修士学位論文

題目

動的情報を利用したソフトウェア部品評価手法

指導教官
井上克郎 教授

報告者
藤井 将人

平成 15 年 2 月 12 日

大阪大学 大学院基礎工学研究科
情報数理系専攻 ソフトウェア科学分野

平成 14 年度 修士学位論文

動的情報を利用したソフトウェア部品評価手法

藤井 将人

内容梗概

ソフトウェア開発において，既存のソフトウェア部品を再利用することにより，生産性と品質を改善し，結果として開発コストを削減することができる．再利用を用いて開発を行うには，開発者に部品選択のための基準を示す必要がある．開発者に部品選択のための基準を示すことで，再利用を用いた開発を円滑に行うことができると考えられる．我々は，開発者は重要であると判断した部品を再利用するという開発モデルを仮定し，利用実績に基づいたソフトウェア部品評価手法 (Component Rank 法, CR 法) を提案している．CR 法を用いることで，多くの部品で利用されている汎用性の高い部品の検索が容易なものとなり，ソフトウェア開発における再利用の支援となる．しかし，ソフトウェア中に存在する機能の再利用を目指した時，再利用性の高い部品の検索だけでなく，その部品が「いつ」「どのように」利用されているかという情報も必要となる．CR 法による部品評価では静的な利用関係を用いているため，これら動的な情報を正確に取得することはできない．　そこで，本研究では，動的な情報を利用したソフトウェア部品手法の提案を行う．動的な情報を用いることにより，実行した方法毎に部品の評価値を得られるだけでなく,「いつ」「どのように」利用されているかという動的な情報の取得できる．実行方法によって異なる利用関係ごとに，部品評価値を求めることで，特定の機能を実装するために重要な部品の情報を提供することができる．また，部品評価値でフィルタリングしたシーケンス図を作成することで，重要な部品間の呼び出し関係の情報を提供することができる．　この提案手法に基づいて，Java アプリケーションを対象とした部品評価システムの実装を行った．その結果，本提案手法による部品評価値は，実行した機能を実装している部品が高い評価値を得られることを確認した．また，フィルタリングを行ったシーケンス図を作図することで，開発者が注目すべき部品を絞り込むことができ，再利用を支援できることを確認した．

主な用語

再利用
Component Rank 法
動的利用関係
シーケンス図

目 次

1　まえがき

近年，ソフトウェアは大規模化・複雑化しており，開発者は高品質なソフトウェアを一定期間内に効率良く開発することを要求されるようになっている．そこで，開発効率を向上させるためのソフトウェア工学技術が様々提案されている．その中の一手法として，**ソフトウェア再利用**が注目されている．

再利用とは，既存のソフトウェア部品を同一システム内や他のシステムで用いることであると定義されている [1]．一般に．ソフトウェアの再利用は生産性と品質を改善し，結果としてコスト削減するといわれており，再利用の実際の効果を報告した論文も多く発表されている [2, 3, 4]．

この再利用を用いた開発では，開発者は既存のソフトウェア部品をコピーして再利用したり，ライブラリとして再利用を行うが，この時，開発者にどの部品を再利用するべきであるのかという，部品の取捨選択の基準 (重要度) を定量的に示す必要がある．

そこで，我々の研究グループでは，利用実績に基づいたソフトウェア部品重要度評価手法 **Component Rank 法**(CR 法) の提案を行っている．提案された CR 法では，以下のような再利用を用いた開発モデルに基づいて部品評価値を求めている [5, 6]．

1. ソフトウェアを構成する部品間には相互に利用関係がある．
2. 開発者は重要だと判断した部品を，コピーして再利用したり，ライブラリとして再利用する．
3. 一般に，時間が経過し，多くのプロジェクト開発で再利用などが行われるに連れて部品の利用関係は変化していく．
4. 十分な時間が経過した状態で，利用関係はある形に収束する．
5. 十分な時間が経過した状態のもとで，被利用数が多い部品は重要である（再利用性が高い）．また，重要な部品から利用されている部品も重要である（再利用性が高い）．

この手法を用いることで，よく利用される汎用性の高い部品を抽出することが可能となり，再利用を用いた開発を支援できということを確認している．しかし，複数の部品で実装されている”機能”について再利用を行う際，機能を実現する重要な部品の有無だけでなく，それらが相互に「いつ」「どのように」利用されているかという情報が必要となる．これらの情報が欠落していれば，開発者はそれら部品が使われるであろう場面を十分に想定することが出来ず，かえって再利用することが非効率になる恐れがあると考えられる [7] .

そこで本研究では，動的な利用関係モデルに基づいて，動的な情報を利用したソフトウェア部品手法の提案を行う．動的な情報を用いて部品の評価を行うことにより，個々の部品の評価値を得られるだけでなく,「いつ」「どのように」利用されているかという動的な情報の取得が可能となる．利用関係は実行方法により変化するため，開発者が再利用したい機能を実行することで，その実行方法に依

存した重要な部品の抽出が可能となり，ソフトウェア中の特定機能の再利用を支援する有効な手法であると考えられる．

また，提案手法で得られた部品評価値の利用方法として，シーケンス図への適用を行う．評価値でフィルタリングを行い，重要な部品間の利用関係のみを出力する．重要度の低い余分な部品が除かれたシーケンス図を参照することで，対象ソフトウェアのモデル，動作が把握しやすくなり，結果ソフトウェア理解の支援につながると考えている．

以降， 2 節では，ソフトウェア部品と利用関係，再利用を用いたソフトウェア開発モデルに各概念について述べる． 3 節では，文献 [6] で提案している，CR 法の定義と部品評価値計算方法について述べる． 4 節では，本提案手法である動的な情報を利用した部品評価手法について述べる． 5 節では，提案手法を基に，Java を対象とした部品評価システムについて述べる． 6 節では，実装したシステムに Java プログラムを適用し，結果より有効性について検証する． 7 節では，部品評価値の応用としてシーケンス図への適用について述べる．最後に 8 節で，まとめについて述べる．

2　準備

準備として，ソフトウェア部品と部品間の利用関係，再利用を用いたソフトウェア開発モデルの各概念について説明する．また，他分野における実績に基づく評価手法を紹介する．

2.1　ソフトウェア部品と部品間の利用関係

一般に**ソフトウェア部品**(Software Comopnent) とは，再利用できるように設計された部品とされている [8, 9]．本論文ではより一般的に，ソースコードファイルやバイナリファイル，ドキュメントなどの種類を問わず，開発者が再利用を行う単位をソフトウェア部品，あるいは単に部品と呼ぶ．

ソフトウェア部品間には互いに利用する，利用されるという関係が存在する．この関係のことを**利用関係**と呼ぶ．例えば，プログラムのソースコードにおけるメソッド呼び出しや継承関係，また，ドキュメントにおけるリンクや参考文献，といったものがこの利用関係にあたる．

ソフトウェア部品を頂点，利用関係を有向辺とみなすことで，利用関係をグラフ上に表現することができる．この利用関係のグラフを，**部品グラフ**(*Component Graph*) と呼ぶ．

図 1 は二つのソフトウェアシステム X と Y を部品グラフ化したものである．X は A から E の 5 つ，Y は F から I の 4 つの部品で構成されており，部品 C は 部品 A および B を利用し，部品 D および E は 部品 C を利用している．同様に部品 H と I は 部品 G を利用し, 部品 G と F はお互いに利用しあっている．

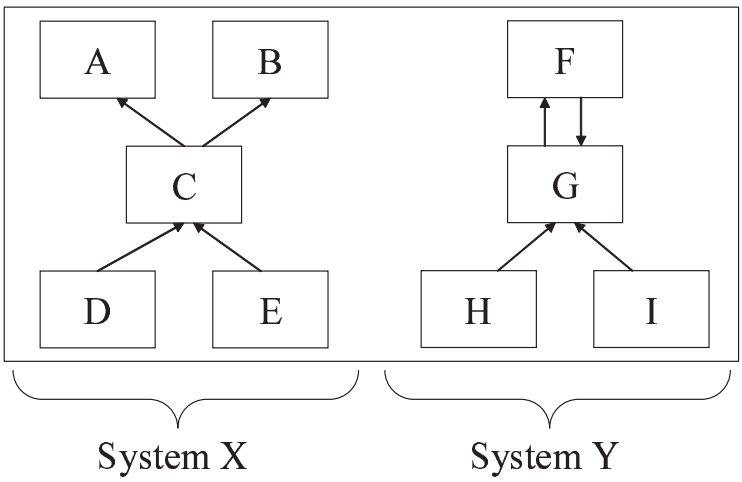

図 1: 部品グラフの例

2.2　再利用を用いたソフトウェア開発モデル

ソフトウェア開発者が，過去に開発されたソフトウェア部品を再利用し，新しいソフトウェアの開発を行う場合を想定する．一般に，開発者は過去に開発されたソフトウェア部品の中で，開発しよう

としているソフトウェアに対して再利用する価値のある部品と判断したものを再利用すると思われる．そこで，既存のソフトウェア部品を再利用するという行為を，その再利用部品への支持投票とみなすことができる．

時間の経過と共に，再利用を用いたソフトウェア開発が何度も繰り返されれば，再利用性の高い部品は何度も利用され，支持投票数が増加する．逆に再利用性の低い部品はあまり利用されず，支持投票数は低い値に留まる．そして，十分な時間が経過した状態で，部品間の利用関係はある形に収束すると仮定する．ソフトウェア部品は獲得した票数に応じた再利用性の評価値を持つと考えられるので，以下の式が成り立つ．

$$(\text{部品の評価値}) = (\text{部品への投票数})$$

このとき，単純に獲得票数を数えるだけでなく，どのような部品から利用されたかによって票に重み付けをする．多くの部品から利用されるような重要なな部品（重要な部品の開発者）に利用されている部品は，再利用性が高いとみなして，同じ一票の支持投票でも，再利用性の評価値の高い部品から投票された場合と，評価値の低い部品から投票された場合では，前者の票の方がより高い重みを持つようにする．

また，個々の部品が利用している部品の数も考慮する．ある部品 A が多数の部品を利用している場合には，各利用部品が A の機能に占める割合が少なくなるので，部品 A の再利用性の評価は分散してしまうとみなす．つまり，ある部品が複数の部品に投票している時は，票の重みは各利用部品にある配分率をもって配分され，以下の式が成り立つ．

$$(\text{票の重み}) = (\text{投票元の評価値}) \times (\text{投票先に対する配分率})$$

このように，部品の利用関係に注目し，どのような部品からどのくらい利用されているかという利用実績に基づいて部品の重要度を計算する手法を，**Component Rank 法**と呼ぶ． 3 節で，この Component Rank 法の説明を行う．

