XMLを用いた汎用的な細粒度ソフトウェアリポジトリの実装

吉田 一[†] 山本 晋一郎[††] 阿草 清滋[†]

本論文では，細粒度のソフトウェア構成要素を管理する細粒度ソフトウェアリポジトリを，XML文書によって実装する方法を提案する．XML関連技術が次第に整備されつつあるため，リポジトリをXML文書として実装することで，計算機環境や開発言語に依らず，CASEツール開発者はリポジトリを幅広く利用できるようになる．この細粒度リポジトリは，ソースプログラムの文，式といった構文情報に加え，コメント，タブ，空白といった字句情報までを管理することができる．またリポジトリはテキスト表現の拡張として，解析対象文書にタグの挿入のみを行う方式で生成される．このような単純なシステムによって，ソースプログラムの変更を行うCASEツールを含む多目的な用途に適用できる．Javaソースプログラムを対象としたモデルをCASEツール開発に役立つよう設計した後，リポジトリを生成し，実用規模のアプリケーションによってリポジトリの有用性を確認した．

A Generic Fine-grained Software Repository Using XML

Hajime Yoshida,[†] Shinichirou Yamamoto[††] and Kiyoshi Agusa[†]

In this paper, we propose the method of implementing fine-grained software repository which manages fine-grained software elements with an XML document. Since XML technologies are fixed gradually, by implementing the repository as an XML document, it depends neither on a computer environment nor a development language. A CASE-tool developer can use the repository widely. In addition to syntactical information, such as statements, and expressions, this fine-grained software repository can handle lexical information, such as comments, blank characters and tabs. And the repository is generated as extention of text representation by the system which performs only tag-insertion in the analyzed document. By such a simple system, it is applicable to the multiple-purpose use containing the CASE-tools which modify source programs. After designing the model for a Java source program to be useful for CASE-tool development, we have generated a repository and evaluate the effectiveness with applications of a practical scale.

1. はじめに

CASE（Computer Aided Software Engineering）ツールの作成には，対象となるテキスト文書の字句・構文に依存する解析器が必要である．しかし，個別に解析器を作成するには多大な労力が必要であり，ソフトウェア工学における大きな問題である．様々なCASEツールに共通して用いられる機能を提供するCASEツール・プラットフォームが存在すれば，解析器を各種のCASEツールの間で共有し，CASEツール間のデータ統合が可能となり，ツールの開発者はそのツールに本質的な機能の実現に専念できる．

CASEツール・プラットフォームが，様々なCASEツールの要求に応えるためには，モジュール単位の管理をはじめ，モジュールの内部構造，コメント，インデントなどの字句情報など，細い粒度の情報を管理する，細粒度ソフトウェアリポジトリを備える必要がある．またプラットフォームの利用性を高めるためには，特定の計算機環境やプログラミング言語に依存せず，より拡張性のある簡潔なシステムであることが求められる．

本研究の目的は，ツール・プラットフォームの中核となる汎用の細粒度ソフトウェアリポジトリをXML[1)]（eXtensible Markup Language）を用いて実現することである．汎用性に関しては次の3点すべてについて満たされることを目指している．

- 解析対象言語
- 利用環境
- 利用目的

XML文書を扱うための関連技術は，一般的な計算

† 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University
†† 愛知県立大学情報科学部
Faculty of Information Science and Technology, Aichi Prefectural University

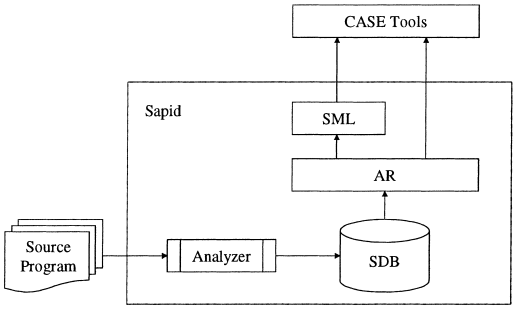

図 1 CASE ツール・プラットフォーム — Sapid
Fig. 1 CASE-tool platform — Sapid.

