ソースコードの可視化・可聴化による メタ認知的学習支援手法の研究 ーメディアアートの要素を取り入れたアプリケーション開発ー

岩田まどか[†1]　隼田尚彦[†2]　向田茂[†2]　齋藤一[†2]　安田光孝[†2]　大島直樹[†2]

これまでプログラムの可視化・可聴化を取り入れた学習支援手法の研究が多く行われてきた。しかし，メディアアートのような視覚的な魅力に着目した研究はあまり見られない。本研究では，ソースコードの可視化・可聴化を行うプログラミング学習支援システムと学習者との間に起こるインタラクションに着目した。この相互作用を通じて獲得されるメタ認知の仮説を示し，メディアアートのエッセンスを用いたアプリケーションのプロトタイプを開発した。

Learning Support Technique Focused on Metacognition Using Visualization and Auralization of Source Codes; Application Development Based on the Ideas of Media Art

MADOKA IWATA[†1]　NAOHIKO HAYATA[†2]　SHIGERU MUKAIDA[†2]
HAJIME SAITO[†2]　MITSUTAKA YASUDA[†2]　NAOKI OSHIMA[†2]

Various studies have been conducted to develop learning support techniques with source techniques of source code visualization and auralization. However, there are only a few researches focused on visual appeal like media art. This paper shows the hypothesis of metacognition through the interaction between the user and the learning support system with visualization and auralization of source codes. The prototype of the application using essences of media art was developed under the above hypothesis.

1. はじめに

これまで，プログラミング学習支援における様々な可視化・可聴化の研究が行われてきた．永原らの研究[1]では，PAD を用いてプログラムのアルゴリズムを可視化させることで，アルゴリズム構造の理解を支援している．西田らの研究[2]では，日本語ベースのプログラミング言語で基礎を学ばせる．またプログラムの実行速度を調節して，可視化されたプログラム実行の様子をじっくり観察できる．そのほか，寺田が開発したプログラムの実行トレースシステム ETV[3]は，トレースしたプログラム実行の様子を視覚的に表示する．

多くは変換対象の全体的な役割や時系列などプログラム構造の明確化を重視し，目や耳を楽しませることにも重点をおいた形態はあまりみられない．プログラムの可聴化にエンターテイメント性を持たせた先行事例として CodeMusician[4]と CodeDrummer [5]を発展させた佐藤和哉の研究[6]が挙げられるが，同じ音が繰り返され単調で飽きてしまう問題があった．プログラミングの可視化・可聴化はある程度パターンが固定化されてしまうので，エンターテイメント性を持たせるならば飽きにくさに留意する必要がある．

そこで本研究では学習支援だけでなく，メディアアートを意識してエンターテイメント性を備えた可視化・可聴化の手法を探る．メディアアートとは，一般的にメディア，特にビデオやコンピュータなどの技術を用いたアートのことである．メディアアートの中には，ソフトウェア自体をアート作品とするものやソースコードを活用した作品も存在する．

exonemo が公開している DISCODER[7]は読み込んだウェブページの URL から疑似的なウェブページを生成し，その HTML コードをキーボード操作によって破壊する．佐藤和哉の CodeMusician はプログラムの可聴化を試みる作品である．変数値の遷移と実行位置の判断が音として生成される．

これらのメディアアートで頻繁に取り入れられるインタラクティブ要素を活用し，メタ認知的に学習させる方法を検討する．プログラム実行中，ユーザ介入がリアルタイム反映されることで，どのようにメタ認知に繋がるのかを考えることとした．そこで本稿では，研究のための前段階として制作したソースコードの可視化・可聴化を試みたプロトタイプの開発を報告する．

2. 作品概要

2.1 「マリンスノウ」の概要

本作品「マリンスノウ」は，読み込まれたソースコード

†1 北海道情報大学大学院経営情報学研究科
Graduate School of Management and Information Sciences, Hokkaido Information University
†2 北海道情報大学情報メディア学部
Faculty of Information Media, Hokkaido Information University

から対象単語を取得して，深海をモチーフとしたアニメーションに変換する．アニメーションは表示されたソースコードに含まれる対象単語によって異なる．

変換アニメーションは，ソースコードの流れを円の回転・移動・変形で表現する．円のオブジェクトを多用したのは，円からソースコードのそれぞれの処理の繋がりをイメージしているためである．

2.2 画面構成

画面構成を図 1 に示す．

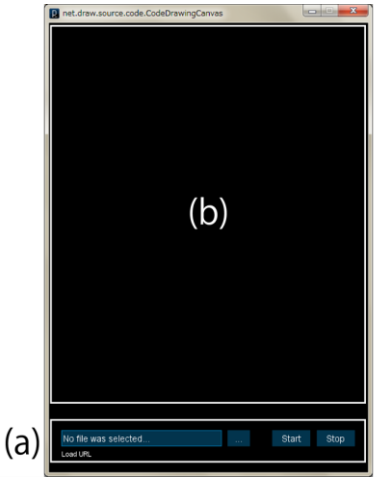

図 1　画面構成

Figure 1　Screen structure.

