cmp_{setv}(in\ sv, l, out\ sv.M', n)|P')$$

$$P \downarrow getv'(sv,l,x) \triangleq \exists n \text{ s.t. } P \equiv (\nu m_1 \ldots m_n)(cmp_{getv}(in\ sv, l, x, n)|P')$$

**定義 6.2**

$$P \Downarrow M \triangleq P \rightarrow^* Q \text{ and } Q \downarrow M$$

これらの barbs では，エージェントおよびホストから出て，$ch$ や $sv$ に入ろうとしているコンポーネントの存在を観測する．これらのコンポーネントの定義（4.3 節）においては，$n$ はブロッキングしているプロセスとの同期に用いられ，また $M'$ はそのプロセスへの帰還パスであった．上の barbs の定義においては，$n$ と $M'$ を無視している．これはリモート動作に関連して振舞いの等価性を考える際に，コンポーネントがどのエージェントやホストから送り出されたのかは区別しないということである．

以下ではさらに，これらの barbs の観測に基づいた双模倣性を考えることにより等価性を定義する．

**定義 6.3（Barbed Bisimulation）** 次を満たす対称な関係 $\mathcal{R}$ を双模倣と呼ぶ．

if $P \mathcal{R} Q$ then
- if $P \downarrow M$ then $Q \Downarrow M$
- if $P \rightarrow P'$ then for some $Q'$, $Q \rightarrow^* Q'$ and $P' \mathcal{R} Q'$

**定義 6.4（Barbed Equivalence）** $P \mathcal{R} Q$ なる双模倣 $\mathcal{R}$ が存在するとき $P \approx Q$.

次節ではこの等価性の定義に基づいて，本研究の物理的なビヘイビアの導入の性質を考える．

### 6.2 BPEL との整合性

本節では，前節で定義したホストの外から見た振舞いに関して，移動およびクローニングの導入が元の BPEL プロセスの振舞いを変えないことを示す．定理 4.1 より，$ACT$ に属するプロセスのみを対象として議論を行うことができる．**図 9** に，$ACT$ に属するプロセスに関し，前節で定義した等価性に関する性質をあげる．

まず，外界と相互作用しない，すなわちリモート動作を含まないプロセス $P$ を考えてみる．この性質は $P \approx 0$ と表現できる．この $P$ は，明らかに所属するエージェントやホストが変化しても，外部から見た振舞いは変化しない（図 9 **(location)**）．形式的には，これは以下の関係が双模倣であることから示せる．

$$\{(h_1[a_1[P|rec\ mem(\bar{l}=\bar{v}, l_a=a_1, l_h=h_1)]], h_2[a_2[P|rec\ mem(\bar{l}=\bar{v}, l_a=a_2, l_h=h_2)]]) \mid h_1, h_2, a_1, a_2 : names, P \approx 0\}$$

**(tau)** および **(ract)** は，2 つのプロセスの外部から見た振舞いの等価性は，内部動作や同じリモート動作を付加しても保たれることを示している．**(tau)** は Mobile Ambients における基本的な導出ルール $P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$ より導かれる．**(ract)** はリモート動作の定義（定義 4.3 および 4.5）から導ける．直感的には，リモート動作を行う際に，エントリ $l_a$，$l_h$ における現在のエージェント名とホスト名が正しくセットされていれば，エージェントおよびホストを出た後の動作は同じであるということである．

