プログラム解析情報のXMLデータベース化 - 提案と実現 -

XML Database for Program Analysis Information

山中 祐介 † 大畑 文明 † 井上 克郎 †
Yuusuke YAMANAKA Fumiaki OHATA Katsuro INOUE
{y-yamank,oohata,inoue}@ics.es.osaka-u.ac.jp
† 大阪大学 大学院基礎工学研究科
Graduate School of Engineering Science, Osaka University

現在，様々なプログラミング言語に対する解析手法・解析ツールが多く提案・開発されている．一般に，解析ツールから得られる情報は，メモリ上に記憶され一時的なものでしかない．また，記憶形式がアプリケーション独自であるため，二次利用が難しいといった問題がある．本稿では，文書構造化言語である XML を用いた，プログラム解析情報のデータベース化手法の提案を行う．また，実際に Java プログラムを対象とする XML データベースを構築し，その有効性について検証，考察する．

1 まえがき

ソフトウェアの大規模化，複雑化，プログラミング言語の高級化に伴い，プログラムデバッグ，プログラム理解はより困難なものとなりつつある．そのため，様々なプログラミング言語に対するプログラム解析手法の提案，及びツールの実装が多くなされてきた．

このようなツールでは，プログラム解析情報はメモリ上にのみ記憶されるのが一般的である．そのため，解析情報を必要とする際にはその都度対象プログラムを解析しなければならず，効率が悪い．解析情報データベースを構築しそれに対する API を定義することで，解析情報の再利用が可能となるツールも存在するが，データベースが独自形式であったり，API が特定のプログラミング言語を前提としているものが多く，解析情報の二次利用が容易であるとは言い難い．

本稿では，XML を用いてプログラム解析情報のデータベース化を行う手法を提案する．データベース化による解析効率の向上はもちろんであるが，XML を対象とするアプリケーションが数多く提供されていることから，二次利用の容易性も期待できる．また，我々が開発している Java プログラム解析フレームワークに対し，提案手法を追加実装し，その有効性を確認した．

以降，2. でプログラム解析情報の XML データベース化の提案を行う．3. で提案手法の実現について述べ，4. で手法の評価および関連研究について考察する．最後に 5. でまとめと今後の課題について述べる．

2 プログラム解析情報のXMLデータベース化

プログラム解析情報の例としては，抽象構文木，手続き呼び出しグラフ（*Call Flow Graph*），制御フローグラフ（*Control Flow Graph*）などがあるが，本稿では意味解析木に着目する．意味解析木は多くのプログラム解析手法においてその存在が前提となる解析情報の一つであり，データベース化による効果は大きい．

以降，意味解析木，XML についてそれぞれ簡単に説明したのち，具体的なプログラミング言語として Java[2] を対象とする，意味解析木の XML データベース化手法の提案を行う．

2.1 意味解析木

一般にプログラムのソースコードはテキストで書かれており，プログラミング言語は言語固有の文法を持っている．抽象構文木（*Abstract Syntax Tree, AST*）とは，この文法に従い記述されたソースコードを木構造で表現したものである．

また，プログラムには，意味情報（*Semantic Information*）と呼ばれる，変数名や型名などの識別子に関する，宣言と参照間の関係が存在する．

そして，これら抽象構文木と意味情報とを組み合わせたものが意味解析木（*Semantic Tree*）である．

2.2 XML

拡張可能マークアップ言語（*eXtensible Markup Language, XML*）[6] は，文書構造化言語の一つである．

XML 文書（*XML Document*）は，要素（*Element*）などのマーク付けにより構造化され，木構造を形成する．また，XML はユーザの目的に応じて要素名や属性（*Attribute*）などを自由に定義することができる．つまり，XML はデータ構造およびそれが持つ情報を容易にかつ的確に表現することができる．

