SDT	System file	rw	C:\tau34\Q1228\Inap\inap.sdt
	Source directory	rw	C:\tau34\Q1228\Inap\
	Text readme	--	TCAP\readme.txt
	Chapter ASN.1 Files		
ASN	ASN.1 Text INCS2datatypes	rw	ASN1\incs2-dt.asn
ASN	ASN.1 Text INCS2SSFSCFopsargs	rw	ASN1\ssf-scf.asn
ASN	ASN.1 Text INCS2BundleArg	rw	ASN1\incs2b.asn
ASN	ASN.1 Text INCS2Internals	rw	ASN1\cs2i.asn
	Chapter IN CS-1 Specification 21		
	Package CS1_INAP 21	rw	CS1\cs1_inap.sun
ASN	Depending on ASN.1 Text INCS2BundleArg		
ASN	Depending on ASN.1 Text INCS2datatypes		
ASN	Depending on ASN.1 Text INCS2Internals		
ASN	Depending on ASN.1 Text INCS2SSFSCFopsargs		
	System Type CS1_INAP 22	rw	CS1\cs1_inap.sst
x:y	Block Instance SSF_CCF_A : SSF_CCF		
x:y	Block Instance SSF_CCF_B : SSF_CCF		
x:y	Block Instance TCAP_Adapter : TCAP_Simulator		
	Virtual Block Type SSF_CCF 30	rw	CS1\ssf1.sbt
x:y	Process Instance CS (0) : CallSegment		
x:y	Process Instance CSA (0) : CallSegmentAssociation		
x:y	Process Instance IH (1,1) : InterfaceHandler		
x:y	Process Instance O_BCSM (0) : OriginatingBCSM		
x:y	Process Instance SSF (0) : SSF_FSM		
x:y	Process Instance SSME (1,1) : SSME_FSM		
x:y	Process Instance T_BCSM (0) : TerminatingBCSM		
	Virtual Process Type InterfaceHandler 36	rw	CS1\ih.spt
	Procedure AllocateCSAId 48	rw	CS1\alldid.spd
	Procedure AddCSAId 49	rw	CS1\adddid.spd
	Procedure GetCSAIdfromCSA 50	rw	CS1\getdid.spd
	Procedure GetCSAfromCSAId 51	rw	CS1\getcvd.spd
	Procedure GetCSAfromSigConId 52	rw	CS1\getcvu.spd

- Procedure SetSigConAssoc 53 rw CS1\setsca.spd
- Procedure SetRemoteAssoc 54 rw CS1\setrema.spd
- Procedure GetCSAfromRemoteId 55 rw CS1\getlcsa.spd
- Procedure IsCSA 56 rw CS1\iscsa.spd

Virtual Process Type CallSegmentAssociation 57 rw CS1\csa.spt

- Procedure AddCS 78 rw CS1\addcs.spd
- Procedure SetLegLocation 79 rw CS1\setlegl.spd
- Procedure GetCSPtr 80 rw CS1\getcsptr.spd
- Procedure GetLegLocation 81 rw CS1\getlegl.spd
- Procedure ExistCS 82 rw CS1\existcs.spd
- Procedure SetLegAssoc 83 rw CS1\setlassc.spd
- Procedure IsCS 84 rw CS1\iscs.spd
- Procedure GetLegIdfromRemoteLegId 85 rw CS1\getlegrc.spd
- Procedure ExistLeg 86 rw CS1\existleg.spd

Virtual Process Type CallSegment 87 rw CS1\cv1.spt

- Procedure AddLeg 113 rw CS1\addleg.spd
- Procedure RemoveLeg 114 rw CS1\remleg.spd
- Procedure SetLegStatus 115 rw CS1\setlstat.spd
- Procedure ReleaseAllLegs 116 rw CS1\releasea.spd
- Procedure GetLegStatus 117 rw CS1\getlstat.spd
- Procedure SetLegAssoc 118 rw CS1\setlass.spd
- Procedure SetLegPtr 119 rw CS1\setlptr.spd
- Procedure GetLegIdfromRemoteLegId 120 rw CS1\getlegr.spd
- Procedure GetLegPtr 121 rw CS1\getlptr.spd
- Procedure IsBCSM 122 rw CS1\isbcsm.spd
- Procedure GetPassiveLegId 123 rw CS1\getpl.spd
- Procedure MapConnectToBCSM 124 rw CS1\mpcobcsm.spd
- Procedure MapSIToBCSM 125 rw CS1\mpsibcsm.spd

Virtual Process Type SSF_FSM 126 rw CS1\ssf_fsm.spt

- ***Virtual*** Procedure InitialiseDPTable 149 rw CS1\initevl.spd
- Procedure ExistLeg 150 rw CS1\exleg.spd
- Procedure AnyDPArmed 151 rw CS1\anyevarm.spd
- Procedure IsDPArmed 152 rw CS1\evarm.spd
- Procedure DisarmDPs 153 rw CS1\disarm.spd
- Procedure DPArmed 154 rw CS1\dparmed.spd
- Procedure CallInformationReportPending 155 rw CS1\callinfo.spd
- Procedure ApplyChargingReportPending 156 rw CS1\applycha.spd
- Procedure ArmTDPs 157 rw CS1\armtdps.spd

- Procedure MatchingServiceFilteringCriteria 158 rw CS1\msfc.spd
- Procedure CheckACG 159 rw CS1\cacg.spd
- Procedure CallFiltered 160 rw CS1\cf.spd
- Procedure ProcessApplyCharging 161 rw CS1\process6.spd
- Procedure ProcessContinue 162 rw CS1\proces12.spd
- Procedure ProcessRequestReportBCSMEvent 163 rw CS1\prreqrep.spd
- Procedure ProcessAnalyseInformation 164 rw CS2\procai.spd
- Procedure ProcessCallInformationRequest 165 rw CS1\process7.spd
- Procedure ProcessDisconnectForwardConnection 166 rw CS1\processd.spd
- Procedure ProcessSendChargingInformation 167 rw CS1\procsci.spd
- Procedure ProcessSelectRoute 168 rw CS2\procsr.spd
- Procedure ProcessCancel 169 rw CS1\process8.spd
- Procedure ProcessEstablishTemporaryConnection 170 rw CS1\processe.spd
- Procedure ProcessEventReportBCSM 171 rw CS1\prevrep.spd
- Procedure ProcessSelectFacility 172 rw CS2\procsf.spd
- Procedure ProcessCollectInformation 173 rw CS1\process9.spd
- Procedure ProcessFurnishChargingInformation 174 rw CS1\proces13.spd
- Procedure ProcessForwardConnectionReleased 175 rw CS1\processf.spd
- Procedure ProcessConnect 176 rw CS1\proces10.spd
- Procedure ProcessInitiateCallAttempt 177 rw CS1\processi.spd
- Procedure ProcessInitialDP 178 rw CS1\pridp.spd
- Procedure ProcessConnectToResource 180 rw CS1\proces11.spd
- Procedure ProcessRequestNotificationChargingEvent 181 rw CS1\proces16.spd
- ***Virtual*** Procedure ProcessDPSpecific 182 rw CS1\ProcDPS.spd
- Procedure ConnnectAnalysis 183 rw CS1\ERROR\conpa.spd

Virtual Process Type OriginatingBCSM 184 rw CS1\ocs1.spt

- Procedure PIC_O_Null 196 rw CS1\pic_o_nu.spd
- Procedure PIC_Analyse_Information 197 rw CS1\pic_anal.spd
- ***Virtual*** Procedure PIC_Send_Call 202 rw CS1\pic_send.spd
- ***Virtual*** Procedure PIC_O_Active 206 rw CS1\pic_o_ac.spd
- Procedure PIC_Authorise_Origination_Attempt 207 rw CS1\pic_auth.spd
- Procedure PIC_Select_Route 208 rw CS1\pic_sele.spd
- ***Virtual*** Procedure PIC_O_Alerting 209 rw CS1\pic_o_al.spd
- Procedure PIC_O_Abandon 213 rw CS1\pic_o_ab.spd
- Procedure PIC_Collect_Information 214 rw CS1\pic_coll.spd
- Procedure PIC_Authorise_Call_Setup 217 rw CS1\pic_aut1.spd
- Procedure PIC_O_Answer 218 rw CS1\pic_o_an.spd
- Procedure PIC_OException 219 rw CS1\pic_oexc.spd
- Procedure PIC_Collect_NDigits 220 rw CS1\pic_col1.spd

Procedure PIC_Collect_NDigits 224 rw CS1\pic_coll.spd
Virtual Procedure DP_origAttemptAuthorised 223 rw CS1\dp_origa.spd
Virtual Procedure DP_Route_Select_Failure 226 rw CS1\dp_route.spd
Virtual Procedure DP_oCalledPartyBusy 228 rw CS1\dp_ocall.spd
Virtual Procedure DP_oDisconnect 229 rw CS1\dpdiscon.spd
Virtual Procedure DP_Collected_Info 232 rw CS1\dp_colle.spd
Virtual Procedure DP_oAnswer 235 rw CS1\dp_o_ans.spd
Virtual Procedure DP_oMidCall 236 rw CS1\dp_omidc.spd
Virtual Procedure DP_Analysed_Information 237 rw CS1\dp_analy.spd
Virtual Procedure DP_oNoAnswer 239 rw CS1\dp_onoan.spd
Virtual Procedure DP_oAbandon 240 rw CS1\dp_oaban.spd
Procedure MapToSIRArg 241 rw CS1\maptosir.spd
Procedure MapToDP 242 rw CS1\maptodp.spd
Procedure MapFromPIC 243 rw CS1\mapfpic.spd
Virtual Process Type TerminatingBCSM 244 rw CS1\tcs1.spt
Procedure PIC_T_Null 253 rw CS1\pic_t_nu.spd
Procedure PIC_Select_Facility 254 rw CS1\pic_sel1.spd
Virtual Procedure PIC_tActive 255 rw CS1\pic_tact.spd
Procedure PIC_Authorize_Termination_Attempt 256 rw CS1\pic_aut2.spd
Virtual Procedure PIC_Present_Call 257 rw CS1\pic_pres.spd
Procedure PIC_TException 259 rw CS1\pic_texc.spd
Virtual Procedure PIC_tAlerting 260 rw CS1\pic_tale.spd
Virtual Procedure DP_termAttemptAuthorized 261 rw CS1\dp_terma.spd
Virtual Procedure DP_tNoAnswer 262 rw CS1\dp_tnoan.spd
Virtual Procedure DP_tDisconnect 263 rw CS1\dp_disco.spd
Virtual Procedure DP_tBusy 266 rw CS1\dp_tbusy.spd
Virtual Procedure DP_tMidCall 267 rw CS1\dp_tmidc.spd
Virtual Procedure DP_tAnswer 268 rw CS1\dp_tansw.spd
Virtual Procedure DP_tAbandon 269 rw CS1\dp_taban.spd
Virtual Process Type SSME_FSM 270 rw CS1\ssme_fsm.spt
Procedure ProcessActivateServiceFiltering 275 rw CS1\procasf.spd
Procedure ProcessCallGap 276 rw CS1\proccg.spd
Virtual Procedure InitialiseTDPTable 277 rw CS1\inittdps.spd
Procedure Mgt_SetTriggerTable 278 rw CS1\mgt_stt.spd
Procedure ArmTDPs 279 rw CS1\armstati.spd
Procedure MatchingServiceFilteringCriteria 280 rw CS1\matching.spd
Virtual Procedure CheckACG 281 rw CS1\checkacg.spd
Procedure CallFiltered 282 rw CS1\callfilt.spd
Virtual Block Type TCAP_Simulator 283 rw TCAP\tcaps.sbt

- Process Instance TCAP_D (0) : TCAP_Dialog
- Process Instance TCAP_IH (1,1) : TCAP_InterfaceHandler
- ***Virtual*** Process Type TCAP_InterfaceHandler 285 rw TCAP\tcapi.spt
  - Operator newDialog 297 rw TCAP\newd.sop
  - Operator initDialogs 298 rw TCAP\initd.sop
  - Operator getDialogID 299 rw TCAP\getd.sop
  - Operator nextFreeDialogID 300 rw TCAP\nextfd.sop
- ***Virtual*** Process Type TCAP_Dialog 301 rw TCAP\tcapd.spt
  - Operator newTCmessage 315 rw TCAP\newtcm.sop

Chapter IN CS−2 Specification

- Package CS2_INAP rw CS2\cs2.sun
  - Depending on ASN.1 Text INCS2BundleArg
  - Depending on ASN.1 Text INCS2datatypes
  - Depending on ASN.1 Text INCS2Internals
  - Depending on ASN.1 Text INCS2SSFSCFopsargs
  - System Type CS2_INAP rw CS2\cs2_inap.sst
    - Block Instance SSF_CCF_A
    - Block Instance SSF_CCF_B
    - Block Instance TCAP_Adapter
    - ***Redefined*** Block Type SSF_CCF rw CS2\ccf_ssf.sbt
      - Process Instance CS
      - Process Instance CSA
      - Process Instance IH
      - Process Instance O_BCSM
      - Process Instance SSF
      - Process Instance SSME
      - Process Instance T_BCSM
      - ***Redefined*** Process Type InterfaceHandler rw CS2\interfac.spt
        - Procedure ProcessMoveCallSegments rw CS2\processm.spd
      - ***Redefined*** Process Type CallSegmentAssociation rw CS2\csa2.spt
        - Procedure NoOfSegments rw CS2\noofsegm.spd
        - Procedure ExportLeg rw CS2\exportl1.spd
        - Procedure ExportCS rw CS2\exportcs.spd
        - Procedure ImportLeg rw CS2\importl1.spd
        - Procedure ImportCS rw CS2\importcs.spd
        - Procedure RemoveLeg rw CS2\removele.spd
        - Procedure RemoveCS rw CS2\removecs.spd

- Procedure MoveLegs rw CS2\movelegs.spd
- Procedure ExportControllingLeg rw CS2\expcl.spd
- Procedure ProcessRRBE rw CS2\processr.spd
- Procedure DisconnectControllingLeg rw CS2\dlc.spd
- Procedure ProcessQueuedRRBE rw CS2\processq.spd
- Procedure ReleaseCall rw CS2\release1.spd

Redefined Process Type CallSegment rw CS2\cv2.spt

- Procedure ImportLeg rw CS2\importle.spd
- Procedure BroadcastToLegs rw CS2\broadcas.spd
- Procedure ExportLeg rw CS2\exportle.spd
- Procedure NoOfLegs rw CS2\nooflegs.spd
- Procedure DPFiltering rw CS2\dpfil.spd
- Procedure MapSFToBCSM rw CS2\msftbcsm.spd
- Procedure DPFilteringUTSI rw CS2\dpfilter.spd
- Procedure MapAIToBCSM rw CS2\maitbcsm.spd
- Procedure MapSRToBCSM rw CS2\msrtbcsm.spd

Redefined Process Type SSF_FSM rw CS2\ssf_fsm2.spt

- Procedure ExportEventRecord rw CS2\expevl.spd
- Procedure ProcessContinueWithArgument rw CS2\pcwa.spd
- Procedure ProcessRequestReportUTSI rw CS2\process3.spd
- Procedure ImportEventRecord rw CS2\impevl.spd
- Procedure ProcessDisconnectLeg rw CS2\processd.spd
- Procedure ProcessSendSTUI rw CS2\process4.spd
- Procedure ProcessReportUTSI rw CS2\process5.spd
- ***Redefined*** Procedure ProcessDPSpecific rw CS2\ProcDPS.spd
- Procedure IsUTSIArmed rw CS2\isutsiar.spd
- Procedure ProcessReportFacility rw CS2\prf.spd
- Procedure ProcessRequestReportFacilityEvent rw CS2\prrfe.spd
- Procedure IsFacilityArmed rw CS2\isfa.spd
- Procedure ProcessSendFacility rw CS2\psf.spd
- ***Redefined*** Procedure InitialiseDPTable rw CS2\initiali.spd

Redefined Process Type SSME_FSM rw CS2\ssmefsm.spt

- Procedure ProcessManageTriggerData rw CS2\procmtd.spd
- ***Redefined*** Procedure CheckACG rw CS2\checkac1.spd
- ***Redefined*** Procedure InitialiseTDPTable rw CS2\inittdp2.spd

Redefined Process Type OriginatingBCSM rw CS2\ocs2.spt

- ***Redefined*** Procedure PIC_O_Active rw CS2\pic_o_ac.spd
- ***Redefined*** Procedure PIC_Send_Call rw CS2\pic_send.spd

- Procedure PIC_O_Suspended rw CS2\pic_o_su.spd
- Procedure PIC_O_Retention rw CS2\pic_o_re.spd
- ***Redefined*** Procedure PIC_O_Alerting rw CS2\pic_o_al.spd
- Procedure PIC_Disconnect rw CS2\pic_disc.spd
- ***Redefined*** Procedure DP_origAttemptAuthorised rw CS2\dp_origa.spd
- ***Redefined*** Procedure DP_oAnswer rw CS2\dp_oansw.spd
- ***Redefined*** Procedure DP_oAbandon rw CS2\dp_oaban.spd
- Procedure DP_oSuspended rw CS2\dp_osusp.spd
- ***Redefined*** Procedure DP_Collected_Info rw CS2\dp_colle.spd
- ***Redefined*** Procedure DP_oNoAnswer rw CS2\dp_onoan.spd
- ***Redefined*** Procedure DP_oDisconnect rw CS2\dp_disco.spd
- ***Redefined*** Procedure DP_Analysed_Information rw CS2\dp_analy.spd
- ***Redefined*** Procedure DP_oCalledPartyBusy rw CS2\dp_ocall.spd
- Procedure DP_oReanswer rw CS2\dp_orean.spd
- ***Redefined*** Procedure DP_Route_Select_Failure rw CS2\dp_rout1.spd
- ***Redefined*** Procedure DP_oMidCall rw CS2\dp_omid1.spd
- Procedure DP_origAttempt rw CS2\dp_orig1.spd
- Procedure DP_oTermSeized rw CS2\dp_oterm.spd

Redefined Process Type TerminatingBCSM rw CS2\tcs2.spt

- ***Redefined*** Procedure PIC_Present_Call rw CS2\pic_pres.spd
- Procedure PIC_Disconnect rw CS2\pic_dis2.spd
- ***Redefined*** Procedure PIC_tAlerting rw CS2\pic_tale.spd
- Procedure PIC_T_Suspended rw CS2\picts.spd
- ***Redefined*** Procedure PIC_tActive rw CS2\picta.spd
- ***Redefined*** Procedure DP_termAttemptAuthorized rw CS2\dp_terma.spd
- ***Redefined*** Procedure DP_tMidCall rw CS2\dp_tmidc.spd
- Procedure DP_termAttempt rw CS2\dpta.spd
- Procedure DP_tReanswer rw CS2\dptr.spd
- ***Redefined*** Procedure DP_tBusy rw CS2\dp_tbusy.spd
- ***Redefined*** Procedure DP_tAbandon rw CS2\dp_taban.spd
- Procedure DP_Facility_Selected_and_Available rw CS2\dpfsaa.spd
- ***Redefined*** Procedure DP_tAnswer rw CS2\dp_tansw.spd
- ***Redefined*** Procedure DP_tDisconnect rw CS2\dp_disc1.spd
- Procedure DP_tSuspended rw CS2\dpts.spd
- ***Redefined*** Procedure DP_tNoAnswer rw CS2\dp_tnoan.spd
- Procedure DP_tCall_Accepted rw CS2\dp_tcall.spd

Redefined Block Type TCAP_Simulator rw CS2\tcap.sbt

- Process Instance TCAP_D
- Process Instance TCAP_IH

Process Instance TCAP_IH1
Redefined Process Type TCAP_InterfaceHandler rw CS2\tcapih.spt
Redefined Process Type TCAP_Dialog rw CS2\tcapdia.spt

Package CS2_SRF rw SRF\geinapty.sun
System Type CS2_SRF rw SRF\gen_srf.sst
Block Instance SRFunction : SRF
Virtual Block Type SRF rw SRF\srf.sbt
Process Instance SRM (1,1) : SRFManager
Process Instance SRSM (1,...) : SRFStateModel
Process Type SRFManager rw SRF\srm.spt
Procedure DetermineIndex rw SRF\detindex.spd
Procedure GetPtrFromIndex rw SRF\getptrin.spd
Procedure GetIndexFromPtr rw SRF\getinptr.spd
Process Type SRFStateModel rw SRF\srsm.spt
Procedure SSF_EstablishConnection rw SRF\estconss.spd
Procedure ProcessPlayAnnouncement rw SRF\procpa.spd
Procedure ProcessEraseMessage [unconnected]
Procedure ProcessPromptAndCollectUserInformation rw SRF\procpacu.spd
Procedure ProcessPromptAndReceiveMessage [unconnected]
Procedure ProcessScriptClose [unconnected]
Procedure ProcessScriptInformation [unconnected]
Procedure ProcessScriptRun [unconnected]
Procedure ProcessCancel [unconnected]

Package CS2_Assisting rw Assisting\cs2a.sun
System Type AssistHandOffSSF −− TCAP\AssistHandOffSSF.sst
Virtual Block Type AssistHandOffSSF rw Assisting\ahossf.sbt
Process Instance AHSSF (0) : AssistHandOffSSF
Process Instance IH (1,1) : InterfaceHandler
Virtual Process Type InterfaceHandler [unconnected]
Virtual Process Type AssistHandOffSSF rw Assisting\ahossf.spt
Procedure ProcessInapInstruction [unconnected]

Package CS2_CUSF rw CUSF\generife.sun
System Type CS2_CUSF rw CUSF\generife.sst
Block Instance CUSF_A : CUSF
Block Instance TCAP : TCAP_Adapter
Virtual Block Type CUSF rw CUSF\fe.sbt
Process Instance BCUSM_A (1,...) : BCUSM

- Process Instance CUSF (1,...) : CUSF_FSM
- Process Instance CUSFM : CUSFManager
- ***Virtual*** Process Type CUSFManager rw CUSF\manager.spt
  - Procedure AllocateIndex rw CUSF\alindex.spd
  - Procedure AddIndex rw CUSF\addindex.spd
  - Procedure GetIndexfromInstance rw CUSF\geixfrin.spd
  - Procedure GetInstancefromIndex rw CUSF\geinfrix.spd
  - Procedure GetIndexfromSigConId rw CUSF\geinfrsc.spd
  - Procedure SetSigConAssoc rw CUSF\sesicoas.spd
  - Procedure IsInstance [unconnected]
- ***Virtual*** Process Type CUSF_FSM rw CUSF\cusf_fsm.spt
  - Procedure ProcessActivateReceivedAndAuthorised [unconnected]
  - Procedure ProcessAssociationReleaseRequested [unconnected]
  - Procedure ProcessComponentReceived [unconnected]
  - Procedure ProcessInitiateAssociation [unconnected]
  - Procedure ProcessReleaseAssociation [unconnected]
  - Procedure ProcessRequestReportBCUSMEvent [unconnected]
  - Procedure InitialiseDPTable_A rw CUSF\inevlist.spd
  - Procedure ArmStaticDPs_A rw CUSF\armstadp.spd
  - Procedure AnyDPArmed_A rw CUSF\anevarmd.spd
  - Procedure IsDPArmed_A rw CUSF\isevarmd.spd
  - Procedure DisarmDPs_A rw CUSF\disarmdp.spd
  - Procedure DPArmed_A rw CUSF\dparmed_.spd
  - Procedure GetDPIndex_A rw CUSF\getevind.spd
- ***Virtual*** Process Type BCUSM rw CUSF\bcusm.spt
  - Procedure MapFromPIA [unconnected]
  - Procedure MapToDP [unconnected]
- Block Type TCAP_Adapter [unconnected]

Chapter System Instantiations

System CS1_INAP rw CS1\cs1.ssy

System CS2_INAP rw CS2\cs2_inap.ssy

Chapter CS−1 Examples

MSC CI_Overview rw EXAMPLES\ciov.msc

MSC CI_Detailed rw EXAMPLES\cid.msc

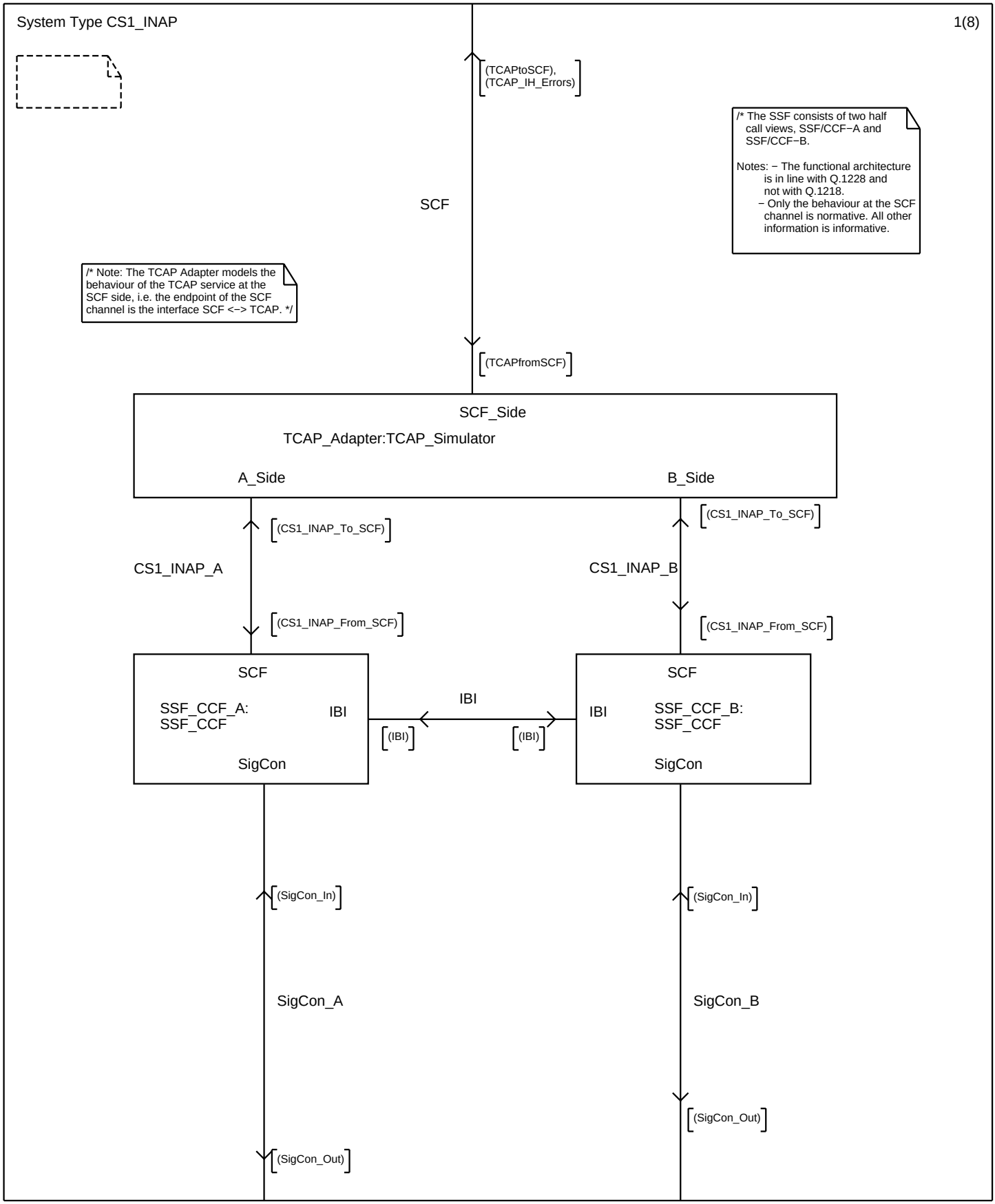
MSC CO_Overview	rw	EXAMPLES\coov.msc
MSC CO_Detailed	rw	EXAMPLES\cod.msc
Chapter CPH Examples		
MSC 3Pty_Overview	rw	EXAMPLES\3ptyov.msc
MSC 3Pty_Detailed	rw	EXAMPLES\3ptyd.msc
MSC CF_Overview	rw	EXAMPLES\cfov.msc
MSC CF_Detailed	rw	EXAMPLES\cfd.msc
Chapter CS−1 Test Purposes		

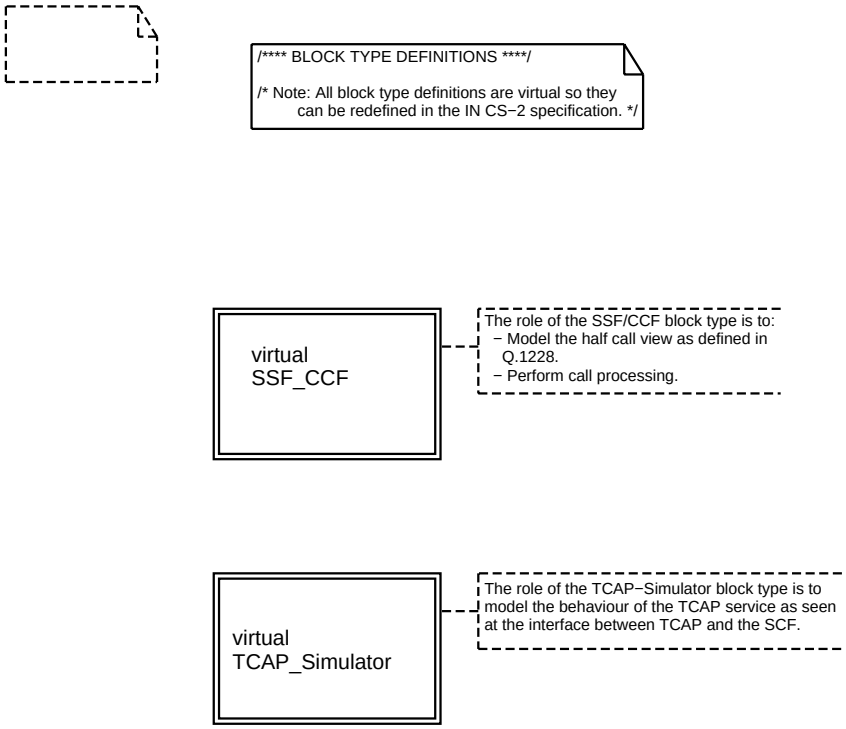
```
USE INCS2datatypes;
USE INCS2SSFSCFopsargs;
USE INCS2BundleArg;
/* Note: Import the ASN.1 modules. */
```

package CS1_INAP

system

CS1_INAP


System Type CS1_INAP
1(8)
(TCAPtoSCF),
(TCAP_IH_Errors)
/* The SSF consists of two half call views, SSF/CCF−A and SSF/CCF−B.
Notes: − The functional architecture is in line with Q.1228 and not with Q.1218.
− Only the behaviour at the SCF channel is normative. All other information is informative.
SCF
/* Note: The TCAP Adapter models the behaviour of the TCAP service at the SCF side, i.e. the endpoint of the SCF channel is the interface SCF <−> TCAP. */
(TCAPfromSCF)
SCF_Side
TCAP_Adapter:TCAP_Simulator
A_Side
B_Side
(CS1_INAP_To_SCF)
(CS1_INAP_To_SCF)
CS1_INAP_A
CS1_INAP_B
(CS1_INAP_From_SCF)
(CS1_INAP_From_SCF)
SCF
SSF_CCF_A:
SSF_CCF
IBI
SigCon
IBI
(IBI)
(IBI)
IBI
SCF
SSF_CCF_B:
SSF_CCF
SigCon
(SigCon_In)
(SigCon_In)
SigCon_A
SigCon_B
(SigCon_Out)
(SigCon_Out)


/**** BLOCK TYPE DEFINITIONS ****/
/* Note: All block type definitions are virtual so they can be redefined in the IN CS−2 specification. */
virtual SSF_CCF
The role of the SSF/CCF block type is to:
− Model the half call view as defined in Q.1228.
− Perform call processing.
virtual TCAP_Simulator
The role of the TCAP−Simulator block type is to model the behaviour of the TCAP service as seen at the interface between TCAP and the SCF.

```
/**** SIGNAL DEFINITIONS FOR THE IN CS−1 SCF−SSF OPERATIONS. ****/

/* Note: − Connect is defined as Connnect because
         Connect is a reserved word in SDL.
*/
```

```
SIGNAL
  ActivateServiceFiltering(InvokeID,ActivateServiceFilteringArg),
  ActivityTest(InvokeID, DialogIDType),
  ActivityTestResult(InvokeID, DialogIDType),
  AnalysedInformation(CSAID, AnalysedInformationArg),
  AnalyseInformation(InvokeID,CSAID,AnalyseInformationArg),
  ApplyCharging(InvokeID,CSAID,ApplyChargingArg),
  ApplyChargingReport(CSAID,ApplyChargingReportArg, Boolean),
  AssistRequestInstructions(CSAID,AssistRequestInstructionsArg),
  CallGap(InvokeID,CallGapArg),
  CallInformationReport(CSAID,CallInformationReportArg,Boolean),
  CallInformationRequest(InvokeID,CSAID,CallInformationRequestArg),
  Cancel(InvokeID,CSAID,CancelArg),
  CancelStatusReportRequest(InvokeID,CancelStatusReportRequestArg),
  CollectInformation(InvokeID,CSAID,CollectInformationArg),
  CollectedInformation(CSAID,CollectedInformationArg),
  Connnect(InvokeID,CSAID,ConnectArg),
  ConnectToResource(InvokeID,CSAID,ConnectToResourceArg),
  Continue(InvokeID,CSAID),
  DisconnectForwardConnection(InvokeID,CSAID),
  EstablishTemporaryConnection(InvokeID,CSAID,EstablishTemporaryConnectionArg),
  EventNotificationCharging(CSAID,EventNotificationChargingArg),
  EventReportBCSM(CSAID,EventReportBCSMArg),
  FurnishChargingInformation(InvokeID,CSAID,FurnishChargingInformationArg),
  HoldCallInNetwork(InvokeID,CSAID,HoldCallInNetworkArg),
  InitialDP(CSAID,InitialDPArg),
  InitiateCallAttempt(InvokeID,CSAID,InitiateCallAttemptArg),
  OAbandon(CSAID,OAbandonArg),
  OAnswer(CSAID,OAnswerArg),
  OCalledPartyBusy(CSAID,OCalledPartyBusyArg),
  ODisconnect(CSAID,ODisconnectArg),
  OMidCall(CSAID,MidCallArg),
  ONoAnswer(CSAID,ONoAnswerArg),
  OriginationAttemptAuthorized(CSAID,OriginationAttemptAuthorizedArg),
  ReleaseCall(InvokeID,CSAID,ReleaseCallArg),
  RequestCurrentStatusReport(InvokeID,RequestCurrentStatusReportArg),
  RequestEveryStatusChangeReport(InvokeID,RequestEveryStatusChangeReportArg),
  RequestFirstStatusMatchReport(InvokeID,RequestFirstStatusMatchReportArg),
  RequestNotificationChargingEvent(InvokeID,CSAID,RequestNotificationChargingEventArg),
  RequestReportBCSMEvent(InvokeID,CSAID,RequestReportBCSMEventArg),
  ResetTimer(InvokeID,CSAID,ResetTimerArg),
  RouteSelectFailure(CSAID,RouteSelectFailureArg),
  SelectFacility(InvokeID,CSAID,SelectFacilityArg),
  SelectRoute(InvokeID,CSAID, SelectRouteArg),
  SendChargingInformation(InvokeID,CSAID,SendChargingInformationArg),
  ServiceFilteringResponse(CSAID,ServiceFilteringResponseArg),
  StatusReport(CSAID,StatusReportArg),
  TAnswer(CSAID,TAnswerArg),
  TBusy(CSAID,TBusyArg),
  TDisconnect(CSAID,TDisconnectArg),
  TermAttemptAuthorized(CSAID,TermAttemptAuthorizedArg),
  TMidCall(CSAID,MidCallArg),
  TNoAnswer(CSAID,TNoAnswerArg),

  Mgt_SetTriggerTable(MGT_SetTriggerTableArg); /* Informative signal, used to model the management of the trigger table. */
```

/* Data type definitions used by the TCAP Simulator */

```
/* Needed for the initialization of the TCAP Simulator */
NEWTYPE IHroleType
  LITERALS
    A_Side, B_Side;
ENDNEWTYPE;

/* Dialog IDs */
/* − Total valid numbers 1 − 100 */
/* − Direction SCF −> SSF: 1 − 50 */
/* − Direction SSF −> SCF: 51 − 100 */
/* − 0 is used as flag, if something strange happened, e.g., for a given csaID no dialogID is found */
SYNONYM maxDialogIDtotal Integer = 100;
SYNONYM maxDialogIDtoSSF Integer = 50;
SYNTYPE DialogIDtype = Integer CONSTANTS 0:maxDialogIDtotal
ENDSYNTYPE;

/* Invoke IDs */
/* − Total valid numbers 1 − 200 */
/* − Direction SCF −> SSF: 1 − 100 */
/* − Direction SSF −> SCF: 101 − 200 */
/* − 0 is used as dummy */
SYNONYM maxInvokeIDtotal Integer = 200;
SYNONYM maxInvokeIDtoSSF Integer = 100;
SYNTYPE InvokeIDtype = Integer CONSTANTS 0:maxInvokeIDtotal
ENDSYNTYPE;

/* For indicating the origin of TCAP messages */
NEWTYPE TCoriginType
  LITERALS
    oSSF, oSCF;
ENDNEWTYPE;

/* refers to basic and prearranged end in TC_EndReq/Ind primitives */
NEWTYPE TCAPterminationType
  LITERALS
    basic, prearranged;
ENDNEWTYPE;

/* For Timeout values for operations, mandatory parameter in TC_InvokeReq primitives */
NEWTYPE TimeoutValType
  LITERALS
    short, medium, long;
ENDNEWTYPE;
```

```
/* Reasons for a TCAP failure */
NEWTYPE TCAPfailReasonType
  LITERALS
    wrongDiaID, /* incorrect dialog ID: incorrect range or no dialog for ID present/available */
    wrongInvokeID, /* incorrect invoke ID */
    wrongCSaID, /* incorrect CSA ID */
    beginRequired, /* TC_BeginReq is required */
    noComp, /* TC_BeginReq without components */
    unknownOP, /* Unknown operation code (within TC Invoke) */
    noDiaIDavail, /* No dialogID available, 'maxDialogIDtoSCF' Diallogs are established */
    cSAnotAvail, /* the provide CSAid is not available */
    unexpSignal, /* the TCAP IH received an unexpected signal */
    signalNotProcessed, /* signal cannot be processed by the TCAP signal handler */
    preArrangedEndReq; /* End signal with wrong termination parameter */
ENDNEWTYPE;
```

```
/* Operation Codes */
NEWTYPE OpCodeType
  LITERALS
  /* No Operation */ NoOperation,
  /* CLASS 1 */ CSA, DL, MTD, MC, MCS, ML, RCS, RES, SL,
  /* CLASS 2 */ ASF, AC, CIRQ, CAN, CSR, CI, CON, CTR, CWA,
              DFC, DFCWA, ETC, FCI, ICA, RE, RFS, RNC, RRB, RRU, RT, SCI, SS,
  /* CLASS 4 */ AT, CG, CUE, RC,
  /* FROM SSF */ ACR, ARI, CIR, IDP, ER, ENC, ERB, RU, SFR, SRP,
  /* RETURN RESULTS FROM SSF */ AT_R, ASF_R, SL_R, DL_R, MCS_R, MC_R,
              ML_R, RCS_R, RES_R, MTD_R, CSA_R;
ENDNEWTYPE;

/* Operation Classes */
SYNTYPE OpClassType = Integer CONSTANTS 0:4
ENDSYNTYPE;
/*− 0 means no class */
```

```
/* SIGNAL Definitions for the TCAP Adapter*/
```

```
/* No Operations, but additional Signals exchanged between TCAP Adapter, SSF and SCF */

SIGNAL
  CSalDreq(DialogIDtype),
  CSalDresp(DialogIDtype,CSAID),
  RegisterSSFreq(IHroleType),
  RegisterSSFresp(IHroleType),
  TCAPFailureInd(TCAPfailReasonType);
```

```
/* TCAP Primitives */

SIGNAL
/* TC_nameReq: Direction SCF−>TCAP */
  TC_InvokeReq(InvokeIDtype,DialogIDtype,OpClassType,OpCodeType,TimeoutValType,ArgType),
  TC_BeginReq(DialogIDtype,TCoriginType),
  TC_ContinueReq(DialogIDtype,TCoriginType),
  TC_EndReq(DialogIDtype,TCAPterminationType),
  TC_AbortReq(DialogIDtype),

/* TC_nameInd: Direction TCAP−>SCF */
  TC_BeginInd(DialogIDtype,TCoriginType,Boolean),
  TC_ContinueInd(DialogIDtype,TCoriginType,Boolean),
  TC_InvokeInd(InvokeIDtype,DialogIDtype,OpCodeType,Boolean,ArgType),
  TC_EndInd(DialogIDtype,TCAPterminationType,Boolean),
  TC_AbortInd(DialogIDtype),
  TC_ErrorInd(InvokeIDtype,DialogIDtype,Boolean),
  TC_ReturnResultInd(InvokeIDtype,DialogIDtype,Boolean,opCodeType,ArgType);
```

```
/**** SIGNAL DEFINITIONS FOR THE IN CS−1 ERROR AND DIALOUGE HANDLING ****/

/* Note: Used between the HalfCalls (SSF/CCF_A and SSF/CCF_B) and the TCAP Adaptor. */
```

```
/* Type definition used for the return error. The values
correspond to the error codes defined in the ASN.1 */
SYNTYPE ErrorArg = NATURAL
  CONSTANTS 0:23
ENDSYNTYPE;
```

```
SIGNAL
  Errror(InvokeID,CSAID,ErrorArg), /* Return error. */
  ApplicationBegin(CSAID),        /* Begin a dialouge. */
  ApplicationContinue(CSAID),
  ApplicationAbort(CSAID),        /* Abort a dialouge. */
  ApplicationEnd(Boolean,CSAID);  /* End a dialouge
                                     indicating whether prearranged end or not. */
```

```
/* SCF to TCAP */

SIGNALLIST TCAPfromSCF =
  TC_InvokeReq,
  TC_BeginReq,
  TC_ContinueReq,
  TC_EndReq,
  TC_AbortReq;
```

```
/* TCAP to SCF */

SIGNALLIST TCAPtoSCF =
  TCAPFailureInd,
  TC_EndInd,
  TC_AbortInd,
  TC_BeginInd,
  TC_ContinueInd,
  TC_InvokeInd,
  TC_ErrorInd,
  TC_ReturnResultInd;
```

```
/* Signallists for the gate definitions */

SIGNALLIST S1 = (TCAPtoSCF), (TCAP_IH_Errors);
SIGNALLIST S2 = (TCAPfromSCF);
SIGNALLIST A1 = (CS1_INAP_From_SCF), (TC_IH_To_SSF);
SIGNALLIST A2 = (CS1_INAP_To_SCF);
SIGNALLIST B1 = (A1);
SIGNALLIST B2 = (A2);
```

```
/* TCAP IH error Messages (to SCF)  */

SIGNALLIST TCAP_IH_Errors = TCAPFailureInd, TC_AbortInd;
```

```
/* TCAP interface to SSF */

SIGNALLIST TC_IH_To_SSF =
  RegisterSSFreq, /* SSF side registration request */
  CSaIDreq; /* request for a CSA ID */
```

```
/***** SIGNAL/PRIMITIVE LIST DEFINITIONS FOR THE IN CS−1 SCF−SSF INTERFACE *****/
```

```
/* SSF −> SCF */

SIGNALLIST CS1_INAP_To_SCF =
  ApplyChargingReport,
  AssistRequestInstructions,
  ActivityTestResult,
  CallInformationReport,
  EventNotificationCharging,
  EventReportBCSM,
  InitialDP,
  ServiceFilteringResponse,

  Errror,
  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd,

/* Signals used to pass information to/from TCAP. */
  RegisterSSFresp,
  CSaIDresp;
```

```
/* SCF −> SSF */

SIGNALLIST CS1_INAP_From_SCF =
  ActivateServiceFiltering,
  ActivityTest,
  AnalyseInformation,
  ApplyCharging,
  CallGap,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continue,
  DisconnectForwardConnection,
  EstablishTemporaryConnection,
  FurnishChargingInformation,
  InitiateCallAttempt,
  ReleaseCall,
  RequestNotificationChargingEvent,
  RequestReportBCSMEvent,
  ResetTimer,
  SelectFacility,
  SelectRoute,
  SendChargingInformation,

  Mgt_SetTriggerTable, /* Operation on the management interface, informative. */


  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd,

/* Extensions due to the needs of the TCAP Adapter */
  RegisterSSFreq,
  CSaIDreq;
```

```
/**** DEFINITION OF THE TYPES AND PRIMITIVES FOR THE SIGNALLING CONTROL INTERFACE ****/

/* Note: The signalling control interface is a generic interface that can be
mapped to e.g. ISUP and DSS.1 */
```

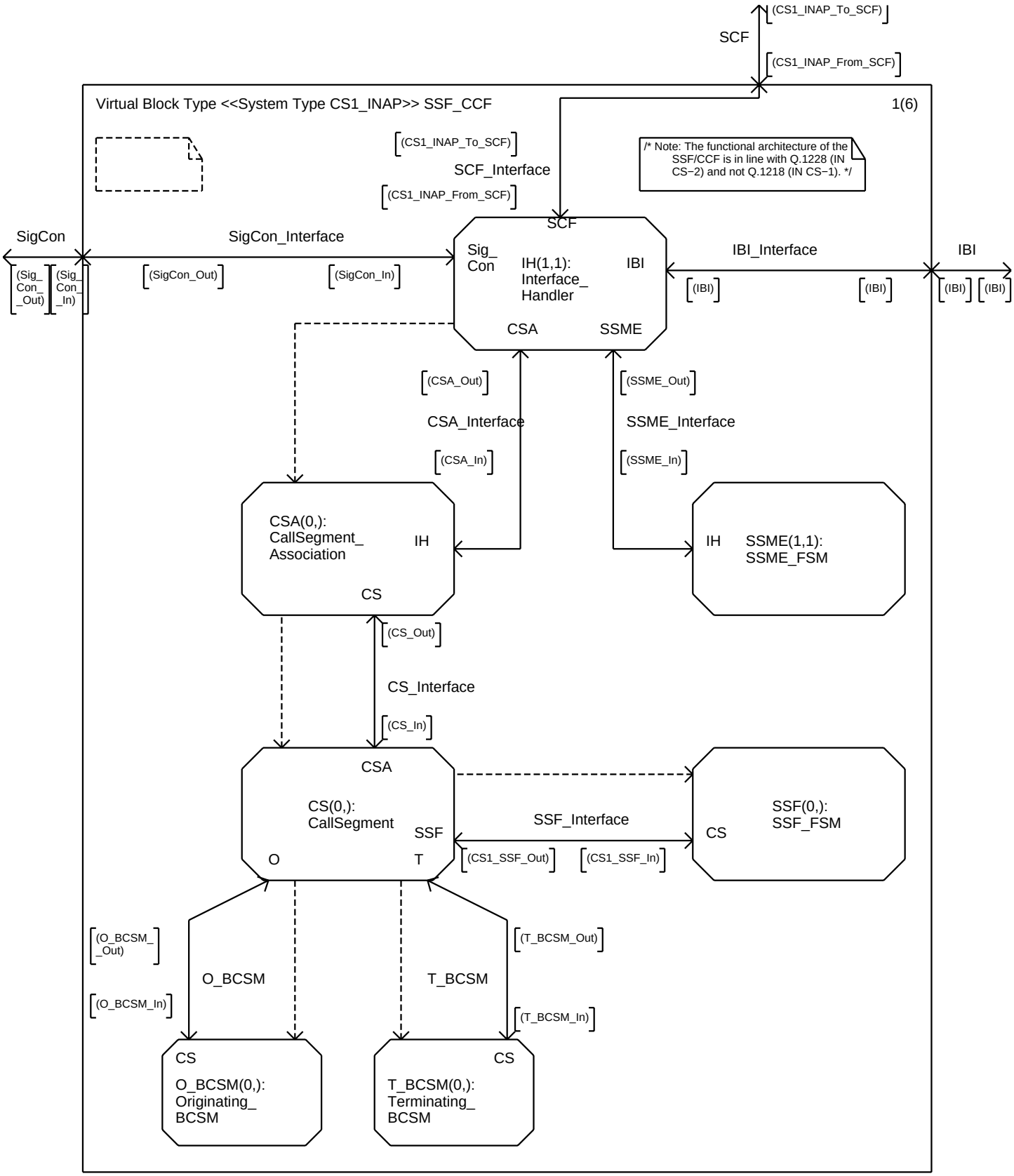
```
/* Definition of primitives for the
Signalling Control (SigCon) interfaces. */

SIGNAL
  AddressEndInd(AddressEndType),
  CallProgressInd(CallProgressType),
  CallProgressReq(CallProgressType),
  FailureInd(FailureType),
  ReleaseInd(ReleaseType),
  ReleaseReq(ReleaseType,CallFlag),
  ServiceFeatureInd(ServiceFeatureType),
  SetupConf(SetupCRType),
  SetupInd(SetupIRType),
  SetupReq(SetupIRType),
  SetupResp(SetupCRType),
  SubsequentAddressInd(SubsequentAddressType),
  SubsequentAddressReq(SubsequentAddressType);
```

```
/* From SigCon to O−BCSM */

SIGNALLIST O_SigCon_In =
  AddressEndInd,
  FailureInd,
  ReleaseInd,
  ServiceFeatureInd,
  SetupInd,
  SubsequentAddressInd;

/* from O−BCSM to SigCon */

SIGNALLIST O_SigCon_Out =
  CallProgressReq,
  ReleaseReq,
  SetupResp,
  SubsequentAddressReq;

/* from T−BCSM to SigCon */

SIGNALLIST T_SigCon_Out =
  ReleaseReq,
  SetupReq,
  SubsequentAddressReq;

/* from called agent to T−BCSM */

SIGNALLIST T_SigCon_In =
  CallProgressInd,
  FailureInd,
  ReleaseInd,
  ServiceFeatureInd,
  SetupConf;
```

```
SIGNALLIST SigCon_In =
  (O_SigCon_In),
  (T_SigCon_In);

SIGNALLIST SigCon_Out =
  (O_SigCon_Out),
  (T_SigCon_Out);
```

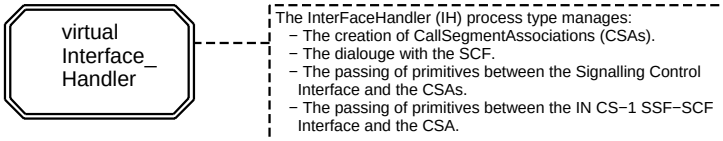
```
/* Definition of primitives for the internal
interface between BCSMs (IBI). */

SIGNAL
  CallProgressReqInd(CallProgressType,CSAID,LegType),
  ReleaseReqInd(ReleaseType,CSAID,LegType, CallFlag),
  SetupReqInd(SetupIRType,CSAID,LegType),
  SetupRespConf(SetupCRType,CSAID,LegType),
  SubsequentAddressReqInd(SubsequentAddressType,CSAID,LegType);
```

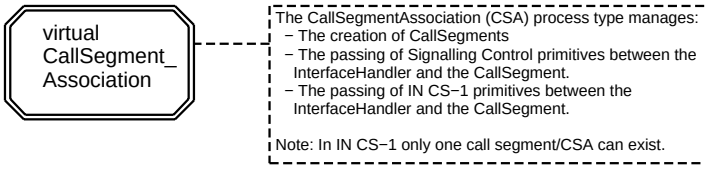
```
/* Terminating to Originating BCSM */

SIGNALLIST IBI_In =
  CallProgressReqInd,
  ReleaseReqInd,
  SetupRespConf;

/* Originating to Terminating BCSM */

SIGNALLIST IBI_Out =
  ReleaseReqInd,
  SetupReqInd,
  SubsequentAddressReqInd;

SIGNALLIST IBI =
(IBI_In),
(IBI_Out);
```

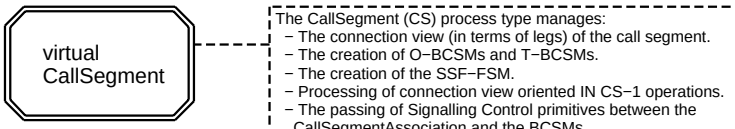

(CS1_INAP_To_SCF)
SCF
(CS1_INAP_From_SCF)
Virtual Block Type <<System Type CS1_INAP>> SSF_CCF
1(6)
(CS1_INAP_To_SCF)
SCF_Interface
(CS1_INAP_From_SCF)
/* Note: The functional architecture of the SSF/CCF is in line with Q.1228 (IN CS−2) and not Q.1218 (IN CS−1). */
SigCon
SigCon_Interface
(Sig_ Con_ _Out)
(Sig_ Con_ _In)
(SigCon_Out)
(SigCon_In)
SCF
Sig_ Con
IH(1,1): Interface_ Handler
IBI
CSA
SSME
IBI_Interface
IBI
(IBI)
(IBI)
(IBI)
(IBI)
(CSA_Out)
CSA_Interface
(CSA_In)
(SSME_Out)
SSME_Interface
(SSME_In)
CSA(0,): CallSegment_ Association
IH
CS
IH
SSME(1,1): SSME_FSM
(CS_Out)
CS_Interface
(CS_In)
CSA
CS(0,): CallSegment
SSF
O
T
SSF_Interface
(CS1_SSF_Out)
(CS1_SSF_In)
CS
SSF(0,): SSF_FSM
(O_BCSM_ _Out)
(O_BCSM_In)
O_BCSM
T_BCSM
(T_BCSM_Out)
(T_BCSM_In)
CS
O_BCSM(0,): Originating_ BCSM
CS
T_BCSM(0,): Terminating_ BCSM

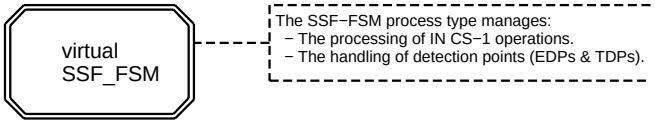
/**** PROCESS TYPE DEFINITIONS ****/

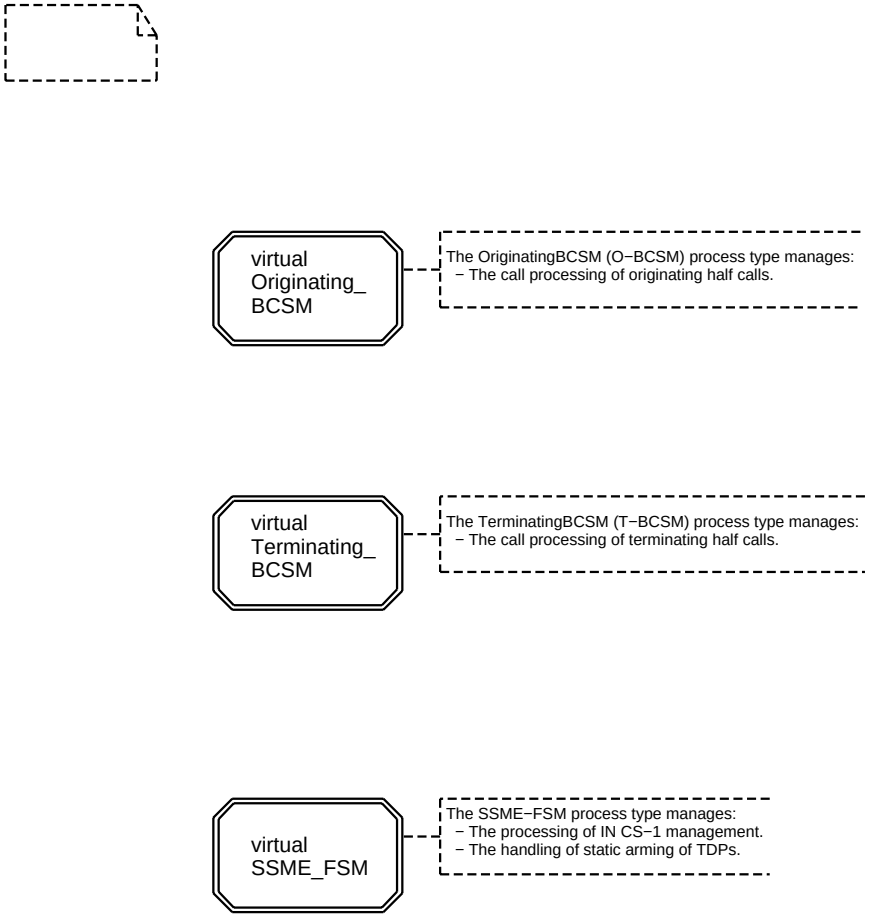
/* Note: All process type definitions are virtual so they can be redefined in the IN CS−2 specification. */










virtual
Originating_
BCSM
The OriginatingBCSM (O−BCSM) process type manages:
− The call processing of originating half calls.
virtual
Terminating_
BCSM
The TerminatingBCSM (T−BCSM) process type manages:
− The call processing of terminating half calls.
virtual
SSME_FSM
The SSME−FSM process type manages:
− The processing of IN CS−1 management.
− The handling of static arming of TDPs.

```
/**** SIGNAL DEFINITIONS FOR DETECTION POINTS ****/

/* Note:   Detection points are sent from a BCSM to the SSF−FSM
           via the CallSegment. */
```

```
SIGNAL
  DP(DPArg),
  DPAbandon(DPArg),
  DPDisconnect(DPArg),
  DPMidCall(DPArg);

/* Signals from O−BCSM to SSF */
SIGNALLIST oDPs =
  DP,
  DPAbandon,
  DPDisconnect,
  DPMidCall;

/* Signals from T−BCSM to SSF */
SIGNALLIST tDPs =
  DP,
  DPAbandon,
  DPDisconnect,
  DPMidCall;

SIGNALLIST DPs =
  (oDPs),
  (tDPs);
```

```
/**** SIGNAL DEFINITIONS FOR POINTS IN CALL ****/

/* Note: − Points in call are sent from the SSF−FSM to a BCSM
          via the CallSegment. */
```

```
SIGNAL
  PIC(PICArg),
  PICResume(LegType, Boolean); /* The second parameter is used in connection with ContinueWithArgument (IN CS−2 operation). */

/* from SSF to BCSM */
SIGNALLIST oPICs =
  PIC,
  PICResume;

SIGNALLIST tPICs =
  PIC,
  PICResume;

SIGNALLIST PICs =
  (oPICs),
  (tPICs);
```

```
/**** DEFINITIONS COMMON TO THE IH, CSA, CS, BCSM AND SSF−FSM PROCESSES. */
```

```
/* Definition of Leg */

NEWTYPE LegStatusType
  LITERALS joined, pending, shared, surrogate;
ENDNEWTYPE;

NEWTYPE BCSMType
  LITERALS Originating, Terminating;
ENDNEWTYPE;

NEWTYPE LegInfo STRUCT
  Used Boolean;            /* Flag indicating whether the leg is in use or not. */
  LegPtr PId;              /* Pointer/Address to the BCSM connected to the leg. */
  BCSM BCSMType;           /* Type (O or T) of BCSM connected to the (passive) leg. */
  LegStatus LegStatusType;   /* Status of the leg. */
ENDNEWTYPE;

NEWTYPE LegArray  /* Array of legs. The LegID of a leg is the index in the array. */
  ARRAY (LegType,LegInfo)
ENDNEWTYPE;
```

```
/* The SSF−FSM and the SSME maintains an Event List for
each LegId. The event list contains the  BCSM event type
and the monitor mode. */


NEWTYPE EventTableType
  ARRAY(LegType,EventRecordType)
ENDNEWTYPE;

NEWTYPE EventRecordType
  ARRAY(EventTypeBCSM,ServiceKeyEventType)
ENDNEWTYPE;

NEWTYPE ServiceKeyEventType
  ARRAY(ServiceKey, BCSMEvent)
ENDNEWTYPE;
```

```
/* An SSF−FSM can start from either the Idle state (initial SSF−FSM)
or from the WFI state (additional SSF−FSMs). */

NEWTYPE SSFStateType
  LITERALS
    Idle,
    WaitingForInstruction;
ENDNEWTYPE;


/* The SSF FSM maintains an Event List for
each LegId. The event list contains the
BCSM event type and the monitor mode. */

SYNTYPE NumberOfBCSMEvents = NATURAL
  CONSTANTS 1:27
ENDSYNTYPE;

SYNONYM numOfBCSMEvents NATURAL = 27;

NEWTYPE EventListType
  ARRAY(NumberOfBCSMEvents,BCSMEvent)
ENDNEWTYPE;

SYNTYPE MaxNumOfLegs = NATURAL
  CONSTANTS 1:numOfLegs
ENDSYNTYPE;

NEWTYPE LegEventListType
  ARRAY(MaxNumOfLegs,EventListType)
ENDNEWTYPE;
```

```
/* Signals used for passing information between the CSA, CS, BCSM and SSF−FSM. */

SIGNAL
  BCSMStop(LegType,PartyType),  /* BCSM −> CS: indication that the BCSM has stopped.*/
  CSStop(CallSegmentID,Cause),  /* CS −> CSA: indication that the CS has stopped. */
  SSFStop;                      /* CS −> SSF−FSM: instruction to stop the SSF−FSM. */
```

```
/* Signals used for passing information between the SSF−FSM and the SSME−FSM. */
SIGNAL
  ArmTDPsReq(CSAID,CallSegmentID),
  ArmTDPsResp(CSAID,CallSegmentID,EventTableType),
  MSFCReq(CSAID,CallSegmentID),
  MSFCResp(CSAID,CallSegmentID,Boolean),
  CACGReq(CSAID,CallSegmentID),
  CACGResp(CSAID,CallSegmentID,Boolean),
  CFReq(CSAID,CallSegmentID),
  CFResp(CSAID,CallSegmentID,Boolean);

SIGNALLIST SSME_Reqs =
  ArmTDPsReq,
  MSFCReq,
  CACGReq,
  CFReq;

SIGNALLIST SSME_Resps =
  ArmTDPsResp,
  MSFCResp,
  CACGResp,
  CFResp;
```

/**** SIGNAL LIST DEFINITIONS ****/

```
SIGNALLIST CS1_INAP_CSA_In =
  AnalyseInformation,
  ApplyCharging,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continue,
  DisconnectForwardConnection,
  EstablishTemporaryConnection,
  FurnishChargingInformation,
  InitiateCallAttempt,
  ReleaseCall,
  RequestNotificationChargingEvent,
  RequestReportBCSMEvent,
  ResetTimer,
  SelectFacility,
  SelectRoute,
  SendChargingInformation,

  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;

SIGNALLIST CS1_INAP_CSA_Out =
  ApplyChargingReport,
  AssistRequestInstructions,
  CallInformationReport,
  EventNotificationCharging,
  EventReportBCSM,
  InitialDP,

  Errror,
  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;
```

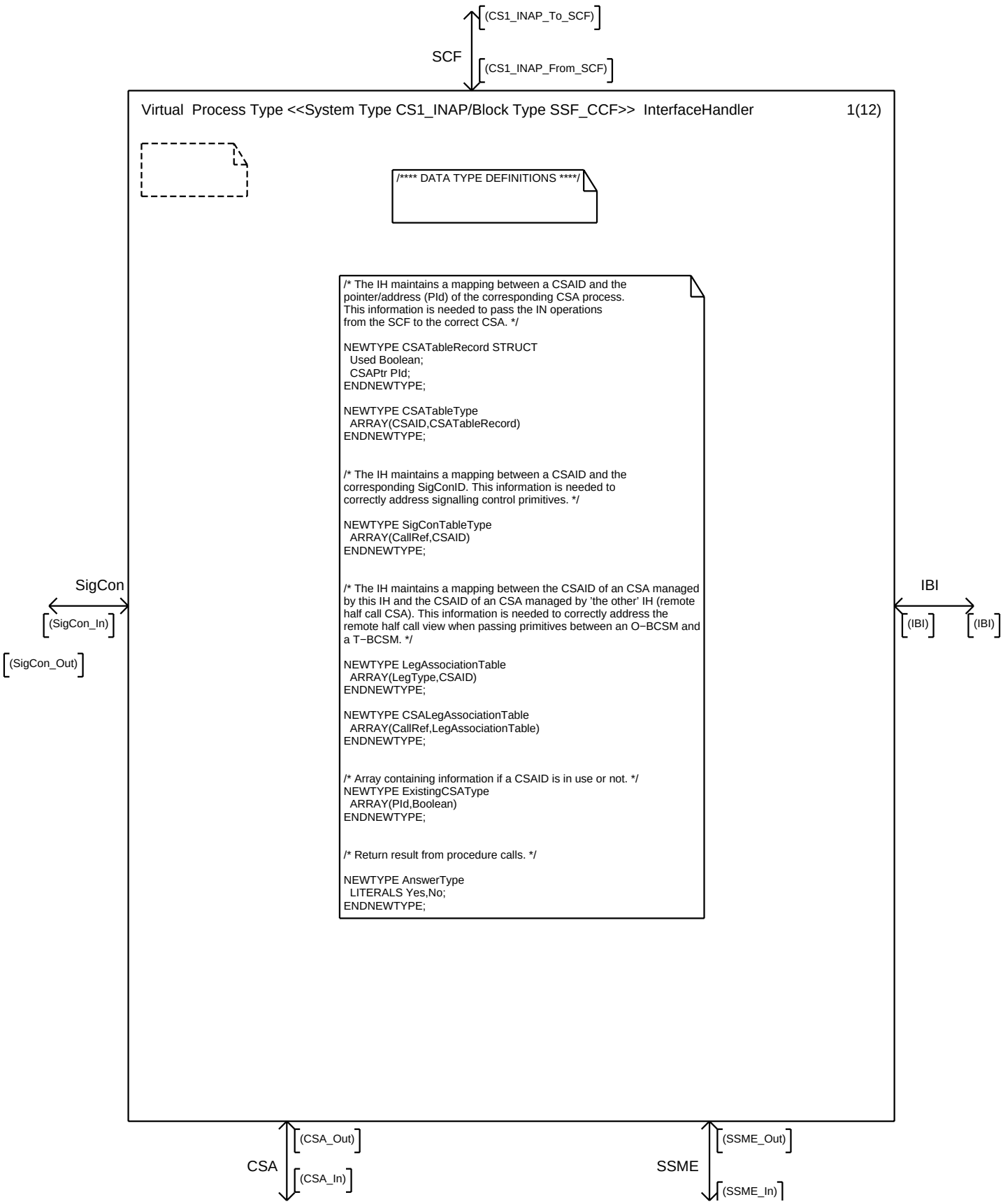
```
SIGNALLIST CS1_INAP_CS_In =
  AnalyseInformation,
  ApplyCharging,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continue,
  DisconnectForwardConnection,
  EstablishTemporaryConnection,
  FurnishChargingInformation,
  InitiateCallAttempt,
  ReleaseCall,
  RequestNotificationChargingEvent,
  RequestReportBCSMEvent,
  ResetTimer,
  SelectFacility,
  SelectRoute,
  SendChargingInformation,

  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;

SIGNALLIST CS1_INAP_CS_Out =
  ApplyChargingReport,
  AssistRequestInstructions,
  CallInformationReport,
  EventNotificationCharging,
  EventReportBCSM,
  InitialDP,

  Errror,
  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;
```

```
SIGNALLIST CS1_INAP_SSF_In =
  AnalyseInformation,
  ApplyCharging,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continue,
  DisconnectForwardConnection,
  EstablishTemporaryConnection,
  FurnishChargingInformation,
  InitiateCallAttempt,
  ReleaseCall,
  RequestNotificationChargingEvent,
  RequestReportBCSMEvent,
  ResetTimer,
  SelectFacility,
  SelectRoute,
  SendChargingInformation,

  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;

SIGNALLIST CS1_INAP_SSF_Out =
  ApplyChargingReport,
  AssistRequestInstructions,
  CallInformationReport,
  EventNotificationCharging,
  EventReportBCSM,
  InitialDP,

  Errror,
  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;
```

```
SIGNALLIST SSME_In =
  ActivateServiceFiltering,
  ActivityTest,
  CallGap,

  (SSME_Reqs),

  Mgt_SetTriggerTable;

SIGNALLIST SSME_Out =
  ActivityTestResult,
  ServiceFilteringResponse,

  (SSME_Resps);
```

```
SIGNALLIST CSA_In =
  (CS1_INAP_CSA_In),
  (SigCon_In),
  (IBI),
  (SSME_Resps);

SIGNALLIST CSA_Out =
  (CS1_INAP_CSA_Out),
  (SigCon_Out),
  (IBI),
  (SSME_Reqs);

SIGNALLIST CS_In =
  (CS1_INAP_CS_In),
  (SigCon_In),
  (IBI),
  (SSME_Resps),
  CSStop;

SIGNALLIST CS_Out =
  (CS1_INAP_CS_Out),
  (SigCon_Out),
  (IBI),
  (SSME_Reqs),
  CSStop;

SIGNALLIST CS1_SSF_In =
  (DPs),
  (CS1_INAP_SSF_In),
  (SSME_Resps),
  SSFStop;

SIGNALLIST CS1_SSF_Out =
  (PICs),
  (CS1_INAP_SSF_Out),
```

```
SIGNALLIST O_BCSM_In =
  (oPICs),
  (O_SigCon_In),
  (IBI_In),
  BCSMStop;

SIGNALLIST O_BCSM_Out =
  (oDPs),
  (O_SigCon_Out),
  (IBI_Out),
  BCSMStop;

SIGNALLIST T_BCSM_In =
  (tPICs),
  (T_SigCon_In),
  (IBI_Out),
  BCSMStop;

SIGNALLIST T_BCSM_Out =
  (tDPs),
  (T_SigCon_Out),
  (IBI_In),
  BCSMStop;
```


(CS1_INAP_To_SCF)
SCF
(CS1_INAP_From_SCF)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler
1(12)
/**** DATA TYPE DEFINITIONS ****/
/* The IH maintains a mapping between a CSAID and the pointer/address (PId) of the corresponding CSA process. This information is needed to pass the IN operations from the SCF to the correct CSA. */
NEWTYPE CSATableRecord STRUCT
Used Boolean;
CSAPtr PId;
ENDNEWTYPE;
NEWTYPE CSATableType
ARRAY(CSAID,CSATableRecord)
ENDNEWTYPE;
/* The IH maintains a mapping between a CSAID and the corresponding SigConID. This information is needed to correctly address signalling control primitives. */
NEWTYPE SigConTableType
ARRAY(CallRef,CSAID)
ENDNEWTYPE;
/* The IH maintains a mapping between the CSAID of an CSA managed by this IH and the CSAID of an CSA managed by 'the other' IH (remote half call CSA). This information is needed to correctly address the remote half call view when passing primitives between an O−BCSM and a T−BCSM. */
NEWTYPE LegAssociationTable
ARRAY(LegType,CSAID)
ENDNEWTYPE;
NEWTYPE CSALegAssociationTable
ARRAY(CallRef,LegAssociationTable)
ENDNEWTYPE;
/* Array containing information if a CSAID is in use or not. */
NEWTYPE ExistingCSAType
ARRAY(PId,Boolean)
ENDNEWTYPE;
/* Return result from procedure calls. */
NEWTYPE AnswerType
LITERALS Yes,No;
ENDNEWTYPE;
SigCon
(SigCon_In)
(SigCon_Out)
IBI
(IBI)
(IBI)
(CSA_Out)
CSA
(CSA_In)
(SSME_Out)
SSME
(SSME_In)

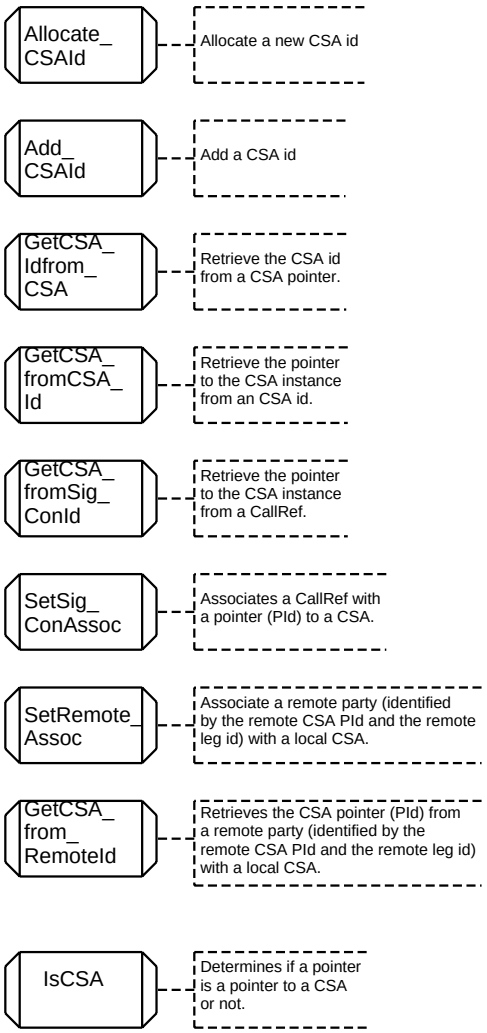
```
/**** VARIABLE DECLARATIONS ****/
```

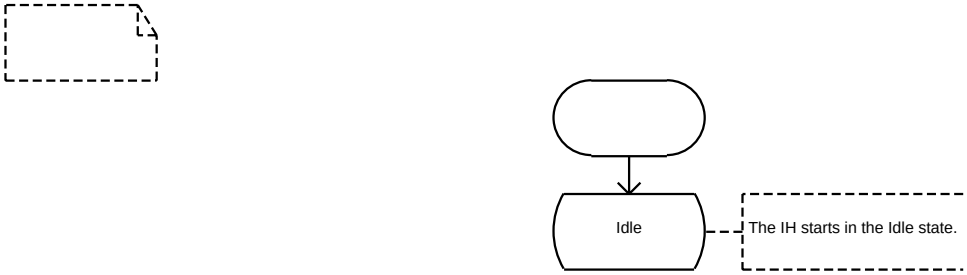
```
DCL
 /* Routing tables. */
 csaTable CSATableType,
 sigConTable SigConTableType,
 csaLegAssoc CSALegAssociationTable,

 /* CSA info. */
 existingCSA ExistingCSAType,

 /* Other variables */
 csa PId,
 csaID,newCSAID CSAID,
 invokeID InvokeID,
 i Integer,
 termination Boolean,
 csID CallSegmentID,
 legID LegType,
 halfCallSide IHRoleType,
 dialogID DialogIDtype,
 callFlag CallFlag,
 eventTable EventTableType,
 r Boolean,
 dID DialogIDtype;
```

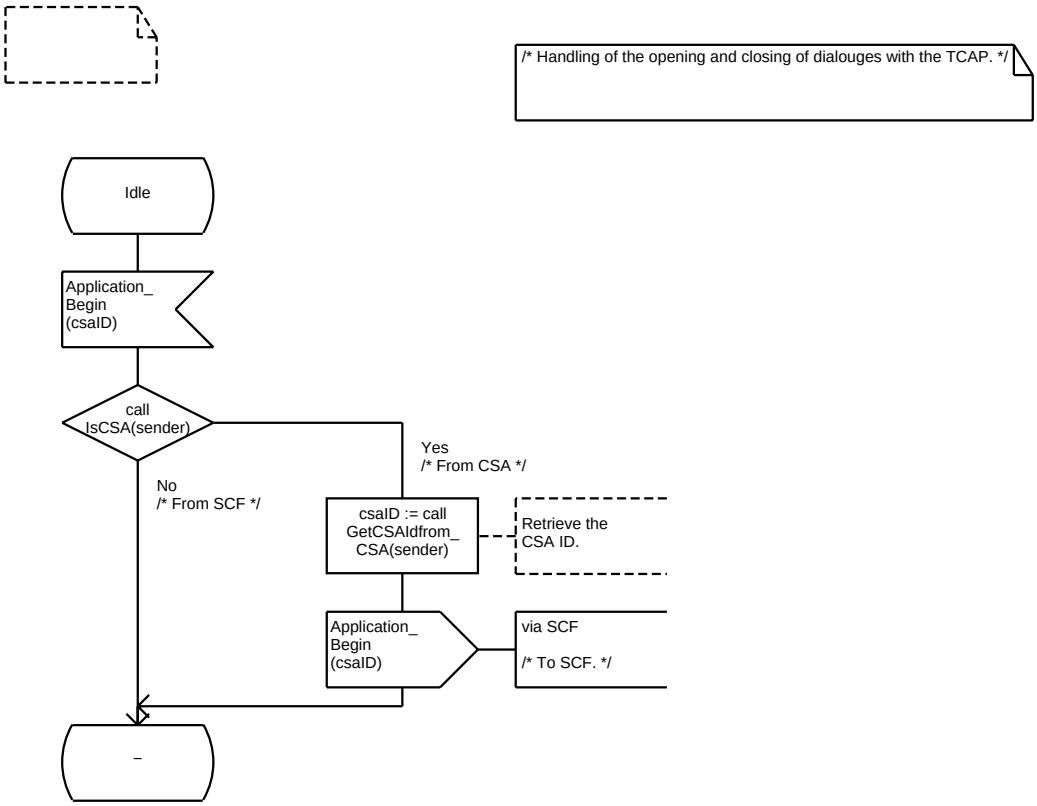
```
DCL
 /* IN CS−1 operation arguments. */
 asfArg ActivateServiceFilteringArg,
 acArg ApplyChargingArg,
 acrArg ApplyChargingReportArg,
 ariArg AssistRequestInstructionsArg,
 cgArg CallGapArg,
 cirArg CallInformationReportArg,
 cirqArg CallInformationRequestArg,
 cArg CancelArg,
 ciArg CollectInformationArg,
 coArg ConnectArg,
 ctrArg ConnectToResourceArg,
 etcArg EstablishTemporaryConnectionArg,
 encArg EventNotificationChargingArg,
 erBCSMArg EventReportBCSMArg,
 fciArg FurnishChargingInformationArg,
 idpArg InitialDPArg,
 icaArg InitiateCallAttemptArg,
 rcArg ReleaseCallArg,
 rnceARg RequestNotificationChargingEventArg,
 rrBCSMEArg RequestReportBCSMEventArg,
 rtARg ResetTimerArg,
 sciArg SendChargingInformationArg,
 sfrArg ServiceFilteringResponseArg,

 ieArg ErrorArg,

 mgt_STTArg MGT_SetTriggerTableArg;
```

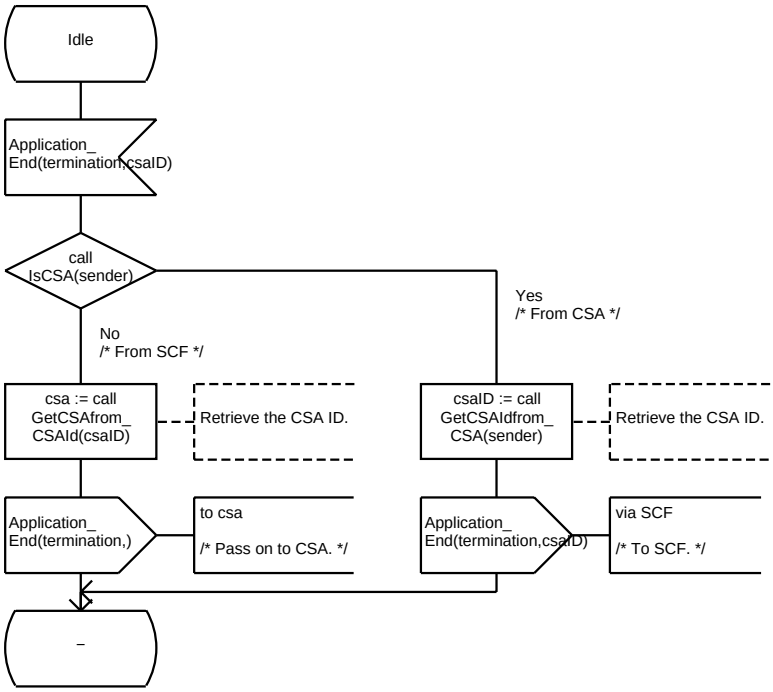
```
DCL
 /* Signalling control primitive parameters. */
 aeArg AddressEndType,
 cpArg CallProgressType,
 fArg FailureType,
 rArg ReleaseType,
 sftArg ServiceFeatureType,
 sirArg SetupIRType,
 scrArg SetupCRType,
 saArg SubsequentAddressType;
```

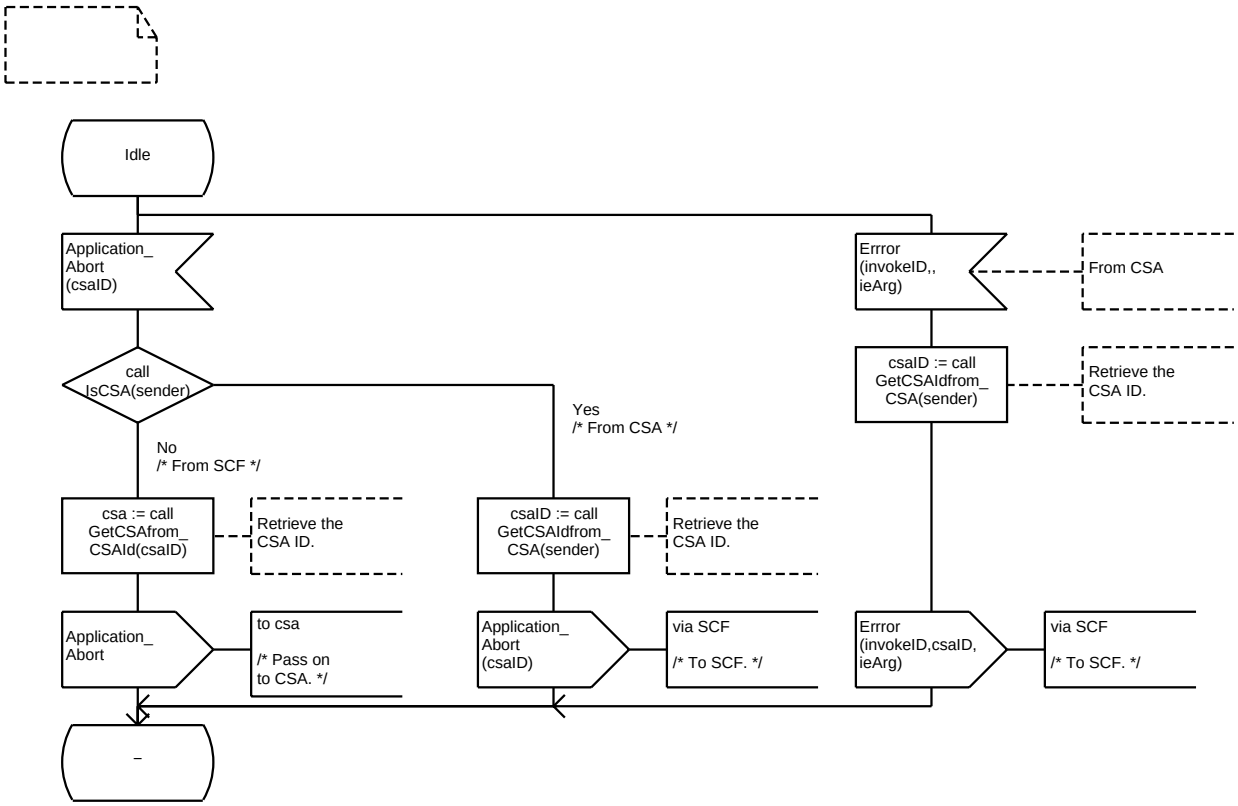
/* DECLARATION OF OPERATIONS ON THE ROUTING TABLES */

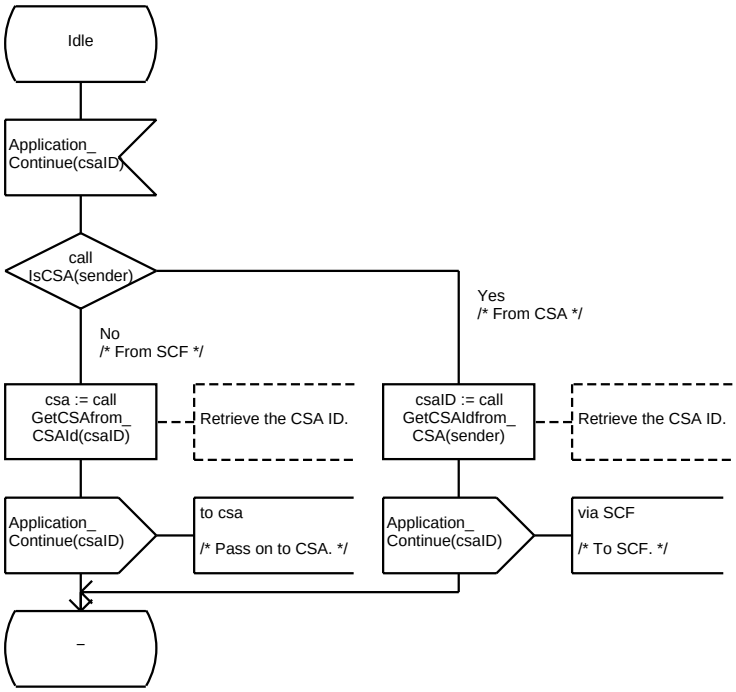


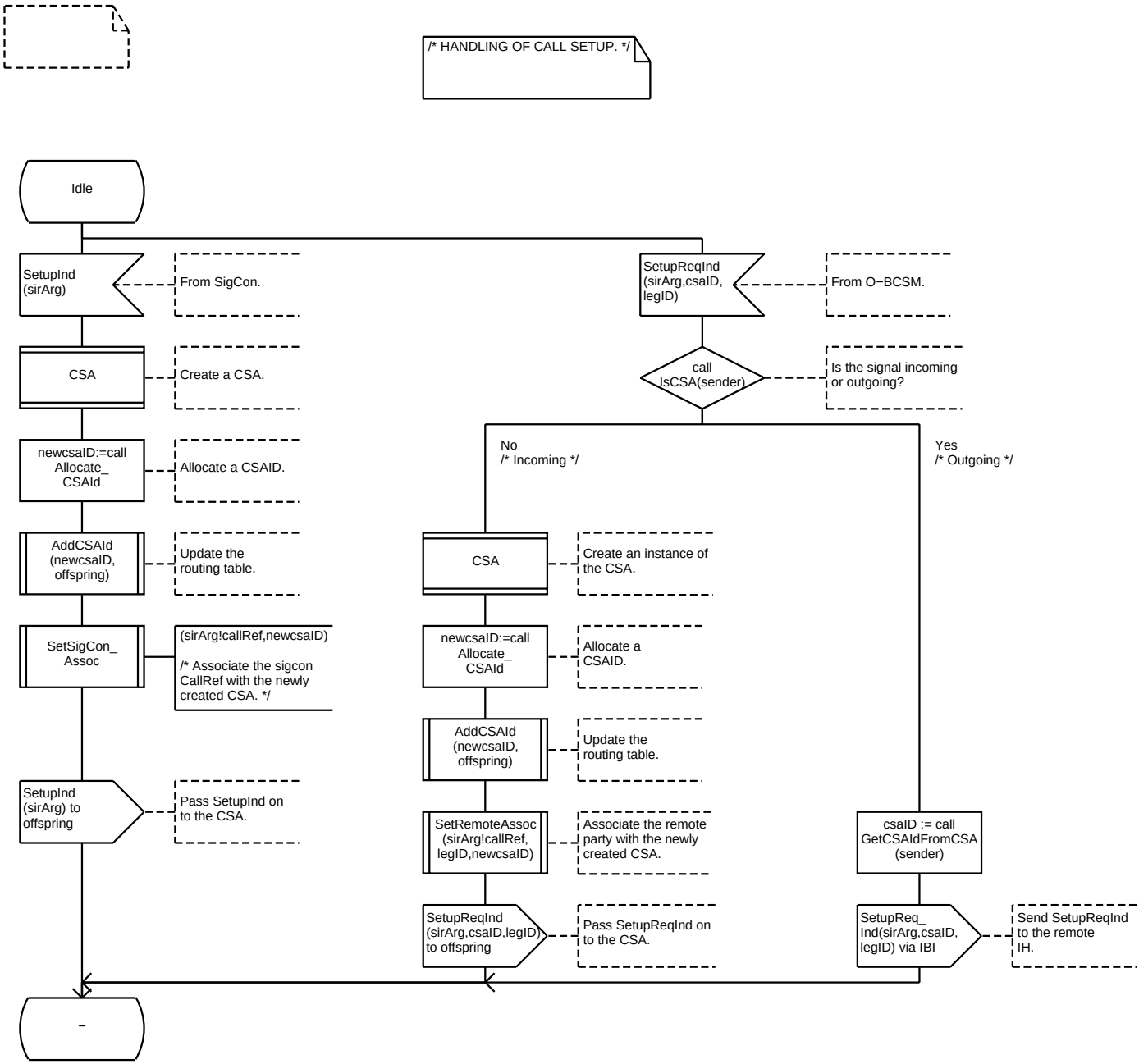

Idle
The IH starts in the Idle state.

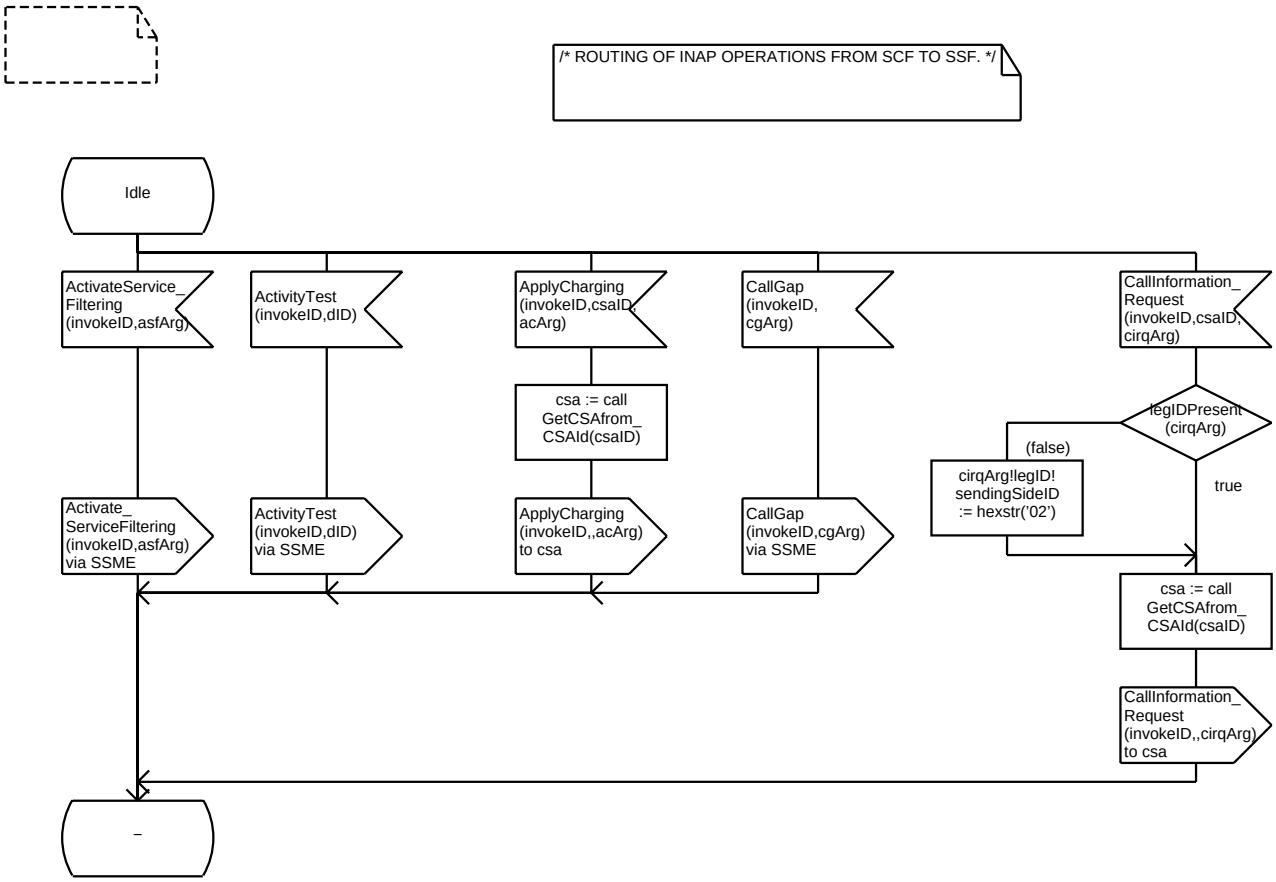

/* Handling of the opening and closing of dialouges with the TCAP. */
Idle
Application_
Begin
(csaID)
call
IsCSA(sender)
Yes
/* From CSA */
No
/* From SCF */
csaID := call
GetCSAIdfrom_
CSA(sender)
Retrieve the
CSA ID.
Application_
Begin
(csaID)
via SCF
/* To SCF. */
_

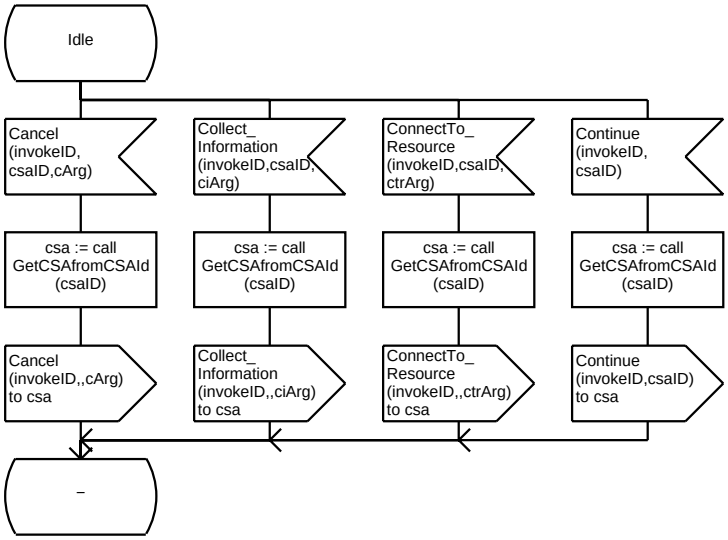

Idle
Application_
End(termination,csaID)
call
IsCSA(sender)
Yes
/* From CSA */
No
/* From SCF */
csa := call
GetCSAfrom_
CSAId(csaID)
Retrieve the CSA ID.
csaID := call
GetCSAIdfrom_
CSA(sender)
Retrieve the CSA ID.
Application_
End(termination,)
to csa
/* Pass on to CSA. */
Application_
End(termination,csaID)
via SCF
/* To SCF. */
_

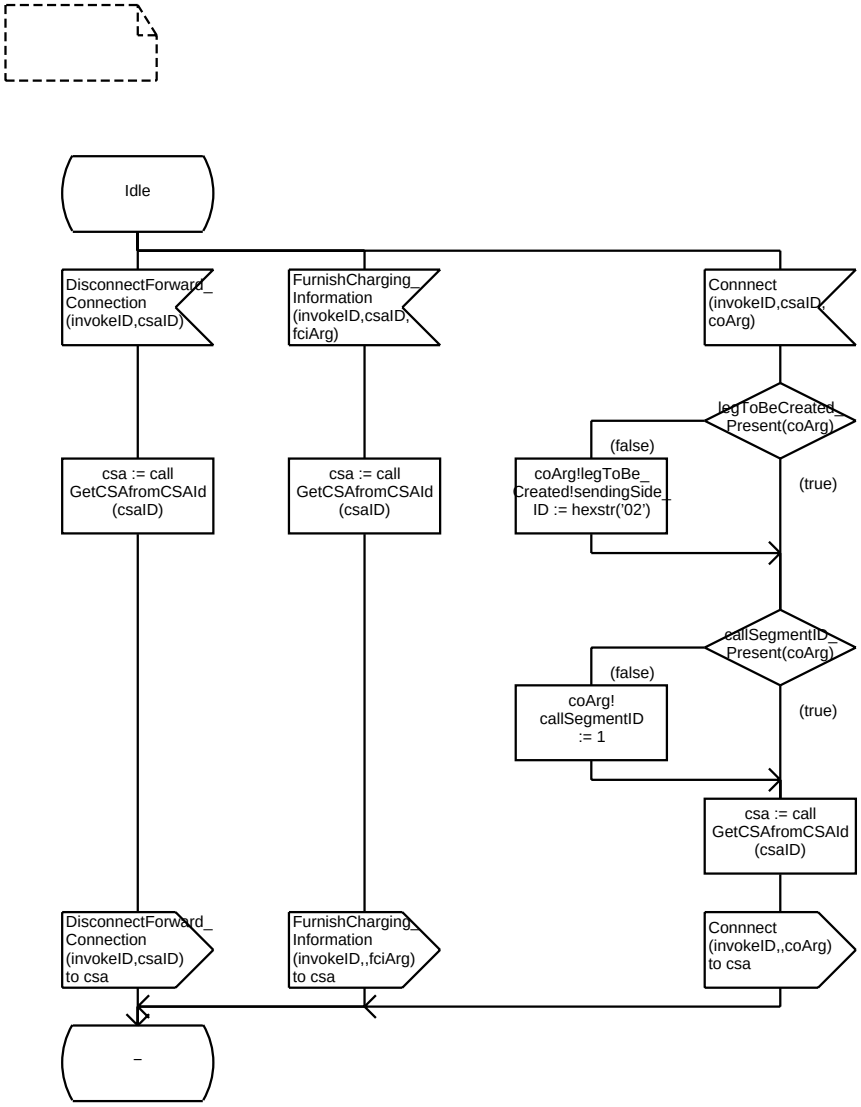

Idle
Application_
Abort
(csaID)
call
IsCSA(sender)
Yes
/* From CSA */
No
/* From SCF */
csa := call
GetCSAfrom_
CSAId(csaID)
Retrieve the
CSA ID.
csaID := call
GetCSAIdfrom_
CSA(sender)
Retrieve the
CSA ID.
Application_
Abort
to csa
/* Pass on
to CSA. */
Application_
Abort
(csaID)
via SCF
/* To SCF. */
Errror
(invokeID,,
ieArg)
From CSA
csaID := call
GetCSAIdfrom_
CSA(sender)
Retrieve the
CSA ID.
Errror
(invokeID,csaID,
ieArg)
via SCF
/* To SCF. */
–

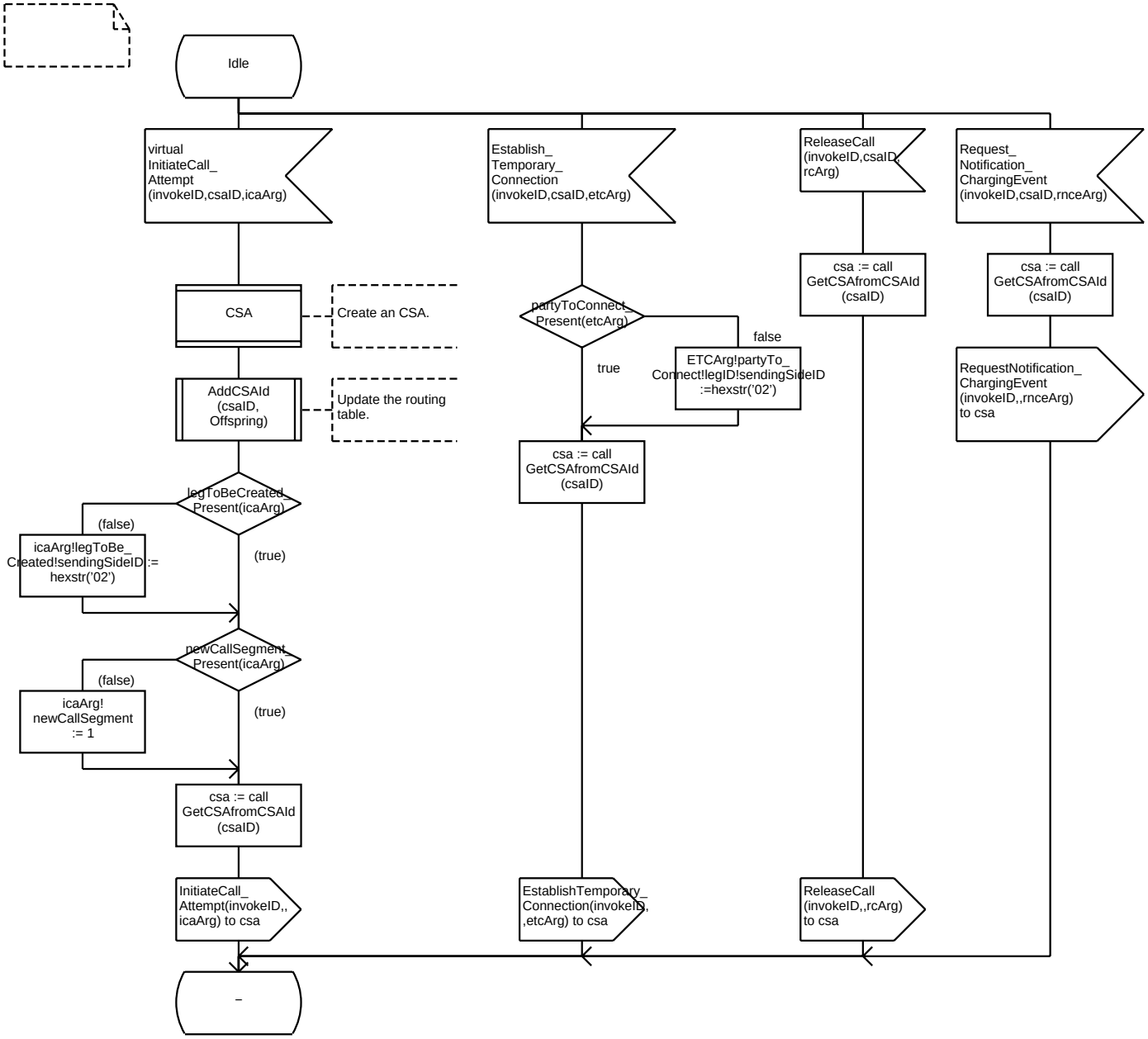

Idle
Application_
Continue(csaID)
call
IsCSA(sender)
Yes
/* From CSA */
No
/* From SCF */
csa := call
GetCSAfrom_
CSAId(csaID)
Retrieve the CSA ID.
csaID := call
GetCSAIdfrom_
CSA(sender)
Retrieve the CSA ID.
Application_
Continue(csaID)
to csa
/* Pass on to CSA. */
Application_
Continue(csaID)
via SCF
/* To SCF. */
–

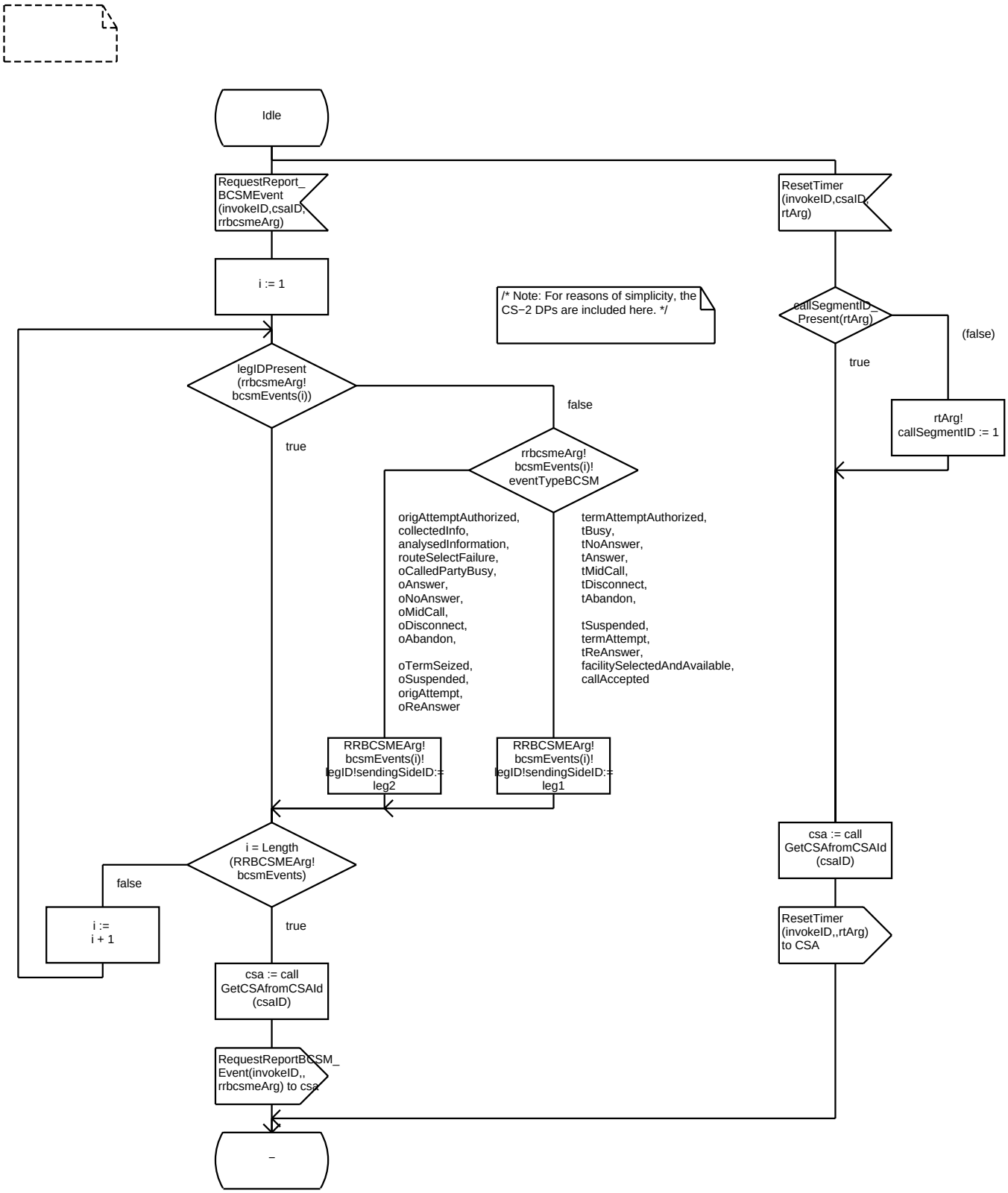

/* HANDLING OF CALL SETUP. */
Idle
SetupInd (sirArg)
From SigCon.
CSA
Create a CSA.
newcsaID:=call Allocate_ CSAId
Allocate a CSAID.
AddCSAId (newcsaID, offspring)
Update the routing table.
SetSigCon_ Assoc
(sirArg!callRef,newcsaID)
/* Associate the sigcon CallRef with the newly created CSA. */
SetupInd (sirArg) to offspring
Pass SetupInd on to the CSA.
SetupReqInd (sirArg,csaID, legID)
From O−BCSM.
call IsCSA(sender)
Is the signal incoming or outgoing?
No
/* Incoming */
Yes
/* Outgoing */
CSA
Create an instance of the CSA.
newcsaID:=call Allocate_ CSAId
Allocate a CSAID.
AddCSAId (newcsaID, offspring)
Update the routing table.
SetRemoteAssoc (sirArg!callRef, legID,newcsaID)
Associate the remote party with the newly created CSA.
SetupReqInd (sirArg,csaID,legID) to offspring
Pass SetupReqInd on to the CSA.
csaID := call GetCSAIdFromCSA (sender)
SetupReq_ Ind(sirArg,csaID, legID) via IBI
Send SetupReqInd to the remote IH.
–

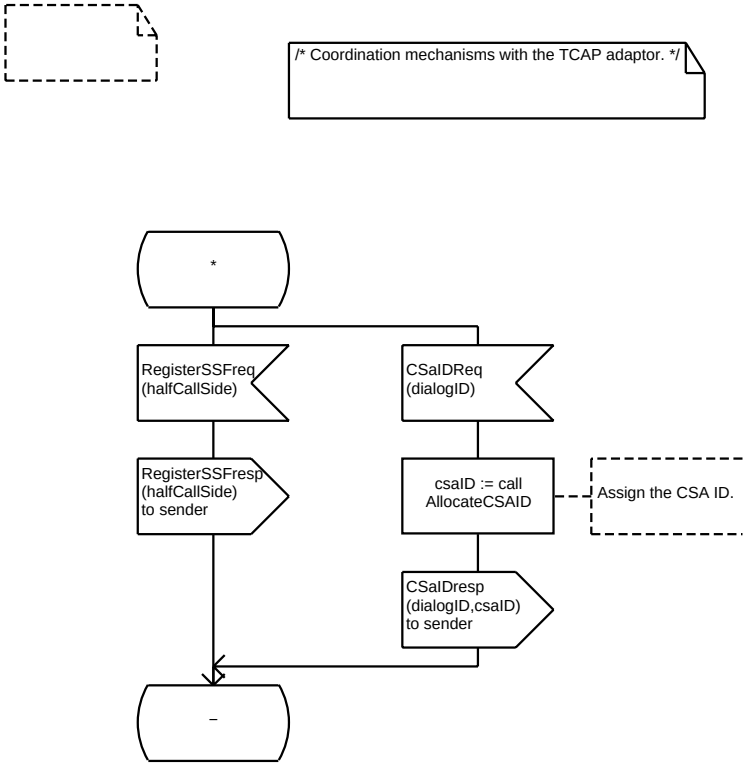

/* ROUTING OF INAP OPERATIONS FROM SCF TO SSF. */
Idle
ActivateService_ Filtering (invokeID,asfArg)
ActivityTest (invokeID,dID)
ApplyCharging (invokeID,csaID, acArg)
CallGap (invokeID, cgArg)
CallInformation_ Request (invokeID,csaID, cirqArg)
csa := call GetCSAfrom_ CSAId(csaID)
legIDPresent (cirqArg)
(false)
true
cirqArg!legID! sendingSideID := hexstr('02')
Activate_ ServiceFiltering (invokeID,asfArg) via SSME
ActivityTest (invokeID,dID) via SSME
ApplyCharging (invokeID,,acArg) to csa
CallGap (invokeID,cgArg) via SSME
csa := call GetCSAfrom_ CSAId(csaID)
CallInformation_ Request (invokeID,,cirqArg) to csa
–


Idle
Cancel (invokeID, csaID,cArg)
Collect_ Information (invokeID,csaID, ciArg)
ConnectTo_ Resource (invokeID,csaID, ctrArg)
Continue (invokeID, csaID)
csa := call GetCSAfromCSAId (csaID)
csa := call GetCSAfromCSAId (csaID)
csa := call GetCSAfromCSAId (csaID)
csa := call GetCSAfromCSAId (csaID)
Cancel (invokeID,,cArg) to csa
Collect_ Information (invokeID,,ciArg) to csa
ConnectTo_ Resource (invokeID,,ctrArg) to csa
Continue (invokeID,csaID) to csa
–


Idle
DisconnectForward_
Connection
(invokeID,csaID)
FurnishCharging_
Information
(invokeID,csaID,
fciArg)
Connnect
(invokeID,csaID,
coArg)
legToBeCreated_
Present(coArg)
(false)
(true)
csa := call
GetCSAfromCSAId
(csaID)
csa := call
GetCSAfromCSAId
(csaID)
coArg!legToBe_
Created!sendingSide_
ID := hexstr('02')
callSegmentID_
Present(coArg)
(false)
(true)
coArg!
callSegmentID
:= 1
csa := call
GetCSAfromCSAId
(csaID)
DisconnectForward_
Connection
(invokeID,csaID)
to csa
FurnishCharging_
Information
(invokeID,,fciArg)
to csa
Connnect
(invokeID,,coArg)
to csa
_

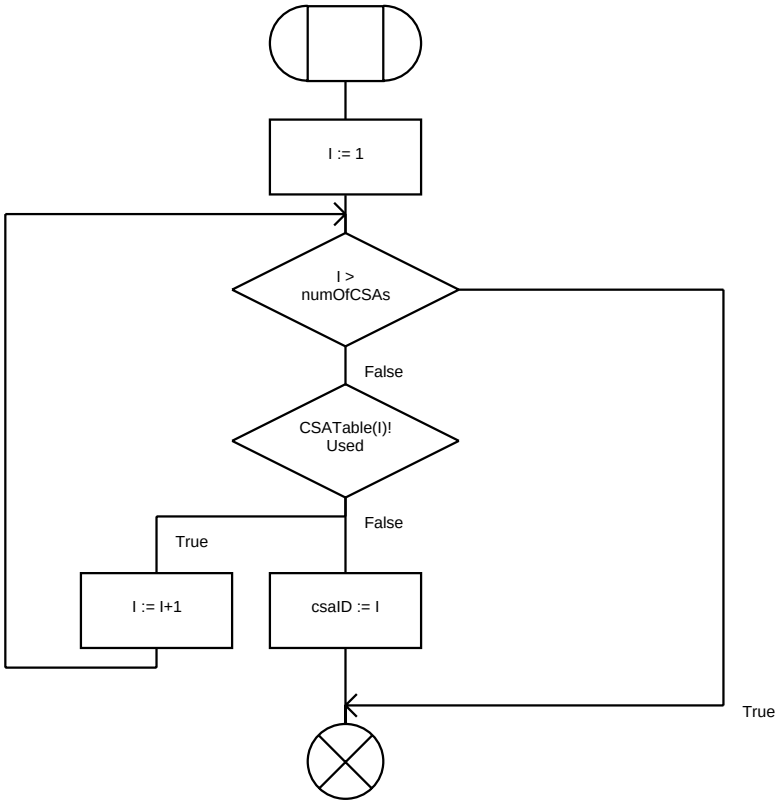

Idle
virtual InitiateCall_ Attempt (invokeID,csaID,icaArg)
Establish_ Temporary_ Connection (invokeID,csaID,etcArg)
ReleaseCall (invokeID,csaID, rcArg)
Request_ Notification_ ChargingEvent (invokeID,csaID,rnceArg)
CSA
Create an CSA.
AddCSAId (csaID, Offspring)
Update the routing table.
legToBeCreated_ Present(icaArg)
(false)
(true)
icaArg!legToBe_ Created!sendingSideID := hexstr('02')
newCallSegment_ Present(icaArg)
(false)
(true)
icaArg! newCallSegment := 1
csa := call GetCSAfromCSAId (csaID)
InitiateCall_ Attempt(invokeID,, icaArg) to csa
partyToConnect_ Present(etcArg)
true
false
ETCArg!partyTo_ Connect!legID!sendingSideID :=hexstr('02')
csa := call GetCSAfromCSAId (csaID)
EstablishTemporary_ Connection(invokeID, ,etcArg) to csa
csa := call GetCSAfromCSAId (csaID)
ReleaseCall (invokeID,,rcArg) to csa
csa := call GetCSAfromCSAId (csaID)
RequestNotification_ ChargingEvent (invokeID,,rnceArg) to csa
–


Idle
RequestReport_
BCSMEvent
(invokeID,csaID,
rrbcsmeArg)
i := 1
/* Note: For reasons of simplicity, the
CS−2 DPs are included here. */
legIDPresent
(rrbcsmeArg!
bcsmEvents(i))
false
true
rrbcsmeArg!
bcsmEvents(i)!
eventTypeBCSM
origAttemptAuthorized,
collectedInfo,
analysedInformation,
routeSelectFailure,
oCalledPartyBusy,
oAnswer,
oNoAnswer,
oMidCall,
oDisconnect,
oAbandon,
oTermSeized,
oSuspended,
origAttempt,
oReAnswer
termAttemptAuthorized,
tBusy,
tNoAnswer,
tAnswer,
tMidCall,
tDisconnect,
tAbandon,
tSuspended,
termAttempt,
tReAnswer,
facilitySelectedAndAvailable,
callAccepted
RRBCSMEArg!
bcsmEvents(i)!
legID!sendingSideID:=
leg2
RRBCSMEArg!
bcsmEvents(i)!
legID!sendingSideID:=
leg1
i = Length
(RRBCSMEArg!
bcsmEvents)
false
i :=
i + 1
true
csa := call
GetCSAfromCSAId
(csaID)
RequestReportBCSM_
Event(invokeID,,
rrbcsmeArg) to csa
–
ResetTimer
(invokeID,csaID,
rtArg)
callSegmentID_
Present(rtArg)
(false)
true
rtArg!
callSegmentID := 1
csa := call
GetCSAfromCSAId
(csaID)
ResetTimer
(invokeID,,rtArg)
to CSA


/* Coordination mechanisms with the TCAP adaptor. */
*
RegisterSSFreq
(halfCallSide)
CSaIDReq
(dialogID)
RegisterSSFresp
(halfCallSide)
to sender
csaID := call
AllocateCSAID
Assign the CSA ID.
CSaIDresp
(dialogID,csaID)
to sender
–

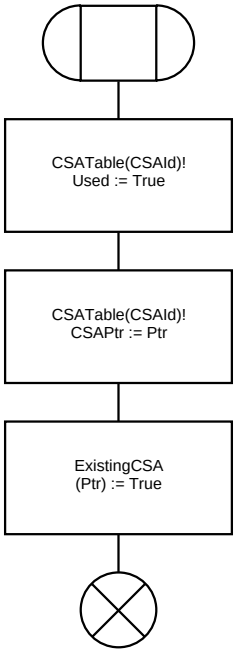
;RETURNS csaID CSAID;

DCL
I Integer;

