有向置換性類似度に基づくコンポーネント検索方式の実現と評価

鷲崎弘宜† 深澤良彰†

コンポーネント指向ソフトウェア開発において，要求に合致するコンポーネントの検索を適切に支援する機構は不可欠である．従来の検索手法は，コンポーネントの単一の特徴面を考慮するため特徴全体の要求に対する適合性を判断できず，また，コンポーネント本体以外の情報を必要とする等準備コストが大きい．本論文では，ソースコードを参照せずにコンポーネントから自動抽出可能な情報を基に，コンポーネント間の構造面・振舞面・粒度面について差異を表し，重視する特徴面を変更可能な有向置換性類似度を提案する．本論文では，検索要求をプロトタイプとして与え，プロトタイプとの有向置換性類似度に基づきリポジトリ中のコンポーネント群を順位付けする検索方式を実現した結果について，その有用性を示す．

Component Retrieval Method Based on Directed Replaceability Similarity and Its Evaluation

Hironori Washizaki† and Yoshiaki Fukazawa†

In component-based software development, the mechanism by which the software components which best satisfy a user's query are retrieved is indispensable. However, conventional retrieval methods cannot evaluate the total characteristics of a component, because they consider a single aspect of the component or require additional description. In this paper, we propose a new similarity metric "directed replaceability similarity" (DRS), which represents how different two components are in detail from the viewpoint of structure, behavior, and granularity. We have developed a retrieval system which uses DRS as a measure of the difference between a user's query prototype component and the components stored in a repository. In this paper, we outline the concept of DRS and the usefulness of our retrieval system.

1. はじめに

ソフトウェア部品の組合せ開発を中心とするコンポーネント技術が，開発期間の短縮とコストの削減を目的として注目されている．コンポーネントはある機能を提供する，独立して交換可能なソフトウェア構成単位である．狭義には，内部構造が公開されず，利用時のインタフェースが公開された，オブジェクトコード形式で配布されるソフトウェアコンポーネントを指す．本論文ではこの狭義のソフトウェアコンポーネントを扱う．コンポーネントの実装はオブジェクト指向言語で行うことが自然であり[1]，現在普及している各種コンポーネントシステム（JavaBeans，EJB 等）はオブジェクト指向をその基礎とするため，本論文では実装をオブジェクト指向言語に設定する．

コンポーネント技術は，コンポーネントの広範な再利用により要求を満すソフトウェアを迅速に開発可能とする一方で，要求を満すコンポーネントを適切に検索するための手法が未整備である[2]．コンポーネントは開発者の所属組織内に限らず，インターネットを含むネットワーク上で広域に配布・流通される[3]．このとき，コンポーネントの流通を円滑に行う機構として，コンポーネントリポジトリと，リポジトリから要求に適合するコンポーネントを検索する検索機構が必要となる．

広義のコンポーネント（ソフトウェア部品全般）が持つ特徴面として以下があげられる[4]．

- 構造： 内部関連要素・公開機能要素
- 振舞い： 機能的な振舞い
- 粒度： 提供する機能の概念的大きさ
- カプセル化： 機能・性質の隠蔽度合い
- 利用段階： 開発プロセス中で想定される利用段階
- ソースコード利用可能性： ソースコードの開示性

本論文では，オブジェクトコード形式で配布され実装段階で用いるコンポーネントを対象とするため，利

† 早稲田大学理工学部
School of Science and Engineering, Waseda University

用段階・ソースコード利用可能性の違いを考慮しない．また，検索結果からコンポーネントを取得後に，ソースコードが非開示なままカプセル化を検証することが可能なため[5]，本論文ではカプセル化を考慮しない．一方，構造・振舞い・粒度は，ソースコードを参照せず外部に公開される範囲で調査可能であり，対象コンポーネントの再利用可能性を判断するための情報となる．そこで本論文では，コンポーネント間の差異を表す特徴面として，構造・振舞い・粒度を取り上げる．

2. コンポーネント検索

2.1 従来の検索手法

コンポーネント技術の登場以前から，ソフトウェア部品の再利用促進のため，部品検索システムの必要性が強く指摘されてきている．以下に，ソフトウェア部品一般について従来提案される検索手法を示す．

- 自動抽出手法

コンポーネント本体から検索に必要な構造・機能情報を自動的に抽出し検索を行う手法[6),7)]：要求はインタフェース名等のキーワードで与えられる．この手法は，特にコンポーネントのソースコードを参照できないときに，検索に十分な情報を抽出できない問題が指摘される[8)]．

- 形式的仕様記述手法（カタログ記述手法）

コンポーネント本体とともに，機能およびインタフェースに関する形式的な仕様をカタログ情報としてリポジトリに保存し，仕様の記述に基づいて検索する手法[2)]：要求はカテゴリ名等のカタログの仕様に基づくキーワードで与えられる．さらに仕様を述語論理に基づき記述することで，検索時における要求を満たすか検証し，要求に対する健全性を保証することもできる[9)]．しかし，追加的な仕様記述を準備する必要があり準備コストが大きい．

- 類似度手法

ソフトウェア部品間の機能的役割の類似性を用いて検索する手法：オブジェクト指向言語のクラス継承関係を用いる手法[10)]や，クラス・メソッド等の要素名の類似性や参照関係を用いる手法[8)]がある．要求はソフトウェア部品のプロトタイプとして与えられる．

- 型情報手法

ソフトウェア部品本体および部品が持つ関数の型情報を基に検索を行う手法[11)]：要求は目的とする機能を実現すると予想する関数等の型情報として与える．構造面のみを考慮し，検索結果が振舞面について要求に合致する保証がない．また，“厳密な適合”・“ゆるい適合”のように，適合性により分けられた検索結果が得られるが，各適合集合中での詳細な順位付けは得られない．

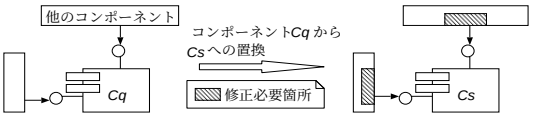

図 1 コンポーネントの置換と有向置換性類似度
Fig. 1 Component replacement and DRS.

2.2 コンポーネント検索の課題

従来提案される各検索手法は，検索にあたり考慮する特徴面が固定され，コンポーネント全体の特徴として要求に合致するかを判断できない[2)]．各手法を単独で使用せず併用することで複数の特徴面を考慮可能となるが，最適な組合せについての提案はなく，容易にコンポーネント全体の特徴を考慮した検索を行うことができない．したがって，単独で複数の特徴面を同時に考慮可能なコンポーネント検索機構が求められる．

一方，コンポーネントはネットワーク上で追加的な仕様記述をともなわずに流通している現状があり，すべてのコンポーネントにカタログ記述等を求めるのは非現実的である[8)]．したがって，コンポーネント本体以外の情報を必要とせずに登録・検索できることが望ましい．

3. 有向置換性類似度

コンポーネント間の差違を表す指標として，有向置換性類似度（Directed Replaceability Similarity: DRS）を提案する．有向置換性類似度 $DRS(c_q, c_s)$ は，コンポーネント c_q が使用されている状況で，要求される機能を同一に保持したまま，コンポーネントを c_q から c_s に置換する際に必要な周囲の修正コストの大きさを表す．本論文では，コンポーネント c_q の持つインタフェースが一様に使用されている文脈を想定する．また，コンポーネントの実装をオブジェクト指向言語で行うため，インタフェースはオブジェクト指向クラスのメソッドとして実現される．**図 1** に置換時の状況を示す．

1 章より，我々はコンポーネント間の差違を表す特徴面として，構造・振舞い・粒度の 3 種を取り上げる．3 種の特徴面について，構造類似度 DRS_S，振舞類似度 DRS_B，粒度類似度 DRS_G を準備し，有向置換性類似度をこれら 3 種の類似度の結合結果として定義する．3 種の類似度は 0 から 1 の間で正規化され，動的重み付け線形結合モデル[12)]に基づき結合される．得られる最終的な有向置換性類似度は 0 から 1 の間で正規化され，結合時の重み付けを変更することで，利用

者が重視する特徴面を変更可能である．$DRS(c_q, c_s)$ の定義を以下に示す．

$$DRS(c_q, c_s) ::= w_1 DRS_S(c_q, c_s) + w_2 DRS_B(c_q, c_s) + w_3 DRS_G(c_q, c_s)$$
$$\sum_{i=1}^{3} w_i = 1,\ w_i \geq 0$$

構造類似度・振舞類似度・粒度類似度の定義にあたり，各定義中で共通に用いる類似度関数 $d(x, y, z)$ を以下に定義する．

有向置換性類似度の測定時に考慮するコンポーネントの何らかの属性の型・種別をタイプと呼ぶ．たとえば，コンポーネントの粒度的な特徴を考慮するとき，コンポーネントが持つ静的なサイズという属性は考慮の対象となる．そのサイズの型は自然数タイプである．自然数タイプの実体である自然数値の集合 $\mathcal{N}$ 上の 2 項関係 $\leq$ を，自然数値 a と b が等しいか，a が b よりも小さいなら，$a \leq b$ と定義するとき，自然数タイプの実体の集合 $\mathcal{N}$ について半順序集合 $\langle \mathcal{N}, \leq \rangle$ が得られる．得られた半順序集合は，自然数値 0 を先頭とした自然数値の連なりであり，自然数値 0 の位置を 1 と呼ぶならば，他の自然数値の位置はその自然数値に 1 を足した値とすることができる（例：自然数値 4 の位置は 5）．

