情報処理学会論文誌 Vol. 50 No. 8 1798–1809 (Aug. 2009)

推薦論文

フォルダ・プログラミング環境「POLDER」

赤間 浩樹[†1] 内藤 一兵衛[†1] 毛受 崇[†1]
長谷川 知洋[†1] 谷口 展郎[†1] 山室 雅司[†1]

我々は，幅広い層のエンドユーザにとって気軽に処理部品群を活用できる環境であり，さらにプログラマにとってはWebのマッシュアップ環境としても活用可能な情報処理基盤の創出を目指している．本論文では，フォルダというメタファを使ってデータフロー処理の視覚化を行い，「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プログラミング初心者にも簡単な処理構成・実行環境を提供するプログラミング環境POLDERの提案を行い，関連技術との比較により効果を考察する．

POLDER: Folder Programming Environment

Hiroki Akama,[†1] Ichibe Naito,[†1] Takashi Menjo,[†1]
Tomohiro Hasegawa,[†1] Noburou Taniguchi[†1]
and Masashi Yamamuro[†1]

We aim to create an information processing environment that allows many end users to combine processing parts easily. Additionally, the environment can be used by programmers to mashup Web programming modules. This paper proposes the programming environment named POLDER; it visualizes data flow by using the folder metaphor. POLDER gives programming beginners an easy programming and execution environment by applying the following simple processing principle; data input is realized by dropping a data object into a folder via DROP; the input data object is processed by the program bound to the folder; the resulting data object is stored in the same folder. Setting a layered folder structure chains the processes of the folders. We also describe a comparison with a conventional programming environment.

1. はじめに

近年，インターネット上のサービスをマッシュアップするための各種APIが提供されつつある．しかしREST[1),2)]やSOAP[3)]などを中心としたそれらのAPIは通常のプログラマによる利用を想定している．

本研究は，これまでプログラミングには馴染みのなかったような幅広い層のエンドユーザ（プログラム初心者）がWeb上の処理部品群を気軽に利用可能な情報処理基盤を提供することを目指している．同時に，その基盤はプログラム初心者が利用するだけにとどまらず，その技能の延長上の姿として存在する通常のプログラマにとっても様々なWeb上の部品を組み合わせるマッシュアップ環境として活用可能な情報処理基盤となっていることが望ましいと我々は考えている．

本論文では，2章でその情報処理基盤を実現するために我々の提案するフォルダ・プログラミング環境[4)]を紹介し，3章で現在の設計と実装および応用例を紹介し，4章にて代表的な関連技術との比較を通してフォルダ・プログラミング環境の効果について考察する．

2. フォルダ・プログラミング環境

フォルダ・プログラミング環境の重要な構成要素として処理付きフォルダがある．そこで，まず処理付きフォルダを説明し，その後，フォルダ・プログラミング環境について，プログラム実行制御機能，データ構造，プログラム開発支援機能の各側面から説明を行う．

2.1 処理付きフォルダ

処理付きフォルダを次の3つの特徴的機能で説明する．

- フォルダへのファイル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．
- フォルダ名内にパラメータを持つ．
- 複数のファイル群の一括DROPによっていっせい実行する．

†1 日本電信電話株式会社NTTサイバースペース研究所
NTT Cyber Space Laboratories, NTT Corporation
本論文の内容は2008年7月のマルチメディア，分散，協調とモバイル（DICOMO2008）シンポジウムにて報告され，同プログラム委員長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

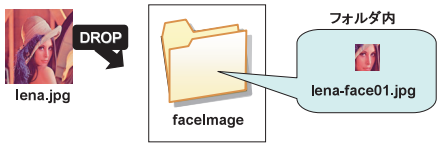

図 1　フォルダへのデータ DROP で処理が起動
Fig. 1　Process starts by Data DROP to folder.

(1)　フォルダへのファイル 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

Microsoft 社の MS 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していた．

一方，処理付きフォルダでは，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．以下，処理フローの概要を簡単に示す．まず事前に各処理名に対してはフォルダ・プログラム処理系内で処理実体が指定されている．処理付きフォルダではフォルダ名の一部が処理名を表している．ここで，その処理付きフォルダにデータファイルを DROP すると，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．そして，その処理結果がその処理付きフォルダ内に格納される．

図 1 にその例を示す．あらかじめ，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 という名称を持つ処理付き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，たとえば顔画像ファイル lena.jpg [5] をその faceImage フォルダに DROP すると，lena.jpg に/opt/bin/extractFaces.pl が適用され，得られた処理結果 lena-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 lena.jpg ファイルは削除される．これが処理付きフォルダの基本機能である．

(2)　フォルダ名内にパラメータを持つ

処理付きフォルダでは，フォルダ名の前半（最初に出現する空白以前）が処理名として処理実体の指定に使われる．さらに，ファイル名の残りの部分については処理実体に渡すパラメータとして利用することができる．

図 2 に簡単な例を示す．textGrep というフォルダ名には，処理実体としてテキストファイルからパラメータで指定される特定文字列を含む行を抽出する処理/bin/grep が指定されているとする．このとき，フォルダ名「textGrep 'ABC'」は，パラメータを 'ABC' として/bin/grep を実行する処理となり，このフォルダに DROP されたデータファイルに適用された結果はフォルダ「textGrep 'ABC'」内に生成される．なお，フォルダ名の一部として空白が利用可能であることが必要である．また，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．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，パラメータ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 [6] やプログラミング言語 Perl [7] 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，パラメータの評価の中でファイル内の XML タグに割り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，パラメータ内のパス表現として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．

図 2　フォルダ名内にパラメータを持つ
Fig. 2　Folder name includes parameters.

