```
specification GPRSGroupcall[ch_Uplink, ch_Downlink]:noexit


      (* written by Pascal Forhan *)
      (* for the INT and the University of Ottawa *)
      (* Ottawa April-June 1997 *)


 library
      Boolean, NaturalNumber, HexDigit
 endlib

(*========================================================*)
(*                 IS8807 ADT Definitions                 *)
(*========================================================*)

(* types Fboolean, Element and Set  contain corrections *)
(* to the library from the International satandard 8807 *)

 type   Fboolean is
 formalsorts FBool
 formalopns     true :-> FBool
            not  : FBool-> FBool

 formaleqns
      forall x: FBool
      ofsort FBool
      not(not(x))=x;

 endtype   (* type Fboolean *)

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

 type Element is Fboolean
 formalsorts Element
 formalopns _eq_, _ne_ : Element, Element -> FBool

 formaleqns
      forall x,y,z : Element
      ofsort Element
            x eq y = true => x=y;
      ofsort FBool
            x=y => x eq y = true;
            x eq y =true, y eq z =true => x eq z =true;
            x ne y = not(x eq y);

 endtype (* type Element *)

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

 type Set is Element, Boolean, NaturalNumber
 sorts Set
 opns   {} : -> Set
      NoElement: -> Element
      Insert, Remove: Element, Set -> Set
      Top: Set -> Element
      _Isin_, _Notin_ : Element, Set -> Bool
```



```
      _Union_, _Ints_, _Minus_ : Set, Set -> Set
      _eq_, _ne_, _Includes_, _IsSubsetof_ : Set, Set -> Bool
      Card: Set-> Nat

 eqns
      forall  x,y : Element,
            s,t : Set
      ofsort Set
            x Isin Insert( y,s) => Insert(x, Insert(y,s))= Insert(y,s);

            Remove(x, {}) = {};
            Remove(x, Insert (x,s)) = s;
            x ne y= true of FBool => Remove(x, Insert(y,s))= Insert(y, Remove(x,s));

            {} Union s = s;
            Insert(x,s) Union t = Insert(x,s Union t);
            {} Ints s = {};
            x Isin t => Insert(x,s) Ints t = Insert(x, s Ints t);
            x Notin t => Insert(x,s) Ints t = s Ints t;

            s Minus {} = s;
            s Minus Insert(x,t) = Remove(x, s) Minus t;

      ofsort Bool
            x Isin {} = false;
            x eq y = true of FBool => x Isin Insert(y,s) = true;
            x ne y = true of FBool => x Isin Insert(y, s) = x Isin s;

            x Notin s = not(x Isin s);

            s Includes {} = true;
            s Includes Insert(x,t) = (x Isin s) and (s Includes t);

            s IsSubsetof t = t Includes s;

            s eq t = (s Includes t) and (s IsSubsetof t);

            s ne t = not(s eq t);

      ofsort Nat
            Card({}) = 0;
            x Notin s => Card(Insert(x,s)) = Succ(Card(s));

      ofsort Element
            Top(Insert(x,s)) = x;
            Top({}) = NoElement;

 endtype (* type Set *)

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



(*=====================================================================*)
(*                GroupCall ADT Definitions                            *)
(*=====================================================================*)
```



```
(* type M_ID: member identifier, mapping on Hexa Decimal numbers *)

 type M_ID is HexDigit renamedby
 sortnames M_ID for Hexdigit

 opnnames
      NoM_ID for 0
      Adm1 for 1
      Init1 for 2
      Init2 for 3
      Mem1 for 4
      Mem2 for 5
      Mem3 for 6
      Mem4 for 7
      Adm2 for 8
      Init3 for 9
      Init4 for A
      Mem5 for B
      Mem6 for C
      Mem7 for D
      Mem8 for E
      Mem9 for F

 endtype (* type M_ID *)

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

(* type IMGI: International Mobile Group Identity, mapping on hexa Decimal Numbers *)

 type IMGI is HexDigit renamedby
 sortnames IMGI for HexDigit

 opnnames
      NoIMGI for 0

 endtype (* type IMGI *)

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

(* type C_ID, Call Identifier based on Natural*)


 type C_ID is NaturalNumber
 sorts C_ID

 opns
      NoC_ID: -> C_ID
      NatNum: C_ID -> Nat
      Succ: C_ID -> C_ID
      _eq_, _ne_: C_ID, C_ID -> Bool

 eqns
      forall cid, cid2: C_ID

      ofsort Nat
      natNum(NoC_id) = 0;
```



```
      NatNum(Succ(cid)) = Succ(NatNum(cid));

      ofsort Bool
      cid eq cid2 = NatNum(cid) eq NatNum(cid2);
      cid ne cid2 = Not( cid eq cid2);

endtype (* type C_ID *)

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

(* type GeoZone: geographical area, mapping on Hexa Decimal numbers *)

 type GeoZone is HexDigit renamedby
 sortnames GeoZone for HexDigit

 opnnames
      NoGeozone for 0

 endtype (* type GeoZone *)

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

(* type QoS: Quality of Service, implementation of the three types of service *)

 type QoS is
 sorts QoS

 opns
      qos1, qos2, qos3 : -> QoS

 endtype (* type QoS *)

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

(* type DTM: Data Transfer Mode, 3 possibilities: *)
(* one_way (one to many), two_way (many to one) and n_way(many to many) *)

 type DTM is NaturalNumber
 sorts DTM

 opns
      noway,one_way, two_way, n_way: -> DTM    (* noway if no DTM defined *)
      h: DTM -> Nat
      _eq_, _ne_ : DTM, DTM -> Bool

 eqns
      forall X,Y: DTM

      ofsort Nat

      h(noway)= 0;
      h(one_way) = Succ(h(noway));
      h(two_way)= Succ(h(one_way));
      h(n_way)= Succ(h(two_way));

      ofsort Bool
```

```
     X eq Y = h(X) eq h(Y);
     X ne Y = h(X) ne h(Y);

endtype (* type DTM *)

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

(* type Request, mapping on Hexa Decimal numbers *)
(* messages sent from the requester to the network *)

type Request is HexDigit renamedby
sortnames Req for HexDigit

opnnames

     Req_Attach for 1
     Req_Detach for 2
     Req_Init for 3
     Req_Term for 4
     Req_Status for 5
     Req_Join for 6
     Req_Leave for 7
     Req_Data for 8
     Req_Change for 9

endtype (* type Request *)

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

(* type Acknowledgment, mapping on hexa decimal numbers *)
(* messages sent from the network to the requester *)

type Acknowledgment is HexDigit renamedby
sortnames Ack for HexDigit

opnnames
     Ack_Attach for 1
     Ack_Detach for 2
     Ack_C_Init for 3
     Ack_C_Term for 4
     Ack_Status for 5
     Ack_Join for 6
     Ack_Leave for 7
     Ack_Sinit for 8
     Ack_Change for 9
     Ack_SAll for A

endtype  (* type Acknowledgment *)

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

(* type Indication,, mapping on Hexa decimal numbers *)
(* messages sent from the network to the receivers *)

type Indication is HexDigit renamedby
sortnames Ind for HexDigit
```



```
opnnames
     Ind_Init for 3
     Ind_Term for 4
     Ind_Join for 6
     Ind_Leave for 7
     Ind_Data for 8


endtype (* type Indication *)

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

(* type Error, mapping on hexa Decimal Numbers *)
(* messages of error sent from the network to the requester *)

type Error is HexDigit renamedby
sortnames Err for HexDigit

opnnames
     Err_Rejatt for 1
     Err_Rejinit for 3
     Err_Rejterm for 4
     Err_Rejstat for 5
     Err_Rejjoin for 6
     Err_Rejleave for 7
     Err_Rejdat for 8
     Err_Wmode for A

endtype (* type Error *)

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

(* type Statusinfotype, mapping on hexa Decimal Numbers *)
(* messages of additional information passed in parameters *)
(* during the status request *)

type Statusinfotype is HexDigit renamedby
sortnames Statusinfo for HexDigit

opnnames
     NoStatusinfo for 0
     Sta_callids for 1
     Sta_joinind for 2
     Sta_geozone for 3
     Sta_memlist for 4
     Sta_nbmem   for 5
     Sta_qos     for 6
     Sta_DTM     for 7

endtype (* type Statusinfotype *)

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

(*  type DBrequest, mapping on hexa decimal Numbers *)
(*  modelizes the requests of of classic DataBase    *)
(* used for the two bases DBCI and DBGC              *)
```



```
type DBRequesttype is HexDigit renamedby
sortnames DBRequest for HexDigit

opnnames
     Insert for 1
     Delete for 2
     Ask for 3

endtype (* type DBRequesttype *)


(******************************************************************************)
(* type Statusinfolist0, set of Statusinfotype *)

type Statusinfolist0 is Set
actualizedby Statusinfotype using
sortnames
     Statusinfo for Element
     Bool for FBool

endtype (* type Statusinfolist0 *)

(* type Statusinfolist1, renaming of statusinfolist0 *)

type Statusinfolist1 is Statusinfolist0 renamedby
sortnames Statusinfolist for Set

endtype(* type Statusinfolist1 *)

(* type Statusinfolist, addition of comparison with NoElement to statusinfolist1 *)
(* permits the use of a list of statusinfotype for the Status request *)

type Statusinfolist is Statusinfolist1

eqns
     ofsort Nat
     NatNum(NoElement) = NatNum(NoStatusinfo);


endtype (* type Statusinfolist *)


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

(* type M_IDlist0, set of member ID *)

type M_IDlist0 is Set
actualizedby M_ID using
sortnames
     M_ID for Element
     Bool for FBool

endtype (* type M_IDlist0 *)

(* type M_IDlist1, renaming of M_IDlist0 *)

type M_IDlist1 is M_IDlist0 renamedby
```



```
sortnames M_IDlist for Set
endtype (* type M_IDlist *)

(* type M_IDlist, addition of comparison with NoElement to M_IDlist1 *)
(* permits the use of a list of M_ID *)

type M_IDlist is M_IDlist1

eqns
     ofsort Nat
     NatNum(NoElement) = NatNum(NoM_ID);

endtype (* type M_IDlist *)


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

(* type Membertype, describes the priviledges of the member *)
(* three possibilities: Cont (Controller, administrator), Init (Initiator) *)
(* part (participant, standard member) *)

type Membertype is NaturalNumber
sorts Membertype

opns
     Nomember, cont, init, part : -> Membertype
     h : Membertype -> Nat
     _eq_, _ne_ : Membertype, Membertype -> Bool

eqns
     forall X,Y: Membertype

     ofsort Nat

     h(Nomember)= 0;
     h(cont) = Succ(h(Nomember));
     h(init)= Succ(h(cont));
     h(part)= Succ(h(init));

     ofsort Bool

     X eq Y = h(X) eq h(Y);
     X ne Y = h(X) ne h(Y);

endtype (* type DTM *)

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

(* type Attach, based on Boolean to provide the Attached/detached status *)
(* not a renaming because lotos want a Boolean expression to evaluate gards *)

type Attach is Boolean
sorts Attach

opns
     Detached, Attached: -> Attach
     _eq_, _ne_: Attach, Attach -> Bool
```

```
    h: Attach -> Bool

eqns
    forall x,y: Attach

    ofsort Bool

    h(Attached) = true;
    h(Detached) = false;

    x eq y = h(x) eq h(y);
    x ne y = not(x eq y);

endtype (* type Attach *)

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

(* type Join_leave, renaming of Attach to provide the Join/Leave indication *)

type Join_leave is Attach renamedby
sortnames Join_leave for Attach

opnnames
    No_join_leave_ind for Detached
    Join_leave_ind for Attached

endtype (* type Join_leave *)

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

(* type Send_to_all, renaming of Attach to provide the send to all option in the *)
(* Join/Leave indication *)

type Send_to_all is Attach renamedby
sortnames Send_to_all for Attach

opnnames
    No_send_to_all for Detached
    Send_to_all_ind for Attached

endtype (* type Send_to_all  *)

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

(* type Call_not, renaming of Attach to provide the call notification option *)

type Call_not is Attach renamedby
sortnames Call_not for Attach

opnnames
    No_call_not for Detached
    Call_not_ind for Attached

endtype (* type Call_not *)

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



```
(* type C_IDlist0, set of C_ID *)

type C_IDlist0 is Set
actualizedby C_ID using
sortnames
    C_ID for Element
    Bool for FBool

endtype (* type C_IDlist0 *)

(* type C_IDlist1, renaming of C_IDlist0 *)

type C_IDlist1 is C_IDlist0 renamedby
sortnames C_IDlist for Set

endtype (* type C_IDlist1 *)

(* type C_IDlist, addition of comparison with NoElement to C_IDlist1 *)
(* permits the use of lists of C_ID *)

type C_IDlist is C_IDlist1

eqns
    ofsort Nat
    NatNum(NoElement) = NatNum(NoC_ID);

endtype (* type C_IDlist *)

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

(* type IMGIlist0, set of IMGI *)

type IMGIlist0 is Set
actualizedby IMGI using
sortnames
    IMGI for Element
    Bool for FBool

endtype (* type IMGIlist0 *)

(* type IMGIlist renaming of IMGIlist0 *)
(* permits the use of lists of IMGI *)

type IMGIlist is IMGIlist0 renamedby
sortnames IMGIlist for Set

endtype (* type IMGIlist *)

