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

田原康之[†] 大須賀昭彦[†] 本位田真一[††,†††]

ネットワーク上を移動しながら動作するモバイルエージェントの実用化においては，アプリケーション開発において多くの問題がある．たとえば，セキュリティが最も大きな問題の1つであると広く考えられている．本論文では，我々が開発し公開した，モバイルマルチエージェントアプリケーションの開発支援ツールIPEditorと，モバイルエージェントアプリケーションのための形式的仕様記述言語Mobile UNITYにより，セキュリティ問題に対し効果的に対処できることを示す．我々の手法では，IPEditorが，エージェントの挙動の記述を視覚的に支援することにより，アプリケーション設計の効率が向上する．また，IPEditorモデルを，エージェントの挙動の形式的仕様記述であるMobile UNITYプログラムに翻訳する．さらに，セキュリティへの要求をMobile UNITY論理により記述する．これにより，Mobile UNITYプログラム，すなわちIPEditorモデルで表されるエージェントの挙動が，Mobile UNITY論理記述，すなわち与えられた要求を満たすかどうかを検証することが可能となる．そして，電子カタログアプリケーションなどの例により，我々の提案の有効性を示す．

Mobile Agent Security with the IPEditor Development Tool and the Mobile UNITY Language

Yasuyuki Tahara,[†] Akihiko Ohsuga[†] and Shinichi Honiden[††,†††]

Many people consider that there are many problems in development of practical applications of mobile agents that move around the network and do their tasks. For example, security is one of the biggest problems. In this paper, we assert that this issue can be effectively managed by using IPEditor, the development support tool of mobile multi-agent applications that we have been released, and Mobile UNITY, a formal specification language of mobile agent applications. In our method, IPEditor helps developers to design applications with visual supports of agent behaviors. We translate an IPEditor model to a Mobile UNITY program that is the formal specification of the agent behaviors. In addition, we describe the requirements by the Mobile UNITY logic notation. Thus we can verify the requirements by proving that the mobile UNITY program, therefore the IPEditor model, satisfies the mobile UNITY logic notation. We present examples including an electronic catalog (e-catalog) application and illustrate the effectiveness of our proposal.

1. はじめに

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．我々は，現在モバイルエージェントを利用したマルチエージェントフレームワークBee-gent[25]の開発を進めている．我々は，Bee-gentによるモバイルエージェント技術の実用化を目的とし，Bee-gentに多くの実用的特徴を組み込んでいる．マルチエージェントアプリケーションのためのグラフィカルな開発支援ツールIPEditorもその1つである．我々は，B to B EC向け電子カタログシステムを含めた，Bee-gentの業務用アプリケーションの開発もいくつか行っている．

モバイルエージェント技術の実用化においては，アプリケーション開発上の問題が多数存在する．たとえば，セキュリティが最大の問題の1つであると考える人は多く[6]，この問題の解決のために，多くの研究者が様々な手法を提案している[2),10)~13),16),21),24)]．しか

† 株式会社東芝研究開発センターコンピュータ・ネットワークラボラトリー
Computer and Network Systems Laboratory, Corporate Research and Development Center, Toshiba Corporation
†† 国立情報学研究所
National Institute of Informatics
††† 東京大学
The University of Tokyo

し，これらの手法は，重要業務向けアプリケーションにおいて実際的に利用できるまでには至っていないと考えられる．

一般のソフトウェアシステムにおいて，高信頼なセキュリティを実現する技術の 1 つとして，形式仕様技術がある．たとえば文献 3) は認証プロトコルの仕様記述と検証のための論理体系を提案し，以来長きにわたって研究されている．したがって，形式仕様技術をモバイルエージェントのセキュリティ問題に適用することは自然である．しかし，モバイルエージェント向けの形式的仕様記述言語はわずかしかないために，そのような適用例はまだほとんどない．そのような言語の 1 つに Mobile UNITY[14] がある．この言語は，モバイルエージェントアプリケーションの仕様記述に適した特徴を多く持っている．

本論文では，IPEditor と Mobile UNITY を利用して，モバイルエージェントのセキュリティ問題を効果的に解決できる手法[19] を提案する．本手法では，IPEditor がエージェントの挙動に対する状態遷移モデルに基づくことを利用し，IPEditor モデルを，エージェントの挙動の形式仕様である Mobile UNITY プログラムに，ほとんど自動的に変換することができる．さらに，セキュリティ要求は Mobile UNITY 論理記法で記述する．このため，Mobile UNITY プログラムが，したがって IPEditor モデルが，Mobile UNITY 論理記法の記述を満たすことを証明することにより，セキュリティ要求を検証することが可能となる．なお，本論文で利用する Mobile UNITY 言語仕様においては，開放型分散環境において，セキュリティ要求の仕様記述および検証を容易にするために，パラメータ化の拡張を実施している．

我々は，本手法を電子カタログアプリケーションに適用した．本論文では，本適用例を通じて我々の提案の有効性を示す．なお，一般に形式仕様技術を適用するセキュリティ問題としては，前述の認証プロトコルなど，個別のプロトコルレベルの問題が多い．しかし本論文で取り上げる例題では，現実的な問題での有効性を検討するため，認証機構を含んだシステム動作が，意図したとおりに機能するかどうか，というアプリケーションレベルの問題に適用した．もちろん，個別のプロトコルレベルの問題の検討も必要であり，今後の課題としたい．

一方本手法では，IPEditor，および Mobile UNITY という，モバイルエージェントアプリケーション向けとしては汎用的な開発ツールと仕様記述言語を利用しているので，セキュリティ問題に限らず，幅広い開発上の問題に適用可能である．そこで本論文では，セキュリティ問題以外への適用例として，以下のものも紹介する．

- 分散トランザクション処理における原子性検証
- 完了期限が定められたサービス連携において，期限が守られるか否かの検証

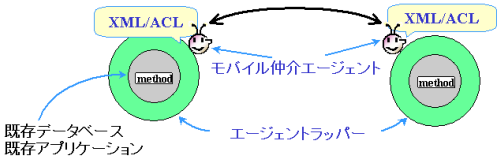

図 1 Bee-gent の特徴
Fig. 1 Features of Bee-gent.

本論文の構成は次のとおりである．2 章では，我々の手法で利用する要素技術の概要を説明する．3 章では，我々の手法におけるモバイルエージェントセキュリティの検証法を詳述する．4 章では，電子カタログアプリケーション例題への適用を通じて，我々の手法の有効性を示す．5 章では関連研究との比較を行い，6 章で結論と今後の課題を述べる．

2. 要素技術の概要

本章では，我々の手法で利用している要素技術の概要について説明する．具体的には以下の 3 技術である．

- マルチエージェントフレームワーク Bee-gent
- マルチエージェントアプリケーションのためのグラフィカルな開発支援ツール IPEditor
- モバイルエージェント向け仕様記述言語 Mobile UNITY

なお，我々の手法は一般のモバイルエージェントプラットホームに適用可能であり，1 つ目の Bee-gent はあくまでもモバイルエージェントプラットホームの一例であるので，必ずしも必要ではなく，他のプラットホームを用いてもかまわないことを注意しておく．

2.1 マルチエージェントフレームワーク Bee-gent

Bee-gent は，ネットワークで接続された分散計算機環境上で動作している既存システムの連携により，開放型分散アプリケーションを構築するためのマルチエージェントフレームワークである．ウェブサイト `http://www2.toshiba.co.jp/beegent/` では，Bee-gent の試用版パッケージを無料で提供している．

Bee-gent の特徴は次のとおりである（**図 1** 参照）．

- モバイル仲介エージェントが，様々なアプリケーション間の連携動作を一元的に管理し，自分自身のプログラム，データ，および状態を運びながら

ネットワーク上を移動する．

- エージェントラッパーが，XML/ACL（XML 形式のエージェント間通信言語〈Agent Communication Language〉）により，既存ソフトウェアに共通インタフェースを提供する．ACL としては，FIPA（Foundation for Intelligent Physical Agents）[1] により策定された国際標準のものを採用している．
- モバイル仲介エージェントとエージェントラッパーが，XML/ACL 形式のメッセージの交換により協調動作を行う．これにより，モバイル仲介エージェントが管理している，アプリケーション間連携動作が実現される．

