

XMLを用いたANSI CのためのCASEツールプラットフォーム

川島 勇人 権藤 克彦
北陸先端科学技術大学院大学
情報科学研究科

概要

CASE ツールの開発は、各々が対象とするソフトウェアに応じた解析器を必要とするため、多くのコストがかかる。効率的な開発には、各々のツールで共通に用いられるデータの統合が有効である。そこで、我々はXMLに注目した。なぜなら、XMLは構造化文書のためのデータ記述フォーマットであるが、CASEツールのデータ統合にも多くの利点を持つからである。例えば、XMLの文書型定義(DTD)を使うと、プログラムの複雑な構造や関係をコンパクトに表現できる。本研究では、XMLを用いたCASEツールプラットフォームを構築し、これを基に、プログラムスライシングツールとクロスリファレンサを作成した。この実装実験では、各々の実現は開発者1人で、わずか約2週間ですみ、開発コストの削減を確認した。

1 はじめに

ソフトウェア開発の過程には、数多くのプロダクトが出現する。プロダクトとは、ソフトウェアを構成する種々の仕様図面や設計文書、プログラム、メモ、マニュアルなど、ソフトウェアを製品として仕上げるために必要となるあらゆる文書である。ソフトウェアの開発保守の過程は、その多くがこれらプロダクトの参照と変更の過程である [22, 28]。

これには、人手による作業では限界があり、困難である。そのため、コンピュータによる支援、つまりCASE¹ツールが必要となる。しかし、単体のCASEツール開発には、各々が対象とするソフトウェアに応じた解析器が必要で、ツール作成には多くのコストがかかる。また往々にして、これらのツールは開放的でなく、ツール間の連携を取ることが、念頭に置かれていない。よって、各々のツールに共通に用いられるデータを統合しなければならぬ。この問題を解決するために、CASEツールプラットフォームが提案、構築されてきた [25]。CASEツールプラットフォームとは、単なるツールの寄せ集めではなく、CASEツールに共通なデータや機能を提供する基盤となるソフトウェアシステムである。これにより、ツール開発コストの削減とツール間の連携が容易になり、統一的で効率的なソフトウェア開発が期待できる。

これまでも、データ統合に着目した技術は、研究開発されてきた。例えば、CDIF [4, 24] や PCTE [3, 23] は、その代表例である。しかし、これらの技術は未だCASEツールにおいて広く使われていない。

ここで、我々はXML [1] とその関連技術に注目した。なぜなら、XMLは構造化文書のためのデータ記述フォーマットであるが、CASEツールのデータ統合にも、多くの利点を持つからである。例えば、XMLの文書型定義(DTD²)を使うと、プログラム構文構造をXMLのエレメントの入れ子構造として、プログラム要素間の関係をID/IDREFリンクとして簡潔に表現できる。

そこで、XMLを用いたCASEツールプラットフォームを提案し実現する。上流工程を支援するCASEツールへのXMLの応用は存在するが(例えば、XMI [6])、下流CASEへの応用はまだほとんどない。我々は、XMLによる下流CASEのデータ統合を目指す。これにより、上流と下流の協調も期待できる。

本研究では、XMLを用いたCASEツールプラットフォーム構築の第一歩として、ANSI C³プログラムのみをターゲットにしたCASEツールプラットフォームを構築した。これを基に、プログラムスライシングツールとクロスリファレンサを作成した。この実装実験では、各々の実現は開発者1人で、わずか約2週間ですみ、開発コストの削減を確認した。また、開発で得ら

¹Computer Aided Software Engineering

²Document Type Definition

³ISO/IEC9899-1990

れた経験として、設計した DTD の改良や、ツール作成の際に生じる定型作業のライブラリ化など、検討すべき点を発見できた。

本論文の構成は次の通り。第 2 節で XML を CASE ツール開発に応用する利点を列挙し、第 3 節で目標とする開発環境と、実現した CASE ツールプラットフォームについて述べる。また、第 4 節で実装実験とその結果を示し、第 5 節で議論を、第 6 節で関連研究を述べる。第 7 節では、結論として、XML が開発コストを削減したことを述べ、将来への課題をまとめる。

2 CASE ツール開発の現状と XML 導入の利点

2.1 CASE ツール開発の現状

CASE ツール開発のコストは非常に高い。それは、たいていの場合、その CASE ツールの内部データを他の CASE ツールで使えないことにある。つまり、非開放的であるか、開放的であっても、そのデータが使いにくいいため、コスト高となる。例えば、GCC[8] の内部で処理されるシンボルや構文、型情報などは、他のツールで使用することが容易にはできない。また、仮にできたとしても、内部表現に依存するので保守性は悪くなる。

開放的で効率的に CASE ツール間でのデータ共有やデータ変換を可能にする技術を発見、開発することが大きな課題である。

2.2 ソフトウェア開発環境における統合技術

CASE ツールのための共通フォーマットのアイデアは、新しいものではない。ここでは、一般的な統合技術について述べる。開発環境に開放性を与えるのは、ツールの持つ機能と情報を取りまとめ、一定の共通性を設定する統合技術である。以下に開発環境における 4 種類の統合機能を挙げる [28]。理想的な開発環境は、これら 4 つの機能を同時に満たすものである。

- データ統合
仕様記述やスクリプト、プログラムなど、プロダクトデータの要素や構成、及び、その扱い方を共通的に規定する。

- 制御統合
ツールの起動・終了やデータアクセスなど、共通的なツールの動作と通信の方法を規定し、ツール間の連携・協調を図る。
- ユーザインタフェース統合
ツールのユーザインタフェースを共通化し、開発環境の使用性を高める。
- プロセス統合
ツールによって支援される開発プロセスを共通的に想定し、個々の開発作業・工程を繋ぐ。

