社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE
SS2002-42(2003-1)

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

藤井 将人† 横森 励士† 山本 哲男†† 井上 克郎†††

† 大阪大学大学院基礎工学研究科 〒560–8531 大阪府豊中市待兼山町 1–3
†† 科学技術振興事業団 〒332-0012 埼玉県川口市本町 4–1–8
††† 大阪大学大学院情報科学研究科 〒560–8531 大阪府豊中市待兼山町 1–3
E-mail: †{m-fujii,yokomori}@ics.es.osaka-u.ac.jp, ††{t-yamamt,inoue}@ist.osaka-u.ac.jp

あらまし ソフトウェア開発の際，再利用性の高いソフトウェア部品を再利用すれば，生産性と品質を改善し，結果として開発コストを削減できる．我々は，"再利用性の高い部品は多くのソフトウェア中で利用されている"という，利用実績に基づいたソフトウェア部品評価手法 (Component Rank 法) を提案している．また，提案手法を基にオブジェクト指向言語 Java を対象とした Component Rank システム (CR システム) の実装を行っている．この，CR システムでは，クラスファイルから静的に利用関係を解析する手法を採用している．しかし，Java ではメソッド呼び出しやフィールド参照など，静的な解析では抽出できない実行時に決定する要素が多く存在する．また，一般的に静的利用関係解析には，構文，意味解析など複雑な解析を行う必要があり，実装が困難である．そこで本論文では，動的な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．提案手法を用いることにより，ソフトウェア実行時に実際に利用された関係だけを抽出するので，解析が静的に比べ容易であり，またソースコードファイルが存在しない部品についても評価が行える．提案手法を用いることにより，どのような部品グラフが生成され，評価値が得られるかを検証するため，既存の CR システムに動的利用関係解析を実装し，評価実験を行い，提案手法から得られる部品評価値の特性について考察を行う．

キーワード 再利用，ソフトウェア部品，Component Rank 法, 利用関係解析

Software Component Ranking using Dynamical Relation Analysis

Masato FUJII†, Reishi YOKOMORI†, Tetsuo YAMAMOTO††, and Katsuro INOUE†††

† Graduate School of Engineering Science, Osaka University
1–3, Machikaneyama-cho, Toyonaka-shi, Osaka 560–8531, Japan
†† Japan Science and Technology Corporation
4–1–8, Honmachi, Kawaguchi-shi, Saitama 332–8531, Japan
††† Graduate School of Information Science and Technology, Osaka University
1–3, Machikaneyama-cho, Toyonaka-shi, Osaka 560–8531, Japan
E-mail: †{m-fujii,yokomori}@ics.es.osaka-u.ac.jp, ††{t-yamamt,inoue}@ist.osaka-u.ac.jp

Abstract Reusing the software components with high reusability improves of development and quality of products. As a new reusability measurement method, we have proposed conmponet rank method based on the frequency of actual usage of components. Also we have implemented component rank system for object-oriented language Java. This CR-System have used static use-relation analysis for class files of Java. However, java contains a lot of dynamically determined issues. It is not easy to implement a complete static analysis, since it needs complicated analyses, such as lexical, syntactic aland semantic analyses. In this paper, We propose a new component rank technique using dynamic information. This technique is simpler one than the static analysis ,and it can analyze use relation without source files. We have added an implementationof this technique to the CR system, and have investigated the result obtained from this technique.

Key words Reuse, Software component, Component Rank, Use Relation

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，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が重要になっている．大規模ソフトウェア開発には，複数のプログラマがチームを組んで開発を行うが，開発期間を短縮するために新たな開発者を投入しても，かえってスケジュールが遅れることがあると言われている [3]．この原因の 1 つとして，開発者間でこれから開発しようとするソフトウェアに必要な部品およびライブラリに関する知識の共有が満足になされていないために，同種のプログラムが別々の場所で，独立して開発を行っている，ということが考えれらる．

知識を共有すれば，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利用する，いわゆるソフトウェアの再利用の効果が最大限に生かされ [5]，生産性と品質を改善し，結果として開発コストが削減される [4] [7] [9]．

そのため，知識の共有を目的としたソフトウェアの部品検索システムが重要となる．我々は，ソフトウェア部品の利用実績に基づいて各部品の重要度を測定し，順位付けし，評価する手法 (Component Rank 法, CR 法) を提案している [11]．この手法では，ソフトウェア部品が他のソフトウェア中にどの程度再利用されているかという実績を調べるによって，部品の重要度を定義している．

