ヒューマンインタフェース 101－4
（２００２．１１．２３）

利用者の多様な処理記述理解を目的とした属性に基づく処理のモデル化の試み

古宇田 フミ子　近山 隆†

現在のコンピュータシステムの利用者インタフェースは、システムごとに異なる個々の処理のプリミティブを提供するのみで、目的を果たすためにこれらのプリミティブをどう組合わせて用いるかは利用者に任されており、慣れない利用者にとっては使いにくいものとなっている。この問題の解決には、利用者は処理の目的を記述し、具体的な処理手順は目的記述から自動生成することが考えられる。このために、計算機システムが提供する処理を共通に記述する枠組を提案する。提案する枠組では、処理の対象の諸属性に注目し、処理前後の属性の変化を処理として捉える。これによって、処理のモデルを属性空間上の写像として構築する方向を示した。

Toward a semantic modelling of computer processings for enabling to interpret what user-descriptions of processing mean

Fumiko Kouda, Takashi Chikayama

Current user interface provides users with OS-driven separate primitives with only a single identifier, which are parts of a purpose ofprocessing and are difficult to combine but specified, so that novices are hard to understand. To resolve this problem our approach is to consider attributes of the computer processings, based on the fact that users have their own abstract concept of computer processings using a user interface. Thus, we observe some concrete processings and extract several common features on the attributes of the processings.

東京大学 情報理工学系研究科 電子情報学専攻　新領域創成科学研究科 基盤情報学専攻 †
Graduate School of Information Science and Technology, AND School of Frontier Sciences†, The University of Tokyo

1 はじめに

1.1 位置づけ

人間の特性を踏まえて機械に向かうヒューマンインタフェース (HCI) の研究では、計算機に関しては専門家向けの利用者インタフェース (UI) が誰でも使えるものと見做し、CSCW 方面や E-Learning 等、新アプリケーションが次々と開発される。利用者インタフェースは処理を理解するための簡潔な統合概念が確立されないままであり、利用者は個別に対応する必要がある。そのため、慣れていない利用者には使用のためのハードルが高い。本研究は、この問題解決のために、利用者中心に利用者インタフェースの在り方を考え直した基礎的なものである。

1.2 背景にある問題点

計算機で行なう処理の提示は実際の処理をある程度抽象化し、利用者インタフェース (UI) として利用者に提供する。

この抽象化がシステム構成に基づいて行なわれ、しかも、利用者インタフェースがプリミティブな形 (コマンド等) で提供されるので、慣れていない利用者[1]には簡潔すぎて理解しにくい (too simple to understand)。即ち、利用者インタフェースでの処理の提示法が計算機の OS 毎に異なり、しかも、プリミティヴコマンドの形式や引数の構文に制限がある等、表示の許容範囲内で厳密性を要求している。また、プリミティヴ表示は処理として可能な解釈のうちの一つの解釈の表現であるので、プリミティヴの表示から受ける印象だけからは正しい処理の内容の予測ができないことがある。例えば、*unix* 系の *mv* コマンドはファイルの場所の移動にもファイル名の変更にも使われるが、後者の使用法は気づきにくい。更に、利用者インタフェースで提示されるコマンドは汎用性を持たせるため小さな単位の単純な変換であるので、利用者が望む処理はこのような小さな単位の処理ををどのように組み合せれば実現できるのかが分かりにくいことがある。そのうえ、利用者インタフェースで提示されるプリミティヴは特定の手順を踏むことで意味を持つものがある (多い)。そのため、利用者はコマンドの実行順序にも通じている必要がある。例えば、*unix* 系の *dvi2ps* は *latex* コマンドを実行した後の対象のみに意味がある。

現在の利用者インタフェースの処理の抽象化にはこのような問題がある。換言すると、道具としての利用者インタフェースでは専門家ならば理解できるが計算機についての常識を欠く素人にはどのような条件で使われるのか、何を行なっているのか、不明なことも多い。尚、ここでは UI の操作法が不便かどうかを問題にしているのではなく UI で提示された内容が理解しにくいかどうかの問題を扱う。

