構造化データの動作記述による表現に関する検討

A Study on Representation of Structured Data Using Behavior Description

前田 和昭

Kazuaki Maeda　kaz@acm.org

中部大学　経営情報学部

College of Business Administration and

Information Science, Chubu University

概要

ページ記述言語として開発された PostScript に代表されるように，動作の記述 (プログラム) を使ってデータを表現する技術が開発されてきた．これまで筆者は，コンパイラの中でもフロントエンドと言われる部分に限定して，コンパイラ内部の動作をデータとして外部に記録し利用することを考えてきた．本稿では，これまでの調査と経験をふまえ，構造化データを動作記述として表現することについて検討する．

1　はじめに

1980 年代，プリンタ記述言語である PostScript[1, 2] が Adobe Systems によって開発され，レーザプリンタに搭載されることによって DTP (デスクトップパブリッシング) の発展に大きく貢献した．PostScript は，スタックベースのプログラミング言語である．PostScript インタプリタを内蔵するプリンタは，PostScript で記述されたプログラムを読み，プログラムに記述された命令にしたがって描画すべきイメージを作り上げ，紙に印刷する．PostScript によるプログラムは「ページを表現したもの」と考えることができるであろう．コンピュータとプリンタ間の速度が遅かった当時，PostScript は転送データ量を減らすために非常に役立つ技術であった．さらに，プリンタの仕様に依存しないように設計されたため，PostScript によるプログラムで表現されたページはデバイスに独立していた．

1990 年頃，PostScript をウィンドウシステム内部に搭載した NeWS が開発された [3]．PostScript がデバイスに独立しているという特徴を生かし，解像度や色数が異なるいろいろなディスプレイに対応できる点でユニークであった．ディスプレイに表示されているイメージを印刷したいときには，PostScript によるプログラムを転送することで，ディスプレイとは解像度が異なるプリンタを使って美しい出力を得ることができた．PostScript では，オペレータを新しく定義することで機能を拡張することができる．NeWS では，PostScript を使ってオペレータを定義してウィンドウサーバ内部に登録すれば，ネットワークを介してクライアントからそのオペレータを遠隔実行することができる．似たようなイメージをクライアントからサーバへ何度も送る場合には，共通のオペレータをサーバ内に登録することで，転送すべきデータ量を大幅に減らすことができた．

グラフ構造を表現するための技術として GEL[4] がある. GEL は，有向グラフを表現するために開発されたスタックベースのプログラミング言語である．スタックを使いながらグラフ構造を作成するときの手順を記述することで，グラフ構造を表現する試みが興味深い．しかも，グラフ構造のデータを読み書きするプログラミング言語に独立となるように設計されている．

本稿では，ソースコードの表現であり，筆者が考案した PALEX (PArsing actions and LExical formatting information in Xml) とその応用について検討する．PALEX によるコードには，ソースコードを構文解析するときの解析動作を記録したものが含まれる．したがって，ソフトウェア開発支援ツールが PALEX コード内の解析動作を読み込めば，その記録にしたがって構文解析を再生することができる．さらには，構文解析の動作が全て記録されているので，構文解析部が構文規則をどの順番でシフトし還元したかかが分かる．PostScript がデバイスに独立することと同じように，XML 文書である PALEX は，その要素と属性にはプログラミング言語に依存するものがないため，プログラミング言語に独立している特徴がある．

2 ソースコード表現 PALEX

高水準プログラミング言語が登場したときから，ソースコードを表現するためにプレーンテキストが使われてきた．プレーンテキストは単純であることから，加工しやすいという特徴を持つ．テキストエディタを使えばテキストを編集することができ，メールを使っての送信・受信も簡単である．プレーンテキストを操作するために，便利なソフトウェアツールが多く開発されてきた．例えば，Unix では，wc コマンドを使って，ソースコードの行数を数えることができる．もし，ある特定の変数名を含む行のリストが欲しいならば，正規表現を使って grep コマンドを使えばよい．さらには，ファイルを修正した後，修正前のファイルとどこが違うのかを知るには，diff コマンドを使えばよい．

プレーンテキストで保持されるソースコードは，コンパイラの入力して使われる．コンパイラは，ソースコードを読み込み，字句と構文を解析し，ソースコードの階層構造を表現するための木を作り上げる．この木のことを AST (抽象構文木, Abstract Syntax Tree) と呼ぶ [5]．AST は，実用的なソフトウェア開発支援ツールに広く採用され，プログラミング言語向きエディタやソースコードトランスレータなどが開発された．

過去 30 年の間，多くの人達がいろいろな AST を開発してきた．例えば，Ada プログラムの AST のために Diana が設計された [6]．Diana は，Ada コンパイラのための中間表現として適しているだけでなく，Ada プログラミング環境や他のツールのために使われた．Diana を実際に経験するには，Scorpion IDL コンパイラキットを使うと良い [7]．Diana の仕様を Scorpion IDL コンパイラキットに読み込ませると，AST をファイルに書き出したり読み込むための便利なプログラムを生成する．Scorpion ツールキットのためのファイル表現は，Scorpion に依存し，Scorpion 以外で使用することを考慮していない．一般的に AST は，特定のプログラミング言語を特定のコンパイラで扱う場合のために設計されている．大抵の場合，それらはコンパイラのためだけに使われ，その他のソフトウェアツールのためには有効ではない．その上，複数のプログラミング言語を組み合わせて使えるように設計されていない．