上記の正確な定義は以下のようになる．

$d(x, y, z)$ は，同一のタイプの実体 X，Y の位置 x，y と，実体 X，Y に共通で位置が最大な実体 Z の位置 z を用いて，実体 X からみて実体 Y がどの程度離れているかを表す．タイプの実体の位置の定義を以下に示す．

タイプ t の実体の集合 τ 上のある 2 項関係 $\preceq$ によって，半順序集合 $\langle \tau, \preceq \rangle$ を形成するものとする．2 つの異なる実体 X，Y について $Y \preceq X$ が成り立つとき，Y は X よりも小さいと呼び，X は Y よりも大きいと呼ぶ．タイプ t のすべての実体 X について $Y \preceq X$ を満たす実体 Y を，タイプ t についての実体 root_t と呼ぶ．ここで，実体 X について $Y \preceq X$ を満たすすべての実体 Y の集合中で，関係 $\preceq$ について実体 X との間に他の実体が存在しない実体 X の直前の実体 Y' を求める操作を f と記述する．直前の実体 Y' は以下を満たす．

$$X, Y' \in \tau \wedge Y' \preceq X \wedge Y' \not\equiv X \wedge \neg(\exists Y'' : t \mid Y'' \in \tau : Y' \preceq Y'' \wedge Y'' \preceq X \wedge Y' \not\equiv Y'' \wedge Y'' \not\equiv X)$$

タイプ t の実体 X の位置 x とは，操作 f を実体 X に適用して直前の実体 Y' を得て，得られた直前の実体 Y' に再び操作 f を適用する形でこの操作 f を繰り返し，最終的に実体 root_t にたどり着くまでにかかる操作の回数に 1 を足した値とする．

同一タイプの 2 つの実体 X，Y に関して，両実体 X，Y と関係 $\preceq$ について小さく，かつ，最も位置の大きい実体を，X，Y についての共通実体とする．実体 X，Y の共通実体 Z は以下に定義される．定義中で M は，X，Y に関して関係 $\preceq$ について小さい実体の集合を表す．

$$M = \{Z' : t \mid Z' \in \tau \wedge Z' \preceq X \wedge Z' \preceq Y : Z'\}$$
$$Z \in M \wedge (\forall Z'' : t \mid Z'' \in M : Z'' \preceq Z)$$

類似度関数 d を定義する方針として以下をあげる．

（方針 1）類似度関数の値を絶対的な指標として用いることを可能とするために，実体 X，Y 間の類似度はつねに 0 から 1 の間とする．

（方針 2）実体 X，Y が同一のときの類似度を 0 とする．

（方針 3）類似度関数を用いる有向置換性類似度 DRS において，置換する前後の対象の順序関係を値に反映するために，実体 X の位置が実体 Y の位置よりも小さいとき，X からみた Y は近く，Y からみた X は遠いものとして類似度に反映する．

（方針 4）共通実体との相対的な位置関係が同一な X，Y，Z と X'，Y'，Z' の 2 組があるとき，共通実体 Z（または Z'）の位置が大きいほど，X，Y 間（X'，Y' 間）の類似度は小さい．

（方針 1）より，$d(x, y, z)$ の得られる値はつねに 0 から 1 の間で正規化されている必要がある．（方針 4）より，x と y の相対的な位置関係が同一であっても，それぞれの位置が大きいほどに，$d(x, y, z)$ の得られる値は小さい必要がある．この 2 つの方針を満たす関数として $1/t^2$ の t に関する始端が 1 以上の区間積分があげられる．区間積分の区間として，本来の置換対象に対応する実体 Y の位置 y を終端とするとき，X からみた Y の相対的な位置関係が考慮された位置を始端とすることで，得られる値が X からみた Y の遠さを 0 から 1 の間で示すことになる．X からみた Y の相対的な位置関係が考慮された位置として，X と Y の共通実体 Z の位置 z を用いた類似度 $sim(x, y) = 2z/(x + y)$ を用いて，$sim(x, y)$ を y に掛け合わせた値 $y \cdot sim(x, y)$ を使用することにする．以上より，正整数 x，y，z について類似度関数 $d(x, y, z)$ を以下に定義する．

$$x, y, z \in (\text{整数の集合}),\ 0 < z \leq x, y$$
$$d(x, y, z) ::= \int_{1+y\frac{2z}{x+y}}^{1+y} \frac{1}{t^2} dt = \frac{y(x+y-2z)}{(y+1)(x+y+2yz)}$$

表 1 構造/振舞いの異なるインタフェース例
Table 1 Examples of methods with different structures/behaviors.

コンポーネント	属性変数宣言・初期化	インタフェース構造	インタフェース本体
C_1	int data = 0;	int calc1(int x)	{ data = x; return x; }
C_2	long data = 0;	short calc2(int x)	{ data = (long) x; return 0; }
C_3	long data = 0;	short calc3(long x)	{ data = (long) x; return x; }
C_4	int data = 0;	long calc4(short x)	{ return 0; }

類似度関数 $d(x, y, z)$ は，上述の方針に対応して以下の性質を持つ．以下において，a，b，c，e はそれぞれ正整数とする．

（性質 1）$c \leq a, b$
$\Rightarrow \lim_{b\to\infty} \int_1^{1+b} \frac{1}{t^2} dt = 1 \Rightarrow d(a, b, c) < 1$
（性質 2）$a = b = c \Rightarrow d(a, b, c) = d(b, a, c) = 0$

（性質 3）$c < a < b$
$\Rightarrow 0 < \frac{b(a+b-2c)}{(b+1)(a+b+2bc)} < \frac{a(a+b-2c)}{(a+1)(a+b+2ac)} < 1$
$\Rightarrow 0 < d(a, b, c) < d(b, a, c) < 1$
（性質 4）$c \leq a, b$
$\Rightarrow \frac{(b+e)(a+b-2c)}{(b+e+1)(a+b+2(b+e)(c+e)+2e)} < \frac{b(a+b-2c)}{(b+1)(a+b+2bc)}$
$\Rightarrow d(a+e, b+e, c+e) < d(a, b, c)$

この類似度関数 $d(x, y, z)$ を用いる有向置換性類似度は定義上，$c_q \not\equiv c_s$ のとき $DRS(c_q, c_s) = DRS(c_s, c_q)$ を満たさないが，$DRS(c_q, c_q) < DRS(c_q, c_s)$ を満たすため，習慣的に類似度と呼ぶことができる[13]．

3.1 構造類似度

コンポーネントの構造面を構成する属性として，コンポーネントの名前と，コンポーネントが持つインタフェースの構造（シグニチャ）があげられる．そこで，構造類似度 $DRS_S(c_q, c_s)$ を，コンポーネントの名前，および，コンポーネントインタフェースの構造（シグニチャ）の差異を反映した類似度として定義する．

たとえば，**表 1** に示すコンポーネント C_1〜C_4 があり，C_1 が持つインタフェース calc1 を使用している状況を考える．4 つのコンポーネントはそれぞれインタフェースを 1 つずつ持つ．calc1 を他のコンポーネントが持つインタフェースと置換するときに，引数のタイプがとりうる実体としての整数値の範囲が calc1 よりも大きいもしくは同一で，戻り値のタイプが取りうる整数値の範囲が calc1 よりも小さいもしくは同一のインタフェースを置換後のものとして用いるならば，calc1 を使用していた部分は構造面について修正される必要がない．各整数値タイプの値の範囲は $\{x : \text{short} \mid: x\} \subset \{x : \text{int} \mid: x\} \subset \{x : \text{long} \mid: x\}$ という包含関係にある．calc2 と calc3 はともに calc1 と比較して，引数のタイプの整数値の範囲が大きいもしくは同一で，戻り値のタイプの整数値の範囲が小さいため，calc1 を置換する際の修正コストが小さいと考えられる．ただし，calc2 は引数について calc1 とタイプが同一であるため，calc3 に対してより修正コストが小さいと考えられる．したがって，calc1 を置換する際に構造面について修正コストが低いと予想される順位は calc2 < calc3 < calc4 となる．コンポーネント間の構造面での差違は，置換前後のコンポーネントが持つインタフェース集合間のこのような構造的な差異の集まりと，コンポーネントが持つ名前の間の差違により求まる．

構造類似度 $DRS_S(c_q, c_s)$ を，コンポーネント名間の文字列類似度 dw とインタフェース構造集合間の実体集合類似度 dr を用いて，以下に定義する．

コンポーネントの構造面 C_S，
インタフェース構造 I_S
$C_S ::= \{name{:}\text{String}, interfaces : \{i_1{:}I_S, ..., i_n{:}I_S\}\}$
$c_q, c_s : C_S$
$c_q = \{name = n_q, interfaces = is_q\}$
$c_s = \{name = n_s, interfaces = is_s\}$
$DRS_S(c_q, c_s) ::= \frac{dw(n_q, n_s) + 2dr(is_q, is_s)}{3}$