**(composition)** は，外部から見た振舞いが等価な 2 組のプロセスを並列合成して得られるプロセスの振舞いもまた等価となることを示している．これは以下のように，$ACT$ に属するプロセスはけっして並列す

るプロセスに直接アクセスしないこと，またリモート動作は適切なチャネルまたはメモリ管理サービスが存在しなければ完了しないこと，により示される☆．

$$\text{If } h[a[P|Q|rec\ mem(\bar{l}=\bar{v})]] \to R$$
$$\text{then there exists some } P_1,P_2,P_3,Q_1,Q_2,Q_3 \text{ s.t.}$$
$$h[a[P|rec\ mem(\bar{l}=\bar{v})]] \to h[a[P_1|rec\ mem(\bar{l}=\bar{v})]|P_2]|P_3,$$
$$h[a[Q|rec\ mem(\bar{l}=\bar{v})]] \to h[a[Q_1|rec\ mem(\bar{l}=\bar{v})]|Q_2]|Q_3,$$
$$R \equiv h[a[P_1|Q_1|rec\ mem(\bar{l}=\bar{v})]|P_2|Q_2]|P_3|Q_3$$

ここであげた補題および $ACT$ の定義（定義 4.6）より，構造的帰納法を用いて次の命題を示すことができる．この命題は，エントリ $l_a$，$l_h$ が適切に設定されていれば，$ACT$ に属するプロセスはどのエージェント，ホストに所属しても外部から見た振舞いは変わらないことを表している．

**命題 6.1（アクティビティの位置非依存性）**
任意の $P \in ACT$ に対して，

$$h_1[a_1[P|rec\ mem(l_a=a_1,l_h=h_1,\bar{l}=\bar{v})]] \approx h_2[a_2[P|rec\ mem(l_a=a_2,l_h=h_2,\bar{l}=\bar{v})]]$$

さらに，以上の議論と同様にしてタスク割当てに関する次の命題を導くことができる．

**命題 6.2（アクティビティの割当て非依存性）**
任意の $P,Q \in ACT$ に対して，

$$h[a[P|Q|rec\ mem(l_a=a,l_h=h,\bar{l}=\bar{v})]] \approx h[a_1[P|rec\ mem(l_a=a_1,l_h=h,\bar{l}=\bar{v})] |a_2[Q|rec\ mem(l_a=a_2,l_h=h,\bar{l}=\bar{v})]]$$

これは図 9 の補題から命題 6.1 を示したように，$P \approx 0, Q \approx 0$ から開始して内部動作またはリモート動作を付加することにより，$ACT$ の構造に関する帰納法で示すことができる．ここではその手順は省略する．

以上をふまえ，実行ブロックへの物理的なビヘイビアの導入について議論する．

**定理 6.1（物理的なビヘイビアの付加）**
任意の $P,Q \in ACT$ に対して，

$$h_1[a[migrete\ ch.P|Q|rec\ mem(l_h=h_1,\bar{l}=\bar{v})]]|h_2[] |ch[!open\ pkt_{io}]||send(in\ ch,h_2) \approx h_1[a[P|Q|rec\ mem(l_h=h_1,\bar{l}=\bar{v})]]|h_2[] |ch[!open\ pkt_{io}]||send(in\ ch,h_2)$$

$$h[a_1[clone.P|Q|rec\ mem(l_a=a_1,\bar{l}=\bar{v})]] |ch[!open\ pkt_{io}]||send(in\ ch,\epsilon) \approx h[a_1[P|Q|rec\ mem(l_a=a_1,\bar{l}=\bar{v})]] |ch[!open\ pkt_{io}]||send(in\ ch,\epsilon)$$

前者は，定義 5.1 において，$migrate$ はエージェントを含むホストを変更し，$l_h$ エントリを適切に更新する動作であるので，命題 6.1 より導くことができる．後者も同様に，定義 5.2 および命題 6.2 より導くことができる．

定理 6.1 は 1 つの実行ブロックについて述べたものであるが，最後に，BPEL 中の複数の実行ブロックに対して物理的なビヘイビアの導入を行った場合について議論する必要がある．まず実行ブロックに対して物理的なビヘイビアを導入したプロセスの集合を定義する．

**定義 6.5（ACT'）** 以下を満たすプロセスの集合のうち最小のものを $ACT'$ とする．

- if $P \in ACT$ then $P \in ACT'$
- if $P \in ACT'$ then $pb.P \in ACT'$ where $pb \in \{migrate\ ch, clone\ ch\}$
- if $P,Q \in ACT'$ then $P|Q \in ACT'$.

