ソフトウェア工学 58－2
（１９８８．２．４）

ソフトウェア設計に関わる作業プロセスのPrologルールによる記述について

大木敦雄、　落水浩一郎

静岡大・工学部・情報知識工学科

ソフトウェアの設計プロセス、および、設計者の諸活動に関わる情報構造を明らかにし、それらにもとづいた設計支援環境の構築を進めている。
ソフトウェアの設計活動を明らかにすることで、1)ツール起動などの煩雑な操作の自動化、2)自身の設計プロセスの分析・改善、3)他人に設計プロセスを伝える、4)良い設計プロセスを共有する、などの効果を期待できる。「設計活動プロセスをPrologのルールで記述する」という方針のもとに、上記の問題に関する研究を進めている。
本報告では、設計プロセスを記述するためにPrologに追加した２つの制御構造とこれを実行するために作成したプロトタイプpshellについて述べる。

DESCRIPTION OF SOFTWARE DESIGN PROCESSES BY PROLOG RULE

Atsuo OHKI　and　Koichiro OCHIMIZU

Dep. of Computer Science, Faculty of Engineering, SHIZUOKA University

5-1, Johoku 3chome, Hamamatsu, 432 Japan

We are going to make software design processes and related information structure clear,　and to develop a software design support environment based on them.
Our goals are　1)to automate tools invocations without complicated operation, 2)to analyse and improve one's design processes, 3)to communicate one's design processes to others, and 4)to share good design processes.
Initially our research effort is concentrated to describe design processes by Prolog rules.
We, here, report two new control structures of Prolog to describe design processes, and prototype of pshell to execute them.

１．はじめに

ソフトウェアの設計活動を明らかにし、支援環境を構築する試みが国内外でなされている。例えば、Arcadia[1]は、プロセス・プログラミング環境を実現することを目的としている。ソフトウェアの開発過程で扱うオブジェクトを中心にとらえ、それらにたいする操作を厳密に記述する試みである。

ソフトウェアの設計活動を明らかにし、それを何らかの方法で記述できれば、以下のような効果が期待できる。

1)ツールの起動などの煩雑な操作を自動化できる。
2)自身の設計プロセスの分析・改善が容易になる。
3)明確に記述されたものが残るので、これを用いて、他人に設計プロセスを伝えることができる。
4)良い設計プロセスを他人と共有できる。

我々は、「設計活動プロセスをPrologのルールで記述する」という方針のもとに、上記の問題に関する研究を進めている。
その理由は、Prologの持つ制御構造とソフトウェア設計という問題解決プロセスとの以下ような対応の良さにある。

1)バックトラックと代替ルールの選択
設計プロセスは直線的に進行するのではなく、行きつ戻りつの繰返しで進行する。Prologのバックトラックは、これを自然に表現できる。

2)プロセスの宣言的記述
「述語の評価」を「活動の起動」に、「活動の成功／失敗」を「述語の評価結果（真偽）」に、「中間／最終成果物」を「項」に対応させることで、ルールによりプロセスを宣言的に記述できる。

本報告では、設計プロセスを記述するためにPrologに追加する２つの制御構造について述べ、これを実行するために作成したプロトタイプpshellについても述べる。

本研究は、ソフトウェア構造設計支援環境Software Designer's Associates(SDA)[2]における一研究活動であり、モデルベース統合方式の検討の一活動である[3][4]。

２．Prologによる設計活動の表現

本節では、Prologのルールを設計活動プロセスとして解釈することを述べ、設計プロセスを表現する上で必要な２つの制御構造について述べる。

２．１　Prologの制御構造と設計活動プロセスとの対応付けの分析

Prologの解釈には、1)宣言的解釈と2)手続的解釈がある。述語Ｐの定義（ルール）

Ｐ　：－　Ｑ，Ｒ，・・・，Ｓ．　(2.1)

