エンドユーザによるメディア処理のための部品統合環境の構成法に関する研究

筑波大学
図書館情報メディア研究科

2013年11月

赤間 浩樹

エンドユーザによるメディア処理のための部品統合環境の構成法に関する研究

赤間 浩樹

デジタルカメラ・スマートフォン・タブレット・ノート PC などの携帯機器の普及に伴って，画像・映像・音声・構造化テキストなどの多様なデータを誰でもが気軽に生成できるようになってきた．それらのデータは，インターネットを介して人々の間で広く流通・共有されるとともに，合成や改変など新たな価値を付加する活動も活発になってきている．

メディアデータに対する処理，すなわち加工・改変をするためのソフトウェアは，近年，オープンソース・ソフトウェア（OSS) 化が進行し，多くの人々が簡単に入手できる環境が整ってきている．例えば，画像処理ライブラリでは ImageMagick，映像処理ライブラリでは FFmpeg，コンピュータビジョンライブラリでは OpenCV など，各メディア分野を代表する処理ライブラリがソースコードも含めて無償で利用できる OSS として提供されている．一方，加工・改変した作品を流通することができる場の整備も進んでおり，多くの人々が趣味あるいは趣味の延長線上で多くの作品を投稿し，共有するようになってきた．

しかし，利用者が選択したマルチメディア処理ライブラリが必ずしも所望の機能を過不足なく具備しているとは限らない．利用者は，不足機能を補って所望の機能を実現するために，複数のソフトウェアを導入し，それらを組合せて実行することを学び，切り替えながら実行しなければならない．こういったマルチメディア処理における所望の機能を実現する作業は，特に，大量のデータに処理を適用する利用者にとって大きな負担となる．その解決への有望なアプローチは，Web サービスとして提供される各種のアプリケーション・プログラム・インタフェース（API）を経由して必要とする処理を統合するマッシュアップ環境の利用である．マッシュアップ環境は既に多数提案されている．Yahoo! Pipes は，代表的なマッシュアップ環境の一つであり，Web 上の XML, RSS, JSON など構造化データの加工や，その結果の Google 地図上への配置などを簡便に行えるデータフロー・ビジュアル・プログラミングを実現している．しかし，これらの環境を利用するには，利用者はプログラミング・スキルを習得しなければならず，日常的にプログラミングを行う機会が少ない一般的な PC 利用者には，利用の障壁が高いままである．

本研究では，このような利用に関わる障壁を低減することで，幅広い PC 利用者がマルチメディア処理のマッシュアップを Do It Yourself で簡便に行えるようにするためのメディア処理部品の統合環境の構成法について提案を行う．

本論文の構成は以下の通りである．まず第 1 章で，本研究の目的と問題設定を行い，本研究の対象のエンドユーザを設定し，想定するメディア処理種別と処理統合の活用シーンの具体化を行い，アプローチの概要を示している．本研究の目的は，次の 2 つの要件を満

たし両立する環境の実現である．その第一の要件はエンドユーザのためのプログラミング環境であること，そして，第二の要件はマルチメディア処理のマッシュアップ環境であることである．その解決のため，本研究で着目したポイントは，すでに多くの利用者が日常的に活用しているフォルダ管理という作業の経験と知識である．それをプログラミングに利用することで，エンドユーザに対するプログラミング・スキルの習得の障壁の低減を目指すアプローチを採っている．

第 2 章では，関連技術を示す．第一の要件であるエンドユーザのためのプログラミング環境に関連する技術として，既存のエンドユーザ・プログラミングの視点からの動向の紹介，その中で特に有望なアプローチとされるビジュアル・プログラミングの動向について示している．第二の要件であるマルチメディア処理のマッシュアップ環境に関連する技術としてマッシュアップ・システムの動向を紹介し，本研究のアプローチに密接に関連するファイルシステムの動向についても示している．

第 3 章では，フォルダ・プログラミング環境を提案している．フォルダ・プログラミング環境の核となる構成要素である『処理付フォルダ』は，通常のフォルダの仕組みに以下の (ff1)〜(ff3) の 3 つの仕組みを加えて拡張する．(ff1) フォルダへのデータファイル Drop で処理が起動し，フォルダ名をもとに処理が動的に決定し実行され，結果がフォルダ内に入る仕組み，(ff2) 処理の引数をフォルダ名に含む仕組み，(ff3) 複数ファイルの Drag&Drop に対して，一斉に実行される仕組み．処理付フォルダを組合せて多様な処理を実現するためのフォルダ・プログラミング環境として，以下に示す (fp1)〜(fp6) の主要な実行制御機能を実現する．(fp1) 処理が連鎖する実行制御，(fp2) 並行の実行制御，(fp3) 条件分岐と else の実行制御，(fp4) 制御移行の実行制御，(fp5) サブルーチン（抽象化と収集）の実行制御，(fp6) データフォルダと処理付フォルダの混在に対する実行制御．更に，プログラム開発支援機能として，ERROR 時の入力データ，デバッグ方法，HELP 情報の参照，フォルダ・プログラムのテキスト表現といった 4 つの機能を実現していることを述べている．次にフォルダ・プログラミング環境を Web フォルダの拡張として実装するためのソフトウェア・アーキテクチャ，プロトタイプの実装の詳細を示している．最後に，本論文の第一の要件である，エンドユーザのためのプログラミング環境としての定性的な評価を，一杉の設計目標などを使って行い，その有効性を述べている．

第 4 章では，本論文の第二の要件であるマルチメディア処理のマッシュアップ環境に対して，フォルダ・プログラミング環境の適用性について議論を行っている．まず，メディア処理部品の統合環境として求められる 5 つの要件 (M1)〜(M5) を示している．それは，(M1) マルチメディア対応性，(M2) Web API 対応性，(M3) 単機能利用の容易性，(M4) 応用利用の容易性，(M5) 管理の容易性，である．

次に，フォルダ・プログラミング環境を使った画像ファイルの簡易な処理に対する利用者実験の結果を示している．更に，フォルダ・プログラミング環境を応用して開発した虹

雲ノートシステムの紹介と，そのシステムを構築する過程で得られた使用経験を示している．その後，フォルダ・プログラミング環境がメディア処理部品の統合環境で必要とされる 5 つの要件に対して適合性が高いことを示している．

第 5 章では，本研究の第一の要件である，エンドユーザのためのプログラミング環境としてのフォルダ・ユーザ・インタフェースの定量的な効果を確認している．フォルダ・プログラミング環境を特徴付けるフォルダ・インタフェースの効果を分析することを目的に，まずコードを記述する従来型のテキスト・インタフェースと比較する利用者実験を行い，その効果を分析している．実験参加者には複数の課題を両方の環境で解いてもらい，作業時間の分析とアンケート分析を行った．課題別の効果分析の結果，単純に処理付フォルダを実行する際の取り組み易さの点と，複数の入力ファイルに対して処理を実行する課題に大きな効果があり，処理の連鎖を使ったプログラムの作業時間短縮の効果を確認している．フォルダ・ユーザ・インタフェースは課題全体で約半分の時間に短縮することを確認している．

次に，他のビジュアル・プログラミング環境として OSS で入手可能な Pure Data と，フォルダ・プログラミング環境とでユーザビリティの比較実験を行った．数名の技術者を対象としたプログラム編集作業の比較実験を行い，作業エラーやアンケートの結果をもとに，Nielsen のユーザビリティ原則の観点で両者を比較している．比較・分析の結果，フォルダ・プログラミング環境は，デスクトップとの一貫性が高いことから取り組みやすく，編集と実行の環境を混乱することもないため操作エラーが少ないことなどが確認している．一方，独自のグラフィカル・ユーザ・インタフェースを使う Pure Data は，利用者の操作エラーが多数発生する結果となった．これらのことからフォルダを使ったユーザ・インタフェースの優位性を確認している．

第 6 章で，本研究が問題とする，エンドユーザのためのプログラミング環境と，マルチメディア処理のマッシュアップ環境とを両立する環境の実現に対して，本研究で提案するフォルダ・プログラミング環境がどのように解決したかを考察しまとめている．まず，ユーザの受容性の側面（学習量，認知思考量，作業量）と環境側面（言語仕様，編集時，実行時，結果確認時，応用時）から分析したフォルダ・プログラミング環境の効果を，以下の (a)〜(g) として抽出している． それは，(a) Web フォルダの知識のみで処理を実行できるので Web API の学習が不要，(b) フォルダ操作の既存知識を活用できるのでプログラミング環境の学習・習熟が極少，(c) メディア表示の既存知識を活用できるのでプログラミング環境の学習・習熟が極少，(d) 処理の即時実行機能とメディアデータの即時表示機能によって学習と認知思考が容易，(e) プログラムの編集と実行の環境の切り替えが不要なので認知思考が容易，(f) 直感的な Drag&Drop 操作が使えるので認知思考が容易，(g) 大量データへの一括処理が Drag&Drop で簡便に実行できるので作業量が低減，である．

更に，エンドユーザがより高度な課題を解いていくために必要となる，学習のなだらか

な傾斜の実現に対して，本研究の成果が，複数の新たな中間ステップを提供することを示している．漸増的にスキル成長させる中間ステップが，DIY で Web サービスを活用するライトな開発者（利用開発者）へと多くの PC 利用者を導く補助的中間ステップになることについて考察している．

第 7 章で本論文をまとめている．学生時代に少しだけプログラミングの授業を履修したがそれ以降は全くプログラミングを行わない，**日常的に PC は利用しているがプログラマとは言えない学生・社会人**は極めて多い．それらの人々が，大量のメディアデータを保有するようになり，加工や整理といったメディア処理を DIY で行いたいという要求が高まっている．本研究では，すでに多くの利用者が日常的に利用しているフォルダ管理という作業の経験と知識に着目し，それをプログラミングに利用することで，エンドユーザに対するプログラミング・スキルの習得の障壁の低減する．そのアーキテクチャとして，Web フォルダをプログラミング環境として拡張するフォルダ・プログラミング環境の提案を行い，その実現性を POLDER の実装によって示した．いくつかの実験とそれらの評価結果から，本研究で提案するフォルダ・プログラミング環境が，本研究が問題として設定する 2 つの要件を十分満たし両立する環境であることを示した．この提案環境によって，日常的に PC は利用しているがプログラマとは言えない利用者が，大量のメディアデータに対するメディア処理を DIY で行うことができるようになる．更に，本環境は，DIY でプログラミングを始めたエンドユーザが，より高度な課題を解いていくために必要となる学習のなだらかな傾斜の実現に対して，複数の新たな中間ステップを提供することで支援することを示した．最後に，本研究の今後の課題を示した．

Studies on Component Integration Architecture for End-User Media-Processing Environment

Hiroki Akama

Since mobile devices such as digital cameras, smart phones, tablet computers, and note PCs have proliferated and usage has increased, users can easily generate data in various media formats such as images, videos, voice, and structured text. These media files are circulated and shared in public through the Internet, and they motivate many people to synthesize and convert these files to create value-added content.

Recently, various types of multimedia-processing software for processing and converting media data have become available through open source making it much easier for many people to access this type of software. For each media type, typical media processing libraries are supplied as open source libraries with free licenses. Some examples are ImageMagick, a typical image processing library, FFmpeg, a typical video processing library, and OpenCV, a typical computer vision library. Places where multimedia contents can be shared have also become popular, and many people post and share their handiwork.

However, user selected multimedia processing libraries do not always have a function that the user desires. To supplement lacking functions or to achieve desired functions, users need to install multiple software packages, learn to use them, combine them, and switch between them to accomplish the desired task. These tasks to achieve the desired functions become a large encumbrance to users especially those who process large amounts of data. One promising approach to address this problem is a Web service mashup technology that integrates Web service components through a Web Application Programming Interface (API). Many mashup environments have been proposed. Yahoo! Pipes is a well-known mashup environment that supports visual dataflow programming, processing of structured data such as XML, RSS, and JSON on the Web, and supports these types of data by facilitating their arrangement on Google maps. However, using these mashup environments require users to learn programming skills. Therefore, this becomes a high hurdle for most PC users who rarely write programs.

In this thesis, we propose an integration-environment architecture for media processing components that overcomes these barriers and makes it easy to mashup the components.

The thesis comprises the following parts. In Chapter 1, we define the end user, present some practical media processing examples and practical usage scenarios for media processing integration. We also define the objective of this study, and present an outline for our approach. The objective is to satisfy the following two requirements sufficiently and simultaneously in one environment architecture. The first requirement is a programming environment for end users, and the second requirement is a mashup environment for multimedia processing. In order to achieve this, we focus on the experiences and knowledge of folder management that many users already use in their daily work. Our research approach is to lower the barriers in learning programming skills that the end users need to use the software.

In Chapter 2 we describe related studies. As technologies related to the first requirement of the end-user programming environment, we describe trends from the point of view of existing end-user programming environments. From among these, we describe visual programming as an example of a promising approach. We also describe trends in multimedia processing mashup environments, the second requirement, and trends in file systems, which are closely related to our approach.

In Chapter 3, we propose a folder programming environment. The core of the folder programming environment is the "functional folder." This folder differs from the standard folder based on three feature mechanisms (ff 1), (ff 2), and (ff 3). (ff 1) In this mechanism, processing begins when a data file is dropped (DROP) onto the folder. The processing entity is determined and executed dynamically based on the folder name. The processing results are stored in the folder, (ff 2) The folder name contains arguments for processing, (ff 3) By dragging and dropping (Drag&Drop) multiple files onto one folder, they can be processed all at once. As the folder programming environment to actualize the functional folder that combines various processes, we implement the following six execution control mechanisms (fp 1) to (fp 6). (fp 1) Execution control mechanism for processing chain, (fp 2) Execution control mechanism for concurrent processing, (fp 3) Execution control mechanism for processing ' else ' and conditional branches, (fp 4) Execution control mechanism for control jump, (fp 5) Execution control mechanism for subroutine processing (abstraction and gathering), (fp 6) Execution control mechanism to enable mixing of data and functional folders.

In this chapter, we also present four features of the programming development support functions: Processing when input data causes an ERROR, debugging methods, a reference method for HELP information, and text expression of the folder program. Next, we describe in detail the software architecture of the proposed folder programming environ-

ment which is an enhancement of the Web folder, and the implementation of a prototype. Moreover, we describe the effectiveness of the programming environment for the end user based on qualitative evaluation using Ichisugi ' s design guidelines.

In Chapter 4, we discuss the applicability of the folder programming environment to a mashup environment for multimedia processing. We describe five requirements, (M1) to (M5), for integrating the media processing components. (M 1) Multimedia adaptability, (M 2) Web API adaptability, (M 3) Ease of using one function, (M 4) Ease of applied usage such as making a simple program, (M 5) Easy component management.

Next, we present experimental results and questionnaires targeting students that are based on experiments comprising simple image-processing tasks that use the folder programming environment. In addition, we introduce the Nijigumo Note system, a system developed by applying the folder programming environment, and describe the practical experience obtained from the process of constructing the system. Based on these, we show that the folder programming environment thoroughly satisfies the five requirements for the media processing component integration environment.

In Chapter 5, we confirm the quantitative effects of the folder user interface in the end user programming environment. To analyze the effect of the folder interface, which is the characteristic feature of the programming environment, we administer user-experiments based on a comparison of the folder interface to a traditional text interface that requires the user to write text program code. Test subjects solved five problems in both the folder and text user-interface programming environments. We analyzed the elapsed time to complete each problem, and we analyzed the collected questionnaires. Based on the comparison results, we confirmed three effects: the time spent to execute a simple functional folder is shorter, the time to execute a folder program for two or more input files is shorter, and the time to execute a folder program using a processing chain is shorter. Because the folder user interface requires only approximately half the time compared to the text user interface on average, the folder interface is easy. Moreover, approximately 90

Next, we present user experiment results regarding usability based on a comparison of the proposed folder programming environment and the visual programming environment "Pure Data" supplied as open source. The comparison results are based on using Nielsen's usability principle. The experiment compares the program editing tasks for several engineers based on observation of work error and the results of the questionnaire. The analysis results showed that since there is a high level of consistency with a desktop environment, and since there is no confusion between editing and executing environ-

ments, the proposed folder programming environment was very user-friendly, and very few operation errors occurred. On the other hand, Pure Data, which uses an original graphical user interface, caused many user-operation errors. Based on the results, we confirmed that the folder user interface was more user friendly.

In Chapter 6, we show how the proposed folder programming environment simultaneously satisfies in one environment architecture the requirements of a programming environment for end users and a mashup environment for multimedia processing. We present detailed analysis results showing a higher level of efficiency of the folder programming environment from the viewpoints of user acceptability (learning, recognition, thinking, and manual workload) and the environment (language specification, editing stage, executing stage, viewing stage of results, and modification stage). These points are summarized below in (a) to (g). (a) The user no longer needs to learn how to use the Web API because he or she can execute the desired processes based only on the knowledge of the Web folder, (b) Barriers to users in terms of the learning curve and the skill required for programming can be lessened because the user can use existing knowledge of folder operations, (c) The user can more easily overcome barriers in terms of the learning curve and the skill required for managing media data because the user can draw on existing knowledge of media expression, (d) The user can more easily learn, recognize, and think about immediate execution of processes and immediate display of media data, (e) The user no longer needs to use separate environments to consider editing and executing programs, (f) The experience becomes more user-friendly based on intuitive Drag&Drop operations, (g) The user can easily execute batch processes of large amounts of data using only one Drag&Drop operation.

The results of this study also show that the proposed environment supplies new middle steps aiming at a gentle learning curve so that the end user can solve more advanced problems. We believe that the proposed environment provides intermediate steps to improve user skill to support many PC users to become Web service DIY developers.

In Chapter 7, we summarize this thesis. There are many people who use PCs in their daily lives but have not written a program or even thought of programming since when they were in school. With the proliferation of the Internet and digital contents, these users now have access to many types of media data. This has given rise to the demand for DIY processes to handle digital contents. Our research approach is to use the user experiences and knowledge of folder management that many users already use in their daily work to lower the barriers in learning programming skills. We proposed an architecture that enhances folder programming in a Web folder environment. We showed

the feasibility of the architecture based on the implementation of Polder. Based on experiments and evaluation results, we also showed how the proposed folder programming environment simultaneously provides a programming environment for end users and a mashup environment for multimedia processing on one architecture. Using this environment, many people who use PCs in their daily lives but do not write programs are able to process many types of media data on their own. Furthermore, end users who are just learning DIY programming can gradually increase their skill level through intermediate steps. As a result, many PC users can become Web service DIY developers who can solve more advanced problems. In addition, we discussed topics for future works.

目 次

表 目 次

図 目 次

第1章　序論

1.1　背景

デジタルカメラ・スマートフォン・タブレット・ノートPCなどの携帯機器の普及に伴って，画像・映像・音声・構造化テキストなどの多様なデータを誰でもが気軽に生成できるようになってきた．それらのデータは，インターネットを介して人々の間で広く流通・共有されるとともに，合成や改変など新たな価値を付加する活動も活発になってきている．

メディアデータに対する処理，すなわち加工・改変をするためのソフトウェアは，近年，オープンソース・ソフトウェア（OSS [1]）化が進行し，多くの人々が簡単に入手できる環境が整ってきている．例えば，画像処理ライブラリではImageMagick [1], 映像処理ライブラリではFFmpeg [2], コンピュータビジョンのライブラリではOpenCV [3] など，各メディア分野を代表する処理ライブラリがソースコードも含めて無償で利用できるOSSとして提供されている．一方，加工・改変した作品を流通することができる場の整備も進んでおり，多くの人々が趣味あるいは趣味の延長線上で多くの作品を投稿し，共有するようになってきた．

しかし，利用者が選択したマルチメディア処理ライブラリが必ずしも所望の機能を過不足なく具備しているとは限らない．利用者は，不足機能を補って所望の機能を実現するために，複数のソフトウェアを導入し，それらを組合せて実行することを学び，切り替えながら実行しなければならない．こういったマルチメディア処理における所望の機能を実現する作業は，特に，大量のデータに処理を適用する利用者にとって大きな負担となる．その解決への有望なアプローチは，Webサービスとして提供される各種のアプリケーション・プログラム・インタフェース（API [2]）を経由して必要とする処理を統合するマッシュアップ環境[3]の利用である．マッシュアップ環境は既に多数提案されている．Yahoo! Pipes [4] は，代表的なマッシュアップ環境の一つであり，Web上のXML [4], RSS [5], JSON [6] など構造化データの加工や，その結果のGoogle地図 [5] 上への配置などを簡便に行えるデータフロー・ビジュアル・プログラミングを実現している．しかし，これらの環境を利用す

[1] Open Source Software
[2] Application Programming Interface
[3] 本論文で，マッシュアップとは，Web上のサービスやデータを組合せて新しいサービスを作成する行為をいう．
[4] Extensible Markup Language
[5] Really Simple Syndication
[6] JavaScript Object Notation

るには，利用者はプログラミング・スキルを習得しなければならず，日常的にプログラミングを行う機会が少ない一般的な PC [7] 利用者には，利用の障壁が高いままである．

1.2 本研究の目的と問題設定

本研究では，このような利用に関わる障壁を低減することで，幅広い PC 利用者がマルチメディア処理のマッシュアップを DIY [8] で簡便に行えるようにするためのメディア処理部品の統合環境を実現することを狙いとしている．その環境は，次の 2 つの要件を満たし両立する必要がある．その第一の要件はエンドユーザのためのプログラミング環境であること，そして，第二の要件はマルチメディア処理のマッシュアップ環境であることである．そこで，本研究の問題はこの 2 つの要件を同時に満たす環境の実現と設定する．

要件 1: エンドユーザのためのプログラミング環境　多くのエンドユーザに受け入れられるためには，言語仕様，及び，プログラムの編集作業から，実行，結果の確認，再編集という一連の流れに対して，使い始めるまでの新たな学習量が少なく，プログラムや編集・実行の状態を頭の中で思い浮かべる負担が少なく，作業量の点でも少ないといった点が重要である．そのため，可視化を効果的に利用するというビジュアル・プログラミングのアプローチが重要となる．

要件 2: マルチメディア処理のマッシュアップ環境　Web API として提供される多様なメディアに対応したメディア処理部品を簡便に利用でき，また柔軟に組合せて各種の応用に適用できる点が重要である．更に多数の部品の管理機能も重要となる．

これら 2 つの要件を同時に満たす環境は知られていない．第 2 章で詳述するように，Spahn(2008) [6] のエンドユーザ開発アプローチの分析の中で，Microsoft Excel に代表される会計パラダイム（表計算）は極めて成功したエンドユーザ・プログラミング環境だとしている．このアプローチは，多くのユーザが日常的に慣れ親しんだ表計算という枠組みを利用することで学習の障壁を下げ，また，即時に計算結果を反映することで認知思考量の削減ができ，結果としてエンドユーザに広く受け入れられている．しかし，本研究でもう一つの要件であるマルチメディア対応は難しく，メディア処理のマッシュアップ環境としては不十分である．

また，同じく，Spahn はデモンストレーションによるプログラミング (PBD [9]) も有望なアプローチだとしている．PBD は，簡単に言えばディスプレイ上のオブジェクトをマ

[7] Personal Computer
[8] Do It Yourself
[9] Programming by Demonstration [7]

ウス等で直接操作[10]し，その結果を記録し，再演することで同じ操作を簡単に実行できるようする仕組みであり，利用者の操作がそのまま記録される点でプログラミングに関する新たな学習を省略できるため，多くのエンドユーザにとって簡便な環境になるといえる．しかし，このアプローチには処理の汎用化もしくは一般化が難しいという本質的な問題が存在する．例えば，PBD の記録結果を汎用的なプログラムにするためには，引数や処理対象データの一般化といった課題，条件分岐や繰り返しの自動合成の課題など，未解決の多数の課題が存在する．更に，実演の失敗，途中部分の再利用・改変に対して簡単でかつ十分なサポートが難しいとう課題もある．

よって，先の 2 つの要件を同時に満たす環境の実現のためには，これらの会計パラダイム（表計算）のアプローチでも，PBD のアプローチでもない，新たなアプローチが求められている．

なだらかなスキル成長の支援

更に，本研究では幅広い PC 利用者がマルチメディア処理のマッシュアップを DIY で簡便に行えるようになるためのエンドユーザのスキルの成長支援も狙いとしている．

エンドユーザがプログラミングを学習していくためには MacLean(1990) [9] や Spahn が指摘するように，なだらかな傾斜となるようなスキル成長の中間ステップの提供が重要である．一般にプログラミングの目的は処理の自動化であり，何度も繰り返す処理の自動化である．しかし，保福 [10][11] が指摘するように，その繰り返し処理のプログラミングの理解やその前提となる変数や代入の理解などが，プログラミングを学習する初期の大きな障壁の一つとなっている．そのイメージを図 1.1 に示す．

本研究では，図中で示される大きな障壁の部分に対して，エンドユーザがプログラミングを徐々に学び，より高度な課題を解いていくために必要なスキルを自ら獲得できるようになるためのスキルの成長の中間ステップを，なだらかな傾斜となるように与えることで，その結果として多くの PC 利用者を DIY で Web サービスをより活用するライトな開発者に成長していくことを最終的な狙いとしている．

[10] Direct Manipulation [8]

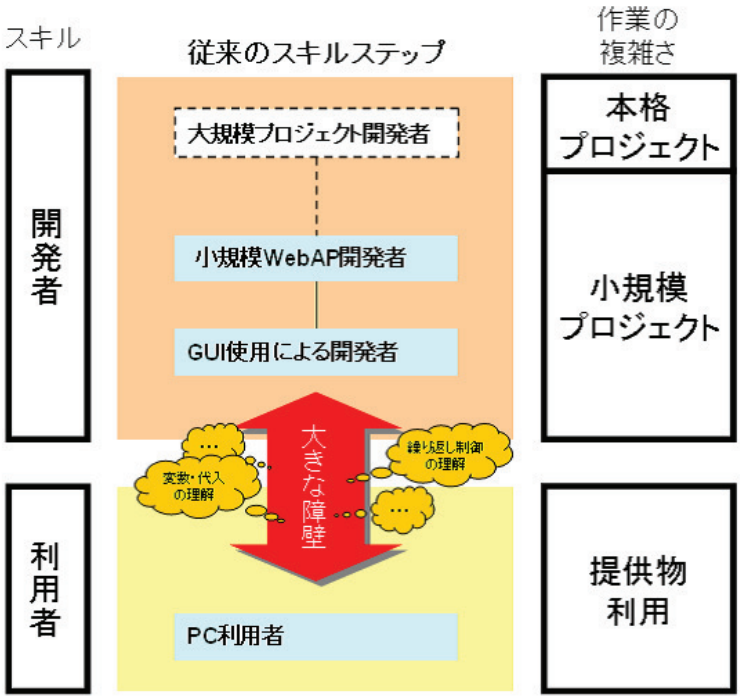


図 1.1: エンドユーザのスキルレベルと成長への障壁

Toffler(1980) の著書「第三の波」[12] では，第一の波である農耕社会では自給自足が中心の世界，第二の波である産業社会では生産者と消費者の分離と生産の交換による経済，そしてそれらを経た第三の波では，再び消費者が小規模な生産も行う DIY 時代が到来するとしている．Toffler はその人々を，**生産消費者（プロシューマ）**と呼んだ．それは，例えば農家や大工職人に生産を任せっきりだった一般消費者が，自らの手で身の回りにかかわる家庭菜園や日曜大工などの小規模な不定期生産を行う生産消費者に成長していく世界である．

同様の潮流は，エンドユーザ・プログラミングにもあり，今後も更に拡大すると考えられる．アプリケーション・ソフトウェアはこれまでパッケージングされて流通し，**開発者**（≒生産者）と**利用者**（≒消費者）は明確に分離されていた．しかし，インターネットでの流通や OSS を利用した Web サービスの台頭によって，所有するデータの増加と，それを自分用に活用する必要分のソフトウェア作成コストが低減し，マッシュアップの流れが登場している．それはまさしく，**利用開発者**（≒生産消費者）による DIY もしくは日曜プログラマ的な活動の拡大と捉えることができる．

開発者（プログラマ）と利用者の間に，自らに必要な小さなプログラムを書く人々，**利用開発者**という層が存在し，その層の拡大が起こっている．しかし，現在のマッシュアップ環境は，多くのエンドユーザにとっての十分に簡便な DIY の道具とはなっていない．

本研究は，より多くのエンドユーザに対して，簡便なマッシュアップ環境を提供し，DIYの流れを飛躍的に加速する技術の開発を目指している．

対象とするエンドユーザ

一般的に，エンドユーザとは，ソフトウェアやシステムを最終的に使うと想定される人を指す．それは，提供者や開発者側ではなく，利用者側である，ということであり多様なスキルの利用者を想定できる．ここで本研究のターゲットとする利用者の具体化を行う．

本研究で着目するエンドユーザは，**日常的にプログラミングを行わない PC 利用者**である．より丁寧に言えば，日ごろ PC を利用しており，過去にプログラミングの教育を受けたことはあるが，その後，ほとんどプログラミングを行っていない利用者である．これは Nielsen の分類 [13] で，初心者，熟練者に次ぐグループである不定期利用者（断続的な利用を行う，特別な状況で利用者に戻る，久しぶりに利用者に戻る等）にも相当する．

学校・職場・家庭など世の中に PC は広く普及し利用が進んでいる．すでに，高等学校では 2003 年度から教科「情報Ａ／Ｂ／Ｃ」が開始されている．また，中学校においても情報の教育は技術科において導入が進められており，現在の中学校の学習指導要領 [14] では,「Ｄ　情報に関する技術」において，図 1.2 のようにプログラミング教育が必須とされ，2012 年度から義務教育における完全実施となっている．このように，プログラミングを体験したことのある層は拡大している．

> (3)　プログラムによる計測・制御について，次の事項を指導する。
> ア　コンピュータを利用した計測・制御の基本的な仕組みを知ること。
> イ　情報処理の手順を考え，簡単なプログラムが作成できること。

図 1.2: 中学学習指導要領 [14]

全くプログラミングの未経験者（これを**経験無・利用者**と呼ぶ）ではなく，プログラミングをよくする**開発者**でもない，その中間に位置するグループの層を我々は，本論文ではプログラミングは過去に経験したことがあるが，プログラミングは日常的には行わない**経験有・利用者**と呼ぶ．

先に述べたように，中学・高校・大学時代に，少しだけプログラミングの授業を（場合によっては受身で）履修したことはあるが，それ以降は利用していない社会人・学生・主婦など，日常的に PC は利用しているがプログラマ（開発者）とは言えない学生・社会人

は極めて多いし，これから中学を卒業する全国民のレベルで増加していくことは間違いない．しかし，多くの人々は，再びプログラミングに取り組むことはないのが実態であろう．

本研究では，このような PC を利用するだけに留まっている彼ら／彼女らが，Web 上で提供される多様なメディアの処理の統合活用，つまり簡単なプログラミング（ライトな開発）行為を行えるようにする環境のソフトウェア・アーキテクチャの具現化を目指している．

メディア処理種別と処理統合の活用シーン

代表的なマッシュアップ環境である Yahoo! Pipes [4] は，主な対象を RSS 等の構造化データに限定することで，成功した事例といえる．しかし，我々の手元の PC には，構造化テキストだけではなく，デジタルカメラ・スマートフォン・タブレット・ノート PC などで撮影記録した画像・映像・音声など多様なメディアが溢れており，端末も各種のメディアに対応する Viewer,Player,Editor を搭載することでデスクトップ上でのマルチメディア・データ管理に対応している．なお，本研究における「メディア」とは，OS がファイルタイプとして認識するファイルとする．それは画像・映像・音声だけに留まらず，Microsoft PowerPoint 文書なども含むものとしている．

多様なメディア処理： それらの多種多様な所有データに対するメディア処理が求められる．本研究では，例えば以下のようなメディア処理部品を想定する．

- カメラで撮影した写真の加工・補正・サムネイル化.
- ビデオカメラで撮影した映像の代表画像群の作成.
- IC レコーダで録音した会議内容の音声認識によるテキスト化.
- 文献の音声合成による音声化.
- スキャニング画像の OCR [11]によるテキスト化.
- 名刺画像の文字認識.
- 画像の内容特徴量やメタ情報による自動分類.
- Web サイトのブックマークの自動分類.
- Microsoft PowerPoint 文書の差分比較.
- メタ情報の DB [12]登録.
- 写真ファイルの Web サービスへのアップロード.

[11] Optical Character Reader
[12] Database

- Exif[13]情報のメンテナンス．

本研究成果の適用範囲となるメディア処理は上記に限定されない．例えば，今後 3D プリンタの急速な普及に伴う 3 次元データの大量流通が予想されるが，本論文で提案する基盤は，それらの新しいメディアの出現に対しても，柔軟に追随することが求められる．

メディア処理の組合せ：　メディア処理部品は，単独で利用するだけではなく，処理の組合せが必要になる．典型的なケースとしては，メディア処理の入出力フォーマットの変換という作業である．手持ちのメディア・フォーマットが，その処理部品の入力仕様に適合しない場合が考えられる．例えば，flv [14]フォーマットの手持ちのコンテンツを mpeg 用の処理に適用したい場合がこれに相当する．また，類似のケースとして出力フォーマットが利用者の期待のフォーマットでないケースも考えられる．例えば，wma [15]フォーマットの音楽を mp3 [16]のプレイヤで再生したい場合がこれに相当する．その他にも映像のコーデック・コンテナの形式・フレームレート，音声のステレオ・モノラル・サンプリングレートなど，多数のケースが存在する．そのような時に必要な入出力データのフォーマット変換の処理を，中心となるメディア処理部品の前後に接続することが必要となる．このようなケースにおいて，簡単なメディア処理の組合せが必要である．また，異なるメディア処理の組合せとしては，映像から代表画像群を作成し，更に，それらをサムネイル化するケースなどがそれに該当する．更に，メディアから抽出されたメタ情報や特徴量を使って自動分類することや，DB に登録すること，Web にアップロードすることなども，組合せのケースに該当する．

大量データへの適用：　複数のメディア処理の組合せに加え，手元の写真データが大量にある場合には，それら処理を繰り返し適用が必要になる．

更に，大量のデータがある場合，その中に複数のメディア種別が混在することがある．例えば，現在のデジタルカメラは，写真撮影も映像撮影も可能になっており，一つのフォルダ内に写真と映像が混在していることが多い．それらに対して，一斉に処理を行う際に，映像だけに，処理を行いたいというケースや，メディア種別毎に異なる処理を行いたいというケースが存在する．

以上のように，メディア処理の統合活用のためには，多様なメディアに対する処理部品の実行に限らず，その部品の組合せ接続や入出力フォーマット変換のための組合せ接続，大量のデータへの繰り返し適用やメディア種別に応じた処理の切り替えなどが，必要となる．

[13]Exchangeable Image File Format
[14]Flash Video
[15]Windows Media Audio
[16]MPEG Audio Layer-3

1.3 本研究のアプローチ

本論文では，エンドユーザによるメディア処理のための部品統合環境の構成法についての提案を行う．

本研究の中心となるアイディアは，デスクトップ環境のフォルダを拡張することで，プログラミング環境を構成することである．それをフォルダ・プログラミング環境と呼ぶ．フォルダ・プログラミング環境におけるプログラミングのためのフォルダの拡張の中心原理は非常にシンプルであり，以下のようなものである．

- メディア処理部品へのデータの入力と実行は，フォルダへのデータファイルの Drag& Drop [15] [16] で行われる．入力されたデータファイルは，予めシステムで定義されたフォルダ名に対応したプログラムによって処理され，処理結果のデータファイルはそのフォルダ内に格納される．
- 階層的なフォルダ構造を作ることで，処理結果のデータファイルは更にその下位の層のフォルダへの入力となり，処理は連鎖する．

フォルダは，デスクトップ上の代表的なオブジェクトとして広く PC 利用者に認知され，日々の作業で利用されている．その経験と知識を最大限に活用することで，日常的にプログラミングを行わない PC 利用者にとっても新たな学習量を最小にしてプログラム作成編集環境とプログラム実行環境を得ることができる．

更に，デスクトップは，既に多様なメディアに対する表示アプリケーション (Viewer)，再生アプリケーション (Player)，編集アプリケーション (Editor) と密接に連動しているため，処理の入出力データの確認等が極めて簡単で，その点においても新たな学習は不要になる．それらのアプリケーションの追加によって，デスクトップは新たなメディアの管理にも追随していく．

また，フォルダはシームレスに Web に対応している．HTTP[17] [18] の拡張規格である WebDAV[19] [20] を利用することで，Web 上に存在するフォルダを簡単にデスクトップからアクセスすることができる．

本論文では，上述のアイディアをプロトタイプの実装や具体的なメディア処理への応用を通して，その実現性と有用性を確認している．

1.4 本論文の構成

第 1 章では本研究の目的と問題設定を行い，本研究の対象のエンドユーザを設定し，想定するメディア処理種別と処理統合の活用シーンの具体化を行い，アプローチの概要を示している．本研究の目的は，次の 2 つの要件を満たし両立する環境の実現である．その第一の要件はエンドユーザのためのプログラミング環境であること，そして，第二の要件はマルチメディア処理のマッシュアップ環境であることである．その解決のため，本研究で着目したポイントは，すでに多くの利用者が日常的に活用しているフォルダ管理という作業の経験と知識である．それをプログラミングに利用することで，エンドユーザに対するプログラミング・スキルの習得の障壁の低減を目指すアプローチを採っている．

第 2 章では，関連技術を示す．第一の要件であるエンドユーザのためのプログラミング環境に関連する技術として，既存のエンドユーザ・プログラミングの視点からの動向の紹介，その中で特に有望なアプローチとされるビジュアル・プログラミングの動向について示している．第二の要件であるマルチメディア処理のマッシュアップ環境に関連する技術としてマッシュアップ・システムの動向を紹介し，本研究のアプローチに密接に関連するファイルシステムの動向についても示している．

第 3 章では，フォルダ・プログラミング環境を提案している．フォルダ・プログラミング環境の核となる構成要素である『処理付フォルダ』は，通常のフォルダの仕組みに以下の (ff1)〜(ff3) の 3 つの仕組みを加えて拡張する．(ff1) フォルダへのデータファイル DROP で処理が起動し，フォルダ名をもとに処理が動的に決定し実行され，結果がフォルダ内に入る仕組み，(ff2) 処理の引数をフォルダ名に含む仕組み，(ff3) 複数ファイルの Drag&Drop に対して，一斉に実行される仕組み．処理付フォルダを組合せて多様な処理を実現するためのフォルダ・プログラミング環境として，以下に示す (fp1)〜(fp6) の主要な実行制御機能を実現する．(fp1) 処理が連鎖する実行制御，(fp2) 並行の実行制御，(fp3) 条件分岐と else の実行制御，(fp4) 制御移行の実行制御，(fp5) サブルーチン（抽象化と収集）の実行制御，(fp6) データフォルダと処理付フォルダの混在に対する実行制御．更に，プログラム開発支援機能として，ERROR 時の入力データ，デバッグ方法，HELP 情報の参照，フォルダ・プログラムのテキスト表現といった 4 つの機能を実現していることを述べている．次にフォルダ・プログラミング環境を Web フォルダの拡張として実装するためのソフトウェア・アーキテクチャ，プロトタイプの実装の詳細を示している．最後に，本論文の第一の要件である，エンドユーザのためのプログラミング環境としての定性的な評価を，一杉の設計目標 [21] などを使って行い，その有効性を述べている．

第 4 章では，本論文の第二の要件であるマルチメディア処理のマッシュアップ環境に対して，フォルダ・プログラミング環境の適用性について議論を行っている．まず，メディア処理部品の統合環境として求められる 5 つの要件 (M1)〜(M5) を示している．それは，(M1) マルチメディア対応性，(M2) Web API 対応性，(M3) 単機能利用の容易性，(M4) 応用利用の容易性，(M5) 管理の容易性，である．

次に，フォルダ・プログラミング環境を使った画像ファイルの簡易な処理に対する利用者実験の結果を示している．更に，フォルダ・プログラミング環境を応用して開発した虹雲ノートシステムの紹介と，そのシステムを構築する過程で得られた使用経験を示している．その後，フォルダ・プログラミング環境がメディア処理部品の統合環境で必要とされる 5 つの要件に対して適合性が高いことを示している．

