修士学位論文

題目

Java ソフトウェア部品検索システム SPARS-J の構築

指導教官
井上 克郎 教授

報告者
西 秀雄

平成 16 年 2 月 12 日

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 ソフトウェア工学講座

平成 15 年度 修士学位論文

Java ソフトウェア部品検索システム SPARS-J の構築

西 秀雄

内容梗概

近年，インターネットの普及により WWW を通じて大量のソフトウェア資産が容易に入手可能となった．プログラムソースの公開と再配布を許可しているオープンソースソフトウェア開発コミュニティでは，ソフトウェアプロダクトが増加するにつれて開発基盤としての価値だけでなく，ソフトウェアを構成する様々な部品を発見して他のソフトウェア開発に再利用したり参考にするといった，ソフトウェア資産を有する大規模なライブラリとしての価値も高まっている．しかし，ソフトウェアは自然言語で記述された文章とは違った側面をもっているため，ソフトウェア部品の分類や運用を適切に行なうことは難しい．

そこで大規模なライブラリを依存や類似といったソフトウェア的な特性をふまえて自動的に部品に分類し，可視化する必要がある．そのため，ライブラリから開発者が必要としている機能を持つ部品を検索可能とし，さらに使用例などの情報を併せて提供する検索システムが必要となる．本研究では Java ソースコード集合のライブラリを対象としたソフトウェア部品検索システムとして，SPARS-J (*Software Product Archive, analysis and Retrieval System for Java*) の構築を行ない性能を評価した．

SPARS-J はソフトウェア部品特有の特性を考慮しながら大規模なライブラリの分類・検索を自動的に行なうシステムである．キーワードによる全文検索を行ない，ソフトウェアのソースコードを効率良く検索することが可能である．さらに，検索結果表示の際にソフトウェア部品に関する詳細な情報を併せて提供する．このシステムを用いることで，ライブラリの知識が無い開発者も有用なソフトウェア部品やそれに付随する有益な情報を容易に入手することができる．性能評価として本システムを 5 つのライブラリに適用して，時間的コスト，空間的コストの評価を行ない，本システムが実用的であることを確認した．

主な用語

ソフトウェア部品 (Software Component)
ソフトウェア検索 (Software Retrieval)

ソフトウェア再利用 (Software Reuse)
プログラム理解 (Program Understanding)
Java

目 次

1　まえがき

近年のインターネットの普及により，WWW を通じて大量のソフトウェア資産が容易に入手可能となった．プログラムソースの公開と再配布を許可しているオープンソースソフトウェア開発コミュニティでは，ソフトウェアプロダクトが増加するにつれて開発基盤としての価値だけでなく，ソフトウェアを構成する様々な部品を発見して他のソフトウェア開発に再利用したり参考にするといった，ソフトウェア資産を有する大規模なライブラリとしての価値も高まっている．SourceForge[31] ではソースコードのストレージやコミュニケーションに必要なメーリングリストなどソフトウェア開発に必要とされる物的資源を提供しており，2004 年 2 月現在 75,000 以上ものプロジェクトを保持し，開発者として登録されているユーザ数は 78 万を越える．

ソフトウェアは自然言語で記述された文章とは違った側面をもっているため，ソフトウェア部品の分類や運用を適切に行なうことは難しい．例えばソフトウェア部品の再利用 [17][8] 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ための代表的なソフトウェア工学技術の一つとして知られているが，その効果を最大限に得るためには，開発者がライブラリに関する十分な知識を持つことが必要となる．しかしながら，多数の開発者が参加する開発プロジェクト内でライブラリに関する知識の共有を行なうことは非常に困難でコストのかかる作業である．このため，再利用可能なソフトウェア部品がライブラリ中に存在するにも関わらず，同種の部品が独立して開発されていることが多々ある．

そこで，大規模なライブラリをソフトウェア的な特性をふまえて自動的に部品に分類し，可視化する必要がある．そのためにライブラリから開発者が必要としている機能を持つ部品を検索可能とし，さらに使用例などの情報を併せて提供する検索システムが必要となる．本論文では Java ソースコード集合のライブラリを対象とした，ソフトウェア部品検索システム SPARS-J (*Software Product Archive, analysis and Retrieval System for Java*) の構築を行ない，その性能を評価した．

SPARS-J は，依存や類似といったソフトウェア部品特有の特性を考慮しながら，大規模なライブラリの分類・検索を自動的に行なうシステムである．キーワードとトークン種類を検索キーとした全文検索を行ない，ソフトウェアのソースコードを効率良く検索することが可能である．さらに，検索結果表示の際にソフトウェア部品に関する詳細情報を併せて提供する．このシステムを用いることで，ライブラリの知識が無い開発者も有用なソフトウェア部品やそれに付随する有益な情報を容易に入手することができる．性能評価では 5 つのライブラリを対象にして本システムを適用し，時間的コスト，空間的コストの比較評価をすることで，システムの性能評価を行なう．

以降，2 節でソフトウェア部品検索システムについて述べ，関連研究の紹介を行なう．3

節で SPARS-J で用いるソフトウェア部品類手法や検索結果の順位付け手法について述べ，4節でシステムの構成について説明する．5 節では SPARS-J の性能評価を行なう．最後に 6 節で本論文のまとめと今後の課題について述べる．

2　ソフトウェア部品検索システムについて

本節ではソフトウェア部品について説明した後，ソフトウェア部品検索システムの概要と，既存の研究について述べる．

2.1　ソフトウェア部品と再利用

一般に**ソフトウェア部品** *(Software Component)* は**再利用** *(reuse)* できるように設計されたソフトウェアの実体とされる [9]．過去のソフトウェア開発における成果物などからなるソフトウェア部品の集合を**ライブラリ** *(library)* という．以下，ソフトウェア部品を単に部品と呼ぶ．ライブラリ中の既存の部品を同一システム内や他のシステムで用いることを再利用といい [8]，ソフトウェア生産性と品質を改善し，結果としてコスト削減するという報告が多く出されている [5][16]．部品という概念はプログラムソースコードやバイナリ実行ファイル，その他に設計仕様や構造，テスト項目，そしてそれらの文書などであったりと，ソフトウェア開発過程で生成されたあらゆる成果物に適用され，その種類は様々である．部品の再利用可能な度合を**再利用性** *(reusability)* といい，特定の設計やコーディングの標準に従うことで，部品の再利用性を向上させることができる．ソフトウェア再利用には，部品の形式や再利用の目的に応じて，**ホワイトボックス再利用** *(white-box reuse)* と**ブラックボックス再利用** *(black-box reuse)* が存在する．

[ホワイトボックス再利用]

仕様や文書，プログラムソースコードの再利用を指す．部品の内部構造に基礎を置くため，部品詳細を把握することが可能であり，用途に応じて修正可能でプラットフォームに非依存である．ホワイトボックス再利用における再利用者は主にソフトウェア開発者である．特にソースコード再利用に関しては，ライブラリ内の部品を利用者の再利用環境に応じて洗練する手法に関する研究 [23] などが存在する．

[ブラックボックス再利用]

主にバイナリ実行ファイルの再利用を指す．具体的には JavaBeans や ActiveX/DCOM, CORBA などがあり，部品の詳細を知らずにその機能だけを再利用したい時に有効である．プログラムに詳しくないエンドユーザのソフトウェア開発で行なわれるもので，開発形態としてはビジュアルプログラミングなどがある．コンパイル不要で迅速に再利用可能な反面，プラットフォームに依存しており詳細な解析は困難である．

2.2　ソフトウェア部品検索システムについて

ソフトウェア部品検索システムとは，部品を用途や信頼度に応じて分類してライブラリとして蓄積し，ユーザの指定に応じて部品を探し出す自動化システムである．支援対象は，設

計者やプログラマなど部品を検索し使用するソフトウェア開発者と，ライブラリの部品を分類し維持管理にも責任を持つライブラリ管理者である．

ソフトウェア部品検索システムが果たす役割は，主に以下の三点である．

1. 部品の分類
   部品の分類は，ソフトウェア部品検索システムにとって重要な課題である．ネジのような機械部品には規格や用途に関する違いが明確に存在しているためその分類は容易だが，ソフトウェア部品には明確な規定や分類の指針が存在しない．整理の良い開発者であれば自身が過去に開発した部品を探しだし使うことはあるかもしれないが，他人の開発した部品から用途に適するものを探し出すことは難しい．そのため，ライブラリは適切な分類手法を用いて適切に整理されている必要がある．現在，WWW を通じて大量の部品が比較的容易に入手可能であるが，それらは自動的に分類されることが望ましい．

