# ユビキタス時代のソフトウェアノンストップ運用支援

（研究課題名：ユビキタス環境を支えるプログラミング言語システム）

「機能と構成」領域　　橋本　政朋

## 要　　旨

ユビキタス環境とは、いつどこででも特別な訓練や知識なしに様々な情報サービスが受けられる便利な環境です。その実現のためにネットワーク化された小さなコンピュータが至る所に埋め込まれ利用されます。いくら小さいとは言っても、それらの上で稼動するソフトウェアは常に最新に保つに越したことはありませんが、従来はほとんどの場合、アップデート時にそのソフトウェアが提供するサービスを終了させる必要がありました。ユビキタス環境では、センサ値のやりとりなど、サービス途絶がサービスの品質を大きく落としてしまう場合が少なからずあると考えられます。サービス途絶なくソフトウェアを更新する方法は幾つか提案されていますが、適用可能な場合が少ない、管理運用のコストが高くついてしまう、などの問題がありました。そこで、ユビキタス環境においてサービス途絶のないアップデートを支援するためのシステムを新たに考案し、プロトタイプを構築してその効果の確認を行いました。

## 1．研究のねらい

近年、ユビキタスコンピューティング（Ubiquitous Computing）という概念が注目を集めています。ユビキタスコンピューティングとは、1980年代後半に入力デバイスのマウスを実用化したことでも有名なXeroxのPalo Alto研究所で提唱された概念です。これからは生活の至る所に埋め込まれたコンピュータが連携することで、利用者は複雑な操作をせずとも適切な情報サービスを時と場所を選ばず受けられるようになるであろうことが予見されていました。この10年以上も前に生み出された概念は、ここ数年のコンピュータ小型化技術、ネットワーク技術やセンサ技術などの著しい発展により急速に現実味を帯びてきています。例えば、VICSと連係して道路状況に応じて経路探索を行なうカーナビは既に実用化され普及しています。他にもユビキタスコンピューティングの実現普及に向けた研究が内外で活発に続けられています。

このユビキタスコンピューティングを本格的に実現普及してゆく上で鍵となる技術要素には幾つかありますが、ここではソフトウェアの無停止運用技術に注目し、さらに停止要因の中でも主にソフトウェアのアップデートを取り上げます。通常アップデート作業は、対象ソフトウェアの稼働を止めてしまい、最初から実行をやり直すことを余儀なくされます。しかしながらその一方で、バグの修正や機能拡張のためソフトウェアのアップデートは今後も必要でありつづけると考えられます。ネットワークが高密度化し、コンピュータウィルスの感染経路や情報漏洩の経路が益々増えてしまうことにもなるため、セキュリティを高める意味でもアップデートは益々重要となるでしょう。

サービスを終了させてしまうことなくソフトウェアをアップデートする方法は今までにも幾つか提案されてきました。例えば、電話交換システムなどのミッションクリティカルなシステムで利用されるハードウェアの冗長化という手段があります。ところが、このような方法をユビキタス環境中の全てのコンピュータに適用することは現実的ではありません。途方もないコストがかかってしまうからです。ハードウェアに頼らず、ソフトウェアのみで実現する方法も幾つか提案されています。例えば、クライアントサーバモデルにおいて、クライアントとサーバとのセッション（一連のやりとり）が十分短ければ、単純にサーバを再起動すれば間に合います。しかし、ユビキタス環境ではセンサ情報のやりとりなど、セッションが長くなる場合もあります。そのような場合に対処可能な方法は、大別すると、動的ロード・リンクに基づく方法と実行状態移送に基づく方法とに分けられます。

動的ロード・リンクに基づく方法は、機能拡張に向いています。例えばウェブブラウザのプラグイン機構が挙げられます。プラグインをブラウザに動的にロード・リンクすることでそれまで表示できなかった画像形式の表示が可能となります。この場合アップデートの単位はプラグインということになります。ただ、それだけではバグの修正等には使えません。機能を拡張するだけでなく、元々あったプログラムの一部を置き換えてしまうことができるシステムも提案されていますが、置き換えた部分とそれ以外の部分との間で整合性をとるためのプログラムを付け足す場合があり、繰り返しアップデートを行うと、どんどんプログラムが大きくなってしまい、実行速度も低下する可能性があります。

