モバイルエージェントアプリケーションのための仕様記述言語 Pigeon

田原　康之[†]　　大須賀昭彦[†]　　本位田真一[††,†††]

Specification Language Pigeon for Mobile Agent Applications

Yasuyuki TAHARA[†], Akihiko OHSUGA[†], and Shinichi HONIDEN[††,†††]

あらまし　モバイルエージェント技術が普及するにつれて，実適用にあたり，例えば開放型のネットワーク上を動き回るため動作の把握が困難であることや，セキュリティの確保が従来のシステムに比べて難しいなど，多くの問題が浮かび上がってきている．このような問題を解決する手段として，形式仕様技術が注目されており，モバイルエージェント向けの仕様記述言語や，そのための理論的基盤となる計算モデルがいくつか提案されている．しかし，モバイルエージェントアプリケーションの仕様を効果的に記述・利用するためには，その記述言語にもカスタマイズ機能や高度な記述能力が要求されるが，従来の言語では対応していない．そこで本論文では，モバイルエージェントアプリケーションの仕様記述に必要な，カスタマイズ機能や高度な記述能力を実現した言語 Pigeon を提案する．Pigeon は，仕様記述言語として，動作仕様と要求仕様の分離や，前者から後者の体系へのモデル化を変更可能とすることにより，カスタマイズ機能を実現する．また，リフレクション機能により，高度な記述能力を実現する．また本論文では，Pigeon を電子カタログアプリケーションに適用し，更に他の言語と比較することにより，Pigeon の評価を行う．

キーワード　モバイルエージェント，仕様記述言語，リフレクション，セキュリティ

1. まえがき

インターネットやイントラネットなどの広域開放型ネットワークが普及するにつれて，ネットワーク上を移動しながら作業を行うソフトウェアである，モバイルエージェントの技術が注目を集めてきている．しかし本技術が普及するにつれて，実適用にあたり，例えば開放型のネットワーク上を動き回るため動作の把握が困難であることや，セキュリティの確保が従来のシステムに比べて難しいなど，多くの問題が浮かび上がってきている．このような問題を解決する手段として，形式仕様技術が注目されており，Mobile UNITY [11] などのモバイルエージェント向けの仕様記述言語や，そのための理論的基盤となる計算モデルがいくつか提案されている．しかし，それらの言語や計算モデルとしては，従来のモデルに移動のためのプリミティブや，その他必要な構成要素をアドホックに導入したものがほとんどである．一方，モバイルエージェントアプリケーションの仕様を効果的に記述・利用するためには，言語に次のような特徴が求められる．まず，モバイルエージェントの基本機能について，例えば移動機能には強い移動と弱い移動の区別があり，またエージェント間通信機能にも同期・非同期の区別があるなど，様々なバリエーションが存在する．したがって，言語にもそのようなバリエーションに対応するための機能として，例えば高度なカスタマイズ機能などが必要になる．また，記述能力として，例えばセキュリティの仕様記述，及び検証が容易でなければならない．そのため，セキュリティ要求の仕様記述に必要な時相論理や様相論理の記述，及び移動を含んだエージェントの動作の正しさが検証できるようなモデル化能力が必要である．しかし，これらのような高度な機能を備えた言語はまだ提案されていない．

そこで本論文では，これらモバイルエージェントアプリケーションの仕様記述に必要な特徴を備えた言語 Pigeon を提案する．Pigeon では，動作仕様を要求仕様の論理体系でモデル化し，更に，このモデル化を変更可能とすることにより，Pigeon 言語の高度なカス

[†]（株）東芝研究開発センター，川崎市
Research & Development Center, TOSHIBA Corporation, Kawasaki-shi, 212–8582 Japan
[††]国立情報学研究所，東京都
National Institute of Informatics, Tokyo, 101–8430 Japan
[†††]東京大学，東京都
The University of Tokyo, Tokyo, 113–8656 Japan

タマイズを実現する．更に，論理体系の部分にリフレクション機能をもつ RL/R を採用することにより，時相論理や様相論理を取り扱えるなど，高度な記述力を実現する．

本論文の構成は次のとおりである．**2.** では，Pigeon を開発した背景について説明する．**3.** では，Pigeon の言語仕様の詳細を述べる．**4.** では，Pigeon の評価を，例題への適用と他言語との比較を通じて行う．**5.** では，結論と今後の課題について述べる．

2. Pigeon 開発の背景

本章では，モバイルエージェントアプリケーションのための仕様記述言語として Pigeon を開発した背景について述べる．開発にあたっては，解決すべき問題を考慮すると，次のような要件を満たすことが必要となる．

- カスタマイズ機能

モバイルエージェントプラットホームは，現状ではまだデファクトともいえるものが存在せず，アプリケーションや動作環境によって適切な選択肢が様々異なってくることが多い．そのため，モバイルエージェントの基本機能について，例えば移動機能には強い移動と弱い移動の区別があり，またエージェント間通信機能にも同期・非同期の区別があるなど，様々なバリエーションが存在する．したがって，本論文が目指す言語にも，これらのバリエーションを取扱うことのできるだけの高度なカスタマイズ機能が必要となる．