2.2 開発支援ツール IPEditor

IPEditor は，GUI 上でエージェントの挙動を状態遷移図で記述できる開発支援ツールである．また本ツールは，メッセージシーケンス図（相互作用図と呼ぶ）によるエージェント間相互作用の記述支援，テンプレート形式による ACL メッセージ定義記述支援，およびエージェントのビヘイビアパターン[17),18)] もサポートしている．IPEditor の試用版は前述の Bee-gent パッケージに含まれており，無料で入手可能である．IPEditor のスクリーンショットの例は次章で提示する．

2.3 モバイル向け仕様記述言語 Mobile UNITY

Chandy らは，文献 5) 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[14),15)] 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．詳しくは，モビリティに関する以下のような側面を取り扱うように拡張している．

- エージェントなどのコンポーネントが稼働している場所を変数で表して，プログラムにより可変とすることにより，エージェントの場所の移動という側面を表す．
- 移動によりローカルな通信が可能，あるいは不可能になることを，新たな同期プリミティブにより表現することにより，移動による通信状況の変化という側面を表す．

なお本論文では，開放型分散環境において，セキュリティ要求の仕様記述および検証を容易にするために，Mobile UNITY 言語仕様に対し，パラメータ化の拡張を実施している．

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 2 つに分けられる．以下に，それぞれの詳細について説明する．

2.3.1 Mobile UNITY プログラム

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，次のものから構成される．

- プログラム名：`System` の直後に記述される．
  例：`System eCatalog`
- program 部：`program ... end` で囲まれた，個々のシステムの構成要素（サーバプログラムやエージェントなど）の記述である．詳しくは，次のものから構成される．
  - **名前と位置宣言**　キーワード program の後に，構成要素の名前を記し，その後キーワード at が続き，最後に構成要素が稼働している場所（変数も可）を記す．場所が変数の場合は，当該構成要素が移動可能であることになる．
    例：`program agent1 at lambda`
    は，構成要素 `agent1` が変数 `lambda` の値で表される場所で稼働することを表す．
  - **declare 部**　変数とその型を宣言する部分．
    例：`declare n : int`
    は，`int` 型の変数 `n` の宣言を表す．
  - **always 部**　関数定義を行う部分．
    例：`always f(x) = x * x`
    は，関数 `f` の定義である（`x` は引数）．
  - **initially 部**　変数の初期値を定義する部分．
    例：`n = 1`
  - **assign 部**　プログラムの動作を，文の非決定選択として記述する．意味的には，いずれかの文を非決定的に実行する，という手順を，実行可能な文が存在する限り無限回行う．ただし，どの文も実行可能な限り無限回実行するものとする．文の詳細については後述．
- Components 部：システムの構成を，program 部のインスタンスの並行合成で表したものである．
  例：`agent1 [] agent2 [] application1`
  は，システムが 3 つのコンポーネント `agent1`，`agent2`，`application1` から構成されることを示す．
- Interaction 部：システムの構成要素間の相互作用を，program 部の assign 部と同様に，非決定的に実行される複数の文で表したものである．

Mobile UNITY の文には，次のものがある．

- 変数への代入（複数の変数への同時代入も含む）

例：`m, n, s = 5, m + 1, "abc"`
は，変数 m への 5 の代入，n への m + 1 の代入，および s への"abc"の代入を同時に行うことを表す．たとえば m の値が 1 のときにこの文を実行すると，n の値は（6 ではなく）2 になる．

- if 条件文
  例：`n = 1 if s = "abc"`
- 逐次実行文（";" で区切られる）
  例：`<m = 1 ; m = m + 1>`
- ラベル付き文（「ラベル」"::" 文）
  例：`l :: <m = 1 ; m = m + 1>`
- 禁止文（"inhibit" ラベル "when" 条件）：条件が成り立つときラベル付き文の実行を禁止する．
  例：`inhibit l when s = "abc"`
- 反応文（文 "reacts-to" 条件）：条件監視付きの条件文
  例：`n = 1 reacts-to s = "abc"`
  前述の if 条件文の例との違いは次のとおりである．この文を含んだプログラムを実行する際には，どの文を実行した後でも条件 `s = "abc"`が成立するかどうかを調べ，成立した場合にはただちにこの文を実行する．

2.3.2 Mobile UNITY 論理

Mobile UNITY の論理における命題は，次のように構成される．

- 最小の構成要素は表明 $\{p\}s\{q\}$．
  ただし，p，q は通常の一階述語論理の命題，s はプログラムの文．p が成立している場合に，文 s を実行した後，q が成立する，ということを表す．
- プログラム F に対し，F の文に関し限量化した表明 $\forall s \in F\,\{p\}s\{q\}$，$\exists s \in F\,\{p\}s\{q\}$ を構成．なお元の UNITY には，ホーア論理[28] とは異なり逐次実行がないので，プログラムの性質はこの 2 種類の命題のみから構成することができる．Mobile UNITY では，文 s が逐次実行文の場合の表明の推論規則を与える必要があるが，それを除けばホーア論理と同様である．
- 限量化した表明と命題論理の演算子 $\wedge$，$\vee$，$\neg$ から命題を構成．含意演算子についても，通常どおり $p \Rightarrow q \stackrel{\text{def}}{\Longleftrightarrow} \neg p \vee q$ と定義．
- 通常の一階述語論理の命題 p と q から命題 $p \longmapsto q$（p leads-to q とも書く）を構成．その直感的意味は，p が成り立てば，そのうち q が成り立つようになる，ということである．

なお，上のように構成される命題のうち，以下のような特別な省略形も用意する．

- p unless q：プログラム F に対し，$\forall s \in F\,\{p \wedge \neg q\}s\{p \vee q\}$ の略．直感的には，p が成り立っていれば，q が成り立たない間は p は成り立っている，という意味．
- p is stable：p unless false の略．一度 p が成立すれば，その後 p は成立し続ける，という意味．
- p is invariant：初期状態を表す命題を i としたときの，$(i \Rightarrow p) \wedge p$ is stable の略．p が始めからずっと成立し続ける，という意味．
- p ensures q：p unless $q \wedge \exists s \in F\,\{p \wedge \neg q\}s\{q\}$ の略．p が一度成り立てば，q が成り立たない間は p は成り立ち続けるが，そのうち q が成り立つ，という意味．

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 5) 52 ページ）を適用することにより行う．

2.3.3 Mobile UNITY に対するパラメータ拡張の提案

なお本論文では，開放型分散環境において，セキュリティ要求の仕様記述および検証を容易にするために，Mobile UNITY 言語仕様に対し，パラメータ化の拡張を実施している．本来モバイルエージェントは，開放型環境，特に構成要素が動的に変化するような環境において特長が発揮されると考えられている．しかし，元々の Mobile UNITY は，構成要素の動的な変化を記述するには不十分である．したがって本論文では，動的に変化する部分をパラメータとして表現できるような仕様拡張を実施した．

パラメータ化で拡張した Mobile UNITY プログラムの例を以下にあげる．

```
System aSystem[Param1 : type1,
               Param2 : type2]
  ...
  program component1 at location1
    declare
      ...
      [] var1 : type1
      ...
    initially ...
    assign
      ...
      [] var1 = Param1
      ...
  ...
end
```

本プログラムは，2 つのパラメータ Param1 と Param2 を持ち，それぞれ type1 および type2 の型を持つ．これらのパラメータは，適切な型を持つ具体値を代入す