実行状態移送に基づく方法では、プログラムを継ぎ接ぎするのではなく、実行状態を再利用することで無停止アップデートを実現します。プログラムの実行は、コンピュータの中枢であるCPU（中央演算処理装置）がプログラムを読み、メモリ（記憶装置）から必要な情報を読み出したり、様々な計算を行ったり、その結果をメモリに格納したりして進んで行きます。実行状態とは、CPUが処理しているプログラムの場所やメモリの中身などの情

報です。今、あるプログラム、Aがコンピュータ C で実行中であるとします。ここで、新しいプログラムへとアップデートするとします。このためには先ず、プログラムの実行を一時停止させ、実行状態Sを取り出します。通常Sは新しいバージョンの実行にはそのまま使えないため変換しS' を得ます。CにA' とS' とをセットし実行を再開すれば、一時停止こそしますが、サービスとしては無停止なアップデートを実現することができます。この方法ではプログラムが無用に大きくなることはありません。但し、既存の方法では、アップデートのタイミングに制限がありました。

図１　動的移行例

この研究では、サービスの停止を伴わないアップデートを安全に実施するためのシステムを安価に実現することを目指しています。このために実行状態移送に基づく方法を改良した新しい方法を考案します。また、本研究ではさらに、実行状態移送を異なる種類のコンピュータ間でも行えるようにすることで、アプリケーションの移動も同時に実現します。これによりコンピュータの交換も、それが提供するサービスを止めること無く実施することができます。このようなアップデートを動的移行と呼びます。例えば図１では、三目並べの盤面の状態が旧コンピュータから新コンピュータへと移送され、かつバージョンも新しくなっています。これは説明のための例に過ぎませんが、ユビキタス環境で稼動するソフトウェア、特にサーバのノンストップ運用を支援することで、ユビキタスコンピューティングの本格的普及の一助となることが期待されます。

## ２．研究方法と成果

動的移行を実現することは容易いことではありません。プログラムの停止性の判定と同様、完全自動を実現することは不可能だということが分かっています。現実には、アップデートの自由度、安全性、実行効率、コスト（手間、費用）、などがトレードオフの関係に

あり、実際にユビキタスコンピューティングの現場で利用可能なシステムの実現普及させてゆくためには、トレードオフの中で最良のバランスを探らなくてはなりません。本研究では、提案する動的移行法のモデル化を行い、さらに実際の利用に耐えうる方式であることを示すため、モデルの近似としてプロトタイプシステムの実装を行いました。プロトタイプシステムは新旧プログラム間の差分解析システムと実行時システムとから構成されます。また、プロトタイプシステムを実際のアプリケーションプログラムに適用しその効果の確認を行いました。以下それぞれについて説明します。

## 2．1　動的移行のモデル

動的移行のためのシステムは今までにも幾つか提案されていますが、特殊であまり利用されていないプログラミング言語専用であったり、適用できる場合が少なかったり、適用に関する制限を低減させる代償として、プログラマに負担を強いたり、と、実際にユビキタスコンピューティングに適用して広く普及するには様々な問題がありました。特に、動的移行が実現できたとしても、その適用によりプログラムの実行に異常を来す可能性があるものもありました。しかしそれでは意味がありません。このような事態を招くことの無いように、動的移行を曖昧性の無い形式的なモデルとして定義し、実行に異常を来さないことを証明する必要があります。この節では提案するモデルについて説明し、従来法に比してどのような部分を改善したのか、また、どのように安全性を確認するのかについて概要を説明します。

プログラムの実行は、プログラムカウンタ（読んでいるプログラムの部分を指す）やメモリの状態を含んだ実行状態の変化の系列として表されます。

$$P : S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_p \rightarrow \cdots \rightarrow S_i \rightarrow \cdots$$