- 記述能力

モバイルエージェント技術においてはセキュリティが最も重要な課題の一つであり，またそのほかにも移動機能に伴う様々な要求が課せられる．そのため，これらの要求を十分に記述できるような言語である必要がある．例えば，セキュリティ要求の仕様記述には，時相論理や様相論理を取り扱う必要がある．

以上の要件を満たすため，Pigeon を次のように設計した．

- Pigeon の仕様記述は，プログラム的記述の動作仕様と，論理体系により記述される要求仕様の 2 部分に分かれる．前者の記述は，後者の体系でモデル化される．更に，このモデル化は変更可能であり，これにより Pigeon 言語の高度なカスタマイズが実現される．
- Pigeon の論理体系の部分は，リフレクション機能をもつ RL/R を採用している．これにより，時相論理や様相論理を取り扱えるなど，高度な記述力を実現する．

3. Pigeon 言語の詳細

本章では，Pigeon 言語の構文や意味論について詳述する．Pigeon は，通常の仕様記述言語と同様に，構文規則と意味論をもつが，カスタマイズ機能と高度な記述能力を実現するため，次のような構成を行っている．

- 仕様記述を動作仕様と要求仕様の二つの部分から構成する．動作仕様は，システムの動作を記述しやすい構文とし，要求仕様は RL/R と呼ばれる論理体系により宣言的に記述する．
- 意味論も，動作仕様の RL/R でのモデル化と，RL/R の意味論の二つの部分に分け，前者を変更可能としている．これにより，カスタマイズ機能を実現する．

3.1 Pigeon の構文

Pigeon による仕様記述は，Mobile UNITY と同様に，動作仕様と要求仕様の二つの部分から構成される．

動作仕様の BNF による記述は次のようになる．

```
behavior-spec ::=
        "parameter" symbol {"," symbol}* ";"
        | agent-definition ";"
        | "location" symbol{"(" symbol
                        {"," symbol}* ")"}
                      {"," symbol}* ";"
        | {reference "="} expression ";"
        | "if" expression block {"else" block} ";"
        | "while" expression block ";"
        | "try" block "catch (" symbol ")" block ";"
        | "throw" expression ","
        | "go" expression ","
        | "return" expression ";"
        | behavior-spec behavior-spec
        | "/*" comment "*/"
        | "//" comment-without-newlines "\n"

agent-definition ::=
        "agent" symbol"@"symbol{"(" expression
                                  {"," expression}* ")"}
                "{" {[attribute-definiton
                      | method-definition ";"}* "}"

attribute-definition ::=
        symbol {"=" expression}

expression ::=
        primitive-expression
        | reference "(" {expression
                         {"," expression}* ")"

reference ::=
        symbol{"."symbol}*

method-definition ::=
        symbol"(" {symbol {"," symbol}*} ")" block ";"
```

```
block ::= "{" behavior-spec "}"
```

以下，本記述の詳細について説明する．

- 動作仕様は，文の列である．文には，定義文，要素文，複合文，及びコメント文がある．
- 定義文で定義する対象には，パラメータ，エージェント，及び場所の 3 種類がある．
- 要素文には，代入文，`throw` 文，`go` 文，及び `return` 文がある．なお，`go` 文の実行により，エージェントは `expression` の値となる場所に移動する．すなわち，Pigeon で扱うエージェントは移動機能をもつ．
- 代入文は，式の値を，エージェントの属性を表す「参照表現」に代入する．エージェント `ag` の属性 `attr` を表す参照表現は，`ag.attr` となる．
- 複合文は，`if` 文，`while` 文，及び `try` 文に分かれる．
- エージェント定義は，名前，(パラメータ化も可能な) 初期場所，属性，及びメソッドの定義から構成される．

要求仕様は，リフレクティブ書換え論理 RL/R [14] で記述する．RL/R の概要は次のとおりである．

- RL/R は，リフレクション機能を追加した書換え論理 [8], [9] である．

書換え論理では，データを表す項の同値類間の書換え関係 $[t] \to [t']$ を，書換え理論と呼ばれる公理系と，推論規則により導出する．ここで，項の同値関係 $t_1 = t_2$ は等式論理により導出される．

一方，RL/R では，項の同値類と書換え理論の対の間の書換え規則 $([t]_{\mathcal{R}}, \mathcal{R}) \to ([t']_{\mathcal{R}'}, \mathcal{R}')$ を導出する．ここで，$[t]_{\mathcal{R}}$ は，書換え理論 $\mathcal{R}$ で許される項 t の $\mathcal{R}$ における同値類を表す．

- RL/R では，ベースレベル記号の可算無限集合 $\mathcal{S}$ をあらかじめ与える．そして，これらベースレベル記号と，各ベースレベル記号のメタ表現 `s` のメタ表現は `'s`) とを記号として用いる．
- 項 t や書換え理論 $\mathcal{R}$ に対しても，それぞれのメタ表現 $\uparrow(t, \mathcal{R})$ 及び $\uparrow\mathcal{R}$ が定義される．
- 特別な関数 `stateCon` を用いた下記の規則によりリフレクションが実現される．