　　* → #，/ → %，" → '

たとえば，画像処理ライブラリとして有名な Image Magick [8),9)] の画像フォーマット変換コマンド convert が処理実体およ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 タグおよび Height タグの持つ値をパラメータ域に展開して-resize オプションを実行させる場合には「convert -resize `V ..%data%A.xml%Width`x`V ..%data%B.xml%Heigth`」というフォルダ名とする．この V コマンドのパス解釈では処理付き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（たとえば 100）を取得しそのパラメータ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（たとえば 200）を取得しそのパラメータ文字列内位置に展開する．各 V コマンドの評価後のパラメータは「-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 100 pixel × 縦 200 pixel に変換すること）が行われる．V

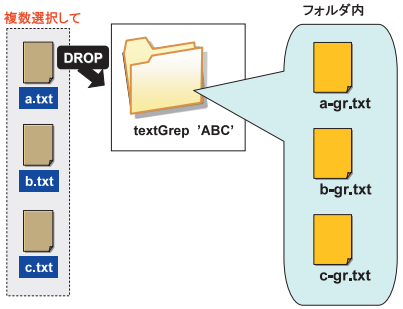

図 3　複数ファイル群の一括 DROP
Fig. 3　DROP at the same time.

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．

(3)　複数のファイル群の一括 DROP によっていっせい実行する

フォルダ操作の GUI では複数のデータファイルを選択してフォルダに一括 DROP することができる．その DROP 先が処理付きフォルダであった場合，各々のデータファイルに対してフォルダ名から決定された処理実体が実行され，処理結果の各々がフォルダ内に格納される（ただし，処理結果が同一ファイル名になるものについては上書きされるので，どの処理結果が残るか保証されない）．

図 3 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 DROP した場合，各々のデータファイルへの/bin/grep 'ABC' の処理結果 a-gr.txt，b-gr.txt，c-gr.txt がフォルダ内に生成される．ファイル名の変換は textGrep が行っている．

2.2　フォルダ・プログラミング環境

前述の複数の処理付きフォルダを組み合わ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．以下フォルダ・プログラミング環境について，フォルダ・プログラム実行制御機能，データ構造，フォルダ・プログラム開発支援機能を説明する．

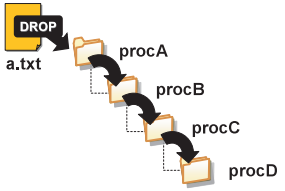

図 4　処理連鎖
Fig. 4　Chain of processing.

2.2.1　フォルダ・プログラム実行制御機能

(1)　処理結果の下位フォルダへの投入連鎖による処理の連鎖

フォルダ・プログラミングの基本は，処理付きフォルダの入れ子構造によって処理フローを構成することである．このフォルダの入れ子構造を処理フローとするために，フォルダ・プログラムの実行制御機能では投入連鎖によって処理連鎖を実現する．これは，フォルダ・プログラムの上位フォルダへのデータファイルの DROP（またはデータファイルの投入に相当する操作）によって上位フォルダに対応した処理が実行されたその結果が，さらにその下位の処理付きフォルダへの投入（DROP 相当）として下位に流れていくという連鎖であり，データフロー処理ともいえる．

図 4 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処理は停止する．なお，各層で処理付き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．

(2)　複数並行実行

通常フォルダ内にはさらに複数のフォルダを格納することができる．投入連鎖において，投入対象となる処理付きフォルダが複数存在した場合には，それら複数の処理付きフォルダへのデータ投入となる．これは並行処理の記述ともいえる．なお，その際の処理順序に規定はない．

図 5 に例を示す．procA の処理結果ファイルは，procB，procC，procD の 3 つの処理付きフォルダに投入され，処理結果は 3 つに分かれる．

(3)　条件分岐

条件分岐は，前述の複数並行実行の特殊な例として実現される．処理付きフォルダ if は条件をパラメータとして持ち，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

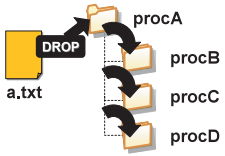

図 5　複数並行実行
Fig. 5　Concurrent processing.

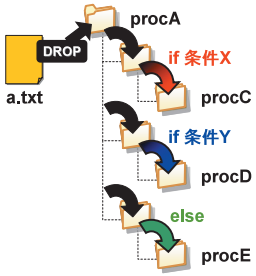

図 6　条件分岐
Fig. 6　Branch on condition.

