システム開発における XML の効果的な活用に関する検討 ～逆エンジニアリングツール開発の経験から～

A Study on Effective Use of XML in System Development ～From Experiences of Developing Reverse Engineering Tools～

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

ここ数年来，XML が注目され，ビジネスでの積極的な利用が進んできている．本稿では，逆エンジニアリングツール開発の経験を踏まえ，オペレーティングシステム，プログラミング言語，ツールなどの観点から，システム開発における XML の効果的な活用について述べる．

1　はじめに

インターネットの発達とコンピュータの飛躍的な性能向上を背景にして，XML (Extensible Markup Language) を使ってデータを表現することが多くなってきた．XML は，データに意味を付記することができるように SGML から派生したマークアップ言語であり，XML で記述されたデータである XML 文書は，人間が読めるようにテキスト形式で表現される．WWW の普及にともない，電子商取引などビジネスで広く活用できるようにいろいろな工夫がなされてきた．

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）に依存しないという XML の特徴を使うことで，異種のプラットフォーム間でのデータ交換が可能となる．例えば，アプリケーション間でデータを交換したり，データベースからデータを取り出して他の製品で活用したり，またネットワークを介してデータ交換する場面など多くの応用分野が広がってきている．

XML 文書を使い方で分類すると，

- 文書中心 XML 文書 (document centric documents)
- データ中心 XML 文書 (data centric documents)

のどちらかになる [1, 2]．文書中心 XML 文書は，主として人間が読むために作られるもので，規則性が少なく，データの粒度が粗い特徴を持つ．例えば，本や記事や電子メールなどを記述したものがそれにあたる．これに対して，データ中心 XML 文書は，コンピュータでの処理やデータベースに格納するためのもので，規則性があり，データの粒度が細かい特徴を持つ．例えば，文献データや注文伝票などを記述したものがそれにあたる．本稿では，データ中心 XML 文書のことを，単に「XML データ」と呼ぶことにする．

2　逆エンジニアリングツールのためのパーザと XML データ出力

ソフトウェアの設計情報と実装後のプログラムが一致しない場面の解決策として，ソースコードから設計情報を生成する逆エンジニアリングツールがいくつか提案されている [3]．この逆エンジニアリングツールの作成には，ソースコードを解析するためのパーザに相当するプログラムが必要である．ところが，最近のプログラミング言語の言語仕様は複雑であり，その複雑な言語仕様にしたがってパーザを作成するにはかなりの工数が必要となる．

コンパイラは，図 1 に示すように，ソースコードを読み込み，字句解析，構文解析，意味解析，最適化の処理を終えた後，オブジェクトコードを生成する．コンパイラの主となる目的は，ソースコードから効率の良いオブジェクトコードを生成することであり，コンパイラを他の目的のために利用することはほとんどない．コンパイラの実装には長年のノウハウが蓄積されている．そこには，優れたアルゴリズムを駆使したプログラムや，別の目的に使えるはずの情報が

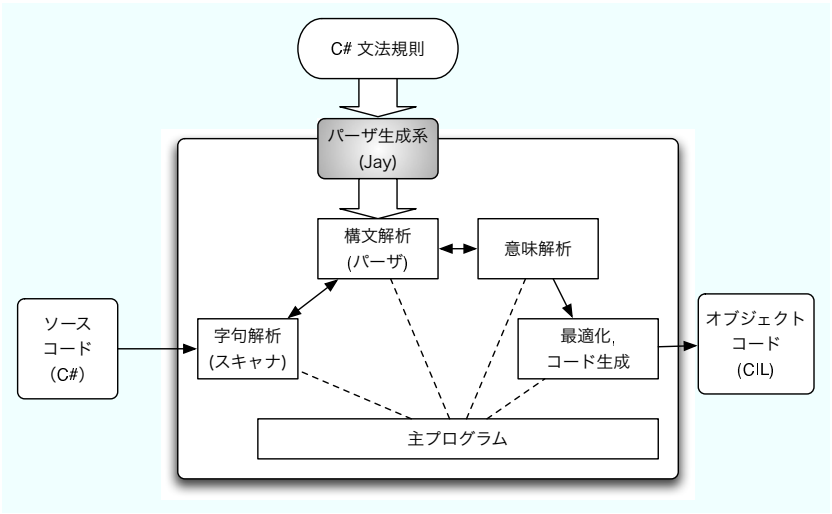

図 1　コンパイラの一般的な構成

```
using System;
namespace Com.Xyz
{
    public class Hello
    {
    }
}
```

図 2　C#ソースコードの例

```
<lex tk="USING" va="" li="1" co="5" />
<lex tk="IDENTIFIER" va="System" li="1" co="12" />
<lex tk="SEMICOLON" va="" li="1" co="13" />
<lex tk="NAMESPACE" va="" li="2" co="9" />
<lex tk="IDENTIFIER" va="Com" li="2" co="13" />
<lex tk="DOT" va="" li="2" co="14" />
<lex tk="IDENTIFIER" va="Xyz" li="2" co="17" />
<lex tk="OPEN_BRACE" va="" li="3" co="1" />
<lex tk="PUBLIC" va="" li="4" co="10" />
<lex tk="CLASS" va="" li="4" co="16" />
<lex tk="IDENTIFIER" va="Hello" li="4" co="22" />
<lex tk="OPEN_BRACE" va="" li="5" co="5" />
<lex tk="CLOSE_BRACE" va="" li="6" co="5" />
<lex tk="CLOSE_BRACE" va="" li="7" co="1" />
```