$([t]_{\mathcal{R}}, \mathcal{R}) \to ([t']'_{\mathcal{R}}, \mathcal{R}') \Leftrightarrow [\texttt{stateCon}(\uparrow(t, \mathcal{R}), \uparrow\mathcal{R})]_{\mathcal{R}} \to [\texttt{stateCon}(\uparrow(t', \mathcal{R}'), \uparrow\mathcal{R}')]_{\mathcal{R}}$ in $\mathcal{R}$

Pigeon では，RL/R のリフレクション機能により高度な記述能力が実現される．例えば，時相オペレータ `always` は，リフレクションを用いて次のように実現される．

```
always(Cond, InitState) = holds(Cond, State)
  if reachable(InitState, State) = true

reachable(State, State) = true
reachable(State1, State2) = true
  if rewrites(State1, State3) = true,
     reachable(State3, State2) = true,
```

ただし，`T1` が `T2` に書き換えられるならば，またそのときに限り `rewrites(T1, T2) = true` である．

3.2 意 味 論

Pigeon の意味論は二つの部分に分かれる．一つ目は RL/R による動作仕様のモデル化で，二つ目は RL/R の意味論である．そして前者は開発者が定義・変更することが可能である．

一つ目は RL/R による動作仕様のモデル化は次のとおりである．

- まず動作仕様記述を構文解析し，結果の構文木を RL/R の項「`parseTree(`左の枝, 右の枝`)`」で表す．
- システムの時系列変化を，システム状態項と呼ばれる項の書換えで表す．システム状態項は，「`systemState(` コンフィギュレーション, 継続, 動作仕様の構文木`)`」の形をとる．ただし，
  - コンフィギュレーションは，Maude 言語 [8] と同様のもので，エージェントやメッセージを表す項を，ACI（結合的，可換，かつ単位元をもつ）オペレータで結合したものである．
  - 継続は，ある時点での残りの処理を構文木の形で表したものである．

Pigeon では，このシステム状態項の書換え規則を，開発者が定義・変更することができる．

動作仕様のモデル化の例を以下に示す．

```
systemState(C, parseTree(";",
    parseTree("while", ..., Cond, ...,
      Block), Continuation), Program)
-> systemState(C, parseTree(";",
       parseTree("if", Cond,
         parseTree(";", Block,
       parseTree("while", ..., Cond, ...,
         Block)), ...,
     Continuation), Program)
```

本書換え規則は，`while` 文を実行した場合の 1 ステッ

プの状態変化を表す．詳しくは，while 文を実行するには，条件判断を if 文で実行し，条件が成り立てば while 文の中のブロックを実行後再び while 文の実行を繰り返す．条件が成り立たなければ，while 文の後の処理（Continuation で表される継続）を実行する．

次に，モデル化の部分を変更することにより，柔軟なカスタマイズが可能になる例を示す．エージェントの移動が弱い移動と強い移動の 2 種類に分けられることが知られている [7]．両者の違いは，強い移動ではコードと実行状態の両方が移動するのに対し，弱い移動ではコードのみ移動する点である．そこで，移動を表す go 文のモデル化を変更することにより，この 2 種類を使い分けられることを示す．まず強い移動のモデル化は以下のとおりである．

```
systemState(C, parseTree(";",
       parseTree("go", Host), Continuation),
       Program)
-> systemState(C', Continuation, Program)
```

ここで，C' は C においてエージェントが Host に移動した場合のコンフィギュレーションを表す．このモデル化では，移動後にも残りの継続がそのまま使われることにより，実行状態が維持されることが表現される．一方，弱い移動は次のようにモデル化される．

```
systemState(C, parseTree(";",
       parseTree("go", Host), Continuation),
       Program)
-> systemState(C', Continuation', Program)
```

ここで Continuation' は，Continuation において，エージェントの継続を初期の状態，つまりプログラム全体に戻したものを表す．これにより，移動前の実行状態が破棄されることが表現される．

意味論の二つ目の部分，すなわち RL/R の意味論は，文献 [14] で説明されている．概略は次のとおりである．

- 本意味論は，書換え論理のカテゴリー理論に基づく意味論 [9] をもとにしている．カテゴリーとは，書換え関係や関数合成などをモデル化する代数構造をもつ数学的対象で，これへのマッピングにより意味論が与えられる．
- RL/R に対する意味領域は，リフレクションを表す構造の入ったカテゴリーである．この構造は，具体的には，意味領域のカテゴリーと，そこから改めて構成された，ベースレベル構造を表すカテゴリーとの間の同型写像（カテゴリー理論ではファンクタと呼ばれる）である．

4. 電子カタログ例題に基づく Pigeon の評価

本章では，モバイルエージェントのための仕様記述言語としての Pigeon を，例題に基づいて，他言語と比較することにより評価する．例題として，電子カタログの検索・収集・登録システムにおける，セキュリティ要求の仕様記述を取り上げる．