さらに，文書型定義（*Document Type Definition, DTD*）を用いることで，要素間の親子関係，属性として持つべき情報を厳密に定義することができる．これにより，XML 文書に潜む致命的な構文間違いを防ぐことが可能となり，データベースが持つべき特性の一つであるデータ一貫性の保証を実現することができる．

2.3 意味解析木の XML データベース化

— 方針

意味解析木の XML 表記に関して，以下 2 つの方針に基づき，Java プログラムの意味解析木に対する DTD 記述を行った．

- 意味解析木と XML 文書は共に木構造を形成している．そこで，XML 文書中の要素に意味解析木の節点を対応させ，意味解析木の構造情報（*Structure Information*）を表現する．

- 意味解析木の各節点には，予約語，演算子などの構文情報（*Syntax Information*）や，前述の意味情報が存在する．これらは，意味解析木に対応する要素の属性に保持させる．

意味解析木節点と要素との対応表の一部を表 1 に，属性の一覧を表 2 に示す．

— 適用例

図 1 に示す Java プログラムに対し，その意味解析木の XML 表記を図 2 に示す．図 1 の代入演算 `b = a + 1` は，図 2 の `<Operation text="=">` に対応する．第一子要素 `type` は演算結果の型を，第二，第三子要素はそれぞれ代入演算の左被演算子，右被演算子を表す．

```
public String toString() {
        ...
  int a = 1, b;
  b = a + 1;
        ...
}
```

図 1: Java プログラム

```
<Method modifiers="public" text="toString"
  id="0x188f">
  <type text="java.lang.String" text_="String"
    ref="0x22" file="./String.xml"/>
  <Block>
    ...
    <Variable modifiers="" text="b" id="0x18ad">
      <type text="int" ref="0x7"/>
    </Variable>
    <Operation text="=">
      <type text="int" ref="0x7"/>
      <variable text="b" ref="0x18ad"/>
      <Operation text="+">
        <type text="int" ref="0x7"/>
        <variable text="a" ref="0x18a9"/>
        <Literal text="1"/>
      </Operation>
    </Operation>
    ...
  </Block>
</Method>
```

図 2: 図 1 意味解析木の XML 表記

3 XML データベースの実現

我々は Java を対象とするプログラム解析フレームワークを構築しており，これまでにこのフレームワークを利用してエイリアス解析ツール [3] を開発している．今回，このフレームワークへの機能追加の形式で提案手法を実現し，その有効性について検証した．

以降，我々の解析フレームワークについて簡単に紹介し，提案手法の実装である，意味解析木-XML 相互変換ライブラリについて述べる．なお，有効性検証に関しては次節で述べる．

3.1 Java プログラム解析フレームワーク

図 3 にその概要を示す．なお，上段が既存のフレームワークであり，下段が提案手法の実現により追加された部分である．

ソースコードが与えられると，解析ライブラリにより意味解析木がメモリ上に構築される．ユーザは解析ライブラリが用意した API を介して意味解析木を参照，更新することができる．また，プログラム解析ツールのひな型となる，テキスト編集機能を持った GUI も用意されている．

表 1: JAVA プログラムと要素の対応表 (一部)

JAVA プログラムでの役割	XML 要素名	要素が持つ子の BNF 記述
メソッド宣言	Method	type Variable* Exception? Block?
変数宣言	Variable	type Expression_?
ブロック	Block	(Variable | ClassType | InterfaceType | FakeType | Statement_)*
演算	Operation	type Expression_ Expression_?
型参照	type	子要素なし
変数参照	variable	子要素なし
リテラル	Literal	子要素なし

表 2: 要素が持つ属性とその役割

属性名	役割
text	ユーザ定義の識別子，また演算子などの情報が入る． ただし，名前は限定名であり Object であれば java.lang.Object として保持する．
text_	属性 text の単純名が入る．java.lang.Object であれば Object として保持する．
modifiers	クラス宣言などにおける修飾子の情報が入る．
id	プログラム中の宣言 (クラス，メソッドなど) や制御文 (for 文など) が持つ属性． 宣言・参照関係の情報である意味情報の宣言側が持つ ID である．
ref	変数や型などの参照側が持つ属性．また，break 文など飛び先を指定する文も持つ． 宣言側が持っている ID を参照している．
file	参照先が同一ファイル内でない場合，この属性の持つ情報により他のファイルへアクセスする．