2. 部品の検索
   部品の検索は，分類されたライブラリから必要な部品を探し出す作業である．自然言語文書の検索システムでは検索しようとするトピックのキーワードをクエリに用いて検索するのが普通であるが，ソフトウェア部品検索システムではその他に部品の特徴をあらわすメトリクスや，記述されている言語，求める機能の抽象表現などをクエリに用いる場合もある．

3. 部品の収集
   部品の収集は，近年の WWW の発展に伴い注目されてきた．部品の入手先が企業の閉じたネットワーク内に限られていた時代には，先述した部品の分類と検索に関する問題が焦点であった．しかし，WWW の発展によって大量の部品を入手可能になると，WWW 上のソフトウェア部品をライブラリと見なしてどのように部品を収集するかといった問題に関する手法が多く登場している．

オープンソースソフトウェア開発コミュニティSourceForge では，ソースコードのストレージやコミュニケーションに必要なメーリングリストなどソフトウェア開発に必要とされる物的資源を提供することを主な目的としている．保持するソフトウェアプロダクトが増加するにつれて，開発基盤としての価値だけでなくソフトウェア部品を検索，発見する場としての価値も高まっている．例えば SourceForge は 2004 年 2 月現在 75,000 以上ものプロジェクトを保持しており，開発者として登録されているユーザ数は 78 万を越える．WWW 上には他にもソフトウェア開発コミュニティも存在しており，以後も発展していくとみられている．このような事実から，現在，大規模なソフトウェア部品集合に対するソフトウェア部品検索システムが求められている．

2.3 関連研究

これまでにソフトウェア部品検索を行なうために様々な手法が提案されている．

WWW のソフトウェア部品を自動収集し，それらを解析することでデータベースに分類・蓄積し，作成したデータベースを対象に検索を行なう形式の検索システムに [27][4][20] がある．Agora[27] はサーチエンジン AltaVista の SDK を利用してソフトウェア部品の自動収集を行なう．CORBA オブジェクトである ORB や JavaBeans を検索可能である．jCentral[4] は IBM が開発した Java のソフトウェア部品検索システムである．jCentral では Java ソースコードの他，アプレット，JavaBeans，ニュース，FAQ，チュートリアルなどの情報を検索することが可能である．亀井ら [20] の提案したシステムでは，クエリにソフトウェアメトリクスなどの特有の値を含めてソフトウェアを検索することができる．

庭山ら [25] はセマンティック Web の手法を利用してソフトウェアのソースコードにメタデータを付与し，検索ワードをオントロジ変換したものとのマッチングを行なうことで，ソフトウェア部品を意味に基づいて検索する手法を提案している．また，鷲崎ら [30] は独立して再利用可能なコンポーネントを，検索および試行可能な検索システムの提案を行なっている．提案システムでは，Java ソースコードからクラス間の依存関係を解析することで，要求された機能を実現する複数のソースコードを JavaBeans コンポーネントとして抽出し，ユーザに提供する．

その他，ソフトウェア部品検索として用いることが可能な自然言語文書検索システムとして，日本語全文検索システム namazu[24] や Web ページ検索システム Google[14] などが存在する．

3 Java プログラムを対象としたソフトウェア部品検索システム

本節では，本研究で構築する SPARS-J の概要と，SPARS-J で用いている主な技術について述べる．

3.1 SPARS-J の概要

SPARS-J*(Software Product Archive, analysis and Retrieval System for Java)* は，Java 言語 [18] で記述されたプログラムを対象としたソフトウェア部品検索システムである．Java の利用者が指定したキーワードと関連する部品を検索する．検索結果はキーワードの出現頻度と，利用関係に基づくソフトウェア部品重要度評価手法 [15] によって順位付けされる．さらに，検索結果に併せて部品間の利用関係や類似部品に関する情報を提供することで，再利用やプログラム理解・保守を行なうことが可能となる．

3.2 SPARS-J におけるソフトウェア部品

SPARS-J では Java の一つのクラス・インタフェースのソースコードを部品として扱う．Java はオブジェクト指向に基づいた言語であり，クラス・インタフェースはソフトウェア部品の概念を満たす．SPARS-J はプログラム理解，保守といったより広範な利用目的を想定している．そのため，部品の詳細を利用者が把握可能なソースコードという形態が適切だと考えた．その他ソースコードの配布により，利用者の目的に応じたカスタマイズが可能であるという利点が存在する．同時に，単に部品を再利用したいという場合にはクラスや Java に関する知識が必要となるが，後述する様々に提供する部品の詳細情報を提供することでその欠点も補うことができると考えている．部品の索引は索引キーの集合から成り立つ．索引キーは Java ソースコード中に現れる予約語以外のキーワードとトークン種類の組である．また，索引キーに用いられるキーワードのことを特に索引語と呼ぶ．同様に，検索時に用いられるキーワードを検索語と呼び，検索語と検索対象のトークン種類の組を検索キーと呼ぶ．

3.3 利用関係

ソフトウェアは複数の部品の集合体として構成される．これらの部品は互いに独立であって，お互いに影響を及ぼさず部品単位で代替可能である．ソフトウェアはそれら独立な部品間でメッセージをやり取りし協調作業を行うことで一つの大きな機能を提供する．Java のクラスを部品として捕らえると，他クラスのフィールド参照やメソッド呼出しなどがメッセージに相当する．このメッセージのやり取りはそれぞれの部品の属性や振る舞いを利用するために行われるが，そのようにある部品がある部品を利用する際，部品間に**利用関係** *(Use*

Relation) が存在すると呼ぶ．利用関係の定義は部品の種類によって異なる．UML であれば依存，汎化などの関係が利用関係にあたるであろうし，バイナリ実行ファイルであれば上述のメッセージを利用関係と見なせるかもしれないが，ここではそのような議論はしない．

本論文では Java のクラス・インタフェースのソースコードを部品として扱うため，Java におけるクラス・インタフェース間の関係を利用関係として抽出する．Java プログラムのソースコード集合について，静的依存解析し依存関係をもとめる．SPARS-J では，以下の 7 種類の依存関係を利用関係と定めている．

1. 継承 : 子クラスが親クラスを利用.
2. 抽象クラス実装 : 実装クラスが抽象クラスを利用.
3. インタフェース実装 : 実装クラスがインタフェースを利用.
4. 変数宣言の型 : 変数宣言したクラスが変数型を定義するクラスを利用.
5. インスタンス生成 : インスタンス化したクラスが生成したクラスを利用.
6. フィールド参照 : 参照元クラスがフィールドを定義するクラスを利用.
7. メソッド呼出し : 呼出し元クラスがメソッドを定義するクラスを利用.

上記の内，実行時に動的に決定される利用関係については静的解析のみで一意に特定する事は不可能である．しかし，全てのソフトウェア部品について実行を行う動的依存解析のは多くのコストがかかり，実行する際に入力が必要な場合もあるので自動化することが難しい．そのため，静的依存解析を行い宣言されている部品に対して利用関係を抽出する．

そして，抽出した利用関係をもとに，**部品グラフ** *(Comopnent Graph)* を構築する．部品グラフは部品間の利用関係をグラフ上に表現したもので，各部品を頂点とし，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で表す．以下では，V を部品（頂点）の集合，E を有向辺の集合として，$G = (V, E)$ と表現する．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．

3.4 類似部品の部品群化

似たような機能を提供する部品を類似部品という．例えば，コピーした部品や，コピーして一部を変更しただけの部品は類似している．ライブラリが様々なソフトウェアから構成さ

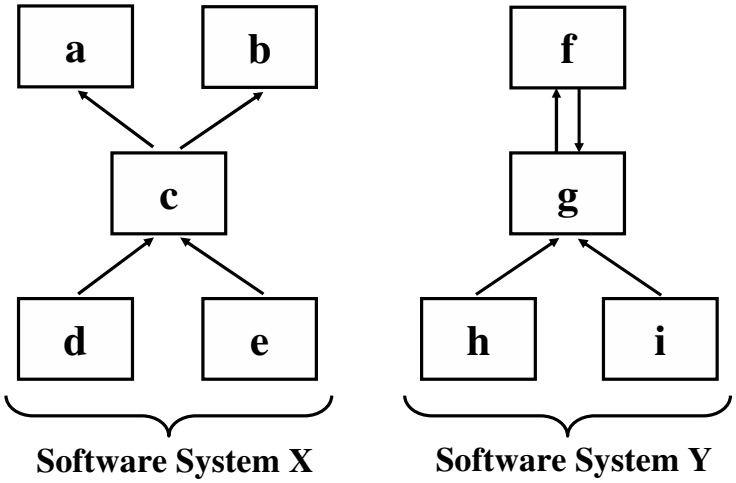

図 1: 部品グラフの例

れる場合，その中にはコピーされたりコピーされた上で大小様々な変更が加えられた部品が少なからず存在し，ライブラリの規模が大きくなるほど類似部品は多くなる．

異なるソフトウェアをまたいで類似部品が現れる場合を考える．もし図 1 で部品 c と g が類似部品であったとき，ソフトウェア X と Y は似たような機能を利用していると推測できる．そのため，利用関係は部品 c と g をまとめたクラスタに対して定義するのが妥当であると考えられる．そこで，提案手法では似た部品をクラスタ化する．このクラスタを部品群，クラスタ化することを部品群化と呼ぶ．そして，ある部品群に属する部品が他の部品群に属する部品を利用している場合には，その 2 つの部品群間に利用関係が存在するとみなしている．図 2 に部品群化によって得られた部品群グラフを示す．