図中の(a)は，コントロールボタン群であり，ソースコードを読み込むためのファイル選択ダイアログや，ソースコード変換の実行・停止ボタンが配置されている．(b)は，変換アニメーション表示部で，読み込んだソースコードを視聴覚表現に変換し，アニメーションとして表示する．

2.3 ソースコードの可視化

ソースコード可視化の流れは図 2 に示すとおりである。まず，ソースコード変換前の準備を行う。ファイル選択ダイアログから変換したい PHP ファイルを選択し, Start ボタンをクリックする。②のように変換の開始を合図するアニメーションが流れ，中央に出現する白い円では PHP のファイル名が表示される。

変換アニメーションを開始すると，ソースコードの内容が中央の円では単語ごとに，変換アニメーションの下では一文ごとに表示される。中央の円に変数名が表示されると，円の周囲に光のパーティクルを生成する。④のように，パーティクルにカーソルを合わせることで，変数の名前と出現回数が表示できる。出現回数が増えるほど，パーティクルの色が変化する。

後述するブロック構造として判断されると，新規の階層として新たな円を生成する。ブロック構造内の変換処理は，新しい円を軸として表現される。ブロック構造の変換アニメーションは⑤で示すように 3 種類ある。

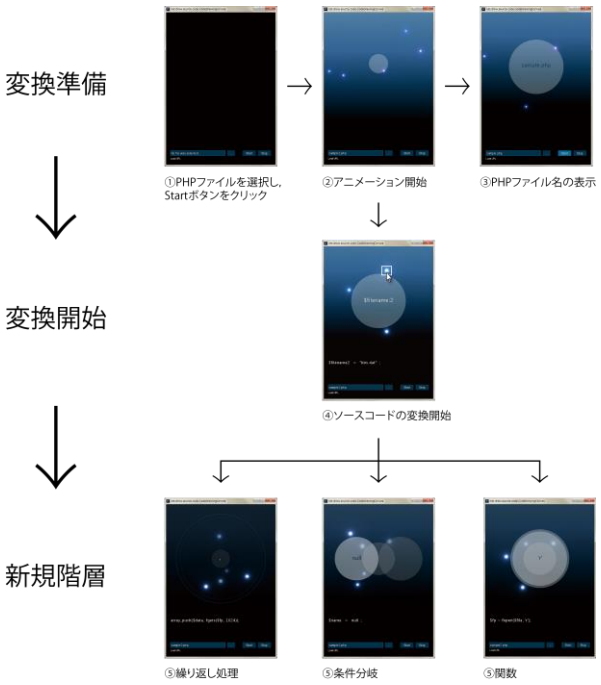

図 2　ソースコード可視化の流れ

Figure 2　Flow of source code visualization.

2.4 開発環境・使用言語

本作品は Java で制作し，グラフィカルな表現には Processing[8]を使用している．Processing は，グラフィック描写やインタラクティブな操作を Java よりも平易なコードで実装できる．ボタン・テキストフィールドなどの GUI については，Processing ライブラリである controlP5[9]を利用している．

ソースコードの構文解析については，パーサジェネレータである ANTLR[10]を使う．ANTLR はコンパイラやインタプリタ，他言語への変換プログラムを生成できる．

ソースコードの変換対象言語は PHP とする．変数に$記号を必要とし関数の定義が function で始まることから，構文解析時に処理しやすいと考えたためである．

3. 構文解析

3.1 変換対象

アニメーションへの変換対象は「変数」「関数」「制御文」の 3 つとする．

変数

今回の変換言語は PHP であるため，先頭に$マークがつく単語を変数として判断する．

関数

ユーザが独自に定義したユーザ定義関数を対象とする．ユーザ定義関数は function 以降に続く単語を関数名として判断する．

制御文

条件分岐（if・switch 文），繰り返し処理（while・for 文）

が対象となる．

関数や制御文はブロック構造であるため，全体のソースコードとは異なる階層として識別される．

3.2 ANTLR の概要

ANTLR は解析したい言語の構文規則が記述された文法ファイル「Grammar」を読み込み，Lexer・Parser のソースファイル，Lexer で分割されたトークンをまとめた tokens ファイルを生成する．

