情報処理学会論文誌　プログラミング　Vol.6　No.2　69–84 (Aug. 2013)

フォルダ・プログラミング環境における エンドユーザインタフェースに関する一考察

赤間 浩樹[1,a)]　鶴岡 行雄[1,b)]　佐藤 哲司[2,c)]

受付日 2012年11月9日, 採録日 2013年4月24日

概要：近年，画像・映像・音声・構造化テキストなど様々なメディアに対する処理部品がオープンソースとして公開されてきている．これらの処理部品を使用して所望の変換を実現するには，複数の部品を組み合わせた処理を繰り返すなどのプログラミングが必要とされてきたが，これは，写真やビデオなど大量のメディアデータを所有するエンドユーザにとっては大きな負担となっていた．そこで，我々は，日常的にプログラミングを行わない PC 利用者にも利用が容易なプログラミング環境 POLDER を提案してきた．PC のデスクトップ上で広く普及しているフォルダ概念を拡張した POLDER は，既存のメディア処理部品を容易にマッシュアップできる環境であり，処理を大量のデータに適用することが簡単に行える．本論文では，POLDER が提供しているプログラミング環境であるフォルダ・インタフェースの効果を調査することを目的に，コードを記述する従来型のテキスト・インタフェースとの比較を行う利用者実験を行っている．その結果，実験参加者のスキルや課題の複雑さによって両インタフェース間で作業時間などに差異が観測されたので報告する．

キーワード：ビジュアル・プログラミング，データフロー・プログラミング，パラダイム

Study of End User Interfaces for a Folder Programming Environment

Hiroki Akama[1,a)]　Yukio Tsuruoka[1,b)]　Tetsuji Satoh[2,c)]

Received: November 9, 2012, Accepted: April 24, 2013

Abstract: Recently, various media processing components for photos, video, voice, and structured text data have been provided as open source software. Users must learn programming skills using programming environments in order to combine and apply these components iteratively to many photos and videos. However, users who have a large quantity of media data may find it difficult to learn and use these programming environments. We proposed the folder programming environment “POLDER,” which is easy for novices who rarely do programming to use. It is based on enhancing the folder concept that prevails on the PC desktop. End users can easily mashup media components in this environment, which simply integrates existing components and to which many forms of data can be applied. To confirm the effectiveness of the POLDER interface, we conducted experiments where users compared the POLDER interface to a traditional text interface that described the code. The experimental results show a difference in the skill level of the subject and the complexity level of the problem between the folder interface and a corresponding text interface during the course of the experiments.

Keywords: visual programming, data flow programming, paradigm

[1] 日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ
NTT Software Innovation Center, NTT Corporation, Musashino, Tokyo 180–8585, Japan
[2] 筑波大学図書館情報メディア系
Faculty of Library, Information and Media Science, University of Tsukuba, Tsukuba, Ibaraki 305–8550, Japan
a) akama@acm.org
b) tsuruoka.yukio@lab.ntt.co.jp
c) satoh@ce.slis.tsukuba.ac.jp

1. はじめに

デジタルカメラなどの高性能化・低廉化にともなって，大量のメディアデータなどを所有する生活者が増えている．一方，Web 上には，ImageMagick [1]，FFmpeg [2]，OpenCV [3] など多数の魅力的なメディア処理部品がオープンソースとして公開され，それらを API として提供す

る Web サービス [4] も広がりつつある．さらに，Web やクラウド上のデータやサービスをマッシュアップ [5] することで，プログラマが様々なシステムやサービスを構築できるプログラミング環境も提供されつつある．すでに提供されているマッシュアップ環境としては Yahoo! Pipes [6] や JackBe Presto [7] などがある．これらのマッシュアップ環境は Web 上の XML，RSS，JSON といった構造化データの加工や，複数情報の画面上への統合配置などを可能にしている．これらの既存のマッシュアップ環境であっても，Web 上のメディア処理部品の組合せ処理を大量のデータに繰り返し適用するには，プログラムの編集・実行・結果の確認などプログラミングに関わるスキルの修得が必要になる．このため，一般にプログラミングとは疎遠である多くの生活者にとっては，Web 上の多様なメディア処理部品を簡単に組み合わせて，撮り貯めした写真や映像などを様々に加工し活用したいと考えても，障壁が高く活用することは難しかった．このように，より多くの生活者が Web 上のメディア処理部品を組み合わせた処理を繰り返し適用できる簡便な活用環境を提供することが課題となっている．

筆者らは，PC のデスクトップ上で広く使われているフォルダ・インタフェースを使ったフォルダ・プログラミング環境 POLDER [8], [9] を提案している．フォルダとは，データの格納単位であるファイルを階層的に整理・管理する広く知られた手法であり，フォルダ・インタフェースはデスクトップの一部として多くの PC 利用者に抵抗なく受け入れられている．このフォルダの概念を基本とする POLDER を用いることで，ふだんプログラミングなど行わない生活者でも，新たなスキル習得を最小にしながら，Web 上で提供される各種のメディア処理部品を組み合わせて自分の目的とする処理を構成し，自分の所有するメディアデータに一括適用できるようになると期待できる．本論文では，POLDER が提供しているプログラミング環境であるフォルダ・インタフェースの効果を調査することを目的に，従来型のコードを記述するテキスト・インタフェースとの比較を行う利用者実験を行っている．また，ユーザビリティの観点でオープンソースのビジュアル・プログラミング言語 Pure Data との比較を行った．

本論文の構成は以下のとおりである．まず，2 章で我々の提案する POLDER を簡単に解説する．3 章では比較対象となるテキスト・インタフェースについて紹介し，36 名の大学生を対象に今回行った比較実験と，その実験結果およびアンケート結果を示し，実験参加者の属性などと比較しながら分析を行い，フォルダ・インタフェースとテキスト・インタフェースとの違いを考察する．4 章で POLDER とビジュアル・プログラミング言語 Pure Data との比較を行う．数名の情報処理技術者を対象とした実験を行い，エラー分析の結果などからユーザビリティ原則についての比較評価を示し今後に向けた課題抽出を行う．5 章ではビジュアル・プログラミング関連研究との比較を行い，6 章でまとめと今後の研究の方向性について示す．

2. フォルダ・インタフェースを使ったプログラミング環境 POLDER

フォルダ概念に基づいて設計・実装した POLDER [8], [9] の概要を，利用者実験で実験参加者に提示した資料を使って簡略に述べる．

2.1 処理付きフォルダ

ファイルの格納と整理を目的とする一般的なフォルダもしくはネットワーク・フォルダでは，利用者がファイルを Drag & Drop することで Drop したフォルダの中にファイルがそのまま格納される（図 **1** 上段）．一方，POLDER の要素的機能である処理付きフォルダでは，ファイルを Drag & Drop した際，そのフォルダ名に対応させた処理が実行され，その結果がファイルとしてフォルダ内に生成される．ここで，フォルダ名と処理の対応付けは，フォルダ名の一部をキーとしてあらかじめ設定されているとする．特定の処理を対応付けたフォルダを処理付きフォルダと呼び，図 1 下段の例では，処理付きフォルダ getFaces によって顔抽出された結果の顔画像写真が getFaces フォルダの中に生成される様子を表している．

2.2 フォルダ・プログラミング

処理付きフォルダを複数組み合わせて作成したプログラムをフォルダ・プログラムという．普通のフォルダと同様に，処理付きフォルダは，そのフォルダ内にさらなる処理付きフォルダの構造を持つことができる．フォルダ・プログラムは連鎖，並行，条件分岐および抽象化などの処理フローから構成される．POLDER はフォルダ・プログラムを構成し実行する環境である．具体的なフォルダ・プログラムの応用事例は文献 [8], [10] に詳しく述べられているので，以下では基本的なフォルダ構造を紹介する．

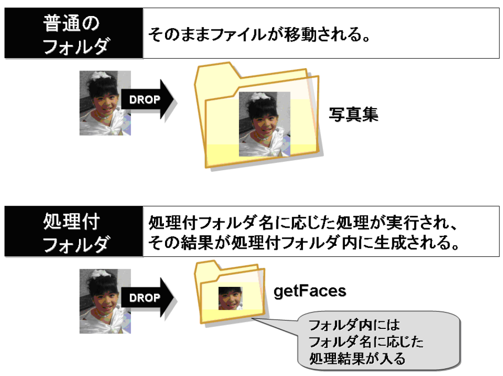

図 **1**　普通のフォルダと処理付きフォルダ
Fig. 1　Original folder and functional folder.

図 2　処理の連鎖と応用

Fig. 2　Process chain and application.

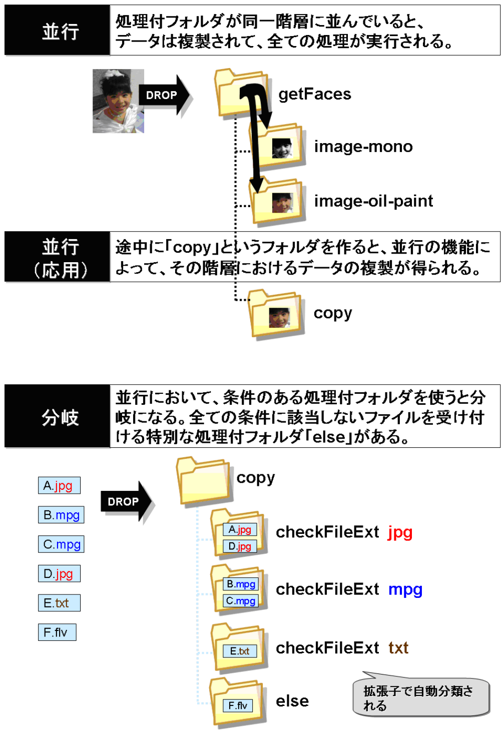

図 3　並行処理と応用

Fig. 3　Concurrency and application.