図 3　字句情報の XML データ出力の一部

埋もれていると筆者は考えている．

.NET の開発・実行環境をオープンソースソフトウェアとして提供することを目的とした Mono プロジェクト [4] の一部として，C#コンパイラ gmcs が提供されている．gmcs では，図 1 に示すように，構文解析プログラムを開発するためにパーザ生成系の Jay が使われている[*1]．Jay は C#の文法規則を読み込み，C#で書かれた構文解析プログラムを生成する．そこで，内部の情報を出力する機能をコンパイラに埋め込むために，Jay を，新しく開発したパーザ生成系 MJay に入れ替えることを考えた．Jay と MJay を入れ替え，gmcs のソースコードに若干の修正を施すだけで，字句情報，構文情報，解析木の 3 種類が XML データとして出力可能となる．これらの XML データは，逆エンジニアリングツール開発に役立てることを考えて作られたものである．以下，XML データとして出力される 3 種類の XML データについて簡単に述べる．

字句情報の XML データ出力

図 2 に C#ソースコードの例を示す．このソースコードは XML データ出力を説明するためだけに用意した極端に単純な例である．MJay に入れ替え済みの gmcs を使って，図 2 のソースコードを読み込み構文解析を行うと，図 3 に示すような字句情報を XML データとして出力できる．

字句情報の XML データには，図 3 から分かるように，字句の種類，字句のテキストイメージ，行番号，列番号が含まれる．したがって，字句情報の XML データだけから，入力となる C#ソースコードを復元することが可能である．C#ソースコードを復元するとき，読みやすいように字体や色を変えるなど工夫した HTML 文書を生成することも可能である．

[*1] ここで使われている Jay は，Java のために開発された Jay[5] を C#のためにに書き換えた修正版である．

```
<parse name="hello.cs">
<lex st="0" tk="USING" va="" li="1" co="5" />
<shi fr="0" to="3" />
<lex st="3" tk="IDENTIFIER" va="System" li="1" co="12" />
<shi fr="3" to="30" />
<lex st="30" tk="SEMICOLON" va="" li="1" co="13" />
<red st="30" ru="326" />
<red st="72" ru="325" />
<red st="33" ru="322" />
<red st="31" ru="28" />
<shi fr="32" to="74" />
<red st="74" ru="21" />
<red st="14" ru="18" />
<red st="11" ru="10" />
<red st="9" ru="7" />
<lex st="6" tk="NAMESPACE" va="" li="2" co="9" />
```

図 4　パーザ動作の XML データ出力の一部

```
<compilation_unit>
  <outer_declarations>
    <outer_declarations>
      <outer_declaration>
        <using_directive>
          <using_namespace_directive>
            <USING va="using"/>
            <namespace_name>
              <namespace_or_type_name>
                <member_name>
                  <IDENTIFIER va="System"/>
                </member_name>
              </namespace_or_type_name>
            </namespace_name>
            <SEMICOLON va=";"/>
          </using_namespace_directive>
        </using_directive>
```

図 5　解析木の XML データ出力の一部

構文情報の XML 出力

MJay が生成するパーザは，Yacc[6] と同じように，LALR 法に基づく上向き構文解析を行うパーザである．このパーザは，字句読み込み，シフト，還元などの動作を繰り返すことで，入力となるソースコードの構文を解析する．

MJay に入れ替え済みの gmcs を使って，図 2 のソースコードを読み込み構文解析を行うと，図 4 に示す XML データが出力される．この XML データは，パーザの動作を表しているので，ここでは「パーザ動作」と呼ぶことにする．パーザ動作で使われる要素の名前と意味を表 1 に，属性の名前と意味を表 2 に示す．パーザ動作は，MJay に入れ替え済みの gmcs のパーザが，どのような動作をどのような順番で行ったかを記録した XML データと考えて良い．

表 1　パーザ動作で使われる要素の一部

名前	要素の意味
parse	一つのソースファイルの情報を表現
lex	トークンの読み込み
shi	シフト
red	還元

表 2　パーザ動作で使われる属性の一部

名前	属性の意味
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

解析木の XML データ出力

通常のコンパイラでは，構文解析が終了すると抽象構文木ができあがる．抽象構文木は，入力となるソースコードの構造を保持しながら，コンパイラ開発者の直感に基づいた表現を持つ木構造データである．抽象構文木を作り上げるには，構文解析の途上に，抽象構文木を作り上げるためのプログラムを埋め込まなければならない．

コンパイラの教科書では，構文解析の解説のために解析木を使うことがよくある [7]．プログラミング言語の文法規則が定義され，その文法規則にしたがってソースコードの構文を解析するとき，解析木を自動的に作り上げることができる．ところが，構文解析の結果としては抽象構文木を使うことがほとんどで，解析木を使うことはない．解析木は，構文解析部内部の解説やデバッグのためにしか使われていない．

そこで，解析木を XML データとして出力し利用することを考える．MJay に入れ替え済みの gmcs を使って，図 2 の C#ソースコードを読み込み構文解析を行うと，図 5 に示す解析木を XML データとして出力することができる．

3 XML データを利用したツールの作成

XML データとして出力されたパーザ動作を利用することで，パーザを迅速に開発することが可能になる．パーザを 2 つのステップに分割することから，この手法を「2 段階パージング」と呼び，以下のように開発を進める [8].

