108107: 超好意的解釈コンパイラによりかかったプログラマのための激好意的解釈コンパイラ

中橋 雅弘[1,a)]　加藤 邦拓[2]　宮下 芳明[2,3]

概要：これまで著者らは，心が折れやすいプログラミング初心者のために，多少のミスでも大目に見て好意的に解釈するコンパイラを開発してきた．しかし，この好意的解釈を逆手にとって簡略なソースコードを書き，高速にプロトタイピングを行う中級プログラマが現れた．彼らが書くのは，コンピュータに対して細かく何をどうしろとは言わない，いい加減なソースコードである．行末にセミコロンを打つこともなく，開いた括弧を閉じることもなく，命令語も最初の数文字しか書かない．それどころかそもそも命令を書かなかったりする．本稿では，彼らがさらにスピーディーにコーディングを行えるよう好意的解釈コンパイラを改良し，たとえばファイル名が記されていたらその拡張子によって適切な命令を推測するようにした．また，1 行入力するたびにコンパイル・実行を行ってサムネイルを表示するヒストリウィンドウを用意し，システムによって行われた好意的解釈全てを，容易に修正できるようにした．さらに，複数の解釈が考えられる場合でも，その選択を保留にしたままコーディングを続けられる内包型タブインタフェースを導入した．これにより，一貫性がなくずぼらで優柔不断であっても，スピーディーに実装を行える新しいプログラミングスタイルを提案できると考えている．

108107:The Extreme Generous Compiler for Programmers that rely on the Super Generous Compiler.

Masahiro Nakahashi[1,a)]　Kunihiro Kato[2]　Homei Miyashita[2,3]

Abstract: We had developed the compiler that tolerate small mistakes favorably interpret the program for the programming beginners who tends to be disheartening. However, taking advantages of this favorable interpretation, the intermediate programmers who do fast prototyping by writing easy codes appeared. What they write is inaccurate codes that do not instruct computer what to do in details. They do not put a semicolon at the end of the line, do not close the brackets, write only leading a few letters in the commands, on the contrary they sometimes do not write even commands. In this paper we improved the favorable interpreting compiler to make coding much faster. For example the compiler speculates the relevant commands from the extension of the file name. Furthermore this system has a history window that shows the thumbnail images of the result of compile and execution at the every change of the program in 1 line unit so user can easily modify all the favorable interpreting executed by the system. By an internal tabbed Interface was introduced, user can continue coding by leaving the option open even in cases that several interpreting can be considered. We are now able to propose a new programming style that can be executed speedy even by the users who are inconsistent, inaccurate and indecisive.

[1] 明治大学 理工学研究科 新領域創造専攻 ディジタルコンテンツ系 Program in Digital Contents Studies, Program in Frontier Science and Innovation, Graduate School of Science and Technology, Meiji University
[2] 明治大学 理工学部 情報科学科
Department of Computer Science, Meiji University
[3] 独立行政法人科学技術振興機構, CREST
JST, CREST
[a)] bori42195@gmail.com

1. はじめに

「おい お茶」 祖父のその一言で祖母はそつなくお茶を淹れる．「おい あれはどこにある?」 その一言で祖母はハンコを持ってくる．文章に動詞や目的語が抜けていたとしても，祖父の状況や身振りから祖母にはなんとなくわかってしまうのだ．

2. HMMMML の好意的解釈コンパイラ

これまで著者らは，心が折れやすいプログラミング初心者のために，HMMMML (Homei Miyashita's Motivating Multilingual Markup Language) の開発を行ってきた [1], [2], [3], [4]．ユーザのプログラミングに対するモチベーションが下がる要因として “エラー” が挙げられる．HMMMML ではシステムが「好意的解釈」を行うことで，エラーを極力出さないように心がけ，ユーザのプログラミングに対するモチベーション維持を支援している．