2.2.1　連鎖

処理付きフォルダの中に，さらに別の処理付きフォルダが存在する場合には，上位の処理付きフォルダの処理結果が下位の処理付きフォルダの入力となって処理が連鎖する．図 **2** 上段の例では，最上位層の処理付きフォルダ getFaces に Drag & Drop された画像は，getFaces によって顔画像が抽出され，image-resize によってサイズが変更され，image-frame によって茶色の外枠が付与された画像として，最下層の処理付きフォルダ内に生成される．図 2 下段の例はこの連鎖の応用を示している．処理付きフォルダ階層の途中から下位を実行させることや，階層の途中までを実行させることなどが簡単に制御できる．

2.2.2　並行・条件分岐

図 **3** 上段に示すように処理付きフォルダの中に，処理付きフォルダが並行の階層として存在する場合には，それらすべてが並行して実行される．その際，上位の処理付きフォルダの処理結果は必要な数だけ複製されて下位の処理付きフォルダの入力として渡され，下位の処理付きフォルダで実行された結果は各々の処理付きフォルダに格納される．この応用例として，実効的には何も処理をしない処理付きフォルダ copy を並行する階層に作成することで，その copy フォルダ中に処理直前のファイルを保存することもできる．図 3 の下段に示す条件分岐は，並行の特殊な応用として実装されているが，本論文で述べる利用者実験では使用しないので説明は省略する．

図 4　複数ファイルへの一括処理

Fig. 4　Processing two or more files in bulk.

2.2.3　一括

フォルダ間で複数ファイルを移動，コピーする際に，それらのファイルを一括選択しておき，1 回の Drag & Drop 操作で実行することができる．処理付きフォルダにおいても，同様に 1 回の Drag & Drop 操作で複数のファイルを投入し処理することができる．図 **4** の例は，選択した 2 つのファイルに getFaces を処理させる例である．この例では，1 つのファイルの写真に 3 人の顔画像が写っていることから，合計で 4 つの顔画像ファイルが生成されている．このように，通常は繰返しをプログラミングしなければならない処理を，一括投入で実現できるという点も POLDER の大きな特徴となっている．

2.2.4　抽象化・収集

並行の処理フローを実行すると，処理結果は複数のフォルダの最下層に分散して格納される．異なる処理を適用した結果（のファイル）に共通した処理を適用する応用も多いことから，POLDER では複数の処理をサブルーチンとして利用する仕組みとその結果群を 1 つのフォルダに収集

情報処理学会論文誌　プログラミング　Vol.6　No.2　69–84 (Aug. 2013)

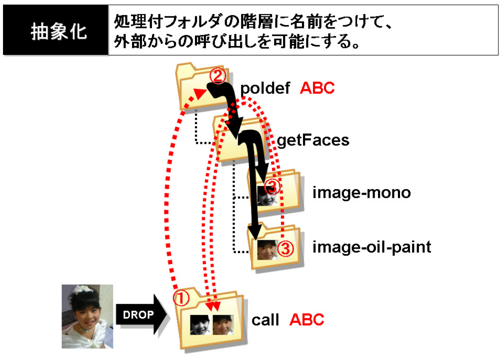

図 5　抽象化（サブルーチン）
Fig. 5　Abstraction (Sub-routine).

する仕組みを提供する．そのための特別な処理付きフォルダが poldef であり，poldef で定義されたフォルダは，それ以下の階層で実行されたすべての結果を poldef 直下に収集する．さらに，「poldef 名前」で定義された処理付きフォルダは，別の処理付きフォルダ「call 名前」によってサブルーチンとして利用することができることから，処理付きフォルダ poldef は，処理の抽象化と考えることができる．**図 5** の例では，call ABC に投入されたファイルは名前 ABC が一致する処理付きフォルダ poldef ABC に送られ，そのフォルダ構造に応じて処理が実行され，その結果として得られたすべてのファイル群がいったん poldef ABC 直下に収集された後，呼び出し元の処理付きフォルダ call ABC 直下に移動する．なお，poldef は call からの呼び出しとは独立に単独で利用することも可能であり，今回の比較環境の構築ではこの poldef を活用して後述するリモート実行プログラムを簡便に実装することができている．また，poldef と call による収集の仕組みは，複数の処理結果を合流させる仕組みとしても活用できる．

2.3　フォルダ・プログラミング環境：POLDER

POLDER のアーキテクチャを**図 6** に示す．クライアントとサーバ間は，広く普及しているネットワーク・フォルダ WebDAV を用いて実装している．ファイルが処理付きフォルダに HTTP-PUT [*1] されたことをトリガとして，そのファイルに対するフォルダ・プログラム処理系が起動される．処理付きフォルダ名で指定された処理が選択・実行され，その結果が当該フォルダ内に生成され，さらに下位に処理付きフォルダが存在すれば，再帰的にフォルダ・プログラム処理系を呼び出す．POLDER は，HTTP プロトコルの拡張である WebDAV 上で実装されているため，Microsoft Windows Explorer や，Microsoft Internet Explorer，Apple Mac OS X Finder，さらには，Microsoft Windows Explorer の拡張プラグインである Computer Hi-

*1　HTTP-PUT は WebDAV（RFC4918）で規定されているデータ転送のプロトコル．後述の HTTP-GET など HTTP-で始まるものは，いずれも WebDAV のプロトコルである．

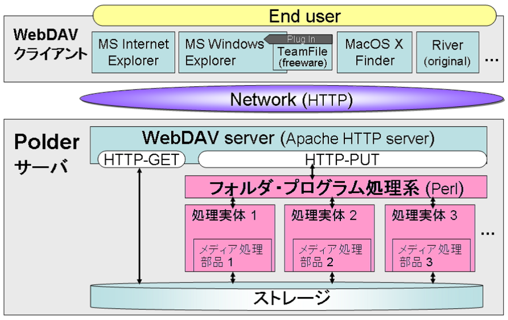

図 6　POLDER アーキテクチャ
Fig. 6　System architecture of POLDER.

Tech Inc. の TeamFile [11] など多くの WebDAV クライアントからアクセスすることができる．たとえば POLDER 上に作成されたマッシュアップ・アプリケーションである虹雲ノートシステム [10] では，映像処理，画像処理，名刺認識処理，OCR 処理，テキスト処理，音声認識処理，音声合成処理，Microsoft Office 文書処理など多彩なメディア処理部品が処理付きフォルダとして組み込まれている．利用者は，それらのメディア処理部品を WebDAV クライアントが動作する PC から簡便に利用できる．

3.　テキスト・インタフェースによる POLDER の利用と比較

3.1　プログラムのテキスト表現

POLDER では，フォルダの階層構造がプログラム表現であった．今回比較対象として導入する POLDER のテキスト表現のプログラムでは，それと同等の機能をインデント（字下げ表現）によって実現する．

3.2　設計と実装

テキスト・インタフェースでのプログラム記述と実行の過程を**図 7** に示す．利用者がクライアント側で，テキストエディタ上のインデントを使ってフォルダ・プログラムを記述するプログラム記述過程，利用者がクライアント側のコマンドプロンプトで，そのプログラムをコンパイルし実行ファイルを得るコンパイル過程，利用者がクライアント側のコマンドプロンプトで，実行ファイルに処理対象の引数ファイルを指定して実行する実行過程，からなり，利用者はクライアント側の実行ディレクトリに実行結果のファイル群を得る．

3.2.1　プログラム記述過程

テキスト版プログラム記述過程は，メモ帳などの利用者が慣れ親しんだ通常のテキストエディタを利用する．字下げ数はプログラム内で固定値（たとえば半角空白 2 個）にする．図 7 の最上段は，具体的なプログラムの記述例であり，3 階層のプログラムとなっている．

3.2.2　プログラムのコンパイル過程

フォルダ・プログラム・コンパイラ（polderMake.pl）は，

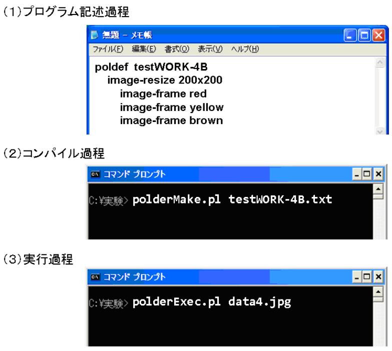

図 7　POLDER のテキスト・インタフェースの例
Fig. 7　Example of text interface of POLDER.

(入力)P: フォルダ・プログラムが記載されたテキストファイル．
(step1)入力ファイル P の各行 Li について以下を繰り返す．
　(step1-1)Li が空行の場合は何もしない．
　(step1-2)Li 行末に，もしプログラム開始点記号（‘<’）があれば削除し，Li を実行開始行 Lr として記録する．
　(step1-3)Li 行をフルパスに変換し，前後をシングルクオート文字（’）で囲み，その前に文字列 “polderMake_mkdir.pl” を付加したコマンド行 Ci を生成する．
　(step1-4)生成したコマンド行 Ci を子プロセスとして実行する．ここで polderMake_mkdir.pl は，引数として与えられるフルパスをもとに，HTTP-MKCOL を使って POLDER サーバ上にフォルダの生成を行う．
(step2)実行開始行 Lr がなければ，プログラムの先頭行 L1 を実行開始行 Lr として記録．（この実行開始行 Lr は次節実行過程にて利用する．）

図 8　テキスト版プログラムのコンパイル過程
Fig. 8　Compile steps of text-based program.