CR 法では，部品間の利用関係を抽出する手法について，静的な解析手法か動的な解析手法かを定めていない．しかし，開発者がどの部品を再利用したのかという情報は，ソースコードやドキュメント等の上に静的な情報として反映される．そのため，静的な解析手法を用いることで，開発者の視点で部品評価を行えると考えられる．我々では，提案手法を基に，オブジェクト指向言語 Java を対象とした，Component Rank システム (CR システム) の実装を行っており [12]，クラス (.class) ファイルから静的に利用関係を抽出する手法を実装している．

しかし，オブジェクト指向言語の動的束縛や例外処理のように，実行をして初めて動作が決定される要素をもつソフトウェアは数多く存在しており，これらのソフトウェアでは，どの部品が実際によく利用されているのかという情報は，静的な解析からは判定できない．また，一般的に静的な解析には，構文，意味解析等複雑な解析が必要となり，実装が困難であることや，ソースコードファイルが存在しない部品については評価が行えないといった，問題点も存在する．

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評価手法を提案する．提案手法では，ソフトウェア実行時に利用される部品を追跡することにより，利用関係が抽出可能であるため，静的解析よりも容易に実装が可能であり，ソースコードファイルが存在しないような部品についても評価が可能である．また，提案手法により得られる部品グラフ，評価値の結果が，静的解析と同等の結果が得られるのかどうかを検証するため，提案手法を CR システムへ実装し，java プログラムに対し適用実験を行い，考察を行う．

以降，2. 節では，文献 [11] で提案した CR 法の定義と部品評価値計算方法について述べる．3. 節では，CR システムと，本研究の提案手法である動的な利用解析手法について述べる．4. 節では，Java プログラムへの適用評価実験ならびその評価・考察について述べる．5. 節では，関連研究である SPARS システムについて簡単に述べる．最後に 6. 節では，まとめと今後の課題について述べる．

2. Component Rank 法

ここでは，部品の概念をモデル化し，その上で利用実績に基づいた部品重要度の評価手法 (Component Rank 法, CR 法) について説明する．

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 [1], [8]．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．

これら部品間には互いに利用する，利用されるという利用関係が存在する．例えば，ソースコードファイルでの，メソッド呼び出しや継承関係，ドキュメントでのリンクや参考文献，といったものが利用関係にあたる．CR 法では，各部品を頂点，部品間の利用関係を利用する側からされる側への有向辺として，グラフ上に表現する．以下，この部品間の利用関係を表現したグラフを部品グラフ *(Comopnent 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．

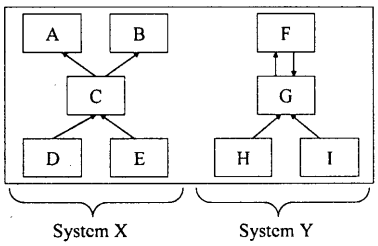


図 1　部品グラフの例

2.2 重みの定義

CR 法では，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与え，対応する頂点の重みをもとに各部品の評価値を求める．最初に，頂点の重みの総和を定義する．部品グラフ $G = (V, E)$ 上の各頂点 $v \in V$ は $0 \leqq w(v) \leqq 1$ 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）

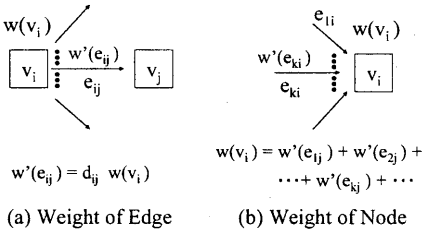
$$\sum_{v \in G} w(v) \quad = 1 \tag{1}$$

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.



(a) Weight of Edge (b) Weight of Node

図 2 重みの定義

d_{ij} は配分率とよび, $0 \leqq d_{ij} \leqq 1$ かつ $\sum_i d_{ij} = 1$ を満たす値とする. 頂点 v_i から v_j へ利用関係が存在しない場合, ここでは $d_{ij} = 0$ とする.

最後に, 頂点と辺の重みの関係を定義する. $\mathrm{IN}(v_i)$ を v_i を終点とす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.

2.3 重みの計算

2.2 節で定義した, 式 (3) に式 (2) を代入することで, 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と,

[定義 4] (ベクトル W, 行列 D)

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

n 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. 式 (5) を満たす解 W は, D^t に関する定常分布となるため, D^t における絶対値最大の固有値 1 の固有ベクトルを計算する事により, 求めることができる.

