モバイルエージェントの将来

ここでは，モバイルエージェントに関する将来について述べてみたい．

モバイルエージェントと一言で言っても，そこにはいろいろな概念・技術が含まれていて，その将来は１つにはならない．そこで，モバイルエージェントという言葉（ラベル）が内包する，それぞれの技術・概念について，順を追って，その将来がどのようになりそうかを，私の視点からまとめてみたい．

- 移動アプリケーション(Migratory Application)

移動アプリケーションとモバイルエージェント

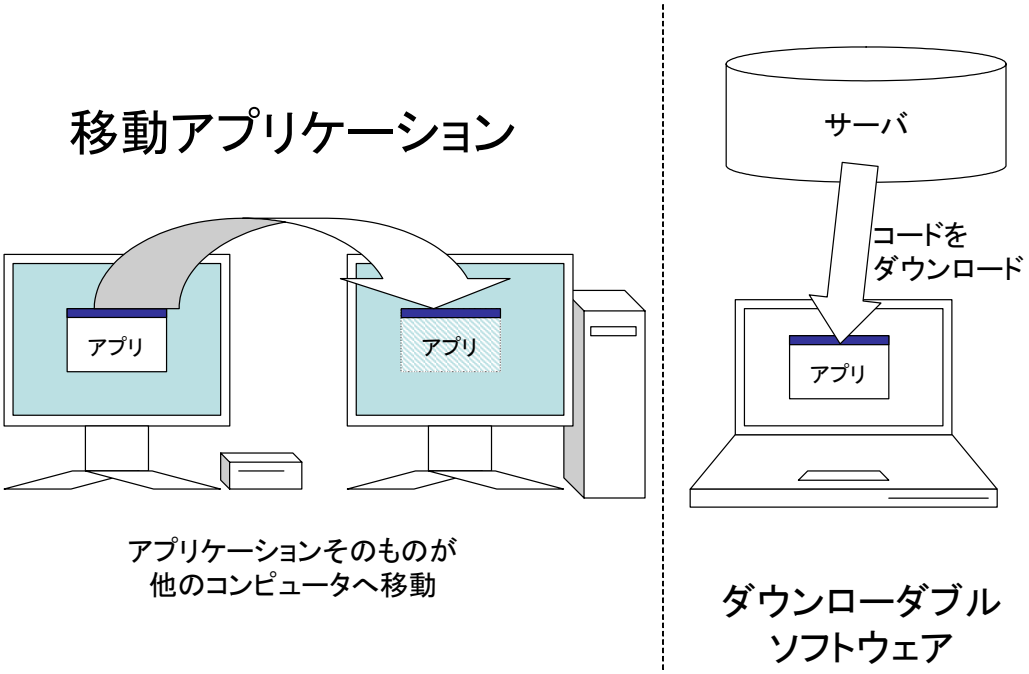


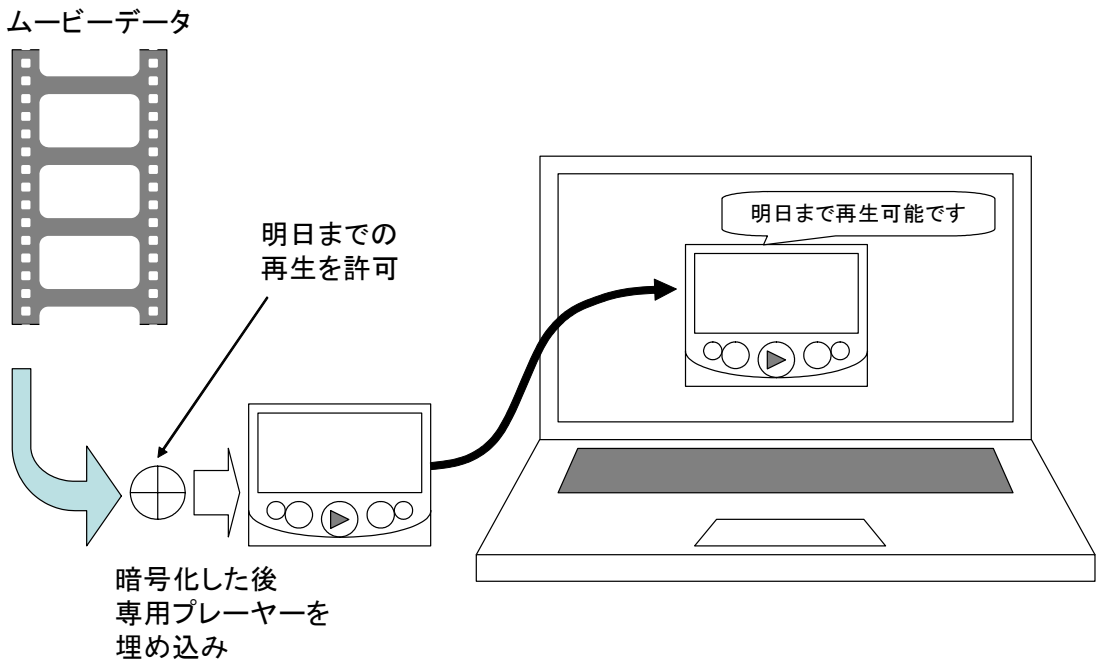
図: 移動アプリケーションとダウンローダブルソフトウェア

移動アプリケーションとは，モバイルエージェントの背景になっている「概念」である．移動アプリケーションは，アプリケーションソフトウェアでありながら，複数のコンピュータ間を「移動」できる能力を持つ．この定義は，かなり広くとらえることができ，広い意味では，ダウンローダブルソフトウェア(Downloadable Software)も移動アプリケーションに入る(図)．ダウンローダブルソフトウェアには，たとえば，Java Applet や，Flash アプリケーション，Ajax なども(Ajax は条件によってはその扱いが微妙だが)含めることができる．つまり，非常に広い意味では，すでに移動アプリケーションが普及しつつある．

この（原始的な）移動アプリケーションは，データの保存機能を備えるようになることで，より（本質的な）移動アプリケーションに近くなっていく．たとえば，Google が買収した Ajax ワードプロセッサアプリケーションの Writely では，保存していた文書を（ブラウザさえ動けば）どこでも編集することができる．つまり，これは別のコンピュータにワードプロセッサを移動させて使うことができることに，非常に近い操作感覚を生む．現状では，Ajax 技術を使って作成されたアプリケーションは，インターネットに接続されていないと使えないが，Flash や他の技術をうまく併用することで，まさにアプリケーションソフトウェアをいろいろなコンピュータに移動させながら使うことができるようになる．

さらに，(GMail のような Ajax を使っていない)Web メールシステムのような，サーバサイドで動作する Web アプリケーションであっても，ユーザの目に見えないところで，必要に応じて別のサーバ上に移動して動作を続けさせるような技術が実現されつつある．このように，ユーザの目に見えない形で，移動アプリケーションが使われていくというシナリオも考えられる．

移動アプリケーションと DRM



図：DRM 機構を埋め込んだ移動アプリケーションの例

移動アプリケーションの将来を握る鍵の 1 つは，デジタルデータに対する管理(DRM, Digital Rights Management)機構にある．DRM 機構の実現には，デジタル化された著作物（音楽・動画・画像など）に対する利用の形態（閲覧・改変・２次利用・利用期間など）を，安全に，柔軟に，かつユーザにとっても扱いやすい形で管理することが求められる．