我々は，エージェントプラットホームとして Bee-gent [12] を適用することにより，モバイルエージェントが，ネットワーク環境において，企業間電子商取引で扱われる，製品や部品の電子カタログ情報に対し，自動的に検索，収集，及び登録プロセスを実行するシステムを開発している．Bee-gent は，電子カタログ DB などの既存アプリケーションを，エージェントラッパーによりエージェント化する．エージェントラッパーと後述の仲介エージェントは，国際標準のエージェント間通信言語 XML/ACL で通信する．更に，電子カタログシステムの利用手順を組み込み，ネットワーク上を移動しながら当該利用手順を実行する仲介エージェントにより，システム利用を自動化している．

図 1 に，Bee-gent を適用した電子カタログの検索・収集・登録システムの概念図を示す．本システムでは，まず各電子カタログ DB のためのエージェントラッパーにより，各 DB のインタフェースの違いを吸収し，XML/ACL によって検索や登録操作が可能となるようにする．そして，仲介エージェントには，各 DB が稼動しているサイトを巡回し，検索/収集作業を実行して，最後にもとのサイトに帰って登録作業を実行する手順を組み込んでいる．

本適用例では，電子カタログサービス提供者 (eCatalogServiceProvider)，及び二つの製品メーカ，メーカ 1 (manuf1) とメーカ 2 (manuf2) が存在しているものとする．Bee-gent 適用においては，eCatalogServiceProvider はモバイル仲介エージェント plibRetriever を生成し，以下の作業を実行させる．

- plibRetriever は生成された後，エージェント操作担当者から検索要求を受け取る．
- plibRetriever は，manuf1，manuf2 の順でネットワークを巡回し，eCatalogServiceProvider

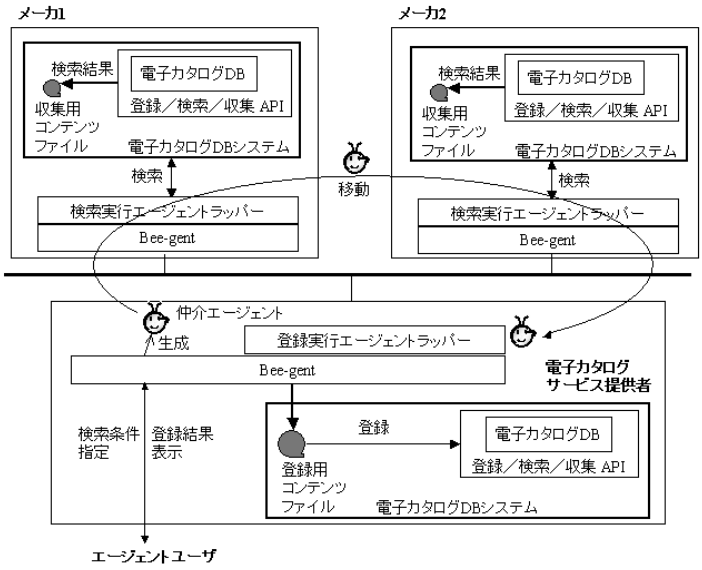

図 1　電子カタログの検索・収集・登録システム
Fig. 1　E-catalog retrieval, gathering and registration system.

に戻る．

- 各製品メーカにおいて，plibRetriever は要求されたデータを検索する．検索されたデータはその量によって，メーカのサーバに一時的に格納されるか，またはエージェントによって運ばれる．
- eCatalogServiceProvider に戻ると，plibRetriever はデータをeCatalogServiceProvider のサーバ上の電子カタログ DB に登録する．

本適用例においては，多くのセキュリティ要求が存在する．本論文では，次のような認証の問題を取り上げる．すなわち，正当な権限をもたないホストが，モバイル仲介エージェントに誤った検索結果を渡し，その結果誤ったデータが登録される，ということを防ぎたいという要求である．このような問題の解決のために，モバイル仲介エージェントが訪れたサーバを認証する必要がある．そこで，認証機構によって正当な認証が実現できるかどうかを，我々の手法で検証することとする．正当な認証というのは，モバイル仲介エージェントが正当なサーバから結果を受け取り，登録を行っていれば認証が成功し，そうでなければ認証は失敗する，という意味である．

以下，本例題の仕様記述を，Pigeon 及び Mobile UNITY で行った例を示す．更に，Pigeon の長所を，他の言語，Mobile UNITY，Ambient Calculus，及び Join Calculus と比較することにより検討する．

4.1　Pigeon による記述

本例題の Pigeon による記述例を付録 **1.** に示す．なお，付録 **1.1** で示したように，Pigeon 言語では移動の強弱のカスタマイズが可能であるため，それぞれに対応した動作仕様記述を示した．本記述例において，動作仕様は次のように構成されている．