今、状態が$S_i$の時にプログラムPをアップデートするとします。この時、プログラムの変更がどのようなものであれ、$S_i$以前の実行系列、$S_0 \rightarrow \cdots \rightarrow S_i$ に影響を与えなければ、プログラムを変更しても、実行状態$S_i$から実行を継続しても安全と言えます。プラグインの場合は全く新しい機能を追加するのでこのパタンに相当します。しかし、一般には$S_i$以前の状態系列に何らかの影響を与えてしまう可能性があります。プログラムの変更により影響を受ける最初の実行状態が$S_p$だったとします。この場合は、新しいプログラムP' の実行は$S_p$から開始すれば良いわけですが、$S_p$が$S_i$に近いとは限りません。$S_0$に近いと、再起動をした方がよくなってしまいます。従来法ではそれを避けるため、実行状態$S_i$を直接変換するプロググ

ラムを人手で書くこととしていました。しかし、そのような変換プログラムの安全性を確認することは一般には不可能であることが知られています。ここが従来法の限界でした。提案モデルではこのような場合にも対処するため、$S_p$からの実行系列がプログラム変更前の実行系列中の$S_p$と$S_i$との間のいずれかの実行状態$S_q$と等価な実行状態へ至るならば、$S_i$から実行を継続することとしました。

$$
\begin{array}{llllllllllll}
P: & S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow & S_p & \rightarrow \cdots \rightarrow & S_q & \rightarrow \cdots \rightarrow & S_i \rightarrow \cdots \\
 & & \| & & \| & & \\
 & P': & S_p & \rightarrow \cdots \rightarrow & S_q & & S_i \rightarrow \cdots
\end{array}
$$

$S_p$から実行系列が全く変わってしまう場合は諦める他はありません。また、上の場合ですとP'への動的移行により$S_p$と$S_q$との間でプログラムが実行されることになるため、Pでの実行に加えてその部分が二重に実行されることになります。例えば通信でメッセージを送る処理がそのような部分にあって、二重送信を避けたいと思うなら、利用者があらかじめその扱いを指定しておく必要があります。このようにモデルを設定することで、従来提案されてきた方式と比べて、より多くの場合で安全な動的移行が可能となります。例えばC言語と対象としたあるシステムでは、main関数の変更があると再起動しかありませんでしたが、提案法ではそのような場合でも条件が揃えば動的移行が可能です。

## 2.2 プロトタイプシステム

上述のモデルに基づき、Java言語で記述されたプログラムを対象とする動的移行システムのプロトタイプを構築しました。提案モデルは、飽くまでも設定した安全性の確認のためのものであったため、それ以外の要素に関しては考慮されていません。例えば、現実のCPUの処理速度やメモリ容量や速度などです。従って、実際に動作するシステムを構築するには、モデルをそのまま実装するのではなく近似的に実装する必要があります。主な近似要素は実行状態系列です。モデルではプログラム実行のステップ毎に実行状態を記録しますが、現実問題として特に埋め込み型の小さなコンピュータでそれを行うことはあり得ません。実際には実行系列を間引いて記録することになります。また、実行状態系列が全て記録されていれば、どの実行状態がプログラムの変更により影響をうけるかは、単純に変更箇所をそこから検索するだけで済みますが、間引いた分を補完するため影響を静的に解析する必要が出てきます。この部分に関しては、従来からのフロー解析技術を用います。

プロトタイプシステムは、差分解析システムと実行時システムとから構成されます。このシステムではプログラムは実行時システムの管理下で実行する必要があります。差分解析システムはアップデートにおいてプログラムのどの部分がどのように変更されたのかを解析し、実行時システムが対象プログラムを新しいバージョンへと動的移行させるために必要な情報を抽出します。

### 2.2.1 差分解析システム

文字通り新旧プログラム間の差分を調べ、そこから動的移行に必要な情報を抽出します。この研究では、詳細に差分を調べるために構文木レベルの比較を行ないます。木構造同士の比較は非常に計算量が大きいためプロトタイプでは精度は落ちますが様々な工夫により計算量を押さえています。