ることができ，すべてのパラメータに具体値を代入することにより，具体的なプログラムが得られる．

論理記述に関しても，命題 p はやはりパラメータを持つことができる．ただしこの場合，パラメータは通常の変数として扱われ，したがって検証の対象としては，限量子（$\forall$ あるいは $\exists$）により束縛する必要がある．たとえば，パラメータ x を持つプログラムが，命題 $P(x)$ で表される性質を持つとき，かつそのときに限り，命題 $Q(x)$ で表される性質を持つ，という課題は，命題 $\forall x\,(P(x) \Leftrightarrow Q(x))$ の検証で解決できる．

3. 検証手法

本章では，モバイルエージェントのセキュリティ検証手法の詳細を説明する．本手法は以下のステップから構成される．

(1) IPEditor により設計を行うと，その結果が IPEditor 検証機能により自動的に Mobile UNITY プログラムに変換される．正確にいうと，IPEditor 仕様記述の形式的表現が変換対象となる．

(2) Mobile UNITY プログラムを，必要に応じて人手で修正する．

(3) Mobile UNITY 論理記法により，セキュリティ要求仕様を記述する．

(4) Mobile UNITY プログラムがセキュリティ要求仕様を満たすかどうかを検証する．

3.1 IPEditor 仕様の形式的表現

IPEditor 仕様を Mobile UNITY プログラムに変換するためには，何らかの仕様の形式的表現が必要である．以下に，そのような表現の枠組みについて説明する．

IPEditor 仕様は 3 つの構成要素，すなわち，各エージェントの状態遷移図，各状態の相互作用図，およびメッセージ定義から構成される．相互作用図は各状態ごとに 1 つずつ用意される．またメッセージ定義は，相互作用図の各メッセージ矢印ごとに与えられる．

以降で，各構成要素を形式的に表現し，その後どのように結合するかを検討する．各構成要素の形式的表現は以下のとおりである．

- エージェント A_i の状態遷移図は，有向グラフ $(N_i, E_i, \iota_i, \epsilon_i)$（ただし，$E_i \subseteq N_i \times N_i; \iota_i, \epsilon_i \in N_i$ はそれぞれ始状態と終状態）で表現する．各状態 $s \in N_i$ には 1 つの相互作用図が割り当てられる．その形式的表現は，メッセージ定義の列とメッセージ定義の集合との対 $(\langle M_j | j = 1, 2, \ldots, n\rangle, Ms)$ である．Ms の要素は，この状態において最後に発生しうるメッセージ交換を表し，複数存在する可能性がある．次状態は，どのメッセージ交換が発生するかによって変化する．
- 上述した列に含まれるメッセージ定義 M_j は，組 (s, r, c) で表される．ここで s，r，c は以下のものを指す．
  - s はメッセージを送信するエージェント
  - r はメッセージを受信するエージェント
    ただし，s か r のいずれか一方は，状態遷移図で指定されている A_i
  - c はメッセージの内容を表す文字列
- Ms の要素であるメッセージ定義 M は，前述のメッセージ定義 M_j とは異なり，組 (s, r, c, ns) によって形式的に表現される．ここで，前述のメッセージ定義の形式的表現にはなかった，次状態 ns が含まれていることに注意する．これは，Ms の要素である M は，状態遷移を引き起こすイベントに相当するからである．したがって，M は当該状態の相互作用図においては終端のメッセージとなるので，「終端メッセージ定義」と呼ぶ．
  また，各 M_1，$M_2 \in Ms$，$M_i = (s_i, r_i, c_i, ns_i)$ に対し，$s_1 = s_2$，$r_1 = r_2$．すなわち，s と r は，すべての $M \in Ms$ について，それぞれ共通である．
  さらに，状態 s から s' への遷移を表すエッジ (s, s') が存在する場合は，遷移を引き起こすイベントとして，s' を次状態とする Ms の要素の存在が必要．また逆に，第 3 の状態 s'' に対し，次状態が s'' となるような Ms の要素が存在すれば，エッジ (s, s'') が存在することが必要である．

3.2 1 エージェントに対する全メッセージグラフの生成

次に IPEditor は，状態遷移図，相互作用図，および各メッセージ定義から，1 エージェントに対する全メッセージグラフを生成する．全メッセージグラフは有向グラフ構造で表現される．

詳しくは，エージェント A_i の状態遷移図 $(N_i, E_i, \iota_i, \epsilon_i)$ と相互作用図の全体 $\{(\langle M_{\sigma j} | j = 1, \ldots, n_\sigma\rangle, Ms_\sigma) | \sigma \in N_i\}$ から，有向グラフ $(\bigcup_\sigma \{M_{\sigma j} | j = 1, \ldots, n_\sigma\} \cup \bigcup_\sigma Ms_\sigma, B)$ を生成する．ここで，エッジの集合 B は，次の条件を満たすものである．

$$
\begin{aligned}
&(M^1, M^2) \in B \\
&\Leftrightarrow \exists \sigma, j\ (M^1 = M_{\sigma j}, M^2 = M_{\sigma(j+1)}) \\
&\quad \text{or } \exists \sigma\ (M^1 = M_{\sigma n_\sigma}, M^2 \in Ms_\sigma) \\
&\quad \text{or } \exists (\sigma^1, \sigma^2) \in E_i (M^1 = (A_1, r, c, \sigma^2) \in \\
&\qquad Ms_{\sigma^1}, M^2 = M_{\sigma^2 1})
\end{aligned}
$$

この条件は，各エッジは相互作用図と同じ順番で隣り合っているメッセージを結ぶか，またはある状態の最後のメッセージからその次の状態の最初のメッセージへと延びるかのいずれかである，ということを意味する．

3.3 全メッセージグラフの Mobile UNITY プログラムへの変換

全メッセージグラフを Mobile UNITY プログラムに変換する手順は以下のとおりである．

(1) 各エージェントごとに `Program` 部を用意する．

(2) 各エージェントごとに，以下の変数を宣言する．
- メッセージ交換を行う他のエージェントごとの，通信チャネル変数．
- 各通信チャネルごとの，通信状態変数．`"notUsed"`，`"sent"`および`"received"`の 3 通りの状態をとりうる．
- 各メッセージ交換ごとに値が変化する状態変数．

(3) 各エージェントに対する `Program` 部の記述において，各メッセージ送信動作は，該当する通信チャネル変数と通信状態変数へのメッセージ記述の代入と，それにともなう状態遷移を表すための，状態変数への代入とに変換する．
またメッセージ受信動作は，該当する通信チャネル変数と通信状態変数に関する条件判断をともなった，状態変数への代入に変換する．
なお Bee-gent では，モバイル仲介エージェントは，エージェントラッパーに移動要求メッセージを送信することにより移動を行うことができる．その後，移動先のラッパーから移動成功通知メッセージを受信すると，移動が完了となる．したがって，受信したメッセージが移動成功通知であった場合には，場所変数への移動先の場所の代入も追加する☆．

(4) `Interaction` 部は，メッセージ交換を行う各エージェントペアについて，共有する通信チャネル変数に関する等式を合成して生成する．

3.4 プログラム記述の要求仕様に対する検証

以上で得られたプログラム記述が，与えられたセキュリティ要求などの要求仕様を満たすことの検証は，形式的証明を与えることによって行う．Mobile UNITY は，形式的証明のために，公理と推論規則を持っている．公理は次のようなものである．

- 一階述語論理の公理．
- 公理となる表明．たとえば，$\{p\}x := E\{p \wedge x = E\}$．

推論規則は次の 2 種類に分れる．

- 一階述語論理の推論規則．
- プログラム実行に対する推論規則．これらの規則は，一般のプログラム記述に対し，実行後に成立する命題を，すでに正しいことが分かっている表明から導出する．詳しい説明は，文献 5) の 46 ページにある．

Mobile UNITY では，時相論理のオペレータは，表明を持つ一階述語論理により定義されるので，そのための特別な公理や推論規則は不要である．

4. 適 用 例

本章では，いくつかの適用例を通じ，我々の手法の有効性を示す．本章で取り上げる例は以下のとおりである．

- 電子カタログアプリケーションにおけるセキュリティ検証
  本例題は，非機能的特性としてのセキュリティ要求の検証例である．
- 分散トランザクション処理における原子性検証
  本例題は，機能的特性への要求の検証例である．
- 完了期限が定められたサービス連携において，期限が守られるか否かの検証
  本例題は，(セキュリティ以外の) 非機能的特性の検証例である．