それをもっとも確実に実現する方法が，デジタルデータそのものに，その閲覧用アプリケーションなどを組み込んでしまう方法である．たとえば，さきごろ Google 社に買収された YouTube では，Flash アプリケーションとして実装された専用の閲覧ソフトウェアを Web ページ内にダウンロードして，その閲覧ソフトウェア上でのみ映像を見られるようにしている．これも，一種の（原始的な）移動アプリケーションであるといえる．

現在でも，たとえば Windows Media Player のように，独自の DRM 技術を組み込んだプレイヤーを用意し，その DRM 技術に沿ってデジタルデータの利用を管理することができる．デジタルデータとそのプレイヤーアプリケーションをセットにする利点は，その管理方法を，プレイヤーの実装に左右されることなく柔軟に設計できる点にある．

移動アプリケーションの DRM への応用を，より抽象的な観点からみると，それは，データがコード（プログラム）でカプセル化されるということである．これは，オブジェクト指向の考え方が，アプリケーションのレベルでも実現されてくるということである．そこに，単にデータがコードでカプセル化されるというだけでなく，そのデータの移動にもコードが積極的に関与するようになるという点が，移動アプリケーションを使った場合の特徴になりうる．将来は，たとえば「今手元でみた映画をお友達に貸す」といったような，我々にとってもっとなじみの深いデータの「移動」の動作が実現できるようになってくると考えられる．

より技術的な観点に話を移すと，移動アプリケーションによる DRM の高度化の鍵になるのは，「効率」と「頑健性」の２つになると思われる．ここでいう「効率」とは，アプリケーションの実行の動作速度であり，同時に，アプリケーションのダウンロードにかかるネットワーク負荷のことである．

前述の Flash による動画プレイヤーは，実際にはその動作の核となる Flash Plugin を何らかの手段でインストールしておく必要があり，その点で，すべてのコードをアプリケーション自体が含んでいるというわけではない．つまり，Plugin という固定部分と，Flash という動的なダウンロード部分の２つの種類のコードが存在する．

モバイルエージェントの「効率」は，どの部分を「移動しないコード」，つまりプラットフォーム（この場合だとプラグインそのもの）に含め，どの部分を「移動するコード」とするかに強く依存する．「効率」の問題は少し複雑で，ネットワークを転送されるコードの分量も考慮する必要があり（データ転送効率），なおかつ「移動しないコード」と「移動するコード」では，それが実行されるときの動作速度（動作効率）が異なる場合がある．たとえば，Flash プラグイン内部にエフェクト処理を組み込むのと，Flash ファイルにエフェクトのような処理をプログラムで書いておくのとでは，動作速度は前者のほうが高速になる場合がほとんどである．この，移動するコードと移動しないコードをバランスさせるのは難しい問題だが，今の段階では，「すべてのコードを移動するコードとしなくても良い」という観点だけが重要であるので，これ以上の議論はしないことにする．

もう１つの「頑健性」とは，ここでは特に「耐タンパ性」の意味である．耐タンパ性を

持ったソフトウェアとは，ソフトウェアとして正常な動作をしつつ，なおかつそのソフトウェアが動作するメカニズムや内部のコードの解析・改変を困難にしているソフトウェアである．たとえば，３日間しか視聴できないようにしたプレーヤーのコードを勝手に改変して，３００００日間視聴できるようにしてしまえると，DRM としての意味をなさなくなってしまう．一般に，プラットフォーム（移動しないコード）とその上にダウンロードされるコード（移動するコード）とでは，移動しないコードのほうが解読されにくいことが多い．しかし，移動しないコードは，仮に解読されてしまった時に，再び解読されないようにするための修正を行いにくい．（この問題に関しては，モバイルコードの問題とは少し外れるが，DVD Video における CSS 暗号の問題がよく知られている．CSS 暗号を扱うコードは，DVD プレーヤー内の ROM に格納されていて書き換えが非常に困難なため，CSS 暗号を破られた後も，すでに売られた DVD プレーヤーで再生できるようにするために，使い続ける必要が出てきている．）

移動アプリケーションの将来とモバイルエージェント技術

移動アプリケーションの将来に対して，モバイルエージェント技術側に起きる変化がどんなものかを想像してみる．

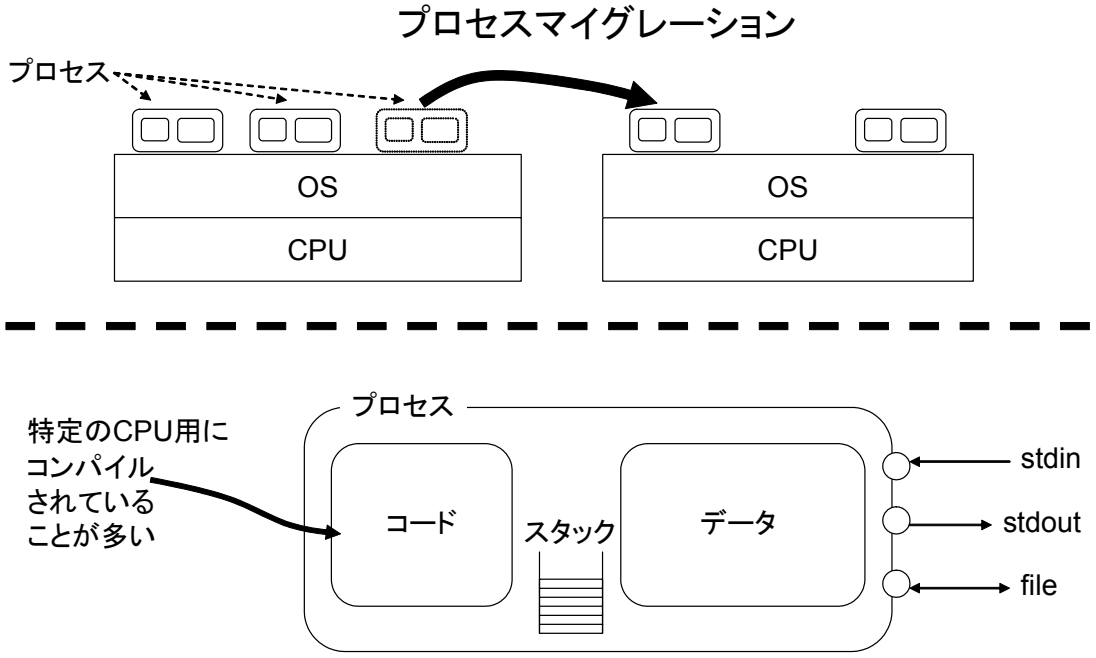
移動アプリケーションを動かすためには，プラットフォームが必要である．そのプラットフォームは，できるだけ共通ですでに普及しているものであったほうが，アプリケーションを配布するときには有利である．今，もっとも有力な共通プラットフォームは，Web である．Web 上では，すでに，Ajax や Flash アプリケーションなど，様々なアプリケーションがダウンローダブルコード（１度だけ移動可能なコード）として利用可能となっている．１つの技術的な目標としては，Web というプラットフォームで，ダウンローダブルコードをモバイルコード（何度でも移動可能なコード）にすることがある．Web をプラットフォームにする場合，すでにあるプラットフォームを利用する都合上，どうしても制約事項が多く発生してしまい，強モビリティのようなプログラマビリティは確保することが難しい．そこで，まず最初の１歩としては，弱モビリティ程度の最低限のプログラマビリティを確保しつつ，そこで具体的にアプリケーションを作っていくことが重要となる．この点で，MiSpider は，この視点に立って時代を先取りしたアプリケーションフレームワークである．

