依存関係の一般化による作業の再利用支援インターフェース

西田 衣織[1] 守田 了[2] 田中 稔[2]

[1] 大学院理工学研究科知能情報システム工学専攻
[2] 知能情報システム工学科

作業の再利用性を高めるために過去に行なわれた作業を一般化し、対象となるファイルを置き換えることによって、過去の作業に含まれるファイルの数に依存しない新しい作業を作成するシステムを提案する。新しい対象の入力の負担を軽減するためにシステムが動的に新しい対象を取得し新しい作業を作成する。既存の C シェルなどのヒストリ機能は、ユーザが過去の入力を再利用する場合、履歴の中から探す必要がある。ユーザが使用頻度の高い作業に対して、明示的に保持できるインタフェースを実装する。マウスを用いた操作を可能とし、ユーザの操作の簡便性を高める。本インターフェースにより、ユーザの作業の繰返し時間の短縮、コマンド再入力の負担の軽減、ユーザのストレスの軽減における有効性を評価する。

Key Words : *command user interface, task reuse, history, generalized dependency*

1 はじめに

一般にコマンドをキーボードから入力するインタフェースシステムは、コマンドライン単位で入力し実行するため、ユーザが複数のコマンドで構成されたコマンドラインの再入力を行なう負担は大きい。本研究ではこの問題点に着目し、効率良く作業の再利用を行なうためのインターフェースを設計する。

従来の作業の繰返し支援機能である make や シェルスクリプトは、Makefile やシェルスクリプトファイルの記述と知識が必要でありビギナーにとって使いにくいものであった。Dmake[1] は簡単な操作で入出力ファイル間の依存関係からなる作業を取得し、ユーザの作業の再実行を行うシステムである。しかしファイル名の異なる作業に対して支援されていなく、作業自体を入力し直す必要があった。GenHist[2] はコマンドラインを一般化した記述 (汎化履歴) を用いて作業を一般化し、異なるファイル名を適用することで新しい作業を作成し、実行するシステムである。しかし GenHist は一つのファイル名に対し、1 つの新しいファイル名を置き換えるシステムのため、新しい作業に適用範囲が限定されていた。また置き換える際にファイル名の入力をユーザに要求しているのでファイル名を入力する手間があった。本論文では, 一般化した作業に対し、新しい作業の生成規則を適用し過去の作業に依存しない新しい作業の作成する機能を導入する。またユーザのファイル名の入力の負担を軽減するために、対象となるファイル名を本システムが自動的に取得し新しい作業を作成する機能を実現する。

GenHist はユーザのコマンドラインの履歴しか用いないので、複数の作業を再利用する場合、その度に履歴の中から必要なコマンドラインを探す必要がある。過去の作業を効率良く再利用するためには, 簡単に作業を保持することができ過去の作業を容易に確認できることが必要になる. 本論文では、Greenberg らの提案したコマンド再利用システム Workbench[3] の機能の 1 つである toolcache を用いる。またコマンド入力の手間を軽減するために、各機能をマウス操作で実現し、操作の簡便性を高める。表 1 は本研究と他の関連研究の支援機能内容を比較したものである。

2. では作業の再利用の定義について説明し、3. では作業の再利用支援インターフェースについて述べ、4. では既存の CUI 支援機能との比較を行なうことにより有効性を評価する。

表 1 関連研究との比較

	本研究	Dmake	GenHist	Toolcache
作業の繰返し実行	○	○	○	×
新しい作業の実行	○	×	○	×
作業の保存	○	×	×	○
マウスによる操作	○	×	×	○
複数の対象の適用	○	×	×	×
動的な作業の作成	○	×	×	×