4.1 電子カタログシステムへの適用例

近年，いわゆる B to B EC，すなわち企業間電子商取引がさかんになるにつれて，製品や部品の取引情報の電子化，およびネットワークを通じた流通へのニーズが急速に高まっている．その中で，電子化された製品や部品のカタログ情報，いわゆる電子カタログについては，そのコンピュータ可読なデータ形式の国際標準規格が ISO13584，あるいは PLIB (Parts Library) として，ISO により策定されている．このように，系列取引などの従来の枠組みを超えた，開放的な取引環境への期待が高まっている．

しかし，PLIB 標準に基づいた電子カタログ情報の，ネットワークを通じた流通を実現するにあたっては，現実には以下のような問題が存在する．

- 電子カタログ情報の利用においては，多様な検索・収集・登録要求に基づいて，ネットワーク上の多数の電子カタログ DB を操作する必要がある．したがって，すべて人手で行うのは多大なコストが

☆ Bee-gent 以外のモバイルエージェントプラットホームに適用する場合は，プラットホーム固有の移動タイミングに対し，この手続きを行う．

かかり，自動化する場合も，多種多様な要求を扱うシステムを構築するコストがやはり大きなものとなる．

- 開放的な取引環境実現のためには，ネットワーク上の各電子カタログ DB について，問合せ言語やプロトコルなど，様々な異なった DB インタフェースを扱う必要がある．しかし，このような多種多様なシステムを統合的に利用するのは困難である．また，開放性のために頻繁なシステム構成の変更が発生するが，そのような変更に対応するのはいっそう困難のものとなる．

そこで我々は，このような問題を解決するため，電子カタログシステムに Bee-gent[25] を適用した．本適用例では，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，および登録プロセスを実行するものである．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 で通信可能である．これにより各アプリケーションごとのインタフェースの違いを吸収する．さらに，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，システム利用を低コストで自動化することができる．また，システム構成変更の際にも，アプリケーションとエージェントラッパーを変更せずに，仲介エージェントの変更だけで対応できる場合が多く，コストを抑えることが可能である．これにより，ネットワーク上に分散した，電子カタログシステムの利用・保守・管理が容易になることが期待できる．

図 2 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 が稼働しているサイトを巡回し，検索/収集作業を実行して，最後に元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`），および 2 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，

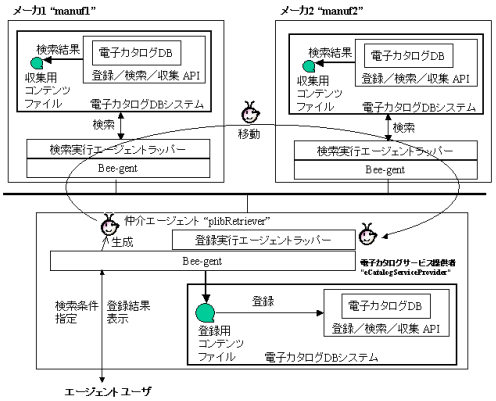

図 2 電子カタログの検索・収集・登録システム
Fig. 2 E-catalog retrieval, gathering and registration system.

`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` 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，正当な権限を持たないホストが，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．

4.1.1 例題の IPEditor 記述

本例題におけるモバイル仲介エージェントの IPEd-

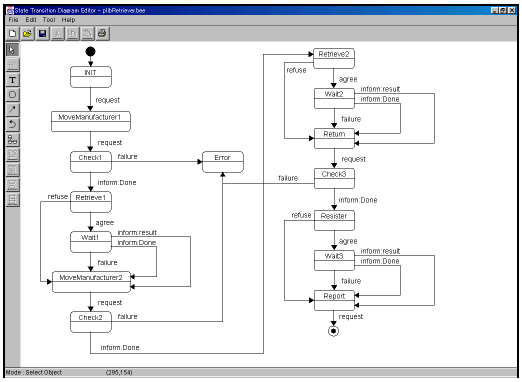

図 3 本例題の IPEditor 記述の一部：状態遷移図
Fig. 3 A part of the IPEditor specifications of this example: STD.

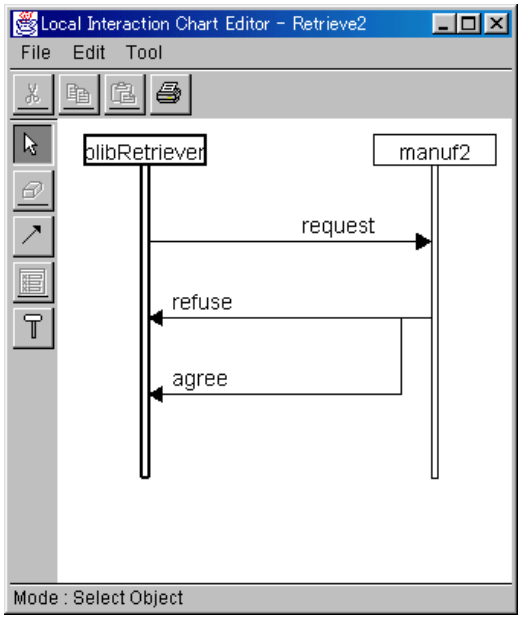

図 4 本例題の IPEditor 記述の一部：相互作用図
Fig. 4 A part of the IPEditor specifications of this example: interaction chart.

itor 記述の一部を**図 3**，**4**，**5** に示す．これらの図において，相互作用図は状態 `Retrieve2` のものであり，メッセージ定義はその相互作用図中の `request` メッセージのものである．その中で，`action` パラメータに設定されている `retrieveECat` 動作は，引数にある検索キーによる検索動作を表す．

本記述は，Request-Itinerary パターン[17] に基づく．したがって，状態遷移図と相互作用図は，IPEditor によって自動的に生成されたものである．その後人手により，相互作用図中の記号を修正し，メッセージ定義の項目を記述している．

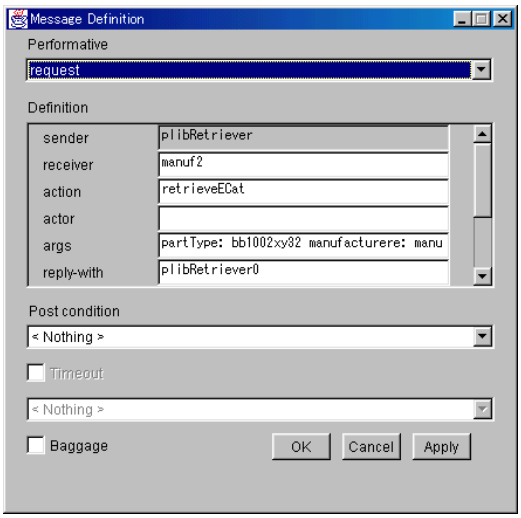

図 5 本例題の IPEditor 記述の一部：メッセージ定義
Fig. 5 A part of the IPEditor specifications of this example: message definition.

