ソフトウェア工学 107－6
（1996. 1. 18）

Cプログラムに対するカプセル発見手法とその支援ツールについて

小田 章夫　鯵坂 恒夫

京都大学工学部

一般にオブジェクト指向分析・設計とオブジェクト指向言語によるプログラムの間の距離の方が，従来の構造化分析・設計と既存言語によるプログラムより近いといえる．よって後者のプログラムに対しリバースエンジニアリングを行い前者のプログラムを得ることができれば，プログラムより設計を読み取ることが行いやすい．

本稿では，対象となる C プログラムの型とグローバル変数および関数に注目し，それらの依存・参照関係を定性的，定量的に解析し評価することにより，オブジェクトのカプセルを導き出し，オブジェクト指向的デザインを復元する手がかりを得る．

A Method and Tool for Extracting Potential Capsules in C programs

ODA Akio　AJISAKA Tsuneo

Faculty of Engineering, Kyoto University
Yoshida honmachi, Sakyo-ku, Kyoto 606-01, JAPAN

The distance between OOA/OOD and OOP is usually shorter than that between SA/SD and conventional languages. Therefore, if a program in a conventional language can be translated into that in OOP through reverse engineering, the design is more easily understood and recovered.

This paper describes methods to obtain a clue to the OO design recovery. They are analyzing qualitative dependency among types, global variables and functions, evaluating quantitative relations among them, and identifying and extracting capsules in C programs.

1 はじめに

ソフトウェアの生産性と品質を確実に高める技術の一つとして，再利用技術が挙げられる．再利用される部品としてはモデル，ルール，プログラム，データなど様々である．しかしながら，現在までの再利用技術は，プログラムならプログラムとして同一レベルで再利用するものであり，特に再利用する部品をそれ自身がもつ抽象度より高いレベルに変換（デザインリカバリ，リバースエンジニアリング）して再利用することは難しいのが現状である．

最近ソフトウェアの開発現場でも関心を集めているオブジェクト指向方法論においても，再利用がその特徴の一つである．しかしながら，これも例えば C++ によるプログラミング工程なら再利用する部品も同じ C++ のレベルのものである．一方，C++ では，世の中に豊富に存在する C で書かれた抽象度の低いプログラムを C++ で再利用することができるが，これは Cのまま使えるというだけであって，そのプログラムを C++ の抽象度レベルまで上げて部品として再利用することを明確に意図しているわけではない．

これに対して，既存の言語で書かれたプログラムからオブジェクト指向的なデザインを導き出すことに関する従来の研究には，その大まかで概念的な手法を述べたもの [1] や，対象となるプログラム内のグローバル変数と関数のそれぞれの参照関係だけから階層的システム構造を見つけ出すもの [2] 等がある．

本稿では，対象となる C プログラムの型とグローバル変数および関数に注目し，それらの依存・参照関係を定性的，定量的に解析し評価することにより，オブジェクトのカプセルを導き出すという研究について述べている．

一般にオブジェクト指向分析・設計とオブジェクト指向言語によるプログラムの間の距離の方が，従来の構造化分析・設計とC言語等によるプログラムより近いというのは確らしいので，本稿はさらに上位のオブジェクト指向的デザインを復元する手がかりともなる．

2 研究の概略

2.1 対象とするプログラム

対象となるプログラムが，非構造的であったり，また関数毎に機能的に書かれていなかったりした場合，そのようなプログラムをオブジェクト指向的に解析するということは，人間が行っても非常に困難なことである．

そこで本研究では，対象となるCプログラム自体がある程度オブジェクト指向を意識して機能的に書かれているという前提をする．

2.2 オブジェクト要素の候補

前項に述べた前提のもとでは，プログラムは機能的に分割されモジュール化されている．従って，次のようなプログラムのグローバルなスコープから見ることができるものを，オブジェクトを構成する要素の候補として考える．