このうち、我々は特にデータ統合に注目する。なぜなら、CASE ツール開発のコスト削減には、データ統合が最も重要だと、我々は考えているからである。その理由は大きく 2 つある。

1. データ統合はプロダクトの最も根幹を統合するので、開発コストに密接に関係する。例えば、改行コードが 1 つに統合されれば、異なる改行コードの処理コードが不要になり、開発保守が容易になる。
2. それにもかかわらず、データ統合は他の 3 つの統合に比べて遅れている。既存の開発環境の多くは、制御統合やユーザインタフェース統合しか提供していない。

2.3 既存のデータ統合技術 : CDIF と PCTE

この節では、現在までに構築されたデータ統合技術として CDIF[4] と PCTE[3] を挙げる。これらの技術は、データ統合に貢献してきているが、未だ CASE ツール開発に広く使われてはいない。これは、本質的にデータ統合が難しいからである。それゆえ、XML を含めて、様々な手法を試して、有用性を調べるのが、データ統合技術の進歩に必要である。

2.3.1 CDIF : CASE Data Interchange Format

CDIF は、CASE ツールやリポジトリの仕様ではなく、CASE ツール間のデータ交換に用いられる標準フ

フォーマットである。CDIF が扱うデータは主に上流工程で扱われる開発対象システムの仕様や構造の情報である。例えば、実体関連図やデータフロー図である。CDIF では、このような仕様図の意味情報と表示のための情報を区別して扱う。この区別により、必要な情報のみを CASE ツールの方で容易に選択できる。

しかし、下流工程への支援が遅れているため [24]、我々の目的には CDIF は適さない。また、XML への変換が試みられている [18]。

2.3.2 PCTE : Portable Common Tool Environments

PCTE はソフトウェアリポジトリの機能構成を明確に示した代表例である。PCTE は、ソフトウェア開発をサポートする環境の一部として、そこで扱われるデータの処理機能群に対するプラットフォーム独立なアクセスを提供する [23]。重要となる概念は、オブジェクトとリンク、属性、スキーマで、すなわち、実体関連モデルである。1992 年頃には製品も出回り、1995 年には国際規格 (翌年に JIS 規格) となったが、実用的には普及してない。これは、仕様の複雑さ、実装速度が遅いことなどが原因であると我々は考えている。

2.4 XML による CASE ツール開発支援

2.4.1 XML の適用とその利点

XML を CASE ツールプラットフォームに適用することで、データ統合が実現できる。なぜなら、DTD を決めることで、共通フォーマットが規定でき、DTD に適合した (妥当な)XML 文書であれば、ツール間でデータ交換ができるからである。この他にも、XML を CASE ツール開発に適用するには、次の利点がある。

- データ統合に不可欠なプロダクトのスキーマ定義を XML のタグとして簡潔に定義できる。例えば、プログラム構文構造は、XML のエレメントの入れ子構造として自然に表現できる。また、プログラム要素間の関係を ID/IDREF リンクとして表現できる。この表現は簡潔で扱いやすい。
- 妥当な XML 文書として扱われないが、整形形式な XML 文書として表現すれば、不完全なデータ (例えば、バグのあるコード) でさえも扱うことができ

きる。

- XMI(UML⁴図の XML 表現形式)[6] の標準化が進んでいる。下流 CASE ツールに XML を適用すれば、上流と下流の間のデータ統合が期待できる。XMI については、2.4.2 節で述べる。

また、次に挙げる XML の一般的な特長は、CASE ツール開発においても有効である。

- XML は、構造化データに加えてプレーンテキストとしての利点も持つ。そのため、人が読むことができるし、sed や grep、perl のようなテキスト処理ツールも XML 文書に適用できる。
- テキストデータは、改行コードの問題を除けば、バイト順のようなデータ変換の際に発生する問題とは無縁なため、異なったプラットフォーム間で XML 文書を容易に交換できる。
- XML パーサや、XSLT[11]、DOM[10] など、XML 文書を柔軟に処理する標準的なツールが多く存在する。
- Java を始め、多くの汎用プログラミング言語で操作が可能で、Web ブラウザなど、既存のユーザインタフェースが活用できる。

2.4.2 上流工程を支援する XMI

現在、XML によるソフトウェア開発の上流工程を支援する枠組みは確立されつつある。XMI[6] は、XML による UML のテキスト表現形式で、OMG⁵によって標準化が進められている。

XMI は、ツール間やデータベース間、アプリケーション間の UML 情報交換に役立つ。例えば、XMI によって、1つのビジュアル・モデリング・ツールから、他の設計ツール、アプリケーション、データベースへの UML 図の交換を可能にする。また、1つの開発ツールに縛られることなく、選択肢が拡大する。これにより、上流 CASE の開発コストが削減できる。

実際に XMI をサポートする CASE ツールも開発が進んでいる。例えば、Rational Rose[7] には、作成した UML 図を XMI に従った XML 文書に変換できる。