SPARS-J の類似度判定は [22] で提案された手法を用いている．[22] では，Java ソースコードから静的解析によって各クラスの静的特性メトリクスを抽出する．そして，それらのメトリクス値からクラスの類似度を判定する．

- 構成トークン類似度メトリクス
  プログラムの表層的特徴の一つとして，トークンの種類別出現頻度が考えられる．Java における全トークン，すなわち予約語，演算子，記号，識別子，合計して全 96 種類のトークンについて，それぞれ種類別出現頻度をメトリクスとして構成トークン類似度を分析する．プログラムを構成しているトークンの数と種類が良く似ているプログラム同士は類似度が高いとみなす．

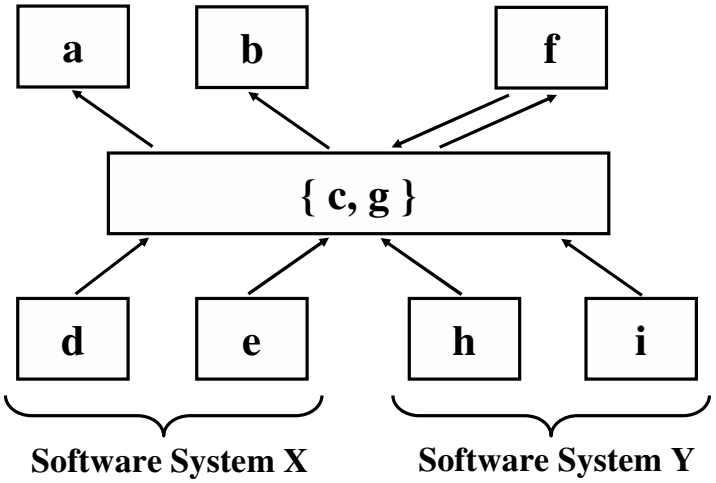

図 2: 部品群グラフの例

- 複雑性類似度メトリクス
  プログラムの構造的複雑さの特徴を表すために，表 1 に示すメトリクスを，ソースコードの複雑さの指標として記録する．さらにそれぞれのメトリクスに対して，類似判定に関する閾値を設定する．

部品 P, Q の類似判定方法について説明する．まず，各複雑性類似度メトリクスの差分が表 1 で設定した閾値以内に収まっていれば部品 P, G は複雑性類似度において類似であると判定する．

続いて，構成トークン類似度の判定を行なう．部品 P をそれを構成する 96 種類のトークンの出現頻度によって $P = \{p_1, \cdots, p_m\}$ と表現する．各 p_i は P における各トークンの出現数である．$P = \{p_1, \cdots, p_{96}\}, Q = \{q_1, \cdots, q_{96}\}$ としたとき，P，Q の類似判定は次のように行なう．P の全トークン数を $Ttotal(P)$，P と Q の各トークン数の差分の和を $\mathit{diff}\,(P, Q)$ とすると次式のようになる．

$$Ttotal(P) \equiv \sum_{k=1}^{96} p_k$$

$$\mathit{diff}\,(P, Q) \equiv \sum_{k=1}^{96} |p_k - q_k|$$

$\mathit{diff}\,(P, Q)$ が 0 なら，全く同じ種類のトークンを，全く同じ回数用いてプログラムを構成していることになるので，コピー部品であると推測できる．また，$\mathit{diff}\,(P, Q)$ が同じ 30 トー

メトリクス名	説明	閾値
cyclomatic	サイクロマチック数	0
declamethodnbr	メソッド宣言の数	1
callmethodnbr	メソッド呼出しの数	2
nestingdepth	ネストの深さ	0
NOclass	クラスの宣言数	0
NOinterface	インタフェースの宣言数	0

表 1: 複雑性類似度メトリクス

クンだとしても，全トークンが 40 トークン中の差分が 30 トークンであるのと，10000 トークン中に差分が 30 トークンあるのとでは，その性質が全く違うため，非類似度を求める際には全トークン数で正規化した値を求める．部品 P と 部品 Q の非類似度 $D(P,Q)$ を次のように定義する．

$$D(P,Q) \equiv \frac{diff(P,Q)}{min(Ttotal(P), Ttotal(Q))}$$

ここで，$D(P,Q) < 0.03$，つまり部品 P,Q の各要素の差分の合計が全体の 3% 以下であるものを構成トークン類似度において類似とみなす．この値は経験的な値であり，ライブラリの規模やライブラリ管理者の意図によって随時変更されるべきものである．

以上，複雑性類似度メトリクス，構成トークンメトリクスの類似判定のどちらにおいても類似であれば部品 P,Q は類似部品であると判定する．

3.5 順位付け手法

部品検索をキーワードによって行なうと，検索キーと合致する索引キーを持つ部品のが多数存在する場合がある．そのため，検索結果を提示する際に適当な順位で表示することが必要となる．自然言語文書検索システムで一般的に用いられている手法では，検索対象となる文書集合から各文書の特徴を表すキーワードである索引語を抽出し，索引語の集合によってその文書の内容を近似する．登録するそれぞれの文書の特徴を的確に表すように付与された，索引語の集合などからなる情報のことを索引キーと呼ぶ．検索キーは検索者の要求の内容を近似しているため，検索キーと索引キーを用いて検索キーと文書の適合度を測り，順位付けするのが一般的である．

しかし，ソフトウェア部品を対象としてを検索する場合は，検索者の要求の内容と適合する部品であるかどうかの他，その部品がよく利用されている部品かどうかに関しても考慮す

る必要がある．検索キーと適合度が高い部品よりも，よく利用されている部品を上位に提示した方が，利用例も多く，部品の再利用をスムーズに行ない易いと考えられる．そのため，利用しやすい部品かどうか定量的に評価するための指標を導入し，検索キーと部品の適合度も併せて両面を考慮した部品の順位付けを行なう必要がある．

3.5.1 Keyword Rank 法

索引キーの中には部品の内容と密接に関係したものもあれば，関係の薄いものも存在する．抽出された索引キーが部品の内容を表すうえでどれだけの重要度を持っているか測ることができれば，より望ましく順位付けされた検索結果が提供できる．このために用いられるのが索引キーの重み付けである．索引キーの重みを利用することによって，同じ索引キーを含む部品でもその索引キーの各部品中での重要度を考慮して，検索キーに対する部品の適合度を計算し部品を順序付けすることが可能になる．検索者が与えた検索キーがある部品の索引キーにヒットしたとき，その索引キーの重みの総和を検索キーと部品の適合度とする．SPARS-J における索引キーの重み付け手法として，情報検索の分野で一般的に用いられる TF-IDF 法 [21] を用いる．TF-IDF 法は任意の部品中における特定の索引キーの出現頻度 TF(*Term Frequency*)，および特定の索引キーを含む部品数の逆数 IDF(*Inverse Document Frequency*) の値を正規化して重みを算出する．TF は部品内で出現頻度の低い索引キーと高い索引キーを差別化し，部品をより特徴付ける語を選別するためのものである．IDF は部品集合内の他の部品の索引キーの分布について考慮するためのもので，ある索引キーが，どの程度その部品に特徴的に現れるのかという特定性を示す．検索キーと部品の適合度の高い順に順位付けすることを，後述する CR 法に対して ***KR* 法** ***(Keyword Rank* 法)*** と呼ぶことにする．また，測定した適合度の値を KR 値と呼ぶ．

SPARS-J では，索引キーの重みを算出する際に，そのトークンの種類によって異なる重みを与えている．表 2 に区別する索引語のトークン種類と重み付けの例を示す．どの種類の索引語にどの程度の重みを与えると良い結果が得られるかは知られていないが，経験的にクラス定義名やメソッド定義名など，その部品の概念を象徴した名前が付けられる傾向の索引キーに対して大きな重みを設定している．

