ソフトウェア工学 46－18
（１９８６．２．７）

ユーザ・モデルを用いた
ＬＩＳＰインタフェースの製作と評価

中島 覚　　永田 守男
（慶応義塾大学 理工学部 管理工学科）

要 旨

現在のコンピュータのユーザ・インターフェースは機械中心に設計されているため，そのユーザには，わかりにくい，使いにくいといった不満が多い。もっとユーザにとって使いやすい，さらには各ユーザに対応した人間中心のインタフェースの出現が望まれている。このようなインタフェースを実現するための基礎を与えることが本研究の目的である。

一般にユーザ・インタフェースの研究といってもその範囲は広いので，その中でユーザがシステム上でエラーをおこしたとき，返されるエラーメッセージの内容を各ユーザにとってわかりやすいものにする方法について研究をおこなった。

対象としたシステムはＬＩＳＰとした。ＬＩＳＰシステムのエラーメッセージの表現を各ユーザにとってわかりやすいものにするシステムを作成した。このシステムを作るにあたっては，ＬＩＳＰ上のＦＬＡＶＯＲというオブジェクト・オリエンテッドシステムを用いて，各ユーザごとのモデル化を行なった。このシステムの製作と実際に使ってもらうことによる評価を通して，ユーザにとってそれぞれに使いやすいＬＩＳＰインタフェースについて研究をした。

１．はじめに

最近，コンピュータのユーザが急激に増加してきた。それらのユーザにはコンピュータの専門家から初心者までいろいろあるが，どのユーザでもそのシステムがわかりにくく，使いにくいといった不平をいうことが多い。これはそのユーザ・インタフェースに問題があるわけで，コンピュータのユーザの増加とともに，使いやすいユーザ・インタフェースの出現が望まれている。

この点については，今までハードウェア的な改善は数多くなされてきたが，ソフトウェア的な改善はあまりなされていない。そこで，まずソフトウェア的な観点からユーザ・インタフェースの問題をとりあげてみると

(1) エラーをおこしたとき，そのエラーの場所，原因，及びその修正法などについてあまりサポートされていない。

(2) 同じようなシステムを異なる環境で使用するとき，システムの機能や操作法などのどこが同じで，違うのがどこかがユーザにわかりにくい。

(3) メッセージの内容や用語が専門的でわかりにくい。

(4) システム側の対応がすべてのユーザに対して画一的であるため，ある人にはわかるが別の人にはわからないことが多い。

などが挙げられる。

本研究では上記の問題点のなかで (1),(3) 及び (4)の問題点に注目した。すなわち，各ユーザにあわせてシステム側がその対応を変えてゆくことができれば，使いやすいインタフェースが実現できるのではないかと考えた。このようなインタフェースを作ることによってソフトウェア的に使いやすいユーザ・インタフェースの設計指針を与えることを本研究の目的とした。

ここで考えたのはＬＩＳＰのインタプリタ上で作業をおこなうユーザを対象としたインタフェースを作成した。そのインタプリタ上でエラーをおこしたとき，システムから各ユーザに返されるエラーメッセージをユーザのレベルに合わせてわかりやすい内容にすることを研究の中心にした。

2．研究の概要

ユーザ・インタフェースの問題といっても幅が広い。ここではユーザはＬＩＳＰのインタプリタ上で，何かしらの組込み関数を使用するか，もしくは自ら関数を定義して作業を行うものとする。

一般に，ＬＩＳＰインタプリタ上でユーザが何かエラーをしてしまった場合，システムから返されるエラー・メッセージの内容がユーザにとってわかりにくいことに注目し，もっとユーザにとってわかりやすいエラーメッセージにすることを考えた〔2〕。さらに，そのメッセージを画一的なものでなく，各ユーザの状況に柔軟に対応できるものとすることも目的とした。

そこでユーザがエラーをした場合，各ユーザの知識レベルに対応したわかりやすい内容のエラー・メッセージを返す。また，同一のユーザに対しても，そのユーザの知識レベルの変化に対応してメッセージの内容を変えることを考えた。平均的な知識レベルを持ったユーザにとって使いやすい画一的な対応していた従来のインタフェースではなく，全てのユーザにとってそれぞれに使いやいインタフェースをつくることにした。

そのために，まずユーザのひとりずつをモデル化してとらえた。また，ＬＩＳＰの組み込み関数ごとにそれぞれ三種類のエラーメッセージを用意する。そして，ユーザがある関数についてエラーをしたときに，このひとのモデルになっている情報を利用して，最も適切であると思われるエラーメッセージを出力するシステムを作成した〔4〕。このシステムを我々の研究室にあるＭｉｃｒｏ ＶＡＸ-II 上のＣｏｍｍｏｎ-ＬＩＳＰのひとつＶＡＸ-ＬＩＳＰで作って，さまざまなレベルのＬＩＳＰユーザに実験的に使ってもらい，使用感を聞き出すことでこの方法で作ったインタフェースの評価を行った。