⁴Unified Modeling Language

⁵Object Management Group

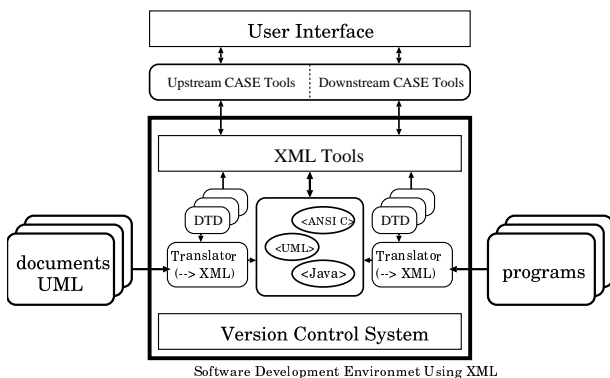


図 1: XML を用いた CASE ツールプラットフォーム

しかし、XML による下流 CASE の統合に関する研究はまだ少ない。下流 CASE の統合が実現すれば、上流のモデリングと下流のソースプログラムの双方向変換が可能になる。つまり、ソフトウェア開発の上流部と下流部間の大きなギャップを XML 関連技術を使うことで埋めることができる。我々が提案する「XML による下流 CASE のデータ統合」のゴールの 1 つは、この上流と下流の協調に役立つことである。

3 XML を用いた CASE ツールプラットフォームの提案・実現

3.1 XML を用いた CASE ツールプラットフォームの提案

我々がゴールと考える CASE ツールプラットフォーム (図 1) では、多種多様なプロダクトを XML 文書に変換する。例えば、仕様図面が UML 図なら XMI に、ソースコードが ANSI C コードなら ACML (3.2.1 節) に従って変換する。これにより、プロダクト間の一貫性を維持すると共に CASE ツールプラットフォームと各ツールの間に開放性を与えることを図る。

XML 文書化されたプロダクトは、既存の XML 関連技術を利用して作成、解析される。また、上流 CASE と下流 CASE の統合化が期待でき、協調作業が可能になる。情報管理も XML のまま行えば、プロダクトに含まれる様々な意味的情報を考慮に入れた管理システムとしての機能を発揮する。例えば、仕様図からソースコードまでの関連付けができ、一部のプロダクトで

変更が生じて、一貫して変更作業ができる。

特に、対象ソフトウェアについては、種々のプログラミング言語に対応し、各々、既存のプログラム、開発中のプログラム、エラーを含んだプログラムを扱えることを目指す。

3.2 実現した CASE ツールプラットフォームの概要

3.1 節で述べた CASE ツールプラットフォームは、大規模なため、一度にすべてを実現するのは難しい。そこで、本研究では XML を用いたソフトウェア開発環境を構築する第一歩として、ANSI C プログラムのみをターゲットにした CASE ツールプラットフォームを実現した (図 2)。XML が実用的に活用できるか確認するためには、我々は言語のサブセットではなく、フルセットをサポートすべきと考えている。我々のシステムは、ほぼフルセット⁶の ANSI C をサポートしている。

実現した CASE ツールプラットフォームの構成要素は、次の 3 つである。

- ANSI C 用に定義した DTD (ACML と呼ぶ)
- ANSI C プログラムをタグ付けされた XML 文書に変換するコンバータ XCI (3.2.2 節)
- アンパーサや、応用例として、プログラムスライシングツールとクロスリファレンサ

図 2 は、これら 3 つの連携を示す。まず、ANSI C プログラムは、XCI によってタグ付けをし、ACML 文書に変換する。その ACML 文書は XML パーサを通じて DOM や XSLT で、プログラムスライスを出したり、ソースコードを表示したりするために処理する。

3.2.1 ACML : ANSI C Markup Language

ACML は、我々が開発した ANSI C 用マークアップ言語で、データ統合のためのスキーマである。ANSI C コードは、ACML の DTD に従いタグ付ける。ACML の DTD は紙面の都合で省略する ([5] より入手可能)。

⁶現在は前処理命令 (例えば、`#include`) やライブラリ関数 (例えば、`signal`)、システムコール (例えば、`fork`) をサポートしていない。

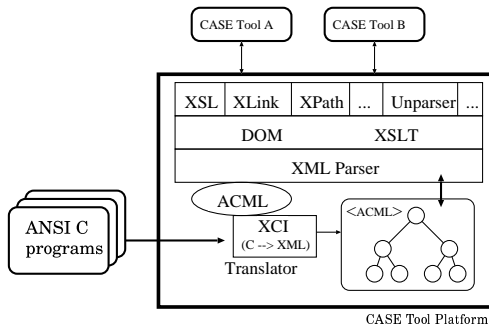


図 2: ANSI C のための CASE ツールプラットフォーム

DTD は、直感的には拡張 BNF 記法に属性を与えたものである。そのため、ANSI C 構文規則 [27] をより簡単に表現できる。例として、次の関数を挙げる。

```
int foo (int a, int b)
{
    if (a < b) return a;
    else return b;
}
```

これは XCI により、次の ACML 文書へ変換される⁷。

```
1 <function_definition id="X8086760">
2   <int/>
3   <declarator rhs="pointer_null">
4     <declarator rhs="func_new">
5       <declarator rhs="id">
6         <identifier ref="X8086760">
7           foo </></>
8         <parameter_declaration rhs="dec">
9           <int/>
10          <declarator rhs="pointer_null"
11            id="X8084e28">
12            <declarator rhs="id">
13              <identifier>
14                a </></></></>
15          <parameter_declaration rhs="dec">
16            <int/>
17            <declarator rhs="pointer_null"
18              id="X8085078">
19              <declarator rhs="id">
20                <identifier>
21                  b </></></></></>
22          <statement rhs="compound">
23
24      <!-- if-else 文 -->
25      <statement rhs="if-else">
26
27          <!-- 条件分岐部 -->
28          <expression rhs="less">
29            <expression>
```

```
26   <identifier ref="X8084e28">
27     a </></>
28   <expression>
29     <identifier ref="X8085078">
30       b </></></>
31
32   <!-- THEN 部 -->
33   <statement rhs="return">
34     <expression rhs="identifier">
35       <identifier ref="X8084e28">
36         a </></></>
37
38   <!-- ELSE 部 -->
39   <statement rhs="return">
40     <expression>
41       <identifier ref="X8085078">
42         b </></></></></>
```

