特別研究報告

題目

XML による Java プログラム解析情報のデータベース化

指導教官
井上 克郎 教授

報告者
山中 祐介

平成 13 年 2 月 22 日

大阪大学 基礎工学部 情報科学科

平成 12 年度 特別研究報告

XML による Java プログラム解析情報のデータベース化

山中 祐介

内容梗概

近年，ソフトウェアの大規模化・複雑化に伴い，ソフトウェア開発・保守には多大なコストを要する．そのため，プログラム構造解析と呼ばれる，プログラムを解析し情報抽出を行うソフトウェア開発支援技術が頻繁に利用されるようになった．現在では，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．

我々の研究グループにおいても，Java を対象言語とするプログラム解析フレームワークの開発を行なっている．その解析フレームワークによって得られる Java プログラムの解析情報はメモリ上に保持される．しかし，解析情報はメモリ上にのみ情報が蓄積されるため，実行の度に対象のソースコードを解析しなければならない．また，解析情報が独自形式であるため他のアプリケーションでの二次利用が難しいといった問題がある．

そこで，本研究では Java プログラムを解析した結果を保存するデータベースの作成を行う．データベース化を行うことによって，一時的でしかなかった情報を二次利用することが可能となる．また，解析情報をデータベース化したことによってプログラム解析手順を省略し全体の効率が向上することが期待できる．

データベース化を行うための手段として文書構造化言語である XML を採用する．XML を用いることによってプログラム解析情報の構造を定義する．XML をサポートしているアプリケーションは数多くあることから，そのアプリケーションを利用することにより解析情報の二次利用を容易に行うことができる．

実際に，Java プログラム解析フレームワークによって得られる解析情報と XML データベースが持つ情報を相互に変換するためのライブラリ，および XML データベースを対象とする 4 つのアプリケーションの実現を行う．

主な用語

Java
XML
データベース (DataBase)

目 次

1 まえがき **3**

2 XML **5**

2.1 XML の特徴 5

2.2 XML について 6

2.2.1 文書構成 6

2.2.2 XML プロセサ 8

2.2.3 XML 文書へのアクセス 8

2.2.4 XML 文書へのスタイル指定 9

2.3 XML を用いることの利点 12

3 Java プログラム解析情報の XML データベース化 **13**

3.1 Java プログラムと意味解析木 13

3.2 データベースの構成 13

3.2.1 タグ名の定義 13

3.2.2 タグ情報の定義 15

3.2.3 タグの生成規則の定義 17

4 ツールの実装 **21**

4.1 全体構成 21

4.2 意味解析木-XML 変換ライブラリ 22

4.2.1 仕様 22

4.2.2 アルゴリズム 23

4.2.3 実装 23

4.2.4 検討と評価 24

4.3 解析情報の XML 化による応用アプリケーション 25

4.3.1 XML からソースプログラムに変換 25

4.3.2 XML から HTML に変換 26

4.3.3 使用例 27

4.3.4 XML から HTML に変換 (深さ制限) 27

4.3.5 XML データベースの変数置換 29

5 あとがき **32**

謝辞 **33**

参考文献 **34**

1 まえがき

プログラムを記述するためのプログラム言語というものは様々な目的のため数多く作られてきた．年々ソフトウェアの規模は増大しており，そのソフトウェアを作成するためのプログラム言語の構造の規模も増大し，また複雑化しているといえる．このようにプログラムの規模が大きくなれば大きくなるほど，それを理解することは困難なものとなる．

そこで，人はプログラムを少しでも容易に理解し，作成するためのツールを欲するものである．そのような経緯から現在では，様々なプログラム言語に対する解析方法，及びツールが提案され実装されている．実際，井上研究室でも Java プログラムに対する解析フレームワークを開発している．そして，その解析フレームワークによって得られる Java プログラムの解析情報とはメモリ上に保持されるものである．また，解析フレームワークによって得られる情報は単一のものではなく多数存在する．

ここで，次の問題点が生じる．解析情報はメモリ上に保持されるものだから，一時的なものでしかないということである．また，Java プログラムはクラスライブラリが提供されているので再利用できるものが多い．よって，解析情報を得るためには実行する度に，ソースコードを解析しなければならない．これらのことから解析効率が悪いと言える．また，Java プログラム解析フレームワークによって得られる解析情報はそのアプリケーション独自の形式であるため，他のアプリケーションとの情報共有が難しい．

本研究では Java プログラム解析情報をデータベース化することを行う．プログラム解析情報をデータベース化することによって，一時的でしかなかった解析情報を二次利用することが可能となる．また，解析情報をデータベース化したことによって，解析フレームワークにおける解析手順を省略することができ，解析全体の効率が向上することが期待できる．

データベース化するためには解析情報とはどのような情報を持っているかを知る必要がある．一般的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，そのプログラム固有の構文に基づいて書かれている．Java プログラムのソースもテキストである．そして，プログラムのソースコードは定まった構文 [3] によって書かれているので，それを木構造の形で表現することが出来る．これを抽象構文木と呼ぶ．また，変数，メソッド，型の宣言，参照の関係といった意味情報がプログラムにある．これら抽象構文木と意味情報を合わせたものを意味解析木と呼ぶ．解析フレームワークの情報は意味解析木としての情報を持っているので，その情報をデータベース化する．

次に，文書構造化言語である XML[12] を用いて Java プログラム解析情報の構造を定義する．意味解析木は木構造をしており，XML による構造化によって意味解析木の持っている情報を容易に表現することが出来る．また，XML 文書を扱うためにさまざまなアプリケーションが提供されているので，XML を容易に扱うことが出来る．このことから，XML に

よってデータベース化されたJavaプログラム解析情報を容易に利用することができる．

Javaプログラム解析フレームワークとXMLデータベースとの間でデータをやりとりするため，プログラム解析ツールが構築する意味解析木とデータベースとして保存されているXML文書をお互い変換するためのライブラリを実装する．また，データベース化された解析情報を利用するためのアプリケーションを実装する．このアプリケーションの実装により，XMLデータベース化されたJavaプログラムの解析情報を幅広く利用することが可能となる．

以下，2節でXMLについて述べる．3節では解析情報のデータベース化について述べる．4節では実装したツールについて述べる．最後に5節ではまとめと今後の課題について述べる．

2 XML

XML(eXtensible Markup Language) は，文書構造化言語を記述するためのメタ言語である．

文書構造化言語とは，文書構造を記述するためのものであり，文書構造化言語を利用することにより構造が明確で整合性を持つ文書を記述することが可能となる．

文書構造化言語の一つの例として，TeX/LaTeX があげらる．これらは本来自然科学系の論文を書くために開発されたものだが，精通すれば非常に多種多様な文書を書くことができる．また，インターネットのホームページを書くときに利用する HTML(HyperText Markup Language) も文書構造化言語の代表的なもので，電子的参照用に単純化されたものである．

XML は, そのような文書構造化言語を定義するためのものであり，最も代表的な文書構造化言語 SGML(Standard Generalized Mrakup Language) をベースとする言語である．

XML はインターネット上での使用を考慮して作られており，文書やデータの交換，配布に対して標準となる可能性の高い重要な言語である．また，データの格納保存や処理対象としても非常に有用な言語である．本研究では XML の持つデータの格納保存や処理に対する能力を評価し，採用した．

