ソフトウェア工学 93－6
（1993. 7. 8）

動的振舞いを用いたソフトウェア性能分析支援システムの構築

岩永将幸　　大森健児

法政大学大学院工学研究科システム工学専攻

本論文では，ソフトウェアをチューンナップするために，プログラムの静的な構造や動的な振舞いを知らせるGUIを用いた性能分析環境の構築について説明する．静的な情報や動的な振舞いをどのように提示するかによって分析の作業効率は大きく左右される．そこで，静的な構造であるユーザ定義関数の親子関係を，単純なツリーによる表現に変換するグルーピング処理を提案する．また，動的な振舞いを視覚化するために，関数単位の実行時間と呼出し回数に応じて，関数のツリー上のノードサイズの大きさが変化し，また，これに対して棒グラフの表示などを行うインタフェースを構築した．このソフトウェア性能分析システムを利用して，あるプログラムのチューンナップをしたところ，その実行時間は11.5％向上した．

Building a Software Performance Analysis Supporting System Visualizing Dynamic Information

Masayuki Iwanaga　　Kenji Ohmori

Graduate School of Engineering, Hosei University

This paper describes how to build a performance analysis system which is used to tune up a program. The easiness to tune up a program depends on how the static and dynamic information of the program is visualized. The paper presents Grouping method which represents static structure of user defined functions by a simple tree. For visualizing dynamic information, we provide nodes which represent functions and change their size according to how many times they are called. By analyzing a software with this system, performance of a program was improved 11.5% in execution time.

1　はじめに

開発的な要素を含んだ大規模なソフトウェアを開発する場合には，最初からすべてのことが判明しているわけではない．このため，あるところはトップダウン的に，また，あるところはボトムアップ的にプログラムを開発することになり，開発前に想定していたプログラムの構造とは異なる場合が少なくない．しかし，改めて最初からきれいに書き直す，というわけにもいかないので，完成したプログラムの静的な構造や動的な振舞いを何らかの方法で知らせ，チューンナップの機会を与えてあげると助かる場合が多い．また，目に見えない情報を視覚的に提示し，その情報を簡単に理解することができるようにしてくれるとさらに助かる[1)-4)]．そこで，本論文では，プログラムの静的な構造や動的な振舞いの視覚化を用いた，ソフトウェア性能分析を支援する環境の構築について説明する．

2　視覚化

プログラムの構造を視覚化するために，ユーザ定義関数の静的情報を利用する．静的情報である関数の親子関係をもとに，関数をノードとし，親（呼ぶ側）と子（呼ばれる側）をアークで結ぶ構造表現を用いて情報の視覚化を行った．しかし，一般に，この表現法を用いると有向グラフになり，アークが複雑に絡み合ってしまうため，構造を理解するには複雑すぎることから適切な表現法とはいえない．そこで，この複雑さを除去するために，この有向グラフによる表現を単純なツリーに変換する手法として，グルーピングという手法を提案する．グルーピングとは，複数の関数から呼ばれる関数をルートとする別のグループを構成することによって，有向グラフをツリーに変換する手法である．ツリーによって親子関係を視覚化することによって，ツリーのルートからリーフの方向へ親子関係が必ず成り立つことになり，ソフトウェアの構造を理解することが容易に行えるようになる．また，プログラムをチューンナップするときに，性能を判断する目安として，動的情報であるユーザ定義関数の実行時間と呼出し回数を利用する．これらの情報は，基本的には，ツリー上の各ノードがそれぞれ保持する．

さらに，分析を効率よく行うために，情報を視覚化する機能をいくつか持たせている．1つめは，動的情報と関数構造とを同時に分析するためにノードサイズを情報に合わせて変更する機能で，2つめは，関数間の動的情報を比較するために棒グラフを表示する機能である．3つめは，動的情報に対して閾値を定めることによって，ツリー上のノードの表示／非表示を決定する機能で，4つめは，関数名のリストから関数名を選択することによって，対応するノードを選択するための機能である．5つめは，呼出しの階層ごとに分析を行うために，その階層の情報を提示する機能である．最後は，ソースコードの分析を行うための関数ごとのソースコードを提示する機能である．

