CSI 5109 Project :
FPS 2000
Fax Plain and Simple

Patrick Bihan-Faou
student # 684327

#anuar

Abstract

The purpose of this project is to do the reverse engineering of a simple fax protocol. This protocol is explained in a set of document containing some time sequence diagrams and part of the user documentation. In a second part, I will show how I obtained the LOTOS specification, and finaly I'll try to test it.

Contents

List of Figures

Chapter 1

Analysis of the given documents

1.1 Description of the problem

The document describes the way a simple fax machine works along with communication diagrams and user interface specifications. But the information provided are incomplete and they only describe some functioning case. What we have is :

- The description of the user interface
- One page of user documentation describing how to send a fax
- Three time sequence diagrams (TSD) showing one normal case and two error case :
  - sending a fax in automatic mode
  - sending a fax in automatic mode when the destination is busy
  - sending a fax in manual mode while there is an incoming call

This insufficient to derive a full (even a simple) fax protocol specification. But still we can make some guesses starting from these information. The point that need a guess work are :

- How do we receive a fax in automatic mode ?
- How do we send a fax in manual mode ?
- How do we receive a fax in manual mode ?

Moreover the proposed TSD are incorrect because they don't take into account the telephone network. This has no consequence when two fax machine communicate together, but it is of paramount importance for the connection and the disconnection time. And this is exactly when the two error case happen.

So to be able to write the specification of a simple fax protocol, we should first eliminate the dark spaces left by the documentation. Thus we have to modify the existing TSD in order to take into account the network, create some new TSD for the case not described (cf above).

1.2 How does a fax work ?

In order to write a specification of a fax protocol, we first have to interpret the way a real fax work.

1.2.1 The functioning modes

The purpose of a fax-machine is to send or receive faxes. This particular machine (the FPS 2000) has two distinct functioning modes : either automatic or manual. This lead us to four different cases :

1. sending in automatic mode
2. receiving in automatic mode
3. sending in manual mode
4. receiving in manual mode

Case #1 and #2 can be derived from the 1^{st} TSD with little modifications concerning the connection and disconnection phases in order to take into account the network. Case #3 can partially be derived from the third TSD, but we don't have any indication on how the 4^{th} case work.

1.2.2 The entities involved in the process

Sending or receiving a fax puts several entities to work together :

- the sending person,
- the sending machine,
- the network,
- the receiving machine,
- and eventually the receiving person.

Thus the new TSD will have to show those entities.

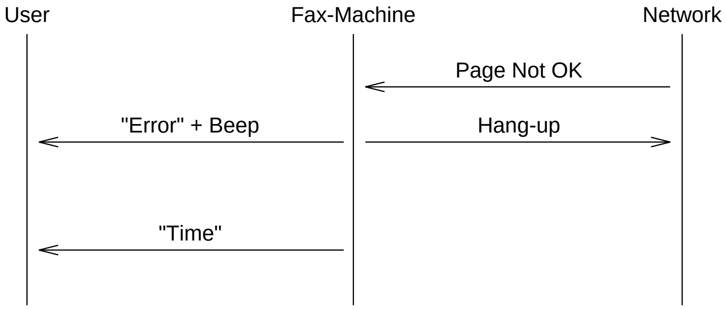

Figure 1.1: Behavior of the fax-machine in case of error

1.3 The new time sequence diagrams

1.3.1 Sending a fax in automatic mode

The original TSD has been modified in order to display more refinement in the connection and disconnection phase. Some new messages are specific to the network (such as dial-tone or ring for example), some other were introduced to make the data exchange between the two fax machines more simple and straightforward (i.e. Page 1 Begin, Page 1 Data, Page 1 End). The TSD is displayed in figure 1.2.

Error cases

We also have to determine what kind of error might happen in the sequence of events. They can occur at any time and be of any kind, but in order to keep the protocol simple, I will focus only on some frequent and likely error case :

1. the destination is busy,
2. the destination doesn't answer,
3. a page is not aknowledged (bad transmission).

In these three cases and whenever the fax encounters an unexpected situation, the fax-machine should hang-up, display an error message, and after five second display the time as shown in figure 1.1.

1.3.2 Receiving a fax in automatic mode

For this part of the protocol, we have less information. But we can easyly build a time sequence diagram for receiving a fax from the one we built in the previous section.

