4L－2

オブジェクト指向フレームワークによる異なるシステム間での表現変換 (2) 実現

竹内 幹雄　金山 恵子
日本アイ・ビー・エム (株) 東京基礎研究所

1 はじめに

(2) 実現は主に設計について述べる。任意のシステム表現間での変換を可能にするため、まず (1) 分析で抽出した共通オブジェクトモデルから、中間表現である抽象構文木のノードを決定する。次に分析結果から得られた要求を基に中間表現とシステム表現間を双方向変換する C++ によるフレームワークの設計を提案する。

2 フレームワークに対する要求と設計方針

分析で得られたフレームワークに対する要求は以下の通りである。

1. 直接記述できない概念の変換
2. 複数の表現ができる概念の変換
3. システムの追加への対応

1は変換の前段階として入力側独自の概念を、出力側の概念を用いて同等の表現に変換し構文木を変形する (図 1)。

2は複数の変換アルゴリズムを表現するクラスをあらかじめ用意し、それを変換手続内部で用いることで、実行時に変換方法を取り替えられるようにする。利用者が独自のアルゴリズムを追加することも可能にする。

3の実現には、システム独自の要素を扱えるよう中間表現を拡張でき、それをフレームワークから無理なく扱えなくてはならない。そのため共通オブジェクトモデルに相当する部分のノードのクラスを定義する。クラスの候補は分析で得られた共通構文要素から、その大きさや意味のある単位を考慮して決定する。システム独自の要素は共通ノードクラスのサブクラスを定義し、属性を追加する。

追加した要素の変換を可能にするために、変換手続はノードクラスのメソッドとして実現する。変換メソッドは仮想関数にし、必要な拡張はサブクラスで行なう。変換メソッドが行なう処理には、そのノードの属性の変換と、サブノードへの処理の伝搬があるが、これらを一つのメソッドで実現するとそれらの一部の変更の際にも全てを記述し直す必要がある。これを避けるために変換メソッドをテンプレートとして与え、ノードの属性の変換を別の仮想関数にすることで、変換メソッド全体を再定義しなくても必要な部分だけを変更できるようにする。

ノードをクラスにし、変換手続をそのメソッドにしたことにより、ノードの追加変更の影響する範囲がそのクラスに限定される。そのため従来のコンパイラに新しい構文要素を追加する手間に比べて格段に小さな労力で実現できる。

新しいシステム表現を読み込むにはパーサも拡張できなくてはならない。そこでパーサもクラスにし、パースメソッドを仮想関数にする。フレームワークの利用者は追加システム用のパーサを作成し、そのサブクラスとすることでフレームワークから利用可能にする。

さらに入力側のパーサクラスと出力側のノードクラスの結合を実行時まで遅らせ、フレームワークの再利用性を高めるため、ノード作成のためのクラスを用意する。ノードの種類を指定すれば必要な変換メソッドを持ったノードクラスのインスタンスが得られるようにし、ノード作成時のクラス指定を抽象化する。

3 フレームワークの概要

本フレームワークは次の 4 種類のクラス群からなる (図 2)。

1. パーサ及びストリームクラス
2. ノードクラス
3. ストラテジクラス
4. ファクトリクラス

1のパーサクラスは各システム用のものをあらかじめ用意する。それらはシステム別のスキャナクラスが切り出すトークンを基にノードクラスのインスタンスからなる抽象構文木を返す。スキャナクラスはファイルや標準入出力などを抽象化したストリームクラスを用いる。ストリームクラスはエラー処理などのために現在行、桁などを記録するストリーム情報クラスを持つ。

Object-Oriented Framework for Representation Transformation (2) Prototyping
Mikio Takeuchi and Keiko Kanayama
IBM Research, Tokyo Research Laboratory