- 構造を持った型 (struct, union)
- ファイルスコープで定義される変数（以下，グローバル変数とする）
- 関数

以降ではそれぞれの集合を T, V, F と表しており，また，特に指定していない限り型とは struct または union 型を指す．

2.3 操作の流れ

本研究においては，上に述べたような型，グローバル変数，関数に対しそれぞれの依存

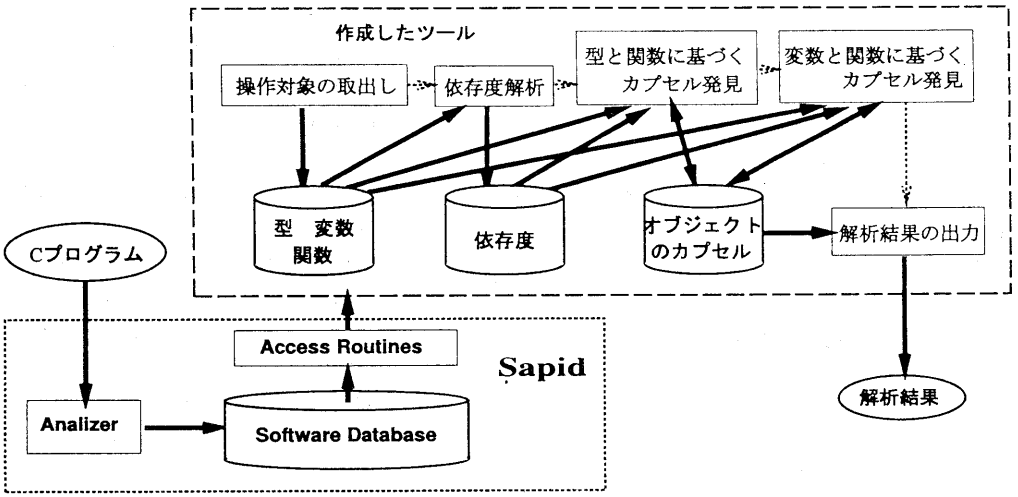

図 1: ツールの構成

度なるものを静的に解析・評価し，その値に基づいて相互に依存度の高いもの同士を組み合わせて一つのオブジェクトとするという手法をとっている．

本研究における操作の大まかな流れは次のようになっている．

1. Cプログラムに対する言語的な構文解析・意味解析．
2. 型，グローバル変数，関数の抜き出し．
3. 関数に対する定性的，定量的な依存・参照の解析．
4. 型と関数に基づくカプセル発見．
5. 関数とグローバル変数に基づくカプセル発見．

2.4 ツールの構成

操作の内 1. については Sapid [3] を使用している．作成しているツールはこの Sapid の元で動くツールとなっている．

Sapid は 細粒度のソフトウェアリポジトリに基づいたツールプラットフォームである．作成したツールでは，対象となるソフトウェアを I-model, C-model と呼ばれる Sapid のソフトウェアモデルに基づき解析を行い得られた実体や関連情報を格納する SDB(Software DataBase) と，SDB への基本的なアクセスルーチン AR (Access Routines) を使用している．

2.3 に基づき作成したツールの構成を図 1 に示す．

3 関数に関する解析と評価

個々の関数において，自分の外に見える要素，すなわち次のような点ついて解析および評価を行っている．

- 戻り値の型
- 引数の型．
- 引数への依存度．

- グローバル変数への依存度.
- 関数への依存度.

ここでの依存度とは，変数については関数がその変数をどの程度参照し，また書き換えているか，関数ならばどの程度その関数を呼び出しているかということを評価した値としている．

3.1 依存度の評価

調べたい関数がどのようにどの程度，型や変数，関数に依存しているかということを正確に知ることは容易なことではない．特にプログラムを書いた関係者以外がプログラムだけを見て知るのはほとんど不可能である．

そこで，それらの依存の度合いを擬似的に評価することになるが，ここでは問題が２つ考えられる（ここで述べる依存関係とは関数全体としての定性的な関係である）．

まず，依存関係を正確に抽出できるか，という問題がある．左辺値，右辺値とも単なる変数や定数であるような代入文などだけであれば，左辺値か右辺値であるかという関係だけで依存関係が求まるので問題はない．しかし，これに条件式が入ると依存関係は唯一には決まらなくなる可能性がある．それぞれの条件について依存関係そのものは求めることができるが，それらを組み合わせて関数全体としての正しい依存関係を決めるためには，動的な解析が必要である．また，ポインタや配列によるエイリアスの問題が発生する．これらを使われると，左辺値は単なる記憶領域であってそれらはほとんどの場合動的に決まるものである．Cプログラムではこれらの動的にしか解決できない問題があり複雑である．