機環境およびプログラミング言語に対して次第に整備されつつある．細粒度リポジトリを XML 文書を用いて表現することで，いろいろなツールおよび場面から関連技術を活用してリポジトリを利用することができるようになり，利用環境に対する汎用性は確保できる．XML ベースの他のソフトウェア関連文書との連携も容易である．たとえば，XMI[2]（XML Metadata Interchange）との連携により，UML（Unified Modeling Language）ダイアグラムとソースプログラムとの整合性チェックなどが考えられる．

2. 既存の CASE ツール・プラットフォーム

我々の研究室では，これまでに CASE ツールの作成を支援するプラットフォームとして Sapid，Japid，StreamedSapid の開発を行ってきた．現状の問題点と XML 技術による解決策について考察する．

2.1 Sapid

Sapid[3]（Sophisticated APIs for CASE tool Development）は，ソフトウェアデータベース（SDB: Software Database），アクセスルーチン（AR: Access Routines），ソフトウェア操作言語[4]（SML: Software Manipulating Language）を主要な構成要素とする CASE ツール・プラットフォームである．概観を**図 1** に示す．

SDB は Sapid の基盤であり，要求されたソフトウェアをソフトウェアモデルに基づいて解析する解析器と，実体・関連情報を格納する関係データベースからなっている．SDB の各モデルにはソースプログラム構成要素の情報である実体クラス，実体クラス間の関連，ソースプログラム全体のテキストが含まれている．関連は，‘構成関連’ と ‘参照関連’ に区別できる．‘構成関連’ は構文規則の左辺・右辺に基づいた関係であり，それぞれのソースプログラム上の該当範囲は包含関係になる．‘参照関連’ は意味解析によって得られる関係，主に識別子の参照個所と宣言個所の関係である．

AR は SDB のデータにアクセスするための C 言語 API であり，CASE ツール作成者にとって必要かつ基本的な関数を提供する．AR には，データベーススキーマを得るためのクラス・属性名関数，データ属性値取得関数，関連取得関数，実体取得関数などがある．

SML はソフトウェアの開発・保守を行う際にソースプログラムに対して行われる，様々な変更操作や情報取得操作を簡潔で直感的に記述するための言語である．このほかに，ソースプログラムの制御依存関係，データ依存関係を扱う高度なビューを提供する API 群などが用意されている．

2.1.1 Japid

Japid[5] は Sapid パッケージに含まれる，Java に特化した CASE ツール・プラットフォームである．中核であるリポジトリは Java 言語の抽象化されたプログラム要素をモデル化したデータベースであり，Java クラスライブラリによるアクセスルーチン（AR）を介して，オブジェクト指向データベースと見なすことができる．

2.2 StreamedSapid

StreamedSapid[6] は，ソースプログラムの空白やコメントを保存した書き換えを行う CASE ツールを容易に実現するためのツール・プラットフォームである．ソースプログラムは解析器によって StreamCode と呼ばれるテキスト形式の細粒度リポジトリに変換される．これは Sapid と同等の細粒度の情報を保持している．

StreamCode はスタック型の仮想計算機により解釈可能な命令系列である．perl などのテキスト処理が得意な言語を用いて，解釈器を作ることで StreamCode の解釈実行を可能とする．このように CASE ツールを StreamCode の書き換えを行うフィルタとして実現することによって，パイプなどによりツール間の連携が可能である点が特徴である．

2.3 問 題 点

以上のシステムには次のような問題点がある．

Sapid および Japid は独自のデータベース形式でソフトウェア情報を格納しているため，CASE ツール製作者がリポジトリの情報を利用するには AR のようなアクセスルーチンが必要となる．多くの開発環境から利用できるようにするためには，アクセスルーチンを各開発環境に対して用意する必要があるが，現在のところ，アクセスルーチンが提供されているプログラミング言語は限られており，利用環境の汎用性が満たされていない．

多くの開発環境から基本的な情報の読み取りを可能

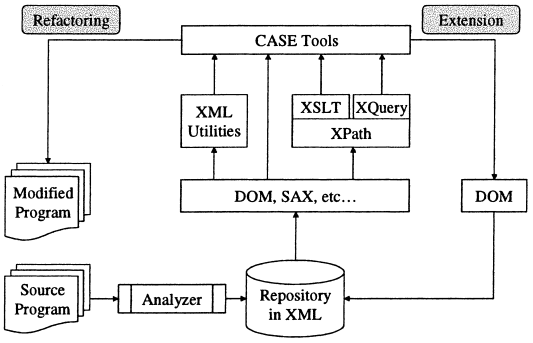

図 2 XML 技術を用いたツール・プラットフォーム
Fig. 2 CASE-tool platform using XML technologies.