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

(* type Statusmembertype, record of the caracteristics of a member *)

type Statusmembertype is M_ID, Membertype, IMGIlist, C_IDlist, Attach
sorts Statusmember

opns
    SM: M_ID, Membertype, IMGIlist,C_IDlist, Attach -> Statusmember
```



```
    _eq_, _ne_ : Statusmember, Statusmember -> Bool
    M_IDSta: Statusmember -> M_ID
    MemSta: Statusmember -> Membertype
    IMGISta: Statusmember -> IMGIlist
    C_IDSta: Statusmember -> C_IDlist
    AttSta: Statusmember -> Attach

eqns   forall s1, s2 : Statusmember, mid, mid2: M_ID, mem, mem2: Membertype,
         imgil,imgil2: IMGIlist, cidl, cidl2: C_IDlist, att, att2: Attach
    ofsort Bool
    mid eq mid2 =>
      SM(mid, mem, imgil, cidl, att) eq SM(mid2, mem2, imgil2, cidl2, att2)
                                                  = true;
    mid ne mid2 =>
      SM(mid, mem, imgil, cidl, att) eq SM(mid2, mem2, imgil2, cidl2, att2)
                                                  = false;
    s1 ne s2 = not(s1 eq s2);

    ofsort M_ID
    M_IDSta(SM(mid, mem, imgil, cidl, att)) = mid;

    ofsort Membertype
    MemSta(SM(mid, mem, imgil, cidl, att)) = mem;

    ofsort IMGIlist
    IMGISta(SM(mid, mem, imgil, cidl, att)) = imgil;

    ofsort C_IDlist
    C_IDSta(SM(mid, mem, imgil, cidl, att)) = cidl;

    ofsort Attach
    AttSta(SM(mid, mem, imgil, cidl, att)) = att

endtype  (* type Statusmembertype *)

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

(* type Callmembertype, record of the call ID and the members ID *)

type Callmembertype is M_IDlist, C_IDlist
sorts Callmember

opns
    CM: C_ID, M_ID, M_IDlist -> Callmember
    _eq_, _ne_ : Callmember, Callmember -> Bool
    C_IDCM: Callmember -> C_ID
    InitCM: Callmember -> M_ID
    M_IDCM: Callmember -> M_IDlist

eqns   forall c1, c2 : Callmember, cid, cid2: C_ID, mid, mid2: M_ID,
           midl, midl2: M_IDlist

    ofsort Bool
    cid eq cid2 => CM(cid, mid, midl) eq CM(cid2, mid2, midl2) = true;
    cid ne cid2 => CM(cid, mid, midl) eq CM(cid2, mid2, midl2) = false;
    c1 ne c2 = not( c1 eq c2);
```



```
    ofsort C_ID
    C_IDCM(CM(cid, mid, midl)) = cid;

    ofsort M_ID
    InitCM(CM(cid, mid, midl)) = mid;

    ofsort M_IDlist
    M_IDCM(CM(cid, mid, midl)) = midl

endtype (* type Callmembertype *)

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

(* type Geomembertype, record of the Goegraphical area and the Member ID *)

type Geomembertype is GeoZone, M_IDlist
sorts Geomember

opns
    ZS: GeoZone, M_IDlist -> Geomember
    _eq_, _ne_ : Geomember, Geomember -> Bool
    GeoZS: Geomember -> GeoZone
    M_IDZS: Geomember -> M_IDlist

eqns   forall g1, g2 : Geomember, zg, zg2: GeoZone, midl, midl2: M_IDlist

    ofsort Bool
    zg eq zg2 => ZS(zg, midl) eq ZS(zg2, midl2) = true;
    zg ne zg2 => ZS(zg, midl) eq ZS(zg2, midl2) = false;
    g1 ne g2 = not( g1 eq g2);

    ofsort GeoZone
    GeoZS(ZS(zg, midl)) = zg;

    ofsort M_IDlist
    M_IDZS(ZS(zg, midl)) = midl

endtype (* type Geomembertype *)

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

(* type GroupCalltype, record of IMGI and C_ID *)

type GroupCalltype is IMGI, C_IDlist
sorts GroupCall

opns
    GC: IMGI, C_IDlist -> GroupCall
    _eq_, _ne_ : GroupCall, GroupCall -> Bool
    IMGIGC: GroupCall -> IMGI
    C_IDGC: GroupCall -> C_IDlist

eqns   forall g1, g2 : GroupCall, imgi, imgi2: IMGI, cidl, cidl2: C_IDlist

    ofsort Bool
    imgi eq imgi2 => GC(imgi, cidl) eq GC(imgi2, cidl2) = true;
    imgi ne imgi2 => GC(imgi, cidl) eq GC(imgi2, cidl2) = false;
```

```
      g1 ne g2 = not( g1 eq g2);

      ofsort IMGI
      IMGIGC(GC(imgi, cidl)) = imgi;

      ofsort C_IDlist
      C_IDGC(GC(imgi, cidl)) = cidl

 endtype (* type GroupCalltype *)

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

(* type Callinfotype, record of information about a call *)

 type Callinfotype is C_ID, DTM, QoS, GeoZone, Join_leave, Send_to_all, Call_not
 sorts Callinfo

 opns
      CI: C_ID, DTM, QoS, GeoZone, Join_leave, Send_to_all, Call_not -> Callinfo
      _eq_, _ne_ : Callinfo, Callinfo -> Bool
      C_IDCI: Callinfo -> C_ID
      DTMCI: Callinfo -> DTM
      QoSCI: Callinfo -> QoS
      GeoCI: Callinfo -> GeoZone
      JoiCI: Callinfo -> Join_leave
      SenCI: Callinfo -> Send_to_all
      CalCI: Callinfo -> Call_not

 eqns   forall c1, c2 : Callinfo, cid, cid2: C_ID, dtm, dtm2: DTM,
                  qos, qos2: QoS, geo, geo2: GeoZone,
                  joi, joi2: Join_leave, sen, sen2: Send_to_all,
                  cal, cal2: Call_not

      ofsort Bool
      cid eq cid2 =>
       CI(cid, dtm, qos, geo, joi, sen, cal) eq CI(cid2, dtm2, qos2, geo2, joi2, sen2, cal2) = true;
      cid ne cid2 =>
       CI(cid, dtm, qos, geo, joi, sen, cal) eq CI(cid2, dtm2, qos2, geo2, joi2, sen2, cal2) = false;
      c1 ne c2 = not( c1 eq c2);

      ofsort C_ID
      C_IDCI(CI(cid, dtm, qos, geo, joi, sen, cal)) = cid;

      ofsort DTM
      DTMCI(CI(cid, dtm, qos, geo, joi, sen, cal)) = dtm;

      ofsort QoS
      QoSCI(CI(cid, dtm, qos, geo, joi, sen, cal)) = qos;

      ofsort GeoZone
      GeoCI(CI(cid, dtm, qos, geo, joi, sen, cal)) = geo;

      ofsort Join_leave
      JoiCI(CI(cid, dtm, qos, geo, joi, sen, cal)) = joi;

      ofsort Send_to_all
      SenCI(CI(cid, dtm, qos, geo, joi, sen, cal)) = sen;
```



```
      ofsort Call_not
      CalCI(CI(cid, dtm, qos, geo, joi, sen, cal)) = cal;

 endtype (* type Callinfotype *)

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

(* type Statusmemberlist0, set of Statusmembertype *)

 type Statusmemberlist0 is Set
 actualizedby Statusmembertype using
 sortnames
      Statusmember for Element
      Bool for FBool

 endtype (* type Stautsmemberlist0 *)

(* type Statusmemberlist1 renaming of  Statusmemberlist0 *)

 type Statusmemberlist1 is Statusmemberlist0 renamedby
 sortnames Statusmemberlist for Set

 endtype (* type Statusmemberlist1 *)

(* type Statusmemberlist, Statusmemberlist1 + some operations *)
(* provides the DBSM database *)

 type Statusmemberlist is Statusmemberlist1

 opns   RowSM: M_ID, Statusmemberlist -> Statusmember

 eqns   forall mid, mid2: M_ID, mem: Membertype, imgil: IMGIlist, cidl: C_IDlist,
                      att: Attach, stml: Statusmemberlist

      ofsort Statusmember
      mid eq mid2 => RowSM(mid, Insert(SM(mid2, mem, imgil,cidl , att), stml)) = SM(mid2, mem, imgil, cidl, att);
      mid ne mid2 => RowSM(mid, Insert(SM(mid2, mem, imgil,cidl , att), stml)) = RowSM(mid, stml);
      RowSM(mid, {}) = SM(NoM_ID, Nomember, {} of IMGIlist, {} of C_IDlist, Detached);

 endtype (* type Statusmemberlist *)

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

(* type Callmemberlist0, set of Callmembertype *)

 type Callmemberlist0 is Set
 actualizedby Callmembertype using
 sortnames
      Callmember for Element
      Bool for FBool

 endtype (* type Callmemberlist0 *)

(* type Callmemberlist1 renaming of Callmemberlist0 *)

 type Callmemberlist1 is Callmemberlist0 renamedby
```



```
 sortnames Callmemberlist for Set

 endtype (* type Callmemberlist1 *)

(* type Callmemberlist, Callmemberlist1 + some operations *)
(* provides the DBCM database *)

 type Callmemberlist is Callmemberlist1

 opns   RowCM: C_ID, Callmemberlist -> Callmember
      C_IDLCM: M_ID, Callmemberlist -> C_IDlist

 eqns   forall cid, cid2: C_ID, cidl1, cidl2: C_IDlist, mid: M_ID,
            midl: M_IDlist, cm: Callmember, cml: Callmemberlist

      ofsort Callmember
      RowCM(cid, {} of Callmemberlist) = CM(NoC_ID, NoM_ID, {} of M_IDlist);
      cid eq cid2 => RowCM(cid, Insert(CM(cid2, mid, midl), cml)) = CM(cid2, mid,  midl);
      cid ne cid2 => RowCM(cid, Insert(CM(cid2, mid, midl), cml)) = RowCM(cid, cml);

      ofsort C_IDlist
      C_IDLCM(mid, {} of Callmemberlist) = {} of C_IDlist;
      mid Isin M_IDCM(cm) => C_IDLCM(mid, Insert(cm, cml)) = Insert(C_IDCM(cm), C_IDLCM(mid, cml));
      mid Notin M_IDCM(cm) => C_IDLCM(mid, Insert(cm, cml)) = C_IDLCM(mid, cml)

 endtype (* type Callmemberlist *)

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

(* type Geomemberlist0, set of Geomembertype *)

 type Geomemberlist0 is Set
 actualizedby Geomembertype using
 sortnames
      Geomember for Element
      Bool for FBool

 endtype (* type Geomemberlist0 *)

(* type Geomemberlist1 renaming of Geomemberlist0 *)

 type Geomemberlist1 is Geomemberlist0 renamedby
 sortnames Geomemberlist for Set

 endtype (* type Geomemberlist1 *)

(* type Geomemberlist, Geomemberlist1 + some operations *)
(* provides the DBZS database *)

 type Geomemberlist is Geomemberlist1

 opns   RowZS: GeoZone, Geomemberlist -> Geomember
      GeoZS: M_ID, Geomemberlist -> GeoZone

 eqns   forall zg, zg2: GeoZone, mid: M_ID, midl: M_IDlist, gm: Geomember,
            gml: Geomemberlist
```



```
      ofsort Geomember
      RowZS(zg, {} of Geomemberlist) = ZS(NoGeozone, {} of M_IDlist);
      zg eq zg2 => RowZS(zg, Insert(ZS(zg2, midl), gml)) = ZS(zg2, midl);
      zg ne zg2 => RowZS(zg, Insert(ZS(zg2, midl), gml)) = RowZS(zg, gml);

      ofsort GeoZone
      GeoZS(mid, {} of Geomemberlist) = NoGeozone;
      mid Isin M_IDZS(gm) => GeoZS(mid, Insert(gm, gml)) = GeoZS(gm);
      mid Notin M_IDZS(gm) => GeoZS(mid, Insert(gm, gml)) = GeoZS(mid, gml)


 endtype (* type Geomemberlist *)

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

(* type GroupCalllist0, set of GroupCalltype *)

 type GroupCalllist0 is Set
 actualizedby GroupCalltype using
 sortnames
      GroupCall for Element
      Bool for FBool

 endtype (* type GroupCalllist0 *)

(* type GroupCalllist1 remaning of GroupCalllist0 *)

 type GroupCalllist1 is GroupCalllist0 renamedby
 sortnames GroupCalllist for Set

 endtype (* type GroupCalllist1 *)

(* type GroupCalllist, groupCalllist1 + some operations *)
(* provides the DBGC database *)

 type GroupCalllist is GroupCalllist1

 opns   RowGC: IMGI, GroupCalllist -> GroupCall
      IMGC: C_ID, Groupcalllist -> IMGI

 eqns   forall imgi, imgi2: IMGI, cid: C_ID, cidl: C_IDlist, gc: Groupcall,
            gcl: GroupCalllist

      ofsort GroupCall
      RowGC(imgi, {} of GroupCalllist) = GC(NoIMGI, {} of C_IDlist);
      imgi eq imgi2 => RowGC(imgi, Insert(GC(imgi2, cidl), gcl)) = GC(imgi2, cidl);
      imgi ne imgi2 => RowGC(imgi, Insert(GC(imgi2, cidl), gcl)) = RowGC(imgi, gcl);

      ofsort IMGI
      IMGC(cid, {} of Groupcalllist) = NoIMGI;
      cid Isin C_IDGC(gc) => IMGC(cid, Insert(gc, gcl)) = IMGIGC(gc);
      cid Notin C_IDGC(gc) => IMGC(cid, Insert(gc, gcl)) = IMGC(cid, gcl)

 endtype (* type groupCalllist *)

