プロセス記述によるサービス合成のパーベイシブコンピューティングへの適用

石川冬樹[†] 吉岡信和[††] 本位田真一[†,††]

パーベイシブコンピューティングの実現を目指す取り組みにおいて，センサやデバイスをサービスによりカプセル化するアプローチがある．パーベイシブコンピューティングにおいては状況に応じての利用サービスの選択や，状況の変化に応じての実行中断，サービスの切替えが必要となる．しかしそれらを合成フローに交えてすべて手続き的に記述してしまうと，その制御フローは複雑で再利用が困難なものとなってしまう．一方サービス指向コンピューティングのための枠組みでは，サービス合成プロセスとサービス選択機構を分離して扱うことができるが，実行時のサービスの切替えなどには対応できない．そこで本研究においては，サービスの切替えや実行の中断を考慮した形で再利用可能な合成プロセスを定義するためのモデルを提案するとともに，パートナの選択や切替えに関する指定を別途与えるための枠組みを提案する．この枠組みを用いることにより，中断時のサービスへのキャンセルの送信なども含めた形で合成プロセスを再利用可能とし，環境や嗜好に応じて容易にサービスの選択や切替えのロジックを与えたり差し替えたりすることができる．

Adoption of Process-based Service Composition to Pervasive Computing

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

Encapsulation of sensors and devices by networked services has been used for realization of pervasive computing. In pervasive computing, it is necessary to select services according to the current situation and to abort execution to switch services upon change in the situation. However, weaving such actions into service composition flow leads to very complex and non-reusable flow. On the other hand, although frameworks for Service-Oriented Computing provide mechanisms to manage service composition flow and mechanisms for service selection separately, switch of services at runtime has not been dealt with in such frameworks. This study proposes a model to describe a reusable composition process, considering switch of services and execution abort. This study also proposes a framework where behaviors for service selection and switch are described separately. The framework facilitates to describe reusable composition processes, including behaviors for cancelling upon execution abort, and to add or change behaviors for service selection and switch according to environmental conditions and preferences.

1. はじめに

近年注目されているサービス指向コンピューティングにおいては，分散されたコンポーネント（Web サービス）が自身の機能やアクセス方法を外部に公開し，他のソフトウェアがネットワーク経由でその機能を利用することができる[1)]．これによりソフトウェアが，必要な機能を提供するサービスをネットワーク上から発見し，連携させて新たなサービスを提供することができる（サービス合成）．サービス合成に関する 1 つの主要なアプローチは，既存のサービスを連携させて新しいサービスを提供するプロセスの定義をあらかじめ人間が与えるものである．このアプローチではあらかじめ利用可能なサービスの種類が定まっている際に，アプリケーション提供者が質の高い合成ロジックを構成し，独自の検索や選択の仕組みを組み合わせて提供できる．この際，どのような順序で既存のサービスを呼び出しその入出力をつなぎ合わせるか，というロジックの決定と実装が中心的な作業となる．この作業に関しては，合成プロセスの標準記述言語として

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

BPEL（Business Process Execution Language）が提案されている[2]．

一方，パーベイシブコンピューティングはモバイルコンピューティングの発展形であり，ユーザの周りに埋め込まれたコンピューティング環境がユーザの気づかないところでユーザを支援する振舞いを行うものである[3]．パーベイシブコンピューティングにおいては，ユーザのモバイルデバイスやその周りの環境に設置されたソフトウェアが，位置検知などのセンサの情報を利用しながらディスプレイなどのデバイスと相互作用していく．その際，それらのセンサやデバイスをソフトウェアとして仮想化する，すなわちそれらを管理するサービスでラッピングし，そのサービス経由でアクセスさせることも想定されている[4)~6)]．これにより多種多様なセンサやデバイスの機種依存性を隠蔽するとともに，サービス指向の枠組みを用いて必要な機能の検索やそれへのアクセスを実現することができる．

パーベイシブコンピューティングにおけるサービス指向ソフトウェアに固有の振舞いとしては，ユーザの位置など状況に基づいて利用サービスを選択したり，状況の変化に応じて現在の処理を中断し利用するサービスを変更したりするものがある．このような振舞いを，メッセージング，検索，イベントや状態の管理といったプリミティブ API を組み合わせてロジックを実装した場合，サービスの合成ロジック，検索や選択のロジック，切替えロジックなどが織り交ぜられた複雑なコードになってしまう．一方，BPEL などサービス指向に基づいたプロセスの記述言語では，個々のパートナに求めるインタフェースのみを宣言した形で合成プロセスを記述し，別途検索や選択の機構を組み合わせることができる[7]．この分離により合成ロジックの再利用性が高まり，サービスの検索や選択に関するカスタマイズが容易となっている．しかし合成プロセスの実行中に状況の変化に応じてパートナを切り替えることは考えられておらず，パートナの参照の変更処理を明示的にプロセスに挿入しなければならない．

そこで本研究においては，サービスの切替えや実行の中断を考慮した形で再利用可能な合成プロセスを定義するためのモデルを示すとともに，状況に応じたパートナの選択や切替えに関する指定を別途与えるための枠組みを提案する．この枠組みを用いることにより，中断時のパートナへのキャンセルの送信なども含めた形で合成プロセスを再利用可能とし，環境や嗜好に応じて容易にパートナの選択や切替えのロジックを与えたり差し替えたりすることができる．本論文の残りではまず 2 章において背景技術と本論文で扱う課題について述べる．次に 3，4，5 章において提案する枠組みと現在の実装を示す．最後に 6，7 章において評価と議論を行い今後の展望を述べる．

2. 課　　題

2.1 背　　景

BPEL は Web サービスの抽象的な，または実行可能な合成プロセスを記述するための標準言語である[2]．BPEL は，各**パートナ**（相互作用の相手）へのメッセージ送受信やデータの操作といったアクティビティ間の制御フローおよびデータフローを定義するための記述を提供する．BPEL においては，各パートナについてそれに求められる型（**ロール**）が宣言される．ロールとは，複数サービス間の連携手順を定めた相互作用プロトコル（コレオグラフィ）において，1 つの参加者が満たすべきインタフェース（期待される可能なメッセージ送受信列）のことである．

BPEL などのサービス合成プロセス記述においては，このロールの概念に基づき相互作用の実装とパートナの検索や決定のロジックとを分離している．たとえば本の購入のためのプロトコルにおいて「購入者」ロールを持つプロセスを実装する際には，「販売者」ロールを持つあるコンポーネントがパートナとなると宣言し，そのパートナに対してのメッセージ送受信をプロトコルに従い実装する．実際のパートナの参照は，検索や選択のための外部の仕組みにより与える[7]．ただし現在の BPEL などの記述では，プロセスの実行中にパートナを決定したり変更したりする場合にはこのように明確な分離はできず，明示的にプロセス内でディレクトリサービスなどを起動して参照を代入するような処理が必要となってしまう．