にするには，プログラム要素の情報を一般の関係データベースに格納し，SQL による照会によって利用することも考えられる．しかし，SQL のクエリ機能を用いるだけではツリー構造を持つプログラム要素の情報を効率良く得られない．StreamedSapid は，テキストを扱える一般の開発環境から情報の読み取りが可能であるが，同じくテキスト処理のみでは情報を効率良く得られない．

このほか，Sapid SDB の各モデルにはソースプログラムの構成要素に対応するテキスト表記を参照するために，バイト単位のオフセット情報を用いており，また構成関連と参照関連が混在しているなど，ソースプログラムの変更操作の実現を困難にする要因がある[7]．モデルの問題については 3 章で解決方法を述べる．

2.4 XML 技術による利用環境の汎用化

XML は W3C（World Wide Web Consortium）によって規定された，拡張可能なマークアップ言語である．XML は仕様が簡潔であり，関連技術が豊富に用意されている．

Sapid AR のようなソースプログラムの木構造を扱うアクセスライブラリはツリー上のトラバース，検索，構成要素のテキスト表記の取得といった XML 関連技術の持つ機能と類似する部分が多く見られる．リポジトリの XML 文書化がなされれば，アクセスルーチンの機能を XML 関連技術の実装に委譲でき，同時に利用環境の汎用性も満たすことができる．

ソフトウェアリポジトリの実現に XML を用いた場合，CASE ツール・プラットフォームとして**図 2** のような利用形態が考えられる．

DOM[8]（Document Object Model）は，XML 文書のツリーを操作するための，プラットフォームとプログラミング言語に中立の API である．CASE ツールはメモリ上に構築されるオブジェクトのツリーを DOM インタフェースを用いて操作し，リポジトリに対する読み書きが可能である．これは Sapid の AR に近いインタフェースといえる．

SAX[9]（Simple APIs for XML）は，XML 文書を先頭から順番に解析し，イベントを通知するイベント駆動型の API である．CASE ツールはこのイベントを受け取り，内部状態を遷移させて XML 文書の解析を行うことができる．一般的に DOM より高速なため，検索などの処理を行う場合には有効である．これは StreamedSapid における解釈器に近いインタフェースといえる．

XPath[10]（XML Path Language）は，XML 文書内で任意の条件に合うノードを道のりをたどる形で探索し，選択するための表記規定である．XPath を用いることで，ソースプログラム要素を簡潔な記述で選択できる．組込み関数 id() は，特定の ID を持つノードを選択することができ，文書内のリンクをたどることもできる．

XSLT[11]（XSL Transformations）は，整形式 XML 文書の木構造を変換するための言語である．対象となるパターンを XPath を用いて記述し，テンプレートを適用して変換が行われる．CASE ツールは，XML 文書であるリポジトリを処理目的に応じた利用しやすい形に加工することができる．また，ツールの出力として HTML を用いる場合にも，XSLT による XML 文書の変換は特に有用である．

このほか，XML を扱える Web ブラウザが一般的になった場合，GUI を備える CASE ツールの実装手段として活用できる．

3. リポジトリの設計

リポジトリのスキーマについて述べ，モデル化の方針や将来の拡張ついての考察を行う．

3.1 細粒度リポジトリ XSDML

一般のソースプログラムには，読みやすさを向上させるために作者がインデントや改行個所を工夫している．SPIE[12]（Source Program Information Explorer）のようなレビュー支援ツールでは，リポジトリからソースプログラムのテキスト断片を完全な形で復元し，表示できることが望ましい．リポジトリがこのような字句情報を保持していない場合，リポジトリの情報からソースプログラムを再構築しなければならないが，インデントや改行個所は元に戻らない点が問題である．

XML 文書はプレーンテキスト文書の内容を明確に要素分けし，タグ付けの形で構造化した文書と見ることができる．XML 文書を用いて，解析対象のテキ

ストを完全な形で保持するには，対象文書のテキスト断片を直接タグ付けする方法が自然である．このように，ソフトウェア文書に対して直接タグを挿入した，XML 文書による細粒度ソフトウェアリポジトリを，XSDML（eXtensible Software Document Markup Language）と呼ぶことにする．各文書用のモデルは別に定義する．