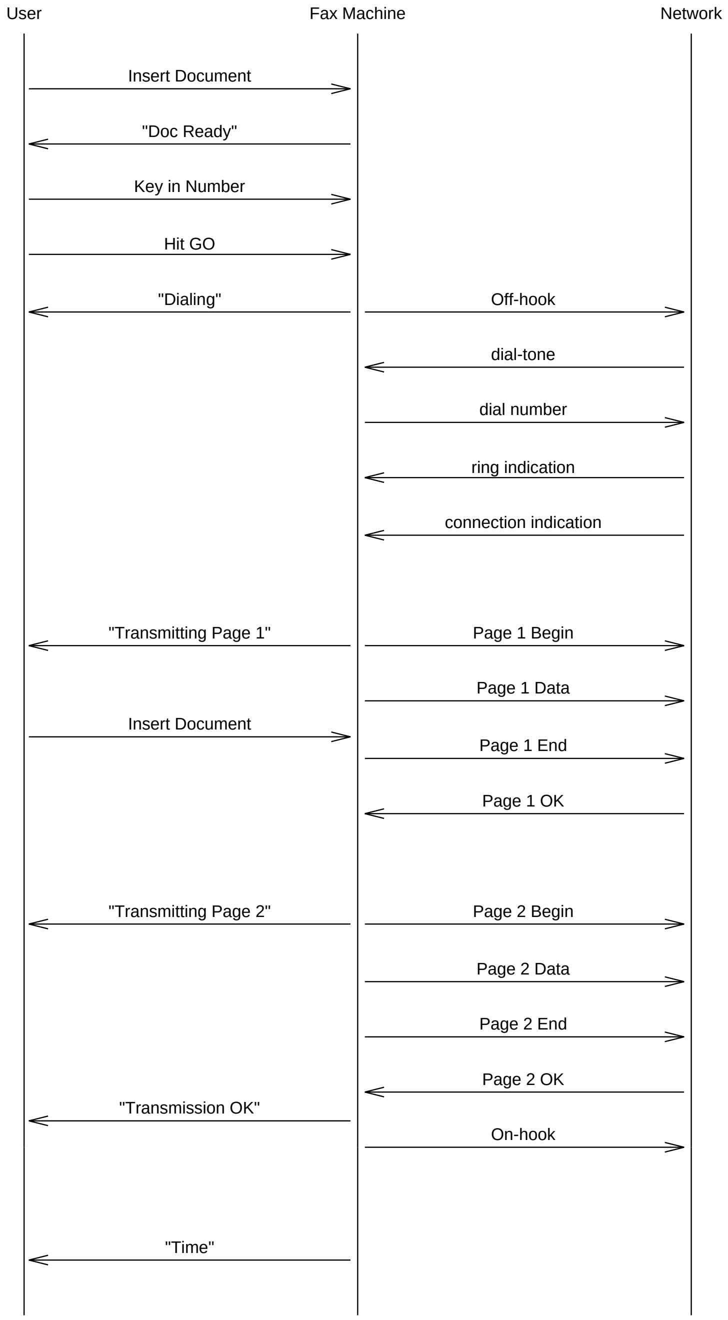

Figure 1.2: Sending a fax in automatic mode

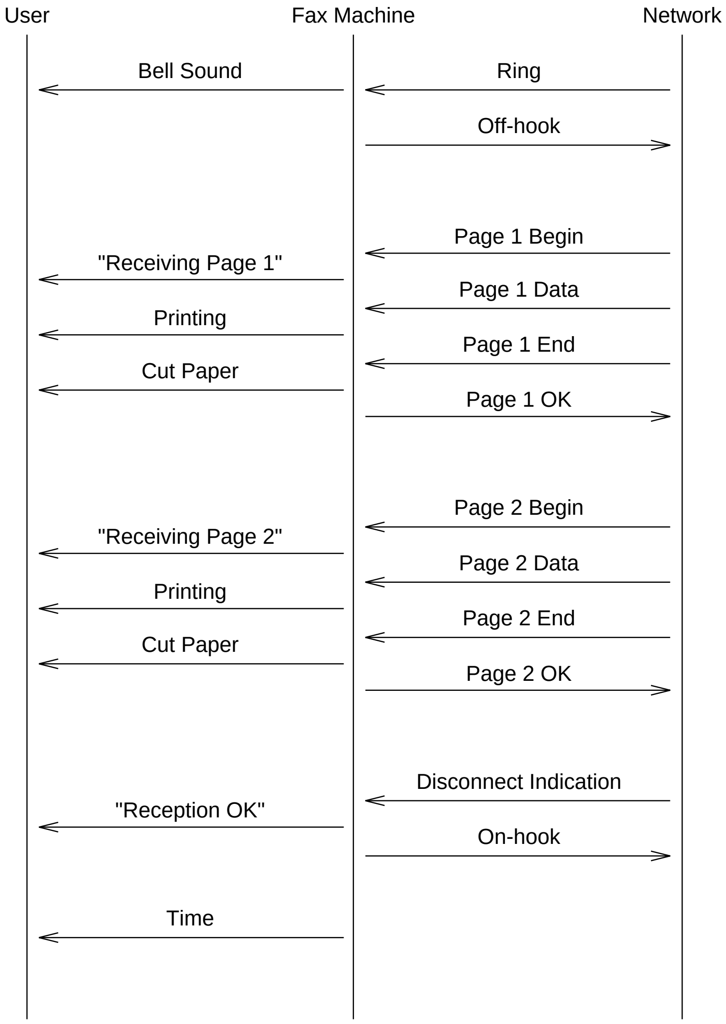

Figure 1.3: Receiving a fax in automatic mode

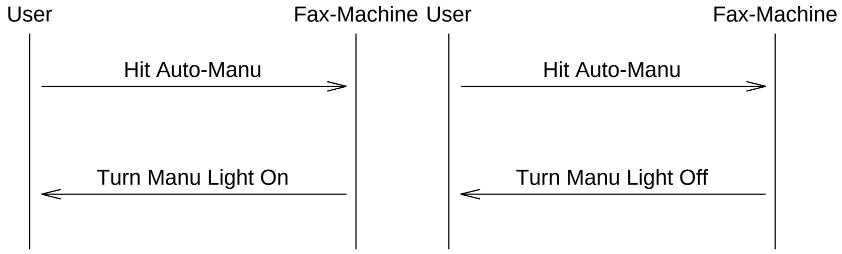

Figure 1.4: Switching to manual mode and back to automatic mode

1.3.3 The manual mode

The manual mode is not explained nor shown in the time sequence diagrams I had to work with. But the second exposed error case ("Feuding faxes") happens in this mode. Thus I have built a time sequence diagram showing how to send a fax in manual mode but because there is no way to tell how a fax works for receiving in manual mode, I haven't built anything for it.

Switching to manual mode and back to automatic mode

This mode is specific because it needs to be activated by the user. That is why we have to show how to go from one state to the other (figure 1.4).

Establishing and releasing a connection

In the manual mode, the user has to establish and release the connection himself. This is done using the telephone set on the fax machine as a normal telephone. The transmission of a document is then started whenever the user wants. In the same way, the connection is finally released by the user by hanging-up the phone.

Because these two actions are done in the same way than if the user were using a normal phone, I won't detail those procedures any further at this level of specifications.

Sending a fax in manual mode

Sending a fax can be done whenever the user wants to do it after a connection has been established. The time sequence diagram explaining how to perform this task is shown in figure 1.5.

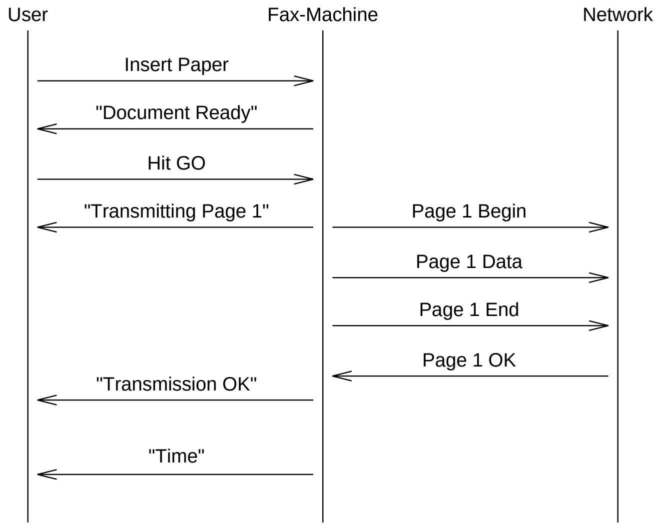

Figure 1.5: Sending a fax in manual mode

Receiving a fax in manual mode

Because there is no way in the documentations to guess how a fax would work in manual mode for receiving documents, I won't implement this feature.

Error recovery in manual mode

As for the automatic mode, transmitting documents in manual mode can be subject to errors. But because the connection must be established and released by the user, the error handling can't be exactly the same as in the automatic mode. The time sequence diagram corresponding to an arbitrary error case is shown in figure 1.6.