- パラメータの Manuf2Signature は manufacturerSite(2) の署名を表す．パラメータで表すことにより，manufacturerSite(2) が悪意のあるホストになり済まされた状態も表現できる．
- 場所は eCatalogServiceProviderSite と manufacturerSite(i) (i = 1, 2) がある．更に，同じ仕様をもつエージェントが，manufacturerSite(i) のそれぞれに存在している．本エージェントは，retrieve() メソッドにより，電子カタログ DB への問合せを処理する．
- モバイル仲介エージェント plibRetriever には，いくつかの属性とメソッドがある．属性 state は，要求仕様に対する検証に用いられる．retrieve() メソッドでは，plibRetriever は製品メーカ間を移動し，各 DB のコンテンツを収集し，eCatalogServiceProviderSite に戻って，署名の認証が成功すれば収集したコンテンツを DB に登録する．

要求仕様記述は，本例題に対するセキュリティ要求の仕様を表す．なお，簡単のためソートは省略している．要求仕様は次のものから構成される．

- 最初の三つの等式は，ディジタル署名機構を実現する関数への要求を表す．
- 四つ目の等式は，すべてのコンテンツが正しいサイトから収集された場合の要求を表す．省略された部分 (...) は初期状態の記述である．
- 最後の等式は，悪意をもってなり済ましたサイトからコンテンツを収集してしまった場合の要求を表す．

4.2 Mobile UNITY による記述

次に，Mobile UNITY による仕様記述を示すが，その前に Mobile UNITY の概要を説明する．

Chandy と Misra は, [4] において，並列・並行プログラムの仕様を記述するための言語として UNITY (Unbounded Nondeterministic Iterative Transformations) を提案した．Mobile UNITY は，UNITY をモバイルコンピューティング向けに拡張したものである．

UNITY の記述は，プログラミング記述（以下，簡単のため「プログラム」と記す）とプログラミング論理（以下，簡単のため「論理」と記す）の二つに分けられる．

Mobile UNITY プログラムは，システムの挙動を記述するものである．プログラムは，`System ... end` で囲まれ，プログラムの動作を変数への代入による状態遷移で表す．プログラムは，program 部，Components 部，及び Interaction 部から構成される．

Mobile UNITY の論理における命題は，最小の構成要素である表明，またはその限量化（限量子とプログラムの文 $\forall s$，$\exists s$ を追加）から，時相論理の演算子 $\longmapsto$ などで構成される．

Mobile UNITY では，要求仕様の検証は，要求仕様を表す命題を証明することにより行う．証明は，公理として与えられる正しい表明（$\{p\}s\{$ true $\}$ など）に，一階述語論理の推論規則と $\longmapsto$ の定義（文献 [4] p.52）を適用することにより行う．

本例題の Mobile UNITY による仕様記述を付録 **2.** に示す．本仕様記述は，文献 [13] から引用したものである．

4.3 仕様記述言語の比較

次に，Pigeon とその他の 3 言語との比較を，電子カタログ例題の記述も参考にして実施する．比較の観点は次のとおりである．

- まず，モバイルエージェントアプリケーションのための仕様記述言語として特に重要な特徴として，**2.** で取り上げたカスタマイズ機能と記述能力について比較と評価を行う．
- 更に，以下のような一般的な観点について，各言語の紹介文献の記述をもとに比較を行う．
  - 一般に仕様記述言語として重要な特徴：

    計算モデル
    意味論
    手続き型，あるいは宣言的記述などの，各種計算パラダイムに対する記述力
    検証機能

  - モバイルエージェントアプリケーションのための仕様記述言語として重要な特徴：

    エージェント移動の強さ
    エージェントや非エージェントアプリケーション間の協調機能
    オブジェクト指向機能．例えば，モジュール化，カプセル化，及び継承など

  - その他特筆すべき特徴をもつ言語については，その特徴．

以下，Pigeon 及び Mobile UNITY と比較する各言語の概要を述べる．

4.3.1 Ambient Calculus

Ambient Calculus [2], [3] は，π-計算 [10] などと同様の，並行計算の形式的モデルであり，エージェントやホストなどを表すことのできる ambient と呼ばれる構成要素をもつことにより，モバイルエージェントを表現できることを特徴とする．同様な形式的モデルの中では，最も普及しているものの一つである [1].

Ambient Calculus は，π-計算と同様に，通信チャネルや変数などを表す「名前 (name)」を基本要素とし，プロセス間送受信や並行合成オペレータなどから構成されたプロセスの遷移（変形，reduction）により，システムの挙動を表す．また，特徴的な要素である ambient は，ひとまとまりのプロセスに名前を付けてカプセル化したものである．ambient はプロセスの構成要素になれるので，ambient を入れ子にすることができる．更に，子 ambient は親 ambient を出入りすることができる．特に，エージェント ambient がホスト ambient を出入りすることにより，モバイルエー

ジェントが表現できる．

4.3.2 Join Calculus

Join Calculus [5], [6] は，やはりモバイルエージェントを表現できる並行計算の形式的モデルである．Join Calculus は，Reflexive CHemical Abstract Machine (RCHAM) という計算モデルに基づいている．更に Join Calculus では，モバイルエージェントを表現するために，RCHAM を強化した DRCHAM (Distributed RCHAM) をベースとする．