XML を使ったいくつかのソースコード表現がある．XML を使えば複数のプラットフォーム (コンピュータ, オペレーティングシステム, プログラミング言語) で利用可能となる．ソースコードを解析して XML で出力することができれば，ソフトウェア開発支援ツールを複数のプログラミング言語で作成可能となる．JavaML [8] と XMLizer [9] は，AST を XML で表現する代表的なソースコード表現である．それらは，プログラミング言語の構文に関する情報を入手しやすいようになっている．XSDML [10] と srcML [11] は，AST の表現の他に，空白やコメント情報などの字句情報を追加したものである．したがって，XSDML や srcML で表現されている字句情報を使えば，XML 文書からオリジナルのソースコードを復元することができる．

筆者は，XML を使ったソースコード表現 PALEX の開発を進めながら，その応用を検討している．PALEX は，記録された解析動作と，空白・コメントを含む字句情報からなり，次の 2 つの特徴を持つ．

- PALEX の要素と属性にプログラミング言語に依存するものがないため，PALEX はプログラミング言語と独立している．
- PALEX 文書からオリジナルのソースコードを再現するための十分な情報が含まれている．

Java コンパイラである GNU GCJ を改造し，Java ソースコードを読み込み PALEX 文書を書き出すための処理系 Mogcj を作成した．一般的に，コンパイラがソースコードを解析し終えると，解析動作 (シフト・還元・トークン読み込み) を記録することができる．PALEX には，解析動作と字句情報，さらには空白やコメントも含まれる．このような PALEX の概略を説明するために，Java で書かれた次の例を使う．

```
import java.*; // simple statement
```

これは，コメントを含む import 文である．Mogcj は，ソースコードを解析し，図 1 に示すような PALEX コードを生成する．表 1 に要素とその意味，表 2 に属性とその意味を示す．要素名と属性名は，XML 文書のサイズを節約するために短い名前を採用した．

3 PALEX のソースコード差分解析への適用

ファイルを修正した後，修正前のファイルとの差分を解析するツールとして diff コマンドが有名である．diff コマンドは，2 つのファイル間で行単位の差分を見つける．その差分解析のために，LCS (Longest Common Subsequence) アルゴリズムを使っている [12].

行ベースの差分解析アプローチは，ソースコードの構文を考慮しないため，プログラミング言語に独立であり，解析対象のファイルがどのようなプログラミング言語で記述されようが関係なく解析が可能である．ところが，構文での変更がないのにもかかわらず，行ベースでの解析では変更があったと扱われることがあり，利用者にとっては嬉しくない場合がある．例えば，プリティプリンタを使ってソースコードを美しくフォーマットした場合，このファイルの構文に

```
<?xml version="1.0" encoding="us-ascii"?>
<parseFiles lang="java" pg="bison" ver="0.5">
 <parse name="importExample.java">
<lex st="0" tk="IMPORT_TK" va="import" li="1" co="1"/>
<sft fr="0" to="3"/><wsc va=" "/>
<lex st="3" tk="ID_TK" va="java" li="1" co="8"/>
<sft fr="3" to="22"/>
<rdc st="22" ru="24"/>
<stc fr="3" to="26"/>
<rdc st="26" ru="22"/>
<stc fr="3" to="24"/>
<rdc st="24" ru="20"/>
<stc fr="3" to="23"/>
<lex st="23" tk="DOT_TK" va="." li="1" co="8"/>
<sft fr="23" to="43"/>
<lex st="43" tk="MULT_TK" va="*" li="1" co="12"/>
<shi fr="43" to="59"/>
<lex st="59" tk="SC_TK" va=";" li="1" co="13"/>
<sft fr="59" to="80"/>
<rdc st="80" ru="45"/>
<stc fr="0" to="15"/>
<rdc st="15" ru="41"/>
<stc fr="0" to="13"/>
<rdc st="13" ru="33"/>
<stc fr="0" to="10"/>
<wsc va=" // simple statement&#xA;"/>
<lex st="10" tk="$end" va="" li="2" co="0"/>
<rdc st="10" ru="27"/>
<stc fr="0" to="9"/>
<rdc st="9" ru="1"/>
<stc fr="0" to="8"/>
</parse>
<accept/>
</parseFiles>
```

図 1　Mogcj が生成した PALEX コードの例

は変更はなく，字句の位置が変わるだけである．このような場合，行ベースのアプローチでは数多くの行で変更があったと報告することになる．また，JavaDoc コメントだけを変更した場合，たくさんあるファイルから，プログラムに変更があったファイルを捜し出そうとすると，利用者の希望と違った結果を報告することになる．