は、1)に従えば、

述語Ｑ，Ｒ，・・・，Ｓがすべて満足されたとき、述語Ｐが満足される

であり、2)に従えば、

手続きＰは、手続きＱ，Ｒ，・・・，Ｓを順に呼び出すことで実行される

と解釈される。Prolog処理系は主に2)の解釈に従い、ゴールＰを達成するために、(2.1)のサブゴールＱ，Ｒ，・・・，Ｓを順次達成することを試みる。この過程であるサブゴールが偽と評価されるとバックトラックを起こし、もっとも近い分岐点にもどり、再評価を試みる。

設計プロセスをルールで表現するために、述語を活動とみなす。つまり、「述語の評価」を「活動の起動」に、「活動の成功／失敗」を「述語の評価結果（真偽）」に、「中間／最終成果物」をルールの「項」に対応させる。こうすると、(2.1)は次のように解釈できる。

活動Ｐは、順に活動Ｑ，Ｒ，・・・，Ｓを行うことで達成される

活動の成功／失敗に応じ、対応する述語の真偽を決定し評価を行う。バックトラックは次に示すように行われる。

２．２　制御構造の拡張

Prologには、バックトラックを制御するためのカット演算子だけがある。これは、上でのべた2)の解釈に従うProlog処理系の解の探索を制御する目的で導入されたものである。ソフトウェア設計の活動を表現する場合には、これだけでは不十分である。ソフトウェア設計活動では、

1)一連の活動を、所定の結果が得られるまで繰返す
2)ある成果物を、いろいろな角度から検討する

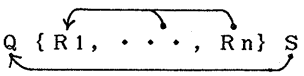
などの行為を表現できる必要がある。
そのために、２つの制御構造を導入する。

２．２．１　繰返し作業の表現

上の1)を表現するために繰返しを

・・・, Q {R1, ・・・, Rn} S, ・・・.

で表現する。記号‘{’はR1に無数の分岐点があるようにし、記号‘}’は、対応する‘{’との間にあるすべての分岐点を取り去る。{と}で囲まれたRi（1≦i≦n）を順次実行したときに、ある活動Rj（1≦j）が失敗すると、R1を分岐点とみなして再評価を行う。さらに、{と}で囲まれた分岐点は、その外部からは見えない。つまり、S以降の活動が失敗しても、{と}の中にはバックトラックしない。バックトラックは以下のようになる。



２．２．２　多角度からの検討の表現

多角度からの検討は、並列的に複数の活動を行うことである。このために、並列的な実行を

・・・, Q (R1 | ・・・ | Rn) S, ・・・.

で表現する。括弧で囲まれたRi（1≦i≦n）は同時に実行される。すべてが、成功すれば実行はSに進む。もし、１つでも失敗すれば、すべてのRiの実行を直ちに終了し、Q以前の分岐点にバックトラックする。また、S以降から括弧の中にバックトラックすることはない。つまり、バックトラックは以下のようになる。

２．３　設計プロセスの記述例

例えば、デザイナのプロセスは以下のように記述できる。このような、ルールの記述の集りをスクリプトと呼ぶ。

```
designer_action(User_spec,Arch_design) :-
  { listup(User_spec,IO_data_and_event),
    find_threads(IO_dataand_event,Threads),
    write_rqt_spec(Threads,Some_notation),
    validate_it(Threads,Some_notation,
                Results_of_analysis) }
  apply_methods(Some_notation,Arch_design),
  propose_arch_design(User_spec,Arch_design).

determine_components(Arch_design,Components) :-
   break down Arch_design to Components.

propose_arch_design(User_spec,Arch_design) :-
  talk with conceptualizer and validate Arch_design.

plan_project(Component,Work_assign) :-
  talk with conceptualizer and validate Work_assign.

start_work(Work_assign) :-
  { assign_tasks(Work_assign, P1, ... , Pn)
    ( programming(P1) | ... | programming(Pn) ) }
```

３．スクリプト解釈実行系のプロトタイプ
－　pshell　－

C-Prologを拡張して、ルールで記述した設計プロセスを解釈・実行するpshellのプロトタイプを実現した。表１には
1)ウィンドウシステムとのインタフェース
2)UNIXコマンドの実行
のために追加した述語の一覧を示す。