2.1 XML の特徴

XML の特徴の一つは，XML で書かれた文書は読みやすく，また理解しやすいということである．なぜなら，XML はテキスト文書として書かれたものであり，また XML の構文は単純でありなおかつ厳密なものであるからである．

次の特徴として，文書構造化言語のメタ言語であることから，タグを自由に定義することができるということである．自分で定義をすることができるということは，タグとその内容を目的に応じて関連付けることができるということである．例えば，”<date> 平成 13 年 2 月 22 日 </date>” と記述されていれば，date というタグで囲まれた” 平成 13 年 2 月 22 日” という内容が日付であると関連付けることができる．

他の特徴として，XML は HTML と違い，大文字と小文字の区別をする．また，使用する文字コードが宣言されていれば，タグ名にはアルファベットや数字以外，例えば日本語など，を使用することができる．

2.2 XML について

2.2.1 文書構成

XML 文書は大きく分けて，XML 宣言，DTD，XML インスタンスの三つの部分から構成されている．

XML 宣言 (XML declaration) は XML バージョン宣言，文字コード宣言，スタンドアロン文書宣言の三つからなる．XML バージョン宣言は XML のバージョン情報を XML を処理するプロセッサに指定するためのものである．文字コード宣言は文書中で使用されている文字コードを指定するためのものである．スタンドアロン文書宣言は，現在対象となっている文書以外に処理する文書が存在するかどうかを指定するためのものである．XML 宣言において，XML バージョン宣言は必須のものであり，その他二つは必ずしも必要なものではない．

DTD(Document Type Definition) は XML 文書内に出現する要素，属性，エンティティを定義するためのものである．XML 文書内に現れる要素は要素型宣言で定義することができる．要素型宣言によって要素の名前，要素の親子関係を定義することができる．要素が持つ属性は属性リスト宣言によって定義することができる．属性リスト宣言は，要素の持つ属性の名前を定義し，その属性はどのような型 (文字データ，ID，ID 参照など) を取るか，または列挙型であればどのデータをとるか，その属性は必須であるかどうか，そしてデフォルト値を指定することができる．エンティティ宣言は XML 文書内で参照したいエンティティを定義するためのものである．文書内で参照し変換する文字列や外部ファイル名を指定する．

XML インスタンスは，実際に書かれているタグ付き文書である．文書は，文字データ及びマーク付けから成る．マーク付けは，開始タグ，終了タグ，空要素タグ，実体参照，文字参照，コメント，CDATA セクションの区切り子をとる．文書中の各要素の境界は，開始タグ及び終了タグによって区切られる．要素が空要素のときは，空要素タグで示す．また，XML 文書は文書要素という要素が一つだけ存在し，これは，他の要素の内容に含まれない．これ以外のすべての要素は，その開始タグが他の要素の内容に含まれれば，対応する終了タグも同じ要素の内容に含まれる．つまり，要素は，開始タグ及び終了タグによって区切られ，入れ子構造をなす．

ここで，XML 文書には XML 宣言と DTD は必ずしも必要なものではない．すなわち，タグ付き文書において開始タグと終了タグの対応がとれており，厳密に入れ子構造を満たしておれば，それだけで XML 文書として成り立つということである．

XML 文書の例を以下に示す．

XML 文書の例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE skill SYSTEM "sample.dtd">
<document>
  <author id="9087">山中祐介</author>
  <date>
    <year>2001</year>
    <month>1</month>
    <day>24</day>
  <date>
</document>
```

1 行目に書かれている <?xml · · · > にあたるのが XML 宣言である．ここでは，バージョン情報と文字コード情報が書かれている．2 行目は，文書型宣言と呼ばれるもの，この宣言によって XML 文書に対する DTD を指定する．3 行目以降が実際の文書である．”document” タグの中に”author” などが含まれていて，これは厳密に入れ子を満たしている．また，”author” 中の”id” を属性と呼ぶ．この XML 文書を DTD によって記述したものが次のようになる．

DTD の例

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT document (author, date)>
<!ELEMENT author (#PCDATA)>
<!ATTLIST author id ID #REQUIRED>
<!ELEMENT date (year, month, day)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT day (#PCDATA)>
```

<!ELEMENT に始まるのが要素型宣言である．”()” により子要素にどのような要素を指定することができる．”#PCDATA” であれば，子要素はなく文字列をもつ．また，<!ATTLIST に始まるのが属性リスト宣言である．”ID” によって属性 id は ID 型であることをさす，それ以外には CDATA などがある．#REQUIRED によってその属性を必要性を示している．この場合は，属性は必須であることを示す．

2.2.2 XML プロセサ

XML 文書を処理するためには XML プロセサというソフトウェアモジュールを使用する．XML プロセサは XML 文書を読み込み，その内容及び構造へのアクセスを提供するために用いる．また，XML プロセサは他のモジュールのために動作することを前提としており，そのモジュールをアプリケーションという．

XML 文書は処理上の観点から二つのレベルに分類することができる．その一つが整形式 XML 文書 (well-formed XML document) であり，もう一つが検証済み XML 文書 (valid XML document) と呼ばれるものである．

整形式 XML 文書とは，開始タグと終了タグの対応がとれており，親子関係にある要素のタグがきちんとした入れ子構造になっているなど XML で規定されているタグ付けの規則にしたがって記述されている XML 文書である．

検証済み XML 文書とは，DTD によって行われる要素型宣言，属性リスト宣言において定義された要素の親子関係，および属性の型に従ってタグ付けが行われている XML 文書のことをいう．

これら二種類の文書の違いは DTD による要素，属性の定義が行われているかどうかである．すなわち，検証済み XML 文書とは整形式 XML 文書であり，なおかつ DTD による要素に対する定義が行われているものである．

XML プロセサは整形式であることを満たしているかどうかを解析する．もし，DTD による要素や属性に対する定義が行われていれば，XML インスタンスがその定義を満たしていかどうかも解析する．この整形式であることや DTD の定義に対する違反があれば XML プロセサは，その違反を報告する．XML プロセサはこの解析結果をアプリケーションに受け渡す．

2.2.3 XML 文書へのアクセス

XML 文書の内容，構造，スタイルにアクセスし，XML 文書の参照や更新をするために DOM(Document Object Model) という API が XML パーサーより提供されている．DOM は XML 文書を一つの木構造とみなし，DOM ツリーと呼ばれる木を作っている．DOM ツリーは XML 文書解析時にメモリ上に構築される．この木に対して参照する手段を DOM は提供しているので，これを利用することによって XML 文書内のデータの検索や，要素や属性の追加および削除などが容易に行なうことができる．