以下，システムの詳しい構成について説明する．

3　システムの概要

システムの構成を図1に示す．システムは，分析対象ソフトウェアから情報を取り出す「前処理部」と，これらの情報を視覚化したインタフェースを利用して対話的に分析を行う「情報分析部」から成る．前処理部は，分析対象ソフトウェアの

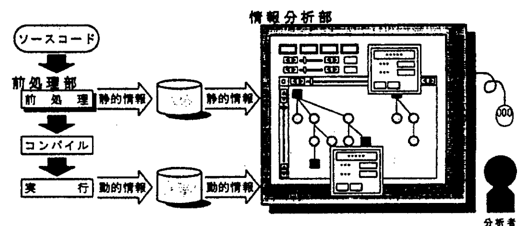

図1　システムの構成

ソースコードを入力にとる（ただし，C言語で書かれたもの）．そして，構文解析を行い，各ユーザ定義関数の静的情報を取り出す．さらに，動的情報を取り出すために，入力ソースコードにフック関数を埋め込む．前処理部の出力は，このフック関数を埋め込んだソースコードである．この出力ソースコードをコンパイルし，実行するとフック関数の作用により，動的情報が得られる．前処理によって得られた静的情報と動的情報は，それぞれファイルに格納される．これらの情報は，情報分析部で利用される．情報分析部は，GUIを用いた対話環境を提供する．分析を行う際の基本画面には，ユーザ定義関数の親子関係を示したツリーを表示したインタフェースをもつ．ツリー上の各ノードが保持する情報は，そのノードをマウスでクリックすることによって得られる．さらに，情報を様々な方法で視覚化したインタフェースを提供している．これらのインタフェースについては，5章で詳しく述べる．

4　前処理部

前処理部の主な働きは，分析に必要なソフトウェアの諸情報を取り出すことである．情報には，ソースコードレベルで不変な静的情報と，実行時の入力によって変化する動的情報がある．本システムでは，静的情報としてユーザ定義関数の親子関係，型そしてそれぞれのソースコードを取り出し，動的情報としては，ユーザ定義関数ごとの実行時間，呼び出し回数を測定し，利用する．

前処理部は，分析対象となるソースコードを入力に取り，まず，構文解析を行いユーザ定義関数を識別し，静的情報を取り出す．そしてこの情報は，ファイルに格納される．次に，動的情報を取り出すために，入力コードにフック関数を埋め込む．この変更されたソースコードが，前処理部の出力となる．

フック関数は，呼ばれた時間を測定し，その測定時間をあらかじめ確保しておいたメモリに蓄える手続きである．つまり，この関数を各ユーザ定義関数の入口と出口に埋め込むことによって，それぞれの実行時間が測定できるようになる．フック関数の動作の例を図2に示す．時間の測定には，UNIXシステムコールを用いており，1/100秒単位で測っている．

前処理部の出力ソースコードを，コンパイルし，実行すると，フック関数の作用により動的情報がメモリ上に格納される．この情報は，ソフトウェアの実行が終了した後，ファイルに転送される．

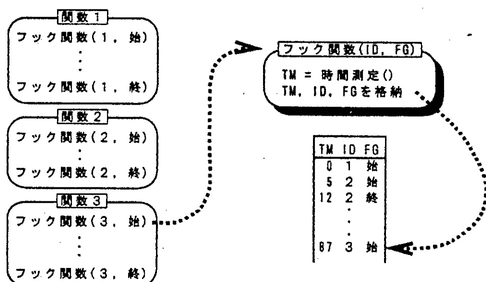

図2　フック関数の動作

5　情報分析部

5．1　グルーピング処理

情報分析部は，GUIを用いて構築した，マウス操作による対話環境を提供するものである．基本のインタフェースは，ユーザ定義関数の親子関

```
procedure Grouping_root(node)
  G:=Make_Group(node);
  Grouping(G, node);
end

procedure Grouping(G, node)
  children:=Get_Children(node);
  while children ≠∅ do
    child:=Get_Child(children);
    if child の親が複数 then
      Add_Grouping_Node(G, node, child);
      Grouping_root(child);
    else                    /* 親は node だけ */
      Add_Normal_Node(G, node, child);
      Grouping(G, child);
    endif
  endwhile
end
```