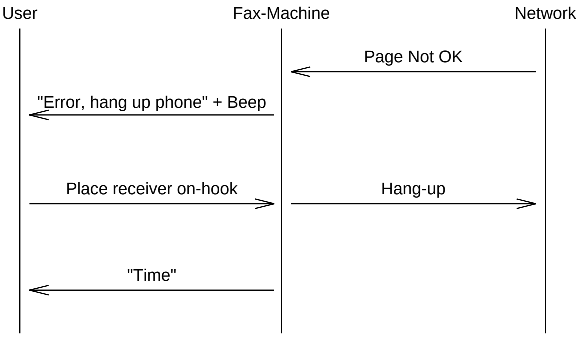

Figure 1.6: Error handling in manual mode

Chapter 2

Implementation in LOTOS

The specification of this simple fax protocol will be written in LOTOS using the constraint oriented style because of the high level of abstraction it can achieve. This implies that we must first identify the constraints at different level :

Local constraints are used to enforce the appropriate sequences of events at each fax-machine. These constraints are not the same according to the working mode of the fax machine (receiving, sending, automatic, manual).

End-to-end constraints are related to each connection. They enforce the correct sequence of actions between the two fax machine during a connection.

Global constraints are system-wide constraints. They are mainly induces by the telephone system (one can call only one other person provided that he or she is not already using the phone).

Because lots of the constraint are defined by the telephone system (the network), I will use the work done on the Plain Old Telephone System (POTS).

2.1 Identifying the constraints

2.1.1 Local constraints

The local constraints should be identified for each functioning mode of the fax machine. These are :

- sending in automatic mode,
- receiving in automatic mode,
- sending in manual mode.

The constraints can be easily identified in the time sequence diagrams. They correspond to the correct sequencing of the events in which the fax machine is involved.

An example of local constraint for the fax-machine can be : do "Off-hook" before "dial number" before "Page Begin" etc.

For the fax machine, we can represent the local constraints by two LOTOS behaviours : one for the sending machine (in either automatic or manual mode), and one for the receiving machine. The behaviours describe the sequence of actions between the fax and the user and the fax and the network.

The LOTOS behaviors representing the local constraints are *Fax_Send* and *Fax_Receive*.i Both behaviours are then subdivided in two parts: one for the manual and one for the automatic behaviour of the fax machine. For sending a fax, these behaviours are Auto_Send and *Manu_Send*. They are executed exclusively (a fax can not function in the two modes simultaneously). The choice operator is used to choose which behaviour to use. For receiving a fax, the automatic mode is the only one described yet (in *Auto_Receive*).

There is an additional behavior (*Switch_Auto_Manu*) describing the sequence of action to be taken in order to change the functioning mode of the fax-machine from automatic to manual and vice versa.

2.1.2 End-to-end constraints

These constraints are also very easily identified in the time sequence diagrams as being the correct ordering of the all set of events between two symmetrical entities (i.e. the receiving and the sending fax machine). These constraints correspond to the correct sequencing of event at the level of the network.

Considering the interactions between the fax machine and the network, these constraints are the same whatever the functioning mode of the fax machine (either in automatic or manual mode).

An example of end-to-end constraints between the two fax-machine is : "[receiver]off_hook" before "[sender]connection_ind" (the receiving fax-machine must go off-hook before the sending fax-machine receives a connection indication).

The LOTOS behavior representing the end-to-end constraints is *Network*. This behavior is unique, because it must make the link between the sending and the receiving fax.

2.1.3 Global constraints

The global constraints are related for one part to the use of the network, and in the other part to the fax machines.

For the network, the constraint is that when a phone (or fax) number is used (i.e. a fax is already interacting with the network), it can not take part in another interaction. The behaviour keeping track of the use of the phone numbers and insuring the respect of these constraints is *Users_List_Handler*. This behaviour is directly derived from the POTS specification, only minor modifications have been done on it so that it can accommodate with the dialogs between the fax machines.

For the fax machine, the constraint is that a first a fax machine must work according to the mode that has been selected (manual or automatic) and second that it cannot be used for two actions at the same time. The behaviour describing those constraints is *Fax_Mode_Handler*.

2.2 Integrating the different constraints

As said previously, the LOTOS specification of the fax protocol is based on the POTS specification. Behaviors represent the constraints and are interleaved or synchronized together depending on the kind of constraints they express and the function the have.

2.2.1 Integrating the local and end-to-end constraints together

To do that, we must define how each behaviour works. We can identify to different use of the fax machine :

- communicating with another fax (receiving or sending documents),
- or switching between manual and automatic mode.

The first case involves two fax machine : one in receiving mode, and one in sending mode, with the communication between them going through the network. Thus this action is composed of the behaviours *Fax_Receive* and *Fax_Send* (representing the local constraints at each fax) interleaved together and synchronized with *Network* representing the end-to-end constraints. The second case involves only one fax in a "local" mode. Thus the only constraints are local.

These two cases can be grouped in a single behaviour that I call *Actions*. Because, each expression of an action (i.e. communication or switching mode) sets the constraints for a single use of the fax machine, the behaviour *Actions* must be able to create more than one instance of a fax use at the same time. Thus the behaviour is interleaved with itself.

2.2.2 Adding the global constraints

Then the behaviour describing the actions must be interleaved with the global constraints. Because some of the global constraints for the network occur on the same actions than some of the global constraints for the fax-machines, the two behaviour representing the global constraints have to synchronize together for the system work correctly.

2.2.3 The resulting LOTOS specification

In order to keep the specification easy to read, I have used different gates name for different kind of actions. Thus, the constraints' behaviours don't have to synchronize on all the gates, but only on a meaningful subset of them.

The first version of the specification doesn't do any error checking and recovery at all. Here is a global overview of the specification of te fax system in LOTOS .

```
Specification FPS_System [User_Actions, Fax_Actions, Network_Actions]:  noexit

Behaviour
(
   Actions[User_Actions, Fax_Actions, Network_Actions]
   |[User_Actions, Network_Actions]|
   Global_Constraints[User_Actions, Network_Actions](...)
)
Where

   Process Actions[User_Actions, Fax_Actions, Network_Actions]:noexit:=
   (
      (
         Single_Fax_Connection[User_Actions, Fax_Actions, Network_Actions]
         []
         Single_Mode_Switch[User_Actions]
      )
      |||
      i; Actions[User_Actions, Fax_Actions, Network_Actions]
   )
   Where

      Process Single_Mode_Switch[u_action]:noexit:=
      (
         u_action ? me: DecDigit ! switch_mode;
         stop
      )
      EndProc (* Single_Switch_Mode *)

      Process Single_Fax_Connection[u_action, f_action, n_action]:noexit:=
      (
         (     (**** The local constraints ****)
            Fax_Send[u_action, f_action, n_action]
            |||
```

```
         Fax_Receive[u_action, f_action, n_action]
      )
      |[n_action]|
      Network[n_action]    (**** The end-to-end constraints ****)
   )
   Where
      ...
   EndProc (* Single_Fax_Connection *)

EndProc (* Actions *)

Process Global_Constraints[u_actions, n_actions](...):noexit:=
(
   Users_List_Handler[n_actions](...)
   |[n_actions]|
   Fax_Mode_Handler[u_actions, n_actions](...)
)
Where
   ...
EndProc (* Global_Constraints *)

EndSpec
```

2.3 Handling the error cases

In the specification, we can deal with different type of error handling :

- Detecting local errors (local in a temporal way) which are identified and known.
- Detecting and recovering from general errors : errors that can happen at any time of the connection and that are not necessarily known.

