ソフトウェアの動的更新におけるデータ変換のための XSL コード作成支援

佐伯 智之* 河口 信夫*‡ 稲垣 康善†

*名古屋大学大学院情報科学研究科
‡名古屋大学情報連携基盤センター
†愛知県立大学情報科学部

概要

近い将来実現するであろうユビキタスコンピューティング環境では，組込み型の情報機器が様々な場所に多数存在する．それらの機器に対して，新機能の追加や不具合の改修のたびにソフトウェアのアップデート作業を行うことは煩雑であり，多くのコストがかかることが予想される．また，アップデートの際には，機器の再起動によるサービス停止の必要がないことが望ましい．ソフトウェアの更新作業が容易で，機器の再起動が不要な，ソフトウェアの動的な更新が可能なシステムが望まれる．ソフトウェアの動的な更新とは，動作中のソフトウェアのコードを，それが動作中に持つ内部変数などのデータを保持したまま，新しいコードへと変更することである．我々は，モバイルエージェントに基づいて動的更新を実現する手法を提案してきた．本稿では，この手法において，データ構造が異なるバージョンのソフトウェア間の動的更新を実行する際に必要となるデータ変換コードの作成支援について述べる．我々が提案する手法において，データ変換コードは XSL で記述され，これを用いた XSLT によって，XML 化されたデータの変換を行う．XSL コード作成の支援に必要となる機能について検討し，その機能を実装したツールを実際に作成し，その有効性を確認した．

1 はじめに

近年，情報機器の小型・軽量化，低価格化によって，携帯端末や情報家電が急速に普及しつつある．それに伴い，小型の計算機が埋め込まれた機器がいたるところに存在する，ユビキタスコンピューティング環境の実現が期待されている．そのような環境においては，多くの情報機器が身の回りのさまざまな場所に存在するため，新しい機能の追加や，不具合の改修のたびにおこなわれるソフトウェアの更新作業に多大なコストが必要となることが予想される．また，そのような環境で動作しているソフトウェアは，更新の際に，サービスの停止を伴うこととなる機器の再起動などの必要がないことが望ましい．具体的には，録画中等にビデオレコーダのソフトウェアをアップデートしたい場合や，通話中にビデオ会議システムのソフトウェアをアップデートしたい場合などが挙げられる．このように，ユビキタス環境では，各機器で動作しているソフトウェアを，それが動作中にもつ内部変数などのデータを保持したまま，機器を再起動することなく更新できるようなシステムが望まれる．動作中のソフトウェアを更新する手法としては，OS にプログラムの状態監視などの新たな機能を実装し，プロセスレベルでの動的更新を実現する手法 [11][12] などが提案されている．しかし，現実的に，メモリなど実行環境に制約がある組込み機器が多く存在するであろうユビキタスコ

ンピューティング環境においては，OS に対する特別な変更の不要な，よりシンプルで，複数の他端末上で動作しているソフトウェアの更新が容易に実現可能な手法が求められる．

我々は，モバイルエージェントに基づいてソフトウェアの動的な更新を実現する手法を提案してきた [1][2]．ここで，動作中のソフトウェアのコードを，それが動作中に持つ内部変数などのデータを保持したまま，新しいコードへと更新することを，ソフトウェアの動的更新と呼ぶ．この動的更新を行う際には，更新前後のソフトウェア間でそのデータ構造が異なる場合に，正しい更新が行えない場合がある，という問題があった．この問題に対して，我々は，コードの更新に合わせてデータの方も変換することによって，データ構造が異なるコードへの更新も可能にする手法も提案した [3]．また，提案した動的更新手法を利用し，ネットワークを介して複数の端末上のソフトウェアを更新する手法と，データ変換を応用して，やりとりするデータの構造が異なるバージョンのソフトウェア間のデータ通信を可能にする手法をそれぞれ提案した [4]．

