構文木に着目して XML マークアップされたソースプログラム間の差分抽出

高橋 透　　大久保 弘崇　　粕谷 英人　　山本 晋一郎

愛知県立大学大学院 情報科学研究科

要 旨

XSDML は，CASE ツールで利用される情報がマークアップされたソースプログラムの XML 表現である．本稿では，木に対する差分抽出アルゴリズムを XSDML に適応させた XSDML Diff を提案する． XSDML Diff を用いることでプログラムのバージョン間の木構造を意識した差分を CASE ツールで扱えるようになる．XSDML Diff の特徴は，一般に大きな計算量を必要とする木の差分抽出アルゴリズムに対して実用的な時間で動作すること，差分を構成する編集操作がプログラムに対する編集の操作に対応していること，そして CASE ツール応用に適した差分表現の出力形式を持つことである．

Change Detection for CASE tool of Markup Language for Source Program

Tooru TAKAHASHI　　Hirotaka OHKUBO　　Hideto KASUYA
Shinichiro YAMAMOTO

Graduate School of Information Science and Technology, Aichi Prefectural University

Abstract

This paper presents **XSDML Diff** that is an algorithm for detecting changes for XSDML, which is XML representation of source program for CASE tool platform. XSDML Diff compares two files with concentrating on their tree structures. Features of XSDML Diff are : (1) it works in practical time, while usual tree diff algorithms require large amount of time. (2) tree edit operations are chosen to match source program edit operations. (3) its output format suits CASE tool applications.

1　はじめに

近年 Web 分野をはじめとして様々な局面で XML が利用されている．ソフトウェア開発においてもソフトウェアプロダクトの一部やツールの設定ファイルなどに XML を利用する機会が増えている．XML の一面はテキスト文書へのマークアップによる情報の追加であり，DocBook はこの性質を利用したドキュメント形式である．XML のもう一面は可変長の構造を持ったデータのための汎用のファイル形式であり，XMI はこの性質を利用した UML 図の交換形式である．また，ソースプログラムも CASE では XML によって表現される．多くのプログラミング言語の文法は文脈自由文法により定義され，現在では自由プログラム形式の言語が主流である．このため

文法に基づく構文木でソースプログラムを扱うことが CASE ツールの第一歩であり基盤である．よってソースプログラムを XML で扱うことは自然であると考えられる．Sapid[1] では情報を付加したソースプログラムの交換形式として XSDML を定義している．

一方，ソフトウェアプロセスに必須の技術である差分抽出について考える．差分抽出はデバッグ作業による変化の取得やプログラムの理解支援で必要となる．現在実用化されている差分抽出器は一次元の行指向のアルゴリズムに基づいている．このため自由プログラム形式の木構造によって構成されるソースプログラムの差分抽出に適用することは問題が多い．さらに，情報を付加した XSDML ファイルのような高度な意味情報を含むデータ間の差分抽出を考えるとき，木構造指向の差分抽出アルゴリズムが必要である．

本論文では，ソースプログラムの XML 表現である XSDML に注目し，その間の差分抽出器を提案，実装し CASE ツールに応用する．本論文の目的は以下の 3 つである．

- 行指向差分については，多くの差分表現が提案され実際に開発の場で用いられている．一方，木構造に関しては十分な評価を得ている差分表現がない．本論文では CASE ツールに適した差分表現を提案する．
- 行指向差分と比べ構造の複雑な木構造間の差分では，抽出のために必要な計算コストが大きい．そのため実用的な時間，空間計算量で動作するアルゴリズムを選択する．
- 差分抽出システムを実際に CASE ツール・プラットフォームに応用し，現実的な問題に適用する．

本論文の構成は，2 章で差分抽出アルゴリズムの一般形について述べ，抽出過程で用いられる編集マッピング，エディットスクリプト，差分表現について述べる．3 章では，本研究で扱う XSDML の特徴について述べ差分抽出に必要な要素を明らかにする．4 章では，Chawathe らのアルゴリズム [2] を応用した XSDML 間の差分抽出アルゴリズムを提案する．5 章では，差分抽出システムを CASE ツール・プラットフォームに応用する．

