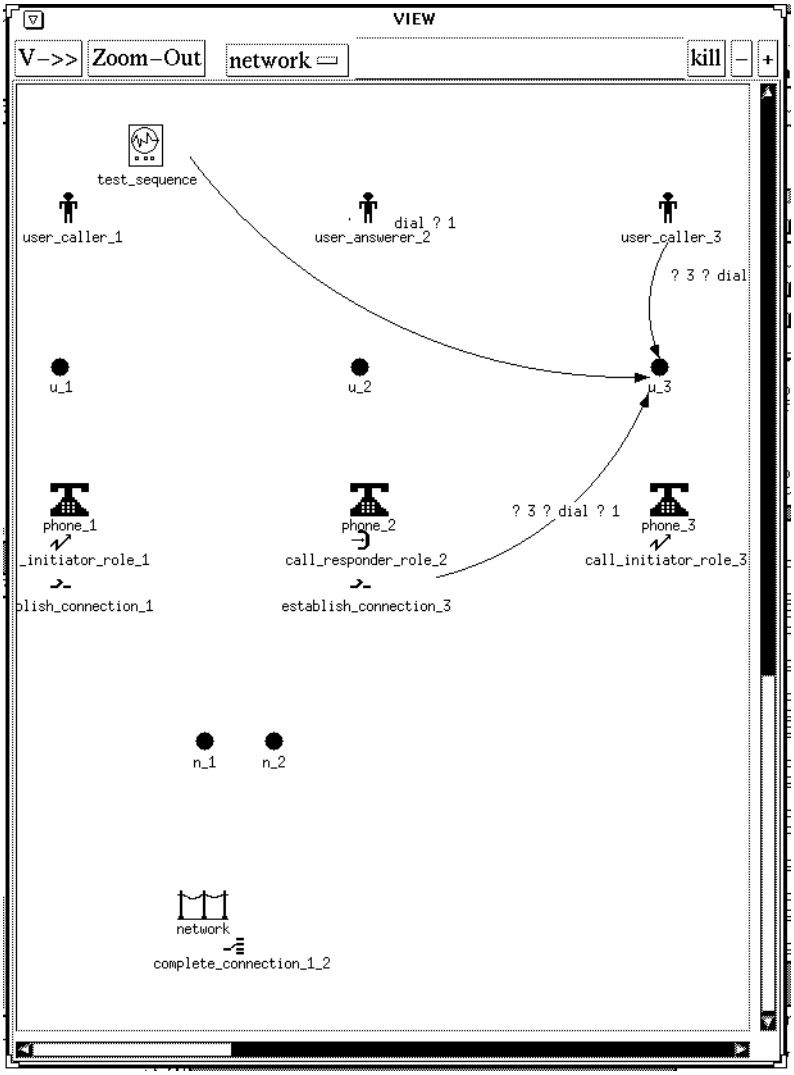
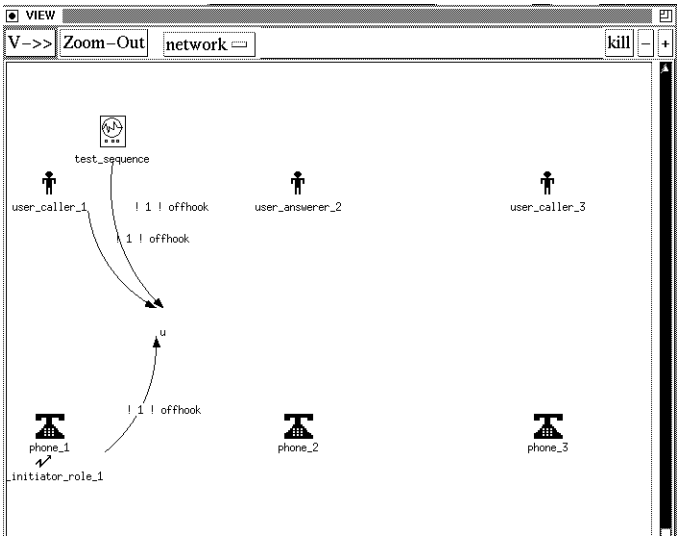

VIEW
V->>
Zoom-Out
network
kill
test_sequence
user_caller_1
dial ? 1
user_answerer_2
user_caller_3
? 3 ? dial
u_1
u_2
u_3
phone_1
phone_2
? 3 ? dial ? 1
phone_3
_initiator_role_1
call_responder_role_2
call_initiator_role_3
blish_connection_1
establish_connection_3
n_1
n_2
network
complete_connection_1_2

