SYRF Case Studies
The SCHNEIDER example
An Experience with Mode-automata

Florence Maraninchi Yann Rémond Fabien Gaucher

VERIMAG*– Centre Equation, 2 Av. de Vignate – F38610 GIERES
http://www-verimag.imag.fr/PEOPLE/Florence.Maraninchi
(Florence.Maraninchi|Yann.Remond|Fabien.Gaucher)@imag.fr
Fax : (33) 4.76.63.48.50

Abstract

The SCHNEIDER case study is a program controlling the start-up and shut-down phases of a nuclear plant. The electric power generated by the plant is controlled by measuring the growing rate of the proportion of neutrons in the combustible, and acting on the rods. For ensuring fault tolerance, the physical system has two distinct measurement devices; moreover, each device is itself composed of three identical detectors (called *ictometers*) that receive electric power separately; a voting system manages the measures coming from these three detectors.

It is given as a SCADE program, with informal documentation (see details below).

The informal description talks about *modes* of the system: it is a typical case to which we can try to apply *mode-automata*. Mode-automata have been proposed in [MR98], as a means of introducing some imperative control structure in the dataflow language Lustre.

We aim at demonstrating that this new structure is useful in a dataflow language. This is not a case study for evaluating the efficiency of the compilation process. Indeed, we used the first implementation of mode-automata, i.e. a translation into pure Lustre.

1 The Case Study

1.1 System Interface and informal behavior

The inputs of the regulation system are of three kinds:

- Inputs from the physical measurement devices (values given by the ictometers). This information is given by two integers: the first one (CCC) is proportional to the number of neutrons; the second one (CCTT) is a time (a number of instants).

*Verimag is a joint laboratory of Université Joseph Fourier (Grenoble I), CNRS and INPG

- Logical inputs from the electronic components of the physical devices, reporting about correct functioning.
- Commands from the command panel (from a human being): setup mode, testing mode, ...

The outputs of the regulation system are of two kinds:

- Outputs to the command panel (lights, beeps, ...)
- Actions on the rods, in two different modes: constant rod drop at a constant speed, or reactor shut-down by gravity rod drop.

1.2 Sources

The case study is given as a Lustre-SCADE program, with an informal specification of the behavior. Three *modes* appear in the informal specification:

- Normal mode: normal computing of the outputs
- Setup mode: in this mode, detectors can be inspected. This mode can be entered only if the plant is not running.
- Defect mode: the plant enters this mode when a defect is detected somewhere. It leads to the complete stop of the plant.

2 Building a Mode-Automaton Program

2.1 Structure of The SCADE program

The SCADE program is structured in three nodes (see Figure 1):

- `TRAIT_NOMINAL` (normal computing) computes the thresholds from the inputs, and the protection actions. It is mainly made of calls to C functions and small Lustre nodes encoding flip-flops, counters, etc.
- `CHOIX_SORTIE` (Choose outputs) chooses the outputs to the display, depending on the current functioning mode (normal, setup or defect).
- `MODES_DE_FONCT` (running modes) computes the conditions under which the reactor must be shut down, the values of the variables corresponding to the setup and defect modes, and a variable that encodes the cause of the defect. This node is mainly combinational, with Boolean calculus.

2.2 Main modes and their influence on the outputs

We chose to encode the second one (`CHOIX_SORTIE`) with a mode-automaton. The structure of the automaton is given Figure 2.

The conditions for changing modes are given directly by *inputs* from the human user (`SetupMode`) or by a Boolean computed somewhere in the program (`DefectMode`).

The equations attached to states are given in the targos file (page 9).