4.4 比較の結果

まず，本例題の仕様記述も利用して，Pigeon と他の言語に関し，**2.** で取り上げた，カスタマイズ機能と記述能力を評価し，本例題のセキュリティ要求の記述と検証の可能性と容易性を論じる．

- カスタマイズ可能性

Pigeon では，**3.** で示したように，動作仕様の RL/R でのモデル化を変更することにより，移動のための go オペレータの意味を，強い移動と弱い移動のどちらにも使用することが可能である．したがって，付録 1.1 に示した 2 通りの動作仕様記述のいずれに対しても，セキュリティ要求の検証が可能である．一方 Mobile UNITY では，移動は場所変数の値の変更でモデル化されているため，強い移動しかサポートされておらず，本例題で必要なカスタマイズが容易ではない．他の 2 言語も，移動の前後ではエージェントの稼働場所以外の情報は保存されることにより，強い移動しかサポートしていないので，同様である．

- 記述能力

Pigeon では，**3.** に示したように，RL/R のリフレクション機能を用いて時相論理オペレータを記述でき，更にそのオペレータによって，例題のセキュリティ要求を記述し，検証することが可能である．Mobile UNITY や Ambient Calculus でもやはり時相論理オペレータを用いて同様の記述が可能である．したがって，Pigeon は，本例題に関してはこれらの言語と同等の記述能力により，セキュリティ要求の記述・検証

表 1 比較の結果
Table 1 Results of the comparison.

比較対象 / 比較項目	Mobile UNITY	Ambient Calculus	Join Calculus	Pigeon
移動方式	強(実行状態は変数の値として保持される)	強	強	強、弱両方
オブジェクト指向仕様記述	モジュール化・カプセル化あり	モジュール化・カプセル化あり(ambientで表現すれば)	モジュール化・カプセル化あり(locationで表現すれば)	モジュール化、カプセル化、継承
検証能力	全手動	半自動(自動型検査、手動証明)	半自動(自動型検査、手動証明)	手動証明
協調方式	通信チャネル変数	通信ポート	Join Patternによるパターンマッチ	様々な方式が可能(言語仕様定義に依存)
基礎計算モデル	状態遷移モデル	π 計算	化学抽象機械(Chemical Abstract Machine)	書換え論理
意味論	公理的意味論	時相論理、双模倣	双模倣	代数的意味論
計算パラダイム記述能力	手続き型	手続き型、関数型(ただし、独自の定義要)	手続き型、関数型(ただし、独自の定義要)	様々な方式が可能(言語仕様定義に依存)
その他記述能力	種々の同期プリミティブ、時相論理	階層的ロケーション	階層的ロケーション	リフレクション(時相論理や様相論理が表現可能)
その他特徴	仕様記述言語としての特徴(モジュール構造、変数宣言など)	普及度最高、実行系(Ambit)	型推論機能付き実行系(Join Calculus Language, JoCAML)	言語仕様のカスタマイズ

が可能である．

次に，その他の観点からの比較を行った結果を表 1 に示す．

表で示した以外に，下記のコメントを追加する．

- Mobile UNITY に関するコメント

– 本言語は最初から仕様記述言語として設計されているので，モジュール化やカプセル化のための構成要素は十分用意されている．

– エージェントの移動は，場所変数の値の変化という非常に簡単なモデルで表されている．その代わり，様々な種類の移動が表現できるかどうかが不明である．

- Ambient Calculus と Join Calculus に関するコメント

– これらの言語は，それぞれ ambient と「場所」という，局所性 (locality) の概念をもっており，これによりエージェントとホストの両方を表現できる．ambient の方が，Join Calculus の場所よりも単純である．

– これらの言語は，仕様記述言語としては整備されていない．特に，新たなプリミティブを導入しない限り，協調機構や関数定義などの高度な概念を記述するのは難しい．

- Pigeon に関するコメント

– Mobile UNITY と同様に，Pigeon は最初から仕様記述言語として設計されている．例えばモジュール化とカプセル化のためのプリミティブをもつ．

– Pigeon のカスタマイズ機能により，移動の強弱や，同期・非同期メッセージ交換といった，様々な種類の移動や協調機構を記述することができる．

5. むすび

本論文では，モバイルエージェントアプリケーションの仕様記述に必要な，カスタマイズ機能や高度な記述能力を実現した言語 Pigeon を提案した．Pigeon は，仕様記述言語として，動作仕様と要求仕様の分離，及び前者から後者の体系へのモデル化を変更可能とすることにより，カスタマイズ機能を実現している．また，リフレクション機能により，高度な記述能力を実現している．また本論文では，Pigeon を電子カタログアプリケーションに適用し，更に他の言語と比較することにより，Pigeon の評価を行った．

今後は，Pigeon の適用経験を増やすことにより，言語のさらなる改良や，実アプリケーションの実装支援，及び検証手法の検討，並びにカスタマイズのガイドラインなど，方法論の整備などを行う予定である．

文　献

[1] Wikipedia: Ambientcalculionline. http://www.wikipedia.com/wiki/AmbientCalculiOnline.

