論　文

イディオム検索のための関数呼出依存グラフのクラスタリング手法

渥美　紀寿[†a)]　　山本晋一郎[††]　　阿草　清滋[†††]

Clustering Method of Function Call Dependency Graphs for Idioms Retrieval on Software

Noritoshi ATSUMI[†a)], Shinichirou YAMAMOTO[††], and Kiyoshi AGUSA[†††]

あらまし　プログラム開発者はライブラリを利用する際，マニュアルや実例を参照し，その利用法を調べる．マニュアルには利用例が示されているが，典型的な例のみであり，すべての組合せが示されていない．また，既存のソフトウェアから文字列検索により検索する場合では，多くの検索結果が得られ，必要な情報を選択することが困難である．我々はライブラリの利用において他のライブラリとの組合せが特に重要であると考え，依存関係によるライブラリの組合せを表現する関数呼出依存グラフを提案した [14]．本論文では，既存のソフトウェアから抽出した関数呼出依存グラフ群からライブラリの組合せを検索した結果を分類と順位付けを行う手法を提案し，実装した．本システムを用いて実際に `FreeBSD` の/usr/src/usr.sbin にある平均 1,500 行のソフトウェア 152 個を対象に利用頻度の高いライブラリ関数に対して関数呼出依存グラフの分類及び順位付けを行った．その結果，実際によく利用されているライブラリ関数の組合せが得られた．また，平均約 40 個の関数呼出依存グラフから平均 5 個のカテゴリーに分類することができた．

キーワード　ソフトウェア再利用，ソフトウェア検索，イディオム，類似度，クラスタリング

1. まえがき

ライブラリは汎用的なソフトウェア部品であり，様々なソフトウェアで利用されている．ライブラリには一般にマニュアルが存在し，ライブラリ利用者はマニュアルを参照して，利用法を調べる．マニュアルには引数に渡す値や返り値に関する情報が自然言語で記述されている．また，典型的なライブラリの利用法が示されている．しかし，自然言語による説明では実際にどのように利用するか分からない．そのために，利用例が示されているが，様々な使い方が考えられるため，すべてを示すことができない．

自然言語では，辞書を引くことで単語のもつ様々な意味を知ることができる．しかし，我々は単語の意味を知っているだけでは文章を書くことはできない．単語には複数の意味が存在し，状況や他の単語との組合せに合わせて適切な単語を選択しなければならない．また，イディオムのような慣用表現が存在する．このことと同じようにプログラミング言語において個々のライブラリの使い方を知っていてもプログラムを作成することはできない．他のライブラリとの組合せにより，そのライブラリの役割が変化するため，組合せ方を知らなければならない．これらの知識は経験により習得しているのが現状である．

自然言語においては実際の利用例を示した用例辞典が出版されている [7], [10]．また，自然言語の翻訳に関する研究では，用例を用いて翻訳を行う研究が数多く行われている [8], [12], [13]．プログラミング言語において用例を調べることはプログラムを開発する上で重要である．ソフトウェア開発者はライブラリの使い方をマニュアルを調べるだけでなく，実際にライブラ

† 南山大学数理情報学部情報通信学科，名古屋市
Department of Information and Telecommunication Engineering, Faculty of Mathematical Sciences and Information Engineering, Nanzan University, Nagoya-shi, 466–8673 Japan
†† 愛知県立大学情報科学部情報システム学科，愛知県
Department of Information Systems, Faculty of Information Science and Technology, Aichi Prefectural University, Aichi-ken, 480–1198 Japan
††† 名古屋大学情報科学研究科情報システム学専攻，名古屋市
Department of Information Engineering, Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8603 Japan
a) E-mail: natsumi@it.nanzan-u.ac.jp

リを利用しているソフトウェアから grep などの文字列検索ツールを用いて用例を検索する．

Web ページの検索エンジンとして Google がよく利用されているが，Google では各 Web ページの質をリンクされている数によって評価し，良質なページを上位にランク付けている [11]．これは膨大な検索結果すべてを参照することが大変であり，一般的に上位の数件から数十件しか参照しないため，順位付けを行うことによってユーザが求めていると思われる良質なページを上位に表示させるためである．このような順位付けは Web ページ検索だけではなくソフトウェアの検索でも重要である．

ソースコードから再利用可能なコード断片を検索する研究は数多く行われている [2], [3], [5]．これらの検索手法では，膨大な数の結果が得られる．再利用する際には開発中のソフトウェアに適したコード断片を膨大な検索結果から選択することは困難である．我々は，ライブラリの組合せを関数呼出依存グラフとして抽出し，得られた関数呼出依存グラフに基づいてライブラリの組合せを検索する手法を提案した [9]．本論文では，検索結果の分類と順位付けを行う手法を提案する．本手法は，既存のソフトウェアのソースコードに現れるライブラリの組合せの出現頻度に基づき，検索結果の分類と順位付けを行う新しい手法である．我々の知る限り，コード検索技術において，ライブラリの組合せに着目した検索を行う研究は行われていない．

以下 **2.** ではイディオムを関数呼出依存グラフに基づいて定義する．**3.**，**4.** ではイディオムを検索するために用いる関数呼出依存グラフの分類手法と順位付け手法について述べる，**5.** で作成したイディオム検索システムについて述べる．**6.** では本論文のまとめと今後の課題について述べる．

2. 関数呼出依存グラフとイディオム

2.1 ライブラリ関数の組合せ

高級言語では，共通の語彙としてライブラリを提供している．ライブラリ開発者はできるだけ汎用的に利用できるようにライブラリを作成する．そのため，ライブラリは様々なプログラムで利用される．ライブラリは単独で利用されることはほとんどなく，多くの場合，他のライブラリと組み合わせることによって，要求される機能を実現する．我々はこれらの組合せを関数呼出依存グラフ（FCDG）[14] として定義した．

例えば，次に示す手続きは多くのネットワークプログラムで行われているサーバ側の手続きである．

（1） 通信するために，ライブラリ関数 `socket` を使って *socket* を作成する．

（2） ライブラリ関数 `bind` を使って *socket* にローカルプロトコルアドレスを割り当てる．

（3） ライブラリ関数 `listen` を使って *socket* 上の接続を待つ．

（4） ライブラリ関数 `accept` を使って *socket* 上の接続を受け入れる．

（5） ライブラリ関数 `close` を使って *socket* を閉じる．

（6） ライブラリ関数 `read` や `write` を使って通信を行う．

（7） ライブラリ関数 `close` を使って `accept` が返した *socket* を閉じる．

ライブラリ関数 `bind listen, accept` 及び (5) の `close` はライブラリ関数 `socket` で作成された *socket* を引数にとる．ライブラリ関数 `read, write` と (7) の `close` はライブラリ関数 `accept` が返した *socket* を引数にとる．