ある XML 文書に対して構築される DOM ツリーは次のような形状をしている．

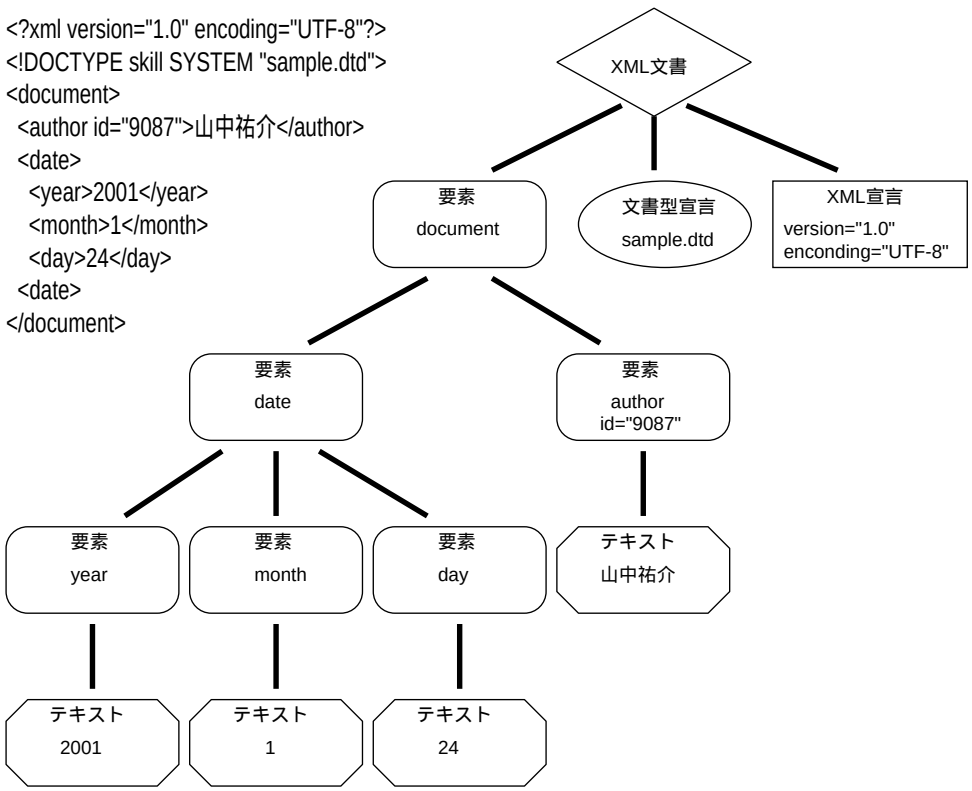

図 1: DOM ツリーの概観

2.2.4 XML 文書へのスタイル指定

XML は，基本的にスタイル情報の記述を行わない．そのため，HTML のような使用するタグセットが固定化され基本的なスタイルと一体化した文書とは違い，タグ付き文書を記述したからと言ってそれをすぐに表示することは不可能である．XML は使用できるタグセットが固定化されていないので，XML 文書を表示するためにはユーザが定義したタグに対してそれがどんなスタイルで表示されるべきなのかを決める必要がある．そのためのスタイル指定言語に XSL(eXtensible Stylesheet Language) がある．

XSLT(XSL Transformations) は，XSL から Tree Construction の部分を抜き出したものである．XSLT では，ソースツリーに対して，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り，これを利用することにより XML 文書を他の XML 文書や HTML 文書に変換することができる．

XSLT を使った XML 文書から HTML 文書へと変換した例をこれから示す．

まず，以下の XML 文書を変換する．

変換対象の XML 文書

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
  <person id="1">
    <name>Yamamoto</name>
    <age>23</age>
  </person>
  <person id="2">
    <name>Yamanaka</name>
    <age>22</age>
  </person>
  <person id="3">
    <name>Yamada</name>
    <age>21</age>
  </person>
</persons>
```

この XML 文書には人物の名前と年齢，そしてそれぞれの ID を情報として持っている．XML 文書の変換を行なうためにはその変換のルールが XSLT によって記述されたスタイルシートを用意する．そのスタイルシートを XML 文書とともに XSLT プロセッサに渡すことによって変換することができる．ここでこの XML 文書を変換するためのスタイルシートの例を次に示す．

XSLT 文書による変換ルール

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
<xsl:output method="html" encoding="UTF-8"/>

<xsl:template match="/">
<HTML>
<BODY>
<xsl:apply-templates select="/persons/person"/>
</BODY>
</HTML>
</xsl:template>

<xsl:template match="person">
  No<xsl:value-of select="@id"/>:
  <xsl:value-of select="name"/><BR/>
</xsl:template>
</xsl:stylesheet>
```

4 行目の”xsl:output” によって出力形式を指定する．今は，HTML 形式で文字コードは UTF-8 である．”xsl:template” によって目的のノードに一致させる．一つ目はルートであり，二つ目は”person” である．テンプレートの中で出力する文字列を指定する．一つめのテンプレートでは”HTML” と”BODY” タグを出力する．また，”xsl:apply-templates” によって任意の目的のノードを探索する．ここでは，”persons” 要素の子”person” 要素を探索する．探索され発見されるごとに，二つ目のテンプレートを呼び出す．このテンプレート中の”No” と”:” は文字列として出力される．”xsl:value-of” によって XML 文書の内容を出力する．”@” を付加していれば属性が出力される．変換して結果は次の通り．

HTML 文書に変換された XML 文書

```
<HTML>
    <BODY>
  No1:
  Yamamoto<BR>
  No2:
  Yamanaka<BR>
  No3:
  Yamada<BR>
    </BODY>
</HTML>
```

2.3 XML を用いることの利点

本研究において，XML を利用するということは次のような利点を挙げることが出来る．

一つ目として，文書構造化言語であるということである．このことから XML は意味解析木と同様に木構造を形成している．よって Java プログラムの情報を容易に構造化して保管できるようになる．また，構造化することによってプログラム内の関係を表現できる．

二つ目はタグの決定の自由さである．XML は自分で目的にあったタグを定義することができる．そのことから，Java プログラムの持つ情報を定義したタグと関連づけることにより，タグ名から求める情報を参照することが出来るようになる．これにより，データベースとしてアクセスすることが容易になる．

三つ目は DTD の存在である．整形式の XML 文書として Java プログラムに関する情報を蓄積することもできるが，それでは各要素間の関係を保証することは出来ない．しかし，DTD を用いることにより，要素間の親子関係，属性として持つ情報を厳密に定義することができ，データに対して致命的な構文間違いを防ぐことになる．これはデータベースの持つべきデータの一貫性を保証するということが実現できることになる．

最後に，汎用性の高さと将来性である．XML 文書にアクセスする手段として DOM が提供されており，また，情報の視覚かという観点では XSLT が提供されている．これらにより，XML 文書として蓄積されたデータを扱うことが容易といえ，プラットフォームを選ばないともいえる．現在，XML は非常に注目されており，XML によるデータの配布，交換もますます盛んになると思われる．このことからも XML により，データを蓄積すると言うことは意味のあることだといえよう．

3　Java プログラム解析情報の XML データベース化

本研究では XML を用いて，Java のプログラム解析情報を保存する．保存するといっても，Java プログラムの何を保存するのか，また，解析情報とはどういう形をしているかを明示しなければならない．ここでは，Java プログラムの構造と解析情報である意味解析木について述べ，XML によってデータベース化するためのルールを定義する．

3.1　Java プログラムと意味解析木

一般的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，そのプログラムには固有の構文を持っている．

Java は有名なオブジェクト指向言語の一つである．Java の基本的な分解単位はクラスである．クラスにおいてメソッドと呼ばれる関数に似たものを定義することによって，その振舞いが定義できる．また，クラスのインスタンスに関連付けられる状態変数を定義できる．このようにしてクラスは一つのオブジェクトセットとしての振舞いを指定する．また，クラスはそれが持つ振舞いを別のクラスに継承することができる．