タグ付けの結果，SDB に混在していた構成関連と参照関連のうち，構成関連に関しては，XML 要素間の親子関係として表現でき，モデルの見通しのよさを向上できる．また，解析文書のテキスト断片を要素のテキストノードとして割り当てることで，SDB に存在していた，オフセット情報によるテキスト断片の参照を排除でき，ソースプログラムに対する操作を比較的容易に実現できる．

XML 文書はさらに ID/IDREF 型の属性を付加することで文書内部におけるリンクの概念を表現可能である．SDB の参照関連のうちファイル内部に存在する関連は，IDREF 型の属性による内部リンクとして表現できる．

3.2 Java ソースプログラムのモデル JX-model

オブジェクト指向言語 Java（JDK 1.3）を対象として，ソースプログラム構成要素を抽象化したモデル JX-model 1.3.7 を設計した．含まれる構成要素を**表 1** に示す．

JX-model 1.3.7 では全部で 20 の ‘非終端要素’ と 7 つの ‘終端要素’ によって Java ソースプログラムを表している．細かい種別は要素の属性 ‘sort’ に記述している．CASE ツール開発者の視点によりプログラム要素を抽象化して表現し，構成される木構造が見通しよくなるよう工夫している．ファイル内部の関連の表現としては，識別子の要素に対して，その定義箇所へのリンクを表す属性 ‘defid’ が存在する．このほか，変数を参照する式の要素に対して，読み書きの属性 ‘read’, ‘write’ などが存在する．

モデル要素のうち，‘終端要素’ は字句情報であり，子ノードにソースプログラム断片であるテキストノードのみを持つ．‘非終端要素’ は構文情報であり，子ノードに他のモデル要素を持ち，テキストノードを持たない．

XML では DTD の指示によりロード時に文書が構造制約を満たしているかどうかを検証できる．しかし，XSDML が保持しているソースプログラムの妥当性を検証するには，DTD による構造制約の記述のみでは不十分である．プログラムの妥当性の検証は XSDML からソースプログラムを復元してコンパイラに通すことで実現されるため，DTD ファイルには厳密な制約の記述は行っておらず，各要素の子要素に含まれうるノードの指定のみを記述している．

表 1 JX-model 1.3.7 の要素
Table 1 Elements in JX-model 1.3.7.

要素名	Java ソースプログラム上の要素
File	ソースプログラムファイル全体を表す
Package	パッケージ宣言を表す
Import	パッケージインポート宣言を表す
Class	クラス定義を表す
Intf	インタフェース定義を表す
Ctor	コンストラクタ宣言を表す
Method	メソッド宣言を表す
SInit	static 初期化子を表す
Field	フィールド宣言を表す
Param	手続き宣言内のパラメータ宣言を表す
Local	手続き内のローカル変数宣言を表す
ExtdOpt	型宣言における派生元宣言を表す
ImplOpt	型宣言における実装インタフェース宣言を表す
ThrwOpt	手続き宣言における例外宣言を表す
Members	クラス・インタフェースの本体を表す
QName	パッケージ名と未解決の修飾名を表す
Type	型を構成するトークンを表す
Stmt	文を表す
Label	ラベル定義を表す
Expr	式を表す
ident	識別子を表す
literal	リテラルを表す
comment	コメントを表す
kw	キーワードを表す
op	演算子を表す
sp	空白文字とタブを表す
nl	改行文字を表す

3.3 ファイルをまたがる要素間の関連

リンクの表現に ID/IDREF 属性が利用できる範囲は文書内に限られるため，SDB の参照関連のうちファイル間にまたがる関連にはこの方法を適用できない．複数のソースプログラムの解析結果を 1 つの XML 文書内に配置し，1 ファイルの場合と同様に扱うことが考えられるが，この方法ではツリーが巨大化してしまい，実用的ではない．このため，XSDML 文書にはファイル内部の関連のみを記述し，ファイルをまたがる関連の表現には別のモデルを用意する．

XML 要素間の関係を表現する方法には XLink [13] および RDF [14]（Resource Description Framework）があげられる．XLink が主にハイパーリンクの指定および横断のための記述である一方，RDF ではより簡潔に要素間の関係について記述可能である．XSDML では細粒度の要素を扱うため，XLink によるリンクベース機能を用いる方法では，情報量が増大し，冗長であると考えられる．このためファイル間にまたがる関連は RDF を用いたメタ情報の形式として構成し，