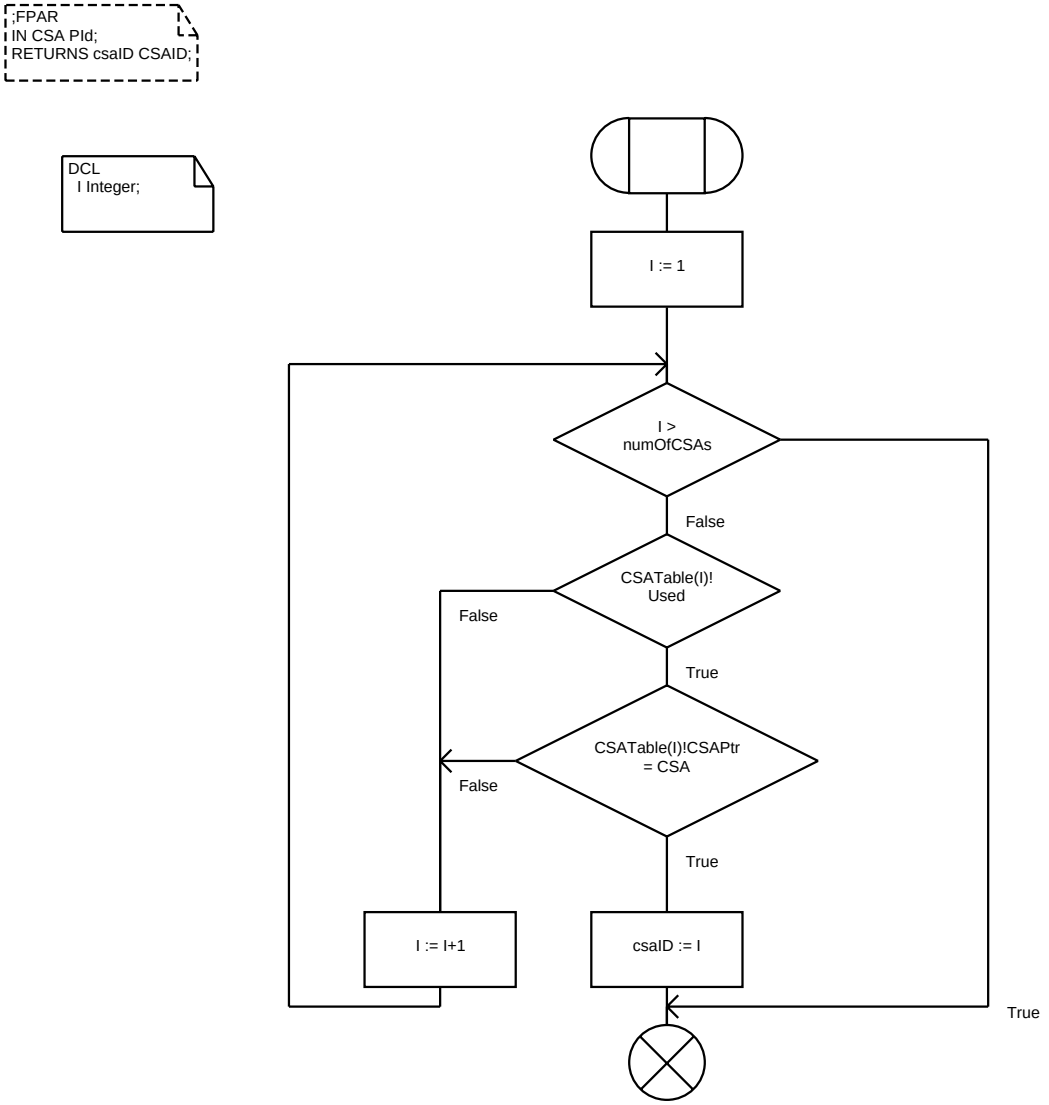


Procedure AddCSAId

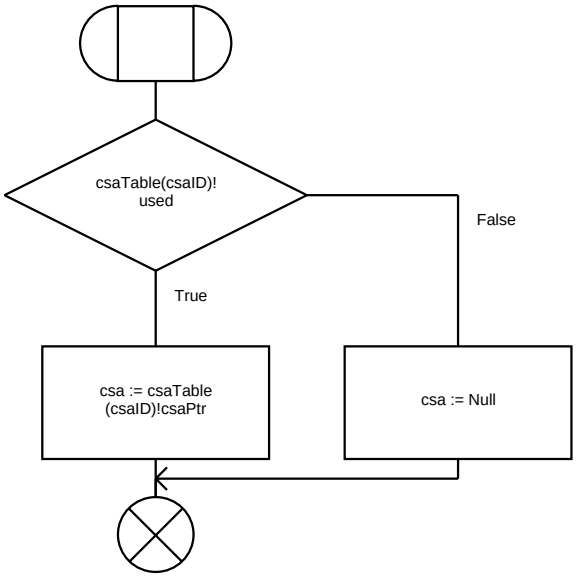

```
;FPAR
IN csaID CSAID, /* The CSA id to be assigned. */
IN Ptr PId; /* The CSA pointer. */
```



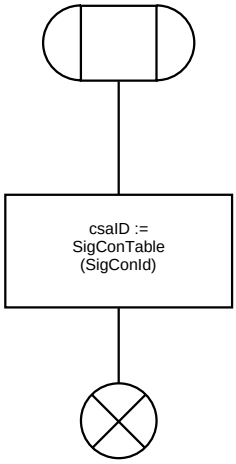
Procedure GetCSAIdfromCSA



```
;FPAR
IN csaID CSAID;
RETURNS csa PId;
```

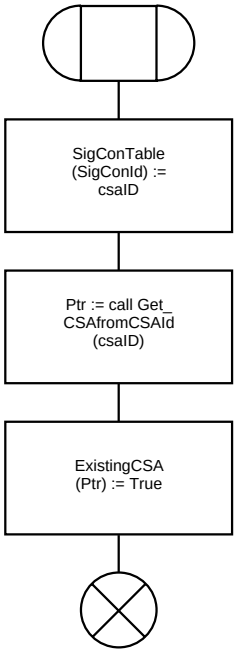


;
FPAR
IN SigConId CallRef;
RETURNS csaID CSAID;



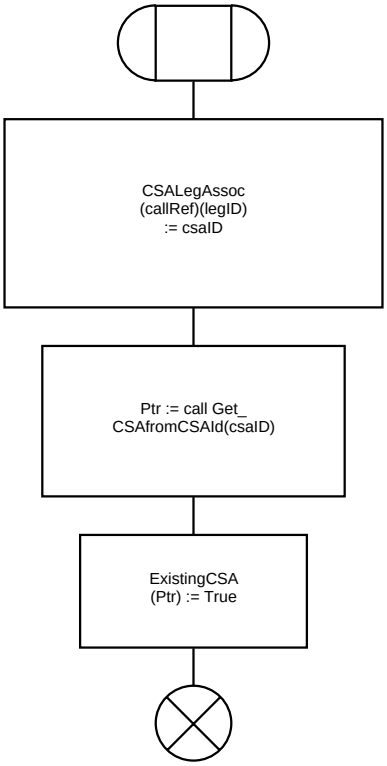
```
;FPAR
  IN SigConId CallRef,
  IN csaID CSAID;
```

```
DCL
  Ptr PId;
```

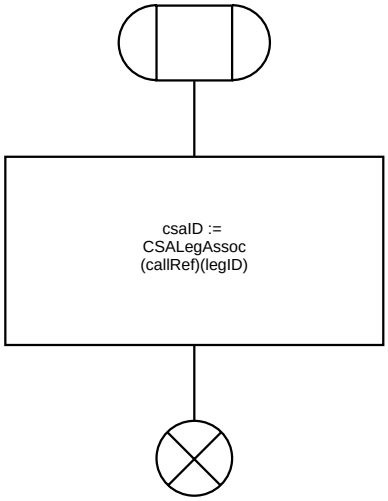


```
;FPAR
  IN callRef CallRef,
  IN legID LegType,
  IN csaID CSAID;
```

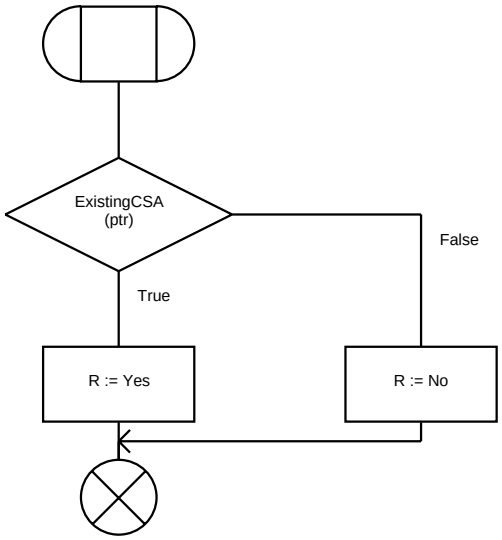
```
DCL
  Ptr PId;
```



```
;FPAR
  IN callRef CallRef,
  IN legID LegType;
RETURNS csaID CSAID;
```



```
;FPAR
  IN Ptr PId;
RETURNS R AnswerType;
```



/**** DATA TYPE DEFINITIONS ****/

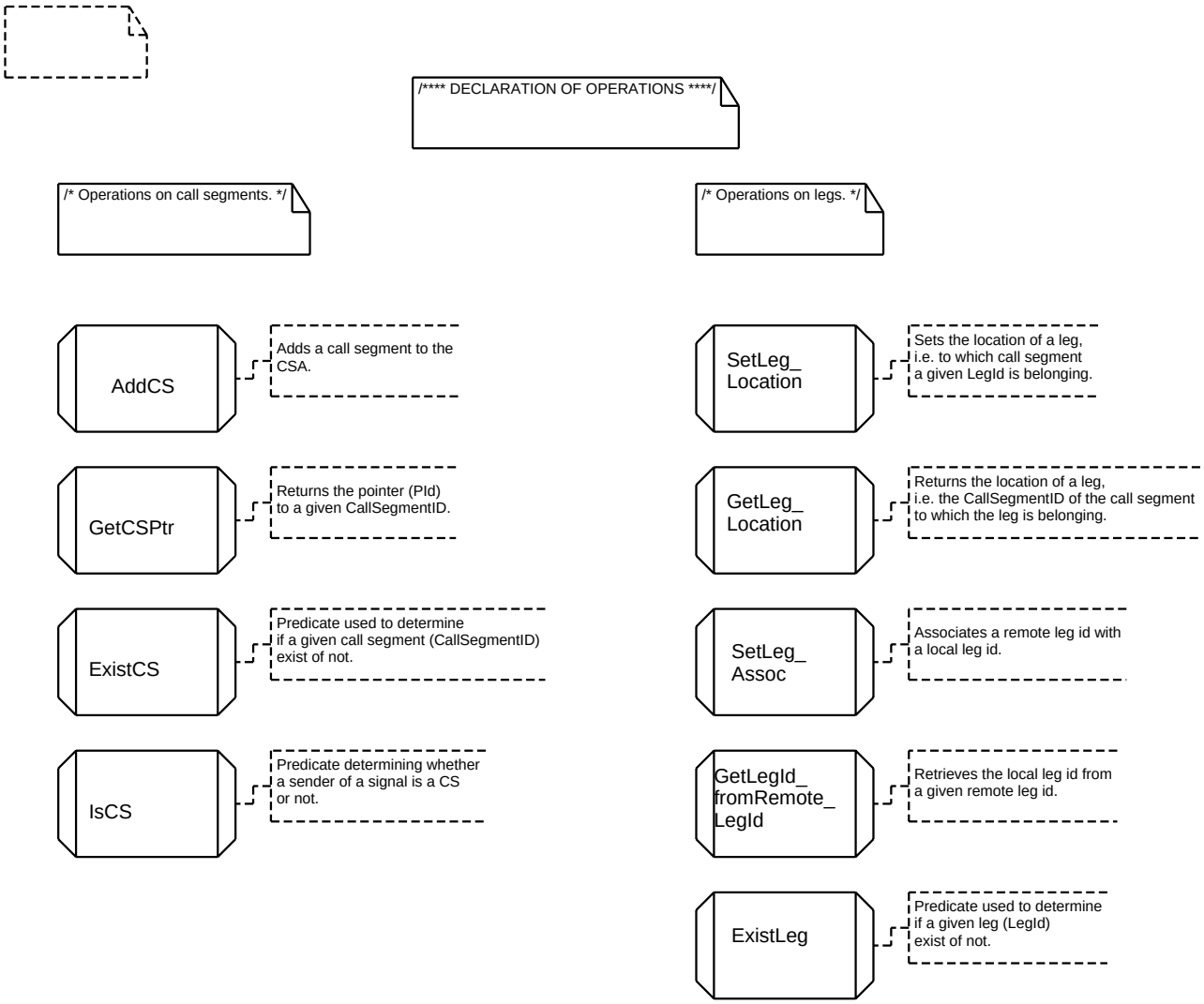
```
/* The Call Segment Association (CSA) maintains information
about the Call Segments (CS) of the CSA. */

NEWTYPE CS STRUCT
  Used Boolean; /* Flag indicating whether the segment exists or not. */
  CSPtr PId;    /* Pointer/Address of the instance of the CS process associated
                   with the segment. */
ENDNEWTYPE;

NEWTYPE CSAType /* The call segment association. */
  ARRAY(CallSegmentID,CS)
ENDNEWTYPE;


/* The CSA maintains a mapping from LegID to CallSegmentID. This information
is needed to route primitives to the correct call segment. */

NEWTYPE LegLocationTable
  ARRAY(LegType,CallSegmentID)
ENDNEWTYPE;


/* The CSA maintains a leg association table, which
associates a 'remote' LegID with a 'local' leg id. This
information is needed to correctly address the BCSM when
passing primitives between an O−BCSM and a T−BCSM. */

NEWTYPE LegAssociationTable
  ARRAY(LegType,LegType)
ENDNEWTYPE;


/* Definition of return results from procedure calls. */

NEWTYPE AnswerType
  LITERALS Yes, No;
ENDNEWTYPE;

NEWTYPE ResultType
  LITERALS Successful, Failed;
ENDNEWTYPE;
```

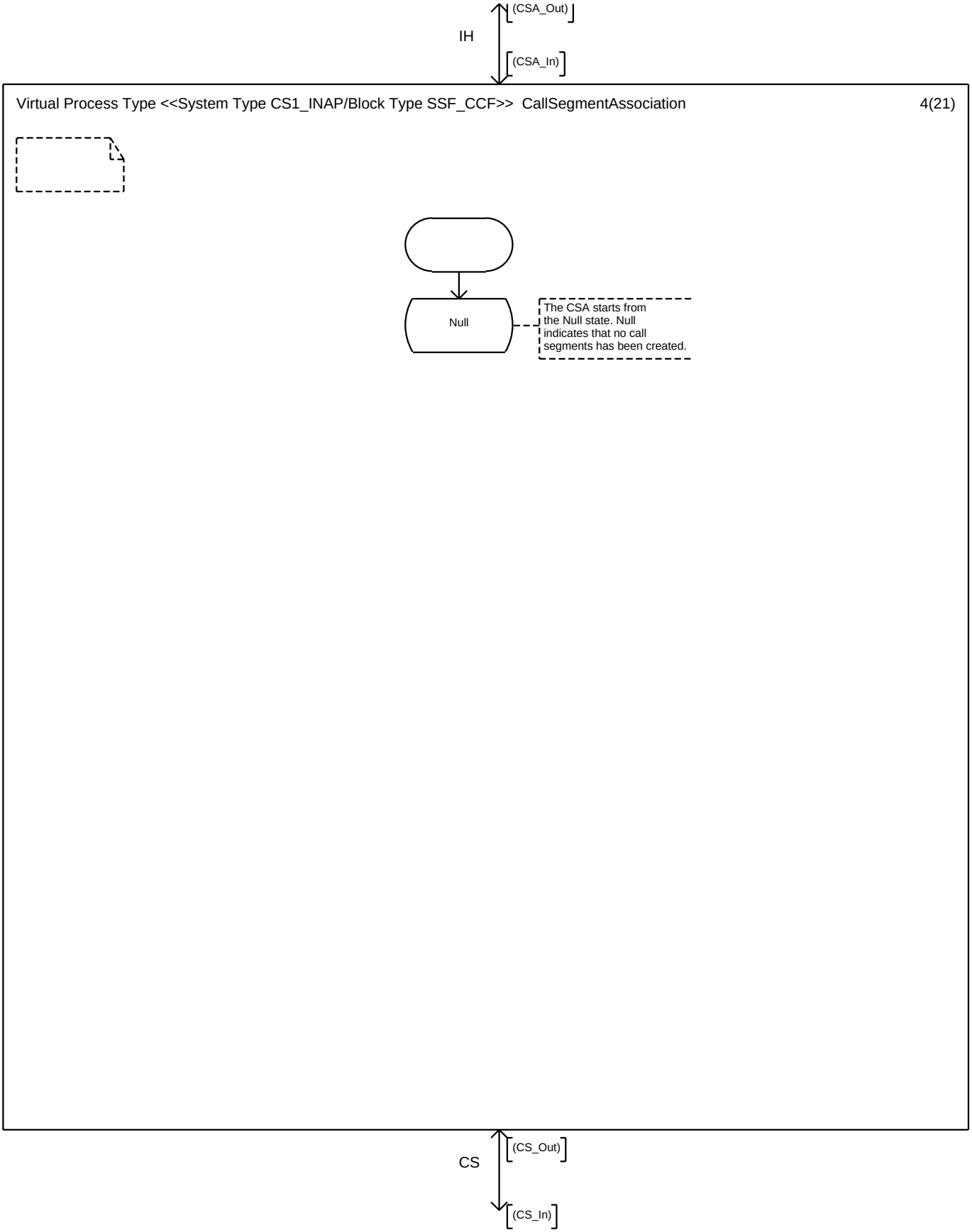
/**** VARIABLE DECLARATIONS ****/

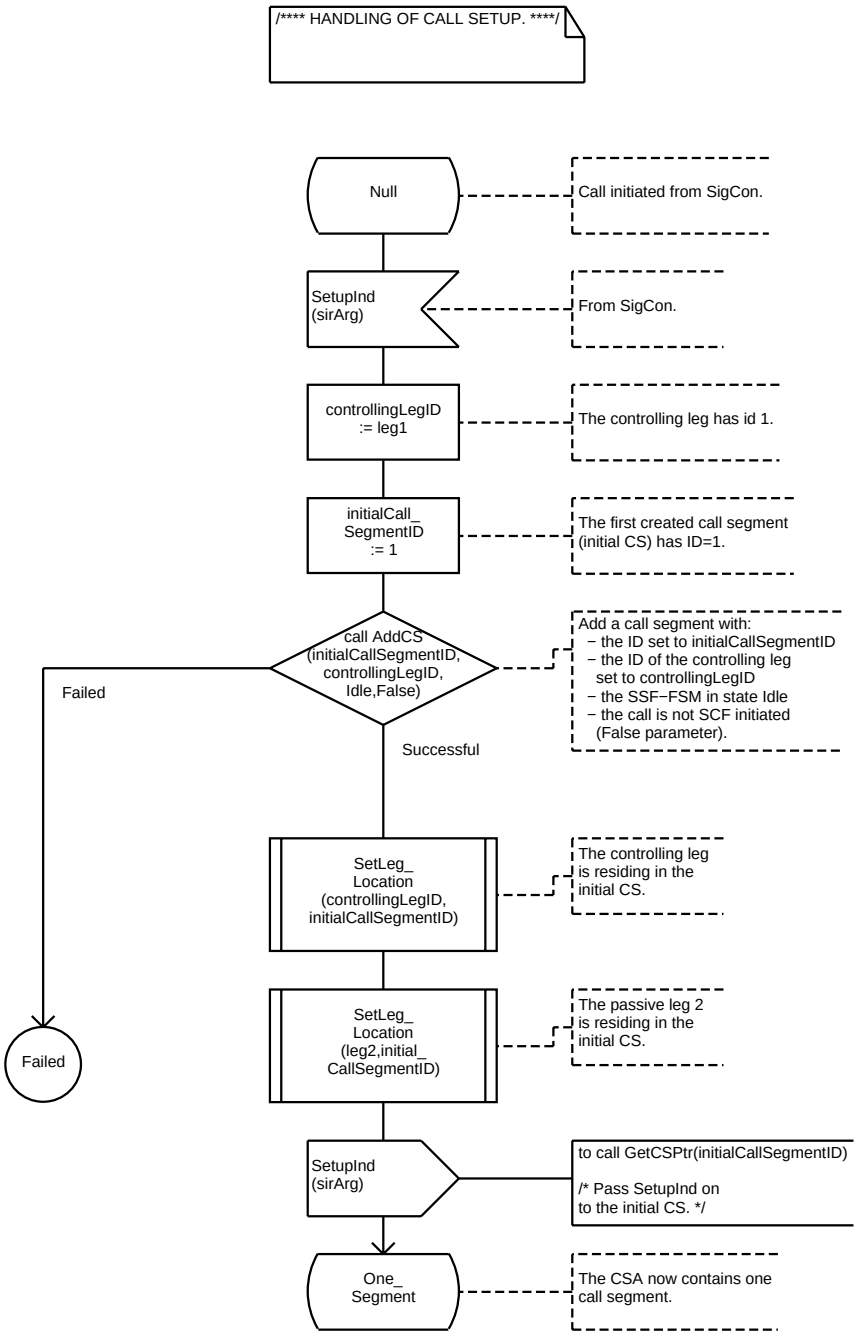
```
DCL
 /* The CSA */
 CSA CSAType,

 /* The leg location table. */
 LL LegLocationTable,

/* ID of the initial (first created) call segment. */
 initialCallSegmentID CallSegmentID,

 /* The association of local and remote leg ids. */
 LegAssoc LegAssociationTable,

 /* The ID of the controlling Leg (1 or 2) */
 controllingLegID LegType,

 numOfAppls Integer,        /* Variable that holds the current number of open
                              applications/dialogues. */

 /* Other variables. */
 csaID CSAID,
 csID CallSegmentID,
 invokeID InvokeID,
 termination Boolean,
 legID, newLegID LegType,
 callFlag CallFlag,
 eventTable EventTableType,
 r Boolean;
```

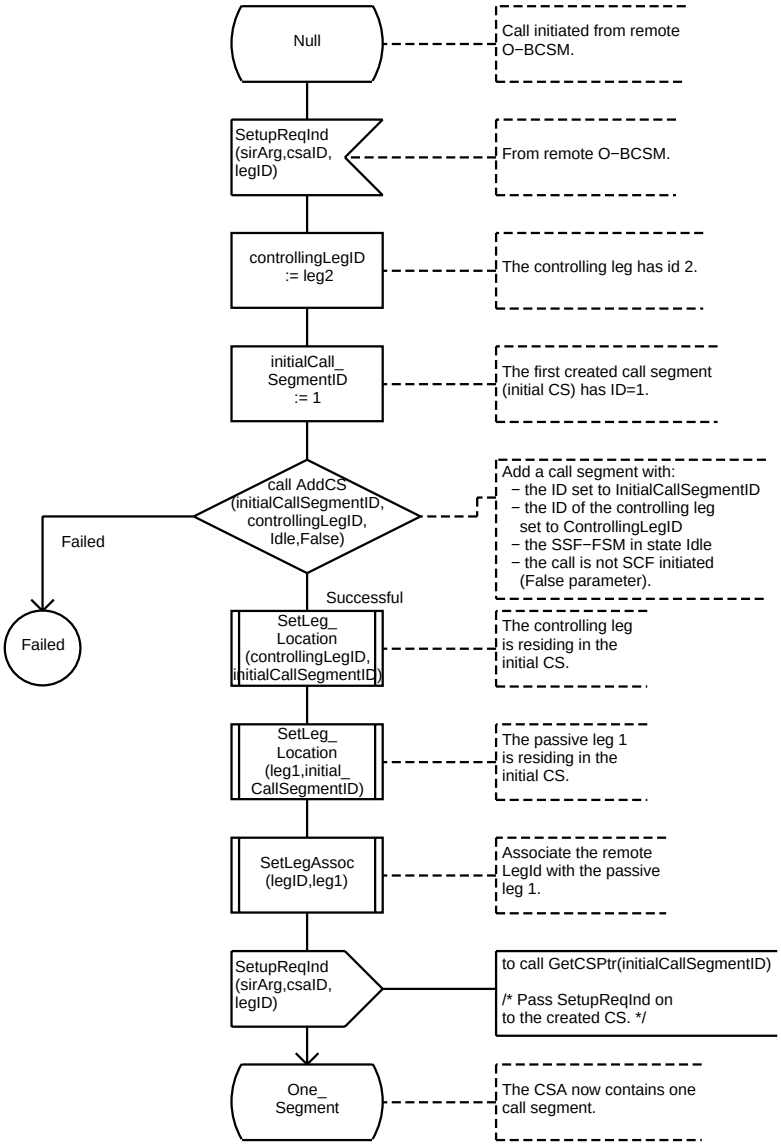
```
DCL
 /* IN CS−1 operation arguments. */
 acArg ApplyChargingArg,
 acrArg ApplyChargingReportArg,
 ariArg AssistRequestInstructionsArg,
 aiArg AnalyseInformationArg,
 cirArg CallInformationReportArg,
 cirqArg CallInformationRequestArg,
 cArg CancelArg,
 ciArg CollectInformationArg,
 coArg ConnectArg,
 ctrArg ConnectToResourceArg,
 ctArg ContinueWithArgumentArg,
 dfcArg DisconnectForwardConnectionWithArgumentArg,
 etcArg EstablishTemporaryConnectionArg,
 encArg EventNotificationChargingArg,
 erbcsmArg EventReportBCSMArg,
 fciArg FurnishChargingInformationArg,
 idpArg InitialDPArg,
 icaArg InitiateCallAttemptArg,
 rcArg ReleaseCallArg,
 rnceARg RequestNotificationChargingEventArg,
 rrbcsmArg RequestReportBCSMEventArg,
 rtARg ResetTimerArg,
 sciArg SendChargingInformationArg,
 srArg SelectRouteArg,
 ieArg ErrorArg;
```

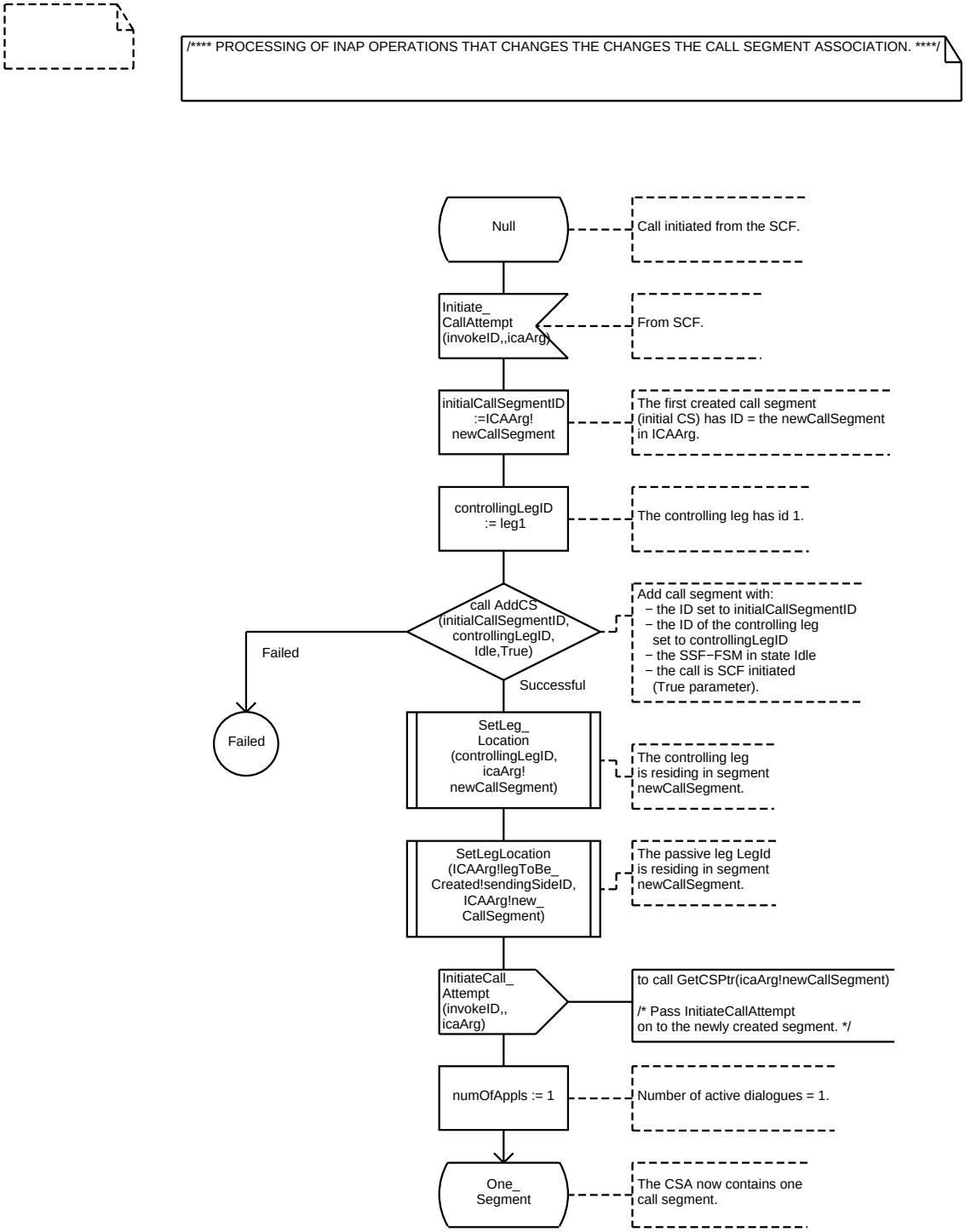
```
DCL
 /* Signalling control primitive parameters. */
 AEArg AddressEndType,
 CPArg CallProgressType,
 FArg FailureType,
 RArg ReleaseType,
 SFtArg ServiceFeatureType,
 SIRArg SetupIRType,
 SCRArg SetupCRType,
 SAArg SubsequentAddressType;
```

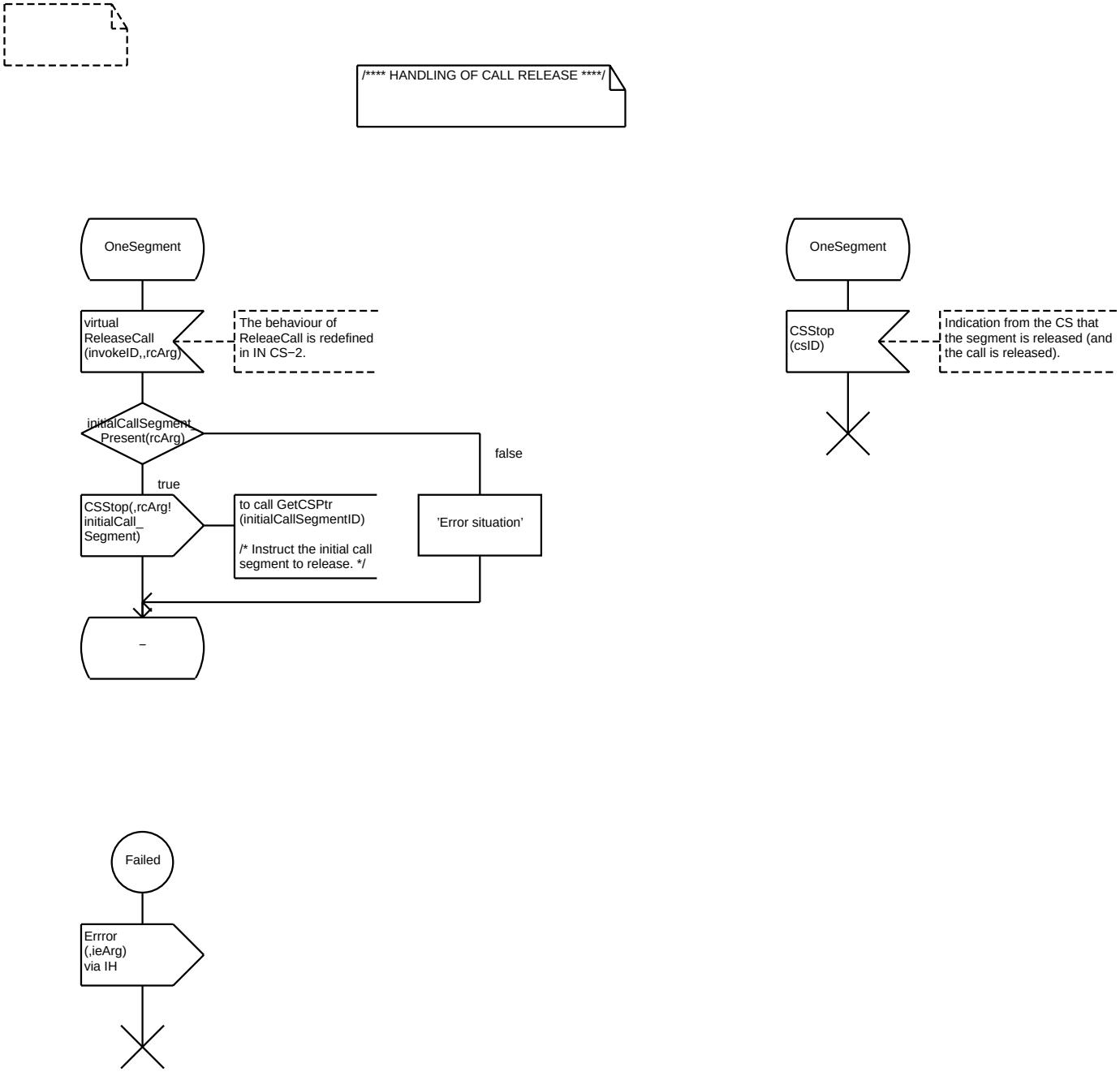

/**** DECLARATION OF OPERATIONS ****/
/* Operations on call segments. */
/* Operations on legs. */
AddCS
Adds a call segment to the CSA.
GetCSPtr
Returns the pointer (PId) to a given CallSegmentID.
ExistCS
Predicate used to determine if a given call segment (CallSegmentID) exist of not.
IsCS
Predicate determining whether a sender of a signal is a CS or not.
SetLeg_ Location
Sets the location of a leg, i.e. to which call segment a given LegId is belonging.
GetLeg_ Location
Returns the location of a leg, i.e. the CallSegmentID of the call segment to which the leg is belonging.
SetLeg_ Assoc
Associates a remote leg id with a local leg id.
GetLegId_ fromRemote_ LegId
Retrieves the local leg id from a given remote leg id.
ExistLeg
Predicate used to determine if a given leg (LegId) exist of not.

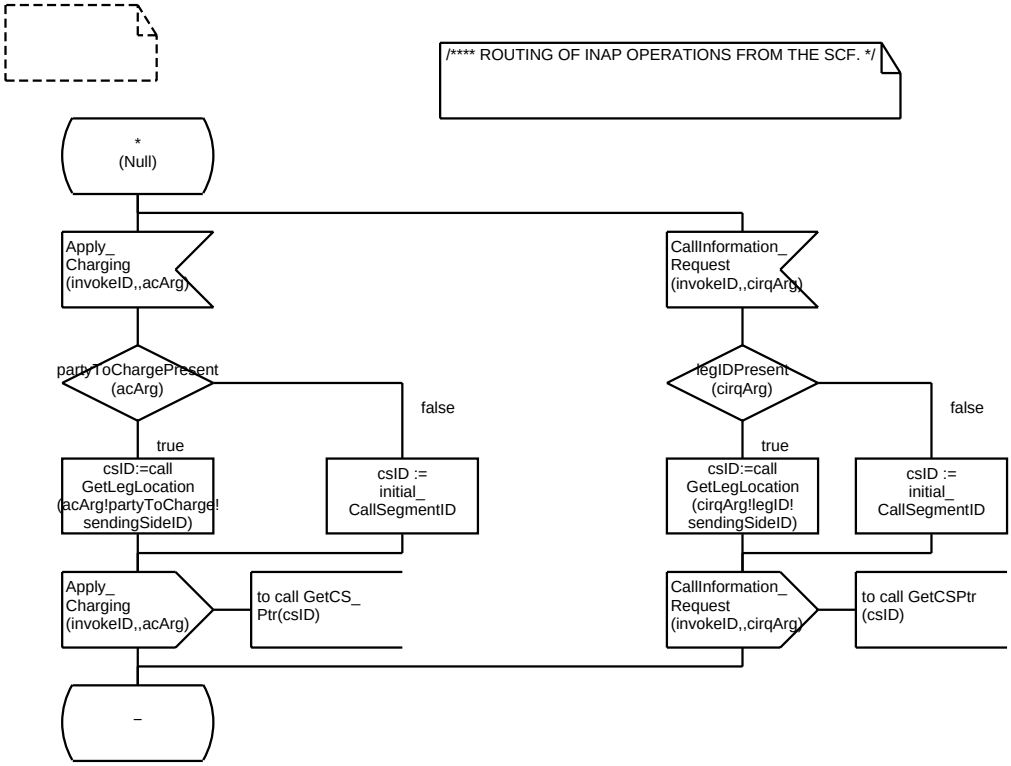

IH
(CSA_Out)
(CSA_In)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAssociation
4(21)
Null
The CSA starts from the Null state. Null indicates that no call segments has been created.
CS
(CS_Out)
(CS_In)

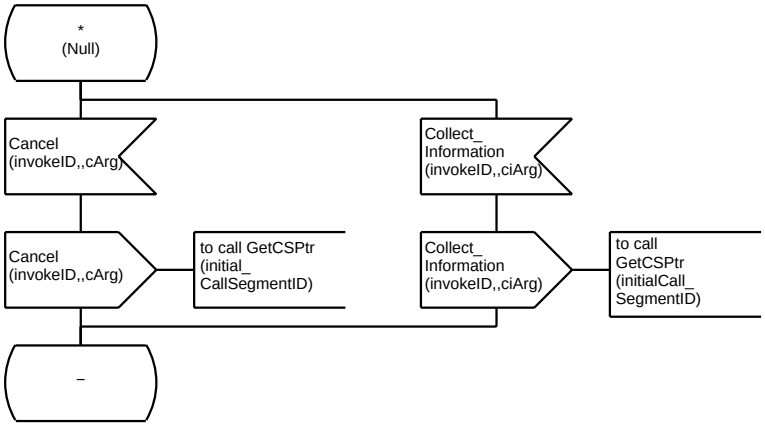

/**** HANDLING OF CALL SETUP. ****/
Null
Call initiated from SigCon.
SetupInd (sirArg)
From SigCon.
controllingLegID := leg1
The controlling leg has id 1.
initialCall_ SegmentID := 1
The first created call segment (initial CS) has ID=1.
call AddCS (initialCallSegmentID, controllingLegID, Idle,False)
Add a call segment with:
− the ID set to initialCallSegmentID
− the ID of the controlling leg set to controllingLegID
− the SSF−FSM in state Idle
− the call is not SCF initiated (False parameter).
Failed
Successful
SetLeg_ Location (controllingLegID, initialCallSegmentID)
The controlling leg is residing in the initial CS.
SetLeg_ Location (leg2,initial_ CallSegmentID)
The passive leg 2 is residing in the initial CS.
Failed
SetupInd (sirArg)
to call GetCSPtr(initialCallSegmentID)
/* Pass SetupInd on to the initial CS. */
One_ Segment
The CSA now contains one call segment.

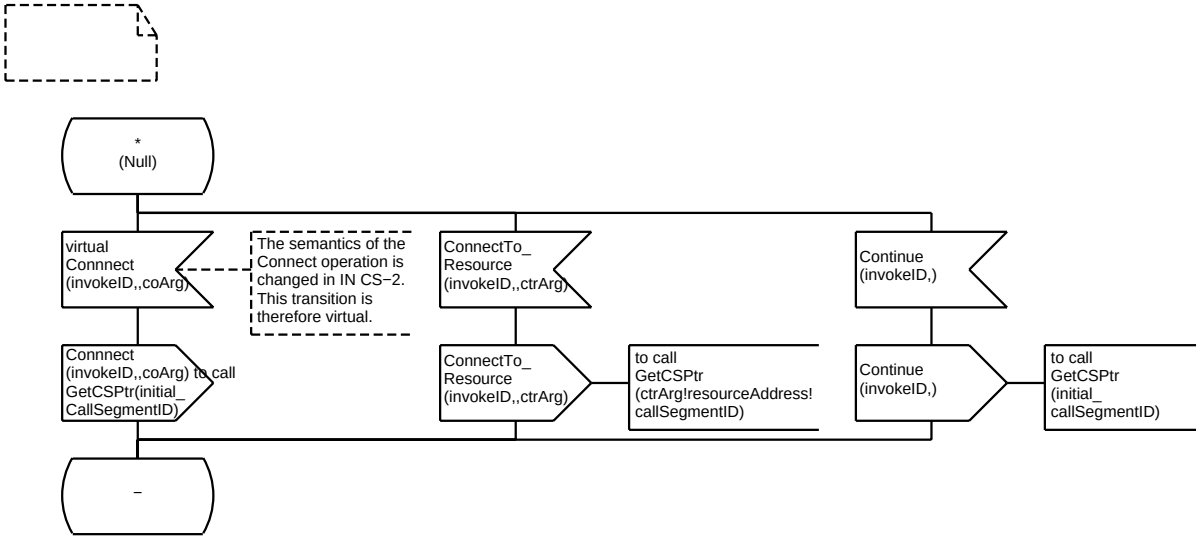

Null
Call initiated from remote O−BCSM.
SetupReqInd (sirArg,csaID, legID)
From remote O−BCSM.
controllingLegID := leg2
The controlling leg has id 2.
initialCall_ SegmentID := 1
The first created call segment (initial CS) has ID=1.
call AddCS (initialCallSegmentID, controllingLegID, Idle,False)
Add a call segment with:
− the ID set to InitialCallSegmentID
− the ID of the controlling leg set to ControllingLegID
− the SSF−FSM in state Idle
− the call is not SCF initiated (False parameter).
Failed
Failed
Successful
SetLeg_ Location (controllingLegID, initialCallSegmentID)
The controlling leg is residing in the initial CS.
SetLeg_ Location (leg1,initial_ CallSegmentID)
The passive leg 1 is residing in the initial CS.
SetLegAssoc (legID,leg1)
Associate the remote LegId with the passive leg 1.
SetupReqInd (sirArg,csaID, legID)
to call GetCSPtr(initialCallSegmentID)
/* Pass SetupReqInd on to the created CS. */
One_ Segment
The CSA now contains one call segment.

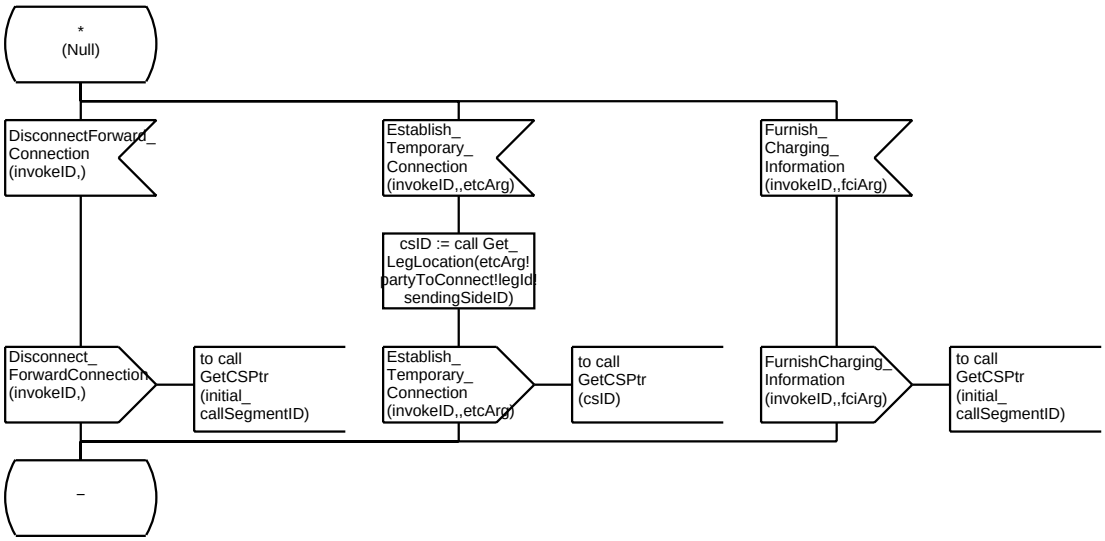

/**** PROCESSING OF INAP OPERATIONS THAT CHANGES THE CHANGES THE CALL SEGMENT ASSOCIATION. ****/
Null
Call initiated from the SCF.
Initiate_ CallAttempt (invokeID,,icaArg)
From SCF.
initialCallSegmentID :=ICAArg! newCallSegment
The first created call segment (initial CS) has ID = the newCallSegment in ICAArg.
controllingLegID := leg1
The controlling leg has id 1.
call AddCS (initialCallSegmentID, controllingLegID, Idle,True)
Add call segment with:
− the ID set to initialCallSegmentID
− the ID of the controlling leg set to controllingLegID
− the SSF−FSM in state Idle
− the call is SCF initiated (True parameter).
Failed
Successful
Failed
SetLeg_ Location (controllingLegID, icaArg! newCallSegment)
The controlling leg is residing in segment newCallSegment.
SetLegLocation (ICAArg!legToBe_ Created!sendingSideID, ICAArg!new_ CallSegment)
The passive leg LegId is residing in segment newCallSegment.
InitiateCall_ Attempt (invokeID,, icaArg)
to call GetCSPtr(icaArg!newCallSegment)
/* Pass InitiateCallAttempt on to the newly created segment. */
numOfAppls := 1
Number of active dialogues = 1.
One_ Segment
The CSA now contains one call segment.

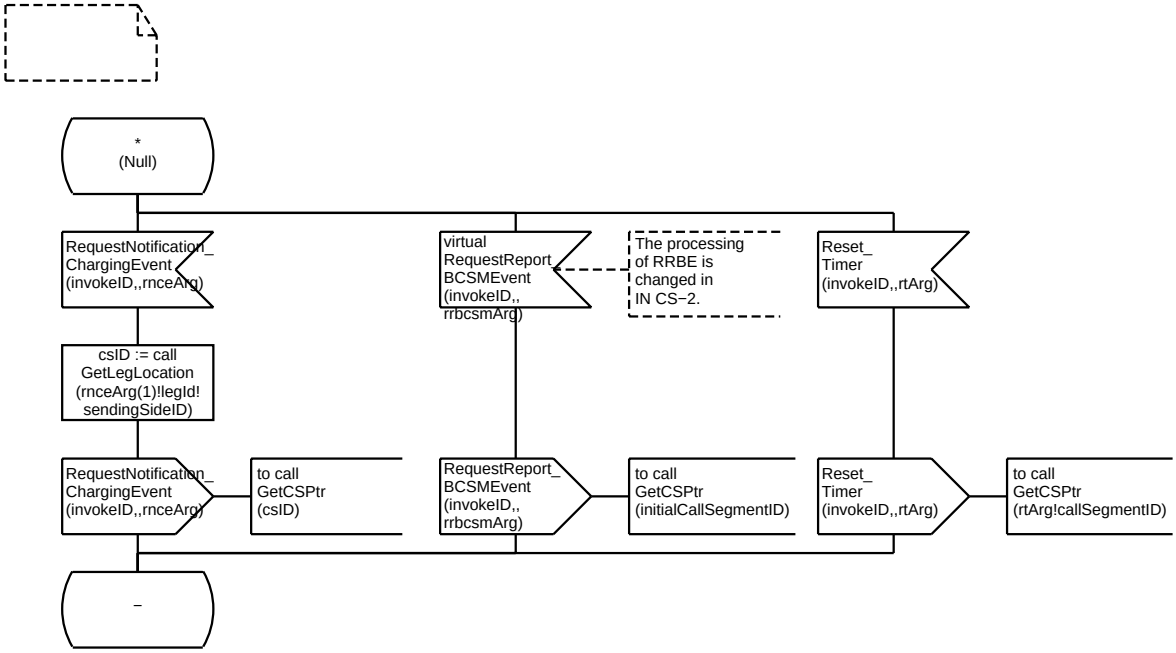

/**** HANDLING OF CALL RELEASE ****/
OneSegment
virtual
ReleaseCall
(invokeID,,rcArg)
The behaviour of
ReleaeCall is redefined
in IN CS−2.
initialCallSegment
Present(rcArg)
false
true
CSStop(,rcArg!
initialCall_
Segment)
to call GetCSPtr
(initialCallSegmentID)
/* Instruct the initial call
segment to release. */
'Error situation'
–
OneSegment
CSStop
(csID)
Indication from the CS that
the segment is released (and
the call is released).
Failed
Errror
(,ieArg)
via IH

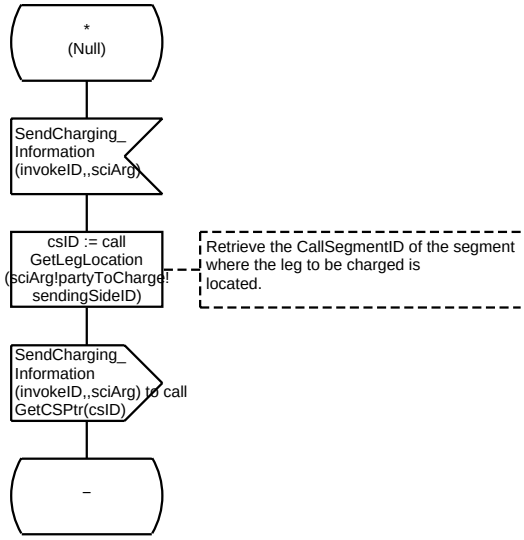

/**** ROUTING OF INAP OPERATIONS FROM THE SCF. */
*
(Null)
Apply_
Charging
(invokeID,,acArg)
CallInformation_
Request
(invokeID,,cirqArg)
partyToChargePresent
(acArg)
legIDPresent
(cirqArg)
false
true
false
true
csID:=call
GetLegLocation
(acArg!partyToCharge!
sendingSideID)
csID :=
initial_
CallSegmentID
csID:=call
GetLegLocation
(cirqArg!legID!
sendingSideID)
csID :=
initial_
CallSegmentID
Apply_
Charging
(invokeID,,acArg)
to call GetCS_
Ptr(csID)
CallInformation_
Request
(invokeID,,cirqArg)
to call GetCSPtr
(csID)
_

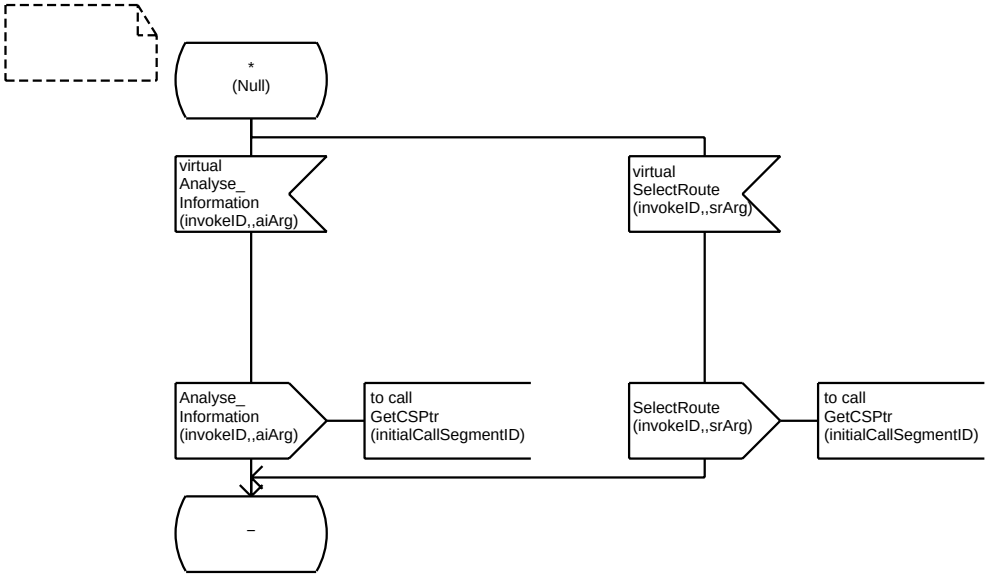

*
(Null)
Cancel
(invokeID,,cArg)
Collect_
Information
(invokeID,,ciArg)
Cancel
(invokeID,,cArg)
to call GetCSPtr
(initial_
CallSegmentID)
Collect_
Information
(invokeID,,ciArg)
to call
GetCSPtr
(initialCall_
SegmentID)
_

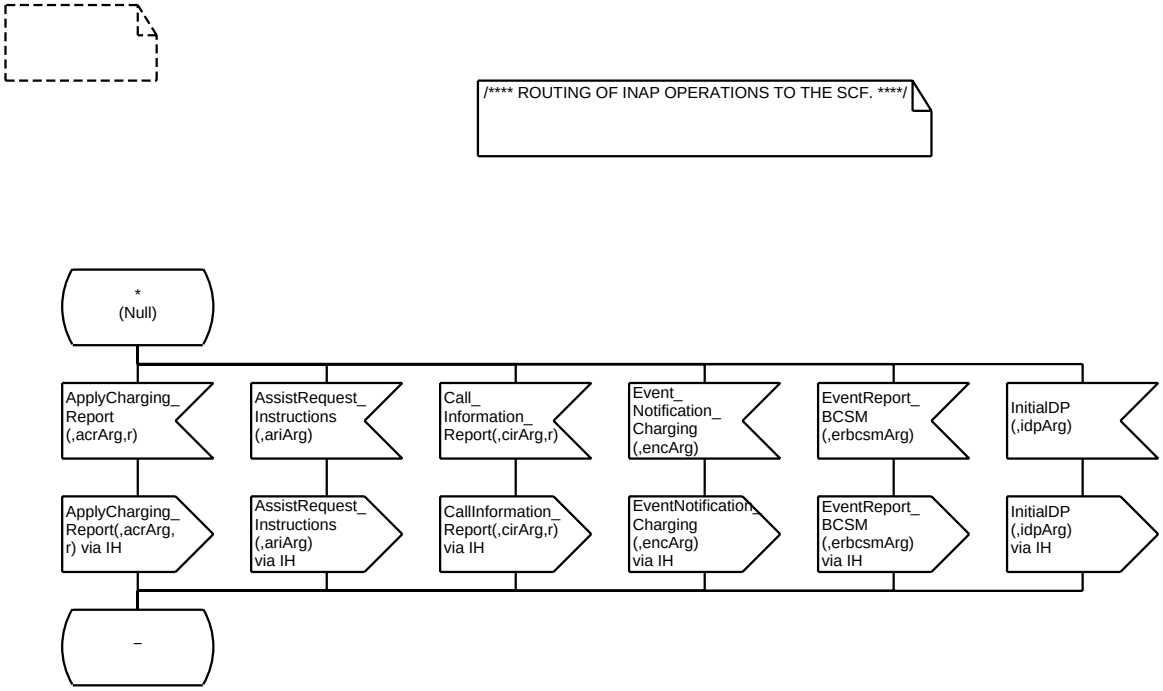

*
(Null)
virtual
Connnect
(invokeID,,coArg)
The semantics of the Connect operation is changed in IN CS−2. This transition is therefore virtual.
ConnectTo_
Resource
(invokeID,,ctrArg)
Continue
(invokeID,)
Connnect
(invokeID,,coArg) to call
GetCSPtr(initial_
CallSegmentID)
ConnectTo_
Resource
(invokeID,,ctrArg)
to call
GetCSPtr
(ctrArg!resourceAddress!
callSegmentID)
Continue
(invokeID,)
to call
GetCSPtr
(initial_
callSegmentID)
–

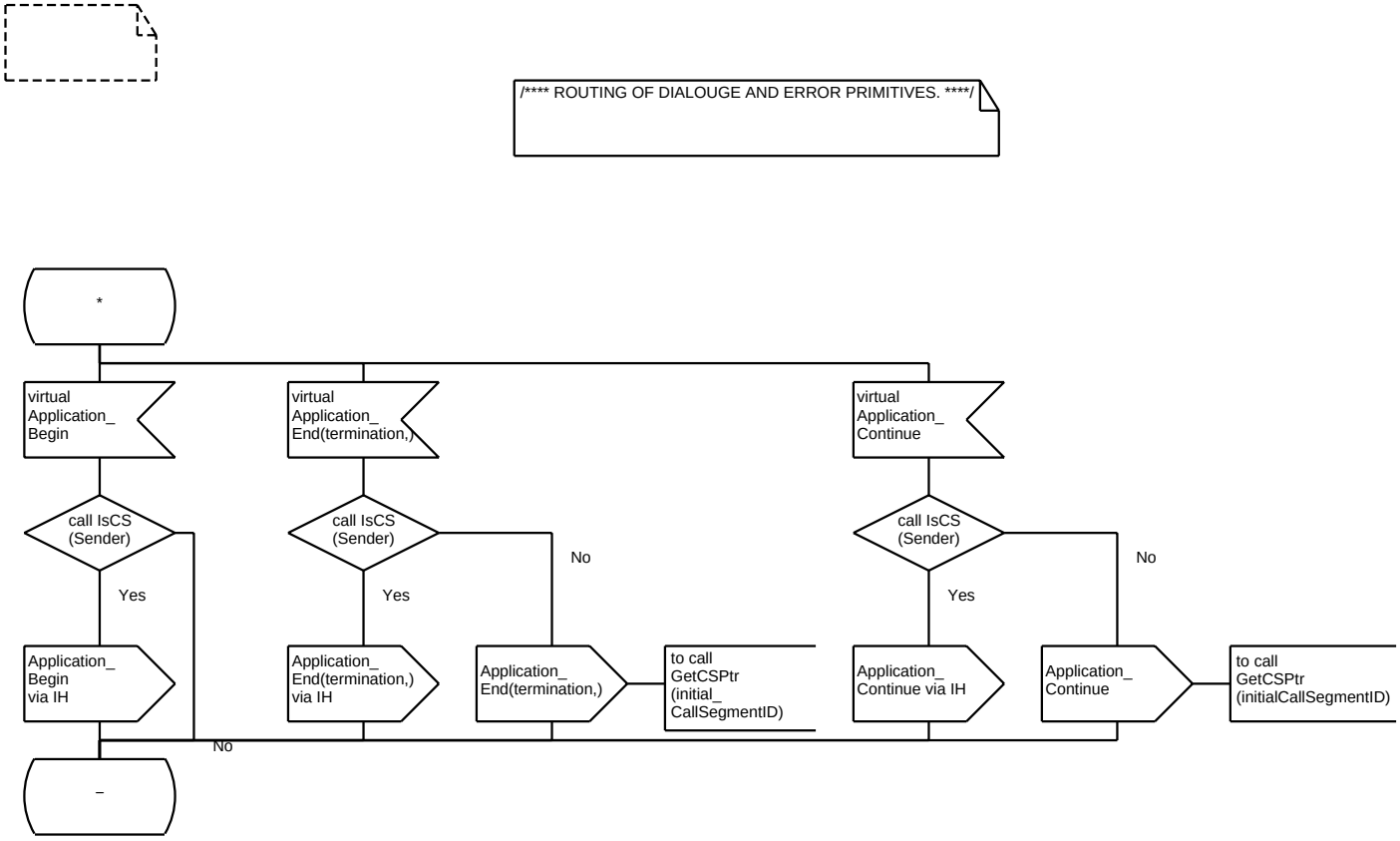

*
(Null)
DisconnectForward_
Connection
(invokeID,)
Establish_
Temporary_
Connection
(invokeID,,etcArg)
Furnish_
Charging_
Information
(invokeID,,fciArg)
csID := call Get_
LegLocation(etcArg!
partyToConnect!legId!
sendingSideID)
Disconnect_
ForwardConnection
(invokeID,)
to call
GetCSPtr
(initial_
callSegmentID)
Establish_
Temporary_
Connection
(invokeID,,etcArg)
to call
GetCSPtr
(csID)
FurnishCharging_
Information
(invokeID,,fciArg)
to call
GetCSPtr
(initial_
callSegmentID)
–

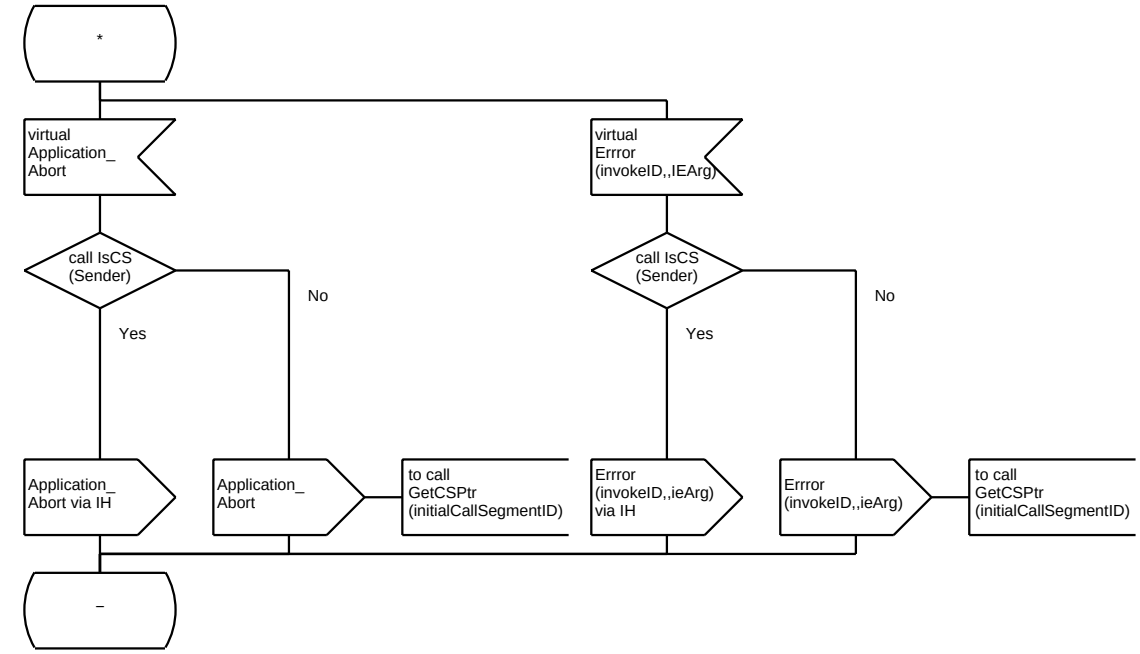

*
(Null)
RequestNotification_
ChargingEvent
(invokeID,,rnceArg)
virtual
RequestReport_
BCSMEvent
(invokeID,,
rrbcsmArg)
The processing
of RRBE is
changed in
IN CS−2.
Reset_
Timer
(invokeID,,rtArg)
csID := call
GetLegLocation
(rnceArg(1)!legId!
sendingSideID)
RequestNotification_
ChargingEvent
(invokeID,,rnceArg)
to call
GetCSPtr
(csID)
RequestReport_
BCSMEvent
(invokeID,,
rrbcsmArg)
to call
GetCSPtr
(initialCallSegmentID)
Reset_
Timer
(invokeID,,rtArg)
to call
GetCSPtr
(rtArg!callSegmentID)
–


*
(Null)
SendCharging_
Information
(invokeID,,sciArg)
csID := call
GetLegLocation
(sciArg!partyToCharge!
sendingSideID)
Retrieve the CallSegmentID of the segment
where the leg to be charged is
located.
SendCharging_
Information
(invokeID,,sciArg)
to call
GetCSPtr(csID)
–

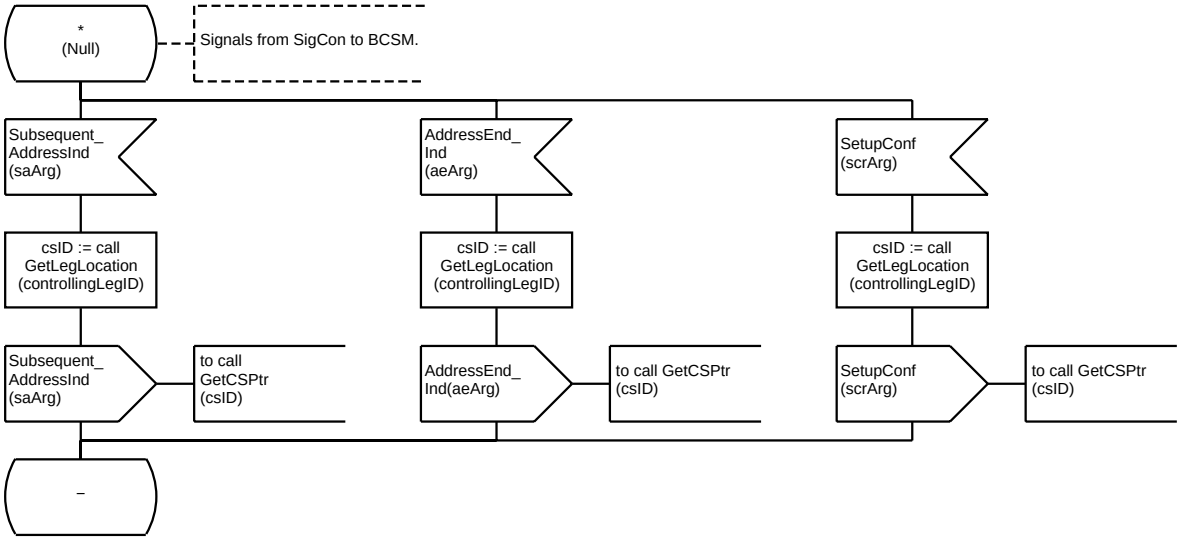

*
(Null)
virtual
Analyse_
Information
(invokeID,,aiArg)
virtual
SelectRoute
(invokeID,,srArg)
Analyse_
Information
(invokeID,,aiArg)
to call
GetCSPtr
(initialCallSegmentID)
SelectRoute
(invokeID,,srArg)
to call
GetCSPtr
(initialCallSegmentID)
_

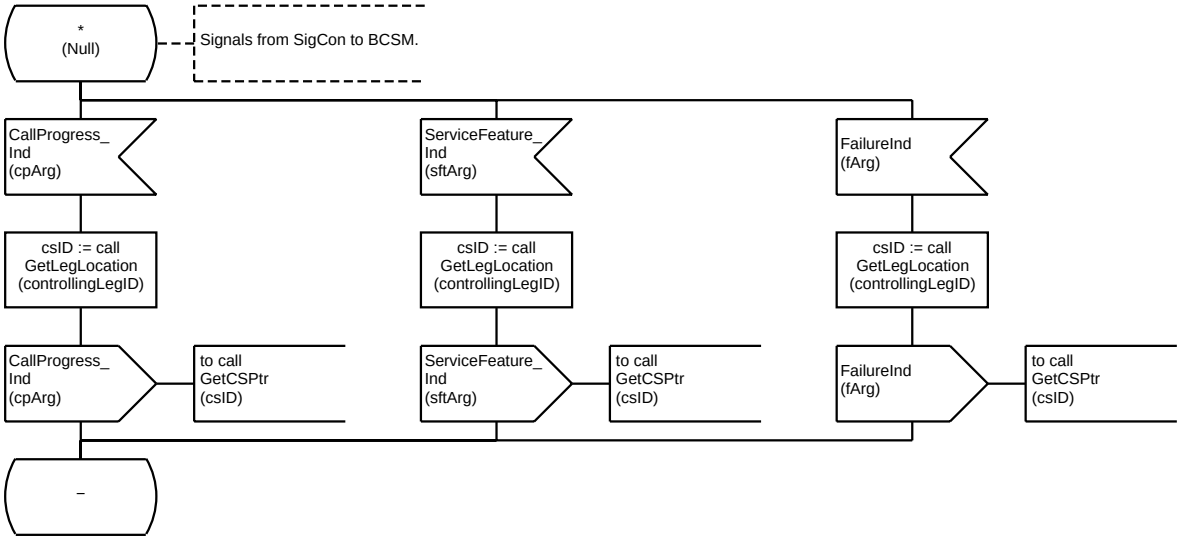

/**** ROUTING OF INAP OPERATIONS TO THE SCF. ****/
*
(Null)
ApplyCharging_
Report
(,acrArg,r)
AssistRequest_
Instructions
(,ariArg)
Call_
Information_
Report(,cirArg,r)
Event_
Notification_
Charging
(,encArg)
EventReport_
BCSM
(,erbcsmArg)
InitialDP
(,idpArg)
ApplyCharging_
Report(,acrArg,
r) via IH
AssistRequest_
Instructions
(,ariArg)
via IH
CallInformation_
Report(,cirArg,r)
via IH
EventNotification_
Charging
(,encArg)
via IH
EventReport_
BCSM
(,erbcsmArg)
via IH
InitialDP
(,idpArg)
via IH
_

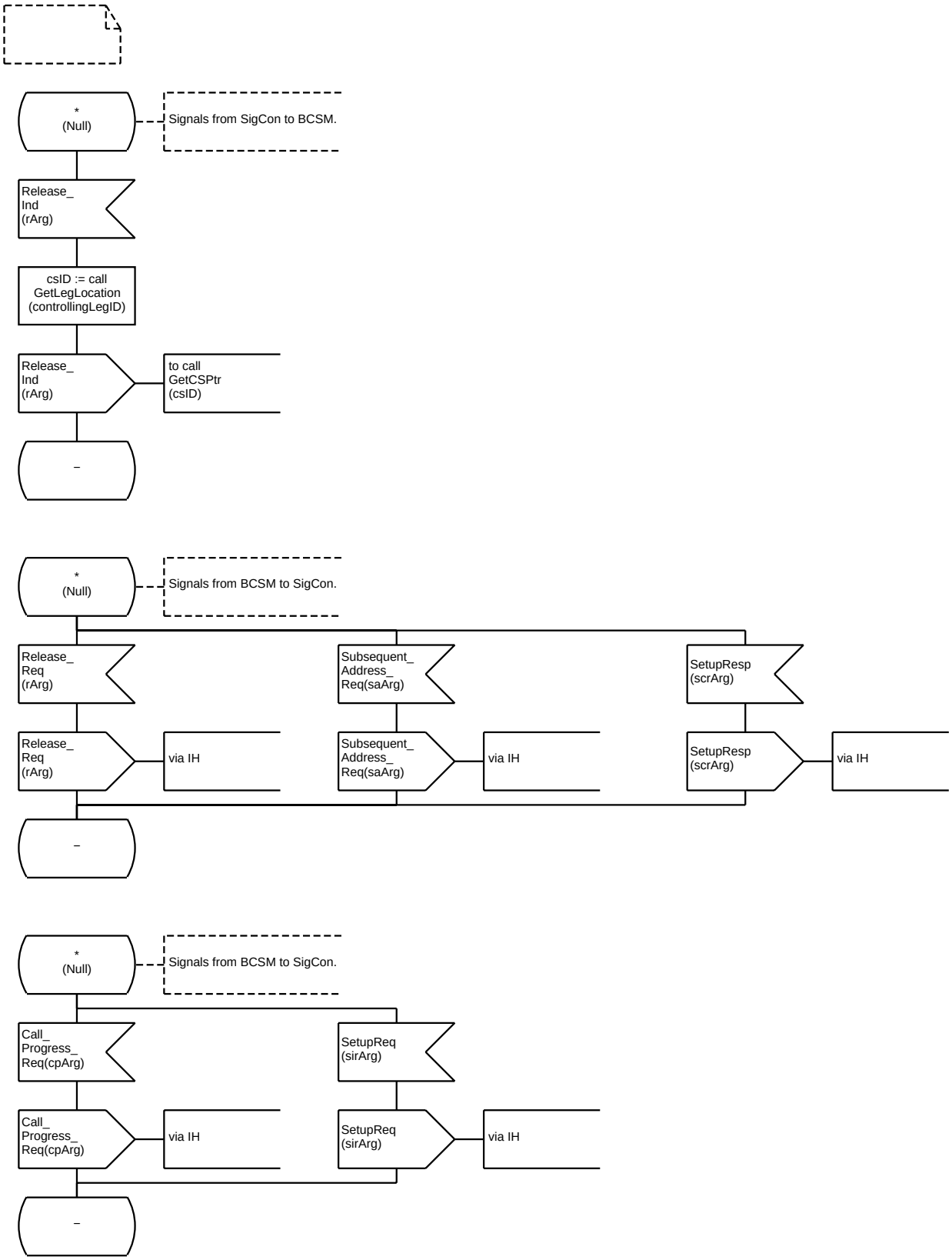

/**** ROUTING OF DIALOUGE AND ERROR PRIMITIVES. ****/
*
virtual Application_ Begin
virtual Application_ End(termination,)
virtual Application_ Continue
call IsCS (Sender)
call IsCS (Sender)
call IsCS (Sender)
Yes
No
Yes
No
Yes
Application_ Begin via IH
Application_ End(termination,) via IH
Application_ End(termination,)
to call GetCSPtr (initial_ CallSegmentID)
Application_ Continue via IH
Application_ Continue
to call GetCSPtr (initialCallSegmentID)
No
–

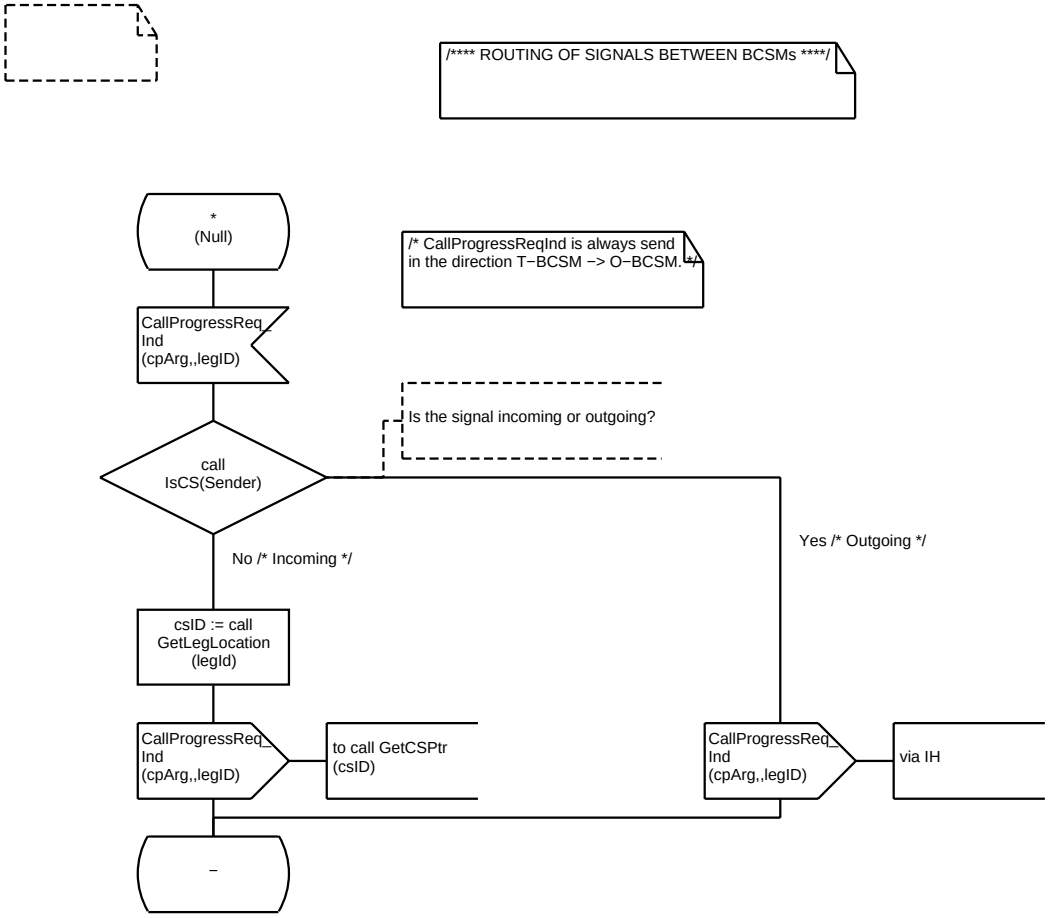

*
virtual Application_ Abort
virtual Errror (invokeID,,IEArg)
call IsCS (Sender)
call IsCS (Sender)
Yes
No
Yes
No
Application_ Abort via IH
Application_ Abort
to call GetCSPtr (initialCallSegmentID)
Errror (invokeID,,ieArg) via IH
Errror (invokeID,,ieArg)
to call GetCSPtr (initialCallSegmentID)
–

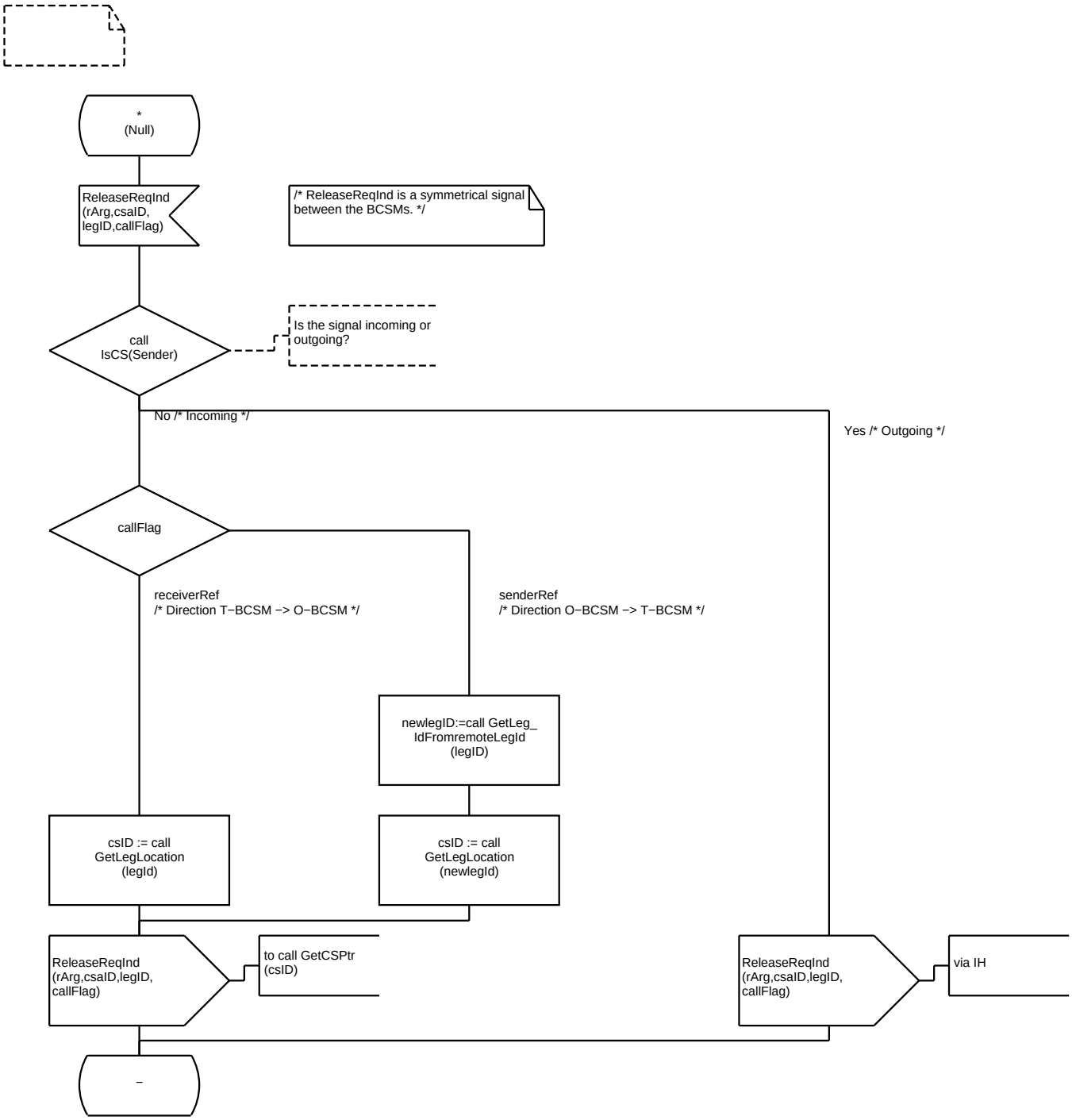
/* ROUTING OF SIGNALS TO/FROM SIGCON (VIA THE IH). */

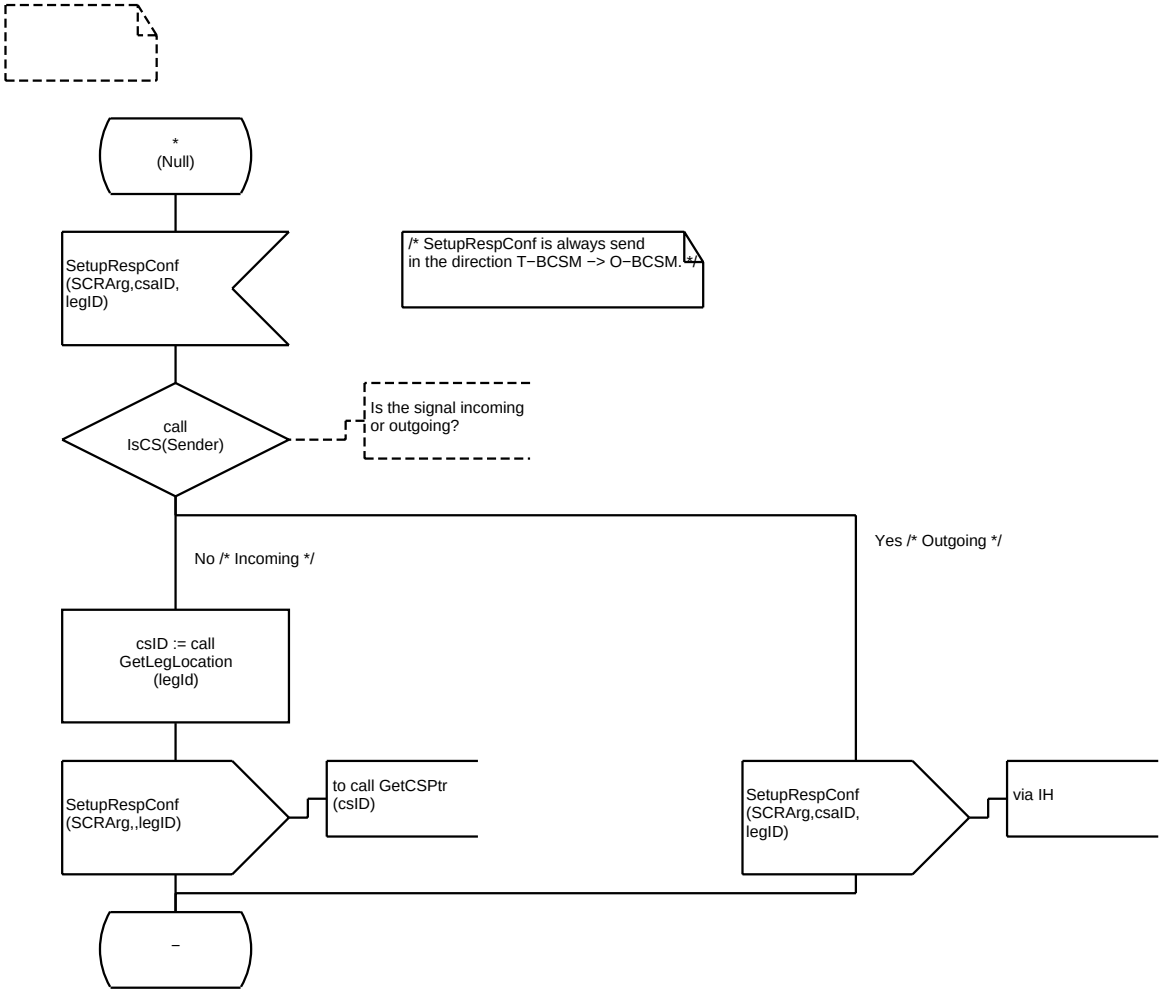


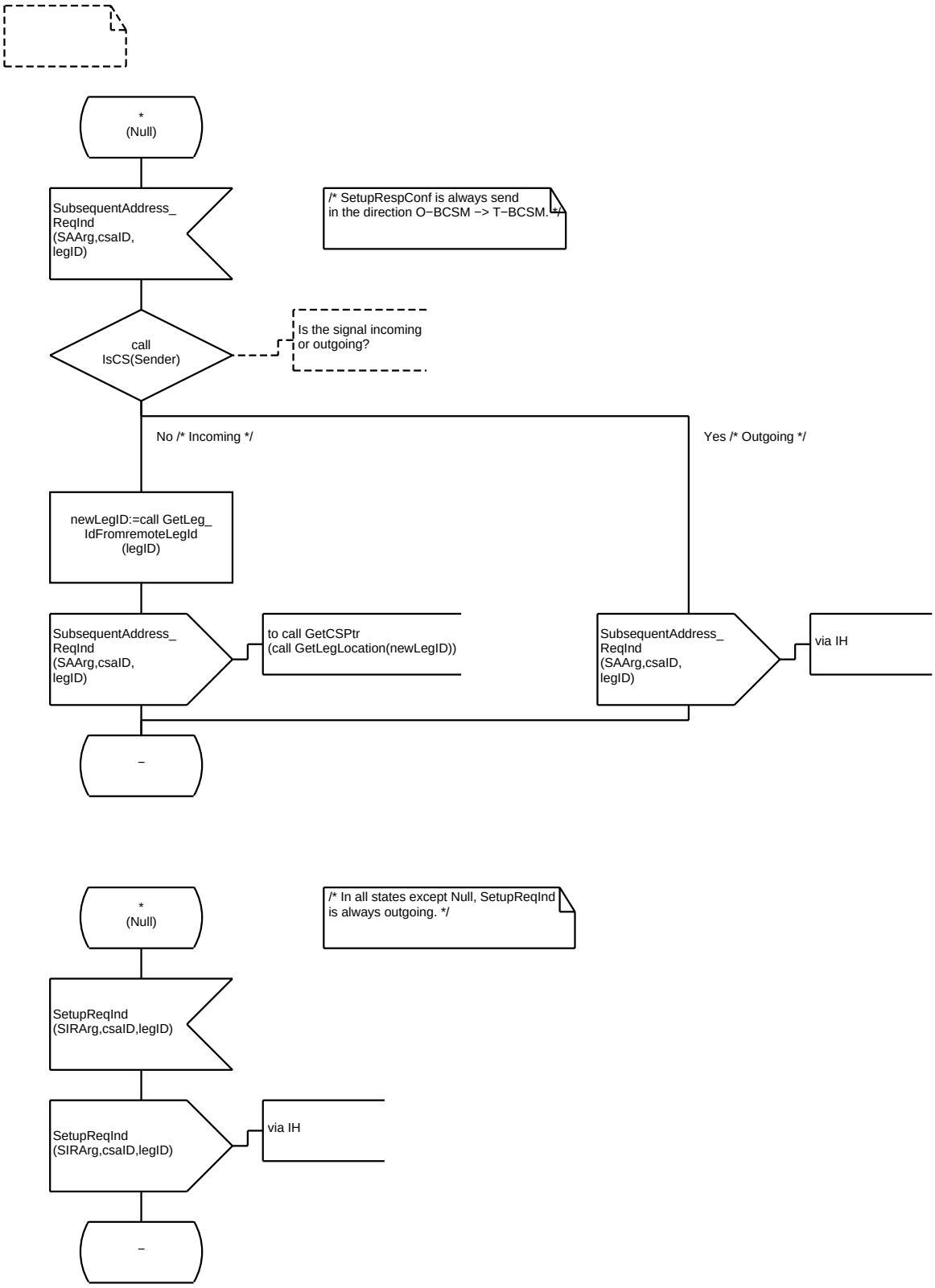


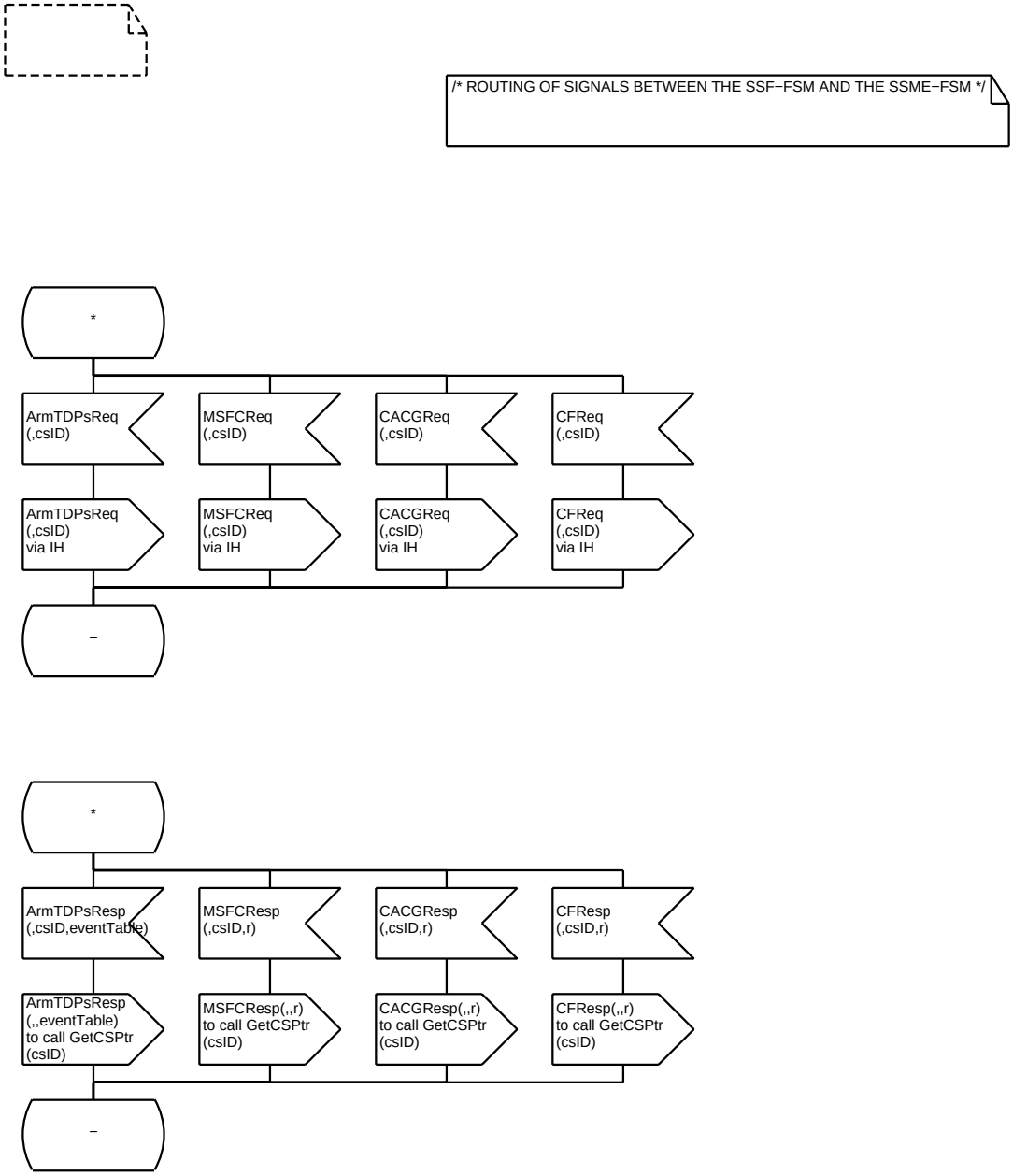

*
(Null)
Signals from SigCon to BCSM.
Release_
Ind
(rArg)
csID := call
GetLegLocation
(controllingLegID)
Release_
Ind
(rArg)
to call
GetCSPtr
(csID)
–
*
(Null)
Signals from BCSM to SigCon.
Release_
Req
(rArg)
Subsequent_
Address_
Req(saArg)
SetupResp
(scrArg)
Release_
Req
(rArg)
via IH
Subsequent_
Address_
Req(saArg)
via IH
SetupResp
(scrArg)
via IH
–
*
(Null)
Signals from BCSM to SigCon.
Call_
Progress_
Req(cpArg)
SetupReq
(sirArg)
Call_
Progress_
Req(cpArg)
via IH
SetupReq
(sirArg)
via IH
–


/**** ROUTING OF SIGNALS BETWEEN BCSMs ****/
*
(Null)
/* CallProgressReqInd is always send
in the direction T−BCSM −> O−BCSM. */
CallProgressReq_
Ind
(cpArg,,legID)
Is the signal incoming or outgoing?
call
IsCS(Sender)
Yes /* Outgoing */
No /* Incoming */
csID := call
GetLegLocation
(legId)
CallProgressReq_
Ind
(cpArg,,legID)
to call GetCSPtr
(csID)
CallProgressReq_
Ind
(cpArg,,legID)
via IH
–

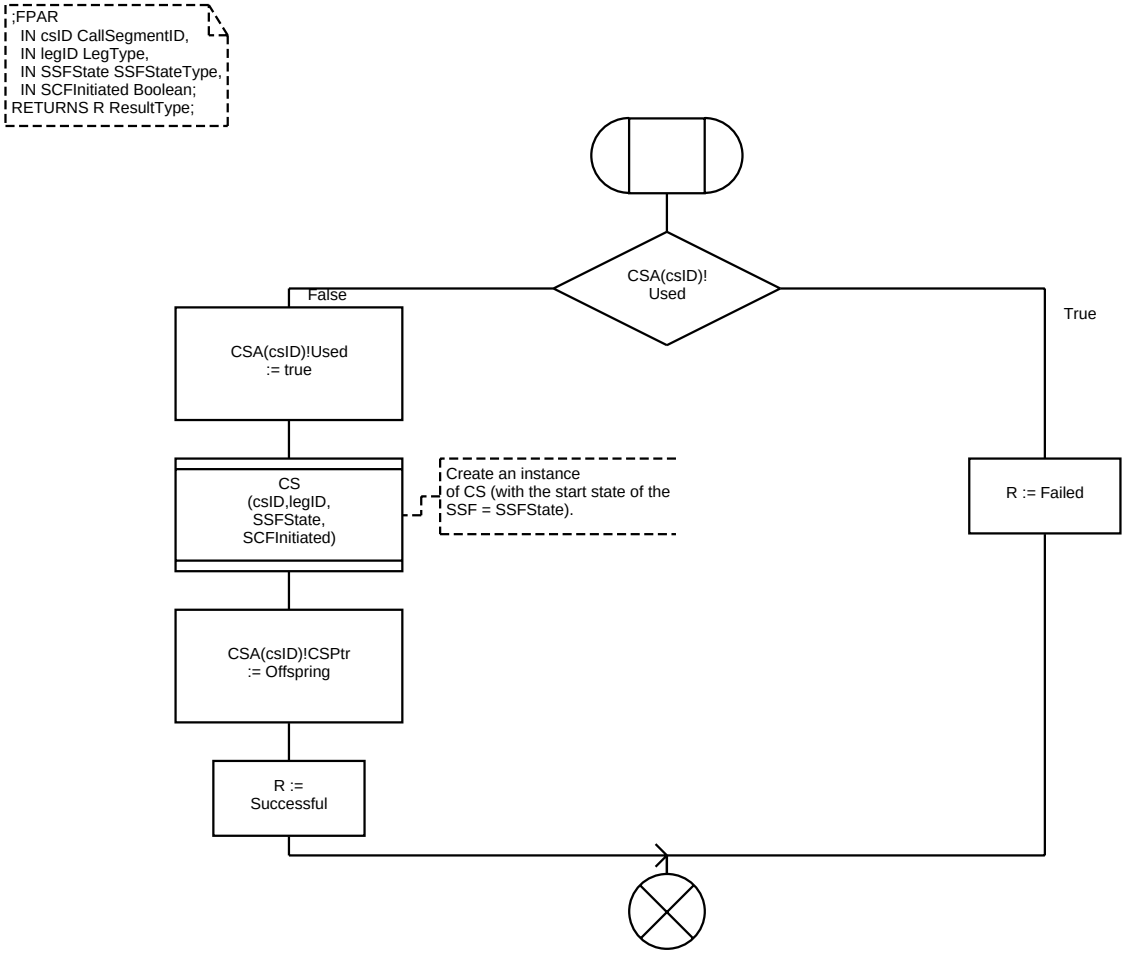

*
(Null)
ReleaseReqInd
(rArg,csaID,
legID,callFlag)
/* ReleaseReqInd is a symmetrical signal between the BCSMs. */
call
IsCS(Sender)
Is the signal incoming or outgoing?
No /* Incoming */
Yes /* Outgoing */
callFlag
receiverRef
/* Direction T−BCSM −> O−BCSM */
senderRef
/* Direction O−BCSM −> T−BCSM */
newlegID:=call GetLeg_
IdFromremoteLegId
(legID)
csID := call
GetLegLocation
(legId)
csID := call
GetLegLocation
(newlegId)
ReleaseReqInd
(rArg,csaID,legID,
callFlag)
to call GetCSPtr
(csID)
ReleaseReqInd
(rArg,csaID,legID,
callFlag)
via IH
−


*
(Null)
SetupRespConf
(SCRArg,csaID,
legID)
/* SetupRespConf is always send
in the direction T−BCSM −> O−BCSM. */
call
IsCS(Sender)
Is the signal incoming
or outgoing?
No /* Incoming */
Yes /* Outgoing */
csID := call
GetLegLocation
(legId)
SetupRespConf
(SCRArg,,legID)
to call GetCSPtr
(csID)
SetupRespConf
(SCRArg,csaID,
legID)
via IH
−


*
(Null)
SubsequentAddress_
ReqInd
(SAArg,csaID,
legID)
/* SetupRespConf is always send
in the direction O−BCSM −> T−BCSM. */
call
IsCS(Sender)
Is the signal incoming
or outgoing?
No /* Incoming */
Yes /* Outgoing */
newLegID:=call GetLeg_
IdFromremoteLegId
(legID)
SubsequentAddress_
ReqInd
(SAArg,csaID,
legID)
to call GetCSPtr
(call GetLegLocation(newLegID))
SubsequentAddress_
ReqInd
(SAArg,csaID,
legID)
via IH
_
*
(Null)
/* In all states except Null, SetupReqInd
is always outgoing. */
SetupReqInd
(SIRArg,csaID,legID)
SetupReqInd
(SIRArg,csaID,legID)
via IH
_

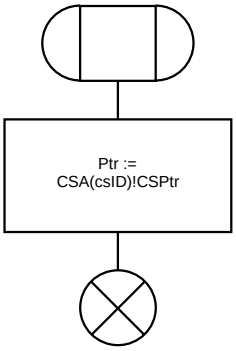

/* ROUTING OF SIGNALS BETWEEN THE SSF−FSM AND THE SSME−FSM */
*
ArmTDPsReq (,csID)
MSFCReq (,csID)
CACGReq (,csID)
CFReq (,csID)
ArmTDPsReq (,csID) via IH
MSFCReq (,csID) via IH
CACGReq (,csID) via IH
CFReq (,csID) via IH
−
*
ArmTDPsResp (,csID,eventTable)
MSFCResp (,csID,r)
CACGResp (,csID,r)
CFResp (,csID,r)
ArmTDPsResp (,,eventTable) to call GetCSPtr (csID)
MSFCResp(,,r) to call GetCSPtr (csID)
CACGResp(,,r) to call GetCSPtr (csID)
CFResp(,,r) to call GetCSPtr (csID)
−

Procedure <<Block Type SSF_CCF/Process Type CallSegmentAssociation>> AddCS 1(1)

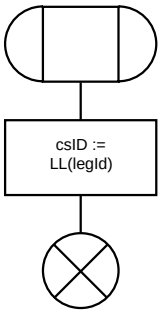



;FPAR
IN LegId LegType,
IN csID CallSegmentID;
LL(LegId)
:= csID

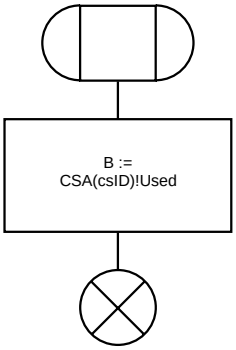
```
;FPAR
  IN csID CallSegmentID,
RETURNS Ptr PId;
```



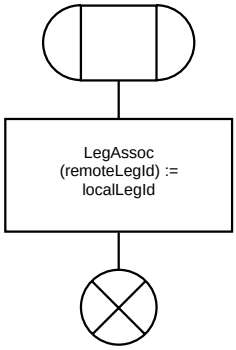
```
;FPAR
  IN legId LegType;
RETURNS csID CallSegmentID;
```



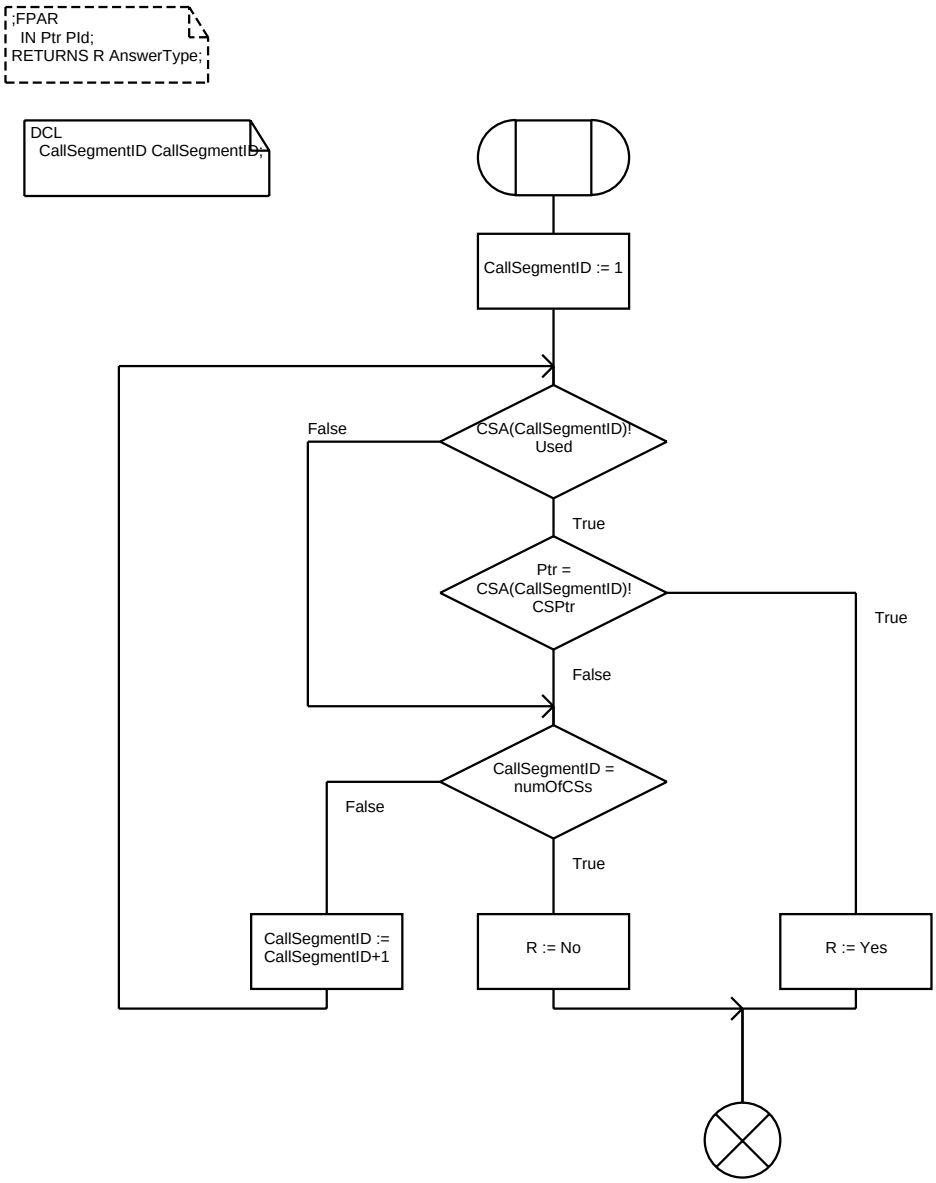
```
;FPAR
  IN csID CallSegmentID,
RETURNS B Boolean;
```



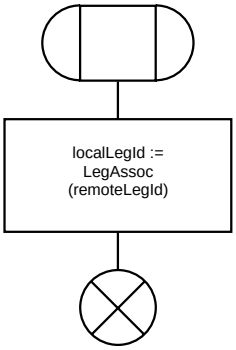
```
;FPAR
  IN remoteLegId LegType,
  IN localLegId LegType;
```



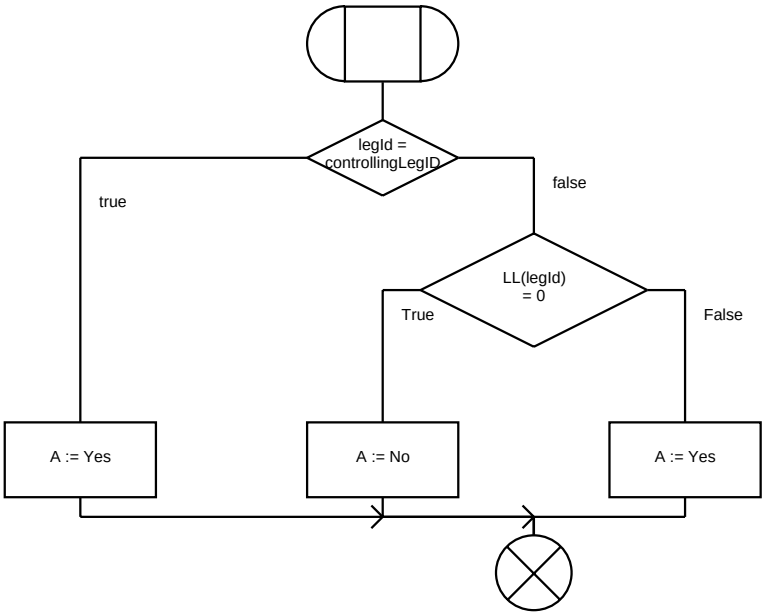
Procedure IsCS

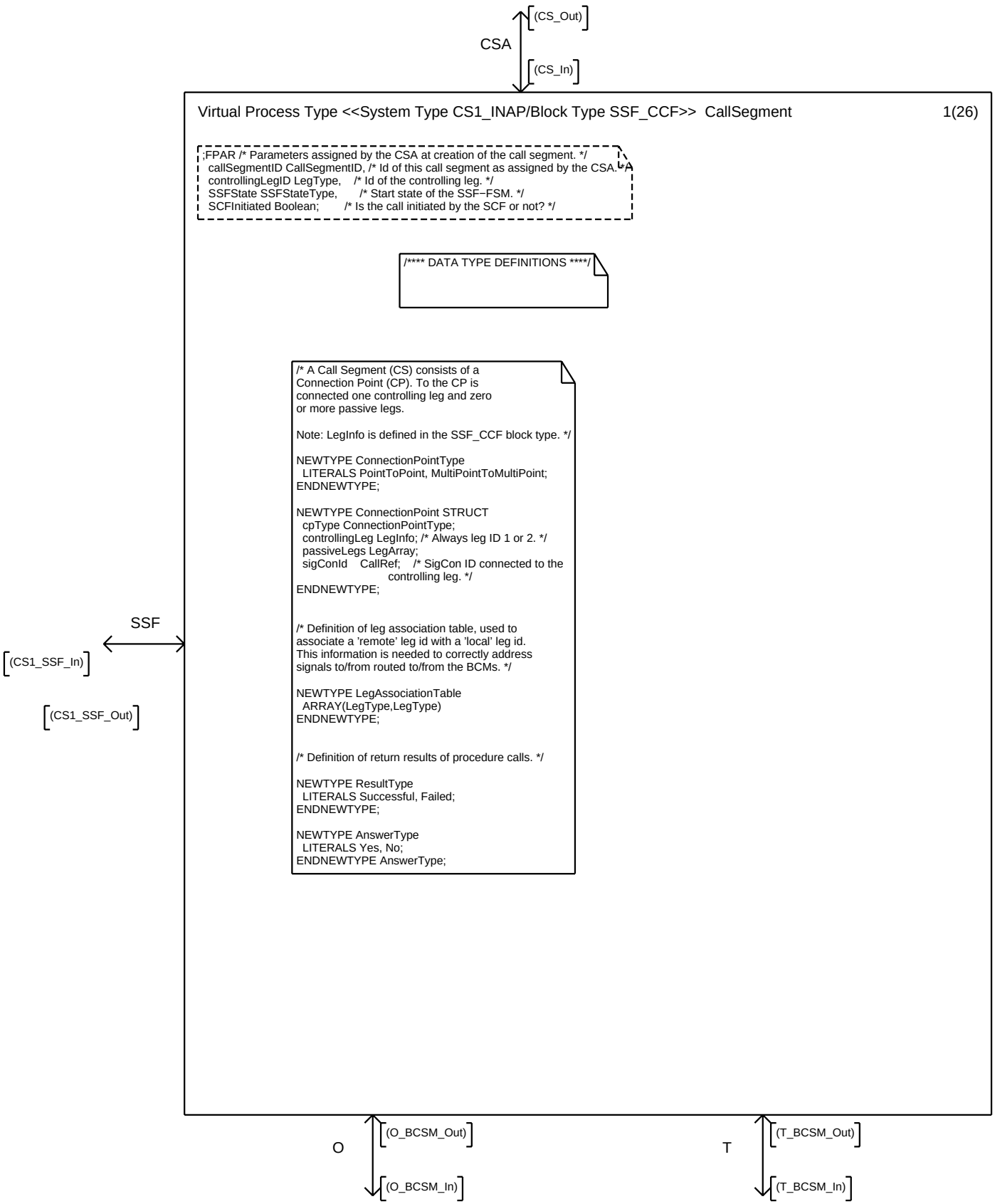


```
;FPAR
  IN remoteLegId LegType;
RETURNS localLegId LegType;
```



```
;FPAR
  IN legId LegType;
RETURNS A AnswerType;
```




CSA
(CS_Out)
(CS_In)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment
1(26)
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
SSFState SSFStateType, /* Start state of the SSF−FSM. */
SCFInitiated Boolean; /* Is the call initiated by the SCF or not? */
/**** DATA TYPE DEFINITIONS ****/
/* A Call Segment (CS) consists of a
Connection Point (CP). To the CP is
connected one controlling leg and zero
or more passive legs.
Note: LegInfo is defined in the SSF_CCF block type. */
NEWTYPE ConnectionPointType
LITERALS PointToPoint, MultiPointToMultiPoint;
ENDNEWTYPE;
NEWTYPE ConnectionPoint STRUCT
cpType ConnectionPointType;
controllingLeg LegInfo; /* Always leg ID 1 or 2. */
passiveLegs LegArray;
sigConId CallRef; /* SigCon ID connected to the
controlling leg. */
ENDNEWTYPE;
/* Definition of leg association table, used to
associate a 'remote' leg id with a 'local' leg id.
This information is needed to correctly address
signals to/from routed to/from the BCMs. */
NEWTYPE LegAssociationTable
ARRAY(LegType,LegType)
ENDNEWTYPE;
/* Definition of return results of procedure calls. */
NEWTYPE ResultType
LITERALS Successful, Failed;
ENDNEWTYPE;
NEWTYPE AnswerType
LITERALS Yes, No;
ENDNEWTYPE AnswerType;
SSF
(CS1_SSF_In)
(CS1_SSF_Out)
O
(O_BCSM_Out)
(O_BCSM_In)
T
(T_BCSM_Out)
(T_BCSM_In)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

```
/**** VARIABLE DECLARATIONS ****/
```

```
DCL
  /* The connection point of this
     call segment. */
  CP ConnectionPoint,

  /* Pointer to the SSF_FSM associated
     with this segment. */
  SSF PId,

  /* The association of local and remote
     leg ids. */
  LegAssoc LegAssociationTable,

  /* Pointer to the CSA in which this call
     segment belongs. */
  CSA PId,

  /* Dialogue status. */
  DialogueActive Boolean := false,

  /* Other variables */
  csaID CSAID,
  invokeID InvokeID,
  termination Boolean,
  obcsmPars OBCSMPars,
  bcsmStopped Boolean := false,
  legId LegType,
  newLegID LegType,
  pic PICArg,
  dp DPArg,
  callFlag CallFlag,
  eventTable EventTableType,
  r Boolean;
```

```
DCL
  /* IN CS−1 operation arguments. */
  acArg ApplyChargingArg,
  acrArg ApplyChargingReportArg,
  ariArg AssistRequestInstructionsArg,
  aiArg AnalyseInformationArg,
  cirArg CallInformationReportArg,
  cirqArg CallInformationRequestArg,
  cArg CancelArg,
  ciArg CollectInformationArg,
  coArg ConnectArg,
  ctrArg ConnectToResourceArg,
  etcArg EstablishTemporaryConnectionArg,
  encArg EventNotificationChargingArg,
  erBCSMArg EventReportBCSMArg,
  fciArg FurnishChargingInformationArg,
  idpArg InitialDPArg,
  icaArg InitiateCallAttemptArg,
  rcArg ReleaseCallArg,
  rnceARg RequestNotificationChargingEventArg,
  rrBCSMEArg RequestReportBCSMEventArg,
  rtARg ResetTimerArg,
  sciArg SendChargingInformationArg,
  sfArg SelectFacilityArg,
  srArg SelectRouteArg,
  ieArg ErrorArg;
```

```
DCL
  /* Signalling control primitive parameters. */
  AEArg AddressEndType,
  CPArg CallProgressType,
  FArg FailureType,
  RArg ReleaseType,
  SFtArg ServiceFeatureType,
  SIRArg SetupIRType,
  SCRArg SetupCRType,
  SAArg SubsequentAddressType;
```

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

```
/**** DECLARATION OF OPERATIONS ****/
```

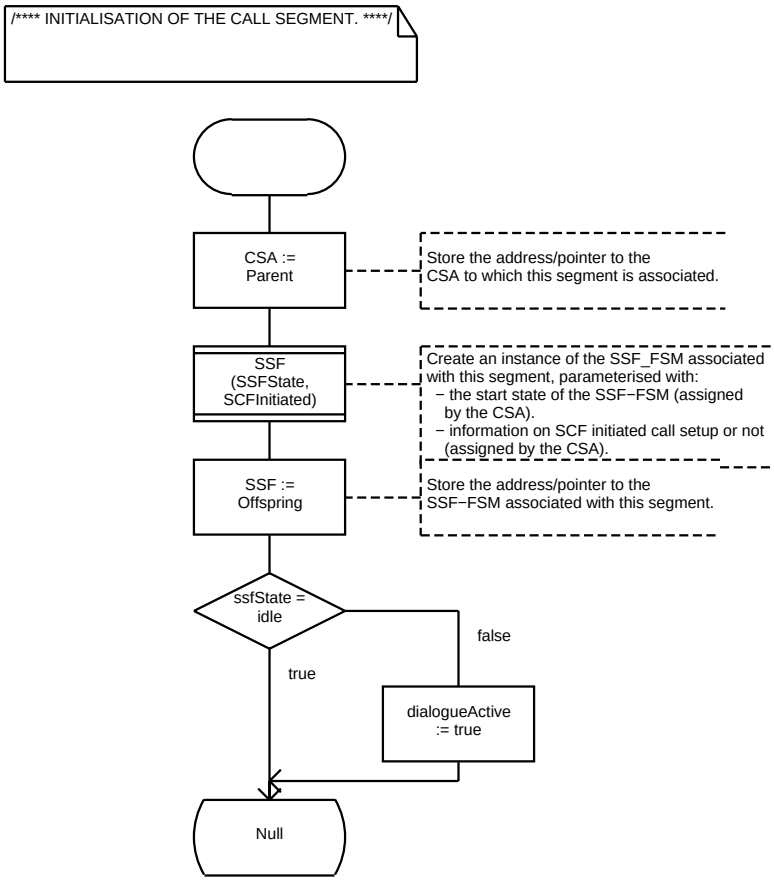
```
/* Operations on legs */
```

- AddLeg — Adds a leg to a call segment (connection point).
- Remove_Leg — Removes a leg.
- SetLeg_Status — Sets the status of a leg.
- Release_AllLegs — Removes all legs associated with this segment.
- GetLeg_Status — Returns the status of a leg.
- SetLeg_Assoc — Associates a remote leg id with a local leg id.
- SetLeg_Ptr — Connects the leg to a BCSM instance.
- GetLegId_from_Remote_LegId — Retrieves the local leg id from a given remote leg id.
- GetLeg_Ptr — Returns the pointer to a BCSM instance.
- IsBCSM — Predicate determining whether a sender of a signal is a BCSM or not.
- Get_Passive_LegId — Returns the id of the passive leg in the connection point.

```
/* Procedures for building the information
to be passed to an O−BCSM at creation time. */
```