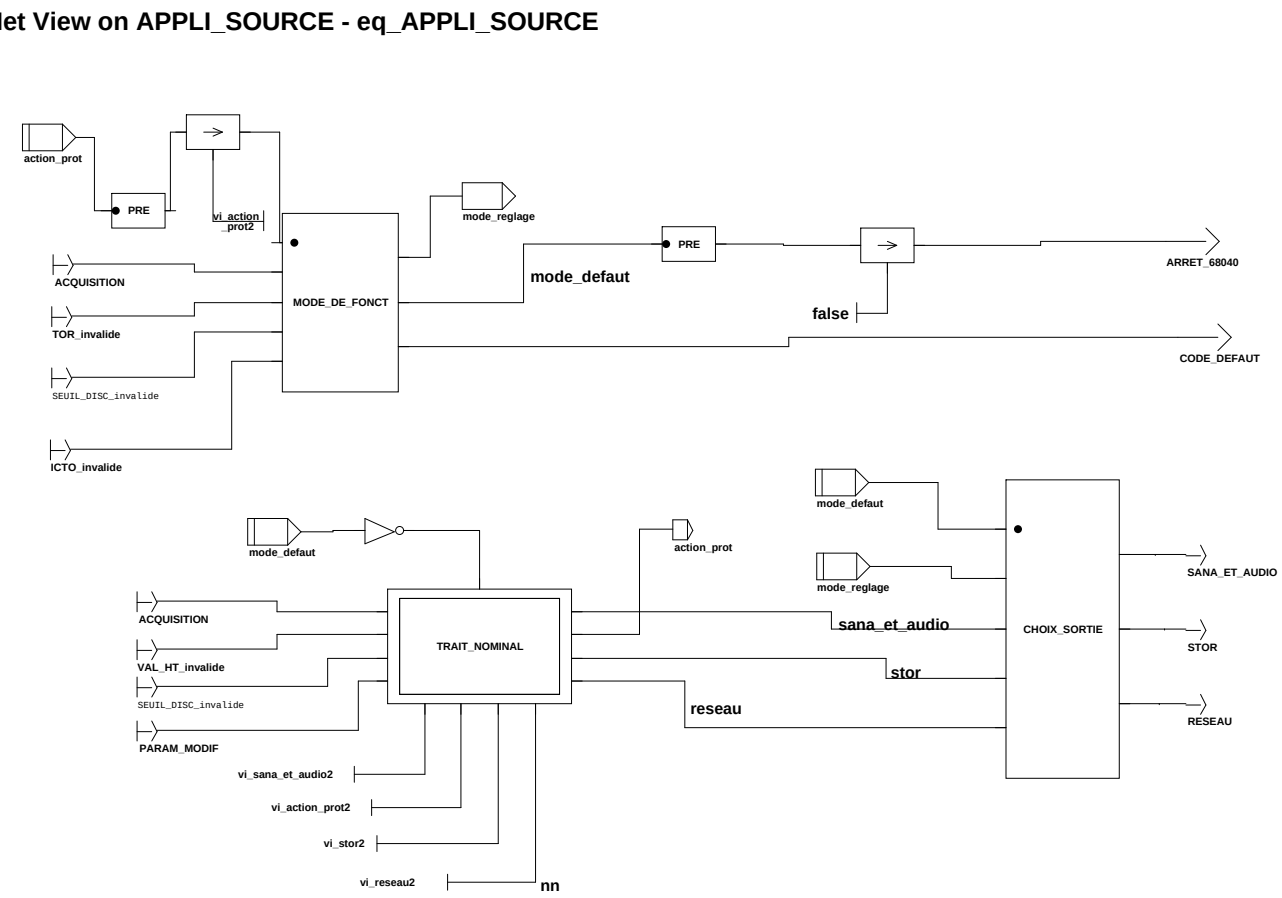

Figure 1: Main node of the SCADE program

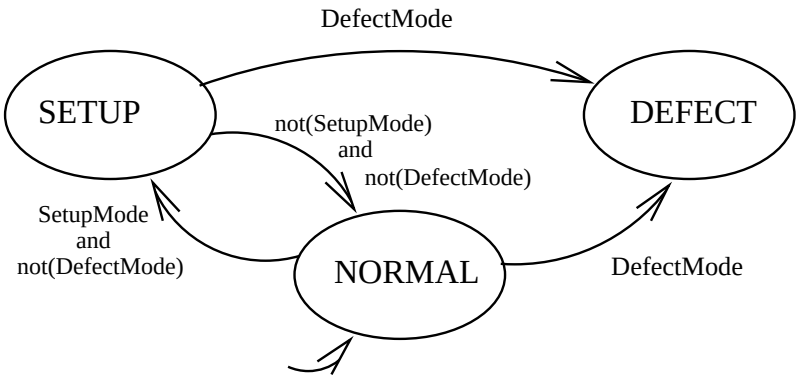

Figure 2: Main modes

- The equations attached to the states do not use the `pre` operator: the mode-automaton is in fact a Moore machine.
- The input `SetupMode` is related directly to a button of the command panel: the system isin setup mode as long as this input is true. The concrete syntax of mode-automata omits self-loops on the states, but the state `normal` should have a self-loop labeled by `not (SetupMode) and not (DefectMode)`, and the state `Setup` should have a self-loop labeled by SetupMode and not (DefectMode).
- The state `DefectMode` is a sink state (it should have a self-loop labeled by `true`).

2.3 Alternative description of small components

Some of the small components of the program cannot really be said to have "modes", but they are conveniently described using automata. We take one example here.