具体的なRDFスキーマについての研究を進めている．

3.4 高度な情報の追加

制御依存グラフやデータ依存グラフのような高度な情報は，XSDMLを解析して得ることができる．XSDMLを解析対象とすることで，他の言語の解析に解析部を再利用しやすい．解析した情報は，新たなRDFモデルとして構築するか，DOMを用いてXSDMLの情報を拡張するなどして追加できる．

このように段階的に情報を追加すると，あらかじめ詳細な情報をリポジトリに納める場合と比べて処理時間がかかると考えられるが，リポジトリのサイズを抑えた状態で，個々のCASEツールは必要に応じて情報の加工ができる．利用目的に応じた情報量などのレベルを設定する方針である．

4. XSDMLの生成系

XSDMLを生成するソフトウェアは，ソフトウェア文書の解析を行う部分と，XMLのタグ付けを行う部分の連携によって行う．この作業の分割により，他のプログラミング言語や関連文書からリポジトリを生成する際には，共通の作業であるタグ付けを省略できる．

現在，これらのソフトウェアはSapid Home Page [15]で配布されている．

4.1 ソースプログラムの解析部

この処理では，ソースプログラムを解析し，XSDML要素に関する情報を出力する．ここでは，JX-modelについて述べる．

Javaソースプログラムの解析にはJapidを利用すると柔軟に行えるため，現在の版ではこれを用いて，Java言語によるコマンドとして解析部を実装している．実行にはJapidによって生成されたSDBが必要である．なお，字句情報が一部不足するため，Japidの解析部からもJX-model要素に関する情報を出力する．

例として，**図3**の簡単なJavaプログラムを入力すると，45個のJX-model要素情報が出力される．各要素情報は‘offset’，‘length’，‘weight’，‘name’，‘attributes’からなる．‘offset’と‘length’は，モデル要素のソースプログラム上の文字列範囲を表す．‘weight’は，範囲が重なった場合の親子関係を解決するための値で，大きいほど親ノードになる．‘name’はJX-model要素名であり，‘attributes’は要素の開始タグに対して付加される属性リストである．

4.2 XML要素のタグ付け部

この処理は，プログラム要素リストの情報を用いて，ソースプログラムにタグ付け作業を行い，XSDMLを生成する．タグ付け部はC++によるコマンドとして実装している．

```
1: public class sample
2:     extends samplebase {
3:     private int a;
4:     public sample(int a) {
5:         if (a<0) a=0;
6:         this.a = a;
7:     }
8: }
```

図3 解析対象Javaソースプログラム
Fig. 3 A Java source program for analysis.

```
<?xml version="1.0" encoding="ASCII"?>
<!DOCTYPE File SYSTEM "JX-model3.dtd">
<File id="s5087690753"><Class id="s5091885057" access="Publi..
</nl><sp>    </sp><kw>extends</kw><sp> </sp><ExtdOpt typefir..
</nl><sp>    </sp><Field id="s5100273666" typefirst="s510866..
</nl><sp>    </sp><Ctor id="s5104467969" access="Public"><kw..
</nl><sp>        </sp><Stmt id="s5121245187" sort="IFELSE"><..
</nl><sp>        </sp><Stmt id="s5121245186" sort="EXPR"><Ex..
</nl><sp>    </sp><op>}</op></Stmt></Ctor><nl>
</nl><op>}</op></Members></Class><nl>
```

図4 JX-model XSDML文書
Fig. 4 A JX-model XSDML document.

例として，図3のソースプログラムと，解析部で出力した要素リストを入力すると，**図4**に示すXSDMLを出力する．解析対象文書の空白，タブ，改行文字はそのまま保持され，構造と意味の情報が付加されている．このため結果の各行は解析対象文書の各行に対応する．

XSDMLの木構造を把握しやすくするには，インデントを行うとよい．図4のXSDMLをインデントし，空白文字を無視すると，**図5**のようになる．インデントを施すとXSDMLの構造が把握しやすいが，逆に解析対象文書における構成要素の位置情報は簡単に得られない．このため，位置情報を要素に付加するような拡張を行っている．この情報は終端要素に対して設定すれば十分である．