図3　グルーピングアルゴリズム

係を視覚化したものを用いた．ユーザ定義関数をノード，その親子関係をアークで表現すると，一般に，有向グラフとなる．しかし，この表現は複雑で，分析者にとってその構造を理解することは大変困難な作業となる．そこで，この複雑な有向グラフを単純なツリーに変換するために，グルーピング処理を用いる．

グルーピング処理とは，２つ以上のユーザ定義関数から呼ばれるユーザ定義関数が存在した場合，その関数以下のユーザ定義関数をそのツリーでは無視し，別にその関数をルートとする別のツリーを作成する処理である．アルゴリズムを図３に示す．手続き Grouping_root は，グループのルートとなる関数（node）を引数に取る．手続き Make_Group は，引数 node をルートとするグループリストを返す．手続き Grouping は，グループリスト（G）と関数（node）を引数に取り，その関数の全ての子をグループリストに加えていく再帰関数である．手続き Get_Children は，引数で与えられた関数から呼ばれる全ての子のリストを返す．手続き Get_Child は，子のリストから１つの子を取り去り，返り値としてその子を戻す．手続き Add_Grouping_node と Add_Normal_node は，グループリスト，親関数，子関数を引き数に取り，グルーピングされたノードとして，また，普通のノードとして，それぞれグループリストに加える．

グルーピング処理の例を図４に示す．この例では，５つのユーザ定義関数が存在する．関数は，それぞれ０～４の識別子を持つ．図４の左側は，

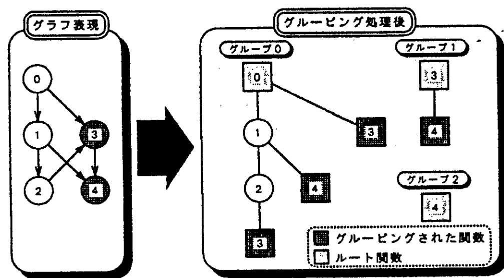

図４　グルーピング処理の例

ユーザ定義関数の親子関係を有向グラフで表現したものである．この図から，関数３と４が２つの関数から呼ばれていることが分かる．これにグルーピング処理を施すと，結果は，図４の右側のようになる．関数０をルートとするツリーと，新たに関数３と４をルートとしたツリーが作成され，結局，３つのグループに分けられることになる．グルーピングされた関数は，普通の関数（円で表現）と区別するため四角で表現される．各グループで，ルートノード以外の四角のノードは，そのノードをルートとしたグループが他に存在することを意味する．

グルーピング処理を行う利点としては，次のようなものがあげられる．

①各ノードが固有の情報を保持する

図４の関数３を考えると，有向グラフでは，関数０から呼ばれた場合，関数２から呼ばれた場合，そしてその合計，と３種類の情報を１つのノードで保持しなければならない．しかし，グルーピング処理を施した後では，この３つの情報は，それぞれに対応する別のノードが保持することになる．ただし，グルーピングされた関数のルート以外に（リーフとして）存在するノードが保持する情報には，そのノードの子の情報も含まれる．

②親子関係が明確になる

各グループ内のノードは，ツリー構造になっているため，必ず，親が唯一である．つまり，あるノードの親を分析したいときは，単純にその上のノードを分析すれば良いことになる．これにより，ツリーのルートからリーフの方向へ，親子関係が必ず成り立ち，ツリーの深さがそのまま，呼び出しの深さになることも保証している．

③各グループが一般的機能を持つ可能性が高い

グルーピングされた関数は，複数から呼び出される関数である．普通，複数から呼び出される関数は，一般的な機能を果たす．つまり，グルーピング処理を施すことにより構成された各ツリー