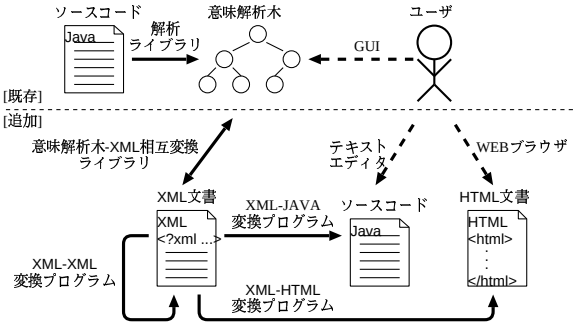

図 3: JAVA プログラム解析フレームワーク

3.2 意味解析木-XML 相互変換ライブラリ

提案手法を，意味解析木と XML 文書とを相互変換するライブラリとして実装した（図 3 左下）．なお，XML パーザには libxml[5] を採用した．

ソースコードを読み込みメモリ上に構築された意味解析木を XML 文書に変換しファイルシステム上に保存する．このような XML 文書の集合が XML データベースとなる．以降，ソースコードに対応する XML 文書が既にデータベースに存在する場合，ソースコードではなく XML 文書を読み込み，メモリ上に意味解析木を再構築する．

4 評価

1. において，XML データベース化により期待される効果として，解析効率の向上，二次利用の容易性を挙げた．本節では，それぞれの観点から提案手法の検証を行う．具体的には，前者はデータベースの有無による解析時間の違いを比較し，後者はいくつかの応用アプリケーションを試作してみた．

4.1 実験 [解析効率の向上]

まず，JAVA Deleloppers Kit(JDK)1.3 付属クラスライブラリの全ソースコード (計 25MB) の意味解析木を XML 文書に変換し，XML データベースを構築した．この変換には 45.6 秒を要し，XML 文書の大きさは計 62MB となった．[1]

次に，意味解析木に関して，JAVA ソースコードからの構築時間とデータベースからの再構築時間との比較を行なった．ソースコードからの構築は 37 秒，データベースからの再構築は 24 秒[2]であり，解析コストは約 40%削減されたことになる．

4.2 応用アプリケーション [二次利用の容易性]

ここでは，XML データベースに保存された解析情報の二次利用の例として，試作したいくつかのアプリケーションを紹介する（図 3 下）．

— XML-JAVA 変換 [XML 文書をソースコードのテキスト表記に変換]

意味解析木の持つ予約語，識別子，演算子などの

[1] libxml には gzip 圧縮された XML 文書の読み込み機能があり，それを併用することで大きさは 10MB にまで軽減される．なお，圧縮ファイルの展開に要するコストは十分に小さいことが確認されている．

[2] 再構築時間は XML パーザーの性能に大きく依存する．そのため，より高速な XML パーザーを利用することで再構築時間の短縮が期待される．

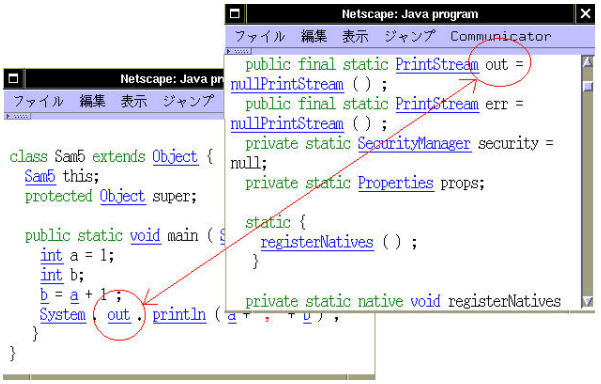

図 4: XML-HTML 変換

構文情報を利用し，コンパイル可能でかつ十分に閲覧可能なソースコードの復元を行う．

これには libxml を利用した C++実装と，*XSL Transformations, XSLT*[7][3]による実装がある．なお，実際の変換を行う XSLT プロセッサには Xalan[4] を利用した．

