Annex A:

Table of Contents

Organizer View 1
Text readme 17
Text changelog 18
ASN.1 Text INCS2BundleArg 19
ASN.1 Text INCS2datatypes 22
ASN.1 Text INCS2Internals 42
ASN.1 Text INCS2opsargs 46
ASN.1 Text INCS2SCFSRFopsargs 63
Package CS1_INAP 66
System Type CS1_INAP 67
Block Type SSF_CCF 75
Process Type InterfaceHandler 81
Procedure AllocateCSAId 101
Procedure AddCSAId 102
Procedure GetCSAIdfromCSA 103
Procedure GetCSAfromCSAId 104
Procedure GetCSAfromSigConId 105
Procedure SetSigConAssoc 106
Procedure SetRemoteAssoc 107
Procedure GetCSAfromRemoteId 108
Procedure IsCSA 109
Process Type CallSegmentAssociation 110
Procedure AddCS 130
Procedure SetLegLocation 131
Procedure GetCSPtr 132
Procedure GetLegLocation 133
Procedure ExistCS 134
Procedure SetLegAssoc 135
Procedure IsCS 136
Procedure GetLegIdfromRemoteLegId 137
Procedure ExistLeg 138
Process Type CallSegment 139
Procedure AddLeg 166
Procedure RemoveLeg 167
Procedure SetLegStatus 168
Procedure ReleaseAllLegs 169
Procedure GetLegStatus 170
Procedure SetLegAssoc 171
Procedure SetLegPtr 172
Procedure GetLegIdfromRemoteLegId 173
Procedure GetLegPtr 174
Procedure IsBCSM 175
Procedure GetPassiveLegId 176
Procedure MapConnectToBCSM 177
Procedure MapSIToBCSM 178
Process Type SSF_FSM 179
Procedure InitialiseDPTable 204
Procedure ExistLeg 205
Procedure AnyDPArmed 206
Procedure IsDPArmed 207
Procedure DisarmDPs 208
Procedure DPArmed 209
Procedure CallInformationReportPending 210
Procedure ApplyChargingReportPending 211
Procedure ArmTDPs 212
Procedure MatchingServiceFilteringCriteria 213
Procedure IncrementOutstandingRequests 214
Procedure CheckACG 215
Procedure OustandingRequests 216
Procedure CallFiltered 217
Procedure ProcessApplyCharging 218
Procedure ProcessContinue 219

Annex A:

Procedure ProcessRequestReportBCSMEvent 220
Procedure ProcessCallInformationRequest 221
Procedure ProcessDisconnectForwardConnection 222
Procedure ProcessSendChargingInformation 223
Procedure ProcessCancel 224
Procedure ProcessEstablishTemporaryConnection 225
Procedure ProcessEventReportBCSM 226
Procedure ProcessCollectInformation 227
Procedure ProcessFurnishChargingInformation 228
Procedure ProcessForwardConnectionReleased 229
Procedure ProcessConnect 230
Procedure ProcessInitiateCallAttempt 232
Procedure ProcessInitialDP 233
Procedure ProcessConnectToResource 235
Procedure ProcessRequestNotificationChargingEvent 236
Procedure ConnnectAnalysis 237
Process Type OriginatingBCSM 238
Procedure PIC_O_Null 249
Procedure PIC_Analyse_Information 250
Procedure PIC_Send_Call 255
Procedure PIC_O_Active 259
Procedure PIC_Authorise_Origination_Attempt 260
Procedure PIC_Select_Route 261
Procedure PIC_O_Alerting 262
Procedure PIC_O_Abandon 266
Procedure PIC_Collect_Information 267
Procedure PIC_Authorise_Call_Setup 270
Procedure PIC_O_Answer 271
Procedure PIC_OException 272
Procedure PIC_Collect_NDigits 273
Procedure DP_origAttemptAuthorised 276
Procedure DP_Route_Select_Failure 279
Procedure DP_oCalledPartyBusy 281
Procedure DP_oDisconnect 282
Procedure DP_Collected_Info 285
Procedure DP_oAnswer 288
Procedure DP_oMidCall 289
Procedure DP_Analysed_Information 290
Procedure DP_oNoAnswer 292
Procedure DP_oAbandon 293
Procedure MapToSIRArg 294
Procedure MapToDP 295
Procedure MapFromPIC 296
Process Type TerminatingBCSM 297
Procedure PIC_T_Null 305
Procedure PIC_Select_Facility 306
Procedure PIC_tActive 307
Procedure PIC_Authorize_Termination_Attempt 308
Procedure PIC_Present_Call 309
Procedure PIC_TException 311
Procedure PIC_tAlerting 312
Procedure DP_termAttemptAuthorized 313
Procedure DP_tNoAnswer 314
Procedure DP_tDisconnect 315
Procedure DP_tBusy 318
Procedure DP_tMidCall 319
Procedure DP_tAnswer 320
Procedure DP_tAbandon 321
Process Type SSME_FSM 322
Procedure ProcessActivateServiceFiltering 326
Procedure ProcessCallGap 327
Procedure InitialiseTDPTable 328
Procedure Mgt_SetTriggerTable 329
Procedure ArmTDPs 330
Procedure MatchingServiceFilteringCriteria 331

Annex A:

Procedure CheckACG 332
Procedure CallFiltered 333
Block Type TCAP_Simulator 334
Process Type TCAP_InterfaceHandler 336
Operator newDialog 348
Operator initDialogs 349
Operator getDialogID 350
Operator nextFreeDialogID 351
Process Type TCAP_Dialog 352
Operator newTCmessage 366
Package CS2_INAP 367
System Type CS2_INAP 368
Block Type SSF_CCF 373
Process Type InterfaceHandler 377
Procedure ProcessMoveCallSegments 388
Process Type CallSegmentAssociation 389
Procedure NoOfSegments 409
Procedure ExportLeg 410
Procedure ExportCS 411
Procedure ImportLeg 412
Procedure ImportCS 413
Procedure RemoveLeg 414
Procedure RemoveCS 415
Procedure MoveLegs 416
Procedure ExportControllingLeg 417
Procedure ProcessRRBE 418
Procedure DisconnectControllingLeg 419
Procedure ProcessQueuedRRBE 420
Procedure ReleaseCall 421
Process Type CallSegment 423
Procedure ImportLeg 471
Procedure BroadcastToLegs 472
Procedure ExportLeg 474
Procedure NoOfLegs 475
Procedure DPFiltering 476
Procedure DPFilteringUTSI 480
Process Type SSF_FSM 481
Procedure ExportEventRecord 497
Procedure ProcessContinueWithArgument 498
Procedure ProcessRequestReportUTSI 500
Procedure ImportEventRecord 501
Procedure ProcessDisconnectLeg 502
Procedure ProcessSendSTUI 503
Procedure ProcessReportUTSI 504
Procedure IsUTSIArmed 505
Procedure InitialiseDPTable 506
Procedure IncrementRequests 507
Process Type SSME_FSM 508
Procedure ProcessManageTriggerData 509
Procedure CheckACG 510
Procedure InitialiseTDPTable 511
Process Type OriginatingBCSM 512
Procedure PIC_O_Active 522
Procedure PIC_Send_Call 524
Procedure PIC_O_Suspended 528
Procedure PIC_O_Retention 531
Procedure PIC_O_Alerting 533
Procedure PIC_Disconnect 536
Procedure DP_origAttemptAuthorised 537
Procedure DP_oAnswer 539
Procedure DP_oAbandon 541
Procedure DP_origAttempt 542
Procedure DP_Collected_Info 544
Procedure DP_oNoAnswer 546
Procedure DP_oDisconnect 548

Annex A:

Procedure DP_oSuspended 551
Procedure DP_Analysed_Information 553
Procedure DP_oCalledPartyBusy 555
Procedure DP_oReanswer 557
Procedure DP_Route_Select_Failure 559
Procedure DP_oMidCall 561
Procedure DP_oTermSeized 564
Process Type TerminatingBCSM 566
Procedure PIC_Present_Call 575
Procedure PIC_Disconnect 577
Procedure PIC_tAlerting 578
Procedure PIC_T_Suspended 580
Procedure PIC_tActive 583
Procedure PIC_Select_Facility 585
Procedure DP_termAttemptAuthorized 586
Procedure DP_tMidCall 588
Procedure DP_termAttempt 590
Procedure DP_tReanswer 593
Procedure DP_tBusy 596
Procedure DP_tAbandon 598
Procedure DP_Facility_Selected_and_Available 599
Procedure DP_tAnswer 602
Procedure DP_tDisconnect 604
Procedure DP_tSuspended 609
Procedure DP_tNoAnswer 612
Procedure DP_tCall_Accepted 615
Block Type TCAP_Simulator 617
Process Type TCAP_InterfaceHandler 618
Process Type TCAP_Dialog 621
System CS1_INAP 628
System CS2_INAP 629
MSC CI_Overview 630
MSC CI_Detailed 632
MSC CO_Overview 648
MSC CO_Detailed 651
MSC Process_Interworking 667
MSC 3Pty_Overview 669
MSC 3Pty_Detailed 672
MSC CF_Overview 691
MSC CF_Detailed 693
MSC OutGoingCall 707

Annex A: View

SDT	System file	rw	/home/mulligan/INAP/CPH_CoreINAP/inap.sdt
	Source directory	rw	/home/mulligan/INAP/CPH_CoreINAP/
	Text readme	rw	readme.txt
	Text changelog	rw	changelog.txt
	ASN.1 Files		
ASN	ASN.1 Text INCS2BundleArg	rw	ASN1/incs2b.asn
ASN	ASN.1 Text INCS2datatypes	rw	ASN1/incs2−dt.asn
ASN	ASN.1 Text INCS2Internals	rw	ASN1/cs2i.asn
ASN	ASN.1 Text INCS2opsargs	rw	ASN1/ssf−scf.asn
ASN	ASN.1 Text INCS2SCFSRFopsargs	rw	ASN1/ssf−srf.asn
	IN CS−1 Specification		
	Package CS1_INAP	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 INCS2opsargs		
	System Type CS1_INAP	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	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	rw	CS1/ih.spt
	Procedure AllocateCSAId	rw	CS1/alldid.spd
	Procedure AddCSAId	rw	CS1/adddid.spd

Annex A: View

- Procedure GetCSAIdfromCSA rw CS1/getdid.spd
- Procedure GetCSAfromCSAId rw CS1/getcvd.spd
- Procedure GetCSAfromSigConId rw CS1/getcvu.spd
- Procedure SetSigConAssoc rw CS1/setsca.spd
- Procedure SetRemoteAssoc rw CS1/setrema.spd
- Procedure GetCSAfromRemoteId rw CS1/getlcsa.spd
- Procedure IsCSA rw CS1/iscsa.spd

Virtual Process Type CallSegmentAssociation rw CS1/csa.spt

- Procedure AddCS rw CS1/addcs.spd
- Procedure SetLegLocation rw CS1/setlegl.spd
- Procedure GetCSPtr rw CS1/getcsptr.spd
- Procedure GetLegLocation rw CS1/getlegl.spd
- Procedure ExistCS rw CS1/existcs.spd
- Procedure SetLegAssoc rw CS1/setlassc.spd
- Procedure IsCS rw CS1/iscs.spd
- Procedure GetLegIdfromRemoteLegId rw CS1/getlegrc.spd
- Procedure ExistLeg rw CS1/existleg.spd

Virtual Process Type CallSegment rw CS1/cv1.spt

- Procedure AddLeg rw CS1/addleg.spd
- Procedure RemoveLeg rw CS1/remleg.spd
- Procedure SetLegStatus rw CS1/setlstat.spd
- Procedure ReleaseAllLegs rw CS1/releasea.spd
- Procedure GetLegStatus rw CS1/getlstat.spd
- Procedure SetLegAssoc rw CS1/setlass.spd
- Procedure SetLegPtr rw CS1/setlptr.spd
- Procedure GetLegIdfromRemoteLegId rw CS1/getlegr.spd
- Procedure GetLegPtr rw CS1/getlptr.spd
- Procedure IsBCSM rw CS1/isbcsm.spd
- Procedure GetPassiveLegId rw CS1/getpl.spd
- Procedure MapConnectToBCSM rw CS1/mpcobcsm.spd
- Procedure MapSIToBCSM rw CS1/mpsibcsm.spd

Virtual Process Type SSF_FSM rw CS1/ssf_fsm.spt

- ***Virtual*** Procedure InitialiseDPTable rw CS1/initevl.spd
- Procedure ExistLeg rw CS1/exleg.spd
- Procedure AnyDPArmed rw CS1/anyevarm.spd
- Procedure IsDPArmed rw CS1/evarm.spd
- Procedure DisarmDPs rw CS1/disarm.spd
- Procedure DPArmed rw CS1/dparmed.spd

Annex A: View

- Procedure CallInformationReportPending rw CS1/callinfo.spd
- Procedure ApplyChargingReportPending rw CS1/applycha.spd
- Procedure ArmTDPs rw CS1/armtdps.spd
- Procedure MatchingServiceFilteringCriteria rw CS1/msfc.spd
- Procedure IncrementOutstandingRequests rw CS1/ior.spd
- Procedure CheckACG rw CS1/cacg.spd
- Procedure OustandingRequests rw CS1/or.spd
- Procedure CallFiltered rw CS1/cf.spd
- Procedure ProcessApplyCharging rw CS1/process6.spd
- Procedure ProcessContinue rw CS1/proces12.spd
- Procedure ProcessRequestReportBCSMEvent rw CS1/prreqrep.spd
- Procedure ProcessCallInformationRequest rw CS1/process7.spd
- Procedure ProcessDisconnectForwardConnection rw CS1/processd.spd
- Procedure ProcessSendChargingInformation rw CS1/procsci.spd
- Procedure ProcessCancel rw CS1/process8.spd
- Procedure ProcessEstablishTemporaryConnection rw CS1/processe.spd
- Procedure ProcessEventReportBCSM rw CS1/prevrep.spd
- Procedure ProcessCollectInformation rw CS1/process9.spd
- Procedure ProcessFurnishChargingInformation rw CS1/proces13.spd
- Procedure ProcessForwardConnectionReleased rw CS1/processf.spd
- Procedure ProcessConnect rw CS1/proces10.spd
- Procedure ProcessInitiateCallAttempt rw CS1/processi.spd
- Procedure ProcessInitialDP rw CS1/pridp.spd
- Procedure ProcessConnectToResource rw CS1/proces11.spd
- Procedure ProcessRequestNotificationChargingEvent rw CS1/proces16.spd
- Procedure ConnnectAnalysis rw CS1/ERROR/conpa.spd

Virtual Process Type OriginatingBCSM rw CS1/ocs1.spt

- Procedure PIC_O_Null rw CS1/pic_o_nu.spd
- Procedure PIC_Analyse_Information rw CS1/pic_anal.spd
- ***Virtual*** Procedure PIC_Send_Call rw CS1/pic_send.spd
- ***Virtual*** Procedure PIC_O_Active rw CS1/pic_o_ac.spd
- Procedure PIC_Authorise_Origination_Attempt rw CS1/pic_auth.spd
- Procedure PIC_Select_Route rw CS1/pic_sele.spd
- ***Virtual*** Procedure PIC_O_Alerting rw CS1/pic_o_al.spd
- Procedure PIC_O_Abandon rw CS1/pic_o_ab.spd
- Procedure PIC_Collect_Information rw CS1/pic_coll.spd
- Procedure PIC_Authorise_Call_Setup rw CS1/pic_aut1.spd
- Procedure PIC_O_Answer rw CS1/pic_o_an.spd
- Procedure PIC_OException rw CS1/pic_oexc.spd

Annex A: View

- Procedure PIC_OException rw CS1/pic_oexc.spd
- Procedure PIC_Collect_NDigits rw CS1/pic_col1.spd
- ***Virtual*** Procedure DP_origAttemptAuthorised rw CS1/dp_origa.spd
- ***Virtual*** Procedure DP_Route_Select_Failure rw CS1/dp_route.spd
- ***Virtual*** Procedure DP_oCalledPartyBusy rw CS1/dp_ocall.spd
- ***Virtual*** Procedure DP_oDisconnect rw CS1/dpdiscon.spd
- ***Virtual*** Procedure DP_Collected_Info rw CS1/dp_colle.spd
- ***Virtual*** Procedure DP_oAnswer rw CS1/dp_o_ans.spd
- ***Virtual*** Procedure DP_oMidCall rw CS1/dp_omidc.spd
- ***Virtual*** Procedure DP_Analysed_Information rw CS1/dp_analy.spd
- ***Virtual*** Procedure DP_oNoAnswer rw CS1/dp_onoan.spd
- ***Virtual*** Procedure DP_oAbandon rw CS1/dp_oaban.spd
- Procedure MapToSIRArg rw CS1/maptosir.spd
- Procedure MapToDP rw CS1/maptodp.spd
- Procedure MapFromPIC rw CS1/mapfpic.spd
- ***Virtual*** Process Type TerminatingBCSM rw CS1/tcs1.spt
  - Procedure PIC_T_Null rw CS1/pic_t_nu.spd
  - ***Virtual*** Procedure PIC_Select_Facility rw CS1/pic_sel1.spd
  - ***Virtual*** Procedure PIC_tActive rw CS1/pic_tact.spd
  - Procedure PIC_Authorize_Termination_Attempt rw CS1/pic_aut2.spd
  - ***Virtual*** Procedure PIC_Present_Call rw CS1/pic_pres.spd
  - Procedure PIC_TException rw CS1/pic_texc.spd
  - ***Virtual*** Procedure PIC_tAlerting rw CS1/pic_tale.spd
  - ***Virtual*** Procedure DP_termAttemptAuthorized rw CS1/dp_terma.spd
  - ***Virtual*** Procedure DP_tNoAnswer rw CS1/dp_tnoan.spd
  - ***Virtual*** Procedure DP_tDisconnect rw CS1/dp_disco.spd
  - ***Virtual*** Procedure DP_tBusy rw CS1/dp_tbusy.spd
  - ***Virtual*** Procedure DP_tMidCall rw CS1/dp_tmidc.spd
  - ***Virtual*** Procedure DP_tAnswer rw CS1/dp_tansw.spd
  - ***Virtual*** Procedure DP_tAbandon rw CS1/dp_taban.spd
- ***Virtual*** Process Type SSME_FSM rw CS1/ssme_fsm.spt
  - Procedure ProcessActivateServiceFiltering rw CS1/procasf.spd
  - Procedure ProcessCallGap rw CS1/proccg.spd
  - ***Virtual*** Procedure InitialiseTDPTable rw CS1/inittdps.spd
  - Procedure Mgt_SetTriggerTable rw CS1/mgt_stt.spd
  - Procedure ArmTDPs rw CS1/armstati.spd
  - Procedure MatchingServiceFilteringCriteria rw CS1/matching.spd
  - ***Virtual*** Procedure CheckACG rw CS1/checkacg.spd
  - Procedure CallFiltered rw CS1/callfilt.spd

Annex A: View

- ***Virtual*** Block Type TCAP_Simulator rw TCAP/tcaps.sbt
  - x:y Process Instance TCAP_D (0) : TCAP_Dialog
  - x:y Process Instance TCAP_IH (1,1) : TCAP_InterfaceHandler
  - ***Virtual*** Process Type TCAP_InterfaceHandler rw TCAP/tcapi.spt
    - Operator newDialog rw TCAP/newd.sop
    - Operator initDialogs rw TCAP/initd.sop
    - Operator getDialogID rw TCAP/getd.sop
    - Operator nextFreeDialogID rw TCAP/nextfd.sop
  - ***Virtual*** Process Type TCAP_Dialog rw TCAP/tcapd.spt
    - Operator newTCmessage rw TCAP/newtcm.sop

IN CS−2 Specification

- Package CS2_INAP rw CS2/cs2.sun
  - 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

Annex A: View

- 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 DPFilteringUTSI rw CS2/dpfilter.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
  - Procedure IsUTSIArmed rw CS2/isutsiar.spd
  - ***Redefined*** Procedure InitialiseDPTable rw CS2/initiali.spd
  - Procedure IncrementRequests rw CS2/ir.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_origAttempt rw CS2/dp_orig1.spd
  - ***Redefined*** Procedure DP_Collected_Info rw CS2/dp_colle.spd
  - ***Redefined*** Procedure DP_oNoAnswer rw CS2/dp_onoan.spd

Annex A: View

- ***Redefined*** Procedure DP_oDisconnect rw CS2/dp_disco.spd
- Procedure DP_oSuspended rw CS2/dp_osusp.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_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 PIC_Select_Facility rw CS2/pic_sf.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
  - ***Redefined*** Process Type TCAP_InterfaceHandler rw CS2/tcapih.spt
  - ***Redefined*** Process Type TCAP_Dialog rw CS2/tcapdia.spt

System Instantiations

System CS1_INAP rw CS1/cs1.ssy

System CS2_INAP rw CS2/cs2_inap.ssy

CS−1 Examples

Annex A: View

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
CPH Examples		
MSC Process_Interworking	rw	EXAMPLES/callsetupdetailed.msc
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
CTM Examples		
MSC OutGoingCall	rw	EXAMPLES/CTMOut.msc

Annex A: View

Annex A: View

Annex A: View

Annex A: View

Annex A: View

Annex A: View

Annex A: View

Annex A: View

Annex A: readme

EN 301 140−1−1 Annex A
SDL Specification of the ETSI Core INAP CS−2

ETSI Secretariat
F−06921 Sophia Antipolis Cedex
France
Tel: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
secretariat@etsi.fr http:\\www.etsi.fr

Copyright Notification
==========
No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in
all media.

Note:
This SDL specificaiton forms the normative behaviour of the
ETSI Core INAP CS−2. An introduction to the modelling approach
can be found in Annex A, section 1 of EN 301 140−1.

Version December 18 1998 (version for Vote).

Any further information may be obtained from:
Ultan Mulligan
ETSI PEX Group
F−06921 Sophia Antipolis Cedex
France
Tel: +33 4 92 94 43 88 Fax: +33 4 93 65 38 51
ultan.mulligan@etsi.fr or pex@etsi.fr

This SDL specification is best viewed with the font settings
Arial 8 for Headings and Arial 6 for symbol texts.

Annex A: changelog

Changes to CoreINAP CS−2 SDL model as a result of PE resolution meeting of EN 301 140−1

DP_oNotReachable removed from O_BCSM for CS−2
DP_tNotReachable removed from T_BCSM for CS−2
References to these DPs removed from ASN.1 files, InterfaceHandler and elsewhere
Terminating 1−Party Setup Removed from CallSegment CS−2
CallSegment CS−2: DisconnectLeg in Transfer no longer included with Stable_Multi_Party and On_Hold. Redrawn on next page, DL(c) now error, Disconnect_Leg now defined for Stable_1_Party (leads to Null on BCSMStop)
CallSegment CS−2: Disconnect_Leg added for Forward
CallSegment CS−2: ReleaseReqInd in Transfer redefined
Call Segment: ReleaseReqInd received in Transfer now causes fallback to Forward.
Connect allowed in Forward − causes termination and reinstantiation of BCSM for pending leg
O_ and T_BCSMs and procedures: handling of termination of BCSM added using BCSMStop message from CS.
Call Segment CS−2: Connect operation corrected − corrected start state of O_BCSM created as result of Connect. Connect in On_Hold now starts new O_BCSM.
O_BCSM CS−2: start redefined to allow correct start state
CallSegment CS−2: Incoming ReleaseReqInd received in Stable_2_Party for T_BCSM − stay in Stable_2_Party, don’t backtrack to Terminating_Setup
CallSegment CS−2: ExportLeg in Forward now leads to removal of CS − no backtrack.

Annex A: incs2b

```
INCS2BundleArg
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
IMPORTS
    -- 29.09.1998 changed to contain SRF
    -- arguments. For that purpose INCS2SSFSCFopsargs
    -- and INCS2SCFSRFopsargs have been merged to
    -- INCS2opsargs
 CancelFailed,
 RequestedInfoError,
 SystemFailure,
 TaskRefused,
 ActivateServiceFilteringArg,
 ApplyChargingArg ,
 ApplyChargingReportArg ,
 AssistRequestInstructionsArg ,
 CallGapArg ,
 CallInformationReportArg ,
 CallInformationRequestArg ,
 CancelArg ,
 CollectInformationArg ,
 ConnectArg ,
 ConnectToResourceArg,
 ContinueWithArgumentArg ,
 CreateCallSegmentAssociationArg ,
 CreateCallSegmentAssociationResultArg ,
 DisconnectForwardConnectionWithArgumentArg ,
 DisconnectLegArg ,
 EntityReleasedArg ,
 EstablishTemporaryConnectionArg ,
 EventNotificationChargingArg ,
 EventReportBCSMArg ,
 FurnishChargingInformationArg ,
 InitialDPArg ,
 InitiateCallAttemptArg ,
 ManageTriggerDataArg ,
 ManageTriggerDataResultArg  ,
 MergeCallSegmentsArg ,
 MoveCallSegmentsArg ,
 MoveLegArg ,
 ReleaseCallArg ,
 ReportUTSIArg ,
 RequestNotificationChargingEventArg,
 RequestReportBCSMEventArg ,
 RequestReportUTSIArg ,
 ResetTimerArg ,
 SendChargingInformationArg ,
 SendSTUIArg ,
 ServiceFilteringResponseArg ,
 SplitLegArg ,

 -- from now on SRF arguments
 MessageReceivedArg ,
 PlayAnnouncementArg ,
 PromptAndCollectUserInformationArg ,
 PromptAndReceiveMessageArg ,
 ReceivedInformationArg ,
 ScriptCloseArg ,
 ScriptEventArg ,
 ScriptInformationArg ,
 ScriptRunArg ,
 SpecializedResourceReportArg

FROM INCS2opsargs
```

Annex A: incs2b

```
−− FROM INCS2SSFSCFopsargs
;

Empty ::= NULL

ArgType ::= CHOICE
 { aSFArg    [0] ActivateServiceFilteringArg,
  aSFRArg   [1] Empty,
              aTArg            [2] Empty,
              aTRArg           [3] Empty,
  aCArg   [4] ApplyChargingArg ,
  aCRArg   [5] ApplyChargingReportArg ,
  aRIArg   [6] AssistRequestInstructionsArg ,
  cGArg   [7] CallGapArg ,
  cIRArg   [8] CallInformationReportArg ,
  cIRQArg   [9] CallInformationRequestArg ,
  cANArg  [10] CancelArg ,
  cIArg  [11] CollectInformationArg ,
  cONArg  [12] ConnectArg ,
  cTRArg  [13] ConnectToResourceArg,
  cWAArg  [14] ContinueWithArgumentArg ,
  cSAArg  [15] CreateCallSegmentAssociationArg ,
  cSARArg  [16] CreateCallSegmentAssociationResultArg ,
  cUEArg          [17] Empty,
  dFCArg  [99] Empty,
  dFCWAArg [18] DisconnectForwardConnectionWithArgumentArg ,
  dLArg  [19] DisconnectLegArg ,
  dLRArg  [20] Empty,
  eRArg  [21] EntityReleasedArg ,
  eTCArg  [22] EstablishTemporaryConnectionArg ,
  eNCArg  [23] EventNotificationChargingArg ,
  eRBArg  [24] EventReportBCSMArg ,
  fCIArg  [25] FurnishChargingInformationArg ,
  iDPArg  [26] InitialDPArg ,
  iCAArg  [27] InitiateCallAttemptArg ,
  mTDArg  [28] ManageTriggerDataArg ,
  mTDRArg  [29] ManageTriggerDataResultArg,
  mCArg  [30] MergeCallSegmentsArg,
  mCRArg  [31] Empty,
  mCSArg  [32] MoveCallSegmentsArg,
  mCSRArg  [33] Empty,
  mLArg  [34] MoveLegArg,
  mLRArg  [35] Empty,
  rCArg  [36] ReleaseCallArg ,
  rUArg  [37] ReportUTSIArg ,
  rNCArg  [38] RequestNotificationChargingEventArg ,
  rRBArg  [39] RequestReportBCSMEventArg ,
  rRUArg  [40] RequestReportUTSIArg ,
  rTArg  [41] ResetTimerArg ,
  sCIArg  [42] SendChargingInformationArg ,
  sSArg  [43] SendSTUIArg ,
  sFRArg  [44] ServiceFilteringResponseArg ,
  sLArg  [45] SplitLegArg ,
  sLRArg  [46] Empty ,
   −− SRF
  pAArg  [47] PlayAnnouncementArg ,
  mRArg  [48] MessageReceivedArg ,
  pACArg  [49] PromptAndCollectUserInformationArg ,
  pARArg  [50] PromptAndReceiveMessageArg ,
  rIArg  [51] ReceivedInformationArg ,
  sCArg  [52] ScriptCloseArg ,
  sEArg  [53] ScriptEventArg ,
  sIArg  [54] ScriptInformationArg ,
  sRArg  [55] ScriptRunArg ,
  sRRArg  [56] SpecializedResourceReportArg
```

Annex A: incs2b

```
 }

ErrorType ::= CHOICE
 { cancelPar    [0] Empty,
  cancelFailedPar    [1] CancelFailed,
  chainingRefusedPar   [2] Empty,
  eTCFailedPar    [3] Empty,
  improperCallerResponsePar  [4] Empty,
  missingCustomerRecordPar  [5] Empty,
  missingParameterPar   [6] Empty,
  parameterOutOfRangePar   [7] Empty,
  requestedInfoError   [8] RequestedInfoError,
  systemFailurePar   [9] SystemFailure,
  taskRefusedPar   [10] TaskRefused,
  unavailableResourcePar  [11] Empty,
  unexpectedComponentSequencePar [12] Empty,
  unexpectedDataValuePar  [13] Empty,
  unexpectedParameterPar  [14] Empty,
  unknownLegIDPar   [15] Empty,
  unknownRecordedMessageID [16] Empty,
  unknownResource   [17] Empty,
  unknownSubscriber  [18] Empty
 }
END
```

Annex A: incs2−dt

```
INCS2datatypes {ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) cs−2(20) modules(0) in−cs2−datatypes (0) ver

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

−−IMPORTS
−−tc−Messages, classes FROM  IN−CS2−object−identifiers
−−  { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) CS−2(20) module(0) in−cs2−object−identifiers(17) versio
−− InvokeIdType
−−FROM TCAPMessages tc−Messages
−− EXTENSION,
−−  PARAMETERS−BOUND,
−−  SupportedExtensions { }
−−FROM IN−CS2−classes classes;

−− *****
−− DEFINITIONS OF TYPES WHICH ARE IMPORTED OR THE DEFINITION IS MISSING (Informative information).
NumberMatch ::= OCTET STRING
ChargingTariffInformation ::= OCTET STRING
ChargingPriceInformation ::= OCTET STRING
AddOnChargingInformation ::= OCTET STRING
FacilityGroup ::= OCTET STRING
InvokeIdType ::= INTEGER

−− End of added definitions.
−− *****

AccountNumber ::= NumericString (SIZE (1..151))
AChBillingChargingCharacteristics   ::= OCTET STRING (SIZE(minAChBillingChargingLength..maxAChBillingChargingLength))
−− The AChBillingChargingCharacteristics parameter specifies the charging related information
−− to be provided by the SSF and the conditions on which this information has to be reported
−− back to the SCF with the ApplyChargingReport operation.
−− Examples of charging related information to be provided by the SSF may be: bulk counter
−− values, costs, tariff change and time of charge, time stamps, durations, etc.
−− Examples of conditions on which the charging related information are to be reported may be:
−− threshold value reached, timer expiration, tariff change, end of connection configuration, etc
ActionIndicator  ::= ENUMERATED {
activate   (1),
deactivate  (2),
retrieve  (3)
 }
−− indicates the action to be performed by the ManageTriggerData operation (activate, deactivate
−− or retrieve the status of a TDP.

ActionPerformed  ::= ENUMERATED {
activated  (1),
deactivated  (2),
alreadyActive  (3),
alreadyInactive  (4),
isActive  (5),
isInactive  (6)
 }
−− indicates the result of the operation ManageTriggerData
−− activated: response of activate TDP
−− deactivated: response of deactivate TDP
−− alreadyActive: response of activate TDP
−− alreadyInactive: response of deactivate TDP
−− isActive: response of retrieve status of TDP
−− isInactive: response of retrieve status of TDP

ActivableServices ::= BIT STRING {
 callingLineIdentificationPresentation (1),
 callingLineIdentificationRestriction (2),
 connectedLineIdentificationPresentation (3),
 connectedLineIdentificationRestriction (4),
```

Annex A: incs2−dt

```
 callForwardingOnNoReply (5),
 callForwardingUnconditional (6),
 callForwardingOnBusy (7),
 callForwardingOnNotReachable (8),
 reverseCharging (9),
 adviceOfChargeOnStart (10),
 adviceOfChargeAtEnd (11),
 adviceOfChargeDuringCall (12),
 timeDependentRouting (13),
 callingPartingDependentRouting (14),
 outgoingCallBarring (15),
 incomingCallBarring (16)
 }
AdditionalCallingPartyNumber   ::= Digits
−− Indicates the Additional Calling Party Number. Refer to Q.763 for encoding.
AlertingPattern ::= OCTET STRING (SIZE(3))
−− Indicates a specific pattern that is used to alert a subscriber (e.g. distinctive ringing, tones, etc.).
−− Only the trailing OCTET is used, the remaining OCTETS should be sent as NULL (zero)
−− The receiving side ignores the leading two OCTETS.
ApplicationTimer ::=INTEGER (0..2047)
−− Used by the SCF to set a timer in the SSF. The timer is in seconds.
AssistingSSPIPRoutingAddress   ::= Digits
−− Indicates the destination address of the SRF for the assist procedure.
BackwardGVNS   ::= OCTET STRING (SIZE(minBackwardGVNSLength..maxBackwardGVNSLength))
−− Indicats the GVNS Backward information. Refer to Q.735, º6 for encoding.

BackwardServiceInteractionInd ::= SEQUENCE {
 conferenceTreatmentIndicator [1] OCTET STRING (SIZE(1)) OPTIONAL,
 −− acceptConferenceRequest ’xxxx xx01’B
 −− rejectConferenceRequest ’xxxx xx10’B
 −− network default is accept conference request,
 callCompletionTreatmentIndicator [2] OCTET STRING (SIZE(1)) OPTIONAL
 −− acceptCallCompletionServiceRequest ’xxxx xx01’B,
 −− rejectCallCompletionServiceRequest ’xxxx xx10’B
 −− network default is accept call completion service request
 }


BCSMEvent   ::= SEQUENCE {
 eventTypeBCSM  [0] EventTypeBCSM,
 monitorMode  [1] MonitorMode,
 legID   [2] DTLegID OPTIONAL,
 dpSpecificCriteria [30] DpSpecificCriteria OPTIONAL
 }
−− Indicates the BCSM Event information for monitoring.
BCUSMEvent ::= SEQUENCE{
 eventType [0] EventTypeBCUSM,
 monitorMode [1] MonitorMode
 }

BearerCapabilities ::= BIT STRING {
 speech (0),
 bc64kbits (1),
 bc2x64kbits (2),
 bc384kbits (3),
 bc1536kbits (4),
 bc1920kbits (5),
 multirate (6),
 restrictedDigitalInfo (7),
 bc3−1khzAudio (8),
 bc7khzAudio (9),
 video (10)
 }
BearerCapability   ::= CHOICE {
 bearerCap [0] OCTET STRING (SIZE(2..maxBearerCapabilityLength)),
```

```
 tmr  [1] OCTET STRING (SIZE(1))
 }
−− Indicates the type of bearer capability connection to the user. For bearerCapability, either
−− DSS 1 (Q.931) or the ISUP User Service Information (Q.763) encoding can be used. Refer
−− to the Q.763 Transmission Medium Requirement parameter for tmr encoding.

BothwayThroughConnectionInd ::= ENUMERATED {
 bothwayPathRequired (0),
 bothwayPathNotRequired (1)
 }
−− The default is as specifief in Q.1600

CallConditions   ::= CHOICE {
 userAbandon [0] NULL,
 callFailure [1] CauseValue,
 noReply [2] INTEGER, −− time expressed in seconds
 callRelease [3] NULL,
 ss−invocation [4] InvokableService,
 creditLimitReached [5] INTEGER,
 callDuration [6] INTEGER,
 calledNumber [7] NumberMatch ,
 answeredCall [8] NULL
 }

CalledPartyBCDNumber  ::= OCTET STRING (SIZE(
   minCalledPartyBCDNumberLength ..
   maxCalledPartyBCDNumberLength))
−−  Indicates the Called Party Number, including service selection information. Refer to GSM 04.08 [24]
−−  for encoding. This data type carries only the "type of number", "numbering plan
−−  identification" and "number digit" fields defined in [24]; it does not carry the "called party
−−  BCD number IEI" or "length of called party BCD number contents".

CalledPartyNumber   ::= OCTET STRING (SIZE(minCalledPartyNumberLength..maxCalledPartyNumberLength))
−− Indicates the Called Party Number. Refer to Q.763 for encoding.
CallIdentifier  ::= INTEGER (1..2147483647)
CallingPartyBusinessGroupID  ::= OCTET STRING(SIZE(minCallingPartyBusinessGroupIDLength ..maxCallingPartyBusinessGrou
−− Indicates the business group of the calling party. The value of this octet string is network
−− operator specific.
CallingPartyNumber   ::= OCTET STRING (SIZE (minCallingPartyNumberLength..maxCallingPartyNumberLength))
−− Indicates the Calling Party Number. Refer to Q.763 for encoding.
CallingPartySubaddress  ::= OCTET STRING(SIZE(minCallingPartySubAddressLength ..maxCallingPartySubAddressLength))

−− Indicates the Calling Party Subaddress. Refer to Q.931 for encoding.
CallingPartysCategory ::= OCTET STRING (SIZE(1))
−− Indicates the type of calling party (e.g. operator, payphone, ordinary subscriber). Refer to Q.763
−− for encoding.

CallRecord   ::= SEQUENCE {
 callDuration   [0] XDuration,
 callingPartyNumber [1] CallingPartyNumber ,
 calledPartyNumber [2] CalledPartyNumber
 }
CallResult   ::= OCTET STRING (SIZE (minCallResultLength..maxCallResultLength))

−− This parameter provides the SCF with the charging related information previously requested
−− using the ApplyCharging operation. This shall include the partyToCharge parameter as
−− received in the related ApplyCharging operation to correlate the result to the request
−− The remaining content is network operator specific.
−− Examples of charging related information to be provided by the SSF may be: bulk counter values,
−− costs, tariff change and time of change, time stamps, durations, etc.
−− Examples of conditions on which the charging related information are to be reported may be:
−− threshold value reached, timer expiration, tariff change, end of connection configuration, etc.
DTCallSegmentID   ::= INTEGER (1..numOfCSs)
dtinitialCallSegment INTEGER ::= 1
−− the initial call segment represents the call segment that was there when the CSA was created, ie. the CS where
```

Annex A: incs2−dt

```
−− the trigger took place or the CS that was created by an InitateCallAttempt within a TC−BEGIN message.

Carrier  ::= OCTET STRING(SIZE(minCarrierLength ..maxCarrierLength))
−− This parameter contains the Transit Network Selection. Refer to Q.763 for encoding.

Cause   ::= OCTET STRING (SIZE (minCauseLength..maxCauseLength))
−− Indicates the cause for interface related information. Refer to the Q.763 Cause parameter for encoding.
−− For the use of cause and location values refer to Q.850
CauseValue ::= OCTET STRING (SIZE (1)) −−type extracted from Cause parameter in Q.763.
CGEncountered ::= ENUMERATED {
 noCGencountered(0),
 manualCGencountered(1),
 scpOverload(2)
 }
−− Indicates the type of automatic call gapping encountered, if any.
ChargingEvent   ::= SEQUENCE {
 eventTypeCharging [0] EventTypeCharging ,
 monitorMode [1] MonitorMode,
 legID [2] DTLegID OPTIONAL,
 eventTypeTariff [50] EventTypeTariff OPTIONAL
 }
−− This parameter indicates the charging event  type and corresponding
−− monitor mode and LegID
ChargingParameters    ::= SEQUENCE {
 unitsPerInterval [0] INTEGER (0..maxUnitsPerInterval),
 timePerInterval [1] INTEGER (0..maxTimePerInterval),
 scalingFactor [2] INTEGER (0..maxScalingFactor),
 initialUnitIncrement [3] INTEGER (0..maxInitialUnitIncrement)  OPTIONAL,
 unitsPerDataInterval [4] INTEGER (0..maxUnitsPerDataInterval)  OPTIONAL,
 segmentsPerDataInterval [5] INTEGER (0..maxSegmentsPerDataInterval) OPTIONAL,
 initialTimeInterval [6] INTEGER (0..maxInitialTimeInterval)    OPTIONAL
 }
CollectedDigits ::= SEQUENCE {
 minimumNbOfDigits [0] INTEGER (1..127)  DEFAULT 1,
 maximumNbOfDigits [1] INTEGER (1..127),
 endOfReplyDigit  [2] OCTET STRING (SIZE (1..2)) OPTIONAL,
 cancelDigit  [3] OCTET STRING (SIZE (1..2)) OPTIONAL,
 startDigit  [4] OCTET STRING (SIZE (1..2)) OPTIONAL,
 firstDigitTimeOut [5] INTEGER (1..127)  OPTIONAL,
 interDigitTimeOut [6] INTEGER (1..127)  OPTIONAL,
 errorTreatment  [7] ErrorTreatment, −− DEFAULT reportErrorToScf (forward value references does note work with SDT)
 interruptableAnnInd [8] BOOLEAN   DEFAULT TRUE,
 voiceInformation [9] BOOLEAN   DEFAULT FALSE,
 voiceBack  [10] BOOLEAN   DEFAULT FALSE
 }
−− The use of voiceBack is network operator specific.
−− The endOfReplyDigit, cancelDigit, and startDigit parameters have been designated as OCTET STRING,
−− and are to be encoded as  BCD, one digit per octet only, contained
−− in the four least significant bits of each OCTET. The usage is service dependent.
CollectedInfo ::= CHOICE {
 collectedDigits [0] CollectedDigits,
 iA5Information [1] BOOLEAN
 }

ConnectedNumberTreatmentInd ::= ENUMERATED {
 noINImpact (0),
 presentationRestricted (1),
 presentCalledINNumber (2),
 presentCalledINNumberRestricted (3)
 }
−− The default is as specifief in Q.1600

Constraints ::= SEQUENCE {
 maximumNumberOfDigits [1] INTEGER (1..127),
 minimumNumberOfDigits [2] INTEGER (1..127),
```

Annex A: incs2−dt

```
 typeOfRequestedInfo [3] InfoType, −− DEFAULT numericString (forward value references does not work with SDT)
 numberOfAllowedRetries [4] INTEGER (0..127) DEFAULT 0
 }

ControlType ::= ENUMERATED {
 sCPOverloaded(0),
 manuallyInitiated(1),
 destinationOverload(2)
 −− other values FFS
 }

CorrelationID  ::= Digits
−− used by SCF for correlation with a previous operation. Refer to clause 17 for a description of the procedures
−− associated with this parameter.
CounterAndValue ::= SEQUENCE {
  counterID [0] CounterID,
  counterValue [1] Integer4
  }
CounterID ::= INTEGER (0..99)
−− Indicates the counters to be incremented.
−− The counterIDs can be addressed by using the last digits of the dialed number.
CountersValue ::= SEQUENCE SIZE(0..numOfCounters) OF CounterAndValue
Credit  ::= CHOICE {
  currency CurrencyValue ,
  units CreditUnit
   }
CreditUnit ::= INTEGER (0..maxCreditUnit)
CriticalityType ::= ENUMERATED {
  ignore(0),
  abort(1)
  }
CSAID  ::= INTEGER (1..numOfCSAs)
−− Indicates the SSF CSA identifier
CUApplicationInd  ::=  CHOICE{
 localValue [0] INTEGER,
 global [1] OBJECT IDENTIFIER
 }
−− Parameter 'cUApplicationInd' identifies the triggered application. Possible applications: CCBS, CCNR, Q.SIG, etc
CurrencyID ::= PrintableString (SIZE (3) ) −− ISO 639 code
CurrencyValue  ::= SEQUENCE {
 currency CurrencyID,
 amount INTEGER (0..maxAmount)
 }
CutAndPaste ::= INTEGER (0..22)
−− Indicates the number of digits to be deleted.

DateAndTime ::= OCTET STRING (SIZE(6))
−− Indicates, amongst others, the start time for activate service filtering. Coded as YYMMDDHHMMSS
−− with each digit coded BCD.
−− The first octet contains YY and the remaining items are sequenced following.
−− For example, 1993 September 30th, 12:15:01 would be encoded as:
−− Bits     HGFE  DCBA
−− leading octet  3 9
−−     9 0
−−     0 3
−−     2 1
−−     5 1
−−     1 0
DestinationRoutingAddress  ::= SEQUENCE SIZE(1) OF CalledPartyNumber
−− Indicates the Called Party Number.
Digits  ::= OCTET STRING (SIZE (minDigitsLength..maxDigitsLength))
−− Indicates the address signalling digits. Refer to the Q.763 Generic Number and Generic Digits parameters
−− for encoding. The coding of the subfields 'NumberQualifier' in Generic Number and 'TypeOfDigits' in
−− Generic Digits are irrelevant to the INAP, the ASN.1 tags are sufficient to identify the parameter.
−− The ISUP format does not allow to exclude these subfields, therefore the value is network operator specific.
```

Annex A: incs2−dt

```
−− The following parameters should use Generic Number:
−− CorrelationID for AssistRequestInstructions, AssistingSSPIPRoutingAddress for EstablishTemporaryConnection,
−− calledAddressValue for all occurrences,callingAddressValue for all occurrences.
−− The following parameters should use Generic Digits: prefix, all
−− other CorrelationID occurrences, dialledNumber filtering criteria, callingLineID filtering criteria, lineID for ResourceID
−− type, digitResponse for ReceivedInformationArg, iNServiceControlLow / iNServiceControlHighfor MidCallInfoType,
−− iNServiceControlCode for MidCallInfo.
DisplayInformation  ::= IA5String (SIZE (minDisplayInformationLength..maxDisplayInformationLength))
−− Indicates the display information.
−− Delivery of DisplayInformation parameter to Private Networks cannot be guaranteed due to signalling
−− interworking problems, solutions are currently under study
DpSpecificCriteria  ::= CHOICE {
  numberOfDigits [0] NumberOfDigits,
  applicationTimer [1] ApplicationTimer,
  midCallControlInfo [2] MidCallControlInfo
  }

−− The SCF may specify the number of digits to be collected by the SSF for the EventReportBCSMEvent event.
−− When all digits are collected, the SSF reports the event to the SCF.
−− The SCF may set a timer in the SSF for the No Answer event. If the user does not answer the call
−− within the allotted time, the SSF reports the event to the SCF
XDuration ::= INTEGER (−2..86400)
−− Values are seconds

ElementaryMessageID ::= Integer4

Entry ::=CHOICE {
 agreements [0] OBJECT IDENTIFIER,
 networkSpecific [1] Integer4
 }

ErrorTreatment ::= ENUMERATED {
  reportErrorToScf(0),
  help(1),
  repeatPrompt(2)
  }
−− reportErrorToScf means returning the "ImproperCallerResponse" error in the event of an error
−− condition during collection of user info.
EventSpecificInformationBCSM  ::= CHOICE {
 collectedInfoSpecificInfo [0] SEQUENCE {
   calledPartynumber [0] CalledPartyNumber
   },
 analysedInfoSpecificInfo [1] SEQUENCE {
   calledPartynumber [0] CalledPartyNumber
   },
 routeSelectFailureSpecificInfo [2] SEQUENCE {
   failureCause [0] Cause  OPTIONAL
   },
 oCalledPartyBusySpecificInfo [3] SEQUENCE {
   busyCause [0] Cause  OPTIONAL
   },
 oNoAnswerSpecificInfo [4] SEQUENCE {
   null NULL −− no specific info defined −−
   },
 oAnswerSpecificInfo [5] SEQUENCE {
   backwardGVNS [0] BackwardGVNS OPTIONAL
   },
 oMidCallSpecificInfo [6] SEQUENCE {
   connectTime [0] Integer4 OPTIONAL,
   oMidCallInfo [1] MidCallInfo OPTIONAL
   },
 oDisconnectSpecificInfo [7] SEQUENCE {
   releaseCause [0] Cause  OPTIONAL,
   connectTime [1] Integer4 OPTIONAL
   },
```

Annex A: incs2−dt

```
tBusySpecificInfo [8] SEQUENCE {
  busyCause [0] Cause  OPTIONAL
  },
tNoAnswerSpecificInfo [9] SEQUENCE {
  null NULL −− no specific info defined −−
  },
tAnswerSpecificInfo [10] SEQUENCE {
  null NULL −− no specific info defined −−
  },
tMidCallSpecificInfo [11] SEQUENCE {
  connectTime [0] Integer4 OPTIONAL,
  tMidCallInfo [1] MidCallInfo OPTIONAL
  },
tDisconnectSpecificInfo [12] SEQUENCE {
  releaseCause [0] Cause  OPTIONAL,
  connectTime [1] Integer4 OPTIONAL
  },
oTermSeizedSpecificInfo [13] SEQUENCE {
  null NULL −− no specific info defined
  },
oSuspended [14] SEQUENCE {
  null NULL −− no specific info defined
  },
tSuspended [15] SEQUENCE {
  null NULL −− no specific info defined
  },
origAttemptAuthorized [16] SEQUENCE {
  null NULL −− no specific info defined
  },
oReAnswer [17] SEQUENCE {
  null NULL −− no specific info defined
  },
tReAnswer [18] SEQUENCE {
  null NULL −− no specific info defined
  },
facilitySelectedAndAvailable [19] SEQUENCE {
  null NULL −− no specific info defined
  },
callAccepted [20] SEQUENCE {
  null NULL −− no specific info defined
  },
oAbandon [21] SEQUENCE {
  abandonCause [0] Cause OPTIONAL
  },
tAbandon [22] SEQUENCE {
  abandonCause [0] Cause  OPTIONAL
  },
terminationAttemptAuthorized [24] SEQUENCE {
  null NULL −− no specific info defined
  }
}

−− Indicates the call related information specific to the event.
−− The connectTime indicates the duration between the received answer indication from the called party side
−− and the release of the connection for ODisconnect, OException, TDisconnect, or TException or between
−− the received answer indication from the called party side and the time of detection of the required
−− mid call event.
−− The unit for the connectTime is 100 milliseconds

−−EventSpecificInformationBCUSM  ::=  CHOICE {
−−  componentRecievedSpecificInfo [0] SEQUENCE {
−−   componentReceivedInfo [0] OCTET STRING{SIZE (minCompReceivedInforLen..maxCompRecievedInfoLen) OPTIONAL,
−− Its content is network signalling/operator specific.
−− The internal structure of this parameter can be defined using ASN.1 and the related Basic
−− Encoding Rules (BER). In such a case the value of this paramter (after the first tag and length
```

Annex A: incs2−dt

```
−− information) is the BER encoding of the defined ASN.1 internal structure.
−− The tag of this parameter as defined by ETSI is never replaced.
−−   },
−− associationReleaseRequestedSpecificInfo [1] SEQUENCE{
−−   associationReleaseInfo [0]  OCTET STRING(SIZE(minAssReleaseInfoLength..maxAssReleaseInfoLength) OPTIONAL,
−−   releaseCause [1]  Cause OPTIONAL
−−  }
−− }
EventSpecificInformationCharging  ::= OCTET STRING (SIZE (minEventSpecificInformationChargingLength..maxEventSpecificInf
−− defined by network operator.
−− Indicates the charging related information specific to the event.
−− An example data type definition for this parameter is given below:
−−  chargePulses  [0] Integer4,
−−  chargeMessages  [1] OCTET STRING (SIZE (min..max))
EventTypeBCSM ::= ENUMERATED {
 origAttemptAuthorized(1),
 collectedInfo(2),
 analysedInformation(3),
 routeSelectFailure(4),
 oCalledPartyBusy(5),
 oNoAnswer(6),
 oAnswer(7),
 oMidCall(8),
 oDisconnect(9),
 oAbandon(10),
 termAttemptAuthorized(12),
 tBusy(13),
 tNoAnswer(14),
 tAnswer(15),
 tMidCall(16),
 tDisconnect(17),
 tAbandon(18),
 oTermSeized(19),
 oSuspended(20),
 tSuspended(21),
 origAttempt(22),
 termAttempt(23),
 oReAnswer(24),
 tReAnswer(25),
 facilitySelectedAndAvailable(26),
 callAccepted(27)
 }

−− Indicates the BCSM detection point event. Refer to Q.1224 for additional information on the events.
−− Values origAttempt and termAttempt can only be used for TDPs

EventTypeBCUSM ::= ENUMERATED{
  componentReceived(127),
  associationReleaseRequested(126),
  activationReceivedAndAuthorized(125)
  }

EventTypeCharging  ::= OCTET STRING (SIZE (minEventTypeChargingLength..maxEventTypeChargingLength))
−− This parameter indicates the charging event type. Its content is network operator specific.
−−
−− An example data type definition for this parameter is given below:
−− EventTypeCharging ::= ENUMERATED {
−−     chargePulses (0),
−−     chargeMessages (1)
−−     }
EventTypeTariff ::= ENUMERATED {
 chargingTariffInformation (0),
 addOnChargingInformation (1),
 chargingAcknowledgementInformation (2),
 chargingAcknowledgeTimerExpired (3)
```

```
 }
ExtensionField  ::= OCTET STRING −− Note: Changed to be in line with Z.105
−−This parameter indicates an extension of an argument data type. Its content is network operator specific

FCIBillingChargingCharacteristics  ::= CHOICE {
 fCIBCCcs1 OCTET STRING (SIZE (minFCIBCcs1Length..maxFCIBCcs1Length)),
 fCIBCCsequencecs2 [51] SEQUENCE {
  fCIBCC [0] OCTET STRING (SIZE (minFCIBCCcs2Length ..maxFCIBCCcs2Length)) OPTIONAL,
  tariff [1] CHOICE {
   crgt [0] ChargingTariffInformation,
   aocrg [1] AddOnChargingInformation
   } OPTIONAL
  }
 }

−− This parameter indicates the billing and/or charging characteristics. Its content is network operator specific.
−− An example datatype definition for this parameter is given below:
−− FCIBillingChargingCharacteristics ::= CHOICE {
−−  completeChargingrecord  [0] OCTET STRING (SIZE (min..max)),
−− correlationID    [1] CorrelationID,
−− scenario2Dot3    [2] SEQUENCE {
−−       chargeParty  [0] DTLegID  OPTIONAL,
−−       chargeLevel  [1] OCTET STRING (SIZE (min..max))
−−            OPTIONAL,
−−       chargeItems  [2] SET OF Attribute OPTIONAL
−−       }
−− }
FilteredCallTreatment  ::= SEQUENCE {
 sFBillingChargingCharacteristics [0] SFBillingChargingCharacteristics,
 informationToSend   [1] InformationToSend   OPTIONAL,
 maximumNumberOfCounters   [2] MaximumNumberOfCounters  OPTIONAL,
 releaseCause    [3] Cause    OPTIONAL,
 sFTariffMessage    [50] CHOICE {
  crgt     [0] ChargingTariffInformation} OPTIONAL
 }
−− If releaseCause is not present, the default value is the same as the ISUP cause value decimal 31.
−− If informationToSend is present, the call will be released after the end of the announcement
−− with the indicated or default releaseCause.
−− If maximumNumberOfCounters is not present, ServiceFilteringResponse will be sent with
−− CountersValue::= SEQUENCE SIZE (0) OF CountersAndValue
FilteringCharacteristics ::= CHOICE {
 interval [0] INTEGER (−1..32000),
 numberOfCalls [1] Integer4
 }
−− Indicates the severity of the filtering and the point in time when the ServiceFilteringResponse is to be sent.
−− If = interval, every interval of time the next call leads to an InitialDP and a ServiceFilteringResponse is sent to
−− the SCF. The interval is specified in seconds.
−− If = NumberOfCalls, every N calls the Nth call leads to an InitialDP and a ServiceFilteringResponse
−− is sent to the SCF.
−− If ActivateServiceFiltering implies several counters − filtering on several dialled numbers −,
−− the numberOfCalls would include calls to all the dialled numbers.
FilteringCriteria  ::= CHOICE {
 serviceKey  [2] ServiceKey,
 addressAndService [30] SEQUENCE {
   calledAddressValue [0] Digits ,
   serviceKey  [1] ServiceKey,
   callingAddressValue [2] Digits OPTIONAL,
   locationNumber  [3] LocationNumber OPTIONAL
   }
 }
−− In case calledAddressValue is specified, the numbers to be filtered are from calledAddressValue
−− up to and including calledAddressValue + maximumNumberOfCounters−1.
−− The last two digits of calledAddressvalue can not exceed 100−maximumNumberOfCounters.
FilteringTimeOut ::= CHOICE {
 duration [0] XDuration,
```

Annex A: incs2−dt

```
 stopTime [1] DateAndTime
 }
−− Indicates the maximum duration of the filtering. When the timer expires, a ServiceFilteringResponse
−− is sent to the SCF.
ForwardCallIndicators ::= OCTET STRING (SIZE(2))
−− Indicates the Forward Call Indicators. Refer to Q.763 for encoding
ForwardGVNS  ::= OCTET STRING (SIZE(minForwardGVNSLength..maxForwardGVNSLength))
−− Indicats the GVNS Forward information. Refer to Q.735, º6 for encoding.
ForwardServiceInteractionInd ::= SEQUENCE {
 conferenceTreatmentIndicator [1] OCTET STRING (SIZE(1)) OPTIONAL,
 −− acceptConferenceRequest ’xxxx xx01’,B
 −− rejectConferenceRequest ’xxxx xx10’B
 −− network default is accept conference request
 callDiversionTreatmentIndicator [2] OCTET STRING (SIZE(1)) OPTIONAL,
 −− callDiversionAllowed ’xxxx xx01’B
 −− callDiversionNotAllowed ’xxxx xx10’B
 −− network default is Call Diversion allowed
 callOfferingTreatmentIndicator [3] OCTET STRING (SIZE(1)) OPTIONAL,
 −− callOfferingNotAllowed ’xxxx xx01’B,
 −− callOfferingAllowed  ’xxxx xx10’B
 −− network default is Call Offering not allowed
  callingPartyRestrictionIndicator [4] OCTET STRING (SIZE(1)) OPTIONAL
  −− noINImpact    ’xxxx xx01’B,
  −− presentationRestricted  ’xxxx xx10’B
  −− network default is noINImpact
 }

GapCriteria  ::= CHOICE {
 calledAddressValue [0] Digits ,
 gapOnService  [2] GapOnService,
 gapAllInTraffic  [3] NULL,
 calledAddressAndService [29] SEQUENCE {
   calledAddressValue [0] Digits ,
   serviceKey  [1] ServiceKey
   },
 callingAddressAndService [30] SEQUENCE {
   callingAddressValue [0] Digits ,
   serviceKey  [1] ServiceKey,
   locationNumber  [2] LocationNumber OPTIONAL
   }
 }
−− Both calledAddressValue and callingAddressValue can be
−− incomplete numbers, in the sense that a limited amount of digits can be given.
−−
−− For the handling of numbers starting with the same digit string refer to the detailed procedure
−− of the CallGap operation
GapOnService ::= SEQUENCE {
 serviceKey [0] ServiceKey
 }
GapIndicators ::= SEQUENCE {
 duration [0] XDuration,
 gapInterval [1] Interval
 }
−− Indicates the gapping characteristics. No gapping when gapInterval equals 0, and gap all calls when
−− gapInterval equals −1.
GapTreatment   ::= CHOICE {
 informationToSend [0] InformationToSend ,
 releaseCause  [1] Cause ,
 both   [2] SEQUENCE {
      informationToSend [0] InformationToSend ,
      releaseCause  [1] Cause
   }
 }
−− The default value for Cause is the same as in ISUP.
GenericName   ::= OCTET STRING (SIZE(minGenericNameLength..maxGenericNameLength))
```

Annex A: incs2−dt

```
−− Refer to Q.931 Display Information for encoding
GenericNumber   ::= OCTET STRING (SIZE(minGenericNumberLength..maxGenericNumberLength))
−− Refer to Q.763 Generic Number for encoding.
GenericNumbers  ::= SET SIZE(1..numOfGenericNumbers) OF GenericNumber
HighLayerCompatibilities ::= BIT STRING {
 telephony (0),
 facsimileGroup2−3 (1),
 facsimileGroup4classeI (2),
 teletexMixedMode (3),
 teletexProcessableMode (4),
 teletexBasicMode (5),
 syntaxBasedVideotex (6),
 internationalVideotex (7),
 telexService (8),
 messageHandlingSystem (9),
 osiApplication (10),
 audioVisual (11)
 }
HighLayerCompatibility ::= OCTET STRING (SIZE (highLayerCompatibilityLength))
−− Indicates the teleservice. For encoding, DSS1 (Q.931) is used.
InbandInfo   ::= SEQUENCE {
 messageID [0] MessageID ,
 numberOfRepetitions [1] INTEGER (1..127) OPTIONAL,
 duration [2] INTEGER (0..32767) OPTIONAL,
 interval [3] INTEGER (0.. 32767) OPTIONAL
 }
−− Interval is the time in seconds between each repeated announcement. Duration is the total
−− amount of time in seconds, including repetitions and intervals.
−− The end of announcement is either the end of duration or numberOfRepetitions, whatever comes first.
−− duration with value 0 indicates infinite duration
InformationToRecord  ::= SEQUENCE {
 messageID [0] ElementaryMessageID OPTIONAL,
 messageDeletionTimeOut [1] INTEGER (1..3600) OPTIONAL,
 −− Time units = hours
 timeToRecord [3] INTEGER (0..maxRecordingTime) OPTIONAL,
−− Time units = seconds
 controlDigits [4] SEQUENCE {
  endOfRecordingDigit [0] OCTET STRING (SIZE(1..2)) OPTIONAL,
  cancelDigit [1] OCTET STRING (SIZE(1..2)) OPTIONAL,
  replayDigit [2] OCTET STRING (SIZE(1..2)) OPTIONAL,
  restartRecordingDigit [3] OCTET STRING (SIZE(1..2)) OPTIONAL,
  restartAllowed [4] BOOLEAN  DEFAULT FALSE,
  replayAllowed [5] BOOLEAN  DEFAULT FALSE
  }
 }
InformationToSend  ::= CHOICE {
 inbandInfo [0] InbandInfo ,
 tone [1] Tone,
 displayInformation [2] DisplayInformation
 }
InfoToSend  ::= CHOICE {
 messageID [0] MessageID ,
 toneId [1] ToneId,
 displayInformation [2] DisplayInformation
 }
InfoType ::= ENUMERATED {
 numericString (0),
 characterString (1),
 iA5String (2)
 }
INServiceCompatibilityIndication  ::= SEQUENCE SIZE (1..numOfInServiceCompatibilityIndLength) OF Entry
INServiceCompatibilityResponse  ::= Entry
Integer4 ::= INTEGER(0..2147483647)
Interval ::= INTEGER (−1..60000)
−− Units are milliseconds.
```

Annex A: incs2−dt

```
InvokableService ::= ENUMERATED {
 callingLineIdentificationRestriction (1),
 connectedLineIdentificationRestriction (2),
 callWaiting (3),
 callHold (4),
 reverseCharging (5),
 explicitCallTransfer (6),
 callCompletionOnBusySubscriber (7)
 }
InvokeID ::= InvokeIdType
−− Operation invoke identifier.
IPAvailable  ::= OCTET STRING (SIZE (minIPAvailableLength..maxIPAvailableLength))
−− defined by network operator.
−− Indicates that the resource is available.
IPRoutingAddress  ::= CalledPartyNumber
−− Indicates the routing address for the IP.
IPSSPCapabilities  ::= OCTET STRING (SIZE (minIPSSPCapabilitiesLength..maxIPSSPCapabilitiesLength))
−− defined by network operator.
−− Indicates the SRF resources available at the SSP.
ISDNAccessRelatedInformation  ::= OCTET STRING(SIZE(minISDNAccessRelatedInformationLength ..maxISDNAccessRelatedInf

−− Indicates the destination user network interface related information. Refer to the Q.763 Access
−− Transport parameter for encoding.
Language ::= PrintableString (SIZE (3) ) −− ISO 639 codes only;

DTLegID ::= CHOICE {
 sendingSideID [0] DTLegType,
 receivingSideID [1] DTLegType
 }
−− Indicates a reference to a specific party in a call. OPTIONAL denotes network operator specific use
−− with a choice of unilateral ID assignment or bilateral ID assignment.
−− OPTIONAL for LegID also denotes the following:
−−when only one party exists in the call, this parameter is not needed (as no ambiguity exists);
−−when more than one party exists in the call, one of the following alternatives applies:
−−  1. LegID is present and indicates which party is concerned.
−−  2. LegID is not present and a default value is assumed (e.g. calling party in the case of the
−−     ApplyCharging operation).
−− Choice between these two alternatives is kept a network operator option.
DTLegType ::= OCTET STRING (SIZE(1))
dtleg1 DTLegType ::= ’01’H
dtleg2 DTLegType ::= ’02’H
LocationNumber  ::= OCTET STRING (SIZE (minLocationNumberLength..maxLocationNumberLength))
−− Indicates the Location Number for the calling party. Refer to Q.763 (White book) for encoding.
MailBoxID  ::= OCTET STRING (SIZE(minMailBoxIDLength..maxMailBoxIDLength))
MaximumNumberOfCounters ::= INTEGER (1..numOfCounters)
Media ::= ENUMERATED {
 voiceMail (0),
 faxGroup3 (1),
 faxGroup4 (2)
 }

MessageID  ::= CHOICE {
 elementaryMessageID [0] Integer4,
 text [1] SEQUENCE {
    messageContent [0] IA5String (SIZE (minMessageContentLength..maxMessageContentLength)),
    attributes [1] OCTET STRING (SIZE (minAttributesLength..maxAttributesLength)) OPTIONAL
    },
 elementaryMessageIDs [29] SEQUENCE SIZE (1.. numOfMessageIDs) OF Integer4,
 variableMessage [30] SEQUENCE {
    elementaryMessageID [0] Integer4,
    variableParts [1] SEQUENCE SIZE (1..5) OF VariablePart
    }
 }
−− OPTIONAL denotes network operator specific use.
MidCallControlInfo  ::= SEQUENCE SIZE (
```

```
  minMidCallControlInfoNum ..
  maxMidCallControlInfoNum) OF SEQUENCE {
   midCallInfoType  [0] MidCallInfoType ,
   midCallReportType [1] ENUMERATED {
        inMonitoringState(0),
        inAnyState(1)
       } DEFAULT inMonitoringState
   }
MidCallInfo  ::= SEQUENCE {
 iNServiceControlCode [0] Digits
 }
MidCallInfoType  ::= SEQUENCE {
 iNServiceControlCodeLow [0] Digits ,
 iNServiceControlCodeHigh [1] Digits  OPTIONAL
 }
DTMiscCallInfo ::= SEQUENCE {
 messageType [0] ENUMERATED {
    request(0),
    notification(1)
    }
 }
−− Indicates detection point related information.
MonitorMode  ::= ENUMERATED {
 interrupted(0),
 notifyAndContinue(1),
 transparent(2)
 }
−− Indicates the event is relayed and/or processed by the SSP.
−− If this parameter is used in the context of charging events, the following definitions apply for the
−− handling of charging events:
−− Interrupted means that the SSF notifies the SCF of the charging event using
−− EventNotificationCharging, does not process the event but discard it.
−− NotifyAndContinue means that SSF notifies the SCF of the charging event using
−− EventNotificationCharging, and continues processing the event or signal without waiting for SCF instructions.
−− Transparent means that the SSF does not notify the SCF of the event. This value is used to end the monitoring
−− of a previously requested charging event. Previously requested charging events are monitored
−− until ended by a transparent monitor mode, or until the end of the connection configuration.
−− For the use of this parameter in the context of BCSM events refer to clause 17.
Notification ::= ENUMERATED {
 userAbandon (0),
 callFailure (1),
 noReply (2),
 callRelease (3),
 ssInvocation (4),
 creditLimitReached (5),
 callDuration (6),
 calledNumber (7),
 answeredCall (8)
 }
NotificationInformation  ::= CHOICE {
 userAbandonSpecificInfo [0] NULL,
 callFailureSpecificInfo [1] SEQUENCE {
   failureCause [0] Cause OPTIONAL
  },
 noReplySpecificInfo [2] NULL,
 callReleaseSpecificInfo [3] SEQUENCE {
   releaseCause [0] Cause   OPTIONAL,
   timeStamp [1] DateAndTime  OPTIONAL
  },
 ssInvocationSpecificInfo [4] SEQUENCE {
   invokedService [0] InvokableService
  },
 creditLimitReachedSpecificInfo [5] SEQUENCE {
   timeStamp [0] DateAndTime  OPTIONAL
  },
```

```
 callDurationSpecificInfo [6] SEQUENCE {
   timeStamp [0] DateAndTime  OPTIONAL
  },
 calledNumberSpecificInfo [7] SEQUENCE {
   calledNumber [0] CalledPartyNumber  OPTIONAL
  },
 answeredCallSpecificInfo [8] SEQUENCE {
   timeStamp [0] DateAndTime  OPTIONAL
  }
 }

NumberOfDigits ::= INTEGER (1..255)
−− Indicates the number of digits to be collected

OCSIApplicacable ::= NULL
−−  Indicates that the OCSI, if present, shall be applied on the outgoing call leg created with a Connect operation
OperationCode ::= CHOICE {
 globalCode OBJECT IDENTIFIER,
 local INTEGER
 }
−− contains the operation value, or error value (object identifier), or problem value of the FACILITY IE,
−− and the argument, the result, or the reject part of the same FACILITY IE that are received with DSS1
−− message from the user.  (see Q.932 8.2.2 for encoring)

OriginalCalledPartyID  ::= OCTET STRING (SIZE( minOriginalCalledPartyIDLength..maxOriginalCalledPartyIDLength))
−− Indicates the original called number. Refer to the Q.763 Original Called Number for encoding.
ProfileIdentifier   ::= CHOICE {
access [0]  CalledPartyNumber ,
group  [1]  FacilityGroup
 }
−− Please note that 'CalledPartyNumber' is used to address a subscriber access line.
−−The data type was reused from the existing types to avoid the definition of a new one.

Reason  ::= OCTET STRING(SIZE(minReasonLength..maxReasonLength))
ReceivedInformation  ::= SEQUENCE SIZE (minReceivedInformationLength..maxReceivedInformationLength) OF IA5String
ReceivedStatus ::=ENUMERATED {
 messageComplete (0),
 messageInterrupted (1),
 messageTimeOut (2)
 }
RecordedMessageID ::= Integer4
RedirectingPartyID  ::= OCTET STRING (SIZE (minRedirectingPartyIDLength..maxRedirectingPartyIDLength))
−− Indicates redirecting number. Refer to the Q.763 Redirecting number for encoding.
RedirectionInformation ::= OCTET STRING (SIZE(2))
−− Indicates redirection information. Refer to the Q.763 Redirection Information for encoding.
RegistratorIdentifier    ::= OCTET STRING(SIZE(minRegistratorIdentifierLength ..maxRegistratorIdentifierLength))


RequestedInformationList  ::= SEQUENCE SIZE (1..numOfInfoItems) OF RequestedInformation
RequestedInformationTypeList ::= SEQUENCE SIZE (1..numOfInfoItems) OF RequestedInformationType
RequestedInformation  ::= SEQUENCE {
 requestedInformationType [0] RequestedInformationType,
 requestedInformationValue [1] RequestedInformationValue
 }
RequestedInformationType ::= ENUMERATED {
 callAttemptElapsedTime(0),
 callStopTime(1),
 callConnectedElapsedTime(2),
 calledAddress(3),
 releaseCause(30)
 }
RequestedInformationValue  ::= CHOICE {
 callAttemptElapsedTimeValue [0] INTEGER (0..255),
 callStopTimeValue  [1] DateAndTime,
 callConnectedElapsedTimeValue [2] Integer4,
```

```
 calledAddressValue  [3] Digits ,
 releaseCauseValue  [30] Cause
 }

−− The callAttemptElapsedTimeValue is specified in seconds. The unit for the
−− callConnectedElapsedTimeValue is 100 milliseconds
RequestedNotifications  ::= SET OF CallConditions
RequestedType ::= INTEGER (0 .. 127)
RequestedUTSI  ::= SEQUENCE {
 uSIServiceIndicator [0] USIServiceIndicator ,
 uSImonitorMode  [1] USIMonitorMode
 }

RequestedUTSIList  ::= SEQUENCE SIZE (minRequestedUTSINum..maxRequestedUTSINum) OF  RequestedUTSI


ResponseCondition ::= ENUMERATED {
 intermediateResponse(0),
 lastResponse(1)
−− additional values are for further study
 }
−− ResponseCondition is used to identify the reason why ServiceFilteringResponse operation is sent.
−− intermediateresponse identifies that service filtering is running and the interval time is expired and
−− a call is received, or that service filtering is running and the threshold value is reached.
−− lastResponse identifies that the duration time is expired and service filtering has been finished or
−− that the stop time is met and service filtering has been finished.
RouteList  ::= SEQUENCE SIZE(1..3) OF OCTET STRING (SIZE (minRouteListLength..maxRouteListLength))
−− Indicates a list of trunk groups or a route index. See Q.1224 for additional information on this item.
RoutingAddress  ::= CHOICE {
 routingProhibited [0] NULL,
 destinationRoutingAddress [1] DestinationRoutingAddress
 }
ScfAddress  ::= OCTET STRING (SIZE (minScfAddressLength..maxScfAddressLength))
−− ISDN address
ScfID  ::= OCTET STRING (SIZE (minScfIDLength..maxScfIDLength))
−− defined by network operator.
−− Indicates the SCF identity.
−− Used to derive the INAP address of the SCF to establish a connection between a requesting FE
−− and the specified SCF.
−− When ScfID is used in an operation which may cross an internetwork boundary, its encoding must
−− be understood in both networks; this requires bilateral agreement on the encoding.
−− Refer to 3.5/Q.713 "calling party subaddress" parameter for encoding. It indicates the SCCP address of the SCF.
−− Other encoding schemes are also possible as an operator specific option.
SCIBillingChargingCharacteristics  ::= OCTET STRING (SIZE (minSCIBillingChargingLength..maxSCIBillingChargingLength))
−− This parameter indicates the billing and/or charging characteristics. Its content is network operator specific.
−− An example datatype definition for this parameter is given below:
−− SCIBillingChargingCharacteristics  ::= CHOICE {
−−  chargeLevel  [0] OCTET STRING (SIZE (min..max),
−−  chargePulses [1] Integer4,
−−  chargeMessages [2] OCTET STRING (SIZE (min..max)
−− }
ServiceInteractionIndicators  ::= OCTET STRING (SIZE (minServiceInteractionIndicatorsLength..maxServiceInteractionIndicatorsLe
−− Indicators which are exchanged between SSP and SCP to resolve interactions between IN based services
−− and network based services, respectively between different IN based services.
−− The contents are network specific.
−− Note this parameter is kept in CS2 for backward compatibility to CS−1R, for CS2 see new
−− parameter ServiceInteractionIndicatorsTwo
ServiceInteractionIndicatorsTwo ::=  SEQUENCE {
 forwardServiceInteractionInd  [0] ForwardServiceInteractionInd OPTIONAL,
 −− applicable to operations IDP, CON, ICA, CWA.
 backwardServiceInteractionInd  [1] BackwardServiceInteractionInd OPTIONAL,
 −− applicable to operations IDP, CON, CTR, CWA, ETC.
 bothwayThroughConnectionInd  [2] BothwayThroughConnectionInd  OPTIONAL,
 −− applicable to operations CTR, ETC.
 suspendTimer    [3] SuspendTimer   OPTIONAL,
```

```
 −− applicable to operations CON, ICA, CWA.
 connectedNumberTreatmentInd  [4] ConnectedNumberTreatmentInd  OPTIONAL,
 −− applicable to operations CON, CTR, CWA, ETC.
 suppressCallDiversionNotification [5] BOOLEAN     OPTIONAL,
 −− applicable to CON, ICA, CWA
 suppressCallTransferNotification [6] BOOLEAN     OPTIONAL,
 −− applicable to CON, ICA, CWA
 allowCdINNoPresentationInd  [7] BOOLEAN    OPTIONAL,
 −− applicable to CON, ICA, CWA
 −− indicates whether the Number Presentation not allowed indicator of the ISUP
 −− "called IN number" shall be set to presentation allowed (TRUE) or presentation not allowed (FALSE)
 userDialogueDurationInd   [8] BOOLEAN    DEFAULT TRUE
 −− applicable when interaction with the user is required, if the interaction
 −− TRUE means the user interaction may last longer than 90 seconds. Otherwise the
 −− indicator should be set to FALSE.
 −− used for delaying ISUP T9 timer.
 }
−− Indicators which are exchanged between SSP and SCP to resolve interactions between IN based services
−− and network based services, respectively between different IN based services.

ServiceKey ::= Integer4
−− Information that allows the SCF to choose the appropriate service logic.
SFBillingChargingCharacteristics  ::= OCTET STRING (SIZE (minSFBillingChargingLength..maxSFBillingChargingLength))
−− This parameter indicates the billing and/or charging characteristics for filtered calls.
−− Its content is network operator specific
SubscriberId  ::= GenericNumber
SupplementaryServices ::= BIT STRING {
 callingLineIdentificationPresentationSS (1),
 callingLineIdentificationRestrictionSS (2),
 connectedLineIdentificationPresentationSS (3),
 connectedLineIdentificationRestrictionSS (4),
 callForwardingOnNoReplySS (5),
 callForwardingUnconditionalSS (6),
 callForwardingOnBusySS (7),
 callForwardingOnNotReachableSS (8),
 callWaitingSS (9),
 callHoldSS (10),
 reverseChargingSS (11),
 explicitCallTransferSS (12),
 callCompletionOnBusySubscriberSS (13),
 adviceOfChargeOnStartSS (14),
 adviceOfChargeAtEndSS (15),
 adviceOfChargeDuringCallSS (16),
 timeDependentRoutingSS (17),
 callingPartingDependentRoutingSS (18),
 outgoingCallBarringSS (19),
 incomingCallBarringSS (20)
 }
SuspendTimer ::= INTEGER (−1..120) −−value in seconds
−− The value −1 indicates the network specific suspend timer (T6) must be used.
TerminalType ::= ENUMERATED {
 unknown(0),
 dialPulse(1),
 dtmf(2),
 isdn(3),
 isdnNoDtmf(4),
 spare(16)
 }
−− Identifies the terminal type so that the SCF can specify, to the SRF, the appropriate type of capability
−− (voice recognition, DTMF, display capability, etc.).
DTTimerID ::= ENUMERATED {
 tssf(0)
 −− others ffs
 }
−− Indicates the timer to be reset.
```

Annex A: incs2−dt

```
TimerValue ::= Integer4
−− Indicates the timer value (in seconds).
Tone ::= SEQUENCE {
 toneID [0] Integer4,
 duration [1] Integer4 OPTIONAL
 }
−− The duration specifies the length of the tone in seconds, value 0 indicates infinite duration.
ToneId ::= CHOICE {
 local [0] Integer4,
 global [1] OBJECT IDENTIFIER
 }
TraceInformation  ::= SEQUENCE OF TraceItem
TraceItem  ::= SET {scf [0] ScfID}
TriggerDataIdentifier  ::= SEQUENCE {
 triggerID  [0] EventTypeBCSM,
 profileIdentifier [1] ProfileIdentifier
 }
−− It is for further study whether al TDP types really apply

UnavailableNetworkResource ::= ENUMERATED {
 unavailableResources(0),
 componentFailure(1),
 basicCallProcessingException(2),
 resourceStatusFailure(3),
 endUserFailure(4)
 }
−− Indicates the network resource that failed.
UserCredit  ::= Credit
UserInteractionModes ::= BIT STRING {
 voiceMessage (0),
 tone (1),
 display (2)
 }
USIInformation  ::= OCTET STRING (SIZE (minUSIInformationLength..maxUSIInformationLength))
−− Indicates the length of the USIInformation element, maxUSIInformationlength will depend on the constraints
−− imposed by the network signalliing used to transport the USI information.
USIMonitorMode ::= ENUMERATED {
 monitoringActive (0),
 monitoringInactive (1)
 }

−−  Indicates if the monitoring relationship for the specified UTSI IE should be activated or deactivated.

USIServiceIndicator  ::= CHOICE{
 global OBJECT IDENTIFIER,
 local OCTET STRING (SIZE (minUSIServiceIndicatorLength..maxUSIServiceIndicatorLength))
 }


VariablePart  ::= CHOICE {
 integer [0] Integer4,
 number [1] Digits,   −− Generic digits
 time [2] OCTET STRING (SIZE(2)), −− HH:MM, BCD coded
 date [3] OCTET STRING (SIZE(3)), −− YYMMDD, BCD coded
 price [4] OCTET STRING (SIZE(4))
 }
−− Indicates the variable part of the message.
−− BCD coded variable parts are encoded as described in the examples below.
−− For example, time = 12:15 would be encoded as:
−− Bits     HGFE   DCBA
−−  leading octet 2 1
−−   5 1
−− date = 1993 September 30th would be encoded as:
−− Bits     HGFE  DCBA
−− leading octet  3 9
```

Annex A: incs2−dt

```
−− 9 0
−− 0 3

−− The Definition of range of constants Follows
highLayerCompatibilityLength INTEGER ::= 2
minCauseLength   INTEGER ::= 1
numOfCounters   INTEGER ::= 100
numOfInfoItems   INTEGER ::= 5
maxCreditUnit    INTEGER ::= 65536

−− Note: The following range constant definitions are not part of EN 03038−1. They are
−−       included to enable simulation of the SDL model (informative).
maxAmount     INTEGER ::= 10000
minAChBillingChargingLength                    INTEGER ::= 1
maxAChBillingChargingLength                    INTEGER ::= 1
minAttributesLength    INTEGER ::= 1
maxAttributesLength    INTEGER ::= 1
minBackwardGVNSLength    INTEGER ::= 1
maxBackwardGVNSLength    INTEGER ::= 1
maxBearerCapabilityLength   INTEGER ::= 2
minCalledPartyNumberLength   INTEGER ::= 1
maxCalledPartyNumberLength   INTEGER ::= 1
minCalledPartyBCDNumberLength   INTEGER ::= 1
maxCalledPartyBCDNumberLength   INTEGER ::= 1
minCallingPartyNumberLength   INTEGER ::= 1
maxCallingPartyNumberLength   INTEGER ::= 1
minCallingPartySubAddressLength   INTEGER ::= 1
maxCallingPartySubAddressLength   INTEGER ::= 1
minCallingPartyBusinessGroupIDLength  INTEGER ::= 1
maxCallingPartyBusinessGroupIDLength  INTEGER ::= 1
minCallResultLength    INTEGER ::= 1
maxCallResultLength    INTEGER ::= 1
minCarrierLength    INTEGER ::= 1
maxCarrierLength    INTEGER ::= 1
maxCauseLength     INTEGER ::= 3
minDigitsLength     INTEGER ::= 1
maxDigitsLength     INTEGER ::= 1
minDisplayInformationLength   INTEGER ::= 1
maxDisplayInformationLength   INTEGER ::= 1
minEventSpecificInformationChargingLength INTEGER ::= 1
maxEventSpecificInformationChargingLength INTEGER ::= 1
minEventTypeChargingLength   INTEGER ::= 1
maxEventTypeChargingLength   INTEGER ::= 1
minFCIBCcs1Length    INTEGER ::= 1
maxFCIBCcs1Length    INTEGER ::= 1
minFCIBCCcs2Length    INTEGER ::= 1
maxFCIBCCcs2Length    INTEGER ::= 1
minFCIBillingChargingLength   INTEGER ::= 1
maxFCIBillingChargingLength   INTEGER ::= 1
minForwardGVNSLength    INTEGER ::= 1
maxForwardGVNSLength    INTEGER ::= 1
minGenericNameLength    INTEGER ::= 1
maxGenericNameLength    INTEGER ::= 1
minGenericNumberLength    INTEGER ::= 1
maxGenericNumberLength    INTEGER ::= 1
maxInitialTimeInterval    INTEGER ::= 1
maxInitialUnitIncrement    INTEGER ::= 1
minIPAvailableLength    INTEGER ::= 1
maxIPAvailableLength    INTEGER ::= 1
minIPSSPCapabilitiesLength   INTEGER ::= 1
maxIPSSPCapabilitiesLength   INTEGER ::= 1
minISDNAccessRelatedInformationLength  INTEGER ::= 1
maxISDNAccessRelatedInformationLength  INTEGER ::= 1
minLocationNumberLength    INTEGER ::= 1
maxLocationNumberLength    INTEGER ::= 1
```

Annex A: incs2−dt

```
minMailBoxIDLength    INTEGER ::= 1
maxMailBoxIDLength    INTEGER ::= 1
minMessageContentLength    INTEGER ::= 1
maxMessageContentLength    INTEGER ::= 1
minMidCallControlInfoNum   INTEGER ::= 1
maxMidCallControlInfoNum   INTEGER ::= 1
minOriginalCalledPartyIDLength   INTEGER ::= 1
maxOriginalCalledPartyIDLength   INTEGER ::= 1
minReasonLength     INTEGER ::= 1
maxReasonLength     INTEGER ::= 1
minReceivedInformationLength   INTEGER ::= 1
maxReceivedInformationLength   INTEGER ::= 1
maxRecordingTime    INTEGER ::= 1
minRedirectingPartyIDLength   INTEGER ::= 1
maxRedirectingPartyIDLength   INTEGER ::= 1
minRegistratorIdentifierLength   INTEGER ::= 1
maxRegistratorIdentifierLength   INTEGER ::= 1
minRequestedUTSINum    INTEGER ::= 1
maxRequestedUTSINum    INTEGER ::= 1
minRouteListLength    INTEGER ::= 1
maxRouteListLength    INTEGER ::= 1
maxScalingFactor    INTEGER ::= 1
minScfAddressLength    INTEGER ::= 1
maxScfAddressLength    INTEGER ::= 1
minScfIDLength     INTEGER ::= 1
maxScfIDLength     INTEGER ::= 1
minSCIBillingChargingLength   INTEGER ::= 1
maxSCIBillingChargingLength   INTEGER ::= 1
maxSegmentsPerDataInterval   INTEGER ::= 1
minServiceInteractionIndicatorsLength  INTEGER ::= 1
maxServiceInteractionIndicatorsLength  INTEGER ::= 1
minSFBillingChargingLength   INTEGER ::= 1
maxSFBillingChargingLength   INTEGER ::= 1
maxTimePerInterval    INTEGER ::= 1
maxUnitsPerDataInterval    INTEGER ::= 1
maxUnitsPerInterval    INTEGER ::= 1
minUSIInformationLength    INTEGER ::= 1
maxUSIInformationLength    INTEGER ::= 1
minUSIServiceIndicatorLength   INTEGER ::= 1
maxUSIServiceIndicatorLength   INTEGER ::= 1

numOfBCSMEvents     INTEGER ::= 28
numOfChargingEvents    INTEGER ::= 1
numOfCSs     INTEGER ::= 10
numOfCSAs     INTEGER ::= 10
numOfExtensions     INTEGER ::= 1
numOfGenericNumbers    INTEGER ::= 1
numOfInServiceCompatibilityIndLength  INTEGER ::= 1
numOfLegs     INTEGER ::= 10
numOfMessageIDs     INTEGER ::= 1
numOfServiceKeys    INTEGER ::= 2

−− Note: Definition of the cause values (informative)
noIndication Cause ::= ’00’H
originationDenied Cause ::= ’01’H
collectInformationFailure Cause ::=  ’02’H
invalidCollectedInformation Cause ::=  ’03’H
authorisationFailure Cause ::=  ’04’H
routeSelectFailure Cause ::= ’05’H
calledPartyBusy Cause ::=  ’06’H
oNoAnswer Cause ::= ’07’H
ss7Failure Cause ::= ’08’H
bptyNoAnswer Cause ::= ’09’H
tException Cause ::= ’0A’H
routeFailure1 Cause ::= ’0B’H
```

Annex A: incs2−dt

```
routeFailure2 Cause ::= ’0C’H
bPtyBusyUDUB Cause ::= ’0D’H
bPtyBusyNDUB Cause ::= ’0E’H
aPtyAbandon Cause ::= ’0F’H
aPtyDisc Cause ::= ’10’H
bPtyDisc Cause ::= ’11’H
terminationDenied Cause ::= ’12’H
presentationFailure Cause ::= ’13’H
calledPartyCallRejected Cause ::= ’14’H
failureOfConnection Cause ::= ’15’H
releaseCall Cause ::= ’16’H
bPtyAlerted Cause ::= ’17’H
accessOutOfOrder Cause ::= ’18’H
accessMaintenanceBusy Cause ::= ’19’H
accessCustomerBusy Cause ::= ’1A’H
pstnLineBusy Cause ::= ’1B’H
isdnLineBusy Cause ::= ’1C’H
trunkGroupBusy Cause ::= ’1D’H
presentCallFailure Cause ::= ’1E’H
collectDigits Cause ::= ’1F’H
notReachable Cause ::= ’20’H

END
```

Annex A: cs2i

```
INCS2Internals

DEFINITIONS IMPLICIT TAGS ::=
BEGIN
IMPORTS
        CalledPartyNumber,
        CallingPartyNumber,
        CallingPartysCategory,
        Cause,
        DestinationRoutingAddress,
        Digits,
        DTLegType,
        EventTypeBCSM,
        ForwardCallIndicators,
 GenericNumbers,
        DTLegID,
        MonitorMode,
 OriginalCalledPartyID,
 RedirectingPartyID,
 RedirectionInformation,
 RouteList,
        ServiceKey,
        USIServiceIndicator,
        USIInformation
FROM INCS2datatypes;

−− DEFINITION OF SIGNALLING CONTROL PRIMITIVES

CallFlag ::= ENUMERATED { −− Note: Used on the signals between BCSMs.
  senderRef(0),
  receiverRef(1)
}

CallRef ::= INTEGER

InfoDirection ::= ENUMERATED {
  utsi(0),
  stui(1)
}

AddressEndType ::= SEQUENCE {
  callRef [0] CallRef,
  digits  [1] Digits OPTIONAL
}

CallProgressType ::= SEQUENCE {
  callRef [0] CallRef,
  cause   [1] Cause
}

FailureType ::= SEQUENCE {
  callRef  [0] CallRef,
  cause    [1] Cause
}

NetworkSRType ::= SEQUENCE { −− Note: Used for IN CS−2
  callRef [0] CallRef
}

ReleaseType ::= SEQUENCE {
  callRef  [0] CallRef,
  cause    [1] Cause
}

ServiceFeatureType ::= SEQUENCE {
```

```
  callRef    [0] CallRef
}

SetupCRType ::= SEQUENCE {
  callRef          [0] CallRef,
  connectedNumber  [1] CalledPartyNumber OPTIONAL
}

SetupIRType ::= SEQUENCE {
  callRef                    [0] CallRef,
  calledPartyNumber          [1] CalledPartyNumber,
  callingPartyNumber         [2] CallingPartyNumber     OPTIONAL,
  callingPartysCategory      [3] CallingPartysCategory  OPTIONAL,
  locationNumber             [4] OCTET STRING           OPTIONAL,
  originalCalledNumber       [5] OriginalCalledPartyID  OPTIONAL,
  userTeleserviceInformation [6] OCTET STRING           OPTIONAL,
  genericNumber              [7] GenericNumbers         OPTIONAL,
  forwardCallIndicators      [8] ForwardCallIndicators  OPTIONAL,
  redirectingNumber          [9] RedirectingPartyID     OPTIONAL,
  redirectingInformation     [10] RedirectionInformation OPTIONAL,
  uSIServiceIndicator        [11] USIServiceIndicator    OPTIONAL,
  uSIInformation             [12] USIInformation         OPTIONAL
}

SubsequentAddressType ::= SEQUENCE {
  callRef [0] CallRef,
  digits  [1] Digits
}

UserDataType ::= SEQUENCE {
  callRef            [0] CallRef,
  uSIServiceIndicator [1] USIServiceIndicator,
  uSIInformation     [2] USIInformation,
  infoDirection      [3] InfoDirection
}


−− DEFINITION OF THE INFORMATIVE MANAGEMENT OPERATION ARGUMENTS

MGTSetTriggerTableArg ::= SEQUENCE OF SEQUENCE {
  legID        [0] DTLegType,
  serviceKey   [1] ServiceKey,
  eventTypeBCSM [2] EventTypeBCSM,
  monitorMode  [3] MonitorMode
}


−− DEFINITION OF TYPES USED INTERNALLY IN THE SDL SPECIFICATION

BCSMStateType ::= ENUMERATED {
    nullState(0),                −− Normal case
    waitOrigAttemptAuthorized(1), −− Used with ICA
    analyseInformation(2),       −− Used with Connect
    waitMidCall(3),              −− Note: Used for IN CS−2 CPH
    selectRoute (4)              −− Note: Used for IN CS−2 CPH
}

PartyType ::= ENUMERATED {
  localParty(0),
  remoteParty(1),
  surrogateParty(2)
}
```

Annex A: cs2i

```
DPArg ::= SEQUENCE {
  legID                    [0] DTLegType,
  bcsmEvent                [1] EventTypeBCSM,
  party                    [2] PartyType              OPTIONAL,
  cgPN                     [3] CallingPartyNumber     OPTIONAL,
  cdPN                     [4] CalledPartyNumber      OPTIONAL,
  callingPartysCategory    [5] CallingPartysCategory  OPTIONAL,
  locationNumber           [6] OCTET STRING           OPTIONAL,
  originalCalledNumber     [7] OriginalCalledPartyID  OPTIONAL,
  userTeleserviceInformation  [8] OCTET STRING        OPTIONAL,
  genericNumber            [9] GenericNumbers         OPTIONAL,
  forwardCallIndicators    [10] ForwardCallIndicators OPTIONAL,
  redirectingNumber        [11] RedirectingPartyID    OPTIONAL,
  redirectingInformation   [12] RedirectionInformation OPTIONAL,
  uSIServiceIndicator      [13] USIServiceIndicator   OPTIONAL,
  uSIInformation           [14] USIInformation        OPTIONAL,
  digits                   [15] Digits                OPTIONAL,
  cause                    [16] Cause                 OPTIONAL
}

DPUTSIArg ::= SEQUENCE {
  legID              [0] DTLegType,
  uSIServiceIndicator [1] USIServiceIndicator,
  uSIInformation     [2] USIInformation,
  party              [3] PartyType
}

OBCSMPars ::= SEQUENCE {
  startState         [0] BCSMStateType,
  legID              [1] DTLegType,
  cgPN               [2] CallingPartyNumber OPTIONAL,
  routingInfo        [3] CHOICE {
    cdPN             [0] CalledPartyNumber,
    routingAddress   [1] DestinationRoutingAddress
  },
  routeList                [4] RouteList              OPTIONAL,
  callingPartysCategory    [5] CallingPartysCategory  OPTIONAL,
  locationNumber           [6] OCTET STRING           OPTIONAL,
  originalCalledNumber     [7] OriginalCalledPartyID  OPTIONAL,
  userTeleserviceInformation  [8] OCTET STRING        OPTIONAL,
  genericNumber            [9] GenericNumbers         OPTIONAL,
  forwardCallIndicators    [10] ForwardCallIndicators OPTIONAL,
  redirectingNumber        [11] RedirectingPartyID    OPTIONAL,
  redirectingInformation   [12] RedirectionInformation OPTIONAL,
  uSIServiceIndicator      [13] USIServiceIndicator   OPTIONAL,
  uSIInformation           [14] USIInformation        OPTIONAL
}

PointInCall ::= ENUMERATED {
  analyseInformation(0),
  authoriseOriginationAttempt(1),
  collectInformation(2),
  collectNDigits(3),
  oActive(4),
  oAlerting(5),
  oException(6),
  oNull(7),
  selectRoute(8),
  sendCall(9),
  authoriseTerminationAttempt(10),
  presentCall(11),
  selectFacility(12),
  tActive(13),
  tAlerting(14),
  tNull(15),
```

Annex A: cs2i

```
  releaseCall(16),          −− Used with the ReleaseCall operation

  disconnectLeg(17),        −− Note: IN CS−2 PIC
  sendSTUI(18),             −− Note: IN CS−2 PIC
  sendUTSI(19),             −− Note: IN CS−2 PIC
  authorizeTermination(20) −− Note: IN CS−2 PIC
}

ChargingEventIndArg ::= SEQUENCE {
  callRef                   [0] CallRef,
  eventTypeCharging         [1] EventTypeCharging
}

PICArg ::= SEQUENCE {
  legID                     [0] DTLegType,
  pic                       [1] PointInCall,
  routingInfo               [2] CHOICE {
    cdPN          [0] CalledPartyNumber,
    routingAddress [1] DestinationRoutingAddress
  },
  routeList                 [3] RouteList              OPTIONAL,
  callingPartyNumber        [4] CallingPartyNumber     OPTIONAL,
  callingPartysCategory     [5] CallingPartysCategory  OPTIONAL,
  locationNumber            [6] OCTET STRING           OPTIONAL,
  originalCalledNumber      [7] OriginalCalledPartyID  OPTIONAL,
  userTeleserviceInformation  [8] OCTET STRING         OPTIONAL,
  genericNumber             [9] GenericNumbers         OPTIONAL,
  forwardCallIndicators     [10] ForwardCallIndicators OPTIONAL,
  redirectingNumber         [11] RedirectingPartyID    OPTIONAL,
  redirectingInformation    [12] RedirectionInformation OPTIONAL,
  uSIServiceIndicator       [13] USIServiceIndicator   OPTIONAL,
  uSIInformation            [14] USIInformation        OPTIONAL,
  infoDirection             [15] InfoDirection         OPTIONAL,
  party                     [16] PartyType             OPTIONAL,
  cause                     [17] Cause                 OPTIONAL
}

END
```

Annex A: ssf−scf

```
INCS2opsargs
−− ++++++++++++++++++++++++++++++++++
−− Date  : 29.09.1998
−− Change: Merge SSF−SCF and SCF−SRF ASN.1 files into this one to get around a SDT problem which
−−        does not allow to do import types etc. from more then one file
−−        − Type Code has been added for uiScriptId  to be used
−− ++++++++++++++++++++++++++++++++++
−−INCS2SSFSCFopsargs { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) cs−2(20) modules(0) in−cs2−ssf−sc
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
IMPORTS
−− errortypes, datatypes, operationcodes, classes, ros−InformationObjects
−−FROM IN−CS2−object−identifiers
−− { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) CS−2(20) modules(0) in−cs2−object−identifiers(17) versio
−− OPERATION
−−FROM Remote−Operations−Information−Objects ros−InformationObjects

−−tc−Messages, classes FROM  IN−CS2−object−identifiers
−−  { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) CS−2(20) module(0) in−cs2−object−identifiers(17) versio
−− InvokeIdType
−−FROM TCAPMessages tc−Messages

−− AddOnChargingInformation,
−− ChargingTariffInformation,
−− ChargingMessageType

−−FROM Tariffing−Data Types {itu−t(0) identified−organisation etsi (0) 1296 version1(1)}


−− IMSI,
−− Ext−BasicServiceCode

−−FROM MAP−CommonDataTypes { ccitt(0) identified−organization(4) etsi(0) mobileDomain(0) gsm−Network(1) modules(3) map

−− LocationInformation,
−− SubscriberState

−−FROM MAP−MS−DataTypes { ccitt(0) identified−organization(4) etsi(0) mobileDomain(0) gsm−Network(1) modules(3) map−MS

−− CallReferenceNumber,
−− SuppressionOfAnnouncement

−−FROM MAP−CH−DataTypes { ccitt(0) identified−organization(4) etsi(0) mobileDomain(0) gsm−Network(1) modules(3) map−CH

−− ISDN−AddressString

−−FROM MAP−CommonDataTypes { ccitt identified−organization(4) etsi(0) mobileDomain(0) gsm−Network (1) modules (3) map−(


−−  PARAMETERS−BOUND
−−FROM IN−CS2−classes classes
−− opcode−activateServiceFiltering,
−− opcode−activityTest,
−− opcode−applyCharging,
−− opcode−applyChargingReport,
−− opcode−assistRequestInstructions,
−− opcode−callGap,
−− opcode−callInformationReport,
−− opcode−callInformationRequest,
−− opcode−cancel,
−− opcode−collectInformation,
−− opcode−connect,
−− opcode−connectToResource,
−− opcode−continue,
−− opcode−continueWithArgument,
```

Annex A: ssf−scf

```
−− opcode−createCallSegmentAssociation,
−− opcode−disconnectForwardConnection,
−− opcode−dFCWithArgument,
−− opcode−disconnectLeg,
−− opcode−entityReleased,
−− opcode−establishTemporaryConnection,
−− opcode−eventNotificationCharging,
−− opcode−eventReportBCSM,
−− opcode−furnishChargingInformation,
−− opcode−initialDP,
−− opcode−initiateCallAttempt,
−− opcode−manageTriggerData,
−− opcode−mergeCallSegments,
−− opcode−moveCallSegments,
−− opcode−moveLeg,
−− opcode−releaseCall,
−− opcode−reportUTSI,
−− opcode−requestNotificationChargingEvent,
−− opcode−requestReportBCSMEvent,
−− opcode−requestReportUTSI,
−− opcode−resetTimer,
−− opcode−sendChargingInformation,
−− opcode−sendSTUI,
−− opcode−serviceFilteringResponse,
−− opcode−splitLeg
−−FROM IN−CS2−operationcodes operationcodes
 ActionIndicator,
 ActionPerformed,
 AChBillingChargingCharacteristics,
 AdditionalCallingPartyNumber,
 AlertingPattern,
 AssistingSSPIPRoutingAddress,
 BackwardGVNS,
 BCSMEvent ,
 BearerCapability,
 CalledPartyNumber,
 CallingPartyBusinessGroupID,
 CallingPartyNumber,
 CallingPartysCategory,
 CallingPartySubaddress,
 CallResult,
 CallSegmentID,
 Carrier,
 Cause,
 CGEncountered,
 ChargingEvent,
 ControlType,
 CorrelationID,
 CountersValue,
 CSAID,
 CutAndPaste,
 DateAndTime,
 DestinationRoutingAddress,
 Digits,
 DisplayInformation,
 EventSpecificInformationBCSM,
 EventSpecificInformationCharging,
 EventTypeBCSM,
 EventTypeCharging,
 ExtensionField,
 FCIBillingChargingCharacteristics,
 FilteredCallTreatment,
 FilteringCharacteristics,
 FilteringCriteria,
 FilteringTimeOut,
```

```
ForwardCallIndicators,
ForwardGVNS,
GapCriteria,
GapIndicators,
GapTreatment,
GenericName,
GenericNumbers,
HighLayerCompatibility,
initialCallSegment,
INServiceCompatibilityIndication,
INServiceCompatibilityResponse,
Integer4,
InvokeID,
IPAvailable,
IPRoutingAddress,
IPSSPCapabilities,
ISDNAccessRelatedInformation,
LegID,
leg1,
LocationNumber,
MiscCallInfo,
MonitorMode,
NumberingPlan,
OriginalCalledPartyID,
Reason,
RedirectingPartyID,
RedirectionInformation,
RegistratorIdentifier,
RequestedInformationList,
RequestedInformationTypeList,
RequestedUTSIList,
ResponseCondition,
RouteList,
ScfID,
SCIBillingChargingCharacteristics,
ServiceInteractionIndicators,
ServiceInteractionIndicatorsTwo,
ServiceKey,
TerminalType,
TimerID,
TimerValue,
TriggerDataIdentifier,
USIInformation,
USIServiceIndicator,
numOfCSs,
numOfExtensions,
numOfLegs,
numOfChargingEvents,
numOfBCSMEvents ,
 −− Imports for SCF/SRF
 CallSegmentID,
CollectedInfo,
−− Digits,
−− ExtensionField,
InformationToRecord,
InformationToSend,
−− LegID,
MailBoxID,
Media,
GenericNumber,
ReceivedStatus,
RecordedMessageID

FROM INCS2datatypes datatypes
−− cancelFailed,
```

```
−− eTCFailed,
−− improperCallerResponse,
−− missingCustomerRecord,
−− missingParameter,
−− parameterOutOfRange,
−− requestedInfoError,
−− systemFailure,
−− taskRefused,
−− unavailableResource,
−− unexpectedComponentSequence,
−− unexpectedDataValue,
−− unexpectedParameter,
−− unknownLegID,
−− unknownResource
−−FROM IN−CS2−errortypes errortypes
;

−− *****
−− Definitions of MAP and ISUP types imported above.

CallReferenceNumber ::= INTEGER

ChargingMessageType ::= INTEGER

Ext−BasicServiceCode ::= CHOICE { −− Note: Ext−BasicServiceCode is not defined in MAP!
 bearerService [2] OCTET STRING (SIZE(1)),
 teleService [3] OCTET STRING
 }

IMSI ::= OCTET STRING (SIZE (3..8))

ISDN_AddressString ::= OCTET STRING

LocationInformation ::= OCTET STRING

OCSIApplicable ::= BOOLEAN

SubscriberState ::= INTEGER

SuppressionOfAnnouncement ::= BOOLEAN

−− *****
−− TYPE DEFINITIONS FOR IN CS−2 ERRORS (Informative).

CancelFailed ::= SEQUENCE {
   problem [0] ENUMERATED {
    unknownOperation(0),
    tooLate(1),
    operationNotCancellable(2)
    },
   operation [1] InvokeID
   }

−− The parameter was not as expected (e.g. missing or out of range).

RequestedInfoError  ::= ENUMERATED {
   unknownRequestedInfo(1),
   requestedInfoNotAvailable(2)
   −− other values FFS
   }

−− The requested information cannot be found.

SystemFailure ::= UnavailableNetworkResource
```

```
−− The operation could not be completed due to a system failure

TaskRefused::=  ENUMERATED {
   generic(0),
   unobtainable (1),
   congestion(2)
   −−other values FFS
   }
−− An entity normally capable of the task requested cannot or chooses not
−− to perform the task at this
−− time.  This includes error situations like congestion and unobtainable
−− address as used in e.g. the Connect operation.
−− *****

−− *****
−− Due to a restriction in SDT it is not possible to denote values of types defined in an inmported
−− module (for example in DEFAULT statements). Therefore the types for which DEFAULT statements are
−− used in this module, are redefined below. The corresponding type definitions in IN−CS2−Datatypes
−− are changed to DTxxx.

CallSegmentID   ::= INTEGER (1..numOfCSs)
initialCallSegment INTEGER ::= 1

LegID ::= CHOICE {
 sendingSideID [0] LegType,
 receivingSideID [1] LegType
 }

LegType ::= OCTET STRING (SIZE(1))
leg1 LegType ::= ’01’H
leg2 LegType ::= ’02’H

MiscCallInfo ::= SEQUENCE {
 messageType [0] ENUMERATED {
    request(0),
    notification(1)
    }
 }

TimerID ::= ENUMERATED {
 tssf(0),
 tcusf(1)
 −− others ffs
 }

−− *****


−− Direction: SCF −> SSF, Timer: Tasf
−− When receiving this operation, the SSF handles calls to destination in a specified manner
−− without  sending queries for every detected call. It is used for example for providing
−− televoting or mass calling services. Simple registration functionality (counters) and
−− announcement control may be  located at the SSF. The operation initializes the specified
−− counters in the SSF.

ActivateServiceFilteringArg ::= SEQUENCE {
 filteredCallTreatment [0] FilteredCallTreatment,
 filteringCharacteristics [1] FilteringCharacteristics,
 filteringTimeOut [2] FilteringTimeOut ,
 filteringCriteria [3] FilteringCriteria,
 startTime [4] DateAndTime OPTIONAL,
 extensions [5] SEQUENCE SIZE(1..numOfExtensions) OF
        ExtensionField  OPTIONAL
 }
```

Annex A: ssf−scf

```
−− Direction: SCF −> SSF, Timer: Tat
−− This operation is used to check for the continued existence of a relationship between the SCF
−− and SSF. If the relationship is still in existence, then the SSF will respond. If no reply is
−− received, then the SCF will assume that the SSF has failed in some way and will take the
−− appropriate action.

−− Direction: SCF −>  SSF, Timer: Tac
−− This operation is used  for  interacting  from  the SCF with the SSF charging mechanisms. The ApplyChargingReport
−− operation provides the  feedback from the SSF to  the SCF.

ApplyChargingArg ::= SEQUENCE {
 aChBillingChargingCharacteristics [0] AChBillingChargingCharacteristics,
 sendCalculationToSCPIndication   [1] BOOLEAN OPTIONAL,
 partyToCharge     [2] LegID OPTIONAL,
 extensions     [3] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField  OPTIONAL
 }

−− The partyToCharge parameter indicates the party in the call to which the ApplyCharging operation
−− should be applied.


−− Direction: SSF −>  SCF, Timer: Tacr
−− This operation is used  by the SSF to  report to the SCF the occurrence of a specific charging event
−− as requested
−− by the SCF using the ApplyCharging operation.

ApplyChargingReportArg ::= CallResult
−− Note: When the SSF sends the ApplyChargingReport operation as the last event from the Call Segment, the
−−    lastEventIndicator parameter is needed for indicating whether
−−    the event is last to the SCF.  However, because there is no consideration for the parameter expansion in the
−−   CS−1, this parameter cannot be added.  There are two alternatives for the solution.  One is to be included
−−    into the CallResult parameter.  And the other is to specify a new operation with this parameter.  The latter is
−−    ffs.


−− Direction: SSF  −> SCF or SRF  −> SCF, Timer: Tari
−− This operation is used when there is an assist or a hand−off procedure and may be sent by the SSF
−− or SRF to the SCF. This operation is sent by the assisting SSF to SCF, when the initiating SSF has
−− set up a connection to the SRF or to the assisting SSF as a result of receiving an EstablishTemporaryConnection
−− or Connect operation (in the case of hand−off) from the SCF.
−− Refer to clause 17 for a description of the procedures associated with this operation.
AssistRequestInstructionsArg ::= SEQUENCE {
 correlationID   [0] CorrelationID,
 iPAvailable   [1] IPAvailable OPTIONAL,
 iPSSPCapabilities [2] IPSSPCapabilities OPTIONAL,
 extensions   [3] SEQUENCE SIZE(1..numOfExtensions) OF  ExtensionField  OPTIONAL
 }
−− OPTIONAL denotes network operator specific use. The value of the correlationID may be the
−− Called Party Number supplied by the initiating SSF.

−− Direction: SCF −>  SSF, Timer: Tcg
−− This operation is used to request the SSF  to reduce the rate at which specific service requests are sent to
−− the  SCF..

CallGapArg ::= SEQUENCE {
 gapCriteria [0] GapCriteria,
 gapIndicators [1] GapIndicators,
 controlType [2] ControlType OPTIONAL,
 gapTreatment [3] GapTreatment OPTIONAL,
 extensions [4] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL
 }
−− OPTIONAL denotes network operator optional. If gapTreatment is not present, the SSF will use
−− a default treatment depending on network operator implementation.

−− Direction: SSF −>  SCF, Timer: Tcirp
```

Annex A: ssf−scf

```
−− This operation is used to send specific call information for a single call to the SCF as requested by the SCF
−− in a previous CallInformationRequest.

CallInformationReportArg ::= SEQUENCE {
 requestedInformationList [0] RequestedInformationList,
 extensions   [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 legID    [3] LegID OPTIONAL
 }
−− OPTIONAL denotes network operator optional.

−− Direction: SCF −>  SSF, Timer: Tcirq
−− This operation is used to request the SSF to  record specific information about a single call and report it to
−− the SCF (with a CallInformationReport operation).

CallInformationRequestArg ::= SEQUENCE {
 requestedInformationTypeList [0] RequestedInformationTypeList,
 extensions   [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 legID    [3] LegID OPTIONAL
 }
−− OPTIONAL denotes network operator optional.

−− Direction: SCF −>  SSF, or SCF  −> SRF, Timer:  Tcan
−− This operation cancels the correlated previous operation or all previous requests. The following operations can be
−− canceled: PlayAnnouncement, PromptAndCollectUserInformation, PromptAndReceiveMessage.
CancelArg ::= CHOICE {
 invokeID  [0] InvokeID,
 allRequests  [1] NULL,
 callSegmentToCancel [2] SEQUENCE {
   invokeID [0] InvokeID,
   callSegmentID [1] CallSegmentID
   }
 }
−− The InvokeID has the same value as that which was used for the operation to be cancelled.

−− Direction: SCF −>  SSF, Timer: Tci
−− This operation is used to request the SSF to perform the originating basic call processing actions to prompt
−− a calling party for destination information, then collect destination information according to a specified
−− numbering plan (e.g. for virtual private networks).

CollectInformationArg ::= SEQUENCE {
 extensions [4] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField  OPTIONAL
 }



−− Direction: SCF −>  SSF, Timer: Tcon
−− This operation is used to request the SSF to perform the call processing actions to route or forward a call to
−− a specified destination. To do so, the SSF may or may not use destination information from the calling party
−− (e.g. dialed digits) and existing call setup information (e.g. route index to a list of trunk groups), depending on
−− the information provided by the SCF.
−−  − When address information is only included in the Connect operation, call processing resumes at the
−−    Analyzed_Information PIC in  the O−BCSM.
−−  − When address information and routing information is included, call processing resumes at the
−−    Select_Route PIC.

ConnectArg ::= SEQUENCE {
 destinationRoutingAddress [0] DestinationRoutingAddress,
 alertingPattern   [1] AlertingPattern   OPTIONAL,
 correlationID   [2] CorrelationID   OPTIONAL,
 cutAndPaste   [3] CutAndPaste    OPTIONAL,
 iSDNAccessRelatedInformation [5] ISDNAccessRelatedInformation OPTIONAL,
 originalCalledPartyID  [6] OriginalCalledPartyID  OPTIONAL,
 routeList   [7] RouteList    OPTIONAL,
 scfID    [8] ScfID    OPTIONAL,
 extensions   [10] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
```

```
 carrier    [11] Carrier    OPTIONAL,
 serviceInteractionIndicators [26] ServiceInteractionIndicators OPTIONAL,
 callingPartyNumber  [27] CallingPartyNumber   OPTIONAL,
 callingPartysCategory  [28] CallingPartysCategory  OPTIONAL,
 redirectingPartyID  [29] RedirectingPartyID   OPTIONAL,
 redirectionInformation  [30] RedirectionInformation  OPTIONAL,
 displayInformation  [12] DisplayInformation   OPTIONAL,
 forwardCallIndicators  [13] ForwardCallIndicators  OPTIONAL,
 genericNumbers   [14] GenericNumbers   OPTIONAL,
 serviceInteractionIndicatorsTwo [15] ServiceInteractionIndicatorsTwo OPTIONAL,
 iNServiceCompatibilityResponse [16] INServiceCompatibilityResponse OPTIONAL,
 forwardGVNS   [17] ForwardGVNS   OPTIONAL,
 backwardGVNS   [18] BackwardGVNS   OPTIONAL,
 callSegmentID   [20] CallSegmentID   OPTIONAL,
 legToBeCreated   [21] LegID    OPTIONAL,
 locationNumber   [50] LocationNumber   OPTIONAL,
 bearerCapability  [51] BearerCapability   OPTIONAL,
 suppressionOfAnnouncement [55] SuppressionOfAnnouncement  OPTIONAL,
 oCSIApplicable   [56] OCSIApplicable   OPTIONAL
 }



-- Direction: SCF -> SSF, Timer: Tctr
-- This operation is used to connect a call from the SSP to the physical entity containing the SRF.
-- Refer to clause 17 for a description of the procedures associated with this operation.

ConnectToResourceArg ::= SEQUENCE {
 resourceAddress CHOICE {
  ipRoutingAddress [0] IPRoutingAddress,
  legID   [1] LegID,
  ipAddressAndLegID [2] SEQUENCE {
    ipRoutingAddress [0] IPRoutingAddress,
    legID   [1] LegID
    },
  xnone   [3] NULL,
  callSegmentID  [5] CallSegmentID ,
  ipAddressAndCallSegment [6] SEQUENCE {
    ipRoutingAddress [0] IPRoutingAddress,
    callSegmentID  [1] CallSegmentID
    }
  },
 extensions   [4] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 serviceInteractionIndicators [30] ServiceInteractionIndicators OPTIONAL,
 serviceInteractionIndicatorsTwo [7] ServiceInteractionIndicatorsTwo OPTIONAL
 }


-- Direction: SCF -> SSF, Timer: Tcue
-- This operation is used to request the SSF to proceed with call processing at the DP at which it
-- previously suspended call processing to await SCF instructions (i.e. proceed to the next point
-- in call in the BCSM). The SSF continues call processing without substituting new data from SCF.
-- This operation is not valid for a single call segment CSA with more than 2 legs or a multi call segment CSA


-- Direction: SCF -> SSF, Timer: Tcwa
-- This operation is used to request the SSF to proceed with call processing at the DP a which it previously
-- suspended call processing to await SCF instructions.
-- It is also used to provide additional service related information to a User (Called Party or Calling Party) whilst
-- the call processing proceeds.

ContinueWithArgumentArg ::= SEQUENCE {
 legorCSID CHOICE {
  legID [0]  LegID,
  csID [9] CallSegmentID}, -- DEFAULT legID:sendingSideID:leg1,
```

```
 alertingPattern   [1] AlertingPattern   OPTIONAL,
 genericName   [2] GenericName    OPTIONAL,
 iNServiceCompatibilityResponse [3] INServiceCompatibilityResponse OPTIONAL,
 forwardGVNS   [4] ForwardGVNS    OPTIONAL,
 backwardGVNS   [5] BackwardGVNS   OPTIONAL,
 extensions   [6] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL,
 serviceInteractionIndicatorsTwo [7] ServiceInteractionIndicatorsTwo OPTIONAL,
 locationNumber   [51] LocationNumber   OPTIONAL
 }


−− Direction SCF −> SSF, Timer Tcsa
−− This operation is used to create a new CSA. The new CSA will not contain any Call Segments after creation.
−− The SSF is responsible for specifying a new CSA identifier for the created CSA which is unique within
−−  the SSF.
CreateCallSegmentAssociationArg::= SEQUENCE {
 extensions [0] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }

CreateCallSegmentAssociationResultArg ::= SEQUENCE {
 newCallSegmentAssociation [0] CSAID,
 extensions   [1] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }


−− Direction: SCF −>  SSF, Timer: Tdfc
−− This operation is used to disconnect a forward temporary connection or a connection to a resource.
−− Refer to clause 17 for a description of the procedures associated with this operation.
−− This operation is not valid for a multi call segment CSA.


−− Direction: SCF −>  SSF, Timer: Tdfcwa
−− This operation is used to disconnect a forward temporary connection or a connection to a resource.
−− Refer to clause 17 for a description of the procedures associated with this operation.

DisconnectForwardConnectionWithArgumentArg ::= SEQUENCE {
 partyToDisconnect CHOICE {
   legID  [0] LegID,
   callSegmentID [1] CallSegmentID
   },
 extensions [2] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }



−− Direction: SCF −> SSF. Timer: T dl
−− This operation  is issued by the SCF  to release a specific leg associated with the call and retain any
−− other legs not specified in the DisconnectLeg. Any leg may be disconnected, including the controlling
−− leg, without completely releasing all legs.
−− Refer to clause 17 for a description of the procedures associated with this operation.

DisconnectLegArg ::= SEQUENCE {
 legToBeReleased [0] LegID,
 releaseCause [1] Cause OPTIONAL,
 extensions [2] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }



−− Direction SSF −> SCF, Timer: Ter
−− This operation is used by SSF to inform the SCF of an error/exception

EntityReleasedArg  ::= CHOICE {
```

```
 cSFailure [0] SEQUENCE{
   callSegmentID [0] CallSegmentID,
   reason  [1] Reason OPTIONAL,
   cause  [2] Cause OPTIONAL
   },
 bCSMFailure [1] SEQUENCE{
   legID  [0] LegID,
   reason  [1] Reason OPTIONAL,
   cause  [2] Cause OPTIONAL
   }
 }

−− Direction: SCF −>  SSF, Timer: Tetc
−− This operation is used to create a connection to a resource for a limited period of time
−− (e.g. to play an announcement, to collect user information); it implies the use of the assist
−− procedure. Refer to clause 17 for a description of the procedures associated with this operation.

EstablishTemporaryConnectionArg ::= SEQUENCE {
 assistingSSPIPRoutingAddress [0] AssistingSSPIPRoutingAddress,
 correlationID   [1] CorrelationID   OPTIONAL,
 partyToConnect CHOICE {
  legID  [2] LegID,
  callSegmentID [7] CallSegmentID
   }       OPTIONAL,
 scfID    [3] ScfID    OPTIONAL,
 extensions   [4] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 serviceInteractionIndicators [30] ServiceInteractionIndicators OPTIONAL,
 serviceInteractionIndicatorsTwo [6] ServiceInteractionIndicatorsTwo OPTIONAL
 }


−− Direction: SSF −>  SCF, Timer: Tenc
−− This operation is used  by the SSF to  report to the SCF  the occurence of a specific charging event
−− type as  previously requested  by the SCF in a RequestNotificationChargingEvent operation.

EventNotificationChargingArg ::= SEQUENCE {
 eventTypeCharging   [0]  EventTypeCharging,
 eventSpecificInformationCharging [1]  EventSpecificInformationCharging OPTIONAL,
 legID     [2]  LegID    OPTIONAL,
 extensions    [3]  SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 monitorMode    [30] MonitorMode, −− DEFAULT notifyAndContinue
 eventTypeTariff   [50]  EventTypeTariff   OPTIONAL,
 eventSpecificInformationTariff [51]  ChargingMessageType   OPTIONAL
 }
−− OPTIONAL denotes network operator specific use.


−− Direction: SSF  −> SCF, Timer: Terb
−− This operation is used to notify the SCF of a call−related event (e.g. BCSM events such as busy or
−− no answer) previously requested by the SCF in a RequestReportBCSMEvent operation.

EventReportBCSMArg ::= SEQUENCE {
 eventTypeBCSM   [0] EventTypeBCSM,
 eventSpecificInformationBCSM [2] EventSpecificInformationBCSM OPTIONAL,
 legID    [3] LegID    OPTIONAL,
 miscCallInfo   [4] MiscCallInfo DEFAULT {messageType request},
 extensions   [5] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL
 }



−− Direction: SCF −>  SSF, Timer: Tfci
−− This operation is used to request the SSF  to generate,  register a call record or to include some information
−− in the default call
−− record. The registered call record is intended for off line charging of the call.
```

Annex A: ssf−scf

```
FurnishChargingInformationArg ::= FCIBillingChargingCharacteristics


−− Direction: SSF −> SCF, Timer: Tidp
−− This operation is used after a TDP to indicate request for service.

InitialDPArg ::= SEQUENCE {
 serviceKey   [0] ServiceKey,
 calledPartyNumber  [2] CalledPartyNumber   OPTIONAL,
 callingPartyNumber  [3] CallingPartyNumber   OPTIONAL,
 callingPartyBusinessGroupID [4] CallingPartyBusinessGroupID  OPTIONAL,
 callingPartysCategory  [5] CallingPartysCategory  OPTIONAL,
 cGEncountered   [7] CGEncountered   OPTIONAL,
 iPSSPCapabilities  [8] IPSSPCapabilities   OPTIONAL,
 iPAvailable   [9] IPAvailable    OPTIONAL,
 locationNumber   [10] LocationNumber   OPTIONAL,
 originalCalledPartyID  [12] OriginalCalledPartyID  OPTIONAL,
 terminalType   [14] TerminalType   OPTIONAL,
 extensions   [15] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 highLayerCompatibility  [23] HighLayerCompatibility  OPTIONAL,
 serviceInteractionIndicators [24] ServiceInteractionIndicators OPTIONAL,
 additionalCallingPartyNumber [25] AdditionalCallingPartyNumber OPTIONAL,
 forwardCallIndicators  [26] ForwardCallIndicators  OPTIONAL,
 bearerCapability  [27] BearerCapability   OPTIONAL,
 eventTypeBCSM   [28] EventTypeBCSM   OPTIONAL,
 redirectingPartyID  [29] RedirectingPartyID   OPTIONAL,
 redirectionInformation  [30] RedirectionInformation  OPTIONAL,
 cause    [17] Cause    OPTIONAL,
 iSDNAccessRelatedInformation [21] ISDNAccessRelatedInformation OPTIONAL,
 iNServiceCompatibilityIndication [22] INServiceCompatibilityIndication OPTIONAL,
 genericNumbers   [31] GenericNumbers   OPTIONAL,
 serviceInteractionIndicatorsTwo [32] ServiceInteractionIndicatorsTwo OPTIONAL,
 forwardGVNS   [33] ForwardGVNS   OPTIONAL,
 createdCallSegmentAssociation [34] CSAID    OPTIONAL,
 uSIServiceIndicator  [35] USIServiceIndicator  OPTIONAL,
 uSIInformation   [36] USIInformation   OPTIONAL,
 carrier    [37] Carrier    OPTIONAL,
 iMSI    [50] IMSI    OPTIONAL,
 subscriberState   [51] SubscriberState   OPTIONAL,
 locationInformation  [52] LocationInformation  OPTIONAL,
 ext−basicServiceCode  [53] Ext−BasicServiceCode  OPTIONAL,
 callReferenceNumber  [54] CallReferenceNumber  OPTIONAL,
 mscAddress   [55] ISDN−AddressString   OPTIONAL,
 calledPartyBCDNumber  [56] CalledPartyBCDNumber  OPTIONAL
 }


−− OPTIONAL for iPSSPCapabilities, iPAvailable, cGEncountered, and miscCallInfo denotes network
−− operator specific use.
−− OPTIONAL for callingPartyNumber, and callingPartysCategory refer to clause 17 for the trigger
−− detection point processing rules to specify when these parameters are included in the message.


−− Direction: SCF −> SSF, Timer: Tica
−− This operation is used to request the SSF to create a new call to one call  party using address
−− information provided by the SCF.

InitiateCallAttemptArg ::= SEQUENCE {
 destinationRoutingAddress [0] DestinationRoutingAddress,
 alertingPattern   [1] AlertingPattern   OPTIONAL,
 iSDNAccessRelatedInformation [2] ISDNAccessRelatedInformation OPTIONAL,
 extensions   [4] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 serviceInteractionIndicators [29] ServiceInteractionIndicators OPTIONAL,
 callingPartyNumber  [30] CallingPartyNumber   OPTIONAL,
 legToBeCreated   [5] LegID     OPTIONAL,−− DEFAULT  sendingSideID:’02’H,
```

```
 newCallSegment   [6] CallSegmentID    OPTIONAL,−− DEFAULT 1,
 iNServiceCompatibilityResponse [7] INServiceCompatibilityResponse OPTIONAL,
 serviceInteractionIndicatorsTwo [8] ServiceInteractionIndicatorsTwo OPTIONAL,
 carrier    [50] Carrier    OPTIONAL,
 locationNumber   [51] LocationNumber   OPTIONAL,
 bearerCapability  [52] BearerCapability   OPTIONAL
 }


−− Direction: SCF  −>  SSF,  Class 1, Timer: Tmtd
−−  This operation is used to activate, deactivate or retrieve
−− the status of a trigger detection point linked to a subscriber profile known at the switch, e.g. related to an access line.
ManageTriggerDataArg  ::= SEQUENCE {
 actionIndicator [0] ActionIndicator,
 triggerDataIdentifier [1] TriggerDataIdentifier,
 registratorIdentifier  [2] RegistratorIdentifier  OPTIONAL,
 extensions [3] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }


ManageTriggerDataResultArg  ::= SEQUENCE {
 actionPerformed [0]  ActionPerformed,
 extensions [1] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }



−− Direction: SCF −> SSF. Timer: T mc
−− This operation  is issued by the SCF  to merge two associated CSs with a single controlling leg into one
−− CS with that controlling leg.
−− For additional information on this operation, refer to Q.1224.

MergeCallSegmentsArg ::= SEQUENCE {
 sourceCallSegment [0] CallSegmentID,
 targetCallSegment [1] CallSegmentID DEFAULT initialCallSegment,
 extensions  [2] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }


−−  Direction: SCF −> SSF, Timer Tmcs
−−This operation moves a CS from the source CSA to the target CSA
ExportLegsType ::= SEQUENCE SIZE (1..numOfLegs) OF SEQUENCE {
  sourceLeg [0] LegID,
  newLeg    [1] LegID
}

MoveCallSegmentsArg ::= SEQUENCE {
 targetCallSegmentAssociation [0] CSAID,
 callSegments   [1] SEQUENCE SIZE (1..numOfCSs) OF SEQUENCE {
  sourceCallSegment  [0] CallSegmentID DEFAULT initialCallSegment,
  newCallSegment   [1] CallSegmentID
  },
 legs    [2] ExportLegsType,
 extensions   [2] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }
−− The double use of the tag value 2 is from pure ASN.1 point of view correct. From the way the tagging is used within
−− INAP however it is incorrect(sequential tagging is used). As the possibility that ITU−T adapts its tagging
−− is rather small no adaptation will be done here (yet).

−− Direction : SCF  −>SSF, Timer: T ml
−− This operation is issued by the SCF to move a leg from one CS to another with which it is associated.
MoveLegArg ::=SEQUENCE {
 legIDToMove  [0] LegID,
 targetCallSegment [1] CallSegmentID DEFAULT 1,
 extensions  [2] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
```

```
 }
−− For the OPTIONAL parameters, refer to clause 17 for the trigger
−− detection point processing rules to specify when these parameters are
−− included in the message.


−− Direction: SCF −>  SSF, Timer: Trc
−− This operation is used to tear down an existing call at any phase of the call for all parties
−− involved in the call.
ReleaseCallArg ::= CHOICE {
 initialCallSegment Cause,
 associatedCallSegment [1] SEQUENCE {
  callSegment [0] INTEGER (2..numOfCSs),
  releaseCause [1] Cause  OPTIONAL
  },
 allCallSegments [2] SEQUENCE {
   releaseCause [0] Cause  OPTIONAL
  }
 }
−− A default value of decimal 31 (normal unspecified) should be coded appropriately.



−− Direction: SSF −> SCF. Timer: Tru
−− This operation is issued by the SSF in the context of the USI feature.  It is used to report the receipt
−− of a User to Service Information (UTSI) IE to the SCF.

ReportUTSIArg ::= SEQUENCE {
 uSIServiceIndicator [0] USIServiceIndicator,
 legID   [1] LegID DEFAULT receivingSideID:leg1,
 uSIInformation  [2] USIInformation,
 extensions  [3] SEQUENCE SIZE(1..numOfExtensions)  OF ExtensionField OPTIONAL
 }


−− Direction: SCF−>SSF, Timer: Trnc
−− This operation  is used by the SCF to instruct the SSF on how to manage the charging events
−− which are received  rom other FE's and not under control of the service logic instance.
RequestNotificationChargingEventArg ::= SEQUENCE SIZE(1..numOfChargingEvents) OF ChargingEvent

−− Direction: SCF −>  SSF, Timer: Trrb
−− This operation is used to request the SSF to monitor for a call−related event (e.g. BCSM events such as
−− busy or no answer), then send a notification back to the SCF when the event is detected.
−− NOTE: −
−− − Every EDP must be explicitly armed by the SCF via a RequestReportBCSMEvent operation. No
−−    implicit arming of EDPs at the SSF after reception of any operation (different from
−−    RequestReportBCSMEvent) from the SCF is allowed.

RequestReportBCSMEventArg ::= SEQUENCE {
 bcsmEvents [0] SEQUENCE SIZE(1..numOfBCSMEvents) OF BCSMEvent,
 extensions [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField  OPTIONAL
 }
−− Indicates the BCSM related events for notification.


−− Direction: SCF −> SSF. Timer: Trru
−− This operation is issued by the SCF in the context of the USI feature to request the SSF to monitor for
−− a User to Service Information (UTSI) information element, which are received from a user.
RequestReportUTSIArg ::= SEQUENCE {
 requestedUTSIList [0] RequestedUTSIList,
 extensions  [1] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 legID   [2] LegID DEFAULT sendingSideID:leg1
 }
```

Annex A: ssf−scf

```
−− Direction: SCF−>SSF, Timer: Trt
−− This operation is used to request the SSF to refresh an application timer in the SSF.

ResetTimerArg ::= SEQUENCE {
 timerID  [0] TimerID DEFAULT tssf,
 timervalue [1] TimerValue,
 extensions [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 callSegmentID [3] CallSegmentID OPTIONAL
 }

−− Direction: SCF −>  SSF, Timer: Tsci
−− This operation is used to  instruct the SSF on the charging information to send by the SSF.
−− The charging information can either be sent back by means of signalling or internal
−− if  the SSF is located in the local exchange. In the local exchange
−− this information may be used to update the charge meter or to create a standard call record.

SendChargingInformationArg ::= SEQUENCE {
 sCIBillingChargingCharacteristics [0] SCIBillingChargingCharacteristics,
 partyToCharge [1] LegID,
 extensions [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 tariffMessage [3] ChargingMessageType OPTIONAL
 }


−− Direction: SCF −> SSF. Timer: Tss
−− This operation is issued by the SCF in the context of the USI feature. It is used to request the SSF
−− to send a Service to User Information (STUI) information element to the indicated user.

SendSTUIArg ::= SEQUENCE {
 uSIServiceIndicator [0] USIServiceIndicator,
 legID [1] LegID DEFAULT sendingSideID:leg1,
 uSIInformation  [2] USIInformation,
 extensions  [3] SEQUENCE SIZE(1..numOfExtensions)  OF ExtensionField OPTIONAL
 }


−− Direction: SSF−>SCF, Timer: Tsfr
−− This operation is used to send back to the SCF the values of counters specified in a previous
−− ActivateServiceFiltering operation

ServiceFilteringResponseArg ::= SEQUENCE {
 countersValue  [0] CountersValue,
 filteringCriteria [1] FilteringCriteria,
 extensions  [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OPTIONAL,
 responseCondition [3] ResponseCondition OPTIONAL
 }


−−  Direction: SCF −> SSF. Timer: T sl
−−  This operation  is issued by the SCF  to separate one joined leg from a multi−way connection
−−  or to interrupt the bearer connection between the involved legs of a single 2 party Call segment.
SplitLegArg ::= SEQUENCE {
 legToBeSplit [0] LegID,
 newCallSegment [1] INTEGER (2..numOfCSs),
 extensions [2] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField OPTIONAL
 }
−− Inclusion of operation and arguments of SCF/SRF
−− INCS2SCFSRFopsargs { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) modules(0) in−cs2−scf−srf−ops−a

−− OPERATION
−−FROM Remote−Operations−Information−Objects ros−InformationObjects
−− opcode−playAnnouncement,
−− opcode−promptAndCollectUserInformation,
−− opcode−promptAndReceiveMessage,
```

Annex A: ssf−scf

```
−− opcode−scriptClose,
−− opcode−scriptEvent,
−− opcode−scriptInformation,
−− opcode−scriptRun,
−− opcode−specializedResourceReport
−−FROM IN−CS2−operationcodes operationcodes
−− +++++++++++++
−− Imports have been moved to the top
−− +++++++++++++
−− canceled,
−− improperCallerResponse,
−− missingParameter,
−− parameterOutOfRange,
−− systemFailure,
−− taskRefused,
−− unavailableResource,
−− unexpectedComponentSequence,
−− unexpectedDataValue,
−− unexpectedParameter
−−FROM IN−CS2−errortypes errortypes
−− UISCRIPT,
−− SupportedUIScripts,
−− PARAMETERS−BOUND
−−FROM IN−CS2−classes classes
−− ros−InformationObjects, operationcodes, datatypes, errortypes, classes
−−FROM IN−CS2−object−identifiers { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) modules(0) in−cs2−obje

Code ::= CHOICE {
   local INTEGER(1..65353),
   global OBJECT IDENTIFIER
  }

−− Direction: SCF −> SRF, Timer: Tpa
−− This operation is to be used after Establish Temporary Connection (assist procedure with a second SSP)
−− or a Connect to Resource (no assist) operation. It may be used for inband interaction with an analog user,
−− or for interaction with an ISDN user. In the former case, the SRF is usually collocated with the SSF for
−− standard tones (congestion tone...) or standard announcements. In the latter case, the SRF is always
−− collocated with the SSF in the switch. Any error is returned to the SCF. The timer associated with this
−− operation must be of a sufficient duration to allow its linked operation to be correctly correlated.

PlayAnnouncementArg  ::= SEQUENCE {
 informationToSend [0] InformationToSend ,
 disconnectFromIPForbidden [1] BOOLEAN DEFAULT TRUE,
 requestAnnouncementComplete [2] BOOLEAN DEFAULT TRUE,
 extensions [3] SEQUENCE SIZE(1..numOfExtensions) OF
    ExtensionField   OPTIONAL,
 connectedParty CHOICE {
   legID [4] DTLegID,
   callSegmentID [5] DTCallSegmentID
   }  OPTIONAL
 }


−− Direction: SCF −>  SRF, Timer: Tpc
−− This operation is used to interact with a user to collect information.
PromptAndCollectUserInformationArg  ::= SEQUENCE {
 collectedInfo [0] CollectedInfo,
 disconnectFromIPForbidden [1] BOOLEAN DEFAULT TRUE,
 informationToSend  [2] InformationToSend  OPTIONAL,
 extensions [3] SEQUENCE SIZE(1..numOfExtensions) OF
    ExtensionField   OPTIONAL,
 callSegmentID [4] DTCallSegmentID   OPTIONAL
 }
ReceivedInformationArg  ::= CHOICE {
 digitsResponse [0] Digits ,
```

```
 iA5Response [1] IA5String
 }

−−  Direction: SCF −>SRF, Timer: Tprm
−−  Used to prompt a user to store a message
PromptAndReceiveMessageArg  ::= SEQUENCE {
 disconnectFromIPForbidden [0] BOOLEAN DEFAULT TRUE,
 informationToSend  [1] InformationToSend  OPTIONAL,
 extensions   [3] SEQUENCE SIZE(0..numOfExtensions) OF ExtensionField OPTIONAL,
 subscriberID   [4] GenericNumber  OPTIONAL,
 mailBoxID   [5] MailBoxID  OPTIONAL,
 informationToRecord  [6] InformationToRecord ,
 media    [7] Media,  −− DEFAULT voiceMail,
 callSegmentID   [8] DTCallSegmentID  OPTIONAL
 }

MessageReceivedArg  ::= SEQUENCE {
 receivedStatus [0] ReceivedStatus,
 recordedMessageID [1] RecordedMessageID OPTIONAL,
 recordedMessageUnits [2] INTEGER(1..maxRecordedMessageUnits) OPTIONAL,
 extensions [3] SEQUENCE SIZE(1..numOfExtensions) OF
       ExtensionField  OPTIONAL
 }

−− Direction:SCF−> SRF. Timer :TCl
−− This operation is issued by the SCF to deallocate the resources used to perform the
−− instance of the "User Interaction" script : the context is released.
ScriptCloseArg  ::= SEQUENCE {
 uIScriptId  [3] Code,
 uIScriptSpecificInfo [0] OCTET STRING OPTIONAL,
 extensions  [1] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField  OPTIONAL,
 callSegmentID  [2] DTCallSegmentID  OPTIONAL
 }

−− Direction:SRF−> SCF. Timer :TRe
−− This operation is issued by the SRF to return information to the SCF on the results of the
−− execution of the instance of User Interaction script.
ScriptEventArg  ::= SEQUENCE {
 uIScriptId  [3] Code,
 uIScriptResult  [0] OCTET STRING OPTIONAL,
 extensions  [1] SEQUENCE SIZE (1..numOfExtensions) OF
       ExtensionField  OPTIONAL,
 callSegmentID  [2] DTCallSegmentID  OPTIONAL,
 lastEventIndicator [3] BOOLEAN  DEFAULT  FALSE
 }


−− Direction:SCF−> SRF. Timer :Tinf
ScriptInformationArg ::= SEQUENCE {
 uIScriptId  [3] Code,
 uIScriptSpecificInfo [0] OCTET STRING OPTIONAL,
 extensions [1] SEQUENCE SIZE(0..numOfExtensions) OF
       ExtensionField  OPTIONAL,
 callSegmentID [2] DTCallSegmentID  OPTIONAL
 }

−− Direction:SCF−> SRF. Timer :Tru
−− This operation is issued by the SCF to allocate the necessary resources to perform the
−− instance of the "User Interaction" script and then to activate this "User Interaction" script
−− instance. A context is partially defined for it if necessary.
ScriptRunArg  ::= SEQUENCE {
 uIScriptId  [3] Code,
 uIScriptSpecificInfo [0] OCTET STRING OPTIONAL,
 extensions [1] SEQUENCE SIZE (1..numOfExtensions) OF
      ExtensionField  OPTIONAL,
```

```
 disconnectFromIPForbidden [2] BOOLEAN DEFAULT TRUE,
 callSegmentID [3] DTCallSegmentID  OPTIONAL
 }

−− Direction: SRF −>  SCF, Timer: Tsrr
−− This operation is used as  the response to a PlayAnnouncement operation when the announcement completed
−− report indication is set.

SpecializedResourceReportArg ::=   NULL


END
```

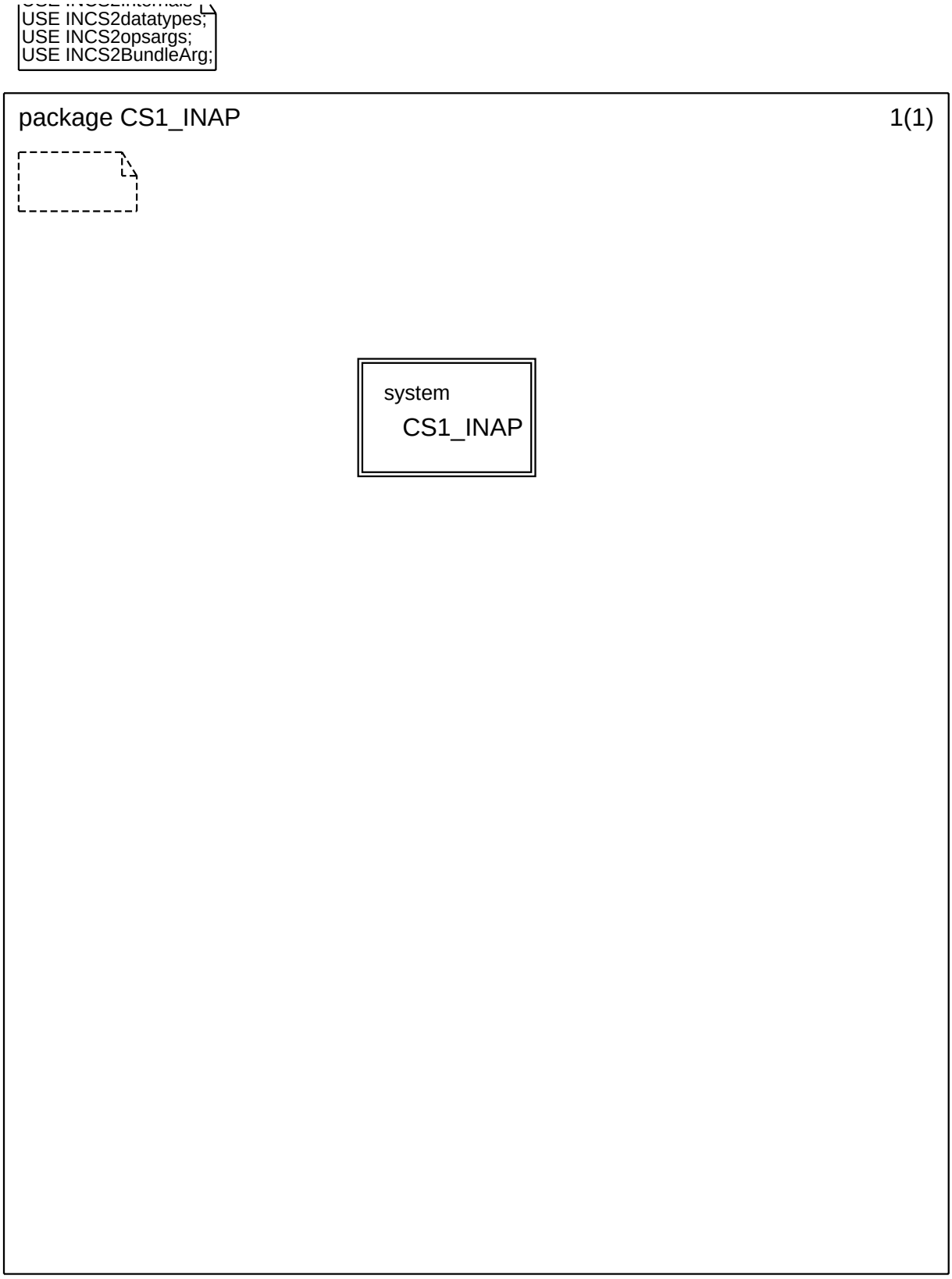
Annex A: ssf−srf

```
INCS2SCFSRFopsargs { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) modules(0) in−cs2−scf−srf−ops−args
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
IMPORTS
−− OPERATION
−−FROM Remote−Operations−Information−Objects ros−InformationObjects

−− opcode−playAnnouncement,
−− opcode−promptAndCollectUserInformation,
−− opcode−promptAndReceiveMessage,
−− opcode−scriptClose,
−− opcode−scriptEvent,
−− opcode−scriptInformation,
−− opcode−scriptRun,
−− opcode−specializedResourceReport
−−FROM IN−CS2−operationcodes operationcodes
 CallSegmentID,
 CollectedInfo,
 Digits,
 ExtensionField,
 InformationToRecord,
 InformationToSend,
 LegID,
 MailBoxID,
 Media,
 GenericNumber,
 ReceivedStatus,
 RecordedMessageID
FROM INCS2datatypes datatypes
−− canceled,
−− improperCallerResponse,
−− missingParameter,
−− parameterOutOfRange,
−− systemFailure,
−− taskRefused,
−− unavailableResource,
−− unexpectedComponentSequence,
−− unexpectedDataValue,
−− unexpectedParameter
−−FROM IN−CS2−errortypes errortypes
−− UISCRIPT,
−− SupportedUIScripts,
−− PARAMETERS−BOUND
−−FROM IN−CS2−classes classes
−− ros−InformationObjects, operationcodes, datatypes, errortypes, classes
−−FROM IN−CS2−object−identifiers { ccitt(0) identified−organization(4) etsi(0) inDomain(1) in−network(1) modules(0) in−cs2−obje
;

−− Direction: SCF −> SRF, Timer: Tpa
−− This operation is to be used after Establish Temporary Connection (assist procedure with a second SSP)
−− or a Connect to Resource (no assist) operation. It may be used for inband interaction with an analog user,
−− or for interaction with an ISDN user. In the former case, the SRF is usually collocated with the SSF for
−− standard tones (congestion tone...) or standard announcements. In the latter case, the SRF is always
−− collocated with the SSF in the switch. Any error is returned to the SCF. The timer associated with this
−− operation must be of a sufficient duration to allow its linked operation to be correctly correlated.

PlayAnnouncementArg  ::= SEQUENCE {
 informationToSend [0] InformationToSend ,
 disconnectFromIPForbidden [1] BOOLEAN DEFAULT TRUE,
 requestAnnouncementComplete [2] BOOLEAN DEFAULT TRUE,
 extensions [3] SEQUENCE SIZE(1..numOfExtensions) OF
    ExtensionField   OPTIONAL,
 connectedParty CHOICE {
   legID [4] DTLegID,
   callSegmentID [5] DTCallSegmentID
```

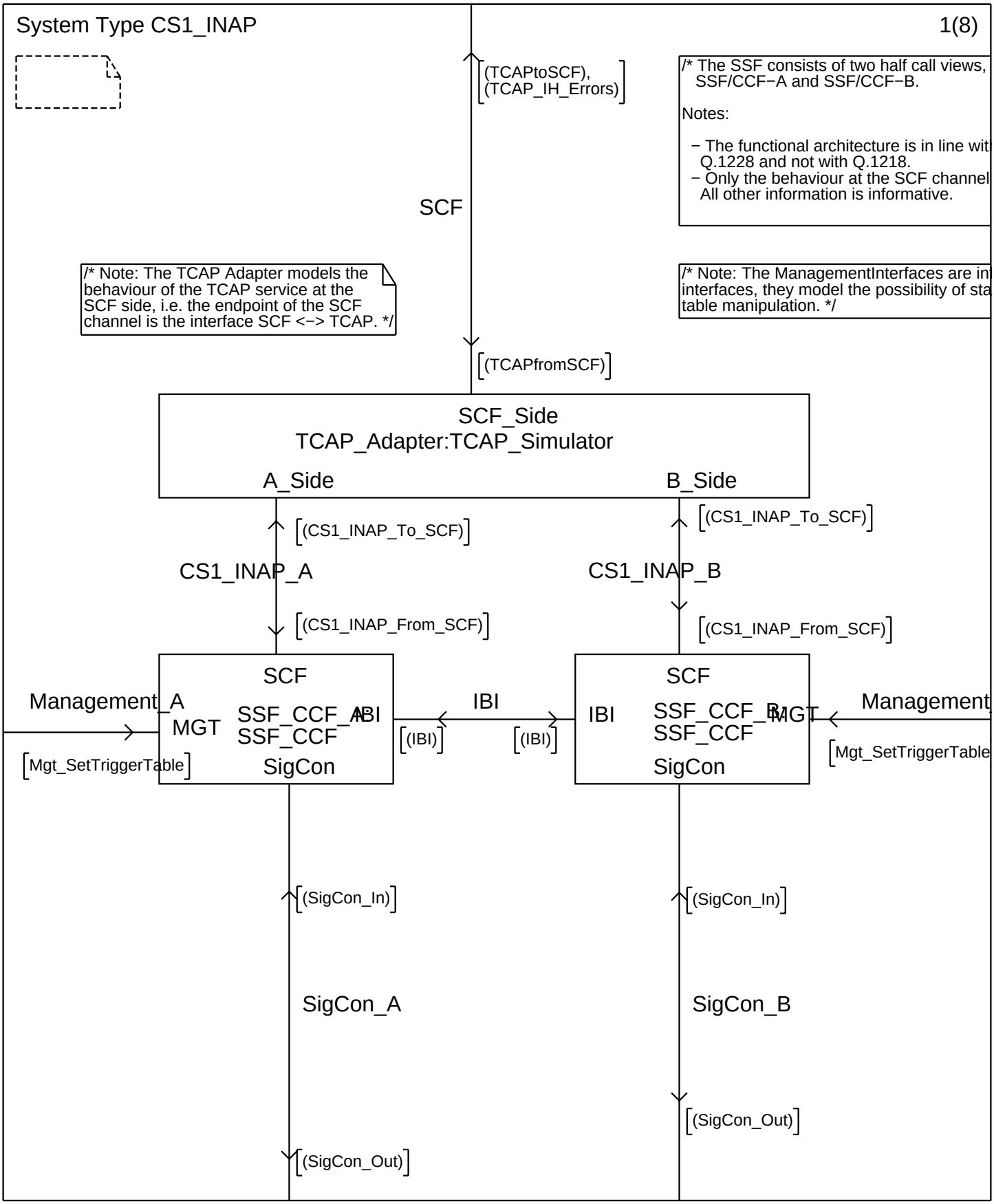
```
  }  OPTIONAL
 }


−− Direction: SCF −>  SRF, Timer: Tpc
−− This operation is used to interact with a user to collect information.
PromptAndCollectUserInformationArg  ::= SEQUENCE {
 collectedInfo [0] CollectedInfo,
 disconnectFromIPForbidden [1] BOOLEAN DEFAULT TRUE,
 informationToSend  [2] InformationToSend  OPTIONAL,
 extensions [3] SEQUENCE SIZE(1..numOfExtensions) OF
    ExtensionField   OPTIONAL,
 callSegmentID [4] DTCallSegmentID   OPTIONAL
 }
ReceivedInformationArg  ::= CHOICE {
 digitsResponse [0] Digits ,
 iA5Response [1] IA5String
 }

−−  Direction: SCF −>SRF, Timer: Tprm
−−  Used to prompt a user to store a message
PromptAndReceiveMessageArg  ::= SEQUENCE {
 disconnectFromIPForbidden [0] BOOLEAN DEFAULT TRUE,
 informationToSend  [1] InformationToSend  OPTIONAL,
 extensions   [3] SEQUENCE SIZE(0..numOfExtensions) OF ExtensionField OPTIONAL,
 subscriberID   [4] GenericNumber  OPTIONAL,
 mailBoxID   [5] MailBoxID  OPTIONAL,
 informationToRecord  [6] InformationToRecord ,
 media    [7] Media,  −− DEFAULT voiceMail,
 callSegmentID   [8] DTCallSegmentID  OPTIONAL
 }

MessageReceivedArg  ::= SEQUENCE {
 receivedStatus [0] ReceivedStatus,
 recordedMessageID [1] RecordedMessageID OPTIONAL,
 recordedMessageUnits [2] INTEGER(1..maxRecordedMessageUnits) OPTIONAL,
 extensions [3] SEQUENCE SIZE(1..numOfExtensions) OF
       ExtensionField  OPTIONAL
 }

−− Direction:SCF−> SRF. Timer :TCl
−− This operation is issued by the SCF to deallocate the resources used to perform the
−− instance of the "User Interaction" script : the context is released.
ScriptCloseArg  ::= SEQUENCE {
 uIScriptId  [3] OCTET STRING,
 uIScriptSpecificInfo [0] OCTET STRING OPTIONAL,
 extensions  [1] SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField  OPTIONAL,
 callSegmentID  [2] DTCallSegmentID  OPTIONAL
 }

−− Direction:SRF−> SCF. Timer :TRe
−− This operation is issued by the SRF to return information to the SCF on the results of the
−− execution of the instance of User Interaction script.
ScriptEventArg  ::= SEQUENCE {
 uIScriptId  [3] OCTET STRING,
 uIScriptResult  [0] OCTET STRING OPTIONAL,
 extensions  [1] SEQUENCE SIZE (1..numOfExtensions) OF
       ExtensionField  OPTIONAL,
 callSegmentID  [2] DTCallSegmentID  OPTIONAL,
 lastEventIndicator [3] BOOLEAN  DEFAULT  FALSE
 }


−− Direction:SCF−> SRF. Timer :Tinf
ScriptInformationArg ::= SEQUENCE {
```

```
 uIScriptId  [3] OCTET STRING,
 uIScriptSpecificInfo [0] OCTET STRING OPTIONAL,
 extensions [1] SEQUENCE SIZE(0..numOfExtensions) OF
       ExtensionField  OPTIONAL,
 callSegmentID [2] DTCallSegmentID  OPTIONAL
 }

−− Direction:SCF−> SRF. Timer :Tru
−− This operation is issued by the SCF to allocate the necessary resources to perform the
−− instance of the "User Interaction" script and then to activate this "User Interaction" script
−− instance. A context is partially defined for it if necessary.
ScriptRunArg  ::= SEQUENCE {
 uIScriptId  [3] OCTET STRING,
 uIScriptSpecificInfo [0] OCTET STRING OPTIONAL,
 extensions [1] SEQUENCE SIZE (1..numOfExtensions) OF
       ExtensionField  OPTIONAL,
 disconnectFromIPForbidden [2] BOOLEAN DEFAULT TRUE,
 callSegmentID [3] DTCallSegmentID  OPTIONAL
 }

−− Direction: SRF −>  SCF, Timer: Tsrr
−− This operation is used as  the response to a PlayAnnouncement operation when the announcement completed
−− report indication is set.
SpecializedResourceReportArg ::=   NULL
END
```

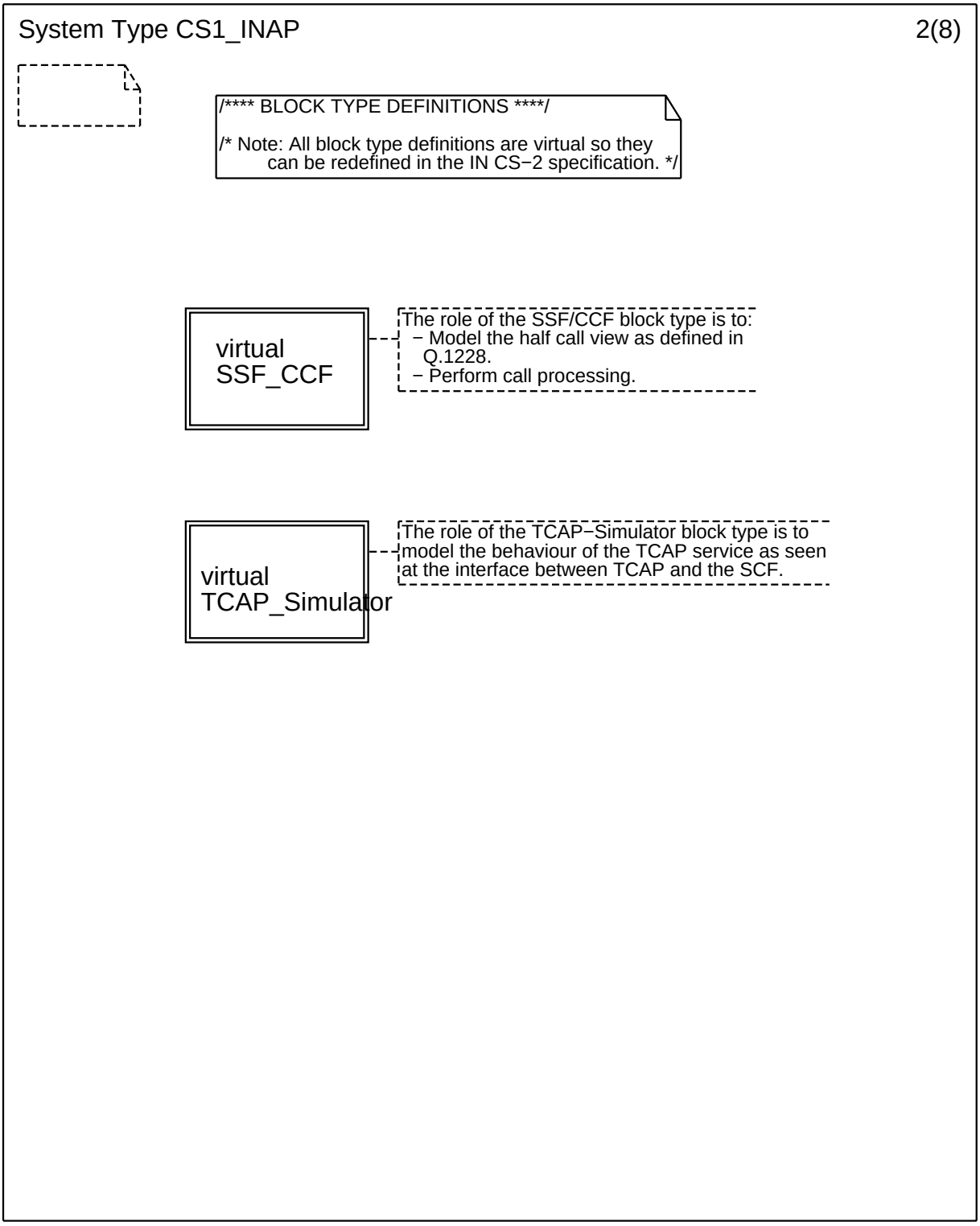
Annex A:


USE INCS2datatypes;
USE INCS2opsargs;
USE INCS2BundleArg;
package CS1_INAP
1(1)
system
CS1_INAP

Annex A:


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 wit Q.1228 and not with Q.1218.
− Only the behaviour at the SCF channel 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. */
/* Note: The ManagementInterfaces are in interfaces, they model the possibility of sta table manipulation. */
(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
SCF
Management_A
MGT
SSF_CCF_A
SSF_CCF
IBI
IBI
(IBI)
(IBI)
IBI
SSF_CCF_B
SSF_CCF
MGT
Management
Mgt_SetTriggerTable
SigCon
SigCon
Mgt_SetTriggerTable
(SigCon_In)
(SigCon_In)
SigCon_A
SigCon_B
(SigCon_Out)
(SigCon_Out)

Annex A:


System Type CS1_INAP
2(8)
/**** 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.