このような行ベースのアプローチの欠点を克服するために，より強力な方法として，構文ベースの差分解析が提案されている [13, 14]．しかし，構文ベースのアプローチでは，2 つの木構造を比較し，最適な差分を計算する必要があり，性能上問題になることがある．また，コンパイラにおけるフロントエンドでは，空白やコメントなどの字句情報を無視することが通常であるため，字句レベルでの変更を詳細に解析することができない．

意味ベースのアプローチは，構文ベースのアプローチよりも多くの機能を提供する [15]．例えば，意味情報を使えば，変数の宣言に変更があったときに，その変更が波及する場所を全て探し出すことができる．しかし，意味ベースの差分解析アプローチは，単純なプログラミング言語での研究報告しかなく，商用で使われているプログラミング言語でどこまで利用できるか明らかではない．

そこで，PALEX を使ってソースコードの差分解析を行う PalexDiff を開発することにした．PalexDiff は，字句差分解析と構文差分解析の 2 種類を提供する．字句を解析の最小単位とする字句差分解析は，字句単位での LCS を計算するときの違いにより，弱解析と強解析の 2 種類を準備することにした．弱解析を指定すると，字句の位置情報を無視して，字句が持つ文字列だけを使って差分解析を行う．強解析は，字句の種類・文字列・位置情報の全てを使って差分解析を行う．

PALEX は，図 1 に示すような線形の構造を持つデータである．ソースコード解析後に，本来は木構造となるデータを線形構造で表現しているしたがって，木構造間での差分解析よりも性能の良いツールを提供できると考えている．

表 1　パーザ動作で使われる要素の一部

名前	要素の意味
parseFiles	ルート要素
parse	一つのソースファイルの情報を表現
wsc	空白とコメント
lex	トークンの読み込み
sft	シフト動作
rdc	還元動作
stc	別の状態へ遷移
xdc	ドキュメンテーションコメント
accept	解析終了

表 2　パーザ動作で使われる属性の一部

名前	属性の意味
lang	プログラミング言語名
pg	パーザ生成系
ver	PALEX のバージョン番号
name	ソースファイル名
st	状態を識別する番号
fr	シフトする前の状態の識別番号
to	シフトした後の状態の識別番号
tk	トークンの種類
va	トークンの文字列イメージ
li	ソースファイル中の行番号
co	ソースファイル中の列番号
ru	文法規則を識別する番号

4　おわりに

本稿では，ソースコード表現である PALEX について述べ，その応用の一例としてソースコード差分解析 PalexDiff について簡単に述べた．今後，PalexDiff の実装を進めて性能評価を行い，また他のソフトウェア開発支援ツールへの応用も検討することで PALEX の優位性を追求していきたい．

参考文献

[1] Adobe Systems. *PostScript Language Tutorial and Cookbook*. Addison-Wesley, 1985.

[2] Adobe Systems. *PostScript Language Program Design*. Addison-Wesley, 1988.

[3] Sun Microsystems. *NeWS 2.1* **ネットワークプログラマーズガイド**. 1990.

[4] J. F. Th. Kamperman. GEL, a Graph Exchange Language, CWI Report CS-R9440, 1994.

[5] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers : principles, techniques, and tools, 2nd Ed.* Pearson Education, 2006.

[6] G. Goos and Wm. A. Wulf. Computer Science Technical Report. Technical report, Carnegie-Mellon University, 1981.

[7] Richard Snodgrass. *The Interface Description Language: Definition and Use*. Computer Science Press, 1989.

[8] Greg Badros. JavaML: A Markup Language for Java Source Code. In *9th International World Wide Web Conference*. http://www9.org/w9cdrom/index.html, 2000.

[9] Gregory McArthur, John Mylopoulos, and Siu Kee Keith Ng. An extensible tool for source code representation using xml. In *9th Working Conference on Reverse Engineering*, pp.199–209, 2002.

[10] Katsuhisa Maruyama and Shinichiro Yamamoto. A CASE Tool Platform Using an XML Representation of Java Source Code. In *4th IEEE International Workshop on Source Code Analysis and Manipulation*, pp.158–167, 2004.

[11] Jonathan I. Maletic, Michael Collard, and Huzefa Kagdi. Leveraging XML Technologies in Developing Program Analysis Tools. In *4th International Workshop on Adoption-Centric Software Engineering*, pp.80–85, 2004.

[12] J. W. Hunt and T. G. Szymanski, A Fast Algorithm for Computing Longest Common Subsequences, Communications of the ACM, Vol.20, No.5, pp.350–353, 1977.

[13] Bernhard Westfechtel, Structure-oriented merging of revisions of software documents, *The 3rd International Workshop on Software Configuration Management*, pp.68–79, 1991.

[14] Jim Buffenbarger, Syntactic Software Merging, Software Configuration Management, *Lecture Notes in Computer Science*, pp.153–172, Vol.1005, Springer, 1995.

[15] Susan Horwitz, Jan Prins and Thomas Reps, Integrating Noninterfering Versions of Programs, *ACM Transactions on Programming Languages and Systems*, pp.345–387, Vol.11, No.3, 1989.