HMMMML による好意的解釈具体例

・セミコロンの有無やドット抜け
・命令やタグの括弧閉じ忘れ
・if 文の条件式内での == と = の混同
・大文字小文字の混同
・配列変数を含む宣言忘れ
・全角スペースや 2 バイト文字の混入
・命令のスペルミス

スペルミスの具体例を挙げると，たとえば ptint “Hello” という文字列があったとき，これは print を意図して記述されている可能性が高いので，コンパイル時に print と置き換えて実行するようにした．HMMMML の言語仕様では，p で始まって文字列が続く命令が print しかないため，p “Hello” と記述しても print “Hello” と解釈される．これに対し，s “Hello” と書いた場合は，say(人工音声で読み上げ) と show(ダイアログ表示) の 2 通りが考えられる．このように複数通りの解釈がなりたつ場合は，その全ての解釈を実行することとした．実行される複数の解釈は，タブの切り替えで確認することができ，「採用」ボタンを押すことでその解釈を選択することができる (**図 1**)．

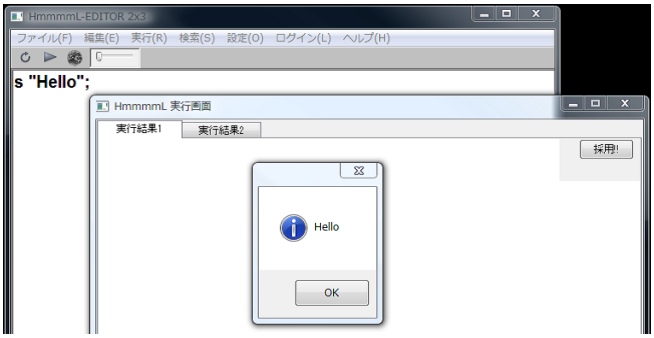

図 1 複数解釈によるタブ表示

たとえば　s “Hello”と書いた場合，タブ 1 を選択すると「Hello」が読み上げられ (say としての解釈) タブ 2 を選択すると「Hello」というダイアログが表示される)show としての解釈)．タブ 1 を選択して採用ボタンを押すと，ソースコードは**図 2** のように修正される．HMMMML では「早く実行結果が見たい」というユーザの気持ちを優先させまず実行する．そのあとで正しい表現が何であったかをコメントにより緩やかに提示し，ユーザの学習を促すインタラクションデザインとした．

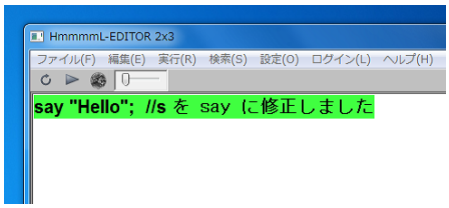

図 2 解釈結果の実行後提示

3. 好意的解釈によりかかったプログラマ

超好意的解釈コンパイラでは，命令を全て記述しなくても動作するため，それを逆手に取って簡略な記述でプログラムを作成する中級プログラマが現れた．彼らは好意的解釈を巧みに利用して高速にプロトタイピングを行う．

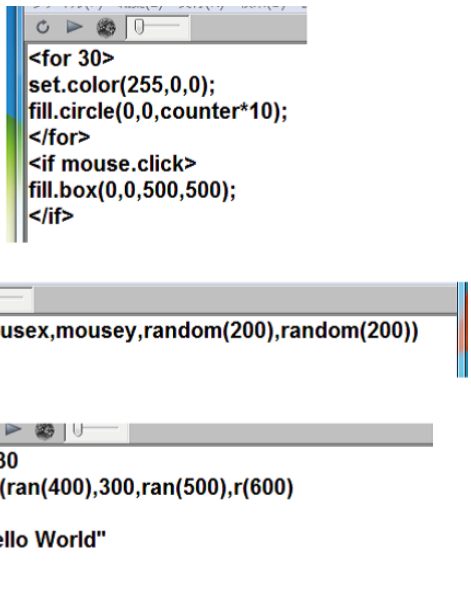

図 3 HMMMML の想定しているプログラム (上) 簡略化されたプログラム (中) よりかかったプログラム (下)