しかし，これはすでに実現されていることであって，将来ではない．MiSpider のようなアプリケーションの実現の過程でわかってきたことであるが，Ajax アプリケーションが持つネットワークの切断に対して弱いという性質は，Ajax の枠組みに納まる限り，どうしても改善に限界がある．そこで，さらにその次には，ネットワーク切断にも頑健な，Web をプラットフォームとしたモバイルコードの実現が来ると予想される．その１つの可能性に，Web でなく Web ブラウザをプラットフォームとすることがあるかもしれない．言葉の綾でわかりにくいかもしれないが，Web そのものは，本質的にネットワークそのものであるか

ら，ネットワークの切断に弱いのは仕方がない．しかし，Web ブラウザはローカル（ユーザの手元）で動くアプリケーションソフトウェアなので，（シンクライアントのように手元でアプリケーションが動作しないような場合を除けば）ネットワークの切断にもある程度柔軟に対応できる．では，Web ブラウザをプラットフォームとするとはどのようなことかというと，その 1 つは，Web ブラウザに組み込まれる機会が増えてきている Flash や Java などの仕組みをうまく利用することである．実際，Flash を用いれば，ローカルに文書が保存可能なテキストエディタを作ることは容易である．セキュリティ要件を確保した状態で，Web ブラウザ上のアプリケーションと Web が連携していくのには，まだハードルが高い．Web ブラウザ上でのセキュリティ機能の要件によって, Web と Web ブラウザのどちらがプラットフォームになり得るのかが，決まってくる．今の段階では，技術的にはその両者ともを追求していき，とにかく移動アプリケーションを普及させてしまうことが重要と思う．

移動アプリケーションが，本当に一般に深く浸透し，その利便性が広く一般のユーザに受け入れられたとき，さらにもう一段上の技術革新が必要になる．そのための基礎技術を積み上げておくことが，MiLog を作った最も重要な理由の 1 つである．

- OS のメカニズムとプロセス・バーチャルマシンマイグレーション
  プロセスマイグレーション・OS・仮想化



図：OS 上のプロセスとプロセスマイグレーション

モバイルエージェントの研究が始まる少し前，OS(オペレーティングシステム)のメカニ

ズムの１つとして，動作中の計算プロセスを他のコンピュータに移動させる技術としてプロセスマイグレーションが研究されていた．プロセスマイグレーションでは，計算プロセス(UNIX でいうなら ps コマンドで表示されるような，あのプロセス)を移動の単位としていた．プロセスは自身で I/O(入出力: Input/Output)を持つため，その仮想化が１つの大きな課題であった．（個人的な見解としては，この仮想化で難しいのは，I/O 仮想ライブラリを用意することではなく，むしろそのプロセスがプログラミングされたときの様々な誤った仮定を前提としても頑健に動作する仕組みを実現することだと考えている．たとえば，ファイルへのアクセスに１度でも失敗したらその場でプログラムを異常終了するようなプログラムを作ることはよくあるが，このようなプログラムでは，プロセスマイグレーションには耐えられない可能性が高い．この場合，プログラム（＝コード）を直すだけでなく，それを作るプログラマ（＝人）の認識を改める必要もある．この意識の変化を浸透させるには，かなり時間がかかると思われる．現に，「脆弱性を持たないプログラムを作るようにする」という方向への意識の変化ですら，あれだけコンピュータウィルスが蔓延してようやく，浸透し始めたのであるから.）

また，プロセスはその OS が動作する CPU に最適化されたコードとしてコンパイルされてから実行されることがほとんどだったため，別の CPU アーキテクチャ上で動作する OS 上にプロセスを動かすには，非常に複雑な仕組みが必要とされた．

プロセスマイグレーションは，SONY のロボット犬，AIBO に搭載された OS の前身のものには搭載されていたらしいが，我々が普段使うオペレーティングシステムでは，利用できない場合がほとんどである．将来，プロセスマイグレーションが OS に搭載されるようになるかどうかについては，個人的見解としては「おそらく搭載されないのではないか」と思う．

ただし，プロセスマイグレーションとは別の側面から，急速にモビリティと呼んでもよいような機能が普及してくる（すでにかなり普及しつつある）と考えられる．プロセスマイグレーションでの課題は，プロセスという単位を移動の単位として扱ったことであった．この単位をより大きくするアプローチがあっても良い．それが，OS そのものの仮想化と，OS を動かすコンピュータそのものの仮想である．

すでに，多くの Macintosh ユーザには，バーチャルマシン（仮想機械上）で，Windows を動作させていることに，なじみがある．仮想化技術の１つのボトルネックは処理速度であったが，すでに CPU そのものに仮想化を支援する（すなわち，仮想化による速度の低下を小さく押さえる）ための機構が搭載されつつある．これらのハードウェア側からの支援をうまく利用すると，非常に小さいオーバヘッドで，OS やコンピュータそのものを仮想化できるようになる. OSやコンピュータそのものが仮想化できれば, その動作を一旦 suspend してからデータファイルをまるごろ別のコンピュータにコピーして，コピー先で動作を再開させてやれば，簡単に，仮想化された OS やコンピュータを「移動」できるようになる．すでに，仮想化ソフトウェアの中にはこの機能を備えるものもすでにあるようで，急速に

技術的な問題が解決されつつある．これを，ここではバーチャルマシンマイグレーションと呼ぶことにする．

バーチャルマシンマイグレーションは，すでに現実のことであり，将来のことではない．ここでは，バーチャルマシンマイグレーションが一般化したときに，何が起きるのか，どのような技術的・非技術的なアプローチや問題が出てくるのかを想像してみる．

バーチャルマシンマイグレーションが一般化すると，マイグレーションを前提としたOS,CPU やコンパイラの最適化技術が開発されてくることが予想される．具体的には，OS 自身のフットプリント（定常的メモリ消費量やハードディスク占有量）を小さく済むようにしたコンパクトな OS や，休止状態への移行に必要な時間・データ領域を高速化・コンパクト化した OS，複数の CPU アーキテクチャに対応した異種命令実行を考慮した CPU の実装技術，さらにコードをできるだけコンパクトにしつつ動作速度とのバランスを取るようなコンパイラのコード生成技術などが，(すでに現在でも一部あるが)今以上に進歩することが予想される．また，OS の基幹部分のみを CPU ネイティブコードで動かし，OS 上のアプリケーションのほとんどすべてをバーチャルマシン上で動かす，Java や.NET のアプローチも，コードサイズが小さい点などから，今よりも優位性が高くなる可能性がある．

MiLog とバーチャルマシンマイグレーション

MiLog の実装では，各エージェントごとに１つずつバーチャルマシン（インタプリタ）を用意し，そのバーチャルマシンごと強モビリティで移動させるというアプローチを取っている．その上で，そのバーチャルマシン上で動作するコードをいかに小さなフットプリントで表現するか，実行途中の過程をいかに小さなデータとして表現可能とするか，その表現変換にかかるオーバヘッドをいかに小さくできるかの３点について，検討を行っている．これは，（言語として Prolog 言語の亜種がその検討対象として適切だったかどうかはともかくとして）上述のバーチャルマシンマイグレーションの普及を先取りしたものである．その過程でわかったことの１つに，バーチャルマシンマイグレーションで要求される条件は必ずしもこれまでに経験したことのある最適化手法とは一致しないため，従来は注目されていなかったプログラミング言語・OS・CPU アーキテクチャが，ここにきて大きく注目を集め使われるようになる可能性があるという点である．（ただし，現時点で，具体的にどの言語や OS がそうなるかは，まだ予測がつかない．）