- gmcs で使われている Jay を MJay に入れ替え，ステップ 1 で使う．
- ステップ 1 では，gmcs のソースコードを修正して利用する．ただし，パーザの処理が終了したところでコンパイラの処理を止めることにより，構文解析までを実行する．
- ステップ 2 のための文法規則は，MJay が自動生成する．また，ステップ 2 を駆動するためのパーザ動作を XML データとして，ステップ 1 が出力できるようにする．
- ステップ 2 では，ステップ 1 で出力された XML データを使って，ステップ 1 で既に終わった構文解析と同じ動きを再現する．ただし，自動生成された文法規則から，通常のパーザ生成系を使ってステップ 2 のパーザを生成するため，ステップ 2 のパーザの機能は構文解析のみであり，抽象構文木を生成する機能は持たない．

2 段階パージングを使ったパーザを含む逆エンジニアリングツールは，2006 年春から市販製品の一部として組み込まれて出荷されている．この開発のためのプログラミング言語としては C#を使い，Mac OS X 上で Mono を使って開発を進めた．開発後，完成したパーザを市販製品に組み込むために，ソースコードを Windows へ転送してビルド作業を行った．このときのビルド作業では，ソースコードで入手した Mono を Windows 上で動かすこと以外に問題は起こらなかった．これは，XML データと C#が，オペレーティングシステムに依存しないことを意味している．

2 段階パージングのステップ 2 では，MJay が生成した文法規則を使う．この文法規則は，Yacc の文法規則と互換性があるように作られている．すなわち，ステップ 2 の文法規則と，パーザ動作のための XML データは，両方ともプログラミング言語から独立することになる．最近のプログラミング言語には，XML を処理するライブラリが提供されている．したがって，ステップ 1 として C#で作成された gmcs をそのままを使い，ステップ 2 を C#以外のプログラミング言語（例えば Java）を使って実装することが可能となる．

パーザ動作を表す XML データは大きくなる傾向がある．実験を行った結果，7,052 行の C#ソースコードを解析すると，約 7M バイトの XML データが生成されることが分かった．データを XML を使って出力すれば，そのデータを操作するときに便利であるものの，この XML データの大きさは通常の XML データの大きさを越えているように思える．しかし，生成される XML 文書は作業用の一時的なものであり，ステップ 2 の終了後に削除されるため重大な問題とはならないと考えている．出力される大きな XML データは，システムの性能劣化を引き起こすので注意が必要である．

XML データとして出力された解析木に対しては，XSLT を使うことで有益な情報を取り出すことができる．例えば，ソースコード内にある名前空間，クラス，クラス内のメソッドなどの一覧は，XSLT を使って生成するができる．

4 おわりに

本稿では，逆エンジニアリングツールを開発するときに考えた XML データについて述べた．これは，XML データの効果的な活用を追求するための最初の一歩にすぎない．XML が注目されビジネスでの積極的な利用が進んできている．XML の良さは，標準化されていることと，オペレーティングシステムやプログラミング言語に独立しているところにある．XML を使って，これまでシステムが内部に保持してきた情報を積極的に公開することで，システムの興味深い活用方法が生まれてくると考えている．今後，より効果的な XML の活用方法を追求していくことで，システム開発の実践および研究開発の場に対して，何らかの貢献をしていきたい．

参考文献

[1] Akmal B. Chaudhri, and others, XML Data Management, Addison Wesley, 2003.

[2] Ronald Bourret, “XML and Databases,” *http://www.rpbourret.com/xml/XMLAndDatabases.htm* .

[3] 竹下 亨, ソフトウェアの保守・再開発と再利用, 共立出版, 1992.

[4] Main Page - Mono, *http://www.mono-project.com/Main_Page* .

[5] jay Homepage, *http://www.informatik.uni-osnabrueck.de/alumni/bernd/jay/* .

[6] S. C. Johnson, “Yacc: Yet Another Compiler Compiler, ” *UNIX Programmer’s Manual*, Vol. 2, pp. 353–387, 1979.

[7] A. V. エイホ, R. セシィ, J. D. ウルマン, コンパイラ — 原理・技法・ツール，サイエンス社, 1990.

[8] Kazuaki Maeda, “Experience of Quick Parser Development from Free and Open Source Compilers,” IEEE International Symposium on Communications and Information Technologies, 2006.