図 3 の上に示したのが，HMMMML での正式な命令を用いて書かれたプログラム例である．真ん中の例では命令のドットを記述せず，さらにはセミコロンも記述していない．下の例では一見するともはや何を書いているのかわからない．1 行目に記述された <f 30 はおそらく <for 30> のことを指示しており，閉じ括弧を書くことすらしていない．2 行目に記述された d.c は，d で始まりドットを含み，c をドットの後に含む命令を表していると推測できる．すなわち draw.circle のことである．引数として渡してい

る ran(400) は random(400) のことだと容易に予想できる．興味深いのは最後の引数として渡している r(600) である．このユーザはついに random を ran と書くことすらやめ，r の一文字で表現したものと思われる．3 行目の </ はその文字列から閉じタグのことであると推測できる．最後の “Hello World” は命令すら書かれていない．この場合は “文字列を使った何か” であることはわかるため，考えられる全ての解釈を実行する．好意的解釈コンパイラによって図 3 の下の例が訂正されたものが**図 4** である．最後の行はユーザが提示された複数の解釈の中から say を選択したため，say が追加されている．

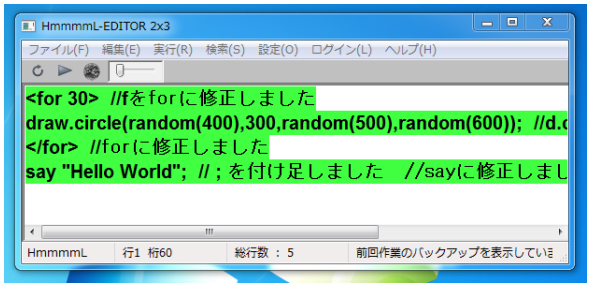

図 4 図 3 下の例の解釈結果

このように好意的解釈によりかかったユーザは命令を正確に記述することをせず，「これならば好意的に解釈してくれるだろう」という個人的な裁量のもとプログラミングを行うことがわかった．また，命令の記述の仕方にも一貫性がなく，その時々で変化することがわかった．

4. 激好意的解釈コンパイラ

本稿では，前章で述べたようなユーザがよりスピーディーにコーディングを行えるように，超好意的解釈コンパイラの改良版である激好意的解釈コンパイラを搭載した「108107」を提案する．以下では激好意的解釈コンパイラの特徴と，それに伴い追加された機能を述べる．

4.1 拡張子から推測

なによりもまず好意的解釈の強化を行った．前章で述べたようなユーザの一貫性のない命令記述にも，よりスマートに答えるように，引数の種類からその内容にまで，推測と解釈の範囲を拡張した．たとえば文字列を引数として受け取った際に，その内容が “img3.jpg”，“letitbe.mp3” となっていた場合，拡張子の内容からおのずと命令が推測できる．前者の jpg などの画像ファイルに対応した拡張子だった場合は，その画像を表示したいか，もしくはその名前で保存したいかのどちらかである可能性が高いと推測できる．このように，たとえユーザが命令を書かなかったとしても，引数の内容からある程度提示する実行結果の優先順位を決定することができる．

4.2 ヒストリウィンドウ

好意的解釈によりかかったユーザは，システムによる解釈を予測し，それを利用して簡略な記述を行う．しかし，ユーザが想定している好意的解釈結果と，システムによる解釈が異なることは多分に存在する．ここで新たに提案するのは，システムによって行われた好意的解釈全てをユーザが確認しながら，容易に編集・訂正できる機能である．それに伴い，エディタウィンドウの右側に新たにヒストリウィンドウを追加した (**図 5**)．

