ソフトウェア自動合成シェル - EasySOFTEX - (2)
- アプリケーションジェネレータ設計のための問題向け設計言語 -

3W−5

徳岡宏樹、友部 実、山之内 徹
NEC C&C メディア研究所
e-mail: {tokuoka,tomobe,yamanouchi}@ccm.cl.nec.co.jp

1 はじめに

特定ドメイン向けにアプリケーションプログラムジェネレータ (APジェネレータ) を構築することで、アプリケーション開発を効率化、高品質化する手法が広く用いられている [1][2]。

APジェネレータの開発においては、開発効率や開発後の保守性の観点から、目的のプログラムのソースコード出力部の可読性が重要である。しかし、通常のプログラミング言語では、出力部を直観的に記述するのが困難である。

関数型言語をベースに設計されたAPジェネレータ開発用言語EasySOFTEX[4]では、このような問題を解決するために、ソースコード出力部の記述に特化したテンプレート記述言語を用いる。本稿ではこのテンプレート記述言語について詳説する。

2 テンプレート記述言語

2.1 ソースコード出力における課題

APジェネレータは、プログラムの仕様記述を読み込み、その仕様記述を解釈・変換し、目的のプログラムのソースコードを出力する。APジェネレータ開発の際には、そのうちソースコード出力を行うための記述をソースコード出力部として他の部分とは区別して記述しておくのが望ましい。そうしておくことで、途中でどういう変換が行われているかを全く知らなくても、対象のアプリーケーションのドメインについての知識さえあれば、出力部に着目するだけでジェネレータの保守や改造を行う事ができるからである。

APジェネレータは、定型的に行う処理の中に仕様によって変化する部分が存在したり、少しずつ違うが似通っている処理を繰り返すなど、機械的なソースコードの生成が可能なアプリケーションドメインを対象とする。従って、APジェネレータの出力記述部は、仕様記述によらず定型のソースコードとして出力される定型部の中に、仕様記述によって生成されるソースコードである可変部を埋め込むような記述となる場合が多い。

例えば、図1に示すようなコードを考える。四角で囲った部分が仕様によって変更されうる可変部、その他を定型部とする。このようなコードを出力するためのソースコード出力部は、通常のプログラミング言語で記述すると、図2のようになるが、これは可読性が低い。可変部の埋め込み位置やインデントなどの整形処理は文字列の連結操作等で記述する必要があり、扱う対象がプログラムのソースコードであるために特殊文字のエスケープ (\nなど) の多用を強いられる。その結果、生成されるソースコードとはかけはなれてしまい、上で述べた保守性の向上の効果が充分に得られない。

```
void *mesOutPut(char *message, int mode) {
    if(mode == M_MODE1){
        printf("%s\n", message);
    } else {
        logOutPut(message);
        changeMode(mode);
    }
}
```
（可変部）

図 1: 出力したいコードのイメージ

```
fun mesTemp(mode, arg) =
[ "void *mesOutPut(char *message, int mode) {",
  "    if(mode == " ^ mode ^ "){",
  "        printf(\"%s\\n\", message);"
  "    } else {"] @
  indent(8, cTemp(arg)) @
[ "    }",
  "}"];
```

図 2: 通常の言語による記述

3 テンプレート記述言語を用いたソースコード出力部の記述

以上で述べてきた出力部の可読性の問題を解決するために、EasySOFTEXではテンプレート記述言語というソースコード出力部の記述専用の構文を用意した。テンプレート記述言語では、出力するソースコードの定型部はフリーテキストで記述し、可変部については埋め込む箇所と内容をHTMLのようなタグによって指定できる。これにより、ソースコードに近い記述を可能にする。テ

Hiroki TOKUOKA,Minoru TOMOBE,Toru YAMANOUCHI
C&C Media Research Laboratories, NEC Corporation

ンプレート記述言語によって前出の図1に示すようなソースコードの出力部を記述すると、図3のようになる。

```
<template mesTemp(mode, arg)>
void *mesOutPut(char *message, int mode) {
    if(mode == <value(mode)>){
        printf("%s\n", message);
    } else {
        <insert(cTemp(arg))>
    }
}
</template>
```