利用者にとっては、部分的、計算機構造に依存した処理の抽象化ではなく、包括的に何ができるか、という処理目的が重要であろう。しかし、(文献検索などのアプリケーションを便利にするものはあるが、) この点 (処理では何ができるか) に関するサポートは (少) ない。そのため、余り馴れていない利用者にとっては計算機を利用するための大きなハードルとなる。

1.3 利用者にやさしい UI の必要性

1.3.1 解決法

概念の説明にモデルを立てることで物事の本質を明瞭、簡明、共通に説明できることがある。

処理の本質を適切に記述する記述モデルがあれば、利用者には計算機処理の理解を助ける点で役立ち、計算機社会の発展に寄与すると考えられる。しかし、現実には計算効率を上げるため、個別処理向きに考えられたモデルは存在する (例えば、現存する RAM モデル、HMM モデル、UMH モデル等は sort, FFT, 行列を効率良く実行する [AAC87][CKPS93] ためのアルゴリズムの基となる計算機構造のモデル化である)。利用者のモデル [JK] も存在するが、利用者自体の趣向の分類であり、利用者が処理をどう見るか、の観点ではない。利用者の方向を向いた処理の共通モデルは存在しない。このため、利用者のは処理の本質が見えにくくなっていると言う問題がある。

解決法の一つとして利用者インタフェースを OS を反映した抽象化に基づくものとしてではなく、計算機を使う視点に立ち利用者の知識に基づいた処理記述をできるようにすることが挙げられる。処理手順を細かく提示するのではなく、処理目的が何であるかを記述することで処理の指示を可能としたい。最終目標は利用者が行ないたい処理を通常の言語を用いて記述することで処理を行なうための (処理部品としての) プリミティヴの組み合せ (組み立て) を提示できるようにする (実行させるのではない。その理由は複数通りの可能性があるので提示に留める) ことであるが、中間目標として、利用者が述語 (記述要素間の関係が明確に分かる) の形で記述すると、現実に存在するコマンド (列) を (OS の種類の違いに応じて) 返すような応用プログラムを構成することを目指す。

[1] [Kay1995] によると利用者は beginner, intermediate, guru の三種類に分けられる。また、[Strachan97] では novice, intermediate, expert と分けている。この論文では主に beginner, intermediate について考える。

1.3.2 属性を利用する手法

利用者の抱く処理の概念は計算機の処理を抽象的に捉えた [Ramscar1997] ものであるので、利用者が処理目的として「何をどうする」という処理の基本記述の背景には計算機上の処理が大きく係わる。この点に注目して、利用者を考慮したモデルの基礎をなす部分として、計算機が利用者インタフェースに提示する処理の特徴を調べ、計算機上で処理の本質を抽出することで処理を共通化する。その手法として、処理では処理の対象の属性を変えることに着目する。

異なる OS で実行される各々の処理は処理のやり方やそのプリミティヴ表示が異なっていても同じ結果をもたらせば、処理は共通なものと見做してよい。即ち、計算機の処理では個別システム毎に処理のやり方が異なることはあっても、処理を行ない、ある結果を得る、という点では共通性がある。ここで「同じ結果」という意味は結果として得られた対象が「同じ属性を持つ」と言い換えられる。このことから、共通化を図るために処理の対象の属性に注目する。更に、処理動作は対象そのものではなく、対象の属性に働きかける、と見る。即ち、処理は属性間の対応と捉える。このことで、処理動作は、特定の対象に限定されることなく広く適用可能となる。

1.2節で述べたように、現在の計算機の利用者インタフェースの問題点の一つは、処理の解釈がシステム (= 計算機設計者) が示した一通りのみであるので、それ以外の解釈に従って処理に向かおうとすると破綻する点にある。処理の解釈を従来の一種類の解釈だけでなく、可能な解釈も加えて、このような解釈に基づいた複数の属性を持つ、と見ることで一つの処理を多面的に捉え得る。このことを利用すると、同じ処理を異なる属性から記述した場合に処理の異同を対応付ける方法があるので利用者の多様な視点も対処可能となる。

