5U-8

ＡＳＮ．１実装支援ツール（II）
－フォーマット処理記述言語ｆｉｘによる開発－

大浦慎司*、小林茂**、横山達也*

*日本電気(株) Ｃ＆Ｃシステムインタフェース技術本部、**日本電気技術情報システム開発(株)

１.はじめに

最近、ＩＳＯのプロトコル記述等に形式的記述法(ＦＤＴ)が適用されつつある。フォーマット処理記述言語ｆｉｘおよびその処理系は、ＦＤＴを利用してソフトウェア開発を効率化する目的で開発された[1]。

一方、ＯＳＩによる標準化に伴い、その上位層プロトコルの実装を効率化する必要が生じており、各社から様々なＡＳＮ．１支援ツール[3,4]が発表されている。

我々はｆｉｘを利用して、ＡＳＮ．１支援ツールＡｃｃｅｌのプロトタイプ(ｆｉｘ版)を作成し、その概要を報告した[2]。今回はそこで明らかになった問題点を考慮して、実用的な実装ツールとなるよう改良を行なったので、ｆｉｘによるツール開発の効率化を中心に報告する。

２.ＡＳＮ．１支援ツール

ＡＳＮ．１は二つの規約からなっている。ひとつは応用層のデータ構造を記述するための抽象構文記法である。もうひとつは、抽象構文とプレゼンテーション層のデータ（転送構文）を対応づけるための符号化規則である。ＯＳＩの応用層では、抽象構文記法にしたがってプロトコルが記述されている。

アプリケーションの開発者は、抽象構文と符号化規則の両方を熟知した上で、開発を行なわなければならない。ところが、符号化規則が複雑であり、また抽象構文は規模の大きなものになるので、アプリケーションを転送構文レベルで開発するのは困難な作業である。

そこで、我々は、抽象構文を利用してＯＳＩアプリケーションの開発を効率化するツールＡｃｃｅｌを開発した。

３.プロトタイプ版の問題点

プロトタイプ[2]では、ｆｉｘにより、抽象構文から専用のＡＳＮ．１デコーダのｆｉｘ言語によるコードを自動生成した。つまり、専用デコーダの生成のためのコマンドと、デコーダの記述用言語の両方にｆｉｘを適用した。自動生成されたｆｉｘ言語によるデコーダは専用デコーダであるため、処理速度の問題はないが、生成される実行モジュールの所要メモリが大きくなりすぎる欠点があり、ＰＣ等への実装には問題がある。また、エンコーダの機能は提供していない。

４.実用版の機能

そこで、実用版では、実行モジュールのプログラムサイズを重視し、抽象構文から汎用のコーダのための駆動テーブルを生成する方式を採用した。また、名前による転送構文の要素の指定、ライブラリ関数の整備、関数間のインタフェースなどを考慮して設計を行なった。これらの特徴から、ユーザは抽象構文を理解していればアプリケーションの開発ができ、符号化規則については知っている必要がない。コンパイラの構成を図１に示す。

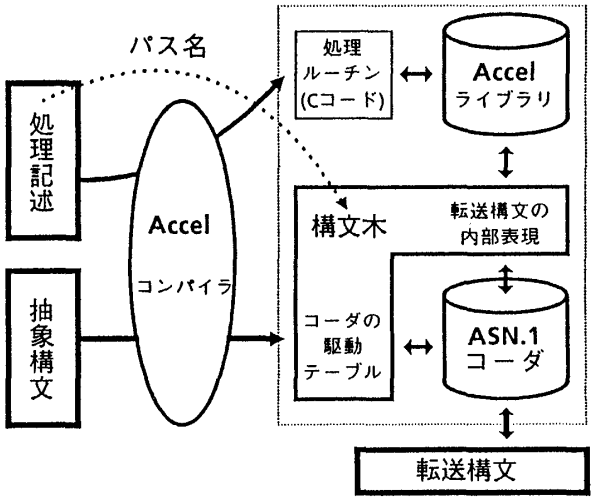

図1 コンパイラの構成

コンパイラは抽象構文から汎用コーダの駆動テーブルを自動生成する。これは同時に転送構文の内部表現となり、パス名によるインタフェースにより、値の参照・代入ができる。

ｆｉｘ版ではｆｉｘの組込機能により、パス名による転送構文の要素の指定を実現していたが、実用版でもパス名で各要素を指定できる。実用版ではライブラリ関数によりその機能を代替し、ユーザインタフェースを改良した。