本システムの設計，製作および評価を通して，ユーザにとって使いやすいインタフェースを作ることについての考察を加える。

3．システム構成

3.1 ユーザ・モデル

従来からユーザをモデル化してとらえて，使いやすいユーザ・インタフェースを実現しようとする研究がなされてきた。しかし，そのほとんどの研究で用いられていたモデルはすべてのユーザを単一モデル（しかも時間とともに変化しない静的なモデル）で記述してきた〔3〕。

本研究におけるユーザのモデル化はこれら従来の研究とは異なり，ユーザごとに個人モデルを作る。しかも，このモデルはその内部に時間とともに変化するパラメータをもつ動的なモデルである。各モデルは，パラメータとしてそのユーザのＬＩＳＰの関数の使用回数とエラーの回数についての情報を持っている（図１）。

これらの情報は，ユーザがこのシステムを使って作業をするごとに書き換えられ，これによりユーザの変化を常にとらえられる〔1〕。

ユーザ名 ： A

関数名	使用回数	エラー回数
CAR	(U 5)	(E 2)
CDR	(U 3)	(E 1)
CONS	(U 1)	(E 0)
EQUAL	(U 0)	(E 0)
・		
・		
・		

図１ ユーザモデルの概念図

3．2 オブジェクト・オリエンテッドシステムの適用

こうしたモデルをＬＩＳＰ上のＦＬＡＶＯＲというオブジェクト・オリエンテッドシステムを利用して実際に作成した（図２）。

(1) クラスとインスタンスの概念の適用

ＦＬＡＶＯＲのクラスとインスタンスの概念の適用によりモデルの枠組み（ＵＳＥＲというクラス）をDEFFLAVORによって定義し，各ユーザ・モデルをそのインスタンスとして表現した。

ユーザの情報は関数別にインスタンス変数の値として記述する。ユーザ・モデルの具体的な情報は過去における使用回数とエラーの回数を各関数別に記述した。このようにモデルの枠組みは共通でも，その内部情報によって異なる個人別に対応するモデルを作ることができる。

(2) メッセージ・パシングの概念の適用

この各ユーザ・モデルの情報に関する諸操作手続き（モデル情報の初期化，修正，レベルの設定など）は，DEFMETHODによってクラスＵＳＥＲのメソッドとして定義した。それらの諸手続きの実行はモデルに対してメッセージを送ることによって行なわれる。

3．3 ユーザ・モデルによるレベル分け

ユーザのＬＩＳＰに関する知識は関数の使用回数が多いほど知識レベルは高いと考えられる。しかし，使用回数が多くてもエラーの回数が多いとその知識レベルはあまり高いとはいえないと考えた。したがって，この研究では関数についてのユーザの知識レベルを関数の使用回数とエラーの回数によって推定する。

本システムでは，現在，関数ごとにユーザのレベルを三段階に分けて考えている。そして，システムはユーザの入力したＳ式を調べ，もしエラーがあれば，そのユーザのレベルに対応したエラー・メッセージを返す。もしエラーがなければ普通のＬＩＳＰインタプリタと同じようにＳ式を評価して返す。

このようなユーザとシステムとのやりとりの間，絶えずユーザ・モデルの情報は書きかえられる。そして，ユーザ・モデルの情報をもとにシステムがユーザの知識レベルが変化したと判断すると，エラー・メッセージの内容もそれに合わせて変化させる。またユーザが本システムをぬけたとき，その情報はファイルに記録される。そして再び同じユーザが本システムに入ると，ファイルの情報をもとにモデルが構成される。