第 5 章では，本研究の第一の要件である，エンドユーザのためのプログラミング環境としてのフォルダ・ユーザ・インタフェースの定量的な効果を確認している．フォルダ・プログラミング環境を特徴付けるフォルダ・インタフェースの効果を分析することを目的に，まずコードを記述する従来型のテキスト・インタフェースと比較する利用者実験を行い，その効果を分析している．実験参加者には複数の課題を両方の環境で解いてもらい，作業時間の分析とアンケート分析を行った．課題別の効果分析の結果，単純に処理付フォルダを実行する際の取り組み易さの点と，複数の入力ファイルに対して処理を実行する課題に大きな効果があり，処理の連鎖を使ったプログラムの作業時間短縮の効果を確認している．フォルダ・ユーザ・インタフェースは課題全体で約半分の時間に短縮することを確認している．

次に，他のビジュアル・プログラミング環境として OSS で入手可能な Pure Data と，フォルダ・プログラミング環境とでユーザビリティの比較実験を行った．数名の技術者を対象としたプログラム編集作業の比較実験を行い，作業エラーやアンケートの結果をもとに，Nielsen のユーザビリティ原則の観点で両者を比較している．比較・分析の結果，フォルダ・プログラミング環境は，デスクトップとの一貫性が高いことから取り組みやすく，編集と実行の環境を混乱することもないため操作エラーが少ないことなどが確認している．一方，独自のグラフィカル・ユーザ・インタフェースを使う Pure Data は，利用者の操作エラーが多数発生する結果となった．これらのことからフォルダを使ったユーザ・インタフェースの優位性を確認している．

第 6 章で，本研究が問題とする，エンドユーザのためのプログラミング環境と，マルチメディア処理のマッシュアップ環境とを両立する環境の実現に対して，本研究で提案するフォルダ・プログラミング環境がどのように解決したかを考察しまとめている．まず，ユーザの受容性の側面（学習量，認知思考量，作業量）と環境側面（言語仕様，編集時，実行時，結果確認時，応用時）から分析したフォルダ・プログラミング環境の効果を，以下の

(a)〜(g) として抽出している．それは，(a) Web フォルダの知識のみで処理を実行できるので Web API の学習が不要，(b) フォルダ操作の既存知識を活用できるのでプログラミング環境の学習・習熟が極少，(c) メディア表示の既存知識を活用できるのでプログラミング環境の学習・習熟が極少，(d) 処理の即時実行機能とメディアデータの即時表示機能によって学習と認知思考が容易，(e) プログラムの編集と実行の環境の切り替えが不要なので認知思考が容易，(f) 直感的な Drag&Drop 操作が使えるので認知思考が容易，(g) 大量データへの一括処理が Drag&Drop で簡便に実行できるので作業量が低減，である．

更に，エンドユーザがより高度な課題を解いていくために必要となる，学習のなだらかな傾斜の実現に対して，本研究の成果が，複数の新たな中間ステップを提供することを示している．漸増的にスキル成長させる中間ステップが，DIY で Web サービスを活用するライトな開発者（利用開発者）へと多くの PC 利用者を導く補助的中間ステップになることについて考察している．

最後に第 7 章で，本論文を要約し，今後の課題と展望について述べている．

第2章　関連技術

本章では，本研究の問題とする 2 つの要件の解決に関連する技術を示す．まず，エンドユーザ向けのプログラミング環境に関連する技術として，エンドユーザ・プログラミングの動向，及び，その中で特に有望なアプローチとされるビジュアル・プログラミングの動向について示す．次に，マルチメディア処理のマッシュアップ環境に関連する技術としてマッシュアップ・システムの動向を示す．最後に，本研究のアプローチに密接に関連するファイルシステムの動向について示す．

2.1　エンドユーザ・プログラミング

2.1.1 プログラミング

プログラミングとは「プログラムを作成すること」である．**プログラム**は,「コンピューターに対して、どのような手順で仕事をすべきかを、機械が解読できるような特別の言語などで指示するもの（広辞苑 [22]）」,「コンピュータによって所定の問題を解くとき、コンピュータに与える処理手順を順序よく並べたもの（日本国語大辞典 [23]）」などとなっている．

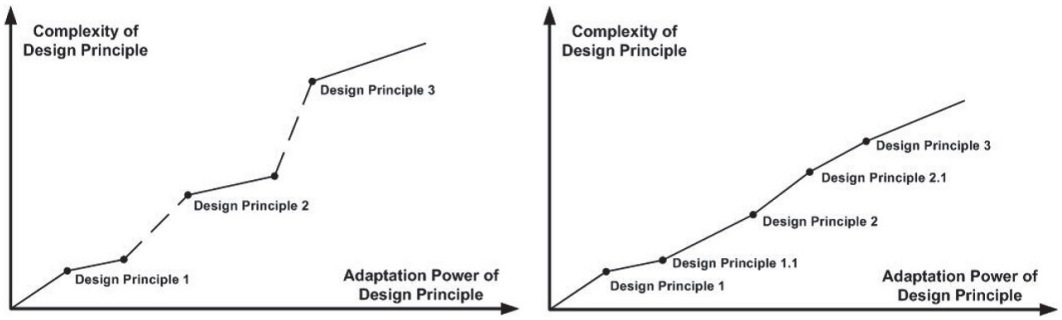
より丁寧にプログラミングを解釈すると，**プログラミング**とは「計算機が処理可能な人工言語で記述した処理手順を使って計算機に指示を与えること」である．処理手順（プログラム）は，複数の処理の順番を言っており，本論でいう処理の連鎖は必須となる．よって，例えば，処理の途中で人間がパラメータを調整する行為自体は，プログラミングには含まれないと考えられる．また例えば，後に議論する直接操作を使ったプログラミング等では，逐次処理のみのプログラムの合成が可能である．一方，チューリングマシンの計算能力を考えた場合には，繰り返しや条件分岐といった処理手順が必要となる．そこで，本研究のターゲットとするプログラムは，**逐次処理**に加え，**繰り返し処理**，**条件分岐処理**が実現できることを必要条件として考える．

更に，プログラムを作成する場である**プログラミング環境**は,「プログラムの作成，実行，結果確認，修正，の一連の作業を行う統合された開発実行環境」とする．

2.1.2 エンドユーザ・プログラミングにおける学習の障壁

MacLean(1990) [9] は，利用者による仕立て可能なシステム (User-Tailorable System) について発表を行った．その中で，仕立てのために要求される利用者のスキルの高さと，仕立ての能力の大きさの 2 つの軸を持つグラフを利用してプログラミングのスキル成長の議論を行った．そのグラフでは，左下領域が「仕事をしたいがコンピュータシステムには興味が無い人々(Worker)」を表し，右上領域が「プログラミングできる人々(Programmer)」を表す．そこにスキルの高さに応じて仕立ての能力が向上するという右上がりの単調増加曲線を考え，その途中に急な傾斜があると，利用者はそれ以上の高度な仕立てのためのスキル獲得の障害となり，そこに，いくつかの中間ステップを提供してなだらかな傾斜にすることで，利用者には徐々にスキルが獲得され，自ら仕立てが可能になるとした．

Spahn(2008) [6] は MacLean の主張を参照しながら，エンドユーザ開発について同様の議論を行っている．図 2.1 において，縦軸を設計原理の複雑さ，横軸を設計原理の適応能力として，なだらかな傾斜の有用性を主張している．



急な傾斜　　なだらかな傾斜

図 2.1: 設計原理の複雑さと適応能力（Spahn [6]）

この急な傾斜の具体例として，保福 (2013) は「なぜプログラミングは難しいのか？ 繰り返しの理解構造」[10] [11] において，高校生に対するプログラミング教育の経験から，プログラミングで躓きやすい箇所として「繰り返し」を抽出している．更に，その理解の難しさを，繰り返しの理解の前に，変数，代入，インクリメント，比較演算子，条件式，条件分岐，ブロックなど多数の事前理解が必要になることと分析し，それらを小さなステップとして良い順序で教えていくことの重要性を述べている．単純な固定回数の繰り返しの概念は小学生でも理解ができるほど簡単であるが，それが汎用化されたプログラミング言語を使うと変数が必要となるためにそれほど単純にはならず，理解が難しくなると分析している．

一般にプログラミングの目的は処理の自動化であり，何度も繰り返す処理の自動化である．しかし，その繰り返し処理のプログラミングの理解が，プログラミングを学習する初

期の大きな障壁の一つとなっている.

2.1.3 エンドユーザ開発アプローチの分類

更に，Spahn は表 2.1 に示すエンドユーザ開発 (EUD) [1] のアプローチの分類を行った.

表 2.1: エンドユーザ開発アプローチの分類（Spahn [6]）

EUD Approaches				Supportive EUD Approaches
Complexity / Adaptation Power	Customisation	Integration	Extension	
Programmers			Programming	**Testing:** Question-based testing; WYSIWYT; Integrity checks; Exploration environments **Community aspects:** Configuration files **Appropriation support**
Local Developers		Component swapping at runtime; Separated tailoring interfaces	Natural Programming; Scripting	
Non-Programmers	Interface customisation; Parameterisation	Programming by demonstration (PBD); Accountants paradigm; Integrated tailoring interfaces		

この表 2.1 の縦軸は，設計原理の複雑さはエンドユーザへの要求スキルに対応するため，Nardi (1990) に従いエンドユーザを以下の 3 種類に分類した.

Non-programmers: 非プログラマは，プログラミング教育が極めて少ないか無く，コンピュータへの興味が不足する人々.

Local developper: ローカル・デベロッパは，特定の分野のエキスパートで，特有のプログラムの知識を持っている人々.

Programmers: プログラマは，コンピュータの良い教育を受け，広い技術知識を持っている人々.

横軸は，設計原理の適応能力[2]は，Morch(1997) に従い以下の 3 つに分類した.

Customisation: 最初のレベルであるカスタマイゼーションは，アプリケーションの Look and Feel の仕立てとして使われるアプローチなど.

[1] Spahn は，End User Programming, End User Computing, End User Tailoring をまとめて End User Development と呼んでいる.

[2] 本論文では表中の EUD Approachs についてのみ考慮し，Supportive EUD Approaches については扱わない.

Integration: 2 つめのレベルである統合は，ある種のモデルを使うことでアプリケーションの内部設計変更を許す全てのアプローチ.

Extension: 3 つめのレベルである拡張は，プログラムコードの変更を含む全てのアプローチ.

ここでは本研究ターゲットである **Integration** の部分に含まれる各項目に着目する.

実行時のコンポーネント交換 (Component swapping at runtime) のアプローチはコンポーネントのビジュアライズな配置でシステムを構成し，コンポーネント (例:JavaBeans) の実行時の変更，追加，及び構成からの削除を許す情報システム.

デモンストレーションによるプログラミング (PBD) のアプローチ [7] は，エンドユーザにコンピュータ上でのアルゴリズムの実演を可能にし，コンピュータはそのアクションを追跡して記録し，それを再実行 (再生) を許すものである．マクロ記録機能などとして多くのアプリケーションに導入されている.

会計パラダイム (Accountants paradigm) のアプローチは，データの表形式を使ったスプレッドシートであり，このパラダイムは，多くの人にとって事前知識がなくても理解が容易であるため広く普及している．セルの間の関係式を記述することがプログラムを書くことになるため，未経験の利用者にとっても容易である.

統合されたテーラーリング・インタフェース (Integrated tailoring interfaces) のアプローチは，アプリケーションの設計時と実行時のビューをシームレスにする．つまりエンドユーザの結果の表示を即時に提供しながら，設計時と実行時のモードスイッチを必要としない．例えば，スプレッドシートのプログラムロジックは，数式の変更や生成によっていつでも変更できるため，統合されたテーラーリング・インタフェースである.

Microsoft Excel に代表されるスプレッドシートは極めて成功したエンドユーザ・プログラミング環境だと言える．スプレッドシートでは，会計パラダイムのアプローチと統合されたテーラーリング・インタフェースのアプローチ，実行時のコンポーネント交換のアプローチを活用している．しかし，本研究で狙いとするマルチメディアには対応できておらず，メディア処理のマッシュアップの分野では成功していない.

デモンストレーションによるプログラミング (PBD) も期待が大きいアプローチの一つである．初期には Gould(1984) による Programming by Rehearsal システム [24] がある．また，後述の Automator [25] ，Photoshop [26] の Action，Pursuit [27] ，更には，Microsoft

Excel も PBD でマクロを記録することができる．このように逐次処理のプログラムの記録と再演には，既に広く利用されている技術と言える．

しかし，この PBD アプローチの素直な延長上には汎用プログラムに至るなだらかな傾斜は今のところ見つかっていない．まず，このアプローチには汎用化もしくは一般化が難しいという本質的な課題が存在する．例えば，PBD の結果を汎用的なプログラムにするためには，引数や処理対象データの一般化といった課題，条件分岐や繰り返しの自動合成などの課題 [28][29] の解決が必要となる．その課題に対して，例えば，EAGER[30] は HyperCard 上でのマクロを生成するシステムであり，ユーザの操作に繰り返しを発見すると，ユーザに次の操作を予測して表示して確認を求める方法で繰り返しの半自動合成を行っている．また，より詳細な情報を入力するために操作のトレースを使って帰納推論を行うアプローチ [28][31]，帰納推論を言語構造自身に組込むアプローチの [31] など，多数のチャレンジ [7][28][29] が存在する．しかし，帰納推論 [32] は本質的に難しい課題であるため，広い問題領域に対して解決できる十分な手法は確立していない．これまでの多くのチャレンジは，GUI 分野などに限定した狭い問題領域を設定し，丁寧な例示と適切なユーザ対話によるプログラミング補助によって条件分岐や繰り返しを合成するものに留まっている．また，実演の失敗，途中部分の再利用・改変に対して簡単でかつ十分なサポートが難しいという別の課題もある．

2.1.4 エンドユーザ向けのスクリプト言語の設計目標

表 2.1 の Extension のアプローチに分類された表の中に，自然言語によるプログラミング（Natural Programming）が存在した．プログラミング言語を母国語である日本語に近づけようとする試みは，これまで多数行われている．Forth [33] の逆ポーランド記法に着目し日本語表現を可能にした片桐による「Mind」[34]，簡易な日本語表現を使い充実したオフィス処理の連携機能が提供される酒徳（クジラ飛行机）による「ひまわり」と，その後継としてオブジェクト指向を導入し更に日本語らしい記述を可能にした「なでしこ」[35][36]，Visual Basic をベースに日本語化を図った馬場による TTSneo [37] と，その後継で.NET Framework をベースに更に日本語らしい表現を可能にした「プロデル」[38]，JavaVM 上で動作可能にしプログラミング教育を目的として自然な文章による表現を目指した岡田による「言霊」[39] と，その後継として Squeak [40] をベースに日本語化をした「ことだま on Squeak」[41]，プログラミング教育を目的として Logo の考え方やオブジェクト指向を取り入れた兼宗による「ドリトル」[42][43] [44]，更に，エンドユーザ・プログラミングを志向したスクリプト言語として一杉のチャミー [21] などがある．

これらのプログラミング言語の日本語化による初心者向けプログラミング言語としての評価として，少なくとも英単語に抵抗のある初心者に対しては初期の障壁の一つを下げる効果があると言えるだろう．更に一定のプログラミング言語の文法理解を前提とすれば，

命令や変数などを日本語化することで，可読性を向上させる効果もある．しかし，日本語プログラミング言語といっても全ての日本語表現を受け入れるわけではないのが現状で，各言語仕様が要求する日本語らしさを目指す独自の制限を持つ文法をプログラミングの初心者がどう受け入れるかについては，プログラミング言語の日本語対応という特徴以外の部分に大きく依存している．

産業技術総合研究所の一杉によって設計されたチャミーの IDE(統合開発環境) は，日本語化された構造エディタ，実行トレースの可視化表示などの機能を持ち，初心者にとって習得しやすいプログラミング言語はいかにあるべきかを考察しながら，文法，制御構造，組み込みデータ型，標準ライブラリ，開発環境（エディタ，デバッガ）の設計を行っている点に特徴がある．一杉は，初心者にとって習得しやすいエンドユーザ向けのスクリプト言語チャミーの設計目標として 6 項目の設計目標をあげている（図 2.2）．

(ch1) 一度基本を習得すれば，マニュアルを見なくてもプログラムの読み書きができる．

(ch2) 母国語だけでプログラムが読み書きできる．

(ch3) よく行う実用的処理が簡単に書ける．

(ch4) 自然な編集操作で，間違いのないプログラムが書ける．

(ch5) デバッグが簡単にできる．

(ch6) プログラマにも違和感なく使えるように，またプログラミング言語の入門用に用いられるように，普通の手続き型プログラミング言語にできるだけ近い実行モデルをもつ．

図 2.2: エンドユーザ向けのスクリプト言語の設計目標 (一杉 (2005)) [21]

これら目標は，我々の想定する日常的にプログラミングを行わない PC 利用者に対しても重視すべき要件になると考えられる．この一杉の設計目標は，エンドユーザ・プログラミングを指向した言語であっても既存のプログラマも違和感がなく使えることの重要性を指摘している点 (ch6) や，よく行う実用的な処理が簡単に書ける点 (ch3) など，現実的側面も含むところに特徴がある．だたし，チャミーが対象としたエンドユーザのスキルは，

既存のプログラムの大体の意味を理解することができるレベルであり，我々の狙いである繰り返し処理にも抵抗のある利用者よりは高いプログラミング・スキルを要求している．

2.1.5 ユーザビリティ評価法

ユーザと対話を行うシステムのユーザ・インタフェースのユーザビリティに関する評価基準がいくつか存在する．Nielsen(1990) によるユーザ・インタフェースのユーザビリティの観点は広く利用されており，ユーザビリティは図 2.3 に示す 5 つの特性からなる多角的な構成要素を持っている．特に，学習しやすさが最も根本的としている．

- **学習しやすさ:** システムは，ユーザがそれを使って作業をすぐ始められるように，簡単に学習できるようにすべき．全部を完全にマスターできるかではなく，必要な作業をこなせるまでの学習の時間の短さ．
- **効率性:** システムを学習し習熟したユーザにとって，高い生産性が上げられるような効率的な使用を可能にすべき．
- **記憶しやすさ:** システムは，しばらく使っていなかった不定期利用のユーザが，再び使うときに覚えなおさないで使えるよう，覚えやすくすべき．
- **エラー発生率:** ユーザがシステム使用中にエラーが発生しにくく，発生しても回復できるようにすべき．致命的エラーは起こさない．
- **主観的満足度:** システムを使うことで，満足する，好きになる，楽しく利用できるにすべき．

図 2.3: ユーザビリティ特性 (Nielsen(1990)) [13]

このユーザビリティ[3]を実現するための原理や指針としては，1980 年代から多数のものが提案されている．Shneiderman(1987) [45] は，優れた UI の実現に必要な取り組みを網羅的に紹介する中で，**対話設計における 8 つの黄金律** (図 2.4) を提唱している．人間中心設計のアプローチを推進した Norman(1988) [46] は，**ユーザ中心デザインの 7 つの原則** (図 2.5) を提唱している．更に，Nielsen(1990) [13][47] は，一般的なエンドユーザ向けシ

[3]ここではユーザビリティに類するものも含める．

ステムのユーザ・インタフェース・デザインについて，**10 ヒューリスティックス・ユーザビリティ原則** (図 2.6) を示している．この Nielsen と Norman は NNg(Nielsen and Norman Group)[48] においてユーザビリティおよびユーザエクスペリエンスに関する活動を継続している．IBM, Apple, Google などの大手企業もそれぞれ独自の基準を有している．また，国際標準では ISO 9241-110(2006) [49], JIS Z8520(2008) [50] が，「人間工学 -人とシステムとのインタラクション-」において，**対話の原則**として，インタラクティブシステムを設計し，評価する上で重要なものとして，次の 7 つの原則 (図 2.7) を揚げている．

(us1) 一貫性を持たせる．
(us2) 頻繁に使うユーザには近道を用意する．
(us3) 有益なフィードバックを提供する．
(us4) 段階的な達成感を与える対話を実現する．
(us5) エラーの処理を簡単にさせる．
(us6) 逆操作を許す．
(us7) 主体的な制御権を与える．
(us8) 短期記憶領域の負担を少なくする．

図 2.4: 対話設計における 8 つの黄金律 (Shneiderman(1987)) [45]

(uc1) 外界にある知識と頭の中にある知識の両方を利用する.
(uc2) 作業の構造を単純化する.
(uc3) 対象を目に見えるようにして，実行のへだたりと評価のへだたりに橋をかける.
(uc4) 対応付けを正しくする.
(uc5) 自然の制約や人工的な制約などの制約の力を活用する.
(uc6) エラーに備えたデザインをする.
(uc7) 以上の全てがうまくいかないときには標準化する.

図 2.5: ユーザ中心デザインの 7 つの原則 (Norman(1988)) [46]

(uh1) シンプルで自然な対話を提供する.
(uh2) ユーザの言葉を使う.
(uh3) ユーザの記憶負担を最小限にとどめる.
(uh4) 一貫性を保つ.
(uh5) フィードバックを提供する.
(uh6) 出口を明らかにする.
(uh7) ショートカットを提供する.
(uh8) 適切なエラーメッセージを使う.
(uh9) エラーを防ぐ.
(uh10) ヘルプとドキュメンテーション.

図 2.6: 10 ヒューリスティックス・ユーザビリティ原則（Molich and Nielsen(1990)) [13]

(is1) 仕事への適合性.（インタラクティブシステムが，ユーザーが仕事を完了する上での助けとなり，完了を促進する）

(is2) 自己記述性.（ユーザがシステムとの対話において，自分が何についての対話をしているか，対話のどのステップにいるのか，どのような操作が許されていてどのように操作を実行すればよいかが常に明らか）

(is3) ユーザーの期待への一致.（対話が状況に応じて予想されるユーザーの必要性及び広く受け入れられている習慣と調和している．一貫性も含まれる）

(is4) 学習への適合性.（対話において，ユーザーがシステムの使い方を学習することを支援しその案内を与える）

(is5) 可制御性.（ユーザーが目標を達成するまで，やり取りの方向及びベースを主導し制御できる）

(is6) 誤りに対しての許容度.（入力で明らかな間違いがあったにもかかわらず，ユーザーによる最小限の修正で意図する結果が得られる）

(is7) 個人化への適合性.（個人の能力及び必要性に応じてインタラクション及び情報の提供を変更できる）

図 2.7: 対話の原則 (ISO 9241-110(2006)[49],JIS Z8520(2008)[50])

これらのユーザビリティの観点は，いずれも多くの点において類似したものとなっている[51]．その中で現在広く使われているのは，Nielsen のユーザビリティ原則である．Nielsen は，設計ではなく評価の観点でのユーザビリティに着目し，5 人程度，少なくとも 3 人の評価者から評価を集めると妥当な結果が得られることを示している．この 10 ヒューリスティックス・ユーザビリティ原則は，プログラミング言語を対象としたものではないが，視覚的環境を利用するビジュアル・プログラミング環境の評価観点としても適用できるため，本研究では評価観点の一つとして，これを利用する．

2.1.6 エンドユーザ・プログラミングの分析観点

本節では，更なる分析の観点として文献 [52] で利用されたエンドユーザ・プログラミングの分析観点テンプレート（表 2.2）を示す．このテンプレートを以下では EUP 分析観点テンプレートと呼ぶ．この表は，利用者の内的要因である**ユーザの受容性側面**の「u1:学習量」「u2:認知思考量」「u3:作業量」と，外的要因である**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:応用時」の 2 つの側面での分析を支援する．ここで，内的要因であるユーザ受容性側面における学習量は，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，認知思考量[4]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，操作に関する量である．一方の，外的要因は，プログラミング言語と環境に対する分析の観点を提供しており，言語仕様と，利用者がプログラミング環境を利用する際の各フェーズを列挙したものである．

本研究ではエンドユーザ・プログラミングの分析観点として，この表を利用する．

表 2.2: エンドユーザ・プログラミングの分析観点テンプレート

		環境側面（外的）				
		e1 言語仕様	e2 編集時	e3 実行時	e4 結果確認時	e5 応用時
ユーザ受容性側面（内的）	u1 学習量					
	u2 認知思考量					
	u3 作業量					

[4]認知 (心理学用語) とは，人間が外界にある対象を知覚した上でそれが何であるかを判断したり解釈したりする過程のこと．思考とは，心に色々な事柄を思い浮かべる行動を通じてそれらの関係を構築する作業のこと．

2.2 ビジュアル・プログラミング

2.2.1 ビジュアル・プログラミングの概要と分類

プログラミング言語を自然言語に近づけるアプローチのほかに，エンドユーザ・プログラミングに関するもう一つの大きなアプローチが視覚を活用するビジュアル・プログラミングである．それは，エンドユーザにとって理解の難しいプログラムという抽象的な言語の世界を，視覚化を通してより理解しやすくすることを目指している．一般的なプログラミングが分析的，論理的，言語的に思考を中心に左脳主体で行われるのに対し，ビジュアル・プログラミングでは視覚を利用することで右脳も有効に活用すると Shu(1988) [53] は主張している．

ビジュアル・プログラミングの歴史は古く，多数のビジュアル・プログラミング環境が提唱されている．そして，それらに対する分類の提案とサーベイも多数行われている．Myers(1986) [54] は，ビジュアル・プログラミングの初期の分類として，ビジュアル・プログラミングか否か？　例によるプログラミング (PBE)[5]か否か？　対話かバッチか？　の 3 つの基準を提示している．また，プログラムの可視化については，データか？コードか？　また，静的か？動的か？　という基準を提示している．Shu(1988) [53] は，ビジュアル・プログラミングを「プログラム開発のプロセスにおいて，意味のある図的な表現を使用すること」と定義し，ビジュアル・プログラミングを大きく視覚的言語と視覚的環境とに分け，図 2.8 のように分類している．Chang(1987) [55] は，ビジュアル言語のチュートリアルとサーベイの中で，視覚言語を，(1) 視覚対話支援言語，(2) ビジュアル・プログラミング言語，(3) 視覚情報処理言語，(4) アイコン視覚情報処理言語，の 4 つのタイプに分類している．最後のアイコン視覚情報処理言語は Shu の分類とは異なっている．Hils(1992) は，データフロー・ビジュアル・プログラミング言語のサーベイをし，HI-VISUAL [56] [57] [58], VIVA [59], Cantata [60] [61] , LabVIEW [62] [63] [64]，Prograph [65] [66] など，15 種の言語について比較している．Howard(1994) [67] は，ビジュアル・プログラミングのコンセプトと実装を LabVIEW と Prograph を対象に調査している．Green(1996) [68] は，BASIC のサンプルコードを LabVIEW と Prograph とで記述し，認知的特質のフレームワークを使ってビジュアル・プログラミングのユーザビリティ分析を行っている．田中 (1996) [69] はビジュアル・プログラミングをサーベイし，更に並列論理型言語 GHC [70] [71] をベースとしてビジュアル・プログラミングを可能にした PP [72] を紹介している．Burnett(1994) [73] は，ビジュアル・プログラミング言語のための分類体系を提案している．Burnett の分類体系では，まず，ビジュアルな入力（視覚的情報の解析），ビジュアルな処理（ビジュアル・プログラミング言語），ビジュアルな出力（情報可視化），ビジュアルな保管（ビジュアル・データベース）の 4 つに大きく分類し，更に処理の部分を図 2.8 の

[5]Programming by Example

ように分類している．Burnett は，この分類体系を使って，IEEE Symposium on Visual Languages と，Journal of Visual Language and Computing の論文を中心に分類した文献一覧を Web 上で公開 [74] している．また Burnett(1999) は文献 [75] でもビジュアル・プログラミングをサーベイしている．Blackwell(2001) [76] は，ダイアグラムによるプログラミングの認知因子について論じている．Gostagliola(2002) [77] はビジュアル言語の設計を支援する分類フレームワークを提案している．Myers(2003) [78] は，グラフィカルなユーザ・インタフェース・プログラミングについて広くサーベイしている．Johnston(2004) [79] は，データフロー・プログラミング言語の進展について歴史からサーベイを行い，4 つのオープン問題（テキスト・データフロー言語での繰り返しの提供，DFVPL[6]の対話構造，データ構造の利用，非決定性）をあげ議論している．Boshernitsan(2004) [80] は，ビジュアル・プログラミング言語 ARK, VIPR, Prograph, Forms/3, Cube について比較を行っている．Kelleher(2005) [81] は，初心者プログラマのためのプログラミング環境と言語の分類学についてサーベイを行っている．Ryder(2005) [82] は，近代的プログラミング言語へのソフトウェア工学研究の影響を述べる中で，ビジュアル・プログラミングの研究から実用への変遷を考察している．

[6]Data Flow Visual Programming Language

(vp1) 視覚化言語
- (vp1a) 視覚情報を操作する言語
- (vp1b) 視覚的対話を支援する言語
- (vp1c) 視覚的表現を用いたプログラミング言語 (ビジュアル・プログラミング言語)
  - (vp1c-1) 図式システム (フローチャート,N-S 図式, データフロー・ダイアグラム)
  - (vp1c-2) アイコニック・システム
  - (vp1c-3) テーブル/フォームベース・システム

(vp2) 視覚化環境
- (vp2a) 視覚化
  - (vp2a-1) データとデータ構造情報
  - (vp2a-2) プログラムと実行
  - (vp2a-3) ソフトウェア設計
- (vp2b) ビジュアル・コーチング（例によるプログラミング (PBE)/デモンストレーションによるプログラミング (PBD)）

図 2.8: ビジュアル・プログラミングの分類 (Shu(1988)[53])

(VPL-I) VPL のための環境とツール
(VPL-Ⅱ) 言語の分類
(A) パラダイム
(1) 並行言語, (2) 制約型言語, (3) データフロー言語,
(4) フォームベース/スプレッドシートベース言語,
(5) 関数型言語, (6) 手続き型言語, (7) 論理型言語,
(8) マルチパラダイム言語, (9) オブジェクト指向言語,
(10) デモンストレーションによるプログラミング言語,
(11) ルールベース言語
(B) 視覚的な表現法
(1) ダイアグラム言語, (2) アイコン言語,
(3) 静止な絵の列に基づく言語
(VPL-Ⅲ) 言語の特徴
(A) 抽象化（データや手続き）
(B) 制御フロー
(C) データ型や構造
(D) ドキュメント
(E) イベント・ハンドリング
(F) 例外ハンドリング
(VPL-Ⅳ) 言語の実装法 （省略）
(VPL-V) 言語の目的.
(A) 汎用言語
(B) データベース言語
(C) 画像処理言語
(D) 科学計算可視化用言語
(E) ユーザ・インタフェース生成用言語
(F) Web アプリケーション・プログラミング言語
(G) 教育向け言語
(VPL-Ⅵ) VPL の理論 （省略）
(VPL-Ⅶ) VPL によるソフトウェア工学
(A) 再利用
(B) テスト
(C) デバッグ

図 2.9: ビジュアル・プログラミング言語ための分類体系 (Burnett(1994)[73])

2.2.2 具体的なビジュアル・プログラミング言語と環境

先のサーベイでも紹介したように多数のビジュアル・プログラミング言語および環境が提案されている．以下では，まず初期に画像処理を対象分野として提案されたビジュアル・プログラミング言語について示し，まとめて分析する．

HI-VISUAL: Ichikawa の HI-VISUAL [56] は，画像処理やオフィス環境の処理を対象としている．HI-VISUAL は，データフロー的な側面も見えるが，アイコニックなビジュアル・プログラミングを志向している．HI-VISUAL は 7 種類のアイコン・タイプ，DATA（文字，数字，画像）, DATA CLASS（データのクラス）, PRIMITIVE（組み込み関数）, PANEL（画面）, PROGRAM（ユーザ定義プログラム）, CONTROL（制御の流れ）, COMMAND（システムの処理状態)，からなる．アイコンはその下にテキスト（例えば，CONTROL アイコンの下に「IN」, COMMAND アイコンの下に「QUIT」も表示され意味が分かりやすくなっている．プログラミングは，メニューから選択したアイコンを結合していくことで行われる．HI-VISUAL は，技術の発展に応じて，特徴の少し異なる複数の版が存在する．後の HI-VISUAL [57] [58] では，アイコンの重ね合わせによる処理の実行の例示からプログラムが生成される PBD システムになっている．アイコンの重ね合わせによって継承が起こるなど，オブジェクト指向の機能も取り入れている．この版の HI-VISUAL では，データフローの特徴が見えにくくなっている．

VIVA: Tanimoto [59] は，4 つレベルからなる Liveness を定めた．

- Level 1: Informative
- Level 2: Informative + Significant
- Level 3: Informative + Significant + Responsive
- Level 4: Informative + Significant + Responsive + live

そして VIVA [7] は，その Liveness を目標としている．VIVA は，データストリームの処理結果の表示を常に更新し続けるストリーム駆動の更新を特徴とし，Liveness のレベル 4 を達成している．VIVA はデータフローのモデルとして，電子回路メタファを利用している．

Cantata/Khoros: Khoros システム [61] は，当初は 180 個以上のアルゴリズムのライブラリを保有した画像処理の研究のためのソフトウェア環境として開発された．Cantata [60] は，その Khoros システム上のデータフロー・ビジュアル・プログラミング言語であ

[7] Visualization of Vision Algorithms

る．デジタル信号処理などの応用があり，データ駆動が主であるが要求駆動も選択できるハイブリッド方式である点にも特徴がある．

これらのデータフロー・ビジュアル・プログラミング環境は，その汎用性の高さを，アイコンの拡充で対応したため，多数のアイコンが存在している．更に，それらを編集する独自の編集環境が必要とされ，結線の方法などを含めると複雑なプログラミング環境になってしまっている．そのため初心者にとっての初期実行までの学習負担が高くなっており，我々の想定する日常的にプログラミングを行わない PC 利用者を対象としたプログラミング環境としては十分とは言えない．

更に以下では，より最近の代表的なデータフロー・ビジュアル・プログラミング環境として，Pure Data [83] [84] [85]，Prograph [65] [66]，LabVIEW [62] [63] [64]，MATLAB/Simulink [86] [87] [88] の分析を示す．

Pure Data: Puckette による Pure Data [83] [84] [85] 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から始まったために，コンピュータミュージックや MIDI [8] を活用したアプリケーションに多く利用されている．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．プログラムが複雑になった場合には，サブパッチとして一部を切り出すことも可能である．ボックスには，オブジェクト，メッセージ，ナンバー，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．Pure Data は，リアルタイムな処理が可能であるという点で，音声や映像の処理に適している．

Pure Data は，音楽分野での利用実績が多く存在するが，独自の編集エディタの利用が必要な点，エディタモードとランモードでインタフェースが異なる点，4 種類のボックスを使い分ける必要がある点，などが初心者には難しいと考えられる．基本的にはプログラマを対象としたプログラミング環境である．オーディオ処理を行わない利用者には，逆に若干面倒なプログラミング環境となっている．

Prograph: Prograph [65] [66][9]は，汎用的なプログラミングを可能にするデータフロー・ビジュアル・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極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．そのための独自のプログラム編集環境を持っ

[8] Musical Instrument Digital Interface
[9] Marten は Prograph の Mac OS X 上での実装である．

ている．一般的なプログラムの可視化を目的としており，非常に複雑な構文と環境を持っている．

Prographは，複雑なプログラムを記述するプログラマを対象とするため，プログラミングの初心者には利用は難しい．

LabVIEW: LabVIEW [10] [62] [63] [64] は，計測制御用のデータフロー・ビジュアル・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[11]と呼ばれる関数に相当するモジュールとなる．VIは，ユーザ・インター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一対一に対応する端子と関数アイコン，VIをワイヤと呼ばれるデータの流れで接続することで，プログラムを作成する．マルチタスク／マルチスレッド処理を行うこともできる．LabVIEWは，計測制御ソフトウェアの作成を得意とする．波形チャート・波形グラフ，スライドバー等の計測制御に適したユーザ・インターフェースと，GPIB [12]，Serial，TCP/IP等の多数の計測器制御インタフェース，及び，解析機能を装備している．LabVIEWは，テストや計測，データ集録や制御，科学的調査，プロセス監視，工場の自動化などに利用されている．

LabVIEWも計測制御分野などのプログラマを対象としているため，日常的にプログラミングを行わないPC利用者を対象としたプログラミング環境とはいえない．

MATLAB/Simulink: MATLAB [13] [86] は，数値計算・可視化のためのプログラミング環境である．MATLABは，行列計算やベクトル演算，グラフ化や3次元表示などの豊富なライブラリを持ち，ツールボックスと呼ばれる拡張パッケージをインストールすることで各種の分野に応じた拡張ができ，科学技術計算，信号処理や通信，画像や動画の処理，制御システム，テストおよび計測など，広範な領域に適用されている．Simulink [87] [88] はMATLAB環境で動作するマルチドメインのシミュレーションとモデルベースのデザインのためのビジュアル環境である．Simulinkの主なインタフェースはグラフィカルなブロックダイアグラムツールと，カスタマイズ可能なブロックライブラリのセットである．Simulinkには，動的システムのモデル化とシミュレーションのためのグラフィカルエディタ，カスタマイズ可能なブロックライブラリなどが用意されている．SimulinkはMATLABと統合されており，モデルにMATLABアルゴリズムを組み込むことや，シミュレーション結果をMATLABにエクスポートして詳細な解析を行うことが可能である．専門分野ごとにブロックがまとめられたブロックセットが多数提供されており，例えばデジタル信

[10] Laboratory Virtual Instrumentation Engineering Workbench
[11] Virtual Instruments
[12] General Purpose Interface Bus
[13] MATrix LABoratory

号処理で頻繁に使用する Signal Processing ブロックセットを利用する場合，MATLAB の Signal Processing ツールボックスも合わせて利用することになる．類似の可視化ソフトウェアとして AVS[89] がある．

MATLAB/Simulink は専門分野の可視化シミュレーションの用途で数多く使われているが，日常的にプログラミングを行わない PC 利用者を対象としたプログラミング環境とはいえない．

その他のビジュアル・プログラミング関連技術: 岡田 [90] は，インタフェースボート用のビジュアル・プログラミングするシステムを提案し，データフローとペトリネットを用いている．しかし，ペトリネットは一般の PC 利用者には馴染みが無い．そのため日常的にプログラミングを行わない PC 利用者を対象としたプログラミング環境としては問題がある．

小川 [91] は，プログラム編集の全てと実行時ビューのカスタマイズを Drag&Drop で可能にしたビジュアル・プログラミングシステム CafePie を提案している．CafePie は重ね合わせでアクションが発生するなどの新しいユーザ・インタフェースを導入している．また，CafePie は独自の代数的仕様記述言語 CafeOBJ をベースとしている．しかし，日常的にプログラミングを行わない PC 利用者を対象としたプログラミング環境ではない．

早野は Drag&Drop でスクリプトの記述が可能なシェル [92] [93] を提案している．引数の接続を Drag&Drop 時に選択する方法に限定された提案であり，プログラミング環境全般についての提案ではない．

データフロー・ビジュアル・プログラミング環境の適用先として，映像や信号処理を扱うものが多いが，それらに対する処理中のインタラクションの研究も行われている．小林 [94] [95] は映像の即興的な加工や合成のためのデータフロービジュアル言語を提案している．しかし，本研究の対象とは異なる．

2.2.3 デモンストレーションによるプログラミング (PBD) 環境

デモンストレーションによるプログラミング (PBD) 環境は，有望な手法である．表 2.1 では Integration アプローチに含まれていた．PBD は既に多くの実用的なシステムの中に組み込まれている．

Automator: MacOS X の Automator [25] もエンドユーザ・プログラミングを指向して PBD を提供している．PBD は，一連の作業を記録し，その再生によって，何度も同じ作業を簡単に再現できる．また，Automator はデスクとトップと統合されていることから，多様なマルチメディア処理の連携が可能になっている．

Automator は，ネットワークを経由した処理の統合や，Web 上の処理部品の統合を目的としていない．また，独自の開発環境であり，記録と再演という明確なモードの切り替えの存在が初心者の負担となる点や，既存のプログラムの部分利用や改変・処理の一般化が難しいという問題点もある．

Photoshop Action and Batch: Photoshop [26] の Action 機能も，MacOS の Automator に類似する．Action 機能は Photoshop 上で行った一連の作業を記録し，その再生によって，何度も同じ作業を簡単に再現できる．また，Action 機能で登録したアクション処理をバッチ処理として特定のフォルダ以下の全ての画像に適用することも可能である．

Photoshop Action and Batch は，Automator と同様に，PBD の限界を持つ．独自の開発環境であり，記録と再演という明確なモードの切り替えの存在が初心者の負担となる点や，既存のプログラムの部分利用や改変・処理の一般化が難しいという問題点がある．

Pursuit: Andrew(2011) らによるサーベイ [96] においてエンドユーザ・プログラミングとして記載され，ファイル管理のドメインに分類されるものに Pursuit(1997) [27] が存在する．Pursuit は，ファイルとフォルダがアイコンとしてあらわされるビジュアル・シェルである．Pursuit は PBD のアプローチを採り，デモンストレーションでプログラムを作ることによってプログラミングを簡素化することを目指している．Pursuit の PBD では，プロローグとエピローグとして表示されるコミック・ストリップ・メタファを使っている点に独自性がある．

Pursuit は独自の開発環境である点や，既存の利用者からの知識経験とは大きく異なる点など，学習に対する障壁が高い．

2.2.4 テキスト・プログラムの図式化

テキストのプログラムをフローチャートのような図形表現するものもビジュアル・プログラミングの一種である．N-S 図式[14] [97] や二村らの PAD[15] [98] [99] などは，テキスト・プログラムをそのまま図的に表現する．

これらはプログラムコードの可視化を狙いとして提案されており，日常的にプログラミングを行わない PC 利用者を対象としたプログラミング環境ではない．また，プログラム開発環境の進化により，開発の現場においてこのアプローチは現在ではほとんど使われておらず，既に役割は終えていると考えられる．

一方，テキストでのプログラミングとビジュアル・プログラミングを統合し，複数のビューの同時利用を可能にするアプローチが提案されている．

[14] Nassi-Shneiderman Diagram
[15] Program Analysis Diagram

Leogo: Leogo [100] は，幼児用教育言語の Logo [101] をベースに，アイコニック・プログラミングと，デモンストレーションによるプログラミングと，テキストによるプログラミングの 3 つのプログラミング・パラダイムを同時に持ち相互に連携している．Logo のタートルが動く画面が直接操作の対象となっている．一つのパラダイムでの入力が，他の 2 つのパラダイムに即時反映され，利用者はパラダイムを選択することができる．

PP: 田中の PP [16] [72] [69] は，並列論理型言語 GHC [17][70] [71] をベースとしたビジュアル・プログラミング環境である．ビジュアルなプログラムと，テキスト記述のプログラムは，相互に結合しており，どちらからでも入力でき，それはもう一方に反映される点にもう一つの特徴がある．

これらのハイブリッドなアプローチは，ベースとなる言語（Logo や GHC）を利用・維持することで，大規模プログラムへの対応力，入力の手間の削減，幅広い分野への応用力，ポータビリティの欠如といったビジュアル・プログラミングのもつ基本課題への対応を行う現実的なアプローチである．また，ビジュアルな表現とプログラムコードが素直に対応すれば，学習の容易さを向上させる可能性もある．

しかし，Leogo と PP は，我々の想定する日常的にプログラミングを行わない PC 利用者を対象としたプログラミング環境ではない．また，複数の環境の操作方法を新たに覚えなければならないとすると，初心者にとっての障壁は，より高いものになる可能性がある．

2.3 マッシュアップ・システム

マッシュアップとは，Web 上のサービスやデータを組合せて新しいサービスを作成する行為をいう．Dornan(2007) [103] は，マッシュアップを Presentation, Data, Logic の 3 種類に分類している．Presentation のマッシュアップは，複数のコンテンツ組合せて同一の Web ページ内に表示する．iGoogle などのポータルサイトが該当する．Data のマッシュアップは，複数の情報源から得たデータを合成してサービスとして提供する．地図上に口コミの情報を重ねて表示することなどが相当する．Logic のマッシュアップは，更に高度に，複数の Web サービスの入出力を連結させる．一般にはプログラミングが必要になる [104]．本研究では，それぞれ，表現統合，データ統合，処理統合と呼ぶ．また，マッシュアップは，利用者主導での統合，小規模な統合，Web API を経由した統合という側面の特徴も持つ [105].