付録に、このpshellを用いたツールの起動

実行例を示す。ツールを起動する際の煩雑な操作をルールで記述した自動化の例である。

4．設計プロセスの並行性について

現在実現したレベルでは、「はじめに」の3)、4)で述べた目標にたいして不十分である。

1)現在のpshellは、ユーザが指定した活動（ルール）を逐次的に実行（評価）するだけである。ある局面で何種類かの作業を行える自由度がある場合、どの作業を選択するかはユーザに任せるべきある。これは、2節で述べた並列性（この場合、作業を強制する）とは別の意味で並列的にルールまたはスクリプトの評価を可能にする機構が必要であることを示す。

2)ルールの記述に階層性を取り入れる。

ツール起動に加えて、各種の設計方法論を

ウィンドウ・システムとのインタフェース	
stream_tell(S)	出力ストリームをオープンする
stream_told	出力ストリームをクローズする
stream_telling(S)	出力ストリームをSに切り換える
stream_exists(S)	出力ストリームSの存在の有無
stream_tell_again(S)	出力ストリームSの再オープン
stream_write(T)	ストリームにTを出力する
stream_writeq(T)	ストリームにTを出力する
stream_putc(N)	ストリームに文字Nを出力する
stream_tab(N)	ストリームにタブを出力する
stream_nl	ストリームに改行を出力する
create_window(W)	ウィンドウを作成する
destroy_window(W)	ウィンドウを削除する
map_window(W)	ウィンドウを画面に表示する
unmap_window(W)	ウィンドウの表示を消す
map_stream_to_window(S,W)	ウィンドウにストリームを割付ける
オブジェクトの参照	
is_object(ObjName)	オブジェクトか否かの確認
attrib_of(ObjName,AttrList)	オブジェクトの属性リストを取得
value_of(ObjName,Val)	オブジェクトの属性の値を取得
ファイル操作	
file_state(FileName,Type,Value)	ファイルの管理情報の取得
chdir(NewDir,OldDir)	作業ディレクトリの変更
file_list(Type,FileNameList)	ファイル名一覧の取得
UNIXコマンドの実行	
fork_exec(Cmd,Arglist)	コマンドを実行する
fork_child(P,Pid)	Prologのプロセスをフォークする
exit_child(S)	子プロセスを終了する
wait_child(Pid,Stat)	子プロセスの終了を待つ
sence_child(Pid,Stat)	子プロセスの状態を取得する
change_xterm	画面の切り換え

表1　拡張述語一覧

表現するスクリプトやそれらを適用する上でのノーハウを記述するような場合、これらを階層的に記述する機構が必要である。
3)ルールの起動条件を分類しておく必要がある。

一方、成果物（オブジェクト）を管理する情報格納庫についても以下の点についての考察が残されており、pshellとの関係もつめる必要がある。

1)成果物の登録／状態管理／履歴管理機構の組込み。
2)ある作業に関わるルール／スクリプトの管理方法。
3)分散環境での共同作業を支援する上での情報格納庫の役割。

さらに、設計作業が進行していく過程で実際のスクリプトの内容が明らかになるような場合の扱いについても考察が必要である。

設計活動を明らかにするためには、各種の設計技法をルールで記述したり、実務での設計活動を記述する等の努力が今後必要である。

参考文献

[1] R.N.Taylor, L.Clarke, L.J.Osterweil, J.C.Wileden and M.Young, "Arcadia: A Software Development Environment Reseach Project", Proceedings of the 2nd International Conference on Ada Applications and Environments, Miami Beach, Florida, April 1986, pp.137-149

[2] W.E.Riddle, "Software Designer's Associates: A Preliminary Description", the 20th HICSS, January 1987, pp.371-381

[3] K.Kishida, T.Katayama, et al., "A Novel Approach to Software Environment Design and Construction", (will appear on 10th ICSE, April 1988)

[4] 「ソフトウェア設計環境の新しい統合技術」、第１回ＳＤＡシンポジウム、東京、1987年11月

付録

ここでは、pshellによるツール起動の自動化の例を示す。

otagsはユーザの対話的な指示に従い、Cプログラム中の名前や式のデータ型を提示するツールであり、Emacsエディタの１つのコマンドとして使用する。otagsを使うには、関連するソースから解析情報を抽出したデータベースファイルが必要である。