プログラム言語はそれぞれ固有の正規表現を持っている．その正規表現は各文字を組み合わせてトークンを形成する方法を表している．また，プログラム言語は固有の文法を持っている．その文法は他の構成体や基本的なトークンを組み合わせることによって高いレベルの構成体を機能的に形成する方法を定義している．この正規表現と文法によってあるプログラム言語のソースコードを木構造の形で表現することができる．また，変数，メソッド，型の宣言，参照の関係といった意味情報がプログラムにある．これらを合わせたものを意味解析木と呼ぶ．意味解析木はプログラムの構造をより直接的に表現し，プログラムの理解を手助けすることができる．[11]

3.2　データベースの構成

上記の意味解析木の情報を XML によってデータベース化するにあたって，定義しなければならないことは，要素名 (タグ名) をどのように定義するか，各々の要素はどのような情報を持つか，そして，それぞれの要素間の関係はどうなっているかである．

3.2.1　タグ名の定義

Java プログラムの解析情報を保存するために XML を用いて構造化する．XML の要素名はその要素がどのような内容を保持しているかを強く表現するので，構造化する対象を表現するのに適した名前でなければならない．構造化の対象は意味解析木になるので，意味解析木の構造を XML タグの名前で表現する．

しかし，ただ定義するだけでは実用性に欠ける．本研究ではこのデータベース化された情報の利用先は，解析ツールのために意味解析木を渡すことと，XSLT で変換しての利用としている．意味解析木をそのままの形で定義するのも良いが，XSLT での使用をみこし，構造ができるだけ単純になるように，またタグの種類を必要最小限にとどめるように設計した．

XML タグの Java プログラムにおける意味合いの関連を次に示す．

XML タグの名前	Java プログラム
Java	
CompilationUnit	コンパイル単位
Package	パッケージ宣言
Import	インポート宣言
ClassType	クラス宣言
InterfaceType	インターフェイス宣言
FakeType	
Extends	クラス継承
Implements	インターフェース継承
Method	メソッド宣言
Constructor	コンストラクタ
Exception	例外 (キーワード”throw”)
Variable	変数宣言
InstanceInitializer	インスタンス初期化
StaticInitializer	静的初期化
Block	ブロック
If	if 文
While	while 文
Do	do 文
Switch	switch 文
SwitchBlock	switch 文内ブロック
Case	case 文，defaule 文

表 1: XML タグと Java プログラムの関連表 1

XML タグの名前	Java プログラム
For	for 文
ForInit	for 文初期化部
ForExp	for 文判定部
ForRenew	for 文更新部
Try	try 文
Catch	catch ブロック
Finally	finally ブロック
Synchronized	synchronized 文
Break	break 文
Continue	continue 文
Retutn	return 文
Throw	throw 文
Labeled	ラベル文
Empty	空文
variable	変数参照
type	型参照
method	メソッド呼びだし
Literal	リテラル
ClassInstanciation	クラスインスタンス化
ArrayInstanciation	配列インスタンス化
ArrayInitializer	配列初期化
Operation	演算

表 2: XML タグと Java プログラムの関連表 2

3.2.2 タグ情報の定義

タグ名を定義することによって，XML 文書はどのような構造になっているかを表現することができた．しかし，それだけでは Java プログラムの情報やその解析情報をどのような形で保持しているかは表現されていない．そこで，プログラムの持っている情報を XML 文書においてどのような形で保持しているかを定義する．

Java プログラムには，キーワード，ユーザが定義したクラス名や変数名など，また演算子

などの情報がある．それらの情報を先ほど定義した各要素の属性に持たせるとする．属性に情報をすべて持たせると要素の内容で持たせることとは違い，要素の開始タグに情報を集約させることができる．そうすれば，内容に情報をもたせたためにその情報が飛び飛びなってしまい検索に不都合が生じるということがなくなる．また，XML 文書が冗長なものとなってしまうこともなくなる．

ここで，各要素がもつ属性を次のように定義する．

- text
  Java プログラムの持つクラス名，メソッド名，変数名，また参照先のファイル名や演算子とソースプログラムにおける目に見える情報を格納するために定義した属性である．

- text_
  この属性は，属性 text の亜種といえる．属性 text はプログラムの持つ情報を保持しており，また名前においては完全な形で保持している．例えば，java.lang.Object などである．ここで，インポート宣言がされている場合，このような完全な形で保持する必要がない．そこでこの属性 text_によってソースプログラムに影響を与えない程度に省略された情報を保持することができるようにする．例えば，java.lang.Object が Object として格納される．また，親クラスへの参照とする super もこの属性によって適用される．

- modifiers
  これは，クラス名，メソッド名，変数名などにおける修飾子の情報を格納するための属性である．この属性の特徴として複数の修飾子がついていたとしても，一つにまとめられて格納されるということである．

- id
  プログラムにおいて，常に定義する側と参照する側が存在する．属性 id はクラス名や変数名，ラベル名，また for 文などの制御文が持つ属性でソースプログラムにおいて定義されていること，そしてそれが参照される可能性を持っていることを表すために定義した属性である．また，属性 id を機能させるために同一ファイル内には id の番号は一つしかないものとする．

- ref
  属性 ref は参照する側がもつ属性である．変数参照，型参照，メソッド参照といった定義されているものを参照する項目，また break 文，continue 文における飛び先の指定をするために使用される属性である．

- file
  この属性は，属性 ref の拡張のためにある．メソッドや変数の参照先は必ずしも同一ファイル内に存在するとは限らない．それにより，参照先のファイルがいったいどこにあるかという情報が欲しい．その情報を格納するためにこの属性 file を定義する．

属性の使用例を次に示す．

```
<ClassType modifiers="public final" text="String" id="0x22"/>
<type text="java.lang.Object" text_="Object" ref="0x1a" file="./Object.xml"/>
<Literal text="&quot;length must be &gt;= 0&quot;"/>
<Operation text="++"/>
```

3.2.3 タグの生成規則の定義

ここまでで定義された XML タグを用いて XML 文書を作成することによって，Java プログラムの情報，また解析情報をデータベース化することができる言える．しかし，これだけでは各要素間の持つ関係を厳密に表すことができない．例えば，ここまでの定義だけでタグの構造をきちんと記述すれば，整形式の XML 文書として XML プロセサを通すことはできるが，抽象構文木の構造を表し，意味のある情報として活用することができない．そこで，XML の各要素間の親子関係を定義し，DTD を提供することによってばらばらであった XML タグの情報を意味あるものとして結び付ける．

各要素の生成規則を BNF で記述する．

- XML 文書としてのファイル単位を表す．また，Java プログラムとしてのコンパイル単位でもある．コンパイル単位はパッケージ宣言を子に持つことがあり，インポート宣言，及びクラス宣言などを任意の数だけ子に持つ．

  ```
  Java ::= CompilationUnit.
  CompilationUnit ::= Package? Import*
                      (ClassType | InterfaceType | FakeType)*.
  ```

- クラス宣言，インターフェース宣言は子に継承を持つことがあり，メソッド宣言などの各インスタンス宣言を任意の数だけ持つ．

  ```
  ClassType ::= Extends? Implements?
                (Method | Constructor | Variable | ClassType |
                 InterfaceType | FakeType | InstanceInitializer |
  ```

```
                StaticInitializer)*.
InterfaceType ::= Extends?
                    (Method | Variable | ClassType | InterfaceType |
                     FakeType)*.
```