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

```
(* type Callinfolist0, set of Callinfotype *)

 type Callinfolist0 is Set
 actualizedby Callinfotype using
 sortnames
       Callinfo for Element
       Bool for FBool

 endtype (* type Callinfolist0 *)

(* type Callinfolist1 renaming of Callinfolist0 *)

 type Callinfolist1 is Callinfolist0 renamedby
 sortnames Callinfolist for Set

 endtype (* type Callinfolist1 *)

(* type Callinfolist, Callinfolist1 + some operations *)
(* provides the DBCI database *)

 type Callinfolist is Callinfolist1

 opns   RowCI: C_ID, Callinfolist -> Callinfo

 eqns   forall cid, cid2: C_ID, dtm: DTM, qos: QoS, zg: GeoZone,
        joi: Join_leave, sen: Send_to_all, cal: Call_not, cil: Callinfolist

       ofsort Callinfo
       RowCI(cid, {} of Callinfolist) = CI(NoC_ID, noway, qos1, NoGeozone, No_join_leave_ind, No_send_to_all, No_call_not);
       cid eq cid2 =>
             RowCI(cid, Insert(CI(cid2, dtm, qos, zg, joi, sen, cal), cil)) = CI(cid2, dtm, qos, zg, joi, sen, cal);
       cid ne cid2 =>
             RowCI(cid, Insert(CI(cid2, dtm, qos, zg, joi, sen, cal), cil)) = RowCI(cid, cil);

 endtype (* type Callinfolist *)

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

(* type Msgtype, this type provide the codage and uncodage of informations sent*)

 type Msgtype is GeoZone, IMGIlist, C_IDlist, DTM, QoS, Join_leave, Send_to_all, Call_not, M_IDlist, Statusinfolist
 sorts Msg

 opns
          (*================================================================*)
          (*           These messages are for simulating the behaviour        *)
          (*    Cause:      raison of rejection or disconnection              *)
          (*    Data:       the user's data                                   *)
          (*    Cipher_key: operator to code the key                          *)
          (*================================================================*)

       Nomsg, Cause, Data: -> msg
       (* Cause and Data modelize two parameters foreseen to be operators *)
       (* with the real cause or the real datas in parameters *)
       Cipher_key: C_ID -> Msg
       h: Msg -> Nat
       _eq_, _ne_ : Msg, Msg -> Bool
```



```
       (*================================================================*)
       (*      Packet for initiate scenario                                *)
       (*================================================================*)

Encode: IMGI, DTM, QoS, GeoZone, Join_leave, Send_to_all, Call_not ->Msg

GetIMGI: Msg -> IMGI
GetDTM: Msg -> DTM
GetQoS: Msg -> QoS
GetGeo: Msg -> GeoZone
GetJoin: Msg -> Join_leave
GetSend : Msg -> Send_to_all
GetCallnot: Msg -> Call_not

Encode: IMGI, C_ID, Msg -> Msg
GetCipher: Msg -> Msg


       (*================================================================*)
       (*      Packet to provide rejection                                 *)
       (*================================================================*)

Encode : Msg -> Msg
GetCause: Msg -> Msg

Encode : IMGI, Msg -> Msg

       (*================================================================*)
       (*        Packet for attach scenario                                *)
       (*================================================================*)

Encode : GeoZone -> Msg

       (*================================================================*)
       (*     Packet for detach scenario                                   *)
       (*================================================================*)

Encode : C_ID, Msg -> Msg
GetC_ID : Msg -> C_ID

Encode : M_ID, C_ID -> Msg
GetM_ID: Msg -> M_ID

       (*================================================================*)
       (*    Packet for terminate scenario                                 *)
       (*================================================================*)

Encode : IMGI, C_ID -> Msg

       (*================================================================*)
       (*   Packet for Status scenario                                     *)
       (*================================================================*)

Encode : IMGI, C_ID, Statusinfolist -> Msg
GetStatusinfo : Msg -> Statusinfolist
```



```
       Encode: IMGI, C_ID, Statusinfolist, M_IDlist, Nat -> Msg

       Encode : IMGI, C_IDlist -> Msg
       GetC_IDlist: Msg -> C_IDlist

       Encode : IMGI, C_ID, M_IDlist, Nat, Join_leave, DTM, QoS, GeoZone -> Msg
       GetNbmem: Msg -> Nat
       GetM_IDlist : Msg -> M_IDlist

          (*================================================================*)
          (*    Packet for data transfer scenario                             *)
          (*================================================================*)

       Encode : M_ID, C_ID, Msg -> Msg
       GetData: Msg -> Msg

          (*================================================================*)
          (*   Packet for Change zone scenario                                *)
          (*================================================================*)

       Encode : IMGI, GeoZone -> Msg

 eqns
       forall  msg1, msg2 : Msg,
             mid : M_ID, midlist: M_IDlist,
             cid : C_ID, cidlist : C_IDlist,
             imgi : IMGI, imgilist : IMGIlist,
             dtm : DTM, qos : QoS, join : Join_leave,
             send : Send_to_all, geo : Geozone,
             callnot : Call_not, n : Nat,
             statusinfolist: Statusinfolist

       ofsort Nat
             h(Nomsg) = 0;
             h(Cause) =Succ(h(Nomsg));
             h(Data) = Succ(h(Cause));

             GetNbmem(Encode(imgi, cid, midlist, n, join, dtm, qos, geo )) = n;
             GetNbmem(Encode(imgi, cid, statusinfolist, midlist, n)) = n;

       ofsort Bool
             msg1 eq msg2 = h(msg1) eq h(msg2);
             msg1 ne msg2 = not (msg1 eq msg2);

       ofsort M_ID
             GetM_ID(Encode(mid, cid)) = mid;
             GetM_ID(Encode(mid,cid,msg1)) = mid;

       ofsort M_IDlist
             GetM_IDlist(Encode(imgi, cid, midlist, n, join, dtm, qos, geo)) = midlist;
             GetM_IDlist(Encode(imgi, cid, statusinfolist, midlist, n)) = midlist;

       ofsort IMGI
             GetIMGI(Encode(imgi, dtm, qos, geo, join, send, callnot)) = imgi;
             GetIMGI(Encode(imgi, cid,msg1)) = imgi;
             GetIMGI(Encode(imgi,msg1)) = imgi;
             GetIMGI(Encode(imgi, cid)) = imgi;
```



```
             GetIMGI(Encode(imgi,cid, statusinfolist)) = imgi;
             GetIMGI(Encode(imgi, cid, statusinfolist, midlist, n)) = imgi;
             GetIMGI(Encode(imgi,cidlist)) = imgi;
             GetIMGI(Encode(imgi, cid, midlist, n, join, dtm, qos, geo)) = imgi;
             GetIMGI(Encode(imgi, geo)) = imgi;

       ofsort C_ID
             GetC_ID(Encode(imgi, cid, msg1)) = cid;
             GetC_ID(Encode(cid, msg1)) = cid;
             GetC_ID(Encode(mid, cid)) = cid;
             GetC_ID(Encode(imgi, cid)) = cid;
             GetC_ID(Encode(imgi, cid, statusinfolist)) = cid;
             GetC_ID(Encode(imgi, cid, statusinfolist, midlist, n)) = cid;
             GetC_ID(Encode(imgi, cid, midlist, n, join, dtm, qos, geo)) = cid;
             GetC_ID(Encode(mid, cid, msg1)) = cid;

       ofsort DTM
             GetDTM(Encode(imgi, dtm, qos, geo, join, send, callnot)) = dtm;
             GetDTM(Encode(imgi, cid, midlist, n, join, dtm, qos, geo)) = dtm;

       ofsort QoS
             GetQoS(Encode(imgi, dtm, qos, geo, join, send, callnot)) = qos;
             GetQoS(Encode(imgi, cid, midlist, n, join, dtm, qos, geo)) = qos;

       ofsort GeoZone
             GetGeo(Encode(imgi, dtm, qos, geo, join, send, callnot)) = geo;
             GetGeo(Encode(geo)) = geo;
             GetGeo(Encode(imgi, cid, midlist, n, join, dtm, qos, geo)) = geo;
             GetGeo(Encode(imgi, geo)) = geo;

       ofsort Join_leave
             GetJoin(Encode(imgi, dtm, qos, geo, join, send, callnot)) = join;
             GetJoin(Encode(imgi, cid, midlist, n, join, dtm, qos, geo)) = join;

       ofsort Send_to_all
             GetSend(Encode(imgi, dtm, qos, geo, join, send, callnot)) = send;

       ofsort Call_not
             GetCallnot(Encode(imgi, dtm, qos, geo, join, send, callnot)) = callnot;

       ofsort Msg
             GetCipher(Encode(imgi, cid, Cipher_key(cid))) = Cipher_key(cid);
             GetCause(Encode(msg1)) = msg1;
             GetCause(Encode(imgi, msg1)) = msg1;
             GetCause(Encode(cid, msg1)) = msg1;
             GetData(Encode(mid, cid, msg1)) = msg1;
             GetData(Encode(imgi, cid, msg1)) = msg1;
             GetCause(Encode(imgi, cid, msg1)) = msg1;

       ofsort Statusinfolist
             GetStatusinfo(Encode(imgi, cid, statusinfolist)) = statusinfolist;
             GetStatusinfo(Encode(imgi, cid, statusinfolist, midlist, n)) = statusinfolist;

       ofsort C_IDlist
             GetC_IDlist(Encode(imgi, cidlist)) = cidlist;
```

```
endtype (* type Msgtype *)

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


(*==========================================================================*)
(*                                Behaviour                                 *)
(*==========================================================================*)

behaviour

Subscriber[ch_Uplink, ch_Downlink]

|[ch_Uplink, ch_Downlink]|

PLMN[ch_Uplink, ch_Downlink]

where

        (*==================================================================*)
        (*      process Subscriber, set of MS with their M_ID              *)
        (*==================================================================*)

    process Subscriber[ch_Uplink, ch_Downlink]: noexit:=
    (
        MS[ch_Uplink, ch_Downlink] (Adm1)
     |||
        MS[ch_Uplink, ch_Downlink] (Init1)
     |||
        MS[ch_Uplink, ch_Downlink] (Init2)
     |||
        MS[ch_Uplink, ch_Downlink] (Mem1)
     |||
        MS[ch_Uplink, ch_Downlink] (Mem2)
     |||
        MS[ch_Uplink, ch_Downlink] (Mem3)
     )
    endproc (* process Subscriber *)

        (*==================================================================*)
        (*      process MS, mobile station or eventually fixed station      *)
        (*==================================================================*)

    process MS[ch_Uplink, ch_Downlink] (n: M_ID) : noexit:=

        Requester[ch_Uplink,ch_Downlink] (n)
        |||
        Receiver[ch_Downlink](n)

    endproc  (* process MS *)

        (*==================================================================*)
        (*     process Requester, provides the request behaviour            *)
        (*==================================================================*)

    process Requester[ch_Uplink,ch_Downlink] (n: M_ID): noexit :=
```



```
        ch_Uplink!n!Req_Attach?msg: Msg ;
        (* msg= Encode(GeoZone) *)
        (
        (
            ch_Downlink!n!Ack_Attach!NoMsg;
            ( (*_PROBE_MS_*)
            RequesterAttached[ch_Uplink, ch_Downlink] (n, {} of C_IDlist)
            [>
            (
                ch_Uplink!n!Req_Detach!Nomsg;
                ch_Downlink!n!Ack_Detach!NoMsg; (*_PROBE_MS_*)
                Requester[ch_Uplink, ch_Downlink](n)
            )
            )
        )
        []
        (
            ch_Downlink!n!Err_Rejatt?msg: Msg;
            (* msg= Encode(Cause) *) (*_PROBE_MS_*)
            Requester[ch_Uplink,ch_Downlink] (n)
        )
        )

    endproc (* process Requester *)

        (*==================================================================*)
        (*    process RequesterAttached, provides the behaviour of a        *)
        (*                   station when it is attached                    *)
        (*==================================================================*)

    process RequesterAttached[ch_Uplink, ch_Downlink](n: M_ID, cidl: C_IDlist): noexit :=
    (
    (       (* request of initiation of a call *)
        ch_Uplink!n!Req_Init?msg:Msg[GetDTM(msg) ne Noway];
        (* msg = Encode(IMGI, DTM, QoS, GeoZone, Join_leave, Send_to_all, Call_not) *)
        (
        (
            ch_Downlink!n!Err_Rejinit?msg: Msg;
            (* msg= Encode( IMGI, Cause) *) (*_PROBE_MS_*)
            RequesterAttached[ch_Uplink, ch_Downlink](n, cidl)
        )
        []
        (
            ch_Downlink!n!Ack_C_Init?msg: Msg;
            (* msg = Encode(IMGI, C_ID, Cipher_key(C_ID)) *)
            (
            (
                let cid: C_ID = GetC_ID(msg) in
                ( (*_PROBE_MS_*)
                RequesterInitiator[ch_Uplink, ch_Downlink](n, GetIMGI(msg), cid)
                [>
                 (
                    ch_Downlink!n!Ind_Term?message: Msg[GetC_ID(message) eq cid];
                    (* message = Encode(cid, Cause) *) (*_PROBE_MS_*)
                    exit
                )
                )>>  (*_PROBE_MS_*)stop
```



```
                )
                |||
                    RequesterAttached[ch_Uplink, ch_Downlink](n, Insert(GetC_ID(msg),cidl))
                )
            )
            )
        )
        []
        (       (* request status *)
            ch_Uplink!n!Req_Status?msg: Msg;
            (* msg =Encode(IMGI, Statusinfolist) *)
            (
            (
                ch_Downlink!n!Ack_Status?msg2: Msg;
                (
                [Sta_callids Isin GetStatusinfo(msg)] -> (*_PROBE_MS_*)
                   RequesterAttached[ch_Uplink, ch_Downlink](n, GetC_IDlist(msg2) Union cidl)
                []
                [Sta_callids Notin GetStatusinfo(msg)] -> (*_PROBE_MS_*) RequesterAttached[ch_Uplink, ch_Downlink](n, cidl)
                )
            )
            []
            (
                ch_Downlink!n!Err_rejstat?msg: Msg;
                (* msg= Encode( IMGI, Cause) *) (*_PROBE_MS_*)
                RequesterAttached[ch_Uplink, ch_Downlink](n, cidl)
            )
            )
        )
        []
        (       (* request join a call *)
            ch_Uplink!n!Req_Join?msg: Msg [(GetC_ID(msg)) Isin cidl];
            (* msg= Encode(IMGI, C_ID) *)
            (
            (
                ch_Downlink!n!Ack_Join?msg: Msg;
                (* msg = Encode(IMGI, C_ID, Cipher_key(C_ID)) *)
                (
                    let cid: C_ID = GetC_ID(msg) in
                    ( (*_PROBE_MS_*)
                    RequesterMember[ch_Uplink, ch_Downlink](n, GetIMGI(msg), cid)
                    [>
                    (
                    (
                        ch_Downlink!n!Ind_Term?message: Msg[GetC_ID(message) eq cid];
                        (* message = Encode(cid, Cause) *) (*_PROBE_MS_*)
                        exit
                    )
                    []
                    (
                        ch_Downlink!n!Ind_Leave!Encode(n, cid); (*_PROBE_MS_*)
                        exit
                    )
                    )
                    )>>  (*_PROBE_MS_*)stop
                )
                |||
```



```
            RequesterAttached[ch_Uplink, ch_Downlink](n, cidl)
        )
        []
        (
            ch_Downlink!n!err_Rejjoin?msg2: Msg;
            (* msg = encode(IMGI, C_ID, Cause) *)
            ( (*_PROBE_MS_*)
            RequesterAttached[ch_Uplink, ch_Downlink](n, Remove(GetC_ID(msg2), cidl))
            )
        )
        )
    )
    []
    (       (* request change zone *)
        ch_Uplink!n!Req_Change?msg: Msg;
        (* msg = Encode(GeoZone) *)
        ch_Downlink!n!Ack_Change!NoMsg; (*_PROBE_MS_*)
        RequesterAttached[ch_Uplink, ch_Downlink](n, cidl)
    )
    )

    endproc (* process RequesterAttached *)

        (*==================================================================*)
        (*         process RequesterInitiator, provides the behaviour of an *)
        (*                             initiator                            *)
        (*==================================================================*)

    process RequesterInitiator[ch_Uplink, ch_Downlink](n: M_ID, imgi: IMGI, cid: C_ID): exit:=

    (       (* request termination of the current initiated call *)
        ch_Uplink!n!Req_Term!Encode(imgi, cid);
        (
        (
            ch_Downlink!n!Ack_C_Term!Encode(imgi, cid); (*_PROBE_MS_*)
            exit
        )
        []
        (       (* should not happen *)(* network overload or something like that *)
            ch_Downlink!n!Err_Rejterm?message: Msg[(GetC_ID(message) eq cid) and (GetIMGI(message) eq imgi)];
            (* message = Encode(imgi, cid, Cause) *) (*_PROBE_MS_*)
            RequesterInitiator[ch_Uplink, ch_Downlink](n, imgi, cid)
        )
        )
    )
    []
    (       (* request send data *)
        ch_Uplink!n!Req_Data?msg: Msg;
        (* msg =Encode(imgi, cid, Data) *)
        (
        (       (* should not happen *)(* network overload or something like that *)
            ch_Downlink!n!Err_Rejdat!Encode(imgi, cid, Cause);  (*_PROBE_MS_*)exit
        )
        []
        (
            ch_Downlink!n!Ack_SAll!Encode(imgi, cid); (*_PROBE_MS_*) exit
        )
```

```
	) >> (*_PROBE_MS_*)
	RequesterInitiator[ch_Uplink, ch_Downlink](n, imgi, cid)
)
endproc   (* process RequesterInitiator *)

	(*===================================================================*)
	(*         process RequesterMember, provides the behaviour of a       *)
	(*                          participant                               *)
	(*===================================================================*)

process RequesterMember[ch_Uplink, ch_Downlink](n: M_ID, imgi: IMGI, cid: C_ID): exit:=

(	(* request leave call *)
	ch_Uplink!n!Req_Leave!Encode(imgi, cid);
	(
	(
		ch_Downlink!n!Err_Rejleave?message: Msg[(GetC_ID(message) eq cid) and (GetIMGI(message) eq imgi)];
		(* message = Encode(imgi, cid, Cause) *) (*_PROBE_MS_*)
		RequesterMember[ch_Uplink, ch_Downlink](n, imgi, cid)
	)
	[]
	(
		ch_Downlink!n!Ack_Leave!Encode(imgi, cid); (*_PROBE_MS_*)
		exit
	)
	)

)
[]
(	(* request send data *)
	ch_Uplink!n!Req_Data!Encode(imgi, cid, Data);
	(
	(	(* should not happen *)(* network overload or something like that *)
		ch_Downlink!n!Err_Rejdat!Encode(imgi, cid, Cause); (*_PROBE_MS_*) exit
	)
	[]
	(
		ch_Downlink!n!Ack_SAll!Encode(imgi, cid);  (*_PROBE_MS_*) exit
	)
	[]
	(
		ch_Downlink!n!Ack_SInit!Encode(imgi, cid); (*_PROBE_MS_*) exit
	)
	[]
	(
		ch_Downlink!n!Err_Wmode?message: Msg[(GetC_ID(message) eq cid) and (GetIMGI(message) eq imgi)];
		(* message = Encode(imgi, cid, Cause) *) (*_PROBE_MS_*)
		exit
	)
	) >> (*_PROBE_MS_*)
	RequesterMember[ch_Uplink, ch_Downlink](n, imgi, cid)
)
endproc  (* process RequesterMember *)

	(*===================================================================*)
	(*    process Receiver, passive behaviour of the station,             *)
	(*                    attending packets of indication                 *)
```



