オブジェクト指向ソフトウエア開発におけるプログラム理解支援

6D−3 中村 宏明 安田 和 大平 剛 三ッ井 欽一
日本アイ・ビー・エム(株)東京基礎研究所

1 はじめに

オブジェクト指向ソフトウエア開発では共通のモデルを用いて分析・設計・実装が行なうことができる。各フェーズが継目なくつながることで、ソフトウエアの変更・修正・追加・再利用などの容易化が期待できる。この利点を生かすためにはフェーズ間を逆方向へ戻れることも重要で、特にプログラムを理解することによって分析・設計に関する情報を得ることが、反復的・進化的なソフトウエアの開発の鍵になる [1]。

ところがオブジェクト指向の主要な性質である継承と多相性は、プログラムを理解することを困難なものにしている [2]。例えば次のような C++ の関数呼び出しを理解するために、関数を定義している場所を求める過程を考える。

```
table.put(token);
```

(1) table がデータ・メンバーである場合、この関数呼び出しを含むメンバー関数が宣言されているクラスの継承階層の中で table が定義されているクラスを探し、table の型を知る。(2) 関数の多義性を解決するために、同様にして token の型を調べる。(3) table のクラスの継承階層の中で、メンバー関数 put を定義しているクラスを探す。この他に C++ には、マクロ、多義演算子、自動型変換、例外処理など、プログラムを書くことを簡潔にするが、読むことを複雑にする要素が多く含まれている。

したがって、オブジェクト指向の利点を生かしたソフトウエア開発を行なうためには、プログラムの理解を支援する環境が不可欠である。本稿では、まずプログラムの理解支援に対する要件を考察し、次に、我々が作成した C++ ソース・コードの理解支援システムの構成と、その使用例を紹介する。

2 プログラム理解支援に必要なもの

プログラムに関する知識 与えられたプログラムの構文・意味に関する知識が検索できる状態で保持されている必要がある。このような知識にはシンボル・型・参照関係などに関する情報が含まれる。

抽象化 プログラムの理解は、記号列として表現されるプログラムを人間に分かりやすい形式に変換する抽象化プロセスとしてとらえることができる。

視覚化 プログラムを抽象化した結果を人間が理解するためには、それを視覚的に具体化することが必要である。

理解の方法に関する知識 目的等に応じて多岐に渡るプログラムの抽象化の方法を定式化・保存しておいて、必要な方法を理解のガイドに利用できることが望ましい。

対話 大まかな理解と詳しい理解を場面に応じて選択することや、実装の際に欠落した分析・設計の情報を復元することなどのために、プログラムの理解過程には人間の介在が不可欠であり、対話的な環境が望まれる。

Program Understanding Support for Object Oriented Software Development
Hiroaki Nakamura, Kazu Yasuda, Tsuyoshi Ohira, and Kin'ichi Misui

以上のような検討から、プログラム理解支援のためのモデルは図1のようになる。

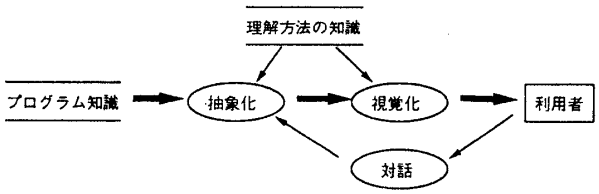

図 1: プログラム理解支援のためのモデル

3 理解支援システムの構成

我々は前節で述べた要件を考慮して、図2のような理解支援システムを作成した。また、図3にユーザ・インタフェースの表示例を示す。

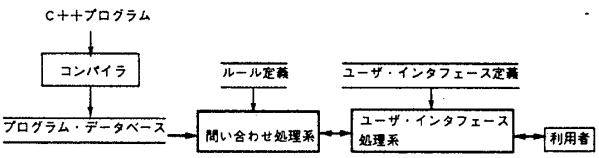

図 2: プログラム理解支援システム

プログラム・データベースには、コンパイラによるプログラムの解析結果を検索しやすい形式に変換したものを格納しておく。システムは、プログラムに関する情報を抽象化するのに用いられる問い合わせ処理系と、情報の視覚化・ユーザとの対話を実現するユーザ・インタフェース処理系の2つからなっている。問い合わせ処理系には Prolog インタプリタ [3] が、ユーザ・インタフェース処理系には GUI 記述言語のインタプリタ [4] がそれぞれ組み込まれていている。さまざまな理解の方法をこれらの言語で記述しておくことによって、システムが柔軟にプログラムの理解をサポートできるようになっている。

4 システムの使用例

我々が実現したプログラムの理解を助けるための機能をいくつか紹介する。

4.1 関数呼出しの追跡

プログラムの機能的な側面を理解するためには関数の呼出し関係を知ることが重要である。しかし、第1節で述べたように、オブジェクト指向プログラムでは関数呼び出しの追跡は複雑な操作を必要とする。