2.3 他分野における実績に基づく評価手法

実績に基づく評価手法は，計量社会学においては以前から存在し，様々な分野に応用されている．ここでは，他の分野における実績に基づく評価手法を取り上げる．

Influence Weight 論文の引用解析の分野においては，1970 年代には実績に基づいて論文の重要度を評価する手法が提案されている．Narin らは，論文の重要度を論文がどの程度引用されているかという実績に基づいて評価する手法「Influence Weight」を提案している [10, 11, 12]．この手法では，(1) 多くの論文から引用されている論文は重要である，(2) 重要な論文から引用されている論文はまた重要である，という 2 つの考えに基づいて論文の重要度を評価している．

Page Rank 実績に基づく評価手法は，Web ページ検索の分野にも応用されている．Page らは，Web ページの重要度をその Web ページがどの程度リンクされているかという実績に基づいて評価する手法「Page Rank」を提案している [13, 14]．この手法では (1) 多くの Web ページからリンクされている Web ページは重要である，(2) 重要な Web ページからリンクされている Web ページはまた重要である，という 2 つ考えに基づいて Web ページの重要度を評価している．有名な検索サイト「Google」[15] はこの Page Rank による評価を用いた検索サービスを提供している．Google では，検索語の一致によって得られた検索結果を Page Rank の重要度で順位づけして表示し，より正確な検索を可能にしている．

3 Component Rank 法

従来のソフトウェア部品再利用性評価手法は，部品の構造や部品の構造やインターフェイスなど部品そのものの持つ静的な特性を計算して再利用性を評価するものとなっている [16, 17, 18, 19]．また，それらの有効性の検証にはプログラマによる再利用性評価やプログラマの実装時間などが用いられている．

しかしながら，実際のソフトウェア開発においては，静的な特性からは再利用性が低いと評価されていても，何度も再利用されている部品も存在すると考えられる．たとえば，プログラムの内部構造が複雑で，静的な特性としては再利用性が低くても，開発組織内で長年にわたって利用されている部品であれば頻繁に再利用されるであろう．

そこで我々は，再利用を用いたソフトウェア開発モデルに従い，その部品がどの程度利用されているかという実績に基づいて部品の重要度 (再利用性) を評価する手法，**Component Rank 法**(単に，CR 法) の提案を行っている [5, 6].

CR 法では， 2.1 節で説明した部品グラフを利用し，繰り返し計算により各頂点の重みを求めることで，それに対応する部品の評価値を定める．以降，部品評価値計算に必要となる，重みの定義，重みの計算，評価値計算に関する補正について説明し，その後，部品評価値計算例を示す.

3.1 重みの定義

ソフトウェア部品 C_i に対応する，部品グラフの $G = (V, E)$ 上の頂点を $v_i \in V$ とし，部品グラフ上の個々の辺および頂点に対して非負な重みを持つものとする．

まず最初に，頂点の重みの総和を定義する．部品グラフ上の各頂点 $v \in V$ は $0 \leq w(v) \leq 1$ の重みを持ち，G の頂点の重みの総和は 1 とする．

定義 1 (頂点の重みの総和)

$$\sum_{v \in G} w(v) \qquad = 1 \tag{1}$$

次に，頂点 v_i から v_j への辺 e_{ij} に関するの重み $w'(e_{ij})$ を次のように定義する．

定義 2 (辺の重み)

$$w'(e_{ij}) = d_{ij} \times w(v_i) \tag{2}$$

図 2 (a) はこの定義を図示したものである．d_{ij} は配分率とよび，頂点 v_i の重みをどのような比重で各有向辺の重みに配分するかを決定する．配分率は各有向辺ごとに与えられ，$0 \leq d_{ij} \leq 1$ かつ $\sum_i d_{ij} = 1$ を満たす値とする．また，頂点 v_i から v_j へ利用関係が存在しない場合，$d_{ij} = 0$ とする．

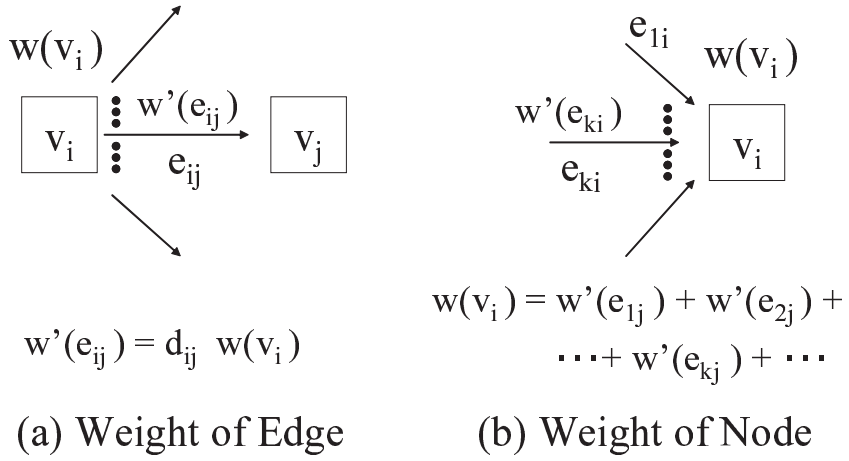

図 2: 重みの定義

最後に，辺の重みを基に頂点の重みを再計算する．この時，IN(v_i) を v_i を終点とする (v_i に入ってくる) 有向辺の集合とした時，頂点 v_i の重みは，v_i が終点となる有向辺 e_{ki} の重みの総和と定義する．

定義 3 (頂点の重み)

$$w(v_i) = \sum_{e_{ki} \in \mathrm{IN}(v_i)} w'(e_{ki}) \tag{3}$$

図 2 (b) はこの定義を図示したものである．

3.2 重みの計算

3.1 節で定義した，式 (3) に式 (2) を代入することで，次式 (4) が得られる．

$$w(v_i) = \sum_{e_{ki} \in \mathrm{IN}(v_i)} d_{ki} \times w(v_k) \tag{4}$$

各頂点に関してこの方程式を立てる事で，$n(= |V|)$ 個の連立方程式が生成できる．また，ベクトル W ，行列 D として，次のように定義することで，

定義 4 (ベクトル W ，行列 D)

$$W = \begin{pmatrix} w(v_1) \\ w(v_2) \\ \vdots \\ w(v_n) \end{pmatrix} \quad D = \begin{pmatrix} d_{11} & d_{12} & \ldots & d_{1n} \\ d_{21} & d_{22} & \ldots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \ldots & d_{nn} \end{pmatrix}$$

$n(=|V|)$ 個の連立方程式を次のように表す事ができる．

$$W = D^t W \tag{5}$$

ここで，行列 D^t は D の転置行列を表す．

定義 1 から，行列 D^t は推移確率行列の性質を満たしており，D^t を用いて開発者が部品をどのように参照するかをマルコフ連鎖モデルとして表現している．すなわち，開発者はある部品を参照した後に，辺に沿って利用関係のある部品の一つを参照していく．

ここで，式 (5) を満たす解 W は D^t に関する定常分布となり，対象とする部品の集合の中での参照を十分長い間時間繰り返した場合の，各部品が参照されている確率を示すことになる．つまり，重みが高い部品ほど開発者によって頻繁に参照されることになる．

定常分布が一意にもとめられる事を示すためには，D^t が既約であることが必要となるが，CR 法では，既約である事を保証するために後述する補正を加えている．

この場合，D^t における絶対値最大の固有ベクトルを求める事で，定常分布を求めることができるが，行列 D^t の絶対値最大の固有値は 1 であるため，累乗法 (power method) を用いて適当な初期ベクトルに対して繰り返し D^t を掛ける事で，近似解を容易に求める事ができる．

3.3 評価値計算に関する補正

前節で評価値計算に関する説明を行ったが，実際に運用する場合, 不具合が起こる場合がある．例えば，部品 C_i がどの部品も利用していない場合，頂点 v_i から全ての頂点への配分率 d_{ix} が全て 0 になり，配分率に関する定義（頂点からの辺への配分率の総和が 1）を満たさなくなる．また，グラフが強連結でない場合，つまり，任意の 2 頂点 v_i，v_j について v_i から v_j へ到達可能，かつ，v_j から v_i へ到達可能，が成り立たない場合，到達不可能頂点の重みが 0 になってしまい正しく評価が行えない問題がある．例えば 図 3 では，頂点 v_1 の重みが 0 になってしまい，v_1 から v_2 への利用関係を正しく評価する事ができない．このような場合，与えられた行列が既約であることを保証できず，定常分布が一意に定まらない場合がある．

そこで，全ての頂点を 図 4 のように低い配分率の擬似辺で結ぶ事で，与えられたグラフを強連結なグラフに変換し，行列が既約である事を保証する．この擬似辺は，各部品における自分を含めた全ての部品への明示的でない参照を意図している．

頂点 v_i を始点とする辺への補正を加えた配分率 $d'(e_{ij})$ を次のように定義する．

定義 5 (補正を加えた配分率)

$$d'_{ij} = \begin{cases} p \times d_{ij} + (1-p)/n & \text{if } v_i \text{ を始点とする有向辺が存在} \\ 1/n & \text{if } v_i \text{ を始点とする有向辺が存在} \end{cases} \tag{6}$$

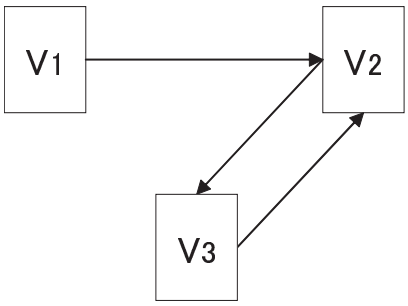

図 3: 強連結でない部品グラフ

ここで，p は実際の辺と擬似辺の重みの配分比率を指し，我々は p=0.85 という値を採用している．

この補正は，前節で説明したマルコフ連鎖モデル上での参照行為を,「ある部品を参照した後に，辺に沿って利用関係のある部品の一つを参照するが，ある確率（$1-p$）で，ランダムに全部品の中の一つを参照する」のように修正する．この修正により，参照行為をより自然な形でモデル化することができる．

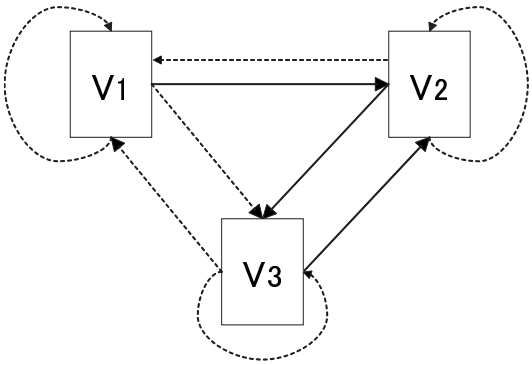

図 4: 評価値計算に関する補正（擬似辺の追加）

3.4 部品評価値計算例

図 5 は，部品評価値計算の例である．この図では，部品 C_1 が部品 C_2，C_3 を，部品 C_2 が部品 C_3 を，部品 C_3 が部品 C_1 をそれぞれ利用している． 図 5 における，部品評価値手順は次のようになる．