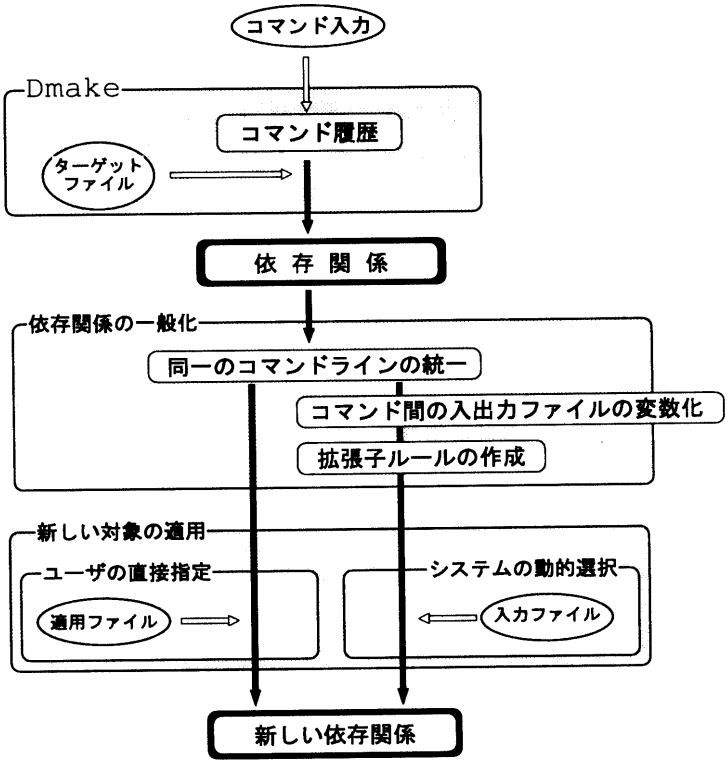


図 1: システムの構成

2 作業の再利用

本インターフェースにおける作業の再利用とは、過去に実行した作業を使ってファイル名が異なる新しい作業を作成し、実行することである。ここでは本システムの概略を述べる。

本システムの構成を図 1に示す。システムは、まずユーザの過去の作業の取得を行なう。これは Dmake の手法を用いた。この手法は最後に生成するファイルであるターゲットファイルを用いて入出力ファイルやコマンドラインの情報を記述したコマンド履歴から入出力ファイルの依存関係を辿って Makefile に類するものを作成する。この手法で取得した作業を依存関係と呼ぶ。

つぎにシステムは、取得した依存関係をファイル名が異なる作業である新しい依存関係に適用するため、依存関係の一般化を行なう。

最後に一般化した依存関係に異なるファイル名である新しい対象を 2 通りの手法で適用する。ここで新しい依存関係の生成規則を用いることで、過去の依存関係に依存しない新しい依存関係が作成される。次に実際の詳しい手順を示す。

2.1 依存関係の一般化

図 2 は Dmake 手法によって取得した作業のツリー構造の例である。ルートはターゲットファイルを生成

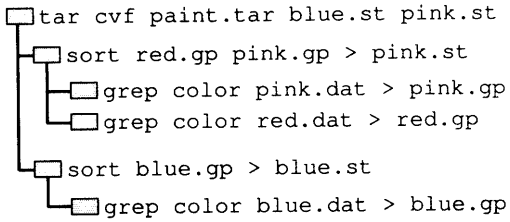


図 2: 取得した依存関係のツリー構造

したコマンドラインであり、他のノードとは入出力ファイルの依存関係によって結ばれている。依存関係の一般化は 3 つの操作で構成される。これらの操作を図 2の例を用いて説明を行なう。

2.1.1 コマンド間の入出力ファイルの変数化

コマンド間の入出力ファイルの変数化は、図 3のようにツリー構造内のコマンドラインに含まれている各ファイルの拡張子を除いた同じ文字列に同じ変数名 ($1 ～ $4) を与える操作を行なう。この操作により 1 つのコマンドライン内のファイル名の変更を行なうと、他のコマンドライン内の同じファイル名が変更される仕組みとなり、ユーザがファイル名を変更する箇所を減少することができる。

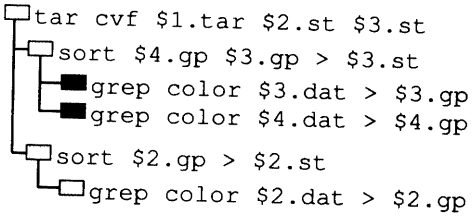
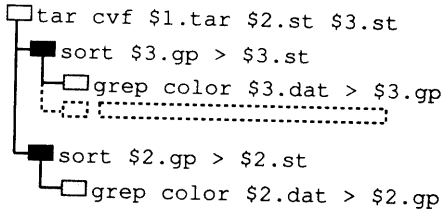