2 差分抽出

差分抽出アルゴリズムの一般的な流れを図 1 に示す．比較対象 T_0, T_1 が与えられたとき，それらの構成要素間の対応関係である**編集マッピング**を求める．次に編集マッピングに基づいて T_0 を T_1 に変換する**編集操作**の列である**エディットスクリプト**を求める．最後に，エディットスクリプトを再利用性の高い形式で表現する．

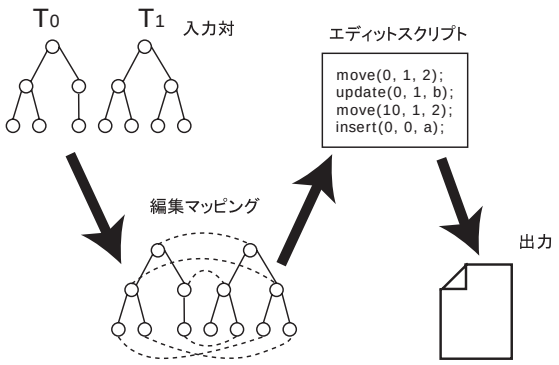

図 1: 一般的な差分抽出の流れ

編集マッピング作成ではエディットスクリプト生成のコストを減らすために，対応の探索範囲に制限を設けることで枝刈りを行う．さらに，編集マッピングを一意に定めるために何らかの尺度により編集マッピングのスコアを与えることで最適な編集マッピングを決定する．

編集操作とは対象の変換の最小単位のことである．対象は扱う構造によって異なり，行指向差分なら一行，木構造差分ならノードや部分木が相当する．構造によって編集操作は異なるが，編集操作の基本的な考えは人間がエディタで編集するときに直観的に扱う編集操作に由来する．よく用いられる基本的な編集操作として挿入と削除があり，それ以外にどのような操作を加えるかによってアルゴリズムの特色が現れる．

差分の表現形式は対象の構造に依存して設計される．行指向差分には多くの差分表現が提案され，人間が読むことを考慮した形式も含め，実際にソフトウェア開発の場で用いられている．一方，木構造差分における差分表現で十分な評価を得ているものはない．また，論文では編集マッピングとエディットスクリプトの生成までが主な感心であり，その差分表現について言及されて

いるものは少ない．

既存の XML 差分抽出ツールはエディットスクリプトを XML や XMLDB の操作言語または独自形式によって表現している．このように単純にエディットスクリプトを差分表現としてそのまま利用すると差分の理解が困難になる．これは，編集操作を適用する度に，対象とする木の構造が変化することに起因する．ある操作は，それに先行する操作によって変化した木を対象としているため，エディットスクリプトともとの文書から各操作の実際の編集対象を理解するのが困難になる．この問題を解決するために，差分表現を 4.3 節で提案する．

3 XSDML

XSDML は CASE ツールの利用する情報がマークアップされたソースプログラムの XML 表現である．CASE ツール・プラットフォーム Sapid のリポジトリとして利用されている．その特徴は主に 2 つある．1 つは元のソースプログラムをタグ付けするフォーマットであるため，タグを全て外すことによってソースプログラムに戻すことができること，もう 1 つは**混合内容**を持たないことである．混合内容とは XML 表現において要素の直接の子要素に要素と文字列が混在していることをいう．

XSDML はプログラミング言語の構文定義に含まれる終端記号と非終端記号に基づいた規則に従って XML マークアップを行う．小文字から始まる終端記号によって文字列を，大文字から始まる非終端記号によって終端記号と非終端記号を囲んでいる．図 2 に Java プログラムの XSDML 断片を示す．これは局所変数宣言 “`int␣x`” を表している．見やすさのためにインデントを付加し，一部属性値を省略している．

```
1: <Local id="s0" typefirst="s1">
2:   <Type id="s1" sort="primitive">
3:     <kw>int</kw>
4:   </Type>
5:   <sp> </sp>
6:   <ident defid="s0">x</ident>
7: </Local>
```