5. XSDMLの利用例と評価

5.1 テキストの復元

XSDML文書が保持しているソフトウェア文書を取り出すには，**図6**のXSLTプログラムを用いる．あるソフトウェア文書と，これからXSDMLを構成し，図6のプログラムによって復元したソフトウェア文書との間にはテキスト比較による差異は存在しない．このような復元性により，インデントや改行個所を元の状態に保ったまま，ソフトウェア文書に対する変更操

```
<sp/>
<Stmt id="s5121245186" sort="EXPR">
   <Expr id="s5125439489" sort="Assign">
      <Expr id="s5125439490" sort="DOT">
         <Expr id="s5125439491" sort="This">
            <kw>this</kw>
         </Expr>
         <op>.</op>
         <Expr id="s5125439492" sort="VarRef" write="yes">
            <ident defid="s5100273666">a</ident>
         </Expr>
      </Expr>
      <sp/>
      <op>=</op>
      <sp/>
      <Expr id="s5125439493" sort="VarRef" read="yes">
         <ident defid="s5100273665">a</ident>
      </Expr>
   </Expr>
   <op>;</op>
</Stmt>
<nl/>
```

図 5 インデントを施した JX-model XSDML 文書（ソースプログラム 6 行目）
Fig. 5 An indented JX-model XSDML document (line 6).