[16] Program-less Programming or Pictorial Programming

[17] GHC は，論理型言語 Prolog [102] にコミットオペレータを導入し，並列実行の機能をもたせた言語である．

新谷 (2009) は「知的 Web のためのマッシュアッププログラミング」[104] において，マッシュアップ・プログラミングについての事例の紹介や Web API の発見と利用について考察を行っている．以下では，代表的なマッシュアップ関連システムについて紹介する．

Yahoo! Pipes: 2007 年に発表された Yahoo! Pipes [4] [106] は，マッシュアップ分野を代表するマッシュアップ環境である．データフロー処理をベースとした Web 上のビジュアル・プログラミング環境を提供する．利用者はグラフィカルな Web ベースのエディタを使って pipe と呼ばれるマッシュアップを生成できる．利用者は pipe を公開し，他者と共有することもできる．RSS や XML, JSON などの構造化データに関するマッシュアップをターゲットとし，RSS データを取捨選択し地図データ上に配置するという処理に広く利用されている．入力データタイプの限定や Web サイトの限定などはあるが十分実用的な環境となっている．

地図の表示などはできるものの，データタイプに対する制限が強いという問題点があり，本研究が目指すマルチメディア処理の統合には向いていない．

更に，プログラミング環境としては，Yahoo! Pipes の開発環境は独自 GUI で 5 種のウインドウ（コマンドメニュー，部品選択，部品解説，部品配置，デバッグ）からなる，あくまでもプログラマ向けの開発環境である．多数の開発ウインドウを使ったプログラミングが必要で，作成・編集モードと実行モードの切り替えを必要とし，処理の修正にも通常のプログラミング能力やデバッグ能力が求められる．Yahoo! Pipes は，統合開発環境 (IDE) に慣れたプログラム開発者にとっての違和感は少ないが，日常的にプログラミングを行わない PC 利用者にとっては利用が難しい環境と言える．

この Yahoo! Pipes と類似のマッシュアップ環境はいくつか存在する．

JackBe Presto: JackBe Presto Platform [105] は，企業向けのマッシュアップ・ソリューションを提供する製品群である．マッシュアップを記述する EMML(Enterprise Mashup Markup Language) に対するマッシュアップのエンジンを提供する Presto Server，Excel・SOAP・Java などプログラミング言語への各種 API などを提供する Presto Connector，そして Mashlet と呼ばれるマッシュアップを簡単に作成するツール群を提供する Presto Mashup Components の大きく 3 つの製品群からなる．この中で Presto Mashup Components は，更に Presto Wire, Presto Mashup Studio, Presto Mashlets を製品として提供し，効率的なマッシュアップを支援する．Presto Wire は，IT ビジネスユーザに，ブラウザをベースとした Drag & Drop で視覚的にマッシュアップを構成するツールであり Yahoo! Pipes に類似する．Presto Mashup Studio は，マッシュアップの設計・テスト・デバッグ・配布の制御のためにプログラマに対して Eclipse の Plug-in を提供する．Presto Mashlets は，ポータブルでユーザ・インタフェースを共用可能なマッシュアップを作るための企業用の

Widgets である．

ビジネスユーザ向けの部品接続を容易にする製品として提供される JackBe Presto は，基本的には Yahoo! Pipes とほぼ同じであり，利用者の一定レベル以上のプログラミング・スキルを前提としている．

Mashup Center, Popfly 等: そのほか，多数の類似システムが存在する．IBM Mashup Center [107] も Yahoo! Pipes や JackBe Presto 系のビジュアル・プログラミング環境である．Microsoft Popfly [108] [109] は，Microsoft Silverlight [110] と統合されたマッシュアップ環境である．3D の Block を接続したデータフローのビジュアル・プログラミングが可能である．Google Mashup Editor [111] は，は HTML のタグをベースにテキストエディタでマッシュアップ開発を支援する環境である．Intel MashMaker [112] はクライアントサイドで Web 画面をマッシュアップする環境である．Grammel [113] は Yahoo! Pipes, IBM Mashup Center , Microsoft Popfly, Google Mashup Editor, Intel MashMaker 等について 6 つの観点（抽象化のレベル，学習支援，コミュニティの特徴，発見性，ユーザ・インタフェース構築，ソフトウェア工学技法）で比較を行っている[18] ．

これらは，Yahoo! Pipes を本格的なプログラミング環境や多様なプログラミング環境に発展させる位置づけにあり，より高度なプログラマ向けの本格的なプログラミング環境をサポートするが，日常的にプログラミングを行わない PC 利用者にも使える環境は提供していない．

MIRO: MIRO[114] はマルチメディア処理をマッシュアップする作業環境で，ユーザフレンドリーなプログラミング環境を提供する．MIRO は，Microsoft Popfly や Intel MashMaker などに比べてより簡単に利用でき，Yahoo! Pipes に比べて RSS 以外の画像・映像・音楽なども扱えるとしている．

MIRO は，Yahoo! Pipes にマルチメディア加工対応の機能を統合しているが，開発者向けのインタフェースと利用者向けのインタフェースは分離され，独自の環境となっている点を始め，基本的には Yahoo! Pipes と同様のプログラミング能力やデバッグ能力を利用者に要求する．このため MIRO は初心者に要求する知識量が多く，日常的にプログラミングを行わない PC 利用者にとって利用は難しい環境と言える．

[18] これらのうち Microsoft Popfly と Google Mashup Editor については既に提供が中止されている．

2.4 ファイルシステム

本節では，本研究のアプローチに密接に関連するファイルシステムやその拡張についての動向を示す.

2.4.1 Web フォルダ (WebDAV)

Web 上のフォルダとして WebDAV [19] [20] が存在する．WebDAV は RFC4918 （当初は RFC2518）によって規定された，HTTP 拡張によって Web フォルダ・システムを実現するプロトコルであり，HTTP1.0 [17] , HTTP1.1 [18] と合わせて以下のような機能を提供する.

HTTP-PUT: リソースの作成

HTTP-DELETE: リソース／コレクションの削除

HTTP-MKCOL: コレクションの作成

ここでリソースがファイル，コレクションがフォルダに相当する．HTTP-PUT はリソースを生成する，HTTP-DELETE はリソース／コレクションを削除する，HTTP-MKCOL はコレクションを生成する．Web フォルダにおいてデスクトップ上でファイルをフォルダへ Drag&Drop する操作は，Web フォルダではファイルの送信 (HTTP-PUT) となる．既に WebDAV は広く普及しており，サーバとしては Apache の HTTP server [115]，クライアントとしては Microsoft Windows の Explorer [116] や Internet Explorer, MacOS X の Finder [117] などがサポートしている．WebDAV はネットワーク・フォルダの CIFS [118] [119] （もしくは Samba [120] ）やユーザ拡張可能なファイルシステム FUSE [19] [121] などに比べ，HTTP 上のプロトコルである点に優位性がある.

HTTP/WebDAV では，処理の実行は HTTP-GET や HTTP-POST を経由した CGI [20] は用意されているが，複数のサービスを組合せて実行する仕組みは用意されていない.

本研究では，この WebDAV の HTTP-PUT の契機をハンドリングしてフォルダ・プログラミング処理系を起動し，処理連鎖を実現することで Web サービスのマッシュアップ環境に発展させている.

[19] File system in User space
[20] Common Gateway Interface

2.4.2 フォルダ・アクション系

MacOS X Folder Action: MacOS X [122] のフォルダ・アクション [123] 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に対して AppleScript [124] [125] で記述された処理を登録できる仕組みである．イベントには，フォルダを開いた時，フォルダを閉じた時，フォルダを移動した時，フォルダに項目を追加した時，フォルダから項目を削除した時，の 5 つが存在する．MacOS X の Automator [126] で作成した処理をフォルダ・アクションに登録することも可能である．

フォルダ・アクションは単に 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留まっており，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ことや Web サービスのマッシュアップを行う環境ではない．

Dropbox Automator: Dropbox Automator [127] はネットワークフォルダである Dropbox [128] の拡張サービスである．選択したフォルダに事前にアクションをルールとして設定しておくことで，フォルダにファイルを Drag&Drop した際に，そのルールが実行され，実行結果を得ることができる．(処理前のファイルは processed フォルダ内に移動する)．アクションは拡張子に応じて規定のものから選択可能で，Documents に対しては"Convert to PDF", "PDF to TXT", "Upload to Googole Docs" などの処理を，Pictures に対しては，"Upload to Facebook", "Convert image", "Write text on image" などの処理を，Any file に対しては，"e-mail", "Zip file", "Rename" などの処理を，選択することができる．Dropbox 上のファイルと，Web 上のサービスを結びつけることが可能であり，今後の発展が期待されるサービスである．

Dropbox Automator は処理の組合せ制御はできない点や，同じファイルタイプへのルールを複数設定することはできない点などの制約が強くプログラミングを行うことはできない．

Photoshop Droplet: Photoshop には Droplet [129] という機能も提供されている．Photoshop のアクションを Droplet という実行ファイル化することで，利用者はその実行ファイルのアイコンにファイルやフォルダを 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 Photoshop アクションを一斉に適用してくれるものである．簡単に言えばショートカットである．Droplet は MacOS X のフォルダ・アクション機能に類似する仕組みである．

Photoshop Droplet は，処理連鎖など，複数の Droplet を組合せてプログラミングできるものではない．単純な逐次処理しかできないため，例えば条件に応じた処理の変更などの処理にも対応できない．

2.5 関連技術のまとめ

これまで述べた主な関連技術について，本研究の要件への適合状況を図 2.10 に示す．このように従来の関連技術で，本研究の 2 つの要件を同時に満足するものは知られていない．

技術分類		代表的システム	【要件1】エンドユーザプログラミング環境	【要件2】マルチメディアマッシュアップ環境
エンドユーザ・プログラミング	会計パラダイム	Excel	○	×
ビジュアル・プログラミング	PBD	Automator (MacOS)	×（一般化NG，分岐処理NG）	○
	データフロービジュアル	PureData	×（高スキル必要）	○
マッシュアップ・システム		Yahoo! Pipes	△（スキル必要）	×（RSS）
ファイル・システム拡張		FolderAction (MacOS)	—	—

図 2.10: 主な関連技術の状況

第3章　フォルダ・プログラミング環境の提案

3.1　はじめに

本研究では，エンドユーザ向けのプログラミング環境として，フォルダ・プログラミング環境 (POLDER) [130] を提案している．フォルダ・プログラミング環境は，ファイルシステムの GUI として広く普及しているフォルダに着目し，それをプログラミング環境として拡張するアプローチを採っている．

本章では，本研究で提案するフォルダ・プログラミング環境を定義する．まず本研究のアプローチを概説し，フォルダ・プログラミング環境の重要な構成要素である処理付フォルダを 3 つの特徴的な仕組みから定義し，それを組合せてプログラム化するためのフォルダ・プログラミング環境について，主要な 6 つの実行制御機能によって定義する．更に，フォルダ・プログラミング環境を Web フォルダの拡張として実装するためのソフトウェア・アーキテクチャを示し，プロトタイプの実装の詳細を述べる．最後に，処理付フォルダの部品例やデータ分類への簡単な応用例を示すとともに，一杉のエンドユーザ向けスクリプトの設計目標の観点を使ってフォルダ・プログラミング環境がエンドユーザに提供する効果について考察する．

3.2　フォルダ拡張のアプローチ

PC 利用者の入り口となるユーザ・インタフェースとしてデスクトップ環境が広く使われている．フォルダはその中の重要な構成要素である．デスクトップ及びフォルダは 1981 年 Xerox Star ワークステーション [131] [132] に最初に搭載されて以来，MacOS や Microsoft Windows などにも受け継がれている．

デスクトップ環境の利用者はデータファイル・アイコンのダブルクリックにより，そのファイルタイプに対応した特定のアプリケーション起動することができる．Microsoft Windows などのデスクトップ環境では，各種メディアのファイルタイプに対応した Viewer, Player, Editor といったアプリケーションを備えており，プログラミング・スキルのない利用者であってもマルチメディアデータのファイルを簡単に取り扱うことができる．

データファイルのダブルクリック操作以外にも，デスクトップ環境上では，特定のアプリケーションにリンクされたショートカット・アイコン上へのデータファイルの Drag&Drop 操作で，それらのファイルを処理対象としてそのアプリケーションの起動をすることもで

きる．しかし，ショートカット・アイコンはフォルダのような階層構造をもつことはできず，処理を組合せるなどの高度な利用に発展することはできなかった．

本研究で提案しているフォルダ・プログラミング環境は，データ保管に利用している普通のフォルダを処理付フォルダに拡張し，処理付フォルダの入れ子構造を使った処理フローの構成でプログラミングを行う．これをフォルダ・プログラミングと呼び，そのフォルダ階層をフォルダ・プログラムと呼ぶ．フォルダ・プログラミング環境は，フォルダ・プログラムの開発環境であるとともに，フォルダ・プログラムの実行環境でもある．デスクトップに統合されたこのプログラミング環境は，画像，映像，音楽，音声，テキスト，構造化テキスト，オフィス文書といった多様な型のメディアデータを平易に取り扱うことができる．フォルダという広く普及したコンセプトに基づくプログラミング環境であるため，多くの利用者が，既存のフォルダ操作の知識と経験をベースとして，このプログラミング環境を学び利用することができる．

フォルダ・プログラミング環境の構成要素である処理付フォルダは，普通のフォルダと比較して，その仕組みに 3 つの特徴的な差異がある．なお，処理付フォルダの起動は Drag&Drop の Drop 操作後に起動するため，本論文では単に DROP と記載する．

(ff1) フォルダへのデータファイル DROP で処理が起動し，フォルダ名をもとに処理が動的に決定し実行され，結果がフォルダ内に入る仕組み

(ff2) 処理の引数をフォルダ名に含む仕組み

(ff3) 複数ファイルの DROP に対して，一斉に実行される仕組み

更に，フォルダ・プログラミング環境は 6 つの主要な実行制御機能をもつ．

(fp1) 処理が連鎖する実行制御

(fp2) 並行の実行制御

(fp3) 条件分岐と else の実行制御

(fp4) 制御移行の実行制御

(fp5) サブルーチン（抽象化と収集）の実行制御

(fp6) データフォルダと処理付フォルダの混在に対する実行制御

3.3 処理付フォルダの定義

処理付フォルダを定義する特徴的な 3 つの仕組みを示す.

(ff1) フォルダへのデータファイル DROP で処理が起動し，フォルダ名をもとに処理が動的に決定し実行され，結果がフォルダ内に入る仕組み

Microsoft 社の Windows デスクトップなどの GUI 環境における操作では，データファイルを Drag し，特定のフォルダ上に Drop した場合，そのデータファイルをそのフォルダの階層内に格納される．一方，処理付フォルダでは，フォルダへの DROP 操作に応じてフォルダ名に対応した処理が実行される．

仕組み：

- 予め，各処理名に対する処理実体をフォルダ・プログラム処理系内で指定しておく．処理付フォルダではフォルダ名の一部が処理名を表す.
- その処理付フォルダにデータファイルが DROP された際，そのデータファイルは，フォルダ名から動的に決定された処理名に対応する処理実体に渡され，処理が実行される．
- 処理結果がその処理付フォルダ内に格納される．処理元のデータファイルは削除される．
- 処理付フォルダでは，1 ファイルの入力に対して複数の処理結果が得られても構わないし，処理結果が 0 個であっても構わない.
- 処理結果が複数生成される場合のファイル名の重複排除は，各処理実体に任される．

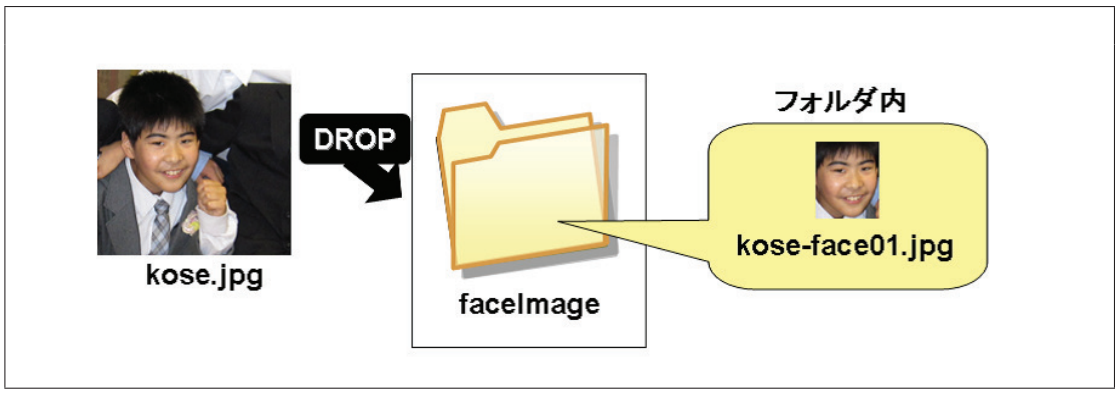


図 3.1: フォルダへのデータファイル DROP で処理が起動する仕組み

例： 図 3.1 にその例を示す．予め，faceImage というフォルダ名（＝処理名）に，入力画像内から顔画像を抽出する処理/opt/bin/extractFaces.pl が処理実体として指定してあったとする．次に faceImage という名称をもつ処理付フォルダをフォルダ・プログラミング環境内の任意の場所に作成する．そして例えば顔画像ファイル kose.jpg をその faceImage フォルダに DROP すると，kose.jpg に/opt/bin/extractFaces.pl が適用され，得られた処理結果 kose-face01.jpg が faceImage フォルダ内に生成される．同時に入力ファイルであった kose.jpg ファイルは削除される．

これが処理付フォルダの基本機能である．その処理対象データは原則として当該のフォルダの外部から与えられ，処理結果は，当該フォルダ内部に生成される．また，入力ファイルは処理結果ファイルの生成の契機で削除される．先の例で言えば，入力ファイル「kose.jpg」が結果ファイル「kose-face01.jpg」の生成によって削除されている．これは，フォルダ・プログラミングが UNIX Bash のパイプやデータフロー・プログラミングを意識していることと，フォルダ階層中に多数の中間処理結果を残さないためである．入力ファイルを消さない方法は，別の場所にコピーを作っておく方法の他にも，Microsoft Windows であれば Ctrl キーを押しながら Drag&Drop する方法や，後にフォルダ・プログラムのデバッグ法で紹介する並行フォルダを作る方法が存在する．

(ff2) 処理の引数をフォルダ名に含む仕組み

仕組み：

- 処理付フォルダでは，フォルダ名の最初に出現する区切り文字（通常半角空白）より前が処理名として処理実体の指定に使われる．
- フォルダ名の残りの部分は処理に対する実引数として処理実体に渡される．
- 引数の形式と解釈は，各処理実体に任される．これは，UNIX 系のコマンドと同様のポリシーによる．

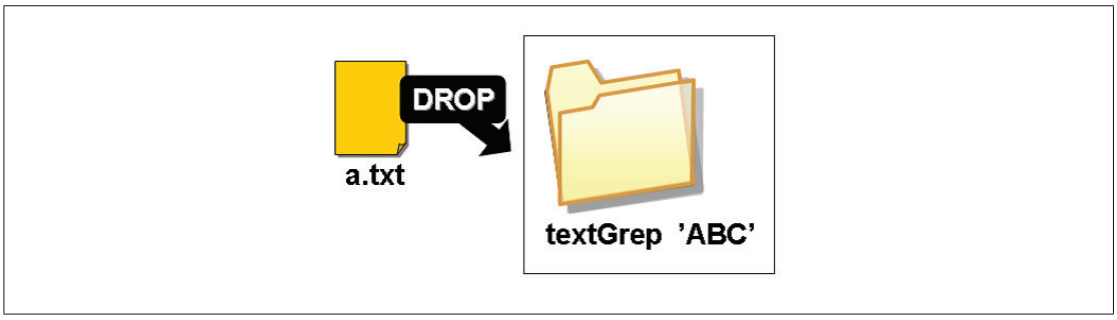


図 3.2: 処理の引数をフォルダ名に含む仕組み

簡単な例:　図 3.2 に簡単な例を示す.「textGrep」という処理名には，処理実体としてテキストファイルから実引数で指定される特定文字列を含む行を抽出する処理「/opt/polder/bin/mygrep.sh」が予め指定されているとする．このとき，処理付フォルダ名「textGrep 'ABC'」は，実引数を 'ABC' とし，処理実体「/opt/polder/bin/mygrep.sh」を実行する処理となり，この処理付フォルダに Drop されたデータファイルに適用された処理結果はフォルダ「textGrep 'ABC'」内に新たなテキストファイルとして生成される．

このようにフォルダ名の一部として空白文字を積極的に利用することになる．本論文上の表現として，フォルダ名内に空白文字を含むことを明示する場合には，フォルダ名を「」で囲んだ表記としている．

複雑な例： フォルダに投入されたデータに応じて処理を行うために，実引数は実行時に評価することもできる．評価対象の特定はUNIXのシェルであるBash [133] やプログラミング言語Perl [134] に準じて，バッククオート（`）で囲まれた区間の文字列とする．以下では，引数の評価の中でファイル内のXMLタグに割当てられた値を参照するコマンドVを利用した例を説明する．

なお，引数内のパス表現としてCIFS [118] のファイルやフォルダの禁止文字を避けるため，以下の変換ルールに従った修正パス表現を利用する．

* → # ， / → % ， ″→ ´

例えば，画像処理ライブラリImage Magick [1] [135] の画像フォーマット変換コマンドconvertが処理実体及び処理名として定義してあるとき，別のXMLファイル内のWidthタグ及びHeightタグのもつ値を実引数域に展開して-resizeオプションを実行させる場合には図3.3に示すフォルダ名となる．

```
convert -resize `V ..%data%A.xml%Width` x `V ..% data%B.xml%Heigth`
```

図 3.3: 複雑な引数を含むフォルダ名の例

このVコマンドのパス解釈では処理付フォルダが置かれているディレクトリをカレントとしている．最初のVコマンドでは「../data/」（つまり1段上のdataフォルダ内）の位置にあるA.xml内のWidthタグの値（例えば100）を取得しその引数内文字列位置に展開する．2番目のVコマンドは「../data/」の位置にあるB.xml内のHeightタグの値（例えば200）を取得しその引数文字列内位置に展開する．各Vコマンドの評価後の引数は「-resize 100x200」となり，フォルダ名は「convert -resize 100x200」と解釈されて投入データファイルに対して処理（この例では画像サイズを横100pixel ×縦200pixelに変換すること）が行われる．Vコマンドでは，ディレクトリの階層とタグの階層を同じ区切り文字（%）で表現している．また，投入ファイル自体を参照する場合にはパス表現の先頭にワイルドカード文字（#）を，投入フォルダ自体を参照する場合には変数（$0）を用いることができる．

引数の形式は，各処理実体に任されている．これは，UNIX系のコマンドと同様のポリシーである．引数と形式は，例えばBashのgetopts/getoptの形式（コマンド名 -v -c -f "xxx"）や，そのロングオプション形式（コマンド名 –verbose –count –file "xxx"）など，様々な形式が存在する．フォルダ・プログラミング環境では，引数の形式は統一しない．

後の実装例で紹介するような処理付フォルダを作成を支援するテンプレートでは，多数の処理引数を指定する方法として，処理付フォルダ内に「conf/parameter.conf」ファイルを作成し，その中に，key=value 形式で記述しておくこともサポートしている．同じ key 値の場合には，処理付フォルダ名で指定した引数が優先される．この仕組みは，3.4 節 (fp6) で述べる処理付フォルダ内にデータフォルダが混在できる特性を利用したものである．処理付フォルダ内のデータファイル内に引数を記述し保存できることで，利用者による処理付フォルダの複製・移動の際にも，引数がともに複製・移動される．

(ff3) 複数ファイルの DROP に対して，一斉に実行される仕組み

デスクトップにおいて，複数のファイルを一括選択し，1 回の Drag&Drop 操作でフォルダに移動することはよく行われる．同様に，処理付フォルダに対しても，1 回の Drag&Drop 操作で複数のファイルを投入し処理させることができる．

仕組み：

- 処理付フォルダに入力するデータファイルが複数ある場合には各々の入力データファイル毎に処理が起動される．
- 各々の処理の実行順序は保証されない．

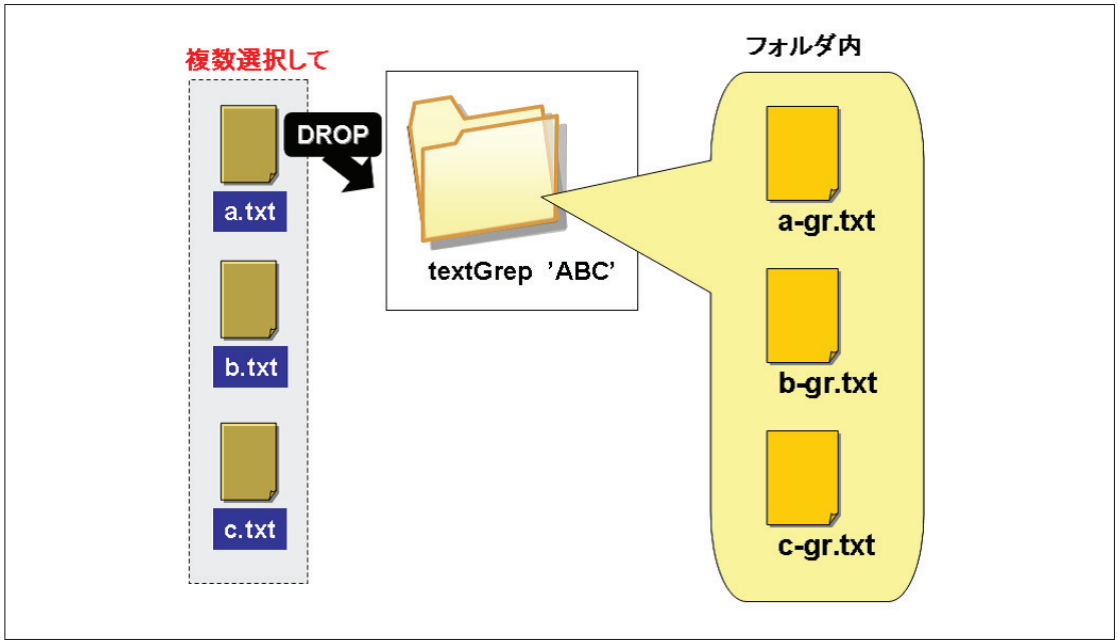


図 3.4: 複数ファイルの DROP に対して一斉に実行される仕組み

例： 図 3.4 にその例を示す.「a.txt」,「b.txt」,「c.txt」という 3 つのデータファイルを選択し,「textGrep 'ABC'」フォルダに一度に Drag&Drop した場合，各々のデータファイルに対して「/opt/polder/bin/mygrep.sh 'ABC'」が行われ，その処理結果「a-gr.txt」,「b-gr.txt」,「c-gr.txt」が処理付フォルダ内に生成される．処理実体 mygrep.sh では，出力結果ファイル名の作成と「/bin/grep」への入力ファイルの適用を行っている．

3.4 フォルダ・プログラムの実行制御機能の定義

複数の処理付フォルダを組合せて新たな処理フローを構成することを本論文ではフォルダ・プログラミング，また，その処理フロー自身をフォルダ・プログラムと呼ぶ．以下では，フォルダ・プログラム実行制御機能，フォルダ・プログラム開発支援機能を説明する．フォルダ・プログラム実行制御機能は次の 6 つの主要な実行制御機能で定義できる．

(fp1) 処理が連鎖する実行制御

フォルダ・プログラミングの基本は，処理付フォルダの入れ子構造を構成し，以下に示す処理付フォルダの処理連鎖によって処理フローを構成することである．これは，一種のデータフロー・プログラミングとも言える．UNIX Bash のパイプもコマンドの出力を次のコマンドに接続するデータフロー・プログラミングの一種であるが，フォルダ・プログラムの処理の連鎖も，それを素直にフォルダ構造にマッピングしたものと考えることもできる．

仕組み：

- 処理付フォルダの中に，更に処理付フォルダが存在する場合には，上位の処理結果ファイルは下位フォルダへの投入ファイルとなり，処理が連続して実行される．
- 投入連鎖途中の処理結果ファイルは，下位フォルダによる処理結果ファイルの生成の契機で削除される．

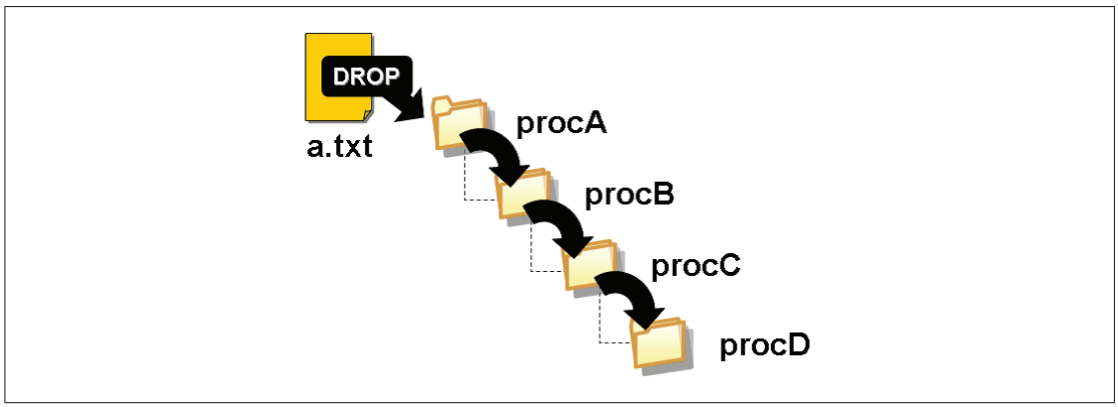


図 3.5: 処理が連鎖する実行制御

例： 図 3.5 に例を示す．a.txt は procA の処理を実行され，その結果は procB の処理を実行され，…，と続き，最下層の procD フォルダの中に最終的な処理結果が格納されて連鎖の処理は停止する．なお，各層で処理付フォルダの投入対象となった中間処理結果ファイルは削除される．つまり，フォルダ procA, procB, procC の中にはデータは残らず，procD の中にデータ（最終処理結果データ）が残る．入力データの a.txt は，最初の procA に DROP した際に削除される．[1]

(fp2) 並行の実行制御

フォルダ内には，複数のフォルダを格納することができる．この場合には並行処理[2]となる．

仕組み：

- 処理の連鎖において，下位に複数の処理付フォルダが存在した場合には，上位の処理付フォルダの処理結果ファイルはそれら複数の処理付フォルダへの並行処理の投入となる．
- その際，投入対象の処理結果ファイルは，複製されて全ての投入対象フォルダに投入される．
- 各並行処理の処理順序に規定はない[3]．

[1] Windows のデスクトップの機能では，Control キーを押したまま a.txt をフォルダに Drag&Drop することで，複製されたデータファイルを当該フォルダに投入することになる．その場合には，元の a.txt は残ったままとなる．

[2] 本論文で，並行処理とは論理的な同時実行処理を言う．また並列処理とは物理的な同時実行処理を言う．

[3] なお，後に述べる else では順序性が発生する．

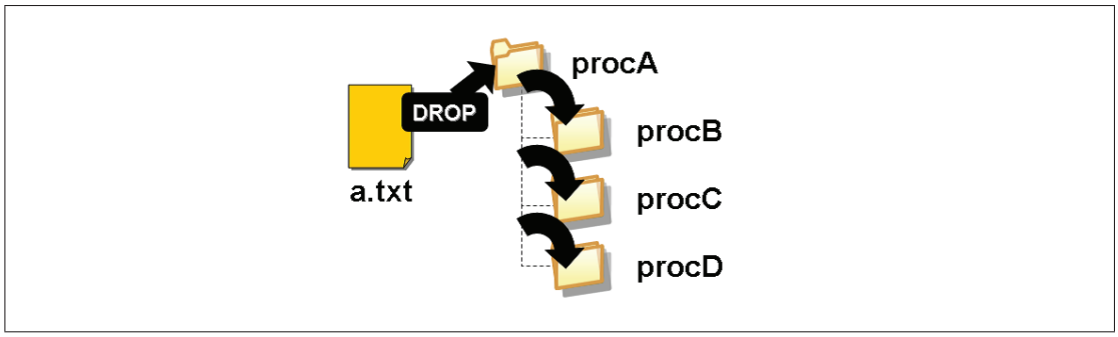


図 3.6: 並行の実行制御

例： 図 3.6 に例を示す．procA の処理結果ファイルは，procB, procC, procD の 3 つの処理付フォルダに投入され，処理結果は 3 つに分かれる．つまり，フォルダ procA の中にはデータは残らず，procB, procC, procD の中に異なるデータ（最終処理結果データ）が残る．入力データの a.txt は，最初の procA に DROP した際に無くなる．

この応用例として，実質的に何も行わない処理付フォルダ copy を並行する階層に作成することでその中に処理前のファイルのコピーを保存することもできる．これは後述のデバッグの手法として活用することができる．

(fp3) 条件分岐と else の実行制御

条件分岐は，並行実行において処理付フォルダが条件を持つケースとして実現される．

仕組み：

- 処理付フォルダを構成する処理実体は,「成功」,「失敗」の他に,「照合なし」の戻り値を持つことができる．
- 入力データファイルに対して同一階層に存在する else フォルダを除く全ての処理付フォルダが「照合なし」を返す場合に else フォルダへの連鎖が発生する．
- 並行実行には処理の順序関係が存在しないが，それらと else の間には順序関係が発生する．

else フォルダは同階層のいずれの処理付フォルダにも投入されなかったデータファイルが，処理連鎖で投入される特殊な処理付フォルダである．フォルダ・プログラムの条件分

岐を通常のプログラミング言語のように利用したい場合には，各条件を排他的に記述しなければならない．

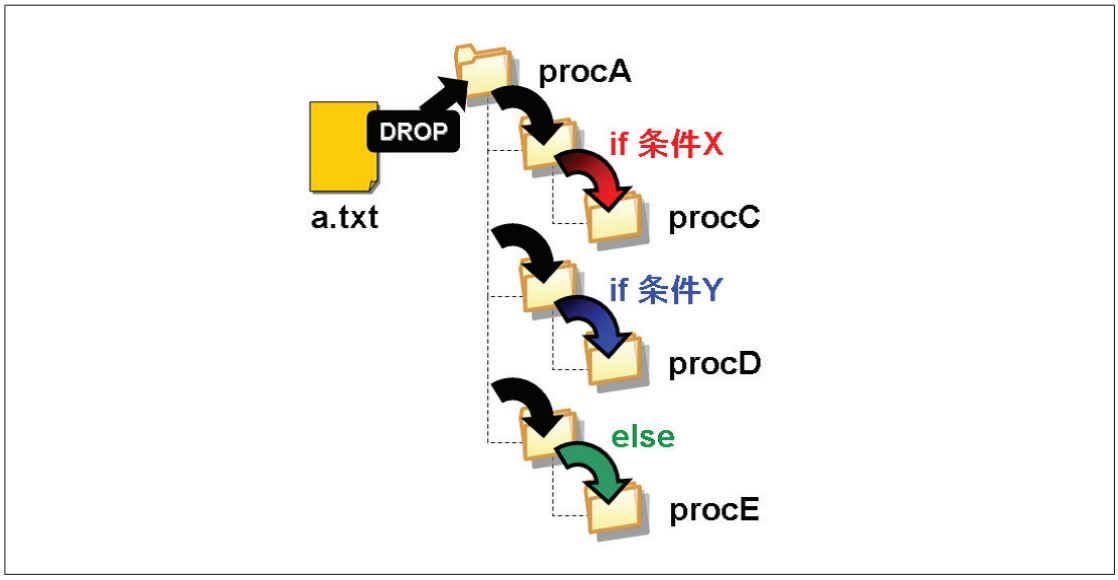


図 3.7: 条件分岐の実行制御

例： 処理付フォルダ if は条件を引数としてもち，条件を評価し満足する場合にのみ投入データを下側の階層に入力データとして渡す処理実体が指定されている．1 つのフォルダ階層に引数の異なる複数の if フォルダが存在する場合，各々の if フォルダは並行に実行される．

