3R-7

オブジェクト指向言語における ブラウジングの方法について

明石孝祐　青柳龍也　河野眞治　田中英彦
(東京大学　工学部)

1．はじめに

Smalltalk やFlavors などのオブジェクト指向言語のプログラミング環境はエディタ・デバッガ・ブラウザなどを統合したシステムを提供しているが、クラスの管理に関しては未だ十分でない。オブジェクトはモジュール性の高い記述であるから、それらの間の関係をなんらかの手段で把握することが重要である。

本稿では、プログラミングを多角的に支援するORAGAの一部として、ソースコードから得られる情報を、いろいろなビューからブラウジングすることを考える。

2．　目標

ブラウジングには、複雑な情報を活かす上で構造化された表示が欠かせない。多くの情報に対してその全体構成を見渡したり、必要な情報に素早くアクセスでき、不要な情報は見せないこと、一つではなく多くの表示の仕方が目的に応じて選択できることが必要である。

オブジェクト指向のプログラミングで特に視覚化して有効なのは

1)　オブジェクトへの分割と、その間の関係付け

おおまかな機能の単位としてオブジェクトをいくつかのグループに分けることで、プログラミングの方針をたてやすくする。例えば、Smalltalk のMVC モデルのように分ける。一般的な分割の戦略を考えることはできないにしても、分割したときの関係を記述しておけば，全体の構成を把握したり、再分割するときの情報として役立てられる。

2)　インヘリタンスの関係

ソースを読むときインヘリタンスが何段にも及ぶと、メソッドやスロットがどのレベルに定義されているかがわかりづらい。また、インヘリタンスの役割にも、データ構造を継承するとき、新しい構造の内部構造として利用するとき、機能だけを継承したいときなどがある。

従って、インヘリタンスの関係といってもスロットのみ、メソッドのみに注目した見方などがほしい。

3)　クラスのメソッド・スロットの参照関係

既存のクラスのメソッドに対しその定義を参照することがある。逆に、オブジェクトの定義に対しそのメソッドやスロットをアクセスするクラスを調べることがある。また、オブジェクトを定義するとき、他のオブジェクトはスロットやメソッド名だけを決めておき、後でその定義を埋めていくことも多い。ひとつのメソッドを定義する際参照するクラスは比較的少ないので、出来るだけ同時に見えるのがよい。

これらを自由にたどるインターフェースを提供することが目標である。情報はソースコードだけでは必ずしもわからないが、プログラマはなんらかの意図を持っているはずである。これをプログラマに明示することにすればよいので、ここではそれを得る方法については問わないことにする。

その他再利用のため既存のライブラリの中から自分の望む仕様のクラスを探すために、ＯＲＡＧＡでは名前に着目したクラスの管理を考えている[1] 。そのためには、名前を環境全体に渡って管理するデータベースのようなものが必要である。

3．　方針

2の目標を果たすためには、それらを効率良く選択しながら見る機構が必要である。従来の構造エディタではそこまで扱わなかったし、ウィンドウシステムはサポートする表示のレベルが低すぎる。

ここでは、その際に必要とされる基本的な機能を以下に考察する。

1)　表示要素

①　アイコン化と本体の呼び出し

アイコンといっても主な情報は名前である。その本体の内容を開いたり閉じたりできる。

Browsing for object-oriented languages
Takahiro AKASHI, Tatsuya AOYAGI, Shinji KONO, Hidehiko TANAKA
Tokyo University

② テンプレート機能

構造に特定のパターンがある場合、必要な部分のみをながめることができる。オブジェクト定義の構文やインヘリタンスにおけるスーパー・サブクラスなどである。

2） 関係の表示

ユーザは目標に達するために、関係をたどっていく。そのために、図1のようにインヘリタンス、メソッド参照関係を表示する。しかし、

① 一つ以上の見方

をサポートする必要がある。たとえ同じ構造を見るにしてもユーザが必要としている情報は目的によって異なる。たとえばインヘリタンス関係を見る場合、クラス間の関係を知りたいときにはネットワークで、メソッドの検索順が問題なときは表であらわす。