図 3 は, 与えられたグラフにおいて各頂点の重みを計算した結果である. v_1 は 2 つの有向辺の始点で, v_1 の重み 0.4 は二つの辺に 0.2 ずつ等分されている (つまり, $d_{12} = d_{13} = 0.5$). また, v_3 は 2 つの有向辺の終点で, それぞれの辺が 0.2 の重みを持つため, v_3 の重みは 0.4 であることがわかる.

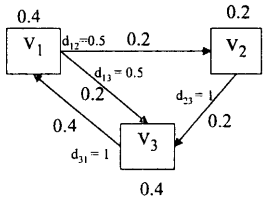


図 3 重みの計算例

2.4 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 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, グラフが図 4(a) のように強連結でない場合, 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.

そこで, 全ての頂点を図 4 (b) 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する. 配分率を次のように定義する.

[定義 5] (配分率) 頂点 v_i を始点とする辺への配分率 $d'(e_{ij})$ を次のように定義する.

$$d'_{ij} = \begin{cases} p \times d_{ij} + (1-p)/n & \text{if } v_i \text{ からの辺が存在} \\ 1/n & \text{if } v_i \text{ からの辺がない} \end{cases}$$

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.

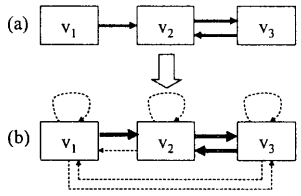


図 4 評価値計算に関する補正 (擬似辺の追加)

3. Component Rank システム

前節まで説明した CR 法に基づいて, Java プログラムを対象とした部品評価システム「Component Rank システム (以降, CR システム)」を構築行った. まず 3.1 節で Java への適用する際の, モデルと概念との対応についてのべ, その後 3.2 節で利用関係解析手法についての説明を行う. また, 3.3 節では, 実装した CR システムの構成について説明を行う.

3.1 Java への適用

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: 抽出する部品間の利用関係は，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．これらの抽出方法については，次節で説明する．

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．

3.2 利用関係解析

2.1 節で説明したように，ソフトウェア部品の間には，お互いに利用する，利用されるといる利用関係が存在する．2. 節で説明した，CR 法で部品評価値を求めるためにはこ利用関係の抽出が不可欠である．このソフトウェア間の利用関係を抽出するための解析手法を，利用関係解析と呼ぶ．

以降，3.2.1 節で静的利用関係解析，3.2.2 節で動的利用関係解析について説明を行う．

3.2.1 静的利用関係解析

静的利用関係解析は，対象ソフトウェアを動作させずに利用関係を抽出する解析手法のことである．静的利用関係解析は，実行を伴わない解析手法であるため，一般的に動的利用関係解析より解析コストは小さい．また，開発者がどの部品を再利用したのかという情報は，ソースコードやドキュメント等の上に静的な情報として反映される．そのため，静的利用関係解析では，開発者の視点からの部品評価が行える．

Java プログラムでは，ソースコード (.java) ファイルやクラス (.class) ファイルから，静的利用関係解析が行える．CR システムでは，クラスファイルを解析する手法を利用している．その理由として，1) ソースコードファイルから抽出できる情報とほとんど同じである，2) ソースコードファイルが存在しないライブラリ等の部品評価が行える，3)javac コンパイラですでに構文，意味解析が行われているため，解析が容易である，が挙げられる．

しかし，一般的にソフトウェア部品間の利用関係を抽出するためには，ソースコードファイルの構文，意味解析を行う必要がある．構文，意味解析はコンパイラで用いられている解析手法であり，その実現は複雑は容易でない．また，ソースコードファイルが存在しない部品については評価が行えない，といった問題点も存在する．

3.2.2 動的利用関係解析

動的利用関係解析とは，対象ソフトウェアを動作させて得られる情報から利用関係を抽出する解析手法のことである．動的利用関係解析は，実行を伴う解析手法であるため，利用される部品が一意に決定される．静的利用関係解析では，利用する可能性がある部品は全て抽出するため，一般的に次の式が成り立つ．

$$\text{動的な利用関係の集合} \subseteqq \text{静的な利用関係の集合} \qquad (6)$$

つまり，動的利用関係解析で得られる部品グラフは，静的利用関係解析で得られる部品グラフの部分グラフになる．

動的利用関係解析では，実行ファイルが存在していれば，ソースコードファイルが存在しなくても部品の評価を行うことが可能である．しかし，ソフトウェア実行し，その履歴を解析しなければならないため，解析コストは増大する．また，実行できないソフトウェアについては解析が行えない問題点もある．