2.3.1 Local error cases

Dealing with local errors is not very difficult. It is done by offering the alternative (the choice operator) at a certain point between the normal sequence of actions and the identified error case(s). In the first version of the FPS2000 specification, I treat two such error cases :

- When trying to establish a connection, a fax might not get the dial-tone from the network (*no_dial_tone* error message) if another phone set is currently holding the line.
- When trying to establish a connection, the called number can be busy (*busy_destination* error message).

2.3.2 General or global error cases

Recovering from general errors is much more difficult. This is due to the fact that a certain error doesn't have the same consequence depending on when it occurs. For example, if the connection is closed between the transfer of two pages (after the end of the first page and after the beginning of the second one), this does not mean that the transfer is not successful for the receiving fax (it has no way to know how many pages should have been received). But if this happens before having received completely a page, this is an error case. Still in this case we are lucky, because we can detect that the connection has been lost when we receive the appropriate message from the network.

But whenever we can't find such messages, the errors are much more difficult to detect. This kind of error happens when some data are lost by the network. Then the only way to know that an error happened lies in the fact that we don't receive the message(s) we were looking for.

In the current version of the specification, the system reacts poorly to such events, but still tries to handle them. The major cause of the problems I am facing there is the result of an incorrect use of the interrupt operator at some places of the specification (especially in the process *Auto_Receive* where the fax still agrees to produce an error message after a successful transfer has been terminated.)

2.4 Modifications made to the POTS

In order to make the fax protocol specification work, I have used some parts of the POTS specification. Mainly the part describing the global constraints (the *Users_List_Handler*). But then I have found some problems with this specification. So I had to modify it in two ways: make it accept the messages I have introduced with the fax protocol, and modify slightly some constraints.

2.4.1 The new messages

A modification I have done to the system is the fact that the carrying of message through the network is now decomposed in two parts: one of the party says something and te other side listens to it. This allows me to describe more precisely the local constraints at each fax machine. In the original POTS specification, there was no need to show one of the user listening to the other because the interest of the specification was to show how the connections are made. Thus we have two new messages: *say* and *listen*. Each of these messages can have one or two following "words". This is due to the need to tell the number of the page we are transmitting.

Finally, I have added a message indicating to one of the party that the other party has released the connection (he has hung-up his phone). This message is known as *no_connection_signal*. It can also occur after the *dial-tone* message if the user has not dialed the destination number before the end of a time-out. Yet this option is not implemented in the actual version of the specification.

2.4.2 The problems and their modifications

One of the major problem is in the way a connection is initiated: whenever an action “off-hook” is done for a phone number, no other “off-hook” action can be performed for that phone number. This is due to the way the POTS specification remembers which are the phone number being used. A more desirable property would be that when we try to perform the second off-hook, we can do it, but we don’t get any dial tone. Then the connection should not be released until all telephone set involved are hung-up. This means that our user can change the telephone set he/she is using during a conversation.

Modifying the process *Users_List_Handler* in order to produce this property implies some major modifications in the way the systems keeps track of the established connections. These modifications are being done but are not yet finished.

Chapter 3

Testing the specification

Once the specification is written, it is interesting to test it so we can find and point out some problems with it (provided that the problems were not introduced in the LOTOS expression of the specification of course !!).

In this particular case, we should first point out the two error cases described in the original documents. As I built this specification using them, the behaviour shown by the execution of the LOTOS specification should be "equivalent" if not equal to the one described in the documents.

Another way to test the specification is to use the tool LOTOS *Model Checker* (*LMC*). This tools allows us to describe some temporal properties that we think the specification should have, and test them against the specification.

3.1 Checking the error case shown in the documents

The documents describe two different error cases (in addition to a normal case). In this section, I'll try to reproduce each of the traces shown in the source document.

To build those use cases, I will use the tool ISLA which allows me to go through a LOTOS specification step by step.