ACML は ANSI C 構文を、XML エLEMENT の入れ子構造として表現する。例えば、23 行目以降が if-else 文を示している。この 23 行目の statement には、3 つの子ELEMENTがあり、各々、24 行目の expression 以下が条件式部を、31 行目の statement 以下が then 部、35 行目の statement 以下が else 部を示している。また、プログラム間の要素関係は、ID/IDREF 属性を使って表現する。例えば、26 行目の identifier の属性 ref が、a の定義部である 10、11 行目の declarator を指している。

また、ANSI C プログラムには、条件分岐部以外で、明示的に制御フローを変えることがある。例として、次のプログラム断片を示す。

```
for (i = 0; i < 10; i++) {
    for (j = 0; j < 5; j++) {
        if (buf[i][j] < 0) goto EXIT_LOOP;
    }
}
EXIT_LOOP:
```

ACML では、この制御フローの明示的な変化について、ID/IDREF 属性によりリンクを張ることで、次のように表現する。

```
<statement rhs="for">
  <!-- for 文の最初にある 3 つの式は省略 -->
  <statement rhs="compound">
    <statement rhs="for">
      <!-- for 文の最初にある 3 つの式は省略 -->
      <statement rhs="compound">
        <statement rhs="if">
```

⁷紙面の都合により、属性については、一部の 'rhs' と 'id'、'idref' 以外は省略した。また、全ての閉めタグは、</> に短縮した。

```

        <!-- if 文の条件式は省略 -->
<!-- goto 文 -->
    <statement rhs="goto"
        goto_ref="X80899c8"> <!-- ID -->
        <identifier>
            EXIT_LOOP </></></></></></></>
<!-- ラベル付き文 -->
    <statement rhs="label"
        id="X80899c8"> <!-- IDREF -->
        <identifier>
            EXIT_LOOP </></>

```

上の例では、goto 文の行き先 ('goto_ref') を、ID 値 'X80899c8' で判別することによって、適切なラベル付き文への参照を与えている。これと同様に、case 文や default 文、break 文、continue 文も表現することができる。

このように、ACML は XML が持つ柔軟な表現力を活かし、プログラムの静的意味情報を表現する。つまり、抽象構文木やシンボル、型に関しては、XML エレメント構造を、定義・参照間の関係や制御フローに関しては、ID/IDREF を用いて表現する。このため、ACML は静的スライサーやクロスリファレンサ、静的テストケース生成器のような静的にプログラムを解析する CASE ツール開発で力を発揮できることが期待できる。実際に ACML は、プログラムスライシングツールとクロスリファレンサの実験的実現において有用であることを 4 節で示す。

DTD のエレメントの粒度には議論が必要である。ACML は細粒度であり、例えば、識別子も 1 つのエレメントとして表現する。その一方、Badros[2] は、Java 用のマークアップ言語として Java Markup Language(JavaML) を提案した。Badros は、パースする際に用いる文法は、冗長すぎるので適切でない主張した。Badros の目標の 1 つは、オブジェクト指向プログラミング言語で、共通に使われるマークアップ言語を開発することである。

多種多様なソフトウェアプロダクトに応じて、適切な粒度は異なる。上流工程での種々のプログラミング言語をまとめようとする要求には、JavaML のような粒度の粗い DTD が有用であるが、下流工程では、ACML のような細粒度の DTD が有用である。

3.2.2 XCI : Experimental C Interpreter

XCI[5] は、'--xml' オプションを付けて起動すると ANSI C プログラムを ACML 文書に変換する。'--xml'

オプションを外すと、XCI は ANSI C インタプリタとして動作する。XCI は、約 14,000 行の ANSI C プログラムから成り、現在、Windows2000 上の Cygwin1.3.6 と Solaris8 の下で動作する。XCI の最初のバージョンを実現するのに、約 3ヶ月を費した。

XCI の内部データの設計が、ACML を定義するきっかけとなった。開発の当初、プログラマが内部の抽象構文木データ構造に直接アクセスできる API を持つように XCI を設計した。これは Sapid[25] の方法に類似している。しかし、我々の意見では、そのような API はプログラマが習得するのに多大なコストがかかるし、XCI の内部実装に依存しやすい。また、データ統合を考慮すると、API を通じて内部データにアクセスするよりも、内部データのための共通データフォーマットを提供する方が望ましい。そこで、ACML を導入し、'--xml' オプションを付けて起動するとき、ACML 文書を出力するように XCI の設計を変更した。

4 CASE ツール作成実験

XML を用いた CASE ツールプラットフォームを基に、実際に CASE ツールを作成した。本研究では、CASE ツールにおいて基本的、かつ必要性の高い静的解析ツールであるプログラムスライシングツールとクロスリファレンサを実験的に実現した。どちらもプログラム要素間の関係を処理するため、CASE ツールプラットフォームの有効性を計る例題として適切である。この実装実験では、各々の実現は開発者 1 人で、わずか約 2 週間ですみ、開発コストの削減を確認した。

4.1 スライシングツール

4.1.1 プログラムスライシングの概要

プログラムスライシングとは、プログラムの中のある変数に影響を与える全ての文を抽出する技術である。これは、プログラムのデバックやプログラムの理解支援に非常に役に立つ技術である。1982 年に Weiser[9] が考案して以来、現在でも活発な研究が行われている。