図１に、otagsの起動ルールを示す。始めに、解読対象のソース・ファイル名を取り出す（述語otags)。ソースに対応したデータベースファイルが存在する場合は、直ちにotagsを起動するためにEmacsエディタを起動するが、データベースファイルが存在しない場合には、それを作成した後、Emacsエディタを起動する(述語do_otags)。otagsを起動するための準備をルールで記述してあるので、ユーザは起動の詳細を知る必要はない。

図２(a)は、pshellを起動した画面例である。最下のウィンドウが、pshellのウィンドウである。ここで、otagsを起動すると、必要な操作がルールとして実行され、図２(b)に示すような画面になる。画面中ほどにできた３つのウィンドウがotagsに関連したものである。

```
otags(O) :-
        is_object(O), !, otags_proj(O).

otags(O) :-
        write(O), write(' is not object.'), nl.

otags_proj(O) :-
        attrib_of(O,A), member(src,A), !,
        cat_name(O,':src',Osrc), value_of(Osrc,V), !,
%       write('value is '), write(V), nl,       % debug
        value_of(O,V0), chdir(V0,D), ls_dir(V), !,
        do_otags(V), chdir(D,_).

otags_proj(O) :-
        write(O), write(' has no src.'), nl.

do_otags(V) :-
        chdir(V,_),
        exists('OTAGS'), write('OTAGS exist'), nl, !,
        exec_otags.

do_otags(V) :-
        write('make OTAGS ...'), nl,
        get_c_file(L), fork_exec(otags, L),
        write('    done.'), nl, !,
        exec_otags.
do_otags(V) :-
        write('something is wrong.'), nl.


exec_otags :- emacs.
```

