研究課題別評価

1　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

2　研究者氏名：橋本　政朋

3　研究の狙い：

近年、ユビキタスコンピューティング（Ubiquitous Computing）という概念が注目されるようになってきている。ユビキタスコンピューティングとは、1980 年代後半に Xerox の Palo Alto 研究所で提唱された概念である。生活空間の至る所に埋め込まれたコンピュータやセンサなどが連携することで、利用者に煩雑な操作を一切強いることなく、適切な情報サービスを時と場所を選ばず提供することを目的としている。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る。この研究ではユビキタスコンピューティングを本格的に実現・普及させてゆく上で鍵となると考えられる技術のうち、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特にソフトウェアのアップデートを想定する。アップデートはバグの修正や機能拡張などのため今後も必須であり続け、特にセキュリティのためにも益々頻発化すると考えられる。このアップデート作業は通常、対象ソフトウェアの再起動を余儀なくするため、そこかしこに存在する数多のコンピュータが連携して情報サービスを行うようなユビキタス環境では、そのような再起動は無視できないサービス低下を招くと予想される。しかし、一般にプログラムの実行を止めずにアップデートを行うことは不可能であるため、アップデートに対し制約を課すなどの特化を行わなくてはサービス終了なくアップデートを行うことはできない。今までに幾つかの提案はあったが、残念ながら様々な問題がありユビキタスコンピューティング環境への適用には難があった。この研究では、ユビキタス環境への適用を考慮し、サービスの終了を伴わないアップデートを実現することを目指している。ユビキタス環境への適用のために、計算資源に対してスケーラブルで、処理の大部分が自動化可能でかつ、ヘテロ環境対応であることを要件とする。ユビキタス環境で稼動するソフトウェア、特にサーバに対してこのような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を目指している。

4　研究成果：

この研究では、スケーラブルかつ自動化可能かつヘテロ環境対応であることをユビキタス環境への適用要件と考え、それを満たす、サービス終了を伴わないアップデート方式を提案し、そのモデル化を行った。また、その方式が実際の利用に耐えうるものであることを示すため、プロトタイプシステムの実装を行い、実際のソフトウェアに対する適用実験を行った。

4．1　アップデートモデル

実行時にアップデートを行うためのシステムは今までにも幾つか提案されているが、システムが扱える変更に対する制限が厳し過ぎたり、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、アップデート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ったりなど、実際の利用には難があった。特に安全性に関しては問題で、実行時のアップデート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る。以降では提案モデルについて説明し、従来法に対する改善点、安全性の確認法について概要を説明した後、実装について触れる。

プログラムの実行は、プログラムカウンタ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る。

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

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仮定する。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 からの実行の継続は安全と言える。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る。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 だったとする。この場合は新しいプログラム P'の実行は S_p から開始すれば安全と言えるが、S_p が S_i に近いとは限らない。S_0 に近ければ再起動した方がよいであろう。従来法に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ラムをシステムに与えるものもあった。しかし、そのような変換プログラムの作成は一般には自動化不可能である。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た。

$$\begin{array}{l} P: S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_p \rightarrow \cdots \rightarrow S_q \rightarrow \cdots \rightarrow S_i \rightarrow \cdots \\ \qquad\qquad\qquad\qquad\quad \| \qquad\quad \| \\ \qquad\qquad\qquad\quad P': S_p \rightarrow \cdots \rightarrow S_q \qquad\quad S_i \rightarrow \cdots \end{array}$$

S_p から実行系列が全く変わってしまう場合はその時点からの実行を続ける他はないが、自動化は可能である。ここで注意が必要なのは例えば上の場合、P'への移行により S_p と S_q との間でプログラムが部分的に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るということである。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る。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実行時のアップデートが自動で可能となる。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するしかなかったが、提案法ではそのような場合でも条件が揃えば実行時のアップデートが全自動で可能である。