Annex A:

System Type CS1_INAP 3(8)

```
/**** 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
 /* SCF − SSF operations */
 ActivateServiceFiltering(InvokeID,ActivateServiceFilteringArg),
 ActivityTest(InvokeID, DialogIDType),
 ActivityTestResult(InvokeID, DialogIDType),
 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),
 CollectInformation(InvokeID,CSAID,CollectInformationArg),
 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),
 InitialDP(CSAID,InitialDPArg),
 InitiateCallAttempt(InvokeID,CSAID,InitiateCallAttemptArg),
 ReleaseCall(InvokeID,CSAID,ReleaseCallArg),
 RequestNotificationChargingEvent(InvokeID,CSAID,RequestNotificationChargingEventArg),
 RequestReportBCSMEvent(InvokeID,CSAID,RequestReportBCSMEventArg),
 ResetTimer(InvokeID,CSAID,ResetTimerArg),
 SendChargingInformation(InvokeID,CSAID,SendChargingInformationArg),
 ServiceFilteringResponse(CSAID,ServiceFilteringResponseArg),

 /* SCF − SRF operations */
 PlayAnnouncement(InvokeID, PlayAnnouncementArg),
 PromptAndCollectUserInformation(InvokeID, CSAID, PromptAndCollectUserInformationArg),
 PromptAndCollectUserInformationResult(CSAID, ReceivedInformationArg),
 SpecializedResourceReport(CSAID, SpecializedResourceReportArg),

 /* Informative operations */
 Mgt_SetTriggerTable(MGTSetTriggerTableArg); /* Informative signal, used to model the management of the trigger tabl
```

Annex A:

System Type CS1_INAP 4(8)

/* 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
    wrongDialID, /* 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) */
    noDialDavail, /* 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,
               PA, PAC, PAR, SC, SR, SI, SE, SRR,
  /* 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 */
```

Annex A:

System Type CS1_INAP 5(8)

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

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

SIGNAL
  CSaIDreq(DialogIDtype),
  CSaIDresp(DialogIDtype,CSAID),
  RegisterSSFreq(IHroleType),
  RegisterSSFresp(IHroleType),
  TCAPFailureInd(TCAPfailReasonType);

/* TCAP Primitives */

SIGNAL
/* TC_nameReq: Direction SCF->TCAP */
  TC_InvokeReq(InvokeIDtype,DialogIDtype,OpClassType,OpCodeType,TimeoutValType
  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 Adap

/* 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. */
```

Annex A:

System Type CS1_INAP 6(8)

```
/* 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 */
```

Annex A:

System Type CS1_INAP 7(8)

```
/***** 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,

  PromptAndCollectUserInformationResult,
  SpecializedResourceReport,

  Errror,
  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd,

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

```
/* SCF −> SSF */

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

  PlayAnnouncement,
  PromptAndCollectUserInformation,

  Mgt_SetTriggerTable, /* Operation on the management inte


  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd,

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

Annex A:

System Type CS1_INAP 8(8)

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

/* 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);
```

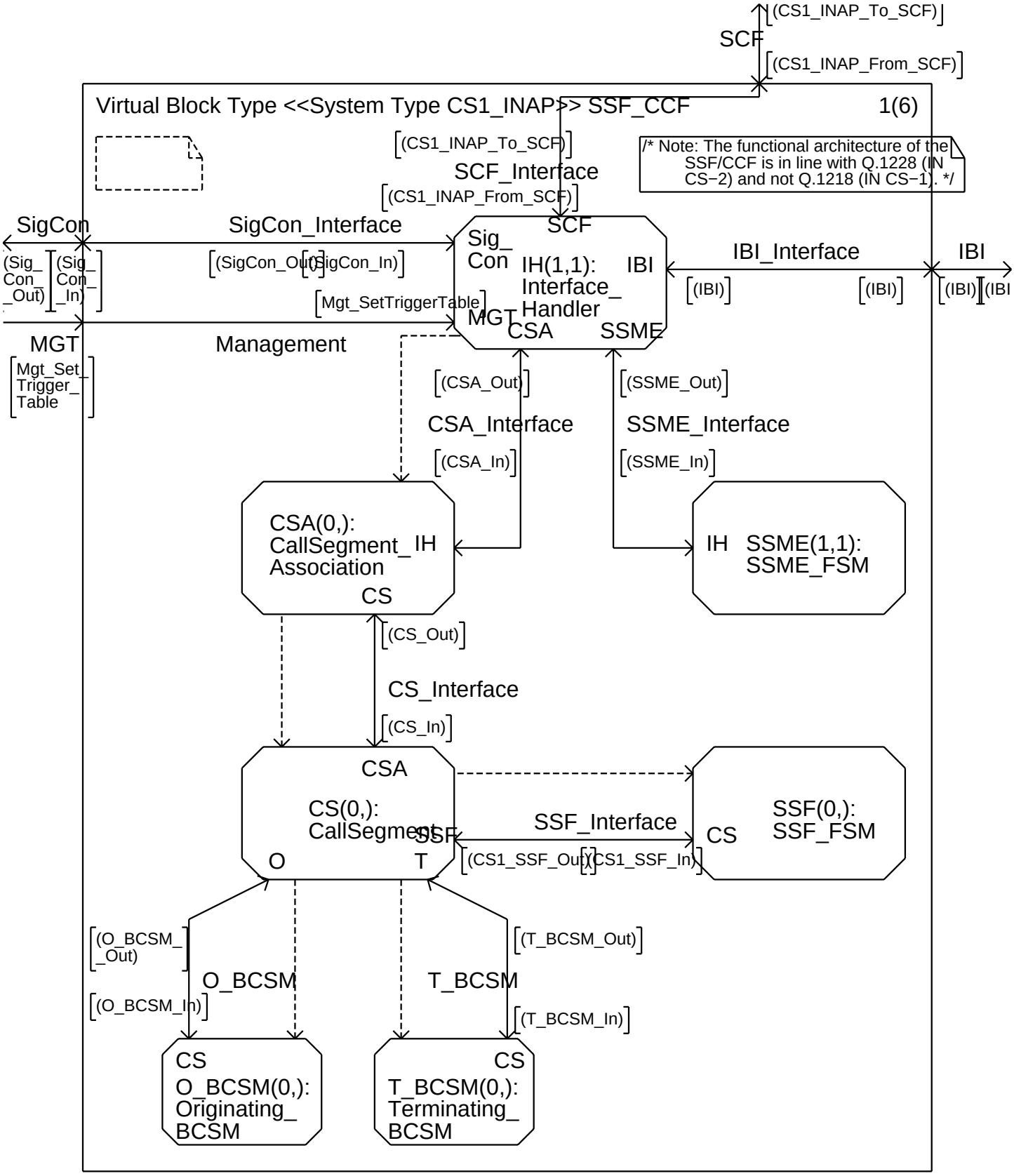
```
SIGNALLIST SigCon_In =
  (O_SigCon_In),
  (T_SigCon_In);

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

```
/* 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;
```

```
/* 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);
```

Annex A:


(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)
Sig_ Con
SCF
IH(1,1): Interface_ Handler
IBI
IBI_Interface
IBI
(IBI)
(IBI)
(IBI)
(IBI
Mgt_SetTriggerTable
MGT
MGT
CSA
SSME
Management
Mgt_Set_ Trigger_ Table
(CSA_Out)
(SSME_Out)
CSA_Interface
SSME_Interface
(CSA_In)
(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)
(T_BCSM_Out)
O_BCSM
T_BCSM
(O_BCSM_In)
(T_BCSM_In)
CS
O_BCSM(0,): Originating_ BCSM
CS
T_BCSM(0,): Terminating_ BCSM

Annex A:

Virtual Block Type <<System Type CS1_INAP>> SSF_CCF 2(6)

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

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

virtual Interface_Handler

The InterFaceHandler (IH) process type manages:
− The creation of CallSegmentAssociations (CSAs).
− The dialouge with the SCF.
− The passing of primitives between the Signalling Control Interface and the CSAs.
− The passing of primitives between the IN CS−1 SSF−SCF Interface and the CSA.

virtual CallSegment_Association

The CallSegmentAssociation (CSA) process type manages:
− The creation of CallSegments
− The passing of Signalling Control primitives between the InterfaceHandler and the CallSegment.
− The passing of IN CS−1 primitives between the InterfaceHandler and the CallSegment.

Note: In IN CS−1 only one call segment/CSA can exist.

virtual CallSegment

The CallSegment (CS) process type manages:
− The connection view (in terms of legs) of the call segment.
− The creation of O−BCSMs and T−BCSMs.
− The creation of the SSF−FSM.
− Processing of connection view oriented IN CS−1 operations.
− The passing of Signalling Control primitives between the CallSegmentAssociation and the BCSMs.
− The passing of IN CS−1 primitives between the CallSegmentAssociation and the SSF−FSM.
− The passing of DPs and PICs between the BCSMs and the SSF−FSM.

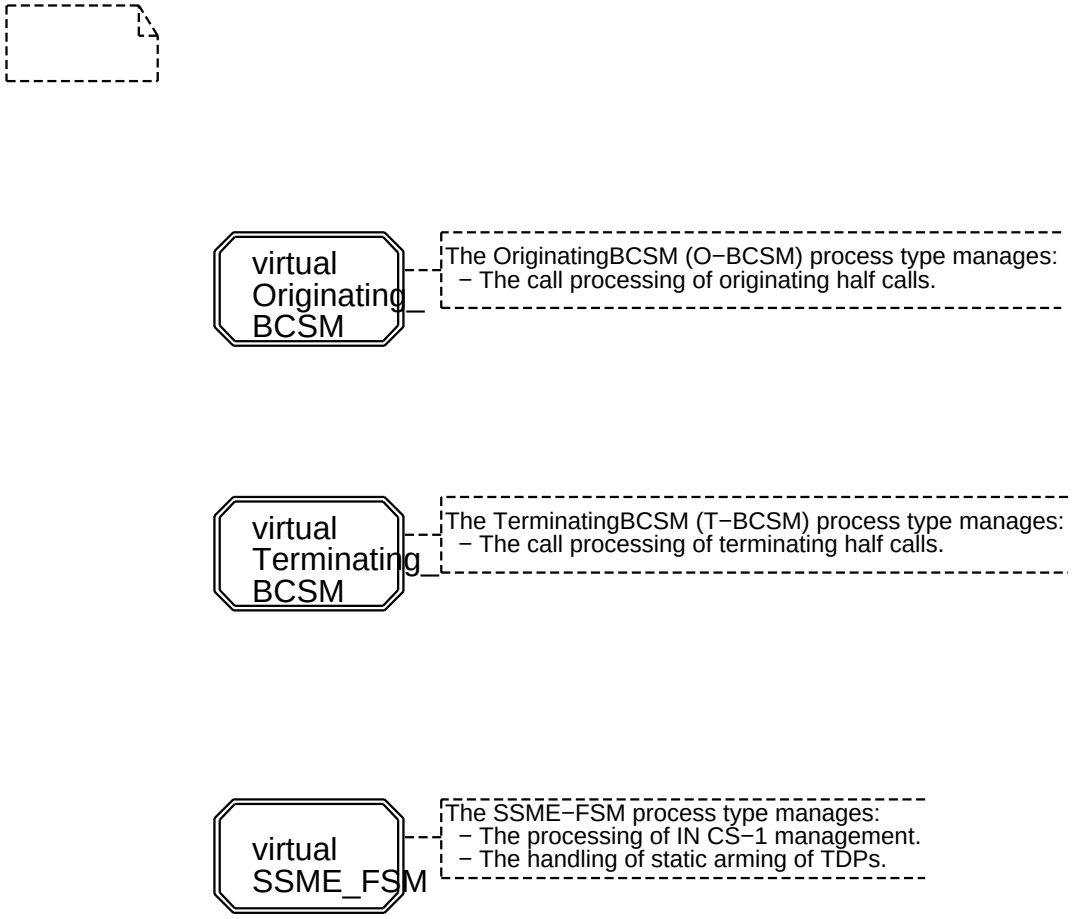
Note: Only one SSF−FSM/CS can exist.

virtual SSF_FSM

The SSF−FSM process type manages:
− The processing of IN CS−1 operations.
− The handling of detection points (EDPs & TDPs).

Annex A:

Virtual Block Type <<System Type CS1_INAP>> SSF_CCF 3(6)



Annex A:

Virtual Block Type <<System Type CS1_INAP>> SSF_CCF 4(6)

```
/**** 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

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

SIGNALLIST tPICs =
  PIC,
  PICResume;

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

Annex A:

Virtual Block Type <<System Type CS1_INAP>> SSF_CCF 5(6)

```
/**** 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
each LegId. The event list contains the  BCSM event
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 a

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

/* Signals used for passing information between the SSF−FSM and
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;
```

Annex A:

Virtual Block Type <<System Type CS1_INAP>> SSF_CCF 6(6)

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

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

  PlayAnnouncement,
  PromptAndCollectUserInformation,

  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;

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

  PromptAndCollectUserInformationResult,
  SpecializedResourceReport,

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

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

  PlayAnnouncement,
  PromptAndCollectUserInformation,

  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;

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

  PromptAndCollectUserInformationResult,
  SpecializedResourceReport,

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

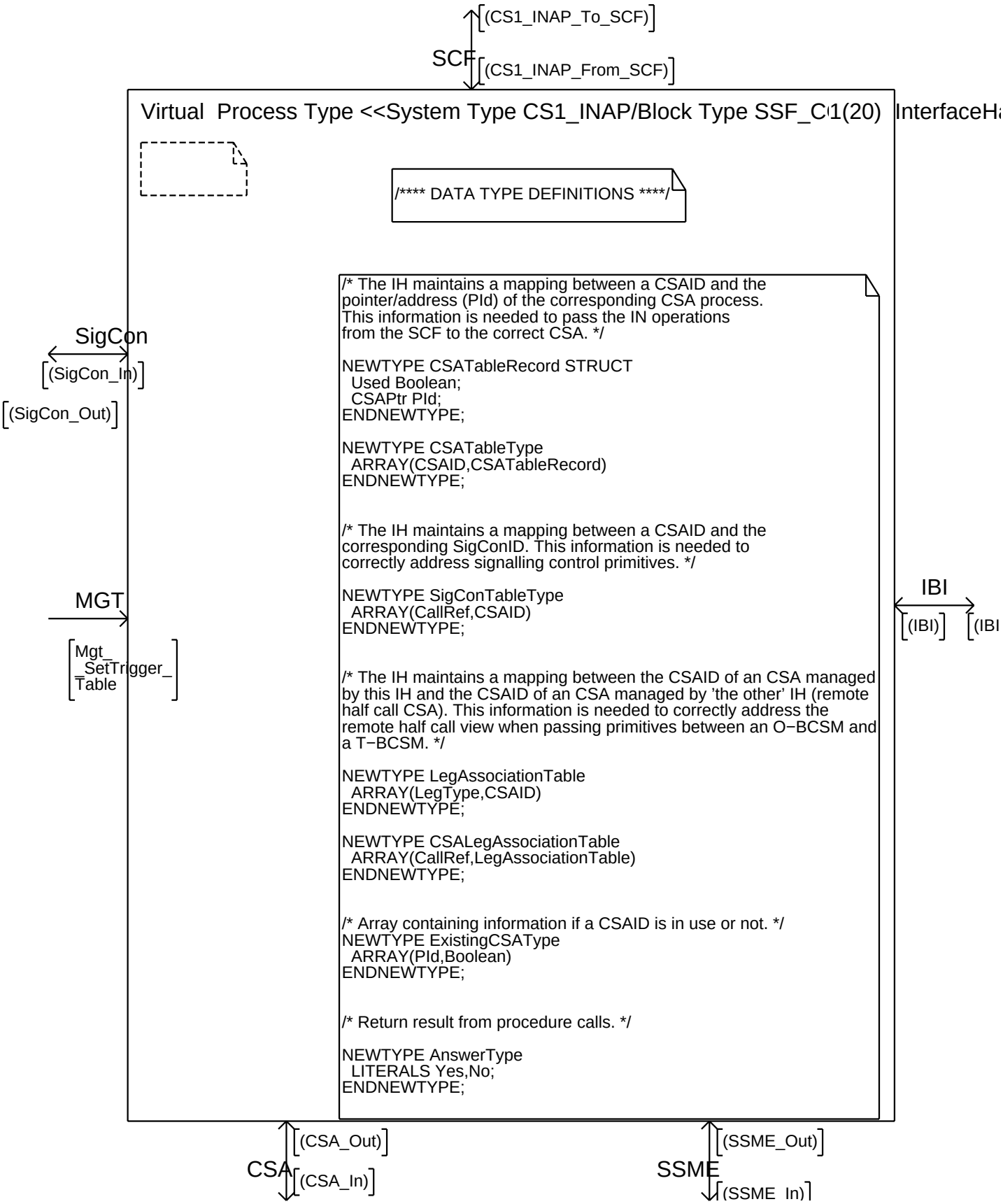
```
SIGNALLIST CS1_INAP_SSF_In =
  ApplyCharging,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continue,
  DisconnectForwardConnection,
  EstablishTemporaryConnection,
  FurnishChargingInformation,
  InitiateCallAttempt,
  ReleaseCall,
  RequestNotificationChargingEvent,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

  PlayAnnouncement,
  PromptAndCollectUserInformation,

  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd;

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

  PromptAndCollectUserInformationResult,
  SpecializedResourceReport,

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

```
SIGNALLIST SSME_In =
  ActivateServiceFiltering
  ActivityTest,
  CallGap,

  (SSME_Reqs),

  Mgt_SetTriggerTable;

SIGNALLIST SSME_Out
  ActivityTestResult,
  ServiceFilteringRespon

  (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),
```

```
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;
```

Annex A:


(CS1_INAP_To_SCF)
SCF
(CS1_INAP_From_SCF)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_C1(20) InterfaceH
/**** 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)
MGT
Mgt_
SetTrigger
Table
IBI
(IBI)
(IBI
(CSA_Out)
CSA
(CSA_In)
(SSME_Out)
SSME
(SSME_In)

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 2(20)

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

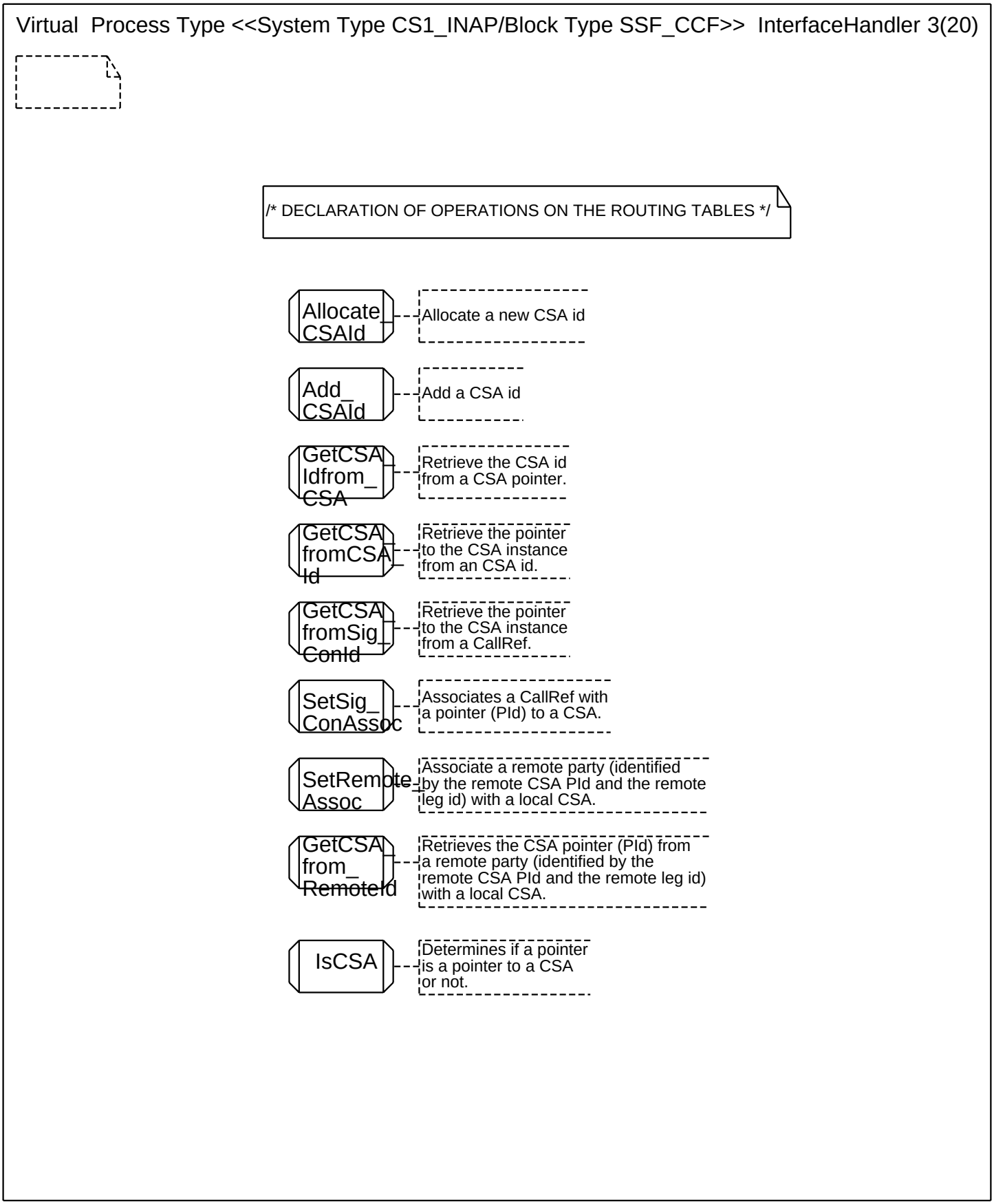
```
DCL
 /* Routing tables. */
 csaTable CSATableType,
 sigConTable SigConTableType,
 csaLegAssoc CSALegAssociationTable,

 /* 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 MGTSetTriggerTableArg;
```

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

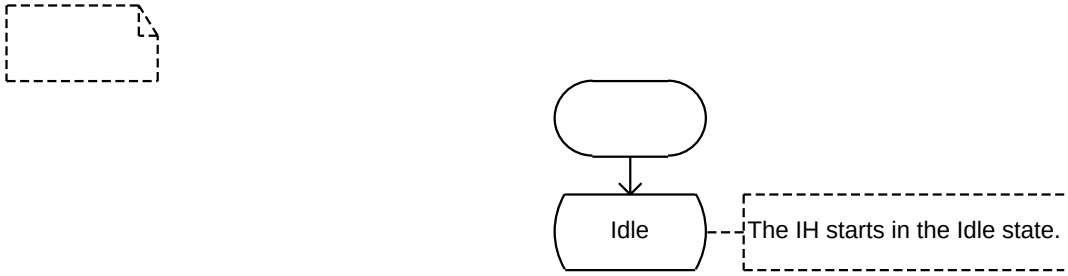
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 3(20)

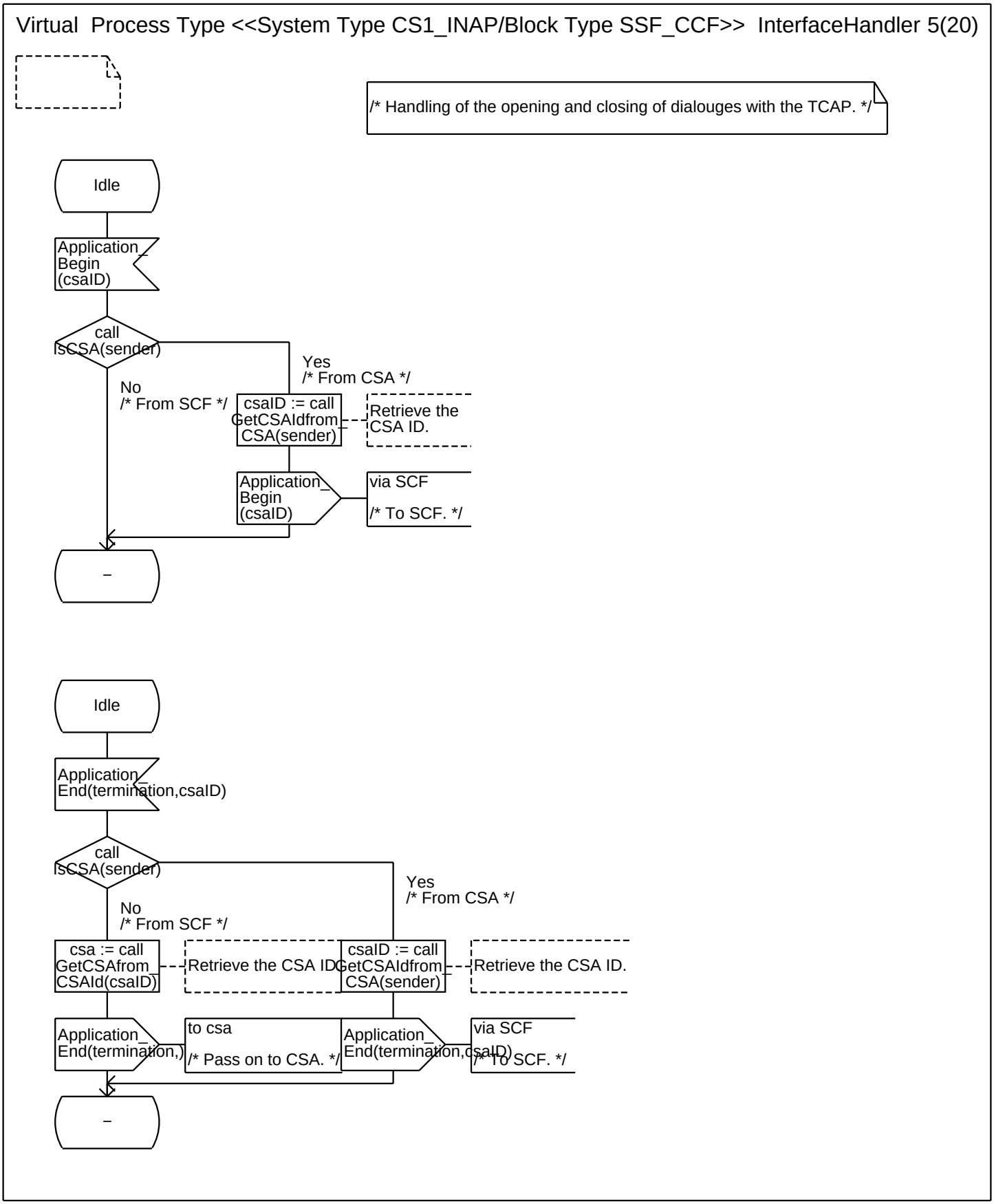


Annex A:

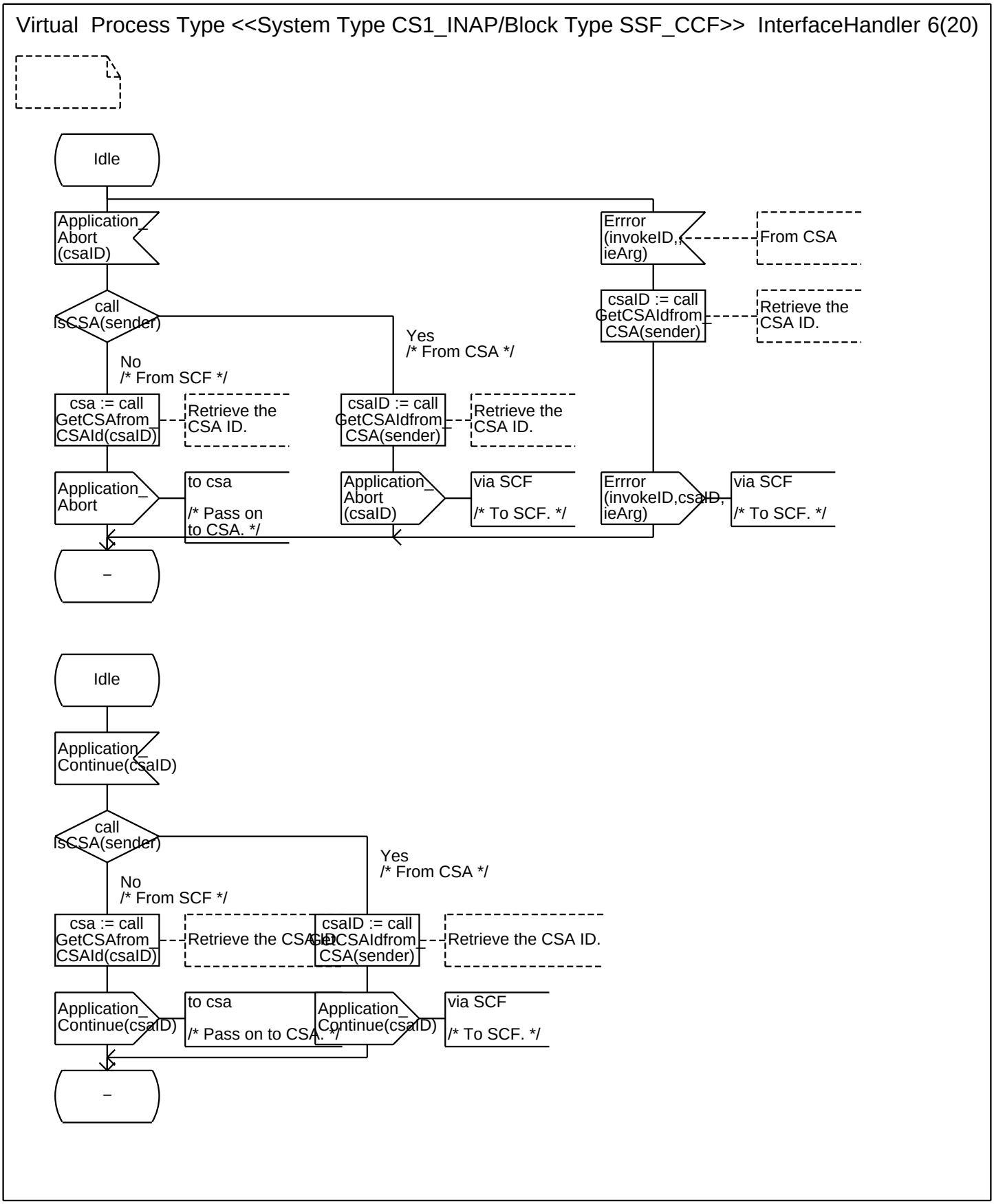
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 4(20)



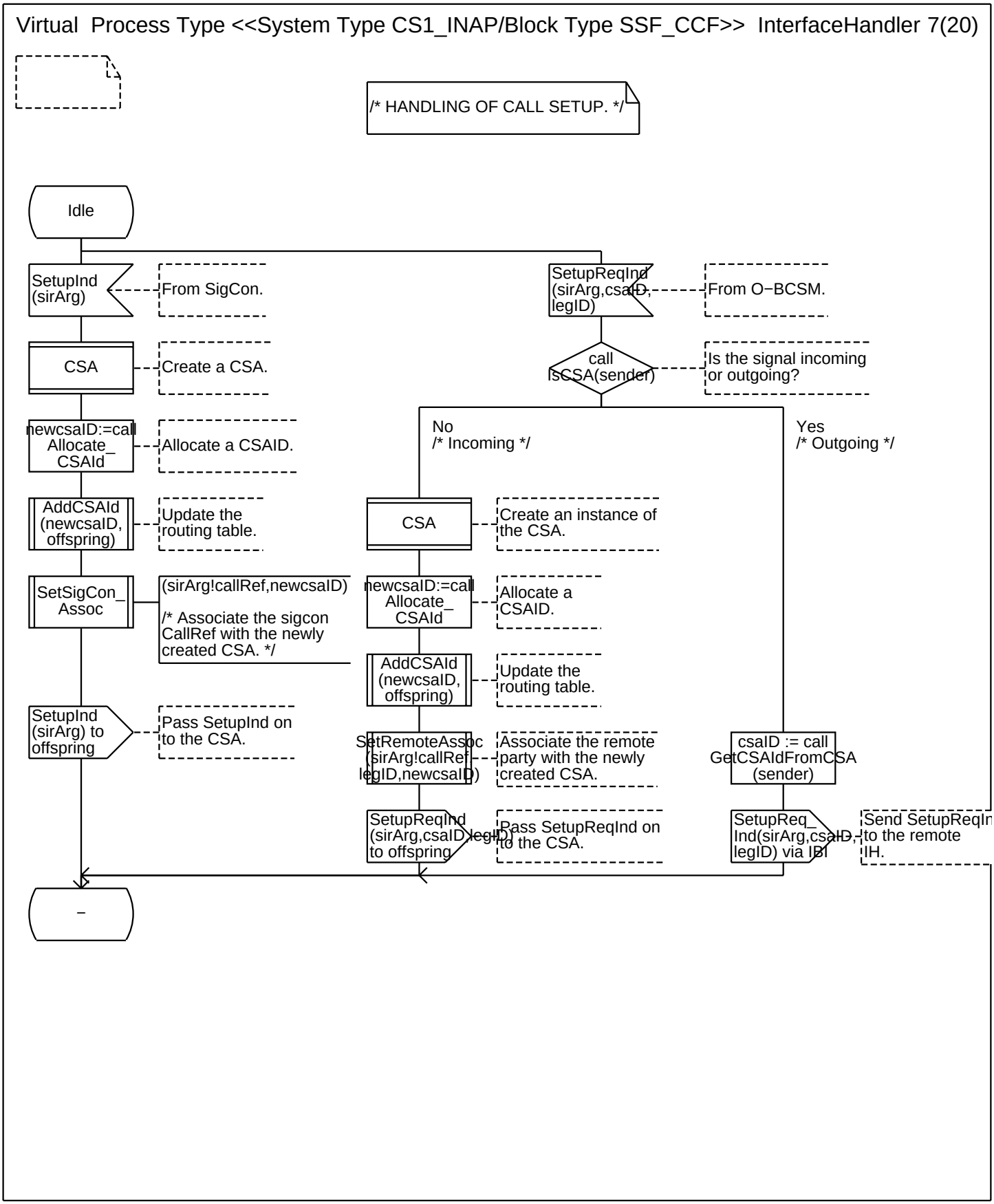
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 5(20)
/* 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. */
–

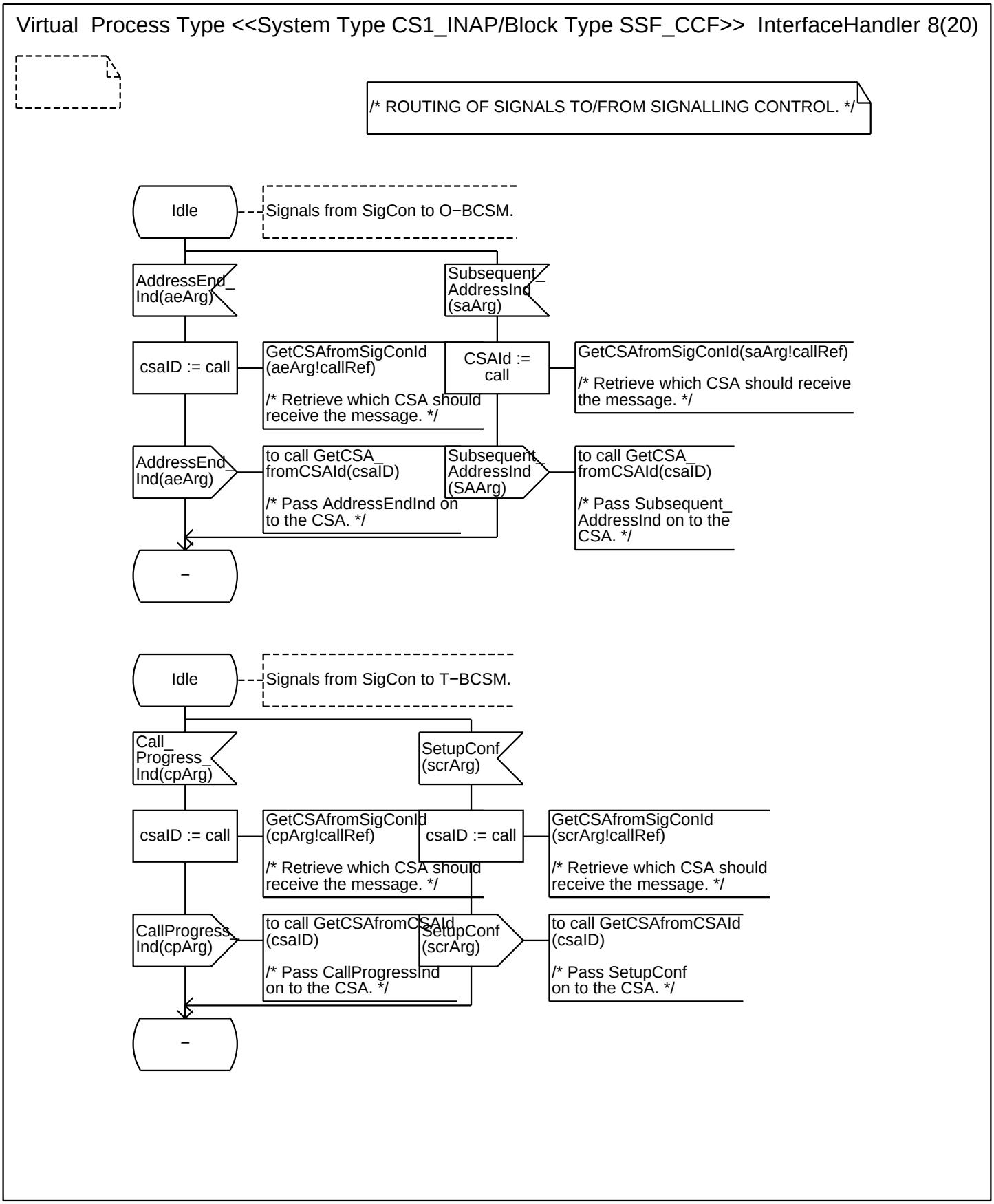
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 6(20)
Idle
Application_ Abort (csaID)
Errror (invokeID,ieArg)
From CSA
call IsCSA(sender)
csaID := call GetCSAIdfrom_ CSA(sender)
Retrieve the CSA ID.
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,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. */
–

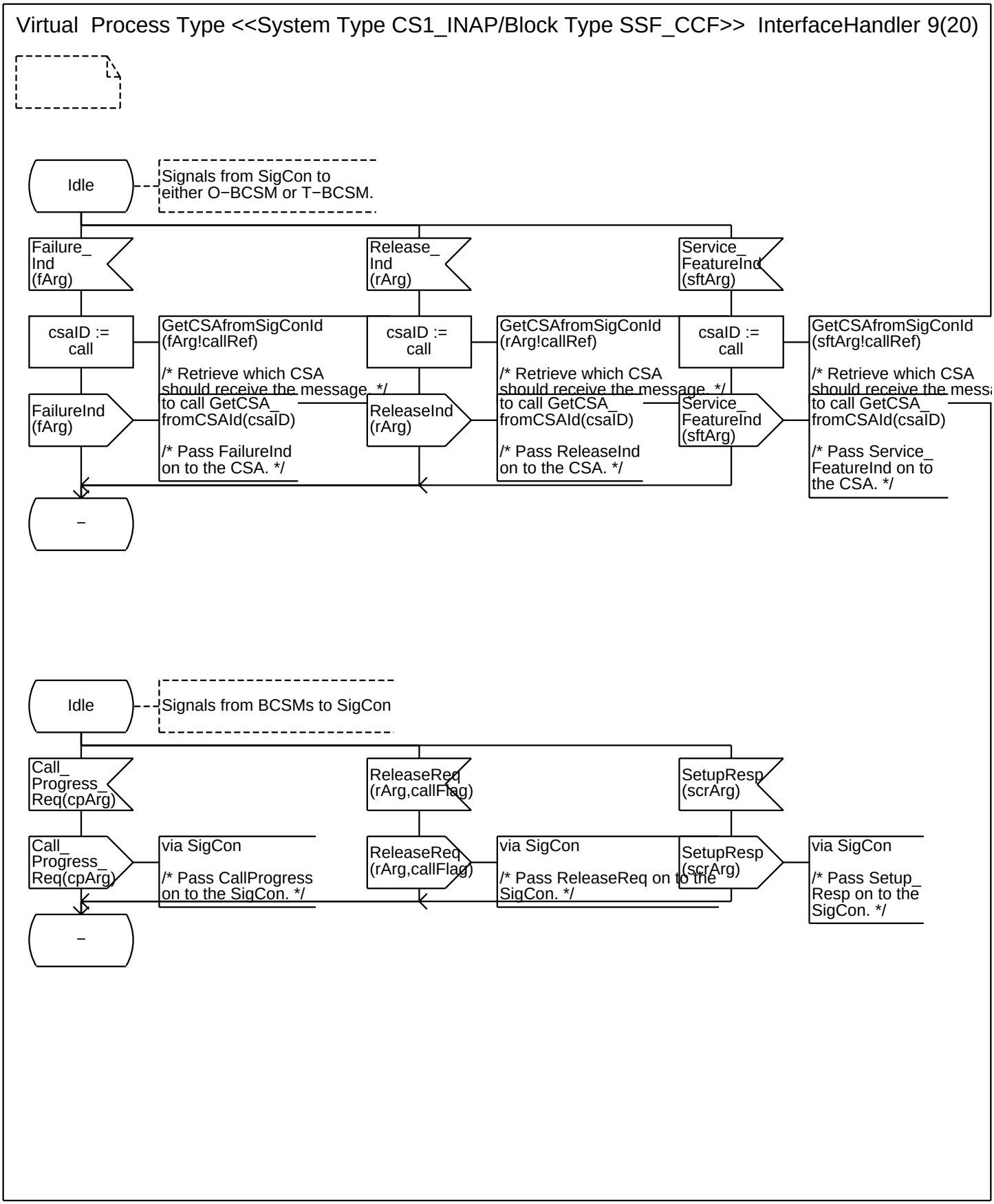
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 7(20)
/* 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 SetupReqIn to the remote IH.
–

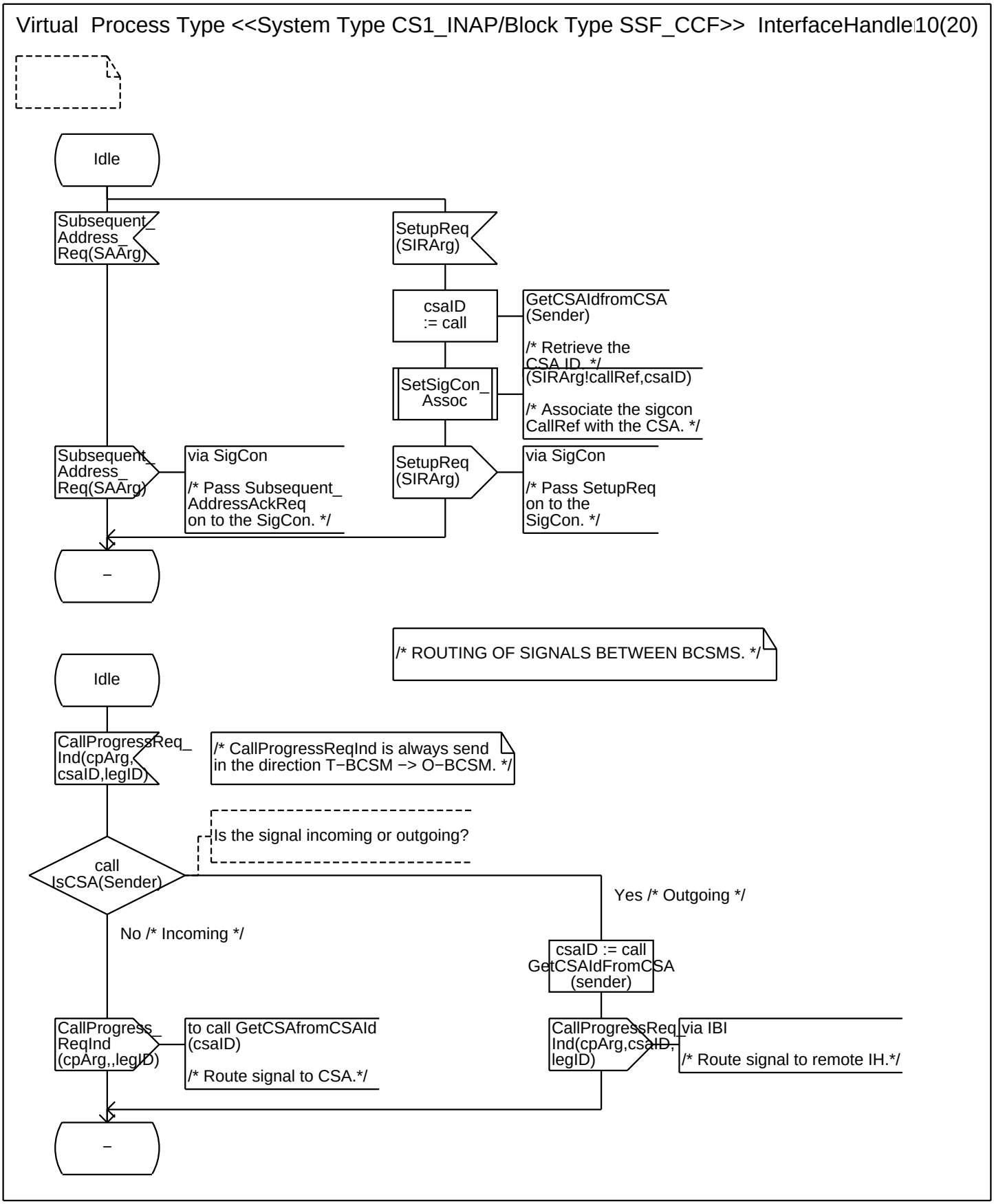
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 8(20)
/* ROUTING OF SIGNALS TO/FROM SIGNALLING CONTROL. */
Idle
Signals from SigCon to O−BCSM.
AddressEnd_ Ind(aeArg)
Subsequent_ AddressInd (saArg)
csaID := call
GetCSAfromSigConId (aeArg!callRef)
/* Retrieve which CSA should receive the message. */
CSAId := call
GetCSAfromSigConId(saArg!callRef)
/* Retrieve which CSA should receive the message. */
AddressEnd_ Ind(aeArg)
to call GetCSA_ fromCSAId(csaID)
/* Pass AddressEndInd on to the CSA. */
Subsequent_ AddressInd (SAArg)
to call GetCSA_ fromCSAId(csaID)
/* Pass Subsequent_ AddressInd on to the CSA. */
–
Idle
Signals from SigCon to T−BCSM.
Call_ Progress_ Ind(cpArg)
SetupConf (scrArg)
csaID := call
GetCSAfromSigConId (cpArg!callRef)
/* Retrieve which CSA should receive the message. */
csaID := call
GetCSAfromSigConId (scrArg!callRef)
/* Retrieve which CSA should receive the message. */
CallProgress_ Ind(cpArg)
to call GetCSAfromCSAId (csaID)
/* Pass CallProgressInd on to the CSA. */
SetupConf (scrArg)
to call GetCSAfromCSAId (csaID)
/* Pass SetupConf on to the CSA. */
–

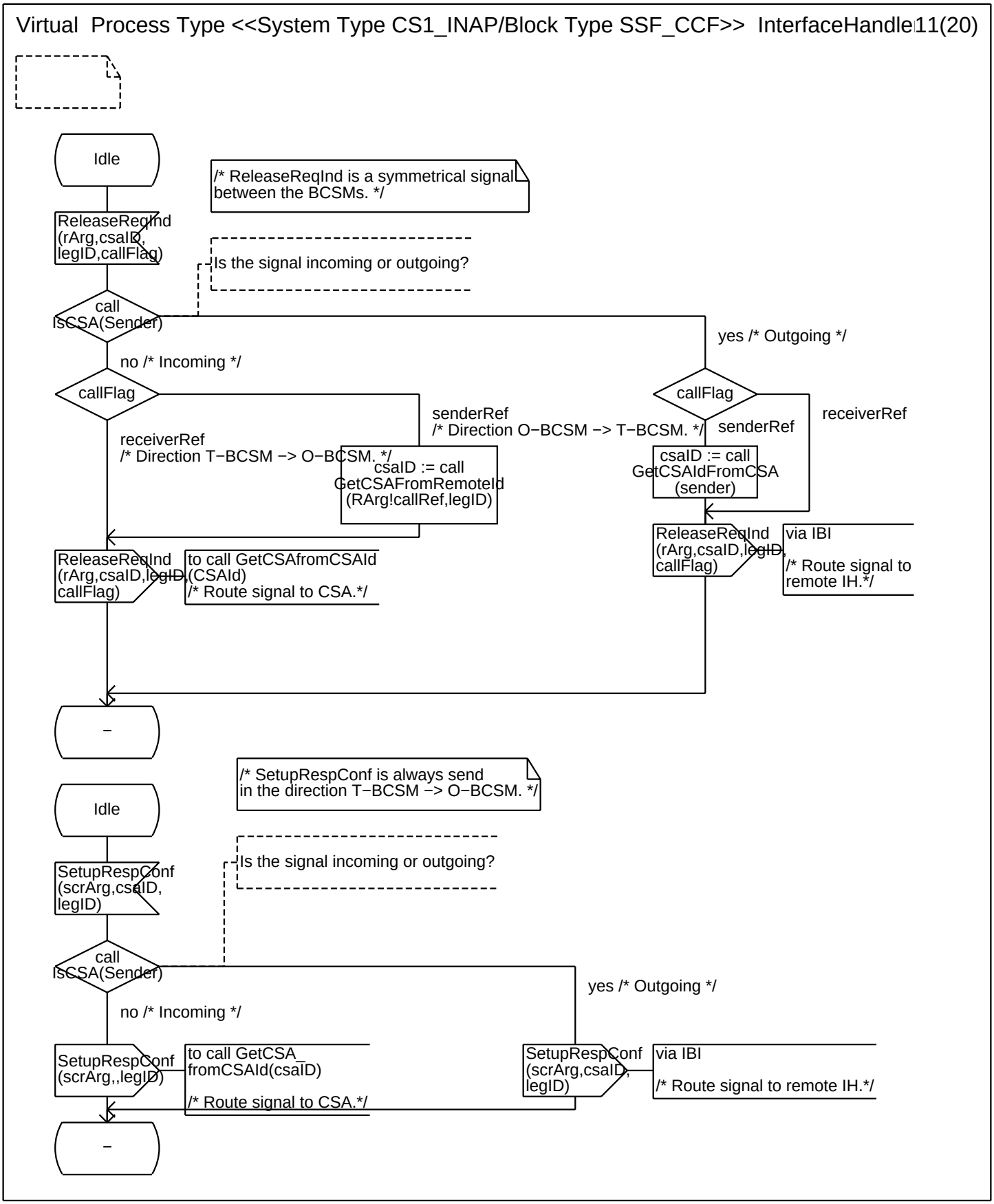
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandler 9(20)
Idle
Signals from SigCon to either O−BCSM or T−BCSM.
Failure_ Ind (fArg)
Release_ Ind (rArg)
Service_ FeatureInd (sftArg)
csaID := call
GetCSAfromSigConId (fArg!callRef)
/* Retrieve which CSA should receive the message. */
FailureInd (fArg)
to call GetCSA_ fromCSAId(csaID)
/* Pass FailureInd on to the CSA. */
csaID := call
GetCSAfromSigConId (rArg!callRef)
/* Retrieve which CSA should receive the message. */
ReleaseInd (rArg)
to call GetCSA_ fromCSAId(csaID)
/* Pass ReleaseInd on to the CSA. */
csaID := call
GetCSAfromSigConId (sftArg!callRef)
/* Retrieve which CSA should receive the messa
Service_ FeatureInd (sftArg)
to call GetCSA_ fromCSAId(csaID)
/* Pass Service_ FeatureInd on to the CSA. */
Idle
Signals from BCSMs to SigCon
Call_ Progress_ Req(cpArg)
ReleaseReq (rArg,callFlag)
SetupResp (scrArg)
Call_ Progress_ Req(cpArg)
via SigCon
/* Pass CallProgress on to the SigCon. */
ReleaseReq (rArg,callFlag)
via SigCon
/* Pass ReleaseReq on to the SigCon. */
SetupResp (scrArg)
via SigCon
/* Pass Setup_ Resp on to the SigCon. */

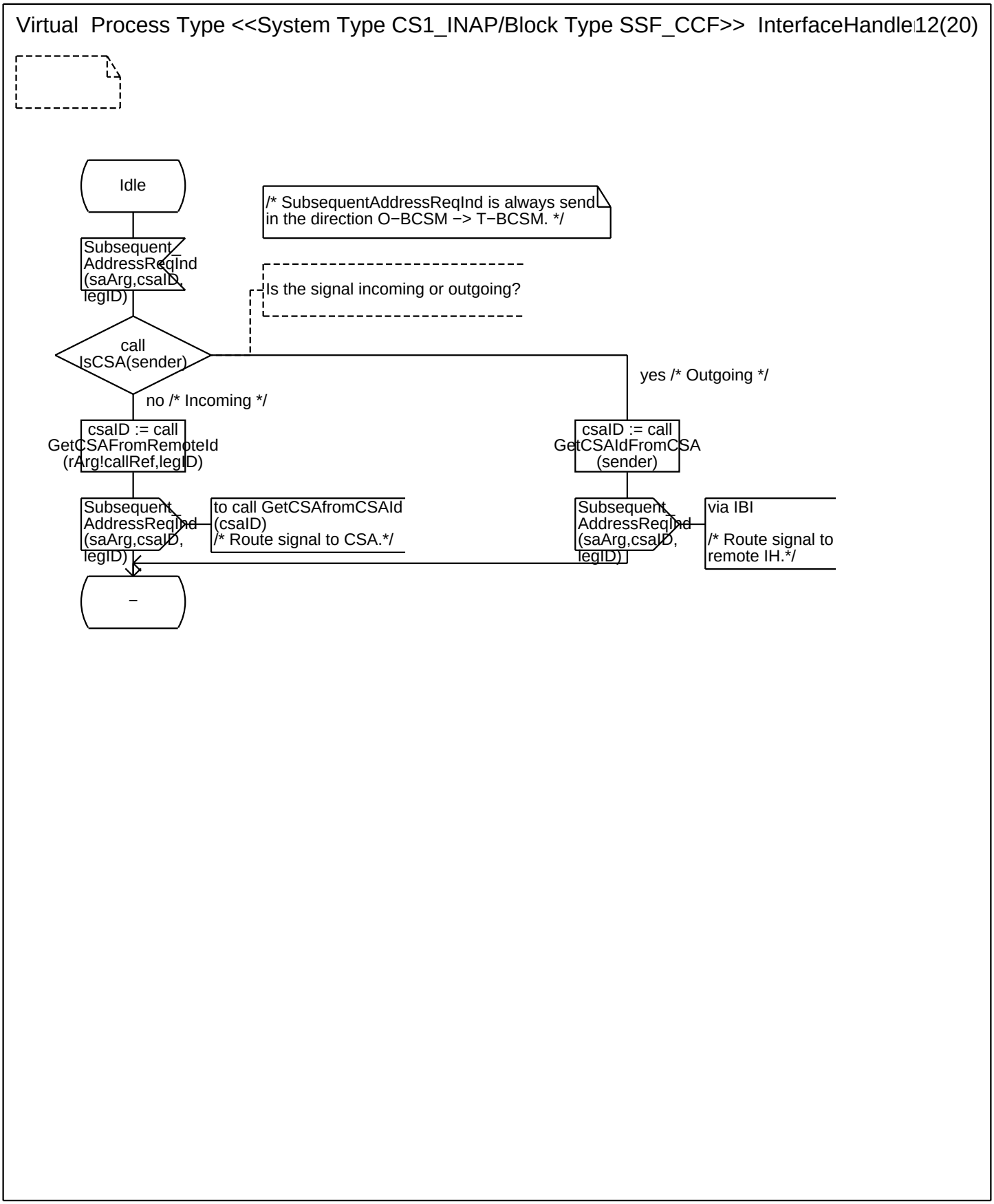
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 10(20)
Idle
Subsequent_ Address_ Req(SAArg)
SetupReq (SIRArg)
csaID := call
GetCSAIdfromCSA (Sender)
/* Retrieve the CSA ID. */
SetSigCon_ Assoc
(SIRArg!callRef,csaID)
/* Associate the sigcon CallRef with the CSA. */
Subsequent_ Address_ Req(SAArg)
via SigCon
/* Pass Subsequent_ AddressAckReq on to the SigCon. */
SetupReq (SIRArg)
via SigCon
/* Pass SetupReq on to the SigCon. */
–
/* ROUTING OF SIGNALS BETWEEN BCSMS. */
Idle
CallProgressReq_ Ind(cpArg, csaID,legID)
/* CallProgressReqInd is always send in the direction T−BCSM −> O−BCSM. */
Is the signal incoming or outgoing?
call IsCSA(Sender)
Yes /* Outgoing */
No /* Incoming */
csaID := call GetCSAIdFromCSA (sender)
CallProgress_ ReqInd (cpArg,,legID)
to call GetCSAfromCSAId (csaID)
/* Route signal to CSA.*/
CallProgressReq_ Ind(cpArg,csaID, legID)
via IBI
/* Route signal to remote IH.*/
–

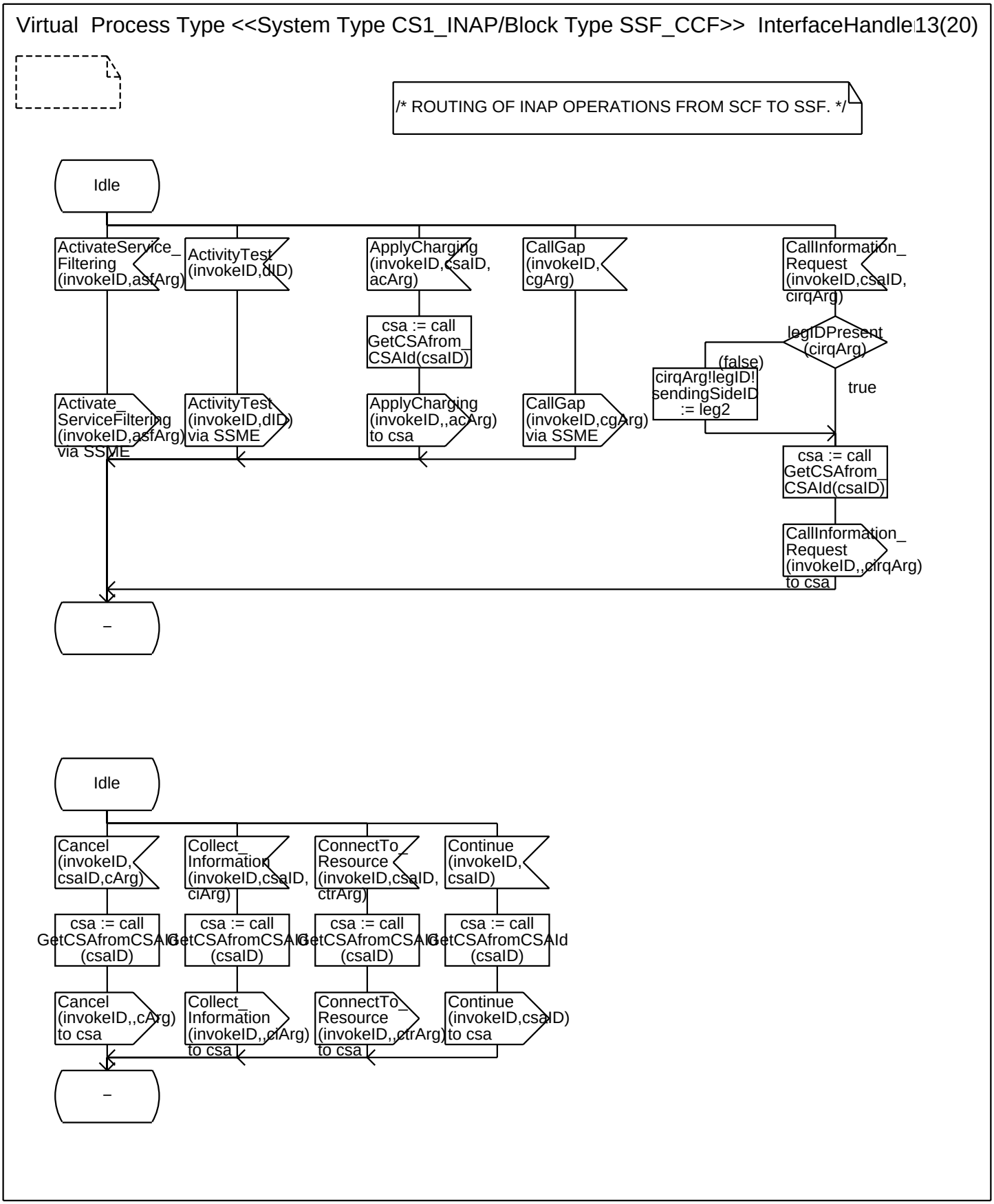
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 11(20)
Idle
/* ReleaseReqInd is a symmetrical signal between the BCSMs. */
ReleaseReqInd (rArg,csaID, legID,callFlag)
Is the signal incoming or outgoing?
call IsCSA(Sender)
yes /* Outgoing */
no /* Incoming */
callFlag
callFlag
senderRef
/* Direction O−BCSM −> T−BCSM. */
receiverRef
senderRef
receiverRef
/* Direction T−BCSM −> O−BCSM. */
csaID := call GetCSAFromRemoteId (RArg!callRef,legID)
csaID := call GetCSAIdFromCSA (sender)
ReleaseReqInd (rArg,csaID,legID, callFlag)
to call GetCSAfromCSAId (CSAId)
/* Route signal to CSA.*/
ReleaseReqInd (rArg,csaID,legID, callFlag)
via IBI
/* Route signal to remote IH.*/
−
/* SetupRespConf is always send in the direction T−BCSM −> O−BCSM. */
Idle
Is the signal incoming or outgoing?
SetupRespConf (scrArg,csaID, legID)
call IsCSA(Sender)
yes /* Outgoing */
no /* Incoming */
SetupRespConf (scrArg,,legID)
to call GetCSA_ fromCSAId(csaID)
/* Route signal to CSA.*/
SetupRespConf (scrArg,csaID, legID)
via IBI
/* Route signal to remote IH.*/
−

Annex A:

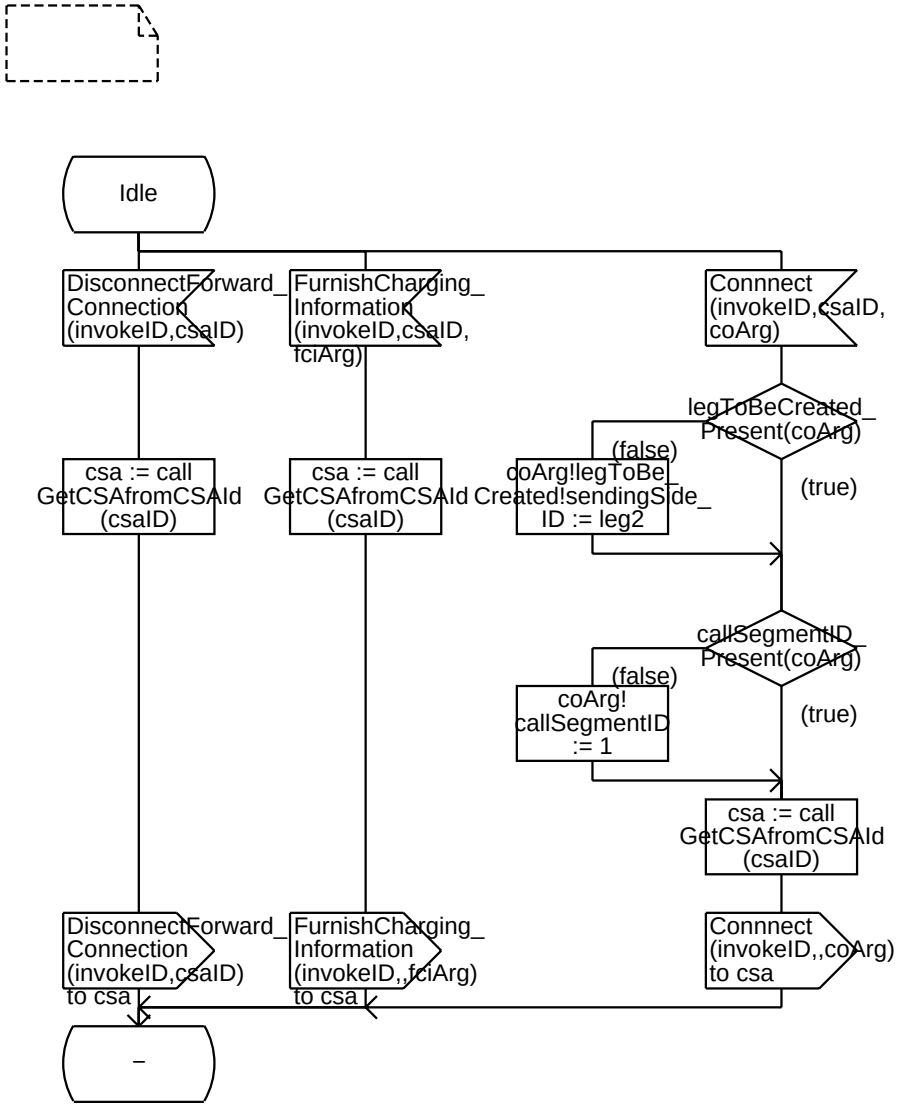

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 12(20)
Idle
/* SubsequentAddressReqInd is always send in the direction O−BCSM −> T−BCSM. */
SubsequentAddressReqInd (saArg,csaID, legID)
Is the signal incoming or outgoing?
call IsCSA(sender)
yes /* Outgoing */
no /* Incoming */
csaID := call GetCSAFromRemoteId (rArg!callRef,legID)
csaID := call GetCSAIdFromCSA (sender)
SubsequentAddressReqInd (saArg,csaID, legID)
to call GetCSAfromCSAId (csaID) /* Route signal to CSA.*/
SubsequentAddressReqInd (saArg,csaID, legID)
via IBI
/* Route signal to remote IH.*/
–

Annex A:

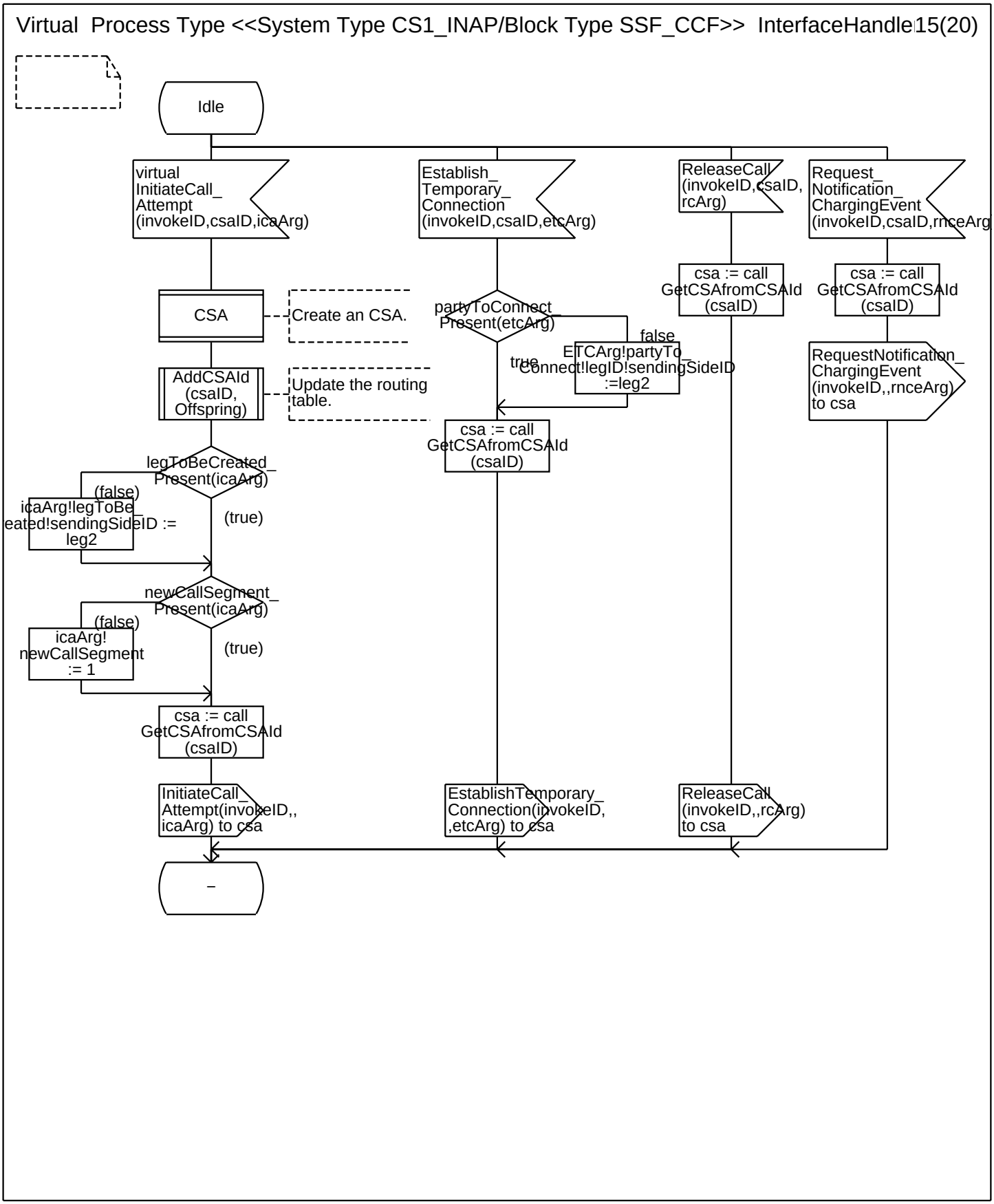

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle13(20)
/* ROUTING OF INAP OPERATIONS FROM SCF TO SSF. */
Idle
ActivateService_Filtering (invokeID,asfArg)
ActivityTest (invokeID,dID)
ApplyCharging (invokeID,csaID, acArg)
CallGap (invokeID, cgArg)
CallInformation_Request (invokeID,csaID, cirqArg)
csa := call GetCSAfrom_CSAId(csaID)
legIDPresent (cirqArg)
(false)
cirqArg!legID! sendingSideID := leg2
true
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
–

Annex A:

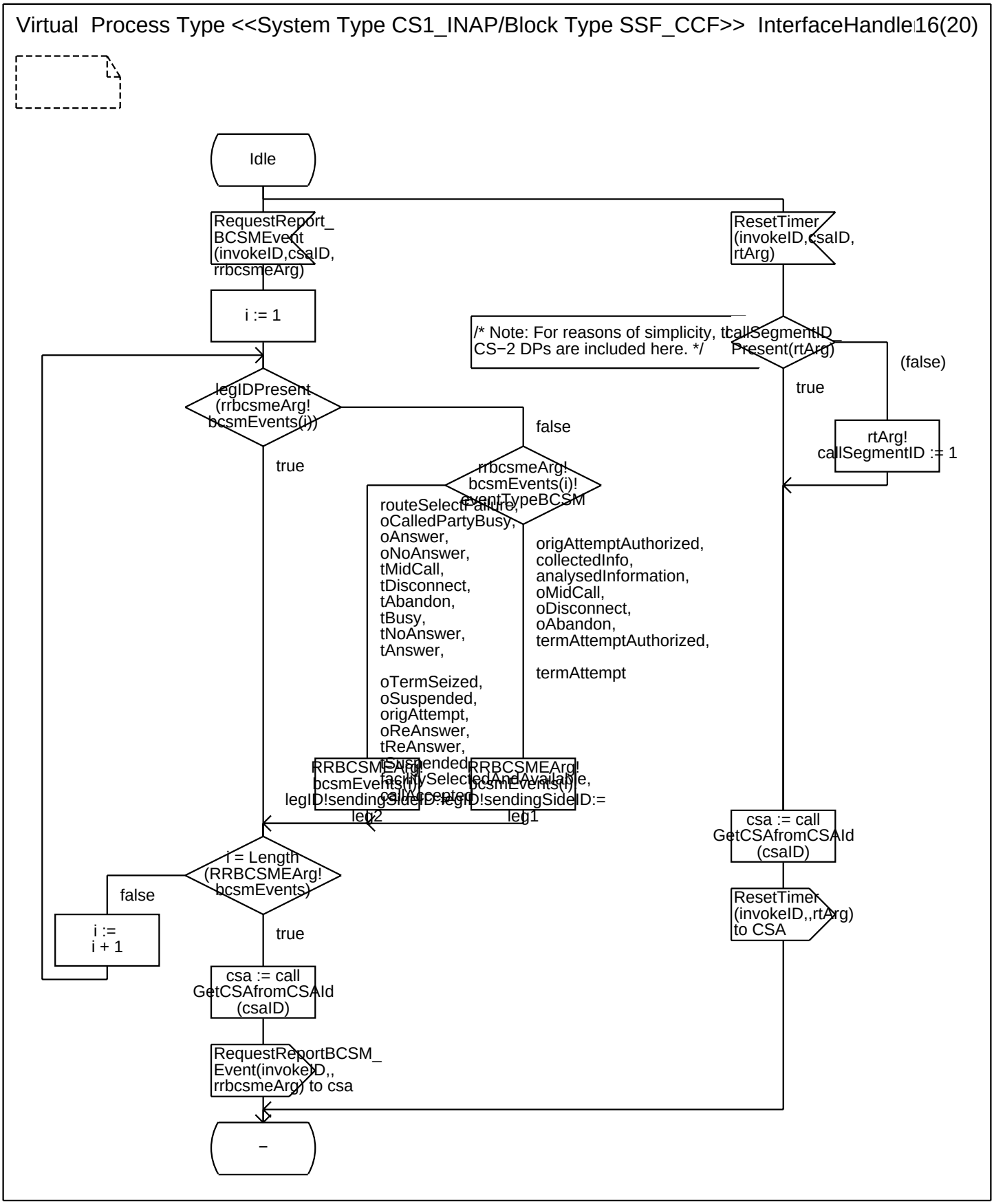
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 14(20)



Annex A:

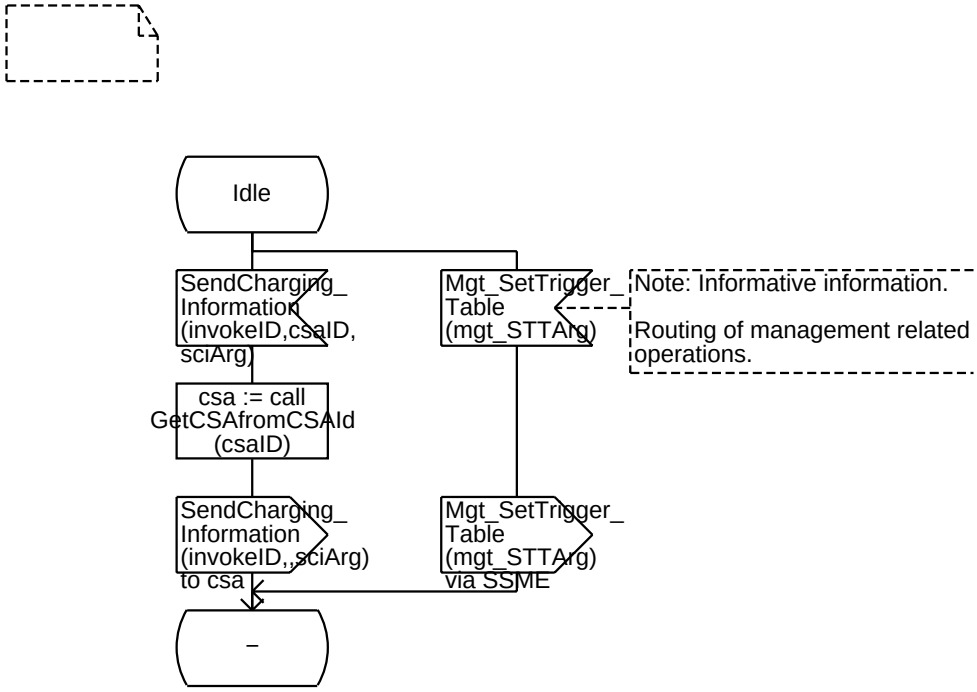

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle15(20)
Idle
virtual InitiateCall_ Attempt (invokeID,csaID,icaArg)
Establish_ Temporary_ Connection (invokeID,csaID,etcArg)
ReleaseCall (invokeID,csaID, rcArg)
Request_ Notification_ ChargingEvent (invokeID,csaID,rnceArg)
CSA
Create an CSA.
AddCSAId (csaID, Offspring)
Update the routing table.
legToBeCreated_ Present(icaArg)
(false)
icaArg!legToBe_ eated!sendingSideID := leg2
(true)
newCallSegment_ Present(icaArg)
(false)
icaArg! newCallSegment := 1
(true)
csa := call GetCSAfromCSAId (csaID)
InitiateCall_ Attempt(invokeID,, icaArg) to csa
partyToConnect_ Present(etcArg)
true
false
ETCArg!partyTo_ Connect!legID!sendingSideID :=leg2
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
–

Annex A:

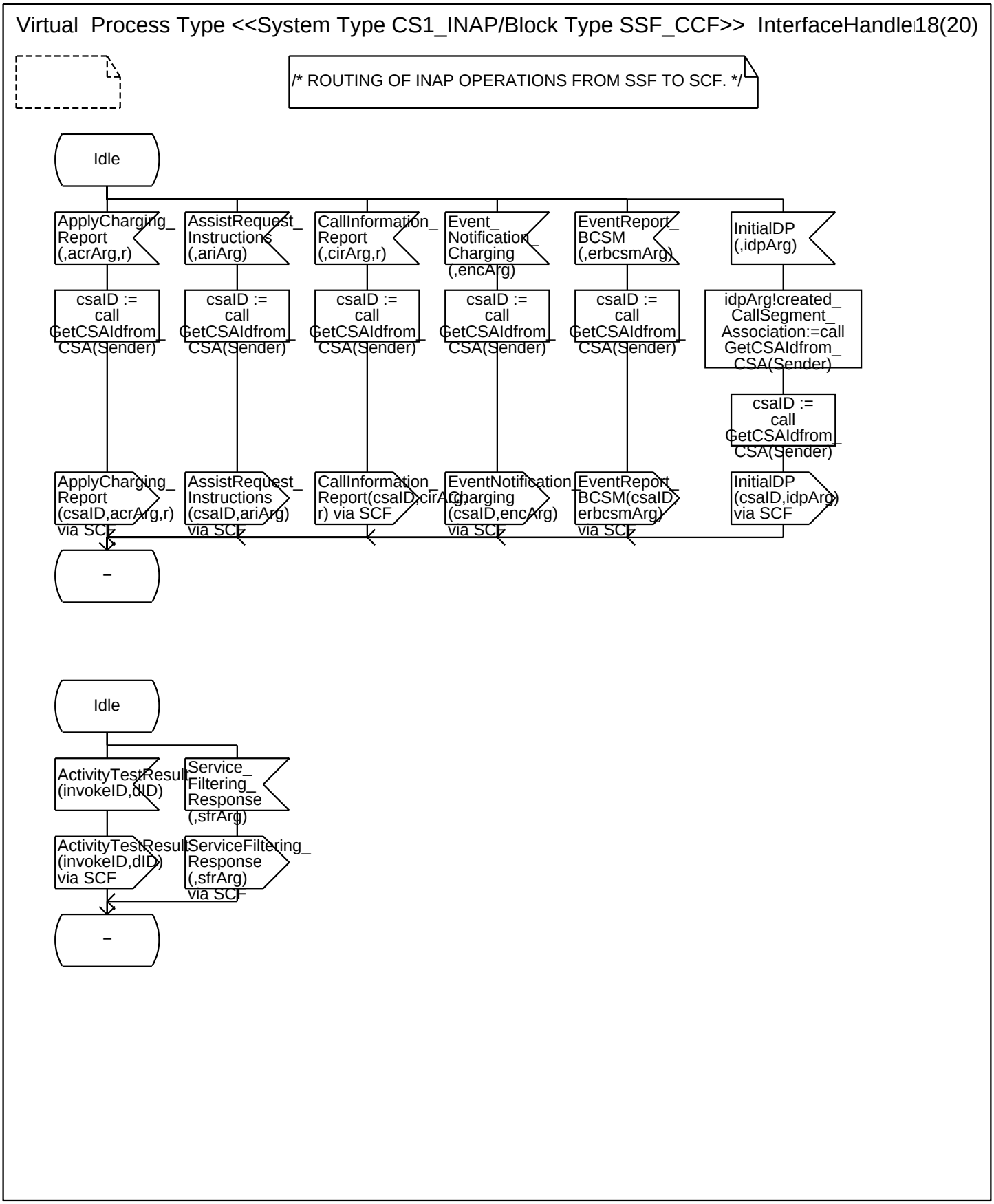

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 16(20)
Idle
RequestReport_ BCSMEvent (invokeID,csaID, rrbcsmeArg)
i := 1
legIDPresent (rrbcsmeArg! bcsmEvents(i))
true
false
rrbcsmeArg! bcsmEvents(i)! eventTypeBCSM
routeSelectFailure, oCalledPartyBusy, oAnswer, oNoAnswer, tMidCall, tDisconnect, tAbandon, tBusy, tNoAnswer, tAnswer,
oTermSeized, oSuspended, origAttempt, oReAnswer, tReAnswer, tSuspended
origAttemptAuthorized, collectedInfo, analysedInformation, oMidCall, oDisconnect, oAbandon, termAttemptAuthorized,
termAttempt
RRBCSMEArg! bcsmEvents(i)! legID!sendingSideID:= leg2
RRBCSMEArg! bcsmEvents(i)! facilitySelectedAndAvailable, callAccepted 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)
/* Note: For reasons of simplicity, tcallSegmentID_ CS−2 DPs are included here. */
callSegmentID_ Present(rtArg)
(false)
true
rtArg! callSegmentID := 1
csa := call GetCSAfromCSAId (csaID)
ResetTimer (invokeID,,rtArg) to CSA

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 17(20)

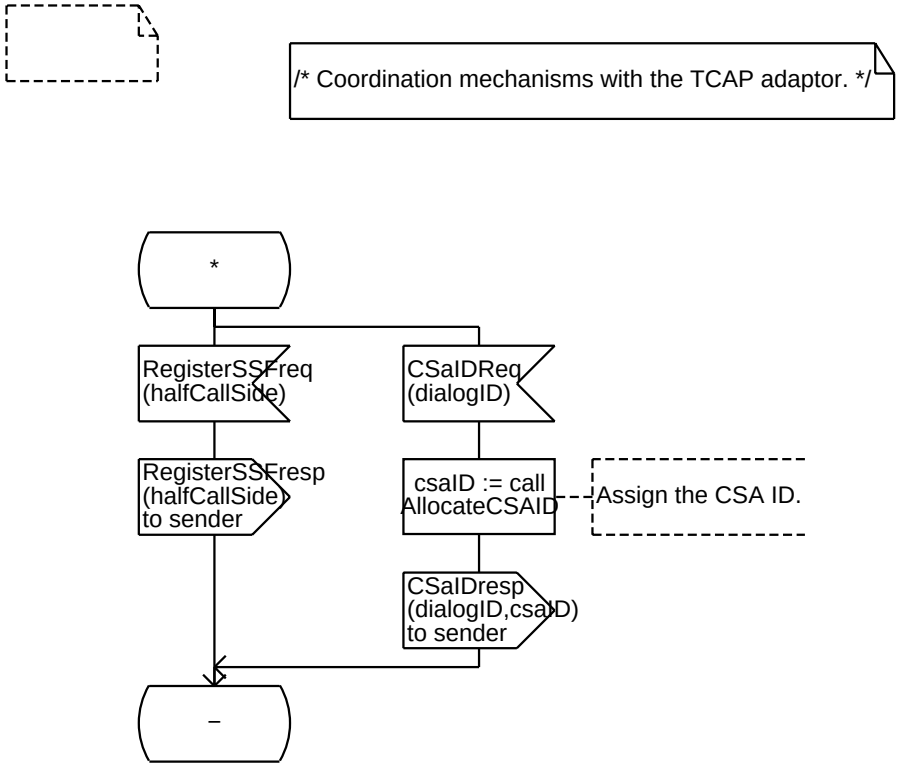


Annex A:

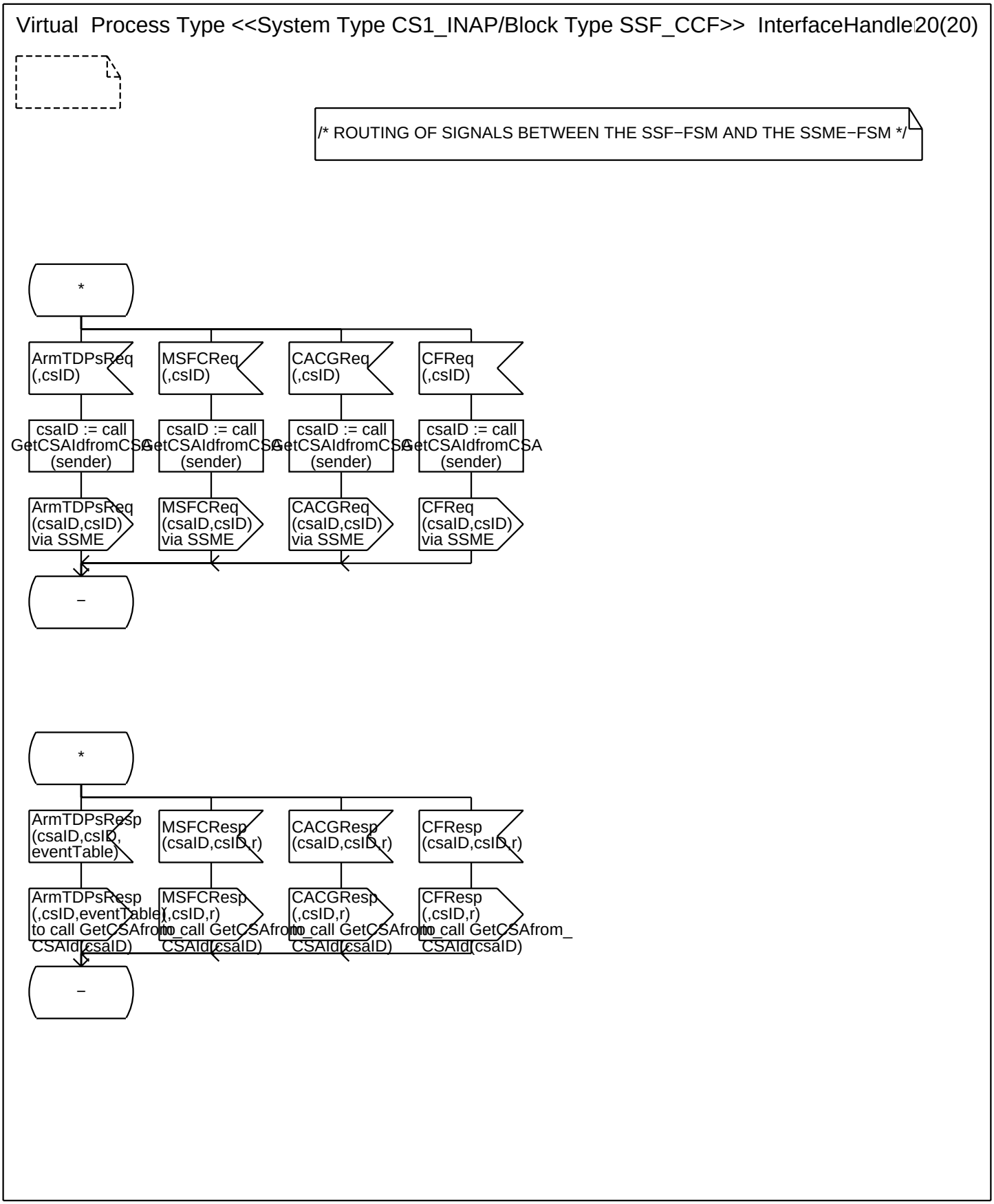

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 18(20)
/* ROUTING OF INAP OPERATIONS FROM SSF TO SCF. */
Idle
ApplyCharging_ Report (,acrArg,r)
AssistRequest_ Instructions (,ariArg)
CallInformation_ Report (,cirArg,r)
Event_ Notification_ Charging (,encArg)
EventReport_ BCSM (,erbcsmArg)
InitialDP (,idpArg)
csaID := call GetCSAIdfrom_ CSA(Sender)
idpArg!created_ CallSegment_ Association:=call GetCSAIdfrom_ CSA(Sender)
ApplyCharging_ Report (csaID,acrArg,r) via SCF
AssistRequest_ Instructions (csaID,ariArg) via SCF
CallInformation_ Report(csaID,cirArg, r) via SCF
EventNotification_ Charging (csaID,encArg) via SCF
EventReport_ BCSM(csaID, erbcsmArg) via SCF
InitialDP (csaID,idpArg) via SCF
–
Idle
ActivityTestResult (invokeID,dID)
Service_ Filtering_ Response (,sfrArg)
ActivityTestResult (invokeID,dID) via SCF
ServiceFiltering_ Response (,sfrArg) via SCF
–

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 19(20)

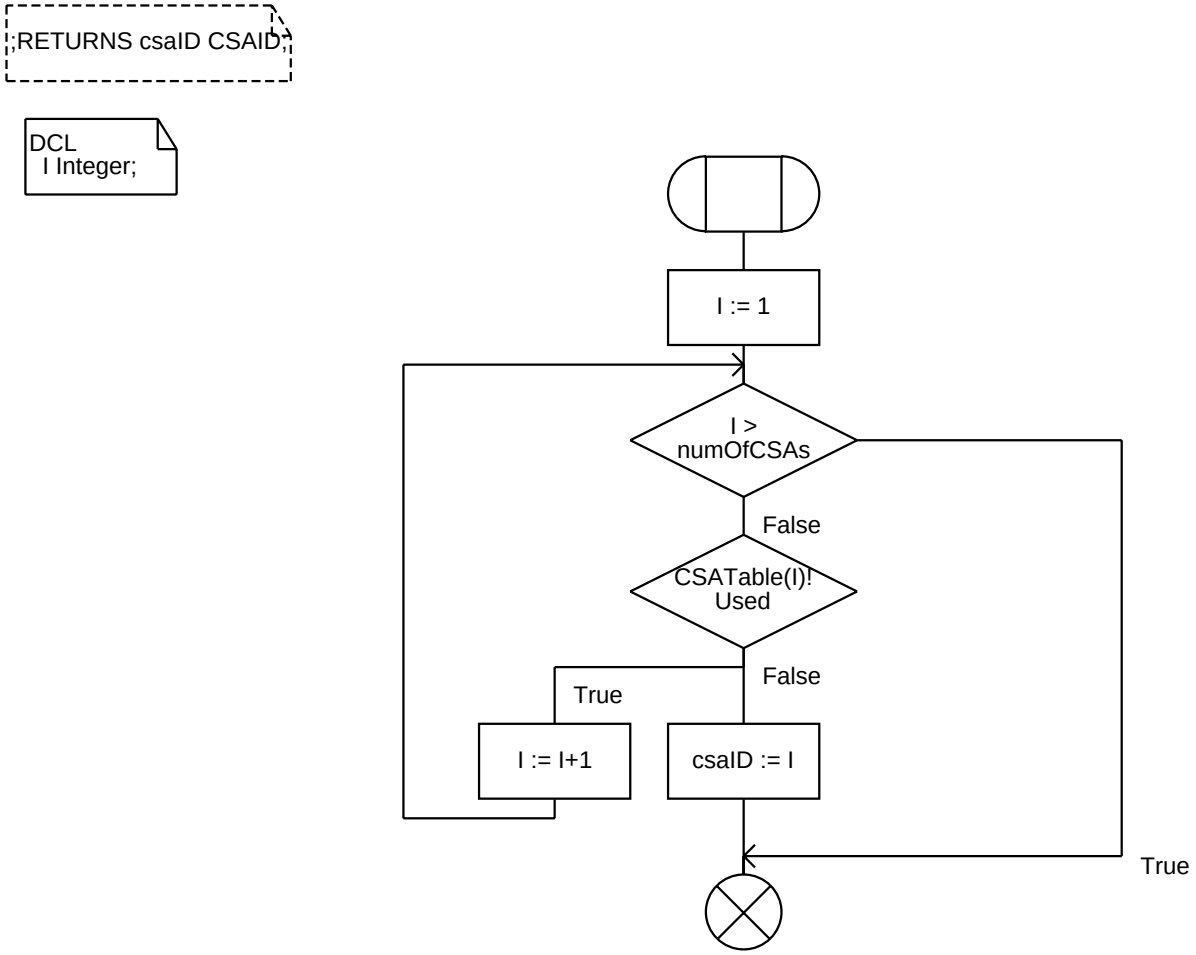


Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> InterfaceHandle 20(20)
/* ROUTING OF SIGNALS BETWEEN THE SSF−FSM AND THE SSME−FSM */
*
ArmTDPsReq (,csID)
MSFCReq (,csID)
CACGReq (,csID)
CFReq (,csID)
csaID := call GetCSAIdfromCSA (sender)
csaID := call GetCSAIdfromCSA (sender)
csaID := call GetCSAIdfromCSA (sender)
csaID := call GetCSAIdfromCSA (sender)
ArmTDPsReq (csaID,csID) via SSME
MSFCReq (csaID,csID) via SSME
CACGReq (csaID,csID) via SSME
CFReq (csaID,csID) via SSME
−
*
ArmTDPsResp (csaID,csID, eventTable)
MSFCResp (csaID,csID,r)
CACGResp (csaID,csID,r)
CFResp (csaID,csID,r)
ArmTDPsResp (,csID,eventTable) to call GetCSAfrom_CSAId(csaID)
MSFCResp (,csID,r) to call GetCSAfrom_CSAId(csaID)
CACGResp (,csID,r) to call GetCSAfrom_CSAId(csaID)
CFResp (,csID,r) to call GetCSAfrom_CSAId(csaID)
−

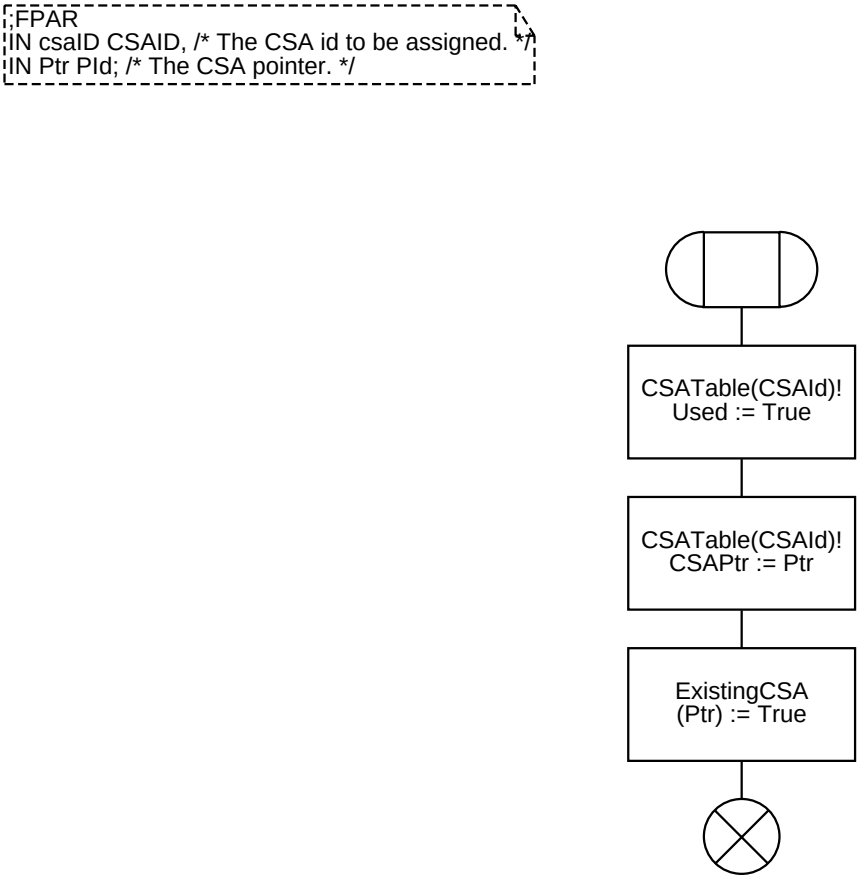
Annex A:

Procedure AllocateCSAId 1(1)

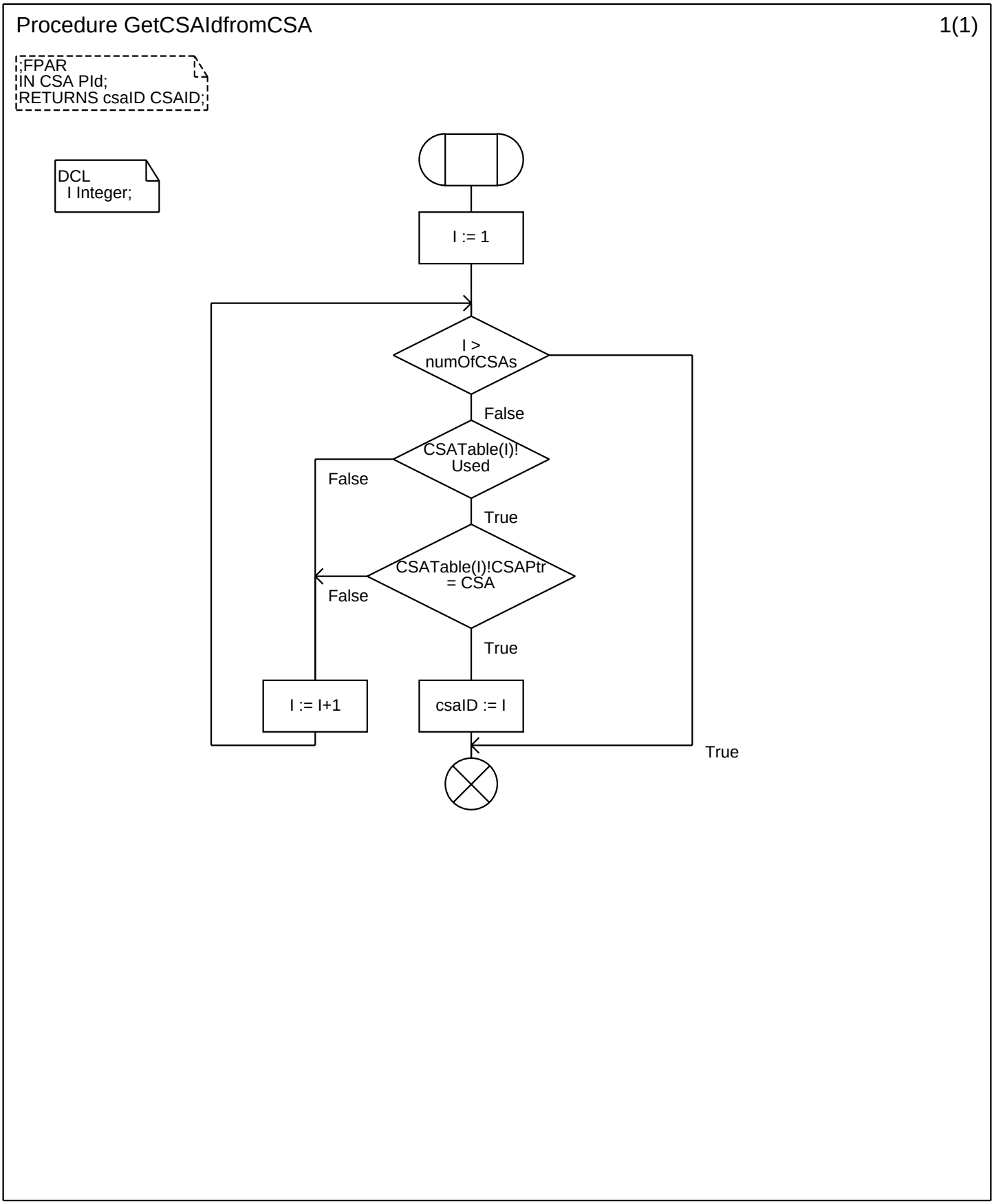


Annex A:

Procedure AddCSAId 1(1)

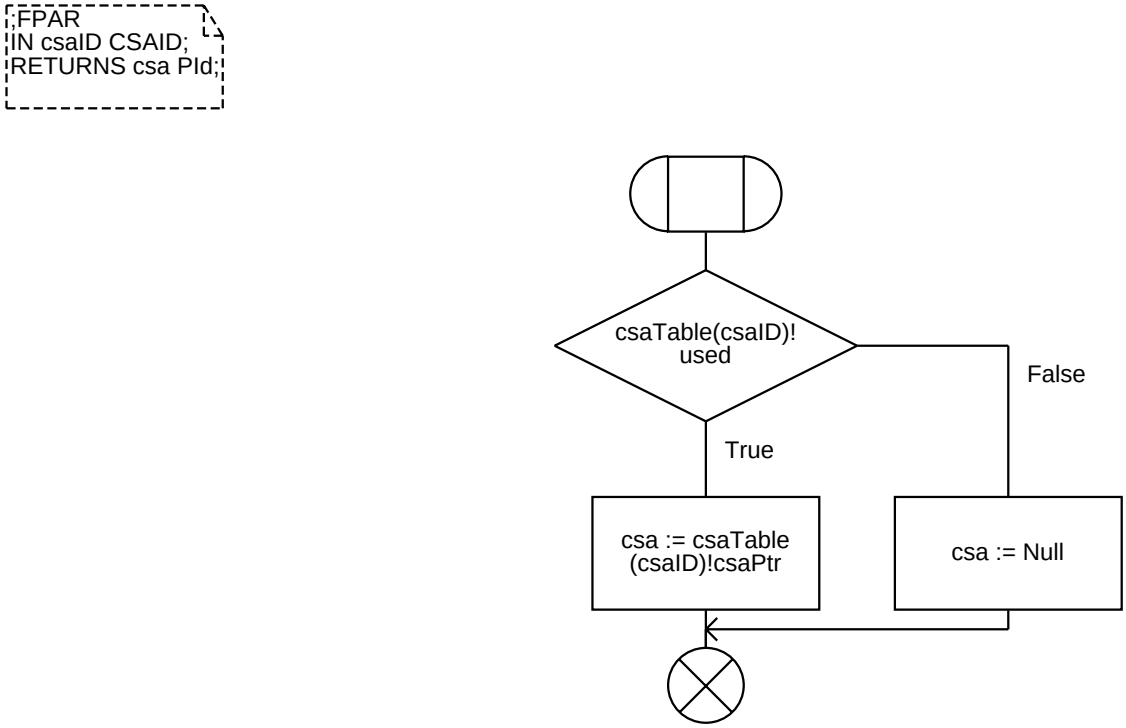


Annex A:


Procedure GetCSAIdfromCSA
1(1)
;FPAR
IN CSA PId;
RETURNS csaID CSAID;
DCL
I Integer;
I := 1
I > numOfCSAs
False
CSATable(I)! Used
False
True
CSATable(I)!CSAPtr = CSA
False
True
I := I+1
csaID := I
True

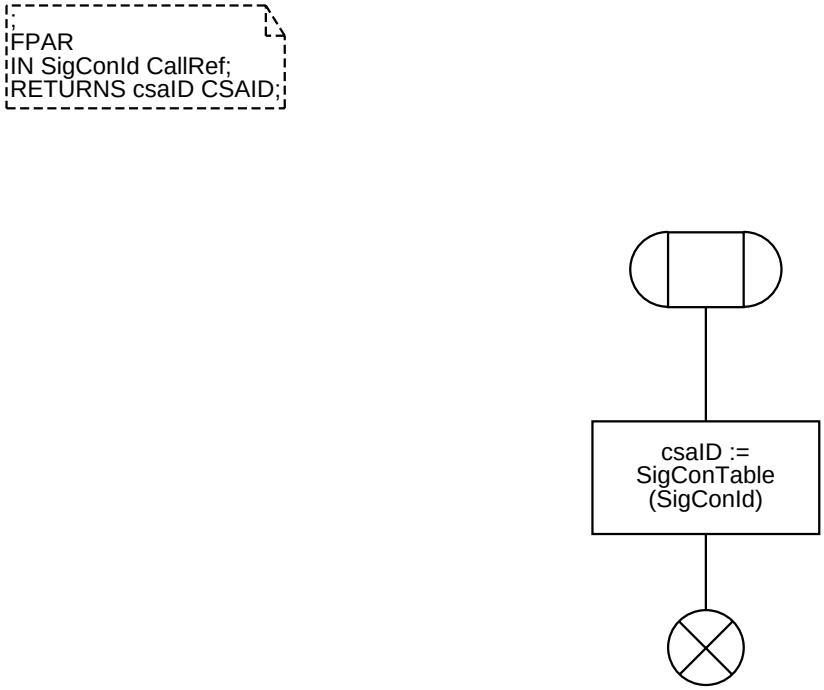
Annex A:

Procedure GetCSAfromCSAId 1(1)

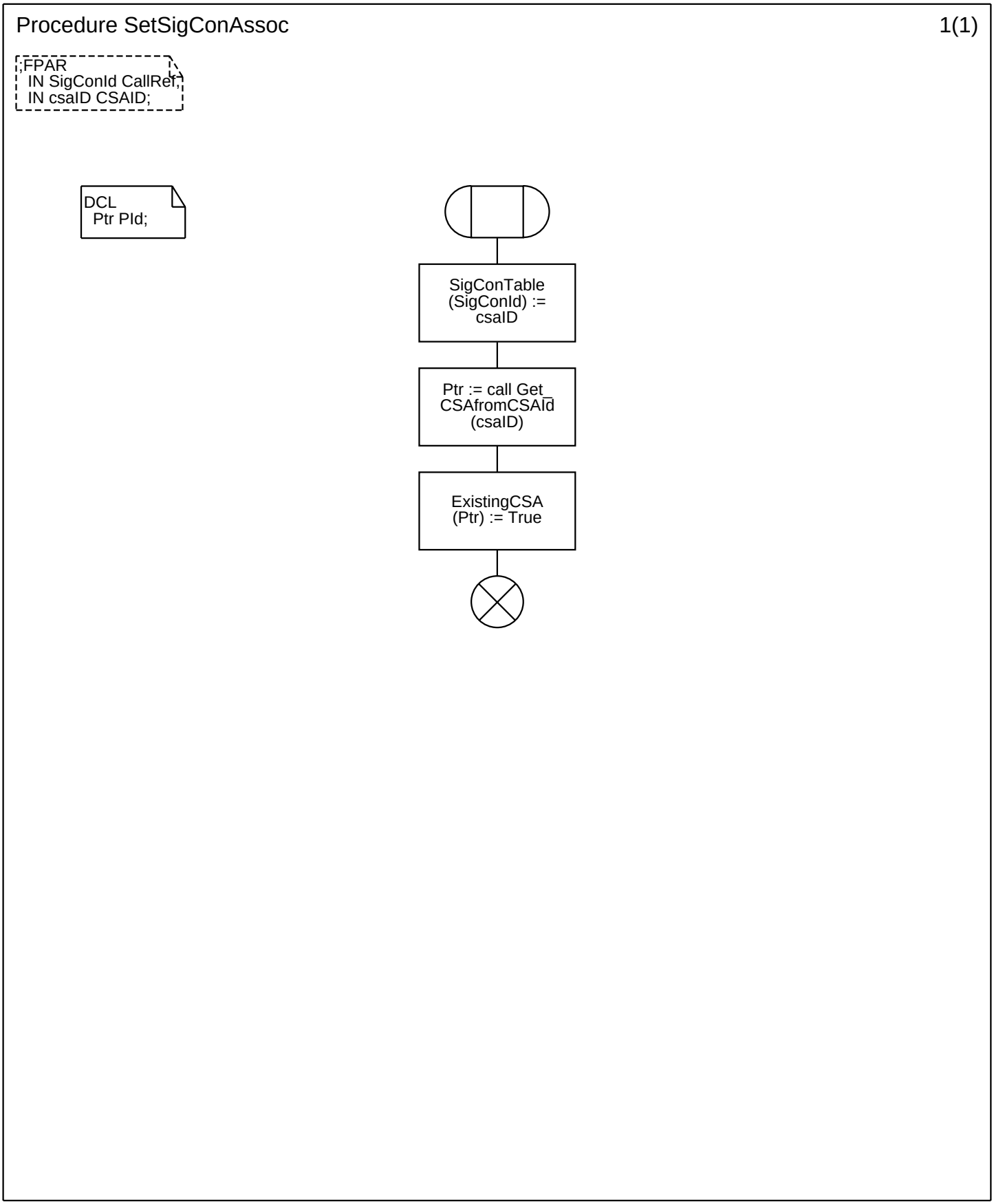


Annex A:

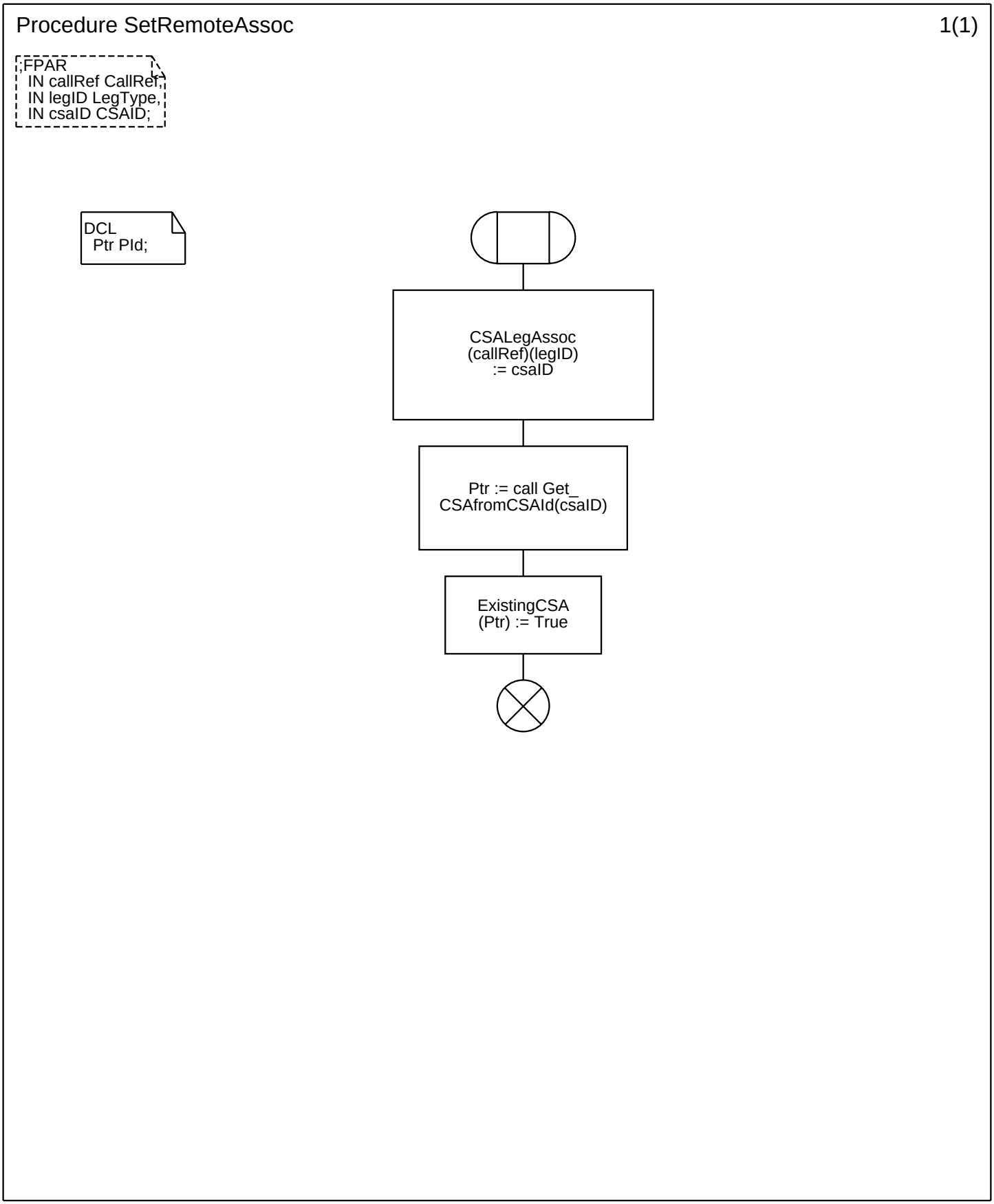
Procedure GetCSAfromSigConId 1(1)



Annex A:

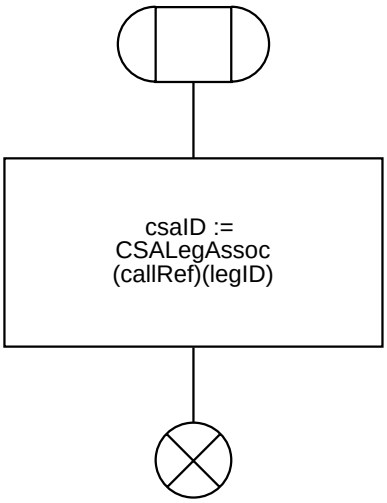

Procedure SetSigConAssoc
1(1)
;FPAR
IN SigConId CallRef;
IN csaID CSAID;
DCL
Ptr PId;
SigConTable
(SigConId) :=
csaID
Ptr := call Get_
CSAfromCSAId
(csaID)
ExistingCSA
(Ptr) := True

Annex A:


Procedure SetRemoteAssoc
1(1)
;FPAR
IN callRef CallRef,
IN legID LegType,
IN csaID CSAID;
DCL
Ptr PId;
CSALegAssoc
(callRef)(legID)
:= csaID
Ptr := call Get_
CSAfromCSAId(csaID)
ExistingCSA
(Ptr) := True

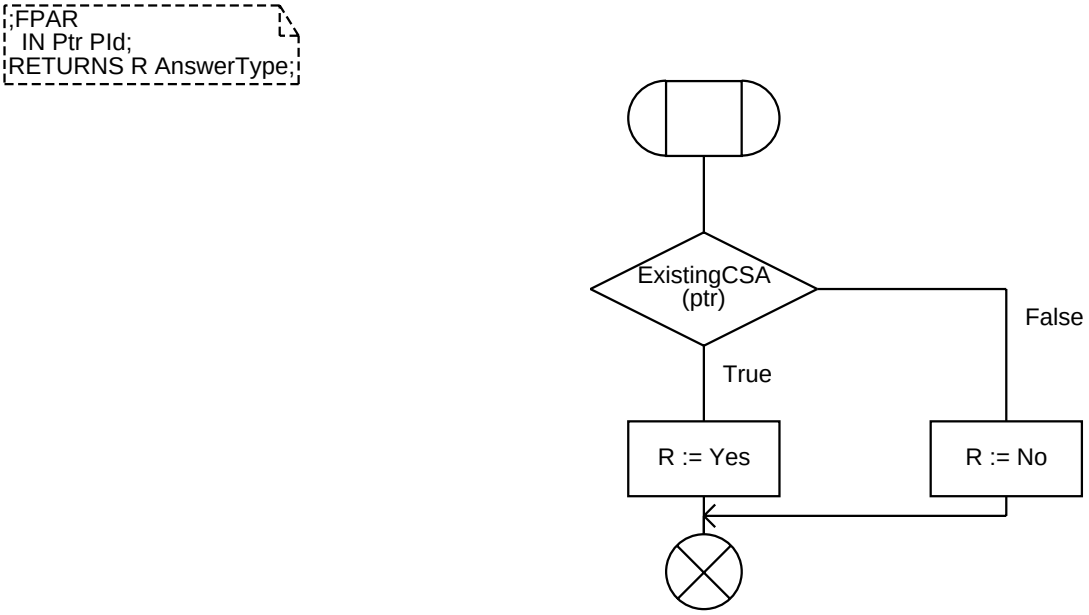
Annex A:

Procedure GetCSAfromRemoteId 1(1)



Annex A:

Procedure IsCSA 1(1)



Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso1(20)n

```
/**** 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;
```

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso2(20)n

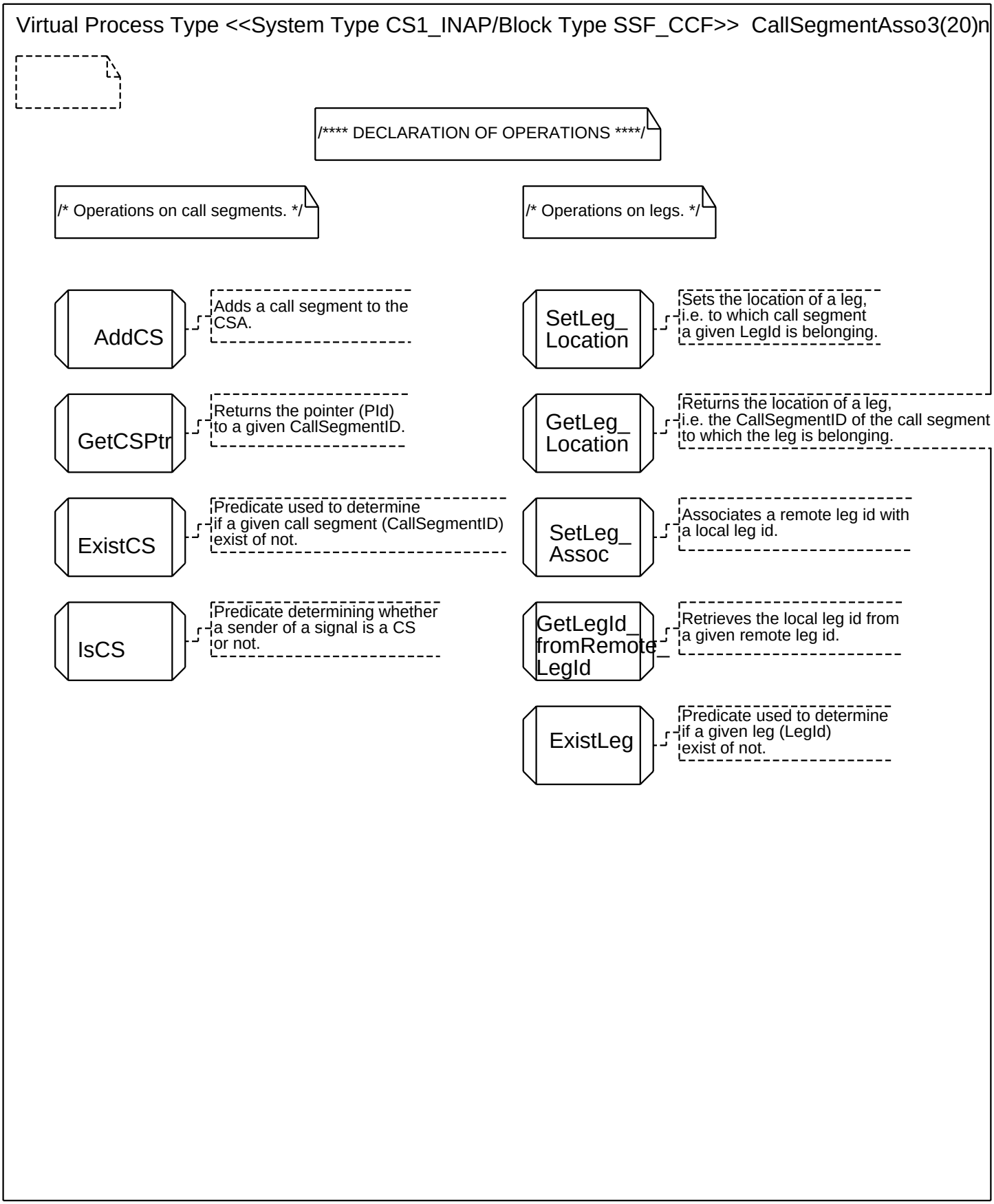
/**** VARIABLE DECLARATIONS ****/

```
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;
```

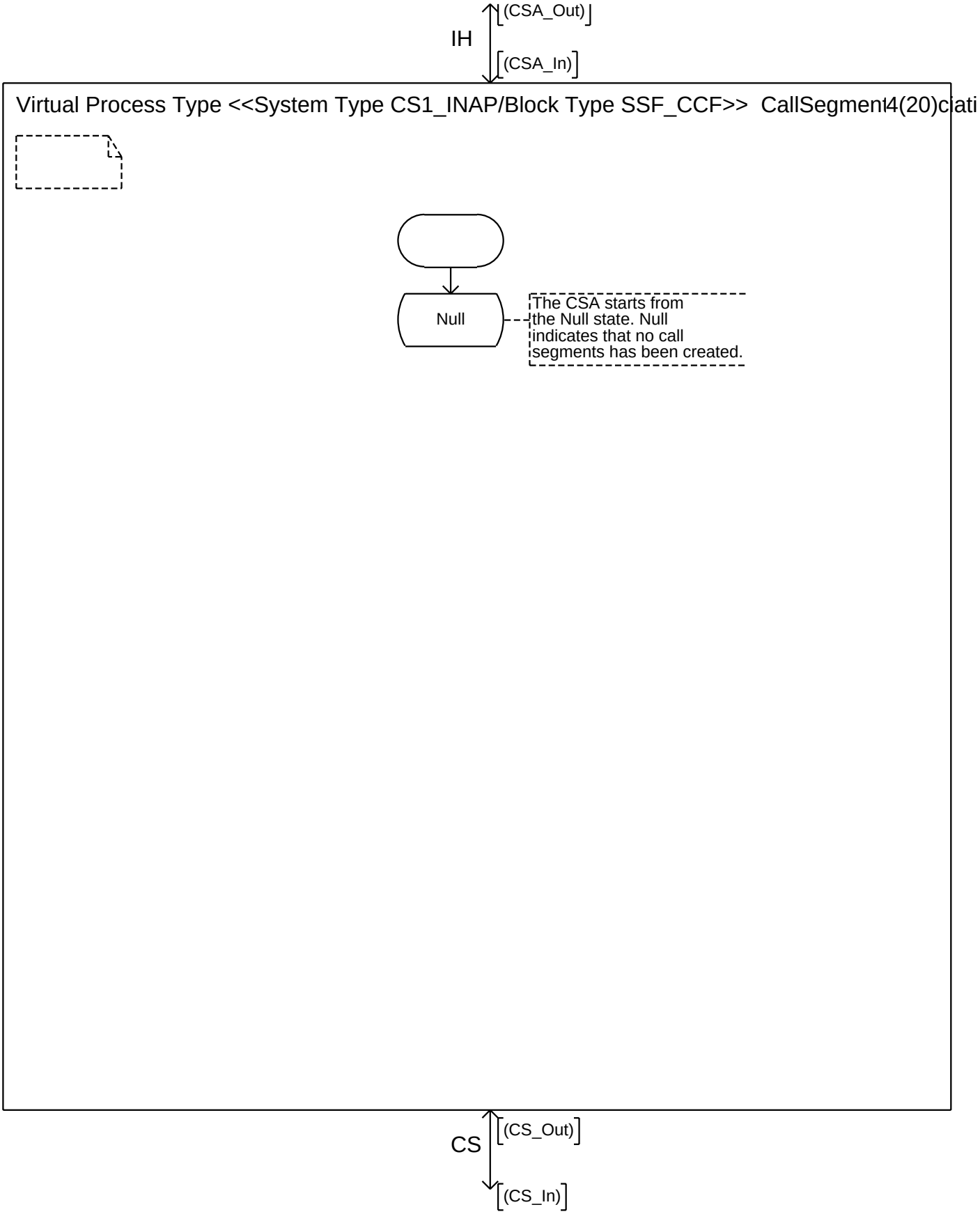
```
DCL
 /* INCS−1 operation arguments. */
 ACArg ApplyChargingArg,
 ACRArg ApplyChargingReportArg,
 ARIArg AssistRequestInstructionsArg,
 CIRArg CallInformationReportArg,
 CIRqArg CallInformationRequestArg,
 CArg CancelArg,
 CIArg CollectInformationArg,
 CoArg ConnectArg,
 CTRArg ConnectToResourceArg,
 CtArg ContinueWithArgumentArg,
 DFCArg DisconnectForwardConnectionWithArgumentArg,
 ETCArg EstablishTemporaryConnectionArg,
 ENCArg EventNotificationChargingArg,
 ERBCSMArg EventReportBCSMArg,
 FCIArg FurnishChargingInformationArg,
 IDPArg InitialDPArg,
 ICAArg InitiateCallAttemptArg,
 RCArg ReleaseCallArg,
 RNCEARg RequestNotificationChargingEventArg,
 RRBCSMEArg RequestReportBCSMEventArg,
 RTARg ResetTimerArg,
 SCIArg SendChargingInformationArg,
 IEArg ErrorArg;
```

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

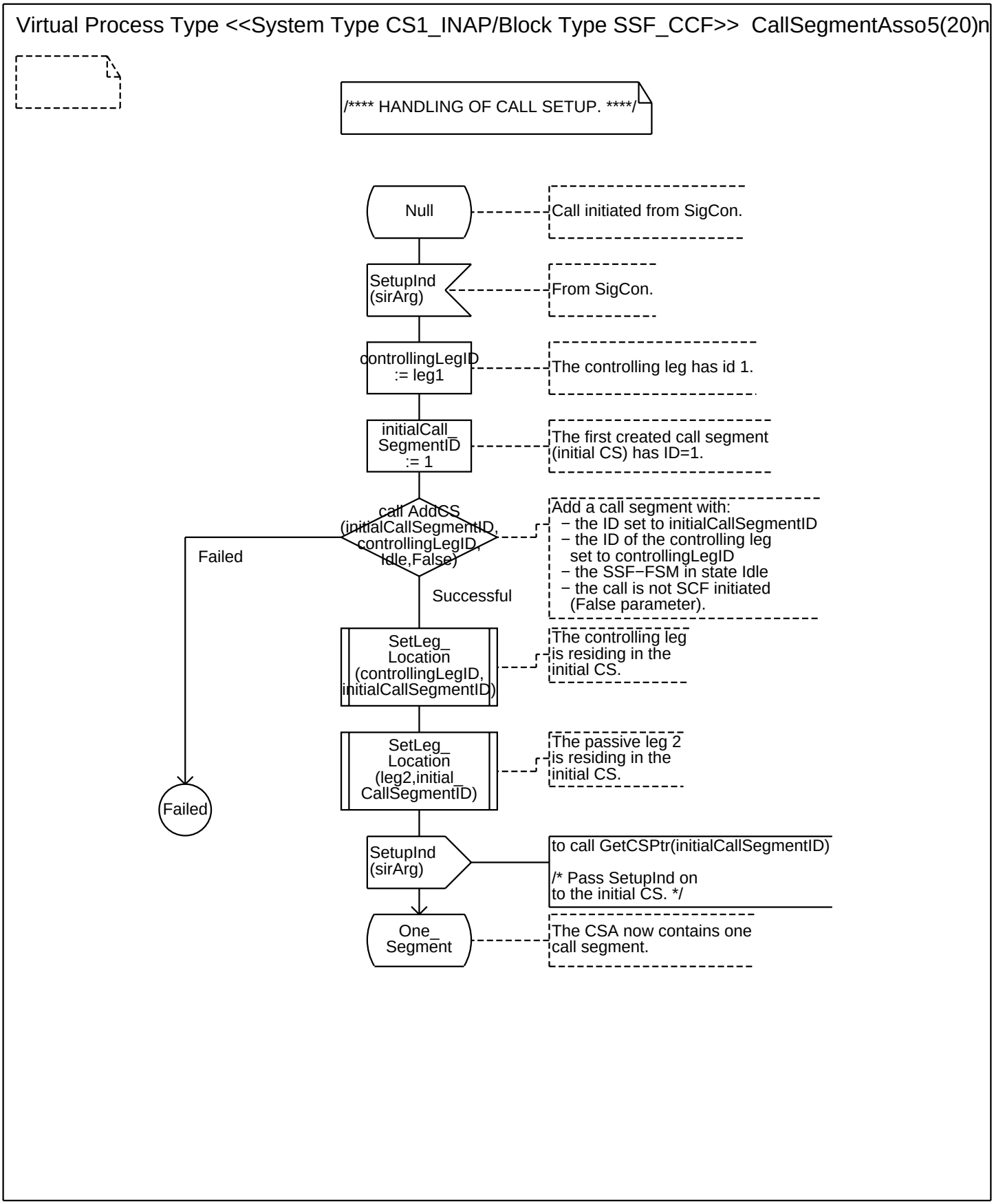
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso3(20)n
/**** 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.

Annex A:

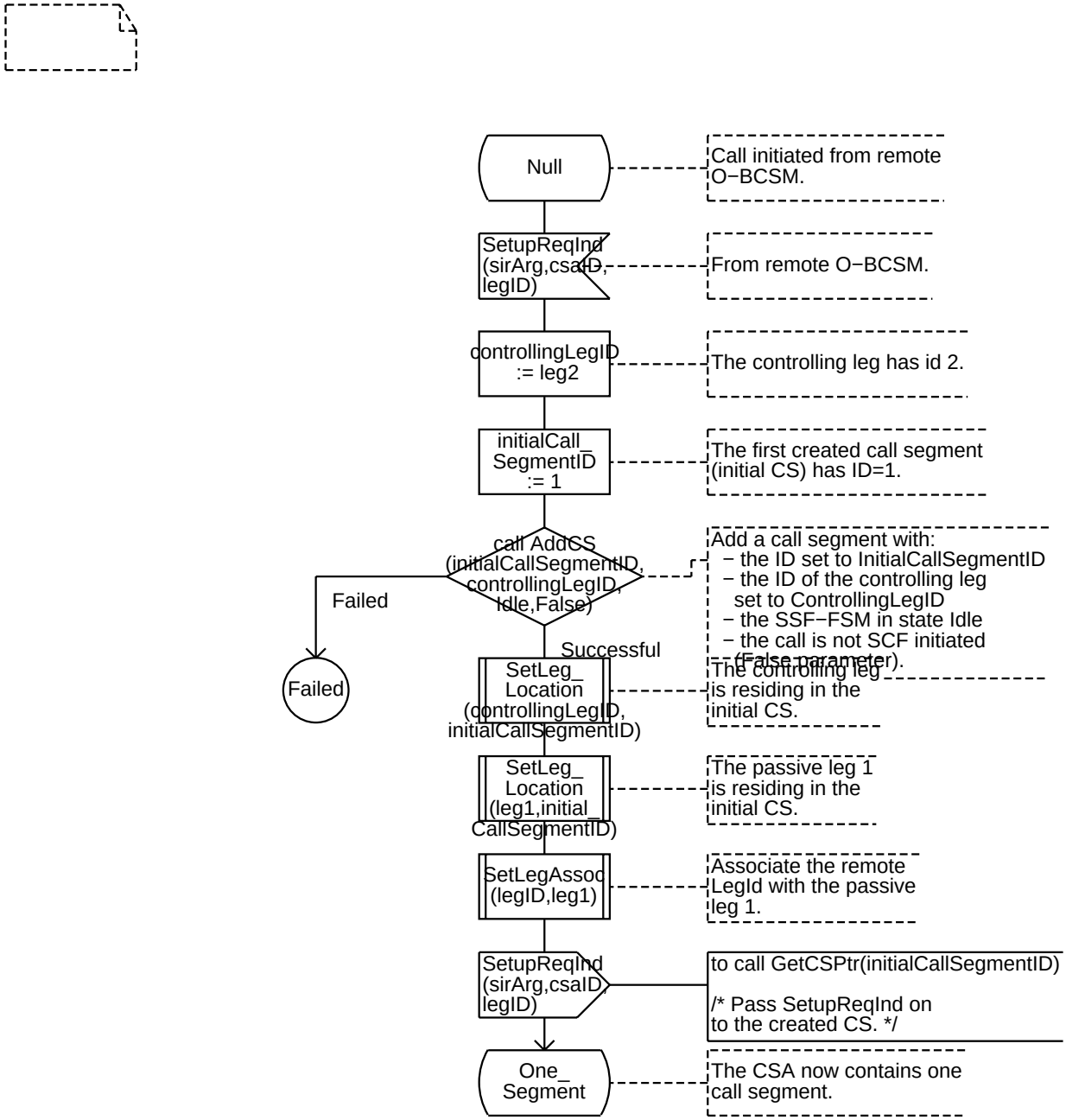

IH
(CSA_Out)
(CSA_In)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment4(20)ciati
Null
The CSA starts from the Null state. Null indicates that no call segments has been created.
CS
(CS_Out)
(CS_In)

Annex A:

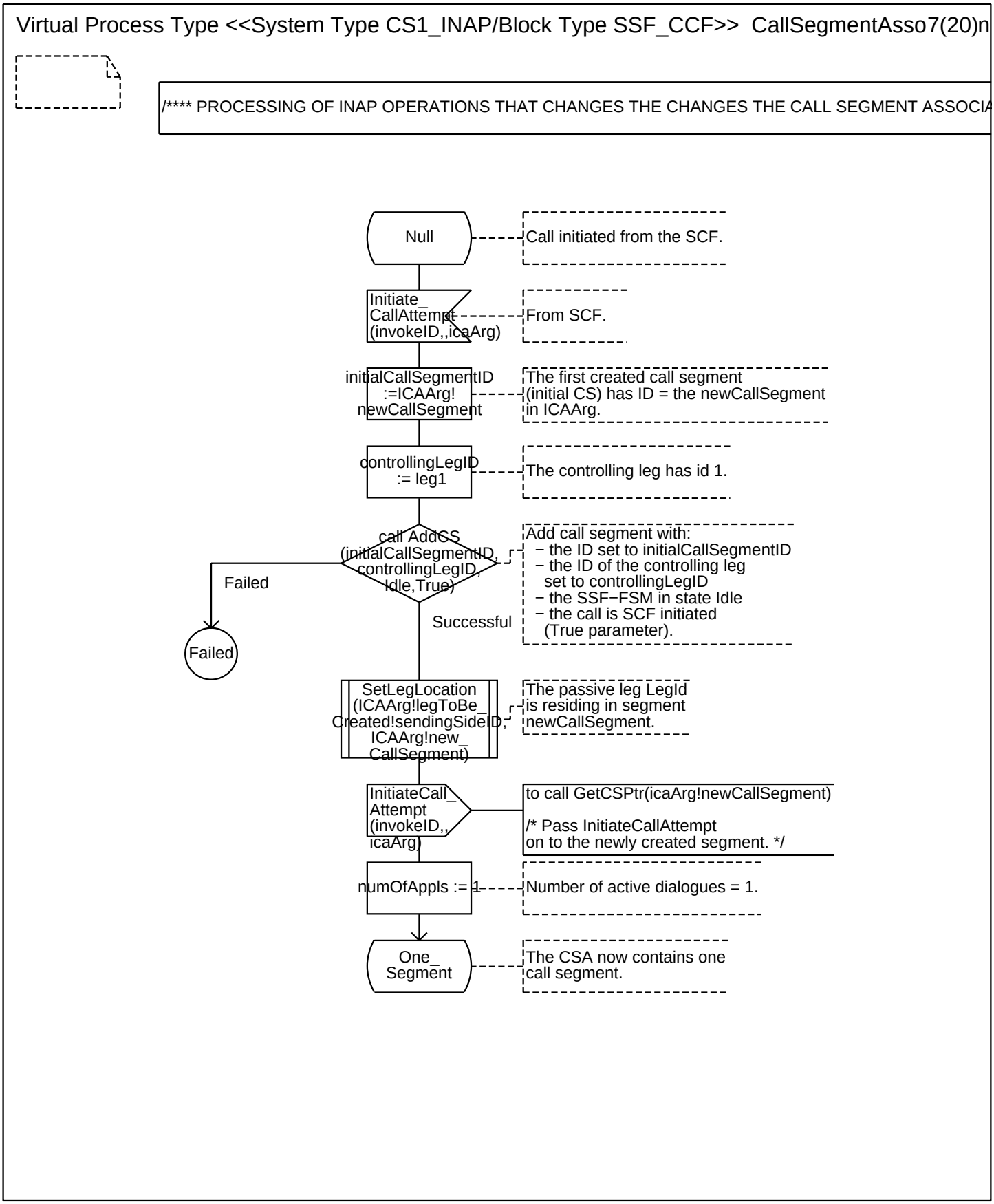

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso5(20)n
/**** HANDLING OF CALL SETUP. ****/
Null
Call initiated from SigCon.
SetupInd (sirArg)
From SigCon.
controllingLegID := leg1
The controlling leg has id 1.
initialCall_ SegmentID := 1
The first created call segment (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.

Annex A:

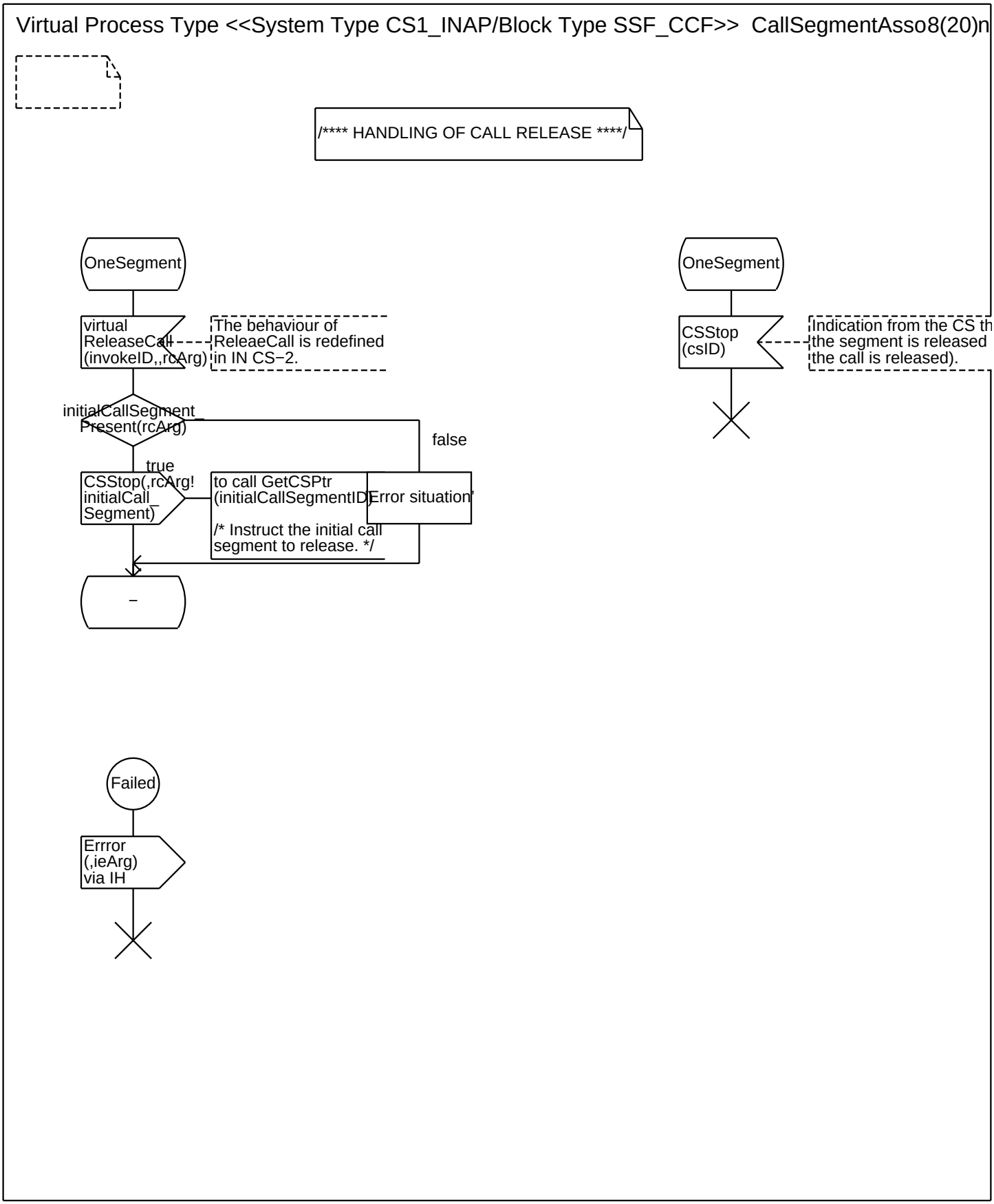
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso6(20)n



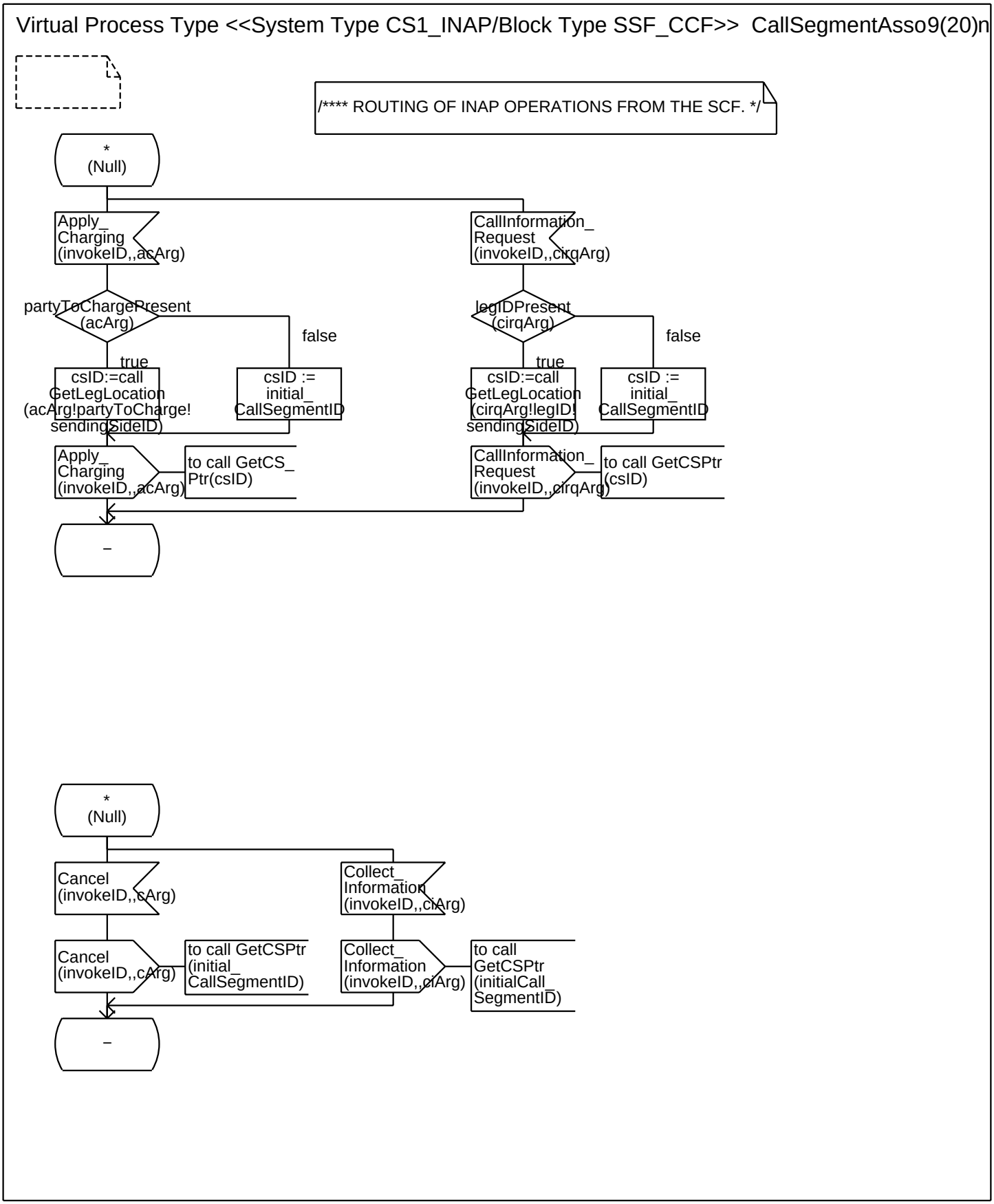
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso7(20)n
/**** PROCESSING OF INAP OPERATIONS THAT CHANGES THE CHANGES THE CALL SEGMENT ASSOCIA
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
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.