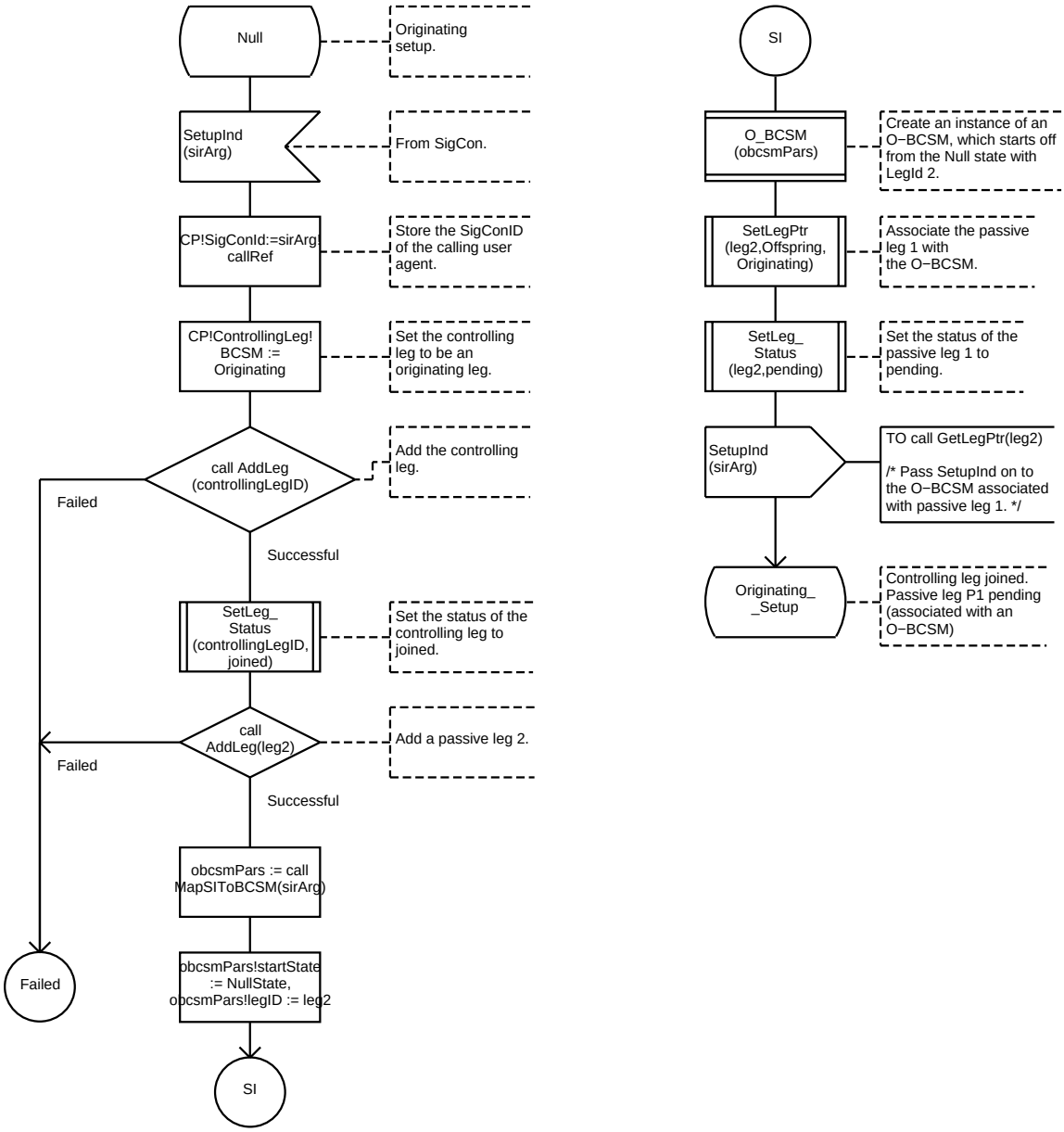
- MapConnect_ToBCSM
- MapSI_ToBCSM

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

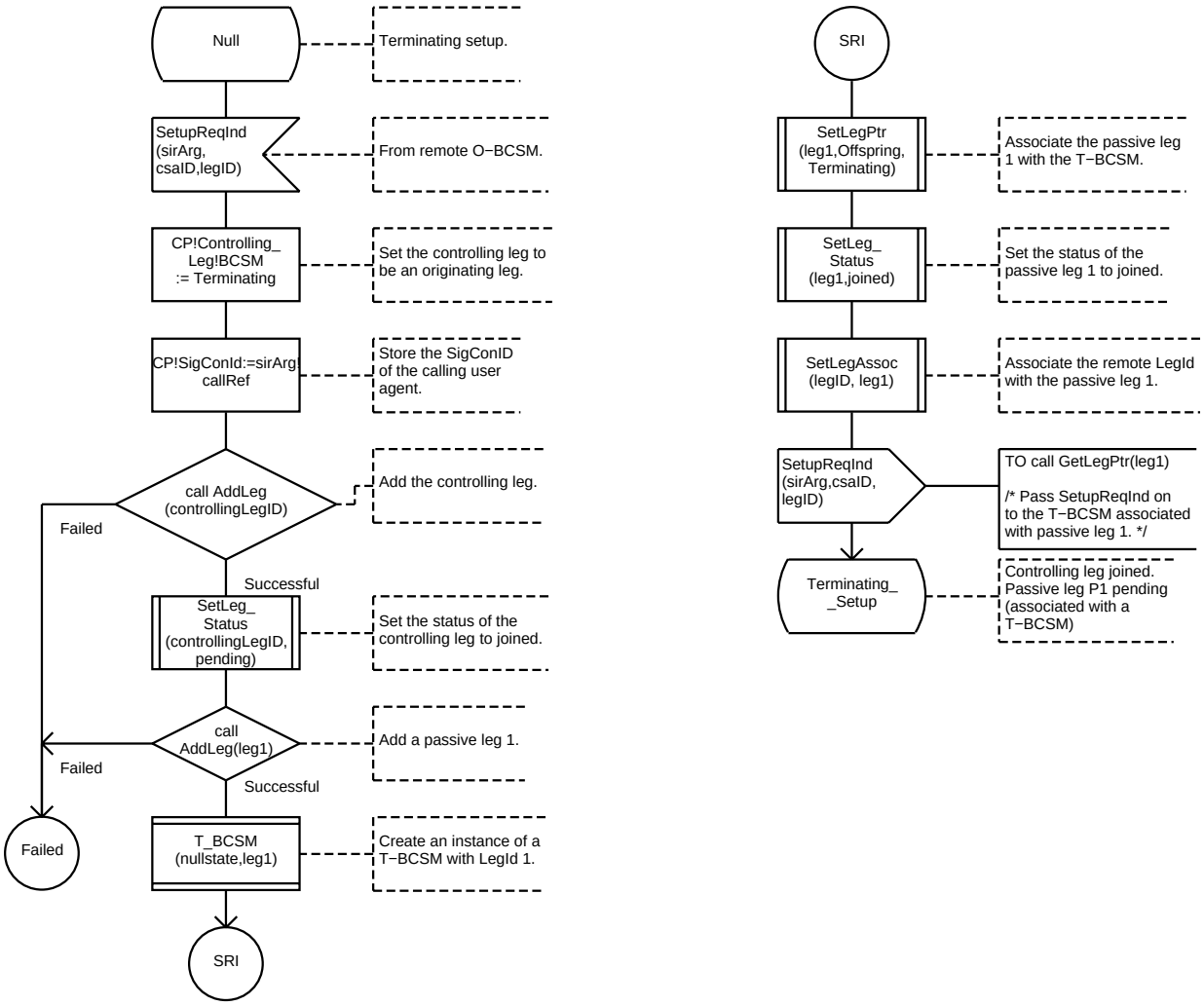


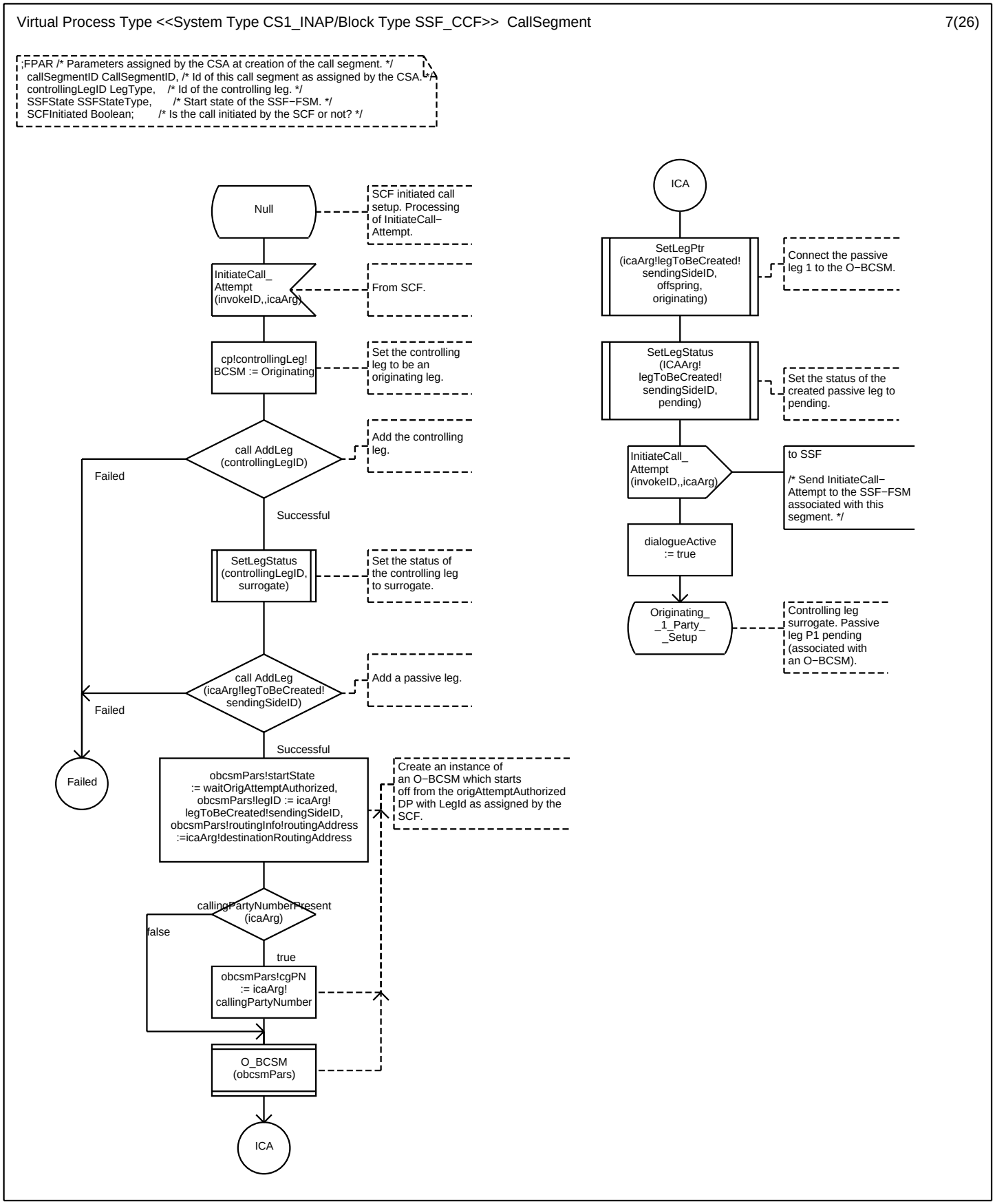
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

/**** HANDLING OF CALL SETUP. ****/



```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

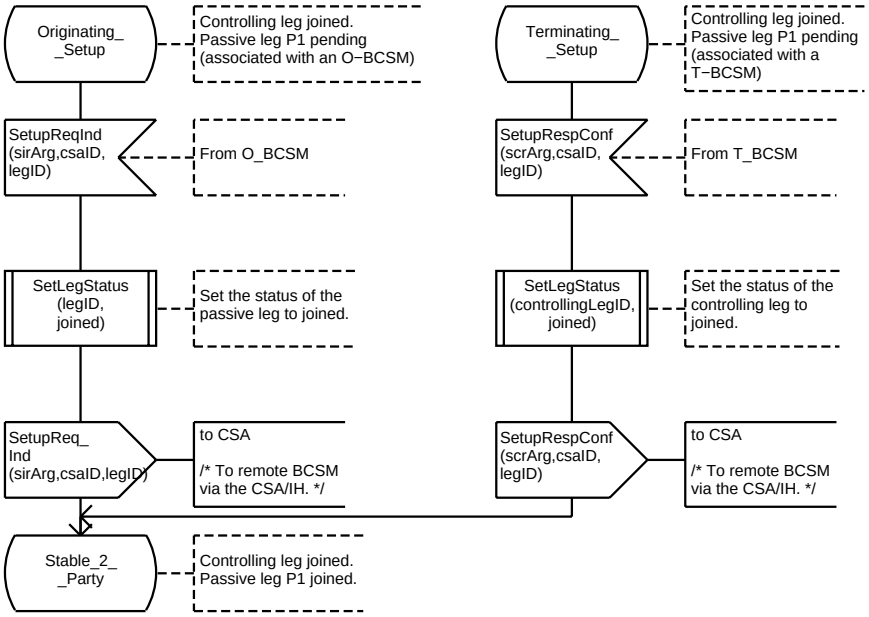



;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
SSFState SSFStateType, /* Start state of the SSF−FSM. */
SCFInitiated Boolean; /* Is the call initiated by the SCF or not? */
Null
SCF initiated call setup. Processing of InitiateCall− Attempt.
InitiateCall_ Attempt (invokeID,,icaArg)
From SCF.
cp!controllingLeg! BCSM := Originating
Set the controlling leg to be an originating leg.
call AddLeg (controllingLegID)
Add the controlling leg.
Failed
Successful
SetLegStatus (controllingLegID, surrogate)
Set the status of the controlling leg to surrogate.
call AddLeg (icaArg!legToBeCreated! sendingSideID)
Add a passive leg.
Failed
Failed
Successful
obcsmPars!startState := waitOrigAttemptAuthorized, obcsmPars!legID := icaArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=icaArg!destinationRoutingAddress
Create an instance of an O−BCSM which starts off from the origAttemptAuthorized DP with LegId as assigned by the SCF.
callingPartyNumberPresent (icaArg)
false
true
obcsmPars!cgPN := icaArg! callingPartyNumber
O_BCSM (obcsmPars)
ICA
ICA
SetLegPtr (icaArg!legToBeCreated! sendingSideID, offspring, originating)
Connect the passive leg 1 to the O−BCSM.
SetLegStatus (ICAArg! legToBeCreated! sendingSideID, pending)
Set the status of the created passive leg to pending.
InitiateCall_ Attempt (invokeID,,icaArg)
to SSF
/* Send InitiateCall− Attempt to the SSF−FSM associated with this segment. */
dialogueActive := true
Originating_ _1_Party_ _Setup
Controlling leg surrogate. Passive leg P1 pending (associated with an O−BCSM).

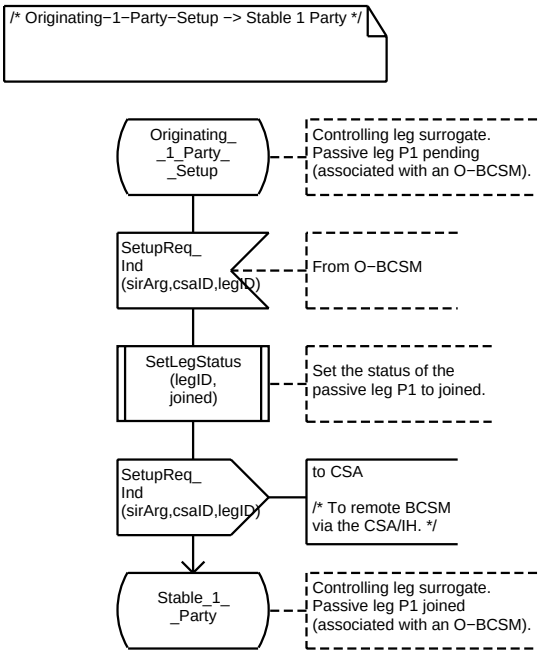
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

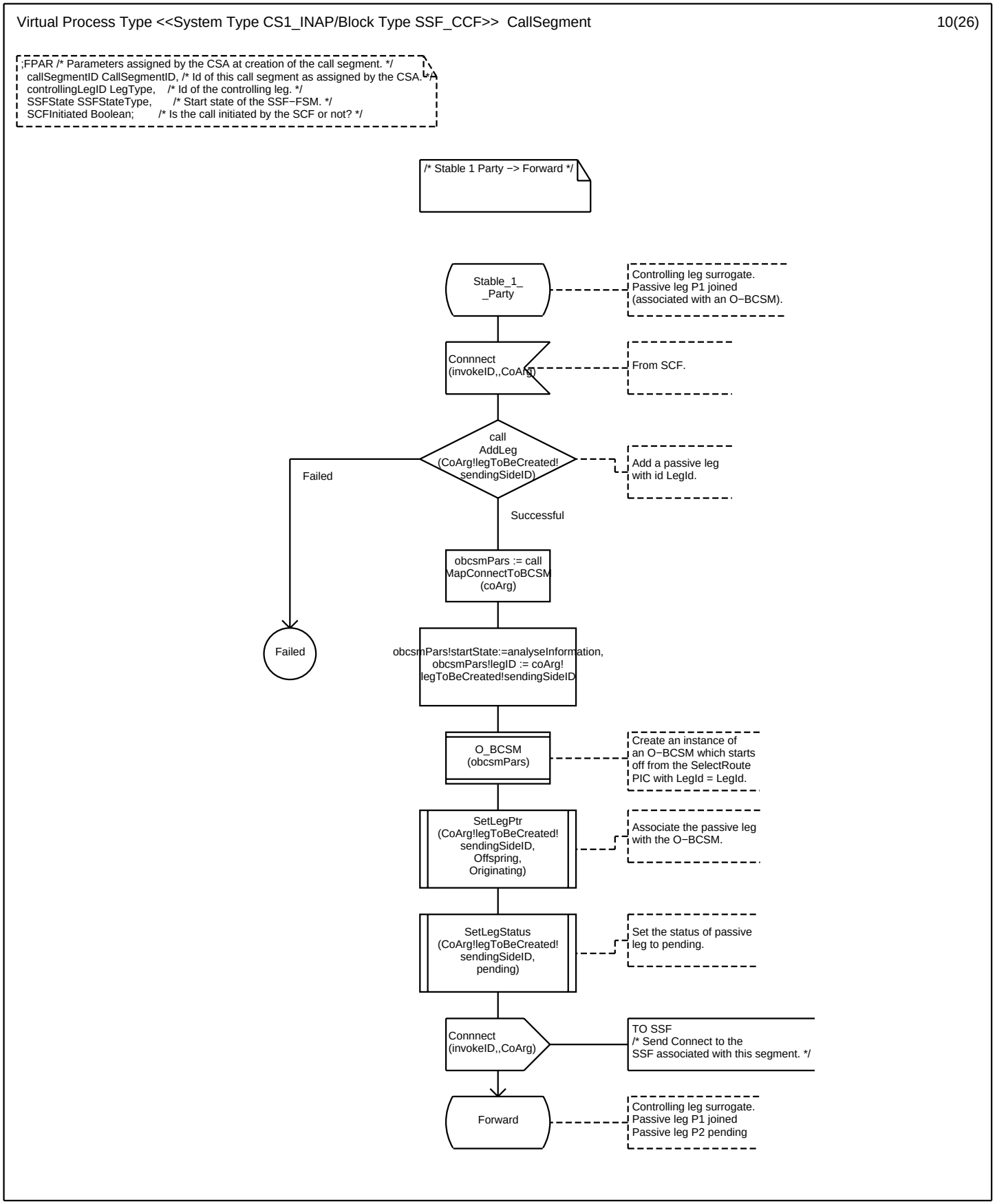
```
/**** TRANSITIONS BETWEEN THE CONNECTION VIEW STATES OF THE CALL SEGMENT. ****/

/* Note: The states of the call segment are defined in Q.1224. */
```

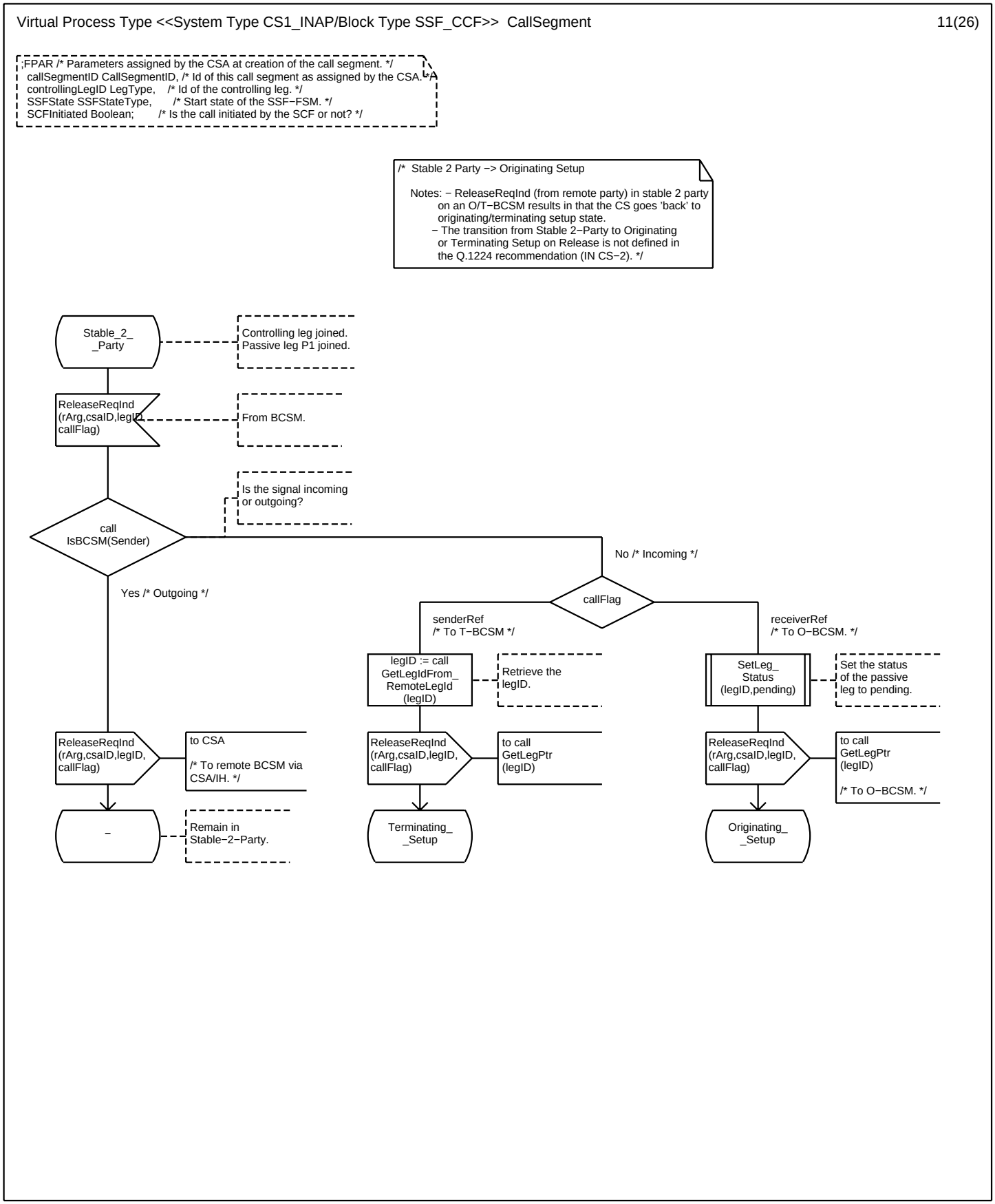
```
/* Originating Setup −> Stable 2 Party
   Terminating Setup −> Stable 2 Party */
```



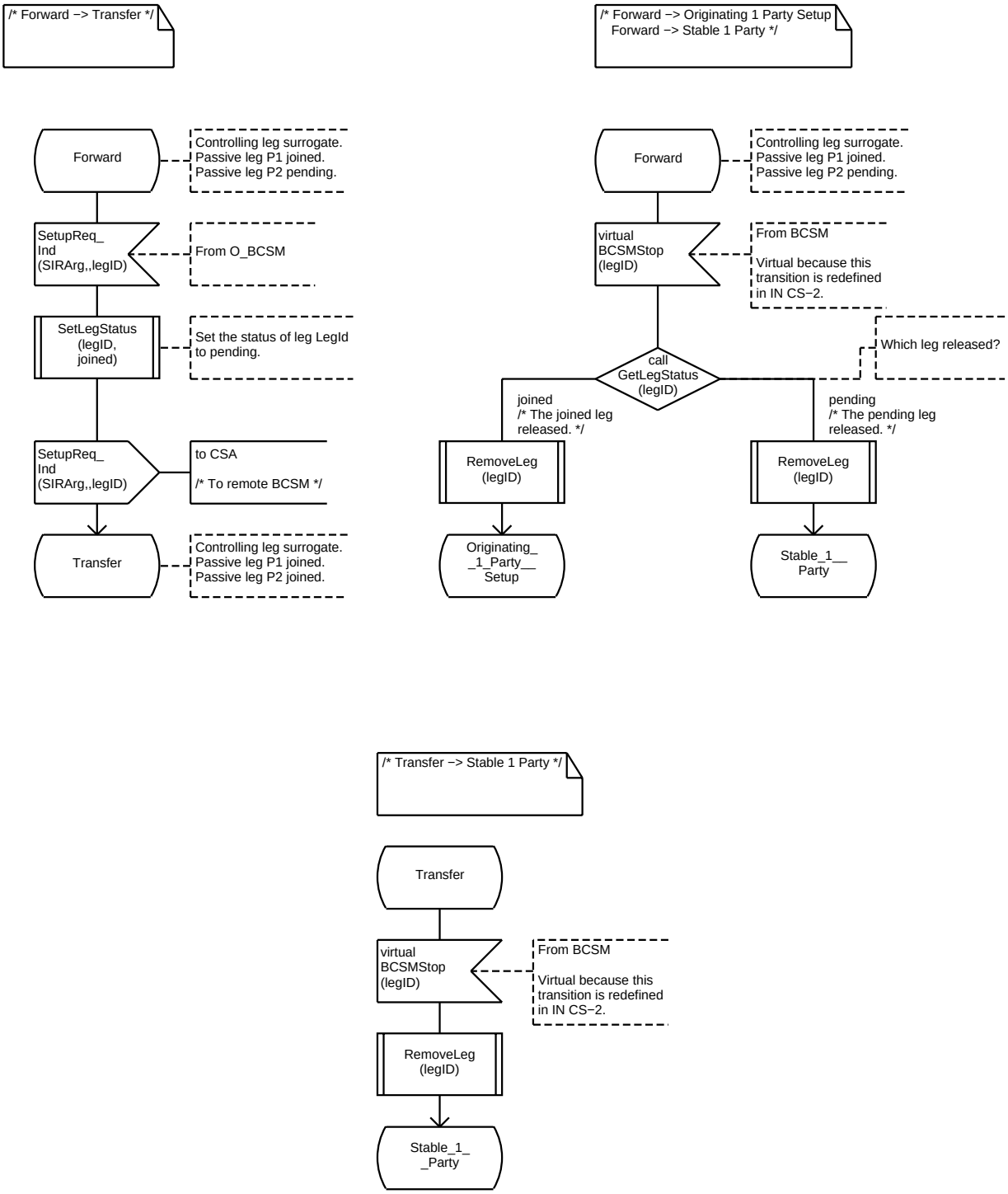
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```




;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
SSFState SSFStateType, /* Start state of the SSF−FSM. */
SCFInitiated Boolean; /* Is the call initiated by the SCF or not? */
/* Stable 1 Party −> Forward */
Stable_1_
_Party
Controlling leg surrogate.
Passive leg P1 joined
(associated with an O−BCSM).
Connnect
(invokeID,,CoArg)
From SCF.
call
AddLeg
(CoArg!legToBeCreated!
sendingSideID)
Add a passive leg
with id LegId.
Failed
Successful
obcsmPars := call
MapConnectToBCSM
(coArg)
Failed
obcsmPars!startState:=analyseInformation,
obcsmPars!legID := coArg!
legToBeCreated!sendingSideID
O_BCSM
(obcsmPars)
Create an instance of
an O−BCSM which starts
off from the SelectRoute
PIC with LegId = LegId.
SetLegPtr
(CoArg!legToBeCreated!
sendingSideID,
Offspring,
Originating)
Associate the passive leg
with the O−BCSM.
SetLegStatus
(CoArg!legToBeCreated!
sendingSideID,
pending)
Set the status of passive
leg to pending.
Connnect
(invokeID,,CoArg)
TO SSF
/* Send Connect to the
SSF associated with this segment. */
Forward
Controlling leg surrogate.
Passive leg P1 joined
Passive leg P2 pending

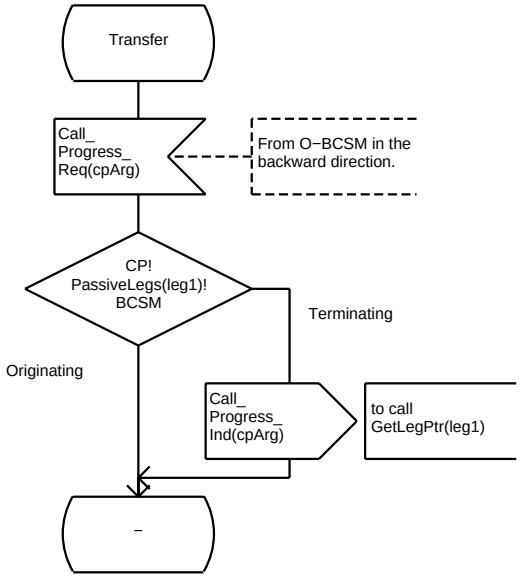

;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
SSFState SSFStateType, /* Start state of the SSF−FSM. */
SCFInitiated Boolean; /* Is the call initiated by the SCF or not? */
/* Stable 2 Party −> Originating Setup
Notes: − ReleaseReqInd (from remote party) in stable 2 party on an O/T−BCSM results in that the CS goes 'back' to originating/terminating setup state.
− The transition from Stable 2−Party to Originating or Terminating Setup on Release is not defined in the Q.1224 recommendation (IN CS−2). */
Stable_2_ _Party
Controlling leg joined. Passive leg P1 joined.
ReleaseReqInd (rArg,csaID,legID, callFlag)
From BCSM.
Is the signal incoming or outgoing?
call IsBCSM(Sender)
No /* Incoming */
Yes /* Outgoing */
callFlag
senderRef /* To T−BCSM */
receiverRef /* To O−BCSM. */
legID := call GetLegIdFrom_ RemoteLegId (legID)
Retrieve the legID.
SetLeg_ Status (legID,pending)
Set the status of the passive leg to pending.
ReleaseReqInd (rArg,csaID,legID, callFlag)
to CSA
/* To remote BCSM via CSA/IH. */
ReleaseReqInd (rArg,csaID,legID, callFlag)
to call GetLegPtr (legID)
ReleaseReqInd (rArg,csaID,legID, callFlag)
to call GetLegPtr (legID)
/* To O−BCSM. */
_
Remain in Stable−2−Party.
Terminating_ _Setup
Originating_ _Setup

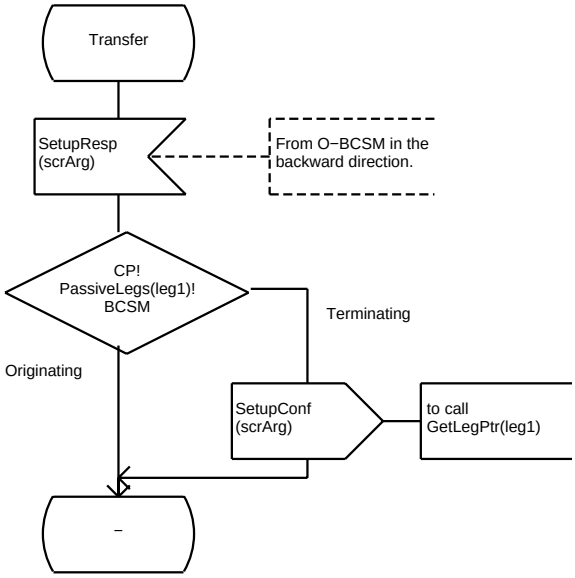
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

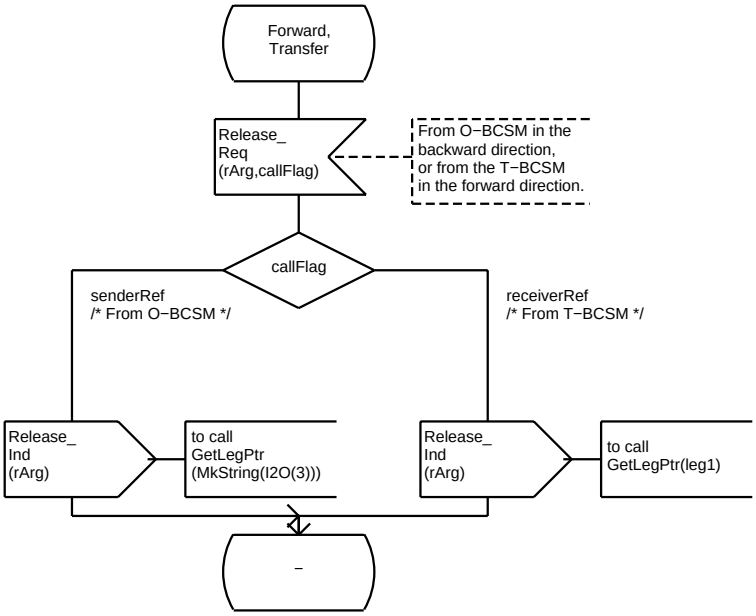


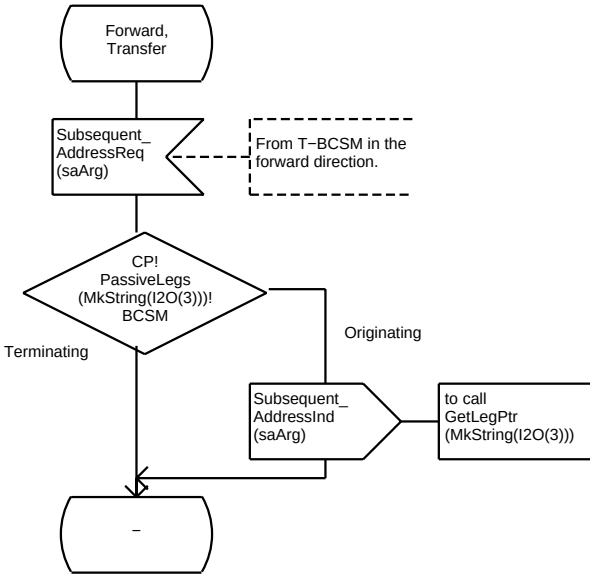
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

```
/* In the Transfer state, all backward and forward
signalling messages to/from the BCSMs must be relayed
since the controlling leg is surrogate.

In the Forward state, only ReleaseReq and Subsequent−
AddressReq needs to be relayed. */
```



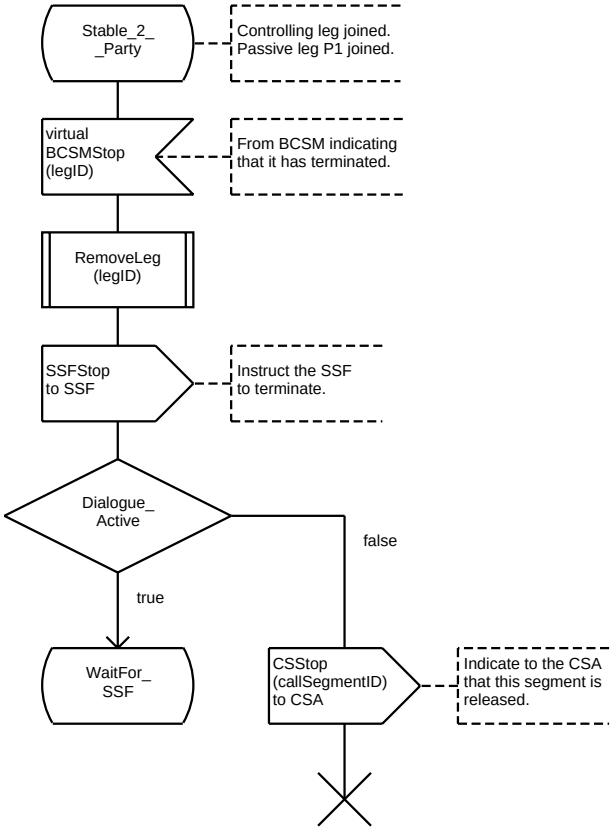


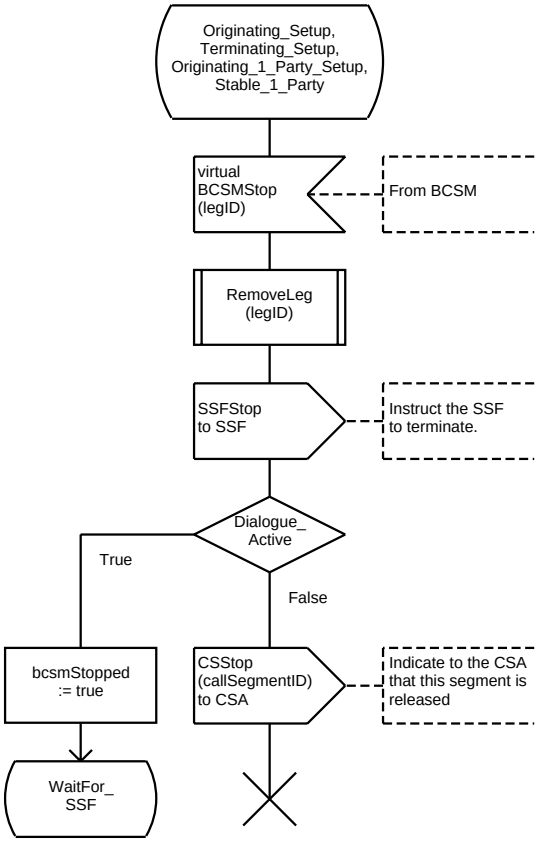




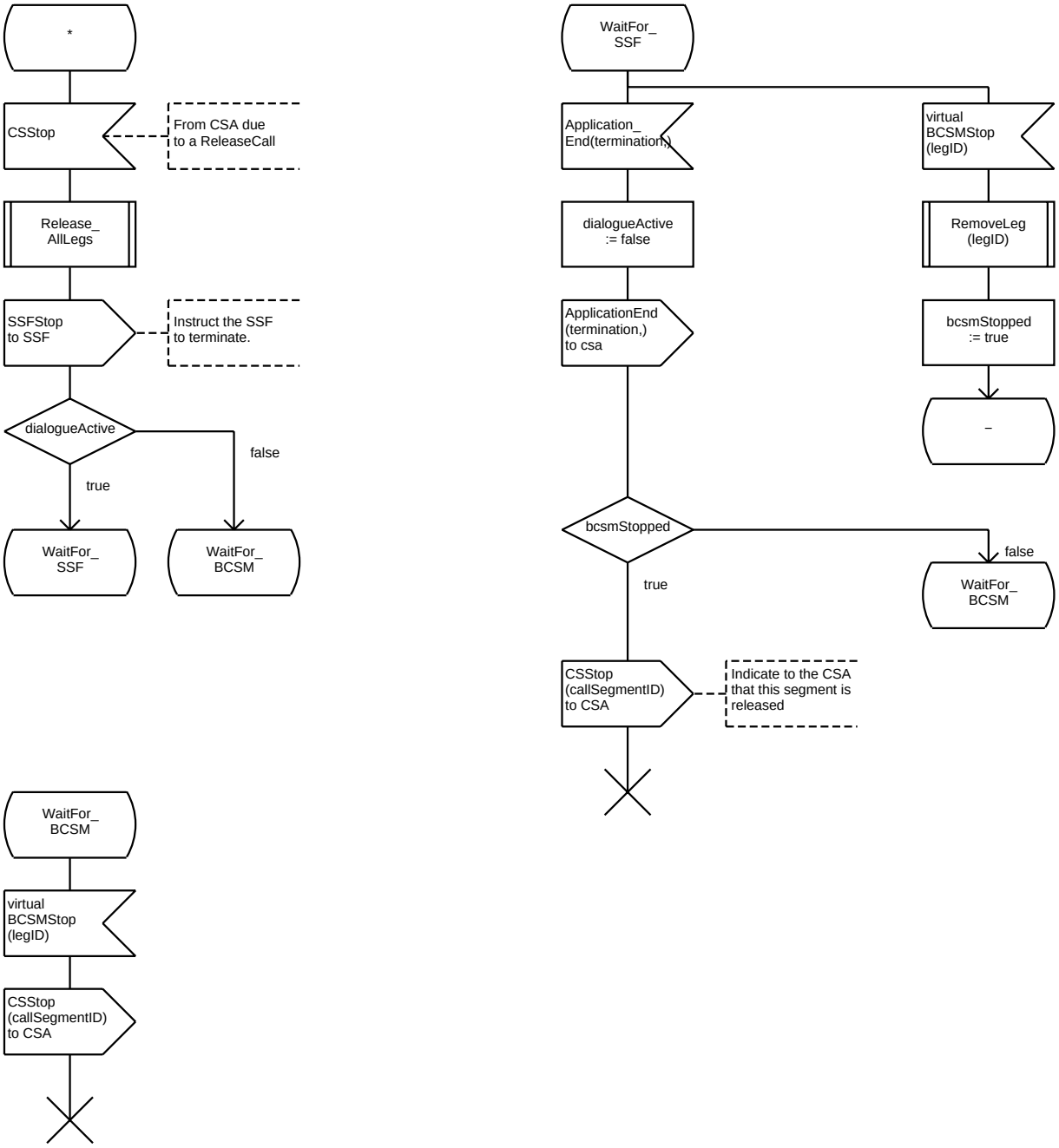
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

```
/**** HANDLING OF CALL RELEASE ****/
```

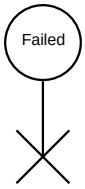


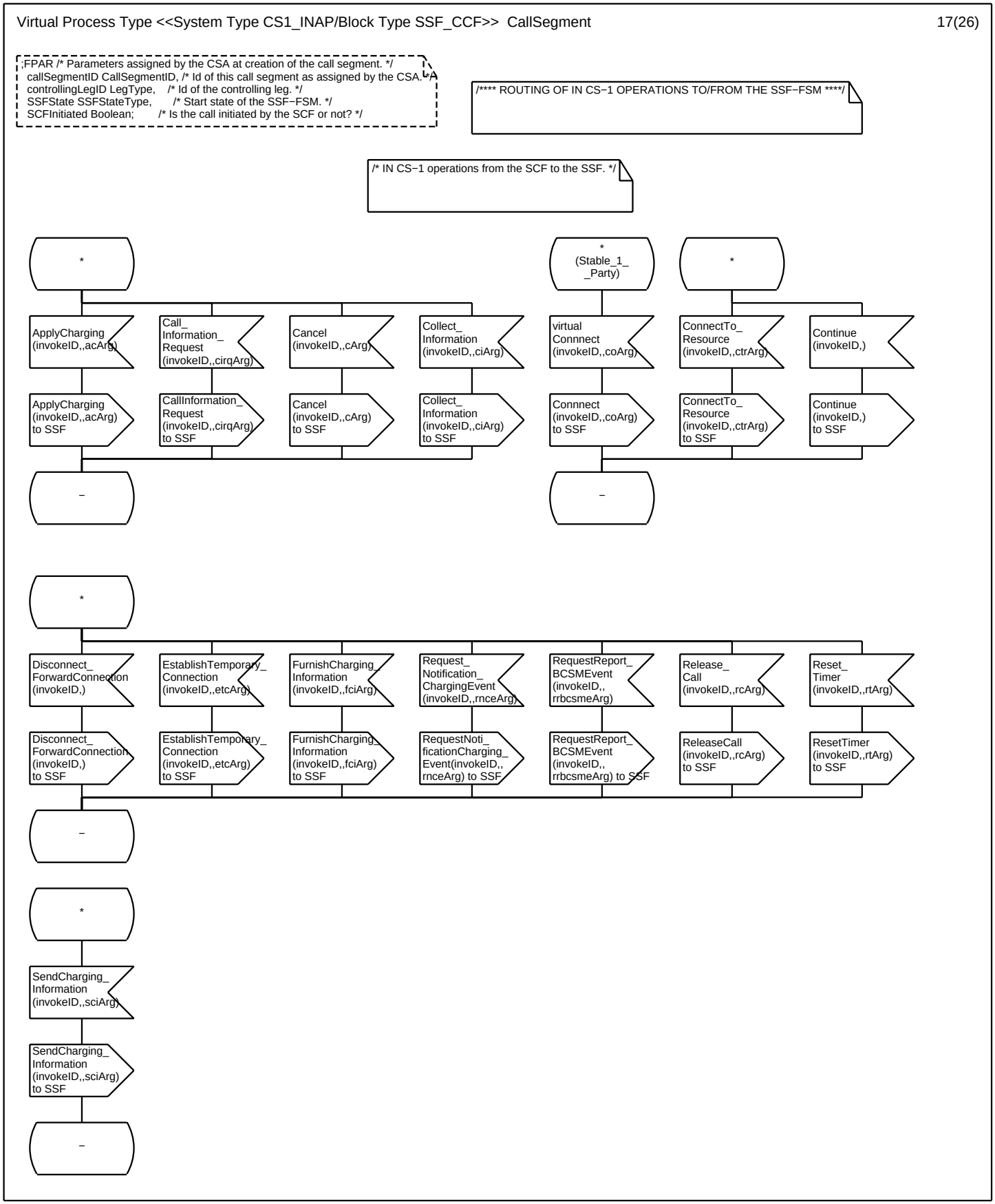


```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

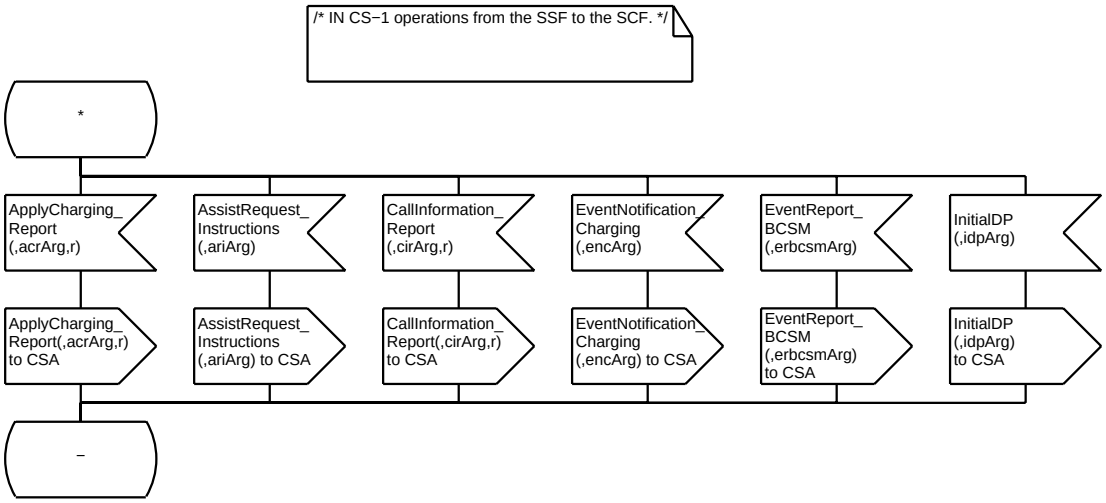


```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```



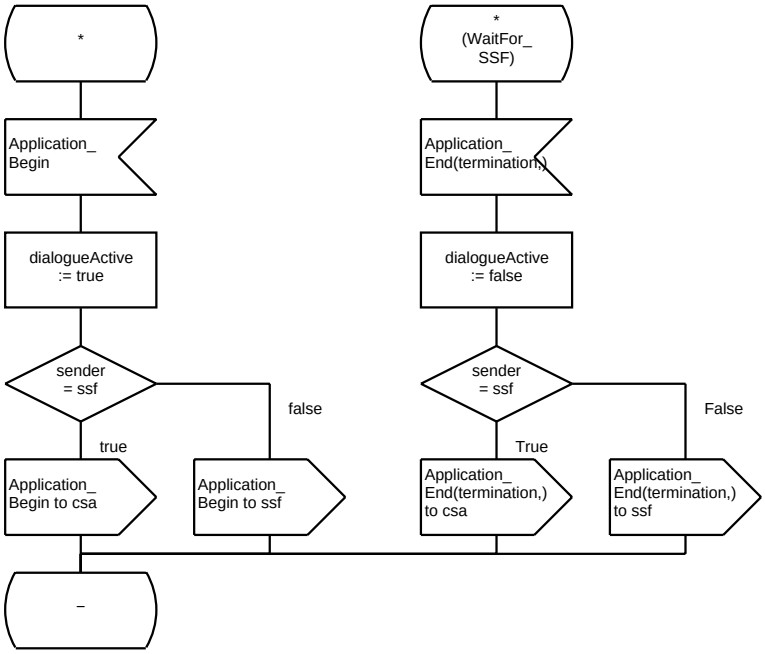

;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
SSFState SSFStateType, /* Start state of the SSF−FSM. */
SCFInitiated Boolean; /* Is the call initiated by the SCF or not? */
/**** ROUTING OF IN CS−1 OPERATIONS TO/FROM THE SSF−FSM ****/
/* IN CS−1 operations from the SCF to the SSF. */
*
ApplyCharging (invokeID,,acArg)
Call_ Information_ Request (invokeID,,cirqArg)
Cancel (invokeID,,cArg)
Collect_ Information (invokeID,,ciArg)
ApplyCharging (invokeID,,acArg) to SSF
CallInformation_ Request (invokeID,,cirqArg) to SSF
Cancel (invokeID,,cArg) to SSF
Collect_ Information (invokeID,,ciArg) to SSF
*
*
(Stable_1_ _Party)
virtual Connnect (invokeID,,coArg)
ConnectTo_ Resource (invokeID,,ctrArg)
Continue (invokeID,)
Connnect (invokeID,,coArg) to SSF
ConnectTo_ Resource (invokeID,,ctrArg) to SSF
Continue (invokeID,) to SSF
*
Disconnect_ ForwardConnection (invokeID,)
EstablishTemporary_ Connection (invokeID,,etcArg)
FurnishCharging_ Information (invokeID,,fciArg)
Request_ Notification_ ChargingEvent (invokeID,,rnceArg)
RequestReport_ BCSMEvent (invokeID,, rrbcsmeArg)
Release_ Call (invokeID,,rcArg)
Reset_ Timer (invokeID,,rtArg)
Disconnect_ ForwardConnection (invokeID,) to SSF
EstablishTemporary_ Connection (invokeID,,etcArg) to SSF
FurnishCharging_ Information (invokeID,,fciArg) to SSF
RequestNoti_ ficationCharging_ Event(invokeID,, rnceArg) to SSF
RequestReport_ BCSMEvent (invokeID,, rrbcsmeArg) to SSF
ReleaseCall (invokeID,,rcArg) to SSF
ResetTimer (invokeID,,rtArg) to SSF
*
SendCharging_ Information (invokeID,,sciArg)
SendCharging_ Information (invokeID,,sciArg) to SSF

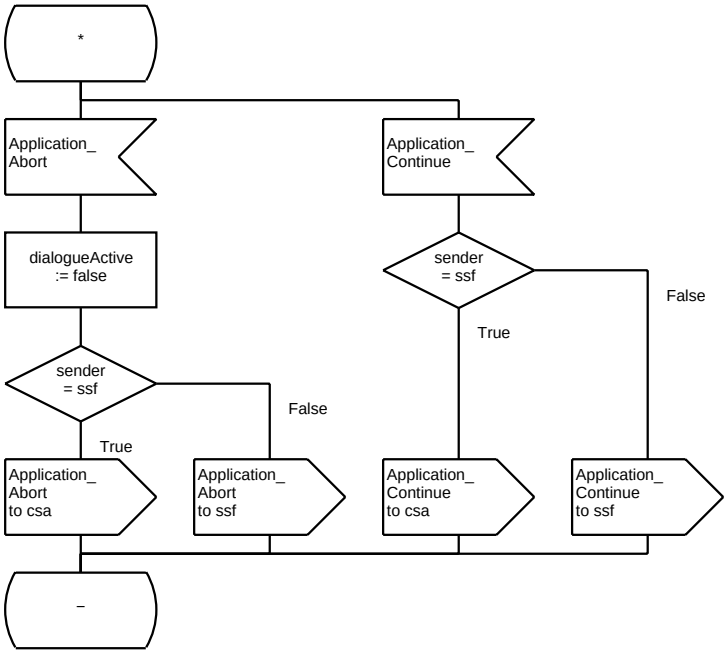
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

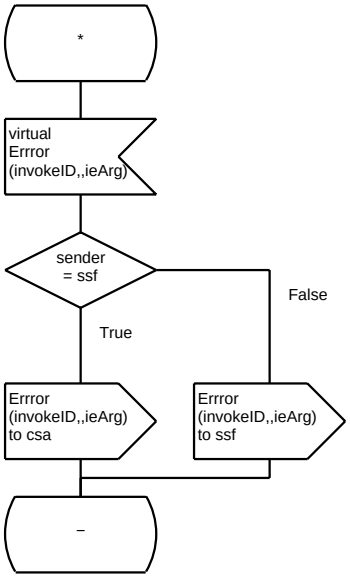


```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

```
/**** ROUTING OF DIALOUGE AND ERROR PRIMITIVES. ****/
```

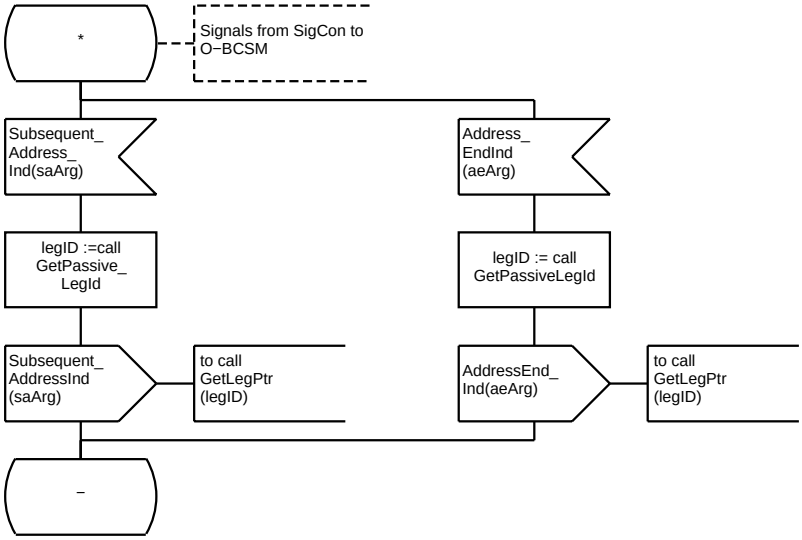


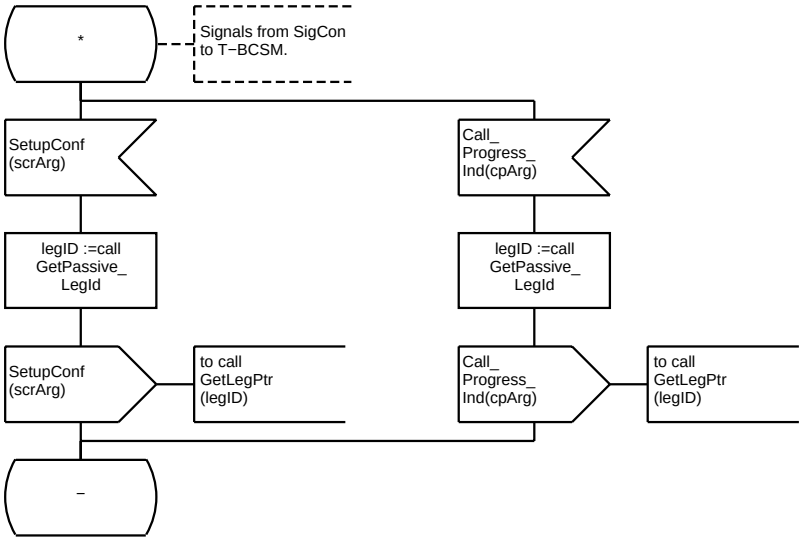




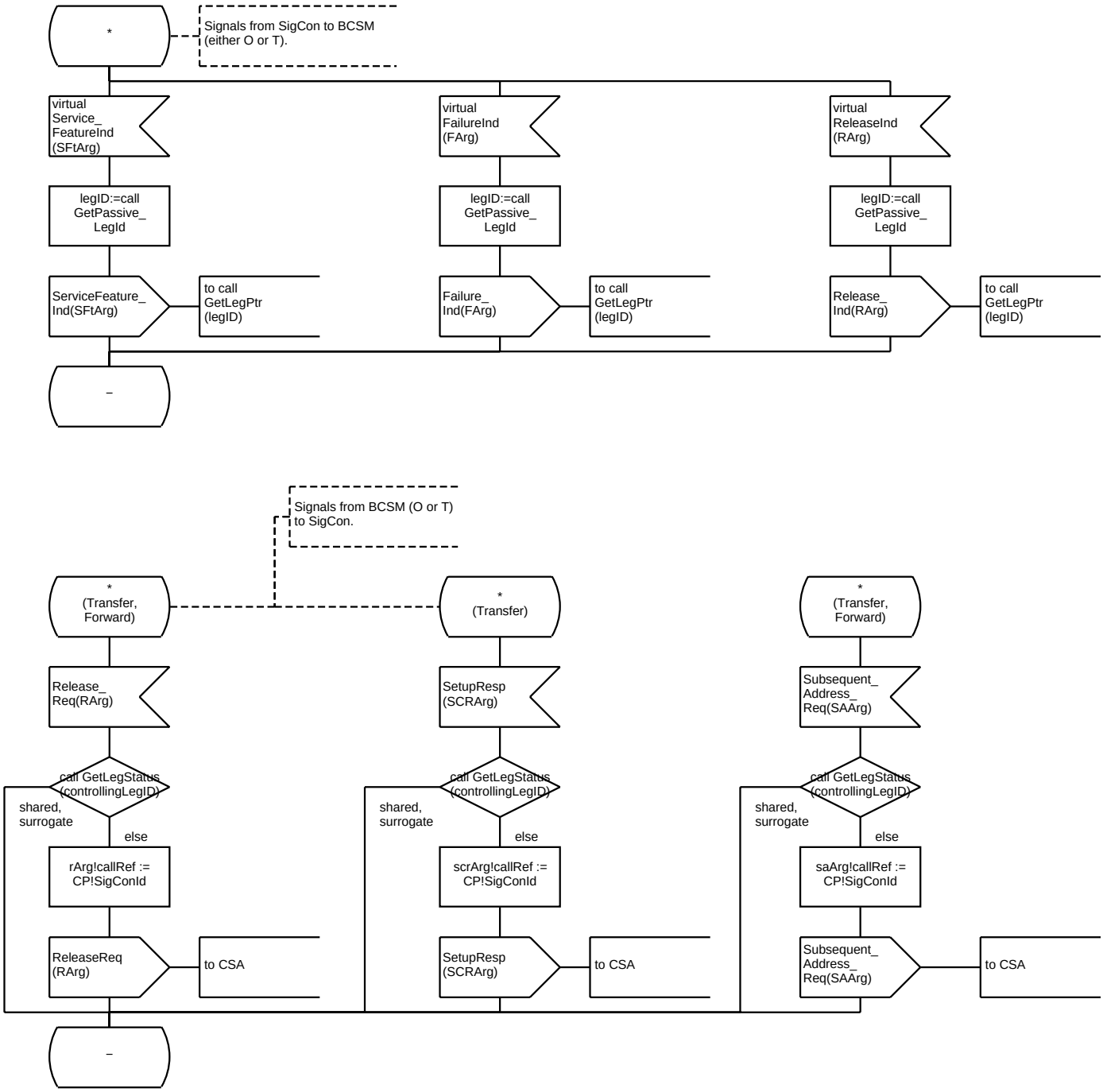
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

```
/* ROUTING OF SIGNALS TO/FROM SIGCON (VIA THE CSA/IH). */
```

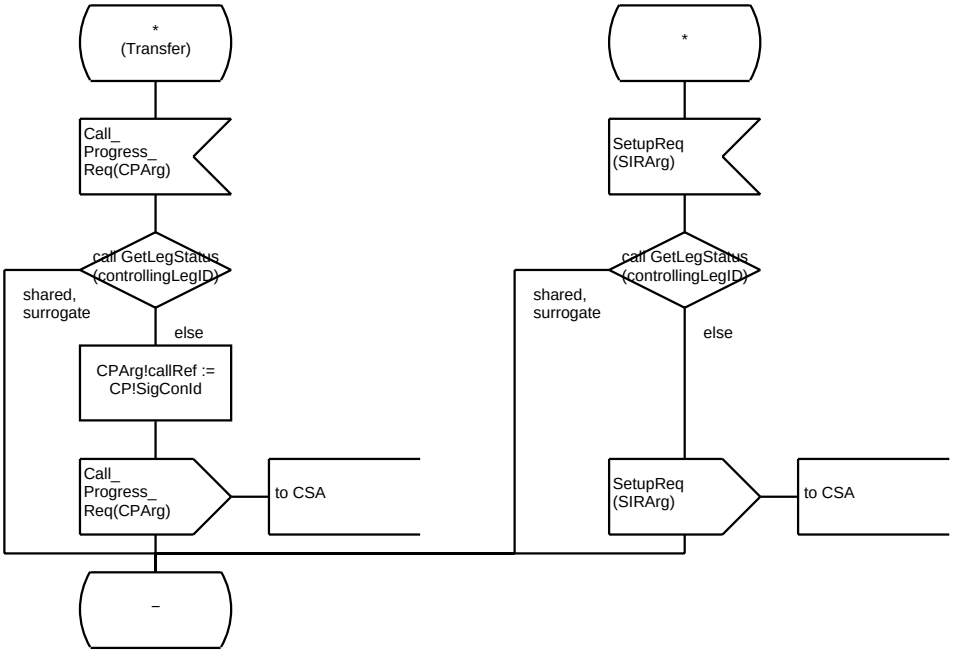


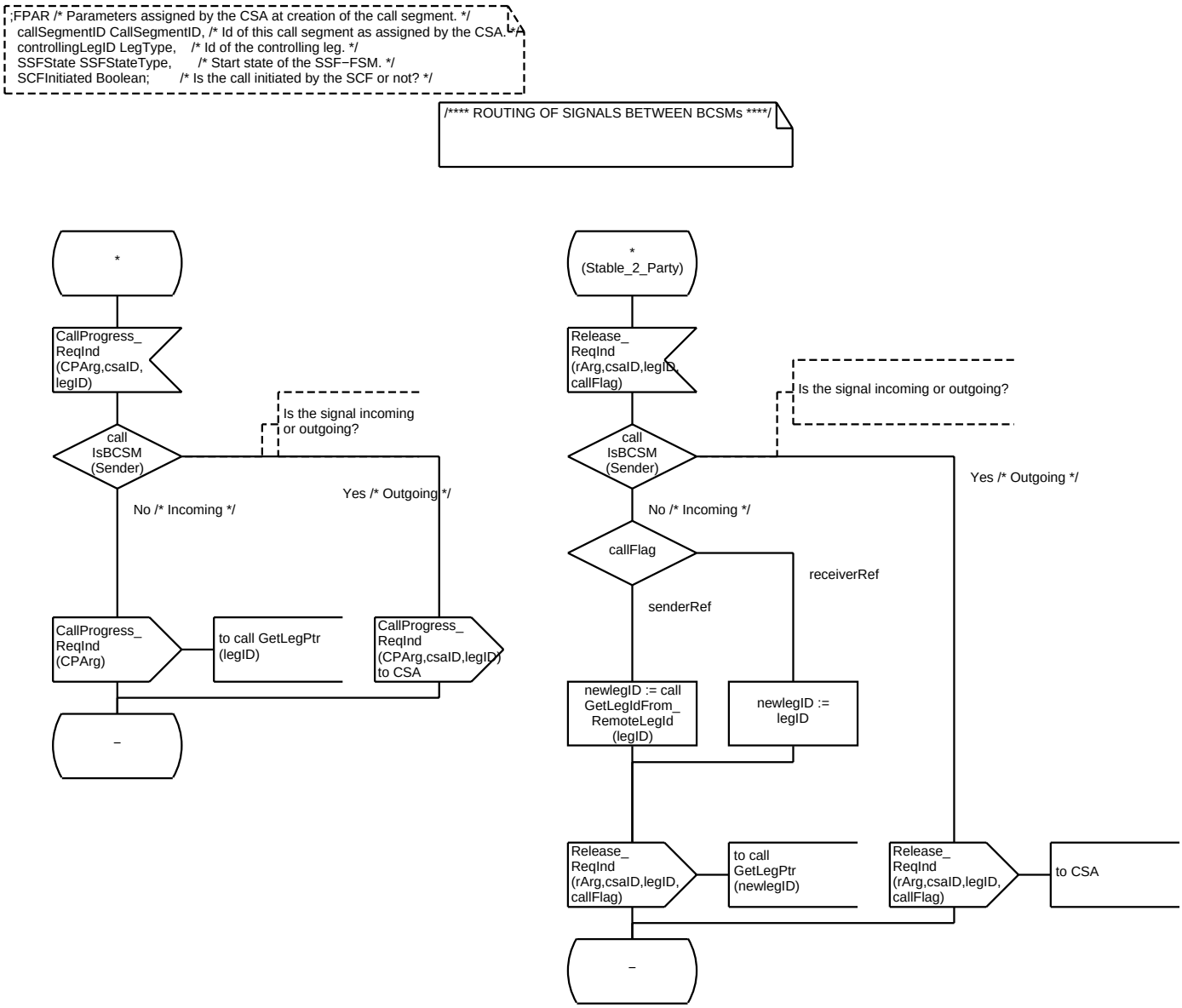


```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

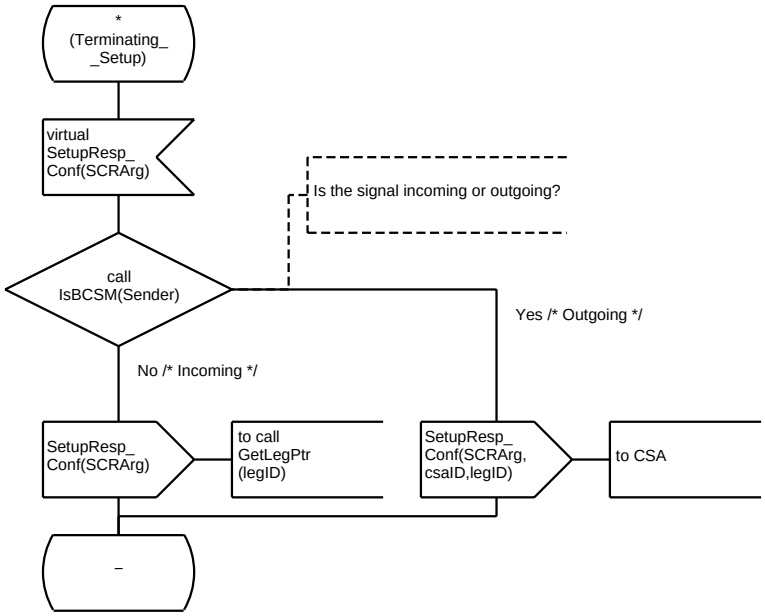


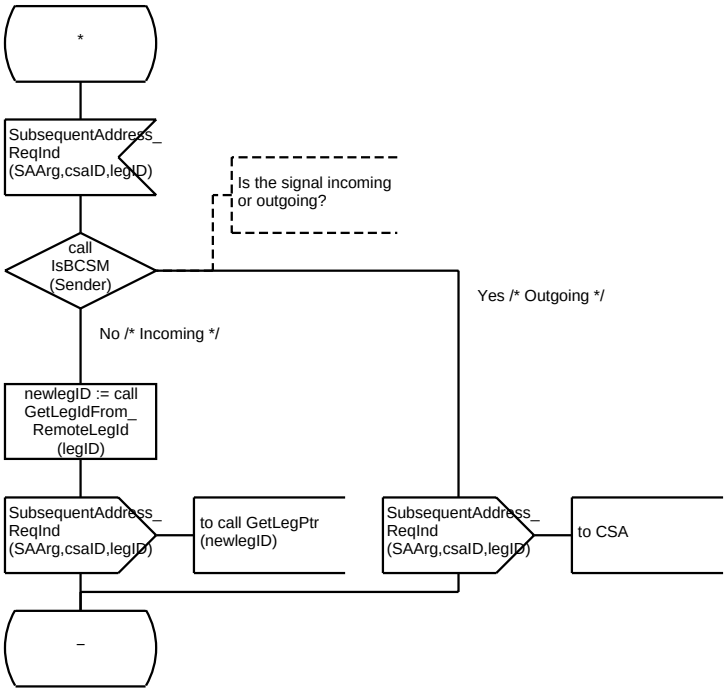
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

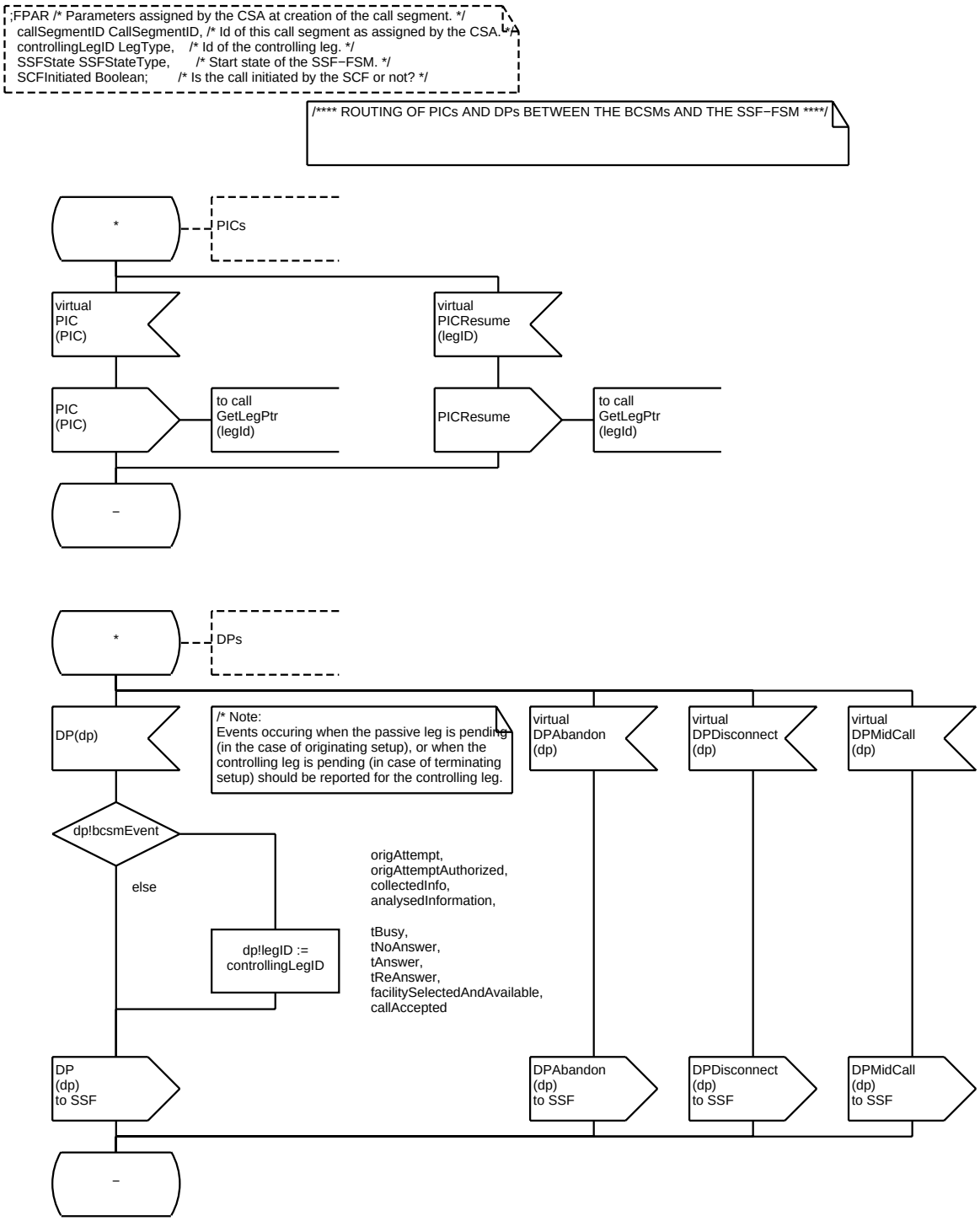



;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
SSFState SSFStateType, /* Start state of the SSF−FSM. */
SCFInitiated Boolean; /* Is the call initiated by the SCF or not? */
/**** ROUTING OF SIGNALS BETWEEN BCSMs ****/
*
CallProgress_ ReqInd (CPArg,csaID, legID)
call IsBCSM (Sender)
Is the signal incoming or outgoing?
No /* Incoming */
Yes /* Outgoing */
CallProgress_ ReqInd (CPArg)
to call GetLegPtr (legID)
CallProgress_ ReqInd (CPArg,csaID,legID) to CSA
–
*
(Stable_2_Party)
Release_ ReqInd (rArg,csaID,legID, callFlag)
Is the signal incoming or outgoing?
call IsBCSM (Sender)
Yes /* Outgoing */
No /* Incoming */
callFlag
receiverRef
senderRef
newlegID := call GetLegIdFrom_ RemoteLegId (legID)
newlegID := legID
Release_ ReqInd (rArg,csaID,legID, callFlag)
to call GetLegPtr (newlegID)
Release_ ReqInd (rArg,csaID,legID, callFlag)
to CSA
–

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

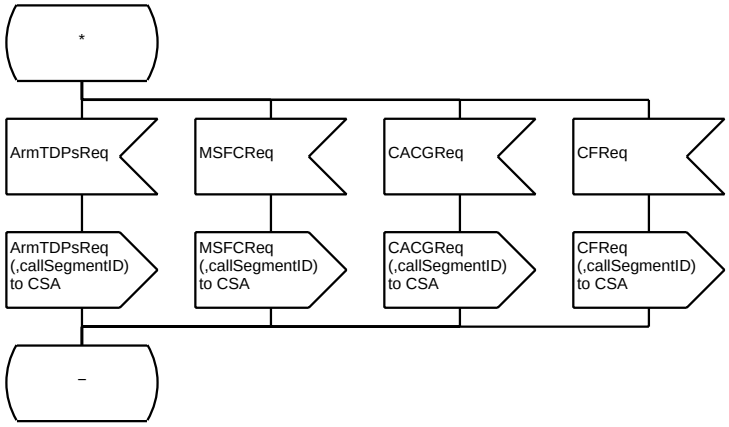


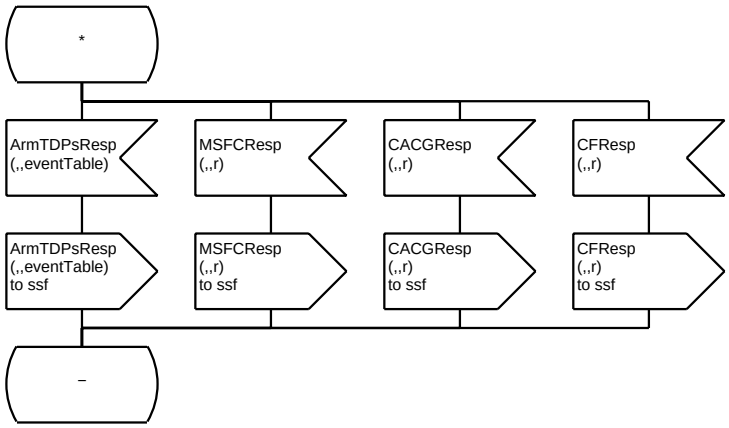



;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
SSFState SSFStateType, /* Start state of the SSF−FSM. */
SCFInitiated Boolean; /* Is the call initiated by the SCF or not? */
/**** ROUTING OF PICs AND DPs BETWEEN THE BCSMs AND THE SSF−FSM ****/
*
PICs
virtual PIC (PIC)
virtual PICResume (legID)
PIC (PIC)
to call GetLegPtr (legId)
PICResume
to call GetLegPtr (legId)
−
*
DPs
DP(dp)
/* Note: Events occuring when the passive leg is pending (in the case of originating setup), or when the controlling leg is pending (in case of terminating setup) should be reported for the controlling leg.
virtual DPAbandon (dp)
virtual DPDisconnect (dp)
virtual DPMidCall (dp)
dp!bcsmEvent
else
dp!legID := controllingLegID
origAttempt,
origAttemptAuthorized,
collectedInfo,
analysedInformation,
tBusy,
tNoAnswer,
tAnswer,
tReAnswer,
facilitySelectedAndAvailable,
callAccepted
DP (dp) to SSF
DPAbandon (dp) to SSF
DPDisconnect (dp) to SSF
DPMidCall (dp) to SSF
−

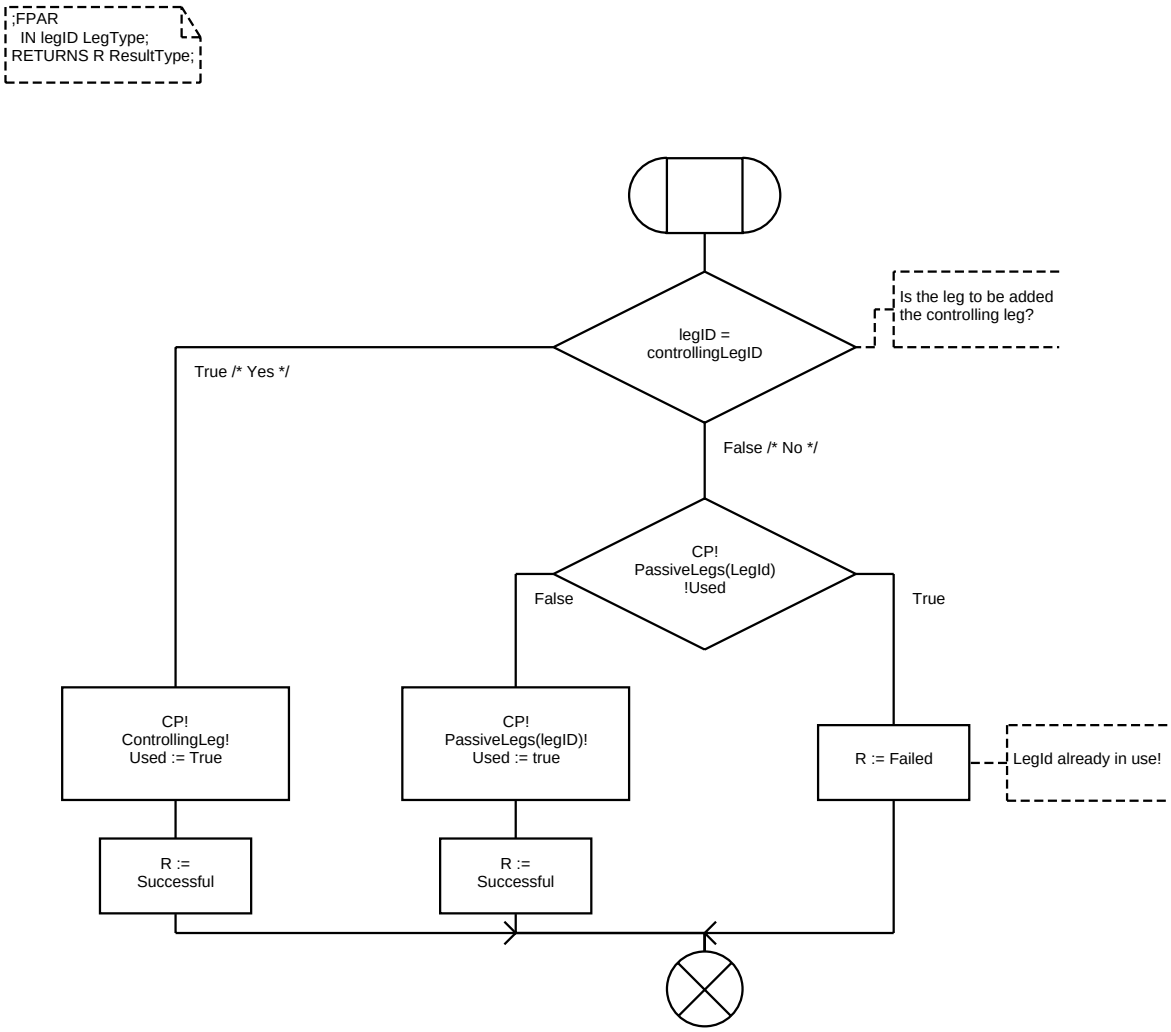
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  SSFState SSFStateType,       /* Start state of the SSF−FSM. */
  SCFInitiated Boolean;        /* Is the call initiated by the SCF or not? */
```

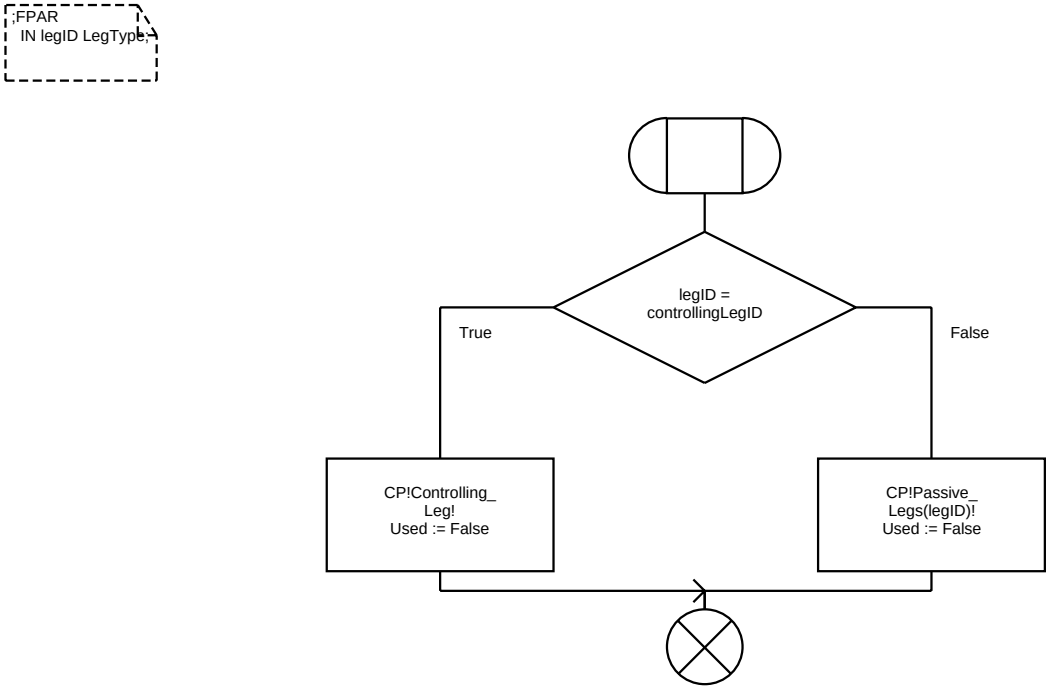
/* ROUTING OF SIGNALS BETWEEN THE SSF−FSM AND THE SSME−FSM */





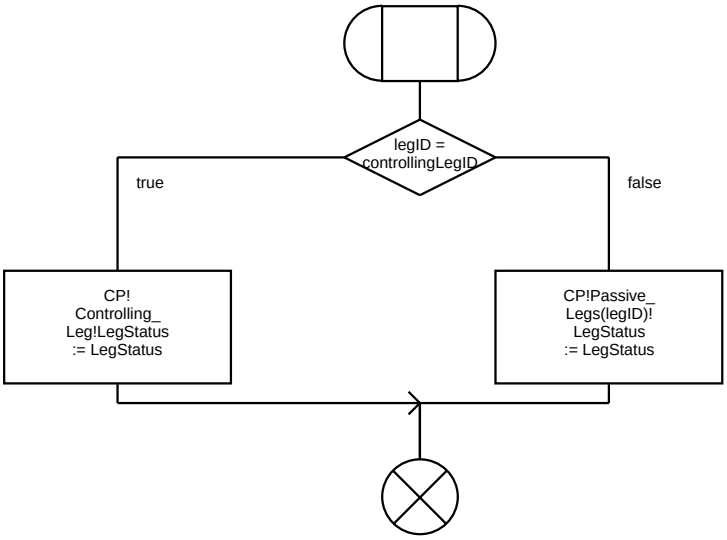
Procedure AddLeg

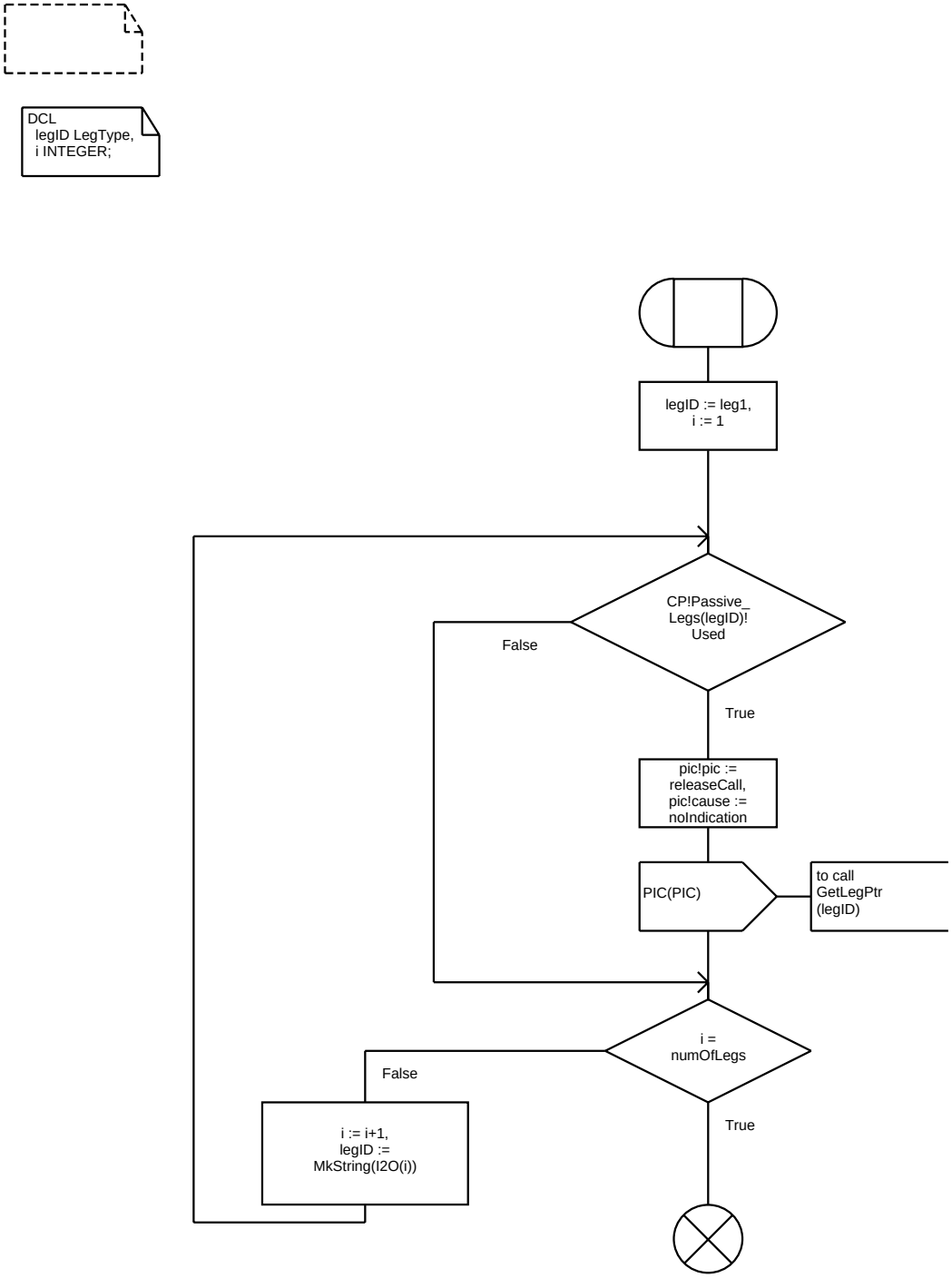



;FPAR
IN legID LegType;
legID =
controllingLegID
True
False
CP!Controlling_
Leg!
Used := False
CP!Passive_
Legs(legID)!
Used := False

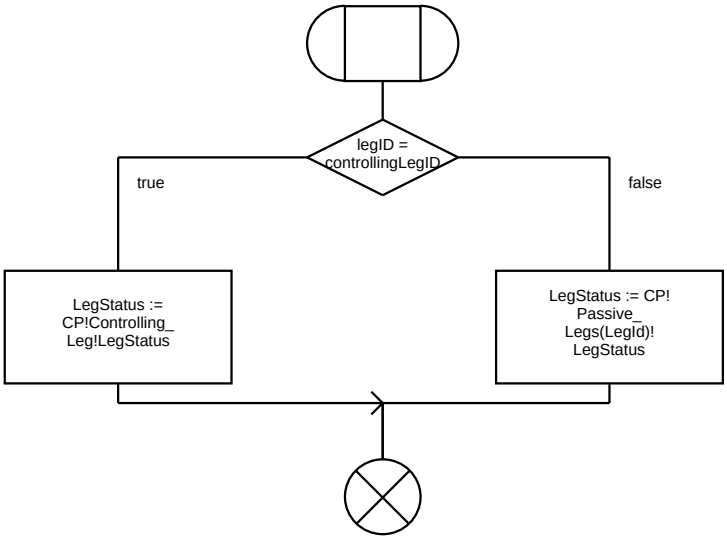
Procedure SetLegStatus


```
;FPAR
  IN legID LegType,
  IN legStatus LegStatusType;
```

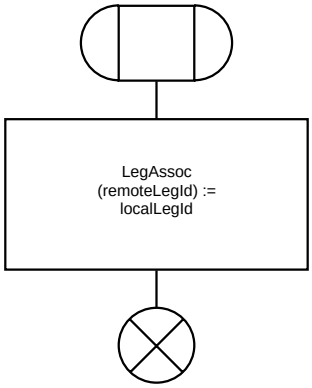



DCL
legID LegType,
i INTEGER;
legID := leg1,
i := 1
CP!Passive_
Legs(legID)!
Used
False
True
pic!pic :=
releaseCall,
pic!cause :=
noIndication
PIC(PIC)
to call
GetLegPtr
(legID)
i =
numOfLegs
False
True
i := i+1,
legID :=
MkString(I2O(i))

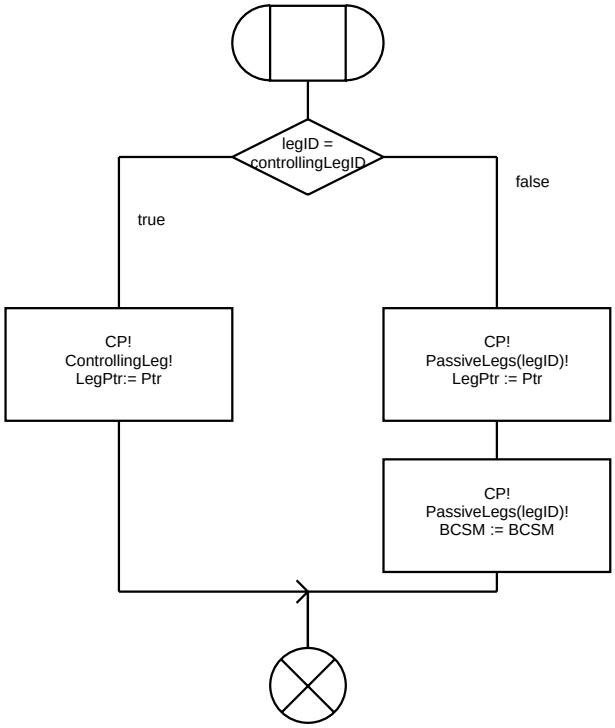
```
;FPAR
  IN legID LegType;
RETURNS LegStatus LegStatusType;
```



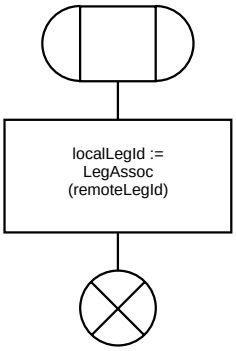
```
;FPAR
  IN remoteLegId LegType,
  IN localLegId LegType;
```



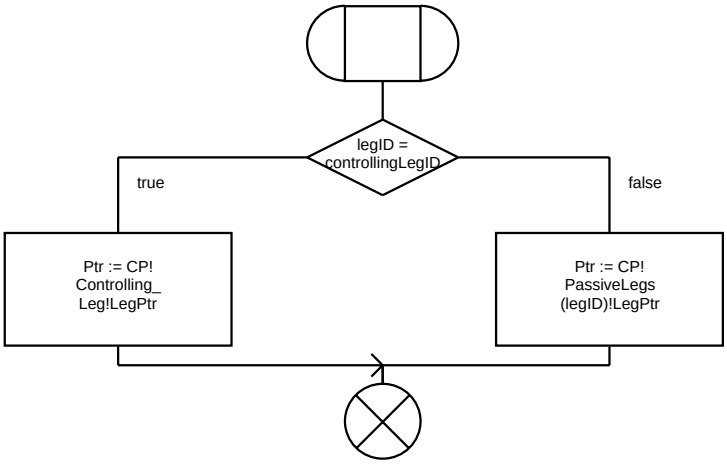
```
;FPAR
  IN legID LegType,
  IN Ptr Pid,
  IN BCSM BCSMType;
```



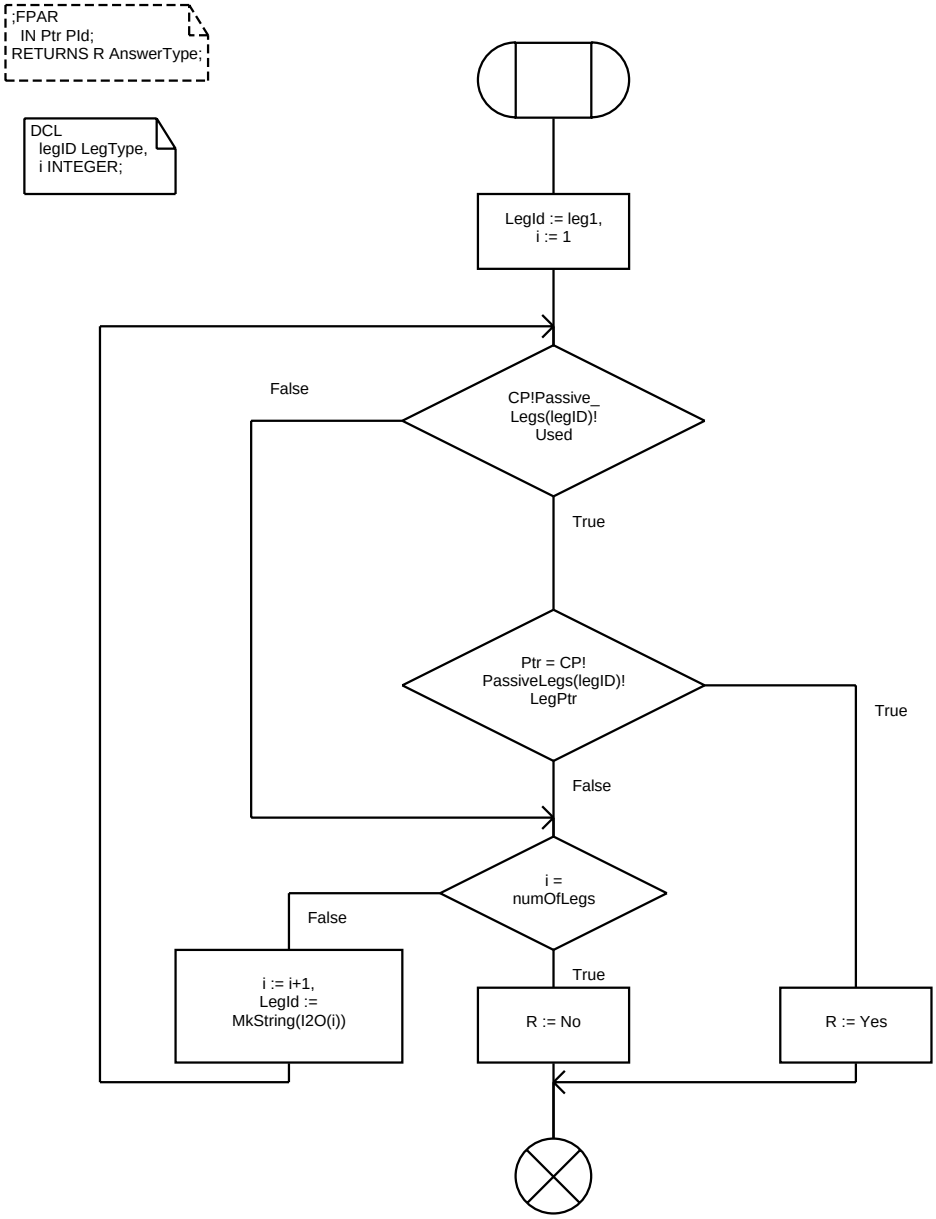
```
;FPAR
  IN remoteLegId LegType;
RETURNS localLegId LegType;
```

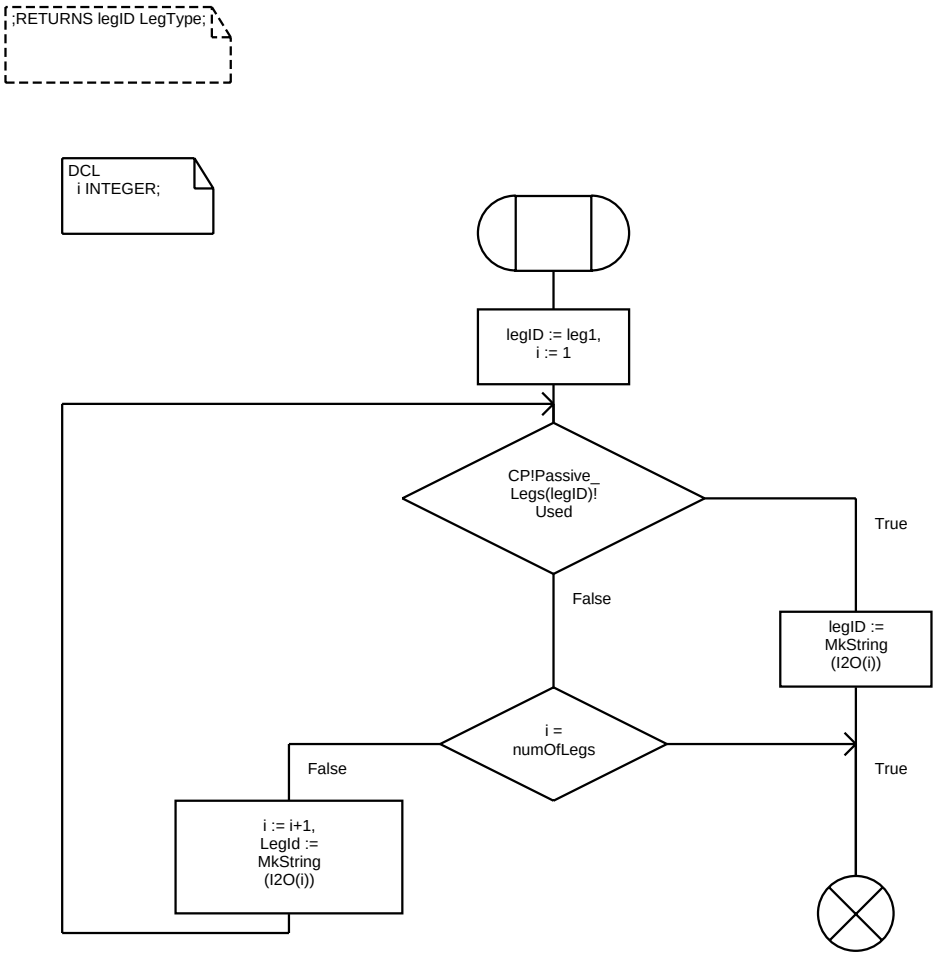


```
;FPAR
  IN legID LegType;
RETURNS Ptr PId;
```

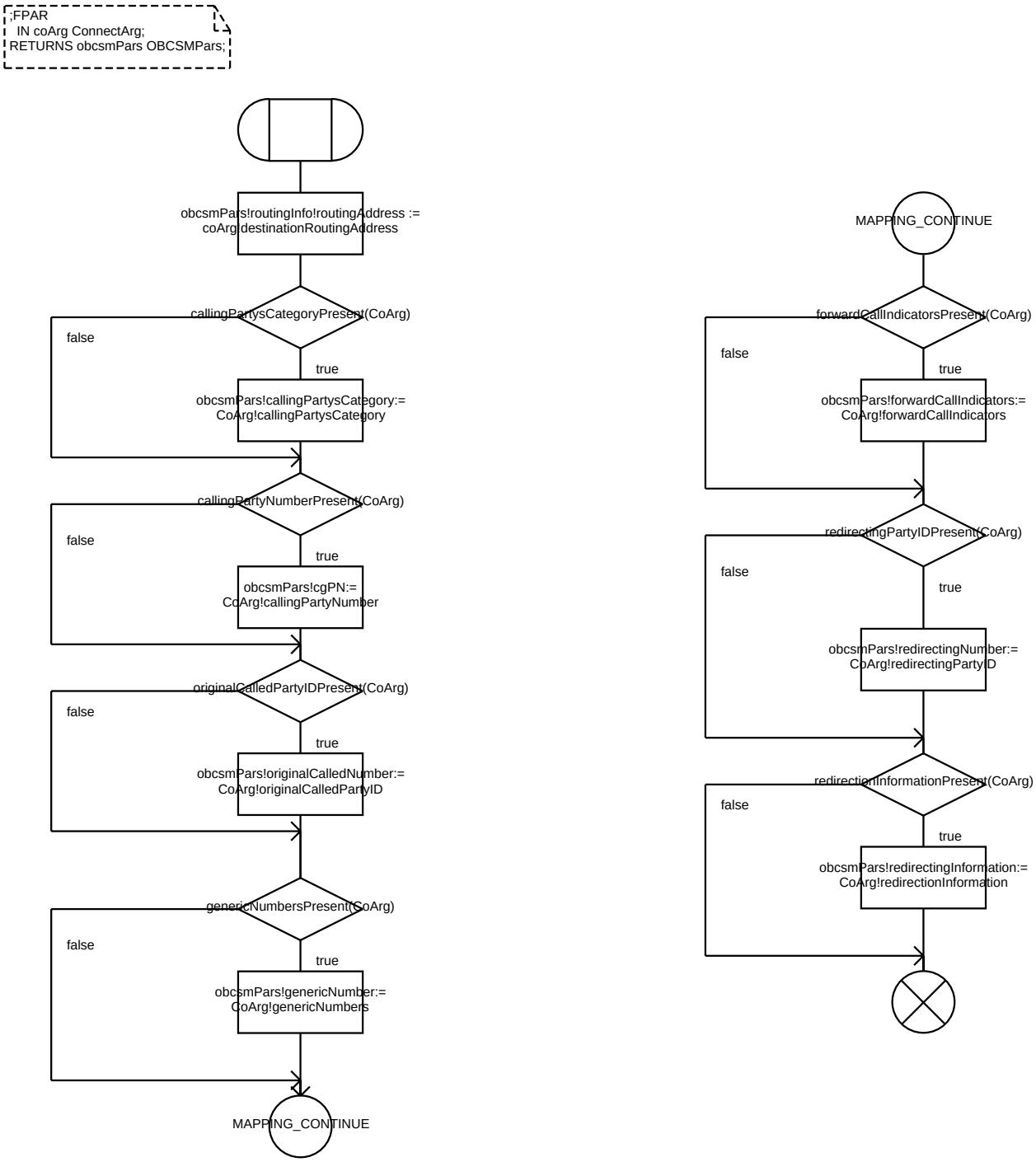


Procedure IsBCSM



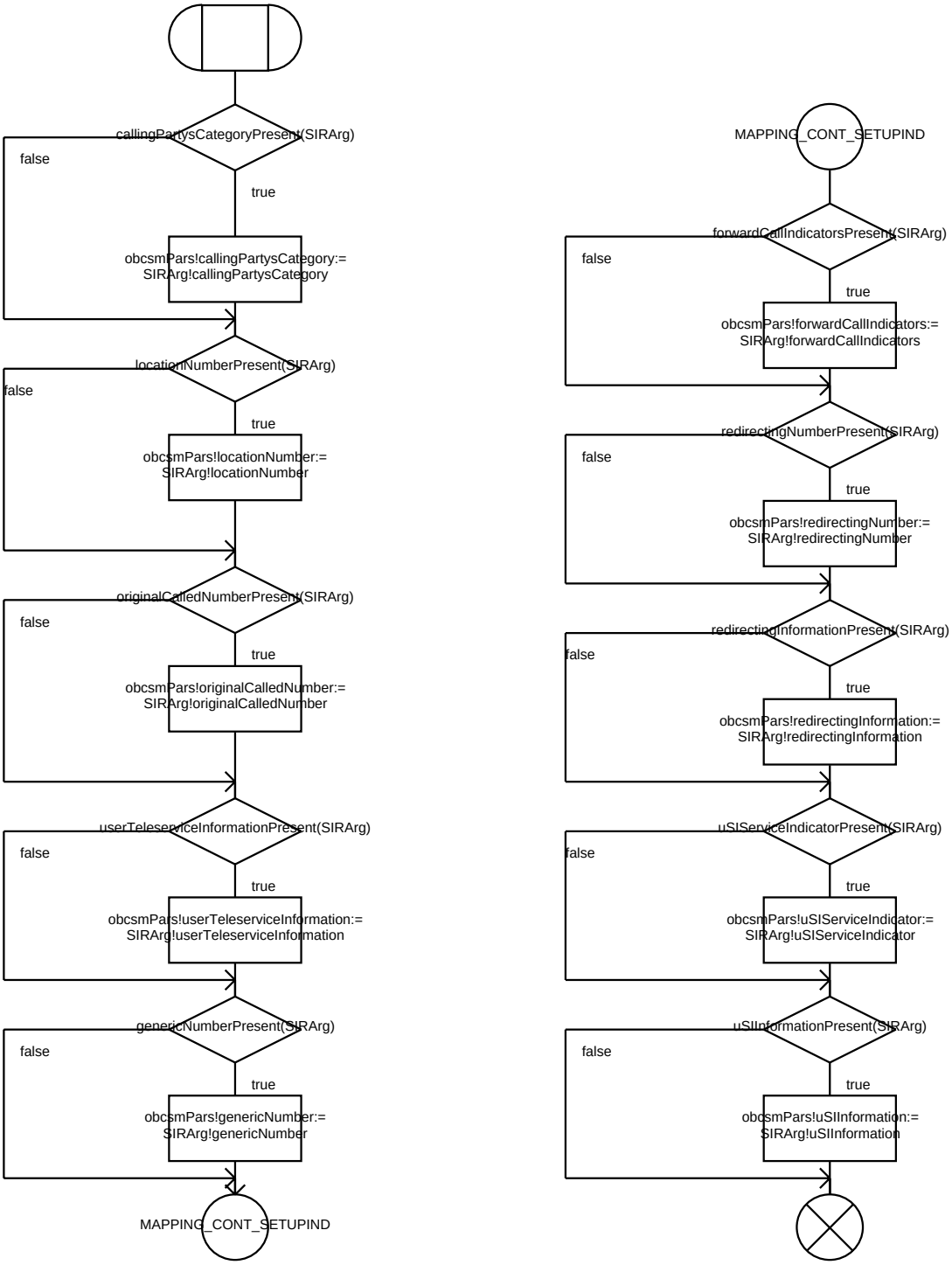

;RETURNS legID LegType;
DCL
i INTEGER;
legID := leg1,
i := 1
CP!Passive_
Legs(legID)!
Used
True
False
legID :=
MkString
(I2O(i))
i =
numOfLegs
False
True
i := i+1,
LegId :=
MkString
(I2O(i))

Procedure MapConnectToBCSM



Procedure MapSIToBCSM


```
;FPAR
  IN sirArg SetupIRType;
RETURNS obcsmPars OBCSMPars;
```



```
;FPAR
  StartState SSFStateType,
  SCFInitiated Boolean;
```

```
/**** DATA TYPE DEFINITIONS ****/
```

CS

(CS1_SSF_In)

(CS1_SSF_Out)

```
/* The SSF−FSM maintains information about the leg IDs
managed by this SSF−FSM. */

NEWTYPE LegIdTableType
  ARRAY(LegType, Boolean)
ENDNEWTYPE;

NEWTYPE DialogPrimitive
 LITERALS
  Begin,
  Continue,
  End,
  Abort;
ENDNEWTYPE;
```

```
;FPAR
  StartState SSFStateType,
  SCFInitiated Boolean;
```

```
/**** VARIABLE DECLARATIONS ****/
```

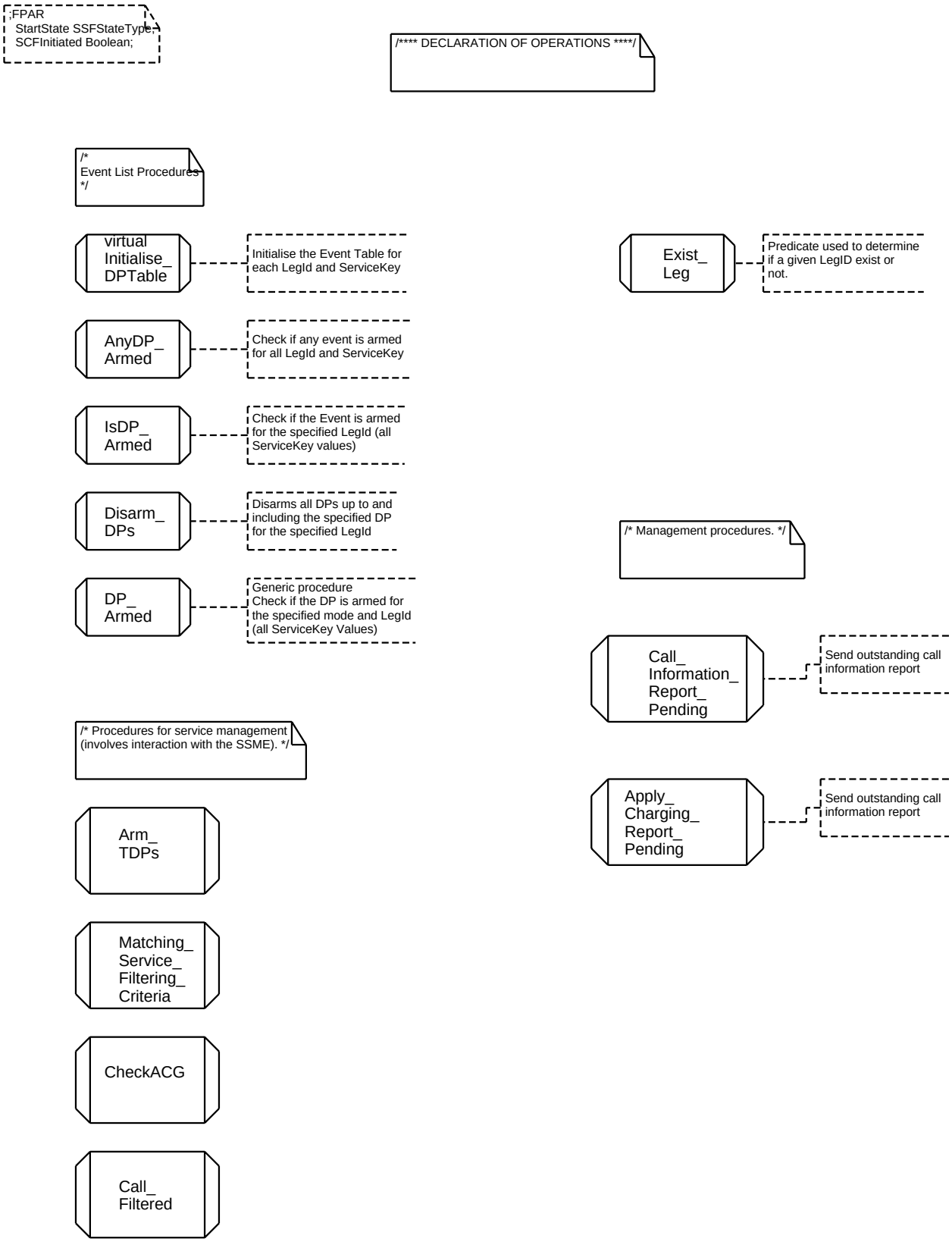
```
DCL
  /* IN CS−1 operation arguments. */
  acArg ApplyChargingArg,
  acrArg ApplyChargingReportArg,
  ariArg AssistRequestInstructionsArg,
  aiArg AnalyseInformationArg,
  cirArg CallInformationReportArg,
  cirqArg CallInformationRequestArg,
  cArg CancelArg,
  ciArg CollectInformationArg,
  coArg ConnectArg,
  ctrArg ConnectToResourceArg,
  ctArg ContinueWithArgumentArg,
  dfcArg DisconnectForwardConnectionWithArgumentArg,
  etcArg EstablishTemporaryConnectionArg,
  encArg EventNotificationChargingArg,
  erBCSMArg EventReportBCSMArg,
  fciArg FurnishChargingInformationArg,
  hcinArg HoldCallInNetworkArg,
  idpArg InitialDPArg,
  icaArg InitiateCallAttemptArg,
  rcArg ReleaseCallArg,
  rnceARg RequestNotificationChargingEventArg,
  rrBCSMEArg RequestReportBCSMEventArg,
  rtARg ResetTimerArg,
  sciArg SendChargingInformationArg,
  sfArg SelectFacilityArg,
  srArg SelectRouteArg,
  ieArg ErrorArg;
```

```
DCL
  eventTable EventTableType,
  applicationActive Boolean := false,
  applyChargingReportPending Boolean := false,
  callInformationReportPending Boolean := false,
  eventNotificationChargingPending Boolean := false,
  dp DPArg,
  invokeID InvokeID,
  termination Boolean,
  flagContinue Boolean,
  pic PICArg,
  cs PId,
  lastPrimitive DialogPrimitive,
  sfEncountered Boolean := false,

  userInteractionActive Boolean := false,
  temporaryConnectionActive Boolean := false,

  suspendedLegID LegType := leg2, /* Used with Continue */

  /* Signal Parameter Declarations */
  legIdTable LegIdTableType,
  serviceKey ServiceKey;
```

```
;FPAR
  StartState SSFStateType,
  SCFInitiated Boolean;
```

```
/**** TIMER DECLARATIONS ****/
```

```
/*
The timer Tssf is used to prevent excessive
call supsension time and to guard the
association between the SSF and the SCF.
*/

TIMER     Tssf;

/*
The timer Tssf can have four different values

Tssf_Duration1 − Sending an InitialDP
Tssf_Duration2 − Entering state Waiting_for_Instruction
Tssf_Duration3 − Receiving an HoldCallInNetwork
Tssf_Duration4 − Entering Waiting_for_End_of_User_Interaction or
                 Waiting_for_End_of_Temporary_Connection
*/

/* The Tssf durations have been initialised to nominal values (1hr) */
DCL
Tssf_Duration1     DURATION := 3600000,
Tssf_Duration2     DURATION := 3600000,
Tssf_Duration3     DURATION := 3600000,
Tssf_Duration4  DURATION := 3600000,

Tssf_Duration DURATION;

/* The following flags are used to control the operation */
/* of the timer Tssf in Waiting_for_Instruction state */

DCL
InitialTssfActive BOOLEAN := FALSE,
InitialTssfReset BOOLEAN := FALSE;
```


;FPAR
StartState SSFStateType;
SCFInitiated Boolean;
/**** DECLARATION OF OPERATIONS ****/
/*
Event List Procedures
*/
virtual Initialise_ DPTable
Initialise the Event Table for each LegId and ServiceKey
AnyDP_ Armed
Check if any event is armed for all LegId and ServiceKey
IsDP_ Armed
Check if the Event is armed for the specified LegId (all ServiceKey values)
Disarm_ DPs
Disarms all DPs up to and including the specified DP for the specified LegId
DP_ Armed
Generic procedure
Check if the DP is armed for the specified mode and LegId (all ServiceKey Values)
/* Procedures for service management (involves interaction with the SSME). */
Arm_ TDPs
Matching_ Service_ Filtering_ Criteria
CheckACG
Call_ Filtered
Exist_ Leg
Predicate used to determine if a given LegID exist or not.
/* Management procedures. */
Call_ Information_ Report_ Pending
Send outstanding call information report
Apply_ Charging_ Report_ Pending
Send outstanding call information report

```
;FPAR
  StartState SSFStateType,
  SCFInitiated Boolean;
```

```
/*
IN CS−1 Operation processing procedures
*/
```