この定義においては，3.2 節で述べたように，物理的なビヘイビアを関連づけた実行ブロックが別のブロックに入れ子となることが可能だが，ブロックの境界にまたがることはできない．

$ACT'$ に属するプロセスに関しても，同様にして物理的なビヘイビアをさらに付加した際の振舞いの等価性を示すことができる．

**定理 6.2（物理的なビヘイビアの段階的な付加）**
任意の $P,Q \in ACT'$ に対して，

$$h_1[a[migrete\ ch.P|Q|rec\ mem(l_h=h_1,\bar{l}=\bar{v})]]|h_2[] |ch[!open\ pkt_{io}]||send(in\ ch,h_2) \approx h_1[a[P|Q|rec\ mem(l_h=h_1,\bar{l}=\bar{v})]]|h_2[] |ch[!open\ pkt_{io}]||send(in\ ch,h_2)$$

$$h[a_1[clone.P|Q|rec\ mem(l_a=a_1,\bar{l}=\bar{v})]] |ch[!open\ pkt_{io}]||send(in\ ch,\epsilon) \approx h[a_1[clone.P|Q|rec\ mem(l_a=a_1,\bar{l}=\bar{v})]] |ch[!open\ pkt_{io}]||send(in\ ch,\epsilon)$$

これは，通常の BPEL プロセスに対し，それに含まれる実行ブロックに物理的なビヘイビアを付加することを繰り返しても，個々のアクティビティの外部から見た振舞いは変わらない，という本研究の枠組みにおいて重要な性質を示している．

☆ 定義 4.4 においては，$setv$ は $cmp_{setv}$ がメモリ管理サービスを更新して戻ってくるまで $mem$ を書き換えないようになっている．

## 7. 関 連 研 究

### 7.1 モバイルエージェントの Web サービスへの適用

Web サービスとその連携に関しては多くの研究がなされており，モバイルエージェントを用いるものも見られる．しかし，BPEL のような連携のための仕様と組み合わせているものはまだない．移動性の活用について考えると，まずいくつかの研究では，つねにローカルアクセスするというような単純なモデルを考えている[13]．そのようなモデルでは，やりとりするデータ量や通信路の速度によっては移動のオーバヘッドの方が大きくなる等，様々な環境に対応することができない．また，多くの研究が動的な資源管理によるコストの見積りと最適化に基づいたモデルを採用している[14)~16)]．本研究のビヘイビアルール記述は，利用可能な移動先リストを静的に指定したりホスト割当てサービスを用いたり，というようにドメインごとに移動先の決定方法を変える等，様々なモデルの選択，組合せの容易な実現を目指している．

また，モバイルエージェントの移動性をアプリケーションロジックから分離し，宣言的に記述するものとしては文献 17) がある．本研究においては，local 指定等 Web サービスの動的な側面を考慮している点や，クローニングの宣言的な制御も可能となっている点が大きな特徴である．

### 7.2 形 式 化

本論文においては，我々のモデルにおけるモバイルエージェントの振舞いの形式的な定義を行った．これは今後の取り組みの基盤とするためのものであり，ツールのサポートや検証に用いていきたい．モバイルエージェントの振舞いは比較的複雑になりやすく，また実際にテストする分散環境を構築する手間もかかるため，実際に実行する前に様々な性質の検証等を行えることが望ましい．

要求仕様の記述に関して，Mobile Ambients に基づいた様相論理である Ambient Logic [18] がある．Ambient Logic では，eventually，always という通常の時間に関する様相に加え，somewhere，everywhere という場所に関する様相を扱うことができる．これにより，「特定のデータをいつかどれかのエージェントが持つ」といった様々な要件を表現することができる．

Web サービス記述言語，特に複雑なグラフ指向の記述を行うものに関して記述の検証は重要となる[19),20)]．本論文の直接の目的ではないが，本論文で示した BPEL の意味定義を元に，連携プロセス自体の要件についての検証を行うことも考えられる．