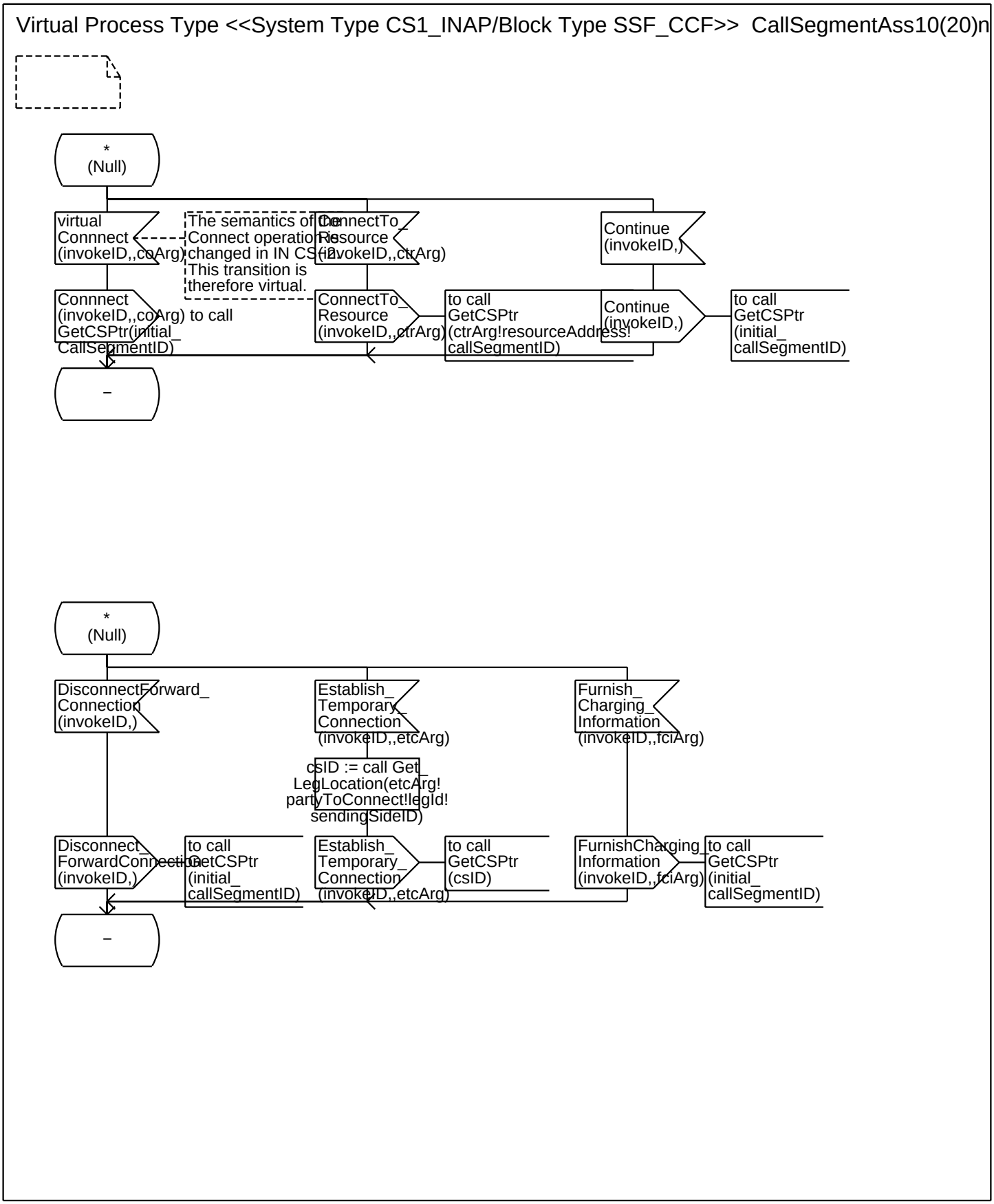
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso8(20)n
/**** 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)
Error situation
/* Instruct the initial call segment to release. */
–
OneSegment
CSStop (csID)
Indication from the CS th
the segment is released
the call is released).
Failed
Errror (,ieArg) via IH

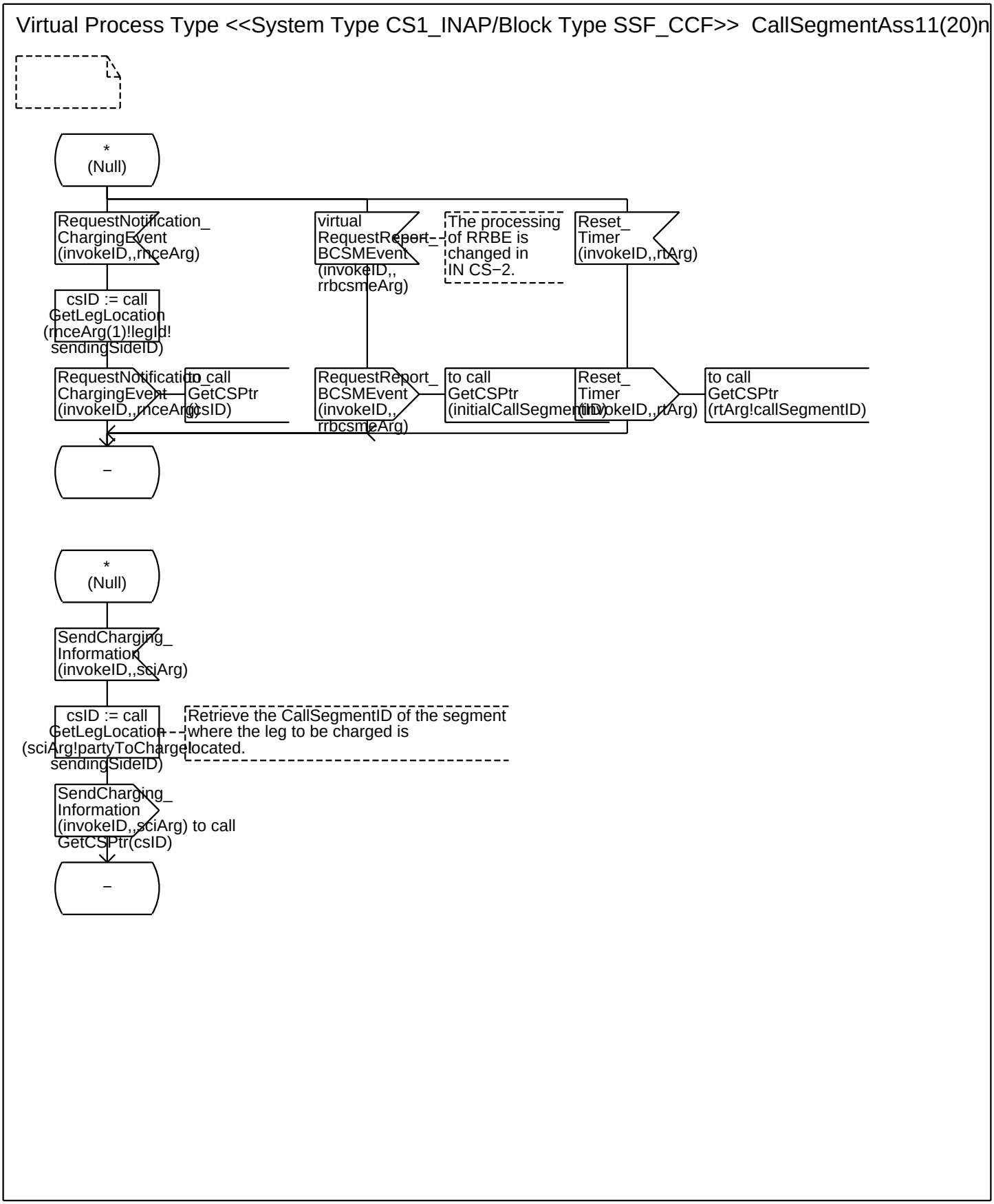
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAsso9(20)n
/**** 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)
–

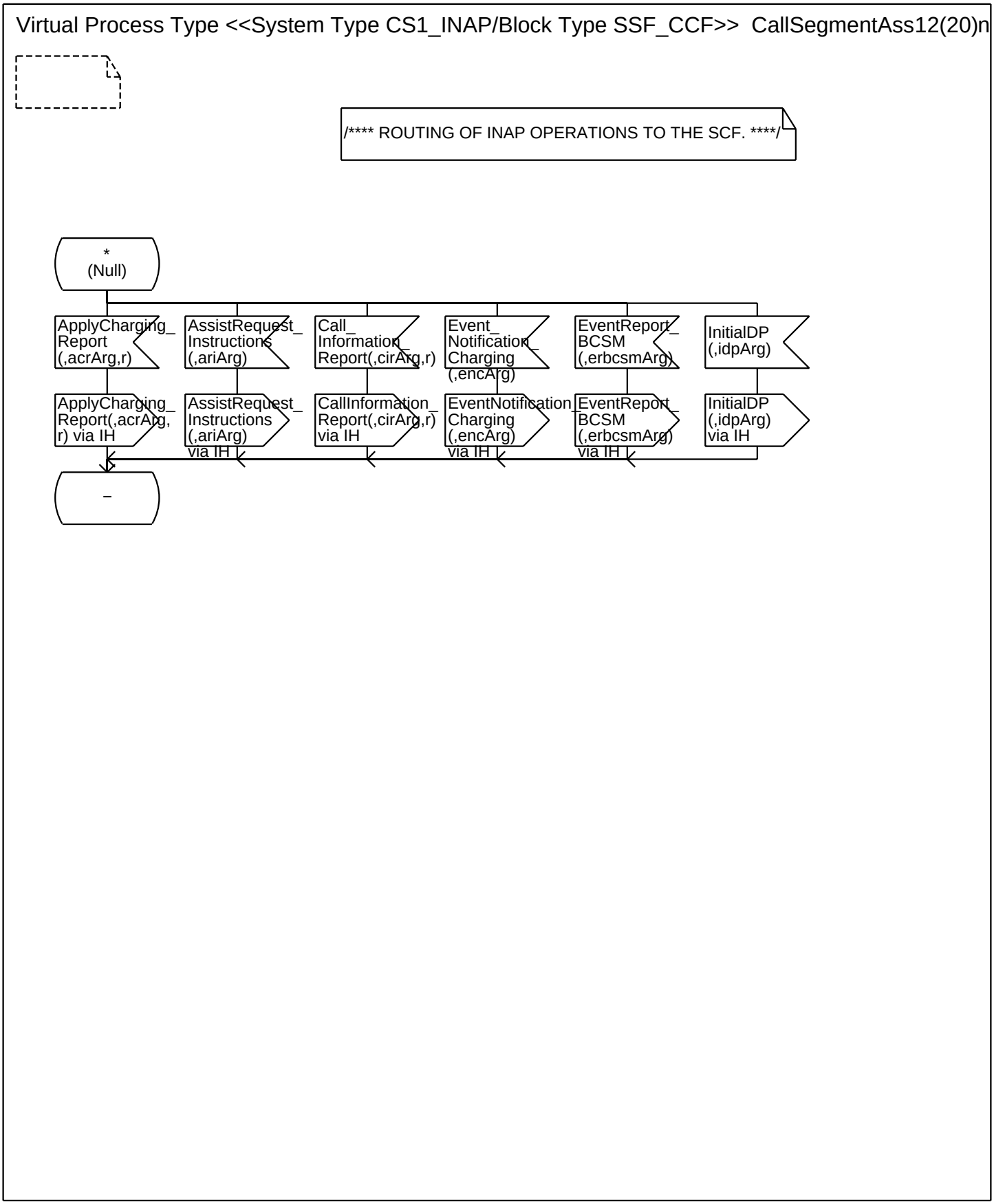
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss10(20)n
*
(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)
–

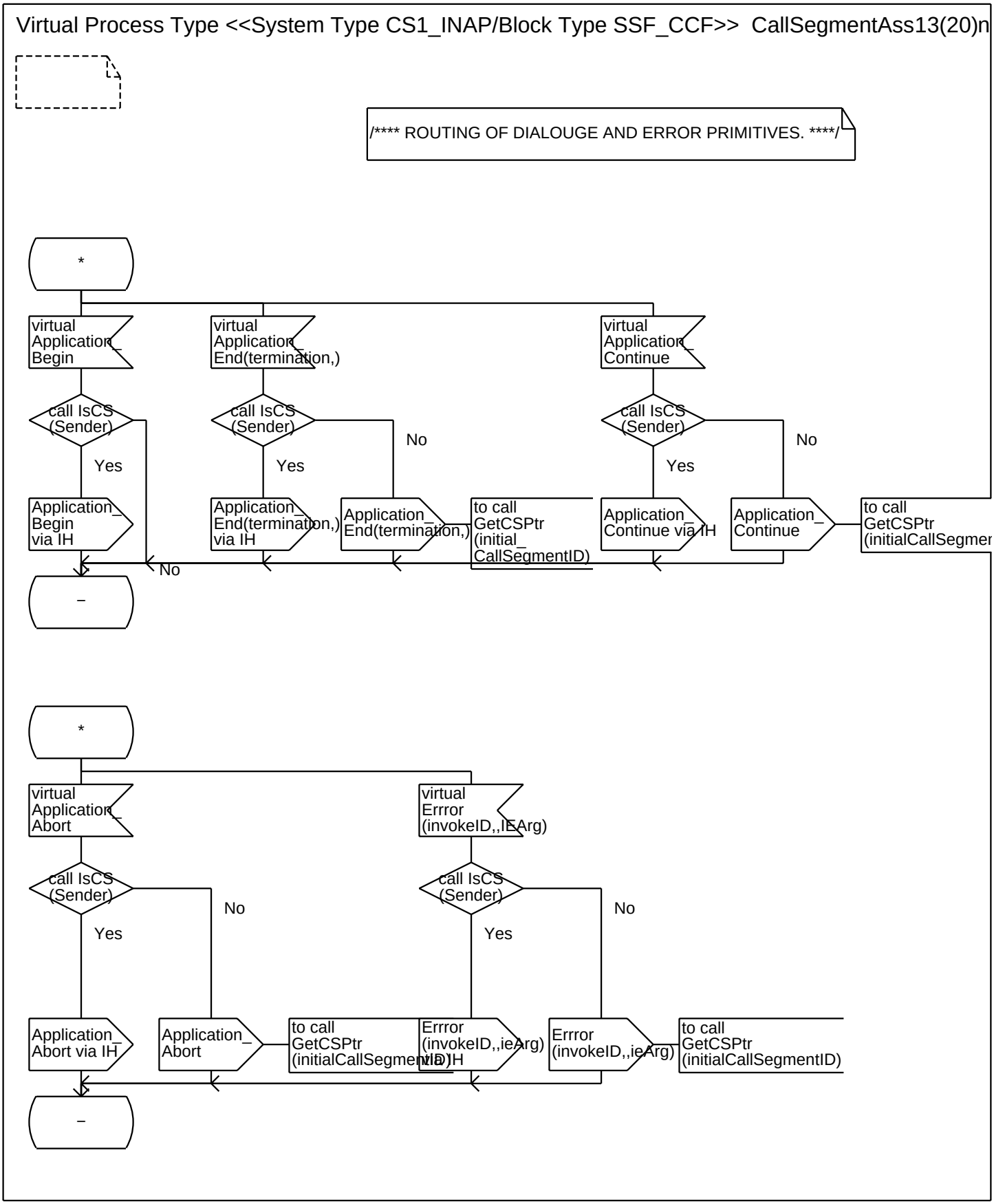
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss11(20)n
*
(Null)
RequestNotification_
ChargingEvent
(invokeID,,rnceArg)
virtual
RequestReport_
BCSMEvent
(invokeID,,
rrbcsmeArg)
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,,
rrbcsmeArg)
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)
–

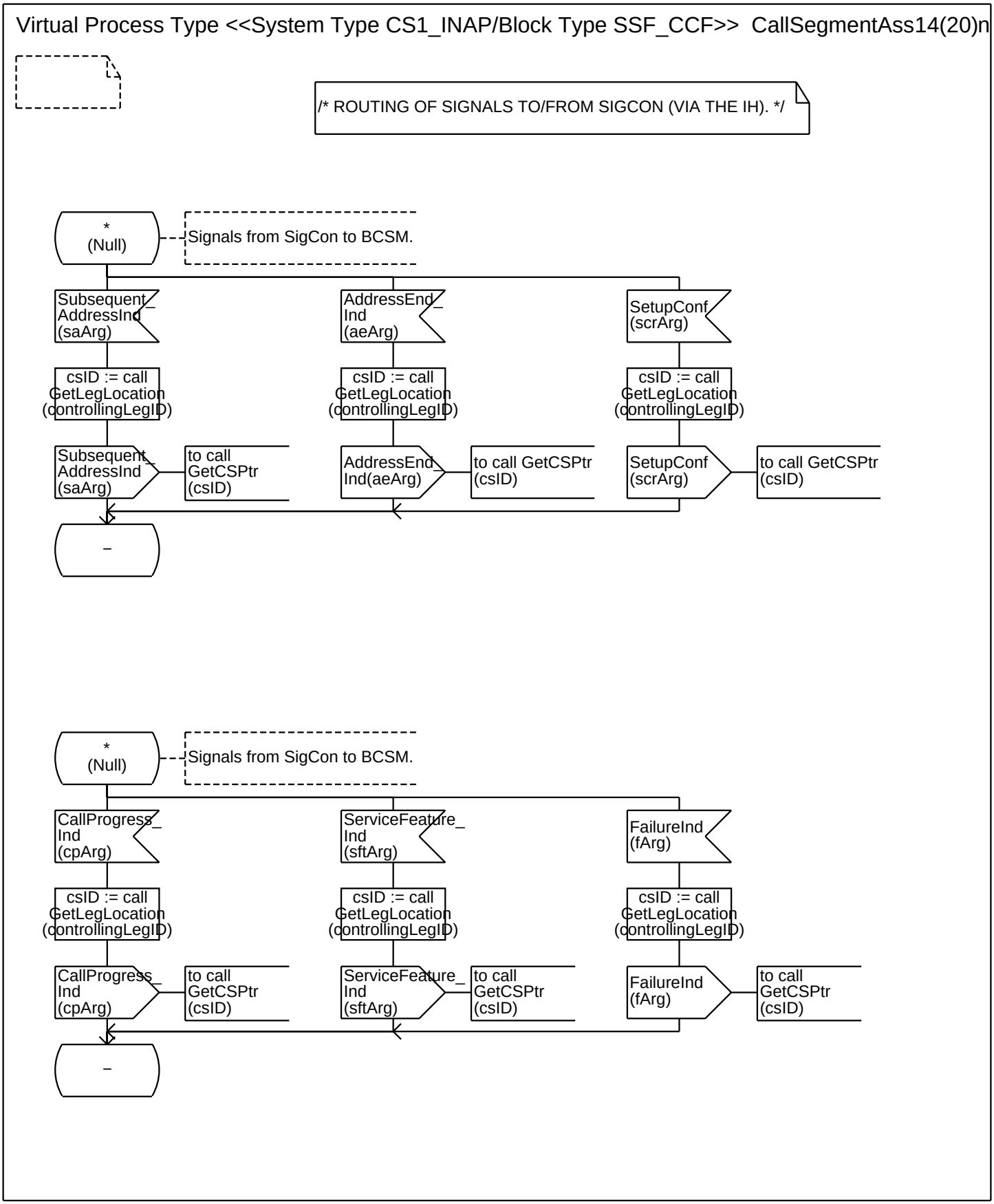
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss12(20)n
/**** 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
–

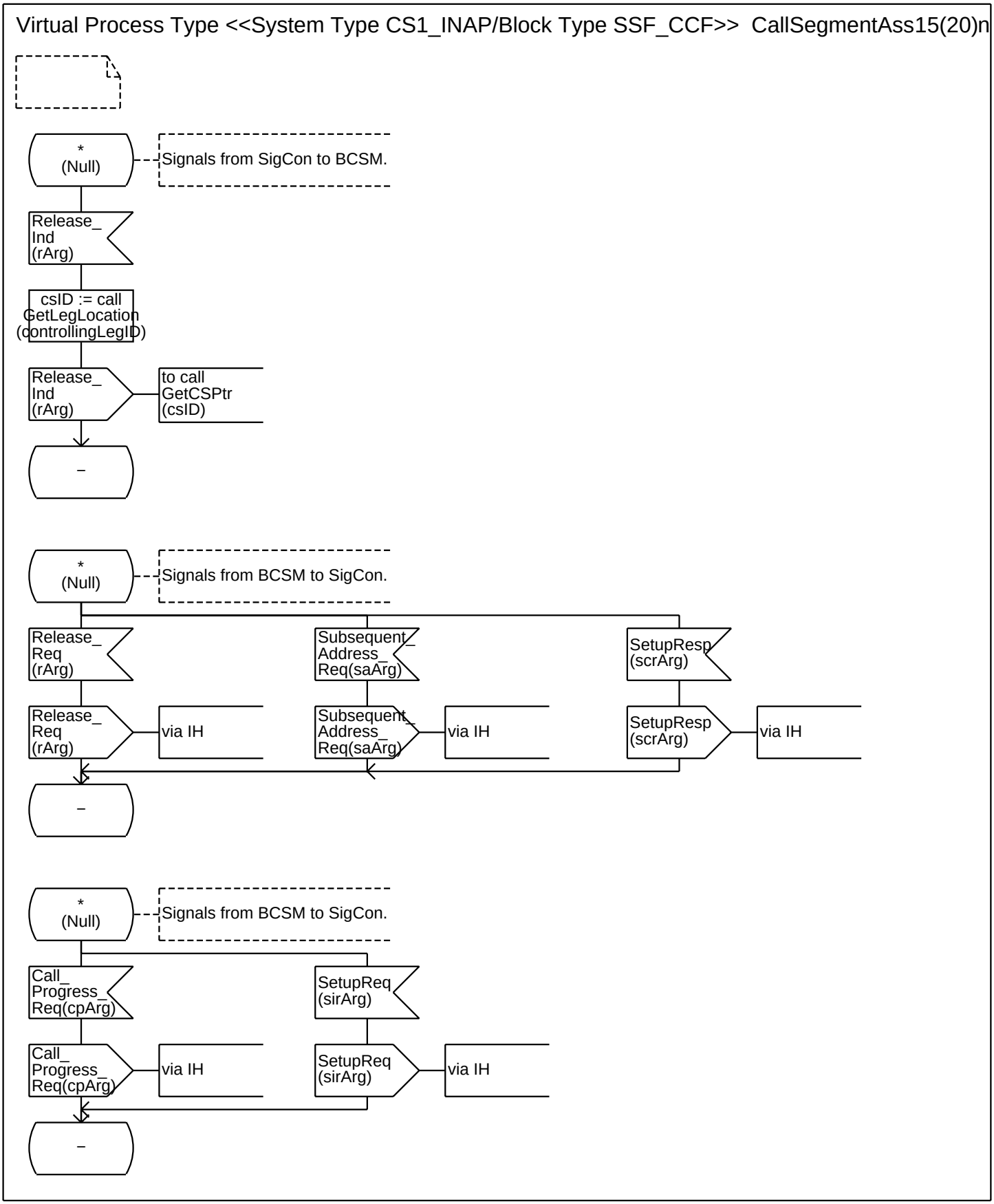
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss13(20)n
/**** ROUTING OF DIALOUGE AND ERROR PRIMITIVES. ****/
*
virtual Application_ Begin
virtual Application_ End(termination,)
virtual Application_ Continue
call IsCS (Sender)
Yes
No
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 (initialCallSegmer
–
*
virtual Application_ Abort
virtual Errror (invokeID,,IEArg)
Application_ Abort via IH
Application_ Abort
to call GetCSPtr (initialCallSegmentID)
Errror (invokeID,,ieArg) via IH
Errror (invokeID,,ieArg)
–

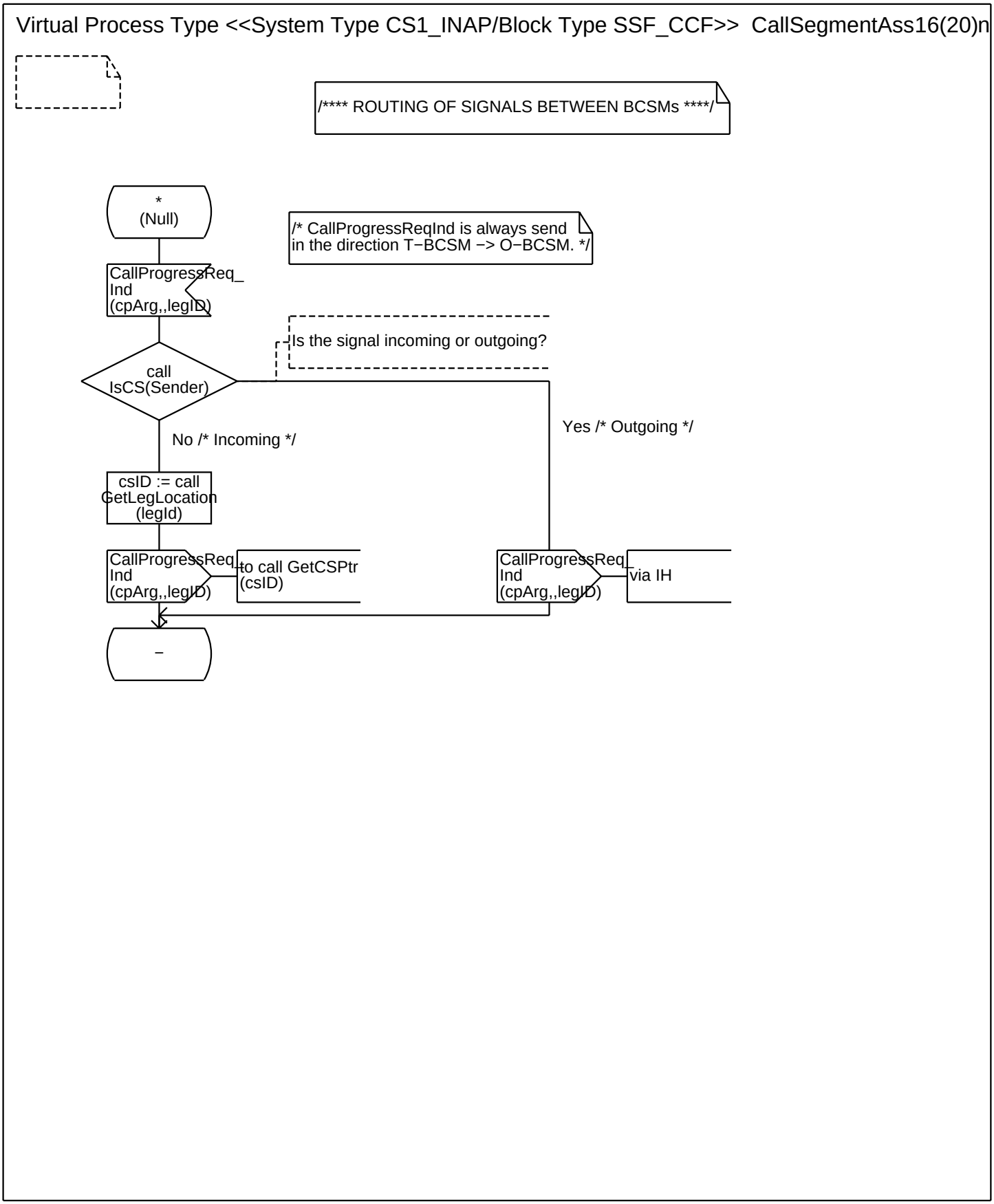
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss14(20)n
/* ROUTING OF SIGNALS TO/FROM SIGCON (VIA THE IH). */
*
(Null)
Signals from SigCon to BCSM.
Subsequent_
AddressInd
(saArg)
AddressEnd_
Ind
(aeArg)
SetupConf
(scrArg)
csID := call
GetLegLocation
(controllingLegID)
csID := call
GetLegLocation
(controllingLegID)
csID := call
GetLegLocation
(controllingLegID)
Subsequent_
AddressInd
(saArg)
to call
GetCSPtr
(csID)
AddressEnd_
Ind(aeArg)
to call GetCSPtr
(csID)
SetupConf
(scrArg)
to call GetCSPtr
(csID)
–
*
(Null)
Signals from SigCon to BCSM.
CallProgress_
Ind
(cpArg)
ServiceFeature_
Ind
(sftArg)
FailureInd
(fArg)
csID := call
GetLegLocation
(controllingLegID)
csID := call
GetLegLocation
(controllingLegID)
csID := call
GetLegLocation
(controllingLegID)
CallProgress_
Ind
(cpArg)
to call
GetCSPtr
(csID)
ServiceFeature_
Ind
(sftArg)
to call
GetCSPtr
(csID)
FailureInd
(fArg)
to call
GetCSPtr
(csID)
–

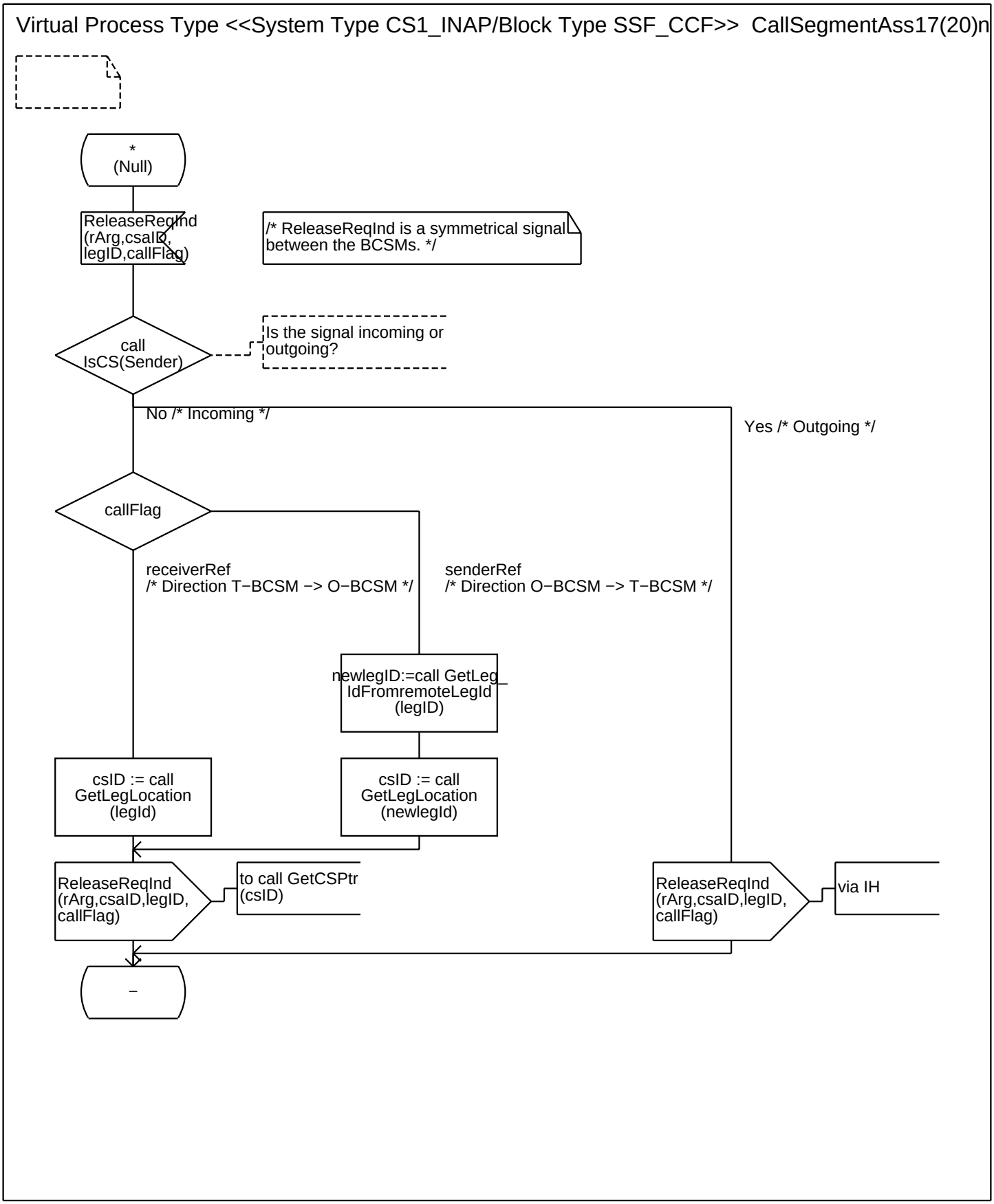
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss15(20)n
*
(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
–

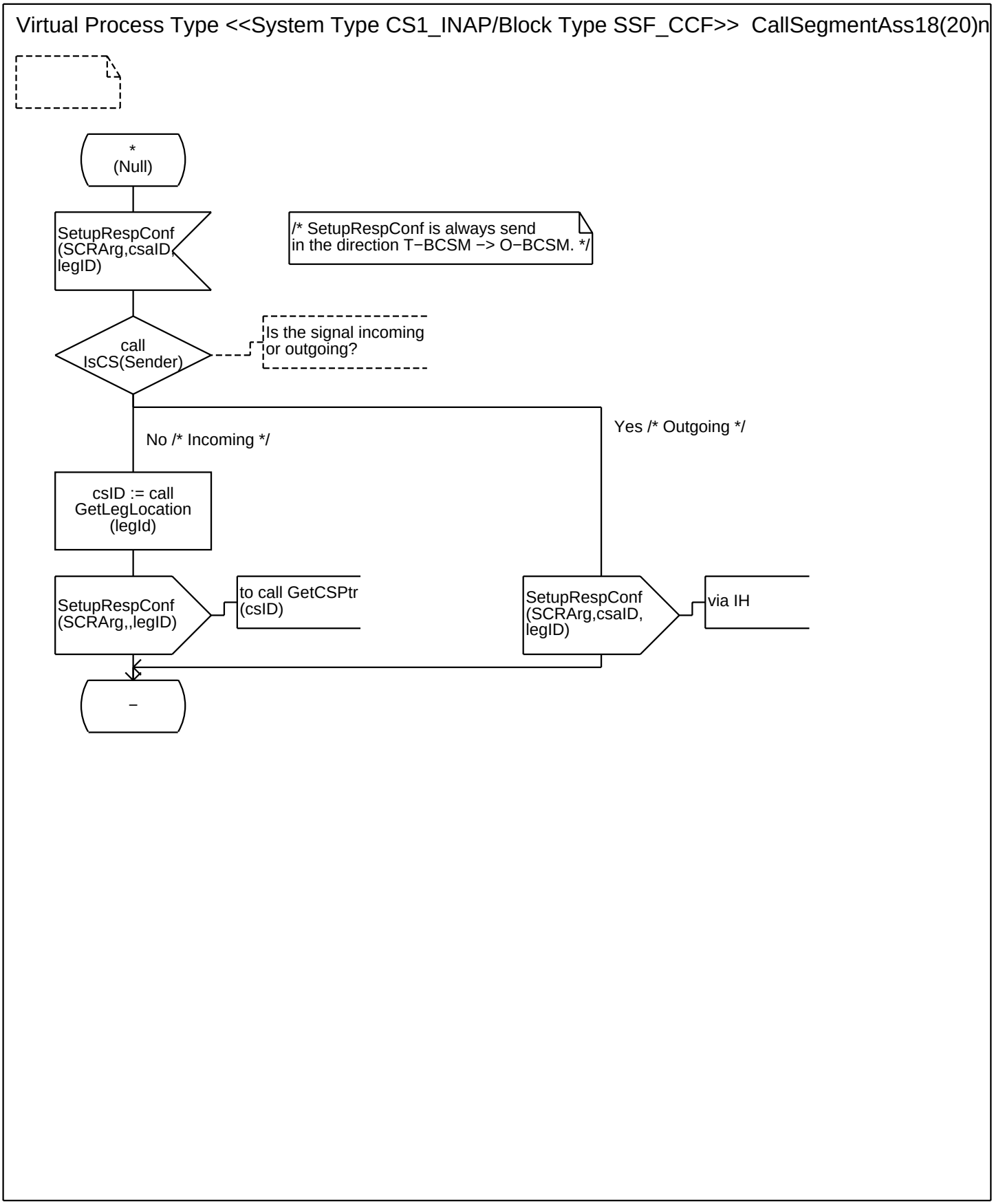
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss16(20)n
/**** 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
−

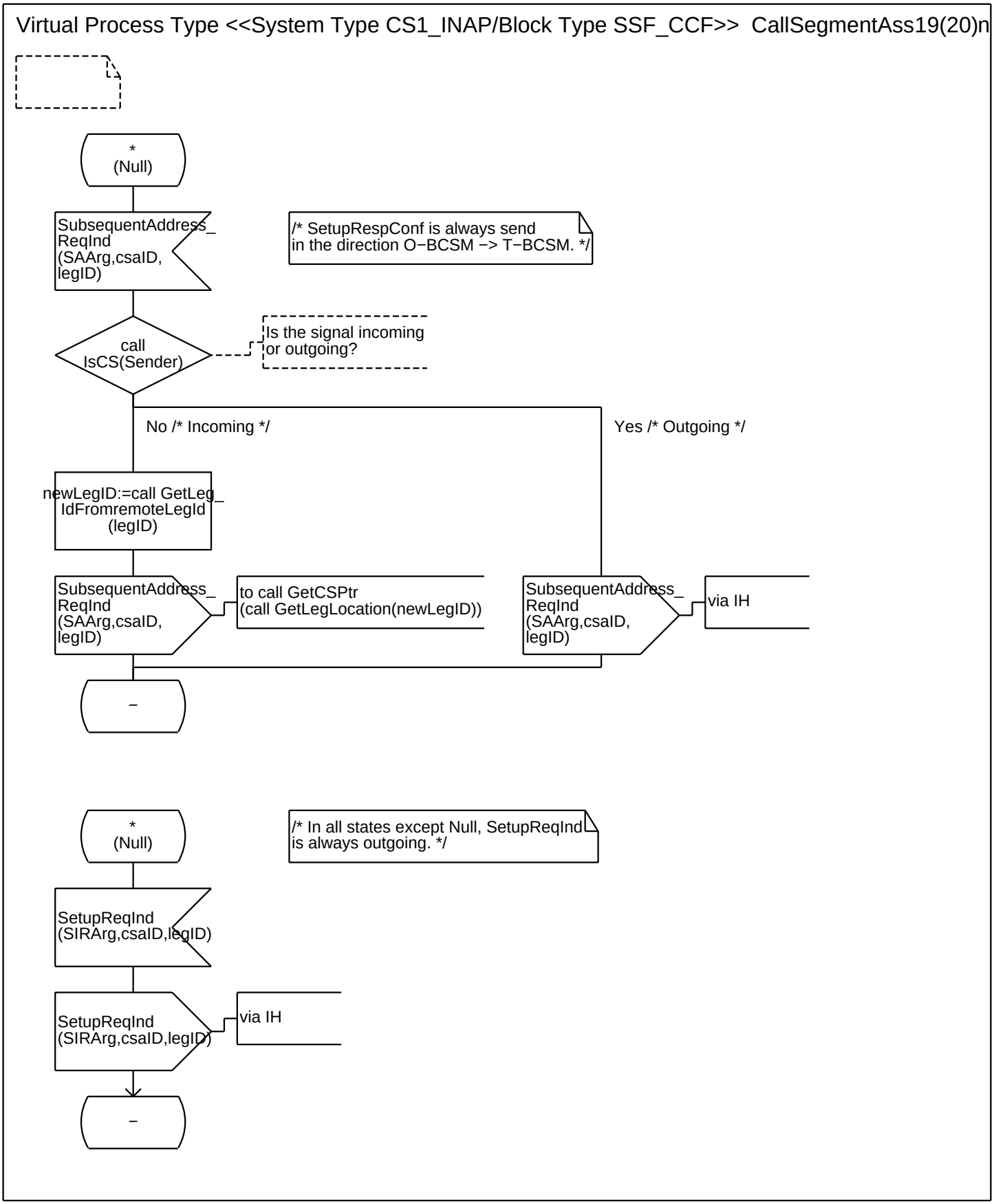
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss17(20)n
*
(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
–

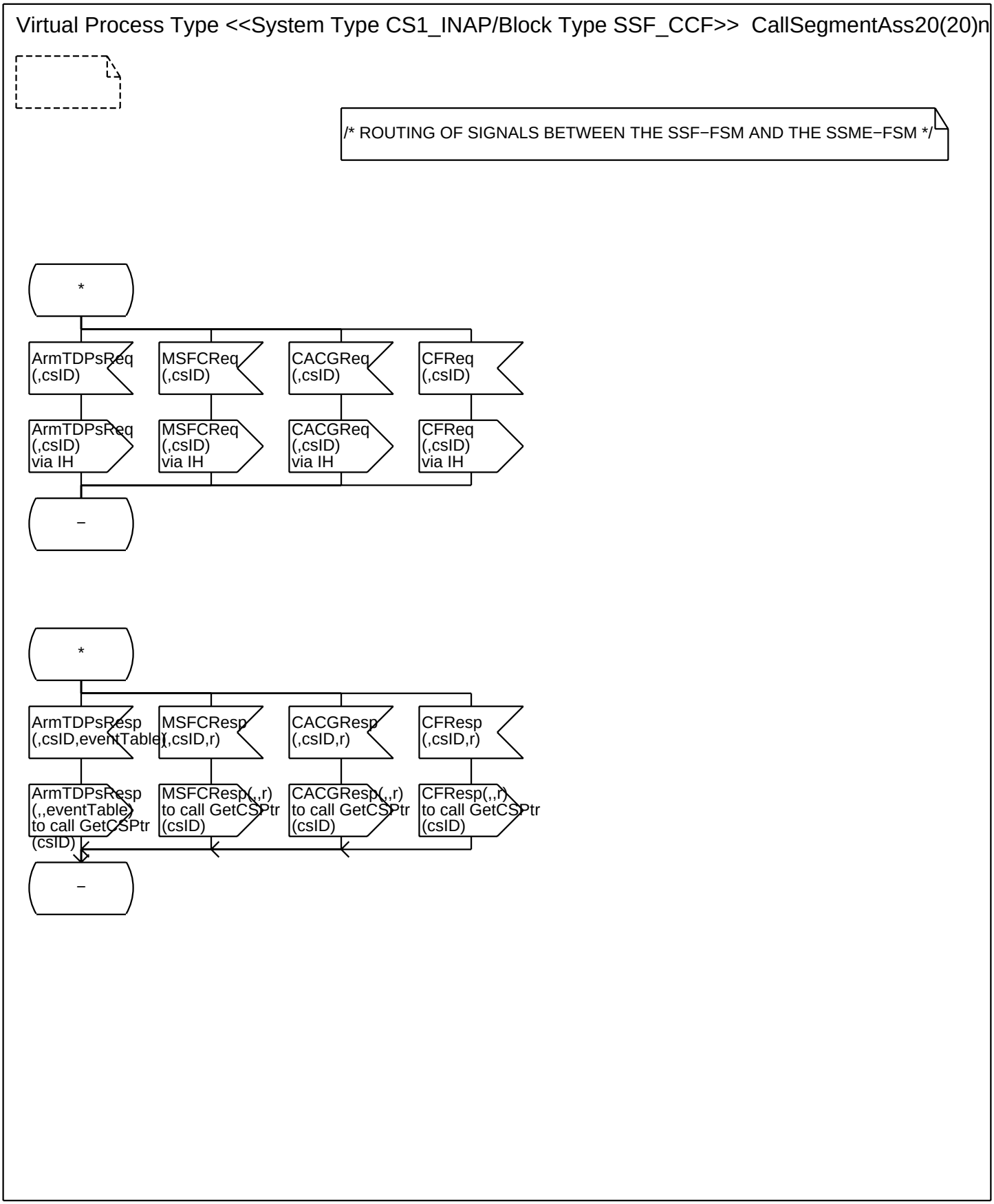
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss18(20)n
*
(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
−

Annex A:

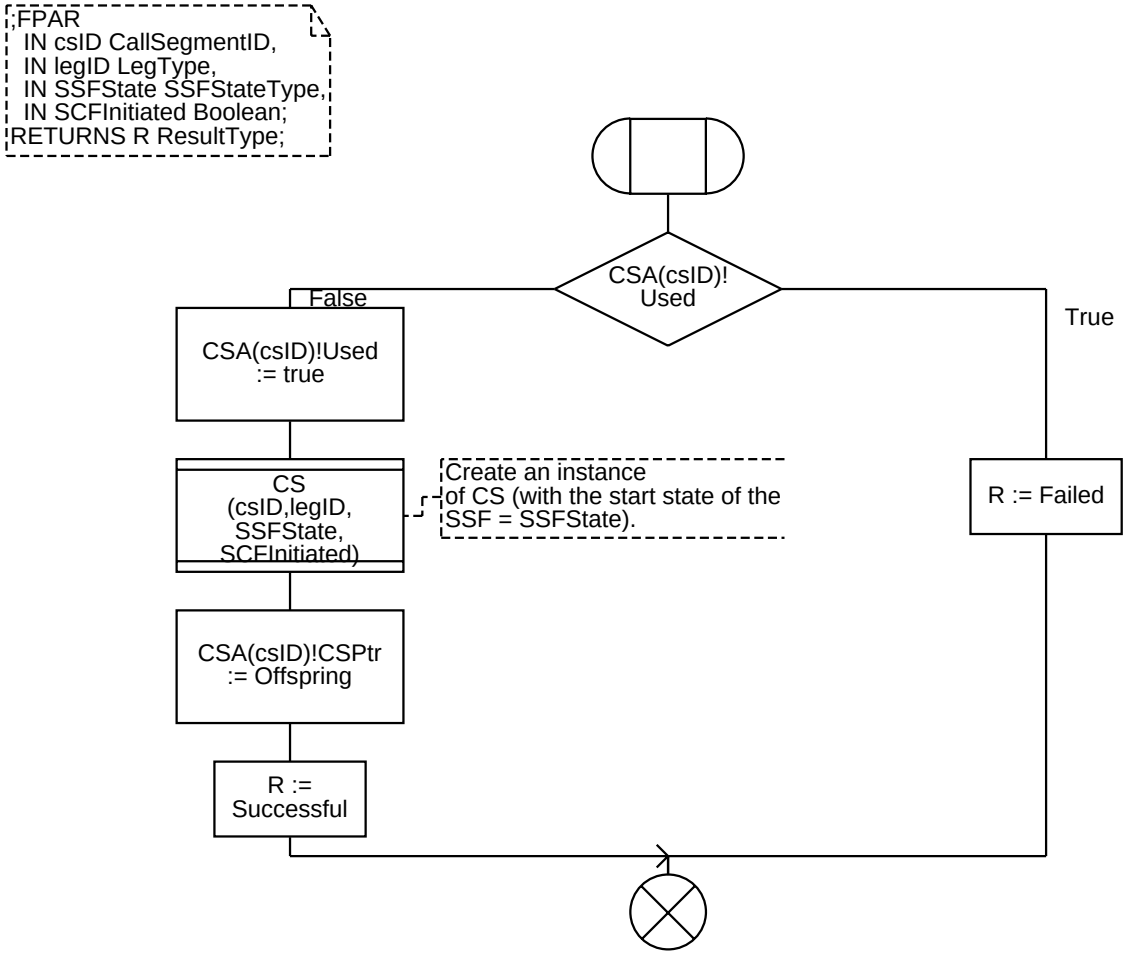

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss19(20)n
*
(Null)
SubsequentAddress_
ReqInd
(SAArg,csaID,
legID)
/* SetupRespConf is always send
in the direction O−BCSM −> T−BCSM. */
call
IsCS(Sender)
Is the signal incoming
or outgoing?
No /* Incoming */
Yes /* Outgoing */
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
−

Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegmentAss20(20)n
/* 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)
−

Annex A:

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

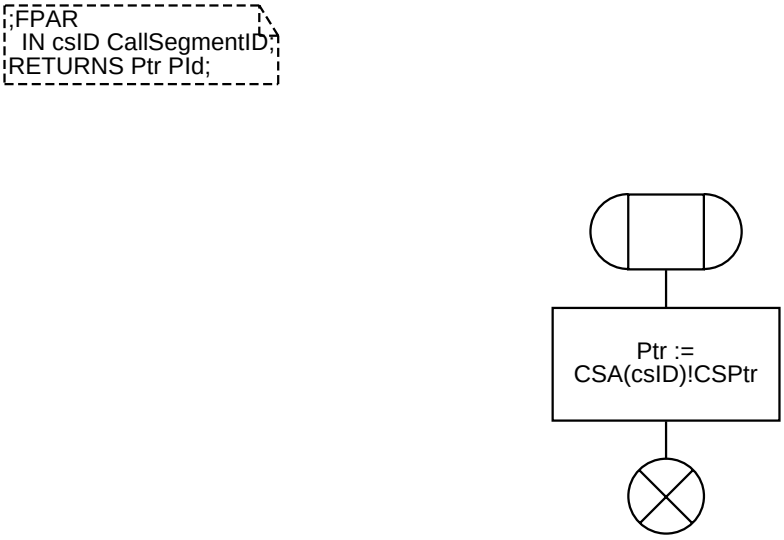


Annex A:

Procedure SetLegLocation 1(1)

Annex A:

Procedure GetCSPtr 1(1)

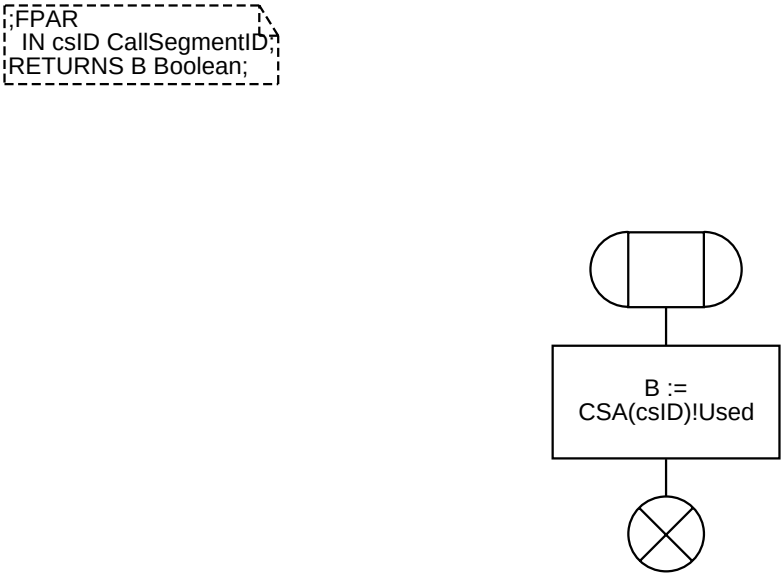


Annex A:

Procedure GetLegLocation 1(1)

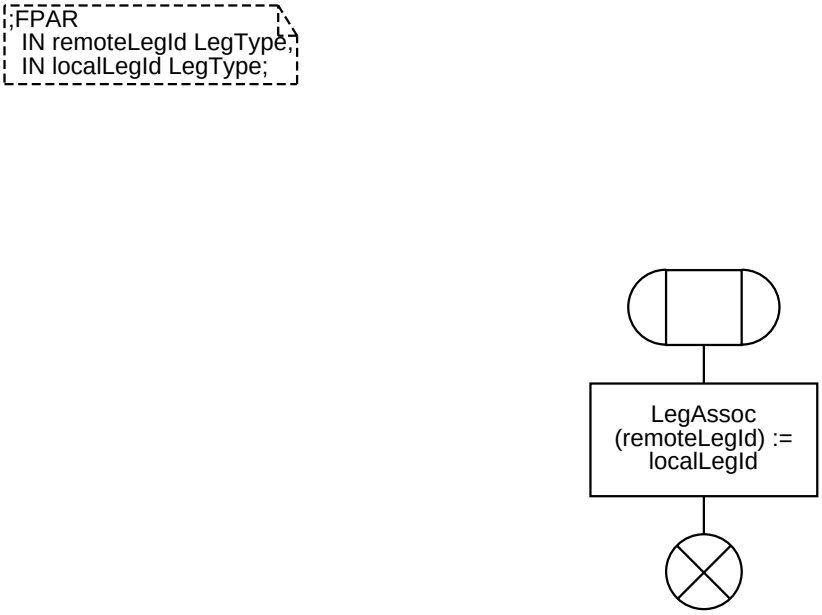
Annex A:

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

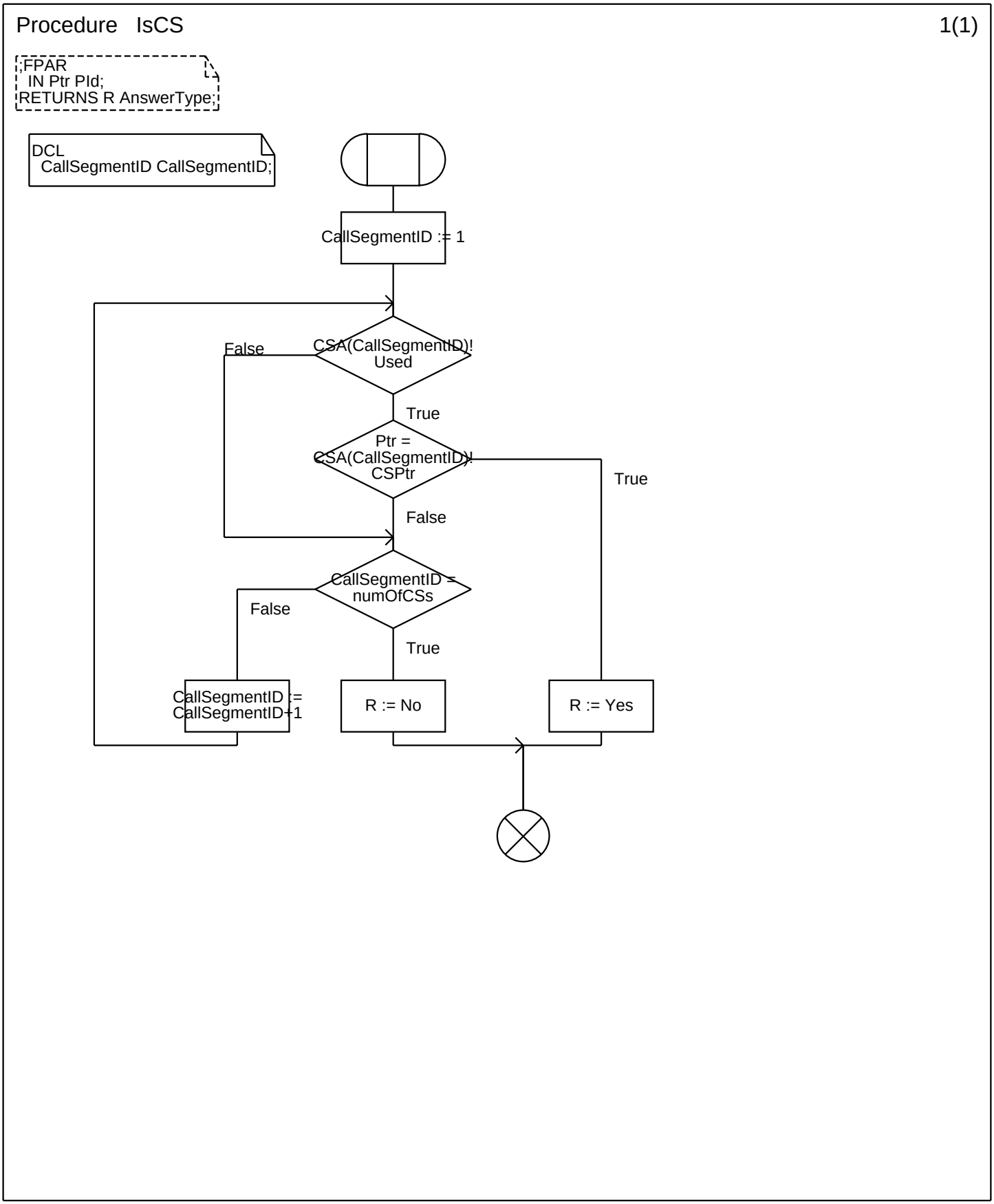


Annex A:

Procedure <<Process Type CallSegmentAssociation>> SetLegAssoc 1(1)

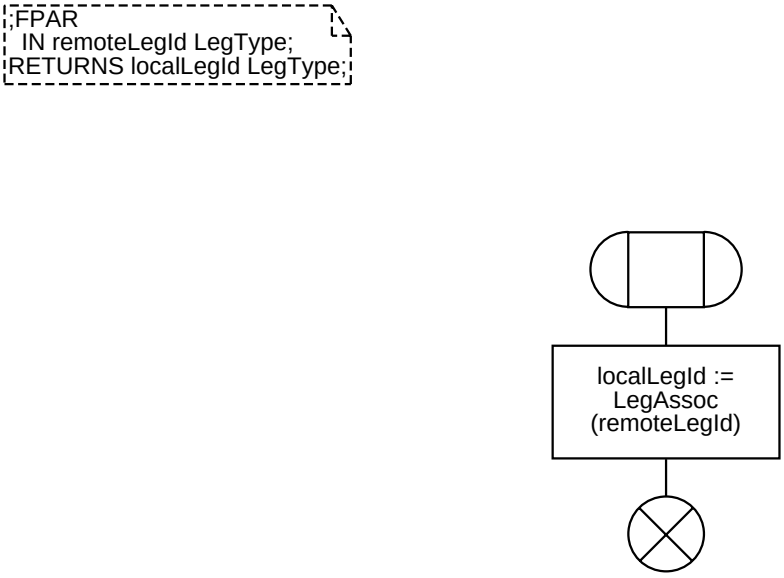


Annex A:


Procedure IsCS
1(1)
;FPAR
IN Ptr PId;
RETURNS R AnswerType;
DCL
CallSegmentID CallSegmentID;
CallSegmentID := 1
CSA(CallSegmentID)!
Used
False
True
Ptr =
CSA(CallSegmentID)!
CSPtr
True
False
CallSegmentID =
numOfCSs
False
True
CallSegmentID :=
CallSegmentID+1
R := No
R := Yes

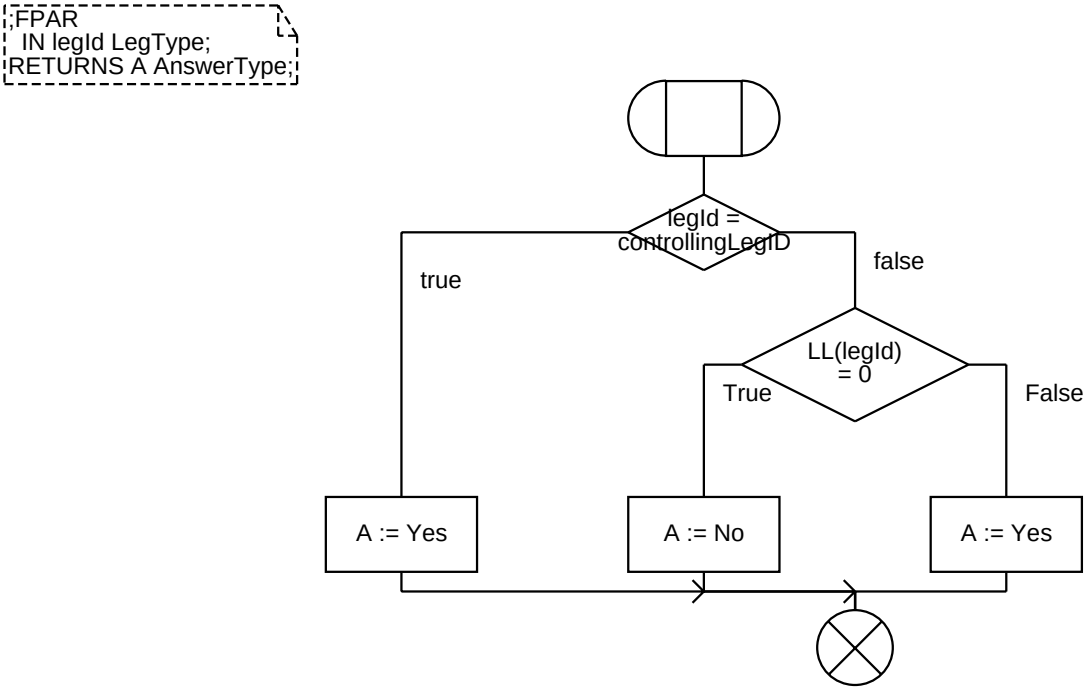
Annex A:

Procedure <<Process Type CallSegmentAssociation>> GetLegIdfromRemoteLegId 1(1)

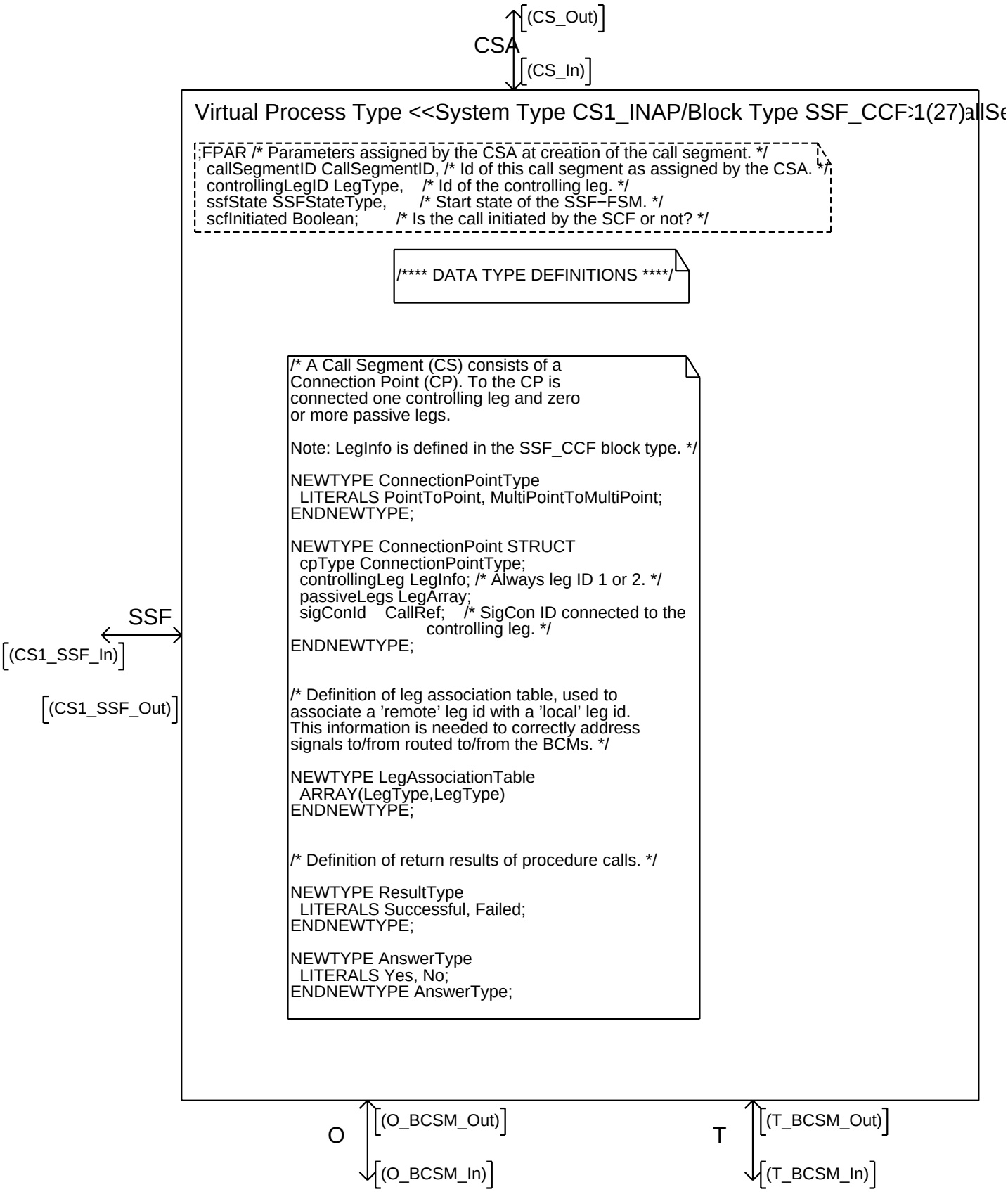


Annex A:

Procedure <<Process Type CallSegmentAssociation>> ExistLeg 1(1)



Annex A:


(CS_Out)
CSA
(CS_In)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF:1(27)allSe
;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)

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 2(27)

```
;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,
 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,
 ieArg ErrorArg;
```

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

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 3(27)

```
;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 */

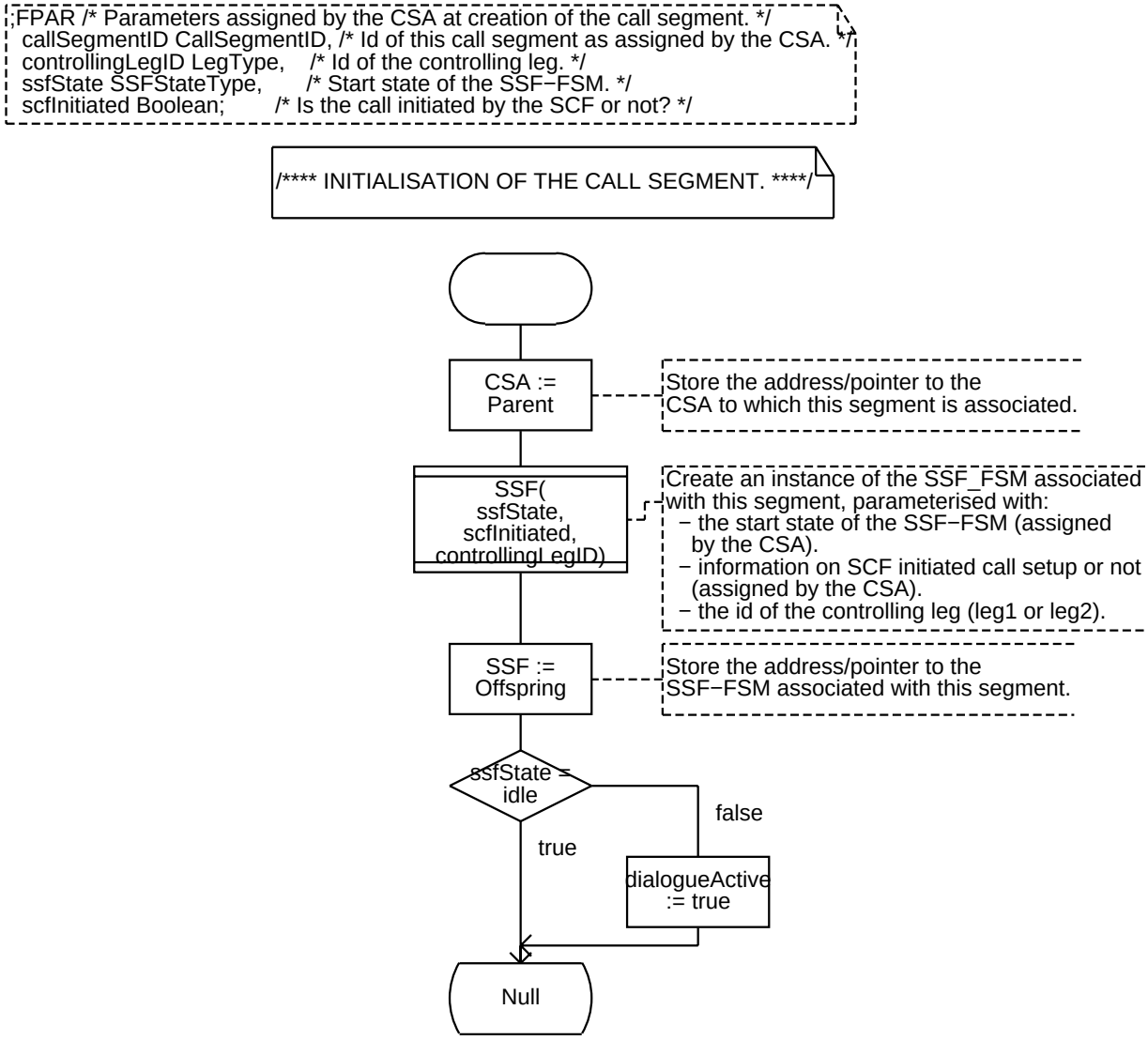
- AddLeg — Adds a leg to a call segment (connection point).
- Remove_Leg — Removes a leg.
- SetLeg_Status — Sets the status of a leg.
- Release_AllLegs — Removes all legs associated with this segment.
- GetLeg_Status — Returns the status of a leg.
- SetLeg_Assoc — Associates a remote leg id with a local 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

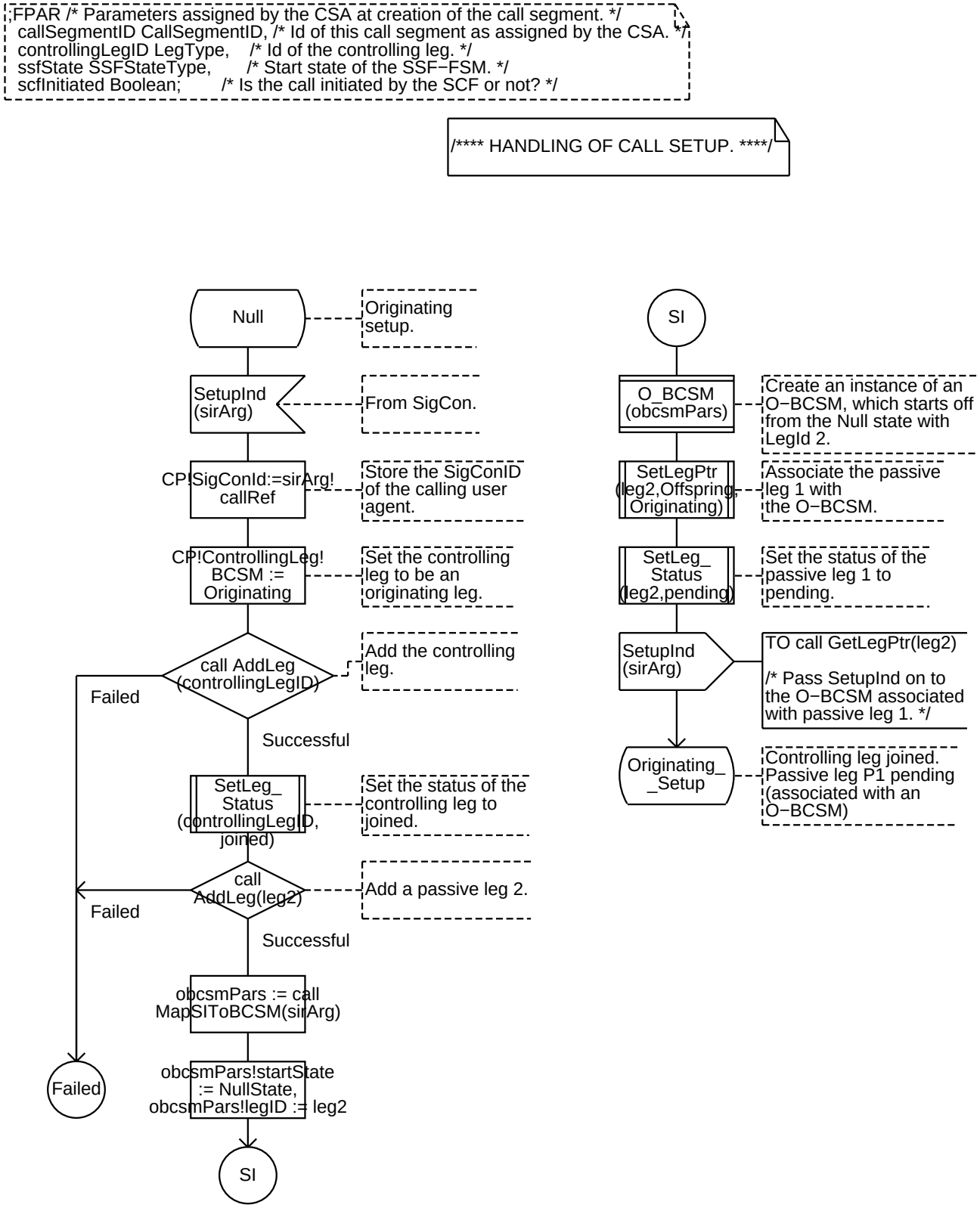
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 4(27)



Annex A:

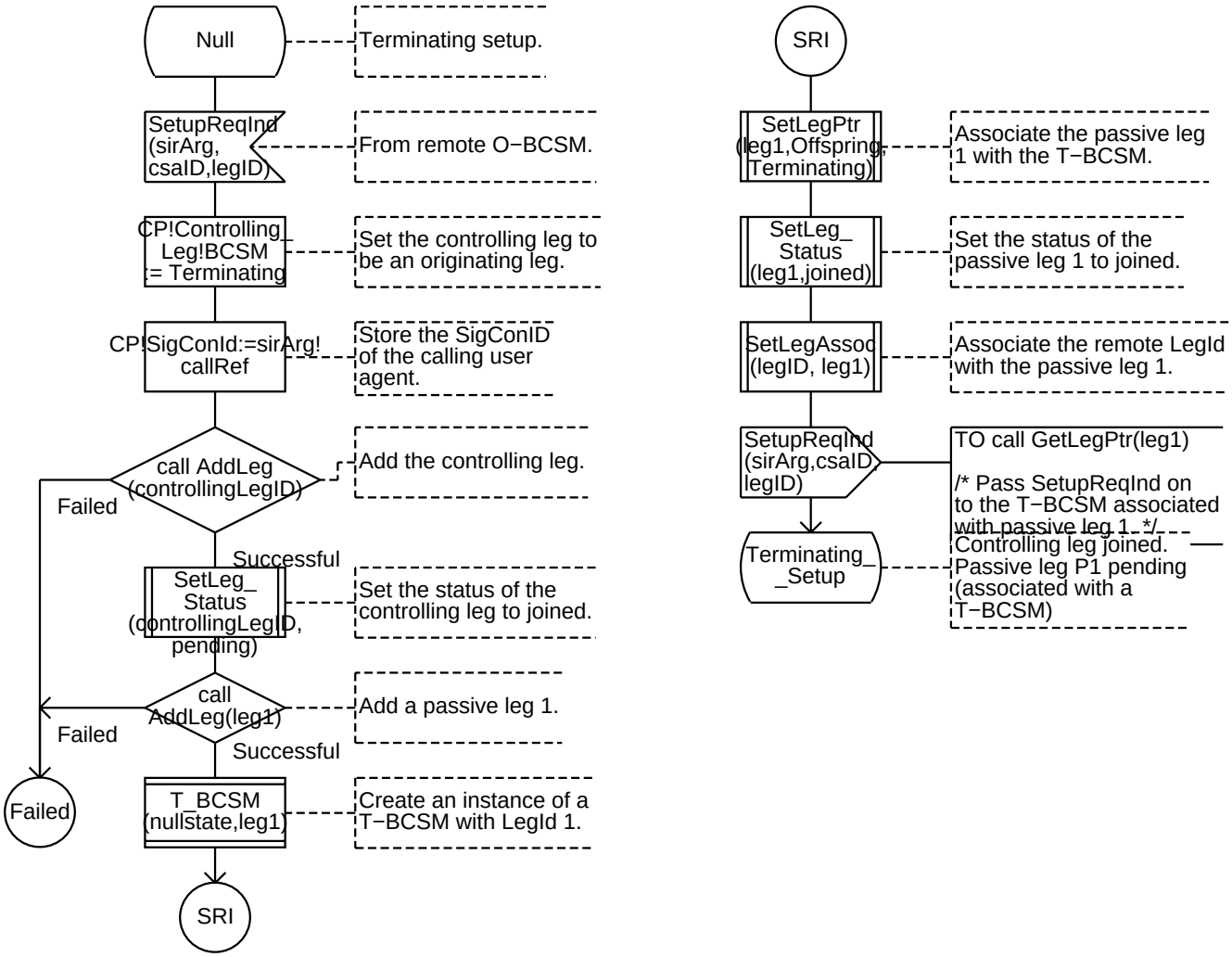
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 5(27)



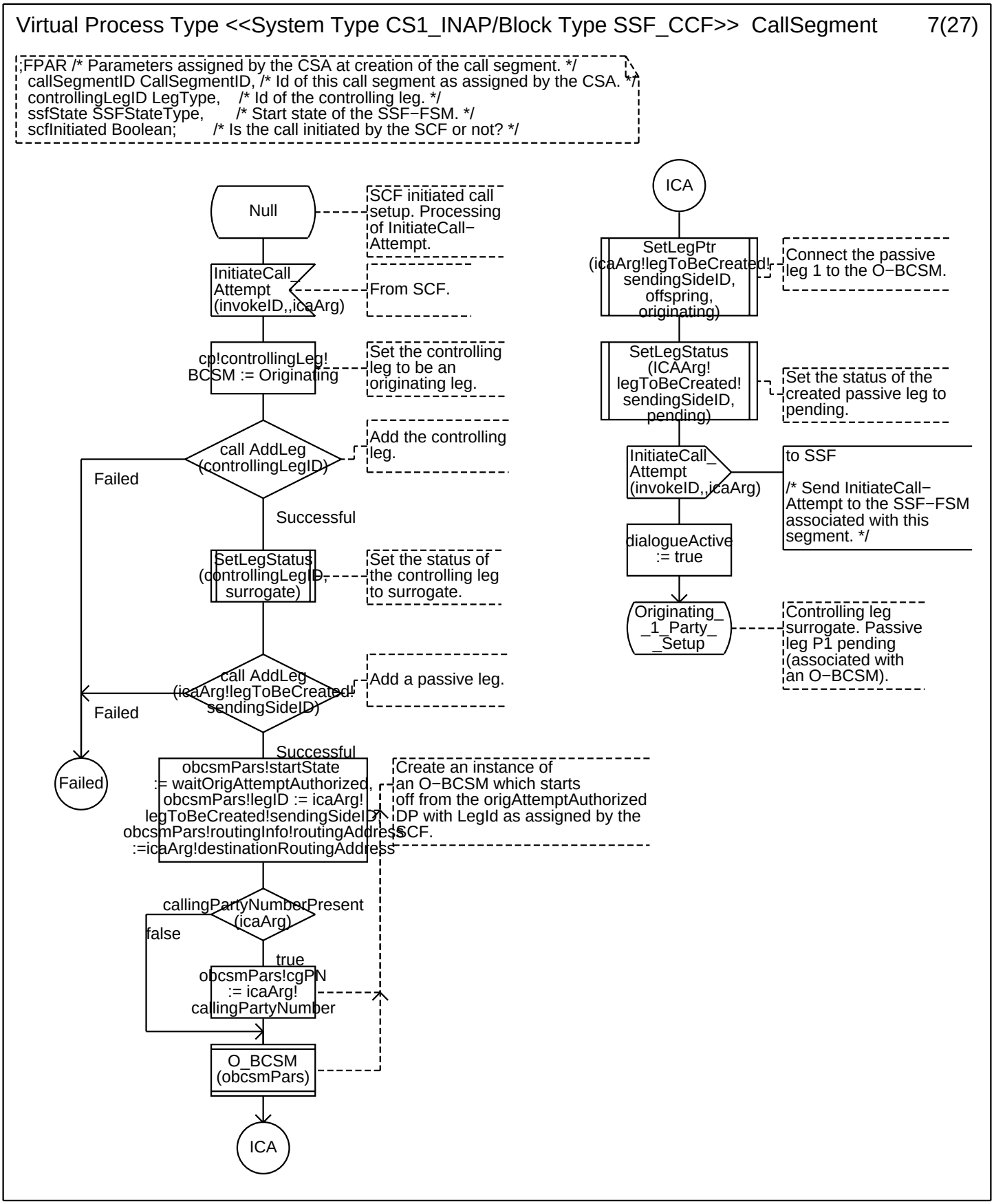
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 6(27)

```
;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? */
```



Annex A:

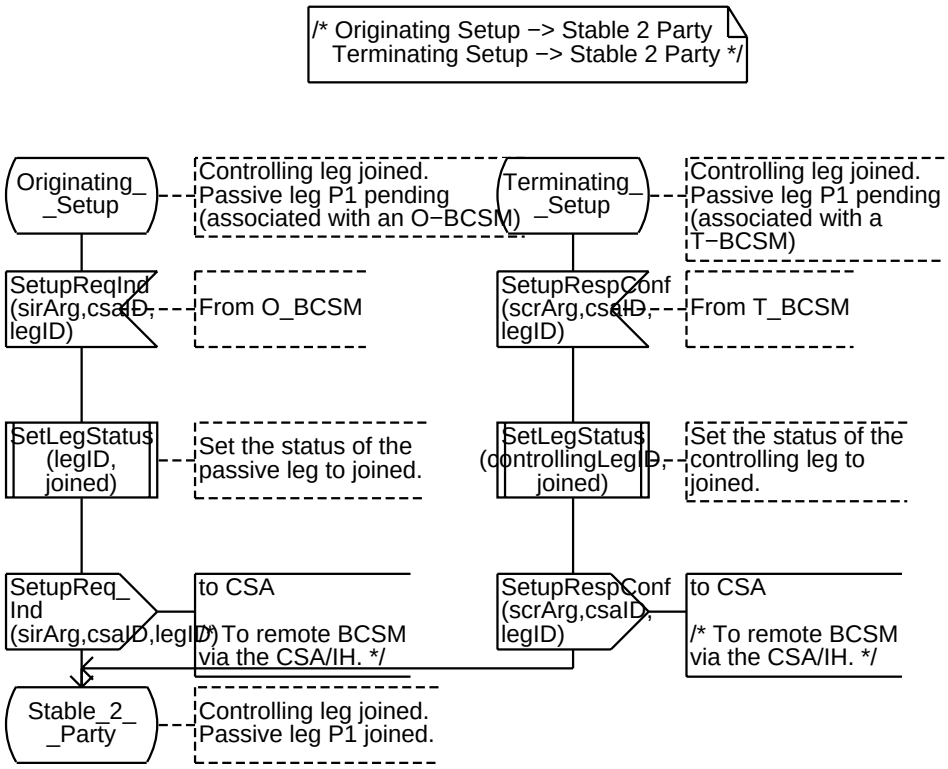

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 7(27)
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
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).

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 8(27)

```
;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? */
```

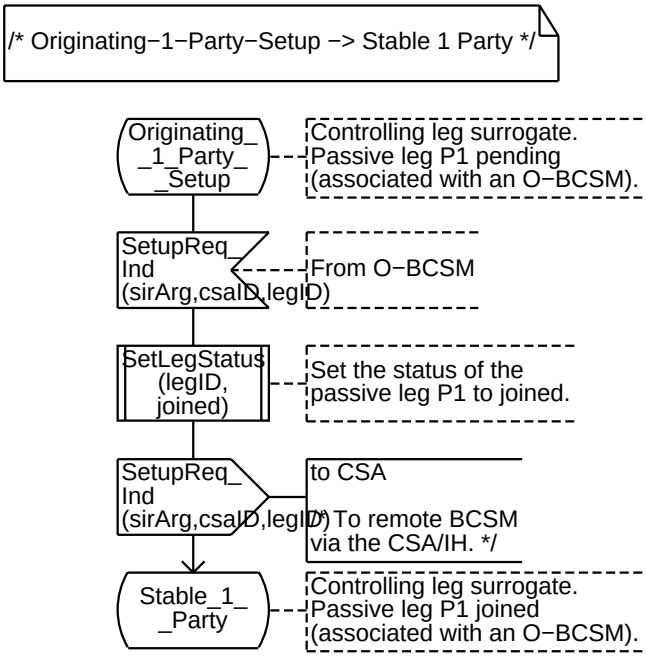
```
/**** TRANSITIONS BETWEEN THE CONNECTION VIEW STATES OF THE CALL SEGME

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



Annex A:

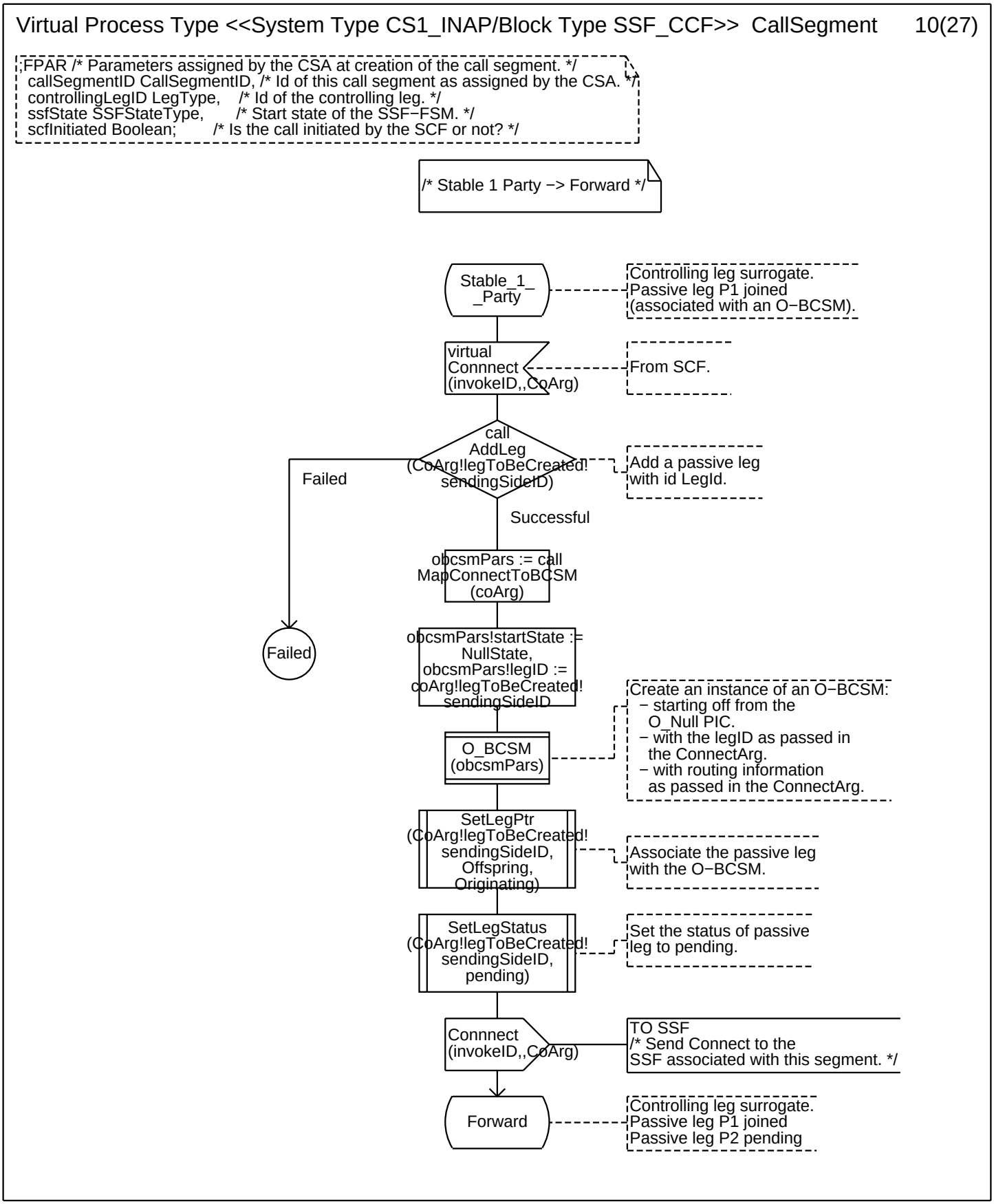
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 9(27)

```
;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? */
```



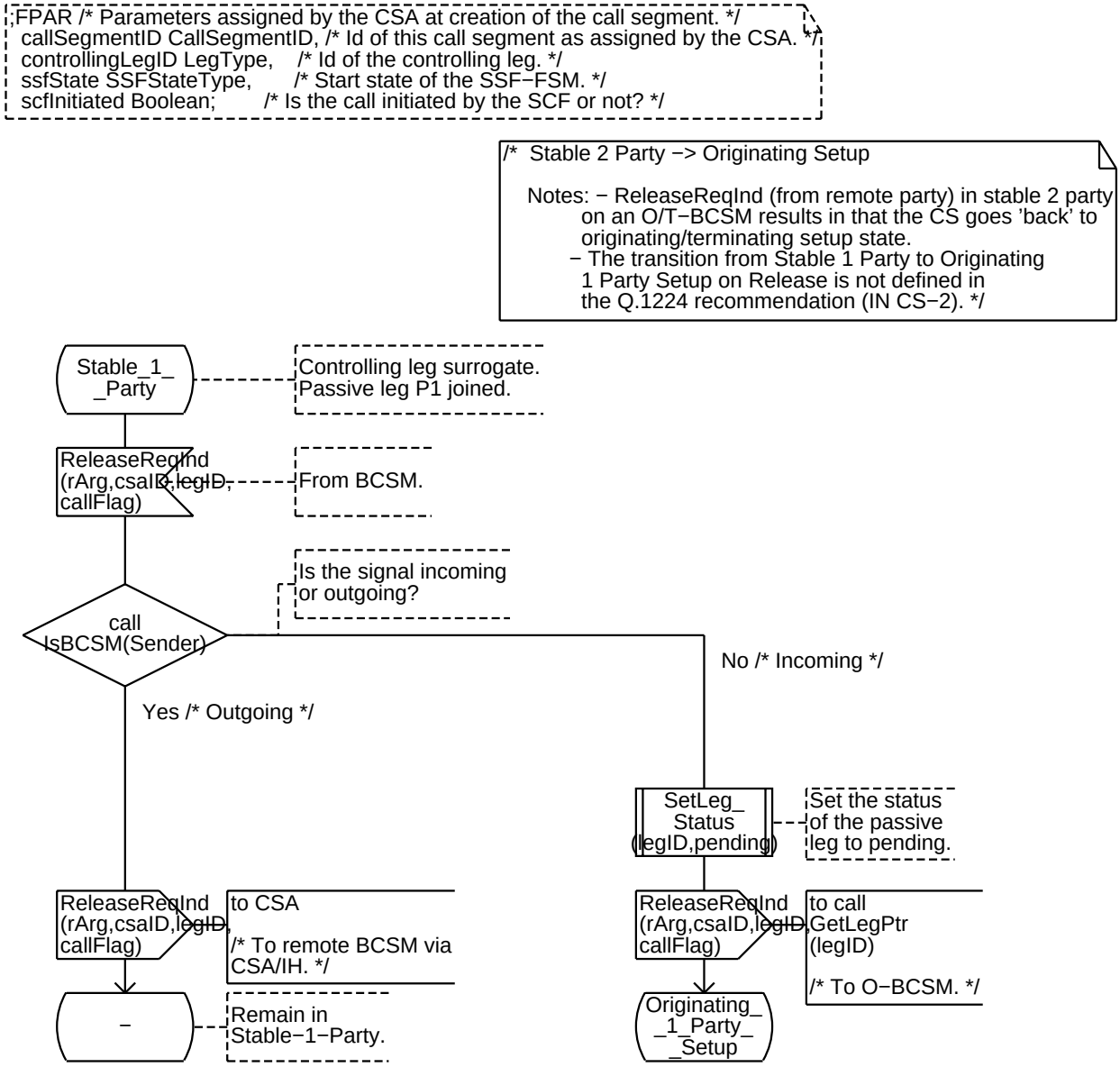
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 10(27)



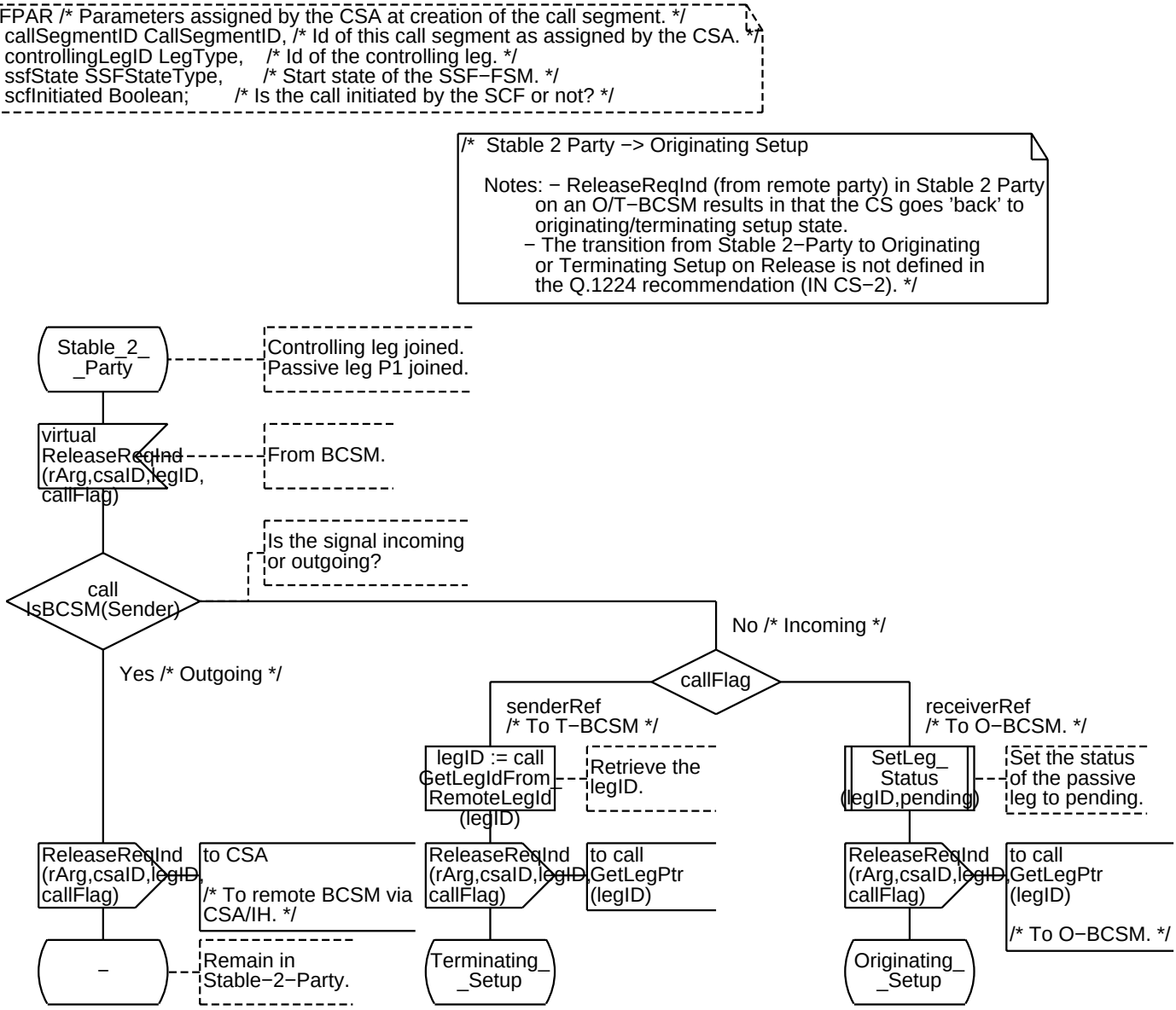
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 11(27)



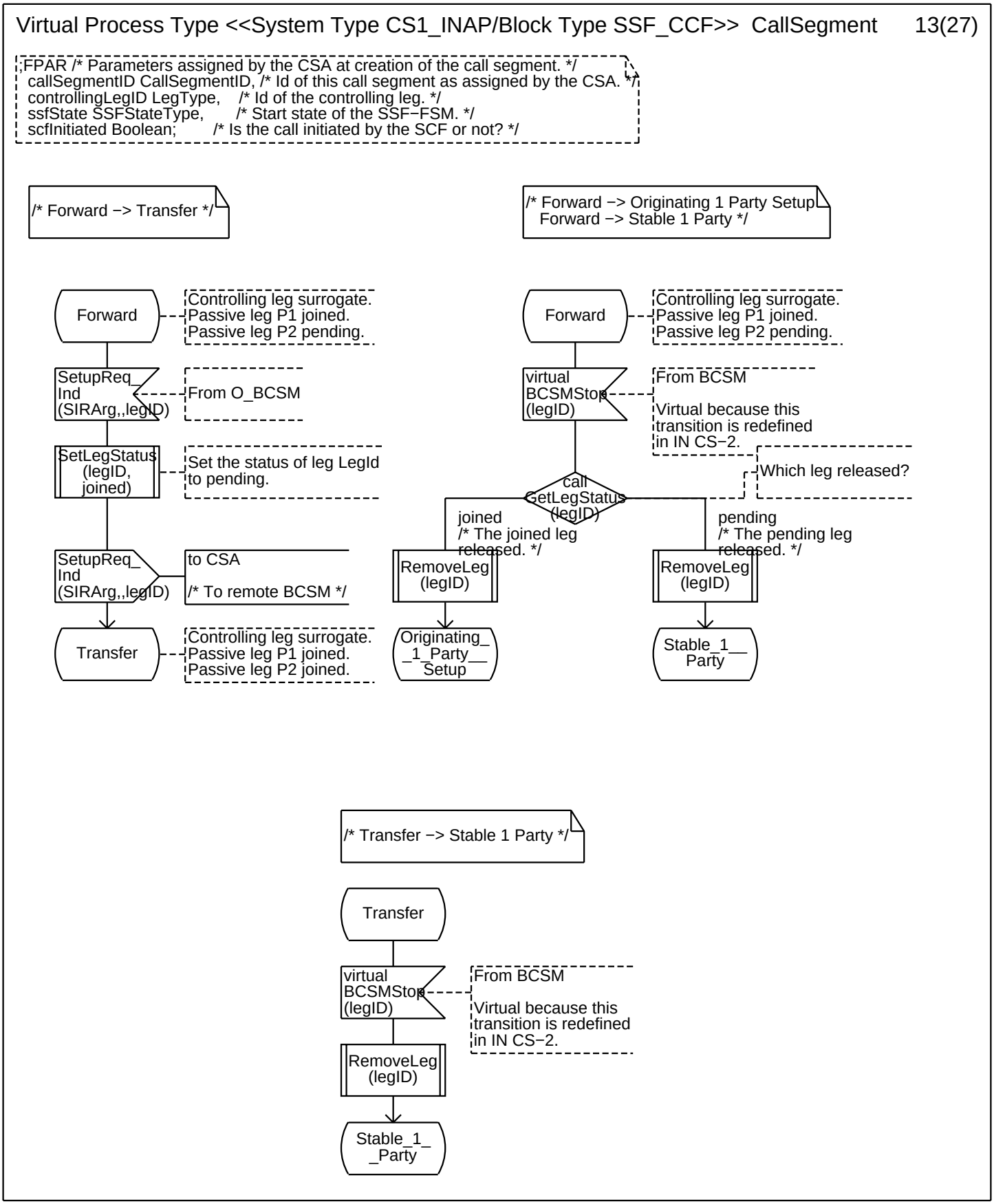
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 12(27)

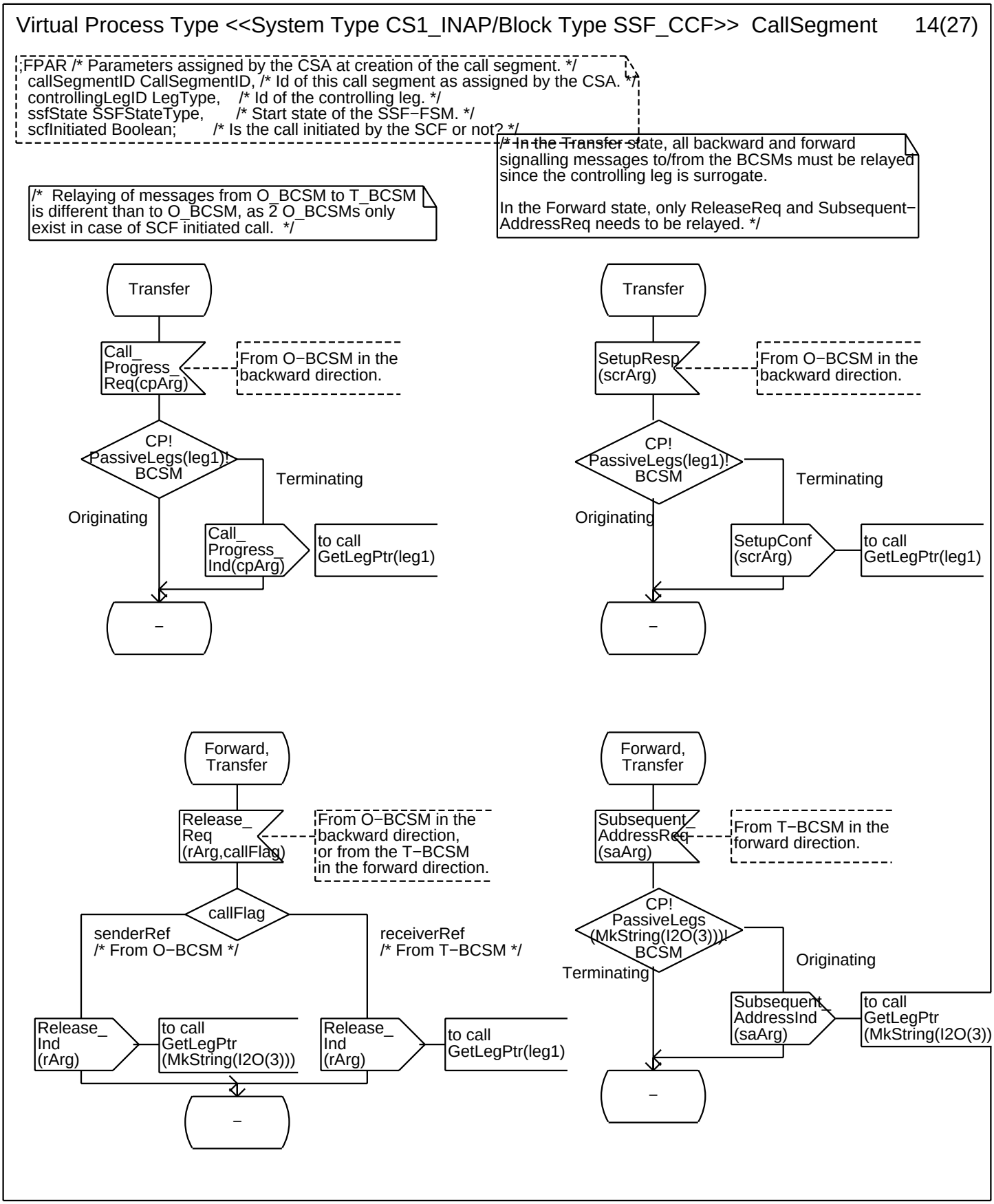


Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 13(27)

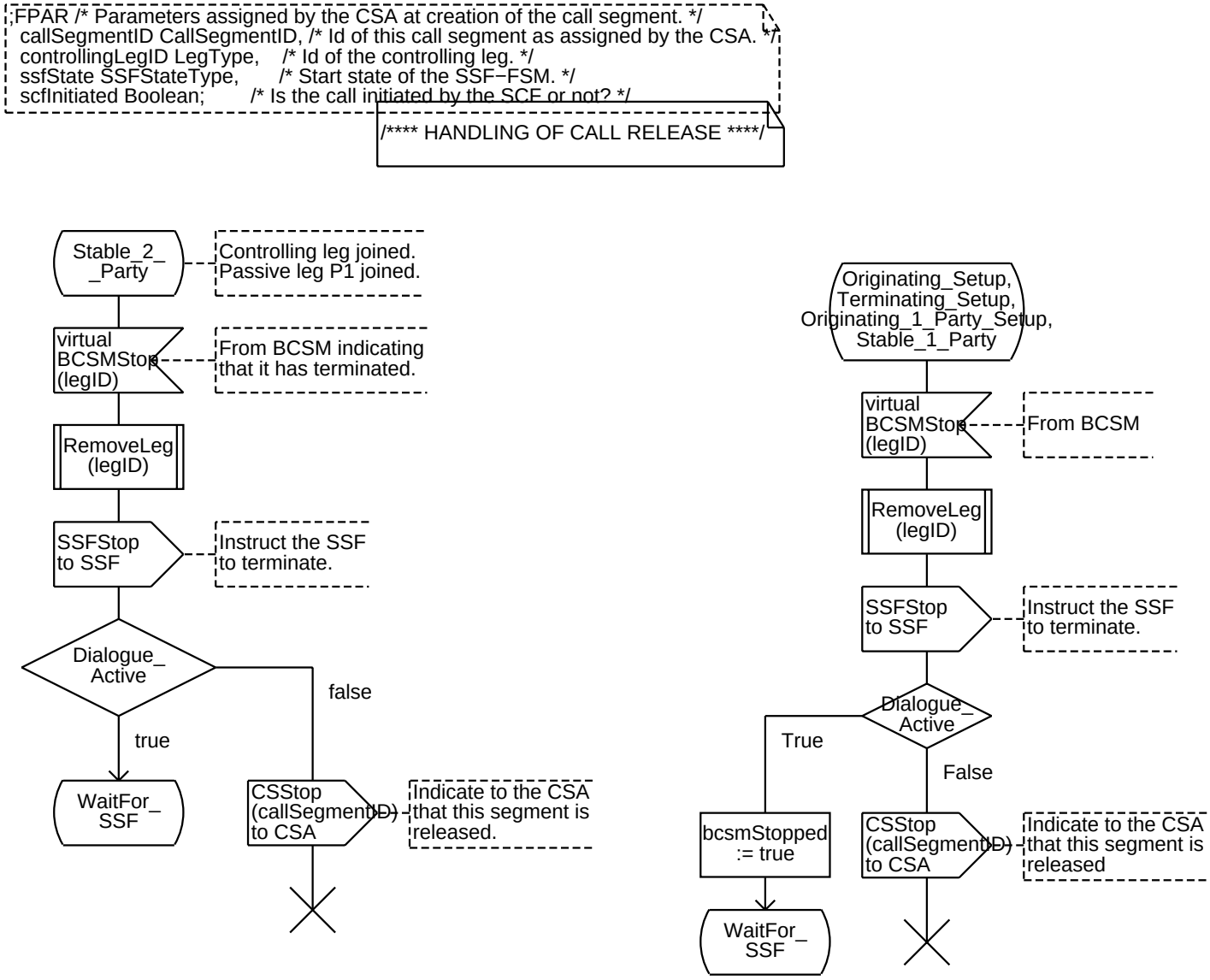


Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 14(27)
;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. */
/* Relaying of messages from O_BCSM to T_BCSM is different than to O_BCSM, as 2 O_BCSMs only exist in case of SCF initiated call. */
Transfer
Call_ Progress_ Req(cpArg)
From O−BCSM in the backward direction.
CP! PassiveLegs(leg1)! BCSM
Terminating
Originating
Call_ Progress_ Ind(cpArg)
to call GetLegPtr(leg1)
−
Transfer
SetupResp (scrArg)
From O−BCSM in the backward direction.
CP! PassiveLegs(leg1)! BCSM
Terminating
Originating
SetupConf (scrArg)
to call GetLegPtr(leg1)
−
Forward, Transfer
Release_ Req (rArg,callFlag)
From O−BCSM in the backward direction, or from the T−BCSM in the forward direction.
callFlag
senderRef /* From O−BCSM */
receiverRef /* From T−BCSM */
Release_ Ind (rArg)
to call GetLegPtr (MkString(I2O(3)))
Release_ Ind (rArg)
to call GetLegPtr(leg1)
−
Forward, Transfer
Subsequent_ AddressReq (saArg)
From T−BCSM in the forward direction.
CP! PassiveLegs (MkString(I2O(3)))! BCSM
Originating
Terminating
Subsequent_ AddressInd (saArg)
to call GetLegPtr (MkString(I2O(3))
−

Annex A:

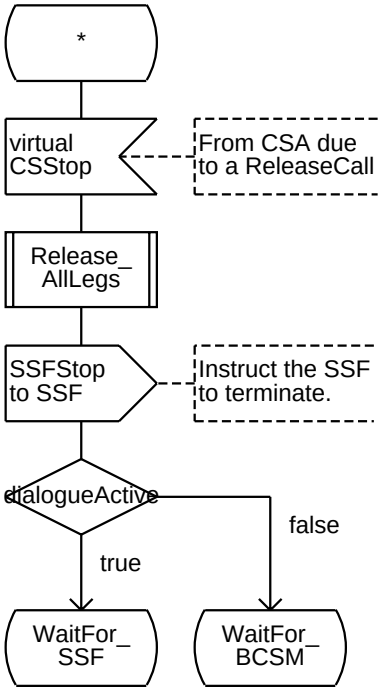
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 15(27)



Annex A:

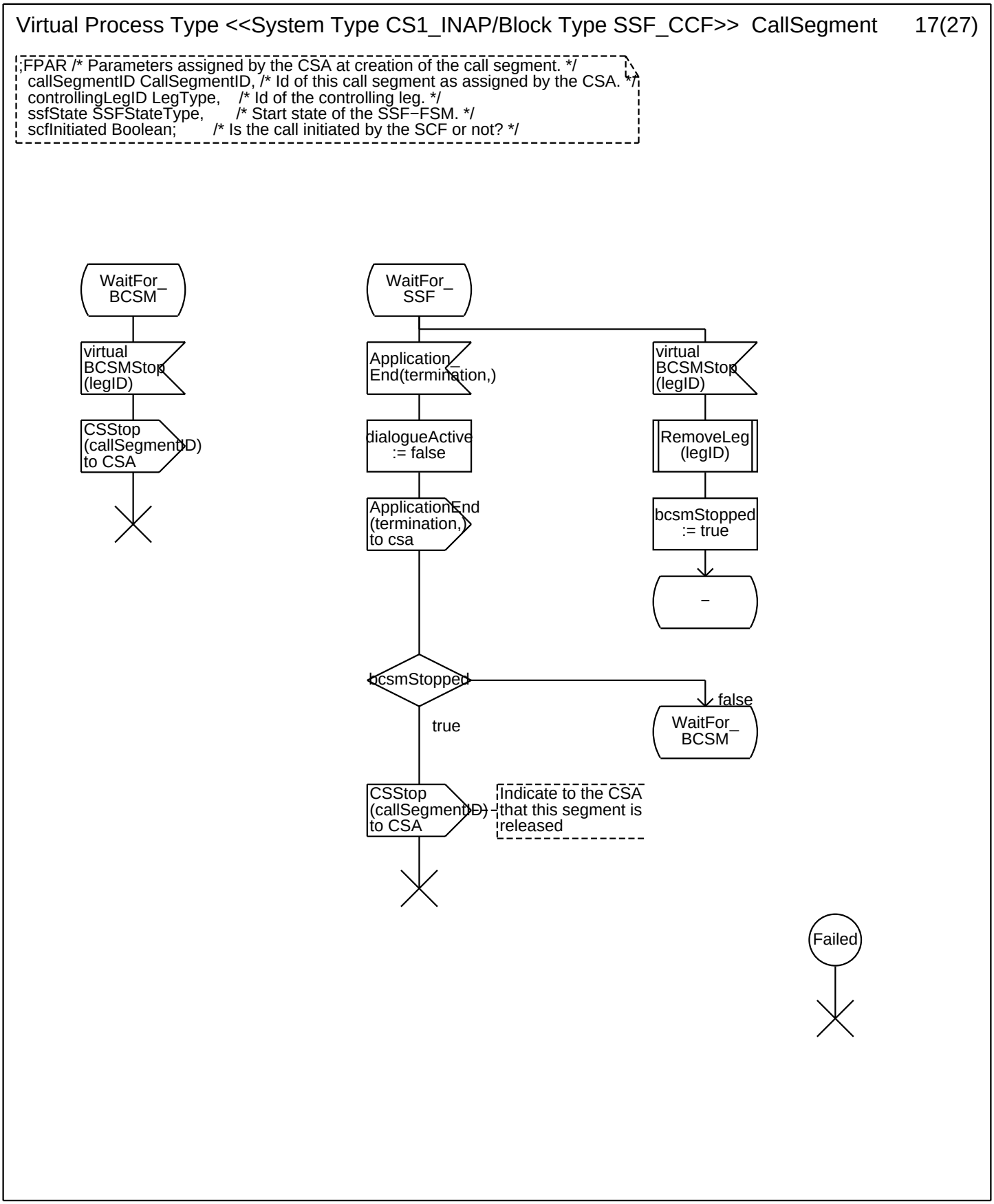
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 16(27)

```
;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? */
```

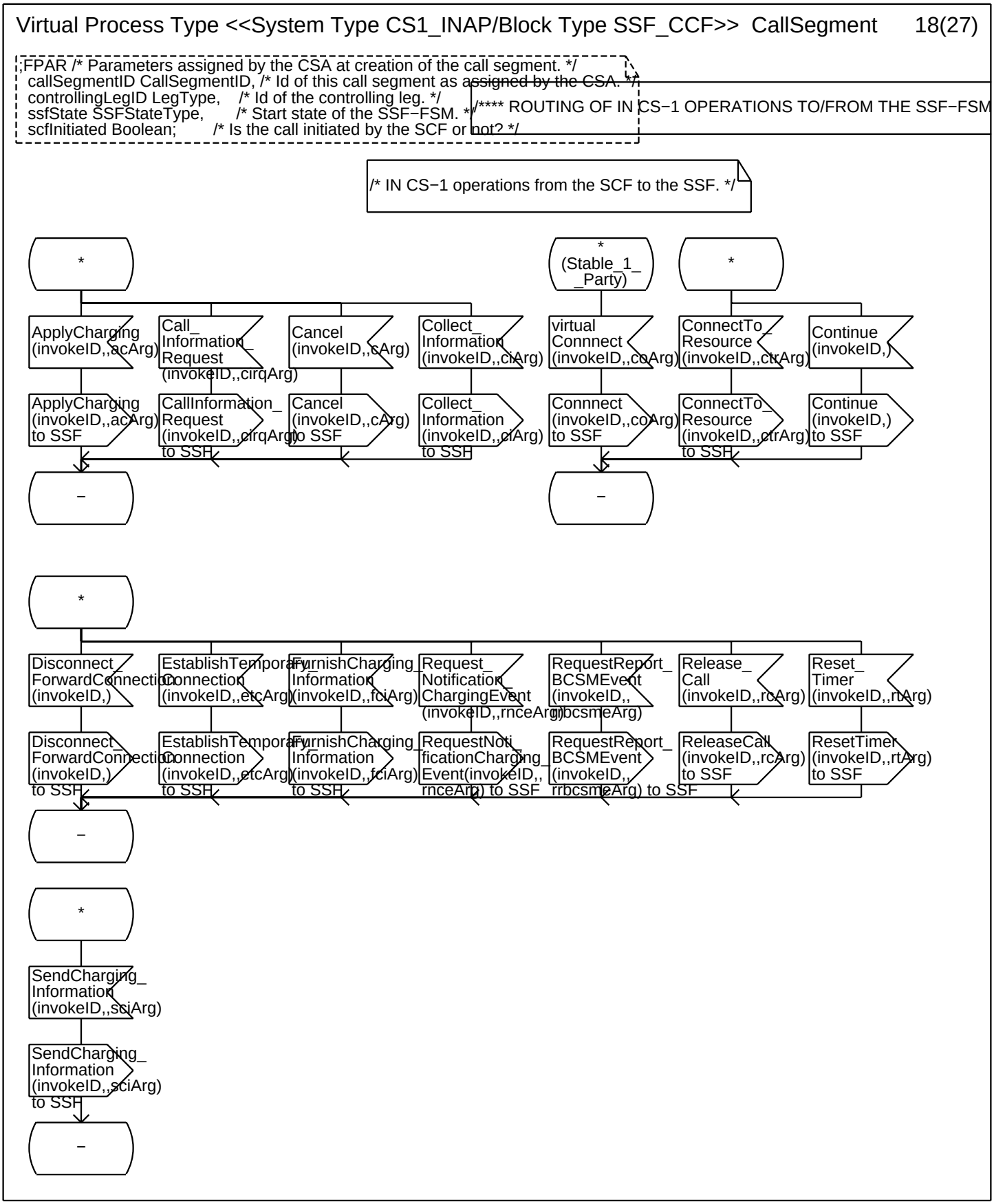


Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 17(27)

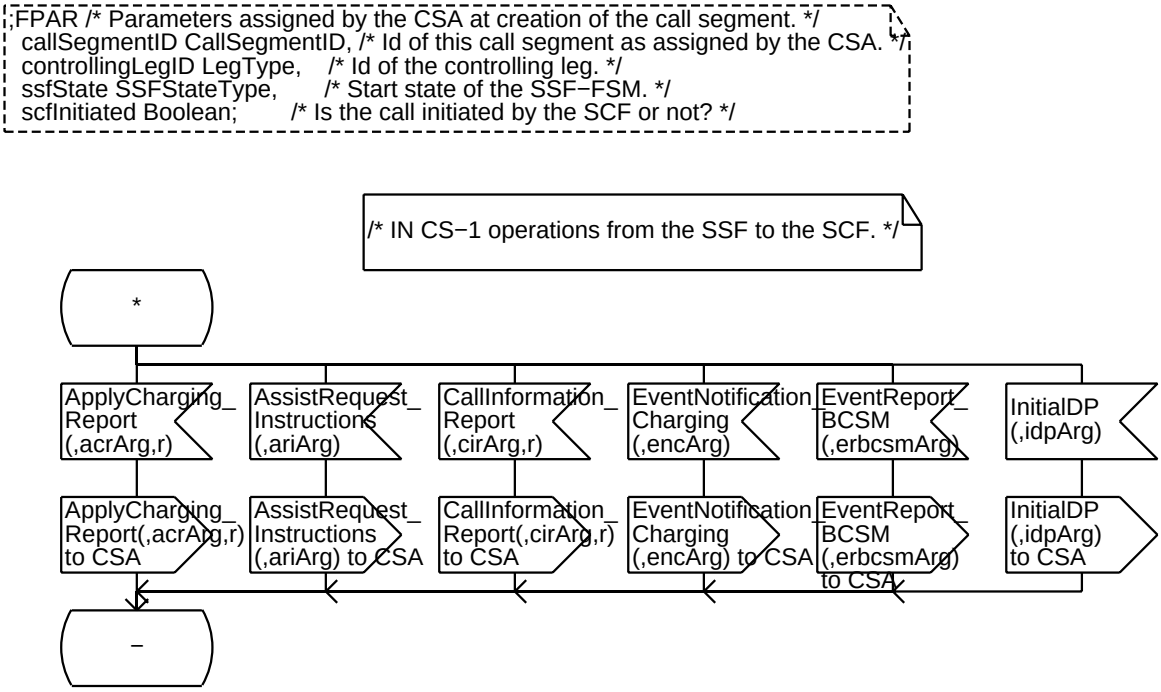


Annex A:

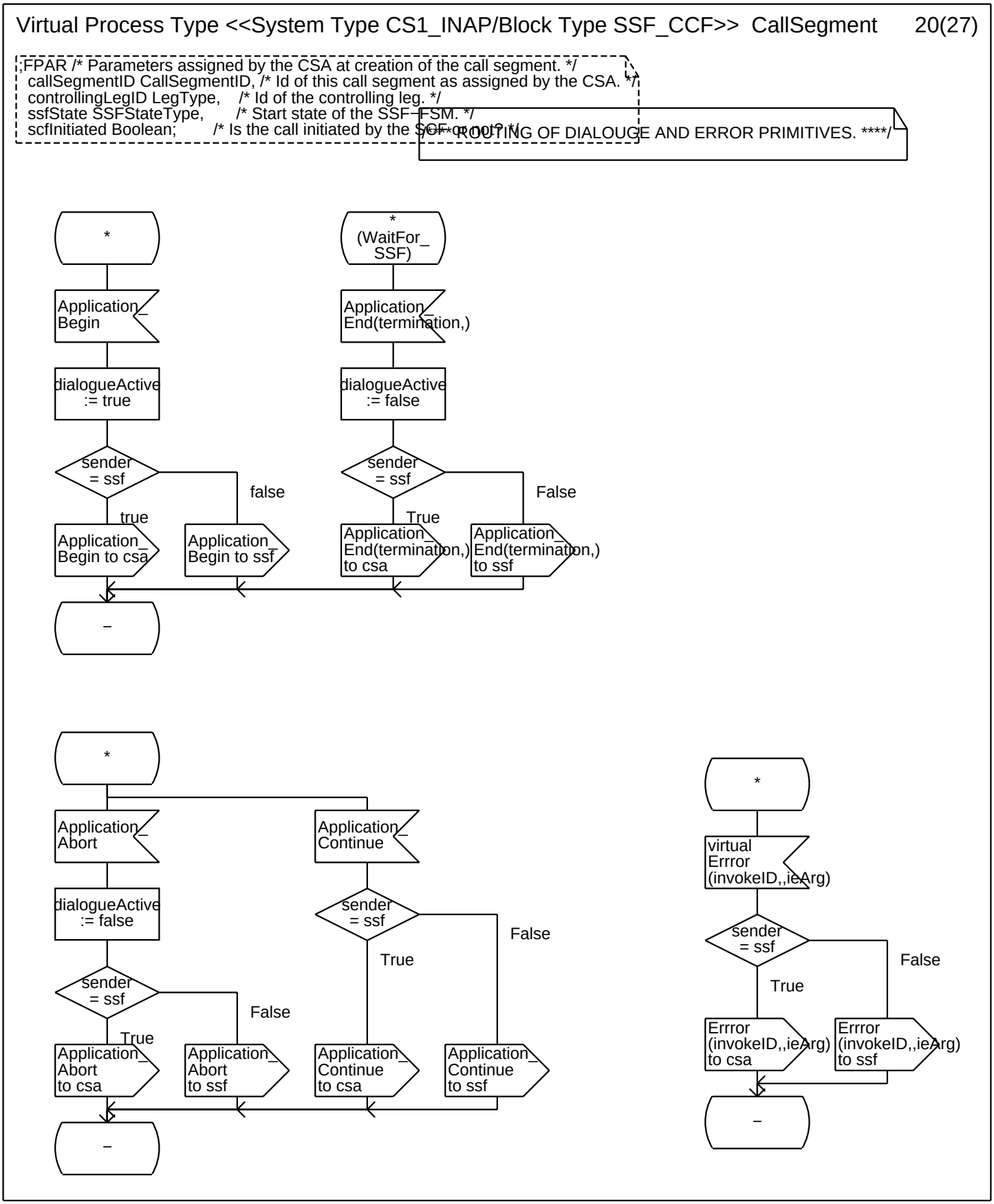

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 18(27)
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
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. */
*
*
(Stable_1_ _Party)
*
ApplyCharging (invokeID,,acArg)
Call_ Information_ Request (invokeID,,cirqArg)
Cancel (invokeID,,cArg)
Collect_ Information (invokeID,,ciArg)
virtual Connnect (invokeID,,coArg)
ConnectTo_ Resource (invokeID,,ctrArg)
Continue (invokeID,)
ApplyCharging (invokeID,,acArg) to SSF
CallInformation_ Request (invokeID,,cirqArg) to SSH
Cancel (invokeID,,cArg) to SSF
Collect_ Information (invokeID,,ciArg) to SSH
Connnect (invokeID,,coArg) to SSF
ConnectTo_ Resource (invokeID,,ctrArg) to SSH
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 SSH
EstablishTemporary_ Connection (invokeID,,etcArg) to SSH
FurnishCharging_ Information (invokeID,,fciArg) to SSH
RequestNoti_ ficationChargingEvent(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 SSH
−

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 19(27)

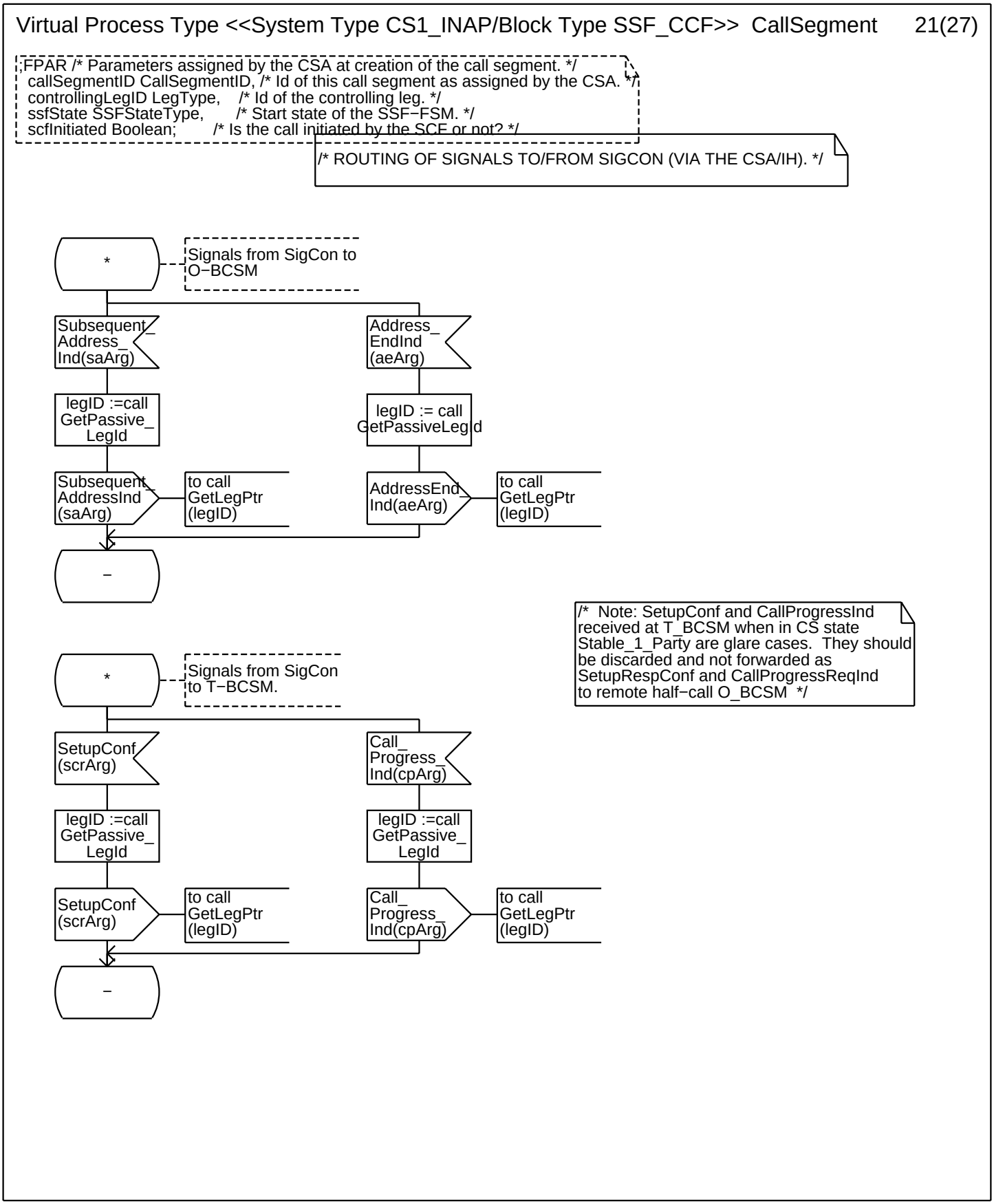


Annex A:

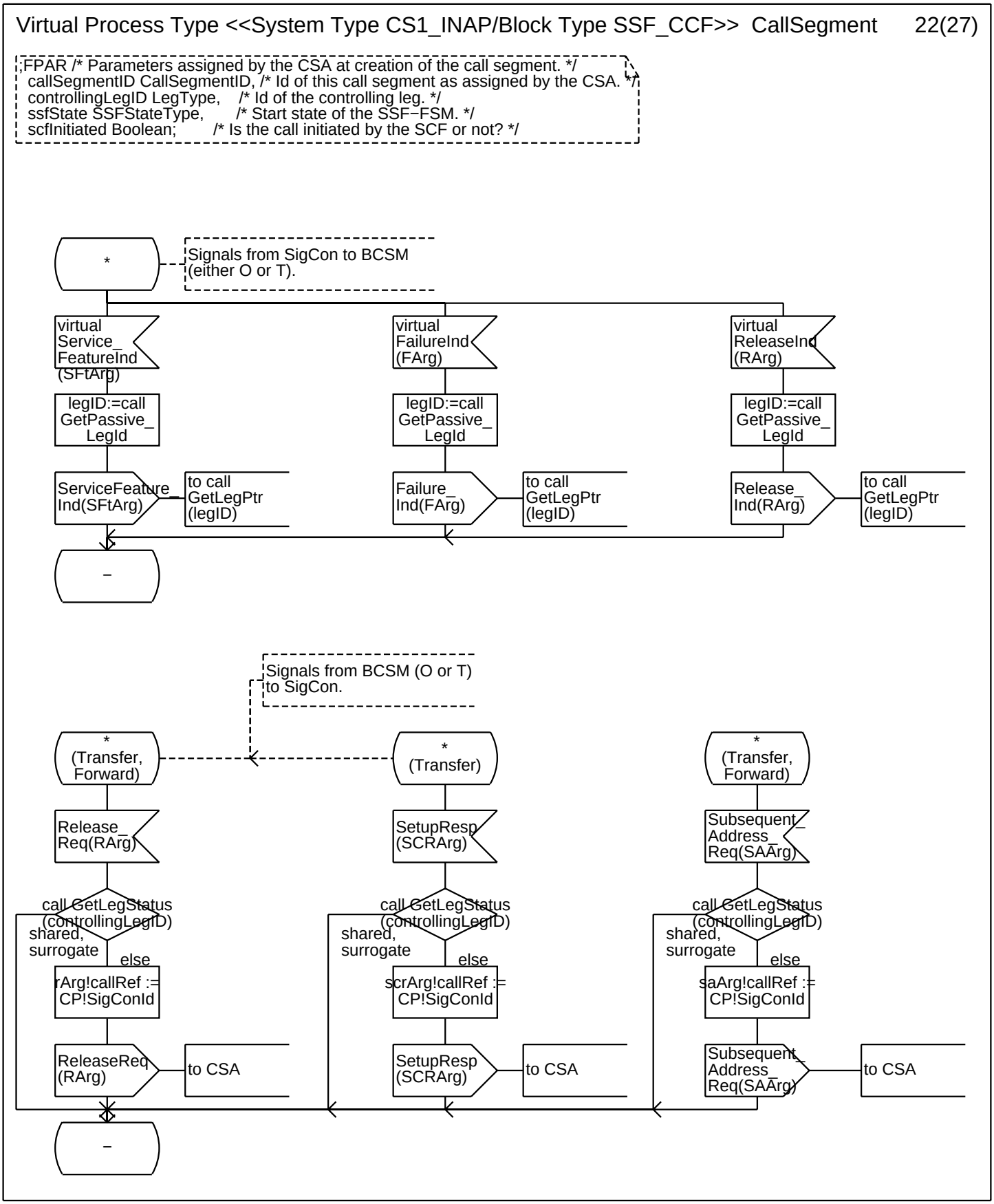

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 20(27)
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
controllingLegID LegType, /* Id of the controlling leg. */
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. ****/
*
Application_ Begin
dialogueActive := true
sender = ssf
true
false
Application_ Begin to csa
Application_ Begin to ssf
–
*
(WaitFor_ SSF)
Application_ End(termination,)
dialogueActive := false
sender = ssf
True
False
Application_ End(termination,) to csa
Application_ End(termination,) to ssf
*
Application_ Abort
dialogueActive := false
sender = ssf
True
False
Application_ Abort to csa
Application_ Abort to ssf
–
Application_ Continue
sender = ssf
True
False
Application_ Continue to csa
Application_ Continue to ssf
*
virtual Errror (invokeID,,ieArg)
sender = ssf
True
False
Errror (invokeID,,ieArg) to csa
Errror (invokeID,,ieArg) to ssf
–

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 21(27)



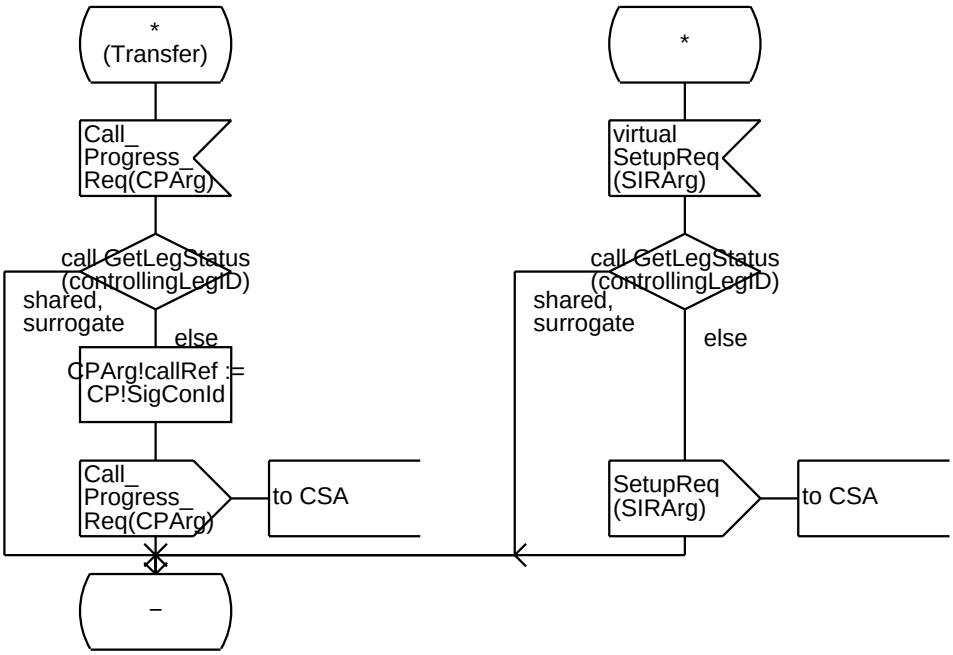
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 22(27)
;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? */
*
Signals from SigCon to BCSM (either O or T).
virtual Service_ FeatureInd (SFtArg)
virtual FailureInd (FArg)
virtual ReleaseInd (RArg)
legID:=call GetPassive_ LegId
legID:=call GetPassive_ LegId
legID:=call GetPassive_ LegId
ServiceFeature_ Ind(SFtArg)
to call GetLegPtr (legID)
Failure_ Ind(FArg)
to call GetLegPtr (legID)
Release_ Ind(RArg)
to call GetLegPtr (legID)
−
Signals from BCSM (O or T) to SigCon.
* (Transfer, Forward)
* (Transfer)
* (Transfer, Forward)
Release_ Req(RArg)
SetupResp (SCRArg)
Subsequent_ Address_ Req(SAArg)
call GetLegStatus (controllingLegID)
shared, surrogate
else
rArg!callRef := CP!SigConId
call GetLegStatus (controllingLegID)
shared, surrogate
else
scrArg!callRef := CP!SigConId
call GetLegStatus (controllingLegID)
shared, surrogate
else
saArg!callRef := CP!SigConId
ReleaseReq (RArg)
to CSA
SetupResp (SCRArg)
to CSA
Subsequent_ Address_ Req(SAArg)
to CSA
−

Annex A:

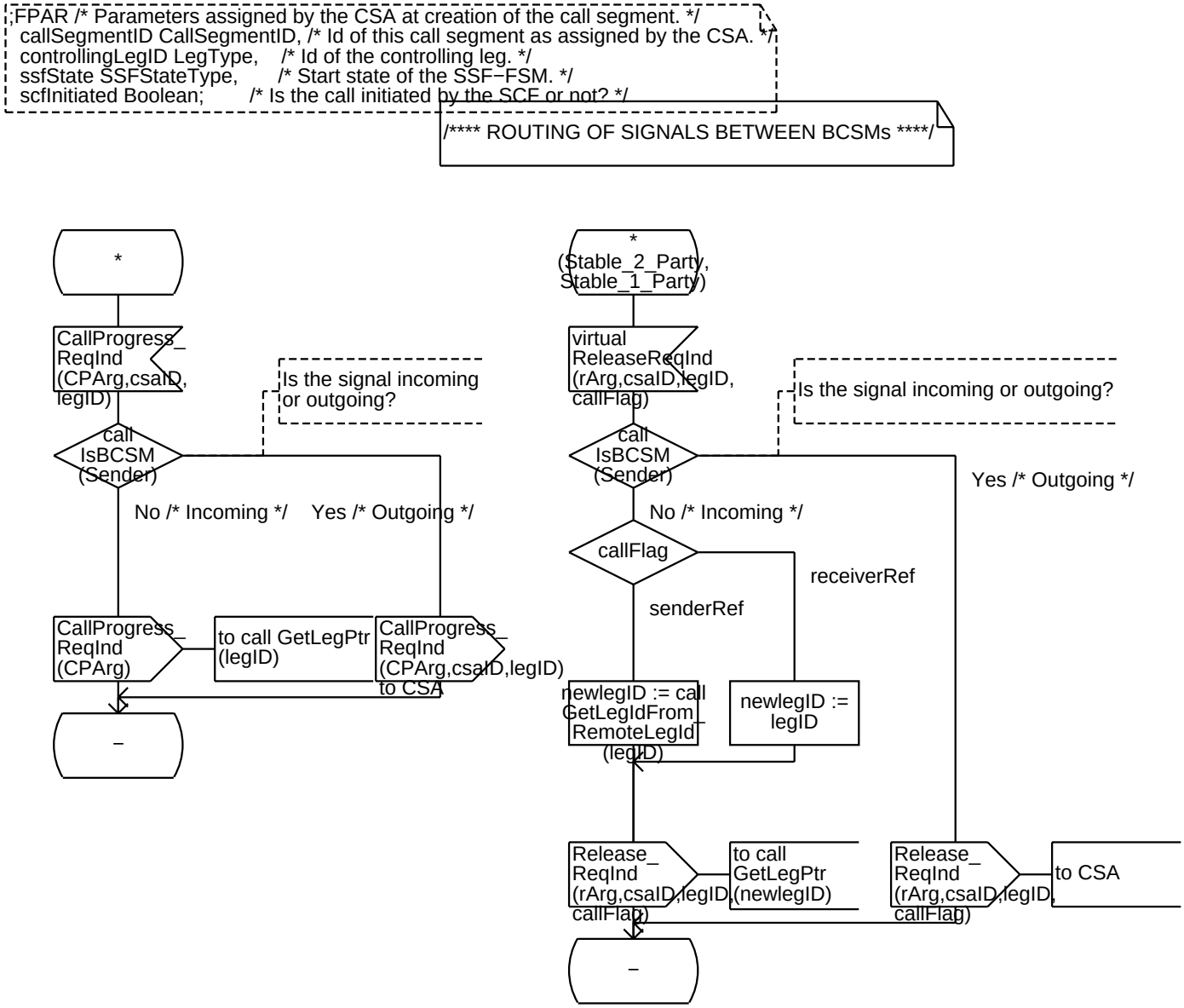
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 23(27)

```
;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? */
```



Annex A:

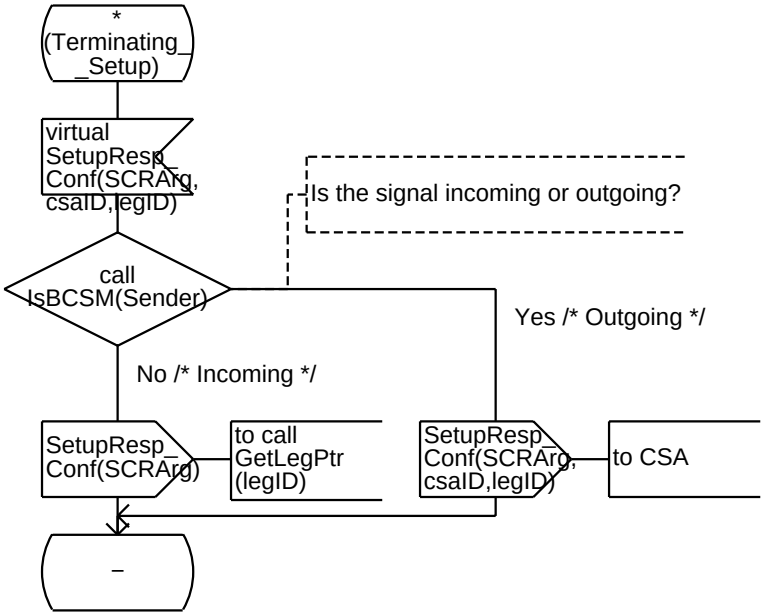
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 24(27)

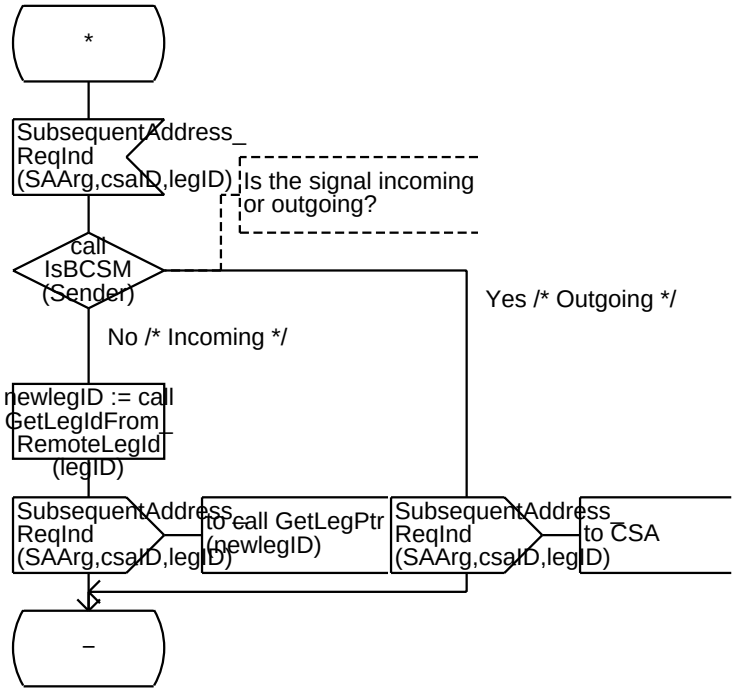


Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 25(27)

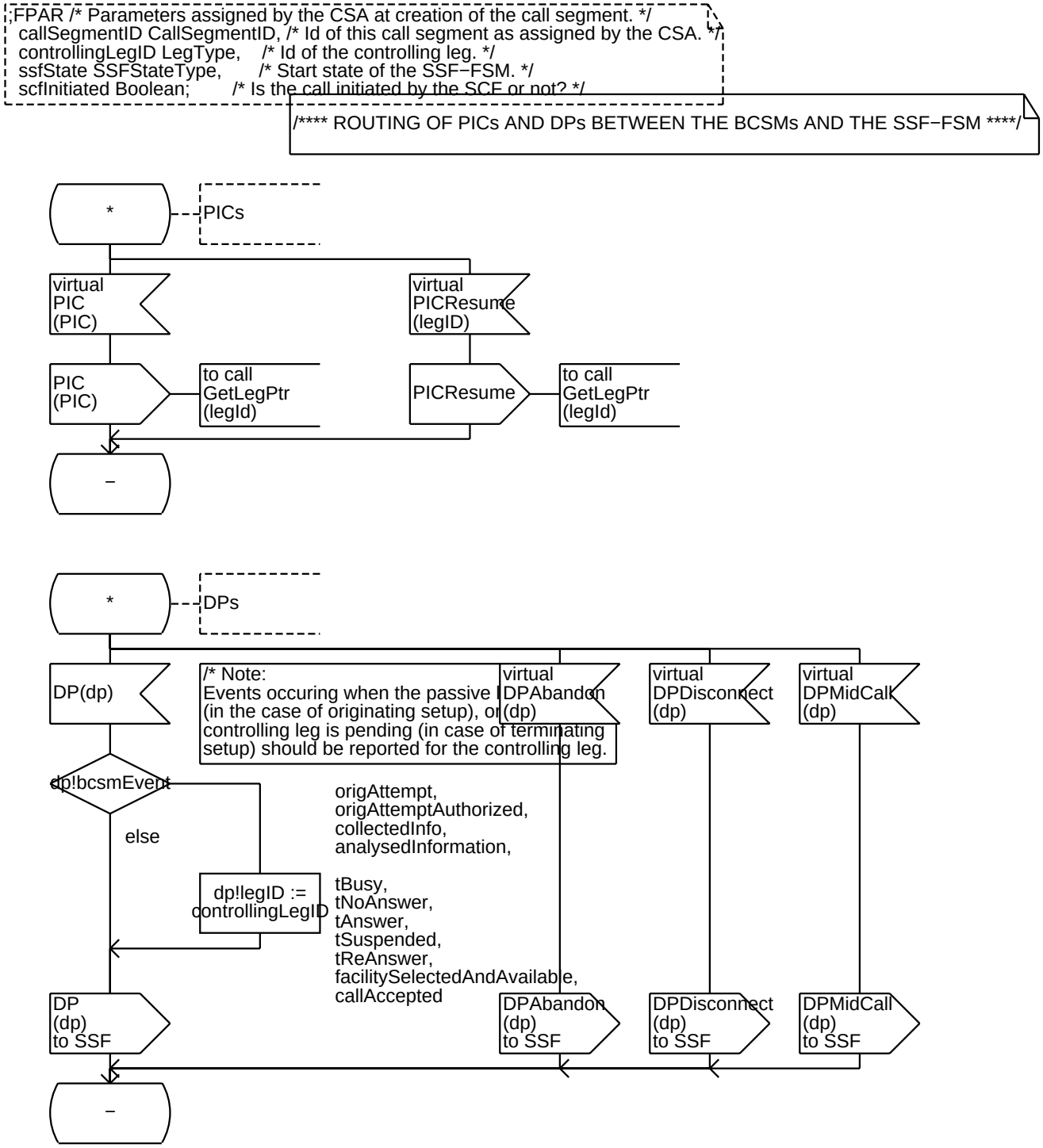
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
 callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
 controllingLegID LegType,    /* Id of the controlling leg. */
 ssfState SSFStateType,       /* Start state of the SSF−FSM. */
 scfInitiated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





Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 26(27)

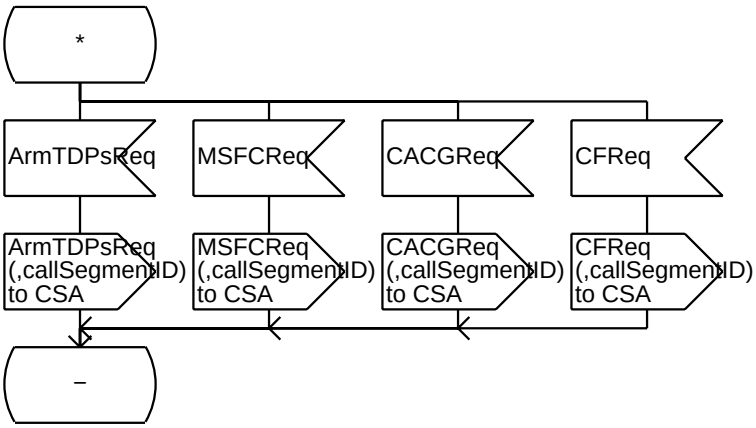


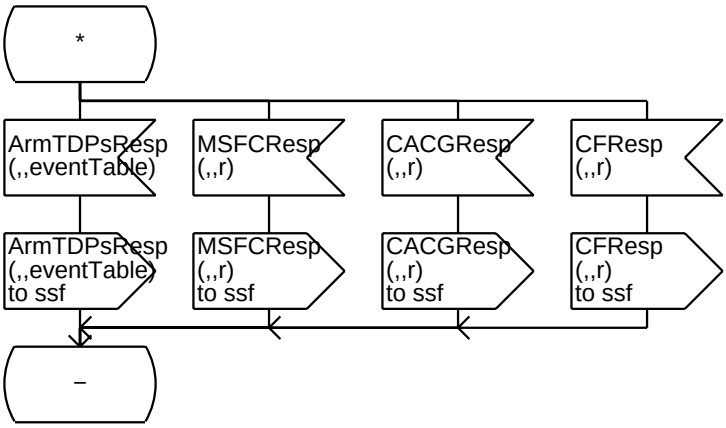
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> CallSegment 27(27)

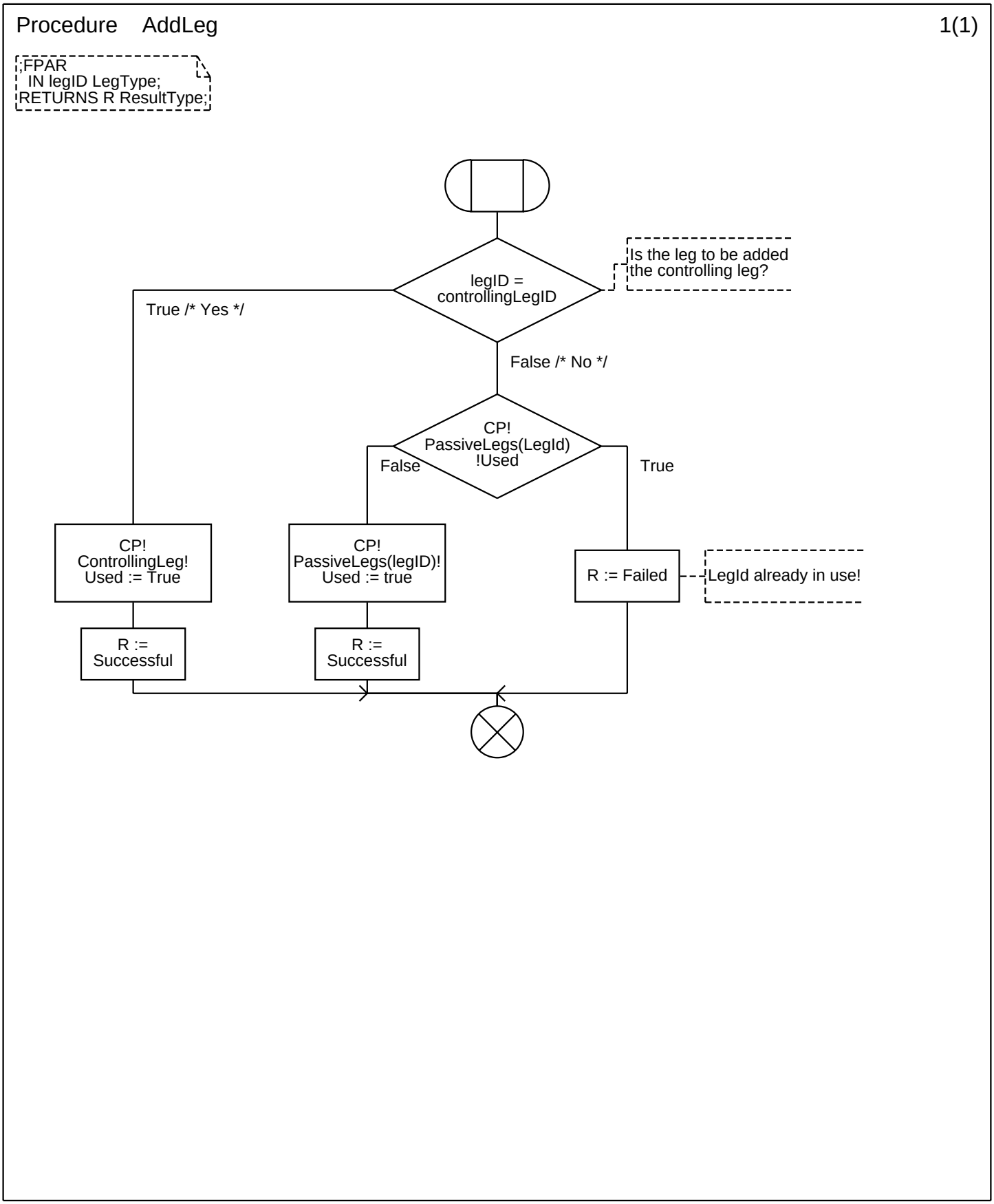
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
 callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
 controllingLegID LegType,    /* Id of the controlling leg. */
 ssfState SSFStateType,       /* Start state of the SSF−FSM. */
 scfInitiated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

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



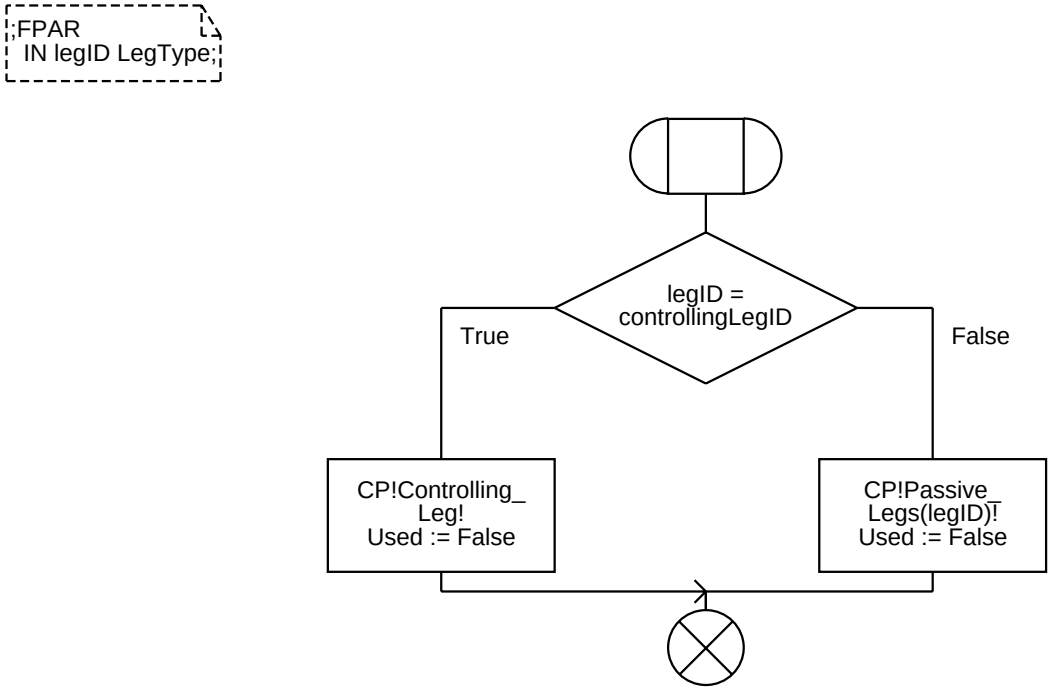


Annex A:


Procedure AddLeg
1(1)
;FPAR
IN legID LegType;
RETURNS R ResultType;
legID =
controllingLegID
Is the leg to be added
the controlling leg?
True /* Yes */
False /* No */
CP!
PassiveLegs(LegId)
!Used
False
True
CP!
ControllingLeg!
Used := True
CP!
PassiveLegs(legID)!
Used := true
R := Failed
LegId already in use!
R :=
Successful
R :=
Successful

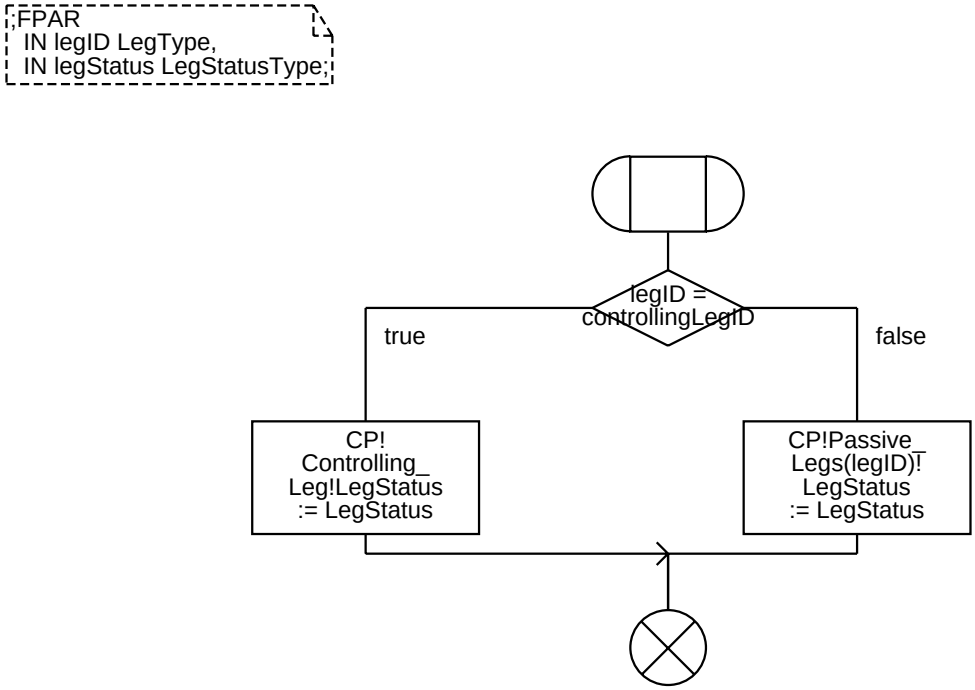
Annex A:

Procedure <<Process Type CallSegment>> RemoveLeg 1(1)

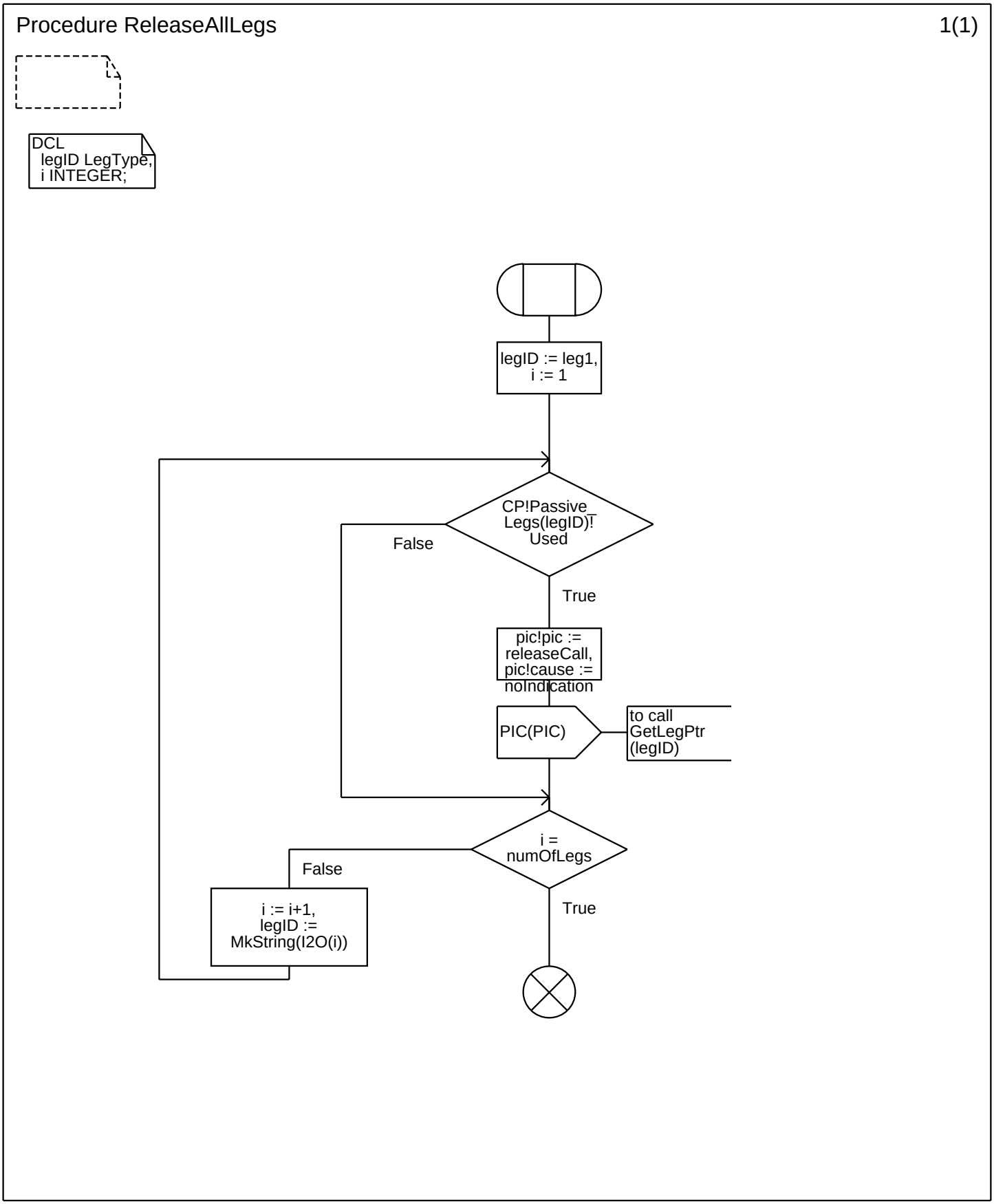


Annex A:

Procedure SetLegStatus 1(1)