部品評価値手順例

1. 各頂点の重みの初期値を与える．

   - $w_0(v_i) = 1/3 = 0.333$　　(総和 1/部品数 3)

2. 頂点の重みより有向辺の重みを再計算

   - 有向辺への配分率は全て等しいものとする

3. 有向辺の重みより頂点の重みを再計算

4. 頂点の重みが収束するまで 2,3 を繰り返す

5. 収束した頂点の重みを，その頂点に対応する部品の評価値として出力

以上の計算により最終的に得られる部品評価値は，部品 C_1，C_2，C_3 それぞれ 0.400, 0.200, 0.400 となる．

3.5 部品評価値計算結果

オブジェクト指向言語 Java のクラスを部品として，CR 法による部品評価値の結果を 表 1 に示す．評価対象の部品として，Java 2 Software Development Kit, Standard Edition 1.3.0 (以下 JDK) および，いくつかの Java アプリケーションを用いた．全クラス数は 6192 クラスである．

評価値の結果により，JDK ライブラリが上位に位置していることがわかる．これらライブラリは再利用を目的として作成された部品であり，これらが再利用性が高い部品として出力されることは正しい結果である．またその中でも，String クラスや Object クラスなど，実際のプログラムでも頻繁に利用するクラスや，言語仕様上利用しなければならないクラスが上位に位置している．この結果より，多くの部品から利用されている部品は重要な部品であるというモデルにおいて，CR 法は有効な手法であると考えられる．

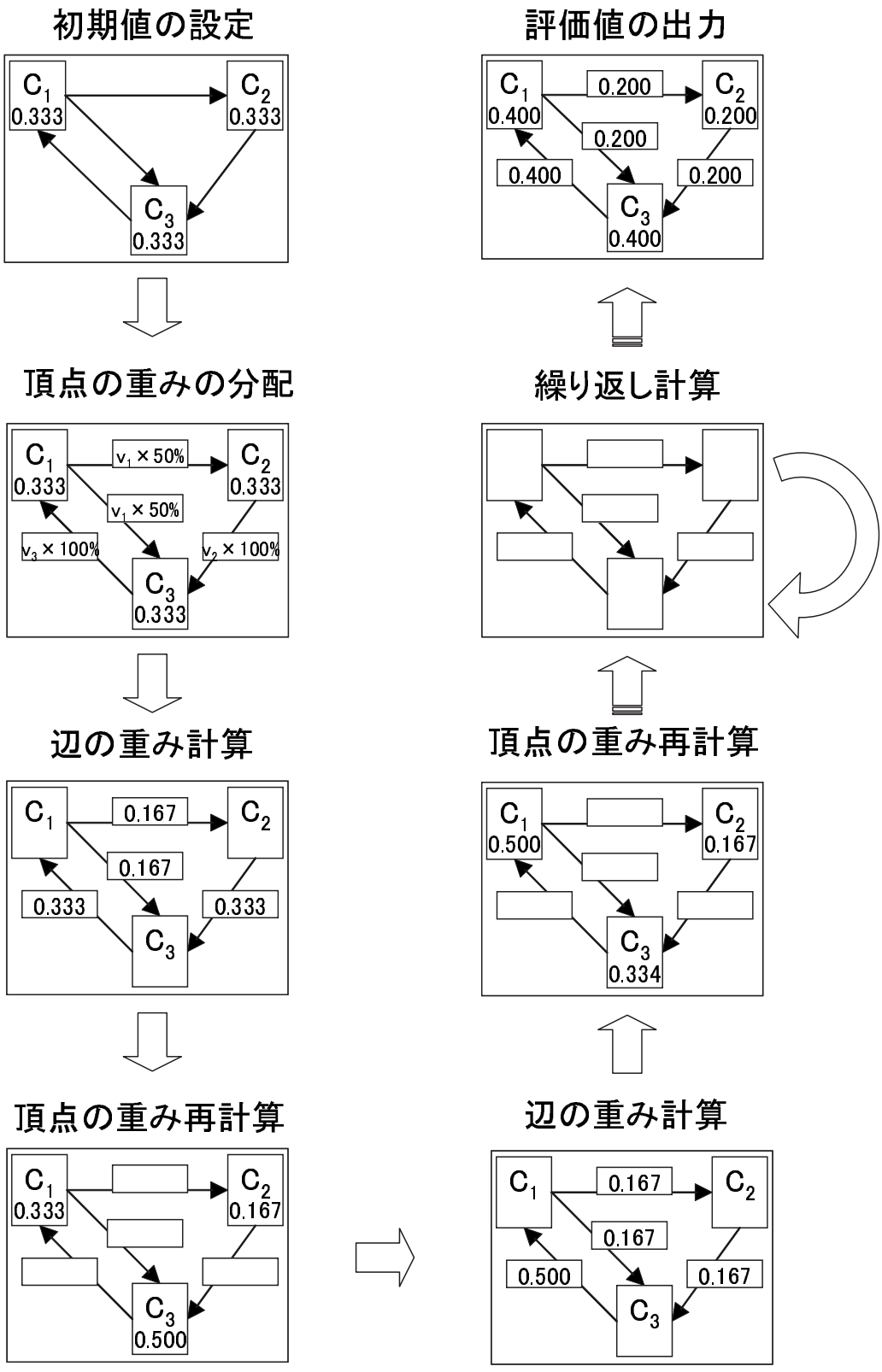
初期値の設定
C1 0.333
C2 0.333
C3 0.333
頂点の重みの分配
v1 × 50%
v1 × 50%
v3 × 100%
v2 × 100%
辺の重み計算
0.167
0.167
0.333
0.333
頂点の重み再計算
C1 0.333
C2 0.167
C3 0.500
辺の重み計算
0.167
0.167
0.500
0.167
頂点の重み再計算
C1 0.500
C2 0.167
C3 0.334
繰り返し計算
評価値の出力
C1 0.400
C2 0.200
C3 0.400
0.200
0.200
0.400
0.200

図 5: 重みの計算例

表 1: 部品評価値計算結果 (CR 法)

順位	クラス名	部品評価値
1	java.lang.String	8103415
2	java.lang.Object	7962612
3	java.io.EOFException	5528391
3	java.io.IOException	5528391
3	java.io.UTFDataFormatException	5528391
3	java.lang.ArrayStoreException	5528391
3	java.lang.ClassCastException	5528391
3	java.lang.ClassFormatError	5528391
3	java.lang.CloneNotSupportedException	5528391
3	java.lang.Error	5528391
3	java.lang.Exception	5528391
3	java.lang.IllegalArgumentException	5528391
3	java.lang.IllegalMonitorStateException	5528391
⋮	⋮	⋮
70	org.omg.CORBA.portable.OutputStream	714960
⋮	⋮	⋮
6191	chapter12.CalcClient	5825

3.6 関連研究：SPARS-J

静的利用関係モデルによる部品評価値の応用として，我々の研究グループでは，Java のクラスを対象としたソフトウェア部品検索エンジン **SPARS-J** の開発を行っている [20]．様々な部品を web を利用して収集し，収集した部品全ての評価値を求めておく．利用者は，Web 検索エンジンのように，自分が再利用したい部品や機能を検索キーとして入力，検索を行うと，それに関連のある部品が評価値の高い順にソートされて出力される． 図 6 に，SPARS-J のシステム構成図を示す．

SPARS-J では，CR 法による部品評価値計算のほかに，部品類似度の計算を行っている．部品が再利用される際，全ての場合で元の部品がそのまま再利用されるわけではなく，いくつかの変更を加えて再利用されることもある．そのため，個々の部品を別々に評価するよりも，類似した部品を一つにまとめて評価値を与える方が適切であると考えれるためである．

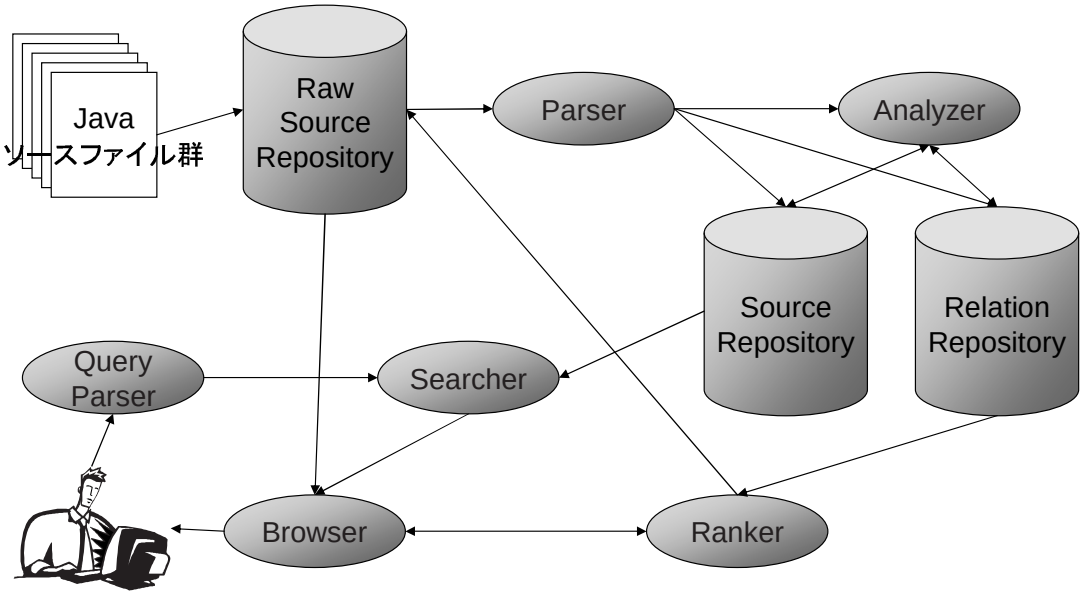

図 6: SPARS-J

SPARS-J システムを用いることにより，同じ操作を行う複数の部品の中から，多くの部品で利用されている汎用性の高い部品の選択が可能となり，再利用の支援になる．

4　動的情報を利用した部品評価手法

3 節で説明した CR 法は，再利用を用いたソフトウェア開発モデルに従い，利用実績に基づいて部品評価値を求めている．しかし，複数の部品で実装される”機能”単位での再利用を目指した時，従来の CR 法では不十分である．そこで，本研究では，機能の再利用を目的とした，動的情報を利用したソフトウェア部品評価手法の提案を行う．