差分を調べるためによく使われるのは一次元的な文字列の比較です。例えばUNIXのdiffコマンドでは行単位で変更が表示されます。非常に高速に動作しますが、情報の粒度は荒くなります。プログラムのどのような要素がどのように変わったのかは人手で判断しなくてはなりません。プロトタイプではそうした判断をある程度自動化するため、より細かい粒度で変更を分析します。プログラムを単なる文字列として比較するのではなく、プログラムの構文木の比較を行ないます。構文木とはプログラムを構文解析して得られる二次元的な構造です。構文木を比較することで例えば、「関数ｆのパラメタが定数３から８へと変更された」といったことが自動的に導かれます。但し、木構造同士の比較は計算量が大きく普通に実装するとごく小さなプログラムにしか適用できなくなってしまうため、なるべく精度を落とさずに節を減らす工夫（対応付けが確定した枝の刈り取り、木の枝の折り畳みと展開、複数節の圧縮）を行なっています。

### 2.2.2 実行時システム

アプリケーションプログラムの実行を管理し、動的移行を実現します。今回のJava言語用システムでは、JPDA（Java Platform Debugger Architecture）とBCEL（Byte Code Engineering Library）とを用いて実装されています。アプリケーションプログラムの通常実行時の速度低下はほとんどありませんが、アプリケーションプログラムのクラスファイルのサイズが約２割程度増加します。

実行時システムでは、次に挙げる段階を経て動的移行を実現します。

**実行の一時停止**

**実行状態の取得**

Java言語の場合は、スレッド毎にプログラムカウンタやメソッド呼び出しスタックとそこから指されるオブジェクトの情報を芋蔓式に取得して行き、実行状態の写しを構成します。実行状態の写し自体は通常のデータ構造なので、改変が簡単にできるようになっています。また、新しいプログラムでの実行再開時に得られればよいような情報（例えば現在時刻）やその他不要な情報は明示的にカットすることができます。

**実行状態の調整**

差分解析システムからの情報を基に実行状態を新しいプログラムに会わせて調整します。例えば、文の追加を行った場合は、その文以降でプログラムカウンタの値がずれるため調整の必要があります。

**実行状態の移送**

新しいプログラム用に調整された実行状態の写しは、補助記憶に記録されたりネットワーク等を通じて他のコンピュータに送られたりします。

**実行の再開**

実行状態の写しから実行状態を再構成します。呼び出しスタックの再構成では、特に指定されない限り、プログラム変更により影響を受けるスタックフレームまでは実行状態の写しから単純に内容が転写され、そのフレーム以降から実行を再開します。保存されている実行状態の写しのスタックフレームに等価なものが見つかれば、実行を再び一時停止し再構成を再開します。もしあらかじめ指定された時間内に等価なフレームが見つからない場合は、そのまま実行を継続します。

### 2.2.3　実験

ユビキタスコンピューティングでは、利用者の状況や、環境の状況を把握するために、センサ システムを用います。その中でも近年注目されているのが、センサネットワークと呼ばれるシステムです。各種センサが搭載された非常に小型で低消費電力のコンピュータ（センサノード）が無線でアドホック通信（基地局不要の通信）することでセンサ値情報を交換します。センサノードは消費電力を押さえるため、特殊な無線と特殊なプロトコルを用いて通信します。また、センサ値を用いた処理は、他の処理能力の高いコンピュータで行われるため、センサネットワークの通信と通常のネットワークとの仲介をするセンサゲートウェイと呼ばれるサーバがよく用いられます。今回は、Mica Mote (Xbow社) と呼ばれるセンサネッ