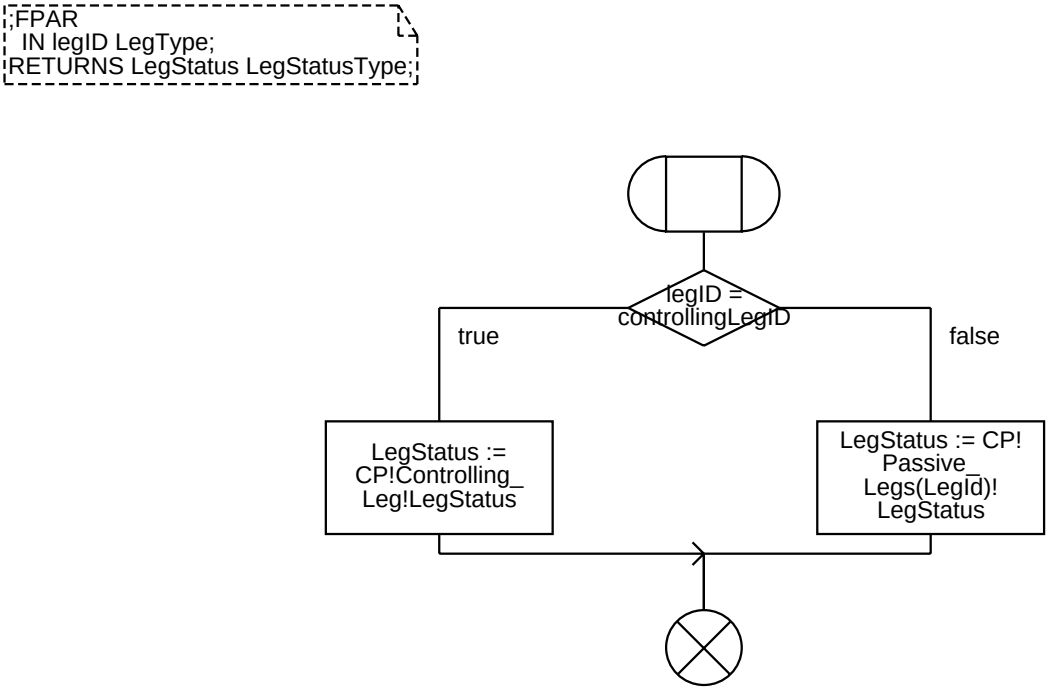


Annex A:


Procedure ReleaseAllLegs
1(1)
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))

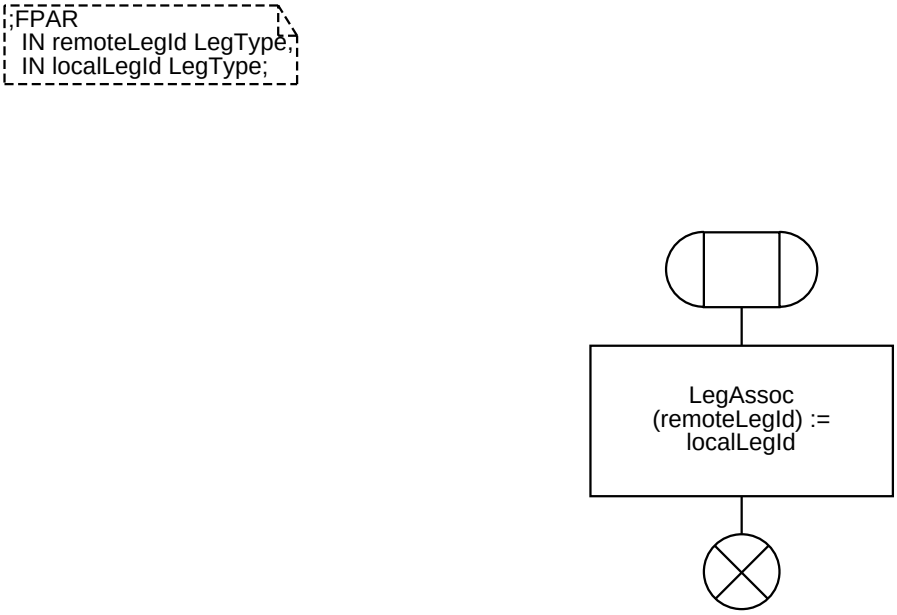
Annex A:

Procedure GetLegStatus 1(1)



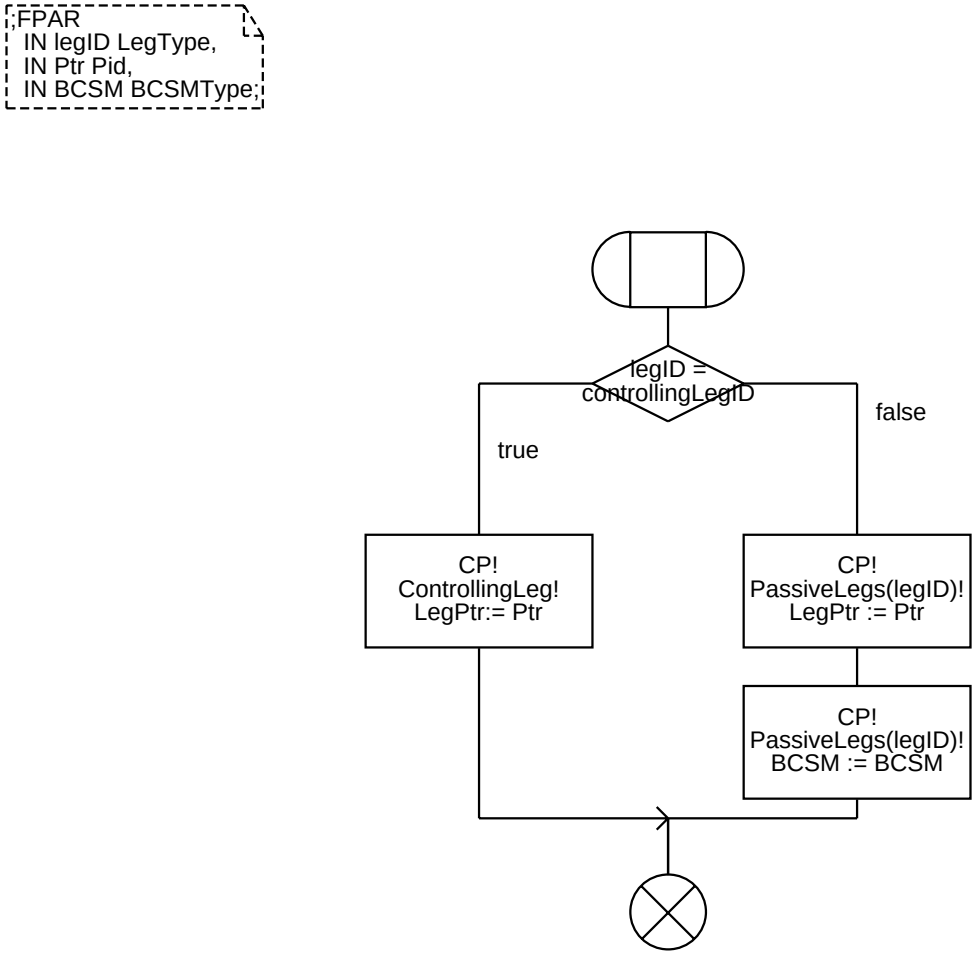
Annex A:

Procedure <<Process Type CallSegment>> SetLegAssoc 1(1)



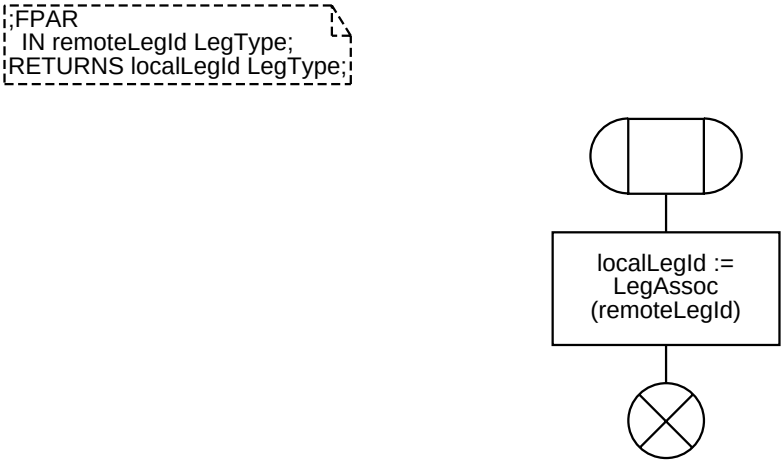
Annex A:

Procedure SetLegPtr 1(1)



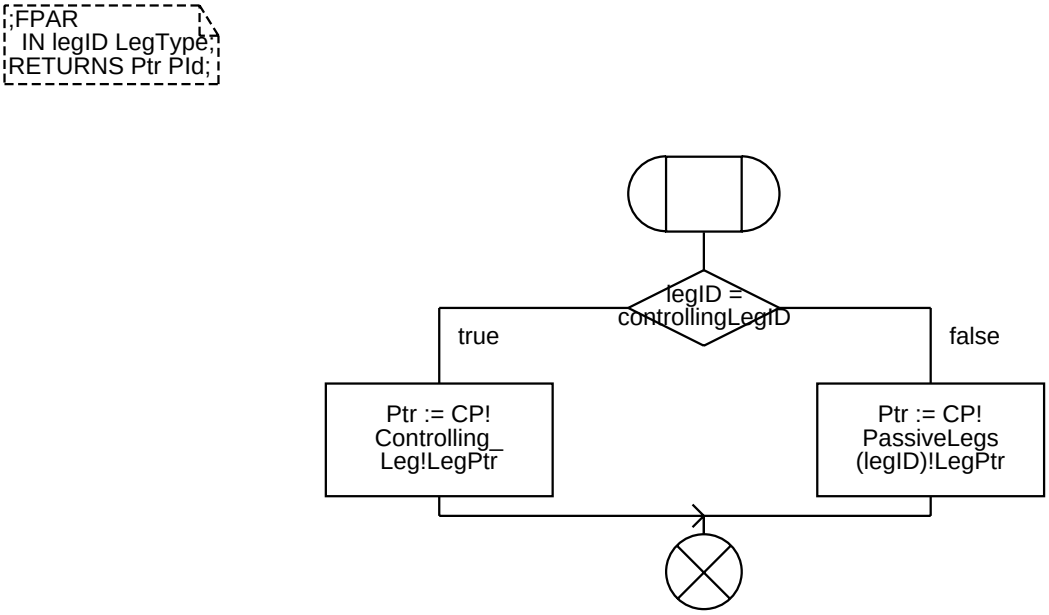
Annex A:

Procedure <<Process Type CallSegment>> GetLegIdfromRemoteLegId 1(1)

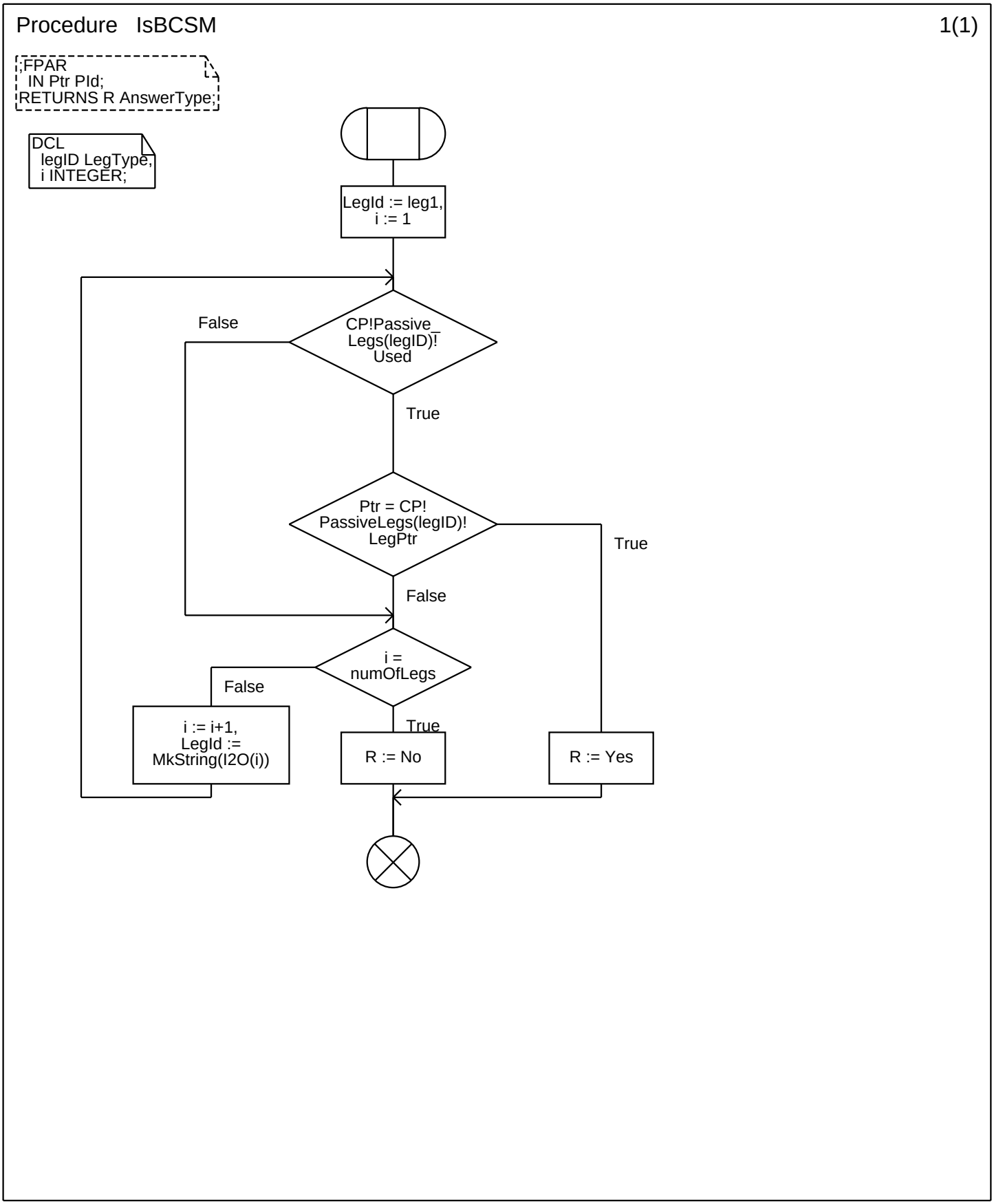


Annex A:

Procedure GetLegPtr 1(1)

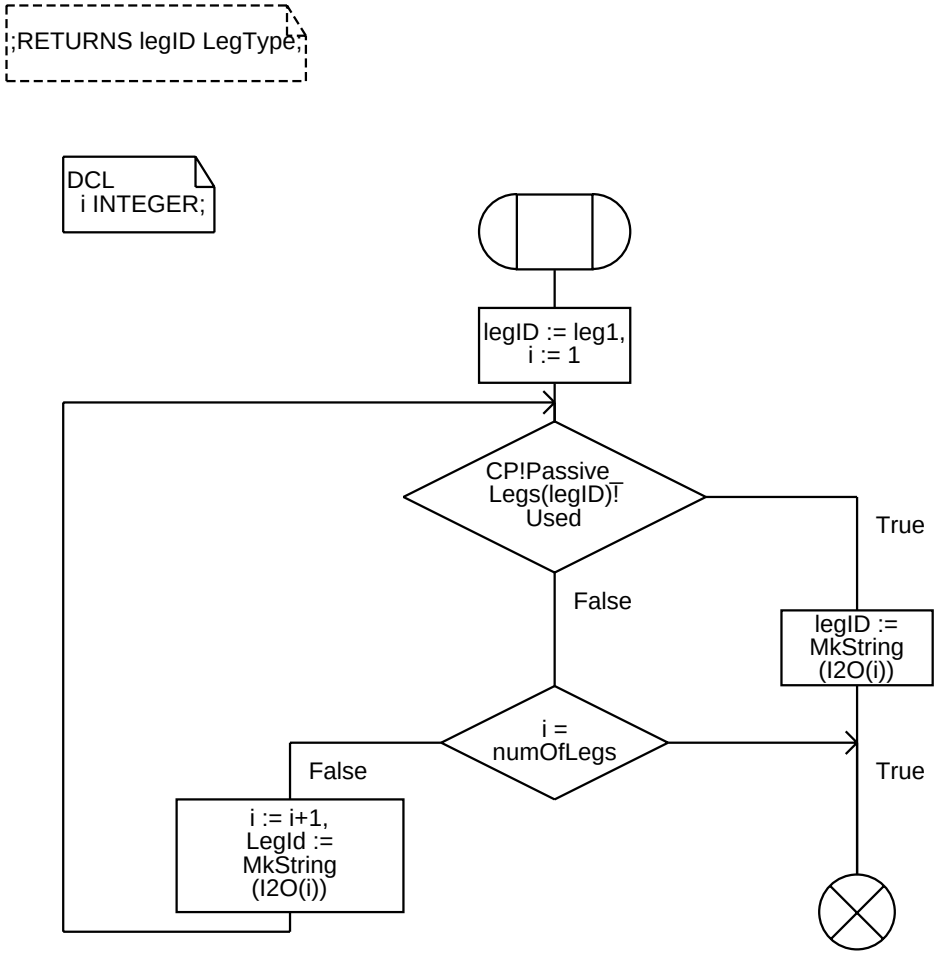


Annex A:

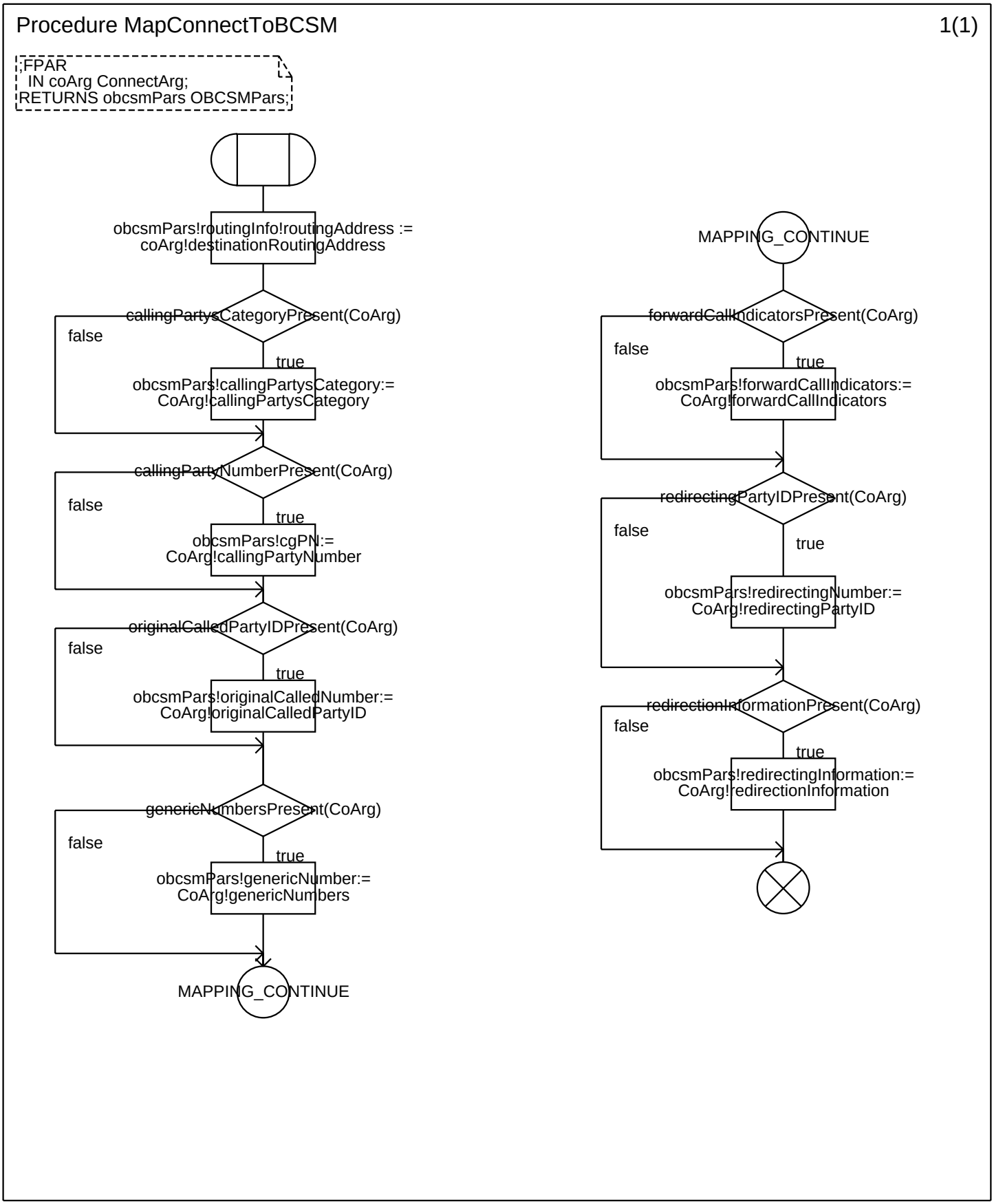

Procedure IsBCSM
1(1)
;FPAR
IN Ptr PId;
RETURNS R AnswerType;
DCL
legID LegType,
i INTEGER;
LegId := leg1,
i := 1
CP!Passive_
Legs(legID)!
Used
False
True
Ptr = CP!
PassiveLegs(legID)!
LegPtr
True
False
i =
numOfLegs
False
True
i := i+1,
LegId :=
MkString(I2O(i))
R := No
R := Yes

Annex A:

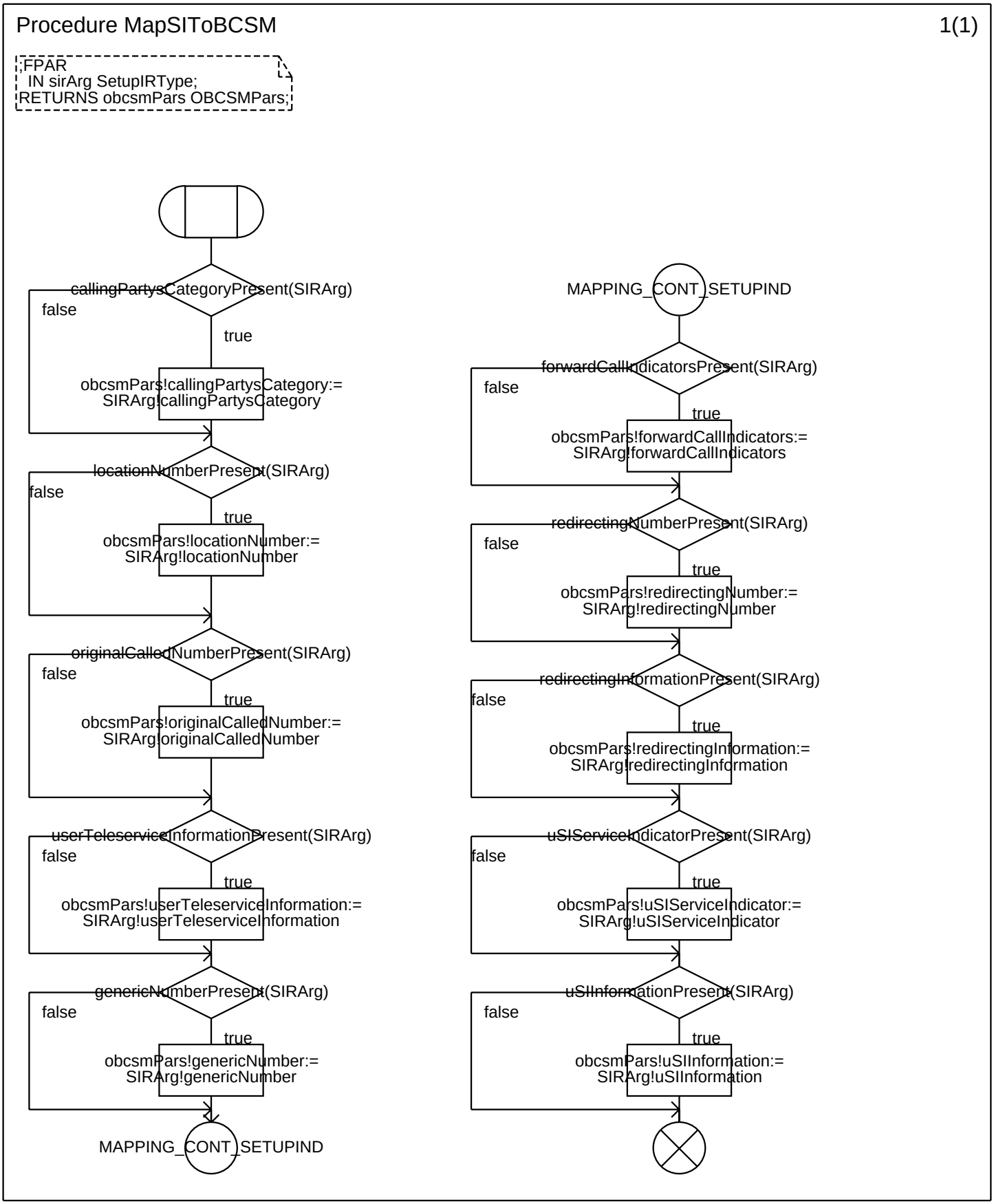
Procedure GetPassiveLegId 1(1)



Annex A:


Procedure MapConnectToBCSM
1(1)
;FPAR
IN coArg ConnectArg;
RETURNS obcsmPars OBCSMPars;
obcsmPars!routingInfo!routingAddress :=
coArg!destinationRoutingAddress
callingPartysCategoryPresent(CoArg)
false
true
obcsmPars!callingPartysCategory:=
CoArg!callingPartysCategory
callingPartyNumberPresent(CoArg)
false
true
obcsmPars!cgPN:=
CoArg!callingPartyNumber
originalCalledPartyIDPresent(CoArg)
false
true
obcsmPars!originalCalledNumber:=
CoArg!originalCalledPartyID
genericNumbersPresent(CoArg)
false
true
obcsmPars!genericNumber:=
CoArg!genericNumbers
MAPPING_CONTINUE
MAPPING_CONTINUE
forwardCallIndicatorsPresent(CoArg)
false
true
obcsmPars!forwardCallIndicators:=
CoArg!forwardCallIndicators
redirectingPartyIDPresent(CoArg)
false
true
obcsmPars!redirectingNumber:=
CoArg!redirectingPartyID
redirectionInformationPresent(CoArg)
false
true
obcsmPars!redirectingInformation:=
CoArg!redirectionInformation

Annex A:


Procedure MapSIToBCSM
1(1)
;FPAR
IN sirArg SetupIRType;
RETURNS obcsmPars OBCSMPars;
callingPartysCategoryPresent(SIRArg)
false
true
obcsmPars!callingPartysCategory:=
SIRArg!callingPartysCategory
locationNumberPresent(SIRArg)
false
true
obcsmPars!locationNumber:=
SIRArg!locationNumber
originalCalledNumberPresent(SIRArg)
false
true
obcsmPars!originalCalledNumber:=
SIRArg!originalCalledNumber
userTeleserviceInformationPresent(SIRArg)
false
true
obcsmPars!userTeleserviceInformation:=
SIRArg!userTeleserviceInformation
genericNumberPresent(SIRArg)
false
true
obcsmPars!genericNumber:=
SIRArg!genericNumber
MAPPING_CONT_SETUPIND
MAPPING_CONT_SETUPIND
forwardCallIndicatorsPresent(SIRArg)
false
true
obcsmPars!forwardCallIndicators:=
SIRArg!forwardCallIndicators
redirectingNumberPresent(SIRArg)
false
true
obcsmPars!redirectingNumber:=
SIRArg!redirectingNumber
redirectingInformationPresent(SIRArg)
false
true
obcsmPars!redirectingInformation:=
SIRArg!redirectingInformation
uSIServiceIndicatorPresent(SIRArg)
false
true
obcsmPars!uSIServiceIndicator:=
SIRArg!uSIServiceIndicator
uSIInformationPresent(SIRArg)
false
true
obcsmPars!uSIInformation:=
SIRArg!uSIInformation

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_1(25)>> SSF_FSM

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType;
  SIGNALSET DPAbandon;
```

```
/**** 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;

/* The SSF−FSM maintains information about the number of
outstanding requests per leg. */

NEWTYPE OutstandingRequestsType
  ARRAY(LegType, Integer)
ENDNEWTYPE;


NEWTYPE DialogPrimitive
 LITERALS
  primBegin,
  Continue,
  primEnd,
  Abort;
ENDNEWTYPE;
```

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 2(25)

```
;FPAR
 startState SSFStateType,
 scfInitiated Boolean,
 controllingLegID LegType;
 SIGNALSET DPAbandon;
```

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

```
DCL
 /* IN CS−1 operation arguments. */
 acArg ApplyChargingArg,
 acrArg ApplyChargingReportArg,
 ariArg AssistRequestInstructionsArg,
 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,
 rrBCSMEArg RequestReportBCSMEventArg,
 rtARg ResetTimerArg,
 sciArg SendChargingInformationArg,

 paArg PlayAnnouncementArg,
 pacuiArg PromptAndCollectUserInformationArg,

 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,
 outstandingRequests OutstandingRequestsType,

 storedDP DPArg,          /* Last received TDP−R or EDP−R. */
 dpStored Boolean := false, /* Flag to indicate if a DP is stored.

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

Annex A:

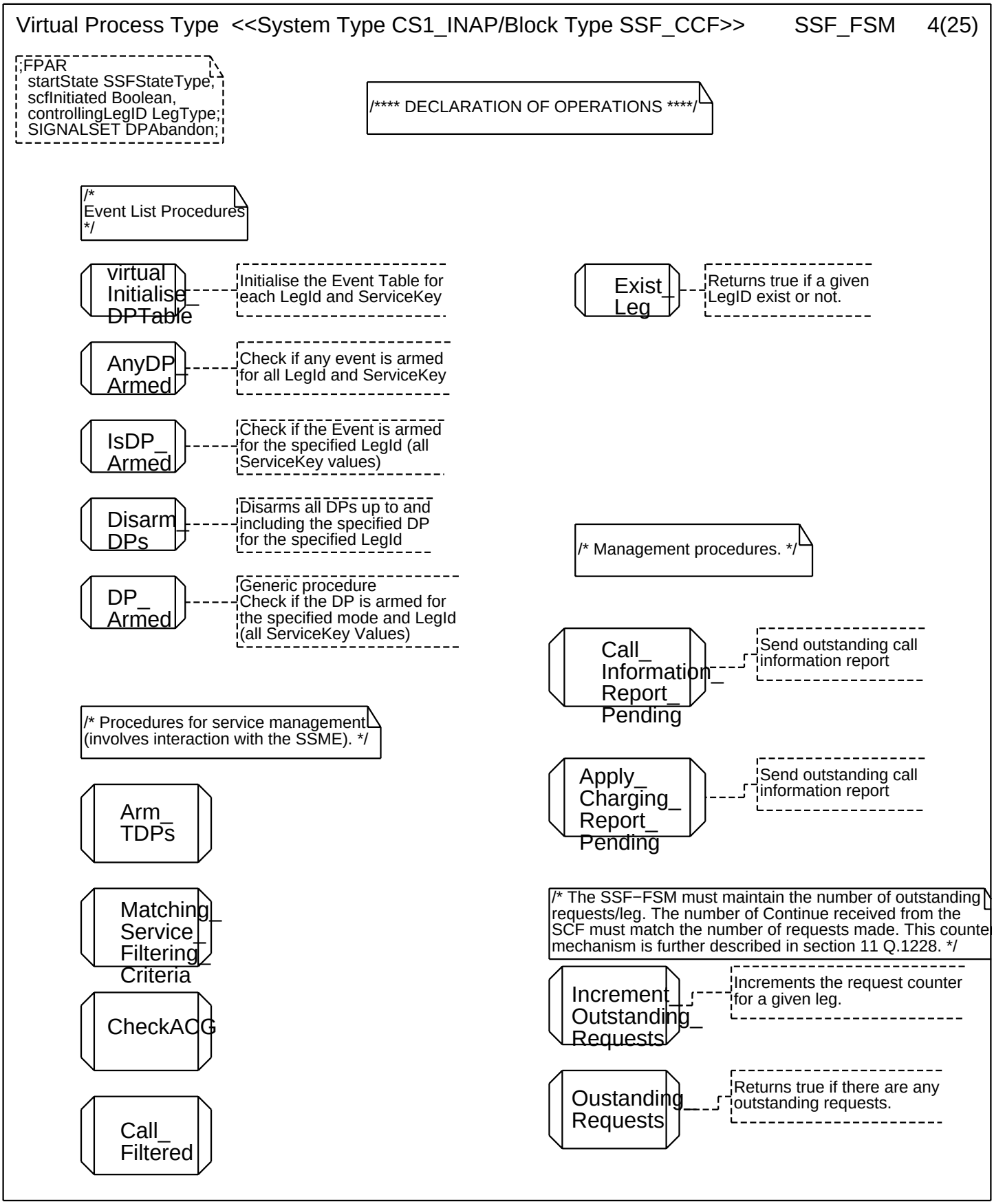
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 3(25)

```
;FPAR
 startState SSFStateType,
 scfInitiated Boolean,
 controllingLegID LegType;
 SIGNALSET DPAbandon;
```

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

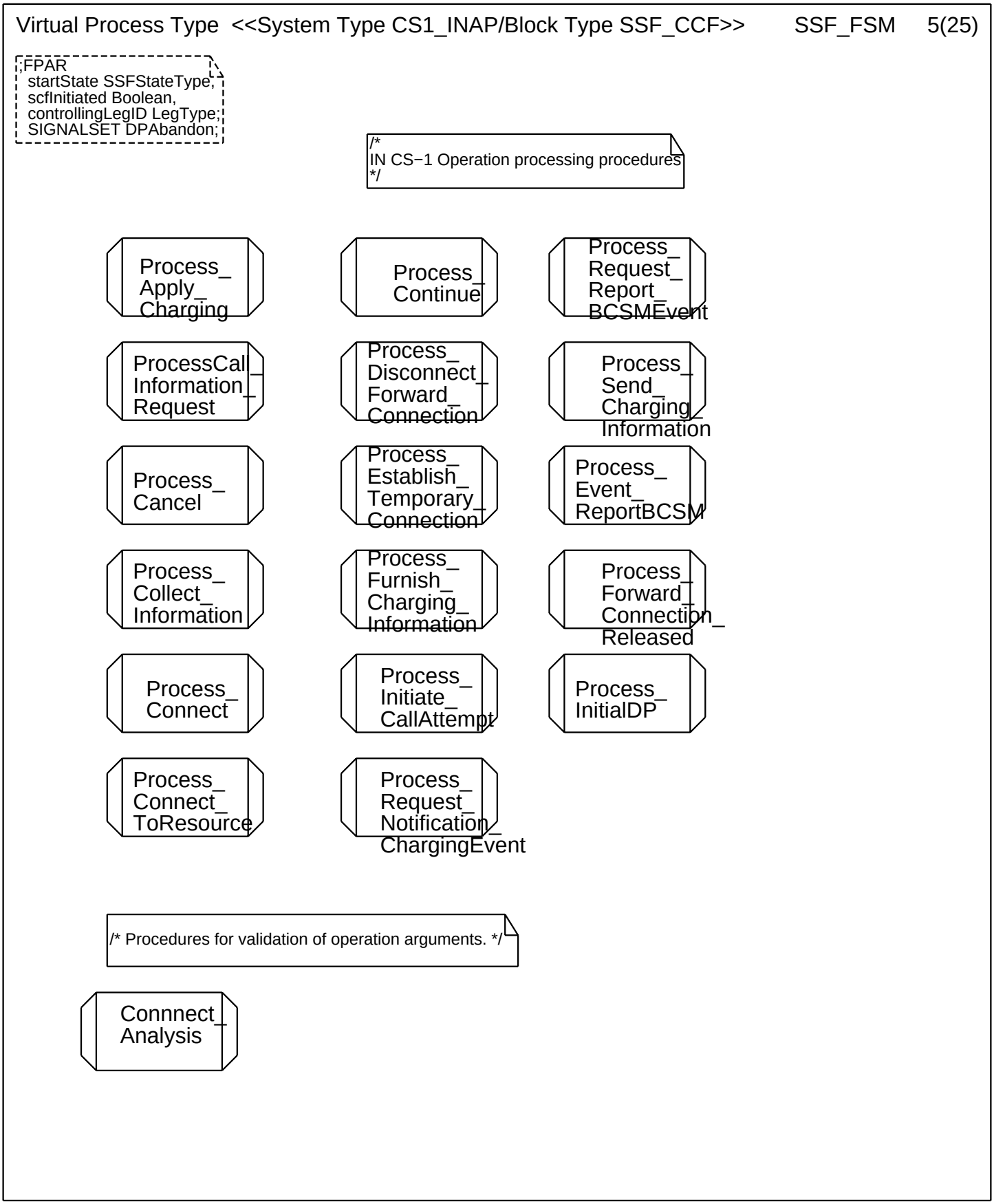
```
/*
The timer Tssf is used to prevent excessive
call supsension time and to guard the
association between the SSF and the SCF.
*/

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;
```

Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 4(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/**** 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
Returns true 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
/* The SSF−FSM must maintain the number of outstanding requests/leg. The number of Continue received from the SCF must match the number of requests made. This counter mechanism is further described in section 11 Q.1228. */
Increment_Outstanding_Requests
Increments the request counter for a given leg.
Oustanding_Requests
Returns true if there are any outstanding requests.

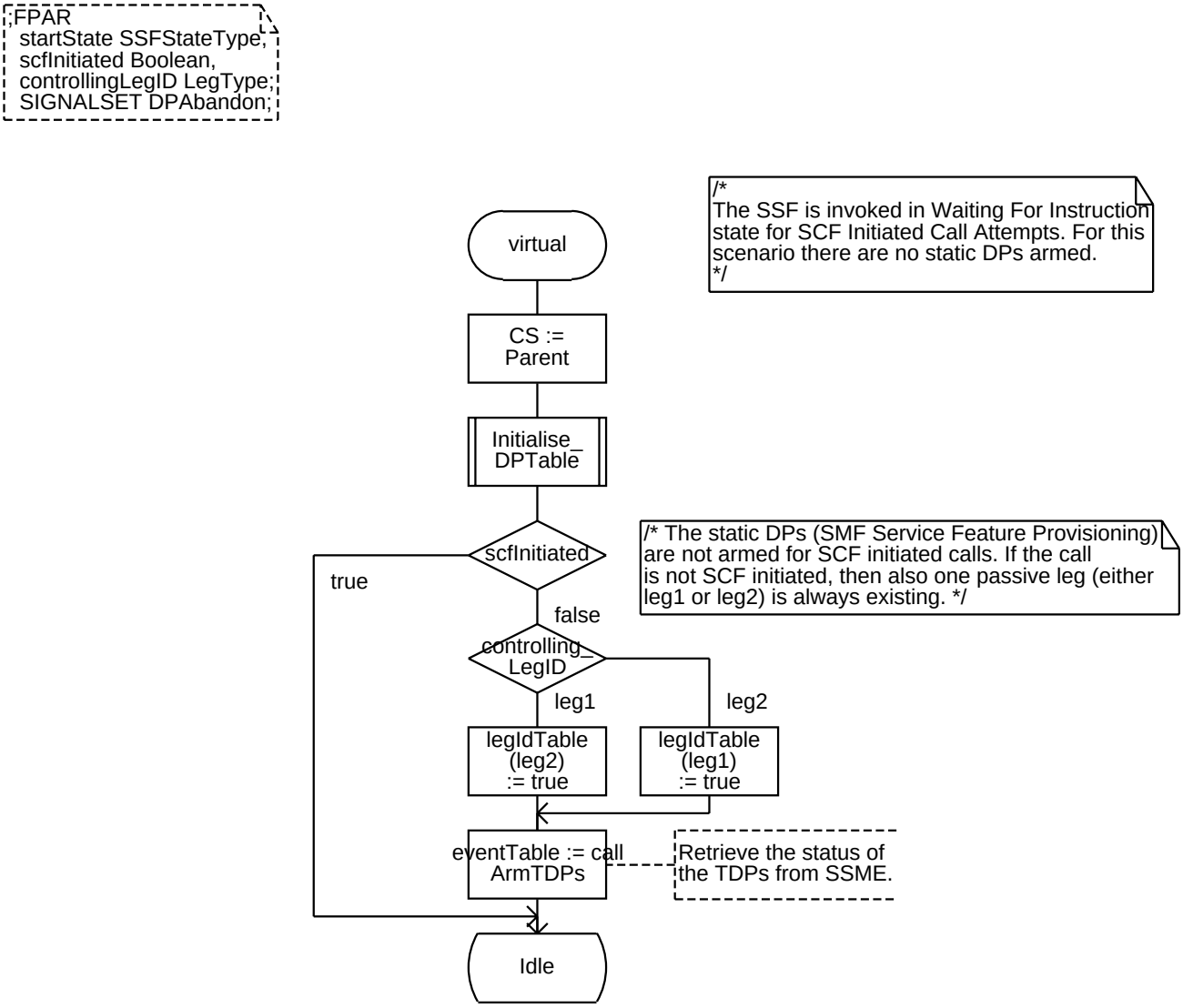
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 5(25)

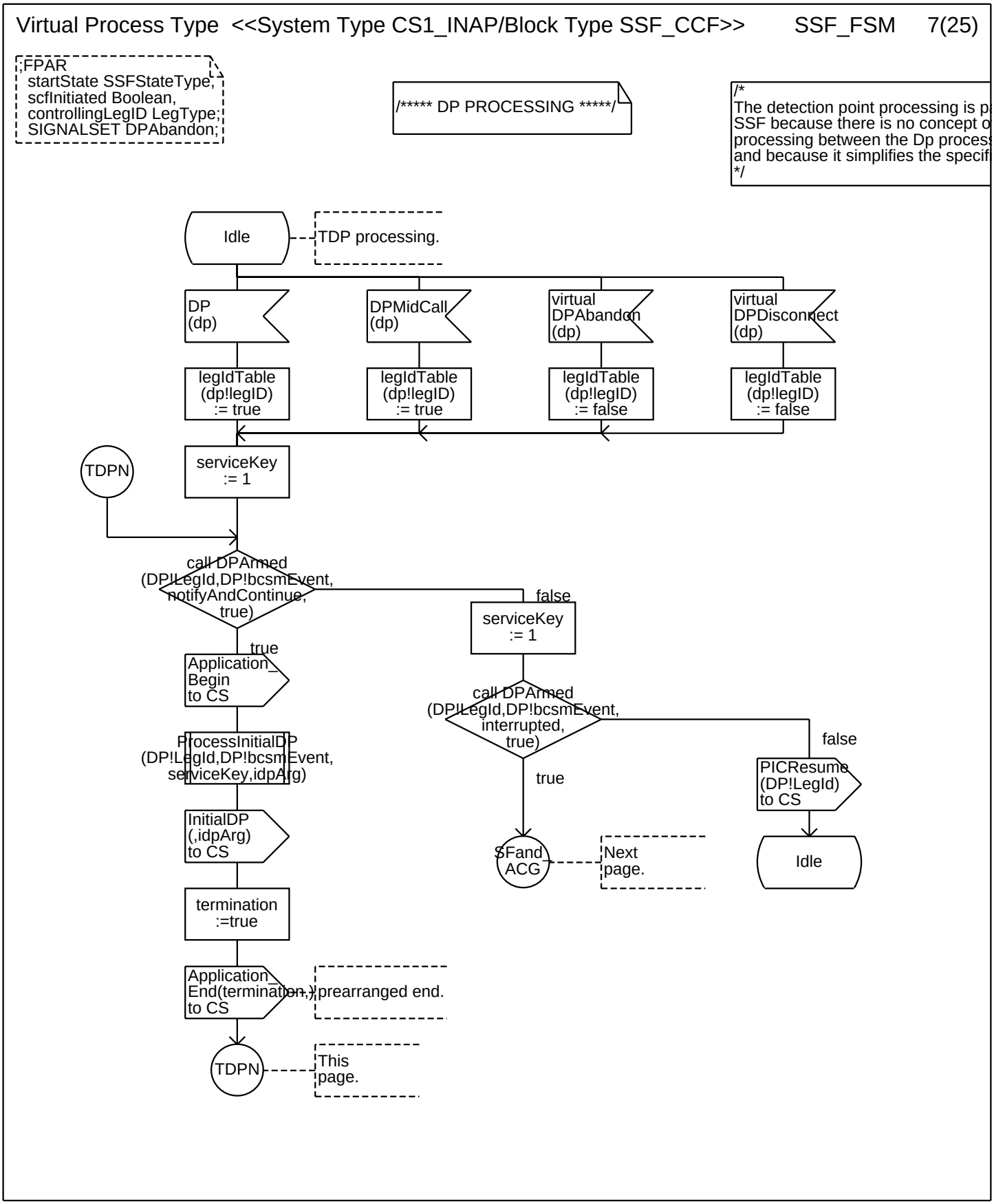


Annex A:

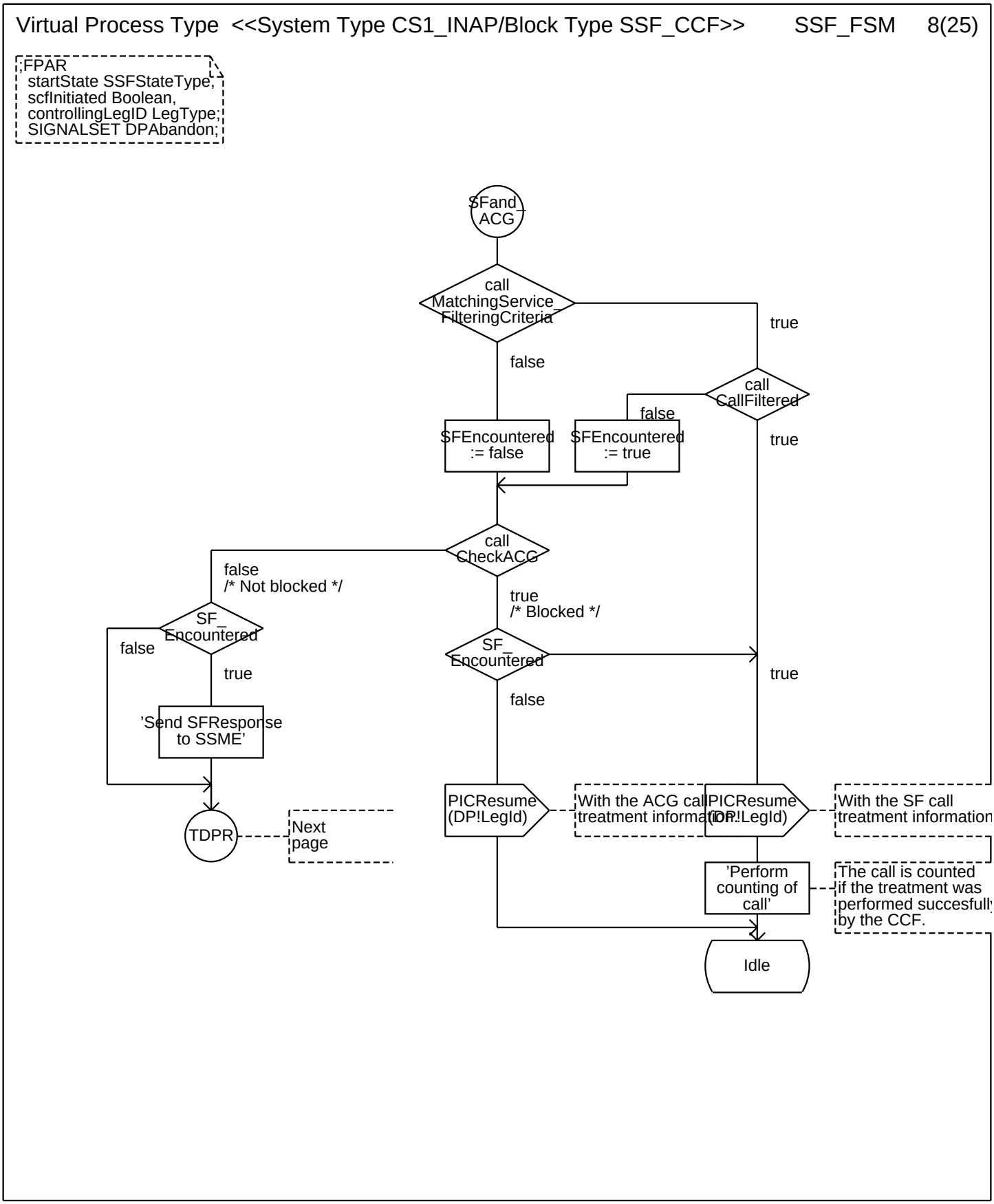
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 6(25)



Annex A:

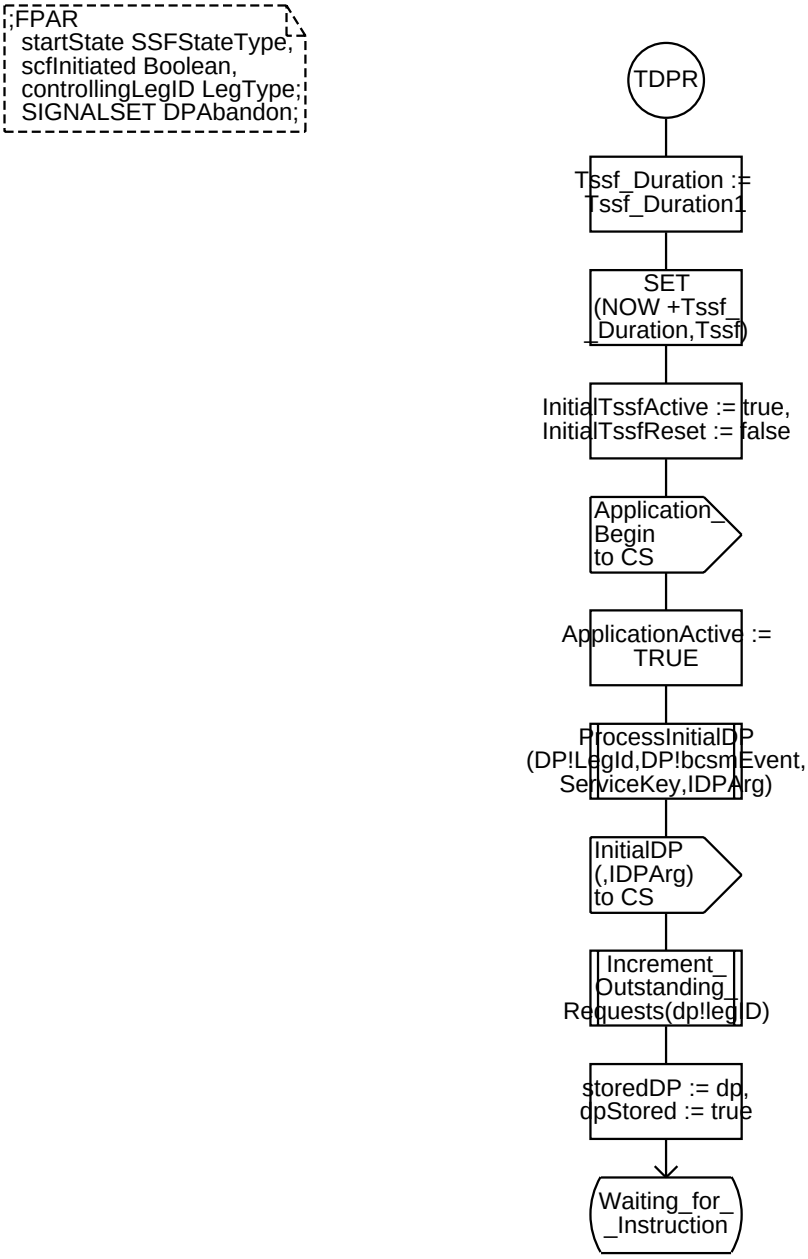

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 7(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/***** DP PROCESSING *****/
/*
The detection point processing is p
SSF because there is no concept o
processing between the Dp process
and because it simplifies the specif
*/
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
serviceKey := 1
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue, true)
false
serviceKey := 1
true
Application_Begin to CS
call DPArmed (DP!LegId,DP!bcsmEvent, interrupted, true)
false
ProcessInitialDP (DP!LegId,DP!bcsmEvent, serviceKey,idpArg)
true
PICResume (DP!LegId) to CS
InitialDP (,idpArg) to CS
SFand ACG
Next page.
Idle
termination :=true
Application_End(termination,) to CS
prearranged end.
TDPN
This page.

Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 8(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
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

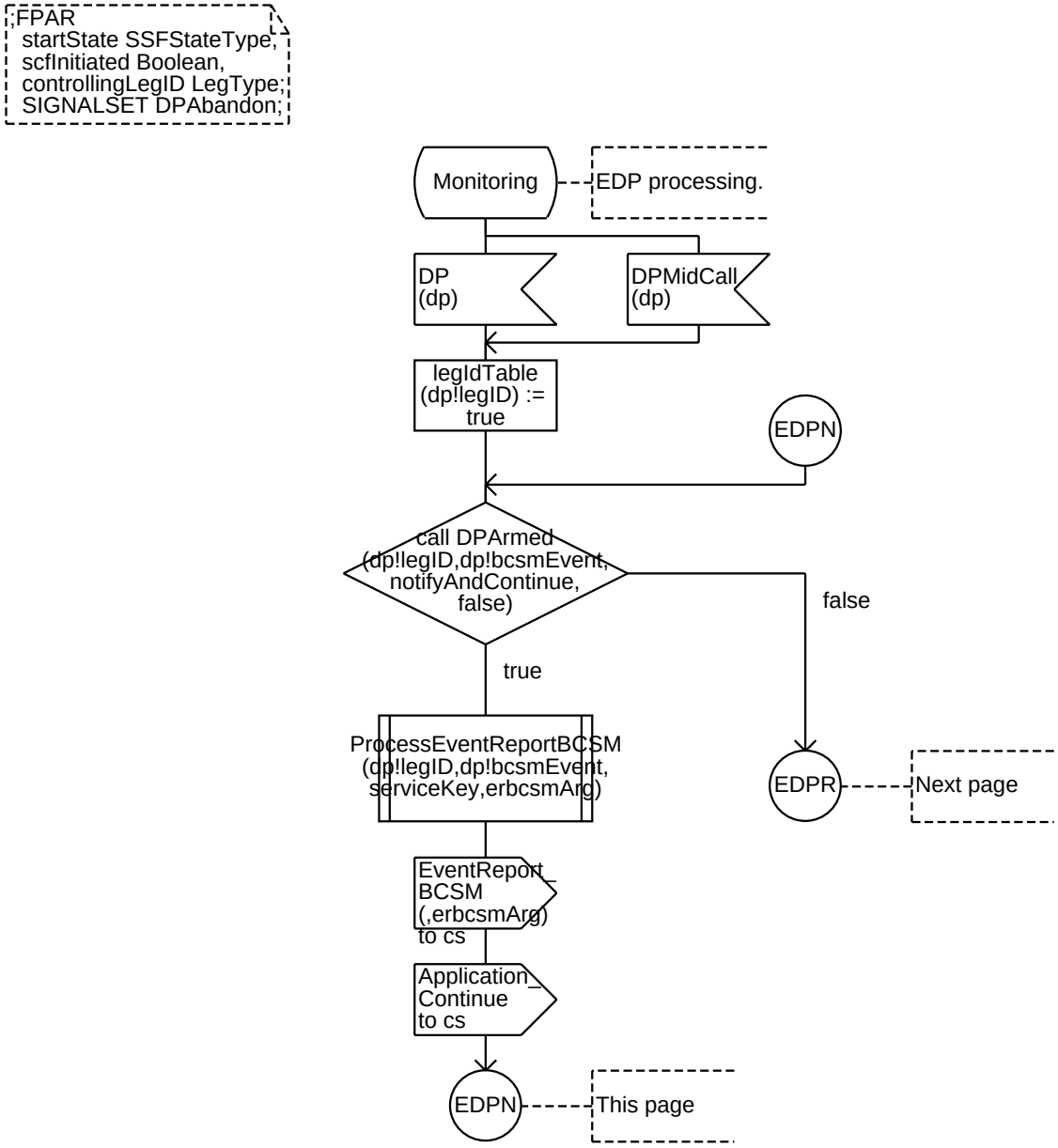
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 9(25)

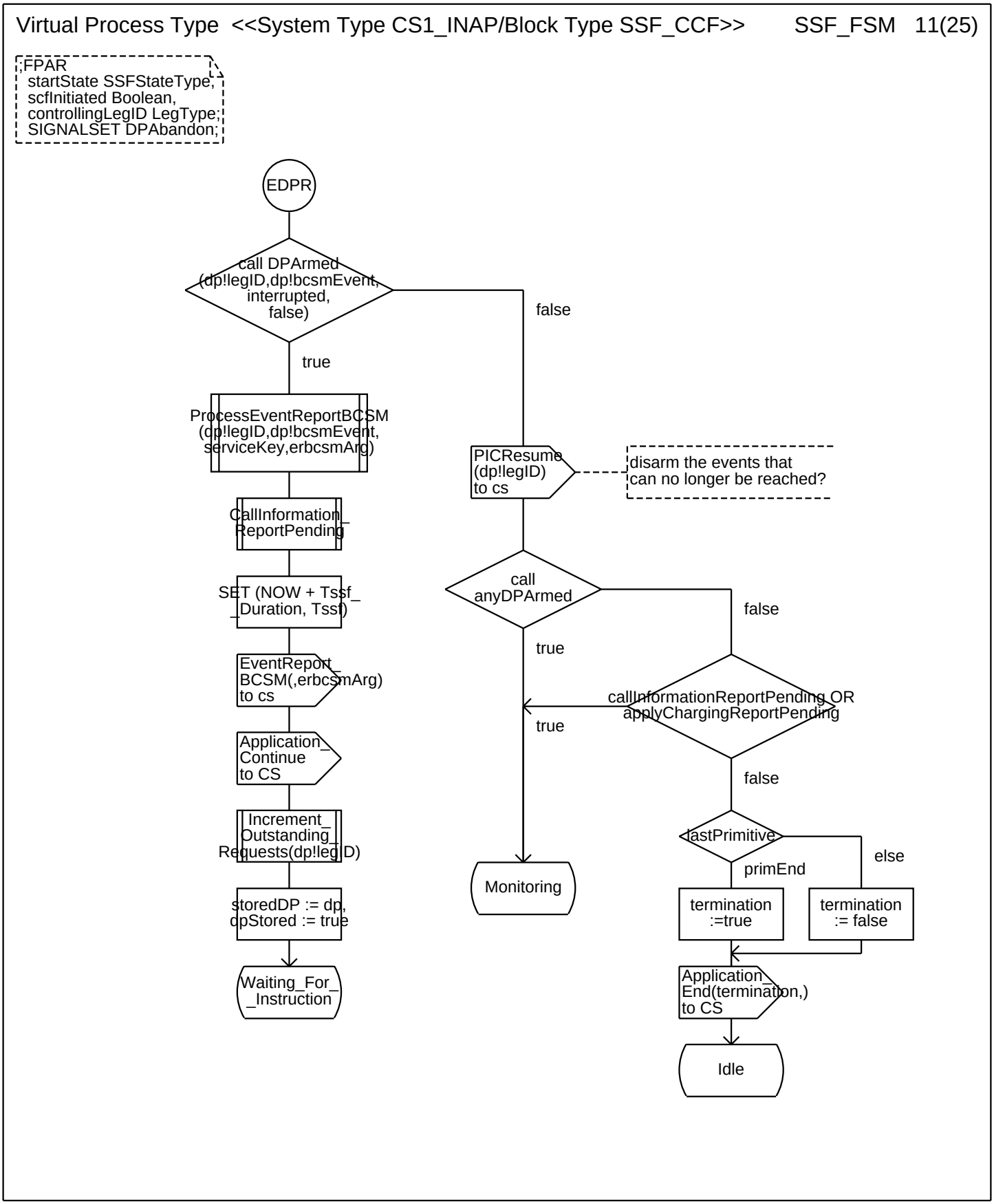


Annex A:

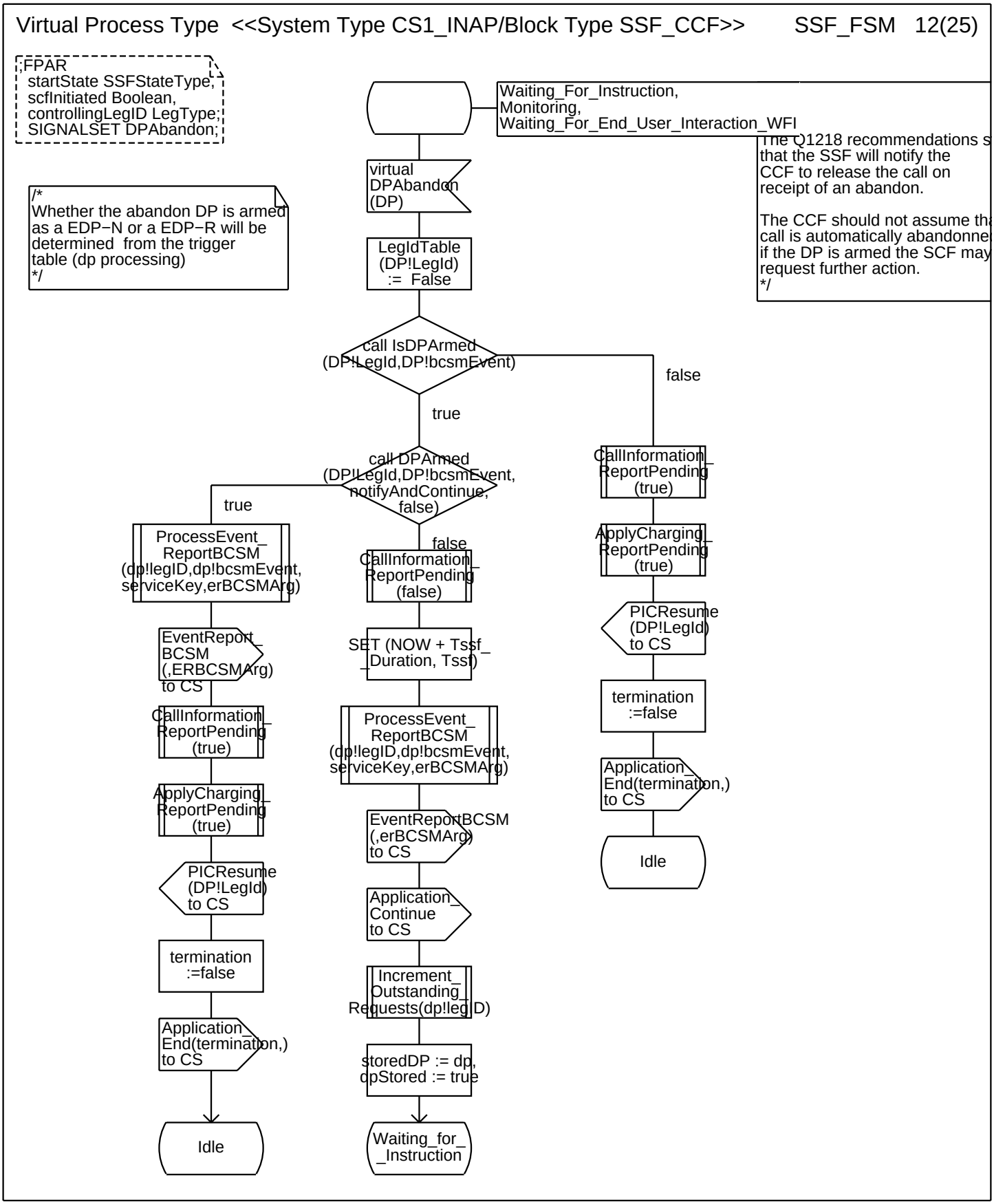
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 10(25)



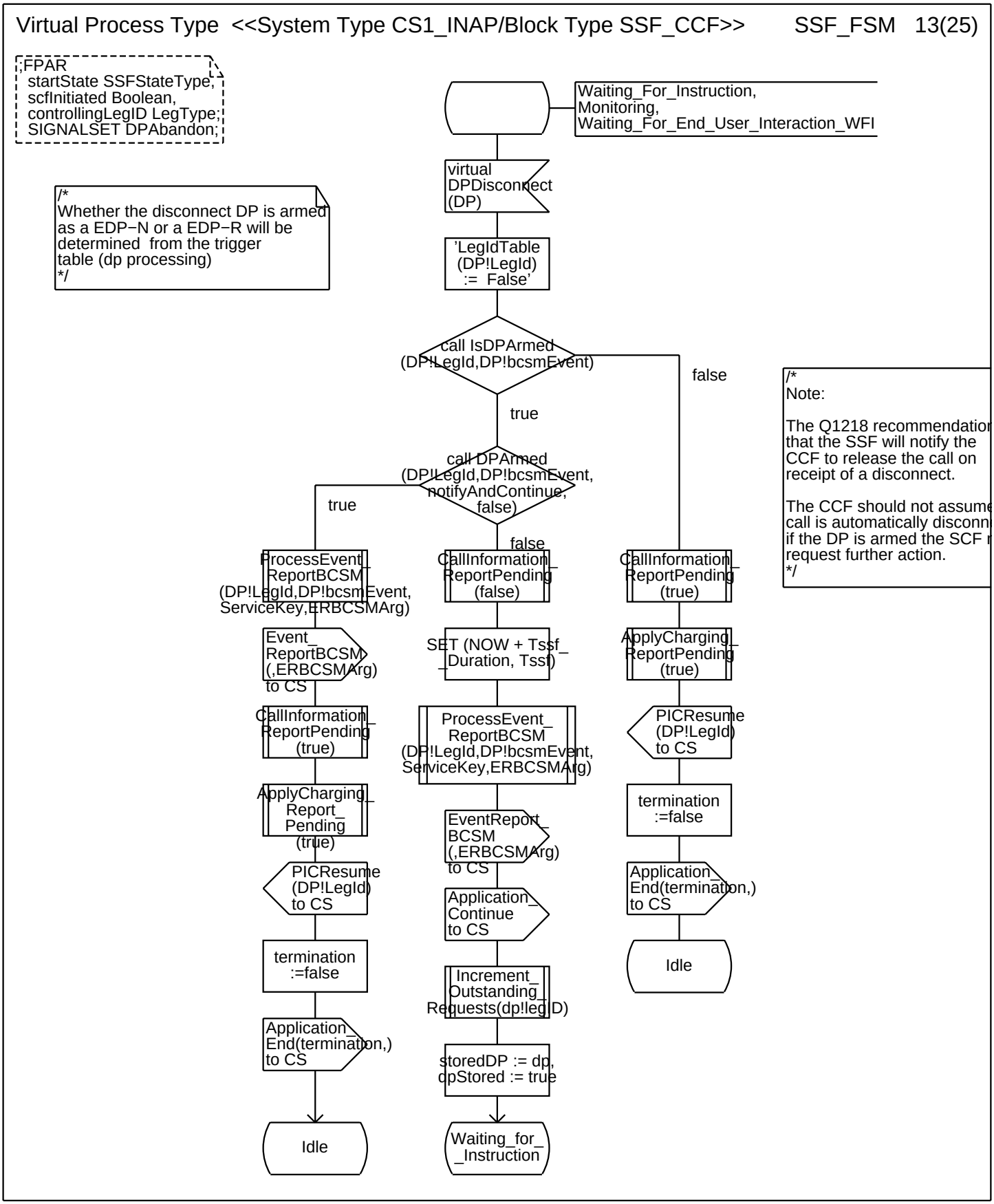
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 11(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
EDPR
call DPArmed (dp!legID,dp!bcsmEvent, interrupted, false)
false
true
ProcessEventReportBCSM (dp!legID,dp!bcsmEvent, serviceKey,erbcsmArg)
CallInformation_ ReportPending
SET (NOW + Tssf_ _Duration, Tssf)
EventReport_ BCSM(,erbcsmArg) to cs
Application_ Continue to CS
Increment_ Outstanding_ Requests(dp!legID)
storedDP := dp, dpStored := true
Waiting_For_ _Instruction
PICResume (dp!legID) to cs
disarm the events that can no longer be reached?
call anyDPArmed
false
true
callInformationReportPending OR applyChargingReportPending
true
false
lastPrimitive
else
primEnd
Monitoring
termination :=true
termination := false
Application_ End(termination,) to CS
Idle

Annex A:

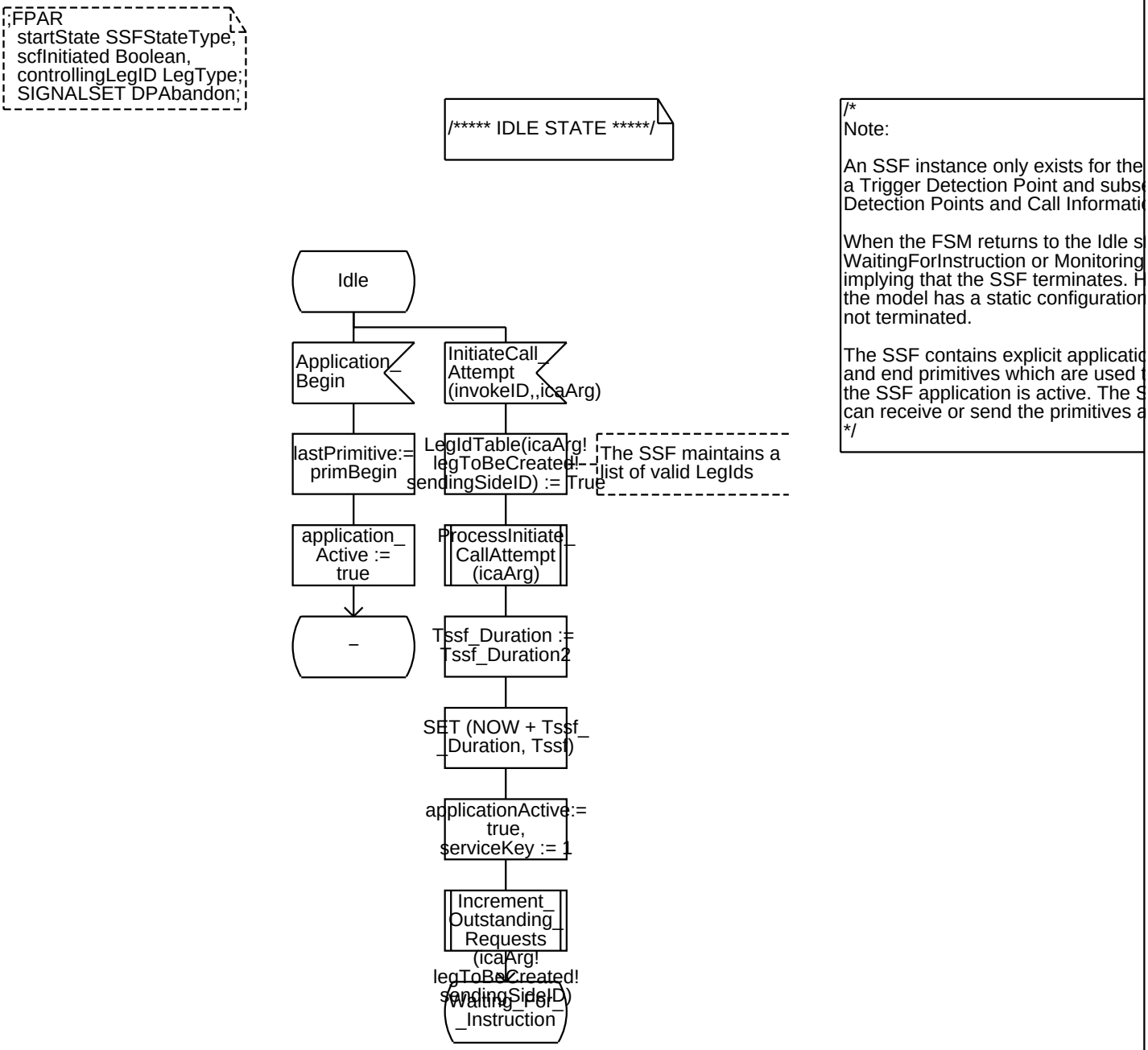

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 12(25)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/*
Whether the abandon DP is armed as a EDP−N or a EDP−R will be determined from the trigger table (dp processing)
*/
Waiting_For_Instruction,
Monitoring,
Waiting_For_End_User_Interaction_WFI
The Q1218 recommendations s
that the SSF will notify the CCF to release the call on receipt of an abandon.
The CCF should not assume tha
call is automatically abandonne
if the DP is armed the SCF may
request further action.
*/
virtual DPAbandon (DP)
LegIdTable (DP!LegId) := False
call IsDPArmed (DP!LegId,DP!bcsmEvent)
false
true
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue, false)
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
Increment_ Outstanding_ Requests(dp!legID)
storedDP := dp, dpStored := true
Waiting_for_ _Instruction
CallInformation_ ReportPending (true)
ApplyCharging_ ReportPending (true)
PICResume (DP!LegId) to CS
termination :=false
Application_ End(termination,) to CS
Idle

Annex A:

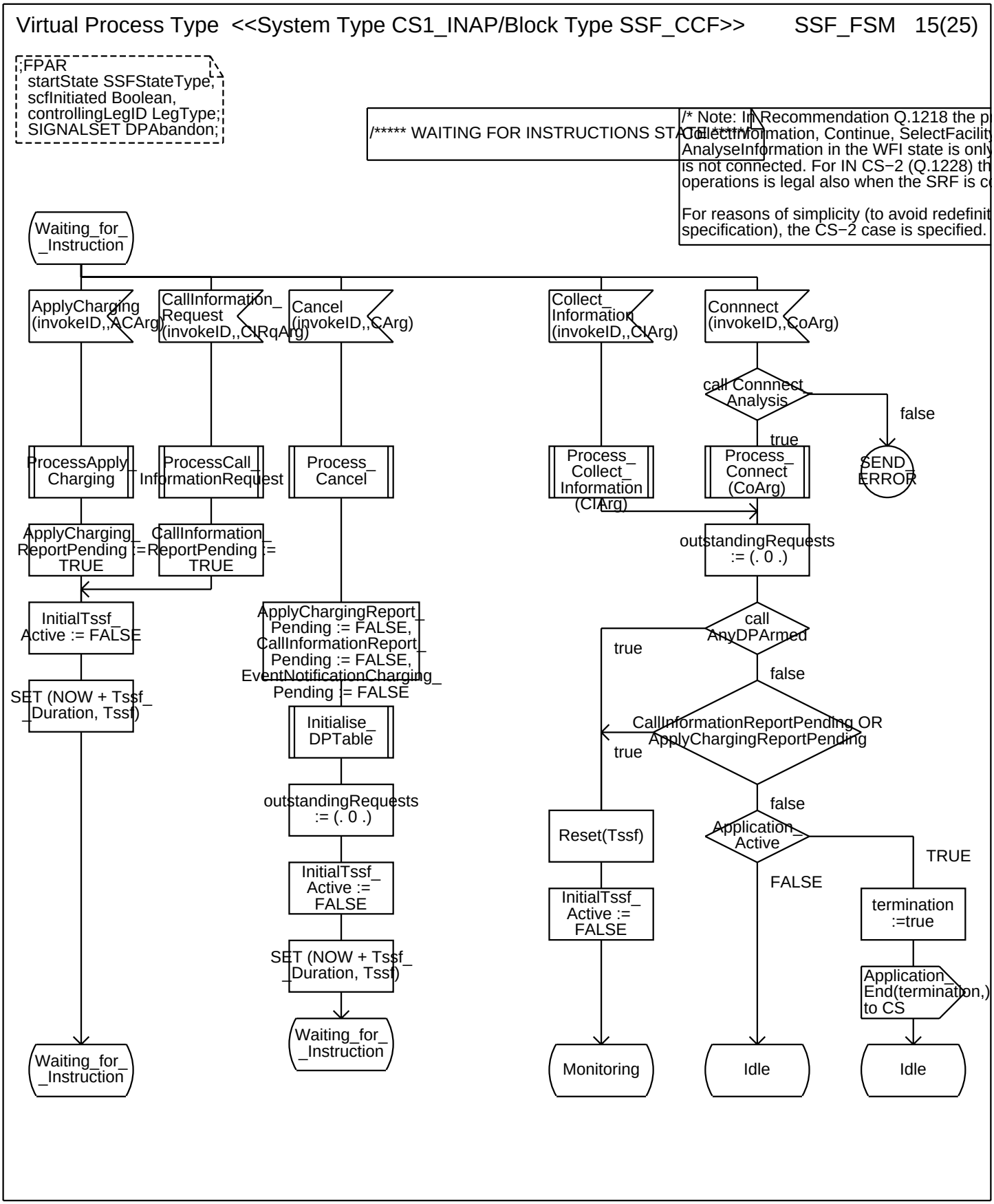

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 13(25)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/*
Whether the disconnect DP is armed as a EDP−N or a EDP−R will be determined from the trigger table (dp processing)
*/
Waiting_For_Instruction,
Monitoring,
Waiting_For_End_User_Interaction_WFI
virtual DPDisconnect (DP)
'LegIdTable (DP!LegId) := False'
call IsDPArmed (DP!LegId,DP!bcsmEvent)
true
false
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue, false)
true
false
/*
Note:
The Q1218 recommendation that the SSF will notify the CCF to release the call on receipt of a disconnect.
The CCF should not assume call is automatically disconn if the DP is armed the SCF r request further action.
*/
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
Increment_ Outstanding_ Requests(dp!legID)
storedDP := dp, dpStored := true
Waiting_for_ _Instruction
CallInformation_ ReportPending (true)
ApplyCharging_ ReportPending (true)
PICResume (DP!LegId) to CS
termination :=false
Application_ End(termination,) to CS
Idle

Annex A:

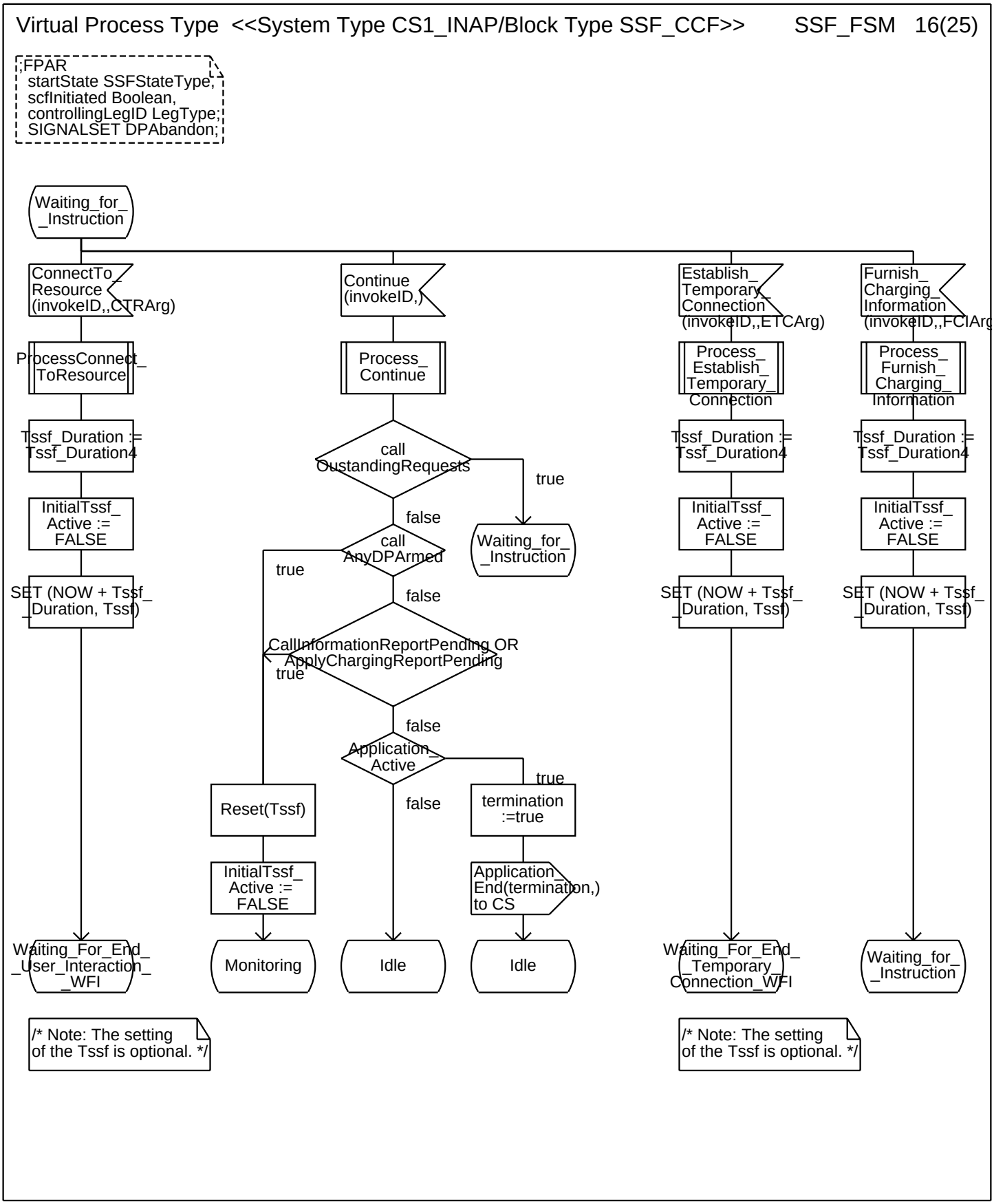
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 14(25)



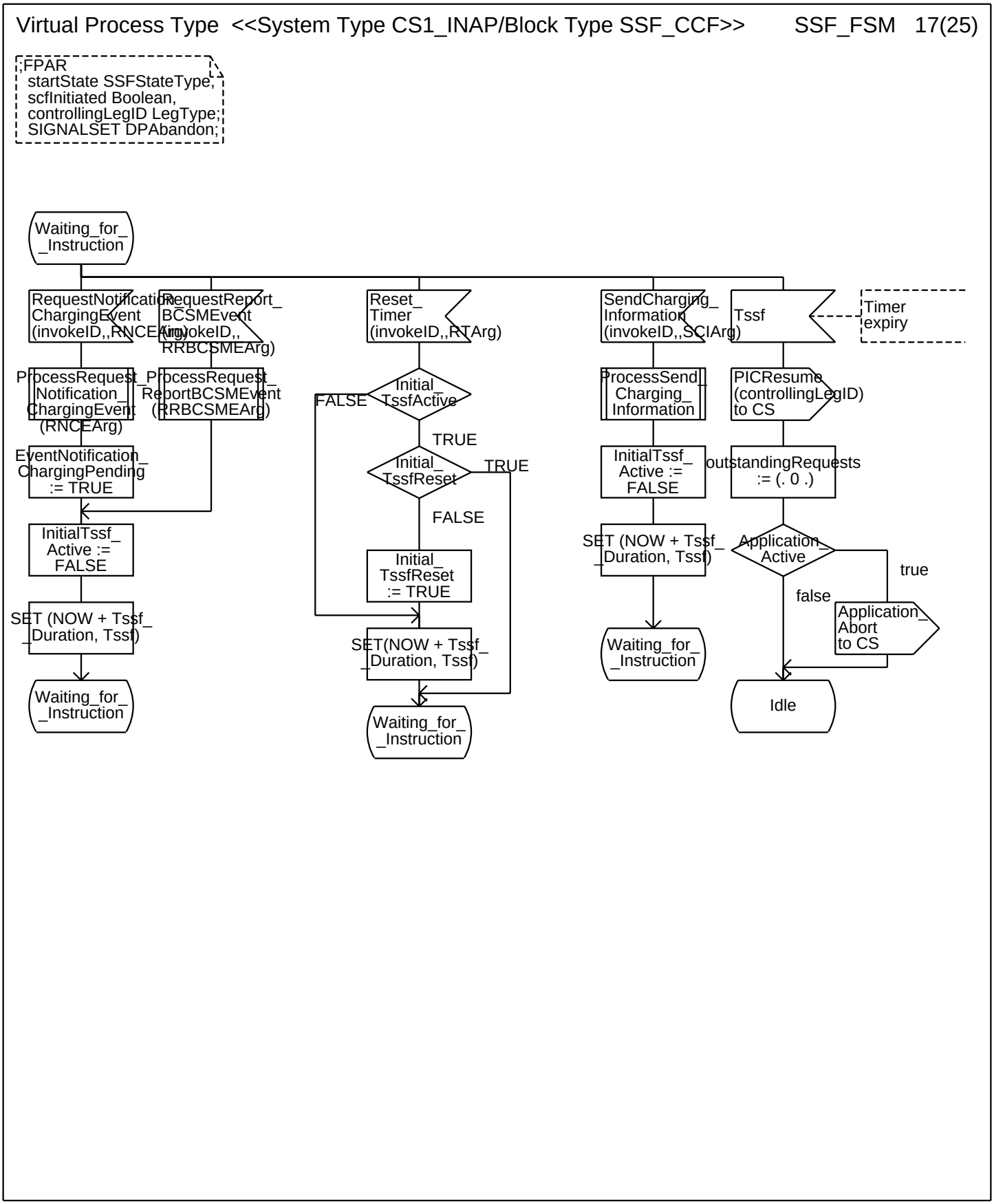
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 15(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/***** WAITING FOR INSTRUCTIONS STATE *****/
/* Note: In Recommendation Q.1218 the p
CollectInformation, Continue, SelectFacility
AnalyseInformation in the WFI state is only
is not connected. For IN CS−2 (Q.1228) th
operations is legal also when the SRF is c
For reasons of simplicity (to avoid redefinit
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
ProcessApply_
Charging
ProcessCall_
InformationRequest
Process_
Cancel
Process_
Collect_
Information
(CIArg)
Process_
Connect
(CoArg)
SEND_
ERROR
ApplyCharging_
ReportPending :=
TRUE
CallInformation_
ReportPending :=
TRUE
outstandingRequests
:= (. 0 .)
InitialTssf_
Active := FALSE
ApplyChargingReport_
Pending := FALSE,
CallInformationReport_
Pending := FALSE,
EventNotificationCharging_
Pending := FALSE
call
AnyDPArmed
true
false
SET (NOW + Tssf_
_Duration, Tssf)
Initialise_
DPTable
CallInformationReportPending OR
ApplyChargingReportPending
true
false
outstandingRequests
:= (. 0 .)
Reset(Tssf)
Application_
Active
TRUE
FALSE
InitialTssf_
Active :=
FALSE
InitialTssf_
Active :=
FALSE
termination
:=true
SET (NOW + Tssf_
_Duration, Tssf)
Application_
End(termination,)
to CS
Waiting_for_
_Instruction
Waiting_for_
_Instruction
Monitoring
Idle
Idle

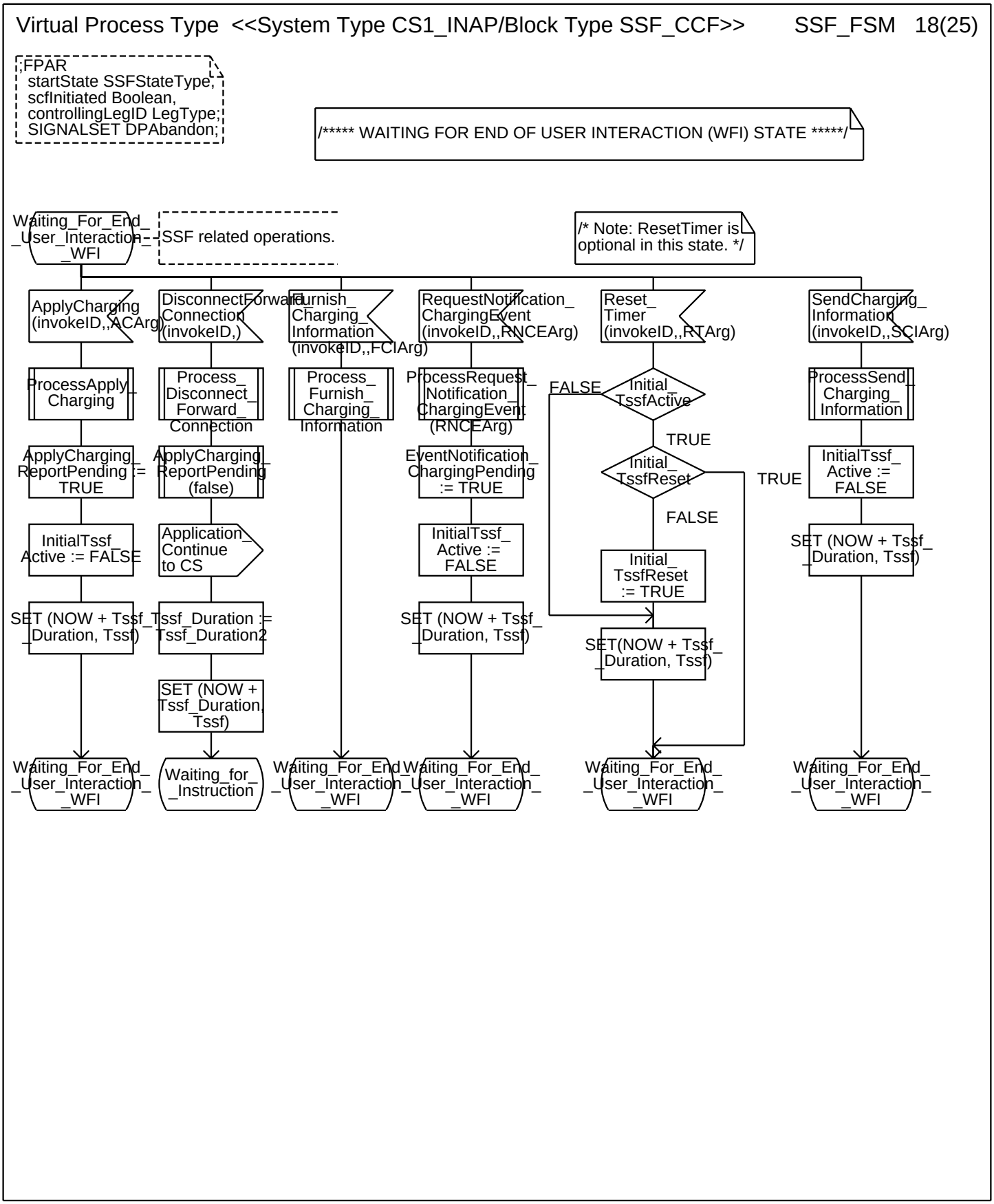
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 16(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
Waiting_for_
_Instruction
ConnectTo_
Resource
(invokeID,,CTRArg)
Continue
(invokeID,)
Establish_
Temporary_
Connection
(invokeID,,ETCArg)
Furnish_
Charging_
Information
(invokeID,,FCIArg
ProcessConnect_
ToResource
Process_
Continue
Process_
Establish_
Temporary_
Connection
Process_
Furnish_
Charging_
Information
Tssf_Duration :=
Tssf_Duration4
call
OustandingRequests
true
false
Waiting_for_
_Instruction
InitialTssf_
Active :=
FALSE
call
AnyDPArmed
true
false
SET (NOW + Tssf_
_Duration, Tssf)
CallInformationReportPending OR
ApplyChargingReportPending
true
false
Application_
Active
true
false
Reset(Tssf)
termination
:=true
Application_
End(termination,)
to CS
Waiting_For_End_
_User_Interaction_
_WFI
Monitoring
Idle
Idle
Waiting_For_End_
Temporary
Connection_WFI
Waiting_for_
_Instruction
/* Note: The setting
of the Tssf is optional. */
/* Note: The setting
of the Tssf is optional. */

Annex A:

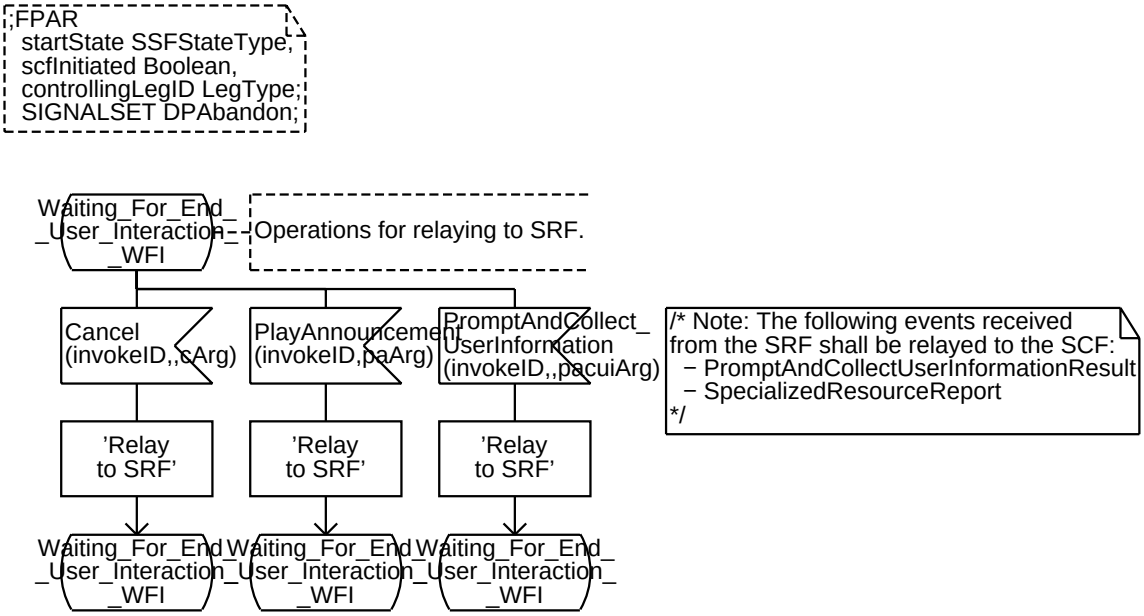

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 17(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
Waiting_for_
_Instruction
RequestNotification_
ChargingEvent
(invokeID,,RNCEArg)
RequestReport_
BCSMEvent
(invokeID,,
RRBCSMEArg)
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
(controllingLegID)
to CS
EventNotification_
ChargingPending
:= TRUE
InitialTssf_
Active :=
FALSE
outstandingRequests
:= (. 0 .)
InitialTssf_
Active :=
FALSE
Initial_
TssfReset
:= TRUE
SET (NOW + Tssf_
_Duration, Tssf)
Application_
Active
true
false
Application_
Abort
to CS
SET (NOW + Tssf_
_Duration, Tssf)
SET(NOW + Tssf_
_Duration, Tssf)
Waiting_for_
_Instruction
Waiting_for_
_Instruction
Waiting_for_
_Instruction
Idle

Annex A:

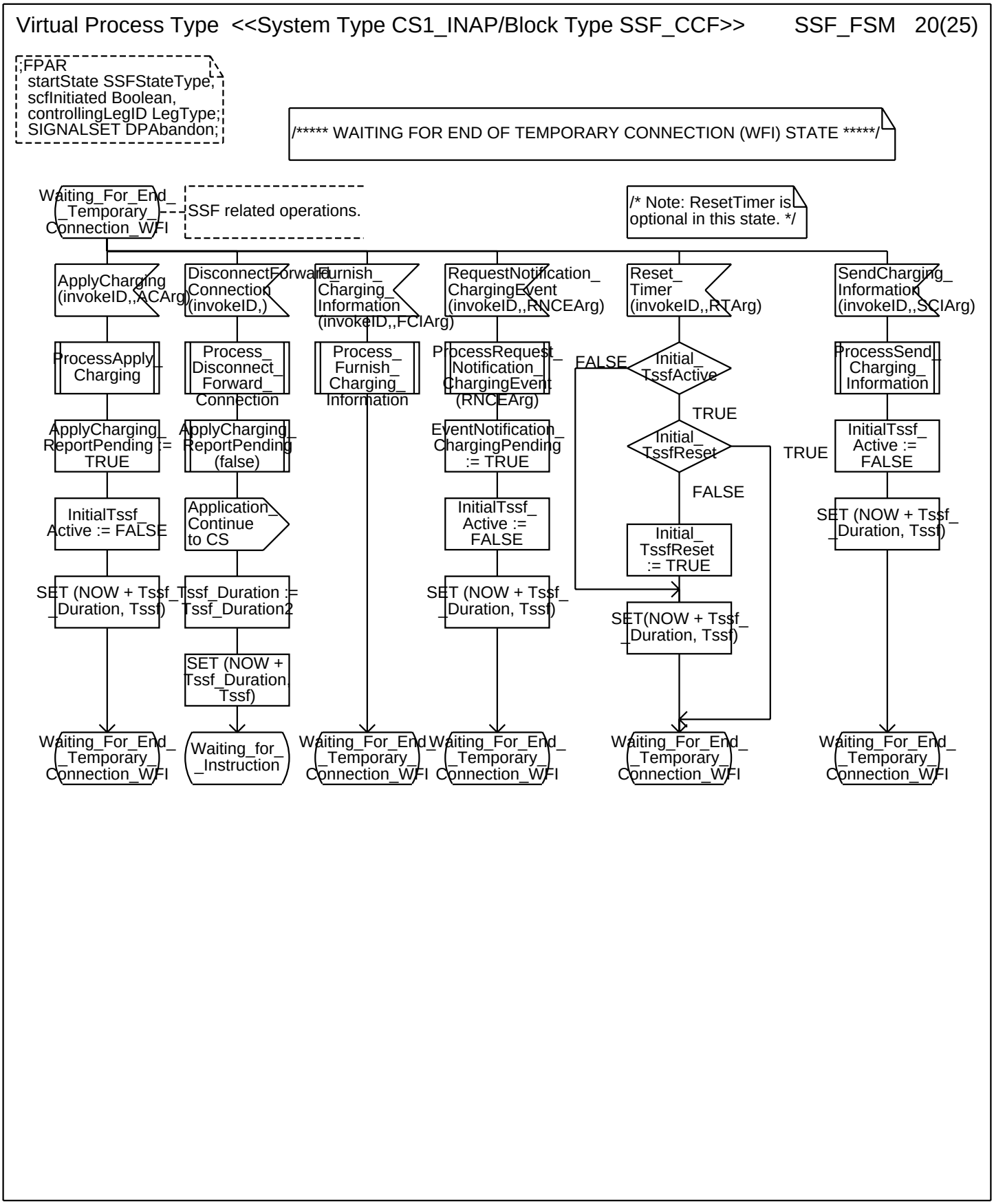

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 18(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/***** WAITING FOR END OF USER INTERACTION (WFI) STATE *****/
Waiting_For_End_User_Interaction_WFI
SSF related operations.
/* Note: ResetTimer is optional in this state. */
ApplyCharging (invokeID,,ACArg)
DisconnectForwardConnection (invokeID,)
Furnish_Charging_Information (invokeID,,FCIArg)
RequestNotification_ChargingEvent (invokeID,,RNCEArg)
Reset_Timer (invokeID,,RTArg)
SendCharging_Information (invokeID,,SCIArg)
ProcessApply_Charging
Process_Disconnect_Forward_Connection
Process_Furnish_Charging_Information
ProcessRequest_Notification_ChargingEvent (RNCEArg)
FALSE
Initial_TssfActive
TRUE
ProcessSend_Charging_Information
ApplyCharging_ReportPending := TRUE
ApplyCharging_ReportPending (false)
EventNotification_ChargingPending := TRUE
Initial_TssfReset
TRUE
InitialTssf_Active := FALSE
FALSE
InitialTssf_Active := FALSE
Application_Continue to CS
InitialTssf_Active := FALSE
Initial_TssfReset := TRUE
SET (NOW + Tssf_Duration, Tssf)
SET (NOW + Tssf_Duration, Tssf)
Tssf_Duration := Tssf_Duration2
SET (NOW + Tssf_Duration, Tssf)
SET(NOW + Tssf_Duration, Tssf)
SET (NOW + Tssf_Duration, Tssf)
Waiting_For_End_User_Interaction_WFI
Waiting_for_Instruction
Waiting_For_End_User_Interaction_WFI
Waiting_For_End_User_Interaction_WFI
Waiting_For_End_User_Interaction_WFI
Waiting_For_End_User_Interaction_WFI

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 19(25)

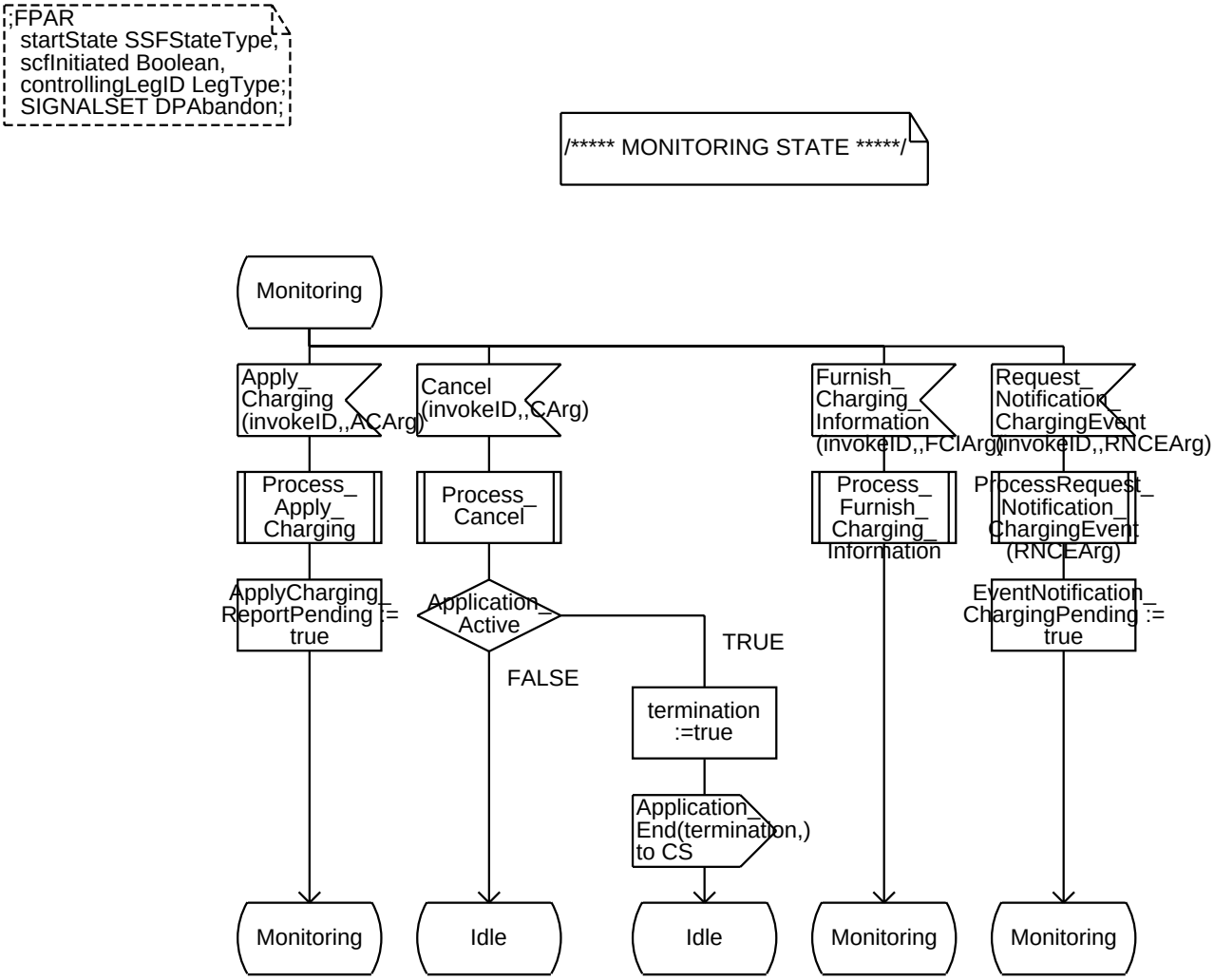


Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 20(25)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/***** WAITING FOR END OF TEMPORARY CONNECTION (WFI) STATE *****/
Waiting_For_End_
Temporary
Connection_WFI
SSF related operations.
/* Note: ResetTimer is
optional in this state. */
ApplyCharging
(invokeID,,ACArg)
DisconnectForwardConnection
(invokeID,)
Furnish_
Charging_
Information
(invokeID,,FCIArg)
RequestNotification_
ChargingEvent
(invokeID,,RNCEArg)
Reset_
Timer
(invokeID,,RTArg)
SendCharging_
Information
(invokeID,,SCIArg)
ProcessApply_
Charging
Process_
Disconnect_
Forward_
Connection
Process_
Furnish_
Charging_
Information
ProcessRequest_
Notification_
ChargingEvent
(RNCEArg)
FALSE
Initial_
TssfActive
ProcessSend_
Charging_
Information
TRUE
ApplyCharging_
ReportPending :=
TRUE
ApplyCharging_
ReportPending
(false)
EventNotification_
ChargingPending
:= TRUE
Initial_
TssfReset
TRUE
InitialTssf_
Active :=
FALSE
FALSE
InitialTssf_
Active := FALSE
Application_
Continue
to CS
InitialTssf_
Active :=
FALSE
Initial_
TssfReset
:= TRUE
SET (NOW + Tssf_
_Duration, Tssf)
SET (NOW + Tssf_
_Duration, Tssf)
Tssf_Duration :=
Tssf_Duration2
SET (NOW + Tssf_
_Duration, Tssf)
SET(NOW + Tssf_
_Duration, Tssf)
SET (NOW +
Tssf_Duration,
Tssf)
Waiting_For_End_
Temporary
Connection_WFI
Waiting_for_
_Instruction
Waiting_For_End_
Temporary
Connection_WFI
Waiting_For_End_
Temporary
Connection_WFI
Waiting_For_End_
Temporary
Connection_WFI
Waiting_For_End_
Temporary
Connection_WFI

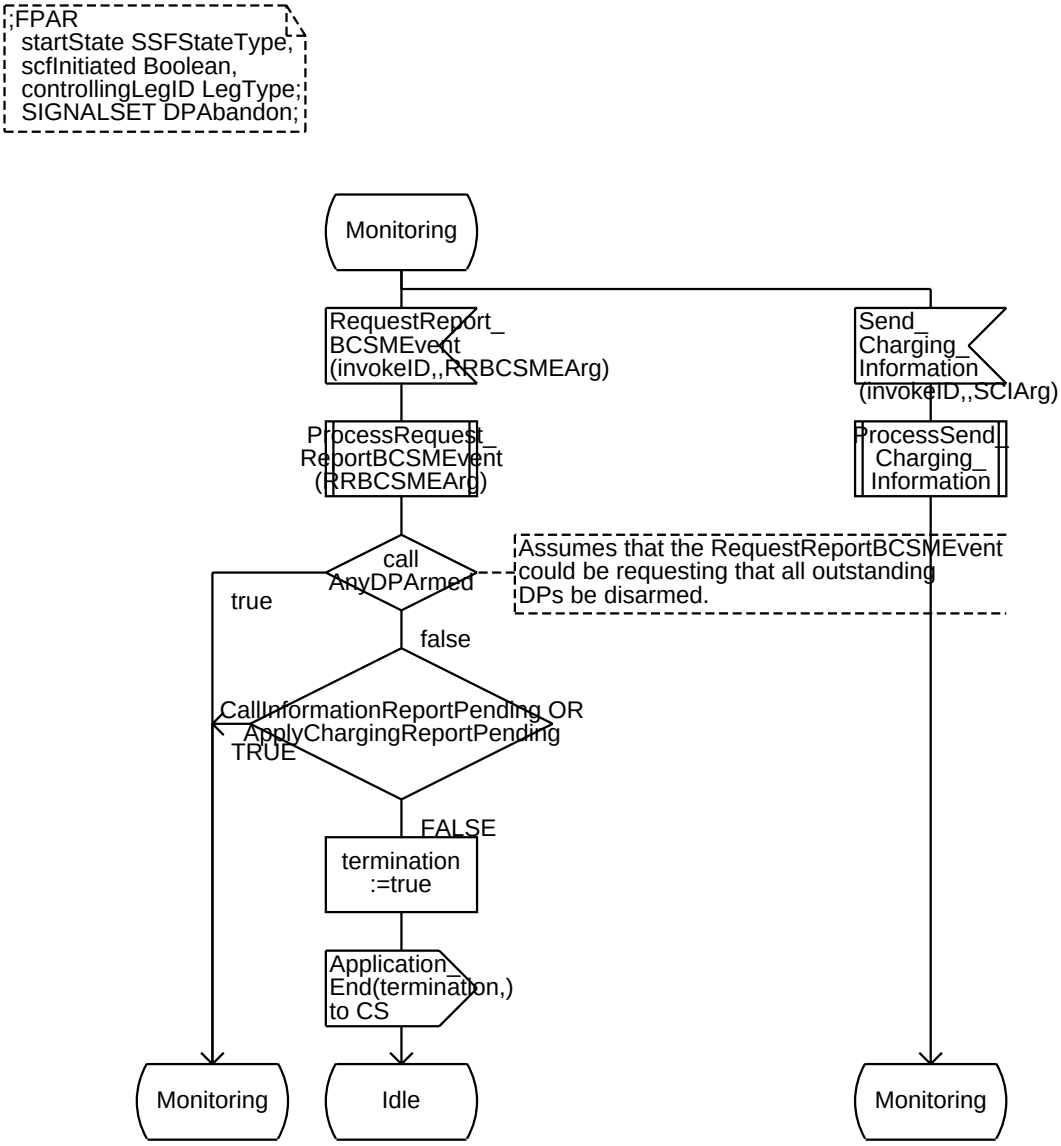
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 21(25)

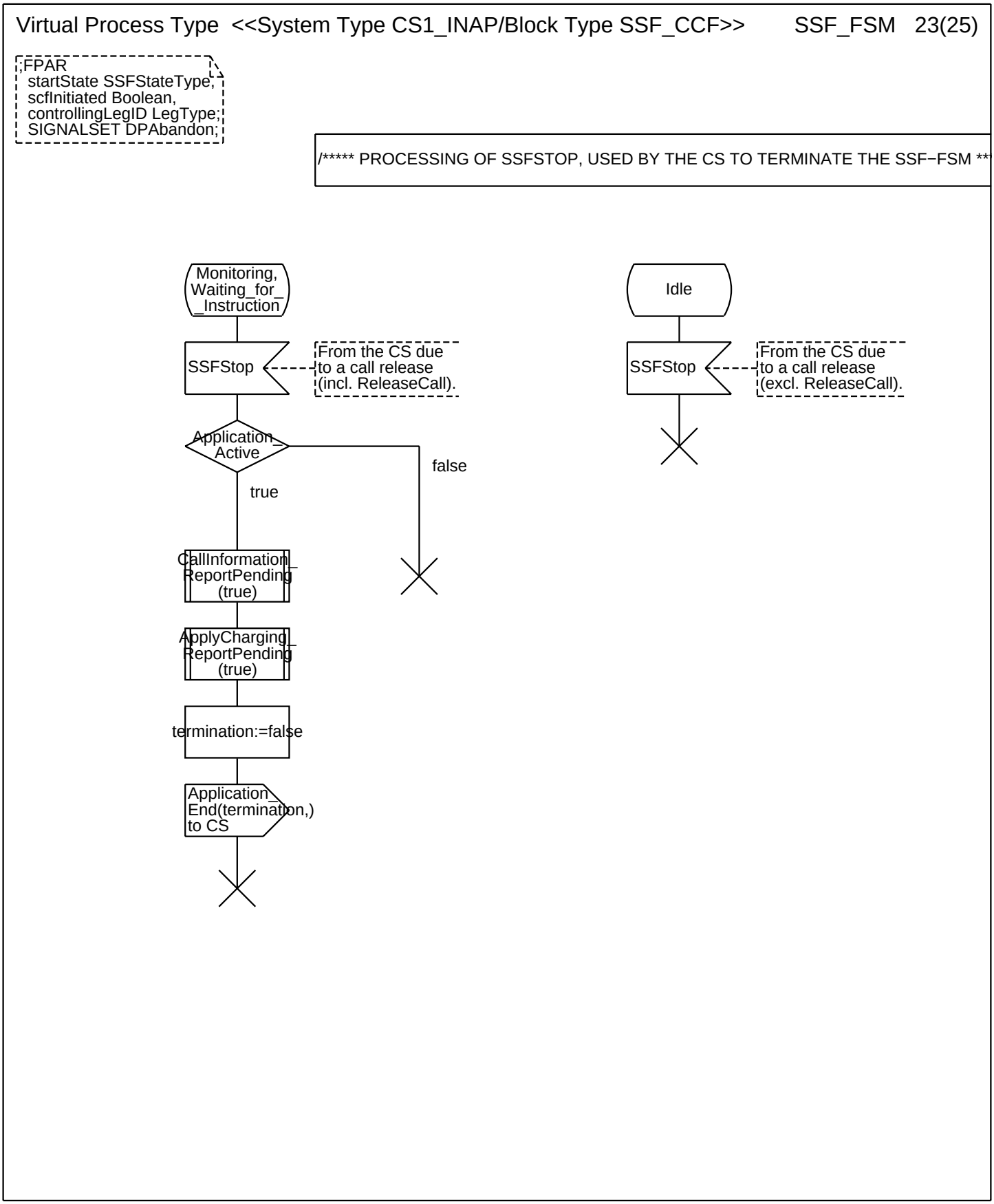


Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 22(25)

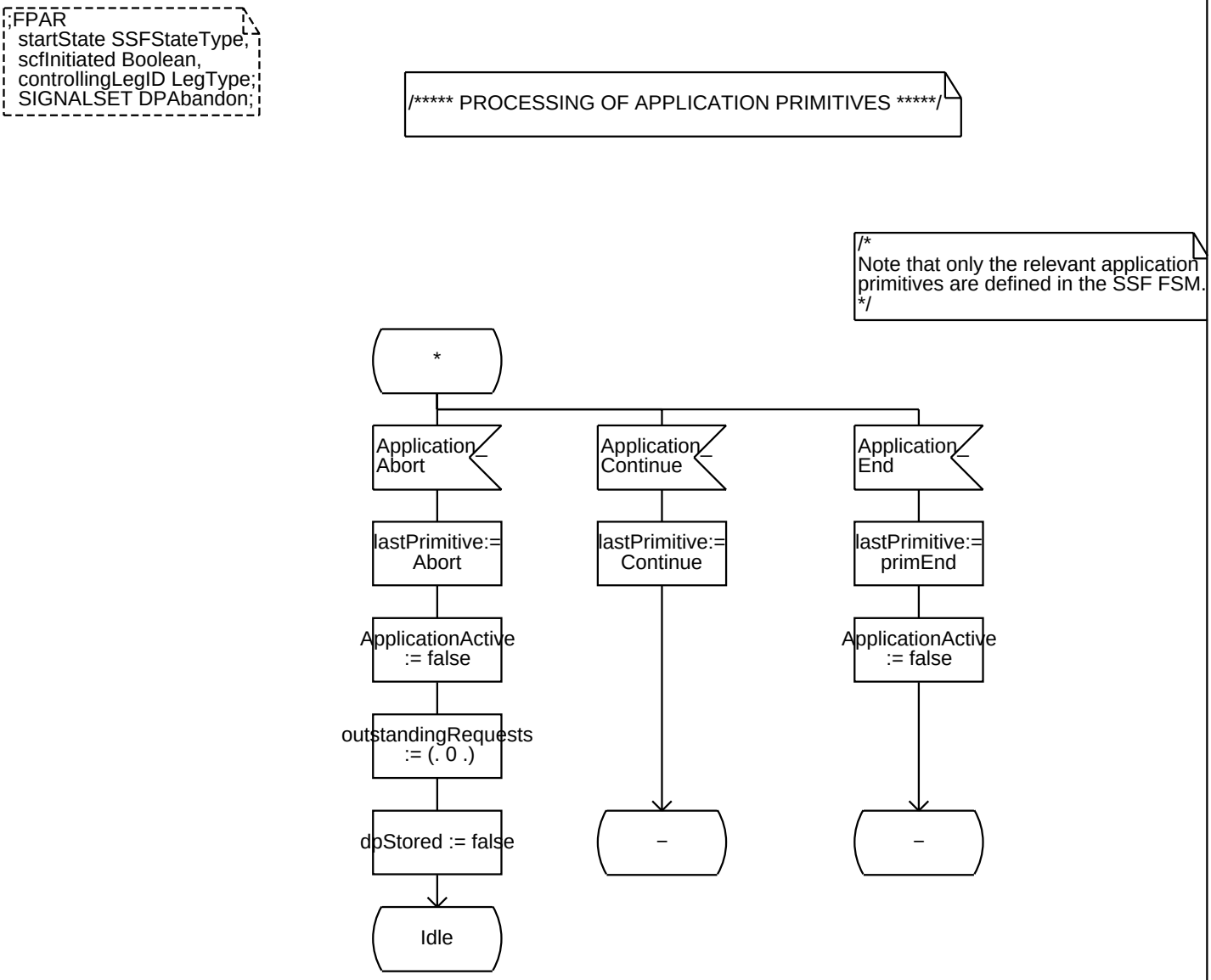


Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 23(25)
;FPAR
startState SSFStateType;
scfInitiated Boolean,
controllingLegID LegType;
SIGNALSET DPAbandon;
/***** 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).

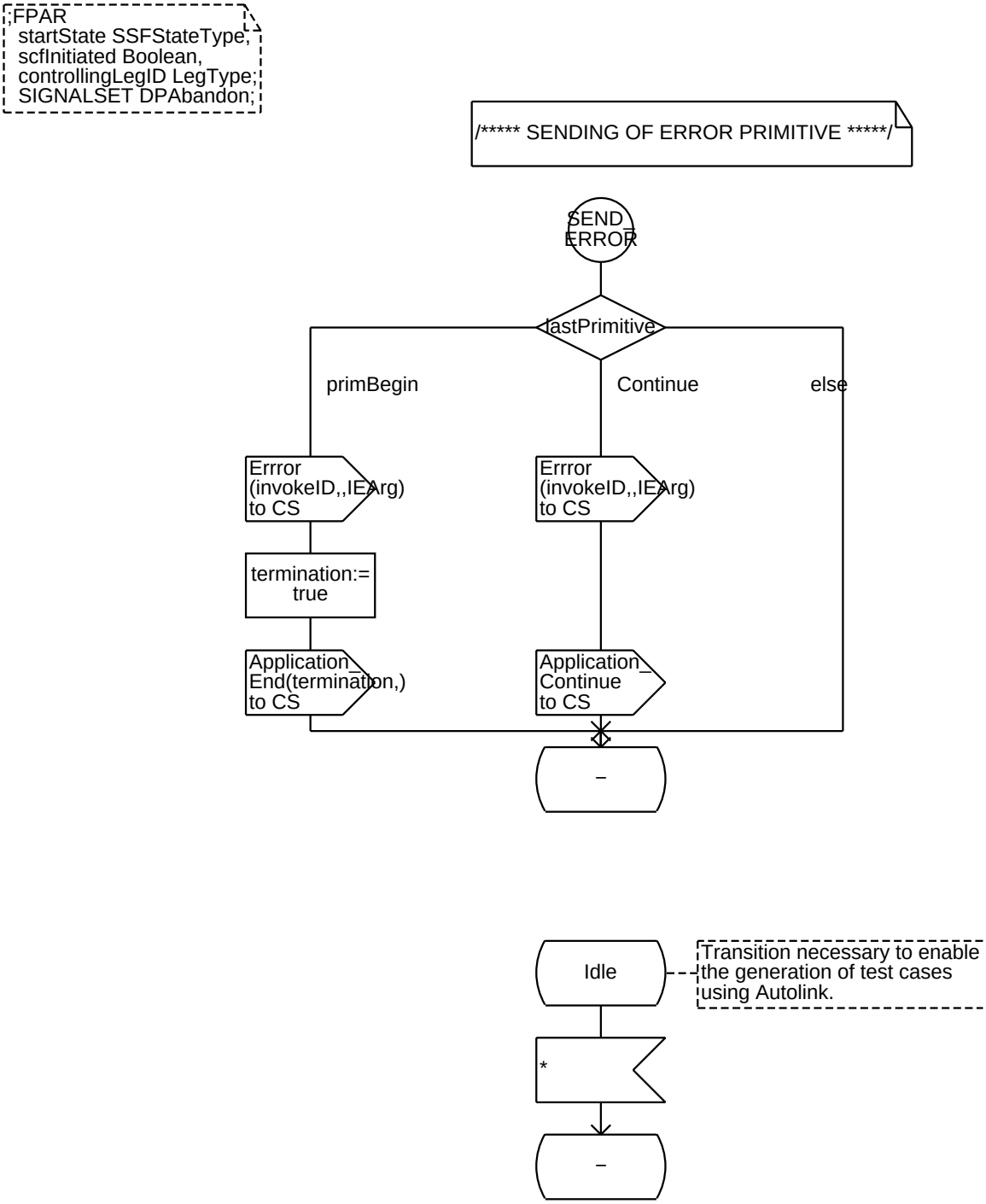
Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 24(25)

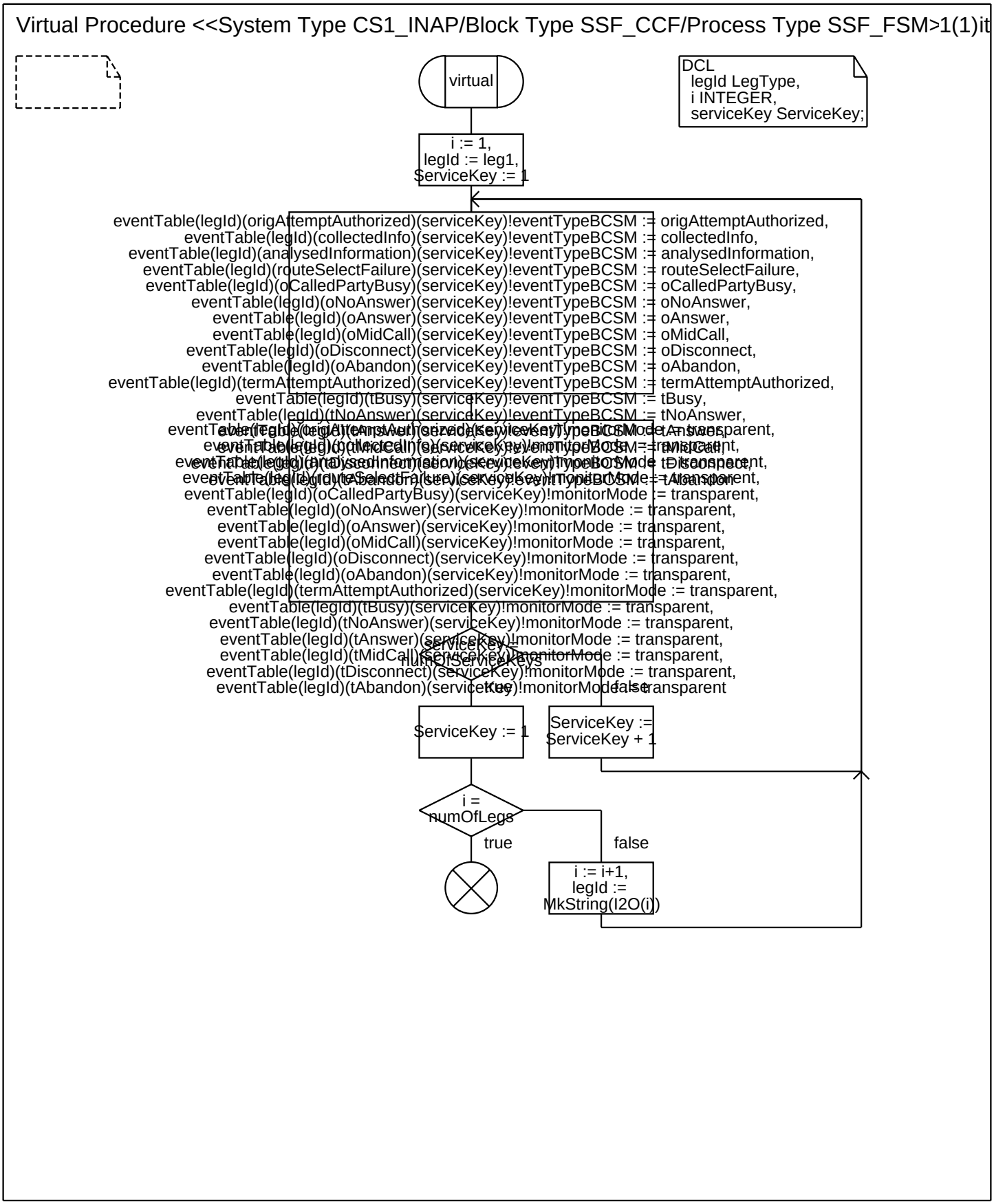


Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> SSF_FSM 25(25)

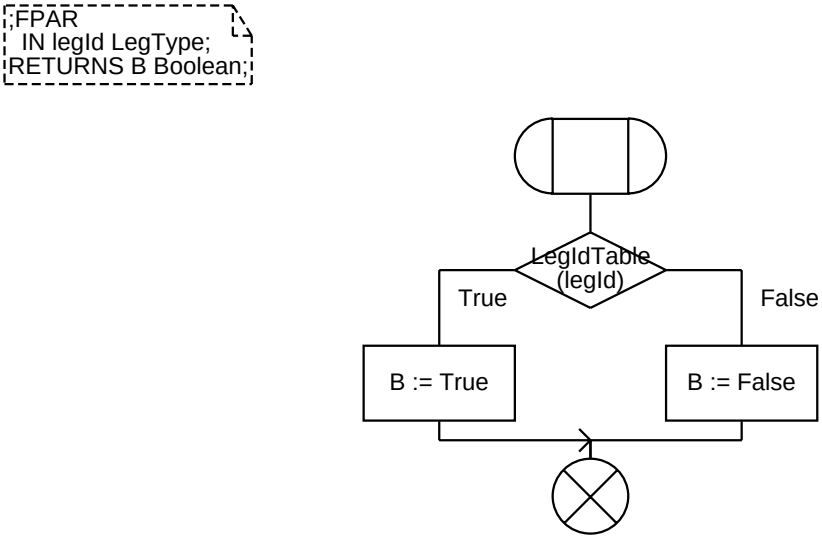


Annex A:

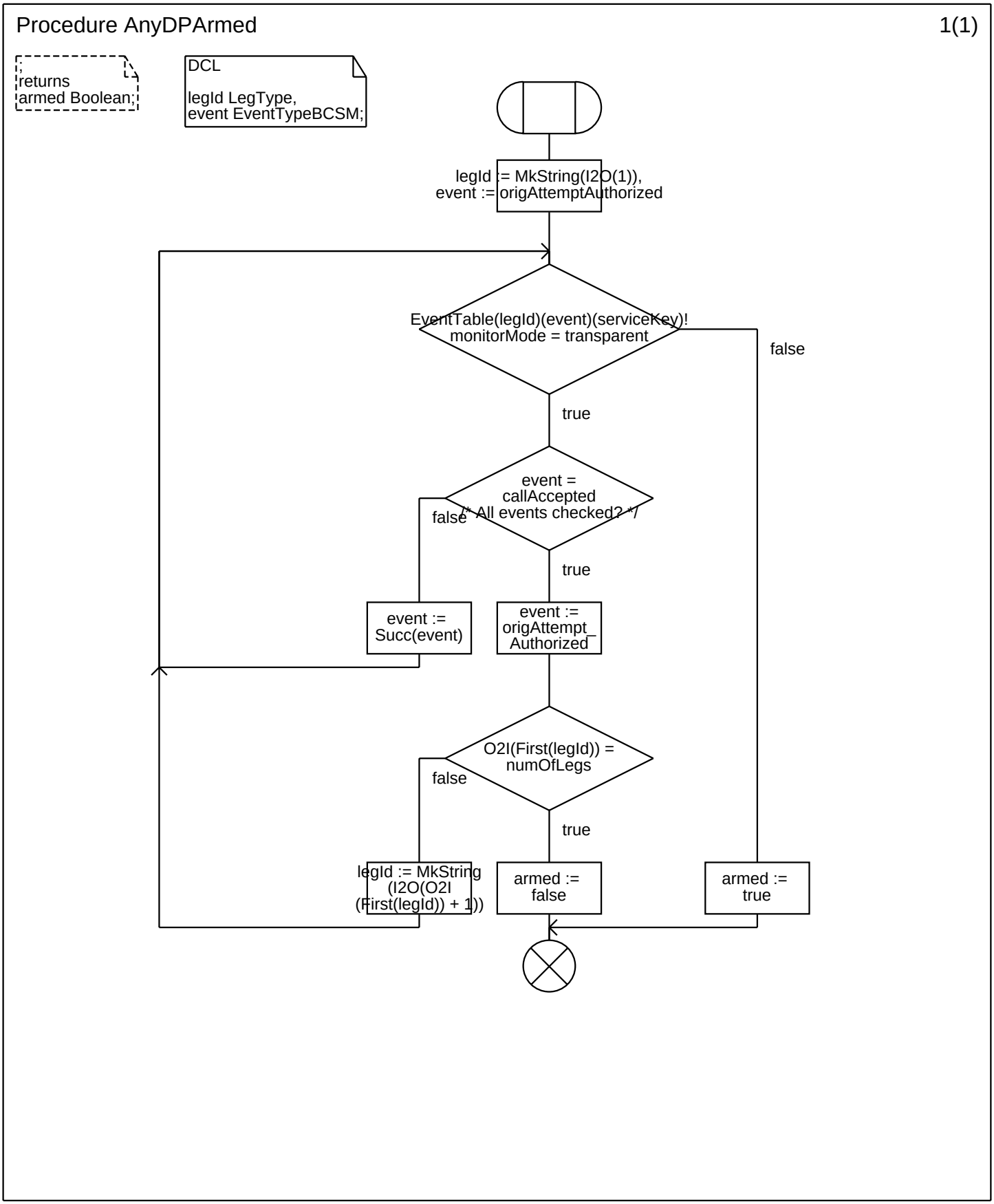

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type SSF_FSM>1(1)it
virtual
DCL
legId LegType,
i INTEGER,
serviceKey ServiceKey;
i := 1,
legId := leg1,
ServiceKey := 1
eventTable(legId)(origAttemptAuthorized)(serviceKey)!eventTypeBCSM := origAttemptAuthorized,
eventTable(legId)(collectedInfo)(serviceKey)!eventTypeBCSM := collectedInfo,
eventTable(legId)(analysedInformation)(serviceKey)!eventTypeBCSM := analysedInformation,
eventTable(legId)(routeSelectFailure)(serviceKey)!eventTypeBCSM := routeSelectFailure,
eventTable(legId)(oCalledPartyBusy)(serviceKey)!eventTypeBCSM := oCalledPartyBusy,
eventTable(legId)(oNoAnswer)(serviceKey)!eventTypeBCSM := oNoAnswer,
eventTable(legId)(oAnswer)(serviceKey)!eventTypeBCSM := oAnswer,
eventTable(legId)(oMidCall)(serviceKey)!eventTypeBCSM := oMidCall,
eventTable(legId)(oDisconnect)(serviceKey)!eventTypeBCSM := oDisconnect,
eventTable(legId)(oAbandon)(serviceKey)!eventTypeBCSM := oAbandon,
eventTable(legId)(termAttemptAuthorized)(serviceKey)!eventTypeBCSM := termAttemptAuthorized,
eventTable(legId)(tBusy)(serviceKey)!eventTypeBCSM := tBusy,
eventTable(legId)(tNoAnswer)(serviceKey)!eventTypeBCSM := tNoAnswer,
eventTable(legId)(tAnswer)(serviceKey)!eventTypeBCSM := tAnswer,
eventTable(legId)(tMidCall)(serviceKey)!eventTypeBCSM := tMidCall,
eventTable(legId)(tDisconnect)(serviceKey)!eventTypeBCSM := tDisconnect,
eventTable(legId)(tAbandon)(serviceKey)!eventTypeBCSM := tAbandon
eventTable(legId)(origAttemptAuthorized)(serviceKey)!monitorMode := transparent,
eventTable(legId)(collectedInfo)(serviceKey)!monitorMode := transparent,
eventTable(legId)(analysedInformation)(serviceKey)!monitorMode := transparent,
eventTable(legId)(routeSelectFailure)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oCalledPartyBusy)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oNoAnswer)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oAnswer)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oMidCall)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oDisconnect)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oAbandon)(serviceKey)!monitorMode := transparent,
eventTable(legId)(termAttemptAuthorized)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tBusy)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tNoAnswer)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tAnswer)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tMidCall)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tDisconnect)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tAbandon)(serviceKey)!monitorMode := transparent
ServiceKey = numOfServiceKeys
false
ServiceKey := 1
ServiceKey := ServiceKey + 1
i = numOfLegs
true
false
i := i+1,
legId := MkString(I2O(i))

Annex A:

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

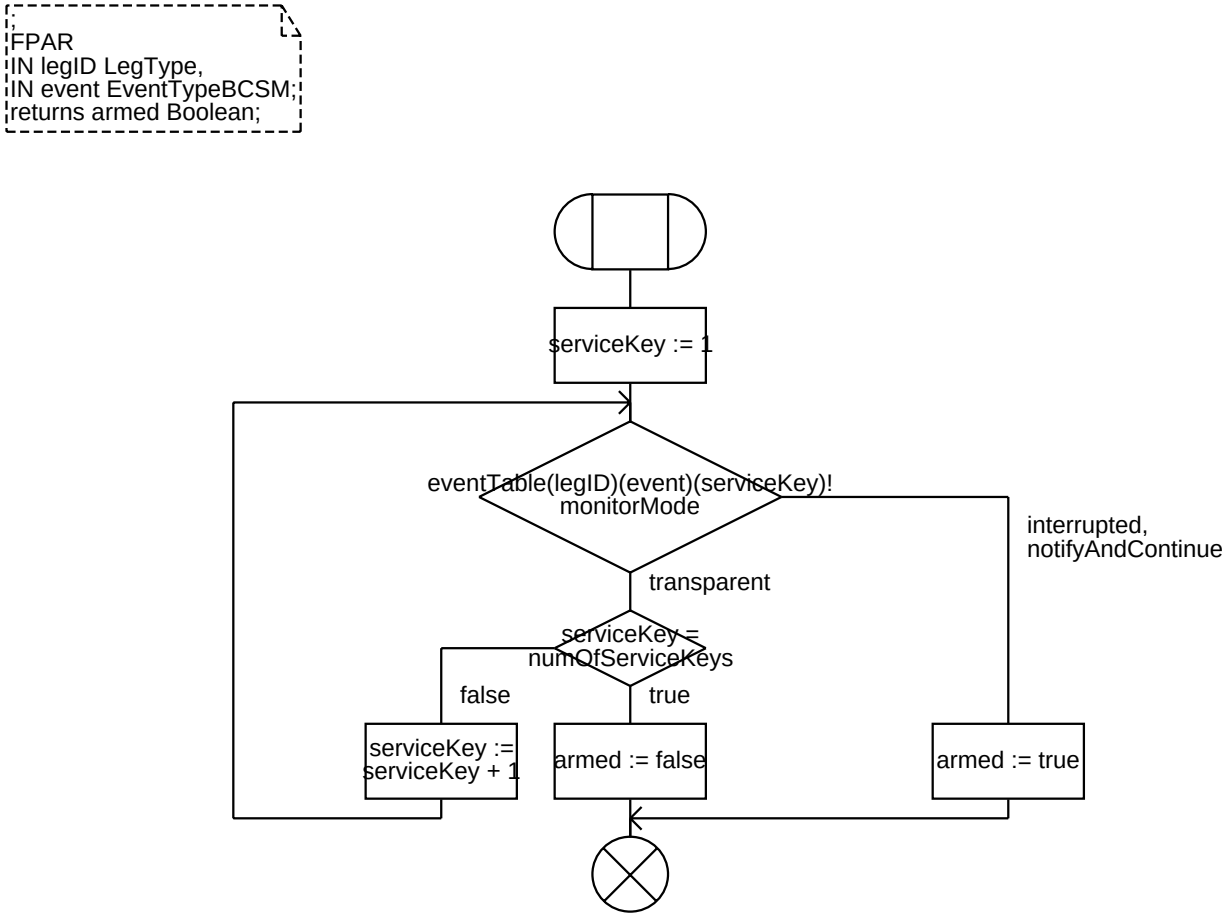


Annex A:


Procedure AnyDPArmed
1(1)
;
returns
armed Boolean;
DCL
legId LegType,
event EventTypeBCSM;
legId := MkString(I2O(1)),
event := origAttemptAuthorized
EventTable(legId)(event)(serviceKey)!
monitorMode = transparent
false
true
event =
callAccepted
/* All events checked? */
false
true
event :=
Succ(event)
event :=
origAttempt_
Authorized
O2I(First(legId)) =
numOfLegs
false
true
legId := MkString
(I2O(O2I
(First(legId)) + 1))
armed :=
false
armed :=
true

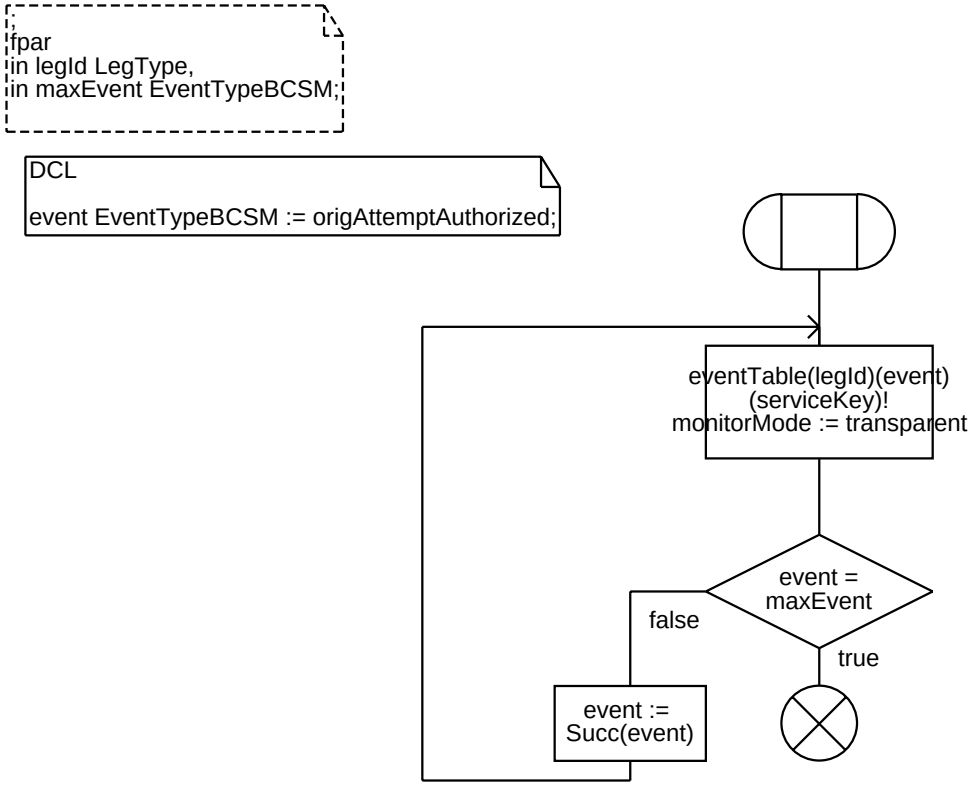
Annex A:

Procedure IsDPArmed 1(1)



Annex A:

Procedure DisarmDPs 1(1)

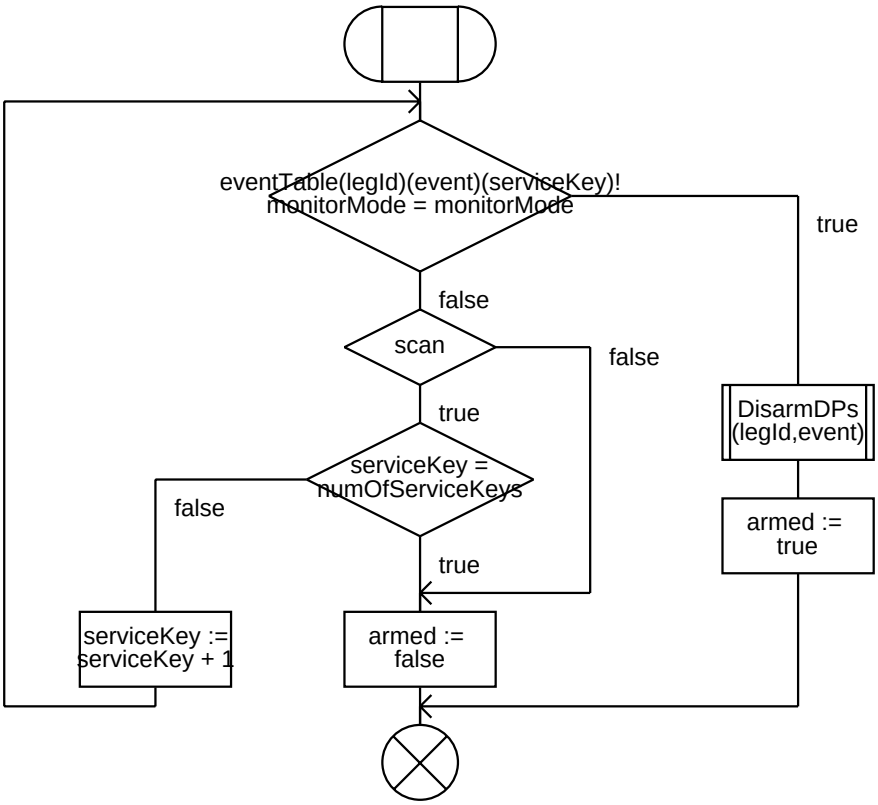


Annex A:

Procedure DPArmed 1(1)

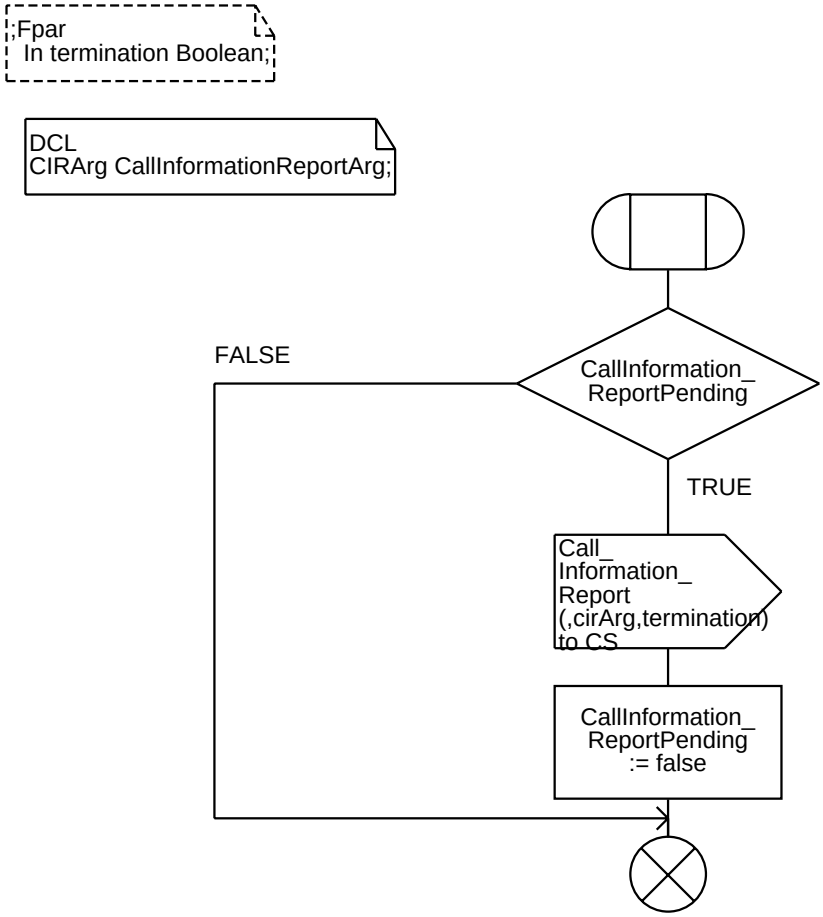
```
;
fpar
in legId LegType,
in event EventTypeBCSM,
in monitorMode MonitorMode,
in scan Boolean;
returns armed Boolean
;
```

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



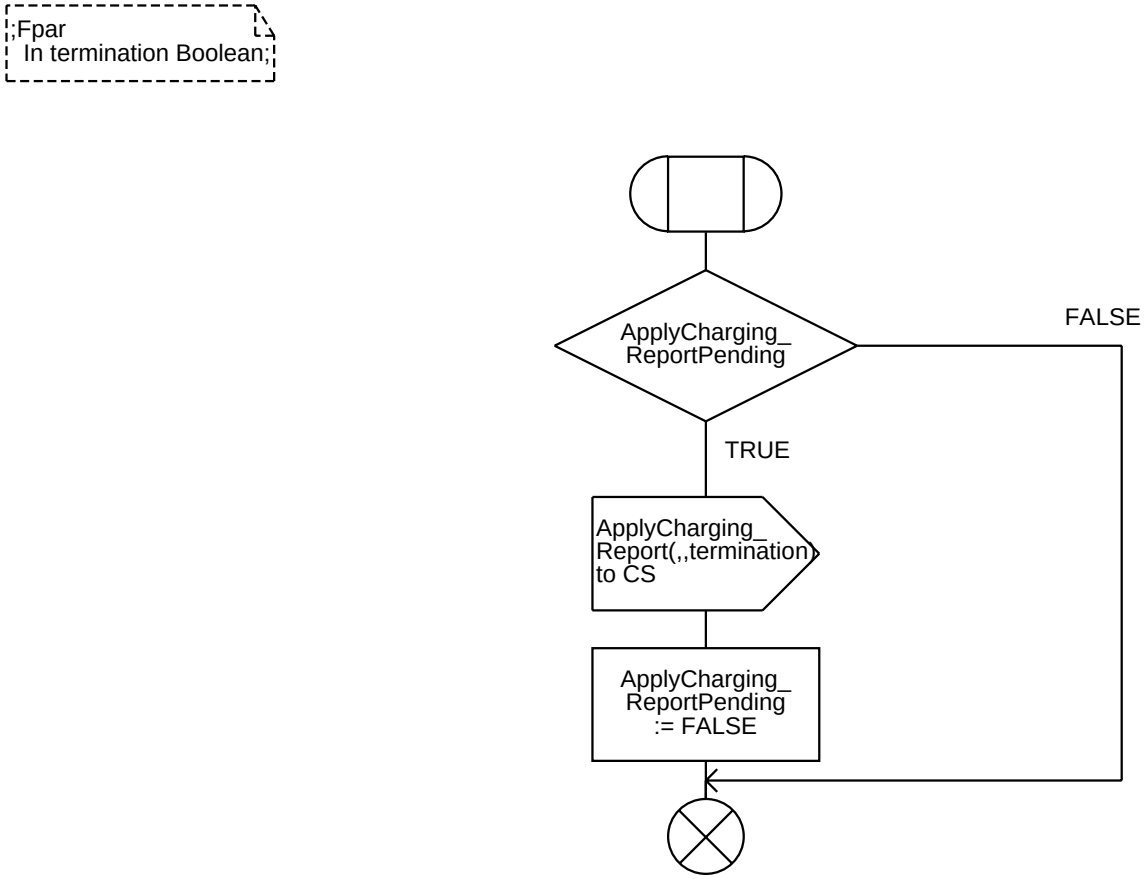
Annex A:

Procedure CallInformationReportPending 1(1)



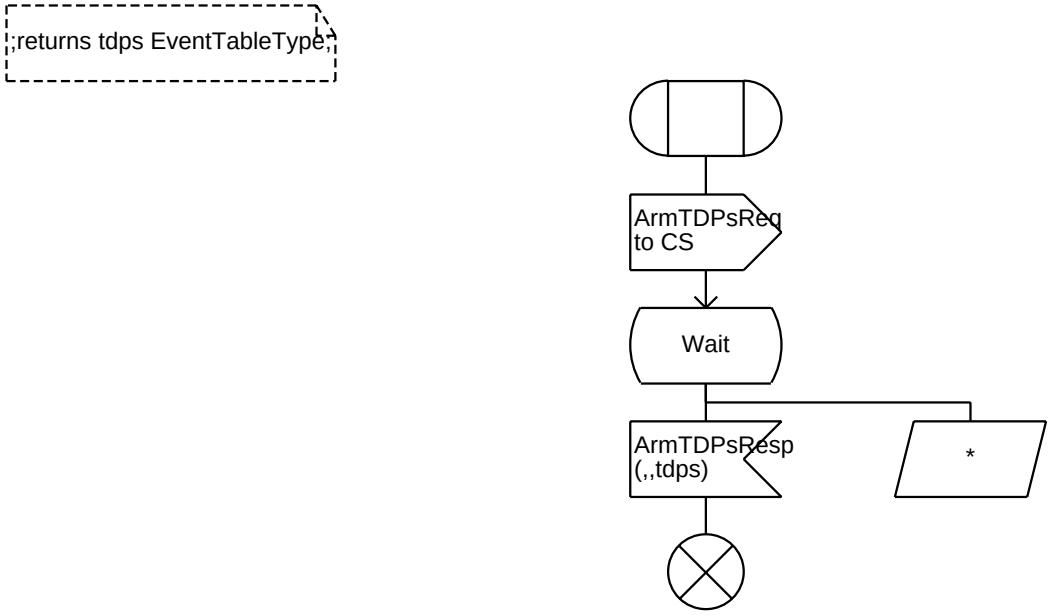
Annex A:

Procedure ApplyChargingReportPending 1(1)



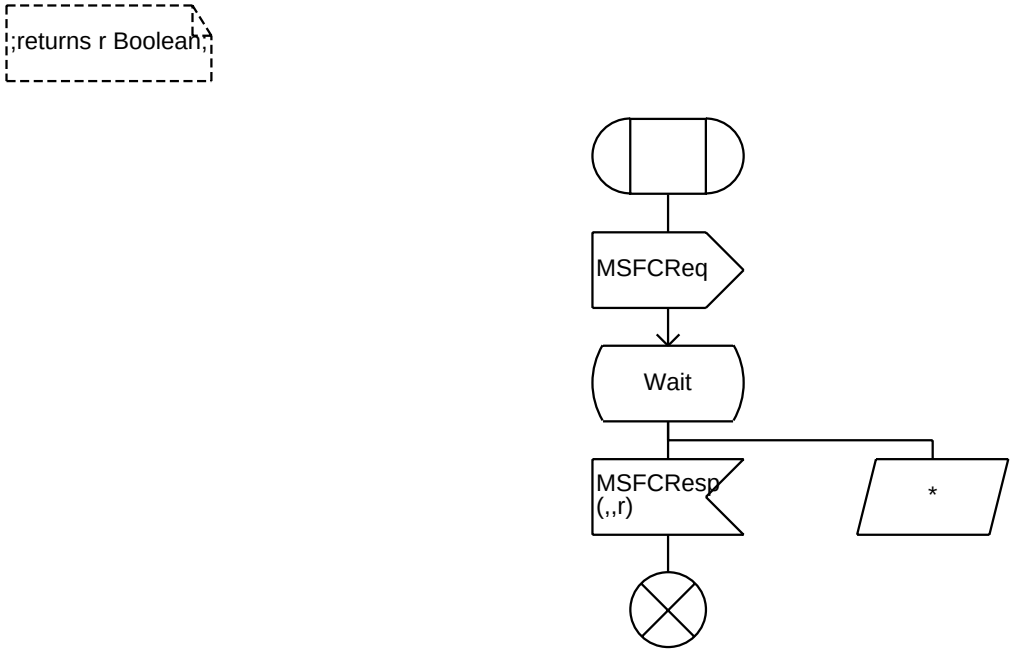
Annex A:

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



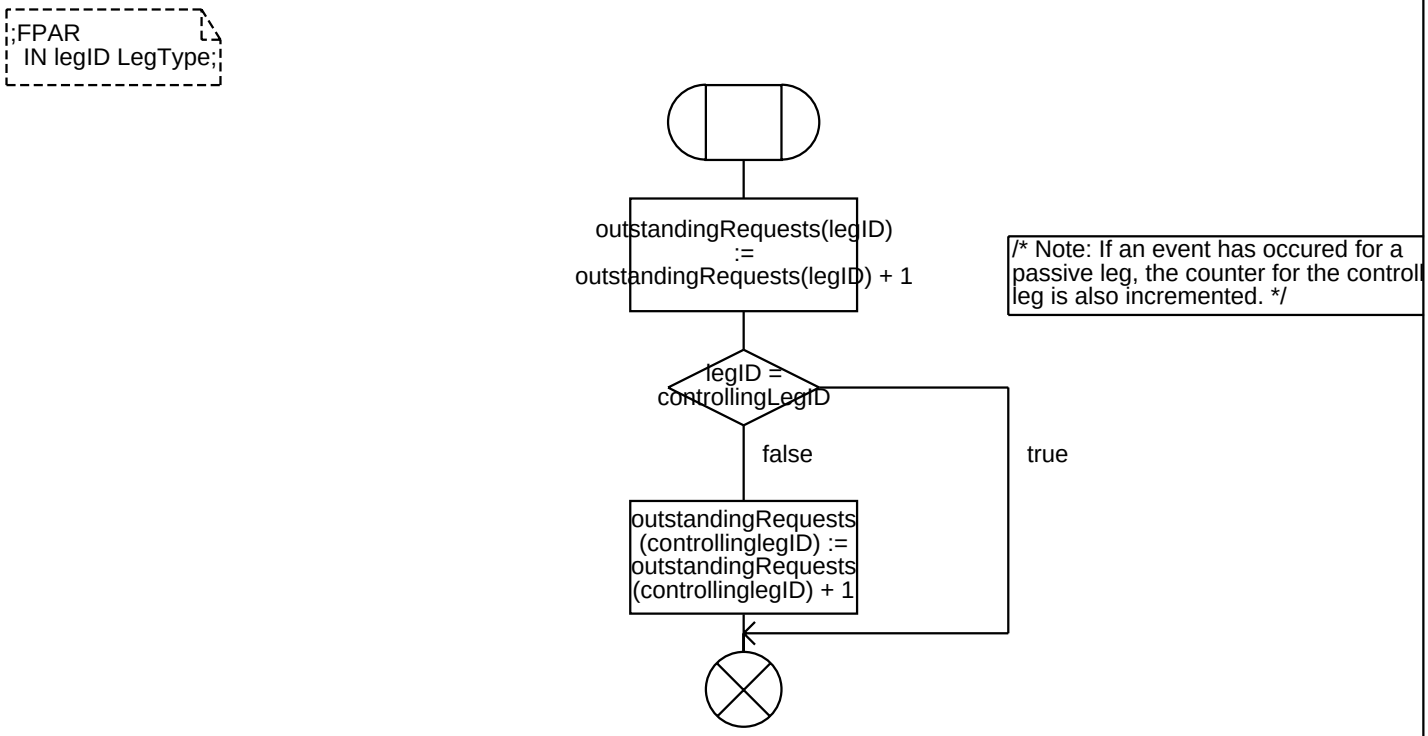
Annex A:

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



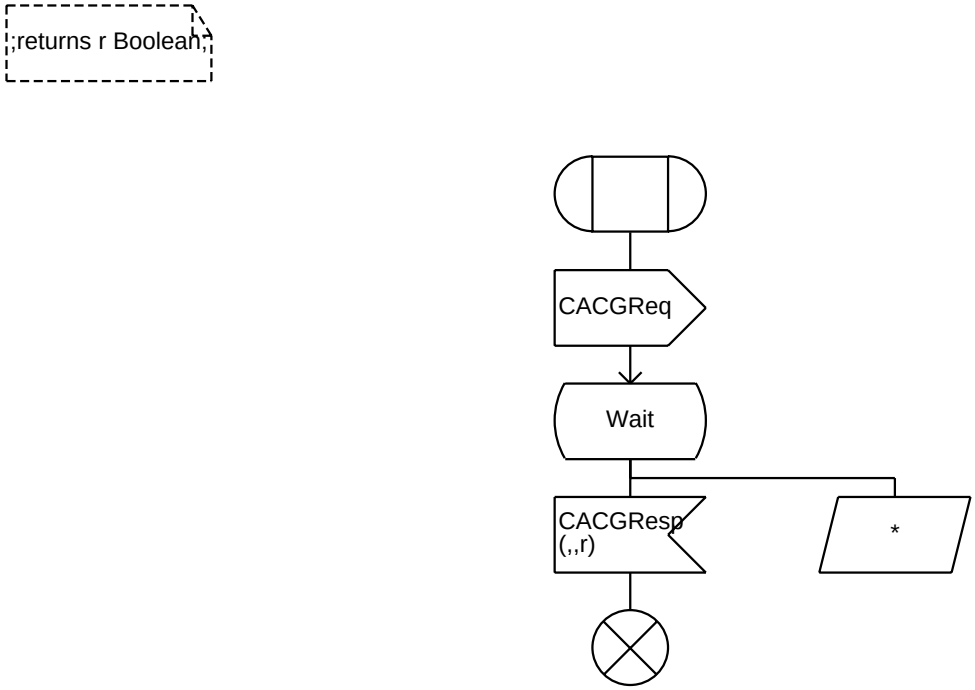
Annex A:

Procedure IncrementOutstandingRequests 1(1)



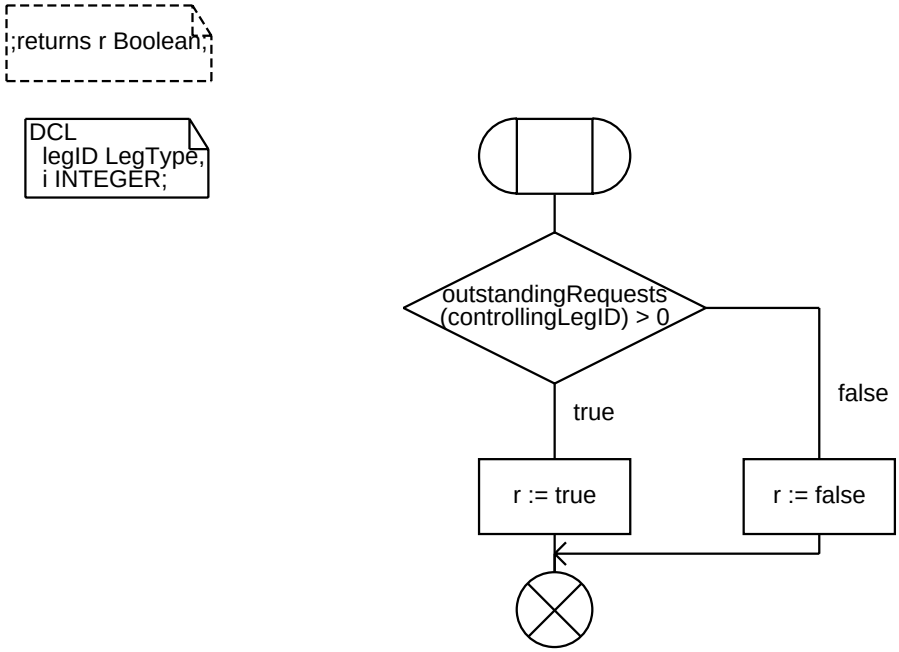
Annex A:

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



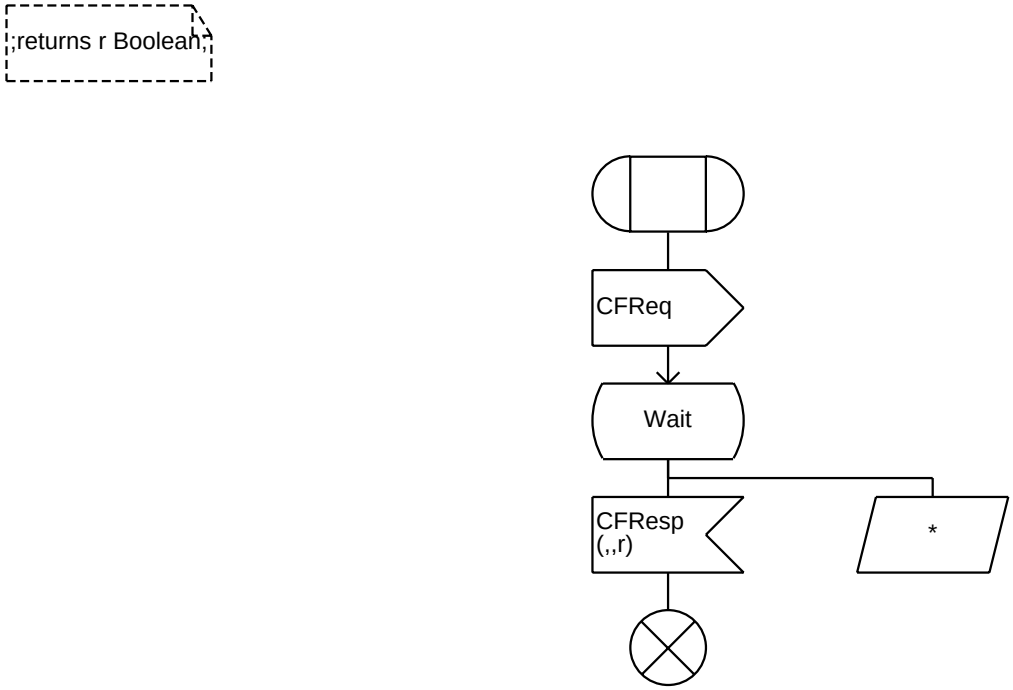
Annex A:

Procedure OustandingRequests 1(1)



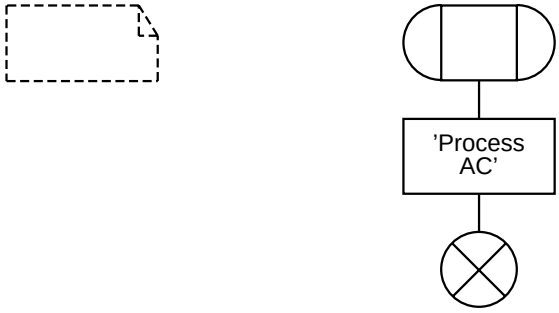
Annex A:

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

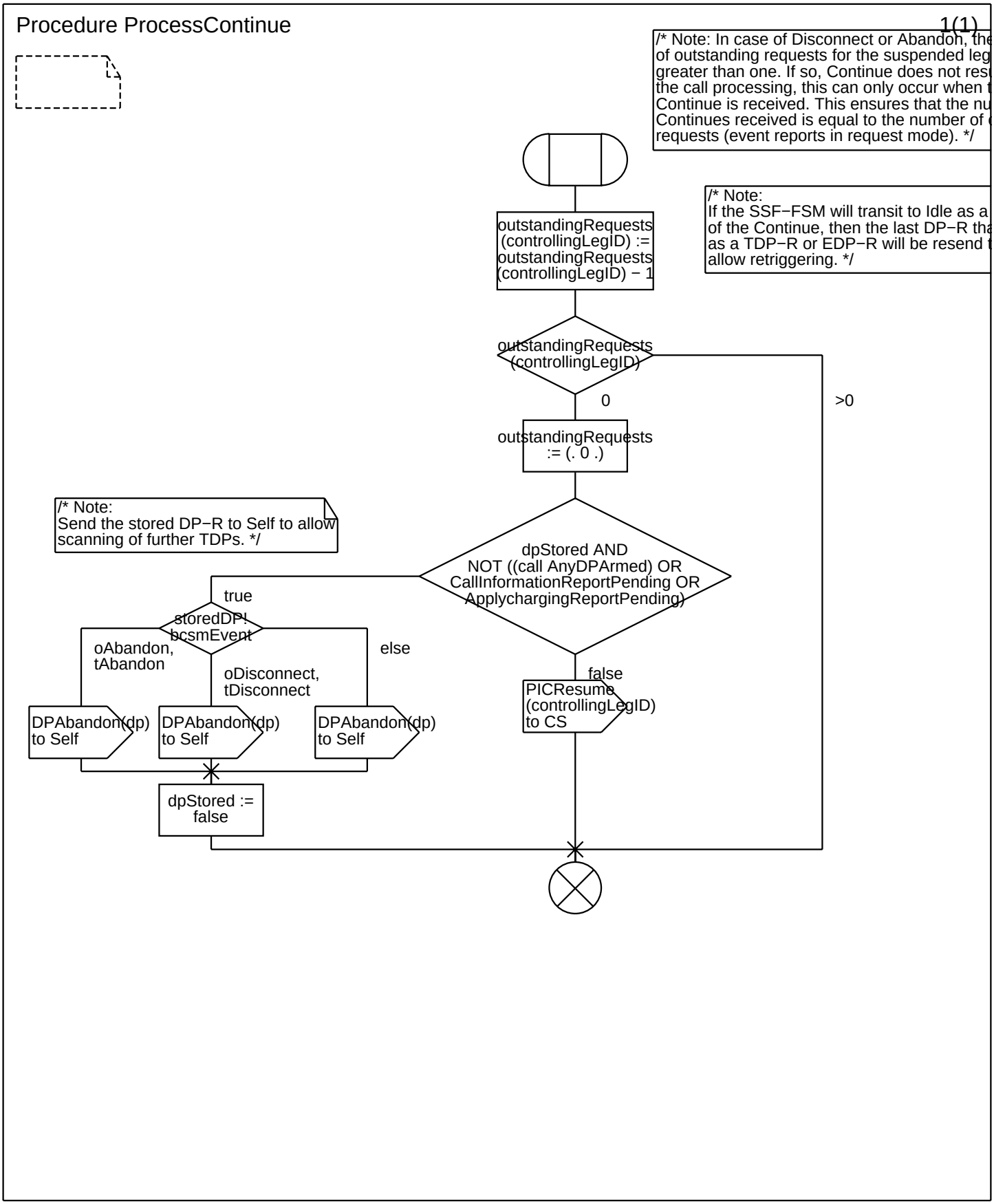


Annex A:

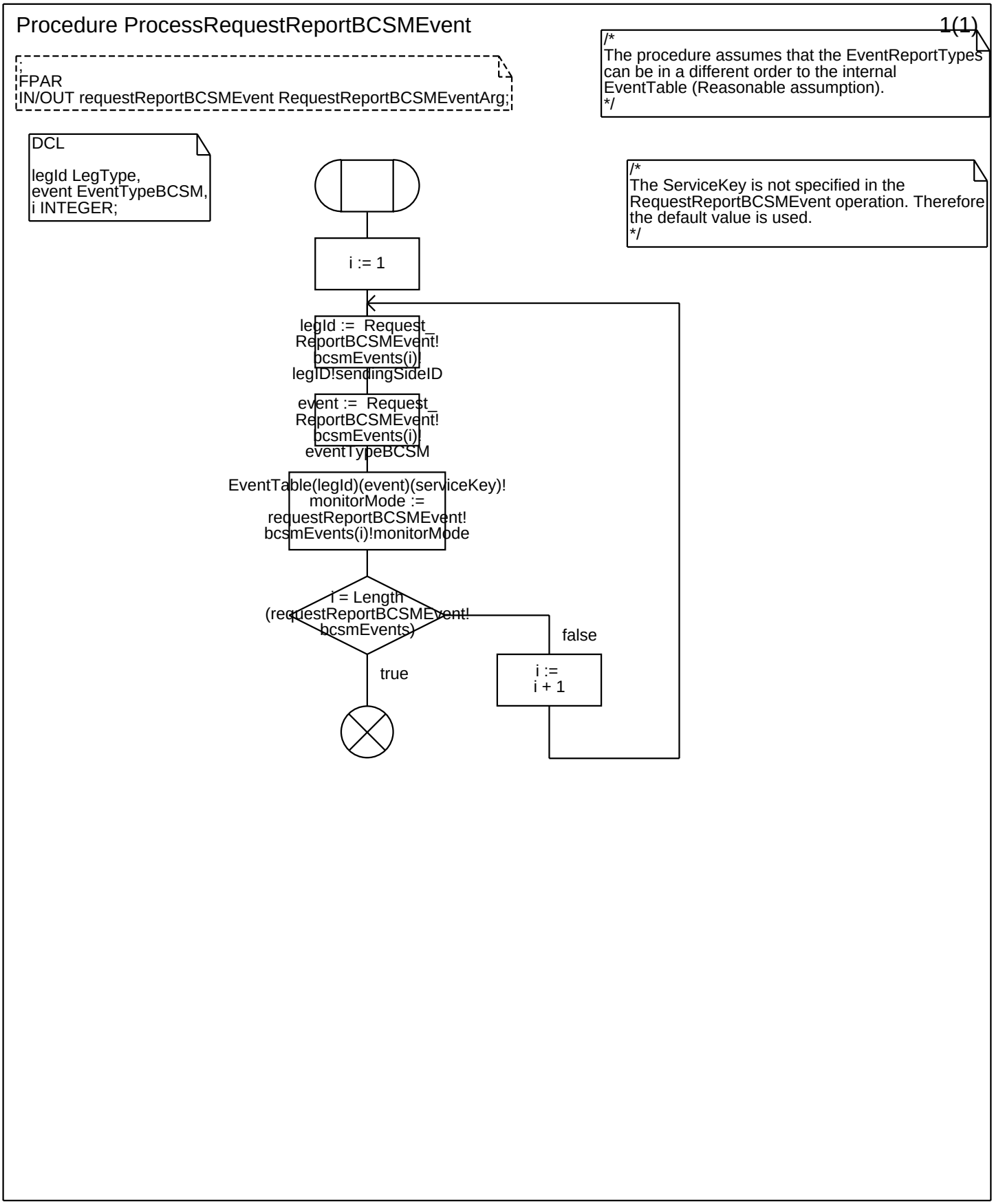
Procedure ProcessApplyCharging 1(1)



Annex A:

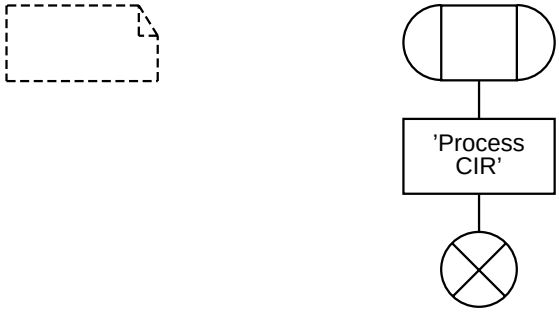

Procedure ProcessContinue
1(1)
/* Note: In case of Disconnect or Abandon, the
of outstanding requests for the suspended leg
greater than one. If so, Continue does not res
the call processing, this can only occur when t
Continue is received. This ensures that the nu
Continues received is equal to the number of
requests (event reports in request mode). */
/* Note:
If the SSF−FSM will transit to Idle as a
of the Continue, then the last DP−R tha
as a TDP−R or EDP−R will be resend t
allow retriggering. */
outstandingRequests (controllingLegID) := outstandingRequests (controllingLegID) − 1
outstandingRequests (controllingLegID)
0
>0
outstandingRequests := (. 0 .)
/* Note:
Send the stored DP−R to Self to allow
scanning of further TDPs. */
dpStored AND NOT ((call AnyDPArmed) OR CallInformationReportPending OR ApplychargingReportPending)
true
storedDP! bcsmEvent
oAbandon, tAbandon
oDisconnect, tDisconnect
else
false
PICResume (controllingLegID) to CS
DPAbandon(dp) to Self
DPAbandon(dp) to Self
DPAbandon(dp) to Self
dpStored := false

Annex A:


Procedure ProcessRequestReportBCSMEvent
1(1)
;
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,
i INTEGER;
/*
The ServiceKey is not specified in the RequestReportBCSMEvent operation. Therefore the default value is used.
*/
i := 1
legId := Request_ReportBCSMEvent! bcsmEvents(i) legID!sendingSideID
event := Request_ReportBCSMEvent! bcsmEvents(i) eventTypeBCSM
EventTable(legId)(event)(serviceKey)! monitorMode := requestReportBCSMEvent! bcsmEvents(i)!monitorMode
i = Length (requestReportBCSMEvent! bcsmEvents)
false
true
i := i + 1

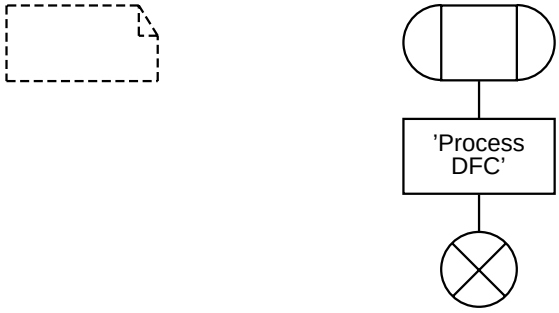
Annex A:

Procedure ProcessCallInformationRequest 1(1)



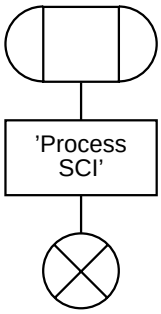
Annex A:

Procedure ProcessDisconnectForwardConnection 1(1)



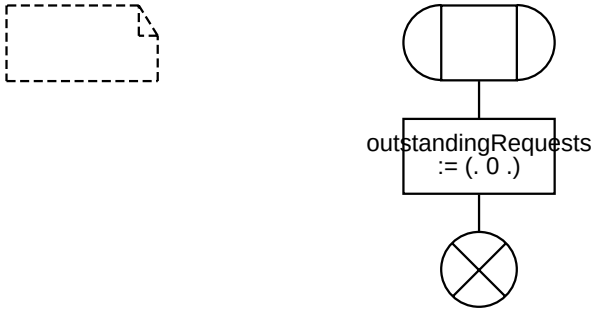
Annex A:

Procedure ProcessSendChargingInformation 1(1)



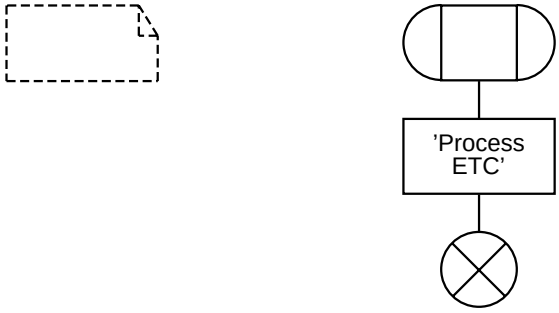
Annex A:

Procedure ProcessCancel 1(1)

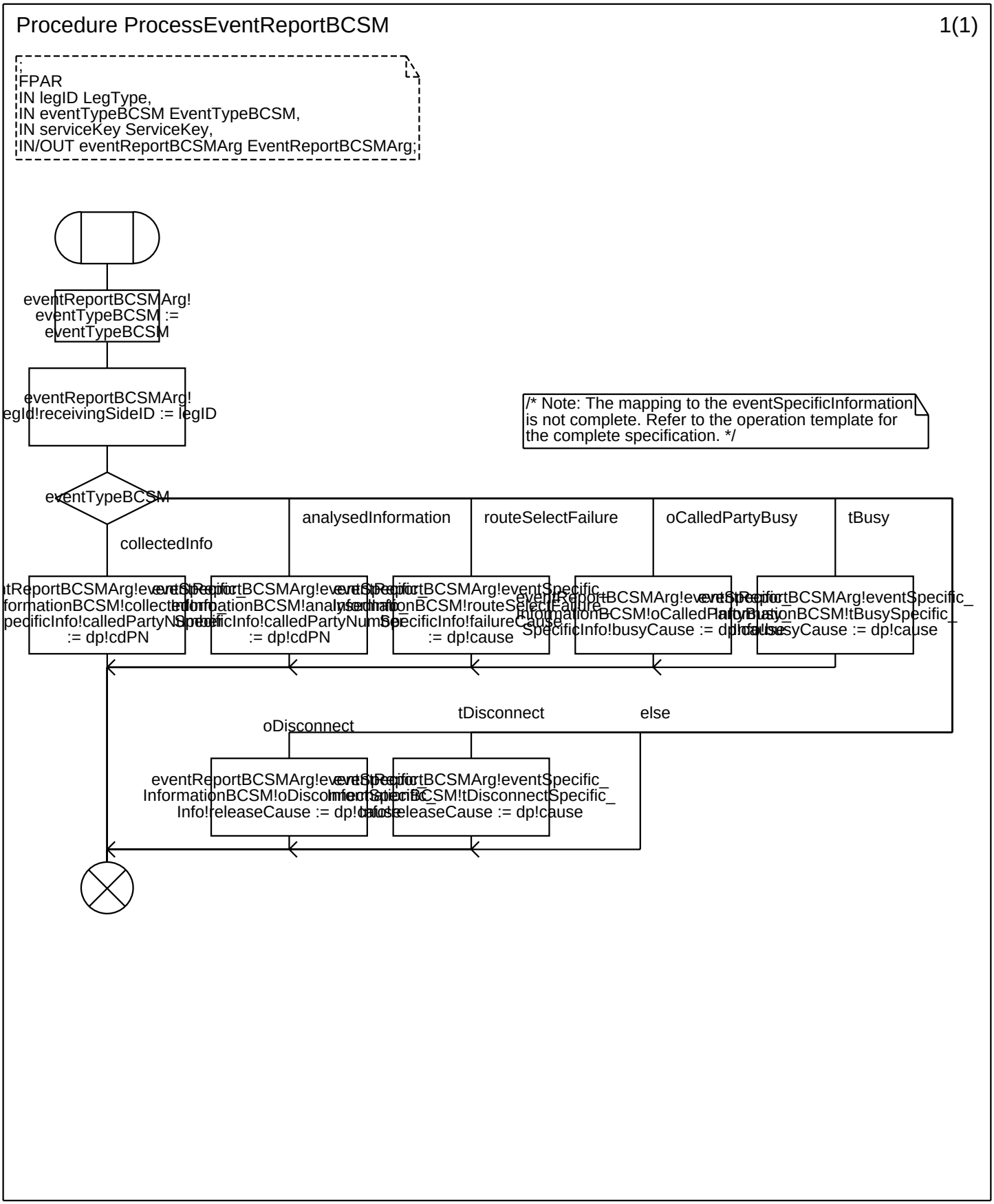


Annex A:

Procedure ProcessEstablishTemporaryConnection 1(1)

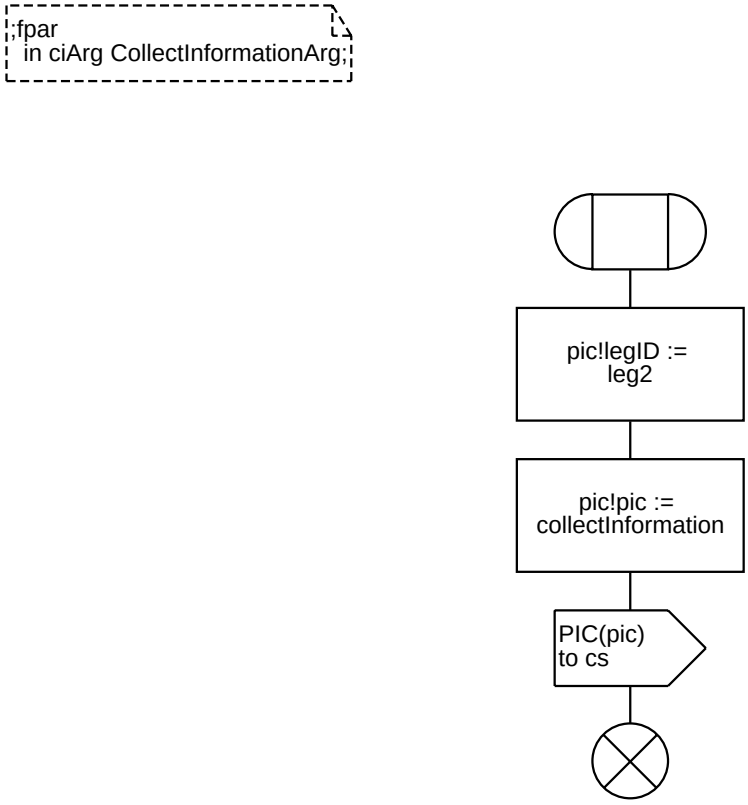


Annex A:


Procedure ProcessEventReportBCSM
1(1)
;
FPAR
IN legID LegType,
IN eventTypeBCSM EventTypeBCSM,
IN serviceKey ServiceKey,
IN/OUT eventReportBCSMArg EventReportBCSMArg;
eventReportBCSMArg!
eventTypeBCSM :=
eventTypeBCSM
eventReportBCSMArg!
legID!receivingSideID := legID
/* Note: The mapping to the eventSpecificInformation
is not complete. Refer to the operation template for
the complete specification. */
eventTypeBCSM
collectedInfo
analysedInformation
routeSelectFailure
oCalledPartyBusy
tBusy
:= dp!cdPN
:= dp!cdPN
:= dp!cause
SpecificInfo!busyCause := dp!cause
:= dp!cause
else
tDisconnect
oDisconnect
InformationBCSM!oDisconnectSpecific_
Info!releaseCause := dp!cause
InformationBCSM!tDisconnectSpecific_
releaseCause := dp!cause

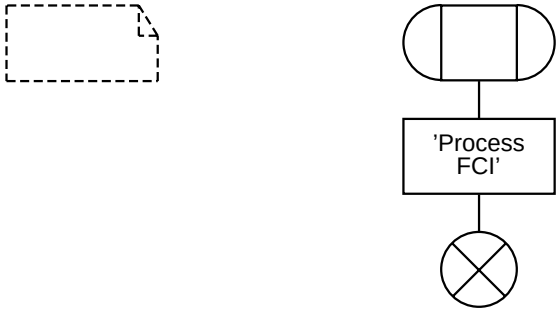
Annex A:

Procedure ProcessCollectInformation 1(1)



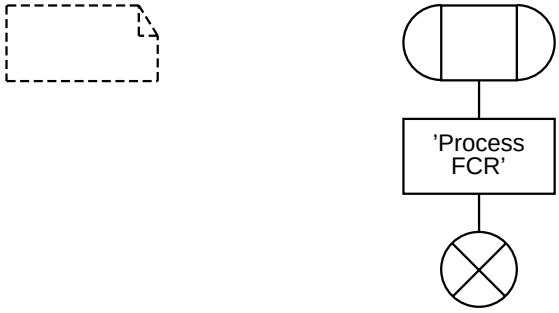
Annex A:

Procedure ProcessFurnishChargingInformation 1(1)

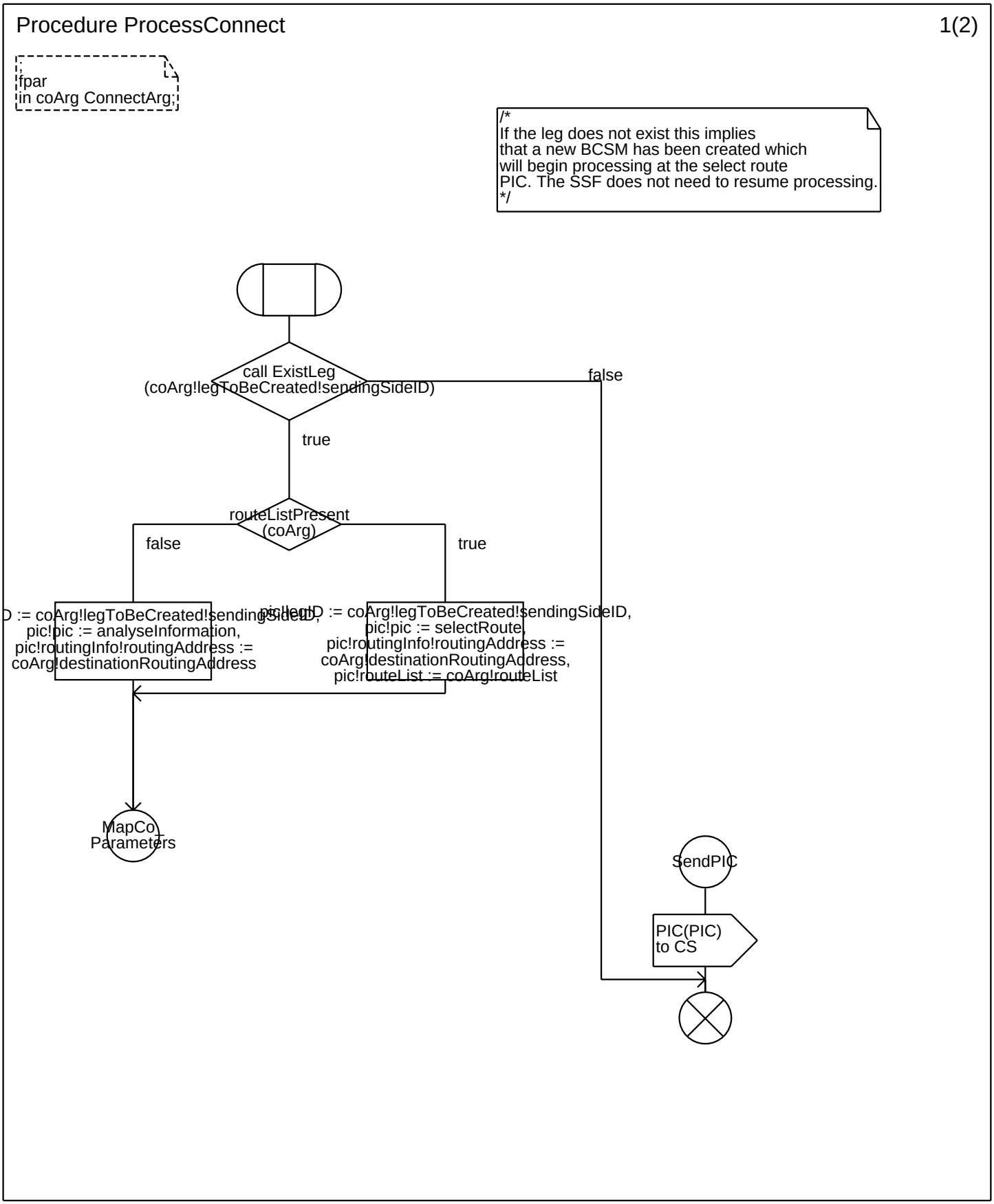


Annex A:

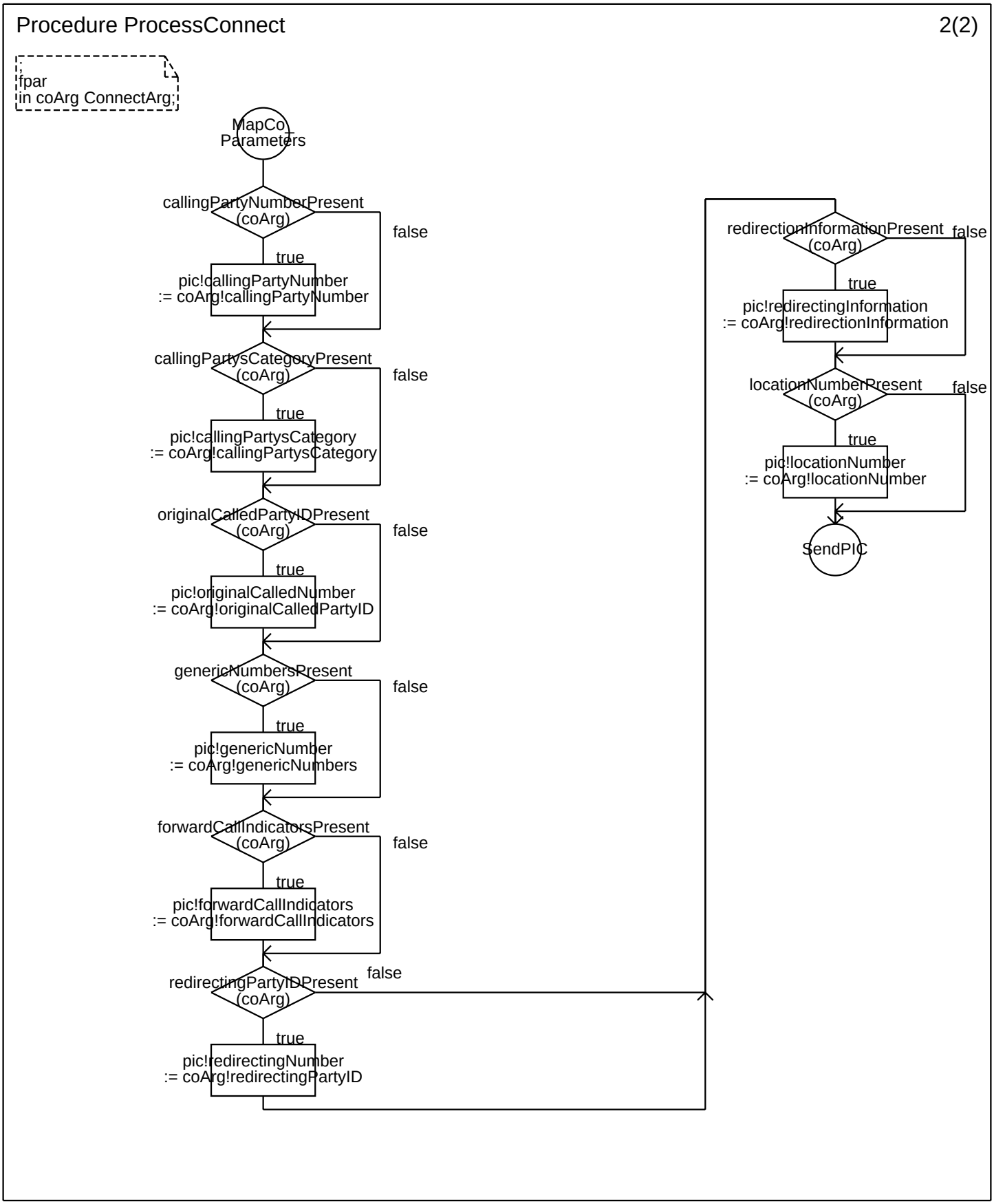
Procedure ProcessForwardConnectionReleased 1(1)



Annex A:

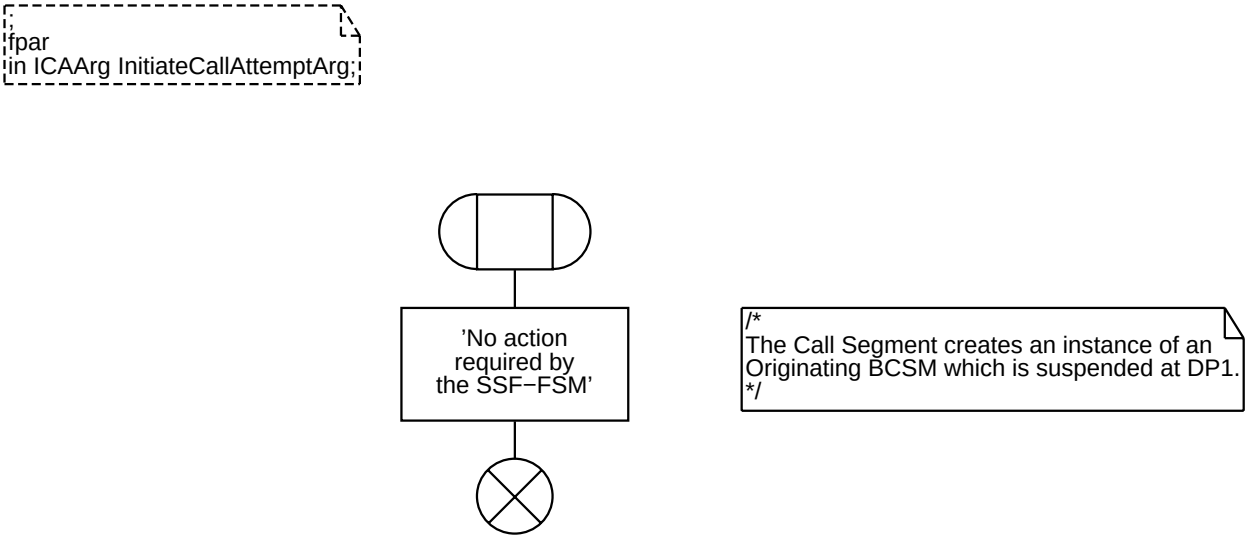

Procedure ProcessConnect
1(2)
;
fpar
in coArg ConnectArg;
/*
If the leg does not exist this implies
that a new BCSM has been created which
will begin processing at the select route
PIC. The SSF does not need to resume processing.
*/
call ExistLeg
(coArg!legToBeCreated!sendingSideID)
false
true
routeListPresent
(coArg)
false
true
D := coArg!legToBeCreated!sendingSideID,
pic!pic := analyseInformation,
pic!routingInfo!routingAddress :=
coArg!destinationRoutingAddress
pic!legID := coArg!legToBeCreated!sendingSideID,
pic!pic := selectRoute,
pic!routingInfo!routingAddress :=
coArg!destinationRoutingAddress,
pic!routeList := coArg!routeList
MapCo
Parameters
SendPIC
PIC(PIC)
to CS

Annex A:

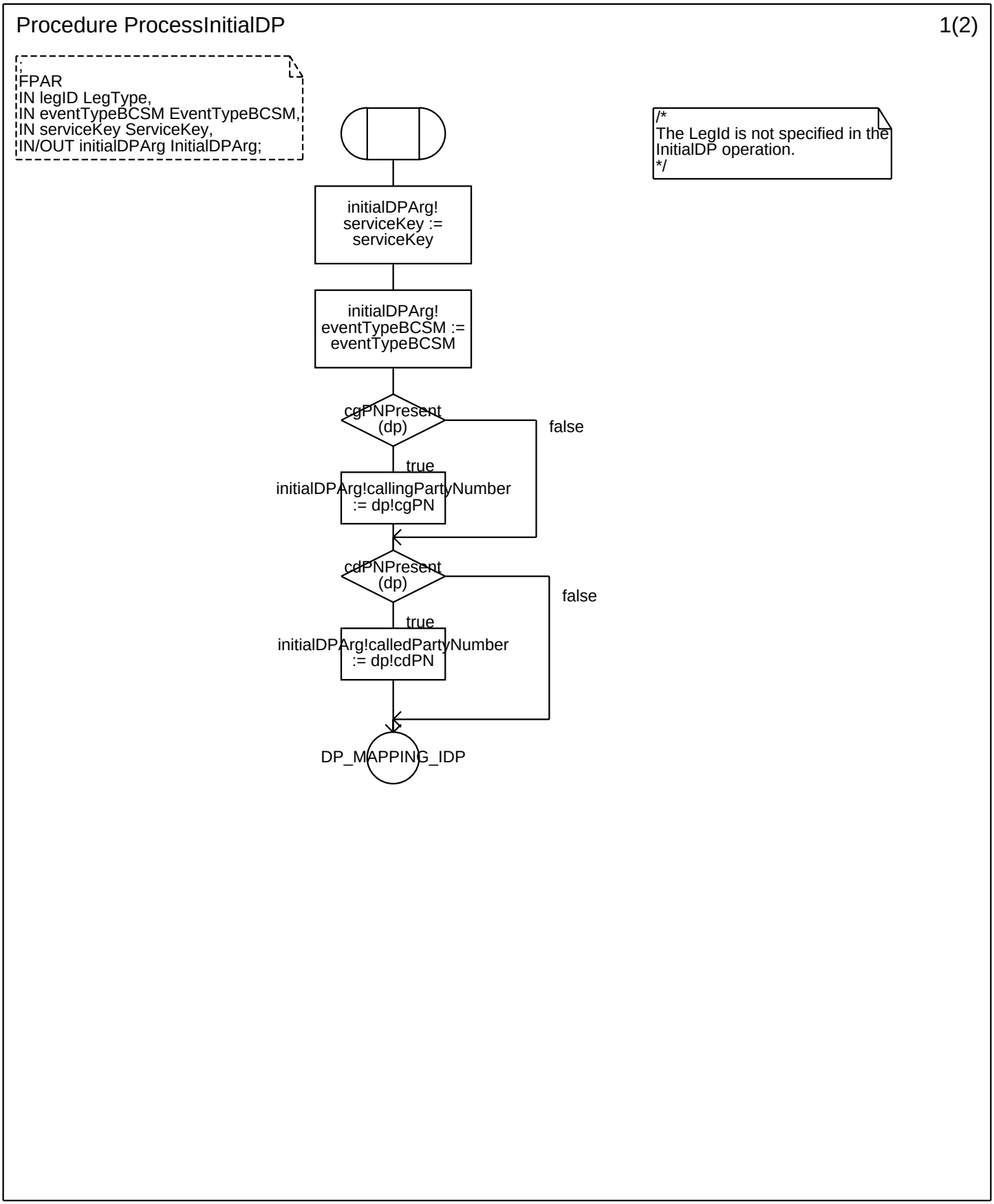

Procedure ProcessConnect
2(2)
fpar
in coArg ConnectArg;
MapCoParameters
callingPartyNumberPresent (coArg)
false
true
pic!callingPartyNumber := coArg!callingPartyNumber
callingPartysCategoryPresent (coArg)
false
true
pic!callingPartysCategory := coArg!callingPartysCategory
originalCalledPartyIDPresent (coArg)
false
true
pic!originalCalledNumber := coArg!originalCalledPartyID
genericNumbersPresent (coArg)
false
true
pic!genericNumber := coArg!genericNumbers
forwardCallIndicatorsPresent (coArg)
false
true
pic!forwardCallIndicators := coArg!forwardCallIndicators
redirectingPartyIDPresent (coArg)
false
true
pic!redirectingNumber := coArg!redirectingPartyID
redirectionInformationPresent (coArg)
false
true
pic!redirectingInformation := coArg!redirectionInformation
locationNumberPresent (coArg)
false
true
pic!locationNumber := coArg!locationNumber
SendPIC

Annex A:

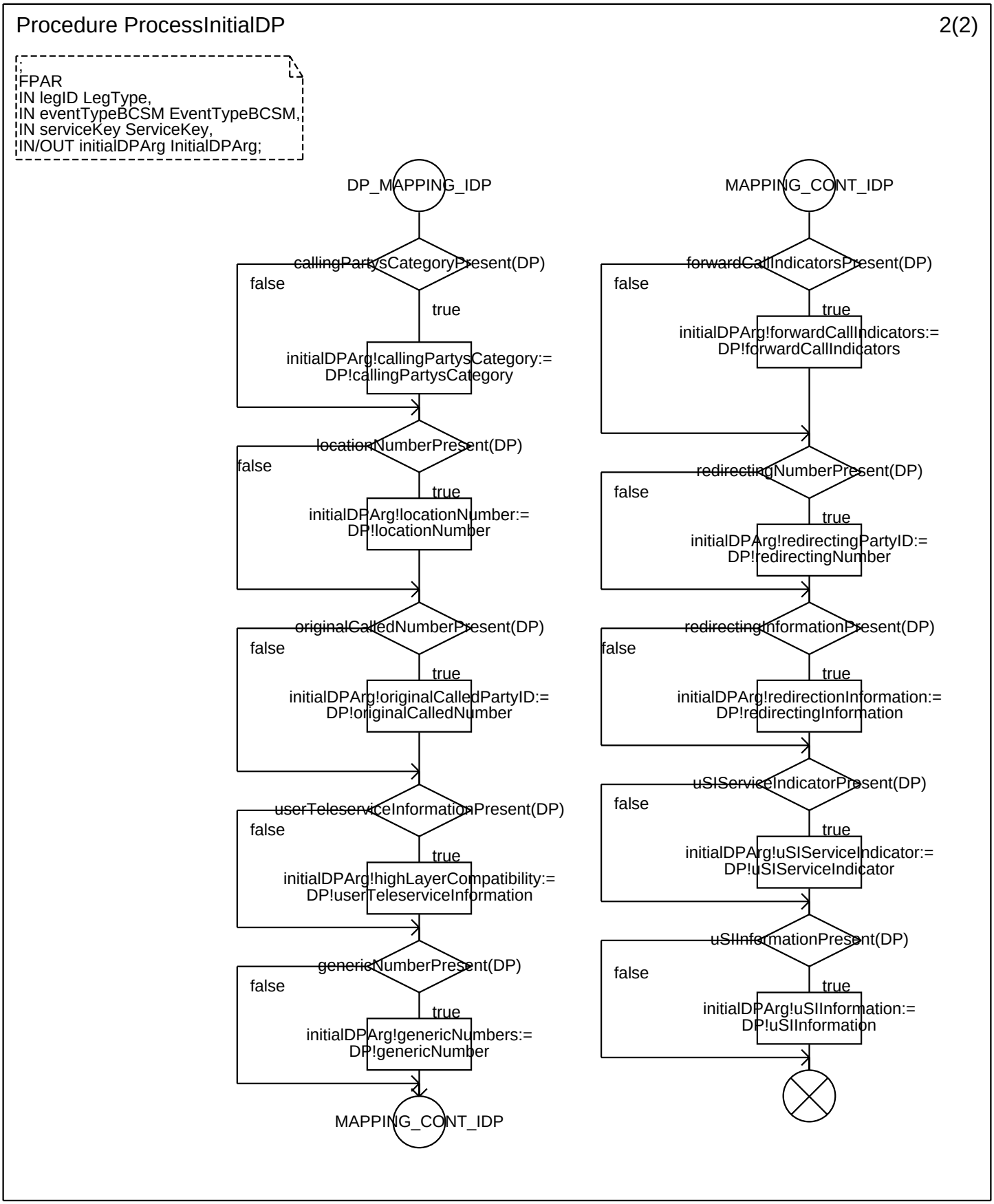
Procedure ProcessInitiateCallAttempt 1(1)



Annex A:

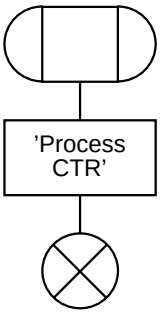

Procedure ProcessInitialDP
1(2)
;
FPAR
IN legID LegType,
IN eventTypeBCSM EventTypeBCSM,
IN serviceKey ServiceKey,
IN/OUT initialDPArg InitialDPArg;
/*
The LegId is not specified in the InitialDP operation.
*/
initialDPArg! serviceKey := serviceKey
initialDPArg! eventTypeBCSM := eventTypeBCSM
cgPNPresent (dp)
false
true
initialDPArg!callingPartyNumber := dp!cgPN
cdPNPresent (dp)
false
true
initialDPArg!calledPartyNumber := dp!cdPN
DP_MAPPING_IDP

Annex A:


Procedure ProcessInitialDP
2(2)
FPAR
IN legID LegType,
IN eventTypeBCSM EventTypeBCSM,
IN serviceKey ServiceKey,
IN/OUT initialDPArg InitialDPArg;
DP_MAPPING_IDP
callingPartysCategoryPresent(DP)
false
true
initialDPArg!callingPartysCategory:=
DP!callingPartysCategory
locationNumberPresent(DP)
false
true
initialDPArg!locationNumber:=
DP!locationNumber
originalCalledNumberPresent(DP)
false
true
initialDPArg!originalCalledPartyID:=
DP!originalCalledNumber
userTeleserviceInformationPresent(DP)
false
true
initialDPArg!highLayerCompatibility:=
DP!userTeleserviceInformation
genericNumberPresent(DP)
false
true
initialDPArg!genericNumbers:=
DP!genericNumber
MAPPING_CONT_IDP
MAPPING_CONT_IDP
forwardCallIndicatorsPresent(DP)
false
true
initialDPArg!forwardCallIndicators:=
DP!forwardCallIndicators
redirectingNumberPresent(DP)
false
true
initialDPArg!redirectingPartyID:=
DP!redirectingNumber
redirectingInformationPresent(DP)
false
true
initialDPArg!redirectionInformation:=
DP!redirectingInformation
uSIServiceIndicatorPresent(DP)
false
true
initialDPArg!uSIServiceIndicator:=
DP!uSIServiceIndicator
uSIInformationPresent(DP)
false
true
initialDPArg!uSIInformation:=
DP!uSIInformation

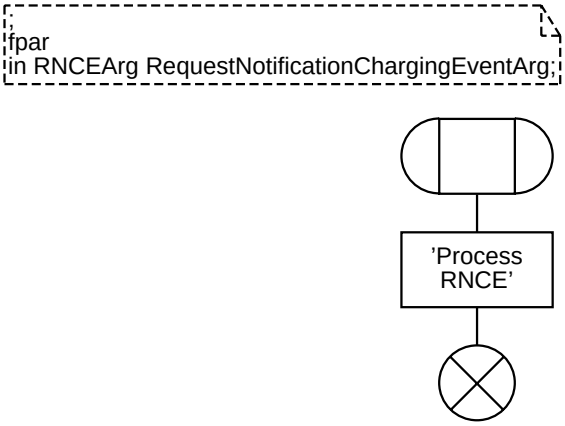
Annex A:

Procedure ProcessConnectToResource 1(1)



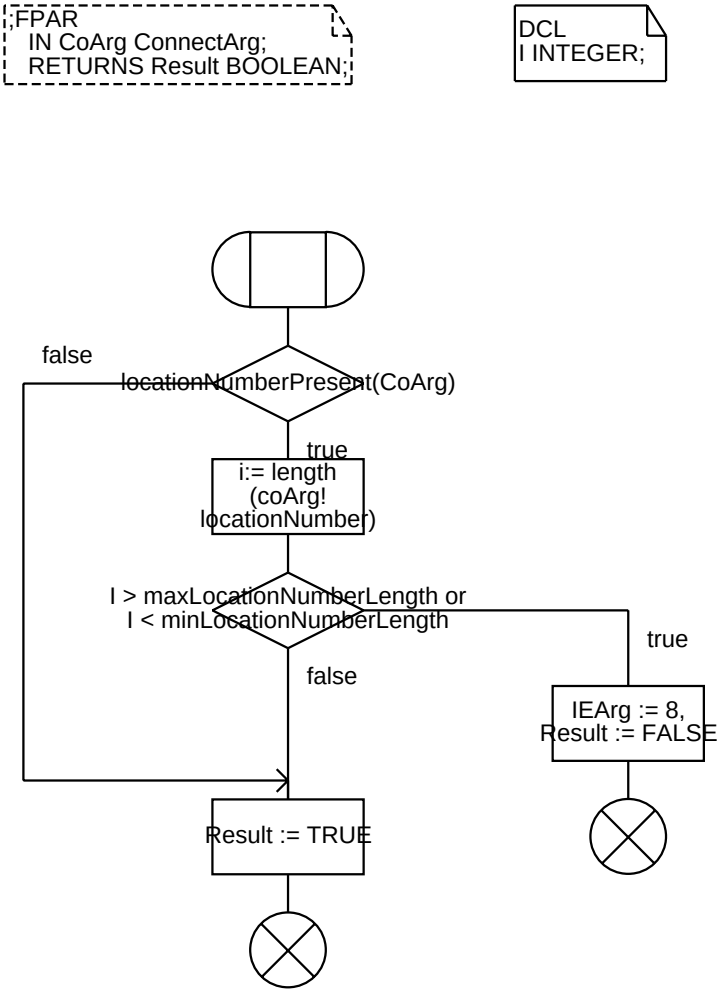
Annex A:

Procedure ProcessRequestNotificationChargingEvent 1(1)



Annex A:

Procedure ConnnectAnalysis 1(1)



Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingE1(11)l

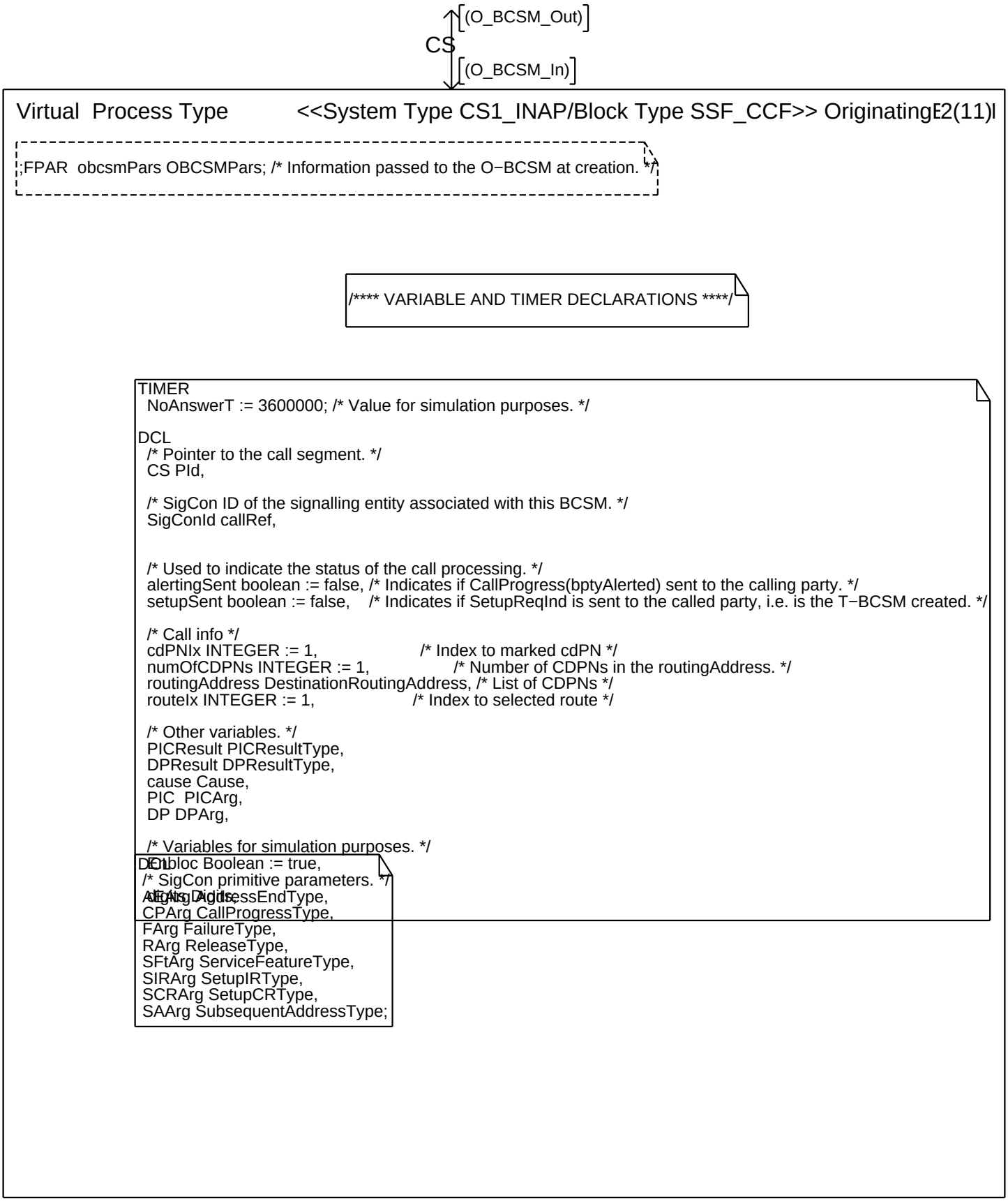
```
;FPAR  obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
```

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

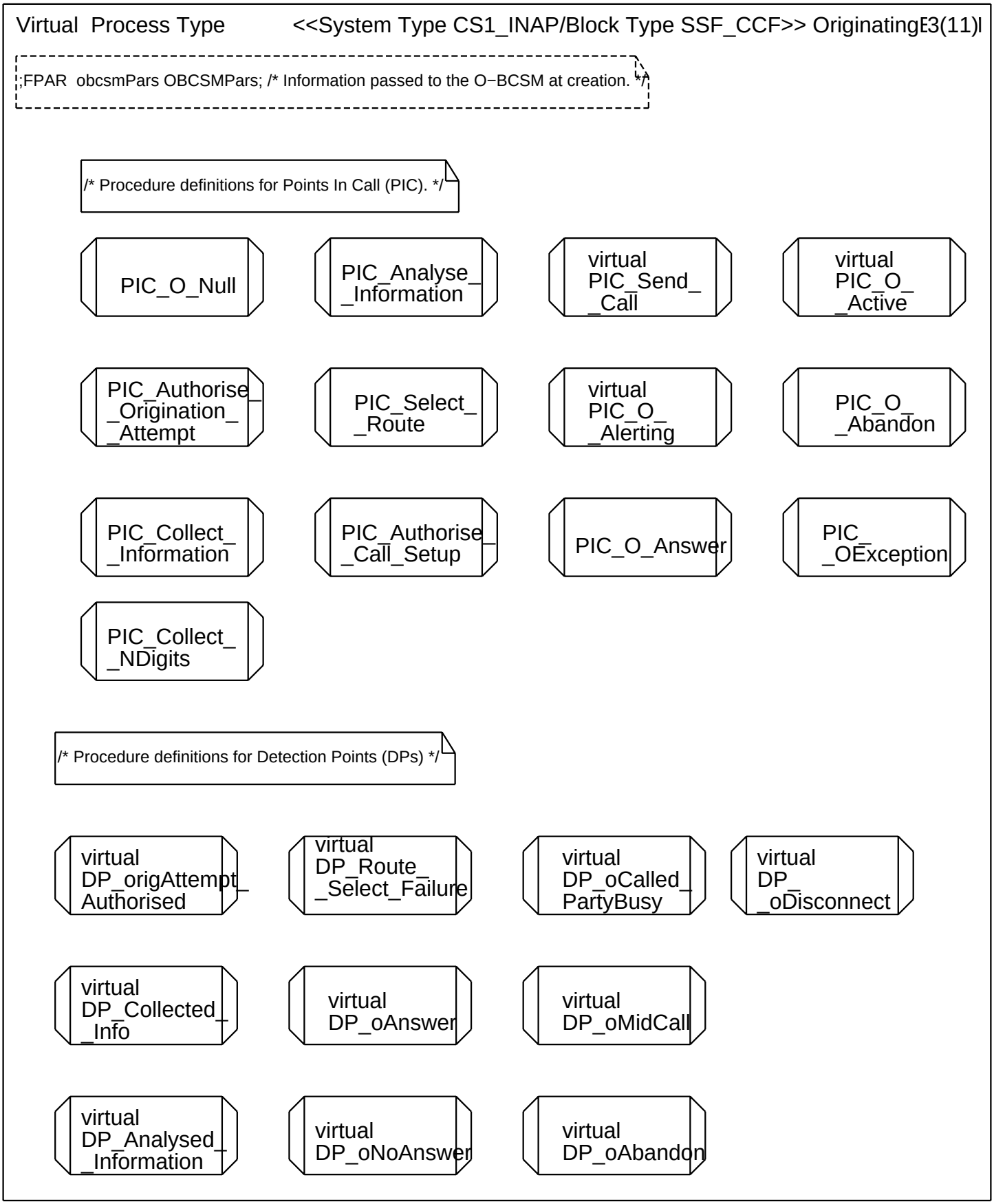
```
/***** DATA TYPE DEFINITIONS *****/
```

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

    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 */

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

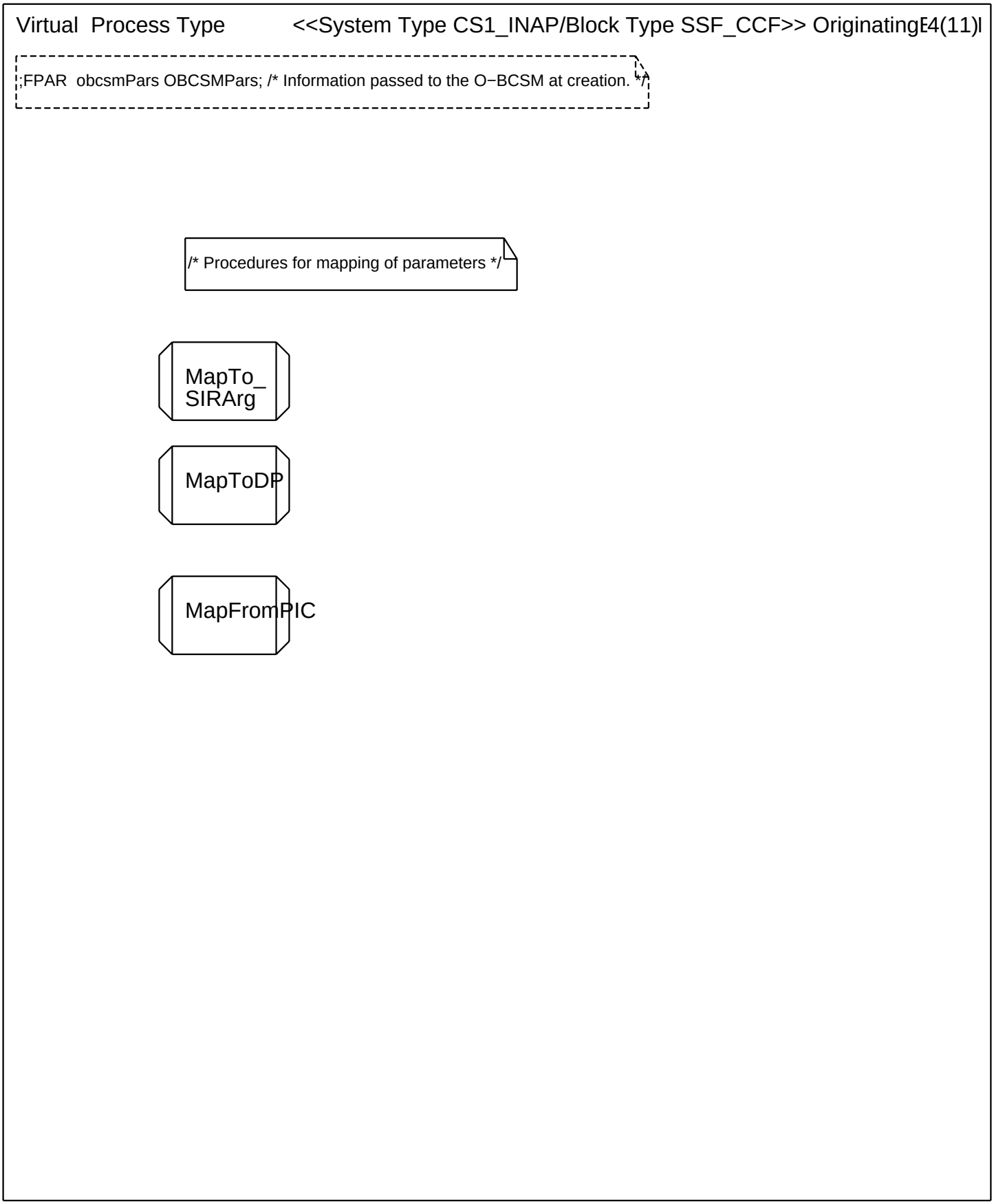
Annex A:


(O_BCSM_Out)
CS
(O_BCSM_In)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingE2(11)l
;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. */
DCL
/* SigCon primitive parameters. */
CPArg CallProgressType,
FArg FailureType,
RArg ReleaseType,
SFtArg ServiceFeatureType,
SIRArg SetupIRType,
SCRArg SetupCRType,
SAArg SubsequentAddressType;

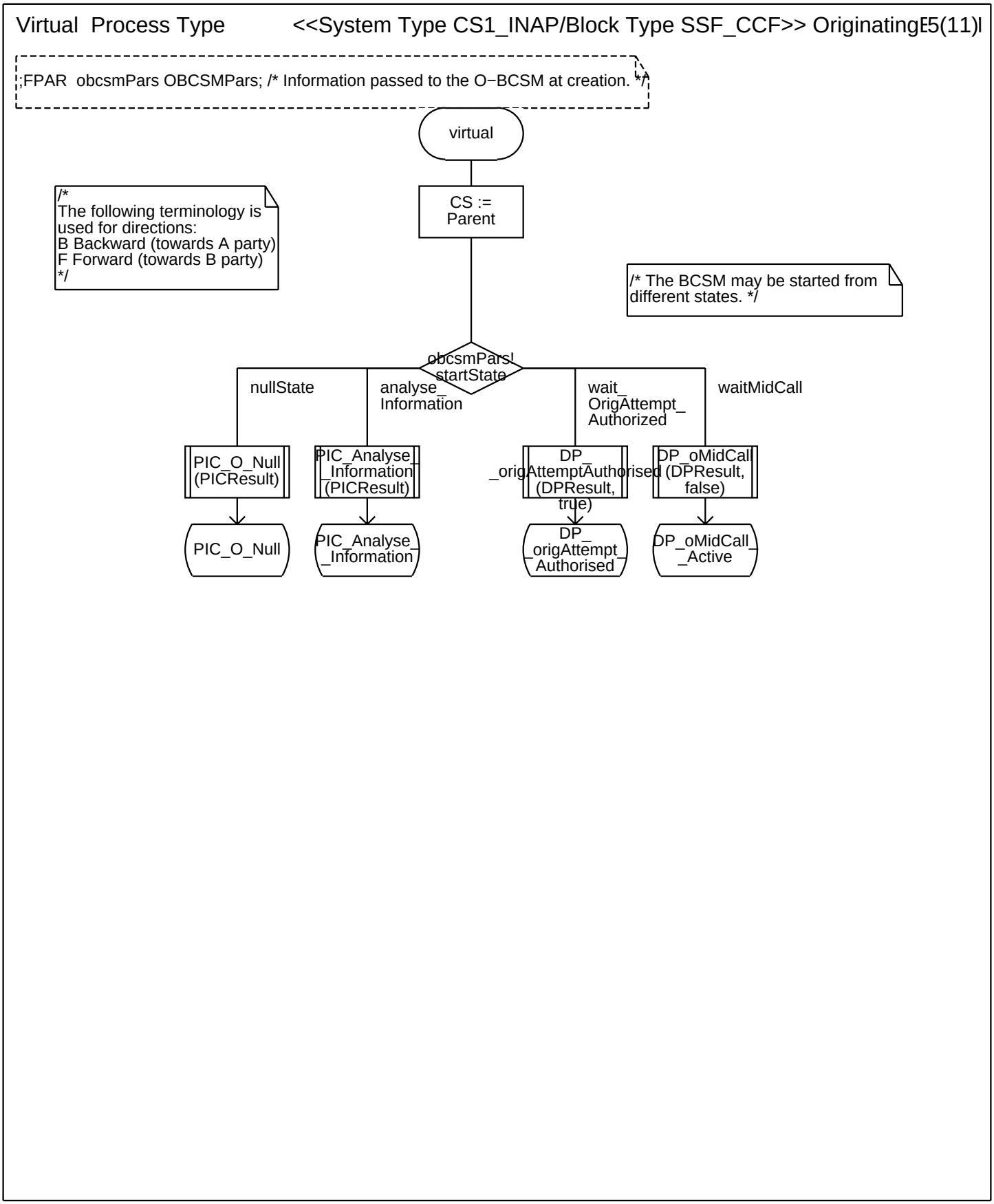
Annex A:


Virtual Process Type
<<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingE3(11)l
;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

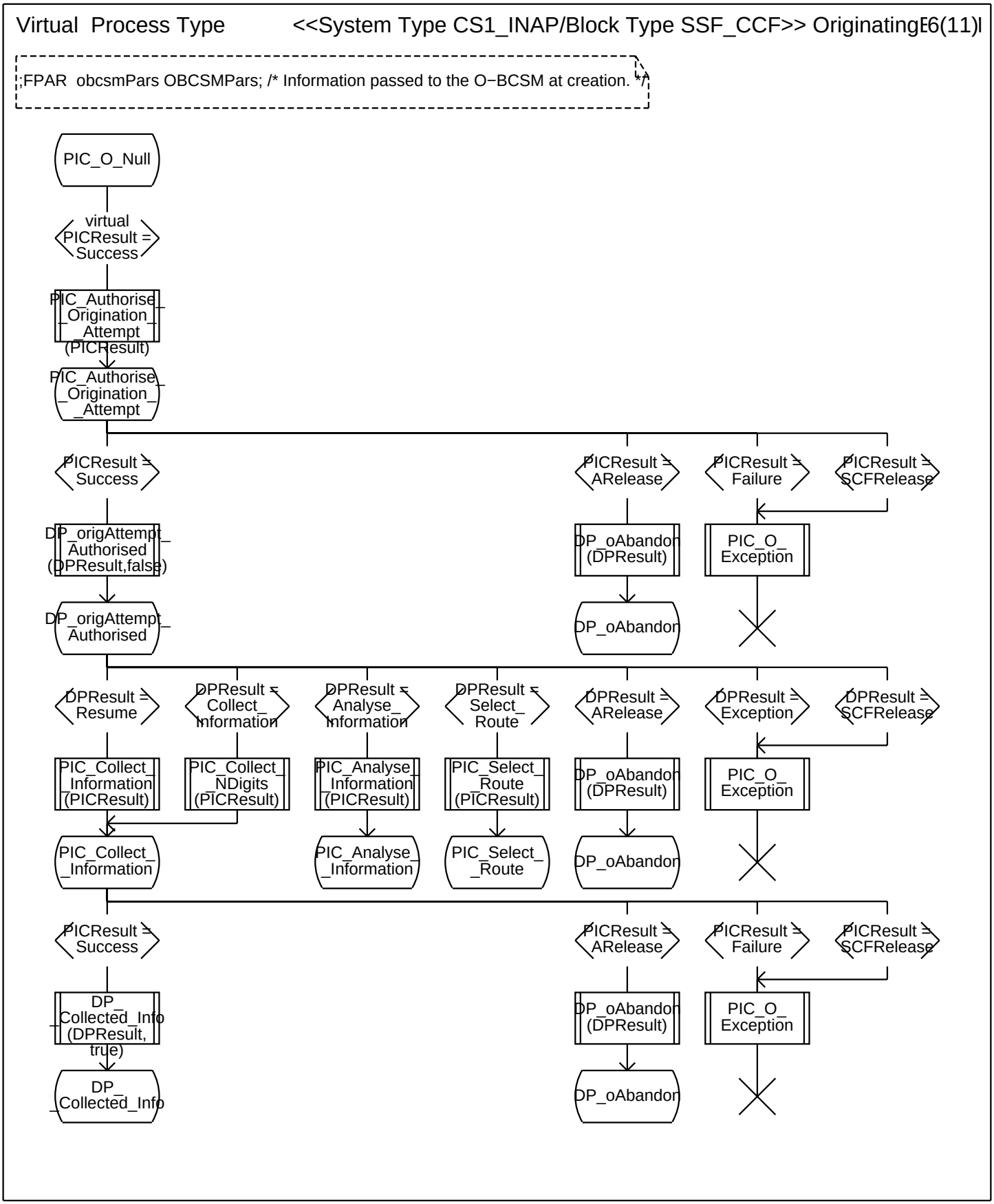
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingE4(11)I
;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
/* Procedures for mapping of parameters */
MapTo_
SIRArg
MapToDP
MapFromPIC

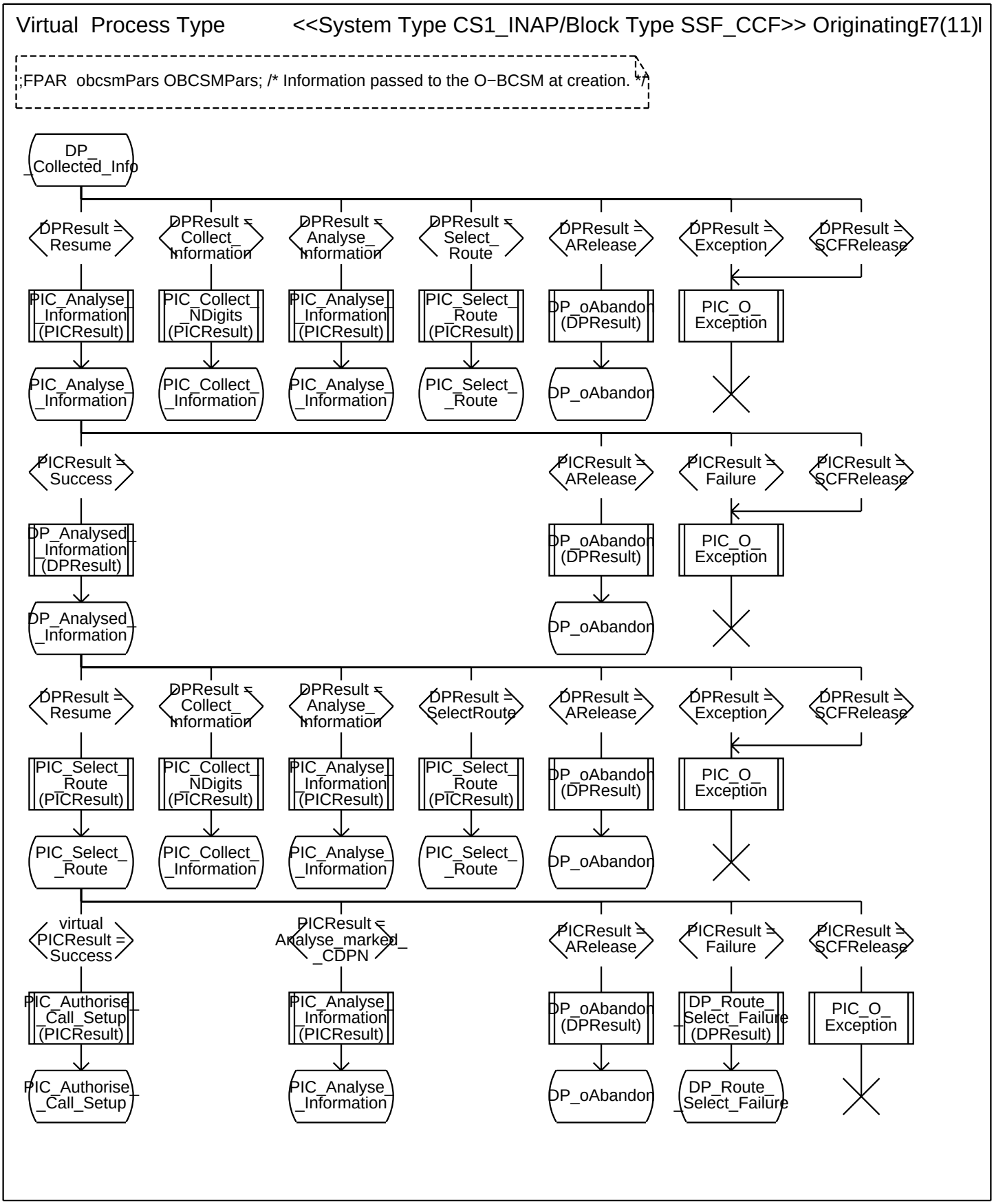
Annex A:


Virtual Process Type
<<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingE5(11)l
;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
virtual
CS := Parent
/* The following terminology is used for directions: B Backward (towards A party) F Forward (towards B party) */
/* 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

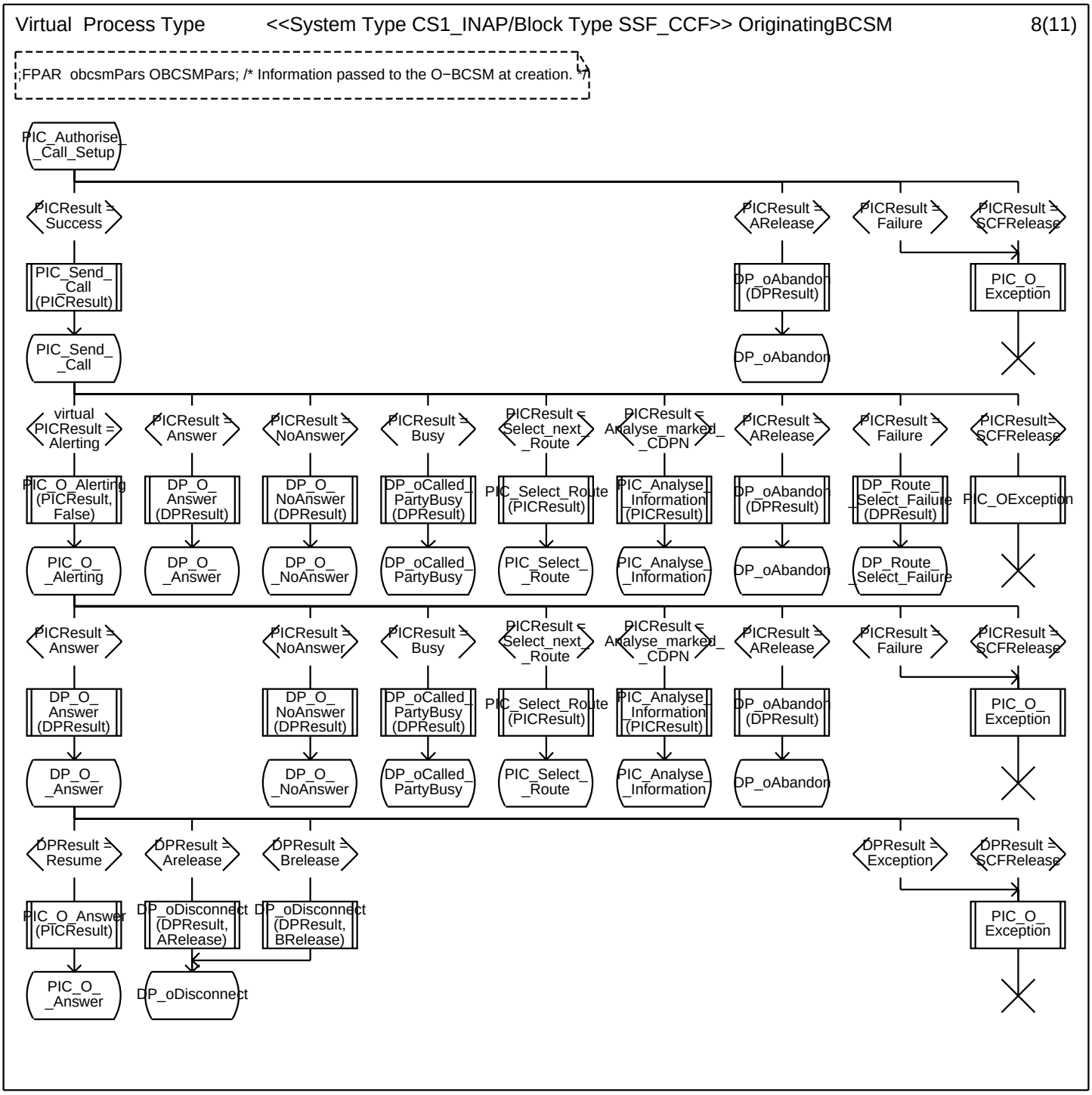
Annex A:


Virtual Process Type
<<System Type CS1_INAP/Block Type SSF_CCF>> Originating
E6(11)
;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,false)
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

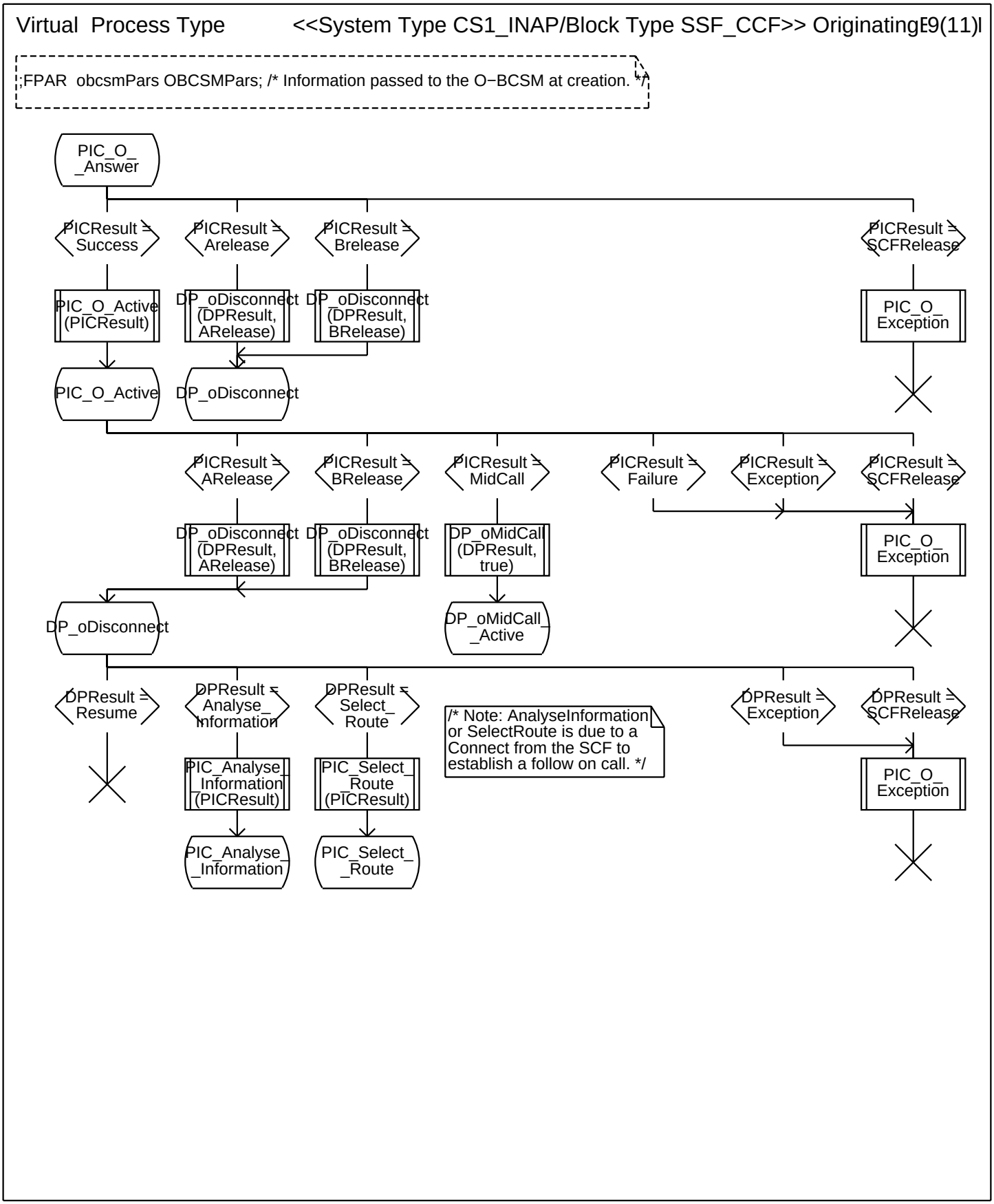
Annex A:


Virtual Process Type
<<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingE7(11)I
;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

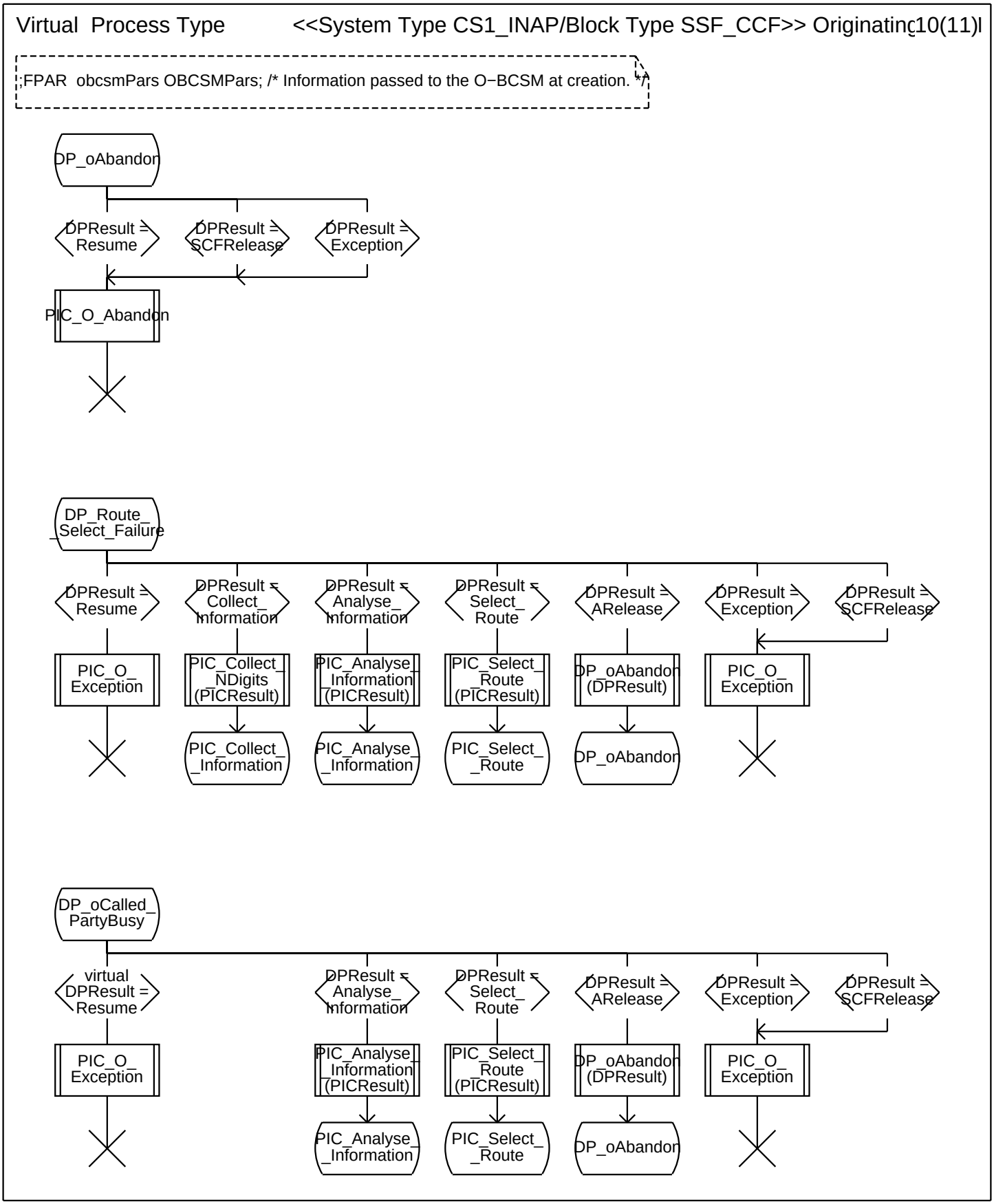
Annex A:


Virtual Process Type
<<System Type CS1_INAP/Block Type SSF_CCF>> OriginatingBCSM
8(11)
;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
PIC_Authorise_
_Call_Setup
PICResult =
Success
PICResult =
ARelease
PICResult =
Failure
PICResult =
SCFRelease
PIC_Send_
_Call
(PICResult)
DP_oAbandon
(DPResult)
PIC_O_
Exception
PIC_Send_
_Call
DP_oAbandon
virtual
PICResult =
Alerting
PICResult =
Answer
PICResult =
NoAnswer
PICResult =
Busy
PICResult =
Select_next_
_Route
PICResult =
Analyse_marked_
_CDPN
PICResult =
ARelease
PICResult =
Failure
PICResult=
SCFRelease
PIC_O_Alerting
(PICResult,
False)
DP_O_
Answer
(DPResult)
DP_O_
NoAnswer
(DPResult)
DP_oCalled_
PartyBusy
(DPResult)
PIC_Select_Route
(PICResult)
PIC_Analyse_
_Information
(PICResult)
DP_oAbandon
(DPResult)
DP_Route_
Select_Failure
(DPResult)
PIC_OException
PIC_O_
_Alerting
DP_O_
_Answer
DP_O_
_NoAnswer
DP_oCalled_
PartyBusy
PIC_Select_
_Route
PIC_Analyse_
_Information
DP_oAbandon
DP_Route_
_Select_Failure
PICResult =
Answer
PICResult =
NoAnswer
PICResult =
Busy
PICResult =
Select_next_
_Route
PICResult =
Analyse_marked_
_CDPN
PICResult =
ARelease
PICResult =
Failure
PICResult =
SCFRelease
DP_O_
Answer
(DPResult)
DP_O_
NoAnswer
(DPResult)
DP_oCalled_
PartyBusy
(DPResult)
PIC_Select_Route
(PICResult)
PIC_Analyse_
_Information
(PICResult)
DP_oAbandon
(DPResult)
PIC_O_
Exception
DP_O_
_Answer
DP_O_
_NoAnswer
DP_oCalled_
PartyBusy
PIC_Select_
_Route
PIC_Analyse_
_Information
DP_oAbandon
DPResult =
Resume
DPResult =
Arelease
DPResult =
Brelease
DPResult =
Exception
DPResult =
SCFRelease
PIC_O_Answer
(PICResult)
DP_oDisconnect
(DPResult,
ARelease)
DP_oDisconnect
(DPResult,
BRelease)
PIC_O_
Exception
PIC_O_
_Answer
DP_oDisconnect

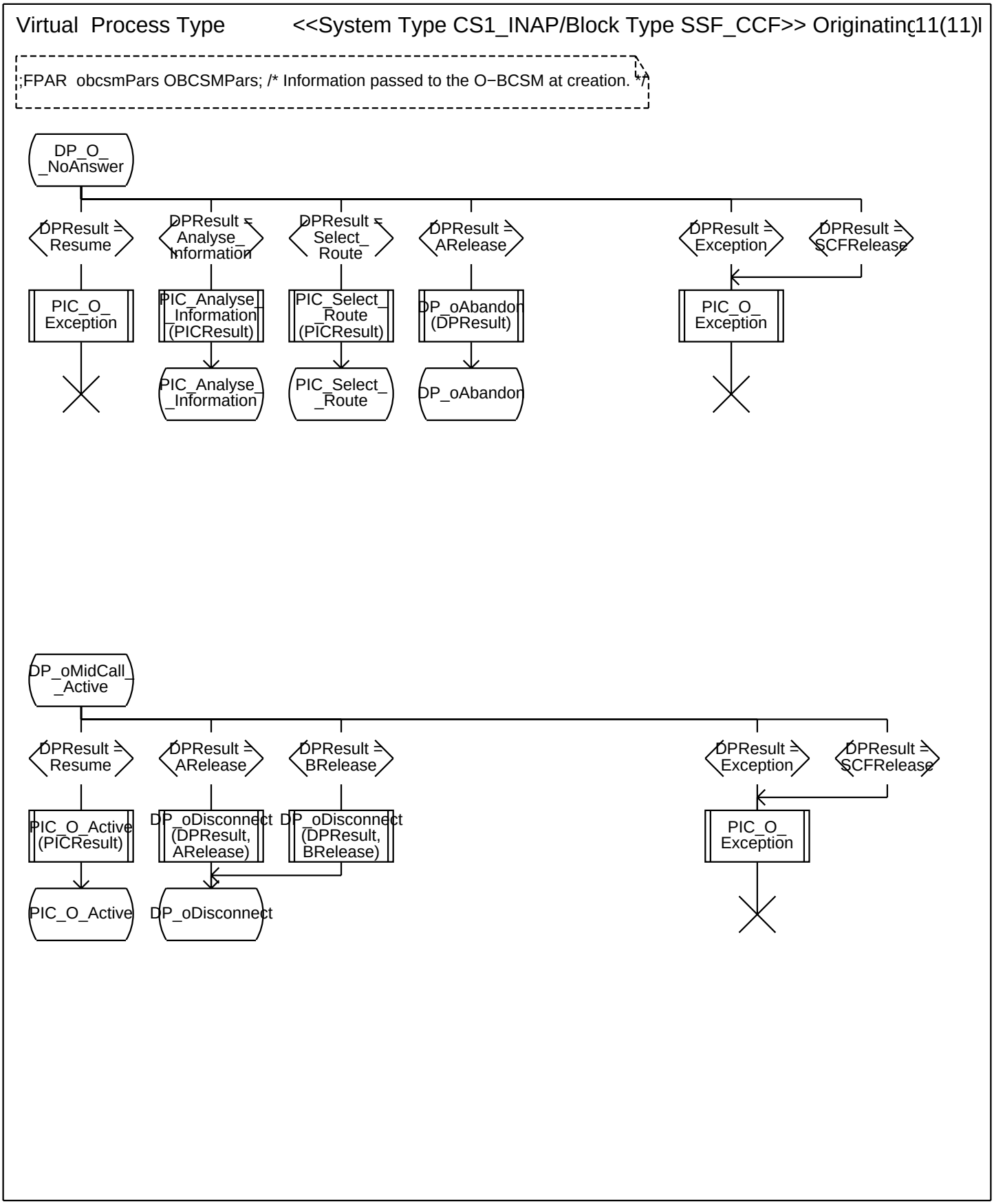
Annex A:


Virtual Process Type
<<System Type CS1_INAP/Block Type SSF_CCF>> Originating
E9(11)
;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
PIC_O_ _Answer
PICResult = Success
PICResult = Arelease
PICResult = 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

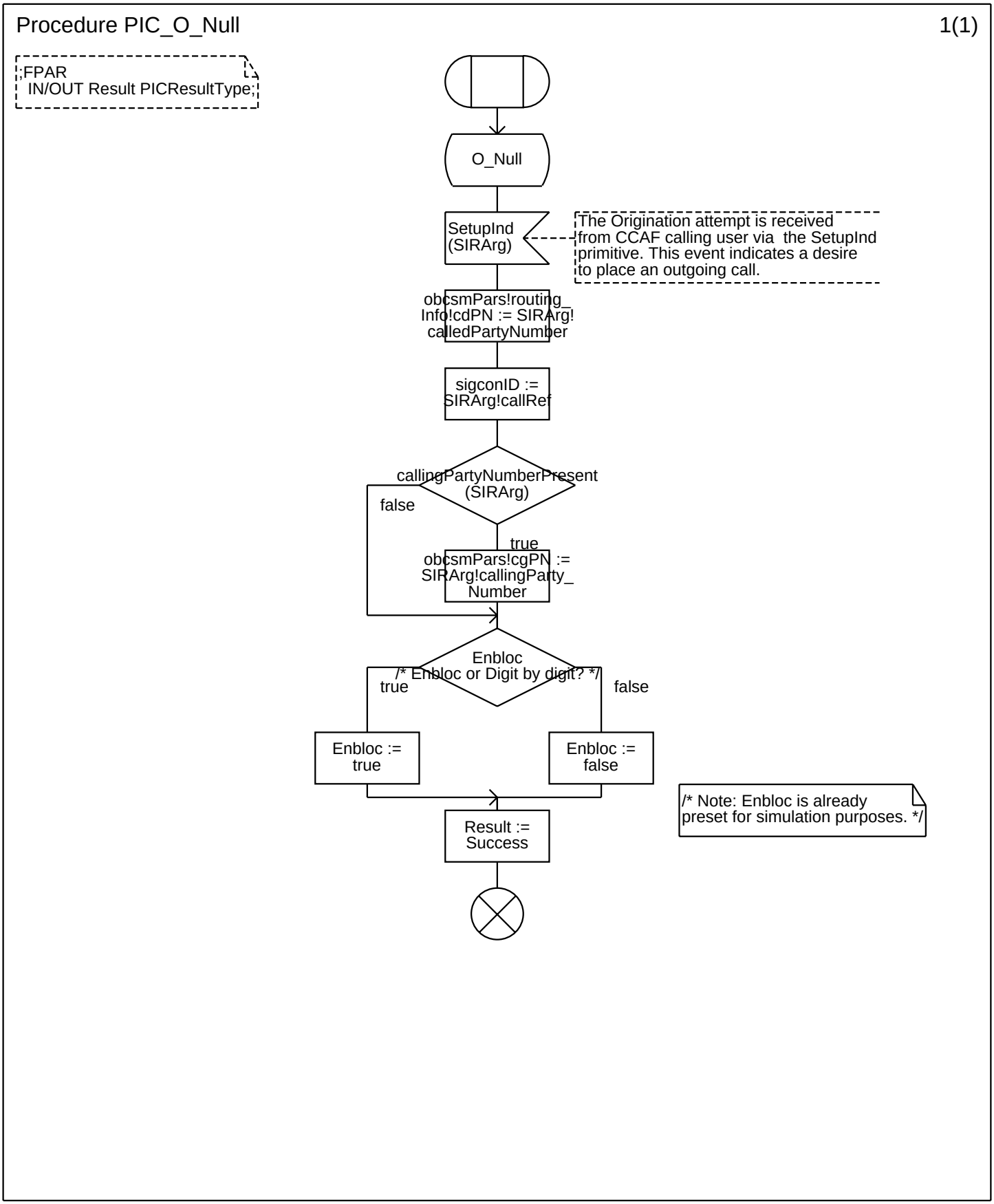
Annex A:


Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> Originating 10(11)
;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

Annex A:

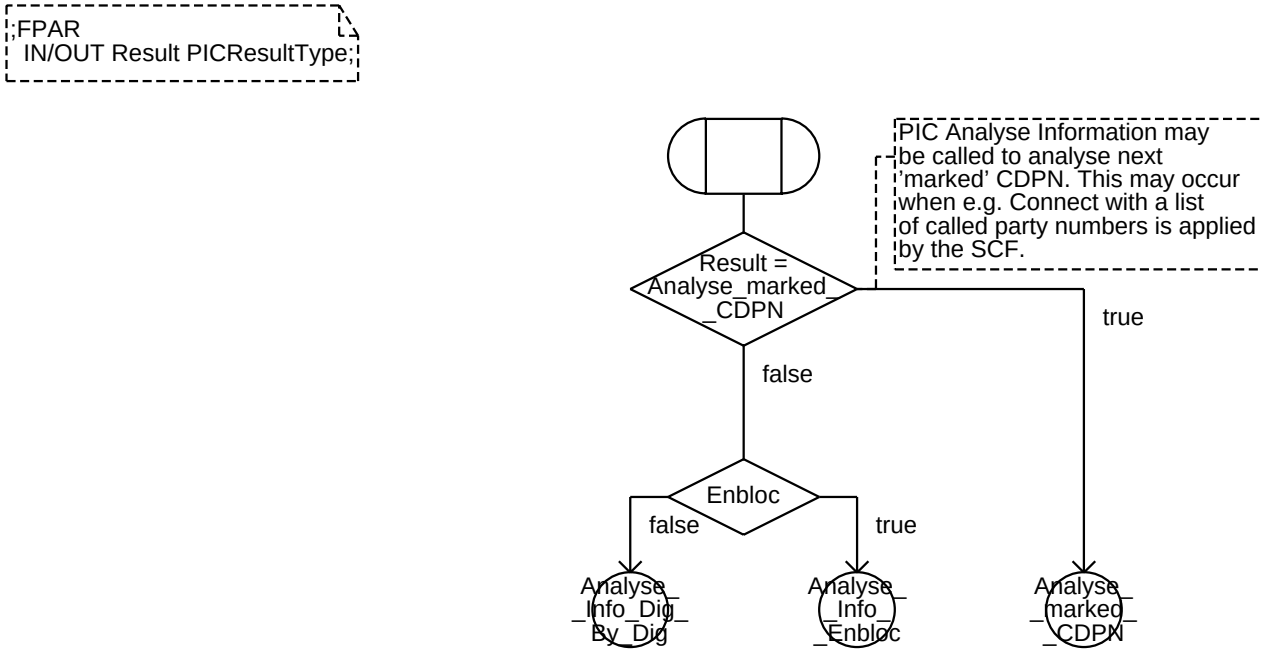

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> Originating 11(11)
;FPAR obcsmPars OBCSMPars; /* Information passed to the O−BCSM at creation. */
DP_O_ _NoAnswer
DPResult = Resume
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = 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
DPResult = ARelease
DPResult = 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

Annex A:


Procedure PIC_O_Null
1(1)
;FPAR
IN/OUT Result PICResultType;
O_Null
SetupInd
(SIRArg)
The Origination attempt is received from CCAF calling user via the SetupInd primitive. This event indicates a desire to place an outgoing call.
obcsmPars!routing_Info!cdPN := SIRArg!calledPartyNumber
sigconID := SIRArg!callRef
callingPartyNumberPresent
(SIRArg)
false
true
obcsmPars!cgPN := SIRArg!callingParty_Number
Enbloc
/* Enbloc or Digit by digit? */
true
false
Enbloc := true
Enbloc := false
/* Note: Enbloc is already preset for simulation purposes. */
Result := Success

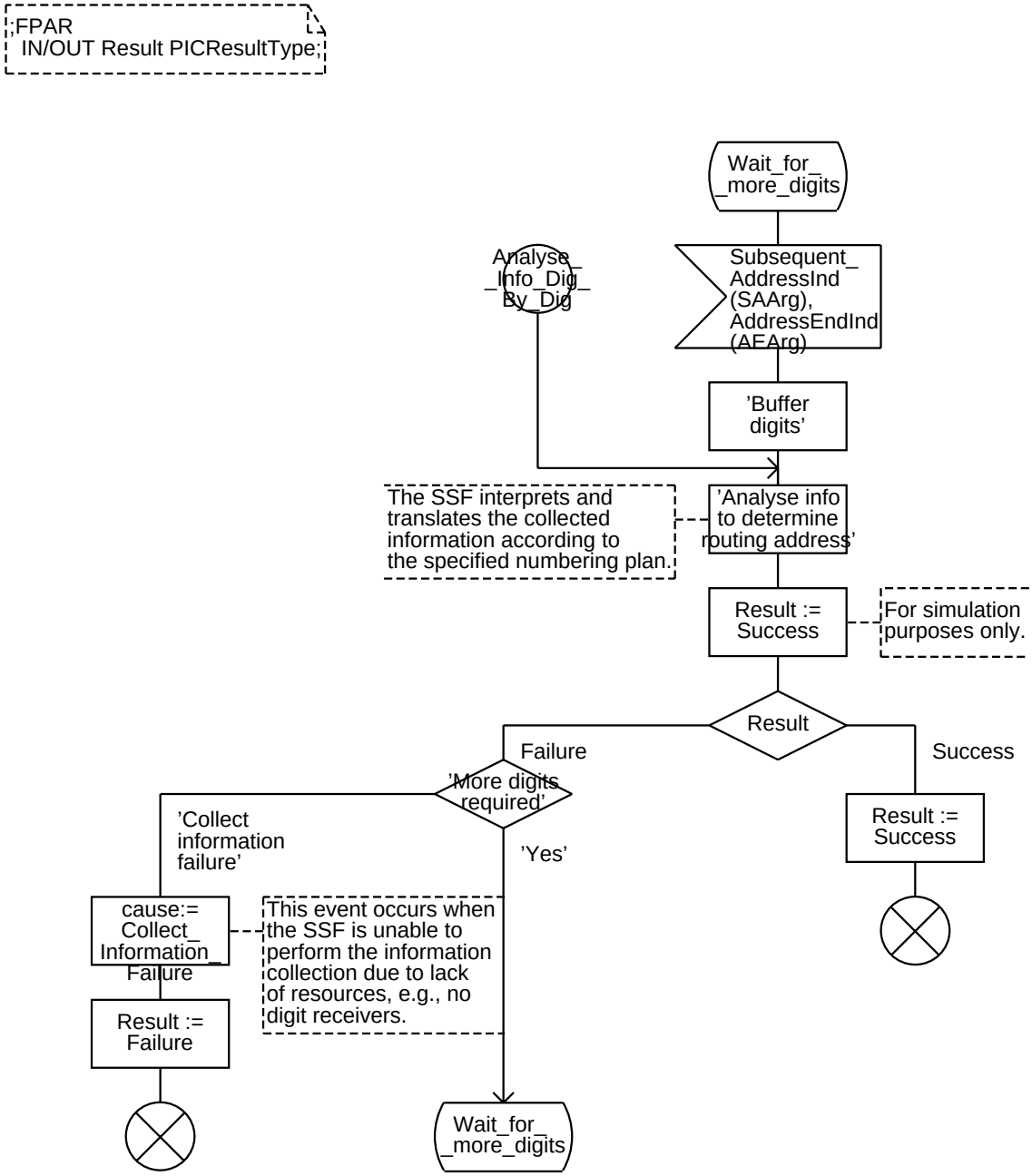
Annex A:

Procedure PIC_Analyse_Information 1(5)

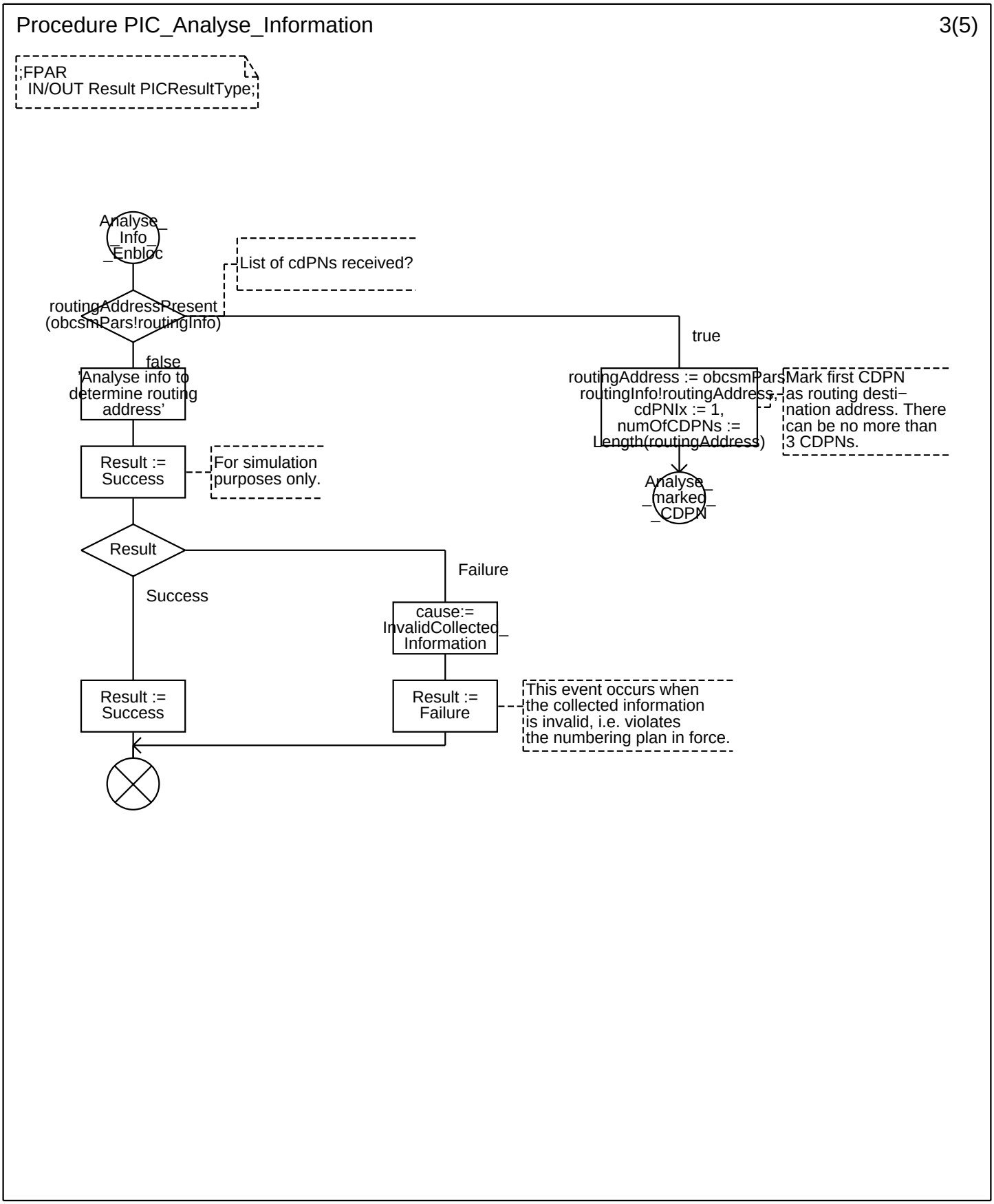


Annex A:

Procedure PIC_Analyse_Information 2(5)

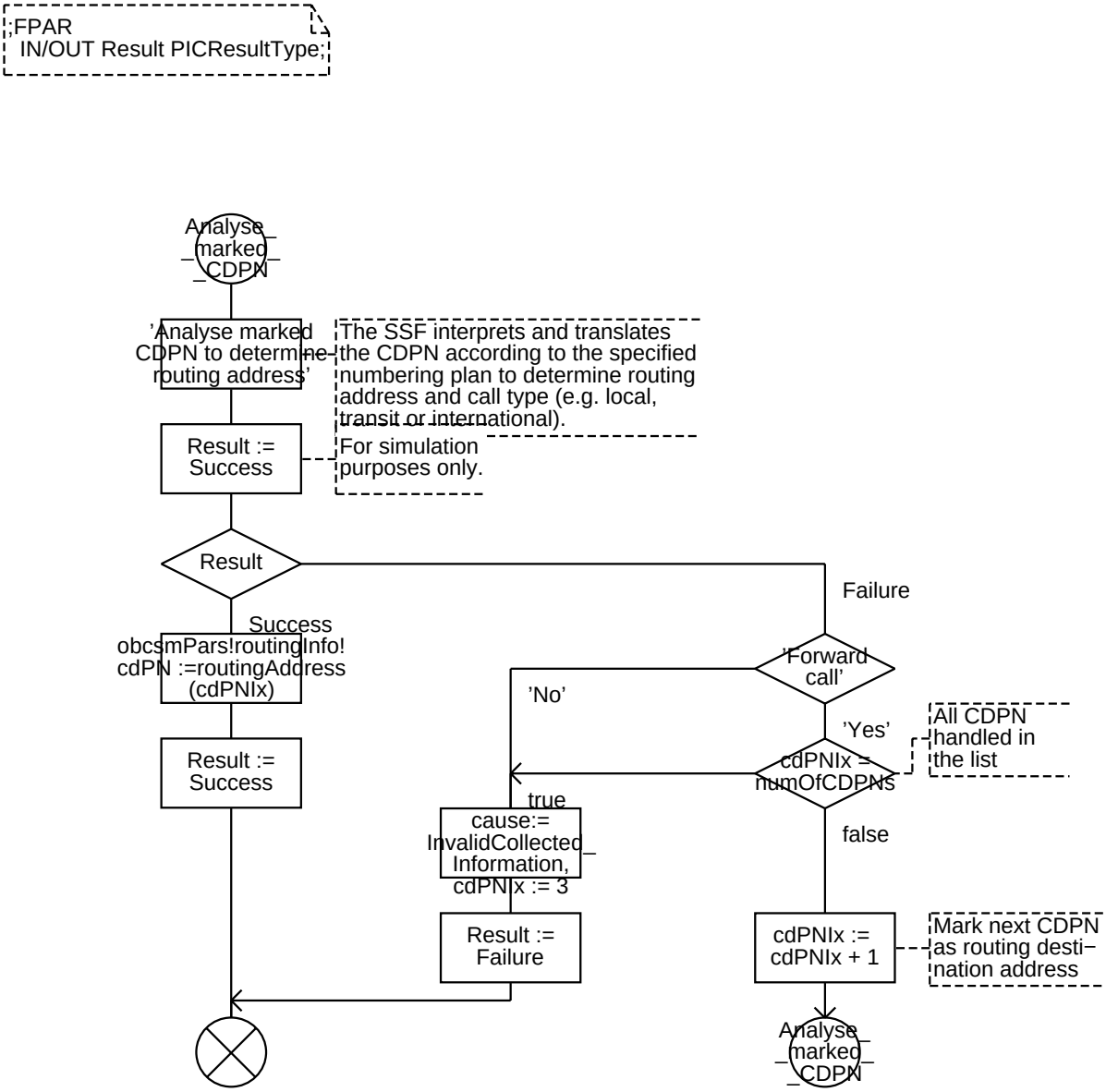


Annex A:


Procedure PIC_Analyse_Information
3(5)
;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.

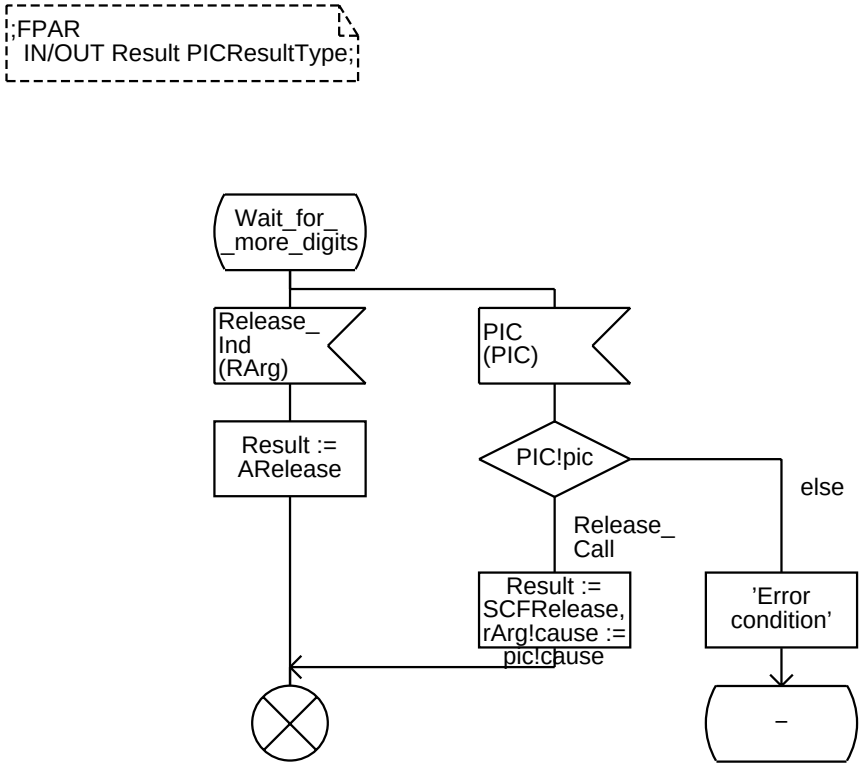
Annex A:

Procedure PIC_Analyse_Information 4(5)



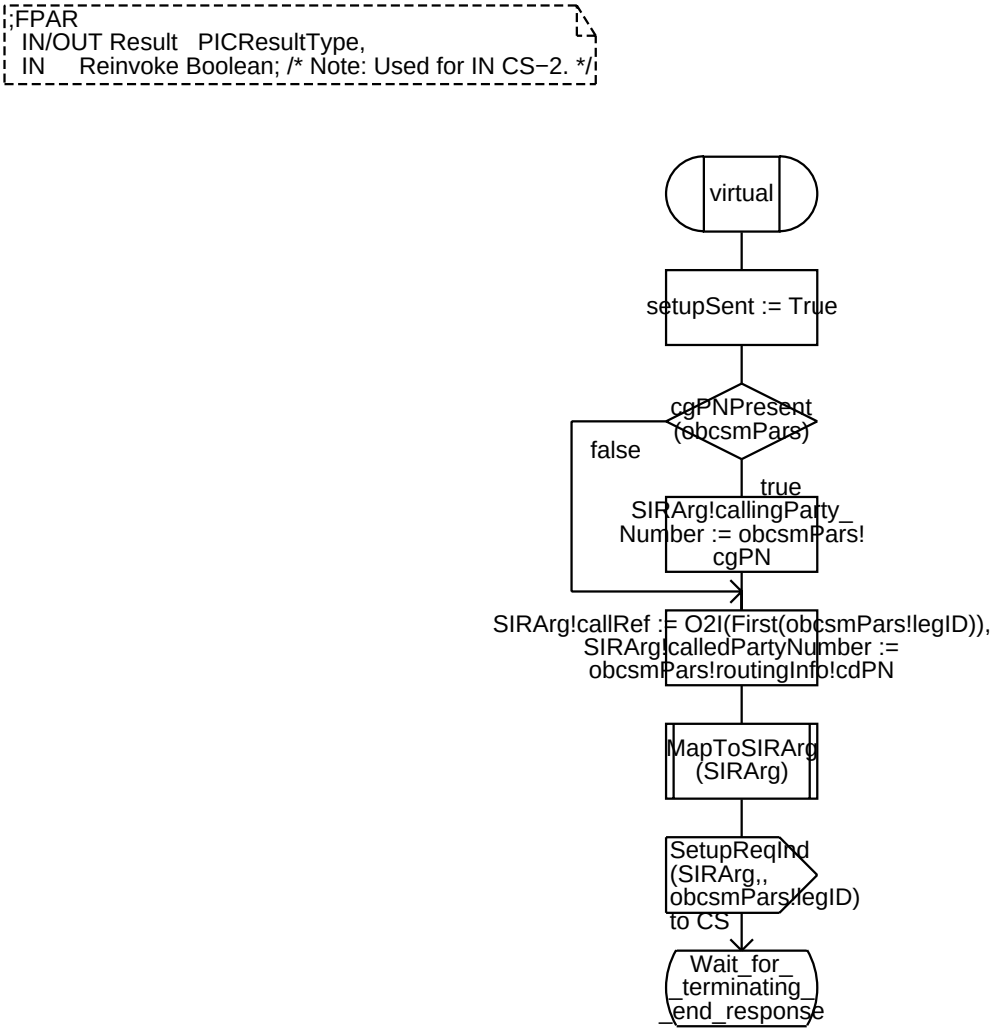
Annex A:

Procedure PIC_Analyse_Information 5(5)

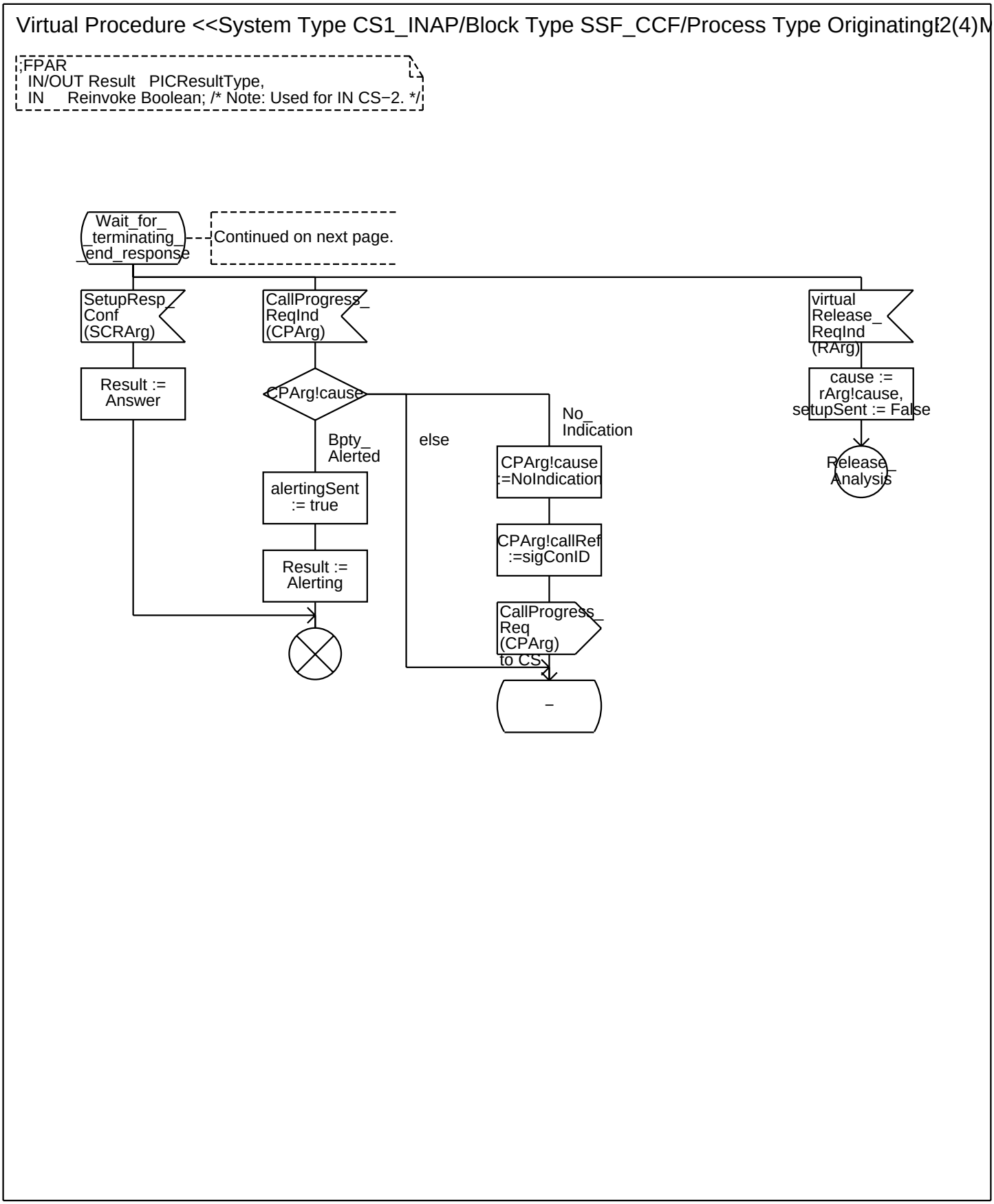


Annex A:

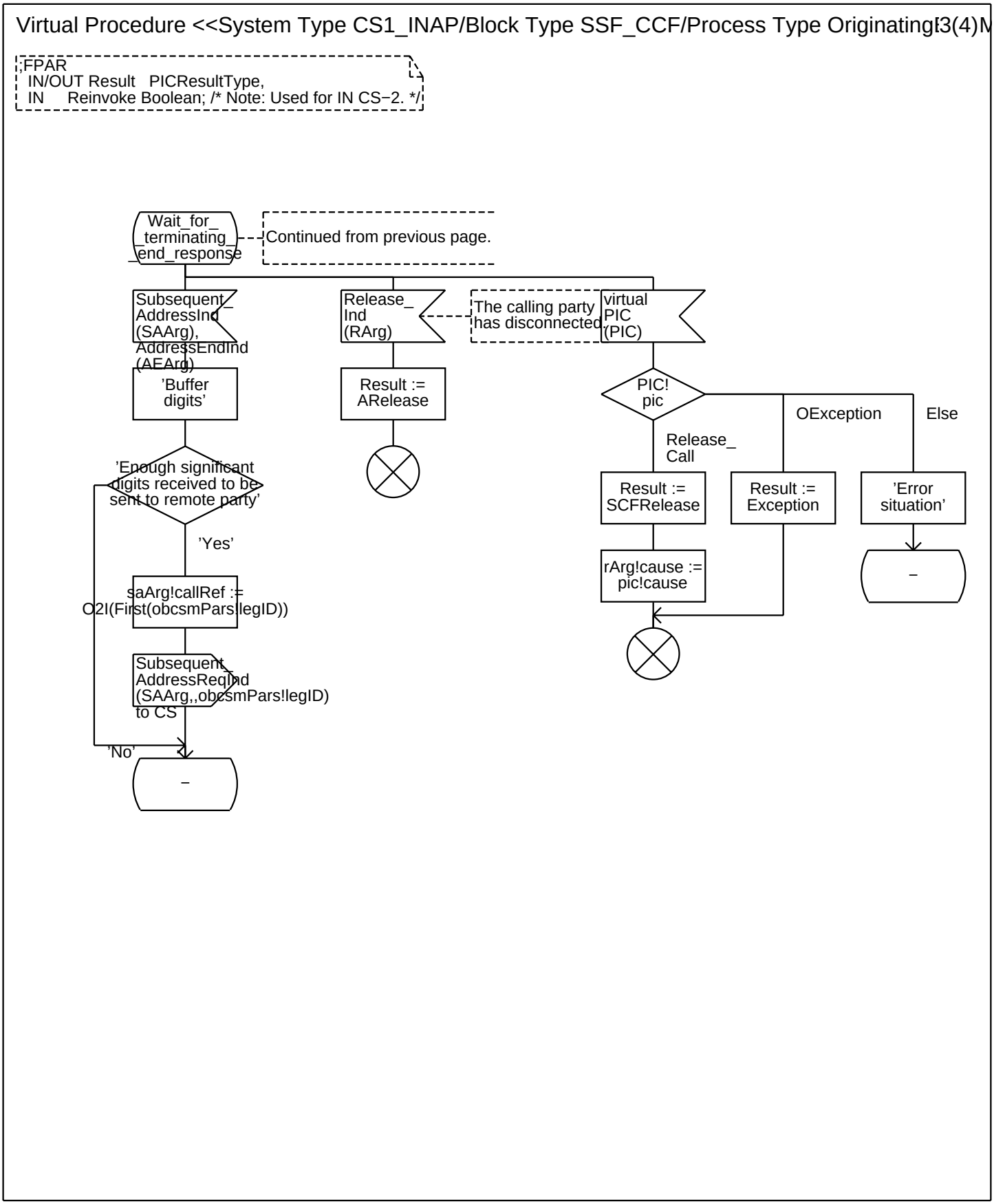
Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 1(4)M



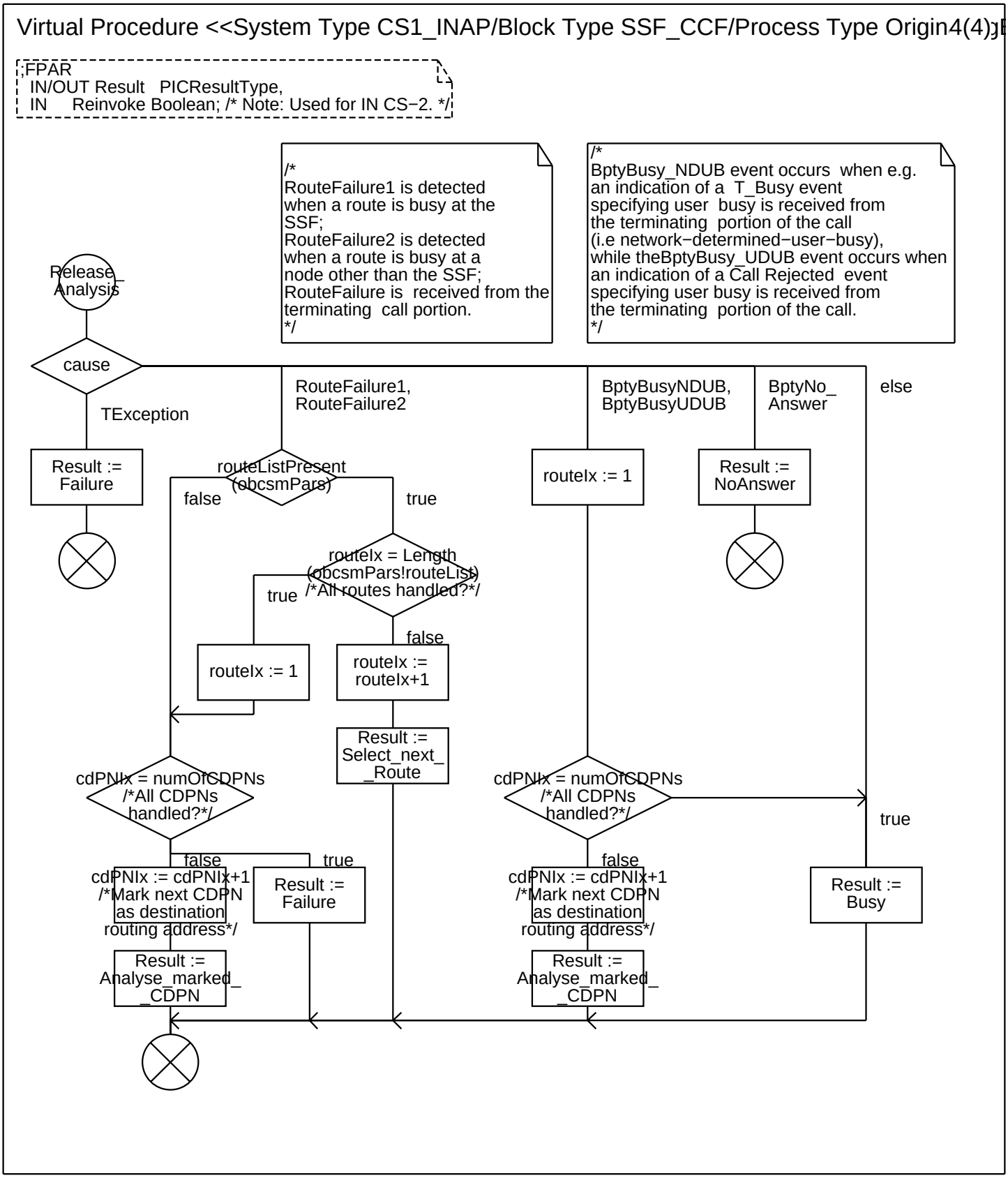
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating!2(4)M
;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)
virtual
Release_
ReqInd
(RArg)
Result :=
Answer
CPArg!cause
cause :=
rArg!cause,
setupSent := False
No_
Indication
Bpty_
Alerted
else
CPArg!cause
:=NoIndication
Release_
Analysis
alertingSent
:= true
CPArg!callRef
:=sigConID
Result :=
Alerting
CallProgress_
Req
(CPArg)
to CS
–

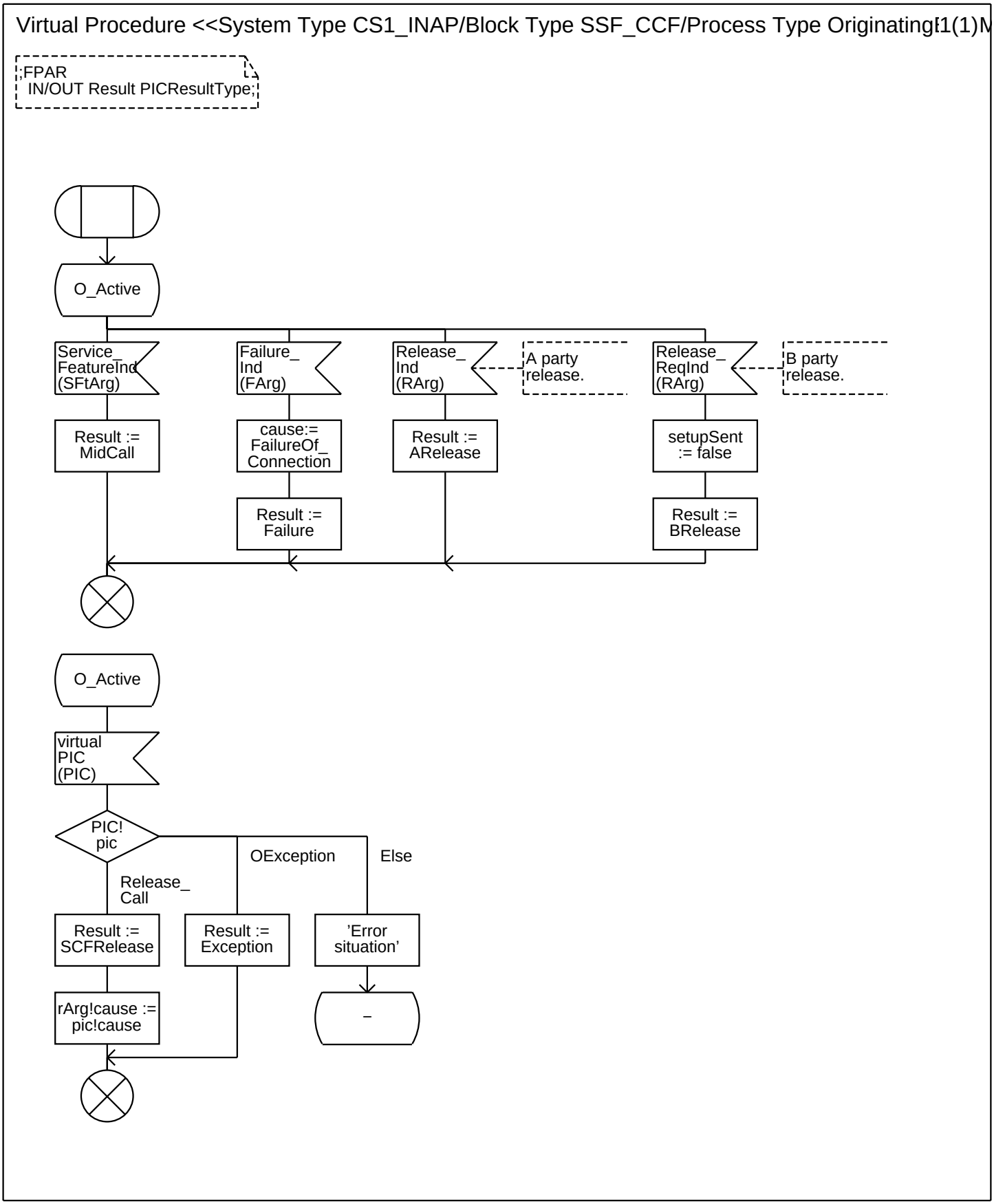
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 3(4)
;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 :=
O2I(First(obcsmPars!legID))
Subsequent_
AddressReqInd
(SAArg,,obcsmPars!legID)
to CS
'No'
–
–

Annex A:

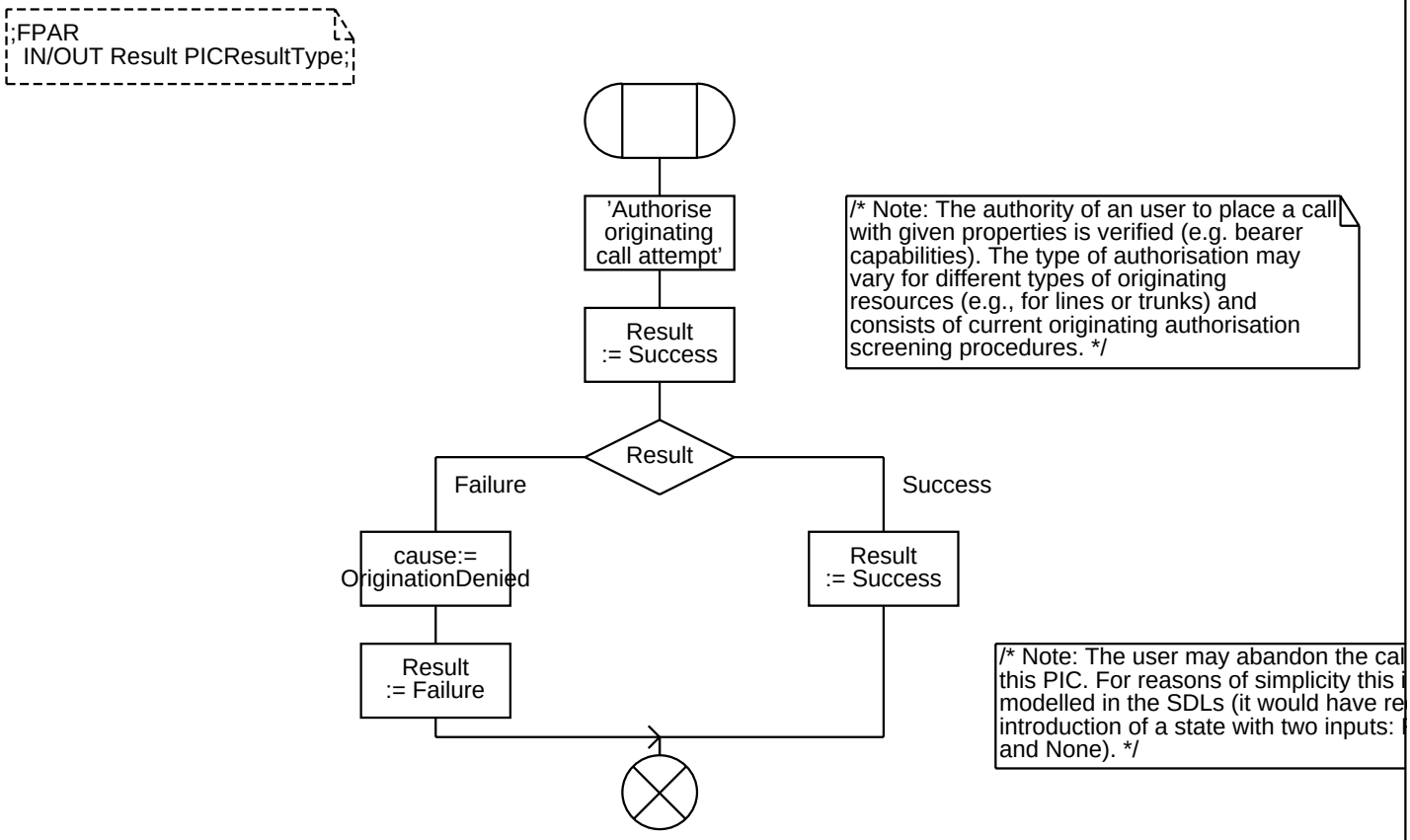

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Origin4(4)>>
;FPAR
IN/OUT Result PICResultType,
IN Reinvoke Boolean; /* Note: Used for IN CS−2. */
/*
RouteFailure1 is detected
when a route is busy at the
SSF;
RouteFailure2 is detected
when a route is busy at a
node other than 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
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Failure
false
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Busy
Result :=
Analyse_marked_
_CDPN
Result :=
Analyse_marked_
_CDPN

Annex A:

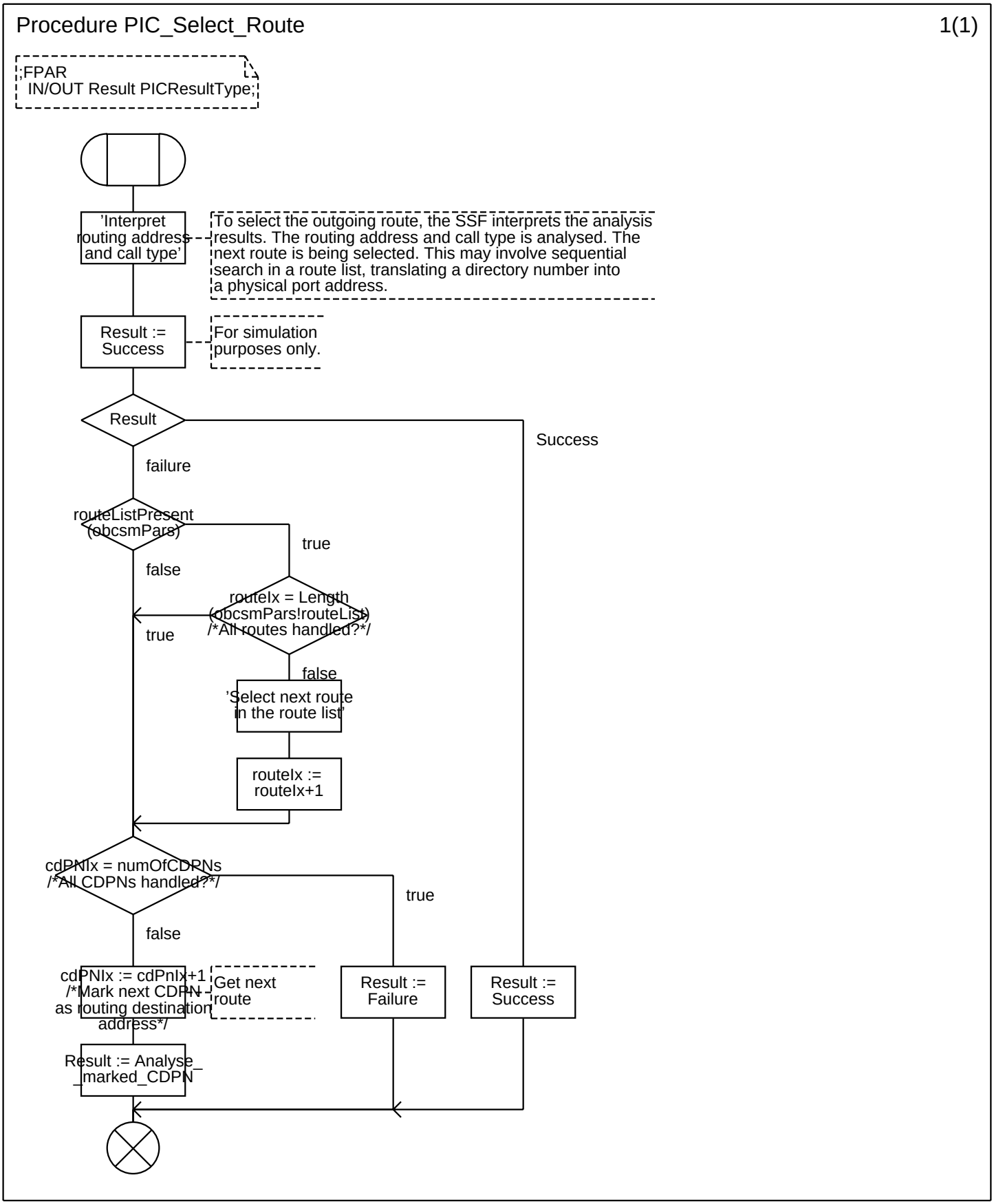

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
1(1)
;FPAR
IN/OUT Result PICResultType;
O_Active
Service_ FeatureInd (SFtArg)
Failure_ Ind (FArg)
Release_ Ind (RArg)
A party release.
Release_ ReqInd (RArg)
B party release.
Result := MidCall
cause:= FailureOf_ Connection
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
–

Annex A:

Procedure PIC_Authorise_Origination_Attempt 1(1)

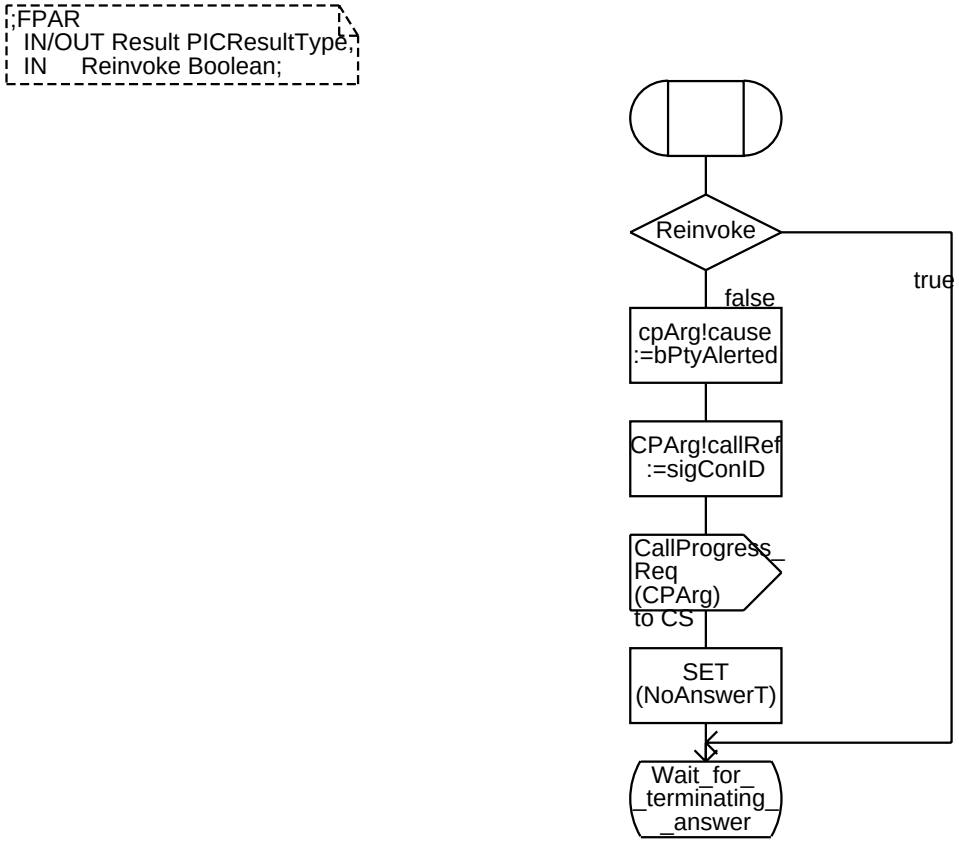


Annex A:

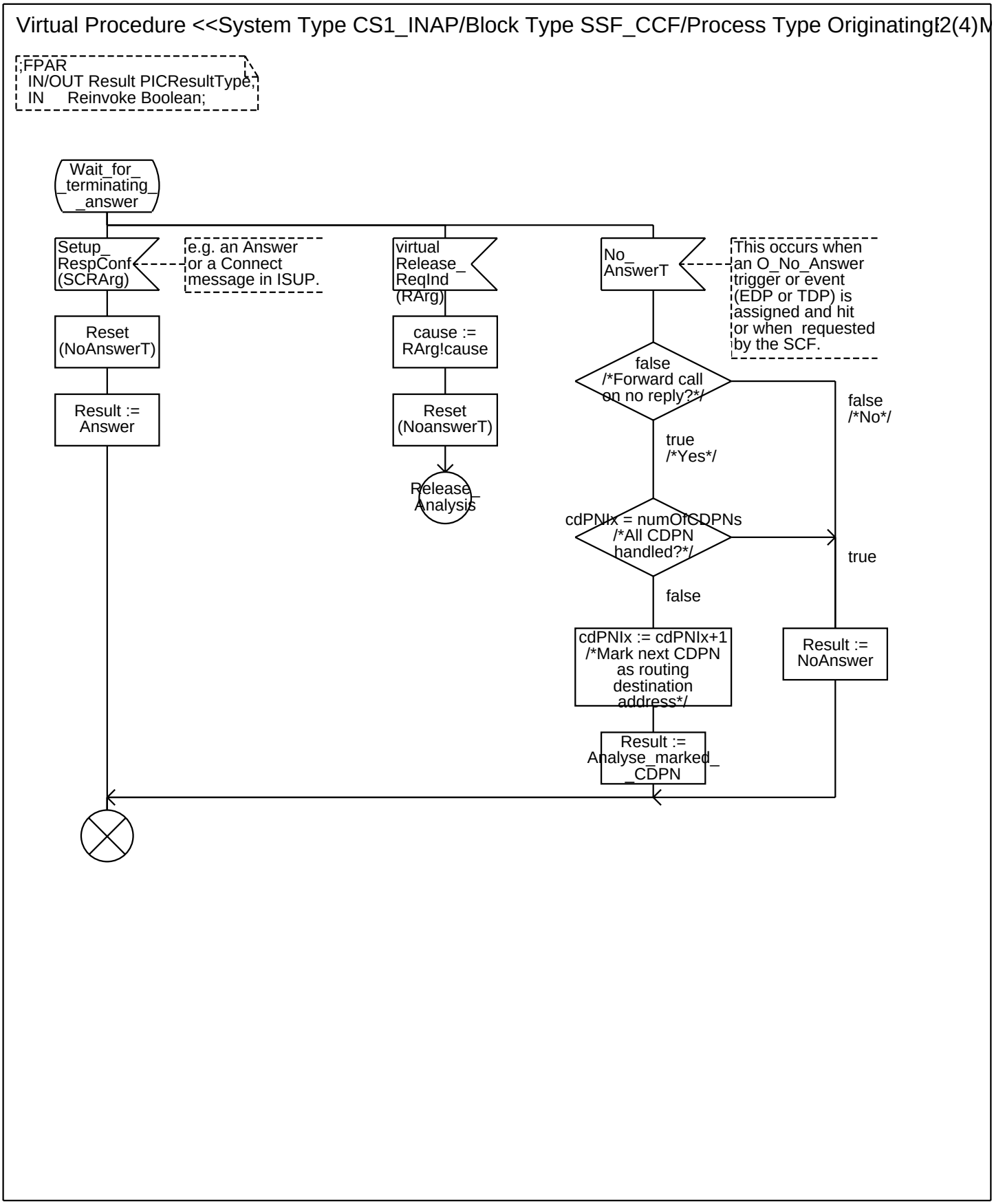

Procedure PIC_Select_Route
1(1)
;FPAR
IN/OUT Result PICResultType;
'Interpret routing address and call type'
To select the outgoing route, the SSF interprets the analysis results. The routing address and call type is analysed. The next route is being selected. This may involve sequential search in a 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

Annex A:

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 1(4)M

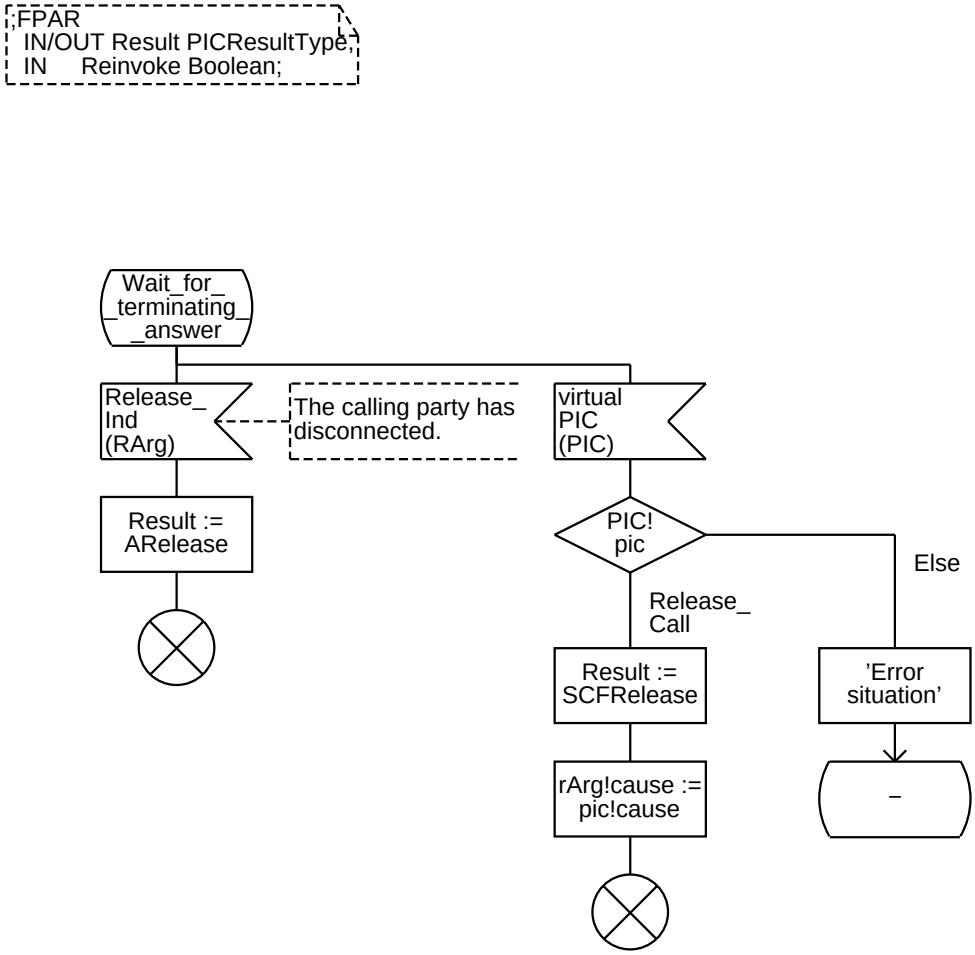


Annex A:

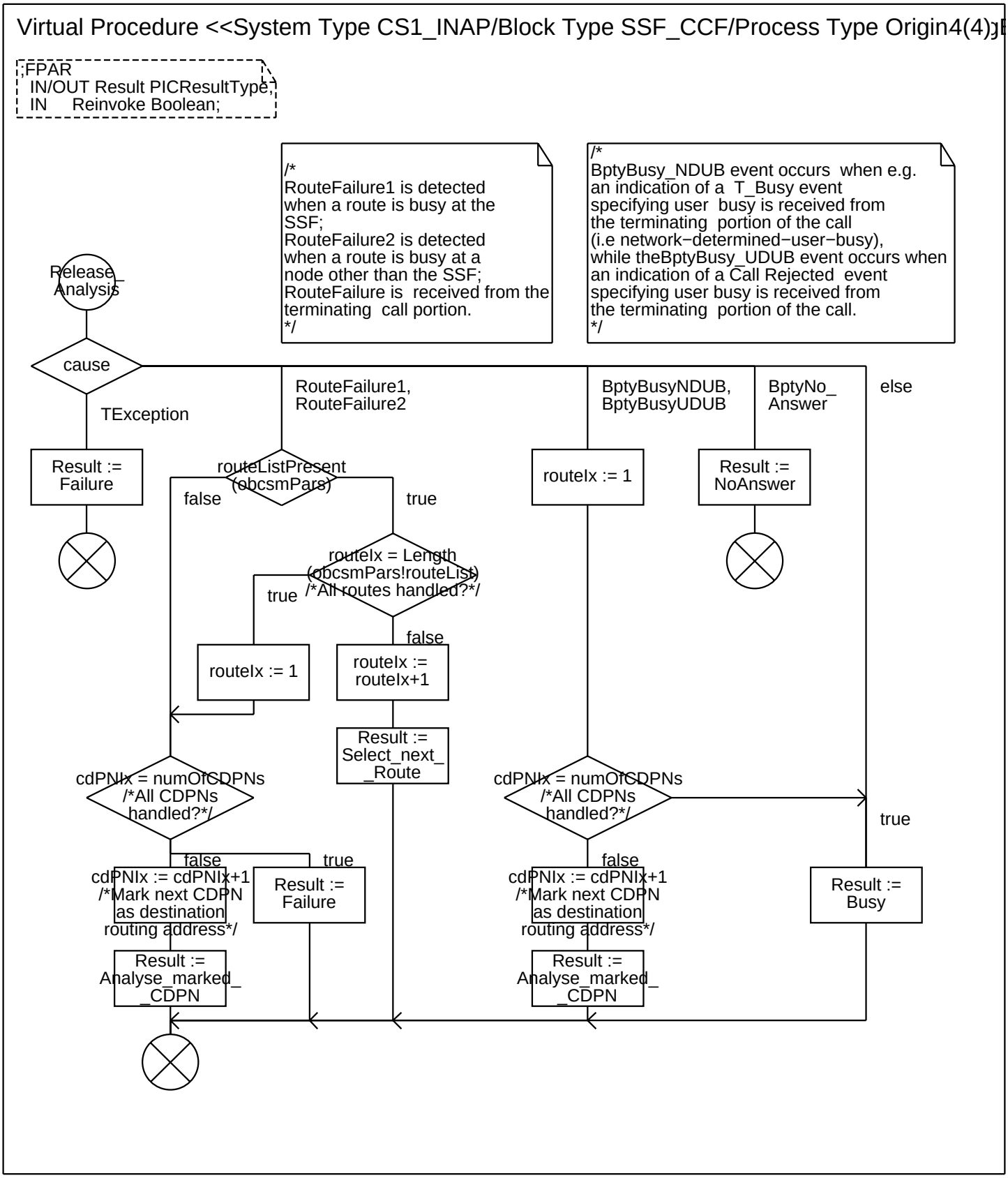

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
2(4)M
;FPAR
IN/OUT Result PICResultType;
IN Reinvoke Boolean;
Wait_for_
terminating
_answer
Setup_
RespConf
(SCRArg)
e.g. an Answer
or a Connect
message in ISUP.
virtual
Release_
ReqInd
(RArg)
No_
AnswerT
This occurs 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

Annex A:

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 3(4)M

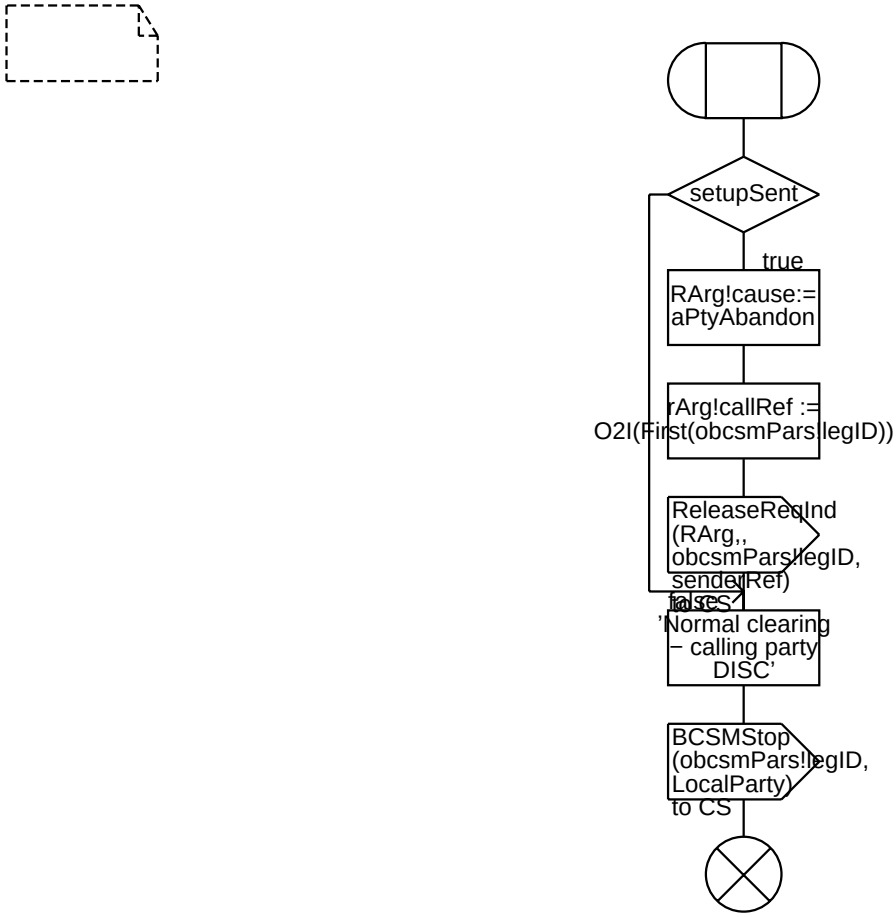


Annex A:

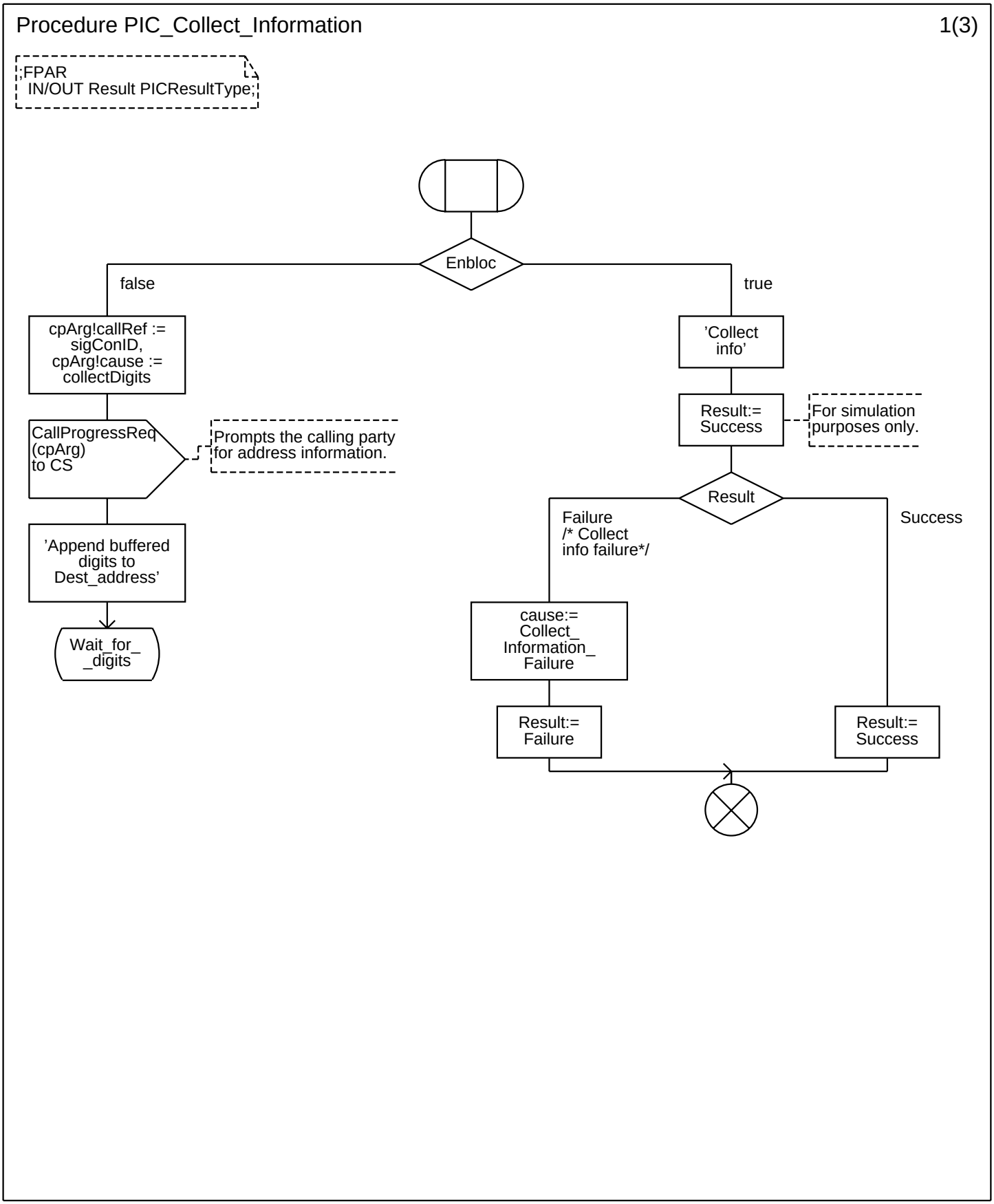

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Origin4(4)>>
;FPAR
IN/OUT Result PICResultType;
IN Reinvoke Boolean;
/*
RouteFailure1 is detected
when a route is busy at the
SSF;
RouteFailure2 is detected
when a route is busy at a
node other than the SSF;
RouteFailure is received 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
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Failure
false
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Busy
Result :=
Analyse_marked_
_CDPN
Result :=
Analyse_marked_
_CDPN

Annex A:

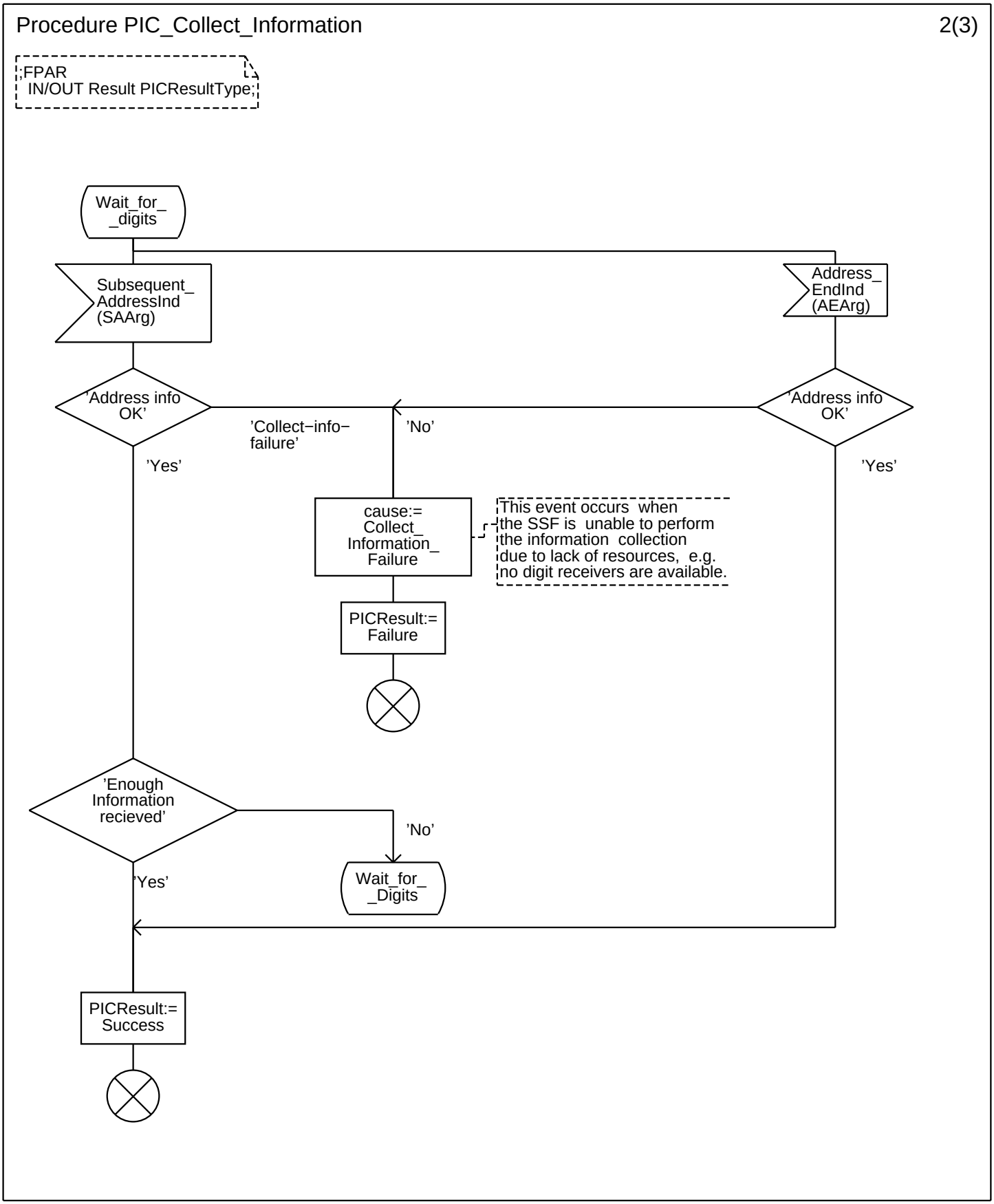
Procedure PIC_O_Abandon 1(1)



Annex A:

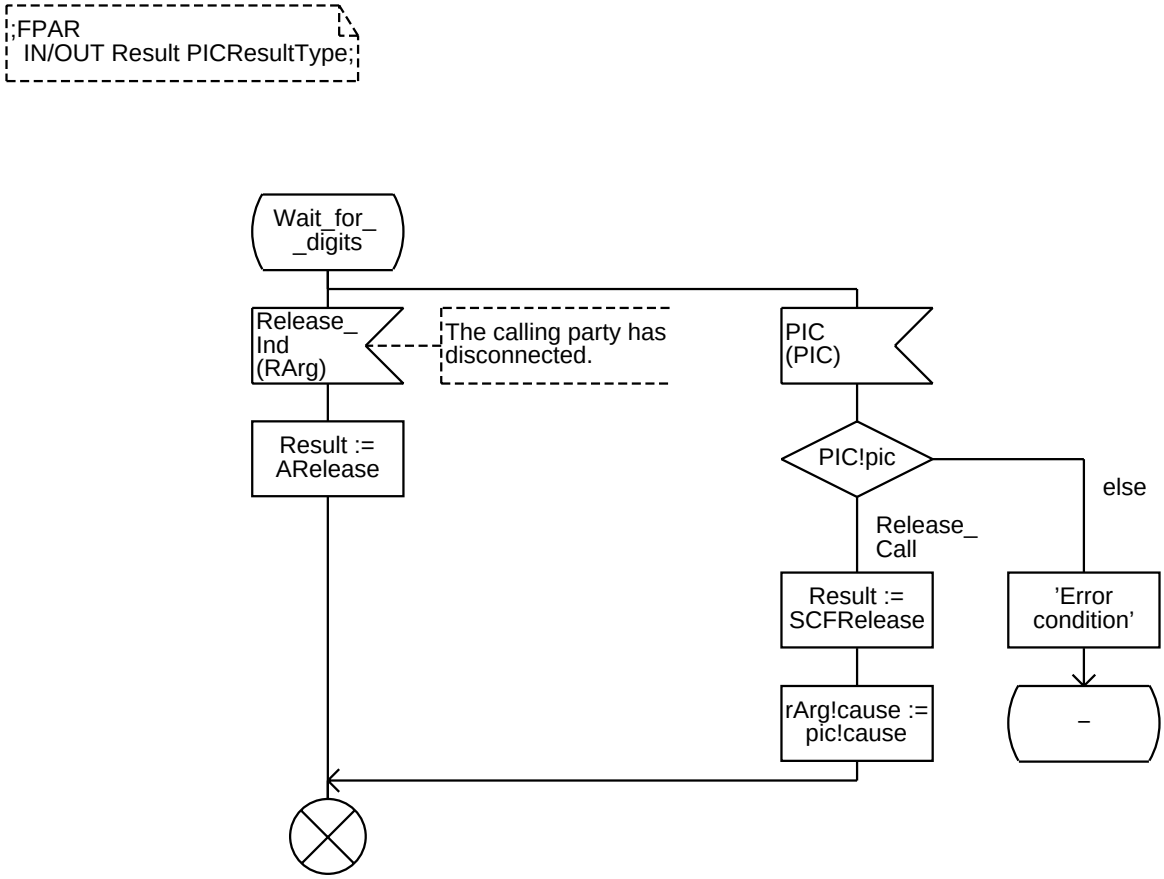

Procedure PIC_Collect_Information
1(3)
;FPAR
IN/OUT Result PICResultType;
Enbloc
false
true
cpArg!callRef :=
sigConID,
cpArg!cause :=
collectDigits
CallProgressReq
(cpArg)
to CS
Prompts the calling party
for address information.
'Append buffered
digits to
Dest_address'
Wait_for_
_digits
'Collect
info'
Result:=
Success
For simulation
purposes only.
Result
Failure
/* Collect
info failure*/
Success
cause:=
Collect_
Information_
Failure
Result:=
Failure
Result:=
Success

Annex A:

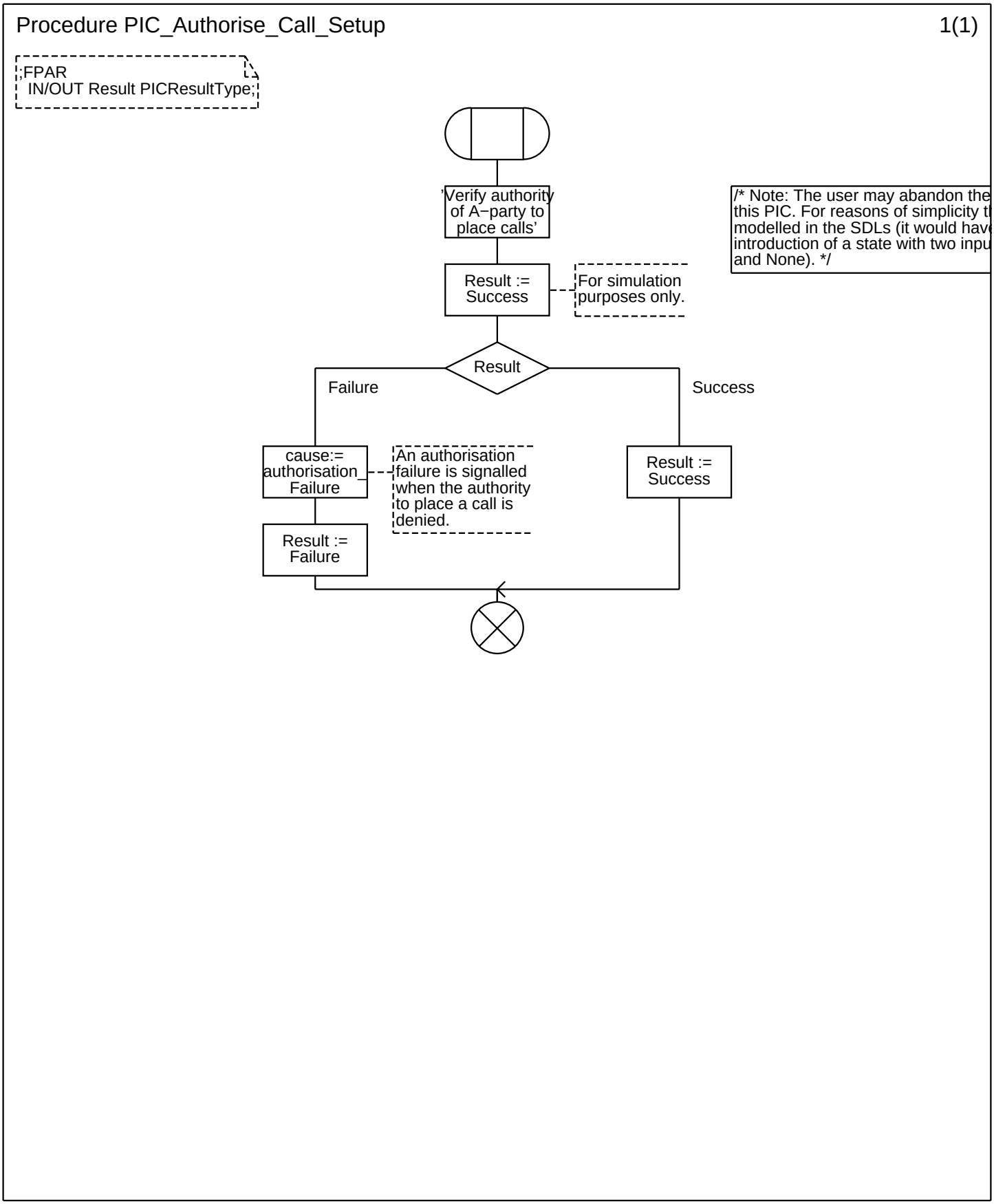

Procedure PIC_Collect_Information
2(3)
;FPAR
IN/OUT Result PICResultType;
Wait_for_
_digits
Subsequent_
AddressInd
(SAArg)
Address_
EndInd
(AEArg)
'Address info
OK'
'Address info
OK'
'Collect−info−
failure'
'No'
'Yes'
'Yes'
cause:=
Collect_
Information_
Failure
This event occurs when
the SSF is unable 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

Annex A:

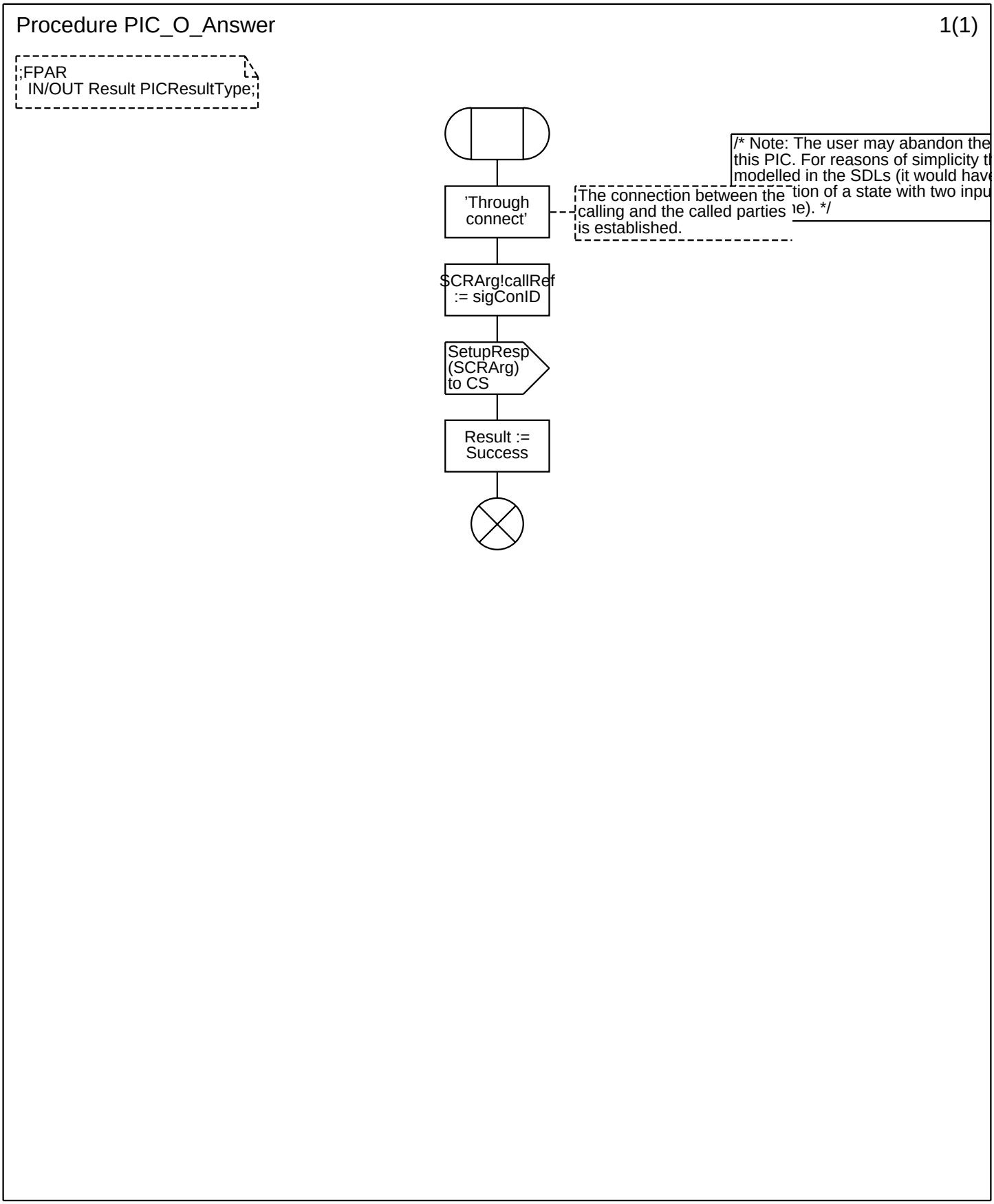
Procedure PIC_Collect_Information 3(3)



Annex A:

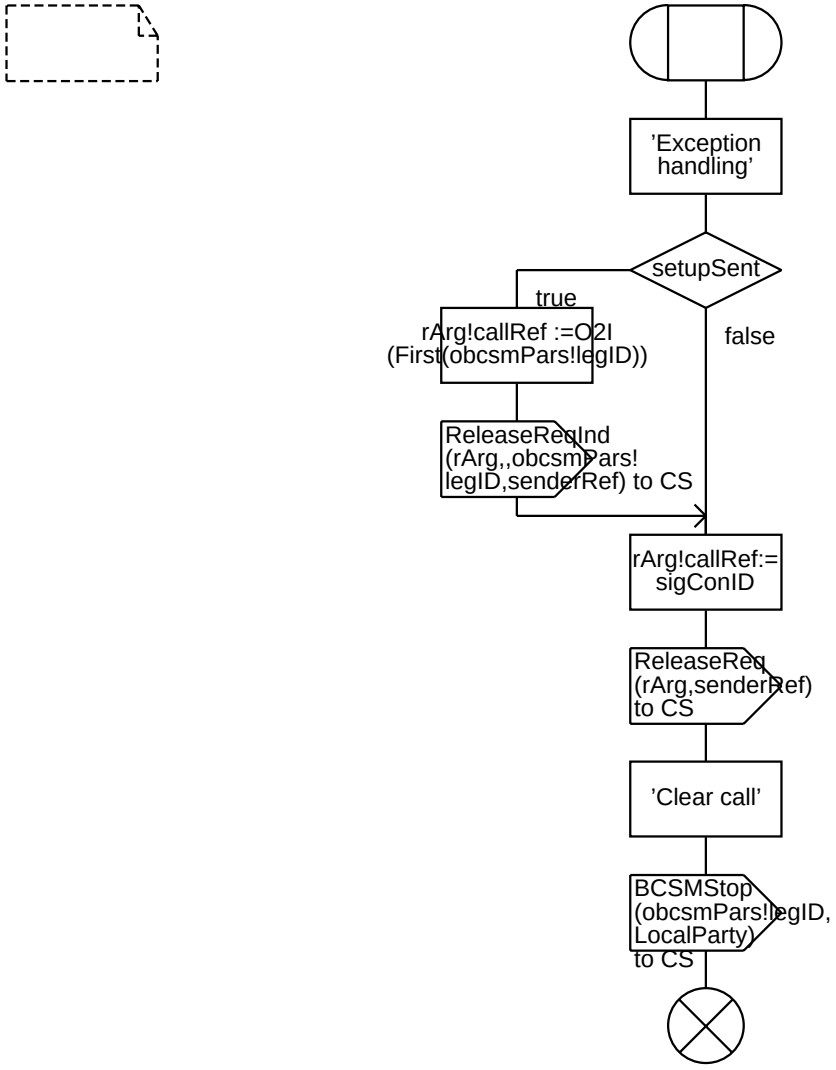

Procedure PIC_Authorise_Call_Setup
1(1)
;FPAR
IN/OUT Result PICResultType;
'Verify authority of A−party to place calls'
Result := Success
For simulation purposes only.
/* Note: The user may abandon the
this PIC. For reasons of simplicity t
modelled in the SDLs (it would have
introduction of a state with two inpu
and None). */
Result
Failure
Success
cause:= authorisation_ Failure
An authorisation failure is signalled when the authority to place a call is denied.
Result := Failure
Result := Success

Annex A:


Procedure PIC_O_Answer
1(1)
;FPAR
IN/OUT Result PICResultType;
'Through connect'
The connection between the calling and the called parties is established.
/* Note: The user may abandon the
this PIC. For reasons of simplicity t
modelled in the SDLs (it would have
tion of a state with two inpu
ne). */
SCRArg!callRef := sigConID
SetupResp (SCRArg) to CS
Result := Success

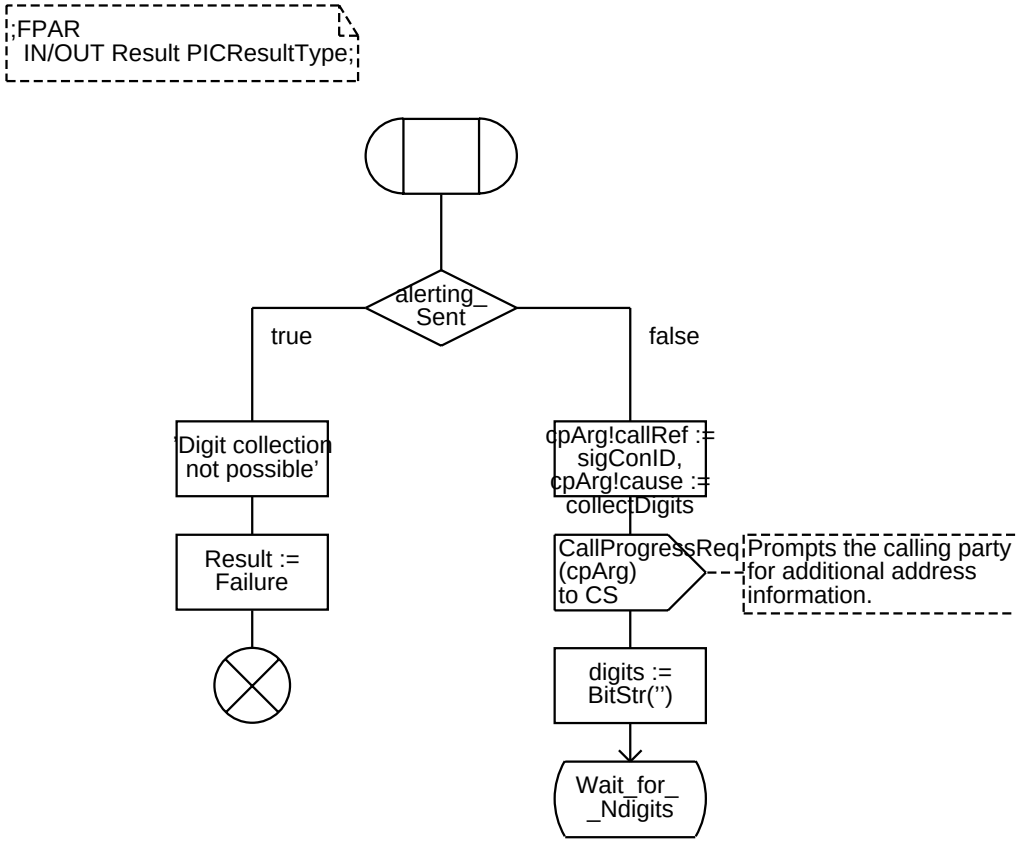
Annex A:

Procedure PIC_OException 1(1)

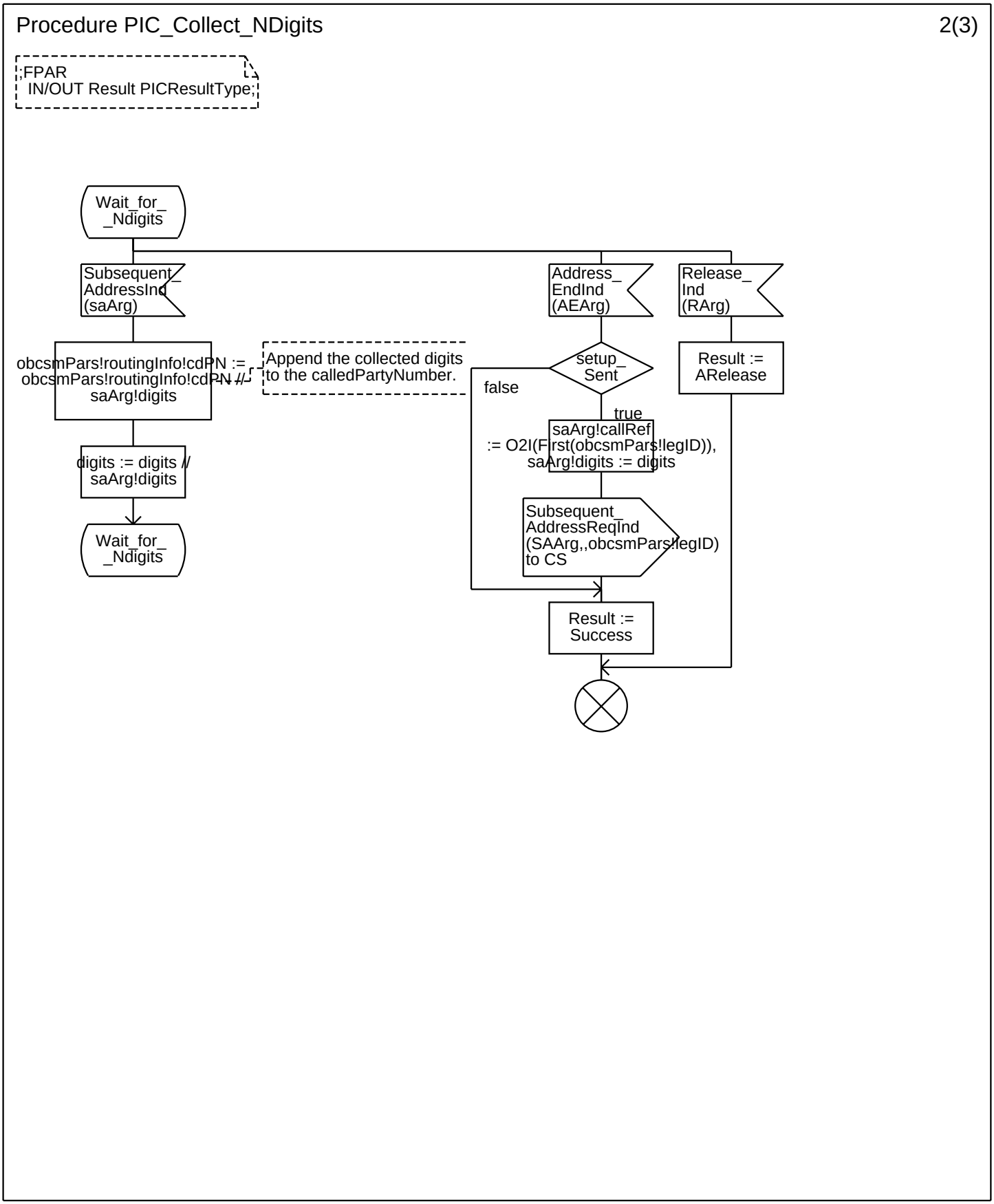


Annex A:

Procedure PIC_Collect_NDigits 1(3)

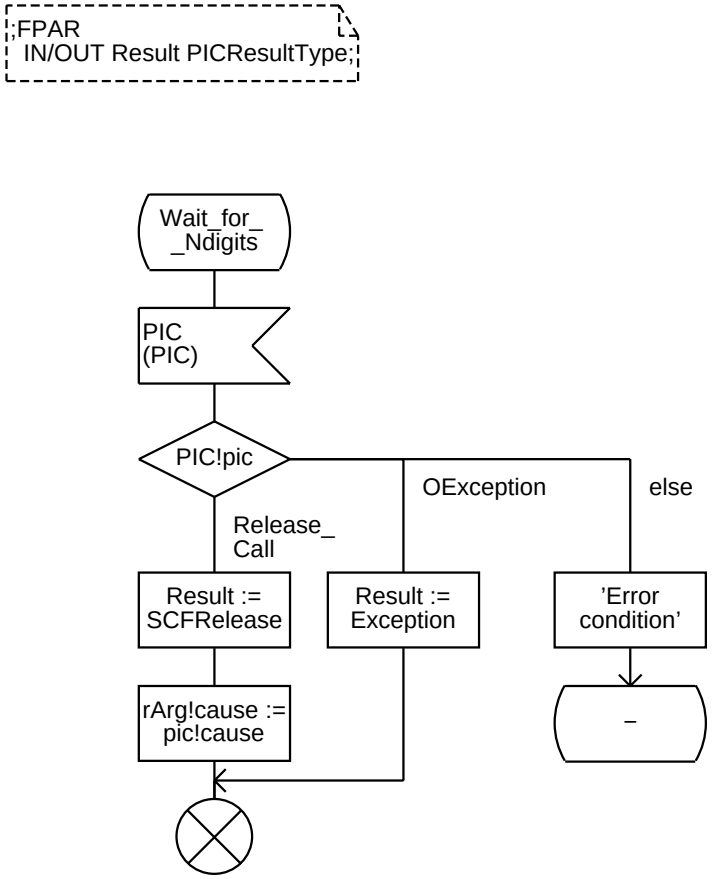


Annex A:

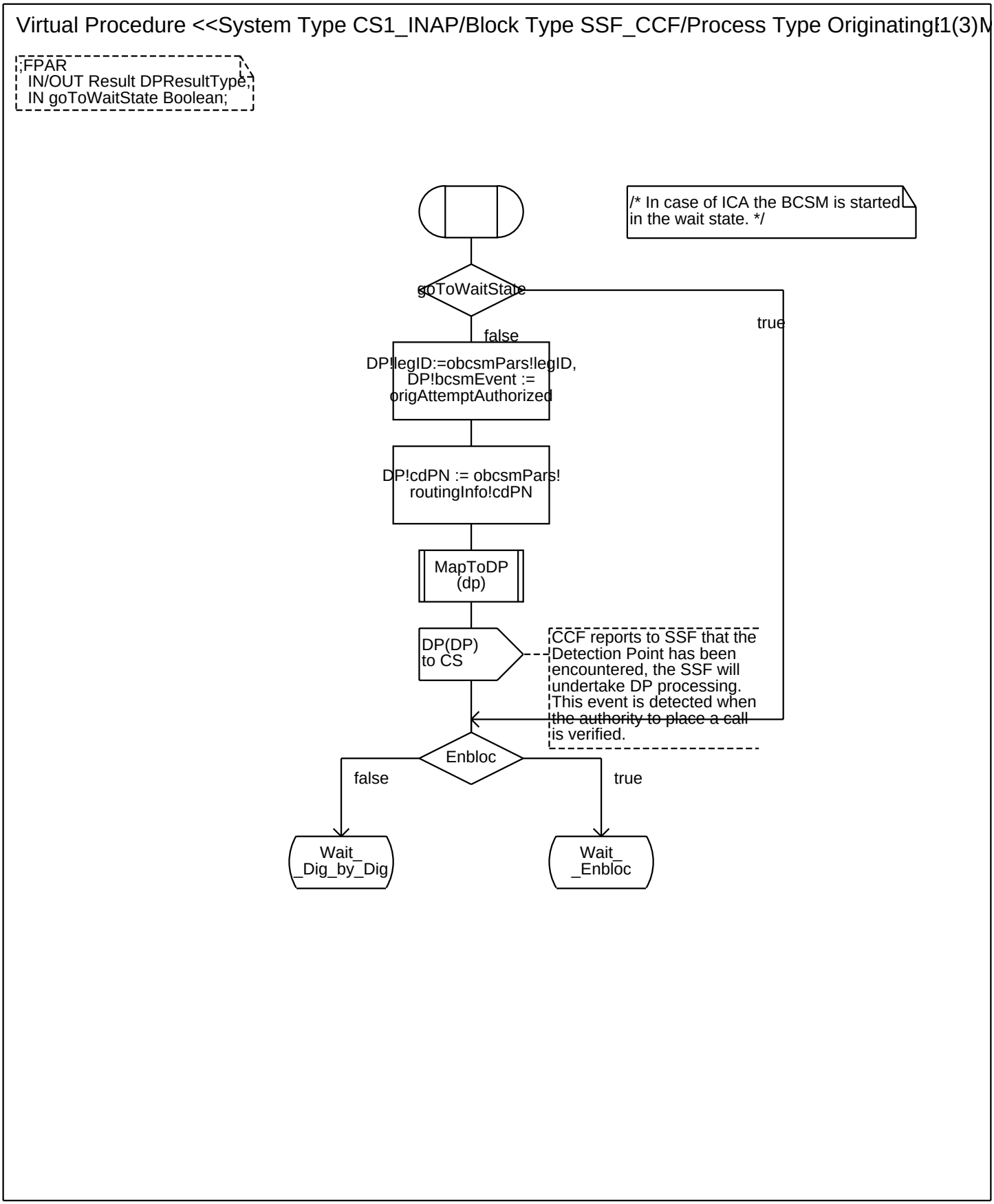

Procedure PIC_Collect_NDigits
2(3)
;FPAR
IN/OUT Result PICResultType;
Wait_for_
_Ndigits
Subsequent_
AddressInd
(saArg)
obcsmPars!routingInfo!cdPN :=
obcsmPars!routingInfo!cdPN //
saArg!digits
Append the collected digits
to the calledPartyNumber.
digits := digits //
saArg!digits
Wait_for_
_Ndigits
Address_
EndInd
(AEArg)
setup_
Sent
false
true
saArg!callRef
:= O2I(First(obcsmPars!legID)),
saArg!digits := digits
Subsequent_
AddressReqInd
(SAArg,,obcsmPars!legID)
to CS
Release_
Ind
(RArg)
Result :=
ARelease
Result :=
Success

Annex A:

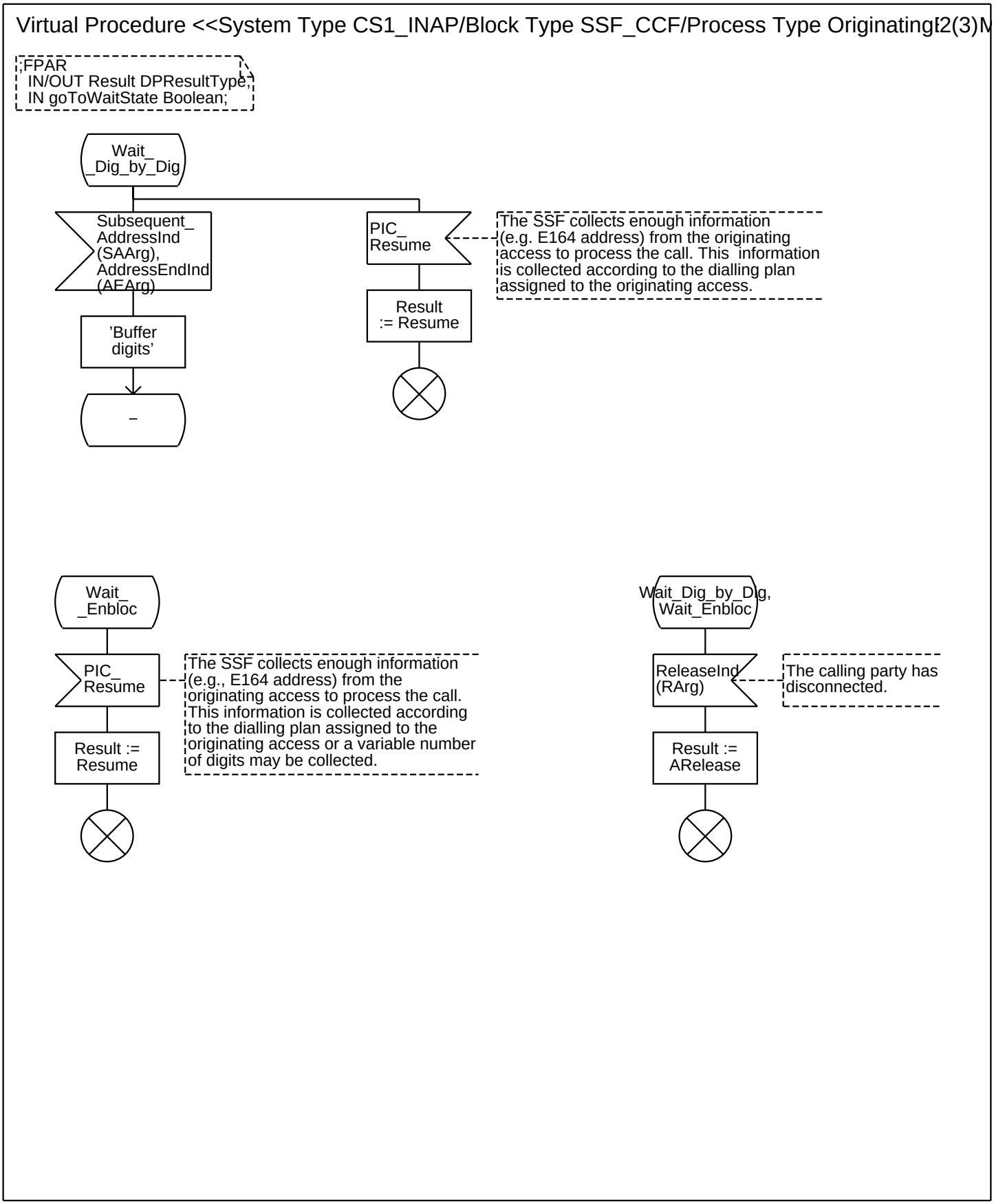
Procedure PIC_Collect_NDigits 3(3)



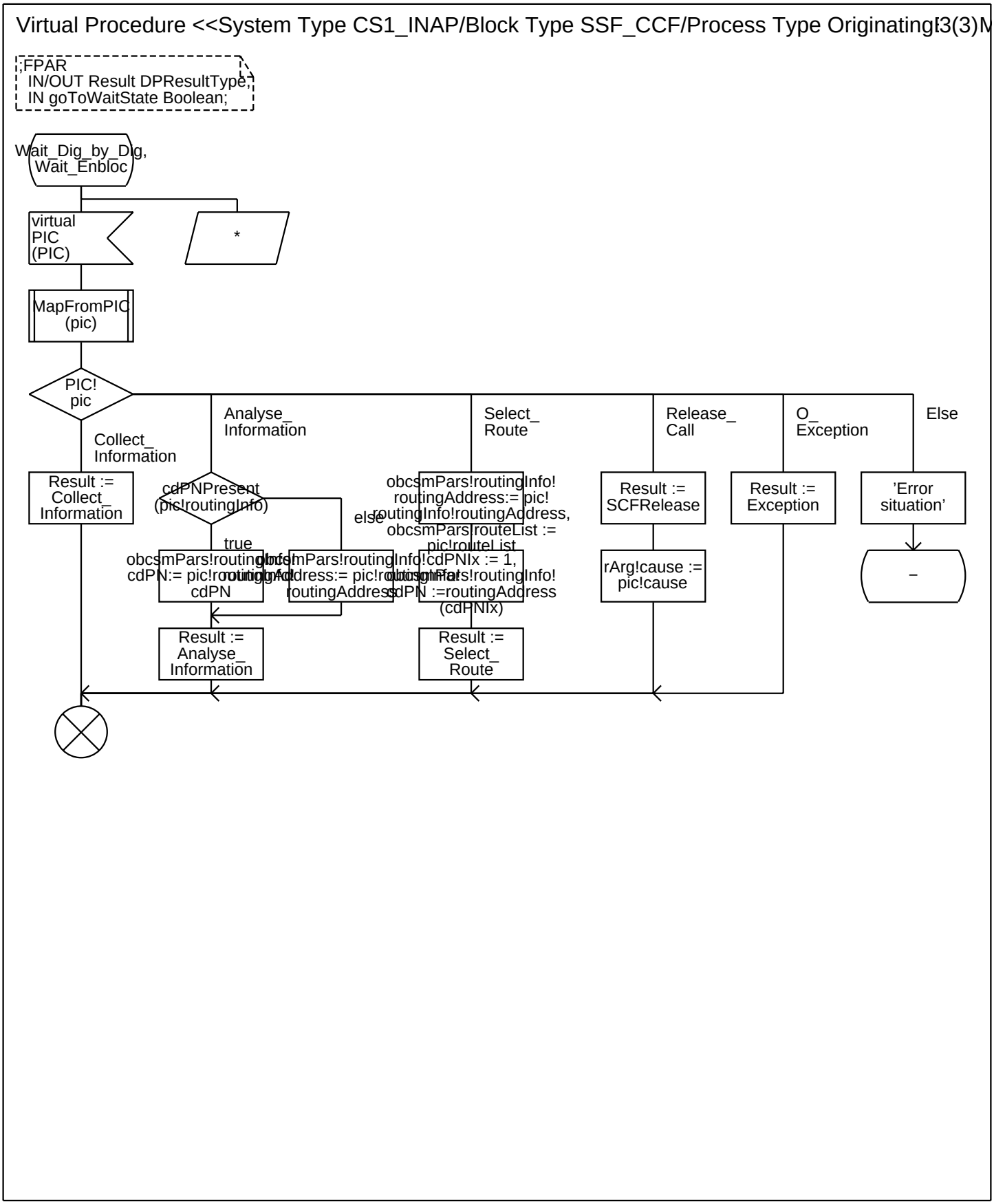
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 1(3)M
;FPAR
IN/OUT Result DPResultType;
IN goToWaitState Boolean;
/* In case of ICA the BCSM is started in the wait state. */
goToWaitState
true
false
DP!legID:=obcsmPars!legID, DP!bcsmEvent := origAttemptAuthorized
DP!cdPN := obcsmPars! routingInfo!cdPN
MapToDP (dp)
DP(DP) to CS
CCF reports to SSF that the Detection Point has been encountered, the SSF will undertake DP processing. This event is detected when the authority to place a call is verified.
Enbloc
false
true
Wait_ _Dig_by_Dig
Wait_ _Enbloc

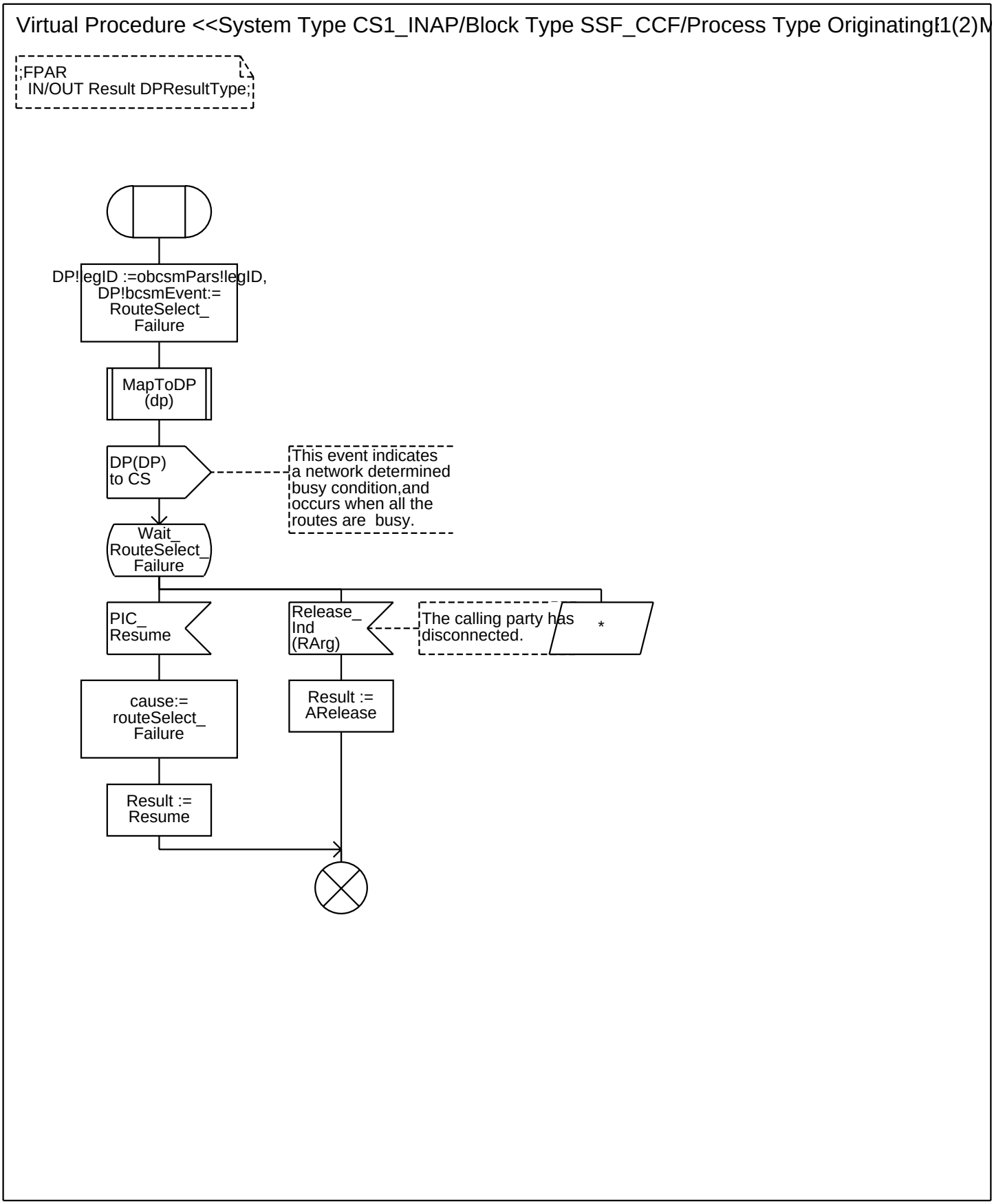
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
2(3)M
;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

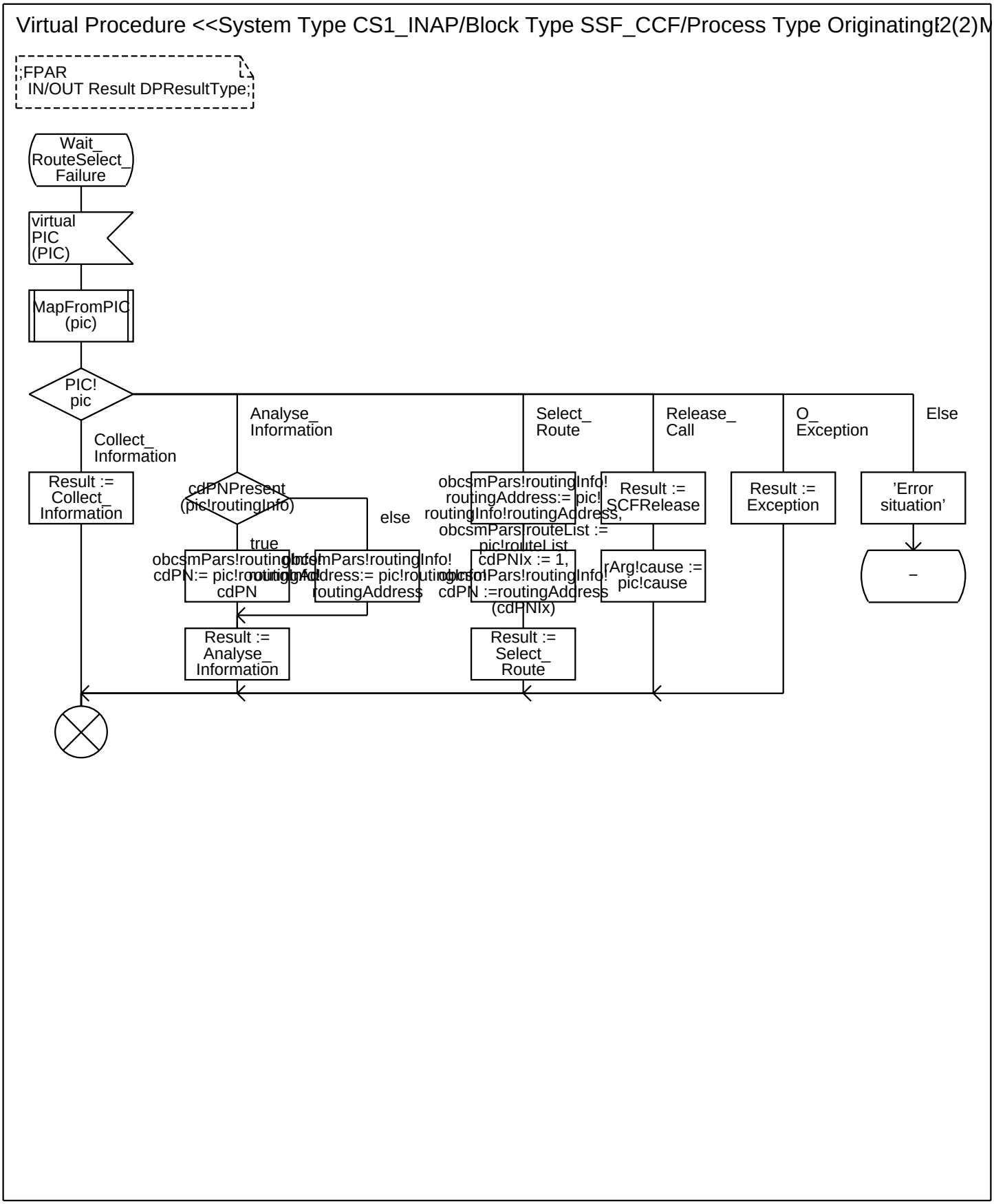
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
;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
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
obcsmPars!routingInfo!cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Analyse_
Information
Result :=
Select_
Route
Result :=
SCFRelease
rArg!cause :=
pic!cause
Result :=
Exception
'Error
situation'
–

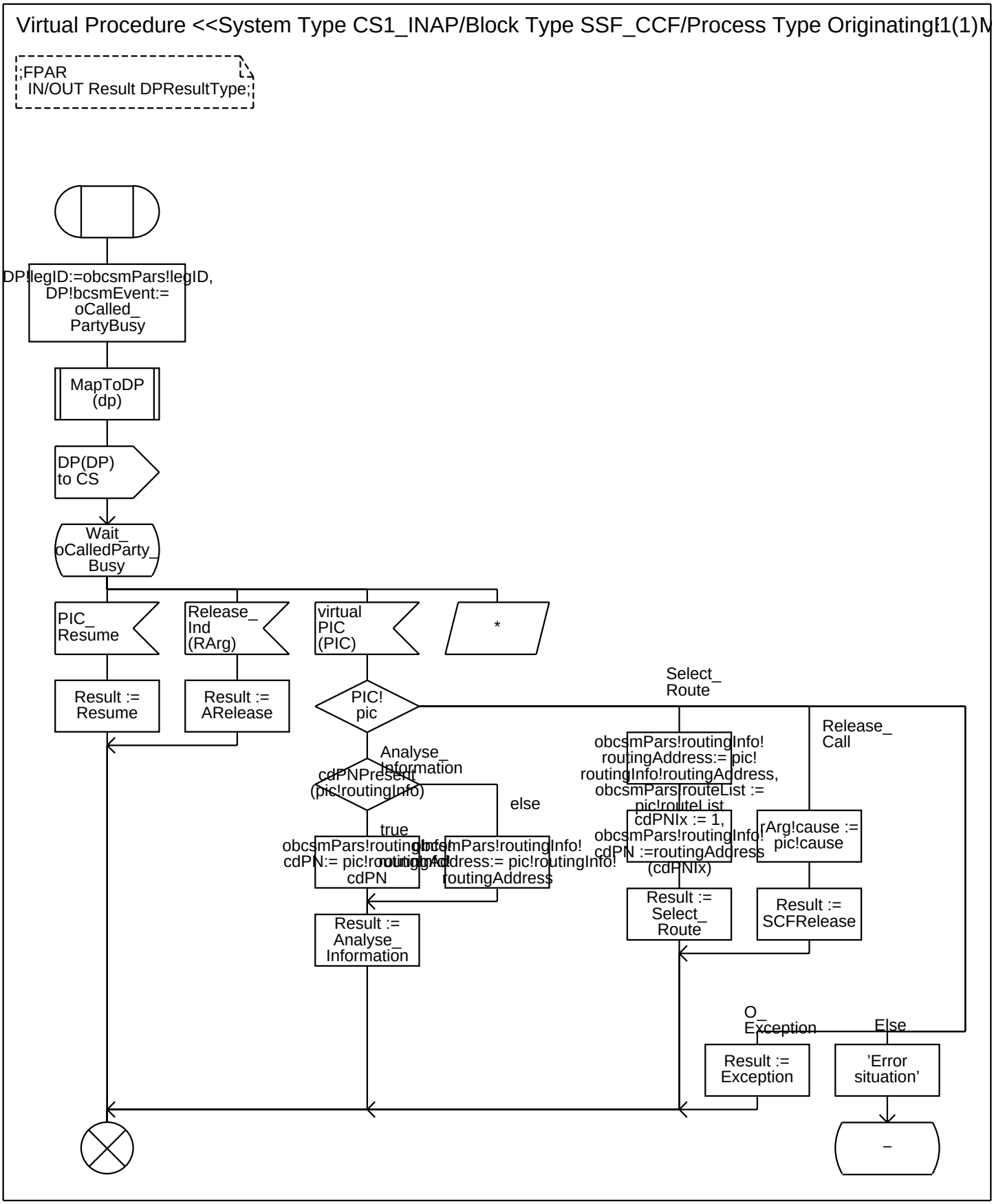
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 1(2)M
;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

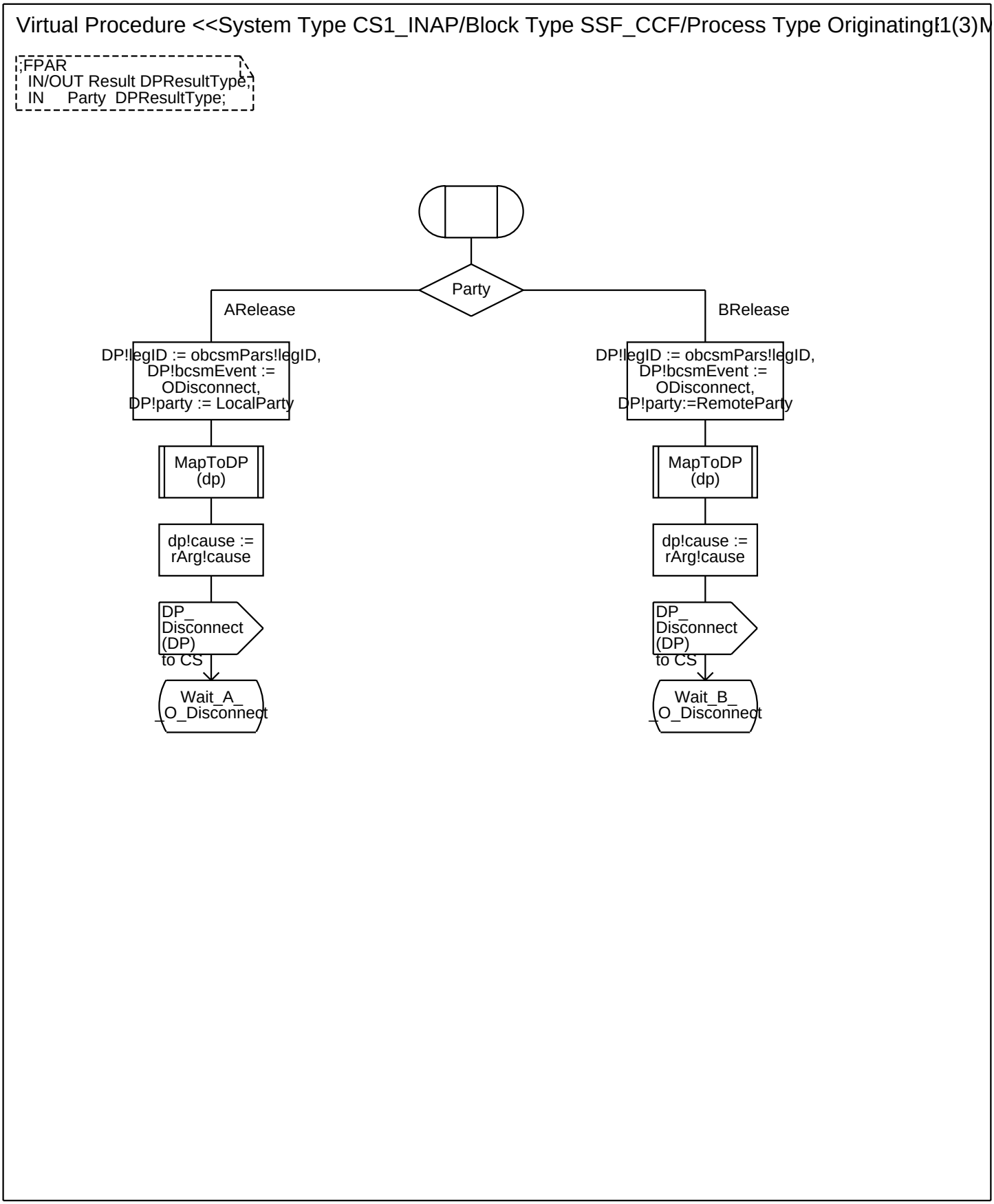
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
;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'
–

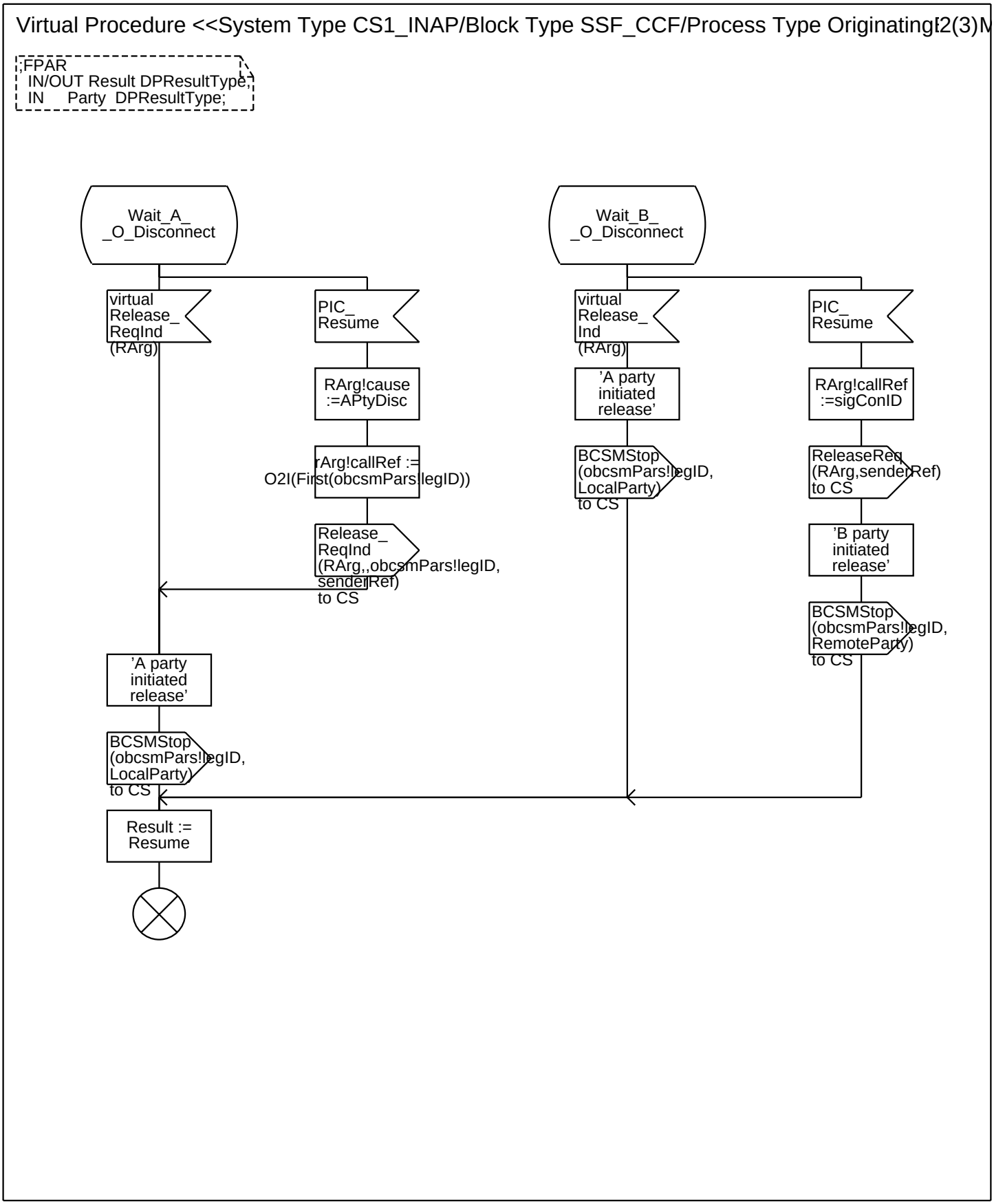
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
1(1)M
;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent:=
oCalled_
PartyBusy
MapToDP
(dp)
DP(DP)
to CS
Wait_
oCalledParty_
Busy
PIC_
Resume
Release_
Ind
(RArg)
virtual
PIC
(PIC)
*
Result :=
Resume
Result :=
ARelease
PIC!
pic
Select_
Route
Release_
Call
Analyse_
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)
rArg!cause :=
pic!cause
Result :=
Select_
Route
Result :=
SCFRelease
O_
Exception
Else
Result :=
Exception
'Error
situation'
–

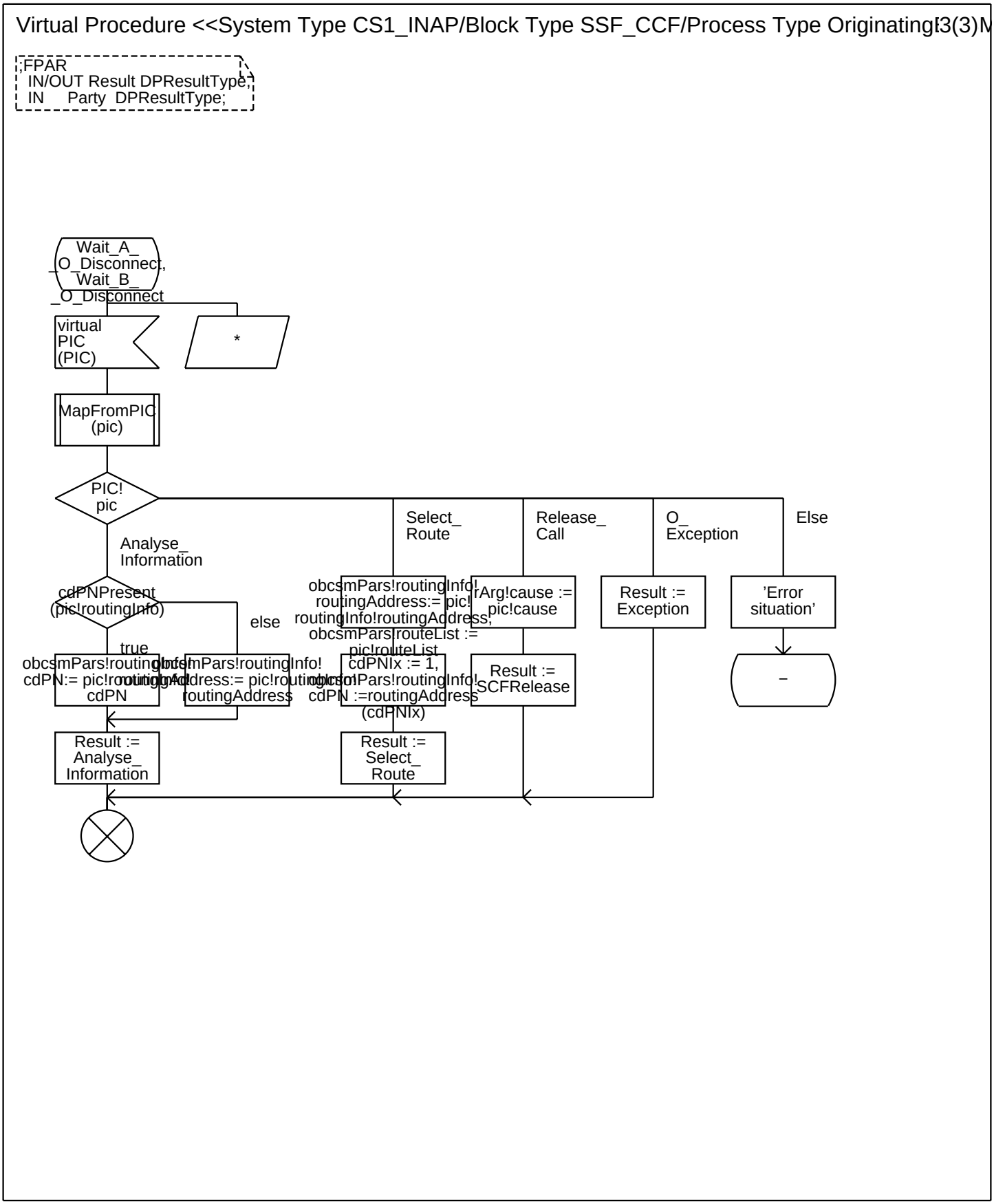
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 1(3)M
;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Party
ARelease
BRelease
DP!legID := obcsmPars!legID,
DP!bcsmEvent :=
ODisconnect,
DP!party := LocalParty
DP!legID := obcsmPars!legID,
DP!bcsmEvent :=
ODisconnect,
DP!party:=RemoteParty
MapToDP
(dp)
MapToDP
(dp)
dp!cause :=
rArg!cause
dp!cause :=
rArg!cause
DP_
Disconnect
(DP)
to CS
DP_
Disconnect
(DP)
to CS
Wait_A_
O_Disconnect
Wait_B_
O_Disconnect

Annex A:

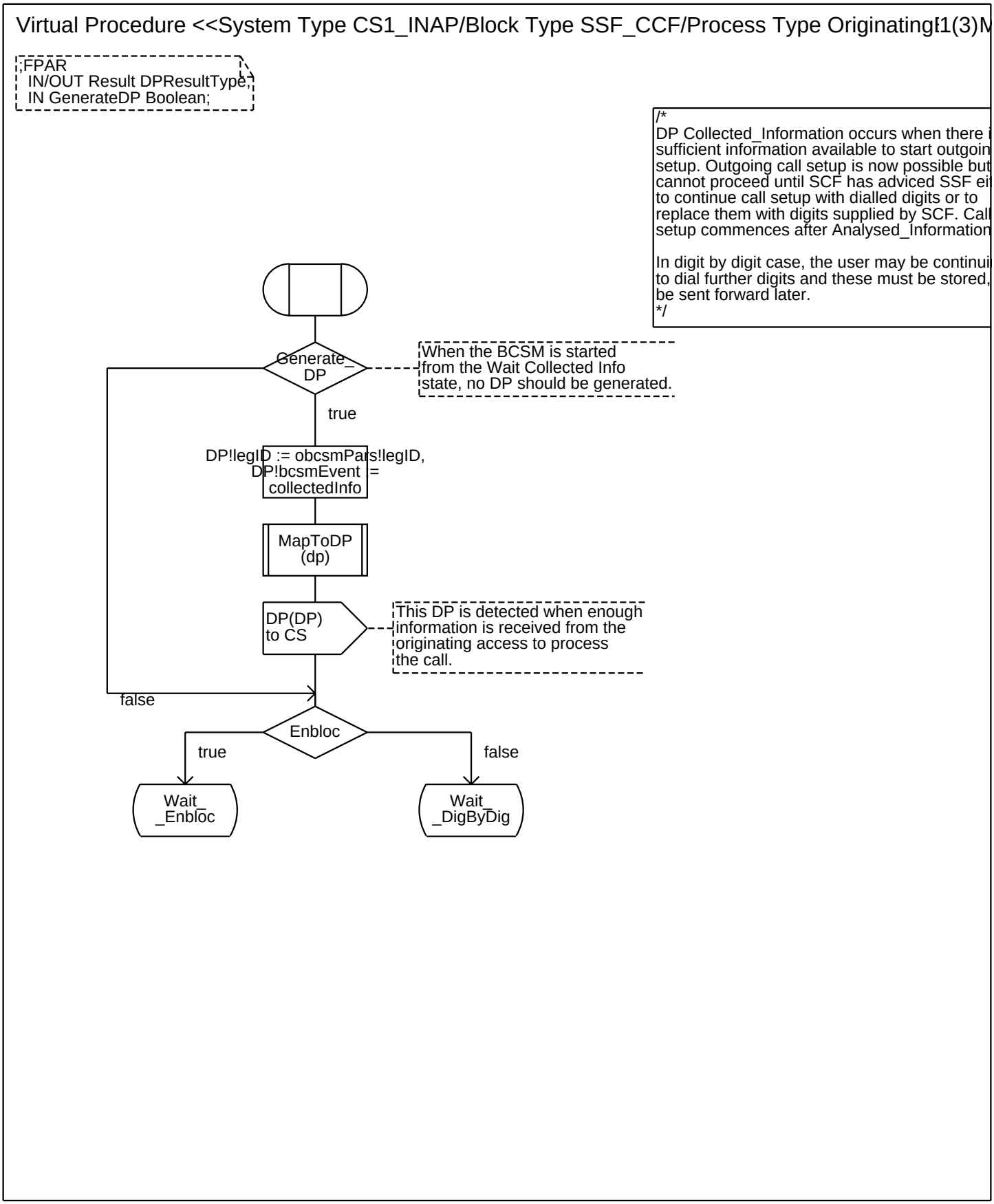

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
2(3)M
;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Wait_A_
_O_Disconnect
virtual
Release_
ReqInd
(RArg)
PIC_
Resume
RArg!cause
:=APtyDisc
rArg!callRef :=
O2I(First(obcsmPars!legID))
Release_
ReqInd
(RArg,,obcsmPars!legID,
senderRef)
to CS
'A party
initiated
release'
BCSMStop
(obcsmPars!legID,
LocalParty)
to CS
Result :=
Resume
Wait_B_
_O_Disconnect
virtual
Release_
Ind
(RArg)
'A party
initiated
release'
BCSMStop
(obcsmPars!legID,
LocalParty)
to CS
PIC_
Resume
RArg!callRef
:=sigConID
ReleaseReq
(RArg,senderRef)
to CS
'B party
initiated
release'
BCSMStop
(obcsmPars!legID,
RemoteParty)
to CS

Annex A:

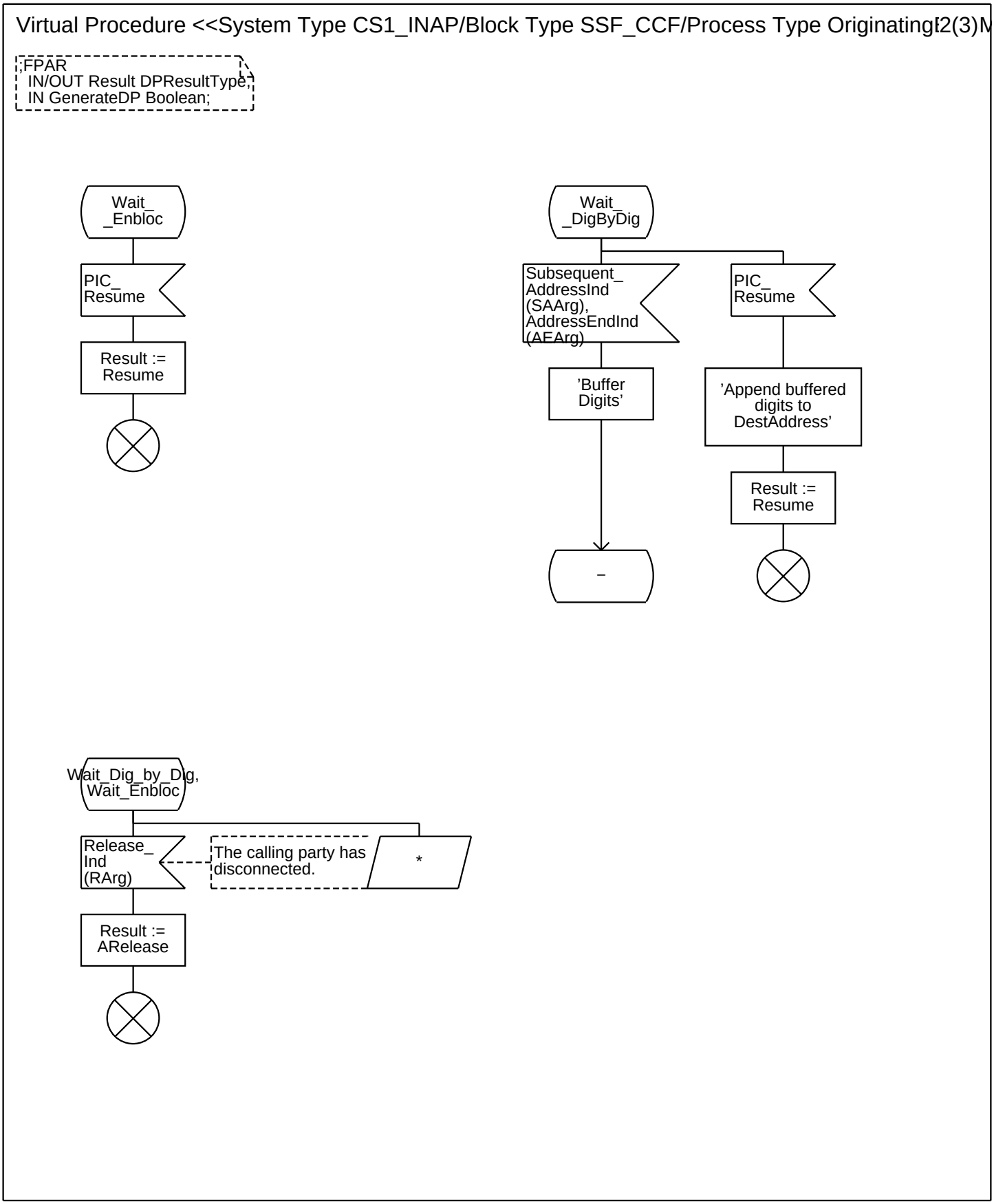

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Wait_A_
_O_Disconnect,
Wait_B_
_O_Disconnect
virtual
PIC
(PIC)
*
MapFromPIC
(pic)
PIC!
pic
Analyse_
Information
Select_
Route
Release_
Call
O_
Exception
Else
cdPNPresent
(pic!routingInfo)
true
else
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
rArg!cause :=
pic!cause
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
–
Result :=
Analyse_
Information
Result :=
Select_
Route

Annex A:

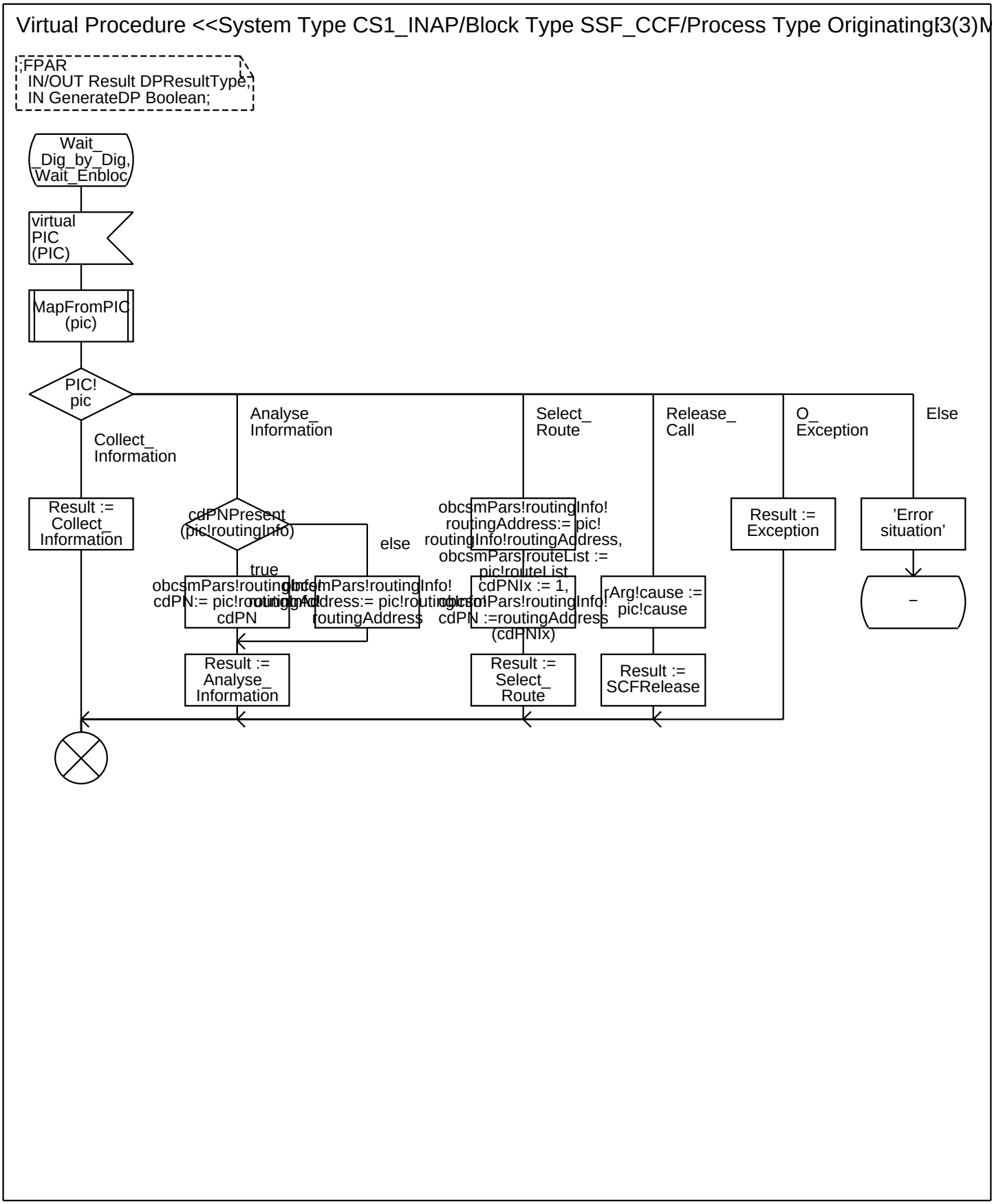
Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 1(3)M



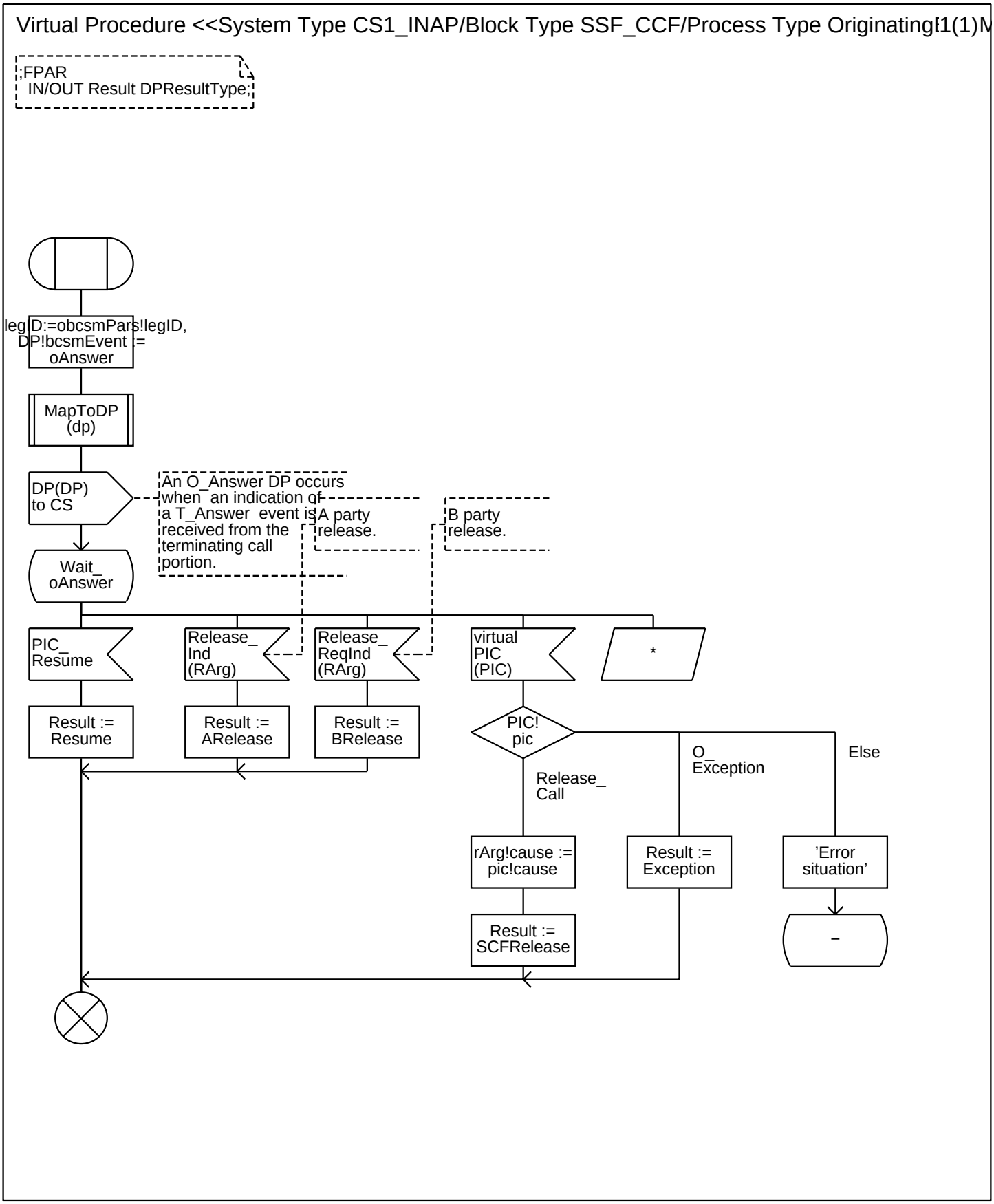
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
2(3)M
;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

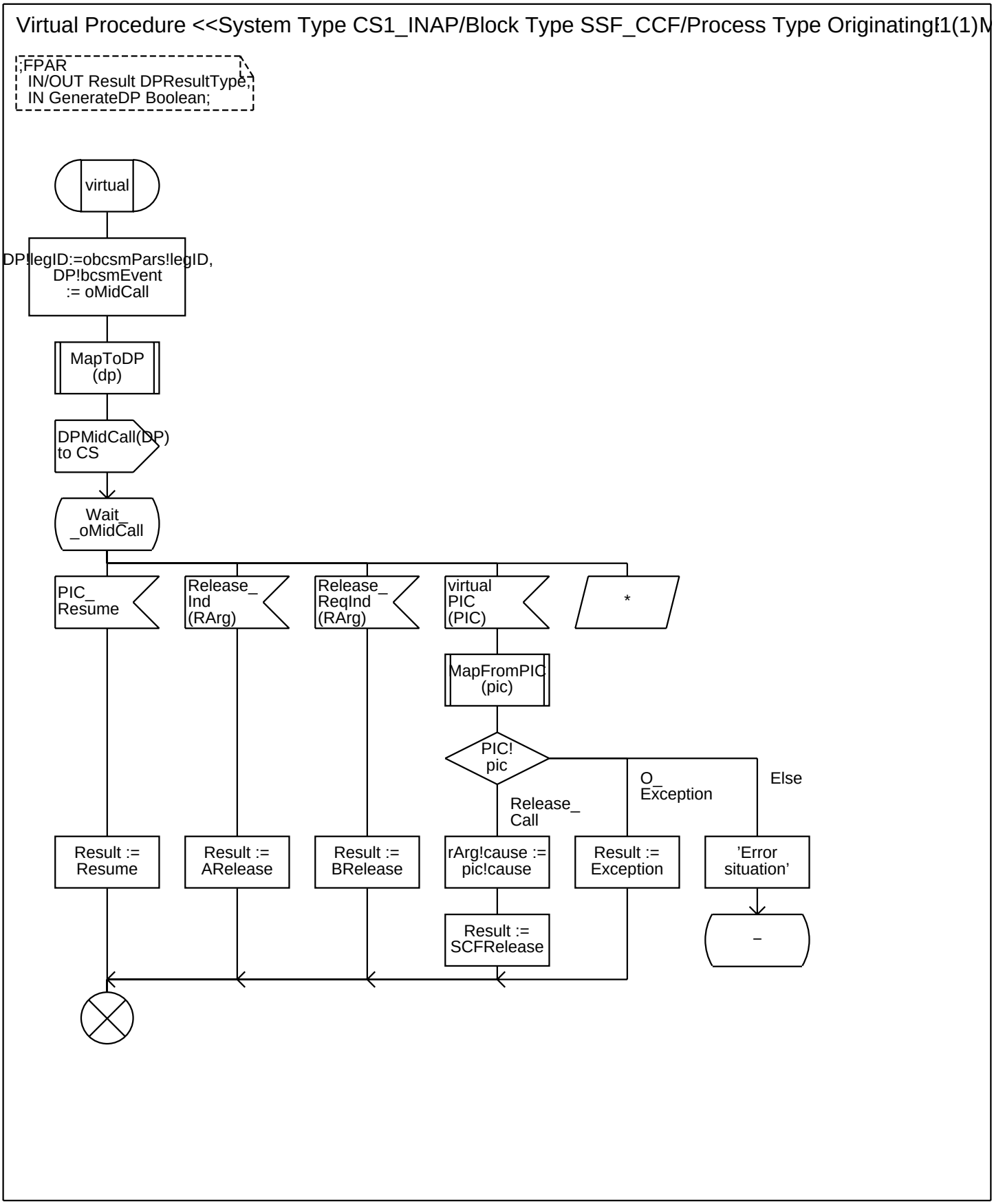
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
3(3)M
;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'
–

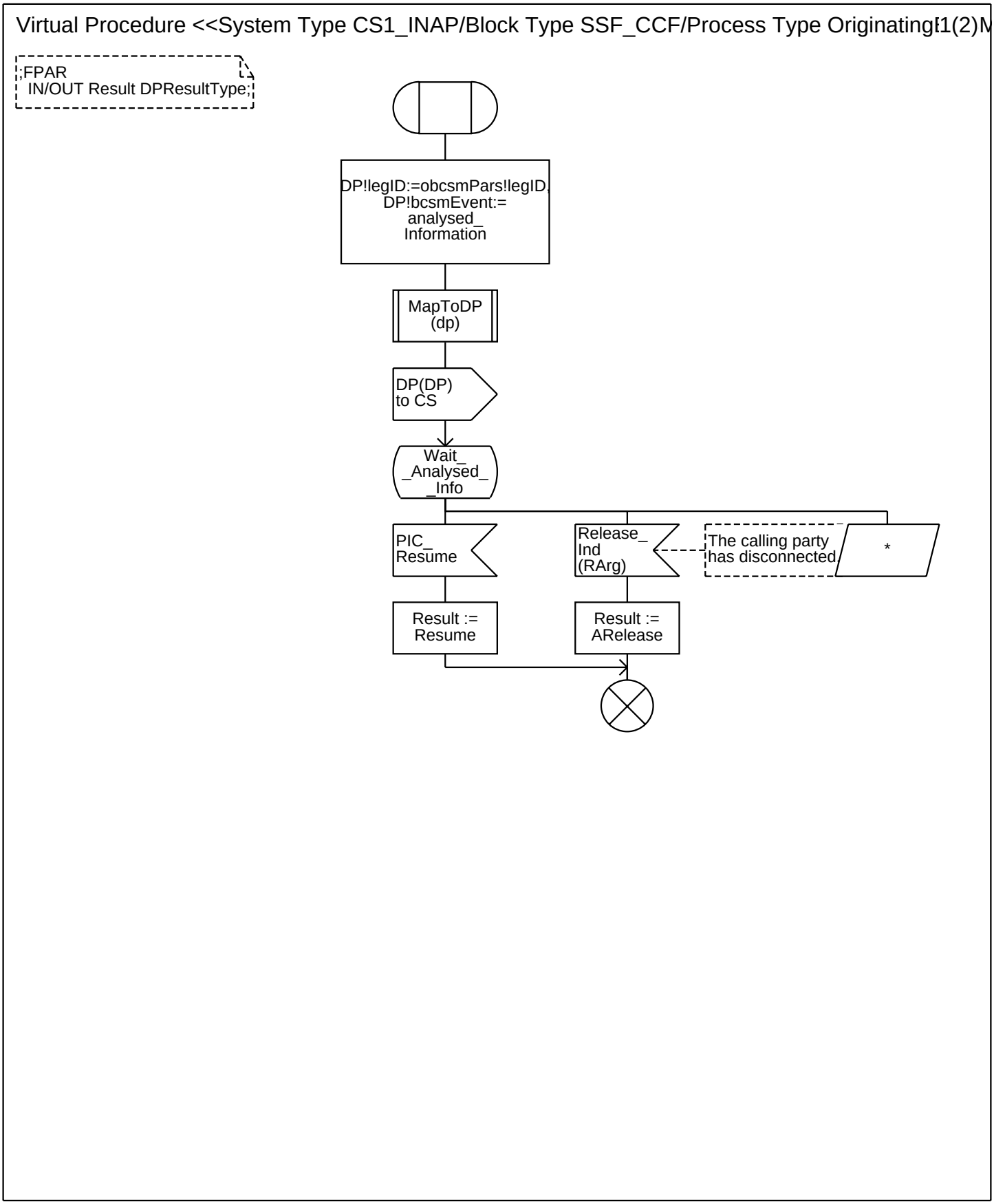
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
1(1)M
;FPAR
IN/OUT Result DPResultType;
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
–

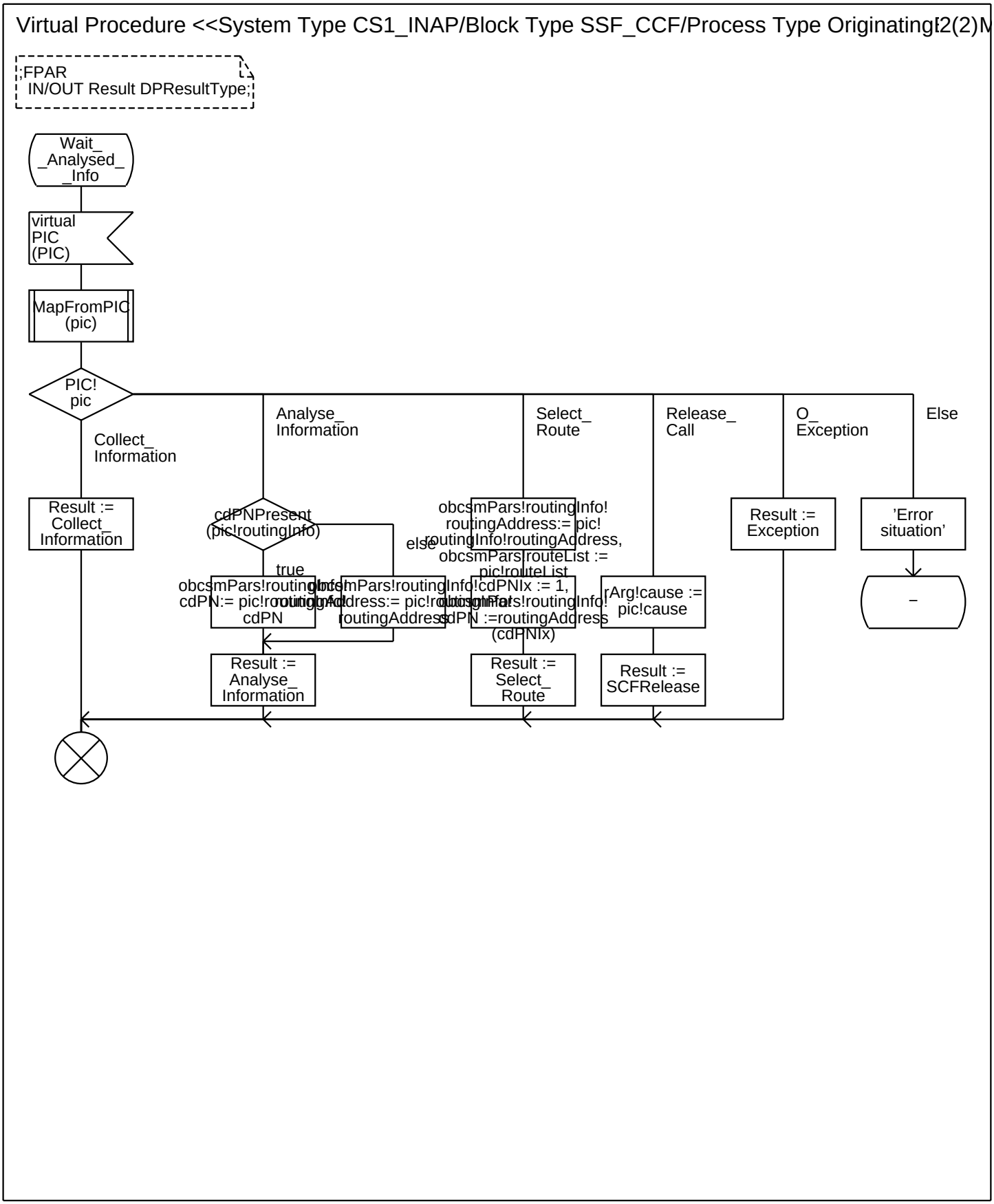
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
1(1)M
;FPAR
IN/OUT Result DPResultType;
IN GenerateDP Boolean;
virtual
DP!legID:=obcsmPars!legID,
DP!bcsmEvent
:= oMidCall
MapToDP
(dp)
DPMidCall(DP)
to CS
Wait_
_oMidCall
PIC_
Resume
Release_
Ind
(RArg)
Release_
ReqInd
(RArg)
virtual
PIC
(PIC)
*
MapFromPIC
(pic)
PIC!
pic
Release_
Call
O_
Exception
Else
Result :=
Resume
Result :=
ARelease
Result :=
BRelease
rArg!cause :=
pic!cause
Result :=
Exception
'Error
situation'
Result :=
SCFRelease
–

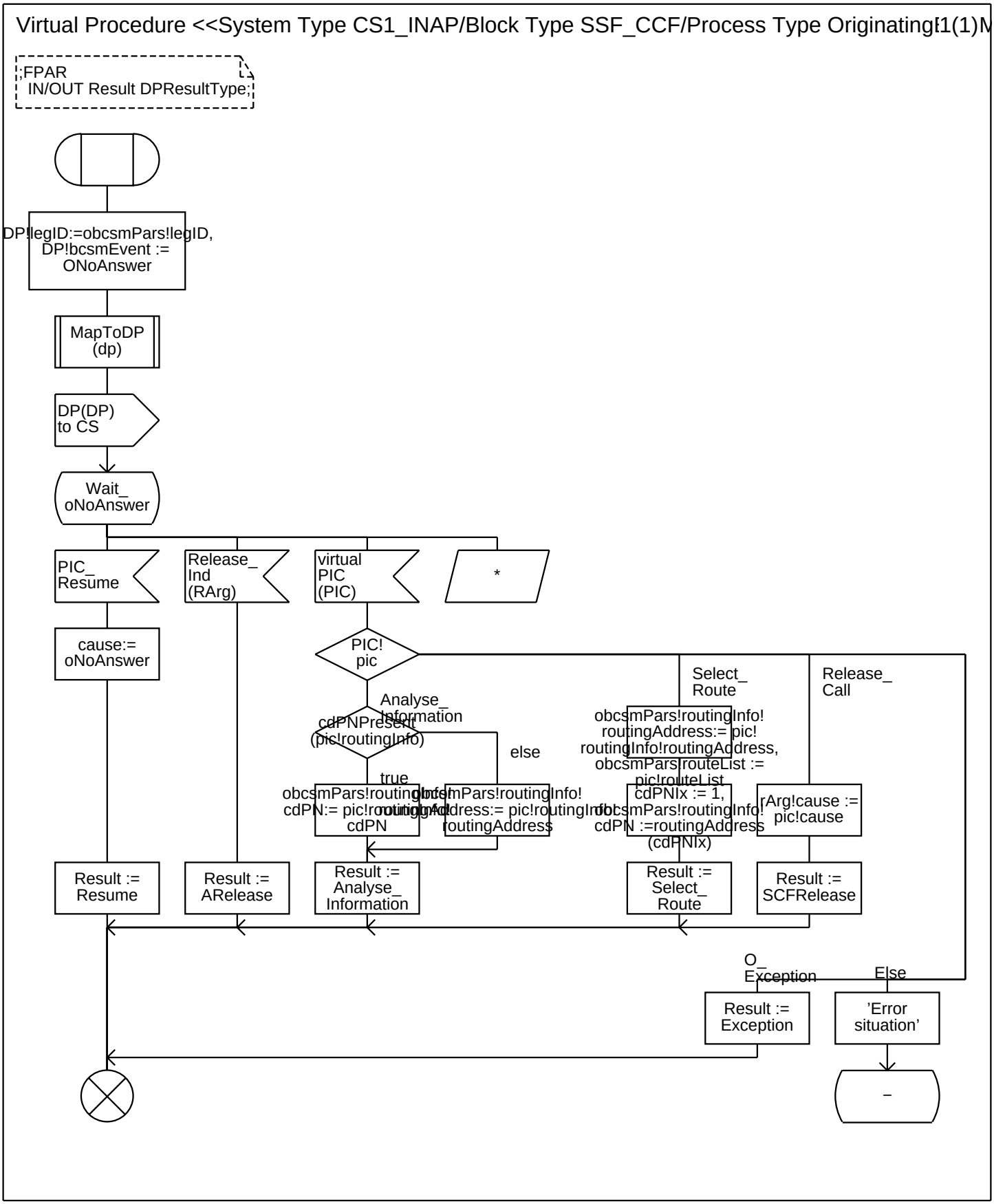
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating 1(2)M
;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

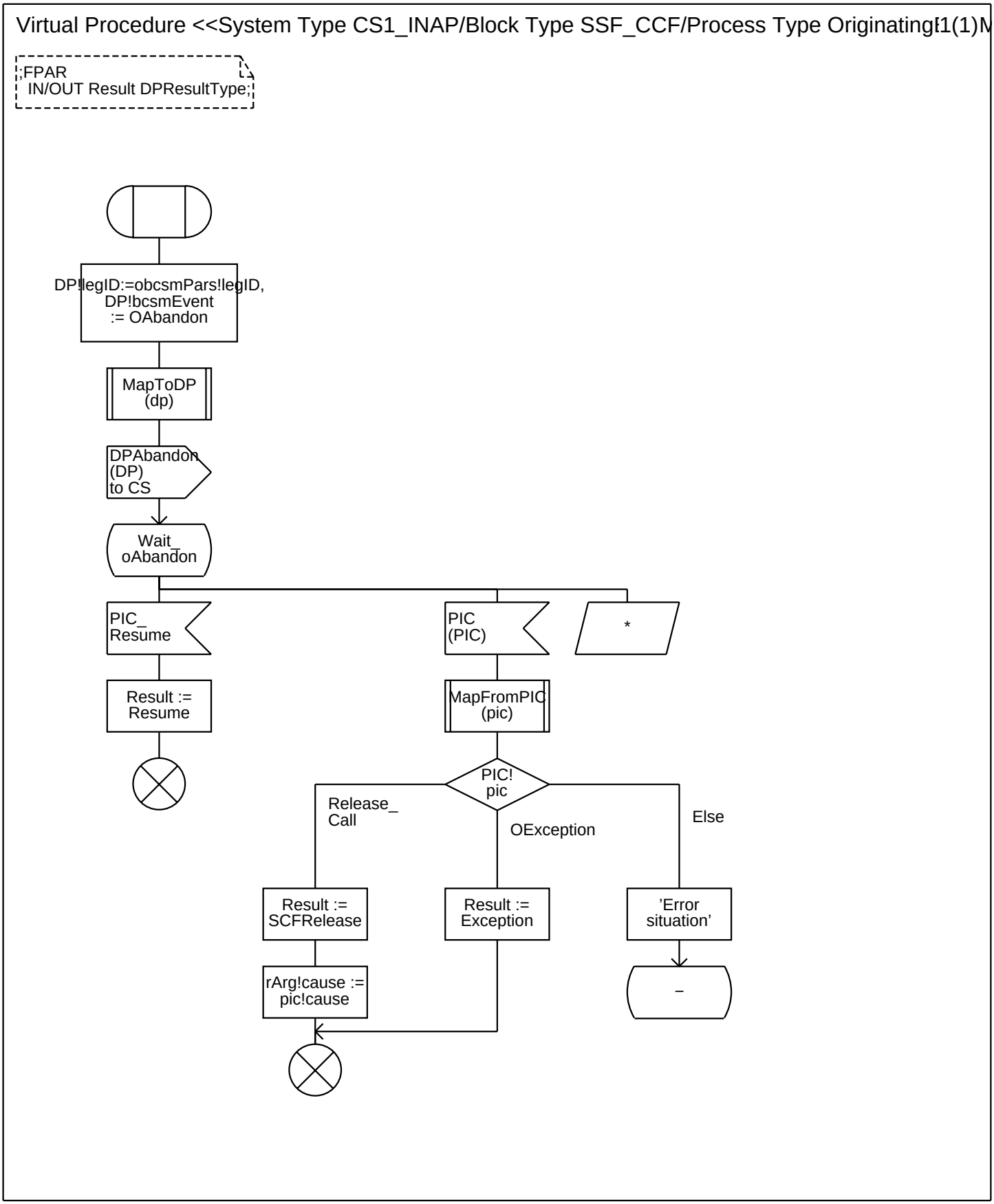
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
;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)
true
else
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
obcsmPars!routingInfo!cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
rArg!cause :=
pic!cause
Result :=
SCFRelease
Result :=
Exception
'Error
situation'
–

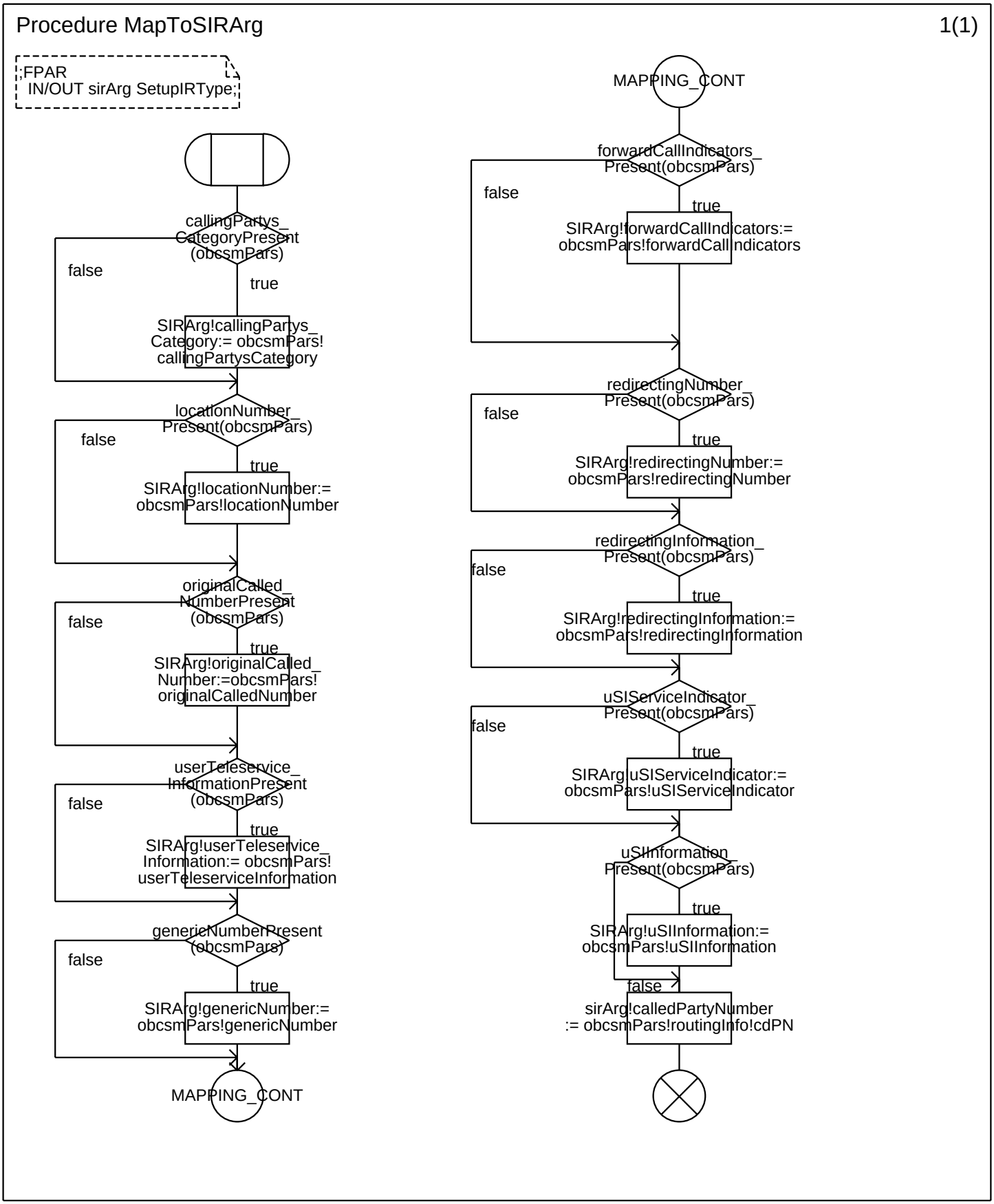
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
1(1)M
;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
ONoAnswer
MapToDP
(dp)
DP(DP)
to CS
Wait_
oNoAnswer
PIC_
Resume
Release_
Ind
(RArg)
virtual
PIC
(PIC)
*
cause:=
oNoAnswer
PIC!
pic
Analyse_
Information
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Select_
Route
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Release_
Call
rArg!cause :=
pic!cause
Result :=
Resume
Result :=
ARelease
Result :=
Analyse_
Information
Result :=
Select_
Route
Result :=
SCFRelease
O_
Exception
Else
Result :=
Exception
'Error
situation'
–

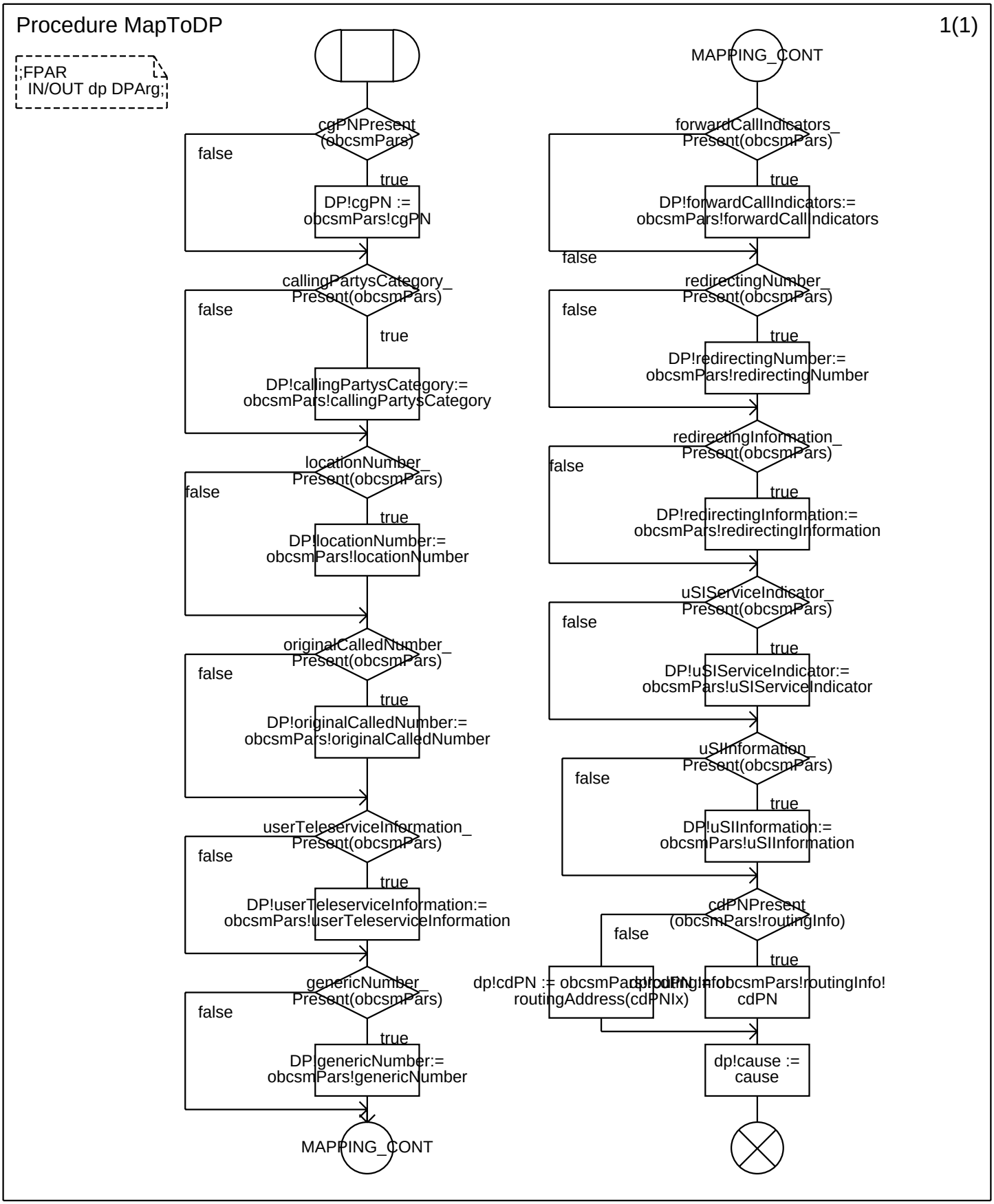
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Originating
1(1)M
;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
–

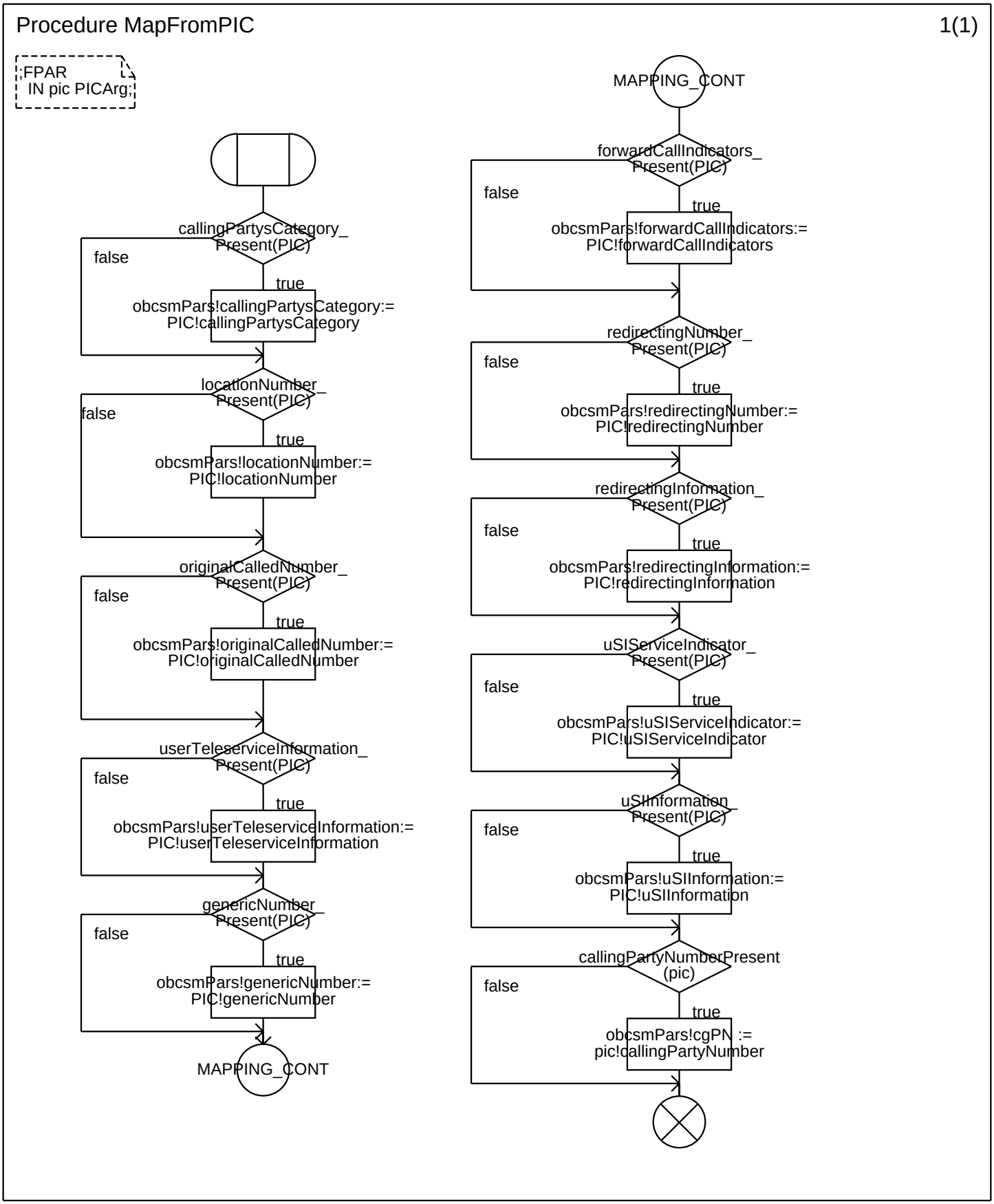
Annex A:


Procedure MapToSIRArg
1(1)
;FPAR
IN/OUT sirArg SetupIRType;
callingPartys_
CategoryPresent
(obcsmPars)
false
true
SIRArg!callingPartys_
Category:= obcsmPars!
callingPartysCategory
locationNumber_
Present(obcsmPars)
false
true
SIRArg!locationNumber:=
obcsmPars!locationNumber
originalCalled_
NumberPresent
(obcsmPars)
false
true
SIRArg!originalCalled_
Number:=obcsmPars!
originalCalledNumber
userTeleservice_
InformationPresent
(obcsmPars)
false
true
SIRArg!userTeleservice_
Information:= obcsmPars!
userTeleserviceInformation
genericNumberPresent
(obcsmPars)
false
true
SIRArg!genericNumber:=
obcsmPars!genericNumber
MAPPING_CONT
MAPPING_CONT
forwardCallIndicators_
Present(obcsmPars)
false
true
SIRArg!forwardCallIndicators:=
obcsmPars!forwardCallIndicators
redirectingNumber_
Present(obcsmPars)
false
true
SIRArg!redirectingNumber:=
obcsmPars!redirectingNumber
redirectingInformation_
Present(obcsmPars)
false
true
SIRArg!redirectingInformation:=
obcsmPars!redirectingInformation
uSIServiceIndicator_
Present(obcsmPars)
false
true
SIRArg!uSIServiceIndicator:=
obcsmPars!uSIServiceIndicator
uSIInformation_
Present(obcsmPars)
true
SIRArg!uSIInformation:=
obcsmPars!uSIInformation
false
sirArg!calledPartyNumber
:= obcsmPars!routingInfo!cdPN

Annex A:


Procedure MapToDP
1(1)
;FPAR
IN/OUT dp DPArg;
cgPNPresent (obcsmPars)
false
true
DP!cgPN := obcsmPars!cgPN
callingPartysCategory_ Present(obcsmPars)
false
true
DP!callingPartysCategory:= obcsmPars!callingPartysCategory
locationNumber_ Present(obcsmPars)
false
true
DP!locationNumber:= obcsmPars!locationNumber
originalCalledNumber_ Present(obcsmPars)
false
true
DP!originalCalledNumber:= obcsmPars!originalCalledNumber
userTeleserviceInformation_ Present(obcsmPars)
false
true
DP!userTeleserviceInformation:= obcsmPars!userTeleserviceInformation
genericNumber_ Present(obcsmPars)
false
true
DP!genericNumber:= obcsmPars!genericNumber
MAPPING_CONT
MAPPING_CONT
forwardCallIndicators_ Present(obcsmPars)
true
DP!forwardCallIndicators:= obcsmPars!forwardCallIndicators
false
redirectingNumber_ Present(obcsmPars)
false
true
DP!redirectingNumber:= obcsmPars!redirectingNumber
redirectingInformation_ Present(obcsmPars)
false
true
DP!redirectingInformation:= obcsmPars!redirectingInformation
uSIServiceIndicator_ Present(obcsmPars)
false
true
DP!uSIServiceIndicator:= obcsmPars!uSIServiceIndicator
uSIInformation_ Present(obcsmPars)
false
true
DP!uSIInformation:= obcsmPars!uSIInformation
cdPNPresent (obcsmPars!routingInfo)
false
true
dp!cdPN := obcsmPars!routingInfo! routingAddress(cdPNIx)
dp!cdPN := obcsmPars!routingInfo! cdPN
dp!cause := cause

Annex A:


Procedure MapFromPIC
1(1)
;FPAR
IN pic PICArg;
callingPartysCategory_
Present(PIC)
false
true
obcsmPars!callingPartysCategory:=
PIC!callingPartysCategory
locationNumber_
Present(PIC)
false
true
obcsmPars!locationNumber:=
PIC!locationNumber
originalCalledNumber_
Present(PIC)
false
true
obcsmPars!originalCalledNumber:=
PIC!originalCalledNumber
userTeleserviceInformation_
Present(PIC)
false
true
obcsmPars!userTeleserviceInformation:=
PIC!userTeleserviceInformation
genericNumber_
Present(PIC)
false
true
obcsmPars!genericNumber:=
PIC!genericNumber
MAPPING_CONT
MAPPING_CONT
forwardCallIndicators_
Present(PIC)
false
true
obcsmPars!forwardCallIndicators:=
PIC!forwardCallIndicators
redirectingNumber_
Present(PIC)
false
true
obcsmPars!redirectingNumber:=
PIC!redirectingNumber
redirectingInformation_
Present(PIC)
false
true
obcsmPars!redirectingInformation:=
PIC!redirectingInformation
uSIServiceIndicator_
Present(PIC)
false
true
obcsmPars!uSIServiceIndicator:=
PIC!uSIServiceIndicator
uSIInformation_
Present(PIC)
false
true
obcsmPars!uSIInformation:=
PIC!uSIInformation
callingPartyNumberPresent
(pic)
false
true
obcsmPars!cgPN :=
pic!callingPartyNumber

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCS1(8)

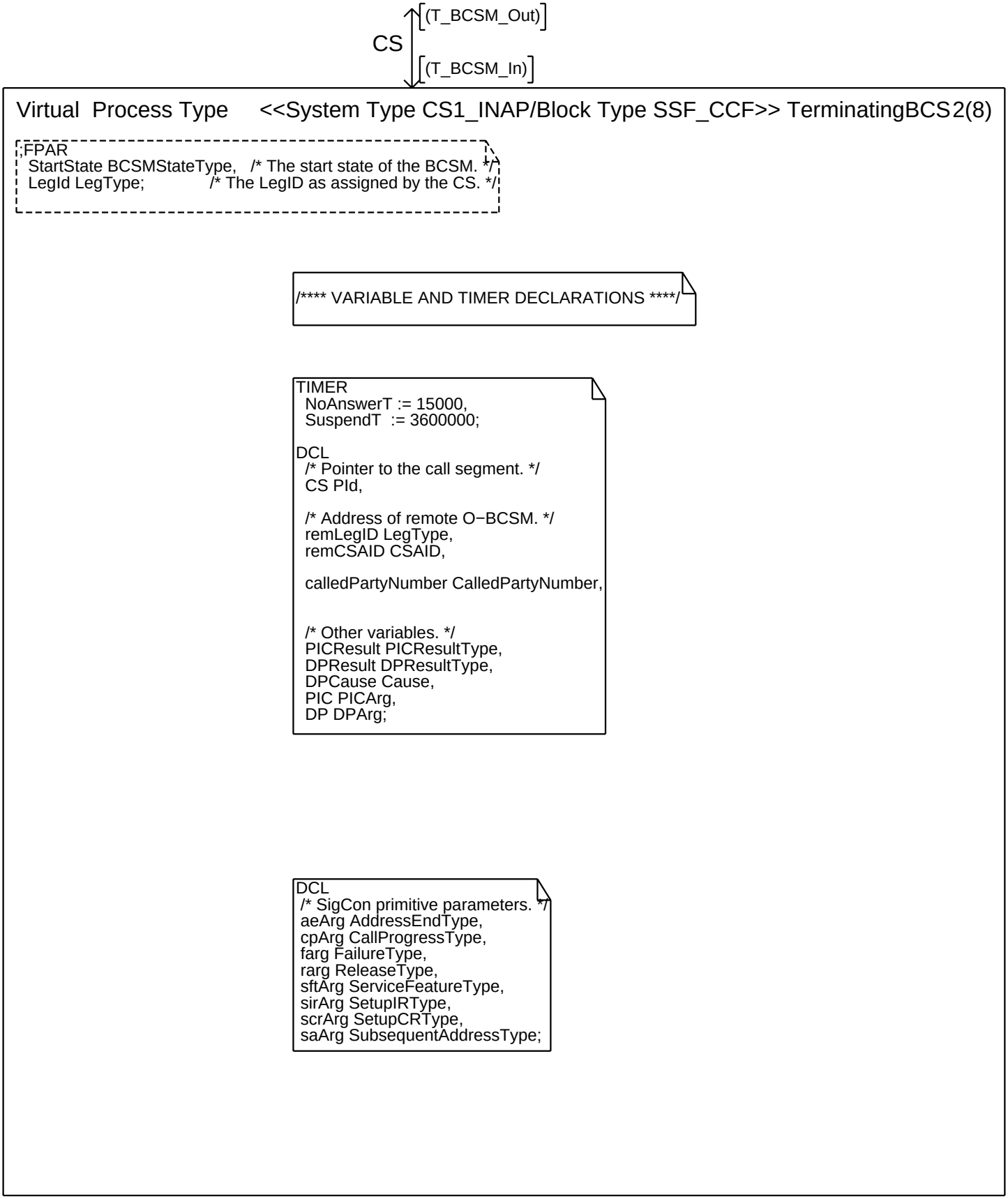
```
;FPAR
 StartState BCSMStateType,   /* The start state of the BCSM. */
 LegId LegType;              /* The LegID as assigned by the CS. */
```

```
/***** 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,
    NotReachable,
    DL_A, DL_B; /* Used with DisconnectLeg */
ENDNEWTYPE;
```

Annex A:


CS
(T_BCSM_Out)
(T_BCSM_In)
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCS2(8)
;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;

Annex A:

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCS3(8)

```
;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

virtual
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

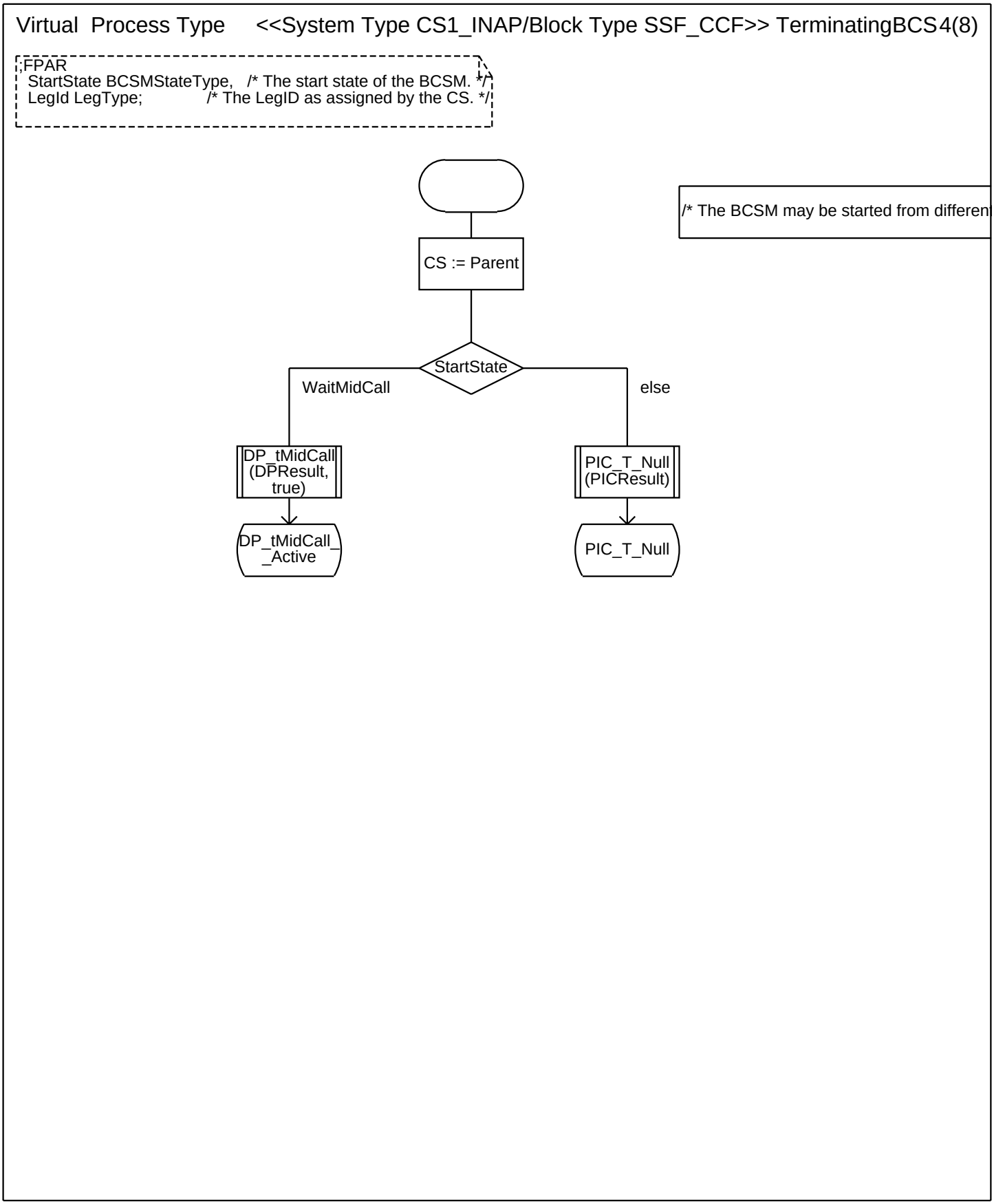
virtual
DP_tBusy

virtual
DP_tMidCall

virtual
DP_tAnswer

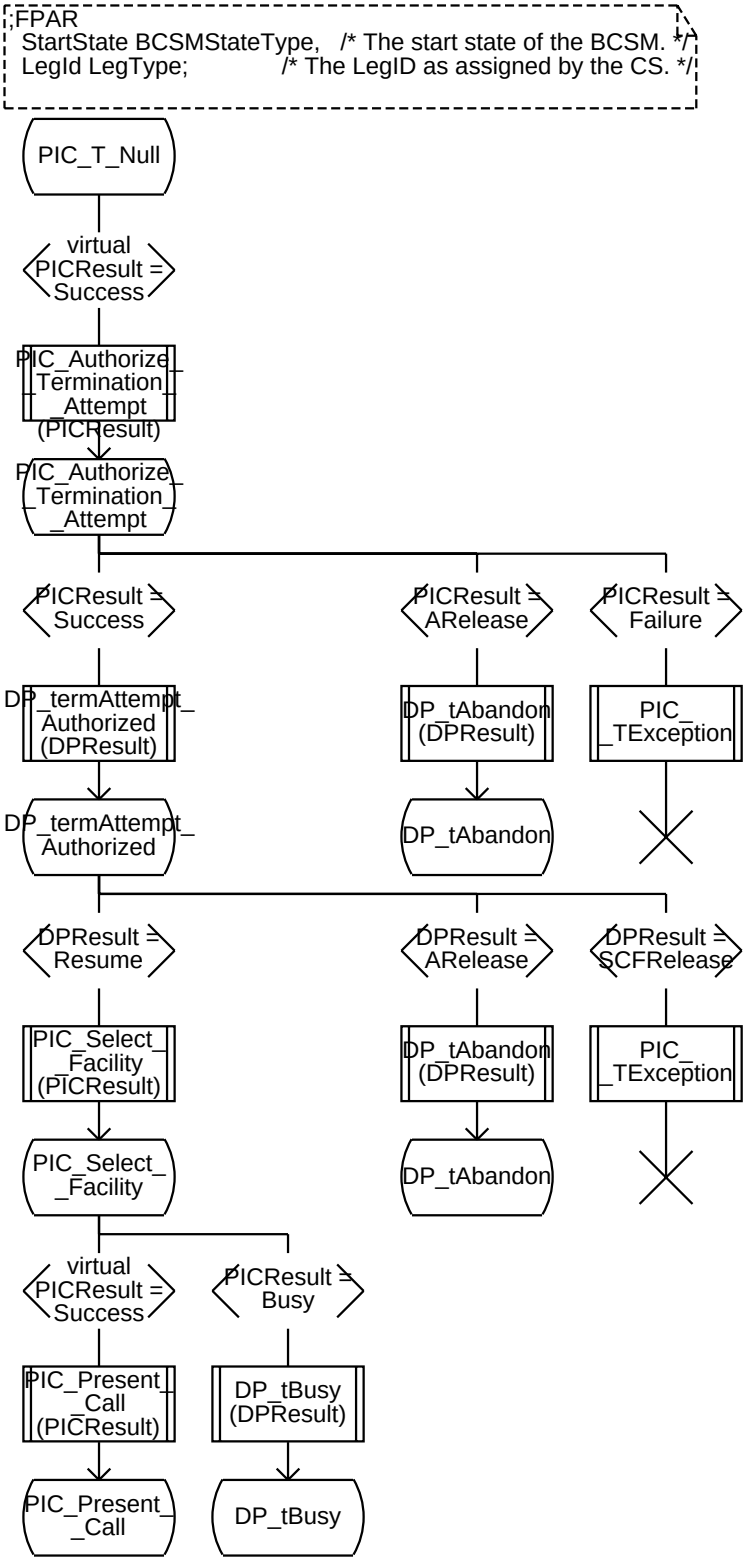
virtual
DP_tAbandon

Annex A:

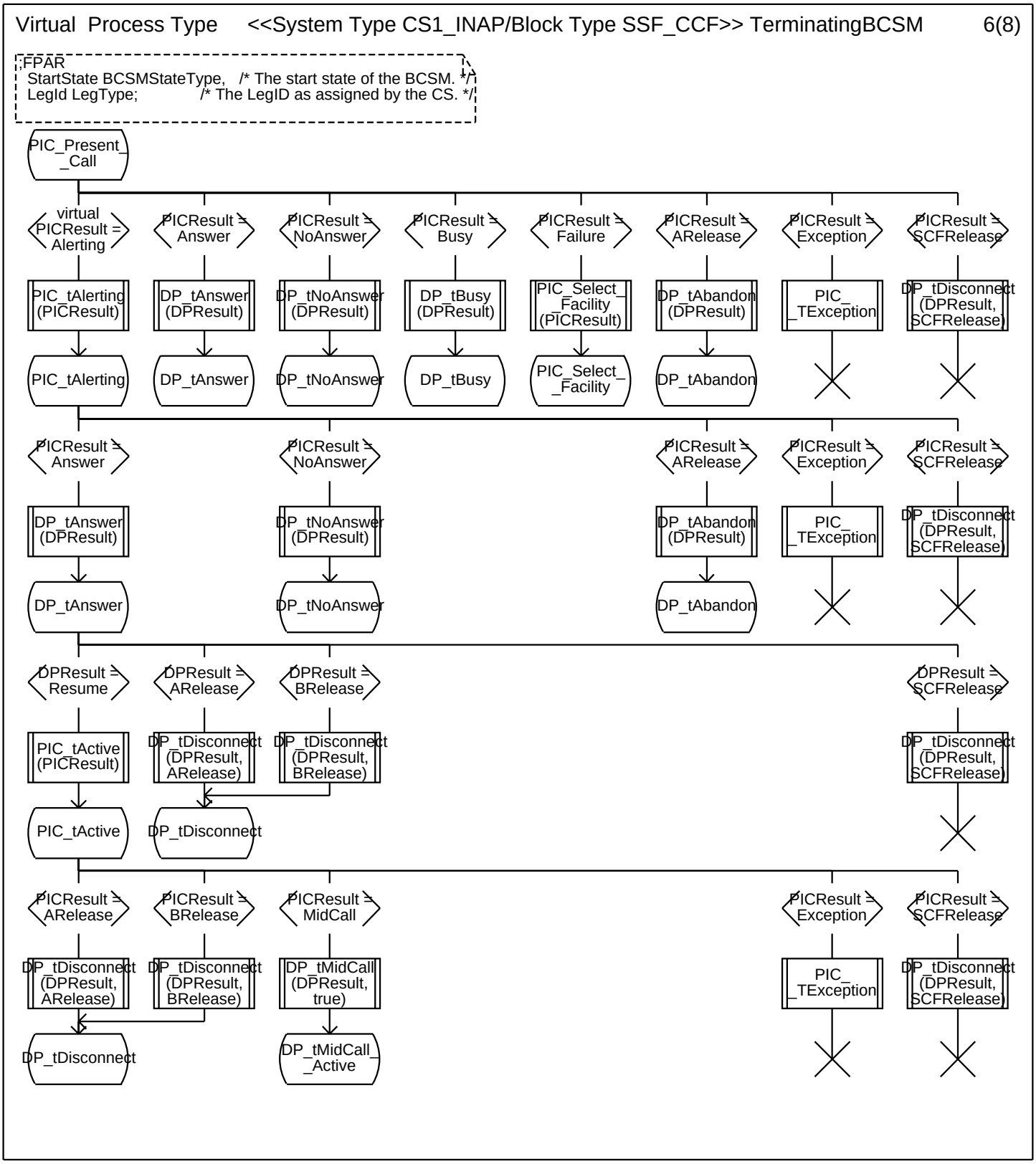

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCS4(8)
;FPAR
StartState BCSMStateType, /* The start state of the BCSM. */
LegId LegType; /* The LegID as assigned by the CS. */
/* The BCSM may be started from differen
CS := Parent
StartState
WaitMidCall
else
DP_tMidCall (DPResult, true)
PIC_T_Null (PICResult)
DP_tMidCall_ _Active
PIC_T_Null

Annex A:

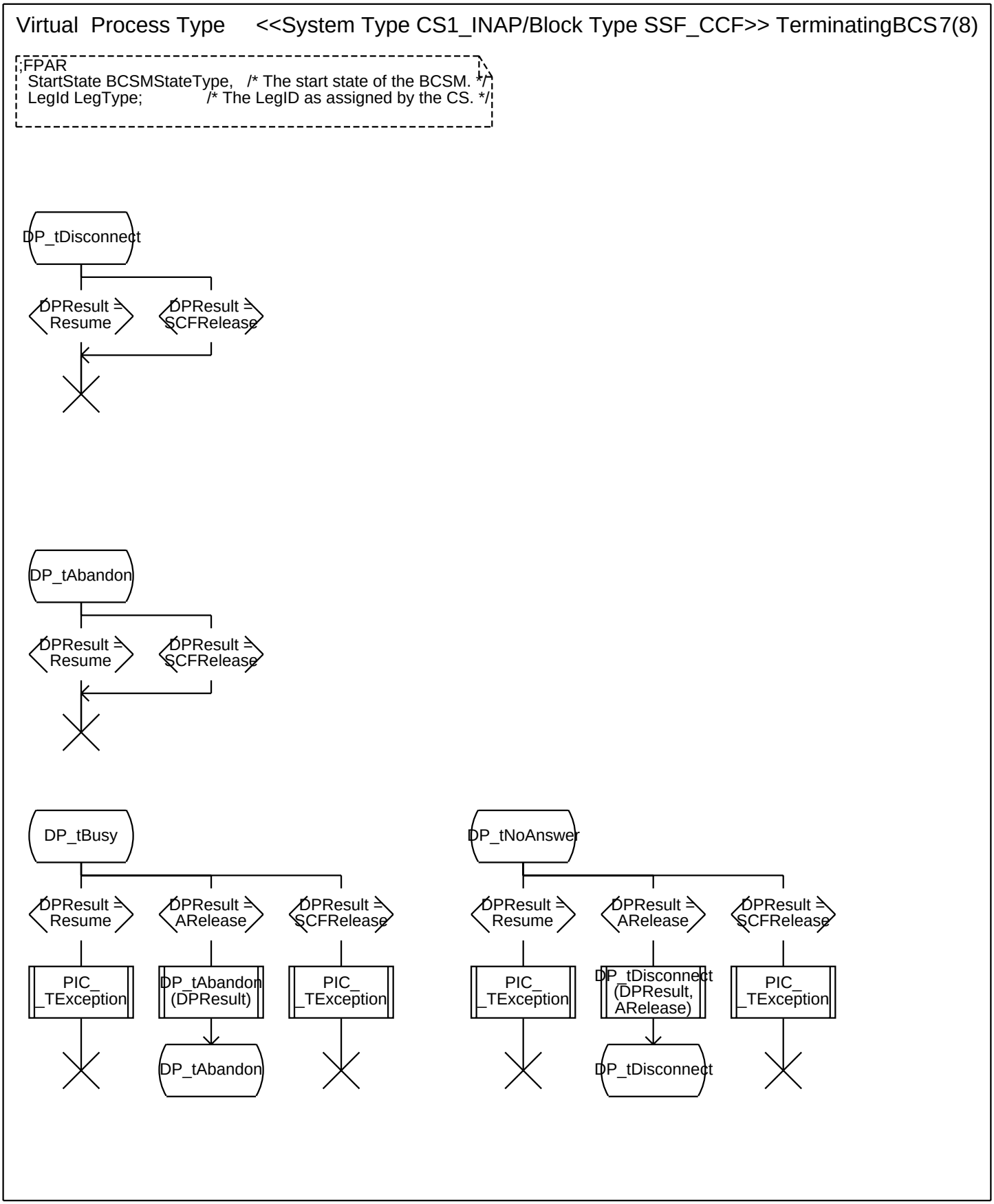
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCS5(8)



Annex A:

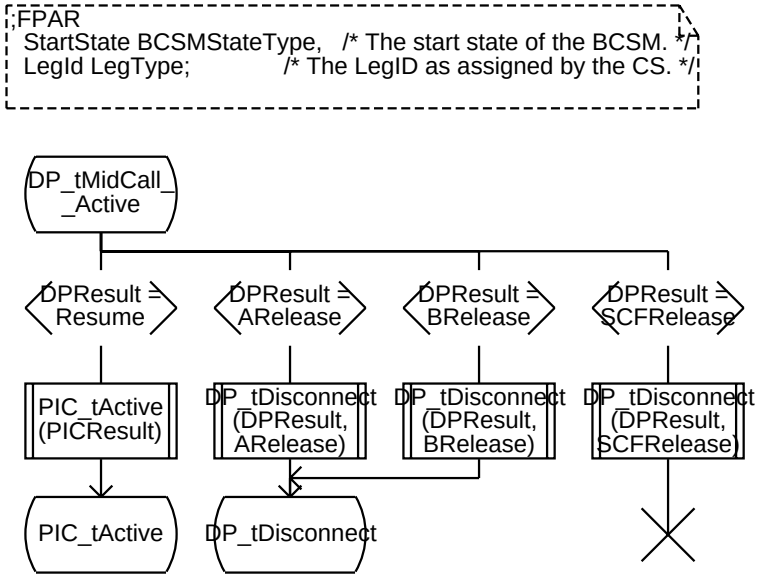

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCSM 6(8)
FPAR
StartState BCSMStateType, /* The start state of the BCSM. */
LegId LegType; /* The LegID as assigned by the CS. */
PIC_Present_ _Call
virtual PICResult = Alerting
PICResult = Answer
PICResult = NoAnswer
PICResult = Busy
PICResult = Failure
PICResult = ARelease
PICResult = Exception
PICResult = SCFRelease
PIC_tAlerting (PICResult)
DP_tAnswer (DPResult)
DP_tNoAnswer (DPResult)
DP_tBusy (DPResult)
PIC_Select_ _Facility (PICResult)
DP_tAbandon (DPResult)
PIC_ _TException
DP_tDisconnect (DPResult, SCFRelease)
PIC_tAlerting
DP_tAnswer
DP_tNoAnswer
DP_tBusy
PIC_Select_ _Facility
DP_tAbandon
PICResult = Answer
PICResult = NoAnswer
PICResult = ARelease
PICResult = Exception
PICResult = SCFRelease
DP_tAnswer (DPResult)
DP_tNoAnswer (DPResult)
DP_tAbandon (DPResult)
PIC_ _TException
DP_tDisconnect (DPResult, SCFRelease)
DP_tAnswer
DP_tNoAnswer
DP_tAbandon
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = SCFRelease
PIC_tActive (PICResult)
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
DP_tDisconnect (DPResult, SCFRelease)
PIC_tActive
DP_tDisconnect
PICResult = ARelease
PICResult = BRelease
PICResult = MidCall
PICResult = Exception
PICResult = SCFRelease
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
DP_tMidCall (DPResult, true)
PIC_ _TException
DP_tDisconnect (DPResult, SCFRelease)
DP_tDisconnect
DP_tMidCall_ _Active

Annex A:

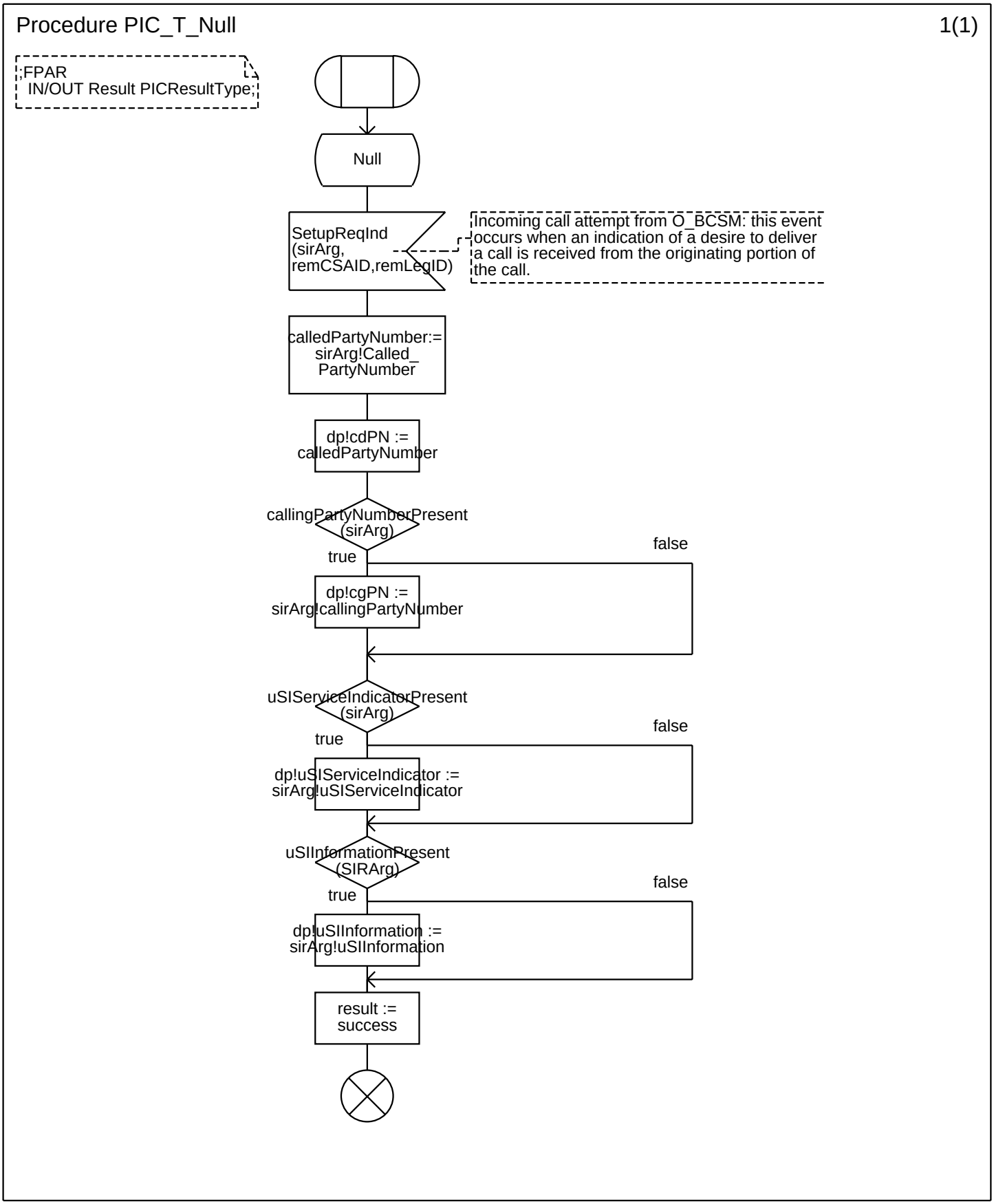

Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCS7(8)
;FPAR
StartState BCSMStateType, /* The start state of the BCSM. */
LegId LegType; /* The LegID as assigned by the CS. */
DP_tDisconnect
DPResult = Resume
DPResult = SCFRelease
DP_tAbandon
DPResult = Resume
DPResult = SCFRelease
DP_tBusy
DPResult = Resume
DPResult = ARelease
DPResult = SCFRelease
PIC_ _TException
DP_tAbandon (DPResult)
PIC_ _TException
DP_tAbandon
DP_tNoAnswer
DPResult = Resume
DPResult = ARelease
DPResult = SCFRelease
PIC_ _TException
DP_tDisconnect (DPResult, ARelease)
PIC_ _TException
DP_tDisconnect

Annex A:

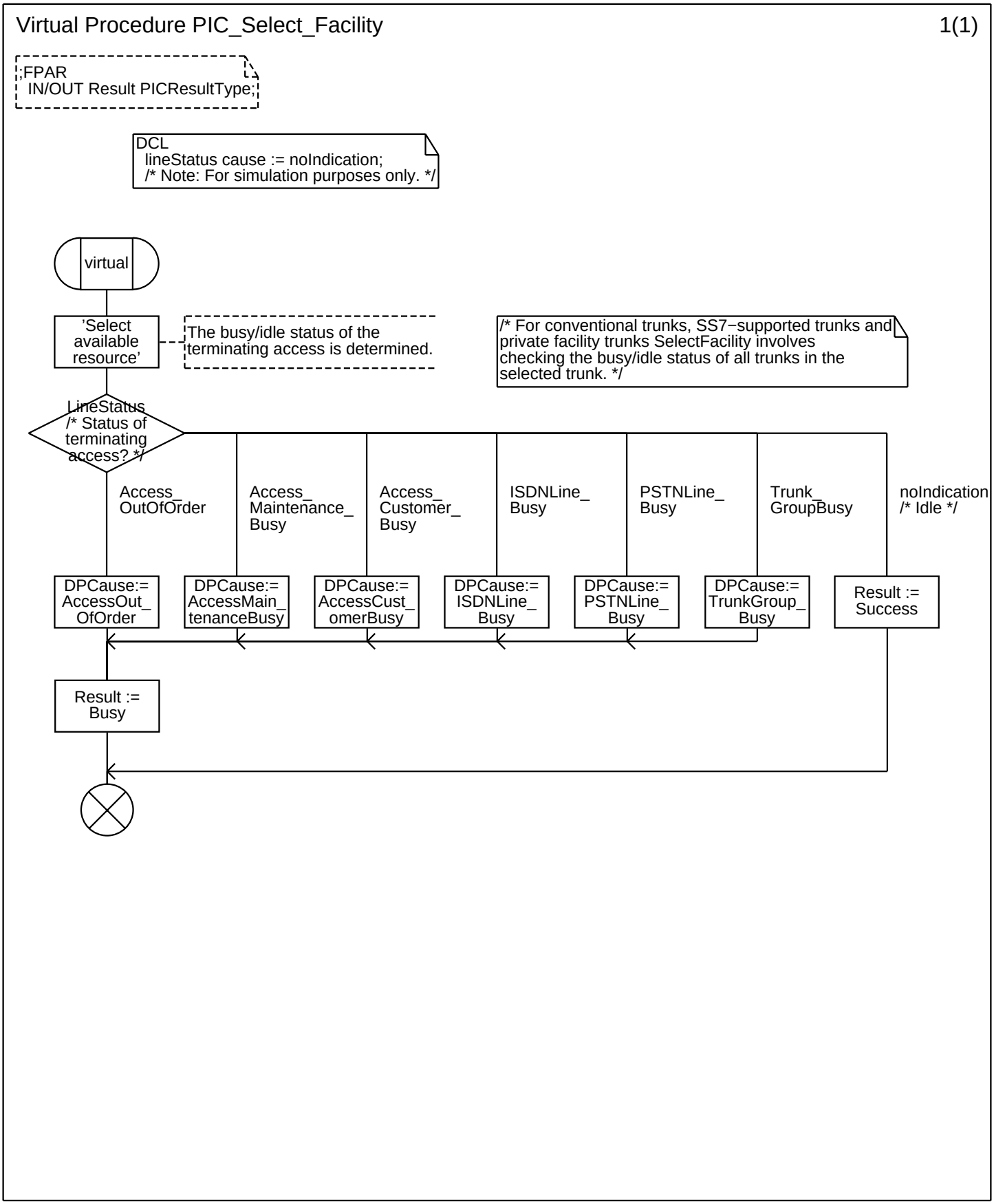
Virtual Process Type <<System Type CS1_INAP/Block Type SSF_CCF>> TerminatingBCS8(8)



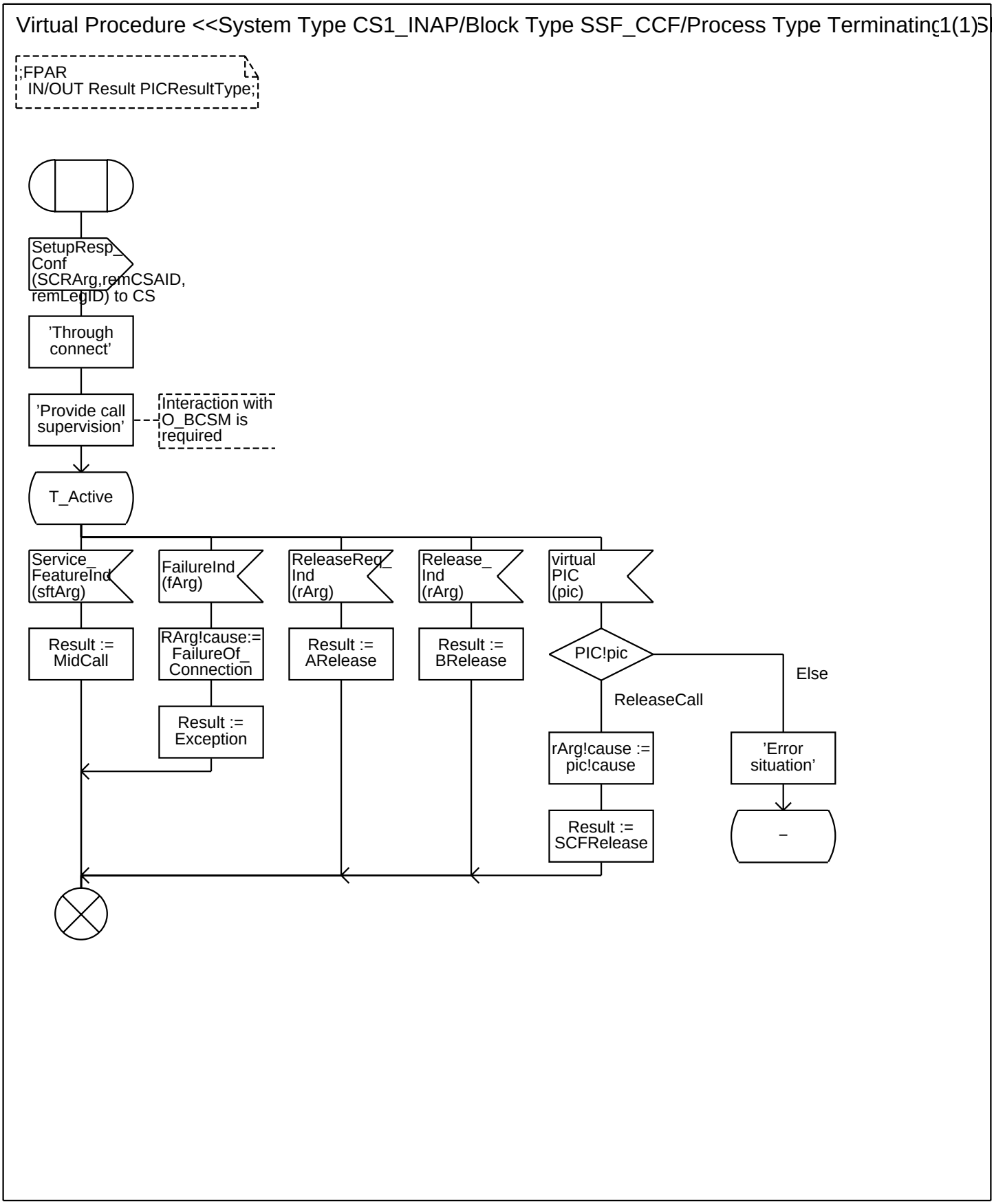
Annex A:


Procedure PIC_T_Null
1(1)
;FPAR
IN/OUT Result PICResultType;
Null
SetupReqInd
(sirArg,
remCSAID,remLegID)
Incoming call attempt from O_BCSM: this event occurs when an indication of a desire to deliver a call is received from the originating portion of the call.
calledPartyNumber:=
sirArg!Called_
PartyNumber
dp!cdPN :=
calledPartyNumber
callingPartyNumberPresent
(sirArg)
true
false
dp!cgPN :=
sirArg!callingPartyNumber
uSIServiceIndicatorPresent
(sirArg)
true
false
dp!uSIServiceIndicator :=
sirArg!uSIServiceIndicator
uSIInformationPresent
(SIRArg)
true
false
dp!uSIInformation :=
sirArg!uSIInformation
result :=
success

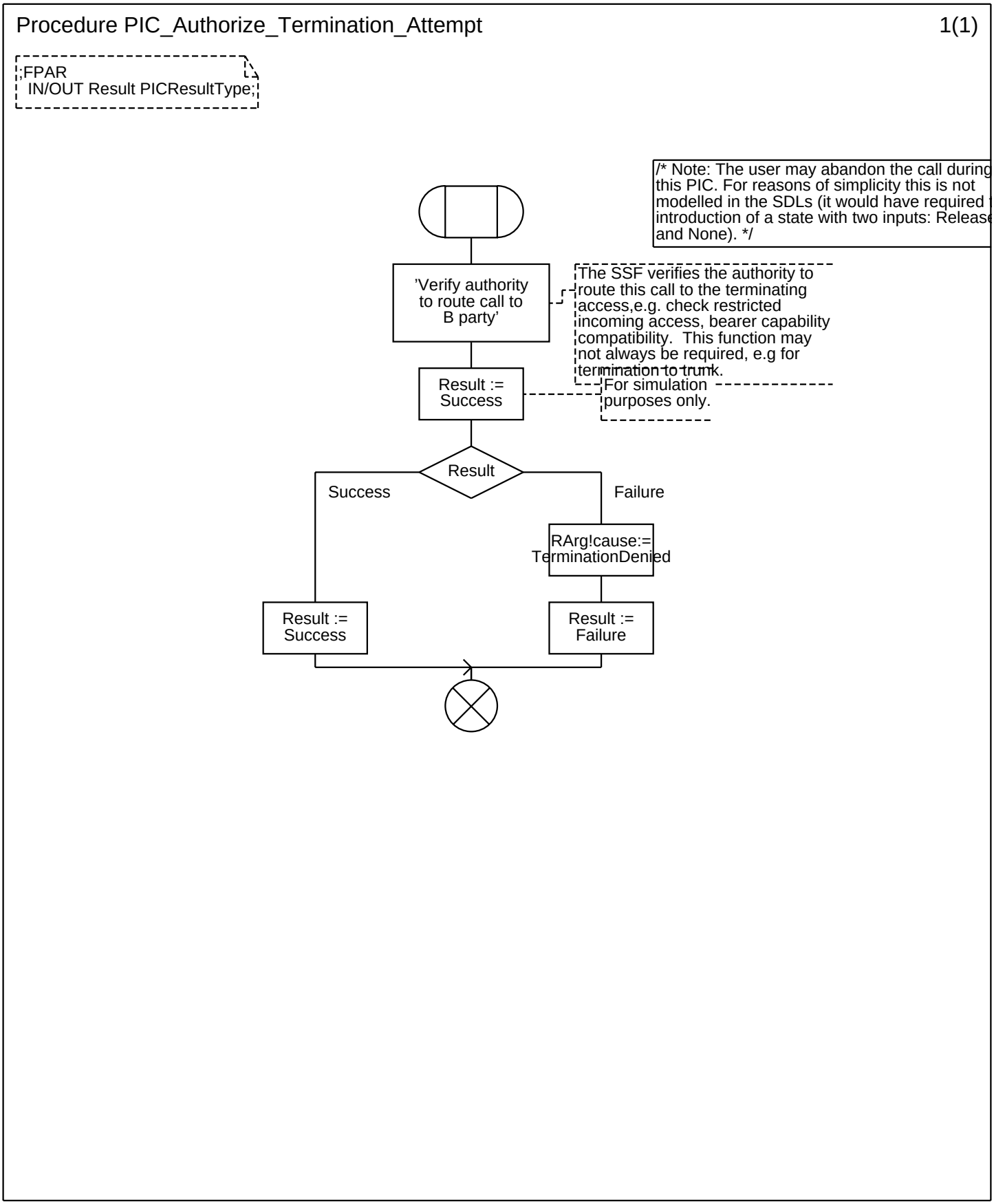
Annex A:


Virtual Procedure PIC_Select_Facility
1(1)
;FPAR
IN/OUT Result PICResultType;
DCL
lineStatus cause := noIndication;
/* Note: For simulation purposes only. */
virtual
'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
Result := Busy

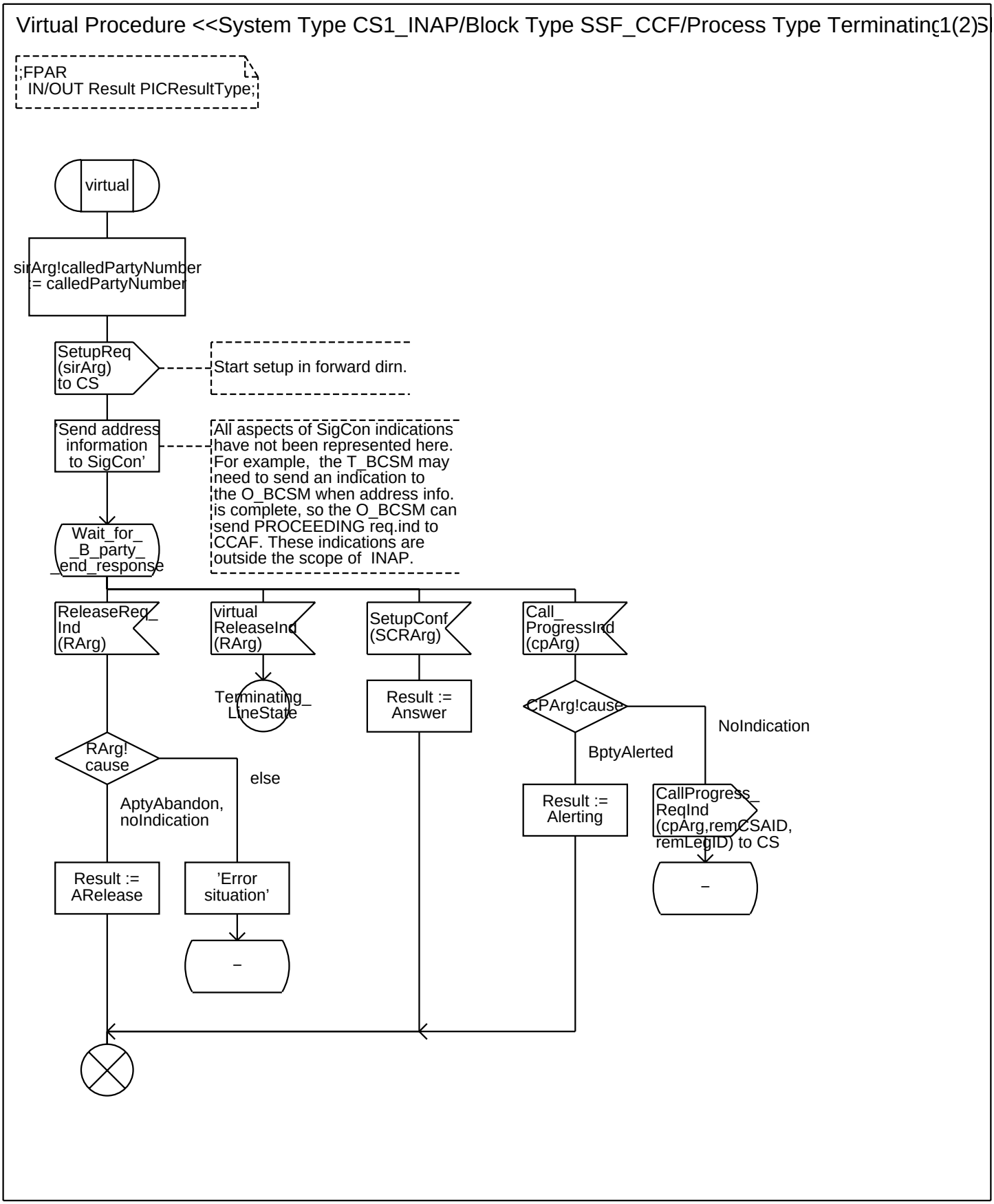
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(1)S
;FPAR
IN/OUT Result PICResultType;
SetupResp_
Conf
(SCRArg,remCSAID,
remLegID) to CS
'Through
connect'
'Provide call
supervision'
Interaction with
O_BCSM is
required
T_Active
Service_
FeatureInd
(sftArg)
FailureInd
(fArg)
ReleaseReq_
Ind
(rArg)
Release_
Ind
(rArg)
virtual
PIC
(pic)
Result :=
MidCall
RArg!cause:=
FailureOf_
Connection
Result :=
Exception
Result :=
ARelease
Result :=
BRelease
PIC!pic
Else
ReleaseCall
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
–

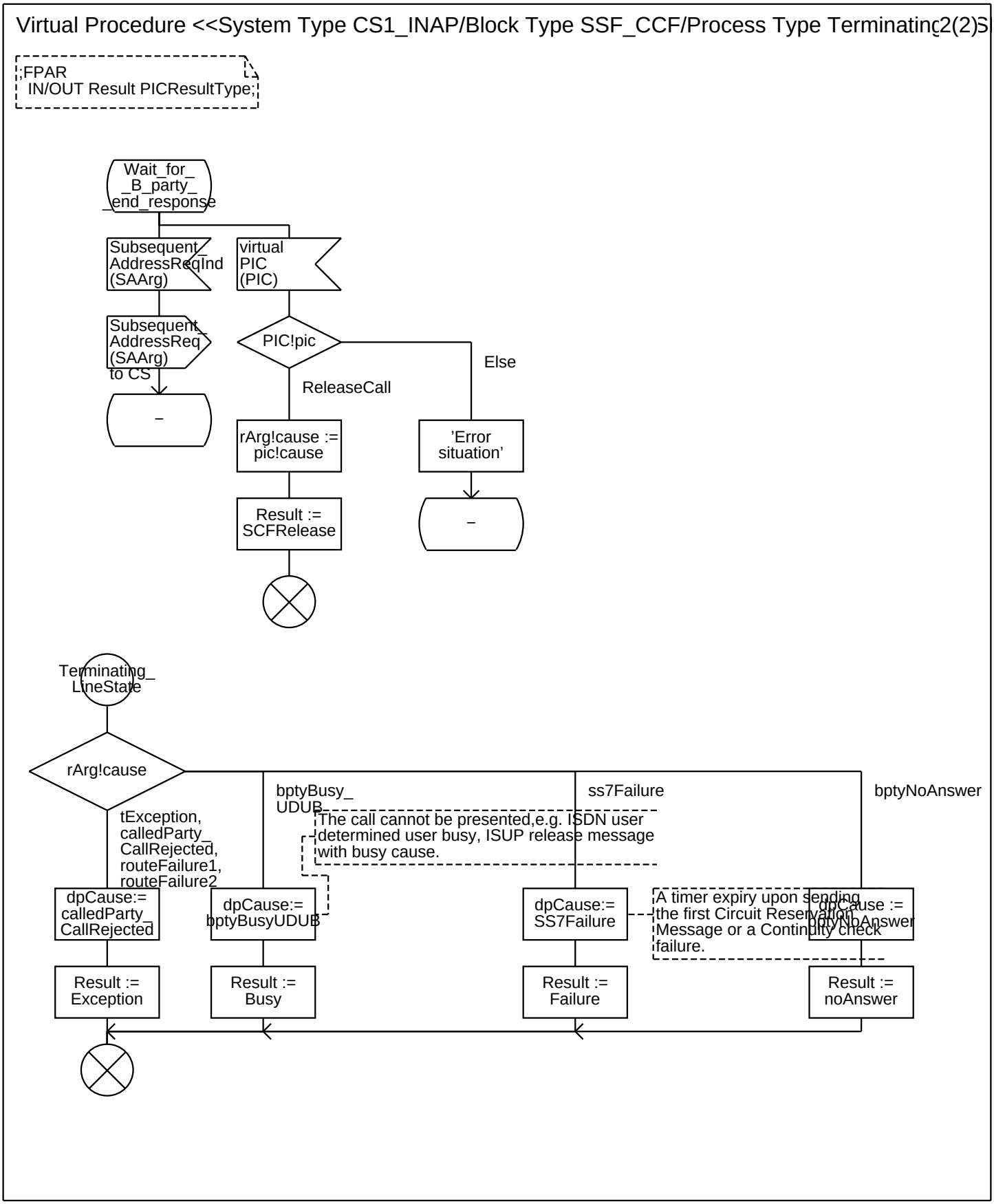
Annex A:


Procedure PIC_Authorize_Termination_Attempt
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 introduction of a state with two inputs: Release 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

Annex A:

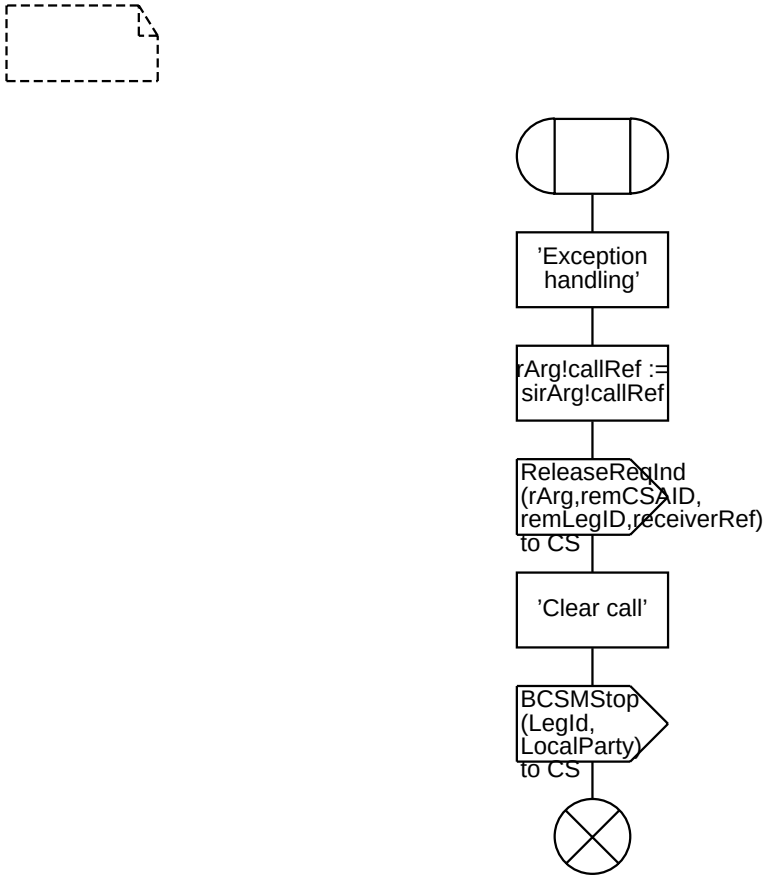

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(2)S
;FPAR
IN/OUT Result PICResultType;
virtual
sirArg!calledPartyNumber
:= calledPartyNumber
SetupReq
(sirArg)
to CS
Start setup in forward dirn.
'Send address
information
to SigCon'
All aspects of SigCon indications
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'
–
–

Annex A:

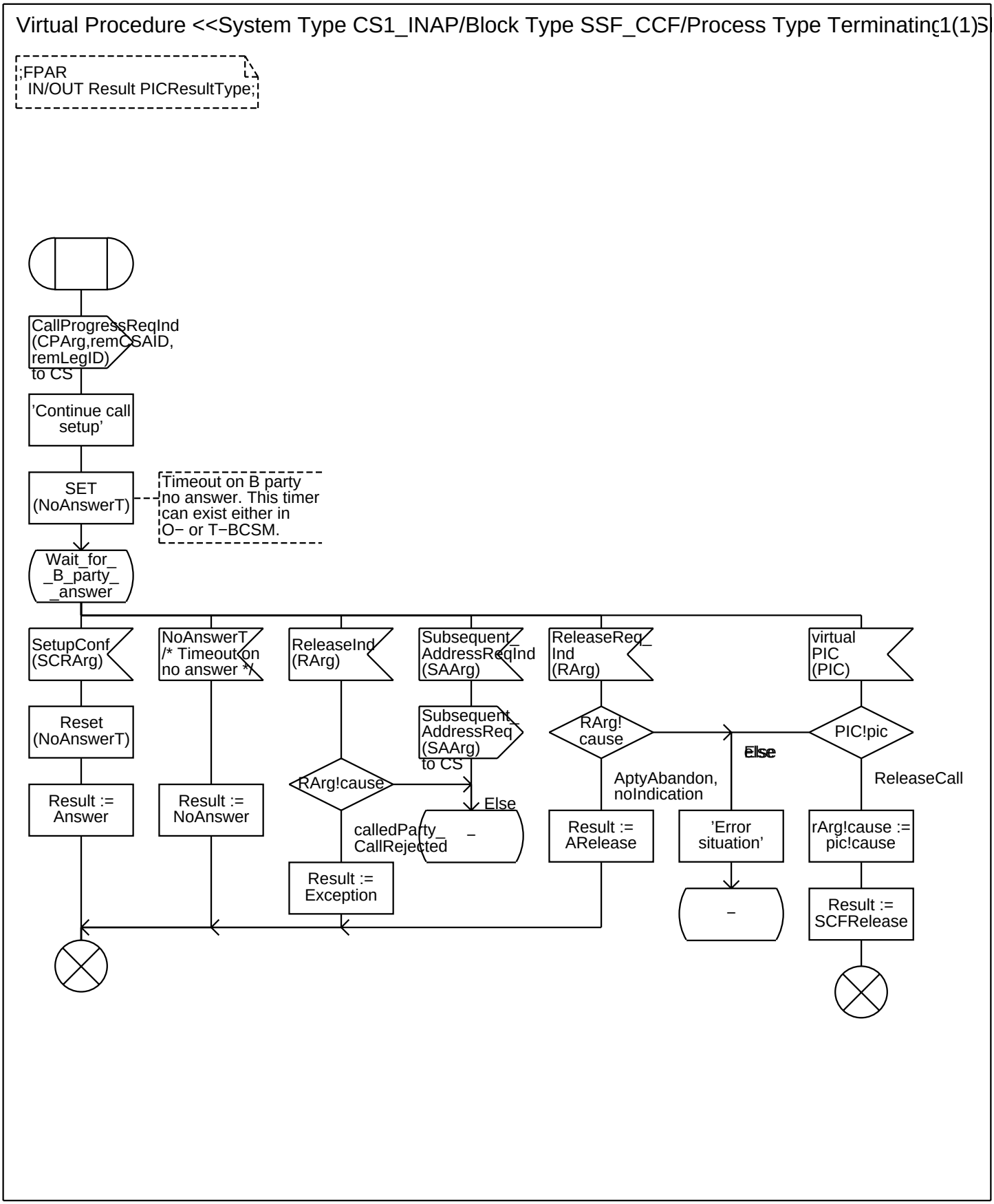

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating2(2)S
;FPAR
IN/OUT Result PICResultType;
Wait_for_
_B_party_
_end_response
Subsequent_
AddressReqInd
(SAArg)
virtual
PIC
(PIC)
Subsequent_
AddressReq
(SAArg)
to CS
PIC!pic
Else
ReleaseCall
–
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
–
Terminating_
LineState
rArg!cause
bptyBusy_
UDUB
ss7Failure
bptyNoAnswer
tException,
calledPartyCallRejected,
routeFailure1,
routeFailure2
The call cannot be presented,e.g. ISDN user
determined user busy, ISUP release message
with busy cause.
dpCause:=
calledPartyCallRejected
dpCause:=
bptyBusyUDUB
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

Annex A:

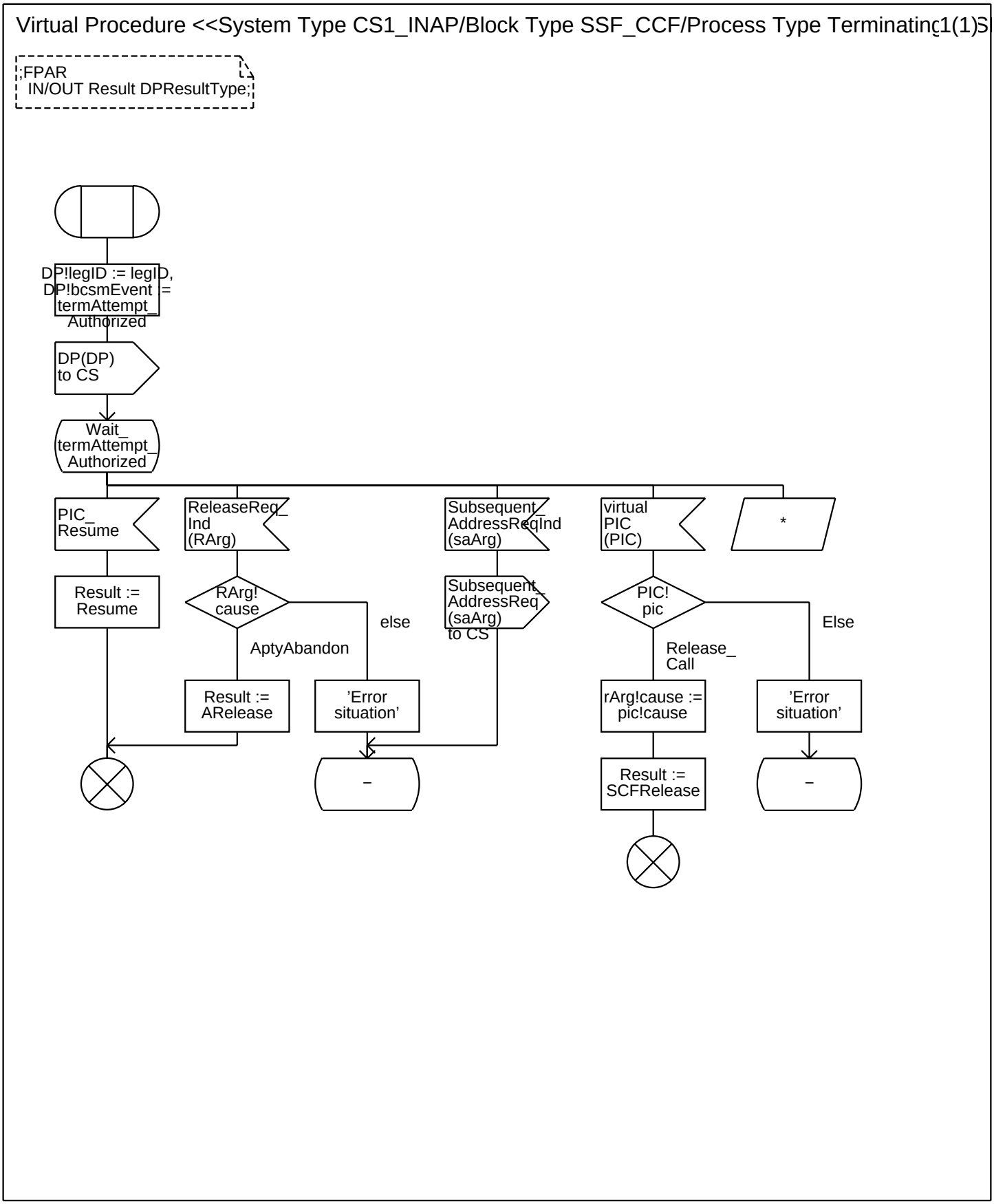
Procedure PIC_TException 1(1)



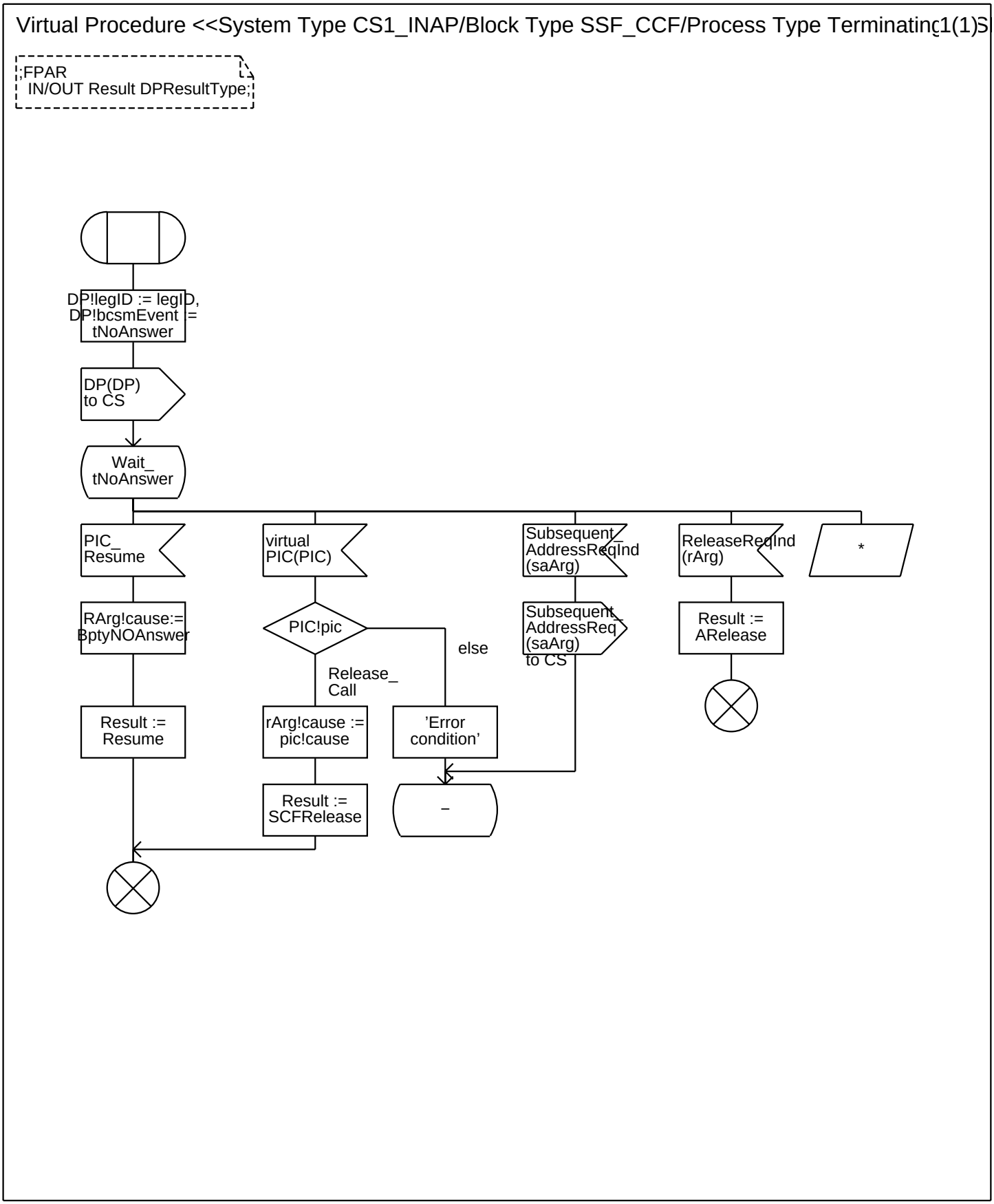
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(1)S
;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
Else
AptyAbandon, noIndication
ReleaseCall
Result := Answer
Result := NoAnswer
calledParty_ CallRejected
Result := ARelease
'Error situation'
rArg!cause := pic!cause
Result := Exception
Result := SCFRelease

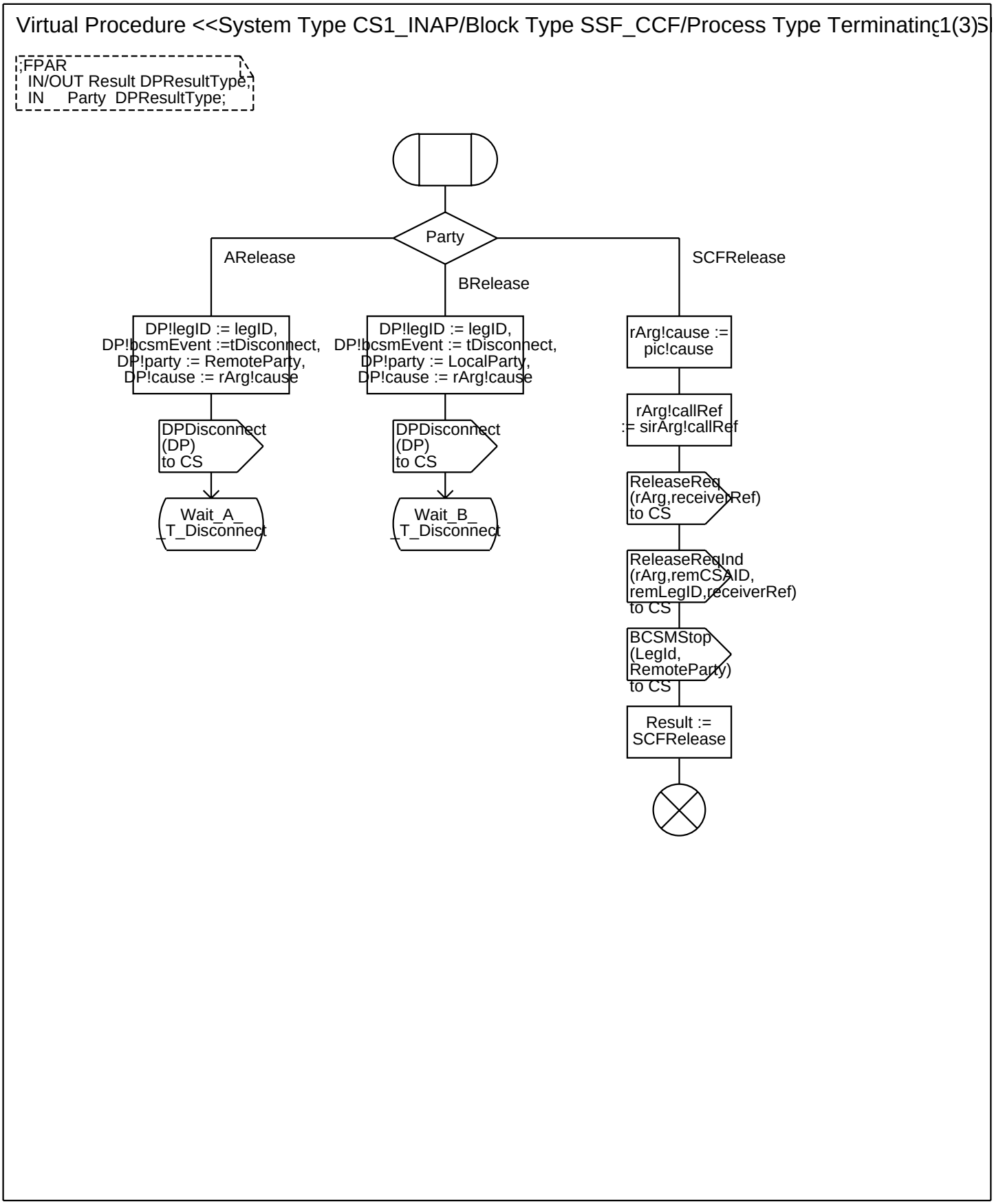
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating1(1)S
;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
–

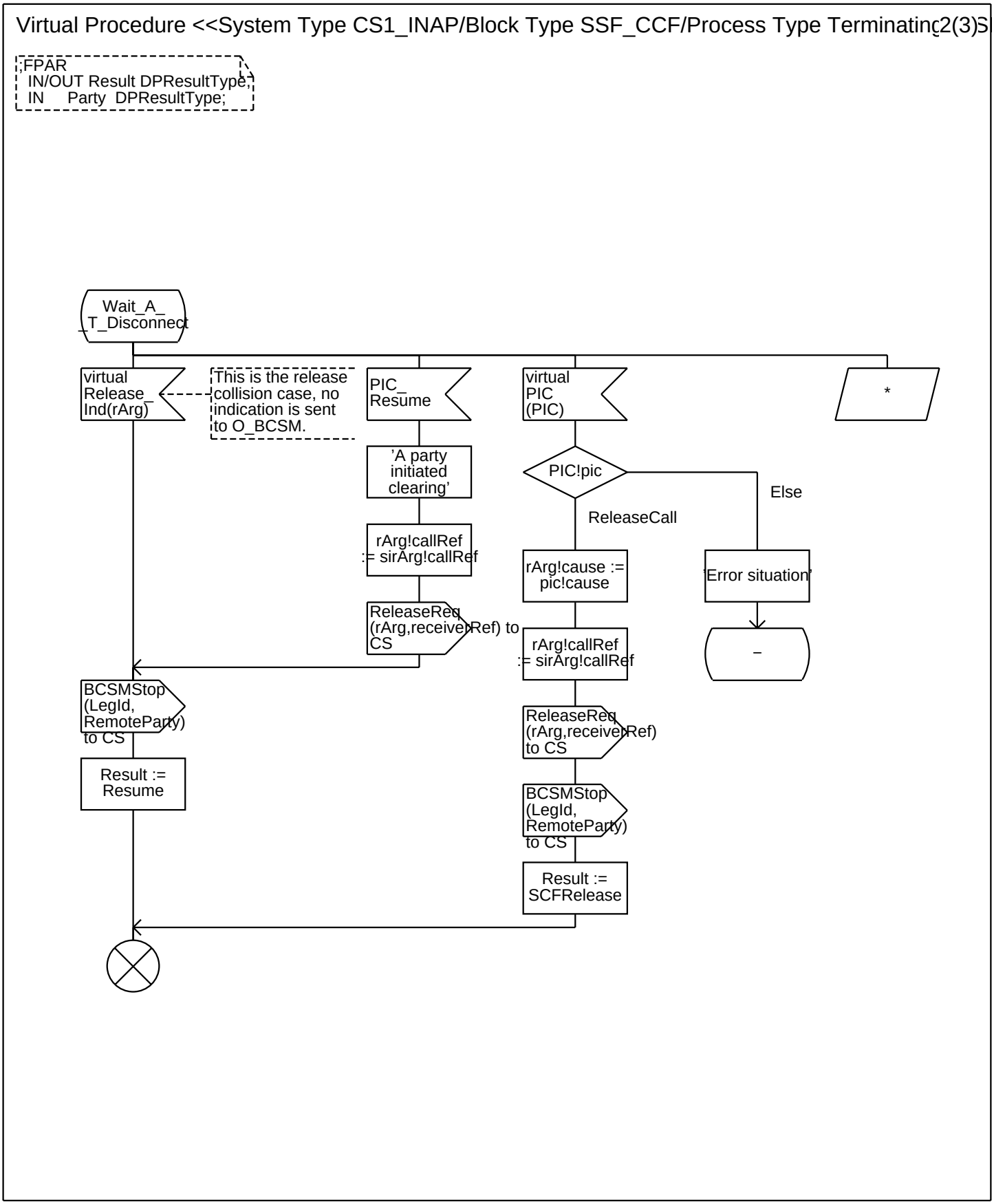
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating1(1)S
;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
Subsequent_
AddressReq
(saArg)
to CS
Result :=
ARelease
Release_
Call
Result :=
Resume
rArg!cause :=
pic!cause
'Error
condition'
Result :=
SCFRelease
–

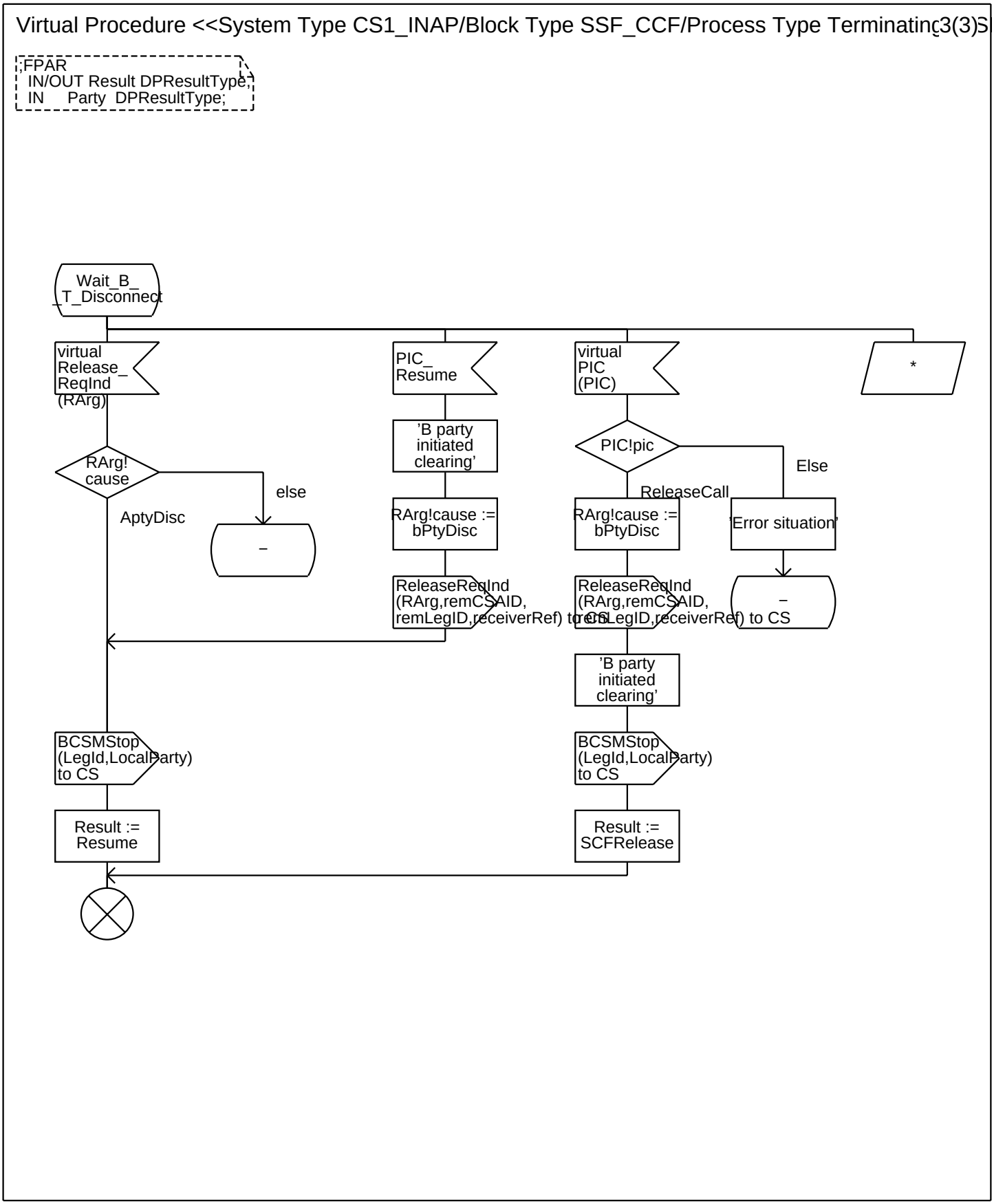
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(3)S
;FPAR
IN/OUT Result DPResultType;
IN Party DPResultType;
Party
ARelease
BRelease
SCFRelease
DP!legID := legID,
DP!bcsmEvent :=tDisconnect,
DP!party := RemoteParty,
DP!cause := rArg!cause
DP!legID := legID,
DP!bcsmEvent := tDisconnect,
DP!party := LocalParty,
DP!cause := rArg!cause
rArg!cause :=
pic!cause
DPDisconnect
(DP)
to CS
DPDisconnect
(DP)
to CS
rArg!callRef
:= sirArg!callRef
ReleaseReq
(rArg,receiverRef)
to CS
Wait_A_
_T_Disconnect
Wait_B_
_T_Disconnect
ReleaseReqInd
(rArg,remCSAID,
remLegID,receiverRef)
to CS
BCSMStop
(LegId,
RemoteParty)
to CS
Result :=
SCFRelease

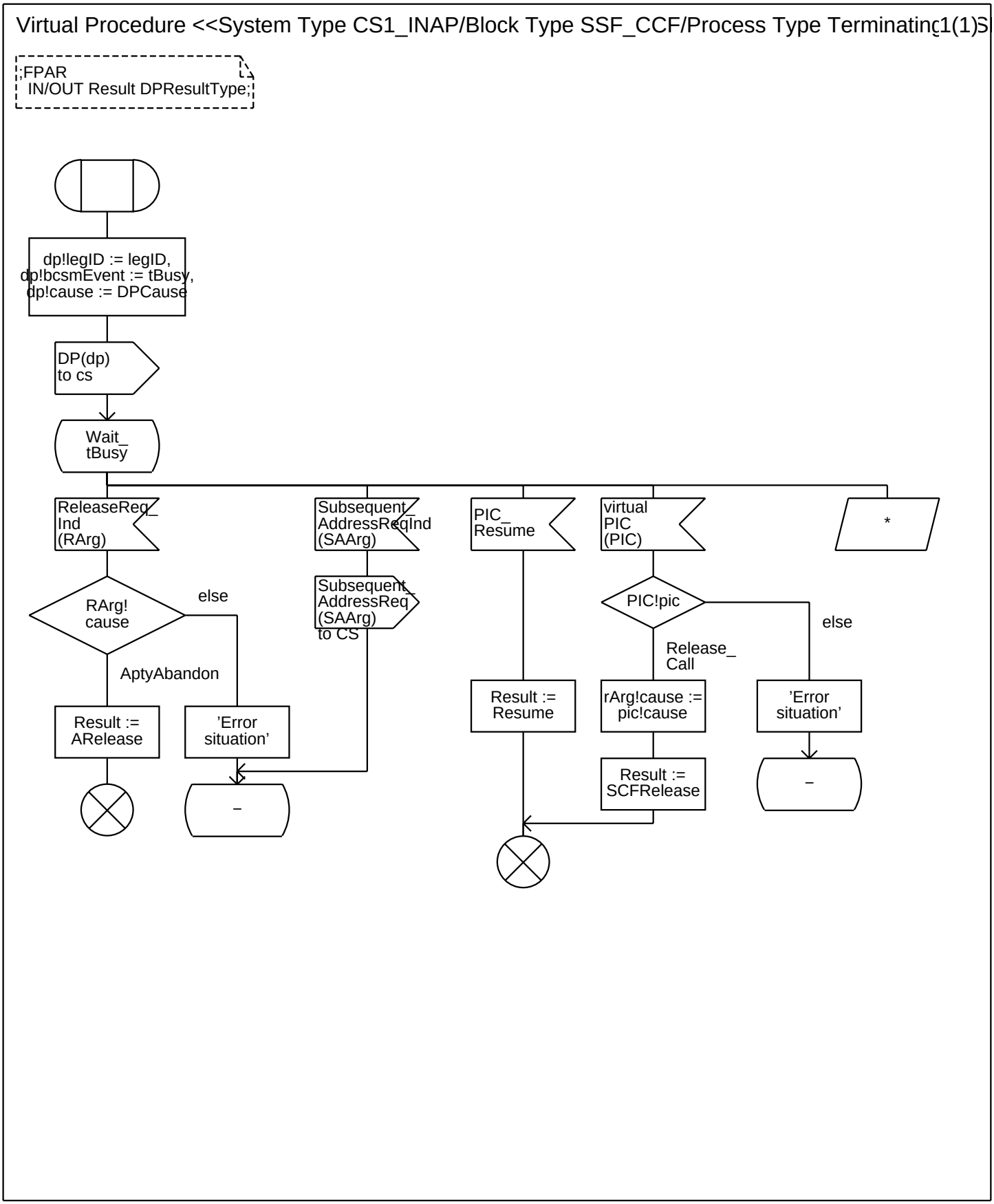
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating2(3)S
;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

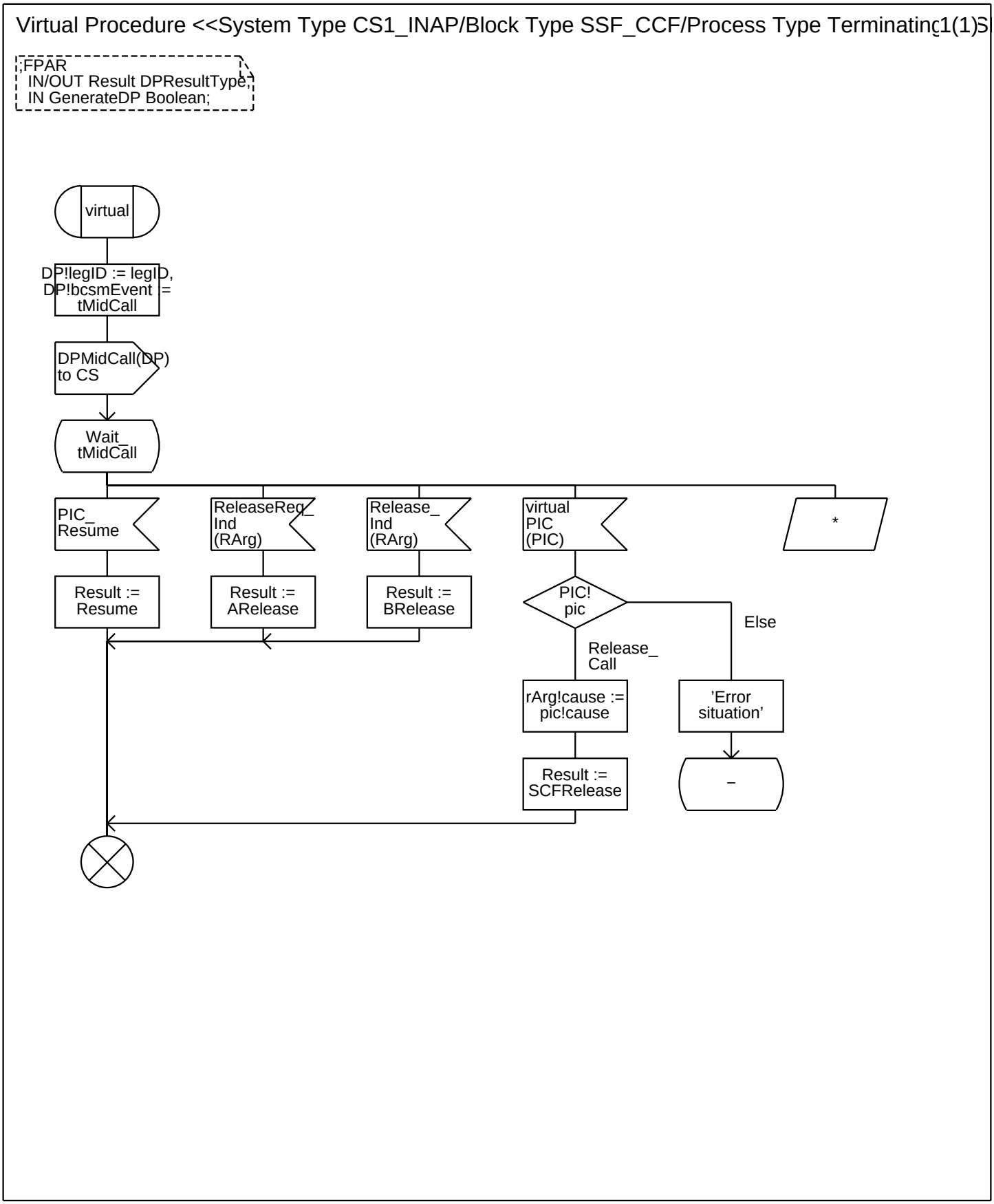
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating3(3)S
;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
RArg!
cause
else
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

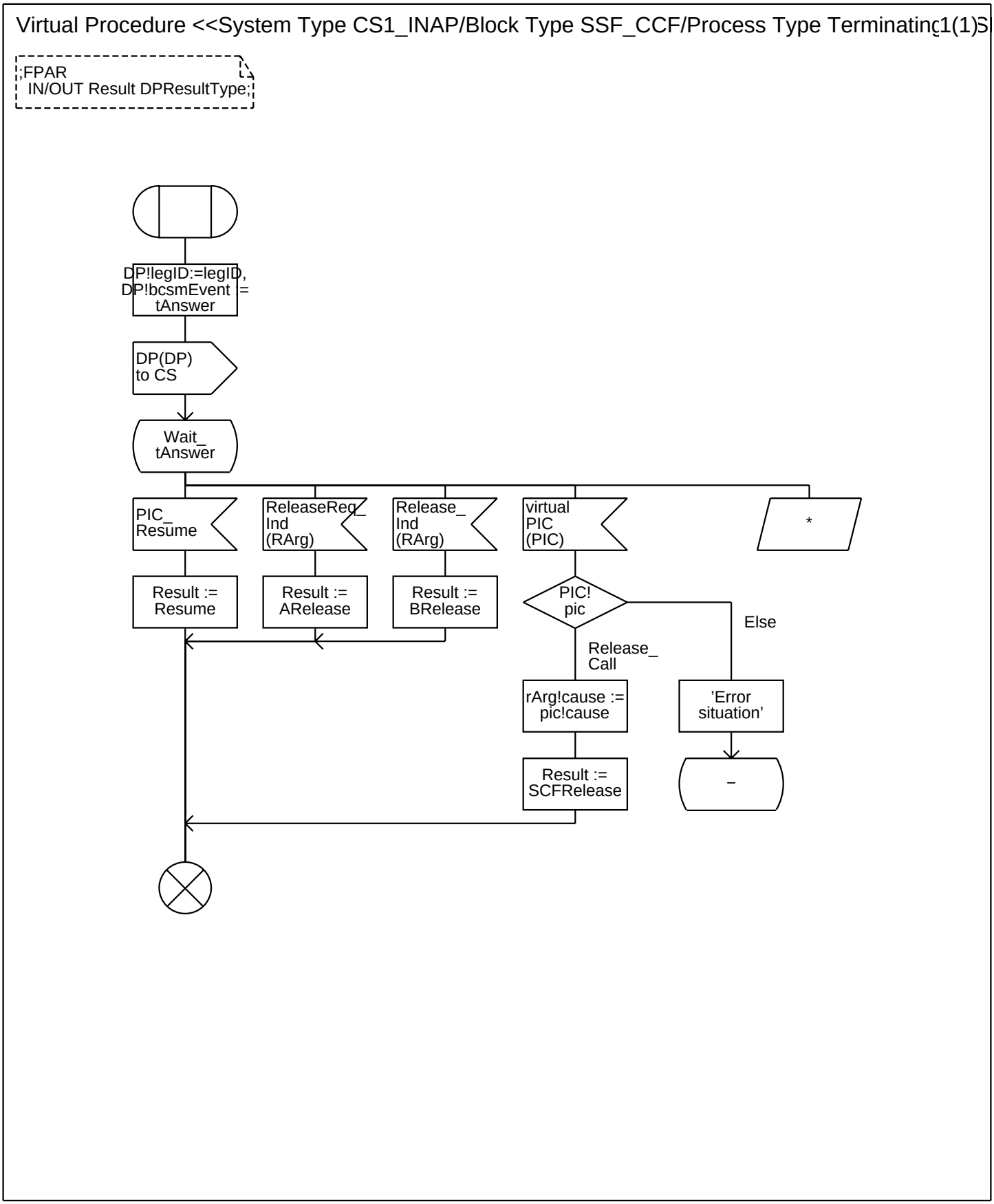
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(1)S
;FPAR
IN/OUT Result DPResultType;
dp!legID := legID,
dp!bcsmEvent := tBusy,
dp!cause := DPCause
DP(dp)
to cs
Wait_
tBusy
ReleaseReq_
Ind
(RArg)
Subsequent_
AddressReqInd
(SAArg)
PIC_
Resume
virtual
PIC
(PIC)
*
RArg!
cause
else
AptyAbandon
Subsequent_
AddressReq
(SAArg)
to CS
PIC!pic
else
Release_
Call
Result :=
ARelease
'Error
situation'
Result :=
Resume
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
–
–

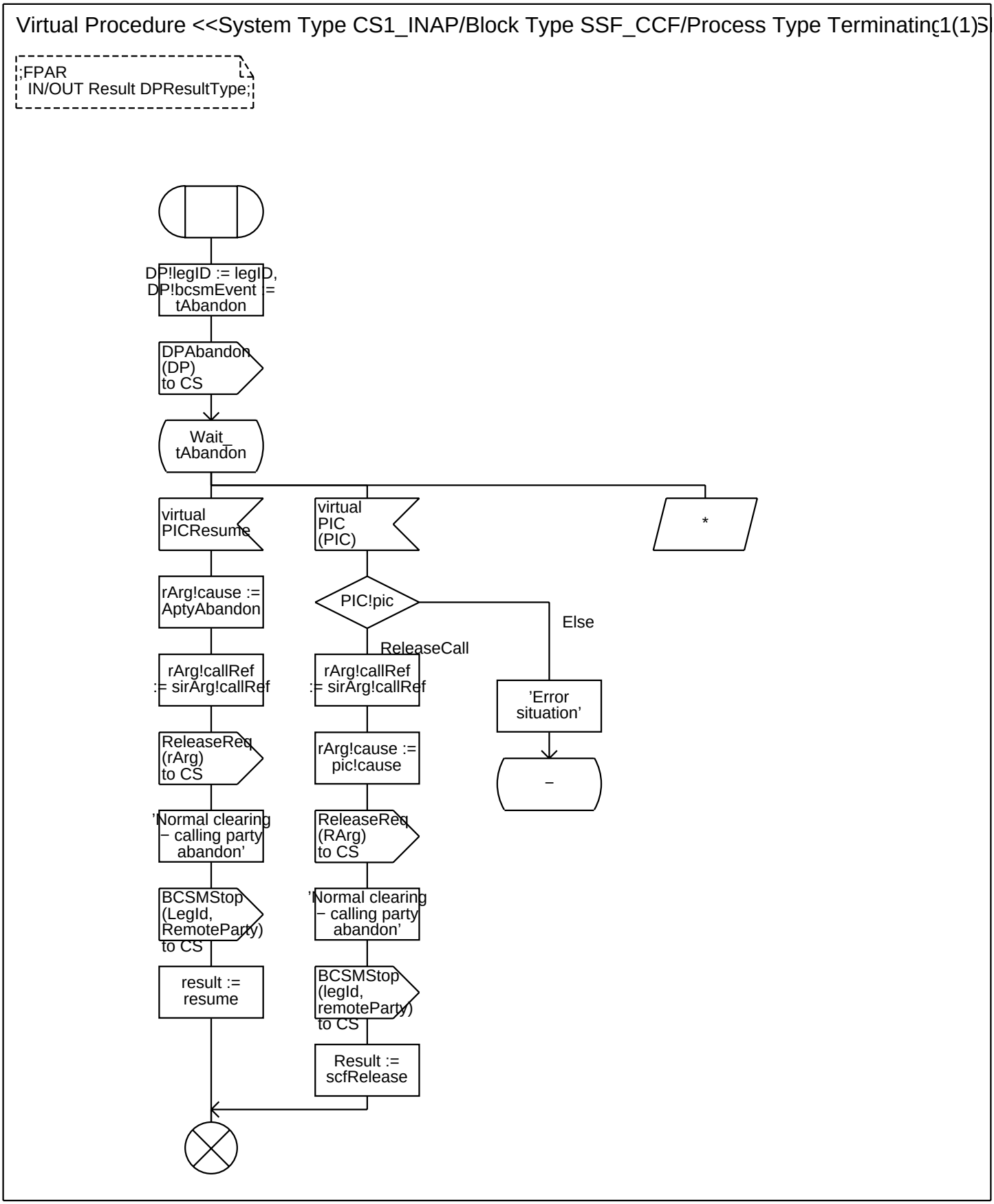
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(1)S
;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
–

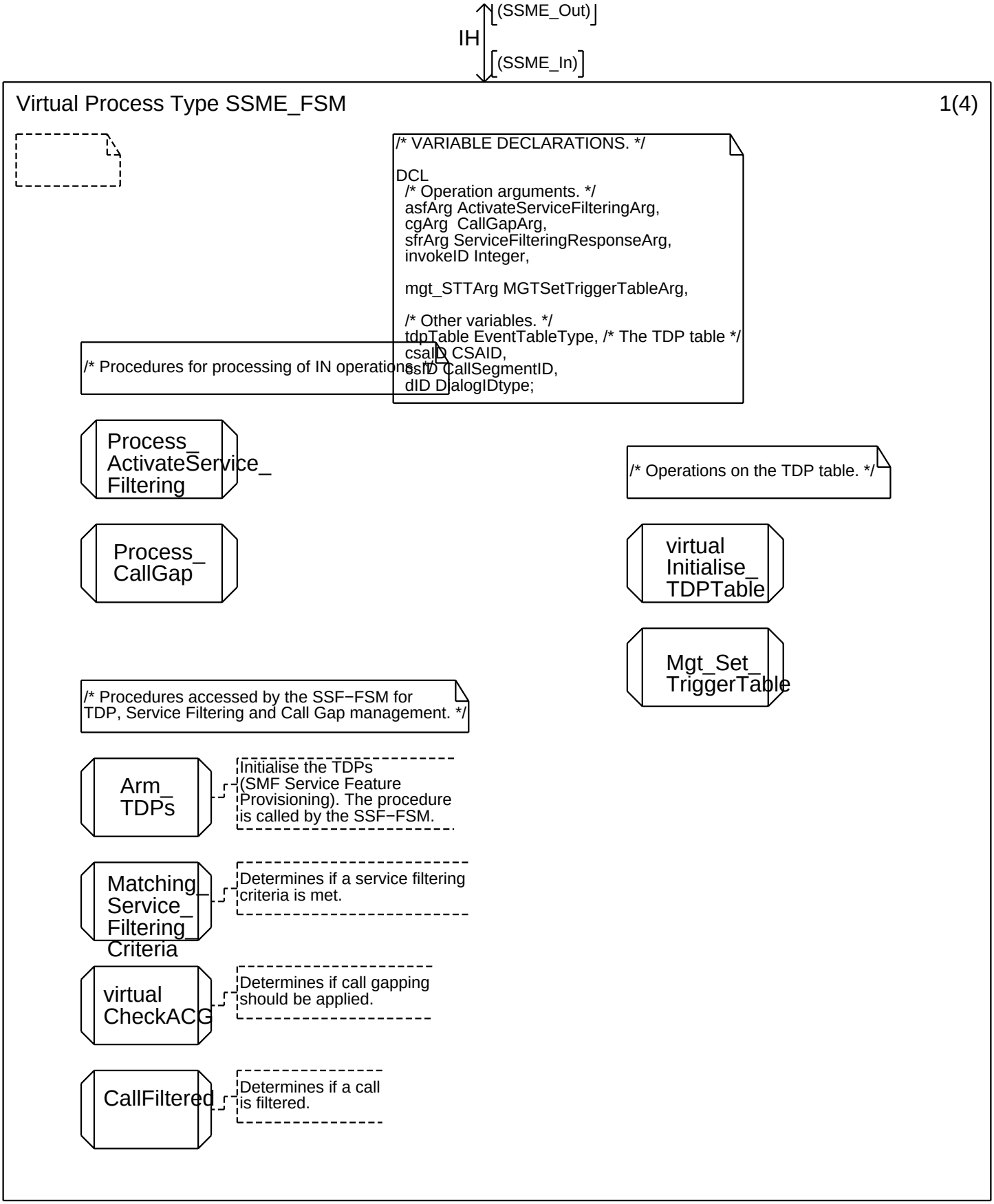
Annex A:


Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(1)S
;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
–

Annex A:

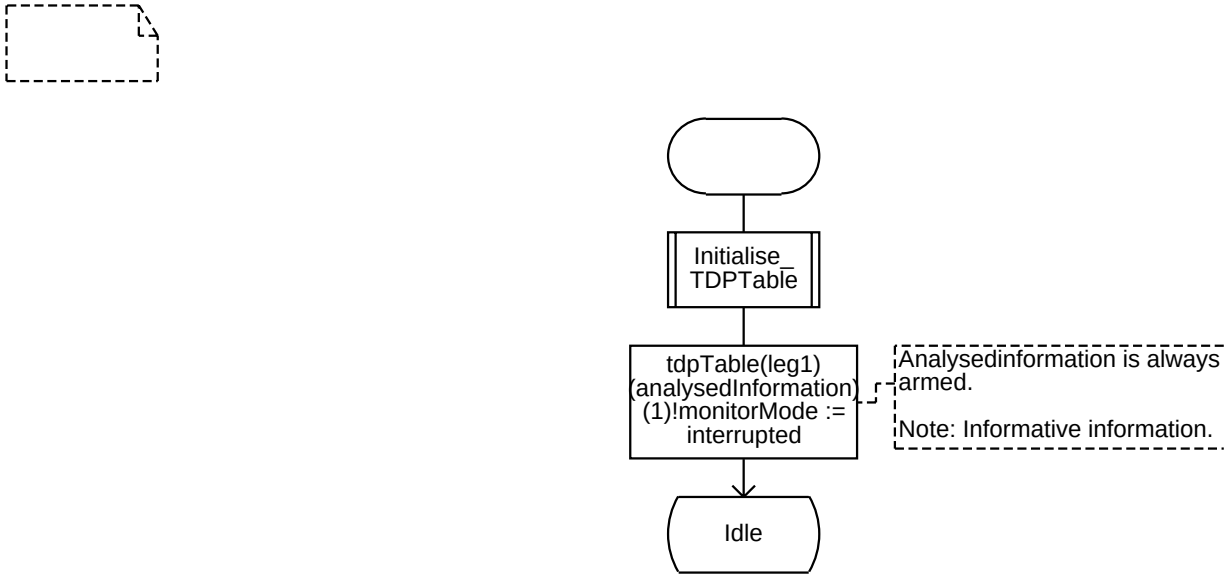

Virtual Procedure <<System Type CS1_INAP/Block Type SSF_CCF/Process Type Terminating 1(1)S
;FPAR
IN/OUT Result DPResultType;
DP!legID := legID,
DP!bcsmEvent =
tAbandon
DPAbandon
(DP)
to CS
Wait_
tAbandon
virtual
PICResume
virtual
PIC
(PIC)
*
rArg!cause :=
AptyAbandon
PIC!pic
Else
ReleaseCall
rArg!callRef
:= 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

Annex A:


(SSME_Out)
IH
(SSME_In)
Virtual Process Type SSME_FSM
1(4)
/* VARIABLE DECLARATIONS. */
DCL
/* Operation arguments. */
asfArg ActivateServiceFilteringArg,
cgArg CallGapArg,
sfrArg ServiceFilteringResponseArg,
invokeID Integer,
mgt_STTArg MGTSetTriggerTableArg,
/* Other variables. */
tdpTable EventTableType, /* The TDP table */
csaID CSAID,
csfID 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.

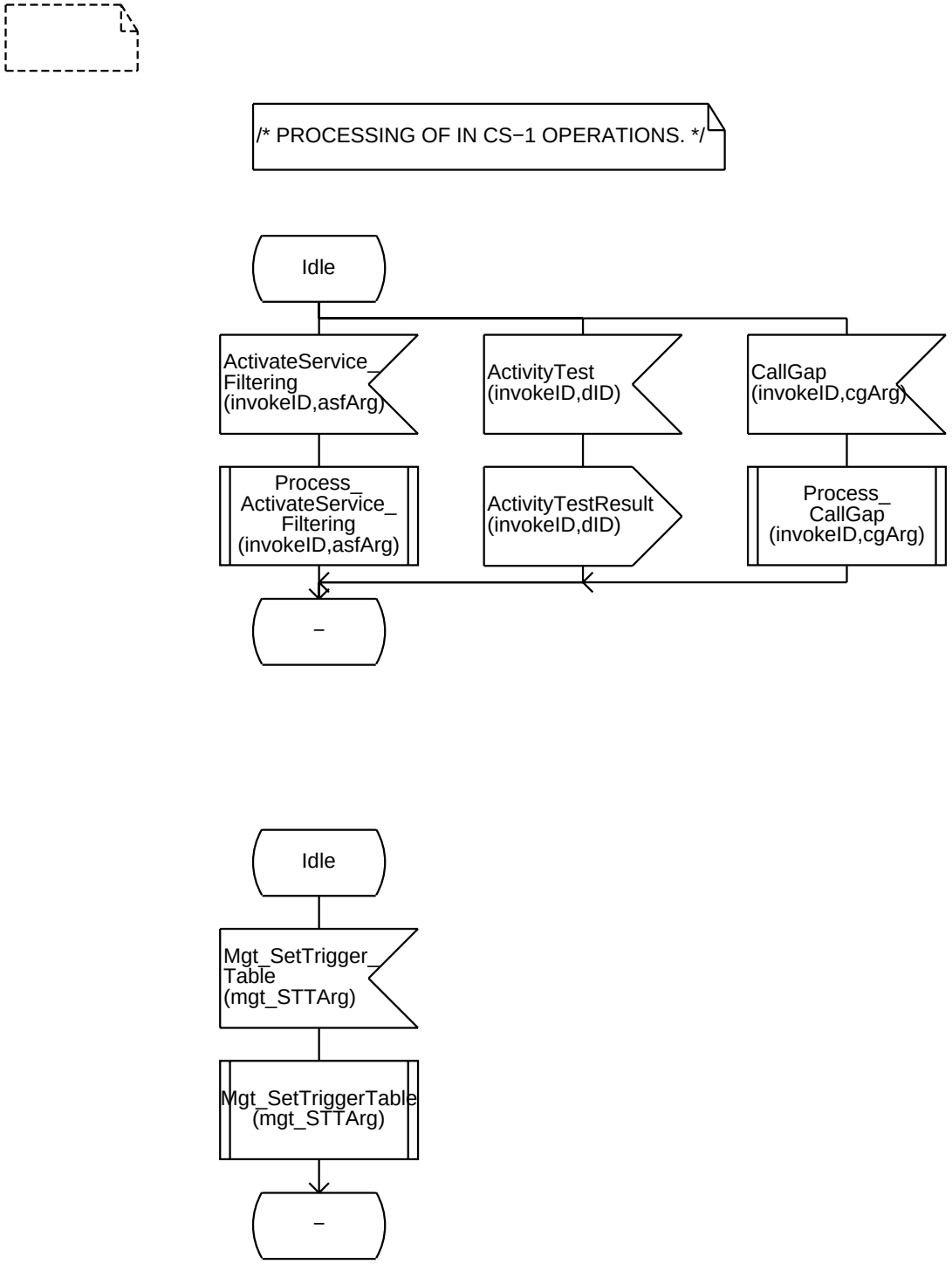
Annex A:

Virtual Process Type SSME_FSM 2(4)



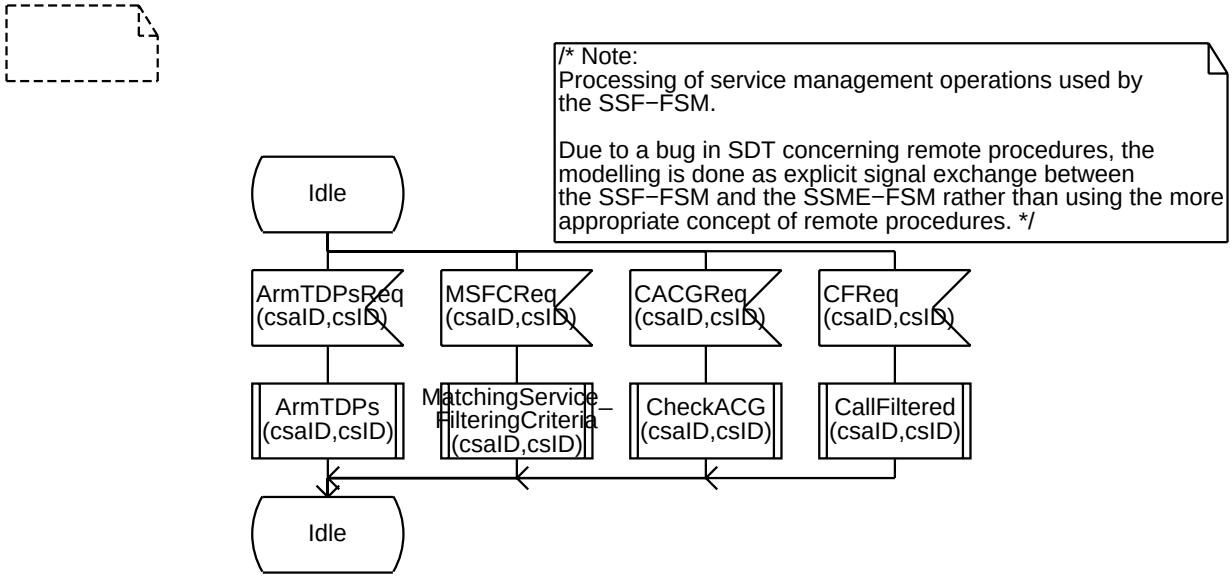
Annex A:

Virtual Process Type SSME_FSM 3(4)



Annex A:

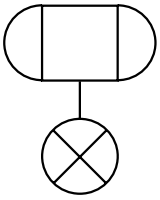
Virtual Process Type SSME_FSM 4(4)



Annex A:

Procedure ProcessActivateServiceFiltering 1(1)

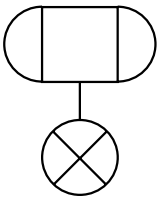
```
;FPAR
 IN invokeID Integer,
 IN asfArg   ActivateServiceFilteringArg;
```



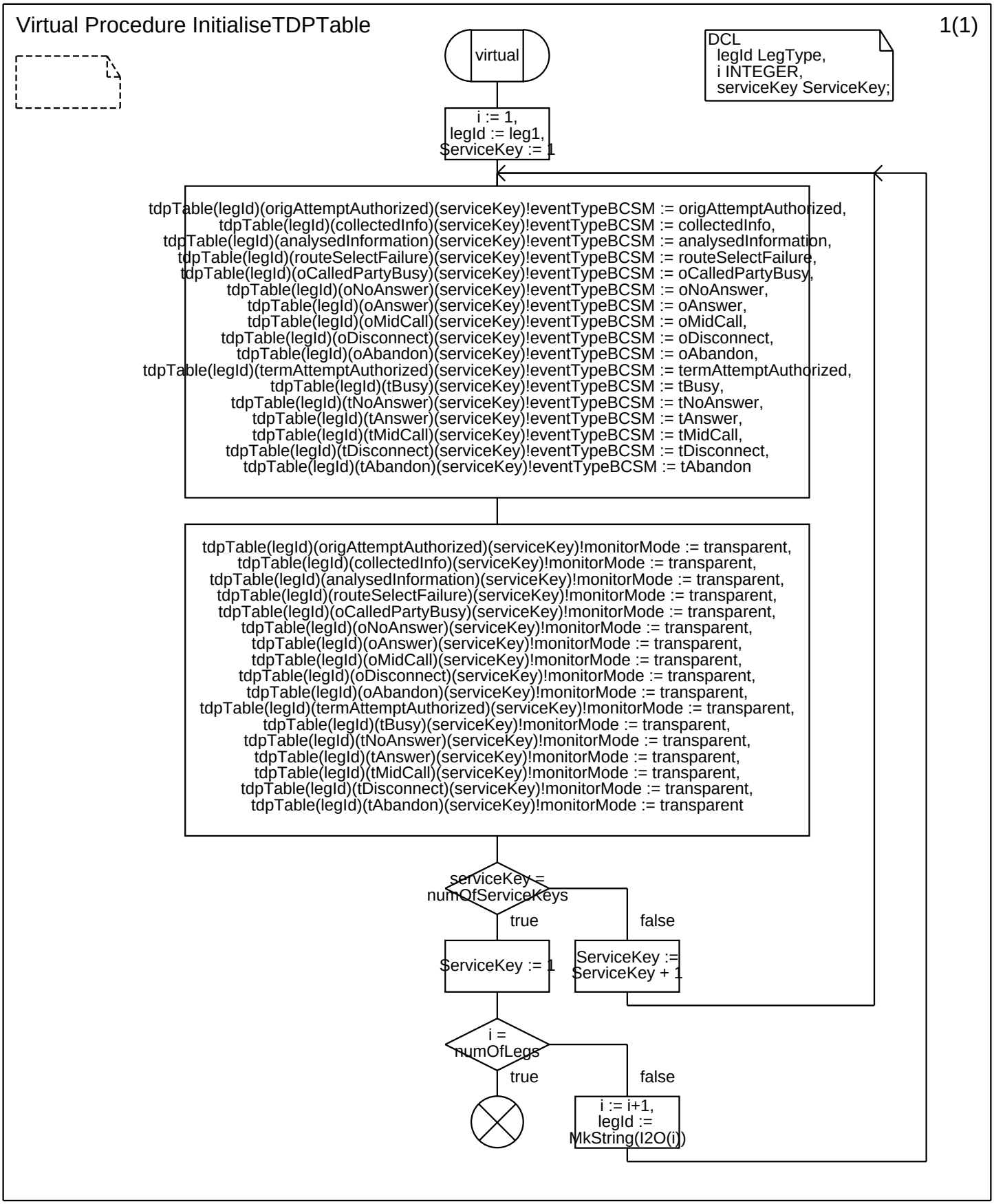
Annex A:

Procedure ProcessCallGap 1(1)

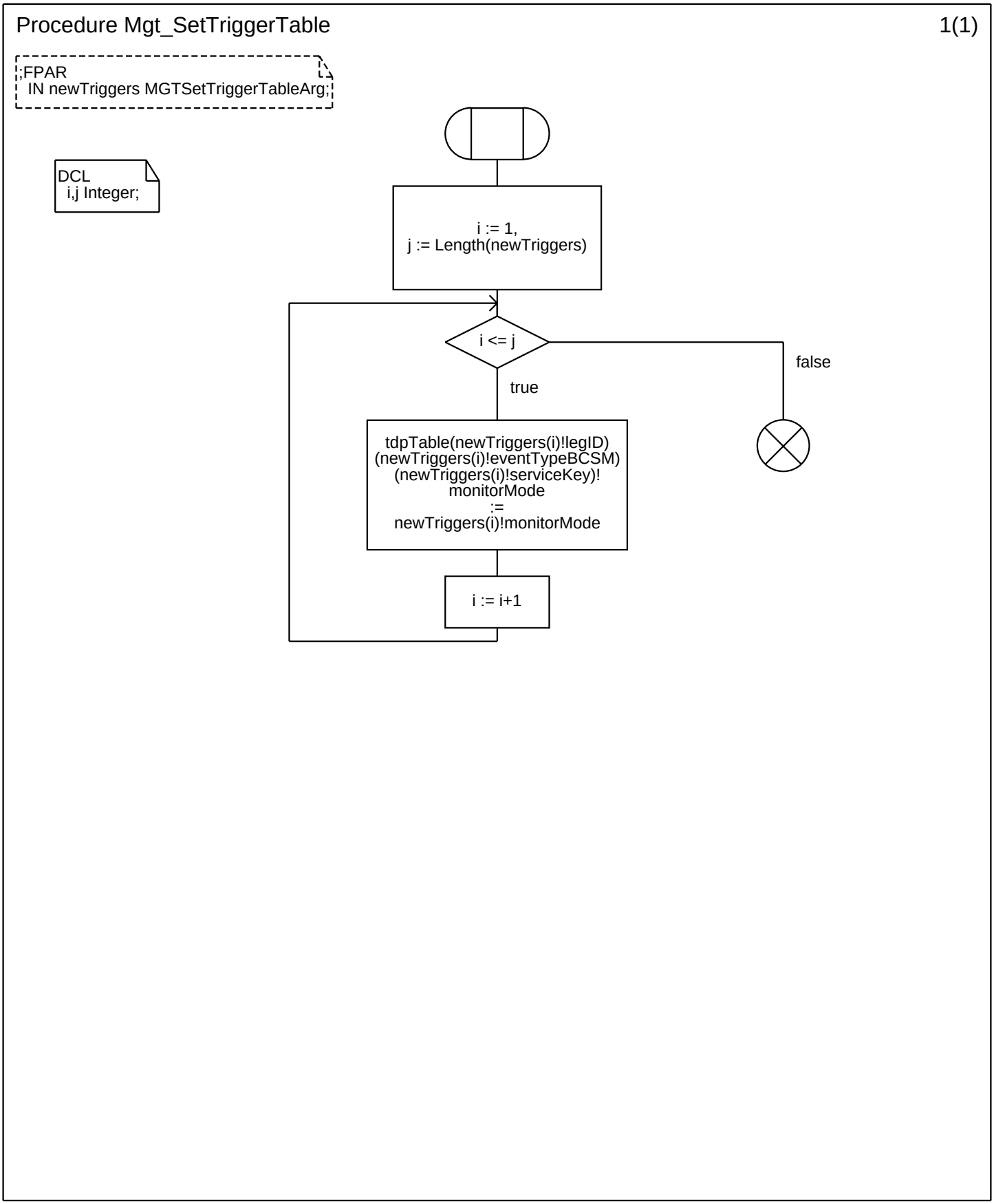
```
;FPAR
 IN invokeID Integer,
 IN cgArg    CallGapArg;
```



Annex A:

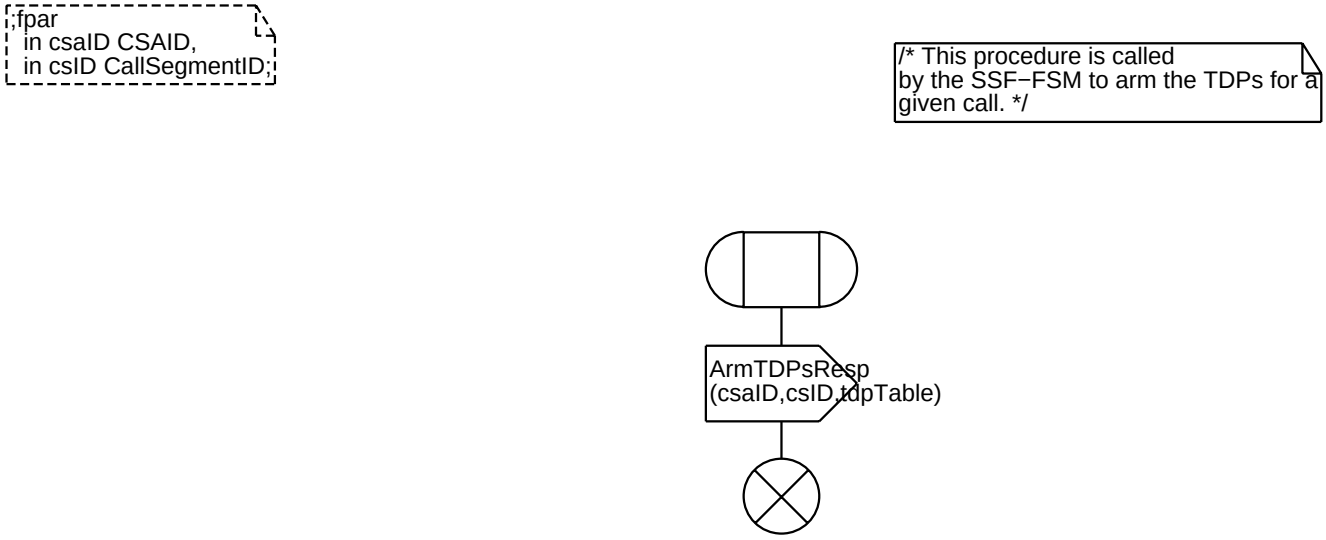

Virtual Procedure InitialiseTDPTable
1(1)
virtual
DCL
legId LegType,
i INTEGER,
serviceKey ServiceKey;
i := 1,
legId := leg1,
ServiceKey := 1
tdpTable(legId)(origAttemptAuthorized)(serviceKey)!eventTypeBCSM := origAttemptAuthorized,
tdpTable(legId)(collectedInfo)(serviceKey)!eventTypeBCSM := collectedInfo,
tdpTable(legId)(analysedInformation)(serviceKey)!eventTypeBCSM := analysedInformation,
tdpTable(legId)(routeSelectFailure)(serviceKey)!eventTypeBCSM := routeSelectFailure,
tdpTable(legId)(oCalledPartyBusy)(serviceKey)!eventTypeBCSM := oCalledPartyBusy,
tdpTable(legId)(oNoAnswer)(serviceKey)!eventTypeBCSM := oNoAnswer,
tdpTable(legId)(oAnswer)(serviceKey)!eventTypeBCSM := oAnswer,
tdpTable(legId)(oMidCall)(serviceKey)!eventTypeBCSM := oMidCall,
tdpTable(legId)(oDisconnect)(serviceKey)!eventTypeBCSM := oDisconnect,
tdpTable(legId)(oAbandon)(serviceKey)!eventTypeBCSM := oAbandon,
tdpTable(legId)(termAttemptAuthorized)(serviceKey)!eventTypeBCSM := termAttemptAuthorized,
tdpTable(legId)(tBusy)(serviceKey)!eventTypeBCSM := tBusy,
tdpTable(legId)(tNoAnswer)(serviceKey)!eventTypeBCSM := tNoAnswer,
tdpTable(legId)(tAnswer)(serviceKey)!eventTypeBCSM := tAnswer,
tdpTable(legId)(tMidCall)(serviceKey)!eventTypeBCSM := tMidCall,
tdpTable(legId)(tDisconnect)(serviceKey)!eventTypeBCSM := tDisconnect,
tdpTable(legId)(tAbandon)(serviceKey)!eventTypeBCSM := tAbandon
tdpTable(legId)(origAttemptAuthorized)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(collectedInfo)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(analysedInformation)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(routeSelectFailure)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oCalledPartyBusy)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oNoAnswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oAnswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oMidCall)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oDisconnect)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oAbandon)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(termAttemptAuthorized)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tBusy)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tNoAnswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tAnswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tMidCall)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tDisconnect)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tAbandon)(serviceKey)!monitorMode := transparent
serviceKey = numOfServiceKeys
true
false
ServiceKey := 1
ServiceKey := ServiceKey + 1
i = numOfLegs
true
false
i := i+1,
legId := MkString(I2O(i))

Annex A:


Procedure Mgt_SetTriggerTable
1(1)
;FPAR
IN newTriggers MGTSetTriggerTableArg;
DCL
i,j Integer;
i := 1,
j := Length(newTriggers)
i <= j
true
false
tdpTable(newTriggers(i)!legID)
(newTriggers(i)!eventTypeBCSM)
(newTriggers(i)!serviceKey)!
monitorMode
:=
newTriggers(i)!monitorMode
i := i+1

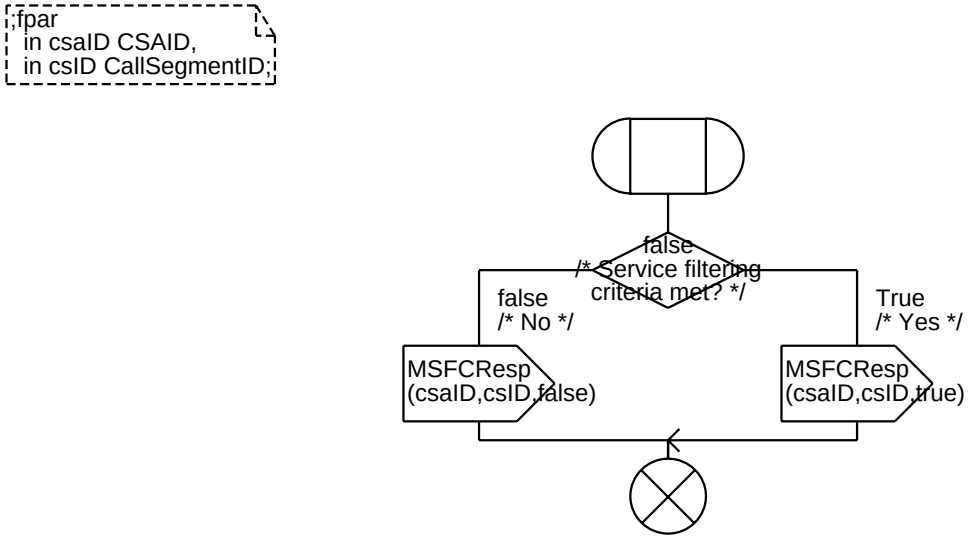
Annex A:

Procedure <<Process Type SSME_FSM>> ArmTDPs 1(1)



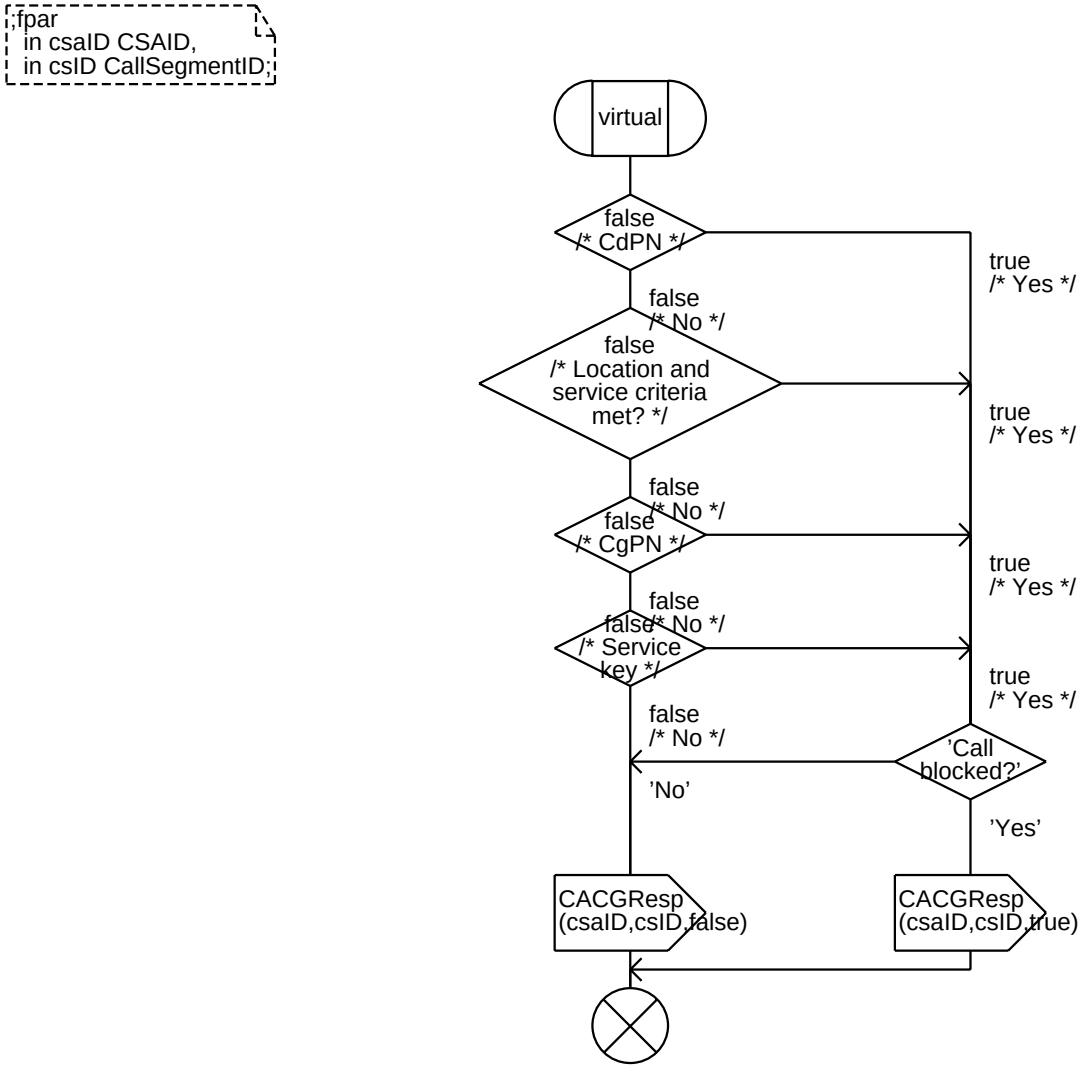
Annex A:

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



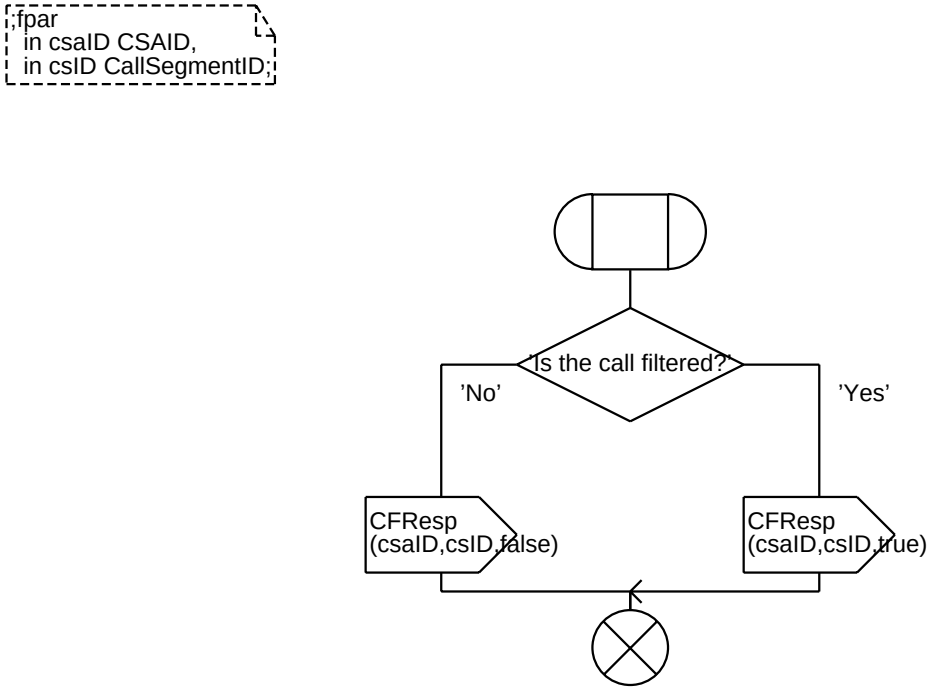
Annex A:

Virtual Procedure <<Package CS1_INAP/System Type CS1_INAP/Block Type SSF_CCF/Pro1(1);

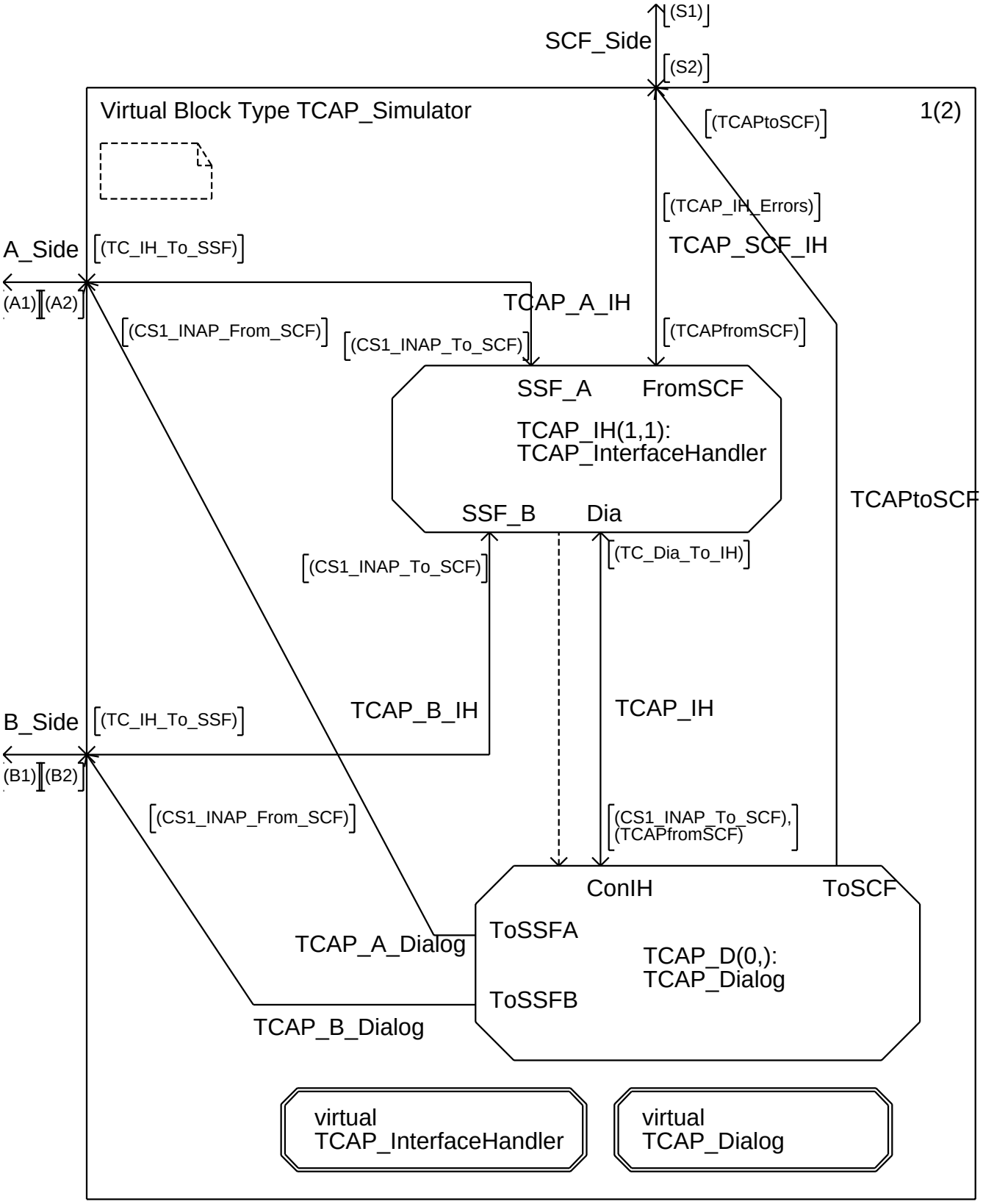


Annex A:

Procedure <<Process Type SSME_FSM>> CallFiltered 1(1)



Annex A:


(S1)
SCF_Side
(S2)
Virtual Block Type TCAP_Simulator
1(2)
(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

Annex A:

Virtual Block Type TCAP_Simulator 2(2)

```
/* 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;
```

Annex A:

Virtual Process Type TCAP_InterfaceHandler 1(12)

```
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;
```

operator
newDialog

operator
initDialogs

operator
getDialogID

operator
nextFreeDialogID

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

Annex A:

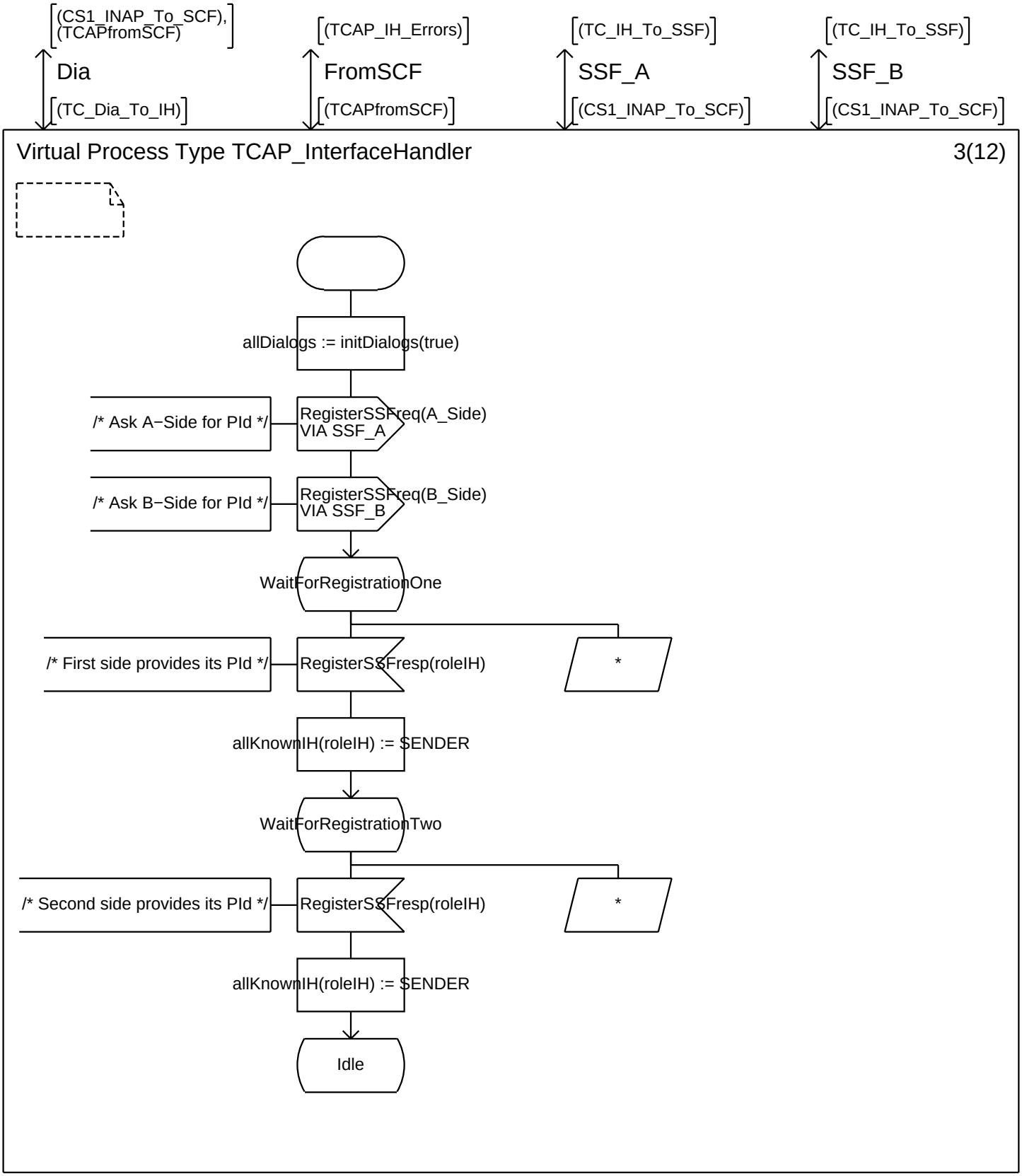
Virtual Process Type TCAP_InterfaceHandler 2(12)

```
DCL allKnownIH IHresourceType;
DCL roleIH IHroleType;

DCL allDialogs AllDialogsType;
```

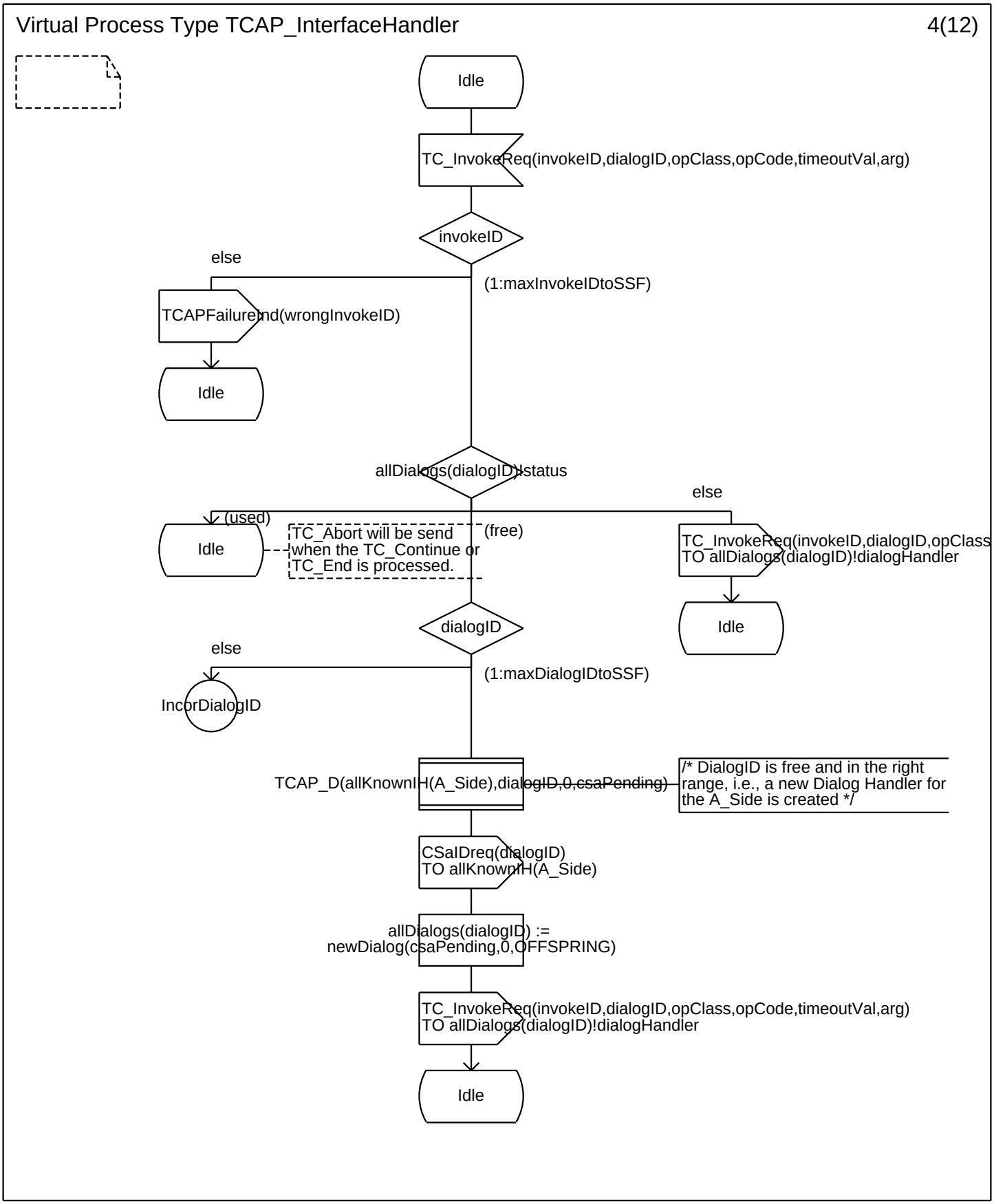
```
DCL invokeID InvokeIDtype;
DCL dialogID DialogIDtype;
DCL opClass OpClassType;
DCL opCode OpCodeType;
DCL timeoutVal TimeoutValType;
```

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

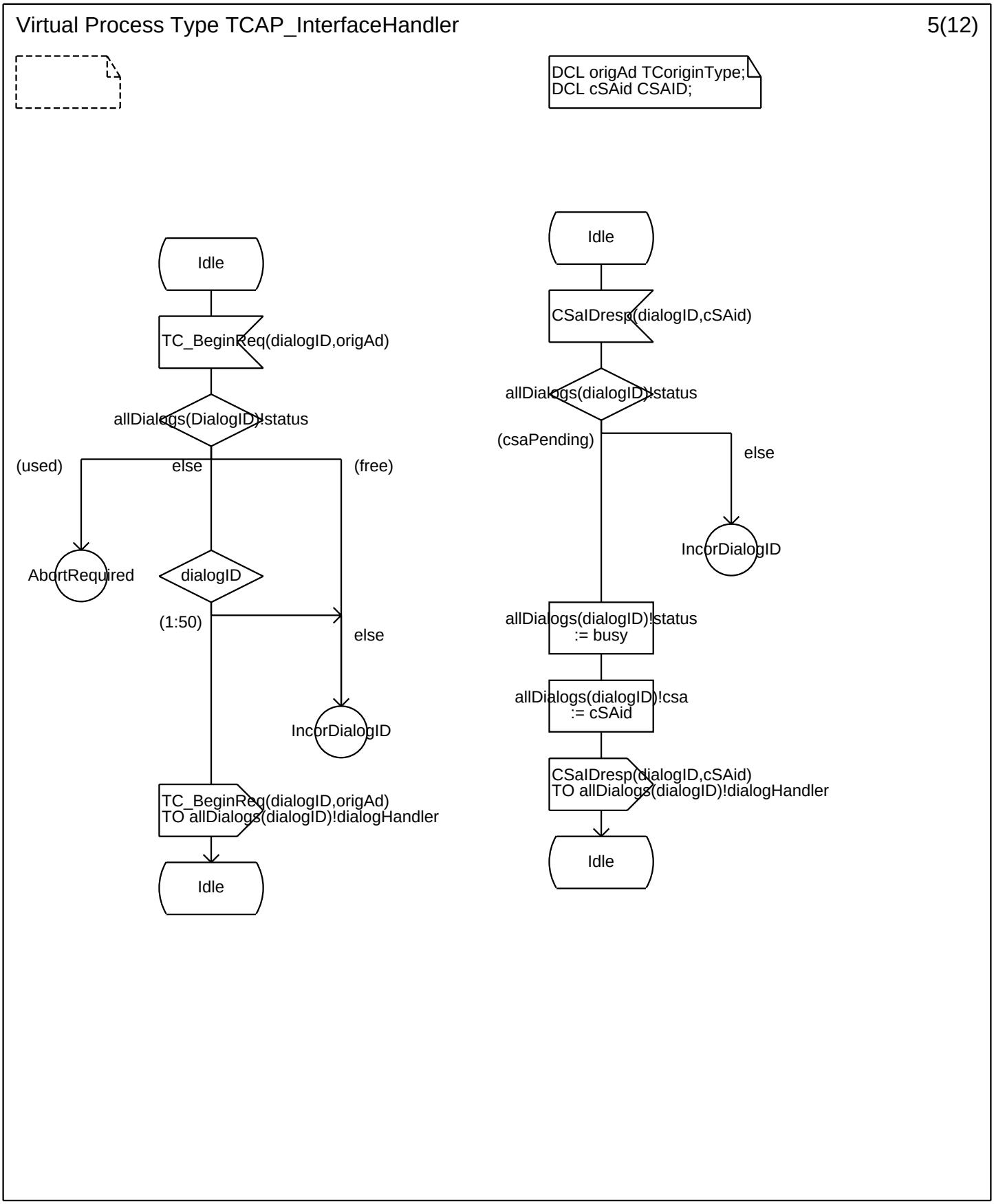
Annex A:


(CS1_INAP_To_SCF),
(TCAPfromSCF)
Dia
(TC_Dia_To_IH)
(TCAP_IH_Errors)
FromSCF
(TCAPfromSCF)
(TC_IH_To_SSF)
SSF_A
(CS1_INAP_To_SCF)
(TC_IH_To_SSF)
SSF_B
(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

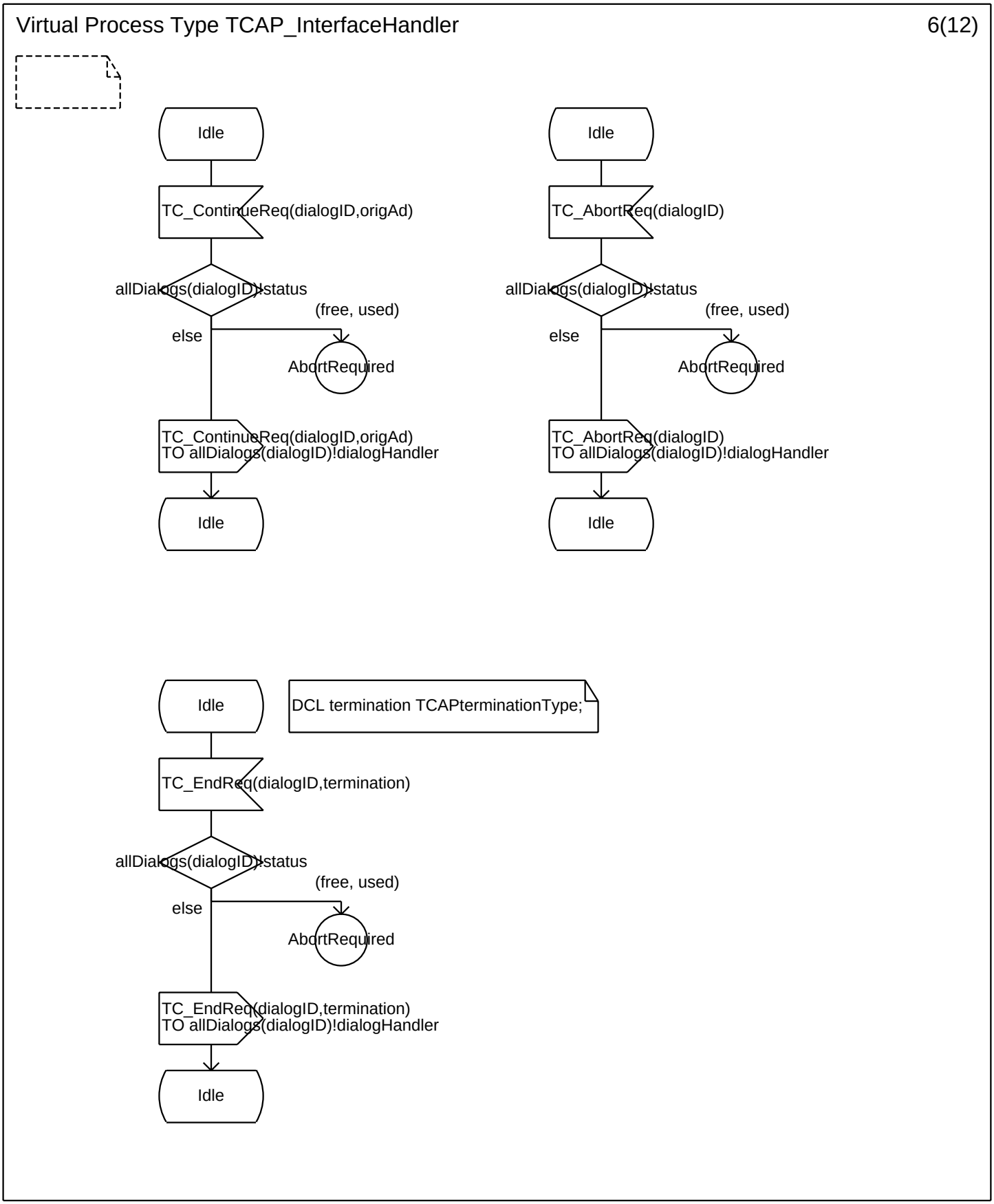
Annex A:


Virtual Process Type TCAP_InterfaceHandler
4(12)
Idle
TC_InvokeReq(invokeID,dialogID,opClass,opCode,timeoutVal,arg)
invokeID
else
(1:maxInvokeIDtoSSF)
TCAPFailureInd(wrongInvokeID)
Idle
allDialogs(dialogID)!status
else
(used)
Idle
TC_Abort will be send
when the TC_Continue or
TC_End is processed.
(free)
TC_InvokeReq(invokeID,dialogID,opClass
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

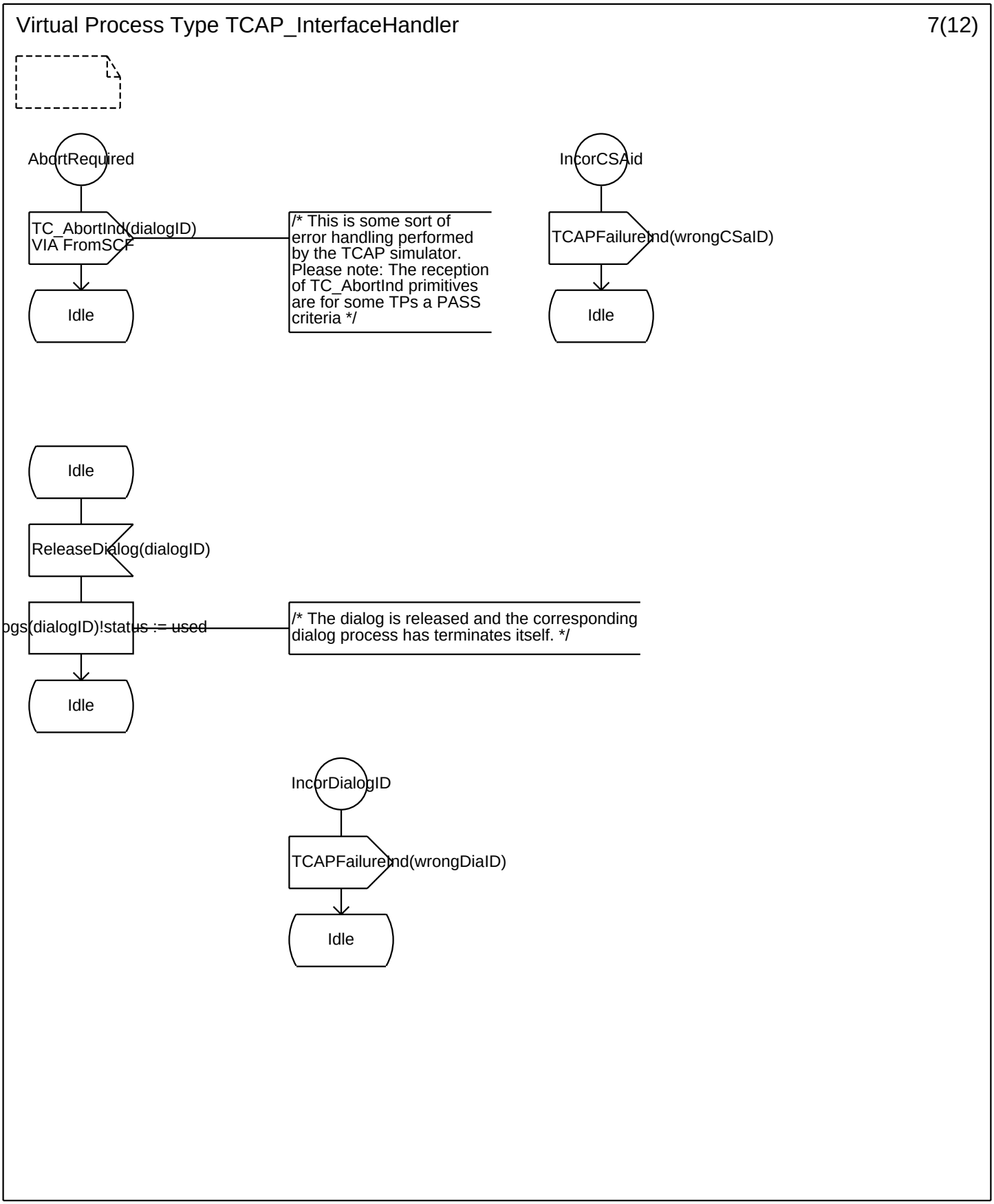
Annex A:


Virtual Process Type TCAP_InterfaceHandler
5(12)
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

Annex A:

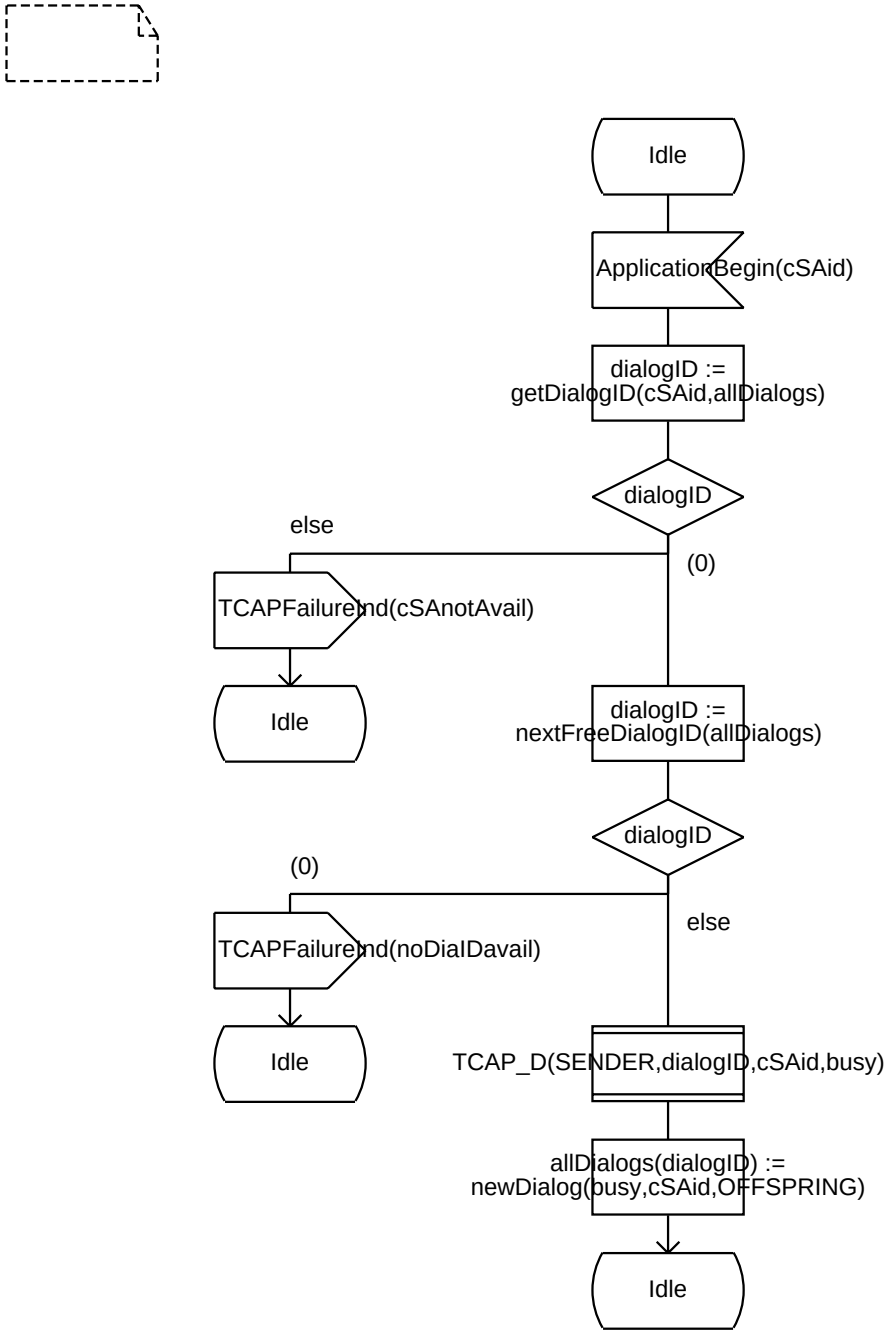

Virtual Process Type TCAP_InterfaceHandler
6(12)
Idle
TC_ContinueReq(dialogID,origAd)
allDialogs(dialogID)!status
(free, used)
else
AbortRequired
TC_ContinueReq(dialogID,origAd)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
TC_AbortReq(dialogID)
allDialogs(dialogID)!status
(free, used)
else
AbortRequired
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

Annex A:


Virtual Process Type TCAP_InterfaceHandler
7(12)
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)
ogs(dialogID)!status := used
/* The dialog is released and the corresponding dialog process has terminates itself. */
Idle
IncorDialogID
TCAPFailureInd(wrongDiaID)
Idle

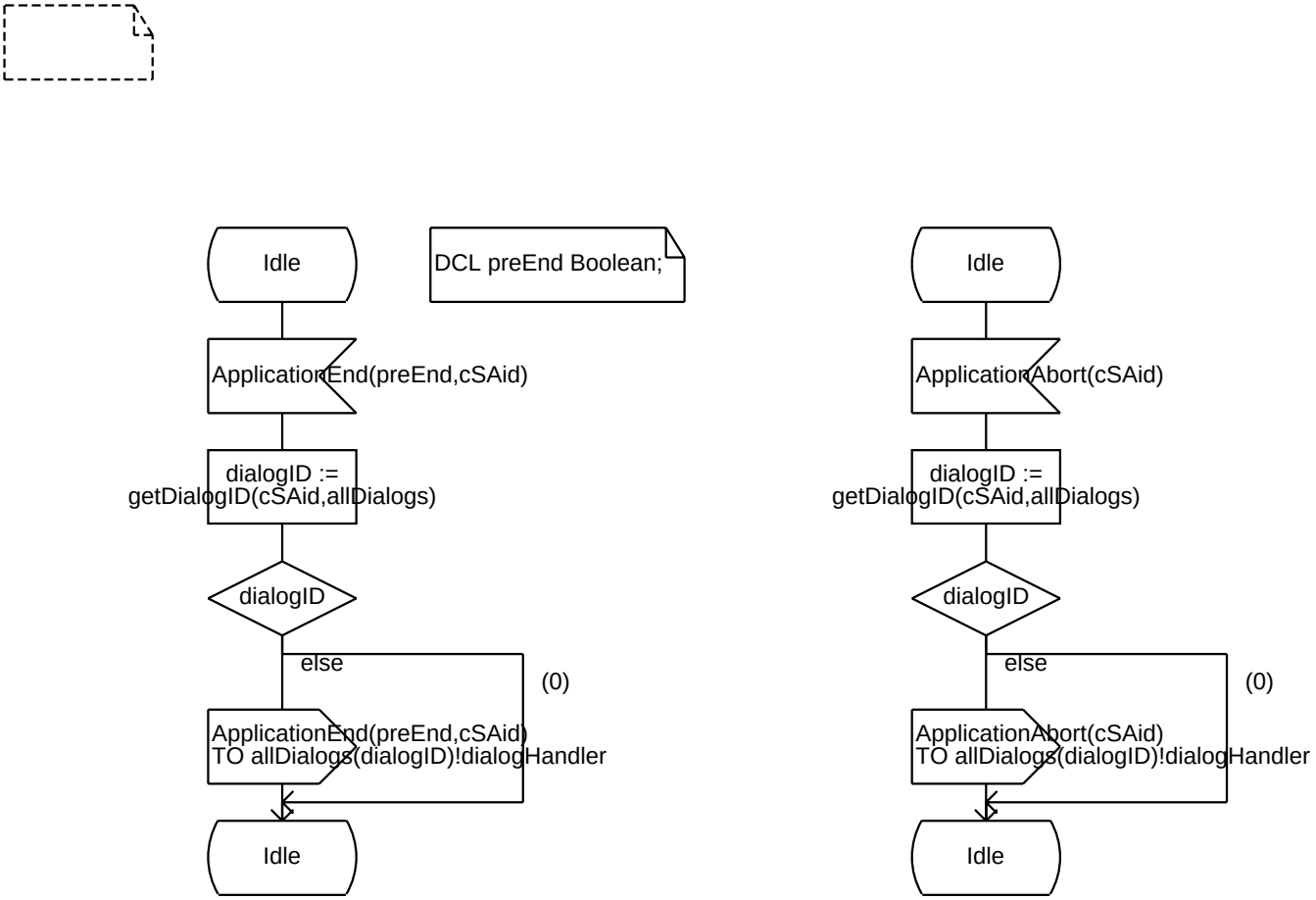
Annex A:

Virtual Process Type TCAP_InterfaceHandler 8(12)

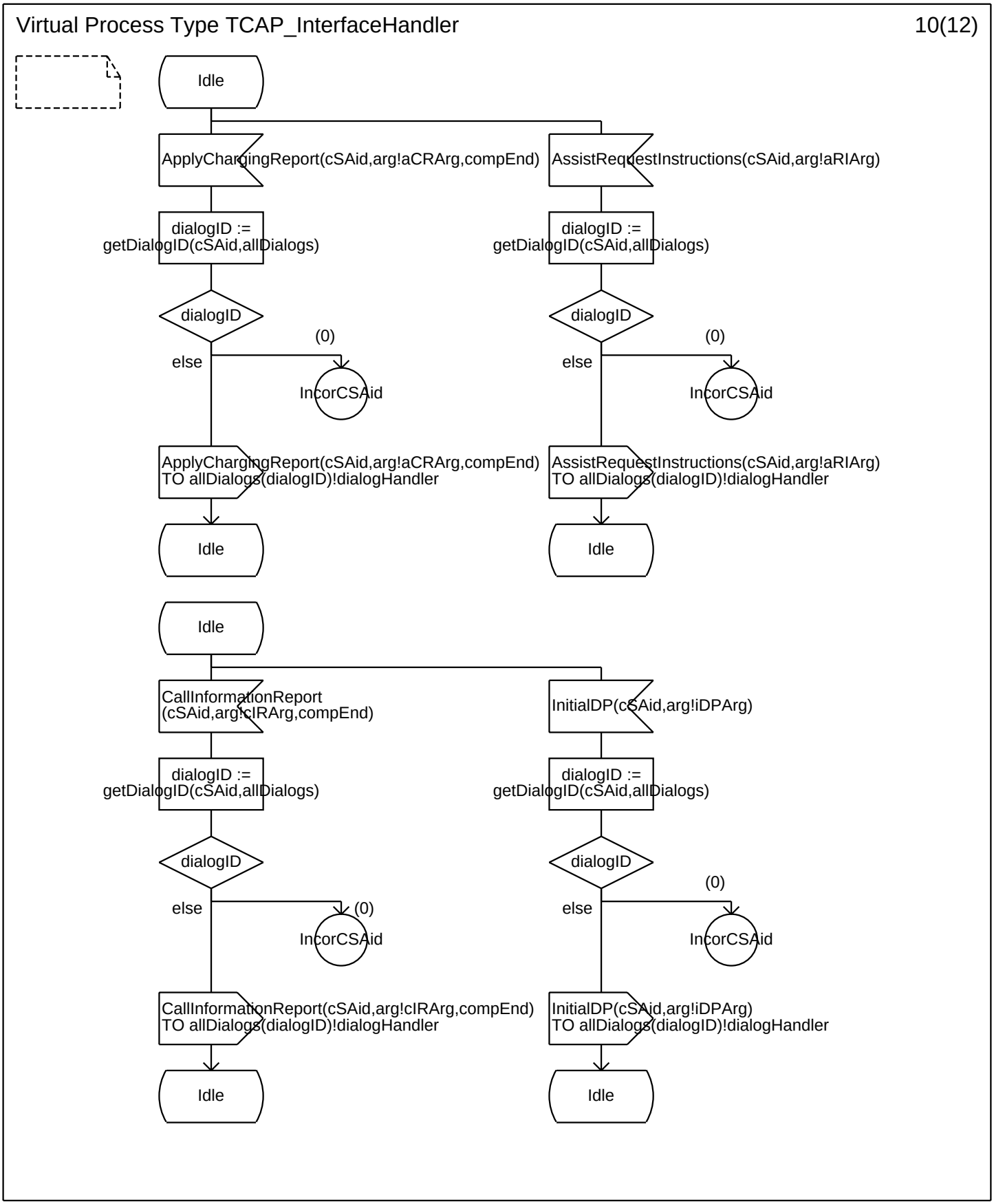


Annex A:

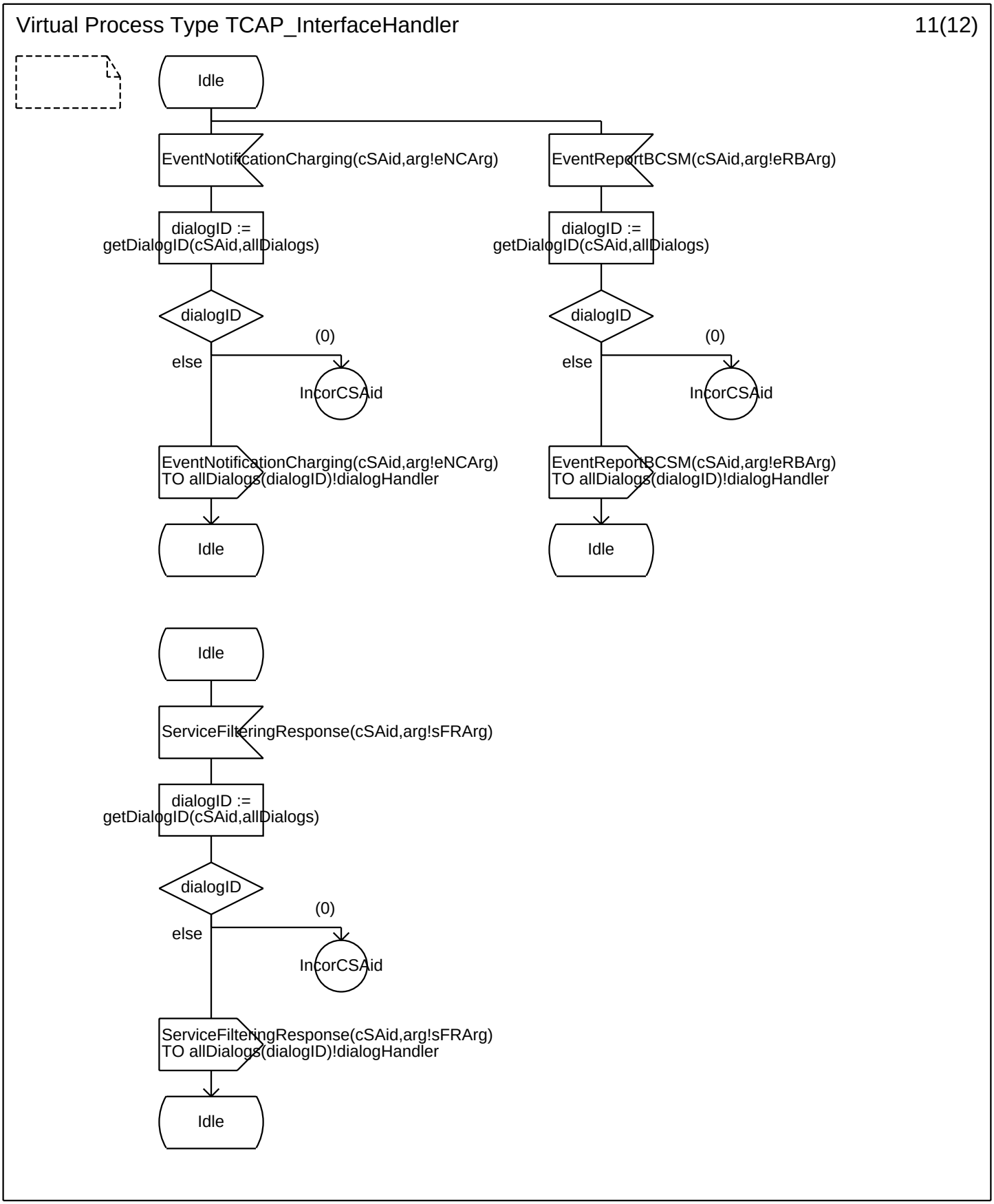
Virtual Process Type TCAP_InterfaceHandler 9(12)



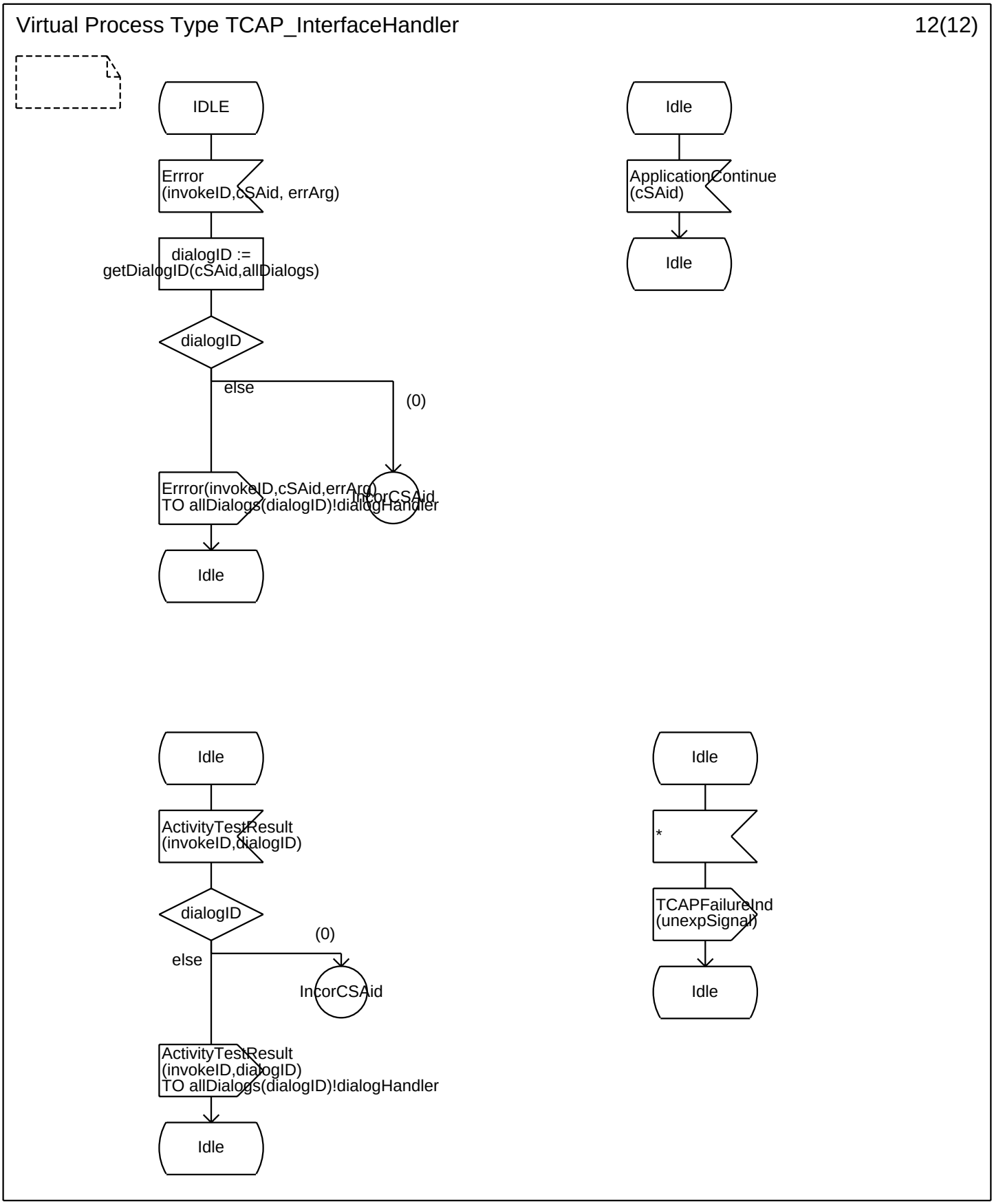
Annex A:


Virtual Process Type TCAP_InterfaceHandler
10(12)
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

Annex A:

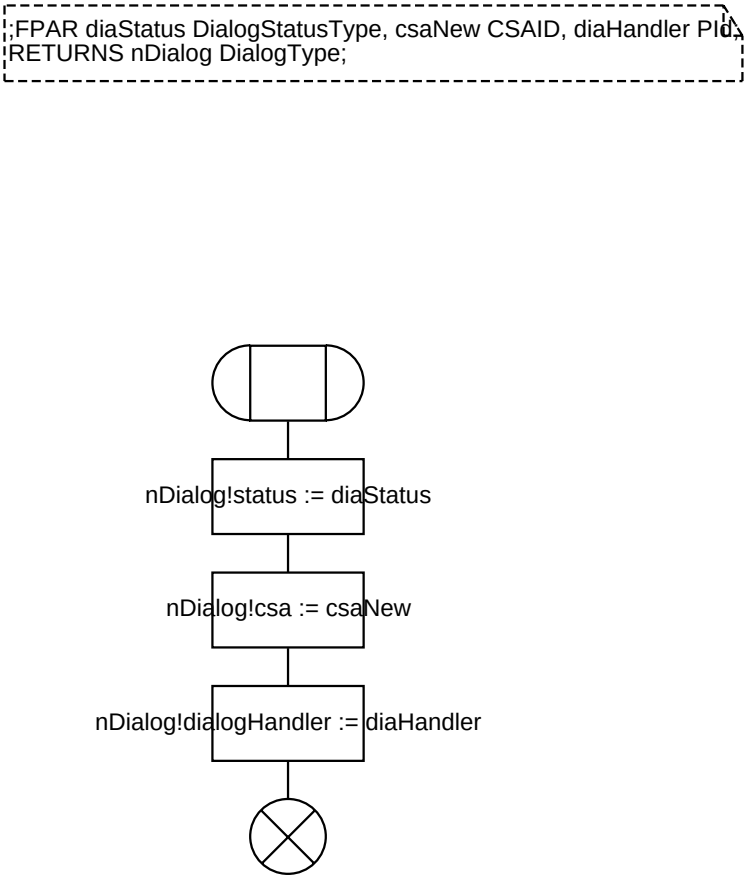

Virtual Process Type TCAP_InterfaceHandler
11(12)
Idle
EventNotificationCharging(cSAid,arg!eNCArg)
EventReportBCSM(cSAid,arg!eRBArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
dialogID
(0)
(0)
else
else
IncorCSAid
IncorCSAid
EventNotificationCharging(cSAid,arg!eNCArg)
TO allDialogs(dialogID)!dialogHandler
EventReportBCSM(cSAid,arg!eRBArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
Idle
ServiceFilteringResponse(cSAid,arg!sFRArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
IncorCSAid
ServiceFilteringResponse(cSAid,arg!sFRArg)
TO allDialogs(dialogID)!dialogHandler
Idle

Annex A:

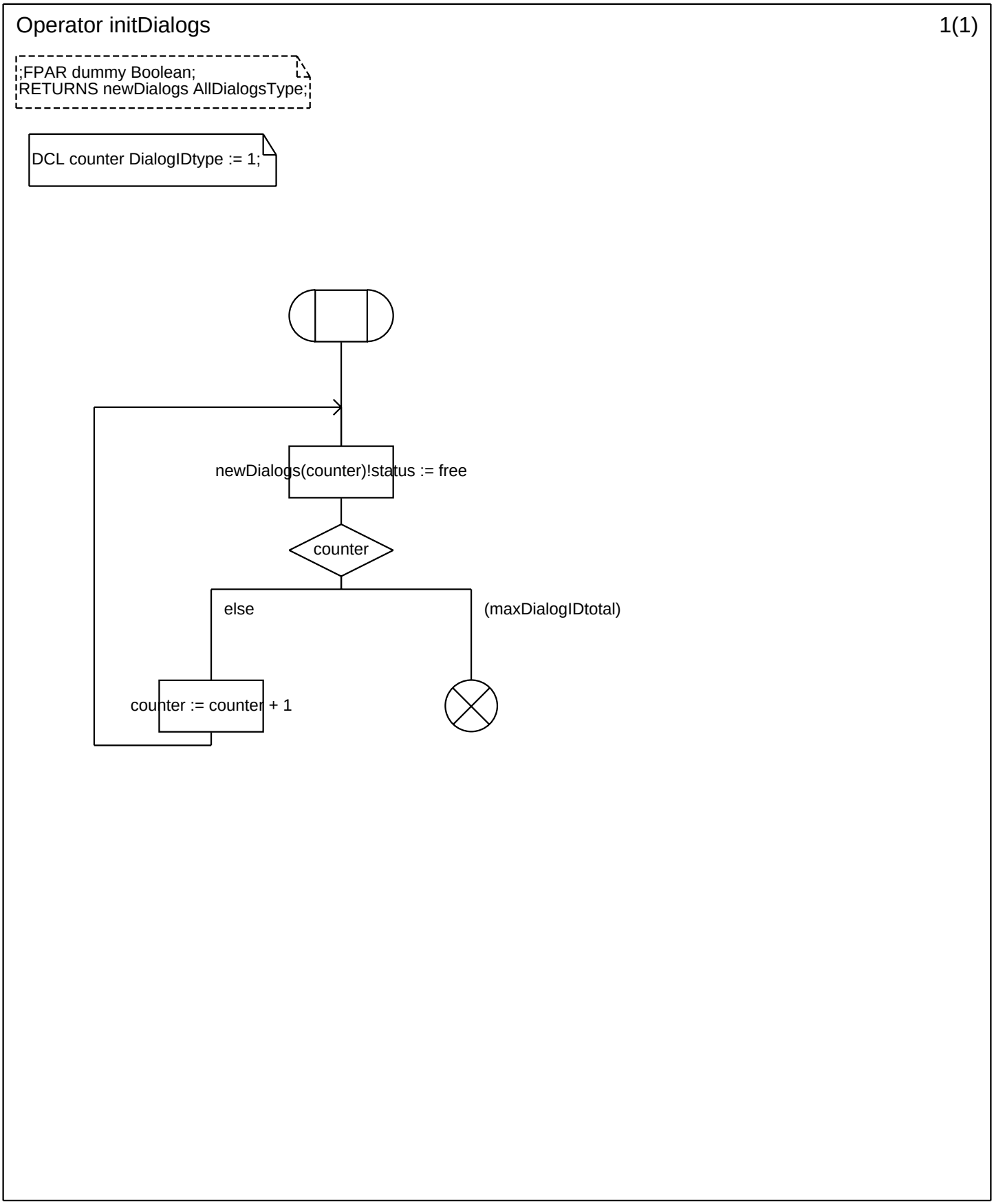

Virtual Process Type TCAP_InterfaceHandler
12(12)
IDLE
Errror
(invokeID,cSAid, errArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
else
(0)
Errror(invokeID,cSAid,errArg)
TO allDialogs(dialogID)!dialogHandler
IncorCSAid
Idle
Idle
ApplicationContinue
(cSAid)
Idle
Idle
ActivityTestResult
(invokeID,dialogID)
dialogID
(0)
else
IncorCSAid
ActivityTestResult
(invokeID,dialogID)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
*
TCAPFailureInd
(unexpSignal)
Idle

Annex A:

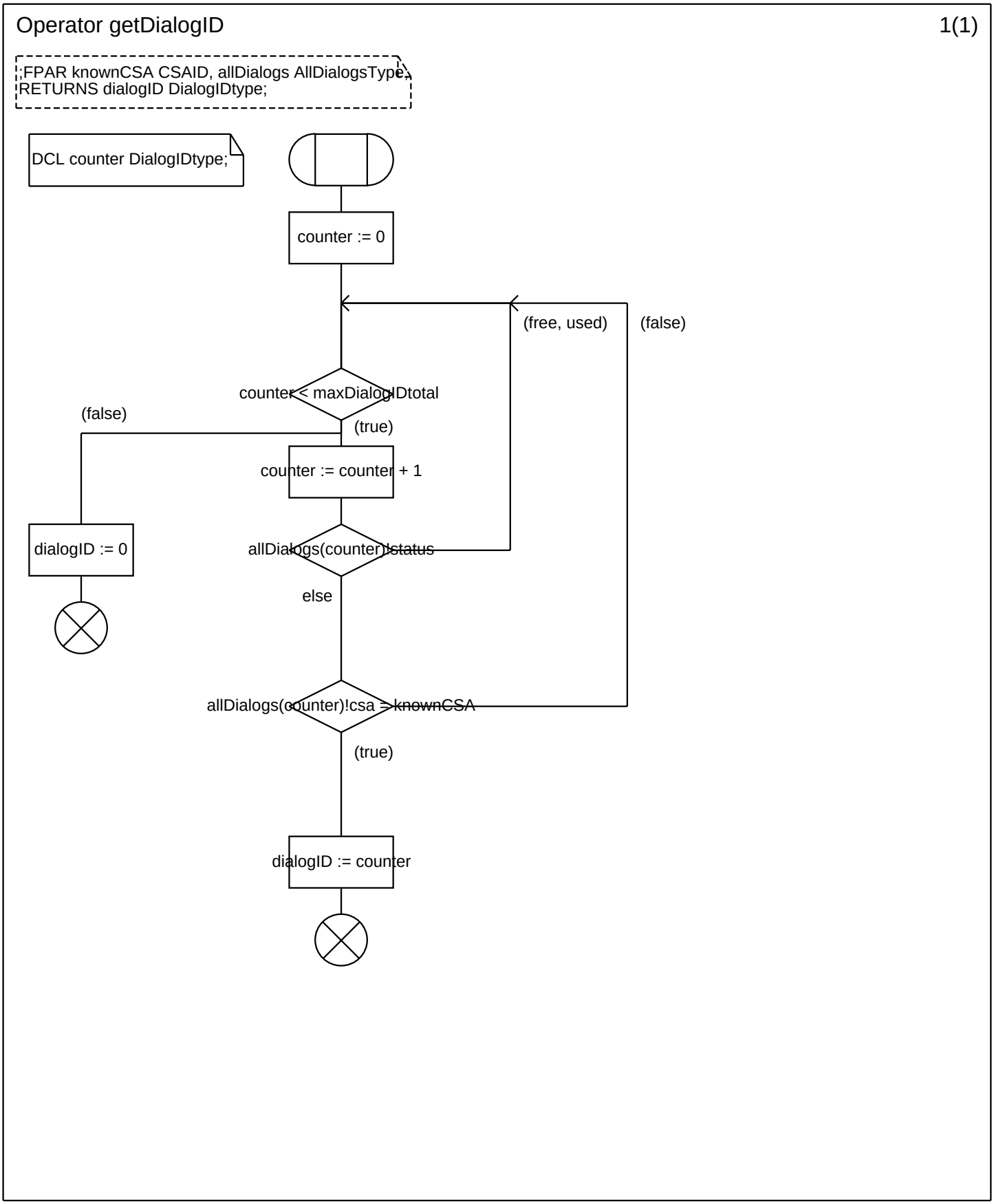
Operator newDialog 1(1)



Annex A:

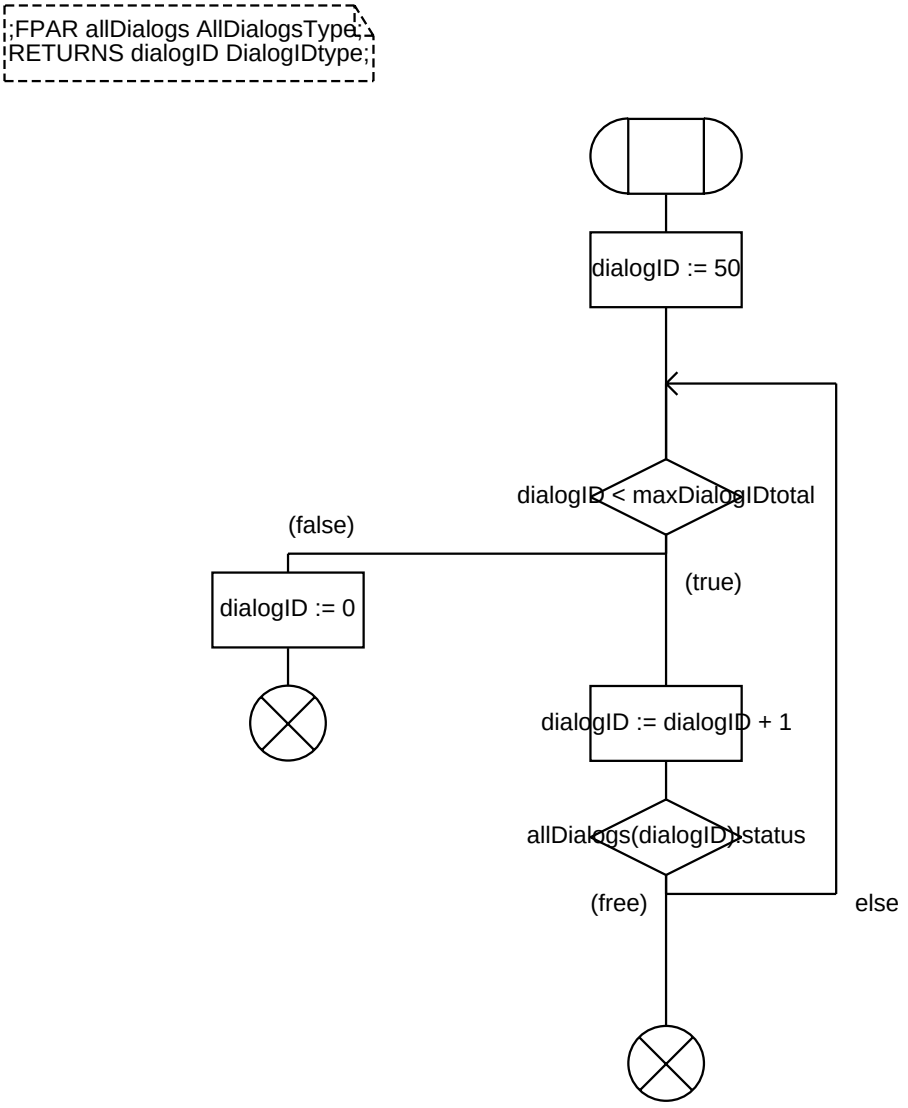

Operator initDialogs
1(1)
;FPAR dummy Boolean;
RETURNS newDialogs AllDialogsType;
DCL counter DialogIDtype := 1;
newDialogs(counter)!status := free
counter
else
(maxDialogIDtotal)
counter := counter + 1

Annex A:

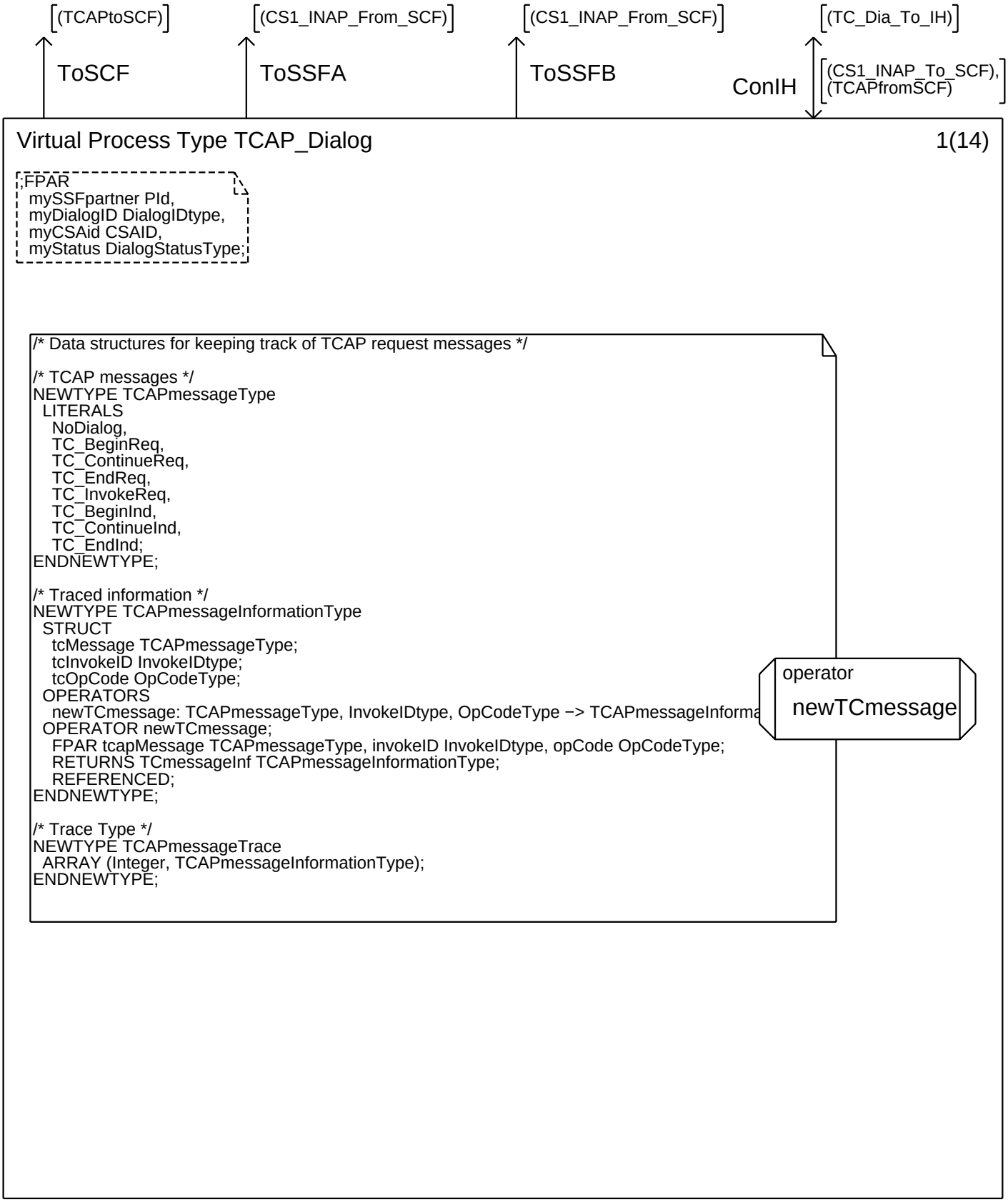

Operator getDialogID
1(1)
;FPAR knownCSA CSAID, allDialogs AllDialogsType;
RETURNS dialogID DialogIDtype;
DCL counter DialogIDtype;
counter := 0
(free, used)
(false)
counter < maxDialogIDtotal
(false)
(true)
counter := counter + 1
dialogID := 0
allDialogs(counter)!status
else
allDialogs(counter)!csa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

Annex A:

Operator nextFreeDialogID 1(1)



Annex A:


(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 −> TCAPmessageInforma
OPERATOR newTCmessage;
FPAR tcapMessage TCAPmessageType, invokeID InvokeIDtype, opCode OpCodeType;
RETURNS TCmessageInf TCAPmessageInformationType;
REFERENCED;
ENDNEWTYPE;
/* Trace Type */
NEWTYPE TCAPmessageTrace
ARRAY (Integer, TCAPmessageInformationType);
ENDNEWTYPE;
operator
newTCmessage

Annex A:

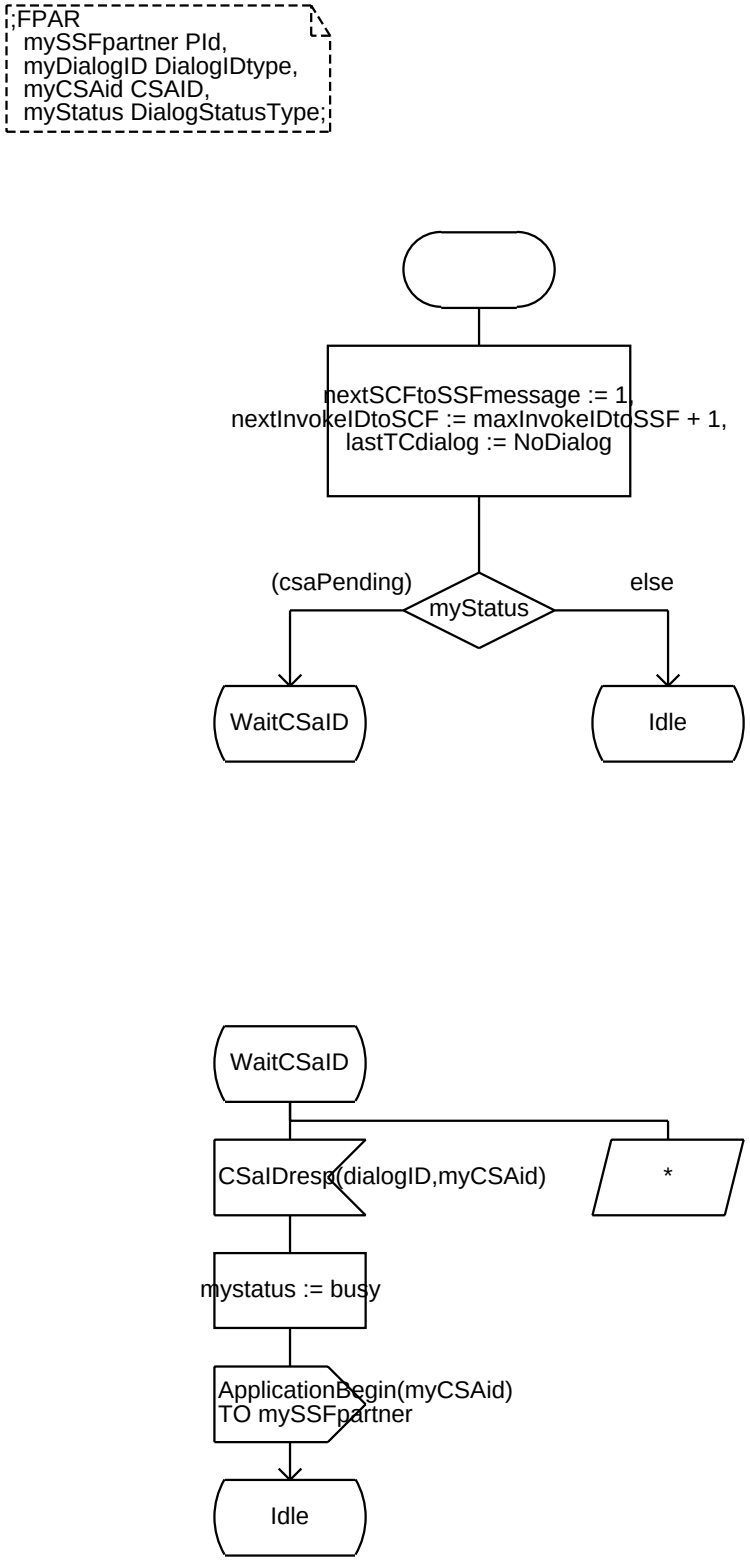
Virtual Process Type TCAP_Dialog 2(14)

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

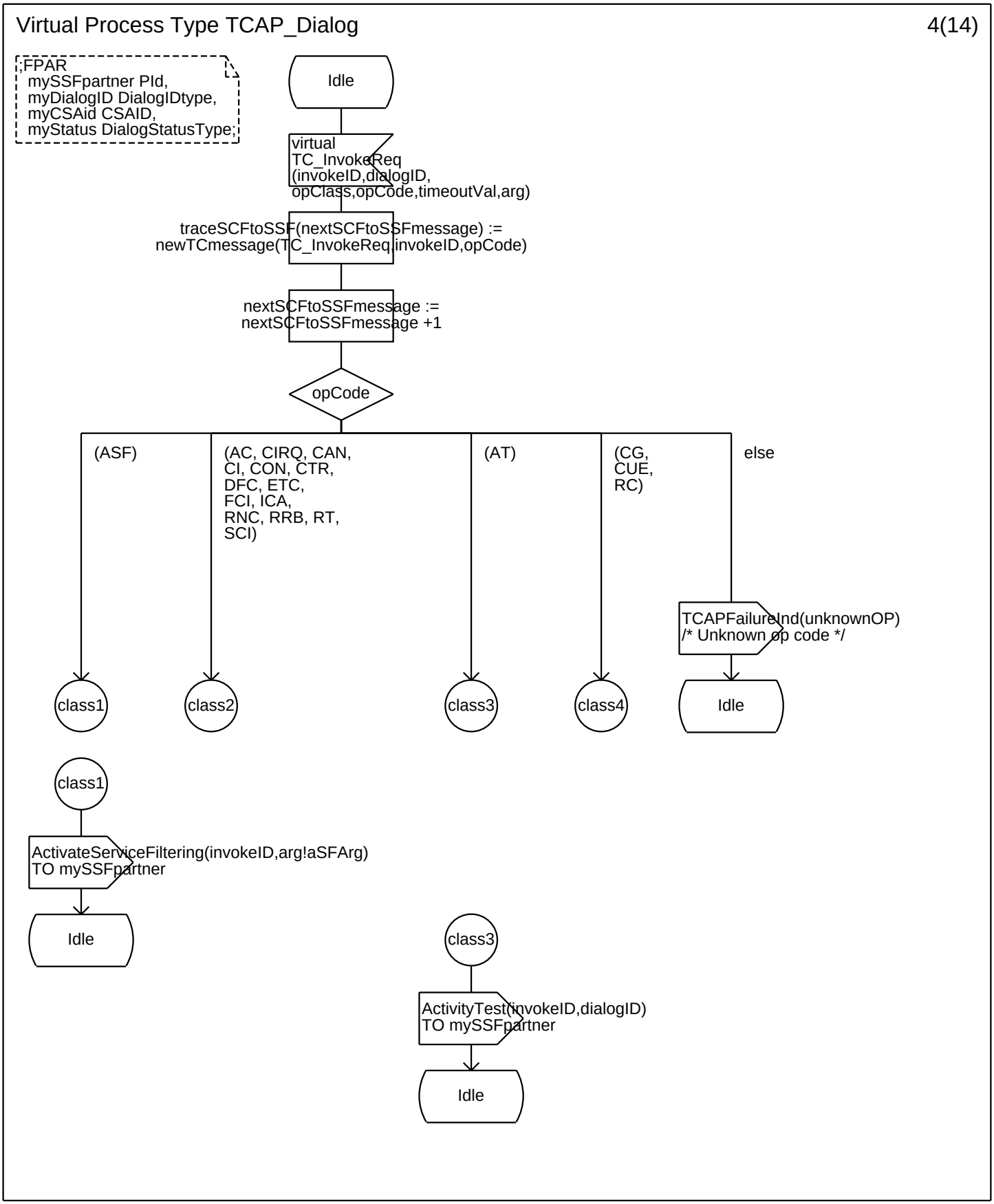
```
/* State variables, to be updated during the entire lifetime of this process */