提案手法では，次の順で部品評価値を求める．

1. ソフトウェアを実行し，動的利用関係を抽出
2. 利用関係をもとに，部品グラフを作成
3. CR 法を用い，部品評価値を求める

以降，まず機能再利用モデルの概念について 4.1 節で述べ，その後提案手法について説明を行う．

4.1　機能再利用モデル

CR 法による部品評価値は，様々なソフトウェアで利用されるライブラリのような汎用性の高い部品を検索するのに有効な手法である．しかし，開発者が再利用を行う際，これら汎用性の高い部品の検索が可能となっても，それらが機能的に「何を」実装している部品なのかわからなければ,「再利用できるであろう」部品でしかなく，効率のよい開発を行うことができない．また，個々の部品が「何を」実装しているか理解できても，それが他の部品から「いつ」「どのように」利用されているか十分に想定できなければ，再利用を行っても不具合を検出する恐れがある．

よって，ある機能を実装している部品を再利用したいと考えたとき，個々の部品だけでなく，機能を実装している部品が相互に「いつ」「どのように」利用しているかという情報を加えて開発者に示す必要があると考えれる．

そこで本研究では，ソフトウェアを実行し，実行時に生じる利用関係を元に部品評価値を計算する手法の提案を行う．本提案手法を用いることにより，実行した機能実装に重要となる部品の検出を行うと共に，動的な利用関係から,「いつ」「どのように」利用されているかという情報の出力が可能となる．

以降，本提案手法について説明を行う．

4.2　動的利用関係

再利用を用いたソフトウェア開発モデルでは，開発者が重要であると判断された部品が再利用され，時間の経過と共にその利用関係も洗練されていくというモデルであった．このモデルにおける部品の再利用では，開発者は再利用した部品を，開発中の部品に静的な情報として記述を行う．よって，部品間に存在する利用関係は静的な利用関係である．

これに対し，動的な利用関係とは，ソフトウェアを実行した際に得られる呼び出し情報から得られる利用関係である．実際に実行時に呼び出された部品にのみ，利用関係が存在するため，実行方法によって，部品間の利用関係は異なる．また，静的な利用関係が存在する部品でも，実行時に呼び出されなければ，利用関係は存在しないものとして扱う．

動的な情報を用いることで，部品間の利用関係だけでなく，それらが「いつ」「どのように」利用されているかという情報の取得も可能となる．よって，機能の再利用を目的とした場合，この動的な利用関係を用いて，部品評価値を求める手法が妥当であると考えられる．

静的な利用関係では，利用する側の部品が実際に実行されるかどうかは問題にならないため，利用される部品に重みが置かれる．しかし，動的な利用関係では，部品が利用するされるかどうかは，利用する部品に依存している．つまり，利用する部品が利用される部品を支配している．

よって，動的な利用関係を用いた部品の評価値では，次の式が成り立つ．

$$(\text{部品の評価値}) = (\text{支配している部品数})$$

また，静的な利用関係と同様，単純に支配数だけで評価値を決定するのではなく，どのような部品から利用されたかによって重みづけをする．多くの部品を支配しているような重要な部品をさらに支配している品は，再利用性が高いとみなして，高い重みを持つようにする．

$$(\text{部品の重み}) = (\text{支配元部品の評価値}) \times (\text{支配先に対する配分率})$$

4.3 CR 法の利用

動的な利用関係モデルを， 3 節で説明した CR 法を利用し，部品評価値を求める． 4.2 節で説明したように，動的な利用関係モデルによる部品評価値計算では，支配されている部品から支配している部品へ重みを分配する．よって，辺の重み，頂点の重みの定義を次のように再定義する．

定義 6 (辺の重み (動的利用関係))

頂点 v_i から v_j への辺 e_{ij} に関するの重み $w'(e_{ij})$ を次のように定義する．

$$w'(e_{ij}) = d_{ij} \times w(v_j) \tag{7}$$

定義 7 (頂点の重み (動的利用関係))

$\mathrm{OUT}(v_i)$ を v_i を始点とする (v_i から出て行く) 有向辺の集合とした時，頂点 v_i の重みは，v_i が始点となる有向辺 e_{ik} の重みの総和と定義する．

$$w(v_i) = \sum_{e_{ik} \in \mathrm{OUT}(v_i)} w'(e_{ik}) \tag{8}$$

図 7 はこれら定義を図示したものである．

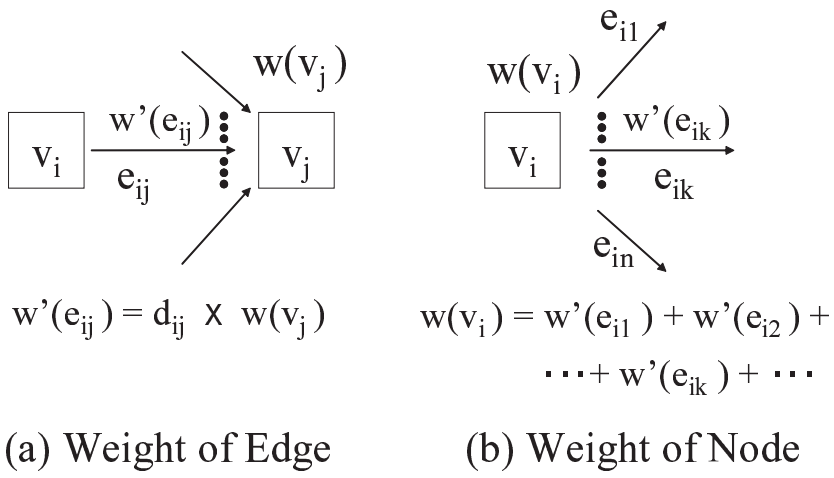

図 7: 重みの定義 (動的利用関係)

この時，式 (8) に式 (7) を代入して，次式を得る．

$$w(v_i) = \sum_{e_{ik} \in \mathrm{OUT}(v_i)} d_{ik} \times w(v_k) \tag{9}$$

また，補正を加えた配分率も以下のように変更する．

定義 8 (補正を加えた配分率 (動的利用関係))

$$d'_{ij} = \begin{cases} p \times d_{ij} + (1-p)/n & \text{if } v_j \text{ を終点とする有向辺が存在} \\ 1/n & \text{if } v_j \text{ を終点とする有向辺が存在} \end{cases} \tag{10}$$

この式を元に，で説明した CR 法による部品評価値計算手法を元に，部品評価値を求める．

5　部品評価システム

4 節で説明した，動的情報を利用したソフトウェア部品評価手法を基に，オブジェクト指向言語 Java を対象とした，ソフトウェア部品評価システム「**Component Rank System**」(以降，CR システム) の実装を行った [21, 22]．以降，CR モデルと Java の概念の対応，システムの構成，について説明を行う．

5.1　CR モデルと Java の概念の対応

CR 法を Java に適用する際の，モデルと Java の概念との対応を以下に示す．

- モデルと Java の概念との対応

  部品:　クラスを部品の単位とする．
  ただし，Java は原則として 1 つのソースコードファイルには 1 つのクラスを記述するため，ソースコードファイルも部品単位とみなすことが可能である．

  利用関係:　クラスの継承，インターフェースおよび抽象クラスの実装，メソッドの呼びだし，フィールド参照とする．

  分配率:　利用関係の種類・頻度にかかわらず，分配率は全て等しくする．

  辺と擬似辺の重みの配分比率 p:　p として 0.85 を採用する．

5.2　システム構成

CR システムによる部品評価値計算手順と，各処理部との対応を，以下に示す．

1. プログラムを実行し，実行履歴から，利用関係を抽出
   - 利用関係解析部
2. 部品間の利用関係から，部品グラフを生成
   - 部品グラフ生成部
3. CR 法による部品評価値計算し，順位付けを行い出力
   - 部品評価値計算部

以降，5.2.1 節で利用関係解析部，5.2.2 節で部品グラフ生成部，5.2.3 節で部品評価値計算部の説明を行う．

5.2.1 利用関係解析部

Java では，Java Virtual Machine Debug Interface (JVMDI) と呼ばれる，VM によって実装されているネイティブインタフェースが用意されており，これを利用することで容易にメソッド呼び出しの情報を得ることができる [23].

JVMDI のクライアントは、アプリケーションプログラム内で発生する多くのイベントについての通知を受けることができ，必要なイベントを監視することで利用関係を取得する． 5.1 節で定義した，Java における利用関係の取得と，イベントとの関係を以下に示す.

- クラス継承
  - JavaVM ではクラスインスタンスが生成される際，インスタンス初期化メソッド $\langle init \rangle$ が実行される．サブ (子) クラスの $\langle init \rangle$ メソッドが実行されると，そのメソッドからスーパー (親) クラスの $\langle init \rangle$ メソッドが実行される．よって，クラス継承の利用関係の取得は，メソッド呼び出しの利用関係と同じ処理で取得可能である.
- インタフェースならびに抽象クラスの実装
  - 新しいクラスが実行されるには，必ずそのクラスがロードされる．この時，VM 内でクラスロードイベント (JVMDI_EVENT_CLASS_LOAD) が発生する．JVMDI ではこのクラスロードイベント発生時に，クラスの情報を取得することができ，実装しているインタフェース，抽象クラス名を取得できる.
- メソッド呼び出し
  - メソッドが呼び出される時には，メソッド呼び出しイベント (JVMDI_EVENT_METHOD_ENTRY) が，終了するときには，メソッド終了イベント (JVMDI_EVENT_METHOD_EXIT) が発生する．これらイベントを監視し，メソッドが呼び出される毎に，スタックの先頭にその情報を保存し，メソッド終了時に，スタックの先頭を削除する．スタックを用意することで，新しくメソッドが実行された際に，スタックの先頭がそのメソッドを呼び出したクラス (メソッド) であることがわかり，利用関係の取得が行える．また，スレッド毎にスタックを用意することで，マルチスレッドに対応した利用関係の取得も可能である.
- フィールド参照
  - フィールドがアクセスまたは変更される際，フィールドアクセスイベント (JVMDI_EVENT_FIELD_ACCESS) が発生する．この時，メソッド呼び出しで作成したスタックの先頭クラスから，アクセスされたフィールドのクラスへの利用関係の取得が可能である.

これらの利用関係は，プログラムの開始 (JVMDI_EVENT_VM_INIT) から終了 (JVMDI_EVENT_VM_DEATH) までの間に取得される．利用関係とそれが取得できる VM イベントとの対応表を, 表 2 に示す.