図 3.7 において procA の処理結果ファイルは,「if 条件 X」，「if 条件 Y」の各々への投入対象となる．処理結果ファイルが条件 X を満足すれば処理結果は procC への投入対象となる．その結果とは独立に，処理結果ファイルが条件 Y を満足すれば処理結果は procD への投入対象となる．また，条件 X も条件 Y も満足しない場合には，特別なフォルダである else フォルダの投入対象となり，処理連鎖を経由して procE への投入対象となる．条件分岐の例として，先に紹介した textGrep がある.「textGrep 'ABC'」のフォルダと同じ階層に else フォルダを作成した場合，else フォルダには，文字列"ABC"を含まないファイルが格納されることになる．

(fp4) 制御移行の実行制御

仕組み：

- 処理付フォルダ link は，引数でパス指定された他のフォルダに対してデータを投入する．そのフォルダが処理付フォルダの場合には，処理の連鎖が継続する．
- 引数の指定がパスでは無い場合にはラベルと解釈し，同一ラベルを持つ処理付フォルダにデータを投入する[4]．

link は通常のプログラミング言語の goto 文に相当する．link と if によって原始的な繰り返し処理を実現することが可能になる．この機能は，フォルダ・プログラムの計算可能性の議論をシンプルにするために導入された．

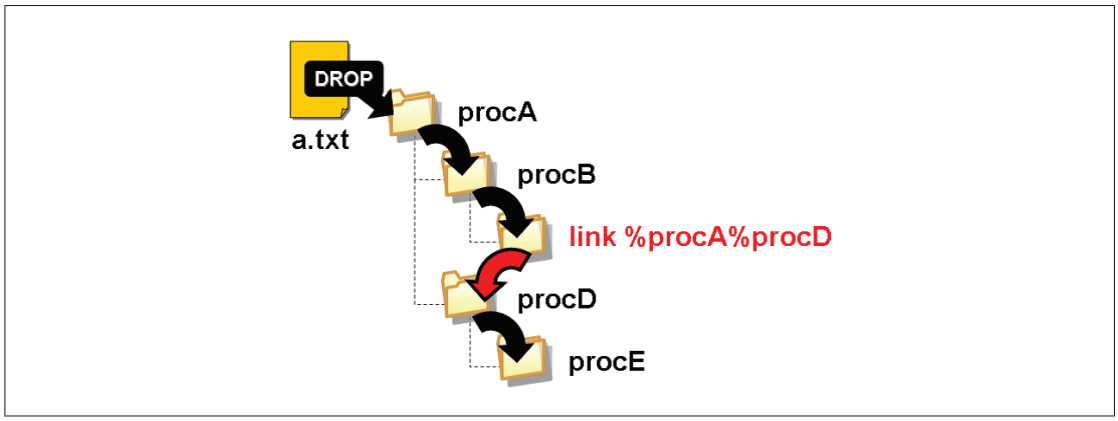


図 3.8: 制御移行 (リンク) の実行制御

例： 図 3.8 に示す例では，procA, procB と処理された後，link の引数で指定された相対パスによって，procD,procE と処理が実行される．link の引数は利用者のホームディレクトリを root とする絶対パスであっても構わない．

[4] この仕組みは，後に述べる poldef と call でも利用される．

(fp5) サブルーチン（抽象化と収集）の実行制御

並行の処理フローを利用すると，それらの処理結果は複数のフォルダに分散されて蓄積される．しかし，処理結果を参照する場合に分散しているアクセスに手間がかかる．そこでこれらの結果を 1 つに纏め上げる処理として，特殊な収集フォルダ poldef がある．また，この仕組みを利用する call フォルダによってサブルーチンを実現することも可能になる．

仕組み：

- poldef フォルダは，それら下位に生成される処理結果ファイルを全て poldef フォルダ直下に移動させ収集する．
- poldef フォルダは引数でラベルを持ち，ラベルのパスは処理系で管理する．
- call フォルダは link フォルダと同様の仕組みで，入力データファイルを対応するラベルをもつ poldef フォルダに投入し，その処理結果を poldef フォルダから自身のフォルダに戻す．call フォルダ内に処理付フォルダがあれば処理連鎖が継続する．

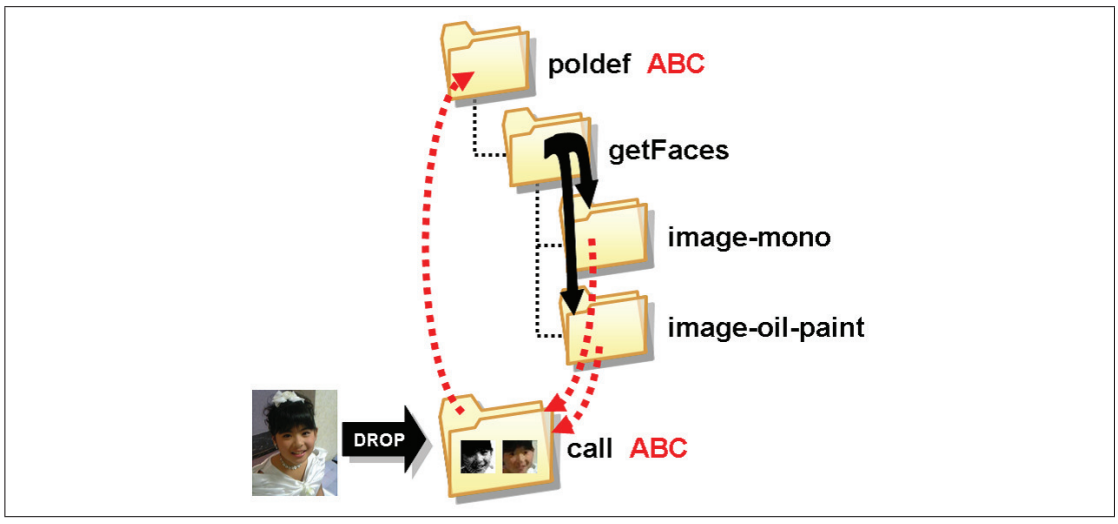


図 3.9: サブルーチンの実行制御

例： 図 3.9 は，その様子を示している．call フォルダに投入されたファイルは call フォルダの引数部分のラベル（ここでは"ABC"）に対応する poldef フォルダに送られ，poldef 以下のフォルダ構造に応じて処理が実行され，一度，poldef フォルダ直下に処理結果ファ

イルが収集された後，その全ての結果を呼び出し元の処理付フォルダ call 直下に移動される．これにより poldef フォルダ配下の処理をサブルーチン化として利用できる．

(fp6) データフォルダと処理付フォルダの混在に対する実行制御

処理付フォルダ内には，通常のデータフォルダが混在できる．同一階層に処理付フォルダとデータフォルダが混在しても構わない．

仕組み：

- 処理名が定義されていないフォルダは，通常のデータフォルダとして扱われ，処理の連鎖は発生しない．

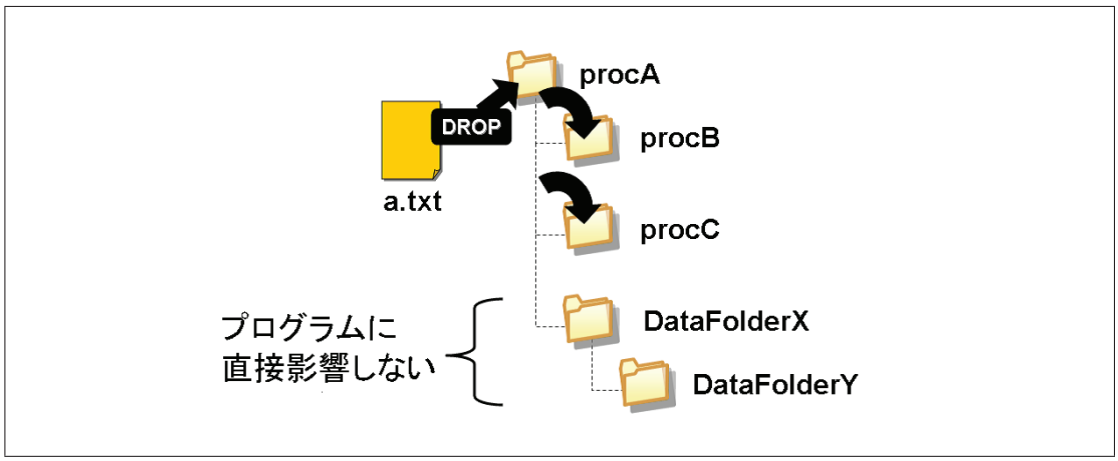


図 3.10: データフォルダと処理付フォルダの混在に対する実行制御

例： 図 3.10 において，procA の中には，処理付フォルダ procB, procC とデータフォルダ DataFolderX が存在する．procA の処理結果が連鎖して下位フォルダに渡される際，procB 及び procC は処理実体があるので処理が連鎖するが，DataFolderX については処理実体が未定義であるため投入連鎖は発生せず処理は連鎖しない．また，DataFolderX 内にあるデータファイルは直接の処理対象にはならない．この仕組みにより，処理付フォルダの引数を格納するケース（3.3 節 (ff2)）や，デバッグ用のテストデータを保管しておくケース，複数のデータの受付を待って処理が実行される処理付フォルダを実現する際に一時的なデータ保管域として使うケースなどの活用シーンが存在する．

3.5　フォルダ・プログラム開発支援機能

(fs1) ERROR 時の入力データ

仕組み：

- 図 3.11 に示すように，特定の処理付フォルダを中心として見た場合，入力データファイルは処理付フォルダの外側にあり，処理後のデータファイルは処理付フォルダ内に生成される．
- 処理付フォルダの処理が失敗したら，データファイルは改名されてそのフォルダの外側に配置される．そのファイル名は時刻やプロセス識別子等のシステム ID を含むユニークな名前になる．

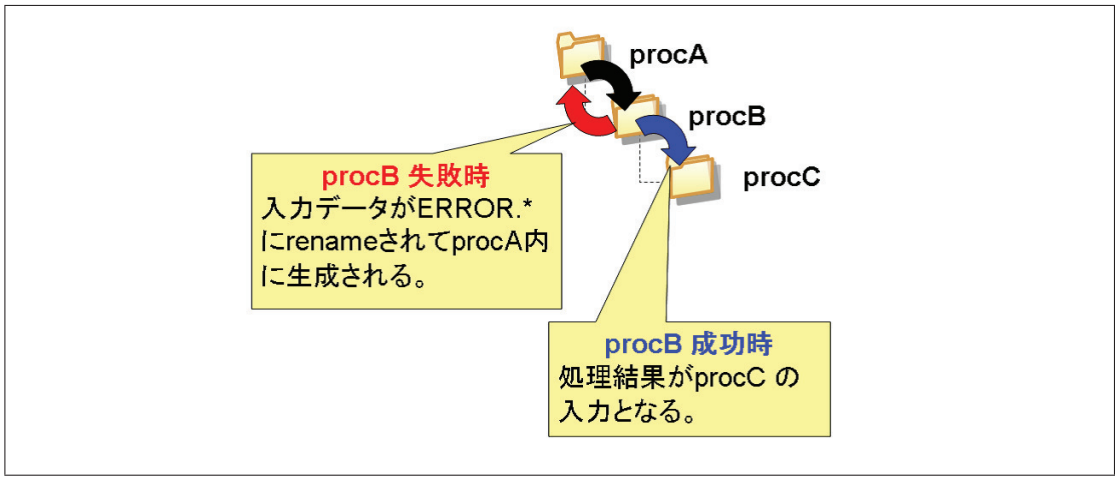


図 3.11: DEBUG の支援と ERROR の制御

(fs2) デバッグ方法

フォルダ・プログラミング環境は，デスクトップ上の Viewer,Player,Editor アプリケーションと統合されているため，入出力データの確認が容易にできる．それに加えて，フォルダ・プログラミング環境では，各処理単位へのデータ投入がどの階層にでも可能である．従来のフォルダ管理の枠内で，少し工夫するだけで，以下のような処理の部分実行が容易に実現できる．

単独の実行： フォルダ・プログラムの要素となる処理付フォルダは，単独のフォルダとして作成することで単独で処理を実行することが可能である．これにより，個々の処理の問題を確認できる．

下位層の実行： フォルダ・プログラミング環境では，フォルダ階層中のどの階層にでもデータファイルの投入が可能である．よって，フォルダ・プログラムの途中から処理を実行して確認することが容易である．

上位層の実行： フォルダ・プログラムのある階層のフォルダまでを実行したければ，それより下のフォルダを処理未定義のフォルダ名に改名（例えば，先頭に'XXX' を付ければ良い）して実行すれば，そのフォルダまでの実行結果を得ることができる．

部分階層の実行： 前述の，上位層実行，下位層実行を使えば，任意の部分階層を実行し，また，再開するなどのデバッグが可能になる．また，より単純には，フォルダ階層の部分階層（フォルダ・プログラムの一部分）を別のエリアに複製して，そこで処理を実行させて確認してもかまわない．

処理途中データファイルの参照： フォルダ・プログラムの処理では，処理の連鎖の成功に伴って，連鎖途中のデータファイルは消えてしまう．特定の処理の結果を保管しておきたい場合には，その下位フォルダとして copy フォルダを作成しておけば，その時点の処理結果（の複製）が copy フォルダ内に保存される．また，必要に応じてその保存されたデータファイルを Drag&Drop することで，下位層の処理を実行させることもできる．

これらの部分実行容易性や，その入出力の可視容易性によって，簡便なデバッグを支援する．

(fs3) HELP 情報の参照

処理付フォルダに割り当てられた処理に関する HELP 情報（具体的な使用法など）が必要な場合には，以下の 2 通りの方法によって HELP 情報（HTML-処理名.html）当該処理名フォルダ内に得ることができる．

HELP 方式 1： 処理付フォルダ名に対する実引数として HELP を指定したフォルダに，任意のファイルを Drag&Drop する．

HELP 方式 2：「HELP」（拡張子なし）というファイル名の任意のファイルを処理名フォルダに Drag&Drop する．

(fs4) フォルダ・プログラムのテキスト表現

フォルダの階層構造はインデント（字下げ）を利用することで，テキストで表現することが可能になる．また，フォルダ・プログラムにおいて入力データファイルの投入フォルダが処理の開始点として重要であるため，それを明示したい場合には，その開始点となるフォルダ名の右側に開始点マーク（‹）を記述する．このテキスト表現の存在により，フォルダ・プログラムの GUI 上でのフォルダ表現とテキスト表現を相互に変換することが可能になる．これによりプログラムを他者と共用したり，プログラム言語を学習したりすることを容易にする．これは，田中によるビジュアル・プログラミング言語 PP [72] の考え方と同じである．3.7 節などで例を示す．

3.6 フォルダ・プログラミング環境の設計と実装

本節では，提案したフォルダ・プログラミング環境 POLDER の設計と実装について示す．

3.6.1 Web フォルダ拡張アーキテクチャ

フォルダ・プログラミング環境を Web 上のフォルダ・サービスとして提供するために Web フォルダである WebDAV[5] を利用する．そのアーキテクチャを図 3.12 に示す．

WebDAV の実装として Apache HTTP server[115] を利用する．WebDAV はイントラネットのネットワークフォルダとして多く利用されている CIFS [118] や Samba [120] に比べ，WebDAV には Web との接続で優位性がある．

[5] RFC4918（当初は RFC2518）

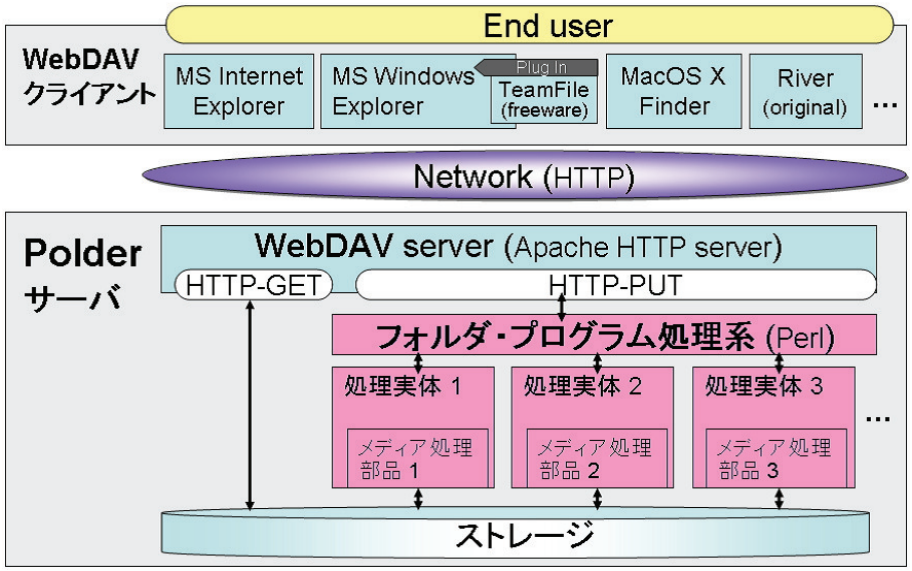


図 3.12: POLDER のシステム・アーキテクチャ

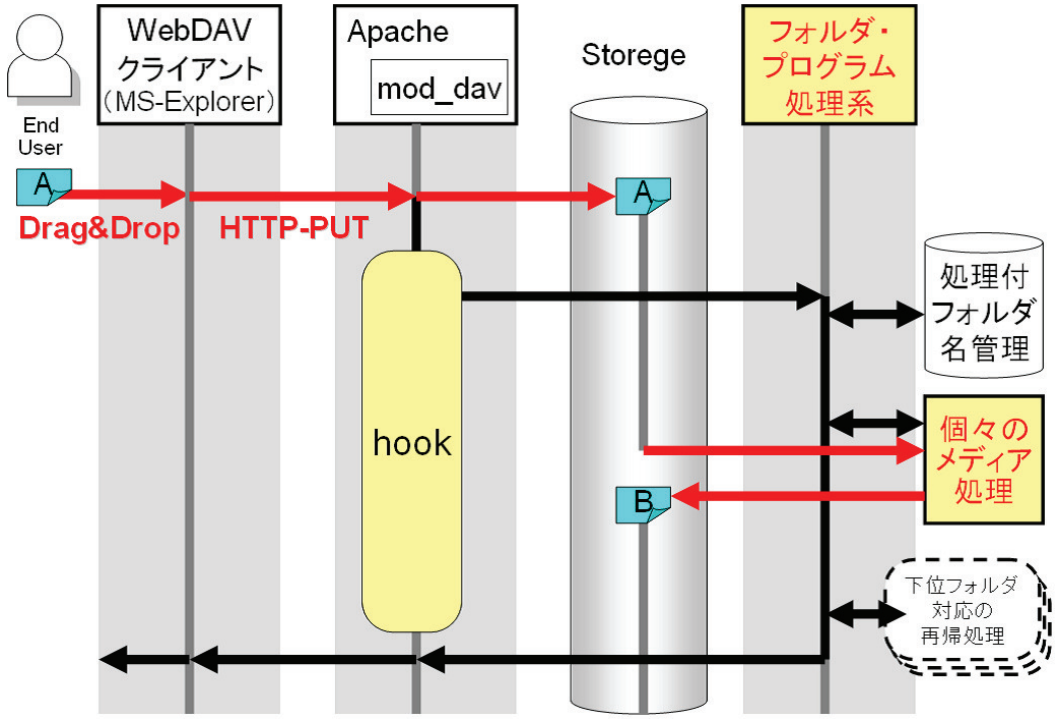


図 3.13: フォルダ拡張後の処理シーケンス

フォルダ・プログラミング環境のサーバシステムは，Apache HTTP server の WebDAV モジュールである mod_dav(a DAV module for Apache)[136] からフォルダ・プログラム処理系 (POLDER Processor) を呼び出す実装となっている．その処理シーケンスを図 3.13 に示す．mod_dav には，WebDAV の各メソッドをハンドリングする部分が存在し，フォルダ・プログラム処理系は HTTP-PUT [6] された直後に，そのファイル実体のフルパスを引数として，子プロセスとして呼び出される．

フォルダ・プログラム処理系は，フォルダ・プログラムの解釈実行と，処理付フォルダに対応する各処理部品実体の呼び出しを行う．処理の連鎖は，フォルダ・プログラム処理系の再帰呼び出しによって実現されている．

フォルダ・プログラミング環境のクライアントは，WebDAV クライアントであれば良い．WebDAV クライアントとフォルダ・プログラミング環境との通信は WebDAV に従う．利用者からは，フォルダ・プログラミング環境は単なるフォルダとしてしか見えない．HTTP や WebDAV を使うことによって発生する各種セキュリティの懸念は，単純には HTTPS をベースとする認証付きの WebDAV による個人フォルダを使って解決する．

3.6.2 フォルダ・プログラム処理系の処理実装

フォルダ・プログラム処理系は，Apache の WebDAV フォルダにファイルを受け付けた直後に，そのファイルの絶対パスを引数として呼び出される．

フォルダ・プログラム処理系は Cent OS 5 [137] (Linux [138] 64 bit) 上，Perl 5.8 [134] で開発管理されている．

フォルダ・プログラム処理系は処理名と処理実体を関連付ける処理名定義表をもつ．この処理名定義表は，処理名と処理実体のフルパスが空白区切りで定義されており，処理名は UTF-8 の日本語であっても構わない．

フォルダ・プログラム処理系の処理骨格を図 3.14 に示す．

[6] フォルダ内でのファイル移動時にもフォルダ・プログラム処理系が実行されるように HTTP-PUT 以外にも HTTP-COPY, HTTP-MOVE も同様のフォルダ・プログラム処理系が実行されるようにしている．ただし，同一フォルダ内での HTTP-MOVE（ファイル名の変更）の場合には，混乱を避けるため，フォルダ・プログラム処理系を起動しないようにしている．

入力：入力データファイルのフルパス，事前に処理名と処理実体の対応を定義した処理名定義表．

step1 引数として与えられる入力データのフルパスから，処理名の候補，処理実体用実引数，処理対象データファイル名を決定する．

step2 処理名の候補が処理名定義表に含まれない場合，データフォルダと判定して終了する．含まれる場合には，処理名を決定し，処理名定義表から処理実体（フルパス）を得る．

step3 処理実体用実引数が 'HELP' の場合には，処理名に対応する HELP ファイルを当該フォルダに複製して終了する．

step4 処理実体を子プロセスで実行する．処理実体への実引数には，処理名，処理実体用実引数，処理対象データファイル名の他，処理結果ファイル名のリストを受け取るファイル（これを OUTLIST ファイルと呼ぶ）名を渡す．処理実体は,「成功」,「失敗」,「照合なし」のいずれかを返却するとともに，処理結果のファイル名一覧を OUTLIST ファイル内に作成する．

step5 OUTLIST ファイルに含まれる各処理結果ファイルについて，以下の step5-1〜step5-5 を適用する．(各処理結果ファイルに対して繰り返し処理であっても並行処理であっても構わない.)

step5-1 現在より一階層下位のフォルダから，処理付フォルダを探す．

step5-2 else を除く任意の処理付フォルダが存在すれば，その全てについて，処理付フォルダ内に処理結果ファイルを複製して，フォルダ・プログラム処理系を再帰的に実行する．

step5-3 上記 step5-2 の全ての処理付フォルダの処理結果を得た後，その全てが「成功」であれば，先の処理対象データは削除する．

step5-4 上記 step5-2 の全ての処理付フォルダの処理結果を得た後，その全てが「照合なし」であって，else フォルダが存在する場合には，else フォルダ内に処理結果ファイルを複製し，フォルダ・プログラム処理系を再帰的に実行する．

step5-5 上記 step5-4 の else 処理の処理結果を得た後,「成功」であれば，先の処理対象データは削除する．

step6 上記 step5 の結果を総合し，全てが「成功」の場合は「成功」,「失敗」が存在する場合は「失敗」,「成功」と「照合なし」が混在する場合には「照合なし」を返却する.「失敗」の場合には，処理結果ファイル名にエラー時刻やプロセス番号などのエラー情報を付加し，エラー発生フォルダの上位フォルダ内に配置する．

図 3.14: フォルダ・プログラム処理系の処理骨格

フォルダ・プログラム処理系が各処理実体を呼び出す際のインタフェースを図 fig:処理実体の呼び出しインタフェースに示す．処理実体名及び次に示す 4 つの引数からなり，各処理は別プロセスとして実行される．

-c 処理名
-p 処理実体用引数
-o OUTLIST ファイル名
処理対象データファイル名

図 3.15: 処理実体の呼び出しインタフェース

処理実体は，戻り値として「成功」,「失敗」,「照合なし」のいずれかを返却するとともに,「成功」の場合には OUTLIST ファイルの中に，処理結果として生成されたファイル群のファイル名のリストを返却する．写真の中から顔画像を抽出した例のように，処理実体は 1 つ入力ファイルに対して 2 つ以上の処理結果を返すこともできる．そのため，処理結果ファイル名のリストは OUTLIST ファイルの中に生成される．OUTLIST ファイルに含まれるファイルが処理連鎖の対象となる．また，写真の中に顔画像が含まれなかった場合には処理実体の戻り値は「照合なし」となり，else フォルダに処理が連鎖される．else フォルダは並行階層に存在する他の処理付フォルダの全てが「照合なし」の場合にのみ連鎖される．処理の成功後に処理対象データファイルは削除されるが，処理結果ファイルが同一名の場合には，削除されない．

3.6.3 処理付フォルダの各処理部品の実装

各処理実体は，Linux のコマンドとして実装されている処理部品を Bash [133] や Perl [134] によって処理付フォルダの部品用にラッピングすることで実装される．処理は，基本的にはフォルダ・プログラム処理系が子プロセスを起動したディレクトリで実行され，その結果も，同一フォルダに生成される[7]．処理結果の状態は,「正常」,「失敗」,「照合なし」の 3 種類のいずれかとなる．

新たな処理実体の組み込みは，既存のものと同インタフェースの処理部品の拡充であれば，ほんの数行の変更で済む．異なるインタフェースの処理部品の組み込みでも，テンプ

[7] この後,「失敗時」は処理系によって上位フォルダに移動される.「照合なし」時はそのまま削除される.

レートをベースに修正することで，容易に導入できるが，特に引数の解釈部分や，エラー処理に対する救済処理，複数同時に実行される場合のワークファイルの競合，その他の個々の処理部品に特有の制限[8]がある場合には，それに対応するためのプログラム追加が必要になる．処理実体がLinuxコマンドとして準備できた後に，フォルダ・プログラム処理系の処理名定義表に既存の処理名と重複しないこと「処理名」と「コマンドフルパス」を空白区切りで1行追加定義することで，処理付フォルダとして利用が可能になる．例えば，オープンソース・ソフトウェアのImageMagickでは「/usr/bin/convert」コマンドが処理部品に対応し，「/opt/polder/bin/image-resize.sh」は，そのconvertコマンドをBashでラッピングした処理実体である．処理名は「image-resize」や「画像サイズ変更」など，複数行登録することができる．更に，ヘルプ時に参照されるファイルを,「処理名.html」として用意する必要がある．

3.6.4 外部サーバとの連携

フォルダ･プログラミング環境が複数ある場合には，それらの処理をフォルダ･プログラミング環境上で接続することが可能である．図3.16に，その接続構成を示す．HTTP/HTTPSは，GatewayやFireWallを越えて処理を接続することが可能である[9]．Apache HTTP server [115] のWebDAV（mod_dav [136] ）経由で処理をリモートから呼び出す処理部品を作成することで，一連のフォルダ・プログラムとして連携して処理を実行させることが可能である．remotecallは3.4節で述べたcallと類似のプログラムロジックをもつ処理付フォルダである．remotecallはローカルファイル操作のcopy/moveの代わりに，HTTP-PUT/HTTP-GET/HTTP-DELETEを使う．HTTP-PUTによりファイルをリモートサーバに送ることで処理が実行され，処理結果が得られた後，結果ファイルをHTTP-GETで取得する（複数の結果がある場合は繰り返される）．最後にHTTP-GETしたファイルをHTTP-DELETEで削除する．これにより，インターネットで接続された異なるロケーションに存在するサーバ上のフォルダ・プログラミング環境においてサブルーチン定義されたフォルダ・プログラムを簡単に利用することができる．

現在のフォルダ･プログラミング環境の標準的な開発･実行環境はLinux 64 bit環境である．しかし，処理部品は処理ライブラリの制限などで，Linux 32 bitやMicrosoft Windows [139] などに限定されるものがある．それらの処理部品であっても，フォルダ・プログラミング環境に組み込まれていれば，インターネット(WebDAV)を経由して接続し利用できる．また，フォルダ・プログラム処理系のフルセットは動作しない場合であっても，WebDAVを経由して処理付フォルダだけが利用可能である部分セット（処理付フォルダ実行環境）

[8]ファイルパスに日本語や半角空白を許容しないコマンド等も時々存在する．その場合には，/tmp等のワークエリアにデータを一旦移動して処理を行う必要がある．

[9]その他，認証が必要になる場合もある．

が用意できれば，図 3.16 の右下（処理付フォルダ実行環境 3）に示すような簡易接続も可能である．この簡易接続の構成は，メディア処理を WebAPI として提供したいが，フォルダ・プログラミング環境までは提供したくない場合にも利用できる接続方法である．

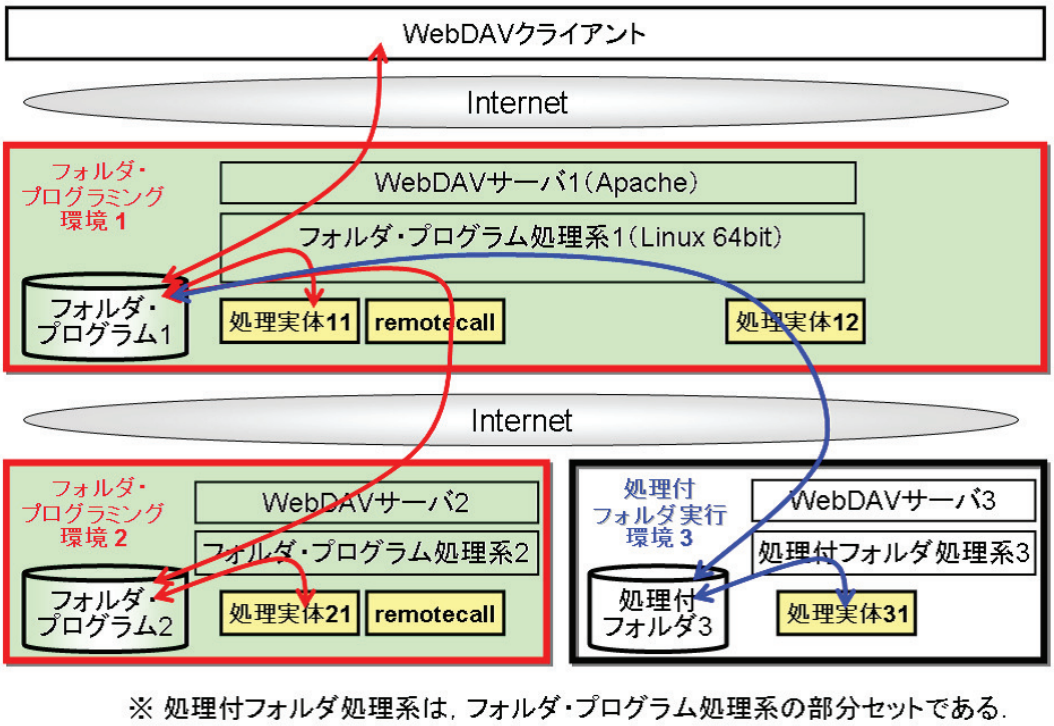


図 3.16: Web 経由での処理の接続

3.6.5 クライアント

クライアントは，WebDAV [19] [20] [140] に対応しているものであれば良い．Microsoft Windows Explorer [116]，Microsoft Internet Explorer [141]，Apple Mac OS X Finder [117] や，Microsoft Windows Explorer の拡張プラグインである Computer Hi-Tech Inc. の TeamFile [142] (freeware)，その他オープンソースの WebDAV クライアント[10]など，多くの WebDAV クライアントからアクセスすることが可能である．

フォルダへの DROP は，GUI 上では複数のパターンが存在する．Microsoft Windows Explorer の場合，図 3.17 に示す以下の 4 つのケースで，いずれも同一の挙動となる．ケー

[10]オープンソースの WebDAV クライアント選択時の注意点としては，日本語対応度，OS の Viewer などのアプリケーションとの統合度，キャッシュの更新契機，複数ファイルの一括投入の可否，タイムアウト制御の可否などがある．

ス 4 の場合，プログラマは図 3.18 に示すような REST 的なプログラミング・インタフェースで，フォルダ・プログラミング環境を使うことが可能になる．

ケース 1: ツリービューで直上フォルダを選択し，ターゲットフォルダのアイコン上に Drag&Drop するケース

ケース 2: ツリービュー上のターゲットフォルダ上に Drag&Drop するケース

ケース 3: ツリービューでターゲットフォルダを選択し，空きスペースに Drag&Drop するケース

ケース 4: curl 等のコマンドライン・ツールから，HTTP-PUT するケース

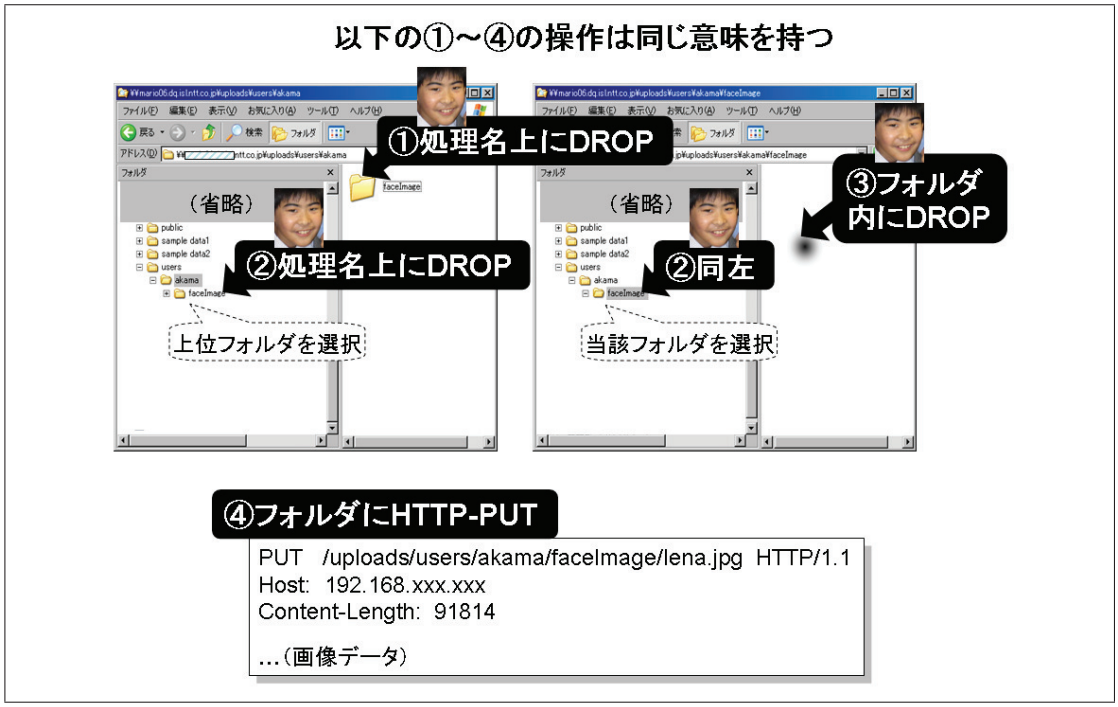


図 3.17: フォルダへの 3 つの DROP のパターンと HTTP-PUT の同一性

Step1　結果が生成されるべき場所のファイルを DELETE

DELETE　/uploads/users/akama/OUT.txt　HTTP/1.1　Host: 192.168.xxx.xxx

Step2　フォルダ・プログラムのエントリにデータを PUT

PUT /uploads/users/akama/textGrep 'ABC'/INP.txt HTTP/1.1
Host: 192.168.xxx.xxx Content-Length: 2800
…（2800 byte のデータ）

Step3　処理結果が生成されるまで GET を繰り返すことで待つ（繰り返しの実現は JavaScript 等を活用）

GET　/uploads/users/akama/OUT.txt　HTTP/1.1　Host: 192.168.xxx.xxx

図 3.18: フォルダ・プログラム処理系を HTTP-PUT で直接利用する例

3.7　簡単な応用例

3.7.1 処理付フォルダの部品例

処理付フォルダとしては，初期の実装では以下のような部品を用意した．なお，osaka-ben や ascii-art などは，試用者のモチベーション・アップやデモンストレーションのために用意した．

テキスト処理部品系

- textGrep, textSort, textUniq: UNIX の基本コマンド
- charactor-code: 文字コード変換（nkf 相当）

- osaka-ben: 大阪弁文書に変換

画像処理部品系

- image-resize, image-nega, image-info: 画像基本操作
- image-format: 画像フォーマット変換
- faceImage, faceData: 画像内顔画像検出，領域検出
- ascii-art: アスキーアートの txt もしくは html 作成

数式部品系

- expEval: ファイル内数式を評価（計算）

プログラミング支援部品系

- pFo: プログラムへの名前付け，投入連鎖
- fileCreate: 引数を評価し，ファイル化
- fileRename: 投入ファイル名を変更
- putText: 指定 URL に引数を PUT
- link: 指定 URL に投入ファイルを PUT
- if: 引数の条件を満たせば投入連鎖

3.7.2 データ分類への応用例

応用事例については，既にここまでにいくつか述べてきているが，図 3.19 はファイル内の行の分類を行うプログラム例を示している．

```
users
   demouser
      pFo demo-Grouping  <
         textGrep 'A'
            fileRename GA.txt
               link %users%demo%RESULT
         textGrep 'B'
            fileRename GB.txt
               link %users%demo%RESULT
         textGrep 'C'
            fileRename GC.txt
               link %users%demo%RESULT
         else
      RESULT
```

図 3.19: データ分類への簡単な応用例

この例では，フォルダ構造をインデントを使ったテキスト表現 (3.5 節 (fs4)) で記述している．このプログラムは「pFo demo-Grouping」フォルダに入力テキストファイルを DROP すると，'A' が含まれる行，'B' が含まれる行，'C' が含まれる行に分類（重複行あり）し，各々GA.txt, GB.txt, GC.txt という名称で 「/users/demo/RESULT」フォルダ内に結果を格納する.「textGrep」は入力ファイルに対して，引数の文字列の Grep 結果を結果ファイルとして作成する.「fileRename」は入力ファイルに対して，引数のファイル名に名称を変更する.「link」は入力ファイルを指定のフォルダに移動する.「else」の働きによって，'A','B','C' のいずれも含まれない行は，いずれのファイル内にも出力されない.

3.8 言語処理系の組み込み例

3.8.1 簡易 Perl(PPL13pl) を組込む処理付フォルダの例

フォルダ・プログラミング環境は，UNIX のシェルの拡張と考えることもできる．シェルが bc[11], sed[12], awk[13], Perl などの言語処理系を活用できるのと同様に，POLDER でも個々の処理付フォルダとして，多彩な言語処理系を組み込み連携することが可能である．ここでは，Perl を処理付フォルダとして組込む例を紹介する．

処理付フォルダ pl の仕様

Perl 言語 [143] [134] は，シェルをベースに，sed, awk, C などの言語の特徴を吸収して拡張した言語である．その言語コンセプトは実用性と多様性であり，その言語仕様の複雑さには様々な批判があるものの，広く支持され用いられている[14]．

今回実装する言語を PPL13pl [15] と呼ぶことにし，処理付フォルダ pl としてサンプル実装した．以下はその仕様である．

- 投入ファイルを代入文形式で表現された変数空間の推移履歴とする．そのファイルを Perl で評価実行すると最新の変数空間が再現されるものとする．
- 原則として Perl の文法に従う．処理付フォルダの引数で指定された文字列を Perl 文と解釈し，投入ファイルの最終行に追加したものをテンポラリファイルとして作成し，そのファイル全体を Perl で実行し評価を行う．
- 実行評価で変更された変数値を，投入ファイルの最終行に変数への代入文として追加する．これによって変数空間の更新を行う．(ただし非 if 文のケース)
- 変更される変数値を推移履歴に反映させるための代入文は，以下の構文パターンに応じた処理を行う．
    - 単純代入文（例 $SUM = $SUM + $FG や $SUM = sqrt($SUM + 4)）なら，右辺の評価結果の左辺への代入文を生成する．右辺には sqrt などの数値関数が入っても構わない．

[11] 任意精度の数値計算言語
[12] ストリームエディタ
[13] CSV などのテキスト処理を得意とする言語
[14] 本研究の POLDER エンジンも，約 1KLine の Perl で実装されている．
[15] Polder Programming Language 2013 based on PerL

- 副作用あり[16]配列関数（push, pop, delete, shift, unsift）の代入文（例 $C=shift @INP）なら，上記の代入文，及び副作用の結果反映のための配列代入文を生成.