3.1.1 Normal case : sending one page in automatic mode

This a normal, basic utilization of the fax machine : the user wants to send a one page long document to a remote fax-machine using the automatic mode. The trace obtained with ISLA is :

```
(* Tree generated by Isla *)
```

```
(* modified in order to be clearer *)

(* DD stands for type DecDigit *)
(* FA stands for type Fax_Action *)
(* UA stands for type User_Action *)
(* NA stands for type Network_Action *)
(* PC stands for type Page_Counter *)
(* FD stands for type Fax_Display *)
(* FM stands for type Fax_Message *)


(* Actions happening at the sending fax start at the first column *)
(* Actions happening at the receiving fax start at the fifth column *)

FPS_System[User_Actions, Fax_Actions, Network_Actions]()

(* first we establish the connection *)

User_Actions ? me,me = 0:DD, ! insert_doc:UA, ! false:Bool, [not(me IsIn {})] ;
             at line(s) 234,564
Fax_Actions !0:DD !display:FA !doc_ready:FD; at line(s) 257
User_Actions ! 0:DD, ! key_in:UA, ? number,n = 1:DD; at line(s) 258,567
User_Actions !0:DD !hit_go:UA; at line(s) 259,570
Fax_Actions !0:DD !display:FA !dialing:FD; at line(s) 260
Network_Actions !0:DD !off_hook:NA [true]; at line(s) 261,409,490,587
Network_Actions !0:DD !dial_tone:NA; at line(s) 265,410,494,593
Network_Actions !0:DD !dial:NA !1:DD; at line(s) 266,411,497,596
      Network_Actions !1:DD !ring:NA !false:Bool [true and true];
                     at line(s) 347,414,501,584
      Fax_Actions !1:DD !bell_sound:FA; at line(s) 360
      Network_Actions !1:DD !off_hook:NA [true]; at line(s) 361,415,490,587
      Network_Actions !1:DD !say:NA !i_am_here:FM;
                      at line(s) 362,443,529,608
Network_Actions !0:DD !listen:NA !i_am_here:FM;
                at line(s) 271,444,532,611

(* the connection phase is finished, we can start sending a page *)

Network_Actions !0:DD !say:NA !page_begin:FM !add(none):PC;
                at line(s) 293,431,523,602
Fax_Actions !0:DD !display:FA !transmitting_page:FD !add(none):PC;
             at line(s) 294
      Network_Actions !1:DD !listen:NA !page_begin:FM !add(none):PC;
                      at line(s) 371,432,526,605
      Fax_Actions !1:DD !display:FA !receiving_page:FD !add(none):PC;
                  at line(s) 372
Network_Actions !0:DD !say:NA !page_data:FM !add(none):PC;
```

```
                at line(s) 295,431,523,602
      Network_Actions !1:DD !listen:NA !page_data:FM !add(none):PC;
                      at line(s) 373,432,526,605
      Fax_Actions !1:DD !printing:FA; at line(s) 374
Network_Actions !0:DD !say:NA !page_end:FM !add(none):PC;
                at line(s) 296,431,523,602
      Network_Actions !1:DD !listen:NA !page_end:FM !add(none):PC;
                      at line(s) 375,432,526,605
      Fax_Actions !1:DD !cut_paper:FA; at line(s) 376
      Network_Actions !1:DD !say:NA !page_OK:FM !add(none):PC;
                      at line(s) 377,435,523,602
      hidden internal_sync !1:DD;
Network_Actions !0:DD !listen:NA !page_OK:FM !add(none):PC;
                at line(s) 297,436,526,605
hidden internal_sync !0:DD;

(* now the transmission is finished, we can close the connection *)

Fax_Actions !0:DD !display:FA !transmission_OK:FD; at line(s) 320
Network_Actions !0:DD !hang_up:NA; at line(s) 321,452,506,590
      Network_Actions !1:DD !no_connection_signal:NA;
                      at line(s) 388,453,535,614
      Network_Actions !1:DD !hang_up:NA; at line(s) 389,454,506,590
Fax_Actions !1:DD !display:FA !reception_OK:FD; at line(s) 390
hidden wait_5_sec !0:DD;
Fax_Actions !0:DD !display:FA !time:FD; at line(s) 323
      hidden wait_5_sec !1:DD;
      Fax_Actions !1:DD !display:FA !time:FD; at line(s) 392
```

The trace shows that the fax-machine on both side have a behaviour consistent with the one described in the original documents.

3.1.2 Error case : destination busy

This is a frequent error case. It happens whenever the fax number the user tries to reach is already being used (either receiving or sending a fax). The error is detected and reported to the fax-machine by the network. Then the fax-machine displays an error message and hangs-up the line. The trace obtained with ISLA is :

```
(* Tree generated by Isla *)
(* modified in order to be clearer *)

(* DD stands for type DecDigit *)
(* FA stands for type Fax_Action *)
```

```
(* UA stands for type User_Action *)
(* NA stands for type Network_Action *)
(* FD stands for type Fax_Display *)

(* Actions happening at the sending fax start at the first column *)
(* Actions happening at the receiving fax start at the fifth column *)
(* Actions happening at the third fax machine start at the first column *)

FPS_System[User_Actions, Fax_Actions, Network_Actions]()

(* first we establish a connection between to fax machines *)

User_Actions ? me,me = 0:DD, ! insert_doc:UA, ! false:Bool, [not(me IsIn {})] ;
             at line(s) 234,564
Fax_Actions !0:DD !display:FA !doc_ready:FD; at line(s) 257
User_Actions ! 0:DD, ! key_in:UA, ? number,n = 1:DD; at line(s) 258,567
User_Actions !0:DD !hit_go:UA; at line(s) 259,570
Fax_Actions !0:DD !display:FA !dialing:FD; at line(s) 260
Network_Actions !0:DD !off_hook:NA [true]; at line(s) 261,409,490,587
Network_Actions !0:DD !dial_tone:NA; at line(s) 265,410,494,593
Network_Actions !0:DD !dial:NA !1:DD; at line(s) 266,411,497,596
      Network_Actions !1:DD !ring:NA !false:Bool [true and true];
                      at line(s) 347,414,501,584
      Fax_Actions !1:DD !bell_sound:FA; at line(s) 360
      Network_Actions !1:DD !off_hook:NA [true]; at line(s) 361,415,490,587

(* now there is a connection established *)

i (specified explicitly) (* to start another action *)

(* the user #2 tries to send a fax to user #1 *)

User_Actions ? me,me = 2:DD, ! insert_doc:UA, ! false:Bool, [not(me IsIn {})] ;
 at line(s) 234,564
Fax_Actions !2:DD !display:FA !doc_ready:FD; at line(s) 257
User_Actions ! 2:DD, ! key_in:UA, ? number,n = 1:DD; at line(s) 258,567
User_Actions !2:DD !hit_go:UA; at line(s) 259,570

Fax_Actions !2:DD !display:FA !dialing:FD; at line(s) 260
Network_Actions !2:DD !off_hook:NA [true]; at line(s) 261,409,490,587
Network_Actions !2:DD !dial_tone:NA; at line(s) 265,410,494,593
Network_Actions !2:DD !dial:NA !1:DD; at line(s) 266,411,497,596

(* the network detects that the user #1 is already on line *)

Network_Actions !2:DD !busy_signal:NA [true]; at line(s) 268,420,517,599
```

```
(* the fax machine handles this error case *)

Fax_Actions !2:DD !display:FA !dest_busy:ErrorType; at line(s) 219
Network_Actions !2:DD !hang_up:NA; at line(s) 220,421,506,590
hidden wait_5_sec !2:DD;
Fax_Actions !2:DD !display:FA !time:FD; at line(s) 222

(* the connection is closed after displaying a error message *)
```

Again, in this case, we have a behaviour corresponding to the one described in the original document. The network detects that the destination number is busy, reports this problem to the fax machine which handles it.

3.1.3 Error case : feuding fax

This error case involves trying to send a fax in manual mode, while somebody tries to send you a fax. Because the process of sending fax manually is not yet described in the LOTOS specification, this error case could not be tested.

3.2 Testing using LMC

LMC is a tool to check some properties on a LOTOS specification. These properties are expressed as temporal logic functions such as:

> If this is true at instant t, then this must be true for at least one of the future.

This kind of tools can help us check if a specification has some desirable properties or not, and eventually find some problem.

3.2.1 Identifying some properties

No mode mix

One of the properties we might want to check is that when a fax-machine is in manual (resp. automatic) mode, it can not send a fax in automatic (resp. manual) mode.