本稿では，更新前後のソフトウェア間でデータ構造が異なる場合の動的更新の際に，コードの更新に合わせたデータ変換に必要となる，データ変換コードの作成支援について述べる．我々が提案している動的更新の実現手法では，データ変換コードは XSL で記述する．XSL によるデータ変換の処理記述は，XPath 等を多用しており，XSL に慣れない開発者にとって煩雑であることが多い．そこで，データ変換のための XSL 記述の支援について検討し，実際に支援ツールを作成した．

2 動的更新

ここでは，我々が提案してきた，モバイルエージェントに基づくソフトウェアの動的更新手法とその実現方法について簡単に述べる（詳細は [3][4] を参照されたい）．

2.1 更新手法

モバイルエージェントとは，その実行状態を保持したまま，自律的に計算機間をネットワークを介して移動し，移動先の計算機上で処理を継続するソフトウェアである．このモバイルエージェントは，プログラムコードと，実行状態などのデータから構成されているが，移動の際にはこれらを一旦通信可能なデータ形式に変換（シリアライズ）してからネットワーク上に転送され，移動先の端末に到着したときに逆の変換（デシリアライズ）を行い，復元される．

動的更新は，このエージェントの移動を同一端末上で行うことで実現する．すなわち，データ構造の変更を伴わない単純なコードのみの更新であるなら，コードだけを更新したものと入れ替え，データはそのままシリアライズし，同一端末上でデシリアライズすることで，更新されたコードからなるエージェントが更新前の実行状態を保持したまま復元され，動的な更新が実現できる．

しかし，更新前後で異なる構造のデータが求められる場合，上記のようにそのままのデータを新しいコードへと適用すると，正しい更新が行われない場合がある．例えば，新しいコードで，新たな変数が追加されている場合，その変数に対するデータが古いバージョンにはないため，問題となる．このようなデータ構造の変更を伴うような更新も可能とするための手法として，以下の手順をふむ動的更新手法を提案した．

1. エージェントをシリアライズする
2. コードを更新したものと入れ替える
3. シリアライズされたエージェントのデータを，更新先コードに適合するように変換する
4. 変換後のデータをデシリアライズし，エージェントを復元する

このように，コードのみを更新するのではなく，データもコードの更新に合わせて変換することにより，更新前後のソフトウェア間でデータの構造が異なっている場合にも，正しい更新を実現する（図 1）．

2.2 実現方法

提案した動的更新手法を，Java で実装されたモバイルエージェントシステム cogma[5] を利用して実現した．コードの動的更新については，Java のクラスローダを用いて実現している．また，データ変換については，シリアライズ・デシリアライズに JSX[6] を利用し，変換自体に XSLT[7] を利用して実現している．データ変換は以下の手順で行う．まず，動作しているエージェントを構成するオブジェクト（これがデータにあたる）を，

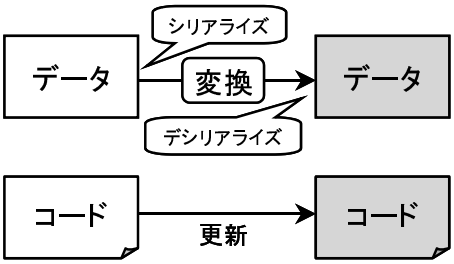

図 1: データ構造の変更に対応した動的更新

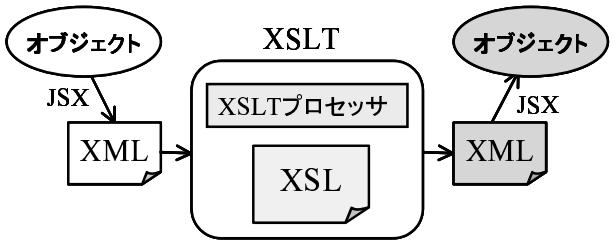

図 2: データ変換

JSX を用いて XML データに変換する．次にこの XML データを，XSLT によって更新先コードのデータ構造に適合するよう変換する．この変換処理は，XSL で記述する（これがデータ変換コードにあたる）．最後に，再び JSX を用いて，XSLT によって変換された XML データをオブジェクトへと戻す（図 2）．本実装では，このようにして動的更新を実現している．