処理に関係する属性をすべて考慮することで、属性間の関係を基に処理合成の記述ができる。合成を繰り返すことで、処理の粒度を利用者が目指している程度と対応させることができる。合成された処理の記述で処理の最初の入り口と最後の結果に注目すれば、処理のやり方に依存しない結果となる。利用者の記述を、処理の入り口の対象と所望の結果を表す形に解釈し、入り口や結果における処理の対象の属性間の関係を利用することでプリミティヴ処理を合成したものと対応させることができる。逆に、利用者記述と対応する処理の入り口と結果の属性の組を属性の持つ制約を満たしながら分解することでプリミティヴな処理の列に分けられる。このとき、分解法は一通りとは限らない。このような形で処理目的との対応付けが可能となる。

1.3.3 新しいインタフェースの要件

大局的な処理目的や視点を変えた処理の解釈 (単に微視的なものだけではなく統合的なものも) 等の記述を理解する機構を加えて計算機の利用者インタフェースを構成すれば、利用者の常識の一部を計算機側で直接理解した、と見做せるであろう。

このような構成ができたとして、各種の解釈を計算機側で提示して利用者が選択するのは煩雑すぎて現実的ではない。そこで、このような (利用者が処理を理解しにくい) 問題の解決法の一つとして、利用者が自らの知識に基づき処理目的の記述をすることで処理機能の問い合わせが可能となるようにし、計算機側で利用者の記述を理解する、換言すると、計算機側に処理記述を属性間の対応関係として解釈し、多様な記述から処理の異同の判定法を追加することで、利用者が希望する処理機能と実在する処理との対応関係の乖離 (leap) を計算機側で埋める方法を採った。

現実問題として、利用者の知識を厳密に定義することは難しい。そこで、利用者の意図は「計算機を用いてある結果を得る」ことにある、と限定する。その表明として処理の記述を行なうものとする。

1.4 前提

以下の議論では計算機環境は単一の場合も複数台接続された場合も同様に扱う。これが可能な理由は、複数接続された場合は計算機の違いを隠蔽し得るミドルウェア以上のレベルで考えると単一の場合と同様に見做せるため [Ta2002] である。この他に、バッチ処理を想定する。対話型処理は今後の課題とする。処理の対象はファイル媒体上にあり、利用者は処理の対象を概念として捉え、概念は言語として表現される、と仮定する。

2 実在の処理の観察

2.1 利用者インタフェースの特徴

UI で提示している処理の抽象化の特徴は、
(SS-i : 入出力対応を持つ処理部品の提供) 単一処理は一組の入力に対して一組の結果を返す。処理そのものが何を意味しているかの解釈は提供しない。これらを組み合せると、より粒度の大きな処理が実現できる。組み合せに自由度が大きいので異なる方法が可能なものもある。
(SS-ii : 構造の抽象化) 計算機に置かれる資源の物理構造の違いを隠蔽して共通に扱えるように抽象化し、その表現の単位として 1 と 0 の単位ビット (bit) を用いる。この単位を基にして組み合せることで、計算機資源はすべて論理構造として把握できる。例えば、アスキーコードは英文字を 8 ビットからなるバイト (Byte) 単位のコードと対応させて表現したものである。

(SS-iii : 一次元化) (SS-ii) の構造は 0, 1 の並びを基にして一次元に展開し得る。実際には、飛び飛びの配置もあるが、資源の配置には連番のアドレスを振ることができる。アドレスと一対一に対応させることで一次元化し得る。
(SS-iv : 処理と属性) 実際の処理では、処理対象の属性を計算機上のある構造に対応させ、この解釈に従って、構造を変えることで属性間の変換をする。構造変換により属性が変化した、と見做す。
(SS-v : 特定の構文を利用) 個別処理は実行のために入力する対象は特定の構文をとる。結果として生じた対象も特定の構文を持つ。利用者にとってはこの事実を無視すると希望の処理が得られない。
(SS-vi : 基本表現) 計算機上の論理構造を持つ対象表現の基本単位は (SS-ii) より {1, 0} で表される。これより細かくは見ない。
(SS-vii : プリミティヴの特徴) 利用者インタフェースで提示されるプリミティヴは英語の動詞が使われ短い表現である、処理の属性の一つを代表して表す、 OS により提供される表現が異なり得る、引数があり対象の構造に依存性がある、
等が挙げられる。