表 2: 利用関係とイベント

利用関係	イベント名	発生方法
クラスの継承	JVMDI_EVENT_METHOD_ENTRY	メソッドが呼び出される
インターフェース実装 抽象クラス実装	JVMDI_EVENT_CLASS_LOAD	クラスがロードされる
メソッド呼び出し	JVMDI_EVENT_METHOD_ENTRY JVMDI_EVENT_METHOD_EXIT	メソッドが呼び出される メソッドが終了する
フィールド参照	JVMDI_EVENT_FIELD_ACCESS	フィールドをアクセスまたは変更する
プログラムの開始	JVMDI_EVENT_VM_INIT	VM の初期化が完了する
プログラムの終了	JVMDI_EVENT_VM_DEATH	VM が終了する

また，JVMDI を利用したプロファイリング機能の実装構成図は 図 8 のようになる．

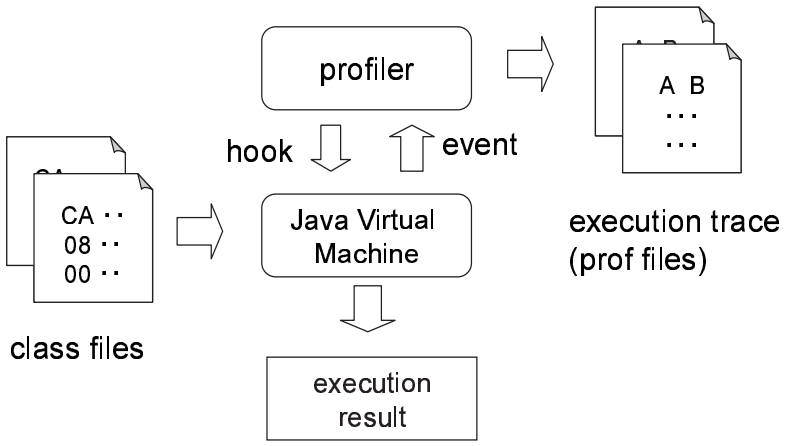

図 8: プロファイラー

5.2.2 部品グラフ生成部

部品グラフは，$n \times n$ の行列 D 上 (部品数 n) に表現する．部品 C_i が部品 C_j を利用するとき，行列 D(i,j) の値を 1 とし，部品 C_i が部品 C_j を利用しないとき，行列 D(i,j) の値を 0 とする．

- 部品グラフの生成

  for($i = 1$; $i <= n$; $i++$)
  　for($j = 1$; $j <= n$; $j++$)
  　　if (C_i が C_j を利用)
  　　　$D(i,j) = 1$
  　　else
  　　　$D(i,j) = 0$

5.2.3 部品評価値計算部

4.3 節で説明したように，部品の評価値，すなわち頂点の重みは，行列の固有値計算から求められる．まず，部品グラフ生成部で作成した行列 D に，補正を加えた配分率を適応する．式 (6) より，

$$D(i,j) = \begin{cases} p \times d_{ij} + (1-p)/n & \text{if } \exists j \ D(i,j) = 1 \\ 1/n & \text{if } \forall j \ D(i,j) = 0 \end{cases} \tag{11}$$

となる．CR システムでは，$p = 0.85$，配分率 $d_{ij} = 1/m$(ただし，m は頂点 v_i を始点とする有向辺の数) 配分率としている．適用後の行列 D の転置行列 D^t について，固有値計算を行うが，この計算には Java で行列演算を行うパッケージ JAMA を利用している [27].

5.3 システムの構成

CR システムの構成図を 図 9 に示す.

Java 実行ファイル (クラスファイル) を JavaVM 実行する際，利用関係解析部 (Profiler) を用いて実行履歴を取得し，利用関係 (prof file) を抽出する．次に抽出された利用関係から部品グラフ生成部 (Dynamic Relation Analyzer) が部品グラフを生成する．最後に生成された部品グラフから部品評価値計算部 (Ranker) で部品評価値を計算し，出力する．

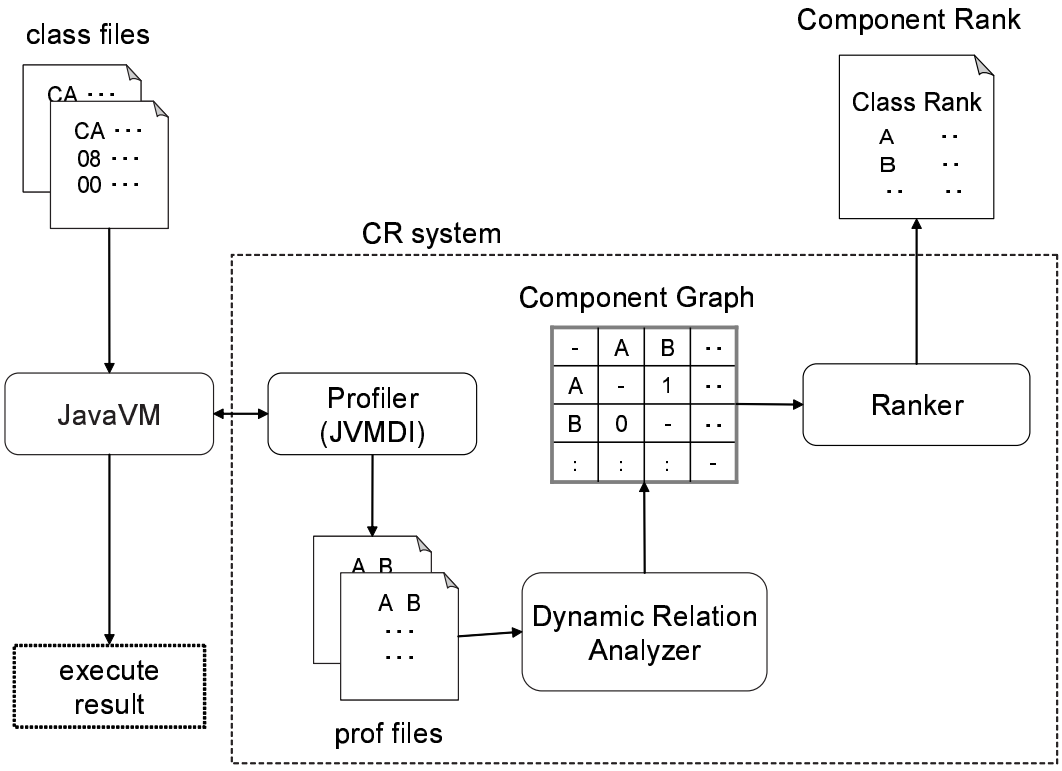

図 9: CR システムの構成

6　評価実験

5 節で説明した CR システムを用いて評価実験を行った．以降，実験結果ならびに考察について述べる．

6.1　実験内容

対象プログラムとして，Java 2 Software Development Kit, Standard Edition 1.4.0 付属の j2sdk1.4.0_01.demo.jfc.Notepad と，javac を選んだ．前者プログラムは，19 クラス (内部クラスも含む) から成り立つ，単純なエディタアプリケーションであり，後者プログラムは，全 159 クラスから成り立つ，Java コンパイラである． 図 10 はこの前者 Notepad プログラムのスナップショットである．

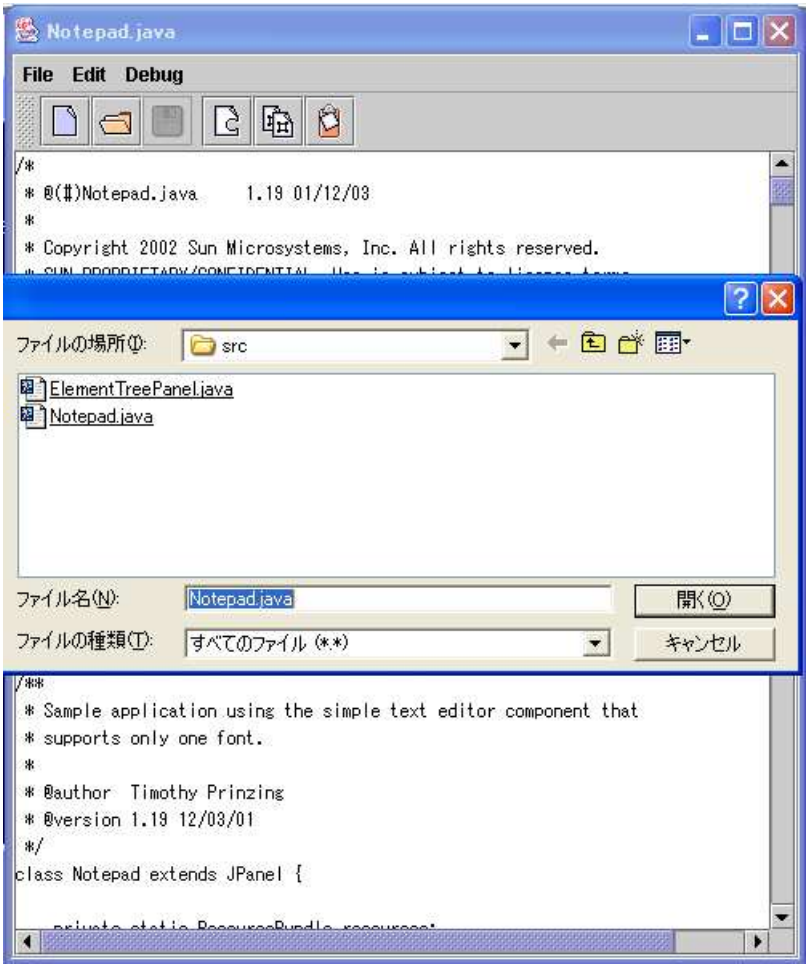

図 10: Notepad スナップショット

6.2 実験結果

動的利用関係を行うには，当然対象プログラムを実行する必要がある．Notepad アプリケーションの実行を行う際，以下の実行方法を用いた．

1. いくつかの機能をランダムに実行
2. 「ファイルを開く」のみ実行
3. 「コピー&ペースト」のみ実行

以上の実行方法により得られた部品評価値ならびに，その部品 (クラス) が利用する部品数を，それぞれ 表 3 ， 表 4 ， 表 5 に示す．

また，javac プログラムの実行には，次の引数を与えた．

1. 正常にコンパイルが終了するプログラム
2. 構文エラーを出力し終了するプログラム
3. 存在しないプログラム
4. 引数に何も与えない

以上の実行方法により得られた部品評価値ならびに，その部品が利用する部品数を，それぞれ上位 15 位まで 表 6 ， 表 7 ， 表 8 ， 表 9 に示す．

なお，部品評価値は，従来小数で表示されるが，わかりやすく表示するため 1 億倍した値を記述している．また，$\langle System \rangle$ クラスは，ユーザが記述したクラスではなく，VM 内部で実行されるクラス (例えば，クラスをロードする ClassLoader クラスや，イベントを扱う EventHandler クラスなど) を表している．

6.3 考察

6.3.1 計算手法の有効性

実験結果より，利用する部品数の多い部品ほど上位に位置することがわかる．spearman の順位相関係数を用いて，部品評価順位と部品利用数との間の順位相関係数を求めたところ，Notepad では平均 -0.9906 と非常に高い負の相関が見られた．また，javac でも -0.8760 と高い負の相関が見られた．