しかし、多くは小さな言語が対象で、実際のプログラム言語を対象にした研究は少ない。フルセットに近い

もの、例えば、Wisconsin Program Slicing Tool[14]、Unravel[15] は、XML を用いていない。本研究では、ANSI C 言語のフルセットに対応するスライシングを目指している。

4.1.2 スライシングツールの実現

現在、制限された ANSI C 言語を対象に、Weiser のプログラムスライシングを行う CASE ツールを実現した。XML を用いた CASE ツールプラットフォームを使用することで、プログラム中の条件を満たす変数の抽出と必要な文の抽出は比較的容易だった。DOM[10] を用いたことも開発を容易にした。このツールは約 2,000 行の Java プログラムで、作成期間は、開発者 1 人で約 2 週間を費やした。その仕様を以下に示す。

- スライシングの基準として、任意の文の任意の 1 変数が指定できる。
- if-else 文、および while 文を持つプログラムから実行可能な必要部分を抽出する。switch-case 文、goto 文、for 文は扱わない。
- ポインタ、配列の解析、および関数間を跨る変数の解析は行わない。

例えば、次のファイルの行数と語数、文字数を計算する関数を対象にスライシングを行ったところ、行数部、語数部、文字数部のみを取り出せた。

<pre> 1 iw = 0; /*in a word*/ 2 nl = 0; 3 nw = 0; 4 nc = 0; 5 c = fgetc(stdin); 6 while (c != EOF) { 7 nc = nc + 1; 8 if (c == '\n') 9 nl = nl + 1; 10 if (isspace(c)) 11 inword = 0; 12 else if (iw == 0) { 13 inword = 1; 14 nw = nw + 1; 15 } 16 c = fgetc(stdin); 17 } 18 printf("%d\n", nl); 19 printf("%d\n", nw); 20 printf("%d\n", nc); </pre>	<pre> 2 nl = 0; 5 c = fgetc(stdin); 6 while (c != EOF) { 8 if (c == '\n') 9 nl = nl + 1; 16 c = fgetc(stdin); 17 } 18 printf("%d\n", nl); </pre>
---	--

元のプログラム

スライス結果 (18, nl)

結果として、粗い比較にはなるが、XCI の構文解析、静的意味解析に費した労力 (2 人月) に比べると、大幅に少ない労力 (0.5 人月) で済んだので、この種の小さな CASE ツールの作成においては、開発コストが削減でき XML を用いる有用性を確認した。

4.2 クロスリファレンサ

4.2.1 クロスリファレンサの概要

クロスリファレンサとは、ソースプログラムの構成要素間にリンクを張り、その関係を明らかにするためのツールである。例えば、ソースプログラムの中から、指定した関数の定義部分と参照部分を見つけ出したり、複数のサブディレクトリや、プログラムから構成される大規模ソフトウェアを扱い、ファイル名からそのソースファイルへの参照を可能にする。例えば、既存のツールには、GLOBAL[16] や cxref[17] がある。

4.2.2 クロスリファレンサの実現

ACML と XCI を用いて、簡単な参照機能を持つ ANSI C プログラム用のクロスリファレンサを実現した。これは、フルセットの ANSI C 言語に対応している。表示は、既存の Web ブラウザを使用することにした。そのため、ソースプログラムを読むために HTML への変換が必要であった。ACML から HTML への変換は XSLT[11] で行った。これは、DOM でも可能だが、XSLT の利便性を探るために、今回は XSLT を活用した。スタイルシートは、約 2,000 行で、作成期間は開発者 1 人で約 2 週間を要した。その主な仕様を以下に示す。

- ソースプログラム中の関数呼び出し部分、変数や型の出現部分から、各々の定義部分への参照
- ファイル名のリストアップと、そのソースファイルへの参照
- 関数名、大域変数名、ローカル変数名、構造体と enum のタグ名、メンバ名のリストアップと、各々の定義部分への参照
- 自動整形機能 (ACML には、表現情報がないため)

実際に、ANSI C プログラム (GLOBAL[16] のソースコード) を表示したが、色分け、リンク先が正しく処理され、ACML から HTML への変換が成功したことを確認した。図 3 は HTML への変換結果の表示例である。

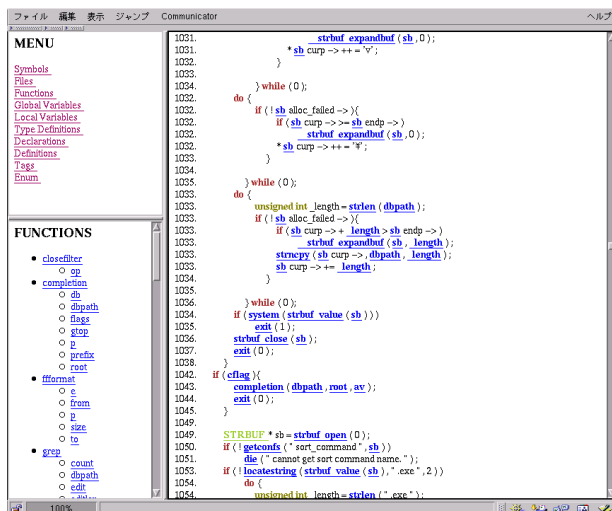


図 3: Web ブラウザによる表示例

結果として、スライシングツールと同様、この種の小さな CASE ツールの作成においては、開発コストが十分に削減でき、XML を用いる有用性を確認した。

5 議論

5.1 XML を用いる有効性

5.1.1 開発コスト削減の要因

実際のツール作成は、ACML と XCI、既存の XML 関連ツールの連携の下で行った。実際に、スライシングツールとクロスリファレンサの作成に費した時間は、各々 1 人で 2 週間ずつで非常に短期間ですんだ。この実装実験から、我々は、次の 4 点が開発コストの削減に貢献したと考える。