ライブラリ関数の組合せは `socket` だけではなく，他のライブラリ関数についても存在する．この組合せはプログラムによって全く異なった組合せ方がされるのではなく，ある決まった組合せ方が存在している．我々は多くのソフトウェアで利用されているライブラリ関数の組合せをイディオムとして抽出する．イディオムを抽出するために必要となるライブラリ関数の組合せを表現するために我々が提案した関数呼出依存グラフの定義を付録に示す．

2.2 出現頻度の高い関数呼出依存グラフ

既存のソフトウェアで利用されているライブラリ関数の組合せの中で，出現頻度の高い関数呼出依存グラフの多くは返り値検査を表すグラフである．このほかには，返り値検査に比べて相対的に数は少ないけれど，ライブラリ関数の組合せを表すグラフが得られる．本節ではこれらの関数呼出依存グラフの特徴について述べる．

返り値検査

実行中にエラーが発生した場合に，特定の値を返すライブラリ関数が数多く存在する．ソフトウェアの動作を安定させるために，ライブラリ関数が返す値を検査し，エラー値の場合にはプログラムを終了させたり，安全に動作するように別の処理をさせるなどする必要がある．そのため，ソフトウェア開発者はライブラリ

関数を利用する際，返り値検査をする必要があるか否かを知らなければならない．

長年利用され続けているソフトウェアは，安定して動作しており，ライブラリ関数の返り値検査が適切に記述されていることを期待できる．そのため，これらのソフトウェアから関数呼出依存グラフを抽出し，得られたグラフから返り値検査を含むグラフの出現頻度を調べることにより，返り値チェックが必要か否かを決定することが可能である．

FreeBSD 4.5-RELEASE のソーストリーから得られた関数呼出依存グラフ群の中で，頻繁に現れる返り値検査に関するグラフを表 1 に示す．表 1 の『関数呼出依存グラフ』の列には，抽出された関数呼出依存グラフに対応する擬似コードを示している．『同じグラフの出現頻度』の列の分子は，擬似コードで示された関数呼出依存グラフと同じグラフの数を示している．分母は，関数呼出依存グラフのルートノードに当たるライブラリ関数を含む関数呼出依存グラフの数を示している．括弧内は同じグラフの出現頻度を示している．

表 1　返り値検査に関する関数呼出依存グラフ
Table 1　Function call dependency graphs for the return value check.

FCDG	Occurrence Frequency

`$1 = getpwnam();`
`if ($1 == NULL) {`
`  ...`
`}`
	67 / 72 (93.1%)

`$1 = getenv();`
`if ($1 == NULL) {`
`  ...`
`}`
	80 / 182 (44.0%)

`$1 = localtime();`
`if ($1 == NULL) {`
`  ...`
`}`
	44 / 100 (44.0%)

`if (($1 = getopt()) != -1) {`
`  if ($1) {`
`  ...`
`  }`
`}`
	178 / 428 (41.6%)

`$1 = open();`
`if ($1 == -1) {`
`  ...`
`}`
	92 / 302 (30.5%)

`$1 = malloc();`
`if ($1 == NULL) {`
`  ...`
`}`
	212 / 870 (24.4%)

`$1 = fopen();`
`if ($1 != NULL) {`
`  ...`
`}`
	71 / 346 (20.5%)

表 1 の一番上に示した関数呼出依存グラフにの出現頻度は，直感的にはライブラリ関数 getpwnam が既存のソフトウェアの中で利用されている個所が 72 あり，そのうち関数呼出依存グラフとして表現されているコードと同じ使い方をしている個所が 67 存在していたことを示している．

以下に表 1 に示した関数呼出依存グラフで利用されているライブラリ関数が返り値の検査を必要とする理由を述べる．

- getpwnam

引数に与えられたユーザの情報をパスワードデータベースから検索する．もし，引数に与えられたユーザが存在していなければ NULL ポインタを返す．

- getenv

環境変数の値を取得する．もし，現在の環境で，その変数がない場合，NULL ポインタを返す．

- localtime

指定された時間をユーザが指定したタイムゾーンでの時刻要素へ変換する．この変換に失敗すると，NULL ポインタを返す．

- getopt

この関数はコマンドライン引数を解釈し，繰り返し呼び出されるごとに，次の文字を返す．オプション文字がそれ以上見つからなくなると -1 を返す．

- open

引数に与えられたファイルを，読取りまたは書込み用にオープンする．もしオープンに失敗すると，-1 を返す．

- malloc

この関数は引数で与えられたサイズ分のメモリを確保する．もしメモリの確保に失敗すると，NULL ポインタを返す．

- fopen

引数に与えられたファイルを，読取りまたは書込み用にオープンする．もしオープンに失敗すると，NULL ポインタを返す．

これらの返り値検査はソフトウェアの動作を安定させるために必要な記述であるため他のライブラリ関数との組合せよりも頻繁に現れる．そのため出現頻度が高くなるが，組合せ方を検索したい場合にこれらのコード記述が検索結果として上位にあると，求めている情報がなかなか得られない．しかし，これらのグラフに現れる依存関係はライブラリ関数を利用する上で重要な情報である．

ライブラリ関数の組合せ

表 2 に FreeBSD 4.5-RELEASE のソーストリーで頻繁に利用されているライブラリ関数の組合せを示す．表 2 は表 1 と同様に関数呼出依存グラフに対応する擬似コードと抽出した関数呼出依存グラフ群の中の同じグラフの出現頻度を示している．以下に表 2 に示した関数呼出依存グラフで示されるコード記述について述べる．

- fopen

図 1 に示したような設定項目とそれに対する値が記述された設定ファイルからその値を取得するために記述される．

- socket

プロセス通信を行う際に利用される．ライブラリ関数 socket は通信を行うための *socket* を返す．ライブラリ関数 setsockopt は *socket* のオプションを設定する．ライブラリ関数 bind は *socket* に名前をつける．接続するために他のプロセスはこの名前を指定する．ライブラリ関数 listen は他のプロセスからの接続要求を OS に通知する．ここまでが，プロセス通信を行うための準備段階である．

表 2　頻繁に利用されている組合せ
Table 2　Frequently used combinations.