しかし，javac で上位に位置する部品に注目した場合，決して利用部品数が多いほうが評価値が高いわけではないことがわかる．特に，com.sun.tools.javac.Main クラスは，全ての実行方法で利用部品数が 2 個であるにも関わらず，上位に位置している．com.sun.tools.javac.Main クラスは，この javac プログラムを起動する際，1 番初めに呼び出されるクラスであり，このクラスが存在しなければ，プ

表 3: Notepad：ランダム

評価値順位	クラス名	部品評価値	利用部品数	利用数順位
1	$\langle System \rangle$	16207350	16	1
2	Notepad	13356511	12	2
3	Notepad$UndoAction	5732073	5	3
4	Notepad$OpenAction	5361253	4	4
5	Notepad$ShowElementTreeAction	5063501	3	6
6	Notepad$UndoHandler	4803297	3	6
7	ElementTreePanel	4740721	4	4
8	Notepad$3	4732369	3	6
9	Notepad$FileLoader	4731713	2	10
10	Notepad$2	4358630	2	10
11	ElementTreePanel$ElementTreeModel	4306154	3	6
12	ElementTreePanel$1	3811945	2	10
13	Notepad$NewAction	3430323	1	13
14	Notepad$RedoAction	3189917	1	13
15	Notepad$ExitAction	2695707	0	15
15	Notepad$ActionChangedListener	2695707	0	15
15	Notepad$1	2695707	0	15
15	Notepad$StatusBar	2695707	0	15
15	Notepad$AppCloser	2695707	0	15
15	ElementTreePanel$2	2695707	0	15

表 4: Notepad：ファイルオープン

評価値順位	クラス名	部品評価値	利用部品数	利用数順位
1	Notepad	28475829	12	1
2	$\langle System \rangle$	14920808	4	2
3	Notepad$OpenAction	10882916	3	3
4	Notepad$FileLoader	7715919	1	4
5	Notepad$ShowElementTreeAction	3800453	0	5
5	Notepad$UndoHandler	3800453	0	5
5	Notepad$UndoAction	3800453	0	5
5	Notepad$NewAction	3800453	0	5
5	Notepad$RedoAction	3800453	0	5
5	Notepad$ExitAction	3800453	0	5
5	Notepad$ActionChangedListener	3800453	0	5
5	Notepad$1	3800453	0	5
5	Notepad$StatusBar	3800453	0	5
5	Notepad$AppCloser	3800453	0	5

表 5: Notepad：コピー&ペースト

評価値順位	クラス名	部品評価値	利用部品数	利用数順位
1	Notepad	30736333	11	1
2	$\langle System \rangle$	14178601	3	2
3	Notepad$UndoHandler	12237346	3	2
4	Notepad$UndoAction	5178274	1	4
4	Notepad$OpenAction	5178274	1	4
6	Notepad$ShowElementTreeAction	4061397	0	6
6	Notepad$NewAction	4061397	0	6
6	Notepad$RedoAction	4061397	0	6
6	Notepad$ExitAction	4061397	0	6
6	Notepad$ActionChangedListener	4061397	0	6
6	Notepad$1	4061397	0	6
6	Notepad$StatusBar	4061397	0	6
6	Notepad$AppCloser	4061397	0	6

表 6: javac：正常終了

評価値順位	クラス名	部品評価値	利用部品数	利用数順位
1	com.sun.tools.javac.v8.Main	6230715	17	11
2	com.sun.tools.javac.v8.JavaCompiler	5401857	24	7
3	com.sun.tools.javac.v8.comp.Gen	3308156	36	1
4	com.sun.tools.javac.Main	2911060	2	53
5	com.sun.tools.javac.v8.comp.Attr	2844601	34	2
6	com.sun.tools.javac.v8.comp.Enter	2588095	28	3
7	com.sun.tools.javac.v8.comp.TransInner	2007288	27	4
8	com.sun.tools.javac.v8.code.ClassReader	1949322	26	5
9	com.sun.tools.javac.v8.comp.Enter$CompleteEnter	1821256	16	14
10	com.sun.tools.javac.v8.comp.TransTypes	1797646	26	5
11	com.sun.tools.javac.v8.tree.TreeMaker	1791180	19	10
12	com.sun.tools.javac.v8.parser.Parser	1779365	8	20
13	com.sun.tools.javac.v8.comp.Enter$MemberEnter	1699103	17	11
14	com.sun.tools.javac.v8.comp.Flow	1671419	21	8
15	com.sun.tools.javac.v8.comp.Symtab	1581438	15	15

表 7: javac：エラー終了

評価値順位	クラス名	部品評価値	利用部品数	利用数順位
1	com.sun.tools.javac.v8.Main	7410380	17	7
2	com.sun.tools.javac.v8.JavaCompiler	6999128	22	4
3	com.sun.tools.javac.v8.comp.Attr	3939811	34	1
4	com.sun.tools.javac.v8.comp.Enter	3435564	28	2
5	com.sun.tools.javac.Main	3429715	2	43
6	com.sun.tools.javac.v8.comp.Enter$CompleteEnter	2500022	16	10
7	com.sun.tools.javac.v8.code.ClassReader	2486254	26	3
8	com.sun.tools.javac.v8.comp.Flow	2466759	22	4
9	com.sun.tools.javac.v8.tree.TreeMaker	2444630	19	6
10	com.sun.tools.javac.v8.parser.Parser	2234598	8	16
11	com.sun.tools.javac.v8.comp.Enter$MemberEnter	2180470	17	7
12	com.sun.tools.javac.v8.comp.Symtab	2037506	15	11
13	com.sun.tools.javac.v8.comp.Resolve	1691420	15	11
14	com.sun.tools.javac.v8.util.Log	1516791	7	17
15	com.sun.tools.javac.v8.code.Symbol	1497920	11	13

表 8: javac：存在しないプログラム

評価値順位	クラス名	部品評価値	利用部品数	利用数順位
1	com.sun.tools.javac.v8.JavaCompiler	11287694	19	1
2	com.sun.tools.javac.v8.Main	10479894	17	3
3	com.sun.tools.javac.Main	4768826	2	23
4	com.sun.tools.javac.v8.comp.Symtab	4212854	15	5
5	com.sun.tools.javac.v8.comp.Enter	3954821	17	3
6	com.sun.tools.javac.v8.code.ClassReader	3149856	18	2
7	com.sun.tools.javac.v8.util.Log	2028882	5	12
8	com.sun.tools.javac.v8.comp.Enter$CompleteEnter	1966526	7	10
9	$\langle System \rangle$	1886880	1	42
10	com.sun.tools.javac.v8.tree.Tree$TopLevel	1771217	3	18
11	com.sun.tools.javac.v8.comp.Gen	1588532	9	6
12	com.sun.tools.javac.v8.code.Symbol	1586177	9	6
13	com.sun.tools.javac.v8.code.Type	1532599	9	6
14	com.sun.tools.javac.v8.code.Symbol$PackageSymbol	1457690	5	12
15	com.sun.tools.javac.v8.comp.Attr	1362777	8	9

表 9: javac：引数なし

評価値順位	クラス名	部品評価値	利用部品数	利用数順位
1	com.sun.tools.javac.v8.Main	28144312	15	1
2	com.sun.tools.javac.Main	10803013	2	2
3	com.sun.tools.javac.v8.Main	8341994	2	2
4	com.sun.tools.javac.v8.comp.Symtab	5643229	1	6
5	com.sun.tools.javac.v8.Main	3689524	2	2
5	com.sun.tools.javac.v8.Main	3689524	2	2
7	com.sun.tools.javac.v8.util.Log	3686118	1	6
7	com.sun.tools.javac.v8.comp.Enter$CompleteEnter	3686118	1	6
7	$\langle System \rangle$	3686118	1	6
10	com.sun.tools.javac.v8.Main$1	3182210	1	6
10	com.sun.tools.javac.v8.Main$2	3182210	1	6
10	com.sun.tools.javac.v8.Main$3	3182210	1	6
10	com.sun.tools.javac.v8.Main$4	3182210	1	6
10	com.sun.tools.javac.v8.Main$5	3182210	1	6
10	com.sun.tools.javac.v8.Main$6	3179750	1	6

ログラムが実行されることはない．よって，プログラム内で重要な部品の 1 つであると考えられる．しかし，単純に利用部品数だけで部品の評価を行ったならば，com.sun.tools.javac.Main クラス 50 位前後に位置する．

提案手法を用いた部品評価値では，利用部品数に加え，利用している部品の重要度も加味しているため，ほとんど他の部品を利用しないようなシステムの末端で利用される部品 (クラスの) 評価値をおさえると共に，利用数は少ないが，プログラムを実行する上で重要となる部品の評価値を高くなる．よって，本提案手法は，機能の再利用を実現する上で，有効な手法であると考えられる．

しかし，今回の実験では，対象プログラムが小規模なプログラムであったため，大規模プログラムに適応した場合においても，有効な手法であるか検証する必要がある．また，他の部品評価手法と比較する必要もあると考えられる．

6.3.2 部品評価値と実行方法

実行方法が異なれば，当然動的な利用関係も変化する．実行方法がどの程度部品評価値に影響を与えているか考察する．

Notepad プログラムの実行方法の 2 では「ファイルを開く」という動作を行っている．その時の部品評価値の結果では，Notepad$OpenAction クラスや Notepad$FileLoader クラスが，他の結果と比較して上位に位置していることがわかる．これらクラスは,「ファイルを開く」という機能の動作を行った際に，実際に呼び出されるクラスであり，部品評価値計算でこれら部品が上位に位置することは，正しい結果であると考えれる．

また，javac プログラムでも，エラーを出力して終了するプログラムを引数に与えて実行すれば，エラー出力を行う com.sun.tools.javac.v8.util.Log が正常に終了する場合と比較して，上位に位置していることがわかる．

実験を行ったサンプル数は少ないが，以上の実験結果よりでも，本提案手法を用いて部品評価を行うことで，実行方法に依存した部品評価値が得られることがわかる．この結果を用いることにより，特定の機能を実現している部品の再利用が可能になるのではないかと考えられる．

7　部品評価値の利用

動的利用関係モデルによる部品評価値の応用として，リバースエンジニアリングへの適応を行う．リバースエンジニアリングとは，通常のソフトウェア開発プロセスの逆の流れでソフトウェアの開発等をサポートしようとするものである．既存のソフトウェアを解析し，そのソフトウェアがどのようなモデルであるかを示すことで，ソフトウェア理解につながり，その結果再利用支援にもつながると考えれる．

7.1　シーケンス図

部品評価値のリバースエンジニアリングへの応用として，**シーケンス図**(Sequence Diagram) への適応を行う．シーケンス図とは，部品間の利用関係を表すと共に，実行方法によって変化する動的な利用関係を同時に表現する関係図である． 図 11 にシーケンス図の簡単な例を示す．