図１　otags起動ルール

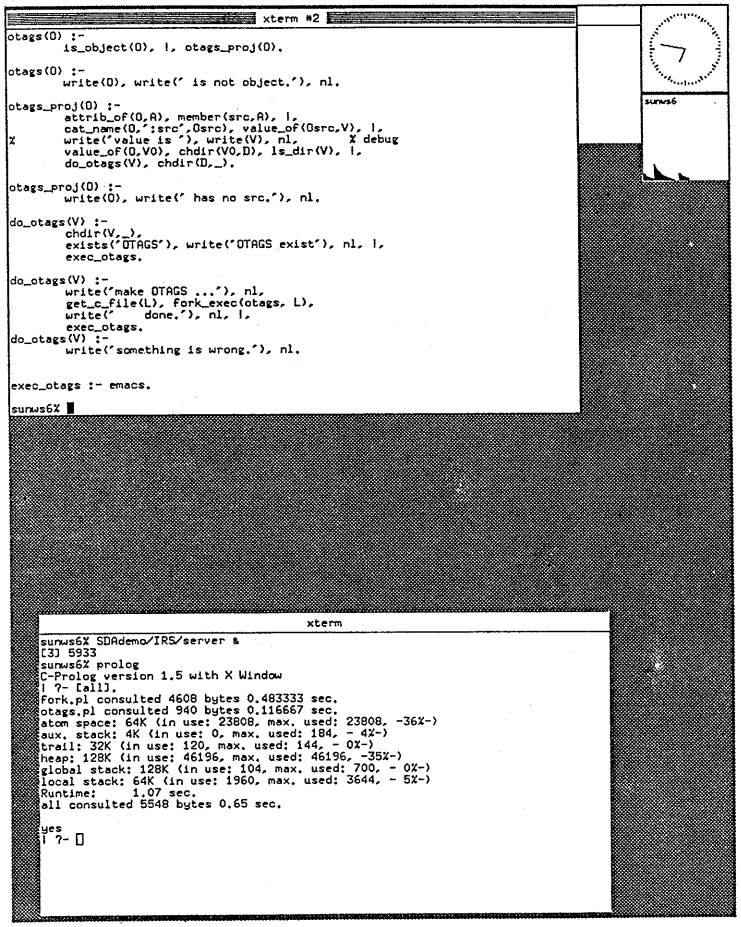


図２(a)　pshellの起動

最下部のウィンドウでpshellが起動される。

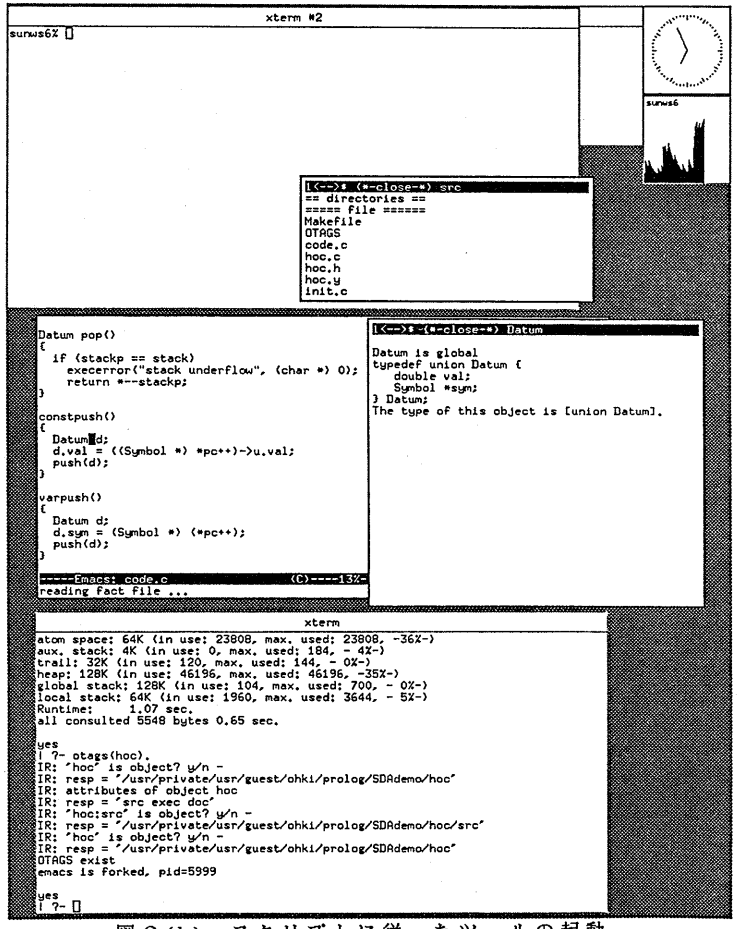


図２(b)　スクリプトに従ったツールの起動

中段２つのウィンドウは、pshellにから起動されたツールotagsが開設したウィンドウ。