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

process Receiver[ch_Downlink](n: M_ID) : noexit :=

(
(	(* receiving of general indication *)
	ch_Downlink!n?ind: Ind?msg: Msg [((ind ne Ind_Term) and (ind ne Ind_Leave))]; (*_PROBE_MS_*)
	receiver[ch_Downlink](n)
)
[]
(	(* receiving leave indication, for others *)
	ch_Downlink!n!Ind_leave?msg: Msg [GetM_ID(msg) ne n]; (*_PROBE_MS_*)
	receiver[ch_Downlink](n)
)
)
endproc  (* process Receiver *)

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

	(*===================================================================*)
	(*            process PLMN, network supporting GPRS                   *)
	(*===================================================================*)

process PLMN[ch_Uplink, ch_Downlink]: noexit :=
hide ch_Answer, ch_Sending, ch_ID in
(
	Senders_team[ch_Downlink, ch_Sending]
	|[ch_Sending]|
	Controller_team[ch_Uplink, ch_Answer, ch_Sending, ch_ID]
	|[ch_Answer]|
	Answer_team[ch_Downlink, ch_Answer]
)
endproc (* process PLMN *)

	(*===================================================================*)
	(*           process Controller_team, instantiate the controller_team  *)
	(*===================================================================*)


process Controller_team[ch_Uplink, ch_Answer, ch_Sending, ch_ID]: noexit:=

(
	Ident[ch_ID](Succ(NoC_ID))
	|[ch_ID]|
	Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](Insert(ZS(1 of Geozone,{} of M_IDlist),{} of Geomemberlist),
			{} of Callmemberlist,
			Insert(SM(Adm1, cont, Insert(1, Insert(2, Insert(3, {} of IMGIlist))), {} of C_IDlist, Detached),
			Insert(SM(Mem2, part, Insert(1, {} of IMGIlist), {} of C_IDlist, Detached),
			Insert(SM(Init1, init, Insert(2, Insert(1, {} of IMGIlist)), {} of C_IDlist, Detached),
			Insert(SM(Mem1, part, insert(2, Insert(1, {} of IMGIlist)), {} of C_IDlist, Detached),
			{} of Statusmemberlist)))))

   (* DBZS should be filled with the list of GeoZone and {} as list of M_ID, *)
   (* DBCM is empty, DBSM should be filled with all the members known *)
)
endproc   (* process Controller_team *)
```



```
	(*===================================================================*)
	(*            process Ident, provides the Calls IDs                   *)
	(*===================================================================*)

process Ident[ch_ID](int: C_ID): noexit:=

	ch_ID!int;
	Ident[ch_ID](Succ(int))
endproc  (* process Ident *)

	(*===================================================================*)
	(*     process Controller, THE controller of the network              *)
	(*===================================================================*)

process Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
			DBZS: Geomemberlist, DBCM: Callmemberlist,
			DBSM: Statusmemberlist): noexit:=

(
	ch_Uplink ?n: M_ID ?req: Req ?msg: Msg;
	(
		[req eq Req_Attach] ->
			ContAttach[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n,
						GetGeo(msg))
	[]
		[req eq Req_Detach] ->
			ContDetach[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n)
	[]
		[req eq Req_Init] ->
			ContInit[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n, GetIMGI(msg),
				GetDTM(msg), GetQoS(msg), GetGeo(msg),
				GetJoin(msg), GetSend(msg),
				GetCallnot(msg))
	[]
		[req eq Req_Term] ->
			ContTerm[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n, GetC_ID(msg),
				GetIMGI(msg))
	[]
		[req eq Req_Status] ->
			ContStatus[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n, GetIMGI(msg),
				GetC_ID(msg), GetStatusinfo(msg))
	[]
		[req eq Req_Join] ->
			ContJoin[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n, GetIMGI(msg),
				GetC_ID(msg))
	[]
		[req eq Req_Leave] ->
			ContLeave[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n, GetC_ID(msg),
				GetIMGI(msg))
	[]
```



```
		[req eq Req_Data] ->
			ContData[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n, GetIMGI(msg),
				GetC_ID(msg), GetData(msg))
	[]
		[req eq Req_Change] ->
			ContChange[ch_Uplink, ch_Answer, ch_Sending,
				ch_ID](DBZS, DBCM, DBSM, n, GetGeo(msg))
	)
)
endproc (* process Controller *)

	(*===================================================================*)
	(*    process ContAttach, provides the attachment to the GPRS service  *)
	(*                         Controller point of view                   *)
	(*===================================================================*)

process ContAttach[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
		DBZS: Geomemberlist, DBCM: Callmemberlist,
		DBSM: Statusmemberlist, n: M_ID, zg: Geozone):noexit:=
(
	let S: Statusmember = RowSM(n, DBSM) in
	(
		[M_IDSta(S) eq n] -> (* requester is registered in the database *)
		(
			let Z: Geomember = RowZS(zg, DBZS) in
			(
			ch_Answer!n!Ack_Attach!Nomsg; (*_PROBE_PLMN_*)
			Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
				Insert(ZS(zg, Insert(n, M_IDZS(Z))),
				Remove(Z,DBZS)), DBCM,
				Insert(SM(n, MemSta(S), IMGISta(S),
				C_IDSta(S),  Attached), Remove(S, DBSM)))
			)
		)
		[]
		[M_IDSta(S) ne n] -> (* requester not registered, M_IDSta(S)=NoM_ID *)
			ch_Answer!n!Err_rejatt!Encode(Cause); (*_PROBE_PLMN_*)
			Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
				DBZS, DBCM, DBSM)
	)
)
endproc  (* process ContAttach *)

	(*===================================================================*)
	(*    process ContDetach, provides the detachment service             *)
	(*                         Controller point of view                   *)
	(*===================================================================*)

process ContDetach[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
			DBZS: Geomemberlist, DBCM: Callmemberlist,
			DBSM: Statusmemberlist, n: M_ID): noexit :=
(
	let S: Statusmember = RowSM(n, DBSM) in
	(
	  let Z: GeoZone = GeoZS(n, DBZS) in
	  (
```

```
[Z eq NoGeozone] -> (* should not happen, requester not attached *)
(
      ch_Answer!n!Ack_Detach!NoMsg; (*_PROBE_PLMN_*)
      Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
            DBZS, DBCM, DBSM)
)
[]
[Z ne NoGeozone] -> (* requester normally attached *)
(
  let C: C_IDlist = C_IDLCM(n, DBCM) in
  let X: Geomember = RowZS(Z,DBZS) in
  (
  ch_Answer!n!Ack_Detach!NoMsg;
  (
   [C eq {} of C_IDlist] -> (* requester not member of any call *) (*_PROBE_PLMN_*)
      Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
              Insert(ZS(Z,Remove(n, M_IDZS(X))),Remove(X,DBZS)),
              DBCM,
              Insert(SM(n, MemSta(S), IMGISta(S),
                   C_IDSta(S), Detached),
                   Remove(S, DBSM)))
   []
   [C ne {} of C_IDlist] -> (* requester member of some call *)
   (
      [((MemSta(S) eq init) or (MemSta(S) eq cont)) and (C_IDSta(S) ne {})] ->
      (* requester initiator of active calls *)
             ContMICTerm[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
                  Insert(ZS(Z, Remove(n, M_IDZS(X))), Remove(X, DBZS)),
                  DBCM,
                  Insert(SM(n, MemSta(S), IMGISta(S), {} of C_IDlist, Detached), Remove(S, DBSM)),
                  C_IDSta(S), n)
      []
      [((MemSta(S) ne init) and (MemSta(S) ne cont)) or (C_IDSta(S) eq {})] ->
      (* requester participant of some active calls *) (*_PROBE_PLMN_*)
      ContMIMLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
            Insert(ZS(Z, Remove(n, M_IDZS(X))), Remove(X, DBZS)),
            DBCM,
            Insert(SM(n, MemSta(S), IMGISta(S), {} of C_IDlist, Detached), Remove(S, DBSM)), C, n)
   )
  )
  )
 )
 )
)
)
where

      (*=====================================================================*)
      (*    process ContMICTerm, provides the updating of the database DBCM  *)
      (*    when the initiator termitates the call                           *)
      (*       Controller point of view                                      *)
      (*=====================================================================*)

process ContMICTerm[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                        DBSM: Statusmemberlist, cidl: C_IDlist, n: M_ID): noexit:=
(
      let T: C_ID = Top(cidl) in
```