上で述べたような検証等のために，実際に本研究で記述したような Mobile Ambients の式をどのようにして解析するかという問題が残されている．この問題に関しては，文献 21) のように Mobile Ambient 式の実行や解析を行う既存の様々なツールをふまえて，現在検討を行っている．

## 8. む す び

本論文では，Web サービス連携にモバイルエージェントを適用するための枠組みを示し，その意味定義を行った．この枠組みでは BPEL プロセス記述に変化を加えることなく，適用環境に応じて物理的なビヘイビアを付加，変更することができる．現在はこの意味定義に基づいた実装や検証の実現に取り組むとともに，障害からの復帰等より高度な記述の導入を検討している．

## 参 考 文 献


1) Web Services. http://www.w3.org/2002/ws/
2) Thatte, S., et al.: Business Process Execution Language for Web Services, Version 1.1 (2003). http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
3) alphaWorks: Web Services Tool Kit for Mobile Devices (2002). http://www.alphaworks.ibm.com/tech/wstkmd
4) Milojicic, D.: Mobile Agent Applications, *IEEE Concurrency*, Vol.7, No.3, pp.7–13 (1999).
5) Chess, D., Harrison, C. and Kershenbaum, A.: Mobile Agents: Are They a Good Idea?, Technical Report (1994).
6) 石川冬樹，吉岡信和，本位田真一：パーベイシブネットワークにおける Web サービス連携のためのモバイルエージェントシステム，電子情報通信学会論文誌（D-I）（掲載予定）．
7) Leymann, F.: Web Services Flow Language (WSFL1.0) (2001). http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
8) Cardelli, L. and Gordon, A.D.: Mobile Ambients, *Foundations of Software Science and Computation Structures: 1st International Conference* (1998).
9) Parrow, J.: An Introduction to the pi-Calculus, *Handbook of Process Algebra*, pp.479–543 (2001).
10) Web Service Description Language (WSDL) 1.1 (2001). http://www.w3.org/TR/wsdl
11) Odersky, M.: Applying *pi*: Towards a Basis for Concurrent Imperative Programming, *ACM*

**(*migrate*)**

$h_1[a[migrate\ ch.P|Q|rec\ mem(l_h = h_1, \bar{l} = \bar{v})]|R]|h_2[S]|ch[!open\ pkt_{io}]|send(in\ ch, h_2)$
$\equiv h_1[a[(\nu\ b)(recv'(ch, th).getloc(a_c, h_c).out\ h_c.in\ th.setr\ mem\ l_h\ th.open\ b|b[P])|Q|rec\ mem(l_h = h_1, \bar{l} = \bar{v})]|R]$
$\quad |h_2[S]|ch[!open\ pkt_{io}]]|send(in\ ch, h_2)$
$\rightarrow (\nu\ b)(h_1[a[getloc(a_c, h_c).out\ h_c.in\ h_2.setr\ mem\ l_h\ h_2.open\ b|b[P]|Q|rec\ mem(l_h = h_1, \bar{l} = \bar{v})]|R]|h_2[S]|ch[!open\ pkt_{io}])$
$\rightarrow (\nu\ b)(h_1[a[out\ h_1.in\ h_2.setr\ mem\ l_h\ h_2.open\ b|b[P]|Q|rec\ mem(l_h = h_1, \bar{l} = \bar{v})]|R]|h_2[S]|ch[!open\ pkt_{io}])$
$\rightarrow (\nu\ b)(h_1[R]|h_2[S|a[setr\ mem\ l_h\ h_2.open\ b|b[P]|Q|rec\ mem(l_h = h_1, \bar{l} = \bar{v})]]|ch[!open\ pkt_{io}])$
$\rightarrow (\nu\ b)(h_1[R]|h_2[S|a[open\ b|b[P]|Q|rec\ mem(l_h = h_2, \bar{l} = \bar{v})]]|ch[!open\ pkt_{io}])$
$\rightarrow h_1[R]|h_2[S|a[P|Q|rec\ mem(l_h = h_2, \bar{l} = \bar{v})]]|ch[!open\ pkt_{io}]$