図 2: XSDML の例

4 XSDML 間の差分抽出アルゴリズム

XSDML 間の差分抽出手続きは 3 つのステップで構成される．最初に，木構造差分抽出アルゴリズムを XSDML に適用するために木構造に変換する．次に，木構造に対する差分抽出アルゴリズムにより，エディットスクリプトを求める．最後に，XSDML の差分表現に適した方法で差分表現を行う．第 2 ステップの差分抽出アルゴリズムは，Chawathe らのアルゴリズム [2] を応用した．

4.1 木構造への変換

まず，XSDML を順序付き，ラベル付き木に変換する．XSDML は混合要素を持たないため，要素を木のノードに，要素名とその要素によって囲まれるテキストノードの組をラベルとする．また，XML の属性については編集マッピング作成のための指標に使わない．その理由を以下に挙げる．

識別子の対応　XSDML では，ID/IDREF 型の属性値を識別子の対応関係を表現するために用いる．この属性値は機械的に割りあてられる値であり，別バージョンの XSDML を比較する際の対応文字列として意味をなさない．そのため無視できる．

CASE ツール用の補助情報　CASE ツール開発のために，XSDML の要素関係から得られる情報を属性に持たせている．

型，文，式の区分　型は基本型，オブジェクト型，配列型であるかどうか，文は `if`, `while` などのどれであるか，式は四則演算や代入などのどれであるかを属性に持たせてある．例えば，図 2 の 2 行目の属性値`"primitive"`は，型の区分を表すものであるが，これは 3 行目が `kw` であることから得られる情報である．このように要素間の関係から復元可能である．そのため無視できる．

修飾子の有無　Java のフィールドの場合，`static` 修飾であるか，アクセス修飾子はどれであるか

などを表す属性が付加されている．型，文，式の区分と同様に要素間の関係から復元できる．

元ソースプログラム上での位置　元のソースプログラム上での行や列の位置を保持している．ID/IDREF 型の属性値と同様に別バージョンでは比較する際に文字列として意味をなさない．

これらの理由から，構文木で差分をとるという目的において，比較に関与させるべき情報は XSDML の属性には存在しない．以上のように編集マッピング生成には，属性を用いることなくタグ名とテキストノードだけで対応づけを行う．そのため属性の異なるノードが対応づけられた場合，エディットスクリプトにはその異なる属性を修正するような編集操作の列を別に追加する必要がある．図 3 に図 2 の XSDML に対応する木構造表現を示す．

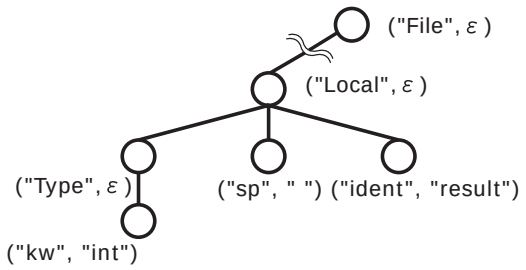

図 3: XSDML の木構造表現の例

4.2　差分抽出アルゴリズム

既存のアルゴリズムの特徴を表 1 に示す．まず，一行目に時間的効率を示す．n は木のノード数である．実用的な時間で動作させるため，木構造の差分アルゴリズムでは比較的速い $O(n^2)$ のアルゴリズムを用いる．次に編集操作の種類は，ソースプログラムの編集時の動作と対応する人間のメンタルモデルに合った編集操作であることが望ましい．論文 [2, 3] では，部分木の移動操作を定義している．これは，プログラムの編集において頻出する，ある構造を持った部分を他に移動させる操作に相当すると考えられる．文献 [3] では，これに加えて部分木の複製操作を用いている．

以上より，時間的効率と編集操作の種類より文献 [2] で提案されている LaDiff アルゴリズムを用いる．LaDiff の対象はラベル付きの順序木である．ただし，ラベル以外に Value と呼ばれるもう 1 つ別のラベルを持つ．4.1 節の変換でのテキストノードの内容がこの Value に対応する．の組で持つことによりラベル木の拡張であると見なすことができる．

LaDiff が用いる編集操作は，葉ノードの挿入，葉ノードの削除，ラベルの更新および部分木の移動の 4 種類である．挿入と削除は葉ノードのみを操作対象としている．ラベルの更新は，Value のみが対象である．