- クラス，インターフェース継承は継承される型を任意の数だけ子に持つ．

```
Extends ::= type+.
Implements ::= type+.
```

- メソッド宣言は戻り値の型を持つ．メソッド宣言，コンストラクタは，引数を任意の数だけ持ち，例外，ブロックを子に持つことがある．

```
Method ::= type Variable* Exception? Block?.
Constructor ::= Variable* Exception? Block?.
```

変数宣言はその型を持ち，初期化のための式を持つことがある．

```
Variable ::= type Expression_.
```

- インスタンス，及び静的初期化子はブロックを子に持つ．

```
InstanceInitializer ::= Block.
StaticInitializer ::= Block.
```

- ここで，XML タグとしては定義していないが説明の便宜のため文を定義する．

```
Statement_ ::= Block | If | While | Do | Switch | For | Try |
               Synchronized | Break | Continue | Return | Throw |
               Labeled | Empty | Expression_.
```

- ブロック中には文の他に変数，クラスなどの宣言を持つ．

```
Block ::= (Variable | ClassType | InterfaceType | FakeType |
           Statement_)*.
```

- if 文は判別式と，then 文と else 文がある．

```
If ::= Expression_ Statement_ Statement_?.
```

- while，do 文は判別式と文を子に持つ．

```
While ::= Expression_ Statement_.
Do ::= Statement_ Expression_.
```

- switch 文は判別式とブロックを子に持つ．ブロック中には case，default 文がある．

```
Switch ::= Expression_ SwitchBlock?.
SwitchBlock ::= (Case | Statement_)*
Case ::= Expression_?
```

- for 文は初期化部，判定部，更新部を持つことがある．また，実行文を子に持つことがある．初期化部は変数宣言または，演算がある．判定部には判別式が，更新部にはメソッド呼び出し，または演算がある．

```
For ::= ForInit? ForExp? ForRenew? Statement_?.
ForInit ::= Variable | Operation*.
ForExp ::= Expression_.
ForRenew ::= method | Operation*.
```

- try 文は try ブロックを持ち，catch ブロックを任意の数だけ持つ．また，finally ブロックを持つことがある．catch ブロックは例外引数とブロックをもつ．finally ブロックはブロックをもつ．

```
Try ::= Block Catch* Finally?.
Catch ::= Variable Block.
Finally ::= Block.
```

- synchronized 文は式とブロックを子に持つ．

```
Synchronized ::= Expression_ Block.
```

- return，throw 文は式を子に持つ．ラベル付き文は文を子に持つ．

```
Return ::= Expression_?.
Throw ::= Expression_?.
Labeled ::= Statement_.
```

- ここで，XML タグとしては定義していないが説明の便宜のため式を定義する．

```
Expression_ ::= variable | type | method | Literal |
                ClassInstanciation | ArrayInstanciation |
                ArrayInitializer | Operation.
```

- メソッド呼び出しは任意の数だけ引数として式を持つ．クラスインスタンス化は，そのクラスと引数をもつ．クラスインスタンス化はメソッド呼び出しとしての性格を持っているので，引数の代わりにメソッド呼び出しを子に持つ．

```
method ::= Expression_*.
ClassInstanciation ::= type method.
```

- 配列インスタンス化はその型を子に持つ．配列インスタンス化と配列初期化子は子として式を任意の数だけ持つ．

```
ArrayInstanciation ::= type Expression_+.
ArrayInitializer ::= Expression_*.
```

- 演算は，その演算結果の型を子に持つ．また，左辺と右辺を式として持つ．ただし，単項演算子の場合は式を一つとする．

```
Operation ::= type Expression_ Expression_?.
```

4　ツールの実装

XML タグの仕様を定義することによって Java プログラムの解析情報を，データベースとして保存することができるようになった．そこで，XML 文書と意味解析木を相互変換するライブラリ，またデータベースとして保存された解析情報を利用するためのアプリケーションを実装した．

4.1　全体構成

XML 文書として解析情報を保存するようになった背景として Java プログラム解析ツールがある．ユーザは解析ツールによって解析された情報を GUI を通して見ることができる．そのことから，データベース化された情報をその解析ツールでにおける意味解析木を相互変換できるようにすることが一つの目的になる．そこで，Java プログラム解析ツールに関連したツールとして意味解析木を XML 文書に変換，また XML 文書から意味解析木に相互変換するためのライブラリを実装する．

次に XML はテキストとして保存されているので，それを理解することは容易である．しかし，それは XML タグのついた文書であるので，必要な情報だけを読みだそうとすれば理解しやすいとはいいがたい．そこで，データベース化された情報を視覚化するためのアプリケーションを XSLT を用いて実装する．XSLT の XML 文書を HTML に変換する機能を用いれば，WWW ブラウザ上で容易にデータベースの持つ情報を得ることができる．また，データベース化された情報を更新するためのアプリケーションも実装する．

これにより，今までユーザがソースプログラムから情報を得ようと思ったときは，エディタなどを通してソースそのものを見るか，解析ツールを用いることによってプログラムの解析情報を得るかであった．そこで，XML によってデータベース化することによってユーザは解析ツールから情報を得るにしてもソースプログラムを逐一解析するのではなく XML 文書から解析情報を得るとことが出来るようになり解析効率が向上する．また，XML データベース化することにから XSLT を用いて HTML 文書へ変換し，閲覧するということもできるようになった．よって，データベース化した XML 文書を Java プログラム解析ツールに対してのみに使用するものではなく，データベース化することによって解析情報を汎用的に使用できることが言えよう．

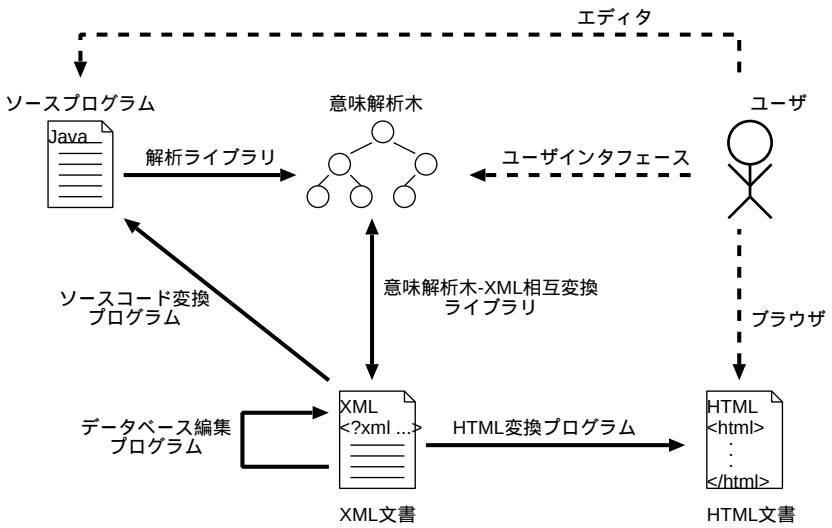

図 2: Java プログラム解析フレームワークの構成図

4.2 意味解析木-XML 変換ライブラリ

Java プログラム解析ツールが扱っている意味解析木と，その意味解析木を XML によって記述された文書とを相互変換するためのライブラリである．

4.2.1 仕様

この意味解析木と XML を相互変換するライブラリの仕様は以下の通りである．

- 意味解析木から XML 文書へ変換
  - 入力
    Java で書かれたソースプログラムである．
  - 出力
    意味解析木の持つ情報を XML によって記述したテキストファイルである．
  - エラー処理
    Java プログラム解析ツールによって構築された意味解析木の構造が Java プログラムの構造として誤りがあった場合，また型参照において宣言されていない型を参照しようとしていたときエラーである．このとき，標準エラー出力に対して出力を行う．
- XML 文書から意味解析木へ変換