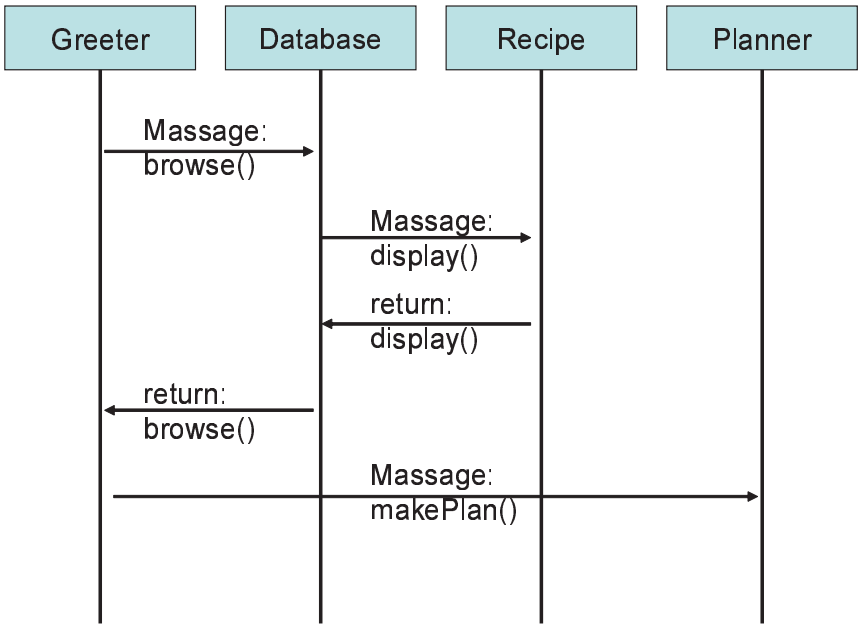

図 11: シーケンス図の例

この図に部品から上下に伸びている線は，時間軸を表し，時間軸から左右にラベルつきで伸びた有向辺は，部品間利用関係 (メッセージのやり取り) を表している．

単純に，ソフトウェアを実行した履歴だけからこのシーケンス図を作図したとする．小規模なソフトウェアの場合なら，部品数，部品間のメッセージのやり取りも少ないために，全ての部品間の利用関係を出力しても問題ない．しかし，少し大規模なソフトウェアになると，部品数，部品間のメッセージが増えるため，全ての部品間の利用関係を出力すれば，複雑な利用関係のためソフトウェア理解支援するどころか，かえって困難にしてしまう恐れがある．

そこで，提案手法で計算された部品評価値でフィルタリングを行い，重要な部品間のみシーケンス図上に表現する．部品 C_i に対応する頂点 v_i の重み $w(v_i)$ とし，フィルタリング値として w_limit が与えられたとする．この時 $w(v_i) \geq w_limit$ を満たす部品 C_i のみ，シーケンス図のノードとして出力する．また，部品間の利用関係については，$w(v_i) \geq w_limit$ かつ $w(v_j) \geq w_limit$ を満たす，部品 C_i と部品 C_j との利用関係のみ出力する．

6 節の実験で利用した，Notepad プログラムのファイルオープンのみ実行した結果についてシーケンス図を作図する． 図 13 にフィルタリングを行っていないシーケンス図の一部を， 図 14 に Notepad$FileLoader クラスの部品評価値 0.07715919 でフィルタリングを行ったシーケンス図の一部を示す．なお，部品の上の数字は，部品評価値 $(*10^8)$ を表している．

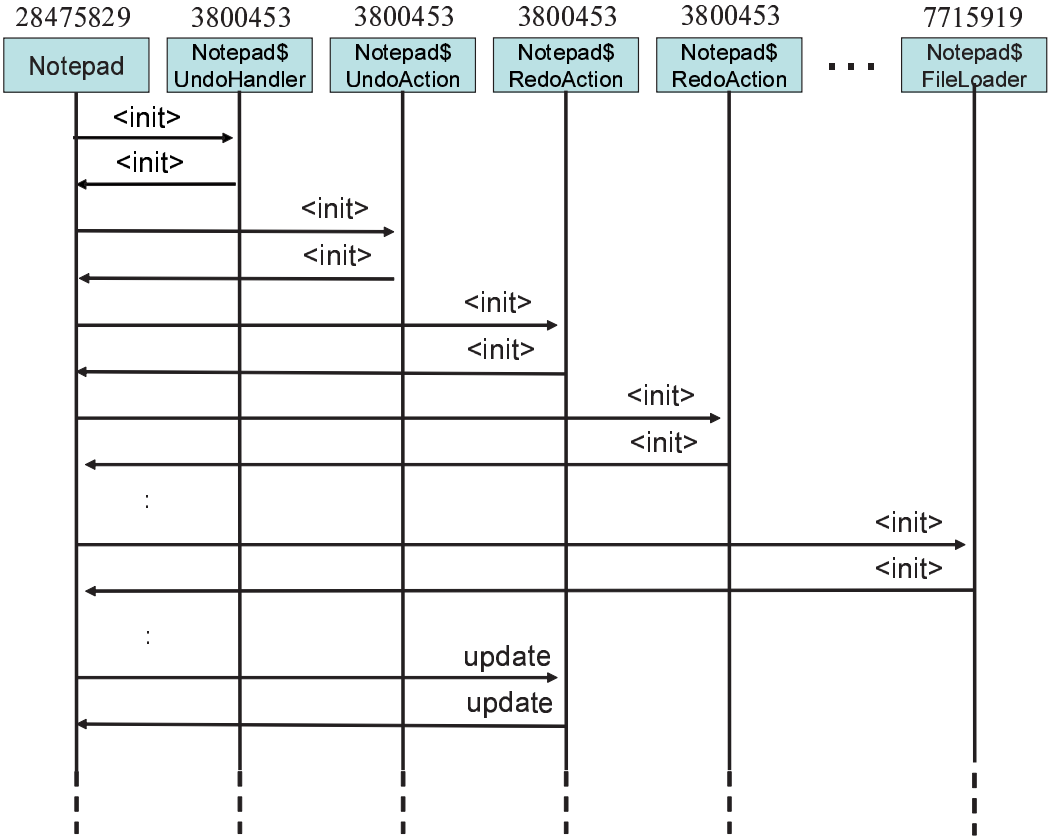

図 12: Notepad シーケンス図 (フィルタリングなし)

出力結果からもわかるように，フィルタリングを行ったシーケンス図では，3 つの部品ならびにそれら部品間の呼び出し関係のみ出力されている．適当な部品評価値でフィルタリグを行うことで，出力する部品を減らすことができ，開発者が注目すべき部品を絞り込める．

また，単純に表示する部品数を減らしているわけではなく，部品評価値でフィルタリングを行っているので，実行した機能実装に必要な部品を出力し，あまり重要でない部品を除くことが可能と

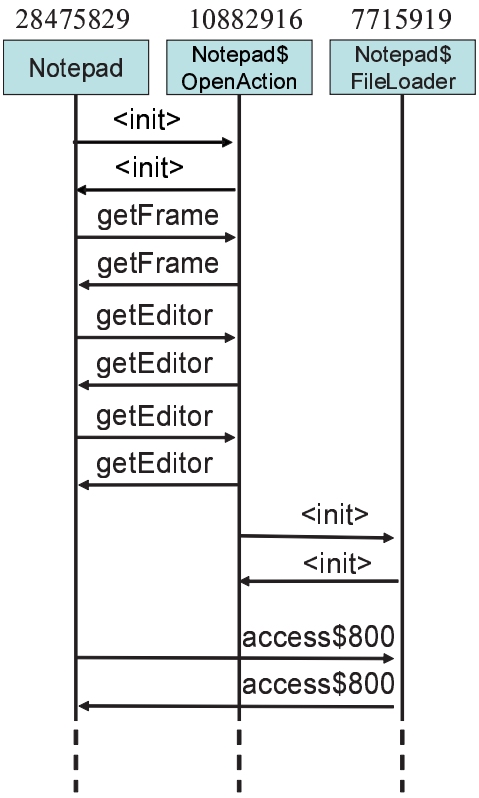

図 13: Notepad シーケンス図 (フィルタリングあり)

なる．Notepad プログラムのファイルオープン機能のみ実行した結果をフィルタリングした例では，Notepad，Notepad$OpenAction，Notepad$FileLoader クラスのみが出力されているが，ファイルオープンを機能実装している部品はこの 3 部品であり，開発者が注目したいはずの部品のみが出力されている．

今回の例では，部品数が少ないため機能実装に必要な部品のみが出力されている．ソフトウェアが大きくなれば，必ずしも機能実装に必要な部品のみが出力されるわけではないが，適当なフィルタリング値を選ぶことで，少なくとも重要でない部品を除いて出力することは可能であると考えられる．

以上のことから，適当な評価値でフィルタリングを行うことで，ユーザが注目すべき部品を絞り込め，結果ソフトウェア理解支援につながると考えられる．

しかし，部品評価値でフィルタリングを行う際，どのような値でフィルタリングを行うのか，という問題がある．今回実装したプロトタイプシステムでは，ユーザがその値を選ぶというかたちで実装を行っているが，実利用を考えたとき，対象プログラムがどのようなプログラムか全く知識がないユーザが，適当な値を選択できるのかどうかわからない．そのため，重要な部品がどれかを判断し，自動的にフィルタリングする部品評価値を決定する機構が必要であると考えられる．また，今回の実装方法では，部品評価値のみフィルタリングの対象としており，利用関係 (有向辺) の重みについて

は対象としていない．その結果，ループが存在するようなソフトウェアについては，何度も同じメッセージのやり取りが出力されてしまい，利用関係図が大きくなってしまう．そのため，辺の重みについてもフィルタリングを行う必要があると考えられる．

さらに，今回の実験結果は，比較的小規模なプログラムを対象としている．その理由として，実行履歴の保存に大きなコストがかかるため，今回実装を行ったプロトタイプシステムでは，大規模プログラムに適応できないためである．そのため，大規模なソフトウェアで同様の実験を行い，有効性を評価する必要があると考えられる．

7.2 CR システムへの追加実装

動的な利用関係と，部品評価値，フィルタリング値をもとに，シーケンス図を出力する機能を，CR システムに追加実装した． 図 14 に出力結果を示す.