3　データ変換コードの作成支援

前節で，更新前後のソフトウェア間でそのデータ構造が異なっていても，データ変換コードを用いて，更新前のソフトウェアのデータを更新後のソフトウェアのコードに適合するようにデータを変換することによって，ソフトウェアの更新を行う手法について述べた．また，データ変換の実現手法として，ソフトウェアのデータであるオブジェクトを JSX を用いて XML データに変換したものに対して，XSLT を利用してデータ変換を実行することを提案した．このとき，変換の処理を記述する XSL コードがデータ変換コードにあたる．XSL によるデータ変換の処理記述は，XPath 等を多用した大規模で煩雑なものとなることが多い．また，データ変換の処理を記述するには，更新前後のソフトウェアにおいて，どの部分がどのように異なっているかを把握する必要がある．そこで，以下の機能をユーザに提供する，データ変換のための XSL の記述を支援するツールを作成した．

- データ構造の相違部分表示
- テンプレートの自動生成
- 複雑な変換処理の記述支援
- データ変換コードのテスト

以下で，個別の機能について解説する．また，ツールの外観を図 3 に示す．

データ構造の相違部分表示

データ変換コードを作成するためには，更新前後のソフトウェアのデータ構造の違いを理解している必要がある．例えば，新しいバージョンのコードにおいて新たな変数が追加されていたり，ある変数の型が変更されていたり，といった変更によるデータ構造の相違を全て理解していなければならない．しかし，このような変更をすべて正しく把握することは，困難な場合がある．そこで，更新前後のソフトウェア間でデータ構造が異なる部分を示す機能を提供することが望ましい．

本ツールでは，更新前後のソフトウェアのデータであるオブジェクトをシリアライズによって XML データ化し，それら 2 つの XML データ間の差分をとることによって，データ構造が異なる部分（＝データ変換が必要となるであろう部分）をユーザに示す．2 つの XML データ間の差分抽出には，X-Diff[8] というツールを利用している．このツールでは，INSERT，DELETE，UPDATE の 3 種類の処理命令 (PI：Processing Instruction) によって，それぞれ XML の要素，属性の追加，削除，変更を示しており，変数追加，変数削除，変数の値・型の変更の判別にそれぞれ利用できる．X-Diff の出力例を次に示す．まず，差分をとる対象となる 2 つの XML データの一部を以下に示す．

更新前 XML

```
<org.cogma.codget.FigDrawer_-Rect x="60" y="40"
   width="90" height="50" />
```