Process_ Apply_ Charging

Process_ Continue

Process_ Request_ Report_ BCSMEvent

Process_ Analyse_ Information

ProcessCall_ Information_ Request

Process_ Disconnect_ Forward_ Connection

Process_ Send_ Charging_ Information

Process_ SelectRoute

Process_ Cancel

Process_ Establish_ Temporary_ Connection

Process_ Event_ ReportBCSM

Process_ SelectFacility

Process_ Collect_ Information

Process_ Furnish_ Charging_ Information

Process_ Forward_ Connection_ Released

Process_ Connect

Process_ Initiate_ CallAttempt

Process_ InitialDP

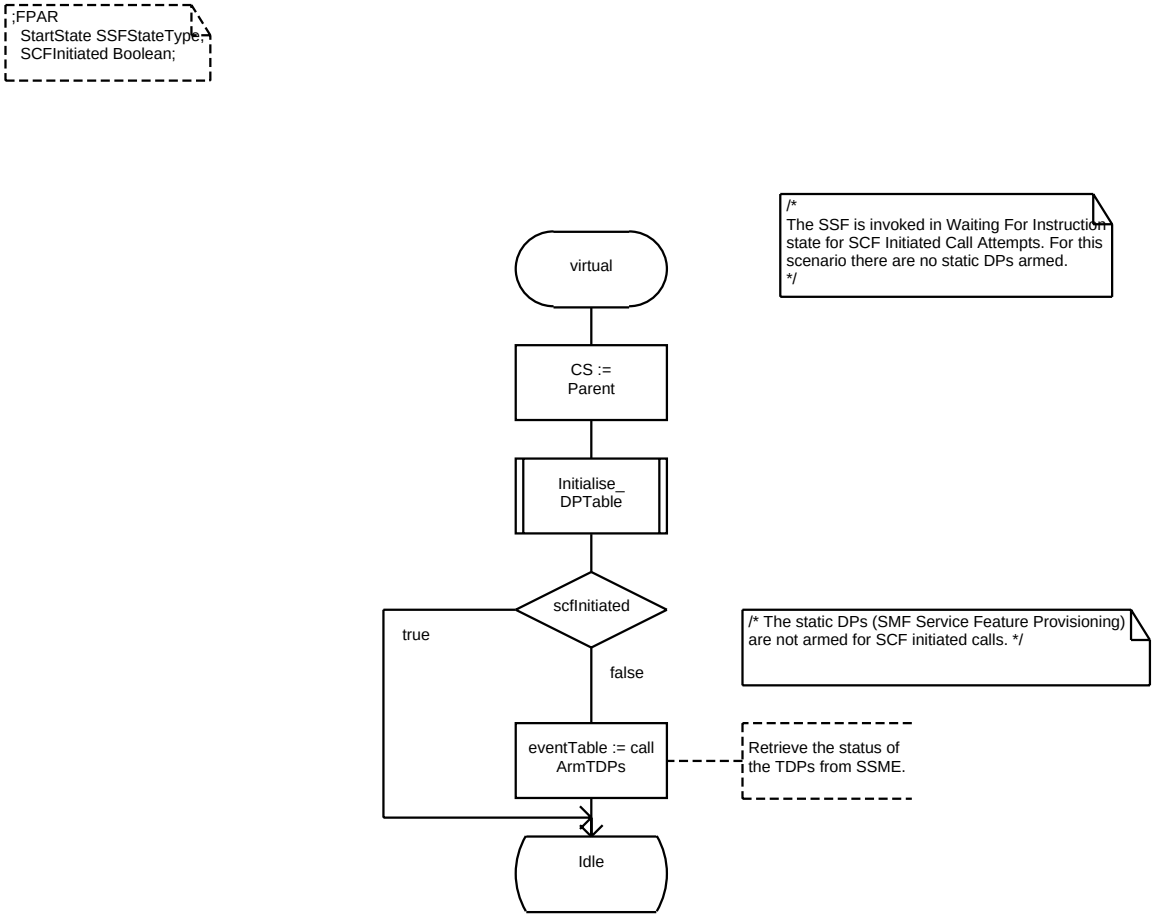
Process_ Connect_ ToResource

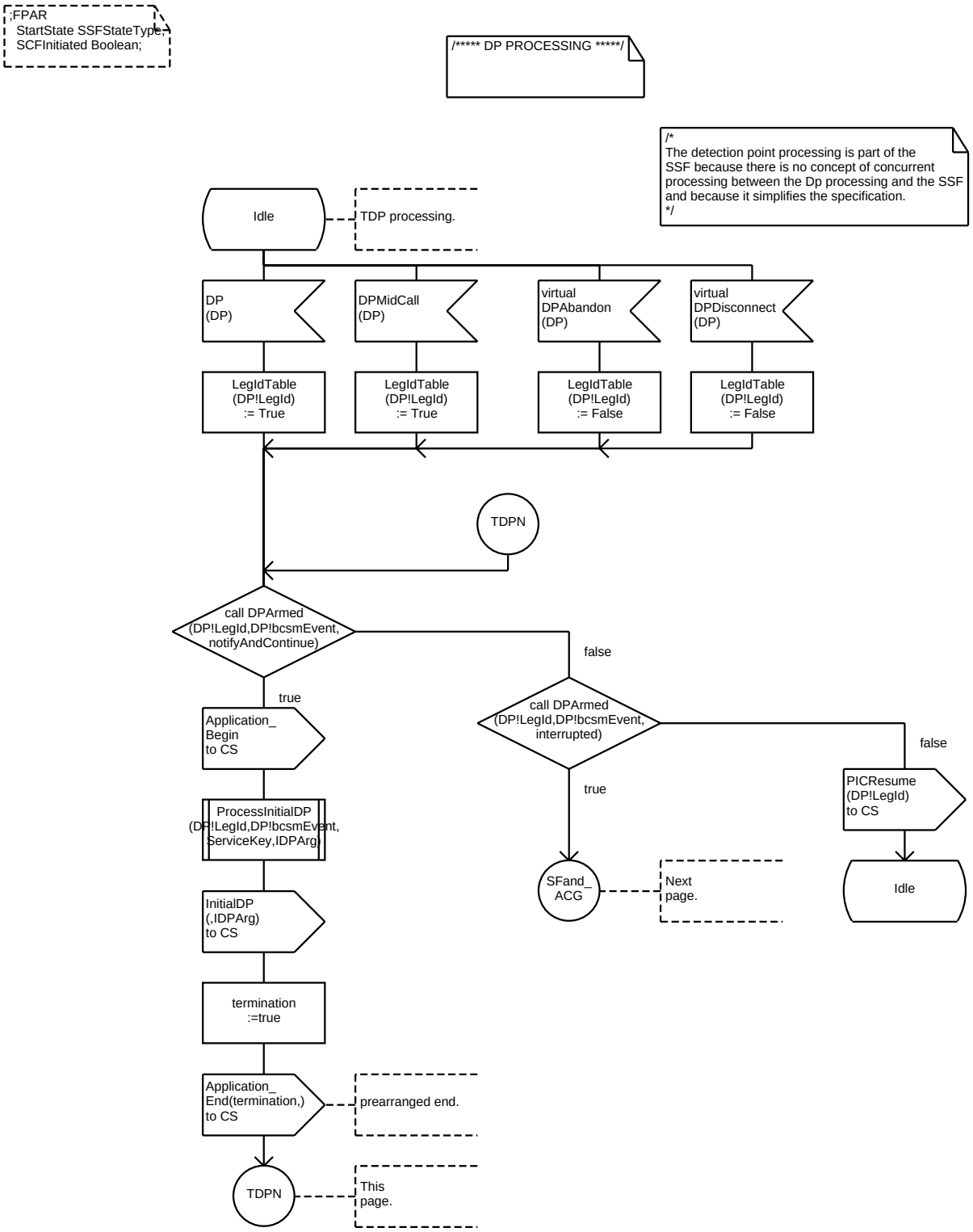
Process_ Request_ Notification_ ChargingEvent

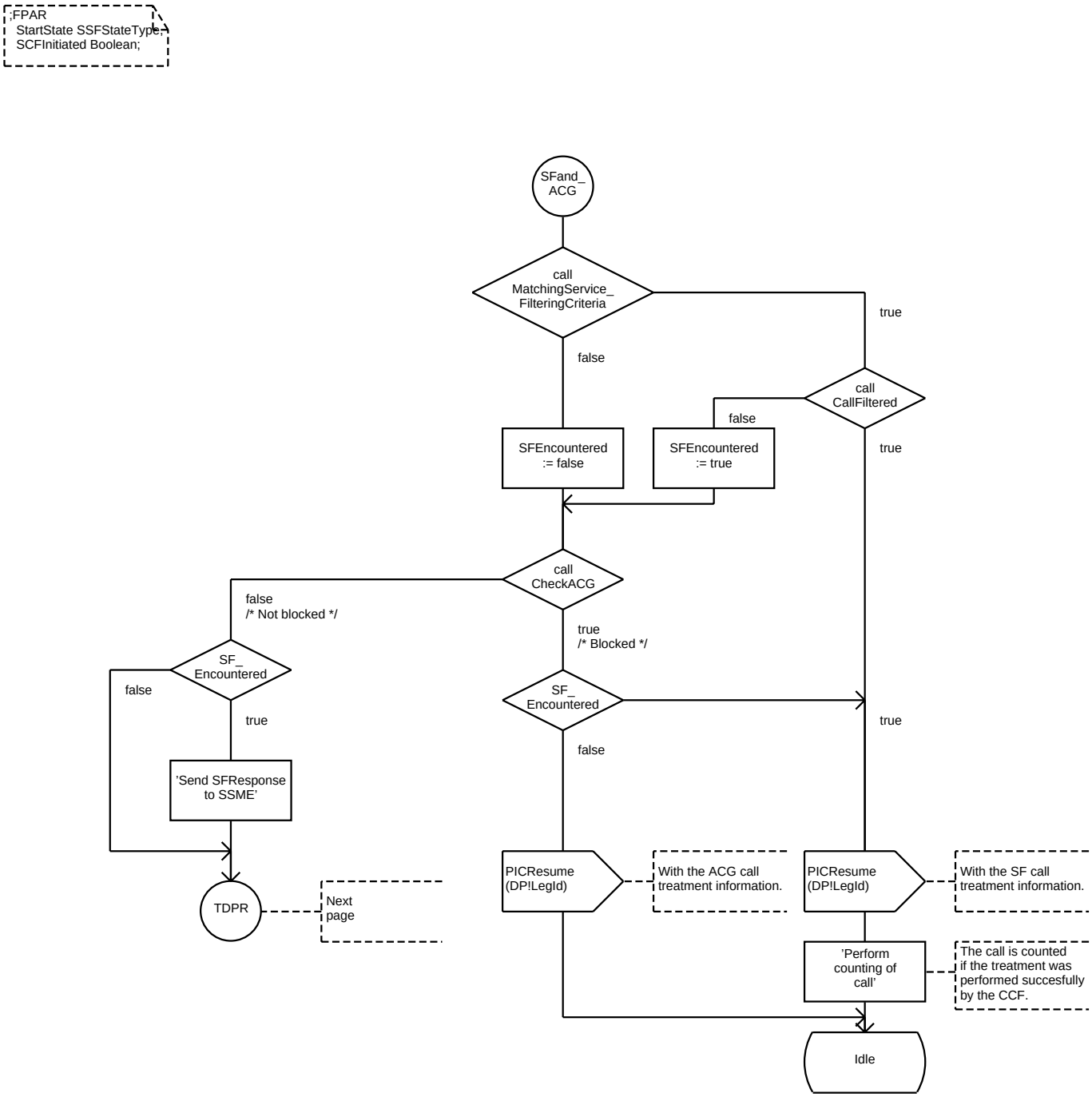
virtual ProcessDP_ Specific

```
/* Procedures for validation of operation arguments. */
```

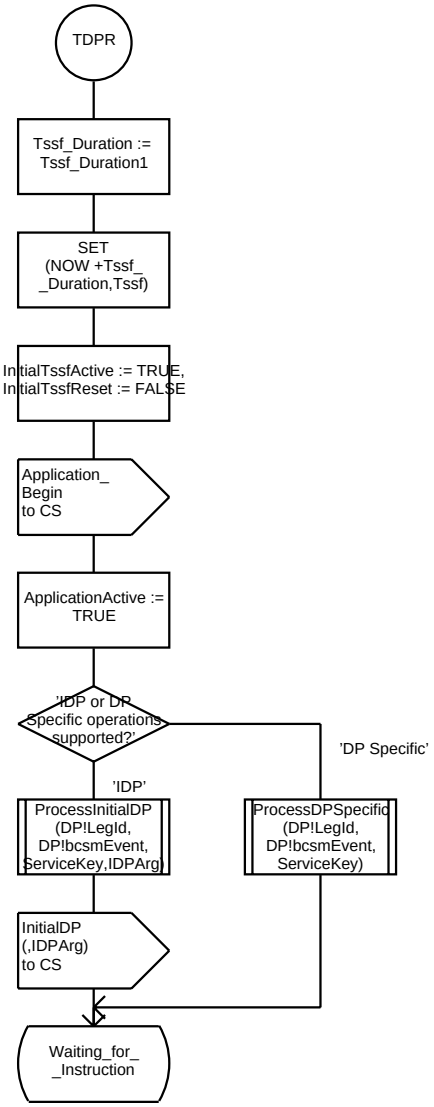
Connnect_ Analysis

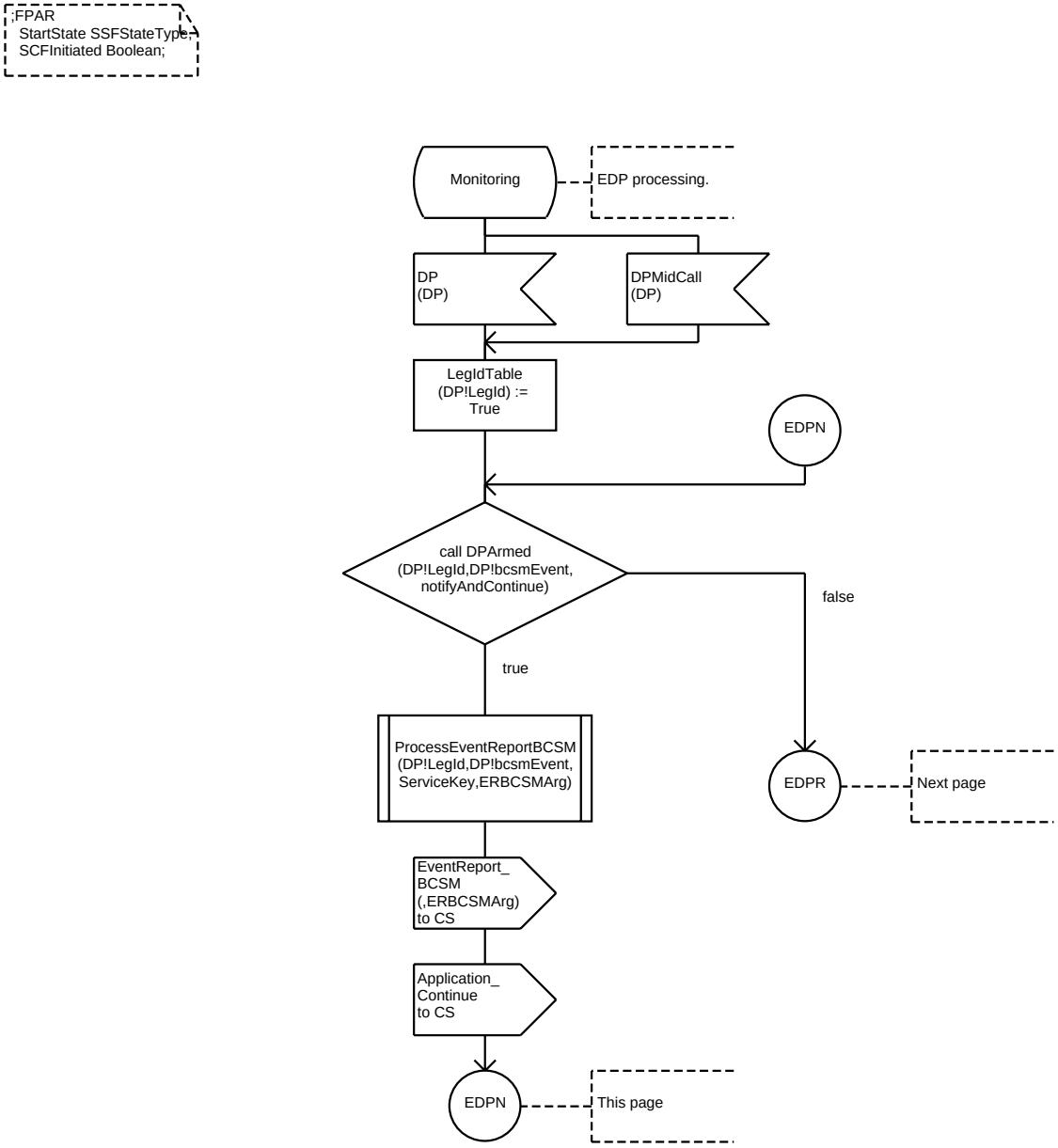

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/*
The SSF is invoked in Waiting For Instruction state for SCF Initiated Call Attempts. For this scenario there are no static DPs armed.
*/
virtual
CS :=
Parent
Initialise_
DPTable
scfInitiated
true
false
/* The static DPs (SMF Service Feature Provisioning) are not armed for SCF initiated calls. */
eventTable := call
ArmTDPs
Retrieve the status of the TDPs from SSME.
Idle

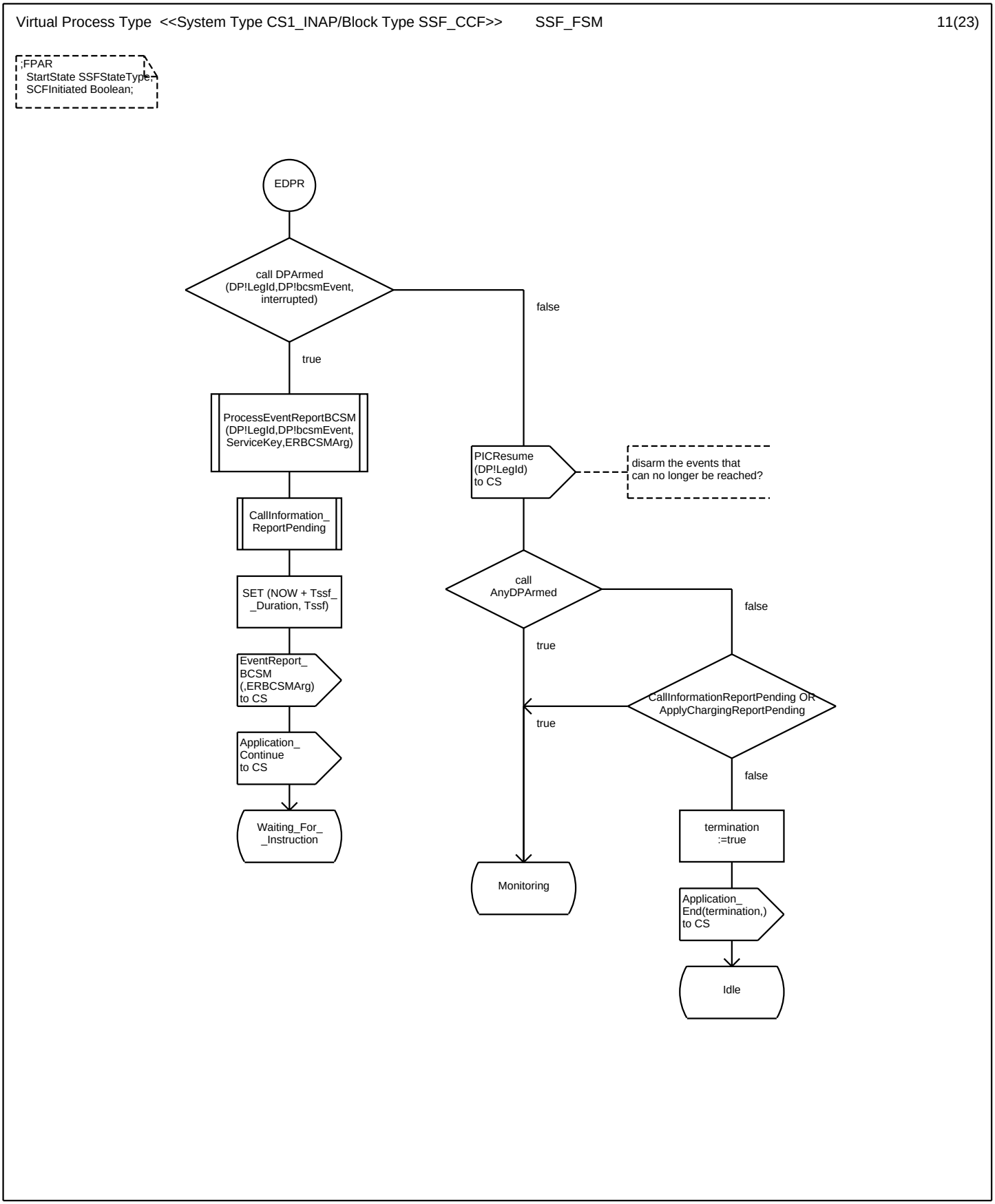

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/***** DP PROCESSING *****/
/*
The detection point processing is part of the SSF because there is no concept of concurrent processing between the Dp processing and the SSF and because it simplifies the specification.
*/
Idle
TDP processing.
DP (DP)
DPMidCall (DP)
virtual DPAbandon (DP)
virtual DPDisconnect (DP)
LegIdTable (DP!LegId) := True
LegIdTable (DP!LegId) := True
LegIdTable (DP!LegId) := False
LegIdTable (DP!LegId) := False
TDPN
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue)
false
true
Application_ Begin to CS
ProcessInitialDP (DP!LegId,DP!bcsmEvent, ServiceKey,IDPArg)
InitialDP (,IDPArg) to CS
termination :=true
Application_ End(termination,) to CS
prearranged end.
TDPN
This page.
call DPArmed (DP!LegId,DP!bcsmEvent, interrupted)
true
false
SFand_ ACG
Next page.
PICResume (DP!LegId) to CS
Idle

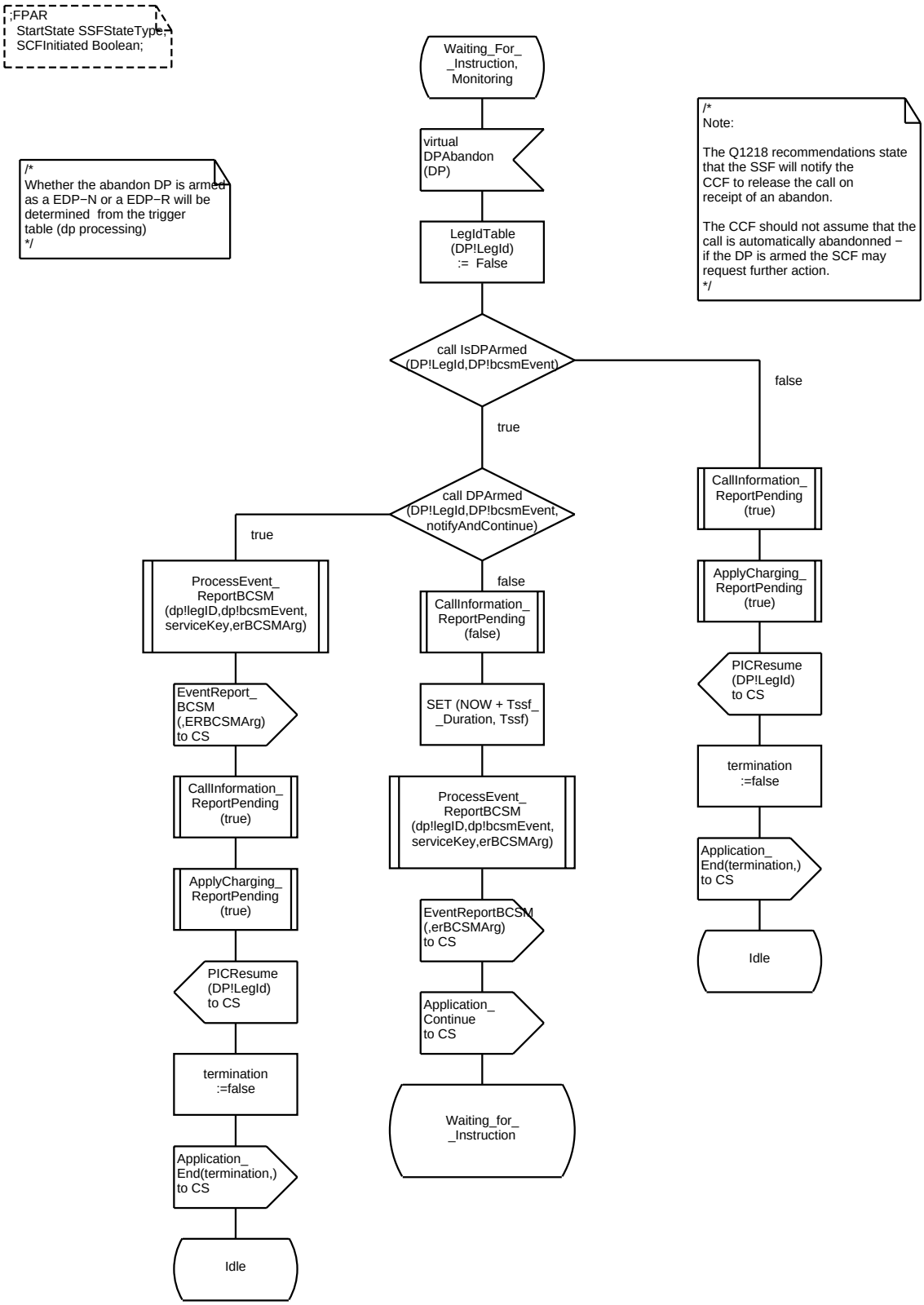

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
SFand_ ACG
call MatchingService_ FilteringCriteria
true
false
call CallFiltered
false
true
SFEncountered := false
SFEncountered := true
call CheckACG
false
/* Not blocked */
true
/* Blocked */
SF_ Encountered
false
true
'Send SFResponse to SSME'
SF_ Encountered
true
false
TDPR
Next page
PICResume (DP!LegId)
With the ACG call treatment information.
PICResume (DP!LegId)
With the SF call treatment information.
'Perform counting of call'
The call is counted if the treatment was performed succesfully by the CCF.
Idle

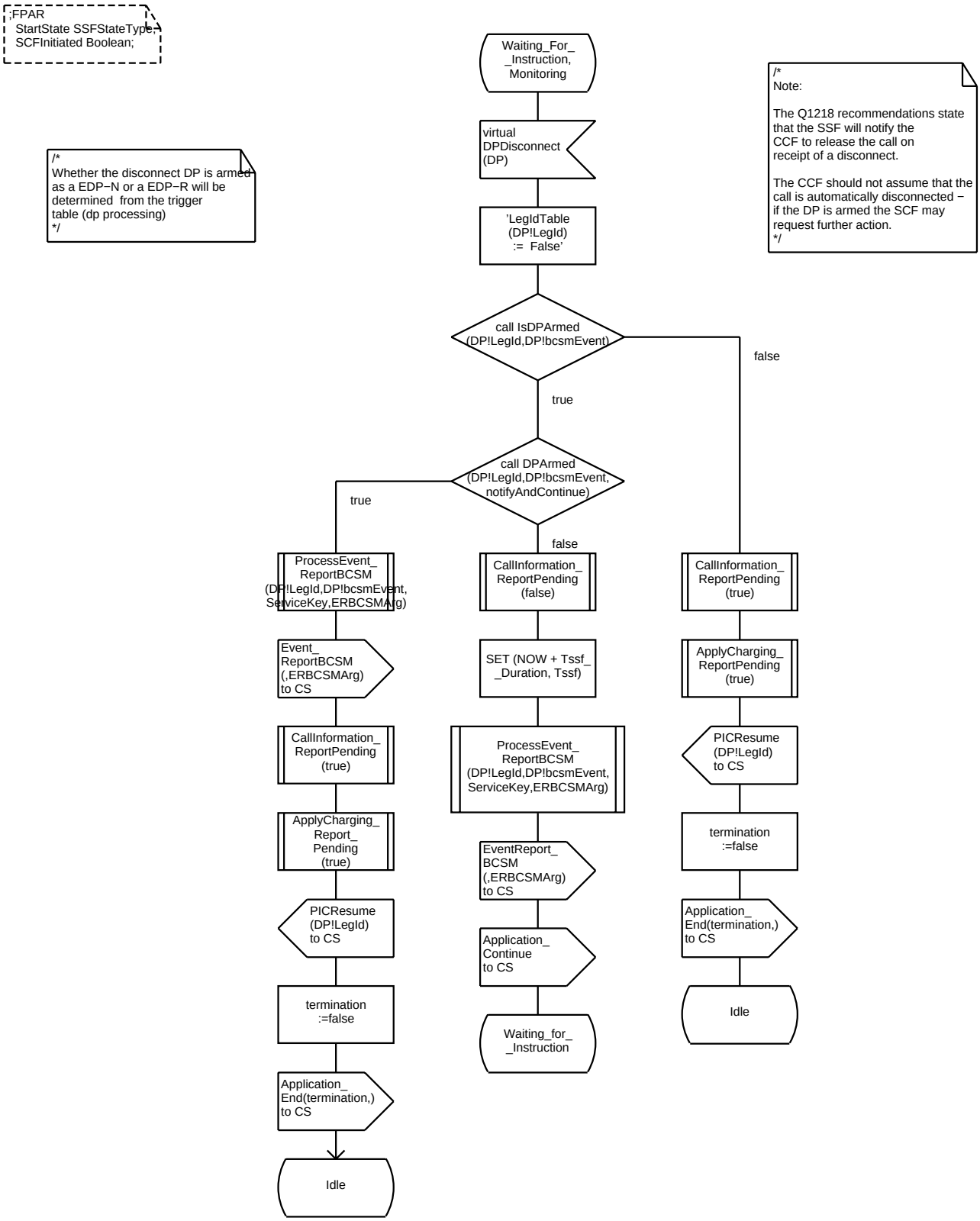
```
;FPAR
  StartState SSFStateType,
  SCFInitiated Boolean;
```

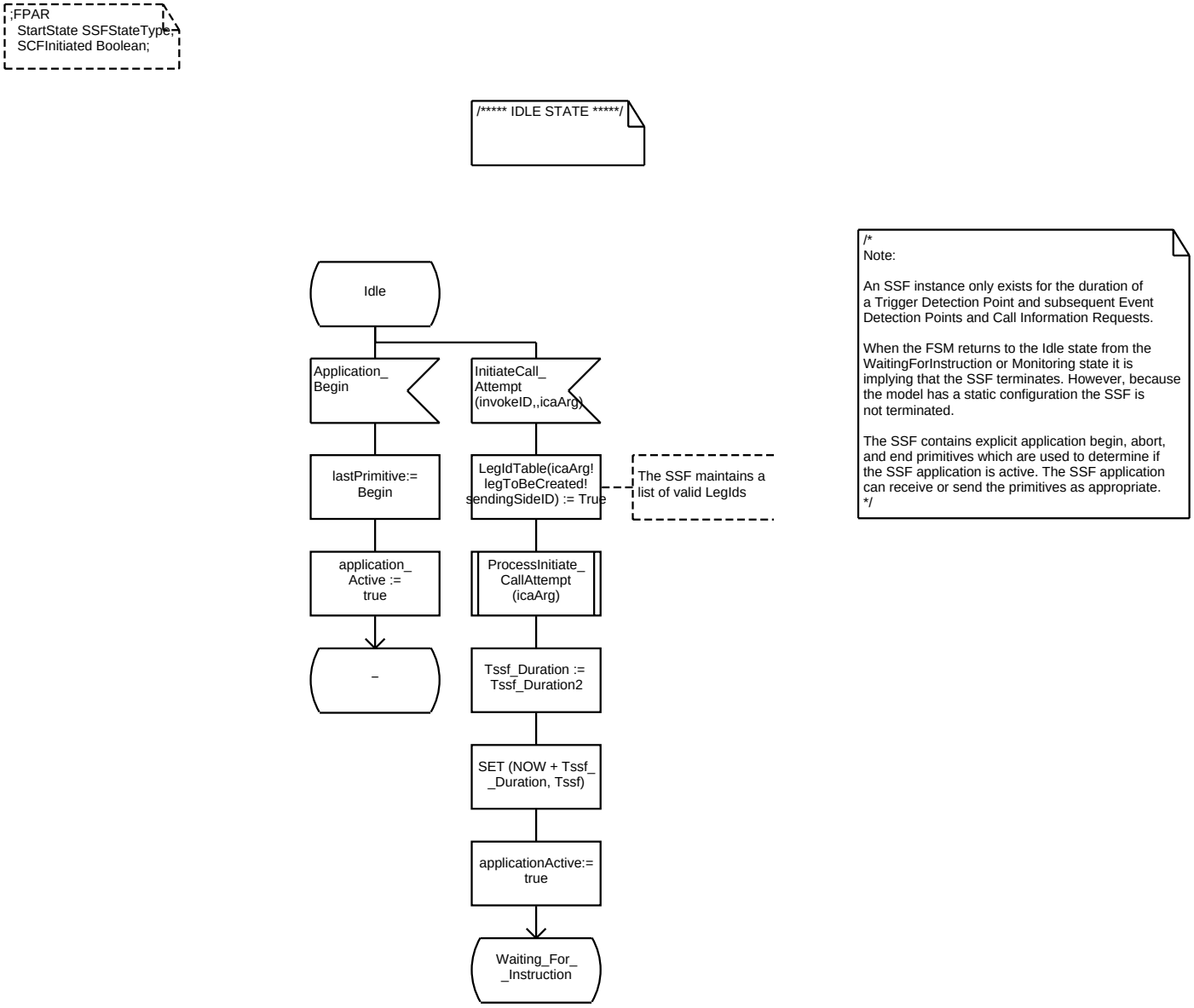


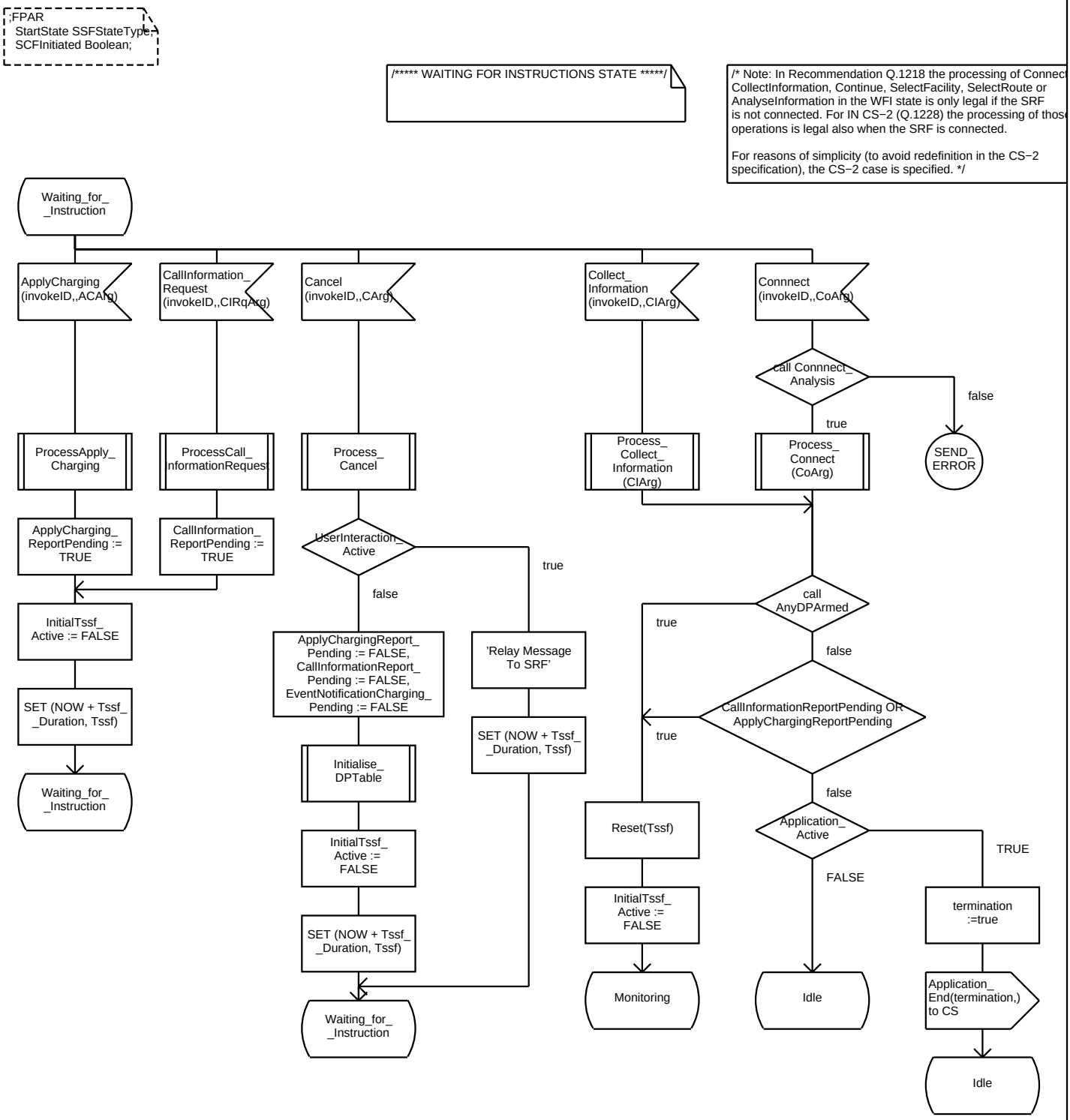

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
Monitoring
EDP processing.
DP
(DP)
DPMidCall
(DP)
LegIdTable
(DP!LegId) :=
True
EDPN
call DPArmed
(DP!LegId,DP!bcsmEvent,
notifyAndContinue)
false
true
ProcessEventReportBCSM
(DP!LegId,DP!bcsmEvent,
ServiceKey,ERBCSMArg)
EDPR
Next page
EventReport_
BCSM
(,ERBCSMArg)
to CS
Application_
Continue
to CS
EDPN
This page

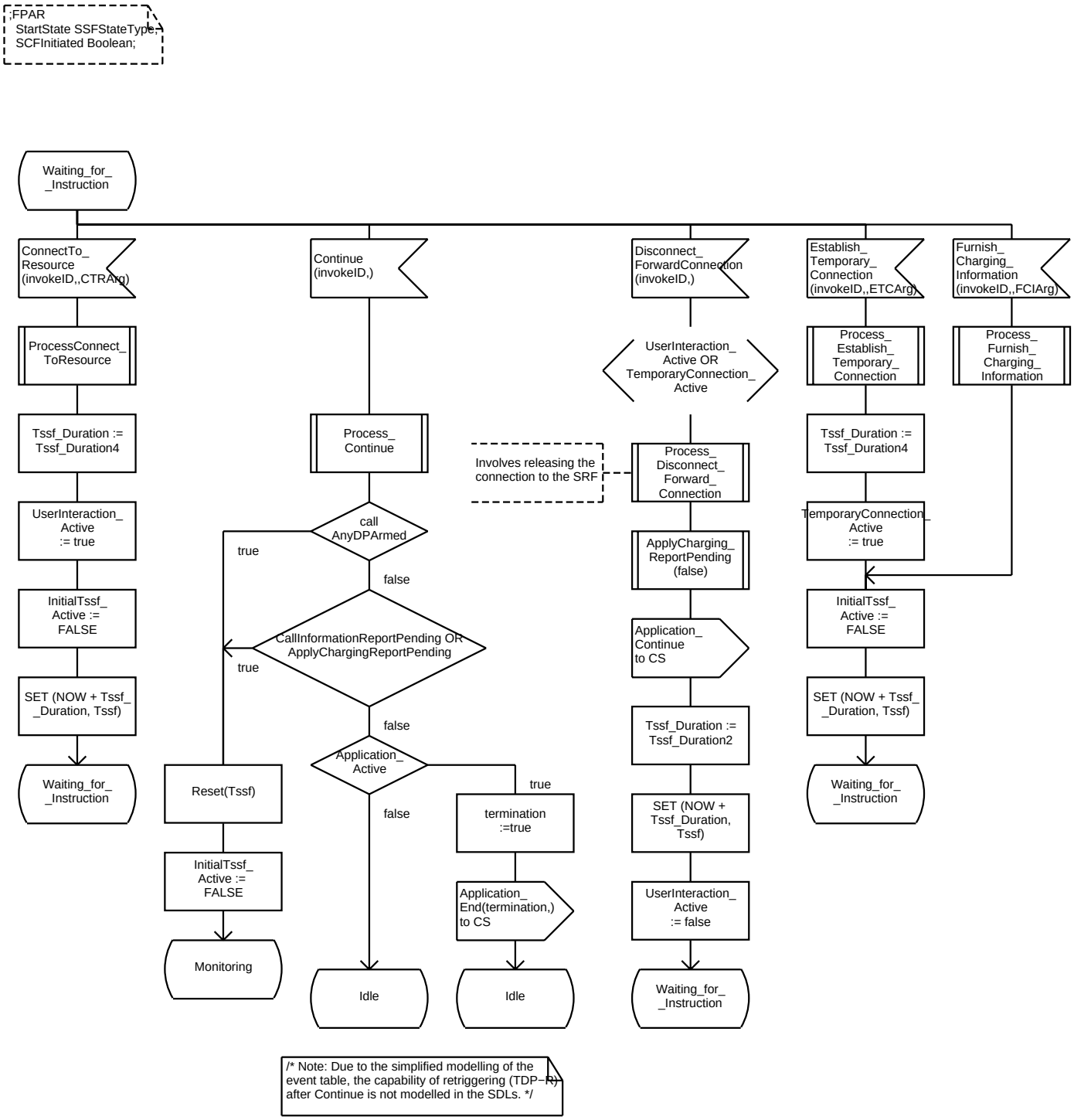

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
EDPR
call DPArmed
(DP!LegId,DP!bcsmEvent,
interrupted)
false
true
ProcessEventReportBCSM
(DP!LegId,DP!bcsmEvent,
ServiceKey,ERBCSMArg)
PICResume
(DP!LegId)
to CS
disarm the events that
can no longer be reached?
CallInformation_
ReportPending
SET (NOW + Tssf_
_Duration, Tssf)
call
AnyDPArmed
false
true
EventReport_
BCSM
(,ERBCSMArg)
to CS
CallInformationReportPending OR
ApplyChargingReportPending
true
Application_
Continue
to CS
false
Waiting_For_
_Instruction
termination
:=true
Monitoring
Application_
End(termination,)
to CS
Idle

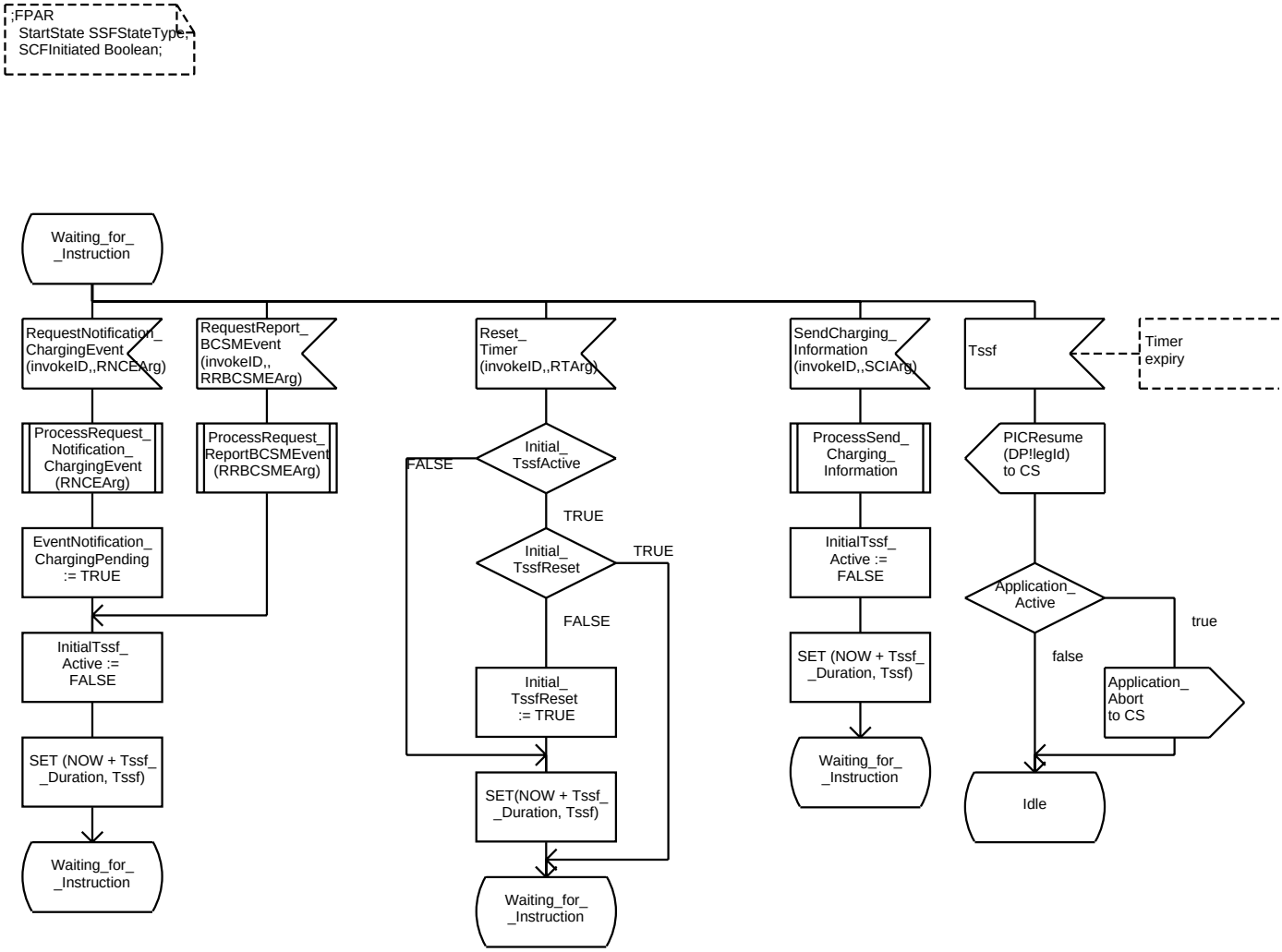

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/*
Whether the abandon DP is armed as a EDP−N or a EDP−R will be determined from the trigger table (dp processing)
*/
Waiting_For_ _Instruction, Monitoring
virtual DPAbandon (DP)
LegIdTable (DP!LegId) := False
/*
Note:
The Q1218 recommendations state that the SSF will notify the CCF to release the call on receipt of an abandon.
The CCF should not assume that the call is automatically abandonned − if the DP is armed the SCF may request further action.
*/
call IsDPArmed (DP!LegId,DP!bcsmEvent)
false
true
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue)
true
false
ProcessEvent_ ReportBCSM (dp!legID,dp!bcsmEvent, serviceKey,erBCSMArg)
EventReport_ BCSM (,ERBCSMArg) to CS
CallInformation_ ReportPending (true)
ApplyCharging_ ReportPending (true)
PICResume (DP!LegId) to CS
termination :=false
Application_ End(termination,) to CS
Idle
CallInformation_ ReportPending (false)
SET (NOW + Tssf_ _Duration, Tssf)
ProcessEvent_ ReportBCSM (dp!legID,dp!bcsmEvent, serviceKey,erBCSMArg)
EventReportBCSM (,erBCSMArg) to CS
Application_ Continue to CS
Waiting_for_ _Instruction
CallInformation_ ReportPending (true)
ApplyCharging_ ReportPending (true)
PICResume (DP!LegId) to CS
termination :=false
Application_ End(termination,) to CS
Idle

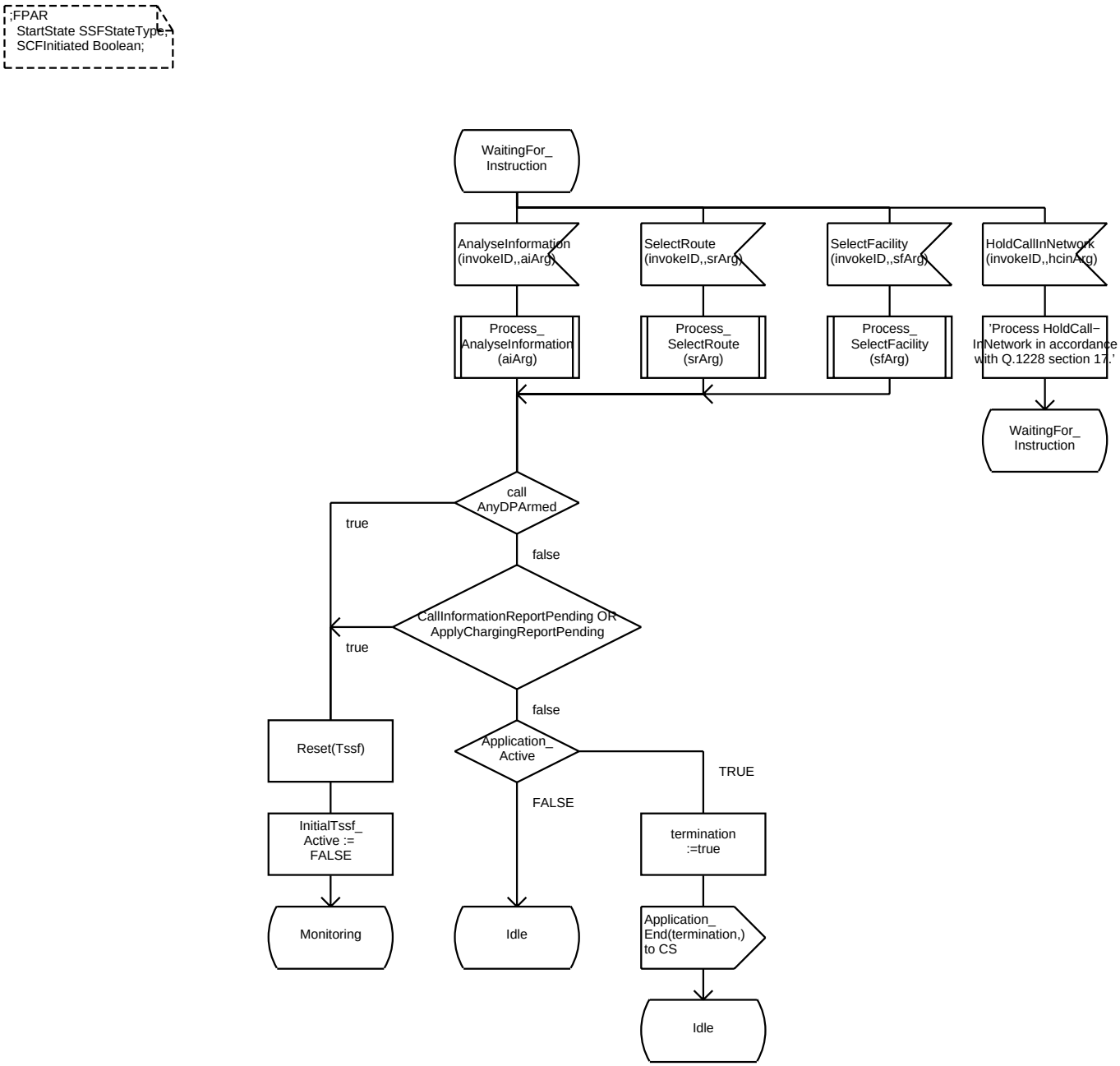

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/*
Whether the disconnect DP is armed as a EDP−N or a EDP−R will be determined from the trigger table (dp processing)
*/
Waiting_For_ _Instruction, Monitoring
virtual DPDisconnect (DP)
'LegIdTable (DP!LegId) := False'
/*
Note:
The Q1218 recommendations state that the SSF will notify the CCF to release the call on receipt of a disconnect.
The CCF should not assume that the call is automatically disconnected − if the DP is armed the SCF may request further action.
*/
call IsDPArmed (DP!LegId,DP!bcsmEvent)
false
true
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue)
true
false
ProcessEvent_ ReportBCSM (DP!LegId,DP!bcsmEvent, ServiceKey,ERBCSMArg)
Event_ ReportBCSM (,ERBCSMArg) to CS
CallInformation_ ReportPending (true)
ApplyCharging_ Report_ Pending (true)
PICResume (DP!LegId) to CS
termination :=false
Application_ End(termination,) to CS
Idle
CallInformation_ ReportPending (false)
SET (NOW + Tssf_ _Duration, Tssf)
ProcessEvent_ ReportBCSM (DP!LegId,DP!bcsmEvent, ServiceKey,ERBCSMArg)
EventReport_ BCSM (,ERBCSMArg) to CS
Application_ Continue to CS
Waiting_for_ _Instruction
CallInformation_ ReportPending (true)
ApplyCharging_ ReportPending (true)
PICResume (DP!LegId) to CS
termination :=false
Application_ End(termination,) to CS
Idle

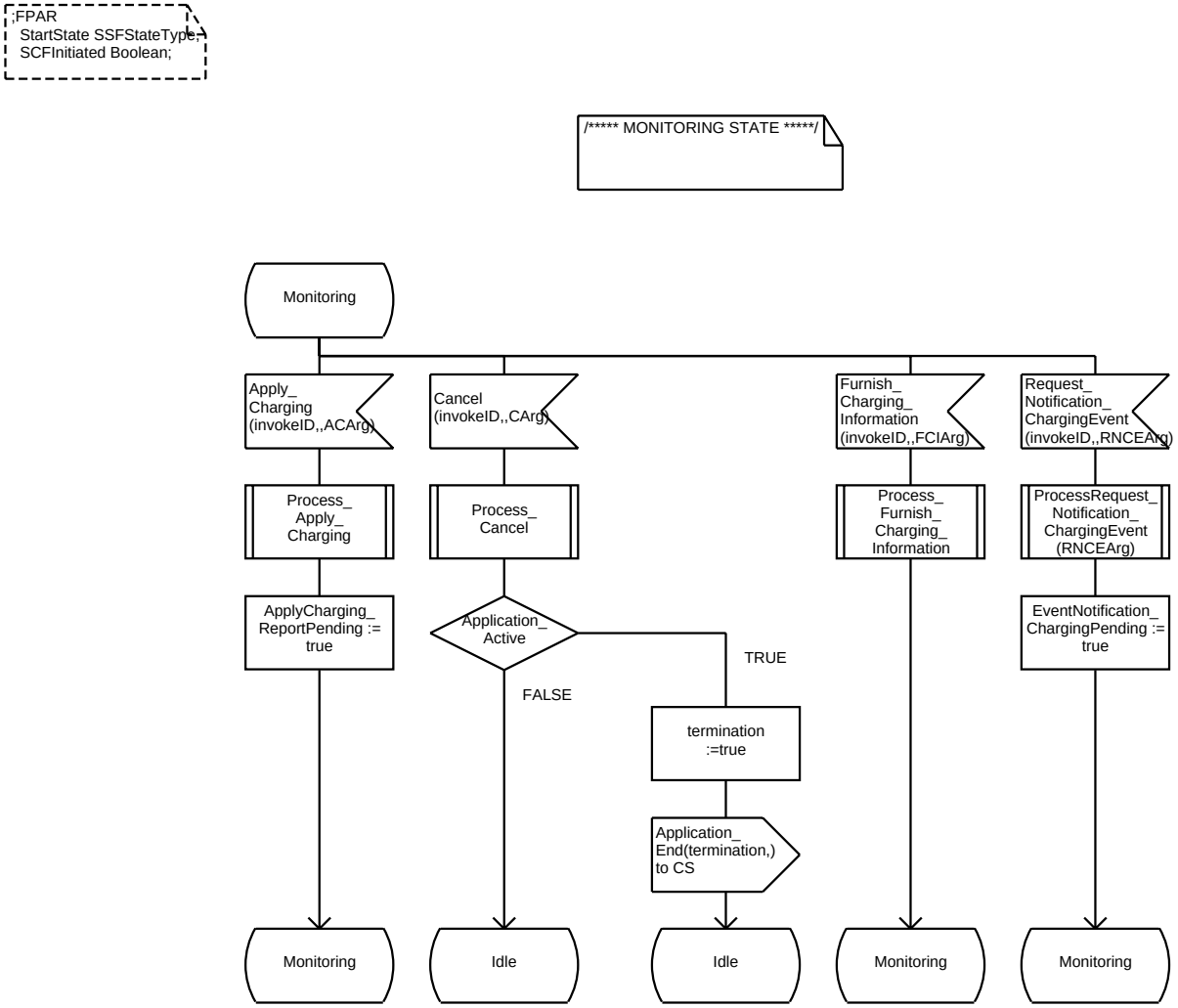

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/***** IDLE STATE *****/
Idle
Application_
Begin
InitiateCall_
Attempt
(invokeID,,icaArg)
lastPrimitive:=
Begin
LegIdTable(icaArg!
legToBeCreated!
sendingSideID) := True
The SSF maintains a
list of valid LegIds
application_
Active :=
true
ProcessInitiate_
CallAttempt
(icaArg)
_
Tssf_Duration :=
Tssf_Duration2
SET (NOW + Tssf_
_Duration, Tssf)
applicationActive:=
true
Waiting_For_
_Instruction
/*
Note:
An SSF instance only exists for the duration of
a Trigger Detection Point and subsequent Event
Detection Points and Call Information Requests.
When the FSM returns to the Idle state from the
WaitingForInstruction or Monitoring state it is
implying that the SSF terminates. However, because
the model has a static configuration the SSF is
not terminated.
The SSF contains explicit application begin, abort,
and end primitives which are used to determine if
the SSF application is active. The SSF application
can receive or send the primitives as appropriate.
*/

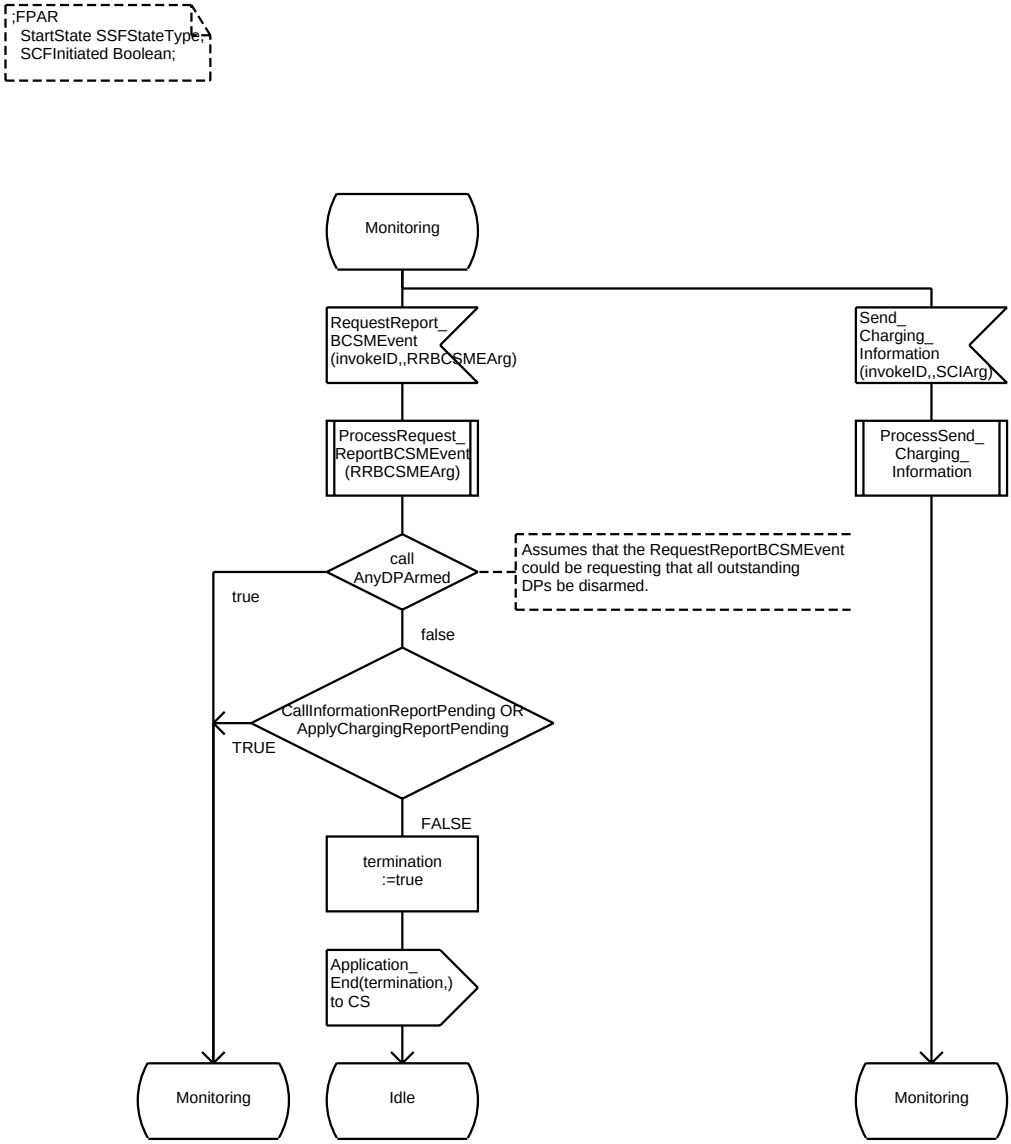

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/***** WAITING FOR INSTRUCTIONS STATE *****/
/* Note: In Recommendation Q.1218 the processing of Connect
CollectInformation, Continue, SelectFacility, SelectRoute or
AnalyseInformation in the WFI state is only legal if the SRF
is not connected. For IN CS−2 (Q.1228) the processing of those
operations is legal also when the SRF is connected.
For reasons of simplicity (to avoid redefinition in the CS−2
specification), the CS−2 case is specified. */
Waiting_for_
_Instruction
ApplyCharging
(invokeID,,ACArg)
CallInformation_
Request
(invokeID,,CIRqArg)
Cancel
(invokeID,,CArg)
Collect_
Information
(invokeID,,CIArg)
Connnect
(invokeID,,CoArg)
call Connnect_
Analysis
false
true
SEND_
ERROR
ProcessApply_
Charging
ProcessCall_
InformationRequest
Process_
Cancel
Process_
Collect_
Information
(CIArg)
Process_
Connect
(CoArg)
ApplyCharging_
ReportPending :=
TRUE
CallInformation_
ReportPending :=
TRUE
UserInteraction_
Active
true
false
call
AnyDPArmed
true
false
InitialTssf_
Active := FALSE
ApplyChargingReport_
Pending := FALSE,
CallInformationReport_
Pending := FALSE,
EventNotificationCharging_
Pending := FALSE
'Relay Message
To SRF'
CallInformationReportPending OR
ApplyChargingReportPending
true
SET (NOW + Tssf_
_Duration, Tssf)
SET (NOW + Tssf_
_Duration, Tssf)
Initialise_
DPTable
false
Reset(Tssf)
Application_
Active
TRUE
FALSE
Waiting_for_
_Instruction
InitialTssf_
Active :=
FALSE
InitialTssf_
Active :=
FALSE
termination
:=true
SET (NOW + Tssf_
_Duration, Tssf)
Monitoring
Idle
Application_
End(termination,)
to CS
Waiting_for_
_Instruction
Idle

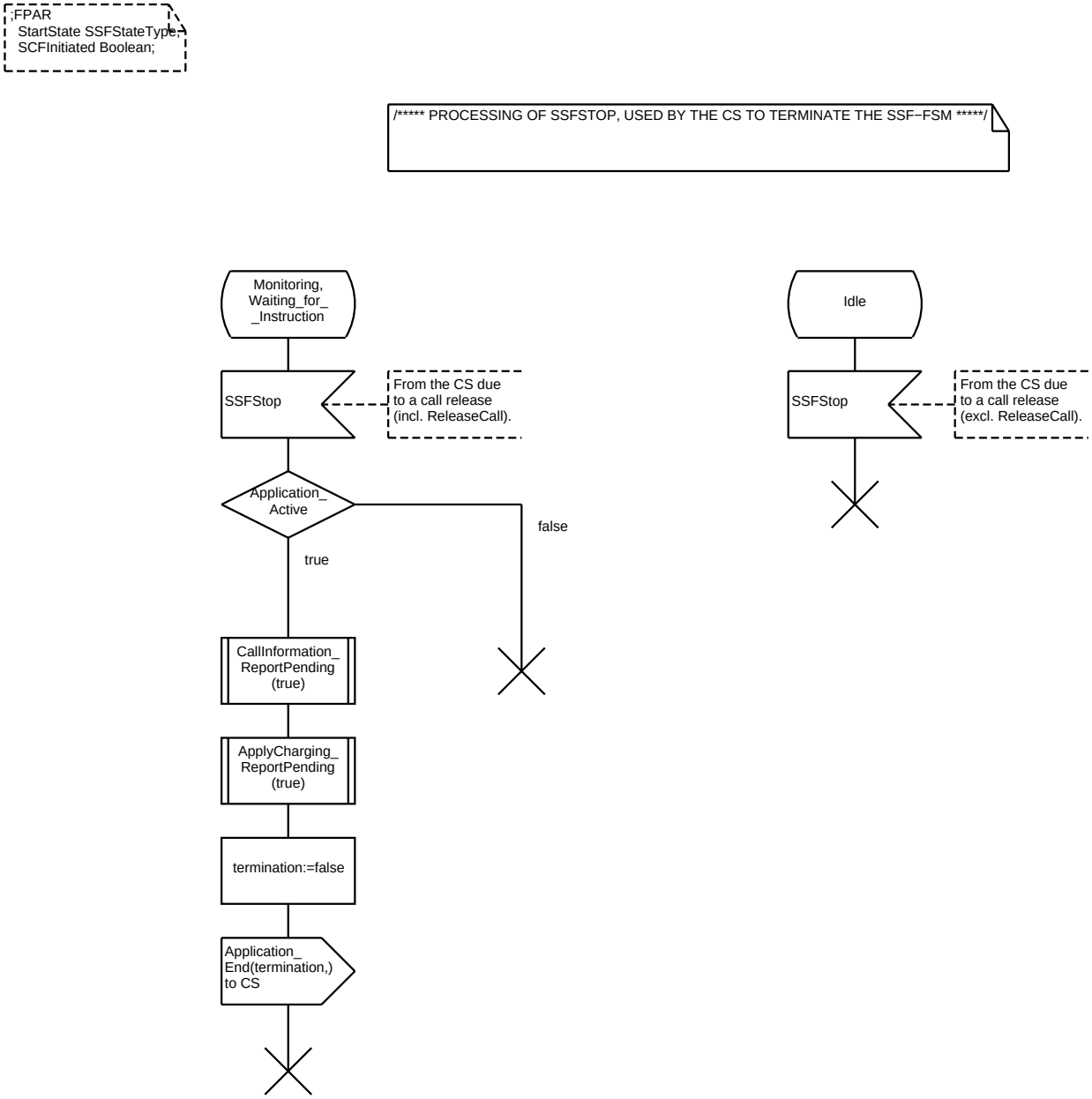

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
Waiting_for_
_Instruction
ConnectTo_
Resource
(invokeID,,CTRArg)
ProcessConnect_
ToResource
Tssf_Duration :=
Tssf_Duration4
UserInteraction_
Active
:= true
InitialTssf_
Active :=
FALSE
SET (NOW + Tssf_
_Duration, Tssf)
Waiting_for_
_Instruction
Continue
(invokeID,)
Process_
Continue
call
AnyDPArmed
true
false
CallInformationReportPending OR
ApplyChargingReportPending
true
false
Application_
Active
true
false
Reset(Tssf)
InitialTssf_
Active :=
FALSE
Monitoring
termination
:=true
Application_
End(termination,)
to CS
Idle
Idle
Disconnect_
ForwardConnection
(invokeID,)
UserInteraction_
Active OR
TemporaryConnection_
Active
Involves releasing the
connection to the SRF
Process_
Disconnect_
Forward_
Connection
ApplyCharging_
ReportPending
(false)
Application_
Continue
to CS
Tssf_Duration :=
Tssf_Duration2
SET (NOW +
Tssf_Duration,
Tssf)
UserInteraction_
Active
:= false
Waiting_for_
_Instruction
Establish_
Temporary_
Connection
(invokeID,,ETCArg)
Process_
Establish_
Temporary_
Connection
Tssf_Duration :=
Tssf_Duration4
TemporaryConnection_
Active
:= true
InitialTssf_
Active :=
FALSE
SET (NOW + Tssf_
_Duration, Tssf)
Waiting_for_
_Instruction
Furnish_
Charging_
Information
(invokeID,,FCIArg)
Process_
Furnish_
Charging_
Information
/* Note: Due to the simplified modelling of the
event table, the capability of retriggering (TDP−R)
after Continue is not modelled in the SDLs. */

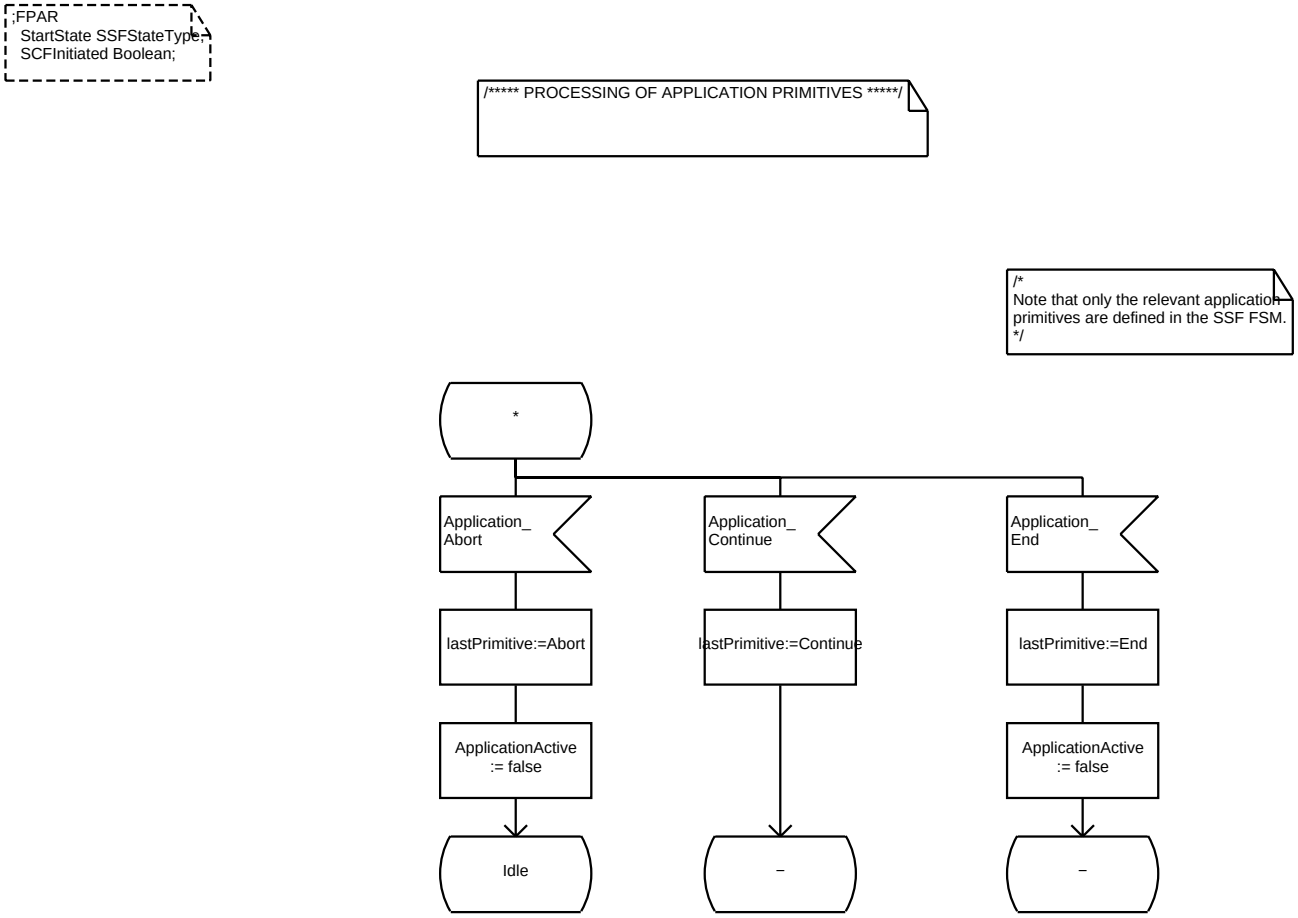

;FPAR
StartState SSFStateType;
SCFInitiated Boolean;
Waiting_for_
_Instruction
RequestNotification_
ChargingEvent
(invokeID,,RNCEArg)
RequestReport_
BCSMEvent
(invokeID,,
RRBCSMEArg)
Reset_
Timer
(invokeID,,RTArg)
SendCharging_
Information
(invokeID,,SCIArg)
Tssf
Timer
expiry
ProcessRequest_
Notification_
ChargingEvent
(RNCEArg)
ProcessRequest_
ReportBCSMEvent
(RRBCSMEArg)
Initial_
TssfActive
FALSE
TRUE
Initial_
TssfReset
TRUE
FALSE
ProcessSend_
Charging_
Information
PICResume
(DP!legId)
to CS
EventNotification_
ChargingPending
:= TRUE
InitialTssf_
Active :=
FALSE
Initial_
TssfReset
:= TRUE
InitialTssf_
Active :=
FALSE
Application_
Active
true
false
Application_
Abort
to CS
SET (NOW + Tssf_
_Duration, Tssf)
SET (NOW + Tssf_
_Duration, Tssf)
SET (NOW + Tssf_
_Duration, Tssf)
Waiting_for_
_Instruction
Waiting_for_
_Instruction
Waiting_for_
_Instruction
Idle

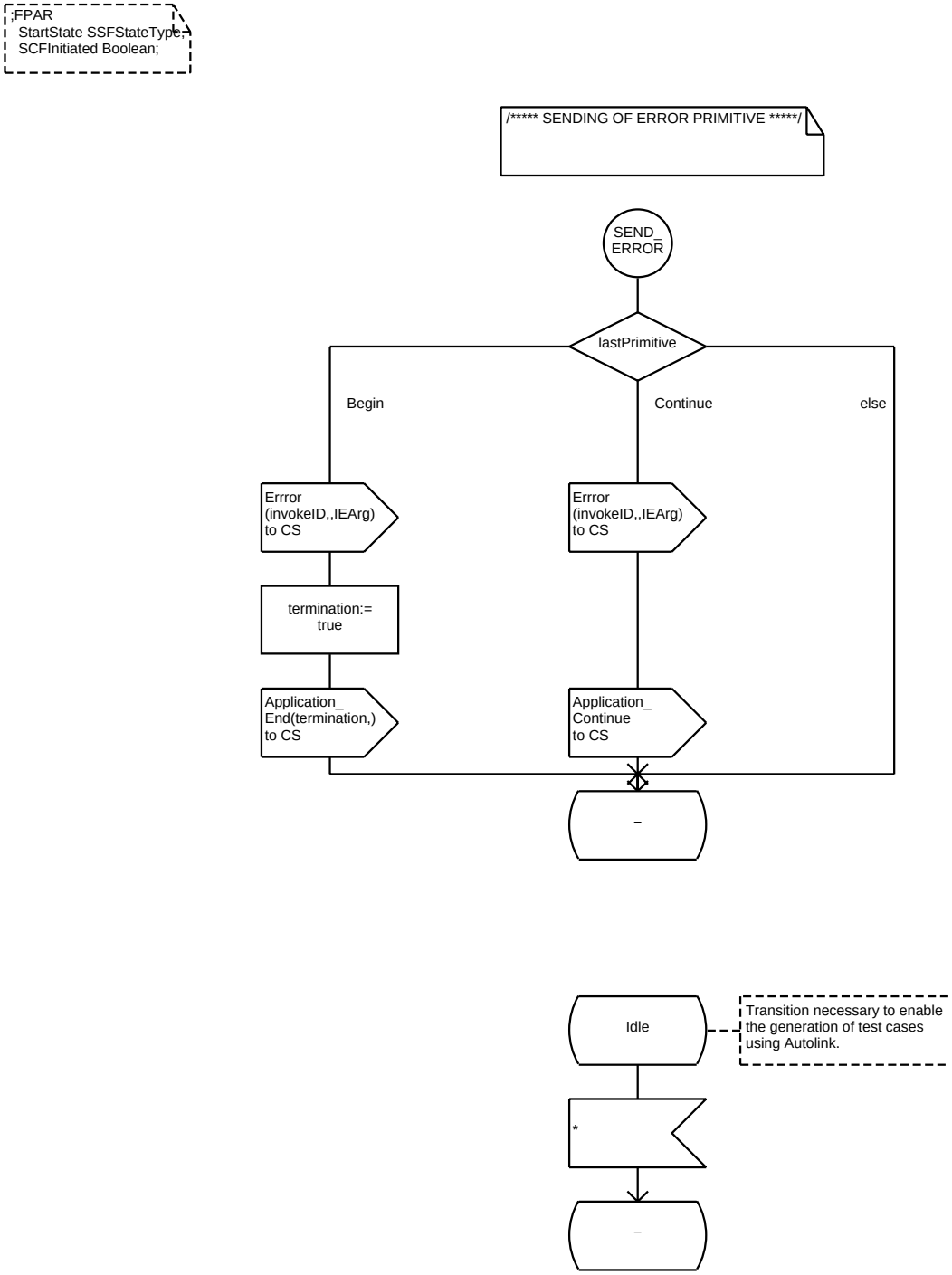

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
WaitingFor_ Instruction
AnalyseInformation (invokeID,,aiArg)
SelectRoute (invokeID,,srArg)
SelectFacility (invokeID,,sfArg)
HoldCallInNetwork (invokeID,,hcinArg)
Process_ AnalyseInformation (aiArg)
Process_ SelectRoute (srArg)
Process_ SelectFacility (sfArg)
'Process HoldCall−InNetwork in accordance with Q.1228 section 17.'
WaitingFor_ Instruction
call AnyDPArmed
true
false
CallInformationReportPending OR ApplyChargingReportPending
true
false
Reset(Tssf)
Application_ Active
TRUE
FALSE
InitialTssf_ Active := FALSE
termination :=true
Monitoring
Idle
Application_ End(termination,) to CS
Idle


;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/***** MONITORING STATE *****/
Monitoring
Apply_ Charging (invokeID,,ACArg)
Cancel (invokeID,,CArg)
Furnish_ Charging_ Information (invokeID,,FCIArg)
Request_ Notification_ ChargingEvent (invokeID,,RNCEArg)
Process_ Apply_ Charging
Process_ Cancel
Process_ Furnish_ Charging_ Information
ProcessRequest_ Notification_ ChargingEvent (RNCEArg)
ApplyCharging_ ReportPending := true
Application_ Active
TRUE
FALSE
EventNotification_ ChargingPending := true
termination :=true
Application_ End(termination,) to CS
Monitoring
Idle
Idle
Monitoring
Monitoring

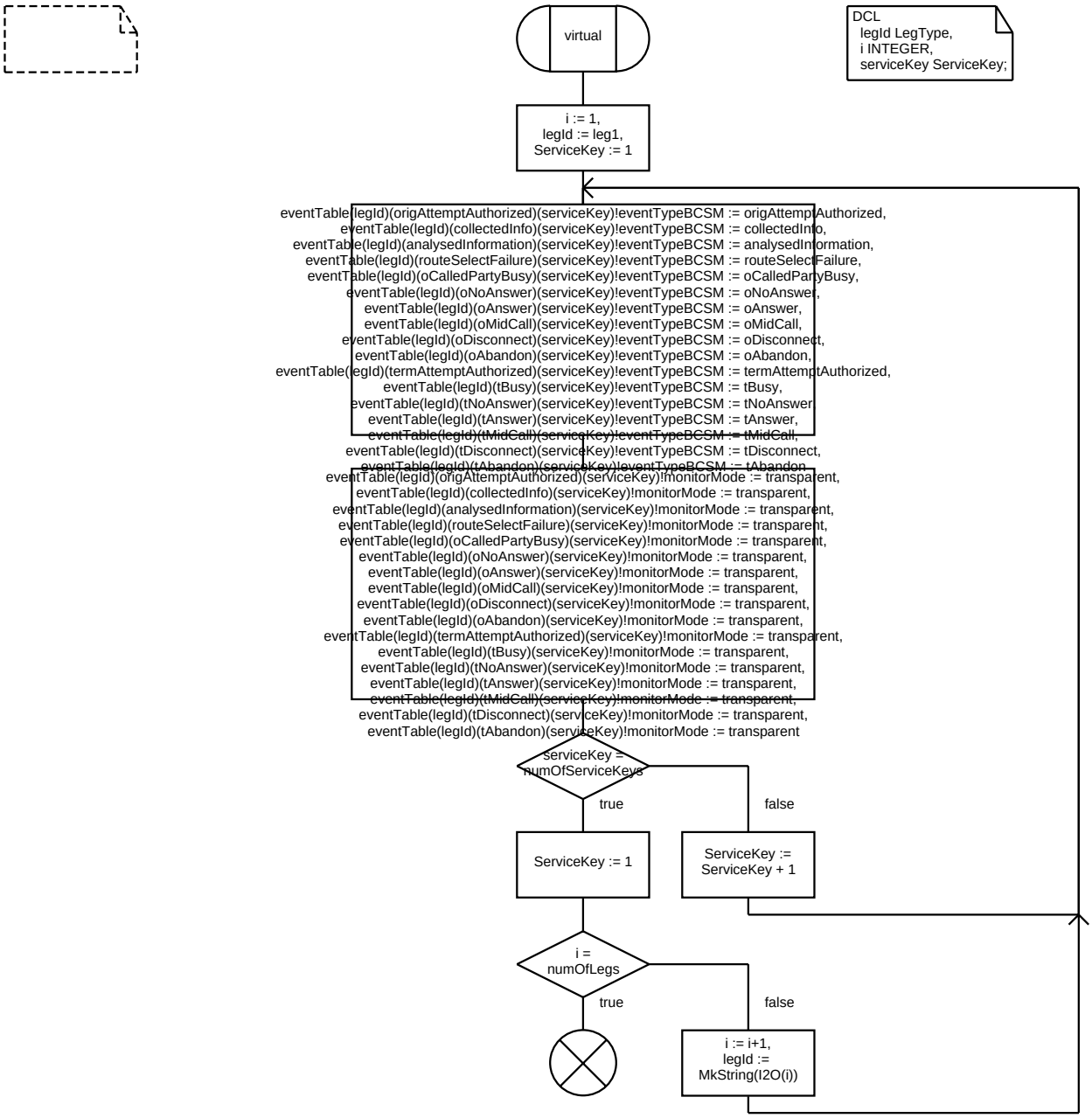

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
Monitoring
RequestReport_
BCSMEvent
(invokeID,,RRBCSMEArg)
Send_
Charging_
Information
(invokeID,,SCIArg)
ProcessRequest_
ReportBCSMEvent
(RRBCSMEArg)
ProcessSend_
Charging_
Information
call
AnyDPArmed
Assumes that the RequestReportBCSMEvent
could be requesting that all outstanding
DPs be disarmed.
true
false
CallInformationReportPending OR
ApplyChargingReportPending
TRUE
FALSE
termination
:=true
Application_
End(termination,)
to CS
Monitoring
Idle
Monitoring


;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/***** PROCESSING OF SSFSTOP, USED BY THE CS TO TERMINATE THE SSF−FSM *****/
Monitoring, Waiting_for_ _Instruction
SSFStop
From the CS due to a call release (incl. ReleaseCall).
Application_ Active
false
true
CallInformation_ ReportPending (true)
ApplyCharging_ ReportPending (true)
termination:=false
Application_ End(termination,) to CS
Idle
SSFStop
From the CS due to a call release (excl. ReleaseCall).

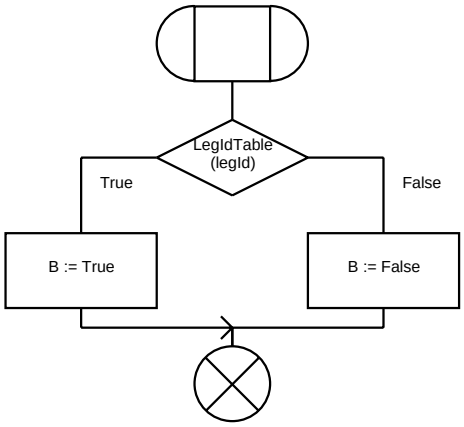

;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/***** PROCESSING OF APPLICATION PRIMITIVES *****/
/*
Note that only the relevant application primitives are defined in the SSF FSM.
*/
*
Application_ Abort
Application_ Continue
Application_ End
lastPrimitive:=Abort
lastPrimitive:=Continue
lastPrimitive:=End
ApplicationActive := false
ApplicationActive := false
Idle
–
–


;FPAR
StartState SSFStateType,
SCFInitiated Boolean;
/***** SENDING OF ERROR PRIMITIVE *****/
SEND_
ERROR
lastPrimitive
Begin
Continue
else
Errror
(invokeID,,IEArg)
to CS
Errror
(invokeID,,IEArg)
to CS
termination:=
true
Application_
End(termination,)
to CS
Application_
Continue
to CS
–
Idle
Transition necessary to enable
the generation of test cases
using Autolink.
*
–

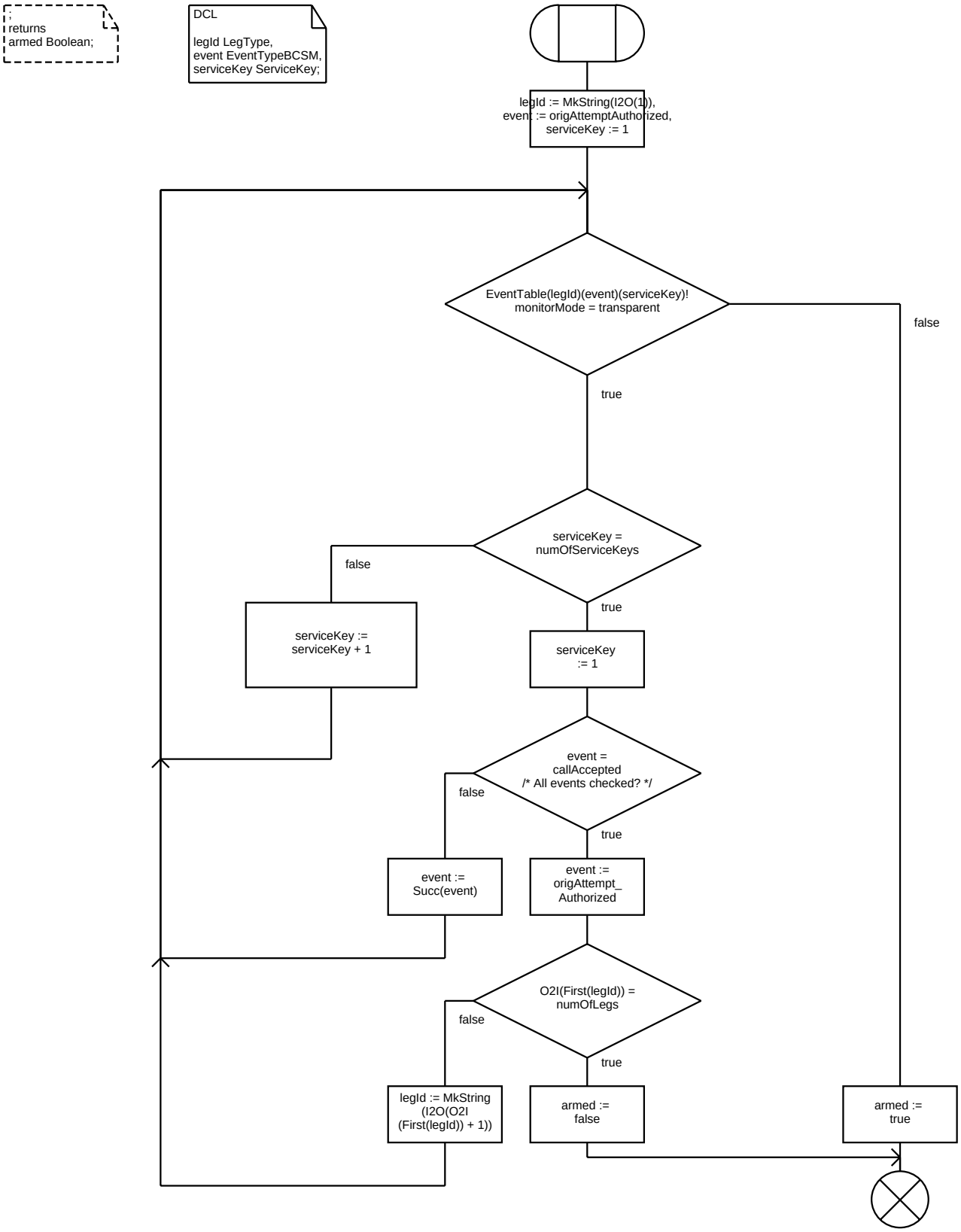
Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type SSF_FSM>> InitialiseDPTable 1(1)



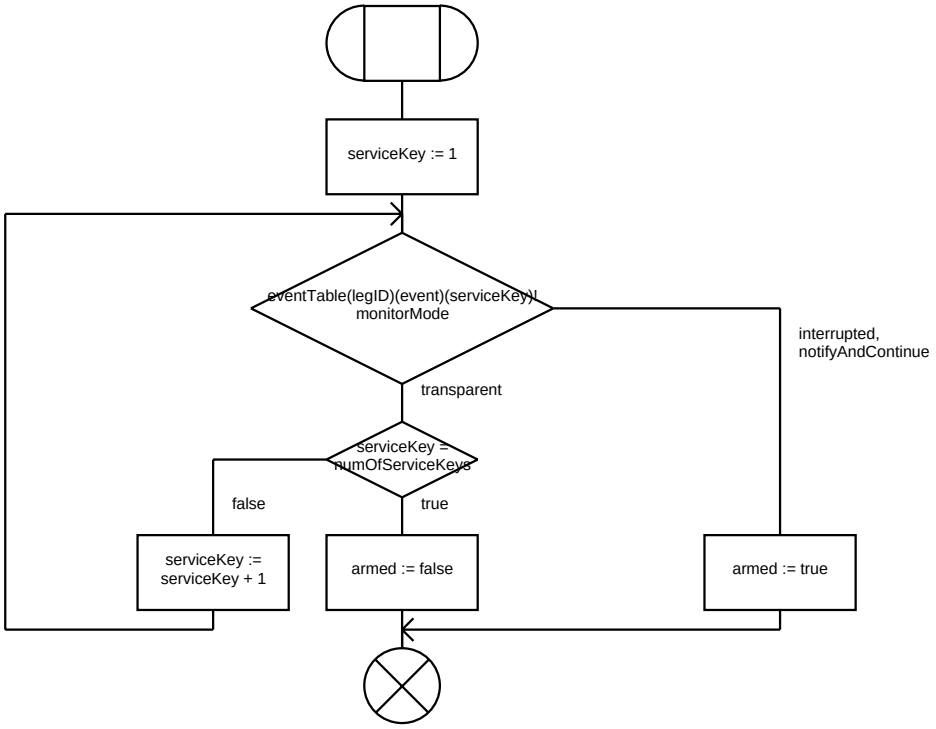
```
;FPAR
  IN legId LegType;
RETURNS B Boolean;
```



Procedure AnyDPArmed

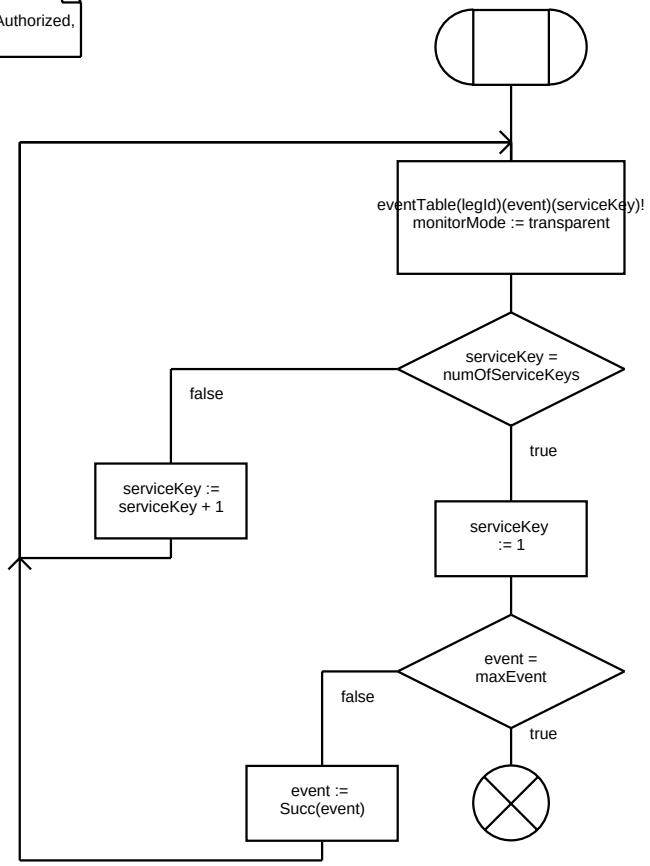


```
;
FPAR
IN legID LegType,
IN event EventTypeBCSM;
returns armed Boolean;
```



```
;
fpar
in legId LegType,
in maxEvent EventTypeBCSM;
```

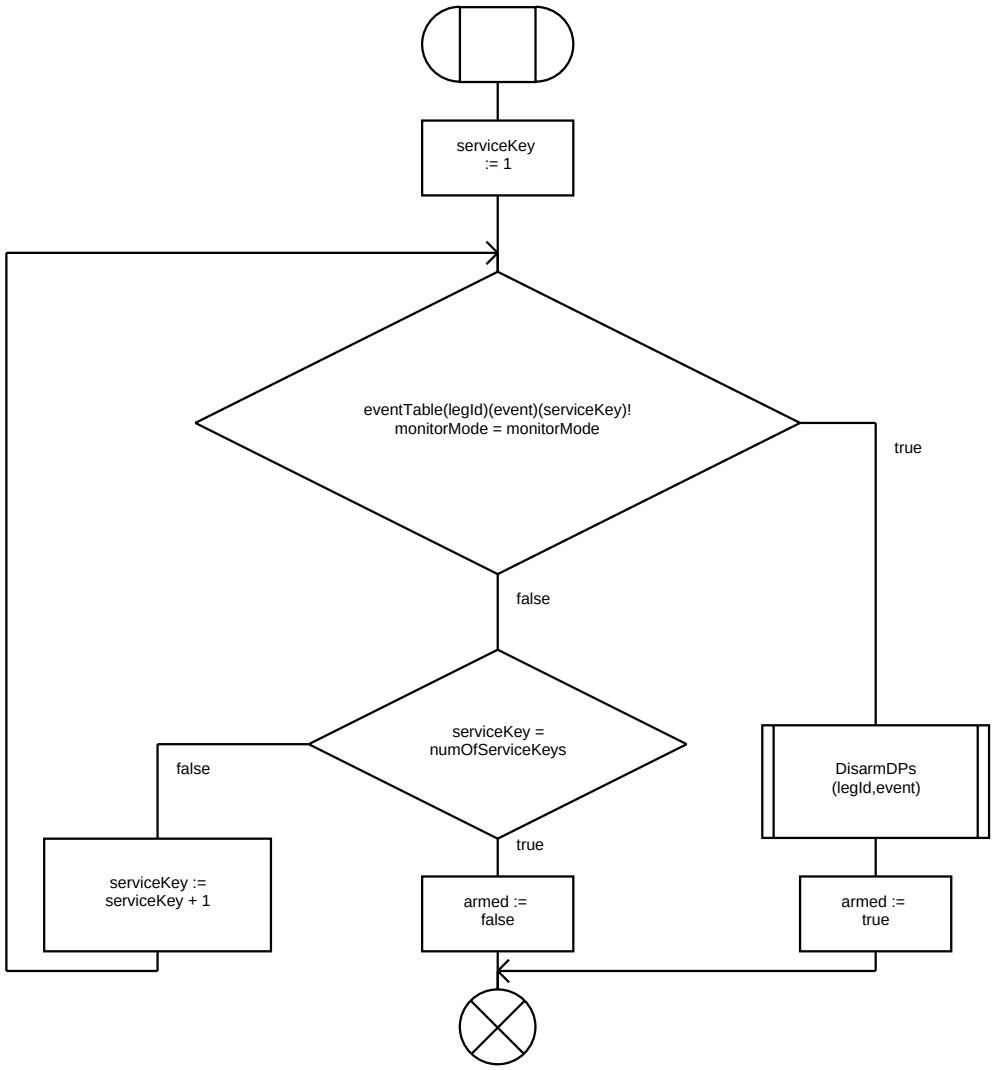
```
DCL

event EventTypeBCSM := origAttemptAuthorized,
serviceKey ServiceKey := 1;
```



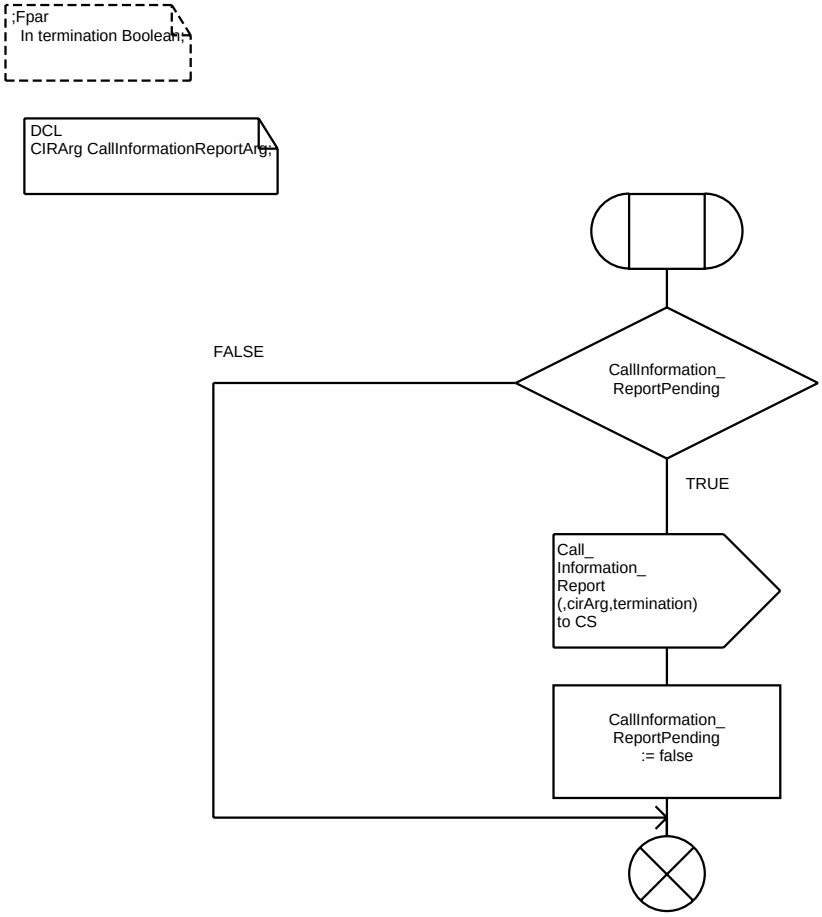
Procedure DPArmed

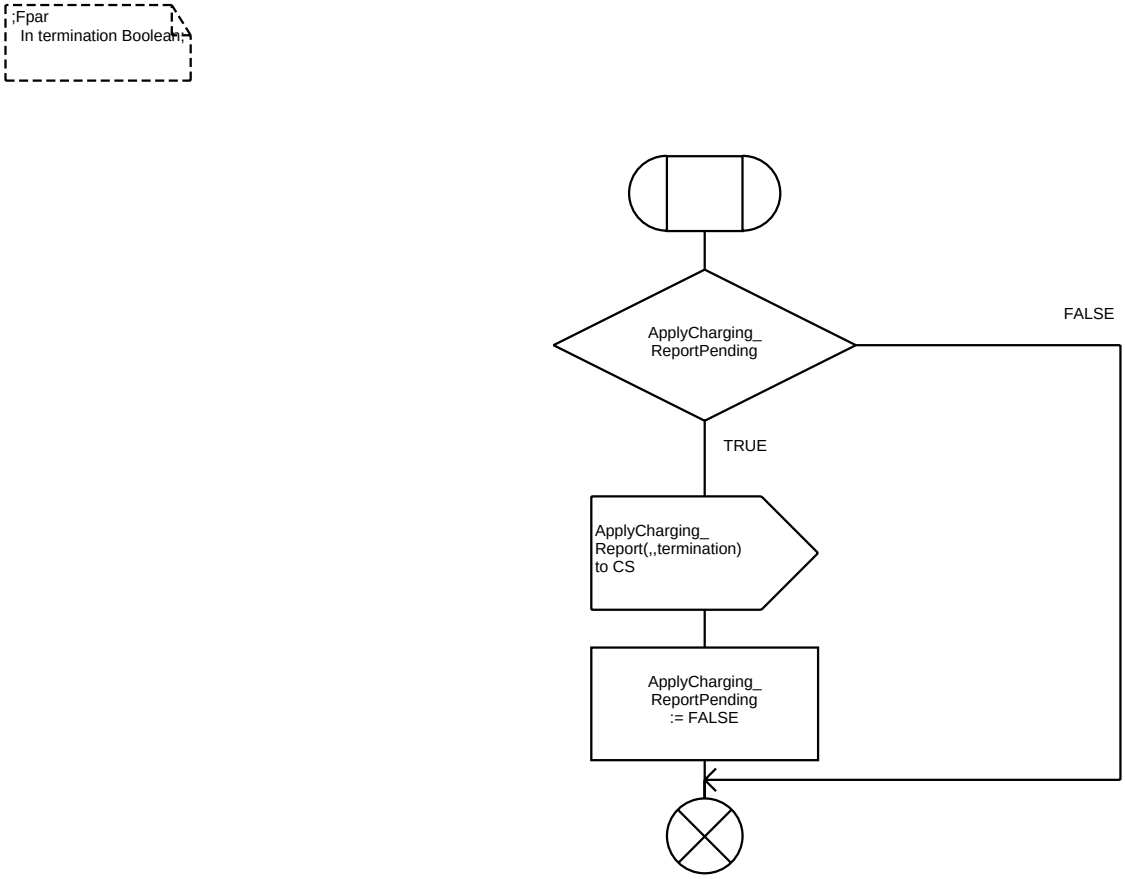

```
;
fpar
in legId LegType,
in event EventTypeBCSM,
in monitorMode MonitorMode;
returns armed Boolean
;
```

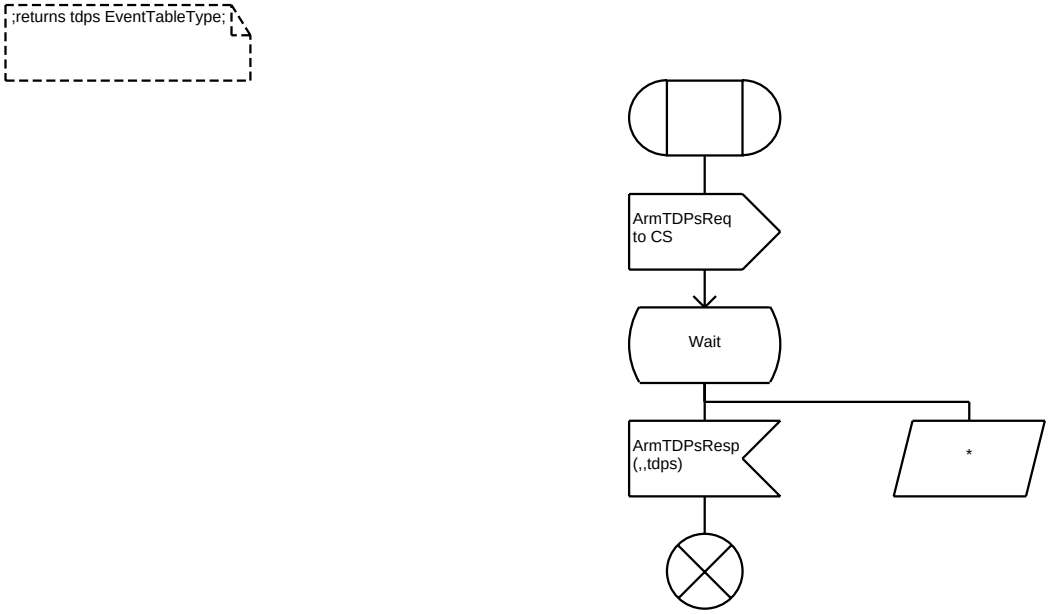
```
/*
The procedure searches for the first
armed instance (i.e. first armed
Service Key) of the DP. If an armed
instance is encountered the instance
is disarmed and the procedure returns
true, else the procedure returns false.
*/
```



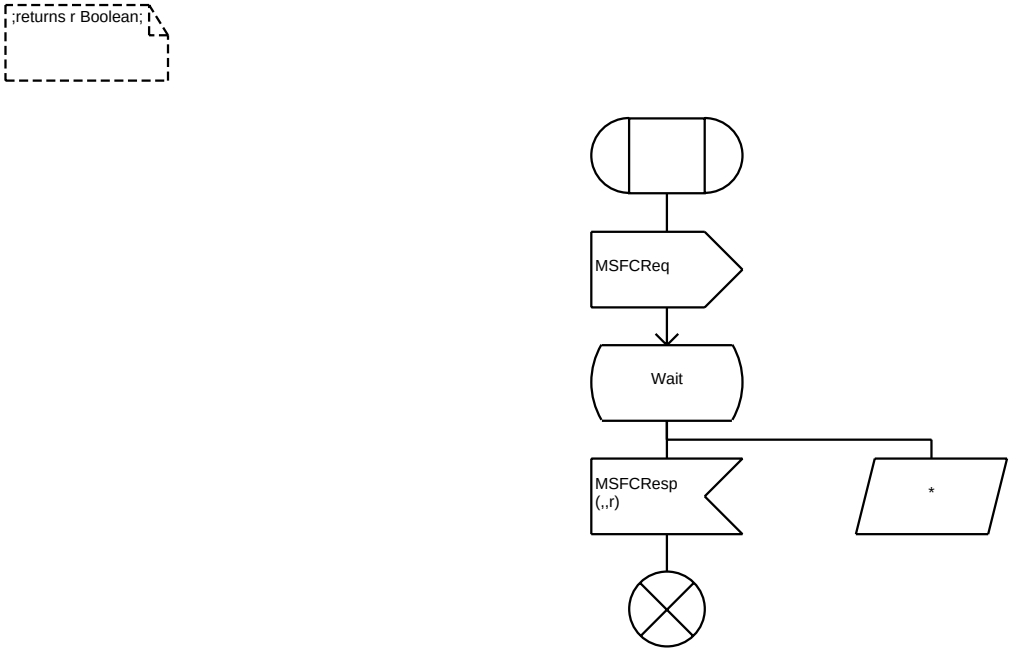
Procedure CallInformationReportPending

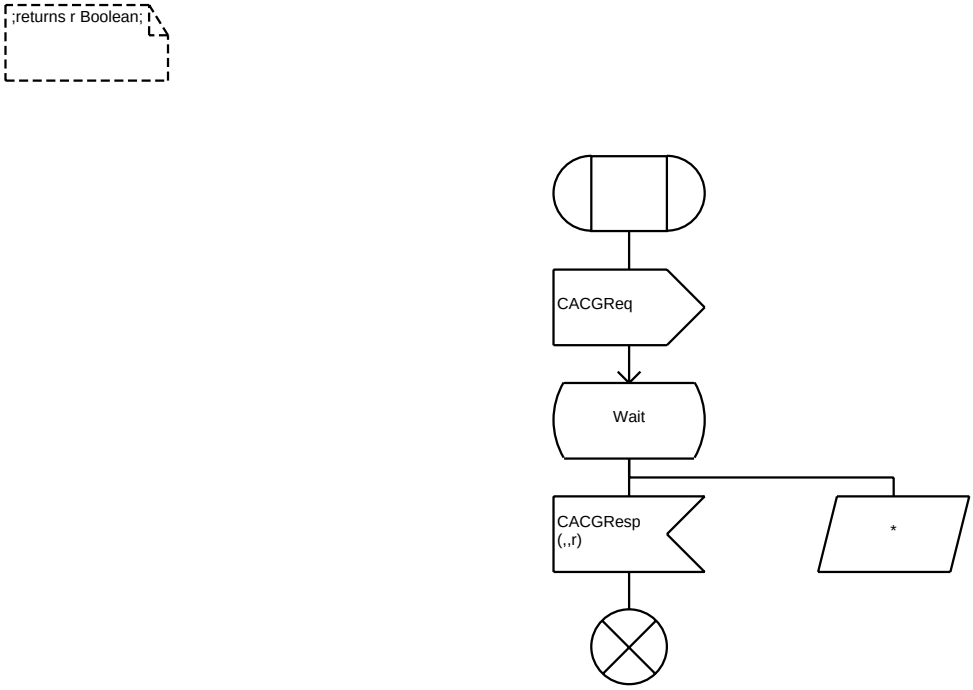


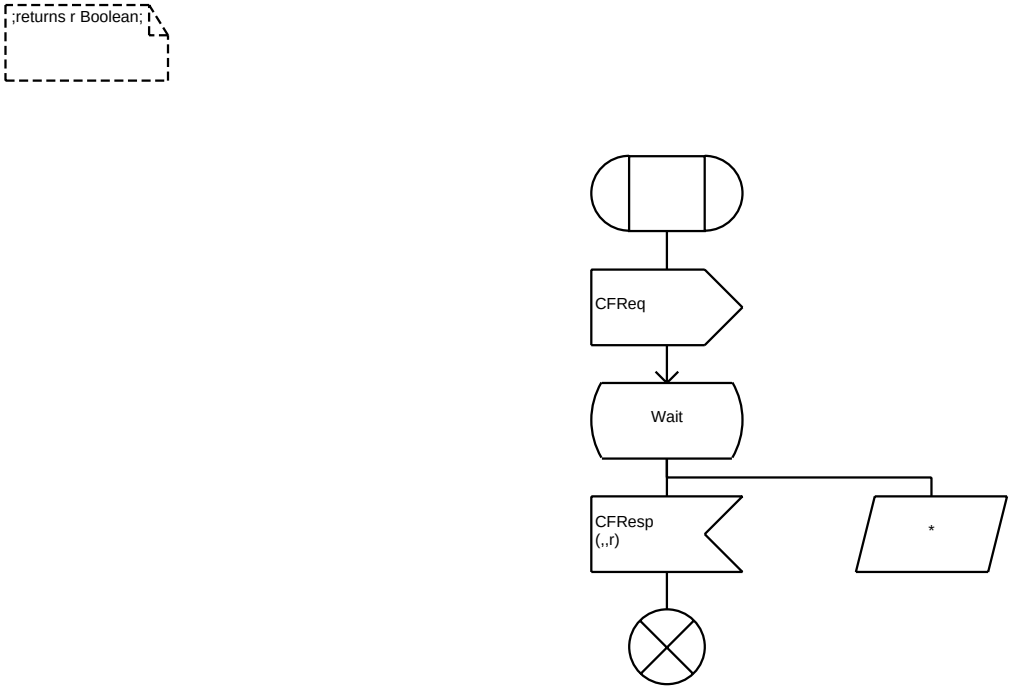

;Fpar
In termination Boolean;
ApplyCharging_
ReportPending
FALSE
TRUE
ApplyCharging_
Report(,,termination)
to CS
ApplyCharging_
ReportPending
:= FALSE

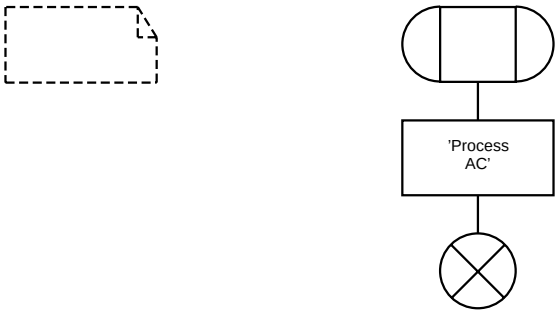

;returns tdps EventTableType;
ArmTDPsReq
to CS
Wait
ArmTDPsResp
(,,tdps)
*

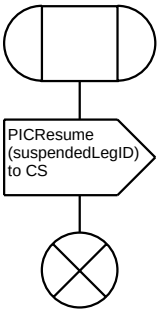
Procedure <<Process Type SSF_FSM>> MatchingServiceFilteringCriteria 1(1)




;returns r Boolean;
CACGReq
Wait
CACGResp
(,,r)
*


;returns r Boolean;
CFReq
Wait
CFResp
(,,r)
*


'Process
AC'


PICResume
(suspendedLegID)
to CS

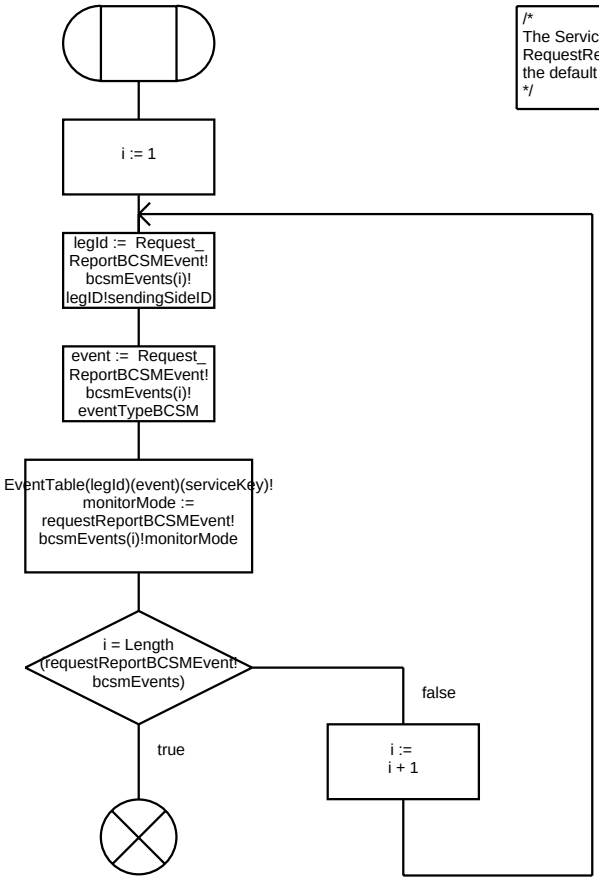
Procedure ProcessRequestReportBCSMEvent


```
;
FPAR
IN/OUT requestReportBCSMEvent RequestReportBCSMEventArg;
```

```
/*
The procedure assumes that the EventReportTypes
can be in a different order to the internal
EventTable (Reasonable assumption).
*/
```

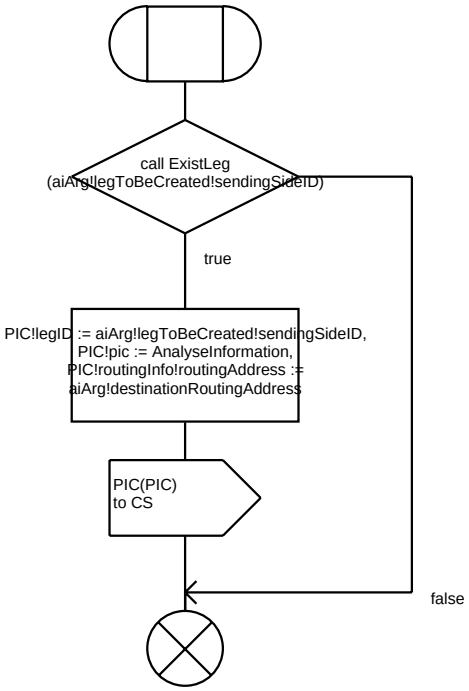
```
DCL

legId LegType,
event EventTypeBCSM,
serviceKey ServiceKey := 1,
i INTEGER;
```

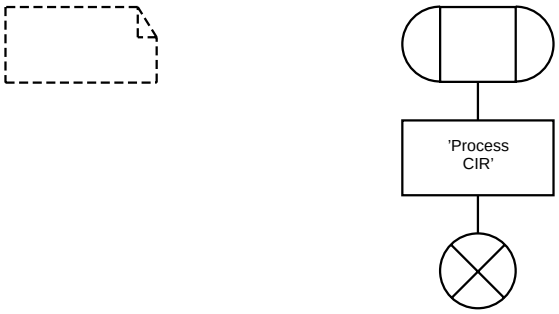
```
/*
The ServiceKey is not specified in the
RequestReportBCSMEvent operation. Therefore
the default value is used.
*/
```

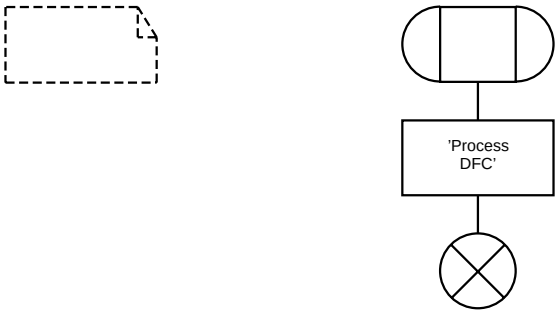


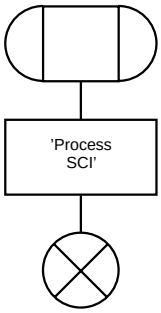
```
;
fpar
in aiArg AnalyseInformationArg;
```

```
/*
If the leg does not exist this implies
that a new BCSM has been created which
will begin processing at the AnalyseInformation PIC
PIC. The SSF does not need to resume processing.
*/
```



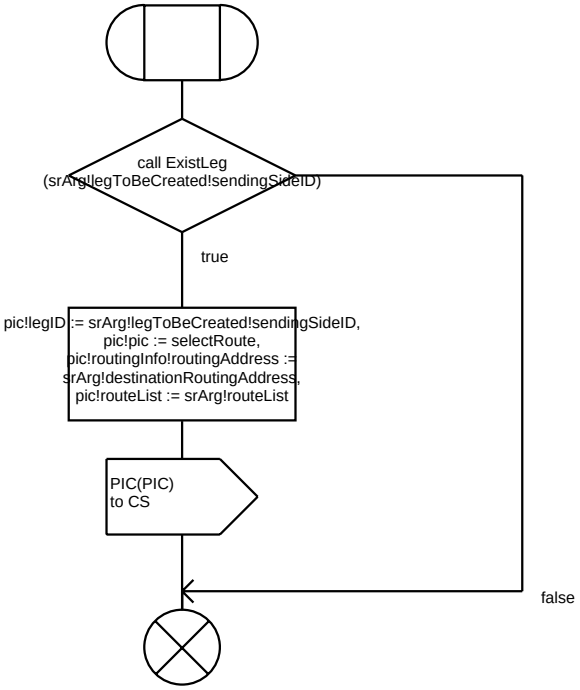

'Process
CIR'

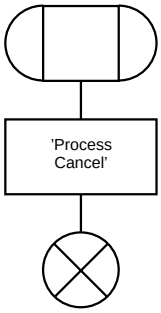

'Process
DFC'