[2] L. Cardelli, “Mobility and security,” in Foundations of Secure Computation, NATO Science Series: Computers & Systems Sciences, ed. F.L. Bauer and M.C. Mult, pp.3–37, 2000.

[3] L. Cardelli and A.D. Gordon, “Mobile ambients,” ed. M. Nivat, Proc. FoSSaCS’98, LNCS1378, pp.140–155, Springer, 1998.

[4] K.M. Chandy and J. Misra, Parallel Program Design – A Foundation, Addison Wesley, 1988.

[5] C. Fournet and G. Gonthier, “The reflexive CHAM and the join-calculus,” Conference Record of POPL’96, pp.372–385, ACM Press, 1996.

[6] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy, “A calculus of mobile agents,” Proc. CONCUR’96, pp.406–421, Springer-Verlag, 1996.

[7] A. Fuggetta, G.P. Picco, and G. Vigna, “Understanding code mobility,” IEEE Trans. Softw. Eng., vol.24, no.5, pp.342–361, 1998.

[8] J. Meseguer, “Rewriting logic and maude: Concepts and applications,” ed. L. Bachmair, Proc. RTA 2000, vol.1833 of LNCS, pp.1–26. Springer, 2000.

[9] J. Meseguer, “Conditional rewriting logic as a unified model of concurrency,” Theor. Comput. Sci., vol.96, pp.73–155, 1992.

[10] R. Milner, Communicating and Mobile Systems: The Pi-Calculus, Cambridge University Press, 1999.

[11] G-.C. Roman, P.J. McCann, and J.Y. Plun, “Mobile UNITY: Reasoning and specification in mobile computing,” ACM Trans. Softw. Eng. Methodol., vol.6, no.3, pp.250–282, July 1997.

[12] （株）東芝．Bee-gent WWW ページ．http://www2.toshiba.co.jp/beegent/

[13] 田原康之，大須賀昭彦，本位田真一，“IPEditor 開発ツールと Mobile UNITY 言語の適用によるモバイルエージェントセキュリティの実現，” 情処学論，vol.43, no.6, pp.1582–1597, 2002.

[14] 田原康之，粂野文洋，大須賀昭彦，本位田真一，“オブジェクト指向におけるリフレクションの代数的意味論，” 情処学論，vol.38, no.4, pp.826–834, 1997.

付　録

1. Pigeon による記述例

1.1 動作仕様

1.1.1 強い移動の場合

```
parameter Manuf2Signature;

location eCatalogServiceProviderSite;

while (i <= 2) {
  location manufacturerSite(i);
```

```
  agent manufacturer@manufacturerSite(i) {

    signature = "";
    result ="";

    retrieve(Query) {
      ... /* 検索手続き */
      result = sign(result, signature);
      return result;
      };
    };
  };

manufacturer@manufacturerSite(1).signature = "manuf1";

manufacturer@manufacturerSite(2).signature
  = Manuf2Signature;

agent plibRetriever at eCatalogServiceProviderSite {

  retrievalRequest = "";
  retrievalResult = "";
  retrievalRequestElement = "";
  nextDestination = "";
  nextQuery = "";
  state = "Wait";

  retrieve(givenRetrievalRequest) {
    state = "Running";
    retrievalRequest = givenRetrievalRequest;
    while (retrievalRequest =\= "") {
      retrievalRequestElement
        = getRetrievalRequestElement(retrievalRequest);
      nextDestination
        = getDestination(retrievalRequestElement);
      nextQuery = getQuery(retrievalRequestElement);
      retrievalRequest
        = updateRetrievalRequest(retrievalRequest);
      go(nextDestination);
      addResult(retrievalResult,
        manufacturer.retrieve(nextQuery));
      };
    go("eCatalogServiceProviderSite");
    if (verifySignature(retrievalResult,
          givenRetrievalRequest))
         == "success" {
      register(retrievalResult);
      state = "Done";
      } else {
      state = "Failed";
      };
    };

  verifySignature(result, request) {
    ...
    if (verify(resultElem, signature) = false) {
      return false;
      }
        /* 結果の各要素の署名を確認 */
    ...
    return true;
    };
    ...   /* 他のメソッド定義 */
  };

plibRetriever.retrieve(...);
```

1.1.2 弱い移動の場合

強い移動の場合の記述に対し，plibRetriever エージェントの動作仕様を，次のように変更.

```
agent plibRetriever at eCatalogServiceProviderSite {

  retrievalRequest = "";
  retrievalResult = "";
  retrievalRequestElement = "";
  nextDestination = "";
  nextQuery = "";
  state = "Wait";
  currentLocation = eCatalogServiceProviderSite
        /* 現在位置 */

  retrieve(givenRetrievalRequest) {
    if (state == "Wait") {
      state = "Running";
      retrievalRequest = givenRetrievalRequest;
      goAndRetrieve();
    } else if (state = "Running"
        and retrievalRequest =\= "") {
      addResult(retrievalResult,
        manufacturer.retrieve(nextQuery));
      goAndRetrieve();
    } else if (currentLocation
        =\= eCatalogServiceProviderSite) {
      go("eCatalogServiceProviderSite");
    } else if (verifySignature(retrievalResult,
        givenRetrievalRequest)) == "success" {
      register(retrievalResult);
      state = "Done";
    } else {
    state = "Failed";
    };
  };

  goAndRetrieve() {
    retrievalRequestElement
      = getRetrievalRequestElement(retrievalRequest);
    nextDestination
      = getDestination(retrievalRequestElement);
    nextQuery = getQuery(retrievalRequestElement);
    retrievalRequest
      = updateRetrievalRequest(retrievalRequest);
    currentLocation = nextDestination;
    go(nextDestination);
  };

  verifySignature(result, request) {
    ...
    if (verify(resultElem, signature) = false) {
      return false;
      }
        /* 結果の各要素の署名を確認 */
    ...
    return true;
```

```
    };

  ...   /* 他のメソッド定義 */
  };
```