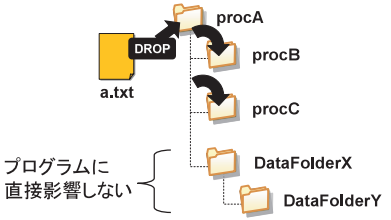

図 7　データフォルダと処理の混在
Fig. 7　Coexistence of data folder and processing.

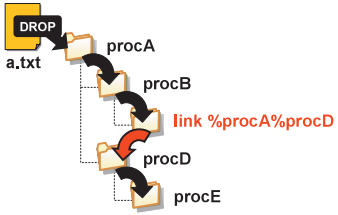

図 8　リンク
Fig. 8　Link.

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パラメータの異なる複数の if フォルダが存在する場合，各々の if フォルダは複数並行に実行される．よって，いずれか 1 つの if フォルダに合致させたい場合には条件を排他的に記述しなければならない．また，特殊な処理付きフォルダとして else フォルダがある．else フォルダは同階層のいずれの処理付きフォルダにもデータが投入されなかったデータファイルが，else フォルダへの投入データファイルとなる特別な制御を行う．

図 6 に例を示す．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．

(4)　データと処理の混在

処理付きフォルダ内には，通常のデータフォルダが混在して存在することができる．フォルダ名に処理の指定がないフォルダについては，投入連鎖は発生しない．

図 7 において，procA の中には，処理付きフォルダ 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 および 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（2.1 節 (2) で述べたようにパラメータから参照されるケースなどはありうる）．

(5)　リンクによる制御移行

図 8 に例を示すように，処理付きフォルダ link は，パラメータで指定された他のフォルダに対してデータを投入する．link は通常のプログラミング言語での GOTO 文にも相当するので，link と if によって原始的な繰返し処理を実現することが可能になる．

2.2.2　データ構造

データの持つ構造（通常のプログラミング言語でのレコード型に相当）についてもフォルダを使って表現・構成する．処理の入力として複数のデータファイルを受け取る場合には，

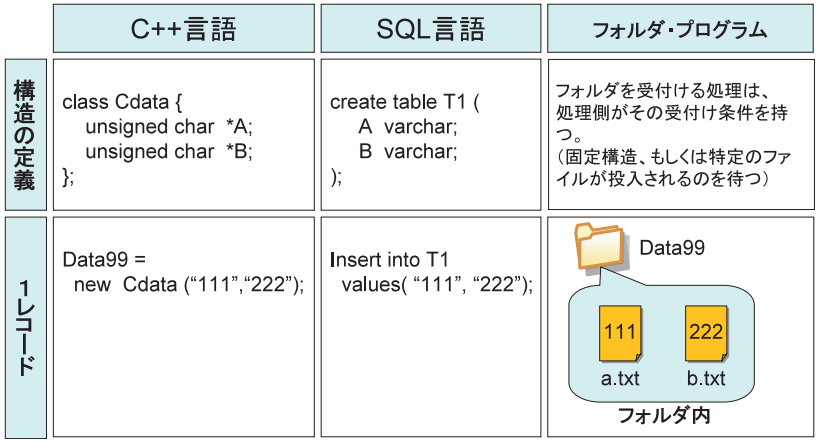

図 9　データ構造
Fig. 9　Data structure.

この表現によるデータ構造を直接利用するか，もしくはそれらを ZIP などでまとめたファイルとして利用する．

図 9 は 2 つのデータ A，B から構成されるデータ構造について，C++言語および SQL 言語と表現の比較を行っている．フォルダ・プログラムでは，a.txt，b.txt からなる構造 Data99 をフォルダとして構成している．どのような構造を受け取るかについては，受付条件を各処理実体側で持つ．

2.2.3　フォルダ・プログラム開発支援機能

(1)　DEBUG，ERROR 処理

図 10 に示すように，処理付きフォルダでは，処理の入力がフォルダの外側にあり，処理の結果がフォルダの内側に入る．よって，処理の受付に失敗したなどのエラーデータについては，フォルダの外側に弾き返されるイメージとなり，フォルダの外側に入力ファイルが “ERROR. 投入ファイル名” と改名されて生成される．必要であればプロセス番号などのシステム ID が名前中に含まれる．

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．つまり，フォルダ階層の途中にデータを投入すれば，それ以下の階層のみが実行され，ある階層のフォルダ名を処理未定義のフォルダ名に改名して実行すれば，そのフォルダの直前までの実行結果が直上のフォルダ内に得られる．また，フォルダ・プログラムの一部

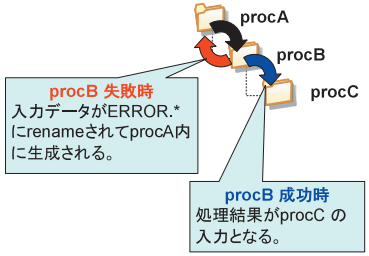

図 10　DEBUG，ERROR 処理
Fig. 10　DEBUG, ERROR processing.