```
      (
      [T eq NoElement of C_ID] -> (* no more call to terminate *) (*_PROBE_PLMN_*)
            ContMIMLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM, C_IDLCM(n, DBCM), n)
      []
      [T ne NoElement of C_ID] -> (* call to terminate *)
      (
            let C: Callmember = RowCM(T, DBCM) in
            (
            ch_Answer!Delete!T!NoMsg;
            ch_Answer!Ask!NoIMGI of IMGI!T;
            ch_Answer?imgi: IMGI!T;
            ch_Answer!Delete!imgi!T;
            ContSend[ch_Sending](Remove(n, M_IDCM(C)), Ind_Term, Encode(T, Cause))
            >>  (*_PROBE_PLMN_*)
            ContMICTerm[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS,
                  Remove(C, DBCM), DBSM, Remove(T, cidl),n)
            )
      )
      )
)
endproc (* process ContMICTerm *)

      (*=====================================================================*)
      (*    process ContMIMLeave, provides the updating of the database DBCM *)
      (*    when a member leaves a  call                                     *)
      (*       Controller point of view                                      *)
      (*=====================================================================*)

process ContMIMLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                        DBSM: Statusmemberlist, cidl: C_IDlist, n: M_ID): noexit:=
(
      let T: C_ID = Top(cidl) in
      (
      [T eq (NoElement of C_ID)] -> (* no more call to leave *) (*_PROBE_PLMN_*)
            Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
      []
      [T ne (Noelement of C_ID)] -> (* call to leave *)
      (
            let C: Callmember = RowCM(T, DBCM) in
            (
            ch_Answer!Ask!T!Nomsg;
            ch_Answer!T?info: Callinfo;
            (
              [JoiCI(info) eq Join_leave_ind] -> (* send leave indication *)
              (
                [SenCI(info) eq Send_to_all_ind] -> (* send leave indication to all *)
                (
                 ContSend[ch_Sending](Remove(n, M_IDCM(C)), Ind_Leave, Encode(n, T))
                 >> (*_PROBE_PLMN_*)
                 ContMIMLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS,
                       Insert(CM(T, InitCM(C), Remove(n, M_IDCM(C))), Remove(C, DBCM)), DBSM, Remove(T, cidl),n)
                )
                []
                [SenCI(info) ne Send_to_all_ind] -> (* send leave indication only to the initiator *)
                (
                 ContSend[ch_Sending](Insert(InitCM(C), {} of M_IDlist), Ind_Leave, Encode(n, T))
                 >> (*_PROBE_PLMN_*)
```



```
                 ContMIMLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS,
                       Insert(CM(T, InitCM(C), Remove(n, M_IDCM(C))), Remove(C, DBCM)), DBSM, Remove(T, cidl),n)
                )
              )
              []
              [JoiCI(info) ne Join_leave_ind] -> (* do not send leave indication *) (*_PROBE_PLMN_*)
                  ContMIMLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS,
                       Insert(CM(T, InitCM(C), Remove(n, M_IDCM(C))), Remove(C, DBCM)), DBSM, Remove(T, cidl),n)

            )
            )
      )
      )
)
endproc (* process ContMIMLeave *)

endproc  (* process ContDetach *)

      (*=====================================================================*)
      (*  process ContInit, provides the initiation of a call                *)
      (*                      Controller point of view                       *)
      (*=====================================================================*)


process ContInit[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
            DBZS: Geomemberlist, DBCM: Callmemberlist,
            DBSM: Statusmemberlist, n: M_ID, imgi: IMGI, dtm:DTM,
            qos: QoS, zg: GeoZone, join: Join_leave, send: Send_to_all,
            callnot: Call_not): noexit :=
(
      let S:Statusmember = RowSM(n, DBSM) in
      (
       [(MemSta(S) eq init) or (MemSta(S) eq cont)] -> (* requester have rigths to initiate a call *)
       (
            let Z:Geomember = RowZS(zg, DBZS) in
            (
            [imgi Isin IMGISta(S)] -> (* the group is valid *)
            (
              [(zg eq NoGeozone) or (n Isin M_IDZS(Z))] -> (* the geozone is valid *)
              (
                ch_ID?cid:C_ID;
                (
                ch_Answer!n!Ack_C_Init!Encode(imgi, cid, Cipher_key(cid));
                ch_Answer!Insert!cid!Encode(imgi, dtm, qos, zg, join, send, callnot);
                ch_Answer!Insert!imgi!cid;
                (
                [callnot eq Call_not_ind] -> (* send notification of cration of a call *)
                ( (*_PROBE_PLMN_*)
                      ContSend[ch_Sending](M_IDZS(Z), Ind_Init, Encode(imgi, cid))
                )
                []
                [callnot eq No_call_not] -> (* do not send the call initiation notification *)
                ( (*_PROBE_PLMN_*)
                      exit
                )
                )>> (*_PROBE_PLMN_*)
                Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
```



```
                      DBZS,
                      Insert(CM(cid, n, Insert(n, {} of M_IDlist)),DBCM),
                      Insert(SM(M_IDSta(S), MemSta(S), IMGISta(S), Insert(cid, C_IDSta(S)), AttSta(S)), Remove(S, DBSM)))
                  )
              )
              []
              [(zg ne NoGeozone) and (n Notin M_IDZS(Z))] -> (* the geozone is not valid *)
              (
                ch_Answer!n!Err_Rejinit!Encode(Cause); (*_PROBE_PLMN_*)
                Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
              )
            )
            []
            [imgi Notin IMGISta(S)] -> (* the group is not valid *)
            (
                Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
            )
            )
       )
       []
       [(MemSta(S) ne init) and (MemSta(S) ne cont)] -> (* the requester does not have rigths to initiate a call *)
            ch_Answer!n!Err_Rejinit!Encode(Cause); (*_PROBE_PLMN_*)
            Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
      )

)
endproc  (* process ContInit *)

      (*=====================================================================*)
      (*   process ContTerm, provides the termination of a call              *)
      (*                         Controller point of view                    *)
      (*=====================================================================*)

process ContTerm[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                        DBSM: Statusmemberlist, n: M_ID, cid: C_ID, imgi:IMGI): noexit :=

(
      let S: Statusmember = RowSM(n, DBSM) in
      let C: Callmember = RowCM(cid, DBCM) in
      (
      [n ne initCM(C)] -> (* should not happen *)
      (
            ch_Answer!n!Err_Rejterm!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
            Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
      )
      []
      [n eq initCM(C)] -> (* requester is initiator of the call *)
      (
            ch_Answer!n!Ack_C_Term!Encode(imgi, cid);
            ch_Answer!Delete!cid!NoMsg;
            ch_Answer!Delete!imgi!cid;
            (
            ContSend[ch_Sending](Remove(n, M_IDCM(C)), Ind_Term, Encode(cid, Cause))
            )>> (*_PROBE_PLMN_*)
            Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
                  DBZS,
```

```
                Remove(C, DBCM),
                Insert(SM(M_IDSta(S), MemSta(S), IMGISta(S), {} of C_IDlist, AttSta(S)), Remove(S, DBSM)))
        )
        )
)
endproc  (* process ContTerm  *)

        (*=====================================================================*)
        (*   process ContStatus, provides the answer to the status request     *)
        (*                      Controller point of view                       *)
        (*=====================================================================*)

process ContStatus[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                    DBSM:Statusmemberlist, n: M_ID, imgi: IMGI,
                    cid: C_ID, statlist:Statusinfolist): noexit:=
(
        let S: Statusmember = RowSM(n, DBSM) in
        (
        [imgi Isin IMGISta(S)] -> (* requester have rigths on the group *)
        (
            [(Sta_memlist Isin statlist) or (Sta_nbmem Isin statlist)] ->
            (* answer to the two requests: number of members and list of members *)
            (
                ch_Answer!n!Ack_Status!Encode(imgi, cid, statlist,
                    M_IDCM(RowCM(cid, DBCM)),
                    Card(M_IDCM(RowCM(cid, DBCM)))); (*_PROBE_PLMN_*)
                Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
            )
            []
            [(Sta_memlist Notin statlist) and (Sta_nbmem Notin statlist)] ->
            (* requester won't know the number of members and the list of members *)
            (
                ch_Answer!n!Ack_Status!Encode(imgi, cid, statlist); (*_PROBE_PLMN_*)
                Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
            )
        )
        []
        [imgi Notin IMGISta(S)] -> (*requester doesn't have rigths on the group *)
        (
            ch_Answer!n!Err_Rejstat!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
            Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
        )
        )
)
endproc (* process ContStatus *)

        (*=====================================================================*)
        (*    process ContJoin, provides the join scenario                     *)
        (*                      Controller point of view                       *)
        (*=====================================================================*)

process ContJoin[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                    DBSM: Statusmemberlist, n: M_ID, imgi: IMGI, cid: C_ID): noexit :=

(
        let S: Statusmember = RowSM(n, DBSM) in
        let C: Callmember = RowCM(cid, DBCM) in
```



```
(
  [imgi Isin IMGISta(S)] -> (* group is valid for this requester *)
  (
     ch_Answer!Ask!cid!Nomsg;
     ch_Answer!cid?info:Callinfo;
     (
     [((GeoCI(info) eq NoGeozone) or (n Isin M_IDZS(RowZS(GeoCI(info), DBZS))))and (cid eq C_IDCI(info))] ->
     (* the geozone is valid and the call exists *)
     (
          ch_Answer!n!Ack_Join!Encode(imgi, cid, Cipher_Key(cid));
          (
          [JoiCI(info) eq Join_leave_ind] -> (* send join indication *)
          (
          [SenCI(info) eq Send_to_all_ind] -> (* send join indication to all *)
          (
               ContSend[ch_Sending](M_IDCM(C), Ind_Join, Encode(n, cid))
               >>  (*_PROBE_PLMN_*)
               Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
                    DBZS, Insert(CM(cid, InitCM(C), Insert(n, M_IDCM(C))), Remove(C, DBCM)),
                    DBSM)
          )
          []
          [SenCI(info) ne Send_to_all_ind] -> (* send join indication only to the initiator *)
          (
               ContSend[ch_Sending](Insert(InitCM(C), {} of M_IDlist), Ind_Join, Encode(n, cid))
               >>  (*_PROBE_PLMN_*)
               Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
                    DBZS, Insert(CM(cid, InitCM(C), Insert(n, M_IDCM(C))), Remove(C, DBCM)),
                    DBSM)
          )
          )
          []
          [JoiCI(info) ne Join_leave_ind] -> (* do not send join indication *) (*_PROBE_PLMN_*)
               Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
               DBZS,
               Insert(CM(cid, InitCM(C), Insert(n, M_IDCM(C))), Remove(C, DBCM)),
               DBSM)
          )
     )
     []
     [((GeoCI(info) ne NoGeozone) and (n Notin M_IDZS(RowZS(GeoCI(info), DBZS)))) or (cid ne C_IDCI(info))] ->
     (* the call doesn't exist or the geozone is invalid *)
     (
          ch_Answer!n!Err_Rejjoin!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
          Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
               DBZS, DBCM, DBSM)
     )
     )
  )
  []
  [imgi Notin IMGISta(S)] -> (* the group is invalid *)(*should not happen *)
  (
     ch_Answer!n!Err_Rejjoin!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
     Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
          DBZS, DBCM, DBSM)
  )
)
```



```
)
endproc (* process ContJoin *)

        (*=====================================================================*)
        (*    process ContLeave, provides the leave scenario                   *)
        (*                      Controller point of view                       *)
        (*=====================================================================*)

process ContLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                    DBSM: Statusmemberlist, n: M_ID, cid: C_ID, imgi: IMGI):noexit :=

(
        let S: Statusmember = RowSM(n, DBSM) in
        let C: Callmember = RowCM(cid, DBCM) in
        (
          [(n Isin M_IDCM(C)) and (n ne InitCM(C))] -> (* requester is member and isn't the initiator *)
          (
             ch_Answer!Ask!cid!Nomsg;
             ch_Answer!cid?info:Callinfo;
             ch_Answer!n!Ack_Leave!Encode(imgi, cid);
             (
             [JoiCI(info) eq Join_leave_ind] -> (*send leave indication *)
             (
                  [SenCI(info) eq Send_to_all_ind] -> (* send leave indication to all *)
                  (
                  ContSend[ch_Sending](Remove(n, M_IDCM(C)), Ind_Leave, Encode(n, cid))
                  >> (*_PROBE_PLMN_*)
                  Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
                       DBZS,
                       Insert(CM(cid, InitCM(C), Remove(n, M_IDCM(C)))), Remove(C, DBCM)),
                       DBSM)
                  )
                  []
                  [SenCI(info) ne Send_to_all_ind] -> (* send leave indication only to the initiator *)
                  (
                  ContSend[ch_Sending](Insert(InitCM(C), {} of M_IDlist), Ind_Leave, Encode(n, cid))
                  >> (*_PROBE_PLMN_*)
                  Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
                       DBZS,
                       Insert(CM(cid, InitCM(C), Remove(n, M_IDCM(C)))), Remove(C, DBCM)),
                       DBSM)
                  )
             )
             []
             [JoiCI(info) ne Join_leave_ind] -> (* do not send leave indication *)
             ( (*_PROBE_PLMN_*)
                  Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
                       DBZS,
                       Insert(CM(cid, InitCM(C), Remove(n, M_IDCM(C)))), Remove(C, DBCM)),
                       DBSM)
             )
             )
          )
          []
          [(n Notin M_IDCM(C)) or (n eq InitCM(C))] -> (* requester is initiator or not member *)
          (
             ch_Answer!n!Err_Rejleave!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
```



```
             Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
          )
        )
)
endproc  (* process ContLeave *)