一方で，パーベイシブコンピューティングにおいては，センサやデバイスをサービスでラッピングし，連携させることが考えられている．パーベイシブコンピューティングにおけるサービス合成のための取り組みとしては，周辺にあるローカルサービスの管理基盤や位置情報や，イベント処理を用いたサービスプログラミングのための枠組み，周辺サービスの意味のある組合せのユーザへの提示などがある[4),6),8)]．しかし，上記のようなサービス指向の合成プロセス記述の枠組みの適用性についての議論はいまだ行われていない．

2.2 例　　題

パーベイシブ環境におけるサービス指向ソフトウェアの例として，地域の情報をモバイルデバイスや近くのディスプレイに表示したり，音楽をモバイルデバイスや近くのスピーカから流したりするアプリケーショ

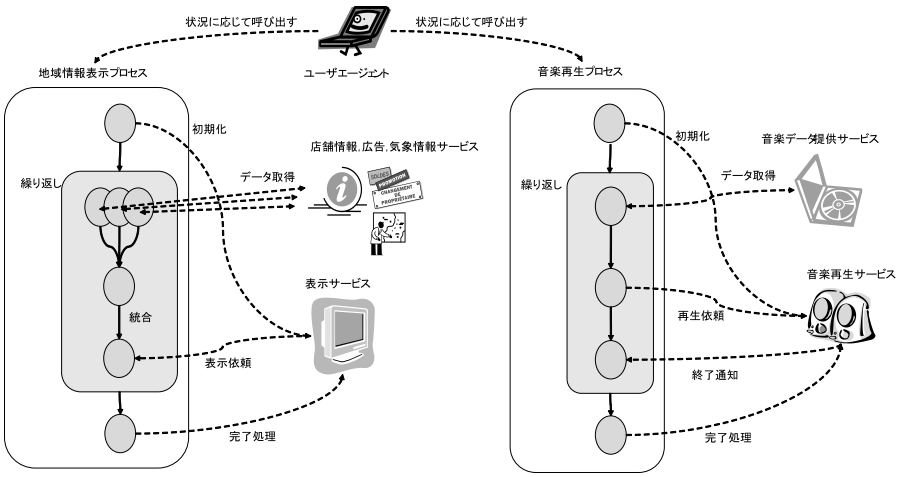

図 1 ユーザを支援するサービス合成プロセス
Fig. 1 Service composition process for user support.

ンを考える．このようなアプリケーションは，情報提供サービスやディスプレイ管理サービス，ユーザの音楽の管理サービスなどを連携させ，新たなサービスを合成するプロセスとして実現することができる．

図 1 に上記のアプリケーションを実現するプロセスの例を示す．実線はプロセスの実行遷移を，破線はサービスやサブプロセスとの相互作用を示している．この例ではユーザエージェントがそのサブプロセスとして，音楽再生や地域情報表示のサービスの合成プロセスを起動している．図では抽象的な合成ロジックのみを示したが，実際には環境や嗜好に対応して各パートナを選択することとなる．たとえば音楽データ提供サービスの候補として次のようなものが考えられる．

- ホームサーバ上で音楽を管理しているサービス．
- モバイルデバイス上で持ち歩いている音楽を管理しているサービスや周りの人の同様のサービス．
- 周辺の販売店の試聴や購入サービス．もしくはインターネット上の同様のサービス．

2.1 節で述べた BPEL 同様の枠組みを用いれば，個々の合成プロセスについては再利用し，環境や嗜好に応じてサービスの参照を与えプロセスを実行することができる．しかし移動するユーザのモバイルデバイス上でこのようなプロセスを走らせる場合，繰り返し音楽を再生する中で状況に応じて利用サービスを切り替えることも考えられる．たとえば，家にいるときはホームサーバ上のサービスを利用し，外にいるときは可能なら周辺の販売店の試聴サービス，そうでなければモバイルデバイス上のサービスを利用することが考えられる．また時間の経過にともないサービスを変更することも考えられる．2.1 節で述べた BPEL などの既存技術ではその場合，合成プロセスをそのまま再利用することはできず，図 1 のプロセスに対し状況を見てパートナを検索，選択するような動作を適切な箇所に挿入することが必要となる．結果できる制御フローは複雑なものとなってしまい，環境や嗜好に応じての選択や切替えのロジックの変更も行いにくい．

さらに，パーベイシブ環境のアプリケーションでは環境の変化による中断やパートナの変更が必要となる．たとえば地域情報を集め表示するプロセスに関し次のような動作が考えられる．

- ユーザの予定表から訪問地域の情報を表示していたが，予定が変更されたので情報を再取得し表示．
- ユーザの移動の際や，故障の際に現在のディスプレイサービスの使用をやめて別のサービスを表示．

このような中断や切替えもまた，合成プロセスをそのまま再利用できずに遷移を追加する必要を生じる．また今のパートナとの相互作用を中止して別のパートナに切り替えるような場合，今のパートナに相互作用プロトコルに応じて正しくキャンセルメッセージなどを送る必要がある．中断や切替えのトリガとなるイベント（状態の変化）は適用環境や嗜好により変化しうるが，プロトコルに従って正しく中断，パートナを変えて再開するという振舞いは再利用できるものであるため，それらの振舞いを意識した形で再利用可能な合成プロセスを記述できるべきである．さらにそのうえで，状態の変化に応じたパートナの切替えを容易に指定できることが望ましい．

3. サービス合成プロセスモデル

本研究では 2.2 節であげた問題に対して，サービス

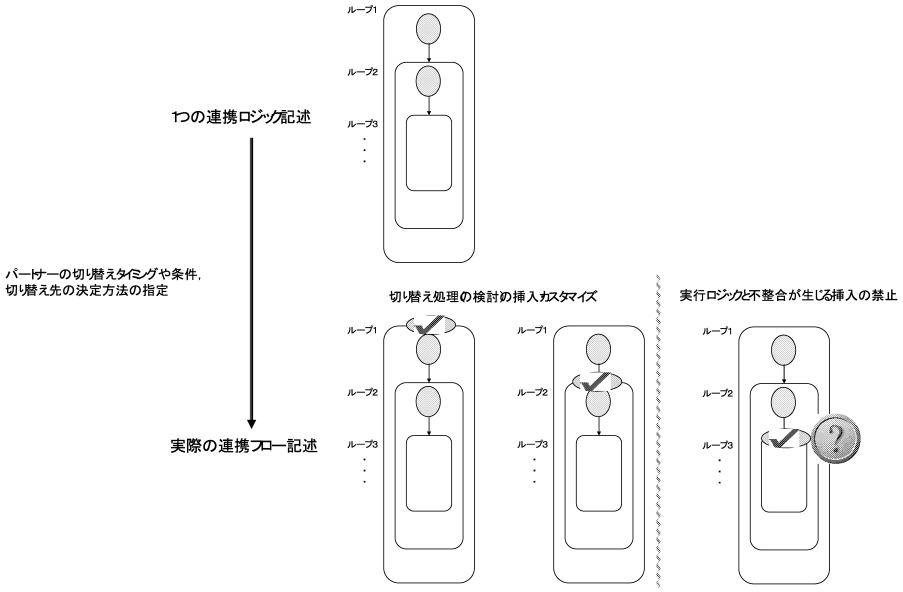

図 2 ループ実行におけるサービス切替え
Fig. 2 Partner switch during loop execution.