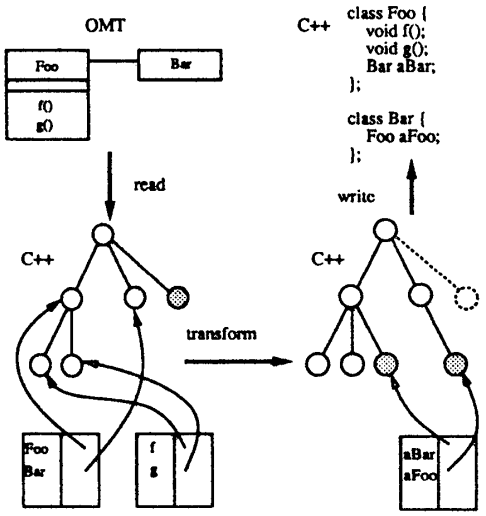

図 1: フレームワークによる変換手順

2は共通ノードクラスと各システム用のノードクラスからなる。後者はシステム表現への変換メソッドを持つ。

3は要求2のアルゴリズムを表現するクラスである。

4は要求3のノード作成のためのクラスである。

4 関連研究

Smalltalk のコンパイラのためのフレームワークに [1] がある。これはプロセッサの違い (RISC,CISC の別やレジスタ数) により異なる最適化、コード生成のアルゴリズムを表現するクラスをあらかじめ用意し、利用者が選択できるようにしている。ただし単一言語を対象としているため中間表現の拡張を考慮していない。またアルゴリズムを表現するクラスはノードクラスとは別に設けられている。

コンパイラ生成系としては [2] がある。これは中間表現を抽象構文木にし、コンパイラの各フェイズをノードクラスのメソッドで実現する方法である。ただし変換手続の変更をメソッド単位でしか考慮していない。

同様にパーサ生成系としては [3] がある。これは Yacc に類似した文法定義から抽象構文木を作成するパーサクラスを自動生成するツールである。ノードクラスの候補は文法定義の際に明示的に指定する。読み込んだプログラムの変換については言及していないが、本フレームワークのパーサを記述するのに利用できる。

5 おわりに

オブジェクト指向をサポートするシステム間のモデルの共通性に注目し、共通オブジェクトモデルを基本とする抽象構文木を中間表現とし、システム表現との

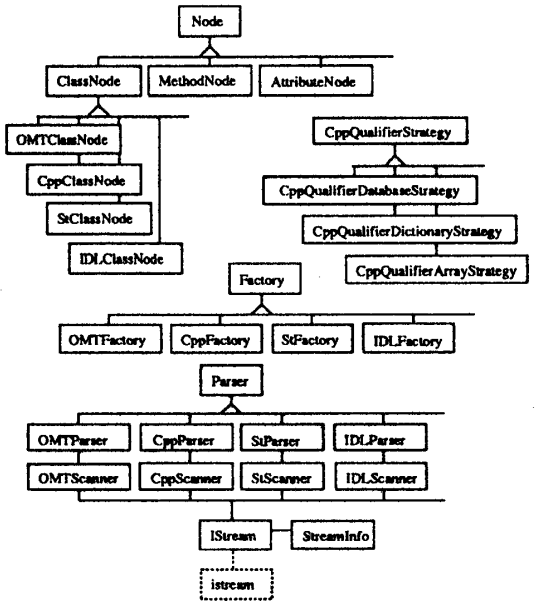

図 2: フレームワークを構成するクラス群

間を変換するフレームワークについて議論した。従来のコンパイラフレームワークと異なり、入力側に特定の言語等を仮定せず任意のシステム表現間での双方向変換を目指す。さらに記述力の高い表現間での変換で生じる、直接記述できない概念や複数の表現ができる概念の変換について考察した。新たなシステムへの対応を最小限の労力で行なうため、変換手続をノードクラスのメソッドにし、さらにノードの属性の変換とサブノードへの処理の伝搬を独立に変更できる方法を考案した。

本フレームワークで用いた中間表現はシステム表現の変換だけでなくブラウザやアナライザ、リファクタ、文書化ツールなどのプログラム情報を扱うツールにも利用できる。中間表現をデータベースにいれ、ツール間で共有する方法は [4] などで議論されている。

参考文献

[1] R. Johnson: The RTL System, OOPSLA94 OO Compilation Workshop.

[2] T. Budd and T. Justice: An Object-Oriented Compiler Construction Tool Kit, OOPSLA94 OO Compilation Workshop.

[3] B. Zino: Yacc++ and the Language Objects Library: An Automated Parsing Framework, OOPSLA94 OO Compilation Workshop.

[4] 北山文彦: オブジェクト指向開発のための CASE アーキテクチャの設計とプロトタイプ, 日本ソフトウェア科学会第 12 回大会論文集, 1995.

[5] E. Gamma, R. Helm, R. Johnson and J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.