データベース中の総部品数を N，部品を c，与える m 個のキーワードを $t_1, \ldots, t_m$，検索対象としたトークン種類の集合を $subject$，c 中の種類 s の索引語 t の出現回数を $tf_s(t, c)$，集合 $subject$ 中の任意のトークン種類の索引語 t を含む部品の数を $df_subject(t)$ と表す．さらに，$kw(s)$ は索引語の種類 s の重みを指し，その値は表 2 に従う．例えば，kw(クラス定義名) は 200 である．そのとき部品 c の KR 値は以下の式で求める．

$$KR = \sum_{i=1}^{m} (log_e(\sum_{s \in subject} kw(s) \cdot tf_s(t_i, c)) \cdot \frac{N}{df_subject(t_i)})$$

トークン種類	重み	トークン種類	重み
クラス定義名	200	メソッド定義名	200
インタフェース名	50	パッケージ名	50
インポートパッケージ名	30	呼出しメソッド名	10
参照フィールド変数名	10	生成したクラス名	10
変数の型名	10	参照する変数名	1
コメント (/*...*/)	30	文書コメント (/**...*/)	50
行末コメント (//...)	10	文字列リテラル	1

表 2: トークン種類とその重み

3.5.2 Component Rank 法

これまでに，利用関係からソフトウェア部品の利用関係を測定し，順位付けする手法 ***(Component Rank* 法，*CR* 法)***[15] が提案されている．CR 法では，十分な時間が経過し利用関係が収束した部品の集合に対して，各部品間に存在する利用関係に基づいてグラフおよび行列を構築し，構築された行列に対して繰り返し計算を行う事で各部品を評価する．求められる値は，開発者が利用関係に沿って参照を行うと仮定した場合の各部品の参照されやすさを表しており，よく利用される部品や，重要な部品から利用される部品の順位は高くなる．CR 法によって測定した適合度の値を CR 値と呼ぶ．以下，CR 法について説明する．

重みの定義

CR 法では，図 1 部品グラフ $G = (V, E)$ 上の個々の辺および頂点に対して重みを計算し，対応する頂点の重みをもとに各部品の重要度を評価する．頂点の重みと，辺の重みを次のように定義する．

定義 1 (頂点の重みの和) 部品グラフ G 上の各頂点 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．すなわち，

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

定義 2 (辺の重み) 頂点 v_i から v_j への辺 e_{ij} に関する辺の重み $w'(e_{ij})$ は，

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

図 3 (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，$0 \leq d_{ij} \leq 1$ 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．

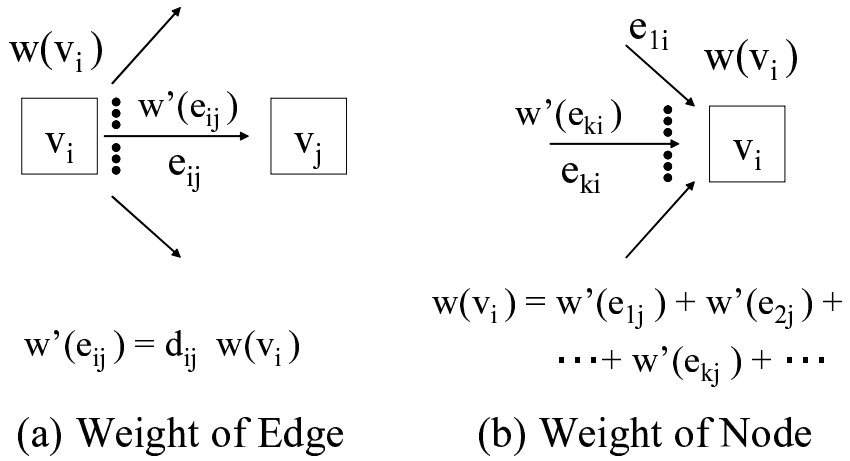

図 3: 重みの定義

この配分率 d_{ij} は，次に示す頂点の重みの定義において，有向辺の終点となる頂点の重みの決定に利用される．図 3 (b) は次の定義を図示したものである．

定義 3 (頂点の重み) *IN(v_i)* を v_i を終点とする有向辺の集合とする．この時，頂点 v_i の重みは v_i が終点となる有向辺 e_{ki} の重みの総和とする．つまり，

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

重みの計算

定義に基づいて，頂点 $w(v_i)$ に対して次の方程式が生成できる．

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

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，各頂点の重みをベクトル W として次のように表現し，

$$W = \begin{pmatrix} w(v_1) \\ w(v_2) \\ \vdots \\ w(v_n) \end{pmatrix}$$

配分率を次の行列 D として表現する事で,

$$D = \begin{pmatrix} d_{11} & d_{12} & ... & d_{1n} \\ d_{21} & d_{22} & ... & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & ... & d_{nn} \end{pmatrix}$$

$n(= |V|)$ 個の連立方程式を次のように表す事ができる．ここで 行列 D^t は D の転置行列を表す.

$$W = D^t W \tag{1}$$

定義 1 から，行列 D^t は推移確率行列の性質を満たしているため，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 [6] でモデル化し表現する．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するとみなす．ここで，式 (1) 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，開発者によって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．この値を重要度とすることで，ただ単に利用数が多い部品だけでなく，利用数が多い部品が利用している部品も重要であると評価することが出来る．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．図 4 は，与えられたグラフにおいて各頂点の重み（重要度）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．

部品重要度計算に関する補正

部品の重要度の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 5(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.

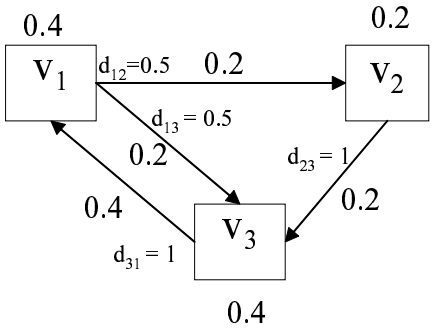

図 4: 重みの計算例

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 5 (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．ここで p は実際の辺と擬似辺の重みの配分比率を指す.

定義 4 (修正配分率) 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．

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

この補正は，前節で説明したマルコフ連鎖を用いた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．

類似部品化の効果

それぞれの類似部品へ指されていた辺が，部品群化を行う事で一つの類似部品群への辺とみなされる．そのため，コピーして再利用される回数が多いほど，その部品群は多くの部品から利用されることになる．このため，コピーされたという利用関係を重要度に反映させる事ができる．この時，G の部品群グラフ（*clustered component graph*）$G' = (V', E')$ を次のように定義する．

定義 5 (部品群グラフ $G' = (V', E')$) V の商集合からなる集合 V' を G' の頂点集合とする．また v_i，v_j が属する V' 中の集合をそれぞれ v'_i，v'_j としたときに，$E' = \{(v'_i, v'_j) | (v_i, v_j) \in E\}$ を G' の辺集合とする．