- 入力
  XML によってデータベース化された Java プログラム解析情報である．
- 出力
  Java プログラム解析ツールが扱う形式の意味解析木である．
- エラー出力
  入力された XML 文書が整形式でない場合，また定義した DTD に準拠されていない場合はエラーであり，XML パーサーがそれを検出しエラーを標準エラー出力に対して行なう．また，解析情報として意味解析木を構築する時，各ノードの整合性が成り立っていない場合，すなわちノードの親子関係や定義と参照の関係に違反がある場合，エラーとする．

4.2.2 アルゴリズム

この相互変換ライブラリの実行アルゴリズムは次の通りである．

- 意味解析木から XML 文書へ変換
  1. Java ソースコードを読み込む
  2. Java プログラム解析ライブラリによりソースコードを解析
     - 意味解析木をメモリ上に構築
  3. 意味解析木をたどり XML 文書へ変換
  4. 出力
- XML 文書から意味解析木へ変換
  1. XML 文書を読み込む
  2. XML パーサーにより文書を解析する
     - DTD を読み込み，構文が正しいかどうかを判定
     - DOM ツリーを生成する
  3. DOM ツリーをたどり意味解析木に変換
  4. 出力

4.2.3 実装

このツールは Java プログラム解析ツールへと結果を渡す．解析ツールは C++で実装されている．そのことを考慮して C++によって実装した．行数は約 4000 行である．

- 方針
  まず各意味解析木とXMLタグに対応したメソッドを用意することである．このようにしてメソッドを用意することにより意味解析木やXMLタグの構文に対して変更が行なわれていもすぐに対処することができる．また，XML文書の解析を行ない構築されたDOMツリーをたどるルーチンをマクロを用いて抽象的に表記した．この表記を行なうことによってDOMインターフェースを提供しているXMLパーサーであればこのライブラリに対してすぐに組み込めるようになる．よって，目的に応じたXMLパーサーの差し替えが可能となる．

- XMLパーサーによる解析
  XMLパーサーはGNUプロジェクトの一機関であるGNOME[13]が提供しているlibxmlをライブラリとして使用した．このライブラリはXMLの解析はもとより，DOMインターフェースも提供しているので，XML文書の参照，更新も容易に行なうことができる．XML文書を読み込んだのち，このライブラリを使用してXML文書が整形式であることとDTDによって定義した構造を満たしていることを解析する．

- DOMツリーの走査
  解析することによってメモリ上に構築されたDOMツリーにはXMLパーサーが提供しているインターフェースを使用してアクセスすることができる．DOMツリーを定義したXMLの構文に従ってたどり，出力がJavaプログラムであれば，各ノードにおけるtext属性で保持されている情報(クラス名，変数名，演算子等)や，プログラムのキーワードを随時出力する．出力が解析情報であれば，Javaプログラム解析ツールが提供している意味解析木を構築するためのインターフェースを使用することによってDOMツリーがもっている情報を意味解析木に埋め込んでいく．

4.2.4 検討と評価

ここで，Javaのソースプログラムからの解析とXMLデータベースからの解析の違いについて検討する．データベースを構築したことに対する評価を行なう．

解析の実行時間とは対象となるJavaソースプログラム，またはXMLのファイルを読み込み意味解析木を構築するまでの時間である．データベース構築の時間はJavaソースプログラムの解析時間と変換されたXML文書を保存する時間である．

JDK1.3に付属しているクラスライブラリを対象にして比較を行なった．

対象	ファイルサイズ	解析時間	メモリ使用量
Java ソースプログラム	25MB	37 秒	42MB
XML データベース	62MB	24 秒	66MB

また，データベース構築には次の時間だけかかっている．

実行時間	メモリ使用量
45.6 秒	43MB

これらのことから言えることは，XML によりデータベースを構築した方が解析を行なうためにかかるコストは少ないといえる．ただし，データベースから解析した場合，メモリ使用量が増えている．これは，XML 文書を解析して構築される DOM ツリーというものはメモリ上に構築されるものであるので，この DOM ツリーの構築が影響していると思われる．また，ファイルサイズはデータベース化を行なった方が増大している．これはデータベース化を行なった時，ソースコードが持っているテキスト上の情報だけではなく，変数やメソッドなどの宣言と参照の関係情報などを付化されているからである．以上のことからデータベース化によって得られるメリットは大きいといえる．

4.3 解析情報の XML 化による応用アプリケーション

XML 文書としてデータベース化された情報を変換するアプリケーションの実装を行なった．

まず，XML 文書からソースプログラムへ変換するアプリケーションである．これはソースプログラムのテキストファイルへと変換する．

次に，XSLT を用いたアプリケーションである．XSLT は XML の説明の項目に記述したとおり，XML 文書のソースツリーに対して一致する条件を探索し，その条件に対して行なう処理を記述するものである．これによって XML 文書を他の XML 文書や HTML 文書に変換することができる．この機能を用いることによって，Java ソースプログラムを HTML 文書に変換，その HTML 文書にたいしてプログラム階層の深さに制限を与えるもの，そしてデータベースの持つ内容を編集する三つのアプリケーションを実装した．

4.3.1 XML からソースプログラムに変換

XML データベースとして保存されている情報から，Java プログラムのソースコードへと変換する．変換されて出力されるソースコードはテキスト形式である．このアプリケーションと意味解析木と XML 文書を相互に変換するライブラリを使用することによって，Java ソースプログラムと XML 文書間の相互変換も可能となる．

このアプリケーションの実装にはXMLパーサーを利用しており，その実行アルゴリズムは次のようになる．

1. XML文書を読み込む
2. XMLパーサーにより文書を解析する
   - DTDを読み込み，構文が正しいかどうかを判定
   - DOMツリーを生成する
3. DOMツリーをたどりJavaソースコードに変換
4. 出力

4.3.2 XMLからHTMLに変換

XMLによりデータベース化されたプログラム解析情報をHTML文書で表現する．Javaソースコードを表現するためにHTMLの持つ特徴を利用した．ここでHTMLへの変換の方針をまとめる．

- キーワードに色をつける
  Javaプログラムにあるキーワードはプログラム中における役割は決まっている．そして，それはユーザ側が定義できるものではない．よって，ユーザ側が定義できるものと区別して表示する方が望ましい．そこで，fontタグを利用して，キーワードに色をつける．この使用法はキーワードに限らず文字列など様々なところ利用すべき方法だといえる．

- ハイパーリンクを使う
  XMLによってデータベース化する際，クラス名や変数名などは属性idをもつことによって各名前などを明示的に区別している．また，参照する側はそれが持つIDを属性refによって参照している．これらの属性はaタグを用いることによってお互いリンクを張ることができる．また，参照する側は属性fileをもっているので，たとえ別のファイル間での参照でも実現することができる．