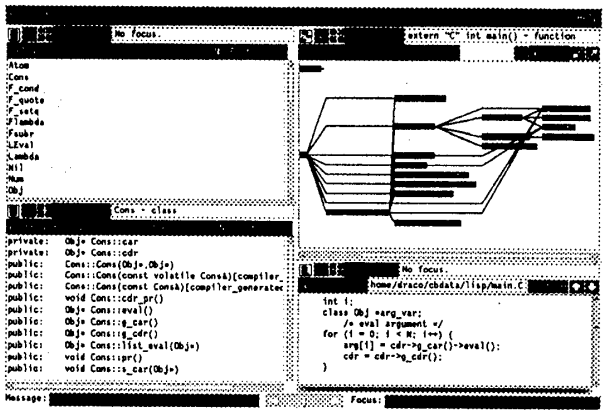

図 3: ユーザ・インタフェース画面

我々は次のような複数の形式で関数呼び出しを表示することによって、利用者の理解をサポートしている。

- 関数呼出し連鎖のグラフ表示
- 呼出しの深さなどの条件にマッチする関数の一覧
- プログラム・テキスト上での関数の宣言・定義・呼出し場所

さらに、これらがハイパーテキスト状に結合されていて、関数名をマウス・クリックすることによって他の表現に移動できる。

4.2 クラスの複数の見方への対応

あるクラスについて知ろうとすると、メンバーなどクラス内で宣言されているものを調べる必要がある。しかし継承のために、そのクラスを単独で見ても理解できないことが多い。またアクセス指定のために、視点によってクラスの見え方が異なる。

そこで、我々は次のような機能によってクラスの理解をサポートするようにしている。

- 継承階層のフラット化
- アクセス指定・フレンド関係などによるフィルタリング

さらにクラスの理解を容易にするために、次のような視点をあらかじめ組み込んでいる。

- クラスの利用者の視点
- クラスの実装者の視点
- 導出クラスの実装者の視点

4.3 クラスの全体ー部分関係図の簡略化

クラスの全体ー部分関係はプログラムの構造を理解するのに重要であるが、これを「あるクラスと、そのデータ・メンバーの型として使われているクラス」というプログラム・テキスト上の関係として定義すると、繁雑で理解しにくい結果しか得られない。これは、1 対 1 でない関連の実装や効率化のために導入される副次的なクラス [5] を、本質的なクラスと区別できないためである。

あるクラス・ライブラリがプログラムで使用されていることを仮定すると、クラスの使い方のパターンに関する知識を用いて、より高いレベルでの理解が可能になる。例えば、データ・メンバーの型に基づいた全体ー部分関係が図 4(a) のようになるプログラムにおいて、(1) 基本的なデータ型を提供するクラス `String` は良く理解されているので表示しない、(2) 集合を実装する目的で用いられるクラス・テンプレート `Set` はクラス間の線分で表現する、という規則を付加してグラフを構成すると、図 4(b) のように本質的な全体ー部分関係だけが抽出できる。

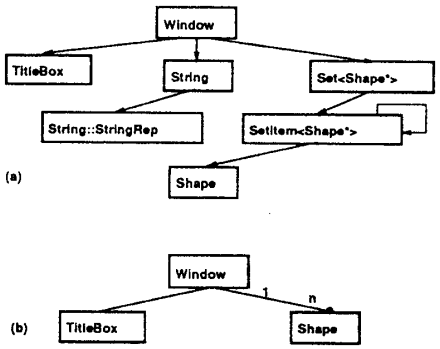

図 4: クラスの全体ー部分関係図の簡略化

4.4 スタイル検査

より抽象的なプログラムの理解の方法として、プログラムがあるコンベンションに従って書かれているかを調べることなど、プログラムのスタイルを検査することがあげられる。このような検査の一例として、オブジェクト指向プログラムのモジュール性を保証するための Demeter の規則 [6] を自動的に検査できるようにした。

5 おわりに

オブジェクト指向プログラムの理解の難しさを解消することを目的としたシステムはいくつかあるが [7][8]、我々のシステムの特徴は、理解の方法を知識としてため込んでいって、機能を変更・拡張していくことが容易なことである。本研究の成果は製品化され、実際のソフトウエア開発に利用されて役に立っている。

参考文献

[1] A.Goldberg: Programmer as Reader, *IEEE Software*, September, 1987, pp 62-70.

[2] N.Wilde and R.Huitt: Maintenance Support for Object-Oriented Programs, *IEEE Trans. Software Eng.*, Vol.18, No.12, 1992, pp 1038-1044.

[3] H.Nakamura: Embeddable PROLOG Interpreter for Data-Intensive Applications, IBM Technical Report, TR-74.091, 1992.

[4] 大平, 三ッ井: 拡張可能な C++ソースコード・ブラウザ - グラフィカルユーザインタフェース, 情報処理学会研究報告,93-PRG-10,10-6,1993.

[5] J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, and W.Lorensen: Object-Oriented Modeling and Design, Prentice Hall, 1991.

[6] K.J.Lieberherr and I.Holland: Assuring Good Style for Object-Oriented Programs, *IEEE Software*, Septmber, 1989, pp 38-48.

[7] M.Lejter, S.Meyers, and S.P.Reiss: Support for Maintaing Object-Oriented Programs, *IEEE Trans. Software Eng.*, Vol.18, No.12, 1992, pp 1045-1052.

[8] J.E.Grass: Object-Oriented Design Archaeology with CIA++: *Computing Systems*, Vol.5, No.1, 1992.