図 3: テンプレート記述

<template()> 〜 </template> タグで囲まれた部分にあるテキストは、出力するソースコードをそのまま記述する部分であり、改行やインデントを含めて出力される。<template()> タグの引数は、このテンプレート記述が受け取る引数であり、次に説明する埋め込みタグ内で使用出来る。

出力するソースコードの可変部は、<value()> <insert()> 等の埋め込みタグによって記述される。このうち、<insert()> は複数行にまたがる内容を埋め込むタグである。<insert()> タグは整形機能を持っており、埋め込まれる各行に対して<insert()> タグの出現位置に対応したインデントが付加される。従って、ジェネレータ開発者は、埋め込みたい場所に横位置を合わせてタグを記述するだけでよい。これらのタグにより別のテンプレートを埋め込むことも出来るので、多重にインデントのかかった複雑な構造も表現可能である。

これらに加え、パターンマッチによる制御構造やリストの展開やローカルバインドなど、関数型言語特有の様々な機能を利用するタグを用意している。一例として、パターンマッチを用いた記述を図4に示す。<switch()> タグに与えられた引数は内部の各<case()> タグとパターンマッチを行い、マッチした<case()> 〜 </case> 内の内容が出力される。<case()> 〜 </case> 内ではパターンマッチによるバインド結果を出来る。

4 議論

ジェネレータの出力となるソースコードは、仕様記述によらない定型部の中に可変部を埋め込んだ形になる場合が多い。そのため、可変部をタグで埋め込む形式で出力部を記述できるテンプレート記述言語は、定型部と可変部の組み合わせの構造を把握しやすいという点で優れている。

さらに、出力にそのまま反映されるフリーテキスト記述を使用しているため、ジェネレータのアルゴリズムやテンプレート記述言語の詳細についてほとんど知識がなくても、対象ドメインの知識さえあればジェネレータの

```
<template cTemp(arg)>
<switch(arg)>
    <case(LOG)>
    logOutPut(message);
    changeMode(mode);
    </case>
    <case(FILE(name))>
    FILE *fp = fopen("<value(name)>", "w");
    fprintf(fp, "%s\n", message);
    fclose(fp);
    </case>
</switch>
</template>
```

図 4: パターンマッチを含むテンプレート

保守を行うことができる。実際、過去にジェネレータ開発後に発生した保守・改造の要求では、ジェネレータの構造全体に関わるものは少なく、出力ソースコードの定型部の簡単な変更で対応出来た場合が大部分であった。フリーテキストによる記述の効果は、[3] でも述べられている。

関数型言語は、仕様記述やその変換過程を表現するデータ型の扱いが容易であるため、AP ジェネレータの開発に適している [4]。EasySOFTEX のテンプレート記述言語では、そのようなデータ型を扱う関数型言語特有の記述法 (パターンマッチ等) と、フリーテキストや埋め込みタグといった可読性の高い記述法を組み合わせることにより、仕様記述がどのようにソースコード出力へ反映されるかを直観的に記述することができる。

5 おわりに

AP ジェネレータ開発用言語 EasySOFTEX における、ソースコード出力部を記述するためのテンプレート記述言語の概要について述べた。EasySOFTEX を用いた AP ジェネレータの開発を現在行っているが、開発者からは通常の言語に比べソースコード出力部の記述が容易であり開発効率が上昇したとの評価を得ている。保守性の向上という観点からの評価は、今後行っていく予定である。

参考文献

[1] J.C.Cleaveland,Building Application Generators,IEEE Software,5(4):25-33,July 1988.

[2] Yamanouchi,T.,Sato, A.,Tomobe M.,Takeuchi, H., Takamura, J. and Watanabe, M. : Software Syntehsis Shell SOFTEX/S,7th KBSE Conf. , 1992.

[3] 登内 敏夫, 中島 震, "言語処理系ツールキット Rosetta による GDMO トランスレータの実現", 信学技報, TM99-10, pp.63–70, Jun. 1999

[4] 友部、徳岡、山之内 : "ソフトウェア自動合成シェル - EasySOFTEX - (1) - 特定ドメイン向け AP ジェネレータの開発プロセスと支援環境 –", 情報処理学会第 59 回全国大会,3W-04,1999