Ａｃｃｅｌライブラリには、コーダ、構文木を生成する関数、構文木をコピーしたり切り貼りしたりする関数、ＣとＡＳＮ．１データの間の値の変換をする関数、デバッグ用の関数などがある。

５.ｆｉｘによるＡｃｃｅｌコンパイラの開発

ｆｉｘはＹａｃｃと同様にＢＮＦ記法(原構文－図３(a))からパーザを生成する。ＡＳＮ．１の抽象構文の文法規則はＢＮＦにより与えられており、それはほぼそのままの形でｆｉｘの原構文やＹａｃｃの文法規則になる。

ASN.1 support tool (2) - Implimentation by format description language "fix"
Shinji OURA*,Shigeru KOBAYASHI**,Tatsuya YOKOYAMA*
*NEC Corporation,**NEC Scientific Information System Development,LTD.

さらに、ｆｉｘではＹａｃｃの機能に加えて、ＢＮＦライクな記法(目的構文－図３(b))で原構文に対する処理の記述を書ける。ｆｉｘの機能は報告済みであるのでここでは詳述しないが、ｆｉｘ言語によるコーディングでは、入力構文と出力構文の対応規則(目的構文に埋め込まれる)により処理のアウトラインが記述できるため、ユーザのコーディングするプログラムがきわめて単純なサブルーチンに還元される。

また、Ｙａｃｃではアクションにより処理を記述するが、アクション実行のタイミングは文法規則の還元のタイミングに依存するため、アクションを埋め込む位置について考慮をしなければならない。一方、ｆｉｘでは入力のパージングが終了してから、入力が展開された構文木を元に処理を行なうので、タイミングについてはほとんど考慮する必要がない(ただし、アクションによる処理の記述も可能である)。この点がＹａｃｃと比較して特に使い勝手が優れている点である。

Ａｃｃｅｌのコンパイラは抽象構文からその内部表現(構文木)を生成するＣプログラムを自動生成する(図２)。このように入力構文の文法と出力構文の文法(もしくは処理の構造)が似ているときには、入力構文のシンボル名を対応する出力構文のシンボル名(もしくは処理関数)に置き換え、両者の対応関係を目的構文として埋め込むことで、目的構文をほとんど機械的に得ることができる。コンパイラのための原構文と目的構文の例を図２に示す。ただし、この例は実際の記述を単純化したものの一部である。

図３の(a)が原構文で、Ｙａｃｃとほとんど同じものである。例えば１行目は型割当てが、参照名、等号、型からなることを表わしている。これらが受理されるとアクションにより、型割当てに対する処理を行なう目的構文が、型割当てに対する構文木を引数として呼び出される。

図３の(b)が目的構文である。型割当てに対する目的構文では、１つのＣ関数を生成(標準出力に出力)する。目的構文の print() はノードの内容を標準出力に出力するＣ関数である。型に対する処理は型の種類により異なるので、型を処理する目的構文が構文木を引数として呼び出される。ただし、ここで処理の都合上、付加的なパラメータも一緒に渡している。さらに、型に対する処理は、それが組込型か定義型かにより分岐し、組込型の場合は、さらにその種類により処理が分岐する。このように処理を書き下していって、最終的にＣ言語により作成しなければならない目的構文は、print() のようなプリミティブな関数のみである。

６.まとめ

ｆｉｘを利用してＡＳＮ.１ツールの開発を行ない、プロトタイプから実用版まで効率的に開発を行なうことができた。特に、ＡＳＮ.１はＢＮＦに基づいた言語であるためｆｉｘとの適合性が高かった。ｆｉｘでは入出力双方のフォーマットをＢＮＦで表現するため、プログラムの文書性も優れており、デバッグ等の保守作業も容易であった。なお、本稿では触れなかったが、ＡｃｃｅｌのＡＳＮ.１コンパイラ以外のツールの多くも、ｆｉｘを利用して開発されており、成果をあげている。およそＹａｃｃが利用できる場面ではｆｉｘを利用でき、Ｙａｃｃよりも大きな効果をあげている。