FCDG	Occurrence Frequency

`$1 = fopen();`
`if ($1 == NULL) {`
`  perror();`
`  exit();`
`}`
`$2 = fgets($1);`
`if ($2 != NULL) {`
`  $3 = strchr($2);`
`  if ($3 != NULL) {`
`  strcpy($3);`
`  }`
`}`
	46 / 346 (13.3%)

`$1 = socket();`
`if ($1 < 0) {`
`  ...`
`}`
`setsockopt($1);`
`bind($1);`
`listen($1);`
`close($1);`
	43 / 300 (14.3%)

`$1 = getopt();`
`if ($1 != -1) {`
`  $2 = atoi();`
`  if ($2 < 0) {`
`  ...`
`  }`
`}`
	61 / 428 (14.3%)

```
ForwardAgent no
ForwardX11 no
RhostsAuthentication no
  …
```

図 1　設定ファイル
Fig. 1　Configuration file.

このように頻繁に利用されるライブラリ関数の組合せが存在しているが，類似した多数の関数呼出依存グラフが存在する．例えば，表 2 の getopt のグラフに関しては atoi ではなく，atol が利用されている場合や if の条件式の [$2 < 0] がない場合などが類似したグラフとして得られる．これらの微細な違いを一まとまりのイディオムとして提示することによって様々な組合せ方を容易に参照することが可能となる．そのため関数呼出依存グラフの分類が必要となる．

2.3　イディオム

イディオムとは様々なソフトウェアに記述されている頻繁に利用されるパターンのことである．本論文ではコード記述において頻繁に利用されているライブラリ関数の組合せをイディオムととらえる．具体的には下記のようにイディオムを定義する．定義中の類似については **3.** で，重要度については **4.** で説明する．

F_l：ライブラリ関数 l を含む $FCDG$ の集合

$C(F_l)$：F_l において類似した $FCDG$ を分類したカテゴリーの集合

I_l：ライブラリ関数 l のイディオムを表す．イディオムとは $C(F_l)$ の各カテゴリーごとの $FCDG$ のうち，重要度の最も高い $FCDG$ を表す．

前節で述べたように関数呼出依存グラフ（FCDG）には返り値検査に関する重要な依存関係を表すグラフとライブラリ関数の組合せを表現するグラフが存在する．これらの中から頻繁に利用されている組合せがイディオムである．しかし，関数呼出依存グラフとして得られるグラフには微細な違いがあるものから全く異なった組合せのグラフまで様々なグラフが得られる．そこで，本研究では，微細な違いを吸収し，同じカテゴリーの組合せとして表現する．その中で返り値検査が必要なライブラリ関数では返り値検査を適切に行っていて，依存関係として必要なライブラリ関数が呼ばれているコードから得られた関数呼出依存グラフを重要度により評価する．この重要度が最も高い関数呼出依存グラフをイディオムとすることによって，適切なライブラリ関数の利用例を提示することを目指している．

3. 関数呼出依存グラフの分類

3.1 関数呼出依存グラフの類似

本章では，関数呼出依存グラフを分類するために，必要となる関数呼出依存グラフの類似度について述べる．まず，この類似度を定義するために，グラフを特徴づける度合を表す依存辺の重みを定義する．

関数呼出依存グラフで表現される依存辺は依存元のノード s と依存先のノード t の組 (s, t) で表現される．s と t はライブラリ関数または条件式を表す．この組が収集した関数呼出依存グラフ群の中に多く存在する場合，依存元 s と依存先 t の依存関係が強いことを示している．

関数呼出依存グラフはグラフを抽出するための処理時間及びメモリ使用量を減らすために関数をまたいだ解析を行っていない．そのため，同じ処理のコード記述でも返り値を関数の引数に渡し，他の関数に処理を託しているコード記述と，一つの関数内ですべての処理を行っているコード記述では，異なる関数呼出依存グラフが得られる．このようなコード記述で，特にライブラリ関数を利用する上で必要となる依存関係の有無による違いを，類似したコード記述として取得するために，これらのコード記述から得られる関数呼出依存グラフの違いが小さくなるように類似度を定義する．ライブラリ関数を利用するために必要となる依存関係か否かの判断はその依存関係が利用されている頻度によって決定する．

自然言語の文書検索では，助詞や助動詞などは文書を特徴づけない．文書中の各単語が文書において特徴を表しているかどうかを判断するために TF-IDF 法 [6] により重み付けを行う．TF-IDF 法では，多くの文書中に出現する単語は一般的に用いられる単語であり，文書を特徴づけないため，重みを低くし，特定の文書にのみ出現する単語の重みを高くすることにより文書を特徴づけているかどうかを判断している．

イディオム検索においては，多くの関数呼出依存グラフに現れる依存関係は，返り値検査や `fopen` と `fclose` の間の依存関係など依存元となるライブラリ関数を利用する上で必要となる依存関係である．イディオム検索ではライブラリ関数の異なった組合せを分類して提示することにより，様々な組合せ方の参照を容易に行うことを目指す．これを実現するために我々は次の経験則を使う．その経験則とは,『特定の関数呼出依存グラフにのみ現れる依存関係がその関数呼出依存グラフをより特徴づける』ということである．

そこで，文書検索で用いられている TF-IDF 法を利用し，多くの関数呼出依存グラフに現れる依存関係の重みを低くするように次のように各依存関係に重み付けを行う．ここで，TF-IDF 法における文書に対応するものが関数呼出依存グラフであり，単語に対応するものが依存辺である．定義式中の l, m は節点に付けられたラベルであり，ライブラリ関数名または条件式を表す．依存関係の出現頻度 $FREQ$ は既存のソフトウェアから抽出した関数呼出依存グラフ群に基づいて定義される．