Mobile UNITY プログラムは本記述から生成され，その後設計者が修正する．まず，自動変換の手続きの例をあげる．

- エージェント `manuf2`，`plibRetriever` に対し，`program` 部宣言 `program manuf2 at "manuf2"`（エージェントラッパーは移動しないので，場所は名前と同じ文字列に固定），および `program plibRetriever at lambda`（モバイル仲介エージェントの場所は変数 `lambda`）が生成され，その下に `program` 部の内容がそれぞれ入る．
- これら 2 エージェントは，お互いに相互作用図においてメッセージ交換を行うので，それぞれ通信チャネル変数と通信状態変数が下記のように用意される．

```
program manuf2 at "manuf2"
  declare
      plibRetrieverChannel : string
    [] plibRetrieverChannelStatus
         : string

program plibRetriever at lambda
  declare
      manuf2Channel : string
    [] manuf2ChannelStatus : string
```

他のエージェントのペアについても同様．

- 各エージェントに状態変数が用意される：`declare` 部に `status : string`
- エージェント `manuf2` の `retrieved` 状態において，`plibRetriever` にメッセージを送信し，状態 `resultSent` に遷移する，ということを表す，全メッセージグラフのエッジに対し，次の代入文が

生成され，assign 部に入る．

```
[] state, plibRetrieverChannel,
   plibRetrieverChannelStatus
   = "resultsSent", ...,
       /* メッセージ文字列 */
   "sent" react-to state = "retrieved"
```

- エージェント plibRetriever については，移動要求メッセージを送信すると，移動を行う．この場合，上記と同様の代入文生成を行うが，下記のとおり変数 lambda への代入もともなう．

```
[] status, manuf2Channel,
   manuf2ChannelStatus, lambda
   = "Check", ..., "notUsed",
     "eCatalogServiceProvider"
   react-to status = "Return"
     and manuf2ChannelStatus = "sent"
```

- Interaction 部には，たとえば上記 2 エージェント間の通信を表すために，次の等式が入る．

```
[] manuf2.plibRetrieverChannel,
   manuf2.plibRetrieverChannelStatus
   = plibRetriever.manuf2Channel,
     plibRetriever.manuf2ChannelStatus
   when plibRetriever.lambda = "manuf2"
     /* Bee-gent では，ホスト内での
        メッセージ交換が中心のため，
        本条件文を挿入 */
```

次に，以上のように生成された記述に対し，開発者が手修正を施す．以下は手修正の例である．

- プログラム名を記述する．

  例：System eCatalog[Manuf2Signature : string]

- DB 用の変数を用意する．

  例：

```
program manuf2 at "manuf2"
  declare
     ...
   [] DB : stringList
   [] query : string
     ...
  initially
     ...
   [] DB = ...
```

- 電子署名文字列のための変数 signature とその値を設定する．

  例：

```
program manuf2 at "manuf2"
  declare
     ...
   [] signature : string
     ...
   initially
     ...
   [] signature = "Manuf2Signature"
```

- DB 検索・登録手続きを記述する．その際，電子署名手続きを導入．

  例：

```
assign
   ...
 [] state, plibRetrieverChannel,
    plibRetrieverChannelStatus
    = "retrieved",
        /* 検索結果通知メッセージ */
      "(inform ..."
      + sign(toString(retrieve(DB,
        query)), signature) + "...)",
      "sent"
    react-to state = "wait"
              and query =\= ""
      /* 問合せが存在する場合 */
```

以上の自動変換手続きとて入力により得られた Mobile UNITY プログラム（の部分）を付録 A.1.1 に示す．

4.1.2 セキュリティ要求仕様記述

まず，署名関数 sign および verify に関する下記の論理式を p とする．

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

これらの記述は，正当な署名者が署名したときのみ，認証手続きが成功する，ということを意味する．

セキュリティ要求仕様記述は次の 2 つの部分から構成される．1 つ目は，すべてのホストが正当な署名を有している場合には，要求された結果が得られる，ということを主張するものである．2 つ目は，あるホストが正当な署名を有していない場合，プログラム中の特定の手続きが実行されない，ということを主張するものである．

本例題で，1 つ目の仕様記述中での要求された結果というのは，ネットワーク上の DB から収集したデータの登録が正しく完了する場合である．このような要求は，Mobile UNITY 論理で

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

すなわち，manuf2 の署名を表すパラメータ Manuf2Signature が，正しい値"manuf2"を持ち，かつ p が成り立てば，そのうち（leads-to）登録が完了する，と記述できる．

セキュリティ要求仕様の 2 つ目の部分に含まれる，不正なデータの登録を防ぎたいという要求の仕様記述は，Mobile UNITY 論理により

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

```
=\= "registered"))
```

すなわち，p が成り立てばつねに (->) 登録完了状態にはならない，と記述される．

4.1.3 セキュリティ要求仕様の検証

我々の手法では，最後のステップにおいて，Mobile UNITY プログラムが Mobile UNITY 論理記述を満たすことを証明することにより，セキュリティ要求仕様を検証する．すなわち，前述の 2 つの命題が成立することを証明すればよい．

たとえば，前半部分（登録が正しく完了する場合）の証明の概要は次のようになる．(Mobile) UNITY プログラムの実行は，原則として各代入文が非決定的に選択されて実行される，という手順の無限ループなので，このような「いつかは成り立つ」というような性質の証明のためには，

- 安定な表明はどの文の実行後でも成り立ち続ける，
- どれか 1 つの文の実行後に，いつかは成り立ってほしい性質が成り立つ，

の 2 つを示すことが必要である．したがって，本例題では以下のようになる．

(1) 署名の確認を行う文を実行する前まで，p が成り立ち続けることを証明する．これは，実行される各文 s に対し，公理を用いて {p} s {p} を示すことにより可能である．

(2) p が成り立つとき，署名の確認に合格することを証明する．ここでは，示す命題は上と同様に {p} s {p} だが，署名確認を行う文が実行可能であることを示すことが中心となる．

(3) 登録を実行する文 s に対し，{p} s { eCatalogServiceProvider.state = "registered" } を示す．

これらを示すことにより，leads-to の定義により証明が完成する．

なお，パラメータ化がない従来の Mobile UNITY では，前述の 2 つの命題それぞれに対し，異なる検証を行う必要があった[27]．これは，不正なホストの存在の場合の検証において，開放型分散環境では実際には一意に特定できない不正なホストの仕様をも，具体的に記述する必要があったことによる．一方本論文では，不正なホストの仕様も，パラメータにより一般的に記述することが可能となり，一意に特定する必要がない．したがって，パラメータ化による拡張を行ったことにより，開放型分散環境への対応も可能となっている．

4.2 トランザクション処理の例

モバイルエージェントは，ネットワーク上に分散する複数のサービスを，1 つのトランザクションとして利用するための技術としても有力視されている．たとえば空路利用の旅行において，複数の航空会社の便を乗り継ぎたい場合，それら複数の便を同時に予約する必要がある．すなわち，いずれか一方の予約だけではサービスとして成立しない，いわゆる原子性を有するトランザクションとして処理すべき，との要求が課せられる．詳しくは，次のいずれかが成り立つべき条件となる．

- 両方の航空会社の便の予約が成立．
- (いずれか一方が満席であるなどの場合) どちらの航空会社の便も予約しない．

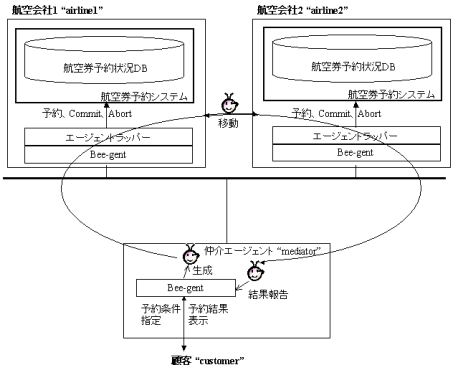

図 6 航空券予約サービスの例
Fig. 6 Example of airline ticket reservation service.