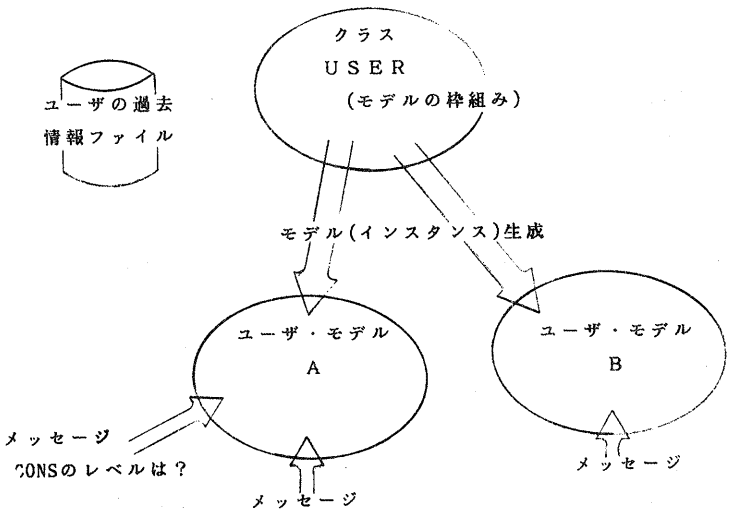

図２ ユーザ・モデルと
システム構成の概念図

なお，エラー・メッセージの内容は日本語で記述した。

4．実行例

ここで実際の実行例を示す。先にのべたようにエラー・メッセージの種類はレベル1からレベル3まで3種類用意した。

例えば，今ユーザの入力したＳ式が

```
(setq X (append 'a '(b c))
```

であったとする。今の場合，ＬＩＳＰのインタプリタは関数apeendを評価しようとしたが，その第1引数にリストではなく，アトムが与えられているのでシステム側はユーザ・モデルの情報に基づきそのユーザにとってわかりやすい内容のエラー・メッセージを返す。

＊ レベル1のエラー・メッセージ

```
エラーがおこった箇所は
  (APPEND 'A '(B C))

関数 APPEND の第1引数にアトムが与えられて
います。APPEND の第1引数にリストを与えて
下さい。

関数 APPEND は以下のように使って下さい。
   (APPEND  '(A) '(B C)) =) (A B C)
   (APPEND  '(A B) '(C D) '(E F)) =) (A B C D E F)
```

レベル1はＬＩＳＰをほとんど使ったことのない，もしくは使ったことはあるが本システム上でＬＩＳＰを使うことが初めてであるユーザとし，メッセージの内容は用例を含んだ詳しいものにした。

レベル1から2，3とレベルが高くなるにつれて，かえって細かな気配りは煩わしいとの配慮から，エラー・メッセージの内容は簡単なものに変化してゆく。

＊ レベル2のエラー・メッセージ

```
エラーがおこった箇所は
  (APPEND 'A '(B C))

関数 APPEND の引数にリストでないものが与え
られています。APPEND の引数はリストでなけれ
ばなりません。
```

＊ レベル3のエラー・メッセージ

```
エラーの箇所は
  (APPEND 'A '(B C))

関数 APPEND の第1引数のエラーです。
```

また，同一のユーザであっても，今後のAPPENDについての知識レベルの変化とともにメッセージの内容も変化してゆく。本システムはこれらのメッセージを日本語で記述し，あとで述べる評価実験のときにも日本語の表示できる端末を使った。

なお，これと同じエラーをすると，現在のＶＡＸＬＩＳＰでは

```
Fatal error in function APPEND (signaled with
ERROR).
The arguments must be lists, received: A
```

というメッセージが返ってくる。

5．使いやすさの評価

5．1 評価実験

このようなユーザ・インターフェースシステムを作成し，実際にユーザに使ってもらうことで評価実験を行った。実験のユーザとしてはＬＩＳＰの初心者からある程度使ったことのある人までの9人にお願いした。その内訳は初心者（ＬＩＳＰの講習を受けたばかりのひと）5人，使ったことのあるひと（卒業研究等でＬＩＳＰを毎日使っているひと）4人である。全員が管理工学科の学生（3年生，4年生および大学院生）であった。

実験の内容は次の通りである。まずはじめに，こちらで指定した組み込み関数をインタプリタのトップレベルで使ってもらう（各関数を用いてできる簡単な例題をやってもらう）。指定した関数とは，

```
car  cdr  cons  list  append
null atom equal listp member
assoc mapcar  + - * / setq
```

の１７種類である。次にｄｅｆｕｎを用いて簡単な関数を自分で定義して使用してもらう。ここで定義するのは，

(1)．第１引数のリストの２番目に第２引数を挿入する。
(2)．引数のリスト中の最後の要素を取り出す。
(3)．引数のリスト中の要素を逆にする。
(4)．引数のリスト中の要素数をカウントする。
(5)．階乗の計算。
(6)．べき乗の計算。
(7)．その他

のうち４つほど選んで作ってもらった。これらの作業をしたあとで，その使いやすさについて定性的な評価をしてもらった。

ひとりのユーザにこれらの実験をしてもらうために要した時間はほぼ１～２時間である。

なお，こうした評価には，心理学的な実験の方法（たとえば実験群と制御群に分けた被験者に同じプログラムを作ってもらって時間を計るとか，プロトコルアナリシスをするなど）によって計量的なデータをとることも考えられる。しかし，我々の目的からすると，ユーザの声を直接聞くことが重要と考え，計量的なものが必ずしも役に立つデータとも限らないので，ここで述べたような実験をしたわけである。