— XML-HTML 変換 [XML 文書をソースコードの HTML 表記に変換]

上記の変換に加え，HTML タグを埋め込むことで WWW ブラウザを介した情報閲覧を可能にする．さらに，識別子の宣言と参照の関係はリンクを用いて表現されている．これは XSLT による実装である．

図 4 にその例を示す．変数 `out` の宣言と参照間にリンクが張られていることが分かる．

— XML-XML 変換 [XML 文書中の識別子を置換]

識別子の置換は一般的なエディタでも可能であるが，置換操作を行う際に異なるスコープ上に存在する同一名の識別子を区別することはできない．そのため，誤操作によって意味情報に不整合を生じさせてしまう可能性がある．

これは XSLT を用いて実装され，置換の際に識別子名と `id` 属性を与えることで上記の問題を回避する．

4.3 関連研究

プログラム解析情報の XML 表記に関する研究として，JavaML[1] がある．JavaML は，ソースコードに対して XML タグを埋め込む方式で，プログラム変換，プログラム理解を主な目的としている．

しかし，意味情報が十分に含まれておらず，意味解析木データベースとしては成り立たない．例えば，単一ファイル内での宣言と参照間との関係はサポートされているが，複数ファイル間におよぶ関係は考慮されていない．

5 まとめと今後の課題

一般に，プログラム解析ツールにより得られる解析情報はメモリ上にのみ存在し，解析情報を再利用することはできない．データベースを利用することで再利用を考慮したツールも存在するが，データベースが独自形式であるために他のアプリケーションでの二次利用が容易ではない．

本稿では，これら 2 つの問題を解決する手法として，プログラム解析情報の XML データベース化を提案した．また，Java プログラム解析フレームワークへの提案手法の追加実装，及びいくつかの応用アプリケーションを試作し，解析効率の向上，二次利用の容易性を確認した．

今後の課題としては，意味解析木以外の解析情報，例えば，手続き呼び出しグラフ，制御フローグラフなどのデータベース化が考えられる．また，データベースの持つ情報量とその管理コストは比例関係にあるため，ユーザの目的に応じて情報量を制御できる，すなわち，XML 表記の粒度をカスタマイズできる機構の実現も考えている．

[3] XSLT は XML 文書のスタイル指定言語である *eXtensible Stylesheet Language, XSL* の一部で，XML 文書の木構造に対し，各節点に一致する条件とその条件に一致した時に行う処理のルールを記述するものである．これを利用することで，XML 文書を他の XML 文書や HTML 文書などに変換することができる．

参考文献

[1] G. J. Badros, “JavaML:a markup language for Java source code,” Computer Networks 33, pp.159–177, 2000.

[2] J. Gosling, B. Joy, and G. Steele, “The Java™ Language Specification,” Addison-Weseley, 1996.

[3] 大畑 文明, 近藤 和弘, 井上 克郎, “エイリアスフローグラフを用いたオブジェクト指向プログラムのエイリアス解析手法,” 電子情報通信学会論文誌 D-I, Vol.J84-D-I, No.5, pp.1-11

[4] `http://xml.apache.org/xalan-c/index.html`, “Xalan-C++ version 1.1.”

[5] `http://xmlsoft.org/`, “The XML C library for Gnome.”

[6] `http://www.w3.org/TR/2000/REC-xml-20001006`, “Extensible Markup Language (XML) 1.0 (Second Edition).”

[7] `http://www.w3.org/TR/xslt`, “XSL Transformations (XSLT).”