ここで各航空会社の航空券予約サービスが，異なるサイトにおいてネットワーク上で提供されている場合，**図 6** のように Bee-gent を用いてトランザクション処理を実現することができる．本例において，トランザクション処理を実現するために，各予約サービスにおいて，通常の DB 操作のほかに，最終的にトランザクションを確定するための Commit 操作と，トランザクションキャンセルのための Abort 操作を実装している．

以下，Bee-gent を用いて実現された本トランザクション処理が，上記の原子性を満たすことを検証することを例題とする．

4.2.1 例題の記述

本例題におけるモバイル仲介エージェントの IPEditor 状態遷移図の一部を**図 7** に示す．本状態遷移図において，“Abort . . .” や “Commit . . .” といった状態では，それぞれ Abort または Commit 操作の request をラッパーに送信している．

付録 A.1.2 に，本例題の Mobile UNITY プログラム記述の一部を示す．本プログラムでは，主に Abort および Commit 操作の実装を，設計者が人手で作成

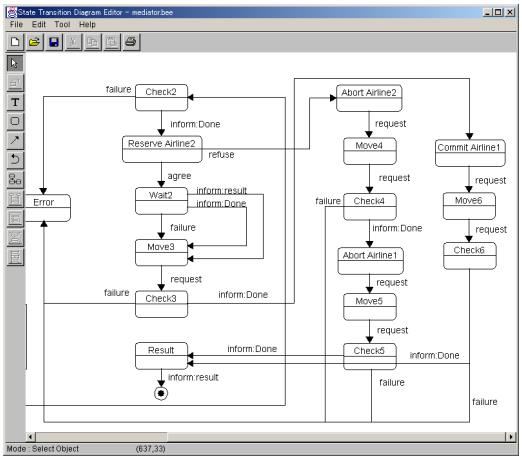

図 7 航空券予約サービス例題の状態遷移図の一部
Fig. 7 A part of the STD of the airline ticket reservation service example.

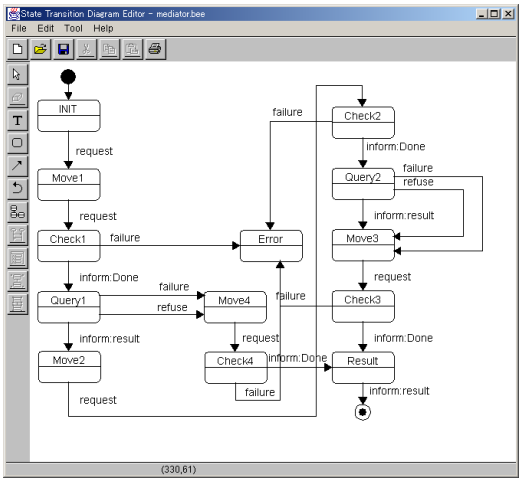

図 8 期限付きサービス連携例題の状態遷移図の一部
Fig. 8 A part of the STD of the "service integration with a deadline" example.

する必要がある．

4.2.2 トランザクション原子性の検証

本例題におけるトランザクションの原子性は，前記 2 条件から構成される．その形式的記述は以下のとおりである．

```
true leads-to (mediator.state = "END" and
  (status(airline1.DB, "researved")
    and status(airline2.DB, "researved"))
  or (status(airline1.DB, "no-contract")
       and status(airline2.DB,
                  "no-contract")))
```

ただし，述語 `status` は別に定義されているものとする．

この要求仕様の検証は次のようになる．すなわち，仲介エージェントの状態が `END` になるまでのどのような実行経路でも，最終的に `or` で結ばれた 2 命題のいずれかが成り立つ状態に到達することを証明することになる．

4.3 期限付きサービス連携の例

完了のための期限が定められたサービスを，ネットワーク上に分散した複数の部分的なサービスの連携により実現する場合，各部分的サービスの実行スケジュールを調整する必要がある．さらに，サービスの実行中でも，状況に応じて，部分的サービスの実行期限を延ばしたり縮めたりして，スケジュールを柔軟に変化させることも必要となる．このようなサービスを，モバイルエージェントを用いて実現する場合，エージェントがスケジュールを管理しながら部分的サービスの連携処理を行うようにすれば，比較的容易に実現することも可能である．いずれにせよ，実際に定められた期限内にサービスが完了するかどうかを，実行前に検証したい，という要求は強い．

具体的には，次のような例を考える．

- 2 つの部分的サービスを引き続き利用することにより 1 つのサービスを実現する．
- 2 つ目の部分的サービスが完了するのに，ある一定時間以上かかるため，1 つ目の部分的サービスはその事実を考慮してスケジュールを考える必要がある．
- 各部分的サービスは，サービスプロバイダが拒否したり，タイムアウトするなど，スケジュール通りに完了しないことが分かれば，その時点でサービス完了失敗とする．

4.3.1 例題の記述

本例題におけるモバイル仲介エージェントの IPEditor モデルの一部を**図 8** に示す．

本例題の Mobile UNITY プログラムを付録 A.1.3 に示す．本例題では，時間の要素を表すために，タイマーエージェント `timer` を用意している．

4.3.2 期限制約の記述と検証

サービスが，定められた期限 `mediator.deadline` までに完了する，という要求仕様を，次のように記述する．

```
true leads-to
 (mediator.state = "END"
  and timer.time <= mediator.deadline)
```

この要求仕様の検証は次のようになる．すなわち，仲介エージェントの状態が `END` になるまでのどのような実行経路でも，最後の時刻 `timer.time` が期限以内に収まっていることの確認を行うこととなる．

4.4 例題についての考察

以上の適用例を通じ，以下において本手法の有効性を考察する．

- Mobile UNITY では，状態遷移やメッセージ交換シーケンスといった，システム動作の基本的な構成要素が，数多くの代入文に埋め込まれるため，直接記述するのが困難であった．
  一方本論文の手法では，IPEditor を利用するため，状態遷移は状態遷移図により，またメッセージ交換シーケンスは相互作用図により，視覚的に表現されるので，開発効率が向上している．
  ただし，セキュリティなどの要求仕様の記述，および検証作業には，IPEditor の利用は無関係である．したがって，これらの作業を支援する機能を IPEditor に組み込んで実現することにより，さらに開発効率を向上させられることが期待できる．
  また，手修正の際には，自動生成された Mobile UNITY プログラム記述を確認する必要があるので，IPEditor による視覚的表現の効果を利用できない．したがって，手修正作業においても，何らかの視覚的効果による支援を実現することにより，さらに開発効率を向上させられることが期待できる．たとえば IPEditor のコンポーネント（状態アイコンや状態遷移矢印など）と，自動生成された Mobile UNITY プログラム記述の部分との関係を，やはり視覚的に提示する，といったような機能が考えられる．
- 本章では，様々な異なる種類の問題を取り上げたが，IPEditor および Mobile UNITY という汎用的な技術を利用したために，これらの問題が統一的な枠組みで解決できることが示された．
  ただし，いずれの種類の問題に対しても，それぞれさらに効果的な解決手段は存在すると考えられる．たとえば，トランザクション例題の場合，分散 DB 特有の手法がすでに多く提案され，実用化もされている．また期限付きサービス連携の例題の場合，リアルタイムアプリケーション向けの仕様記述言語などを用いた方が良い可能性がある．したがって，こういった特定問題向けの手法を適切に取り込めるような，柔軟性の高い手法を検討することも，今後必要となる．
- 本章では，非機能的特性への要求の検証例を 2 つ取り上げた．しかし，いずれの例でも，要求仕様記述においては，内容をある程度具体化して記述せざるをえなかった．しかし，非機能的特性への要求については，このような具体化が適切かどうかの疑問がある．
  たとえば，電子カタログ例題におけるセキュリティ要求については，悪意のあるサーバによるあらゆる攻撃に対し，エージェントの発信元ホストが最終的にそれを検出して対処する，という要求について検証したい．しかし，本例題では，正当な署名を持たないホストを経由したエージェントのデータ登録を防止する，という要求に具体化している．そのため，正当な署名を不正に入手して使用したホストを検出できない．なおこのような状況に対して検証を行うためには，署名の管理を行うコンポーネントの動作を記述する必要がある．このように，ホスト認証に限っても，考えられる限りの具体的なセキュリティ要求をすべて検証するのは困難である．
  また，期限付きサービス連携例題における期限遵守の要求については，変数 `mediator.deadline` で具体的に示される数値で期限を表すことにより記述した．一方，現実的な状況では，期限が守れそうにない場合は，交渉することにより期限を延ばしたい，といった要求もありうる．しかし，考えられるあらゆる状況を考慮して，Mobile UNITY による要求仕様記述を行うのは困難である．
  したがって，非機能的特性への要求の検証については，適切な要求仕様記述の選択，という問題の考慮も必要である．このような状況は，形式的仕様記述手法一般の問題であると考えられるが，今後の課題としたい．