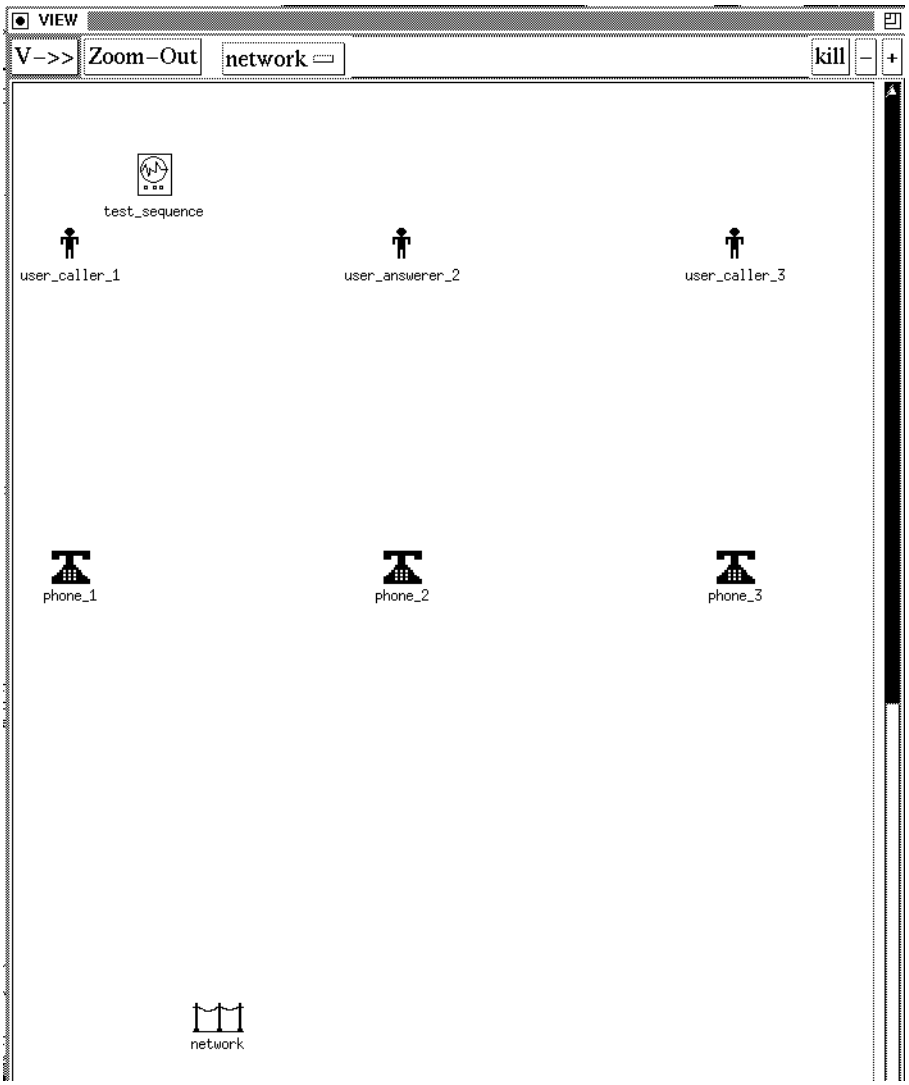


- keep entering carriage retunrs to see more actions appearing

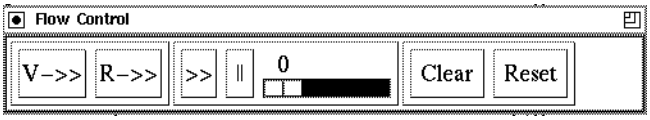
example showing gates instantiation

should be the same TCP/IP address as with the demon start up command.

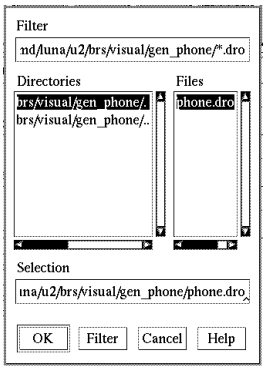
At this point you should see the node creation step execute by drawing icons on the screen:



- ensure your mouse is on the prototype activation window.
- press a cariage return to execute the next available action and display it on the screen