L：関数呼出依存グラフ群中節点ラベルの集合
$l, m, n \in L$

$E_{l,m}$：関数呼出依存グラフの l, m でラベル付けさ

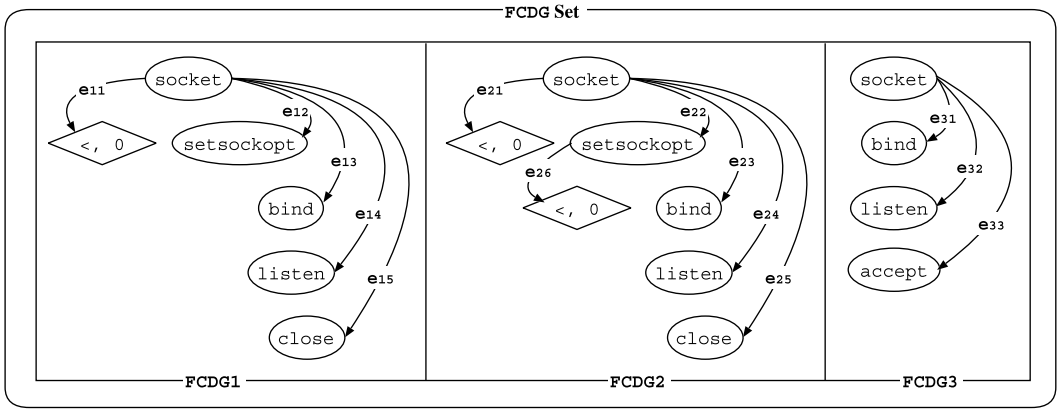

図 2 関数呼出依存グラフ群の例
Fig. 2 Examples of function call dependency graphs.

れた節点間の依存辺の集合

$GNUM_{l,m}$：関数呼出依存グラフ群の中にある依存辺 $e \in E_{l,m}$ をもつ関数呼出依存グラフの数

$GNUM_l$：関数呼出依存グラフ群の中にある l でラベル付けされた節点を含む関数呼出依存グラフの数

$d_{l,m}$：ラベル l, m で表現されるライブラリ関数または条件式間の依存関係

$$FREQ_{l,m} = \begin{cases} 0 & \left(\begin{array}{c} GNUM_{l,m} = 0 \\ \text{または} \\ GNUM_l = 0 \text{ のとき} \end{array} \right) \\ \dfrac{GNUM_{l,m}}{GNUM_l} & (\text{その他}) \end{cases}$$

：関数呼出依存グラフ群における依存関係 $d_{l,m}$ の出現頻度

$$w_{l,m} = \begin{cases} 0 & (FREQ_{l,m} = 0 \text{ のとき}) \\ \dfrac{1}{FREQ_{l,m}} & (\text{その他}) \end{cases}$$

：依存関係の重み

$FREQ$ は依存元である節点のライブラリ関数を利用するために，依存先である節点のライブラリ関数を必要とする度合を表す．$w_{l,m}$ は TF-IDF 法を利用し，頻出する依存関係には重みが小さくなるように定義した．具体的に図 2 に示す関数呼出依存グラフ群に対する $L, E, GNUM, FREQ, w$ を表 3 に示す．

この重みの値は多くの関数呼出依存グラフに存在する依存関係に対しては小さい値となり，特定の関数呼出依存グラフにのみ存在する依存関係に対しては大きい値となる．また，様々な組合せが存在するライブラリ関数に対してはそれぞれの依存関係が特徴的であると考えられるため大きい値となる．

具体的には socket，bind，fopen，getopt などの返り値検査を必要とする関数においては，これらの関数と返り値検査を行うための条件式との間の依存関係はほとんどのプログラムにおいて記述されている．また，socket と bind の間の依存関係のようにほとんどの場合，その組合せとして利用する関数が存在する．このような依存関係は依存元または依存先の関数を利用する上で必要不可欠な依存関係である．

図 2 において socket と bind の間の依存関係は三つの関数呼出依存グラフすべてに現れているので，関数呼出依存グラフを特徴づけないと判断し，これらのグラフを比較する際，これらの違いを軽視する．そのために重みを小さくする．逆に socket と accept の間の依存関係は一つだけしか現れていないので，関数呼出依存グラフを特徴づけると判断して，重視する．そのために重みを大きくする．

表 3　図 2 の FCDG 群に対する諸々の値
Table 3　Each values for FCDGs in Fig. 2.

L	{socket, setsockopt, bind, listen, close, [<, 0], accept}
$E_{socket,[<,0]}$	$\{e_{11}, e_{21}\}$
$E_{socket,setsockopt}$	$\{e_{12}, e_{22}\}$
$E_{socket,bind}$	$\{e_{13}, e_{23}, e_{31}\}$
$E_{socket,listen}$	$\{e_{14}, e_{24}, e_{32}\}$
$E_{socket,close}$	$\{e_{15}, e_{25}\}$
$E_{setsockopt,[<,0]}$	$\{e_{26}\}$
$E_{socket,accept}$	$\{e_{33}\}$
$GNUM_{socket,setsockopt}$	2
$GNUM_{socket,[<,0]}$	2
$GNUM_{socket,bind}$	3
$GNUM_{socket,listen}$	3
$GNUM_{socket,close}$	2
$GNUM_{setsockopt,[<,0]}$	1
$GNUM_{socket,accept}$	1
$GNUM_{socket}$	3
$GNUM_{[<,0]}$	2
$GNUM_{setsockopt}$	2
$GNUM_{bind}$	3
$GNUM_{listen}$	3
$GNUM_{close}$	2
$GNUM_{accept}$	1
$FREQ_{socket,setsockopt}$	0.67
$FREQ_{socket,[<,0]}$	0.67
$FREQ_{socket,bind}$	1.00
$FREQ_{socket,listen}$	1.00
$FREQ_{socket,close}$	0.67
$FREQ_{setsockopt,[<,0]}$	0.50
$FREQ_{socket,accept}$	0.33
$W_{socket,setsockopt}$	1.50
$W_{socket,[<,0]}$	1.50
$W_{socket,bind}$	1.00
$W_{socket,listen}$	1.00
$W_{socket,close}$	1.50
$W_{setsockopt,[<,0]}$	2.00
$W_{socket,accept}$	3.00

関数呼出依存グラフは関数をまたいだ依存解析を行っていないため，他の関数に引数として返り値を渡している場合，これらの依存関係を取得することができない．関数呼出依存グラフの違いを調べる際，これらの依存関係の中で利用頻度が高い場合，$w_{l,m}$ の値が小さくなり，これらの有無による違いを小さくする．