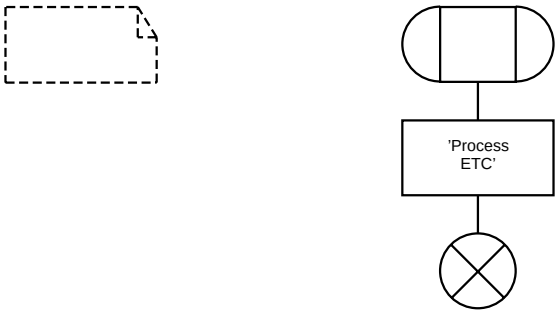

'Process
SCI'

```
;
fpar
in srArg SelectRouteArg;
```

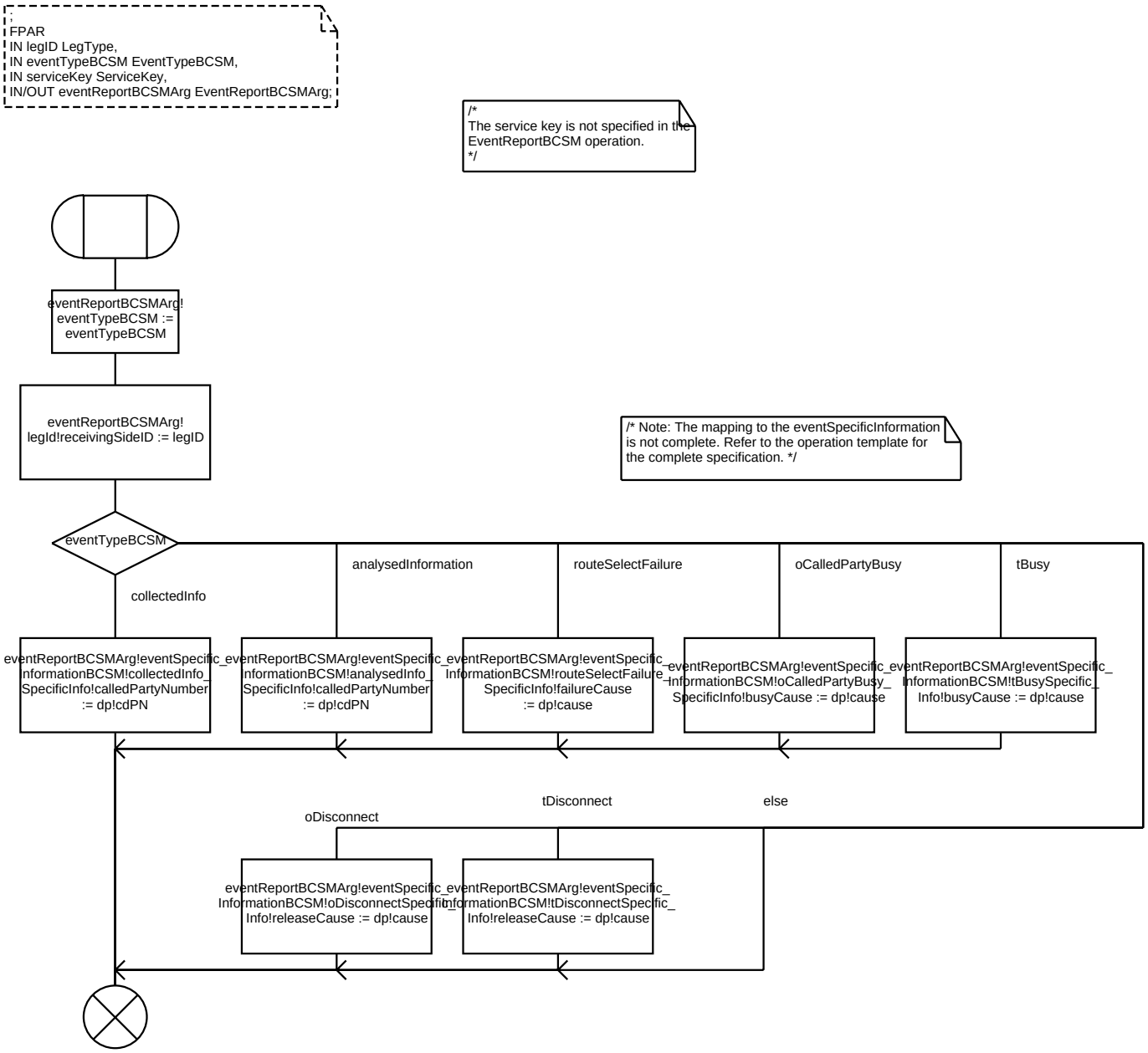
```
/*
If the leg does not exist this implies
that a new BCSM has been created which
will begin processing at the SelectRoute
PIC. The SSF does not need to resume processing.
*/
```




'Process
Cancel'

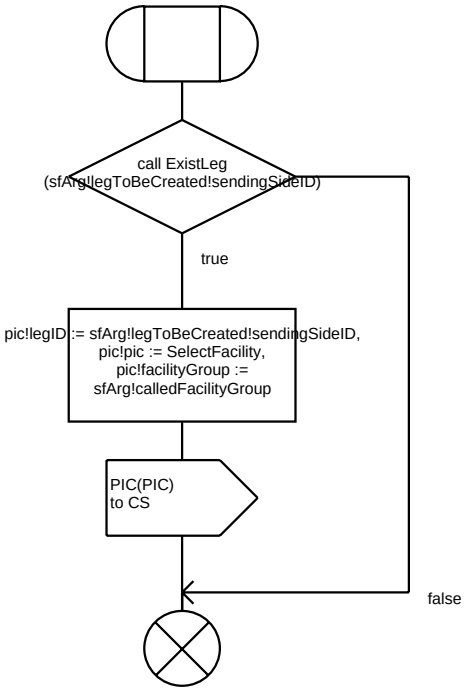

'Process
ETC'

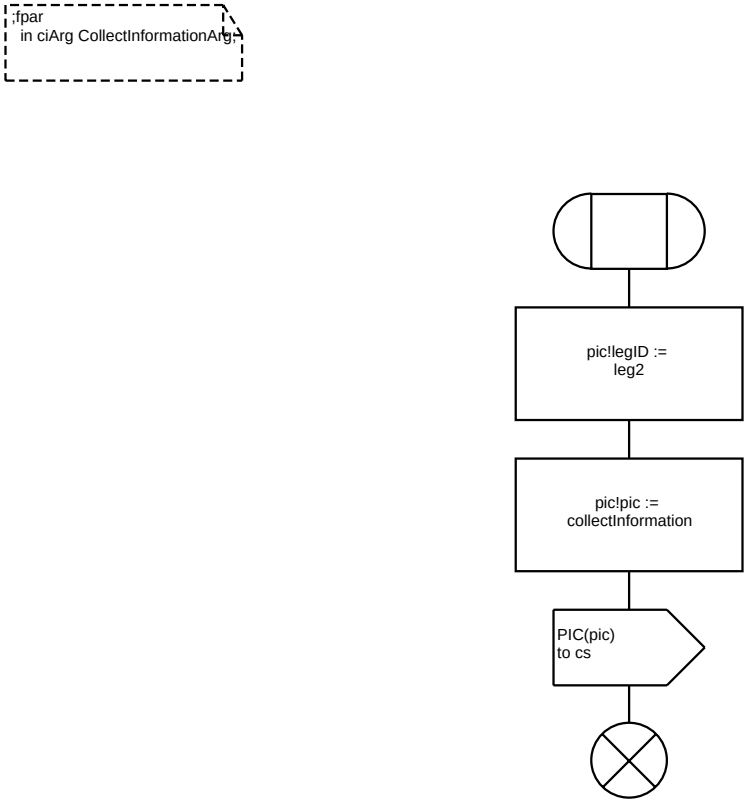
Procedure ProcessEventReportBCSM

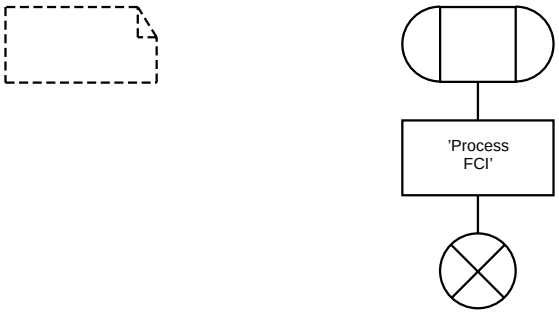


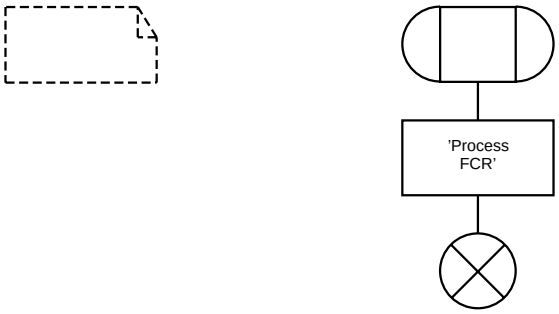
```
;
fpar
in sfArg SelectFacilityArg;
```

```
/*
If the leg does not exist this implies
that a new O−BCSM has been created which
will begin processing at the ONull
PIC. The SSF does not need to resume processing.
*/
```



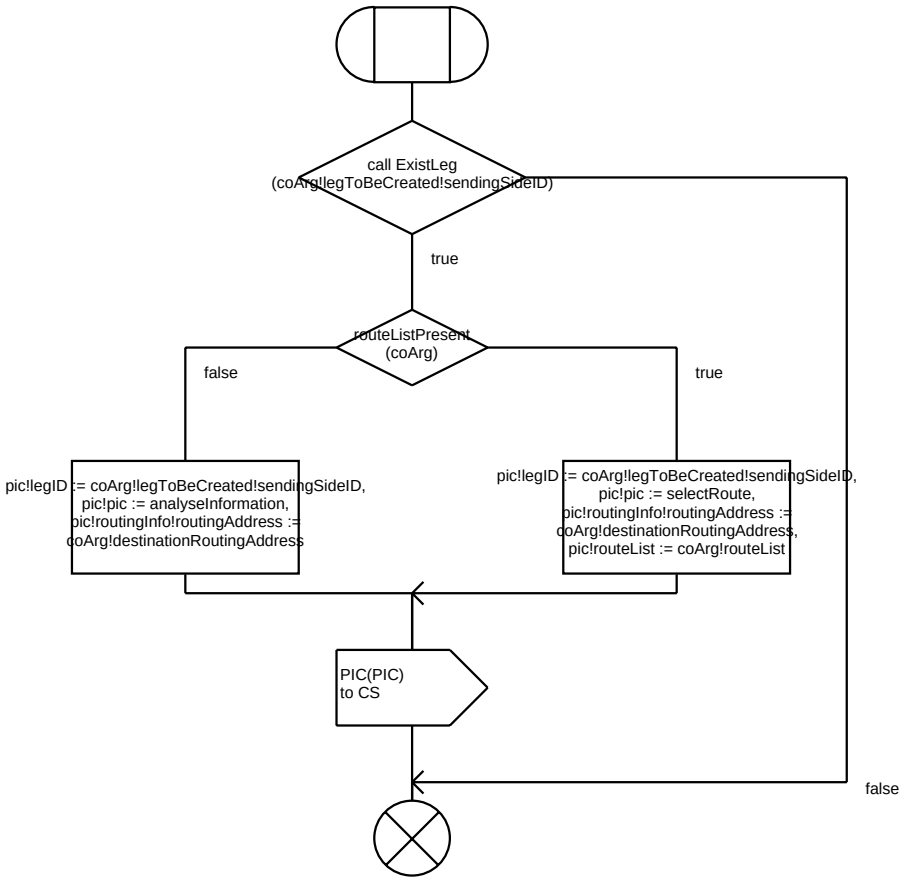

;fpar
in ciArg CollectInformationArg,
pic!legID :=
leg2
pic!pic :=
collectInformation
PIC(pic)
to cs


'Process
FCI'

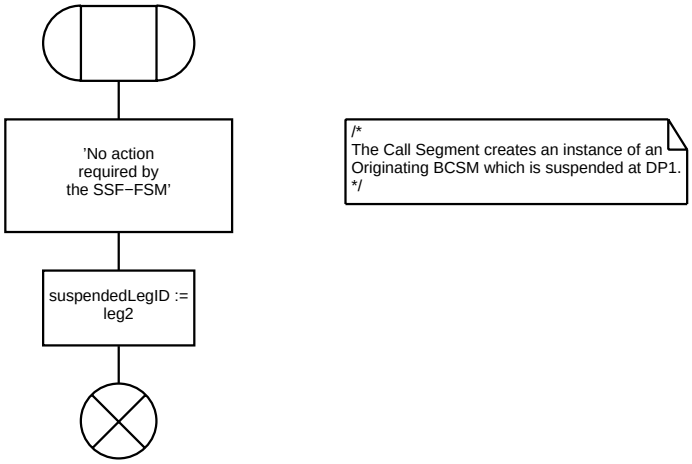

'Process
FCR'

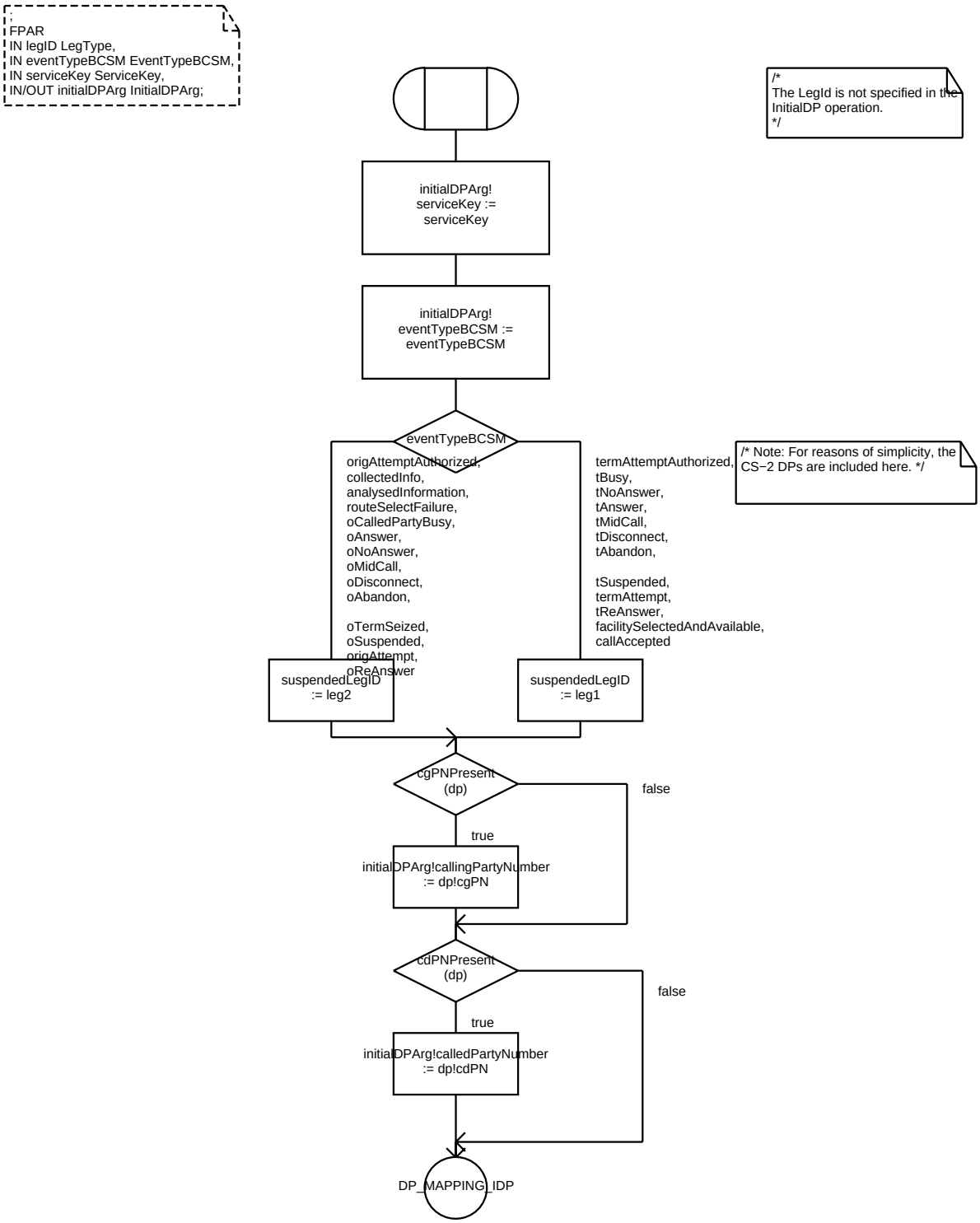
```
;
fpar
in coArg ConnectArg;
```

```
/*
If the leg does not exist this implies
that a new BCSM has been created which
will begin processing at the AnalyseInformation
PIC. The SSF does not need to resume processing.
*/
```

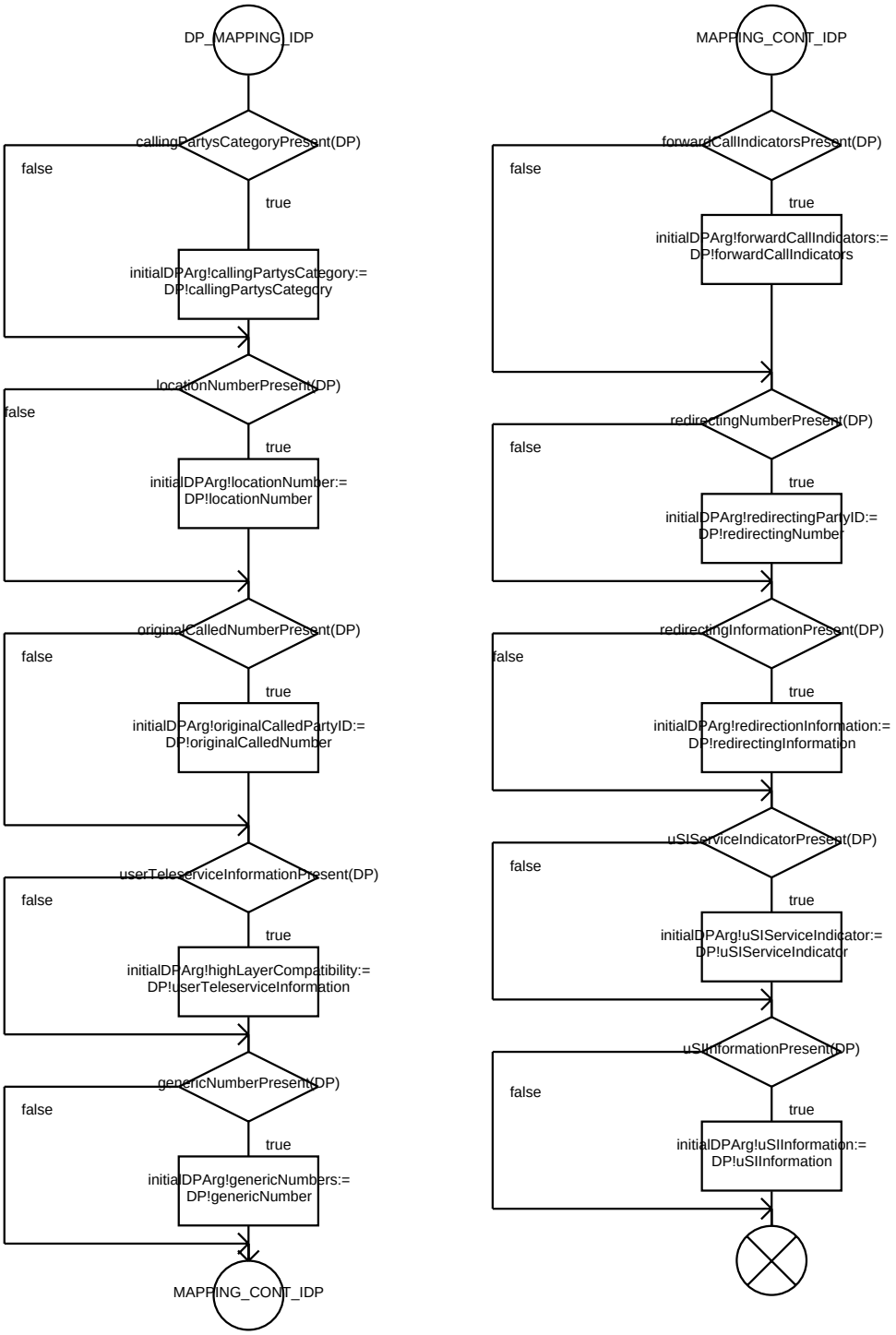


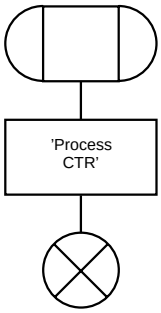
```
;
fpar
in ICAArg InitiateCallAttemptArg;
```

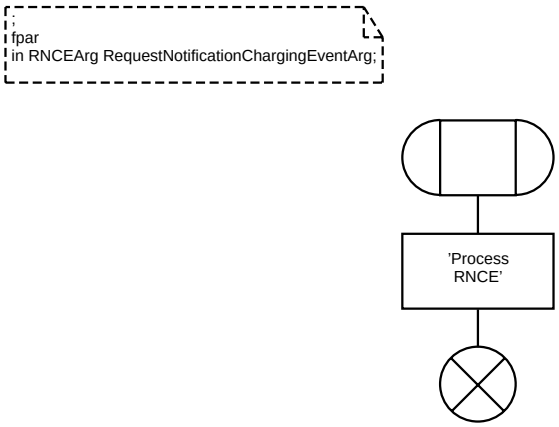



;
FPAR
IN legID LegType,
IN eventTypeBCSM EventTypeBCSM,
IN serviceKey ServiceKey,
IN/OUT initialDPArg InitialDPArg;
/*
The LegId is not specified in the InitialDP operation.
*/
initialDPArg!
serviceKey :=
serviceKey
initialDPArg!
eventTypeBCSM :=
eventTypeBCSM
eventTypeBCSM
origAttemptAuthorized,
collectedInfo,
analysedInformation,
routeSelectFailure,
oCalledPartyBusy,
oAnswer,
oNoAnswer,
oMidCall,
oDisconnect,
oAbandon,
oTermSeized,
oSuspended,
origAttempt,
oReAnswer
termAttemptAuthorized,
tBusy,
tNoAnswer,
tAnswer,
tMidCall,
tDisconnect,
tAbandon,
tSuspended,
termAttempt,
tReAnswer,
facilitySelectedAndAvailable,
callAccepted
/* Note: For reasons of simplicity, the CS−2 DPs are included here. */
suspendedLegID
:= leg2
suspendedLegID
:= leg1
cgPNPresent
(dp)
false
true
initialDPArg!callingPartyNumber
:= dp!cgPN
cdPNPresent
(dp)
false
true
initialDPArg!calledPartyNumber
:= dp!cdPN
DP_MAPPING_IDP

```
;
FPAR
IN legID LegType,
IN eventTypeBCSM EventTypeBCSM,
IN serviceKey ServiceKey,
IN/OUT initialDPArg InitialDPArg;
```




'Process
CTR'

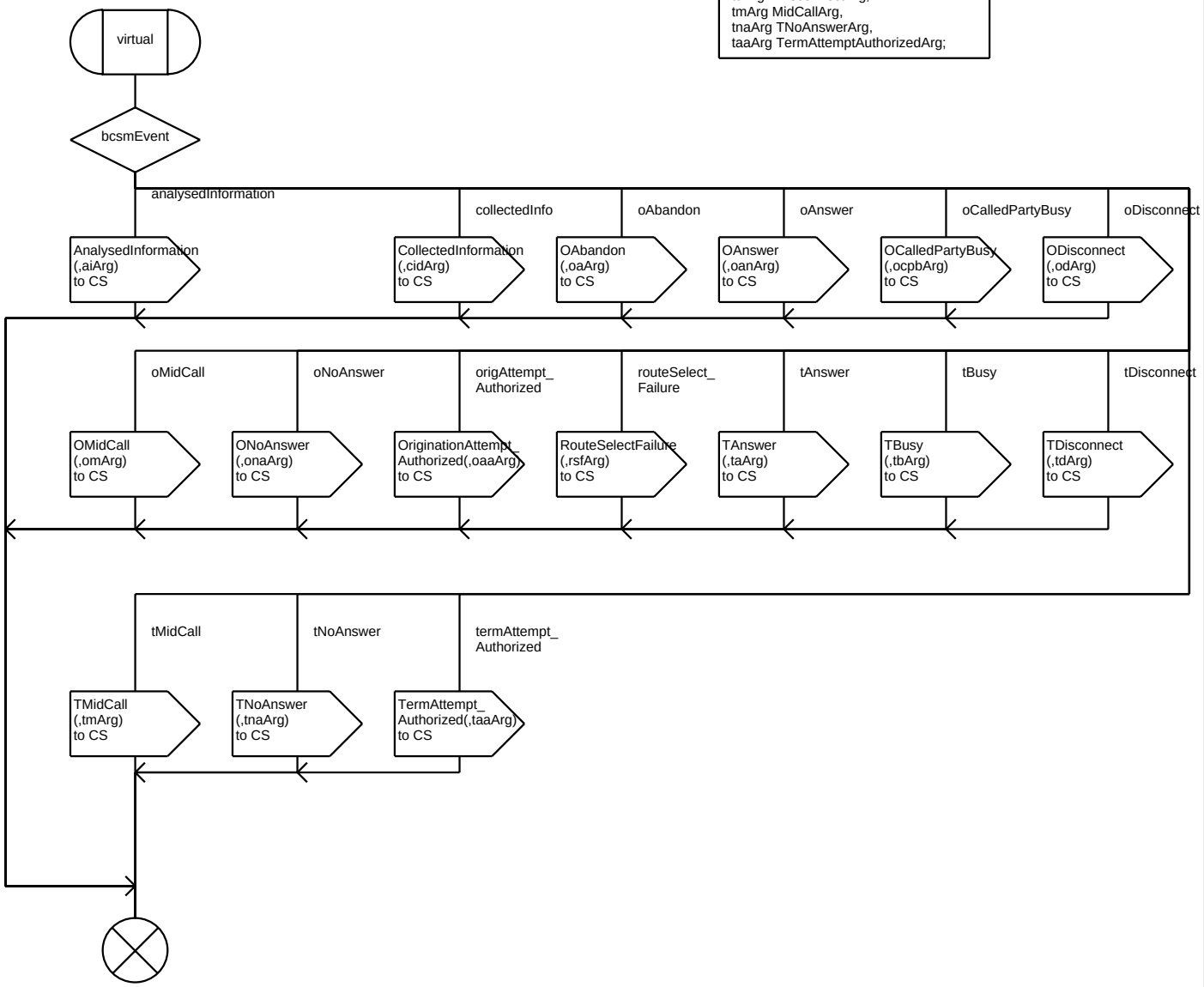

;
fpar
in RNCEArg RequestNotificationChargingEventArg;
'Process
RNCE'

Virtual Procedure ProcessDPSpecific


```
;FPAR
  IN legID LegType,
  IN bcsmEvent EventTypeBCSM,
  IN serviceKey ServiceKey;
```

```
/* Note: The operation arguments
are assigned as defined in Q.1228
section 17. */
```

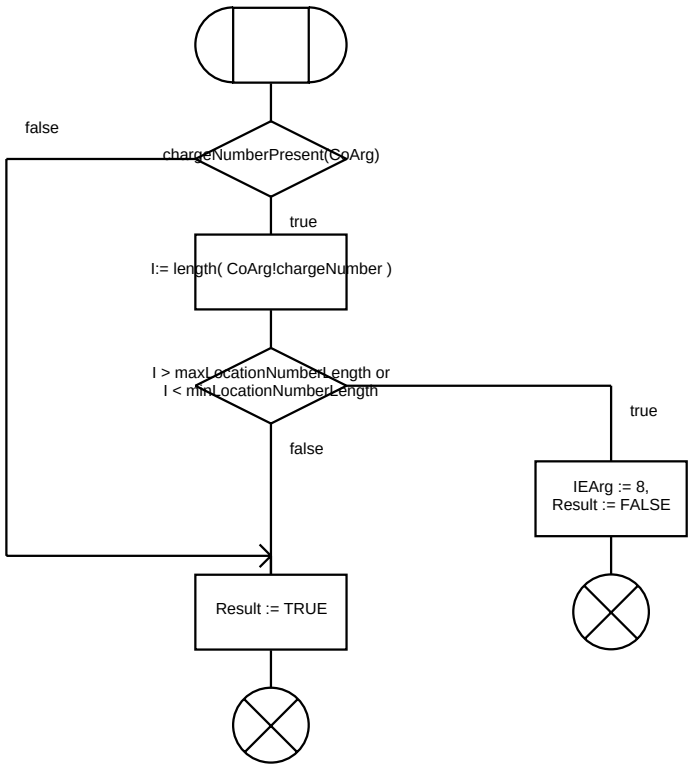
```
DCL
  aiArg AnalysedInformationArg,
  cidArg CollectedInformationArg,
  oaArg OAbandonArg,
  oanArg OanswerArg,
  ocpbArg OCalledPartyBusyArg,
  odArg ODisconnectArg,
  omArg MidCallArg,
  onaArg ONoAnswerArg,
  oaaArg OriginationAttemptAuthorizedArg,
  rsfArg RouteSelectFailureArg,
  taArg TAnswerArg,
  tbArg TBusyArg,
  tdArg TDisconnectArg,
  tmArg MidCallArg,
  tnaArg TNoAnswerArg,
  taaArg TermAttemptAuthorizedArg;
```



Procedure ConnnectAnalysis


```
;FPAR
   IN CoArg ConnectArg;
   RETURNS Result BOOLEAN;
```

```
DCL
I INTEGER;
```

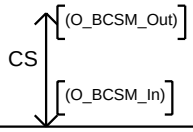


```
;FPAR  obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
```

```
/***** O−BCSM FOR CORE INAP CS−1. *****/
```

```
/***** DATA TYPE DEFINITIONS *****/
```

```
NEWTYPE DPResultType
  LITERALS
    Resume,
    CollectInformation,
    AnalyseInformation,
    SelectRoute,
    Exception,
    ARelease,
    BRelease,
    SCFRelease, /* Used with ReleaseCall */
    Answer,
    Alerting,

    Reconnect,
    DL_A, DL_B, /* Used with DisconnectLeg */
    CS_Stop
ENDNEWTYPE;

NEWTYPE PICResultType
  LITERALS
    Success,
    Failure,
    Select_next_Route,
    Analyse_marked_CDPN,
    Alerting,
    Answer,
    NoAnswer,
    Busy,
    MidCall,
    Exception,
    ARelease,
    BRelease,
    SCFRelease, /* Used with ReleaseCall */

    Reconnect,
    Reanswer,
    DL_A, DL_B, /* Used with DisconnectLeg */
    Suspended
ENDNEWTYPE;
```




```
;FPAR  obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
```

```
/**** VARIABLE AND TIMER DECLARATIONS ****/
```

```
TIMER
  NoAnswerT := 3600000; /* Value for simulation purposes. */

DCL
  /* Pointer to the call segment. */
  CS PId,

  /* SigCon ID of the signalling entity associated with this BCSM. */
  SigConId callRef,


  /* Used to indicate the status of the call processing. */
  alertingSent boolean := false, /* Indicates if CallProgress(bptyAlerted) sent to the calling party. */
  setupSent boolean := false,    /* Indicates if SetupReqInd is sent to the called party, i.e. is the T−BCSM created. */

  /* Call info */
  cdPNIx INTEGER := 1,                    /* Index to marked cdPN */
  numOfCDPNs INTEGER := 1,                /* Number of CDPNs in the routingAddress. */
  routingAddress DestinationRoutingAddress, /* List of CDPNs */
  routeIx INTEGER := 1,                   /* Index to selected route */

  /* Other variables. */
  PICResult PICResultType,
  DPResult DPResultType,
  cause Cause,
  PIC  PICArg,
  DP DPArg,

  /* Variables for simulation purposes. */
  Enbloc Boolean := true,

  digits Digits;
```

```
DCL
  /* SigCon primitive parameters. */
  AEArg AddressEndType,
  CPArg CallProgressType,
  FArg FailureType,
  RArg ReleaseType,
  SFtArg ServiceFeatureType,
  SIRArg SetupIRType,
  SCRArg SetupCRType,
  SAArg SubsequentAddressType;
```

;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */

/* Procedure definitions for Points In Call (PIC). */

PIC_O_Null

PIC_Analyse_
_Information

virtual
PIC_Send_
_Call

virtual
PIC_O_
_Active

PIC_Authorise_
Origination
_Attempt

PIC_Select_
_Route

virtual
PIC_O_
_Alerting

PIC_O_
_Abandon

PIC_Collect_
_Information

PIC_Authorise_
_Call_Setup

PIC_O_Answer

PIC_
_OException

PIC_Collect_
_NDigits

/* Procedure definitions for Detection Points (DPs) */

virtual
DP_origAttempt_
Authorised

virtual
DP_Route_
_Select_Failure

virtual
DP_oCalled_
PartyBusy

virtual
DP_
_oDisconnect

virtual
DP_Collected_
_Info

virtual
DP_oAnswer

virtual
DP_oMidCall

virtual
DP_Analysed_
_Information

virtual
DP_oNoAnswer

virtual
DP_oAbandon

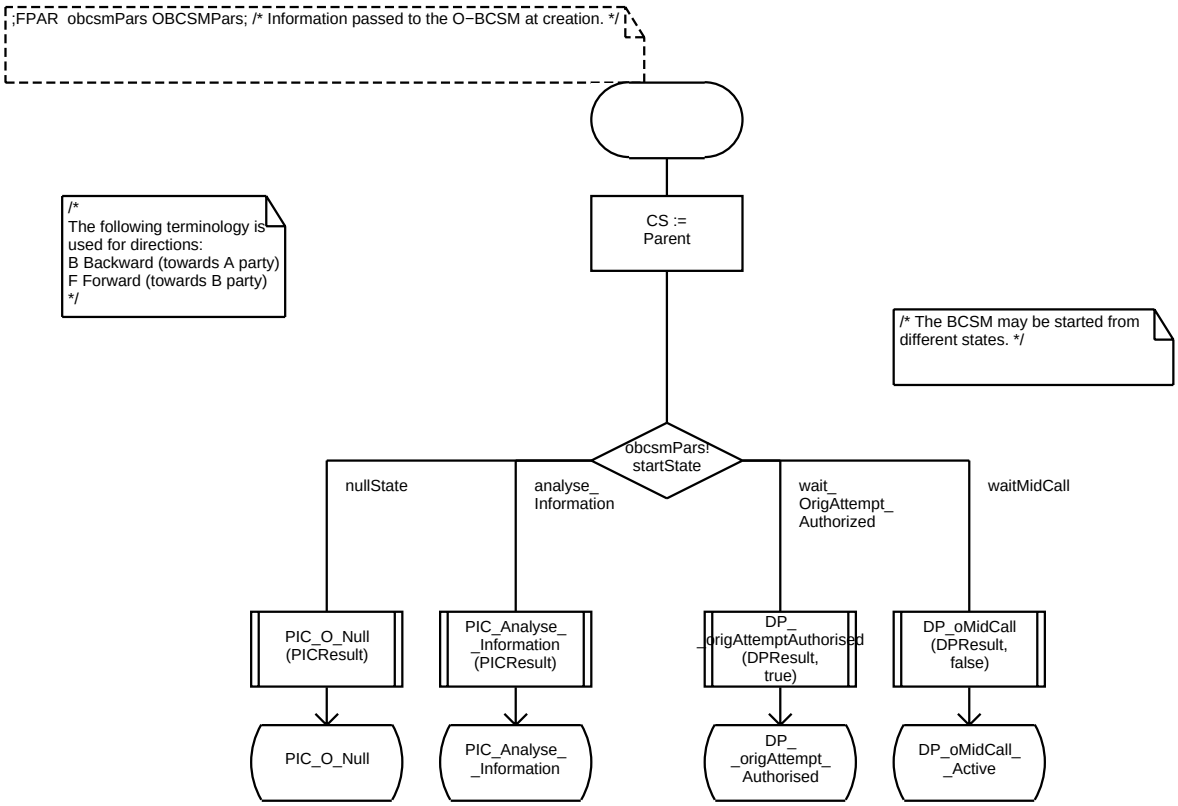
;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */

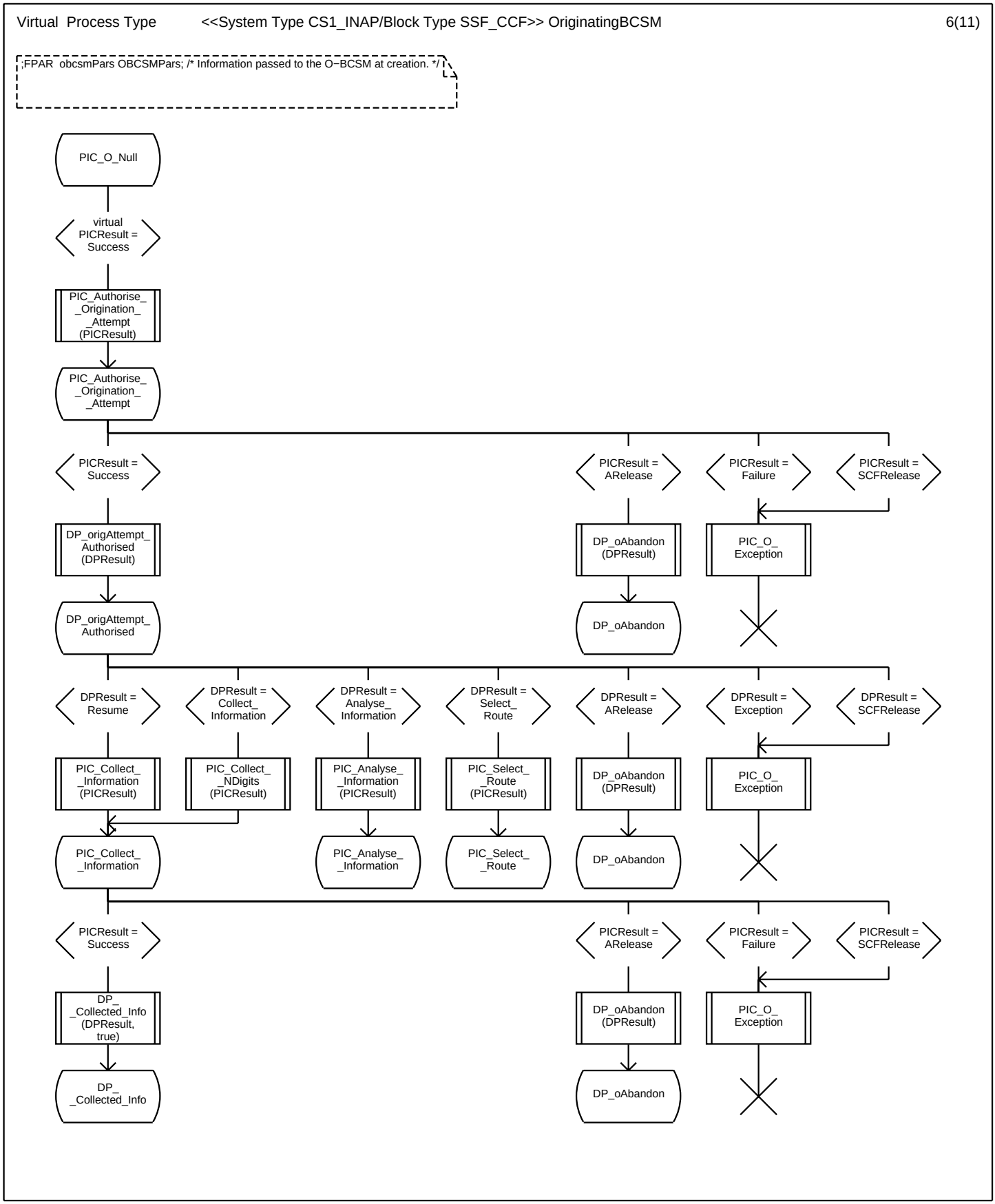
/* Procedures for mapping of parameters */

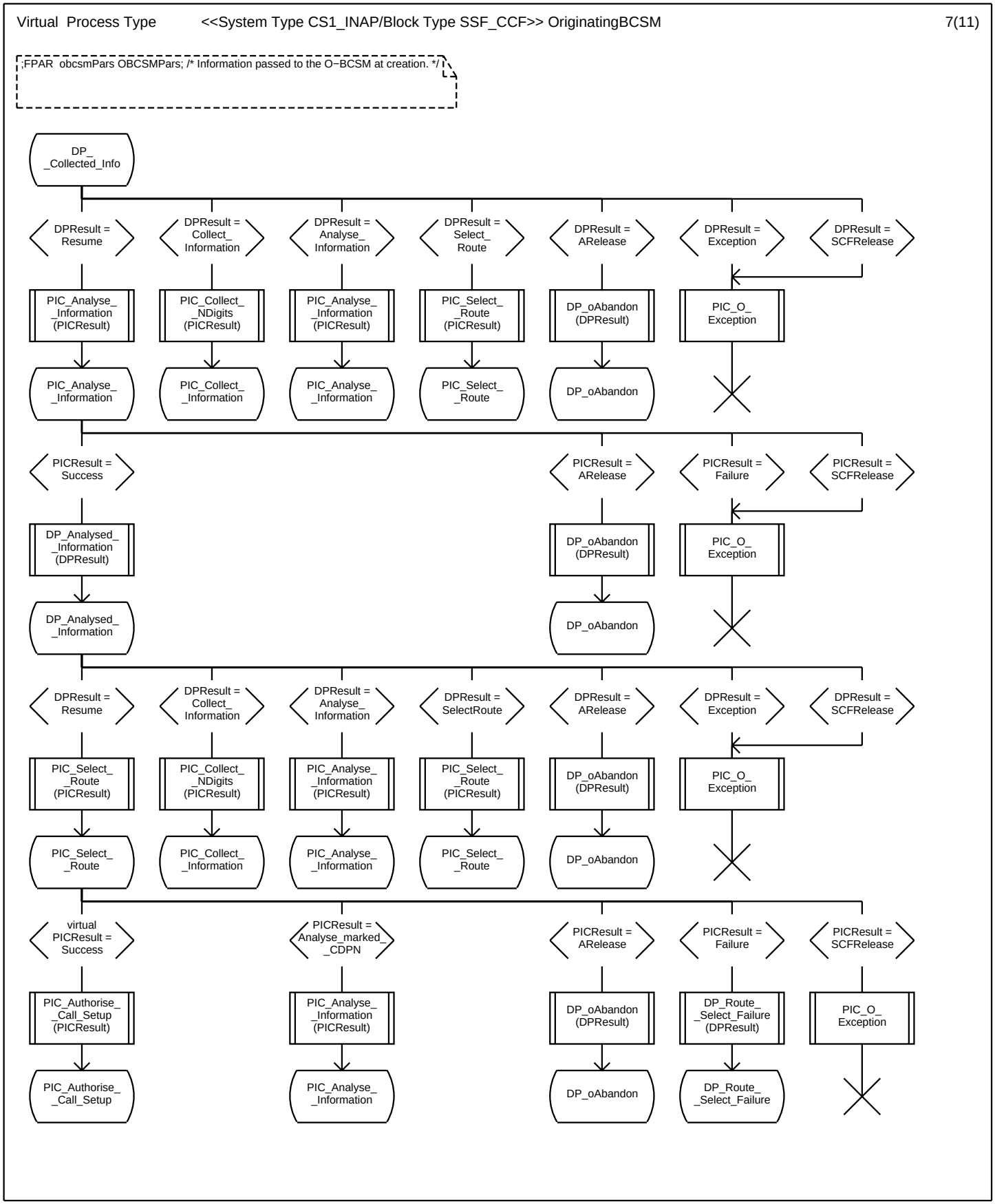
MapTo_
SIRArg

MapToDP

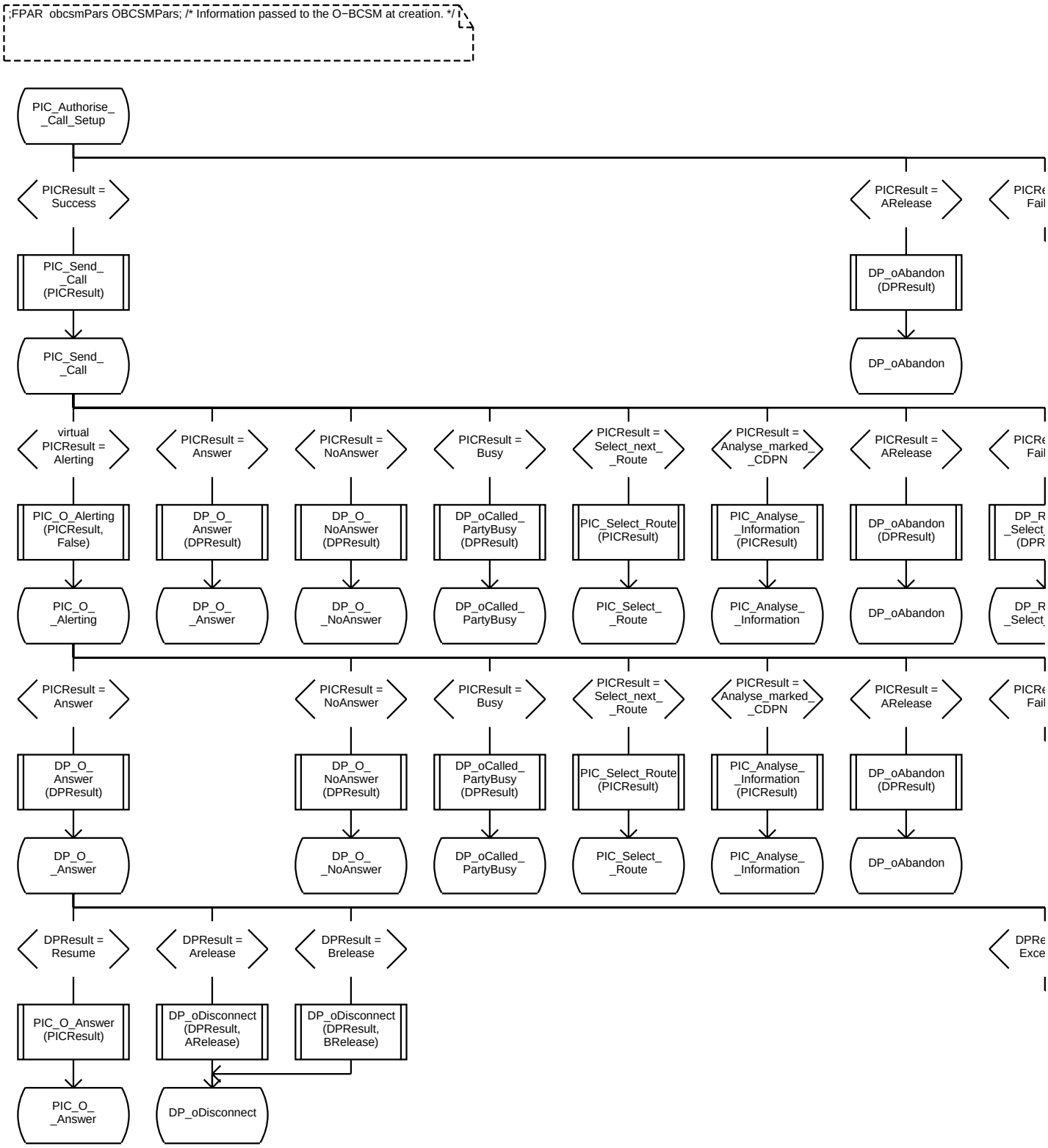
MapFromPIC

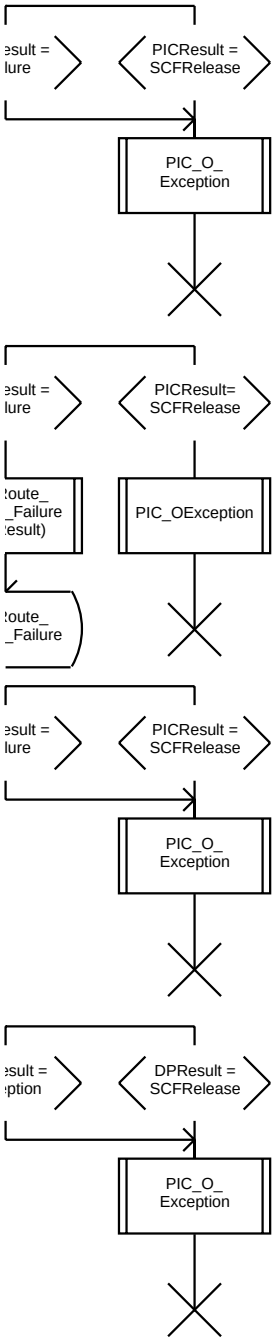

;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
/*
The following terminology is used for directions:
B Backward (towards A party)
F Forward (towards B party)
*/
CS :=
Parent
/* The BCSM may be started from different states. */
obcsmPars!
startState
nullState
analyse_
Information
wait_
OrigAttempt_
Authorized
waitMidCall
PIC_O_Null
(PICResult)
PIC_Analyse_
_Information
(PICResult)
DP_
_origAttemptAuthorised
(DPResult,
true)
DP_oMidCall
(DPResult,
false)
PIC_O_Null
PIC_Analyse_
_Information
DP_
origAttempt
Authorised
DP_oMidCall_
_Active

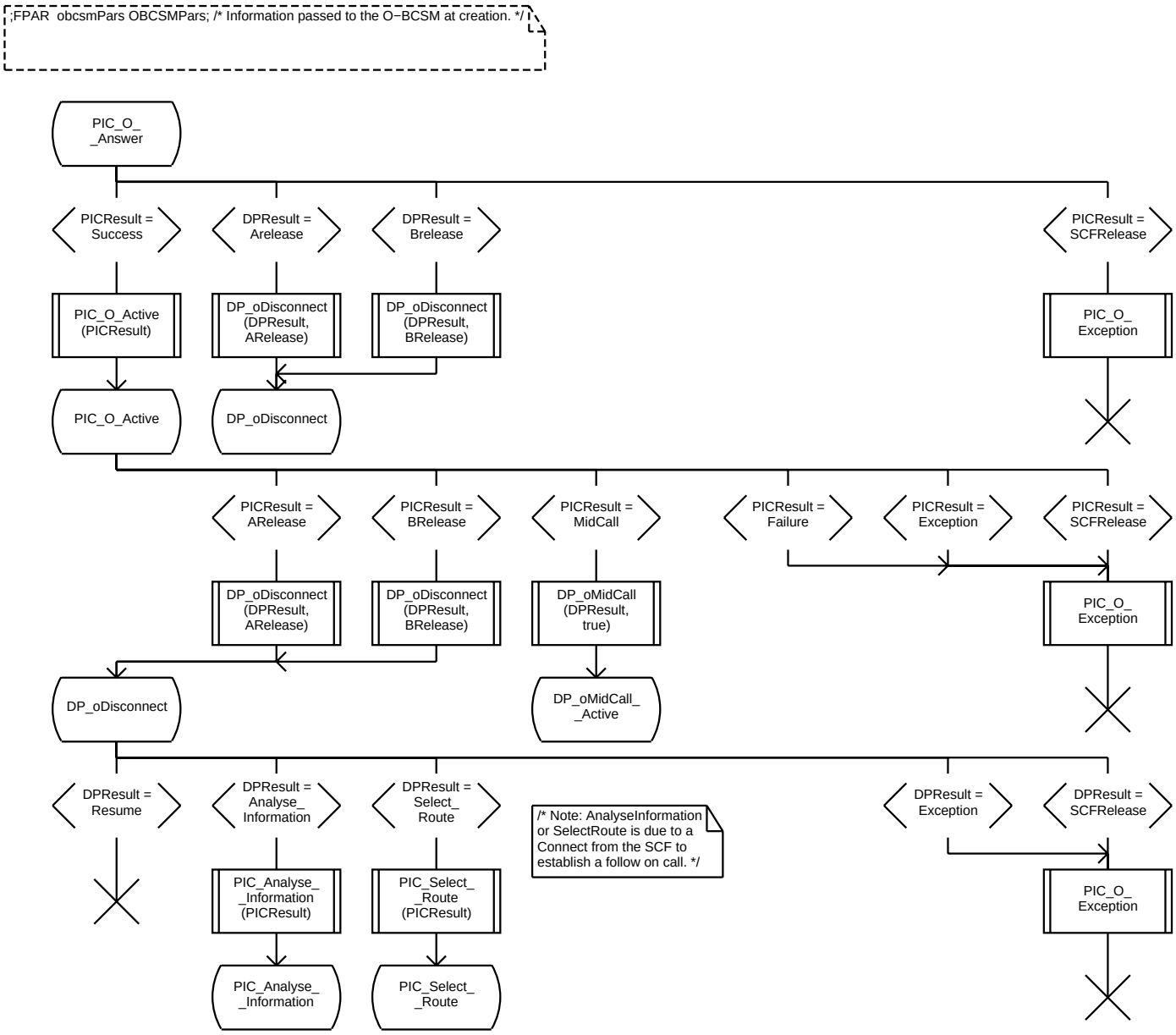

;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
PIC_O_Null
virtual PICResult = Success
PIC_Authorise_ _Origination_ _Attempt (PICResult)
PIC_Authorise_ _Origination_ _Attempt
PICResult = Success
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
DP_origAttempt_ Authorised (DPResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
DP_origAttempt_ Authorised
DP_oAbandon
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
PIC_Collect_ _Information (PICResult)
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
PICResult = Success
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
DP_ _Collected_Info (DPResult, true)
DP_oAbandon (DPResult)
PIC_O_ Exception
DP_ _Collected_Info
DP_oAbandon

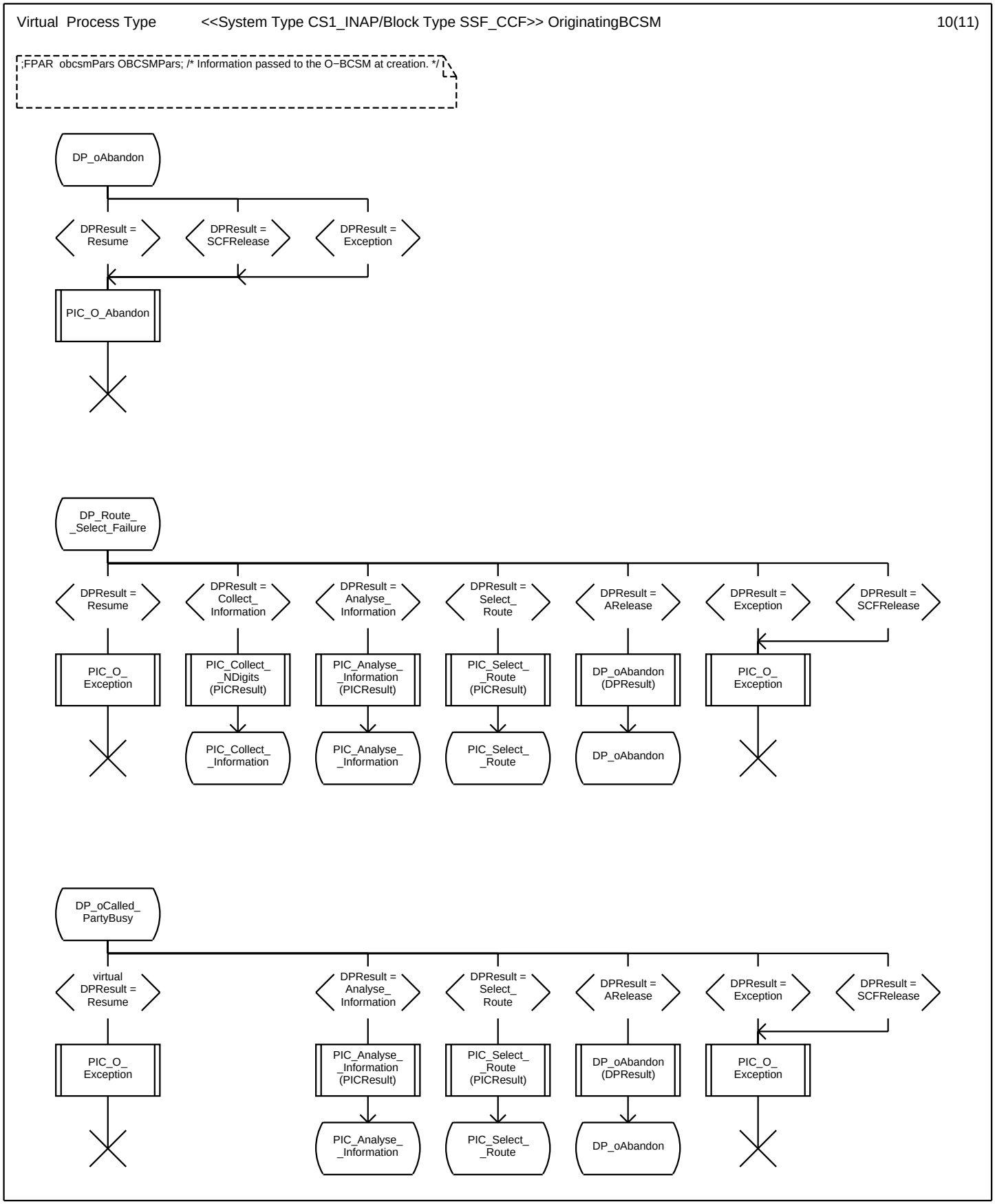

;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
DP_ _Collected_Info
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
PIC_Analyse_ _Information (PICResult)
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_Analyse_ _Information
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
PICResult = Success
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
DP_Analysed_ _Information (DPResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
DP_Analysed_ _Information
DP_oAbandon
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = SelectRoute
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
PIC_Select_ _Route (PICResult)
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_Select_ _Route
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
virtual PICResult = Success
PICResult = Analyse_marked_ _CDPN
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
PIC_Authorise_ _Call_Setup (PICResult)
PIC_Analyse_ _Information (PICResult)
DP_oAbandon (DPResult)
DP_Route_ _Select_Failure (DPResult)
PIC_O_ Exception
PIC_Authorise_ _Call_Setup
PIC_Analyse_ _Information
DP_oAbandon
DP_Route_ _Select_Failure

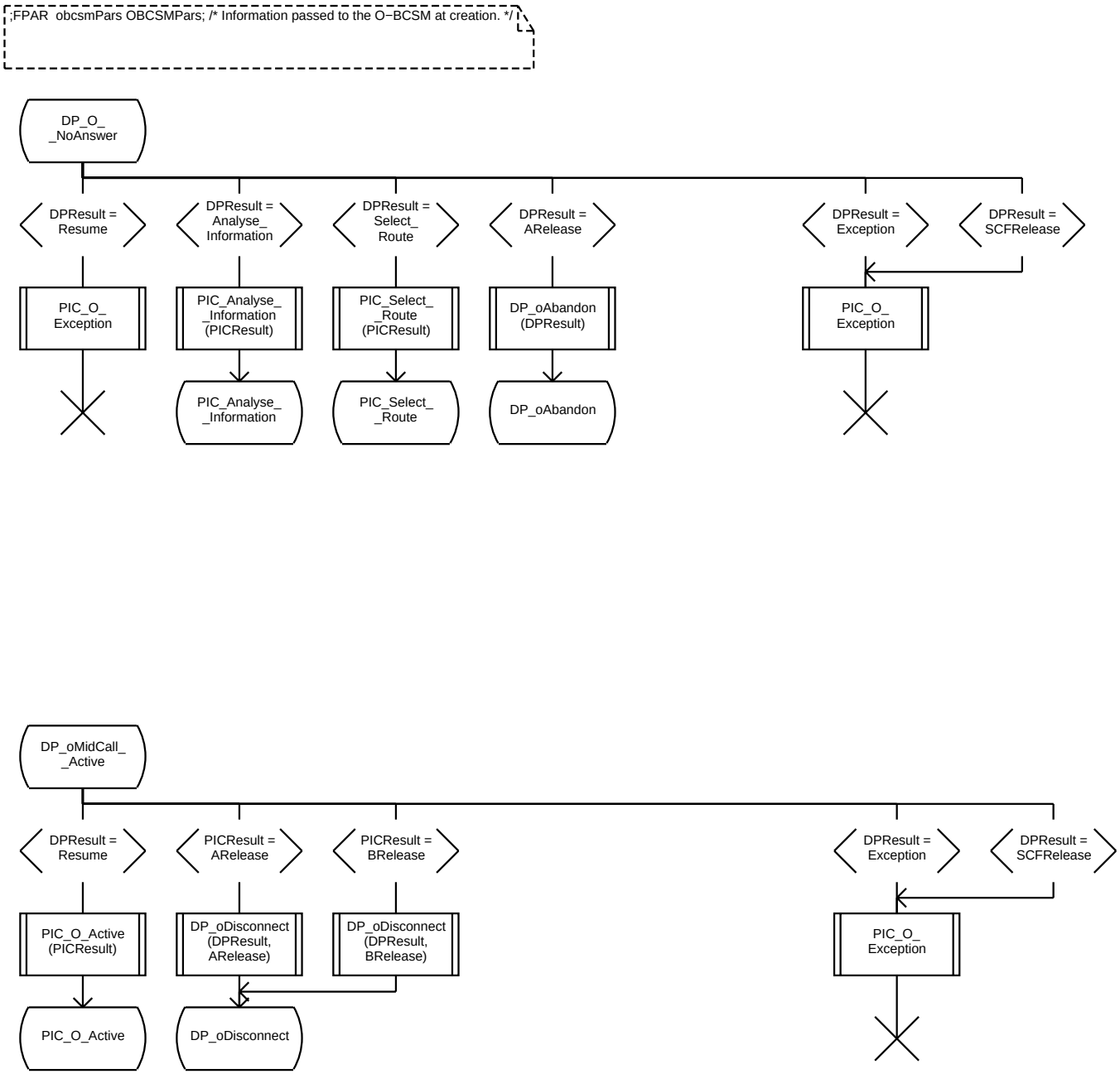
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingBCSM



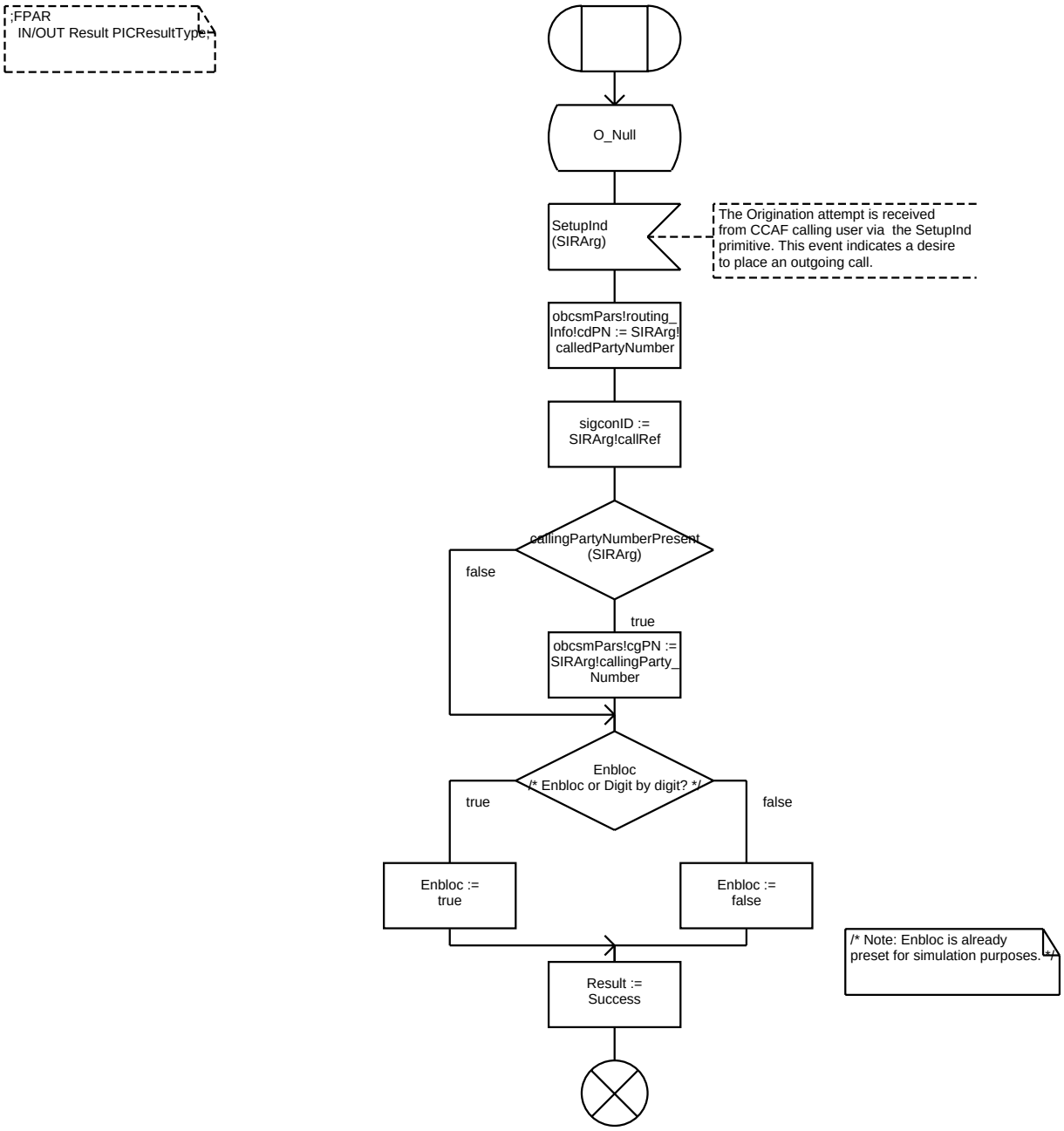

esult =
lure
PICResult =
SCFRelease
PIC_O_
Exception
esult =
lure
PICResult=
SCFRelease
:oute_
_Failure
:esult)
PIC_OException
:oute_
_Failure
esult =
lure
PICResult =
SCFRelease
PIC_O_
Exception
esult =
ption
DPResult =
SCFRelease
PIC_O_
Exception


;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
PIC_O_ _Answer
PICResult = Success
DPResult = Arelease
DPResult = Brelease
PICResult = SCFRelease
PIC_O_Active (PICResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
PIC_O_ Exception
PIC_O_Active
DP_oDisconnect
PICResult = ARelease
PICResult = BRelease
PICResult = MidCall
PICResult = Failure
PICResult = Exception
PICResult = SCFRelease
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
DP_oMidCall (DPResult, true)
PIC_O_ Exception
DP_oDisconnect
DP_oMidCall_ _Active
DPResult = Resume
DPResult = Analyse_ Information
DPResult = Select_ Route
/* Note: AnalyseInformation or SelectRoute is due to a Connect from the SCF to establish a follow on call. */
DPResult = Exception
DPResult = SCFRelease
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
PIC_O_ Exception
PIC_Analyse_ _Information
PIC_Select_ _Route

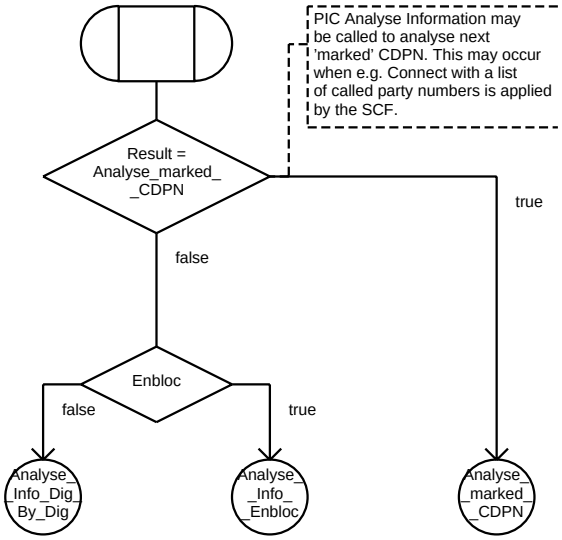

;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
DP_oAbandon
DPResult = Resume
DPResult = SCFRelease
DPResult = Exception
PIC_O_Abandon
DP_Route_ _Select_Failure
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
PIC_O_ Exception
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
DP_oCalled_ PartyBusy
virtual DPResult = Resume
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
PIC_O_ Exception
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon

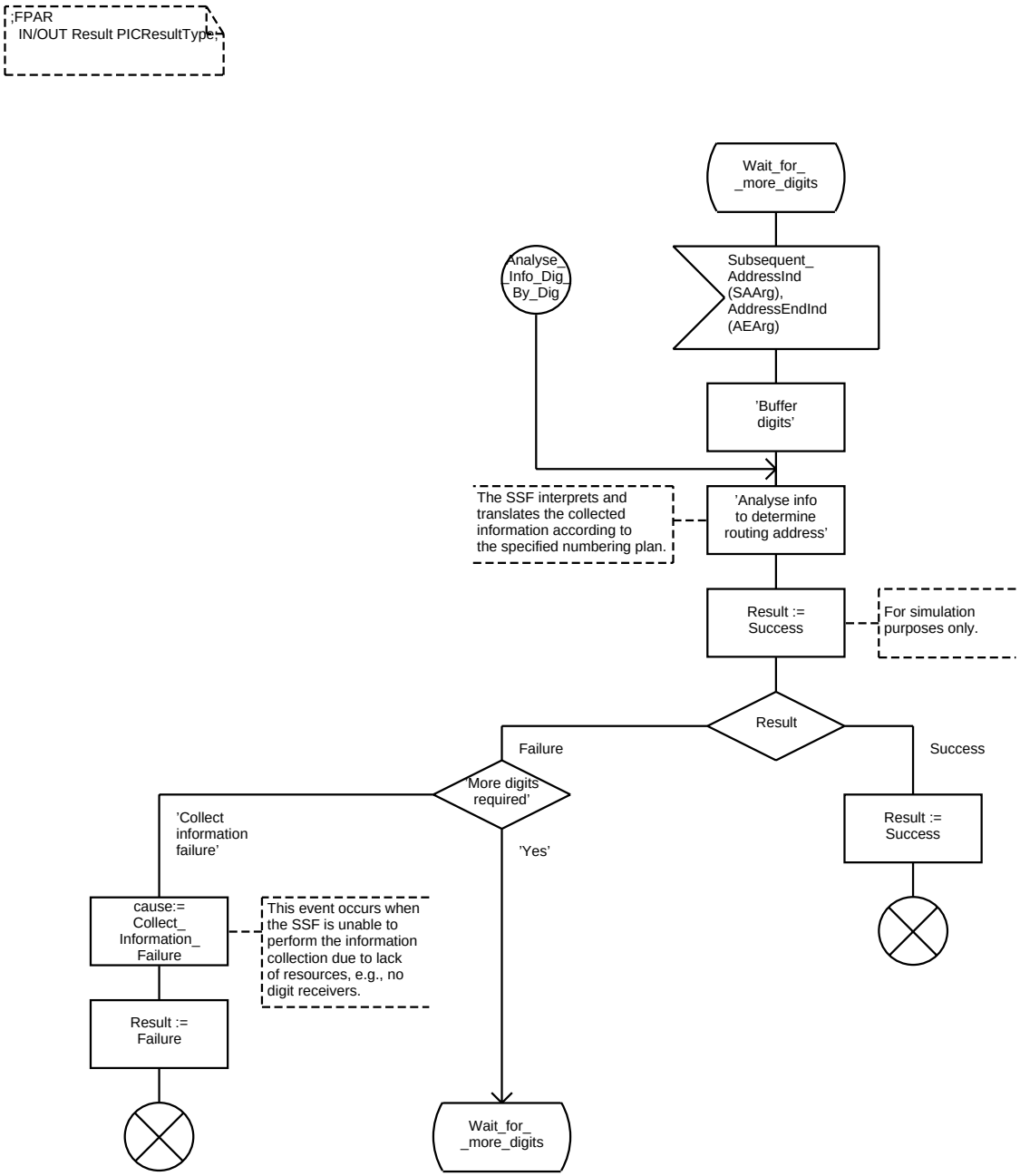

;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
DP_O_
_NoAnswer
DPResult =
Resume
DPResult =
Analyse_
Information
DPResult =
Select_
Route
DPResult =
ARelease
DPResult =
Exception
DPResult =
SCFRelease
PIC_O_
Exception
PIC_Analyse_
_Information
(PICResult)
PIC_Select_
_Route
(PICResult)
DP_oAbandon
(DPResult)
PIC_O_
Exception
PIC_Analyse_
_Information
PIC_Select_
_Route
DP_oAbandon
DP_oMidCall_
_Active
DPResult =
Resume
PICResult =
ARelease
PICResult =
BRelease
DPResult =
Exception
DPResult =
SCFRelease
PIC_O_Active
(PICResult)
DP_oDisconnect
(DPResult,
ARelease)
DP_oDisconnect
(DPResult,
BRelease)
PIC_O_
Exception
PIC_O_Active
DP_oDisconnect

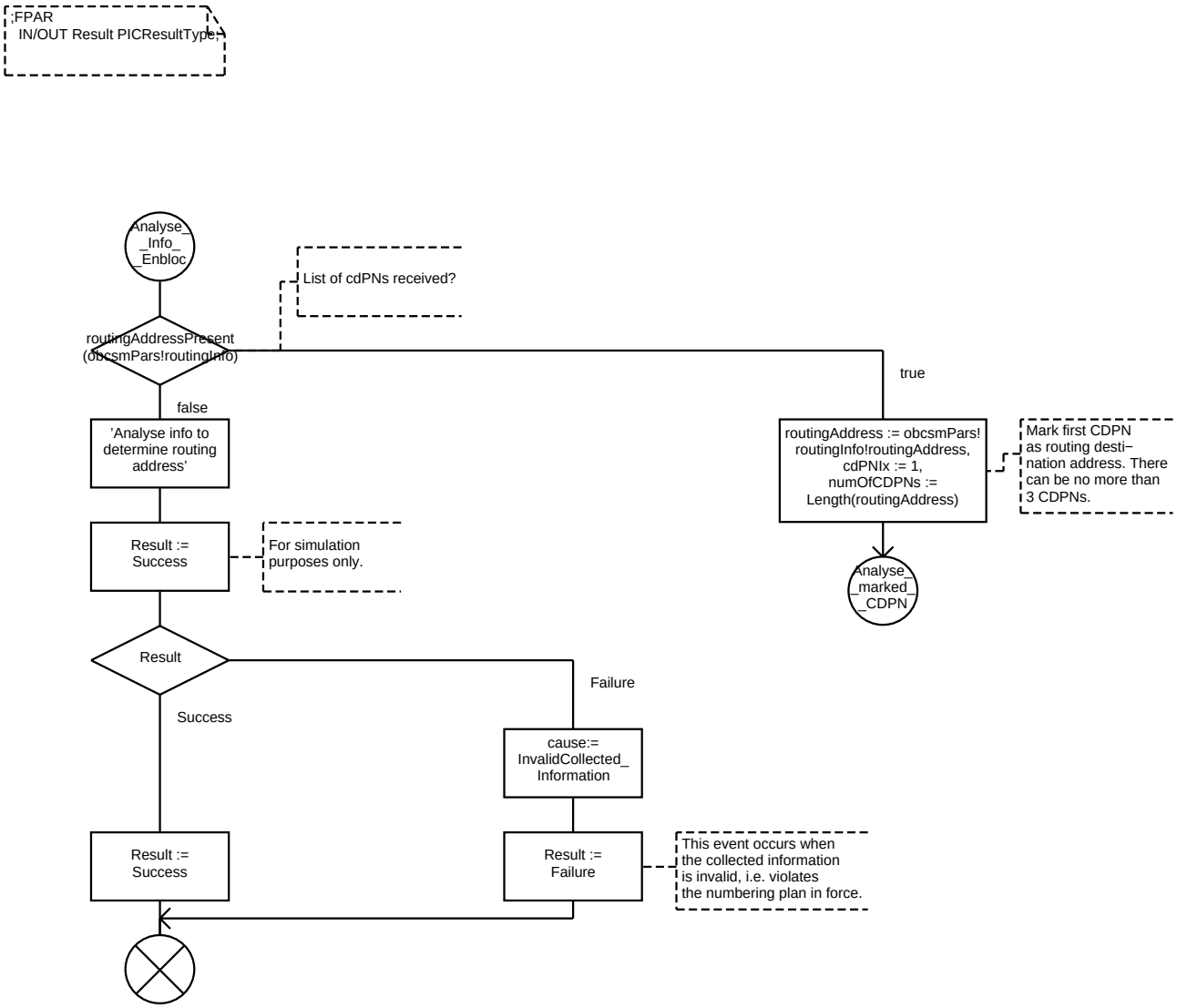
Procedure PIC_O_Null

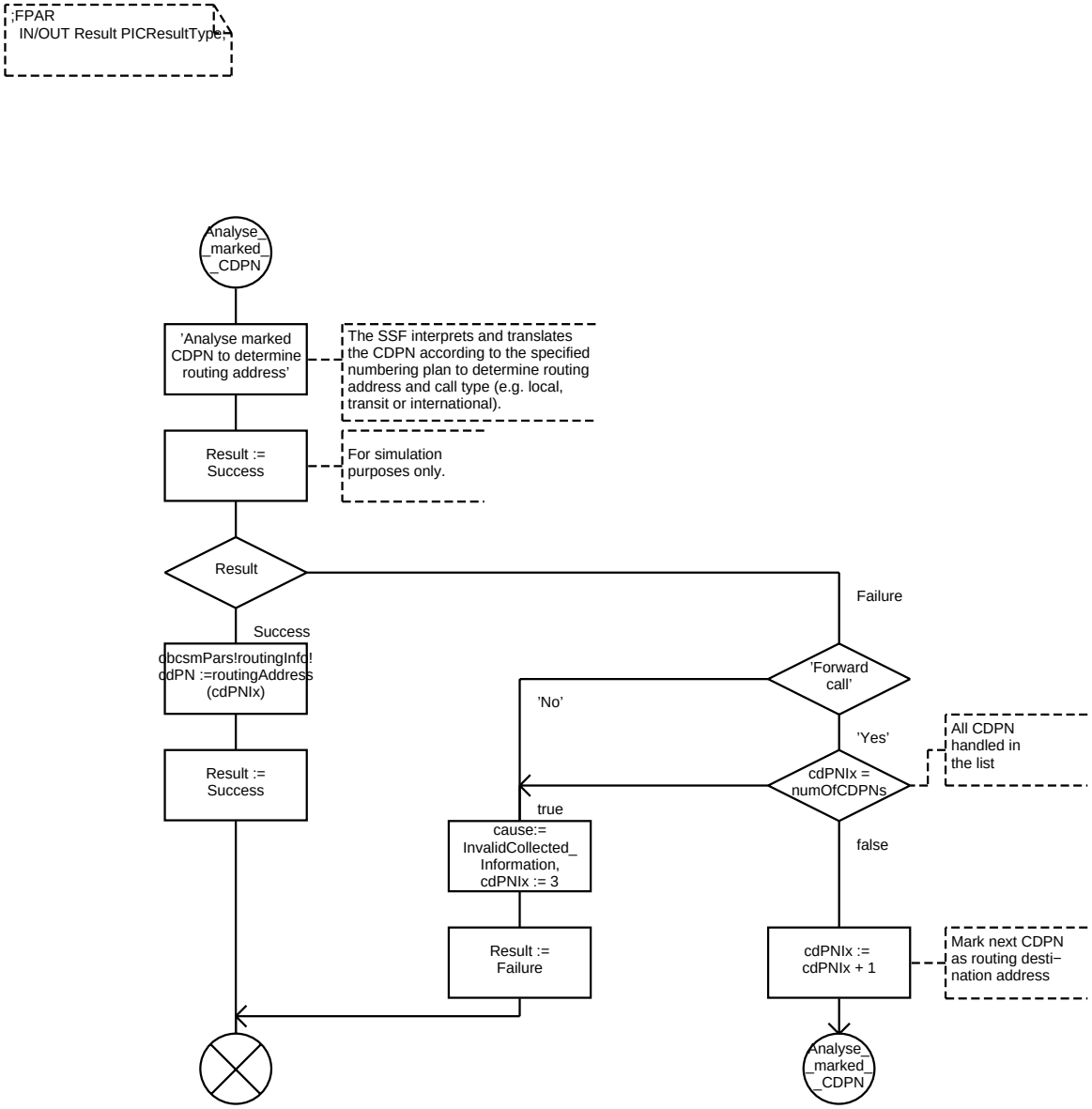


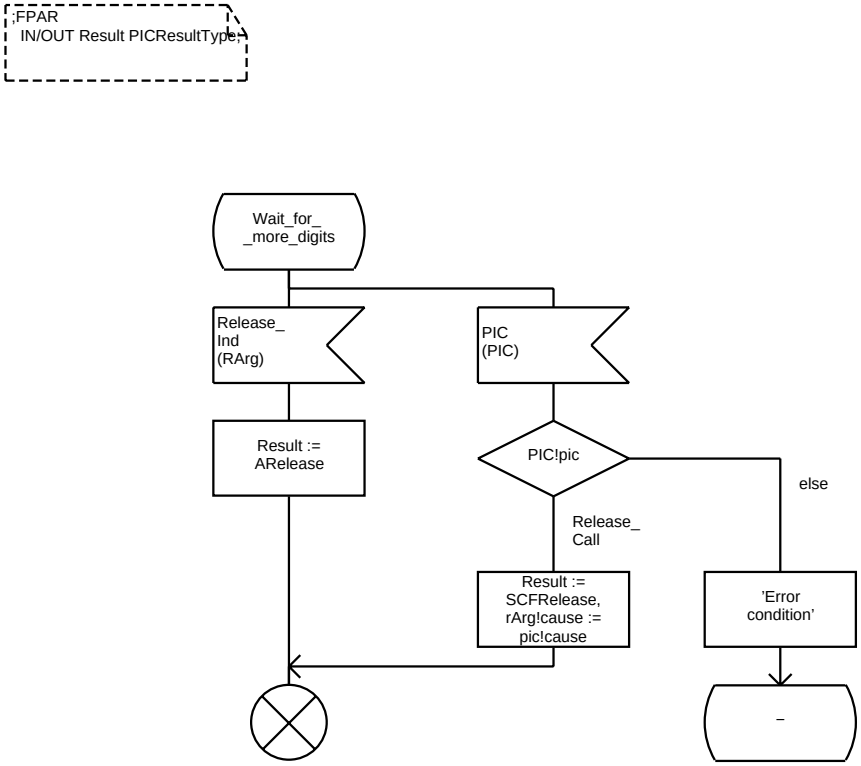
```
;FPAR
  IN/OUT Result PICResultType;
```



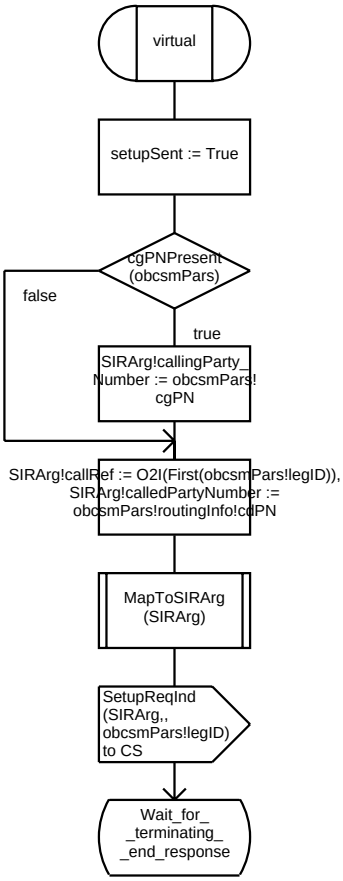

;FPAR
IN/OUT Result PICResultType;
Wait_for_
_more_digits
Analyse_
Info_Dig_
By_Dig
Subsequent_
AddressInd
(SAArg),
AddressEndInd
(AEArg)
'Buffer
digits'
The SSF interprets and
translates the collected
information according to
the specified numbering plan.
'Analyse info
to determine
routing address'
Result :=
Success
For simulation
purposes only.
Result
Failure
Success
'More digits
required'
'Collect
information
failure'
'Yes'
Result :=
Success
cause:=
Collect_
Information_
Failure
This event occurs when
the SSF is unable to
perform the information
collection due to lack
of resources, e.g., no
digit receivers.
Result :=
Failure
Wait_for_
_more_digits

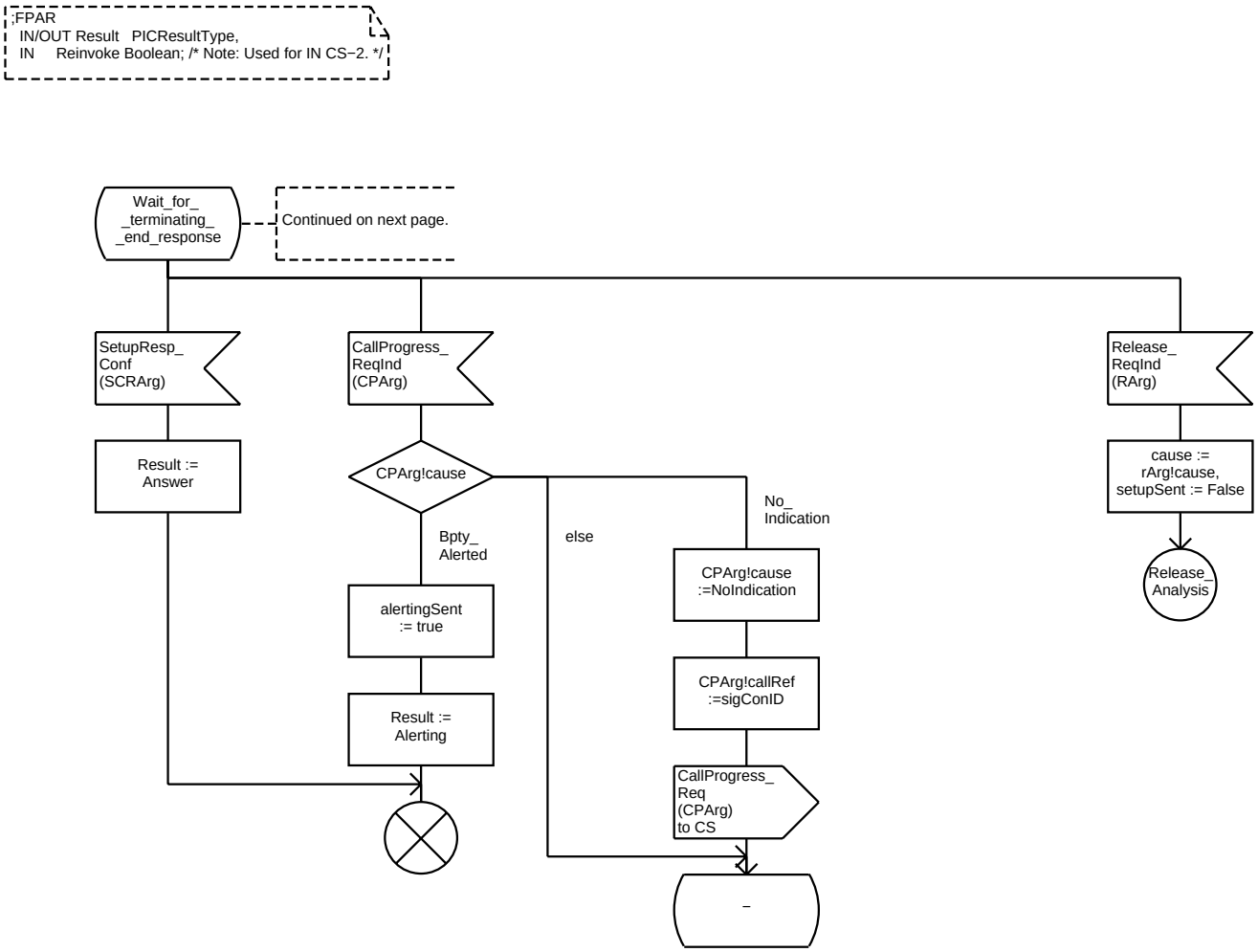

;FPAR
IN/OUT Result PICResultType;
Analyse_
Info
_Enbloc
List of cdPNs received?
routingAddressPresent
(obcsmPars!routingInfo)
true
false
'Analyse info to
determine routing
address'
routingAddress := obcsmPars!
routingInfo!routingAddress,
cdPNIx := 1,
numOfCDPNs :=
Length(routingAddress)
Mark first CDPN
as routing desti−
nation address. There
can be no more than
3 CDPNs.
Result :=
Success
For simulation
purposes only.
Analyse_
marked
_CDPN
Result
Failure
Success
cause:=
InvalidCollected_
Information
Result :=
Success
Result :=
Failure
This event occurs when
the collected information
is invalid, i.e. violates
the numbering plan in force.

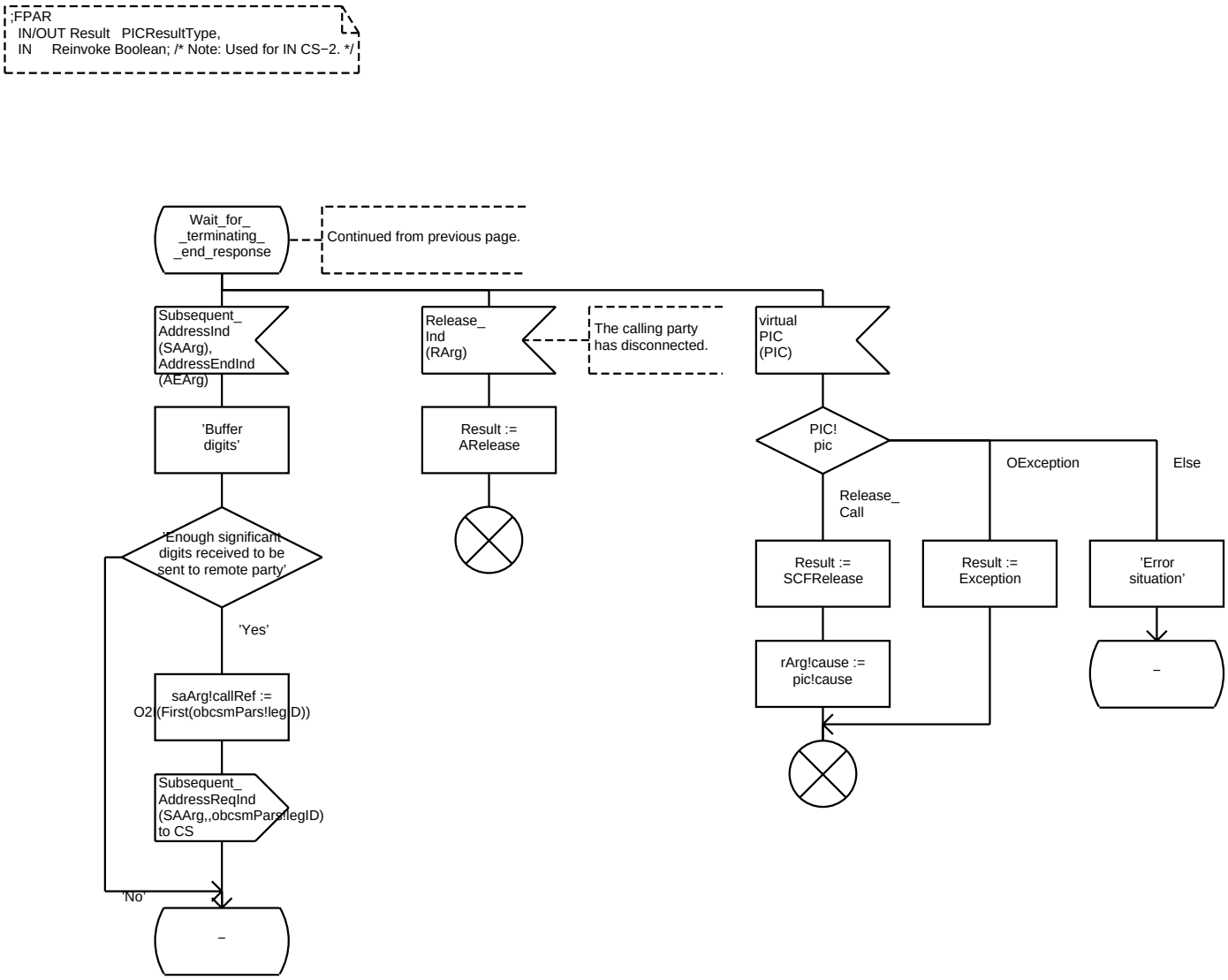

;FPAR
IN/OUT Result PICResultType;
Analyse_
marked
_CDPN
'Analyse marked CDPN to determine routing address'
The SSF interprets and translates the CDPN according to the specified numbering plan to determine routing address and call type (e.g. local, transit or international).
Result := Success
For simulation purposes only.
Result
Failure
Success
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
'Forward call'
'No'
'Yes'
All CDPN handled in the list
Result := Success
cdPNIx = numOfCDPNs
true
cause:= InvalidCollected_ Information, cdPNIx := 3
false
Result := Failure
cdPNIx := cdPNIx + 1
Mark next CDPN as routing desti− nation address
Analyse_
marked
_CDPN

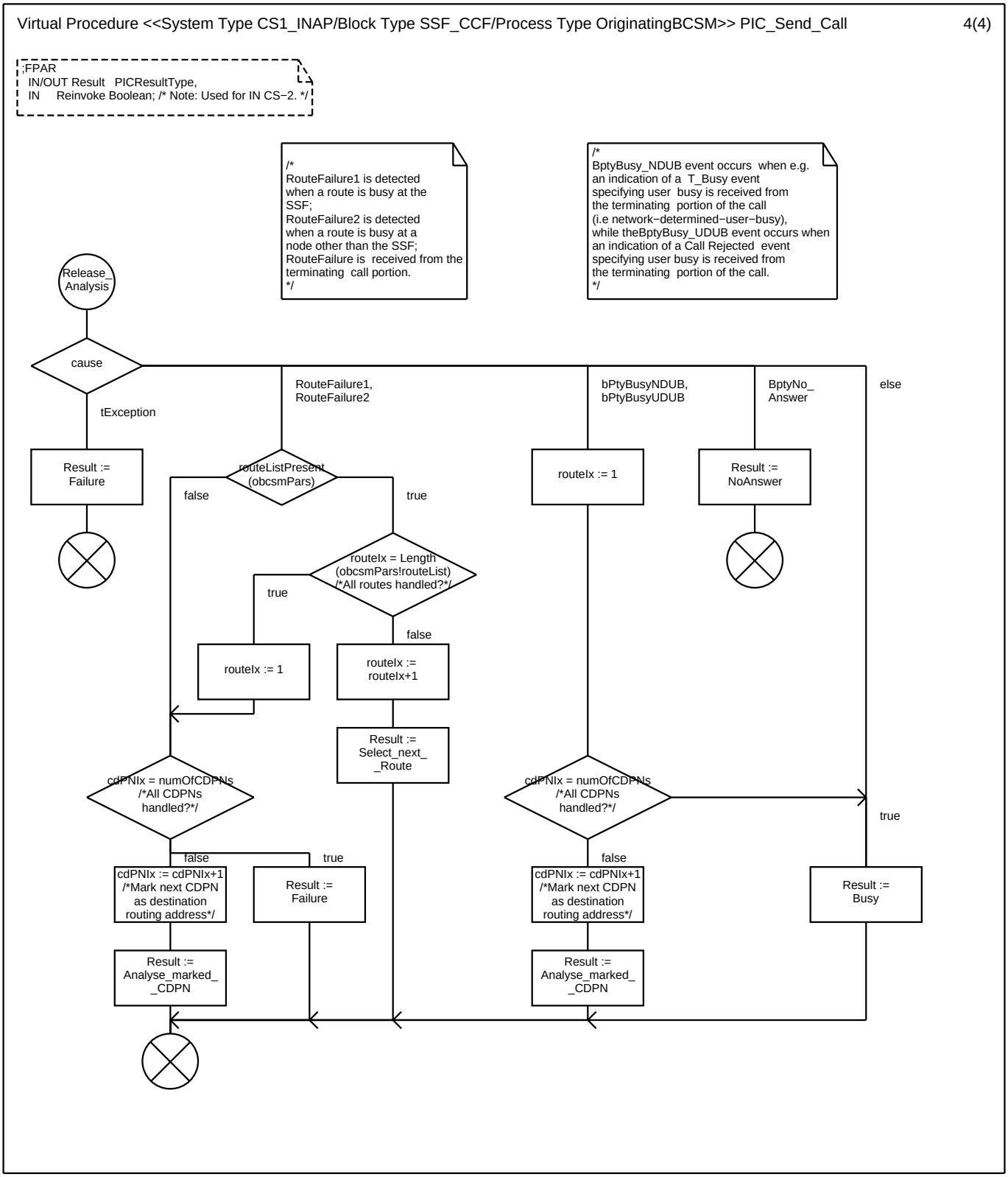

;FPAR
IN/OUT Result PICResultType;
Wait_for_
_more_digits
Release_
Ind
(RArg)
PIC
(PIC)
Result :=
ARelease
PIC!pic
else
Release_
Call
Result :=
SCFRelease,
rArg!cause :=
pic!cause
'Error
condition'
–

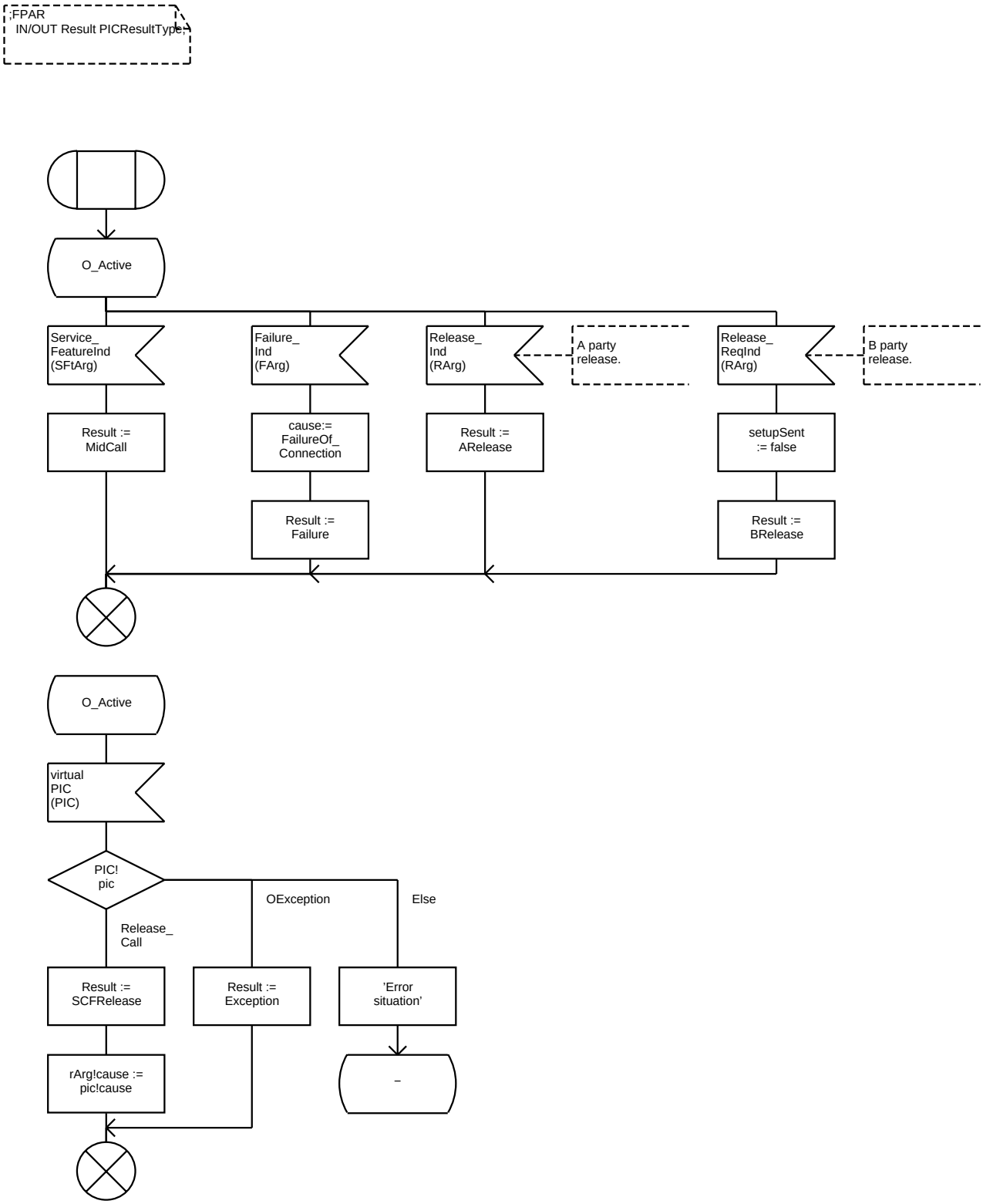
```
;FPAR
  IN/OUT Result   PICResultType,
  IN     Reinvoke Boolean; /* Note: Used for IN CS−2. */
```

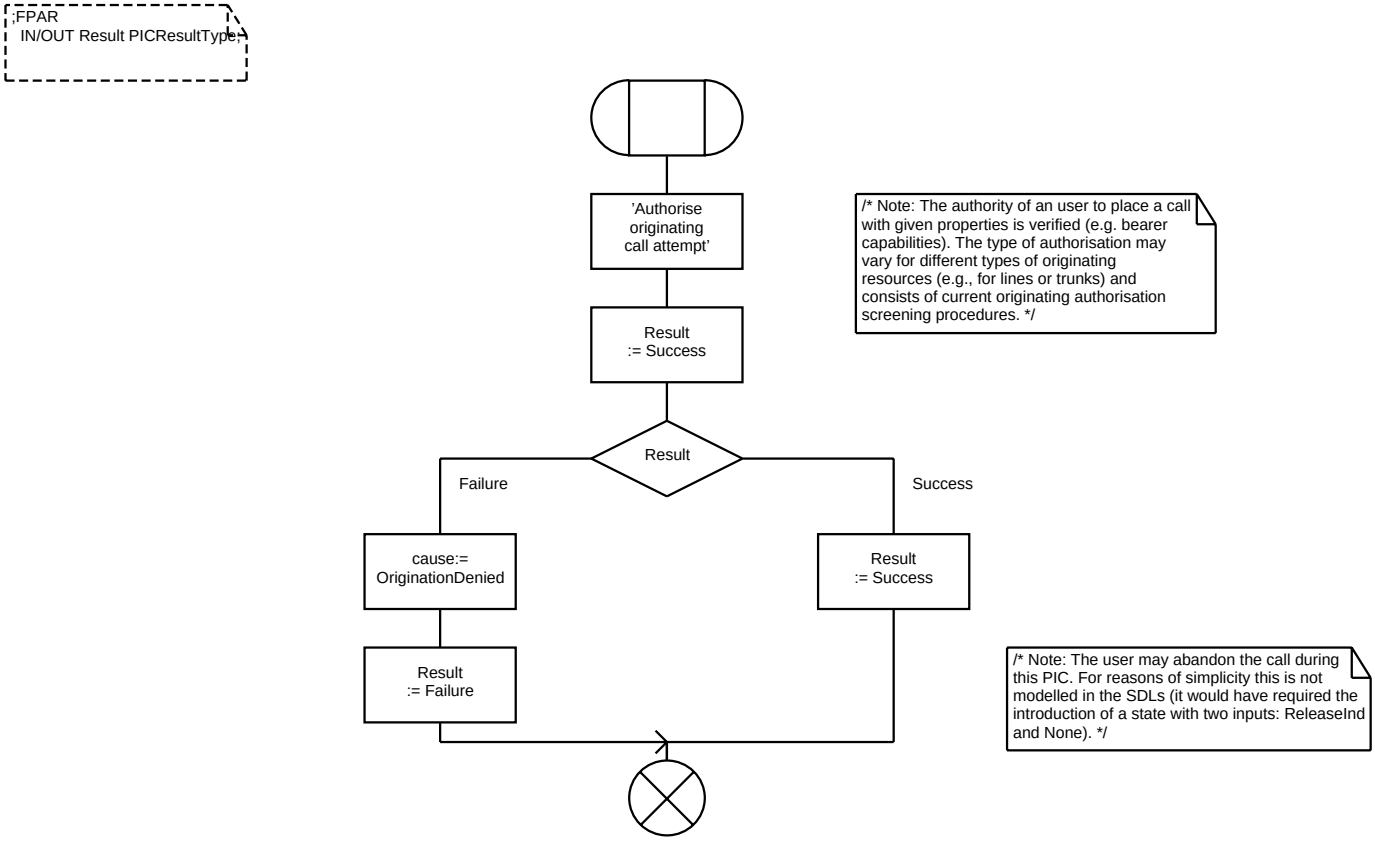


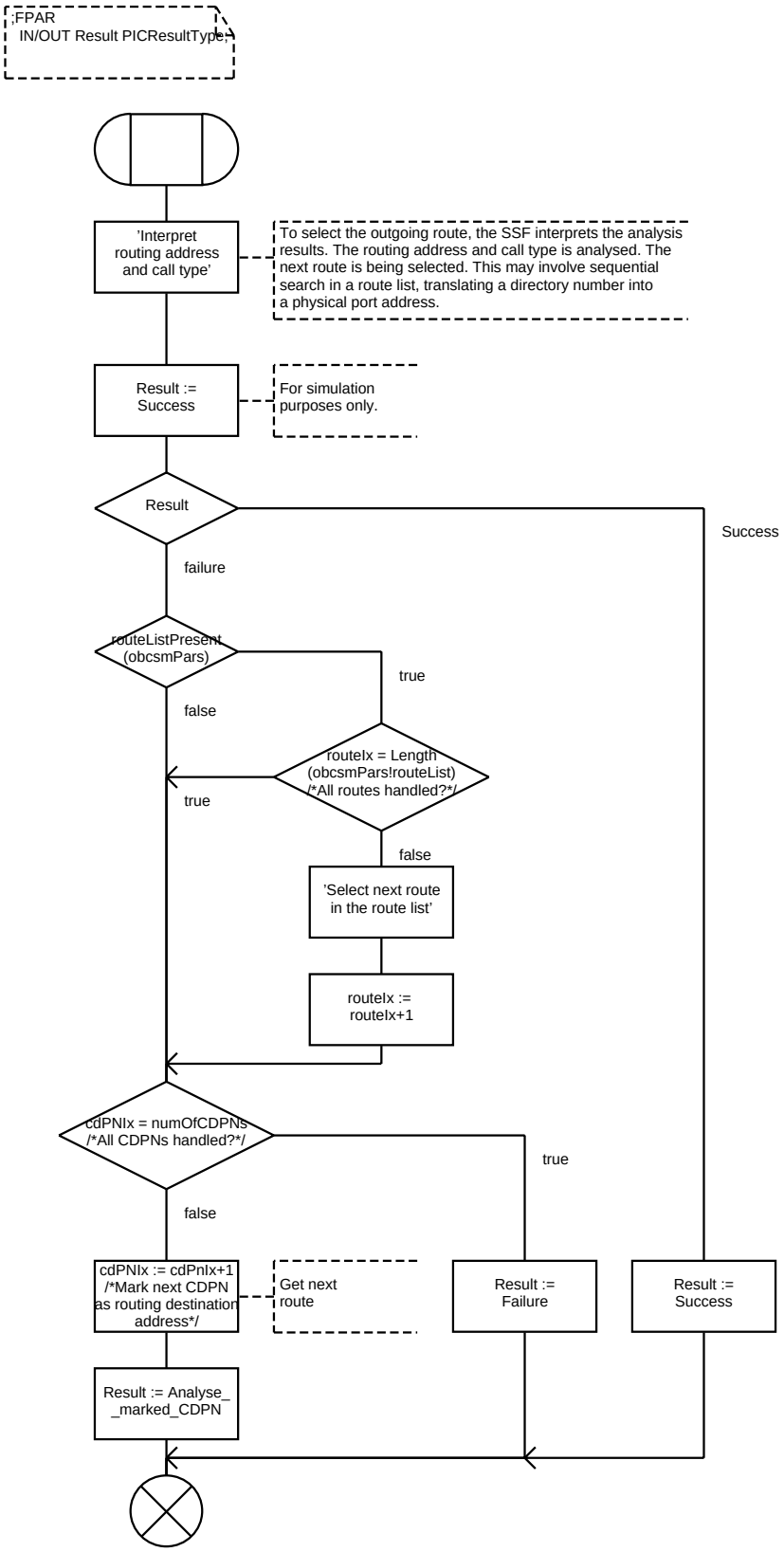

;FPAR
IN/OUT Result PICResultType,
IN Reinvoke Boolean; /* Note: Used for IN CS−2. */
Wait_for_
terminating
_end_response
Continued on next page.
SetupResp_
Conf
(SCRArg)
CallProgress_
ReqInd
(CPArg)
Release_
ReqInd
(RArg)
Result :=
Answer
CPArg!cause
cause :=
rArg!cause,
setupSent := False
No_
Indication
else
Bpty_
Alerted
CPArg!cause
:=NoIndication
Release_
Analysis
alertingSent
:= true
CPArg!callRef
:=sigConID
Result :=
Alerting
CallProgress_
Req
(CPArg)
to CS
_


;FPAR
IN/OUT Result PICResultType,
IN Reinvoke Boolean; /* Note: Used for IN CS−2. */
Wait_for_
terminating
_end_response
Continued from previous page.
Subsequent_
AddressInd
(SAArg),
AddressEndInd
(AEArg)
Release_
Ind
(RArg)
The calling party
has disconnected.
virtual
PIC
(PIC)
'Buffer
digits'
Result :=
ARelease
PIC!
pic
OException
Else
Release_
Call
'Enough significant
digits received to be
sent to remote party'
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
'Yes'
rArg!cause :=
pic!cause
–
saArg!callRef :=
O2(First(obcsmPars!legID))
Subsequent_
AddressReqInd
(SAArg,,obcsmPars!legID)
to CS
'No'
–

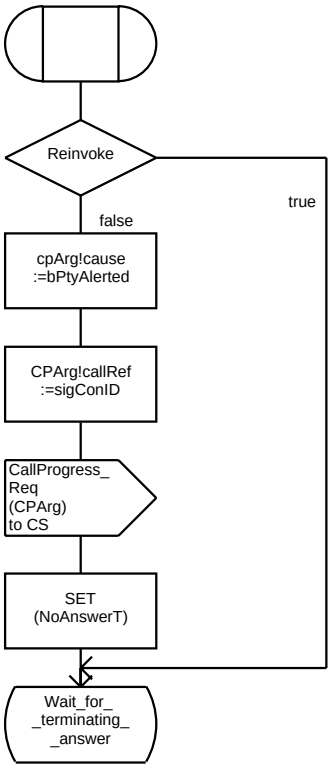

;FPAR
IN/OUT Result PICResultType,
IN Reinvoke Boolean; /* Note: Used for IN CS−2. */
/*
RouteFailure1 is detected
when a route is busy at the
SSF;
RouteFailure2 is detected
when a route is busy at a
node other than the SSF;
RouteFailure is received from the
terminating call portion.
*/
/*
BptyBusy_NDUB event occurs when e.g.
an indication of a T_Busy event
specifying user busy is received from
the terminating portion of the call
(i.e network−determined−user−busy),
while theBptyBusy_UDUB event occurs when
an indication of a Call Rejected event
specifying user busy is received from
the terminating portion of the call.
*/
Release_
Analysis
cause
tException
RouteFailure1,
RouteFailure2
bPtyBusyNDUB,
bPtyBusyUDUB
BptyNo_
Answer
else
Result :=
Failure
routeListPresent
(obcsmPars)
false
true
routeIx := 1
Result :=
NoAnswer
routeIx = Length
(obcsmPars!routeList)
/*All routes handled?*/
true
false
routeIx := 1
routeIx :=
routeIx+1
Result :=
Select_next_
_Route
cdPNIx = numOfCDPNs
/*All CDPNs
handled?*/
cdPNIx = numOfCDPNs
/*All CDPNs
handled?*/
true
false
true
false
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Failure
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Busy
Result :=
Analyse_marked_
_CDPN
Result :=
Analyse_marked_
_CDPN

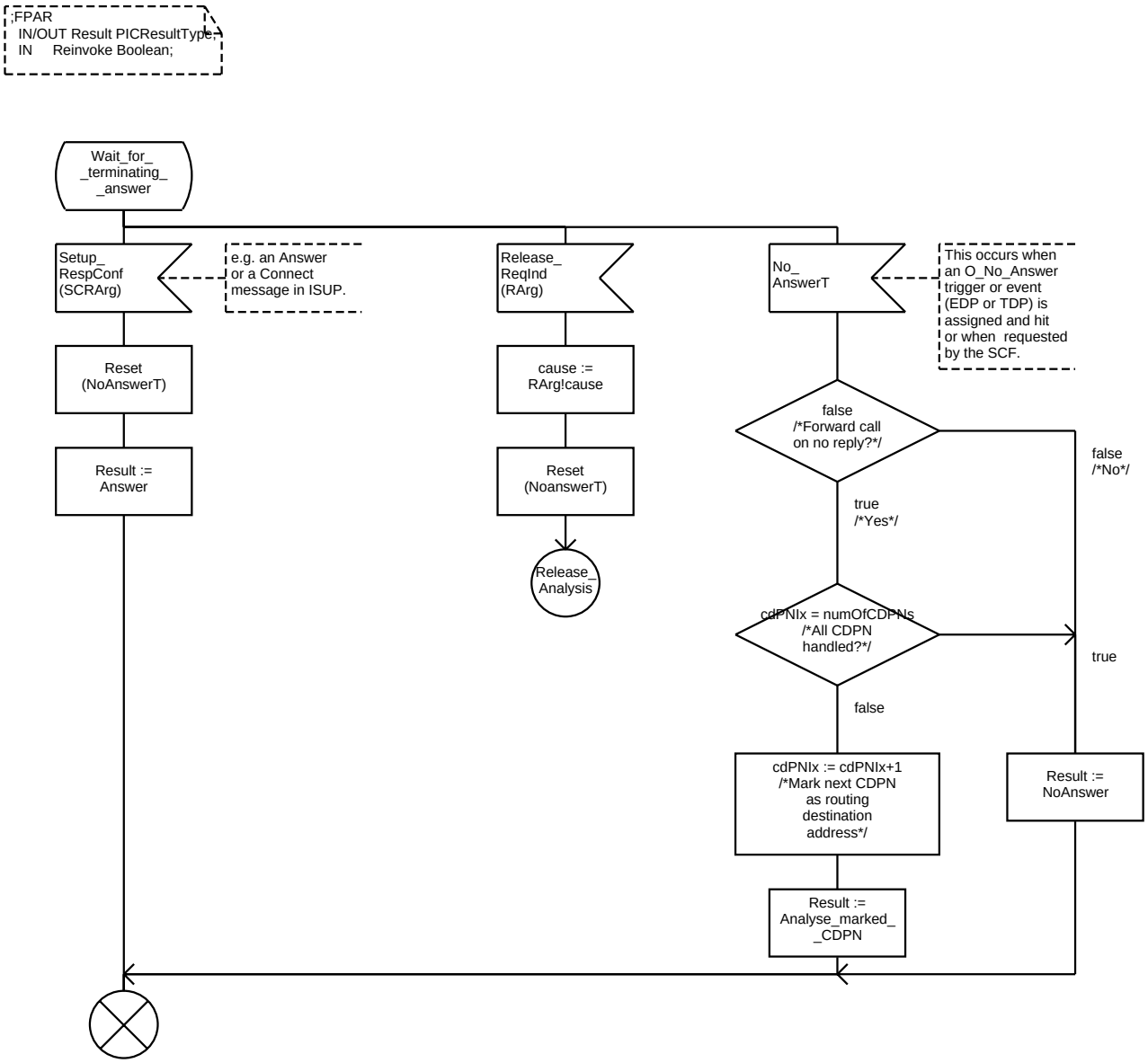

;FPAR
IN/OUT Result PICResultType;
O_Active
Service_
FeatureInd
(SFtArg)
Failure_
Ind
(FArg)
Release_
Ind
(RArg)
A party
release.
Release_
ReqInd
(RArg)
B party
release.
Result :=
MidCall
cause:=
FailureOf_
Connection
Result :=
ARelease
setupSent
:= false
Result :=
Failure
Result :=
BRelease
O_Active
virtual
PIC
(PIC)
PIC!
pic
OException
Else
Release_
Call
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
rArg!cause :=
pic!cause
_

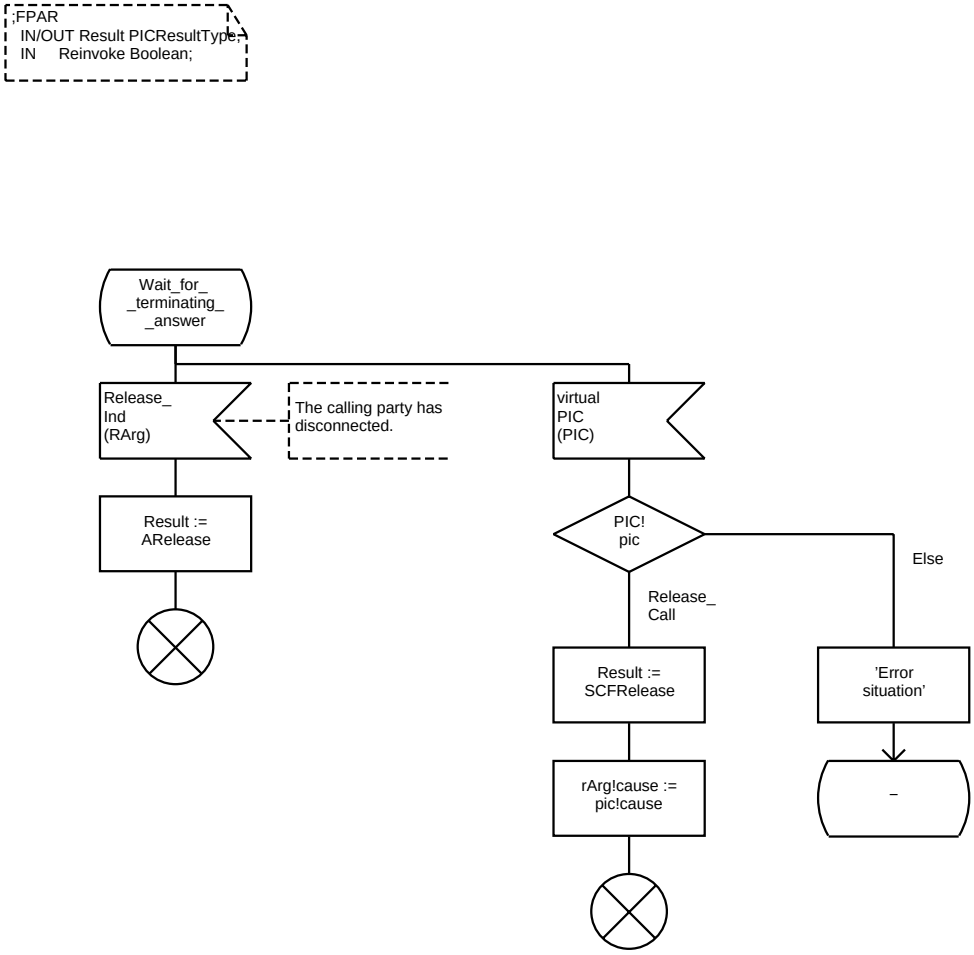

;FPAR
IN/OUT Result PICResultType;
'Authorise originating call attempt'
Result := Success
Result
Failure
Success
cause:= OriginationDenied
Result := Failure
Result := Success
/* Note: The authority of an user to place a call with given properties is verified (e.g. bearer capabilities). The type of authorisation may vary for different types of originating resources (e.g., for lines or trunks) and consists of current originating authorisation screening procedures. */
/* Note: The user may abandon the call during this PIC. For reasons of simplicity this is not modelled in the SDLs (it would have required the introduction of a state with two inputs: ReleaseInd and None). */