時間計算量が小さい理由は編集マッピングを求める際に全探索を行わないことによる．具体的には，ラベルと Value のうち，ラベルが一致するノード同士でのみ対応を調べることによって探索空間を狭くしている．また，編集マッピング作成に最長共通部分列アルゴリズムを用いることで近似している．そのため最適な編集マッピングを得ることが保証されない．しかし，XSDML Diff はバージョンの異なるソースプログラム間の差分抽出を前提としていることから，この近似が十分な結果を与えると予想した．

4.3　差分表現

本論文では，XSDML 間の差分抽出に特化した差分表現手法を提案する．2 章で述べたようにエディットスクリプトによる差分表現は編集操作の対象が明確ではなくなるという問題がある．そこで編集対象に編集操作の種類や操作に必要なパラメータを付加する方法を考える．すなわち，X を Y に変換するエディットスクリプト E を X に埋め込んで表現する．編集操作の順序関係が失われないように X に埋め込むために，Chawathe らの提案する 4 つの編集操作の性質を検討する．4 つの編集操作の組み合せによって生じる依存関係は以下の 6 つである:

1. ノード m を挿入した後に，ノード m の子としてノード n を挿入
2. 葉ノード v を削除した後に，葉ノードとなったノード u を削除
3. 部分木を移動後，移動先にノードを挿入
4. 部分木を移動後，移動先の部分木の葉ノードを削除

表 1: 木構造差分抽出アルゴリズムの比較

文献		LaDiff[2]	MMDiff[4]	MHDiff[3]	KUO79[5]	ZSHA[6]
時間計算量		$O(n^2)$	$O(n^2)$	$O(n^3)$	$O(n^4)$	$O(n^2)$
編集操作	挿入，削除	葉	葉	内部	内部	内部
	移動	有	無	有	無	無
備考		-	空間効率	複製操作	-	非順序木

5. 部分木を移動後，移動先の部分木のノードを更新
6. 部分木を移動後，移動先の部分木の中の部分木をさらに移動

他にも組合せがあるが，それらについては順序を変えてもエディットスクリプトによって得られる木に変化はない．例えば，更新操作については，2 つのノードにそれぞれ更新操作を適用する場合，その適用順序が入れ替わったとしても適用後の木に影響はない．

木に埋め込んだ編集操作の実行が正しい結果をもたらすような埋め込み方を上記の 6 つの場合を中心に検討する．図 4〜7 の上段は 2 つの編集操作を木に適用した際の変化を示している．下段は木に埋め込んだ編集操作の実行の過程を示す．編集操作のノードや部分木は点線で示してあり，これを無視すると上段と同じ木である点に注意してほしい．

図 4 は依存関係 1 を表したものである．編集操作ノード `ins` はそこにノードが挿入されることを意味する．埋め込まれた編集操作を実行するとき，挿入操作は葉ノードのみを操作対象としているため，正しい順序で実行される．

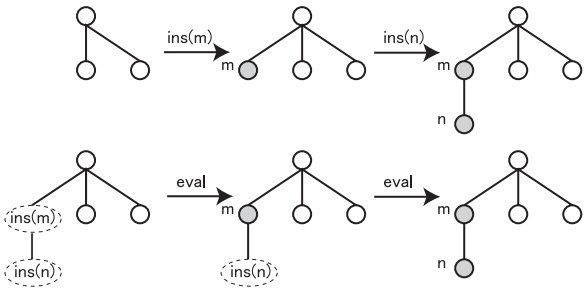

図 4: 挿入 (依存関係 1)

図 5 は依存関係 2 を表している．編集操作ノード `del` はその親ノード削除対象であることを指している．削除操作も同様に，葉ノードのみを対象としているため正しい順序で実行される．

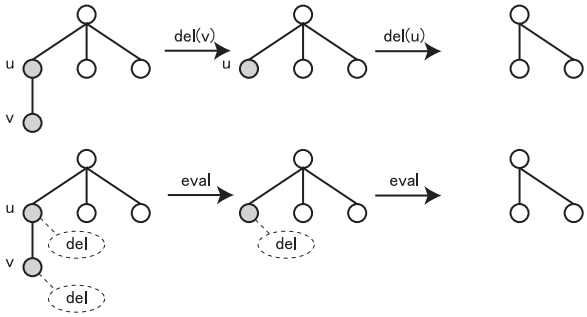

図 5: 削除 (依存関係 2)