Figures 3, 4 and 5 are respectively the SCADE, Lustre and mode-automaton versions of the example called `MAINTIEN_ACT_ICTO`. The behavior is a follows: the example program takes an input `entree` and produces an output `e_maintenue` (meaning "input sustained") that becomes true when the input is true, and stays true during three instants of the basic clock. In the mode-automaton version, the state `M0`, `M1` and `M2` respectively mean: *the output only depends on the input (i.e. the last time the input was true is older than three ticks)*, *the output is true because of an input that was true, one tick ago*, *the output is true because of an input that was true, two ticks ago.*

3 Technical Stuff

This case study is an example in which the main structure is data-flow. Mode-automata are embedded in the program, as usual Lustre nodes. Hence the main node of the global program is not the one produced from mode-automata.

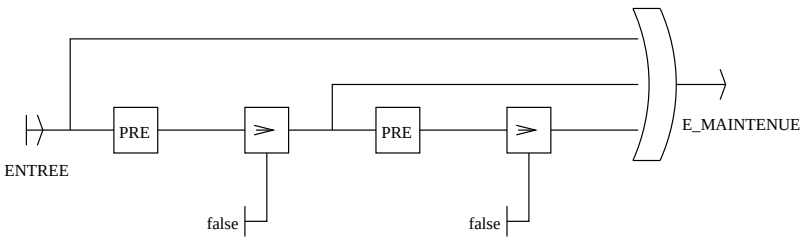

Figure 3: SCADE version of MAINTIEN_ACT_ICTO

```
node MAINTIEN_ACQ_ICTO (ENTREE : bool)
     returns           (E_MAINTENUE : bool) ;
var
     deux_acq    :  bool ;
     trois_acq   :  bool ;
     une_acq     :  bool ;
let
     deux_acq    = (false) -> (pre (une_acq)) ;
     trois_acq   = (false) -> (pre (deux_acq)) ;
     une_acq     = ENTREE ;
     E_MAINTENUE = une_acq or deux_acq or trois_acq ;
tel ;
```

Figure 4: Lustre version of MAINTIEN_ACT_ICTO

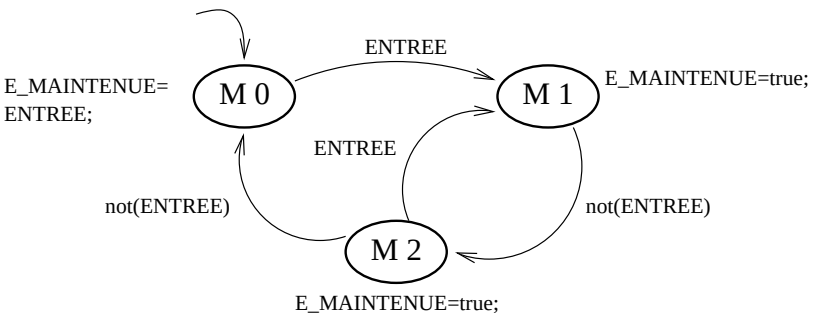

Figure 5: A Mode-automaton for MAINTIEN_ACT_ICTO

For dealing with such programs in which the main node may be written in Lustre, and several other nodes may be described as mode-automata (possibly calling other Lustre nodes), we extended the format used for the textual descripion of mode-automata: it is now possible to write several "processes" (compositions of mode-automata) in the `targos` file, and the end of the file may be a piece of Lustre code (see the annex A.1).

The compilation takes each process of the file, and produces a Lustre node having the same name. These nodes are added to the piece of Lustre code given directly, and this makes the global program. The main node may be in the generated part, or not.

4 Evaluation of the results

4.1 Tools

Using LESAR for Equivalence proofs

When a small component like `MAINTIEN_ACT_ICTO`, originally written in SCADE-lustre, is replaced by a piece of Lustre code produced from a mode-automaton, we need not prove that the global program has the same behavior. It is sufficient to prove that the two versions of `MAINTIEN_ACT_ICTO` are *equivalent*, and this implies that one may replace the other in any program context, without modifying the global behavior of the program.

For proving the equivalence of two Lustre programs, we use Leasr [HLR92]. `MAINTIEN_ACT_ICTO` is purely Boolean, and the equivalence is decidable.

Using Lurette for testing

When pieces of code involving numeric computations are replaced by other pieces of code, the equivalence may become undecidable. In these cases, we use the tool Lurette [RWNH98] for generating relevant test sequences, and we test the two programs on the same set of test sequences.

4.2 Observations

Concerning the use of a mode-automaton (which is in fact a Moore machine) for the component that chooses the outputs, depending on the current running mode, we do not obtain exactly the same behavior as before. The original version uses the Boolean input `SetupMode` (for instance) to characterize the set-up mode of the plant; it means that, as soon as `SetupMode` is true, the outputs to the command panel are the ones corresponding to the Setup mode. (In lustre, this is obvious in an equation like: `output1 = if SetupMode then ... else ...`).