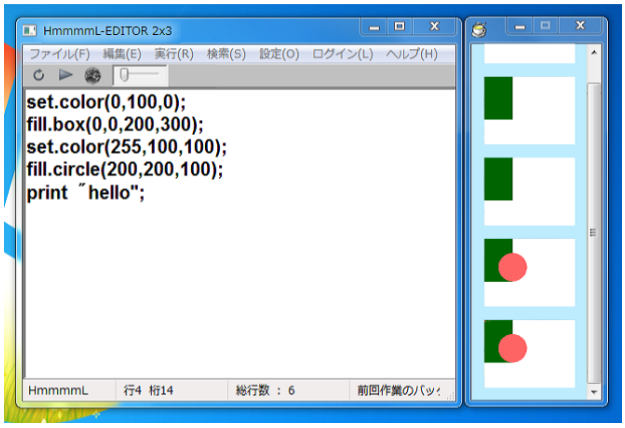

図 5 エディタ (左) とヒストリウィンドウ (右)

本稿第二著者らの提案する手法 [5] を用いて，ユーザが一行入力するごとに実行結果を記録し，ヒストリウィンドウに実行結果のサムネイルを表示する．ユーザがこのサムネイルにマウスオーバーすると，実行結果記録時にユーザが記述した行がハイライト表示される (**図 6**)．

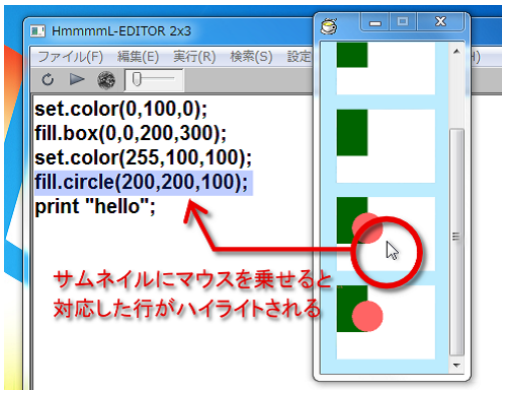

図 6 マウスオーバーによる対応行のハイライト

後の編集で，すでに記述されている行の内容を変更した場合，変更された行に該当するサムネイルは現在に影響がないと判断し，暗く表示される (**図 7**)．図 7 の例では円の色を set.color(255,100,100); で指定していた行を set.color(0,0,255); へと変更している．

実行結果を記録する際，好意的解釈による文章の訂正が行われた場合は，**図 8** のようにサムネイルを周辺の背景を

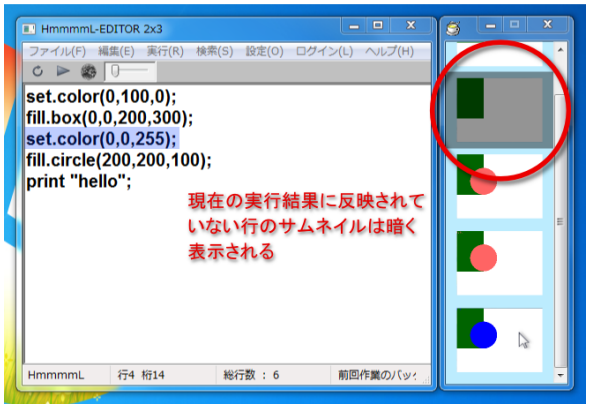

図 7　変更された行のサムネイル

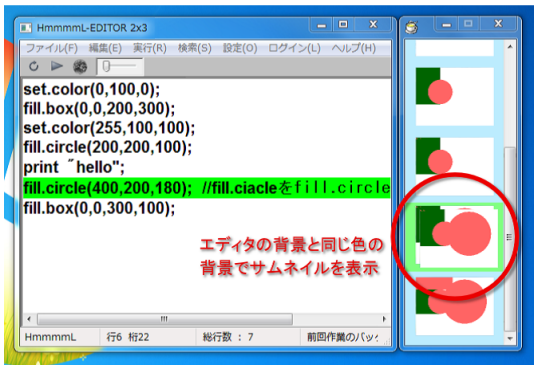

図 8　好意的解釈により訂正された行のサムネイル

ハイライトする．

さらに，好意的解釈が一意に定まらなかった場合は，ヒストリウィンドウ上のサムネイルを**図 9** のように，解釈が一意に定まった場合とは違った背景色でハイライトする．このように強調表示することで，好意的解釈が発生したタイミングをユーザが瞬時に判断できるようにした．