処理の抽象化は処理指示に関するものと資源の表現に関するものに分けられる。上記のうち、処理指示に関するものは (SS-i)(SS-ii)(SS-iv)(SS-vii) で、資源の表現は (SS-ii)(SS-iii)(SS-iv)(SS-v)(SS-vi) となる。 (SS-vi) に関しては計算機資源を単純化してすべてをファイルとして扱う見方に従い、以下では、 {1, 0} で表される対象をバイナリファイル、と呼ぶ。

2.2 例題

前節の特徴に従って、処理の例を挙げる。

(1) *tar cvf new.tar existing_files ..* の場合、 (SS-i) の操作はファイルを一つのものと見て、特定の文字列 (元のファイル名や権限) を「糊」として複数のファイルを繋ぎ、結果として一つのファイルとする操作である。
(SS-ii) の構造抽象化はファイルを基本表現であるバイナリファイルとして捉える。ディレクトリの構造も考えるが、それ以外の構造は存在していても考慮しない。処理で注目する属性はファイル (又はディレクトリ) であること、処理の結果も一つのファイルであること、となる。一次元化可能であり、 (SS-v) の構文は処理の対象がバイナリファイル (又はディレクトリ) であることにある。

(2) *latex* の場合、 (SS-ii) の構造の抽象化は latex 構文と見る、であり、 latex の操作指示を示す表現の部分と本来の文書の部分から成る。文書の部分は文字として区別する。 (SS-i) の操作は latex 構文の指示に沿って、その他の部分の文字コードを座標と文字フォントを表す dvi 構文に変換する。処理に関係する属性は、
定義域では、 A_{lat} = { 文書である、 latex 構文の操作指示である、ファイル媒体で実現された、バイナリファイル、ファイル上では文字コードを単位として解釈 }
値域では A_{dvi} = { 文書である、 {dvi 構文、または、一部が dvi 構文で一部がエラー表示、または、エラー表示のみである }, dvi 構文は座標と文字の形への対応付けとして解釈、ファイルで媒体で実現された、バイナリファイル }
となる。

ファイル媒体で実現されているので、一次元化可能、また、 (SS-v) の構文は入力が、 latex 構文として正しいこと、がある。正しくなければ結果は dvi 構文に変換できない部分を無視してエラーメッセージを出す。

属性間の対応関係は
{ 文書である }↦{ 文書である }
{ ファイル媒体 }↦{ ファイル媒体 }
であり、更に、正しい場合は、
{latex 構文である }↦{dvi 構文である }
{ ファイル上では文字コードを単位として解釈 }↦ { ファイル上では座標と文字の形への対応付けとして解釈 }
となる。

属性間の対応関係はエラーを含む場合は、
{latex 構文である }↦ { 変換可能な部分を dvi 構文に変換 かつ 変換不可能部分のエラー表示 }
となる。

(3) *lpr* の場合、 (SS-i) の操作は表現変換と媒体変換である。 (SS-ii) 構造の抽象化は処理の単位を {1, 0} として、バイナリファイルと見る。結果の単位は、処理系毎に定義される出力のフィルタに依存して異なる。代表的なものは、出力をバイナリファイルとする、又は、定義域でコード化した単位を基にする、又は、定義域の解釈の単位に基づく記述に従って再構成する (例えば、 {1, 0} の並びを文字としてコード化して解釈した場合、文字による記述で図形を表す場合)、等がある。

処理を属性間の変換と見ると、
定義域では A_{dic} = {{1, 0} を変換の単位とする、表現は一次元有限列、ファイル媒体、構文の解釈が特定の構文 (例、 ps 形式、 pdf 形式) を持つ }
値域では A_{pap} = { ビットマップ表示、表現は二次元、値域の解釈の単位は定義域の解釈の単位に基づく記述に従って再構成する、紙媒体、定義域の解釈と同じ解釈に立てば構文の要求を実現した構造 (例、 ps 形式、 pdf 形式の具現化) を持つ }
の属性が用いられる。

これらの間の対応関係は、
{ 変換の単位は {1, 0}}↦{ 解釈の単位は定義域の解釈の単位に基づく記述に従って再構成 }
{ 一次元有限列表現 }↦{ 二次元表現 }

{ ファイル媒体 }↦{ 紙媒体 }
となる。

計算機上の媒体では一次元化可能である。また、入力対象には構文依存性 (txt, ps, dvi, pdf, doc 等、入力形式が特定の構文を持つ) がある。期待されていない 構造の入力の場合は正しい結果が出ない。