In the mode-automaton version, however, the Boolean input `SetupMode` is used for changing states, from `Normal` to `Setup`. On the other hand, when the mode-automaton is in state `Normal` (resp. `Setup`), the outputs to the command panel are those corresponding to normal (resp. set-up) mode. The only difference between the mode-automaton version and the original version appears on the rising edges

of the input `SetupMode`. In the original version, the outputs instantaneously reflect this change; in the mode-automaton version, this rising edge makes the automaton change state, and the effect on the outputs is delayed to the next instant.

This would be the case in any example where the *running modes* are completely determined by inputs, and we want the internal modes of the program to reflect the levels of the outputs. (In other cases, modes are *computed* internally from the values of the inputs, as when a defect mode corresponds to an input crossing some threshold).

The mode-automaton we gave is *not equivalent* to the original version. However, the basic clock rate is 50ms in this example; and we do not consider a 50ms delay between an input from the human user and an output to him, to to be a serious problem.

Moreover, we could modify the mode-automaton in order to make its behavior be equivalent to the behavior of the original component. It would be a little less "natural" (see comments below, about Moore and Mealy machines).

5 Conclusion of the case study

5.1 Technical stuff

This case-study is an example in which the mode-structure is not at the upper level. The main structure is a data-flow network, but a part of it has modes. We added the possibility to write Lustre code in the mode-automaton program, and it works fine.

5.2 Mode-automata, Moore and Mealy machines

We used mode-automata here for describing Moore-like machines. In this case, the equations attached to states do not use `pre` operators. The mode-automaton used for choosing the outputs depending on the current mode (as specified by the inputs $\overline{e}$SetupMode and `DefectMode`) has the simplest form.

In general cases, it seems that replacing pieces of code by mode-automata introduces a delay, because of the Moore-like behavior of mode-automata. This is not true. The simplest use of mode-automata indeed introduces a delay, as in the *choose outputs* example, but there exists a translation from Mealy machines to mode-automata. We used it for the `MAINTIEN_ACT_ICTO` example: the transition from `M0` to `M1` is labeled by `ENTREE`, and the equation attached to the source state `M0` is `E_MAINTENUE=ENTREE`. This ensures a behavior similar to the behavior of a mealy machine where the transition from `M0` to `M1` would be labeled by `ENTREE/E_MAINTENUE`.

This is a general technique for avoiding the delay between inputs and outputs, when using a mode-automaton.

References

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying critical systems by means of the synchronous data-flow programming language LUSTRE. *IEEE Transactions on Software Ingeneering, Special Issue on the Specification and Analysis of Real-Time Systems*, September 1992.

[MR98] F. Maraninchi and Y. Rémond. Mode-automata: About modes and states for reactive systems. In *European Symposium On Programming*, Lisbon (Portugal), March 1998. Springer Verlag, LNCS 1381.

[RWNH98] P. Raymond, D. Weber, X. Nicollin, and N. Halbwachs. Automatic testing of reactive systems. In *19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