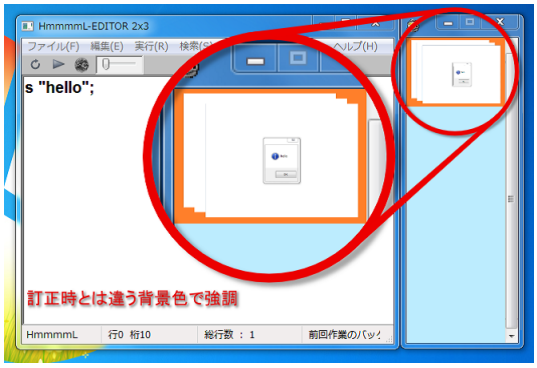

図 9　複数の解釈が発生した際のサムネイル

好意的解釈が行われたサムネイルにユーザがマウスオーバーすると，**図 10** のようにサムネイルが横に広がり，ユーザの望む解釈結果を選択できる．サムネイルを選択すると，エディタウィンドウの内容が選択されたサムネイルのソースコードに変更される．

このように，ユーザが想定していた解釈結果とシステムによる解釈が異なってしまった場合でも，ヒストリウィンドウのサムネイルを確認しながら，いつでもユーザが望んだ状態のソースコードにすることができる．

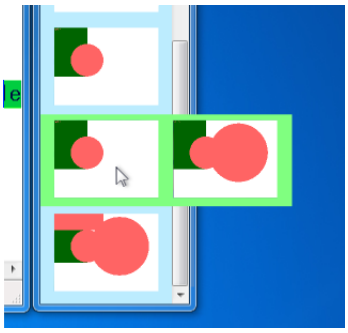

図 10　サムネイルの選択

4.3　複数解釈の保留

ユーザのスペルミスなどにより複数の解釈が考えられる場合，これまではその全てを実行した上で，どの解釈が正しかったのかをユーザにその都度尋ねていた．もしもユーザが解釈を選択せずに実行結果を閉じた場合，その時点でアクティブになっていたタブの解釈が自動的に採用され，エディタにその結果を反映していた．この機能はあくまでユーザのミスで複数の解釈が発生してしまった場合の処置として作成したものである．著者らは，あえて複数の解釈を発生させるプログラマが存在することを想定していなかった．ここでは彼らがよりスピーディーにコーディングを行えるように，複数解釈を保留にしたままプログラミングを続けられる機能を提案する．

複数の解釈を保留にした状態でプログラムを実行すると，保留にしてある解釈全ての組み合わせをを考慮し，実行結果をタブ表示する．つまり，ソースコード内の異なる箇所で 2 つの解釈と，3 つの解釈が存在していた場合，6 通りの実行結果を表示する．その際に**図 11** のような内包型タブで，組み合わせをわかりやすく表現するよう試みた．

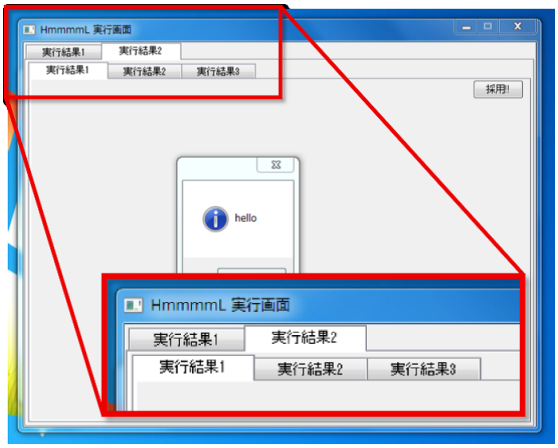

図 11　内包型タブインタフェースによる実行結果の表示

従来のプログラミングでは厳密で正確な記述により，1 つ

の実行結果を確認しながら開発を行うものだった．108107ではこれらの機能を追加したことで，複数の実行結果の可能性を確認しながら開発を行うことができる．また，命令を決定できない優柔不断なユーザであっても，「とりあえず君の解釈全部見せて」といったような対話的な使い方ができると考える．