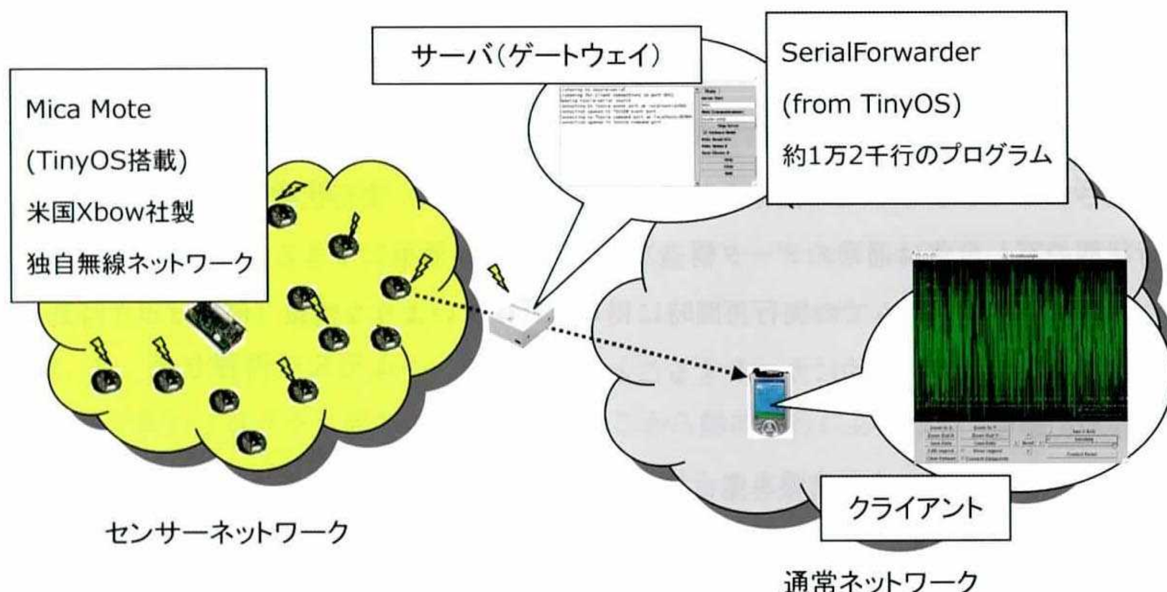


図２　センサネットワークゲートウェイへの適用

トワークシステムのためのOSであるTinyOS（U.C.Berkeley）の配布物に含まれるセンサゲートウェイソフトウェア（SerialForwarder）に対してプロトタイプを適用しました。最近の８つのバージョンに対して更新の分析を行ったところ、実質的に意味のある更新があったのは最後の２つのバージョン間のみでした。更新内容は、通信処理を行うメソッド内でのフラッシング処理の追加と例外処理の追加でした。これらの更新は比較的取り扱いが易しく、全自動で動的移行が可能でした。クラスファイルは約17%のサイズ増加でした。図２では、センサネットワークからのセンサ値がゲートウェイを経由し、単純にセンサ値をグラフ出力するクライアントへ送られる様子が示されています。

## ３．今後の展望

### 適用例の充実

より多くのソフトウェアに対し提案システムを適用し、評価、改良を行っていきたいと考えています。必然的に現在のプロトタイプで足りない部分の実装や、ツールの拡充を行っていくことになります。特に差分解析システムの改良は必須です。今回の実験では変更が比較的小規模だったためあまり問題がありませんでしたが、より大規模で複雑な変更だと現状では人手による修正が必要となってしまいます。

### 他言語への応用

今回のプロトタイプ実装はJava™言語を対象としました。センサネットワークのアプリケーションなどで実際に利用されていることや、実行状態移送が比較的実現しやすいこと等が主な理由です。しかしながら、C言語やC++言語も依然としてよく利用されています。実行効率の高さが主な理由と考えられます。今後の課題として、提案モデルをC言語やC++言語を対象として実装することが考えられます。但し、実行状態移送の実装が難しく、静的解析の精度も落ちるため、適用できないパタンが増えることが予想されます。

## 4. 成果リスト

### 論　文

1. 中島秀之, 橋本政朋. 日常生活のための知的情報基盤. 情報処理学会誌. Vol.43, No.5. 2002.
2. 橋本政朋. 移動計算に基づく実行時更新の実現（仮).（投稿予定）

### 口頭発表

橋本政朋. プログラムの実行時更新を支援するプログラミング言語系の構築に向けて. PPL2002.

### ソフトウェア

SUS-X/Java: A Software Update System for the Java™ Programming Language（仮）（公開予定）