の切替えや処理の中断も考慮した形で再利用可能な合成プロセスを記述するモデルや，そのモデルに対して状況に応じてのサービスの選択や切替えを指示するバインディングポリシ記述を提案する．本章では前者の記述モデルについて述べる．

3.1 基盤モデル

サービス合成プロセス記述としては，BPEL[2] や著者らが以前用いたワークフロー記述[9] を想定している．一般的には以下の条件を満たせばよい．

- メッセージ送受信や変数処理，他のプログラミング言語で定義された振舞いの実行などの動作単位を定義することができ，それらの間の制御フローを記述することにより合成プロセスを記述する．本論文では BPEL にならい，その動作単位のことをアクティビティと呼ぶ．
- Web サービスやエージェント技術におけるホスト間の同期または非同期メッセージングを利用する．
- 制御フローは switch や while などの構造制御やグラフ指向の記述（アクティビティ間の実行順序制約と発火条件などの指定）で与えられる．ただし，グラフ記述においてはループは acyclic に限りループは構造化されているとする．すなわち定義されたサブプロセスのループ実行が可能であるとする．
- 2.1 節で述べた，パートナのメッセージングインタフェースの宣言とそれに基づいた相互作用の記述が可能である．特に各アクティビティがどのパートナとの相互作用（の一部）を実装しているかの情報，すなわち各アクティビティ内で相互作用しうるパートナの集合が得られるとする．

3.2 拡　　張

以下では 3.1 節で述べたプロセス記述に対して本研究が行う拡張について述べる．

3.2.1 サービス切替えの制約

プロセスの実行中において繰り返し同じロールを持つサービスと相互作用する場合，状況に応じてループごとにサービスを切り替えることが考えられるが，その切替えタイミングは様々である．たとえば 2.2 節であげた音楽再生プロセス（図 1）の場合，一体の再生サービスに対して繰り返し再生依頼するたびに音楽データ提供サービスを切り替えることもあれば，より外側の音楽再生プロセスの繰返しの実行のたびに切り替えることもある．このような切替えタイミングは一種のサービス選択ロジックであり，音楽再生プロセスを再利用する側が別途指定できることが望ましい．本研究では**図 2** にあるように，従来のようにプロセス記述に直接パートナの切替え処理を埋め込むのではなく，プロセス記述とは別途与える記述によりその埋め込み方を指定できるようにする（その記述の詳細は 4 章で後述する）．これにより，連携ロジックは同一であるが切替え処理のみが異なる複数のフローを，個別のプロセスとしてではなく，同一記述のカスタマイズと見なして扱う．

しかし図 2 にも示されているように，そのようなサービス切替えの挿入位置によっては元々のロジック

と不整合が生じる場合がある．たとえば音楽再生プロセスにおいて，同じ音楽データ提供サービスを繰り返し用いることを前提として選曲を行うようなロジックが埋め込まれている場合である．その場合，音楽データ提供サービスの切替えは，サブプロセスとしての音楽再生プロセスを繰り返し実行するループ，もしくはそれよりも外側のループにおいてしか許されない．本研究では，再利用可能な合成プロセスの記述者がこのような制約を明示的に記述するため，パートナ宣言のメタデータとして切替えが許されるループアクティビティの名前を指定するように拡張する（パートナ宣言内に AllowedSwitchLoop 要素を追加する）．相互作用が完了していない時点でのパートナ切替えは不整合を招くため，AllowedSwitchLoop で指定されたループは，そのパートナと相互作用するアクティビティをすべて内包しなければならないこととする．またこの記述がなかった場合のデフォルトの値は，この内包の条件を満たすループのうち，最も内側のものとする．合成プロセスの提供者はこの指定を用い，サービスの切替えの可否も考慮してプロセス記述を行う．

3.2.2 中断処理

上記のループごとの切替えとは別に，現在の実行を中止してロールバックし，パートナを切り替えてやり直すような処理も考えられる．このような処理もまた一種のサービス選択ロジックであり，音楽再生プロセスを再利用する側が別途指定できることが望ましい．ここで合成プロセス記述において中断された際のキャンセル処理などを記述しておけば，それを再利用したうえで，中断のトリガや中断後のサービスのパートナ選択ロジックのみを別途カスタマイズすることができる．

中断時の処理の記述のためには，合成プロセス記述内においてアクティビティの集合をスコープとして定義し，そのスコープの実行中に外部から中断やサービス切替えの指示があった際の処理およびその後の実行遷移を記述する．本論文では以下の 2 種類の処理を考える．

OnAbort プロセスの実行が外部から中断された際の処理を記述する．この処理が終わるとプロセスの実行は終了する．

OnSwitchForced パートナの即座の切替えが外部から要求された際に呼ばれる処理で，パートナごとに与えることができる．この処理が終わるとプロセスの実行はスコープの最初から再開する．

実際にこれらの処理を与える際には，一般的な try/catch 型の例外処理記述と同様の記述を用いることができる．いい換えると上記の処理は，中断やパートナの切替えに対する外部からの要求という「例外」に対する catch ハンドラを与えていると見なすことができる．本研究の目標はこのようなハンドラの記述形式自体ではなく，それを状況の変化に対応する共通処理の記述に積極的に活用する記述モデルの提案である．

注意すべき点として，ここでの例外はたとえば Java のそれとは異なり，プロセスの外部からプロセスに与えられるものである．BPEL の場合，元々プロセス自体に並列性が許されているため，例外発生時に実行中であるアクティビティの中止の意味論が定義されている．たとえば invoke などのメッセージ処理アクティビティを実行中である（送信しようとしているまたは受信待ちである）場合，すぐにキャンセルされる（受信の場合その後受信したメッセージは破棄される）．このことより BPEL で上記 2 つのハンドラの記述を行いたい場合，対応する Fault の定義を加えるだけで，文法上は BPEL 自体の catch 記述を用いることができる．本研究のモデルは特に言語に依存しないが，BPEL のように外部から与えられる例外に対する意味論が定義されている言語ではその意味論を用いることとする．そうでない場合，例外処理は現在実行中のアクティビティの実行後に起動されるとして意味論を定めて拡張する☆．