分を別フォルダ配下にコピーして実行することや，単独のフォルダだけを実行させることも可能である．この自由度の高い部分実行性およびその入出力のファイルへの統一による可視容易性によって，デバッグが容易になる．

(2)　HELP

処理の具体的な使用法などの情報が必要な場合には，以下の 2 通りの方法によって HELP 情報（HTML-処理名.html）を当該処理名フォルダ内に得ることができる．

HELP 方式 1：処理名に対するパラメータとして HELP を指定したフォルダに，任意のファイルを 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 DROP する．

(3)　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）に変換することで，テキストのみで表現することが可能になる．フォルダ・プログラムにおいて入力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．具体例は 3.5 節に後述する．これにより，フォルダ・プログラムの GUI 上でのグラフィカルな表現とテキスト表現を相互に等価変換可能になり，プログラムの転送や言語学習などを容易にする．

3.　フォルダ・プログラミング環境の設計と実装

3.1　Web フォルダ・システムとしてのアーキテクチャ

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際のアーキ

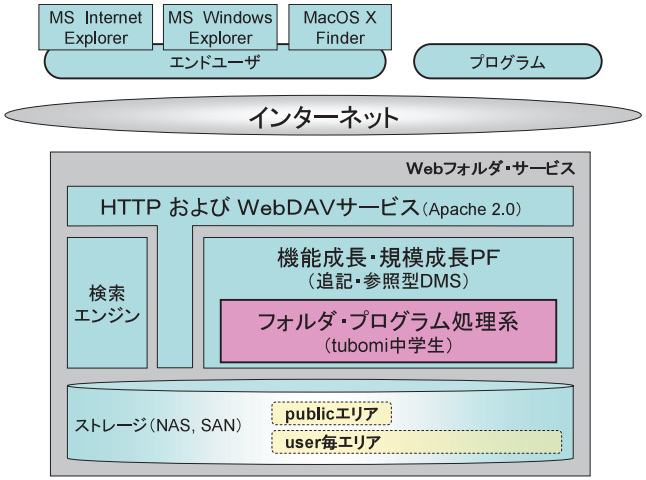

図 11　Web フォルダ・サービスのアーキテクチャ
Fig. 11　Web folder service architecture.

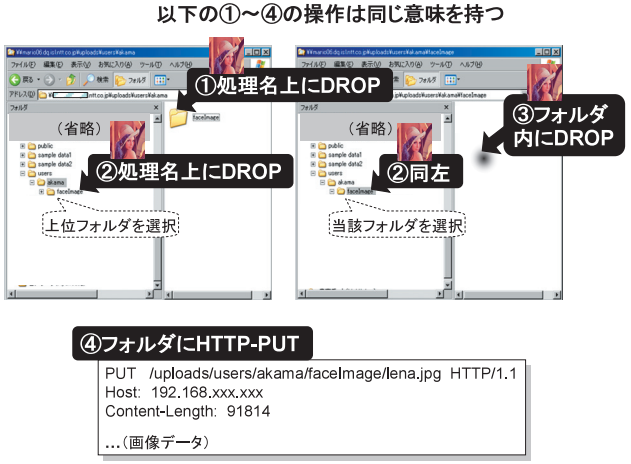

図 12　フォルダへの DROP および HTTP-PUT
Fig. 12　DROP or HTTP-PUT to folder.

テクチャを**図 11** に示す．Web 上のフォルダとして機能するために WebDAV を利用する．

WebDAV[10),11)] とは，RFC4918（当初は RFC2518）によって規定された HTTP 拡張によってネットワーク・ファイルシステムを実現するプロトコルであり，HTTP1.0[12)]，HTTP1.1[13)] と合わせて以下のような機能を提供する．ここでリソースがファイル，コレクションがフォルダと考えると理解しやすくなる．

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

WebDAV はすでに広く普及しており，HTTP サーバの Apache[14)]，Microsoft Windows の Explorer や Internet Explorer，MacOS X の Finder などもサポートしている．我々の実装では，Web フォルダ・システムのサーバとして Apache2.0 を利用している．サーバ側では HTTP-PUT に対応して，3.3 節に述べるフォルダ・プログラム処理系（開発コード：tubomi 中学生）を呼び出すことで実現している．ネットワーク・フォルダの実現方法としては，WebDAV 以外にも社内ネットワークなどでよく利用される CIFS[15),16)]（もしくは Samba[17)]）やユーザ拡張可能なファイルシステム FUSE（File system in Userspace[18)]）を拡張するアプローチも存在する．いずれも対象利用者が限定される場合や HTTP を避けたい場合などに有効である．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 個人フォルダを使って解決するが，より強力な認証が必要な場合には，NGN（Next Generation Network[19)]）で提供する回線認証の仕組みの併用が有効である．また，大規模なマッシュアップ環境としても活用可能な情報処理基盤の実現のためには，多様な処理に対して動的な機能成長・規模成長を可能にする追記・参照型データ管理システム[20)] をフォルダ・プログラム処理系の大規模分散環境として活用する．