- XCI で対象ソフトウェアの構文解析と静的な意味解析まで完了しているため、目的のツールの本質的な機能の実現に専念できた。
- データのやり取りが、XML である ACML で行われたため、XCI から特別なデータ変換をせず、容

易に DOM や XSLT と連携がとれた。

- DOM や XSLT は、公開された汎用的な技術であるため、技術移転コストが低かった。
- ACML がデータ交換の簡潔な仕様として機能したので、プログラマ間のやり取りを少なくできた。実際に XCI の開発者と CASE ツール開発者の間のやり取りは、ACML を定義した DTD のみで、ほとんどすんだ。

これらは、一般の XML によるアプリケーション開発、例えば、企業間でのデータ交換を目的としたビジネスアプリケーションの開発において、期待できる効果と類似する。しかし、これらの効果が CASE ツール開発においても確認できたことは、重要である。一般のビジネスアプリケーションで扱うデータは比較的単純な構造で多量のものであるのに対し、本研究で扱うデータは複雑な構造で多量のものである。つまり、従来の XML によるアプリケーション開発では、扱うことが少ない複雑で多量のデータに対しても、XML の有効性が確認できたのである。

5.1.2 データ変換形式としての妥当性

実装実験を通じて、ACML を基にしたデータ交換が容易であったことを確認した。つまり、CASE ツール間のデータ変換形式として XML を用いることは有効である。特に、XML がデータ変換形式として優れている点は、XML が多種多様なデータ構造 (例えば、木構造、グラフ構造、表構造) をタグ付けによって容易に表現できることである。このため、プログラムのモデル化には、XML が適しているのである。ACML では、次に示す方法で ANSI C プログラムの静的意味情報を表現する。

- 木構造 (エレメントの入れ子構造)
構文情報、型情報
- 表構造 (エレメントのリニア構造)
シンボル情報
- グラフ構造 (ID/IDREF 属性によるリンク)
 - 定義・参照間の関係

- 制御フロー (例えば、goto 文)
- 再帰的型宣言 (例えば、リスト構造)
- シンボル情報部から構文情報部、型情報部への参照

5.2 ACML の改良の余地

ACML は構文規則に忠実に表現しているため、プログラミングスライスとクロスリファレンスに必要な情報も一通り備わっていた。そのため、比較的容易に各々のツールを作成することができた。また、このツール作成の経験から得られた成果として、現時点の ACML の改良の余地を発見することができた。以下で、その主なものを示す。

5.2.1 エlementノードの文脈のための情報とリンク

ACML が表現する木構造には、幾つかの改良点があった。これらは、設計不足というよりも、実装を通じて初めて分かることである。DTD の設計には、多くのトレードオフがあるので、実装実験は重要である。

- エlementノードの文脈

現在の ACML には、Element の文脈、特に、その Element が何番目の兄弟であるかの情報が不足している。特に下方から上方へ解析を行う場合は、親に戻って子供を数えなければ知ることができない。例えば、図 4 で、ある statement が then 部か、else 部かは、その statement(B,C) の属性には書かれていない。木構造が大きくなれば、数えるのは面倒な定型作業である。これは、データ統合として ACML に含めるべきか、アプリケーションで対処すべきかなど検討中である。

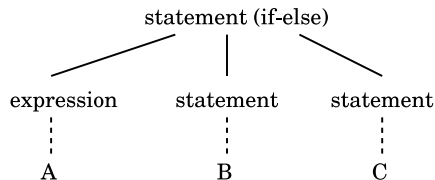


図 4: 3.2.1 節の例を木構造で表現

- リンク

ACML のリンク (ID/IDREF) による情報については、シンボル情報部から構文情報部と型情報部へは、十分に関連付けができています。しかし、現在の ACML では、その逆向きのリンクがない。これも、ACML に含めるべきか、アプリケーションで対処すべきかは検討中である。

5.2.2 前処理に関する情報

前処理系は、ANSI C とは独立した文法 (例えば、マクロ定義や条件付コンパイル)[27] を持つ。XCI は前処理後の ANSI C プログラムにタグ付けするため、現段階ではマクロは展開されたまま復元不可能である。これでは、ソースコードに対して完全には操作できないし、解析結果もソースコードに完全には反映されない。マクロ定義を復元する手法を検討する必要がある。

これは、我々だけが抱える問題ではなく、ANSI C ツールにとって一般的な問題である。Sapid では、部分的に解決している。

5.2.3 字句情報 (空白、コメント、インデント) の保存

ACML 文書をアンパースすると、元のプログラムと意味的には等価であるが、見かけは等価ではない。現在の ACML では、字句情報を保存してないからである。プログラミングスタイル [29] は、ソフトウェアを理解する上で貴重な情報となり、プログラムの意図も明確に表現する。字句情報の保存は、技術的には可能である。しかし、現在、その保存方法について、ACML のデータとして保存するか、あるいは、ACML とは独立して保存するかなど、その実現方法を検討中である。