Ｃコードの解読作業を行う。

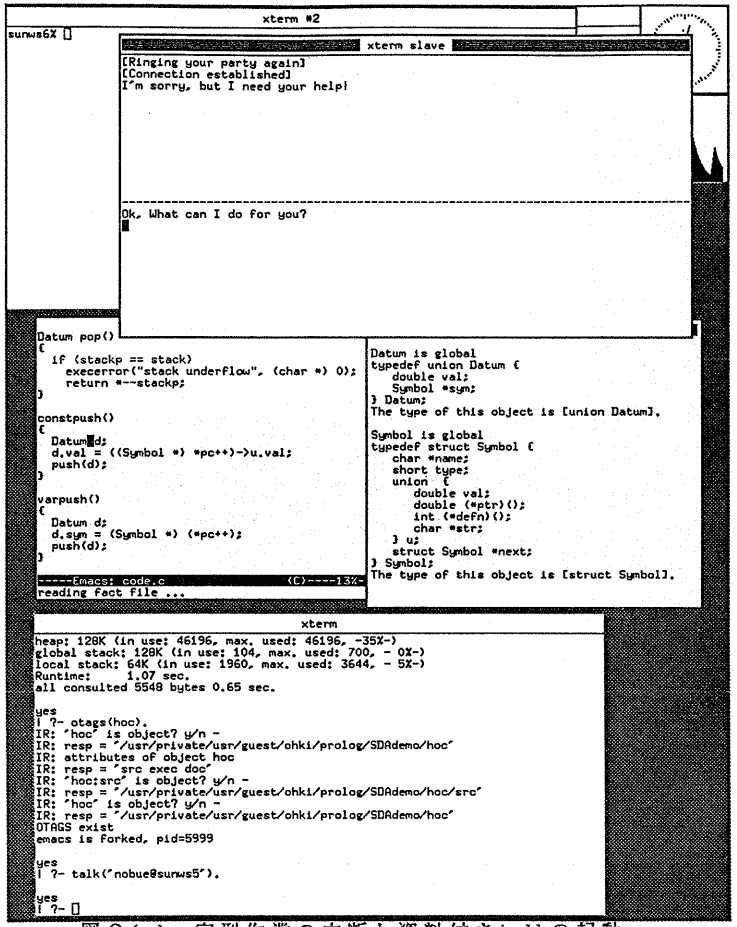


図２(c)　定型作業の中断と資料付きtalkの起動

解読作業中を中断し、プログラム作成者に説明をもとめる。pshellウィンドウからtalkを起動する（最上段のウィンドウ）。

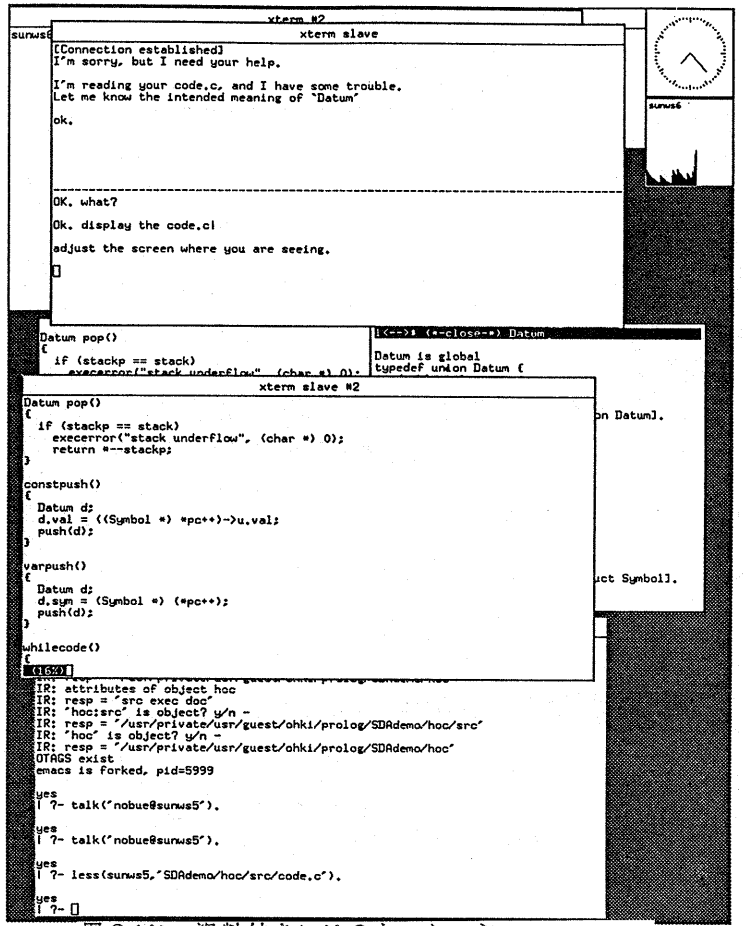


図２(d)　資料付きtalkのセッション