もう一つの問題は，正確な依存関係が求まったとしても，それをどのように定量的に評価するかということである．数多く実行され得る文が，あまり実行されない文よりその関数内での評価が高いということは必ずしも成り立たない．どの関係をどのように評価するかということはやはりプログラムを書いた関係者以外知ることのできないものだと思われる．

このようにいろいろな意味で定性的，定量的な評価は主観的である．従来の研究 [2] でも依存関係を正確に求めようとしているものはなく，擬似的に，それも関数からどの関数やどのグローバル変数が参照されているかといった定性的な関係だけを見ている．

本研究でも依存関係を詳細に調べることはしないが，次のような観点により定量的に擬似的な依存度を求めている．

- 関数内の文単位での解析
- 文が実行される回数が多いほど高く評価
- 左辺値として使われる方が右辺値より高く評価
- ポインタによる左辺値はそのベースとなるポインタ変数を左辺値と見なす

これらにより次のように依存度を算出している．

依存度関数 $D(f,g)$ の算出 ただし $f \in F$, $g \in G_f$ であり，G_f を V と F および f の引数を含むローカル変数の集合

1. 全ての g に対して $D(f,g) = 0$
2. f の文の全集合を抜き出す．これを $S(f)$ とする．
3. 文の重み関数 $L(s)$ $(s \in S(f))$ を作成する．これは，その文 s が関数の一番外側のブロックから見て n 重のループ構造の中にあるとき $L(s) = n+1$ としている．
4. 関数全体の文の重み $L_{total}(f) = \sum_{s \in S(f)} L(s)$ とする．

5. 文 s での各 g に対する依存度を表す関数 $D_{st}(s,g)$ を作成する．これは s の中で g がどのように使われるかにより，
   左辺値として ⇒ $D_{st}(s,g)=2$
   右辺値として ⇒ $D_{st}(s,g)=1$
   使われない ⇒ $D_{st}(s,g)=0$
   としている．また，ポインタを使った左辺値の場合，その左辺値を表す式の中で有効なもっとも先に書かれるポインタ変数を擬似的に左辺値と判断することにしている．

6. これらにより次のように求める．

$$D(f,g)=\frac{1}{L_{total}(f)}\sum_{s\in S(f)}(D_{st}(s,g)\times L(s))$$

さらに，関数の戻り値のその関数に対する依存度を算出する．本研究では，以下のように return 文に現れるもっとも依存度の高い変数や関数の依存度を戻り値の依存度としている．

依存度関数 $D_{return}(f)$ の算出

1. $S(f)$ の中の return 文の中で使われている全ての変数と関数を抜き出す．これを $G_{return}(f)$ とする．
2. $D_{return}(f)=\mathbf{max}_{g\in G_{return}(f)}D(f,g)$

4 型と関数に基づくカプセル発見

型と関数に基づくオブジェクトのカプセルの発見は次のような動機に基づいている．

- struct や union 型があれば，それらに対する何らかの操作関数がある．
- それらの関数ではその型を戻り値か引数の型として，またはそれらポインタが示す記憶領域の型として使っている．

例えば，図 2 に示されるCプログラムを対象にして，をこの節に述べる操作で図 3を得ようという意図である．

```
typedef struct complex {
    double    r, i;
} complex;
typedef struct stack {
    struct complex s[];
} stack;
typedef struct queue {
    struct complex q[];
} queue;
complex add(complex, complex);
complex sub(complex, complex);
void push(stack *, complex);
complex pop(stack *);
void enqueue(queue *, complex);
complex dequeue(queue *);
```

図 2: 対象プログラムの例

```
class complex {
    double    r, i;
public:
    complex add(complex, complex);
    complex sub(complex, complex);
};
class stack {
    complex s[];
public:
    void push(complex);
    complex pop();
};
class queue {
    complex q[];
public:
    void enqueue(complex);
    complex dequeue();
};
```