テキストで記述されたフォルダ・プログラムに対応するフォルダ階層構造をリモートの POLDER サーバ上に生成する．そのコンパイル処理ステップを図 8 に示す．このコンパイラは，テキスト行に対応したフォルダ構造をネットワーク・フォルダ上に作成するだけであり，単純にはコンパイラとは呼べないかもしれない．しかし，利用者から見たアナロジとしての分かりやすさを重視して，実験参加者にはコンパイラとして説明することとした．なお，このコンパイラは，約 80 行の Perl で実装されている．

3.2.3　プログラムの実行過程

フォルダ・プログラム・リモート実行プログラム（polderExec.pl）は，引数で与えられたデータファイルに対して，直前に作成したフォルダ・プログラムを適用し，実行結果を実行ディレクトリに取得する．具体的には図 9 の実行ステップとなる．先の 2.2.2 項で述べた並行が利用されていると，その処理結果が複数個所に分散して蓄積される．また，処理結果はフォルダ階層の最下層に蓄積される．これらの分散した処理結果を収集するために，今回のコンパイラ設計では，POLDER の抽象化機能（2.2.4 項）

(入力)D: 処理対象データファイル．
(step1)直前にコンパイルされた入力ファイル P の実行開始行 Lr に対応する URL に存在するすべてのファイルを HTTP-DELETE にて削除する．(初期化)
(step2)入力された処理対象データファイル D を，先の実行開始行 Lr に対応する URL に HTTP-PUT する．これによって POLDER 処理系が起動される．
(step3)HTTP-PUT の応答を待った後，投入 URL 中に存在するファイル群 Rj を実行カレントディレクトリに HTTP-GET する．

図 9　コンパイルされたプログラムの実行過程
Fig. 9　Execute steps of compiled program.

を利用した．これにより，利用者がプログラム記述ファイルの先頭行に「poldef 処理名」を記載するというルールを設ければ，それ以下の処理結果は「poldef 処理名」フォルダ直下に収集されるため，フォルダ・プログラム・リモート実行プログラムは，HTTP-PUT した場所に生成された結果ファイル群だけを HTTP-GET すればよく，HTTP::DAV を使った今回の実装では，約 30 行の Perl で実装されている．

3.3　実験

POLDER のフォルダ・インタフェース環境とテキスト・インタフェースを使う環境とを用いて，その利用者に対する効果調査のための比較実験を行った．

3.3.1　実験環境

実験環境は，POLDER のサーバに，複数の PC を無線 LAN 経由で接続し，各クライアント PC は 1 人の実験参加者が専有し，さらに各実験参加者はサーバ上の 1 つの Web フォルダのユーザ域を専有した状態とした．各クライアント PC では，次の 2 つの方法で実験ができるように環境を用意した．

【方法 A】 POLDER のサーバを Microsoft Windows XP の Explorer からアクセスするフォルダ・インタフェース環境．POLDER のアクセスには Explorer に TeamFile [11][*2]をプラグインしたものを使う．

【方法 B】 POLDER のサーバを Microsoft Windows XP のコマンドプロンプトからアクセスするテキスト・インタフェース環境．POLDER のアクセスには，まずメモ帳などのエディタでプログラムを作成し，フォルダ・プログラム・コンパイラでフォルダ・プログラム実行プログラムを作成し，その引数にファイルを指定して実行することで処理を行う．

3.3.2　実験の進め方

実験全体の流れは，以下のステップである．なお，方法 X，方法 Y は，方法 A，方法 B のいずれかである．

(1) 実験前アンケート

*2　サーバ側でのファイル生成やファイル名変更を即時に反映する点などが，標準の Microsoft Windows Explorer より優れている．

(2) 方法 X の解説と練習（20 分程度）
(3) 方法 X の課題 1〜課題 5 までの処理（各作業時間測定）
(4) 方法 Y の解説と練習（20 分程度）
(5) 方法 Y の課題 1〜課題 5 までの処理（各作業時間測定）
(6) 実験後アンケート

実験では，1 人の実験参加者が方法 A と方法 B の 2 種類の方法で課題に取り組むことになる．一般に参加者は課題を解くことで学習効果が発生するため，明らかに後に利用した方法での作業時間が短縮され有利になる．そこで，その影響を最小限にするために，本実験では以下の対応をとることとした．まず，問題自身が難しすぎないように配慮した．問題の解き方自体が分からない場合の課題解決支援については本実験の範囲を超えると考え，実験の立会人が適宜サポートする体制とした．また，処理部品の発見の時間を除外する目的で，課題の中で利用する各処理部品については問題とともに与えた．必要に応じて課題を解くためのヒントも与えてある．課題の理解と解決方針の検討時間を排除するため，課題を参加者に与えてからしばらくの間，課題解決のための方針決定の時間として与え，計測時間から除外した．課題を理解し解決方針が立った時点で，実験参加者自身が課題解決に向けた取り組み開始を宣言することとした．実験参加者を方法 A から開始するグループと方法 B から開始するグループに分け，各々がおおむね同人数となるようにした．各環境の習熟のために使用した練習課題は以下である．

【練習課題】 フォルダ「STUDY-0X」内の画像ファイル「lena1.jpg」に「drawText POS=NorthEast Copyright-NTT」処理を適用して，Copyright 付きの画像ファイルを生成してください．処理結果はフォルダ「RESULT-0X」に格納してください．プログラムの先頭行は「poldef STUDY-0X」としてください．すべて半角文字です．

3.3.3 実験参加者の属性

今回の実験の実験参加者として，大学生 36 名を募り，はじめに経験に関するアンケートを実施した．**図 10** に示すとおり，半数の実験参加者（18 名）が「プログラミングをよくする」（以下，開発者と呼ぶ），残り半数弱の実験参加者（16 名）が「プログラミングはあまりしないが，過去に経験した」（以下，経験有・利用者と呼ぶ），また，残りの実験参加者（2 名）は「プログラミングは経験がない」（以下，経験無・利用者と呼ぶ）であった．その他，下記の属性を持っていた．全員がフォルダを使ってファイル整理を行っているのに比較して，コマンドプロンプトの利用率は低いという傾向が見えるが，これは一般的な傾向であると考えられる．

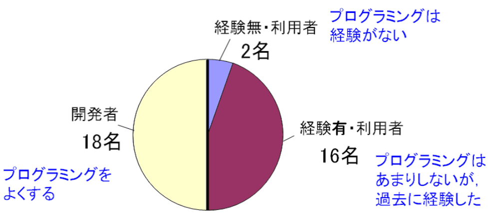

図 10　実験参加者の内訳
Fig. 10　Breakdown of subjects.

- 全員（36 名）が Microsoft Windows デスクトップの利用経験を持っていた．
- ほぼ全員（35 名）が Microsoft Windows は，日常的に利用している（開発者の中で，1 名だけが Microsoft Windows を使っていなかった）．
- 全員が GUI としてのフォルダ操作（Drag & Drop）を知っている．
- 全員がフォルダを使ってファイルを整理している．
- 全員がテキストエディタも日常的に利用している．
- コマンドプロンプトでテキストファイルの表示ができるは約 2/3 の 21 名（経験無・利用者：0 名，経験有・利用者：9 名，開発者：12 名）だけであった．

利用者実験で使用した実験課題は以下の 5 問である．

【課題 1】 フォルダ「WORK-1X」内の映像ファイル「data1.mpg」に「MPGtoDigestJPGs」処理を適用して，画像ファイル群を生成してください．処理結果はフォルダ「RESULT-1X」の中に格納してください．プログラムの先頭行は「poldef WORK-1X」としてください．

【課題 2】 フォルダ「WORK-2X」内の画像ファイル「data2.jpg」に対して，画像ファイルのサイズを 300×300 にして，赤色のフレームを付けてください．処理結果はフォルダ「RESULT-2X」の中に格納してください．

利用する処理：
「`image-resize 300x300`」
「`image-frame red`」

【課題 3】 フォルダ「WORK-3X」内の画像ファイル群に対して，20 枚のうちから好きな 6 枚を選んで縮小した画像ファイル群を生成してください．処理結果はフォルダ「RESULT-3X」の中に格納してください．

利用する処理：
「`image-small`」　or　「`image-resize ???x???`」

【課題 4】 フォルダ「WORK-4X」内の画像ファイル「data4.jpg」に対して，サイズを 200×200 に変更して，赤色のフレームを付けたものを 1 枚，黄色のフレームを付けたものを 1 枚，茶色のフレームを付けたものを 1 枚の合計 3 枚を作ってください．処理結果はフォルダ「RESULT-4X」の中に格納してください．ヒント：フォルダの階層関係を利用しましょう．

【課題 5】 フォルダ「WORK-5X」の画像ファイル群はサイズがバラバラであり，以下のプログラムで処理をすると「250×250 の画像に赤いフレームが付いた画像」が

得られるのですが，フレームの太さが変わってしまうという問題が存在します．このプログラムを参考にして，「250 × 250 の画像に赤いフレームが付いた画像」を得られて，つねにフレームの太さが同じになるようにプログラムを修正し，フォルダ「WORK-5X」内の 4 つの画像ファイル「data501.jpg」「data502.jpg」「Image_0630.JPG」「Image_0717.JPG」に対して，処理結果を得てください．処理結果はフォルダ「RESULT-5X」の中に格納してください．

プログラム：

```
poldef testWORK-5X
  image-frame red
    image-resize 250x250
```

課題 1 は単純に処理付きフォルダを実行する課題で，処理付きフォルダの利用への取り組みやすさの確認を目的としている．課題 2 は，処理の連鎖を使った簡単なフォルダ・プログラムの取り組みやすさの確認を目的としている．課題 3 は複数の入力ファイルに対して同一の処理をする場合の取り組みやすさの確認を目的としている．基本的には，この 3 つのパターンが，デスクトップ上の処理とネットワーク上の処理のマッシュアップ，ネットワーク上の処理部品群のマッシュアップ，ネットワーク上の処理部品への大量データの適用のきわめて簡易なモデルとなっている．課題 4 はさらにプログラムの高度化に向けたケースとして 1 つの入力に対して複数の結果を得る処理，課題 5 はプログラムの間違いを発見して修正する場合の処理となっており，課題 1～課題 3 に比べてより複雑な処理に対する効果の確認を目的としている．