3.2　フォルダへの DROP と Web インタフェースおよびその活用例

フォルダへの DROP と一口にいっても，実は**図 12**（MS Explorer での表示例）に示すような複数のパターンが存在する．まず，フォルダ上に DROP するケース ①，および，ツリー表現時においてフォルダ上に DROP するケース ② では，その DROP 対象フォルダ名から動的に決定される処理が実行される．ツリー表現において対象フォルダ自体を選択した場合には，右側の表示エリアにはその内容が表示される（ケース ③）．その場合には，右側表示エリアの空きスペースに DROP することで処理が実行される．以上の 3 つのケースで，すべて同一の挙動となる．

さらに，我々のフォルダ・プログラミング環境では WebDAV 上で実装しているため，前述

の 3 つのケースのほかに，フォルダ内に HTTP-PUT することでも同じ処理が実行される．この Web インタフェースを使うと，プログラマは REST（Representational State Transfer [1),2)]）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．その例を以下に示す．なお，HTTP-MKCOL などを使用し，HTTP-PUT の独自拡張を利用する本手法を REST スタイルとは厳密には呼べない．また，必要に応じて WebDAV の HTTP-LOCK/UNLOCK [10),11)] を使うことも可能である．

【Web インタフェースの活用例】

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
...（2,800 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．

（繰返しの実現は JavaScript などを活用）

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

3.3　フォルダ・プログラム処理系の実装

フォルダ・プログラムの処理系は，Perl で実装している．処理系の中には処理名と処理実体を関係付けるテーブルを持っている．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は，処理実体名および次に示す 4 つのパラメータからなり，各処理は別プロセスとして実行される．

-c 処理名
-p パラメータ（上記処理名以外の部分）
-o 出力ファイル名が格納されるファイル名
投入データファイル名（フルパス）

処理の連鎖は，このフォルダ・プログラム処理系自身の再帰呼び出しが基本となり，連鎖先が外部サイトの場合には HTTP-PUT に切り替わる．

3.4　処理付きフォルダの実装（部品）

処理付きフォルダとしては，下記のような各種のバリエーションの処理実装を試しながら，プログラミング・スタイルの確立に向けて試行中である．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てある．

- **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：パラメータを評価し，ファイル化
  fileRename：投入ファイル名を変更
  putText：指定 URL にパラメータを PUT
  link：指定 URL に投入ファイルを PUT
  if：パラメータの条件を満たせば投入連鎖

各処理部品は Linux のコマンドとして実装されている処理を bash もしくは Perl によって処理付きフォルダの部品用にラッピングすることで対応する．既存のものと同インタフェースの部品の拡充であれば，数行の変更で済む．さらに，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．

3.5　データ分類への応用事例

応用事例については，すでにここまでにいくつか述べてきているが，**図 13** 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．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

```
users
  demo
    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
    RESULT
```

図 13　フォルダ・プログラムとフォルダ構造の例
Fig. 13　Example of program and folder structure.

称で/users/demo/RESULT フォルダ内に結果を格納する．

4. 関連技術との比較

プログラミング言語や環境の評価は難しく，定式化された評価基準は存在しない．そこで本論文では，フォルダ・プログラミング環境のいくつかの機能や狙いに着目して，それらの代表的と思われる既存技術との差異を述べ，フォルダ・プログラミング環境の効果を明らかにすることを試みる．

4.1 MacOS X のフォルダ・アクション機能

フォルダへのファイルの DROP で処理が実行される機能は，MacOS X[21] でも実現できる．MacOS X では，多様なイベントに対して AppleScript 言語[22] で書かれた処理を割り当てることが可能であり，フォルダへの DROP 操作のイベントに対してユーザ作成の処理を割り当てた場合に，前述の機能を実現したこととなる．しかし，その場合のプログラムはフォルダ階層ではなく AppleScript 言語で記述しなければならないという点でプログラミング初心者向きではなく，さらに以下に述べる点も大きく異なっている．

- 投入連鎖による処理連鎖がないためフォルダ階層を使ったプログラミングにはなっていない．
- 具体的処理（AppleScript にて記述）とフォルダ実体の対応が固定されていて，フォルダ名を変えることで処理を変えるという手軽さがなく，プログラミング初心者向きではない．
- フォルダ名の一部をパラメータとすることはないのでフォルダ名だけからの処理選択の自由度が低い．

4.2 UNIX パイプ機能

増井[23] によると UNIX のパイプ機能[6),24)] も単純なデータフロー処理といえる．一方，我々の提案するフォルダ・プログラミング環境も，一種のデータフロー処理を可能にする．フォルダ・プログラミング環境における処理付きフォルダを UNIX 上でのコマンドと考えて対応付けることも自然にでき，フォルダ・プログラミング環境は，UNIX パイプの拡張系と位置付けることができる．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などのより高度な処理記述と応用範囲の拡大も可能にした．さらに，入力データに構造を持てる点や 4.4 節で述べるように連鎖の途中段階で処理を停止・データを参照・再開できる点，プログラムがつねに保存・永続化されている点など，フォルダ・メタファを中心とした多様な拡張となっている．