出力のためのフィルタの操作を中間段階と見るとフィルタの種類毎に入力属性の構文が定められる。中間出力はプリンタデバイスに従った属性となる。

(4) *latex* +*lpr* の場合、(SS-i) の操作は latex 構文をその指示に従った形で二次元に出力し、内容を固定する動作である。(SS-ii) の構造の抽象化では処理の入り口の対象は latex 構文であると捉える。処理の結果の対象は紙上でビットマップの形状となる。

属性の変化に関しては定義域は A_{lat} = { 文書である、latex 構文として解釈される、ファイル媒体、基本表現はバイナリファイル }
値域では A_{pap} = { 文書である、表現は自然言語の文字や記号、保存可能、内容固定、可視表示、紙媒体、ビットマップ表現 }
という属性間の変換となる。

プリンタが dvi 形式を入力として受け付ける場合は、*latex* の結果の属性は A_{dvi} = { 文書である、dvi 構文である、ファイル媒体、基本表現は $\{1, 0\}$ である、ファイル上では座標と文字の形への対応付けと解釈 }
であり、これを定義域として *lpr* の処理が行なわれる。

プリンタが *ps* 形式だけを入力として受け付ける場合は、中間の処理として *dvi* 形式を *ps* 形式に変換する必要がある。この場合は *dvi2ps* により A_{dvi} を入力として *ps* 形式に変換する。結果の属性として
A_{ps} = { 文書である、*ps* 形式である、ファイル媒体、基本表現はバイナリファイル、 }
であると、これが *lpr* の入力属性となる。

(5) *tar xvf* の場合、(SS-i) は引数に指示されたファイル名を同じく引数で指示された tar 形式のファイル中で「糊」の部分から同じものを探し出しその部分だけを一つのファイルとして取り出す。ファイル名が無い時はすべての糊を剥して複数のファイルを復元する。(SS-ii) の構造抽象化は tar ファイルをファイル一つと「糊」に分けて見る。処理結果のファイルを並べれば、一次元化可能である。(SS-v) の構文は tar 形式であることである。基本表現の他に tar 形式として解釈する構文がある。

(6) *uniq* の場合、(SS-i) の操作は、文書ファイルから同じ形の行が続いて現れる場合は一つに纏める。文書ファイルを行という属性を単位として区切り、各々の部分について、一つ前の部分と同じかどうかを繰り返し調べる。同じでなければ、結果として出力するが、同じならば何もしない。(SS-ii) の構造抽象化は、行であることに注目する。

3 観察に基づく処理の解釈

3.1 処理の対象の属性

(OB-i : 構文と属性の対応) 利用者の抱く概念が計算機に乗るように見えるのはなぜかを調べる。利用者は計算機の具体的な処理を抽象化して理解 [Ramscar1997] し、概念として捉えて言語で表現する。言語表現により概念の属性が表される。言語は文字が基本単位であり、文字は計算機上の構文の単位である {1, 0} のコード化と対応する。文字コード、文字、言語の対応付けがあるので、計算機上の {1, 0} の基本表現の有限列の構文と概念の属性が対応する。このことを「概念が計算機に乗る」と見ている。この対応関係があるので、処理を計算機上の形式変換としてではなく属性の変化として捉え得る。計算機上の単位と思考上の言語を繋ぐのは思考上の表現の単位である文字と計算機上の単位 0, 1 をコード化した対応関係である。このことから処理の対象の表現とも共通に扱える。
(OB-ii : 媒体自体の属性) 処理の対象は仮定からファイル媒体または紙、ディスプレイ上にある。これらの媒体は対象から見ると器 (receptacle) の機能を持つ。対象を表す器の属性は、(SS-iii)(SS-vi) から、

object_file = < {1, 0} が基本単位である、コード化表現可能、個々は一次元有限列、複数並び得る、{1, 0} は変わり得る >

object_paper = < 二次元有限列、可視表示、値は固定 >

object_display = < 二次元有限列、可視表示、値は見かけ上変わり得る >

これらの共通属性には、(OB-i) より < 概念を言語表現の形で埋め込むことができる >、がある。
(OB-iii : 媒体と概念属性) (OB-ii) を別の面から見ると、対象を表す媒体上の構文は概念の属性に対応した構文の他に器の属性も持つ。そのため、*lpr* のように媒体を変える処理は (ファイル媒体 ↦ 紙媒体) のように器の属性が重要になるが、*latex* ではファイル媒体が同一なので概念由来の属性だけを考えればよい。