逆に，特定の関数呼出依存グラフにのみ現れる依存関係の組には重み付けを大きくしているため，これらの依存関係の有無による違いが大きくなる．この重みを用いて関数呼出依存グラフ F_x と F_y の類似度を次のように定義する．

$|E|$：依存辺の集合 E の要素数
$L(F)$：関数呼出依存グラフ F の節点のラベルの集合

$$sim(F_x, F_y) = \begin{cases} 0\ (\forall l, \forall m, w_{l,m} = 0 \text{ のとき}) \\ \dfrac{\sum_l \sum_m w_{l,m} \cdot c_{l,m}}{\sum_l \sum_m w_{l,m}}\ (\text{その他}) \end{cases}$$

$$c_{l,m} = \begin{cases} 1\ (F_x, F_y \text{が依存関係 } d_{l,m} \text{をともにもつ場合}) \\ 0\ (\text{その他}) \end{cases}$$

$$l, m \in L(F_x) \cup L(F_y)$$

3.2 類似関数呼出依存グラフの例

我々が定義した類似度に基づいて FreeBSD のソーストリーから抽出された関数呼出依存グラフに対して計算した類似度の高いものを表 4 に示す．表 4 に示した関数呼出依存グラフにおいて，異なる依存辺に対する出現頻度を表 5 に示す．

最初の例は getopt を用いてコマンドライン引数を解析するためのコード記述である．(a1) と (a2) の違いは，atoi の返り値の検査を行っているか否かの違いである．atoi と返り値の検査のための条件式の依存関係の出現頻度は表 5 に示したように 0.83 であり，高い頻度で行われている．高い頻度の依存関係の重みは小さく，これを片方だけがもつ場合，類似度 $sim(F_x, F_y)$ の分母が小さくなり，類似度は大きくなる．そのため，これらのグラフは類似した関数呼出依存グラフとして得られる．二つ目，三つ目の例についても同様に出現頻度の高い依存関係の有無による違いであるため，類似した関数呼出依存グラフとして得られる．

頻出する依存関係の有無によるグラフの違いが小さくなるように重み $w_{i,j}$ を定義したため，表 5 に示したように返り値検査の有無によるグラフの違いは類似したグラフとして得られた．この例だけでなく，実験対象である FreeBSD 4.5-RELEASE のソーストリーから得られた関数呼出依存グラフでは，ライブラリを利用する際に必要となる返り値検査の有無によるグラフの違いは，すべて類似していると判定された．既存のソフトウェアの中で頻出する依存関係は重要な関係であるとみなし，そのような依存関係がない場合，関数をまたいだ依存関係になっていると仮定した．

類似度が高いと評価された関数呼出依存グラフは人が見ても類似したコードを表現している．逆に，類似度が低いと評価された中には表 6 に示したように人が見ると類似しているコードを表現するグラフが存在していた．今回の実験では類似した機能をもつライブラリ関数を考慮していないため類似していないグラフとして判定される．関数の機能の類似性を利用して分類することにより，機能的に類似した分類を行うことができるが，機能の類似性を利用した分類は今後の課題とする．

表 4 類似度の高いイディオム
Table 4 Idioms with high similarity.

getopt

```
$1 = getopt();
if ($1 != -1) {
    $2 = atoi();
    if ($2 < 0) {

    }
}
```
(a1)

```
$1 = getopt();
if ($1 != -1) {
    $2 = atoi();

}
```
(a2)

fopen

```
$1 = fopen();
if ($1 == NULL) {
    perror();
    exit();
}
$2 = fgets($1);
if ($2 != NULL) {
    $3 = strchr($2);
    if ($3 != NULL) {
        strcpy($3);
    }
}
fclose($1);
```
(b1)

```
$1 = fopen();
$2 = fgets($1);
if ($2 != NULL) {
    $3 = strchr($2);
    if ($3 != NULL) {
        strcpy($3);
    }
}
fclose($1);
```
(b2)

socket

```
$1 = socket();
if ($1 < 0) {

}
setsockopt($1);
bind($1);
listen($1);
close($1);
```
(c1)

```
$1 = socket();
if ($1 < 0) {

}
$2 = setsockopt($1);
if ($2 < 0) {

}
bind($1);
listen($1);
close($1);
```
(c2)

```
$1 = socket();
setsockopt($1);
bind($1);
listen($1);
```
(c3)

表 5 依存関係の出現頻度
Table 5 Occurrence frequency of dependencies.

Dependency	$FREQ$
atoi と [< 0]	0.83
fopen と [== NULL]	0.87
socket と [< 0]	0.90
setsockopt と [< 0]	0.75
socket と close	0.85

表 6 類似度の低いイディオム
Table 6 Idioms with low similarity.

getopt

```
$1 = getopt();
if ($1 != -1) {
    $2 = atoi();
    if ($2 < 0) {

    }
}
```
(a1)

```
$1 = getopt();
if ($1 != -1) {
    $2 = atol();

}
```
(a2)

fopen

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
$2 = fgets($1);
if ($2 != NULL) {
    ...
}
fclose($1);
```
(b1)

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
$2 = fread($1);
if ($2 != 0) {
    ...
}
fclose($1);
```
(b2)