5. 関連研究

プログラミング教育のための言語や，サポートシステムの開発はこれまでにも数多く行われてきた．一言にプログラミング教育といっても様々なものが存在する．プログラムの概念の部分だけの理解を重視することもあれば，C 言語や Java などの汎用的な言語に繋げられるような理解を重視することもある．その違いによって，対象の年齢層，指導方針は大きく変わってくる．以下では教育のためのプログラミング言語研究をとりあげ，HMMMML との目的やアプローチの違いなどを述べる．

学生がプログラミング言語に対して苦手意識を持つのは，プログラムが英単語で記述されているためであると考え，「なでしこ」「OONJ」などの日本語プログラミング言語も開発されている [6], [7]．なでしこは高校生以上をターゲットとし，専用のエディタと単純な文法を心がけることで，これからプログラミングを始めようという人に適した環境を目指している．OONJ は非情報系の科学技術諸分野の専門家が扱う対象世界をオブジェクト指向に基づいてモデリングし，それを自然日本語をベースとして記述するために提案された言語である．これらは，ソースコードのわかりやすさ，もしくは親しみやすさといった観点から日本語を採用している．HMMMML では「本格感」の演出のためには英語表記が望ましいと考え，あえて日本語ではなく一見難しい印象の英語を採用している．英語の理解度不足によるスペルミスなどの間違いは，好意的解釈により補っている．長らは対象を大学生とし，Java への移行を考えたプログラミング言語 Nigari の開発を行った [8]．Nigari は宣言を省略し，プログラムを自動的に可視化することで，プログラムの流れの直感的理解を促した言語である．HMMMML でも同様に，宣言などは付随的な要素であると捉え，省略を行った．また，システムを好意的な解釈から厳密な解釈まで変更できるスライダを設けることで，ユーザが自ら従来の言語へ移行していける仕組みを設けた．中村らは PEN と呼ばれるプログラミングをするためのサポートツールを備えたエディタを用いて，高等学校でのプログラミング教育導入を図った [9]．PEN にはプログラミングのステップ実行，一時停止をするためのボタンや，実行速度を調節するためのスライダ，プログラムの記述を支援するボタンなどが備わっており，他にも変数の内容をダイナミックに確認できるウィンドウが用意されている．本稿で新たに提案するヒストリウィンドウは，ユーザの編集を 1 行ごとに可視化し，さらにプログラムをあとからカスタマイズするためのサポートツールであるといえる．山下らによる Visual ℃ではコンピュータにかかる負荷をプログラムのソースコードにサーモグラフィ表示することで，プログラマに内省を促した [10]．HMMMML でもソースコードの背景色を直接変化させることで，訂正された部分にユーザの意識が向くようにしている．