talkによる会話中に資料を提示する必要が生じた。pshellウィンドウから、双方のスクリーンに指定した資料を提示する（中段最上部のウィンドウ）。

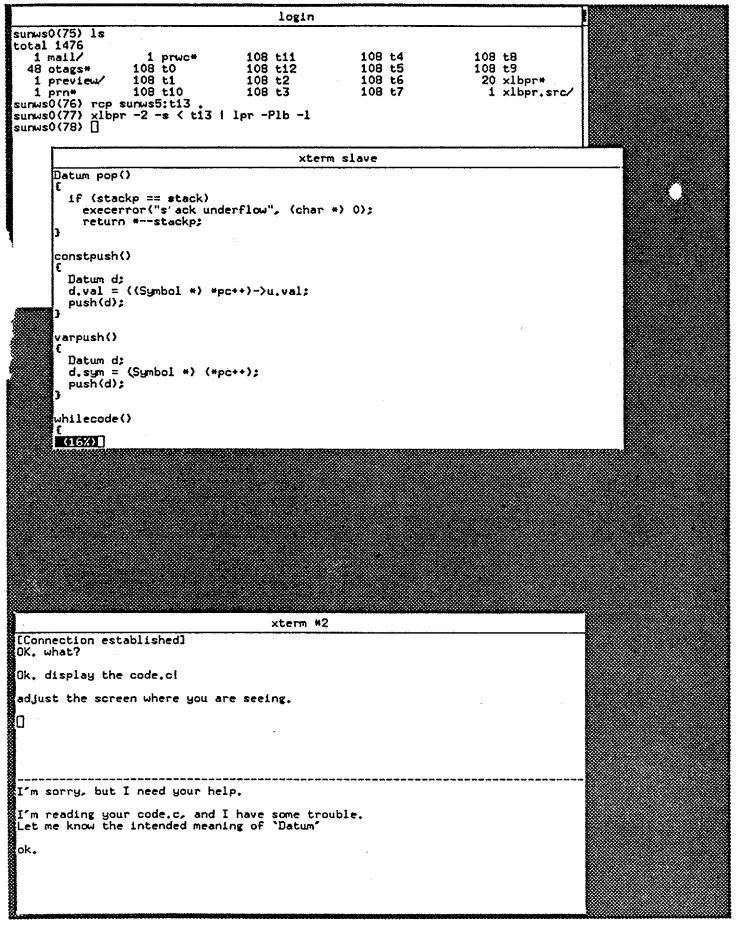


図２(e)　相手側のウィンドウ

図２(d)の状態のときの相手方の画面

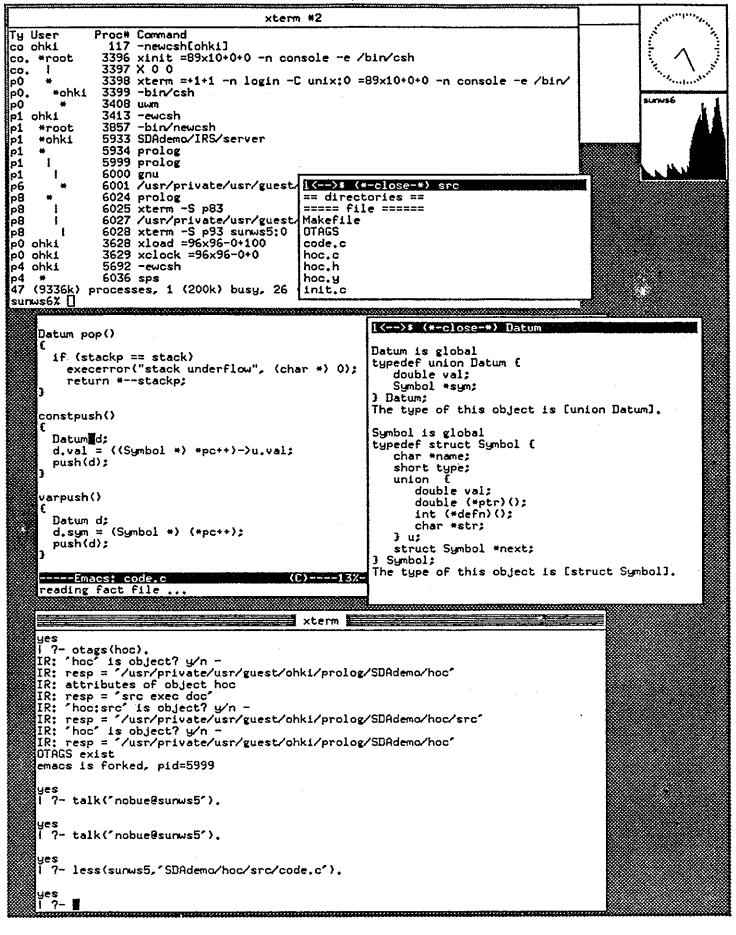


図２(f)　定型作業への復帰

図２(b)の定型作業に戻った状態