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
$2 = fscanf($1);
if ($2 != 0) {
    ...
}
fclose($1);
```
(b3)

4. 関数呼出依存グラフの順位付け

分類された関数呼出依存グラフの中から代表例を抽

出するため，また関数呼出依存グラフの検索結果の順番を決めるために関数呼出依存グラフの重要度を定義する．関数呼出依存グラフの類似度を求めるために各依存辺に対して重み付けを行った．この重み付けは頻出する依存辺は関数呼出依存グラフの違いが相対的に小さくなるように定義した．しかし，頻出する依存辺は，依存元となるライブラリ関数を利用する上で依存先の条件式またはライブラリ関数を必要としていることを示す．例えば，返り値検査や fopen と fclose である．それらの有無は関数呼出依存グラフを分類する上では重要ではないが，これらの依存関係を含んでいることは必要な依存関係が関数呼出依存グラフに含まれていることを示すため，出現頻度の高い依存関係は重要な要素である．

イディオムの検索結果として提示する関数呼出依存グラフには，これらの出現頻度の高い依存関係を含んでいることが望ましい．これは，結果として得られた関数呼出依存グラフを参照することによって必要な情報をすべて得ることが可能となるからである．必要な依存関係を含んだ関数呼出依存グラフを提示するために，重要度を次のように定義する．

$L(F)$：関数呼出依存グラフ F の節点のラベルの集合

$E(F)$：関数呼出依存グラフ F の辺の集合

$l, m \in L(F)$

$$imp(F) = \frac{\sum_{l}\sum_{m} FREQ_{l,m}}{|E(F)|}$$

この重要度により，多くの類似した例を参照することなく，ライブラリ関数を利用する上で必要な情報を容易に参照することができる．表 7 に FreeBSD のソーストリーから得られた関数呼出依存グラフの擬似コードとそれに対する重要度の例を示す．

FreeBSD のソーストリーから得られた各カテゴリーの関数呼出依存グラフでは，返り値検査などのライブラリ関数を利用する上で必要となる依存関係を多くもつグラフがより重要度が高いと評価された．これはそのような依存関係がその他の依存関係の平均出現頻度よりも高いことを示している．また，カテゴリー間では，ノード数の少ない関数呼出依存グラフがより重要度が高く評価された．これは，トップノードに当たるライブラリ関数を利用する上で必要となる依存関係は利用頻度が高いため，そのような依存関係のみから構成されるグラフの方が重要度が高くなるからである．

表 7　関数呼出依存グラフの重要度の例
Table 7　Examples of the importance of FCDGs.

`$1 = socket();`
`if ($1 < 0) {`
`  ...`
`}`
`$2 = setsockopt($1);`
`if ($2 < 0) {`
`  ...`
`}`
`bind($1);`
`listen($1);`
`close($1);`
	`$1 = socket();`
	`if ($1 < 0) {`
	`  ...`
	`}`
	`setsockopt($1);`
	`bind($1);`
	`listen($1);`
	`close($1);`

0.67	0.64

FREQ	
socket, [<, 0]	1
socket, setsockopt	0.4
socket, bind	0.5
socket, listen	0.4
socket, close	0.9
setsockopt, [<, 0]	0.8

これにより，必要な依存関係をもつグラフから順に参照することが可能となる．

5. イディオム検索システム

5.1 システム概要

イディオム検索システムの全体像を図 3 に示す．本システムでは，まず最初に多くのプログラムを Sapid（Sophiscated Application Programming Interface for software Development）[1], [4] を用いて解析する．Sapid は細粒度リポジトリに基づく CASE ツールプラットフォームである．図 3 に示した SDB がこのリポジトリである．

関数呼出依存グラフは，プログラムを解析した結果を格納した SDB を用いてデータ依存関係及び制御依存関係を解析し，抽出する．抽出された関数呼出依存グラフを我々が定義した依存辺の出現頻度に基づく類似度によって分類する．