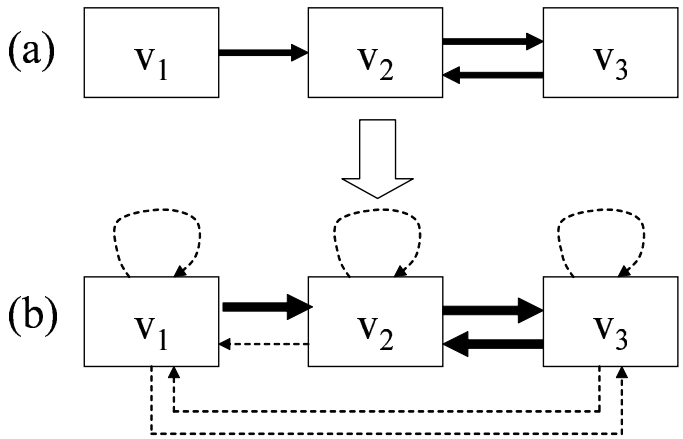

図 5: 部品重要度計算に関する補正（擬似辺の追加）

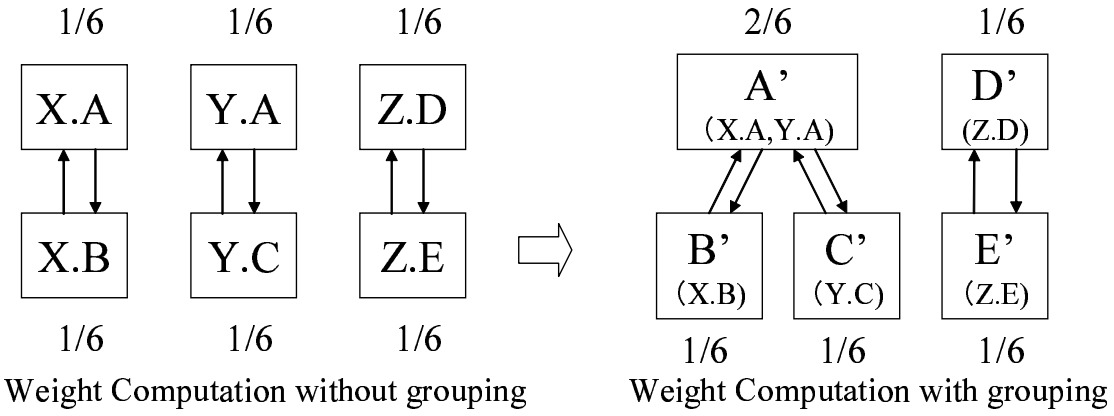

図 6: 部品群化の効果

図 6 は部品群化を行った際に，類似部品の重要度がどのように変わるかを例で示したものである．3 つのシステムのうち，2 つのシステムには $X.A$ および $Y.A$ という同一の部品が存在する．部品がコピーされた場合は，部品が属する名前空間（パッケージ名）が異なる，または部品名が異なるなどの理由から $X.A$，$Y.A$ は別々の部品として扱われてしまう．部品群化を行わないと，コピーされた部品は別々の部品として評価されるため，図 6 の場合すべての部品の重要度が等しくなる．そこで，類似部品群化によって部品 $X.A$ および $Y.A$ へのそれぞれの有向辺が一つの部品群 A' への有向辺に統合され，部品 A の重要度が他の部品よりも高くなる．以降の実際の重要度計算では，部品グラフではなく部品群グラフを対象としており，部品群グラフに対して修正配分率に関する補正を行った上で部品群における重要度を計算している．

3.5.3 順位の統合

KR 法によって得られた順位と CR 法によって得られた順位を統合する．順位の統合はメタ検索システムで利用される手法であり，経験的な (heuristic) なものとして知られている．メタ検索システムとは，複数の検索システムで同一の検索対象集合の検索を行ない，検索結果を統合するシステムである．検索システムそれぞれが持つ検索アルゴリズムの特徴を考慮した検索が可能であり，検索結果の再現率と適合率を改善することができる．順位の統合は，広告目的で作成者が検索結果で意図的に高い順位を得ようと作成した Web ページなど，不相応に高い順位を得ているもの (spam と呼ばれる) の順位を下げ，情報の健全性を守ることができるとされている [10]．最も単純な順位の統合手法に Borda の手法 [7] がある．計算量が少なく，線形時間で解けることが知られている．Borda の手法では，各順位に対して評価点を割り当て，その合計値をもとに最終的に統合された順位を求める．

SPARS-J では，検索結果を KR 法，CR 法で評価し，それぞれの順位を部品の表価点とする．そして評価点を足した合計点によって検索結果を昇順に並べ直し，検索結果として提供する．例を用いて Borda の手法を説明する．表 3 は KR 法，CR 法，および Borda の手法で統合された CR と KR の統合結果 (以降 CR+KR と表記する) を，表 4 はそれに対応した各部品の評価点を表す．表中の A，···，J はそれぞれ部品である．例えば部品 A は KR で 1 位，CR で 4 位であり，評価点は 5 点となる．よって部品 A は順位を統合した結果 2 位となる．

	KR	CR	CR+KR
1 位	A	C	C
2 位	E	F	A
3 位	C	G	E
4 位	I	A	F (3 位)
5 位	J	B	B
6 位	B	D	I (5 位)
7 位	H	I	G
8 位	F	E	J
9 位	G	H	D
10 位	D	J	H (9 位)

表 3: Borda の手法

	KR	CR	CR+KR
A	1	4	5
B	6	5	11
C	3	1	4
D	10	6	16
E	2	8	10
F	8	2	10
G	9	3	12
H	7	9	16
I	4	7	11
J	5	10	15

表 4: 評価点

4 システム構成

本節では構築した Java ソフトウェア部品検索システム SPARS-J について説明する．

4.1 システムの実装

記述言語として C 言語および C++言語を，データベースに BerkeleyDB[28] を，字句解析に [12]，構文解析に [11] を用いてシステムの実装を行なった．さらに日本語検索対応のために日本語わかち書きソフトウェア KAKASI[19]，および日本語表示のために gettext[13] を用いた．ソースコードの規模は，データベース約 9100 行，部品解析部約 13100 行，部品検索部・データ表示部約 11000 行，合計約 33200 行であった．

4.2 システムの機能

本システムが持つ機能は以下の通りである．

- キーワード検索
  ユーザが要求したキーワードを検索語として部品の検索を行なう．日本語を検索語とすることも可能である．’@’ および ’.’，’$’ には特殊な意味がある．’@’ ではじまる検索語はパッケージ指定であり，指定したパッケージのみを対象に検索を行なう．他に検索語を入力していなければ，指定パッケージをパッケージブラウザで表示する．’.’ または’$’ を含む検索語は，部品の完全限定名として解釈され，その部品を検索する．’$’ は内部クラスとして解釈される．この時，他に入力された検索語は無視される．
  - 検索対象とするトークン種類の指定
    キーワード検索の際に，検索対象とするトークンの種類の指定を行なうことができる．例えば，クラス定義名のみを対象に検索を行なうことが可能である．
  - 検索結果の順位付け手法の指定
    検索結果の順位付けを，CR，KR，および CR+KR のいずれを用いるか利用者が選択可能である．利用者の目的に合わせた検索を行なうことができる．
- 類似部品群の提示
  類似していると判定された部品の一覧が得られる．
- 利用関係の提示
  部品間の利用関係を提示することで，部品の使用方法に関する情報を得られる．

- パッケージブラウザ
  パッケージ階層構造をハイパーリンクで表現しており，パッケージ内に含まれる部品の一覧を得られる．

- メソッド一覧表示
  部品内で定義されているメソッドの一覧が表示される．メソッド名はメソッド定義行へのリンクになっており，辿ると定義行へジャンプできる．

- キーワードハイライト
  ヒットしたキーワード部分がハイライトされて表示される．ハイライトをクリックすることで，次のハイライト部分へジャンプできる．

- アーカイブダウンロード
  部品が含まれる jar ファイルなどのアーカイブをダウンロードして使用することができる．

4.3 システムの構成

本システムは以下のような部分から構成される．構成を図 7 に示す.

- ライブラリ
  収集した Java ソースファイルが蓄積されているライブラリ．SPARS-J 上では部品の完全限定名で部品の分類を行なうので，ディレクトリ階層をパッケージに合わせる必要は特にない．検索結果の部品詳細表示では，ライブラリ中の該当部品が存在するファイルを参照し，表示する．

- データベース (File DB, Component DB, Relation DB, Word DB)
  ライブラリ中のファイル情報を管理する File DB，登録された部品に関する情報を管理する Component DB，部品間の利用関係を管理する Relation DB，索引の管理をする Word DB がある．各 DB は，ID からファイル名や部品名，ファイル名や部品名から ID を対応付けるために正引き，逆引き索引を持つ.

  – File DB
    ライブラリ中の Java ファイルのファイルパス，ファイル ID，各ファイルに存在する部品の ID，更新時間などの情報が格納されている．さらに，将来の拡張 (他言語対応など) に備えてファイルの記述言語情報も格納している．