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="text"/>
   <xsl:preserve-space elements="*"/>
   <xsl:template match="/">
      <xsl:value-of select="File"/>
   </xsl:template>
</xsl:stylesheet>
```

図 6 テキストを復元する XSLT プログラム
Fig. 6 An XSLT program to restore the text.

作を行うことも可能である．

5.2 ブラウザを用いた利用例

Web ブラウザを用いたリポジトリの利用例として，あるメソッド内に含まれる，メソッド呼び出し式の情報を表示する簡単なアプリケーションを示す．実現には Microsoft XML Parser（MSXML）4.0 および Internet Explorer 6.0 を用いており，スクリプトおよび XSLT を内部に含む HTML のサイズは約 170 行である．

動作は，HTML 内のスクリプトが MSXML を起動して XSDML を読み取り，ユーザがコンボボックスから呼び出し元メソッドの識別子を選んで検索ボタンを押すと，スクリプトが XSLT を用いて結果を HTML に変換し，Dynamic HTML を使って表示する．**図 7** に実行画面を示す．

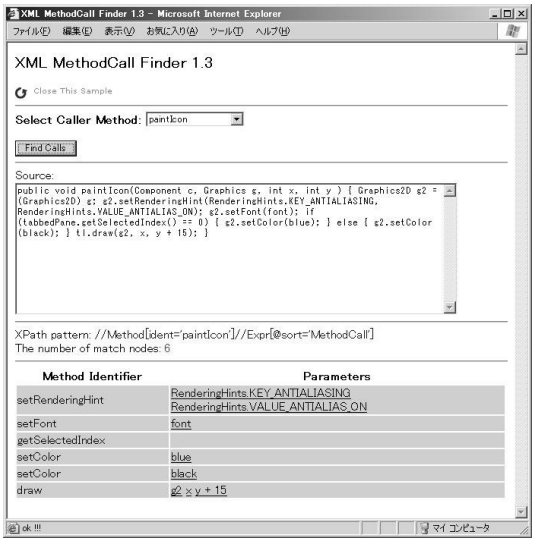

図 7 Web ブラウザを用いた XSDML の利用例
Fig. 7 An example of use for XSDML using Web browser.

表 2 生成されるファイルのサイズ
Table 2 Generated file size.

	SimpleExample	Java2Demo
解析対象文書 [行]	154	539
解析対象文書 [kB]	4.75	20.6
Japid SDB [kB]	30.8	178
XML 要素数 [個]	1,362	7,520
XSDML [kB]	35.0	194

表 3 処理にかかる時間
Table 3 Processing time.

	SimpleExample	Java2Demo
解析部 [ms]	1,380	3,300
タグ付け部 [ms]	50	140

5.3 空間・時間的効率の評価

JDK に含まれるサンプル SimpleExample.java と Java2Demo.java を例として，空間・時間的効率の評価を行った．

Java ソースプログラムおよびアスキー形式の Japid SDB，インデントを施していない JX-model XSDML のサイズはそれぞれ**表 2** のようになった．なお，Japid SDB はソースプログラムを含んでいない．

次に各処理にかかる時間を計測した．CPU: Intel Celeron 1.2 GHz，256 MB RAM のハードウェア，Java2 Runtime Environment 1.4.0_03 を用いて，コマンドの実行時間を計測すると，**表 3** のようになった．

結果より，XSDML のサイズは Japid SDB と同程度で，実用的であった．解析部では若干時間がかかっているが，新たに C++を用いて解析部を作成中であ

り，改善される見通しである．タグ付け部の時間効率は実用的である．

6. 関連研究

ソースプログラムの解析結果の表現にXML文書が用いられている研究をあげる．解析およびXML文書の生成に関して，これらの研究では，特定の言語処理系の利用または書き換えによって実現している．本研究では，利用目的および解析対象言語に対する汎用性を持たせるため，ソースプログラムをコンパイラとは別の側面から解析し，またXMLのタグ付けを行う部分を独立させるなど，他のプログラミング言語や関連文書への再利用を考慮している．

6.1 JavaMarkup

田中らによる文献[17]では，ソフトウェアの理解支援ツールの興味深いアイデアが提案されている．ここで用いられているXML文書は，粒度の細かさやプログラムのテキストを保存するなどの点で本論文で提案するXML文書と類似している．

しかし，田中らのXML文書は理解支援ツールの実現手段として位置付けられており，理解支援ツールについて紙面がさかれているため，文献にはスキーマの詳細などは説明されていない．また，時間・空間的効率についても触れられていない．

本論文では，汎用的なソフトウェアリポジトリをめざしており，リポジトリのスキーマとなるDTDについては，3章で議論し，モデル化の方針や情報量のレベルについても述べた．

6.2 JavaML

JavaML[16]は，Badrosによって考案された，Javaソースプログラムの XMLによる代替的な表記法である．Java言語構文に依存しない抽象化したDTDを定義することで，一般的なオブジェクト指向言語に対して適用できるようなモデルとなっている．クラス，メソッド，フィールド，文字定数，数値定数，コメントといったソースプログラムに含まれている細粒度のソフトウェア部品を，70種余りのXML要素としてモデル化している．

JavaMLのサイズは対応するソースプログラムの約4倍であり，Javaソースプログラムに変換するために，500行程度のXSLTプログラムが用意されている．しかし，ソースプログラムの字句情報は失われているため，プログラマが調整したインデントや改行位置は元の状態に戻らない．したがって，SPIEのような理解支援ツールの実現手段には向いていない．また，ソースプログラムとは異なる構造であるため，学習コストが比較的高い．一般的なオブジェクト指向言語向けにモデルを抽象化した結果，非オブジェクト指向言語や前処理を含む言語への対応は困難である．

6.3 ACML

権藤らによる文献[18),19)]では，XMLを用いたCASEツール・プラットフォームにおけるデータ統合のためのスキーマとして，ACML（ANSI C Markup Language）が考案されている．ACMLはANSI Cソースプログラムの構造と意味情報を表現するXML文書形式である．CASEツールの作成に役立つ解析情報を多く提供しており，実際にスライサを短期間で作成したことで，その有用性が示されている．ACMLでは，ソースプログラム上の表記によらず，同じ意味を持つ構成要素は同じXML要素として，50種余りのXML要素としてモデル化している．