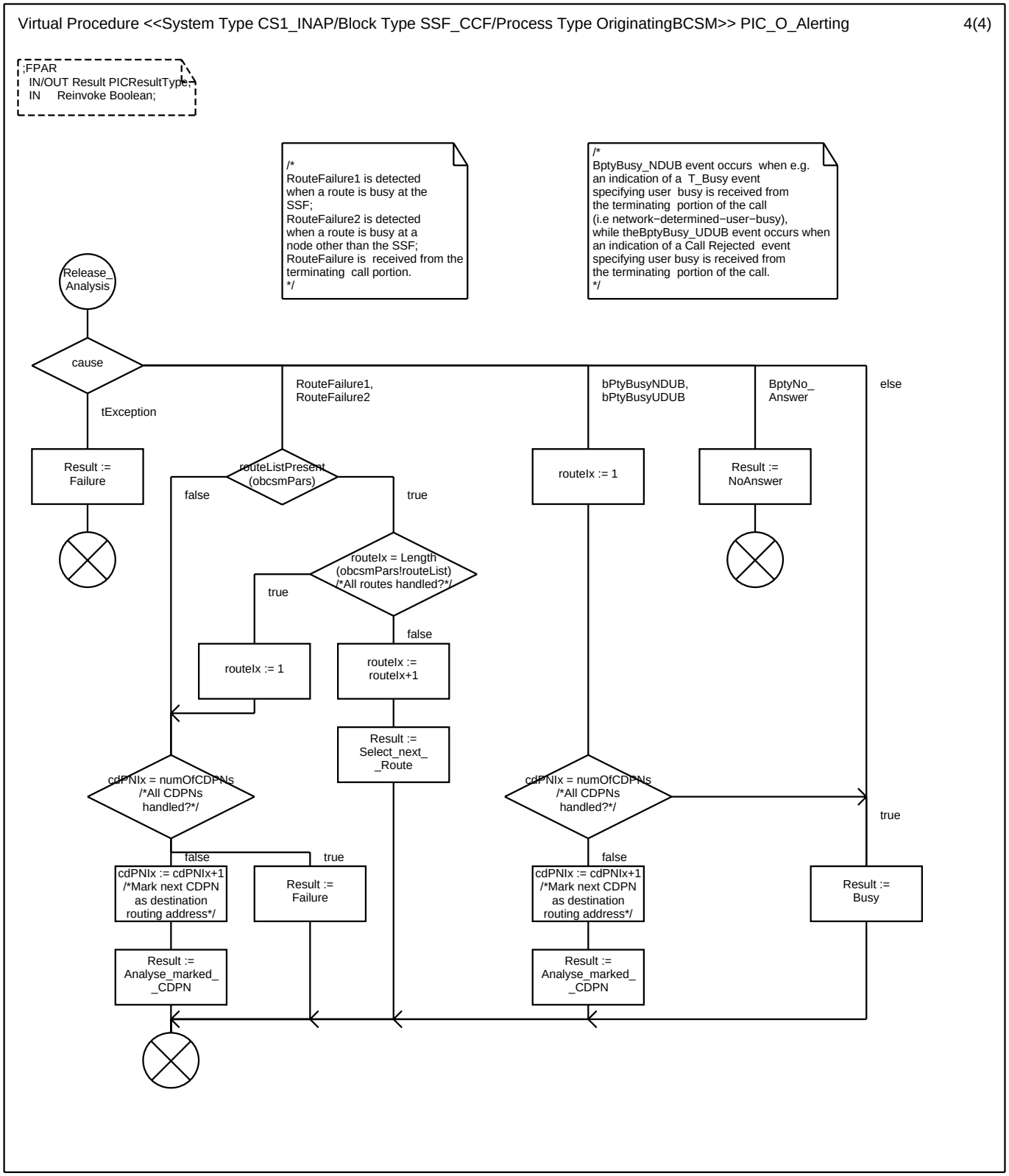

;FPAR
IN/OUT Result PICResultType;
'Interpret routing address and call type'
To select the outgoing route, the SSF interprets the analysis results. The routing address and call type is analysed. The next route is being selected. This may involve sequential search in a route list, translating a directory number into a physical port address.
Result := Success
For simulation purposes only.
Result
Success
failure
routeListPresent (obcsmPars)
true
false
routeIx = Length (obcsmPars!routeList) /*All routes handled?*/
true
false
'Select next route in the route list'
routeIx := routeIx+1
cdPNIx = numOfCDPNs /*All CDPNs handled?*/
true
false
cdPNIx := cdPnIx+1 /*Mark next CDPN as routing destination address*/
Get next route
Result := Failure
Result := Success
Result := Analyse_ _marked_CDPN

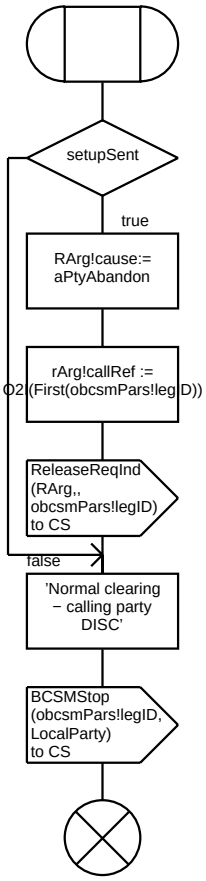
```
;FPAR
  IN/OUT Result PICResultType;
  IN     Reinvoke Boolean;
```

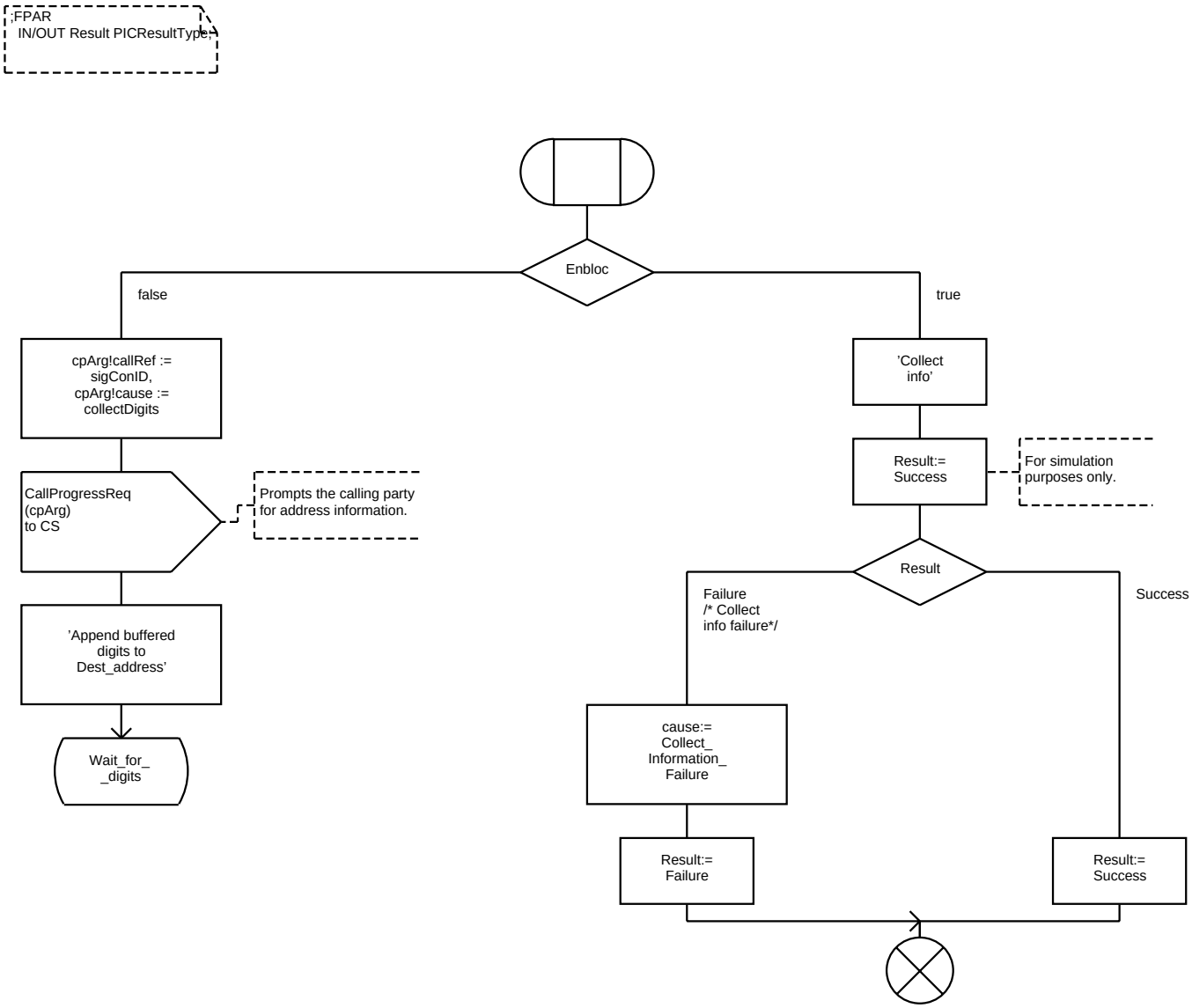


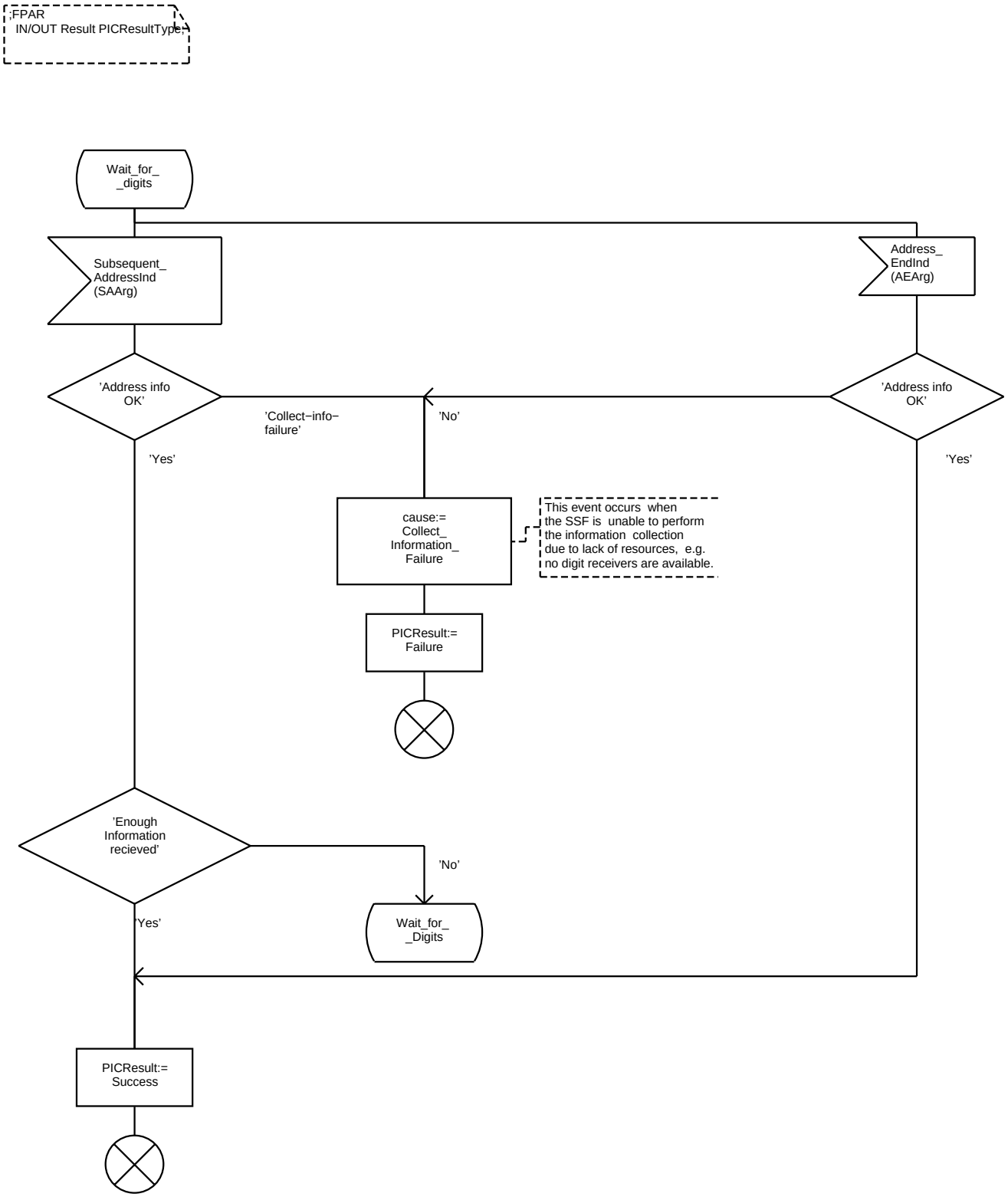

;FPAR
IN/OUT Result PICResultType;
IN Reinvoke Boolean;
Wait_for_
terminating
_answer
Setup_
RespConf
(SCRArg)
e.g. an Answer
or a Connect
message in ISUP.
Release_
ReqInd
(RArg)
No_
AnswerT
This occurs when
an O_No_Answer
trigger or event
(EDP or TDP) is
assigned and hit
or when requested
by the SCF.
Reset
(NoAnswerT)
cause :=
RArg!cause
false
/*Forward call
on no reply?*/
false
/*No*/
Result :=
Answer
Reset
(NoanswerT)
true
/*Yes*/
Release_
Analysis
cdPNIx = numOfCDPNs
/*All CDPN
handled?*/
true
false
cdPNIx := cdPNIx+1
/*Mark next CDPN
as routing
destination
address*/
Result :=
NoAnswer
Result :=
Analyse_marked_
_CDPN

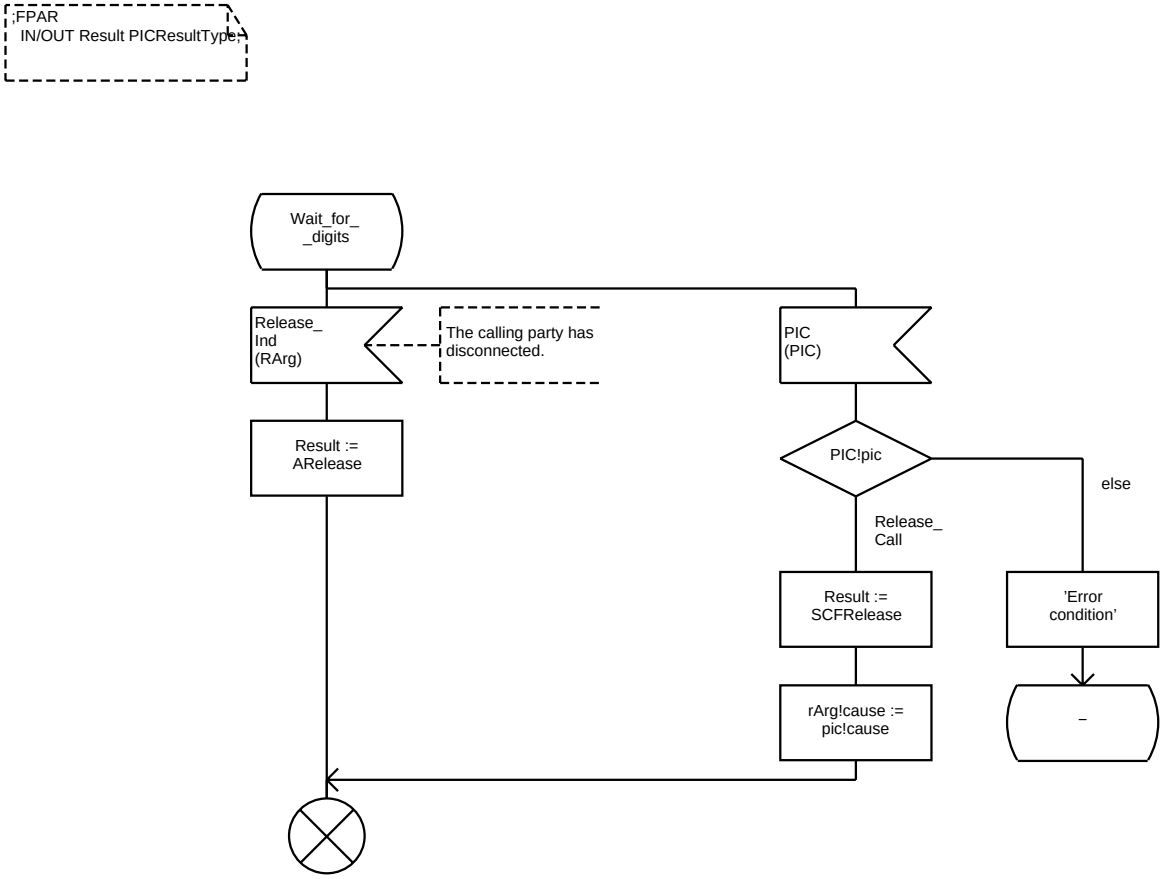

;FPAR
IN/OUT Result PICResultType;
IN Reinvoke Boolean;
Wait_for_
terminating
_answer
Release_
Ind
(RArg)
The calling party has disconnected.
virtual
PIC
(PIC)
Result :=
ARelease
PIC!
pic
Else
Release_
Call
Result :=
SCFRelease
'Error
situation'
rArg!cause :=
pic!cause
_

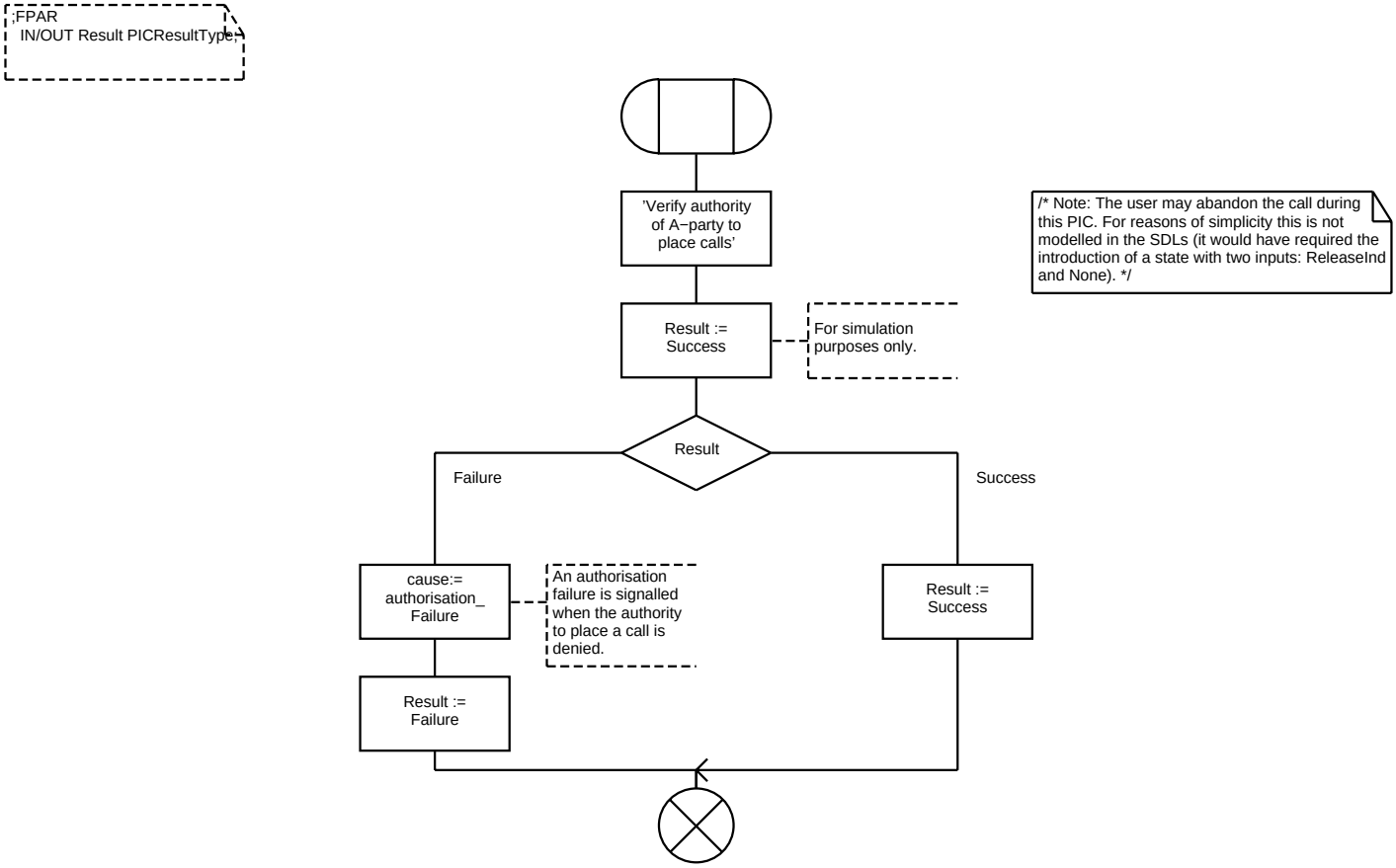

;FPAR
IN/OUT Result PICResultType;
IN Reinvoke Boolean;
/*
RouteFailure1 is detected
when a route is busy at the
SSF;
RouteFailure2 is detected
when a route is busy at a
node other than the SSF;
RouteFailure is received from the
terminating call portion.
*/
/*
BptyBusy_NDUB event occurs when e.g.
an indication of a T_Busy event
specifying user busy is received from
the terminating portion of the call
(i.e network−determined−user−busy),
while theBptyBusy_UDUB event occurs when
an indication of a Call Rejected event
specifying user busy is received from
the terminating portion of the call.
*/
Release_
Analysis
cause
tException
RouteFailure1,
RouteFailure2
bPtyBusyNDUB,
bPtyBusyUDUB
BptyNo_
Answer
else
Result :=
Failure
routeListPresent
(obcsmPars)
false
true
routeIx := 1
Result :=
NoAnswer
routeIx = Length
(obcsmPars!routeList)
/*All routes handled?*/
true
false
routeIx := 1
routeIx :=
routeIx+1
Result :=
Select_next_
_Route
cdPNIx = numOfCDPNs
/*All CDPNs
handled?*/
cdPNIx = numOfCDPNs
/*All CDPNs
handled?*/
true
false
true
false
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Failure
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Busy
Result :=
Analyse_marked_
_CDPN
Result :=
Analyse_marked_
_CDPN


setupSent
true
RArg!cause:=
aPtyAbandon
rArg!callRef :=
O2I(First(obcsmPars!legID))
ReleaseReqInd
(RArg,,
obcsmPars!legID)
to CS
false
'Normal clearing
− calling party
DISC'
BCSMStop
(obcsmPars!legID,
LocalParty)
to CS


;FPAR
IN/OUT Result PICResultType;
Enbloc
false
true
cpArg!callRef :=
sigConID,
cpArg!cause :=
collectDigits
CallProgressReq
(cpArg)
to CS
Prompts the calling party
for address information.
'Append buffered
digits to
Dest_address'
Wait_for_
_digits
'Collect
info'
Result:=
Success
For simulation
purposes only.
Result
Failure
/* Collect
info failure*/
Success
cause:=
Collect_
Information_
Failure
Result:=
Failure
Result:=
Success

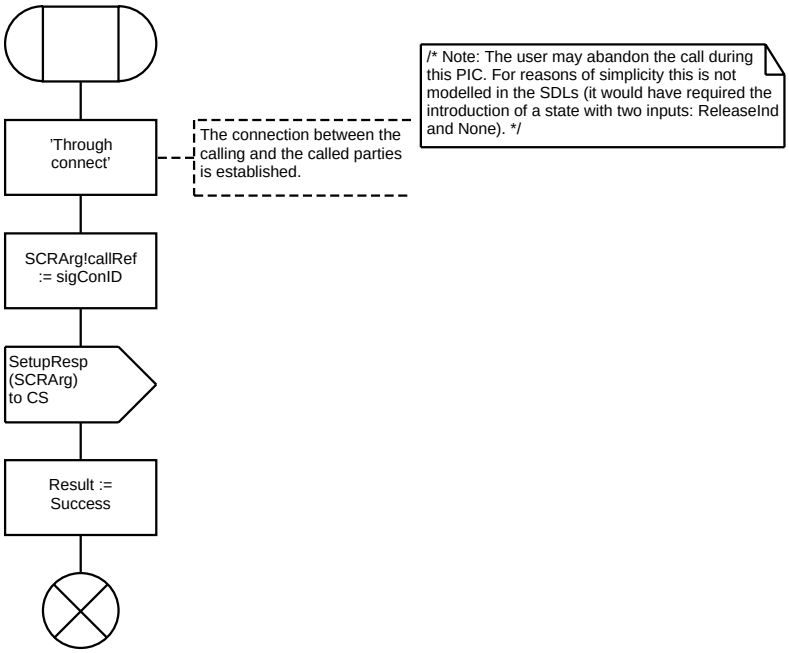

;FPAR
IN/OUT Result PICResultType;
Wait_for_
_digits
Subsequent_
AddressInd
(SAArg)
Address_
EndInd
(AEArg)
'Address info
OK'
'Address info
OK'
'Collect−info−
failure'
'No'
'Yes'
'Yes'
cause:=
Collect_
Information_
Failure
This event occurs when
the SSF is unable to perform
the information collection
due to lack of resources, e.g.
no digit receivers are available.
PICResult:=
Failure
'Enough
Information
recieved'
'No'
'Yes'
Wait_for_
_Digits
PICResult:=
Success

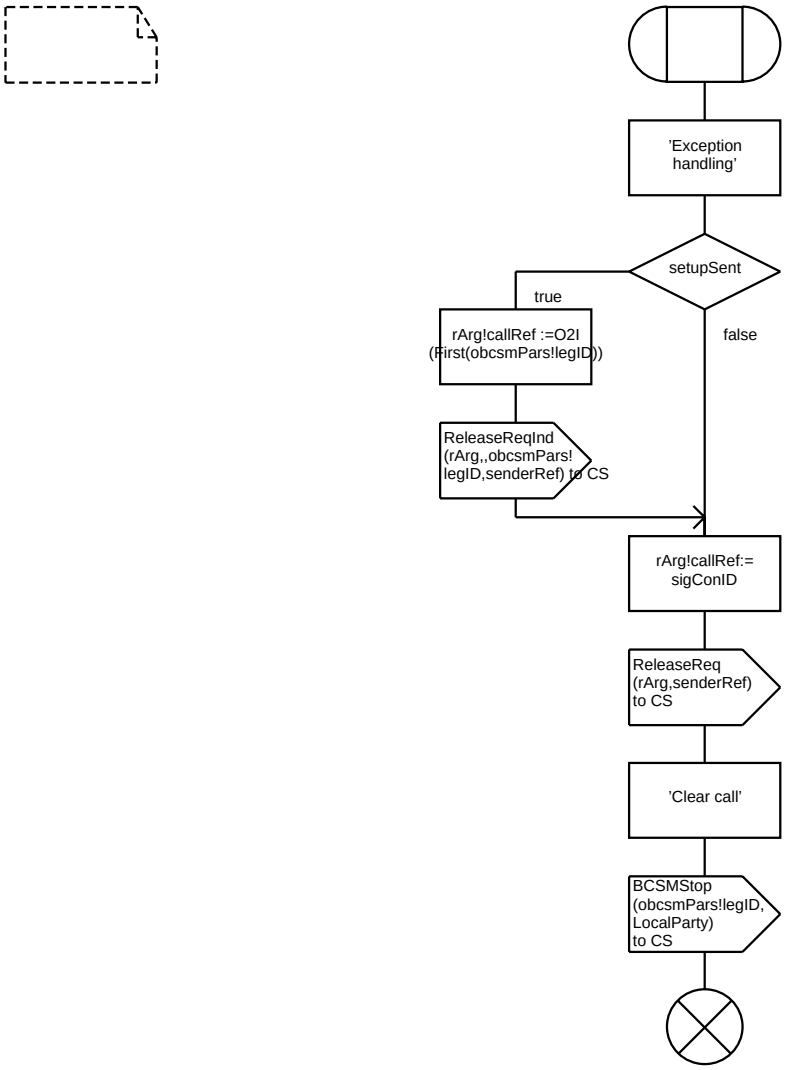

;FPAR
IN/OUT Result PICResultType;
Wait_for_
_digits
Release_
Ind
(RArg)
The calling party has
disconnected.
PIC
(PIC)
Result :=
ARelease
PIC!pic
else
Release_
Call
Result :=
SCFRelease
'Error
condition'
rArg!cause :=
pic!cause
_

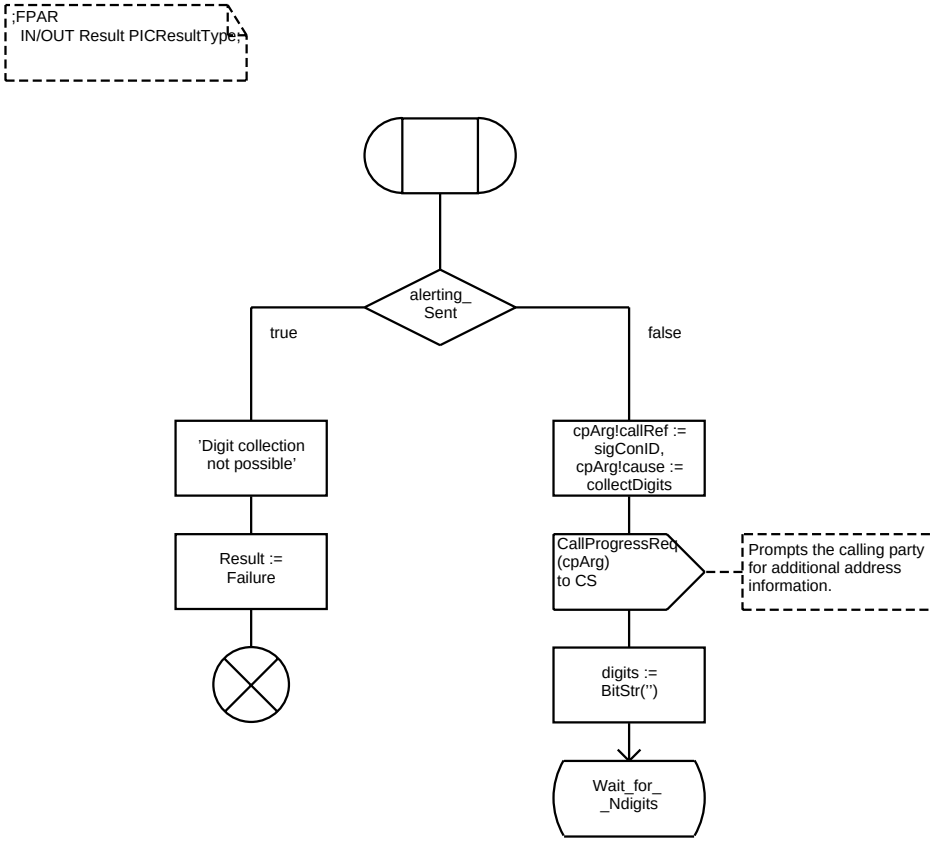

;FPAR
IN/OUT Result PICResultType;
'Verify authority of A−party to place calls'
Result := Success
For simulation purposes only.
/* Note: The user may abandon the call during this PIC. For reasons of simplicity this is not modelled in the SDLs (it would have required the introduction of a state with two inputs: ReleaseInd and None). */
Result
Failure
Success
cause:= authorisation_ Failure
An authorisation failure is signalled when the authority to place a call is denied.
Result := Success
Result := Failure

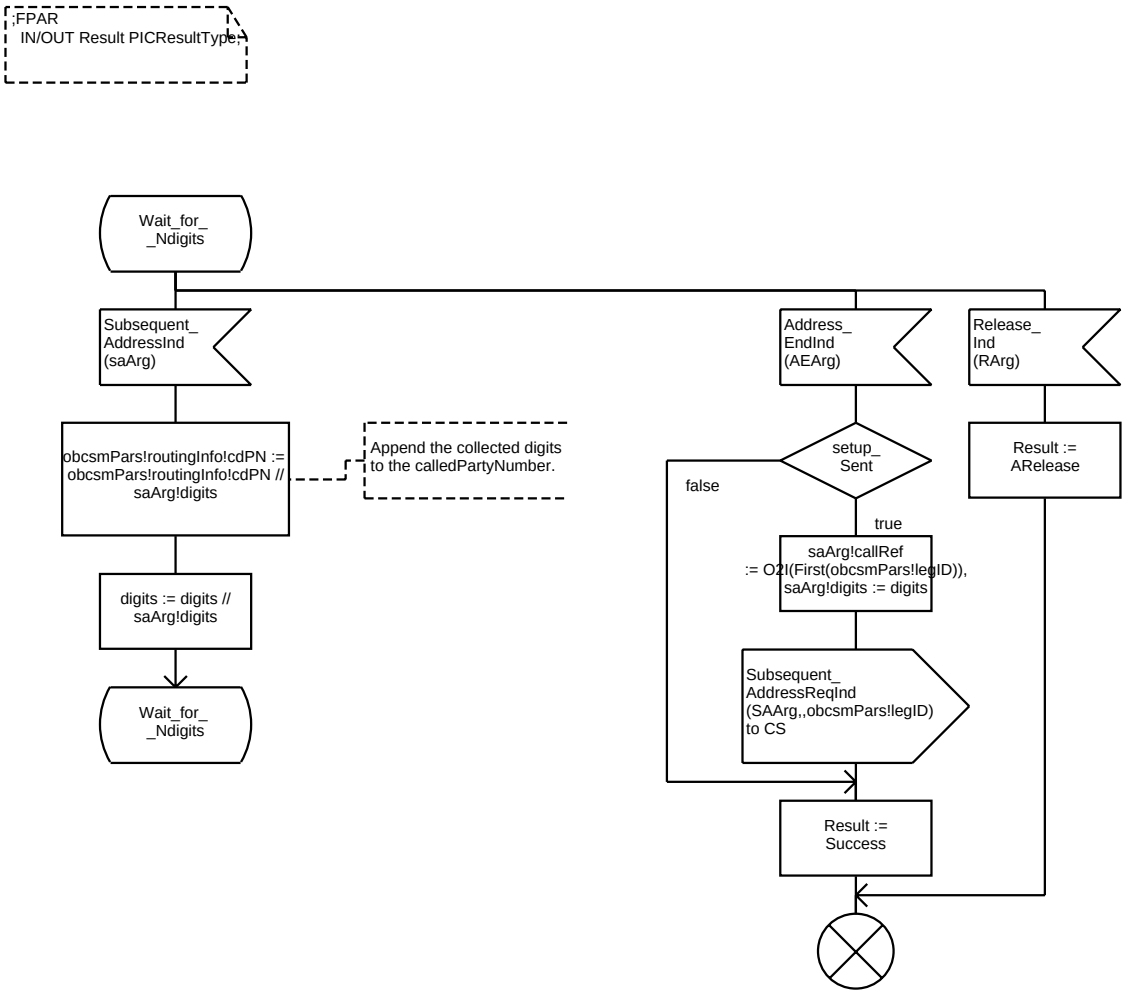
Procedure PIC_O_Answer

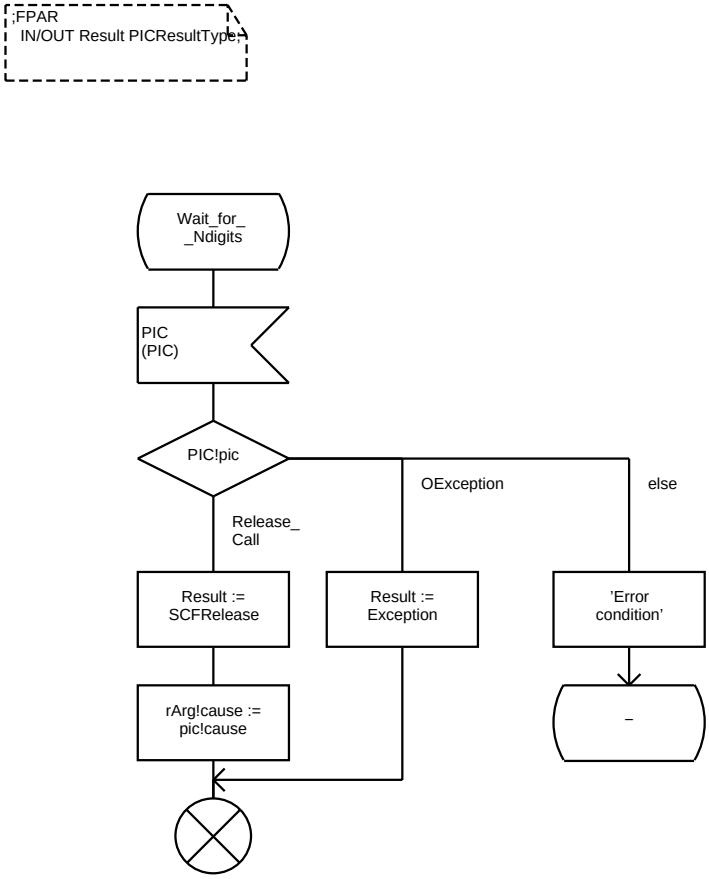

```
;FPAR
  IN/OUT Result PICResultType;
```



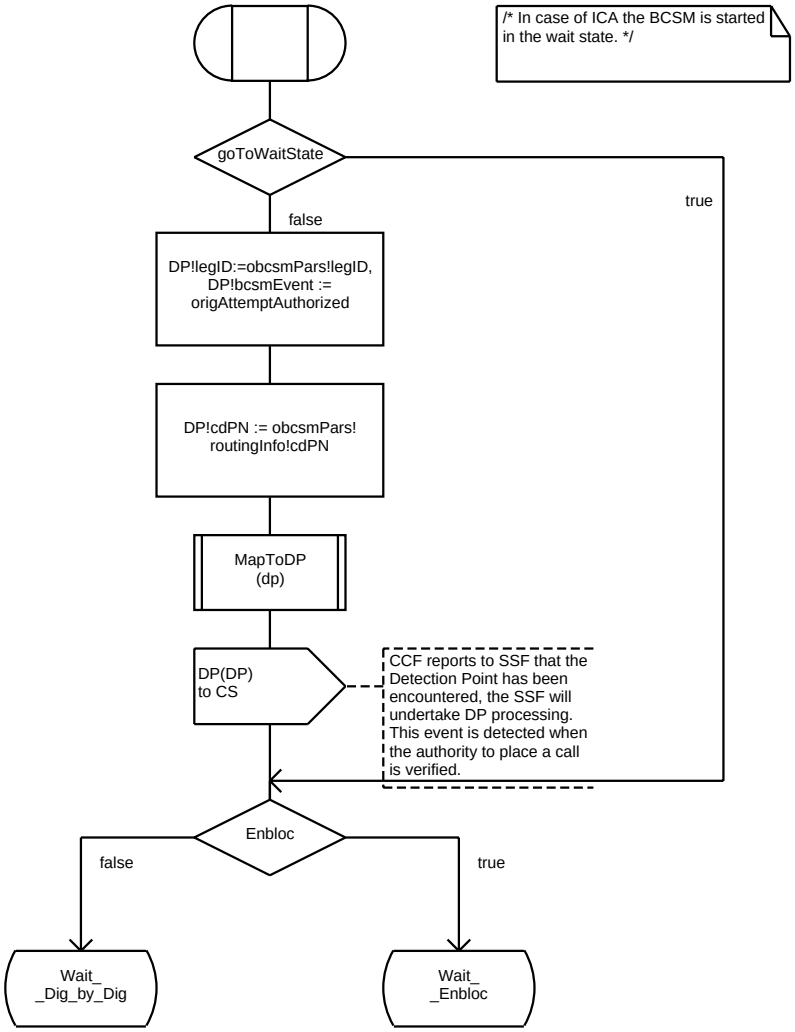

'Exception handling'
setupSent
true
false
rArg!callRef :=O2I (First(obcsmPars!legID))
ReleaseReqInd (rArg,,obcsmPars! legID,senderRef) to CS
rArg!callRef:= sigConID
ReleaseReq (rArg,senderRef) to CS
'Clear call'
BCSMStop (obcsmPars!legID, LocalParty) to CS

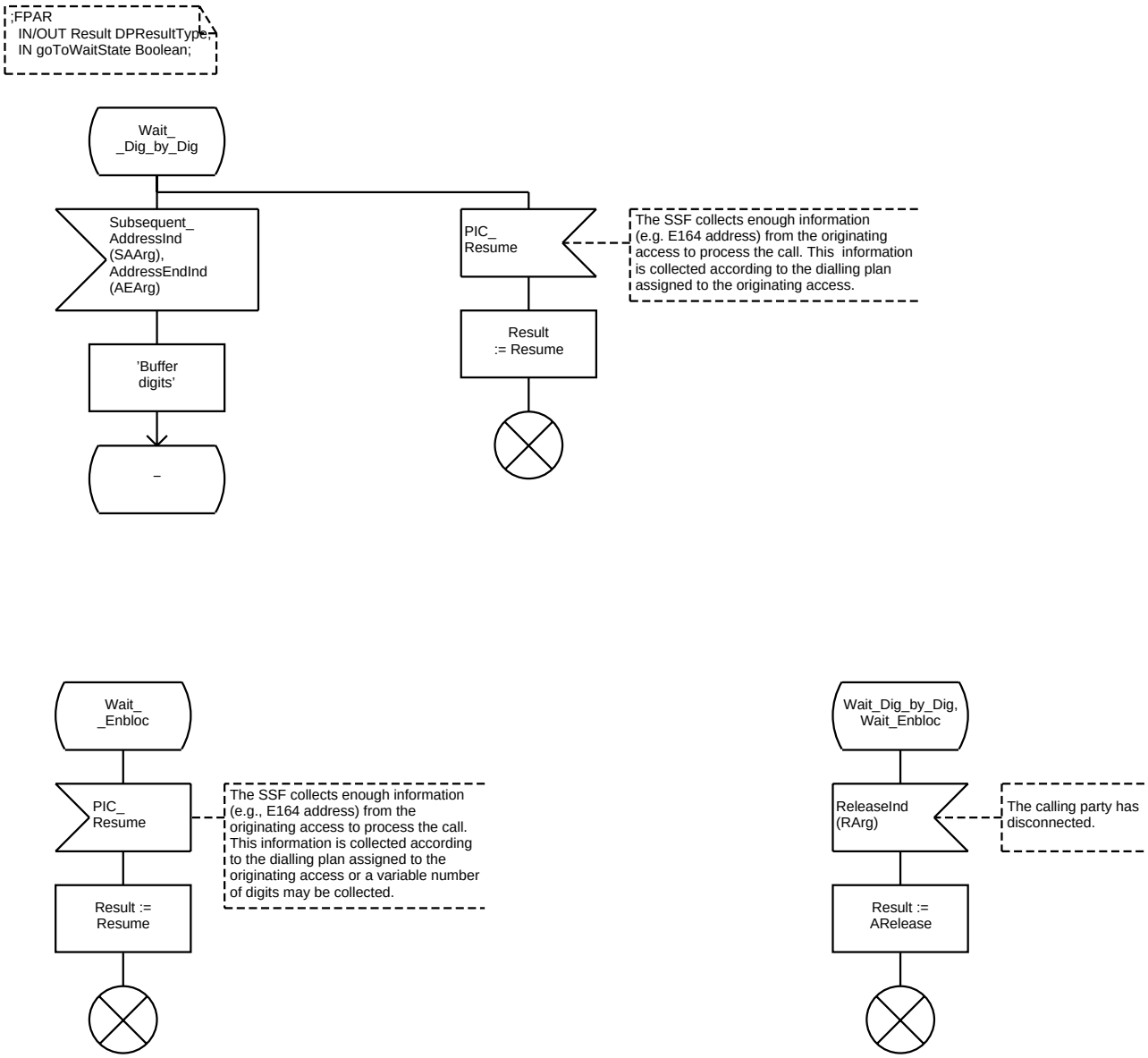

;FPAR
IN/OUT Result PICResultType;
alerting_
Sent
true
false
'Digit collection
not possible'
Result :=
Failure
cpArg!callRef :=
sigConID,
cpArg!cause :=
collectDigits
CallProgressReq
(cpArg)
to CS
Prompts the calling party
for additional address
information.
digits :=
BitStr('')
Wait_for_
_Ndigits

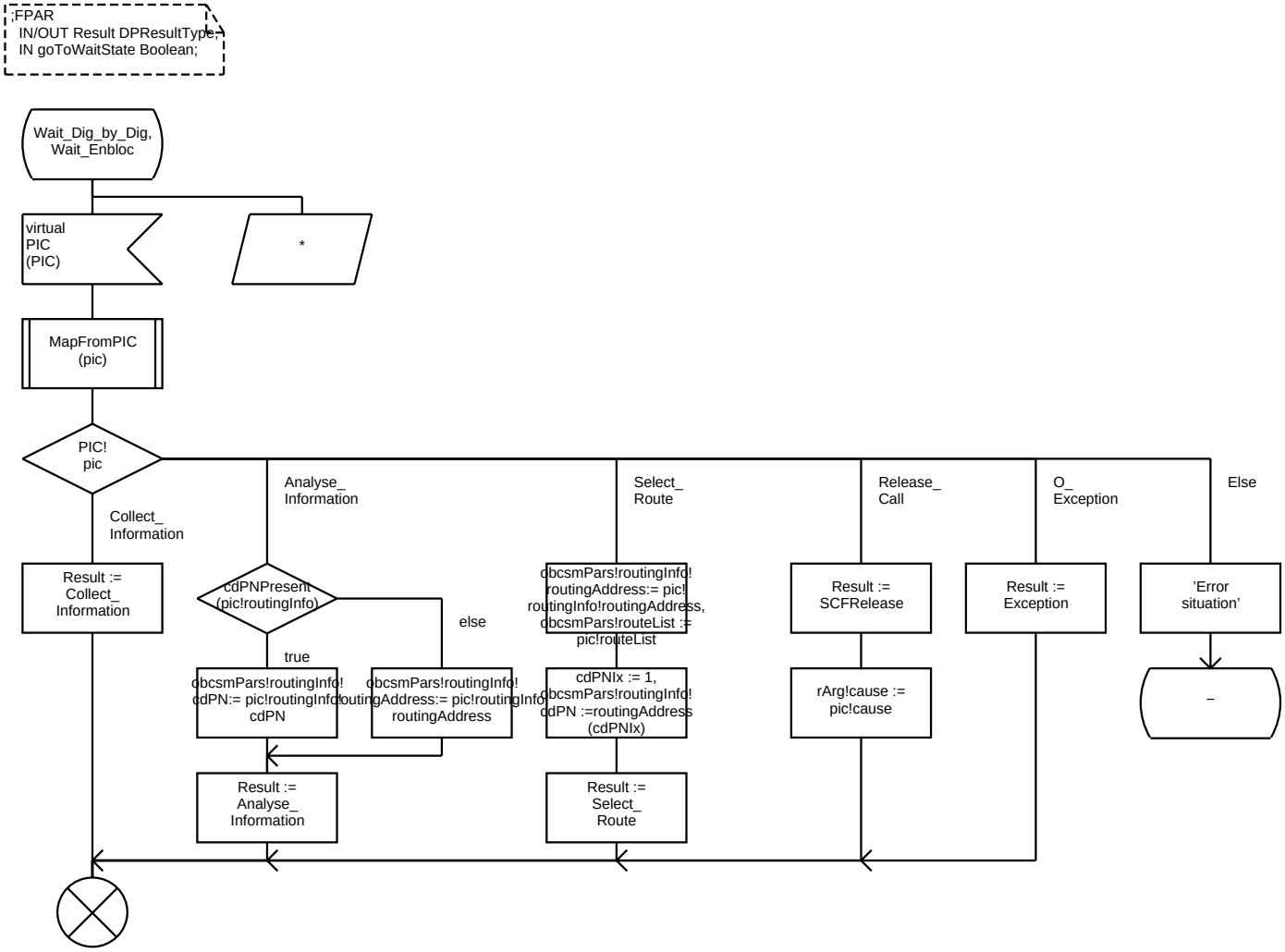

;FPAR
IN/OUT Result PICResultType;
Wait_for_
_Ndigits
Subsequent_
AddressInd
(saArg)
obcsmPars!routingInfo!cdPN :=
obcsmPars!routingInfo!cdPN //
saArg!digits
Append the collected digits
to the calledPartyNumber.
digits := digits //
saArg!digits
Wait_for_
_Ndigits
Address_
EndInd
(AEArg)
setup_
Sent
false
true
saArg!callRef
:= O2I(First(obcsmPars!legID)),
saArg!digits := digits
Subsequent_
AddressReqInd
(SAArg,,obcsmPars!legID)
to CS
Result :=
Success
Release_
Ind
(RArg)
Result :=
ARelease

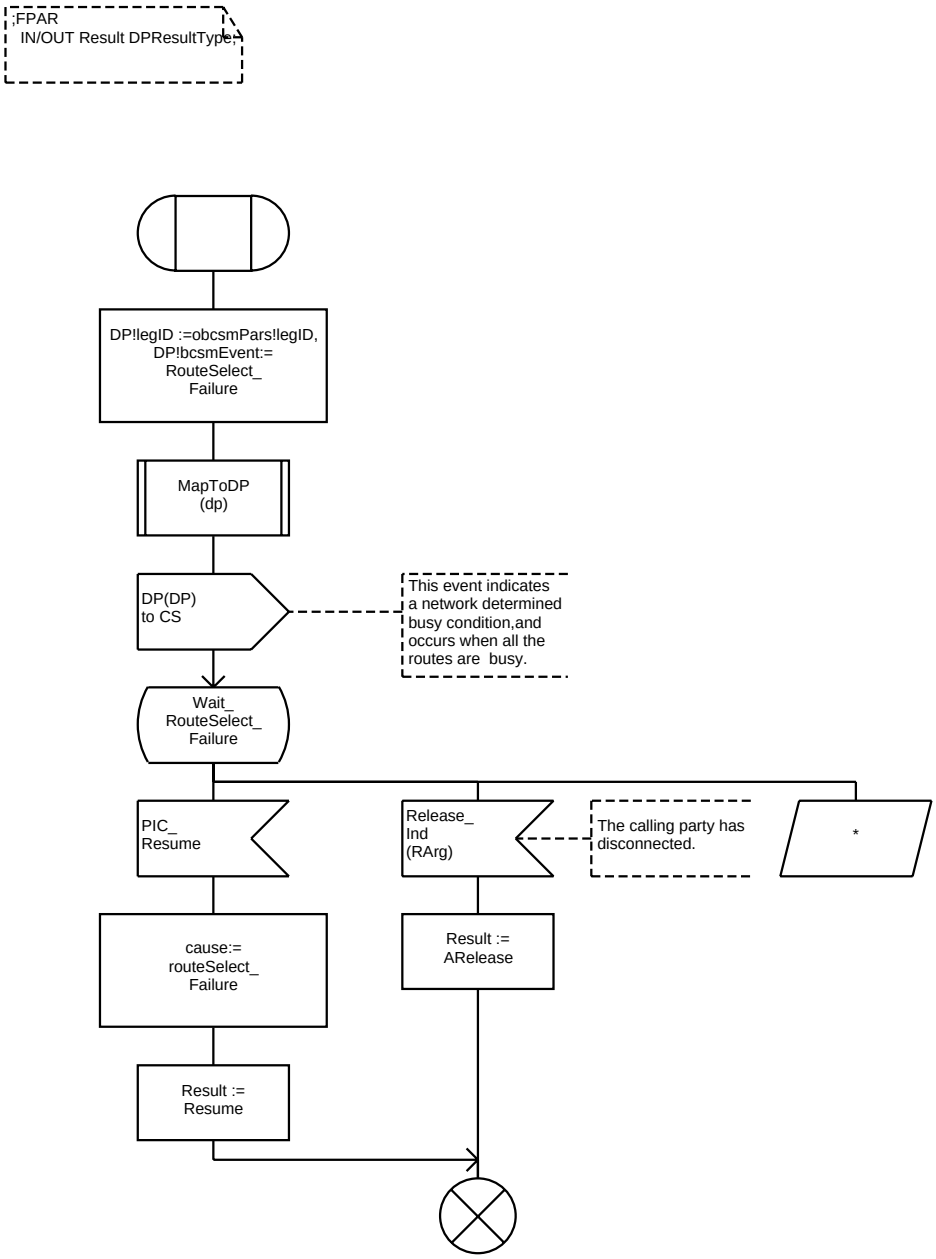

;FPAR
IN/OUT Result PICResultType;
Wait_for_
_Ndigits
PIC
(PIC)
PIC!pic
OException
else
Release_
Call
Result :=
SCFRelease
Result :=
Exception
'Error
condition'
rArg!cause :=
pic!cause
_

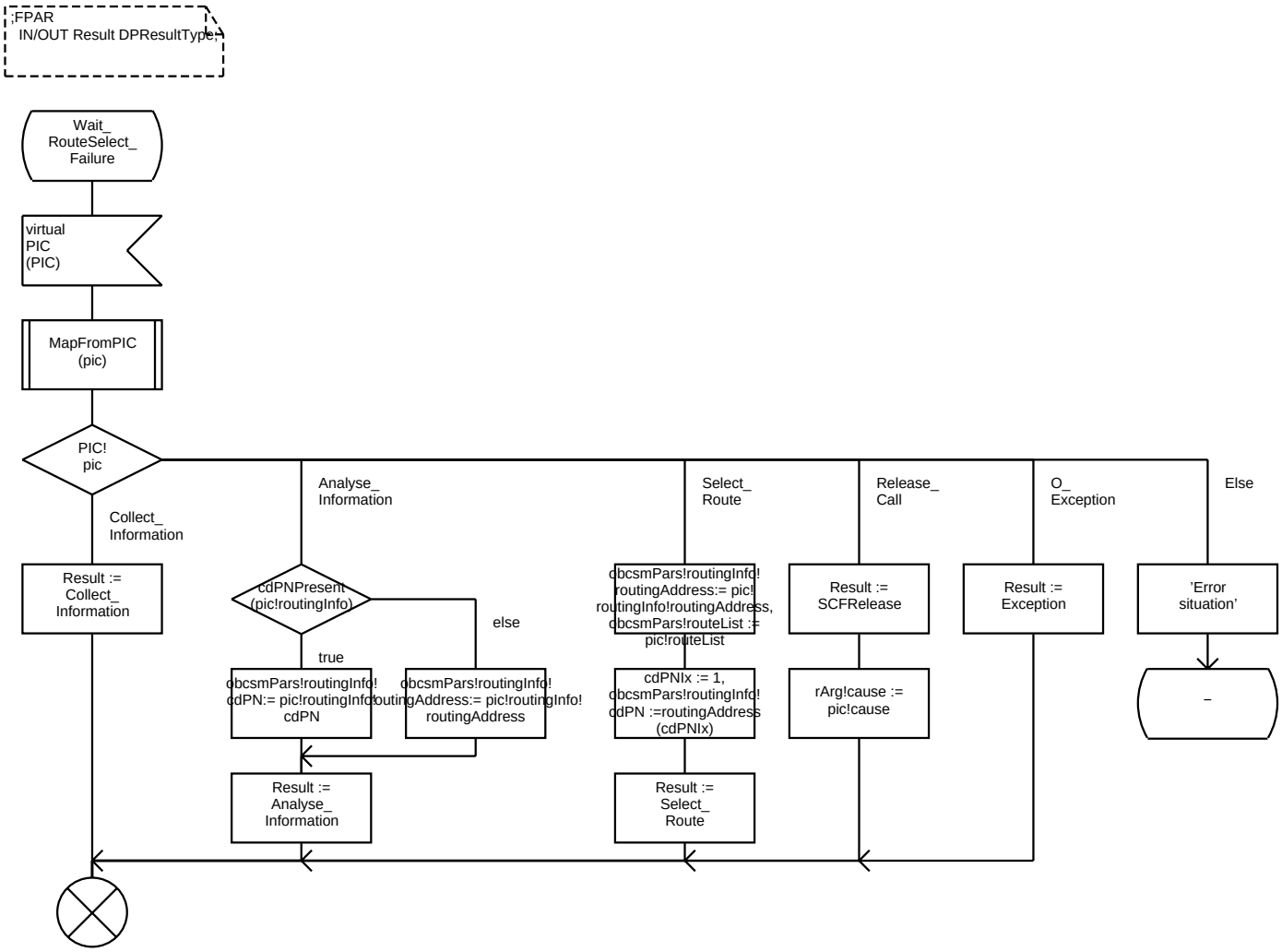
```
;FPAR
  IN/OUT Result DPResultType,
  IN goToWaitState Boolean;
```



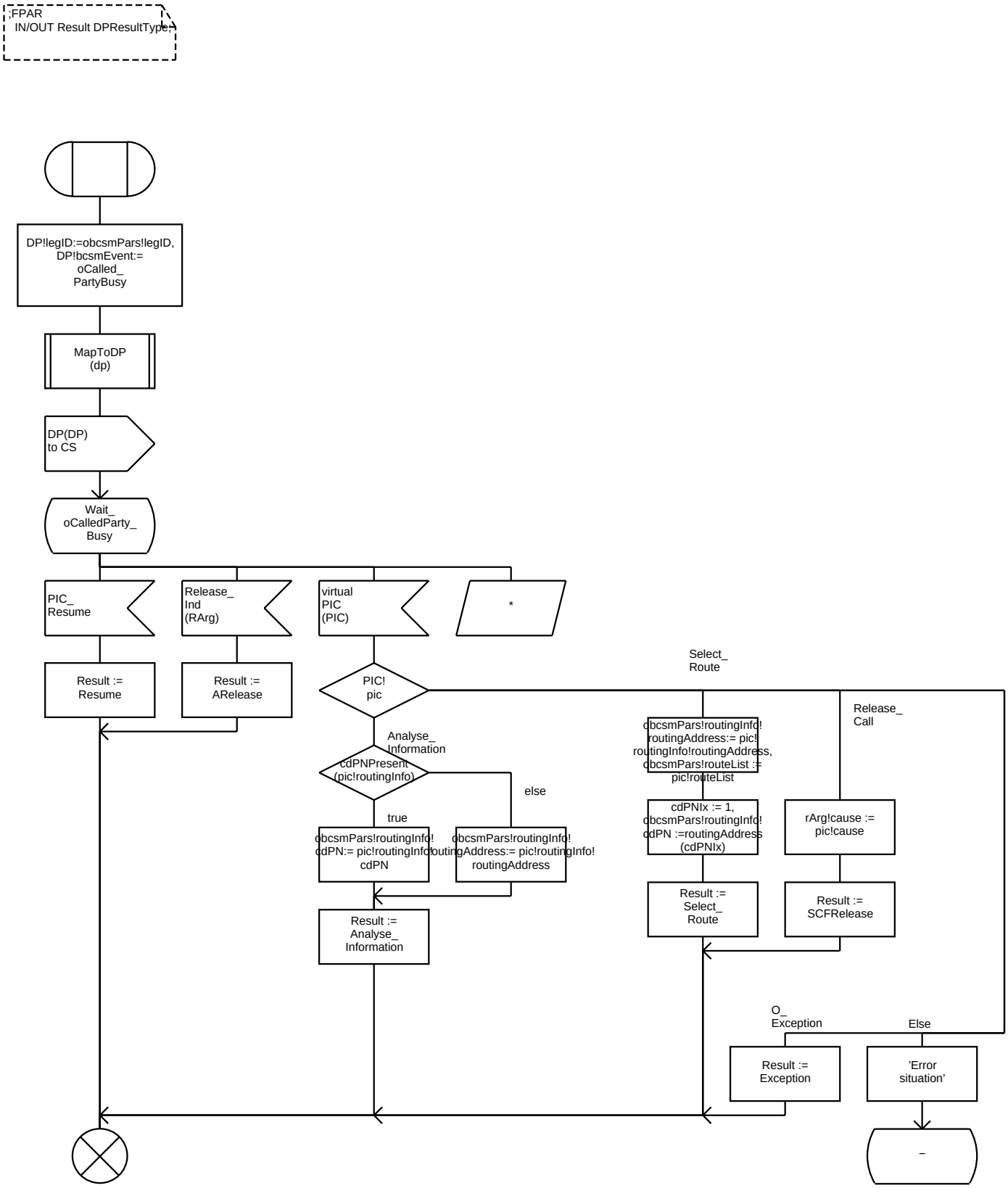

;FPAR
IN/OUT Result DPResultType;
IN goToWaitState Boolean;
Wait_
_Dig_by_Dig
Subsequent_
AddressInd
(SAArg),
AddressEndInd
(AEArg)
'Buffer
digits'
_
PIC_
Resume
The SSF collects enough information (e.g. E164 address) from the originating access to process the call. This information is collected according to the dialling plan assigned to the originating access.
Result
:= Resume
Wait_
_Enbloc
PIC_
Resume
The SSF collects enough information (e.g., E164 address) from the originating access to process the call. This information is collected according to the dialling plan assigned to the originating access or a variable number of digits may be collected.
Result :=
Resume
Wait_Dig_by_Dig,
Wait_Enbloc
ReleaseInd
(RArg)
The calling party has disconnected.
Result :=
ARelease


;FPAR
IN/OUT Result DPResultType,
IN goToWaitState Boolean;
Wait_Dig_by_Dig, Wait_Enbloc
virtual PIC (PIC)
*
MapFromPIC (pic)
PIC! pic
Collect_ Information
Analyse_ Information
Select_ Route
Release_ Call
O_ Exception
Else
Result := Collect_ Information
cdPNPresent (pic!routingInfo)
else
true
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
obcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
Result := Analyse_ Information
obcsmPars!routingInfo! routingAddress:= pic! routingInfo!routingAddress, obcsmPars!routeList := pic!routeList
cdPNIx := 1, obcsmPars!routingInfo! cdPN :=routingAddress (cdPNIx)
Result := Select_ Route
Result := SCFRelease
rArg!cause := pic!cause
Result := Exception
'Error situation'
–

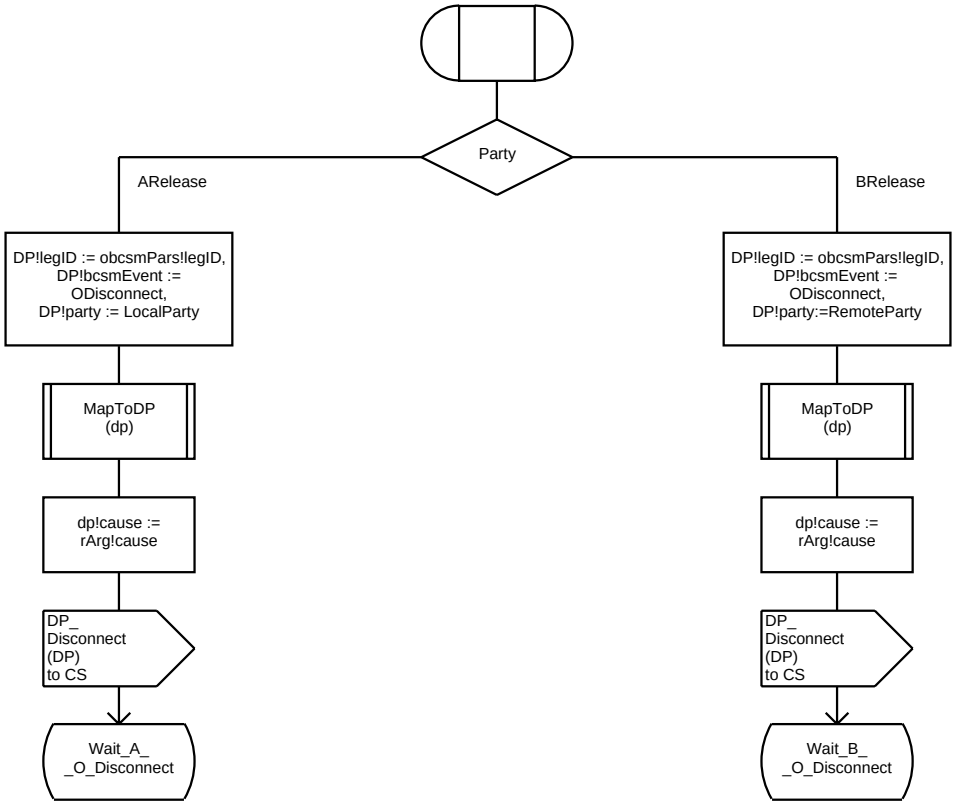

;FPAR
IN/OUT Result DPResultType;
DP!legID :=obcsmPars!legID,
DP!bcsmEvent:=
RouteSelect_
Failure
MapToDP
(dp)
DP(DP)
to CS
This event indicates
a network determined
busy condition,and
occurs when all the
routes are busy.
Wait_
RouteSelect_
Failure
PIC_
Resume
Release_
Ind
(RArg)
The calling party has
disconnected.
*
cause:=
routeSelect_
Failure
Result :=
ARelease
Result :=
Resume

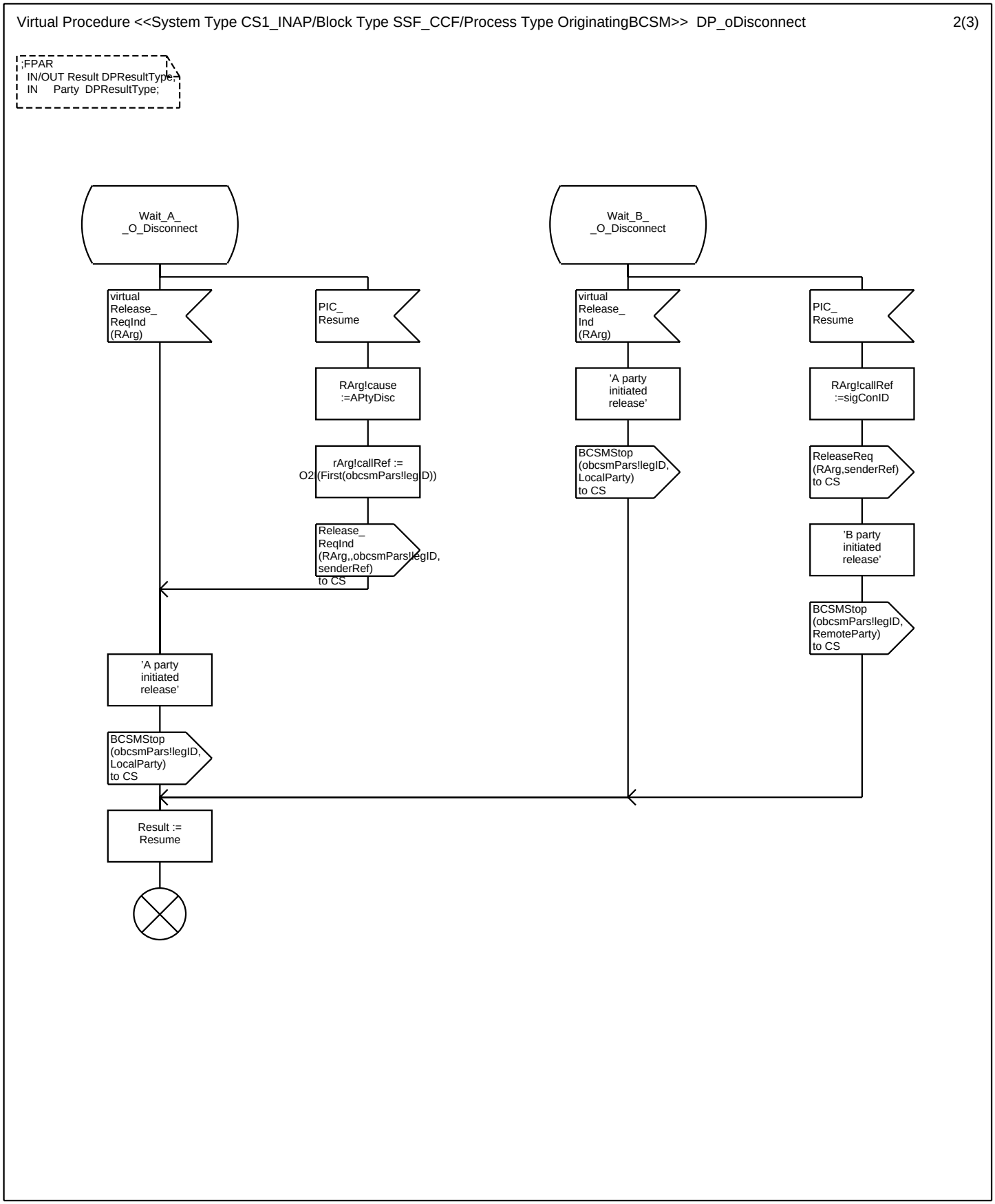

;FPAR
IN/OUT Result DPResultType;
Wait_
RouteSelect_
Failure
virtual
PIC
(PIC)
MapFromPIC
(pic)
PIC!
pic
Collect_
Information
Analyse_
Information
Select_
Route
Release_
Call
O_
Exception
Else
Result :=
Collect_
Information
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Analyse_
Information
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
Result :=
SCFRelease
rArg!cause :=
pic!cause
Result :=
Exception
'Error
situation'
_

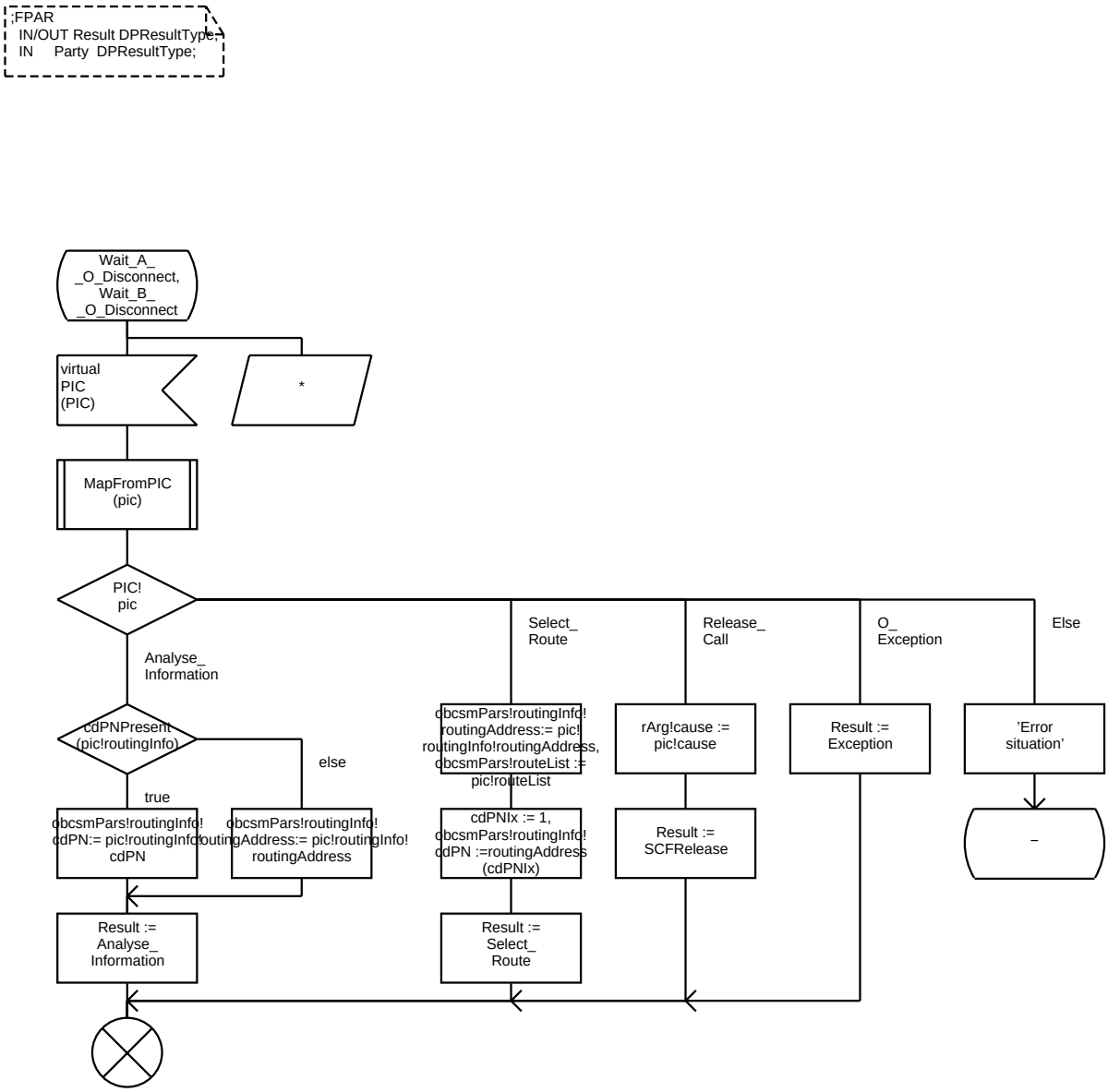
Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type OriginatingBCSM>> DP_oCalledPartyBusy 1(1)



```
;FPAR
  IN/OUT Result DPResultType;
  IN     Party  DPResultType;
```

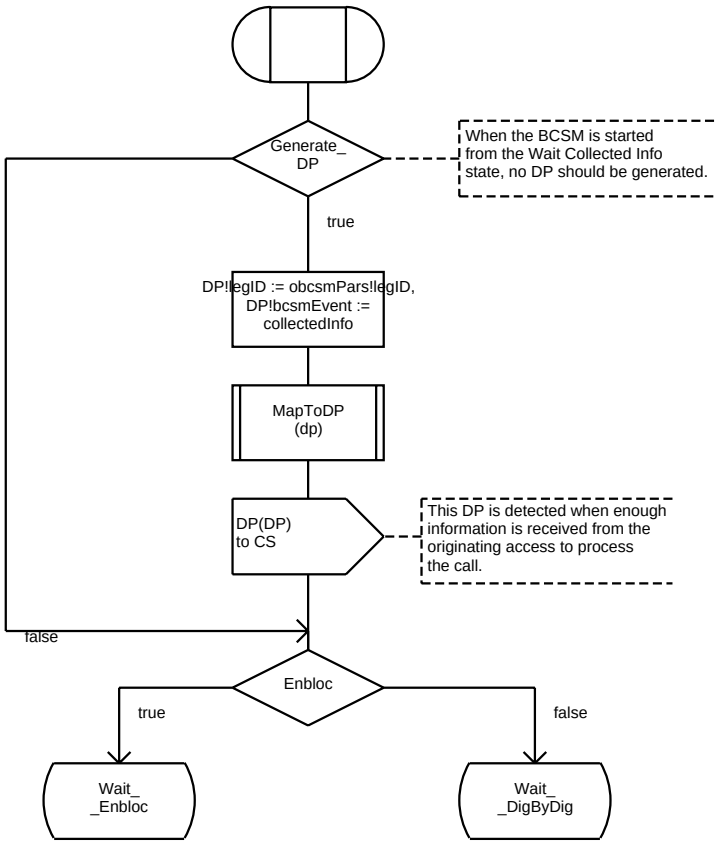


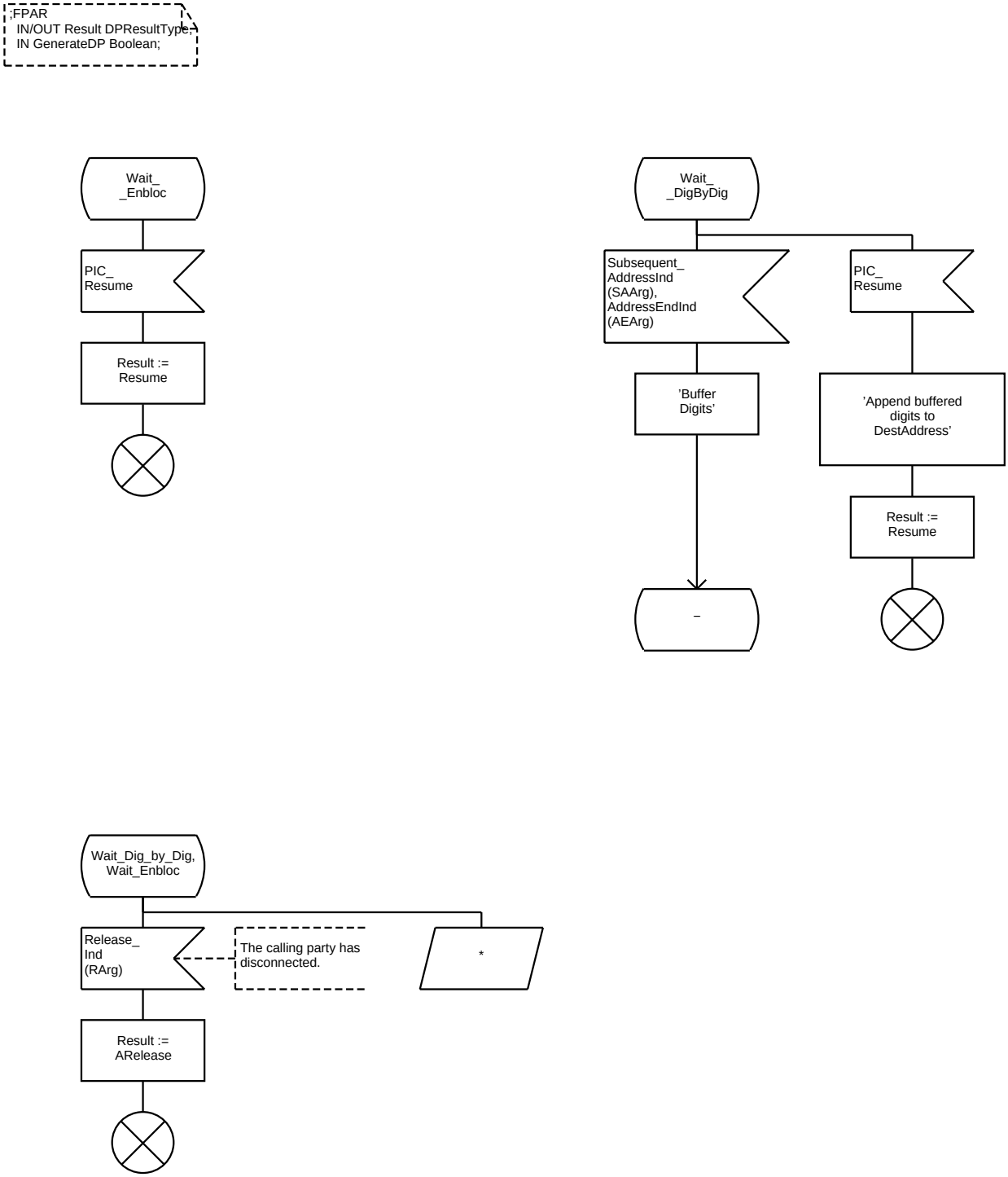

;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Wait_A_
_O_Disconnect
virtual
Release_
ReqInd
(RArg)
PIC_
Resume
RArg!cause
:=APtyDisc
rArg!callRef :=
O2(First(obcsmPars!legID))
Release_
ReqInd
(RArg,,obcsmPars!legID,
senderRef)
to CS
'A party
initiated
release'
BCSMStop
(obcsmPars!legID,
LocalParty)
to CS
Result :=
Resume
Wait_B_
_O_Disconnect
virtual
Release_
Ind
(RArg)
PIC_
Resume
'A party
initiated
release'
RArg!callRef
:=sigConID
BCSMStop
(obcsmPars!legID,
LocalParty)
to CS
ReleaseReq
(RArg,senderRef)
to CS
'B party
initiated
release'
BCSMStop
(obcsmPars!legID,
RemoteParty)
to CS

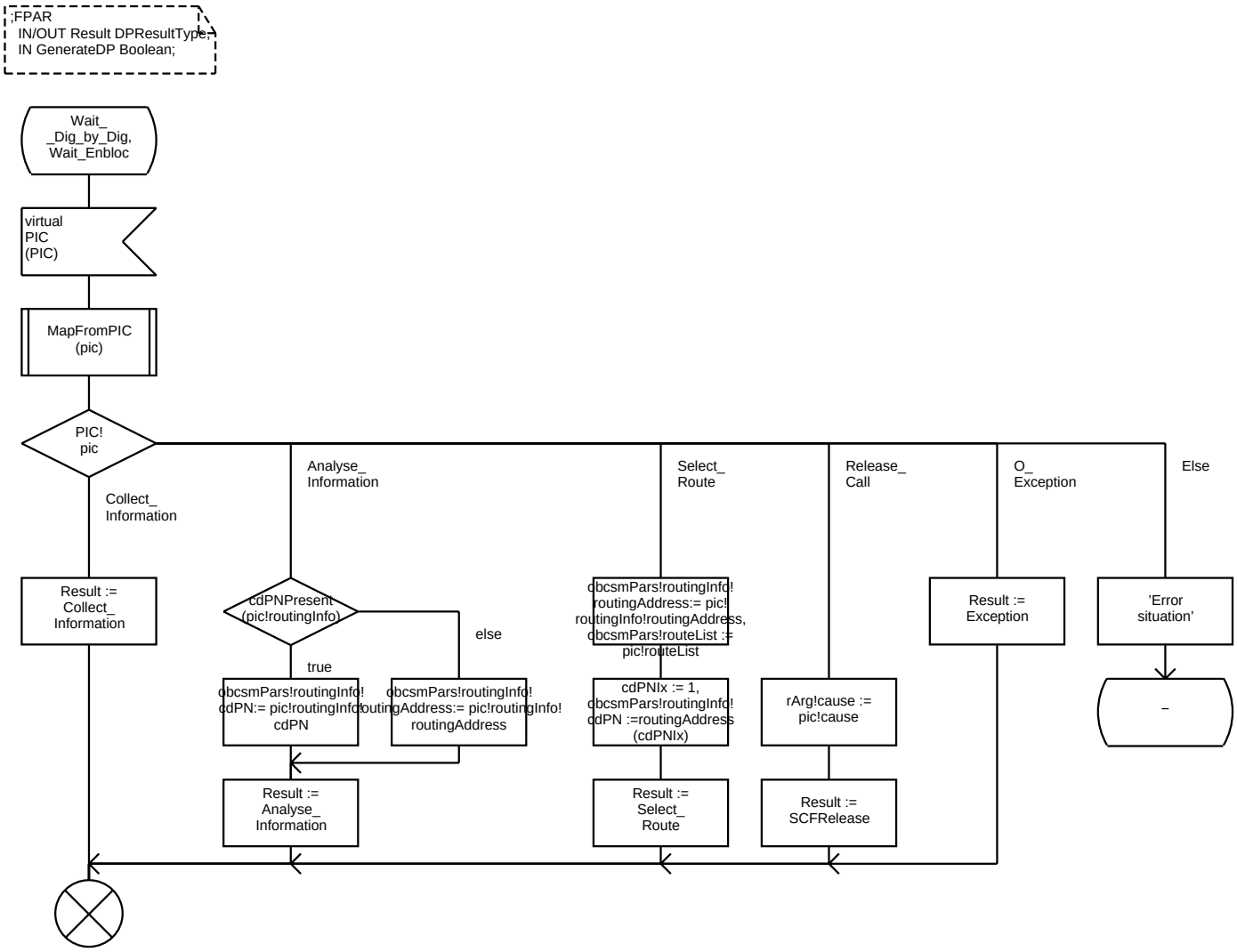

;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Wait_A_
_O_Disconnect,
Wait_B_
_O_Disconnect
virtual
PIC
(PIC)
*
MapFromPIC
(pic)
PIC!
pic
Analyse_
Information
Select_
Route
Release_
Call
O_
Exception
Else
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Analyse_
Information
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
rArg!cause :=
pic!cause
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
_

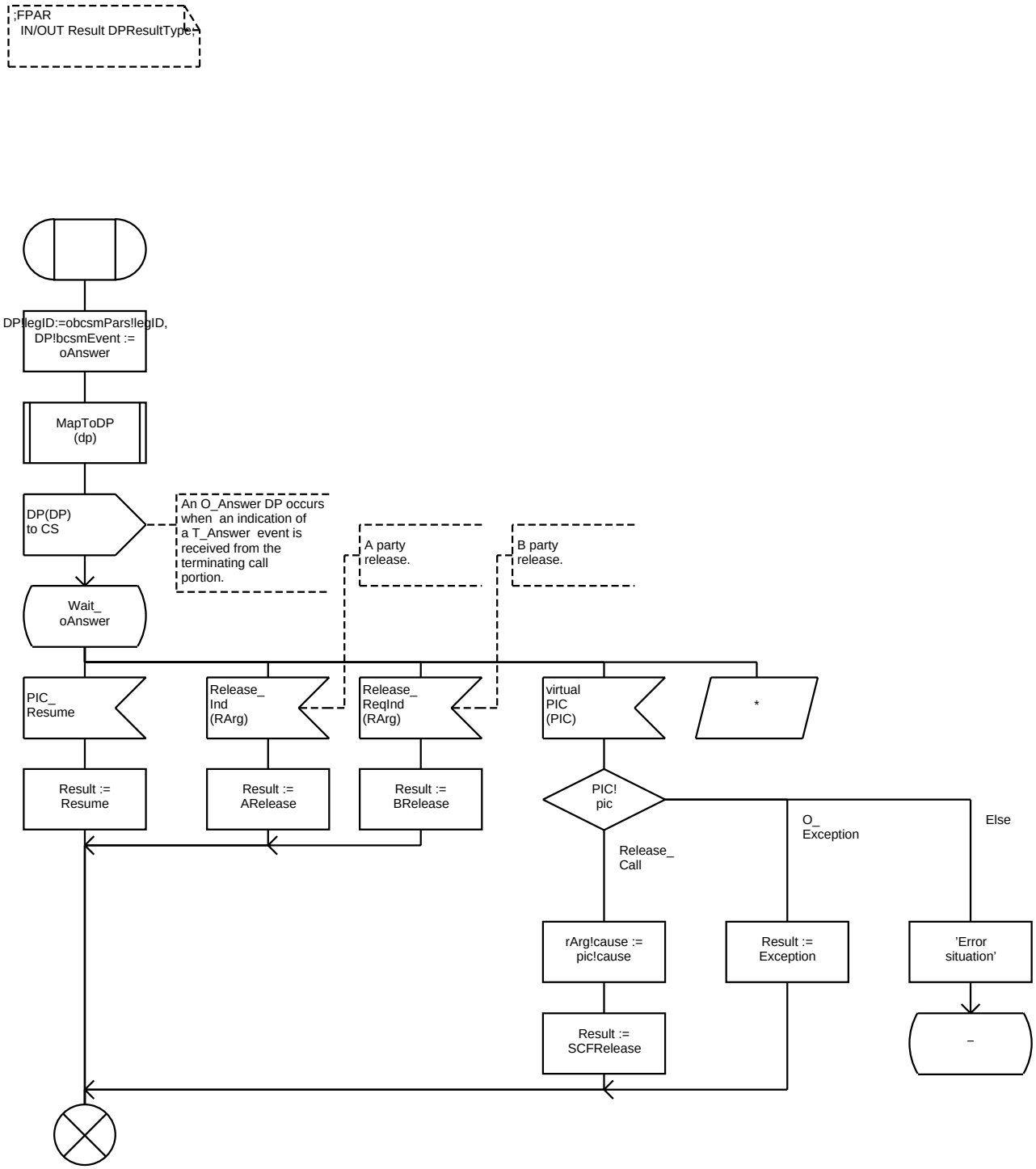
```
;FPAR
  IN/OUT Result DPResultType,
  IN GenerateDP Boolean;
```

```
/*
DP Collected_Information occurs when there is
sufficient information available to start outgoing
setup. Outgoing call setup is now possible but
cannot proceed until SCF has adviced SSF either
to continue call setup with dialled digits or to
replace them with digits supplied by SCF. Call
setup commences after Analysed_Information DP.

In digit by digit case, the user may be continuing
to dial further digits and these must be stored, to
be sent forward later.
*/
```

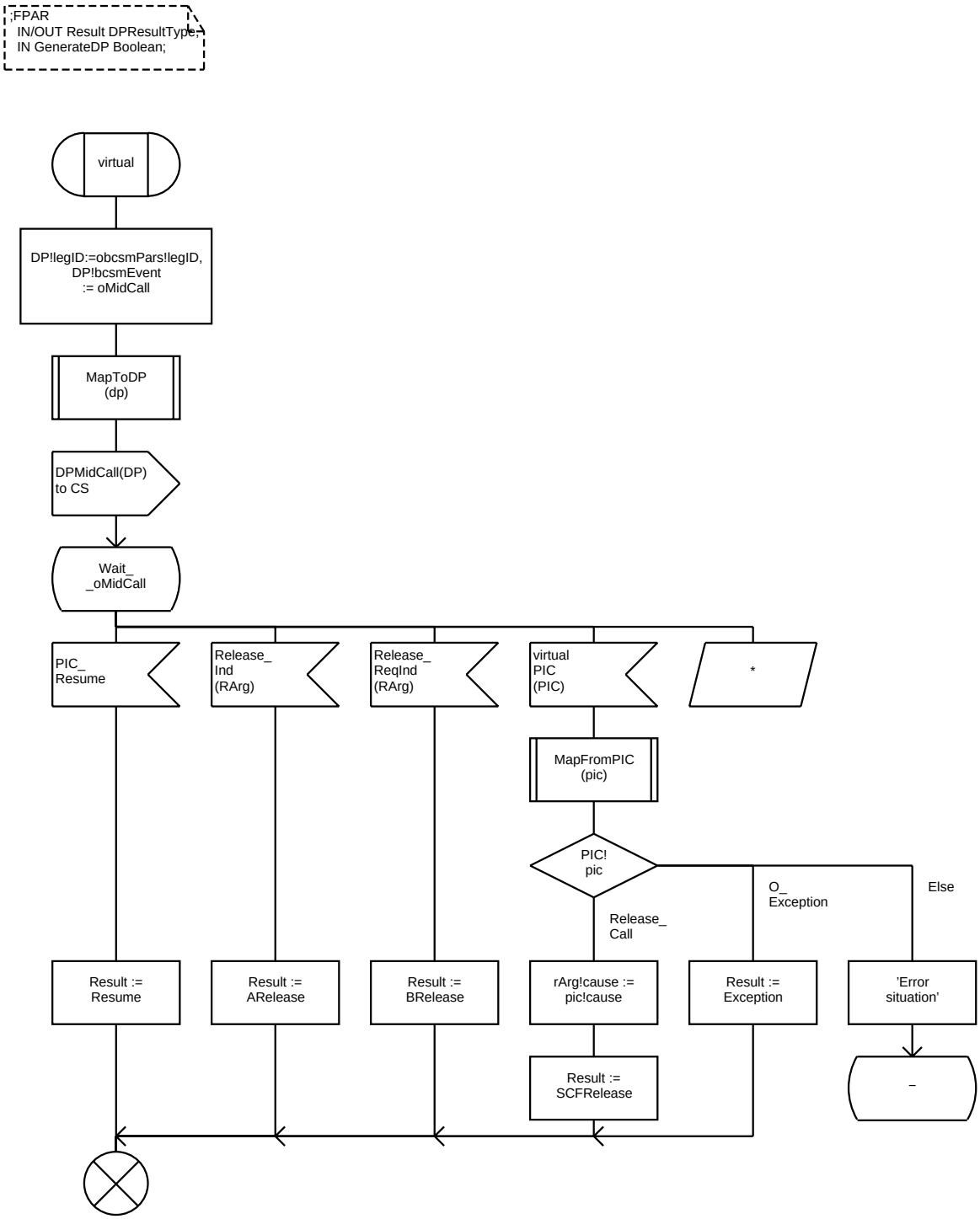


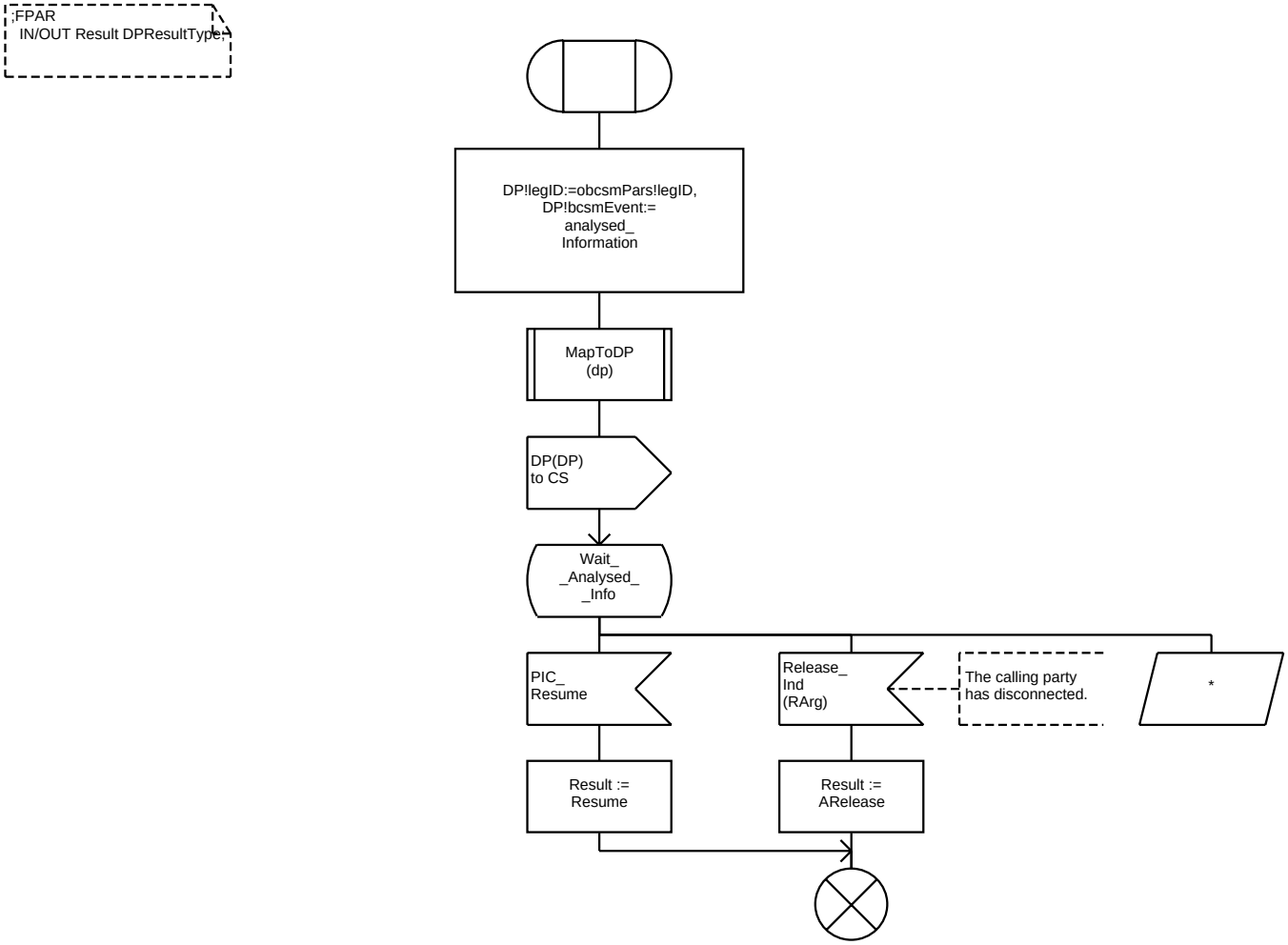

;FPAR
IN/OUT Result DPResultType,
IN GenerateDP Boolean;
Wait_
_Enbloc
PIC_
Resume
Result :=
Resume
Wait_
_DigByDig
Subsequent_
AddressInd
(SAArg),
AddressEndInd
(AEArg)
PIC_
Resume
'Buffer
Digits'
'Append buffered
digits to
DestAddress'
Result :=
Resume
–
Wait_Dig_by_Dig,
Wait_Enbloc
Release_
Ind
(RArg)
The calling party has
disconnected.
*
Result :=
ARelease

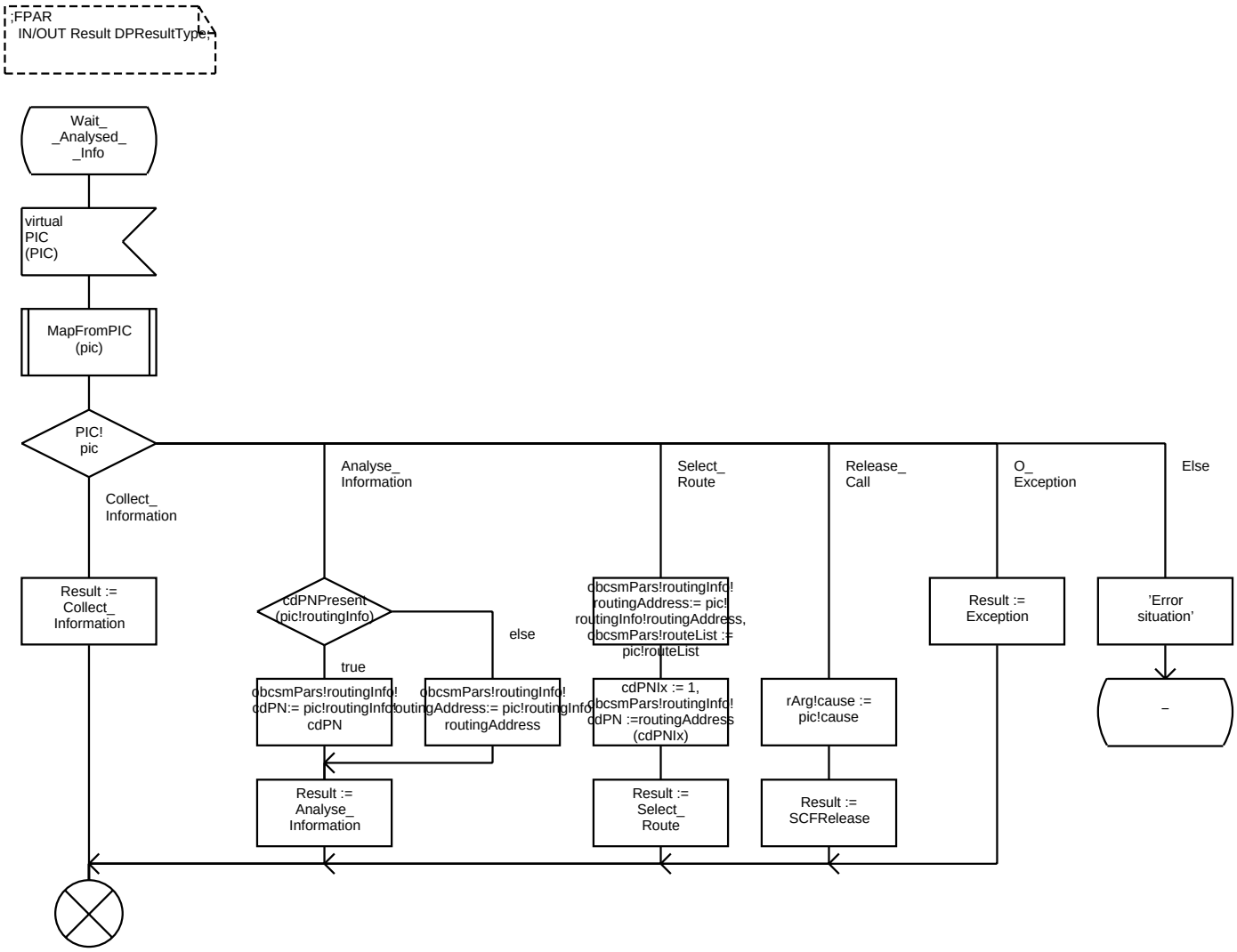

;FPAR
IN/OUT Result DPResultType,
IN GenerateDP Boolean;
Wait_
_Dig_by_Dig,
Wait_Enbloc
virtual
PIC
(PIC)
MapFromPIC
(pic)
PIC!
pic
Collect_
Information
Analyse_
Information
Select_
Route
Release_
Call
O_
Exception
Else
Result :=
Collect_
Information
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Analyse_
Information
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
rArg!cause :=
pic!cause
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
–


;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
oAnswer
MapToDP
(dp)
DP(DP)
to CS
An O_Answer DP occurs when an indication of a T_Answer event is received from the terminating call portion.
A party release.
B party release.
Wait_
oAnswer
PIC_
Resume
Release_
Ind
(RArg)
Release_
ReqInd
(RArg)
virtual
PIC
(PIC)
*
Result :=
Resume
Result :=
ARelease
Result :=
BRelease
PIC!
pic
Release_
Call
O_
Exception
Else
rArg!cause :=
pic!cause
Result :=
Exception
'Error
situation'
Result :=
SCFRelease
_

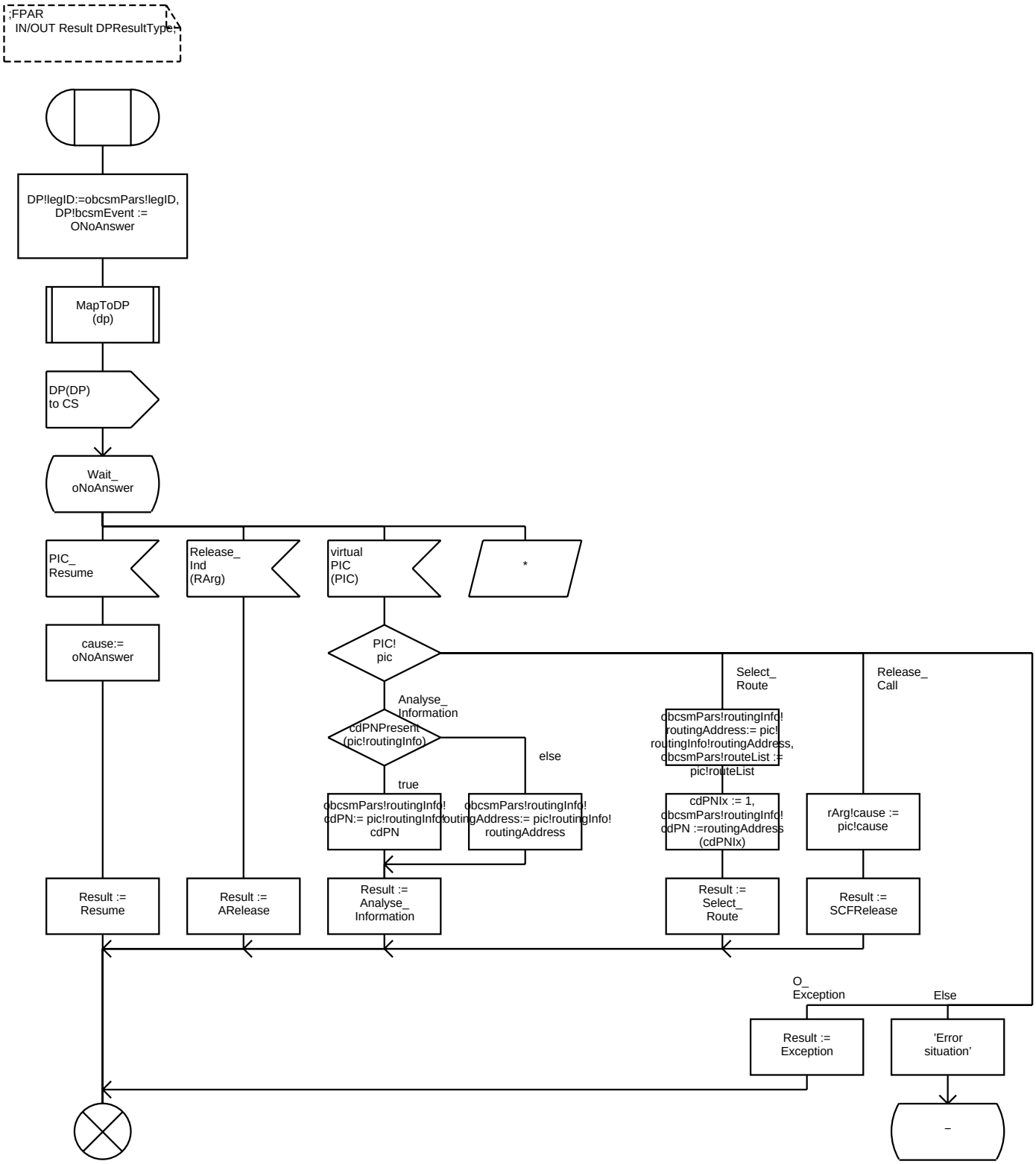
Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type OriginatingBCSM>> DP_oMidCall 1(1)

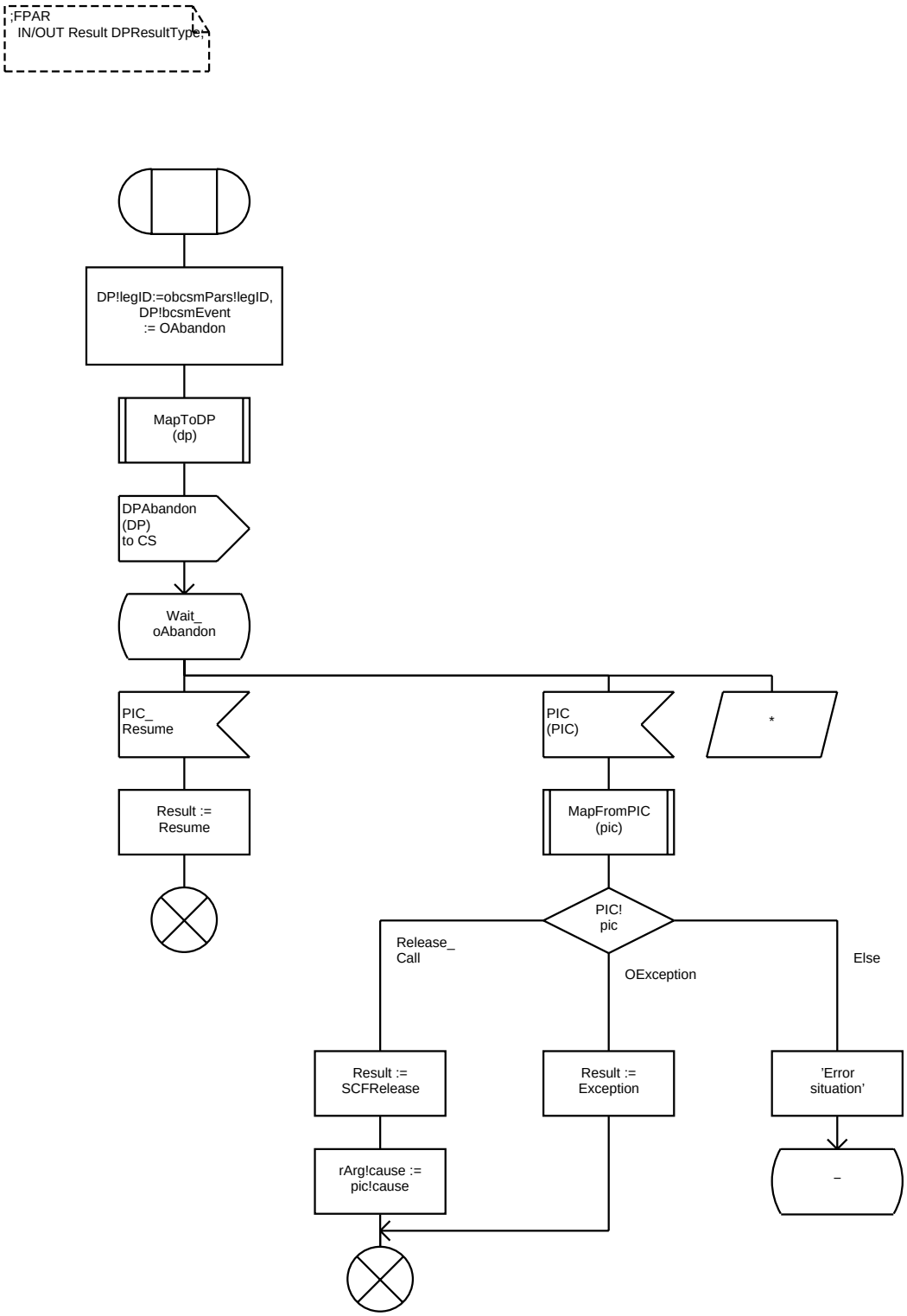



;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent:=
analysed_
Information
MapToDP
(dp)
DP(DP)
to CS
Wait_
Analysed
_Info
PIC_
Resume
Release_
Ind
(RArg)
The calling party
has disconnected.
*
Result :=
Resume
Result :=
ARelease

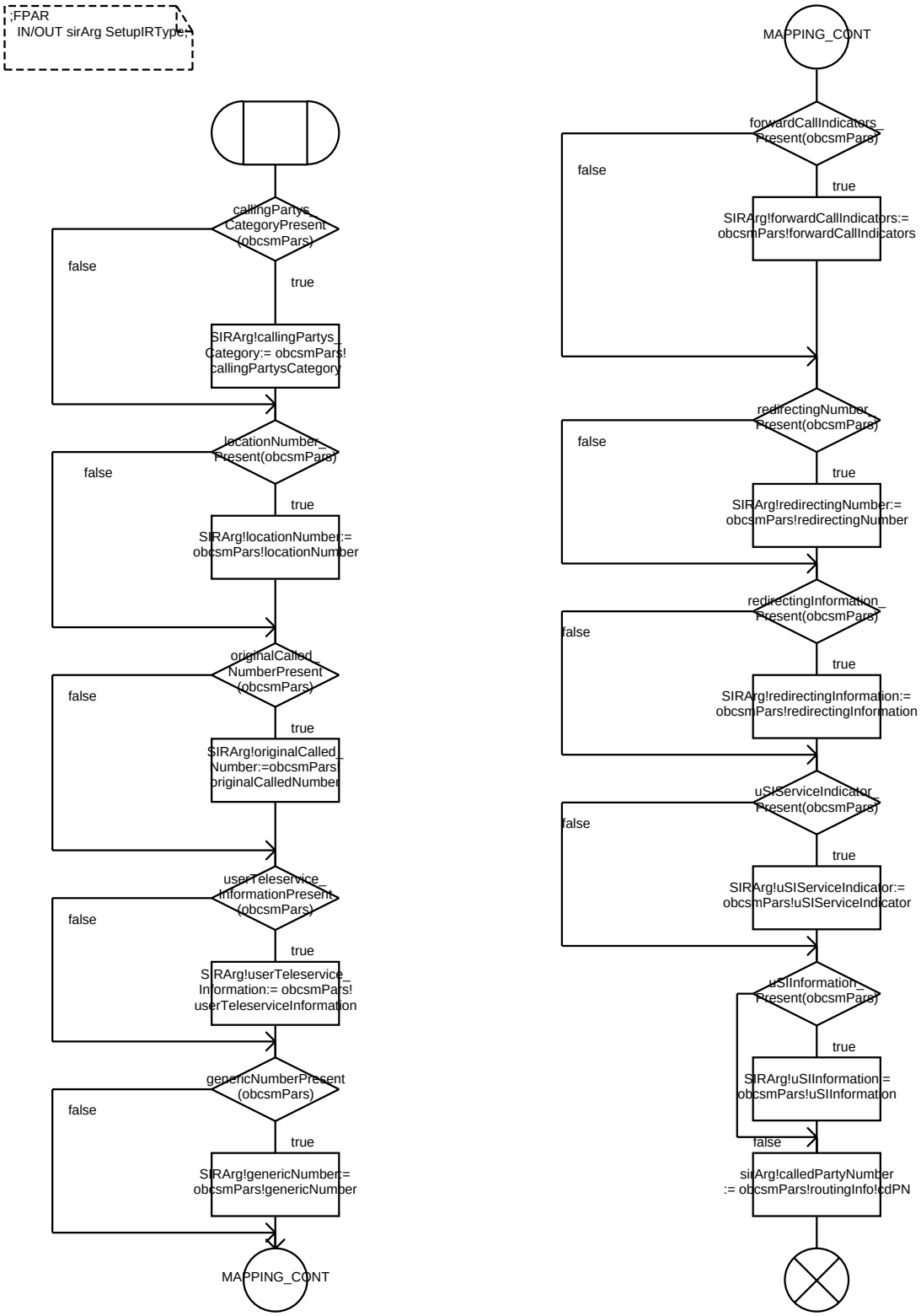

;FPAR
IN/OUT Result DPResultType;
Wait_
Analysed
_Info
virtual
PIC
(PIC)
MapFromPIC
(pic)
PIC!
pic
Collect_
Information
Analyse_
Information
Select_
Route
Release_
Call
O_
Exception
Else
Result :=
Collect_
Information
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Analyse_
Information
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
rArg!cause :=
pic!cause
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
–

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type OriginatingBCSM>> DP_oNoAnswer 1(1)

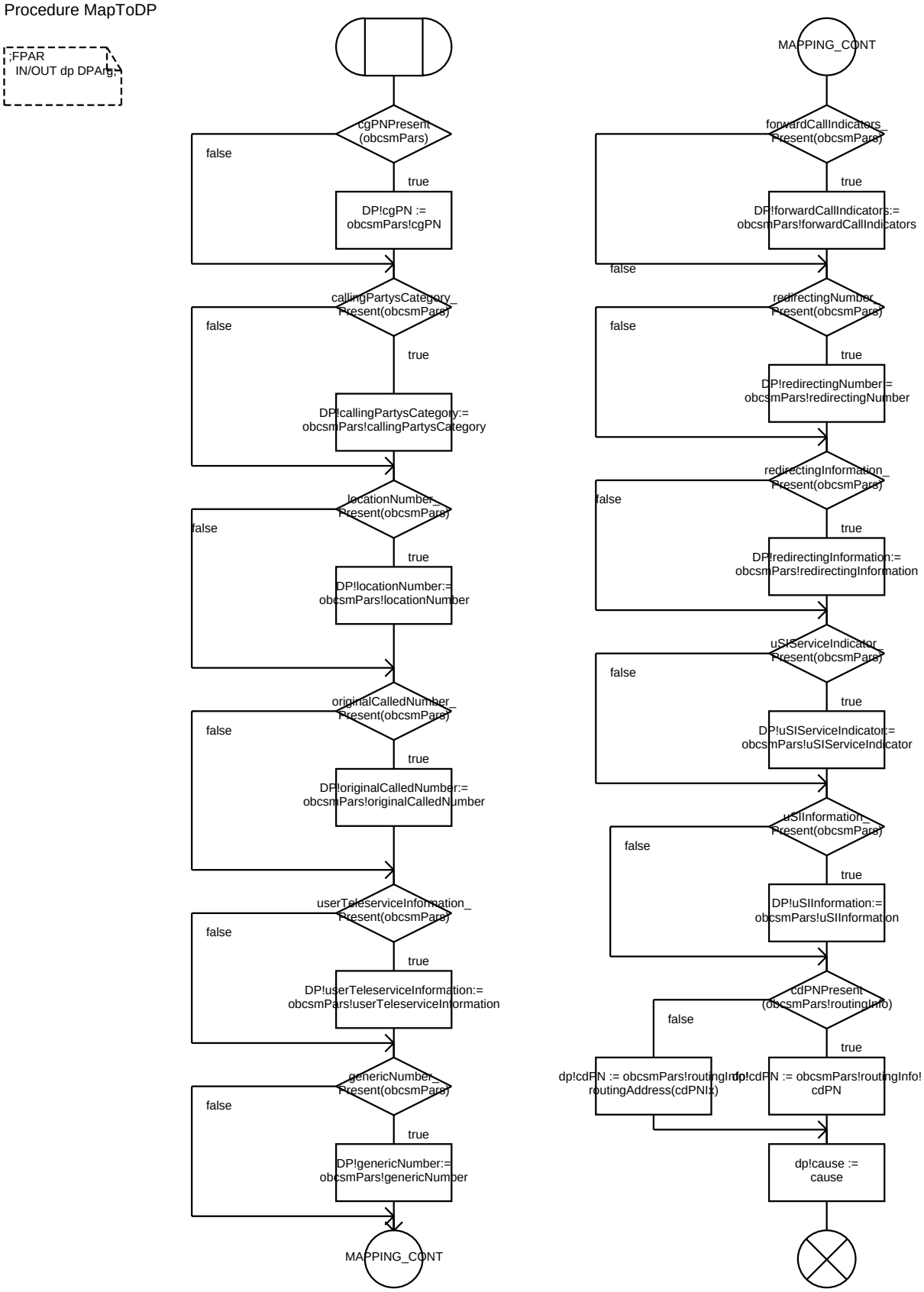



;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent
:= OAbandon
MapToDP
(dp)
DPAbandon
(DP)
to CS
Wait_
oAbandon
PIC_
Resume
PIC
(PIC)
*
Result :=
Resume
MapFromPIC
(pic)
PIC!
pic
Release_
Call
OException
Else
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
rArg!cause :=
pic!cause
–