5. 関 連 研 究

本章では，我々の手法を関連する従来手法と比較する．

モバイルエージェントセキュリティの研究は数多くある[2),10)~13),16),20),21),24)]．特に Necula ら[12)] は，形式的仕様記述に基づいた，PCC（Proof Carrying Code, 証明運搬コード）と呼ばれる手法を提案している．PCC は，モバイルコードの安全性の証明をコード自身に添付し，それらを一体のものとしてネットワークを移動させ，さらにサーバにおいて証明検査機構を動作させることにより，モバイルコードの安全性を保証する枠組みを提供するものである．PCC においては，証明が可能な限り自動的に生成でき，さらに証明検査が完全に自動的に実行できるような環境においてのみ適用可能なため，適用分野は限られたものとなっている．一方我々の手法は，形式的仕様記述の技術が望ましいような，より広範な分野に対し適用すること

ができる．

多くの研究者は，一般的なセキュリティに対するソフトウェア工学について検討しており[7]，形式的なセキュリティ検証技術についても議論されている．しかし，これらの研究は，一般的なソフトウェアのセキュリティ問題を対象としているので，モバイルエージェントに特有なセキュリティ問題[6]に適用できるかどうかは明らかでない．一方我々は，Mobile UNITY を形式的仕様記述および検証の枠組みとして採用しているので，本論文の手法はモバイルエージェントに特有な問題に適用することができる．

なお，仕様記述言語でなくても，モバイルエージェントの理論的な形式モデルをセキュリティ問題に適用する研究はいくつか実施されている[4),9),23)]．これらの研究は，Mobile UNITY とは異なり，理論的モデルとして提案されていて，仕様記述言語に必要な，構造的記述やその他記述を容易にするための構文糖（syntax sugar）などに欠けている．したがって，本論文の手法が対象とするような，大規模システムの記述・検証は困難である．しかし，モバイルエージェントアプリケーションのモデルとしては，メッセージ交換の記述が容易であるなど，Mobile UNITY にはない特徴を備えているので，仕様記述言語の核となるモデルとして利用し，さらに本論文の手法において Mobile UNITY の代わりに採用すれば，さらに効果的な開発手法とできる可能性があると考える．

6. おわりに

本論文では，IPEditor と Mobile UNITY を用いて，モバイルエージェントのセキュリティ問題を解決する手法を提案した．本手法では，IPEditor がエージェントの挙動の状態遷移モデルに基づいているので，IPEditor モデルから，エージェントの挙動の形式的仕様記述である Mobile UNITY プログラムをほぼ自動的に生成することができる．また，セキュリティ要求は Mobile UNITY 論理で記述する．そのうえで，Mobile UNITY プログラムが，したがって IPEditor モデルが，Mobile UNITY 論理記述を満足することを証明することにより，セキュリティ要求を検証することができる．また本論文では，電子カタログなどの例題に適用することにより，本手法の有効性を検討した．

今後は次のような方針で研究を進める予定である．

- モバイルエージェントは，開放型環境，特に構成要素が動的に変化するような環境において特長が発揮されると考えられている．しかし，Mobile UNITY は，例題から分かるように，構成要素の動的な変化を記述するには不十分である．このような問題に対し，次の 2 つの方針が考えられる．1 つ目は，リフレクションなど，動的変化の記述を容易にするような機構の導入である．2 つ目は，Mobile Maude[8] など，モバイルエージェント向けの他の仕様記述言語の検討である．
- 本論文では，文献 15) にあるように，プログラムが Mobile UNITY 論理で記述されたセキュリティ要求を満たすかどうかを検証する，という状況を検討した．一方文献 5) は，論理記述からプログラムを構築する枠組みを提案している．したがって，そのような枠組みを Mobile UNITY にも適用したい．
- 本論文の手法では，IPEditor モデルから Mobile UNITY プログラムが部分的に自動生成され，開発者が手直しを行うが，大規模なシステムになると，手直しのために自動生成されたプログラムを読んで理解するのが困難になる．そのため，手直しをなるべく少なくするか，理解が容易なプログラムを生成するなどの工夫を検討する必要がある．
- 本論文では，4.1.3 項に示したように，文献 5), 15) で紹介された UNITY および Mobile UNITY 特有の手法に基づく検証法を採用した．本検証法は，無限ループもありうる一般のプログラムに対して適用可能なものであるため，一般に完全な自動化が不可能である．

  一方，本論文の手法で作成される Mobile UNITY プログラムは，IPEditor の状態遷移のグラフ構造に基づいているので，有限オートマトンに対する検証法の適用も可能であると考えられる．そのような検証法を用いれば，自動検証の可能性もある．したがって，このような方向も今後検討したい．
- 本論文の手法にどのような課題があるかをさらに検討し，それらの課題の解決を図る必要がある．特に，本論文で取り上げた例題は，本手法の実現可能性を示すという目的のため，小規模かつ単純なものを選んでいる．一方，実際的な有用性を示すには，さらに大規模かつ複雑な例題に適用しなければならない．また，大規模な例題により，定量的な評価も必要である．
- 4.4 節で考察したような，様々な検証手法を取り込める柔軟性の実現や，非機能的特性への要求仕様の記述といった問題を検討したい．
- 本論文の手法を，Plangent[26] や Bee-gent の様々な実用的アプリケーションに適用し，手法の現実的な有効性を検証していきたい．

謝辞 研究の機会を与えてくださいました,(株) 東芝研究開発センターコンピュータ・ネットワークラボラトリー増渕美生元室長には深く感謝いたします.