4.3 データフロー系ビジュアル・プログラミング機能

2007 年に一般公開された Yahoo! Pipes[25] は，データフロー処理を Web 上でビジュアルにプログラミングする環境を提供し，RSS（Rich Site Summary）などの加工処理に広く支持されている．増井[23] の指摘するビジュアル・プログラミングとしてのいくつかの課題を残しつつも，入力を RSS や特定の Web 上のサービスに限定することで十分実用的な環境となっている．我々のフォルダ・プログラミング環境もデータフロー処理を対象とする点で類似しており，並行処理や条件分岐が可能な点や入力データに構造が持てる点でも類似する．以下に，フォルダ・プログラミング環境が持つ優位点を列挙する．

- フォルダ・プログラミング環境では，ビジュアル・プログラミングとしての自動レイアウトやズーミング・インタフェース（「全体像の把握」，「部分を詳細に注目」の切替え）がフォルダ・メタファの木状表示として提供される．一方，Yahoo! Pipes には自動レイアウトもズーミング・インタフェースもない．
- 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）を使いこなす必要があるため，あくまでも一般

プログラマ向けの開発環境である．一方，フォルダ・プログラミング環境では，日常利用しているフォルダ・メタファを活用するためプログラミング初心者にとって新たなGUIを学び慣れるという負担なく，簡単な処理構成と処理実行環境を提供する可能性がある．

- フォルダ・プログラミング環境により生成されたプログラムは2.2.3項(3)および3.5節図13で紹介した形式でテキスト化が可能である．このグラフィカル表現とテキスト表現のシンプルな相互変換性は，プログラムの転送や言語学習などを容易にする．
- フォルダ・プログラミング環境はファイル（群）を処理対象とする処理実行環境であり，RSSやXML処理に限定されない処理を記述することができる．

4.4　エンドユーザ向けのプログラミング言語・環境として

エンドユーザ・プログラミングを指向する言語としてチャミー[26]がある．また，MacOS XのAutomator[27]もユーザの操作の記録を活用したエンドユーザ・プログラミングを指向していると考えられる．フォルダ・プログラミング環境はこれらのいずれともアプローチが異なるが，チャミーの狙いを参考にしつつ，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．なお，ここでいうエンドユーザとは，これからプログラミングを学ぶレベルのプログラミング初心者を想定している．

- **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」などの分かりやす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．
- **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でも簡単に導入可能だと思われる．
- **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きわめて少ないといえる．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．この特長およ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）よりもさら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かか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持つことが可能になる．
- **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つの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とどまるため，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．
- **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替えや，1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み込み作業が存在した．フォルダ・プログラミング環境では，プログラムがつね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み込みなどを意識しなくてよい．さらに，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．
- **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とることになる．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．さらに，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，複数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きフォルダを探し出す方法も可能である．
- **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，複数のファイルをいっせいに選択し，ファイルに一括DROPすることで，その各々に対する処理をいっせいに実施することができ，たとえば，大量の画像ファイルに対するサイズ変更処理が，きわ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．たとえば，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きフォルダが 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）させることが可能になる．

4.5　既存プログラマ向けのプログラミング言語・環境として

チャミー[26] では，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．以下では，フォルダ・プログラミング環境について，既存のプログラマから見た視点から考察する．

- C++[31]，Perl[7] など一般的なプログラミングに慣れたプログラマから見て，データフロー系のプログラミングは，UNIX パイプ[6),24)] や，Yahoo! Pipes[25] など，すでに多くのプログラマに賛同されている実績があるので，違和感は少ないと考える．また，フォルダへの処理の指定（処理付きフォルダ）についても UNIX パイプを許容できるプログラマにとって違和感はないと考える．
- Web を対象としたプログラミングとの違和感については，フォルダ・プログラミング環境がすでに WebDAV[10),11)] 上で実装されていることからも分かるように少ないと予想できる．ただし，従来 HTTP-POST[12] と CGI（Common Gateway Interface[32]）を中心としていたスタイルを HTTP-PUT[13] のスタイルに変更する点に若干違和感が残る可能性はある．しかし，CGI を経由せず Web 上の部品群を直接利用できるようになるなど，より Web-API の利用が簡単になる効果の方が違和感に勝ると推測する．
- Lisp[33] や Prolog[34] などの人工知能的なプログラミング言語についても，フォルダ・プログラミング環境は比較対象となりうる．データ構造とプログラム構造が同一表現であり，動的にプログラム構造を構成（たとえば WebDAV の HTTP-MKCOL[10),11)] によって実行時にフォルダを作成）しながら処理を進めることができ，また，プログラム構造を知識ベースとして永続化し保管することができる点で，人工知能的なプログラミングの環境となりえ，さらに個人のデスクトップの情報との連携もしやすくなるため，この分野でも今後の応用が期待される．