Lexer では，入力された文字やバイトストリーム（例：文字列，バイナリデータなど）を読み込み，文法ファイルに記述されている構文パターンを使用してトークンに分割する．そして，分割したトークンからトークンストリームを生成する．Parser では，トークンストリームを読み込み，対象言語で記述されたルール（パターン）のフレーズに一致する各フレーズ（もしくはサブフレーズ）のセマンティックアクションを実行する．[10]

ANTLR は LL 法を採用している．

3.3 変換対象の解析

構文解析の流れを図 3 に示す．

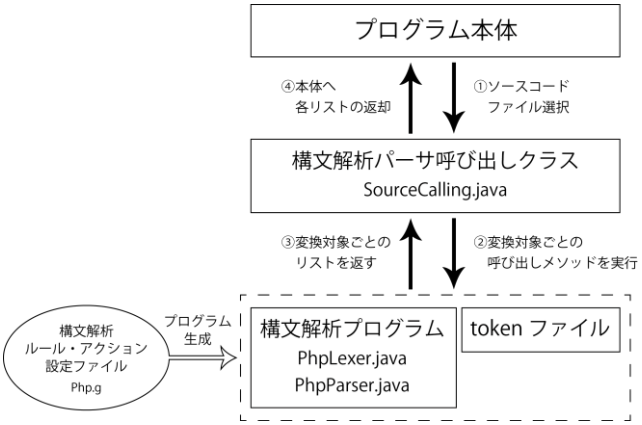

図 3　構文解析の流れ

Figure 3　Flow of parsing.

Grammar は, ANTLR 公式サイトの GrammarList から PHP Parser[11]を使用する．Grammar ファイルは「Php.g」をカスタマイズし，組み込み方法は「php-parser.zip」を参考にした．

パーサプログラムから単語を取得するため，変換対象ごとに一致させるルールを定義してアクション実行時に配列へ文字列を追加している．ANTLR から生成したパーサを使用するには，パーサ呼び出し用のクラスが必要になる．パーサプログラムから変数，関数，制御文が収納された配列を取得する．

4. クラス設計と視聴覚変換

システムのダイアグラムを図 4 に示す．

プログラムを実行すると，メインプログラムであるクラス「CodeDrawingCanvas」から以下のクラスが呼び出される．

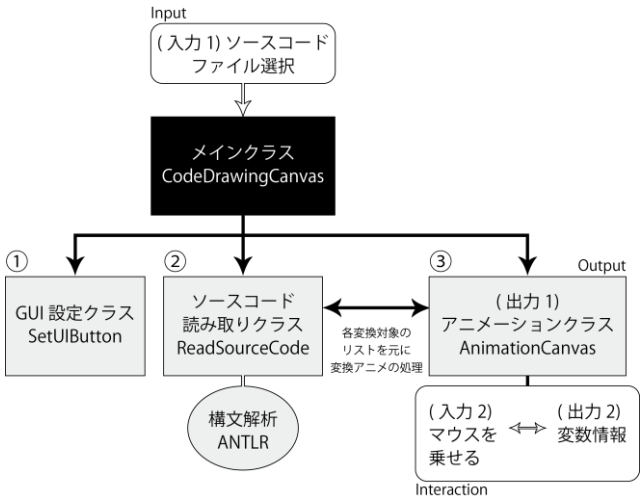

図 4　システムダイアグラム

Figure 4　System diagram.