更新後 XML

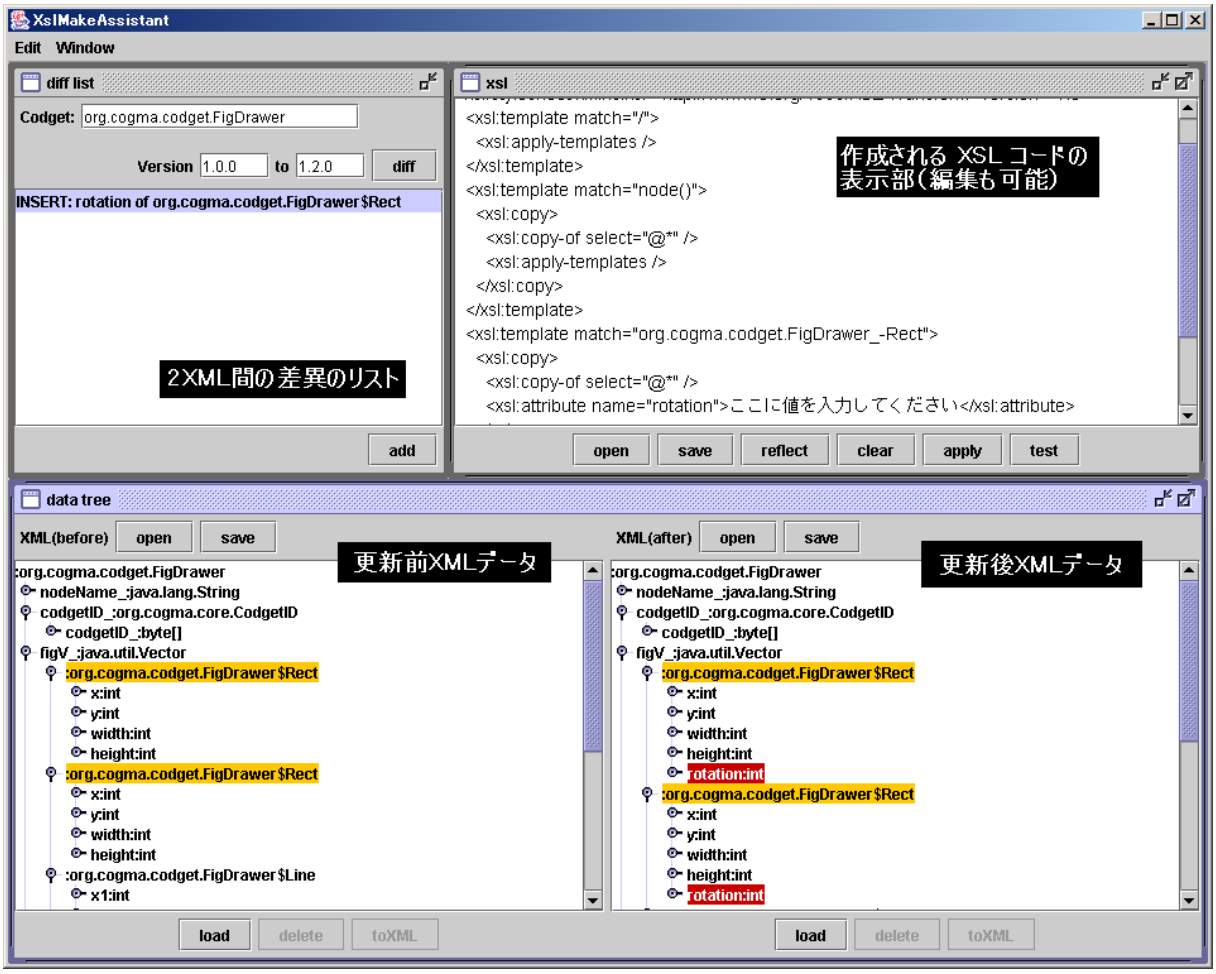

図 3: データ変換コード生成作成ツールの GUI

```
<org.cogma.codget.FigDrawer_-Rect x="60" y="40"
  width="90" height="50" rotation="0"/>
```

これは，あるアプリケーションの更新前後におけるデータの一部を表す．更新前の org.cogma.codget.FigDrawer の内部クラス Rect にはなかった変数 rotation が更新後には追加されていることがわかる．この 2 つの XML データから，X-Diff によって得られた差分を以下に示す．

```
<org.cogma.codget.FigDrawer_-Rect x="60" y="40"
  width="90" height="50" rotation="0">
  <?INSERT rotation?>
</org.cogma.codget.FigDrawer_-Rect>
```

更新による変数追加が検出されていることがわかる．本ツールでは，更新前後のソフトウェアのデータをそれぞれツリーで表しており，上記の XDiff 出力をもとに，2 データ間で異なる部分をそのツリー上に表示する．図 3 の下半分のウィンドウにそのデータツリーが表示されている．