- 副作用なし配列代入文（例 @INP=sort @INP や @INP=split x,@INP）なら，右辺の評価結果の左辺への配列代入文を生成．副作用なしの配列関数（sort, reverse など）も利用できる[17].

- 副作用あり配列単独関数（例 push @INP END）なら，副作用の結果反映のための配列代入文を生成.

- if 条件の場合は，投入ファイルの最終行に if の評価式を追加し，実行評価し，その真偽に応じて成功なら 0 ，失敗なら 100 [18]を，投入ファイルの最終行に追加する．この値はそのまま処理付フォルダ pl の実行結果とする．また，デバッグ情報として評価対象の if 文はコメント行として出力する.
- 投入ファイル名に.V を追加したものを処理結果ファイルとする.
- 変数値等は，投入ファイル内で記述しても，フォルダ名で代入文として記述（例 $INP = QxWxExRxTxYxU）してもよい.
- CIFS の文字コード制限には個別に対処する.
- 最終処理結果ファイルの最終行もしくはそこから上に遡る形で，変数値の最終計算結果が示される.

[16]ここで副作用とは，引数として与えた配列自身を書き換えてしまうことを意味する.
[17]素直に sort を使えば，3.8.3 節の例のような面倒なプログラムを作成しなくても実は同一の結果を得ることができる.
[18]C 言語のように，成功が非 0，失敗が 0，としない理由は，シェルや POLDER の処理部品実装の仕様にあわせたためである．POLDER の処理付フォルダの戻り値は，0 が成功，100 が照合しない，その他の値がエラー，としている.

3.8.2 PPL13pl による繰り返し処理を含む数値計算プログラム例

図 3.20 は，変数の減算と goto 文[19]による繰り返しを使って簡単な数値計算を行うフォルダ・プログラム例[20]である．そして図 3.21 は，そのフォルダ・プログラムへの入力ファイルと最終的な結果を含む出力ファイルを示している．フォルダ・プログラムの数値計算の部分は，そのまま Perl が解釈と実行を行うため abs,int,log,rand,srand,sqrt,sin,cos,atan2,time などの数値関数が利用可能である．フォルダ・プログラム内の conf input_data は単なるデータフォルダであり，計算には影響を与えない．ここにデバッグ用のフォルダ・プログラムやテストデータを格納することが可能である．入力ファイルでは，1 行目は単なるコメント文，2 行目は計算のために変数に初期値を与えている．最終処理結果ファイルは，処理に伴って行数が追加されてきたファイルであり，計算過程の変数空間の推移を表している．最終行に計算結果$SUM = 7 が示されている．

図 3.20: 繰り返し処理を含む数値計算フォルダ・プログラム例

[19]link 文と実体は同一である．分かりやすさのために goto という別名を与えている．
[20]注意：図中の矢印線や吹き出しは図の理解をサポートするために書き加えており，ビジュアル・プログラミングとしての線ではない．

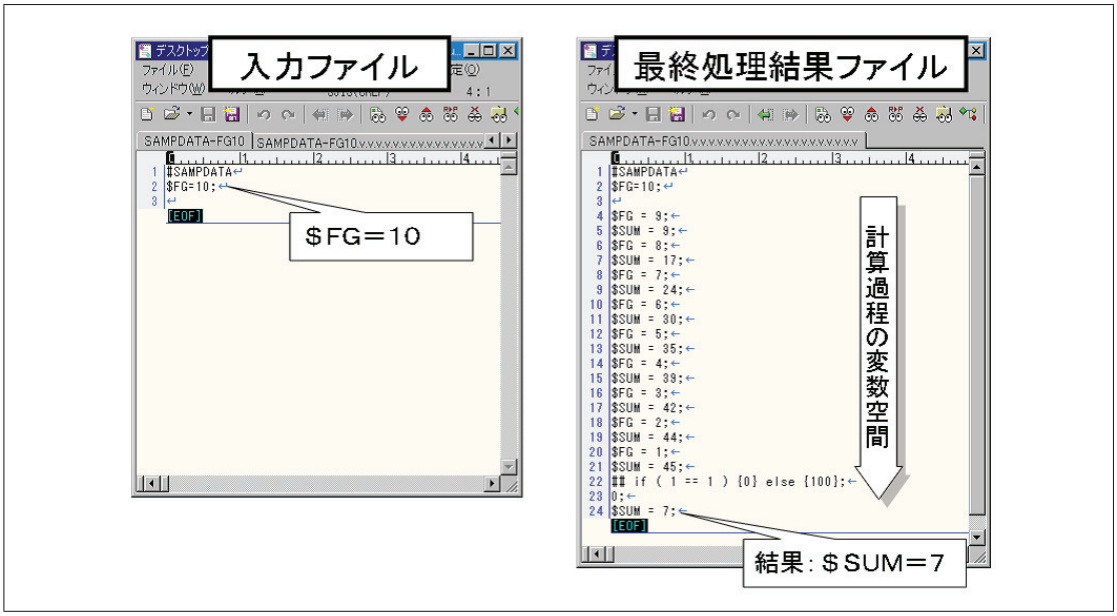


図 3.21: 入力ファイルと最終処理結果ファイル

3.8.3 PPL13pl による最大文字を探す poldef サブルーチン例

図 3.22 は，Perl の文字配列を使った文字列内の文字探索と poldef を使ったサブルーチン定義の簡単なフォルダ・プログラム例である．図 3.23 は，そのフォルダ・プログラムへの入力ファイルと最終的な結果を含む出力ファイルを示している．このフォルダ・プログラムでは，Perl の配列関数 push,pop,delete,shift,unsift を利用している．入力ファイルでは，1 行目は単なるコメント文のみであるが，先の例のように配列値の代入等（例 @INP=(Q,W,E,R,T,Y,U); ）を入力ファイル中で行っても構わない．図 3.22 の例では文字列の前後のクオートを省略している点など，構文内には Perl のシンタックスシュガーが多数利用されている．最終処理結果ファイルの最終行に計算結果$MAX = Y が示されている[21]．これは初期データ配列 [Q,W,E,R,T,Y,U] の中での最大の文字が Y であったことを示している．

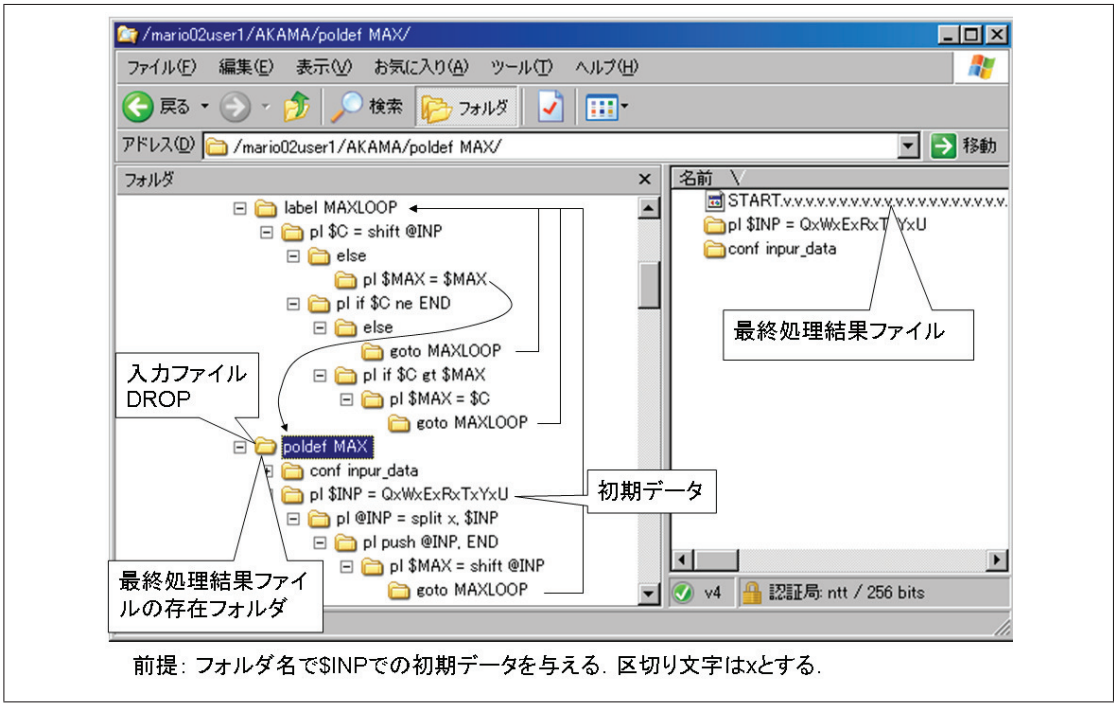


図 3.22: 最大文字を探す poldef サブルーチン例

[21] $MAX=$MAX という行は処理結果ファイルの最終行に$MAX=7 を出力するために行っている．

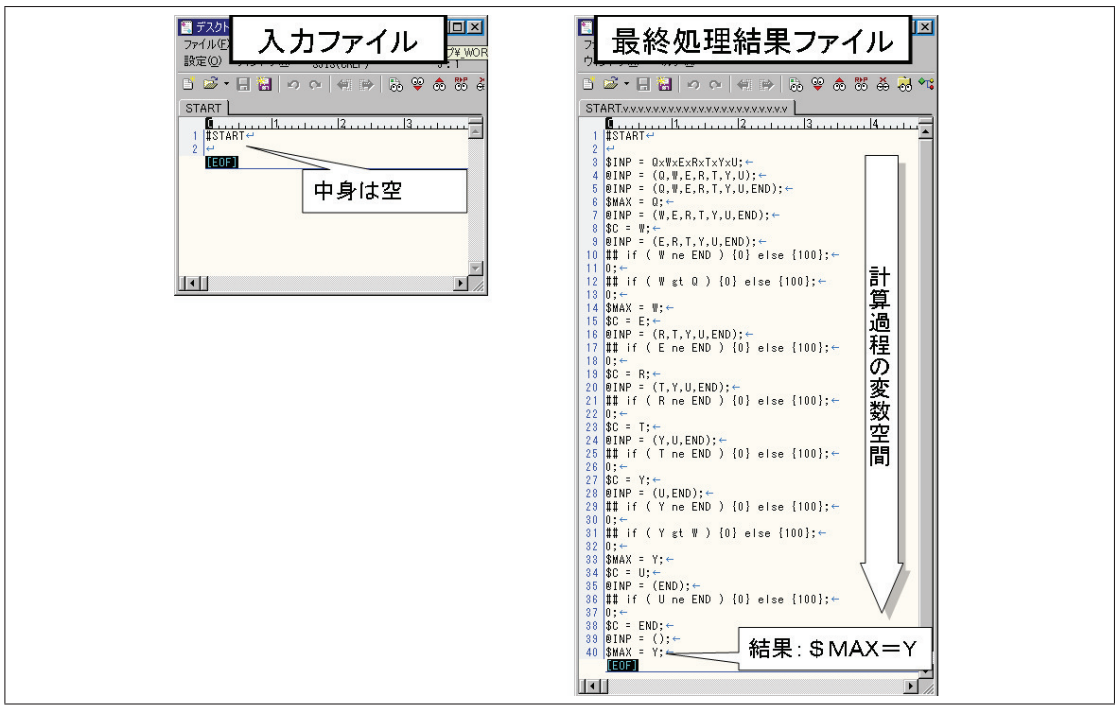


図 3.23: 入力ファイルと最終処理結果ファイル

POLDER に Perl を組み込む例，および，それを用いた数値計算や繰り返しの実現例を示した．今回の簡易言語 PPL13pl の処理付フォルダ pl は Perl で 400 行程度の簡素な実装となっている．しかし，本格的な組み込み言語の仕様については，CIFS[118] 制限の統一的な回避方法などを含めた検討が必要である．

3.9 エンドユーザ向けプログラミング環境としての評価

エンドユーザ・プログラミングを指向するスクリプト言語としてチャミー [21] がある．ここではチャミーの設計目標を参考にしつつ，フォルダ・プログラミング環境がエンドユーザ・プログラミング向けに提供する効果を考察する．

1. **母国語だけでプログラムの読み書きができるか**： 母国語対応は，フォルダ名の多言語機能により対応できる．ただし，条件分岐のような処理を「if」でなく「もし」という名前で用意するなどの徹底は任意である．処理名については「画像内の顔画像抽出」などの分かり易い名称を付与することが可能であり，母国語だけの使用を

望むプログラミング初心者に有効だと考える．ただし，チャミーのような母国語に対する徹底さはない．

2. **基本となる知識・原則の少なさ**： まず「フォルダの外側からデータが投入されてフォルダ名に応じた処理が実行され，その結果がフォルダの中に生成され，それが下側のフォルダに連鎖していく」というシンプルな原則がある．それを既存のフォルダの GUI を使ってそのままビジュアルに把握できるため，新たに GUI 操作方法を学ぶ必要がない．このためプログラミング初心者を簡単に導入可能だと思われる．

3. **初期実行（簡単なプログラムを動かすまで）の学習必要量の少なさ**： 最小の動作が，フォルダを作って，ファイルを DROP するという 2 ステップで完了する点で初期学習量が極めて少ないと言える．入出力はファイル（群）であり，特に入出力インタフェースを用意することなく部品を単独で実行可能であり，初期学習量は最低限で済む．この特長及び後に述べるモードの少なさによって，チャミーが対象としたエンドユーザ（既存のプログラムの大体の意味を理解することができるレベル）よりも更にエンド寄りの利用者，つまりこれまでプログラミングに関わらなかった利用者層がプログラミング環境との接点をもつことが可能になる．

4. **初期編集時（簡単なデバッグ方法を学ぶまで）の学習必要量の少なさ**： 簡単な 3 つ方法を学ぶことで完了する．(1) フォルダを単独で作成して入出力を見る．(部品の単独実行可能)，(2) 連鎖途中のフォルダ名を変更して，当該フォルダ処理直前の入力情報を知る．(途中状態の再現)，(3) 途中階層のフォルダにデータを投入して処理を再開させる．以上は，フォルダ・メタファの素直な応用の範囲の知識に留まるため，デバッグのための初期学習量は少なく，かつ，忘れたとしても想起は容易である．

5. **モードの少なさ**： 従来のプログラミング環境では，開発編集モードと実行モードの切り替えや，一度停止したプログラミング作業の再開のために環境の読込み作業が存在した．フォルダ・プログラミング環境では，プログラムが常にフォルダ階層としてファイルシステム上に永続化されておりプログラムの保存や読込みなどを意識しなくて良い．更に，フォルダに対していつでもファイルを DROP して実行することができ，開発編集モードと実行モードの区別がなく，プログラミング初心者に優しい．

6. **プログラムの部品の探しやすさ**： 基本的にはマニュアルとして作成された Web ページの解説を探すという手法を採ることになる．ブラウザの Web Folder Behaviors[28] を利用すれば，検索して見つけた機能名称のリンクをクリックすると，即，その機能名のフォルダが表示されるようになる．そのフォルダに入力ファイルを DROP す

ると単独の部品としての実行結果がそのフォルダ内に得られ，希望の動作の部品か否かを容易に判断できるようになる．更に，より高度な部品の探し方としては，各機能名のフォルダを横に並べ，並行実行を利用して同時に入力ファイルを処理させ，その後，各フォルダ中に出力として期待のファイルが存在するかを探索することで，目的の機能の処理付フォルダを探し出す方法も可能である．

7. **よく使う実用的な場面に使えるか**： 一般には大量のデータファイルに対して同じ処理を繰り返す行為の自動化が実用的場面では役立つ．フォルダ・プログラミング環境では，複数のファイルを一斉に選択し，ファイルに一括 DROP することで，その各々に対する処理を一斉に実施することができ，例えば，大量の画像ファイルに対するサイズ変更処理が，極めて少ない操作で簡易にできる．また，もう 1 つの実用的な場面として，デスクトップ上のプログラムと Web 上の処理のマッシュアップが可能になることが考えられる．例えば，デスクトップ上の動画編集ツールが MPEG-2[29] しか出力できなかったとして，ネットワーク上の処理付フォルダが MPEG-2 をより圧縮率の高い H.264[30] に変換する機能を提供していれば，それらを連携（動画編集ツールの「ファイルに保存」時にネットワーク上のフォルダを指定）することが可能になる．

3.10 ビジュアル・プログラミング環境としての評価

データフロー・ビジュアル・プログラミング言語を想定した場合の各種課題についてフォルダ・プログラミング環境の対応を考察する．これまでビジュアル・プログラミング言語の問題点は各種指摘されているが，ここでは，Myers [54]，田中 [69]，増井 [144] から抽出した課題を示し，フォルダ・プログラミング環境での対応を分析する．

1. **大規模プログラムへの対応力**： ディスプレイの解像度や計算機の描画処理の能力が大幅に向上しビジュアル・プログラミングの障壁は低減しつつある．しかし，大規模プログラムになれば画面内には納まらず，また，Pure Data や Yahoo! Pipes の例でも時々見られるが，多数の線が入り組んでしまい，ワイヤの迷路が発生することになる．これらに対し，適切なズーミング，モジュール化，抽象化などの支援が必要になる．

   大規模プログラムには，サイズ（プログラム行数や描画エリア）の多さ，処理の複雑さ，開発の多人数化など，いくつかの視点が存在する．フォルダ・プログラミング環境では，行数の多いプログラムであっても，Explorer の木状ビューの各節にある展開（[−] マーク）と折りたたみ（[+] マーク）とでプログラムのズームイン/

アウトを行うことができる．このとき，木状ビュー上でプログラムを作成・更新することも可能である[22]．また，処理の抽象化 (poldef) により，複雑な処理に対してモジュール化開発も可能になっている．ただし，多人数でのプロジェクト開発を進めるような大規模プログラム開発への対応力については，十分とはいえない．

2. **自動レイアウト，入力の手間の削減**： ビジュアル・プログラミング環境では，プログラムの各要素の入力と配置，全体のレイアウトも重要な問題となる．まず，マウスを使って各要素を選択し，入力し，配置し，結線することは手間がかかる．更に，綺麗な要素の配置と結線には更に多くの手間と美的センスが必要になる．PP は，この問題を自動レイアウトで解決していた．しかし，一般論として十分に分かりやすい自動レイアウトの実現は難しい課題である．

   フォルダ・プログラミング環境では，アイコンの選択は不要で，代わりに処理付フォルダ名の入力となる．また，配置については基本的にはフォルダの階層構成の作成だけなので，図的な微調整は不要で，結線も必要ない．そのため大幅な手間の増加はない．更に，フォルダ・プログラミング環境は後に 5.2.1 節で示すようなテキスト・ユーザ・インタフェースを備えることができ，熟練者は高速な入力が可能になる．

3. **分野特化のオモチャのシステムしかできない．幅広い異分野への対応力**： 多くのビジュアル・プログラミング言語は，既存言語と比べ，分野が限定され，機能が不十分で基本機能しかないと指摘されている．例えば，Yahoo! Pipes は RSS のマッシュアップを重点的にサポートしているが，その他の応用には制限が強い．

   ビジュアル・プログラミングの本質として，具体的なものは扱いやすいが，抽象的なものは逆に扱いが難しくなるという特性が存在する．例えば，JPEG 画像から PNG 画像への変換の機能を絵だけで表すのは困難である．よって，一般的な処理を実現するためには，言語の併用は避けられない．HI-VISUAL はアイコンに言葉を付記していたが，分野は限定されていた．一方，PP は表現能力を維持するため，既存の言語 (GHC) をベースとして能力を落とさずビジュアル化し，更に相互連携したことで，トイ・プログラムからの脱出を図っている．

   フォルダ・プログラミング環境の処理付フォルダは，アイコンで指定するのではなく，テキスト名を使っていた．また，組み込み言語の連携が可能という拡張性が存在する．例えば，3.8 節で紹介したように Perl 言語の機能を取り込むことも可能である．UNIX シェルの Bash が，その問題に応じて異言語 Sed,AWK,Perl,C で書かれた部品群を接続していくのと同様に，フォルダ・プログラミング環境も，異分野の処理部品をファイルとデスクトップという仕組みで接続していく．これにより分

[22] WebDAV クライアントに依存する．

野を超えた高い柔軟性が得られると考える．例えば，3D プリンタの登場で，今後のデスクトップには 3D モデルに関するデータが増えてくると予想される．それらを扱うソフトウェアを OS に導入することで，3D モデルに関する処理はフォルダ・プログラミング環境とシームレスに統合される．

4. **ポータビリティの向上**：　テキスト・プログラムに比べ，ビジュアル・プログラムは他者との共有がしにくいという問題がある．初期のビジュアル・プログラミング環境は，クローズな環境で実現され利用されてきた．よって他者がそれを利用することは難しいという問題があった．しかし，現在では Web の登場や OSS やフリーライセンスの文化の発展で，我々は，Pure Data のようなビジュアル・プログラミング環境を手軽に入手できるようになった．また，Yahoo! Pipes のように，Web 上でサービスを立ち上げることで，どこからでもそれを利用することが可能になる．

   フォルダ・プログラムは，それ自身はテキストに変換可能であるため，メール本文に添付して送付したり，文書内に気軽に埋め込んだりすることが容易であり，ポータビリティは高い．更にフォルダ・プログラム環境の処理系は WebDAV をベースに Web サービスにも対応している．よってポータビリティは高いと言える．

UNIX 系のシェルで提供される pipe との比較

Pipe [145] は，簡易なデータフロー処理 [144] であり，ファイルに対して，複数の処理を組合せた一連の処理を行うための便利なメカニズムとして広く利用されている．しかし，ビジュアル環境を活かしたものではない．一方，フォルダ・プログラミング環境もデータフロー処理を基本としており，シェルの pipe をビジュアル化したものと捉えることもできる．フォルダ・プログラミング環境における処理付フォルダを UNIX 上での各コマンドと考えて対応づけることは自然で，フォルダ・プログラミング環境は，シェルの pipe の拡張系と位置づけることもできる．その拡張された機能の代表は並行実行である．これはフォルダ・メタファの木状表示によってユーザに直感的で分かりやすく提供される．また並行実行を活用した条件分岐等のより高度な処理記述と応用範囲の拡大も可能にしている．図 3.24 は主要なポイントの比較を示している．この表に示した全ての点でフォルダ・プログラミング環境は従来のシェルの pipe に比べて，より良い特性を持っていると考えられる．

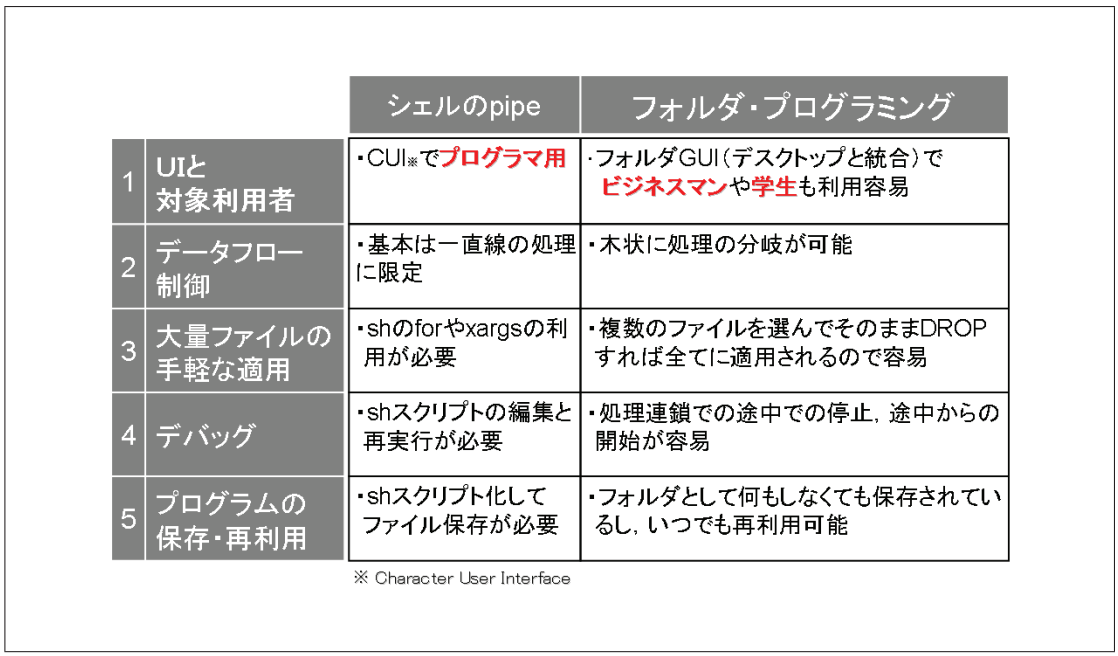


図 3.24: UNIX シェルの pipe とフォルダ・プログラミングとの比較

3.11 おわりに

本章では，我々の提案するフォルダ・プログラミング環境を定義した．それはフォルダを拡張したプログラミング環境であり，「フォルダの外側からデータが投入されてフォルダ名に応じた処理が実行され，その結果がフォルダの中に生成され，それが下側のフォルダに連鎖していく」というシンプルな基本原則を使って PC 利用者にも簡単な処理の組合せと実行環境を提供するプログラミング環境である．一杉の設計目標を使った分析から，フォルダ・プログラミング環境がエンドユーザ向けのプログラミングの設計目標に多くの点で適合していることを定性的に確認した．更に，ビジュアル・プログラミング言語との評価，及びシェルの pipe に対する優位性を確認した．

第4章　フォルダ・プログラミング環境のメディア処理統合への適用

4.1　はじめに

本章では，フォルダ・プログラミング環境をメディア処理部品の統合に利用することの効果について分析と評価を行う．

まず，マルチメディア処理をマッシュアップする環境に対する要件を示す．そして，その要件をユーザ実験と，メディア処理への適用実践を通して分析する．

はじめに学生 10 名を対象とし，フォルダ・プログラミング環境を使って画像のファイルを処理する簡易な実験の結果を示す．その結果，フォルダ・プログラミングのコンセプトは十分受け入れられるというアンケート結果を示す．次に，フォルダ・プログラミング環境を応用して開発した虹雲ノートシステムの紹介と，そこで得られた使用経験を示す．

それらをもとに，フォルダ・プログラミング環境が，メディア統合処理環境の 5 つの要件，(M1) マルチメディア対応性，(M2)Web API 対応性，(M3) 単機能利用の容易性，(M4) 応用利用の容易性，(M5) 管理の容易性，に対して，どのような効果を提供するのかについて考察し，フォルダ・プログラミングがメディア処理部品の統合の要件を満たすことを示す．

4.2　メディア処理のマッシュアップ環境の要件

多様なメディアに対する機能を簡単にマッシュアップ [104] [105] するためには，そのメディア処理部品統合環境には以下の 5 つの要件が求められる．

(M1) マルチメディア対応性

この要件は，マルチメディアつまり多様なデータが利用できることである．ここでマルチメディアとは，画像，映像，音声，音楽はもちろん，テキスト，構造化テキスト，オフィス作業で用いられる Microsoft Word, PowerPoint, Excel なども含んでいる．一般的に PC で扱えるデータの多様性と同等である．

(M2) Web API 対応性

この要件は，サーバやクラウド側で提供されるWebサービスを，クライアントからWebを経由して簡単に利用できることである．一般的なHTTP/HTTPSに対応すれば良い．

(M3) 単機能利用の容易性

この要件は，任意の処理を1つだけ選んだとき，それを単機能として簡単に実行し結果を得ることができることである．学習量や作業量の側面からの容易性が必要となる．

(M4) 応用利用の容易性

この要件は，状況に応じた処理の柔軟な利用が簡単であることである．この要件は，我々の目指すエンドユーザ向けプログラミングの要件となる．特に4点が簡単であることが重要である．

(M4a) **部品合成容易性:** 単機能の部品を複数組合せて連続して利用することが簡単にできること．

(M4b) **手動部品接続性:** 手動で部品の入出力を接続することで簡単に利用できること．

(M4c) **大量適用容易性:** 大量のデータに対して，繰り返し適用作業が簡単にできること．

(M4d) **高度利用への移行容易性:** 単機能だけを利用する利用者と，部品を合成して利用する利用者，大量データへの作業を行う利用者の間の障壁が低いこと．

手動部品接続性が必要になる理由は，GIMP [146] など，日常利用しているデスクトップ上のアプリケーションと処理を連携したいケースや，わざわざ合成部品を作らなくても結果を得たいケースにも対応するためである．

(M5) 管理の容易性

この要件は，以下の3つの管理的側面が簡単であることである．

(M5a) **合成処理保管の容易性:** 作成した処理の保管が簡単であること．

(M5b) **処理発見の容易性:** 利用したい処理の発見が簡単であること．

(M5c) **合成処理変更の容易性:** 作成した処理の変更が簡単であること.

4.3 簡易な実験

4.3.1 実験

10 名の大学生を対象とした実験を行い，その後アンケートを行った．最初に，学生はフォルダ・プログラミングの説明を受け，実装してある処理付フォルダの処理内容や使い方や例を書いたメモをもとにフォルダ・プログラミング環境を使って，以下の課題を解いてもらった.

Task1: ある画像に対し，プログラムを作ってモノクロ化し，サイズ変更し，油絵化し，外枠をつける.

Task2: 与えられた画像に対し，プログラムを作ってこの画像と同じに加工せよ.（ネガポジ反転，モノクロ化，外枠の付与が必要）

全ての学生がこれらのタスクを完了したのち，アンケートは 2 つの質問からなるアンケートを行った.

Q1: 処理の付いたフォルダの組合せでプログラミングをするというコンセプトは理解できたか？

A1: とてもよく理解できた.

A2: よく理解できた.

A3: おおむね理解できた.

A4: 一部理解できない部分があった.

A5: 理解が難しかった.

Q2: フォルダにデータファイルを Drag&Drop することで処理ができるインタフェースは，エンドユーザに容易に受け入れられると思うか？

A1: たいへんそう思う.

A2: そう思う.

A3: どちらともいえない.

A4: あまり思わない.

A5: そうは思わない.

4.3.2 実験結果

更に利用経験に基づいた意見とコメントを収集した．図 4.1 にその結果を示す.

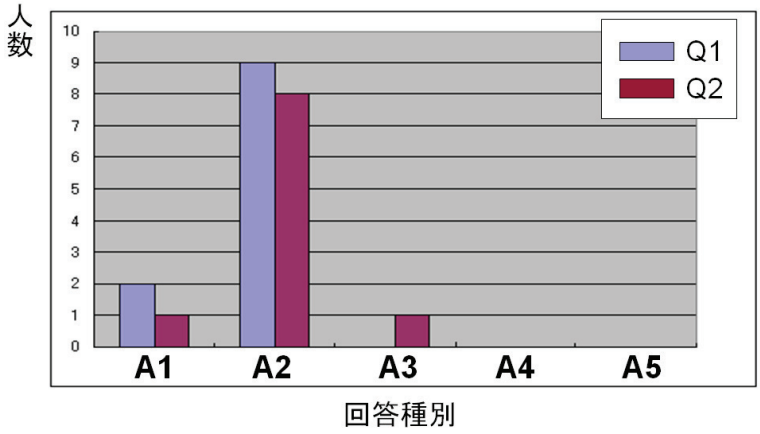


図 4.1: 利用者アンケートの結果

評価アンケートの結果，Q1 に対して全ての生徒は A1 か A2 を選択した．このアンケートの結果によりフォルダ・プログラミングのコンセプトが簡単に理解できることが確認できた.

Q2 に対しては 10 名中 9 名の学生が A1 か A2 を選択し，9 割が一般エンドユーザは受け入れられるとの回答結果を得た．この結果から，エンドユーザに対する障害は低いと考えられる.

実験参加者の学生は，日常的に PC を利用しているため，グラフィカルなデスクトップに慣れている．Windows のデスクトップのフォルダ管理作業の経験とフォルダ・プログラミング環境の利用作業の差が極めてわずかであるため，多くの人々がフォルダ・プログラミング環境を簡単に理解できると考える.

Q2 で A3 と回答した一人の学生は，Drag&Drop でメディアを扱う方法は簡単であったが，入出力データの間の関係を予期するのが難しいと指摘した．より詳細に学生たちから難しかった点に関する意見を集めた結果，それらは大きく 3 つの意見となった.

- エラーがわかりにくい．例えばフォルダ名（処理名）を間違えたときや
  エラーが発生したときに，その情報とその回復方法に付いて利用者に通

知される情報が無いか少ない.

- 必要な処理付フォルダを探す方法がない．必要な処理付フォルダ名が得られた後には，HELP を使うことができる．しかし，ある処理をしたいと考えた場合に必要となる処理付フォルダの名前がわからない場合には，望むべきタスクに辿りつけない.
- 入出力データの間の関係が難しいケースがある．特に，フォルダにおける条件付き実行の場合にどのような入力ファイルが必要になるのかなどがわかりにくい．また，処理付フォルダの名称が利用者の直感に合わない場合にもわかりにくくなる.

一方，フォルダ・プログラミング環境が提供する価値に関するアンケートは以下のようになった.

- 画像や音声の編集が簡単に行える.
- 大量のファイルを一度に処理できる.
- Drag&Drop で簡単かつ直感的にプログラミングできる.
- いろんな処理の組合せが柔軟にできる.

更に，実験運営者からは，処理部品実装の記述言語にかかわらず，フォルダ・プログラミング環境上で組合せることができる点の価値の指摘もあった．これは，従来の UNIX Bash 等のパイプの効果とも類似している.

4.4 メディア処理部品の統合に対する適用

先の小規模実験で，フォルダ・プログラミングがメディア処理にも相性が良いとのアンケート結果を得た．更に，我々は，マルチメディア処理への適応能力を確認するため，フォルダ・プログラミング環境 POLDER を高度に活用してマルチメディア統合処理システム虹雲ノート [147] を開発した．本節では，その虹雲ノートシステムの紹介と，そこで得られた使用経験を示す.