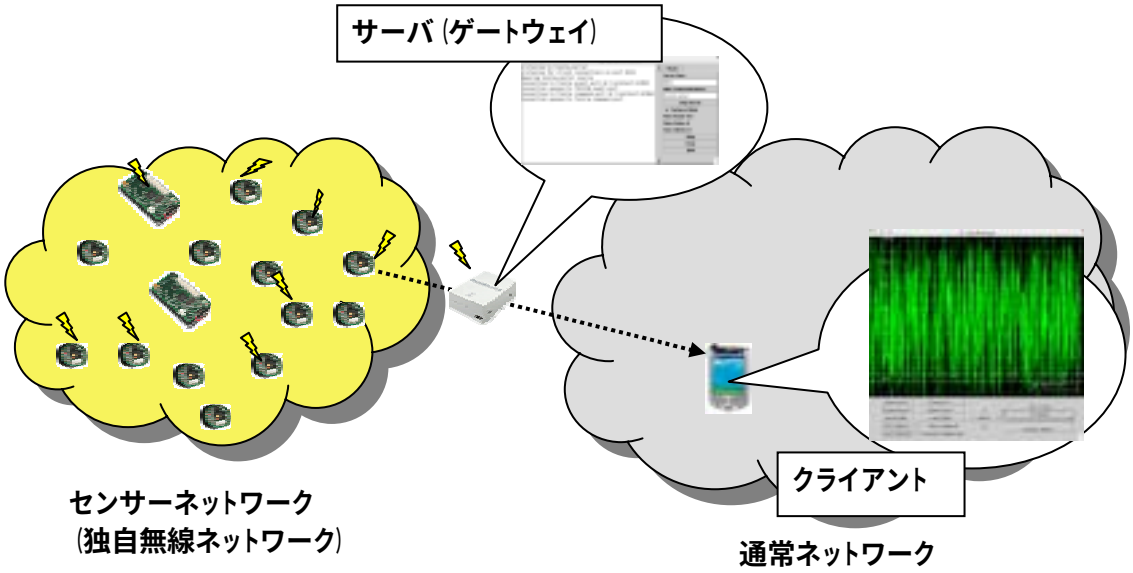
4.2 プロトタイプシステム

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実行時アップデートシステムのプロトタイプを構築した。Java 言語システムを利用することでヘテロ環境に対応している。スケーラビリティに関しては、モデルでは全時点での実行状態を保存し比較できるようになっていたが、実装では保存する実行状態の数や部分的な再実行のステップの上限などをパラメタとして計算資源に応じて与えることになる。また、変更による影響を静的に解析する必要も出てくる。この解析に関しては従来からのフロー解析技法が利用できる。

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る。アプリケーション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る。差分解析システムは文字通り新旧プログラム間の差分を調べ、そこから実行時アップデートに必要な情報を抽出する。プロトタイプシステムでは、詳細に差分を調べるために構文木レベルの比較を行なう。一般には木構造同士の比較は計算量が大きいため、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、部分木の先行解析）を行なっている。実行時システムはアプリケーションプログラムの実行を管理し、実行時アップデートを実現する。今回の Java 言語用システムは、JPDA（Java Platform Debugger Architecture）と BCEL（Byte Code Engineering Library）とを用いて実装されている。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認められない。但し、クラスファイルのサイズが約2～3割程度増加する。

4．3　適用実験

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る。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ある。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することでセンサ値情報を交換する。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する。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ことが多いため、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利用される。



図．センサネットワークゲートウェイへの適用

今回は、Mica Mote（Crossbow 社製）と呼ばれるセンサネッ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：約1万2千行）に対してプロトタイプの適用を行った。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あり、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あった。これらの更新は比較的取り扱いが易しく、全自動で実行時更新が可能であることが確認できた。クラスファイルは約3割のサイズ増加であった。図では、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る。このように通信を行っていても実行時アップデートが可能とするために、通信状態を保存・再構成できる特殊な Socket ライブラリを用いている。

5　自己評価：

本研究では、特にユビキタスコンピューティング環境でよく利用されるコンピュータやプログラムに対して、アーキテクチャやプログラミング言語を問わず、誰でも簡単に使いこなせ、しかも安全な実行時アップデートシステムを実現することを目指した。あらゆる更新に対して全自動で実行時にアップデートを行うことは不可能であるので、動作原理をあらかじめ設定し、その上での安全性を確認するためのモデルを構築する必要があるが、このモデルに関しては極力制約をなくそうと考えたため、研究期間後半に至るまでなかなか仕様が決まらなかった。このため、全般に進行に遅がでてしまったが、結果としては扱える更新パターンは当初の想定を越えるものとすることができ

た。また、実装に関しても、プログラミング言語非依存性までは実現できなかったが、当初考えていたよりも完成度の高い Java 言語用プロトタイプシステムを実現することができた。実験構想初期では、スレッドの処理や通信の処理が複雑となるため、SerialForwarder というプログラムそのものを扱うのは困難と考え、かなり制約を加えた上での適用を想定していたが、研究期間を延長して実装を進めたことで、制約なしに適用することができた。尚、全般（特に論文）に遅れが出てしまったことは反省すべき点と考えている。

6 研究総括の見解：

橋本氏の研究はプログラムの更新による計算機の停止を回避する方式に関するものである．通常の更新では，更新に伴いプログラムは一旦停止し，更新後のプログラムが最初から実行を開始する．プログラムの停止を伴わずに更新を行う試みはこれまでにもあったが，橋本氏の方法は，従来のものに比べてスケーラビリティや自動化に優れ，適用範囲が広いものである．方式の提案とそのためのメカニズム，プロトタイプシステムの構築に関する研究を行った．今後のユビキタス技術に有用な研究であり，研究成果をなるべく早期に取りまとめ公表することを望むものである．

7 主な論文等：

(1) 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．実行状態移送に基づく実行時更新システムの実現（仮）．（投稿予定）

(2) 口頭発表

1. 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．

(3) ソフトウェア

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