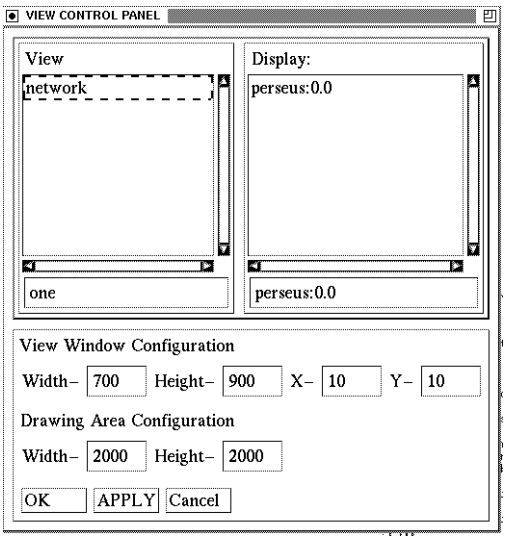


- click the rules icon to select the rules file and click on the .dro file that will appear in the files listbox:



- Click the OK button to exit this window.

- click the view button of the Demon window and you should see a new window where the visualization will be carried out.



- Click on the network entry of the view listbox.and then on the OK button.
A blank display window will then appear.

- click on the run button of the flow control window (>>) to enable your actions to be displayed.

4.2 Running your Lotos prototype

In the other window, run your topo prototype of your specification by typing the command:

<spec_name> <work station name> 1444

where <spec_name> is the name of your specification without its radical, and 1444

4. Running the animation (visualization)

open two windows and place yourself in the directory where you Lotos specification resides.

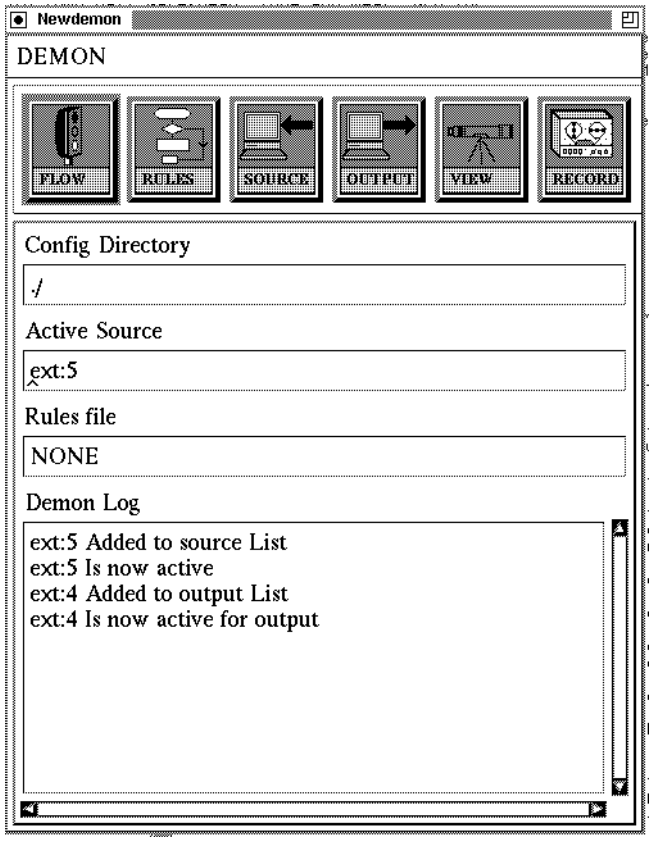
4.1 Startin up the Demon server

In the first window start the Demon server using the command:

boot demon “” ext 1444

where 1444 is used as a TCP/IP port number.

You should see the Demon window:



- click on the flow icon to obtain a tape recorder like window:

```
  idle -u  -t$(SPEC).at < $(SPEC).lsf > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).ldi:   $(SPEC).lsf
  idle -s -r  -t$(SPEC).at < $(SPEC).lsf > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).ldc.c $(SPEC).ldc.hh: $(SPEC).idl
  d2h < $(SPEC).idl > topo.tmp
  $(MOVE) topo.tmp $(SPEC).ldc.hh
  d2c  -p$(SPEC).ldc $(SPEC).ldc.hh < $(SPEC).idl
  touch $@

$(SPEC).lbm:   $(SPEC).lsf
  om -f  $(SPEC) > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).lbc.c: $(SPEC).lbm
  omlbC -i -D $(SPEC).ldc.hh -o $(SPEC).lbc  $(SPEC)
  touch $@

$(SPEC).dro:: $(SPEC).drs
  drp $(SPEC).drs

clean:
  $(RM) $(SPEC).dro $(SPEC).lbc.c $(SPEC).lbm $(SPEC).ldc.c $(SPEC).ldc.hh\\p      $(SPEC).ldi $(SPEC)
$(OBJS) $(SPEC).lfe $(SPEC).lsa $(SPEC).lss \\p      $(SPEC).lsf $(SPEC).lcr $(SPEC).cr $(SPEC).idl $(SPE-
C).lot $(OBJS)
```

The only value to be modified is the variable SPEC at the top that receives you specification name. In the above example replace the statement SPEC = xtp by SPEC = <your_spec_name>

Warning: this makefile is for Topo version 3R1 only. Different version of topo have shown that the makefile varies. Consult your local software support manager to know what makefile to use.