これら 5 つの研究とは違い，よりビジュアルなインタフェースを用いて子供へのプログラミングの概念教育を考えたプログラミング言語が Scratch である [11]．Scratch はあらかじめ用意された簡単なスクリプトを，ブロックを組み立てるように組み合わせることでプログラミングを行うものである．これにより子供でも楽しんでいるうちに条件分岐，繰り返し，オブジェクト指向といった考え方が身につく．原田が開発したビジュアルプログラミング言語 Viscuit は，絵を描きマッチングを指定するだけでプログラミングを行うことができる [12]．マッチングには Fuzzy Rewriting を用いており，これまでのビジュアルプログラミング言語にはみられなかった柔軟な動きを実現した．これにより子供でもプログラミングの楽しさを容易に知ることができる． Viscuit はコンピュータの凄さを伝えることを目的としている．また，文部科学省により開発されたプログラミンでは，自分の描いた絵に「プログラミン」と呼ばれる生き物を積んでいくと，子どもでも簡単にプログラムを作ることができる [13]．プログラミンには「1 秒間で右に 100 進む」,「音をならす」など様々な種類があり，他にも「ずっとくりかえす」,「クリックされたら」などの制御命令を持ったプログラミンも存在する．プログラミンはプログラムの基本となる概念に子どもたちが自発的に触れ，楽しみながらルールを発見していくようデザインされている．これら 3 つの言語が比較的低年齢を対象としているのに対して，HMMMML は高校生や大学生のプログラミング初心者や，プログラミングに苦手意識を持っている人を対象としている．そのため親しみやすいビジュアルなプログラミング環境は避け，代わりに「本格感」を重視したテキストベースのプログラミング環境を採用した．

本稿で提案するヒストリウィンドウのように履歴やタイムラインを利用して，創作活動を支援する研究も数多く行われている．馬場らはビジュアルプログラミングシステムにおいて木構造を用いた UNDO 機能の提案をしている [14]．ユーザの間違った操作を元に戻す方法として，UNDO は非常に有効であるが，記憶できるのは最新の編集のみである．これではユーザが心おきなく試行錯誤を重なることができないと考え，木構造を用いてその解決を図った．大森らはソフトウェアの保守作業におけるプログラム理解のためには，現時点のソースコードを見るだけでは不十分であり，バージョン管理システムに格納されている過去のソースコードを見ても変更内容の詳細を知ることはで

きないとし，作業者の編集操作を全て記録することでプログラム変更の理解支援を行っている [15]．稲葉らはヒストリグラフを用いることでコマンドの再利用を可能とし，作業の試行錯誤支援を行っている [16]．

本稿第二著者の提案する時間とのインタラクションによるプログラミング支援では，ユーザの全ての作業課程を記録し，いつでも好きな過去に戻れるシステムを提案している [5]．ラピッドプロトタイピングにおいては，エラーの発見や修正などの試行錯誤に時間をかけることは本質的ではないとし，素早い開発に適した開発環境の提案を行っている．中小路らはキャンバスに絵を描いた際のストローク情報を記録し，それを再利用できるシステムを開発した [17]．これは過去の自分のストロークとインタラクションすることで創造的情報創出活動の支援を行うものである．Grossman らの Chronicle ではソフトウェアの各操作を映像として記録し，それらを操作履歴を示すタイムラインと対応付けることで，創作過程の閲覧を支援している [18]．本稿第一著者らの提案する創作時間そのものを利用したリミックスでは，他者の創作時間を組み合わせることで，新たな作品を生み出す環境を提案している [19]．タイムラインを操るシステムは，これまでにない新たな体験をもたらしてくれるが，閲覧性や操作が複雑になってしまうことは否めない．本稿で提案するヒストリウィンドウは，過去の改変を行うためのインタフェースではなく，システムにより好意的に解釈された複数の可能性を選択するためのインタフェースである．そのため，選択可能な箇所以外は過程を閲覧するまでにとどめることで，初心者でも扱いやすいインタフェースになるよう心がけた．

6. まとめ

本稿では，超好意的解釈コンパイラによりかかったプログラマ支援のために，激好意的解釈コンパイラを搭載する「108107」を提案した．これにより，命令の記述に一貫性がないずぼらなユーザや，命令を決定できない優柔不断なユーザであってもスピーディーに実装を行える新しいプログラミングスタイルを提案できると考えている．それは，厳密で正確な仕様を決めてから作り始めるプログラミングスタイルではなく，コンパイラと対話しながら一緒に作り上げていくプログラミングスタイルである．もちろん解釈を間違うこともある．間違えたら訂正すればいい，わからなければ聞けばいい．それは人からコンピュータに対しても，コンピュータから人に対しても同じではないだろうか．本稿で提案する激好意的解釈コンパイラは，決して「あなたの言葉はわからない　まず私の理解できる言葉で話せ」とはいわない．

参考文献

[1] 宮下芳明: プログラミングに対するモチベーションを向上させる新言語 HMMMML の開発, 第 51 回プログラミング・シンポジウム論文集, pp.57-64, 2010.