ここで文字列類似度 dw と実体集合類似度 dr は以下のように定義される．まず，文字列 w_q，w_s 間の文字列類似度 $dw(w_q, w_s)$ は，両文字列の最長一致部分文字列 w_p と，類似度関数 d を用いて以下に定義する．

$w, w_q, w_s : \text{String}$
$|w| =$（文字列 w の長さ）
$$dw(w_q, w_s) ::= \begin{cases} d(|w_q|, |w_s|, |w_p|) & (w_p \text{があるとき}) \\ 1 & (w_p \text{が存在しないとき}) \end{cases}$$

実体集合類似度 dr を以下に定義する．実体集合とは，同一タイプの実体の集合を指す．タイプ x の 2 つの実体集合 R_q，R_s 間の類似度は，単純には両実体集合中の実体間のタイプ x に対応した類似度 dx の合計を平均することで求められる．以降において，この類似度 dx を実体集合類似度算出時の内部類似度と記述する．しかし，比較する両実体集合に含まれる

実体の数が異なる場合は，実体数の違いを実体集合類似度に反映する必要がある．以下に，実体数を反映したタイプ x の実体集合 R_q，R_s 間の実体集合類似度 $dr(R_q, R_s)$ を定義する．

まず，R_q に含まれるすべての実体 q と，R_s に含まれるすべての実体 s のすべての組合せ (q, s) について，タイプ x に対応する類似度 $dx(q, s)$ を算出する．すべての組合せについて，$dx(q, s)$ の値が小さいものから順に組合せが持つ両実体が重複しないように組合せを確定して組合せ集合 S_f を得て，S_f に含まれる組合せの類似度 $dx(q, s)$ の合計を得る．その合計値を $f_1(R_q, R_s)$ と記述する．

$$f_1(R_q, R_s) ::= \sum_{(q,s)\in S_f} dx(q, s)$$

たとえば，$R_1 = \{q_1, q_2, q_3\}$，$R_2 = \{q_1\}$，$R_3 = \{s_1, s_2\}$ のときに，$dx(q_1, s_1) < dx(q_3, s_1) < dx(q_3, s_2) < dx(q_2, s_2) < dx(q_1, s_2) < dx(q_2, s_1)$ であるならば，R_1，R_3 について重複しない組合せ集合として $S_f = \{(q_1, s_1), (q_3, s_2)\}$ を得て，$f_1(R_1, R_3) = dx(q_1, s_1) + dx(q_3, s_2)$ となる．

続いて，R_q の実体数よりも R_s の実体数が少ない場合は，組合せ集合 S_f に含まれない R_q 中のすべての実体 q について，タイプ x に対応した実体 root_x との組合せ (q, root_x) を作り，そのすべての組合せの $dx(q, \mathrm{root}_x)$ の合計 $f_2(R_q, R_s)$ を得る．

$|R| ::=$（実体集合 R中の実体数）
$|R_q| > |R_s|$ のときに
$$f_2(R_q, R_s) ::= \sum_{q\in R_q - \{q' : x | q' \in S_f : q'\}} dx(q, \mathrm{root}_x)$$

たとえば，前述の R_1，R_3 を用いるとき，$f_1(R_1, R_3)$ を算出する際の組合せ集合 S_f に含まれない R_q 中の実体として q_2 を得るため，$f_2(R_1, R_3) = dx(q_2, \mathrm{root}_x)$ となる．

一方，R_q の実体数よりも R_s の実体数が大きい場合は，組合せ集合 S_f に含まれない R_s 中のすべての実体 s について，タイプ x に対応した実体 root_x との組合せ (root_x, s) を作り，そのすべての組合せの $dx(\mathrm{root}_x, s)$ の合計 $f_3(R_q, R_s)$ を得る．

$|R_q| < |R_s|$ のときに
$$f_3(R_q, R_s) ::= \sum_{s\in R_s - \{s' : x | s' \in S_f : s'\}} dx(\mathrm{root}_x, s)$$

たとえば，前述の R_2，R_3 を用いるとき，$f_1(R_2, R_3)$ を算出する際の組合せ集合 S_f に含まれない R_3 中の実体として s_2 を得るため，$f_2(R_2, R_3) = dx(\mathrm{root}_x, s_2)$ となる．

最後に，$f_1(R_q, R_s)$，$f_2(R_q, R_s)$，$f_3(R_q, R_s)$ を合計し，その合計値を $|R_q|$ と $|R_s|$ のうちで大きい方の値で割ることで，実体集合類似度 $dr(R_q, R_s)$ を得る．

$$max(|R_q|, |R_s|) ::= \begin{cases} |R_q| \ (|R_q| \geq |R_s| \text{ のとき}) \\ |R_s| \ (\text{それ以外}) \end{cases}$$
$$dr(R_q, R_s) ::= \frac{f_1(R_q,R_s)+f_2(R_q,R_s)+f_3(R_q,R_s)}{max(|R_q|,|R_s|)}$$

実体集合類似度 $dr(R_q, R_s)$ を算出する際の実体集合 R_q と R_s は，有向置換性類似度の定義に組み込む際に，置換前の対象と置換後の対象にそれぞれ対応する．ここで，置換前の対象の要素数が置換後の対象の要素数よりも小さい状況は，置換前の対象の要素数の方が大きい状況よりも好ましいと考えられる．置換後の対象の要素数の方が大きい場合は，置換後の対象の要素集合のうちで置換前の対象の要素に対応させないものを，隠蔽あるいは使用しない対処で済むためである．

したがって，同一のタイプ x についての実体集合 R_1 と，R_1 に任意の数だけ実体を加えた R_2 があるとき，$dr(R_1, R_2) \leq dr(R_2, R_1)$ が満たされる必要がある．以降において，この実体集合類似度が満たすべき条件を実体数考慮条件と記述する．実体集合類似度 $dr(R_1, R_2)$ を算出する際に用いる x の類似度 dx について，R_2 中で R_1 に対して加えられたすべての実体 q について $dx(q, \mathrm{root}_x) \geq dx(\mathrm{root}_x, q)$ が成り立つならば，実体数考慮条件は満たされる．

$$R_2 = R_1 \cup \{q_1 : x, ..., q_n : x\}$$
$$(\forall q : x \mid : q \in R_2 - R_1 \wedge dx(q, \mathrm{root}_x) \geq dx(\mathrm{root}_x, q))$$
$$\Rightarrow dr(R_1, R_2) \leq dr(R_2, R_1)$$

構造類似度算出時のインタフェース構造 I_S は，インタフェース名と，戻り値と引数の型に関する関数タイプから構成される．インタフェース間の構造面類似度 $dis(i_q, i_s)$ を，インタフェース名間の文字列類似度 dw と関数類似度 df を用いて，以下に定義する．

$$I_S ::= \{name : \mathrm{String}, signature : F\} \quad i_q, i_s : I_S$$
$$i_q = \{name = name_q, signature = sig_q\}$$
$$i_s = \{name = name_s, signature = sig_s\}$$
$$dis(i_q, i_s) ::= \frac{dw(name_q,name_s)+2df(sig_q,sig_s)}{3}$$

インタフェース構造集合間の実体集合類似度の算出時には，内部類似度 dx としてインタフェース構造類似度 dis と，インタフェース構造の実体 root_{is} として，$\mathrm{root}_{is} = \{name = \text{“ “}, signature = \{params = \{\} \rightarrow return = \mathrm{root}_T\}\}$ が用いられる．インタフェース構造類似度 dis は，インタフェース構造 I_S のすべての実体 i_s について，$dis(i_s, \mathrm{root}_{is}) \geq dis(\mathrm{root}_{is}, i_s)$

を満たすため，実体数考慮条件が満たされる．

関数タイプ間の関数類似度 $df(f_q, f_s)$ は，引数について実体集合類似度 dr を，戻り値について一般タイプ類似度 dt を用いる．すべての一般タイプを実体として持つパワータイプ T を用いて，$df(f_q, f_s)$ を以下に定義する．

関数タイプ F $f_q, f_s : F$

$F ::= \{params : \{t_1 : T, ..., t_n : T\} \rightarrow return : T\}$

$f_q = \{params = ps_q, return = r_q\}$

$f_s = \{params = ps_s, return = r_s\}$

$df(f_q, f_s) ::= \frac{dr(ps_s, ps_q) + dt(r_q, r_s)}{2}$

オブジェクト指向タイプシステムにおいて，関数タイプのサブタイプ推論規則は，引数について反変であり，戻り値について共変である[14]．サブタイプ推論規則について反変とは，複数のタイプから構成されるタイプ t のサブタイプ推論規則として $t <: t'$ があるときに，t を構成する t_a と t' を構成する t'_a について，$t <: t'$ とは方向が逆転しているサブタイプ推論規則 $t'_a <: t_a$ が求められる状況を指す．