1.2 要求仕様

```
var Text, Signer, NonSigner,
  MaliciousManuf2Signature .

eq verify(sign(Text, Signer), Signer) = true .

eq verify(sign(Text, NonSigner), Signer) = false
     if (NonSigner =\= Signer) = true .

eq decipher(sign(Text, Signer), Signer) = Text .

eq eventually(equal(getValue(
                    concreteState(systemState,
     [(Manuf2Signature, "manuf2")],
     "plibRetriever.state"),
     "Done")), ...) = true .

eq always(equal(getValue(concreteState(systemState,
     [(Manuf2Signature, MaliciousManuf2Signature)],
     "plibRetriever.state"), "Done")), ...) = false
     if (MaliciousManuf2Signature =\= "manuf2")
         = true .
```

2. Mobile UNITY による記述例

2.1 プログラム

```
System eCatalog[Manuf2Signature : string]
  ...
  program manuf2 at "manuf2"
    declare
        plibRetrieverChannel : string
     [] plibRetrieverChannelStatus : string
     [] state : string
     [] signature : string
    initially
        ...
     [] signature = Manuf2Signature
    assign
        ...
     [] state, plibRetrieverChannel,
          plibRetrieverChannelStatus
        = "resultsSent," ...
        /* 署名された検索結果を表す ACL メッセージ */
        , "sent"
          react-to state = "retrieved"

  program plibRetriever at lambda
    declare
        eCatalogServiceProviderChannel : string
     [] manuf1Channel : string
     [] manuf2Channel : string
     [] eCatalogServiceProviderChannelStatus : string
     [] manuf1ChannelStatus : string
     [] manuf2ChannelStatus : string
     [] state : string
    assign
        ...
     [] lambda = "eCatalogServiceProvider"
          react-to manuf2ChannelStatus = "sent"
        ...
  end

  Components
     eCatalogServiceProvider
     [] manuf1 [] manuf2 [] plibRetriever

  Interactions
     ...
     [] manuf2.plibRetrieverChannel,
          manuf2.plibRetrieverChannelStatus
        = plibRetriever.manuf2Channel,
            plibRetriever.manuf2ChannelStatus
        when plibRetriever.lambda = "manuf2"
     [] ...
end
```

2.2 論理記述

```
forall Manuf2Signature
  ((Manuf2Signature = "manuf2" and p) leads-to
     eCatalogServiceProvider.state = "registered")

invariant (forall Manuf2Signature
  ((Manuf2Signature =\= "manuf2" and p)
   -> eCatalogServiceProvider.state =\= "registered"))
```

ただし，p は次の論理式を表す．

```
forall Text, Signer, NonSigner : string (
verify(sign(Text, Signer), Signer) = 1
and NonSigner =\= Signer
    -> verify(sign(Text, NonSigner), Signer) = 0)
```

（平成 14 年 9 月 3 日受付，15 年 1 月 10 日再受付）

田原　康之

1989 東大・理・数学卒．1991 同大大学院理学系研究科数学専攻修士課程了．同年（株）東芝入社．1993〜1996 情報処理振興事業協会に出向．1996〜1997 英国 City 大学客員研究員．1997〜1998 英国 Imperial College 客員研究員．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研究主務．エージェント技術，及びソフトウェア工学などの研究に従事．日本ソフトウェア科学会会員．

大須賀昭彦　（正員）

1981 上智大・理工・数学卒．同年（株）東芝入社．1985〜1989（財）新世代コンピュータ技術開発機構に出向．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．工博（早大）．2002 より電気通信大学大学院客員助教授並びに大阪大学大学院非常勤講師兼任．ソフトウェアのためのフォーマルメソッド，エージェント指向技術などの研究に従事．1986 年度情報処理学会論文賞受賞．IEEE CS，情報処理学会，日本ソフトウェア科学会各会員．

本位田真一

1976 早大・理工・電気卒．1978 同大大学院理工学研究科電気工学専攻修士課程了．（株）東芝を経て 2000 より文部科学省国立情報学研究所教授，現在に至る．2001 より東京大学大学院情報理工学系研究科教授を併任．2002〜2003 英国 UCL 並びに Imperial College 客員研究員（文部科学省在外研究員）．工博（早大）．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，情報処理学会など各会員．