マルチ・バーチャルマシン環境とモバイルエージェント

バーチャルマシン上で複数の OS を同時に動作させることが一般的になると，バーチャルマシン同士でのデータや操作感の連携が重要な要素になってくる．たとえば，すでに Virtual PC 上で Windows XP を動作させる場合，ホスト側のコンピュータとバーチャルマシン上の Windows の表示画面とで，マウスカーソルをスムーズに移動させる技術ができている．これは，現状では，バーチャルマシン上の Windows に特殊なドライバを導入し，バーチャル

マシン管理ソフトウェアとの連携を密に行えるようにすることで，実現されている．将来は，バーチャルマシン・OS 間でのマウス・キーボード操作や簡易なデータ転送がなんらかの形で標準化され，使い勝手が向上していくことが予想される．

その，複数バーチャルマシン利用時の利便性向上の１つのシナリオとして，そこにモバイルエージェント技術が適用される可能性は，ありうる．もっともわかりやすいものは，移動型クリップボードアプリケーションである．移動型クリップボードは，ボタンを押すと目的のバーチャルマシン上に移動することができ，そこに持つクリップボードエリアにさまざまなデータを貼り付けたり切り取ったりすることができる．OS 間でのクリップボード共有という方法もあるが，移動型クリップボードのほうが，わかりやすく，かつ仮想・実体 OS の区別無く使えて，便利なのではないかと思う．MiLog のエージェントをこの用途で使ってみると，非常に直感的で使いやすいことがわかる．

もう１つのモバイルエージェント技術が適用される可能性のあるものに，OS をまたいだモバイルスクリプト処理がある．たとえば，「Windows 上で高性能のかな漢字変換エンジンを使って編集した TeX 文書を Linux 上の Tex プログラムで整形した後，それを Macintosh に対応したプリンタから印刷する」といった処理を，モバイルエージェント技術を使えば，比較的簡単にスクリプト処理として書ける．

移動クリップボードやモバイルスクリプト処理は，必ずしもモバイルエージェント技術のキラーアプリケーションとなりうるほど強力なものには見えないかもしれない．しかし，こうした小さなアプリケーションの普及を通じてその存在が広く認知されていくことで，モバイルエージェント技術を学ぶ人が増え，次なる発展を遂げる，という可能性はあると思っている．

バーチャルマシンマイグレーションとそこで実現されるモビリティの質

バーチャルマシンマイグレーションが一般化すると，OS の機能分化も進むことが予測される．より具体的には，複数のバーチャルマシンをホストとして駆動するためのホスト OS と，個々の目的に特化した(とんがった)OS の２種類に分化していくことが予想される．すでに，リアルタイム OS である TRON(ホスト OS)上で動作する Windows(ゲスト OS)などが使われるようになってきているが，この流れがさらに進むと，アプリケーションソフトウェアを選ぶのと同じような気軽さで，OS を選択的に利用するようになる．こうなると，アプリケーションソフトウェアと OS の間の差が非常に小さなものになる．

ここで重要な点は，アプリケーションソフトウェアには，その実行状態全体を保存・復元する機構を組み込むために高いハードルがあるが，OS とそれを動作させているバーチャルマシンには，その状態保存機能がすでに実現されてしまっている点である．つまり，バーチャルマシンマイグレーションにより，実質的にアプリケーションソフトウェアに強モビリティが備わってしまう状況が起きうる．目的に特化した OS とその特性を生かしたアプリケーションの組み合わせにより，今後，魅力的な移動アプリケーションがたくさんでき

てくるかもしれない. そのときに実現されるモビリティは, 弱モビリティを通り越して, (ほぼ完璧な)強モビリティである.

バーチャルマシンマイグレーションから派生する移動アプリケーションが強モビリティであり，それが弱モビリティよりも急速に普及する可能性があるのなら，強モビリティの実装技術（単なる実現方法ではなく，その効率や頑健製を高める技術）を研究しておくことは非常に重要ではないかと思われる．一方で，強モビリティの実装技術に関する基礎研究は，まだ発展途上である．MiLog で行ったいくつかの技術的チャレンジは，このギャップを埋めることを狙っているが，まだそれを十分に埋めるには至っていない.

バーチャルマシンマイグレーションでは，マイグレーション時の実行効率と，実装にかかるコスト（開発・管理コスト）との関係が，アプリケーションのマイグレーションとは異なってくる可能性がある．バーチャルマシンや OS は非常に複雑な仕組みで動いており，その内部での無駄を探して除去するには，（単一のアプリケーションと比較して）非常に大きなコストを要する．たとえば，OS 起動時のプロセスの数を減らすために複数のプロセスを単一のプログラムにまとめようとすると，そのプロセスを利用する起動スクリプトの調整や権限の管理の問題に波及し，それらをすべて調整するために非常に大きなコストがかかる．この結果，バーチャルマシンマイグレーションでは，移動アプリケーションの実装以上に，マイグレーションのコストが（普及する当初は）重視されなくなる可能性がある．こうなると，弱モビリティの研究者が当初主張していた，強モビリティの３つの問題点（状態保存機構の実装，マイグレーション時の効率，既存のプラットフォーム・言語との互換性）は，いずれも問題にならなく（あるいは，されなく）なる．ここに，強モビリティに関する研究の知見が生かされる余地があると考えている.

- **ソフトウェアコンポーネントとコードモビリティ**

  すでに進みつつある研究

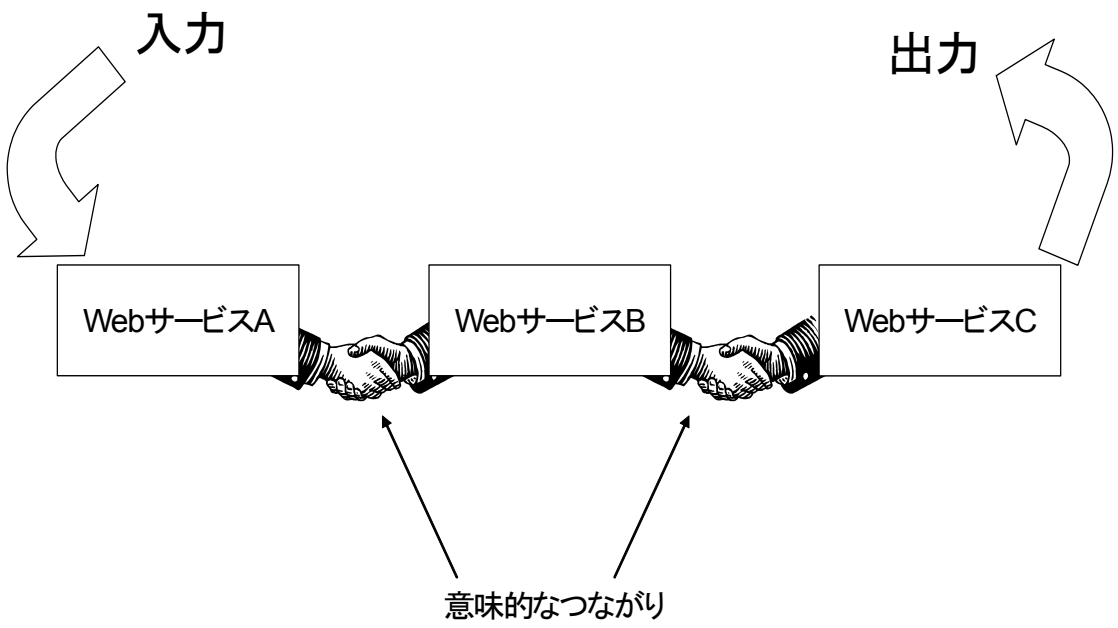
ソフトウェアコンポーネントという単語一般の指す意味は広いが，ここで扱う「ソフトウェアコンポーネント」とは，それ自身が単体では意味のある結果をユーザに返せないが，そのいくつかを組み合わせることで，完成したソフトウェアとして意味のある結果をユーザに返せるようになるような，ソフトウェアの半完成部品である.