4.4.1 メディア処理への適用例

虹雲ノートはメディア処理部品を統合し活用するクライアント-サーバ (図 4.2) のアプリケーションである．虹雲ノートは Evernote [148] に似ている．Evernote はメディア処理として OCR 機能を持っている．同様に，虹雲ノートは OCR や名刺画像認識のような画

像認識技術，代表画像抽出のような高度映像処理技術，音声認識や音声合成技術などの多様なメディアに対する認識技術と高度処理技術を統合する．

虹雲ノート用のクライアントとして River と呼ぶクライアントも開発した．それらの認識技術や高度処理技術のメディア処理部品は，POLDER をベースとして Web フォルダとして実装したため，虹雲ノートは専用クライアントである River 以外の多様な WebDAV クライアントからもアクセス可能である．図 4.3 は独自クライアント River(PC 版) の画面を示している．図 4.4 は River(タブレット版) の画面である．それらは Evernote クライアントのユーザ・インタフェースや，Windows の Explorer の画面にも類似する．情報はノートという単位で管理される．図 4.5 はフリーソフトの WebDAV クライアントである TeamFile の画面を示している．これらの 3 つのクライアントは同一のユーザデータとフォルダ・プログラムを共有できる．

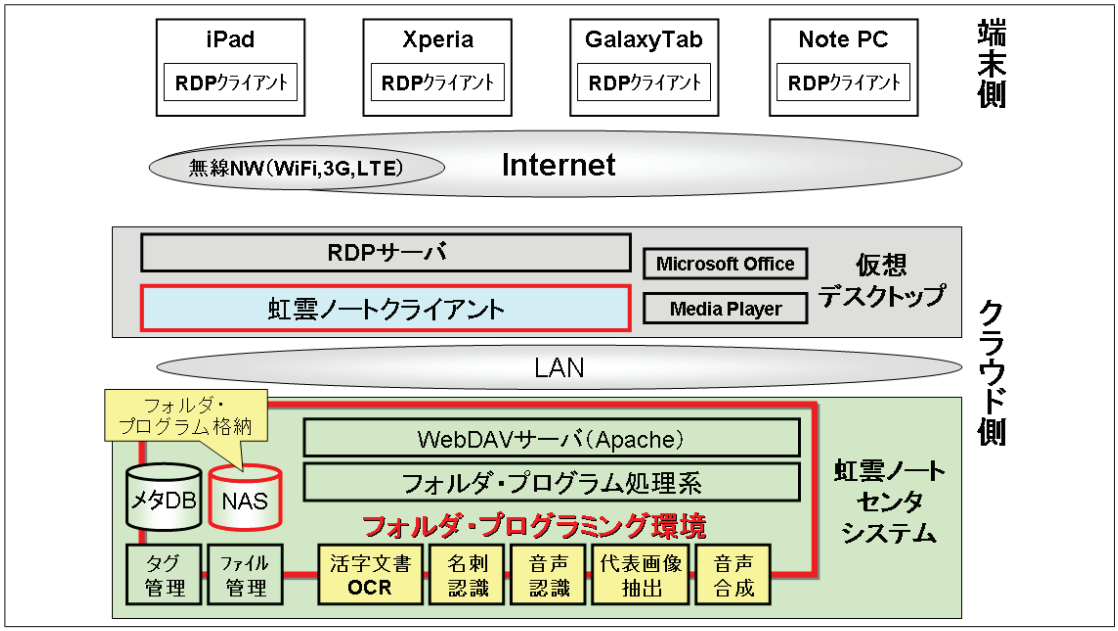


図 4.2: 虹雲ノートシステム・アーキテクチャ

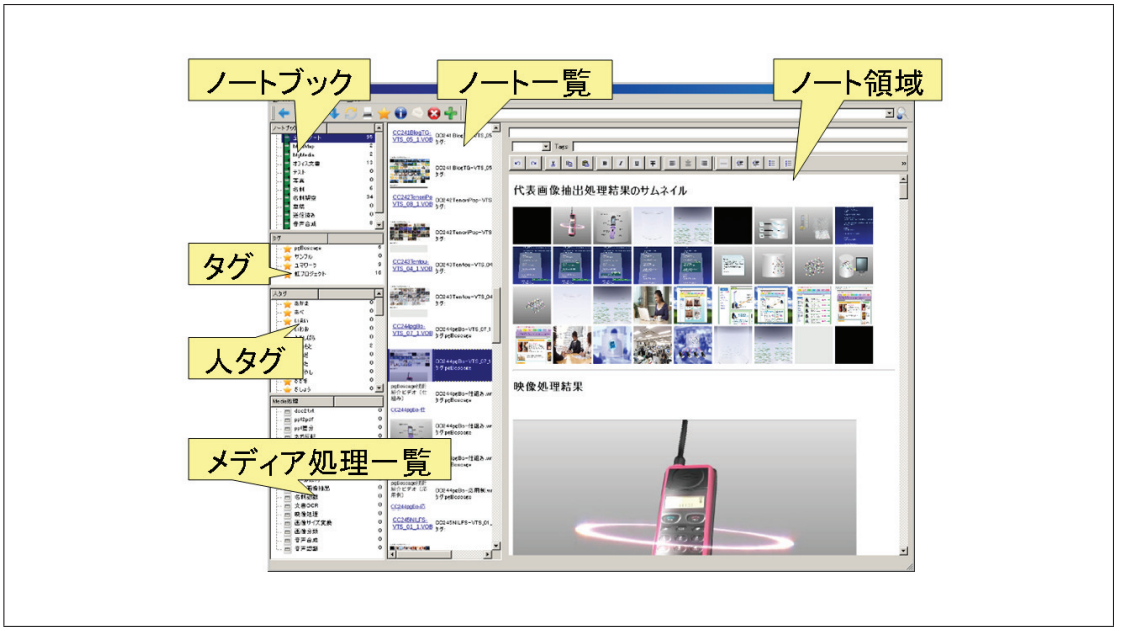


図 4.3: River クライアントの画面 (PC 版)

図 4.4: River クライアントの画面 (タブレット版)

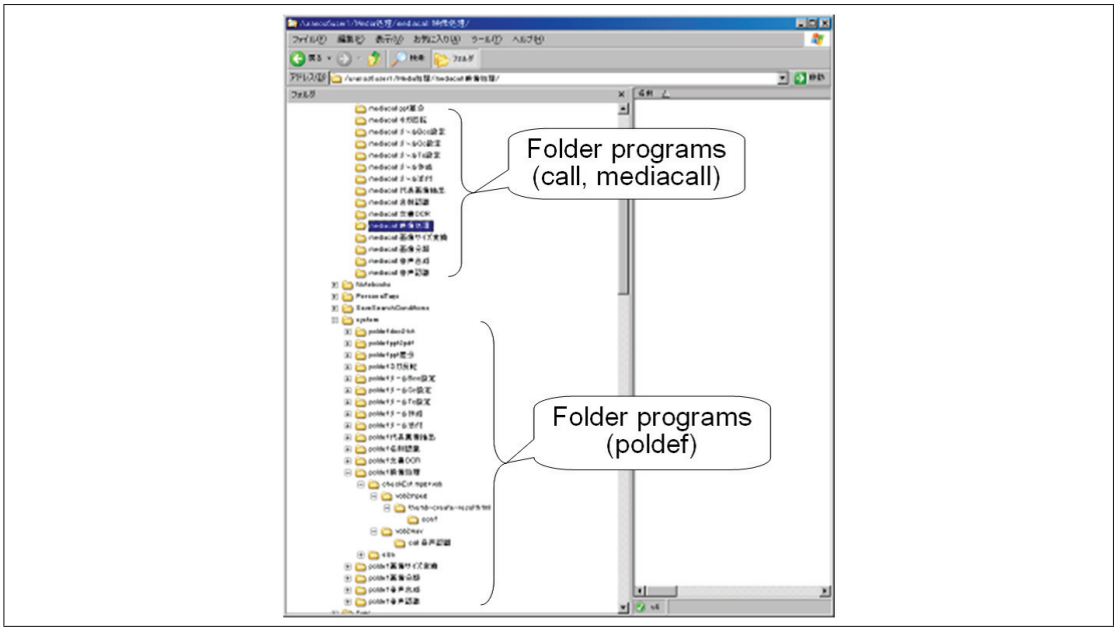


図 4.5: TeamFile クライアントの画面

POLDER 側で利用されたフォルダ・プログラムの例を図 4.6 に示す．インデントはフォルダの階層を示している．

```
call VideoProcessing
call extractTypicalFrame
call VoiceRecognition
poldef VideoProcessing
   decomposeVideoAudio
      matchFileExt mpg
         call extractTypicalFrame
      matchFileExt wav
         call VoiceRecognition
      else
poldef extractTypicalFrame
   matchFileExt mpg
      VideoToolKit TypicalFrame
   matchFileExt mov
      mov2mpg
         VideoToolKit TypicalFrame
   else
poldef VoiceRecognition
   matchFileExt wav
      stereo2mono 16Khz
         VoiceRex
            pass
            MeCab
               tag2db
   else

...
```

図 4.6: フォルダ・プログラムのメディア処理への適用例

ここでは以下の処理付フォルダが使われた．

- VideoToolKit TypicalFrame： 映像から代表画像群を抽出 [149]（図 4.3）
- VoiceRex： wav ファイルを音声認識しテキストファイルを作成 [150]
- Cralinet： テキストファイルから音声合成し wav ファイルを作成 [151]（図 4.7）

- matchFileExt:　ファイル拡張子によるファイルのフィルタリング
- stereo2mono 16Khz:　音声フォーマット変換
- mov2mpg:　映像フォーマットを mov から mpeg に変換
- pass:　入力ファイルの複製出力
- MeCab:　テキストからの形態素の抽出
- tag2db:　形態素抽出結果のデータベースへの登録
- 文書 OCR:　画像内文書の OCR [152]
- 名刺認識:　名刺画像の認識 [153]（図 4.8）

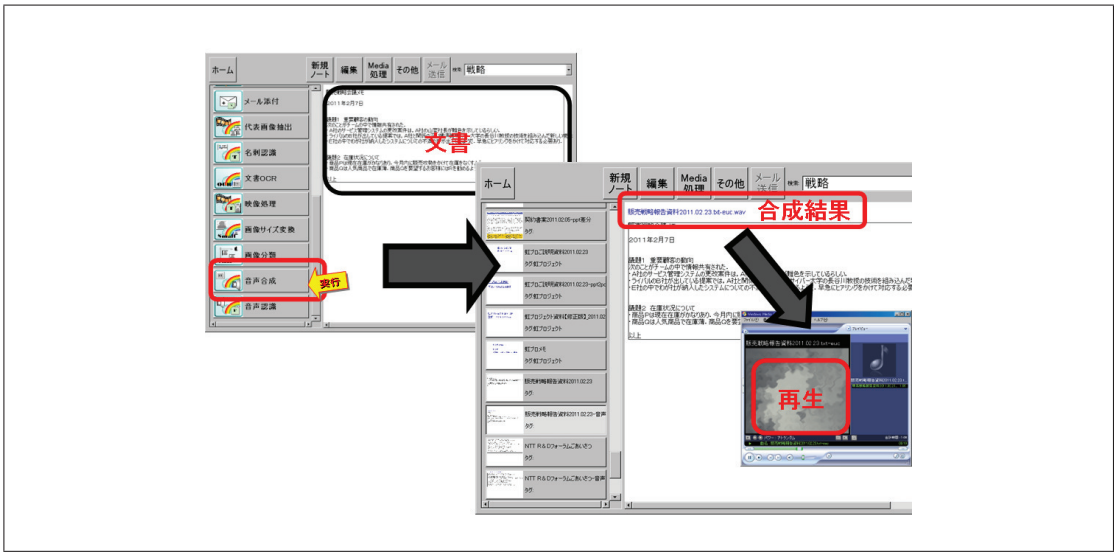


図 4.7: 音声合成の例

図 4.8: 名刺認識の例

River(PC 版) では，メディア処理を表すフォルダアイコン上にファイルを Drag&Drop すると，その処理が実行されその結果がノートとして生成される．TeamFile や Windows Explorer では，ファイルをフォルダに Drag&Drop すると，実行結果はそのフォルダ内に生成される．River(タブレット版) では，タッチすることで画面が遷移していく[1]．

4.5 メディア処理部品統合環境への適合性に関する分析

4.3 節では学生を対象とした簡易な実験の結果とそれに伴うアンケートを示し，4.4 節では虹雲ノートシステムの設計と実装によるフォルダ・プログラミング環境の活用経験を示した．本節では，それらを踏まえ，多様なメディア処理のマッシュアップ環境としての 5 つの要件の観点からフォルダ・プログラミング環境を分析する．

(M1) マルチメディア対応性

要件 (M1) は，マルチメディアつまり多様なデータが利用できることである．

処理途中でのデータの型の変更が可能: POLDER は画像, 音声，映像などの多様な型のデータファイルを扱うことができる．それらは処理の途中でデータ型が変化しても問題ない．今回の利用経験で，特定の入力フォーマットを対象としたメディア高度処理の処理付フォルダに対応して，入力コンテンツのフォーマットを変換・前処理するときに活用さ

[1] タブレット版では，Drag&Drop の替わりに，右フレームに対象を表示し，左フレームで処理を選択する．

れた．例えば，音声認識する処理付フォルダ VoiceRex は 16KHz モノラル音声を要求するので，処理付フォルダ stereo2mono 16KHz が前処理のために使われた．同様に, 代表画像を抽出する処理付フォルダ VideoToolKit は MPEG-1 か MPEG-2 の映像を要求するので，処理付フォルダ mov2mpg を使って入力ファイルを MPEG の映像フォーマットに変換した．

特定のデータ型のみに対する処理が可能:　特定のファイル拡張子に応じた処理を行う場合の条件分岐として matchFileExt は良く用いられた．例えばこれによって，mpeg2 ファイルの場合と，wav ファイルの場合の処理の振り分けが行われた．また想定されていないファイルタイプが処理対象となった場合に対応して，処理付フォルダ「else」を使って，処理を振り分けることも多く行われた．

入出力処理のデータ参照が容易:　フォルダ・プログラミング環境は，多様なデータを扱える Viewer, Player, Editor などを含むデスクトップ環境をベースとするため，マルチメディアデータの表示が簡単である．POLDER が扱える多様なデータには，テキスト，構造化テキスト，オフィス作業で用いられる Microsoft Word, PowerPoint, Excel なども含まれた．

このように，フォルダ・プログラミング環境の利用者が，多様なデータに対応して処理を活用することができた．

一方，処理部品を提供する側からの分析も試みる．

入力数と出力数の自由度:　処理付フォルダが受け付けることができる入出力ファイルの数の柔軟性の活用も確認した．フォーマット変換処理付フォルダは通常 1 入力 1 出力である．DB 登録用フォルダにファイルを Drop したときには，1 入力に対して 0 出力である．顔画像抽出と代表画像抽出は 1 つの入力から複数の出力を生成する．しかし，処理付フォルダが複数の入力を使う場合にエレガントな方法は無い．ただし, POLDER の仕様はデータフォルダと処理付フォルダの共存を許容し，データフォルダを作成する処理付フォルダの作成を妨げない．そこで今回の適用例では，例えば 2 つの PowerPoint ファイルの差分を見つける処理付フォルダが実装された際の処理方法として，最初のデータファイルが届いたときには，データフォルダを作成してその中にデータファイルを保持し，2 番目のデータファイルが届くとデータフォルダ内のファイルとの差分を抽出するという処理付フォルダを実装した．これによって，複数（固定個数）の入力から 1 つの出力を作ることにも対応できることが確認できた．

供給者の開発負担の削減:　メディア処理実体の供給者が処理部品を開発する際，利用者が，適切な変換処理付フォルダを使って入力ファイルを必要なファイル型に変換することを前提とすると，供給者はそれらの多様なデータ型に対応する部分のコード作成が不要になる．これはメディア処理実体の供給者にとって，本質的ではない部分のその開発負担（費用，期間，難易度等）を削減してくれることになる．

(M2) Web API 対応性

要件 (M2) は，サーバやクラウド側で提供される Web サービスを，クライアントから Web を経由して簡単に利用できることである．

Web 経由での利用が容易:　虹雲ノートシステムやその基盤の POLDER の実装においては，サーバ側は Apache をベースに実装され，完全に Web に対応した．また，クライアントとして，Microsoft Internet Explorer や TeamFile，及び Java で作った独自クライアント River などのクライアントから虹雲ノートシステムへのアクセスが可能であった．WebDAV によるネットワーク・フォルダをベースとするため，Web 経由での処理であってもネットワークの存在をほぼ意識する必要がなく，利用者によって簡便に利用することが可能であった．

Web 経由での異種環境部品接続性：　メディア処理は，特定言語や特定 OS のみで供給されるケースがある．

今回のケースでは多様なメディア処理部品は，C/C++，Perl，Java といった多様な言語で記述されていた．POLDER では，処理部品の実体はまずコマンドとして供給され，それらを処理実体として Perl もしくは Bash のテンプレートを使ってラッピングすることで実現される．これにより実装プログラミング言語の違いを無視してメディア処理部品を連携することが可能であった．

また，異なる OS 上に構築されたメディア処理部品を連携することも可能であった．今回のケースにおいて OS 違いは，Linux 64bit を基本サーバ環境とする中で，子環境 1 の Linux 32 bit 上で提供される音声合成処理と，子環境 2 の Microsoft Windows XP 32bit で提供される PowerPonit 処理が存在した．これらを図 3.16 の処理付フォルダ実行環境 3 に示すような簡易接続で接続した．各子環境の処理部品は，基本サーバ環境の処理実体からの処理付フォルダとして呼び出され実行された．

このように，利用者から処理実体のプログラミング言語や OS の違いが隠蔽されることは，多様なメディア処理を組合せたい利用者にとってはもちろん有用であるし，更に開発供給者にとっても，その処理部品の活用範囲が広がるため，非常に有用である．

(M3) 単機能利用の容易性

要件 (M3) は，任意の処理を 1 つだけ選んだとき，それを単機能として簡単に実行し結果を得ることができることである．

単機能実行:　フォルダ・プログラミング環境において，その最小単位である処理付フォルダは，それ単独で実行可能である．よって，単機能の実行を実現していた．

学習量の負担：　処理付フォルダの利用は，従来のフォルダ管理とほぼ同様の知識で利用可能である．3.7 節の実験においても，実験参加者から Drag&Drop は直感的でわかりやすいという回答を得ていた．また，多様なメディアに対する確認処理でも Viewer,Player,Editor について新たな学習負担が発生しない．マルチメディアデータを処理するとき，この容易性は特に重要であると考える．

作業量の負担：　作業量の面からみても，通常のデスクトップ操作と同様であるため，利用者にとっての負担は確認されなかった．

このように処理付フォルダの概念と操作体系は，従来のフォルダの素直な延長線上に存在し，従来のフォルダ・インタフェースと差異がほとんどないため，十分に少ない学習量・作業量で利用が可能になる．処理付フォルダを使えば，フォルダを選び，デスクトップ上のファイルを Drag&Drop することでその処理を実行し，フォルダ内の結果ファイルに対してダブルクリックで Viewer,Player,Editor を起動してその処理結果を確認する，という 3 ステップだけで一連の体験が完了する点で，初心者が十分少ない知識と作業量で処理を作成し実行して結果を得ることが可能である点の効果は大きい．

キャッシュの問題：　今回の実験で利用したクライアント群では，問題は発生しなかったが，他の WebDAV クライアントでは重大な問題が起こる場合がある．その問題は，クライアント上で POLDER に存在しないファイルを表示するということである．そのクライアントはユーザを混乱させる．処理付フォルダの生成する出力ファイル名が入力ファイル名と異なっているとき，この問題は発生する．正しい処理結果を見るために，ユーザはクライアントでキャッシュをクリアしなければならない．

(M4) 応用利用の容易性

要件 (M4) は，状況に応じた処理の柔軟な利用が簡単であることである．この要件は，我々の目指すエンドユーザ向けプログラミングの要件となる．特に，部品合成容易性，手動部品接続性，大量適用容易性，高度利用への移行容易性が簡単であることが重要である．

部品合成容易性: 処理付フォルダの階層構造を作ることで，それらを組合せて連続して利用することが簡単にできた．

虹雲ノートの実装においても多数の部品接続が活用されていた．また，学生を対象とした実験においても，処理付フォルダの階層構成を容易に作成していた．

手動部品接続性: フォルダ・プログラミング環境は，デスクトップ環境と統合されているため，各種メディア編集アプリケーションで処理した結果ファイルをフォルダ・プログラムで処理することや，その逆に，フォルダ・プログラムでの処理結果を編集アプリケーションで加工するなどの，手動での組合せが自由にできる．

例えば，認識率が低かった名刺画像は，画像編集ソフトウェアを使った輝度と位置の補正の手動編集と組合せたことで正しく認識できた．

大量適用容易性: 通常のデスクトップのユーザ・インタフェースである複数のファイルを選択して一度に Drag&Drop する仕組みを使って，大量のデータに対して同じ処理を簡単に実行することができる．

例えば，大量の名刺画像の認識処理や，大量の映像に対するタグ付け処理は，複数のファイルを処理付フォルダに一度に Drag&Drop する操作で簡単に実行できた．

高度利用への移行容易性: 単一機能だけを利用する際の必要知識と，部品を合成して利用する際の必要知識，及び，大量データへの作業を行う際の必要知識が，全て通常のデスクトップ上のフォルダ管理のユーザ・インタフェースで構成されており，利用者は単一機能の利用状態から，部品を合成して大量データへの適用を行う高度利用の状態へと簡単に移行できる．

(M5) 管理の容易性

要件 (M5) は，管理的側面の容易性であり，合成処理保管の容易性，処理発見の容易性，合成処理変更の容易性の 3 つが必要である．

合成処理保管の容易性: フォルダ・プログラミング環境では，プログラムの作成がフォルダの作成になるため，常に作成したプログラムがフォルダ階層として保存された状態となる．つまり，利用者は作成した処理の保管操作を意識しなくて良い．これはエディタなどでプログラム・ファイル作成する場合に比べると保存の失敗が無く，また，プログラム・ファイル名の付与も避けられるので，極めて簡単である．

虹雲ノートでも，このプログラムの永続性を利用している．そのため，他のクライアントから参照しても常にプログラムがフォルダとして存在している．

処理発見の容易性:　フォルダ・プログラミング環境では，保存されたフォルダに対する，フォルダ名検索の仕組みが最も基本的な手法となる．

既に用意されている処理付フォルダについては，その一覧を Web ページとして用意し，クリックによる選択から当該処理付フォルダをオープンまで行うブラウザの仕組みを利用することも可能である．

例えばブラウザの Web Folder Behaviors [154] を利用すれば，検索して見つけた機能名称のリンクをクリックすると，即，その機能名のフォルダが表示されるようになる．これらは従来のプログラミング環境を超えるものではないが，同等のものは存在していると考えることができる．

フォルダ・プログラミング環境においてヘルプ機能は実装されている．しかし，フォルダ名が不明の場合には，求める機能を発見する機能としては十分ではなかった．処理付フォルダを捜し求めるより進化したメカニズムが，将来必要である．

例えば，クラウドパワーを活用する方法として，各機能名のフォルダを横に並べ，並行実行を利用してサンプル入力ファイルを処理させ，その後，各フォルダ中に出力として期待するファイルが存在するかを探索することで，目的の機能の処理付フォルダを探し出す方法なども不可能ではない．

虹雲ノートにおいては，処理の発見については，少し異なる方法を用いた．虹雲ノート・サーバでは，PostgreSQL サーバが動作しており，キーワードと対象フォルダの対応関係を管理していた．名刺認識処理が実行されると，その一連の処理の中で，その人物の姓と名を各々キーワードとして，その投入名刺画像が格納されるフォルダとの対応関係を DB 中に蓄積するとともに，姓もしくは名を引数としてフォルダ名に持つ検索処理付フォルダ「search」を生成した．利用者は姓や名といった引数を目印に search フォルダを決定し，そこに検索イベント発生のための任意のファイルを Drop すると，このように処理付フォルダ内の処理実体で，処理対象を DB に登録する使い方の応用として，新しい処理付フォルダの関連情報を登録することによって処理の発見を支援することも考えられる．

合成処理変更の容易性:　最後の合成処理の変更ついては，フォルダ・プログラミング環境において，他者が作成したフォルダ・プログラムを利用するためには，例えば Public なエリアにフォルダ階層をコピーしてもらいそれを自分専用 Private エリアに再コピーするだけであり，容易である．そのフォルダ・プログラム階層の一部を変更することやフォルダ階層の一部だけを別の場所にコピーすることも，従来のフォルダ管理に慣れた PC 利用者であれば容易である．交換対象の処理付フォルダ部品を単独で実行して確認することも単機能だけの実行を可能にするフォルダ・プログラミング環境であれば容易である．

虹雲ノートの実装においては，mpeg2 フォーマット用として用意されたフォルダ・プログラムに対して，mov フォーマットに対応する機能追加が発生した．この際に，処理付フォ

ルダ「mov2mpg」を容易し，既存の mpeg2 対応のフォルダ階層を複製することで，簡単に機能追加に対応できた．

4.6 おわりに

本章では，フォルダ・プログラミング環境をメディア処理部品の統合に利用することの効果について，学生 10 名を対象としたフォルダ・プログラミング環境を使って画像のファイルを処理する簡易な実験を紹介した．そこで得られたアンケート結果から，フォルダ・プログラミングのコンセプトが簡単に理解できることが確認できた．フォルダ・プログラミング環境は，エンドユーザに対する障害は低いと考えられる．それは，日常的に PC を利用している利用者は，デスクトップ環境に慣れており，それらのフォルダ管理作業の経験とフォルダ・プログラミング環境の利用作業の差が極めてわずかであるため，多くの人々にとってフォルダ・プログラミング環境を簡単に理解できると考えられる．

更に，フォルダ・プログラミング環境を応用して開発した虹雲ノートシステムを紹介した．

これらの利用者実験とメディア処理統合の実践を通して得られた使用経験をもとに，フォルダ・プログラミング環境が，マルチメディア処理のマッシュアップ環境としての 5 つの要件，(M1) マルチメディア対応性，(M2)Web API 対応性，(M3) 単機能利用の容易性，(M4) 応用利用の容易性，(M5) 管理の容易性，に対して，高い適合性を示すことが確認できた．

第5章　フォルダ・プログラミング環境のエンドユーザ・インタフェースの評価

5.1　はじめに

フォルダ・プログラミング環境は，メディア処理部品を GUI 上でマッシュアップできる環境であり，既存のメディア処理部品を組合せることや，その処理を大量のデータに適用することが簡単に行える．我々は，フォルダ・プログラミング環境のフォルダ・インタフェースの効果を分析することを目的に，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た．比較対象のテキスト・インタフェースの実現には，フォルダ・プログラム開発支援機能の 1 つであるフォルダ・プログラムのテキスト表現 (fs4) を活用している．まず我々はそのコンパイラと実行処理系を実装した．次に 36 名の学生に複数の課題を両方の環境で解いてもらった際の作業時間とアンケートを収集しユーザのスキルレベルや，課題の難易度などからの分析を行う．

更に，他のビジュアル・プログラミング環境としてオープンに配布されている Pure Data とフォルダ・プログラミング環境 POLDER との数名の技術者を対象としたユーザビリティの比較実験を行い，実験参加者へのアンケートを収集し，Nielsen のユーザビリティの原則を使って分析を行う．

5.2　フォルダ UI とテキスト UI の比較実験

5.2.1　フォルダ・プログラミング環境のテキスト UI の設計と実装

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．

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 5.1 に示す．

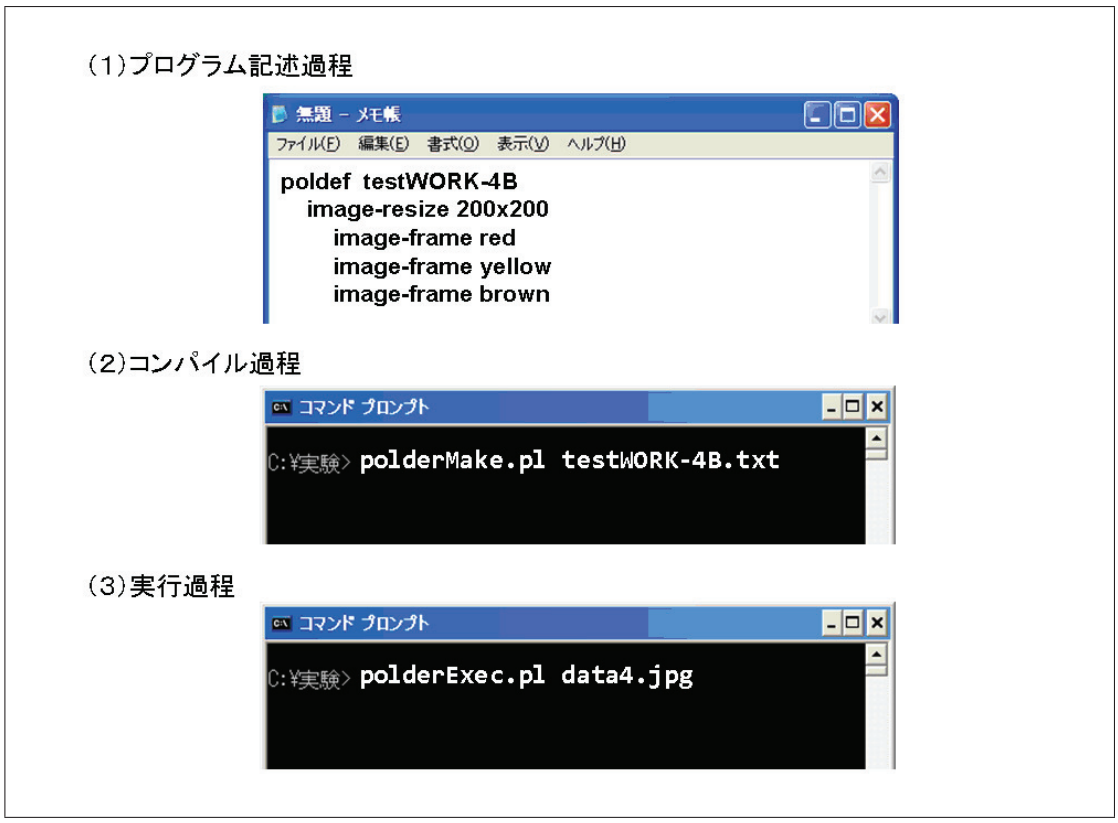


図 5.1: POLDER のテキスト・インタフェースの例

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．

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（例えば半角空白 2 個）にする．図 5.1 の最上段は，具体的なプログラムの記述例であり，フォルダとして考えれば 3 階層のプログラムとなっている．

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，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 5.2 に示す．

(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 が無ければ，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 5.2: テキスト版プログラムのコンパイル過程

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ーとしての分かり易さを重視して，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．

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 5.3 の実行ステップとなる．

(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 に存在する全てのファイルを 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 5.3: コンパイルされたプログラムの実行過程

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.

5.2.2 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．

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 は一人の実験参加者が専有し，更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 つの方法で実験が出来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 [142] をプラグインしたものを使う [1]．

【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，まずメモ帳等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．

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. 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

フォルダ・インタフェースについては，以下のようなスライドをもとに普通のフォルダと処理付フォルダ（図 5.4），処理の連鎖と応用（図 5.5），並行処理と応用（図 5.6），複数ファイルへの一括処理（図 5.7），抽象化・サブルーチン（図 5.8）の説明を実施した．

[1] 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．

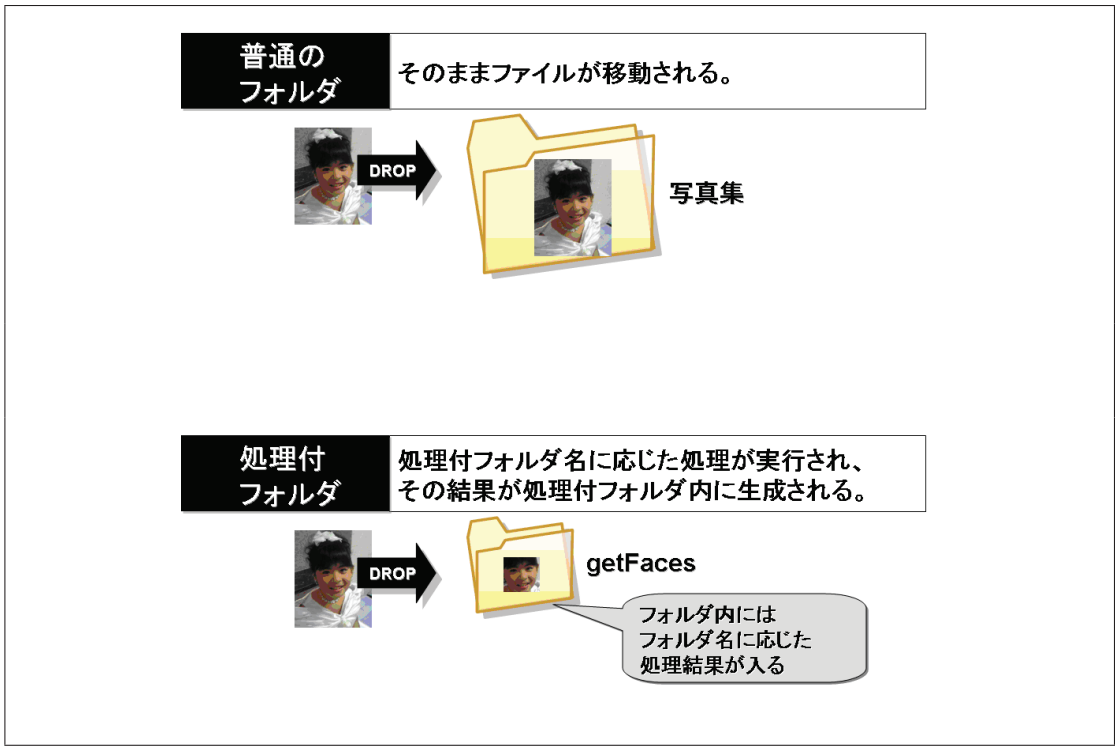


図 5.4: 普通のフォルダと処理付フォルダの動作

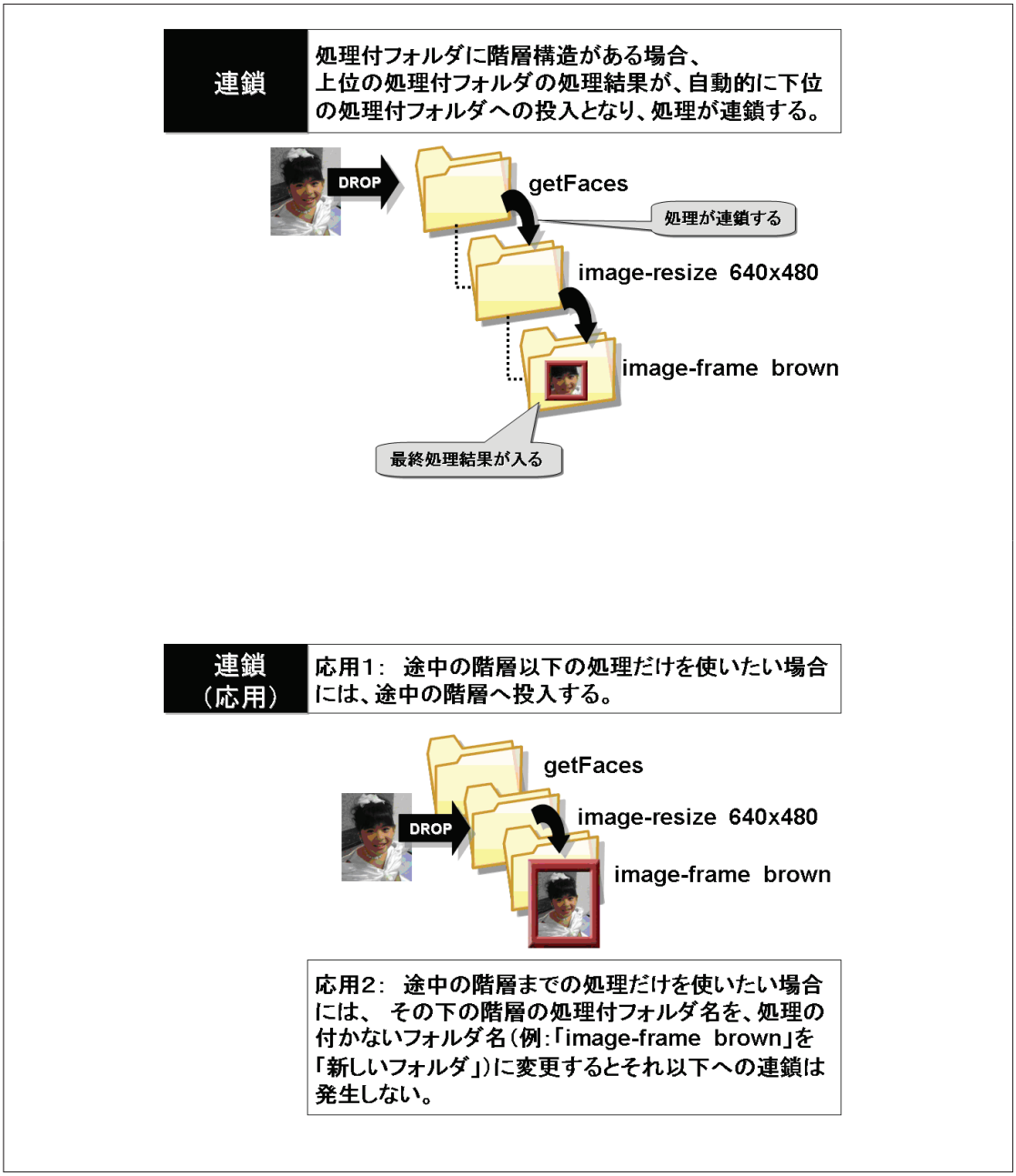


図 5.5: 処理の連鎖とその応用動作

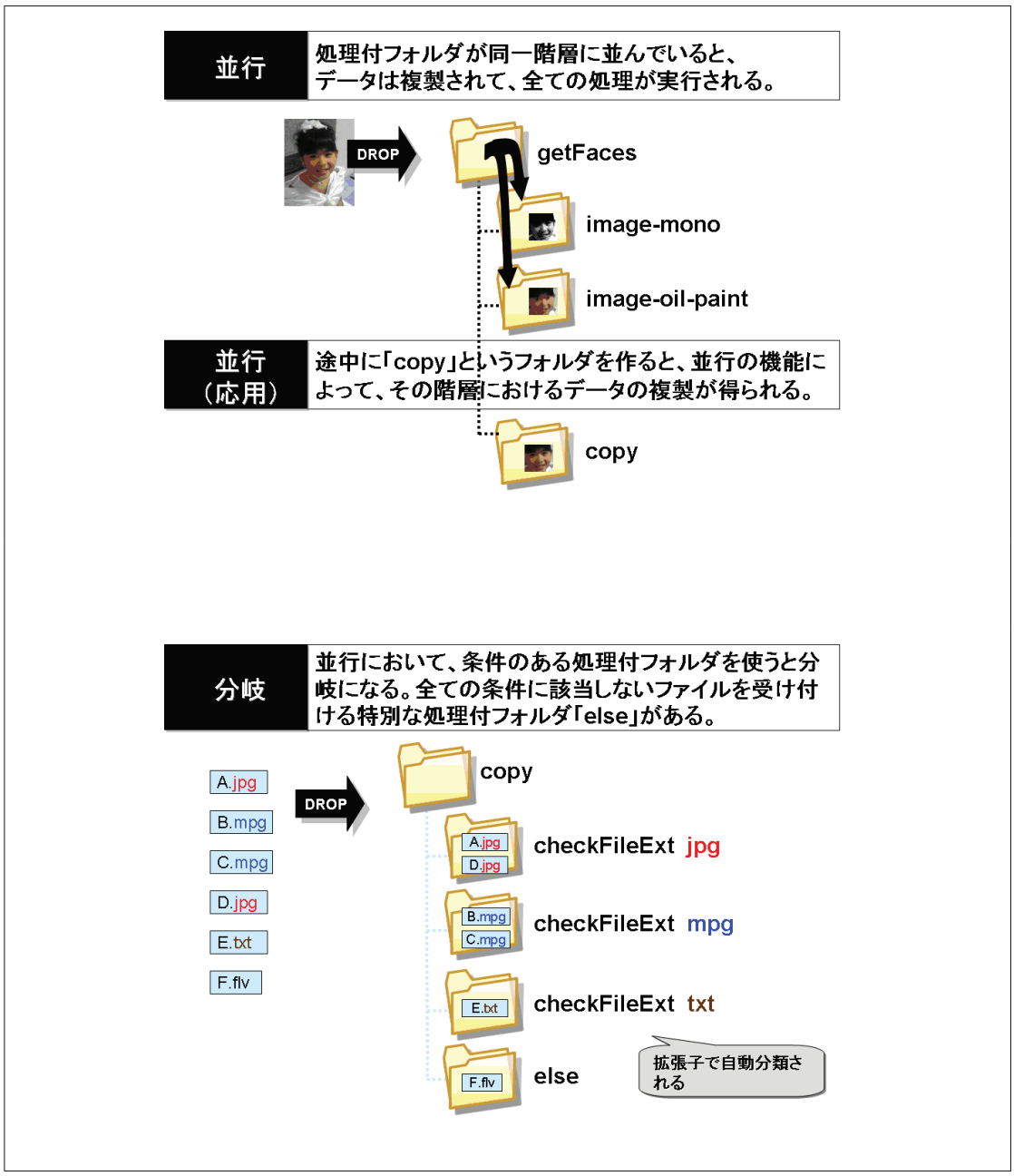


図 5.6: 並行処理とその応用動作

図 5.7: 複数ファイルへの一括処理動作

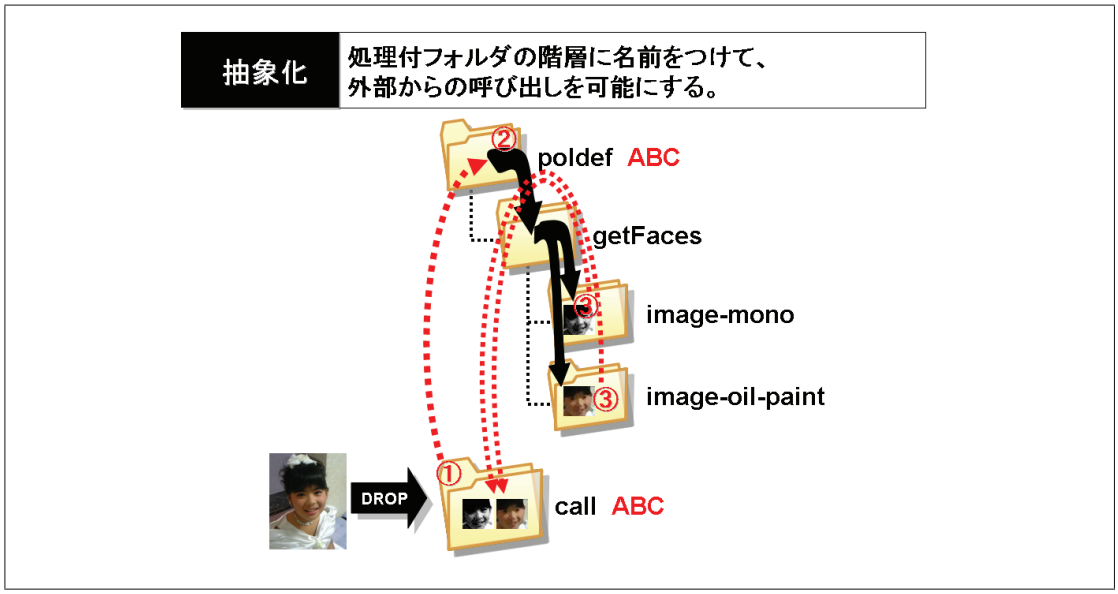


図 5.8: サブルーチンの動作

実験では，一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わ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，各々が概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 CopyrightNTT」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．全て半角文字です．

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 5.9 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．

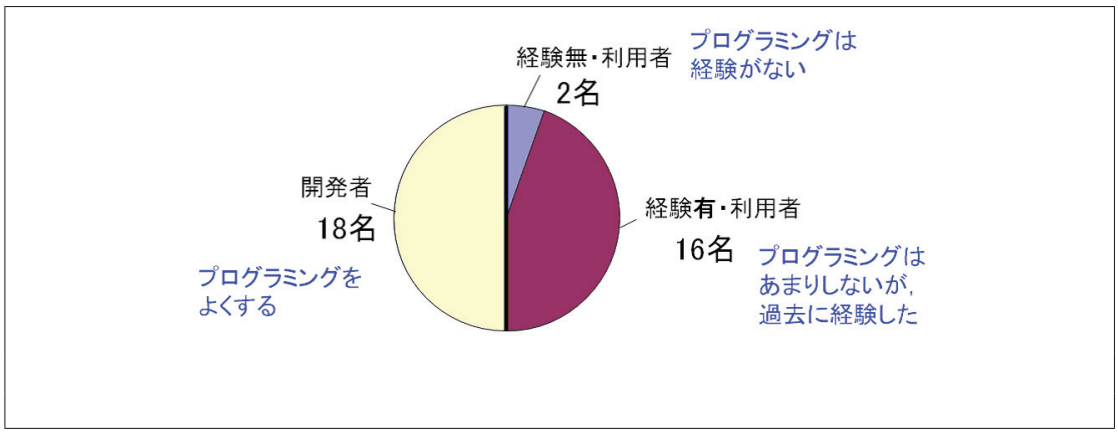


図 5.9: 実験参加者の内訳

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．

- 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 300x300 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 200x200 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「250x250 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，「250x250 の画像に赤いフレームが付いた画像」を得られて，常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 は単純に処理付フォルダを実行するタスクで，処理付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 は更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 5.2.2 節で述べた方法 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，更に，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，全ての課題を完了した．