        (*=====================================================================*)
        (*          process ContData, provides the data transfer               *)
        (*                      Controller point of view                       *)
        (*=====================================================================*)

process ContData[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                    DBSM: Statusmemberlist, n: M_ID, imgi: IMGI, cid: C_ID, dat: Msg): noexit:=
(
        let S: Statusmember = RowSM(n, DBSM) in
        let C: Callmember = RowCM(cid, DBCM) in
        (
        [cid Isin C_IDSta(S)] -> (* requester is initiator of the call *)
             (
             ch_Answer!n!Ack_SAll!Encode(imgi, cid);
             ContSend[ch_Sending](M_IDCM(C), Ind_Data, Encode(n, cid, dat))
             >>  (*_PROBE_PLMN_*)
             Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
             )
        []
        [cid Notin C_IDSta(S)] ->
           (
           [(imgi Isin IMGISta(S)) and (n Isin M_IDCM(C))] ->
           (* group is valid and requester is participant to the call *)
             (
             ch_Answer!Ask!cid!NoMsg;
             ch_Answer!cid?ci: Callinfo;
             (
             [DTMCI(ci) eq one_way] -> (* initiator can only send datas *)
                  (
                  ch_Answer!n!Err_Wmode!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
                  Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
                  )
             []
             [DTMCI(ci) eq two_way] -> (* initiator is a moderator, all datas must be send to him before diffusion *)
                  (
                  ch_Answer!n!Ack_Sinit!Encode(imgi, cid);
                  ch_Sending!InitCM(C)!Ind_Data!Encode(n, cid, dat); (*_PROBE_PLMN_*)
                  Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
                  )
             []
             [DTMCI(ci) eq n_way] -> (* any participant can send datas *)
                  (
                  ch_Answer!n!Ack_SAll!Encode(imgi, cid);
                  ContSend[ch_Sending](M_IDCM(C), Ind_Data, Encode(n, cid, dat))
                  >> (*_PROBE_PLMN_*)
                  Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
                  )
             []
             [DTMCI(ci) eq noway] -> (* should not happen *) (* call doesn't exist *)
                  ch_Answer!n!Err_Rejdat!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
                  Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
```

```
            )
            )
        []
        [(imgi Notin IMGISta(S)) or (n Notin M_IDCM(C))] -> (* should not happen *)
        (* group invalid or requester not participant to the call *)
            (
            ch_Answer!n!Err_Rejdat!Encode(imgi, cid, Cause); (*_PROBE_PLMN_*)
            Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
            )
        )
    )
)
endproc  (* process ContData *)

        (*====================================================================*)
        (*      process ContChange, provides the changing of geographical zone *)
        (*                      Controller point of view                       *)
        (*====================================================================*)

process ContChange[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                        DBSM: Statusmemberlist, n: M_ID, zg: GeoZone):noexit:=

(
    let S: Statusmember = RowSM(n, DBSM) in
    (
    [GeoZS(n,DBZS) eq NoGeozone] -> (* should not happen *) (* requester isn't normally attached*)
    (* it's possible to decide to not send the Ack_Change *)
        ch_Answer!n!Ack_Change!NoMsg;(*_PROBE_PLMN_*)
        Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
    []
    [GeoZS(n, DBZS) ne NoGeozone] -> (* requester is normally attached *)
    (
    ch_Answer!n!Ack_Change!NoMsg;
    (
    let Z: Geomember = RowZS(GeoZS(n, DBZS), DBZS) in
    (
        [MemSta(S) eq part] -> (* requester isn't a call initiator *)
        ( (*_PROBE_PLMN_*)
            ContChangeLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
            Insert(ZS(zg, Insert(n,M_IDZS(RowZS(zg, DBZS)))),Remove(RowZS(zg, DBZS),
            Insert(ZS(GeoZS(Z),Remove(n, M_IDZS(Z))), Remove(Z,DBZS)))),
            DBCM, DBSM,C_IDLCM(n,DBCM), n)
        )
        []
        [(MemSta(S) eq init) or (MemSta(S) eq Cont)] -> (* requester could have initiated some call *)
        ( (*_PROBE_PLMN_*)
            ContChangeQuit[ch_Uplink, ch_Answer, ch_Sending, ch_ID](
            Insert(ZS(zg, Insert(n,M_IDZS(RowZS(zg, DBZS)))),Remove(RowZS(zg, DBZS),
            Insert(ZS(GeoZS(Z), M_IDZS(Z)), Remove(Z,DBZS)))),
            DBCM, DBSM, C_IDSta(S), n)
        )
        []
        [MemSta(S) eq Nomember] -> (* Should not happen *) (* problem with the network, DBSM is corrupted *)
        ( (*_PROBE_PLMN_*)
            Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
        )
    )
```



```
        )
        )
    )
)
where

        (*====================================================================*)
        (*      process ContChangeLeave, provides the leaving of               *)
        (*           call when a changing of  geographical zone is requested   *)
        (*                      Controller point of view                       *)
        (*====================================================================*)

process ContChangeLeave[ch_Uplink,ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                        DBSM: Statusmemberlist, cidl: C_IDlist, n: M_ID):noexit:=
(
    let C: C_ID = Top(cidl) in
    (
    [C eq (NoElement of C_ID)] -> (* no more call to process *) (*_PROBE_PLMN_*)
        Controller[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM)
    []
    [C ne (NoElement of C_ID)] -> (* there is a call to process *)
    (
        ch_Answer!Ask!C!Nomsg;
        ch_Answer!C?ci: Callinfo;
        (
        let Z: Geozone = GeoCI(ci) in
        (
        [Z eq NoGeozone] -> (* no need to leave the call *)  (*_PROBE_PLMN_*)
        (*if the call isn't defined, this means that the databases aren't consistent *)
            ContChangeLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM, Remove(C, cidl), n)
        []
        [Z ne NoGeozone] -> (* the call must be leaved *)
        (
            let cm: Callmember = RowCM(C, DBCM) in
            (
            [JoiCI(ci) eq Join_leave_ind] -> (* send the leave indication *)
            (
                [SenCI(ci) eq Send_to_all_ind] -> (* send the leave indication to all *)
                (
                ContSend[ch_Sending](M_IDCM(cm), Ind_Leave, Encode(n, C))
                >> (*_PROBE_PLMN_*)
                ContChangeLeave[ch_Uplink,ch_Answer, ch_Sending, ch_ID](DBZS,
                    Insert(CM(C, initCM(cm), Remove(n, M_IDCM(cm)))), Remove(cm, DBCM)),
                    DBSM, Remove(C, cidl), n)
                )
                []
                [SenCI(ci) ne Send_to_all_ind] -> (* send leave indication only to initiator *)
                (
                ContSend[ch_Sending](Insert(InitCM(cm), {} of M_IDlist), Ind_Leave, Encode(n, C))
                >> (*_PROBE_PLMN_*)
                ContChangeLeave[ch_Uplink,ch_Answer, ch_Sending, ch_ID](DBZS,
                    Insert(CM(C, initCM(cm), Remove(n, M_IDCM(cm)))), Remove(cm, DBCM)),
                    DBSM, Remove(C, cidl), n)
                )
            )
            []
            [JoiCI(ci) ne Join_leave_ind] -> (* do not send leave indication *) (*_PROBE_PLMN_*)
```



```
                ContChangeLeave[ch_Uplink,ch_Answer, ch_Sending, ch_ID](DBZS,
                    Insert(CM(C, initCM(cm), Remove(n, M_IDCM(cm)))), Remove(cm, DBCM)),
                    DBSM, Remove(C, cidl), n)
            )
        )
        )
        )
    )
    )
)
endproc (* process ContChangeLeave *)

        (*====================================================================*)
        (*      process ContChangeQuit, provides the termination of            *)
        (*           call when a changing of  geographical zone is requested   *)
        (*                      Controller point of view                       *)
        (*====================================================================*)

process ContChangeQuit[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS: Geomemberlist, DBCM: Callmemberlist,
                        DBSM: Statusmemberlist, cidl: C_IDlist, n: M_ID): noexit:=
(
    let C: C_ID = Top(cidl) in
    (
    [C eq (NoElement of C_ID)] -> (* no more call to process *) (*_PROBE_PLMN_*)
        ContChangeLeave[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM, C_IDLCM(n, DBCM), n)
    []
    [C ne (NoElement of C_ID)] -> (* there is a call to process *)
    (
        ch_Answer!Ask!C!Nomsg;
        ch_Answer!C?ci:Callinfo;
        (
        let Z: Geozone = GeoCI(ci) in
        (
        [Z eq NoGeozone] ->  (* no need to terminate the call *)  (*_PROBE_PLMN_*)
        (*if the call isn't defined, this means that the databases aren't consistent *)
            ContChangeQuit[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, DBCM, DBSM, Remove(C, cidl), n)
        []
        [Z ne NoGeozone] -> (* the call must be terminated *)
        (
            let cm: Callmember = RowCM(C, DBCM) in
            let S: Statusmember = RowSM(n, DBSM) in
            (
            ch_Answer!Delete!C!nomsg;
            ch_Answer!Ask!NoIMGI of IMGI!C;
            ch_Answer?imgi: IMGI!C;
            (
            ch_Answer!Delete!imgi!C;
            ContSend[ch_Sending](M_IDCM(cm), Ind_Term, Encode(imgi, C, Cause))
            >> (*_PROBE_PLMN_*)
            ContChangeQuit[ch_Uplink, ch_Answer, ch_Sending, ch_ID](DBZS, Remove(cm, DBCM),
                Insert(SM(n, MemSta(S), IMGISta(S), Remove(C, C_IDSta(S)), AttSta(S)), Remove(S, DBSM)),
                Remove(C, cidl), n)
            )
            )
        )
        )
        )
```



```
    )
    )
)
endproc (* process ContChangeQuit *)

endproc  (* process ContChange *)

        (*====================================================================*)
        (*   process ContSend, provides the sending to the sending team of     *)
        (*      the informations for the receivers                             *)
        (*        Controller point of view                                     *)
        (*====================================================================*)

process ContSend[ch_Sending](midl: M_IDlist, ind: Ind, msg: Msg): exit:=
(
    [Top(midl) eq (Noelement of M_ID)] ->  (*_PROBE_PLMN_*) exit (* no more indication to send *)
    []
    [Top(midl) ne (NoElement of M_ID)] -> (* need to send an indication *)
        ch_Sending!Top(midl)!ind!msg; (*_PROBE_PLMN_*)
        ContSend[ch_Sending](Remove(Top(midl), midl), ind, msg)
)
endproc (* process ContSend *)


        (*====================================================================*)
        (*        process Answer_team, gives the answer to the requester       *)
        (*====================================================================*)

process Answer_team[ch_Downlink, ch_Answer]: noexit:=
hide ch_Req in
(
    Informer[ch_Downlink, ch_Answer, ch_Req]
    |[ch_Req]|
    (
    DatabaseCI[ch_Answer, ch_Req]({} of Callinfolist)
    |||
    DatabaseGC[ch_Answer, ch_Req](Insert(GC(3 of IMGI, {} of C_IDlist), Insert(GC(2 of IMGI, {} of C_IDlist),
                Insert(GC(1 of IMGI, {} of C_IDlist), {} of Groupcalllist))))
      (* DBGC of the Beginning, list of all IMGI with {} as list of C_ID *)
    )
)
endproc (* process Answer_team *)

        (*====================================================================*)
        (*      process Informer, gives the answer to the requester            *)
        (*====================================================================*)

process Informer[ch_Downlink, ch_Answer, ch_Req]: noexit:=
(
    ( (* send the error message *)
    ch_Answer?n: M_ID?err: Err?msg: Msg;
        (
        ch_Downlink!n!err!msg; (*_PROBE_PLMN_*)
        Informer[ch_Downlink, ch_Answer, ch_Req]
        )
    )
    []
```

```
( (* send the acknowledgment message or processing the message if status *)
ch_Answer?n: M_ID?ack: Ack?msg: Msg;
    (
    [ack ne Ack_Status] -> (* send message *)
    (
        ch_Downlink!n!ack!msg; (*_PROBE_PLMN_*)
        Informer[ch_Downlink, ch_Answer, ch_Req]
    )
    []
    [ack eq Ack_Status] -> (* processing message *)
    (
        hide sync in
        (
        let Statlist: Statusinfolist = GetStatusinfo(msg) in
        let imgi: IMGI = GetIMGI(msg) in
        let C: C_ID = GetC_ID(msg) in
        (
        (
        [(Sta_joinind Isin statlist) or (Sta_geozone Isin Statlist) or (Sta_qos Isin Statlist) or (Sta_dtm Isin statlist)] ->
        (* asking to the database DBCI *)
        (
            ch_Req!C;
            ch_Req!C?ci: Callinfo; (*_PROBE_PLMN_*)
            exit(ci)
        )
        []
        [(Sta_joinind Notin statlist) and (Sta_geozone Notin Statlist) and
         (Sta_qos Notin Statlist) and (Sta_dtm Notin statlist)] ->
        (* no need to ask to the DBCI *)
        ( (*_PROBE_PLMN_*)
            exit(CI(NoC_ID, noway, qos1, NoGeozone, No_join_leave_ind, No_send_to_all, No_call_not))
        )
        )>> Accept ci: Callinfo in
        (
        (
            [Sta_callids Isin Statlist] ->
            (
                ch_Req!imgi;
                ch_Req!imgi?gc: Groupcall;
                (
                sync!C_IDGC(gc)?midl: M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM?qos: QoS?zg: GeoZone;
                exit
                )
            )
            []
            [Sta_callids Notin Statlist] ->
            (
                sync!{} of C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM?qos: QoS?zg: GeoZone;
                exit
            )
        )
        |[sync]|
        (
            [Sta_joinind Isin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat!JoiCI(ci)?dtm: DTM?qos: QoS?zg: GeoZone;
                exit
```



```
            )
            []
            [Sta_joinind Notin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat!No_join_leave_ind?dtm: DTM?qos: QoS?zg: GeoZone;
                exit
            )
        )
        |[sync]|
        (
            [Sta_geozone Isin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM?qos: QoS!GeoCI(ci);
                exit
            )
            []
            [Sta_geozone Notin Statlist] ->
            (
              sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM?qos: QoS!NoGeozone of GeoZone;
              exit
            )
        )
        |[sync]|
        (
            [Sta_memlist Isin Statlist] ->
            (
                sync?cidl: C_IDlist!GetM_IDlist(msg)?nb: Nat?joi: Join_leave?dtm: DTM?qos: QoS?zg: GeoZone;
                exit
            )
            []
            [Sta_memlist Notin Statlist] ->
            (
                sync?cidl: C_IDlist!{} of M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM?qos: QoS?zg: GeoZone;
                exit
            )
        )
        |[sync]|
        (
            [Sta_nbmem Isin Statlist] ->
            (
              sync?cidl: C_IDlist?midl: M_IDlist!GetNbmem(msg)?joi: Join_leave?dtm: DTM?qos: QoS?zg: GeoZone;
              exit
            )
            []
            [Sta_nbmem Notin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist!0 of Nat?joi: Join_leave?dtm: DTM?qos: QoS?zg: GeoZone;
                exit
            )
        )
        |[sync]|
        (
            [Sta_qos Isin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM!QoSCI(ci)?zg: GeoZone;
                exit
            )
```