図 3: コマンド間の入出力ファイルの変数化

(a) **同一なコマンドラインの統一**



(b) **同一なノードの統一**

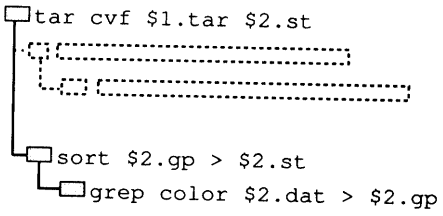


図 4: 同一のコマンドラインの統一

2.1.2 同一のコマンドラインの統一

変数化した依存関係に対して各ノードのマッチングを行う。このノードの統一によって得られた依存関係の構造をここでは make rule と定義する。この make rule を作成することで、依存関係内のコマンドライン及び変数を更に減少することができる。同一のコマンドラインの統一は以下の手順で行なう。

1. 同じ親を持つノードで子のノードを持たない複数のノードに対して、コマンドライン内の各文字列の比較を行ない、同じ並びであれば1のノードとしてまとめる。ここでは grep の2つがその対象となる (図 4-a)。

2. 子のノードを持つ複数のノードに対し、そのノードと子のノードをそれぞれ比較し、同じ並びであれば1のノードとしてまとめる。ここでは sort の2つがその対象になる (図 4-b)。

この操作によりコマンドラインの数は6つから3つ、変数の数は4つから2つに減少することができた。

2.1.3 拡張子ルールの作成

拡張子ルールとは、入出力ファイルの拡張子に依存したコマンドラインの依存関係と定義する。make rule

```
$1.tar : $2.st
  tar cvf $1.tar $2.st
$2.st : $2.gp
  sort $2.gp > $2.st
$2.gp : $2.dat
  grep color $2.dat > $2.gp
```

図 5: 拡張子ルールの作成

である図 4-b から、各変数の拡張子に着目し変数の入出力関係に基づいて図5のような拡張子ルールを作成する。この記述は make の Makefile と似た記述であり、新しい作業を作成する際はこの記述に基づいて新しい対象を割り当てる。

2.2 新しい対象の適用

新しい対象の適用は、一般化した依存関係から新しい依存関係を作成する規則を用いて新しい対象を適用し新しい依存関係を作成することである。最初に新しい依存関係の生成規則を示し、次に2種類の対象ファイルの適用手順を説明する。

2.2.1 新しい依存関係の生成規則

ファイルを変数化した依存関係に、新しい対象を適用する時、各変数と新しい対象が1対1の関係で適用されるのであれば問題はないが、1つの変数に対して0もしくは2つ以上の新しい対象が適用された場合は、コマンド間の依存関係を考慮し依存関係を再構成する必要がある。新しい依存関係の生成規則はこれを定めた規則である。

新しい対象を適用する変数の順番は、ターゲットファイルを持つコマンドラインの出力ファイルから始まり、その入力ファイル、次にそのコマンドラインに依存したコマンドラインの入力ファイル、という順番で適用を行なう。ルートから適用する理由は、各ノードの持つエッジの増減はコマンドラインで使用する入力ファイルによって決定されるからである。

次に新しい対象の数の増減による依存関係の再構成の規則を以下に示す。図 6の $1, $1', $2, $2' は変数名、C1, C2, C3 はそれぞれコマンドを表している。