いずれの課題も 3.3.1 項で述べた方法 A と方法 B で実施し，その作業時間を比較する．課題文中の「X」は「A」もしくは「B」に展開された問題文として配布された．また，今回利用する文字コードは半角であること，プログラムの先頭行は「poldef WORK-NX」（N は課題番号，X は方法種別）とすることの指示，さらに，思考時間と作業時間の分離のため，「方針を検討してください．方針が決まったらチェックシートの開始にチェックを付けて，開始を宣言する bat ファイルを実行してくだい」との指示も各問題文の末尾に記載され作業時間の測定が行われた．問題の不明点は実験管理者に自由に質問することを許容し，いずれの課題も脱落者なく，すべての課題を完了した．

3.4　実験結果と考察

3.4.1　実験による作業時間の比較

今回の実験では，同一の課題に対して，異なる 2 つの手法によるプログラム作成と実行の方法を比較・評価している．その全課題に対する作業時間の合計を方法 A と方法 B に関して比較した結果を**図 11** に示す．縦軸は作業時間を

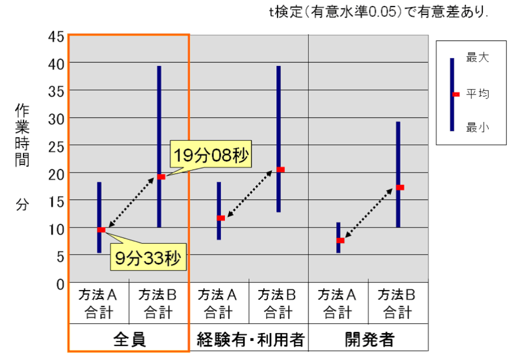

図 11　方法別の作業時間の比較

Fig. 11　Comparison of methods in terms of elapsed time.

示し図中の縦棒は作業時間の最小値と最大値を，短い横棒は平均値を示している．図から明らかなように，作業時間の最大・最小・平均のいずれも開発者が経験有・利用者より短時間で作業を完了できていた．

作業時間の平均値が，経験者・利用者と開発者との間で大きく異ならなかった原因としては，課題が両作業者にとって十分に取り組みやすいレベルであったことが考えられる．一方，方法 A と方法 B の比較では，いずれの実験参加者においても方法 A の作業時間が 1/2 程度と短い結果となった．特に，テキスト・インタフェースに慣れていると思われる開発者であっても，方法 A が有効であったことは特筆すべき結果であるといえる．方法 A と方法 B との作業時間を t 検定（有意水準 0.05）した結果，経験者・利用者および開発者のいずれであっても有意に方法 A の作業時間が短いことが確認できた．

なお，方法 B における polderMake.pl のコンパイル時間は課題 1 および課題 3 が 2 秒，課題 2 が 4 秒，課題 4 が 6 秒，課題 5 が 3 秒であった．polderExec.pl の実行時間は映像処理する課題 1 に一番時間がかかり 12 秒，その他は，1～3 秒であった．編集時間は処理付きフォルダ名の確認時間や入力速度などによってバラツキがあった．4 章の実験参加者である技術者のケースにおいては，今回の課題の中で最も大きいプログラムの編集時間は課題 4A で 50 秒程度，課題 4B で 30 秒程度であった．

次に，各課題についてより詳細に分析を試みる．経験有・利用者の各課題における方法 A と方法 B の作業時間比較を**図 12** に，開発者に対する同じ結果を**図 13** に示す．各図は左から課題 1 に対する方法 A の作業時間，方法 B の作業時間，$\cdots$，課題 5 に対する方法 A の作業時間，方法 B の作業時間と並んでいる．これらの結果から，以下が考察できる．

- いずれの課題においても，開発者と，経験有・利用者は，ほぼ同様の傾向にあり，フォルダ・インタフェースの方が平均作業時間が短い結果となっている．
- 単純な処理付きフォルダの利用である課題 1 と多数のデータへの一括処理である課題 3 で，フォルダ・イン

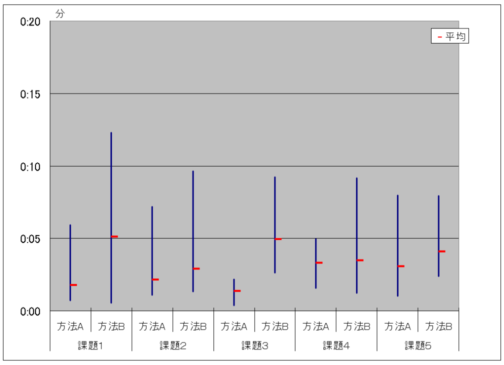

図 12　課題別・方法別の作業時間の比較（経験有・利用者）
Fig. 12　Elapsed time of each task and each method (user).

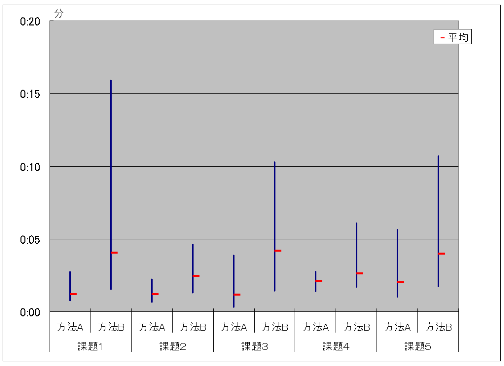

図 13　課題別・方法別の作業時間の比較（開発者）
Fig. 13　Elapsed time of each task and each method (developer).

タフェースの作業時間が顕著に短い結果となっている．
- 課題 4 は平均値で見ると両インタフェースに十分な有意差があるとはいいきれない．しかし，最悪値の面では改善が見られた．

このように個々の課題に対する作業時間の観点においても，おおむねフォルダ・インタフェース（方法 A）が，テキスト・インタフェース（方法 B）に比べて時間短縮の効果があることが確認できた．

3.4.2　実験後のアンケート結果と分析

実験後のアンケート結果を**図 14** に示す．「方法 A と方法 B のどちらが簡単でしたか？」との設問に対し，方法 A と答えたのが 32 名，方法 B が 4 名であった．全体として，89%がフォルダ・インタフェースを使った POLDER の方が簡単であった回答としている．さらに詳細に，「操作は簡単でしたか？」という設問に「易しい」「少し易しい」「普通」「少し難しい」「難しい」の 5 段階でアンケートをとった結果を**図 15** に示す．方法 A では 81%の人が「易しい」か「少し易しい」を選択したのに対し，方法 B では 33%にとどまっており，逆に，「少し難しい」か「難しい」を選択した実験参加者は，方法 A では 3%にとどまったのに対し，方法 B では 26%も存在していた．全体として，方法 A は簡単だが，方法 B は少し難しいという傾向が観察された．

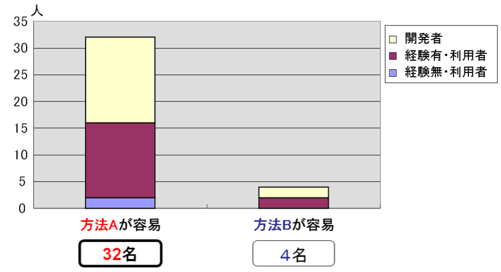

図 14　実験後アンケート（容易な方法）
Fig. 14　Questionnaire after experiment (easy method).

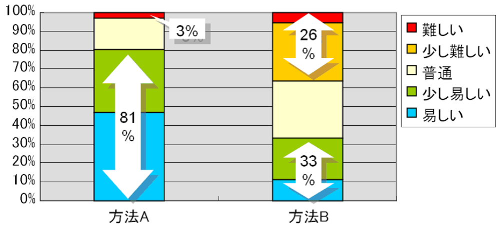

図 15　実験後アンケート（容易さの程度）
Fig. 15　Questionnaire after experiment (level of easiness).

た．これらの結果から，多くの実験参加者にとって，我々の提案する方法 A が，方法 B に比べて，同じ課題に対して，簡単に課題を解決する環境を提供したと考えることができる．

3.4.3　利用者からのコメントの分析

実験直後に自由記述でのコメントを記載してもらった．本項では，そのコメントの分析と考察を行う．方法 A が容易だと回答した 32 名（全体は 36 名）に対して，より詳細な分析を行うために，容易さに差異があった点を自由記述欄に記載してもらった．項目として類似のことを主張しているケースも存在するが，できる限り記載された原文に基づいて分類を試みる．なお，1 人の回答に複数の項目が記載されている場合は，各々の項目にカウントしている．

(a1) 複数ファイルを一括して処理できるので簡単（14 名）．
(a2) マウス操作や Drag & Drop 操作が簡単．タイプの手間が少ないから簡単（15 名）．
(a3) 入力ミスが少ないから簡単（2 名）．
(a4) 馴染みがある操作，使い慣れた操作，直感的操作だから簡単（10 名）．
(a5) 視覚的で分かりやすいから簡単（6 名）．
(a6) 抵抗があるコマンドプロンプトを使わなくてよいから簡単（11 名）．
(a7) コンパイルやその実行の手間が不要なので簡単（3 名）．
(a8) コマンドプロンプト，ファイル操作，メモ帳とさまざまな画面を行ったりきたりする必要がないから簡単（2 名）．

また，今回の実験作業を監督した大学院生の意見を以下に列挙する．

(s1) 実験参加者の取り組みを見ると，一括処理は明らかに

面倒そうで，コマンドプロンプト側では結構苦労していた様子であった．「1 個ずつのやり方は分かるけど，一気にやる方法は分からない . . .」という実験参加者が多いという印象を受けた．これに対し，フォルダ・インタフェースでは一括ドロップするだけで，細かい知識がなくても直感的な操作でできていた．この点はきわめて優位なところだと思った．