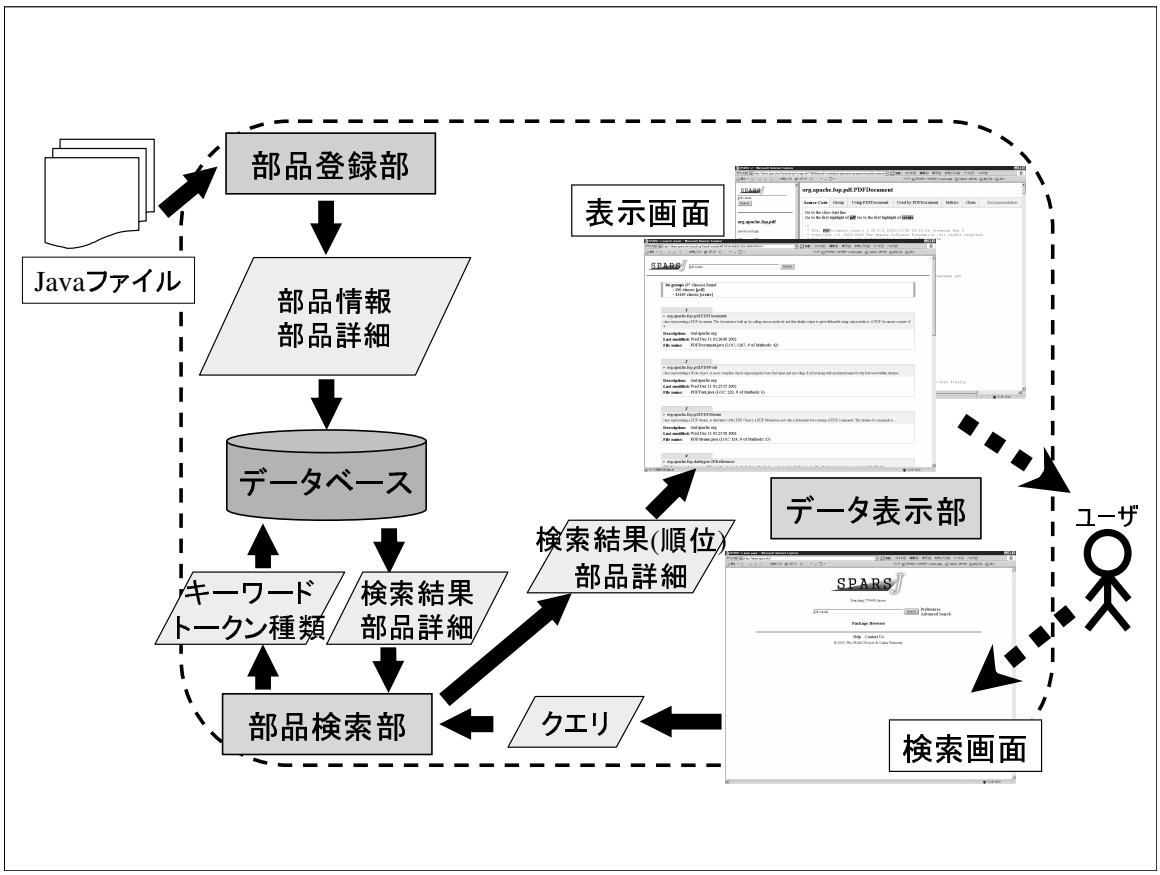

図 7: SPARS-J の構成

- Component DB
  部品名，部品 ID，部品が存在するファイル ID，ファイル中の部品の場所など部品とライブラリ中のファイルを File DB を介して対応付けるための情報の他に，各部品中で定義されたメソッドやフィールド名，類似判定のためのメトリクス，各部品の類似度，部品が属する部品群 ID，CR 値，部品中に現れた索引キー ID や出現頻度，場所など部品に関する全情報が格納されている．
- Relation DB
  部品間の利用関係と，その利用関係の種類が格納されている．その他，登録過程でまだ登録されていない部品への利用関係があった場合，利用する部品名を未解決名として格納し，部品が登録され次第利用関係を追加するための未解決名情報が格納されている．
- Word DB
  登録された部品中で出現した全索引キーとその ID を格納する DB である．検索時の大文字小文字の区別のために区別あり索引情報と，区別なし索引情報を格納している．

- 部品登録部 (Register, Ranker, Sorter)
  部品登録部はライブラリのファイルを解析して，各種情報をデータベースに格納する Register，利用実績に基づいて部品群の CR 値を計測する Ranker，検索時の高速化のために CR 値によって部品群を降順にソートし，各部品の索引キーとその出現頻度から索引キーの重みを計測する Sorter，以上 3 つのモジュールから構成される．

  - Register
    利用者が指定した Java ソースファイルを解析して，部品の切り出しの他，索引，メトリクス，利用関係などの抽出，さらに類似度の測定と部品群化を行なう．ファイル，部品，部品群，索引キーへの ID を割り振り，正引き，逆引き索引も作成する．サブモジュールとして，ソースファイル解析部 Parser と部品群化部 Cluster を持つ．
  - Ranker
    Register が抽出した利用関係から，各部品の CR 値を計測する．
  - Sorter
    SPARS-J は部品検索時に CR と KR の順位の統合を行なうため，データベース構築時にそのオーバーヘッドを軽減しておく必要がある．そのため，CR 値によって部品群を降順にソートしておき部品検索時には既に CR でソート済みのリスト

を取得できるようにする．また，KR 値は部品検索時に計測するが，Sorter 索引キーと出現頻度から索引キーの重みを計測しておくことで，検索時に KR 値の計測を高速に行なうことができる．

- 部品検索部 (Searcher)
  Searcher はクエリの解析，部品の検索を行なうモジュールである．利用者が指定したクエリを解析し，得られたキーワードとトークン種類の組を検索キーとしてデータベースの検索を行なう．得られた結果を順位付けし，各部品の KR 値の計測，CR と KR の順位の統合を行なう．検索結果と併せて，部品の詳細情報として，ソースコードや利用関係，メソッド一覧などを利用者に対して提供する．

- データ表示部 (Web インタフェース)
  データ表示は CGI であり，Web インタフェースを利用している．ユーザから受け付けたクエリを部品検索部に引き渡す．また，Searcher から受け取った検索結果をユーザに対して提供する．検索結果と併せて，部品の詳細情報として，ソースコードや利用関係，メソッド一覧などをユーザに対して提供する．

4.4 データベースの説明

4.4.1 File DB

File DB はライブラリ中の Java ファイルと Component DB 中の部品の対応をとるための情報を格納する．解析対象の Java ファイルは一意なファイル ID を与えられ，ファイル中に存在する部品に割り振られた部品 ID と対応付けられる．同名のファイルが存在しなければ新規ファイルとして格納し，一意なファイル ID を与える．存在していればファイルの更新とみなし，File DB のファイル ID や含まれる部品 ID の情報を更新する．

4.4.2 Component DB

Component DB には Java ファイルを解析することによって得られた，各部品に関する情報が格納される．主に以下のような情報がある．

- 部品の完全限定名と部品 ID
- 部品のファイル中の出現位置
- 部品がインポートしたパッケージ
- 部品中で定義されたコンストラクタ，メソッドと行番号

- 部品中で定義されたフィールド変数と行番号
- 部品の特徴メトリクス値
- 部品が所属する部品群 ID
- 部品の CR 値

4.4.3 Relation DB

各部品間の利用関係を格納する DB である．利用関係は，利用関係の種類と関係を結ぶ部品 ID の組から構成される．さらに，未解決部品のリストも持つ．現在登録している部品がまだ登録されていない部品を利用した場合に，一時的に未解決部品として保存しておき，その部品が登録され次第そのリストから未解決部品を削除して，利用関係として登録する．

4.4.4 Word DB

Word DB には索引キーの情報が格納される．索引キーに対して一意な ID が割り振られており，同じ索引語でも異なったトークン種類であればその ID も異なる．また，検索時の大文字小文字区別のために索引語を全て小文字に直したものと，そのままのものをそれぞれ格納している．さらに，全ての索引キー ID はどの部品の何トークン目に出現したのか，といった情報を持っている．また，部品検索部における KR 法の高速化のために，あらかじめ索引キーの重みを Word DB に格納している．

4.5 部品登録部の説明

部品登録部は，Java ファイルを解析することで，検索時に必要な情報を抽出し，それらの情報を各種 DB に格納する．

4.5.1 Register

部品登録部に入力された Java ファイル群は Register で解析される．Register は Java ソースコードの解析を行ない部品や索引キー，利用関係，部品の特徴メトリクスを抽出する Parser と，特徴メトリクスから部品の類似度を判定し部品群化を行なう Cluster という 2 つのサブモジュールを持つ．

Parser はファイル中からクラスまたはインタフェースを見つけ出し部品として切り出し一意な部品 ID を与える．部品は記述されているファイル ID やファイル中の行番号をファイル位置情報として Component DB に格納する．

さらに，各部品の索引付けを行なう．検索時に利用するために各部品中に含まれる識別子やコメント中の語句を索引語として抽出し，トークン種類毎に出現頻度をカウントする．それらを索引キーとして Component DB 格納する．

また，部品間の利用関係の抽出を行う．抽出した利用関係は Relation DB に格納する．

その他に特徴メトリクスの計測を行い，後述する Cluster によって部品群化を行なう．計測したメトリクス値は Component DB に格納する．部品の特徴メトリクスを計測し比較することで，類似部品をまとめた部品群を作成することが可能になる．すべてのメトリクスは構文解析時に計測可能であるため，Parser で特徴メトリクスの計測だけを行う．

4.5.2 Ranker

Relation DB に格納された部品群間の利用関係から CR 値の計算を行う．CR 値は部品群 ID と対にして Component DB に格納する．

4.5.3 Sorter

Component DB から CR 値を参照して部品群の順位付けを行なう．また，後のキーワード出現頻度に基づく順位づけのために各索引キーの重みを算出しておく．

4.5.4 CR 法の配分比率について

CR 法では，パラメータとして辺と擬似辺の重みの配分比率 p が存在する．配分比率 p に関して p の値を変えて順位の変動を検証したところ，$p = 0$（補正のみしかないため全ての部品が同順位になる）と $p = 1.0$（補正無し）の時以外は，値を変更しても結果として得られる順位に違いはほとんど見られなかった．また，$p = 1.0$（補正なし）の場合は補正を行った場合と比べて 10 倍以上の計算時間を必要とした．このことから，p は順位の決定に影響を与えるパラメーターではなく，計算の収束にのみ必要なパラメータで，p はどの値でも問題ないことがわかった．そこで，現在のところ最初に採用していた値である $p = 0.85$ を利用している．