- GUI 表示を設定するクラス「SetUIButton」
- ソースコードの読み取りクラス「ReadSourceCode」
- アニメーション処理を行うクラス「AnimationCanvas」

このほか,「ReadSourceCode」クラスから構文解析されたパーサ呼び出しクラス,「AnimationCanvas」クラスからソースコードの階層収納クラス，可視化すると中央に表示される円生成クラス，パーティクル生成・動作クラスをそれぞれ呼び出している．

4.1 メインクラス「CodeDrawingCanvas」

メインクラス「CodeDrawingCanvas」は，画面サイズや背景の設定, メインクラスから各クラスの処理の呼び出し，ボタンを押したときの処理などを行う．プログラムを起動すると,「main」メソッドのあるこのクラスを最初に実行する．Processing の機能を使用するために，ライブラリとして Processing を読み込み，「PApplet」を継承している．「main」メソッドでは，Processing で定義されている main メソッドを呼び出している．「CodeDrawingCanvas」のパッケージ名を引数として設定する．この設定により，Processing のプログラミング様式で記述することが可能となる．

4.2 GUI 設定クラス「SetUIButton」

GUI 設定クラス「SetUIButton」は，ボタンやテキストフィールド，ファイル選択ダイアログなどの GUI を表示する．メインクラスと同じく，Processing のインポートと PApplet の継承を行う．また，ボタン・テキストフィールドを設置するためのライブラリ「controlP5」もインポートする．

4.3 ソースコード読み取りクラス「ReadSourceCode」

ソースコード読み取りクラス「ReadSourceCode」は，ファイル選択ダイアログで選択されたファイル名からソースコードを読み取り文字列を取得する．取得文字列を単語ごとに分割し，各変換対象と照らし合わせる．変換対象ごと

に配列を用意し，一致する単語を配列に収める．メインクラスと同じく，Processing のインポートと PApplet の継承を行う．

4.4 アニメーションクラス「AnimationCanvas」

アニメーションクラス「AnimationCanvas」は，ソースコードを変換したアニメーションと効果音の処理を実行する．メインクラスと同じく，Processing をインポートしている．

(1) メタ認知を促す仮説

変換対象とする変数・関数・制御文のアニメーションがどのようにメタ認知につながるのか仮説を立てて説明する．

変数は(2)で後述する光のパーティクルを用いて表現される．ユーザはパーティクルにマウスを合わせたとき，中央に表示される文字情報からパーティクルが変数であることを認識する．パーティクルの色は変数の出現回数によって変化する．ユーザはパーティクルにマウスオーバーすることで，得られる文字情報の数値から出現回数であることに気づく．以降は文字情報がなくとも，パーティクルの色から変数の出現回数が多いか少ないかを判断できる．

制御文・関数は呼び出されるとき，必ず中央に制御文の種類や関数名が表示されてからアニメーションを実行する．中央の文字情報からユーザはそのアニメーションが繰り返し処理か，関数の呼び出しかを判断できる．一度認識すると中央の文字情報を見ていなくても，繰り返されるアニメーションから現在どの処理が行われているのかを理解できると考えた．

また，制御文・関数を呼び出すときは背景が暗くなる．それまで実行していたアニメーションの明度を落として前面に新しく展開されるアニメーションから，ユーザに異なる階層へ移行したことを推察させる．明度が落ちるほど，ユーザは現在の階層が深いと判断するようになると考える．

(2) 視覚表現

図 2 に示したソースコード可視化の流れの詳細を説明する．アニメーション開始のフラグが立つと，背景画像の移動と同時にパーティクルのアニメーションが開始する（図 5）．

背景画像は，アニメーションの表示領域よりも上下に大きいグラデーション画像を用意する．表示させる座標を上へずらすことで，グラデーションが徐々に暗くなる．グラデーションの濃度を上げることで，沈んでいくような雰囲気を示唆させる．

背景の移動に合わせて，画面上部から光のパーティクルを降らせる．パーティクルが画面外に移動したとき，XY 座標をある程度ランダムに設定し，画面上部のランダムな位置へ移動させる．これらの処理をパーティクルの要素分，アニメーション開始フラグが変更されるまで繰り返す．

パーティクルのアニメーションと同時に，白い小さな円が中央へ移動して広がる．そして，中央の円にソースファイル名が浮かび上がる（図 6）．

図 5　パーティクルアニメーション

Figure 5　Particle animation.

図 6　ソースファイル名の表示

Figure 6　Display of source file name.

図 7　変数の変換アニメーション

Figure 7　Transformed animation of variable.