テンプレートの自動生成

データ変換のための XSL 記述は，その記述方法を知らない人にとっては，煩雑で困難な場合もある．しかし，各ソフトウェアの個々の更新に依存する変換処理を表す XSL 記述を，完全に自動生成することは不可能である．そこで，複雑な変換処理の記述はユーザに委ね，データ変換コードのテンプレート（大枠）を自動的に生成することとした．複雑な変換処理の記述が必要である場合には，このテンプレートに対して，必要な記述を追加していく．このテンプレートの生成にも，さきほど述べた X-Diff によって出力される XML 差分を利用する．例えば，変数追加に対する変換記述については，その変数の初期値をどう決めるかの部分はユーザに委ね，他の部分をテンプレートとして生成する．さきほどの変数 rotaion 追加の例に対しては，以下の記述をテンプレートとして生成する．

```
<xsl:template match="org.cogma.codget.FigDrawer_-Rect">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:attribute name="rotation"><!-- ここに値を入力してください --></xsl:attribute>
  </xsl:copy>
</xsl:template>
```

コメント部分はユーザによる値の決定が必要な部分であり，その部分以外をテンプレートとして自動生成する．

複雑な変換処理の記述支援

各更新に依存する複雑な変換処理の記述はユーザに委ねたが，その記述を全てユーザが行うのはやはり困難である．そこで，その記述を支援するために，以下の機能をユーザに提供する．

1 つは，決まったパターンの変換に対する記述の自動生成である．決まったパターンの変換として，例えば，int 型の 2 つの変数から Point クラス (Java API の java.awt.Point) のオブジェクト (x,y という int 型変数をメンバとしてもつ) への変換，配列から Vector(Java API の java.util.Vector) への変換などがある．それら特定のパターンの変換のための記述を自動生成できるよう，ツール側でライブラリを用意しておく．本ツールでは，ユーザが，ツール上のツリーで変換対象である変数と変換先の変数を選択することで，変換のための記述を自動生成することができるようにしている．

もう 1 つは，変換処理の記述について，XSL の形式だけではなく，Java での記述を可能とすることである．XSL の記述方法を詳しく知らない人でも Java による記述ができ，また，XSL では満足に記述できない場合も，それを Java で用意されているライブラリを用いることによって補うことが可能となる．例えば，XSL では三角関数についての関数などは用意されていない．

データ変換コードのテスト

作成した XSL が正しくデータ変換を実行できるかどうかを確かめるために，ツール上で，作成した XSL をデータ変換コードとした動的更新のテストが実行できる．また，実行した更新の前後のソフトウェアのデータを XML 化して保存することができ，それらを用いて，更新前後のデータ間の相違をツリー表示することができる．

次に，実際に本ツールを利用して XSL コードを作成した例を以下で示す．

XSL コード作成例

ここで作成する XSL コードは，部屋のライトを制御するためのアプリケーションの更新に用いるデータ変換コードである．このアプリケーションの更新前バージョンと更新後バージョンの変換対象となるデータについて表 1 に示す．更新前バージョンの変数 frameX と frameY が，更新後バージョンではなくなり，java.awt.Point クラス (int 型変数 x，int 型変数 y をメンバとしてもつ) の変数 frameLocation に変更されている．同様に，更新前バージョンの変数 frameW と frameH は，更新後バージョンでは java.awt.Dimension クラス (int 型変数 width，int 型変数 height をメンバとしてもつ) の変数 frameSize に変更されている．ライトの状態を示す変数 state は，boolean 型配列から int 型配列へと変更されている．これは，更新前バージョンではライトの ON ／ OFF 制御しかできなかったのを，更新後バージョンでは明るさの強弱 (0〜4) を制御できるようにしたためである．

まず初めに，frameX，frameY の frameLocation への変更，すなわち，frameX を frameLocation.x へ，frameY を frameLocation.y へとそれぞれ更新時に割り当てるための XSL コードの作成を行う．本ツールでは，２つの int 型変数から Point クラスの変数への変換コードの自動作成が可能である．図 4 のように，ツリーで表示された更新前後のデータに対して，変換対象の変数 (変換前の変数 frameX，frameY および変換後の変数 frameLocation) を選択し，変換を指定する．すると，変換コードが生成される (図 5，6)．frameW，frameH の frameSize への変換コードも同様にして生成できる．