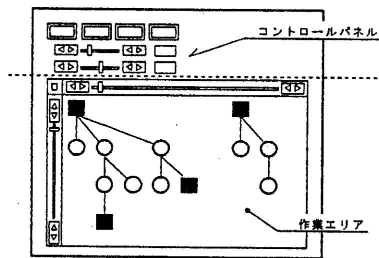

図５　作業画面

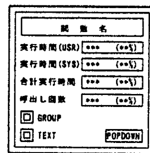

図６　情報ボックス

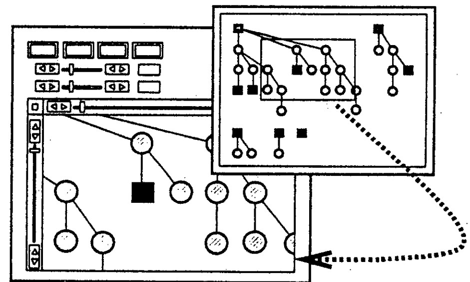

図７　作業エリアと副画面の対応

は，それぞれモジュール化されたものである可能性が高く，記述の抽象度が高まり，その結果，分析を容易に行うことができるようになる．

④チューンナップの影響が簡単に判断できる

プログラムをチューンナップするために，ソースコードを書き換える場合，その影響の範囲を十分考慮しなければならない．つまり，書き換えには，その関数の内部の場合と，その関数の親と子に影響を及ぼすような場合がある．影響範囲が関数内部だけの場合には，グローバル変数の変化にだけ注意すれば良い．しかし，親と子に影響を及ぼす場合には，その影響も考慮しなければならない．例えば，関数の引き数を増やしたり，返り値を変えるような場合には，その関数を呼び出す親全てが影響を受ける．グルーピングを行うことによって，円で表現されたノードは，必ず１つの親しか持たず，四角で表現されたノードは，複数の親を持つようになっているため，このような影響範囲が簡単に判断できる．

５．２　情報分析部の機能

情報分析部では，グルーピング処理によって単純化された，ユーザ定義関数の親子関係をツリーで表現したインタフェースを介して，性能分析を進めていく．全ての分析操作は，マウスを利用して対話的に行っていく．

実際の作業画面は，図５のようになっておりコントロールパネルと作業エリアに分けられる．コントロールパネルはメニュー形式で，分析を支援する機能を呼び出す．作業エリアには，分析対象となるソフトウェアのユーザ定義関数のツリー構造が表示されており，各ノードが保持する情報は，そのノードをクリックすることにより情報ボックス（図６）がポップアップし得られる．

計算機上の画面の大きさには制限があるため，ユーザ定義関数の構造全てを作業エリア内に納めることはできない．そこで本システムでは，スクロールバー方式を採用した．しかし，この方式では，全体の構造を把握することは困難であるため，縮小表示した副画面も用意した．また，表示位置も，スクロールバーによる移動の他に，副画面の表示したい箇所をクリックすることにより，すばやく移動することが可能になっている（図７）．

そしてさらに，分析を効率よく進めるために次のような機能がある．

①ノードサイズの変更

実行時間や呼出し回数の情報と関数の構造とを同時に視覚的に，すばやく得るために，各ノードの大きさを情報に合わせて変化させる機能．ノードの大きさを実行時間に合わせるか呼び出し回数に合わせるかは，コントロールパネルで選択する．各ノードの大きさは，選択した情報の全体に対する割合により変化する．

図４に示した構造を持つソフトウェアで，各関数が表１に示すような実行時間に関する情報を持つものとする．ノードサイズは，最も実行時間の

表1　図4のソフトウェアに対する情報
（全体の実行時間：18）
（最大実行時間：7）

GROUP	親id	関数id	実行時間	実行時間 ÷全体×100	実行時間 ÷最大実行時間×100
0	-	0	2	11.1 %	28.6 %
	0	1	1	5.6	14.3
	1	2	3	16.7	42.9
	0	3	7	38.9	100.0
	2	3	4	22.2	57.1
	1	4	1	5.6	14.3
1	-	3	6	33.3	85.7
	3	4	5	27.8	71.4
2	-	4	6	33.3	85.7