ソースコードの変換アニメーションが始まると，図 7 のように円の中央へ単語ごとにソースコードの内容が表示される．表示される単語が変数であれば，光のパーティクルが生成される．生成されたパーティクルは，円の周囲を拡大･縮小しながら回る．パーティクルにマウスを乗せると，そのパーティクル情報を取得し「変数名：出現回数」を表示する．

関数，もしくは制御文のブロック構造の読み込みが始まると，画面のアニメーションに変化が起こる．ブロック構造のアニメーションは，アルファ値を指定して画面の明度を落とす．また，これまで変換アニメーションを行っていた全ソースコードのアニメーションを縮小して表示する．サイズの縮小と画面の明度を落とすことで，奥行きのある空間をイメージさせる．

繰り返し処理の場合，中央の円が波紋となって広がるループアニメーションが実行される（図 8）．条件分岐では中央に現れた円が 2 つに分裂した後，片方の円で変換アニメーションが実行される（図 9）．ユーザ定義関数の場合，2 つ出現した円の一方が拡大・縮小を繰り返すアニメーションを実行する（図 10）．

(3) **聴覚表現**

本アニメーションの効果音の再生処理には，Processing に用意されたライブラリ「Minim」を利用した．パーティクル生成，もしくは出現回数，関数や制御構造の実行に応じて効果音を再生させる．

5. おわりに

本稿では，プログラミング学習支援の可視化・可聴化において，インタラクティブ要素を取り入れたエンターテイメント性とソースコードの視聴覚表現の手法を検討した．その上でプロトタイプを開発した．

今後は，どのような可視化・可聴化，もしくはインタラクティブ要素がメタ認知を促しプログラミング学習支援につながるのかを調査する．学習支援に対応する変換ツールとしてプログラムを見直し，機能の追加や修正を行う．プログラミング学習支援としての効果を調べるために，アンケートや撮影による行動観察に加え，アイカメラを使用した停留点の計測を行い，メタ認知の働きを検証する予定である．

謝辞　本制作にあたり，ご助力くださった教員，ゼミの方々に感謝の言葉を申し上げます．

参考文献

[1]　永原功策, 桑田正行:PAD エディタを利用した C プログラミング学習支援システム構築に関する研究, 情報処理学会研究報告, Vol.2004, No.13, pp.17-24(2004).
[2]　西田知博ほか:初学者用プログラミング学習環境 PEN の実装と評価, 情報処理学会論文誌, Vol.48, No.8, pp.2736-2747(2007).

図 8　繰り返し処理のアニメーション
Figure 8　Loop animation.

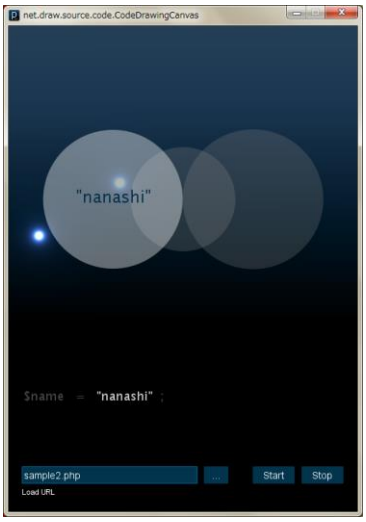

図 9　条件分岐のアニメーション
Figure 9　Conditional branching animation.

図 10　ユーザ定義関数のアニメーション
Figure 10　User-defined functions animation.

[3] 寺田実, “ETV”, http://pr.ice.uec.ac.jp/ETV/index-j.html.(2013/8/9 参照)
[4] 佐藤和哉ほか:CodeMusician : プログラム実行可聴化の試み, 第 3 回エンターテイメントと認知科学シンポジウム, pp.52-55(2009).
[5] 佐藤和哉ほか:CodeDrummer : プログラム実行における関数呼び出しの可聴化手法, 情報処理学会研究報告, Vol. 2011-MUS-89, No.14, pp.1-6(2011).
[6] 佐藤和哉:プログラム実行における関数呼び出しの可聴化手法に関する研究, 電気通信大学平成 22 年度修士論文(2011).
[7] exonemo, “DISCODER”, http://exonemo.com/DISCODER/indexJ.html.(2013/8/9 参照)
[8] Processing, http://processing.org/.(2013/8/9 参照)
[9] controlP5, http://www.sojamo.de/libraries/controlP5/.(2013/8/9 参照)
[10] ANTLR, http://www.antlr.org/.(2013/8/9 参照)
[11] PHP Parser, http://code.google.com/p/phpparser/.(2013/8/9 参照)
[12] Ben Fry(増井俊之監訳, 加藤慶彦訳):ビジュアライジングデータ―Processing による情報視覚化手法, オライリージャパン(2008).
[13] Casey Reas, Chandler McWilliams, LUST(久保田晃弘監訳, 吉村マサテル訳):FORM+CODE―デザイン/アート/建築における,かたちとコード, ビー・エヌ・エヌ新社(2011).