```
            []
            [Sta_qos Notin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM!qos1?zg: GeoZone;
                exit
            )
        )
        |[sync]|
        (
            [Sta_DTM Isin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave!DTMCI(ci)?qos: QoS?zg: GeoZone;
                exit
            )
            []
            [Sta_DTM Notin Statlist] ->
            (
                sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave!noway of DTM?qos: QoS?zg: GeoZone;
                exit
            )
        )
        |[sync]|
        (
            sync?cidl: C_IDlist?midl: M_IDlist?nb: Nat?joi: Join_leave?dtm: DTM?qos: QoS?zg: GeoZone;
            (
            [C ne NoC_ID] ->
            (
                ch_Downlink!n!Ack_Status!encode(imgi, C, midl, nb, joi, dtm, qos, zg);
                exit
            )
            []
            [C eq NoC_ID] ->
            (
                ch_Downlink!n!Ack_Status!Encode(imgi, cidl);
                exit
            )
            )
        )
        )
        )
        )>> (*_PROBE_PLMN_*)
        Informer[ch_Downlink, ch_Answer, ch_Req]
    )
    )
  )
)
endproc (* process Informer *)

    (*======================================================================*)
    (*     process DatabaseCI, provides the DBCI base                       *)
    (*======================================================================*)

process DatabaseCI[ch_Answer, ch_Req](DBCI: Callinfolist): noexit:=
(
(
    ch_Answer?req:DBRequest?cid: C_ID?msg:Msg;
    (
```



```
    (
    [req eq Insert] -> (*_PROBE_PLMN_*)
      DatabaseCI[ch_Answer, ch_Req]
            (Insert(CI(cid, GetDTM(msg), GetQoS(msg), GetGeo(msg), GetJoin(msg), GetSend(msg), Getcallnot(msg)), DBCI)
    )
    []
    (
    [req eq Delete] -> (*_PROBE_PLMN_*)
      DatabaseCI[ch_Answer, ch_Req](Remove(RowCI(cid, DBCI), DBCI))
    )
    []
    (
    [req eq Ask] ->
      Ch_Answer!cid!RowCI(cid, DBCI); (*_PROBE_PLMN_*)
      DatabaseCI[ch_Answer, ch_Req](DBCI)
    )
    )
    )
)
[]
(
    ch_Req?cid: C_ID;
    ch_Req!cid!RowCI(cid, DBCI); (*_PROBE_PLMN_*)
    databaseCI[ch_Answer, ch_Req](DBCI)
)
)
endproc (* process DatabaseCI *)

    (*======================================================================*)
    (*     process DatabaseGC, provides the DBGC base                       *)
    (*======================================================================*)

process DatabaseGC[ch_Answer, ch_Req](DBGC: Groupcalllist): noexit:=
(
(
    ch_Answer?req:DBRequest?imgi: IMGI?cid: C_ID;
    (
    [req eq Insert] ->
    (
        let G: Groupcall = RowGC(imgi, DBGC) in
        ( (*_PROBE_PLMN_*)
        DatabaseGC[ch_Answer, ch_Req](Insert(GC(imgi, Insert(cid, C_IDGC(G))), Remove(G, DBGC)))
        )
    )
    []
    [req eq Delete] ->
    (
        let G: Groupcall = RowGC(imgi, DBGC) in
        ( (*_PROBE_PLMN_*)
        DatabaseGC[ch_Answer, ch_Req](Insert(GC(imgi, Remove(cid, C_IDGC(G))), Remove(G, DBGC)))
        )
    )
    []
    [req eq Ask] ->
    (
        ch_Answer!IMGC(cid, DBGC)!cid; (*_PROBE_PLMN_*)
        DatabaseGC[ch_Answer, ch_Req](DBGC)
```

```
        ) (* here the probe 63 died *)

        )
    )
    []
    (
        ch_Req?imgi: IMGI;
        ch_Req!imgi!RowGC(imgi, DBGC); (*_PROBE_PLMN_*)
        DatabaseGC[ch_Answer, ch_Req](DBGC)
    )
    )
    endproc (* process DatabaseGC *)

        (*====================================================================*)
        (*         process Senders_team, gives the indications to the receivers *)
        (*====================================================================*)

    process Senders_team[ch_Downlink, ch_Sending]: noexit :=

    (
        ch_Sending?n: M_ID?ind: Ind?msg: Msg;
        (
            [ind eq Ind_Init] -> ch_Downlink!n!Ind_Init!Encode(GetIMGI(msg), GetC_ID(msg)); (*_PROBE_PLMN_*) exit
        []
            [ind eq Ind_Term] -> ch_Downlink!n!Ind_Term!Encode(GetC_ID(msg), GetCause(msg)); (*_PROBE_PLMN_*) exit
        []
            [ind eq Ind_Join] -> ch_Downlink!n!Ind_Join!Encode(GetM_ID(msg), GetC_ID(msg)); (*_PROBE_PLMN_*) exit
        []
            [ind eq Ind_Leave] -> ch_Downlink!n!Ind_Leave!Encode(GetM_ID(msg), GetC_ID(msg));
                                      (*_PROBE_PLMN_*) exit
        []
            [ind eq Ind_Data] -> ch_Downlink!n!Ind_Data!Encode(GetM_ID(msg), GetC_ID(msg), GetData(msg));
                                      (*_PROBE_PLMN_*) exit
        ) >>(*_PROBE_PLMN_*)
        Senders_team[ch_Downlink, ch_Sending]
    )
    endproc (* process Senders_team *)

(*=================== tests ====================================================*)

process test_att1[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_attach, ack_attach expected *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    success; stop
)
endproc

process test_att2[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_attach, wrong requester err_rejatt expected *)
(
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    success; stop
```



```
)
endproc

process test_detach1[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_detach no call to leave or terminate *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_detach ! nomsg;
    ch_downlink  ! init1 ! ack_detach ! nomsg;
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_detach2[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_detach, 2 calls to terminate one with 2 members second no members *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem2 ! req_attach ! encode(1);
    ch_downlink  ! mem2 ! ack_attach ! nomsg;
    ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! init1 ! req_detach ! nomsg;
    ch_downlink  ! init1 ! ack_detach ! nomsg;
    ch_downlink  ! mem2 ! ind_term ! encode(succ(noc_id),cause);
    ch_downlink  ! mem1 ! ind_term ! encode(succ(noc_id),cause);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_detach3[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_detach, 2 calls to leave one with send_to_all_ind, the second with no_send_to_all *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,join_leave_ind,no_send_to_all,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
```



```
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink!init1! ind_join!encode(mem1, succ(noc_id));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(succ(noc_id)));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_Downlink!init1! ind_join!encode(mem1, succ(succ(noc_id)));
    ch_uplink  ! mem2 ! req_attach ! encode(1);
    ch_downlink  ! mem2 ! ack_attach ! nomsg;
    ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink!mem1! ind_join!encode(mem2, succ(noc_id));
    ch_Downlink!init1! ind_join!encode(mem2, succ(noc_id));
    ch_uplink  ! mem1 ! req_detach ! nomsg;
    ch_downlink  ! mem1 ! ack_detach ! nomsg;
    ch_Downlink!mem2 !ind_leave!encode(mem1, succ(noc_id));
    ch_Downlink!init1!ind_leave!encode(mem1, succ(noc_id));
    ch_downlink  ! init1 ! ind_leave! encode(mem1,succ(succ(noc_id)));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_detach4[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_detach, 1 call to leave without join_leave_ind *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_detach ! nomsg;
    ch_downlink  ! mem1 ! ack_detach ! nomsg;
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_detach5[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_detach, 1 call to leave, 1 to quit *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! adm1 ! req_attach ! encode(1);
```



```
    ch_downlink  ! adm1 ! ack_attach ! nomsg;
    ch_uplink  ! adm1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! adm1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! adm1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! adm1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink!init1!ind_join!Encode(adm1, succ(noc_id));
    ch_uplink  ! adm1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! adm1 ! ack_c_init !encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! init1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! init1 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! init1 ! req_join ! encode(1,succ(succ(noc_id)));
    ch_downlink  ! init1 ! ack_join ! encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! adm1 ! req_detach ! nomsg;
    ch_downlink  ! adm1 ! ack_detach ! nomsg;
    ch_Downlink!init1!ind_term!encode(succ(succ(noc_id)), Cause);
    ch_Downlink!init1!ind_leave!encode(adm1, succ(noc_id));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_init1[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_init, call_not_ind anser ack expected*)
(
    ch_Uplink!init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!init1!Ack_Attach!nomsg;
    ch_Uplink!mem1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem1!Ack_Attach!nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,call_not_ind);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_downlink!mem1!ind_init!encode(1, succ(noc_id));
    ch_downlink  ! init1 ! ind_init ! encode(1,succ(noc_id));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_init2[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_init wrong IMGI, err_rejinit expected *)
(
    ch_Uplink!init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!init1!Ack_Attach!nomsg;
    ch_uplink  ! init1 ! req_init !encode(3,n_way,qos1,1,no_join_leave_ind,no_send_to_all,call_not_ind);
    ch_downlink  ! init1 ! Err_Rejinit!Encode(Cause);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_init3[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_init, no_call_not, ack_C_Init expected *)
(
    ch_Uplink!init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!init1!Ack_Attach!nomsg;
```

```
	ch_uplink  ! init1 ! req_init !encode(1,one_way,qos2,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
	ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_init4[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_init, requester doesn't have the rights err_rejinit expected *)
(
	ch_Uplink!mem1!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem1!Ack_Attach!nomsg;
	ch_uplink  ! mem1 ! req_init !encode(1,one_way,qos2,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
	ch_downlink  ! mem1 ! Err_Rejinit !encode(cause);
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_init5[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_init, requester is in the wrong Geozone, err_rejinit expected *)
(
	ch_Uplink!init1!Req_Attach!Encode(2 of Geozone);
	ch_Downlink!init1!Ack_Attach!nomsg;
	ch_uplink  ! init1 ! req_init !encode(1,two_way,qos3,1,no_join_leave_ind,no_send_to_all,call_not_ind);
	ch_downlink  ! init1 ! Err_Rejinit!Encode(Cause);
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_term1[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_term, 2 members, expected ack_c_term *)
(
	ch_Uplink!init1!Req_Attach!Encode(2 of Geozone);
	ch_Downlink!init1!Ack_Attach!nomsg;
	ch_uplink  ! init1 ! req_init !encode(1,one_way,qos2,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
	ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_uplink  ! mem1 ! req_attach ! encode(1);
	ch_downlink  ! mem1 ! ack_attach ! nomsg;
	ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
	ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! init1 ! ind_join ! encode(mem1,succ(noc_id));
	ch_uplink  ! mem2 ! req_attach ! encode(1);
	ch_downlink  ! mem2 ! ack_attach ! nomsg;
	ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
	ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! mem1 ! ind_join ! encode(mem2,succ(noc_id));
	ch_downlink  ! init1 ! ind_join ! encode(mem2,succ(noc_id));
	ch_Uplink!init1!Req_Term!Encode(1, succ(noc_id));
```



```
	ch_downlink  ! init1 ! ack_c_term ! encode(1,succ(noc_id));
	ch_downlink  ! mem2 ! ind_term ! encode(succ(noc_id),cause);
	ch_downlink  ! mem1 ! ind_term ! encode(succ(noc_id),cause);
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_term2[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_term, 0 member, expected ack_c_term *)
(
	ch_Uplink!init1!Req_Attach!Encode(2 of Geozone);
	ch_Downlink!init1!Ack_Attach!nomsg;
	ch_uplink  ! init1 ! req_init !encode(1,one_way,qos2,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
	ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_Uplink!init1!Req_Term!Encode(1, succ(noc_id));
	ch_downlink  ! init1 ! ack_c_term ! encode(1,succ(noc_id));
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_status1[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_status, requester doesn't have rights, err_rejstat expected *)
(
	ch_Uplink!init1!Req_Attach!Encode(2);
	ch_Downlink!init1!Ack_Attach!nomsg;
	ch_uplink  ! init1 ! req_init !encode(2,one_way,qos2,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
	ch_downlink  ! init1 ! ack_c_init !encode(2,succ(noc_id),cipher_key(succ(noc_id)));
	ch_uplink  ! mem2 ! req_attach ! encode(1);
	ch_downlink  ! mem2 ! ack_attach ! nomsg;
	ch_uplink  ! mem2 ! req_status ! encode(2,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem2 ! err_rejstat ! encode(2,noc_id,Cause);
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_status2[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_status, sta_callids, ack_status expected *)
(
	ch_Uplink!init1!Req_Attach!Encode(2);
	ch_Downlink!init1!Ack_Attach!nomsg;
	ch_uplink  ! init1 ! req_init !encode(2,one_way,qos2,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
	ch_downlink  ! init1 ! ack_c_init !encode(2,succ(noc_id),cipher_key(succ(noc_id)));
	ch_uplink  ! mem1 ! req_attach ! encode(1);
	ch_downlink  ! mem1 ! ack_attach ! nomsg;
	ch_uplink  ! mem1 ! req_status ! encode(2,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem1 ! ack_status ! encode(2,insert(succ(noc_id), {}));
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc
```