次に，変数 state の boolean 型配列から int 型配列への変換処理を記述する．ここで，更新前の state の値が false であれば，更新後の state の値は 0 とし，更新前が true であれば，更新後は 4 とする変換を行いたい．そこで，この変換処理を Java で記述し，それを XSL から利用することにする．今度は，データツリーで変換前後の変数 state を選択し，関数を利用することを指定する．表示されるデータ入力ダイアログに変換のために用意した Java のメソッド名などを入力すると，更新前の state の各要素を引数 (boolean) としたメソッドの返り値 (int)

表 1: ライト制御用アプリケーションの内容

更新前バージョン	更新後バージョン	説明
int frameX	Point frameLocation	フレームの位置
int frameY		
int frameW	Dimension frameSize	フレームのサイズ
int frameH		
boolean[] state	int[] state	ライトの状態

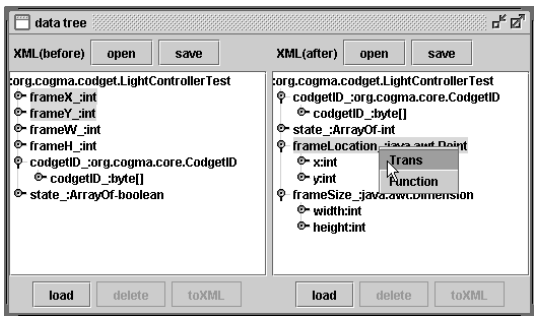

図 4: データツリーウィンドウでの変数選択

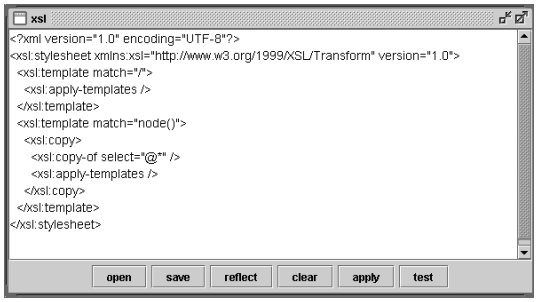

図 5: Point への変換記述追加前の XSL ウィンドウ

を更新後の各要素とする変換コードが生成される．用意したメソッドを以下に示す．

```
public static int lightBooleanToInt(String s){
  Boolean b = Boolean.valueOf(s);
  boolean state = b.booleanValue();
  if(state){
    return 4;
  }else{
    return 0;
  }
}
```

こうして作成した XSL コードを用いて，ライト制御アプリケーションに対する動的更新を行った様子を図 7, 8 に示す．ライトの状態に関するデータはそのままで，

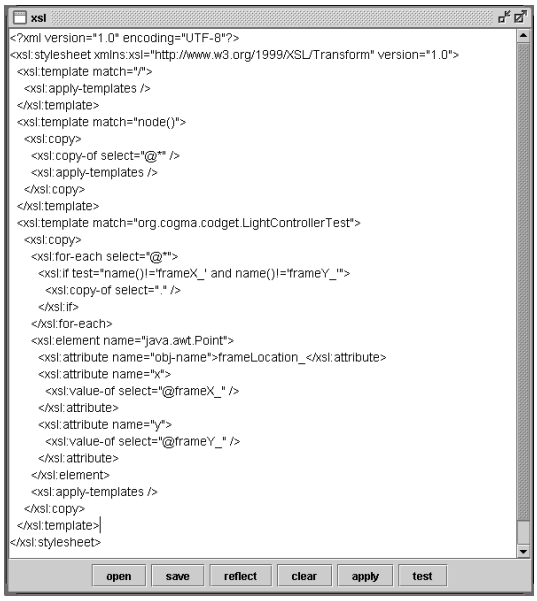

図 6: Point への変換記述追加後の XSL ウィンドウ

明るさの調節機能など新たな機能が追加されていることがわかる．この例で示したように，各種変換のためのライブラリを用意し，XSL での記述が困難な部分は Java で変換用のメソッドを記述することによって，本ツールを利用して XSL コードを容易に作成することが可能となる．