- 演算子順位をつける
  ソースコードを出力する際，演算式中の括弧の扱いが問題になる．人の手で書かれた場合はその人の判断で必要なだけ括弧が書かれる．しかし，プログラムによる自動処理であればプログラム中に括弧が必要かどうかを判定する要素が必要である．そこで，スタイルシート中で演算子に対し，演算子順位を定義し，その順位の大小により括弧

の出力の必要性を判定するようにした．これによりもとのソースと出力されるソースで食い違いをなくすことができる．

- インデントをつける
  ソースコードを見て理解するためにはインデントをつけることは重要である．そのため，HTML へ変換する際に，プログラム構造の深さを計算し強制的なスペースを表示することによってインデントをつけるようにした．

これ以降 XSLT を用いたアプリケーションは，Apache XML Project[9] が提供している XSLT プロセッサである Xalan C++を用いる．Xalan はオプション指定で使用するスタイルシートを選択することができる．

4.3.3 使用例

ここでは，HTML へと変換するためのアプリケーションの使用例をしめす．

このアプリケーションの名前は”x2h” でコマンドラインから一つ目の引数はスタイルシートを，二つ目は出力先を指定することで実行できる．

```
% x2h ~/xsl/java.xsl ~/HTML
```

これをを実行することによって，実行時のカレントディレクトリ以下に含まれている XML 文書を HTML 文書に変換することができる．

また，HTML 文書であるので WWW ブラウザを用いることによって表示することができる．HTML 文書には変数やメソッドなどにおける宣言と参照の関係をハイパーリンクを用いることによって表現されているので，例えば，上記の図で “out” の宣言はいったいどこで行なわれているかを知りたいと思えば，“out” をクリックすることによって宣言元を参照することができる．

4.3.4 XML から HTML に変換 (深さ制限)

上記のアプリケーションによって XML 文書から Java ソースプログラムを HTML 形式で表現したものに変換することができるようになった．しかし，ソースコードの情報が得られるとはいえ，必ずしもすべての情報が必要とは限らない．そこで，情報を制限する一つの手段としてプログラム構造においてある一定の深さ以上のソースは表示しないということが考えられる．そこで，この深さ制限を与えるためのアプリケーションを実装した．

このアプリケーションは HTML 変換の延長であるといえる．HTML に変換する際，インデントを表示するため常にプログラム構造の深さを計算しその値を保持している．このこと

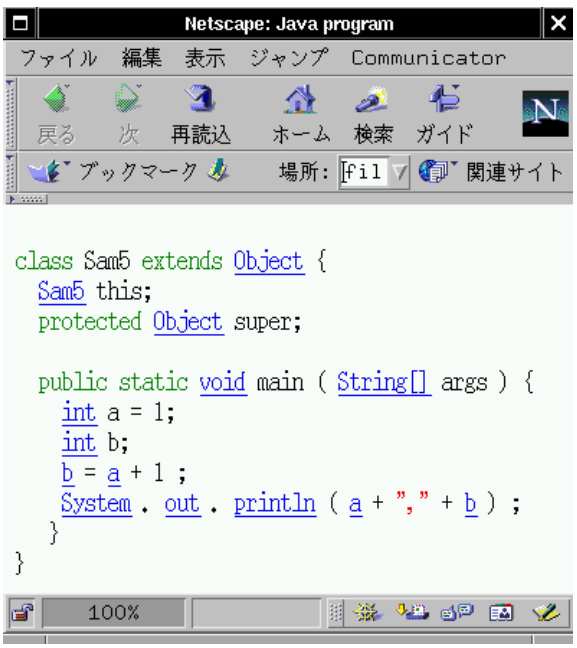

図 3: HTML での表示 1

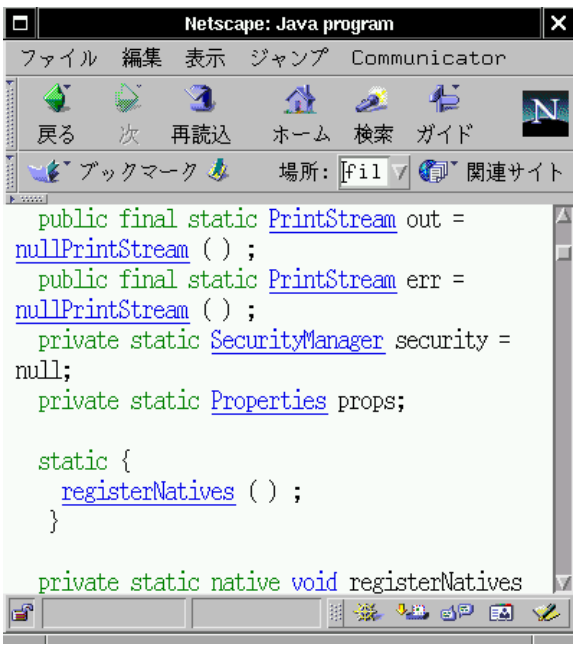

図 4: HTML での表示 2

から深さを制限して出力するためにはそのプログラム階層の深さを判定材料にして出力するかどうかを決定する．

ユーザには表示するソースプログラムの深さを指定すればよい．その深さを指定するためのスタイルシートを用意しており，その内容を指定することによって制限できる．また，隠されたいる部分をどのような形で表示するかを指定することもでき，ユーザはその表示をアレンジすることができる．

提供しているスタイルシートは次の通り．9 行目の <xsl:variable ··· > 中の”select” の数字で深さを指定する．また，<xsl:template ··· > 内の記述が隠している部分で表示される．ここを自由に記述することでユーザにあった表示が可能になる．

このスタイルシートを用いて java.lang クラスの Stirng を変換した．この変換された HTML 文書をブラウザで表示すると次のようになり，情報が制限されていることがわかる．

深さ制限のスタイルシート

```
<?xml version="1.0" encoding="UTF-8"?>
<!--xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml"
                version="1.0"
                exclude-result-prefixes="#default">

<!-- nest -->
<xsl:variable name="nest" select="2"/>

<!-- Output -->
<xsl:template name="Secret_">
<font color="red"><i>Secret!</i></font><br/>
</xsl:template>

</xsl:stylesheet>
```

4.3.5 XML データベースの変数置換

XSLT の機能として XML 文書を違う構造の XML 文書に変換できるというものがある．この機能を用いることによって，Java プログラム解析ツールを通すことなくデータベース化されている XML 文書を編集することができる．そこで，その一例として変数名の一括変換を行なうアプリケーションを実装した．

文字列を変換するだけであれば，エディタでも行なうことができる．しかし，それは変数の文字列を判定材料にしているだけでプログラム中での意味は考慮されていない．しかし，このデータベースの場合は，変数名には属性 id をもつことによって，同じ変数名であっても ID によって区別することができる．そこで，XSLT を用いたの変数置換は変数の名前だけでなくそれがもつ ID も判定の基準として変換を行なう．これにより変数名の置換によってプログラム構造に不具合が生じることはない．

このアプリケーションもプログラムの深さ制限と同様にユーザにスタイルシートを与えることによって実装している．そのスタイルシートは次の通りである．8 行目で変換対象となる変数の ID を指定している．9 行目でその変換対象である変数の名前，10 行目で変換後の名前を指定する．