ソフトウェアコンポーネントで重要な点は，その「半完成部品」自体が普及・流通していて，それらを組み合わせることでアプリケーションソフトウェアの実装をほぼ完成させることができる点である.

ソフトウェアコンポーネント単位でのモビリティの実装については，すでにいくつか興味深い研究がなされてきており，今後も目が離せない分野の１つである．ソフトウェアコンポーネントがモビリティを持つようになると，ブラックボックス的なコンポーネントを組み合わせながらも柔軟にアプリケーションソフトウェアを実現できるようになる．たとえば，デジタルデータをカプセル化したソフトウェアコンポーネントと，デジタルデータ

の再生操作用に特化したソフトウェアコンポーネントを自由に組み合わせて，保護されたコンテンツを好みの操作感で鑑賞することができるようになる. たとえば, ユーザは iTunes のような統合型のプレイヤーで複数のデジタルデータを統合的に管理したいというニーズを持っているが，デジタルデータの提供者側はデジタルデータの利用方法を「同時に再生するのは 1 台だけ」という形に制限したい，といった場合がそうである．このとき，統合型プレイヤーのコンポーネントの元に，デジタルデータ管理コンポーネントが移動してきて自動的に接続されれば，ユーザの利便性は低下させることなく，デジタルデータ提供側の管理方針を反映できるような仕組みを実現できる.

ソフトウェアコンポーネントとサービスとセマンティクス



図：セマンティック Web サービスの基本的な考え方

ソフトウェアコンポーネント技術の 1 つの流れとして，ソフトウェアを部品ではなく「サービス」(＝プログラムコードとその運用計算リソースとのセット) と見なす, SOA(Service Oriented Architecture，近年は SaaS(Software as a Service)という考え方もある)がある. 現状の SOA のアーキテクチャは, XML に基づくデータ通信部分のみを規定した Web サービスに基づくものが多い．この Web サービスに基づく SOA とモビリティ技術とは，あまり相性が良くない. なぜなら，(現状の)Web サービスの基盤はネットワーク上でのロケーションアウェアな仕組みを前提としているが，ソフトウェア（コンポーネント）にモビリティが備わると，このロケーションがころころと変わってしまうことになる．もちろん，

Dynamic DNS などの仕組みを利用することである程度問題を緩和できるが，本質的に「ロケーションアウェアでない仕組み」に移行しない限りは本質的な解決には至らないと思われる．現状の勢いで SOA が発達してしまうと，モビリティという技術を適用しづらい状況が起きてくる．

そこで改めて重要になってくるのが，サービス（あるいはソフトウェアコンポーネント）をアドレスでダイレクトに示すのではなく，そのサービスの内容・意味で指し示すことができるようにする技術である．最近私が研究しているのはこの部分で，どのようにしたらサービスの入出力やその意味を「意味として」表現できるのか，それらをどのようにつなげていったらいいのか，そのときの「スペックに現れない品質」（ユーザの好みへの合致性など）をどのように確保したらいいのかが主な研究対象になっている．研究分野としては，セマンティック Web サービスという名前で呼ばれている．このセマンティック Web サービスという名前は，セマンティック Web という「意味を意味として表現・共有する仕組み」と「Web サービス」という SOA の仕組みを組み合わせることを可能にする，という意味から来ている．この研究がうまくいけば，ソフトウェアコンポーネントが必要に応じて勝手にいくつか集まってきて，所望のサービスをユーザに提供してくれるような世界が実現できる．このときに，所望のサービスを「手元に持ってくる」という形で，あるいは大切な個人情報をカプセル化して守るという形で，モビリティの技術が生かされるのではないかと期待している．

ソフトウェアコンポーネントとエージェント

技術的な視点でモビリティとソフトウェアコンポーネント技術を眺めてみると，アプリケーションソフトウェア同士での連携との本質的な違いがどこにあるのかが気になってくる．どこまでならソフトウェアコンポーネントで，どこまで完成されればアプリケーションなのか．アプリケーションソフトウェアとソフトウェアコンポーネントが違うもの（単位）であることに，本質的な意味はありうるのか．もし違うものであるとしたら，どういう違いであって，どうその違いを生かせばいいのか．こうした問いに答えてくれる研究は，意外と少ないように感じている．ここでは，技術的な観点からみたソフトウェアコンポーネント技術とそのモビリティの生かし方について考察してみる．

モビリティでは，その移動の「単位」が重要になる．つまり，移動をするが故にその移動の単位（別の言葉で言うなら，範囲）が明確に定まるので，その移動の単位がソフトウェアコンポーネントであるなら，そのソフトウェアコンポーネント自身にもその単位（あるいは範囲）が（意図してかどうかはともかくとして）明確化される．たとえば，ソフトウェアコンポーネントがさらに細かなソフトウェアコンポーネントを「含む」という入れ子構造が許されるとき，その「含む」という表現自体が適切かどうかが，モビリティ基準にを考えると怪しくなってくる．それらは単に親コンポーネントと一緒に移動しているというだけで，「含まれる」ものではないのではないかもしれない．もし仮に，特定のソフト

ウェアコンポーネントを移動させるためのプラットフォームが「親」で，そこで移動するソフトウェアコンポーネントが「子」であるとしてもなお，それは互換性が限られたコンポーネントとそれを接続可能にするようなアダプタコンポーネントと見なした方が自然に思えてくる．ソフトウェアコンポーネントを扱うからこそ，入れ子構造になったコンポーネントを移動させるニーズが出てきて，それが新規性であるということを主張する研究者もいるが，私個人としては，モビリティを前提に考えたときには，ソフトウェアコンポーネントには親子関係のような階層構造は本質的なモデルとしてはふさわしくなく，単純にpart-of 関係(ある要素がいくつか集まって，ある別の要素を構成するときの関係)として扱うべきではないかと思うし，part-of そのものは(親の性質を子が引き継ぐといった関係でないから，親子関係を前提とした意味での)階層構造とは異なる構造であると思う．このpart-of 関係そのものは，対象がソフトウェアコンポーネントでなくてはならないというものではないので，アプリケーション間連携やエージェント間協調と同じ土俵で議論してかまわないと思う．