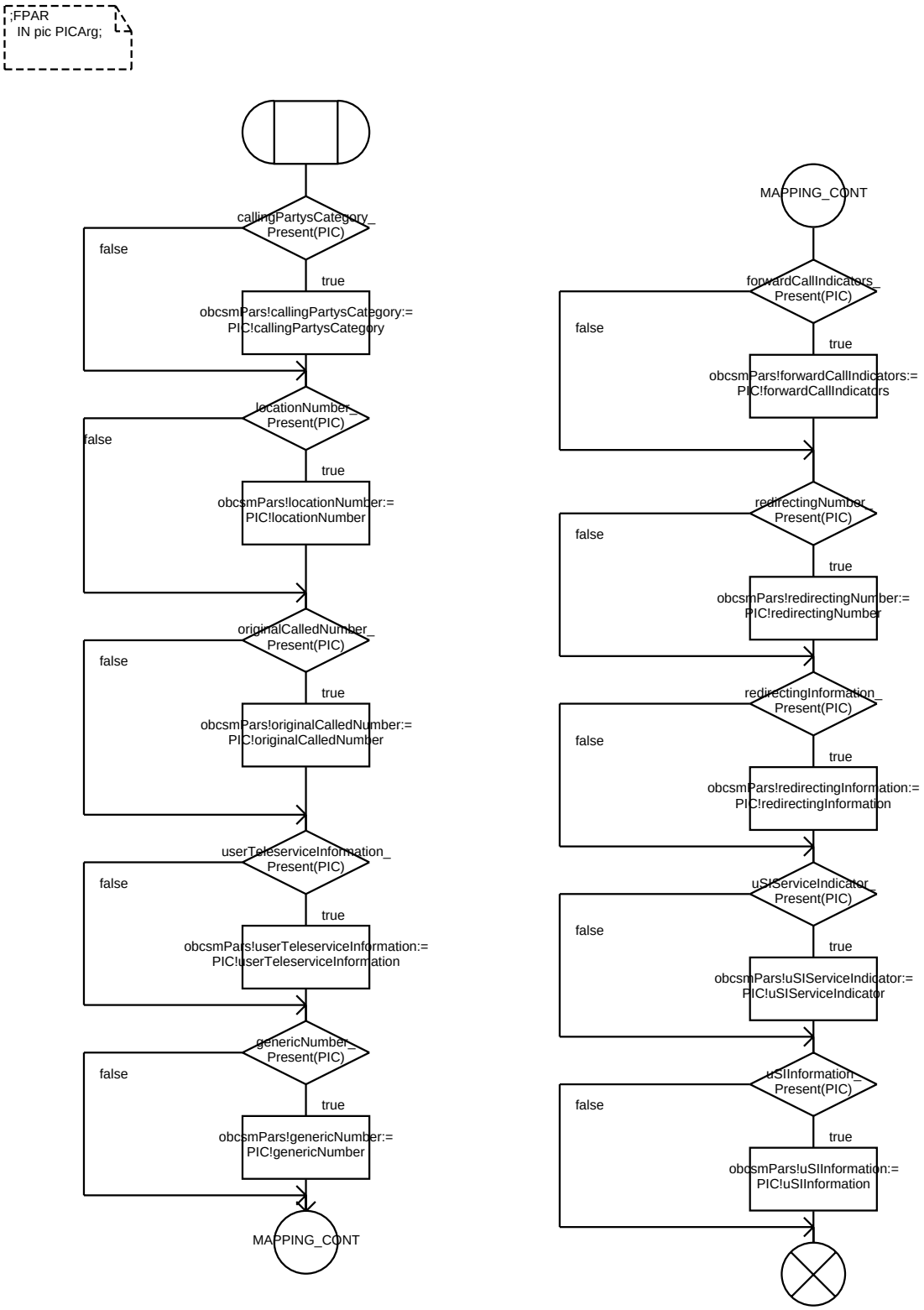
Procedure MapToSIRArg



Procedure MapToDP



Procedure MapFromPIC

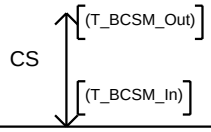


```
;FPAR
  StartState BCSMStateType,   /* The start state of the BCSM. */
  LegId LegType;              /* The LegID as assigned by the CS. */
```

```
/***** T−BCSM FOR CORE INAP CS−1. *****/
```

```
/***** DATA TYPE DEFINITIONS *****/
```

```
NEWTYPE DPResultType
  LITERALS
    Resume,
    SelectFacility,
    PresentCall,
    ARelease,
    BRelease,
    MidCall,
    SCFRelease, /* Used with ReleaseCall. */
    Answer,
    Alerting,


    DL_A, DL_B, /* Used with DisconnectLeg */
    CS_Stop;
ENDNEWTYPE;

NEWTYPE PICResultType
  LITERALS
    Failure,
    Success,
    ARelease,
    BRelease,
    Alerting,
    Busy,
    Answer,
    NoAnswer,
    MidCall,
    Exception,
    SCFRelease, /* Used with ReleaseCall. */

    Reanswer,
    Suspended,
    DL_A, DL_B; /* Used with DisconnectLeg */
ENDNEWTYPE;
```




```
;FPAR
 StartState BCSMStateType,   /* The start state of the BCSM. */
 LegId LegType;              /* The LegID as assigned by the CS. */
```

```
/**** VARIABLE AND TIMER DECLARATIONS ****/
```

```
TIMER
 NoAnswerT := 15000,
 SuspendT  := 3600000;

DCL
 /* Pointer to the call segment. */
 CS PId,

 /* Address of remote O−BCSM. */
 remLegID LegType,
 remCSAID CSAID,

 calledPartyNumber CalledPartyNumber,


 /* Other variables. */
 PICResult PICResultType,
 DPResult DPResultType,
 DPCause Cause,
 PIC PICArg,
 DP DPArg;
```

```
DCL
 /* SigCon primitive parameters. */
 aeArg AddressEndType,
 cpArg CallProgressType,
 farg FailureType,
 rarg ReleaseType,
 sftArg ServiceFeatureType,
 sirArg SetupIRType,
 scrArg SetupCRType,
 saArg SubsequentAddressType;
```

```
;FPAR
 StartState BCSMStateType,   /* The start state of the BCSM. */
 LegId LegType;              /* The LegID as assigned by the CS. */
```

/* Procedure definitions for Points In Call (PICs) */

PIC_T_
_Null

PIC_Select_
_Facility

virtual
PIC_tActive

PIC_Authorize_
Termination
_Attempt

virtual
PIC_Present_
_Call

PIC_
_TException

virtual
PIC_
_tAlerting

/* Procedure definitions for Detection Points (DPs) */

virtual
DP_termAttempt_
Authorized

virtual
DP_tNoAnswer

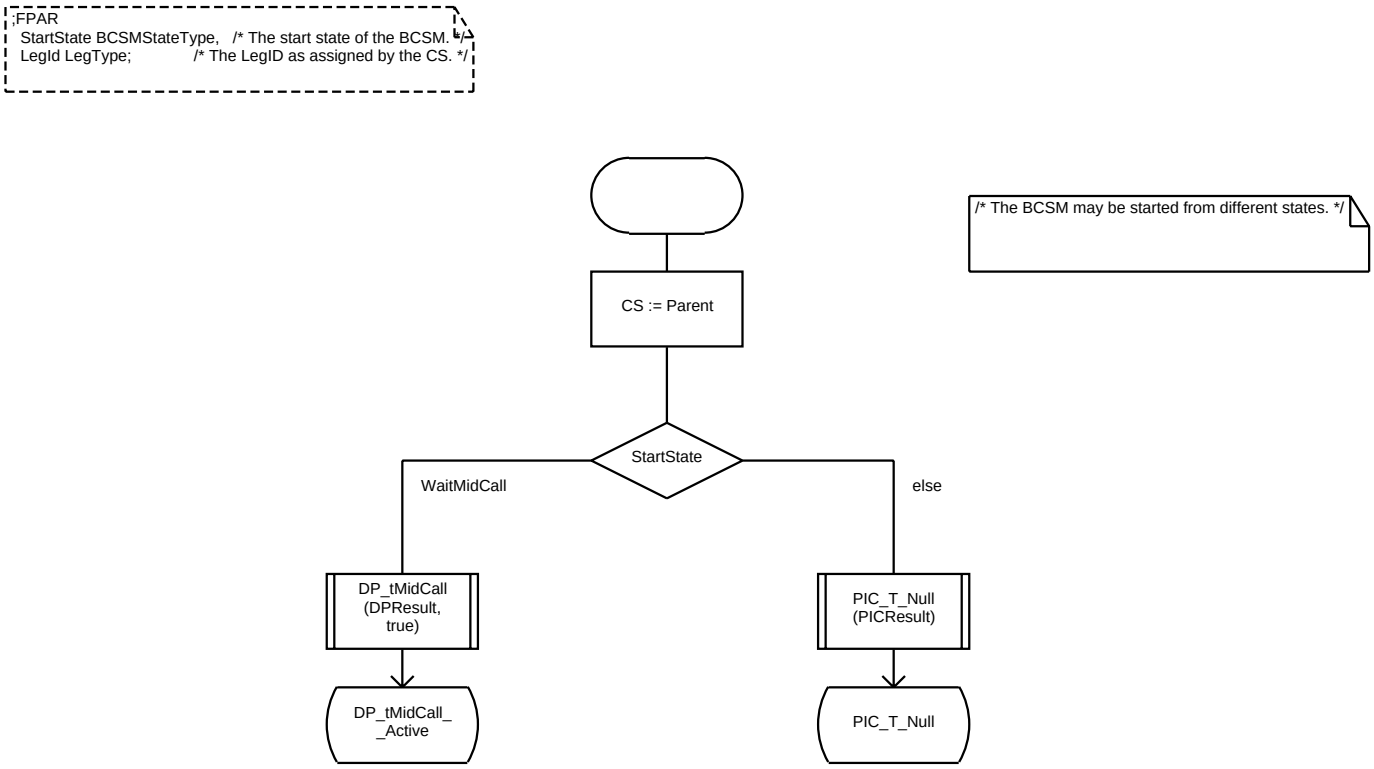
virtual
DP_tDisconnect

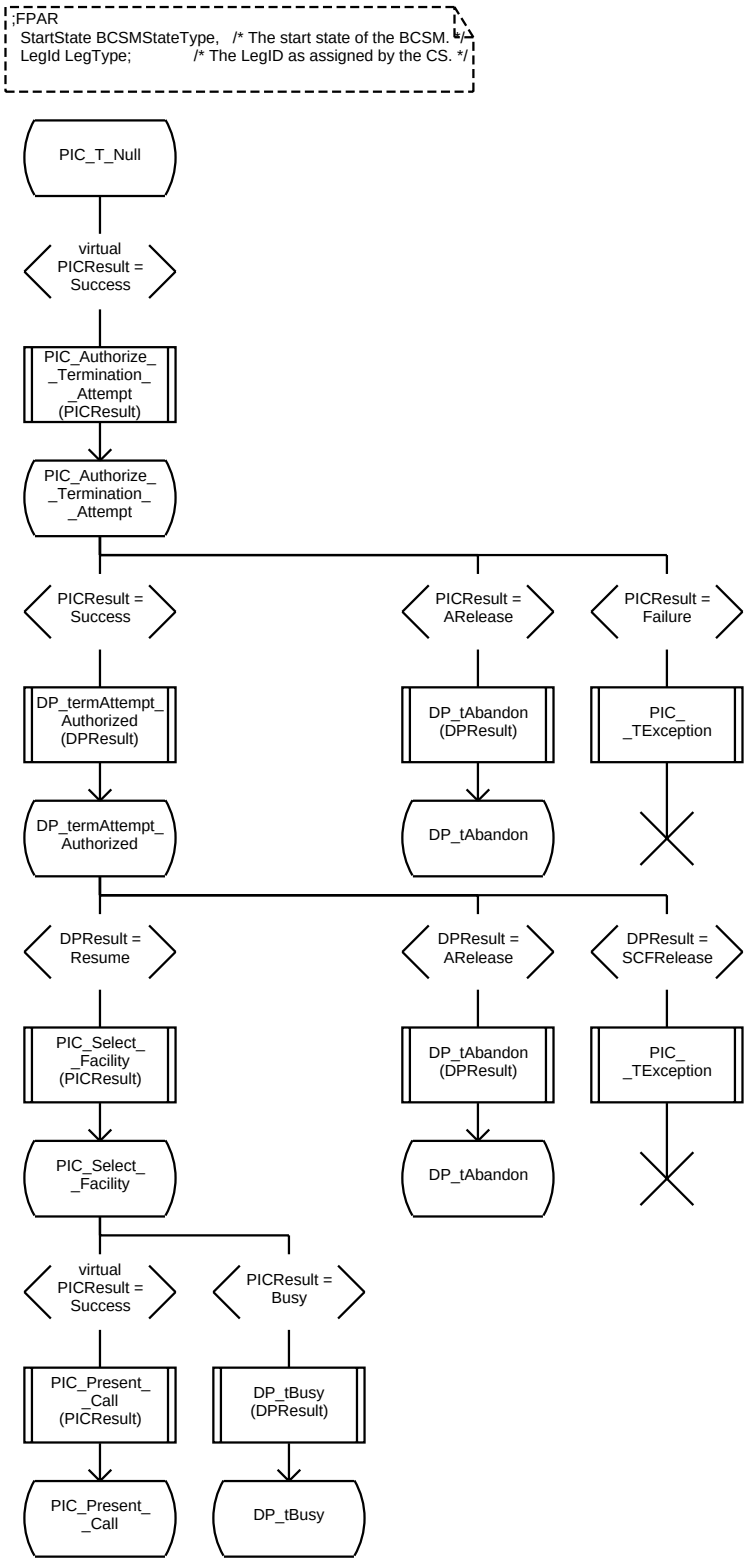
virtual
DP_tBusy

virtual
DP_tMidCall

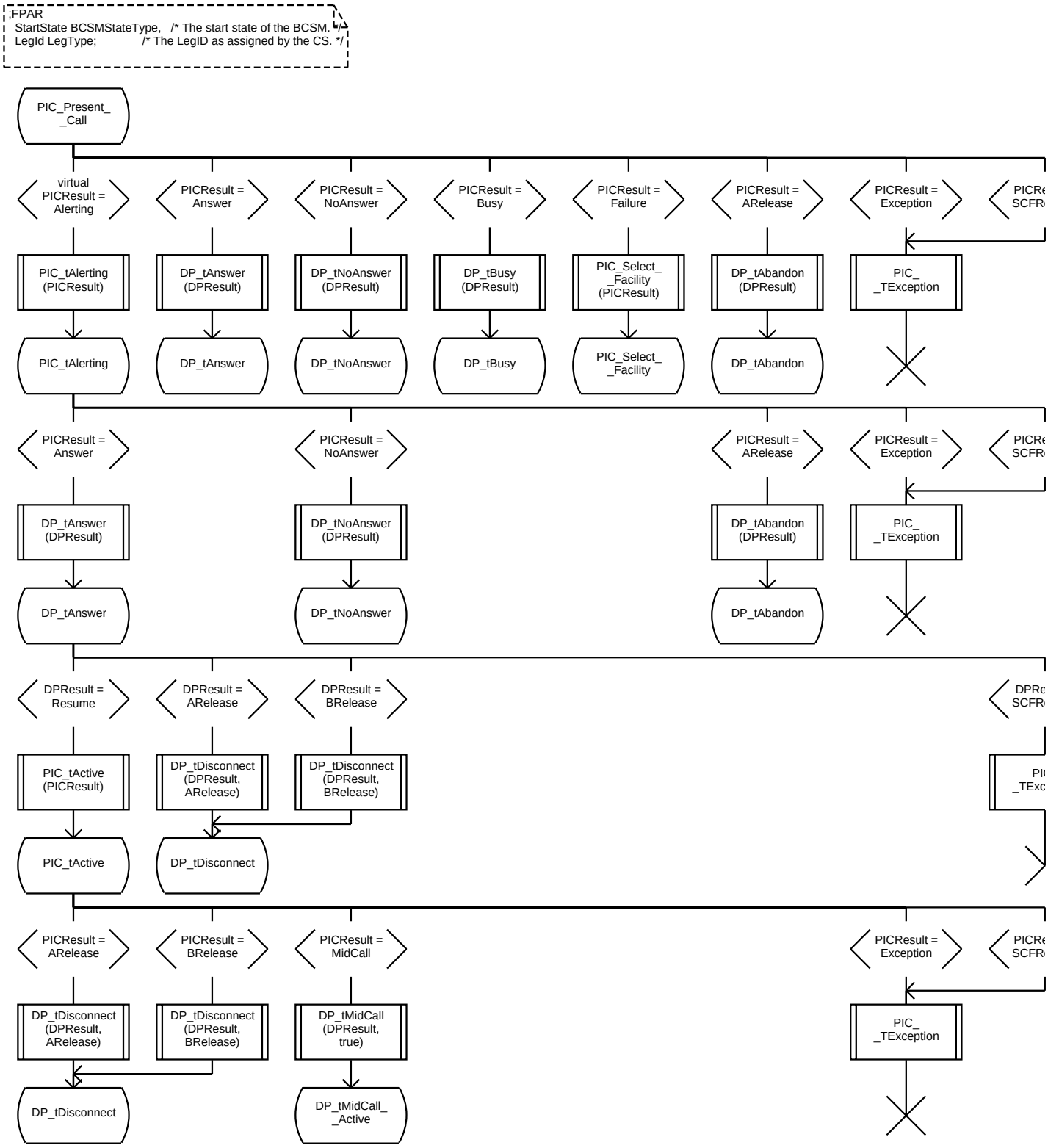
virtual
DP_tAnswer

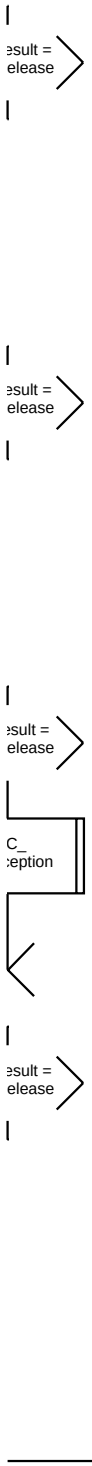
virtual
DP_tAbandon


;FPAR
StartState BCSMStateType, /* The start state of the BCSM. */
LegId LegType; /* The LegID as assigned by the CS. */
/* The BCSM may be started from different states. */
CS := Parent
StartState
WaitMidCall
else
DP_tMidCall (DPResult, true)
PIC_T_Null (PICResult)
DP_tMidCall_ _Active
PIC_T_Null

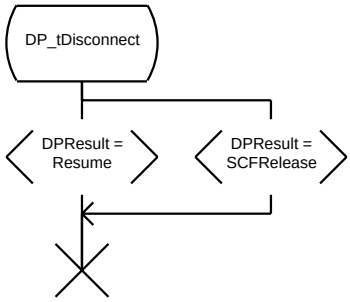

;FPAR
StartState BCSMStateType, /* The start state of the BCSM. */
LegId LegType; /* The LegID as assigned by the CS. */
PIC_T_Null
virtual PICResult = Success
PIC_Authorize_ _Termination_ _Attempt (PICResult)
PIC_Authorize_ _Termination_ _Attempt
PICResult = Success
PICResult = ARelease
PICResult = Failure
DP_termAttempt_ Authorized (DPResult)
DP_tAbandon (DPResult)
PIC_ _TException
DP_termAttempt_ Authorized
DP_tAbandon
DPResult = Resume
DPResult = ARelease
DPResult = SCFRelease
PIC_Select_ _Facility (PICResult)
DP_tAbandon (DPResult)
PIC_ _TException
PIC_Select_ _Facility
DP_tAbandon
virtual PICResult = Success
PICResult = Busy
PIC_Present_ _Call (PICResult)
DP_tBusy (DPResult)
PIC_Present_ _Call
DP_tBusy

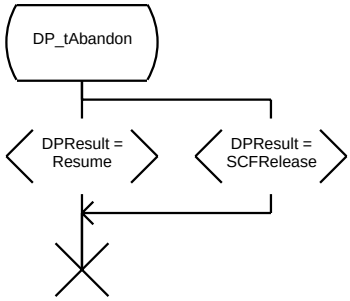
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCSM

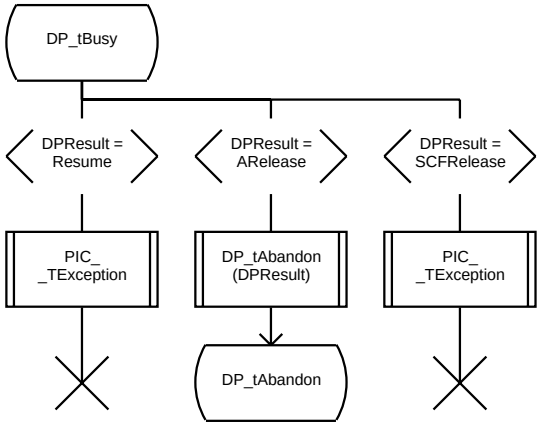


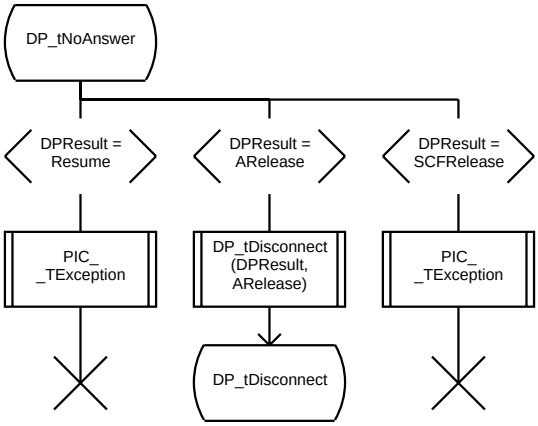

esult =
elease
esult =
elease
esult =
elease
C_
eption
esult =
elease

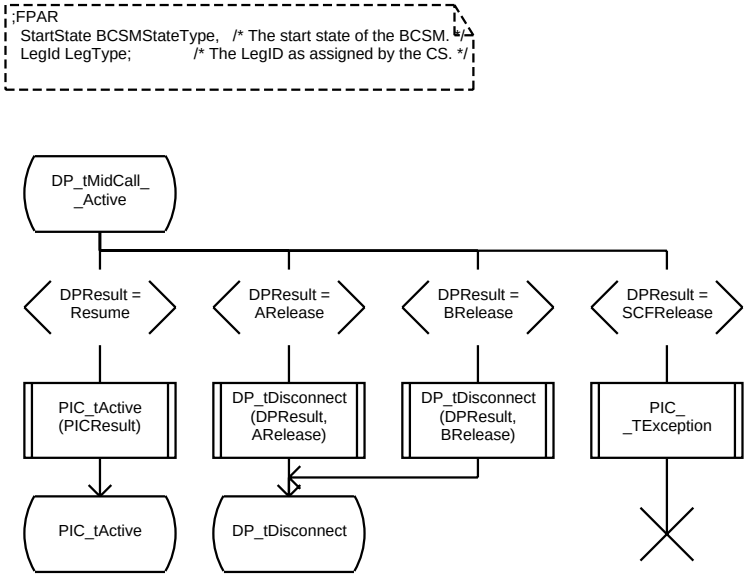
```
;FPAR
 StartState BCSMStateType,   /* The start state of the BCSM. */
 LegId LegType;              /* The LegID as assigned by the CS. */
```



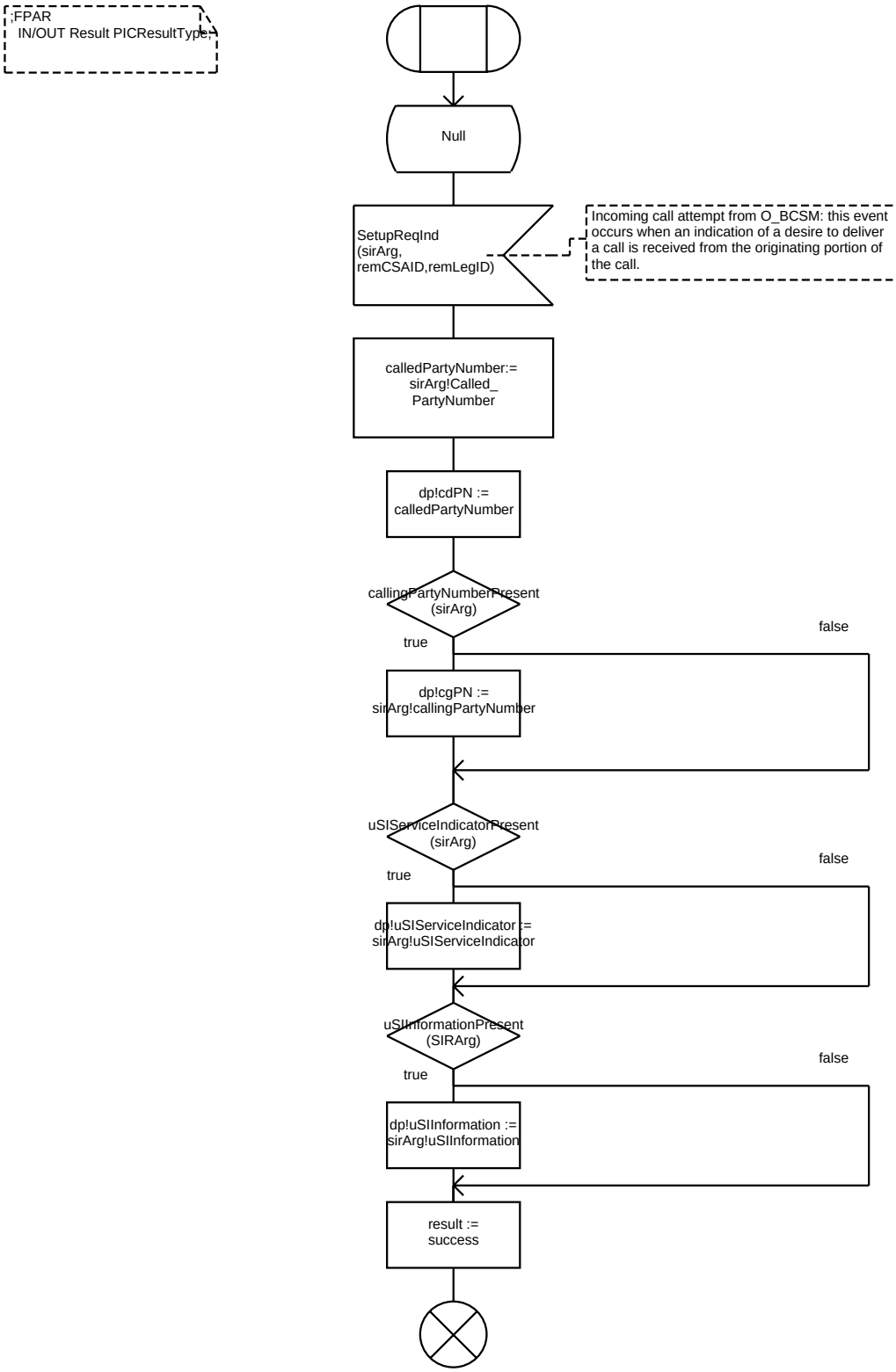


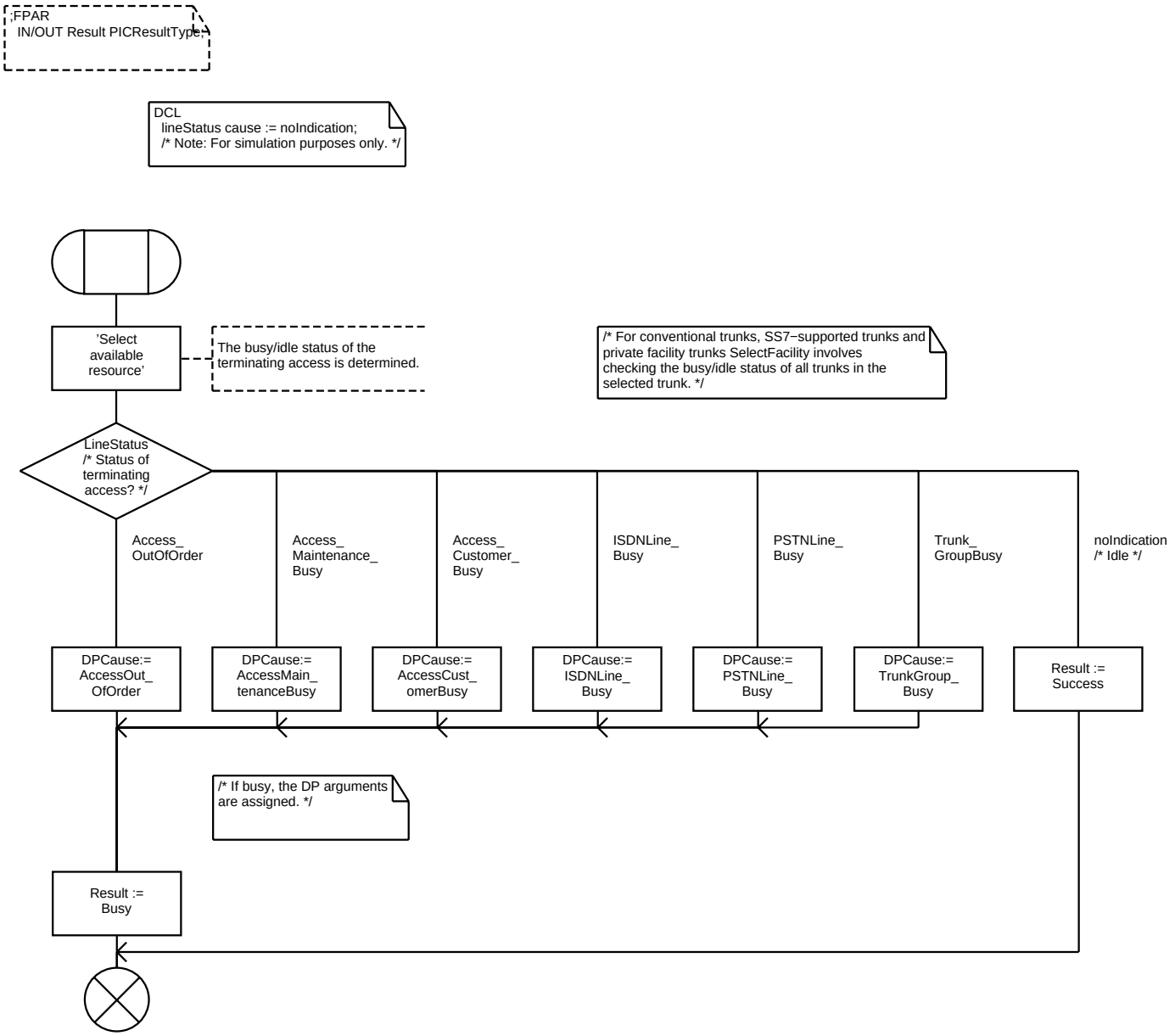




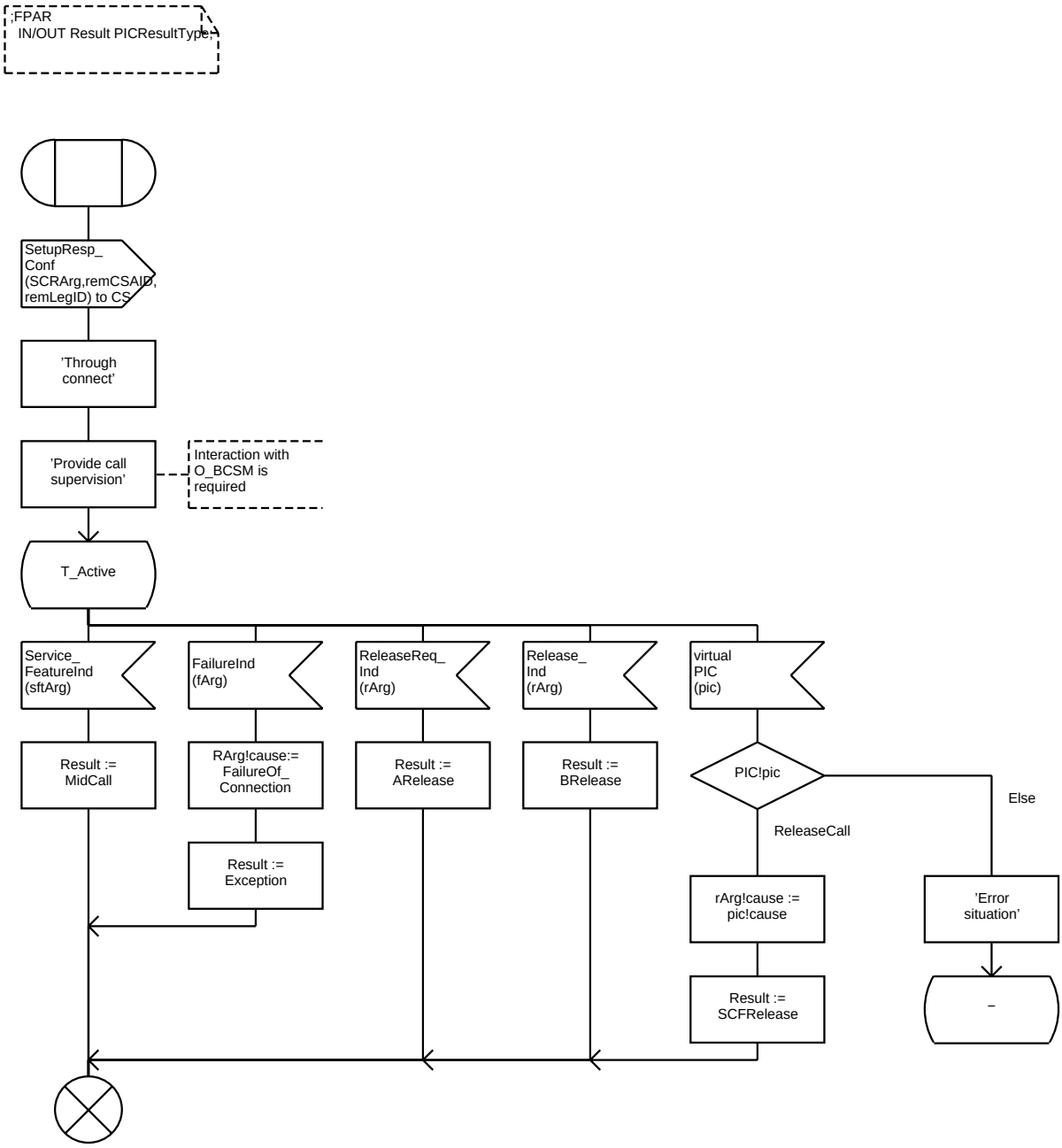

;FPAR
StartState BCSMStateType, /* The start state of the BCSM. */
LegId LegType; /* The LegID as assigned by the CS. */
DP_tMidCall_
_Active
DPResult =
Resume
DPResult =
ARelease
DPResult =
BRelease
DPResult =
SCFRelease
PIC_tActive
(PICResult)
DP_tDisconnect
(DPResult,
ARelease)
DP_tDisconnect
(DPResult,
BRelease)
PIC_
_TException
PIC_tActive
DP_tDisconnect

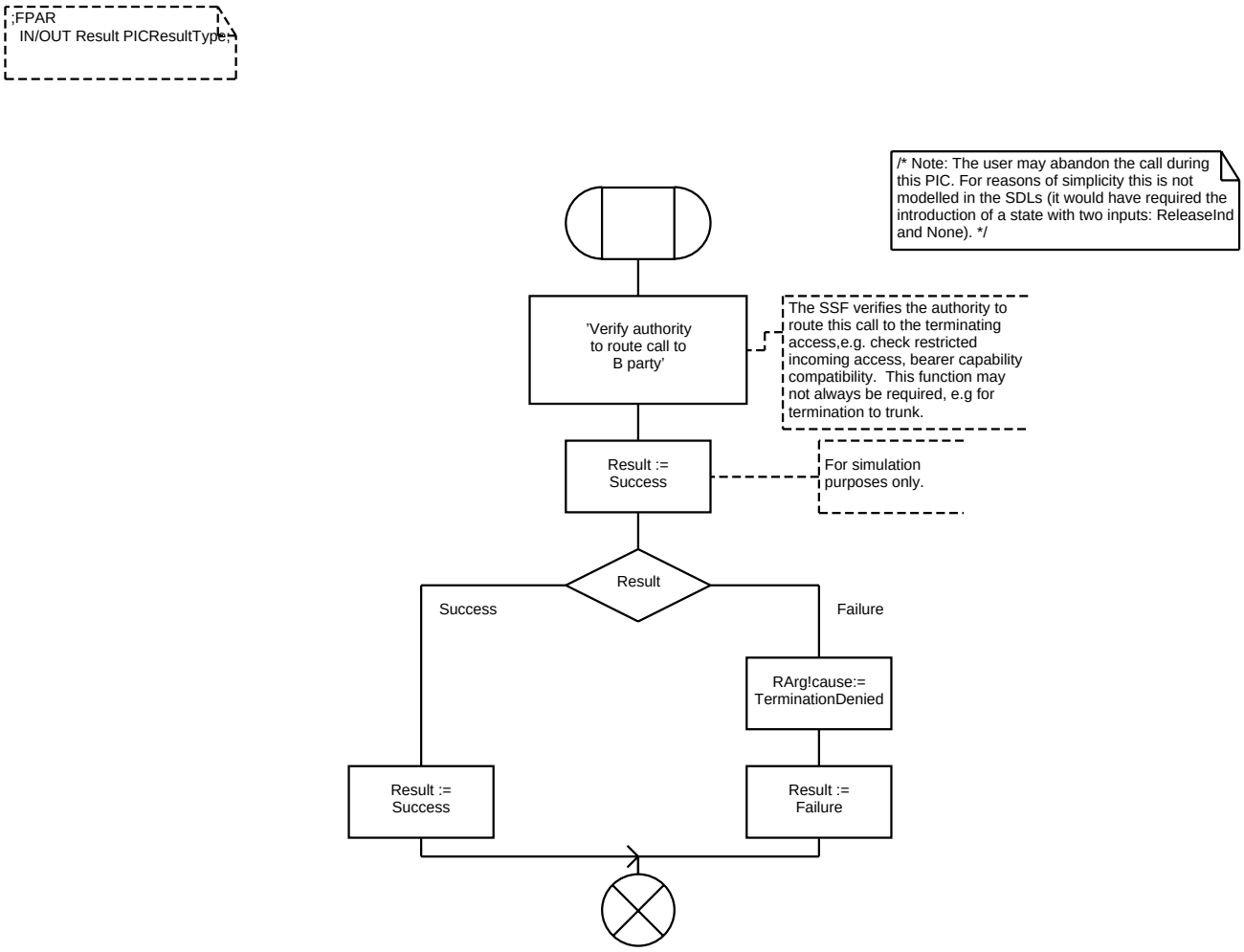
Procedure PIC_T_Null

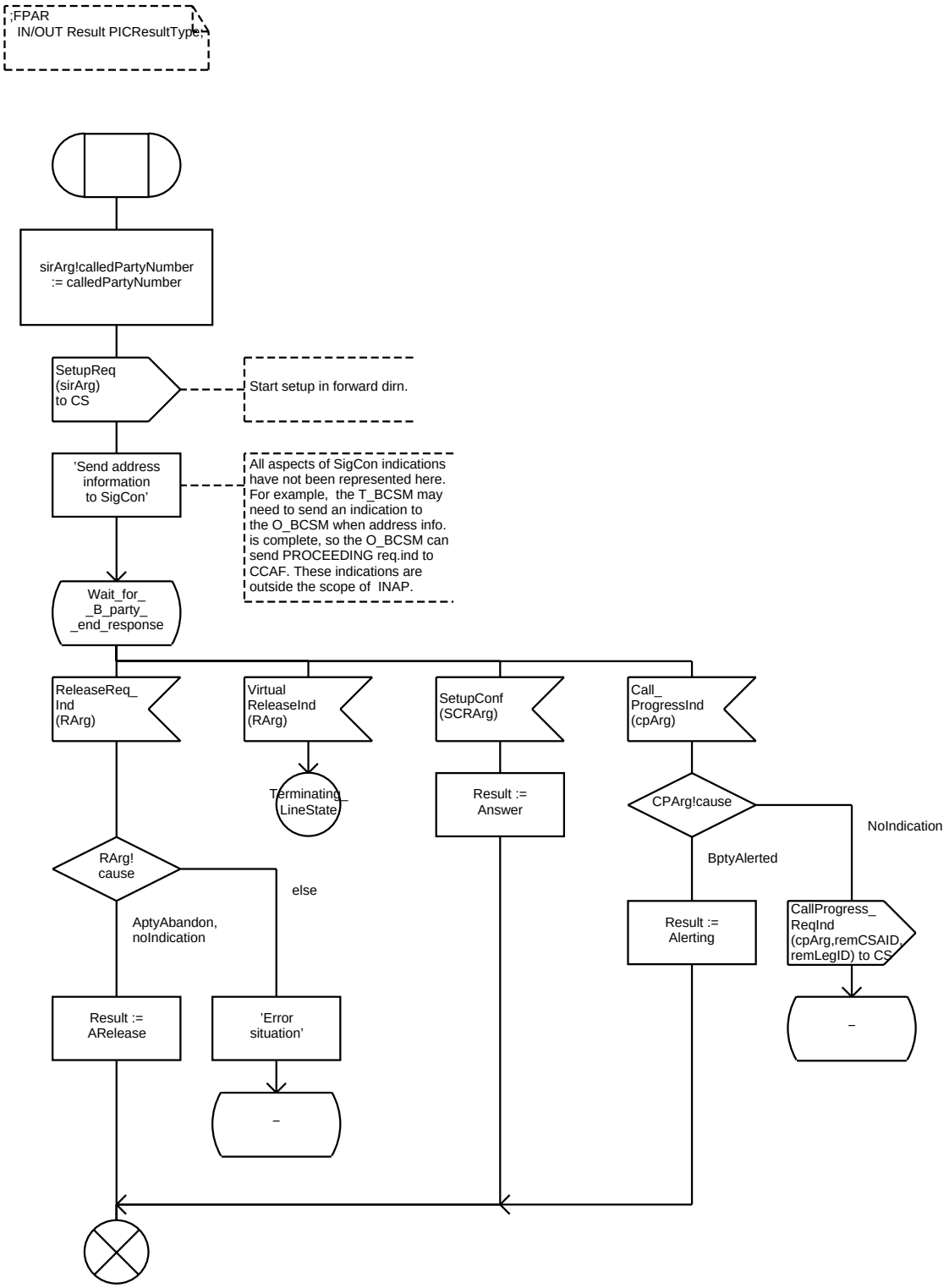


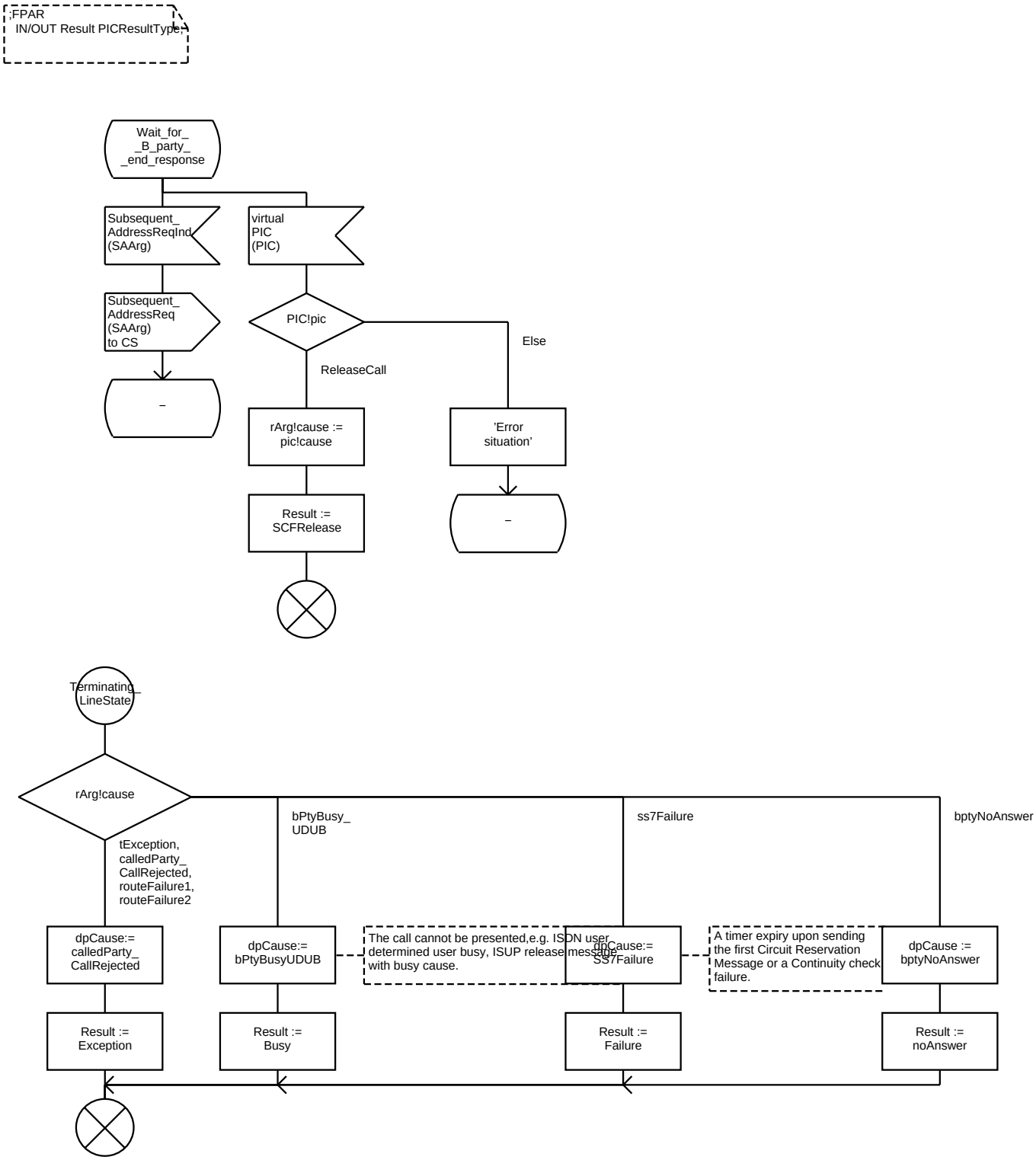

;FPAR
IN/OUT Result PICResultType;
DCL
lineStatus cause := noIndication;
/* Note: For simulation purposes only. */
'Select available resource'
The busy/idle status of the terminating access is determined.
/* For conventional trunks, SS7−supported trunks and private facility trunks SelectFacility involves checking the busy/idle status of all trunks in the selected trunk. */
LineStatus /* Status of terminating access? */
Access_ OutOfOrder
Access_ Maintenance_ Busy
Access_ Customer_ Busy
ISDNLine_ Busy
PSTNLine_ Busy
Trunk_ GroupBusy
noIndication /* Idle */
DPCause:= AccessOut_ OfOrder
DPCause:= AccessMain_ tenanceBusy
DPCause:= AccessCust_ omerBusy
DPCause:= ISDNLine_ Busy
DPCause:= PSTNLine_ Busy
DPCause:= TrunkGroup_ Busy
Result := Success
/* If busy, the DP arguments are assigned. */
Result := Busy

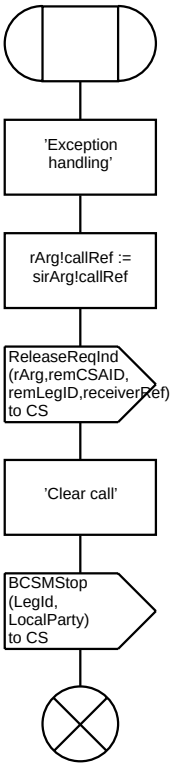
Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type TerminatingBCSM>> PIC_tActive 1(1)

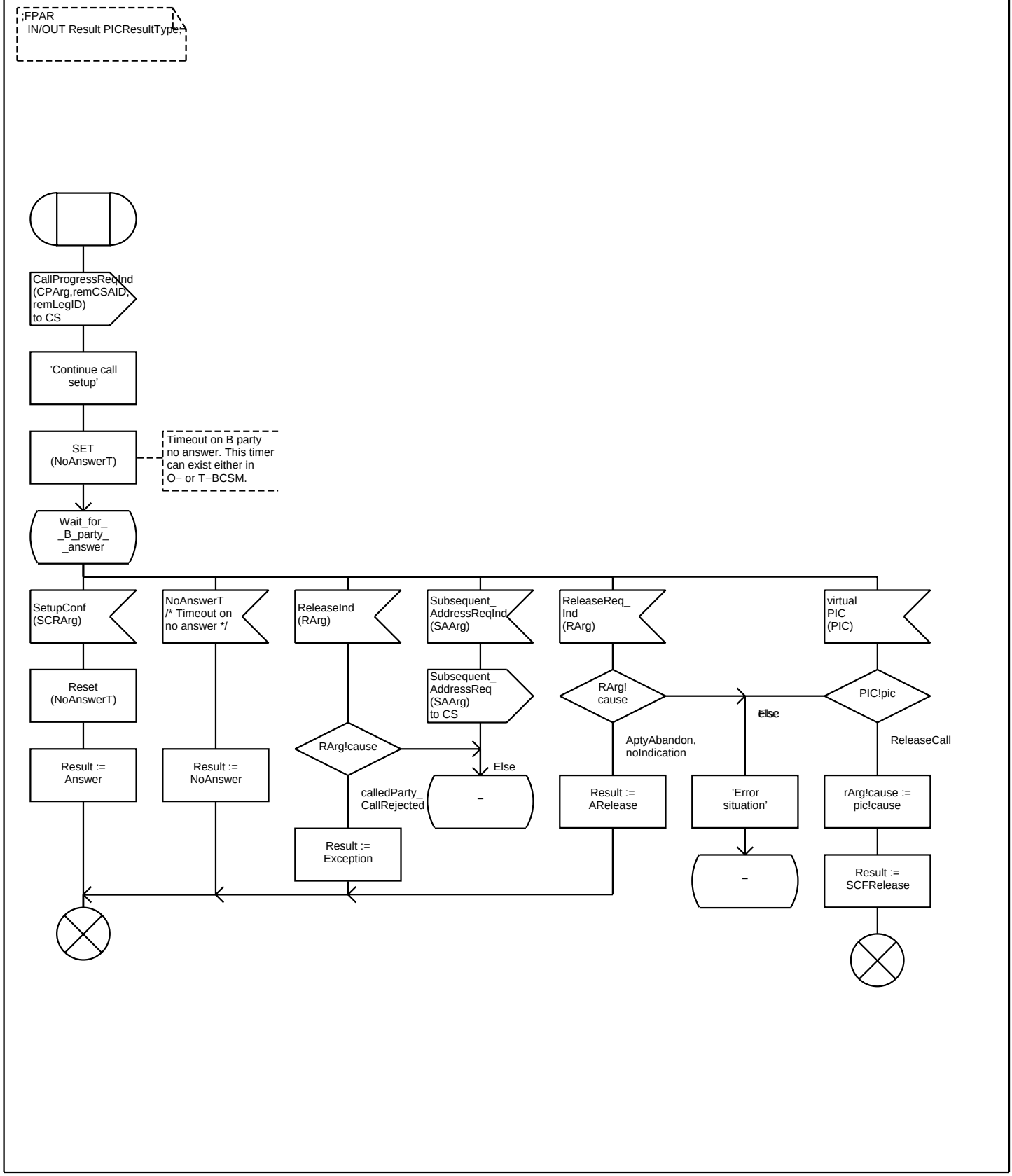


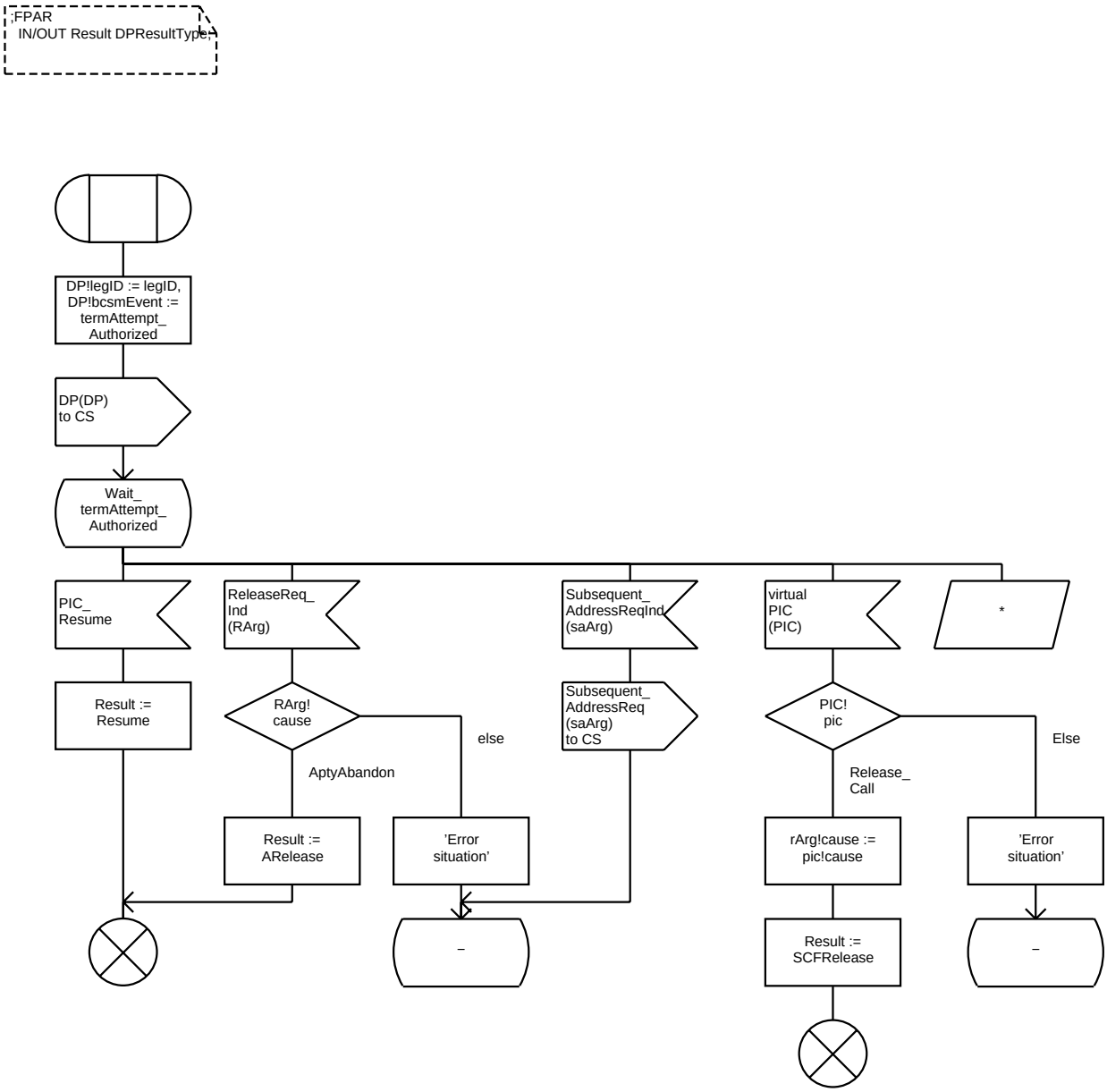

;FPAR
IN/OUT Result PICResultType;
/* Note: The user may abandon the call during this PIC. For reasons of simplicity this is not modelled in the SDLs (it would have required the introduction of a state with two inputs: ReleaseInd and None). */
'Verify authority to route call to B party'
The SSF verifies the authority to route this call to the terminating access,e.g. check restricted incoming access, bearer capability compatibility. This function may not always be required, e.g for termination to trunk.
Result := Success
For simulation purposes only.
Result
Success
Failure
RArg!cause:= TerminationDenied
Result := Success
Result := Failure

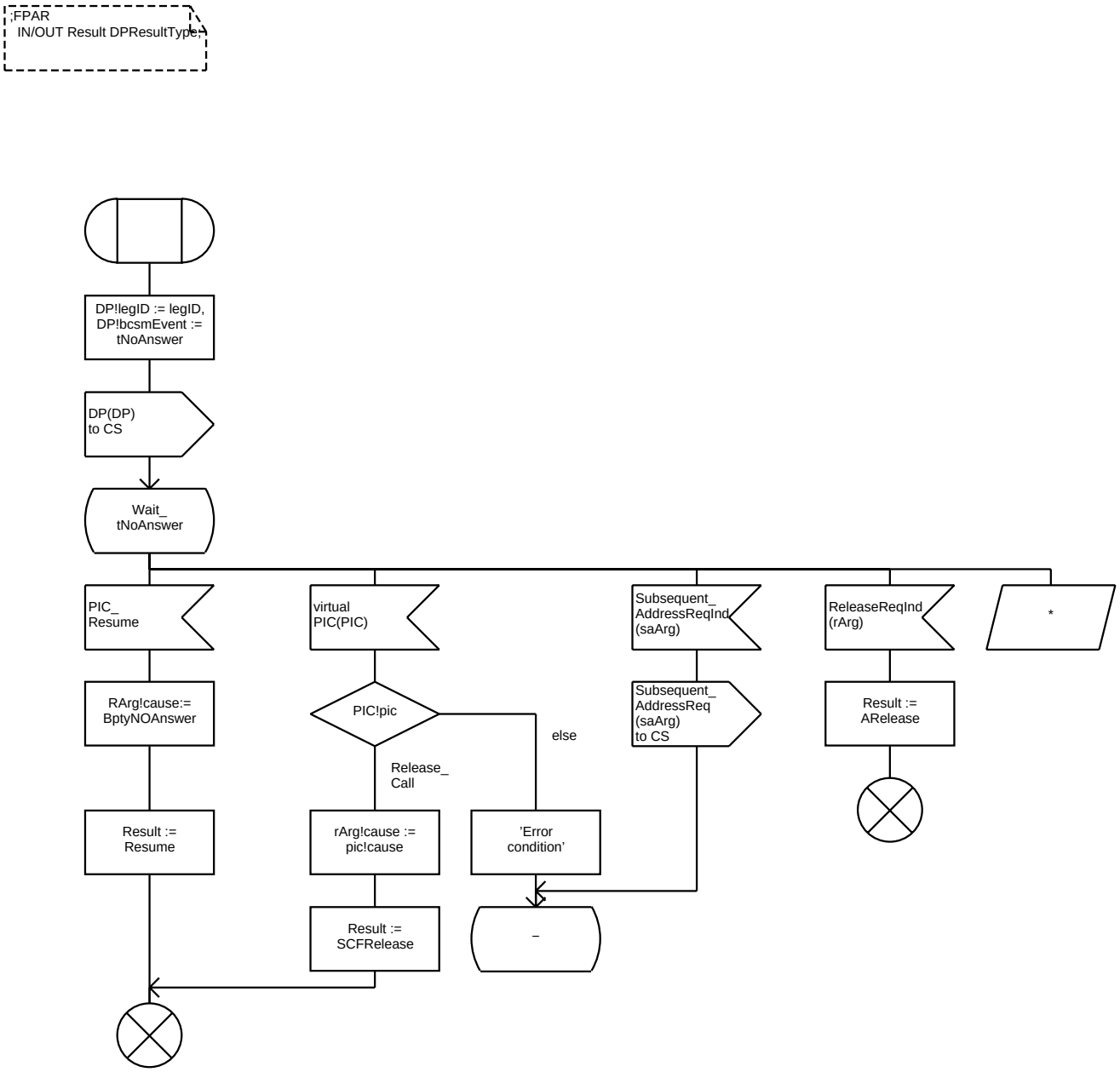

;FPAR
IN/OUT Result PICResultType;
sirArg!calledPartyNumber := calledPartyNumber
SetupReq (sirArg) to CS
Start setup in forward dirn.
'Send address information to SigCon'
All aspects of SigCon indications have not been represented here. For example, the T_BCSM may need to send an indication to the O_BCSM when address info. is complete, so the O_BCSM can send PROCEEDING req.ind to CCAF. These indications are outside the scope of INAP.
Wait_for_ _B_party_ _end_response
ReleaseReq_ Ind (RArg)
Virtual ReleaseInd (RArg)
SetupConf (SCRArg)
Call_ ProgressInd (cpArg)
Terminating_ LineState
Result := Answer
CPArg!cause
NoIndication
BptyAlerted
RArg! cause
else
AptyAbandon, noIndication
Result := Alerting
CallProgress_ ReqInd (cpArg,remCSAID, remLegID) to CS
Result := ARelease
'Error situation'
–
–

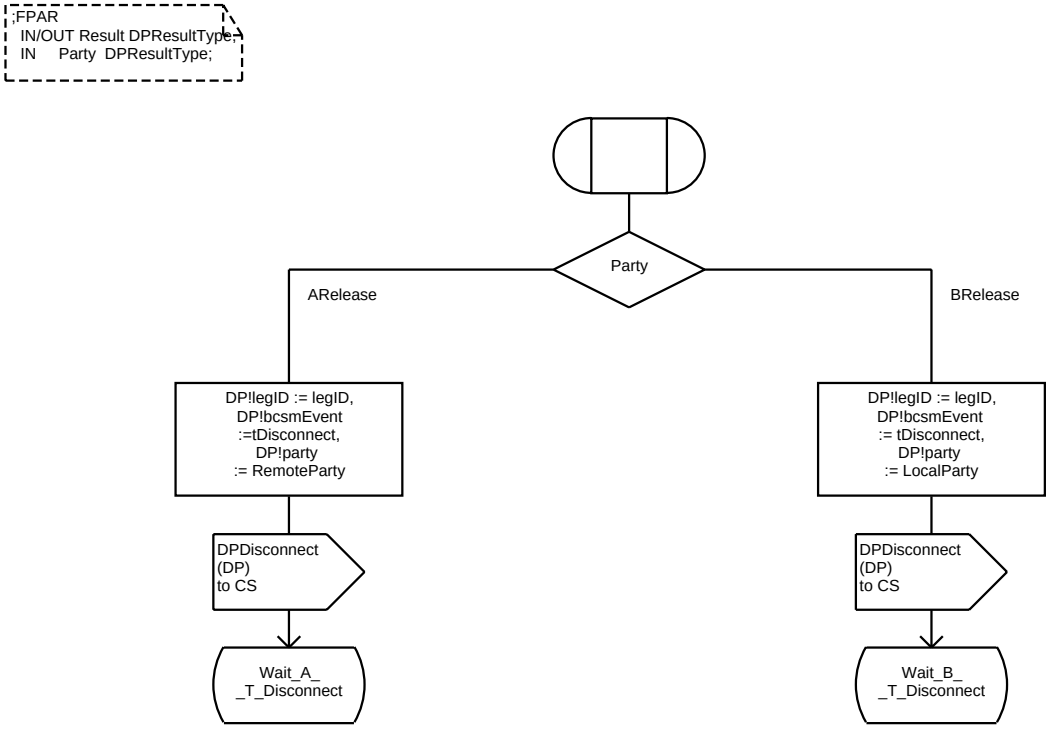

;FPAR
IN/OUT Result PICResultType;
Wait_for_
_B_party_
_end_response
Subsequent_
AddressReqInd
(SAArg)
virtual
PIC
(PIC)
Subsequent_
AddressReq
(SAArg)
to CS
PIC!pic
Else
ReleaseCall
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
Terminating_
LineState
rArg!cause
bPtyBusy_
UDUB
ss7Failure
bptyNoAnswer
tException,
calledPartyBusy,
CallRejected,
routeFailure1,
routeFailure2
dpCause:=
calledPartyBusy_
CallRejected
dpCause:=
bPtyBusyUDUB
The call cannot be presented,e.g. ISDN user determined user busy, ISUP release message with busy cause.
dpCause:=
SS7Failure
A timer expiry upon sending the first Circuit Reservation Message or a Continuity check failure.
dpCause :=
bptyNoAnswer
Result :=
Exception
Result :=
Busy
Result :=
Failure
Result :=
noAnswer

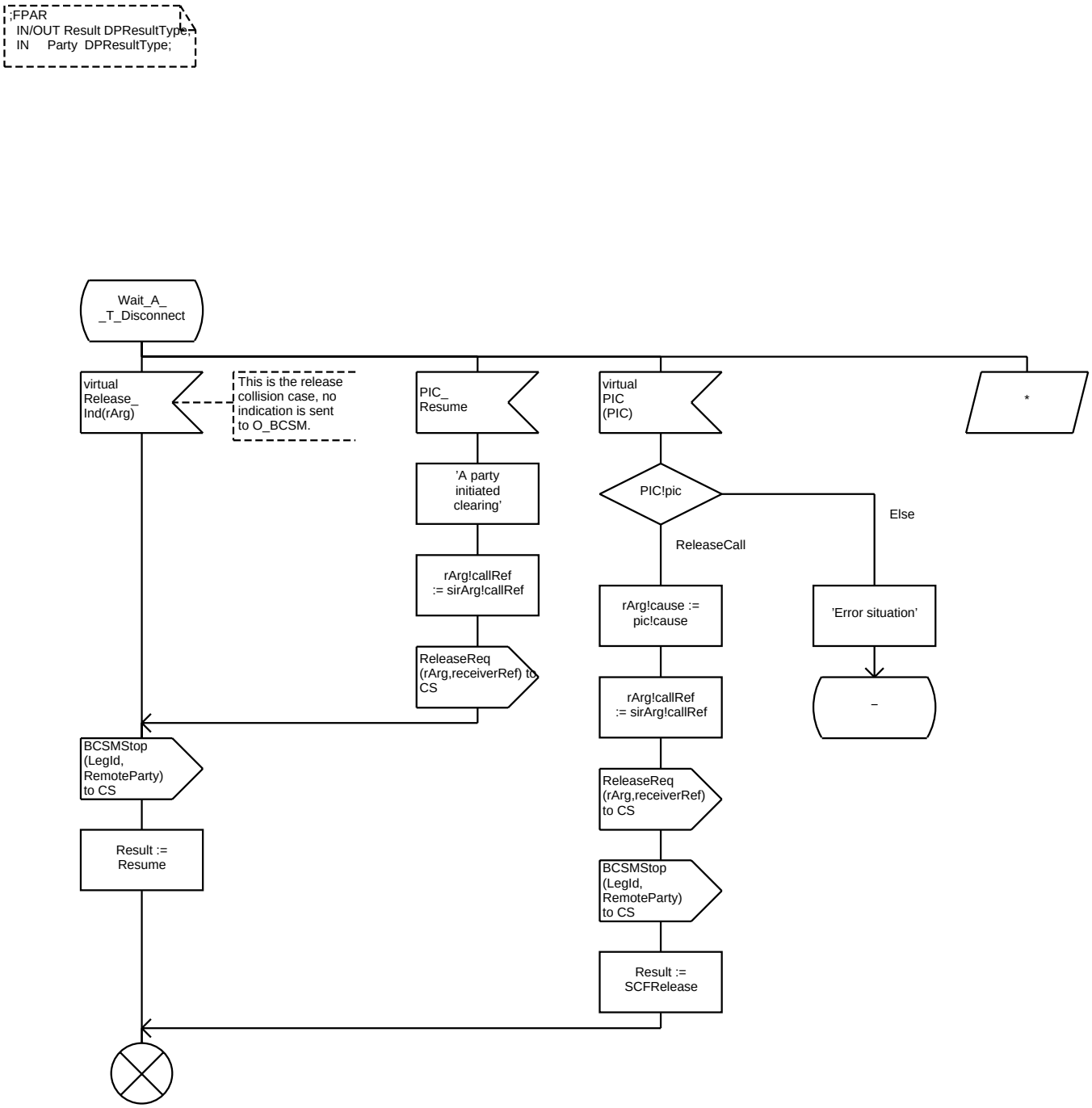

'Exception handling'
rArg!callRef := sirArg!callRef
ReleaseReqInd (rArg,remCSAID, remLegID,receiverRef) to CS
'Clear call'
BCSMStop (LegId, LocalParty) to CS

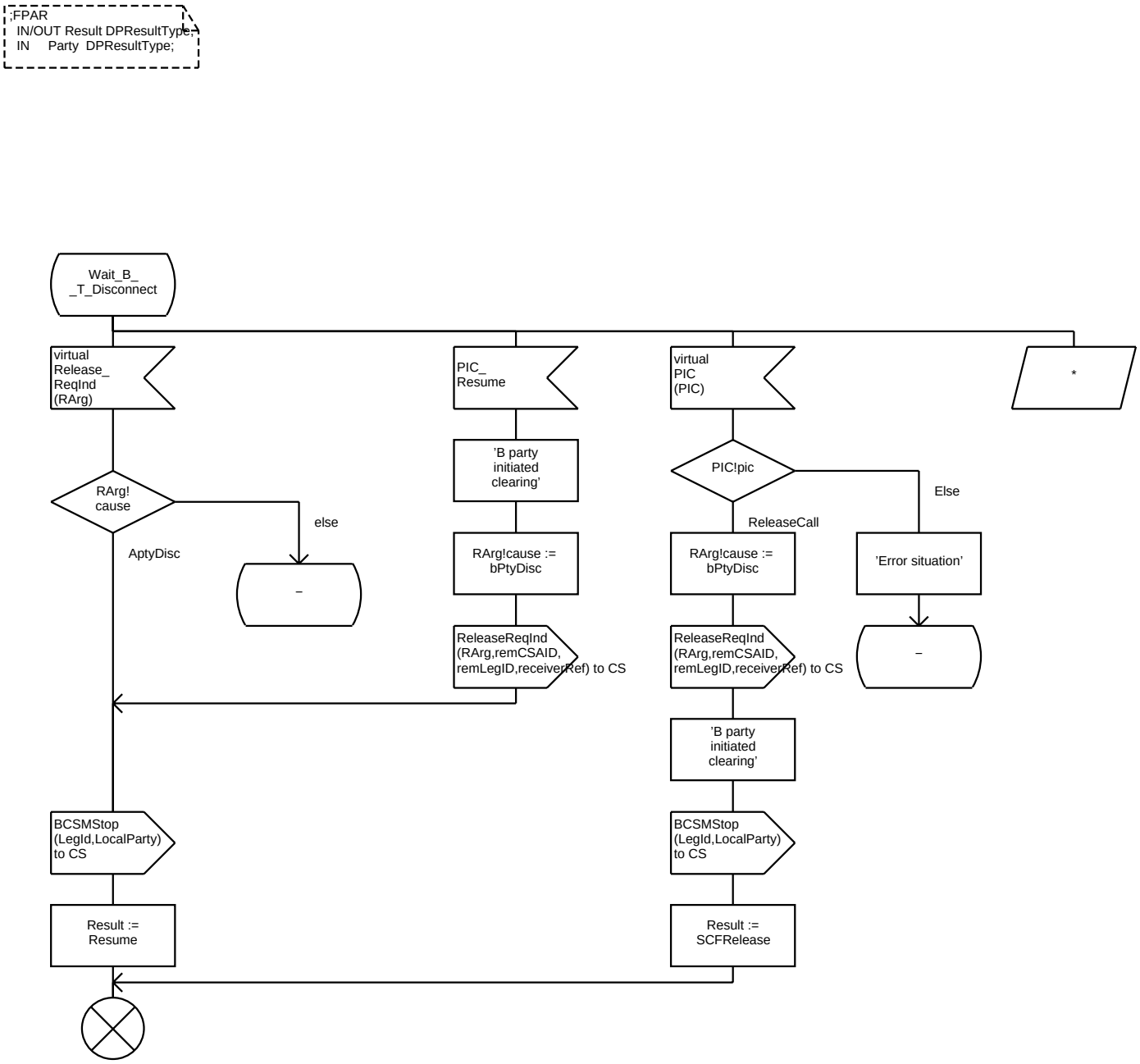

;FPAR
IN/OUT Result PICResultType;
CallProgressReqInd
(CPArg,remCSAID,
remLegID)
to CS
'Continue call
setup'
SET
(NoAnswerT)
Timeout on B party
no answer. This timer
can exist either in
O− or T−BCSM.
Wait_for_
_B_party_
_answer
SetupConf
(SCRArg)
NoAnswerT
/* Timeout on
no answer */
ReleaseInd
(RArg)
Subsequent_
AddressReqInd
(SAArg)
ReleaseReq_
Ind
(RArg)
virtual
PIC
(PIC)
Reset
(NoAnswerT)
Subsequent_
AddressReq
(SAArg)
to CS
RArg!
cause
PIC!pic
Else
Else
RArg!cause
AptyAbandon,
noIndication
ReleaseCall
Result :=
Answer
Result :=
NoAnswer
Else
calledParty_
CallRejected
–
Result :=
ARelease
'Error
situation'
rArg!cause :=
pic!cause
Result :=
Exception
–
Result :=
SCFRelease


;FPAR
IN/OUT Result DPResultType;
DP!legID := legID,
DP!bcsmEvent :=
termAttempt_
Authorized
DP(DP)
to CS
Wait_
termAttempt_
Authorized
PIC_
Resume
ReleaseReq_
Ind
(RArg)
Subsequent_
AddressReqInd
(saArg)
virtual
PIC
(PIC)
*
Result :=
Resume
RArg!
cause
else
AptyAbandon
Subsequent_
AddressReq
(saArg)
to CS
PIC!
pic
Else
Release_
Call
Result :=
ARelease
'Error
situation'
rArg!cause :=
pic!cause
'Error
situation'
–
Result :=
SCFRelease
–

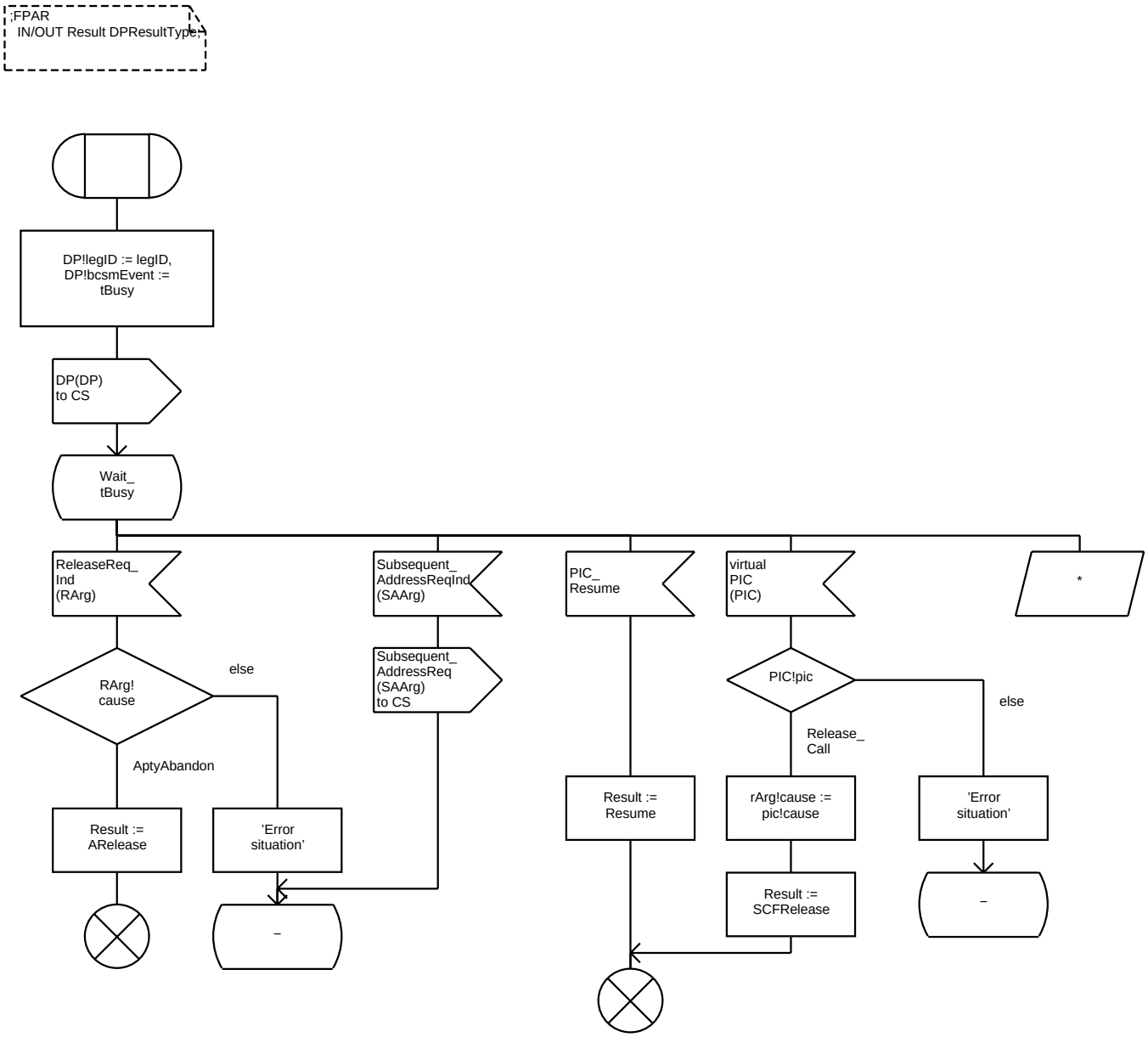

;FPAR
IN/OUT Result DPResultType;
DP!legID := legID,
DP!bcsmEvent :=
tNoAnswer
DP(DP)
to CS
Wait_
tNoAnswer
PIC_
Resume
virtual
PIC(PIC)
Subsequent_
AddressReqInd
(saArg)
ReleaseReqInd
(rArg)
*
RArg!cause:=
BptyNOAnswer
PIC!pic
else
Release_
Call
Subsequent_
AddressReq
(saArg)
to CS
Result :=
ARelease
Result :=
Resume
rArg!cause :=
pic!cause
'Error
condition'
Result :=
SCFRelease
_

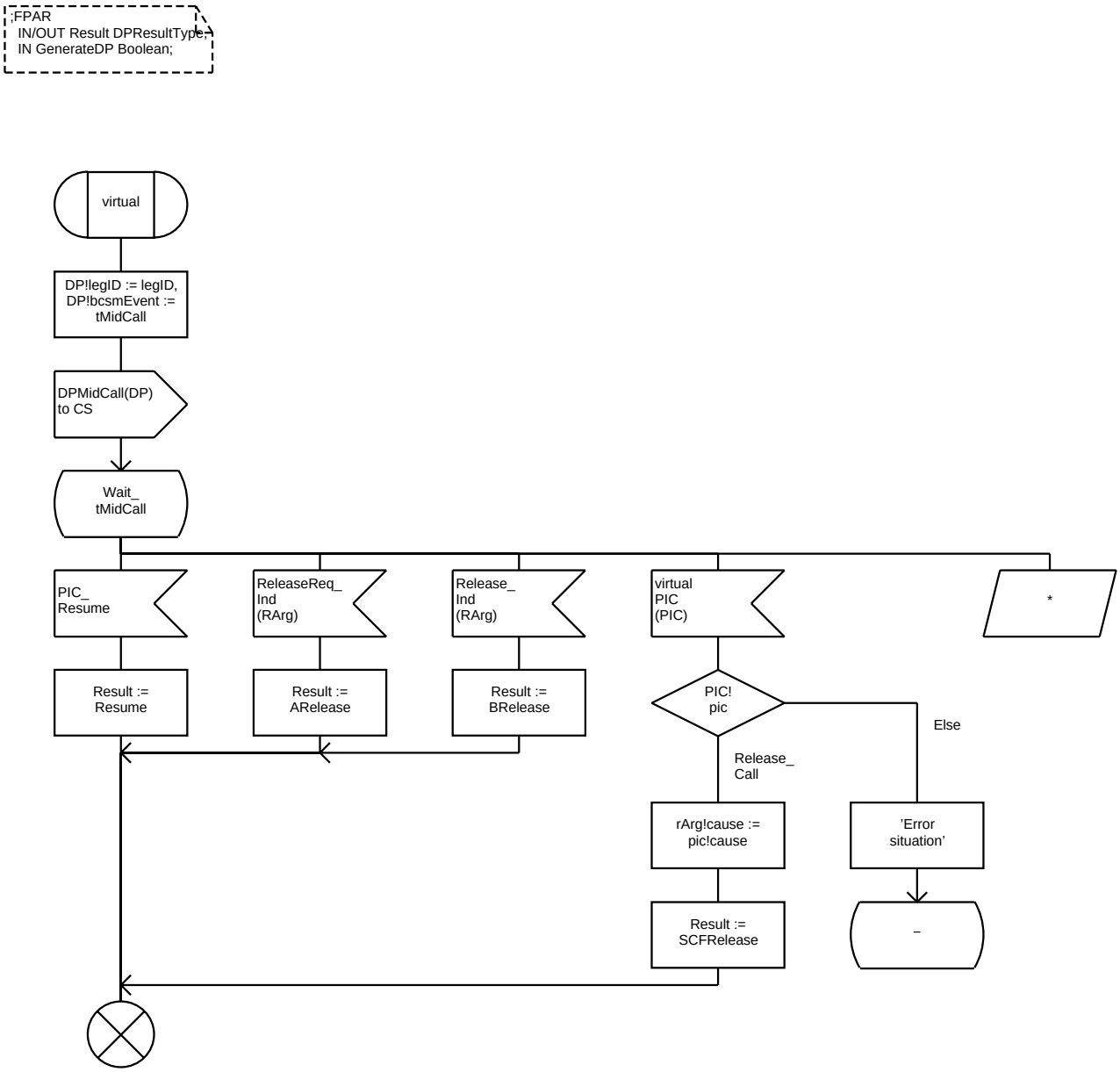

;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Party
ARelease
BRelease
DP!legID := legID,
DP!bcsmEvent
:=tDisconnect,
DP!party
:= RemoteParty
DP!legID := legID,
DP!bcsmEvent
:= tDisconnect,
DP!party
:= LocalParty
DPDisconnect
(DP)
to CS
DPDisconnect
(DP)
to CS
Wait_A_
_T_Disconnect
Wait_B_
_T_Disconnect

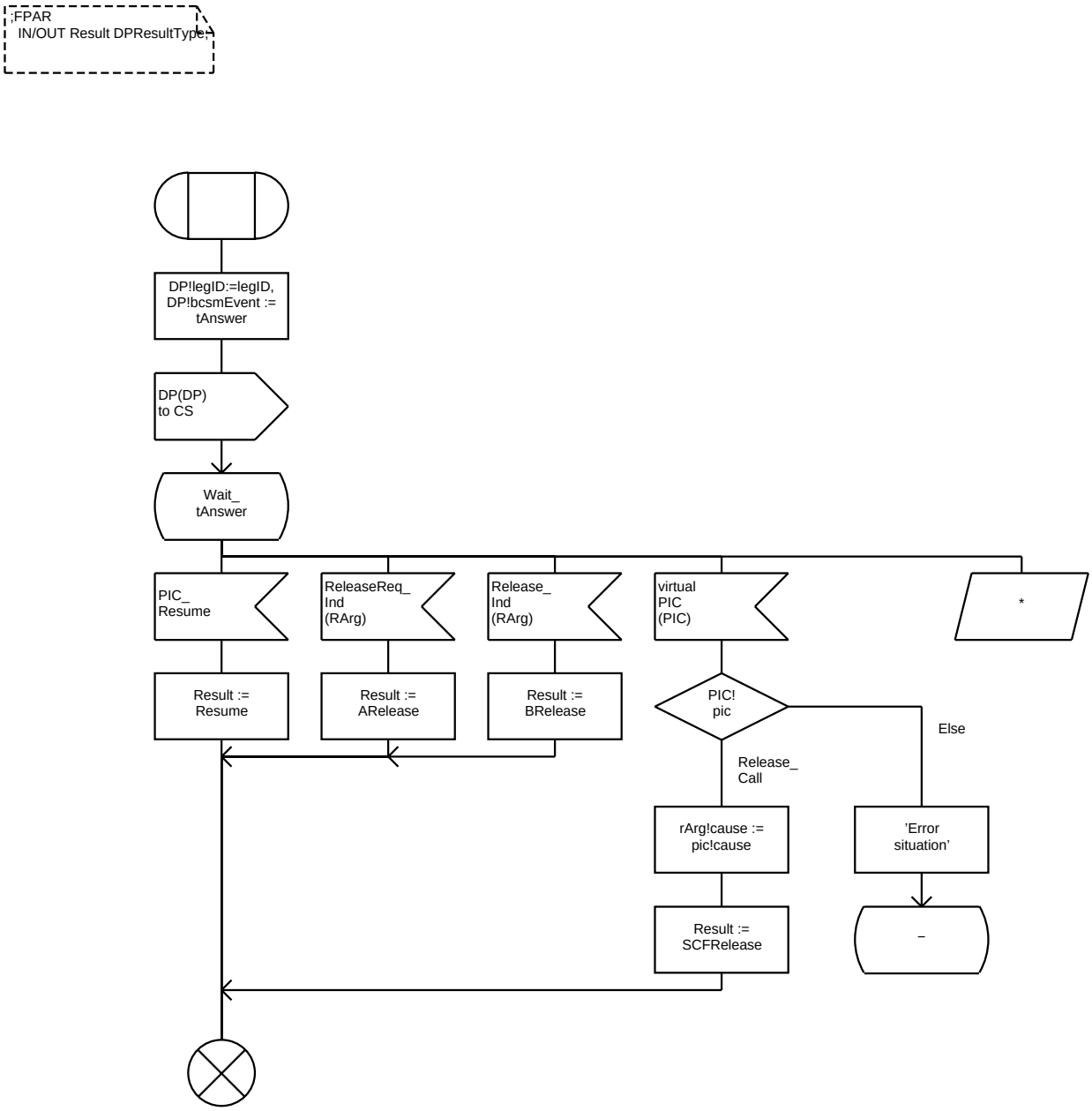

;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Wait_A_
_T_Disconnect
virtual
Release_
Ind(rArg)
This is the release
collision case, no
indication is sent
to O_BCSM.
PIC_
Resume
virtual
PIC
(PIC)
*
'A party
initiated
clearing'
PIC!pic
Else
ReleaseCall
rArg!callRef
:= sirArg!callRef
rArg!cause :=
pic!cause
'Error situation'
ReleaseReq
(rArg,receiverRef) to
CS
rArg!callRef
:= sirArg!callRef
–
BCSMStop
(LegId,
RemoteParty)
to CS
ReleaseReq
(rArg,receiverRef)
to CS
Result :=
Resume
BCSMStop
(LegId,
RemoteParty)
to CS
Result :=
SCFRelease

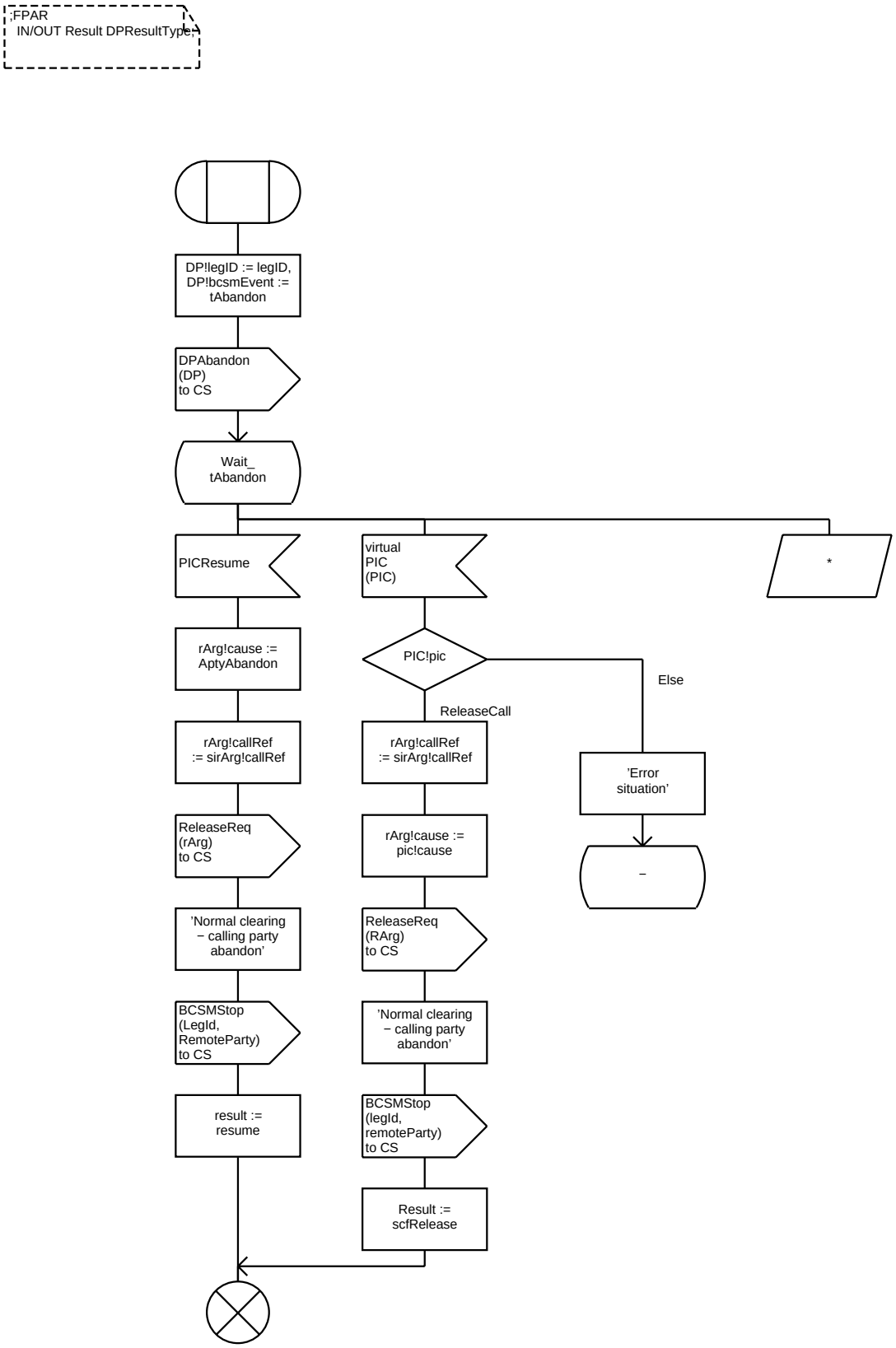

;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Wait_B_
_T_Disconnect
virtual
Release_
ReqInd
(RArg)
PIC_
Resume
virtual
PIC
(PIC)
*
'B party
initiated
clearing'
PIC!pic
Else
RArg!
cause
else
ReleaseCall
AptyDisc
RArg!cause :=
bPtyDisc
RArg!cause :=
bPtyDisc
'Error situation'
–
ReleaseReqInd
(RArg,remCSAID,
remLegID,receiverRef) to CS
ReleaseReqInd
(RArg,remCSAID,
remLegID,receiverRef) to CS
–
'B party
initiated
clearing'
BCSMStop
(LegId,LocalParty)
to CS
BCSMStop
(LegId,LocalParty)
to CS
Result :=
Resume
Result :=
SCFRelease

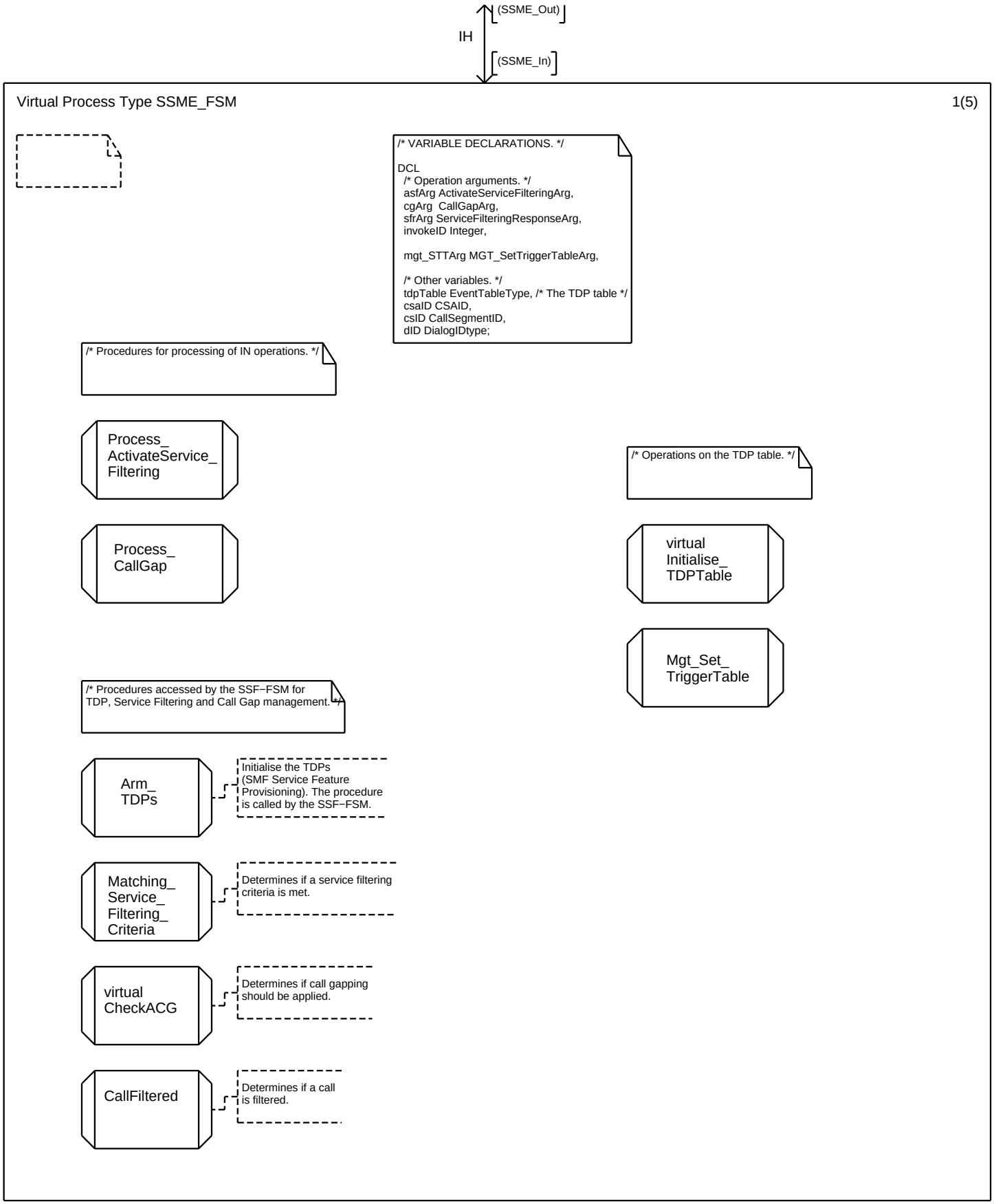
Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type TerminatingBCSM>> DP_tBusy 1(1)

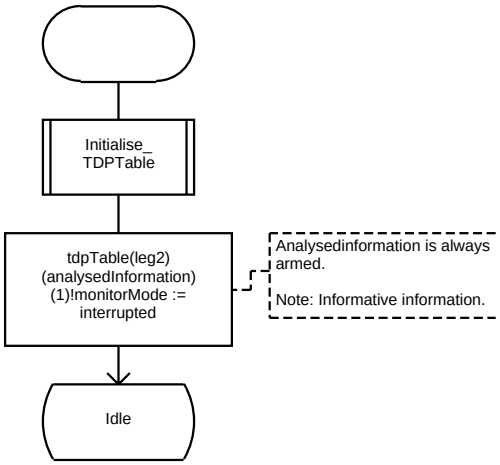


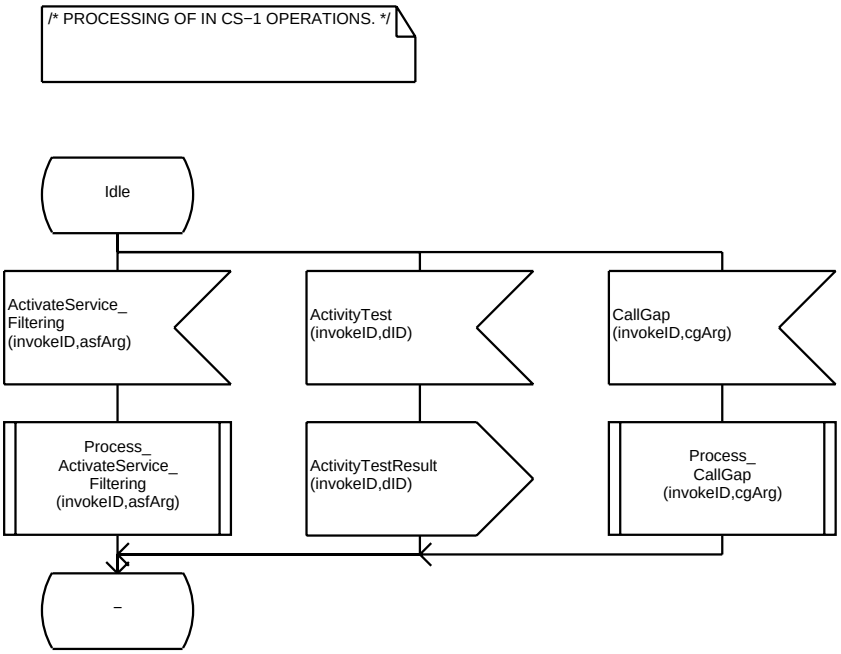

;FPAR
IN/OUT Result DPResultType,
IN GenerateDP Boolean;
virtual
DP!legID := legID,
DP!bcsmEvent :=
tMidCall
DPMidCall(DP)
to CS
Wait_
tMidCall
PIC_
Resume
ReleaseReq_
Ind
(RArg)
Release_
Ind
(RArg)
virtual
PIC
(PIC)
*
Result :=
Resume
Result :=
ARelease
Result :=
BRelease
PIC!
pic
Else
Release_
Call
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
_

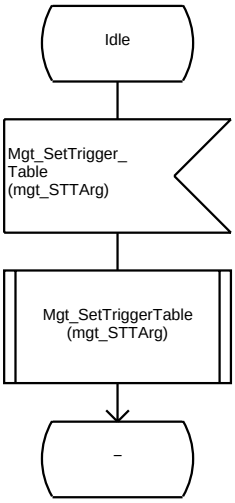

;FPAR
IN/OUT Result DPResultType;
DP!legID:=legID,
DP!bcsmEvent :=
tAnswer
DP(DP)
to CS
Wait_
tAnswer
PIC_
Resume
ReleaseReq_
Ind
(RArg)
Release_
Ind
(RArg)
virtual
PIC
(PIC)
*
Result :=
Resume
Result :=
ARelease
Result :=
BRelease
PIC!
pic
Else
Release_
Call
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
_

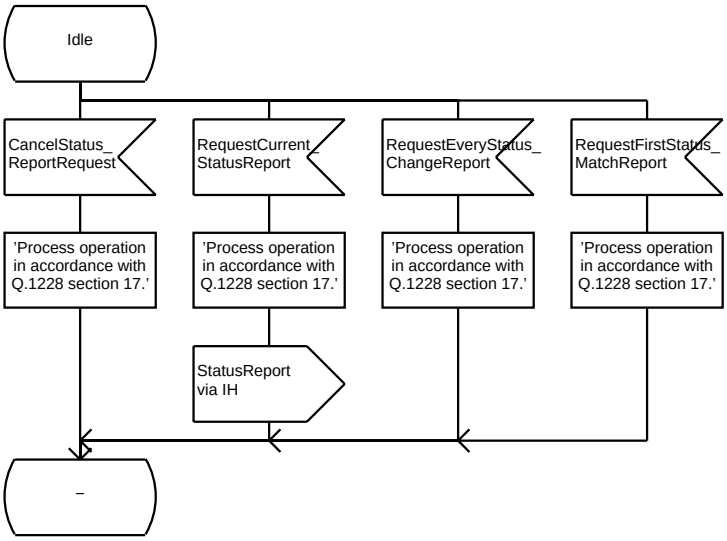

;FPAR
IN/OUT Result DPResultType;
DP!legID := legID, DP!bcsmEvent := tAbandon
DPAbandon (DP) to CS
Wait_ tAbandon
PICResume
virtual PIC (PIC)
*
rArg!cause := AptyAbandon
PIC!pic
Else
ReleaseCall
rArg!callRef := sirArg!callRef
rArg!callRef := sirArg!callRef
'Error situation'
ReleaseReq (rArg) to CS
rArg!cause := pic!cause
–
'Normal clearing – calling party abandon'
ReleaseReq (RArg) to CS
BCSMStop (LegId, RemoteParty) to CS
'Normal clearing – calling party abandon'
result := resume
BCSMStop (legId, remoteParty) to CS
Result := scfRelease

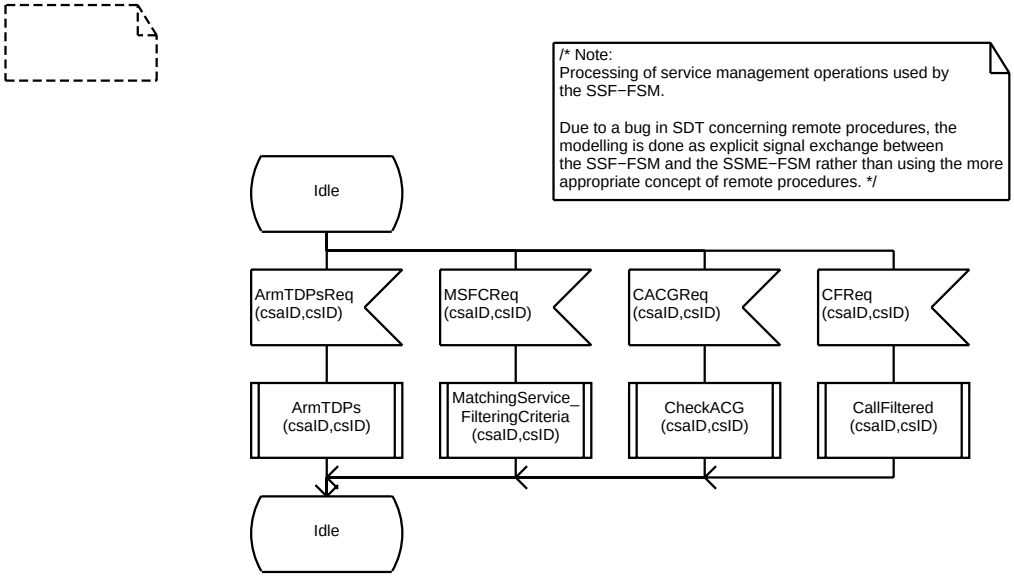

(SSME_Out)
IH
(SSME_In)
Virtual Process Type SSME_FSM
1(5)
/* VARIABLE DECLARATIONS. */
DCL
/* Operation arguments. */
asfArg ActivateServiceFilteringArg,
cgArg CallGapArg,
sfrArg ServiceFilteringResponseArg,
invokeID Integer,
mgt_STTArg MGT_SetTriggerTableArg,
/* Other variables. */
tdpTable EventTableType, /* The TDP table */
csaID CSAID,
csID CallSegmentID,
dID DialogIDtype;
/* Procedures for processing of IN operations. */
Process_ ActivateService_ Filtering
Process_ CallGap
/* Operations on the TDP table. */
virtual Initialise_ TDPTable
Mgt_Set_ TriggerTable
/* Procedures accessed by the SSF−FSM for TDP, Service Filtering and Call Gap management. */
Arm_ TDPs
Initialise the TDPs (SMF Service Feature Provisioning). The procedure is called by the SSF−FSM.
Matching_ Service_ Filtering_ Criteria
Determines if a service filtering criteria is met.
virtual CheckACG
Determines if call gapping should be applied.
CallFiltered
Determines if a call is filtered.