3.2 処理動作と処理の対象の属性

(OB-iv : 処理単位) (SS-iii) より計算機上の対象は一次元の構文として表される。この構文には利用者が概念として抱く属性の構文や、処理でバイナリファイルの基本表現を解釈して構成した構文等、処理で単位と見る構文がある。このような単位を処理単位と呼ぶことにする。処理単位はある属性として識別される。*tar* ではファイル全体を一つと見る、また、*uniq* では一行を一つとして見る、等、処理動作はバイナリファイル上でその処理にとって都合のよいように構文を分ける。同一のファイ

表 1: Pivot Attribute and its Adjacent Attributes

軸属性	隣接属性	
文字、数字	JIS コード	EUC コード
	座標表現	ビットマップ表現
図形	文字による説明	座標表現
	ビットマップ表現	
EUC コード	かな漢字表現	ローマ字表現
	ギリシャ文字表現	キリール文字表現
文書の内容	文字コード表現	ビットマップ表現
(C 言語の) 式	文字、記号表現	ニモニック表現

ル媒体上の処理の対象は処理に応じて構文が異なり得るので、同じ対象に処理単位となる異なる構文が存在する。処理単位となる構文同士の包含関係に注目すると半順序関係がある。そこで、包含関係についてはレベルの概念を導入して区別する。

(OB-v : 注目属性) (OB-iv) から、処理では処理に都合よく単位に分けて属性と見るので、利用者が考えている属性とは異なる属性として捉え得る。例えば、*lpr* で利用者は文字として出力して欲しいと思ったが、結果は 1 や 0 の羅列であることも起こる。

(OB-vi : 軸属性と隣接属性) *latex* では、文書である、ファイル媒体である、という属性は不変化である。また、*lpr* では媒体が紙に変わるが、文書である、という属性は不変化である。(OB-i) の性質から、文書が不変化ならば、その基本単位である「文字である」ことが不変化となる。「文字」属性に注目すると *latex* 処理前は文字と文字コードとが対応し、処理後は文字と文字の座標表示に対応する。*lpr* の結果は文字と二次元ビットマップ表示が対応する。不変化である文字属性を中心に見ると、「文字コード」 → 「文字の座標表示」 → 「二次元ビットマップ表示」、のように文字属性を軸として回転しているものとして捉えられる。この見方に立つと、表 1に示すように不変化属性とその周りを相互変換 (回転する) 属性があることが分かる。そこで、不変化の属性の各々を軸属性 (pivot attribute)、相互変換する属性を隣接属性 (adjacent attribute) と呼ぶことにする。

(OB-vii : カプセル属性と枠属性) ある処理単位の属性が外枠となり、この属性を区切りとして考える場合がある。例えば、「ファイルであること」、という属性は中に含まれる属性は問わない。主に外枠の属性に注目するか、中の入れ子の属性に注目するかで場合分けできる。*tar* ではファイルであること、という属性を満たせば、中に含まれる属性は問わないので、外枠に注目している。これに対して、*uniq* ではファイルであることの下で中に含まれる行属性に注目する。そこで、外枠のみに注目する場合をカプセル属性 (capsule attribute)、枠の中に注目する場合の外枠の属性を枠属性 (frame attribute) として区別する。何れの場合も、より外の属性は何であってもよい。これを「知らない属性 (don't-know attribute)」また、カプセル属性の場合は中に含まれる属性は何であってもよいので、これを「気にしない属性 (don't-care attribute)」と呼ぶことにする。

(OB-viii : カプセル属性と処理) カプセル属性が処理で不変化であれば、中の属性は変わらない。カプセル属性が変わる時は、その中に不変化である軸属性があれば軸属性を中心としてカプセル属性と連動して隣接属性が変わる。変化の行き先はカプセル属性に依存する。例えば、文書の *latex* 処理ではカプセル属性としての latex 構文が dvi 構文に変わる。 latex 構文の中に軸属性として文字属性があるので、結果である dvi 構文には文字属性の隣接属性である文字の座標表示を示す属性が含まれる。