そう考えたときに，アプリケーション間連携やエージェント間協調とソフトウェアコンポーネント間の連携で本質的な違いは何があるのか？１つの見方として，連携に使う通信プロトコルの抽象度が異なるという見方がある．言葉から受ける印象としては，ソフトウェアコンポーネント間での通信はアプリケーション間通信よりも抽象度が低そうに見えるし，エージェント間協調での通信プロトコルは，なんとなくアプリケーション間通信よりも抽象度が高そうに見える．ところが，実際にはソフトウェアコンポーネント間の連携にエージェント間通信プロトコルを使っている例もあったりする．別にソフトウェアコンポーネント間連携にエージェント間通信プロトコルを使って何か害があるわけではないが，それをエージェント間協調とは分けて，あえてソフトウェアコンポーネント間の連携として改めて扱う必要がどこにあるのか，一見してよくわからない．

私は，ソフトウェアコンポーネント間の（知的な）連携を考えるとき，そのキーポイントになるのは，実装上・運用上の制約であろうと思う．たとえば，エージェント間協調では,「協調動作のために必要なメモリは 100kbyte 以下でありその応答時間は 100msec 以内であることが保証されなければならない」といったことは，（特に前者のメモリーの条件などは）あまり議論されない．つまり，ソフトウェアコンポーネントを扱う場合には，その対象とする問題が本質的になんらかの実装上・運用上の制約を含んでいなければおかしいし，その制約の中で最大限の性能を発揮できるようにするための研究であるはずである．もしそうなら，その性能を向上させるための手段，制約をくぐり抜けるための手段として，必然的にモビリティが使われるようになると思う．こうした必然性の議論なしにソフトウェアコンポーネントの移動を扱っている研究に出会うと，少し寂しい気分になる．（と，自戒の念を込めて書いておく．私もそういうことをしないように気をつけるつもりです.）

- プログラミングパラダイムとしてのモバイルエージェント

モバイルエージェントプログラミングというパラダイム

プログラミングパラダイムとしてみたモバイルエージェントは，ソフトウェアのプログラミングの過程に「コンピュータ間を移動できるような主体」という概念を含めることで，プログラム内部の処理モデルを簡略化することを目的にしている．ここでは，移動アプリケーションのようにモビリティがユーザに見えるところに出てくることは必ずしも仮定されないし，モビリティを使って通信路の切断に対する頑健性を向上させたりネットワーク遅延をモビリティで相殺したりといったことを必ずしも考慮しなくて良い．その興味の中心は，いかに簡単にプログラミングできるようにするか，にある．

プログラミングパラダイムとしてみたモバイルエージェントでは，単に移動の主体（単位）を１つのモバイルエージェントとし，（モバイル）エージェントという言葉にはそれ以上の意味はない．エージェントは，いわゆるエージェントらしくなくてもよいし，移動する単位がいわゆる「ソフトウェアコンポーネント」と呼ぶにふさわしいものでなくてもよい．ただし，１つだけ重要な点があり，プログラミングパラダイムであるから，移動そのものをきちんと抽象化でき，それを発展させることができる必要がある．具体的にいうと，「あるホストに移動する」という移動の機能を土台にして，「ある条件を満たすホストをあちこち移動した結果見つけてそこに居座る」という新しい移動に抽象化でき，それを次の移動の機能の抽象化の土台にできるようになっていることである．弱モビリティでは，このような移動の機能の抽象化を行っていくことが，その構造上きわめて困難である．プログラミングパラダイムとしてのモビリティは，強モビリティを出発点とすべきだと思う．それをふまえた上で，その強モビリティの記述を弱モビリティの組み合わせにコンパイルしていく仕組みなどを開発するといった方向性を取るべきだろうと思う．

プログラミング言語とプログラミングパラダイム

新しいプログラミングパラダイムは，常に新しいプログラミング言語とともにある．あるいは，少なくとも密接な関わりを持って相互に進化するものである．そこで，プログラミングパラダイムとしてのモバイルエージェントを考える場合にも，それを実現するプログラミング言語と対で考えることが重要となる．

プログラミングパラダイムとしてのモバイルエージェントを考える場合，私は，それが既存のプログラミング言語の枠には収まりきらないと思う．既存のプログラミング言語やライブラリの設計との互換性を厳しく求めすぎても足かせになるだけで，むしろモバイルエージェントプログラミングというパラダイムにとって最適な言語・ライブラリ設計を考えていくことこそが建設的な方向であると思う．

モバイルエージェントを前提とした場合，すべての I/O 処理はモバイルエージェントから本質的に独立させるべきで，間違っても，ファイルハンドルを持ってデータを先頭から操作するようなことはモバイルエージェントにはさせるべきでないと，私は思う．

モバイルエージェントは移動をする．それがいつ行われるかもわからないし，戻ってくるかどうかもわからない．移動前と移動後に同じファイルを参照したいかもしれない．そういうモバイルエージェントの特性を考えると，I/O 処理，とくにファイル処理に関しては，プログラミング上の仮定を大きく変えないとうまくいかない．単に I/O 処理は常にエラーがつきものであるという（Java でいうところの Exception 処理を強制するといった程度の）扱いでは不十分で，いかなる場合でも I/O 処理のエラーからリカバーできるようなプログラムの設計が必要になる．

この，「エラーからリカバーする」という点を考えると，C や Java をはじめとした既存の多くのプログラミング言語では，サポートが非常に弱いように（私の主観では）感じる．私個人としては，現時点でもっともエラーからのリカバリー能力の高い言語は，論理型言語(特に Prolog)であろうと思う．最近，エラーからの(自律的)リカバリー能力をうたった OS なども出てきているが，エラーリカバリーの頑健性は今後のトレンドの１つに確実になってくると思う．そんなときに，エラーからのリカバリーをまるで考えない言語を使ってプログラミングしていていいものか，と思う．

MiLog では，プログラミング言語として Prolog のコアを使い，I/O ライブラリはほとんど再設計している．これは，Prolog 言語のもつエラーリカバリーの強さと，モバイルエージェントとしての I/O 処理を備えるという観点から，そうしている．こうした私の主張は，Prolog の教科書にはもちろん書かれていないし，それ故か，なかなか信じてもらえない．MiLog の実装を完全にフルスクラッチで行っているのも，その実装上の制約がモバイルエージェントプログラミングというパラダイムの良さをスポイルしないためである．実行効率が悪いとかそもそも要領が悪いとかいろいろ周囲から言われているが，プログラミングパラダイムという観点から研究を進めれば，こうした選択は必然であり必須であると確信している．

プログラミングパラダイムとしてのモバイルエージェントが普及した場合，プログラミング言語のトレンドも，コーディングスタイルも，実装のトレンドも，すべてが一新されるかもしれない．実際には，そのハードルは高く，それは実現しないかもしれないが，そのときのために粛々と技術と知見を積み上げておくことは，１人の研究者としての使命であると思っている．

プログラミングパラダイムとプラットフォーム

プログラミング言語の普及（と，それによるプログラミングパラダイムの普及）がどれほど進むかは，その実行プラットフォームにかかっている．プラットフォームは，一度普及してしまえば，それが最善のものであるかどうかに関わりなく，ある範囲で確実に使われ続ける．プログラミングパラダイムを普及させるには，それが使えるプラットフォームをいかに普及させるかが重要である．