図 3 に，2.2 節における音楽再生プロセスにおけるこれらの処理の定義例を示す．ここでは BPEL のアクティビティ記述，およびその実行中断意味論を用いている．アクティビティは下記のように省略して表している．

invoke/receive(パートナ識別名,
オペレーション名)

パートナ識別名の dataServiceProvider，playServiceProvider はそれぞれ，2.2 節における音楽データ提供サービス，音楽再生サービスを指している．

またプロトコルに関する仮定として音楽データ提供サービスは単発の起動のみで特にキャンセル処理などは要求しないとする．音楽再生サービスは完了通知を必要とし，また再生中の場合はその前に再生停止を必要とするとする．図 3 においては，音楽データ提供サービスに関し，実行中止やパートナ切替えの要求を受けた際（AbortFault や SwitchForcedFault が発生した際）に，上記の再生停止依頼および完了通知を行っている☆☆．パートナ切替えの際には，実行はルー

☆ たとえばアクティビティが Java で定義されたアクションの実行である場合，外部からそのスレッドの実行を強制的に止めることは推奨されていないため．

☆☆ 上記の説明と対応するため，OnAbort に対応する例外を AbortFault と呼んでいるが，実際には，BPEL 自体において定義されている ForcedTermination という Fault を用いることができる．

```
while(...){
  scope{
    sequence{
      invoke(playServiceProvider, initialize);
      while(…){
        invoke(dataServiceProvider, obtainData);
        invoke(playServiceProvider, play);
        scope{
          receive(playServiceProvider, finish);
          catch(AbortFault f){
            invoke(playServiceProvider, stop);
            throw f;
          }
          catch(SwitchForcedFault
                  (partner=playServiceProvider) f){
            invoke(playServiceProvider, stop);
            throw f;
          }
        }
      }
      invoke(playServiceProvider, finalize);
    }
    catch(AbortFault f){
      invoke(playServiceProvider, finalize);
      throw f;
    }
    catch(SwitchForcedFault(partner=playServiceProvider) f){
     invoke(playServiceProvider, finalize);
    }
  }
}
```

図 3 実行中断時の処理
Fig. 3 Handlers for execution abort.

プの最初に戻り改めて行われている．本研究の提案する枠組みではパートナの参照管理は実現基盤によって行われる．この場合，SwitchForcedFault に対する一番外側の catch 節の実行が終わった場合に，（再検索なども含めて）参照の書き換えが行われる．

4. バインディングポリシ記述

本章では，前章で示したモデルに基づく合成プロセス記述に対し，個々のパートナの検索や選択（以下バインド処理と呼ぶ），切替えに関する制御を与えるためのバインディングポリシ記述（以下ポリシ記述）を示す．詳細な文法は付録 A.1 で示すが，以下の各節においてポリシ記述の構成要素について説明する．

4.1 環境モデル

バインディングポリシ記述では，「ユーザが移動したら」「今自宅にいるならば」のようにイベントや環境の属性の値に応じて，サービスの選択や切替えを指定する．バインディングポリシ記述ではイベント発生の通知と環境の属性の読み取りが可能であることを要件とする．実際には実現基盤において，イベントの種類や取得可能な環境の属性として，場所や時間などの語彙を定めて用いることとなる．

4.2 切替えループ

本研究では繰り返されるサービス合成において状況に応じてサービスを選択していくことを考えている．このためポリシ記述では，合成プロセスにおいてパートナの切替えが起こりうるループや切替えが起きる条件を以下の形式で指定することができる．

```
<SwitchLoop name=”アクティビティ名”>
  <SwitchCondition>
    切替えを行う条件
  </SwitchCondition>
</SwitchLoop>
```

SwitchLoop で指定されたアクティビティは繰り返し実行されるアクティビティ（たとえばサブプロセスを繰り返し実行する while アクティビティ）に限定し，指定されたループをそのパートナの切替えループと呼ぶ．切替えループはそのパートナの AllowedSwitchLoop であるか，それを内包するループでなければならない．切替えループ内の処理が繰り返し実行されるたびにパートナの切替えが検討される．実際に切替えが行われるかは，SwitchCondition で指定された条件によって決まる．

Everytime ループのたびにつねにパートナを再バインドする．

OnEvent 指定されたイベントが起きた際，次のループ実行時にパートナを再バインドする．

OnTimePassage 前回にパートナを決定してから指定された時間が経過したら，その次のループ実行においてパートナを再バインドする．

4.3 中断をともなうサービス切替え

上記のループ実行におけるサービスの切替えとは別に，状況の変化に応じて現在の実行を中断しサービスを切り替えて再開する場合がある．この処理は ForcedSwitch 要素により指定することができる．この要素内では上記の OnEvent 要素を記述し，どのようなイベントが発生したときに切替えを行うかを指定する．そのイベントが発生したときには，現在のプロセス実行を中断する（3.2.2 項の OnSwitchForced 処理）．

4.4 バインド処理の起動タイミング

ポリシ記述ではバインド処理の起動タイミングを BindingTiming 要素により次の中から指定することができる．

Initial 合成プロセスとポリシ記述が与えられ起動された際に，バインド処理を行ってからプロセスの実行を開始する．

OnDemand プロセス全体を通して，または切替えループの実行においてループの最初に戻った後に，そのパートナと相互作用するアクティビティが初めて実行可能になった際にパートナの検索，選択を行う．

Ahead 指定されたアクティビティの実行が終わったときに指定された条件が成り立っていればパートナの検索，選択を前もって行っておく．もし前もっての処理が行われていない場合 OnDemand 同様に必要になり次第行う．

OnDemand では，実際に必要になってからパートナのバインド処理を行うため，実行時の分岐によっては必要ではないパートナに対しての処理を行わずに済む．しかしバインド処理を待って実行がブロックされてしまう可能性がある（それらの処理を待たなくても実行可能なアクティビティがあれば先に実行できる）．そこで開発者は Ahead を用いて前もっての（別スレッドでの）バインド処理を行うようチューニングすることができる．

4.5 バインド処理の詳細

パートナの検索，選択に関しては QoS を考慮したブローカによる選択など非常に様々な手法が提案されている[10]．バインディングポリシ記述からはそれらの手法を実現するコンポーネントの起動方法などを指定する．ただし，状況に応じて検索，選択手法を変えることができるように，バインド処理の記述は次のような構造をとっている．

```
<Bindings>
  <DefaultBinding>
    <BindingMethod>
      バインド処理詳細指定
    </BindingMethod>
  </DefaultBinding>
  <Binding>
    <OnCondition>
      この手法を用いる条件
    </OnCondition>
    <BindingMethod>...</BindingMethod>
  </Binding>
  <Binding>...</Binding>
  ...
</Bindings>
```