(s2) コマンドプロンプトでは，引数となるファイルを間違うことが多く発生していた．ファイル名の間違いだけでなく，別のディレクトリに引数のファイルがあることに気づかない実験参加者もいた．コマンドプロンプトでは，対象のファイル名をきっちり調べて入力する必要があるのに対し，フォルダ・インタフェースはファイルをドラッグ・アンド・ドロップで与えるためファイルを間違えない，という点の差が目立った．

(s3) 実験参加者は，テキストファイルにソースを書いてコンパイル・実行という手順にあまり慣れていない様子だった．一方，Microsoft Explorer を使ったドラッグ・アンド・ドロップは，すべての実験参加者が慣れており，スムーズに操作ができていた．コマンドプロンプトの方の説明では，納得し理解されているかが不明であったが，フォルダ・インタフェースの POLDER の方では一括処理や並行処理を説明すると「なるほど」と，素直に受け入れていた感触があった．

(s4) コマンドプロンプトの方では，コマンドプロンプト，テキストファイル，Explorer，の 3 つのウインドウが必要となる．それに対して，フォルダ・インタフェースの POLDER の方では，Explorer のみでよい，という点に差がある．Explorer の画面だけで作業が完結するフォルダ・インタフェースの方では操作を混乱しないが，3 つのウインドウの切替えが必要になるコマンドプロンプトの方では操作に手間どる実験参加者がいた．

これらの意見は，(s1) が (a1) に，(s2) が (a2)–(a3) に，(s3) が (a4)–(a7) に，(s4) が (a8) に，各々対応していると考えられる．方法 A を支持する意見と実験作業監督者の意見はほぼ一致していると考えられる．

複数のファイルを処理する場合に，コマンドプロンプトの場合には bat ファイル化を行うか，1 つ 1 つ指定することになる．実験の作業時間でも確認できたとおり，やはりフォルダ中のファイルをマウスで選択し Drag & Drop で指定できる効果は高い．入力ミスが少ないという点も，主に処理対象として与えるファイル名の入力を指しており，Drag & Drop の効果と同一である．また，実験参加者の属性としても紹介したとおり，多くの実験参加者はフォルダ・インタフェースによる操作に慣れており，コマンドプロンプトには慣れていない．POLDER がふだん慣れたフォルダ操作の拡張である点は心理的障壁を下げる効果があったと考えられる．POLDER がデスクトップのファイル整理などとうまく連携する点を評価する声もあった．

一方，方法 B が容易だと回答した残り 4 名の回答を分類すると以下のようになる．

(b1) 作業を間違えた場合でもファイルが消えない安心感があって簡単（2 名）．

(b2) マウスを利用しなくて済む，もしくは，（マウス操作を含め）手を動かす量が少なくて済むから簡単（2 名）．

(b3) テキストの方が，プログラムの階層全体が理解しやすいから簡単（1 名）．

(b4) テキストで書いた方が，並行フォルダや階層フォルダを一度に作れる点が簡単（1 名）．

処理付きフォルダでは，処理結果ファイルが生成されるとともに入力ファイルは基本的には消滅する．作業ミスに備えてバックアップを作成する必要があると考えて作業を慎重に行う実験参加者には心理的抵抗があったと考えられる．より抵抗感を下げるため UNDO などの仕組みをきちんと整備する必要があることが分かった．プログラム階層の理解のしやすさについては，フォルダ階層を好む意見もあったが，テキストファイルの方が全体像を理解しやすいという意見も存在した．ここは利用者のスキルや処理の大きさが影響する部分だと考えられる．また，フォルダ階層を作成する手間については，テキストで記載してコンパイルする方が手間が少なく，一定規模以上のプログラムを作成する場合には効率的であると考えられる．つまり，当初我々が期待したフォルダ・インタフェースが一方的に優位であるという予想に反し，今回の結果から，利用者のスキルレベルや利用内容に応じた適切な利用局面が存在し，両方が存在し使い分けできることが重要であるということが分かった．

4. Pure Data と POLDER の比較

4.1 ビジュアル・プログラミング言語 Pure Data

Pure Data [16], [17], [18], [20] はオーディオ・映像・グラフィカル処理のためのリアルタイムなグラフィカルなプログラミング環境である．当初はコンピュータ・ミュージック用のグラフィカル・プログラミング環境として開発された．プログラムは，データフロー・プログラミングの形態をとり，ボックスの上部を入力，下部を出力として，線で接続することでプログラム（パッチと呼ぶ）を独自のグラフィカル・エディタ上で構成していく．ボックスには，オブジェクト，メッセージ，ナンバ，シンボル，コメントの 5 種類があり，ボックスの右側の形によって区別される．なお，Pure Data は，効率的にオーディオなどのプログラミングが可能なドメイン特化言語であり，プログラマではない音楽家などにも多数利用されているが，エンドユーザにとって簡単だと主張しているプログラミング環境ではない．

4.2　ユーザビリティによる比較

本章では，POLDER と Pure Data との比較について，ヤコブ・ニールセンによるユーザビリティ原則 [22] に従った評価を試みる．このユーザビリティ原則は，(1) シンプルで自然な対話を提供する，(2) ユーザの言葉を使う，(3) ユーザの記憶負担を最小限にとどめる，(4) 一貫性を保つ，(5) フィードバックを提供する，(6) 出口を明らかにする，(7) ショートカットを提供する，(8) 適切なエラーメッセージを使う，(9) エラーを防ぐ，(10) ヘルプとドキュメンテーション，の 10 項目からなる．同文献においてユーザビリティの評価においては数名（5 名程度，少なくとも 3 名）の評価者がいれば妥当な結果が得られるとあるため，3 章に比べ小規模な人数での実験とした．

Pure Data は長い歴史とコミュニティの実績を持つためすでに多数の機能が提供されている．一方，POLDER は体系的な処理部品の構築は十分ではない．そこで，今回は現状の POLDER に合わせ，Pure Data で OpenCV [3] の機能を利用する GEM（Graphics Environment for Multimedia）[19] を POLDER の比較対象とした．POLDER のフォルダ・インタフェースの特徴の 1 つは，多くの利用者が日常利用するフォルダ・インタフェースを用いている点であり，既存の利用者の知識を活かしたプログラミング環境を提供している点にある．そこで，今回の評価のポイントは小さめのプログラムの作成編集時における障壁の低さに重点をおいて評価を行った．

4.3　実験

4.3.1　実験環境と実験概要

Pure Data 環境は，現時点での最新版である Windows 版 Pd 0.43.4-extended を利用した．実験参加者は情報処理の技術者 6 名で，年齢構成は 20 代 2 名，30 代 1 名，40 代 3 名である．3 章での学生に比べて，高いスキルを有する実験参加者となっている．

POLDER での実験において，ルートとなるフォルダや課題のフォルダ構造の解説は行ったが，フォルダ・インタフェース（Microsoft Explorer の使い方）の説明は必要なかった．一方，Pure Data の編集・実行環境は固有のものであり，説明なしで利用を開始することは難しい．そこで，最低限の事前情報として，(1) オブジェクトボックスとメッセージボックスと bang ボタンの各図形（図 **16** 上部に示す記号群），(2) 編集モードと実行モードの区別があり編集モードで作業しなければならないこと，を伝えた．また，(3) 論理構造が一緒であれば配置の良し悪しは問わないことを説明した．

プログラミング言語としてみた Pure Data は特有の概念を利用するために，その理解には時間がかかると推測されることから，インタフェースのユーザビリティを分析することに限定して実験を行った．実験では，プログラムの作成編集過程に着目することとし，下記の 2 つの課題を与えた．これらは，Pure Data の GEM において最低限の処理を実行するプログラムとなっており，POLDER で与えた問題とは厳密には異なるが，Pure Data で行った課題 7 が，POLDER の課題 1 と同等の課題構造を持つと考えた．

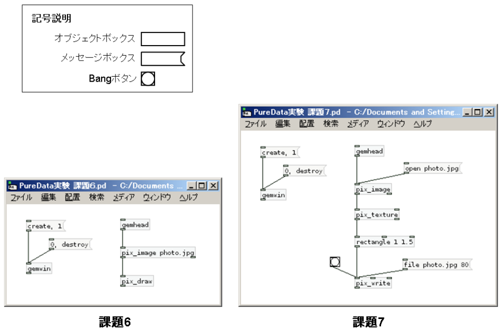

図 16　Pure Data によるプログラム
Fig. 16　Pure Data programs.