4.6　フォルダ・プログラミング環境の課題

フォルダ・プログラミング環境は

- データフロー系のプログラミング環境
- ビジュアル・プログラミング環境

をフォルダという多くのユーザが慣れ親しんだメタファを使って統合したものといえる．プログラム初心者であるエンドユーザに対しては 4.4 節で述べたように，母国語対応，基本知識・原則の少なさ，初期実行や初期編集までの学習量の少なさ，モードの少なさ，プログラム部品の探しやすさ，実用的場面での活用性など，の効果を提供する．また，既存のプログラマに対しても 4.5 節で述べたように，データフロー系の処理の違和感は少なく，WebDAV を通してマッシュアップ環境としても利用可能という広がりを持っている．

しかし，プログラミング言語としての部分には，まだ不足する機能が多い．プログラミング能力については簡単に考察すれば，たとえばファイル内の各行を変数ととらえ，代入，参照，そして各種演算などの部品を組み込み，2.2.1 項で紹介した if や link（GOTO）を使って分岐と反復を実現すれば，万能チューリングマシンと等価な計算能力は持てるだろう．もしくは，本論文では述べなかったがユーザ関数定義と再帰呼び出しを導入することでも同様の能力を持つアプローチも存在する．しかし，これまでプログラミングを自ら行わなかったユーザ層が本プログラミング環境をまず単機能の処理実行環境として使い出し，活用し，その延長線上に徐々に高度なプログラミング環境があるようにするためには，どのようなプログラミング部品を用意すればよいか？　また，どのようなプログラミング・スタイルが望ましいか？　など，今後，さらに検討を続けなければならない．

5. おわりに

本論文では，我々の提案するフォルダ・プログラミング環境について紹介を行った．それはフォルダというメタファを使ってデータフロー処理の視覚化を行い，「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プログラミング初心者にも簡単な処理構成・実行環境を提供するプログラミング環境である．さらにプログラマにとっては Web のマッシュアップ環境としても活用可能な能力を持っている．我々はそのフォルダ・プログラミング環境を代表的な既存技術と定性的に比較し，その有効性の考察を行った．

現在のフォルダ・プログラミング環境は，その処理系の拡充を継続中であり，まだ，プログラミング・スタイルの確立に向けての模索を継続中である．今後は，多数の利用者にフォルダ・プログラミング環境を利用してもらうことで，その利用スタイルの確立と効果検証を行い，ネットワーク・フォルダを中心としたマッシュアップ・サービスのための情報処理基盤の実現に向けて研究開発を進めていきたい．

謝辞　萌芽的である本研究の発表およびデモに際して，貴重なアドバイスをいただいた DICOMO2008 発表会場の皆様方および座長の上原稔先生，また，本研究の将来の可能性と

有用性を深くご理解いただき，論文誌推薦していただいた DICOMO2008 プログラム委員会の皆様方，および，論文構成などに多数のご指摘をいただいた査読者の皆様方に，心より感謝します．