4.6 部品検索部

部品検索部は，検索者によって与えられたクエリを解析して検索キーを取り出し一致する部品を検索する．そして，検索結果の部品を評価値が高い順に順位付けしてデータ表示部に渡す．クエリには検索キー以外にも以下の項目を検索条件として指定することができる．

- 検索対象とするパッケージ名

- 検索対象とするトークン種類
- 順位付けの方法
- 検索方法 (AND-OR 検索) や，検索キーの大文字小文字の区別の有無

部品検索部は，検索条件に応じて Component DB の適切な部分のみを検索し，ヒットした部品の部品 ID のリストを作成する．ヒットしたリスト中の部品 ID を持つ各部品について，索引キー中の索引語の重みの総和を検索キーと部品の適合度とし，KR 法を用いて順位付けする．

その後，求めた KR 法による評価値と，Ranker で求めた CR 法による評価値をもとに検索条件に指定した方法でリストを再度順位付けし，結果をデータ表示部に渡す．指定できる順位付けの方法として，KR のみ，CR のみ，および Borda の手法で統合した CR+CR を選択できる．デフォルトは CR+KR である．

4.7 データ表示部

データ表示部は部品検索部と検索者を結ぶ Web インタフェースである．SPARS-J は Microsoft の Internet Explorer などの Web インタフェースを通して検索機能を提供する．検索者が指定した検索条件を部品検索部に渡し，部品検索部が返してくる検索結果の部品リストを受け取って表示する．

検索結果表示では，ヒットした部品を部品リストの順位で表示する．検索結果表示の様子を図 10 に示す．各部品の行数や部品中で定義しているメソッド数，部品の利用実績を併せて表示することで，再利用に有益な情報を検索者に提供する．詳細を知りたい部品を選択することで，部品の詳細データ表示が可能である．

部品の詳細データ表示では，部品のソースコードや，同一部品群に属する部品，パッケージブラウザによるクラス階層構造，部品間の利用関係，などといった各部品の詳細情報を閲覧することができる．

図 8: 検索画面

図 9: Advanced Search

図 10: 検索結果表示

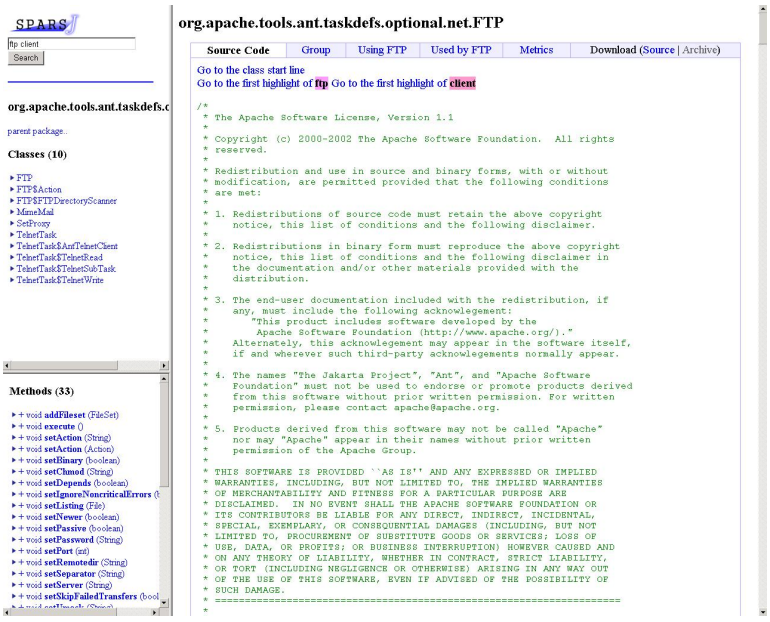

図 11: 部品詳細表示

5 性能評価

本節では，システムの性能評価を行なう．データベース構築時間，構築したデータベースの空間コスト，そしてキーワードの検索時間について評価した．

5.1 評価概要

評価の環境は表 5 で，5 つのライブラリ (JDK13[29]，SourceForge[31]，java.about.com[1]，xml.apache.org[2]，jakarta.apache.org[3]) に対してデータベース構築時間，構築したデータベースのサイズ，それぞれのデータベースに対する検索時間を測定し，性能評価を行なう．表 6 に各ライブラリの情報を示す．

OS	FreeBSD 5.2
CPU	Intel(R) Xeon 2.80GHz (Dual)
メモリ	2GB

表 5: 評価環境

ライブラリ	ファイル数 (サイズ : MB)	部品数	部品群数	索引キー数
JDK13	3593 (35)	5407	5370	574048
SourceForge	6158 (67)	7594	966	132583
java.about.com	9561 (93)	12355	8995	306551
xml.apache.org	13696 (130)	15634	9033	745663
jakarta.apache.org	18384 (192)	20463	13839	886574

表 6: 評価対象ライブラリ

5.2 データベース構築の時間

表 7 にデータベース構築の時間を示す．単位は全て秒である．JDK13 は java.about.com や SourceForge よりも部品数が少ないにも関わらず構築時間は大きくなっている．java.about.com で部品数が 2.5 倍近くあるにもかかわらず Register 時間が JDK13 とほぼ一緒であることを考えると，その原因は JDK13 の索引キー数が多いためと考えられる．SourceForge は類似部品が多く，部品群化に時間がかかっている．また，xml.apache.org の Register 時間が極端に大きくなっているのは，索引キー数が多くメモリのキャッシュが尽きたためである．Ranker

は部品数が 4 倍近くなっても実行時間は 2 倍で 2 秒程度であり，部品の増加にも十分対応できる．Sorter は Register と同様に索引キー数に依存するかたちで増加している．

以上から，データベース構築のボトルネックとなっているのが Register で行う索引付けであり，索引キーの増加に従いデータベース構築時間も大きくなる．

ライブラリ	Register	Ranker	Sorter	合計
JDK13	832	0.7	26	858.7
SourceForge	913	0.7	20	933.7
java.about.com	847	0.9	19	866.9
xml.apache.org	6388	1.5	80	6469.5
jakarta.apache.org	15976	2.1	115	16093.1

表 7: データベース構築時間 (単位 : 秒)

5.3 データベースのサイズ

表 8 に構築したデータベースのサイズを示す．さらに，図 12〜図 16 にそれぞれのデータベースのサイズの割合を円グラフで示す．

データベースのサイズの割合はほぼ似通っている．JDK13 の Relation DB が小さいのは未解決名が少ないためであった．他のライブラリでは JDK への参照は未解決になるため，JDK13 に比べ Relation DB の割合が大きくなっている．データベース構築時間と同様に，データベースのサイズも索引キー数に依存している．

ライブラリ	部品数	索引キー数	DB サイズ (MB)
JDK13	5408	574048	288
SourceForge	7595	132583	288
java.about.com	12356	306551	252
xml.apache.org	15635	745663	800
jakarta.apache.org	20464	886574	1000