プログラミングパラダイムとしてのモバイルエージェントは，プラットフォームの普及

を促す請求力としては，あまり強くない．移動アプリケーションのようにユーザの目に見えるところに出てくるとは限らないし，アプリケーションの動作速度が速くなるとも限らない．これは，ちょうど２０年くらい前のオブジェクト指向プログラミングの位置づけによく似ている．以前は，モバイルエージェントのキラーアプリケーションとは何か？という問いがよくされたが，プログラミングパラダイムとしてのモバイルエージェントは，プログラミングパラダイムであるからキラーアプリケーションなどは最初から存在しないし，当初はプログラミング言語側の実装技術の都合で実行効率もあまりよくないかもしれない．

オブジェクト指向プログラミングが今のように（ある範囲で）普及したのは，そのプラットフォームとしての GUI を備えた OS の普及による影響が大きいと，私は思っている．当初は，GUI を使いたいためにオブジェクト指向プログラミングをすることを強要された，といっても過言ではないかもしれない．プログラミングパラダイムとしてのモバイルエージェントには，オブジェクト指向プログラミングでいうところの GUI に相当するような牽引力のある機能・プラットフォームが，果たしてあるのか？MiLog の開発を始めた当初，私はそのような牽引力のある機能を見つけられなかった．しかし，間接的な面からになるが，プログラミング言語が持つエラーリカバリーの力の高さを牽引するものとして，Web 上の情報を処理するソフトウェア(Web エージェント)に着目した．MiLog では，Web 上の情報を扱うために様々なライブラリを用意しており，Prolog 言語のエラーリカバリーの強さと併用することで，比較的容易に，Web エージェントを作ることができた．Web エージェントの作りやすさを背景に MiLog というプラットフォームと言語が普及すれば，そこでモバイルエージェントというパラダイムに触れ，気に入って使ってくれるようになるプログラマが出てきてくれるのではないかと考えた．実際に，MiLog を利用してくれる人の話を聞くと，モバイルエージェントという話は，最初の段階ではほとんど出てこないことが多い．（ただ，１つの問題は，最後までモバイルエージェントが話題に出てこない MiLog 利用者がかなりいることである．この点は，状況の変化を静かに見守りたい．）

不十分なプラットフォームが運悪く普及してしまうと，それはユーザにとっても開発者にとっても非常に不幸な事態になる．この観点から，たとえば java 言語などの既存の言語の機能を少しハックしただけの，扱いにくく低機能なモバイルエージェントプラットフォームが間違って普及してしまうと，大惨事になる．モバイルエージェントパラダイムを支えるプラットフォームは，そのパラダイムの良さを十分に発揮できるように設計され，かつ最初は別の目的でひっそりと普及させるのが，１つの最良の手段ではないかと考えている．

プログラミングパラダイムとノウハウの蓄積の必要性

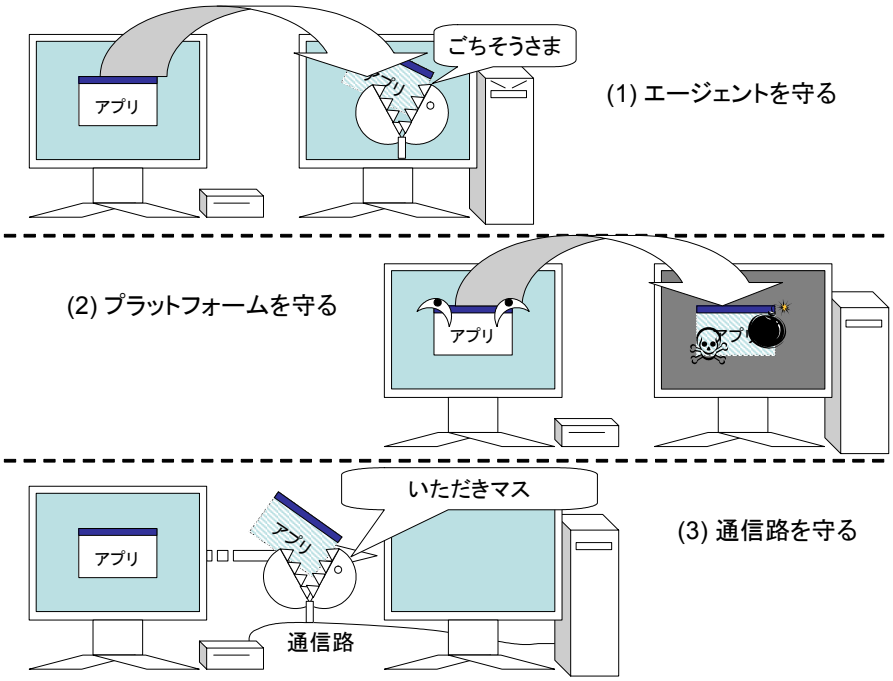
プログラミングパラダイムが普及する仮定では，そのパラダイムをうまく使いこなすためのノウハウの蓄積が不可欠である．たとえば，オブジェクト指向プログラミングなら，継承をどういう場面で使うと効果的か，ポリモーフィズムはどういう場面でのプログラミ

ングを楽にするかといったノウハウの蓄積がないと，その良さをアプリケーション開発でうまく生かせない．

モバイルエージェントというプログラミングパラダイムでは，そのノウハウの蓄積が十分でない段階にあると，私は思う．ところが，モバイルエージェントというパラダイムをプログラマが使おうとするときに，過剰な期待（というより義務感）からか，モバイルエージェントというパラダイムがアプリケーション設計にうまく生かせないことを気にする人が，想像以上に多い．今は，モバイルエージェントプログラミングはまだノウハウを蓄積するべき段階なのだと考え，エージェントの移動を簡易な状態のバックアップ手段として，あるいは簡易なデバッグ手段として使うくらいからスタートするのが，この後のモバイルエージェントプログラミングパラダイムの発展には必要なことではないかと思う．この観点から，最初はごく簡単なスクリプト言語としてモバイルエージェントというパラダイムを使っていくのは悪くないと考えている．

- **セキュリティとモバイルエージェント**

セキュリティ技術の重要性



図：モバイルエージェントに必要なセキュリティ

モバイルエージェントが紹介された当初から，モバイルエージェントは，任意のコンピュータ上で，匿名のソフトウェアとして，匿名のユーザによる実行ができることが，可能性として示唆されていた．ここで必要になる要素技術は，1)悪意あるエージェント（プログ

ラム）からプラットフォームを保護する技術，2)悪意あるプラットフォームからエージェント（プログラム）を保護する技術，および 3)移動途中のエージェントが悪意ある主体によって解読・改ざんされないための技術の３つに大別できる．なお，ここでは，プラットフォームとは，そのエージェントを実行するための計算機とソフトウェアプラットフォーム，さらにその上で動作する他のモバイルエージェントを含めたものをいう．

エージェントが移動中に（移動先でない）ホストを中継しないのなら，3)の移動中のエージェントの保護は，SSL による暗号化通信路を用いることで比較的容易に実現できる．また，1)のプラットフォームの保護は，たとえば Java のサンドボックスのように，プラットフォーム上でエージェントが実行できる機能を制限・監視してやることで，ある程度実現できる．(ただ，より広い意味での保護として，誰のエージェントなら実行させていいのか，どのエージェントにはどこまでの権限を与えるか，といったことは，もっと詳細に OS のレベルで確実に管理できるようになるべきであると思う．)

エージェントが信頼できるものかを調べるには，すでに ActiveX コンポーネントなどに使われている電子署名技術がある程度役立つと思われる．もちろん，動作途中にその形を変えていく（かもしれない）エージェントに対する適切な署名技術は別途研究していく必要があり，すでにいくつかのアプローチが研究として試されつつあるように思う．