1. 適用する対象がない場合は、そのノードの対象ファイルと依存関係を持つ子のノードを削除する。

2. 適用する対象の数が多い場合は、図 6-a のように子のノードを複製し、依存関係の再構成を行なう。

3. ルートに適用する対象の数が多い時は、その増加した各対象をルートとして、複数のツリーを生成する (図 6-b)。この操作により1つの依存関係か

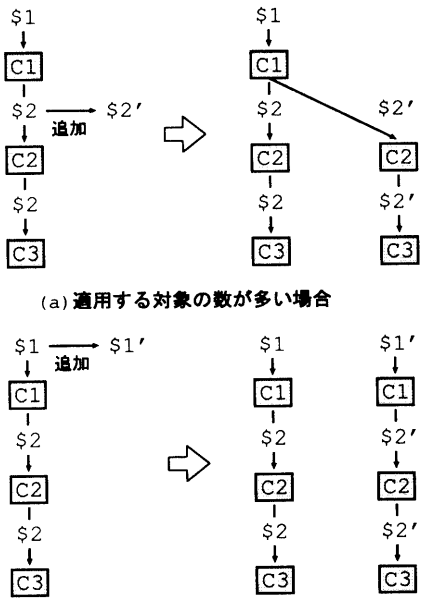


図 6: 新しい依存関係の生成規則

ら、複数の依存関係を同時に生成することが可能となる。

2.2.2 対象ファイルの適用手順

ユーザの直接指定

ユーザの直接指定は、ファイルを変数化した依存関係を用いて、ユーザが各変数に直接新しい対象を指定し、新しい依存関係を作成する手法である。この手法は直接各変数に新しい対象を割り当てるので、新しい依存関係はユーザの意図するものが作成でき、かつ複雑な構造を作成できるという利点がある。しかしユーザが各変数に新しい対象を入力する手間がある。これは次で示すシステムの動的選択を用いることで解消できる。以下にユーザの直接指定の手法の手順を示す。

1. システムは、ルートから下のノードへ順番に変数が全て満たされるまでのコマンドラインの位置を取得する。図 7-a だと 1 , 2 までの位置で、$1 ～ $4 が満たされる。

2. 次にユーザに図 7-b のように変数ではなく、元のファイル名をあてはめたコマンドラインを用いて変更箇所の提示を行ない、ユーザに文字列の変更を促す。

3. 図 7-c は paint はそのままで、blue は gray、pink は green、red は black, white に変更し、適用した構造を示している。

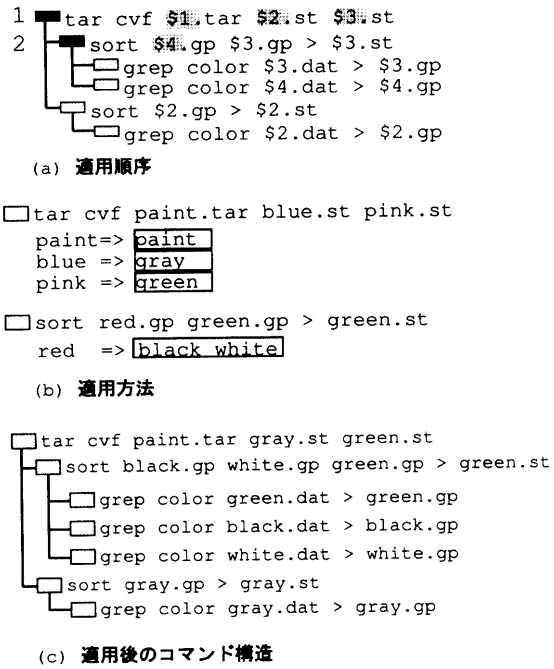


図 7: 対象ファイルの適用手順 (ユーザの直接指定)

システムの動的選択

システムの動的選択の手法は、ユーザの入力の負担の軽減のためにシステムが動的にメニューを選択する手法である. 入力ファイルとなるファイルの拡張子をキーとし、ユーザのカレントディレクトリに存在するファイル群の中から同じ拡張子を持つファイルを入力ファイルとして取得し、そのファイル名を使用する作業を作成する.

この手法は、入力ファイルのみ自動的に選択を行なう手法で、入力ファイル以外の中間ファイル、出力ファイルは元の依存関係に記述されているファイル名をそのまま使用する。もしシステムがユーザの意図にそぐわない結果を提示したときは、編集することができる。これによりユーザは、入力ファイルを指定することなく過去の作業から新しい作業を作成し実行することができる。以下にシステムの動的選択の手法の手順を示す。

1. まず作成した拡張子ルールから、入力ファイルの拡張子の種類の取得を行なう。例では $ 2 .dat が入力ファイルに値する (図 8-a)。

2. 取得した拡張子と一致するファイルをカレントディレクトリのファイル群から取得する。例では、gray.dat, green.dat, black.dat, white.dat の 4 つのファイルがその対象となる (図 8-b)。

3. この取得した入力ファイルから、拡張子ルールを用いて図 8-c のように新しい依存関係を作成する。

```
$1.tar : $2.st
   tar cvf $1.tar $2.st
$2.st : $2.gp
   sort $2.gp > $2.st
$2.gp : $2.dat
   grep color $2.dat > $2.gp
```