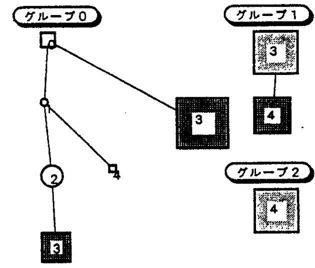

図8　ノードサイズの変更

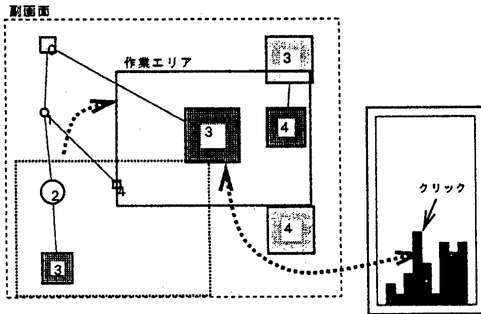

図9　棒グラフによる操作

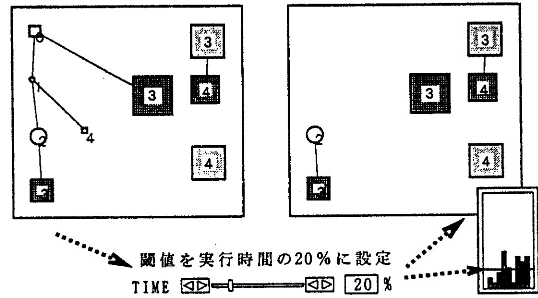

図10　閾値の設定

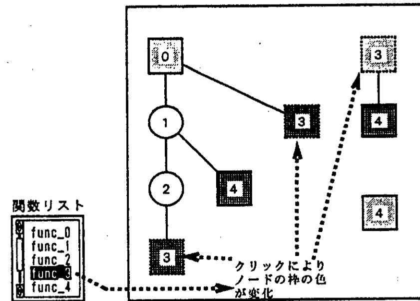

図11　関数名によるノード選択

かかっているノードを規準として相対的に変化させる．つまり，表1において，グループ0の関数0から呼ばれた関数3が実行時間7で最大である．よって相対値は，各関数の実行時間÷7×100（%）で計算される（計算値は表1に概記）．この相対値に従ってノードサイズは，図8に示すように変更される．呼出し回数に関しても同様にして変化させることが出来る．

②棒グラフによる情報の提示

関数間の情報を比較するために，棒グラフにより情報を視覚化する．このグラフは，全体の情報に対する相対値で表示する．また，この棒グラフをクリックすることにより，対応するノードが中心になるように作業エリアが移動する．

表1の情報を用いると，全体の実行時間は18であるから，棒グラフのそれぞれの値は，各関数の実行時間÷18×100（%）によって計算される（計算値は表1に概記）．棒グラフによる操作例を図9に示す．

③ノードの表示／非表示の選択

実行時間と呼び出し回数のそれぞれに対して，ある閾値（これも全体に対する割合による）を定め，ノードの表示／非表示を決定する．分析対象にならないような実行時間の小さい関数を表示しないようにすることで，よりすばやく分析を行えるようになる．閾値は，コントロールパネルのボリュームによって設定する．また，この閾値は，②で述べた棒グラフ上に，グラフに直交する直線

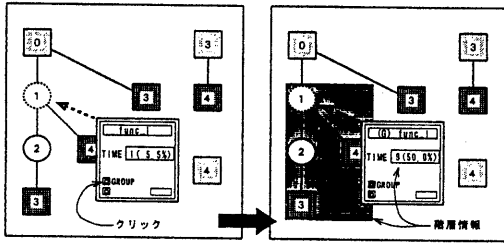

図 12　呼出し階層の情報

の切断線として視覚的に表示される（図 10）.

④関数名によるノードの選択

ノードには，関数名が表示していないため，関数名によりノードを選択することは作業エリアだけでは不可能である．そこで，関数名によるノード選択を実現するために，関数名リストを用意した．関数名リスト上の関数名をクリックすると対応するノードの枠の色が変化する（図 11）.

⑤呼出し階層による情報の提示

通常のモードでのノードの保持する情報は，そのノードに対応する関数の純粋な情報で，その関数が呼び出す関数の情報は含まない．しかし，その関数以下の情報も含めて，ある一定の呼出し階層で分析を行うことも必要である．そこで，階層における合計情報を提示する機能を実現した．情報ウィンドウ内の GROUP ボタンをクリックすることで，それ以下の関数の合計情報がその情報ウィンドウに表示される（図 12）.

⑥関数ごとのソースコードの提示

動的情報による分析を行い，チューンナップを行う必要のある関数を見つけたら，つぎは，その関数のソースコードを分析しなければならない．そこで，各ノードに，それぞれ対応する関数部分だけのソースコードを持たせることで，部分的な分析を行えるようになっている．情報ウィンドウ内の TEXT ボタンをクリックすることによって，テキストウィンドウが開き，ソースコードを参照することができる．

6　システムの検証

6．1　ソフトウェア性能分析

本システムを用いて実際にソフトウェアの性能分析を行った結果を示す．分析対象としたソフトウェア[5)]は，オフライン手書き漢字認識ソフトウェアで，2426 ライン，81 個のユーザ定義関数を持つ大きさのものである．このソフトウェアは，入力する文字によって動的に実行時間が変化するため，数種類の入力に対して分析を行った（ソフトウェアの開発者と分析者は異なる）.