たとえば，置換する前の関数タイプの実体 f_q として，$f_q : \{p_q\} \rightarrow r_q$ があり，関数タイプの実体 f_q を利用する側は，引数としてタイプ p_q の実体を代入しているものとする．このような状況において，関数タイプの実体 f_q を $f_s : \{p_s\} \rightarrow r_s$ なる実体 f_s で置換するときに，$p_q <: p_s$ である場合は，利用する側は引数として代入するタイプ p_q の実体についてタイプ p_s へのタイプの拡大変換を施すことで置換に対応する．逆に，$p_s <: p_q$ である場合は，利用する側は引数として代入するタイプ p_q の実体についてタイプ p_s へタイプの縮小変換を施すことで置換に対応する．タイプの拡大変換が一般に自動的に行われるのに対して，タイプの縮小変換は明示的に行う必要があり，また，タイプの縮小変換はタイプの実体が持つ情報が一部失われる可能性があるため，$p_q <: p_s$ である場合の修正コストは $p_s <: p_q$ である場合の修正コストに比較して小さいと考えられる．

本論文で提案する関数類似度 df は，前述の f_q に対して f_s を比較するときに，引数部分に関する類似度を実体集合類似度 $dr(\{p_s\}, \{p_q\})$ によって算出することで，この関数タイプに関する特徴を考慮している．

なお，提案する関数類似度 $df(f_q, f_s)$ は，置換する前の関数タイプの実体 f_q の引数に，利用する側がどのようなタイプの実体を代入しているかを考慮する必要がない．例として，前述の f_q から f_s に置換する際の利用する側の修正コストを考える．置換する前に

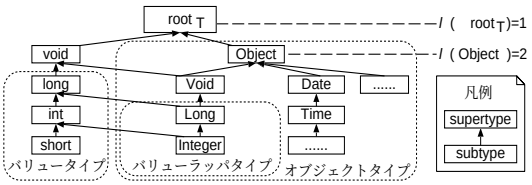

図 2 標準的な Is–a 階層グラフ
Fig. 2 Standard Is–a graph.

利用する側が，引数として p_q の実体を代入している状況 A と，p_q とは異なるタイプである p'_q の実体を代入している状況 B を考える．両状況からそれぞれ f_q の実体を f_s に置換するとき，状況 A において利用する側の必要な修正作業は，引数に代入する p_q の実体のタイプを p_s に変換することでなされる．したがって，関数類似度 $df(f_q, f_s)$ は実際の修正作業の大きさを表すことになる．

一方，状況 B において利用する側の必要な修正作業は，引数に代入する p'_q の実体のタイプを p_s に変換することでなされる．ここで，置換前において利用する側は，すでに引数に代入する p'_q の実体のタイプを引数のタイプである p_q に変換する処理を行っていることになる．したがって，利用する側の必要な実際の修正作業は，すでに行われている p'_q の実体のタイプを p_q に変換する処理の後に続けて，タイプが p_q に変換後の実体のタイプを p_s に変換する処理を追加することでなされる．よって，状況 B においても状況 A と同様に，関数類似度 $df(f_q, f_s)$ が実際の修正作業の大きさを表すことになる．

一般タイプとして，バリュータイプ（int 等），バリューラッパタイプ（Integer 等），オブジェクトタイプ（Object 等）があげられる．すべての一般タイプは，パワータイプ T の実体であるとする．パワータイプ T の実体としての一般タイプは，オブジェクト指向タイプシステムにおけるサブタイプ推論規則を用い，パワータイプ T についての実体 root_T を頂点とした Is-a 階層グラフを形成する．オブジェクトタイプについてはサブクラス関係をサブタイプ推論規則として用いる．バリューラッパタイプは，バリュータイプで表される値を持つので，サブクラス関係に加えて，持つバリュータイプ間のサブタイプ推論規則を用いる．Java 言語上の Is–a 階層グラフを**図 2** に示す．

サブタイプ関係は，式のタイプが t_q である文脈上で，つねにタイプが t_s である式が安全に利用可能なとき，t_s は t_q のサブタイプであると定義され $t_s <: t_q$ と表す．$t_s <: t_q$ が成り立つとき，t_q から t_s への置換に必要なコストは低く，t_s から t_q への置換は逆に大

きなコストがかかると予想される．そこで，一般タイプの実体間の比較関係としてサブタイプ関係を用いる．一般タイプ間の類似度 $dt(t_q, t_s)$ について，$t_s <: t_q$ のとき，$dt(t_q, t_s) < dt(t_s, t_q)$ が成り立つ必要がある．$dt(t_q, t_s)$ を，$t_q <: t_p \wedge t_s <: t_p$ なる Is-a 階層グラフ上で最深のタイプ t_p があるとき，類似度関数 d を用いて以下に定義する．定義中で M は，t_q，t_s の両タイプをサブタイプとするタイプの集合を表す．

$\mathrm{root}_T, t_q, t_s, t_p : T$
$l(t) ::=$（タイプ t の root_Tからの位置）
$l(\mathrm{root}_T) = 1$
$M = \{t : T \mid t_q <: t \wedge t_s <: t : t\}$
$t_p \in M \wedge (\forall t : T \mid t \in M : l(t) \leq l(t_p))$
$dt(t_q, t_s) ::= d(l(t_q), l(t_s), l(t_p))$

$1 \leq l(t_p) < l(t_q) < l(t_s)$
$\Rightarrow 0 < dt(t_q, t_s) < dt(t_s, t_q) < 1$

引数間の実体集合類似度の算出時には，内部類似度 dx としてこの一般タイプ類似度 dt が用いられる．一般タイプ類似度 dt は，パワータイプ T のすべての実体 t について，$dt(t, \mathrm{root}_T) \geq dt(\mathrm{root}_T, t)$ を満たすため，実体数考慮条件が満たされる．

3.2 振舞類似度

コンポーネントの振舞面を構成する属性として，コンポーネントが持つインタフェースの実行結果と，インタフェースを実行した際にコンポーネントの内部で変化する状態があげられる．そこで，振舞類似度 $DRS_B(c_q, c_s)$ を，コンポーネントインタフェースの実行結果，および，観測可能な状態のインタフェース実行前後における変化情報を考慮する類似度として定義する．

たとえば，表 1 に示すコンポーネント C_1〜C_4 があり，C_1 が持つインタフェース calc1 を使用している状況を考える．4 つのコンポーネントはそれぞれインタフェースと属性変数 data を 1 つずつ持つ．属性変数 data は，コンポーネントシステムが提供する内観機構（イントロスペクション機構）により値を観測可能な状態であり，宣言時に対応するタイプの初期値によって初期化されているものとする．引数に int の初期値以外の値を代入して calc1 を実行したときに，変化が観測される状態のタイプは int であり，戻り値は初期値以外の値が得られる．calc2 および calc3 を，引数に初期値以外の値を代入して実行したときに，変化が観測される状態のタイプは long である．long は int と同様に整数値タイプであるため，calc2 および calc3 は振舞いとして calc1 に類似していると考えられる．一方，calc4 は実行時に状態の変化をもたらさない．さらに，calc2 および calc4 は引数に代入する値の種類（初期値か，初期値以外の値か）にかかわらず，実行時に得られる戻り値の値はつねに初期値であり，初期値以外の値ではない．したがって，calc1 を置換する際に振舞面について修正コストが低いと予想される順位は calc3 < calc2 < calc4 となる．コンポーネント間の振舞面での差異は，置換前後のコンポーネントの持つインタフェース集合間の，このような観測可能な振舞い上の差異の集まりにより求まる．

$DRS_B(c_q, c_s)$ を，インタフェース振舞集合間の実体集合類似度 dr を用いて以下に定義する．

コンポーネントの振舞面 C_B，
インタフェース振舞い I_B
$C_B ::= \{interfaces : \{i_1 : I_B, ..., i_n : I_B\}\}$
$c_q, c_s : C_B$
$c_q = \{interfaces = is_q\}\ \ c_s = \{interfaces = is_s\}$
$DRS_B(c_q, c_s) ::= dr(is_q, is_s)$

振舞類似度算出時のインタフェース振舞い I_B は，引数のタイプに基づき，初期値を代入して実行した際の戻り値（rinit）と実行時に値の変化が観測された状態の集合（cinit），および，初期値以外の値を代入して実行した際の戻り値（rany）と変化した状態集合（cany）から構成される．戻り値は型付値タイプ Value を用い，変化した状態は変化状態タイプ Property を用いる．

タイプの初期値とは，対象コンポーネントシステムが用いる実装言語上でタイプの実体を指定なしに生成した際の値を指す．JavaBeans をコンポーネントシステムの対象とするとき実装言語は Java 言語であり，タイプがバリュータイプであれば，変数を宣言した状態の値が初期値となる（例：int 型なら 0，boolean 型なら false）．タイプがバリューラッパタイプ・オブジェクトタイプの場合は，引数なしの初期化子により生成した値が初期値となる．引数なしの初期化子を利用できない場合は，引数を持つ初期化子について引数のタイプに応じた初期値を再帰的に代入することで，最終的に目的とするタイプの初期値を得る．

初期値以外の値としては，統一的な比較のため，バリュータイプについて対象とする実装言語において用いる値を決定する（例：int 型なら 1，boolean 型なら true）．バリューラッパタイプ・オブジェクトタイプについては，初期化子について引数のタイプに応じた初期値以外の値を再帰的に代入することで値を得る．