表 8: データベースのサイズ

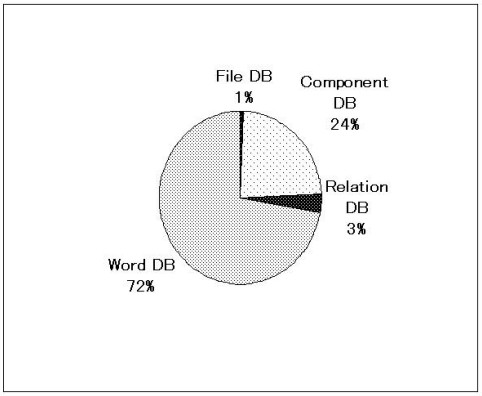

図 12: JDK13

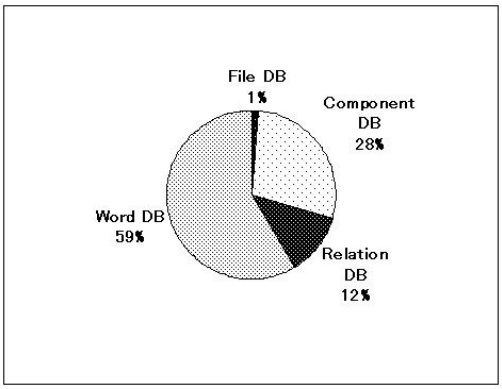

図 13: SourceForge

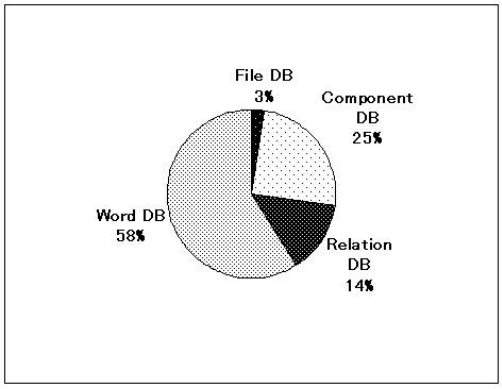

図 14: java.about.com

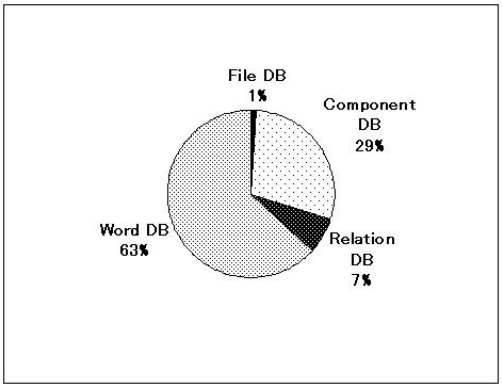

図 15: xml.apache.org

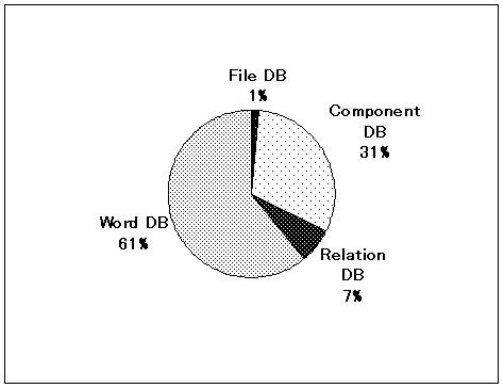

図 16: jakarta.apache.org

5.4 検索時間

表 9 に 300 クエリ検索時の合計時間と，それを一回あたりの平均時間に直したものを表示する．検索時間は部品数に依存しているようである．これは部品検索部ではヒットした部品のソートを行うためだと考えられる．部品数が多いほどヒットする部品も多くなりソート時間がかかる．

ライブラリ	部品数	索引キー数	合計時間 (秒)	平均時間 (秒)
JDK13	5408	574048	4.140	0.0138
SourceForge	7595	132583	4.787	0.0160
java.about.com	12356	306551	5.964	0.0201
xml.apache.org	15635	745663	14.395	0.0480
jakarta.apache.org	20464	886574	19.514	0.0650

表 9: 検索時間

5.5 結果に関する考察

以上，データベース構築および部品検索の時間コストと，データベースの空間コストについて調べた．部品検索の時間コストは部品数が 2 万近くても 1 回あたりの平均は高々0.07 秒であり問題ないと思われる．順位の統合はクイックソートを行なうため $O(nlogn)$ で，他の複雑な順位の統合手法だと精度が向上する可能性があるが，検索時間は急激に増加すると思われる．

データベースの時間コスト，空間コストについては索引キー数に依存しており，少なくとも $O(nlogn)$ である．評価に用いたライブラリの規模を考えれば大きな問題ではないが，ライブラリが大きくなるほど索引キー数は増加するため，将来にはなんらかの改善が必要であると思われる．

6 まとめ

本研究では，Java ソフトウェア部品検索システム SPARS-J を構築した．SPARS-J を用いることで，大量の Java プログラムソースコードから，要求を満たす部品を検索することが可能である．また，部品間の利用関係や詳細情報を提供することで，プログラム理解・保守といった広範な利用も行なうことができる．さらに，各種ライブラリに対して SPARS-J の性能評価を行ない，実用性を確認した．今後の課題として以下が挙げられる．

- 他のソフトウェア部品への対応
  UML や実行形式ファイルなどを対象に，本システムを適用することが考えられる．
- 順位付け手法に関する検討
  KR 法の索引語の種類による重みの調整を行なうことで，より良い順位が得られる可能性がある．順位の統合についても調整を行なう必要がある．
- ライセンス管理
  現状は登録時に部品に関するコメントを入力することができるため，入手先などの特定は可能だが，ライセンスのないユーザに対しては部品を提供しないなどより安全性を考慮した管理手法を導入できればと考えている．

謝辞

本研究において, 常に適切な御指導および御助言を頂きました 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.

本研究において, 常に適切な御指導および御助言を頂きました 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 楠本真二助教授に深く感謝致します.

本研究において, 常に適切な御指導および御助言を頂きました 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 松下 誠助手に深く感謝致します.

本研究において, 常に適切な御指導および御助言を頂きました 独立行政法人 科学技術振興機構 計算科学技術研究員 山本哲男 氏に深く感謝致します.

本研究において, 常に適切な御指導および御助言を頂きました 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 産学官連携研究員 横森励士 氏に深く感謝致します.

最後に, その他様々な御指導, 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いたします.

参考文献

[1] About, Inc., “Focus on Java”, http://java.about.com/.

[2] The Apache Software Foundation, “The Apache XML Project”, http://xml.apache.org/.

[3] The Apache Software Foundation, “The Apache Jakarta Project”, http://jakarta.apache.org/.

[4] M. H. Aviram: “Code-centric search tool strives to reduce Java development time”, *Java World*, June (1998).

[5] 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).

[6] G. Blom, L. Holst and D. Sandell: “Problems and Snapshots from the World of Probability”, *Springer Verlag*, (1994).

[7] J. C. Borda: “M’emoire sur les ’elections au scrutin”, *Histoire de l’Acad’emie Royale des Sciences*, (1781).

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

[9] C. Braun: “NATO Standard for the Development of Reusable Software Components”, *NATO Communications and Information Systems Agency*, (1992).

[10] C. Dwork, R. Kumar, M. Naor, D. Sivakumar: “Rank Aggregation Methods for the Web”, *Proc. of WWW10*, pp. 613-622 (2001).

[11] Free Software Foundation, Inc., “Bison”, http://www.gnu.org/software/bison/.

[12] Free Software Foundation, Inc., “Flex”, http://www.gnu.org/software/flex/.

[13] Free Software Foundation, Inc., “gettext”, http://www.gnu.org/software/gettext/.

[14] Google, Inc., “Google”, http://www.google.com/.

[15] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, S. Kusumoto: “Component Rank: Relative Significance Rank for Software Component Search”, *Proc. of ICSE25*, pp. 14-24 (2003).

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

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

[18] J. Gosling, B. Joy, G. Steele, and G. Bracha: “The Java Language Specification”, *Addison-Wesley* (2000).

[19] KAKASI Project, “KAKASI”, http://kakasi.namazu.org/.

[20] 亀井, 門田, 松本: “WWW を対象としたソフトウェア検索エンジンの構築”, 電子情報通信学会技術研究報告, SS2002-47, Vol. 102, No. 617, pp. 59-64, (2003).

[21] 北, 津田, 獅々堀: “情報検索アルゴリズム”, 共立出版, (2002).

[22] 小堀, 山本, 松下, 井上: “類似度メトリクスを用いた Java ソースコード間類似度測定ツールの試作”, 電子情報通信学会技術研究報告, SS2003-2, Vol. 103, No. 102, pp. 7-12, (2003).

[23] K. Maruyama, K. Shima: “A Mechanizm for Automatically and Dynamically Changing Software Components”, Symposium on Software Reusability, ACM Software Engineering Notes, Vol. 22, No. 3, pp. 169-180, (1997).

[24] Namazu Project, “Namazu”, http://www.namazu.org/.

[25] 庭山, 松尾, 萩原, 金田: “セマンティック Web を利用したソフトウェア検索システムの提案”, 電子情報通信学会技術研究報告, SS2002-57, Vol. 102, No. 704, pp. 27-31, (2003).

[26] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl: “GroupLens: an open architecture for collaborative filtering of netnews”, *Proc. of the 1994 CSCW*, pp. 175-186 (1994).

[27] R. C. Seacord, S. A. Hissam, K. C. Wallnau: “Agora: A Search Engine for Software Components”, *IEEE Internet Computing*, Vol. 2, No. 6, pp. 62-70 (1998).

[28] Sleepycat Software, Inc., “BerkeleyDB”, http://www.sleepycat.com/.

[29] Sun Microsystems, Inc., “Java2 Software Development Kit, Standard Edition 1.3.0”, http://java.sun.com/.

[30] H. Washizaki, Y. Fukazawa: “A Component Extraction-based Retrieval System for OO Program”, IPSJ/SIGSE OO 2003 Symposium (2003).

[31] VA Linux Systems, Inc., “SourceForge”, http://sourceforge.net/.