図 6 は依存関係 6 の「部分木 1 を移動後，移動先の部分木 2 の中の部分木をさらに移動」を表している．編集操作ノード `from` はその親から先の部分木が，対応する操作木 `to` の位置に移動することを指している．`move(1)` によって部分木 2 を含む部分木 1 が移動し，次に `move(2)` によって部分木 2 が移動する．依存関係 3, 4, 5 についても同様である．

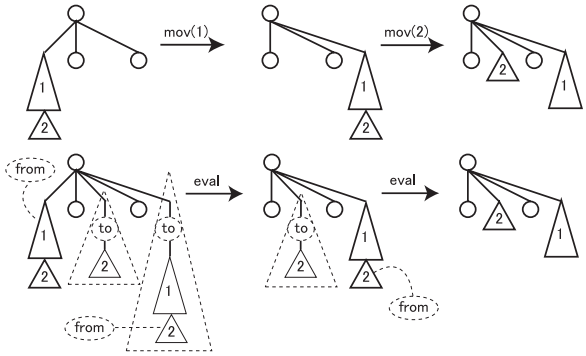

図 6: 移動 (依存関係 6)

図 7 に示すノードの更新操作は，更新対象のノードの子として編集操作ノード upd が付加してある．更新操作は順序関係に依存しないため，どの順序で適用しても同じ結果が得られる．そのため，どこが更新対象かが明確であればよい．

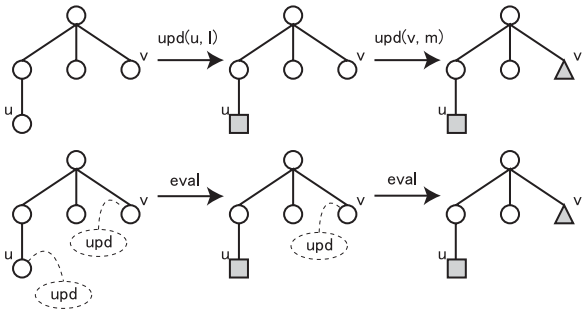

図 7: 更新

このような木を XML 表現として実現するために，XML の**名前空間**を用いる．名前空間によって，同一文書内に異なるマークアップ言語を混在させることができる．この性質を利用し XSDML に差分情報をマークアップによって付加する．この方法の利点として，名前空間に対応している CASE ツールは，差分情報が付加された XSDML に対しても正しく動作する．以下にそれぞれのエディットスクリプトを埋め込んだ木についての XML 表現を示す．

図 8 に示すのは，図 2 の XML に依存関係 1 を埋め込んだ XML 表現である．2–10 行目で Local 要素の 1 番目の子要素として <Type ...></Type> を挿入し，その後 Type 要素の 1 番目の子要素として<kw>int</kw> を挿入することを表している．

```
01: <Local id="s0" typefirst="s1">
02:   <xd:insert>
03:     <xd:element name="Type">
04:       <xd:attributes>
05:         <xd:attribute name="id" value="s1"/>
06:         <xd:attribute name="sort" value="primitive"/>
07:       </xd:attributes>
08:       <xd:element name="kw" value="int"/>
09:     </xd:element>
10:   </xd:insert>
11:   <sp> </sp>
12:   <ident defid="s0">x</ident>
13: </Local>
```

図 8: 挿入操作の XML 表現

図 9 に示すのは，依存関係 2 の編集操作を埋めこんだ XML 表現である．3,4 行目の <xd:delete/> によって，その親要素が葉ノードであるときに削除されることを意味する．この場合，<kw>int</kw> が削除され，その後葉ノードとなった <Type ...></Type> が削除されることを意味する．

```
1: <Local id="s0" typefirst="s1">
2:   <Type id="s1" sort="primitive">
3:     <xd:delete/>
4:     <kw><xd:delete/>int</kw>
5:   </Type>
6:   <sp> </sp>
7:   <ident defid="s0">x</ident>
8: </Local>
```