インタフェース間の振舞面類似度 dib を，戻り値間の型付値類似度 dv と，実行前後の変化状態集合間の実体集合類似度 dr を用いて，以下に定義する．

$$
\begin{aligned}
&I_B ::= \{returns : \{rinit : \text{Value}, rany : \text{Value}\},\\
&\ changes : \{cinit : \{p_1 : \text{Property}, ..., p_m : \text{Property}\},\\
&\ cany : \{p_1 : \text{Property}, ..., p_n : \text{Property}\}\}\}\\
&i_q, i_s : I_B\\
&i_q = \{returns = \{rinit = ri_q, rany = ra_q\},\\
&\ changes = \{cinit = ci_q, cany = ca_q\}\}\\
&i_s = \{returns = \{rinit = ri_s, rany = ra_s\},\\
&\ changes = \{cinit = ci_s, cany = ca_s\}\}\\
&dib(i_q, i_s) ::= \\
&\frac{dv(ri_q, ri_s) + dv(ra_q, ra_s) + dr(ci_q, ci_s) + dr(ca_q, ca_s)}{4}
\end{aligned}
$$

インタフェース振舞集合間の実体集合類似度の算出時には，内部類似度 dx としてインタフェース振舞類似度 dib が用いられ，インタフェース振舞いの実体 root_{ib} として，$\text{root}_{ib} = \{returns = \{rinit = \{type = \text{root}_T, init = \text{false}\}, rany = \{type = \text{root}_T, init = \text{false}\}\}, changes = \{cinit = \{\}, cany = \{\}\}\}$ が用いられる．インタフェース振舞類似度 dib は，インタフェース振舞い I_B のすべての実体 i_b について，$dib(i_b, \text{root}_{ib}) \geq dib(\text{root}_{ib}, i_b)$ を満たすため，実体数考慮条件が満たされる．

型付値タイプ Value は，一般タイプと初期値かどうかを示す真理値から構成される．型付値タイプ間の型付値類似度 dv を，一般タイプ類似度 dt と真理値類似度 db を用いて，以下に定義する．

$$
\begin{aligned}
&\text{Value} ::= \{type : T, init : \text{boolean}\} \quad v_q, v_s : \text{Value}\\
&v_q = \{type = t_q, init = in_q\}\\
&v_s = \{type = t_s, init = in_s\}\\
&dv(v_q, v_s) ::= \frac{dt(t_q, t_s) + db(in_q, in_s)}{2}
\end{aligned}
$$

変化状態タイプ Property は，状態名と一般タイプ，および変化前後において初期値かどうかの真理値 bef，aft から構成される．変化状態タイプ間の変化状態類似度 dp を，状態名の文字列類似度 dw と，一般タイプ類似度 dt，および，変化前後の真理値類似度 db を用いて，以下に定義する．

$$
\begin{aligned}
&\text{Property} ::= \{name : \text{String}, type : T,\\
&\ bef : \text{boolean}, aft : \text{boolean}\}\\
&p_q, p_s : \text{Property}\\
&p_q = \{name = n_q, type = t_q, bef = b_q, aft = a_q\}\\
&p_s = \{name = n_s, type = t_s, bef = b_s, aft = a_s\}\\
&dp(p_q, p_s) ::= \frac{dw(n_q, n_s) + dt(t_q, t_s) + db(b_q, b_s) + db(a_q, a_s)}{4}
\end{aligned}
$$

変化状態集合間の実体集合類似度の算出時には，内部類似度 dx として変化状態類似度 dp が用いられ，変化状態タイプの実体 root_p として，$\text{root}_p = \{name = \text{“ “}, type = \text{root}_T, bef = \text{false}, aft = \text{false}\}$ が用いられる．変化状態類似度 dp は，変化状態タイプ Property のすべての実体 p について，$dp(p, \text{root}_p) \geq dp(\text{root}_p, p)$ を満たすため，実体数考慮条件が満たされる．

真理値間の真理値類似度 db を，類似度関数 d を用いて以下に定義する．

$$
\begin{aligned}
&g(x : \text{boolean}) ::= \begin{cases} 1\ (\text{false のとき}) \\ 2\ (\text{true}) \end{cases}\\
&b_q, b_s : \text{boolean}\\
&db(b_q, b_s) ::= d(g(b_q), g(b_s), g(b_q \wedge b_s))
\end{aligned}
$$

3.3 粒度類似度

コンポーネントの粒度面を構成する属性として，コンポーネントの外部からみた静的な大きさと，コンポーネントの内部の複雑度があげられる．本論文が対象とするコンポーネントはソースコードを得られないため，内部の複雑度を直接的に測定できない．そこで，コンポーネントが持つすべてのインタフェースを同一の実行環境で実際に起動した際の実行時間によって，内部の複雑度を間接的に測定する．粒度類似度 $DRS_G(c_q, c_s)$ を，コンポーネントのサイズ，およびインタフェースの実行時間を考慮する類似度として定義する．

たとえば，コンポーネント C_1（サイズ：10 kbyte，総実行時間：10 msec），C_2（15 kbyte，20 msec），C_3（100 kbyte，150 msec）があるとき，C_1 と C_2 はサイズ・総実行時間ともに近いが，C_3 はサイズ・総実行時間ともに大きく，リソース制約が厳しいとき C_1 が使用されている状況で，C_3 に置き換えることは難しい．粒度類似度はこのような粒度面の違いを表す．

$DRS_G(c_q, c_s)$ を，サイズ間の整数値類似度 dn とインタフェース粒度集合間の実体集合類似度 dr を用いて，以下に定義する．

$$
\begin{aligned}
&\text{コンポーネントの粒度面 } C_G,\\
&\text{インタフェース粒度 } I_G\\
&C_G ::= \{size : \text{int}, interfaces : \{i_1 : I_G, ..., i_n : I_G\}\}\\
&c_q, c_s : C_G\\
&c_q = \{size = size_q, interfaces = is_q\}\\
&c_s = \{size = size_s, interfaces = is_s\}\\
&DRS_G(c_q, c_s) ::= \frac{dn(size_q, size_s) + dr(is_q, is_s)}{2}
\end{aligned}
$$

粒度類似度算出時のインタフェース粒度 I_G は，引

数に初期値を代入した際の実行時間 $init$ と，初期値以外の値を代入した際の実行時間 any から構成される．インタフェースがその実装内部において分岐を持つ場合は，可能なあらゆる入力と状況を試さない限り，内部の複雑度を実行時間を持って測定したことには厳密にはならない．しかし，インタフェースの引数について代表的な値として，初期値と初期値以外の値を代入して測定することで，有向置換性類似度の定義上の理想的な利用文脈を想定した場合のコンポーネント間の差違を表すことができると考える．インタフェース間の粒度面類似度 dig を，各実行時間の整数値類似度 dn を用いて，以下に定義する．

$$I_G ::= \{init : \mathrm{int}, any : \mathrm{int}\} \quad i_q, i_s : I_G$$
$$i_q = \{init = in_q, any = an_q\}$$
$$i_s = \{init = in_s, any = an_s\}$$
$$dig(i_q, i_s) ::= \frac{dn(in_q, in_s) + dn(an_q, an_s)}{2}$$

インタフェース粒度集合間の実体集合類似度の算出時には，内部類似度 dx としてインタフェース粒度類似度 dig が用いられ，インタフェース粒度の実体 root_{ig} として，$\mathrm{root}_{ig} = \{init = 0, any = 0\}$ が用いられる．インタフェース粒度類似度 dig は，インタフェース粒度 I_G のすべての実体 i_g について，$dig(i_g, \mathrm{root}_{ig}) \geq dig(\mathrm{root}_{ig}, i_g)$ を満たすため，実体数考慮条件が満たされる．

0 以上の整数値間の整数値類似度 dn を，類似度関数 d を用いて以下に定義する．

$$x, y, n_q, n_s : \mathrm{int} \qquad 0 \leq n_q, n_s$$
$$min(x, y) ::= \begin{cases} x\ (x \leq y \text{のとき}) \\ y\ (\text{それ以外}) \end{cases}$$
$$dn(n_q, n_s) ::= d(n_q + 1, n_s + 1, min(n_q, n_s) + 1)$$

4. コンポーネント検索システムの構築

有向置換性類似度 $DRS(c_q, c_s)$ は，要求としてのコンポーネント c_q に対する c_s の適合度合いを反映する．そこでリポジトリからコンポーネントを検索する際に，インタフェース定義を中心として検索要求を満し，詳細な実装をともなわないプロトタイプを準備し，プロトタイプコンポーネントとリポジトリ中のコンポーネント間の有向置換性類似度を測ることで，その類似度の最も小さいものが要求に最も適合するコンポーネントとなる．この仕組みは，被検索コンポーネント本体以外の追加的な情報を必要とせず，コンポーネントのソースコードを参照せずに外部から調査可能な情報のみを用いている．

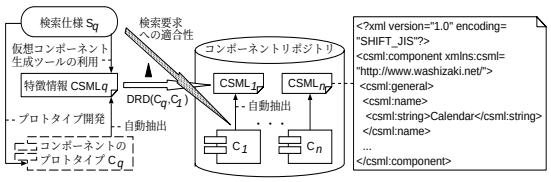

図 3 有向置換性類似度とコンポーネント検索
Fig. 3 Component retrieval using DRS.