参 考 文 献

1) Foundation of intelligent physical agents `http://www.fipa.org/`
2) Berkovits, S., Guttman, J.D. and Swarup, V.: Authentication for mobile agents, In Vigna[22], pp.114–136 (1998).
3) Burrows, M., Abadi, M. and Needham, R.: A logic of authentication, *ACM Trans. Comput. Syst.*, Vol.8, No.1, pp.18–36 (1990).
4) Cardelli, L.: Mobility and security, *Foundations of Secure Computation*, NATO Science Series: Computers & Systems Sciences, Bauer, F.L. and Mult, M.C. (Eds.), pp.3–37 (2000).
5) Chandy, K.M. and Misra, J.: *Parallel Program Design – A Foundation*, Addison Wesley (1988).
6) Chess, D.M.: Security issues in mobile code systems, In Vigna[22], pp.1–14 (1998).
7) Devanbu, P.T. and Stubblebine, S.: Software engineering for security: A roadmap, *The future of Software Engineering*, Finkelstein, A. (Ed.), pp.225–240, ACM (2000).
8) Durán, F., Eker, S., Lincoln, P. and Meseguer, J.: Principles of Mobile Maude, *Proc. ASA/MA 2000*, volume 1882 of *LNCS*, Kotz, D. and Mattern, F. (Eds.), pp.73–85, Springer (2000).
9) Fournet, C., Gonthier, G., Lévy, J.-J., Maranget, L. and Rémy, D.: A calculus of mobile agents, *Proc. CONCUR'96*, pp.406–421, Springer-Verlag (1996).
10) Hohl, F.: Time limited blackbox security; protecting mobile agents from malicious hosts, In Vigna[22], pp.92–113 (1998).
11) Karjoth, G., Lange, D.B. and Oshima, M.: A security model for aglets, *IEEE Internet Computing*, Vol.1, No.4, pp.68–77 (July/August 1998).
12) Necula, G.C. and Lee, P.: Safe, untrusted agents using proof-carrying code, In Vigna[22], pp.61–91 (1998).
13) Ousterhout, J.K., Levy, J.Y. and Welch, B.B.: The Safe-Tcl security model, In Vigna[22], pp.217–234 (1998).
14) Picco, G.P., Roman, G.-C. and McCann, P.J.: Expressing Code Mobility in Mobile UNITY, *Proc. ESEC/FSE '97*, volume 1301 of *LNCS*, Jazayeri, M. and Schauer, H. (Eds.), pp.500–518, Zurich, Switzerland Springer (September 1997).
15) Picco, G.P., Roman, G.-C. and McCann, P.J.: Reasoning about Code Mobility with Mobile UNITY, Technical Report WUCS-97-43, Washington University, St. Louis, MO, USA, Submitted for journal publication (December 1997).
16) Sander, T. and Tschudin, C.F.: Protecting mobile agents against malicious hosts, In Vigna[22], pp.44–60 (1998).
17) Tahara, Y., Ohsuga, A. and Honiden, S.: Agent system development method based on agent patterns, *Proc. ICSE'99*, pp.356–367, IEEE (1999).
18) Tahara, Y., Ohsuga, A. and Honiden, S.: Behavior patterns for mobile agent systems from the development process viewpoint, *Proc. ISADS 2001*, pp.239–242, IEEE Computer Society (2001).
19) Tahara, Y., Ohsuga, A. and Honiden, S.: Mobile agent security with the IPEditor development tool and the Mobile UNITY language, *Proc. Agents 2001*, pp.656–662, ACM Press (2001).
20) Tahara, Y., Yoshioka, N., Ohsuga, A. and Honiden, S.: Secure and efficient mobile agent application reuse using patterns, *Proc. SSR'01*, pp.78–85, ACM Press (2001).
21) Vigna, G.: Cryptographic traces for mobile agents, *Mobile Agents and Security*[22], pp.137–153 (1998).
22) Vigna, G. (Ed), *Mobile Agents and Security*, LNCS 1419, Springer (1998).
23) Vitek, J. and Castagna, G.: Seal: A framework for secure mobile computations, *ICCL Workshop: Internet Programming Languages*, pp.47–77 (1998).
24) Yoshioka, N., Tahara, Y., Ohsuga, A. and Honiden, S.: Security for mobile agents, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, Ciancarini, P. and Wooldridge, M. (Eds.), pp.223–234, Springer (2001).
25) (株) 東芝：Bee-gent WWW ページ. `http://www2.toshiba.co.jp/beegent/`
26) (株) 東芝：Plangent WWW ページ. `http://www2.toshiba.co.jp/plangent/`
27) 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, オブジェクト指向最前線 2001 情報処理学会 OO2001 シンポジウム, 大西 淳, 大須賀昭彦 (編), pp.95–102, 近代科学社 (2001).
28) 林 晋：プログラム検証論, 共立出版 (1995).

付　　録

A.1 例題の Mobile UNITY プログラムの一部

A.1.1 電子カタログ例題

```
System eCatalog[Manuf2Signature : string]
  ...
  program manuf2 at "manuf2"
    declare
        plibRetrieverChannel : string
     [] plibRetrieverChannelStatus : string
     [] state : string
     [] DB : stringList
     [] query : string
     [] signature : string
    initially
        ...
     [] DB = ...
     [] signature = Manuf2Signature
    assign
        ...
     [] state, plibRetrieverChannel,
        plibRetrieverChannelStatus
        = "retrieved",
            /* 検索結果通知メッセージ */
          "(inform ..."
          + sign(toString(retrieve(DB,
            query)), signature)
          + "...)",
          "sent"
          react-to state = "wait"
                   and query =\= ""
            /* 問合せが存在する場合 */
     [] state, plibRetrieverChannel,
        plibRetrieverChannelStatus
        = "resultsSent", ...,
            /* メッセージ文字列 */
        "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
        ...
     [] status, manuf2Channel,
        manuf2ChannelStatus, lambda
        = "Check", ..., "notUsed",
          "eCatalogServiceProvider"
        react-to status = "Return"
          and manuf2ChannelStatus = "sent"
    ...
  end

  Components
      eCatalogServiceProvider
   [] manuf1 [] manuf2 [] plibRetriever

  Interactions
      ...
   [] manuf2.plibRetrieverChannel,
      manuf2.plibRetrieverChannelStatus
      = plibRetriever.manuf2Channel,
        plibRetriever.manuf2ChannelStatus
      when plibRetriever.lambda = "manuf2"
        /* Bee-gent では，ホスト内での
           メッセージ交換が中心のため，
           本条件文を挿入 */
   [] ...
end
```

ここで，キーワード program の後にはシステム構成要素の名前が定義され，次のキーワード at の後には，構成要素が稼働している場所を表す変数が定義される．モバイルエージェントの移動は，本変数の値を assign 部で変更することにより表現される．また，eCatalogServiceProvider の仕様は省略したが，最初登録待ち状態で，データ登録メッセージを受け取ると，登録完了状態になり，その後すぐに登録待ち状態に戻る，という挙動となる．

A.1.2 トランザクション例題

```
System Airticket
  ...
  program airline1 at "airline1"
    declare
        mediatorChannel : string
     [] mediatorChannelStatus : string
     [] state : string
     [] DB : stringList
     [] tmpDB : stringList
     [] request : string
    initially
        ...
     [] DB = ...
    assign
        ...
     [] state, mediatorChannel,
        mediatorChannelStatus, DB
        = "committed", "", "notUsed",
          tmpDB /* Commit 操作 */
        react-to
          state = "wait"
          and mediatorChannel = "..."
               /* Commit 要求メッセージ */
          and mediatorChannelStatus
              = "sent"

  program mediator at lambda
   ...
  end

  Components
      customer [] airline1 [] airline2
   [] mediator

  Interactions
      ...
end
```

A.1.3 期限付きサービス連携例題

```
System Service-Integration-with-a-Deadline
  program timer at "timer"
```

```
  declare
      time : int
  initially
   ...
    [] time = 0
  assign
      time = time + 1
        /* 非同期に時刻を進める */
end

...

program server1 at "server1"
  declare
      mediatorChannel : string
    [] mediatorChannelStatus : string
    [] state : string
    [] query : string
  initially
      ...
  assign
      ...

program mediator at lambda
  declare
      server1Channel : string
    [] server1ChannelStatus : string
    [] server2Channel : string
    [] server2ChannelStatus : string
    [] state : string
    [] query : string
    [] results : stringList
    [] timeout1 : int
        /* server1 のタイムアウト */
    [] deadline : int
  initially
    ...
    [] time = 0
  assign
      ...
    [] state, server1Channel,
       server1ChannelStatus, results
       = "Move1", "", "received",
         add(results,
           resultMsgToResult
             (server1Channel))
           /* 結果通知メッセージから
              内容を抽出*/
       react-to state = "Query1"
         and server1ChannelStatus
           = "sent"
    [] state, mediatorChannel,
       mediatorChannelStatus
       = "Move4", "", "notUsed",
         react-to state = "Query1"
           and timer.time = timeout1
           /* タイムアウト時の処理 */
    ...
end

Components
    mediator [] client [] server1
 [] server2

  Interactions
      ...
end
```

(平成 13 年 10 月 9 日受付)
(平成 14 年 3 月 14 日採録)

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．1993〜1996 年情報処理振興事業協会に出向．1996〜1997 年英国 City 大学客員研究員．1997〜1998 年英国 Imperial College 客員研究員．現在（株）東芝研究開発センターコンピュータ・ネットワークラボラトリーに所属．エージェント技術，およ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（正会員）

1958 年生．1981 年上智大学理工学部数学科卒業．同年（株）東芝入社．1985〜1989 年（財）新世代コンピュータ技術開発機構（ICOT）に出向．現在（株）東芝研究開発センターコンピュータ・ネットワークラボラトリー主任研究員．博士（工学）．2002 年より電気通信大学大学院客員助教授ならびに大阪大学大学院非常勤講師兼任．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．

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 年より国立情報学研究所教授．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．工学博士（早稲田大学）．主としてエージェント技術，オブジェクト指向技術の研究に従事．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会，IEEE，ACM 各会員．著訳書多数．