5.2.4 DTD の決定性問題

現在の ACML を定義する DTD は、XML1.0[1] の仕様には合致しているが、推奨事項に反する Element 宣言をしている。XML1.0 では SGML⁸ との互換性を考慮して決定的内容モデルを推奨しているが、ACML 用の DTD では決定的内容モデルになっていない。そのため、ACML 用の DTD では、正確な妥当性の検証ができない場合がある。しかし、制限のない拡張 BNF

⁸Standard Generalized Markup Language

による DTD 表現は、簡潔で直感的に理解できる。仮に決定的内容モデルだけで、ACML を定義しようとすると、エレメントの数を大幅に増やさなければならず、DTD を複雑にするだけである。それゆえ、現在の ACML の表現の方に利点がある。

また、他のスキーマ言語 (例えば、RELAX[13]) は、非決定的内容モデルの記述を認めているものもある。これについては、十分に検討する必要があるが、次の点においては、DTD の方に利点があることから、我々は、DTD に対して積極的である。

- SGML で使われていた経緯から、成熟した技術で長所と短所が明確である。
- 安定ツールが、他のスキーマ言語よりも DTD の方を多くサポートしている。
- 文書規則をコンパクトに表現することができる。
- データ型はないが、エレメントの入れ子構造を簡潔に表現できる。

5.3 ACML に対する操作・編集

5.3.1 DOM の利便性

DOM[10] は、木構造を基に直感的に操作・編集することができるので、習得が早く使いやすかった。しかし、DOM には低レベルな操作・編集しか定義されていない。これだけは明らかに足りないものがあり、複雑な操作をするには、既存のライブラリを組み合わせなければならない。例えば、任意のエレメント名と属性値を基に、特定のエレメントを検索するような複雑な操作は、エレメント名を参照する関数、属性値を参照する関数、木構造を適切に走査する関数などを組み合わせて定義しなければならない。このため、DOM による操作では、冗長な作業が生じることが多い。

5.3.2 XSLT の利便性

XSLT[11] の基本的な処理はパターンマッチングで、XPath[12] を使って XML 文書内の検索を容易に行える。例えば、5.3.1 節の例 (エレメント名と属性値から特定のエレメントを検索する) は、“//elementName[@attribute='name']” と記述するだけですむ。XSLT は、関数型プログラミングの概念

を基にしているため [19]、経験があれば、非常に簡単に扱うことができる。しかし、スタイルシート自体、XML 文書なのでタグ付け作業が必要で、これはあまりに冗長的である。また、XSLT は関数型プログラミング特有の高階関数が使えないし、リストのようなデータ構造もない。このため、XSLT の能力だけでは、プログラムスライシングのような複雑な依存解析をするのは困難がある。

5.3.3 ライブラリの必要性

上で述べたが、DOM や XSLT を利用する場合、類似するコーディングを何度も行なわなければならない冗長さが生じるため、ライブラリを構築する必要がある。実装実験を行った結果、ツール開発の際には、非終端ノードからノード検索や属性値の参照などの定型作業が生じる。ある程度複雑な作業には、その定型作業を含むライブラリがあると便利である。例として、次の操作を挙げる。

- 特定の識別子を持つステートメントをすべて抽出する。
- 特定の識別子を含んだ最深部のステートメントを抽出する。

また、DOM と XSLT は、各々、一長一短であるため、これらの協調作業による処理は望ましいことである。この協調作業を促進するようなライブラリも必要である。これを踏まえ、XML を用いた CASE ツール作成に適したライブラリを開発中である。

一方、ライブラリでなく ACML 専用ツールを用意することで、ACML に特化した複雑な処理を行なうことも可能であるが、この場合、専用ツール作成のための新たなコストが発生すると共に、XML による汎用性を失う可能性がある。上述の DOM や XSLT を基にしたライブラリであれば、コストを抑え、XML による汎用性も保持することができるため、現時点では、ライブラリの開発を重視している。

この ACML 専用ツールに関連して、最近注目を集めている XML 技術にデータバインディング [20, 21] がある。これは、ある特定のスキーマ (例えば、ACML) から Java クラスを生成し、そのスキーマに準拠した XML 文書のパースや検証、データ構造の構築、編集

作業を行なう。これにより、スキーマによって決まるデータ構造を考慮したプログラミングが期待できる。このように、一般にXMLを扱う開発者の間で「スキーマに特化した処理」を可能にする技術の開発が進められている。

6 関連研究

6.1 Sapid

Sapid[25]は、ANSI CのためのCASEツールプラットフォームである。与えられたANSI Cソースコードに対して、Sapidは、構文構造と静的意味の情報をI-modelと呼ばれるフォーマットとしてファイルに格納する。Sapidは、独自で開発したソフトウェアデータベースとアクセスルーチン、ソフトウェア操作言語から構成され、制御統合を実現している。元々Sapidは、XMLを用いてないが、現在、XMLに基づいた機能を組み入れようとしている [26]。

6.2 JavaML

JavaML[2]は、Java言語のためのマークアップ言語の1つである。JavaMLは、Java特有の構文から独立してJavaプログラムをモデル化しようと試みている。この抽象化により種々のオブジェクト指向言語を一定のフォーマットとして記述できる可能性を持つ。ソフトウェアプロダクトによっては、JavaMLのような粗い粒度のモデルが適当な場合もあるが、下流工程を支援するには力不足である。

7 おわりに

我々は、多種多様なソフトウェアプロダクトを統一的に処理することを目指し、XMLを用いたCASEツールプラットフォームを提案した。本研究では、特に下流CASEのデータ統合に注目し、ANSI CプログラムのみをターゲットにしたCASEツールプラットフォームを実現した。これは、(1)ANSI C構文規則を忠実に表現したACML、(2)ソースコードをACMLに変換するXCI、(3)既存のXML関連技術、から構成される。