A The mode-automaton program

A.1 The program in targos

```
TARGOS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Outputs depending on the running mode
(Moore machine)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AUTOMATON MODE_SORTIES

STATES
NORMAL init [
 az3tco_f = (RESEAU_E.%0).%0;
 piftco_f = (RESEAU_E.%0).%1;
 az1pe_f = (RESEAU_E.%0).%2;
 az3pe_f = (RESEAU_E.%0).%3;

 tcol_bal = SANA_ET_AUDIO_E.%0;
 tcol_sc = SANA_ET_AUDIO_E.%1;
 tcol_sa = SANA_ET_AUDIO_E.%2;
 pel_bal = SANA_ET_AUDIO_E.%3;
 pel_sc = SANA_ET_AUDIO_E.%4;
 pel_sa = SANA_ET_AUDIO_E.%5;
 freq_audio = SANA_ET_AUDIO_E.%6;

 az3tco_nf = STOR_E.%0;
 piftco_nf = STOR_E.%1;
 az1pe_nf = STOR_E.%2;
 az3pe_nf = STOR_E.%3;
 st_az3tco_f = STOR_E.%9;
 st_piftco_f = STOR_E.%10;
 st_az1pe_f = STOR_E.%11;
 st_az3pe_f = STOR_E.%12;
 ]
REGLAGE [
 az3tco_f = true;
 piftco_f = true;
 az1pe_f = true;
 az3pe_f = true;

 tcol_bal = 0;
 tcol_sc = 0;
 tcol_sa = 0;
 pel_bal = 585;
 pel_sc = 2047;
 pel_sa = 585;
 freq_audio = 1;

 az3tco_nf = false;
 piftco_nf = false;
```

```
 az1pe_nf = false;
 az3pe_nf = false;
 st_az3tco_f = true;
 st_piftco_f = true;
 st_az1pe_f = true;
             st_az3pe_f = true;
             ]
    DEFAUT        [
             az3tco_f = (RESEAU_E.%0).%0;
             piftco_f = (RESEAU_E.%0).%1;
             az1pe_f = (RESEAU_E.%0).%2;
             az3pe_f = (RESEAU_E.%0).%3;

             tcol_bal = SANA_ET_AUDIO_E.%0;
             tcol_sc = SANA_ET_AUDIO_E.%1;
             tcol_sa = SANA_ET_AUDIO_E.%2;
             pel_bal = SANA_ET_AUDIO_E.%3;
             pel_sc = SANA_ET_AUDIO_E.%4;
             pel_sa = SANA_ET_AUDIO_E.%5;
             freq_audio = 1;

             az3tco_nf = STOR_E.%0;
             piftco_nf = STOR_E.%1;
             az1pe_nf = STOR_E.%2;
             az3pe_nf = STOR_E.%3;
             st_az3tco_f = STOR_E.%9;
             st_piftco_f = STOR_E.%10;
             st_az1pe_f = STOR_E.%11;
             st_az3pe_f = STOR_E.%12;
            ]
TRANS
    FROM NORMAL TO REGLAGE WITH     [ MODE_REGLAGE and not(MODE_DEFAUT) ]
    FROM NORMAL TO DEFAUT WITH      [ MODE_DEFAUT ]
    FROM REGLAGE TO DEFAUT WITH     [ MODE_DEFAUT ]
    FROM REGLAGE TO NORMAL WITH     [ not(MODE_REGLAGE) and not(MODE_DEFAUT) ]


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
TrueDuring_250ms (= 5 x basic clock)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AUTOMATON VraiPdt_250ms

STATES
        V1 init         [compt=if not ENTREE then 0 else pre(compt)+1;
                         E_PDT_250ms=false;
                        ]
        V2              [compt=0;
                         E_PDT_250ms=ENTREE;
                        ]
TRANS
        FROM V1 TO V2 WITH         [ compt=5 ]
```

```
        FROM V2 TO V1 WITH         [ ENTREE=false ]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
BASCULE_SR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AUTOMATON BASCULE_SR

STATES
        OFF init        [ETAT=set;]
        ON              [ETAT=not(reset and not(set));]

TRANS
        FROM ON TO OFF WITH        [reset and not(set)]
        FROM OFF TO ON WITH        [set]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MAINTIEN_ACQ_ICTO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AUTOMATON MAINTIEN_ACQ_ICTO

STATES
    M1 init    [E_MAINTENUE=ENTREE;]
    M2    [E_MAINTENUE=true;]
    M3    [E_MAINTENUE=true;]
TRANS
    FROM M1 TO M2 WITH  [ENTREE]
    FROM M2 TO M3 WITH  [not(ENTREE)]
    FROM M3 TO M2 WITH  [ENTREE]
    FROM M3 TO M1 WITH  [not(ENTREE)]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
TEMPO_FRT_MONTANT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

AUTOMATON TEMPO_FRT_MONTANT

STATES