ACMLにはプログラムのテキストは保存されていないため，SPIEのような理解支援ツールの実現手段には向いていない．また，ソースプログラムとは異なる構造であるため，学習コストが比較的高い．生成される情報はコンパイラが生成する記号表に近く，やや複雑である．このため含まれる情報量は多く，生成されるファイルサイズは大きい．文献によれば，Cソースプログラムから生成したACML文書が，解析対象文書の100倍を超えるケースもある．

7. おわりに

本論文では，汎用の細粒度ソフトウェアリポジトリXSDMLを提案し，Javaソースプログラムのモデルである JX-modelについて述べた．この方式ではXMLを用いて解析元テキストに対する意味付けを行い，テキスト表現の拡張としてのXML文書を提供する．このため，容易に解析元のテキストを復元することが可能であり，ソフトウェア文書の変更操作を行う各種のCASEツールからも活用できる．また，既存のWebブラウザ，XMLパーサ，XSLTを用いることで，XSDMLを扱うアプリケーションを容易に作成できることを示した．

今後の課題としては，3章で述べた，ファイル間にまたがる関連を表現するRDFスキーマの設計が残されている．依存解析モデルのような高度なビューを用意することも必要である．

参考文献

1) Bray, T., Paoli, J. and Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0, Second Edition, W3C Recommendation

(Oct. 2000).
2) OMG: XML Metadata Interchange. http://www.omg.org/technology/documents/formal/xmi.htm
3) 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 (1998).
4) 吉田　敦，山本晋一郎，阿草清滋：CASE ツール開発のためのソフトウェア操作言語，情報処理学会論文誌，Vol.36, No.10, pp.2433–2441 (1995).
5) Hachisu, Y., Yamamoto, S. and Agusa, K.: A CASE Tool Platform for an Object Oriented Language, *IEICE Trans. on Inf. Syst.*, Vol.E82-D, No.5, pp.997–984 (1999).
6) 吉田　敦，山本晋一郎，阿草清滋：ソースプログラムに対する変更操作が可能な細粒度ソフトウェアリポジトリの提案，日本ソフトウェア科学会 FOSE'98，pp.189–198 (1998).
7) 福安直樹，吉田　敦，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ソースプログラムの安全な変更，日本ソフトウェア科学会コンピュータソフトウェア，Vol.15, No.4, pp.78–81 (1998).
8) W3C: Document Object Model (DOM). http://www.w3.org/DOM
9) SAX Official Page. http://www.saxproject.org/
10) Clark, J.: XML Path Language (XPath) Version 1.0, W3C Recommendation (Nov. 1999).
11) Clark, J.: XSL Transformations (XSLT) Version 1.0, W3C Recommendation (Nov. 1999).
12) 大橋洋貴，山本晋一郎，阿草清滋：ハイパーテキストに基づいたソースプログラム・レビュー支援ツール，電子情報通信学会ソフトウェアサイエンス研究会，Vol.98, No.28, pp.15–22 (1998).
13) DeRose, S., Maler, E. and Orchard, D.: XML Linking Language (XLink) Version 1.0, W3C Recommendation (June 2001).
14) Manola, F. and Miller, E.: RDF Primer, W3C Working Draft (Jan. 2003).
15) Sapid Home Page. http://www.sapid.org/index-ja.html
16) Badros, G.J.: JavaML: A Markup Language for Java Source Code (2000). http://www.cs.washington.edu/homes/gjb/JavaML
17) 田中　哲，一杉裕志：ソースコード理解支援ツール — JavaMarkup，日本ソフトウェア科学会 SPA2002 (2002).
18) Gondow, K. and Kawashima, H.: Towards ANSI C Program Slicing Using XML, *Electronic Notes in Theoretical Computer Science*, Vol.65, No.3, Elsevier Science Publishers (2002).
19) 川島勇人，権藤克彦：XML を用いた ANSI C のための CASE ツールプラットフォーム，日本ソフトウェア科学会 SPA2002 (2002).
（URL は 2003 年 4 月現在）

(平成 14 年 10 月 3 日受付)
(平成 15 年 4 月 3 日採録)

吉田　一

1979 年生．2002 年名古屋大学工学部卒業．同大学院情報科学研究科情報システム学専攻修士課程在学中．ソフトウェアデータベース，ソフトウェア開発環境に関する研究に興味を持つ．

山本晋一郎（正会員）

1962 年生．1987 年名古屋大学工学部卒業後，同大学院に進学．1991 年同大学助手，1996 年講師．1998 年愛知県立大学情報科学部助教授．プログラミング言語処理系，ソフトウェアの形式的開発手法，ソフトウェア開発環境に関する研究に従事．近年は，細粒度のソフトウェア・リポジトリに基づいた CASE ツール・プラットフォームに関する研究を進めている．電子情報通信学会，日本ソフトウェア科学会会員．

阿草　清滋（正会員）

1947 年生．1970 年京都大学工学部電気工学第二学科卒業．1972 年同大学院工学研究科電気工学第二専攻修士課程修了．同博士課程へ進学．1974 年より同情報工学科助手．同講師，助教授を経て 1989 年より名古屋大学教授．現在，同大学院情報科学研究科教授．工学博士．専門分野はソフトウェア工学，ソフトウェア開発方法論，知的開発環境，ソフトウェアデータベース，仕様化技法，再利用技法，マンマシンインタフェース．電子情報通信学会，ソフトウェア科学会，IEEE，ACM 各会員．