この構造では，様々な環境属性の条件に対応するバインド処理（Binding 要素）と，それらの条件がいずれも成り立たない場合のデフォルトのバインド処理（DefaultBinding 要素）が指定されている．条件を指定する OnCondition 要素では「自宅ならば」など環境属性に対する条件を書くほか，trigger 属性においてサービス切替えのトリガを指定することもできる．これにより，指定されたトリガでサービスが切り替えられるときにそのバインド処理が有効になる．このトリガの指定としては，SwitchLoop 内の SwitchCondition で指定した OnEvent などの名前を参照する．複数の BindingCondition が成立する場合，最も上に記述されたものが適用される．バインド方法の指定は実装基盤に依存するが，本研究で実装したフレームワークにおける指定について 5.3 節で後述する．

5. フレームワーク実装

本研究では 3，4 章で述べた合成プロセス記述モデルおよびバインディングポリシ記述に基づき，サービスを利用，提供，合成するソフトウェアエージェントの実装フレームワークを Java を用いて実装した．本章ではこの実装について述べる．

5.1 実行基盤

実装したフレームワークでは，エージェントの生成や実行，メッセージング機構やエージェントの検索のためのディレクトリサービスなど，一般的なエージェントシステムの基盤機能を提供している．このフレームワークでは，Place と呼ばれるエージェントの実行コンテナに対して環境情報を管理するエージェントを関連づけることにより，環境に関するイベントや属性の値の管理を行わせている．本論文の執筆時点ではこのような環境管理エージェントをフレームワークとして標準的に提供しておらず，アプリケーションに応じて位置などの環境管理を行うエージェントを Place に対して関連づけ，また Place に対してイベントの発生や環境の属性値の変更を登録するようにする．Place 上で動作するエージェントは，フレームワークの提供する API を用いてそれらのイベントの発生通知を受け取ったり環境の属性の値を読み取ったりすることができる．この環境管理エージェントの導入により，その環境を利用する側のエージェントがセンサを管理するエージェントなどの存在を意識することなく，抽象化された環境情報を利用して処理を行うことができる．

実行基盤は合成プロセス記述と各パートナに対するバインディングポリシ記述を受け取って，4 章で説明したようにバインド処理を適宜挿入しながら合成プロセスを実行していく．これらの記述の実装詳細については次節以降で述べる．

5.2 合成プロセス記述

合成プロセス記述としては 3.1 節であげた要件を満たすものとして著者らの以前の研究[9] で用いた記述を拡張したものを用いている．この記述はソフトウェアエージェントの実装のための記述であり，メッセージング機構としてはエージェント間通信言語の標準（FIPA ACL）[11] を用いている．拡張点としては，

3.2 節で述べた追加記述の導入と，イベントによるアクティビティの発火や環境の属性に対する条件に基づいた遷移制御の導入がある．また，サブプロセスの実行を非同期に行えるようになっている．ここで実行中のサブプロセスに対して実行中止を指示することもできる．その際には 3.2.2 項で述べた OnAbort 処理が呼ばれる．

5.3 バインディングポリシ記述

バインディングポリシ記述は基本的には 4 章で述べたとおりであるが，実装依存とした部分について説明する．各バインド処理を起動する条件記述については，環境の属性に対する条件文だけでなく，合成プロセスの実行状態（変数の値など）に対する条件文も用いることができる．また 4.5 節において，バインディング方法の詳細についてはバインド処理を行うコンポーネントの起動方法を与えるとした．以下では実装されたフレームワークで可能としている指定について述べる（これらの文法は付録 A.1 で示す）．

5.3.1 静的な指定

StaticBinder 指定によりサービスの参照を静的に与えることができる．これは明確な好みにより使うサービスが定まっている場合などにも用いることができるが，テスト時に非常に有効である．

5.3.2 デフォルトバインダ

DefaultBinder 指定によりフレームワークの提供するデフォルトの検索，選択コンポーネント（バインダ）を用いることができる．このバインダではバインド処理が次の手順で行われる．

検索 合成プロセスで記述されたロールの要件に加えて「質が一定以上」など最低限要求する属性条件を与え，それらの条件を満たすサービス提供者を検索する．

選択 検索したサービス提供者の中から与えられた効用関数を用いて，指定された数のより望ましいサービス提供者を絞り込む．

交渉 入札制度を模倣した契約ネットプロトコル[12]などを実装したコンポーネントを与え，パートナ候補と相互作用，交渉して，パートナとするものを一体決定する．これはディレクトリサービスに公開された情報だけではなく要求を伝え合ってサービスの質を決めるような場合に用いる．たとえば，計算資源などそのときの負荷によって配分（サービスの質）が変わるような場合にその情報を得てサービス提供者を選んだり，トラスト交渉[13]を行って段階的に個人情報を送って認証・認可を行ったりすることを想定している．

パートナ決定までの流れを**図 4** に示す．上記 3 つのステップはすべて行わなければならないわけではなく，それぞれについて指定があった場合のみに行われる．さらに，avoidSameProvider 属性を指定することにより，前回と異なるサービス提供者を選ぶように指定することもできる．これはユーザに見せる情報など変化が求められる場合や特定の提供者の嗜好などに偏らないことが求められる場合に用いることができる．この要素のとりうる値は次のとおりである．

```
if 検索条件が与えられている
  then その条件とロール条件を両方満たす
       提供者を検索した結果のリストを L とする
else
  ロール条件を満たす提供者を検索した結果のリストを L とする
if avoidSameProvider 属性が yes に設定されている
  then L から前回選んだ提供者を除く
else if yesifpossible に設定されている
  then L の長さが 1 でなければ L から前回選んだ提供者を除く
if 効用関数が与えられており，
   L の要素数が指定された絞り込む数 n よりも多い
  then L の各要素に対し効用関数を呼び出し，
       効用が高い順に並べた上位 n 個の提供者のリストを L とする
if 交渉コンポーネントが与えられている
  then L の要素を渡して交渉を行わせる
    if 返り値として提供者の指定が返った
      then その提供者ををパートナとする
    else
      例外を投げる
else
  if L が空ではない
    then L の先頭の提供者をパートナとする
  else
    例外を投げる
```

図 4 パートナバインド処理の手順
Fig. 4 Partner binding procedure.

yes けっして前回と異なるサービス提供者を選ばない．

yesifpossible 基本的に前回と異なるサービス提供者を選ばないが，その提供者以外に利用できない場合はその提供者を選ぶ．

no 特に制約を設けない．

5.3.3 外部バインダ

PluginBinder 指定により，自身で実装したバインダを用いることができる．これは内部でネットワーク上の外部サービスを起動するものも含む．この指定では，フレームワークが提供する抽象クラスを実装する形で，サービス提供者の参照を返すメソッドを実装する．バインディングポリシ記述内では，この実装クラスとそれに与える引数を指定する．