[2] 中橋雅弘, 宮下芳明: HMMMML2：超好意的に解釈するコンパイラ, 情報処理学会夏のプログラミング・シンポジウム報告集, No.2010, pp.107-110, 2011.

[3] 中橋雅弘, 宮下芳明: HMMMML3：他人を意識したモチベーション向上を考えたプログラミング環境, インタラクション 2011 論文集, 2011.

[4] 宮下芳明, 中橋雅弘: 学習者のモチベーション向上のための好意的解釈を行うフィジカルコンピューティング環境のデザイン, ヒューマンインタフェース学会論文誌, Vol13, No.4, pp.303-313, 2011.

[5] 加藤邦拓, 宮下芳明: 時間とのインタラクションによるプログラミング支援, 情報処理学会研究報告 HCI, HCI-149, No.2, pp.1-6, 2012.

[6] 日本語プログラミング言語「なでしこ」 http://nadesi.com/

[7] 畠山正行, 松本賢人: オブジェクト指向自然日本語記述言語 OONJ の設計とその記述例, 情報処理学会研究報告 HPC, HPC-102, No.57, pp.13-22, 2005.

[8] 長慎也, 甲斐宗徳, 川合晶, 日野孝昭, 前島真一, 筧捷彦: Nigari-Java 言語へも移行しやすい初学者向けプログラミング言語, コンピュータと教育研究会報告, pp.13-20, 2003.

[9] 中村亮太, 西田知博, 松浦敏雄: 高等学校での「プログラミング」教育の導入：PEN を用いて, 情報処理学会研究報告, pp.41-47, 2008.

[10] 山下洋毅，宮下芳明: Visual ℃: プログラム負荷に対する内省を促す可視化手法, インタラクション 2010 論文集, 2010.

[11] M. Resnick, J. Maloney, A. Monroy-Hernndez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai: Scratch: Programming for all. Communications of the ACM, Vol.52, No.11, pp.60-67, November 2009.

[12] 原田康徳: 子供向けビジュアル言語 Viscuit とそのインタフェース, ヒューマンインタフェース研究会報告, pp.41-48, 2005.

[13] プログラミン http://www.mext.go.jp/programin/

[14] 馬場昭宏, 田中二郎: ビジュアルプログラミングシステムのための UNDO 機能, 日本情報処理開発協会 ビジュアルインタフェースの研究開発報告書, pp.176-187. 1994.

[15] 大森隆行, 丸山勝久: 開発者による編集操作に基づくソースコード変更抽出, 情報処理学会論文誌, Vol.49, No.7, pp.2349-2359, 2008.

[16] 稲葉由倫, 渋谷雄, 辻野嘉宏, 西田知博: ヒストリグラフを利用したコマンドの再利用の提案と評価, 情報処理学会研究報告 HI, HI-149, No.7, pp.1-8, 2012.

[17] 中小路久美代, 山本恭裕: 創造的情報創出のためのナレッジインタラクションデザイン, 人工知能学会誌, Vol.19, No.2, pp.235-246, 2004.

[18] T. Grossman, J. Matejka, and G. Fitzmaurice: Chronicle: capture, exploration, and playback of document workow histories. In Proceedings of the 23nd annual ACM symposium on User interface software and technology, UIST ’10, pp.143-152. ACM, 2010.

[19] 太田佳敬, 中橋雅弘, 宮下芳明: 創作時間そのものを利用したリミックス, 情報処理学会研究報告 HCI, HCI-148, No.2, pp.1-6, 2012.