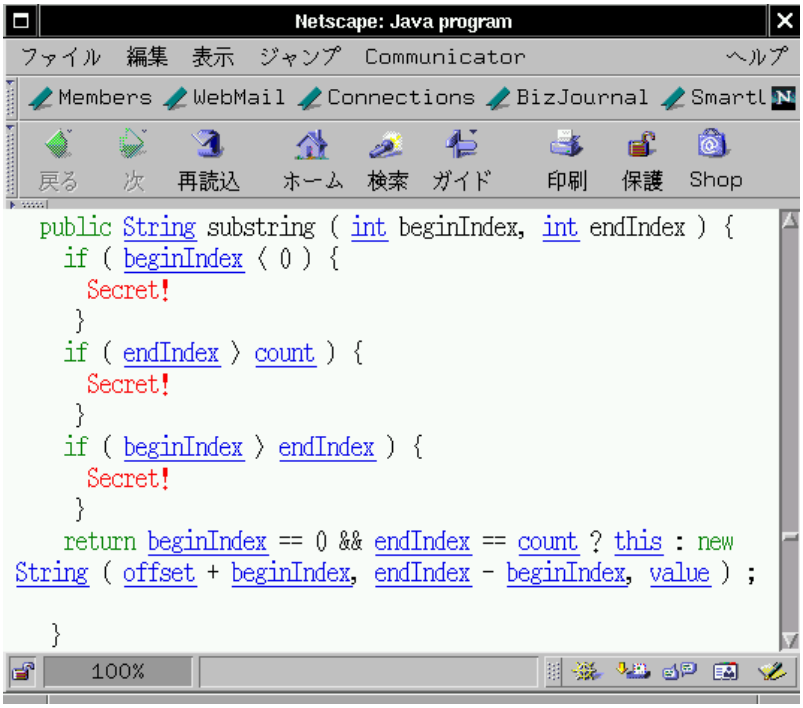

図 5: 深さを制限して変換した HTML 文書の表示例

変数変換のスタイルシート (一部抜粋)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"-->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml"
                version="1.0"
                exclude-result-prefixes="#default">

<xsl:variable name="target_id">0x34</xsl:variable>
<xsl:variable name="change_before">value</xsl:variable>
<xsl:variable name="change_after">after_value</xsl:variable>

</xsl:stylesheet>
```

このスタイルシートを用いることによって java.lang クラスの String を変換した．変換した結果を次に示す．この結果から ID が”0x34” である変数 value は after_value という名前に変換されたが ID が”0x4b17” である変数 value は変更されていない．このように XSLT のス

タイルシートを用いることによってプログラムの意味情報に影響を与えることなく編集を行なうことができるといえる.

変換前の XML 文書 (java.lang.String)

```
                    .
                    .
  <Variable modifiers="private" text="value" id="0x34">
    <type text="char[]" ref="0x9"/>
  </Variable>
  <Variable modifiers="private" text="offset" id="0x3b">
    <type text="int" ref="0x7"/>
  </Variable>
                    .
                    .
<Constructor modifiers="public" text="String" id="0x34e">
  <Variable modifiers="" text="value" id="0x4d17">
    <type text="java.lang.String" text_="String" ref="0x22"/>
  </Variable>
```

変換後の XML 文書 (java.lang.String)

```
                    .
                    .
  <Variable modifiers="private" text="after_value" id="0x34">
    <type text="char[]" ref="0x9"/>
  </Variable>
  <Variable modifiers="private" text="offset" id="0x3b">
    <type text="int" ref="0x7"/>
  </Variable>
                    .
                    .
<Constructor modifiers="public" text="String" id="0x34e">
  <Variable modifiers="" text="value" id="0x4d17">
    <type text="java.lang.String" text_="String" ref="0x22"/>
  </Variable>
```

5 あとがき

本研究では，Java プログラムの解析情報を XML を用いたデータベース化を行なった．これによって従来の Java プログラム解析フレームワークでは，解析情報は一時的なものであったがデータベース化を行なうことによって二次利用することが可能となった．また，データベース化を行なった結果．Java プログラムの解析手順の簡略化から解析効率が向上した．

XML によってデータベース化される対象は，プログラムの抽象構文木と意味情報を合わせた意味解析木である．XML は文書構造化言語であるため，意味解析木の情報を容易に表現することができた．また，定義した XML タグはユーザ側が理解することが容易になるように，タグ名で構文が把握できるように，タグの属性に情報をまとめるように，そして二次利用を見越してできるだけ生成規則が単純になるように定義した．

Java プログラム解析フレームワークと XML データベースとの間でデータをやりとりするため，意味解析木と XML 文書の相互変換を行なうライブラリを実装した．また，XML データベースの二次利用を行なうための応用アプリケーションを実装した．

今後の課題としては，意味情報以外の解析情報のデータベース化が必要である．Java プログラム解析フレームワークによって得られる解析情報には，意味情報以外にデータ依存関係や制御依存関係といった各種依存関係，またエイリアス関係などを含んでいる．このことからこれらの情報をデータベース化することは必要である．また，データベース化することによって二次利用できる情報が増え，さらなる効率の改善が期待できる．

謝辞

本研究に当たり，適切な御指導および御助言を頂きました大阪大学 基礎工学部 情報科学科 井上 克郎 教授に深く感謝致します.

本研究に当たり，適切な御指導および御助言を頂きました同 楠本 真二 助教授に深く感謝致します.

本研究に当たり，適切な御指導および御助言を頂きました同 松下 誠 助手に深く感謝致します.

本研究に当たり，適切な御指導および御助言を頂きました同 大畑 文明 氏に深く感謝致します.

最後に，様々な御指導および御助言を頂いた大阪大学 基礎工学部 情報科学科 井上研究室の皆様に深く感謝致します.

参考文献

[1] A. V. Aho, S. Sethi and J. D. Ullman: “Compilers : Principles, Techniques, and Tools”, Addison-Weseley, (1986).

[2] B. Stroustrup: “The C++ Programming Language(Third edition)”, Addison-Wesley, (1997).

[3] J. Gosling, B. Joy, and G. Steele: “The Java 言語仕様”, 村上 雅章訳.

[4] Martin Bryan: “SGML 入門”, 山崎 俊一監訳, アスキー, (1993).

[5] XML/SGML サロン: “標準 XML 完全解説”, 技術評論社, (1998).

[6] 柴田望洋: “C プログラマのための C++入門”, ソフトバンク, (1992).

[7] 丸山宏, 田村健一, 浦本直彦: “XML と Java による Web アプリケーション開発”, ピアソン, (1999).

[8] 村田 真: “XML 入門〜HTML の限界を打ち破るインターネットの新技術〜”, 日本経済新聞社, (1998).

[9] “Apache XML Project”, http://xml.apache.org/

[10] “Document Object Model (DOM) Level 1 Specification”, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/

[11] “dw:Java technology:JavaML”, http://www.ibm.co.jp/developerworks/java/001201/j_javaml.html

[12] “Extensible Markup Language(XML)”, http://www.w3c.org/XML/

[13] “GNOME”, http://www.gnome.org/

[14] “XML/SGML サロン”, http://www.utj.co.jp/

[15] “XSL Transformations (XSLT) Specification”, http://www.w3.org/TR/1999/WD-xslt-19990421

[16] “新・XML とオブジェクト指向”, http://www.ogis-ri.co.jp/otc/hiroba/technical/xml/index.html