残った，2)の，悪意あるプラットフォームからエージェントを守る技術の実現には，ハードウェアの支援が欠かせない状況にあると思う．現在，Intel などの大手の CPU メーカーにより，ソフトウェアを保護するためのハードウェア機構の開発・実装が進んでいる．これらのハードウェア機能の助けを借りれば，いずれ近い将来に実現できると思われる．また，エージェントの行う計算を複数の「意味のない計算」に分割して，それらを統合したときに初めて意味のある結果になるようにするといった新しい方向性のアイデアも提案されてきているようである．

バランスのいいセキュリティ機能の必要性

モバイルエージェントのためのセキュリティ技術については，長期的な展望ではその解決策が見えて来つつある．ただ，長期的な展望だけでは，今の状況，今の OS，今のプラットフォームでどのようにモバイルエージェントのためのセキュリティ技術を実装するのか，という問題の解決にはならない．私は，モバイルエージェントというものの扱い方が十分にわからないうちは，最低限，3)の通信路上のエージェントの保護と，エージェントの出入りの制限のみを実現し，中途半端に高機能なセキュリティ機構は実装を見送った方が現実的ではないかと思う．

初期のモバイルエージェント研究については，セキュリティ機能を欲張りすぎたために，モバイルエージェントの特性を生かせないがんじがらめのシステムが多くできてしまったように思う．

Java のサンドボックス機能があれば，悪意あるエージェントの動作を監視・ブロックす

ることくらい簡単であるような印象を持つかもしれない．しかし，エージェント同士が互いに通信するような場合を考えると，問題はそれほど簡単ではない．エージェント自身が直接的な悪さをしなくても，他のエージェントを経由していろいろな悪さができてしまう可能性がある．逆説的に言えば，モバイルエージェントは移動できるというくらいなのでそこで持ち得る機能は最小限なものになっているはずであり，他のエージェントなどの助けを借りなければ与えられた目標を達成できないことは，十分考えられる．

通信にかかわる権限の管理は，単にその通信元がどの主体かということだけから管理できるほど簡単ではない．たとえば，「エージェント A が，エージェント B に 「エージェント C 頼んで DB から X をとってきてもらって」」という依頼を送るような，入り組んだ通信が行われる可能性がある．このような可能性を考えずに単純なエージェント単位での権限だけを考えたセキュリティ機構を設計してしまうと，一見頑健そうに見えてじつは非常に脆弱なセキュリティ機構ができあがってしまう．

現状では，モバイルエージェントのためのセキュリティ機構に関する基礎研究と，それとは別の，セキュリティ機構を弱めた（ただし，それを利用者がきちんと認識すれば安全に使える）実行プラットフォームの両者を用意しておく必要があると考えている．MiLog の実装では，この指針に従って，実現するべきセキュリティ機構を限定して実装している．

ソフトウェアの保護とソフトウェアに対する詳細な権限の付与・管理については，OS 一般の技術としても非常に有益でかつ必要性の高い研究課題であると考えられる．これらの技術は，モバイルエージェント固有の技術として開発するメリットは薄く，むしろ OS の実装を巻き込んだ技術として研究していくべきであると思う．また，そうなれば，(たとえモバイルエージェントという技術が普及しなくても）今以上にアプリケーションソフトウェアをいろいろな OS 上で柔軟に扱えるようになると思われる．

大事なことは，モバイルエージェントという技術そのものがどれだけ役立つかではなくて，そこで研究・開発された技術が最終的に我々に何をもたらしてくれるかではないかと思う．

(2006.10.24, 30,31　福田 直樹　静岡大学情報学部)

用語説明・補足など

Ajax(Asynchronous JavaScript + XML)

Web ページ上で動的なページ内容の変更を実現する DHTML と JavaScript に，非同期 XML 処理呼び出し機能を組み合わせて，これまでのページ遷移型の Web アプリケーションとな異なる，リッチなユーザインタフェースを実現する技術．WikiPedia(JP)にも，簡

単な解説が載っている．

DRM(Digital Rights Management)

デジタル化された音楽データや画像データなどの電子データについて，その著作権者などの権利を保護するための技術．音楽データや映像データなどがデジタル化されると，データコピー時の品質劣化が無くなり，無限にコピーが出回ることになる．DRM に含まれる暗号化技術等を用いることで，データの頒布や利用を適切に制限することが可能となる．

OS(Operating System)とプロセス(process)

OS とは，Linux, Mac OS X や Windows XP のように，コンピュータを動作させるための基本となる機能（アプリケーションソフトウェアを起動する機能を含む）を提供する基本ソフトウェアである．プロセスとは，OS 上で実行されるアプリケーションソフトウェアの管理の単位であり，Linux などの OS では，ps コマンドや top コマンドを用いて，現在動作中のプロセスの一覧を見ることができる．詳細は，オペレーティングシステムに関する教科書を参照されたい．

バーチャルマシン(Virtual Machine)と仮想化技術(Virtualization Technology)

一般にバーチャルマシンといった場合には，ある機械的な機構をソフトウェアにより仮想的に表現・実装したものを意味する．本文中では，主に，x86 CPU を搭載した PC/AT 互換機のような, 通常利用する PC のアーキテクチャをそのままソフトウェア的に実現したものを意味する．仮想化技術は，それを実現するためのソフトウェア・ハードウェアの実装技術である．なお，MiLog で実現しているバーチャルマシンは，Prolog プログラムを実行コードとする Prolog 言語仮想機械であって, 上のバーチャルマシンとは意味が少し異なる．

TRON(トロン)

日本が誇る，リアルタイムオペレーティングシステム．NHK 社のプロジェクト X でも紹介された．見えないところで，我々の生活を支えている OS である．詳細は，TRON の公式サイト(http://www.sakamura-lab.org/TRON/TRON-j.html) などを参照されたい．

コードモビリティ(Code Mobility)と移動アプリケーション(Migratory Application)

コードモビリティは，プログラミング言語のレベルでそのプログラムの「移動」を扱う能力を意味する．コードモビリティにはいくつかのレベルがあり，ダウンローダブルソフトウェア(Downloadable Software)と, 弱モビリティ(Weak Mobility), 強モビリティ(Strong Mobility)に大きく分割される．コードモビリティは，あくまでもプログラミング言語レベルでの話であり，そこで作られたアプリケーションソフトウェアがどういうものであるのかには興味がない．コードモビリティについては，Fuggetta らの文献が詳しい．

移動アプリケーションは，コードモビリティとは対局にあり，ユーザから見て「複数のコンピュータ上を移動するように見える」アプリケーションである．ここでは，そのアプリケーションがどのようにプログラミングされたかには興味がない．つまり，コードモビリティを一切持たないプログラミング言語・プラットフォーム上に移動アプリケーションが実現されるようなこともあり得る. 移動アプリケーションに関する文献としては、Bharatらの文献が，私の知る限りもっとも初期のもので，おそらく彼らがこの概念を最初に提案したのではないかと思う．

参考文献

A. Fuggetta, G. P. Picco, and G. Vigna, Understanding Code Mobility, IEEE Transactions on Software Engineering, Vol.24, No.5, pp.342—361, May 1998.

K. A. Bharat, and L. Cardelli, Migratory Applications, ACM UIST'95, pp.133—142, 1995.