参 考 文 献

1) Fielding, R.T. and Taylor, R.N.: Principled Design of the Modern Web Architecture, *ACM Trans. Internet Tech.*, Vol.2, No.2, pp.115–150 (2002).
2) Richardson, L. and Ruby, S.: RESTful Web サービス，オライリー・ジャパン (2007).
3) W3C: W3C Technical Reports and Publications SOAP. 入手先 http://www.w3.org/TR/（参照 2009-03-25）
4) 赤間浩樹，内藤一兵衛ほか：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，pp.1435–1442 (2008).
5) SIDBA (Standard Image Data-BAse): Standard test Image. 入手先 http://en.wikipedia.org/wiki/Standard_test_image（参照 2009-03-25）
6) Newham, C. and Rosenblatt, B.: 入門 bash，オライリー・ジャパン (1998).
7) Schwartz, R.L. and Phoenix, T.: 初めての Perl，オライリー・ジャパン (2003).
8) ImageMagick. 入手先 http://www.imagemagick.org/（参照 2009-03-25）
9) Still, M.: *The Definitive Guide to ImageMagick*, Apress (2005).
10) WebDAV, RFC4918. 入手先 http://www.ietf.org/rfc/rfc4918.txt（参照 2009-03-25）
11) 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 (2004).
12) HTTP1.0, RFC1945. 入手先 http://www.ietf.org/rfc/rfc1945.txt（参照 2009-03-25）
13) HTTP1.1, RFC2616. 入手先 http://www.ietf.org/rfc/rfc2616.txt（参照 2009-03-25）
14) Apache: HTTP SERVER PROJECT. 入手先 http://httpd.apache.org/（参照 2009-03-25）
15) Microsoft: CIFS (Common Internet File System). 入手先 http://msdn.microsoft.com/en-us/library/aa302188.aspx（参照 2009-03-25）
16) 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).
17) Samba. 入手先 http://us3.samba.org/samba/（参照 2009-03-25）
18) FUSE (Filesystem in Userspace). 入手先 http://fuse.sourceforge.net/（参照 2009-03-25）
19) NGN (Next Generation Network). 入手先 http://www.ntt-west.co.jp/open/ngn/interface.html（参照 2009-03-25）
20) 赤間浩樹，内山寛之ほか：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).
21) Apple: MacOS X. 入手先 http://www.apple.com/jp/macosx/（参照 2009-03-25）
22) Neuburg, M.: *Applescript: The Definitive Guide*, Oreilly & Associates (2006).
23) 増井俊之：ビジュアル・プログラミング，インターフェイスの街角—本当に使いやすいユーザ・インターフェイスの極意（UNIX MAGAZINE COLLECTION），アスキー，pp.72–77 (2005). 入手先 http://www.pitecan.com/UnixMagazine/PDF/if9809.pdf（参照 2009-03-25）
24) Tompson, K. and Ritchie, D.M.: The UNIX Time-Sharing System, *Comm. ACM*, Vol.17, No.7, pp.365–375 (1974).
25) Yahoo! Pipes. 入手先 http://pipes.yahoo.com/（参照 2009-03-25）
26) 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 PPL2005．入手先 http://www.graco.c.u-tokyo.ac.jp/ppl2005/papers/1-ichisugi.pdf（参照 2009-03-25）
27) Apple: Automator. 入手先 http://www.apple.com/jp/remotedesktop/automation.html（参照 2009-03-25）
28) Microsoft: Web Folder Behaviors. 入手先 http://msdn.microsoft.com/en-us/library/ms531432.aspx（参照 2009-03-25）
29) MPEG-2, ISO/IEC 13818 (1995).
30) H.264, MPEG-4 Part 10 Advanced Video Coding, ISO/IEC 14496-10 (2003).
31) Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley Publishing (1986).
32) CGI, RFC3875. 入手先 http://www.ietf.org/rfc/rfc3875.txt（参照 2009-03-25）
33) Steele Jr., G.L., et al.: *Common Lisp: The Language*, 2nd ed., Digital Press (1990).
34) Pereira, L.M., Pereira, F.C.N. and Warren, D.H.D.: *User's Guide to DEC System-10 Prolog*, Dept. of Artificial Intelligence, Edinburgh (1978).

(平成 21 年 1 月 6 日受付)
(平成 21 年 5 月 13 日採録)

推 薦 文

本論文は，フォルダを特定の処理に対応させ，フォルダを組み合わせることで任意の処理を表現するプログラミング手法「フォルダ・プログラミング」を提唱している．また，その手法をネットワーク・フォルダを用いたサービスとして実装した．Web サービスをパイプで視覚的に組み合わせる手法と同様に，初心者でも比較的容易に高度なサービスを実現することができる．斬新な発想と新規性は論文誌の推薦論文としてふさわしい．

（マルチメディア，分散，協調とモバイル（DICOMO2008）シンポジウム
プログラム委員長　串間和彦）

赤間　浩樹（正会員）

1990 年東海大学大学院理学研究科数学専攻修士課程修了．同年日本電信電話株式会社入社．ネットワーク向け DBMS，ニュース・オン・デマンド，マルチメディア情報検索の研究開発，FLET'S 系 PF の実用化等を経て，現在，情報統合管理技術の研究開発に従事．電子情報通信学会，日本データベース学会，人工知能学会，ACM 各会員．DICOMO2008 における本研究発表にて最優秀プレゼンテーション賞および最優秀論文賞受賞．

内藤一兵衛

2006 年早稲田大学大学院理工学研究科情報・ネットワーク専攻修了．同年日本電信電話株式会社入社．現在，情報統合管理技術の研究開発に従事．データベース，情報検索等に興味を持つ．日本データベース学会会員．

毛受　　崇（正会員）

2008 年名古屋大学大学院工学系研究科情報システム学専攻修士課程修了．同年日本電信電話株式会社入社．現在，情報統合管理技術の研究開発に従事．日本データベース学会会員．

長谷川知洋

1997 年筑波大学大学院理工学研究科修了．同年日本電信電話株式会社入社．XML 索引技術，高可用化データベース，分散環境におけるデータ処理技術の研究開発に従事．

谷口　展郎（正会員）

1994 年東京大学大学院工学系研究科機械工学専攻修士課程修了．同年日本電信電話株式会社入社．著作権保護，画像検索，プライバシ保護等に関する研究開発に携わる．現在，分散データ処理に関する研究開発に従事．

山室　雅司（正会員）

1987 年早稲田大学大学院理工学研究科数学専攻修士課程修了．同年日本電信電話株式会社入社．1990 年コロンビア大学大学院電気工学研究専攻修士課程修了．ネットワーク設計法，データベース設計・可視化，マルチメディア情報検索，デジタル情報流通基盤の研究開発に従事．博士（工学）．平成 6 年電子情報通信学会学術奨励賞．電子情報通信学会，日本データベース学会，日本ソフトウェア科学会，IEEE-CS 各会員．情報処理学会情報規格調査会規格役員．