Also you must ensure that you have a copy of file driver.c and a copy of a Demon resource file Newdemon in your main directory. Also, if you are using the standard data type make sure to include the mod_is library in the makefile.

separate compilation of the Demon rules program

enter the command: **drp <specname>.drs** and you should see the following message:

```
drp Version 3.0

Copyright and all ancilliary rights of whatever nature are vested in MARI
Applied Technologies LTD whose registered office is situate at MARI House,
Old Town Hall Gateshead, Tyne and Wear, NE8 1HE.

No part of this work may be altered , modified, combined, varied, enhanced
sold, leased licenced, sub-licenced, reproduced or otherwise dealt with in
any form by any means without the prior written permission of MARI Applied
Technologies Ltd.

No software or other program shall be written or developed based on any
```

You should use the following makefile because the Topo compiler requires different steps in the compilation.

In order to compile your application you need three files:

- the Lotos specification (radical ".lot"
- your Demon rules file (radical ".drs")
- a copy of the driver.c file that contains the main function and the Demon message encoders and various other visualization auxiliary functions.

```
MOVE=mv

SPEC=xtp
OBJS= $(SPEC).lbc.o $(SPEC).ldc.o $(SPEC).o

DEMONLIB = /net/jupiter/usr85/SEM/DemonV3.0.1/lib
DEMONINC = /net/jupiter/usr85/SEM/DemonV3.0.1/include

TOPO=/net/jupiter/usr85/SEM/Lite/lite-components/TOPO_3R1
TOPOLIB=$(TOPO)/lib
TOPOINC=$(TOPOLIB)
USELIB=bool_nat

CFLAGS=-g -I$(TOPOINC) -I$(DEMONINC)

$(SPEC): $(OBJS)
  $(CC) -L$(TOPOLIB) -L$(DEMONLIB) -o $@ $(OBJS) -lotos -lkaos -lkaos -lmes

$(OBJS): $(SPEC).ldc.c $(SPEC).lbc.c $(SPEC).c

$(SPEC).lot: $(SPEC).sdt
  sdt2ao $(SPEC).sdt -g

$(SPEC).lfe:   $(SPEC).lot
  lfe  $(SPEC).lot > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).lsa:   $(SPEC).lfe # $(TOPO)/stdlib/ditupm.lsa
  lsa -l $(TOPO)/stdlib/ditupm  < $(SPEC).lfe > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).lss:   $(SPEC).lfe # $(TOPO)/stdlib/ditupm.lsa
  lsa -s  < $(SPEC).lfe > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).lsf:   $(SPEC).lfe # $(TOPO)/stdlib/ditupm.lsa
  lsa -l $(TOPO)/stdlib/$(USELIB) -f -C  < $(SPEC).lfe > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).lcr:   $(SPEC).lfe $(SPEC)
  lsa -c  < $(SPEC).lfe > topo.tmp
  $(MOVE) topo.tmp $(SPEC).lsa

$(SPEC).cr: $(SPEC).lcr
  ast2cr -n$(SPEC) > topo.tmp
  $(MOVE) topo.tmp $@

$(SPEC).idl:   $(SPEC).lsf
```

Exemple:

```
process complete_connection[n](PN,C:number):noexit:=
    (*# region 9  instance_parms  2  instantiated_gates #*)

  n ! C ! ring
  ; n ! C ! connect
  ; n ! PN ! connect
  ; stop

  endproc
```

will generate the name complete_connection_1_2 for instance complete_connection[n](1,2).

2.0 Generating the annotated Lotos specification

Run the program Lotos_vis_gen as follows:

lotos_vis_gen <spec_name> -g

where <spec_name> should have a radical other than ".lot".

Warning: We recommend the use of the radical "**.sdt**" or "**.vis**". This is because the output is saved by default in a file with a radical "**.lot**".

and you should see the following messages:

```
Gesellschaft fuer Mathematik und Datenverarbeitung
Forschungszemtrum fuer Offene Kommunikationssysteme
Standard Data Type to Act One Translator

(C) 1993

Translation of file: phone.sdt starts !

Parsing and syntax checking start !
Parsing and syntax ckecking were successful !

Visualization: annotations generation starts
Visualization: Generation was successful !

Unparsing starts !
Output is written to file: 'phone.lot'

Unparsing successful !

Translation of Standard Data Types was completely successful !!!
```

You have generated two files, one that contains the lotos annotated specification and the other is a Demon rules files that will have the same name as you input file but with the radical "**.drs**".