図 3: 型と関数に基づくカプセルの例

4.1 型間の依存関係

struct や union 型は，そのメンバとして更に他の struct 型等を持っている場合がある．この場合は操作関数がそのメンバの型も戻り値や引数として持っているかも知れない．このような時は，型の依存関係を用いて解決できると考える．もし型 A が型 B をその構成要素として持つならば，A は B の supertype，B を A の subtype と呼ぶことにする．もしそのような依存関係をもつ型同士を戻り値や引数の型としてもつ関数では，supertype の方の型の操作関数と考えた方が自然である．

また，この supertype という関係には推移律が成り立った方が良い．しかしながら型 A と型 B にサイクリックな関係があると，型 A と型 B は全く同じ形の依存関係を持つことになってしまい，どちらが supertype で subtype なのかということが分からなくなってしまうので都合が悪い．実際の型の依存関係はそのようなサイクリックな関係は省く必要がある．

4.2 カプセル発見のアルゴリズム

これらの考え方から，型と関数に基づくオブジェクトのカプセル発見のアルゴリズムは次のようになっている．また，これ以降発見されるオブジェクトのカプセルを $C(x)$ と表記する．

1. 型の依存関係 $N(T,T)$ を計算する．これは全ての $t,t' \in T$ について，t が t' の supertype であれば $N(t,t') = 1$，そうでなければ 0 とする．

2. 関数と型の関係関数を $R(f,t)$ とする．これは次のように初期値を求める．

   - 全ての $f,t,$ について $R(f,t) = 0$ とする．
   - f の戻り値の型が t ならば，
     $R(f,t) = D_{return}(f)$
   - f の全ての引数 a でその型が t あるならば，
     $R(f,t) = R(f,t) + D(f,a)$

3. 型の依存関係を削除する．ある f で $R(f,t) > 0$ であるなら，$N(t,t') = 1$ である全ての t' について，$R(f,t)$ に $R(f,t')$ を足したの後 $R(f,t') = 0$ とする．

4. 型 t 一つにつきオブジェクトのカプセル $C(t)$ を割り当てる．

5. $f \in F$ においてもっとも大きな値を持つ $R(f,t)$ がその次に大きい値を持つ $R(f,t')$ の α 倍より大きければ，f を t の割り当てられている $C(t)$ のメンバとし，F より f を除く．

6. 現在点での全ての $f \in F$ と t に対し次の計算を行う．
   $R(f,t) = R(f,t)$
   $$\times(1 + \sum_{g \in C(t)} (D(f,g) + D(g,f)))$$

7. ある f において $R(f,T)$ の中でもっとも大きい値を持つものを $R(f,t)$ であれば，f を $C(t)$ のメンバとし，F より f を除く．

図 2 を対象とした 2. から 4. までの操作例を図 4に示す．

6. は 5. で候補となる型が幾つか挙げられた時に，それらを解決するために 5. によりオブジェクトのメンバ判定された関数に対する相互の依存度を加味することにより，候補を一つに絞ることを意図したものである．

関数	型			カプセル
	complex	stack	queue	
add	1			I
sub	1			
push	1 → 0	1		II
pop	1 → 0	1		
enqueue	1 → 0		1	III
dequeue	1 → 0		1	

stack と queue は complex の supertype

図 4: 型と関数に基づくカプセル発見例

5 関数とグローバル変数に基づくカプセル発見

関数とグローバル変数間の参照や呼び出しの関係に注目し，もしグローバル変数がある関数の集合からしか参照されていないならば，それらをまとめてオブジェクトとして考えることができる．

この節での手法の概要を 図 5 に示す．

5.1 カプセル発見のアルゴリズム

まず，関数とグローバル変数に関する関係から 4 節での操作の結果えられた $C(t)$ に関する追加的な処理を行う．

8. v がどの唯一の $C(x)$ からのみ参照されていれば，v は $C(x)$ の (静的) メンバと判定し，v を V から除く

9. v が複数の $C(x)$ から参照されていれば，独立なグローバル変数と判定し，v を V から除く

10. $C(x)$ のメンバとされている v を参照する f が存在すれば，f を $C(x)$ のメンバと判定し，f を F から除く．全ての $C(x)$ に対してそのような f が存在しなければ次へ，もしあったなら，8. に戻る．