５．２ 評価結果

評価実験から得られたユーザの評価をまとめてみると

＊メッセージの内容が段階的に変化してわかりやすい。

メッセージの内容を日本語で記述し，それに段階を設けたことはユーザに好評であった。特にレベル１のメッセージのように関数の使用例を組み入れたことは好評で，ユーザにとってマニュアルを参照するといった手間がかからず，関数の習得がはやく容易におこなえた。このようにメッセージの中に使用例を含めたことはシステムの使いやすさに大いに寄与したと考えられ，今後もインタフェース設計にあたっては，用例をうまく含めるべきであると考える。ユーザ側にも，もっと用例を豊富に充実したものにしてほしいという要望が多かった。

しかし，そのレベルのメッセージが本当に今のユーザに対応したものかどうか，ユーザの変化を本当にうまくとらえているかどうかなどについては，まだはっきりした評価を得ず，さらに実験，検討を必要とする。

＊使いやすく，気楽にエラーができる。

エラーをおこすことに恐れるユーザが多いものであるが，このシステムを使っているユーザはエラーをおこすことに恐れるよりもむしろ気楽にエラーをおこしていた。特に，ＬＩＳＰを使ったことのあるユーザの多くはある関数の使用法を知るためにわざと関数名だけ入力するという現象がみられた。これはエラーメッセージの内容がわかりやすいためであると考えられる。

＊ユーザの対応

ユーザに対してレベル１のメッセージを返すと，メッセージの内容よりも関数の使用例を中心に見て理解しようとする傾向が強かった。メッセージの内容がレベル２，３へと変化すると，内容だけで理解できていた。このことからメッセージに段階をもたせることがユーザにとって使いやすかったのではないかと考える。また，おもしろいことに，メッセージの内容が変化していることに気がついたユーザはほとんどいなかった。

＊ スピードが遅い。

関数をひとつひとつ評価するたびに，エラーがあるかどうかをたえずチェックしているため，システム全体のスピードは従来のＬＩＳＰインタプリタよりも遅くなってしまった。しかし，ユーザの評価としては我慢できるスピードであった。今後，システムのスピードを挙げることが問題である。

＊ 使用できる関数の種類，診断できるエラーの種類が少ない。

現在のシステムでは，使用できる関数が１８種類とまだ少なく，また，各関数ごとに診断できるエラーの種類が少ないため，これらの点が，先のスピードの遅さとともにユーザにとって不便であった。

また，システム側が検出したエラーがユーザの意図したことと食い違いを生じたりした（たとえば，カッコのつけ忘れなのに引数の数に関するエラーメッセージがでたり，変数に値が入っていないのか，QUOTEの付けわすれなのかを判断することが難しいなど）。使いやすいインタフェースを実現するのに，ユーザの意図したことを判断するということはかなり難しい問題である。

このように，メッセージの内容に段階をもたせること，メッセージの内容のわかりやすさ，システムの使いやすさについては，従来のシステムより良いという評価が多く，全体的な評価として使いやすいインタフェースが実現できたのではないかと考える。

6．おわりに

このシステムの評価として，ＦＬＡＶＯＲにより容易にモデルの作成ができ，また，メッセージのレベル分けにより，ユーザにとってエラー・メッセージがわかりやすくなった。しかし，三段階のレベルではまだ不十分で，もっと細かい分析と配慮が必要である。

今後の研究方針としては

(1) 本システムの改善，拡張

ユーザ・インタフェースの設計，製作にとっては，そのシステムを実際にユーザに使ってもらって評価し，その結果をもとにシステムを改善してゆくことが重要である。したがって，本システムも多くのユーザに使ってもらって評価し，その結果をもとにさらに各メッセージの改良や，システムの拡張，改善をはかってゆくことが課題である。

(2) 他システムへの応用

このようにユーザにあわせてシステムがその対応をかえてゆくことによって使いやすいインタフェースを実現しようとする研究にいくつかの基礎を与えたと考える。これをもとに他のシステム（たとえばＯＳ，エキスパートシステム，ＣＡＤシステムなど）に応用することを考えている。

一般的に言えば，ここで提案したことは，ソフトウェアのシステムが利用者についての情報を知ってそれを活用することでユーザインタフェースの改善が図れるのではないかというものであある。そして，その改善に要するコストが莫大なものとはならないことも示した。ワードプロセッサの個人用の辞書とかＣＡＩのシステムなどでは個々の利用者への対応が少しではあるが考えられている。もっと広くソフトウェア一般にここでの提案が生かされるようになることを期待したい。

［参考文献］

〔1〕 永田：動的な利用者モデルに基づくシステム構築法，第31回情処大会 2G-6 (1985)

〔2〕 中嶌：ＬＩＳＰプログラミングにおける初心者診断システムの設計・製作，慶応義塾大学 修士論文 (1981)

〔3〕 Card, Moran and Newell : The keystroke-level model for user performance time with interactive systems, CACM Vol.23 No.7 (1980)

〔4〕 中島，永田：ＬＩＳＰの動的な利用者モデルによるユーザ・インタフェースの製作 第31回情処大会 6E-6 (1985)