我々はライブラリ関数に関するイディオムを検索するためのツールを実装した．図 4 に実装した検索システムを示した．左上フレームには我々が実際に解析し，関数呼出依存グラフを抽出した FreeBSD 4.5-RELEASE の/usr/src/usr.sbin で利用されているライブラリ関数のリストを表している．このフレームで，ユーザはライブラリ関数をどのように利用するか，サンプルを参照するために，ライブラリ関数名により検索を行う．左下フレームには検索された関数のイディオムが出力される．左下フレームのイディオムと類似した関

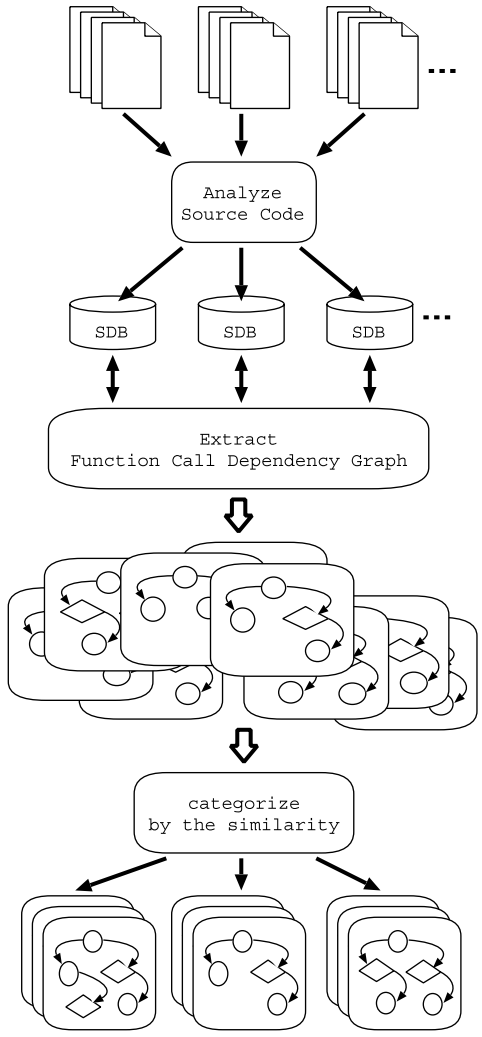

図 3　システムの概要
Fig. 3　Abstract of system.

数呼出依存グラフとなるコード記述が出力される．このフレームには実際にそのコード記述がある個所へのリンクが張られ，実際のソースコードを参照することが可能である．

5.2 分類結果

FreeBSD 4.5-RELEASE の /usr/src/usr.sbin にある平均 1,500 行のソフトウェア 152 個に対して関数呼出依存グラフを作成し，イディオムの抽出を行った．本実験では，利用頻度 $FREQ$ が 0.1 以下の関数呼出依存グラフは分類の対象としていない．これは，利用頻度の低い関数呼出依存グラフは特殊な利用例と考えられるためである．このときのしきい値 0.1 は実験的に定めた値である．

表 8 に 152 個のソフトウェアの中で利用頻度の高いライブラリ関数 fopen と getopt の数呼出依存グラフに対して分類した結果をに示している．表 8 に示した結果は分類された関数呼出依存グラフの中で最も重要度の高い関数呼出依存グラフを擬似コードで表現したものである．fopen は全部で 121 個の関数呼出依存グラフが得られ，これらを 5 個のカテゴリーに分類することができた．getopt は全部で 139 個の関数呼出依存グラフが得られ，これらを 6 個のカテゴリーに分類することができた．

利用頻度が低く分類の対象となっていない関数呼出依存グラフは，fopen では，18 個，getopt では，26 個存在していた．実験対象のソフトウェアでは全部で 211 個のライブラリが利用されていたが，fopen や getopt と同様に 3 個から 8 個の平均 5 個のカテゴリーに分類することができた．また，利用頻度が低く，分類対象となっていない関数呼出依存グラフは平均 2 割であった．そのため，利用例を検索する際には，分類された 3 個から 8 個の例と，利用頻度の低い残りの 2 割ですべての例を示すことができ，効率的に検索を行うことができる．

FreeBSD のソーストリーからの関数呼出依存グラフの作成にかかる時間は Pentium4 2.8 GHz，メモリ 512 MByte の計算機で約 10 時間であり，非常に時間がかる．しかし，約 100 個の関数呼出依存グラフの分類にかかる時間は約 30 秒であり，あらかじめ関数呼出依存グラフを用意しておけば，実用に耐え得る時間で利用例の検索を行うことができる．

6. むすび

本研究では，これまでに作成された多くのソフトウェアからイディオムとなる頻繁に利用されているライブラリの組合せを効率的に検索することを目指している．そこで，我々が提案した関数呼出依存グラフを用いてライブラリ関数間の依存関係による組合せを表現し，得られた関数呼出依存グラフを分類する手法を提案した．

関数呼出依存グラフには，用いられる頻度の異なる依存関係が存在する．我々は依存関係の出現頻度に基づいて類似度を定義した．FreeBSD 4.5-RELEASE のソーストリーから得られた 1,531 個の関数呼出依存グラフでは，211 個の各ライブラリ関数に対して平均 5 個のカテゴリーに分類することができた．これにより，

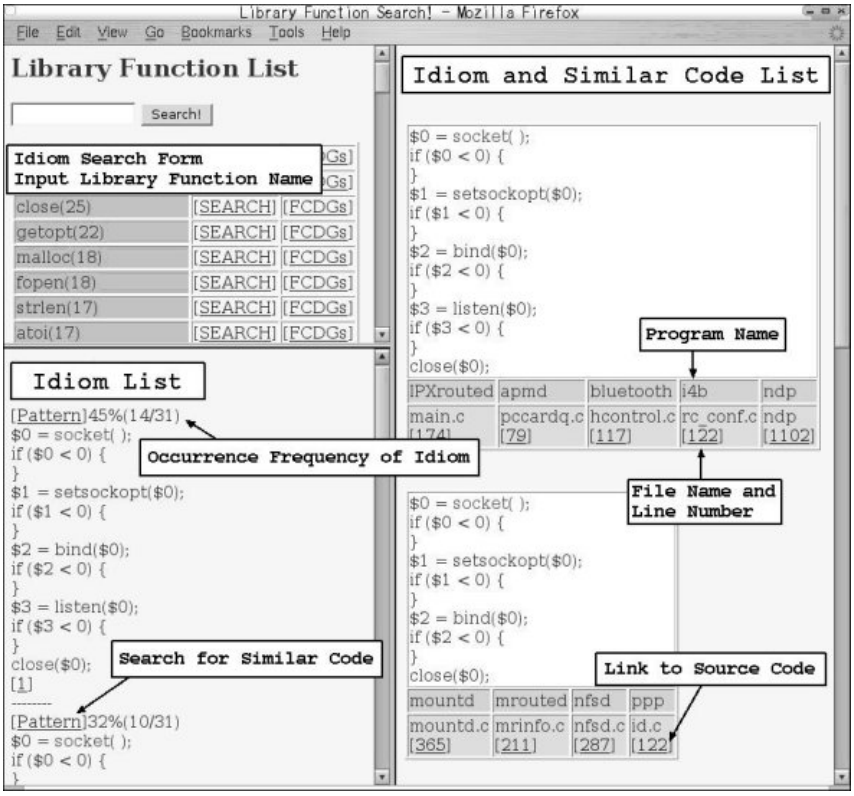

図 4　イディオム検索システム
Fig. 4　Idiom clustering system.

表 8　分類結果
Table 8　The results of categorization.

fopen

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
fclose($1);
```