**(*clone*)**

$h[a_1[clone.P|Q|rec\ mem(l_a = a_1, l_h = h, \bar{l} = \bar{v})]]|ch[!open\ pkt_{io}]|send(in\ ch, \epsilon)$
$\equiv h[a_1[(\nu\ a)recv'(ch, \epsilon).getloc(a_c, h_c).a[out\ a_c.P|rec\ mem(l_a = a, l_h = h_c, \bar{l} = \bar{\epsilon})|!open\ pkt_{mem}]$
$\quad |Q|rec\ mem(l_a = a_1, l_h = h, \bar{l} = \bar{v})]]|ch[!open\ pkt_{io}]|send(in\ ch, \epsilon)$
$\rightarrow (\nu\ a)(h[a_1[getloc(a_c, h_c).a[out\ a_c.P|rec\ mem(l_a = a, l_h = h_c, \bar{l} = \bar{\epsilon})|!open\ pkt_{mem}]|Q|rec\ mem(l_a = a_1, l_h = h, \bar{l} = \bar{v})]]$
$\quad |ch[!open\ pkt_{io}]])$
$\rightarrow (\nu\ a)(h[a_1[a[out\ a_1.P|rec\ mem(l_a = a, l_h = h, \bar{l} = \bar{\epsilon})|!open\ pkt_{mem}]|Q|rec\ mem(l_a = a_1, l_h = h, \bar{l} = \bar{v})]]|ch[!open\ pkt_{io}])$
$\rightarrow (\nu\ a)(h[a_1[Q|rec\ mem(l_a = a_1, l_h = h, \bar{l} = \bar{v})]|a[P|rec\ mem(l_a = a, l_h = h, \bar{l} = \bar{\epsilon})|!open\ pkt_{mem}]]|ch[!open\ pkt_{io}])$

**図 10** 物理的なビヘイビアの導出
Fig. 10 Reductions of physical behaviors.

*SIGPLAN Workshop on State in Programming Languages* (1995).
12) Vigliotti, M. and Phillips, I.: Barbs and Congruences for Safe Mobile Ambients, *F-WAN: Foundations of Wide Area Network Computing* (2002).
13) Padovitz, A., Krishnaswamy, S. and Loke, S.W.: Toward Efficient and Smart Selection of Web Service, *AAMAS'2003 Workshop on Web Services and Agent-based Engineering* (2003).
14) Fukuda, M., Tanaka, Y., Suzuki, N., Bic, L.F. and Kobayashi, S.: A Mobile-Agent-Based PC Grid, *Autonomic Computing Workshop 5th Annual International Workshop on Active Middleware Services* (*AMS'03*) (2003).
15) Ragusa, C., Liotta, A. and Pavlou, G.: Dynamic Resource Management for Mobile Services, *5th International Workshop on Mobile Agents for Telecommunication Applications* (*MATA'03*) (2003).
16) Maamar, Z., Sheng, Q.Z. and Benatallah, B.: On Composite Web Services Provisioning in an Environment of Fixed and Mobile Computing Resources, *Information Technology and Management*, Vol.5, No.3, Kulwar Academic Publishers (2004).
17) Montanari, R. and Tonti, G.: A Policy-Based Infrastructure for the Dynamic Control of Agent Mobility, *3rd International Workshop on Policies for Distributed Systems and Networks* (*POLICY'02*), pp.206–209 (2002).
18) Cardelli, A.G.L.: Anytime, Anywhere, Modal Logics for Mobile Ambients, Typed Lambda Calculi and Applications, *5th International Conference*, pp.46–60 (2001).
19) 中島 震：Web サービスフロー記述のモデル検査検証，情報処理学会論文誌，Vol.44, No.3, pp.942–952 (2003).
20) Foster, H., Uchitel, S., Magee, J. and Kramer, J.: Model-based Verification of Web Service Compositions, *18th IEEE International Conference on Automated Software Engineering* (2003).
21) Braghin, C., Cortesi, A., Filippone, S., Focardi, R., Luccio, F. and Piazza, C.: BANANA — A tool for boundary ambients nesting analysis, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (*TACAS'03*), (2003).