図 9: 削除操作の XML 表現

図 10 に示すのは，依存関係 6 の編集操作を埋め込んだ XML 表現である．3 行目の <xd:moved-from id="m0"/> と 11–21 行目の <xd:moved-to id="m0">...</xd:moved-to> によって，2–5 行目の<Type ...>...</Type> が 10 行目の直後に移動することを意味している．また，18 行目と 7 行目によって，移動した部分木の中の部分木を対象として再び移動操作を適用し，<kw>int</kw> が 6 行目以降に移動したことを示している．

```
01: <Local id="s0" typefirst="s1">
02:   <Type id="s1" sort="primitive">
03:     <xd:moved-from id="m0"/>
04:     <kw>int</kw>
05:   </Type>
06:   <sp> </sp>
07:   <xd:moved-to id="m1">
08:     <xd:element name="kw" value="int"/>
09:   </xd:moved-to>
10:   <ident defid="s0">x</ident>
11:   <xd:moved-to id="m0">
12:     <xd:element name="Type">
13:       <xd:attributes>
14:         <xd:attribute name="id" value="s1"/>
15:         <xd:attribute name="sort" value="primitive"/>
16:       </xd:attributes>
17:       <xd:element name="kw" value="int">
18:         <xd:moved-from id="m1"/>
19:       </xd:element>
20:     </xd:element>
21:   </xd:moved-to>
22: </Local>
```

図 10: 移動操作の XML 表現

最後に，図 11 に示すのは 2 つの更新操作を埋め込んだ XML 表現である．3 行目の <xd:update value="float"/> によって，<kw>int</kw> が <kw>float</kw> に更新されること意味する．同様に，6 行目も <ident>x</ident> が <ident>y</ident> に更新されることを意味する．

```
1: <Local id="s0" typefirst="s1">
2:   <Type id="s1" sort="primitive">
3:     <kw><xd:update value="float"/>int</kw>
4:   </Type>
5:   <sp> </sp>
6:   <ident defid="s0"><xd:update value="y"/>x</ident>
7: </Local>
```

図 11: 更新操作の XML 表現

5 差分抽出の CASE ツール・プラットフォームへの応用

4 節で提案したアルゴリズムを CASE ツール・プラットフォーム Sapid に組み込んだ．このシステムでは，ANSI C と Java のソースプログラムそれぞれの XSDML 表現である CX-model と JX-model を扱う．システムの概要を図 12 に示す．このシステムにより CASE ツール開発者はソースプログラムのバージョンによる違いを XSDML Diff を通して扱うことができるようになる．システムの主な実装には Java を用い，約 5,000 行のプログラムで実装されている．

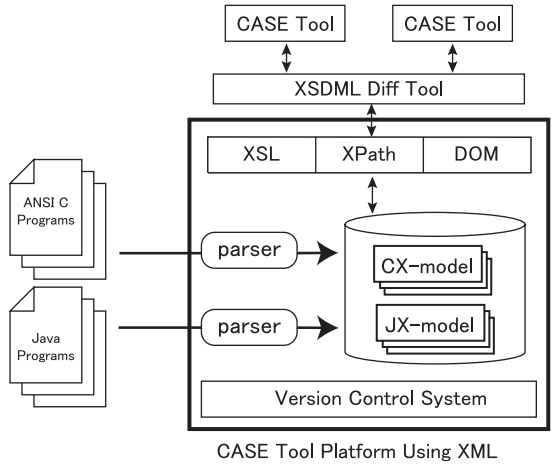

図 12: システム概要

5.1 CASE ツールへの応用

この差分抽出システムを既存の CASE ツールに応用する場合の用途について述べる．

まず，前節で提案した名前空間を用いた差分表現の有用性を確かめるために差分表示ツールを作成した．図 13 のように，左右に旧バージョンのファイルと新バージョンのファイルを並べ，対応する行によって位置をそろえて表示する．その上で構文要素単位での挿入，削除，移動を重ねあわせて表示する．実装は XSLT によって 200 行で記述されている．