(a)入力ファイルの種類の選択

```
gray.dat green.dat
black.dat white.dat
```

(b)入力ファイルの取得

```
tar cvf paint.tar gray.st green.st black.st white.dat
    sort gray.gp > gray.st
        grep color gray.dat > gray.gp
    sort green.gp > green.st
        grep color green.dat > green.gp
    sort black.gp > black.st
        grep color black.dat > black.gp
    sort white.gp > white.st
        grep color white.dat > white.gp
```

(c)適用後のコマンド構造

図 8: 対象ファイルの適用手順 (システムの動的選択)

3 作業の再利用支援インタフェース

作業の再利用のシステムを、図9で示す作業の再利用支援インタフェースに実装する。このインターフェースによって作業の明示的な保持を可能とする。また新しい対象の適用以外の操作をマウスで可能することで操作の簡便性を図る。

コマンド履歴のみを扱ったシステムでは履歴が常に更新するためユーザが実際に作業の再利用に利用できる範囲が狭いという問題点が挙げられる。本インタフェースでは、これを解決するため toolcache の機能を取り入れた。図9の作業の登録場所が、実際の保持を行なう部分である。この機能によりユーザは使用頻度の高い作業を明示的にに保持することが可能となり、適用範囲が狭いという問題点は解決する。またユーザが保持した作業に対して構造的な理解を深め、作業の再利用を行ない易くするために、保持した依存関係の構造をツリー構造で表示を行なった (図 9)。

作業の再利用の操作は「登録」,「更新 (もしくは自動更新)」の2つの手順によって行なう。「登録」は、指定したファイルを出力ファイルとしたコマンドラインの依存関係を作成し、保持することである。「更新」は、保持した依存関係をユーザの直接指定の手法で新しい対象に適用して新しい依存関係を作成し、実行することである。「自動更新」は一般化した依存関係を用い、システムの動的選択の手法により新しい対象を適用して新しい依存関係を作成し、実行することである。

実際の実行画面を図 10に示す。この実行画面は、システムの動的選択の手法を用いて保持した作業の依存関係を一般化し、新しい C のコンパイルの依存関係の作成を行なった図である。

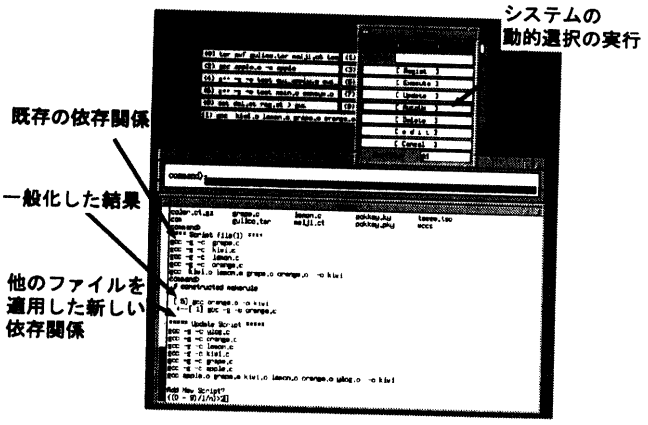


図 10: 実行画面

4 既存の CUI 支援機能との比較

本インターフェースの利用によるユーザの繰返し作業の負担を調べるために、作業の操作時間の測定とアンケート調査を行なった。比較対象は、tcsh、ユーザの直接指定、システムの動的選択の 3 つとした。被験者は、 tcsh 利用経験が 1 年〜3 年の 20 人に対して行なった。実験の所要時間は 1 時間前後であった。実験は以下の手順で行なった。

1. まず被験者にインターフェースの操作説明を行ない、操作を練習してもらう。
2. 次に事前に実行した C のプログラムのコンパイル、TEX 形式のファイルのコンパイル、文書の編集など 21 個のコマンドからなる 5 種類の作業を被験者が確認する。
3. 確認後、被験者に 5 つの作業を異なった入力ファイルを提示し、被験者はそのファイルを見ながら新しい作業を実行する。

この作業を tcsh、ユーザの直接指定、システムの動的選択の 3 つの方法で行ない、その後でアンケート調査を行なう。ここで行なうユーザの直接指定、システムの動的選択の方法は、最初に作業の「登録」の操作を行ない、その後で「更新」または「自動更新」を操作を行なう方法をとった。この実験結果を表 2, 表 3, 図 11に示す。