Using LMC, this property is codded as :

```
ag(Switch ! @1 ! manual --> a(~(send ! @1 ! automatic) until Switch ! @1 ! automatic))
```

No deadlock

This can have several meanings:

- The global system should never deadlock

- A given fax machine should not remain off-hook forever

The first case can be written in LMC in the following way :

```
ag(~(deadlock))
```

The second case is more like:

```
ag(n\_action ! @1 ! off\_hook --> af(n\_action ! @1 ! hang\_up))
```

The problem with this property is that there is no limitation on the number of page a fax-machine can send or receive, and therefor we can be led in a infinite loop. And as LMC does not support fairness this will lead to an inconclusive result.

3.2.2 Testing the properties with LMC

I had no time to check those properties yet, so I can't reproduce any results here.

Appendix A

What to do next...

The specification is not complete yet. There are many thing that need to be finished or corrected. Here is a first list of things to do :

- Write the process *Manu_Send* according to the time sequence diagrams shown in section 1.3.3.
- Add a manual mode for receiving fax (if relevant).
- Make the suggested corrections to the process *Users_List_Handler.* (see section 2.4)
- Add "normal" phone capabilities to the specification.
- Simplify the process of sending fax (for the moment we have three messages *Page_Begin, Page_Data* and *Page_End*, and only one should be enough.)
- Improve the error recovery mechanisms. The actual version performs poorly in the general error cases described in section 2.3.2.
- Do more testing with LMC or any other method in order to find the problems in this specification.

This list is of course not exhaustive, but recall the main points.