2） ウィンドウ間の表示の仕方

① 再帰的な呼び出し

どのウィンドウ上のクラス名からでも図2の情報を呼び出せるようにする。このときに1）①の操作を行うと、関係だけはそのまま表示される（図2）。

② 画面上の最適配置

多くの情報がある場合、同時に見たいものは決してオーバーラップしない。例えば、オブジェクトの定義を開いたときには他の定義表示と重ならないようにする。逆に、同様な操作の繰り返しでは、途中がある程度重なることが許される。

これは出来るかぎりシステム自動的に行うようにするが、ユーザも指示ができる。

3） 効率良く見せること

① グルーピングと不要部分の省略

関係をたどっていく操作を繰り返すと画面がウィンドウで一杯になる。このような場合、たどった途中の経過は不要である。必要になったとき引き出せるような手段さえ残してあれば、それらをグルーピングして画面上では縮小しても構わない。

このとき、2）で示された関係（矢）は残ったままである（図3）。

4） 関係に対するマッチング

ユーザが必要に応じて表示の組み合わせを定義できる。それに対応して操作もマクロな形で定義でき、煩わしさをなくす。図3に示した操作を繰り返し行うことは、あるメソッドが影響を及ぼす可能性のあるクラスを求めることになる。

4． おわりに

現在、2・3における検討をさらに進め、システムを設計している。しかし、本稿では、実行時の支援・デバッグについては考察の対象としなかった。これは言語の処理系に依存するところが多く、また，動的な表示が難しいからでもある。デバッガの研究[2] とも合わせ、今後考慮していきたい。

<参考文献>

[1] 情報処理学会第35回全国大会，3R-8 (1987)
[2] 情報処理学会第35回全国大会，3R-6 (1987)

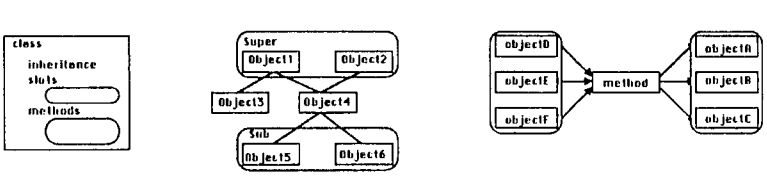

図1．基本の表示

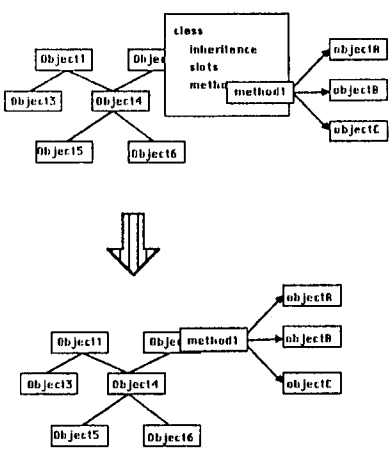

図2．定義を閉じたとき

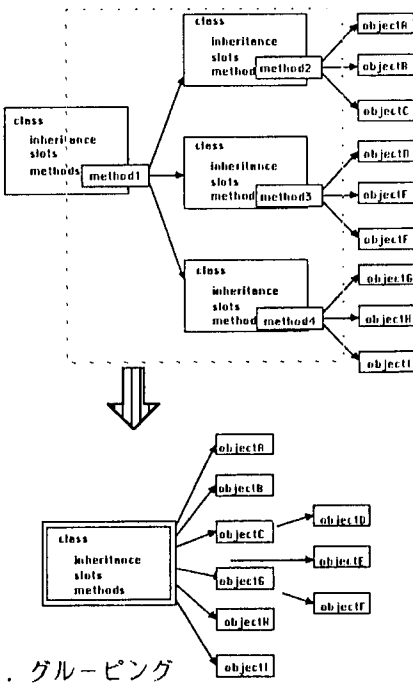

図3．グルーピング