B-005

思考展開図のソースコード読解支援への適用

Appling Thinking Process Development Diagram to Source Code Reading Support

河崎 貴宜† 村川 猛彦‡
Takayoshi Kawasaki Takehiko Murakawa

1. まえがき

現在様々な設計支援システムが開発されている．創造学の分野では，新たな創造をスムーズに行うための支援手段として思考展開図が作られた[1][2]．本研究では思考展開図が「創造物の全体構造を理解して，それを人に伝えるための手段である」点に着目して，オープンソースソフトウェアのソースコード読解支援法を提案する．

2．思考展開図

思考展開図とは，創造の過程をある様式で図として表現したものをいう．図の左半分が機能領域，右半分が具体化領域で，機能領域とは課題や課題要素に相当し，具体化領域はそれに対する解決案となっている．基本的に機能領域と具体化領域が左右対称であり，創造のプロセスを思考展開図に当てはめてみると，自分の考えの抜けている部分が容易に分かりそれを補うことで創造の質が向上するという効果がある．思考展開図には様々なバリエーションがあり，12 列を超えるものや，左右対称でないものが存在するが，本研究では，左から 3 列，右から 3 列が木構造で左右対称の 6 列の思考展開図を用いる．

木構造を形式的に表現すると，以下のようになる．ノード x に対する i 番目の子ノードを x.i と表記する．ノード A を根とする木のノードの集合は，$\{x \mid x=A.i_1.i_2.....i_\ell\}$ と表現できる．ただし，ℓ は非負整数（木の深さを A_{LEVEL} と書けば，$0 \leqq \ell \leqq A_{LEVEL}$），$i_1,i_2,...,i_\ell$ は正整数である．また $i_1.i_2.....i_\ell$ をその木に対する x のパスという（x が根であれば，パスは空系列 ε であり，$x=A.\varepsilon$ と表す）．このとき，本研究で対象とする思考展開図は，機能領域に関する L を根（左端ノード）とする木と，具体化領域に関する R を根（右端ノード）とする木の 2 字組(L,R)で表現できる．ただし，いずれの木も深さは 2 であり，任意のパス p に対して，L.p が存在すれば R.p も存在し（逆も成り立つ），これらは機能と具体化に関する対応関係を持つ．

3. 思考展開図を用いたソースコードの構成

ソースコードのコメントを機能領域とし，ソースコードを具体化領域とした場合，思考展開図に表現できる．これにより，ソースコードに対応するコメントが明確でかつ木構造で表現できるため，全体構造が理解しやすいものとなる．

ソフトウェア全体で思考展開図を作成すると，C 言語や Java であるためクラスや構造が増え 6 列では不十分である．そこで，基本的に 6 列以上の情報を格納しておき，見る時に始点を決め，6 列で表示する形式をとる[3]．

ソースコードファイルから思考展開図の作成方法を述べる．オープンソースソフトウェアを対象としており，思考展開図で用いる列数，列ごとの名称は各言語によって設定している．GNU オープンソースソフトウェアを対象とした場合，列数を 10，名称を左から「ソフトウェアの説明」，「ディレクトリの説明」，「ファイルの説明」，「モジュールに対するコメント」，「コードに対するコメント」，「コード」，「モジュール」，「ファイル」，「ディレクトリ」，「ソフトウェア」と設定する．左 5 列と右 5 列をそれぞれ木構造として，データをサーバが保持しておく．中心から右側 2 列のノードにコード，左側 2 列のノードにコメントを並べる．コードに対応するコメントが存在しない場合，そのコメントのノードは空欄とする．コメント（「/*」と「*/」で囲まれた部分）の対応関係は，コードの直前に来ているもの，コードの同行であり直後に来ているものはそのコードのコメントとする．「モジュール」と「モジュールに対するコメント」のノードについては行番号の情報を先頭に記述する．