        T1 init         [compt=if not ENTREE then 50 else pre(compt)+50;
                         SORTIE_TEMPORISEE=false;
                        ]
        T2              [compt=0;
                         SORTIE_TEMPORISEE=ENTREE;
                        ]
TRANS
        FROM T1 TO T2 WITH         [ compt>=DUREE_EN_S*1000 ]
        FROM T2 TO T1 WITH         [ ENTREE=false ]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FDB_CALC_TEMPO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
AUTOMATON FDB_CALC_TEMPO

STATES
        F1 init         [compt=if demarrage and not(abort) then 50 else 0;
                         fin_tempo = not(not(abort) and demarrage) or compt>=tempo;
                        ]
        F2              [compt= if demarrage and not(abort) then 50
                                else if abort then 0 else pre(compt)+50;
                         fin_tempo = compt>=tempo or abort;
                        ]
TRANS
        FROM F1 TO F2 WITH          [ demarrage and not(abort)]
        FROM F2 TO F1 WITH          [ abort ]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


PROCESS FDB_CALC_TEMPO    [ in(demarrage,abort : bool; tempo : int) ,
                            out(fin_tempo=false : bool) ,
                            local(compt=0 : int)]
  RAFF FDB_CALC_TEMPO


PROCESS TEMPO_FRT_MONTANT   [ in(ENTREE : bool; DUREE_EN_S : int) ,
                              out(SORTIE_TEMPORISEE=false : bool) ,
                              local(compt=0 : int)]
  RAFF TEMPO_FRT_MONTANT


PROCESS MAINTIEN_ACQ_ICTO [in(ENTREE : bool) , out(E_MAINTENUE=false : bool)]
   RAFF MAINTIEN_ACQ_ICTO


PROCESS BASCULE_SR [in(set,reset : bool) , out(ETAT=false : bool)]
  RAFF BASCULE_SR


PROCESS VraiPdt_250ms    [ in(ENTREE : bool) ,
                           out(E_PDT_250ms = false : bool) ,
              local(compt=0 : int)]
  RAFF VraiPdt_250ms


PROCESS MODE_SORTIES [
     in  ( MODE_DEFAUT , MODE_REGLAGE : bool;
           SANA_ET_AUDIO_E : t_sana_et_audio;
           STOR_E : t_stor; RESEAU_E : t_reseau ) ,

     out ( az3tco_f=&(RESEAU_E.%0).%0&, piftco_f=&(RESEAU_E.%0).%1&,
           az1pe_f=&(RESEAU_E.%0).%2&, az3pe_f=&(RESEAU_E.%0).%3& : bool;
```

```
           tcol_bal=&SANA_ET_AUDIO_E.%0&, tcol_sc=&SANA_ET_AUDIO_E.%1&,
           tcol_sa=&SANA_ET_AUDIO_E.%2&, pel_bal=&SANA_ET_AUDIO_E.%3&,
           pel_sc=&SANA_ET_AUDIO_E.%4&, pel_sa=&SANA_ET_AUDIO_E.%5&,
           freq_audio=&SANA_ET_AUDIO_E.%6& :int;

            az3tco_nf=&STOR_E.%0&, piftco_nf=&STOR_E.%1&,
            az1pe_nf=&STOR_E.%2&, az3pe_nf=&STOR_E.%3&,
            st_az3tco_f=&STOR_E.%9&, st_piftco_f=&STOR_E.%10&,
            st_az1pe_f=&STOR_E.%11&,
            st_az3pe_f=&STOR_E.%12& : bool ) ]
  RAFF MODE_SORTIES



%% here starts pure Lustre code, to be added to the Lustre
%% program generated from the above mode-automaton.
%% Note that the main node is in the pure Lustre part. The nodes generated
%% from the mode-automata described above are :
%%
%% FDB_CALC_TEMPO, TEMPO_FRT_MONTANT, VraiPdt_250ms and MODE_SORTIES
%% called directly from the main node TDF (see below)
%% MAINTIEN_ACQ_ICTO, BASCULE_SR
%% called by  nodes themselves called bt TDF, and described in
%% auxiliary files bibli.lus, trait_nom.lus or calc_per.lus
%%

[
include "bibli.lus"
include "type_const.lus"
include "trait_nom.lus"
include "calc_per.lus"

-------------------- Main node TDF ---------------------------------

node TDF (ACQUISITION : t_acquisition ;
          PARAM_MODIF : t_param_modif ;
          VAL_HT_val : bool ;
          SEUIL_DISC_val : bool ;
          BF_ELECT_val : bool ;
          BF_ELECT_val : bool ;
          BF_ELECT_val : bool ; RAF_ICTO_val : bool)