(OB-ix : 枠属性と処理) *uniq* は、ファイルである、という属性を枠として、ファイル中の行である、という属性は残して相対関係を変える。

uuencode 等の暗号化ではファイルであるという属性は変えずに、その中の属性全体の形態を変える。枠属性では要素の出入りが無い時を不変化、要素の出入りがある場合を属性が変化した、と見ることができる。

文書をファイル媒体から紙に印刷する場合では、枠属性は中に含まれる属性を軸として異なる枠属性と対応がつく。

3.3 対象と属性

(OB-x: 対象の属性) 一つの処理の対象は (OB-iv) の形で異なる処理単位に分けられ、対応する属性で表現される。このような属性は同一対象上の属性なので、関係がある。媒体に関係する属性は、構文間の包含関係を属性の順序関係として、概念に基づく属性は対応する構文がある (OB-i) ので、この構文の上での包含関係を順序関係とすると、半順序 (semi-order) がある。

他方、(OB-ii) の属性は、文字と文字コードの関係で対応するが、対象の属性とは別に存在する。異なる半順序の系列は別の種類と見ると、一つの対象に異なる複数の属性のグループが作られる。

3.4 属性と写像

(OB-xi : 処理と属性) 仮に、処理の対象に関わるすべての属性を見渡すことができたとしたら、処理が行なわれることで、複数の属性が関係し、すべての属性について、変化する属性と変化しない属性の区別がつく。この中には処理に関係するものも関係しないものも含まれる。

処理に係わる属性が提示できたとすると、どの処理であっても共通に、これらの中では、変化する属性と同時に明示的に変わってはいけない属性があることが観察さ

れる。例えば、ファイルを *tar* で繋ぐ場合は、媒体の場所は変わってもよいが、内容そのものは変えてはいけない。変化前の属性の組と変化後の属性の組の対応と見ると、この関係が処理動作に当る。

(OB-xii : 属性対応のタイプ)　処理に関係する属性に注目する。カプセル属性または枠属性が (複数個) 観察される。例えば、*tar* では「ファイルである」と「バイナリファイル」がカプセル属性となり、*latex* では「latex 構文である」と「バイナリファイル」がカプセル属性となり、*uniq* では「文書ファイル」が枠属性で「バイナリファイル」がカプセル属性となる。

明示的に不変化な属性がある場合、それはカプセル属性や枠属性の変化時に軸属性となることがある。例えば、*latex* では「文書である」と「バイナリファイル」が不変化である。「文書」は「latex 構文」に含まれる属性なので、カプセル属性に含まれる軸属性となり、写像によりカプセル属性の「latex 構文」が「dvi 構文」に変わる。文書を印刷する *lpr* では「文書である」が軸属性となり枠属性である「ファイル媒体」が「紙媒体」に変わる。

このように処理の属性間の対応には、幾つかのカプセル属性又は枠属性があり、不変化属性の有無により、以下のタイプが存在することが分かる。

(I-a) 処理の写像でカプセル属性が変化しない。
(I-b) 処理の写像でカプセル属性が変化する。
　(I-b-i) カプセル属性に軸属性が含まれない。
　(I-b-ii) カプセル属性に軸属性が含まれる。
(II-a) 処理の写像で枠属性に含まれる属性が変化しない。
(II-b) 処理の写像で枠属性に含まれる属性が変化する。
(II-c) 処理の写像で枠属性に対応する軸属性が存在して、別の枠属性になる。

処理の写像では変域と値域は複数の属性が関係し、それぞれ上記のタイプの組み合せの写像となる。

(OB-xiii: 正則な写像)　属性の対応で変化した結果、新しい属性が付け加わる場合と一部が無くなる場合、1 対 1 に対応する場合がある。*lpr* ではフィルタを通ることでファイル媒体の属性が消える。印刷結果は新たに、「値固定」、「可視表示」の属性が加わる。*latex* では「latex 構文」と「dvi 構文」は 1 対 1 に対応する。

属性の種類が 1 対 1 に対応する場合を正則と呼ぶ。

(OB-xiv: 処理の合成)　*latex* + *lpr* では *latex* の結果の属性が *lpr* の入り口の属性となる。合成処理全体の入り口と結果の属性を見ると、「文書である」は合成処理として軸属性であるが、「バイナリファイル」は前半のみの軸属性であり、後半では隣接属性に変わる。