Appendix B

The FPS2000 LOTOS specification

```
(**************************************************************)
(*  Fax Plain and Simple (FPS 2000) Specification in LOTOS      *)
(*  by Patrick Bihan-Faou,                                      *)
(*  January 1995                                                *)
(**************************************************************)
(*  This specification is based on the POTS specification       *)
(*  (Plain Old Telephone System) written by Luigi Logrippo,     *)
(*  Mohamed Faci and Bernard Stepien.                           *)
(**************************************************************)

Specification FPS_System [User_Actions, Fax_Actions, Network_Actions]:  noexit

library Set, Boolean, DecDigit, NaturalNumber endlib

type DigitPair is DecDigit, Boolean
   sorts DigitPair
   opns
       None: -> DecDigit
       None: -> Nat
       Pair: DecDigit,DecDigit -> DigitPair
       _eq_: DigitPair, DigitPair -> Bool

   eqns
       forall f1,f2,l1,l2:DecDigit, f3:Nat

       ofsort Bool
          f1 eq None     = false;
          None     eq f1 = false;
          Pair(f1,l1) eq Pair(f2,l2) = (f1 eq f2) and (l1 eq l2);
```

```
       ofsort Bool
          f3 eq None     = false;
          None     eq f3 = false;

       ofsort Nat
          NatNum(None) = None
endtype

type Set_Of_Pairs  is DigitPair
     sorts Set_Of_Pairs
     opns
         empty: -> Set_Of_Pairs
         add:   DigitPair,Set_Of_Pairs  -> Set_Of_Pairs
         remove:   DigitPair,Set_Of_Pairs  -> Set_Of_Pairs
         isin: DigitPair,Set_Of_Pairs  -> Bool
         Second_Element:DecDigit,Set_Of_Pairs  -> DecDigit
         notin: DigitPair,Set_Of_Pairs -> Bool

     eqns
         forall x,y:DigitPair, s:Set_Of_Pairs, f1,f2,l1,l2:DecDigit

         ofsort Set_Of_Pairs
            add(x,add(x,s)) = add(x,s);
            remove(x,add(x,s)) = s;
            not(x eq y) =>
              remove(x,add(y,s)) = add(y,remove(x,s));

         ofsort Bool
            isin(x,empty) = false;
            isin(x,add(y,s)) = (x eq y) or (isin(x,s));
            isin(x,remove(x,s)) = false;
            not(x eq y) =>
                  isin(x,remove(y,s)) = isin(x,s);
            notin(x,s) = not(isin(x,s));

         ofsort DecDigit
            Second_Element (f1,empty) = None   ;
            Second_Element (f1,add(Pair(f1,l1),s)) = l1;
            f1 ne f2 => Second_Element (f1,add(Pair(f2,l2),s)) = Second_Element (f1,s);
endtype
```

```
type DecSet is Set
   actualizedby DecDigit
   using
   sortnames
      DecDigit      for Element
      Bool         for FBool
      DecSet      for Set
endtype

(*********************************************************)
(* These are the messages that a fax machine can display *)
(*********************************************************)
type network_action is
   sorts Network_Action
   opns
      off_hook,
      hang_up,
      ring,
      say,
      listen,
      dial_tone,
      dial,
      no_connection_signal,
      busy_signal   :-> Network_Action
endtype

type fax_display is
   sorts Fax_Display
   opns
      doc_ready,
      dialing,
      transmitting_page,
      transmission_OK,
      receiving_page,
      reception_OK,
      time         :-> Fax_Display
endtype

type ErrorType is Boolean
   sorts ErrorType
   opns
      no_dial_tone,
      dest_busy,
      lost_connection,
      tx_not_OK,
      rx_not_OK   :-> ErrorType
endtype
```

```
type fax_action is
   sorts Fax_Action
   opns
      display,
      printing,
      cut_paper,
      manu_light_on,
      manu_light_off,
      bell_sound   :-> Fax_Action
endtype

type user_action is
   sorts User_Action
   opns
      insert_doc,
      key_in,
      switch_mode,
      hit_go      :-> User_Action
endtype

type fax_message is
   sorts Fax_Message
   opns
      user_speech,
      i_am_here,
      page_begin,
      page_data,
      page_end,
      page_OK      :-> Fax_Message
EndType

type page_counter is
   sorts Page_Counter
   opns
      none :-> Page_Counter
      add : Page_Counter -> Page_Counter
endtype
```

```
(*******************************************************)
(* Description of the behaviour of the complete system   *)
(*******************************************************)
behaviour
(
   Actions[User_Actions, Fax_Actions, Network_Actions]
   |[User_Actions, Network_Actions]|
   Global_Constraints[User_Actions, Network_Actions](empty, {} of DecSet, {} of DecSet)
)
where

(*****************************************************************************)

   Process Actions[User_Actions, Fax_Actions, Network_Actions]:noexit:=
   (
      (
         Single_Fax_Connection[User_Actions, Fax_Actions, Network_Actions]
         []
         Single_Mode_Switch[User_Actions]
      )
      |||
      i; Actions[User_Actions, Fax_Actions, Network_Actions]
   )
   Where

   (**************************************************************************)

      Process Single_Mode_Switch[u_action]:noexit:=
      (
         u_action ? me: DecDigit ! switch_mode;
         stop
      )
      EndProc (* Single_Switch_Mode *)

   (**************************************************************************)

      Process Single_Fax_Connection[u_action, f_action, n_action]:noexit:=
      (
         (      (**** The local constraints ****)
            Fax_Send[u_action, f_action, n_action]
            |||
            Fax_Receive[u_action, f_action, n_action]
         )
         |[n_action]|
         Network[n_action]      (**** The end-to-end constraints ****)
      )
      Where
```

```
   Process Fax_Error_Recover[f_action, n_action]
              (me: DecDigit, etype: ErrorType):noexit:=
   hide wait_5_sec in
   (
      f_action ! me ! display ! etype;
      n_action ! me ! hang_up;
      wait_5_sec ! me;
      f_action ! me ! display ! time;
      stop
      []
      stop      (* to exit when we don't have any errors *)
      (* this is to be corrected by a better error-handling *)
   )
   EndProc (* Fax_Error_Recover *)

(************************************************************************)

   Process Fax_Send[u_action, f_action, n_action]:noexit:=
   (
      u_action ? me:   DecDigit ! insert_doc ? manu: Bool;
      (
         [ manu ] -> Manu_Send[u_action, f_action, n_action](me)
         []
         [ not(manu) ] -> Auto_Send[u_action, f_action, n_action](me)
      )
   )
   Where


      Process Manu_Send[u_action, f_action, n_action]
                 (me: DecDigit):noexit:=
      (
         Auto_Send[u_action, f_action, n_action](me)
      )
      EndProc (* Manu_Send *)

   (*********************************************************************)

      Process Auto_Send[u_action,f_action,n_action]
                 (me: DecDigit):noexit:=
      (
         f_action ! me ! display ! doc_ready;
         u_action ! me ! key_in ? number : DecDigit;
         u_action ! me ! hit_go;
         f_action ! me ! display ! dialing;
         n_action ! me ! off_hook;
```

```
   (
      Fax_Error_Recover[f_action, n_action](me, no_dial_tone)
      []
      n_action ! me ! dial_tone;
      n_action ! me ! dial ! number;
      (
         n_action ! me ! busy_signal;
         Fax_Error_Recover[f_action, n_action](me, dest_busy)
         []
         n_action ! me ! listen ! i_am_here;
         Send_Page[u_action, f_action, n_action](me, add(none))
      [> n_action ! me ! no_connection_signal;
         Fax_Error_Recover[f_action, n_action](me, lost_connection)
      )
   )
)
Where

   Process Send_Page[u_action, f_action, n_action]
              (me: DecDigit, counter : Page_Counter):noexit:=
   hide internal_sync in
   (
      Send_This_Page[f_action, n_action, internal_sync](me, counter)
      |[internal_sync]| (**)
      Next_Page_Or_Leave[u_action, f_action, n_action, internal_sync](me, counter)
   )
   Where

         Process Send_This_Page[f_action, n_action, internal_sync]
                    (me: DecDigit, counter: Page_Counter):noexit:=
         (
            n_action ! me ! say ! page_begin ! counter;
            f_action ! me ! display ! transmitting_page ! counter;
            n_action ! me ! say ! page_data ! counter;
            n_action ! me ! say ! page_end ! counter;
            n_action ! me ! listen ! page_OK ! counter;
            internal_sync ! me;
            stop
            |[internal_sync]|
            (
               Fax_Error_Recover[f_action, n_action](me, tx_not_OK)
               []
               internal_sync ! me;
               stop
            )
         )
         EndProc (* Send_This_Page *)
```

```
               Process Next_Page_Or_Leave[u_action, f_action, n_action, internal_sync]
                          (me: DecDigit, counter: Page_Counter):noexit:=
               hide wait_5_sec in
               (
                  (   u_action ! me ! insert_doc;
                     internal_sync ! me;
                     Send_Page[u_action, f_action, n_action](me, add(counter))
                  )
                  []
                  (    internal_sync ! me;
                     f_action ! me ! display ! transmission_OK;
                     n_action ! me ! hang_up;
                     wait_5_sec ! me;
                     f_action ! me ! display ! time;
                     stop
                  )
               )
               EndProc (* Next_Page_Or_Leave *)