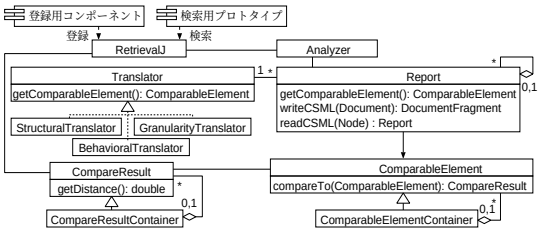

図 4 RetrievalJ 主要構成クラス図
Fig. 4 Class relationships in RetrievalJ.

我々は，有向置換性類似度に基づくコンポーネント検索システム RetrievalJ を Java 言語により開発し，単独起動型と，Web ブラウザを用いる Web 型の 2 種を用意した．コンポーネントシステムとして JavaBeans[15] を対象としている．RetrievalJ の概要を**図 3** に示す．リポジトリは，コンポーネントの登録時に自動抽出される特徴情報の記述ファイル群より構成される．記述形式として自動抽出可能な特徴情報を網羅するコンポーネント仕様記述言語 CSML（Component Specific Markup Language）を XML 言語に従って定義し用いている．

4.1 システム構成

検索システム RetrievalJ は，クラス RetrievalJ, Analyzer, Report, ComparableElement, Translator を中心として構成される（**図 4**）．Report は特徴情報を表す．ComparableElement は特徴間の比較を行う．Translator は Report から ComparableElement への変換を構造・振舞い・粒度の各種別に対応して行う．CompareResult は有向置換性類似度の値を表す．

Analyzer はコンポーネントの静的・動的解析を行う．静的解析では，JavaBeans コンポーネントのメタ情報を表す BeanInfo オブジェクトを内観機構により調査し，名前とサイズ，およびインタフェースの構造情報を取得する．動的解析では，コンポーネントの実体を内部で生成し，すべてのインタフェースを引数に初期値および初期値以外の値を与えて自動実行し，実行にかかる時間と戻り値，および実行前後で変化する状態情報を取得する．コンポーネントは定義より他への依存性を持たないため，単体で生成し動的に解析可

```
public class CalendarProto extends Component {
 private String selectedDate = null;
 public java.lang.String getSelectedDate(){
  return selectedDate; }
```

図 5 選択日付取得インタフェースのプロトタイプ実装
Fig. 5 Prototype implementation of acquiring dates.

能である．

RetrievalJ はプロトタイプの入力を受け付け，検索結果を出力する．利用者は求めるコンポーネントのプロトタイプを準備する必要があるが，開発対象アプリケーション中の特定部分についてプロトタイプを準備することで，あいまいな要求仕様をより完全なものとし，初期段階でテストケースを生成可能となることから，プロトタイプ開発の重要性が指摘される[16]．プロトタイプ開発の例として，カレンダー関連機能を持つコンポーネントを求める状況をあげる．利用者は同機能の実現に必要なインタフェースを考え，たとえば選択日付取得インタフェースを持ったプロトタイプを準備する．準備する際に構造面だけでなく振舞面も考慮し，インタフェースがコンポーネント内部で使用する観測可能な状態も実装する．ただし，プロトタイプが実際にカレンダー関連機能を提供する必要はない．同インタフェースを，観測可能な状態 selectedDate の観測操作としてプロトタイプ実装した例を**図 5** に示す．図 5 の選択日付取得インタフェースは，リポジトリ中のコンポーネント群が持つ同様の機能を実現するインタフェース群（たとえば表 3）との類似性が検出され検索結果に反映される．

また，プロトタイプを作成せず直接 Report オブジェクト群を生成する仮想コンポーネント生成ツールを用意した．利用者は同ツールを用いて，要求仕様をビジュアルな操作で与えて Report オブジェクト群を生成し，リポジトリ中のコンポーネント群との比較に利用できる．

4.2 登録・検索のシナリオ

RetrievalJ を用いたコンポーネントの登録シナリオを (1a)〜(3a) に示す．

(1a) 登録用コンポーネントの入力：登録者は JavaBeans コンポーネントがパッケージ化された Jar ファイル，または Java クラスファイルを RetrievalJ に入力する．

(2a) コンポーネントの自動解析：Analyzer が入力コンポーネントの静的･動的解析を自動的に行い，Report オブジェクト群を生成する．

(3a) CSML 形式での保存：Report オブジェクト群が階層構造に従って自身を CSML 形式で保存する．

また，検索シナリオを (1b)〜(7b) に示す．

(1b) 検索用プロトタイプ開発：利用者は要求仕様を満たすプロトタイプを開発する．

(2b) 検索用コンポーネントの入力：利用者はプロトタイプ，または仮想コンポーネント生成ツールにより生成される Report オブジェクト群を RetrievalJ に入力する．

(3b) 自動解析：Analyzer は検索用コンポーネントの Report オブジェクト群を生成し，指定される Translator オブジェクトの組合せにより要求としての ComparableElement オブジェクト群を生成する．

(4b) 検索条件の設定：利用者は $DRS_{S|B|G}(c_q, c_s)$ の組合せ，重み付け（w_1〜w_3）の値，リポジトリの場所，および検索結果を表示する際に適合・不適合を区別する閾値の各指定を行う．

(5b) CSML ファイルの読み込み：リポジトリ中の全 CSML ファイルを読み込み，Report オブジェクト群へ復元する．指定される Translator オブジェクトの組合せより，復元された Report オブジェクト群から ComparableElement オブジェクト群を生成し，先に生成済の要求としての同オブジェクト群との比較操作を行い，結果として CompareResult オブジェクト群を得る．

(6b) 検索結果の提示：CompareResult オブジェクト群を有向置換性類似度の小さい順に並べ替え，閾値に基づき適合・不適合一覧を表示する．また，一覧表示中から任意のコンポーネントを選択し，要求として与えたコンポーネントとの有向置換性類似度を，インタフェースごとに詳細表示する．利用者はこの表示機能を用いて，リポジトリ中のコンポーネントがどの特徴面について要求と異なるかを詳細に把握可能である．

(7b) ダウンロード：利用者は検索結果のコンポーネント群から任意のものを選択してダウンロードする．

5. 評　　価

本手法の検索性能の評価にあたり以下の環境を扱う．

本手法として用いる有向置換性類似度の設定にあたり，有向置換性類似度を構成する構造類似度・振舞類似度・粒度類似度の重み付けを行う必要がある．本手法が提案する検索システムは，利用者は検索時において一度の検索で完結せずに，適宜重み付けを変更して試行錯誤的に再検索を行うことができる．

そこで，重み付けを変更していくにあたっての基準として，構造面・振舞面・粒度面の 3 種を均等に考慮する以下の DRS' を用いる．

$$DRS'(c_q,c_s)=\frac{1}{3}DRS_S(c_q,c_s)+\frac{1}{3}DRS_B(c_q,c_s)+\frac{1}{3}DRS_G(c_q,c_s)$$

また，問題領域の種別とは無関係に構造面が重要であることが経験的に分かったため，構造類似度を重み付けで最重視し，さらに，コンポーネントシステムを限定していることから粒度面の重要性を軽視した以下の DRS'' も別に用いる．

$$DRS''(c_q,c_s)=0.5DRS_S(c_q,c_s)+0.3DRS_B(c_q,c_s)+0.2DRS_G(c_q,c_s)$$

以降において，DRS' を用いる本手法を [DRS']，DRS'' を用いる本手法を [DRS"] とそれぞれ記述する．

従来手法として Spanoudakis らの類似度手法（以下 [SC94] [10]）と Michail らの類似度手法（[MN99] [8]）を対象とする．両手法はコンポーネント本体以外を必要とせず，準備コストに優れる点で本手法に近い関係にある．[SC94] は要求として与えるコンポーネントに対し，クラスとしての継承階層上の位置が近いものを高く順位付ける．[MN99] はコンポーネント名・インタフェース名を語分割して索引付けした項目集合を用い，要求として与えるコンポーネントが持つ項目集合との頻度を考慮した類似度の大きいものを高く順位付ける．

文献 15)，17)~19) および Web サイト [20)~25)] で公開される合計 257 個の JavaBeans コンポーネントを評価サンプルとする．全サンプルに対して，各サンプルに付属する形式が一定でない説明書（利用文書）を用いて，扱う問題領域に特化して機能的に類似すると人手によって判断されたコンポーネント群を，適合グループとして 13 グループ設定した．各適合グループの名前と主要機能，所属するコンポーネント数，および所属コンポーネント群のオブジェクト指向クラスとしての継承階層上の深さの平均を**表 2** に示す．

適合グループ内のコンポーネントは，それぞれが持つ機能について類似すると判断されたものであり，同一の適合グループに属するコンポーネントどうしでの置換は，他の適合グループ中のコンポーネントや適合グループに属さないコンポーネントとの置換と比較して容易と考えられる．したがって，検索要求として適合グループ内のあるコンポーネントを与えたとき，検索結果として同グループ内の他のコンポーネントが高く順位付けされることが，検索性能の高さを表すことになる．