      returns (SANA_ET_AUDIO : t_sana_et_audio ;
               RESEAU : t_reseau ;
               STOR : t_stor ;
               ARRET_68040 : bool ; CODE_DEFAUT : int) ;
let
    SANA_ET_AUDIO , STOR , RESEAU , ARRET_68040 , CODE_DEFAUT =
       APPLI_SOURCE (ACQUISITION , PARAM_MODIF ,
                     BF_ELECT_val , VAL_HT_val ,
                     SEUIL_DISC_val , RAF_ICTO_val);
tel;
```

```
node APPLI_SOURCE  (ACQUISITION : t_acquisition ;
                    PARAM_MODIF : t_param_modif ;
                    TOR_invalide : bool ;
                    VAL_HT_invalide : bool ;
                    SEUIL_DISC_invalide : bool ;
                    ICTO_invalide : bool)
        returns    (SANA_ET_AUDIO_S : t_sana_et_audio ; STOR_S : t_stor ;
                    RESEAU_S : t_reseau ;
                    ARRET_68040 : bool ; CODE_DEFAUT : int) ;

var
    MODE_DEFAUT, MODE_REGLAGE : bool;
    action_prot : t_action_prot;
    SANA_ET_AUDIO_E : t_sana_et_audio;
    STOR_E : t_stor;
    RESEAU_E : t_reseau;
let
    ARRET_68040 = (false) -> (pre(MODE_DEFAUT));
    SANA_ET_AUDIO_S , STOR_S,RESEAU_S =
          CHOIX_SORTIE(MODE_DEFAUT , MODE_REGLAGE ,
                       SANA_ET_AUDIO_E , STOR_E,RESEAU_E);
    SANA_ET_AUDIO_E , action_prot , STOR_E , RESEAU_E =
          ACT_TRAIT_NOMINAL (not(MODE_DEFAUT) , ACQUISITION ,
                             VAL_HT_invalide , SEUIL_DISC_invalide , PARAM_MODIF);
    MODE_REGLAGE , MODE_DEFAUT , CODE_DEFAUT =
          MODE_DE_FONCT(pre(action_prot) , ACQUISITION , TOR_invalide ,
                        SEUIL_DISC_invalide , ICTO_invalide) ;
tel;


node CHOIX_SORTIE (MODE_DEFAUT : bool ;
                   MODE_REGLAGE : bool ;
                   SANA_ET_AUDIO_E : t_sana_et_audio ;
                   STOR_E : t_stor ; RESEAU_E : t_reseau)
        returns   (SANA_ET_AUDIO_S : t_sana_et_audio ;
                   STOR_S : t_stor ; RESEAU_S : t_reseau) ;

var
    az3tco_f, piftco_f, az1pe_f, az3pe_f, Indic_val_mes, bf_ch_src : bool;
    tcol_bal, tcol_sc, tcol_sa, pel_bal, pel_sc, pel_sa,
    freq_audio, select_fact_div, facteur_div :int;
    az3tco_nf, piftco_nf, az1pe_nf, az3pe_nf,
    t5, t6, t7, t8, t9,
    st_az3tco_f, st_piftco_f, st_az1pe_f, st_az3pe_f : bool;

let
-- call to the node generated from the mode-automaton program
--------------------------------------------------------------
    az3tco_f, piftco_f, az1pe_f, az3pe_f,
    tcol_bal, tcol_sc, tcol_sa, pel_bal, pel_sc, pel_sa, freq_audio,
    az3tco_nf, piftco_nf, az1pe_nf, az3pe_nf, st_az3tco_f,
```

```
    st_piftco_f, st_az1pe_f, st_az3pe_f =
    MODE_SORTIES(MODE_DEFAUT , MODE_REGLAGE , SANA_ET_AUDIO_E , STOR_E , RESEAU_E);

    RESEAU_S = {{az3tco_f,piftco_f,az1pe_f,az3pe_f,Indic_val_mes,bf_ch_src},
            RESEAU_E.%1,RESEAU_E.%2,RESEAU_E.%3,RESEAU_E.%4,
            RESEAU_E.%5,RESEAU_E.%6,RESEAU_E.%7,RESEAU_E.%8,
            RESEAU_E.%9,RESEAU_E.%10,RESEAU_E.%11};
    Indic_val_mes=(RESEAU_E.%0).%4;
    bf_ch_src=(RESEAU_E.%0).%5;

    SANA_ET_AUDIO_S = {tcol_bal,tcol_sc,tcol_sa,pel_bal,pel_sc,pel_sa,freq_audio,
               select_fact_div,facteur_div};
    select_fact_div=SANA_ET_AUDIO_E.%7;
    facteur_div=SANA_ET_AUDIO_E.%8;

    STOR_S = {az3tco_nf,piftco_nf,az1pe_nf,az3pe_nf,t5,t6,t7,t8,t9,
            st_az3tco_f,st_piftco_f,st_az1pe_f,st_az3pe_f};
    t5=STOR_E.%4;
    t6=STOR_E.%5;
    t7=STOR_E.%6;
    t8=STOR_E.%7;
    t9=STOR_E.%8;

tel;


node MODE_DE_FONCT (ACTION_PROT_PRE : t_action_prot ;
                    ACQUISITION : t_acquisition ;
                    TOR_Invalide : bool ;
                    Seuil_Disc_invalide : bool ;
                    ICTO_Invalide : bool)