【課題 6】　図 16 左に示すプログラム（画像ファイル photo.jpg を読み込んでウインドウに表示するプログラム）を入力し，そのプログラムをファイルに保存する．ファイル名は自由．

【課題 7】　図 16 右に示すプログラム（画像ファイル photo.jpg を読み込んで jpg ファイルに出力するプログラム）を入力し，そのプログラムをファイルに保存する．ファイル名は自由．課題 6 の結果を利用して編集してもよい．

課題 6 の作業では，ウインドウの制御のオブジェクトボックス（gemwin，gemhead），画像の読み込みオブジェクトボックス（pix_image），画像の描画オブジェクトボックス（pix_draw），および，ウインドウの生成消去メッセージボックス（create，destroy）を，4 本の線で結ぶ構造を作成する必要がある．課題 7 の作業では，課題 6 の構造に加えて，3 つのオブジェクトボックス（pix_texture，pix_rectangle，pix_write），2 つのオブジェクトボックスの修正や削除（pix_imege，pix_draw），2 つのメッセージボックス（open，file），1 つの bang ボタンの追加，さらに 4 本の線の追加を行う必要がある．これらの作業の様子を撮影したビデオから，作業時間と作業エラーの分析を行った．

4.4　実験の分析

4.4.1　作業時間の分析結果

Pure Data で新規編集画面を開き，作業を開始した時点から，プログラムをファイルに保存した時点までをプログラム作成時間として計測した．各作業の平均時間は課題 6 が 2 分 49 秒，課題 7 が 3 分 38 秒，最短時間は課題 6 が 1 分 38 秒，課題 7 が 2 分 16 秒であった．最短時間で比較すると，同一メンバがフォルダ・インタフェースを使って最もプログラム量の多い課題 4A を行った際のプログラム記述の最小時間 51 秒よりも長い時間となっている．これは，オブジェクトの選択移動や結線する時間，プログラムファ

イルとして書き出すためのパス指定時間，プログラムとして必須部分（ウインドウ処理，ファイルの読み込み・書き出し処理）が多いことによる作業増に起因していた．これらの結果から，POLDER と Pure Data において，小さめの簡易なプログラミングにおいては，POLDER の方がより短い時間で処理を行うことが可能だといえる．

4.4.2　作業エラーの分析結果

Pure Data での作業エラーは，以下に示す多様なものが，多数観測された．

- メッセージとオブジェクトを間違う（4 回）．
- 編集できなくなる，もしくは，実行モードで編集モードの混乱（2 回）．
- 結線できない（下側ボックスから上側ボックスには結線できない），もしくは，結線の失敗（13 回）．
- ボックスの削除のための方法が分からない，もしくは，過剰に選択して削除するミス（5 回）．
- ボックスの移動のための選択方法を探すが見つからない（3 回）．
- 過大スクロール（2 回）．

これらのエラーは特定の実験参加者に偏ることなく，最低でも 3 件以上のミスが発生していた．一方，POLDER のフォルダ・インタフェースでは，Explorer 上での作業であり，フォルダの編集作業で，課題 5A において下位のフォルダをまとめてコピーするというエラーが 1 度あっただけであった．POLDER のテキスト・インタフェースでは，プログラムの作成時に poldef 部分を polder と書き間違える混乱が 2 度発生した．また，ファイル名については，多くの場合はヒストリ機能やタブによるファイル名補完を効果的に利用していたことでエラーを避けていた．全体的に，Pure Data でのエラー発生が多いという結果であった．

4.5　ユーザビリティ原則による分析

これらの実験の経験などから，ユーザビリティ原則の 10 項目について比較を行う．

(1) シンプルで自然な対話を提供する．

Pure Data は，5 種類のボックスやホットとコールドの使い分けが必要な点など，シンプルさの追求を目的とするシステムではない．POLDER は，利用者が日常的に利用しているデスクトップ環境を前提としていることから Pure Data に比べ，自然な形で編集や実行を行うことができる．

(2) ユーザの言葉を使う．

Pure Data は，DSL（ドメイン特定言語）であるため，コンピュータ・ミュージックや波形処理については，その分野のユーザの概念に合わせることが可能である．

POLDER は，利用者が日常的に利用するフォルダ，Explorer，各種メディアの再生ソフトなどを使うという点でユーザの言葉を利用しているといえる．文献 [8] でも示したように，フォルダの拡張であるフォルダ・プログラミングは違和感なく受け入れられているが，図 3 で紹介したような条件付きの処理付きフォルダについてはユーザの混乱の可能性が考えられるため，分析の継続が必要である．なお，WebDAV に準じて，POLDER は日本語フォルダ名もサポートしている．

(3) ユーザの記憶負担を最小限にとどめる．

Pure Data は，編集と実行のモードの切替え，ホットやコールドといった状態の認知思考が必要な点など，記憶負担が大きい．

POLDER は，プログラム作成と編集については，(1)，(2) でも述べたように利用者の記憶負担を削減していると考えられる．

(4) 一貫性を保つ．

Pure Data は，マウスカーソルによるボックスや線の選択方法が Microsoft Windows デスクトップと異なることによって，利用者の作業ミスを誘発していた．一貫性についてはさらなる改善が必要だと思われる．

POLDER は，一般に普及したデスクトップの延長上に操作体系を構築している．しかし，利用者が前提とするデスクトップ環境やプログラミング環境も今後変化していくと考えられるため，POLDER にも継続的な進化が必要であろう．

(5) フィードバックを提供する．

Pure Data は，オブジェクト名の誤入力時には，入力ボックスが確定しない状態に変化し，エラーの表示がコンソールに出力される．これにより実験でも利用者はその名称間違いに容易に気づくことができた．しかし，オブジェクトとメッセージの間違いについては十分なフィードバックが行われていない．

POLDER は，処理付きフォルダ名の誤入力を気づく手段が，現時点では提供されていない．POLDER では，処理が実行されていないという事実から利用者自身で処理付きフォルダ名の間違いを発見する必要がある点については，十分なフィードバックが提供されているとはいえない．さらに，処理時間が長いケースについてのフィードバックについては，双方とも不十分である．POLDER については，Microsoft Windows のビジーカーソル（砂時計）の仕組みが働くが，特に長い処理の場合に進捗率表示をするなどの改善が望まれる．また，Pure Data がユーザとのインタラクションを積極的に使ったプログラムを記述できるのに対し，POLDER は現時点ではプログラムの途中でのインタラクションの仕組みを提供していない．POLDER においても，Pure Data の GEM のウインドウのような GUI を提供するサーバプロセスと連携するなどの，何らかの機能拡張が必要と考えられる．

(6) 出口（exit/undo）を明らかにする．

Pure Data は，ウインドウを削除することで強制的な処理終了が可能である．また，エディタ上の UNDO が可能になっている．今回の実験においても誤削除については，この UNDO 機能が利用され，速やかな作業復旧を支援できていた．

POLDER は，Explorer 上の作業では一般的なファイルやフォルダ操作についての UNDO は可能である．しかし，処理付きフォルダの実行についての UNDO は提供されておらず，3.4.3 項での利用者の意見からも，処理前データが消えることがユーザの不安要因となっているという問題点が抽出されている．今後は，実行前に自動バックアップを行うなど，UNDO への対応が必要である．EXIT の機能については，まだ課題は顕在化していないが，(5) で述べた進捗率表示などとともに，処理付きフォルダの作成指針などを含めて整備する必要があると思われる．

(7) ショートカットを提供する．

Pure Data は，オブジェクトボックスやメッセージボックスの作成について，Ctrl+1，Ctrl+2 といったショートカット・キーが提供されており，メニューバー上から確認できる．今回の実験では 6 名中 4 名が，最終的には Ctrl キーを使ったボックス入力になっており，効果的に利用されていた．

POLDER では，フォルダについてのショートカットが活用できる．これは，Microsoft Windows の機能に依存した動作ではあったが，特定の機能を簡単に呼び出すランチャ機能として作業を効率化していくことが可能になる．また，POLDER ではつねに処理付きフォルダが永続化されていく．この点で，過去の処理の再利用が容易になっていると考えることができる．

(8) 適切なエラーメッセージを使う．

Pure Data は，実行時のエラーメッセージをコンソール上に出力する．コンソールへのメッセージは右下のボタンによって，1：致命的，2：エラー，3：ノーマル，4：デバッグ，5：すべて，の 5 段階で出力のレベルを簡単に切り替えることができる．しかし，エラーメッセージの分かりやすさは必ずしも十分とはいえない．

POLDER は，入力データが処理されずに ERROR を含む名称に改名されるという仕組みは存在するが，処理部品内でのエラーなどについてメッセージが出力されない現状は明らかに不十分であり，エラーの具体的情報や回避に向けた提案など，今後の拡充が必要である．

(9) エラーを防ぐ．

Pure Data では，(5) でも述べたとおり，オブジェクトボックスの誤入力を防ぐ仕組みが機能していた．しかし，(3) で述べたモードの面では問題がある．今回の実験でもモードの間違いによる操作エラーが複数回観測された．Pure Data 環境では編集モードと実行モードの存在が重要であるため，積極的に背景色を変えるなどの，モードの識別を容易にする仕組みの導入が待たれる．

POLDER では，つねに編集と実行が同時に可能な状態であり，モードは存在しない点が Pure Data に比べてエラーの発生の削減に効果がある．しかし，(5) で述べたような処理付きフォルダ名の誤入力を予防する仕組みが必要である．

(10) ヘルプとドキュメンテーション

Pure Data は，環境に付属するヘルプやドキュメントが存在している．しかし，初心者が使うために十分な日本語ドキュメントが充実しているわけではなく，さらなる充実が求められる．なお，今回の実験では，参考となるサンプルプログラムを与えることで実施した．

POLDER でも，Explorer についてはヘルプは不要であった．また，処理付きフォルダにもヘルプを提供する基本的な仕組みは存在する．しかし，POLDER のドキュメントもヘルプも不足しており，今後の整備が必要である．

全体として，Pure Data は，シンプルさや一貫性，記憶の負担に課題があるため，利用者のエラーを誘発しやすい点が多数存在した．しかし，波形データの扱いやマウスによるスライダ制御などの対話性の高いプログラムを汎用プログラミング言語に比べて容易に構築できる点でコンピュータ・ミュージック分野などの DSL（ドメイン特化言語）として有用性がある．

一方，POLDER では，デスクトップとの一貫性が高いことから初心者にでも取り組みやすく，モードも存在しないため操作エラーも少なくなる．しかし，Pure Data に比べ，リアルタイム性とインタラクションのあるプログラム作成については改善の必要があり，システム状態のフィードバック，操作の UNDO，エラー処理や予防についての仕組みについては不足しているといえる．また，POLDER のドキュメントなどの整備も十分ではない．

5. 関連研究

POLDER は，「M1：マルチメディア対応」で，「M2：Web API をベースとした処理の結合」を行い，日常的にはプログラミングを行わない PC 利用者にも「M3：簡単なマッシュアップ」を可能にすることを狙いとしている．ここで，簡単なマッシュアップのためにはエンドユーザ・プログラミング環境としての分析が必要になるが，それを環境側面の「e1：言語仕様」「e2：編集時」「e3：実行時」「e4：結果確認時」「e5：応用修正時」と，ユーザの受容性側面の「u1：学習量」「u2：認知思考量」「u3：作業量」の 2 つの側面で分析を試みる．ここで，学習量は使い始めるまでに必要な記憶に関する努力量や習熟に関する努力量，認知思