ユーザが指定した始点から 6 列で表示し，始点の変更によって表示範囲の拡大縮小を可能にする．構成は，右から 5 列の木構造の根を R_0，左から 5 列の木構造の根を L_0 とし，R_0，$R_0.i$，$R_0.i.j$ のいずれかのノードを R，そのパスを p と書くと，$R=R_0.p$，$R_0.p.i'$，$R_0.p.i'.j'$により表示用の思考展開図の右半分が，$L=L_0.p$，$L_0.p.i'$，$L_0.p.i'.j'$により左半分が構成できる．思考展開図の構成を図 1 に示す．

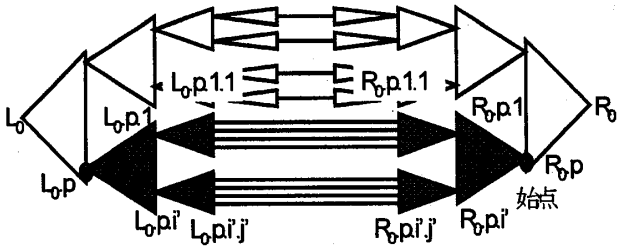

図 1 ソースコード読解支援のための思考展開図構成

4. ソースコード読解への適用

オープンソースソフトウェアを対象に，ソースコードとコメントから思考展開図を作成する．

4.1 GNU Hello への適用

GNU Hello とは，‘Hello,world’ を表示する簡単なプログラムで（ただしコマンドラインオプションにより任意のメッセージを出力できる），GNU ソフトウェアを作る際，どのように GNU コーディング規約に従えば良いかの基本例となっている．

使用したアーカイブファイルは hello-2.3.tar.gz であり，展開すると，ディレクトリは 10 個，ファイルは 208 個あった．ソフトウェア全体の思考展開図を作成した．その中で，最も外側にある列であるソフトウェアを始点にとった場合の思考展開図を図 2 に，hello.c ファイルを始点にとった場合の思考展開図を図 3 にそれぞれ示す．本稿では思考

† 和歌山大学システム工学研究科
‡ 和歌山大学システム工学部

展開図を，ノード間を線で結ばずに階層表現が出来る表形式で表現している．

ソフトウェアの説明	ディレクトリの説明	ファイルの説明	ファイル	ディレクトリ	ソフトウェア
Hello prints a friendly greeting.			config.guess	build-aux	GNU Hello
			ChangeLog	contrib	
			ChangeLog	doc	
		__fpending...	_fpending.c	gnulib/lib	
		closeout.m4...	closeout.m4	gnulib/m4	
			ChangeLog	src	
		hello.c -- ...	hello.c		
			ChangeLog	tests	
			ABOUT-NLS	.	

図 2　GNU Hello の思考展開図例（ソフトウェア始点）

ファイルの説明	モジュールに対するコメント	コードに対するコメント	コード	モジュール	ファイル
hello.c -- print a greeting message and exit. Copyright (C) 1992, 1995, 1996, 1997, 1998, 1999, 2000, ...				(l,20) #include...	hello.c
				(l,36) static ...	
	(メイン関数)		int optc;	(ll,39-127)int main (int argc, char *argv[]);	
			while (...)		
		One goal ...	switch (optc);		
		Print error...	if (lose || ...		
	(ll,131-133) Print help ...	TRANSLAT...	printf (...)...	(ll,135-171)static ...	
		TRANSLAT...	fputs (...)...		
	(l,175) Print...	xgettext...	printf (...)...	(ll,177-193) ...	

図 3　GNU Hello の思考展開図例（hello.c 始点）

4.2　wget への適用

wget とは，GNU で公開されている UNIX 上で使えるダウンロード支援ツール．Web から http 等を用いてファイルをダウンロードすることが出来る．

使用したアーカイブファイルは wget-1.11.3.tar.gz であり，展開すると，ディレクトリは 8 個，ファイルは 211 個あった．ソフトウェア全体の思考展開図を作成し，その中で main.c ファイルを始点にとった場合の思考展開図の一部を図 4 に示す．

ファイルの説明	モジュールに対するコメント	コードに対するコメント	コード	モジュール	ファイル
Command line parsing. Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, ...				(l.31) ...	main.c
	(l.94)...			(l.96) static...	
				(ll.131-253)...	
	(メイン関数)		char **...	(ll.697-1083)int main (int argc, char *const *argv);	
			int i, ret...		
		Initialize...	if (opt... .		
		Fill in ...	url = ...		
			for (i = ...		
	(ll.1096-...		const ...	(ll.1100-...	