5.4 サービス障害

パーベイシブコンピューティングにおいては，利用していたサービスが突然利用不可能となる場合がある．実装されたフレームワークにおいては，利用していたサービスの応答がなくなった場合に，パートナの切替え処理を起動する．すなわち 3.2.2 項の OnSwitch-Forced 処理を起動する．

6. 評　　価

提案されたフレームワークは，パートナのバインディング処理の挿入のカスタマイズをするためのプロセス記述の拡張（3 章）と，そのバインディング処理の詳細を定めるポリシ記述（4 章）からなる．本章ではこれらのそれぞれに関し，実現される再利用性や柔軟性，またその導入の負荷について議論する．さらに，関連研究との関連についても述べる．

6.1 プロセス記述の拡張による再利用性・導入の負荷

6.1.1 ループ実行におけるパートナ切替え

ループ実行におけるパートナの切替えに関し，2.2 節であげた音楽再生プロセスの場合，BPEL など既存の記述では以下のいずれかのみを記述することができる．

- 1 つの再生サービスに対して 1 つの音楽データ提供サービスを繰り返し用いるプロセス．
- 1 つの再生サービスに対して音楽データ提供サービスを変えながら用いるプロセス（バインダサービスを呼び出して音楽データ提供サービスを変更する処理を，ある地点に明示的に挿入した場合）．

本研究の提案モデルの導入における効果は既出の図 2 に示されている．上記のように切替えタイミングが異なるプロセスを複数書くのではなく，1 つのプロセス記述に対して切替えタイミングを別途ポリシ記述で与えるようにしている．このことより，切替えタイミングのカスタマイズに対して連携ロジックを再利用可能となっている．

ただし，図 2 からも分かるように，ここで切替え処理を実現するアクティビティを 1 つ織り込むとしても記述上の手間はそれほど変わらない．しかし切替え処理の挿入位置によっては元来のプロセスロジックと不整合が生じる可能性があるため，本研究ではプロセス記述者が明示的にその挿入位置についての制約を記述できるようにした（AllowedSwitchLoop 記述）．プロセスロジックをよりよく理解する記述者がこの制約を与えることは，特に環境や嗜好に応じてプロセスの記述者とは別の人間がサービスの切替えを指定する際に有効である．

一方で，プロセス記述者側がこの制約について検討，正しく記述する必要がある．本研究においては，デフォルトの AllowedSwitchLoop をプロセスの解析の結果提供することにより，相互作用が完了する前にパートナが切り替えられてしまうような不整合を避けるようにしている．しかしプロセスの意味に応じて制約を強める必要がある場合，プロセス記述者が明示的に意識，プロセスを解析する必要がある．

6.1.2 実行中断をともなうパートナ切替え

提案したフレームワークでは，プロセスの外部からの実行中断やサービス切替え要求に対してのハンドラ処理を記述するモデルを提案した．ここで 3.2.2 項で示したように，この記述方式自体は既存の例外処理のものを用いることができる．しかし実行時の望ましくない失敗に対する対処と復帰だけではなく，イベントなどに応じて実行を中断，ロールバック，またはスキップするために積極的にハンドラ機構を用いている点が異なる．**図 5** (a) に示すように，これらのハンドラにおいてプロセス記述者は相互作用プロトコルに応じた適切な取消し処理などを記述する．これらの処理の記述は，様々な中断や切替えのトリガの導入，変更に対して再利用することができる．

上記は本質的には設計の指針の提案であるが，本研究で提案したような指針がなく，プロセス記述者が通常 BPEL プロセス記述で与えるような連携ロジックの記述のみを行う場合を考える．このとき図 5 (b) のように，環境や嗜好に応じてサービスの切替えや選択ロジックのみを変更する側がそれぞれに，相互作用プロトコルを強く意識，理解して適切な取消し処理を実装する必要が生じてしまい，負担が大きい．また注意深く設計しなければ，中断や切替えの各トリガごとにハンドラを与えてしまい，共通する処理が各ハンドラに重複して現れるような実装を行ってしまう可能性もある．

実行中断やサービス切替え要求に対してのハンドラ処理の実装の際には，網羅性，特に相互プロトコルの遵守に関して注意する必要がある．本研究ではこの点に関して支援がなされていないため，既存のツール[14]を導入するなどして，どのような実行パスでも相互作用プロトコルから期待されるメッセージ動作が正しく行われるような実装になっているかどうか，検証する必要がある．

6.2 提案ポリシ・フレームワークによる容易性・多様性

提案フレームワークにおいては，上記で述べたプロセス記述に対してバインディングポリシ記述を追加することにより挿入されるバインド処理の詳細を与え，それを解釈実行する．このポリシ記述を用いた場合と従来の BPEL などを用いた場合における実装の手順を比較する．**図 6** は，条件分岐に応じて 2 種類のパートナと相互作用する簡単なプロセスに対し，実行時に必要となったパートナを与えられた属性条件に基づき検索する際の実装を表している．提案フレームワーク

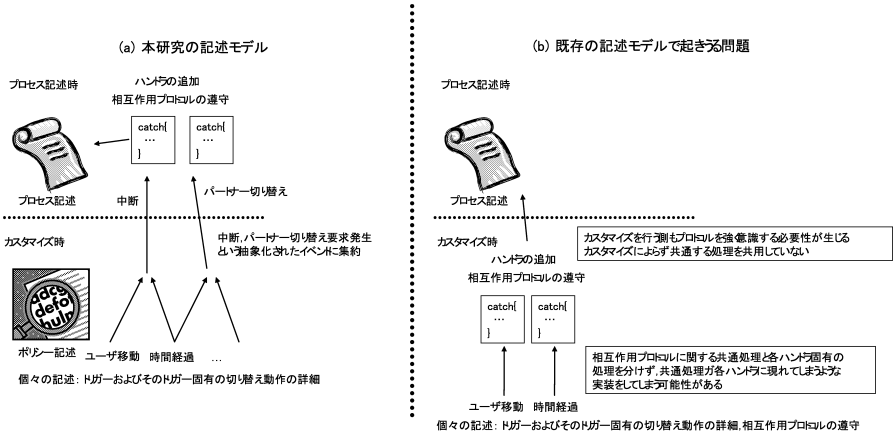

図 5 実行中断をともなうパートナ切替えにおける記述モデル
Fig. 5 Description models for partner switch with execution abort.