      returns (MODE_REGLAGE : bool ; MODE_DEFAUT : bool ; CODE_DEFAUT : int) ;

var
    az3tco_f_pre , piftco_f_pre , az1pe_f_pre , az3pe_f_pre : bool;
    bf_cf_alim , rearm_ht  ,rel_az3tco_nf , rel_piftco_nf ,
    rel_az1pe_nf  ,rel_az3pe_nf , ch_en_tst ,
    ch_en_regl , seuil_h_ht_nf , seuil_b_ht_nf ,
    bf_elect , porte_fermee , bf_ventil : bool;
    val_ht , seuil_disc : int;
    raf_icto : bool;
    cct , ccc , cctt , tps_en_ms : int;
    ACQ_TOR_ANA : t_acq_tor_ana;
    Cond1ou2ou3,Cond4,Cond5,Cond6,Cond7,Cond8,Cond9,Cond10,Cond11 : bool;
    defaut_autre_que_0_cps , perte_de_comptage_confirme , situation_zero_coups,
    perte_de_comptage_non_conf , demarrage ,
    TMAX_est_ecoule , icto_invalide_apres_une_acq : bool;
let
    MODE_REGLAGE = ch_en_regl;
    MODE_DEFAUT = Cond4 or Cond5 or Cond6 or Cond7 or Cond8 or
              Cond9 or Cond1ou2ou3 or Cond10 or Cond11 ;
    CODE_DEFAUT =   if Cond1ou2ou3 then (Code_def_1ou2ou3)
```

```
        else (if Cond4 then (Code_def_4)
        else (if Cond5 then (Code_def_5)
        else (if Cond6 then (Code_def_6)
        else (if Cond7 then (Code_def_7)
        else (if Cond8 then (Code_def_8)
        else (if Cond9 then (Code_def_9)
        else (if Cond10 then (Code_def_10)
        else (Code_def_11)))))))) ;

Cond4 = ch_en_regl and  not (rearm_ht) and ((rearm_ht) -> (pre (rearm_ht))) ;
Cond5 = ch_en_regl and ch_en_tst ;
Cond6 =  not (bf_elect) and  not (ch_en_regl) ;
Cond7 =  not (seuil_h_ht_nf) and  not (ch_en_regl) ;
Cond9 =  not (Seuil_Disc_invalide) and (seuil_disc >= SHDI) and  not (ch_en_regl) ;
Cond10 = VraiPdt_250ms((not(ch_en_regl) and
         (((rel_az1pe_nf) -> (az1pe_f_pre)) = rel_az1pe_nf))) or
           VraiPdt_250ms((not(ch_en_regl) and
           (((rel_az3tco_nf) -> (az3tco_f_pre)) = rel_az3tco_nf))) or
           VraiPdt_250ms((not(ch_en_regl) and
           (((rel_az3pe_nf) -> (az3pe_f_pre)) = rel_az3pe_nf))) or
           VraiPdt_250ms((not(ch_en_regl) and
           (((rel_piftco_nf) -> (piftco_f_pre)) = rel_piftco_nf))) ;

Cond8 = TEMPO_FRT_MONTANT((not(seuil_b_ht_nf) and rearm_ht and not(ch_en_regl)), 2);
Cond11 = TOR_Invalide;

Cond1ou2ou3 = defaut_autre_que_0_cps or perte_de_comptage_confirme ;

defaut_autre_que_0_cps =  not (situation_zero_coups) and
                          icto_invalide_apres_une_acq ;
perte_de_comptage_confirme = TEMPO_FRT_MONTANT(perte_de_comptage_non_conf , 50) ;
situation_zero_coups = (ccc = 0) and (cctt = TMAX_en_1600eme_de_s) ;
perte_de_comptage_non_conf = situation_zero_coups and icto_invalide_apres_une_acq ;
demarrage = true -> false ;
TMAX_est_ecoule = FDB_CALC_TEMPO(demarrage , false , tps_en_ms) ;
tps_en_ms = 1000 * TMAX_ICTO_en_int ;
icto_invalide_apres_une_acq = if TMAX_est_ecoule
                              then (ICTO_Invalide)
                              else ((ICTO_Invalide and raf_icto)) ;


az3tco_f_pre = ACTION_PROT_PRE.%0;
piftco_f_pre = ACTION_PROT_PRE.%1;
az1pe_f_pre = ACTION_PROT_PRE.%2;
az3pe_f_pre = ACTION_PROT_PRE.%3;

ACQ_TOR_ANA = ACQUISITION.%0;
bf_cf_alim = ACQ_TOR_ANA.%2 ;
rearm_ht = ACQ_TOR_ANA.%3 ;
rel_az3tco_nf = ACQ_TOR_ANA.%4 ;
rel_piftco_nf = ACQ_TOR_ANA.%5 ;
rel_az1pe_nf = ACQ_TOR_ANA.%6 ;
```

```
    rel_az3pe_nf = ACQ_TOR_ANA.%7 ;
    ch_en_tst = ACQ_TOR_ANA.%8 ;
    ch_en_regl = ACQ_TOR_ANA.%9 ;
    seuil_h_ht_nf = ACQ_TOR_ANA.%10 ;
    seuil_b_ht_nf = ACQ_TOR_ANA.%11 ;
    bf_elect = ACQ_TOR_ANA.%12 ;
    porte_fermee = ACQ_TOR_ANA.%13 ;
    bf_ventil = ACQ_TOR_ANA.%14 ;
    val_ht = ACQ_TOR_ANA.%15 ;
    seuil_disc = ACQ_TOR_ANA.%16 ;

    raf_icto = (ACQUISITION.%1).%0;
    cct = (ACQUISITION.%1).%1;
    ccc = ((ACQUISITION.%1).%2).%0;
    cctt = ((ACQUISITION.%1).%2).%1;
tel;
]

ENDTARGOS
```