## 付　　録

### A.1 ビヘイビアルール記述の文法

ビヘイビアルール記述の文法を擬似 BNF により以下に示す．ただし，*namespace* : がつくものは既存の記述を利用したものである．*bpws* : は BPEL，*sref* : は serviceReference，*xsd* : は XML Schema Data Types をそれぞれ参照している．

*behavior* := <behavior>*rule*+</behavior>

*rule* := *migrate-rule* | *clone-rule* | *rules*

*rules* := <rules> *rule*+ *handle*? </rules>

*handle* := <handle>*handler* </handle>

*handler* := *rethrow* | *suppress*

*rethrow* := *bpws:assign bpws:throw*

*suppress* := *bpws:empty*

*migrate-rule* :=
<migrate block = '' *bpws:actvityname*+ ''>
<target>*host*</target>
*retry*? *handle*?
</migrate>

*target* := *hosts* | <home />| <local />| *invocation*

*hosts* := *sref:serviceReference*+

*invocation* := *bpws:assign** *bpws:invoke*

```
retry := timeout? change?
timeout := <retry time=''xsd:positiveInteger'' />
change := <change / >

clone-rule :=
  <clone block = '' bpws:activityname+ '' >
    rules* handle?
  </clone>
```

### A.2 物理的なビヘイビアの導出

*migrate* および *clone* の導出の詳細を**図 10** に示す.

(平成 15 年 10 月 15 日受付)
(平成 16 年 3 月 5 日採録)

**石川 冬樹**（学生会員）

1980 年生．2002 年東京大学理学部情報科学科卒業．同年東京大学大学院情報理工学系研究科修士課程進学，現在に至る．2004 年修士課程修了および同博士課程進学の見込み．エージェント技術および Web サービス技術等の研究に興味を持つ．電子情報通信学会学生会員，日本ソフトウェア科学会学生会員．

**田原 康之**（正会員）

1966 年生．1991 年東京大学大学院理学系研究科数学専攻修士課程修了．同年（株）東芝入社．1993 年〜1996 年情報処理振興事業協会に出向．1996 年〜1997 年英国 City 大学客員研究員．1997 年〜1998 年英国 Imperial College 客員研究員．2003 年より文部科学省国立情報学研究所科学研究支援員，現在に至る．博士（情報科学，早稲田大学）．エージェント技術，およびソフトウェア工学等の研究に従事．日本ソフトウェア科学会会員．

**吉岡 信和**（正会員）

1971 年生．1993 年富山大学工学部電子情報工学科卒業．1998 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（情報科学）．同年（株）東芝入社．エージェント技術の研究，ソフトウェア工学の研究に従事．2002 年より文部科学省国立情報学研究所産学官連携研究員，現在に至る．日本ソフトウェア科学会会員．

**本位田真一**（正会員）

1953 年生．1976 年早稲田大学理工学部電気工学科卒業．1978 年同大学大学院理工学研究科電気工学専攻修士課程修了．（株）東芝を経て 2000 年より文部科学省国立情報学研究所教授，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を併任，現在に至る．2002 年 5 月〜2003 年 1 月英国 UCL ならびに Imperial College 客員研究員（文部科学省在外研究員）．工学博士（早稲田大学）．本学会理事．1986 年度情報処理学会論文賞受賞．エージェント技術，オブジェクト指向技術，ソフトウェア工学の研究に従事．IEEE，ACM，日本ソフトウェア科学会等各会員．