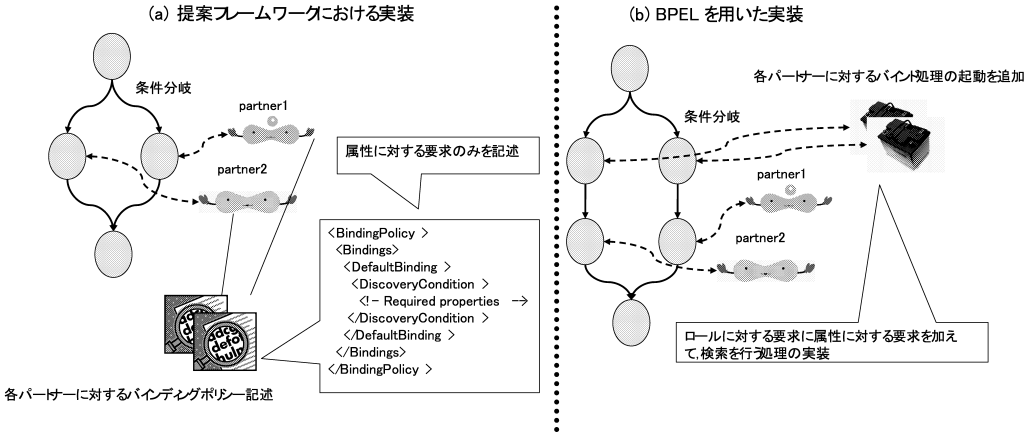

図 6 バインド処理を含む振舞いの実装
Fig. 6 Implementation of behavior including binding.

を用いた場合（図 6 (a)），連携ロジックを記述したプロセスに対し，個々のパートナに対するバインディングポリシとして検索に用いる条件のみを与えればよい．ここでポリシで記述していない要素のデフォルトの振舞いとして，各パートナは実際に必要になってから（条件分岐が判断されてから）バインドされる．またその前段階において，ポリシを与えないことによりロール条件のみで検索，相互作用する振舞いをテストすることも容易である．これに対し BPEL など既存の記述に対する実行エンジンを用いた場合，この挿入されるバインド動作をプロセスに挿入する必要がある（図 6 (b)）．BPEL の場合はこの振舞いは invoke アクティビティの挿入となり，外部コンポーネントとしてバインド動作を行いパートナの参照を返すものを準備することとなる．このコンポーネントにおいては，プロセス記述からの制約としてのロールに属性条件を加えて検索する振舞いを実装する必要がある．この振舞いにおいて実際の属性条件以外はアプリケーション間に共通する振舞いであるため，提案フレームワークにおいてはこの振舞いを提供し各アプリケーション開発者の実装の手間を軽減させている．

上記の効果はバインディングポリシ記述において支援されている振舞いについてのものであるため，実際の有効性はポリシ記述の柔軟性（多様性）に依存する．本論文で示した現在のフレームワーク実装においては，パートナの切替えタイミングやそのトリガに加え，パートナの検索，選択方法の指定が可能となっている．

ここで検索，選択に関しては，公開された属性情報に基づきより良い相手を選ぶための制約条件および評価関数の導入，また公開されていない情報や固定されていないサービスの質に基づきより良い相手を選ぶ

ための交渉の導入が可能となっている．このように，サービス選択が公開情報のみに基づき行われる場合とさらなる交渉などを通して行われる場合の双方の実現を可能としている．ただし，典型的と考えられる特定の評価関数や交渉手法のライブラリ提供など，実現の容易性に関する向上の余地が残っている．また本研究のアプローチでは，合成プロセスとバインド処理の記述を分離している．そのためこれらのロジックが密接に関連する場合にうまく扱えない場合が出てくる．そのような場合の 1 つとして，まずオークションのようにパートナの参照を渡す仲介者も 1 つのパートナとして相互作用プロトコルに含まれている場合がある．このような場合はバインディングポリシ記述を関連づけず，合成プロセス中において仲介者から受け取った参照を代入する処理を実装することとなる．もう 1 つの場合として，パートナの選択の際に外部のバインダを呼ぶだけでなく，選択後もサービス利用の経験をフィードバックとして送ることが必要になるような場合である．応答時間などを扱う場合には実行基盤側が監視すればよいが，値段などアプリケーション依存の項目を考える場合[15]，合成プロセス内で明示的に経験をデータとして保存しなければならない可能性がある．そのような場合には合成プロセスに手を加える必要が生じてしまう．

また本研究ではプロセス記述の再利用性について扱ったが，バインディングポリシ記述の再利用性については未検討である．同記述においては最低限の要件として状況やその変化（イベント）に応じた処理の記述を可能としているが，可能な状況（の変化）の種類の爆発の問題，その網羅性や一般性（再利用性）の評価の問題などは扱っていない．本研究ではサービス指向プロセス記述をパーベイシブ環境に適用する際の課題について検討，解決を行ったが，上記のようなパーベイシブ環境においてオープンな課題に取り組む必要性が残っている．

6.3 関 連 研 究

著者らの以前の研究では，パートナとの相互作用の直前や直後に，ホスト間の移動や，協調的な振舞いに関する合意形成処理などを挿入することを提案した[9]．しかしこの枠組みでは，挿入される移動や協調的な振舞いに主眼をおいており，たとえばパートナの選択については，静的に与えられた条件を満たすサービス提供者を探しその中からランダムに選ぶのみであり，サービスの切替え時など状況に応じて条件を変えたり効用関数を用いたりすることができなかった．またプロセスの中断についても考慮していなかった．

本論文では合成プロセス記述を人間が与え，サービス提供者の選択などを状況に応じて行わせる枠組みを提案した．サービス合成のもう 1 つのアプローチとしては，合成プロセス自体をサービスの意味論から推論するアプローチがある．このアプローチに基づき，またパーベイシブコンピューティングを対象としたものとしてタスクコンピューティングがよく知られている[4]．タスクコンピューティングでは，ユーザのモバイルデバイスや周辺にあるサービスの組合せで意味のありそうなものをユーザに提案する．ユーザのファイルをプロジェクタに写すなど，比較的単純なタスクをユーザの指示で行う場合このような枠組みが適している．これに対しアプリケーションの魅力として比較的複雑な合成ロジックを開発者が与えユーザを能動的に支援できるようにする場合，本研究で扱ったプロセス記述のアプローチが適している．

合成プロセス記述においてサービス提供者のダウンなどに備えて例外処理を記述するべきだ，という考え方はすでに提案されている[16]．本研究ではパーベイシブ環境において，予測に基づいた先読み処理や状況に応じた処理の状況の変化による取消し，特にサービス切替えを容易とするために，例外処理に似た記述を用いることを提案した．

サービスの検索や選択に関しては，第三者として QoS を保証するブローカを用いるものや，サービス利用者の経験を蓄積し利用するものなど様々なものがある[10),15]．しかしそれらは 1 回 1 回の選択のための機構に着目したものであり，本研究のようにそれらの機構を起動するタイミングを与えたり，状況に応じて用いる機構や引数を変更したりすることは考えられていない．