5.1 再現率と精度平均

[DRS']，[DRS"]，[SC94]，[MN99] の検索性能の比較として，Calendar グループ内のコンポーネントを検索要求として与えて他の全 256 個のコンポーネント群から検索を行ったときに，Calendar グループ内のコンポーネント群が高く順位付けられた検索結果を得られるかを評価した．ただし，本手法の有向置換性類似度は方向をともなう置換コストを表すため，同一の適合グループ（この場合は Calendar グループ）から与える検索要求としてのコンポーネントの種別に応じて，検索要求と同一の適合グループ中のコンポーネントの順位付けの高さが異なって得られる可能性がある．そこで，Calendar グループ中のすべてのコンポーネントをそれぞれ検索要求として与えた際に，Calendar グループに属するコンポーネントを適合検索結果とする再現率・精度を求め，再現率のとりうる各値ごとにすべての精度の値を平均した精度平均をもって検索性能の評価を行う．

検索結果の順位 x 位までについての再現率 $R(x)$ と精度 $P(x)$ を以下に設定する．

$$R(x)=\frac{x\text{ 位内で要求と同グループのコンポーネント数}}{\text{要求と同グループ中の全コンポーネント数}}$$
$$P(x)=\frac{x\text{ 位内で要求と同グループのコンポーネント数}}{x}$$

Calendar グループに属するコンポーネント数は 5 つであるため，Calendar グループから 1 つのコンポーネントを検索要求として検索するとき，再現率のとる値は 0.25，0.5，0.75，1.0 の 4 種である．Calendar グループ中のすべてのコンポーネントをそれぞれ検索要求として与えて，[DRS']，[DRS"]，[SC94]，[MN99] の各手法によって 256 個のコンポーネントについて検索を行った結果の再現率と精度を求め，再現率の 4 種の値ごとに精度の値を平均することで精度平均を得る．再現率と精度平均を測定した結果を**図 6** に示す．図 6 において，[DRS"] の結果は，再現率・精度平均のと

表 2 適合グループ
Table 2 List of agreement groups.

グループ名	主要機能	個数	深さ平均
Calendar	カレンダー	5	6.0
ProgressBar	割合バー	3	4.7
SMTP	メール送信	3	3.0
POP3	メール受信	3	3.0
Clock	時計	3	5.3
Gauge	羅針ゲージ	2	5.0
PlotChart	点グラフ描画	2	4.5
Calculator	電卓	2	6.5
Finger	個人情報取得	2	3.0
Stock	株価取得	2	4.5
ScrollBar	スクロールバー	2	5.5
SMTPUI	メール送信用 UI	2	4.5
POP3UI	メール受信用 UI	2	4.5

もに [SC94]，[MN99] を上回っており，検索性能がより高いことが分かる．一方，[DRS'] の結果は [MN99] と同程度であり，[DRS"] の結果を下回った．

この結果は Calendar グループ内のコンポーネントには，カレンダー関連機能の実現のため，特に構造面で類似するインタフェースが共通してあることによる．たとえば，選択された日付の取得機能を実現するインタフェースがあげられる．Calendar [21] を要求として与えた際の，Calendar グループ内のコンポーネントの本手法における選択日付取得インタフェース間の構造類似度，および各検索手法による検索結果順位を**表 3** に示す．

本手法における [DRS"] は [DRS'] に比べてコンポーネントの構造面を大きく考慮するため，これらのインタフェースの構造面での類似性から，[DRS"] では他のコンポーネントに対して Calendar グループ内のものを高く順位付けした．CalendarViewer [17] は選択日付取得機能を持たないため低く順位付けられた．

対して [SC94] では，コンポーネントのクラスとしての継承関係のみを考慮し，インタフェース間の差異を考慮しない．同じ機能を持ったコンポーネントでも開発者が異なるとき，設計するクラス階層が一般に異なるため，本手法に対して大幅に低い性能を示す．[MN99] では，インタフェースを考慮するが，名前に限定され型情報を考慮しないため，検索性能は本手法より低い．

以上より，Calendar グループ中のコンポーネント群について，検索要求として与えるコンポーネントと名前や型等の構造面が類似していることが，本手法で効果的に検索するための前提的な特徴であることが分かる．

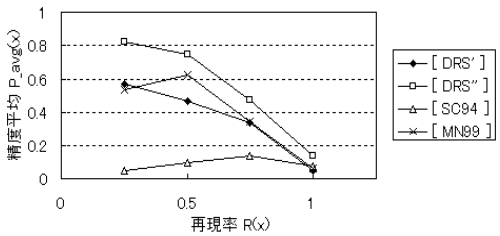

図 6 再現率–精度平均の比較
Fig. 6 Comparison of recall–precision average.

5.2 正規化再現率平均

検索結果全体の評価指標として，正規化再現率 [26] を平均した正規化再現率平均を定義して用いる．正規化再現率は理想的な検索性能への近さを表し，値が最大値 1 のとき要求に適合する n 個のコンポーネントが検索結果中の n 位までにすべて出現することを意味する．正規化再現率は，再現率を順位で重み付けして平均したものとみることができる [27]．

まず，グループ G 中のコンポーネント c_g を検索要求として，残り 256 個のコンポーネントを検索したときの正規化再現率 $R_{norm}(c_g)$ を以下に設定する．

$$|G| ::= (\text{適合グループ } G \text{ 中の全コンポーネント数})$$
$$rank(c) ::= (\text{コンポーネント } c \text{ の検索結果順位})$$
$$R_{norm}(c_g) ::= 1 - \frac{2\sum_{c \in G-\{c_g\}} rank(c) - |G|(|G|-1)}{2(|G|-1)(257-|G|)}$$

この $R_{norm}(c_g)$ は，検索要求 c_g を与えて得られた検索結果について，c_g が属する適合グループ中の他のコンポーネントが高く順位付けされている度合いを 0 から 1 の間で正規化された値として表す．

続いて，グループ G の正規化再現率平均 $R_{avg}(G)$ を，グループ G 中のコンポーネント c による正規化再現率 $R_{norm}(c)$ を用いて以下に定義する．

$$R_{avg}(G) ::= \frac{1}{|G|} \sum_{c \in G} R_{norm}(c)$$

表 2 の全適合グループについて，適合グループからすべてのコンポーネントをそれぞれ検索要求として与える検索を [DRS']，[DRS"]，[SC94]，[MN99] によって行い，各手法の検索結果について正規化再現率平均を測定した．

さらに，検索対象コンポーネント群の順位付けを，コンポーネントが持つ名前の間の最長一致部分文字列の長さによって行う文字列手法を追加し，文字列手法（[文字列] と表記）を用いた検索結果についても同様に正規化再現率平均を測定した．すべての測定結果と

表 3 インタフェース構造の比較と検索結果順位
Table 3 Comparison of interface structures and result of ranking.

コンポーネント名	インタフェース名	戻り値型	構造類似度	[DRS"]	[SC94]	[MN99]
Calendar [21]	getResutSelectedDateAsString	java.lang.String	–	–	–	–
SSCalendar [23]	getAllSelectedDates	ss.util.Message	0.067	1	19	4
CalendarBean [20]	getSelectedDate	java.lang.String	0.011	2	76	14
CalPanel [23]	getDate	int	0.071	5	20	5
CalendarViewer [17]	（該当なし）	（なし）	0.187	12	21	63

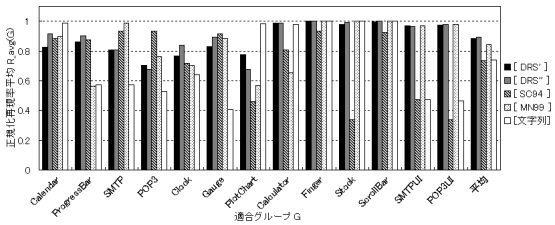

図 7 正規化再現率平均の比較
Fig. 7 Comparison of normalized recall average.

表 4 正規化再現率の得られた測定値
Table 4 Acquired measure value of normalized recall.