実装実験として、このCASEツールプラットフォーム

ムを基にプログラムスライシングツールとクロスリファレンサを作成をしたところ、各々1人で約2週間という非常に短い期間で作成がすんだ。この実験から、この種の小さなCASEツール作成では、CASEツールプラットフォームにXMLを用いる有効性が確認できた。しかし、定量的な評価を行っていないこと、実験対象が小規模であったこと、実験事例が少ないことから有効性が十分に示せたわけではない。さらに実装実験を行う必要がある。

今後の主な課題を以下に示す。

- より制限の少ない静的スライシングツールの実現
- 動的解析に対応するACMLの拡張
- 前処理命令や字句情報の保存への対応
- ACMLを使った他のCASEツールの作成 (例えば、テストケース生成器)
- ACML用の版管理システムの構築
- ACMLとXMIの連携手法の構築
- JavaやC++など他の言語へ応用
- DOMとXSLTを補うライブラリの構築

参考文献

- [1] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0 (Second Edition)*. <http://www.w3.org/TR/REC-xml>.
- [2] Greg J. Badros. *JavaML: A markup language for java source code*. <http://www.cs.washington.edu/homes/gjb/JavaML/>.
- [3] ECMA (European Computer Manufacturers Association). *Portable Common Tool Environment (PCTE) - Abstract Specification*, 1997. <ftp://ftp.ecma.ch/ecma-st/Ecma-149.pdf>.
- [4] Electronic Industries Association CDIF Technical Committee. *CDIF CASE Data Interchange Format - Overview, EIA/IS-106*, 1994. <http://www.eigroup.org/cdif/>.
- [5] 権藤 克彦, 川島 勇人. *XCI (Experimental ANSI C interpreter) Homepage*. 北陸先端科学技術大学院大学 <http://www.jaist.ac.jp/~gondow/xci/>.
- [6] Object Management Group (OMG). *formal/00-11-02 (XML Metadata Interchange (XMI) version 1.1)*. <http://www.omg.org/technology/documents/formal/xmi.htm>.

- [7] Rational Software Corporation. *Rational Rose*. <http://www.rational.com/products/rose/index.jsp>.
- [8] GNU Project. *GCC*. Free Software Foundation. <http://www.gnu.org/>.
- [9] M. Weiser. Program slicing. *IEEE Transaction of Software Engineering*, SE-10(4):352-357, 1984.
- [10] WWW Consortium (W3C). *Document Object Model (DOM)*. <http://www.w3.org/DOM/>.
- [11] WWW Consortium (W3C). *XSL Transformations (XSLT) Version 1.0*. <http://www.w3.org/TR/xslt>.
- [12] WWW Consortium (W3C). *XML Path Language (XPath) Version 1.0*. <http://www.w3.org/TR/xpath>.
- [13] INSTAC XML SWG. *RELAX (Regular Language description for XML)*. <http://www.xml.gr.jp/relax/>.
- [14] University of Wisconsin. *The Wisconsin Program-Slicing Tools*. http://www.cs.wisc.edu/wpis/slicing_tool/.
- [15] The Unravel Project. *The Unravel Program Slicing Tool*. <http://hissa.nist.gov/unravel/>
- [16] Shigio Yamaguchi. *GNU Global-Source Code Tag System for C, C++, Java and Yacc*. <ftp://ftp.gnu.org/gnu/global/global-4.1.tar.gz>.
- [17] Andrew M. Bishop. *The Cxref Homepage*. <http://www.gedanken.demon.co.uk/cxref/>.
- [18] Organization for the Advancement of Structured Information Standards (OASIS). *The XML Cover Pages*. <http://xml.coverpages.org/xml.html>.
- [19] Michael Kay. *What kind of language is XSLT?*. <http://www-106.ibm.com/developerworks/xml/library/x-xslt/>.
- [20] Sun Microsystems, Inc. *The Java Community Process(SM) Program - JSRs: Java Specification Requests - Detail*. <http://jcp.org/jsr/detail/031.jsp>.
- [21] Eric Armstrong. *Data Binding*. <http://java.sun.com/xml/jaxp/dist/1.0.1/docs/binding/DataBinding.html>.
- [22] 鯨坂 恒夫. 開放型 CASE プラットフォーム. コンピュータソフトウェア, Vol.10, No 2, pp.4-12, 1993.
- [23] 鯨坂 恒夫 沢田 篤史 満田 成紀. ソフトウェア評論 Emeraude PCTE. コンピュータソフトウェア, Vol.10, No 2, pp.65-77, 1993.
- [24] 篠木 裕二, 西尾 高典, 吉川 彰弘. CDIF-CASE データ変換形式. コンピュータソフトウェア, Vol.10, No 2, pp.13-25, 1993.
- [25] 福安 直樹, 山本 晋一郎, 阿草 清慈. 細粒度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.
- [26] 戸板 晃一, 山本 晋一郎, 阿草 清滋. XMLを用いたソフトウェア関連文書とソースプログラムの整合性検査ツール. 日本ソフトウェア科学会 FOSE2001, pp.129-140, 2001.
- [27] B.W. Kernighan and D.M. Ritchie 著, 石田 晴久 訳. プログラミング言語 C 第 2 版. 共立出版社, 1989.
- [28] 鯨坂 恒夫, 佐伯 元司. 方法論工学と開発環境. 共立出版, 2001.
- [29] B.W. Kernighan and R. Pike 著, 福崎 俊博 訳. プログラミング作法. ASCII, 2000.