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
$2 = fgets($1);
if ($2 != EOF) {
    ...
}
fclose($1);
```

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
$2 = fscanf($1);
if ($2 != 0) {
    ...
}
fclose($1);
```

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
$2 = fread($1);
if ($2 != 0) {
    fwrite();
}
fclose($1);
```

```
$1 = fopen();
if ($1 == NULL) {
    ...
}
$2 = fgetc($1);
if ($2 != EOF) {
    ...
}
fclose($1);
```

getopt

```
$1 = getopt();
if ($1 != -1) {
    $2 = atoi();
    ...
}
```

```
$1 = getopt();
if ($1 != -1) {
    $2 = strtoul();
}
```

```
$1 = getopt();
if ($1 != -1) {
    memcpy();
    ...
}
```

```
$1 = getopt();
if ($1 != -1) {
    $2 = isdigit();
    if ($2 != 0) {
        ...
    }
}
```

```
$1 = getopt();
if ($1 != -1) {
    $2 = strtol();
}
```

```
$1 = getopt();
if ($1 != -1) {
    strdup();
}
```

ライブラリ関数の組合せ方を参照する際，類似した例を何度も参照することを防ぐことができる．

また，依存関係の出現頻度に基づいて関数呼出依存グラフの重要度を定義した．これにより，ライブラリ関数を利用する上で必要となる依存関係を多く含む例を優先的に提示することが可能となる．

今後の課題として以下のことが挙げられる．

- 関数をまたいだ依存解析

現在は処理効率を軽減するために関数をまたいだ依存解析を行っていないが，関数をまたいだ依存解析をすることにより，より大きな多くの依存関係のつながりを得ることができる．一般には関数をまたいだ依存解析は計算時間が大きくなり，現実的ではない．しかし，本研究で対象としているライブラリ関数の依存関係に限ると複数回にわたって関数をまたぐことが少なく，1 段階のみ関数をまたいだ依存解析を行うだけで十分であることが経験的に分かっている．そのため，1 段階のみ関数をまたいだ依存解析をすることで現実的な時

間で，より大きな多くの依存関係を得ることができる．

- ライブラリの類似性を考慮した分類

現在はライブラリの機能的な類似性を考慮していないため，人が見ると類似したコードでも類似していないコードとして判定される．しかし，シソーラスなどを用いることにより，ライブラリの機能的な類似性を考慮することによって，そのようなコードも類似したコードとして分類することが可能となる．

- 高度な検索機能

現在はライブラリ関数名一つを入力として検索を行っているが，複数のライブラリ関数を入力とした検索やコード記述による検索ができれば，より容易に求めている利用例を検索することができる．

文　献

[1] Sapid home page. http://www.sapid.org/

[2] R.I. Bull, A. Trevors, A.J. Malton, and M.W. Godfrey, “Semantic grep: Regular expressions + relational abstraction,” Proc. Ninth Working Conference on Reverse Engineeering, pp.267–276, 2002.

[3] A. Chen, E. Chou, J. Wong, A.Y. Yao, Q. Zhang, S. Zhang, and A. Michail, “Cvssearch: Searching through source code using cvs comments,” Proc. IEEE International Conference on Software Maintenance, pp.364–375, 2001.

[4] N. Fukuyasu, S. Yamamoto, and K. Agusa, “An evolution framework based on fine grained repository,” Proc. International Workshop on Principles of Software Evolution, pp.43–47, 1999.

[5] A. Michail, “Browsing and searching source code of applications written using a gui framework,” Proc. 24th International Conference on Software Engineering, pp.327–337, 2002.

[6] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” Inf. Process. Manage., vol.24, no.5, pp.513–523, 1988.

[7] 荒木一雄（編），新英文法用例辞典，研究社出版，1997.

[8] 荒牧英治，黒橋禎夫，柏岡秀紀，田中英輝，“用例ベース翻訳における用言句の簡潔な翻訳の実現，” 言語処理学会第 10 回年次大会予稿集（B1-3），March 2004.

[9] 渥美紀寿，山本晋一郎，阿草清滋，“関数呼出依存グラフに基づくプログラミング支援，” 日本ソフトウェア科学会第 18 回大会論文集（2B-2），Sept. 2001.

[10] 林　巨樹（編），現代国語例解辞典，第三版，小学館，2000.

[11] 山名早人，近藤秀和，“解説：サーチエンジン google，” 情報処理，vol.42, no.8, pp.775–780, 2001.

[12] 内元清貴，関根　聡，村田真樹，井佐原均，“用例に基づく手法と機械学習モデルの組み合わせによる訳語選択，” 自然言語処理学会論文誌，vol.10, no.3, pp.87–114, April 2003.

[13] 白井　諭，山本和英，“換言事例の収集—機械翻訳における多様性確保の観点から，” 言語処理学会第 7 回年次大会併設ワークショップ予稿集，pp.3–8, 2001.

[14] 三浦　良，山本晋一郎，阿草清滋，“プログラミングナビゲーションのための関数呼び出し依存グラフ，” 日本ソフトウェア科学会コンピュータソフトウェア別冊，ソフトウェア発展，pp.19–29, Dec. 2000.

付　録

関数呼出依存グラフの定義

関数呼出依存グラフは文献 [9] で次のように定義した．

- 節点
  - 定義節点

関数の返り値を変数に代入しているステートメント

  - 参照節点

関数の返り値を関数呼出しの引数として参照しているステートメント

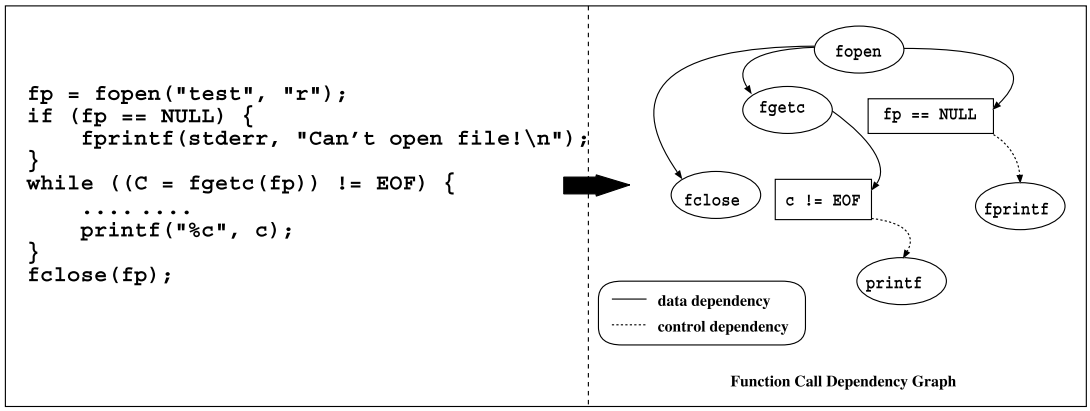

図 A·1　コード断片と対応する FCDG
Fig. A·1　Code fragments and the FCDG.

– 条件式節点
関数の返り値を条件式で参照しているステートメント
– 制御節点
条件式ノードの条件式が真のときに実行される関数呼出し式

- 辺

– データ依存辺
定義節点と参照節点，定義節点と条件式節点間に存在するデータ依存関係を表す有向辺
– 制御依存辺
条件式節点と制御節点の間に存在する制御依存関係を表す有向辺

図 A·1 にソースコードとそのコードに対する関数呼出依存グラフの例を示す．

（平成 16 年 12 月 10 日受付，17 年 4 月 4 日再受付）

渥美　紀寿　（正員）

2000 名大・工・電気電子情報学卒．2002 同大大学院博士前期課程了．2005 同大学院博士後期課程単位取得満期退学．2005 より南山大学講師．ソフトウェアの再利用に関する研究に従事．日本ソフトウェア科学会会員．

山本晋一郎　（正員）

1986 名大・工・電気卒．同大大学院に進学．1991 同大工学部に勤務．同助手，講師を経て，1998 より，愛知県立大学情報科学部助教授．博士（工学）．専門はソフトウェア工学，言語処理系，ユーザインタフェースに関する研究に従事．情報処理学会，ソフトウェア科学会各会員．

阿草　清滋　（正員）

1970 京大・工・電気第二卒．同大大学院に進学．1974 より京都大学工学部情報工学科に勤務．同助手，助教授を経て，1989 より，名古屋大学教授．現在，名古屋大学大学院情報科学研究科情報システム学専攻教授．工博．専門はソフトウェア工学．特に要求分析，仕様化技術，再利用技術に関する研究に従事．現在は軟らかいソフトウェアの実現を進めている．情報処理学会，ソフトウェア科学会各会員．