Ａｃｃｅｌは実用版が完成し、ＯＳＩの実装に利用され、その有効性が確認されている。今後は、マクロ記法をサポートするツールなどもｆｉｘにより開発していきたい。

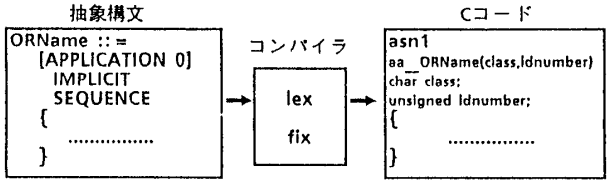

図2 コンパイラの機能

```
Typeassignment  : reference EQUAL Type {d_Typeassignment(Typeassignment);} ;
Type            : BuiltinType | DefinedType ;
DefinedType     : reference ;
BuiltinType     : BooleanType | .... | SequenceType | .... ;
BooleanType     : BOOLEAN ;
SequenceType    : SEQUENCE B_BRA ElementTypeList B_KET | SEQUENCE B_BRA B_KET ;
```

(a)原構文

```
d_Typeassignment(Typeassignment)
        : printf("¥nasn1 aa_") print(.reference) printf("(class,idnumber)¥n")
          printf("char class;¥nunsigned idnumber;¥n{¥nreturn(¥n")
          d_Type(.Type,1,0,0) printf(");¥n}¥n") ;

d_Type(Type,outermost,implicit,tag) int outermost,implicit; NODE *tag;
        : (.BuiltinType) d_BuiltinType(.BuiltinType,outermost,implicit,tag)
        | (.DefinedType) d_DefinedType(.DefinedType,outermost,implicit,tag) ;

d_DefinedType(DefinedType,outermost,implicit,tag) int outermost,implicit; NODE *tag;
        : [(outermost) printf("aaapdname(")]
          printf("aa_") print(.reference) printf("(")
          d_argument(outermost,implicit,tag) printf(")")
          [(outermost) printf(",_") print(Typeassignment.reference) printf(")")] ;

d_argument(outermost,implicit,tag) int outermost,implicit; NODE *tag;
        : ( outermost && implicit) printf("DEFAULTTAG(") d_Tag(tag) printf(")")
        | ( outermost && !implicit) printf("class,idnumber")
        | (!outermost && implicit) d_Tag(tag)
        | (!outermost && !implicit) printf("NOTAG,NULL") ;

d_BuiltinType(BuiltinType,outermost,implicit,tag) int outermost,implicit; NODE *tag;
        : (.BooleanType)  d_BooleanType(.BooleanType,outermost,implicit,tag)
        .....
        | (.SequenceType) d_SequenceType(.SequenceType,outermost,implicit,tag)
        ..... ;

d_BooleanType(BooleanType,outermost,implicit,tag) int outermost,implicit; NODE *tag;
        [(outermost) printf("aaapdname(")]
        printf("aa_BOOLEAN(") d_argument(outermost,implicit,tag) printf(")")
        [(outermost) printf(",_") print(Typeassignment.reference) printf(")")] ;

d_SequenceType(SequenceType,outermost,implicit,tag) int outermost,implicit; NODE *tag;
        [(outermost) printf("aaapdname(")]
        printf("aaseq(") d_argument(outermost,implicit,tag)
        [(.ElmentTypes) d_ElementTypes(.ElementTypes)] printf(",¥nEOL") printf(")")
        [(outermost) printf(",_") print(Typeassignment.reference) printf(")")] ;
```

(b)目的構文

図３　ｆｉｘによるＡｃｃｅｌコンパイラの記述

＜参考文献＞
[1]大竹和雄「プロトコルにおけるフォーマット変換の形式的記述法」第33回情処全国大会（1986）
[2]大竹和雄 他「ＡＳＮ.１からプログラムを自動生成するツール」　第37回情処全国大会（1988）
[3]中川路哲男 他　「ＡＳＮ.１ツールＡＰＲＩＣＯＴの設計」　第35回情処全国大会（1987）
[4]長谷川享 他　「ＡＳＮ.１支援ツールの開発」マルチメディア通信と分散処理研究会(1988)