Java では，Java Virtual Machine Profiling Interface(JVMPI) と呼ばれる，プロファイラ用のインタフェースが用意されており，これを用いることにより，動的に利用関係を抽出することが可能である．実行時に，メソッド呼び出し元クラスと呼び出し先クラスを対として保存し，プロファイル (.porf ファイル) を出力する．図 5 は，実行時にプロファイルが生成される様子を表した図である．出力されたプロファイルを解析することで，動的な利用関係を抽出できる．

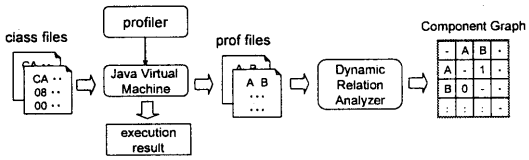


図 5 プロファイルの生成

JVMPI により抽出可能な利用関係は，メソッド呼び出しだけである．クラス継承については直接抽出できないが，メソッド呼び出しから間接的に抽出可能である．クラス継承の場合，子クラスのコンストラクタが実行される際，JavaVM メソッドである ⟨init⟩ メソッドが呼び出される．この時，同時にその親クラスのコンストラクタも実行され，親クラスの ⟨init⟩ メソッドが実行される．このように，子クラスの ⟨init⟩ メソッドから親クラスの ⟨init⟩ メソッドへ利用関係が存在し，クラス継承の利用関係が抽出できる．

ただし，インタフェース，抽象クラス実装関係については，実行時に呼び出しが行われないため，抽出ができない．そのため，動的利用関係解析で得られる利用関係は，クラスの継承，メソッドの呼びだし，フィールド参照のみになる．

3.3 CR システムの構成

図 6 に今回実装した CR システムの構成図を示す．解析の手順は以下の通りである．

(1) クラスファイルは静的利用関係解析を，プロファイルは動的利用関係解析を行い，部品間の利用関係を抽出する．

(2) 部品間の利用関係を元に，部品グラフを作成する．

(3) 部品グラフから行列を構築し，行列の固有値を計算を元に，各頂点の重みを決定する．

(4) 部品グラフの頂点の重みから，部品の評価値を求め，順位付けし出力する．

4. 評価実験

3. 節で説明した静的利用関係解析，動的利用関係解析を CR システムに実装し，実際のソフトウェア部品に対して評価実験を行った．以降，実験結果とその考察を行う．

4.1 実験結果

Java コンパイラ (javac) プログラムに対して，評価実験を行った結果について記述を行う．

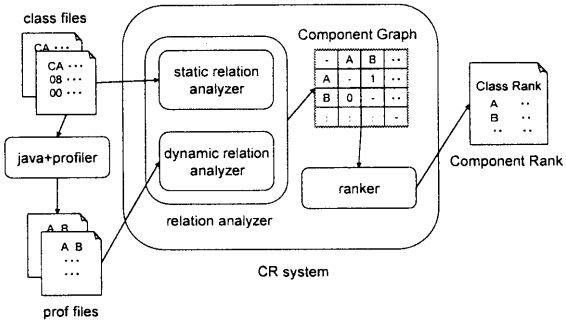


図 6　CR システムの構成

javac プログラムは，159 クラスからなる，java コンパイラプログラムである．動的利用関係解析のために必要となる，このプログラムの実行には，次の 3 種類の方法 (引数) で実行した．

(1) 引数を与えない．

(2) 正常にコンパイルが完了するプログラムを引数として与える．

(3) 構文エラーを出力して終了するプログラムを引数として与える．

この時，(1) ～ (3) の各実行方法にあう，テストデータをいくつか与えたところ，実行で利用されるクラス数は，それぞれ次のようになった．

(1) javac:5 クラス+JDK 標準ライブラリ:137 クラス

(2) javac:53 クラス+JDK 標準ライブラリ:138 クラス

(3) javac:40 クラス+JDK 標準ライブラリ:138 クラス

(4) javac:47 クラス+JDK 標準ライブラリ:138 クラス

ここで (4) は，3 種類の実行時に利用されたクラスについて，和集合を求めたものである，

次に静的利用関係解析で利用関係が存在したクラスファイルを次に示す．

(5) javac:159 クラス+JDK 標準ライブラリ:505 クラス

以上の実行方法と利用クラス数の関係を，表 1 にまとめる．

表 1　実行方法と利用クラス

解析手法	動的				静的
実行方法	(1)	(2)	(3)	(4)	(5)
利用 javac クラス数	5	46	40	47	159
利用標準ライブラリクラス数	137	138	138	138	505