		環境側面				
		e1 言語仕様	e2 編集時	e3 実行時	e4 結果確認時	e5 応用修正時
ユーザ受容性側面	u1 学習量	○POLDER ×Prograph ×PureData ×LabVIEW	○POLDER ×Prograph ×PureData ×LabVIEW ×Y.!Pipes ×JackBe ×Automator ×FolderAction ×Action(P.S.) ×Droplet(P.S.)	○POLDER ×PureData ×Y.!Pipes ×JackBe ×Automator ×FolderAction ×Action(P.S.) ×Droplet(P.S.)	○POLDER ×Y.!Pipes ×JackBe	○POLDER ×Y.!Pipes ×JackBe
	u2 認知思考量	×PureData	○POLDER ○Action(P.S.) ×PureData ×Y.!Pipes ×JackBe ×Automator ×FolderAction ×Droplet(P.S.)	○POLDER ×PureData ×Y.!Pipes ×JackBe ×Automator ×FolderAction ×Droplet(P.S.)	○POLDER ○Action(P.S.) ×Y.!Pipes ×JackBe	○POLDER ×Y.!Pipes ×JackBe
	u3 作業量			○POLDER ○Automator ○FolderAction ○Action(P.S.) ○Droplet(P.S.)		

図 17　ビジュアルプログラミング環境の比較
Fig. 17　A comparison of visual programming environments.

考量は現在のプログラム状態を頭の中で理解再現するための努力量や頭の中で操作するための努力量，作業量は操作に関する量とする．

我々の POLDER は，OS のファイルシステムと統合することでマルチメディアの Player，Viewer，Editor に連携（M1）し，WebDAV 上で実装することで HTTP 上での動作を可能（M2）にしている．さらに，言語として原則を少なくし学習量を削減（e1，u1）し，利用者の既存のフォルダ操作の知識との差分を小さくすることで環境全般の学習量を削減（e2，e3，e4，e5，u1）し，つねに永続化されるフォルダとファイルに対する操作だけですべてを行うことで編集と実行の思考の切替えを削減（e2，e3，u2）し，処理結果はダブルクリックで可視化できることで結果チェックの認知思考量を削減（e4，u2）し，1 行実行や部分実行によって実行時や応用修正時の認知思考量（e2，e4，e5，u2）を削減している．POLDER を使って解いた今回の実験の課題では，特に単純実行時（課題 1，e3）と一括実行時（課題 3，e3）の作業量負担（u3）が少ないことが確認できた．

以下では，既存のビジュアル・プログラミング [12], [13], [14] 環境として，Prograph（Marten）[15]，LabVIEW [23]，マッシュアップ環境として，Yahoo! Pipes [6]，JackBe Presto [7]，ビジュアル・コーチング他として Mac OS [24]（Automator [25]，Folder Action [26]），Adobe Photoshop [27]（Action [28]，Droplet [29]）との定性比較を行う（**図 17**）．

Prograph（Marten）は，汎用的なプログラミングを可能にするビジュアル・データフロー・プログラミング言語であり，オブジェクト指向プログラミング言語でもある．while などの制御構造などを含むきわめて多種のアイコンが存在し，それらを線で接続することで，プログラムを構成していく．Prograph はプログラマを対象とし一般的な課題に対するビジュアル・データフロー・プログラミングを目的としており，非常に複雑で多彩な構文と環境を持っている．そのため非プログラマに対しては言語と環境の学習障壁が非常に高い（e1，e2，u1）という問題がある．

LabVIEW は，計測制御用のビジュアル・データフロー・プログラミング環境である．LabVIEW では，アイコンを順番に並べ，接続することで機能を実現する．プログラムは VI（Virtual Instruments）と呼ばれるモジュールとなる．VI はユーザ・インタフェースとなる制御器やデータ表示器を配置するフロントパネルと制御を記述するブロックダイアグラムから構成される．フロントパネル上のオブジェクトと 1 対 1 に対応する端子と関数アイコン，VI をワイヤで接続することで，プログラムを作成する．LabVIEW は，計測制御ソフトウェアの作成を得意とし，波形チャート・波形グラフ，スライドバーなどの計測制御に適したユーザ・インタフェースと，GPIB，Serial，TCP/IP などの多数の計測器制御インタフェース，および，解析機能を装備している．しかし，この LabVIEW もプログラマを対象としているため，Prograph 同様に非プログラマには言語と環境の学習障壁が高い（e1，e2，u1）．この LabVIEW は計測器制御に特化した機能の充実により，その分野の利用者からの支持を受け続けている．

マッシュアップ環境として代表的な Yahoo! Pipes [2] およびエンタープライズ向けに機能を充実させた JackBe Presto [3] もデータフローをベースとしたビジュアルなプログラミング環境である．しかし，これらのマッシュアップ環境もプログラマを対象とした開発環境が前提となっており，独自のエディタ，デバッガなど，非プログラマに対しては環境の学習障壁と認知思考障壁（e2，e3，e4，u1，u2）が高い．また，マルチメディア処理への対応（M1）も十分ではない [30]．

MacOS X の Automator もエンドユーザ・プログラミングを指向していると考えられる．Automator では，Shu [12] の分類でいう視覚的環境のビジュアル・コーチングの一部と考えることができ，利用者の直接操作を記録するアプローチをとる．Automator は，一連の作業を記録し，大量のデータに繰り返し適用する場合に作業量（e3，u3）の削減が可能で，きわめて有用なツールである．OS と統合していることから，多様なマルチメディア処理の連携が可能（M1）になっている．しかし，Automator も編集と実行環境が分離されている点で学習障壁と認知思考障壁（e2，e3，u1，u2）が高く，記録と実行というモードの切替えの存在が非プログラマの認知思考障壁となる問題がある．また，この直接操作の記録というアプローチではエンドユーザ・プログラミングに対する有望なアプローチではあるものの，条件分岐や処理の一般化は難しいためそもそもプログラミング環境としては本質的に限定されたものになるという制約がある．

一方，MacOS X の Folder Action は，MacOS X のフォルダに対するいくつかのユーザ操作イベント（フォルダを開いたとき，フォルダを閉じたとき，フォルダを移動したとき，フォルダに項目を追加したとき，フォルダから項目

を削除したときなど）に対して AppleScript で記述された処理や Automator で記録された処理を登録できる仕組みであり，実行環境の利便性（e3，u3）を向上させる．しかし，Folder Action は単に OS のフォルダ操作イベントにエンドユーザ・プログラムを割り当てる仕組みにとどまっており，フォルダの階層を作ることによるプログラミングを行う環境でも Web サービスのマッシュアップを行う環境でもなく，処理の作成とフォルダ操作が統合された利便性はないため，編集と実行環境の高い学習障壁と認知思考障壁（e2，e3，u1，u2）がある．

画像処理に特化したものとしては Adobe Photoshop の Action 機能も Photoshop 上で行った一連の操作を自動で記録してくれるビジュアル・コーチンングである．この Action 機能によって画像に加工処理（Filter）を加えた結果を見ながらその一連の Filter 操作を記録することが可能になり，編集や結果確認（e2，e4，u2）の認知思考負担の軽減が可能になる．同じ操作を繰り返し大量の画像に適用する場合には，この Action 機能によって一連の処理を記録し，メニューで選択した特定のフォルダ以下のすべての画像に適用すること（バッチ処理と呼ぶ）も可能であり，作業量の削減（e3，u3）が可能である．この Photoshop の Action 機能は，画像に対する直接操作と WYSIWYG を提供するため 1 度慣れることができれば大変便利な機能だと思われる．しかし，画像に特化した独自の編集・実行環境が学習障壁（e2，e3，u1）になり，また，このアプローチは Automator と同様に条件分岐や処理の一般化などが困難で，さらに，Photoshop は画像処理に特化しておりマルチメディア対応とはいえず（M1），応用領域の広いマッシュアップ環境ではなく，制限された領域特化言語（DSL）にとどまる．なお，Photoshop の Action 作成時の GUI 上で，Filer をフォルダ的アイコンで表現し並べることで処理を記述するインタフェースも存在するが，OS のフォルダとはまったく異なり[*3]，既存のフォルダ操作の知識を活かして非プログラマの学習負担を下げる効果はない．

この Photoshop にも Droplet という OS のショートカット相当の実行ファイル（Droplet）に Photoshop Action 機能を割り当てる機能も提供されている．この割当てにより利用者はその実行ファイルのアイコンにファイルを Drag & Drop すると，そのファイルやフォルダ内のファイル群に対して，事前に割り当てられている Action をいっせいに適用することが可能になり実行の作業量（e3，u3）が削減できる．しかし，Droplet も MacOS X の Folder Action と同様に，処理の作成とフォルダ操作が統合された利便性はないため，編集と実行環境の学習障壁と認知思考障壁（e2，e3，u1，u2）がある．

*3　OS のフォルダと統合されない具体的な点として，他のメディアのファイルやデータフォルダと混在できない点，フォルダ的アイコンの表示順が特別な意味を持っている点，ファイル・アイコンのダブルクリックで Player が起動しない点，ネットワーク・フォルダとの連携ができない点などが存在する．