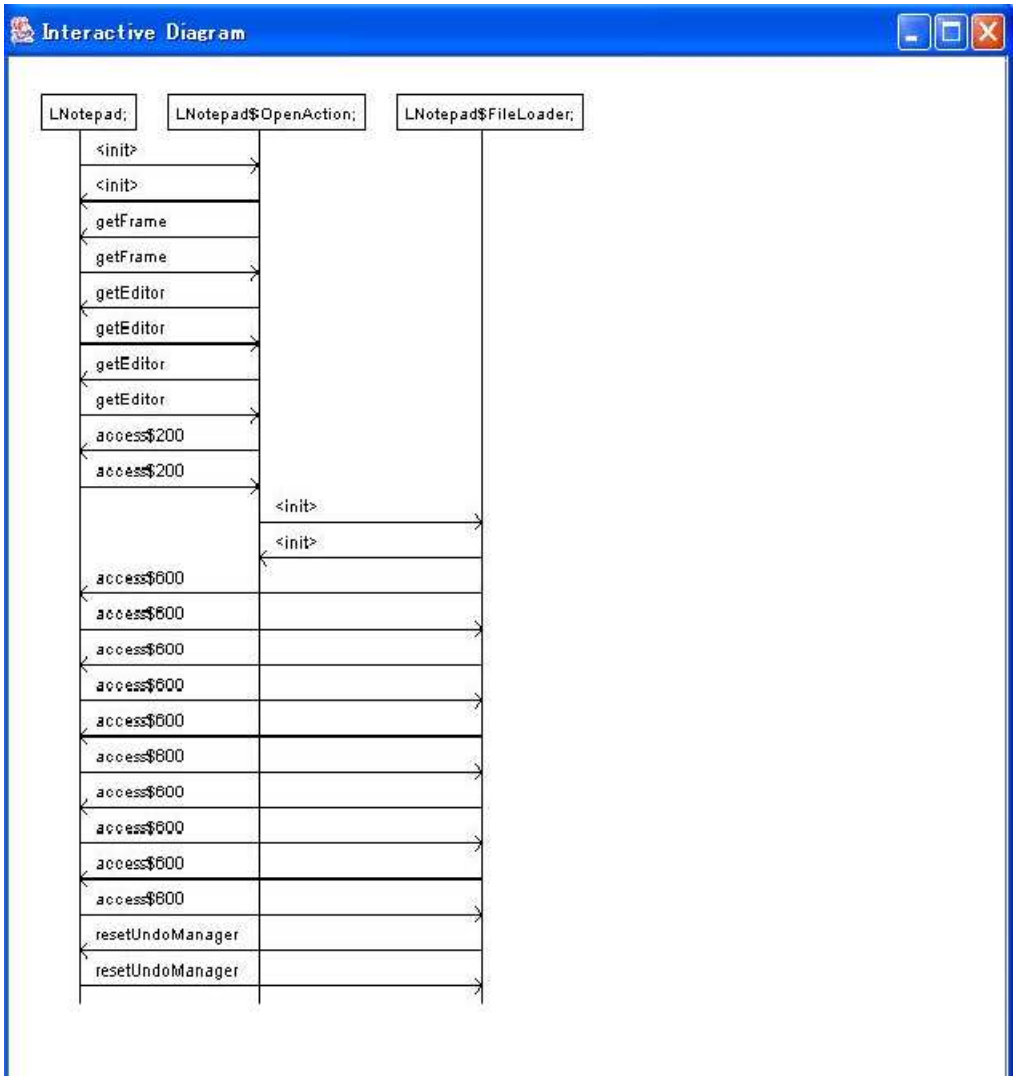

図 14: シーケンス図出力結果

8 まとめ

本研究では，動的な利用関係モデルに基づいて，動的な情報を利用した部品評価手法を提案した．そして，提案手法を評価システムに実装し，実際に Java アプリケーションに適用実験を行い，本手法の有効性について確認を行った．また，部品評価値の利用方法として，シーケンス図への適用を行った．評価値でフィルタリングを行い作図することで，ソフトウェア理解支援につながることを確認した．

本提案手法を用いることで，重要な部品の検索だけでなく，それら部品の相互利用関係も取得できるため，開発者は機能単位での再利用を行うことが可能となり，効率よく再利用が行えられると考えている．

今後の課題として，以下のことが挙げられる．

- より多くの大規模アプリケーションへの適応
  - 実行履歴取得の効率化
- 他の部品評価手法との比較
- 利用関係の重み付け
  - 利用関係の種類による変更
  - 呼び出し回数による変更
- フィルタリングしたシーケンス図の有効性評価
- フィルタリングの対象追加
  - 有向辺の重みでフィルタリング

謝辞

本論文の作成において，常に適切な御指導および御助言を賜りました大阪大学 大学院情報科学研究科 コンピュータサイエンス専攻井上克郎 教授に深く感謝致します.

本論文の作成において，常に適切な御指導を賜りました 同 楠本真二 助教授に深く感謝致します.

本論文の作成において，常に適切な御指導を賜りました 同 松下誠 助手に深く感謝致します.

本論文の作成において，常に適切な御助言を頂きました大阪大学 大学院基礎工学研究科 情報数理系専攻 ソフトウェア科学分野 横森励士氏に深く感謝致します.

本論文の作成において，常に適切な御助言を頂きました 科学技術振興事業団 山本哲男氏に深く感謝致します.

最後に，その他の面で様々な御指導，御助言を頂いた大阪大学 大学院情報科学研究科 コンピュータサイエンス専攻井上研究室の皆様に深く感謝致します.

参考文献

[1] C. Braun: Reuse, in John J. Marciniak, editor, *Encyclopedia of Software Engineering*, Vol. 2, John Wiley & Sons, pp. 1055-1069 (1994).

[2] V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page and S. Waligora: "The software engineering laboratory - an operational software experience," *Proc. of ICSE14*, pp. 370-381 (1992).

[3] S. Isoda: "Experience report on a software reuse project: Its structure, activities, and statistical results," *Proc. of ICSE14*, pp.320-326 (1992).

[4] B. Keepence and M. Mannion: "Using patterns to model variability in product families," *IEEE Software*, Vol. 16, No. 4, pp. 102-108 (1999).

[5] 横森, 藤原, 山本, 松下, 楠本, 井上: "ソフトウェア部品間の利用関係を用いた再利用性評価手法の提案", ソフトウェア・シンポジウム 2002 論文集, pp.216–225, July 16–19, (2002).

[6] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, S. Kusumoto: "Component Rank: Relative Significance Rank for Software Component Search", Technical Report of SE Lab, Dept. of Computer Science, Osaka University, SEL-Oct-8-2002, Oct. (2002).

[7] 汎道: "Think! Software Reuse ソフトウェアの部品化・再利用", http://www5.airnet.ne.jp/handoh/Think_Reuse.htm

[8] 青山, 中所, 向山: コンポーネントウェア, 共立出版, (1998).

[9] I. Jacobson, M. Griss and P. Jonsson: Software Reuse: *Addison Wesley*, (1997).

[10] F. Narin, G. Pinski, and H. H. Gee: "Structure of the Biomedical Literature," *Journal of the American Society for Information Science*, Vol. 27, No. 1, 25-45, (1976).

[11] G. Pinski and F. Narin: "Citation Influence for Journal Aggregates of Scientific Publications: Theory, with Application to the Literature of Physics," *Information Processing and Management*, Vol. 12, No. 5, pp. 297-312, (1976).

[12] G. Pinski and F. Narin: "Structure of the Psychological Literature," *Journal of the American Society for Information Science*, Vol. 30, No. 3, pp. 161-168, (1979).

[13] L. Page, S. Brin, R. Motwani, T. Winograd: "The PageRank Citation Ranking: Bringing Order to the Web," http://www-db.stanford.edu/ backrub/pageranksub.ps

[14] T. H. Haveliwala: "Efficient Computation of PageRank", Stanford Technical Report, (1999).

[15] “Google,” http://www.google.com/

[16] L. H. Etzkorn, W. E. Huges Jr., C. G. Davis: “Automated reusability quality analysis of OO legacy software,” *Information and Software Technology*, Vol. 43, Issue 5, pp. 295-308 (2001).

[17] L. H. Etzkorn, J. Bansiya, C. G. Davis: “Design and code complexity metrics for OO classes”, *Journal of Object-Oriented Programming*, Vol. 12, No. 1, pp. 35-40, (1999).

[18] 山本, 鷲崎, 深澤: “再利用特性に基づくコンポーネントメトリクスの提案と検証,” ソフトウェア工学の基礎ワークショップ (FOSE2001), (2001).

[19] 山本, 鷲崎, 深澤: “静的側面から見たソフトウェアコンポーネントの品質”, 電子情報通信学会技術研究報告, vol. 101, No. 98, pp. 33-40, (2001).

[20] 山本, 横森, 松下, 楠本, 井上: “利用頻度に基づくソフトウェア部品の解析・検索システムの提案”, 電子情報通信学会技術研究報告, SS2002-17, Vol.102, No.329, pp.13-18 (2002).

[21] J. Gosling, B. Joy, G. Steele and G. Bracha: “The Java Language Spwcification Second Edition,” Sun Microsystems,
http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html

[22] J. ゴズリン, B. ジョイ, G. スティール, G. ブラーハ, 村上 (訳): Java 言語仕様 第 2 版, ピアソン・エデュケーション, (2000).

[23] “JavaTM Platform Debugger Architecture” Sun Microsystems,
http://java.sun.com/j2se/1.4/ja/docs/ja/guide/jpda/index.html

[24] 馬場: “Google の秘密 - PageRank 徹底解説,”
http://www.kusastro.kyoto-u.ac.jp/~baba/wais/pagerank.htm

[25] G. Booch: Object-Oriented Analysis and Design with Applications, *The Benjamin/Cummings Publishing* (1994).

[26] H. Fujiwara, S. Kusumoto, K. Inoue, T. Ootsubo and K. Yuura: “Evaluation of a Business Application Framework Using Complexity and Functionality Metrics”, *3rd International Conference on Product Focused Software Process Improvement(PROFES2001)*, pp 371-380, (2001)

[27] “JAMA : A Java Matrix Package,” http://math.nist.gov/javanumerics/jama/

[28] 古賀, 飯田, 松本, 井上: “ソフトウェアコンポーネント利用情報の収集と共有”, 電子情報通信学会技術報告, vol.100, No.472, pp. 1-8, (2000).

[29] “The Source For Java(TM) Technology,” Sun Microsystems, http://java.sun.com/

[30] 藤井, 横森, 山本, 井上: “動的情報を利用したソフトウェア部品評価手法の提案と評価”, 電子情報通信学会技術研究報告, SS2002-42, Vol.102, No.617, pp.31-36, (2003).

[31] M. Pinzger, H. Gall: “Pattern-Supported Architecture Recovery”, 10th International Workshop on Program Comprehension (IWPC’02),pp53-63 (2002).

[32] I. T. Bowman, M. W. Godfrey , R. C. Holt: “Connecting Software Architecture Recovery Frameworks”, In Proceedings of the First International Symposium on Constructing Software Engineering Tools (CoSET’99), May 17-18, (1999).