実行時に利用される javac クラスが，159 クラスに対して極端に少ないのは，インターフェース実装関係が抽出できていないこと，また 159 クラス中 96 クラスが内部クラスであり，そのほとんどが実行時には利用されていなかったことが，その原因となっていた．

(1) から (5) の部品集合に対し，それぞれ CR システムを用い部品評価値を求めた．(4) の結果を動的利用関係解析の結果，(5) の結果を静的利用関係解析の結果として，標準ライブラリを除いた，javac プログラムに含まれる上位 10 クラスの評価値を，それぞれ表 2，3 に示す．なお，評価値については，定義 (1) より 0 以上 1 以下の小数になるため，わかりやすく表示するように 1 億倍した値を出力している．

表 2　javac への実験結果 ((4) の動的利用関係解析)

Rank	Class Name	Weight
9	com.sun.tools.javac.v8.util.List	997645
11	com.sun.tools.javac.v8.util.Name	957841
30	com.sun.tools.javac.v8.comp.Env	563968
31	com.sun.tools.javac.v8.util.StaticName	544410
39	com.sun.tools.javac.v8.code.Symbol	495454
42	com.sun.tools.javac.v8.code.Type	492252
44	com.sun.tools.javac.v8.code.Scope	488402
46	com.sun.tools.javac.v8.util.ListBuffer	477878
53	com.sun.tools.javac.v8.util.Hashtable	447058
59	com.sun.tools.javac.v8.tree.Tree	429795

表 3　javac への実験結果 ((5) の静的利用関係解析)

Rank	Class Name	Weight
6	com.sun.tools.javac.v8.util.List	3017972
9	com.sun.tools.javac.v8.tree.Tree	2325401
14	com.sun.tools.javac.v8.util.Name	1158991
40	com.sun.tools.javac.v8.util.Convert	336325
43	com.sun.tools.javac.v8.code.Symbol	300807
52	com.sun.tools.javac.v8.util.Log	250902
53	com.sun.tools.javac.v8.util.ListBuffer	237411
54	com.sun.tools.javac.v8.code.Scope	227312
66	com.sun.tools.javac.v8.util.Hashtable	176201
68	com.sun.tools.javac.v8.util.StaticName	169437

4.2　順位相関

4.1 節の (1) から (4) の動的利用関係解析の部品評価値と，(5) の静的利用関係解析の部品評価値を元に，動的利用関係解析と静的利用関係解析との順位相関を求める．ただし，3.2.2 節で述べたように

$$\text{動的な利用関係の集合} \subseteqq \text{静的な利用関係の集合} \qquad (7)$$

が成り立っているため，動的に利用されるクラス数と，静的に利用されるクラス数が異なる．よって，動的利用関係解析の上位 30 位までのクラスを基準として，spearman の順位相関係数を求めたところ，表 4 になった．

表 4　順位相関係数

(動的, 静的)	(1,5)	(2,5)	(3,5)	(4,5)
順位相関 (javac クラスのみ)	0.3070	0.6923	0.6309	0.8074
順位相関 (全クラス)	0.5354	0.6289	0.6133	0.6409
動的に利用する javac クラス数	5	46	40	47

4.3　考　　察

実験結果より，静的利用関係解析による部品評価では，動的に利用されているクラスの方よりも，動的には利用されていないクラスの方が，評価値が高い部品が存在することがわかった．例えば，静的利用関係解析では，プログラム実行の元となる「com.sun.tools.javac.Main」(以下，Main クラス) の評価値が最下位であるのに対し，「com.sun.tools.javac.v8.util.Abort」(以下，Abort クラス) の評価値は Main クラスより高い値になっていた．これは，Main クラスは，様々な他のクラスを利用しているが，逆に他のクラスから利用されることがないためである．Abort クラスは，利用されるクラスが存在するため，そのクラスからの辺の重みが得られ，結果 Main クラスより評

価値が高くなる．

しかし，実際に 4.1 節の (1) ～ (3) で実行した場合に，Abort クラスは利用されない．動的利用関係解析では，Abort クラスの評価が行われないため評価値を最下位と見なせば，Main クラスと同じ評価値となる．このことより，動的利用関係解析はより利用実績に反映した部品評価が行えると言える．