6. おわりに

本論文では POLDER のフォルダ・インタフェースに対応する，プログラム・コードを記述するテキスト・インタフェースが容易に実装できることを示した．開発者と経験有・利用者とに分けた 36 名の大学生に対して，両方のインタフェースの効果の差異を，Web 上のメディア処理部品を利用するいくつかの課題を通して比較・分析した．作業時間の比較では，経験有・利用者だけでなく，開発者についてもフォルダ・インタフェースを使うことで半分程度まで作業時間を短縮できるケースが確認できた．さらに，89%の利用者からフォルダ・インタフェースを使った POLDER の方が簡単であったというアンケート結果を得た．しかし，アンケートからは，一方的にフォルダ・インタフェースが優位なのではなく，習熟度の高いと思われる開発者や一定規模以上のプログラムを扱うシーンにおいては，テキスト・インタフェースを使う方が効率的だという意見も得られた．このことから POLDER におけるフォルダ・インタフェースとテキスト・インタフェースは，どちらかが一方的に優位なのではなく，利用者のスキルレベルと利用状況に応じて，使い分けできることに意義あると考えられる．POLDER は，基本的には日常的にプログラミングを行わない学生や社会人などの生活者のための簡便なプログラミング環境を目指しているが，テキスト・インタフェースを並存させることで，より高度なプログラミングが必要となった利用者に対して，その障壁を下げる効果が期待できる．さらに，利用者から入力データが消えてしまうという処理自体がストレスとなるというアンケートの回答があり，UNDO 機能などが必要であることも確認できた．

また，POLDER とビジュアル・プログラミング言語 Pure Data との比較をヤコブ・ニールセンのユーザビリティ原則をもとに行った．6 名の技術者に対して操作エラーの比較分析を行ったところ，POLDER では，デスクトップとの一貫性が高いことから初心者にでも取り組みやすく，モードも存在しないため操作エラーも少なくなるとの結果を得た．しかし，Pure Data に比べ，リアルタイム性とインタラクションのあるプログラム作成については改善の必要があり，システム状態のフィードバック，操作の UNDO，エラー処理や予防についての仕組みについては不足していた．また，POLDER としてのプログラミング・スタイルの確立についてもいまだ多くの課題と変化の可能性が残されており，結果としてドキュメントなども十分ではないという分析結果となった．これらの課題については今後，本論文では比較しきれなかった商用のビジュアル・プログラミング可視化ツール AVS [21] などとの比較も行いながら，継続的な改善を続ける必要がある．

謝辞　本実験を中心的にサポートしてくれた筑波大学大

学院の林大策君に感謝します．

参考文献

[1] ImageMagick, available from ⟨http://www.imagemagick.org/⟩ (accessed 2012-12-01).

[2] FFmpeg, available from ⟨http://ffmpeg.org/⟩ (accessed 2012-12-01).

[3] OpenCV, available from ⟨http://opencv.willowgarage.com/wiki/⟩ (accessed 2012-12-01).

[4] ProgrammableWeb, available from ⟨http://www.programma-bleweb.com/⟩ (accessed 2012-12-01).

[5] 新谷虎松，大囿忠親：知的 Web のためのマッシュアッププログラミング，情報処理，Vol.50, No.5, pp.444–453 (2009).

[6] Yahoo! Pipes, available from ⟨http://pipes.yahoo.com/pipes/⟩ (accessed 2012-12-01).

[7] JackBe Presto, available from ⟨http://www.jackbe.com/produ-cts/⟩ (accessed 2012-12-01).

[8] Akama, H., Yamamuro, M., Menjo, T. and Satoh, T.: Integrating Multimedia Data Processing Parts in Cloud into Folder Programming Environment, *Proc. iiWAS2011* (*The 13th Intr. Conf. on Information Integration and Web-based Applications & Services*), Ho Chi Minh City, Vietnam, pp.130–137, ACM ISBN 978-1-4503-784-0 (Dec. 2011).

[9] 赤間浩樹，内藤一兵衛，毛　受崇，長谷川知洋，谷口展郎，山室雅司：フォルダ・プログラミング環境「POLDER」，情報処理学会論文誌，Vol.50, No.8, pp.1798–1809 (2009).

[10] 赤間浩樹，山室雅司，毛　受崇，佐藤哲司：虹雲ノート：クラウド上でのメディア処理連携システム，信学技報，Vol.111, No.77, PRMU2011-42 (DE2011-11), pp.61–66 (2011).

[11] Computer Hi-Tech Inc: TeamFile, available from ⟨http://www.teamfile.com/modules/releasenote_client/⟩ (accessed 2012-12-01).

[12] Shu, N.C.: *Visual Programming*, Van Nostrand Reinhold (1988).

[13] 田中二郎：ビジュアルプログラミング，ビジュアルインタフェース—ポスト GUI を目指して，bit 別冊，pp.65–78，共立出版 (Feb. 1996).

[14] Boshernitsan, M. and Downes, M.: Visual Programming Languages: A Survey, *Report No.USB/CSD-04-1368*, Univ. of California, Berkeley (Dec. 2004).

[15] Andescotia LLC: Marten 1.4, Graphical Programming for MacOSX, available from ⟨http://www.andescotia.com/products/marten/⟩ (accessed 2012-12-01).

[16] Puckette, M.: Pure data, available from ⟨http://puredata.info/⟩ (accessed 2012-12-01).

[17] Puckette, M.S.: Pure data, *Proc. International Computer Music Conference*, pp.224–227 (1996).

[18] ミラー S. パケット：*Pure Data*：開発の近況，情報処理学会研究報告［音楽情報科学］，Vol.97, No.122, pp.1–4 (1997).

[19] Danks, M.: Graphics Environment for Multimedia, available from ⟨http://gem.iem.at/⟩ (accessed 2013-03-04).

[20] 小嶋秀樹：こじ研（小嶋研究室），宮城大学，入手先 ⟨http://www.myu.ac.jp/~xkozima/lab/avmed-pd1.html⟩（参照 2013-03-03）.

[21] ビジュアルプログラミング可視化ツール AVS，入手先 ⟨http://www.cybernet.co.jp/avs/products/avsexpress/overview.html⟩（参照 2013-03-03）.

[22] ヤコブ ニールセン（著），篠原稔和，三好かおる（訳）：ユーザビリティエンジニアリング原論—ユーザーのためのインタフェースデザイン，東京電機大学出版局 (2011).

[23] NI. LabVIEW, available from ⟨http://www.ni.com/labview/whatis/ja/⟩ (accessed 2012-12-01).

[24] Mac OS, available from ⟨http://www.apple.com/jp/macosx/⟩ (accessed 2012-12-01).

[25] Automator, available from ⟨http://www.macosxautomation.com/automator/index.html⟩ (accessed 2012-12-01).

[26] Randall, N.: A Beginner's Guide to OS X Folder Actions, pcmag.com, article data:Jul.13,2009, available from ⟨http://www.pcmag.com/article2/0,2817,2350034,00.asp⟩ (accessed 2012-12-01).

[27] Adobe Systems: Photoshop, available from ⟨http://www.photoshop.com/⟩ (accessed 2012-12-01).

[28] Adobe Systems: Making Actions, available from ⟨http://tv.adobe.com/watch/photoshop-for-video/making-actions/⟩ (accessed 2012-12-01).

[29] Adobe Systems: Batch processing with Droplets, available from ⟨http://tv.adobe.com/watch/photoshop-for-video/batch-processing-with-droplets/⟩ (accessed 2012-12-01).

[30] Heinzl, S., Seiler, D., Unterberger, M., Nonenmacher, A. and Freisleben, B.: MIRO: A Mashup Editor Leveraging Web, Grid and Cloud Services, *Proc. iiWAS2009*, Kuala Lumpur, Malaysia, ACM ISBN 978-1-60558-660-1 (Dec. 2009).

[31] 赤間浩樹，山室雅司，佐藤哲司：フォルダ・プログラミングに対するフォルダ I/F とテキスト I/F の比較評価，情報処理学会マルチメディア，分散，協調とモバイル DICOMO2012 シンポジウム 6H-4，pp.1722–1730 (July 2012).

[32] 赤間浩樹，松田基弘，毛　受崇，長谷川知洋，内藤一兵衛，山室雅司：メディア処理向けクラウド基盤「虹雲」，情報処理学会全国大会 72th，6E-6, pp.3-37–3-38 (Mar. 2010).

[33] 赤間浩樹，内山寛之，西岡秀一，内藤一兵衛，谷口展郎，長谷川知洋，山室雅司，櫻井紀彦：追記・参照型データ管理システムの設計と評価，情報処理学会論文誌，Vol.49, No.2, pp.749–764 (2008).

赤間 浩樹　（正会員）

1990 年東海大学大学院理学研究科数学専攻修士課程修了．同年日本電信電話株式会社入社．ネットワーク向けインメモリ DBMS，ニュース・オン・デマンド，マルチメディア情報検索の研究開発，FLET'S 系 PF の実用化，分散データストリーム管理技術，メディア処理向けクラウド基盤の研究開発を経て，現在，次世代クラウドの研究開発に従事．電子情報通信学会，日本データベース学会，人工知能学会，ACM 各会員．

鶴岡 行雄　（正会員）

1962 年生．1985 年電気通信大学卒業．1987 年同大学大学院電気通信学研究科応用電子工学専攻修士課程修了．同年日本電信電話株式会社入社．基礎研究所，コミュニケーション科学基礎研究所，情報流通プラットフォーム研究所を経て，ソフトウェアイノベーションセンタに勤務．プログラミング言語，情報セキュリティ，モバイルコンピューティングの研究に従事．博士（工学）．2012 年より電気通信大学大学院情報システム学研究科客員教授．電子情報通信学会会員．

佐藤 哲司　（正会員）

1980 年山梨大学工学部電子工学科卒業．同年日本電信電話公社武蔵野電気通信研究所に入所．以来，論理回路の大規模一括集積技術，データベースマシン，マルチメディアデータベースの研究・開発に従事．1994 年工学博士（大阪大学）取得．2007 年 4 月より筑波大学図書館情報メディア系教授．分散並列処理，情報検索，社会インタラクションに興味を持つ．電子情報通信学会会員．