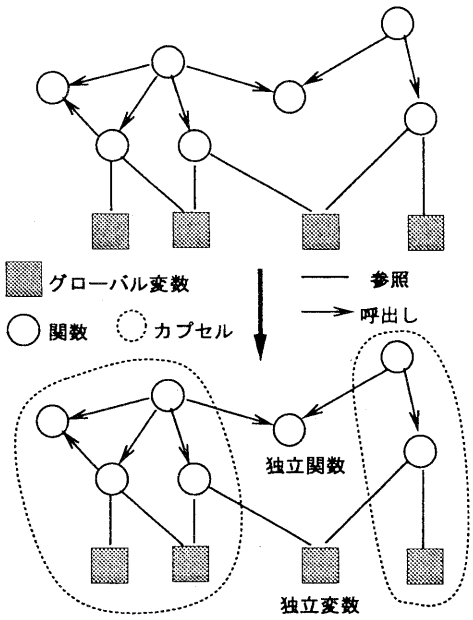

図 5: 関数とグローバル変数に基づくカプセル発見例

ここまでで，型とその型を使用している関数とそれらに付随する グローバル変数，および utility 的な関数についてほぼオブジェクトのカプセルを見つけ出している．

この後，残った関数やグローバル変数に対して，純粋にそれらの参照関係のみから新たなオブジェクトのカプセルの発見を行う．

11. ある v を参照する f を全てまとめてオブジェクトのカプセル $C(v)$ とし，v, f をそれぞれ V, F から除く．このようにして新な $C(v)$ が見つかれば 8. に戻る．

この時点で全ての V について判定をし終えている．最後に残った関数についてそれが唯一のオブジェクトの utility 関数かどうかを調べる．

12. 唯一の $C(x)$ からのみ呼び出され，$C(y)(y \neq x)$ から呼び出さない f を $C(x)$ のメンバと判定し，f を F から除く．

最後に，

13. 残った関数 $f \in F$ を独立な関数とする．

6 おわりに

本稿では既存のCプログラムの型とグローバル変数および関数に注目し，それらの依存・参照関係を定性的，定量的に解析し評価することによって，それらをふるい分け，オブジェクトのカプセルを導き出す手法について提案した．また，同時にこの手法に沿って実際にプログラムを解析するツールを作成している．現在，このツールはSapid [3] 上で稼働するものであり，C++ で3000行ほどのプログラムになっている．

今後は実際にこのツールを使って幾つかのCプログラムに対してオブジェクトのカプセルを導出させて，手法の妥当性や手法で用いているパラメータ等を変化させるとさらにどのようなカプセルを導き出せるかという事について，研究を進めていく．

謝辞

Sapid の利用の便宜，および利用に際する支援を頂いた名古屋大学 阿草研究室の各位に感謝します．

参考文献

[1] Sying-Syang Liu and Norman Wilde, "Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery", Proc. Conf. on Software Maintenance, 1990, pp. 266-271

[2] William C. Chu and Sukesh Patel, "Software Restructuring by Enforcing Localization and Information Hiding", Proc. Conf. on Software Maintenance, 1992, pp. 165-172

[3] 山本 晋一郎, 阿草 清滋, "細粒度リポジトリに基づいたツール・プラットフォームとその応用", 情報処理学会研究報告 95-SE-102, 1995.1, pp.37-42

[4] Margaret A.Ellis and Bjarne Stroustrup, "The Annotated C++ Reference Manual", Addison Wesley, 1992. 足立 高徳, 小山 裕司 訳, "注解 C++ リファレンスマニュアル", トッパン, 1992

[5] Bjarne Stroustrup, "The C++ Programming Language, 2nd edition", Addison Wesley, 1993. 左藤 信男, 三好 博之, 追川 修一, 宇佐見 徹 訳 "プログラミング言語 C++ 第2版", トッパン, 1993