しかし，動的に抽出される利用関係は，当然実行方法によって異なる．表 4 より，実行方法によって部品の評価値が変わってくることがわかる．例えば，例外処理が発生するような実行であるなら，例外処理を行う部品の評価値が高くなる．CR 法の応用として，我々は部品検索を考えているが，本提案手法を用いることにより，「ある入力データの基で」機能を実現している部品，という付加情報を与えた検索が行えるのではないかと考えれれる．

また，現実のソフトウェアにおいては，よく実行される機能と，逆にあまり実行されない機能が存在する．そこで，よく実行される機能を実現している部品は，あまり実行されない機能を実現している部品と比べて，ソフトウェア部品として重要であると考えることもできる．そこで，実行回数によって，辺の重み付けを変更することで，現実によく利用される部品の評価値を上げることができ，静的な解析による CR モデルよりも利用実績をより忠実に捉えた新たなモデルが作れると思われる．

5. SPARS-J システム

我々は，は CR システムを用いたソフトウェア部品検索エンジンとして，現在 SPARS-J (*Software Product Archiving, analyzing, and Retrieving System*) の開発を行っている図 7 は SPARS-J の構成図である．

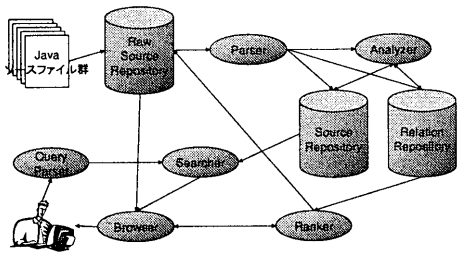


図 7 SPARS-J の構成

SPARS-J は収集してきた Java のソースプログラム，またはクラスファイルに対して解析を行い，リポジトリに保存し，CR システムを用いて各部品を順位付けする．部品を検索したい利用者はブラウザを通じて必要な部品に関する情報を検索キーとして入力する．部品検索部は，検索キーを解析しリポジトリ内を検索し．検索にヒットした部品を CR システムによる順位を元に並び替えて出力する．これにより利用者はよく利用される利用実績の高い部品を容易に取得する事ができる．

6. ま と め

本論文では，動的な情報を利用したソフトウェア部品評価手法を提案し，CR システムへの実装を行った．そして，静的利用関係解析による部品評価値と比較を行い，より利用実績を反映した部品評価が行えることを確認した．また，実行方法によって部品評価値が異なることから，特定の入力データの基で，特定の機能を実現するために重要な部品かどうかを判定するための評価値として利用できるのではないかと考えられる．

さらに，提案手法を用いることにより，1) 実行時エラーを含むような部品については評価を行わない，2) 特定の機能に着目した部品の評価が行える，といった利点も存在すると考えられる．特に，2) についてはシステムの振る舞いを判定できるという点で，Architecture Recovery への応用が行えると考えている．

また，他の今後の課題として，さらに多くのプログラムに対しての評価実験，継承，実装，メソッド呼び出し等の利用関係毎の重み付け，細粒度な部品単位の評価なども挙げられる．

謝辞 本研究は，科学技術振興事業団計算科学技術活用型特定研究開発推進事業 (ACT-JST) の支援を受けている．

文 献

[1] 青山, 中所, 向山: コンポーネントウェア, 共立出版, (1998).
[2] 馬場: "Google の秘密 - PageRank 徹底解説", http://www.kusastro.kyoto-u.ac.jp/baba/wais/pagerank.htm
[3] F. P. Brooks, Jr.: "The Mythical Man-Month: Essays on Software Engineering", Addison-Wesley, Reading, MA, 1975.
[4] 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).
[5] C. Braun: Reuse, in John J. Marciniak, editor, *Encyclopedia of Software Engineering*, Vol. 2, John Wiley & Sons, pp. 1055-1069 (1994).
[6] 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).
[7] S. Isoda: "Experience report on a software reuse project: Its structure, activities, and statistical results", *Proc. of ICSE14*, pp.320-326 (1992).
[8] I. Jacobson, M. Griss and P. Jonsson: Software Reuse, *Addison Wesley*, (1997).
[9] B. Keepence and M. Mannion: "Using patterns to model variability in product families", *IEEE Software*, Vol. 16, No. 4, pp. 102-108 (1999).
[10] 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
[11] 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).
[12] 横森, 藤原, 山本, 松下, 楠本, 井上: "利用実績に基づくソフトウェア部品重要度評価システム", Technical Report of SE Lab, Dept. of Computer Science, Osaka University, SEL-Nov-21-2002, Nov. (2002).