4 関連研究

ソフトウェアの動的更新について，これまでいくつかの研究がなされてきている．

PJama[13][14] では，Java オブジェクトの永続化を行うことができ，Sphere と呼ばれる永続オブジェクトストアを用いてそれを管理し，クラスの更新を行う機能を提供している．このクラスの更新において，PJama では，

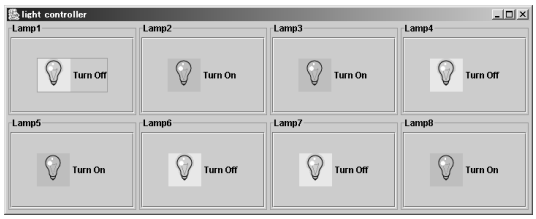

図 7: 更新前のライト制御アプリケーション

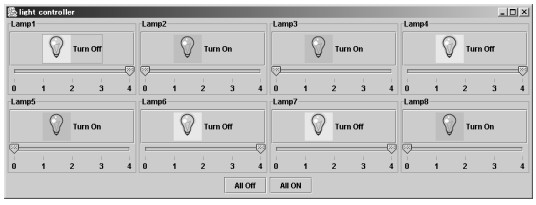

図 8: 更新後のライト制御アプリケーション

更新の正当性を検証し，その変更が相互に整合がとれている場合に更新を実行する．更新において，変数の追加・削除などクラスのフォーマットの変更を伴う場合には，開発者がそのフォーマットの変更に合わせて情報 (クラスのインスタンス) を変換するメソッドを用意 (Java で記述) するが，この変換メソッド記述のための支援等は考えられていない．

Ajmani ら [9] は，異なるバージョンのソフトウェアが各端末で動作するような状況も想定した，分散システムにおけるソフトウェアの動的更新について検討している．この研究では，異バージョンソフトウェアの混在への対応策として，各端末で複数のバージョンを同時にシミュレートする手法を提案している．更新前に，更新先のバージョンをシミュレートするオブジェクト (future-SO) を，更新後に，更新前のバージョンをシミュレートするオブジェクト (past-SO) を保持することによって，異なるバージョンのソフトウェアが混在する環境を可能にすることを考えている．旧バージョンから新バージョンへの状態 (データ) の変換は，Transform Functions(TFs) と呼ばれる手続きによって，更新先のバージョンをシミュレートしているオブジェクトを利用して行う．しかし，システムの実装はおこなわれておらず，この TFs の生成支援についても考えられていない．

Hicks ら [10] は，型安全で C に似た言語 Popcorn で書かれたプログラムを更新するためのシステムについて述べている．このシステムでは，更新されたコードおよび古いバージョンから新しいバージョンへの移行に必要となるコードを含む dynamic patches を用いて，ユーザの好きなときに，プログラムのコード，データ，タイプの更新を可能にしている．この，移行に必要となるコードのスケルトンは，更新前後のコードからデータと関数の変更を特定し，適切な state transformation と stub function を生成することで，ほとんど自動的に作成されるが，やはりプログラマが実際に書かなければならない部分もある．また，今まさに動作しているコードの更新を可能とするために，更新のポイントをユーザがアプリケーション自体にハードコーディングする必要がある．