Initialise_
TDPTable
tdpTable(leg2)
(analysedInformation)
(1)!monitorMode :=
interrupted
Analysedinformation is always armed.
Note: Informative information.
Idle

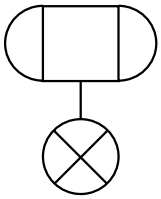

/* PROCESSING OF IN CS−1 OPERATIONS. */
Idle
ActivateService_
Filtering
(invokeID,asfArg)
ActivityTest
(invokeID,dID)
CallGap
(invokeID,cgArg)
Process_
ActivateService_
Filtering
(invokeID,asfArg)
ActivityTestResult
(invokeID,dID)
Process_
CallGap
(invokeID,cgArg)
–


Idle
Mgt_SetTrigger_
Table
(mgt_STTArg)
Mgt_SetTriggerTable
(mgt_STTArg)
–

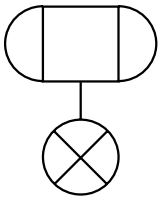

Idle
CancelStatus_
ReportRequest
RequestCurrent_
StatusReport
RequestEveryStatus_
ChangeReport
RequestFirstStatus_
MatchReport
'Process operation
in accordance with
Q.1228 section 17.'
'Process operation
in accordance with
Q.1228 section 17.'
'Process operation
in accordance with
Q.1228 section 17.'
'Process operation
in accordance with
Q.1228 section 17.'
StatusReport
via IH
_


/* Note:
Processing of service management operations used by
the SSF−FSM.
Due to a bug in SDT concerning remote procedures, the
modelling is done as explicit signal exchange between
the SSF−FSM and the SSME−FSM rather than using the more
appropriate concept of remote procedures. */
Idle
ArmTDPsReq
(csaID,csID)
MSFCReq
(csaID,csID)
CACGReq
(csaID,csID)
CFReq
(csaID,csID)
ArmTDPs
(csaID,csID)
MatchingService_
FilteringCriteria
(csaID,csID)
CheckACG
(csaID,csID)
CallFiltered
(csaID,csID)
Idle

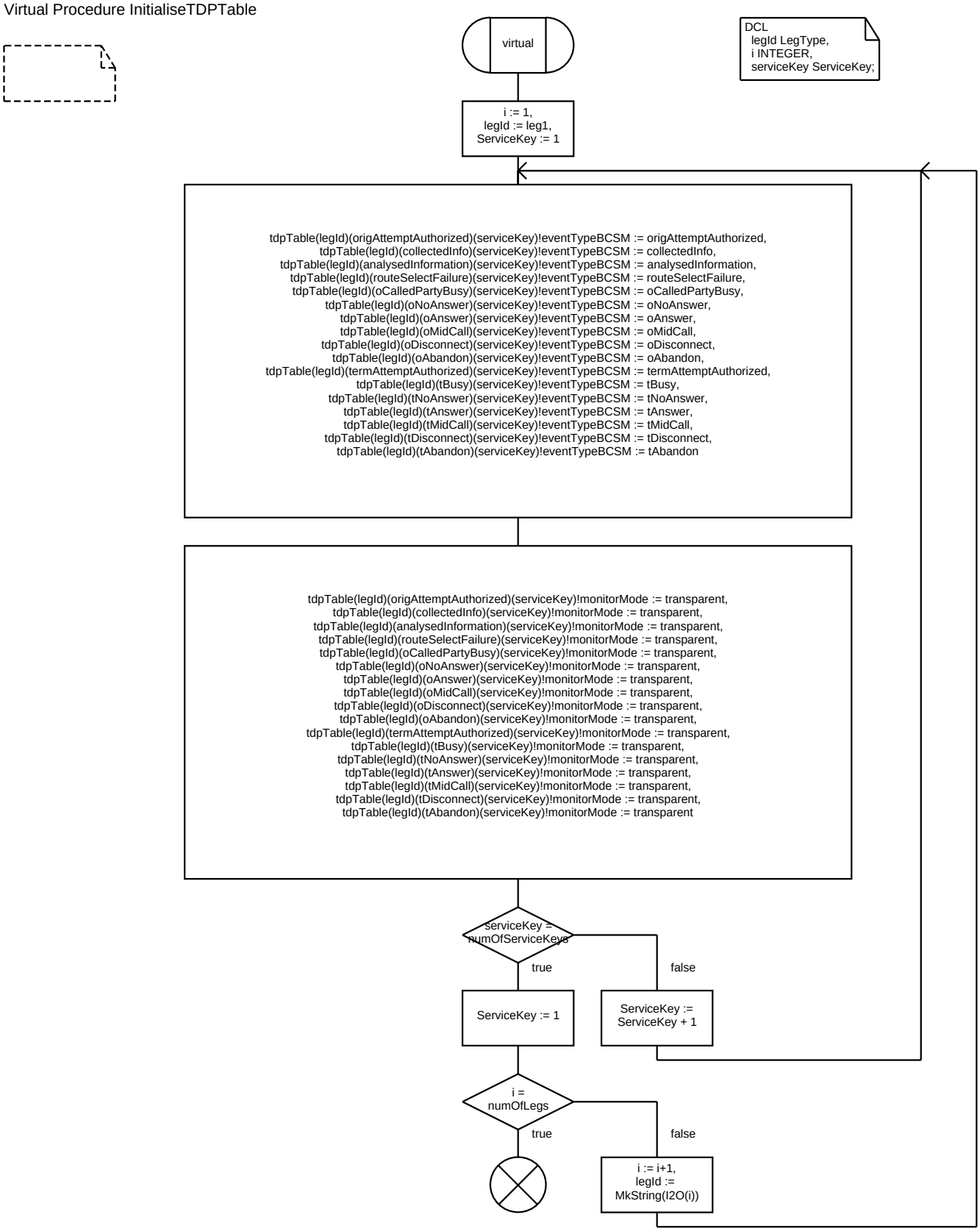
```
;FPAR
  IN invokeID Integer,
  IN asfArg   ActivateServiceFilteringArg;
```



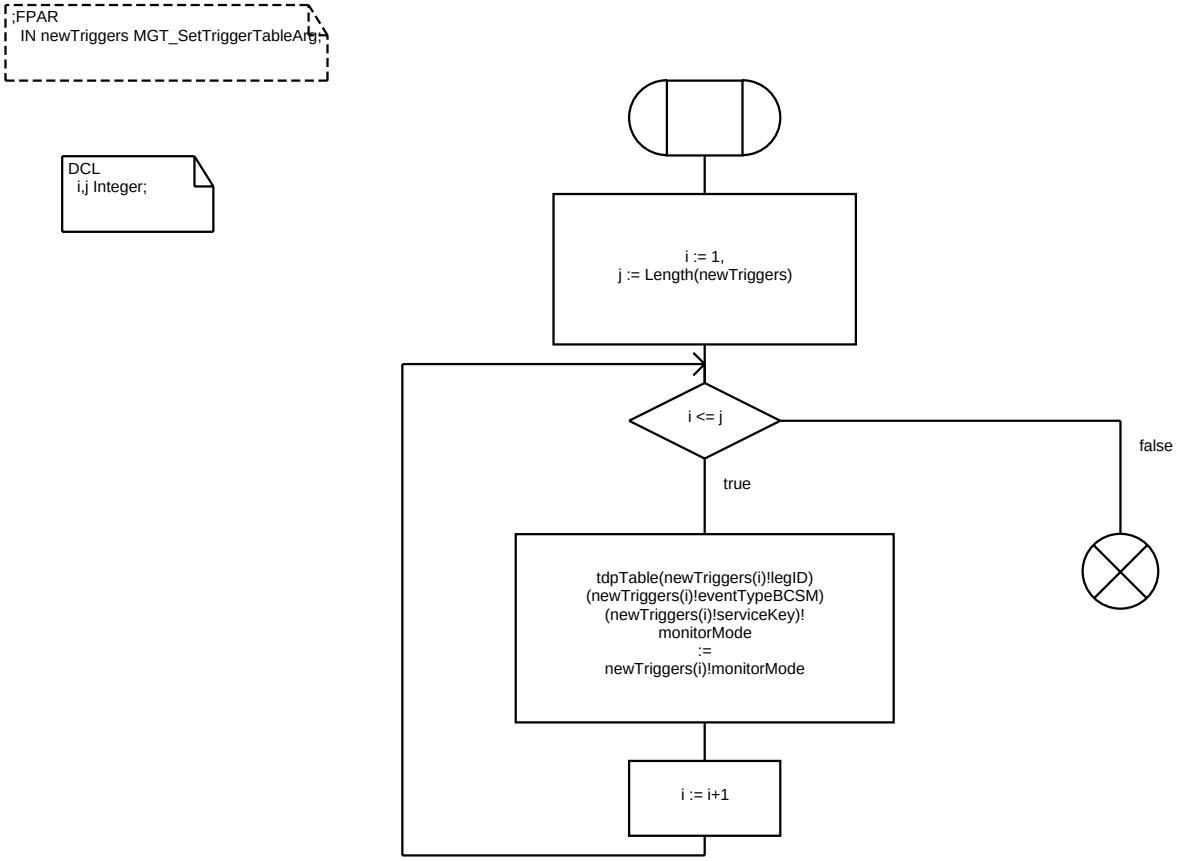
```
;FPAR
  IN invokeID Integer,
  IN cgArg    CallGapArg;
```

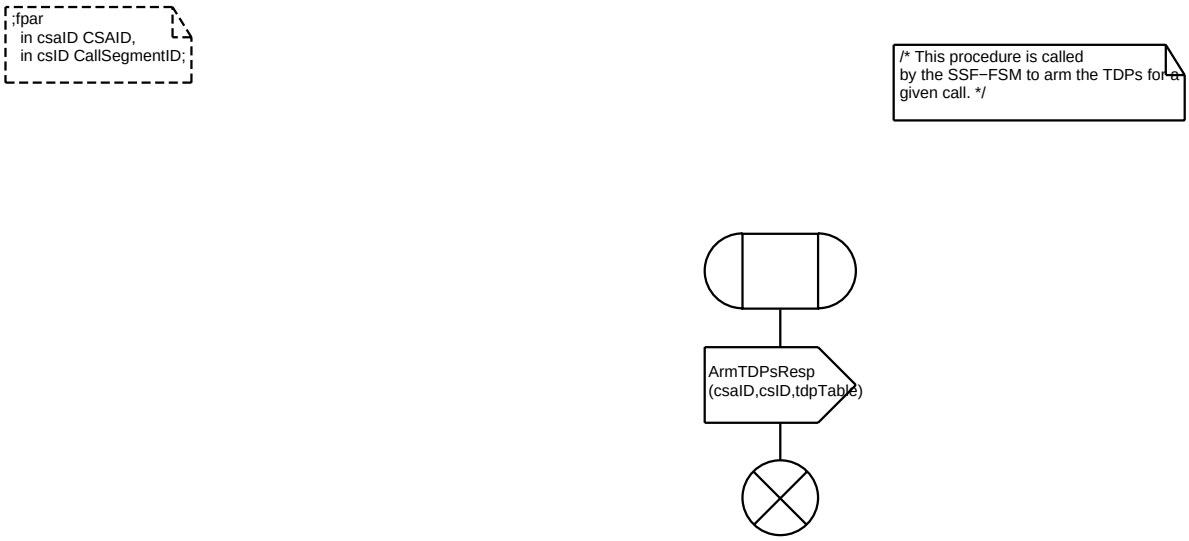


Virtual Procedure InitialiseTDPTable

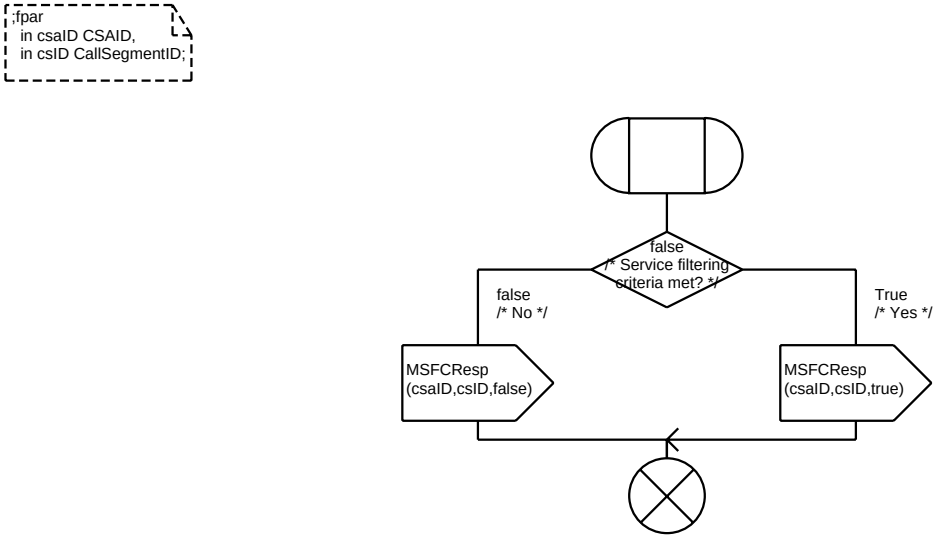


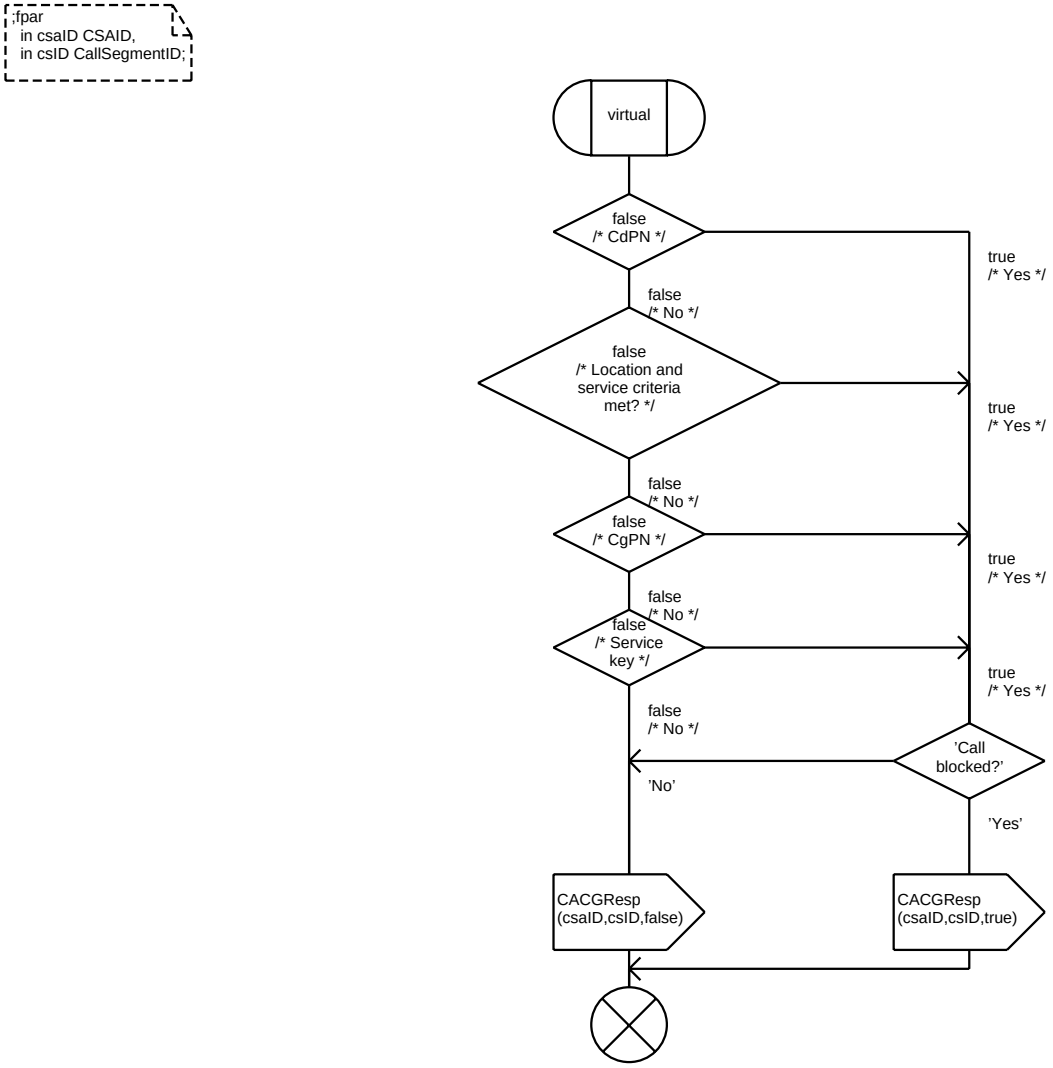
Procedure Mgt_SetTriggerTable



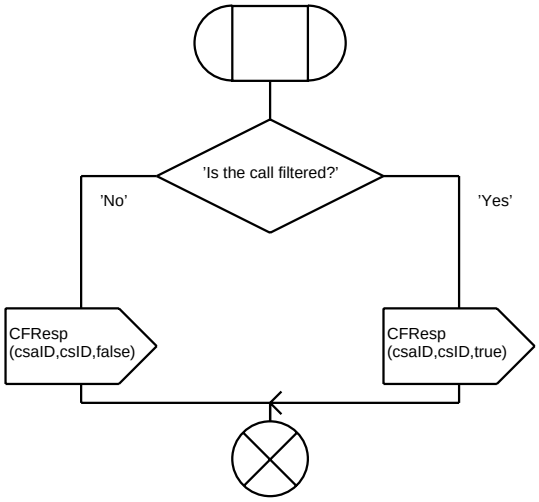

;fpar
in csaID CSAID,
in csID CallSegmentID;
/* This procedure is called
by the SSF−FSM to arm the TDPs for a
given call. */
ArmTDPsResp
(csaID,csID,tdpTable)

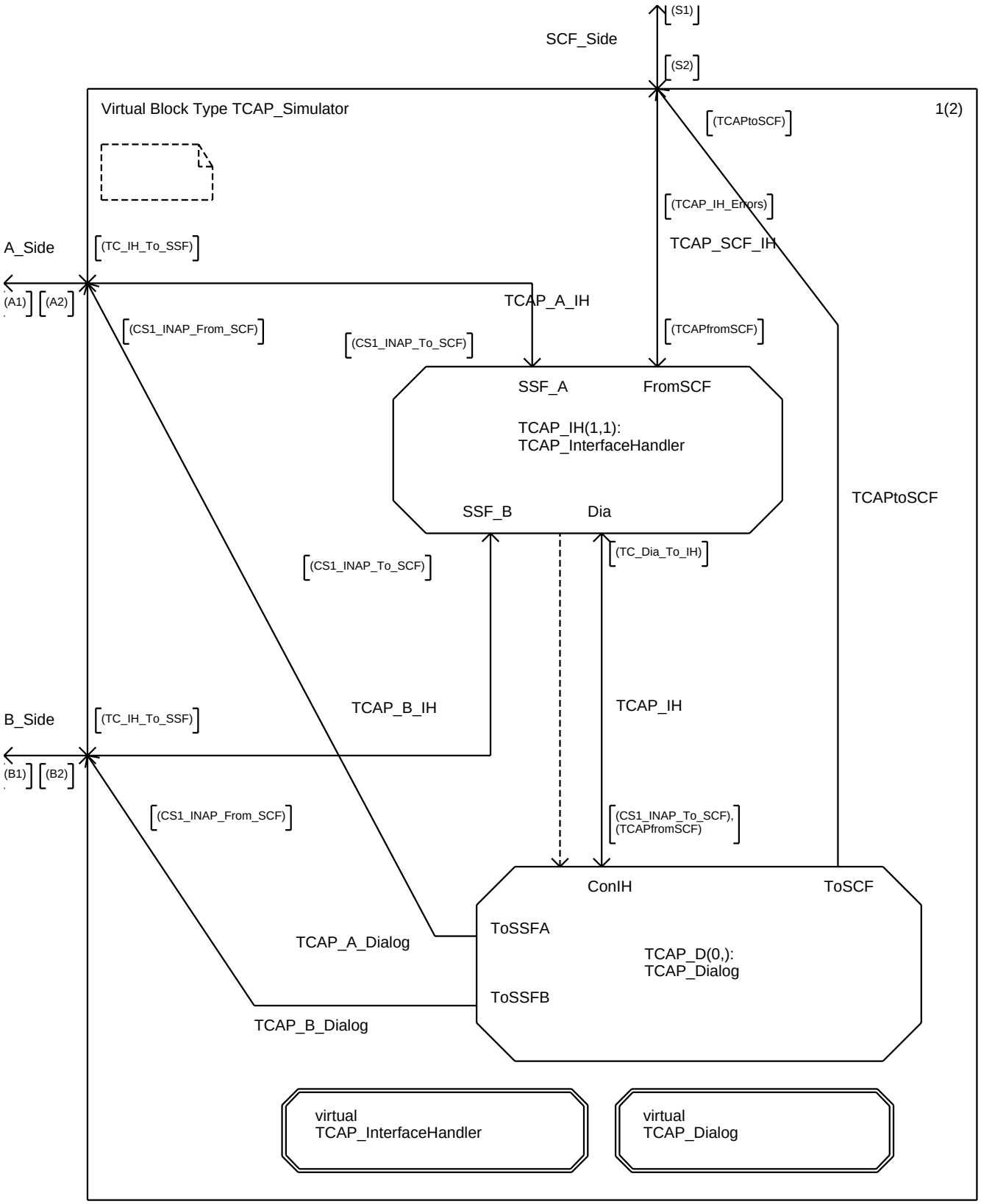
Procedure <<Process Type SSME_FSM>> MatchingServiceFilteringCriteria




;fpar
in csaID CSAID,
in csID CallSegmentID;
virtual
false
/* CdPN */
true
/* Yes */
false
/* No */
false
/* Location and
service criteria
met? */
true
/* Yes */
false
/* No */
false
/* CgPN */
true
/* Yes */
false
/* No */
false
/* Service
key */
true
/* Yes */
false
/* No */
'Call
blocked?'
'No'
'Yes'
CACGResp
(csaID,csID,false)
CACGResp
(csaID,csID,true)

```
;fpar
  in csaID CSAID,
  in csID CallSegmentID;
```




Virtual Block Type TCAP_Simulator
1(2)
SCF_Side
(S1)
(S2)
(TCAPtoSCF)
(TCAP_IH_Errors)
TCAP_SCF_IH
A_Side
(TC_IH_To_SSF)
(A1)
(A2)
TCAP_A_IH
(CS1_INAP_From_SCF)
(CS1_INAP_To_SCF)
(TCAPfromSCF)
SSF_A
FromSCF
TCAP_IH(1,1):
TCAP_InterfaceHandler
SSF_B
Dia
TCAPtoSCF
(CS1_INAP_To_SCF)
(TC_Dia_To_IH)
TCAP_B_IH
TCAP_IH
B_Side
(TC_IH_To_SSF)
(B1)
(B2)
(CS1_INAP_From_SCF)
(CS1_INAP_To_SCF),
(TCAPfromSCF)
ConIH
ToSCF
ToSSFA
TCAP_A_Dialog
TCAP_D(0,):
TCAP_Dialog
ToSSFB
TCAP_B_Dialog
virtual
TCAP_InterfaceHandler
virtual
TCAP_Dialog

```
/* Datatypedefinitions used locally in this block type */
```

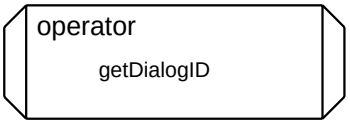
```
/* Used to describe the status of a dialog */

NEWTYPE DialogStatusType
  LITERALS free, busy, csaPending, used;
ENDNEWTYPE;
```

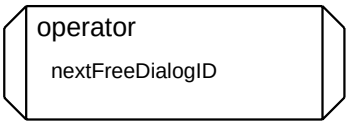
```
/* Signal− and Signallistdefinitions used locally in this block type */
```

```
/* A Dummy Signal */
SIGNAL
  ReleaseDialog(DialogIDtype);
```

```
/* TCAP dialogs to TCAP interface */

SIGNALLIST TC_Dia_To_IH =
  ReleaseDialog;
```

```
NEWTYPE DialogType
  STRUCT
    status DialogStatusType;
    csa CSAID;
    dialogHandler PId;
  OPERATORS newDialog: DialogStatusType, CSAID, PId -> DialogType;
  OPERATOR newDialog;
    FPAR diaStatus DialogStatusType, csaNew CSAID, diaHandler PId;
    RETURNS nDialog DialogType;
    REFERENCED;
ENDNEWTYPE;

/* Remark: Type CSAID (= Integer) is defined in ASN.1 file 'CS2Internals.asn' */

NEWTYPE AllDialogsType
  ARRAY (DialogIDtype, DialogType); /* DialogID is used as index in the array */
  OPERATORS
    initDialogs: Boolean -> allDialogsType;
    getDialogID: CSAID, AllDialogsType -> DialogIDtype;
    nextFreeDialogID: AllDialogsType -> DialogIDtype;
 OPERATOR initDialogs;
    FPAR dummy Boolean;
    RETURNS newDialogs AllDialogsType;
    REFERENCED;
  OPERATOR getDialogID;
    FPAR knownCSA CSAID, allDialogs AllDialogsType;
    RETURNS dialogID DialogIDtype;
    REFERENCED;
  OPERATOR nextFreeDialogID;
    FPAR allDialogs AllDialogsType;
    RETURNS dialogID DialogIDtype;
    REFERENCED;
ENDNEWTYPE;
```



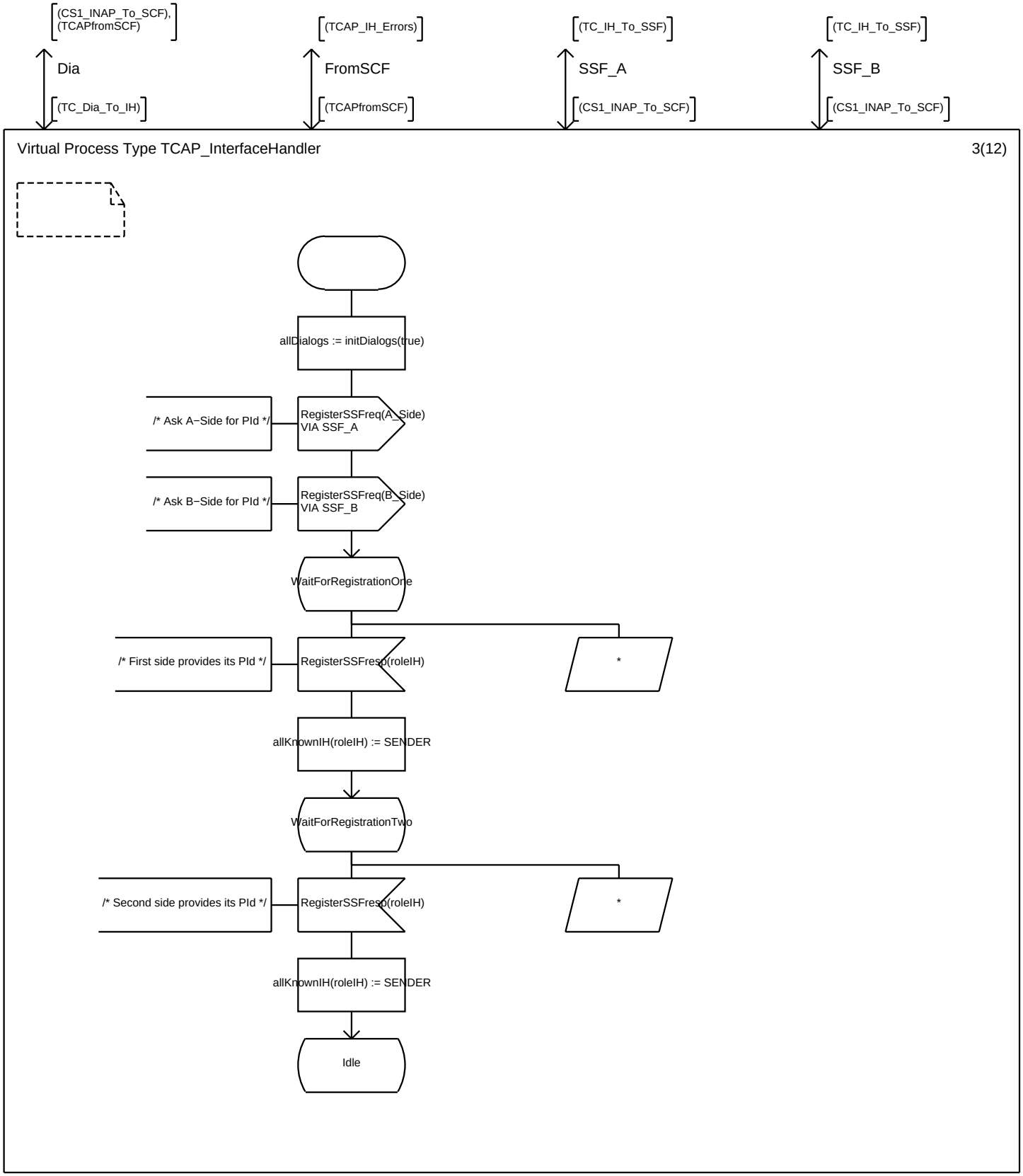


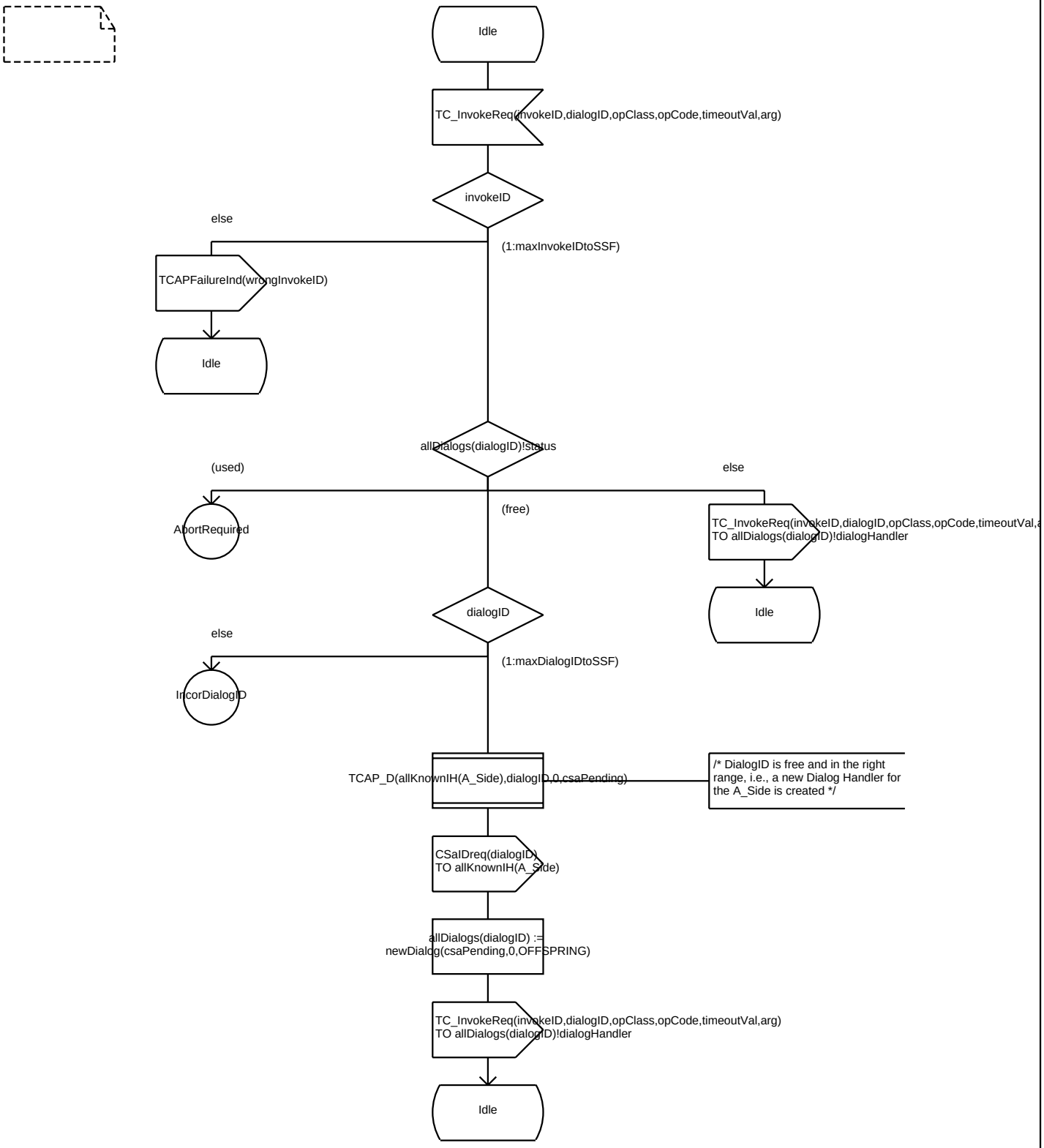

```
NEWTYPE IHresourceType
  ARRAY (IHroleType,PId); /* IHroleType (= A_Side or B_Side) is defined on system level */
ENDNEWTYPE;
```

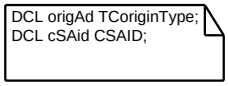
```
DCL allKnownIH IHresourceType;
DCL roleIH IHroleType;

DCL allDialogs AllDialogsType;
```

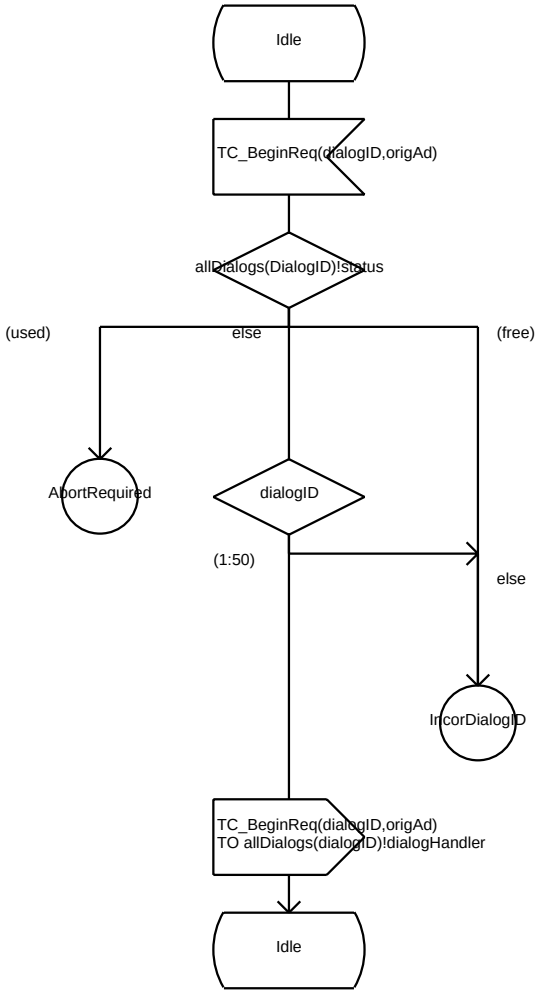
```
DCL invokeID InvokeIDtype;
DCL dialogID DialogIDtype;
DCL opClass OpClassType;
DCL opCode OpCodeType;
DCL timeoutVal TimeoutValType;
```

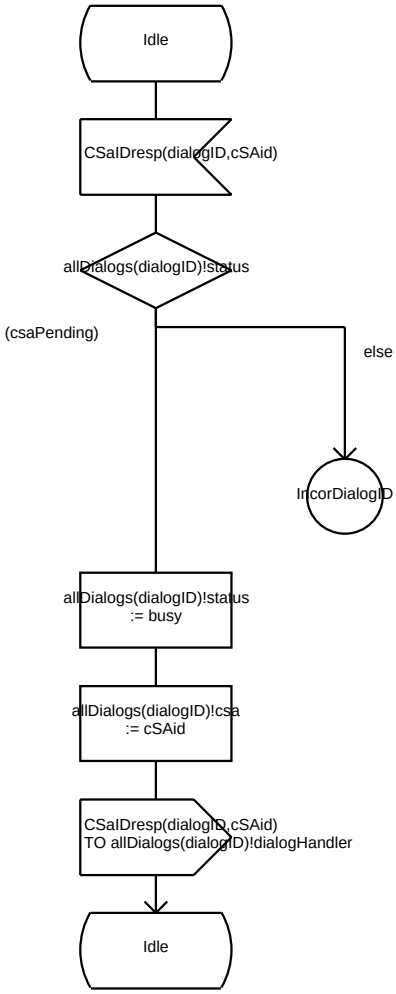
```
DCL arg ArgType; /* refers to ASN.1 Definition in INCS2BundleArg */
DCL errArg errorArg;
DCL compEnd Boolean;
```

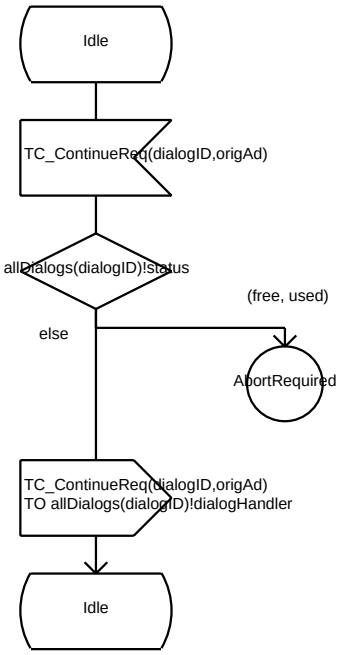

(CS1_INAP_To_SCF),
(TCAPfromSCF)
(TCAP_IH_Errors)
(TC_IH_To_SSF)
(TC_IH_To_SSF)
Dia
FromSCF
SSF_A
SSF_B
(TC_Dia_To_IH)
(TCAPfromSCF)
(CS1_INAP_To_SCF)
(CS1_INAP_To_SCF)
Virtual Process Type TCAP_InterfaceHandler
3(12)
allDialogs := initDialogs(true)
/* Ask A−Side for PId */
RegisterSSFreq(A_Side)
VIA SSF_A
/* Ask B−Side for PId */
RegisterSSFreq(B_Side)
VIA SSF_B
WaitForRegistrationOne
/* First side provides its PId */
RegisterSSFresp(roleIH)
*
allKnownIH(roleIH) := SENDER
WaitForRegistrationTwo
/* Second side provides its PId */
RegisterSSFresp(roleIH)
*
allKnownIH(roleIH) := SENDER
Idle

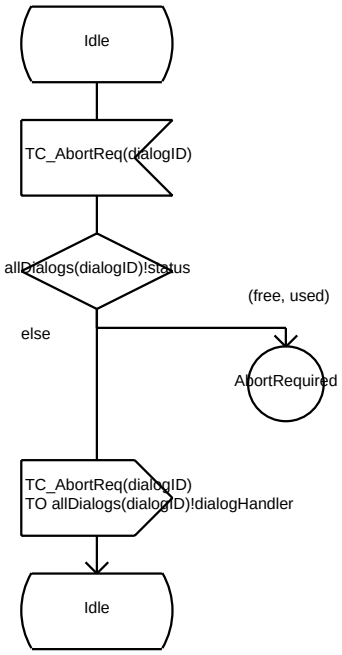

Idle
TC_InvokeReq(invokeID,dialogID,opClass,opCode,timeoutVal,arg)
invokeID
else
(1:maxInvokeIDtoSSF)
TCAPFailureInd(wrongInvokeID)
Idle
allDialogs(dialogID)!status
(used)
else
(free)
AbortRequired
TC_InvokeReq(invokeID,dialogID,opClass,opCode,timeoutVal,a
TO allDialogs(dialogID)!dialogHandler
Idle
dialogID
else
(1:maxDialogIDtoSSF)
IncorDialogID
TCAP_D(allKnownIH(A_Side),dialogID,0,csaPending)
/* DialogID is free and in the right range, i.e., a new Dialog Handler for the A_Side is created */
CSaIDreq(dialogID)
TO allKnownIH(A_Side)
allDialogs(dialogID) :=
newDialog(csaPending,0,OFFSPRING)
TC_InvokeReq(invokeID,dialogID,opClass,opCode,timeoutVal,arg)
TO allDialogs(dialogID)!dialogHandler
Idle

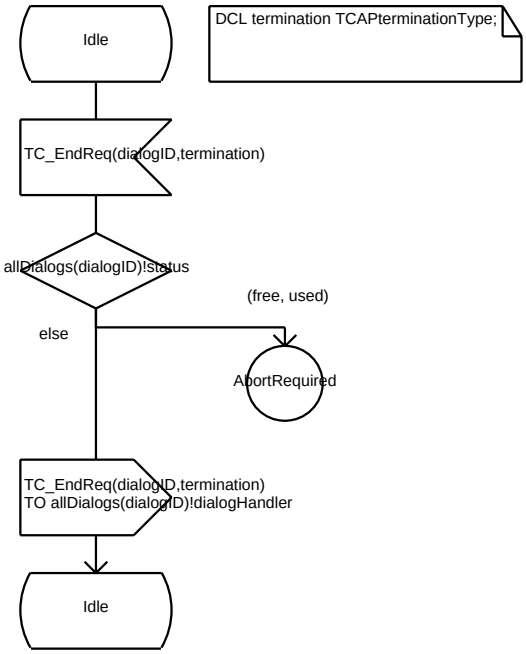

DCL origAd TCoriginType;
DCL cSAid CSAID;

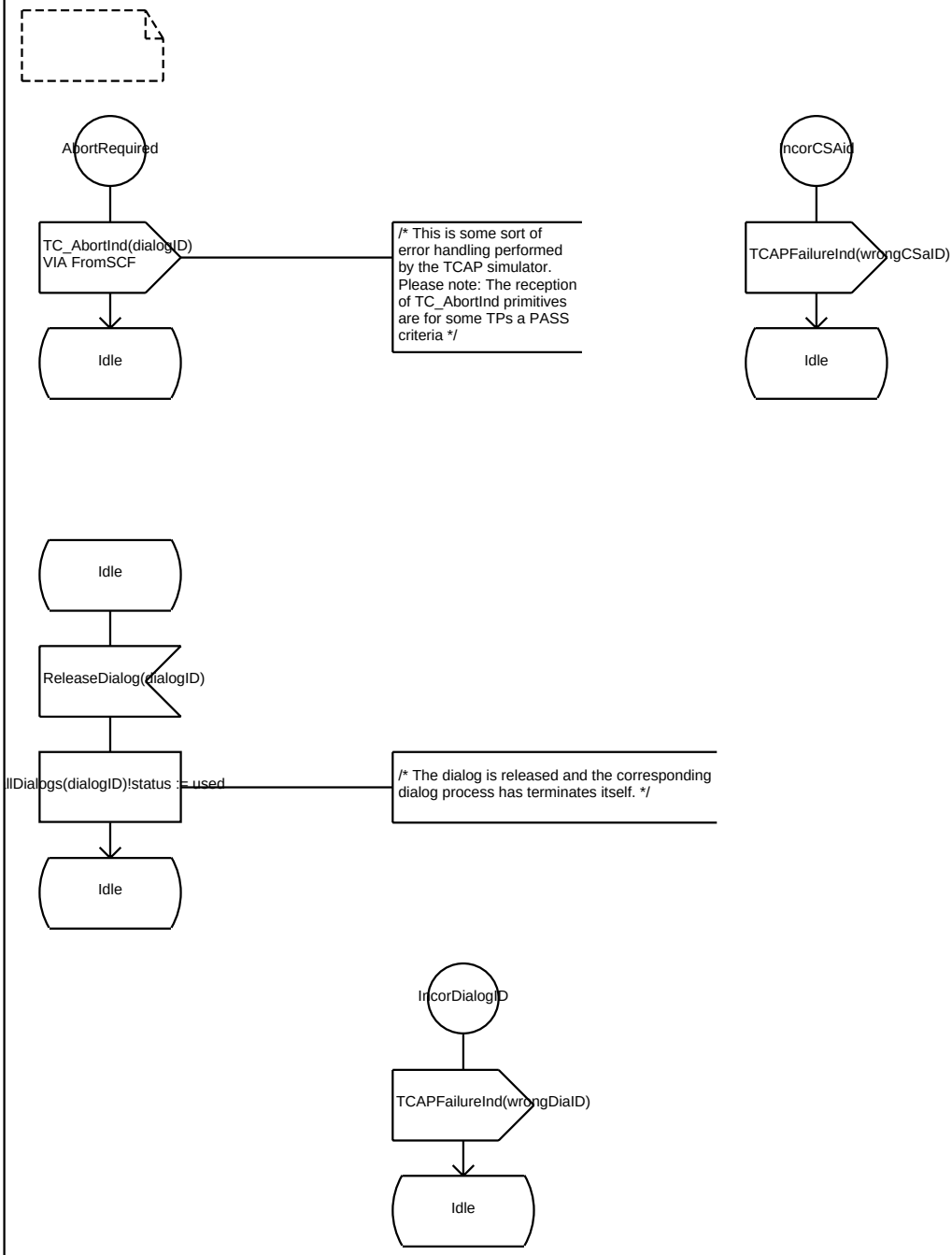

Idle
TC_BeginReq(dialogID,origAd)
allDialogs(DialogID)!status
(used)
else
(free)
AbortRequired
dialogID
(1:50)
else
IncorDialogID
TC_BeginReq(dialogID,origAd)
TO allDialogs(dialogID)!dialogHandler
Idle

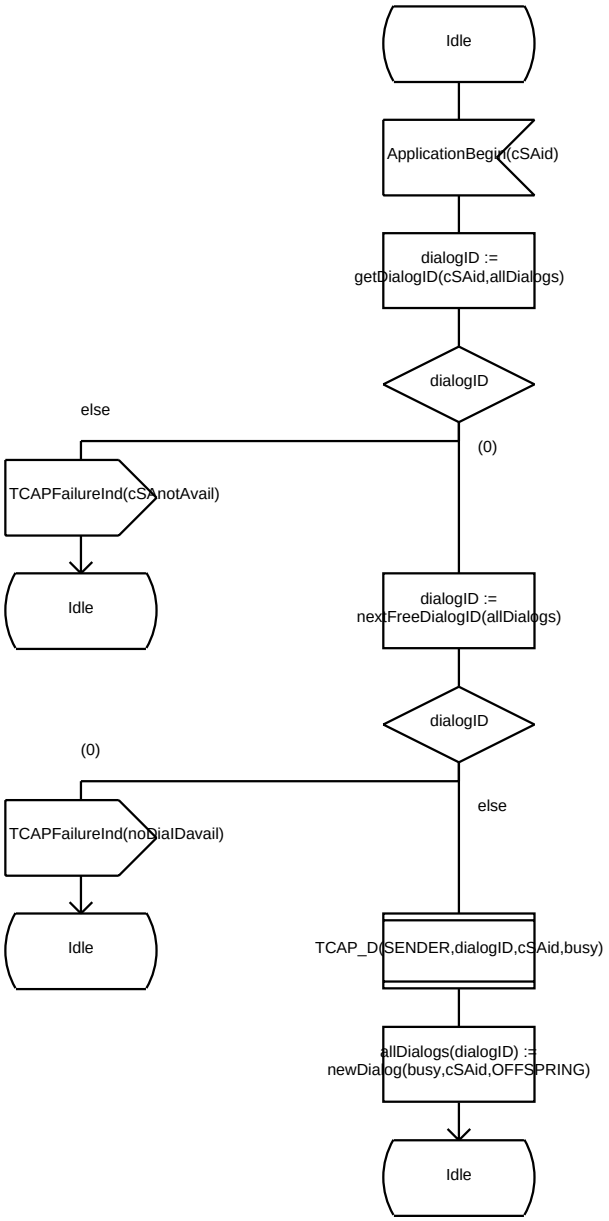

Idle
CSaIDresp(dialogID,cSAid)
allDialogs(dialogID)!status
(csaPending)
else
IncorDialogID
allDialogs(dialogID)!status
:= busy
allDialogs(dialogID)!csa
:= cSAid
CSaIDresp(dialogID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle

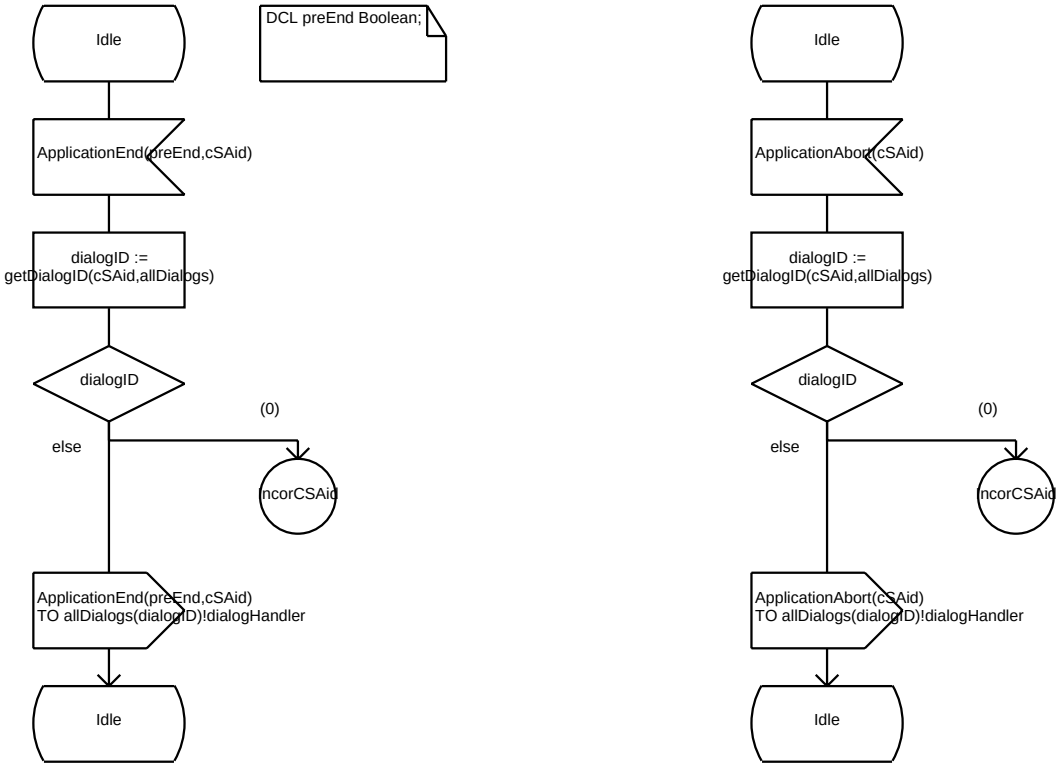

Idle
TC_ContinueReq(dialogID,origAd)
allDialogs(dialogID)!status
(free, used)
else
AbortRequired
TC_ContinueReq(dialogID,origAd)
TO allDialogs(dialogID)!dialogHandler
Idle

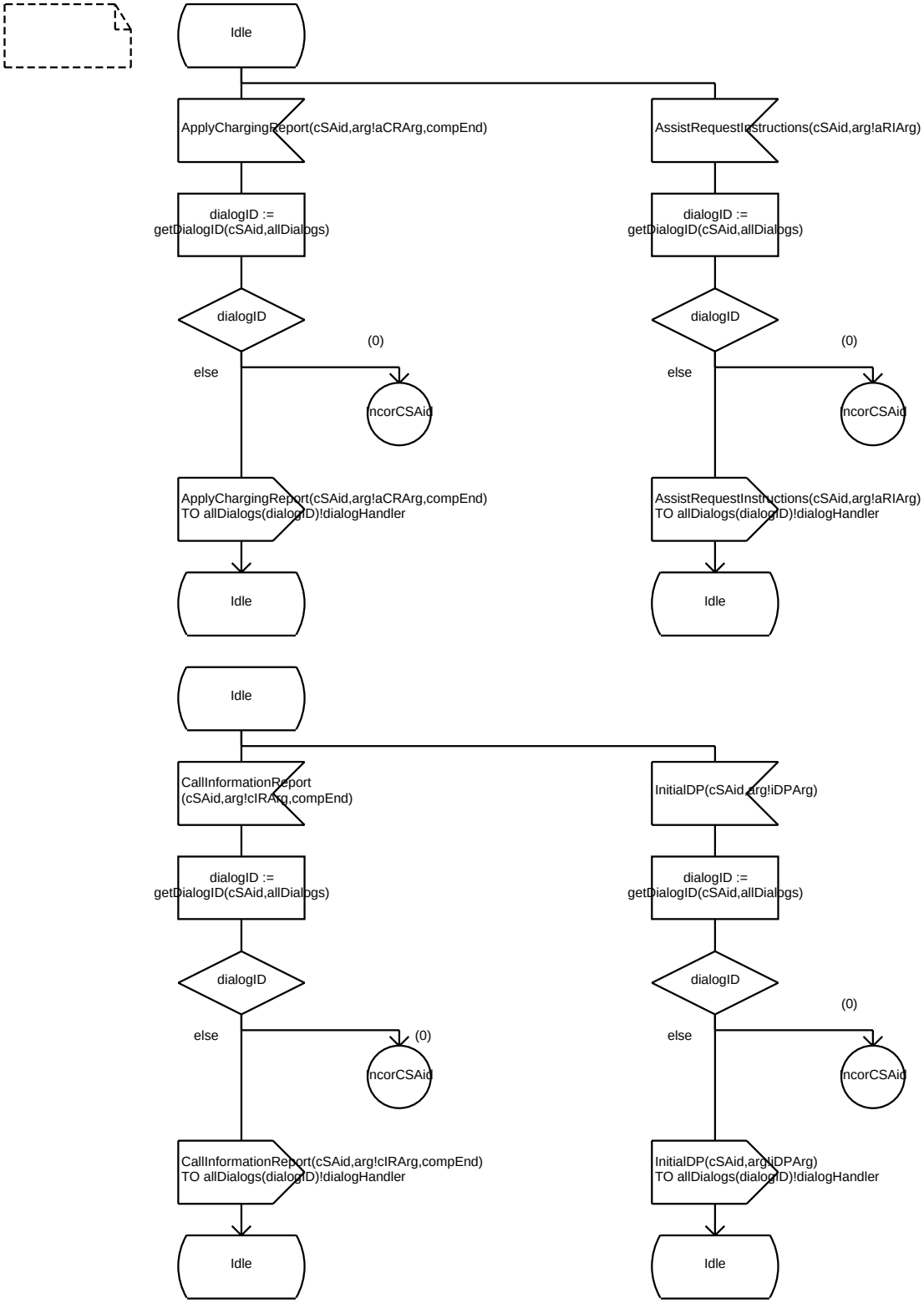

Idle
TC_AbortReq(dialogID)
allDialogs(dialogID)!status
(free, used)
else
AbortRequired
TC_AbortReq(dialogID)
TO allDialogs(dialogID)!dialogHandler
Idle

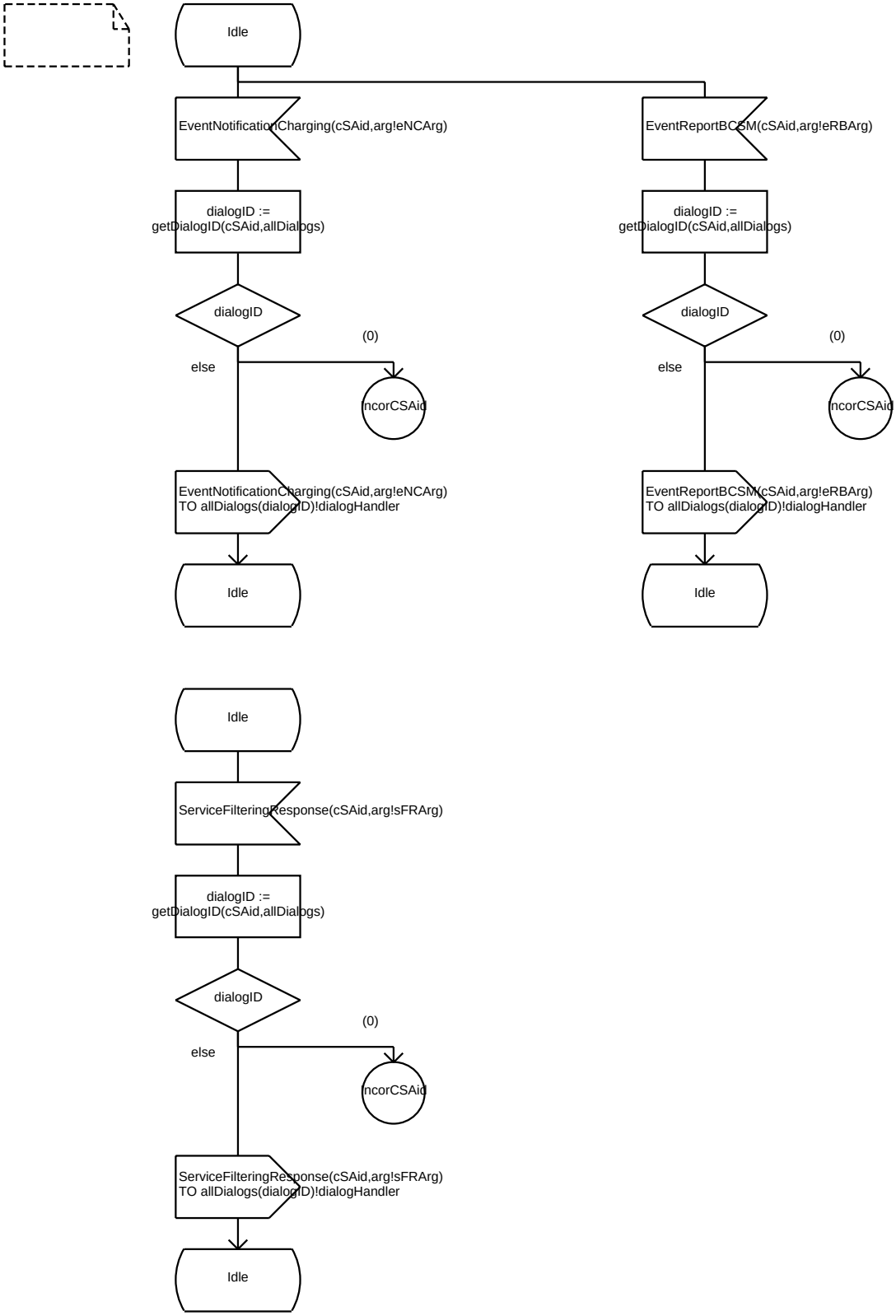

Idle
DCL termination TCAPterminationType;
TC_EndReq(dialogID,termination)
allDialogs(dialogID)!status
(free, used)
else
AbortRequired
TC_EndReq(dialogID,termination)
TO allDialogs(dialogID)!dialogHandler
Idle

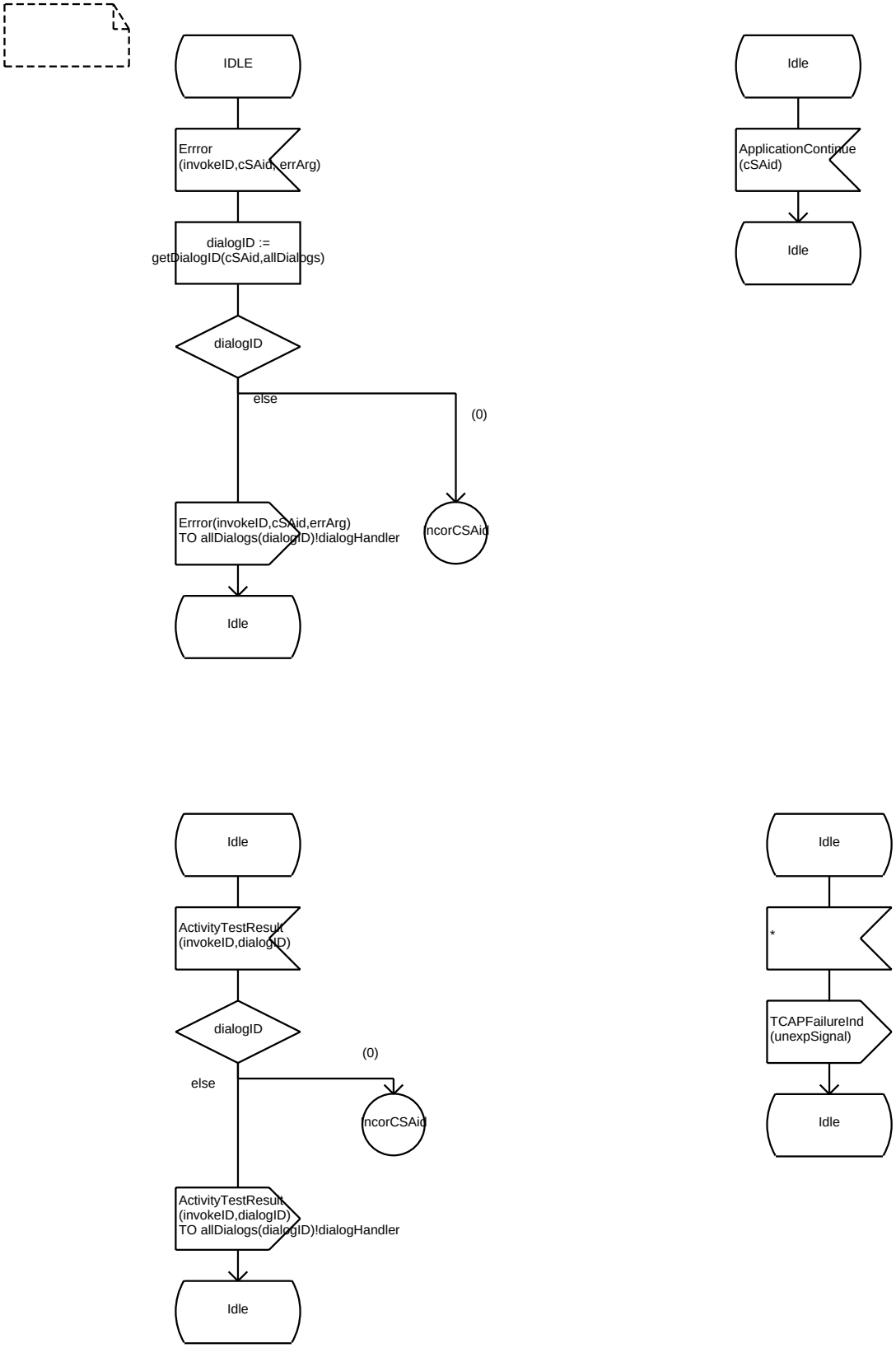

AbortRequired
TC_AbortInd(dialogID)
VIA FromSCF
/* This is some sort of error handling performed by the TCAP simulator. Please note: The reception of TC_AbortInd primitives are for some TPs a PASS criteria */
Idle
IncorCSAid
TCAPFailureInd(wrongCSaID)
Idle
Idle
ReleaseDialog(dialogID)
llDialogs(dialogID)!status := used
/* The dialog is released and the corresponding dialog process has terminates itself. */
Idle
IncorDialogID
TCAPFailureInd(wrongDiaID)
Idle


Idle
ApplicationBegin(cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
else
(0)
TCAPFailureInd(cSAnotAvail)
Idle
dialogID :=
nextFreeDialogID(allDialogs)
dialogID
(0)
else
TCAPFailureInd(noDialDavail)
Idle
TCAP_D(SENDER,dialogID,cSAid,busy)
allDialogs(dialogID) :=
newDialog(busy,cSAid,OFFSPRING)
Idle

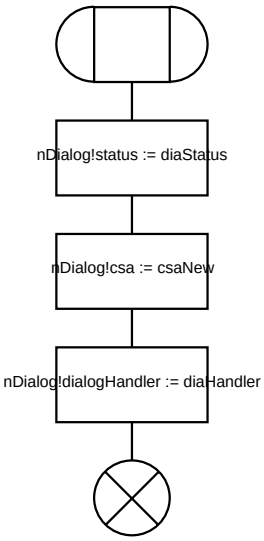

Idle
DCL preEnd Boolean;
ApplicationEnd(preEnd,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
IncorCSAid
ApplicationEnd(preEnd,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
ApplicationAbort(cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
IncorCSAid
ApplicationAbort(cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle


Idle
ApplyChargingReport(cSAid,arg!aCRArg,compEnd)
AssistRequestInstructions(cSAid,arg!aRIArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
dialogID
(0)
(0)
else
else
IncorCSAid
IncorCSAid
ApplyChargingReport(cSAid,arg!aCRArg,compEnd)
TO allDialogs(dialogID)!dialogHandler
AssistRequestInstructions(cSAid,arg!aRIArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
Idle
CallInformationReport
(cSAid,arg!cIRArg,compEnd)
InitialDP(cSAid,arg!iDPArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
dialogID
(0)
(0)
else
else
IncorCSAid
IncorCSAid
CallInformationReport(cSAid,arg!cIRArg,compEnd)
TO allDialogs(dialogID)!dialogHandler
InitialDP(cSAid,arg!iDPArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle

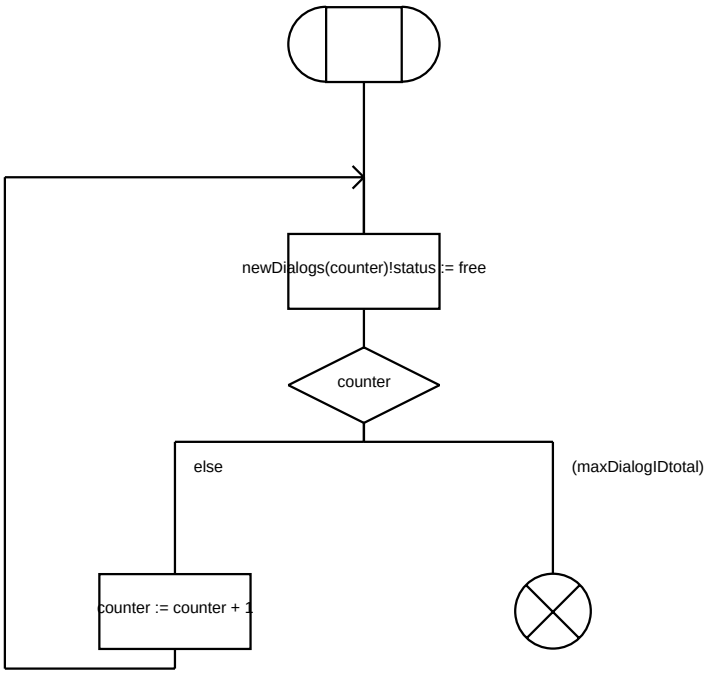

Idle
EventNotificationCharging(cSAid,arg!eNCArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
IncorCSAid
EventNotificationCharging(cSAid,arg!eNCArg)
TO allDialogs(dialogID)!dialogHandler
Idle
EventReportBCSM(cSAid,arg!eRBArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
IncorCSAid
EventReportBCSM(cSAid,arg!eRBArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
ServiceFilteringResponse(cSAid,arg!sFRArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
IncorCSAid
ServiceFilteringResponse(cSAid,arg!sFRArg)
TO allDialogs(dialogID)!dialogHandler
Idle

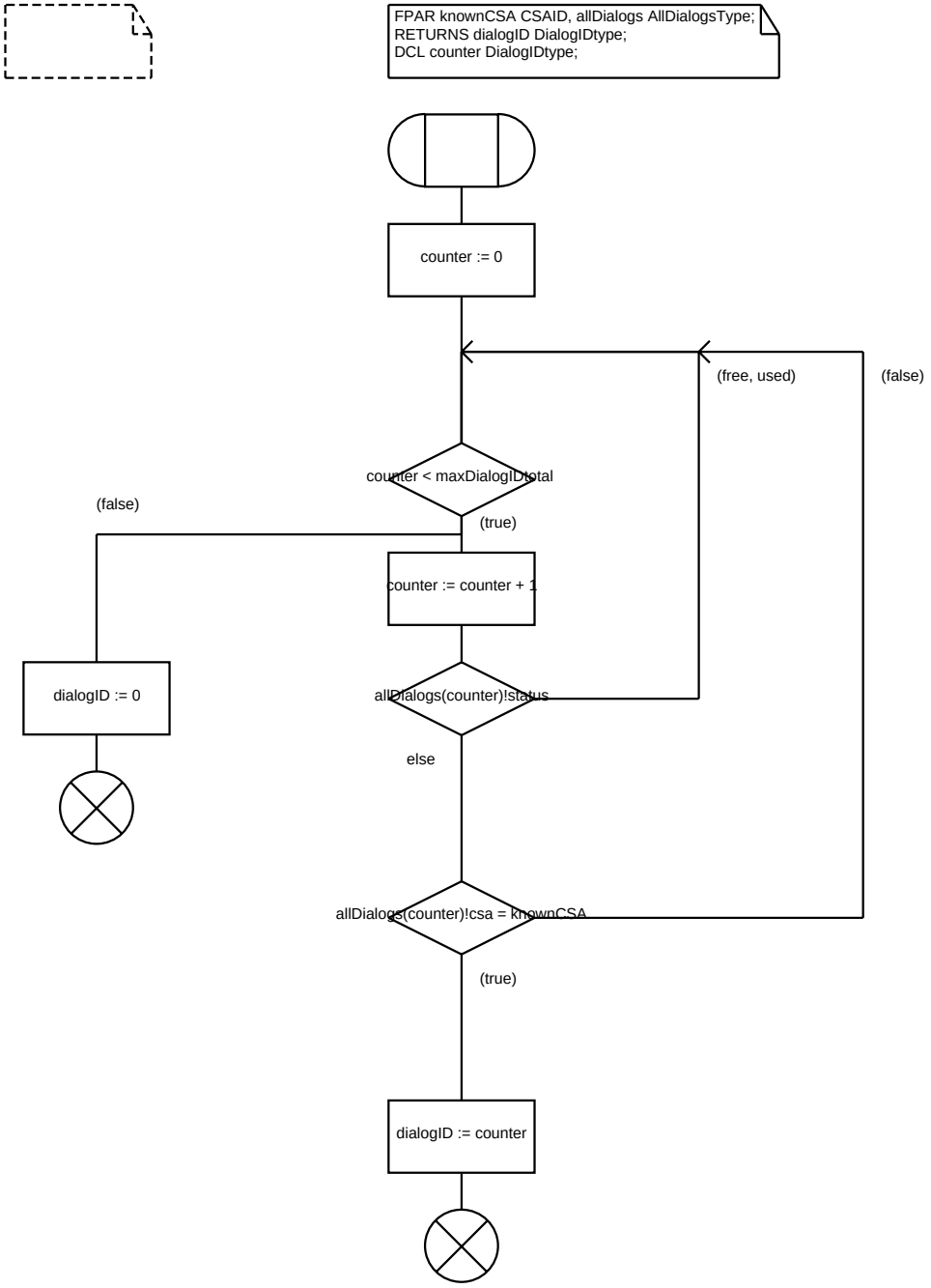

IDLE
Errror
(invokeID,cSAid,errArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
else
(0)
Errror(invokeID,cSAid,errArg)
TO allDialogs(dialogID)!dialogHandler
IncorCSAid
Idle
Idle
ApplicationContinue
(cSAid)
Idle
Idle
ActivityTestResult
(invokeID,dialogID)
dialogID
(0)
else
IncorCSAid
ActivityTestResult
(invokeID,dialogID)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
*
TCAPFailureInd
(unexpSignal)
Idle

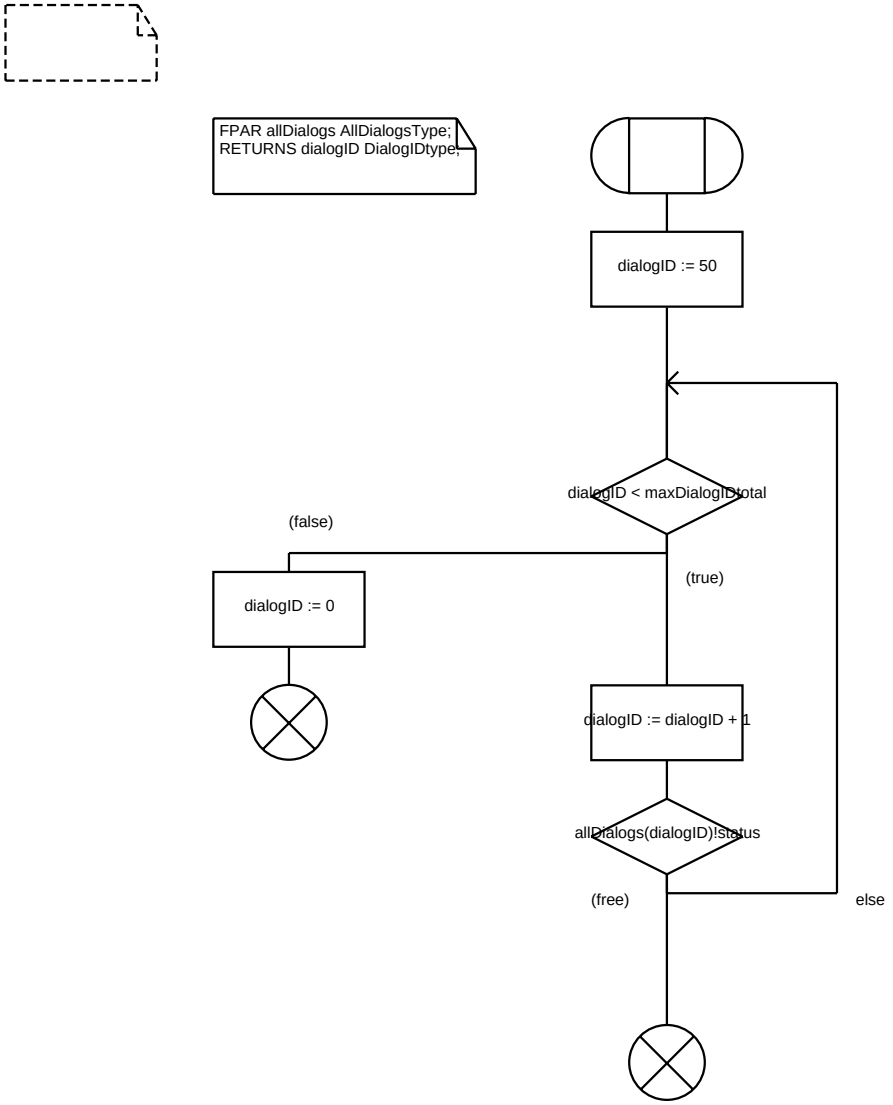
FPAR diaStatus DialogStatusType, csaNew CSAID, diaHandler PId;
RETURNS nDialog DialogType;

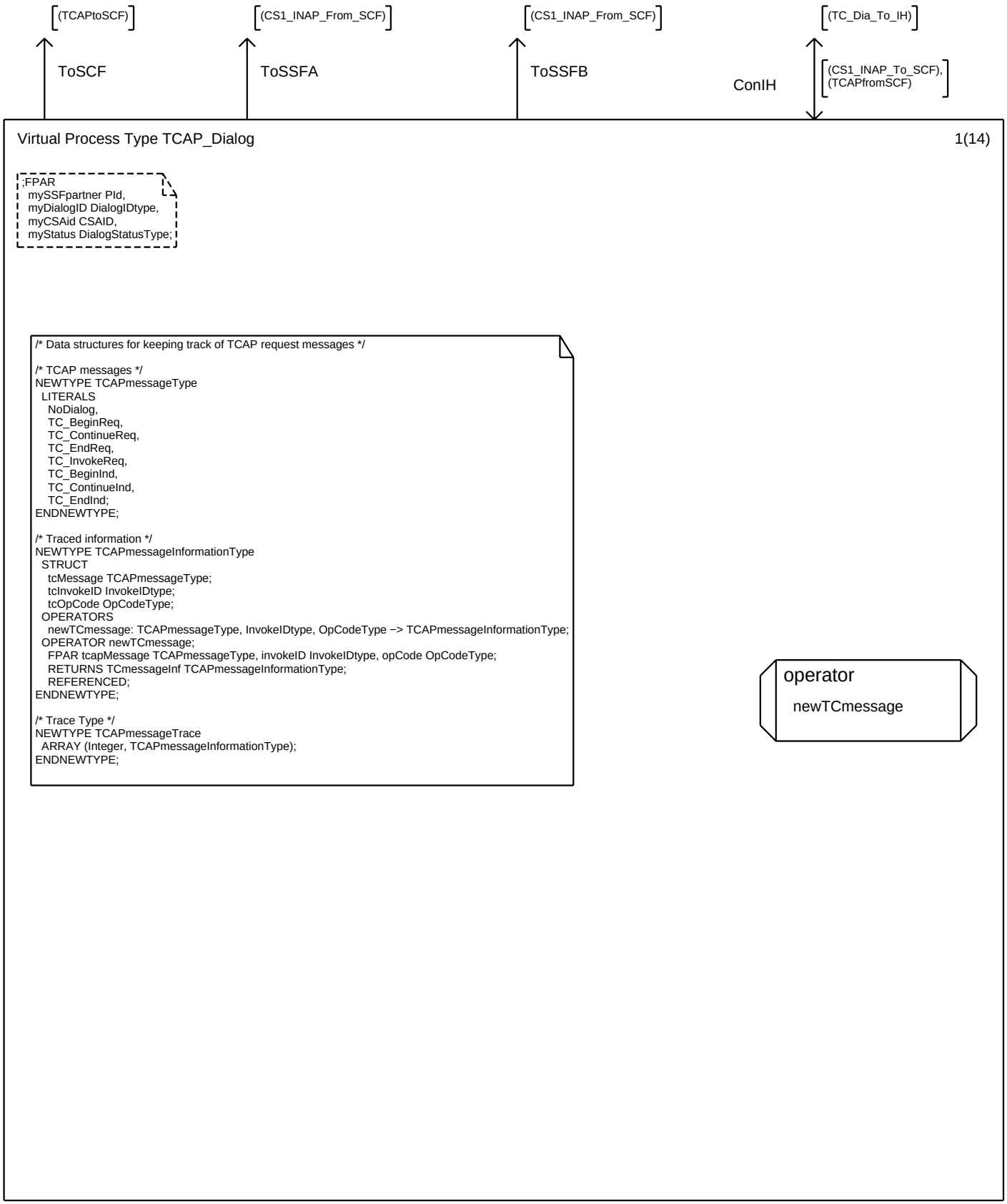


```
FPAR dummy Boolean;
RETURNS newDialogs AllDialogsType;

DCL counter DialogIDtype := 1;
```




FPAR knownCSA CSAID, allDialogs AllDialogsType;
RETURNS dialogID DialogIDtype;
DCL counter DialogIDtype;
counter := 0
(free, used)
(false)
counter < maxDialogIDtotal
(false)
(true)
counter := counter + 1
dialogID := 0
allDialogs(counter)!status
else
allDialogs(counter)!csa = knownCSA
(true)
dialogID := counter

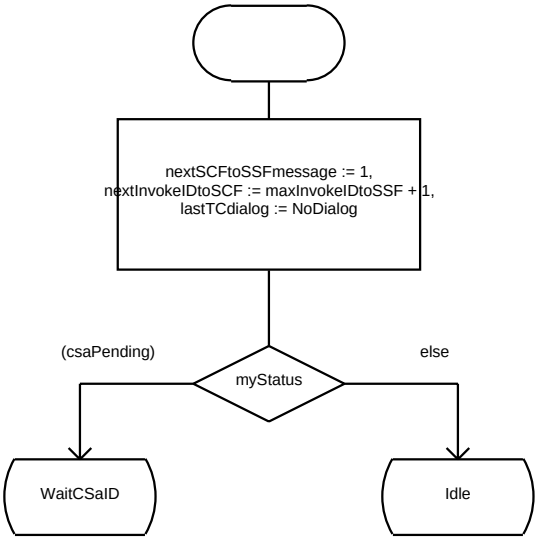

FPAR allDialogs AllDialogsType;
RETURNS dialogID DialogIDtype;
dialogID := 50
dialogID < maxDialogIDtotal
(false)
(true)
dialogID := 0
dialogID := dialogID + 1
allDialogs(dialogID)!status
(free)
else

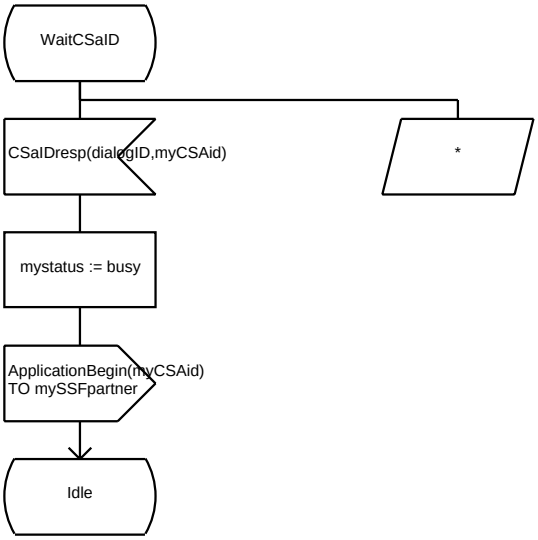

(TCAPtoSCF)
(CS1_INAP_From_SCF)
(CS1_INAP_From_SCF)
(TC_Dia_To_IH)
ToSCF
ToSSFA
ToSSFB
ConIH
(CS1_INAP_To_SCF),
(TCAPfromSCF)
Virtual Process Type TCAP_Dialog
1(14)
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
/* Data structures for keeping track of TCAP request messages */
/* TCAP messages */
NEWTYPE TCAPmessageType
LITERALS
NoDialog,
TC_BeginReq,
TC_ContinueReq,
TC_EndReq,
TC_InvokeReq,
TC_BeginInd,
TC_ContinueInd,
TC_EndInd;
ENDNEWTYPE;
/* Traced information */
NEWTYPE TCAPmessageInformationType
STRUCT
tcMessage TCAPmessageType;
tcInvokeID InvokeIDtype;
tcOpCode OpCodeType;
OPERATORS
newTCmessage: TCAPmessageType, InvokeIDtype, OpCodeType −> TCAPmessageInformationType;
OPERATOR newTCmessage;
FPAR tcapMessage TCAPmessageType, invokeID InvokeIDtype, opCode OpCodeType;
RETURNS TCmessageInf TCAPmessageInformationType;
REFERENCED;
ENDNEWTYPE;
/* Trace Type */
NEWTYPE TCAPmessageTrace
ARRAY (Integer, TCAPmessageInformationType);
ENDNEWTYPE;
operator
newTCmessage

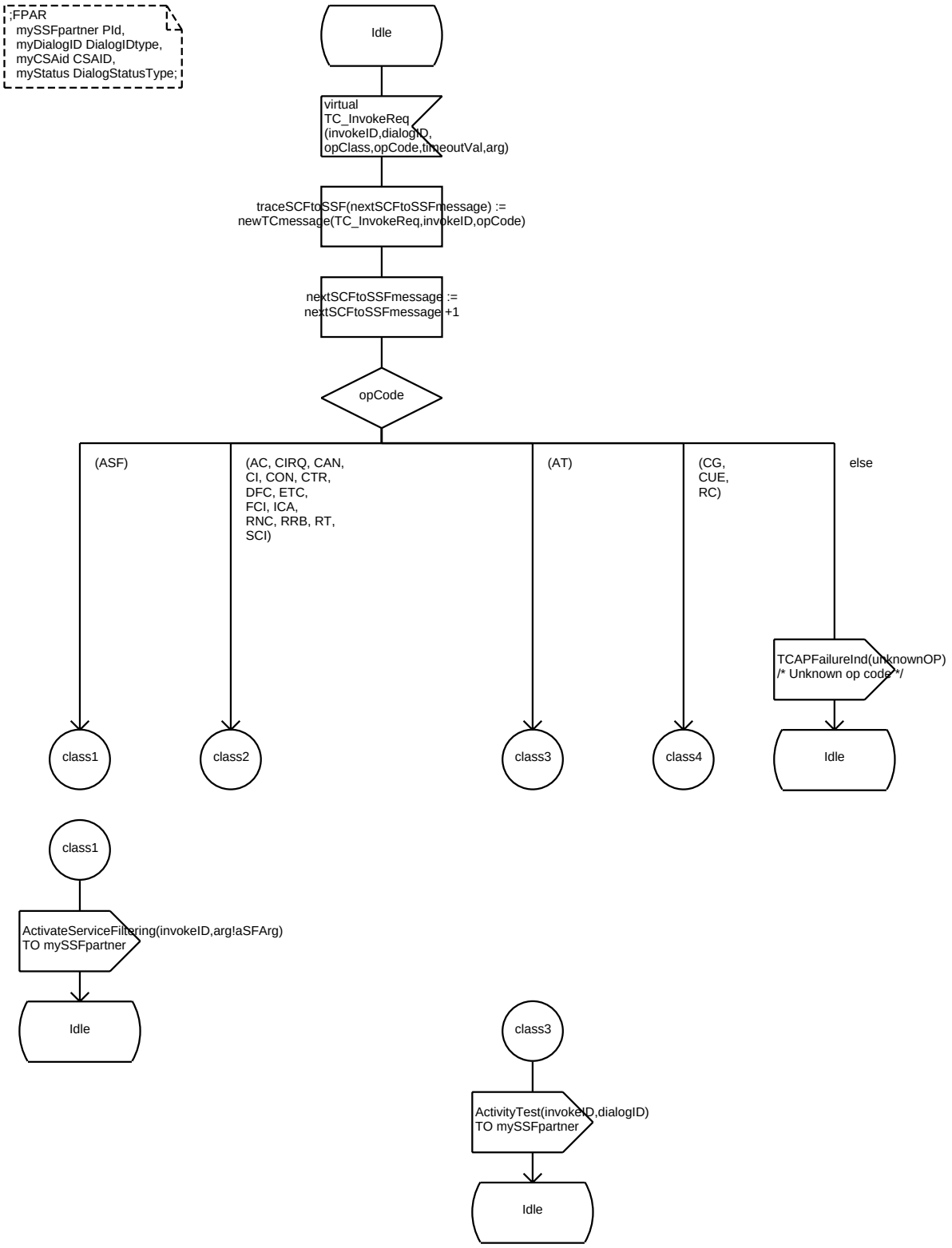
```
;FPAR
  mySSFpartner PId,
  myDialogID DialogIDtype,
  myCSAid CSAID,
  myStatus DialogStatusType;
```

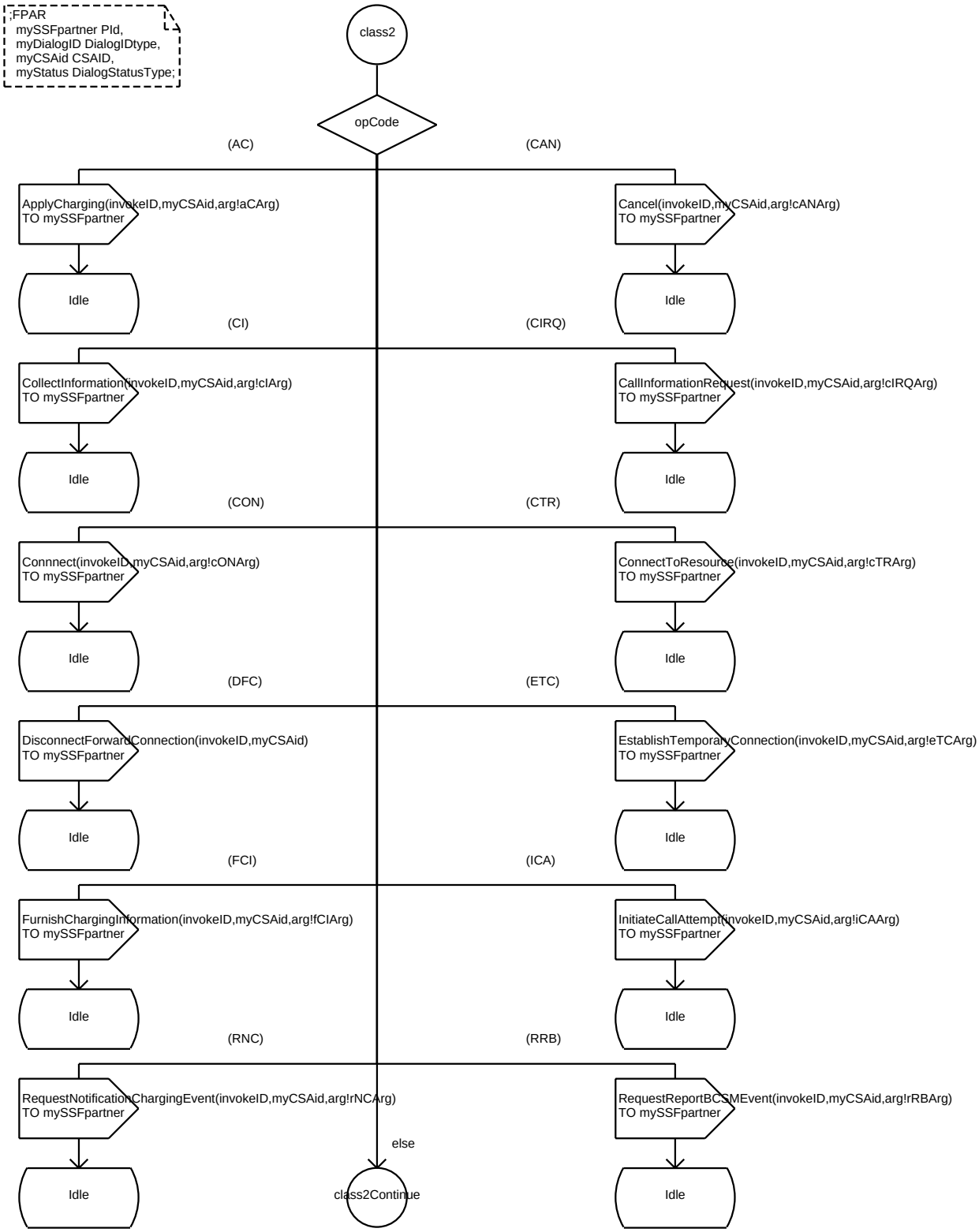
```
/* State variables, to be updated during the entire lifetime of this process */

DCL traceSCFtoSSF TCAPmessageTrace;
DCL nextSCFtoSSFmessage Integer;

DCL lastTCdialog TCAPmessageType;

DCL nextInvokeIDtoSCF InvokeIDtype;

/* To be used to receive message parameters */

DCL dialogID DialogIDtype;
DCL invokeID InvokeIDtype;
DCL opClass OpClassType;
DCL opCode OpCodeType;
DCL timeoutVal TimeoutValType;
DCL origAd TCoriginType;
DCL termination TCAPterminationType;
DCL cSAid CSAID;
DCL compEnd Boolean := false;

DCL arg ArgType; /* refers to ASN.1 definition INCS2BundleArg */
DCL errArg ErrorArg;
```

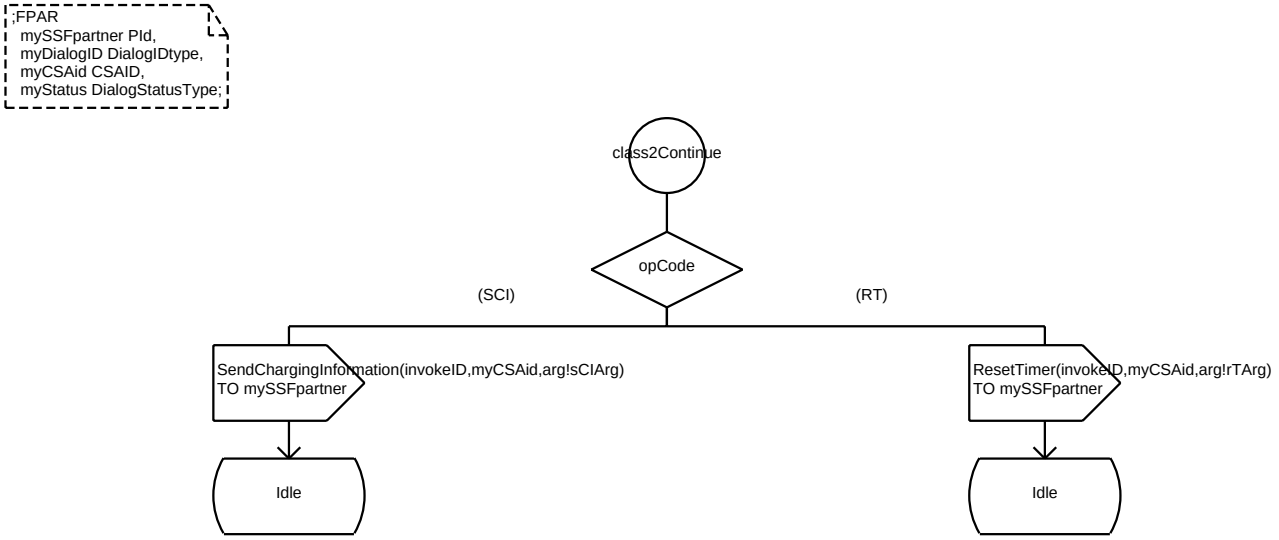
```
;FPAR
  mySSFpartner PId,
  myDialogID DialogIDtype,
  myCSAid CSAID,
  myStatus DialogStatusType;
```

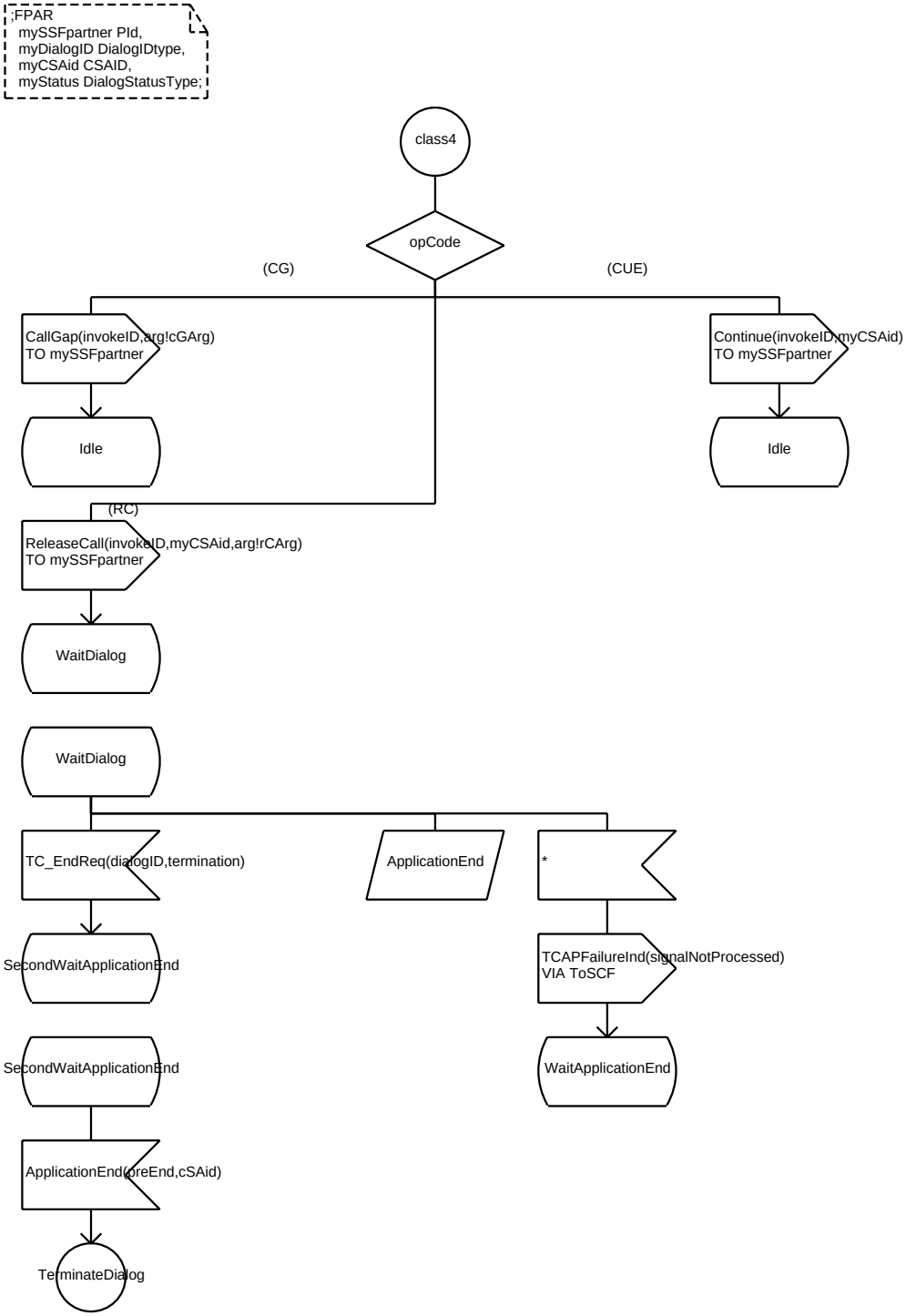


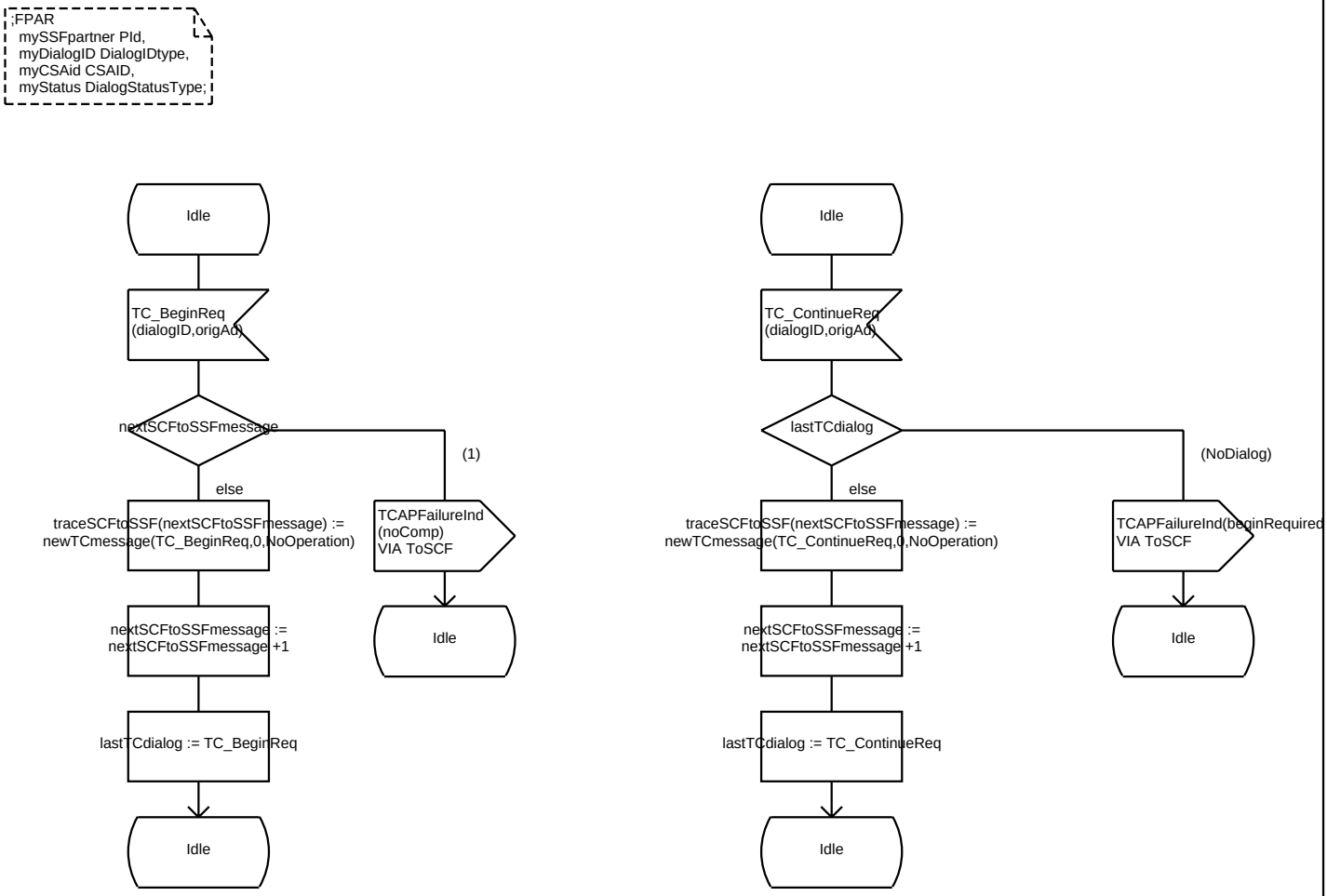


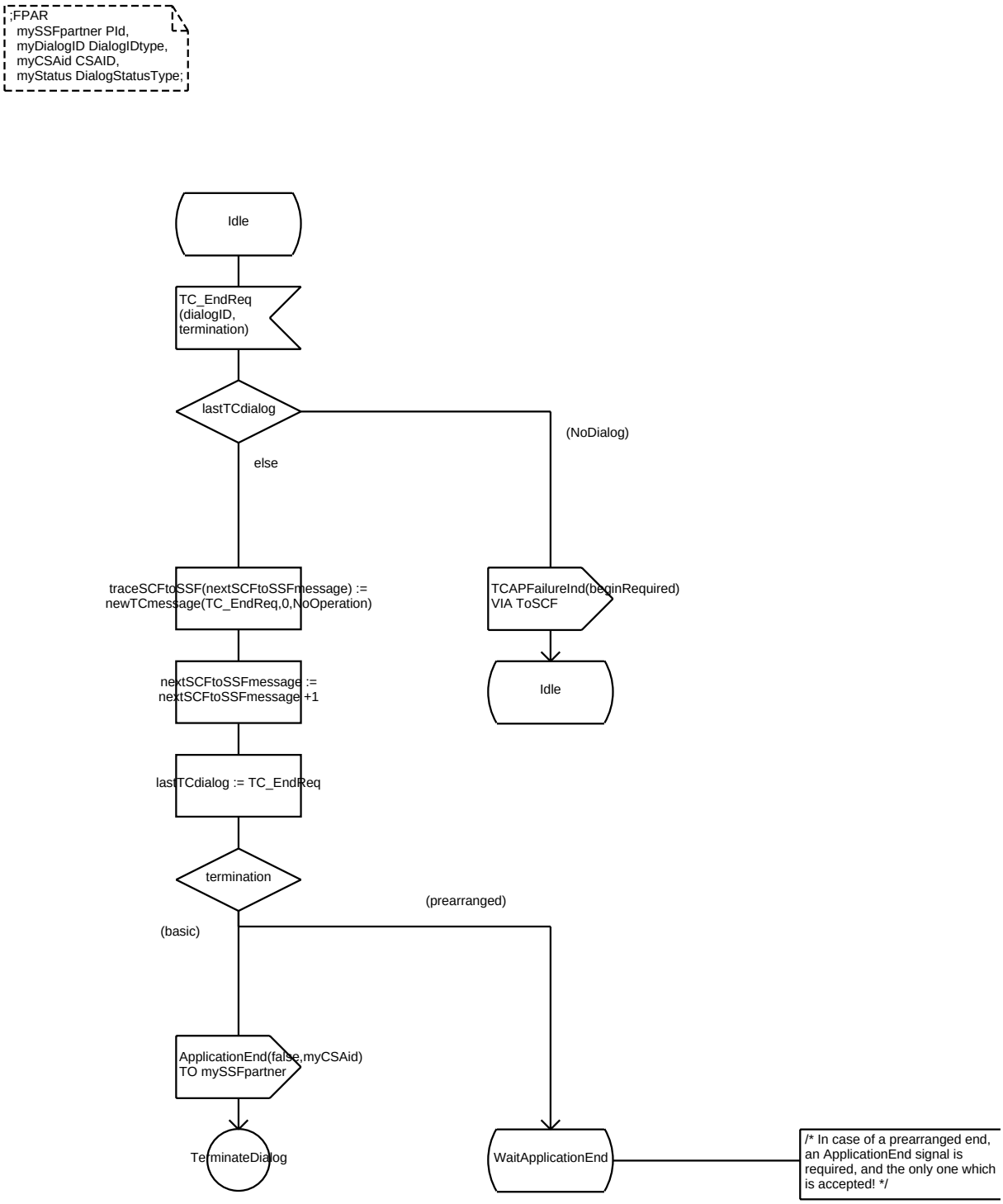

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
Idle
virtual
TC_InvokeReq
(invokeID,dialogID,
opClass,opCode,timeoutVal,arg)
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_InvokeReq,invokeID,opCode)
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
opCode
(ASF)
(AC, CIRQ, CAN,
CI, CON, CTR,
DFC, ETC,
FCI, ICA,
RNC, RRB, RT,
SCI)
(AT)
(CG,
CUE,
RC)
else
TCAPFailureInd(unknownOP)
/* Unknown op code */
class1
class2
class3
class4
Idle
class1
ActivateServiceFiltering(invokeID,arg!aSFArg)
TO mySSFpartner
Idle
class3
ActivityTest(invokeID,dialogID)
TO mySSFpartner
Idle

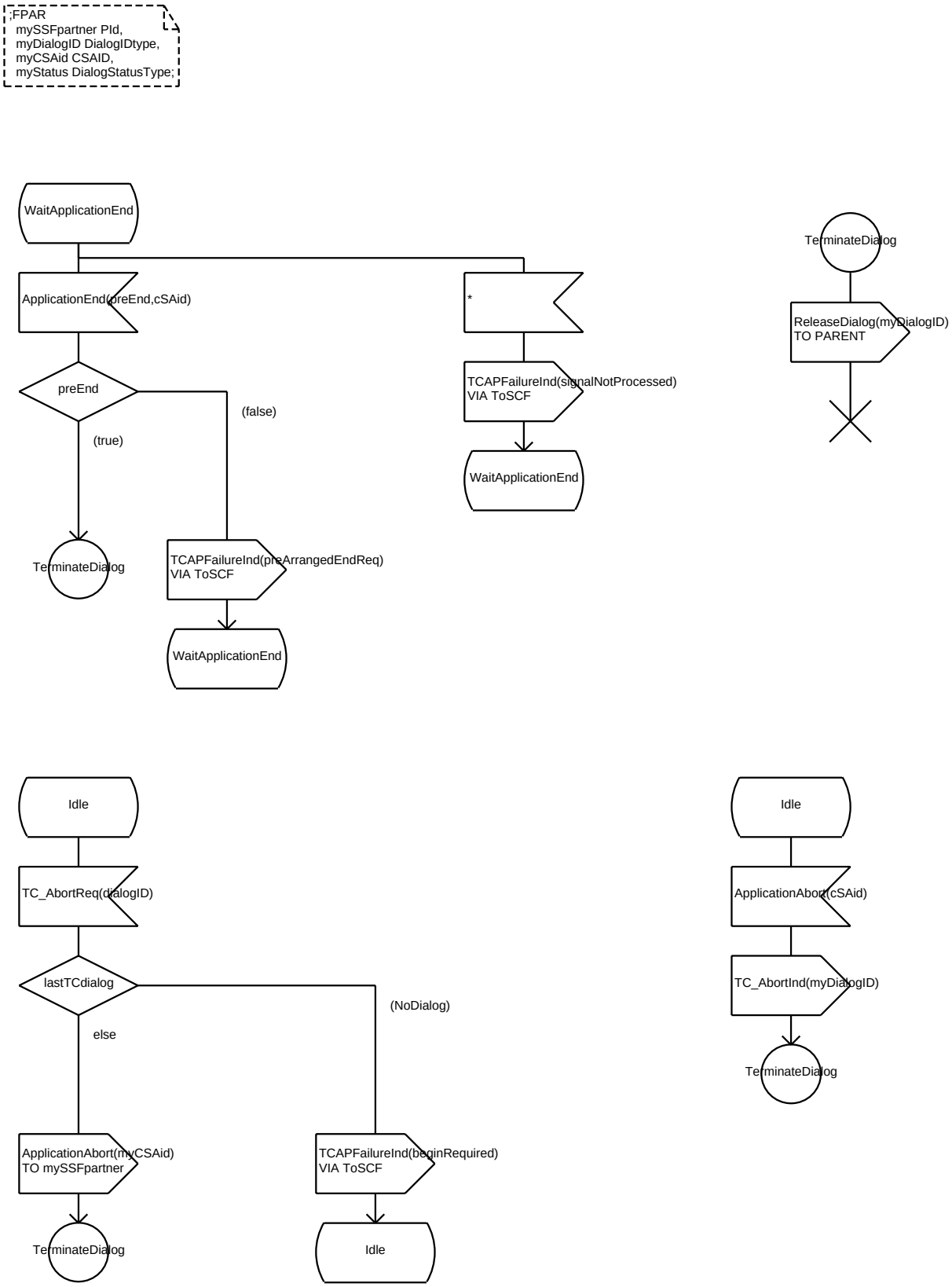

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
class2
opCode
(AC)
(CAN)
ApplyCharging(invokeID,myCSAid,arg!aCArg)
TO mySSFpartner
Cancel(invokeID,myCSAid,arg!cANArg)
TO mySSFpartner
Idle
Idle
(CI)
(CIRQ)
CollectInformation(invokeID,myCSAid,arg!cIArg)
TO mySSFpartner
CallInformationRequest(invokeID,myCSAid,arg!cIRQArg)
TO mySSFpartner
Idle
Idle
(CON)
(CTR)
Connnect(invokeID,myCSAid,arg!cONArg)
TO mySSFpartner
ConnectToResource(invokeID,myCSAid,arg!cTRArg)
TO mySSFpartner
Idle
Idle
(DFC)
(ETC)
DisconnectForwardConnection(invokeID,myCSAid)
TO mySSFpartner
EstablishTemporaryConnection(invokeID,myCSAid,arg!eTCArg)
TO mySSFpartner
Idle
Idle
(FCI)
(ICA)
FurnishChargingInformation(invokeID,myCSAid,arg!fCIArg)
TO mySSFpartner
InitiateCallAttempt(invokeID,myCSAid,arg!iCAArg)
TO mySSFpartner
Idle
Idle
(RNC)
(RRB)
RequestNotificationChargingEvent(invokeID,myCSAid,arg!rNCArg)
TO mySSFpartner
RequestReportBCSMEvent(invokeID,myCSAid,arg!rRBArg)
TO mySSFpartner
else
Idle
class2Continue
Idle

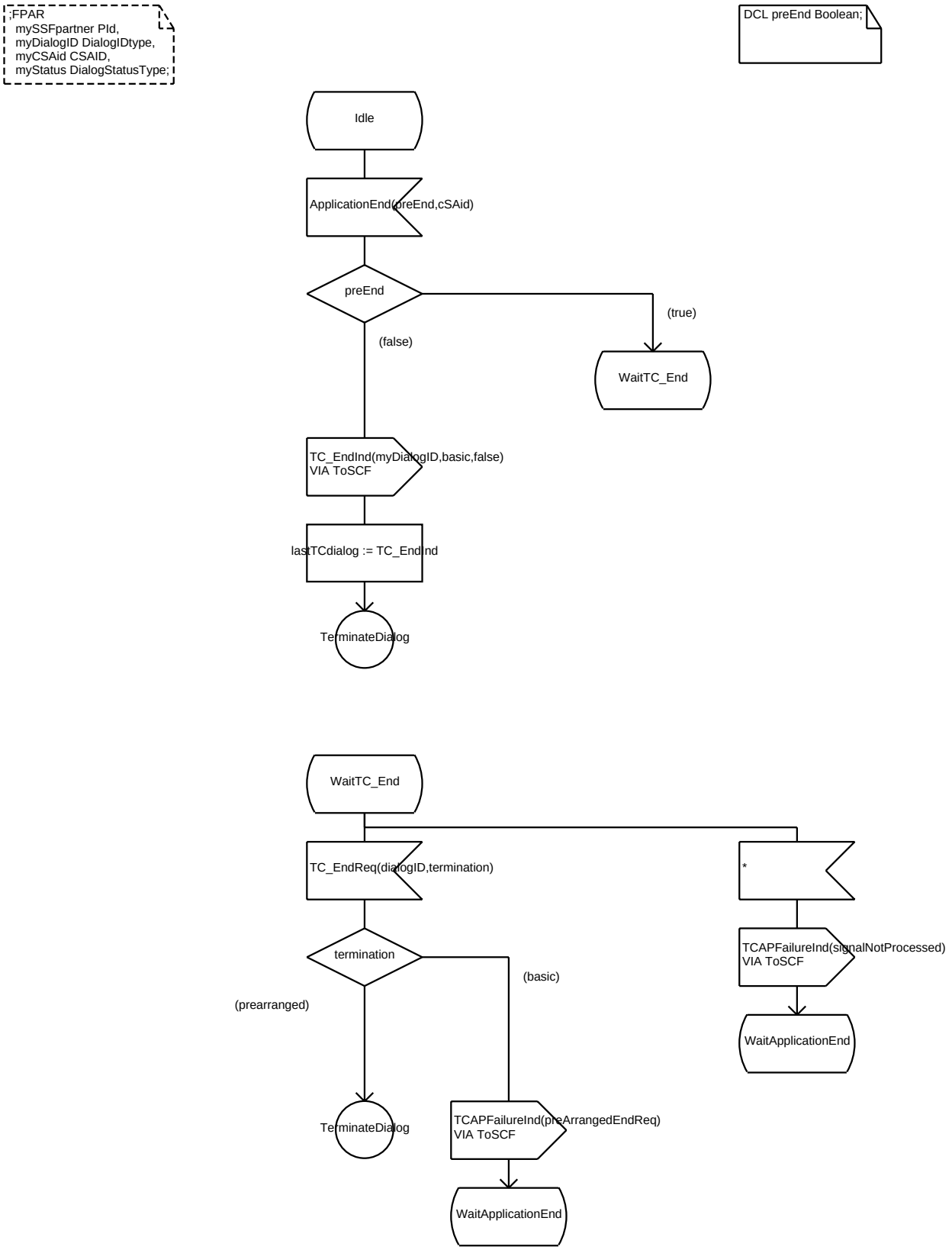

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
class2Continue
opCode
(SCI)
(RT)
SendChargingInformation(invokeID,myCSAid,arg!sCIArg)
TO mySSFpartner
ResetTimer(invokeID,myCSAid,arg!rTArg)
TO mySSFpartner
Idle
Idle

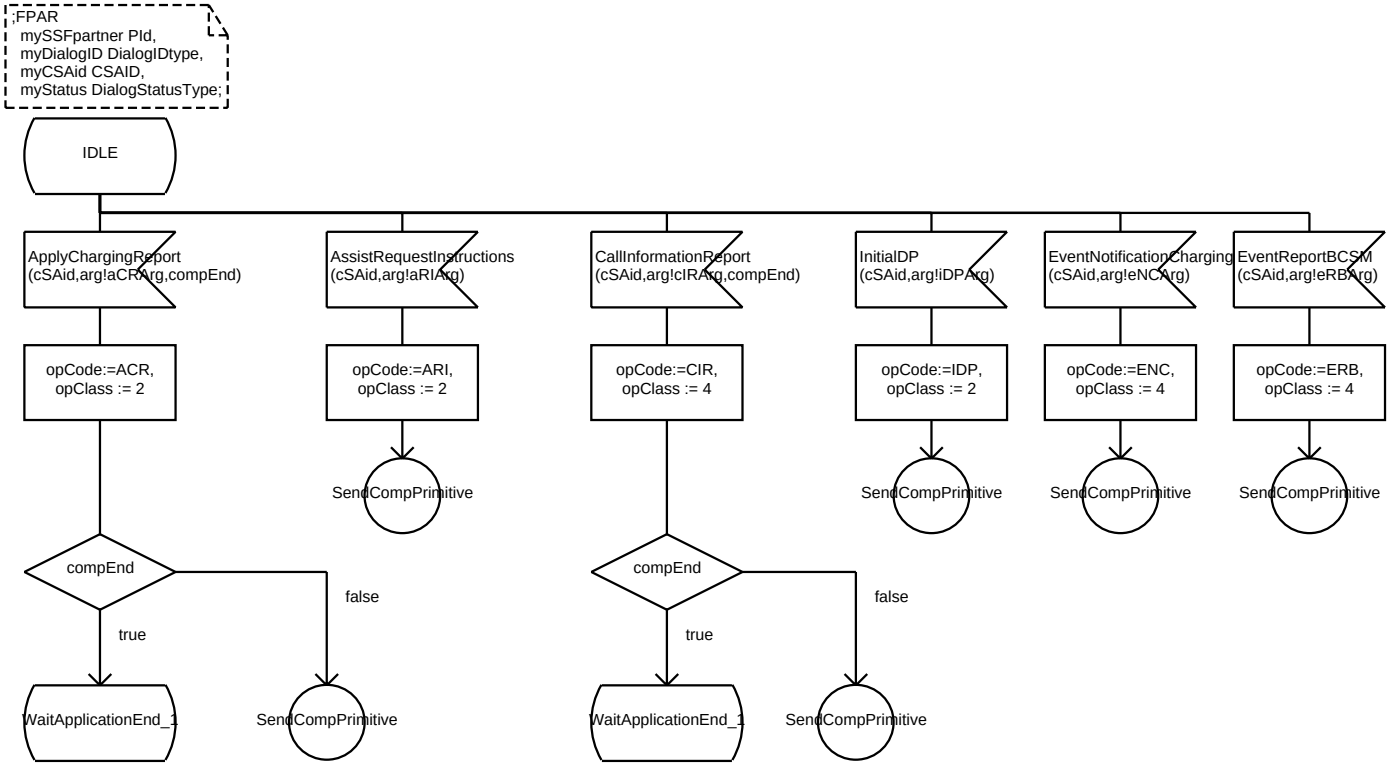

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
class4
opCode
(CG)
(CUE)
CallGap(invokeID,arg!cGArg)
TO mySSFpartner
Continue(invokeID,myCSAid)
TO mySSFpartner
Idle
Idle
(RC)
ReleaseCall(invokeID,myCSAid,arg!rCArg)
TO mySSFpartner
WaitDialog
WaitDialog
TC_EndReq(dialogID,termination)
ApplicationEnd
*
SecondWaitApplicationEnd
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
SecondWaitApplicationEnd
WaitApplicationEnd
ApplicationEnd(preEnd,cSAid)
TerminateDialog

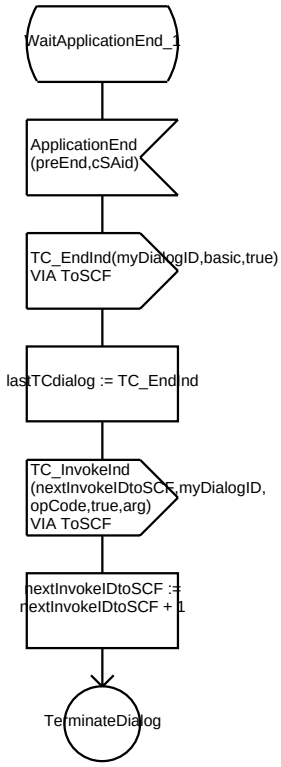

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
Idle
TC_BeginReq
(dialogID,origAd)
nextSCFtoSSFmessage
(1)
else
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_BeginReq,0,NoOperation)
TCAPFailureInd
(noComp)
VIA ToSCF
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
Idle
lastTCdialog := TC_BeginReq
Idle
Idle
TC_ContinueReq
(dialogID,origAd)
lastTCdialog
(NoDialog)
else
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_ContinueReq,0,NoOperation)
TCAPFailureInd(beginRequired
VIA ToSCF
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
Idle
lastTCdialog := TC_ContinueReq
Idle

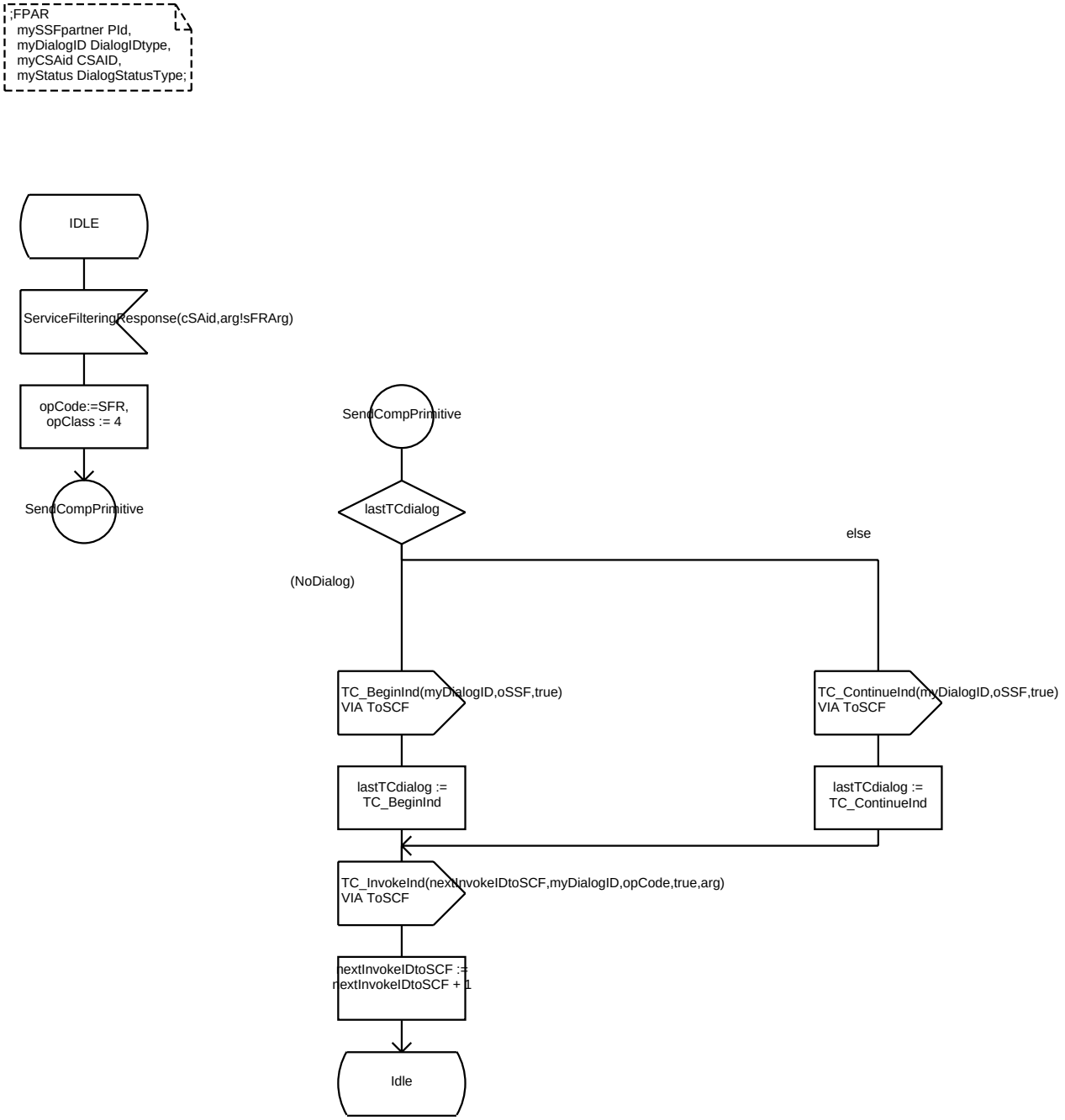

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
Idle
TC_EndReq
(dialogID,
termination)
lastTCdialog
(NoDialog)
else
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_EndReq,0,NoOperation)
TCAPFailureInd(beginRequired)
VIA ToSCF
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
Idle
lastTCdialog := TC_EndReq
termination
(prearranged)
(basic)
ApplicationEnd(false,myCSAid)
TO mySSFpartner
TerminateDialog
WaitApplicationEnd
/* In case of a prearranged end,
an ApplicationEnd signal is
required, and the only one which
is accepted! */

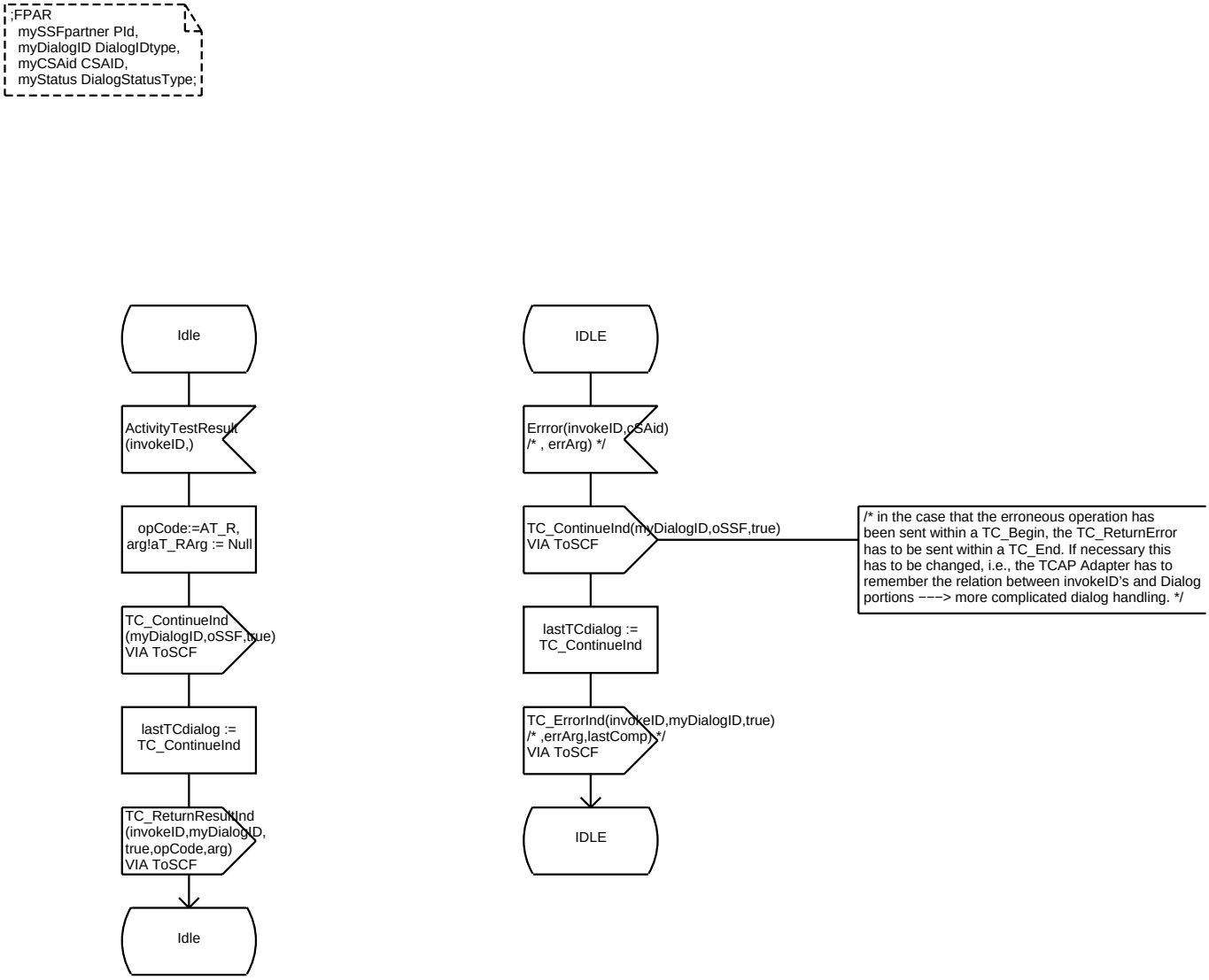

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
WaitApplicationEnd
ApplicationEnd(preEnd,cSAid)
*
preEnd
(false)
(true)
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
WaitApplicationEnd
TerminateDialog
TCAPFailureInd(preArrangedEndReq)
VIA ToSCF
WaitApplicationEnd
TerminateDialog
ReleaseDialog(myDialogID)
TO PARENT
Idle
TC_AbortReq(dialogID)
lastTCdialog
(NoDialog)
else
ApplicationAbort(myCSAid)
TO mySSFpartner
TerminateDialog
TCAPFailureInd(beginRequired)
VIA ToSCF
Idle
Idle
ApplicationAbort(cSAid)
TC_AbortInd(myDialogID)
TerminateDialog

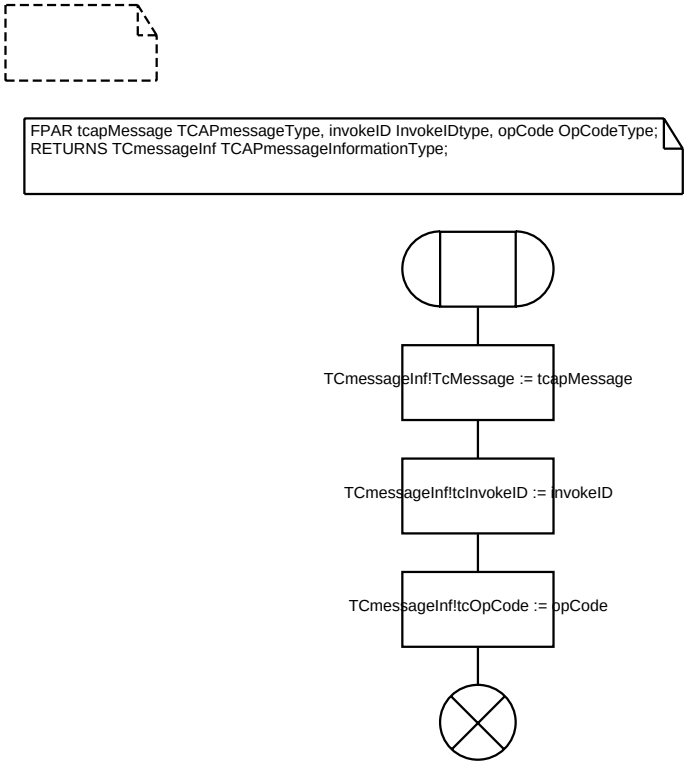

;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
DCL preEnd Boolean;
Idle
ApplicationEnd(preEnd,cSAid)
preEnd
(true)
(false)
WaitTC_End
TC_EndInd(myDialogID,basic,false)
VIA ToSCF
lastTCdialog := TC_EndInd
TerminateDialog
WaitTC_End
TC_EndReq(dialogID,termination)
*
termination
(basic)
(prearranged)
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
WaitApplicationEnd
TerminateDialog
TCAPFailureInd(preArrangedEndReq)
VIA ToSCF
WaitApplicationEnd


;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
IDLE
ApplyChargingReport
(cSAid,arg!aCRArg,compEnd)
AssistRequestInstructions
(cSAid,arg!aRIArg)
CallInformationReport
(cSAid,arg!cIRArg,compEnd)
InitialDP
(cSAid,arg!iDPArg)
EventNotificationCharging
(cSAid,arg!eNCArg)
EventReportBCSM
(cSAid,arg!eRBArg)
opCode:=ACR,
opClass := 2
opCode:=ARI,
opClass := 2
opCode:=CIR,
opClass := 4
opCode:=IDP,
opClass := 2
opCode:=ENC,
opClass := 4
opCode:=ERB,
opClass := 4
SendCompPrimitive
SendCompPrimitive
SendCompPrimitive
SendCompPrimitive
compEnd
false
true
WaitApplicationEnd_1
SendCompPrimitive
compEnd
false
true
WaitApplicationEnd_1
SendCompPrimitive


WaitApplicationEnd_1
ApplicationEnd
(preEnd,cSAid)
TC_EndInd(myDialogID,basic,true)
VIA ToSCF
lastTCdialog := TC_EndInd
TC_InvokeInd
(nextInvokeIDtoSCF,myDialogID,
opCode,true,arg)
VIA ToSCF
nextInvokeIDtoSCF :=
nextInvokeIDtoSCF + 1
TerminateDialog


;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
IDLE
ServiceFilteringResponse(cSAid,arg!sFRArg)
opCode:=SFR,
opClass := 4
SendCompPrimitive
SendCompPrimitive
lastTCdialog
else
(NoDialog)
TC_BeginInd(myDialogID,oSSF,true)
VIA ToSCF
TC_ContinueInd(myDialogID,oSSF,true)
VIA ToSCF
lastTCdialog :=
TC_BeginInd
lastTCdialog :=
TC_ContinueInd
TC_InvokeInd(nextInvokeIDtoSCF,myDialogID,opCode,true,arg)
VIA ToSCF
nextInvokeIDtoSCF :=
nextInvokeIDtoSCF + 1
Idle


;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
Idle
ActivityTestResult
(invokeID,)
opCode:=AT_R,
arg!aT_RArg := Null
TC_ContinueInd
(myDialogID,oSSF,true)
VIA ToSCF
lastTCdialog :=
TC_ContinueInd
TC_ReturnResultInd
(invokeID,myDialogID,
true,opCode,arg)
VIA ToSCF
Idle
IDLE
Errror(invokeID,cSAid)
/* , errArg) */
TC_ContinueInd(myDialogID,oSSF,true)
VIA ToSCF
/* in the case that the erroneous operation has
been sent within a TC_Begin, the TC_ReturnError
has to be sent within a TC_End. If necessary this
has to be changed, i.e., the TCAP Adapter has to
remember the relation between invokeID's and Dialog
portions −−−> more complicated dialog handling. */
lastTCdialog :=
TC_ContinueInd
TC_ErrorInd(invokeID,myDialogID,true)
/* ,errArg,lastComp) */
VIA ToSCF
IDLE


FPAR tcapMessage TCAPmessageType, invokeID InvokeIDtype, opCode OpCodeType;
RETURNS TCmessageInf TCAPmessageInformationType;
TCmessageInf!TcMessage := tcapMessage
TCmessageInf!tcInvokeID := invokeID
TCmessageInf!tcOpCode := opCode