5.2.3 実験結果

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 5.10 に示す．縦軸は作業時間を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．

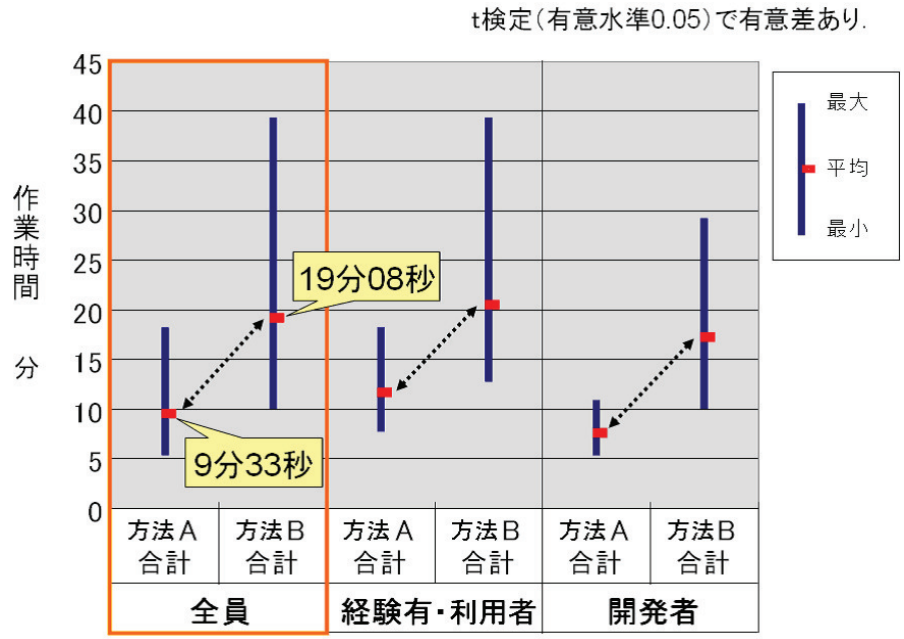


図 5.10: 方法別の作業時間の比較

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言える．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・利用者及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 及び課題 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．編集時間は処理付フォルダ名の確認時間や入力速度等によってバラツキがあった．5.3 節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 5.11 に，開発者に対する同じ結果を図 5.12 に示す．

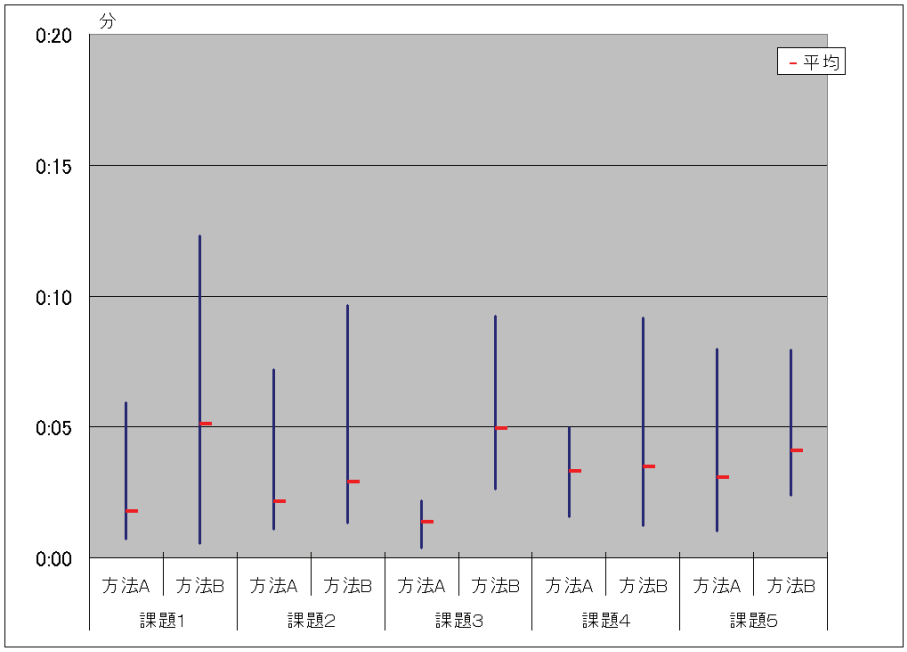


図 5.11: 課題別・方法別の作業時間の比較 (経験有・利用者)

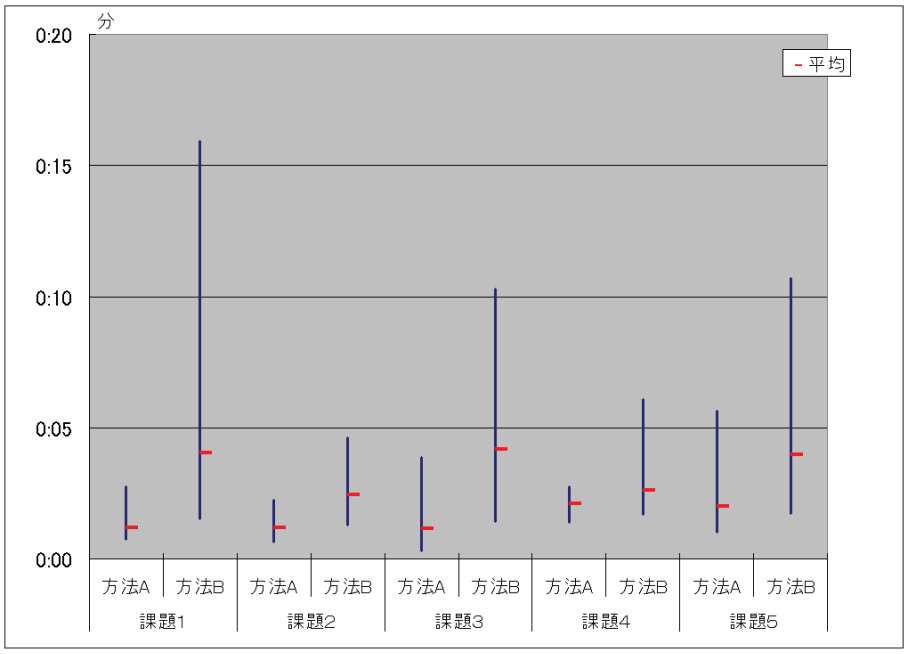


図 5.12: 課題別・方法別の作業時間の比較（開発者）

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，…，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の平均作業時間が短い結果となっている．
- 単純な処理付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言い切れない．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，概ねフォルダ・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．

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 5.13 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．

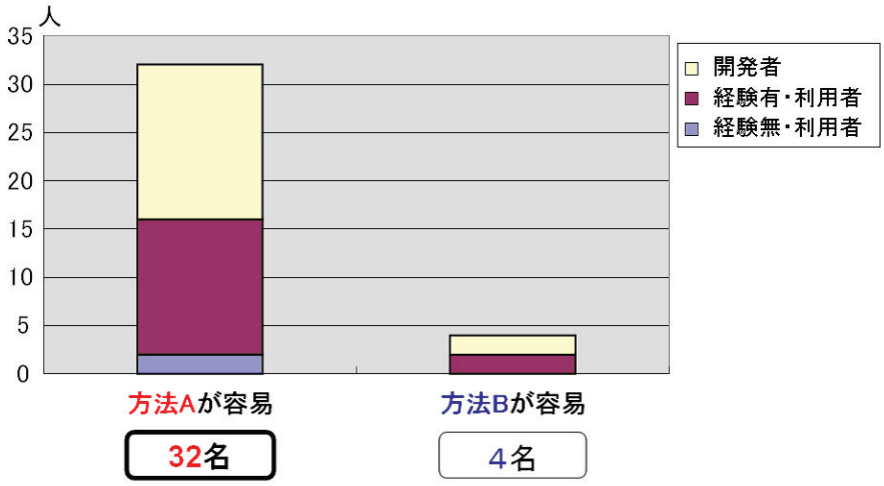


図 5.13: 実験後アンケート（容易な方法）

更に詳細に,「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 5.14 に示す.

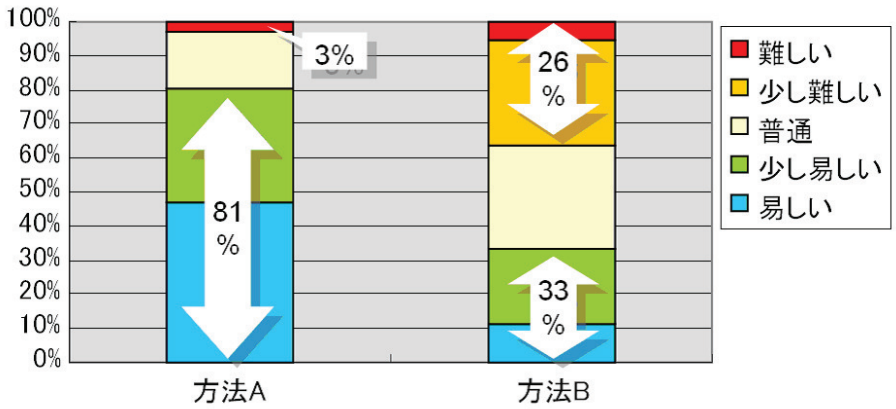


図 5.14: 実験後アンケート（容易さの程度）

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% に留まっており，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% に留まったのに対し，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．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．

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．本節では，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，一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) 視覚的でわ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.「一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．この点は極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，全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り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取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上手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わかった．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わかった．

5.3 フォルダ UI と他のグラフィカル UI との比較実験

5.3.1 ビジュアル・プログラミング環境 Pure Data

Pure Data [83] [85] [155] [156] 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，ナンバー，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．

Pure Data は長い歴史とコミュニティの実績を持つため既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 [2] [157] 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．

本節では，POLDER と Pure Data との比較について，Nielsen によるユーザビリティ原則 [13] に従った評価を試みる．ユーザビリティの評価においては数名（5 人程度，少なくとも 3 人）の評価者がいれば妥当な結果が得られるとあるため，5.2 節に比べ小規模な人数での実験とした．

5.3.2 実験

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．5.2 節での学生に比べて，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）の説明は必要無かった．

[2] Graphics Environment for Multimedia

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 5.15 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．

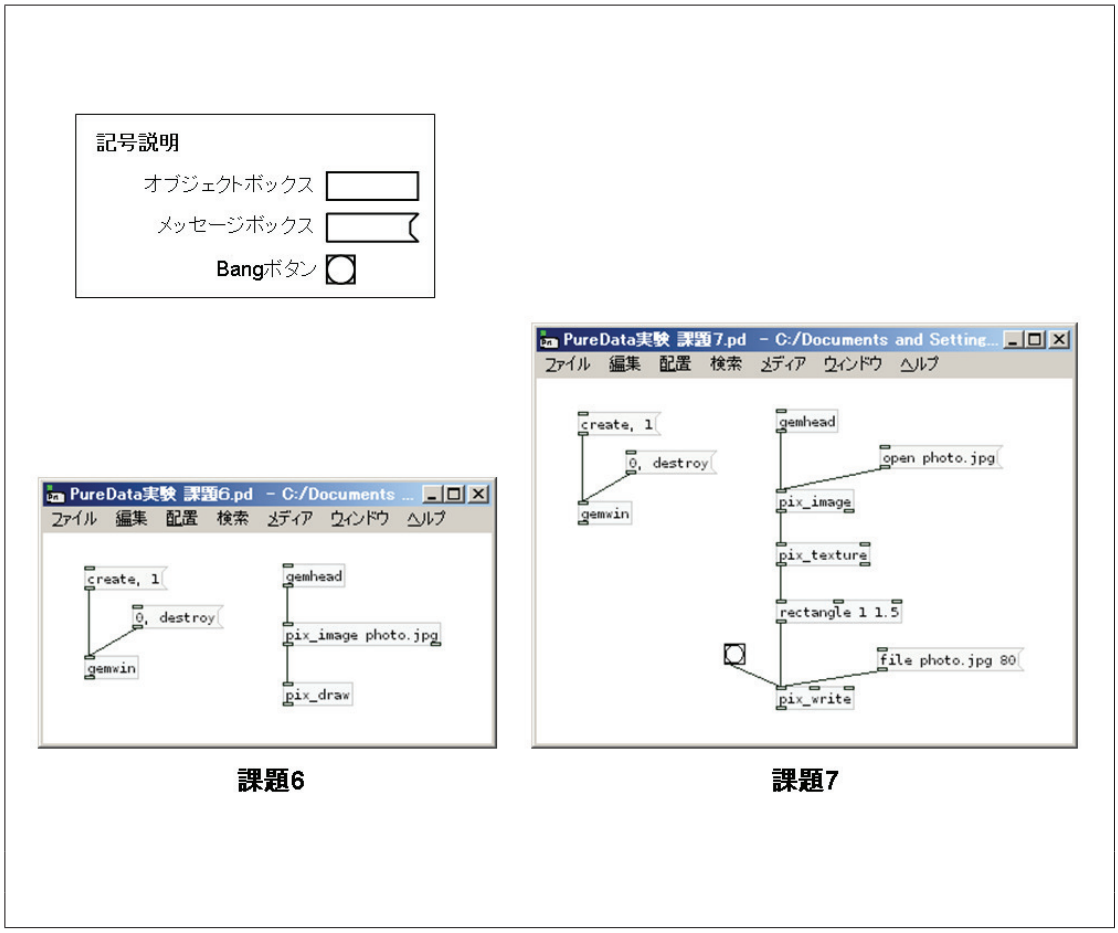


図 5.15: Pure Data によるプログラム作成の課題

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．

【課題 6】図 5.15 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 5.15 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)，及び，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，更に 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．

5.3.3 実験結果

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，ファイルの読込み・書出し処理) 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言える．

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わからない，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)
- 文字入力ミス．(3 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．図 5.16 は，Pure Data と POLDER のフォルダ・インタフェースのエラー回数の比較結果[3]を示している．横軸 a〜f は，各実験参加者に相当している．このように全体的に，Pure Data でのエラー発生が多いという結果であった．

[3]なお，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，ファイル名の入力については，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．

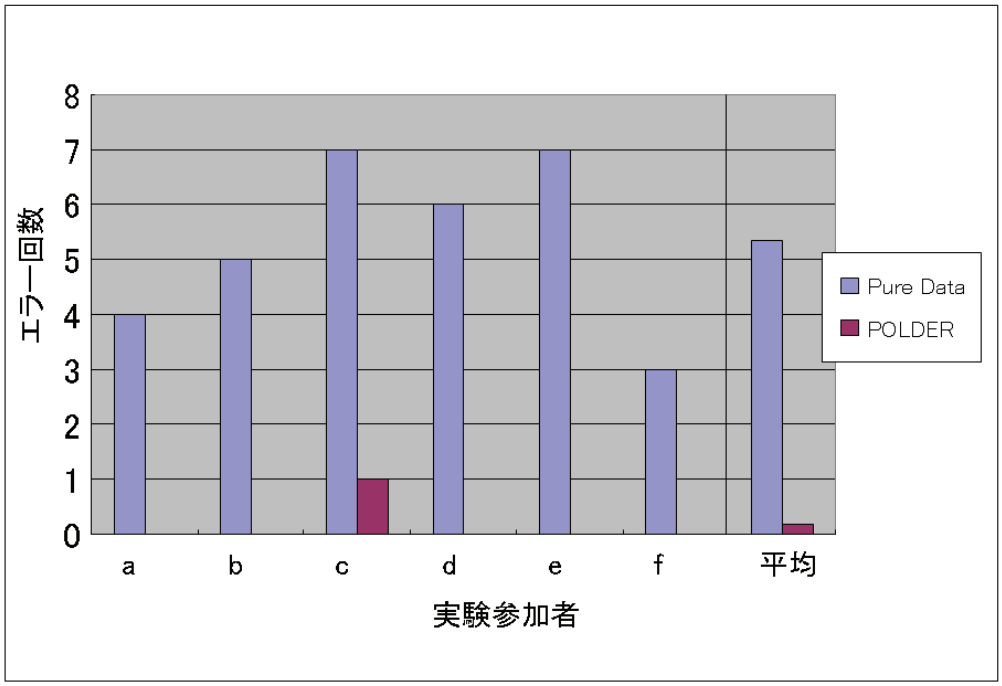


図 5.16: エラー回数の比較 (Pure Data と POLDER のフォルダ・インタフェース)

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．

(uh1) 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．

(uh2) ユーザの言葉を使う.

Pure Data は，DSL[4]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．

[4]Domain Specific Language

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等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 [158] 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 5.6 で紹介したような条件付きの処理付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．

(uh3) 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り替え，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，(uh1)(uh2) 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．

(uh4) 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．一貫性については更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．

(uh5) 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．更に，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．

(uh6) 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, 処理付フォルダの実行についての UNDO は提供されておらず，5.2.3 節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，(uh5) で述べた進捗率表示などとともに，処理付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．

(uh7) 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 では常に処理付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．

(uh8) 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易さは必ずしも十分とは言えない．

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等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．

(uh9) エラーを防ぐ.

Pure Data では，(uh5) 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，(uh3) 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，常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フォルダ名の誤入力を予防する仕組みが必要である．

(uh10) 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，更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 では，Explorer についてのヘルプは不要であった．また POLDER には，処理付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 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，モードのスイッチ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言える．また，POLDER のドキュメント等の整備も十分ではない．

5.4 おわりに

本章では，フォルダ・プログラミング環境のユーザ・インタフェースの効果を分析することを目的に，2 つの比較実験を行った．

まず，コードを記述する従来型のテキスト・インタフェースとフォルダ・インタフェースとの比較の利用者実験を行った．比較対象のテキスト・インタフェースの環境として，フォルダ・プログラム開発支援機能の 1 つであるフォルダ・プログラムのテキスト表現 (fs4) を活用したコンパイラと実行処理系を設計し実装した．次に 36 名の学生に複数の課題を両方の環境で解いてもらった．学生はプログラミングをよくする**開発者**と，そうでない**利用者**に分け，利用者は更に，プログラミング経験の有無により**経験有・利用者**と**経験無・利用者**に分けた．その結果，作業時間を比較したところ，経験有・利用者に効果があり，更に，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が短縮するというケースが確認できた．また，いくつかの課題間の分析において，単純に処理付フォルダを実行する際の取り組み易さの点と，複数の入力ファイルに対して処理を実行する点に大きな効果があり，処理の連鎖を使ったプログラム作成でも効果が見られた．更に，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単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，その成長の中間ステップを提供し障壁を下げる効果が期待できる．

2 つ目の比較実験として，他のビジュアル・プログラミング環境としてオープンに配布されている Pure Data とフォルダ・プログラミング環境 POLDER とのユーザビリティを比較の利用者実験を行った．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いえる．グラフィカルな UI の中でも，日常的に利用しているフォルダの UI が優位であることが確認できた．

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，処理が長い場合のシステム状態のフィードバック（現在はプログレスバーの表示のみ），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ことが判明した．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未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等も十分ではないという分析結果となった．

本論文の第一の要件であるエンドユーザからの受け入れられるかという観点で見れば，5.2.3 節の実験結果から，経験あり・利用者において，従来のテキスト UI のプログラミング環境に比べて，フォルダ UI を使うプログラミング環境の有効性が非常に高いことが確認できた．更に，方法別の分析結果において，課題 3 で示した多数のデータへの一括処理について，フォルダ UI の作業時間が顕著に短縮できており，DIY で重要な大量適用シーンにおいて，有効性が高いことも確認できた．また，実験参加者からのアンケート回答からも，複数ファイルを一括して処理できるので簡単，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，コンパイルやその実行の手間が不要なので簡単，といった，フォルダ・プログラミング環境の利点が十分に伝わり支持されていることも確認できた．

第6章　考察

6.1　本研究の2つの要件の解決

本研究では，幅広いPC利用者がマルチメディア処理のマッシュアップをDIY で簡便に行えるようにするためのメディア処理部品の統合環境を実現することを狙いとしていた．その環境は，次の2つの要件を満たし両立する必要があった．その第一の要件はエンドユーザのためのプログラミング環境であること，そして，第二の要件はマルチメディア処理のマッシュアップ環境であることであった．本研究の問題はこの2つの要件を同時に満たす環境の実現であった．以下，2つの要件を再掲する．

要件 1: エンドユーザのためのプログラミング環境　多くのエンドユーザに受け入れられるためには，言語仕様，及び，プログラムの編集作業から，実行，結果の確認，再編集という一連の流れに対して，使い始めるまでの新たな学習量が少なく，プログラムや編集・実行の状態を頭の中で思い浮かべる負担が少なく，作業量の点でも少ないといった点が重要である．そのため，可視化を効果的に利用するというビジュアル・プログラミングのアプローチが重要となる．

要件 2: マルチメディア処理のマッシュアップ環境　Web APIとして提供される多様なメディアに対応したメディア処理部品を簡便に利用でき，また柔軟に組合せて各種の応用に適用できる点が重要である．更に多数の部品の管理機能も重要となる．

本論文で提案しているフォルダ・プログラミング環境が，本研究の第二の要件であるマルチメディア処理のマッシュアップ環境に適合性するかについては，4.5節でマッシュアップ要件 (M1)〜(M5) について各々分析を行い，十分適合していることが実験や実践の結果として確認できた．

本章では，本研究の第一の要件であるエンドユーザのためのプログラミング環境としてのフォルダ・プログラミング環境の適合性について分析するため，第4章，及び第5章の実験やアンケートから導かれた各種要因を，エンドユーザ・プログラミングの分析観点テンプレート（表6.1）を使って整理する．

この結果確認できるフォルダ・プログラミング環境のエンドユーザ・プログラミング環境としての主要な効果は以下の点である．

表 6.1: フォルダ・プログラミング環境に対する EUP 分析と整理

		環境側面 （外的）				
		e1 言語仕様	e2 編集時	e3 実行時	e4 結果確認時	e5 応用時
ユーザ受容性側面（内的）	u1 学習量	原則は単純 (dataflow)	編集操作の 学習不要	実行操作の 学習不要	表示操作の 学習不要	編集操作の 学習不要
		部品の合成が容易	初心者が，フォルダ作成から実行，結果確認までを， すぐに体験可能			
		繰り返し処理や変数の理解なしで，一括処理の利用が容易	Webを意識しないで編集・実行が可能		習熟済Viewerを 利用可能	
		処理付フォルダ名からHELP入手可能	フォルダ名の検索で 部品の発見が可能			フォルダ名の検索で 部品の発見が可能
		日本語フォルダ名 利用可能				
		フォルダ階層の コピーやテキスト化 で配布が容易				
	u2 認知思考量		編集・実行・結果確認に，思考のスイッチが不要で理解容易			
				1フォルダの処理毎に結果を見ることも可能(copyフォルダ)		
				単独部品で実行 可能で理解容易	多彩なメディアの 内容確認が容易	途中からの 実行が容易
				Drag&Drop操作が 直感的で理解容易		途中までの 実行が容易
				× エラーに対する サポートが不十分		
	u3 作業量	Editor,Viewerの 活用で入出力処理 は最小化	フォルダを作ってファイルをDrag&Drop するだけで実行できるので簡単		習熟済Viewerを 利用可能	
				大量データに対する 一括処理が容易	大量データ用 Viewerで一覧表示 を利用可能	
			プログラムの保存操作が不要で保管容易			保管容易

(a) Web フォルダの知識のみで処理を実行できるので Web API の学習が不要.

(b) 編集時・実行時・結果確認時等の全てのタイミングでフォルダ操作の既存知識を活用できるのでプログラミング環境の学習・習熟が極少.

(c) デスクトップの各種 Viewer,Player,Editor と連動しておりメディア表示の既存知識を活用できるのでプログラミング環境の学習・習熟が極少.

(d) 処理の即時実行機能とメディアデータの即時表示機能によって学習と認知思考が容易.

(e) プログラムの編集と実行の環境の切り替えが不要なので認知思考が容易.

(f) ファイルやフォルダに対して直感的な Drag&Drop 操作が使えるので認知思考が容易.

(g) 大量データへの一括処理が一度の Drag&Drop で簡便に実行できるので作業量が低減.

これらの効果により，本研究で提案するフォルダ・プログラミング環境は，エンドユーザのためのプログラミング環境として，十分高い適合性があると考えられる.

そして，本研究の 2 つの要件であるエンドユーザのためのプログラミング環境と，マルチメディア処理のマッシュアップ環境とを両立できる環境であるフォルダ・プログラミング環境は，エンドユーザによるメディア処理のための部品統合環境として有効なソフトウェア・アーキテクチャであると言える.

6.2 なだらかなスキル成長の支援

MacLean や Spahn が指摘しているように，エンドユーザがより高度な課題を解いていくためには，なだらかな傾斜の成長のための中間ステップの提供が必要である．一般にプログラミングの目的は何度も繰り返す処理の自動化である．しかし，保福が指摘しているように，その繰り返し処理のプログラミングの理解やその前提となる変数の理解などが，プログラミングを学習する初期の大きな障壁の一つとなっている．

従来は，図 1.1 に示したように，開発者のスキルと，利用者のスキルの間には大きなスキルの障壁が存在していた．その障壁を下げようと，多くのビジュアル・プログラミング言語が提案されてきた．例えば，Yahoo! Pipes などの GUI 環境がそれに該当する．

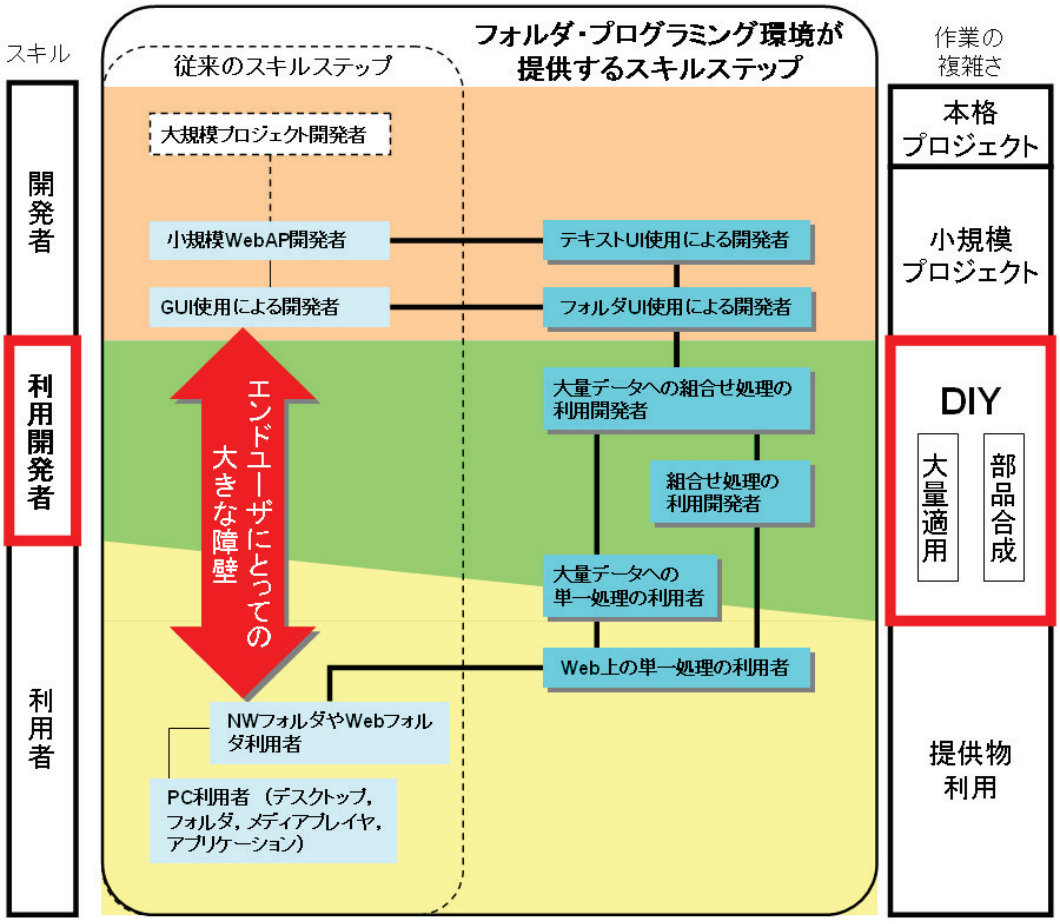


図 6.1: 新たな中間ステップ群となだらかなスキル成長

本研究で提案するフォルダ・プログラミング環境は，図 6.1 のように，複雑な作業に対応するスキル獲得のために複数の新たな中間ステップを提供し，なだらかなスキルの成長を支援する．それは，具体的には以下の（追加A）～（追加D）である．

(既存１) **PC 利用者:** PC のデスクトップ，フォルダ，メディアプレイヤや各種アプリケーション（例．Microsoft Office, Media Player）の利用者

(既存２) **NW/Web フォルダ利用者:** ネットワークフォルダ (例．CIFS, Samba, NAS) や Web フォルダ (WebDAV) の利用者

(追加Ａ) **Web 上の単一処理の利用者:** Web 上で提供される単一の処理の利用者

(追加Ｂ) **大量データへの単一処理の利用者:** 大量データに対して Web 上の単一処理へ適用する利用者

(追加Ｃ) **組合せ処理の利用開発者:** Web 上の処理を組合せて処理活用する利用開発者

(追加Ｄ) **大量データへの組合せ処理の利用開発者:** 大量データに対して Web 上の処理を組合せた処理へ適用する利用開発者

(既存３) **GUI 使用による開発者（≒フォルダ UI 使用による開発者）:** GUI（フォルダ UI 含む）によるプログラミング環境を使って Web 上の API を使ったプログラムを作成する開発者

(既存４) **小規模 WebAP 開発者（≒テキスト UI 使用による開発者）:** テキストによるプログラミング環境を使って Web 上の API を使ったプログラムを作成する開発者

本研究がターゲットとする**日常的にプログラミングを行わない PC 利用者**の多くは，デスクトップやフォルダを使い，各種のメディアプレイヤやアプリケーションを PC 上で利用するだけである．プログラミング環境を使って Web 上の API を使ったプログラムを作成する**小規模 WebAP 開発者**や**GUI 使用による開発者**との間には極めて大きなスキルの障壁が存在していた．

それに対しフォルダ・プログラミング環境は障壁を小さくするための複数の中間ステップを提供している．そして，それらの中間ステップのスキル差も各々小さいものとしている．

PC 利用者から，Web フォルダ利用者へ

CIFS や Samba のようなネットワーク (NW) フォルダは，オフィスにおいても情報共有の手段として広く利用されており，フォルダ利用との障壁は極めて僅かである．WebDAV を使った Web フォルダも同様である．今回我々が利用した TeamFile の WebDAV クライアントは Windows の Explorer に機能追加して利用するが，ほとんどの実験参加者にとっ

て，それが日常利用する Explorer と何が違うかを認識できない程度に差異は小さかった．ソフトウェアの初期導入の設定を除けば，ここにも大きな障壁[1]は存在しない．

Web フォルダ利用者から，Web 上の大量データへの単一処理の利用者へ

WebDAV の拡張であるフォルダ・プログラミング環境 POLDER では，Web フォルダの利用が，そのまま Web 上の単一処理の利用となる．そのフォルダにデスクトップ上の複数のファイルを Drag&Drop すれば，大量データの Web 上の単一処理の利用のレベルまで達する．つまり，Web フォルダ利用者から Web 上の大量データへの単一処理の利用者までのスキル獲得に大きな障壁は存在しない．

そして，このレベルに達した利用者は，プログラミングを行うこと無く，Web 上の処理を使って，大量のデータを処理するという効果を手に入れることができる．この際，一般のプログラミング言語で必要とされる繰り返し処理の知識は不要である．つまり利用者は，利用開発者のスキルレベルに達することなく，プログラミングや DIY の目的の一つである大量のデータへの繰り返し適用が達成できるという満足感を得ることができる．

Web 上の単一処理の利用者から，組合せ処理の利用開発者へ

プログラミングや DIY のもう一つの目的として，処理の組合せ実行の自動化がある．Web 上の単一処理の利用となった利用者は，日常のデスクトップ作業の中でフォルダの階層化を行うのと同じ作業で，Web 上のフォルダ階層を作成することによって，組合せ処理の利用のレベルに達する．ここで，この処理付フォルダの連鎖を構成することに大きな障壁は存在しない．利用者は，いとも簡単に利用開発者のスキルレベルに到達できる．

このレベルに達した利用開発者は，画像のフォーマット変換や加工などを組合せるという処理の統合が可能になる．

大量データへの組合せ処理の利用開発者へ

組合せ処理の利用開発者は，Web フォルダの利用者がそのまま Web 上の単一処理の利用者となったのと同じように，そのフォルダにデスクトップ上の複数のファイルを Drag&Drop することで，大量データへの組合せ処理の利用開発者のレベルまで達する．ここに大きな障壁は存在しない．

このレベルに達した利用開発者は，映像から代表シーンを抽出して画像サイズを縮小する処理を手元の大量のデータに適用することなどの，数多くの応用シーンへの対応が可能になる．

[1] WebDAV クライアントは多種の実装が存在する．それらの全てについて言えることではない．

このようにフォルダ・プログラミング環境では，スキルの獲得の障壁が少ない複数の中間ステップが提供され，利用者が利用開発者に自然に成長し，Web 上の処理の組合せや大量データへの適用といった DIY をすることが容易になるといえる．

図 6.1 で，注目すべき点は DIY のケースで考えられる作業のうち，Web 上の処理に大量データの適用する作業については，利用開発者というスキル段階に成長する前の，単なる利用者のスキルで達成できてしまう点である．これはエンドユーザにとっては，極めて少ない学習量で，従来プログラミングが必要であったレベルの効果を得たことになり，環境の継続利用の強いモチベーションとして働くだろう．

そして，もう一つの注目すべき点は，フォルダ・プログラミング環境によって提供される各中間ステップは，これまで述べたように各々が実用的な意味を持つステップであり，単独で実行でき，その効果がすぐに体験できるという点である．保福の分析のように，旧来のプログラミング言語では，繰り返しの効果を体験するまでに，変数，代入，インクリメント，比較演算子，条件式，条件分岐，ブロックなど多数の事前理解が必要であり，更に，作成から実行までの統合環境操作方法についても理解しなければならなかった．これでは，多くのエンドユーザは，プログラム作成による効果よりもそのために必要な学習量が多すぎると感じてスキル獲得へのモチベーションが低下し学習が進まない．フォルダ・プログラミング環境は，従来に比べて，利用者にとって少ない時間の投資で大きな効果が得られるメディア処理の統合環境を提供しているといえる．

更に高度な処理のための開発者へ

直列に連鎖する処理の記録と再演による処理の再利用，及び大量データへの適用は，EUP の有望なアプローチであるデモンストレーションによるプログラミング (PBD) でも不可能ではなかった．しかし PBD では，幅広い領域に対しての処理の一般化は困難で，一度作った処理の部分利用や修正への障壁も高かった．

フォルダ・プログラミング環境が PBD のアプローチより優れる点は，更に高度な処理を望んだ利用開発者は，ファイルの種別毎の処理を条件分岐で実現したり，集約によって処理結果をトップの階層に自動的に集めたり，それをサブルーチン化して他から呼び出したり，テキスト・ユーザ・インタフェースから利用したりといった，開発者に成長するためのスキル獲得に向けた更なる中間ステップが存在する点にある．

以上のように，フォルダ・プログラミング環境は，成長のための多数の補助的中間ステップを提供することによって PC の利用者が利用開発者になるための漸増的なスキル成長を支援する環境になっている．

第7章　結論

本論文の目的は，デジタルカメラやスマートフォンなどの携帯型デジタル機器の普及に伴って広まってきたメディアデータの加工・編集を誰でも行えるようにすることを狙いとした，エンドユーザのためのメディア処理部品の統合環境を構成する方法を明らかにすることである．

学生時代に少しだけプログラミングの授業を履修したがそれ以降は全くプログラミングを行わない，**日常的に PC は利用しているがプログラマとは言えない学生・社会人**は極めて多い．それらの人々が，大量のメディアデータを保有するようになり，加工や整理といったメディア処理を DIY で行いたいという要求が高まっている．本研究では，すでに多くの利用者が日常的に利用しているフォルダ管理という作業の経験と知識に着目し，それをプログラミングに利用することで，エンドユーザに対するプログラミング・スキルの習得の障壁の低減する．そのアーキテクチャとして，Web フォルダをプログラミング環境として拡張するフォルダ・プログラミング環境の提案を行い，その実現性を POLDER の実装によって示した．フォルダ・プログラミング環境の核となる考え方は，フォルダへのデータファイル DROP で処理が起動し，フォルダ名をもとに処理が動的に決定し実行され，結果がフォルダ内に入るという仕組みである．それに，フォルダの入れ子構造に応じた連鎖・並行・条件分岐といった制御構造を与えることで基礎的なプログラミングを可能とした．また，フォルダへの複数ファイルの Drag&Drop に対して一斉に実行される仕組みを使って，大量のメディアデータへのメディア処理の実現を容易にした．いくつかの利用者実験とそれらの評価結果から，本研究で提案するフォルダ・プログラミング環境が，本研究で問題となる 2 つの要件を十分満たし両立する環境であることを示した．また，比較実験を通して，これらの仕組みが効果的に利用者に訴求することを定量的に明らかにした．この提案環境によって，日常的に PC は利用しているがプログラマとは言えない利用者が，大量のメディアデータに対するメディア処理を DIY で行うことができるようになる．更に，DIY でプログラミングを始めたエンドユーザが，より高度な課題を解いていくために必要となる学習のなだらかな傾斜の実現に対して，本環境は，複数の新たな中間ステップを提供し支援することを論じた．これらの議論を通して，本研究が目的とするエンドユーザのためのメディア処理部品の統合環境の構成方法を明らかにした．

今後の課題

フォルダ・プログラミング環境の今後の課題を示す.

利便性向上に関する課題

現状の素直な実装の延長上で，利便性の向上につなげていくためには，以下のような課題がある.

- UNDO 機能の実現（もしくは，自動履歴管理）.
- ERROR 処理の拡充，発生時のサポート拡充.
- 処理が長い場合のシステム状態のフィードバックの拡充.
- 新たなメディア処理部品の処理付フォルダへの簡単な組み込み.
- 要求する処理付フォルダの発見支援の高度化.
- 並行処理の洗練された実装と高速化.
- セキュリティ対策[1].
- クライアントの各種実装の課題（キャッシュの回避や，既存の Explorer との UI 互換性，初期導入の手間[2]）.

プログラミング環境としての進化の方向性

フォルダ・プログラミング環境は，プログラミング・スタイルの模索を継続しながら今後も進化が必要である．Lisp [159] が CommonLisp [160], Scheme [161], EmacsLisp [162], TAO [163] [164], CLOS [165], Clojure [166], Arc [167] といったたくさんの派生言語を生んだように，フォルダ・プログラミング環境をベースとした各種の派生・拡張言語や DSL が広まることが期待される.

- 多様な分野に対する DSL 化（更に PBD との連携など）.
- 人工知能言語や知識表現としての活用（データ表現とプログラム表現の同一性を活用した高階プログラミング対応など），及びメタファ混乱への対策.

[1] Web サービス全般の課題としてのセキュリティ対策を指す.
[2] HTML5 版クライアントの導入が期待される.