図 4　wget の思考展開図例（main.c 始点）

4.3　lighttpd への適用

lighttpd とは，高速性が重視される環境に最適化された，オープンソースソフトウェアの Web サーバソフトウェア．メモリの消費量が少ないため，サーバの負荷が問題になっている場合や画像などの静的なコンテンツを動的なコンテンツと区別して送信する場合などに適している．

使用したアーカイブファイルは lighttpd-1.4.19.tar.gz であり，展開すると，ディレクトリは 12 個，ファイルは 320 個あった．ソフトウェア全体の思考展開図を作成し，その中で server.c ファイルを始点にとった場合の思考展開図の一部を図 5 に示す．

ファイルの説明	モジュールに対するコメント	コードに対するコメント	コード	モジュール	ファイル
				(l.1) #include..	server.c
				(.67) static ...	
			int i;	(ll.155-221) static ...	
			server ...		
	(メイン関数)		server ...	(ll.479-1493)int main (int argc, char **argv);	
			int print...		
		dump u...	for (i = ...		
		main-...	while (...		
			if (srv-...		

図 5　lighttpd の思考展開図例（server.c 始点）

5. 考察

5.1 充足率

コード一つに対するコメントの有無を判定し，ソースファイル全体でどれぐらいの割合でコメントがあるのかを「充足率」と呼び．コメント数／コード数×100 により求めた．前節で述べた各ソフトウェアに対して，充足率を表 1 に示す．

表 1　充足率表

ソフトウェア名	対象	コメント数	全体	充足率
GNU Hello hello.c	モジュール	14	34	41.18%
	コード	3	9	33.33%
	プリプロセッサ	0	2	0.00%
	宣言	1	8	12.50%
	関数	16	20	80.00%
wget main.c	モジュール	17	74	22.97%
	コード	30	243	12.35%
	プリプロセッサ	3	22	13.64%
	宣言	8	33	24.24%
	関数	21	76	27.63%
lighttpd server.c	モジュール	1	70	1.43%
	コード	21	368	5.71%
	プリプロセッサ	0	35	0.00%
	宣言	0	24	0.00%
	関数	16	125	12.80%

ここから GNU Hello に関して最も充足率が高いことが分かる．これは，GNU の基本例としてしっかりと作られており，コメントが多いものとなっているためと考えられる．

5.2 静的解析

静的解析とは，実際にプログラムを動かすことなく，ソースコードを調べてコーディング上の問題点を洗い出すことで，ソースコードを思考展開図で表すことの意義に関して，主な利用者である開発者と学習者の 2 つの視点で考える．

開発者側に関しては，バグにつながるコーディングミスが見つかる，可読性や保守性を維持したコードになる等が期待される．思考展開図に表現した場合，他人に理解してもらいやすい，プログラムの全体構造が明確になる，対応するコメントが明確になる，充足率の検討やどういった部分にコメントがあるのか把握しやすい等の利点が考えられる．

学習者側に関しては，どういう動作をするのか，どういう方針でディレクトリが分割されているのか，プログラムがどういう作りになっているのか，どういう方針でファイルが分割されているのか等を把握しやすく，思考展開図に表現した場合，拡大縮小が可能でありコード単位で評価可能，関数同士の呼び出し関係を把握しやすい等の利点が考えられる．

6. あとがき

本研究では思考展開図に空欄のノードを設けることを積極的に認めているが，これは思考展開図の理念に反する．しかし，全て埋めてしまうと情報のないコメントや，実際のソースコードが見難くなってしまうということも考えられる．よって，適切な充足率を求める必要がある．

参考文献

[1] 間瀬他: 思考過程の思考展開図表現に基づく機械設計支援システム，人工知能学会誌, Vol.17, No.1, pp.94-103 (2002).
[2] 畑村洋太郎: 創造学のすすめ, 講談社 (2003).
[3] Murakawa, Kawasaki, et al.: Formulation of clamshell diagram and its application to source code reading, Proc. JCKBSE '08 (to appear).