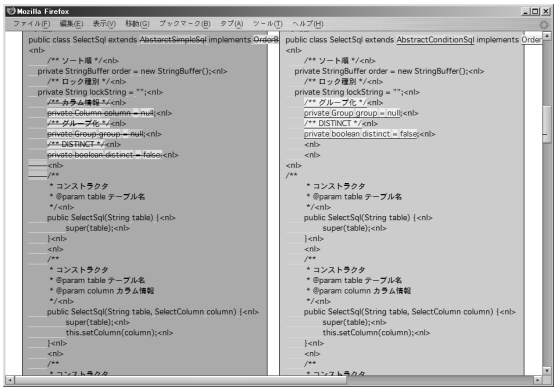

図 13: 差分表示ツール

次に XSDML を用いて作成されたソースプログラムブラウザ JSPIE[7] への応用について述べる．JSPIE の特徴は主に 2 つあり，1 つは定義と参照の間をリンクするクロスリファレンス機能を持つ Java ソースブラウザであること，もう 1 つはソースプログラム上に付箋を付けることができることである．この付箋機能がプログラムの理解を支援し，プログラムを対象とするコミュニケーション機能を提供する．こうして付けられた付箋は貴重な資産であるが，JSPIE はソースプログラムの更新を考慮していないため，あるバージョンのソースプログラムに対する付箋を，新バージョンに反映させることができない．XSDML 間の差分を用いることで，旧バージョンへの付箋を新バージョンでも対応する場所に自動的に移動させて表示することができる．

6 むすびに

構文木に着目して XML マークアップされたソースプログラムである XSDML を対象に差分抽出を行う手法を提案した．そのために差分抽出アルゴリズムの一般的な手法について述べた．次に，木構造差分抽出アルゴリムに適用するために，XSDML を木構造へ変換する手法を提案した．また，部分木の移動操作を含む Chawathe らのアルゴリズムを応用した．さらに，差分抽出アルゴリズムによって得られるエディットスクリプト表現形式について，CASE ツールへの応用を考慮した差分表現を提案した．この差分表現は XSDML と重ね合わせられる名前空間を用いており，編集操作 の対象位置が明確に表示できるという特徴がある．最後に，CASE ツール・プラットフォームに組み込むことによって差分を意識したツール開発を実現した．

今後の課題としては XMI や DocBook など，CASE ツールで扱う他の XML 文書に対応を広げることである．

謝辞

本研究を行うにあたり熱心な御指導，御鞭撻をくださった愛知県立大学情報科学部の稲垣康善教授，滋賀大学経済学部の齋藤邦彦助教授，および山本研究室の皆様に深く感謝すると共に厚く御礼申し上げます．本研究は文部科学省科学研究費補助基盤研究 (A)『「ソフトウェア=プログラム+ドキュメント」の視点に基づく多言語対応大規模コーパス』(課題番号 16200001) を受けて行われた．

参考文献

[1] 福安直樹, 山本晋一郎, 阿草清滋. 細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid. 情報処理学会論文誌, Vol. 39, No. 6, pp. 1990–1998, 6 月 1998.

[2] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change Detection in Hierarchically Structured Information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 493–504, Montréal, Québec, June 1996.

[3] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of the ACM SIGMOD Conference International Conference on Management of Data*, pp. 26–37, June 1997.

[4] Sudarshan S. Chawathe. Comparing Hierarchical Data in External Memory. *Proceedings of the Twenty-fifth International Conference on Very Large Data Bases*, pp. 90–101, September 1999.

[5] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, Vol. 26, No. 3, pp. 422–433, July 1979.

[6] Kaizhong Zhang, Jason Tsong-Li Wang, and Dennis Shasha. On the Editing Distance Between Undirected Acyclic Graphs. *Int. J. Found. Comput. Sci.*, Vol. 7, No. 1, pp. 43–58, March 1996.

[7] 沢田洋平, 大久保弘崇, 粕谷英人, 山本晋一郎. 付箋によるコミュニケーション機能を備えたソフトウェアブラウザ. 電気情報通信学会ソフトウェアサイエンス研究会, Vol. 103, No. 189, pp. 13–18, 7 月 2003.