7. お わ り に

本研究においては，サービス指向コンピューティングでのアプローチをパーベイシブコンピューティングに適用するための提案を行った．その提案としては，まずサービスの切替えや実行の中断を考慮した形で再利用可能な合成プロセスを定義するためのモデルを定め，次に状況に応じたパートナの選択や切替えに関する指定を別途与えるためのポリシ記述を導入した．また，これらの提案に基づき，パーベイシブコンピューティングにおけるサービス連携を行うエージェントの開発のためのフレームワークを実装した．このフレームワークを用いることにより，中断時のサービスへのキャンセルの送信なども含めた形で合成プロセスを再利用可能とし，環境や嗜好に応じて容易にサービス

の選択や切替えのロジックを与えたり差し替えたりすることができる．今後はこのフレームワークを用いて様々なアプリケーションを構築し，その経験を通して洗練化していく．

参 考 文 献

1) Haas, H.: Web Services (2004). http://www.w3.org/2002/ws/ (Access: May 2005).
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) Satyanarayanan, M.: Pervasive Computing: Vision and Challenges, *IEEE Personal Communications*, pp.10–17 (2001).
4) Masuoka, R., Parsia, B. and Labrou, Y.: Task Computing — The Semantic Web meets Pervasive Computing, *2nd International Semantic Web Conference* (*ISWC 2003*), pp.866–881 (2003).
5) Kalasapur, S., Kumar, M. and Shirazi, B.: Seamless service composition (SeSCo) in pervasive environments, *The 1st ACM international workshop on Multimedia service composition*, pp.11–20 (2005).
6) Robinson, J., Wakeman, I. and Owen, T.: Scooby: middleware for service composition in pervasive computing, *The 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pp.161–166 (2004).
7) Mandell, D.J. and McIlraith, S.A.: Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, *2nd International Semantic Web Conference* (*ISWC 2003*), pp.227–241 (2003).
8) Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K.: A Middleware Infrastructure for Active Spaces, *IEEE Pervasive Computing*, Vol.1, No.4, pp.74–83 (2002).
9) 石川冬樹，吉岡信和，田原康之，本位田真一：階層構造制御に注目したモバイルエージェントフレームワークとそのマルチメディア応用，電子情報通信学会論文誌「ソフトウェアエージェントとその応用」特集号，Vol.88-D-I, No.9, pp.1402–1417 (2005).
10) Serhani, M.A., Dssouli, R., Hafid, A. and Sahraoui, H.: A QoS Broker Based Architecture for Efficient Web Services Selection, *IEEE International Conference on Web Services* (*ICWS'05*), pp.113–120 (2005).
11) FIPA: Agent Communication Language (1997). http://www.fipa.org/specs/fipa00003/OC00003A.html
12) FIPA: FIPA Iterated Contract Net Interaction Protocol Specification (2002). http://www.fipa.org/specs/fipa00030/SC00030H.html
13) Seamons, K.E., Winslett, M., Yu, T., Yu, L. and Jarvis, R.: Protecting Privacy during On-Line Trust Negotiation, *Privacy Enhancing Technologies 2002*, pp.129–143 (2002).
14) Quenum, J., Ishikawa, F. and Honiden, S.: Interaction Design in Agent-based Service-oriented Computing Systems, *AAAI Workshop on AI-Driven Technologies for Services-Oriented Computing* (2006).
15) Sensoy, M. and Yolum, P.: A Context-Aware Approach for Service Selection Using Ontologies, *5th International Joint Conference on Autonomous Agents and Multiagent Systems* (*AAMAS2006*), pp.931–938 (2006).
16) Baresi, L., Ghezzi, C. and Guinea, S.: Towards Self-healing Service Compositions, *PRISE'04, 1st Conference on the PRInciples of Software Engineering* (2004).

付 録

A.1 バインディングポリシ記述文法

4 章で提案したバインディングポリシ記述の文法の概要を EBNF により示す．ただし，ここでは小文字で始まるシンボルは，プログラミング言語などにおける命名規則の慣習など一般的な表現を利用するものや，実装手段に依存するものであるため定義を省略している．

BindingPolicy :=
<BindingPolicy partner="*partnerId*">
+*SwitchLoop*
+*ForcedSwitch*
+*BindingTiming*
+*Bindings*
</BindingPolicy>

SwitchLoop :=
<SwitchLoop name="*activityName*">
<SwitchCondition>
SwitchCondition
</SwitchCondition>
</SwitchLoop>

ForcedSwitch :=
<ForcedSwitch>*OnEvent*</*ForcedSwitch*>

OnEvent :=

```
    <OnEvent name="name">
        eventConditionExpression
    </OnEvent>
SwitchCondition :=
    <EveryTime/>
    | OnEvent
    | <OnTimePassage>timeSpan</OnTimePassage>
BindingTiming :=
    <Initial/>
    | <OnDemand/>
    | <Ahead activity="activityName"/>
Bindings :=
    <Bindings>
        <DefaultBinding>
            bindingMethod
        </DefaultBinding>
        Binding*
    </Bindings>
Binding :=
    <Binding>
        <OnCondition Trigger+>
            conditionalExpression
        </OnCondition>
        bindingMethod
    </Binding>
Trigger := trigger="triggerName"
```

5.3 節で述べた現在の実装におけるデフォルトの検索，選択に関する記述の文法を以下に示す（上記文法における *bindingMethod*）．

```
bindingMethod := Static | Default | Plugin
Static :=
    <StaticBinder>serviceReference</StaticBinder>
Default :=
    <DefaultBinder avoidSameProvider="AvoidType">
        Discovery+ Selection+ Negotiation+
    </DefaultBinder>
AvoidType := yes | yesifpossible | no
Discovery :=
    <DiscoveryCondition>
        serviceConditionalExpression
    </DiscoveryCondition>
Selection :=
    <SelectionCondition number="positiveInteger">
        SelectionMethod
    </SelectionCondition>
SelectionMethod :=
    <Random/>
  | <RatingFunction>javaClassName</RatingFunction>
Negotiation :=
    <NegotiationComponent>
        javaClassName
    </NegotiationComponent>
Plugin :=
    <PluginBinder>
        <BinderClass>javaClassName</BinderClass>
        <Arguments>argumentsExpression</Arguments>
    </PluginBinder>
```

(平成 18 年 5 月 23 日受付)
(平成 19 年 1 月 9 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．2004 年同修士課程修了，博士課程進学．2007 年同博士課程修了見込み．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．

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．1995 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 年より国立情報学研究所産学官連携研究員，2004 年より同研究所特任助教授，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 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，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兼任，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．2005 年度パリ第 6 大学招聘教授．早稲田大学客員教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．ソフトウェア工学，エージェント技術，ユビキタスコンピューティングの研究に従事．IEEE，ACM 等各会員，日本ソフトウェア科学会理事，情報処理学会理事を歴任．日本学術会議連携会員．