```
process test_status3[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_status, other status info than sta_callids, ack_status expected *)
(
	ch_Uplink!init1!Req_Attach!Encode(2);
	ch_Downlink!init1!Ack_Attach!nomsg;
	ch_uplink  ! init1 ! req_init !encode(2,one_way,qos2,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
	ch_downlink  ! init1 ! ack_c_init !encode(2,succ(noc_id),cipher_key(succ(noc_id)));
	ch_uplink  ! mem1 ! req_attach ! encode(1);
	ch_downlink  ! mem1 ! ack_attach ! nomsg;
	ch_uplink  ! mem1 ! req_status ! encode(2,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem1 ! ack_status ! encode(2,insert(succ(noc_id), {}));
	ch_Uplink! mem1! req_Status! encode(2, succ(noc_id), insert(sta_nbmem, insert(sta_dtm, {})));
	ch_downlink  ! mem1 ! ack_status ! encode(2,succ(noc_id),{},succ(0),no_join_leave_ind,one_way,qos1,nogeozone);
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_join1[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_join, join_leave_ind, no_send_to_all => only initiator to send ind_join *)
(
	ch_uplink  ! init1 ! req_attach ! encode(1);
	ch_downlink  ! init1 ! ack_attach ! nomsg;
	ch_uplink  ! mem1 ! req_attach ! encode(1);
	ch_downlink  ! mem1 ! ack_attach ! nomsg;
	ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,join_leave_ind,no_send_to_all,call_not_ind);
	ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! mem1 ! ind_init ! encode(1,succ(noc_id));
	ch_downlink  ! init1 ! ind_init ! encode(1,succ(noc_id));
	ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
	ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! init1 ! ind_join ! encode(mem1,succ(noc_id));
	ch_uplink  ! mem2 ! req_attach ! encode(1);
	ch_downlink  ! mem2 ! ack_attach ! nomsg;
	ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
	ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! init1 ! ind_join ! encode(mem2,succ(noc_id));
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_join2[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_join, join_leave_ind, send_to_all => send ind_join to all members *)
(
	ch_uplink  ! init1 ! req_attach ! encode(1);
	ch_downlink  ! init1 ! ack_attach ! nomsg;
	ch_uplink  ! mem1 ! req_attach ! encode(1);
	ch_downlink  ! mem1 ! ack_attach ! nomsg;
	ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,call_not_ind);
```



```
	ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! mem1 ! ind_init ! encode(1,succ(noc_id));
	ch_downlink  ! init1 ! ind_init ! encode(1,succ(noc_id));
	ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
	ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! init1 ! ind_join ! encode(mem1,succ(noc_id));
	ch_uplink  ! mem2 ! req_attach ! encode(1);
	ch_downlink  ! mem2 ! ack_attach ! nomsg;
	ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
	ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_Downlink ! mem1 ! ind_join ! encode(mem2,succ(noc_id));
	ch_downlink  ! init1 ! ind_join ! encode(mem2,succ(noc_id));
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_join3[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_join, call terminted before join expected err_rejjoin *)
(
	ch_uplink  ! init1 ! req_attach ! encode(1);
	ch_downlink  ! init1 ! ack_attach ! nomsg;
	ch_uplink  ! mem1 ! req_attach ! encode(1);
	ch_downlink  ! mem1 ! ack_attach ! nomsg;
	ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,call_not_ind);
	ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_downlink  ! mem1 ! ind_init ! encode(1,succ(noc_id));
	ch_downlink  ! init1 ! ind_init ! encode(1,succ(noc_id));
	ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_Uplink!init1!Req_Term!Encode(1, succ(noc_id));
	ch_downlink  ! init1 ! ack_c_term ! encode(1,succ(noc_id));
	ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
	ch_downlink  ! mem1 ! err_rejjoin ! encode(1, succ(noc_id), Cause);
	ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
	ch_Downlink!mem3!Err_rejatt!encode(Cause);
	success; stop
)
endproc

process test_join4[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_join, no_join_leave_ind expected ack_join, no ind_join sended *)
(
	ch_uplink  ! init1 ! req_attach ! encode(1);
	ch_downlink  ! init1 ! ack_attach ! nomsg;
	ch_uplink  ! mem1 ! req_attach ! encode(1);
	ch_downlink  ! mem1 ! ack_attach ! nomsg;
	ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,no_join_leave_ind,no_send_to_all,no_call_not);
	ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
	ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
	ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
	ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
```

```
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_leave1[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_leave, no_join_leave_ind, ack_Leave expected *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Uplink!mem1!Req_Leave!Encode(1, succ(noc_id));
    ch_Downlink!mem1!Ack_Leave!Encode(1,succ(noc_id));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_leave2[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_leave, join_leave_ind, send_to_all, ind_leave to all *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_downlink  ! init1 ! ind_join ! encode(mem1,succ(noc_id));
    ch_uplink  ! mem2 ! req_attach ! encode(1);
    ch_downlink  ! mem2 ! ack_attach ! nomsg;
    ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink ! mem1 ! ind_join ! encode(mem2,succ(noc_id));
    ch_downlink  ! init1 ! ind_join ! encode(mem2,succ(noc_id));
    ch_Uplink!mem1!Req_Leave!Encode(1, succ(noc_id));
    ch_Downlink!mem1!Ack_Leave!Encode(1,succ(noc_id));
    ch_Downlink!mem2!ind_leave!Encode(mem1, succ(noc_id));
    ch_Downlink!init1!ind_leave!Encode(mem1, succ(noc_id));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
```



```
)
endproc

process test_leave3[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_leave join_leave_ind, no_send_to_all, ind_leave only to initiator *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,join_leave_ind,no_send_to_all,no_call_not);
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_downlink  ! init1 ! ind_join ! encode(mem1,succ(noc_id));
    ch_uplink  ! mem2 ! req_attach ! encode(1);
    ch_downlink  ! mem2 ! ack_attach ! nomsg;
    ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_downlink  ! init1 ! ind_join ! encode(mem2,succ(noc_id));
    ch_Uplink!mem1!Req_Leave!Encode(1, succ(noc_id));
    ch_Downlink!mem1!Ack_Leave!Encode(1,succ(noc_id));
    ch_Downlink!init1!ind_leave!Encode(mem1, succ(noc_id));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_leave4[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_leave, initiator want leave -can only terminate a initiated call- err_rejleave expected *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,join_leave_ind,no_send_to_all,no_call_not);
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! init1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! init1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_downlink  ! init1 ! ind_join ! encode(init1,succ(noc_id));
    ch_Uplink!init1!Req_Leave!Encode(1, succ(noc_id));
    ch_Downlink!init1!Err_Rejleave!Encode(1,succ(noc_id), Cause);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_data1[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_data, requester is initiator, ack_Sall expected *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,no_join_leave_ind,no_send_to_all,no_call_not);
```



```
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Uplink!init1!req_data!Encode(1, succ(noc_id), Data);
    ch_Downlink!init1!ack_Sall!encode(1, succ(noc_id));
    ch_Downlink!mem1!ind_data!encode(init1, succ(noc_id), Data);
    ch_Downlink!init1!ind_data!encode(init1, succ(noc_id), Data);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_data2[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_data, n_way, requester is not initiator, ack_Sall expected *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,n_way,qos1,1,no_join_leave_ind,no_send_to_all,no_call_not);
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Uplink!mem1!req_data!Encode(1, succ(noc_id), Data);
    ch_Downlink!mem1!ack_Sall!encode(1, succ(noc_id));
    ch_Downlink!mem1!ind_data!encode(mem1, succ(noc_id), Data);
    ch_Downlink!init1!ind_data!encode(mem1, succ(noc_id), Data);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_data3[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_data, 2_way, requester is not initiator, ack_SInit expected *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,two_way,qos1,1,no_join_leave_ind,no_send_to_all,no_call_not);
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Uplink!mem1!req_data!Encode(1, succ(noc_id), Data);
    ch_Downlink!mem1!ack_SInit!encode(1, succ(noc_id));
    ch_Downlink!init1!ind_data!encode(mem1, succ(noc_id), Data);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
```



```
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_data4[ch_Uplink, ch_Downlink, Success]: noexit:=
(* req_data, 1_way, requester is not initiator, err_Wmode expected *)
(
    ch_uplink  ! init1 ! req_attach ! encode(1);
    ch_downlink  ! init1 ! ack_attach ! nomsg;
    ch_uplink  ! init1 ! req_init ! encode(1,one_way,qos1,1,no_join_leave_ind,no_send_to_all,no_call_not);
    ch_downlink  ! init1 ! ack_c_init ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Uplink!mem1!req_data!Encode(1, succ(noc_id), Data);
    ch_Downlink!mem1!err_Wmode!encode(1, succ(noc_id), Cause);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_change1[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_change no call to leave or terminate *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_change ! Encode(2);
    ch_downlink  ! init1 ! ack_change ! nomsg;
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_change2[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_change, 2 calls to terminate one with 2 members second no members *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem2 ! req_attach ! encode(1);
    ch_downlink  ! mem2 ! ack_attach ! nomsg;
    ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
```

```
    ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! init1 ! req_change ! Encode(2);
    ch_downlink  ! init1 ! ack_change ! nomsg;
    ch_Downlink ! init1 ! ind_term !encode(succ(succ(noc_id)),cause);
    ch_downlink  ! mem2 ! ind_term ! encode(succ(noc_id),cause);
    ch_downlink  ! mem1 ! ind_term ! encode(succ(noc_id),cause);
    ch_Downlink ! init1 ! ind_term ! encode(succ(noc_id),cause);
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_change3[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_change, 2 calls to leave one with send_to_all_ind, the second with no_send_to_all *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,join_leave_ind,no_send_to_all,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink!init1! ind_join!encode(mem1, succ(noc_id));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(succ(noc_id)));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_Downlink!init1! ind_join!encode(mem1, succ(succ(noc_id)));
    ch_uplink  ! mem2 ! req_attach ! encode(1);
    ch_downlink  ! mem2 ! ack_attach ! nomsg;
    ch_uplink  ! mem2 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem2 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! mem2 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem2 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink!mem1! ind_join!encode(mem2, succ(noc_id));
    ch_Downlink!init1! ind_join!encode(mem2, succ(noc_id));
    ch_uplink  ! mem1 ! req_change ! encode(2);
    ch_downlink  ! mem1 ! ack_change ! nomsg;
    ch_Downlink!mem2 !ind_leave!encode(mem1, succ(noc_id));
    ch_Downlink!mem1 !ind_leave!encode(mem1, succ(noc_id));
    ch_Downlink!init1!ind_leave!encode(mem1, succ(noc_id));
    ch_downlink  ! init1 ! ind_leave! encode(mem1,succ(succ(noc_id)));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_change4[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_change, 1 call to leave without join_leave_ind *)
(
```



```
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_attach ! encode(1);
    ch_downlink  ! mem1 ! ack_attach ! nomsg;
    ch_uplink  ! mem1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! mem1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! mem1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! mem1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! mem1 ! req_change ! encode(3);
    ch_downlink  ! mem1 ! ack_change ! nomsg;
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_change5[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_change, 1 call to leave, 1 to quit *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,1,join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_uplink  ! adm1 ! req_attach ! encode(1);
    ch_downlink  ! adm1 ! ack_attach ! nomsg;
    ch_uplink  ! adm1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! adm1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! adm1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! adm1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink!init1!ind_join!Encode(adm1, succ(noc_id));
    ch_uplink  ! adm1 ! req_init !encode(1,n_way,qos1,1,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! adm1 ! ack_c_init !encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! init1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! init1 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! init1 ! req_join ! encode(1,succ(succ(noc_id)));
    ch_downlink  ! init1 ! ack_join ! encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! adm1 ! req_change ! encode(2);
    ch_downlink  ! adm1 ! ack_change ! nomsg;
    ch_Downlink!init1!ind_term!encode(succ(succ(noc_id)), Cause);
    ch_Downlink!adm1!ind_term!encode(succ(succ(noc_id)), Cause);
    ch_Downlink!adm1!ind_leave!encode(adm1, succ(noc_id));
    ch_Downlink!init1!ind_leave!encode(adm1, succ(noc_id));
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc

process test_change6[ch_Uplink, ch_Downlink, Success]:noexit:=
(* req_change, 1 call not to leave, 1 not to quit: NoGeozone *)
(
    ch_Uplink!Init1!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!Init1!Ack_Attach!Nomsg;
    ch_uplink  ! init1 ! req_init !encode(1,n_way,qos1,NoGeozone,join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! init1 ! ack_c_init !encode(1,succ(noc_id),cipher_key(succ(noc_id)));
```



```
    ch_uplink  ! adm1 ! req_attach ! encode(1);
    ch_downlink  ! adm1 ! ack_attach ! nomsg;
    ch_uplink  ! adm1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! adm1 ! ack_status ! encode(1,insert(succ(noc_id),{}));
    ch_uplink  ! adm1 ! req_join ! encode(1,succ(noc_id));
    ch_downlink  ! adm1 ! ack_join ! encode(1,succ(noc_id),cipher_key(succ(noc_id)));
    ch_Downlink!init1!ind_join!Encode(adm1, succ(noc_id));
    ch_uplink  ! adm1 ! req_init !encode(1,n_way,qos1,NoGeozone,no_join_leave_ind,send_to_all_ind,no_call_not);
    ch_downlink  ! adm1 ! ack_c_init !encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! init1 ! req_status ! encode(1,noc_id,insert(sta_callids,{}));
    ch_downlink  ! init1 ! ack_status ! encode(1,insert(succ(succ(noc_id)),insert(succ(noc_id),{})));
    ch_uplink  ! init1 ! req_join ! encode(1,succ(succ(noc_id)));
    ch_downlink  ! init1 ! ack_join ! encode(1,succ(succ(noc_id)),cipher_key(succ(succ(noc_id))));
    ch_uplink  ! adm1 ! req_change ! encode(2);
    ch_downlink  ! adm1 ! ack_change ! nomsg;
    ch_Uplink!mem3!Req_Attach!Encode(1 of Geozone);
    ch_Downlink!mem3!Err_rejatt!encode(Cause);
    success; stop
)
endproc


endspec (* specification Groupcall *)
```