3. Compiling the lotos annotated specification and the Demon rules file.

value of the first offer of an action:

example:

```
u ! 222 ! offhook
```

will generate an instantiated gate “u_222”

Syntax: **instantiated_gates**

Example:

```
process user_caller[u](PN,C:number):noexit:=
  (*# region 1 instantiated_gates #*)

  u ! PN ! offhook
    ; u ! PN ! dial ! C
; u ! PN ! talk
; stop

endproc
```

1.3 Process instantiation names

The names of process instances are constructed by default using the name of the process and the first leftmost formal parameter value if any. Both the basic name and the number of parameters used can be changed by the user.

1.3.1 changing the basic process instance name

syntax: **node_name <selected name>**

example:

```
process connect_responder[u,n](PN:number):noexit:=
   (*# node_name call_responder_role
      instantiated_gates
      region 7 #*)

  n ! PN ! connect
     ; u ! PN ! talk
     ; stop

endproc
```

1.3.2 changing the number of formal parameters used

The exact number of formal parameters to be used is selected via the instance_parms graphic annotation.

Syntax: **instance_parms <number of leftmost formal parms>**

tions.

Use the keywords **default_region_definitions** or **region_definitions** followed by the Demon looking region definitions as show above.

1.2 Gates and processes region assignments

gates region assignments

Each gate declared in the high-level behavior either via the specification gate list or a hide operator in the high-level behavior expression should have a region assigment visualization annotation immediatly after the gate name. The format of the annotation is the keword region plus an integer number representing the region number.

```
specification phone_spec[  u   (*# region 4 #*) ,
                           n   (*# region 5 #*)
                           ]:noexit
        ...
behavior

        hide     g1   (*# region 6 #*) ,
                 g2   (*# region 7 #*) in
        ...
```

Each region must correspond to a region declared in the region_definitions annotation or be part of the default regions.

process region assignment

A process will correspond a node on the graph and must have a region assignment annotation. This annotation shall be place immediatly after the process definition header (after the functionality.The format of the annotation is the keword region plus an integer number representing the region number.

```
process network[n]:noexit:=
  (*# region 3 #*)

        n ? PN:number ! conreq ? C:number
        ;
        (
         (
              complete_connection[n](PN,C)
            []
              detect_busy_signal[n](PN,C)
           )
          |||
            network[n]
          )
endproc
```

1.3 Gates instance differentiating

Lotos syntax doesnot allow to intantiate gates. A way around this problem is to use the

How to visualize a Lotos specification

By Bernard Stepien
GMD-FOKUS, Berlin

This short manual will indicate you the steps you have to go through to animate a Lotos specification.

1. Lotos specification visualization annotations

The are a number of mandatory and optional information that has to be inserted into your Lotos specification to enable its visualization. This is information that can not be automatically derived from the Lotos specification.

A visualization annotation must be enclosed between the two following delimiters:

(*# ... #*)

1.1 icons representation option and region definition option annotation

These options shall be inserted immediatly after the specification header (after the functionality definitions) and before any library, datatype or behavior expressions. They will work only in that location:

```
specification phone_spec[u (*# region 4 #*) , n (*# region 5 #*) ]:noexit

        (*#     user_defined_icons
        region_definitions
        GRID R1 30 700 8 8 1.3; # Users
        GRID R2 30 450 8 8 1.3; # Phones
         GRID R3 140 100 8 8 0.8; # Network
         GRID R4 155 570 8 8 0.3; # User_gate
         GRID R5 155 245 8 8 0.3; # Network_gate
         GRID R6 30 420 8 8 1.3; # Cinit & Cresp
         GRID R7 30 380 8 8 1.3; # ConEst & ConResp
        GRID R8 30 340 8 8 1.3; # Busy
        GRID R9 180 70 8 8 1.0; # CompCon & DBusy
         GRID R10 90 750 8 8 1.3; # test_sequence
         #*)

 type number is
 sorts number
 opns 1,2,3,4:-> number
 endtype
```

You have the choice to use default icons or user supplied icons.

use the keywords **default_icons** or **user_defined_icons** as shown in the above example.

The region definitions relate to the way Demon defines regions where nodes are displayed. (see Demon reference manual for details). You have a choice to use a default definition of 10 regions that are organized into horizontal layers or to specify your own region defini-