谷口ら [11] は，プロセスとして走行しているプログラムの一部分を変更する方法として，入れ替えの内容の自由度に制限を加えることにより，サービスプログラムが入れ替えの契機を意識する必要がなく，入れ替え対象のプログラム部分が入れ替え対象でない別のプログラム部分を呼び出している場合にもプログラムの入れ替えが可能である，という特徴をもつ方法を提案している．この手法では，プログラムの実行状態を監視するために専用のシステムコール等を用意した OS 上でプログラムの動的更新を実現している．メモリなどの実行環境に制約がある組込み機器が多く存在するであろうユビキタスコンピューティング環境では，動的更新を実現するための機能を追加した特別な OS を必要とせず，複数の他端末上で動作しているソフトウェアの更新が容易に実現可能な手法が求められる．

5 おわりに

本稿では，我々が提案した動的更新手法について簡単に述べるとともに，その動的更新の際に，ソフトウェアの動作時の状態であるデータを変換するときに用いるデータ変換コードの作成支援について述べた．我々の動的更新実現手法においては，データ変換コードは XSL で記述し，更新前後のソフトウェアのデータを表す 2 つの XML データ間の差分をとるツールを利用することにより，変換記述の生成を支援できる．我々は，変換コード作成支援の機能を提供するツールを実際に作成し，その有効性を確認した．今回，変換コードの作成支援ツールを作成したことにより，我々が提案する動的更新手法において，更新の実行から準備段階の支援までが可能となった．

今後の課題としては，他のソフトウェアと連携して動

作しているソフトウェアを動的に更新する手法の検討や，更新を行う権限などセキュリティ面での検討などが挙げられる．

参考文献

[1] 河口信夫，外山勝彦，稲垣康善：モバイルエージェントに基づく動的拡張可能なソフトウェアシステム，情報処理学会夏のプログラミング・シンポジウム，pp.71-78 (1999).

[2] 植田智，河口信夫，稲垣康善：モバイルエージェントシステムにおけるオンデマンドコード移送とバージョン管理，情報処理学会第６４回全国大会，pp.509-510 (2002).

[3] 佐伯智之，河口信夫，稲垣康善：データ構造の変更に対応したソフトウェアの動的更新手法，第６回プログラミングおよび応用のシステムに関するワークショップ (SPA2003)，(2003)

[4] 佐伯智之，河口信夫，稲垣康善：ユビキタス環境におけるモバイルエージェントを用いたソフトウェアの動的更新手法，マルチメディア，分散，協調とモバイル (DICOMO2003) シンポジウム，(2003)

[5] 河口信夫，稲垣康善： cogma:動的ネットワーク環境における組み込み機器間の連携用ミドルウェア，情報処理学会コンピュータシステム・シンポジウム，pp.1-8 (2001).

[6] JSX(Java Serialization to XML)：(http://www.csse.monash.edu.au/~bren/JSX/)

[7] XSLT(XSL Transformations) Version1.0：(http://www.w3.org/TR/xslt)

[8] X-Diff - Detecting Changes in XML Documents：(http://www.cs.wisc.edu/ yuanwang/xdiff.html)

[9] Sameer Ajmani, Barbara Liskov, Liuba Shrira：Scheduling and Simulation:How to Upgrade Distributed Systems，USENIX Workshop on Hot Topics in Operating Systems (HotOS IX)，(2003)

[10] Michael Hicks, Jonathan T. Moore, Scott Nettles：Dynamic Software Updating，In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (2001)

[11] 谷口秀夫，伊藤健一，牛島和夫：プロセス走行時におけるプログラムの部分入れ替え法，電子情報通信学会論文誌，Vol.J78-D-I，No.5，pp.492-499，(1995)

[12] 小林良岳，佐藤友隆，唐野雅樹，結城理憲，前川守：彩：コンパイル時に自動生成される portal をもとに動的再構成可能なオペレーティングシステム，電子情報通信学会論文誌，Vol.J84-D-I，No.6，pp.605-616，(2001)

[13] M.P.Atkinson, M.Dmitriev, C.Hamilton, T.Printezis：Scalable and Recoverable Implementation of Object Evolution for the PJama1 Platform，Persistent Object Systems(POS)，(2000)

[14] The PJama Project：(http://www.dcs.gla.ac.uk/pjava/)