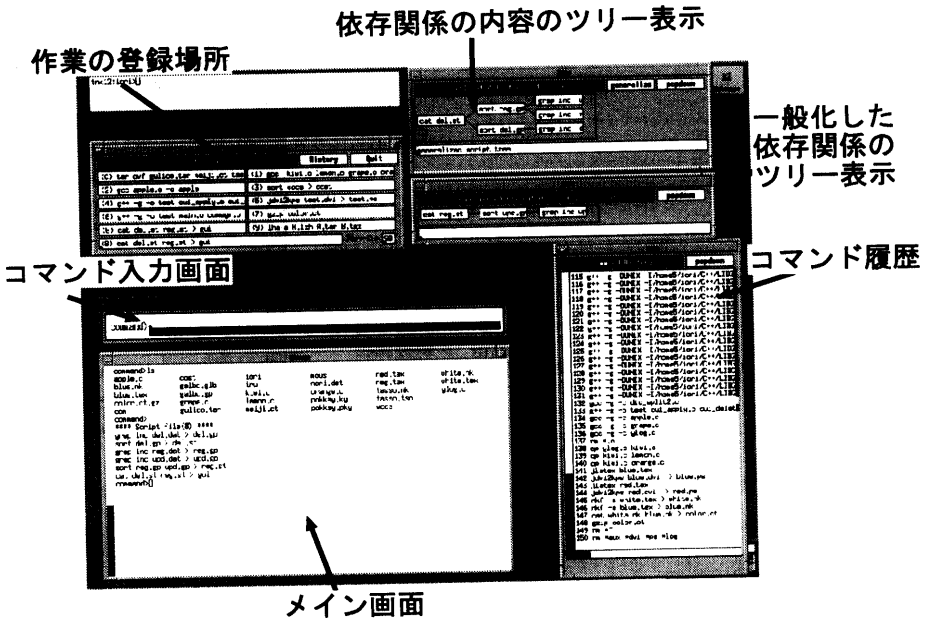


図 9: 実装画面

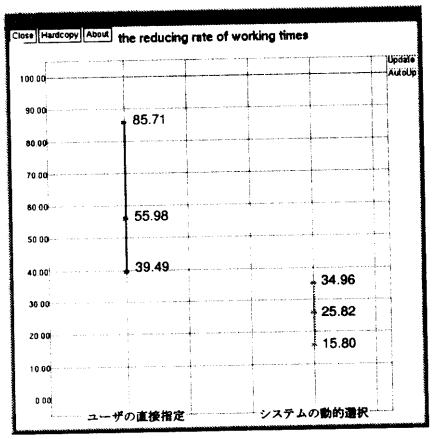


図 11: tcsh に対するユーザの直接指定とシステムの動的選択の作業の短縮率 (単位は%)

表 2 作業時間の計測結果

	tcsh	直接指定	動的選択
平均	16'00"	8'58"	4'03"
標準偏差	911.90	442.40	152.28
tcsh との時間差		7'02"	11'56"
tcsh との作業効率 (最小)		85.71 %	34.96 %
tcsh との作業効率 (平均)		55.98 %	25.82 %
tcsh との作業効率 (最大)		39.49 %	15.80 %

実験結果より、本インターフェースは以下の点で有効性が確認された。

似通った作業の繰返し操作

表 2, 図 11 の作業時間の計測結果のように tcsh を使った作業の再入力よりユーザの直接指定の手法で平均 55.98 %、システムの動的選択の手法で平均 25.82 %に作業の時間を短縮することができた。これは、似た作業の繰返し操作は、tcsh を使った作業の再入力より本インターフェースの方が有効的であることを示す。また表 3 のアンケート結果からコマンド再入力の負担の軽減、ストレスの減少、キーストロークの減少など、被験者から良い印象を受けた感想が多かった。

システムによる新しい作業の作成

システムの動的選択の手法が、ユーザの直接指定の手法に比べて作業時間を短縮することができた。アンケート結果でも、ファイルの自動選択機能が有効であることが確認された。この機能は、マウス操作のみで実行できるので覚えやすく、シェルスクリプトや Makefile の記述または修正と比較してもユーザの入力の負担は軽減されていると思われる。

作業の明示的な保持

tcsh の実験において被験者は、過去に行なった作業がコマンドの履歴の中で古く再利用できにくい場合は、tcsh のヒストリ機能をほとんど使用せず直接コマンドを入力する傾向があった。作業の明示的な保持は、一度作業の保持を行なえばコマンドの履歴に関係なく利用できる。また、2 回目以降は、「更新」のみの操作で