分析を行った結果，どの入力に対しても 2 つの関数の実行時間が，全体の約 30 %を占めていることが分かった．このことから，この 2 つの関数を書き換えることによって，効果的なチューンナップが可能であると判断できる．よって，この 2 つの関数をソースコードレベルで分析を行い，チューンナップするためにプログラムを変更した．

チューンナップによるプログラム変更の効果を確認するために，881 字種の入力に対して，変更前と変更後の実行時間を比較した．変更前の実行時間は，平均 28.93 秒であったのに対して，変更後の実行時間は，平均 25.60 秒であった．つまり，平均 3.33 秒（11.5 %）の改善が達成された．また，881 字種の入力に対して，改善された入力数は 874 字種で，かつ，悪くなったものはなかった．

6．2　考察

プログラムの静的な構造や動的な振舞いを視覚化することで，ソフトウェア性能分析を容易に行うことができ，改善率も 11.5 %と満足する結果が得られた．また，分析者は，このプログラムの構造を全く知らなかったが，短時間で分析を行うことができた．

しかし，プログラムの関数レベルの構造を把握すること，実行時間の大きい関数を判断すること，は容易に行えたが，関数のソースコードを分析する際には，システム側からこのための手段が何も支援されていないため困難な作業であった．この

分析に対しても，情報を視覚化する支援環境が必要である．

また，フック関数により得られる動的情報は，全てメモリに蓄えられるため，メモリ消費量が多きいことも問題の１つである．つまり，ユーザ定義関数の呼ばれる回数が多くなると，メモリが足りなくなり分析できなくなってしまう．現時点では，メモリを50,000,000バイト確保しているが，１回のフック関数呼び出しにより蓄えられる情報は10バイトであるため，5,000,000回までしかフック関数を呼ぶことはできない．これでは，分析対象とするソフトウェアをかなり限定することになってしまう．実際，前節で分析対象としたソフトウェアでも入力によってはメモリが不足することがあった．

さらにもう１つの問題として，ユーザ定義関数の親子関係を静的情報だけによって構成していることである．このため，ポインタ呼び出しによる関数を含んだソフトウェアは，動的な親子関係が構成されるため分析できない．

7 おわりに

完成したプログラムの静的な構造や動的な振舞いを知らせチューンナップの機会を与えてあげるために，これらの情報を視覚化したソフトウェア性能分析を行う環境を提供した．グルーピング処理により単純なツリーの集合となったユーザ定義関数の親子関係を基にしたインタフェースを用い，さらに，分析を支援する様々な機能を提供することによって，効率よく分析を行う環境を構築した．実際にオフライン手書き漢字認識ソフトウェアの性能分析を行い，チューンナップした結果，11.5％の性能改善が達成できた．

今後，さらに効率よく分析を行うために，また，分析対象を広げるために，ソースコード分析の支援，メモリ消費量の効率化，関数の動的呼び出しを含むソフトウェアへの対応など，多くの問題が残っている．

【謝辞】GUI構築に用いたUIMS「鼎」と「ゆず」を提供して下さった日本電気㈱ソフトウェア生産技術開発本部の関係諸氏に感謝いたします．

参考文献

1) Allen, D. M., David, H. H. and David, J. J.: Traceview:A Trace Visualization Tool, IEEE software, Vol. 7, No. 3, pp. 19−28 (1991).
2) Kathleen, M. N.: Performance Tools, IEEE software, Vol. 7, No. 3, pp. 21−30 (1990).
3) Paul O.: Maintenance Tools, IEEE software, Vol. 7, No. 3, pp. 59−65 (1990).
4) 松浦，大林：ソフトウェア開発環境 dmCASE，情報処理学会論文誌，Vol. 31, No. 7pp. 1091−1103 (1990).
5) 春木，大森：仮説推論によるオフライン手書き漢字認識，電子情報通信学会論文誌，Vol. J76-D-II, No. 1, pp. 65−73 (1993).