手法	最小値	最大値	範囲
[DRS']	0.444	1.0	0.555
[DRS"]	0.545	1.0	0.454
[SC94]	0.335	0.935	0.599
[MN99]	0.309	1.0	0.690
[文字列]	0.312	1.0	0.687

各手法ごとの平均を**図 7** に示す．また，正規化再現率の平均を計算する前の各正規化再現率の測定値について，各手法ごとに得られた最小値と最大値，およびその間の範囲（最大値 – 最小値）を**表 4** に示す．

図 7 より，本手法の [DRS'] は全 13 グループのうち，[SC94] に対して 8 つのグループで高い値を示し，5 つのグループで低い値を示した．[MN99] に対して，2 つのグループで同一の値で並び，4 つのグループで高い値を示し，7 つのグループで低い値を示した．[文字列] に対して，1 つのグループで最高値（$R_{avg}(G) = 1$）で並び，8 つのグループで高い値を示し，4 つのグループで低い値を示した．以上の結果から，[DRS'] は検索対象とするコンポーネントの種類によっては，従来手法より検索精度が低いことが分かる．

本手法の [DRS"] は，[SC94] に対して 10 のグループで高い値を示し，3 つのグループで低い値を示した．[MN99] に対して，2 つのグループで最高値で並び，7 つのグループで高い値を示し，4 つのグループで低い値を示した．[文字列] に対して，2 つのグループで最高値で並び，8 つのグループで高い値を示し，3 つのグループで低い値を示した．表 4 より [DRS"] について得られた正規化再現率の範囲は，他の手法と比較して大きい値から始まっており，範囲の大きさが最も小さい．また，正規化再現率平均の全適合グループについての平均値が全手法中で最も大きいことから，[DRS"] の検索精度は，検索対象コンポーネントの種別にかかわらず，他の手法と比較して高く安定していると考えられる．

POP3 グループにおいて，[DRS']，[DRS"] の結果

```
(a) Bean1: public void setText(String newText){
             synchronized(this) {
              text = newText;  // 以降，実装
             } }
(b) Bean2: public void setText(String newText){}
```

図 8 文字列設定インタフェースの実装比較
Fig. 8 Comparison of interface implementations.

表 5 差異の判別可能性
Table 5 Distinction possibilities of difference.

手法	Bean1	Bean2
[SC94]：類似度	0.3935 (2)	0.3935 (2)
[MN99]：類似度	1362.1 (3)	1362.1 (3)
本手法：[DRS"]	0.0238 (1)	0.0304 (5)
本手法：$1.0DRS_S$	0.0399 (1)	0.0399 (1)

が [SC94] を下回った原因について述べる．POP3 グループ中の 3 つのコンポーネントは，インタフェースの構造面や振舞面について類似性がみられず，そのため [DRS']，[DRS"] の結果は低かった．対して [SC94] は，同グループ中のコンポーネントはクラスとして継承階層上で浅いものが集まっていたため，類似度を高く算出している．このように本手法は，対象コンポーネントが要求する機能を持ちながら，そのインタフェース構造・振舞いの予想が困難な場合に適切に検索できないことがある．

[文字列] の結果は 4 つのグループにおいて本手法の [DRS']，[DRS"] を上回ったが，他のグループについて [DRS']，[DRS"] の結果を大きく下回っている．また表 4 より，[文字列] について得られた正規化再現率の範囲は，本手法の [DRS']，[DRS"] と比較して大きく，[文字列] によるコンポーネントの検索精度は安定していない．

以上より，[DRS"] を用いる本手法が，検索精度の安定したコンポーネント検索を実現する手法として最も適していることが分かった．

5.3 特徴全体の考慮可能性

インタフェース構造がすべて同一で，GUI ラベルとして機能するコンポーネント Bean1 と，機能しない Bean2 を用意した．要求として java.awt.Label コンポーネントを与えた際に，Bean1 および Bean2 の要求への適応度の違いを識別可能か評価した．Bean1，Bean2 のプログラムの抜粋として，GUI ラベル上の文字列設定インタフェースの部分を**図 8** (a)，(b) に示す．

従来手法として [SC94]，[MN99] の類似度を用いた．本手法として，6 章で設定した 3 種の類似度を組み合わせたもの（[DRS"]）と構造類似度（$1.0DRS_S$）を用いた．評価結果を**表 5** に示す．括弧内は 257 個のサンプルに両コンポーネントを加えて検索した際の検

索結果順位を示す．[SC94]，[MN99] ではコンポーネントの振舞いを考慮しないため，両コンポーネントの違いを識別不可能である．対して本手法では，動的な解析により，たとえば文字列設定インタフェースの実行時に Bean1 では状態 text の変化を観測し，Bean2 では観測できず外部に対して無変化であることを把握する．したがって本手法では，構造類似度において両コンポーネントの差違を識別できないが，振舞面を含んだ特徴全体を考慮可能な DRS'' によって Bean1 と Bean2 の違いが識別され，要求に対し Bean1 がより適合することが判別される．

5.4 実行効率

正規化再現率平均の評価時に本手法で 256 個のコンポーネント群に対し 12 回検索した際の処理時間を測定した（環境：Sun Java2 RE1.3 / HotSpot-ClientVM, PentiumIII/600 MHz, Memory256 MB, Windows2000）．検索 1 回あたりの処理時間（単位：msec）は最小で 20118，最大で 127433，平均で 58684 であった．平均時間が 1 分弱であり，利便性を考慮して許容される範囲内と考えられる．

6. おわりに

コンポーネント間の有向置換性類似度を提案し，有向置換性類似度を用いて，ソースコードが非開示なコンポーネントに適用可能で，コンポーネント本体以外を必要としないコンポーネント検索システムを実現した．本手法は，複数の特徴面を同時に考慮可能なため，単一の特徴面だけでは識別できなかったコンポーネント間の差違を詳細に識別可能とし，検索精度が従来の手法と比較して高く安定していることが分かった．本論文で述べた RetrievalJ は，http://www.fuka.info.waseda.ac.jp/Project/CBSE/ から利用可能である．今後，本手法が有効なコンポーネントの問題領域の境界を明らかにしていく予定である．

参考文献

1) Hopkins, J.: Component Primer, *Comm. ACM*, Vol.43, No.10, pp.27–30 (2000).
2) Meiling, R., et al.: Storing and Retrieving Software Components: A Component Description Manager, *Proc. Australian Computer Science Conference*, pp.107–117 (2000).
3) Aoyama, M., et al.: Software Commerce Broker over the Internet, *Proc. Annual International Computer Software and Applications Conference*, pp.430–435 (1998).
4) Yacoub, S., et al.: Characterizing a Software Component, *Proc. International Workshop on Component-Based Software Engineering* (1999).
5) 山本浩数，鷲崎弘宜，深澤良彰：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論文集，pp.105–116 (2001).
6) Ahn, S., et al.: A Specification Extraction Technique for JavaBeans Component using XML, *Proc. International Workshop on Software Architecture and Components*, pp.76–82 (1999).
7) Seacord, R., et al.: Agora: A Search Engine for Software Components, *IEEE Internet Computing*, Vol.2, No.6, pp.62–70 (1998).
8) Michail, A. and Notkin, D.: Assessing Software Libraries by Browsing Similar Classes, Functions and Relationships, *Proc. International Conference on Software Engineering*, pp.463–472 (1999).
9) Penix, J., et al.: Efficient Specification-Based Component Retrieval, *Automated Software Engineering*, Vol.6, No.2, pp.139–170 (1996).
10) Spanoudakis, G. and Constantopoulos, P.: Measuring Similarity Between Software Artifacts, *Proc. International Conference on Software Engineering & Knowledge Engineering*, pp.387–394 (1994).
11) Zaremski, A. and Wing, J.: Signature Matching: A Tool for Using Software Libraries, *ACM Trans. Software Engineering and Methodology*, Vol.4, No.2, pp.146–170 (1995).
12) Lai, S. and Yang, C.: A Software Metric Combination Model for Software Reuse, *Proc. Asia-Pacific Software Engineering Conference*, pp.70–77 (1998).
13) 田中栄一：構造をもつものの距離と類似度，情報処理，Vol.31, No.9, pp.1270–1279 (1990).
14) Cardelli, L.: Type Systems, *Handbook of Computer Science and Engineering*, CRC Press (1997).
15) Watson, M.: *Creating JavaBeans: Components for Distributed Applications*, Morgan Kaufmann (1997).
16) Maiden, N. and Ncube, C.: Acquiring COTS Software Selection Requirements, *IEEE Software*, Vol.15, No.2, pp.46–56 (1998).
17) Neil, J.: *JavaBeans Programming from the Ground Up*, McGraw-Hill (1998).
18) Harold, E.: *JavaBeans: Developing Component Software in Java*, IDG Books (1998).
19) Englander, R.: *Developing JavaBeans*, O'Reilly (1997).

20) http://www.jars.com/
21) http://www.alphaworks.ibm.com/alphabeans/
22) http://www.javacats.com/
23) http://www.sjug.org/showcase/
24) http://openlab.ring.gr.jp/kyasu/
25) http://www.bekkoame.ne.jp/~sakamo_m/
26) Bollman, P.: The Normalized Recall and Related Measures, *Proc. Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.122–129 (1983).
27) 徳永健伸：情報検索と言語処理，東京大学出版会 (1990).

(平成 13 年 10 月 5 日受付)
(平成 14 年 3 月 14 日採録)

鷲崎　弘宜

昭和 51 年生．平成 13 年早稲田大学大学院理工学研究科修士課程修了．現在，同大学院理工学研究科博士課程に在学中．平成 14 年同大学理工学部助手．コンポーネント指向ソフトウェア開発の研究に従事．電子情報通信学会，ソフトウェア科学会，ACM，JapanPLoP 各会員．

深澤　良彰（正会員）

昭和 51 年早稲田大学理工学部電気工学科卒業．昭和 58 年同大学院博士課程中退．同年相模工業大学工学部情報工学科専任講師．昭和 62 年早稲田大学理工学部助教授．平成 4 年同教授．工学博士．ソフトウェア工学，コンピュータアーキテクチャ等の研究に従事．ソフトウェア科学会，IEEE，ACM 各会員．