表3 実験のアンケート結果 (単位は人数)

項目	◎	○	△	▲	×	?
tcsh の理解度	1	5	7	7		
make の機能の理解度		3	5	5	5	2
操作の簡単さ	5	13	2			
本システムの機能の理解度	3	11	5	1		
コマンド再入力の負担の軽減	17	3				
キーストロークの減少の程度	13	6	1			
ストレスの減少の程度	9	8	3			
Makefile 作成と比べた負担の軽減	6	2	2	1		9
GUI による入力負担の軽減	6	8	5	1		
入力ファイルの自動選択の有効性	8	10	2			
作業の保持機能の有効性	9	6	5			

よいので作業時間は、実験結果よりも更に早くなることが期待できる。

マウスの併用による操作の簡便さ

コマンドライン入力の場合、各操作を覚える手間が多く、操作に慣れるのに時間がかかるという問題がある。よって各機能にマウスを併用した操作を導入した。この効果は表2の標準偏差より、tcsh より本インタフェースの方がユーザの個人差による影響が少ないという結果から、どのユーザも操作に早く慣れていることが分かる。また被験者からもGUIの実装によるコマンド入力の負担の軽減があると感想を得ることができた。

また、以下のような制限及び限界が生じた。

make rule の利用の制限

全ての依存関係から make rule を作成することはできるが、この make rule から拡張子ルールを作成する時、ファイルの拡張子に依存するので拡張子ルールを作成できる依存関係は制限される。よって特定の作業によってはシステムの動的選択を利用するために、ユーザは予め拡張子の依存関係を考慮して作業を行う必要がある。

システムの動的選択の選択の限界

システムの動的選択は、ユーザが実行した時のカレントディレクトリから入力ファイルを取得するので、1つのディレクトリに同じ拡張子を持つプログラムファイルが2つ以上存在した時、またディレクトリを越えてファイルが存在する時は正しく新しい依存関係を作成することは出来ない。この問題は、新しい依存関係を作成した際にユーザに確認、編集を行なうことで現在は補っている。

入出力ファイルを持たないコマンドを含む作業の再利用

本システムは、作業の実行に make を用いているので Makefile に記述が出来ないコマンドラインを含む作業は再利用が出来ない。

5 おわりに

本論文では類似した作業を繰り返し入力する負担を軽減するため、ユーザの作業を再利用するインタフェースを提案した。本インタフェースにより、ユーザが過去に行なった作業を利用して少ない手間で新しい作業を作成することを可能にした。またシステムの動的選択手法を用いることによりユーザはキーボードから入力を行なうことなしで新しい作業を実行することを可能とした。また、toolcache 機能を加えることによって、簡単に作業を保持することができ過去の作業を容易に確認できるため、再利用の操作を行なうことが可能になった。このことにより、作業時間の短縮、キーストロークの減少、繰り返し作業におけるユーザの負担が軽減されている。課題点としては、一般化した依存関係の利用の制限などがあげられる。

参考文献

1) 中山 健, 宮本 健司, 川合 慧: コマンド履歴からの動的スクリプト生成, WISS94, pp. 155–164 (1994)
2) 宮本 健司: 汎化履歴にもとづく予測インターフェースの拡大, コンピュータソフトウェア, Vol. 14, No. 6, pp. 15–28 (1997)
3) S. Greenberg and I. H. Witten: Supporting command reuse : mechanisms for reuse, Int. J. Man-Machine Studeies, 39, pp. 391–425 (1993)

(1998. 5. 15 受理)

The interface supporting task reuse by generalizing dependency

Iori NISHIDA, Satoru MORITA and Minoru TANAKA

We propose the system which supports the task reuse by generalizing the task which user did. User extracts a task from the generalized task and replaces the objects including in it to new objects. We use two methods in order to generate new task. User replaces a target file to a new file in one method and the system supports by showing new objects to user dynamically in another method. When you reuse a command on a history of C shell, you have to seek it in your history. So, we suggest a interface that can obviously regist the tasks which user uses repeatedly . It is available for a mouse operation owing to be more efficient operation. We confirm to reduce the time which user use for the task and reducing some user's stresses by using the system.