      (* the internal_sync gate is there to make sure that the fax *)
      (* start to send a new page before the previous one has been *)
      (* finished *)
      (* Thus the user must add some paper to transmit *)
      (* before the acknowledgment message is received from the *)
      (* other fax machine...*)

         EndProc (* Send_Page *)

      EndProc (* Auto_Send *)

   EndProc (* Fax_Send *)

(************************************************************************)

   Process Fax_Receive[u_action, f_action, n_action]:noexit:=
   (
      n_action ? me: DecDigit ! ring ? manu: Bool;
      (
         [ manu ] -> Auto_Receive[f_action, n_action](me)
         []
         [ not(manu) ] -> Auto_Receive[f_action, n_action](me)
      )
   )
   Where
```

```
Process Auto_Receive[f_action, n_action]
          (me: DecDigit):noexit:=
(
   (***** Connection Phase *****)
   f_action ! me ! bell_sound;
   n_action ! me ! off_hook;
   n_action ! me ! say ! i_am_here;
   Receive_Page[f_action, n_action](me)
)
Where

   Process Receive_Page[f_action, n_action]
             (me: DecDigit):noexit:=
   hide internal_sync, wait_5_sec in
   (
      n_action ! me ! listen ! page_begin ? counter : Page_Counter;
      f_action ! me ! display ! receiving_page ! counter;
      n_action ! me ! listen ! page_data ! counter;
      f_action ! me ! printing;
      n_action ! me ! listen ! page_end ! counter;
      f_action ! me ! cut_paper;
      n_action ! me ! say ! page_OK ! counter;
      internal_sync ! me;
      stop
      |[internal_sync]|
      (
         Fax_Error_Recover[f_action, n_action](me, rx_not_OK)
         []
         internal_sync ! me;
         (
            Receive_Page[f_action, n_action](me)
            []
            n_action ! me ! no_connection_signal;
            n_action ! me ! hang_up;
            f_action ! me ! display ! reception_OK;
            wait_5_sec ! me;
            f_action ! me ! display ! time;
            stop
         )
      )
   )
   EndProc (* Receive_Page *)

EndProc (* Auto_receive *)

EndProc (* Fax_Receive *)
```

```
(*********************************************************************)

   Process Network[n_action]:noexit:=
   (
      (***** Connection Phase *****)

      n_action ? caller : DecDigit ! off_hook;
      n_action ! caller ! dial_tone;
      n_action ! caller ! dial ? called : DecDigit;
      (
         (
            n_action ! called ! ring ? mode: Bool;
            n_action ! called ! off_hook;
            Discussion[n_action](caller,called)
         [> Hang_Up[n_action](caller, called)
         )
         []
         n_action ! caller ! busy_signal;
         n_action ! caller ! hang_up;
         stop
      )
      (* must still do the hang_up *)
   )
   Where

      Process Discussion[n_action]
                 (caller, called : DecDigit):noexit:=
      (
         n_action ! caller ! say ? what1: Fax_Message ? what2: Page_Counter;
         n_action ! called ! listen ! what1 ! what2;
         Discussion[n_action](caller, called)
         []
         n_action ! called ! say ? what1: Fax_Message ? what2: Page_Counter;
         n_action ! caller ! listen ! what1 ! what2;
         Discussion[n_action](caller, called)
         []
         n_action ! caller ! say ? what1: Fax_Message;
         n_action ! called ! listen ! what1;
         Discussion[n_action](caller, called)
         []
         n_action ! called ! say ? what1: Fax_Message;
         n_action ! caller ! listen ! what1;
         Discussion[n_action](caller, called)
      )
      EndProc (* Discussion *)
```

```
         Process Hang_Up[n_action]
                    (caller, called: DecDigit):noexit:=
         (
            n_action ! caller ! hang_up;
            n_action ! called ! no_connection_signal;
            n_action ! called ! hang_up;
            stop
            []
            n_action ! called ! hang_up;
            n_action ! caller ! no_connection_signal;
            n_action ! caller ! hang_up;
            stop
         )
         EndProc (* Hang_Up *)

      EndProc (* Network *)

   (**********************************************************************)

   EndProc (* Single_Fax_Connection *)

  (***********************************************************************)

  EndProc (* Actions *)

(************************************************************************)

  Process Global_Constraints[u_actions, n_actions]
           (E: Set_Of_Pairs, B: DecSet, M: DecSet):noexit:=
  (
     Users_List_Handler[n_actions](E,B)
     |[n_actions]|
     Fax_Mode_Handler[u_actions, n_actions](M)
  )
  Where

  (***********************************************************************)

     process Users_List_Handler [S_User]
               (E : Set_Of_Pairs, B: DecSet ) :noexit:=
     (
        S_User ? User: DecDigit ! off_hook
        [notin (  Pair(User, Second_Element (User, E)), E)];
        Users_List_Handler [S_User]( add( Pair(User, None ), E), B)
        []
        S_User ? Caller: DecDigit ! dial_tone;
        Users_List_Handler [S_User](E, B)
```

```
   []
   S_User ? Caller: DecDigit ! dial ? Called: DecDigit ;
   Users_List_Handler [S_User]
      (add( Pair( Caller, Called), remove( Pair(Caller, None), E)), B)
   []
   S_User ? Called: DecDigit ! ring ? mode: Bool
      [not(isin( Pair(Called, Second_Element (Called, E)), E)
      or  ( Called IsIn B))];
   Users_List_Handler [S_User](E, Insert(Called, B))
   []
   S_User ? User: DecDigit ! hang_up;
   (  [(User IsIn B)] ->
         Users_List_Handler [S_User]
            (remove(Pair (User, None),E), Remove (User, B))
      []
      [ (User NotIn B)] ->
         Users_List_Handler [S_User]
            (remove(Pair (User,Second_Element (User, E)),E),
            Remove(Second_Element (User, E), B))
   )
   []
   S_User ? Caller: DecDigit ! busy_signal
      [isin( Pair(Second_Element (Caller, E), Second_Element
      (Second_Element (Caller, E), E)), E)
      or ( Second_Element (Caller, E) IsIn B) ];
   Users_List_Handler [S_User](E, B)
   []
   S_User ? User: DecDigit ! say ? what1: Fax_Message ? what2: Page_Counter;
   Users_List_Handler [S_User](E, B)
   []
   S_User ? User: DecDigit ! listen ? what1: Fax_Message ? what2: Page_Counter;
   Users_List_Handler [S_User](E, B)
   []
   S_User ? User: DecDigit ! say ? what1: Fax_Message;
   Users_List_Handler [S_User](E, B)
   []
   S_User ? User: DecDigit ! listen ? what1: Fax_Message;
   Users_List_Handler [S_User](E, B)
   []
   S_User ? User: DecDigit ! no_connection_signal;
   Users_List_Handler [S_User](E, B)
)
EndProc (* Users_List_Handler *)
```

```
(****************************************************************************)

   Process Fax_Mode_Handler[u_action, n_action]
             (M: DecSet):noexit:=

   (* M is the set of fax-machines that are in manual mode *)
   (* each fax machine is identified by the phone number of the line *)
   (* on which it is connected *)

   (

   (* first we describe th global constraint related to the fax-machine *)
   (* when the user performs an action on it *)

      u_action ? me:DecDigit ! switch_mode;
      (
         [ me IsIn M ] -> Fax_Mode_Handler[u_action, n_action](Remove(me,M))
         []
         [ not(me IsIn M) ] -> Fax_Mode_Handler[u_action, n_action](Insert(me,M))
      )
      []
      u_action ? me: DecDigit ! insert_doc ! true [ me IsIn M];
      Fax_Mode_Handler[u_action, n_action](M)
      []
      u_action ? me: DecDigit ! insert_doc ! false [ not(me IsIn M) ];
      Fax_Mode_Handler[u_action, n_action](M)
      []
      u_action ? me: DecDigit ! key_in ? n: DecDigit;
      Fax_Mode_Handler[u_action, n_action](M)
      []
      u_action ? me: DecDigit ! hit_go;
      Fax_Mode_Handler[u_action, n_action](M)
      []
      u_action ? me: DecDigit ! insert_doc;
      Fax_Mode_Handler[u_action, n_action](M)

      []

      (* now we describe the global constraints on the fax machine *)
      (* related to events comming from the network *)

      n_action ? me: DecDigit ! ring ! true [ me IsIn M ];
      Fax_Mode_Handler[u_action, n_action](M)
      []
      n_action ? me: DecDigit ! ring ! false [ not(me IsIn M) ];
      Fax_Mode_Handler[u_action, n_action](M)
      []
```

```
         n_action ? me: DecDigit ! off_hook;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! hang_up;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! dial_tone;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! dial ? n: DecDigit;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! busy_signal;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! say ? w1: Fax_Message ? w2: Page_Counter;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! listen ? w1: Fax_Message ? w2: Page_Counter;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! say ? w1: Fax_Message;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! listen ? w1: Fax_Message;
         Fax_Mode_Handler[u_action, n_action](M)
         []
         n_action ? me: DecDigit ! no_connection_signal;
         Fax_Mode_Handler[u_action, n_action](M)
      )
      EndProc (* Fax_Mode_Handler *)

   (***************************************************************************)

   EndProc (* Global_Constraints *)

(******************************************************************************)

EndSpec
```