- リアルタイム・ストリームデータへの対応，及びインタラクティブなパラメータ調整への対処．例えば，Pure Data やImproV [94] [95]）のようなリアルタイムなストリームデータに対する処理．
- オブジェクト指向の導入．
- Many Core 向けの言語拡張，クラウド環境の高度な活用．

例えばDSL として，いずれかのPBD によって生成されるマクロの結果をPOLDER 上のフォルダ・プログラムにすることは高い効果が期待できるアプローチと考えられる．なぜなら，PBD の持つ条件分岐の作成の難しさや部分実行・修正の難しさなどをフォルダ・プログラミング環境が低減する可能性をもつからである．

更に，フォルダ・プログラミング環境ではデータ構造とプログラム構造がフォルダという同一表現であるため，動的にプログラム構造を構成できるという特性もある．この特性を使えば，Lisp やProlog [168] のような人工知能向きのプログラミングの環境に適用することが考えられ，高階関数の仕組みを導入することも考えられる．

一方，この特性は，負の側面の課題も発生させる．今回の実験では観測されなかったが，フォルダを処理付フォルダに拡張し，フォルダ階層をデータフローに対応付ける本環境は，場合によっては利用者の混乱を誘発し，意外なエラーを発生させるかもしれない．もし，今後それらが検出された場合には，エンドユーザ向けの環境としての改善が必要になると考えられる．

リアルタイム化は，現状でも Motion JPEG のような映像に対する逐次処理は不可能ではない．しかし．近年の多くのビジュアル・プログラミングが得意とする波形（音声など）の処理への対応は今後の課題である．例えば小林 [94][95]）の ImproV のようなストリームデータに対するインタラクティブなパラメータ調整技術との技術統合も今後の方向の可能性として存在する．

オブジェクト指向の導入についても，今後の検討が必要である．過去にも多くの言語がオブジェクト指向言語化を行っている[3]．オブジェクト指向の導入によって，より抽象化を進めた大規模プログラムの開発を支援可能になると考えられる．

その他，フォルダ・プログラミングでは，大量のデータを一括投入する特性とともに，並行処理の記述が自然に記述できる特性があった．これらの特性を活用して，今後，主流となっていく Multi Core, Many Core ベースの並列処理用の言語基盤となる可能性も考えられる．

このように，フォルダ・プログラミング環境の研究には，さらなる広大な課題空間・研究領域が広がっていると考えられる．

[3]C 言語が C++言語へ，Lisp が CLOS へなど

本研究は，学生や社会人などでプログラミングを日常的に行わない PC 利用者が，所有する大量かつ多様なメディアデータを，Web 上で提供される各種のメディア処理部品サービスを自在に組合せて，簡便に DIY で多彩な加工処理を実施する世界の実現に向けて，フォルダを拡張するという新たなソフトウェア・パラダイムからのアプローチを切り開いたと考えている．

謝辞

本論文は，筆者が日本電信電話株式会社 NTT サイバースペース研究所，NTT ソフトウェアイノベーションセンタ在職中，及び，筑波大学大学院 図書館情報メディア研究科博士後期課程に在籍中の研究成果をまとめたものです．本論文の作成にあたり，多数の皆様からのご指導とご支援をいただきました．

まず，はじめに博士後期課程を主指導教員としてご指導をいただきました図書館情報メディア系の佐藤 哲司 教授には，謹んで感謝を申し上げます．また，副指導教員としてご指導いただきました同系の松本 紳 教授，杉本 重雄 教授，田中 和世 教授にも深く感謝申し上げます．更に，学位論文の審査委員としてご指導いただいた，同系の鎮目 浩輔 教授，長谷川 秀彦 教授，並びに，システム情報系の田中 二郎 教授にも，ここに深く感謝申し上げます．皆様からのご指導と議論させていただいた内容を活かして，今後の研究を進めて行きたいと考えています．

本研究では，第 3 章や第 4 章に示したいくつかの実験を行いました．それらの実験では，筑波大学の学生諸氏に多大なご協力をいただきました．特に，本実験の運営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（現 NTT コミュニケーションズ株式会社）をはじめとする佐藤研究室の皆様に感謝いたします．

日本電信電話株式会社の皆様にも大変お世話になりました．特に，山室 雅司 グループリーダ，湯口 徹 グループリーダには，本研究の推進に関する深いご理解とご支援をいただきました．更に，盛合 敏 プロジェクトマネージャ，内川 昌平 プロジェクトマネージャ，藤田 敏昭 ソフトウェアイノベーションセンタ所長，串間 和彦 サービスイノベーション総合研究所長には，業務調整等でご理解いただくとともに，多大なるご支援をいただきました．また，本研究に関わる共著者である皆様，毛受 崇 氏，内藤 一兵衛 氏，谷口 展郎 氏，長谷川 知洋 氏，鶴岡 行雄 氏にも感謝いたします．そして，業務のサポート等でご協力いただいた皆様，車谷 駿介 氏，外山 将司 氏，松尾 嘉典 氏，阿部 剛仁 氏，藤田 将成 氏，松村 聖司 氏，奥村 昌和 氏，小谷 尚也 氏，千葉 一深 氏，富田 千智 氏，清水 亮博 氏，三島 健 氏に感謝いたします．皆様のご理解とご協力が無ければ，仕事と学業の両立は困難でした．

本研究の萌芽期において本研究の可能性を支持していただいた情報処理学会 DPS 研究会の上原 稔 先生，東野 輝夫 先生をはじめとする皆様には大変感謝いたします．また，首都圏空港の閉鎖に伴い大混乱する中で離島での研究会を開催・運営し，活発にご議論いただきました情報処理学会 PRO 研究会の兼宗 進 先生，鈴木 貢 先生，増原 英彦 先生，その他 PRO 研関係の多数の皆様，伊地知 宏 先生，そして評価手法に関して多数のアドバイスをいただきました山之上 卓 先生に心から感謝を申し上げます．

最後に，社会人学生としての活動を理解し支援してくれた，子供たち実萌，滉星，海誠と，妻の久美子に心から感謝いたします．そして震災にも負けずに福島の復興に尽くす両親と故郷の人々にも感謝いたします．

皆様，ありがとうございました．

参考文献

[1] ImageMagick Studio LLC.
`http://www.imagemagick.org/`.

[2] The FFmpeg project.
`http://www.ffmpeg.org/`.

[3] OpenCV (Open Source Computer Vision).
`http://opencv.willowgarage.com/`.

[4] Yahoo! Pipes.
`http://pipes.yahoo.com/pipes/`.

[5] Google Map API.
`https://developers.google.com/maps/`.

[6] Michael Spahn, Christian Dörner, and Volker Wulf. End User Development: Approaches Towards a Flexible Software Design. In *Proceedings of 16th European Conference on Information Systems (ECIS)*, pp. 303–314, 2008.

[7] Allen Cypher, editor. *Watch What I Do -Programming by Demonstration.* MIT Press, 1993.

[8] Ben Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. In R. M. Baecker and W. A. S. Buxton, editors, *Human-Computer Interaction*, pp. 461–467. Morgan Kaufmann, 1987.

[9] Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. User-Tailorable Systems: Pressing the Issues with Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90)*, pp. 175–182. ACM, 1990.

[10] 保福やよい. ぺた語義:なぜプログラミングは難しいのか? 繰り返しの理解構造とCの教科書分析からのアプローチ. 情報処理学会 情報処理, Vol. 54, No. 3, pp. 252–255, 2013.

[11] 保福やよい, 西田知博, 長慎也, 兼宗進. なぜプログラミングは難しいのか? 繰り返しの理解構造と C の教科書分析からのアプローチ. 情報処理学会 情報教育シンポジウム (SSS2012), PD-1, 2012.

[12] Alvin Toffler. *THE THIRD WAVE*. Bantam, 1980.
（アルビン・トフラー, 徳岡孝夫 監訳. 第三の波. 中央公論社,1982）.

[13] Jakob Nielsen. *Usability Engineering*. Academic Press, 1994.
（ヤコブ・ニールセン, 篠原稔和 監訳, 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, 2002）.

[14] 文部科学省. 新学習指導要領・生きる力. 第 2 章 各教科 第 8 節 技術・家庭, 2008.
`http://www.mext.go.jp/a_menu/shotou/new-cs/youryou/chu/gika.htm`.

[15] Annette Wagner, Patrick Curran, and Robert O'Brien. Drag Me, Drop Me, Treat Me Like an Object. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pp. 525–530. ACM Press/Addison-Wesley, 1995.

[16] William Buxton. Chunking and Phrasing and the Design of Human-Computer Dialogues. In *Proceedings of the IFIP World Computer Congress*, pp. 475–480. North Holland Publishers, 1986.

[17] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC1945: Hypertext Transfer Protocol -HTTP/1.0, 1996.
`http://www.ietf.org/rfc/rfc1945.txt`.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC2616: Hypertext Transfer Protocol -HTTP/1.1, 1999.
`http://www.ietf.org/rfc/rfc2616.txt`.

[19] L. Dusseault. RFC4917: HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV), 2007.
`http://www.ietf.org/rfc/rfc4918.txt`.

[20] 宮本久仁男, 山田泰資, 渡邊剛. WebDAV システム構築ガイド. 技術評論社, 2003.

[21] 一杉裕志, 古川浩史. エンドユーザ向けのスクリプト言語 : チャミー. 日本ソフトウェア科学会第 7 回プログラミングおよびプログラミング言語ワークショップ (PPL2005), 2005.
`http://www.graco.c.u-tokyo.ac.jp/`.

[22] 新村 出 (著・編集). 広辞苑 第六版. 岩波書店, 2008.

[23] 小学館 (編集). 日本国語大辞典 〔精選版〕. 小学館, 2005.

[24] L. Gould and W. Finzer. Programming by Rehearsal. *Byte*, Vol. 9, No. 6, pp. 187–210, 1984.

[25] Automator.
`http://www.automator.us/leopard/index.html`.

[26] Adobe Systems. Photoshop.
`http://www.photoshop.com/`.

[27] Francesmary Modugno, Albert T. Corbett, and Brad A. Myers. Graphical Representation of Programs in a Demonstrational Visual Shell An Empirical Evaluation. *ACM Transactions on Computer-Human Interaction*, Vol. 4, No. 3, pp. 276–308, 1997.

[28] 萩谷昌己. 視覚的プログラミングと自動プログラミング. 日本ソフトウェア科学会 コンピュータソフトウェア, Vol. 8, No. 2, pp. 123–135, 1991.

[29] 松岡聡, 宮下健. 例示による GUI 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, 4 章, pp. 79–97, 1996.

[30] Allen Cypher. EAGER: Programming Repetitive Tasks by Example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'91)*, pp. 33–39. ACM, 1991.

[31] 赤間浩樹, 峯崎俊哉, 成嶋弘. 帰納型プロダラムの機械学習システム『Bくん』. 情報処理学会 全国大会講演論文集, pp. 351–352, 1990.

[32] L. Gould and W. Finzer. *Learning from Good and Bad Data*. Kluwer Academic Publishers, 1988.

[33] The Evolution of FORTH.
`http://www.forth.com/resources/evolution/index.html`.

[34] 片桐明. 日本語プログラミング言語 Mind.
`http://www.scripts-lab.co.jp/mind/whatsmind.html`.

[35] くじらはんど. 日本語プログラミング言語「なでしこ」.
`http://nadesi.com/`.

[36] 粗茶@なでしこ. 日本語プログラミング言語「なでしこ」ユーザーズ・マニュアル vol.1(Kindle 版). なでしこ友の会, 2012.

[37] 馬場ゆうと. TTSneo 公式サイト.
http://tts.utopiat.net/.

[38] 馬場ゆうと. 日本語プログラミング言語「プロデル」.
http://rdr.utopiat.net/whats.html.

[39] 岡田健, 中鉢欣秀, 鈴木弘, 大岩元. 日本語プログラミング言語「言霊」. 情報科学技術フォーラム FIT 講演論文集, pp. 119–120, 2002.

[40] Squeak.
http://squeak.org/.

[41] 岡田健, 杉浦学, 松澤芳昭, 大岩元. 教育用プログラミング言語としての「言霊」と「ことだま on Squeak」の試み. 情報処理学会教育用プログラミング言語に関するワークショップ 2006 報告集, pp. 44–49, 2006.

[42] 兼宗進, 御手洗理英, 中谷多哉子, 福井眞吾, 久野靖. 学校教育用オブジェクト指向言語「ドリトル」の設計と実装. 情報処理学会論文誌, Vol. 42, No. SIG11(PRO12), pp. 78–90, 2001.

[43] 兼宗進, 久野靖. プログラミング言語ドリトル：グラフィックスから計測・制御まで. イーテキスト研究所, 2011.

[44] 兼宗進. 工学系学科でのプログラミング入門教育 -ドリトルを利用して-. 情報処理学会 情報処理, Vol. 52, No. 6, pp. 736–739, 2011.

[45] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1987.
（Ben Shneiderman 著, 東 基衛, 井関 治 監訳. ユーザー・インタフェースの設計 -使いやすい対話型システムへの指針-. 日経 BP 社, 1987）.

[46] Donald A. Norman. *The Design of Everyday Things*. Basic Books, 1988.
（ドナルド A. ノーマン著, 野島 久雄 訳. 誰のためのデザイン? -認知科学者のデザイン原論. 新曜社, 1990）.

[47] U-Site. ニールセン博士の Alertbox.
http://www.usability.gr.jp/alertbox/.

[48] NN/g Nielsen Norman Group.
http://www.nngroup.com/.

[49] ISO 9241-110. Ergonomics of Human-System Interaction Part 110:Dialogue Principles, 2006.

[50] JIS Z8520. 人間工学 - 人とシステムとのインタラクション - 対話の原則, 2008.
http://www.jisc.go.jp/app/JPS/JPS00020.html.

[51] 樽本徹也. ユーザビリティエンジニアリング　-ユーザ調査とユーザビリティ評価実践テクニック-. オーム社, 2005.

[52] 赤間浩樹, 鶴岡行雄, 佐藤哲司. 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. 情報処理学会論文誌 プログラミング (PRO), Vol. 6, No. 2, pp. 69–84, 2013.

[53] Nan C. Shu. *Visual Programming*. Van Nostrand Reinhold, 1988.
（Nan C.Shu, 西川博昭 訳. ビジュアル・プログラミング. 日経 BP 社, 1991）.

[54] Brad A. Myers. Visual Programming, Programming by Example and Program Visualization: A Taxonomy. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vol. 17, No. 4, pp. 59–66, 1986.

[55] Shi-Kuo Chang. Visual Languages: A Tutorial and Survey. *IEEE Software*, Vol. 4, No. 1, pp. 29–39, 1987.

[56] Tadao Ichikawa and Masahito Hirakawa. Visual Programming - Toward Realization of User-Friendly Programming Environments. In *Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow*, ACM '87, pp. 129–137. IEEE Computer Society Press, 1987.

[57] M. Hirakawa, S. Iwata, Y. Tahara, M Tanaka, and T. Ichikawa. A Framework for Construction of Icon Systems. In *Proceedings of the 1988 IEEE Workshop on Visual Languages*, pp. 70 – 77. IEEE Computer Society Press, 1988.

[58] Masahito Hirakawa, Minoru Tanaka, and Tadao Ichikawa. An Iconic Programming System, HI-VISUAL. *IEEE Transactions on Software Engineering*, Vol. 16, No. 10, pp. 1178 – 1184, 1990.

[59] Steven L. Tanimoto. VIVA: A Visual Language for Image Processing. *Journal of Visual Languages and Computing*, Vol. 1, No. 2, pp. 127–139, June 1990.

[60] Carla S. Williams and John R. Rasure. A Visual Language for Image Processing. In *Proceedings of the 1990 IEEE Workshop on Visual Languages*, pp. 86 – 91, 1990.

[61] Konstantinos Konstantinides and John R. Rasure. The Khoros Software Development Environment for Image and Signal Processing. *IEEE Transactions on Image Processing*, Vol. 3, No. 3, pp. 243 – 252, 1994.

[62] G. Williams G.M. Vose. LabVIEW: Laboratory Virtual Instrument Engineering Workbench. *BYTE*, pp. 84–92, September 1986.

[63] National Instruments. LabVIEW.
`http://www.ni.com/labview/whatis/ja/`.

[64] Info-LabVIEW Mailing List.
`http://www.info-labview.org/`.

[65] Scott B. Steinman and Kevin G. Carver. *Visual Programming With Prograph CPX*. Prentice Hall, 9 1995.
`http://www.andescotia.com/support/`.

[66] Andescotia LLC. Marten 1.4, Graphical Programming for MacOS X.
`http://www.andescotia.com/products/marten/`.

[67] Elizabeth Vera Howard. *Visual Programming: Concepts and Implementations*. Computer Science and System Analysis Technical Reports, Miami University, 1994.

[68] T. R. G. Green and M. Petre. Usability Analysis of Visual Programming Environments: a 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing*, Vol. 7, No. 2, pp. 131–174, 1996.

[69] 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, 3 章, pp. 65–78, 1996.

[70] 上田和紀. GHC (Guarded Horn Clauses). 人工知能学会誌, Vol. 3, No. 1, 1988.

[71] 藤田博, 奥村晃, 上田和紀. 並列論理型言語 GHC とそのプログラミング技術 (<特集>「第五世代コンピュータ」). 人工知能学会誌, Vol. 4, No. 3, pp. 258–264, 1989.

[72] Jiro Tanaka. PP: Visual Programming System for Parallel Logic Programming Language GHC. In *Parallel and Distributied Computing and Networks '97*, pp. 188–193, 1997.

[73] Margaret M. Burnett and Marla Baker. A Classification System for Visual Programming Languages. *Journal of Visual Languages and Computing*, Vol. 5, No. 3, pp. 287 – 300, 1994.

[74] Margaret M. Burnett. Visual Language Research Bibliography. `http://web.engr.oregonstate.edu/~burnett/vpl.html`.

[75] Margaret M. Burnett. Visual Programming. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999.

[76] Alan F. Blackwell, Kirsten N. Whitley, Judith Good, and Marian Petre. Cognitive Factors in Programming with Diagrams. *Artificial Intelligence Reveview*, Vol. 15, No. 1/2, pp. 95–114, 2001.

[77] G. Costagliola, A. Delucia, S. Orefice, and G. Polese. A Classification Framework to Support the Design of Visual Languages. *Journal of Visual Languages and Computing*, Vol. 13, No. 6, pp. 573–600, 2002.

[78] Brad A. Myers. Graphical User Interface Programming, 2003.

[79] Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. Advances in Dataflow Programming Languages. *ACM Computing Surveys*, Vol. 36, No. 1, pp. 1–34, March 2004.

[80] Marat Boshernitsan and Michael S. Downes. Visual Programming Languages: A Survey. *Technical Report Identifier: CSD-04-1368*, December 2004.

[81] Caitlin Kelleher and Randy Pausch. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Computing Surveys*, Vol. 37, No. 2, pp. 83–137, 2005.

[82] Barbara G. Ryder, Mary Lou Soffa, and Margaret Burnett. The Impact of Software Engineering Research on Modern Progamming Languages. *ACM Transactions on Software Engineering and Methodology*, Vol. 14, No. 4, pp. 431–477, 2005.

[83] Miller Puckette. Pure Data. `http://puredata.info/`.

[84] Miller Puckette. Pure Data. `http://www.crca.ucsd.edu/~msp/`.

[85] Miller S. Puckette. Pure Data. In *Proceedings of International Computer Music Conference*, pp. 224–227, 1996.

[86] MathWorks. MATLAB 数値計算言語. `http://www.mathworks.co.jp/products/matlab/`.

[87] MathWorks. Simulink シミュレーションおよびモデルベースデザイン. `http://www.mathworks.co.jp/products/simulink/`.

[88] 青山貴伸. 使える! MATLAB Simulink プログラミング (KS 理工学専門書). 講談社, 2007.

[89] Cybernet. ビジュアルプログラミング可視化ツール AVS/Express Viz/Dev. `http://www.cybernet.co.jp/avs/products/avsexpress/`.

[90] 岡田栄治, 山口真悟, 田中稔. インタフェースボードを使用するソフトウェアのためのビジュアルプログラミングシステム. 情報処理学会論文誌 プログラミング, Vol. 42, No. SIG 11(PRO 12), pp. 67–77, 2001.

[91] 小川徹, 田中二郎. ドラッグ&ドロップを用いたビジュアルプログラミングシステム. 情報処理学会論文誌 プログラミング, Vol. 43, No. SIG 1(PRO 13), pp. 36–47, 2002.

[92] 早野浩生. ドラッグ&ドロップでスクリプトの記述が可能なシェル. 日本ソフトウェア科学会 コンピュータソフトウェア, Vol. 16, No. 5, pp. 468–471, 1999.

[93] 早野浩生. ドラッグ&ドロップのすすめ. 共立出版 Bit, Vol. 31, No. 5, pp. 40–45, 1999.

[94] Atsutomo Kobayashi, Buntarou Shizuki, and Jiro Tanaka. ImproV: A System for Improvisational Construction of Video Processing Flow. In *Proceedings of the 13th International Conference on Human-Computer Interaction. Part IV: Interacting in Various Application Domains*, pp. 534–542. Springer-Verlag, 2009.

[95] 小林敦友, 志築文太郎, 田中二郎. GPU を利用したライブ映像パフォーマンス向け映像合成システム. 情報処理学会論文誌 プログラミング, Vol. 4, No. 1, pp. 76–89, 2011.

[96] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The State of the Art in End-User Software Engineering. *ACM Computing Surveys*, Vol. 43, No. 3, pp. 21:1–21:44, 2011.

[97] I. Nassi and B. Shneiderman. Flowchart Techniques for Structured Programming. *ACM SIGPLAN Notices*, Vol. 8, No. 8, pp. 12–26, 1973.

[98] 二村良彦, 川合敏雄, 堀越彌, 堤正義. PAD (Problem Analysis Diagram) によるプログラムの設計および作成. 情報処理学会論文誌, Vol. 21, No. 4, pp. 259–267, 1980.

[99] 二村良彦. 構造化プログラム図式. 日本ソフトウェア科学会 コンピュータソフトウェア, Vol. 1, No. 1, pp. 64–77, 1984.

[100] Andy Cockburn and Andrew Bryant. Leogo: An Equal Opportunity User Interface for Programming. *Journal of Visual Languages and Computing*, Vol. 8, pp. 8–5, 1997.

[101] Logo Foundation.
`http://el.media.mit.edu/logo-foundation/`.

[102] L. M. Pereira, F. C. N. Pereira, and D.H.D.Warren. *User's Guide to DEC System-10 PROLOG*. Dept. of Artificial Intelligence, Edinburgh, 1978.

[103] A. Dornan. Mashup Basics: Three for the Money. *Network Computing*, 2007.
`http://www.networkcomputing.com/data-networking-management/mashup-basics-three-for-the-money/229606107`.

[104] 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. 情報処理学会 情報処理, Vol. 50, No. 5, pp. 444–453, 2009.

[105] JackBe Presto.
`http://www.jackbe.com/products/`.

[106] Tony Loton. *Working With Yahoo! Pipes, No Programming Required*. CreateSpace, 2008.

[107] IBM Mashup Center.
`http://www-10.lotus.com/ldd/mashupswiki.nsf`.

[108] Microsoft Popfly.
`http://en.wikipedia.org/wiki/Microsoft_Popfly`.

[109] Michael Muchmore. Microsoft Popfly. *PC Magazine*, 2007.
`http://www.pcmag.com/article/print/218471`.

[110] Microsoft Silverlight.
`http://www.silverlight.net/`.

[111] Google Mashup Editor.
http://en.wikipedia.org/wiki/Google_Mashup_Editor.

[112] Intel MashMaker.
http://mashmaker.intel.com/web/.

[113] Lars Grammel and Margaret-Anne Storey. A Survey of Mashup Development Environments. In *The Smart Internet*, pp. 137–151. Springer-Verlag, 2010.

[114] Steffen Heinzl, Dominik Seiler, M. Unterberger, A. Nonenmacher, and Bernd Freisleben. MIRO: A Mashup Editor Leveraging Web, Grid and Cloud Services. In *Proceedings of the 11th International Conference on Information Integration and Web-based Applications and Services (iiWAS2009)*, pp. 17–24, 2009.

[115] Apache HTTP Server Project.
http://httpd.apache.org/.

[116] Microsoft Windows Explorer.
http://ja.wikipedia.org/wiki/Windows_Explorer.

[117] Apple Mac OS X Finder.
http://ja.wikipedia.org/wiki/Finder.

[118] CIFS (Common Internet File System).
http://msdn.microsoft.com/en-us/library/aa302188.aspx.

[119] 高橋基信. アンドキュメンテッド Microsoft ネットワーク. 翔泳社, 2002.

[120] Samba.
http://us3.samba.org/samba/.

[121] FUSE (Filesystem in Userspace).
http://fuse.sourceforge.net/.

[122] Apple Mac OS X.
http://www.apple.com/jp/macosx/.

[123] Apple. Folder Action.
http://docs.info.apple.com/article.html?artnum=25514.

[124] Hanaan Rosenthal and Hamish Sanderson. *Learn AppleScript: The Comprehensive Guide to Scripting and Automation on Mac OS X*. Apress, 3rd edition, 2010.

[125] Mark Conway Munro. *AppleScript (Developer Reference)*. Wiley, 2010.

[126] Thomas Myer. *Apple Automator with AppleScript Bible*. Wiley, 2009.

[127] Wappwolf. Automator Dropbox.
http://wappwolf.com/dropboxautomator/.

[128] Dropbox.
https://www.dropbox.com/.

[129] Adobe Systems. Droplets, Batch Processing with Droplets.
http://tv.adobe.com/watch/photoshop-for-video/
batch-processing-with-droplets/.

[130] 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.

[131] Xerox Star.
http://en.wikipedia.org/wiki/Xerox_Star.

[132] The Star User Interface: An Overview. In *proceedings of the afips 1982 national computer conference*, pp.515-528, 1982.
http://www.guidebookgallery.org/articles/thestaruserinterfaceanoverview.

[133] Cameron Newham and Bill Rosenblatt. *Learning The Bash Shell*. Oreilly & Associates Inc., 2005.

[134] Randal L. Schwartz, Tom Phoenix, and Brian D. Foy. *Learning Perl*. Oreilly & Associates Inc., 2008.

[135] Michael Still. *The Definitive Guide to ImageMagick*. Apress, 2005.

[136] Apache Module mod_dav.
http://httpd.apache.org/docs/2.0/en/mod/mod_dav.html.

[137] CentOS Project.
http://www.centos.org/.

[138] Linux Foundation.
http://www.linuxfoundation.jp/.

[139] Microsoft Windows.
http://ja.wikipedia.org/wiki/Microsoft_Windows.

[140] Welcome to WebDAV Resources.
http://www.webdav.org/.

[141] Microsoft Internet Explorer.
http://ja.wikipedia.org/wiki/Windows_Explorer.

[142] Computer Hi-Tech Inc. TeamFile client.
http://www.teamfile.com/modules/mydownloads/.

[143] The Perl Programming Language.
http://www.perl.org/.

[144] 増井俊之. インターフェイスの街角, ビジュアル・プログラミング. 本当に使いやすいユーザー・インターフェイスの極意 (UNIX MAGAZINE COLLECTION), pp. 72–77, 2005.
http://www.pitecan.com/UnixMagazine/PDF/if9809.pdf.

[145] The Linux Information Project LINFO. Pipes: A Brief Introduction.
http://www.linfo.org/pipe.html.

[146] The GIMP Team.
http://www.gimp.org/.

[147] 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の連携システム. 電子情報通信学会 パターン認識・メディア理解 (PRMU) 技術研究報告, Vol. 111, No. 77, pp. 61–66, 2011.

[148] Evernote.
http://www.evernote.com/.

[149] Kota Hidaka, Naoya Miyashita, Masaru Fujikawa, Masahiro Yuguchi, Takashi Satou, and Katsuhiko Ogawa. A Video Digest and Delivery System: "Choco-ParaTV". In *Proceedings of the 2007 conference on Human interface: Part I*, pp. 437–445. Springer-Verlag, 2007.

[150] 政瀧浩和, 柴田大輔, 中澤裕一. 顧客との自然な会話を聞き取る自由発話音声認識技術「VoiceRex」(特集 コンタクトセンタ業務に革新をもたらす音声処理技術). NTT技術ジャーナル, Vol. 18, No. 11, pp. 15–18, 2006.

[151] 間野一則, 水野秀之, 中嶋秀治. 顧客へのリアルな音声応答を実現するテキスト音声合成技術「Cralinet」(特集 コンタクトセンタ業務に革新をもたらす音声処理技術). NTT 技術ジャーナル, Vol. 18, No. 11, pp. 19–22, 2006.

[152] Media Drive. 活字文書 OCR ライブラリ. http://mediadrive.jp/products/library/katsuji/index.html.

[153] Media Drive. 名刺認識ライブラリ. http://mediadrive.jp/products/library/meishi/index.html.

[154] Microsoft MSDN. Web Folder Behaviors. http://msdn.microsoft.com/en-us/library/ms531432.aspx.

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

[156] 小嶋秀樹. こじ研（小嶋研究室）宮城大学. http://www.myu.ac.jp/~xkozima/lab/avmed-pd1.html.

[157] Mark Danks. Graphics Environment for Multimedia. http://gem.iem.at/.

[158] Hiroki Akama, Masashi Yamamuro, Takashi Menjo, and Tetsuji Satoh. Integrating Multimedia Data Processing Parts in Cloud into Folder Programming Environment. In *Proceedings of The 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS2011)*, pp. 130–137, 2011.

[159] John McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *Communications of the ACM*, Vol. 3, No. 4, pp. 184–195, 1960.

[160] Guy L. Steele Jr. *Common LISP: The Language*. Digital Press, 2nd edition, 1990.（Guy L. Steele Jr. 他著，後藤栄一監訳，井田昌之訳. Common LISP. 共立出版, 1986）.

[161] Michael Sperber, R. Kent Dybvig, and Matthew Flatt. The Revised6 Report on the Algorithmic Language Scheme. http://www.r6rs.org/.

[162] Bil Lewis, Dan LaLiberte, and Richard Stallman. GNU Emacs Lisp Reference Manual. http://www.gnu.org/software/emacs/manual/elisp.html.

[163] 竹内郁雄. マルチパラダイム言語 TAO.
http://www.nue.org/nue/tao/bitao/.

[164] Hiroshi G. Okuno, Ikuo Takeuchi, Nobuyasu Osato, Yasushi Hibino, and Kazufumi Watanabe. TAO: A Fast Interpreter-Centered System on LISP machine ELIS. In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, LFP '84, pp. 140–149. ACM, 1984.

[165] 井田昌之, 元吉文男, 大久保 清貴編. Common LISP オブジェクトシステム -CLOS とその周辺. bit 別冊, 共立出版, 1989.

[166] Clojure.
http://clojure.org/.

[167] M. D. Conrad Barski. *Land of Lisp*. No Starch Press, 2011.
（Conrad Barski, M. D. 著，川合史朗訳. Land of Lisp. オライリージャパン, 2013）.

[168] W. F. Clocksin and C. S. Mellish. *Programming in Prolog: Using the ISO Standard*. Springer, 5 edition, 2003.

研究業績

博士論文に関する論文

査読付き論文誌

- **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.
- **赤間浩樹**, 鶴岡行雄, 佐藤哲司. 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. 情報処理学会論文誌 プログラミング, Vol. 6, No. 2, pp. 69–84, 2013.

査読付き国際会議論文集

- **Hiroki Akama**, Masashi Yamamuro, Takashi Menjo and Tetsuji Satoh. Integrating Multimedia Data Processing Parts in Cloud into Folder Programming Environment. Proceedings of the 13th International Conference on Information Integration and Web-based Applications& Services (iiWAS2011), Ho Chi Minh City, Vietnam, ACM, Dec. 2011, pp. 130-137.

その他の論文

査読付き論文誌

- 内藤一兵衛, **赤間浩樹**, 長谷川知洋. 追記・参照データ管理システムにおける分散蓄積データの再フィルタ方式. 日本データベース学会論文誌, Vol. 7, No. 4, pp. 43-48, 2009.

- **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, 兵藤正樹, 三浦史光, 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**, 三井一能, 串間和彦. コンテンツ埋込 DB と木状ビューの対応を使ったメタデータ管理インタフェースの設計と実装. 情報処理学会論文誌 データベース, Vol. 42, No. SIG 08, pp. 171-184, 2001.

- 吉田忠城, **赤間浩樹**, 山室雅司, 串間和彦, 楳谷浩章, 辻敏弘, 原田健次. 類似画像検索方式の改変画像検出への適用 -デジタルコンテンツ保護・流通システムへの適用と評価-. 情報処理学会論文誌 データベース, Vol. 42, No. SIG 01, pp. 171-181, 2001.

- 串間和彦, 佐藤路恵, **赤間浩樹**, 山室雅司. 大量画像の閲覧を目的とする階層的分類支援機能 - 画像目録の実装と評価-. 情報処理学会論文誌 データベース, Vol. 41, No. SIG 01, pp. 54-63, 2000.

- 吉田忠城, **赤間浩樹**, 谷口展郎, 山室雅司, 串間和彦. データ圧縮型多次元空間インデックス VA-TREE. 情報処理学会論文誌 データベース, Vol. 41, No. SIG 06, pp.1-11, 2000.

- **赤間浩樹**, 小西史和, 吉田忠城, 谷口展郎, 山室雅司, 串間和彦. 近傍検索向け転置ファイル法における外部キー検索と動的データ追加の実装と評価. 情報処理学会論文誌 データベース, Vol. 40, No. SIG 04, pp. 51-62, 1999.

- 串間和彦, **赤間浩樹**, 紺谷精一, 山室雅司. 色や形状等の表層的特徴量にもとづく画像内容検索技術. 情報処理学会論文誌 データベース, Vol. 40, No. 3, pp. 171-184, 1999.

- 串間和彦, **赤間浩樹**, 紺谷精一, 木本晴夫, 山室雅司. オブジェクトに基づく高速画像検索システム:ExSight. 情報処理学会論文誌, Vol. 40, No. 2, pp. 732-741, 1999.

- 三井一能, 寺西裕一, 小島明, **赤間浩樹**, 佐藤哲司, 寺中勝美, 鎌原淳三, 下條真司, 西尾章治郎. ニュースコンテンツの検索と編集機能を融合するニュースオンデマンド

システムの設計と実現. 電子情報通信学会論文誌 D-II, Vol. 81, No. 3, pp. 516-528, 1998.

査読付き国際会議

- Kazuhiko Kushima, Michie Satoh, **Hiroki Akama**, Masashi Yamamuro. Integrating Hierarchical Classification and Content-based Image Retrieval - ImageCompass -. Proceedings of the International Conference on Intelligent Information Processing, pp. 179-187, IFIP TC12, 2000.
- Kazuhiko Kushima, **Hiroki Akama**, Seiichi Kon'ya, Masashi Yamamuro. ExSight: Highly Accurate Object Based Image Retrieval System Enhanced by Redundant Object Extraction. Proceedings of the Web-Age Information Management, First International Conference (WAIM2000), Springer, Lecture Notes in Computer Science, Vol. 1846, pp.331-343, 2000.

査読付き国内会議

- **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) シンポジウム論文集, 6H-4, pp. 1722-1730, 2012.
- **赤間浩樹**, 内藤一兵衛, 内山寛之, 山中真和, 谷口展郎, 長谷川知洋, 三井一能, 山室雅司. フォルダ・プログラミングとネットワーク・フォルダ・サービス. 情報処理学会 マルチメディア、分散、協調とモバイル (DICOMO2008) シンポジウム論文集 6F-5, pp. 1435-1442, 2008.

研究会・全国大会発表

- **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の連携システム. 電子情報通信学会 パターン認識・メディア理解 (PRMU), 技術研究報告, Vol. 111, No. 77, pp. 61-66, 2011.
- **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 全国大会講演論文集, Vol. 72, No. 3, pp. "3-37"-"3-38", 2010.

- Tomohiro Hasegawa, Ichibe Naito, Takashi Menjo, Motohiro Matsuda, **Hiroki Akama**, Masashi Yamamuro. Data Stream ManagementTechnology for Accumulating and Processing Lifelogs, Platform Technologies for Services that Utilize Lifelogs. NTT Technical Review, NTT, Vol. 9, No. 1, pp. 1-5, 2011.
- 内藤一兵衛, **赤間浩樹**, 山室雅司. 分散データストリーム処理における自律 Pull 制御方式の改善. 情報処理学会 全国大会講演論文集, Vol. 72, No. 3, pp. "3-39"-"3-40", 2010.
- 長谷川知洋, 内藤一兵衛, 毛受崇, **赤間浩樹**, 山室雅司. ストリーム型ログデータ蓄積処理向け無停止 DB 分割方式の提案. 情報処理学会 マルチメディア通信と分散処理 (DPS) 研究報告, Vol. 2010-DPS-142, No. 48, pp. 1-6, 2010.
- 長谷川知洋, **赤間浩樹**, 山室雅司. 追記・参照型データ管理システムにおける Push ／ Pull 混在方式の特性評価. 情報処理学会 マルチメディア通信と分散処理 (DPS) 研究報告, Vol. 2009-DPS-138, No. 20, pp. 205-210, 2009.
- 内藤一兵衛, **赤間浩樹**, 谷口展郎, 山室雅司. タグツリーによる個人コンテンツ管理システム. 電子情報通信学会 総合大会講演論文集, Vol. 2009, No. 1, p.23, 2009.
- 内山寛之, **赤間浩樹**, 山室雅司. 分散データストリーム処理における適応型リソース制御方式の検討. 情報処理学会 マルチメディア通信と分散処理 (DPS) 研究報告, Vol. 2008-DPS-134, No. 21, pp. 49-54, 2008.
- 内山寛之, **赤間浩樹**, 西岡秀一, 内藤一兵衛, 谷口展郎, 長谷川知洋, 三浦史光, 山室雅司, 櫻井紀彦. 分散データストリーム処理アーキテクチャの提案. 情報処理学会 データベースシステム (DBS) 研究報告, Vol. 2007-DBS-143, No. 65, 2007, pp. 327-332.
- **赤間浩樹**, 内山寛之, 三浦史光, 西岡秀一, 内藤一兵衛, 谷口展郎, 山室雅司, 櫻井紀彦. 追記・参照型データ管理プラットフォームアーキテクチャの提案. 情報処理学会 マルチメディア通信と分散処理ワークショップ (DPSWS2006) シンポジウム論文集, pp. 199-204, 2006.
- 吉田忠城, 小西史和, **赤間浩樹**, 山室雅司. データ圧縮型インデックス VA-TREE の検討. 情報処理学会 データベースシステム (DBS) 研究報告, Vol. 2000, No. 10, pp. 29-36, 2000.
- 紺谷精一, **赤間浩樹**, 山室雅司. 画像からの直線検出と直線をキーとした画像の検索. 情報処理学会 データベースシステム (DBS) 研究報告, Vol. 1998, No. 57, pp. 1-8, 1998.

- 梅田昌義，山室雅司，鬼塚真，小林伸幸，三井一能，**赤間浩樹**，串間和彦．画像検索処理における問い合わせ言語への要求．情報処理学会 データベースシステム (DBS) 研究報告, Vol. 1998, No. 2, pp. 1-5, 1998.

- 紺谷精一，**赤間浩樹**，三井一能，串間和彦．エッジ検出と領域融合によるカラー画像のセグメンテーション． 情報処理学会 データベースシステム (DBS) 研究報告, Vol. 1997, No. 64, pp. 161-166, 1997.

- **赤間浩樹**，紺谷精一，三井一能，串間和彦．画像内オブジェクトの自動抽出を使った画像検索システム ExSight -写真 (PhotoDisk) への適用-．情報処理学会 データベースシステム (DBS) 研究報告, Vol. 1997, No. 64, pp. 155-160, 1997.

- **赤間浩樹**，石垣昭一郎．ダウンサイジングにおけるデータベース利用の課題と対策．情報処理学会 データベースシステム (DBS) 研究報告, Vol. 1994, No. 44, pp. 35-41, 1994.

- **赤間浩樹**，武田英昭．RDBMS 設計に対するオブジェクト指向技法の適用．情報処理学会 データベースシステム (DBS) 研究報告, Vol. 1993, No. 77, pp. 1-10, 1993.

- 三井一能，寺西裕一，**赤間浩樹**，佐藤哲司，寺中勝美，八幡孝，菅野昭博，鎌原淳三，下條真司，西尾章治郎，宮原秀夫．シナリオデータベースによるニュース・オン・デマンドシステムの実現． 電子情報通信学会 データ工学 (DE) 技術研究報告, Vol. 1996, No. 54, pp.7-12, 1996.

- 小西 史和, **赤間 浩樹**．関連情報連想システムにおける検索観点自動提示方式．情報処理学会 全国大会講演論文集, Vol. 50, No. 4, pp. 31-32, 1995.

- **赤間浩樹**, 峯崎俊哉, 成嶋弘．帰納型プロダラムの機械学習システム『Ｂくん』．情報処理学会 全国大会講演論文集, pp. 351-352, 1990.