(OB-xv: 処理の逆写像)　*tar cvf* を行なった後で *tar xvf* を実行すると、「ファイルである」属性は元に戻る。この場合、対応関係の写像 f には逆写像 f^{-1} が存在する。

(OB-xvi: 共役写像と独立な属性)　JIS コードの文書を EUC コードに変え、連続する同じ二行を一行に減らして JIS コードに戻す処理を考えると、例えば、*nkf -e→uniq→nkf -j* となる。この場合、*uniq* で行を減らす処理は、*nkf -j* には影響を与えない。*nkf -e* を f, *uniq* を g, で表すと、$f^{-1} \circ g \circ f = g$, 又は、$g \circ f = f \circ g$ となる[2]。このような関係にある時、処理が独立である、と呼ぶことにすると、関係する属性も、g の変化に係わる属性は f の変化に係わる属性の観点からは知らない属性に含まれるので、独立となる。そこで、(OB-x) の対象の属性を処理で独立になるグループに分ける。

4　考察

処理のモデル化の要素について考察する。

4.1　処理のモデル化と属性空間

(OB-i) の観察より、処理の概念と計算機上の構文が対応し、また、(OB-ii) より、計算機媒体も属性で表せるので、計算機上の処理は属性空間の上で考えることができる。このことで、利用者の処理の概念属性と実際の処理属性の対応が図れる。

処理は対象の属性を変えるものであるから、属性空間は対象に関係する属性の包含関係や独立な関係を基に構成する。

4.2　処理のタイプ

(OB-xii) の観察から、処理と処理の対象の属性の間には、処理が提示するカプセル属性又は枠属性と (もしあれば) 軸属性が対象の持つ属性と一致すればよく、他の属性は任意でよい、という関係がある。このことから、処理の合成には対象の持つ複数属性の一部分が処理のカプセル属性、枠属性、軸属性の組と一致すればよいことになる。

4.3　多様性に対処し得るか

処理の入り口と結果で示される属性を明示することで、例えば、「latex 形式の文書を可視表示したい」と記述した場合、これまでの UI では対処しようがなかったが、関係する属性をすべて提示する方法ならば解決の可能性がある。

5　おわりに

利用者の知識に基づいた処理記述の理解を可能とするために、UI で提示される処理の特徴を処理の対象の属性から捉えることで、処理のモデル化の要件を考察した。

[2] 厳密には特定のコードで正しくない。

実際の処理を観察し、処理で変化する属性の特徴を抽出した。処理では属性が変化するものと不変化なもの、見ないもの、があることが観察され、これらを軸属性、隣接属性、カプセル属性、枠属性、知らない属性、気にしない属性、に分類した。これに基づき、処理のタイプの枠組を示した。

利用者の処理概念は属性として捉えると計算機に乗る、と見てよいので、処理のモデルは属性空間で考察できる。

ここでは直接処理対象と対応がつく属性のみについて考察した。しかし、利用者提示の概念にはより抽象的な属性もあり得るので、この点の考察も必要になる。

今後の課題は精密化を行ないプロトタイプの構築、等である。

参考文献

[Ta2002]Tanenbaum, "Distributed Systems principle and paradigms" 2002.

[Kay1995] Judy Kay, "Vive la difference! Individualised interaction with users" IJCAI95, pp.978 – 984, 1995.

[Strachan97] Linda Strachan et al. "Pragmatic User Modelling in a Commercial Software System", UM97, pp. 189 – 200, 1997.

[JK] Judy key "The um toolkit for cooperative user modelling."

[Ramscar97] Michael Ramscar, Helen Pain, and John Lee, "Do We Know What the User Knows, and Does It Matter ? The Epistwmics of User Modelling" CISM1997, pp.429–431, 1997.

[AAC87]Alok Aggarwal et al. "A model for hierchical memory",ACM STOC,pp.305–314, 1987.

[CKPS93]David Culler et al. "LogP:Towards a Realistic Model of Parallel Computation", Proceedings of the fourth ACM SIGPLAN symposium on Principles and Practics of Parallel Programming, 1993.