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;
```

Annex A:

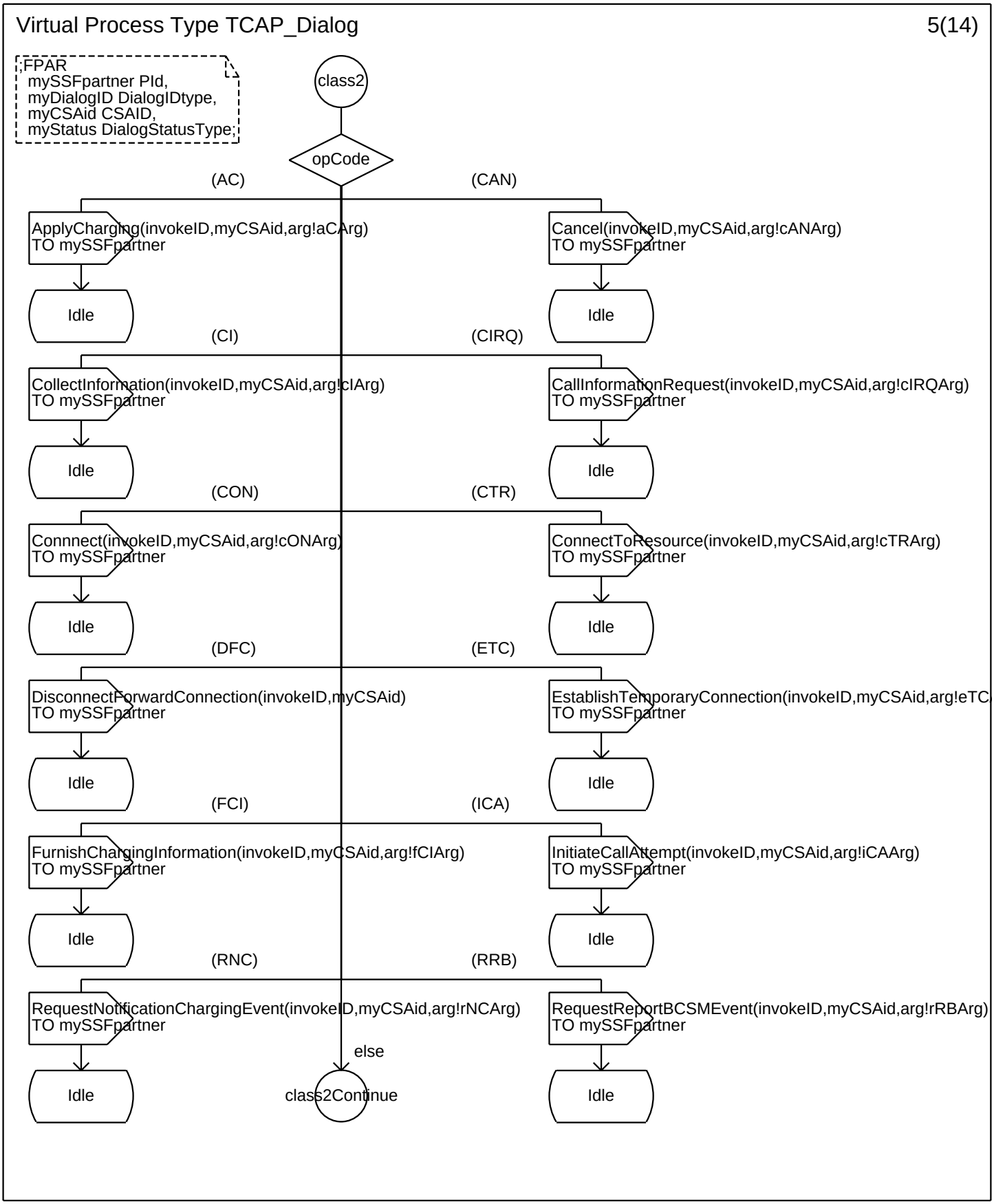
Virtual Process Type TCAP_Dialog 3(14)



Annex A:

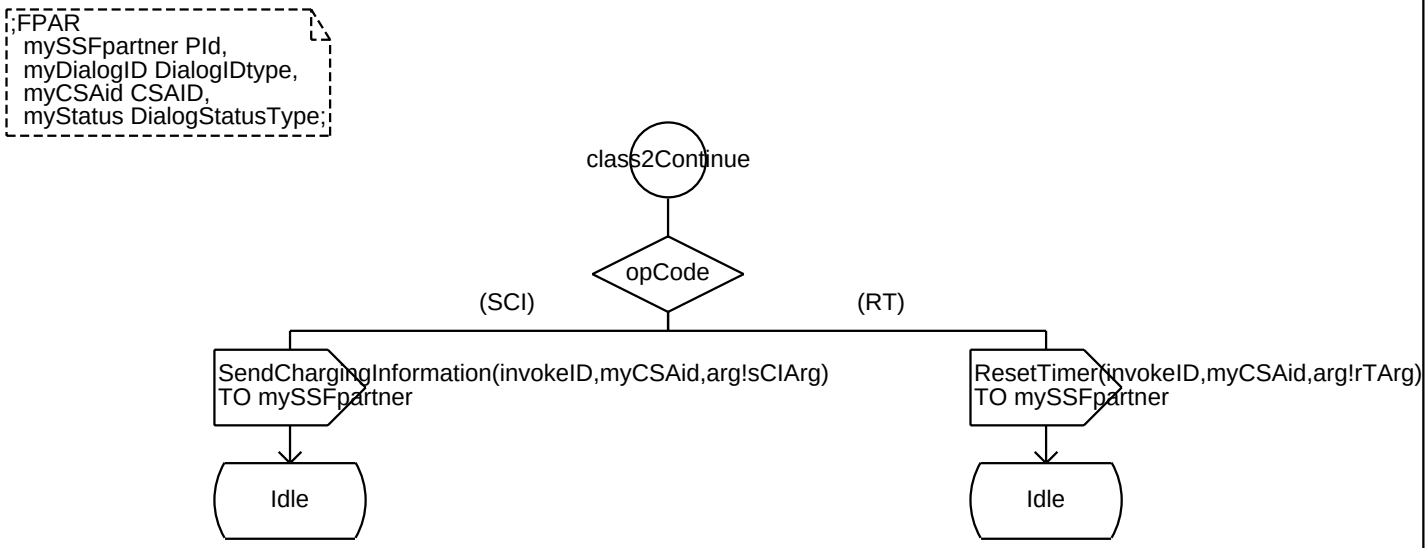

Virtual Process Type TCAP_Dialog
4(14)
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
Idle
virtual
TC_InvokeReq
(invokeID,dialogID,
opClass,opCode,timeoutVal,arg)
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_InvokeReq,invokeID,opCode)
nextSCFtoSSFmessage :=
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

Annex A:

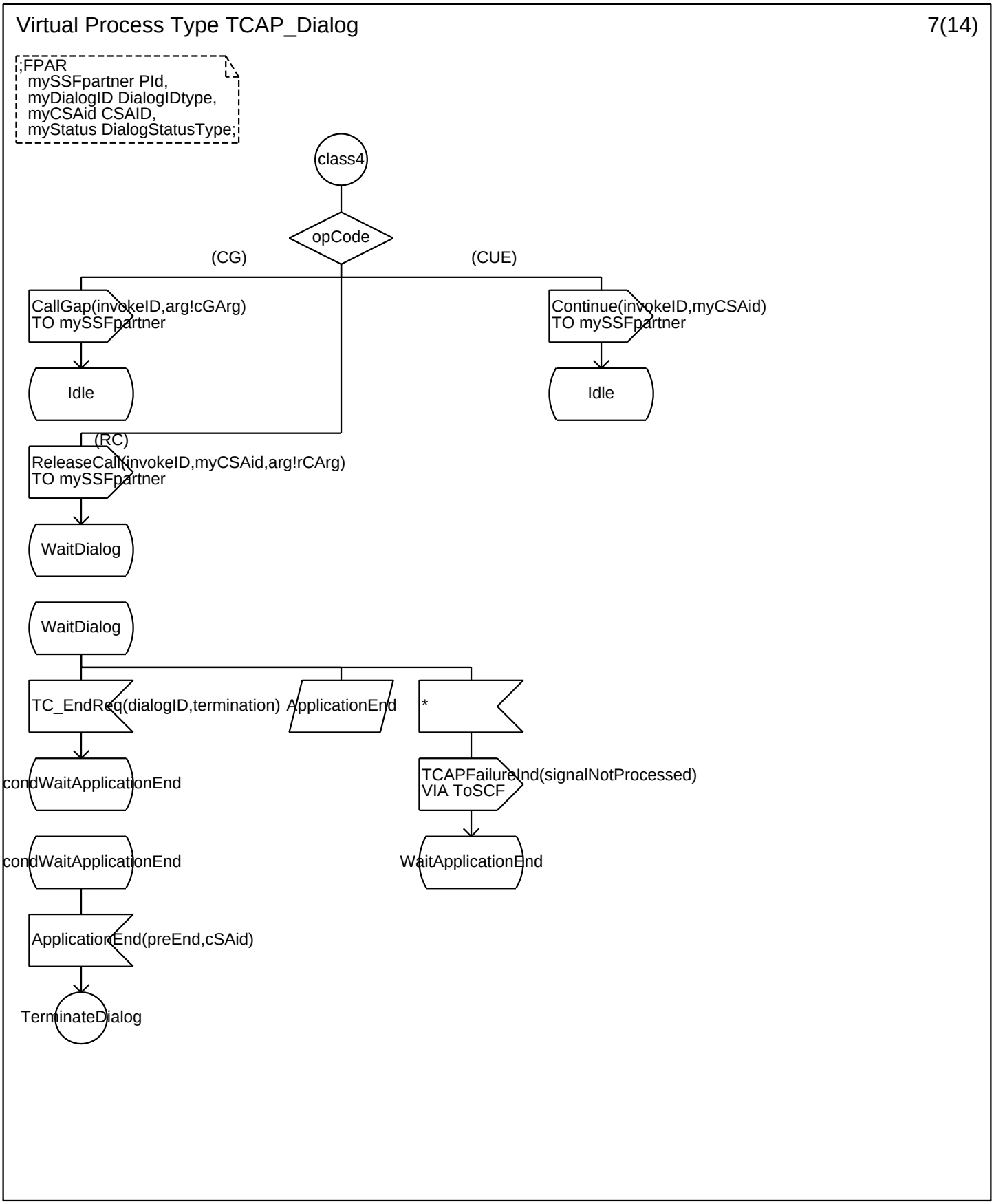

Virtual Process Type TCAP_Dialog
5(14)
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
class2
opCode
(AC)
(CAN)
ApplyCharging(invokeID,myCSAid,arg!aCArg)
TO mySSFpartner
Cancel(invokeID,myCSAid,arg!cANArg)
TO mySSFpartner
Idle
Idle
(CI)
(CIRQ)
CollectInformation(invokeID,myCSAid,arg!cIArg)
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!eTC
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

Annex A:

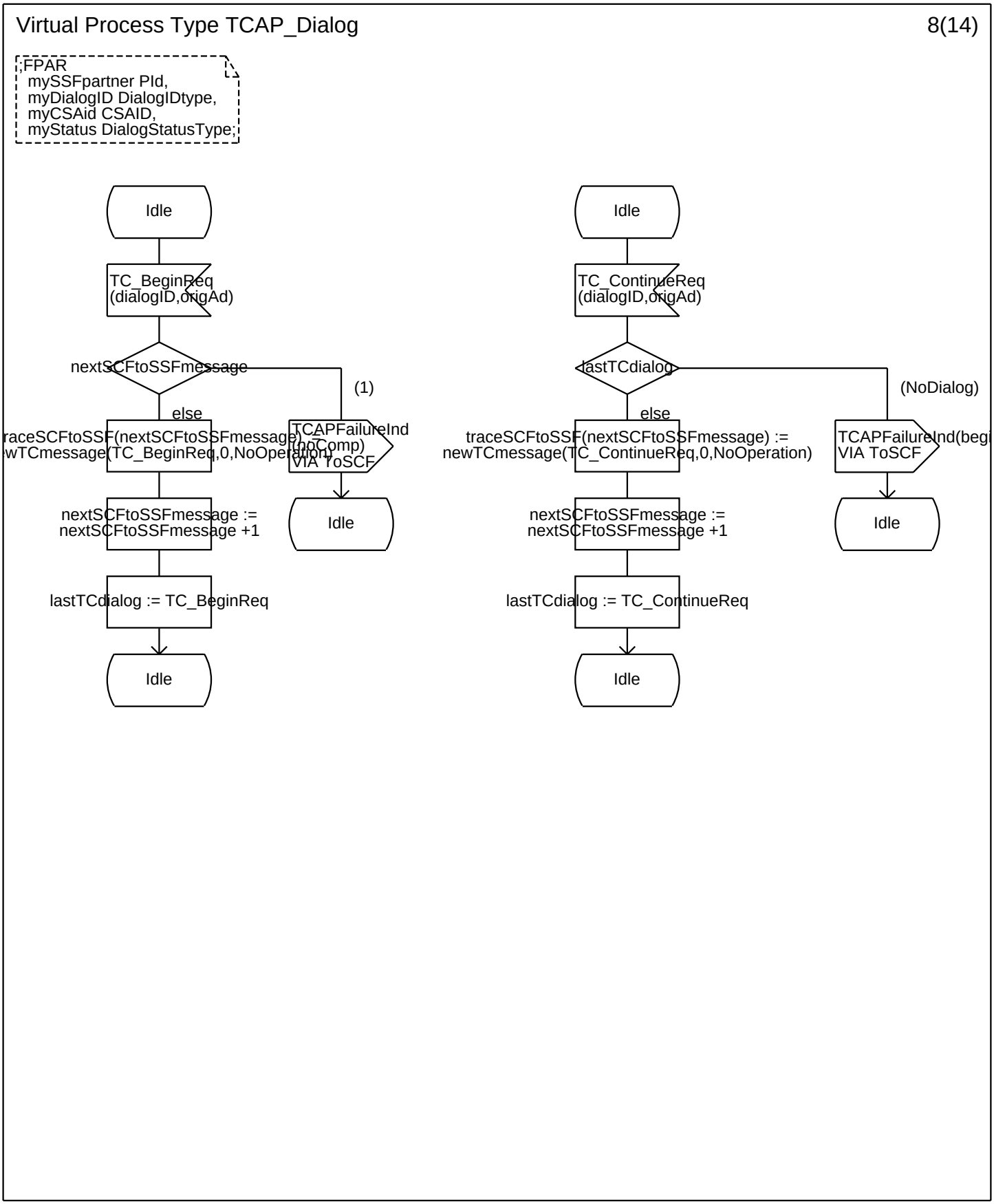
Virtual Process Type TCAP_Dialog 6(14)



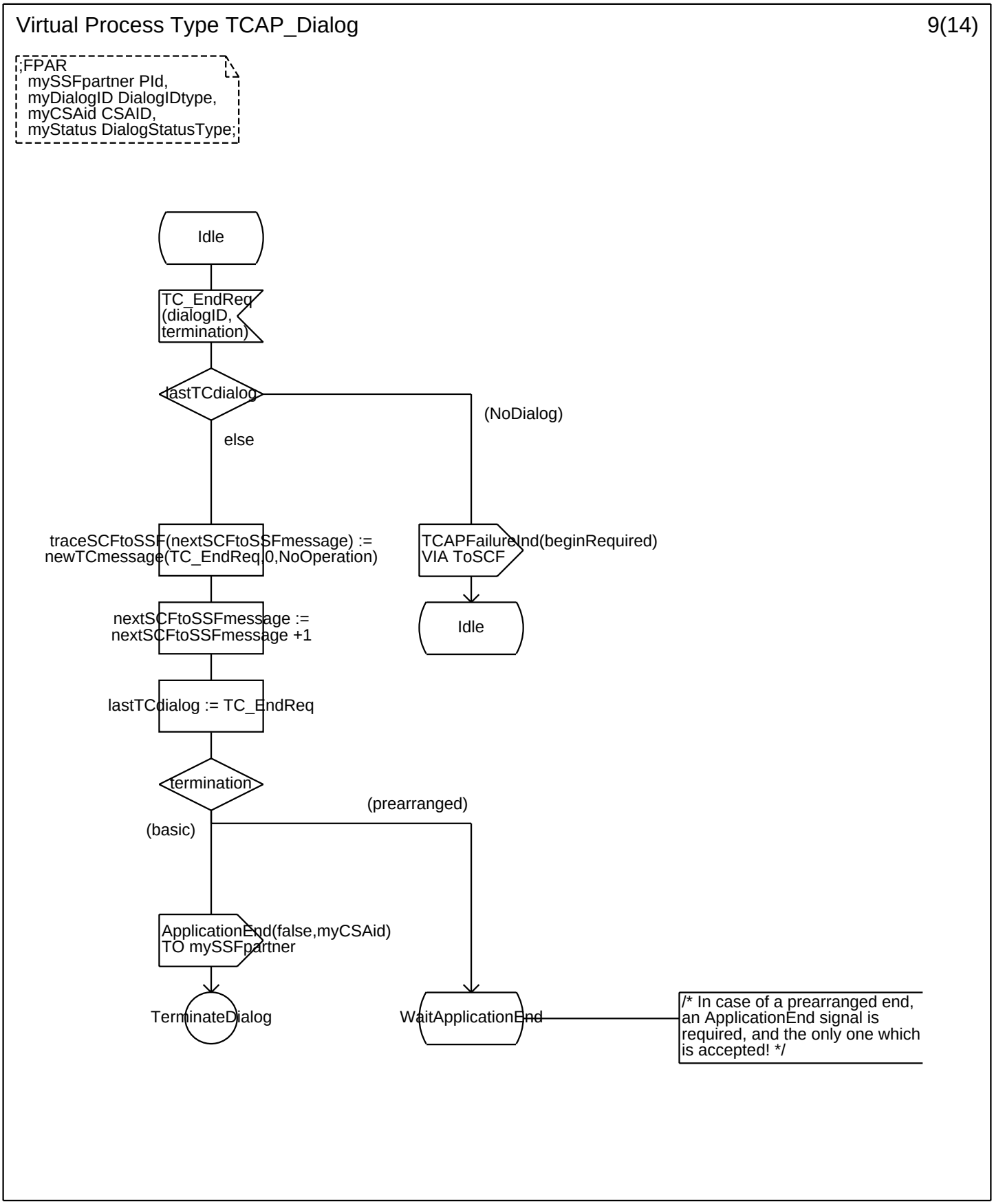
Annex A:


Virtual Process Type TCAP_Dialog
7(14)
;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
*
condWaitApplicationEnd
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
condWaitApplicationEnd
WaitApplicationEnd
ApplicationEnd(preEnd,cSAid)
TerminateDialog

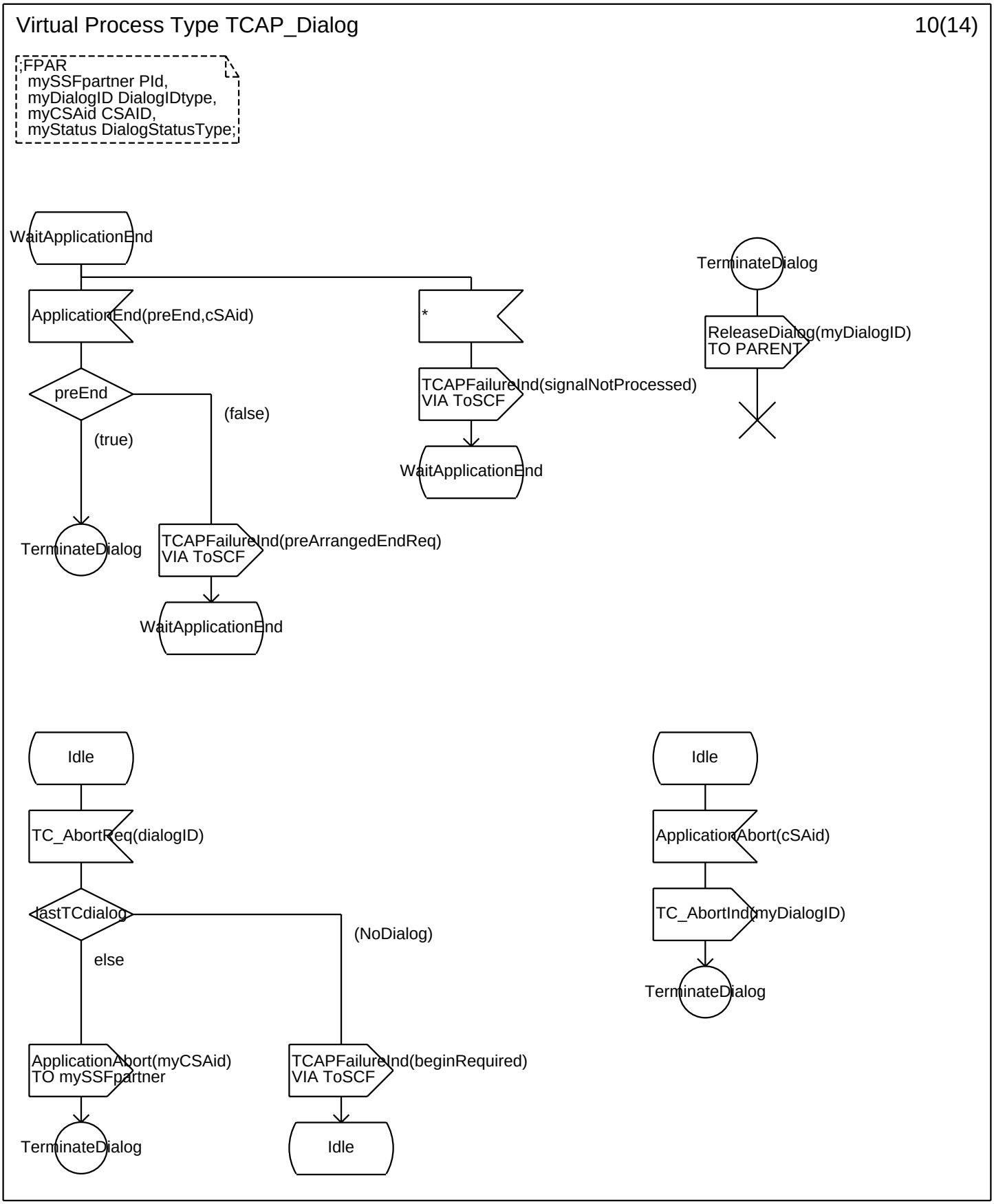
Annex A:


Virtual Process Type TCAP_Dialog
8(14)
;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(begi
VIA ToSCF
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
Idle
lastTCdialog := TC_ContinueReq
Idle

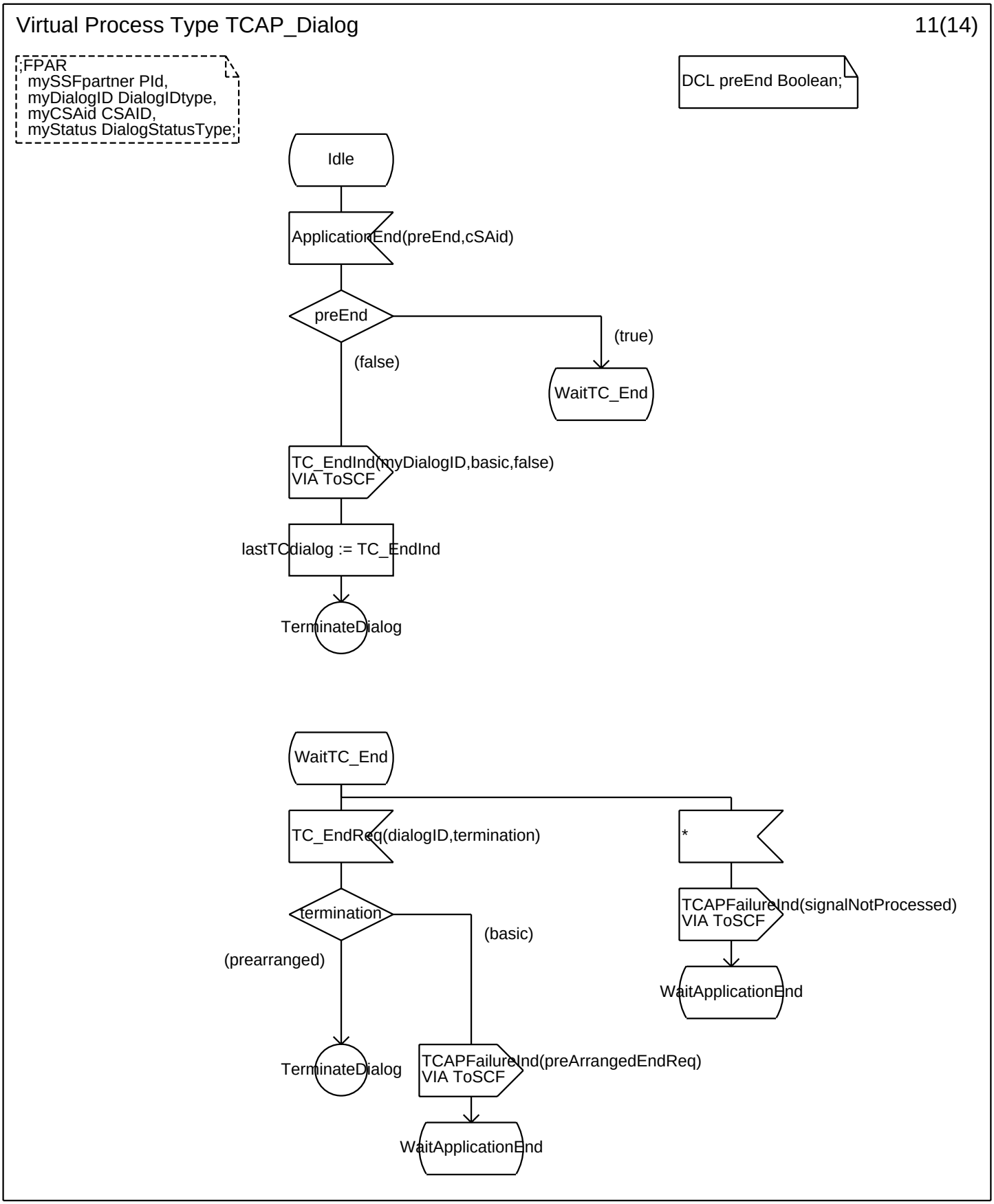
Annex A:


Virtual Process Type TCAP_Dialog
9(14)
;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! */

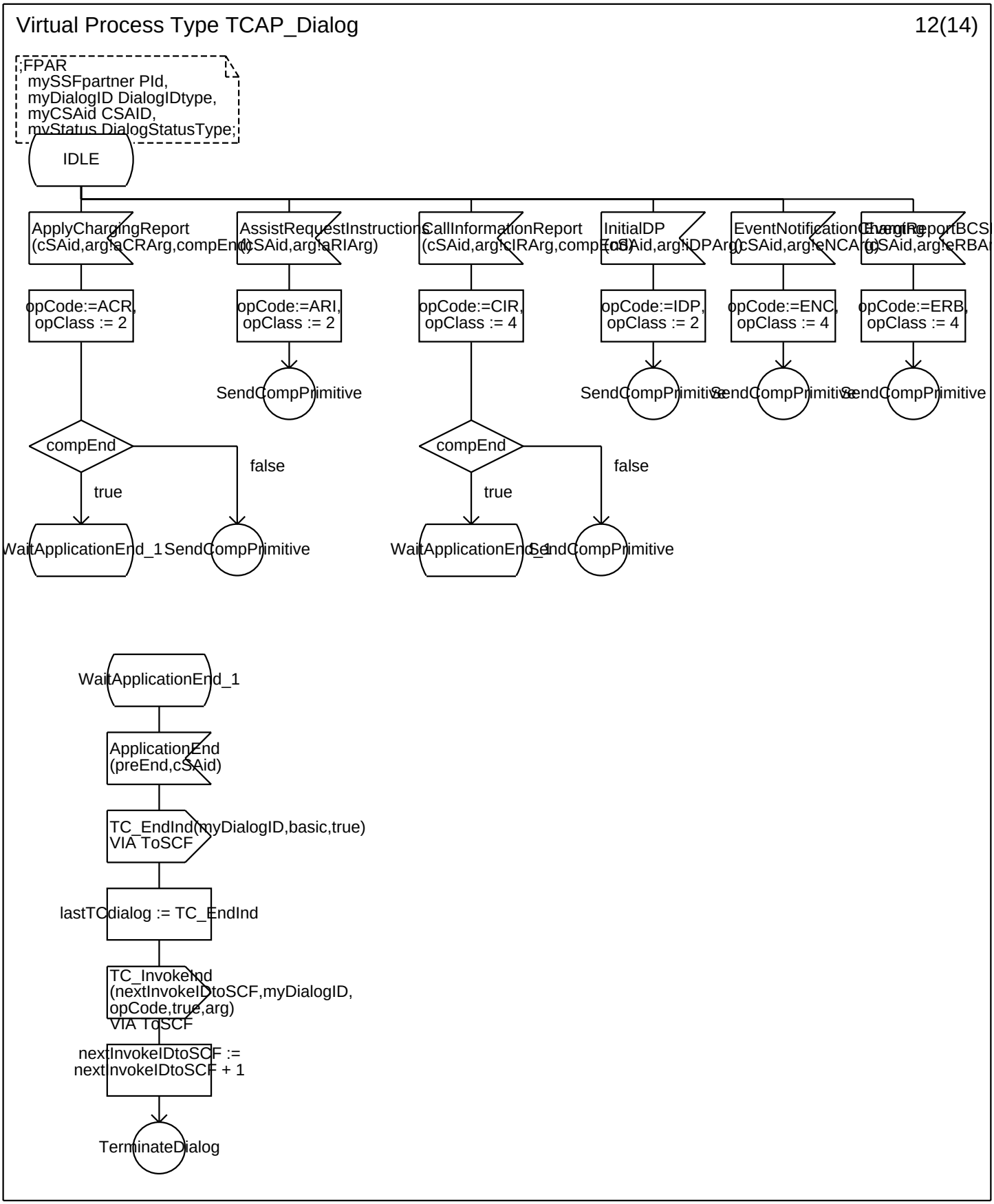
Annex A:


Virtual Process Type TCAP_Dialog
10(14)
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
WaitApplicationEnd
ApplicationEnd(preEnd,cSAid)
*
TerminateDialog
ReleaseDialog(myDialogID)
TO PARENT
preEnd
(false)
(true)
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
WaitApplicationEnd
TerminateDialog
TCAPFailureInd(preArrangedEndReq)
VIA ToSCF
WaitApplicationEnd
Idle
TC_AbortReq(dialogID)
lastTCdialog
(NoDialog)
else
ApplicationAbort(myCSAid)
TO mySSFpartner
TCAPFailureInd(beginRequired)
VIA ToSCF
TerminateDialog
Idle
Idle
ApplicationAbort(cSAid)
TC_AbortInd(myDialogID)
TerminateDialog

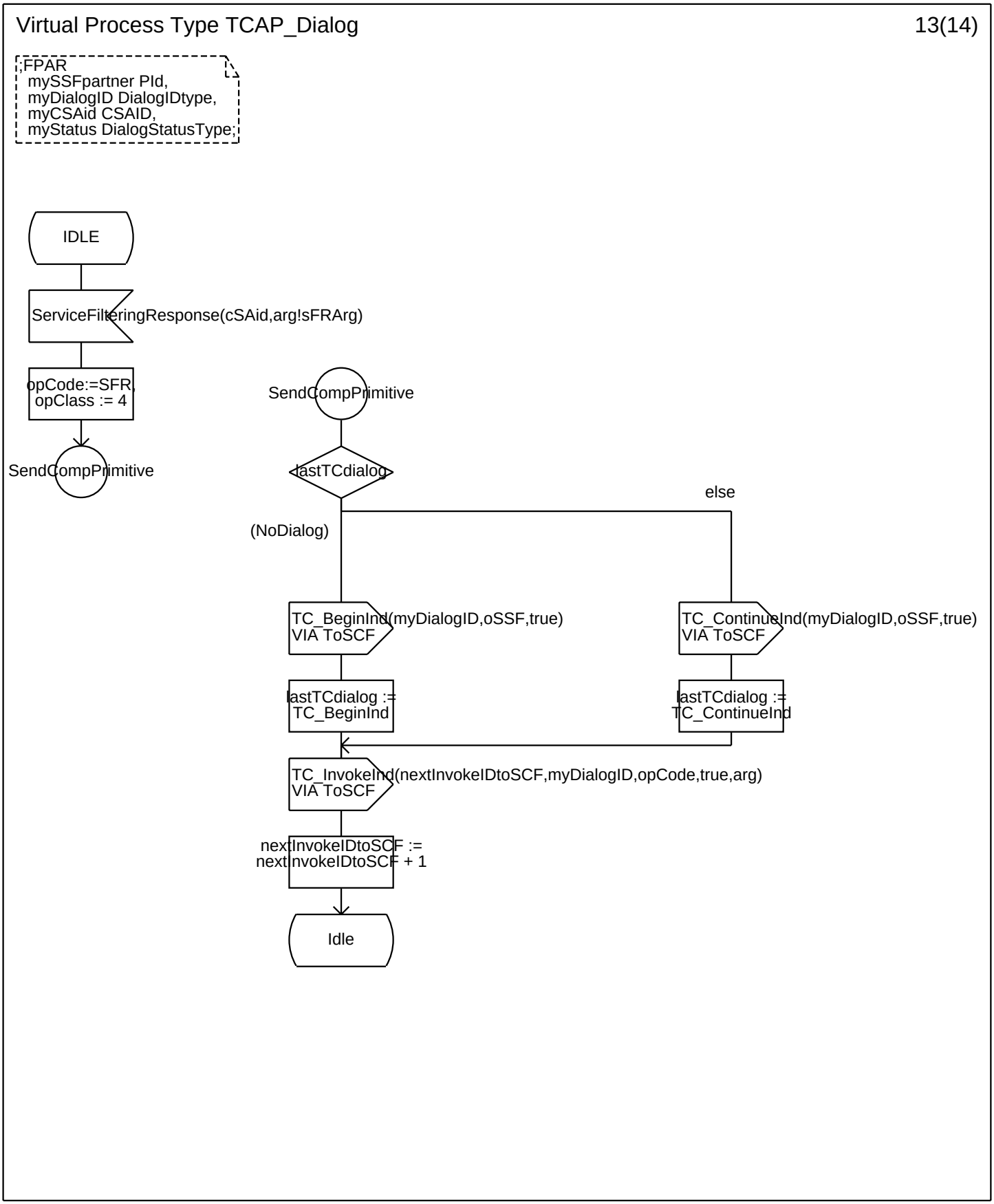
Annex A:


Virtual Process Type TCAP_Dialog
11(14)
;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
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
(basic)
(prearranged)
WaitApplicationEnd
TerminateDialog
TCAPFailureInd(preArrangedEndReq)
VIA ToSCF
WaitApplicationEnd

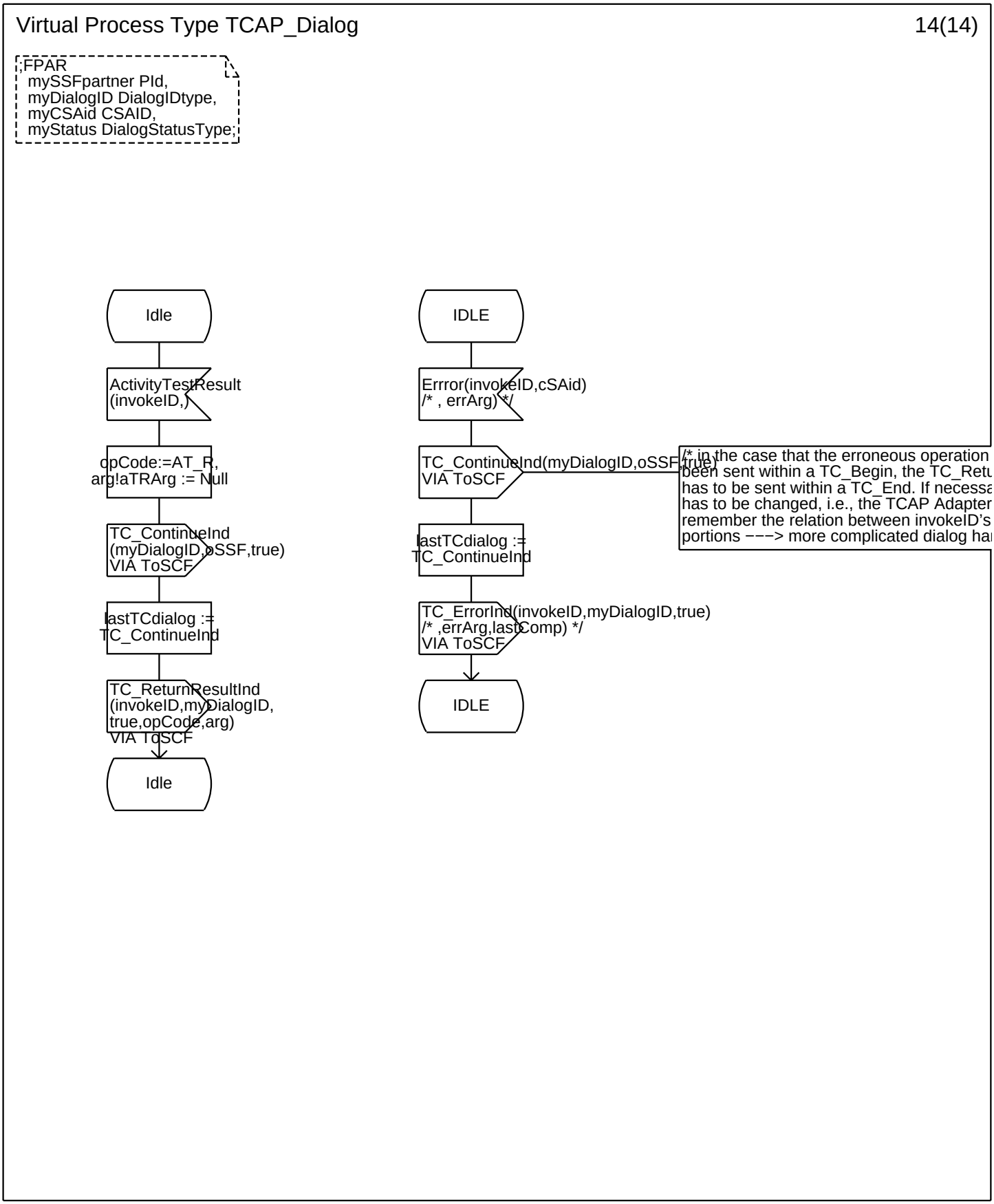
Annex A:


Virtual Process Type TCAP_Dialog
12(14)
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
IDLE
ApplyChargingReport
(cSAid,argaCRArg,compEnd)
AssistRequestInstructions
(cSAid,argaRIArg)
CallInformationReport
(cSAid,argcIRArg,compEnd)
InitialDP
(cSAid,argiDPArg)
EventNotificationChargingReport
(cSAid,argeNCArg)
EventReportBCSM
(cSAid,argeRBArg)
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

Annex A:


Virtual Process Type TCAP_Dialog
13(14)
;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

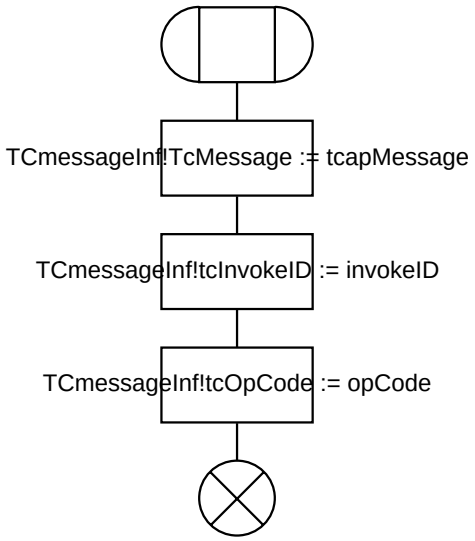
Annex A:


Virtual Process Type TCAP_Dialog
14(14)
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
Idle
ActivityTestResult
(invokeID,)
opCode:=AT_R,
arg!aTRArg := 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
been sent within a TC_Begin, the TC_Retu
has to be sent within a TC_End. If necessa
has to be changed, i.e., the TCAP Adapter
remember the relation between invokeID's
portions −−−> more complicated dialog ha
lastTCdialog :=
TC_ContinueInd
TC_ErrorInd(invokeID,myDialogID,true)
/* ,errArg,lastComp) */
VIA ToSCF
IDLE

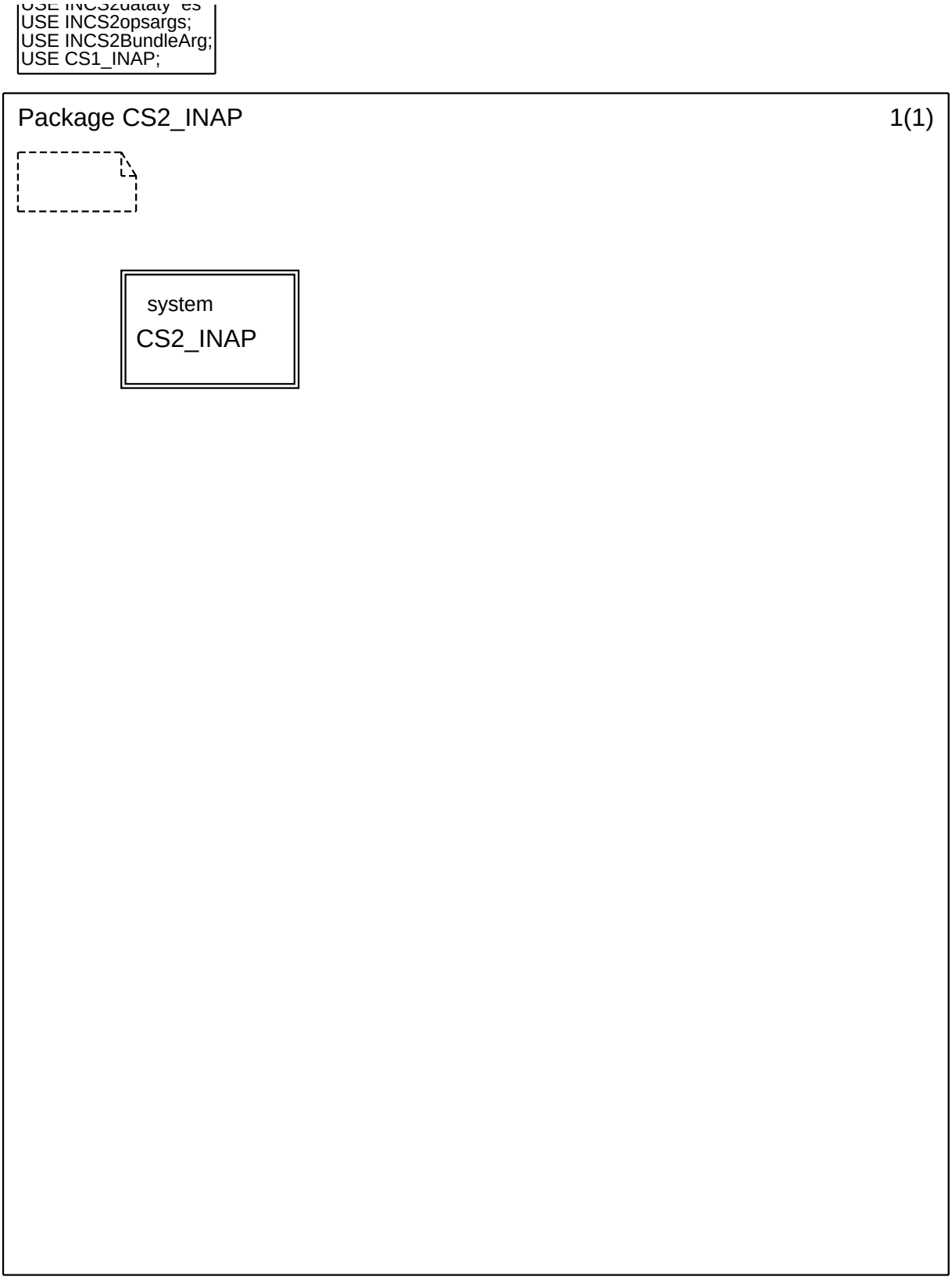
Annex A:

Operator newTCmessage 1(1)

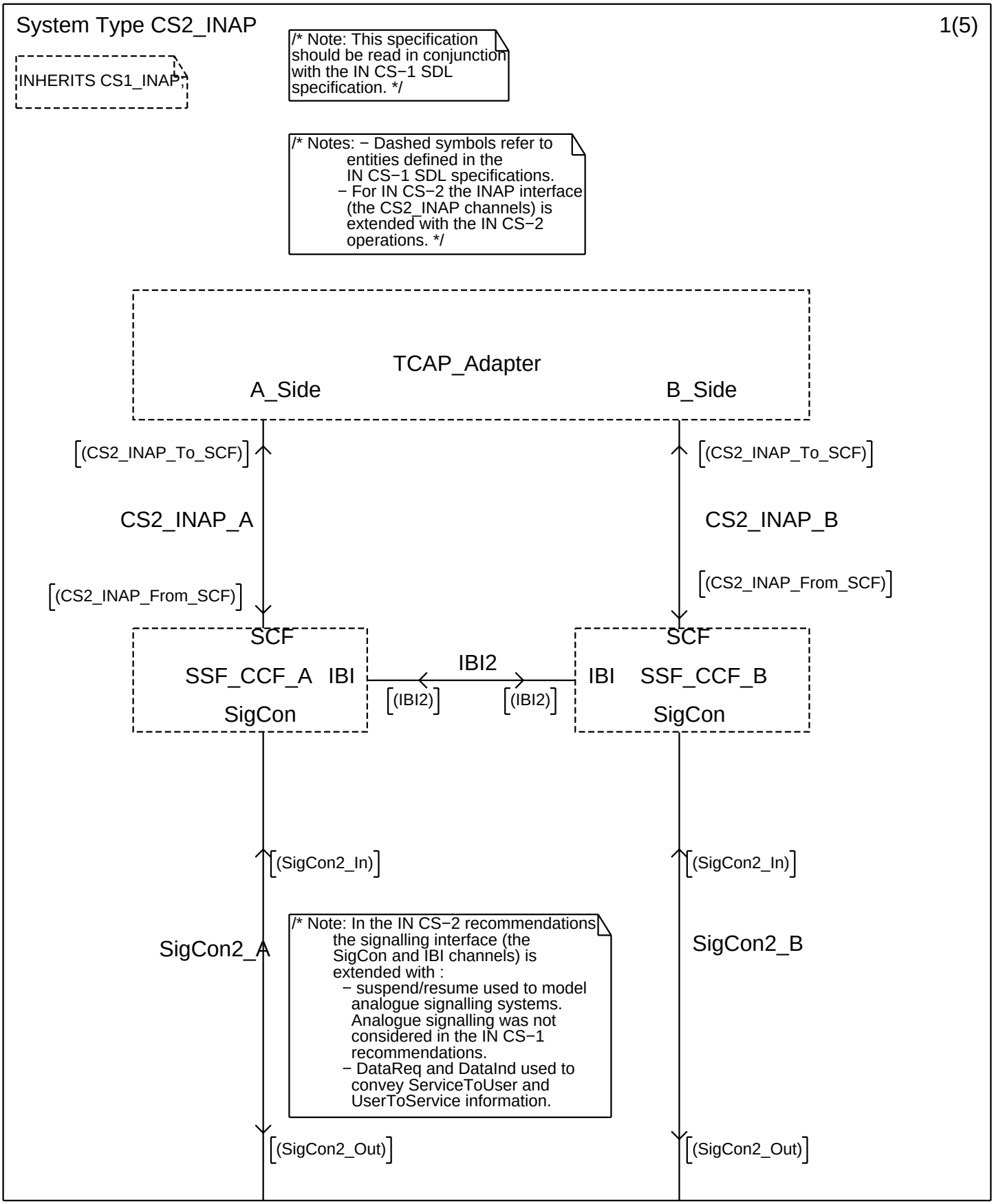
;FPAR tcapMessage TCAPmessageType, invokeID InvokeIDtype, opCode OpCodeType,
RETURNS TCmessageInf TCAPmessageInformationType;



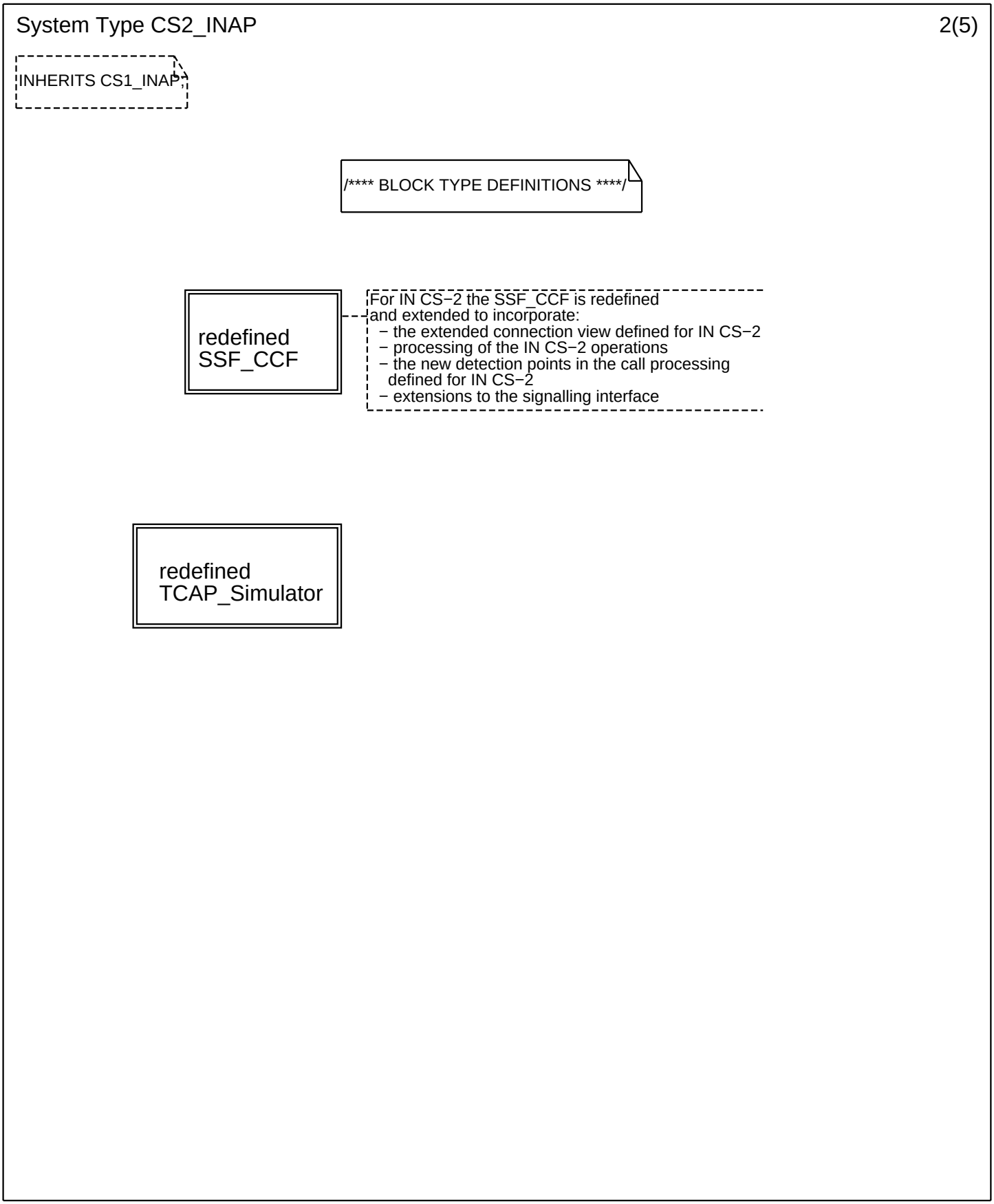
Annex A:


USE INCS2datatypes
USE INCS2opsargs;
USE INCS2BundleArg;
USE CS1_INAP;
Package CS2_INAP
1(1)
system
CS2_INAP

Annex A:


System Type CS2_INAP
1(5)
INHERITS CS1_INAP
/* Note: This specification should be read in conjunction with the IN CS−1 SDL specification. */
/* Notes: − Dashed symbols refer to entities defined in the IN CS−1 SDL specifications.
− For IN CS−2 the INAP interface (the CS2_INAP channels) is extended with the IN CS−2 operations. */
TCAP_Adapter
A_Side
B_Side
(CS2_INAP_To_SCF)
(CS2_INAP_To_SCF)
CS2_INAP_A
CS2_INAP_B
(CS2_INAP_From_SCF)
(CS2_INAP_From_SCF)
SCF
SSF_CCF_A
IBI
SigCon
IBI2
(IBI2)
(IBI2)
SCF
IBI
SSF_CCF_B
SigCon
(SigCon2_In)
(SigCon2_In)
SigCon2_A
SigCon2_B
/* Note: In the IN CS−2 recommendations the signalling interface (the SigCon and IBI channels) is extended with :
− suspend/resume used to model analogue signalling systems. Analogue signalling was not considered in the IN CS−1 recommendations.
− DataReq and DataInd used to convey ServiceToUser and UserToService information.
(SigCon2_Out)
(SigCon2_Out)

Annex A:


System Type CS2_INAP
2(5)
INHERITS CS1_INAP
/**** BLOCK TYPE DEFINITIONS ****/
redefined
SSF_CCF
For IN CS−2 the SSF_CCF is redefined
and extended to incorporate:
− the extended connection view defined for IN CS−2
− processing of the IN CS−2 operations
− the new detection points in the call processing
defined for IN CS−2
− extensions to the signalling interface
redefined
TCAP_Simulator

Annex A:

System Type CS2_INAP 3(5)

```
INHERITS CS1_INAP;

/**** SIGNAL DEFINITIONS FOR THE IN CS−2 SCF−SSF OPERATIONS. ****/

SIGNAL
  ContinueWithArgument(InvokeID,CSAID,continueWithArgumentArg),
  CreateCallSegmentAssociation(InvokeID,CSAID,CreateCallSegmentAssociationArg),
  CreateCallSegmentAssociationResult(InvokeID,CSAID,CreateCallSegmentAssociationResultArg),
  DisconnectLeg(InvokeID,CSAID,DisconnectLegArg),
  DisconnectLegResult(InvokeID,CSAID),
  DisconnectForwardConnectionWithArgument(InvokeID,CSAID,DisconnectForwardConnectionWithArgumentArg),
  EntityReleased(CSAID,EntityReleasedArg),
  ManageTriggerData(InvokeID,DialogIDtype,ManageTriggerDataArg),
  ManageTriggerDataResult(InvokeID,DialogIDtype,ManageTriggerDataResultArg),
  MergeCallSegments(InvokeID,CSAID,MergeCallSegmentsArg),
  MergeCallSegmentsResult(InvokeID,CSAID),
  MoveCallSegments(InvokeID,CSAID,MoveCallSegmentsArg),
  MoveCallSegmentsResult(InvokeID,CSAID),
  MoveLeg(InvokeID,CSAID,MoveLegArg),
  MoveLegResult(InvokeID,CSAID),
  ReportUTSI(CSAID,ReportUTSIArg),
  RequestReportUTSI(InvokeID,CSAID,RequestReportUTSIArg),
  SendSTUI(InvokeID,CSAID,SendSTUIArg),
  SplitLeg(InvokeID,CSAID,SplitLegArg),
  SplitLegResult(InvokeID,CSAID),

  PromptAndReceiveMessage(InvokeID, PromptAndReceiveMessageArg),
  PromptAndReceiveMessageResult(InvokeID, CSAID, MessageReceivedArg),
  ScriptClose(InvokeID, ScriptCloseArg),
  ScriptEvent(InvokeID, ScriptEventArg),
  ScriptInformation(InvokeID, ScriptInformationArg),
  ScriptRun(InvokeID, ScriptRunArg);
```

Annex A:

System Type CS2_INAP 4(5)

INHERITS CS1_INAP;

/**** SIGNAL/PRIMITIVE LIST DEFINITIONS FOR THE IN CS−2 SCF−SSF INTERFACE ****/

```
/* SSF −> SCF */
SIGNALLIST CS2_INAP_To_SCF =
  CreateCallSegmentAssociationResult,
  DisconnectLegResult,
  EntityReleased,
  ManageTriggerDataResult,
  MergeCallSegmentsResult,
  MoveCallSegmentsResult,
  MoveLegResult,
  ReportUTSI,
  SplitLegResult,

  PromptAndReceiveMessageResult,
  ScriptEvent;

/* SCF −> SSF */
SIGNALLIST CS2_INAP_From_SCF =
  ContinueWithArgument,
  CreateCallSegmentAssociation,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  ManageTriggerData,
  MergeCallSegments,
  MoveCallSegments,
  MoveLeg,
  RequestReportUTSI,
  SendSTUI,
  SplitLeg,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun;
```

Annex A:

System Type CS2_INAP 5(5)

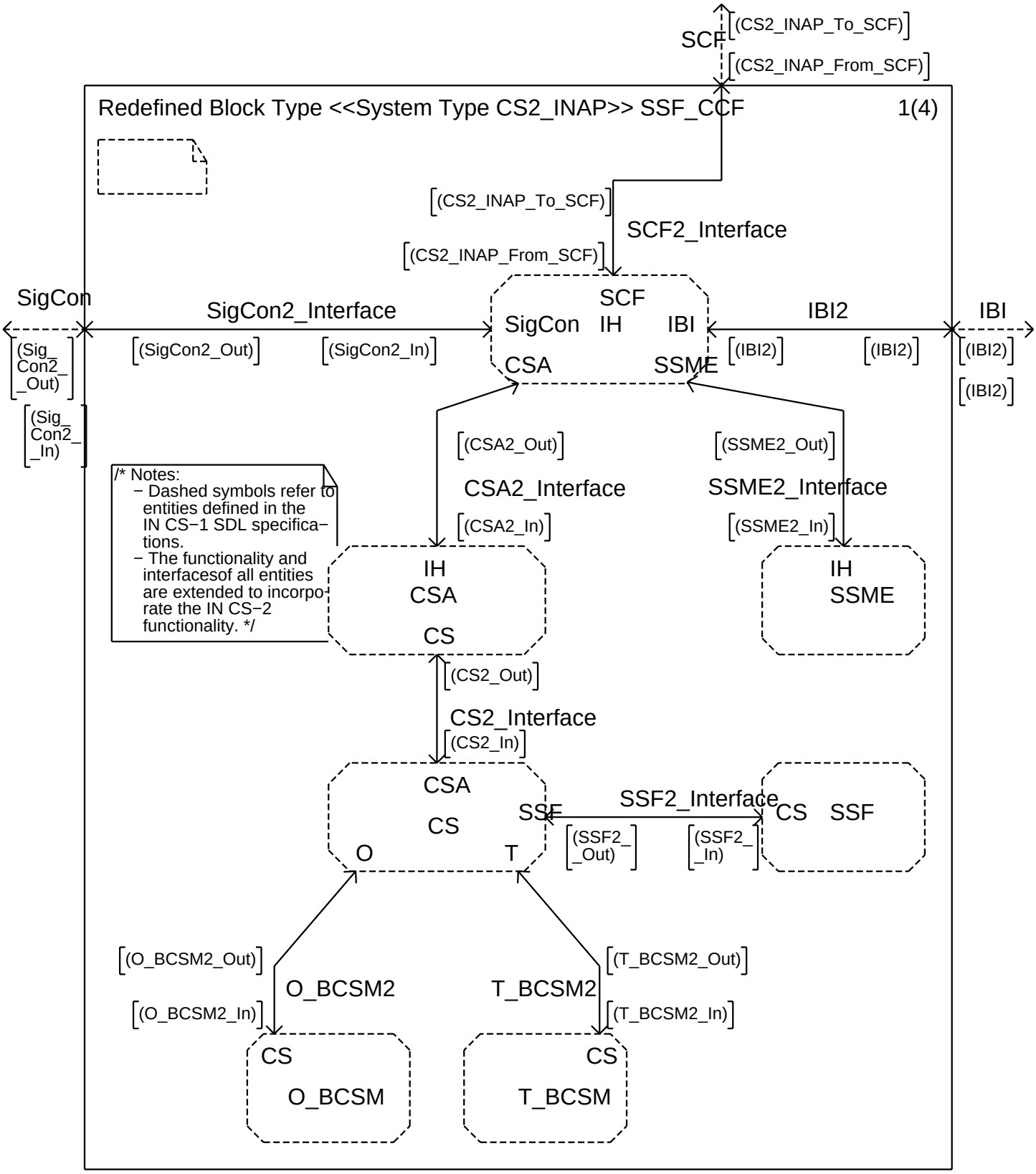
INHERITS CS1_INAP;

```
/**** DEFINITION OF THE 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 */
```

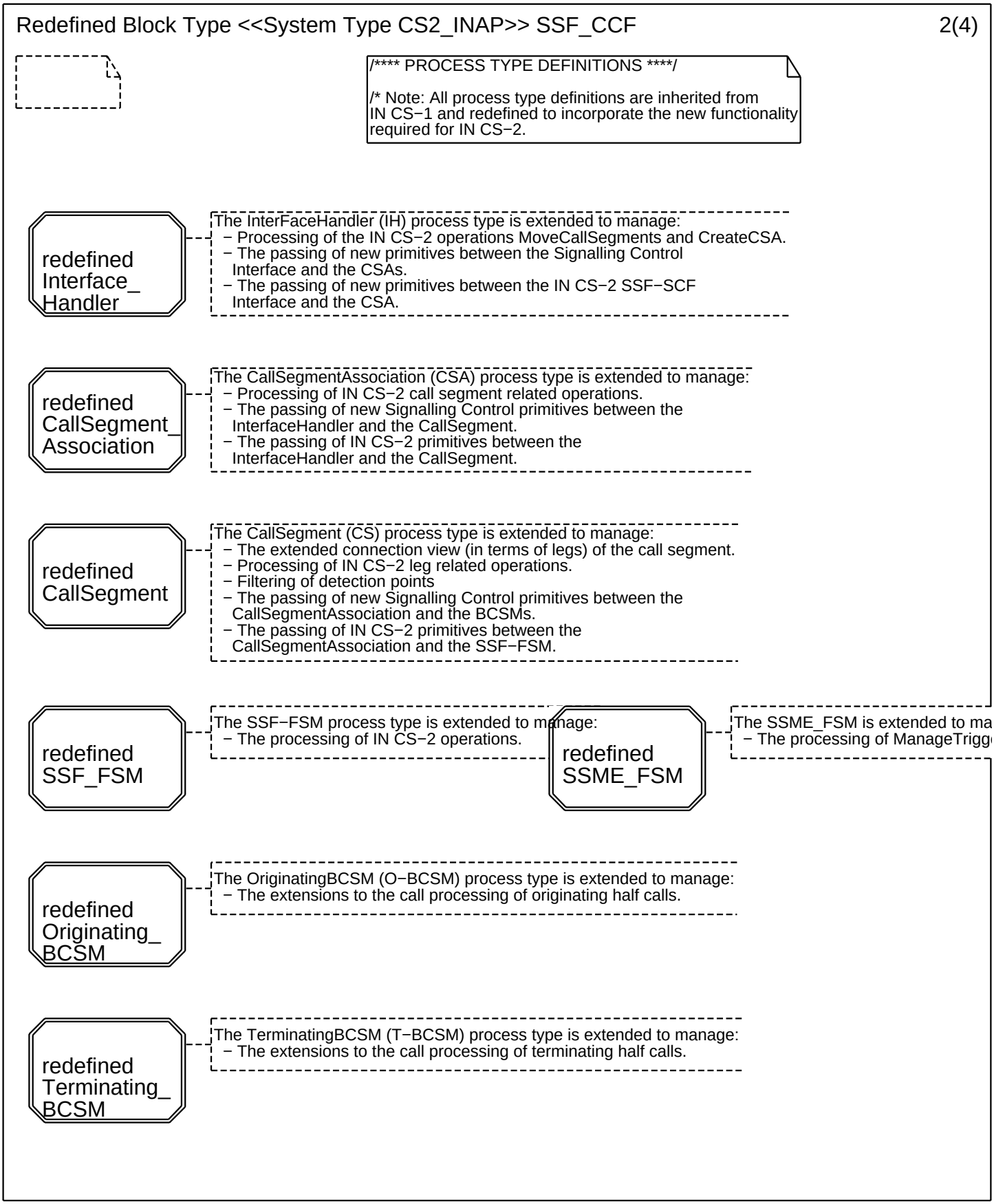
```
/* Signal and signal list definitions for the CS2 extension to
the SigCon interfaces */

SIGNAL
  NetworkSuspendInd(NetworkSRType),
  NetworkSuspendReq(NetworkSRType),
  NetworkResumeInd(NetworkSRType),
  NetworkResumeReq(NetworkSRType),
  DataReq(UserDataType,CallFlag),
  DataInd(UserDataType);

SIGNALLIST SigCon2_Out =
  NetworkSuspendReq,
  NetworkResumeReq,
  DataReq;

SIGNALLIST SigCon2_In =
  NetworkSuspendInd,
  NetworkResumeInd,
  DataInd;
```

```
/* Definition of primitives for the extension to the internal
interface between BCSMs (IBI). */
SIGNAL
  NetworkSuspendReqInd(NetworkSRType, CSAID, LegType),
  NetworkResumeReqInd(NetworkSRType, CSAID, LegType),
  DataReqInd(UserDataType, CSAID, LegType, CallFlag);

SIGNALLIST IBI2 =
  NetworkSuspendReqInd,
  NetworkResumeReqInd,
  DataReqInd;
```

Annex A:


SCF
(CS2_INAP_To_SCF)
(CS2_INAP_From_SCF)
Redefined Block Type <<System Type CS2_INAP>> SSF_CCF
1(4)
(CS2_INAP_To_SCF)
SCF2_Interface
(CS2_INAP_From_SCF)
SigCon
SigCon2_Interface
SCF
SigCon
IH
IBI
IBI2
IBI
(Sig_ Con2_ _Out)
(SigCon2_Out)
(SigCon2_In)
CSA
SSME
(IBI2)
(IBI2)
(IBI2)
(IBI2)
(Sig_ Con2_ _In)
(CSA2_Out)
(SSME2_Out)
/* Notes:
− Dashed symbols refer to entities defined in the IN CS−1 SDL specifications.
− The functionality and interfacesof all entities are extended to incorporate the IN CS−2 functionality. */
CSA2_Interface
SSME2_Interface
(CSA2_In)
(SSME2_In)
IH
CSA
CS
IH
SSME
(CS2_Out)
CS2_Interface
(CS2_In)
CSA
CS
SSF
SSF2_Interface
CS
SSF
O
T
(SSF2_ _Out)
(SSF2_ _In)
(O_BCSM2_Out)
(T_BCSM2_Out)
O_BCSM2
T_BCSM2
(O_BCSM2_In)
(T_BCSM2_In)
CS
O_BCSM
CS
T_BCSM

Annex A:


Redefined Block Type <<System Type CS2_INAP>> SSF_CCF
2(4)
/**** PROCESS TYPE DEFINITIONS ****/
/* Note: All process type definitions are inherited from IN CS−1 and redefined to incorporate the new functionality required for IN CS−2.
redefined Interface_ Handler
The InterFaceHandler (IH) process type is extended to manage:
− Processing of the IN CS−2 operations MoveCallSegments and CreateCSA.
− The passing of new primitives between the Signalling Control Interface and the CSAs.
− The passing of new primitives between the IN CS−2 SSF−SCF Interface and the CSA.
redefined CallSegment_ Association
The CallSegmentAssociation (CSA) process type is extended to manage:
− Processing of IN CS−2 call segment related operations.
− The passing of new Signalling Control primitives between the InterfaceHandler and the CallSegment.
− The passing of IN CS−2 primitives between the InterfaceHandler and the CallSegment.
redefined CallSegment
The CallSegment (CS) process type is extended to manage:
− The extended connection view (in terms of legs) of the call segment.
− Processing of IN CS−2 leg related operations.
− Filtering of detection points
− The passing of new Signalling Control primitives between the CallSegmentAssociation and the BCSMs.
− The passing of IN CS−2 primitives between the CallSegmentAssociation and the SSF−FSM.
redefined SSF_FSM
The SSF−FSM process type is extended to manage:
− The processing of IN CS−2 operations.
redefined SSME_FSM
The SSME_FSM is extended to ma
− The processing of ManageTrigg
redefined Originating_ BCSM
The OriginatingBCSM (O−BCSM) process type is extended to manage:
− The extensions to the call processing of originating half calls.
redefined Terminating_ BCSM
The TerminatingBCSM (T−BCSM) process type is extended to manage:
− The extensions to the call processing of terminating half calls.

Annex A:

Redefined Block Type <<System Type CS2_INAP>> SSF_CCF 3(4)

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

```
/* A Leg can be exported/imported between call segments. */

NEWTYPE ExportLegType STRUCT
  sigConId CallRef; /* Used when exporting the controlling leg. */
  csLegData LegInfo;
  ssfLegData EventRecordType;
  controllingLegStatus LegStatusType;
ENDNEWTYPE;


/* A call segment can be exported/imported between call segment associations. */

SYNTYPE ExportCSType = PId
ENDSYNTYPE;
```

```
SIGNAL
  /* Signals for importing and exporting legs between CS and SSF−FSM. */
  ExportLegReq(LegType),
  ExportLegResp(EventRecordType),
  ImportLegReq(LegType,EventRecordType),
  ImportLegResp,

  /* Signals for importing and exporting legs between CSA and  CS. */
  CSExportLegReq(LegType),
  CSExportLegResp(ExportLegType),
  CSImportLegReq(LegType,ExportLegType),
  CSImportLegResp,

  /* Signal to change the value of the CSA pointer in the CSs when the CS
     has been moved into another CSA. */
  SetCSA,

  /* Signal to change the value of the CS pointer in the BCSMs when the leg
     has been moved into another call segment. */
  SetCS,

  /* Signals for importing and exporting call segments between IH and CSA. */
  ExportCSReq(CallSegmentID, ExportLegsType),
  ExportCSResp(ExportCSType, Boolean),
  ImportCSReq(CallSegmentID,ExportCSType, ExportLegsType, Boolean),
  ImportCSResp,
  SetLegID(LegType,LegType),

  /* Signal to report to the SSF−FSM that an UTSI event has occured. */
  DPUTSI(DPUTSIArg);
```

Annex A:

Redefined Block Type <<System Type CS2_INAP>> SSF_CCF 4(4)

```
SIGNALLIST CS2_INAP_CSA_IN =
  ContinueWithArgument,
  CreateCallSegmentAssociation,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  MergeCallSegments,
  MoveLeg,
  RequestReportUTSI,
  SendSTUI,
  SplitLeg,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun;

SIGNALLIST CS2_INAP_CSA_OUT =
  CreateCallSegmentAssociationResult,
  DisconnectLegResult,
  EntityReleased,
  MergeCallSegmentsResult,
  MoveLegResult,
  ReportUTSI,
  SplitLegResult,

  ScriptEvent,
  PromptAndReceiveMessageResult;
```

```
SIGNALLIST CS2_INAP_CS_IN =
  ContinueWithArgument,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  RequestReportUTSI,
  SendSTUI,
  SplitLeg,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun;

SIGNALLIST CS2_INAP_CS_OUT =
  DisconnectLegResult,
  EntityReleased,
  ReportUTSI,
  SplitLegResult,

  ScriptEvent,
  PromptAndReceiveMessageResult;
```

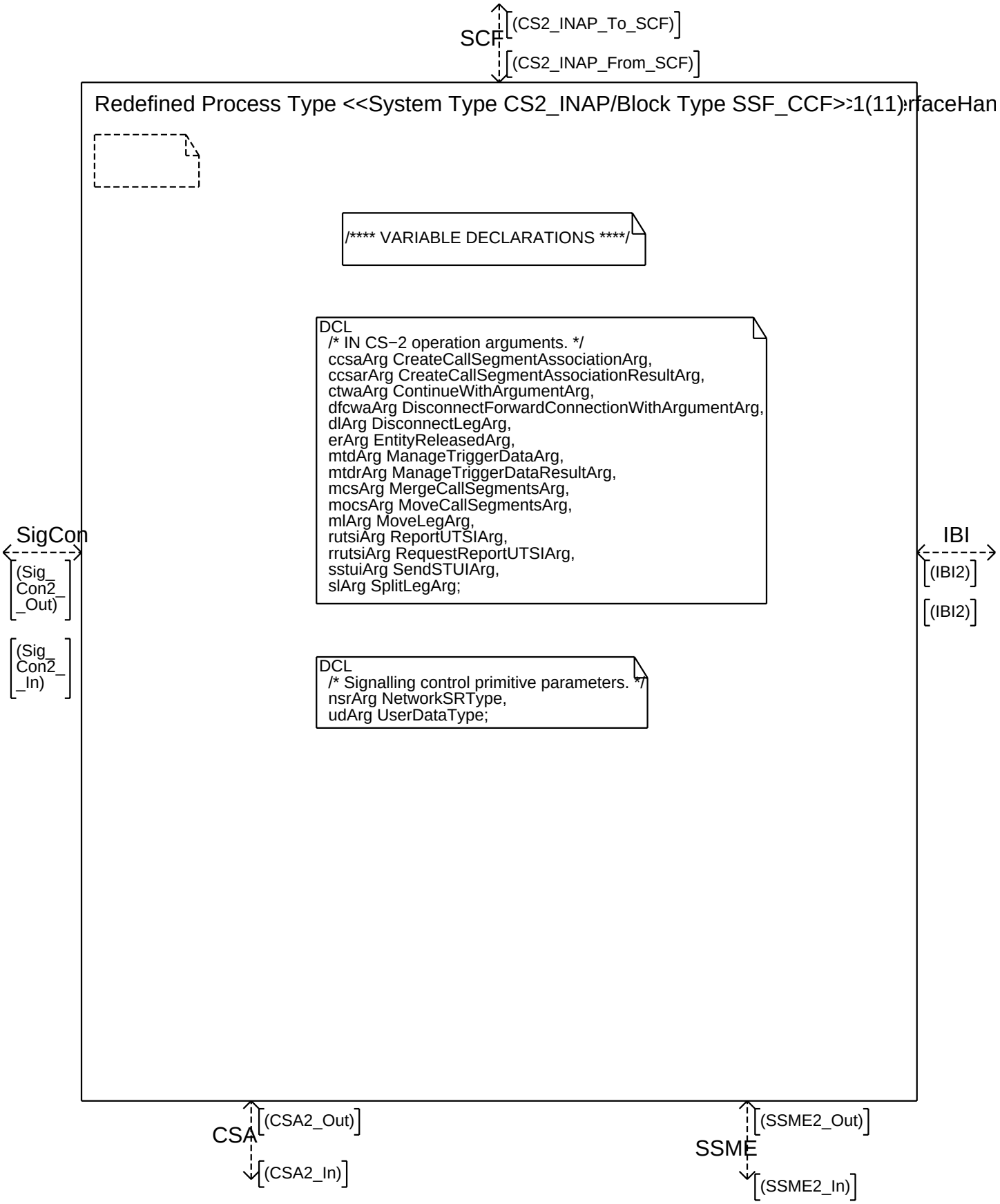
```
SIGNALLIST CS2_INAP_SSME
  ManageTriggerData;
SIGNALLIST CS2_INAP_SSME
  ManageTriggerDataResult;
```

```
SIGNALLIST CS2_INAP_SSF_IN =
  ContinueWithArgument,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  RequestReportUTSI,
  SendSTUI,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun;

SIGNALLIST CS2_INAP_SSF_OUT =
  DisconnectLegResult,
  ReportUTSI,

  ScriptEvent,
  PromptAndReceiveMessageResult;
```

```
/* Definition of signal lists. */

SIGNALLIST CSA2_In =
  (CS2_INAP_CSA_IN),
  (IBI2),
  (SigCon2_In),
  ExportCSReq,
  ImportCSReq;

SIGNALLIST CSA2_Out =
  (CS2_INAP_CSA_OUT),
  (IBI2),
  (SigCon2_Out),
  ExportCSResp,
  ImportCSResp;

SIGNALLIST CS2_In =
  (CS2_INAP_CS_IN),
  (IBI2),
  (SigCon2_In),
  CSExportLegReq,
  CSImportLegReq,
  SetCSA,
  SetLegID;

SIGNALLIST CS2_Out =
  (CS2_INAP_CS_OUT),
  (IBI2),
  (SigCon2_Out),
  CSExportLegResp,
  CSImportLegResp;

SIGNALLIST SSF2_In =
  (CS2_INAP_SSF_IN),
  ExportLegReq,
  ImportLegReq,
  SetLegID,
  DPUTSI;

SIGNALLIST SSF2_Out =
```

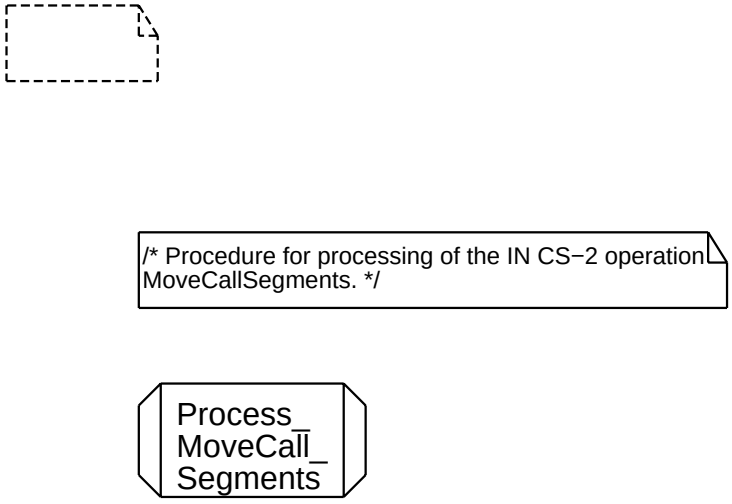
```
SIGNALLIST SSME2_In =
  (CS2_INAP_SSME_IN);

SIGNALLIST SSME2_Out =
  (CS2_INAP_SSME_OUT);

SIGNALLIST O_BCSM2_In =
  NetworkSuspendReqInd,
  NetworkResumeReqInd,
  DataReqInd,
  DataInd,
  SetCS,
  SetLegID;

SIGNALLIST O_BCSM2_Out =
  NetworkSuspendReq,
  NetworkResumeReq,
  DataReqInd,
  DataReq,
  DPUTSI;

SIGNALLIST T_BCSM2_In =
  NetworkSuspendInd,
  NetworkResumeInd,
  DataInd,
  DataReqInd,
  SetCS,
  SetLegID;

SIGNALLIST T_BCSM2_Out =
  NetworkSuspendReqInd,
  NetworkResumeReqInd,
  DataReq,
  DataReqInd,
  DPUTSI;
```

Annex A:

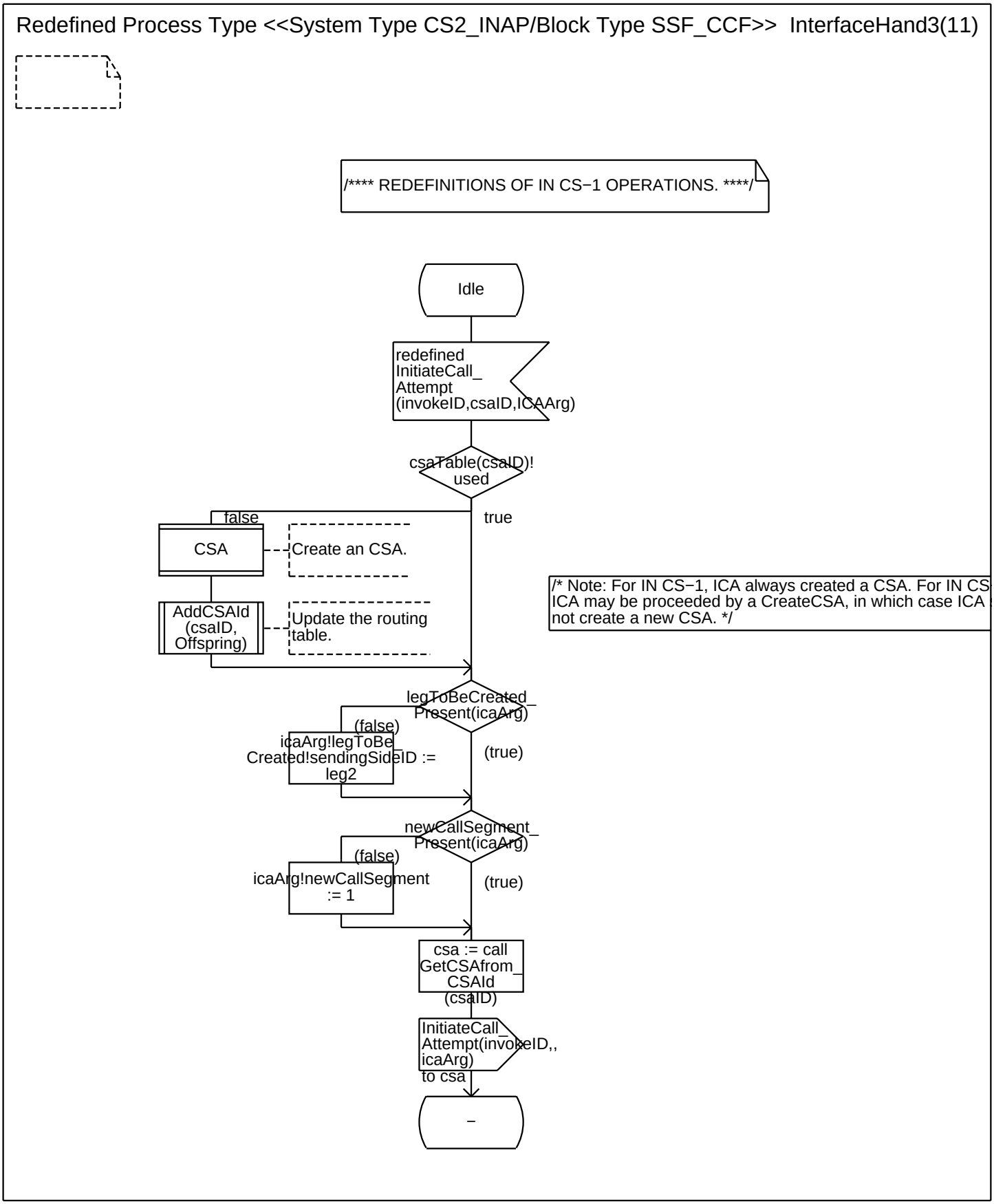

SCF
(CS2_INAP_To_SCF)
(CS2_INAP_From_SCF)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>>1(11):rfaceHan
/**** VARIABLE DECLARATIONS ****/
DCL
/* IN CS−2 operation arguments. */
ccsaArg CreateCallSegmentAssociationArg,
ccsarArg CreateCallSegmentAssociationResultArg,
ctwaArg ContinueWithArgumentArg,
dfcwaArg DisconnectForwardConnectionWithArgumentArg,
dlArg DisconnectLegArg,
erArg EntityReleasedArg,
mtdArg ManageTriggerDataArg,
mtdrArg ManageTriggerDataResultArg,
mcsArg MergeCallSegmentsArg,
mocsArg MoveCallSegmentsArg,
mlArg MoveLegArg,
rutsiArg ReportUTSIArg,
rrutsiArg RequestReportUTSIArg,
sstuiArg SendSTUIArg,
slArg SplitLegArg;
SigCon
(Sig_Con2__Out)
(Sig_Con2_In)
IBI
(IBI2)
(IBI2)
DCL
/* Signalling control primitive parameters. */
nsrArg NetworkSRType,
udArg UserDataType;
CSA
(CSA2_Out)
(CSA2_In)
SSME
(SSME2_Out)
(SSME2_In)

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand2(11)

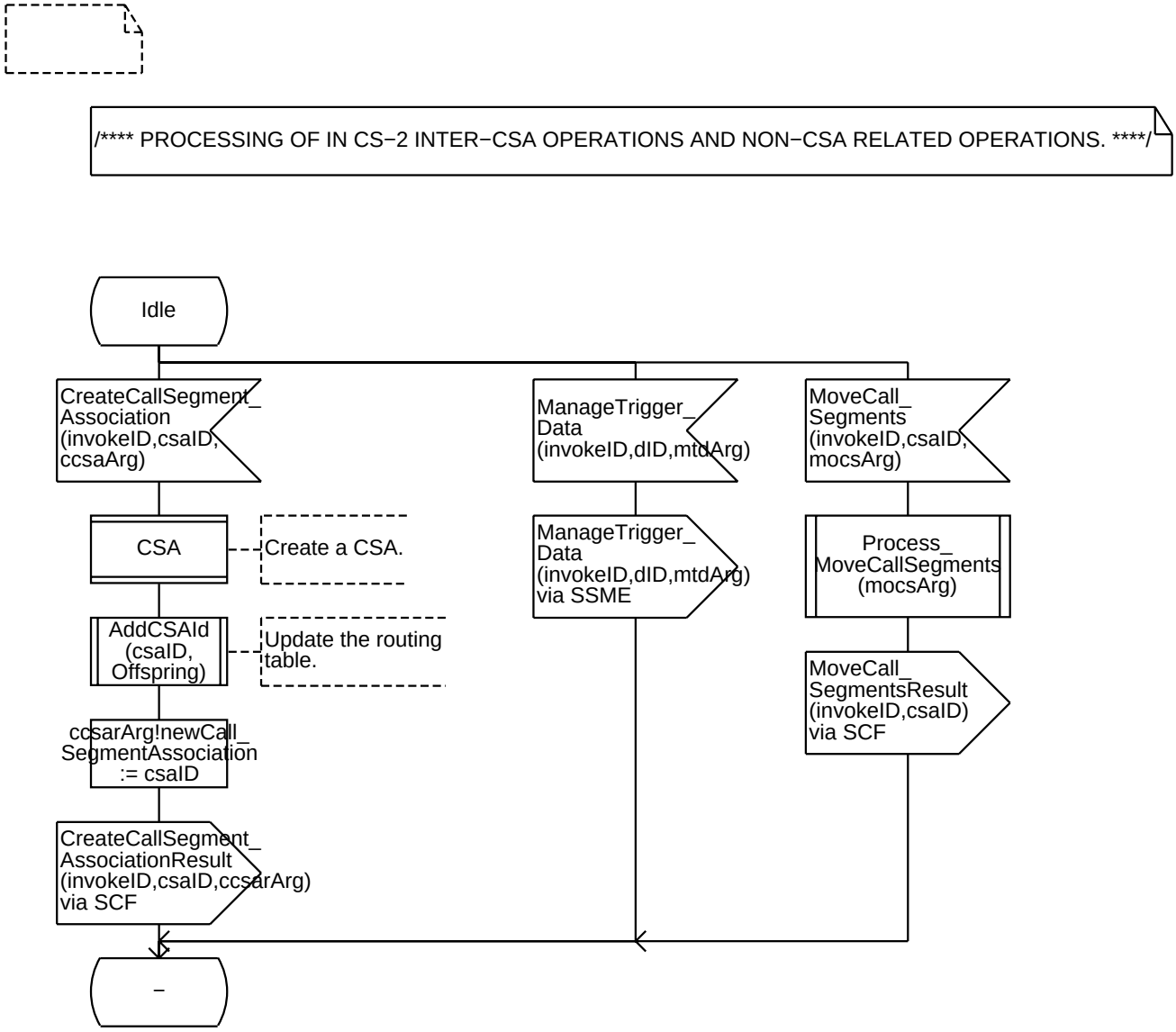


Annex A:

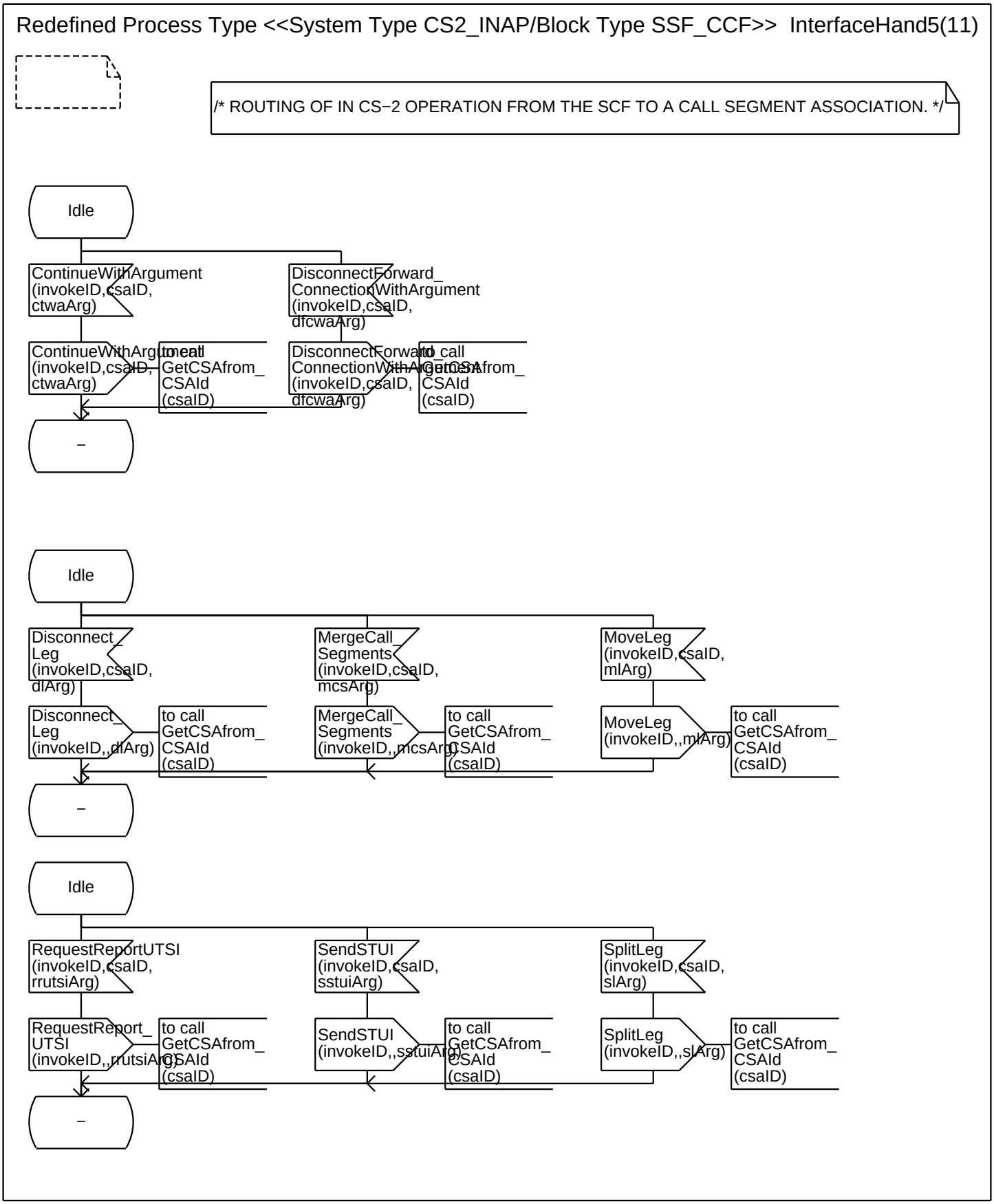

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand3(11)
/**** REDEFINITIONS OF IN CS−1 OPERATIONS. ****/
Idle
redefined InitiateCall_ Attempt (invokeID,csaID,ICAArg)
csaTable(csaID)! used
false
true
CSA
Create an CSA.
AddCSAId (csaID, Offspring)
Update the routing table.
/* Note: For IN CS−1, ICA always created a CSA. For IN CS
ICA may be proceeded by a CreateCSA, in which case ICA
not create a new CSA. */
legToBeCreated_ Present(icaArg)
(false)
icaArg!legToBe_ Created!sendingSideID := leg2
(true)
newCallSegment_ Present(icaArg)
(false)
icaArg!newCallSegment := 1
(true)
csa := call GetCSAfrom_ CSAId (csaID)
InitiateCall_ Attempt(invokeID,, icaArg) to csa
–

Annex A:

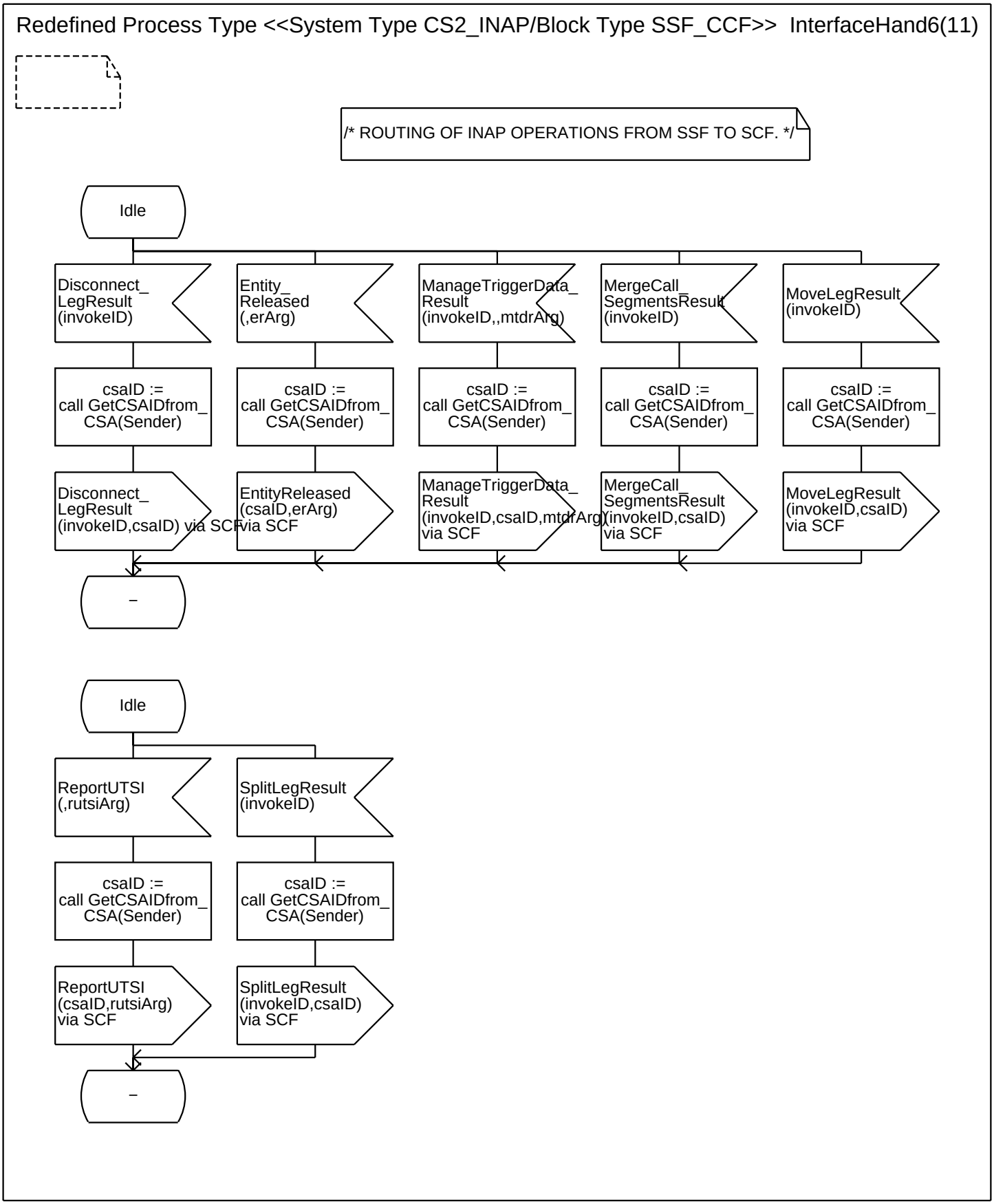
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand4(11)



Annex A:

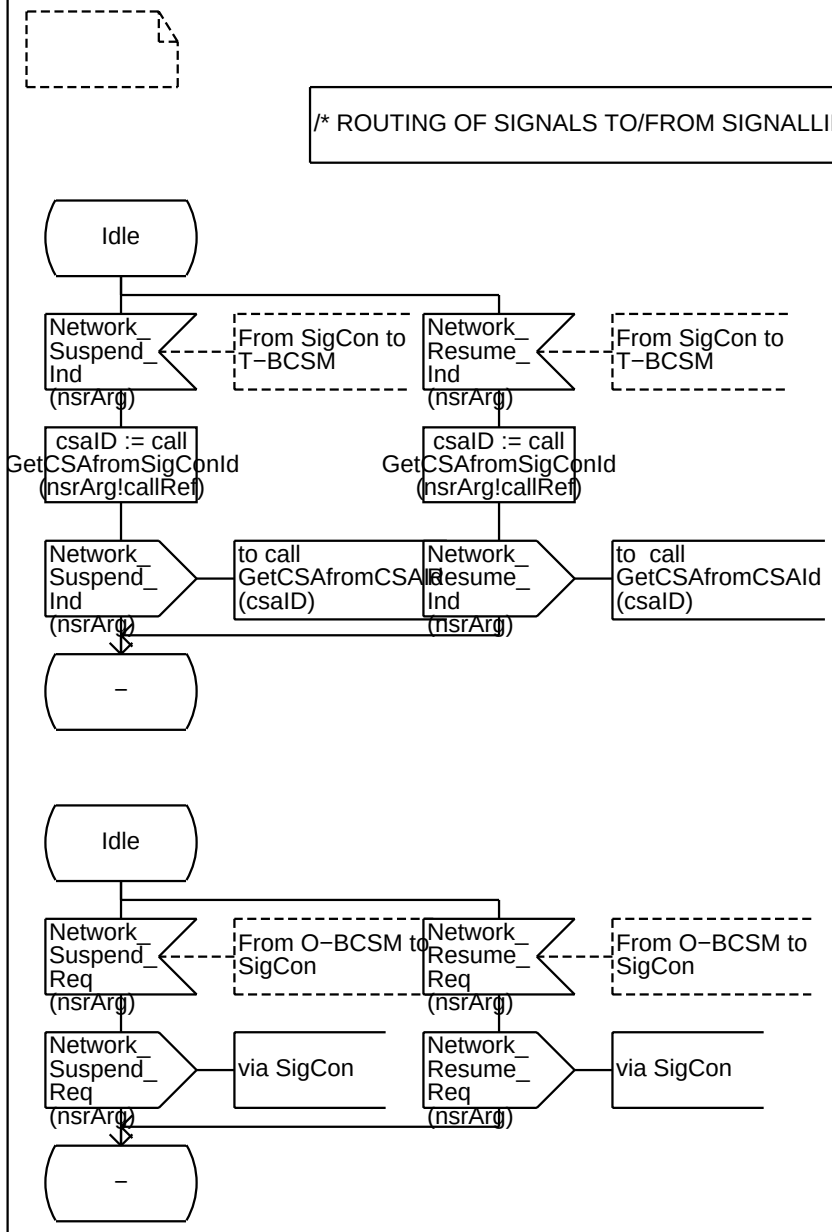

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand5(11)
/* ROUTING OF IN CS−2 OPERATION FROM THE SCF TO A CALL SEGMENT ASSOCIATION. */
Idle
ContinueWithArgument (invokeID,csaID, ctwaArg)
DisconnectForward_ ConnectionWithArgument (invokeID,csaID, dfcwaArg)
ContinueWithArgument (invokeID,csaID, ctwaArg)
to call GetCSAfrom_ CSAId (csaID)
DisconnectForward_ ConnectionWithArgument (invokeID,csaID, dfcwaArg)
to call GetCSAfrom_ CSAId (csaID)
–
Idle
Disconnect_ Leg (invokeID,csaID, dlArg)
MergeCall_ Segments (invokeID,csaID, mcsArg)
MoveLeg (invokeID,csaID, mlArg)
Disconnect_ Leg (invokeID,,dlArg)
to call GetCSAfrom_ CSAId (csaID)
MergeCall_ Segments (invokeID,,mcsArg)
to call GetCSAfrom_ CSAId (csaID)
MoveLeg (invokeID,,mlArg)
to call GetCSAfrom_ CSAId (csaID)
–
Idle
RequestReportUTSI (invokeID,csaID, rrutsiArg)
SendSTUI (invokeID,csaID, sstuiArg)
SplitLeg (invokeID,csaID, slArg)
RequestReport_ UTSI (invokeID,,rrutsiArg)
to call GetCSAfrom_ CSAId (csaID)
SendSTUI (invokeID,,sstuiArg)
to call GetCSAfrom_ CSAId (csaID)
SplitLeg (invokeID,,slArg)
to call GetCSAfrom_ CSAId (csaID)
–

Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand6(11)
/* ROUTING OF INAP OPERATIONS FROM SSF TO SCF. */
Idle
Disconnect_ LegResult (invokeID)
Entity_ Released (,erArg)
ManageTriggerData_ Result (invokeID,,mtdrArg)
MergeCall_ SegmentsResult (invokeID)
MoveLegResult (invokeID)
csaID := call GetCSAIDfrom_ CSA(Sender)
csaID := call GetCSAIDfrom_ CSA(Sender)
csaID := call GetCSAIDfrom_ CSA(Sender)
csaID := call GetCSAIDfrom_ CSA(Sender)
csaID := call GetCSAIDfrom_ CSA(Sender)
Disconnect_ LegResult (invokeID,csaID) via SCF
EntityReleased (csaID,erArg) via SCF
ManageTriggerData_ Result (invokeID,csaID,mtdrArg) via SCF
MergeCall_ SegmentsResult (invokeID,csaID) via SCF
MoveLegResult (invokeID,csaID) via SCF
–
Idle
ReportUTSI (,rutsiArg)
SplitLegResult (invokeID)
csaID := call GetCSAIDfrom_ CSA(Sender)
csaID := call GetCSAIDfrom_ CSA(Sender)
ReportUTSI (csaID,rutsiArg) via SCF
SplitLegResult (invokeID,csaID) via SCF
–

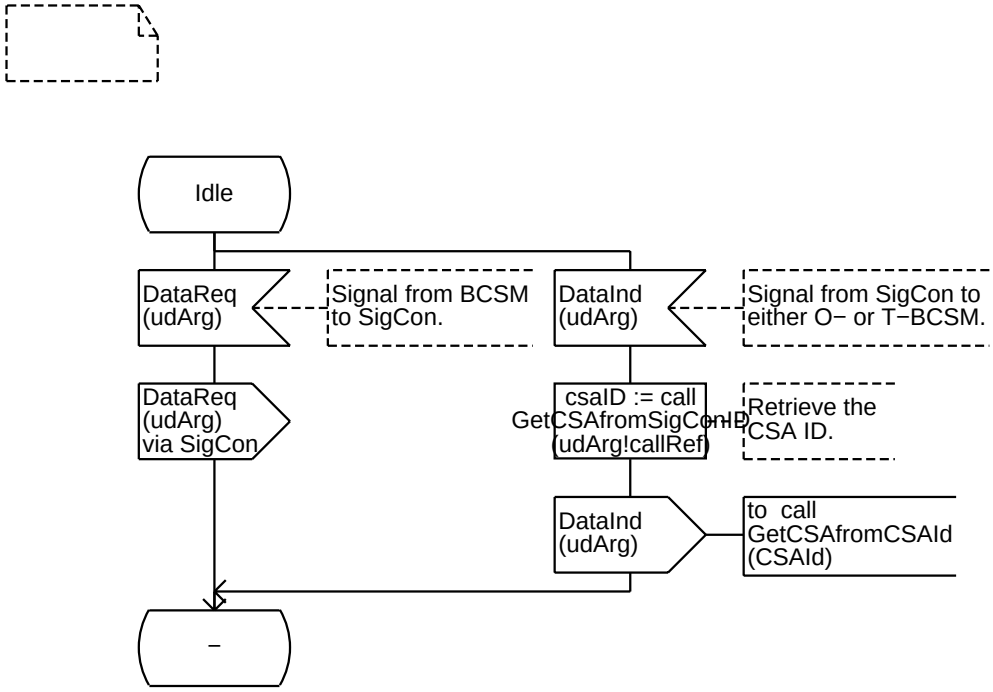
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand7(11)

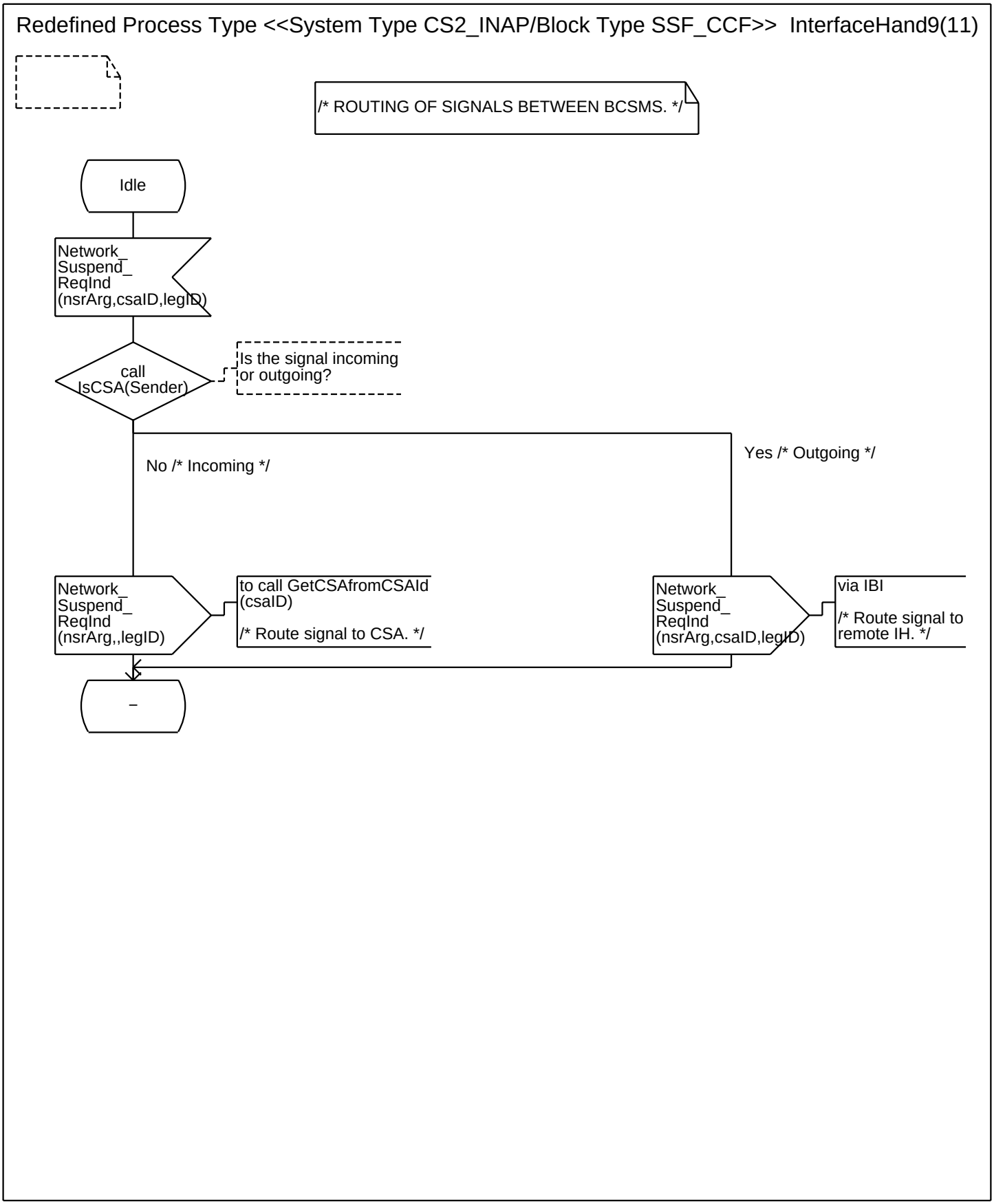


Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand8(11)

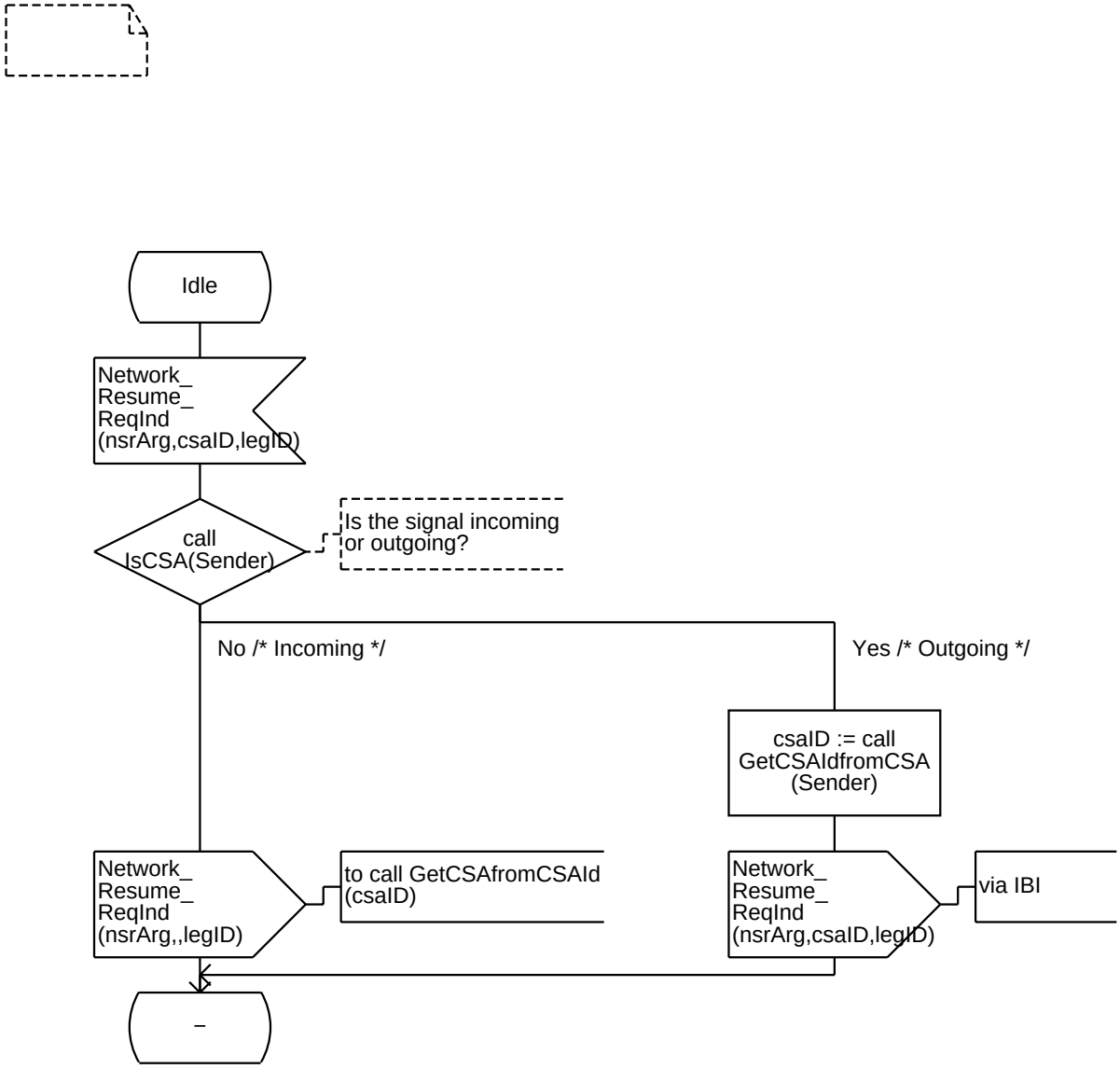


Annex A:

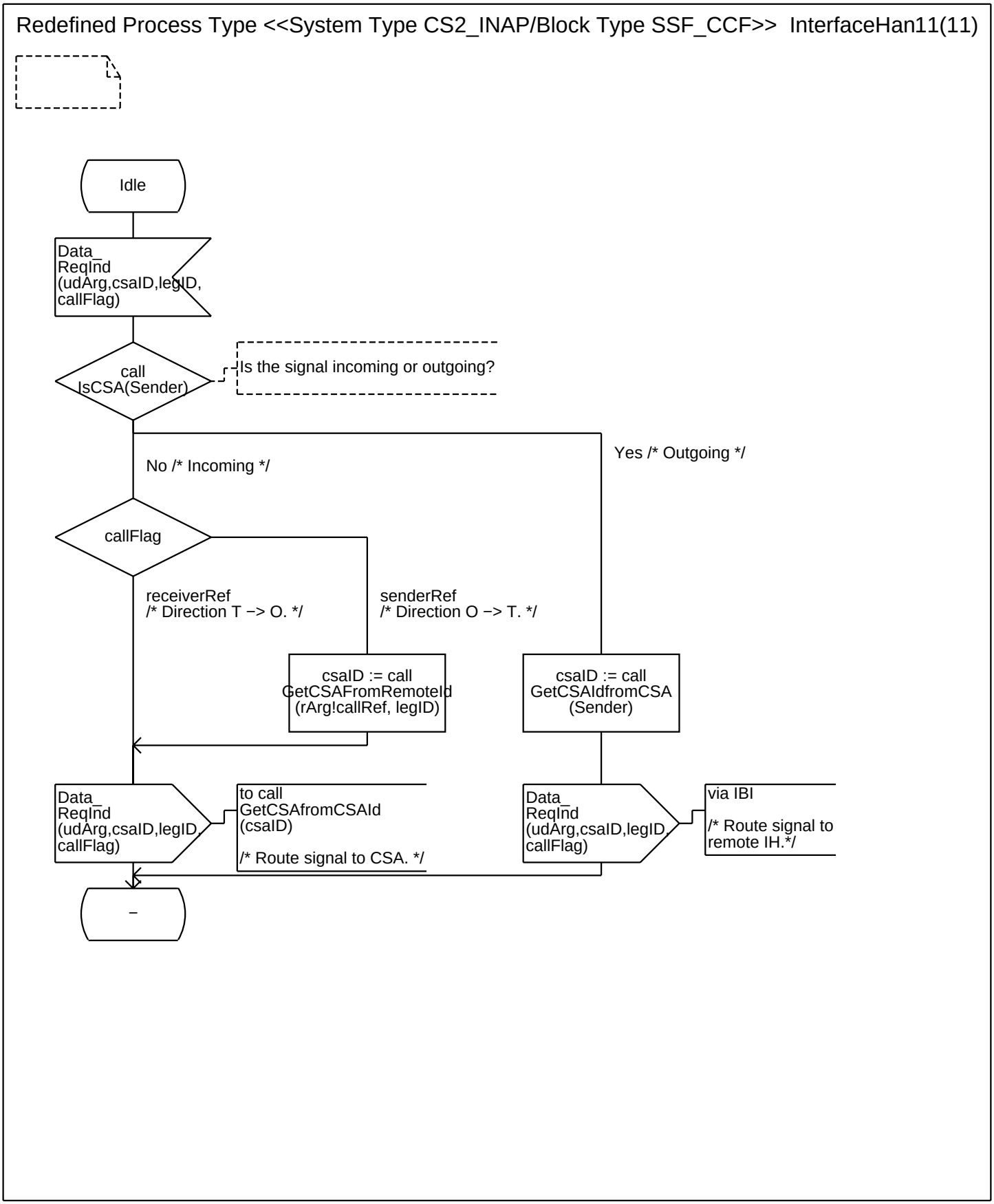

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHand9(11)
/* ROUTING OF SIGNALS BETWEEN BCSMS. */
Idle
Network_ Suspend_ ReqInd (nsrArg,csaID,legID)
call IsCSA(Sender)
Is the signal incoming or outgoing?
No /* Incoming */
Yes /* Outgoing */
Network_ Suspend_ ReqInd (nsrArg,,legID)
to call GetCSAfromCSAId (csaID)
/* Route signal to CSA. */
Network_ Suspend_ ReqInd (nsrArg,csaID,legID)
via IBI
/* Route signal to remote IH. */
–

Annex A:

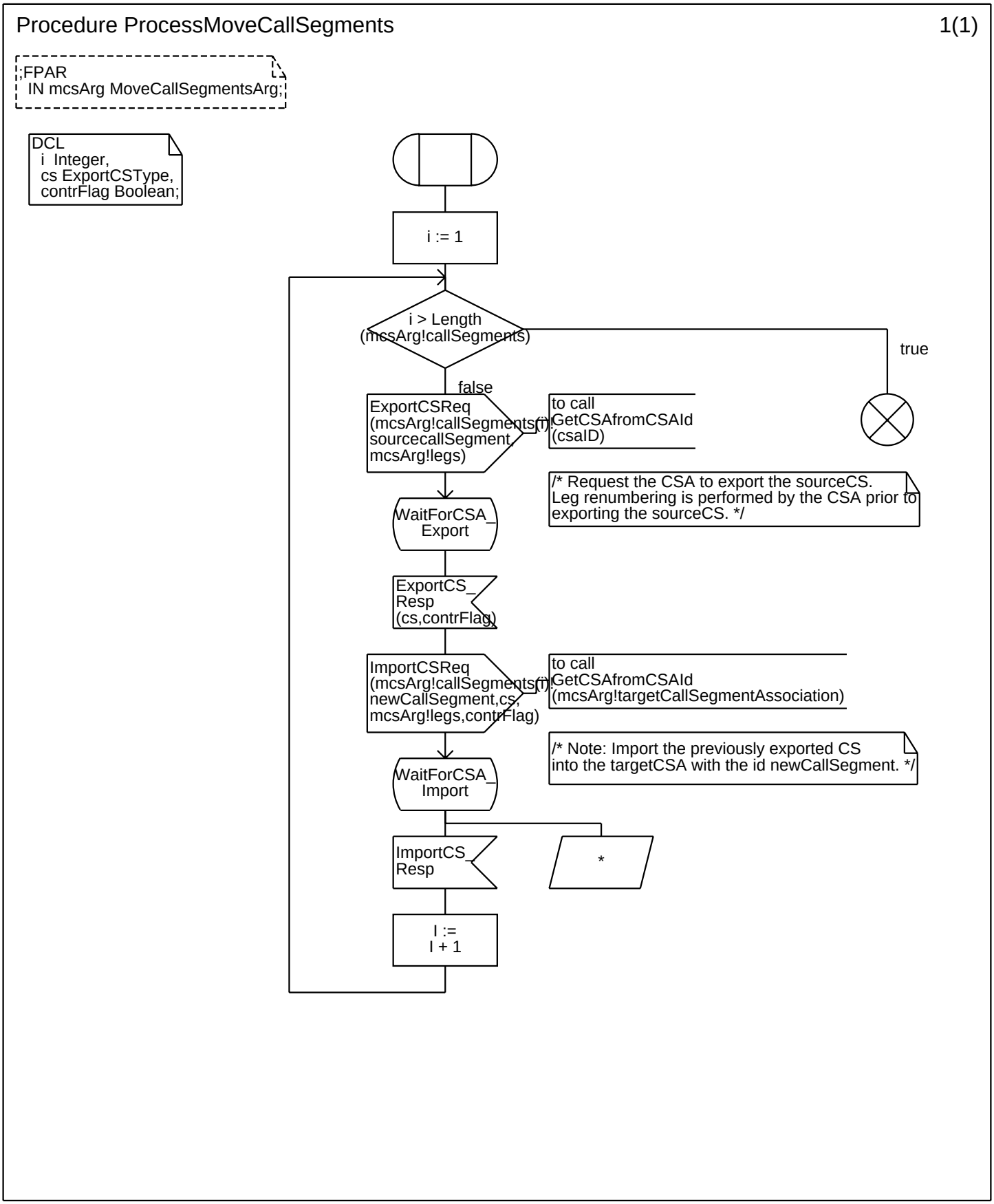
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHan10(11)



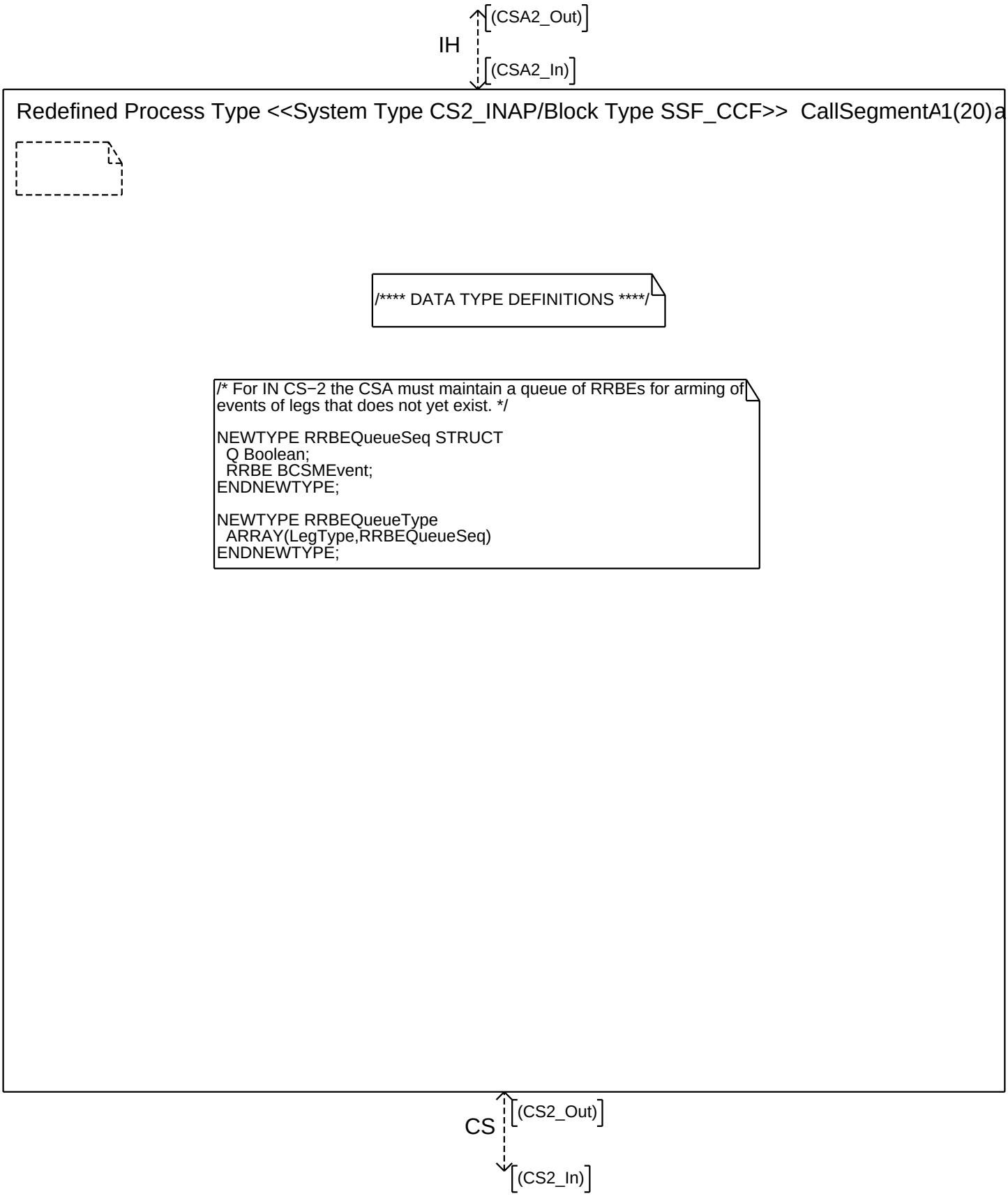
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHan11(11)
Idle
Data_ ReqInd (udArg,csaID,legID, callFlag)
call IsCSA(Sender)
Is the signal incoming or outgoing?
No /* Incoming */
Yes /* Outgoing */
callFlag
receiverRef /* Direction T −> O. */
senderRef /* Direction O −> T. */
csaID := call GetCSAFromRemoteId (rArg!callRef, legID)
csaID := call GetCSAIdfromCSA (Sender)
Data_ ReqInd (udArg,csaID,legID, callFlag)
to call GetCSAfromCSAId (csaID)
/* Route signal to CSA. */
Data_ ReqInd (udArg,csaID,legID, callFlag)
via IBI
/* Route signal to remote IH.*/
–

Annex A:


Procedure ProcessMoveCallSegments
1(1)
;FPAR
IN mcsArg MoveCallSegmentsArg;
DCL
i Integer,
cs ExportCSType,
contrFlag Boolean;
i := 1
i > Length
(mcsArg!callSegments)
true
false
ExportCSReq
(mcsArg!callSegments(i),
sourcecallSegment,
mcsArg!legs)
to call
GetCSAfromCSAId
(csaID)
/* Request the CSA to export the sourceCS.
Leg renumbering is performed by the CSA prior to
exporting the sourceCS. */
WaitForCSA_
Export
ExportCS_
Resp
(cs,contrFlag)
ImportCSReq
(mcsArg!callSegments(i),
newCallSegment,cs,
mcsArg!legs,contrFlag)
to call
GetCSAfromCSAId
(mcsArg!targetCallSegmentAssociation)
/* Note: Import the previously exported CS
into the targetCSA with the id newCallSegment. */
WaitForCSA_
Import
ImportCS_
Resp
*
I :=
I + 1

Annex A:


IH
(CSA2_Out)
(CSA2_In)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA1(20)a
/**** DATA TYPE DEFINITIONS ****/
/* For IN CS−2 the CSA must maintain a queue of RRBEs for arming of events of legs that does not yet exist. */
NEWTYPE RRBEQueueSeq STRUCT
Q Boolean;
RRBE BCSMEvent;
ENDNEWTYPE;
NEWTYPE RRBEQueueType
ARRAY(LegType,RRBEQueueSeq)
ENDNEWTYPE;
CS
(CS2_Out)
(CS2_In)

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA2(20)a

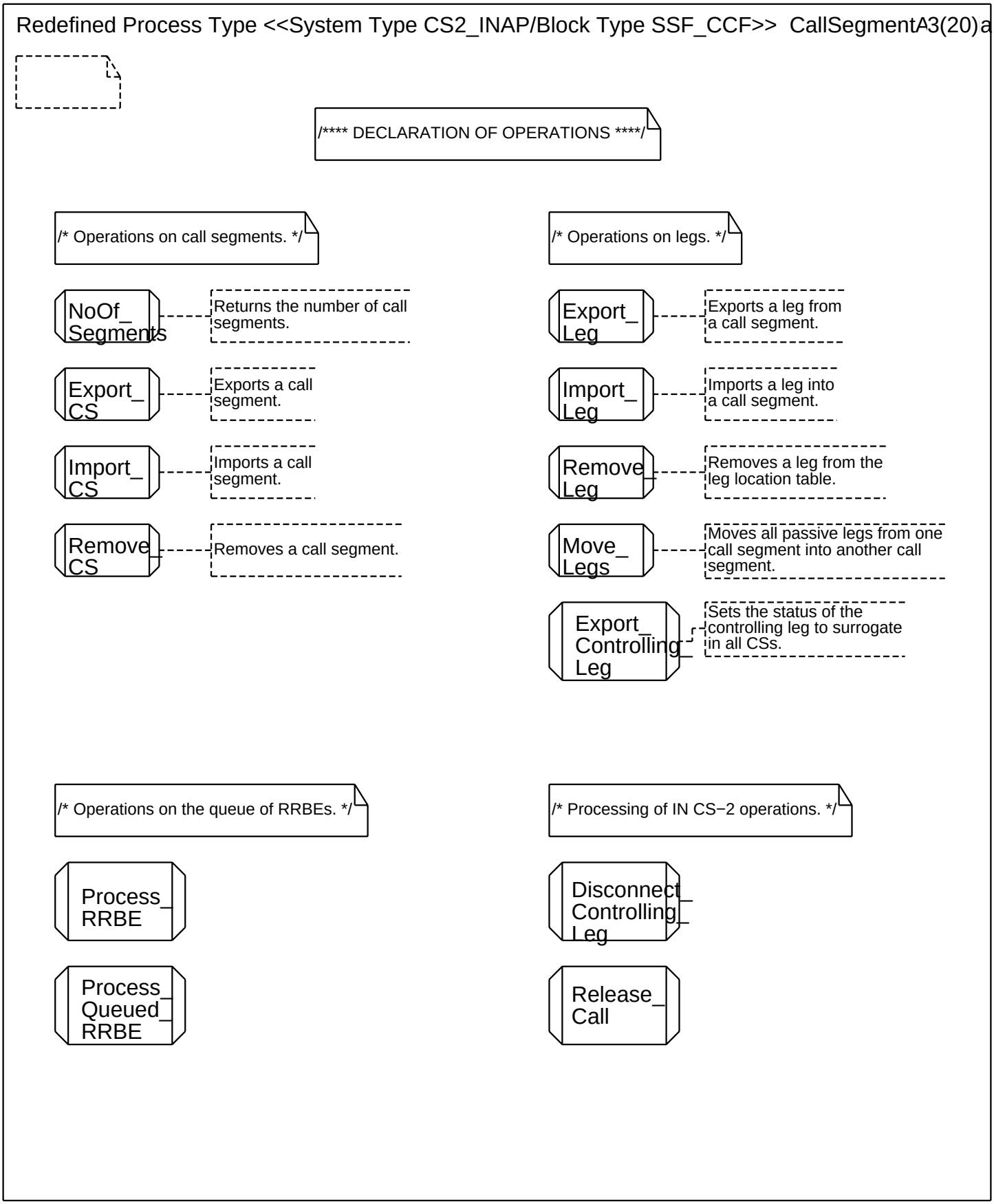
```
/**** VARIABLE DECLARATIONS ****/
```

```
DCL
  /* IN CS−2 operation arguments. */
  ctwaArg ContinueWithArgumentArg,
  dfcwaArg DisconnectForwardConnectionWithArgumentArg,
  dlArg DisconnectLegArg,
  erArg EntityReleasedArg,
  mcsArg MergeCallSegmentsArg,
  mlArg MoveLegArg,
  rutsiArg ReportUTSIArg,
  rrutsiArg RequestReportUTSIArg,
  sstuiArg SendSTUIArg,
  slArg SplitLegArg;
```

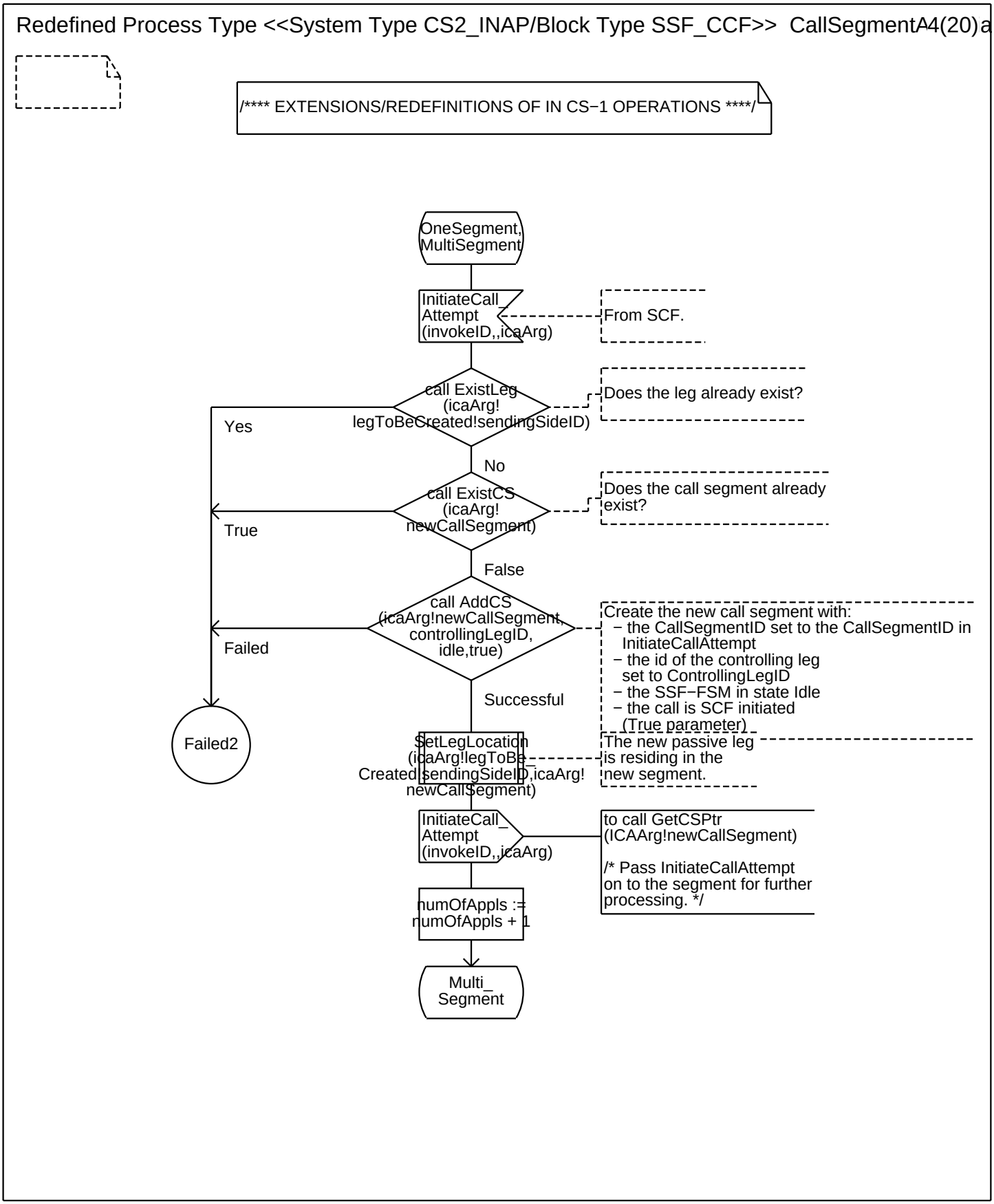
```
DCL
  /* Signalling control primitive parameters. */
  nsrArg NetworkSRType,
  udArg UserDataType;
```

```
DCL
  rrbeQueue RRBEQueueType,     /* The queued RRBEs. */

  contrFlag Boolean,
  leg   ExportLegType,
  legs  ExportLegsType,
  expCS ExportCSType,
  n Integer;
```

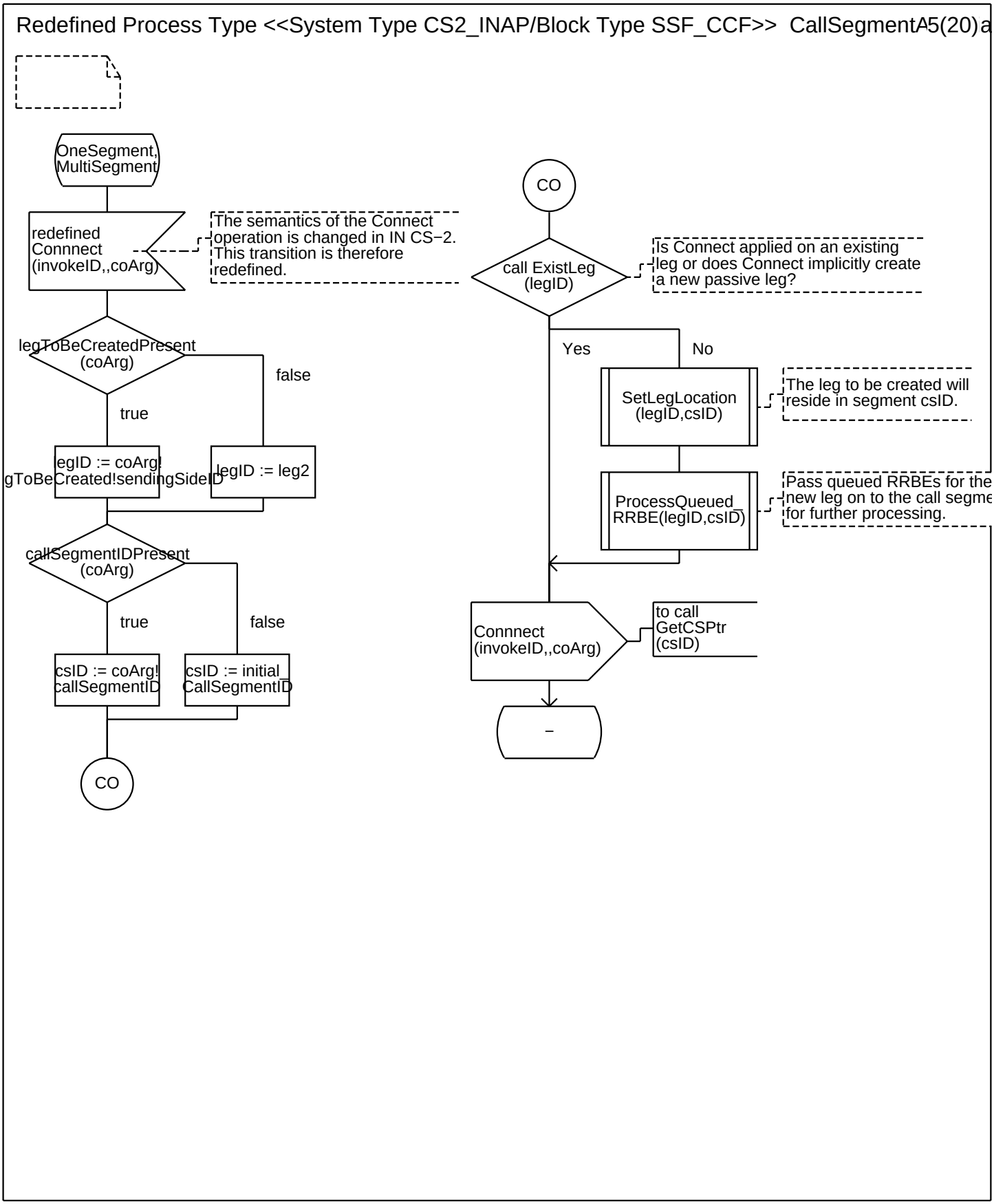
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA3(20)a
/**** DECLARATION OF OPERATIONS ****/
/* Operations on call segments. */
NoOf_Segments
Returns the number of call segments.
Export_CS
Exports a call segment.
Import_CS
Imports a call segment.
Remove_CS
Removes a call segment.
/* Operations on legs. */
Export_Leg
Exports a leg from a call segment.
Import_Leg
Imports a leg into a call segment.
Remove_Leg
Removes a leg from the leg location table.
Move_Legs
Moves all passive legs from one call segment into another call segment.
Export_Controlling_Leg
Sets the status of the controlling leg to surrogate in all CSs.
/* Operations on the queue of RRBEs. */
Process_RRBE
Process_Queued_RRBE
/* Processing of IN CS−2 operations. */
Disconnect_Controlling_Leg
Release_Call

Annex A:

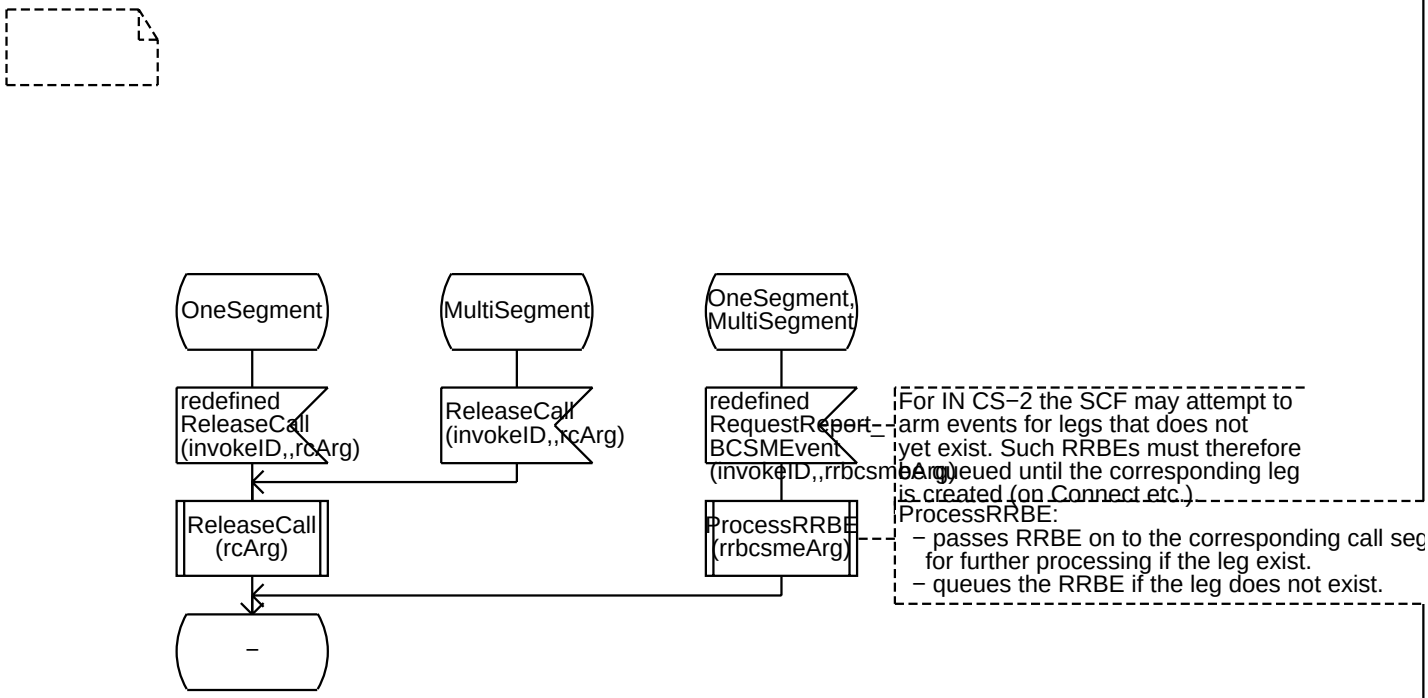

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA4(20)a
/**** EXTENSIONS/REDEFINITIONS OF IN CS−1 OPERATIONS ****/
OneSegment, MultiSegment
InitiateCall_ Attempt (invokeID,,icaArg)
From SCF.
call ExistLeg (icaArg! legToBeCreated!sendingSideID)
Does the leg already exist?
Yes
No
call ExistCS (icaArg! newCallSegment)
Does the call segment already exist?
True
False
call AddCS (icaArg!newCallSegment, controllingLegID, idle,true)
Create the new call segment with:
− the CallSegmentID set to the CallSegmentID in InitiateCallAttempt
− the id of the controlling leg set to ControllingLegID
− the SSF−FSM in state Idle
− the call is SCF initiated (True parameter)
Failed
Successful
Failed2
SetLegLocation (icaArg!legToBe_ Created!sendingSideID,icaArg! newCallSegment)
The new passive leg is residing in the new segment.
InitiateCall_ Attempt (invokeID,,icaArg)
to call GetCSPtr (ICAArg!newCallSegment)
/* Pass InitiateCallAttempt on to the segment for further processing. */
numOfAppls := numOfAppls + 1
Multi_ Segment

Annex A:

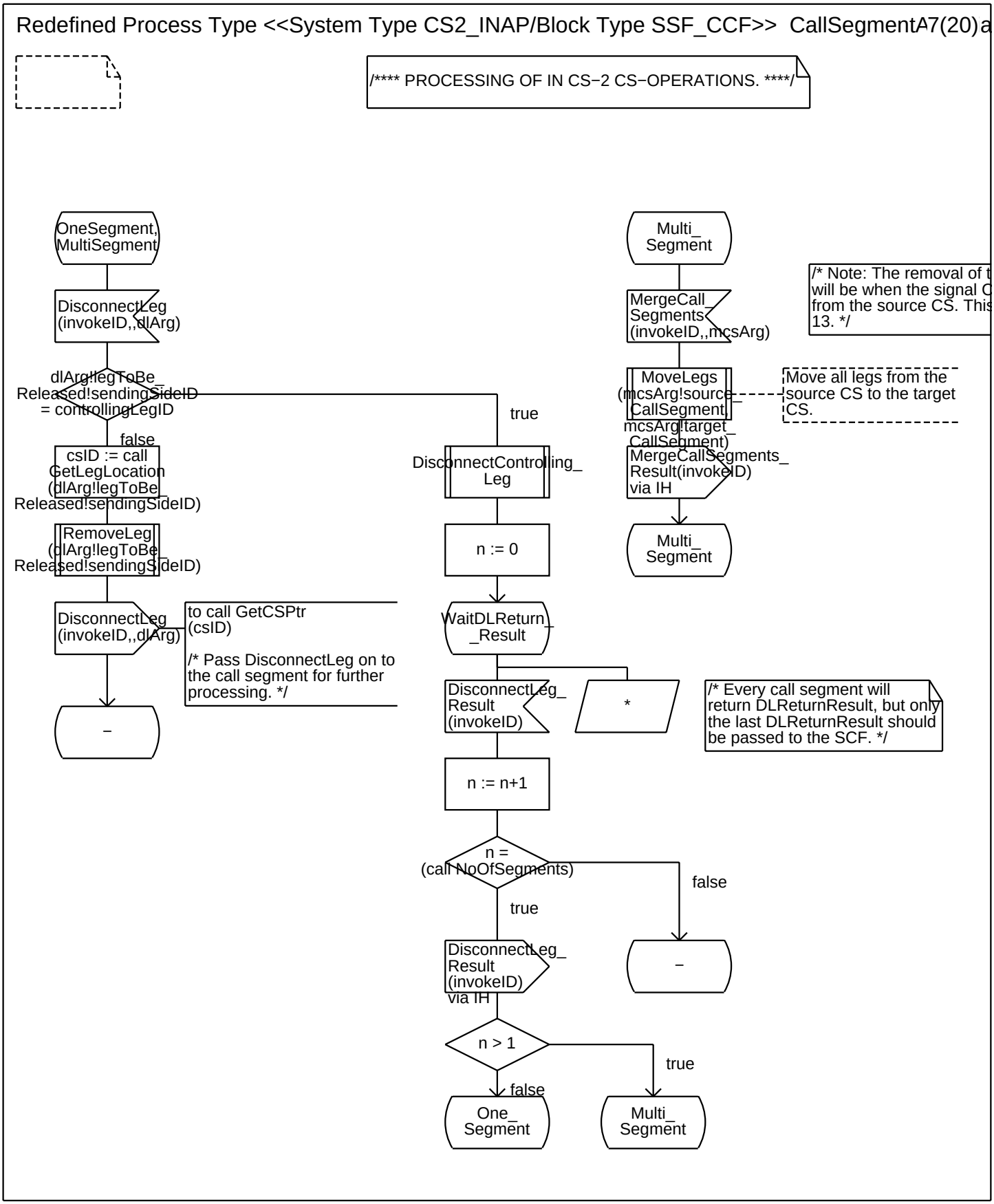

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA5(20)a
OneSegment, MultiSegment
redefined Connnect (invokeID,,coArg)
The semantics of the Connect operation is changed in IN CS−2. This transition is therefore redefined.
legToBeCreatedPresent (coArg)
false
true
legID := coArg! gToBeCreated!sendingSideID
legID := leg2
callSegmentIDPresent (coArg)
true
false
csID := coArg! callSegmentID
csID := initial_ CallSegmentID
CO
CO
call ExistLeg (legID)
Is Connect applied on an existing leg or does Connect implicitly create a new passive leg?
Yes
No
SetLegLocation (legID,csID)
The leg to be created will reside in segment csID.
ProcessQueued_ RRBE(legID,csID)
Pass queued RRBEs for the new leg on to the call segme for further processing.
Connnect (invokeID,,coArg)
to call GetCSPtr (csID)
–

Annex A:

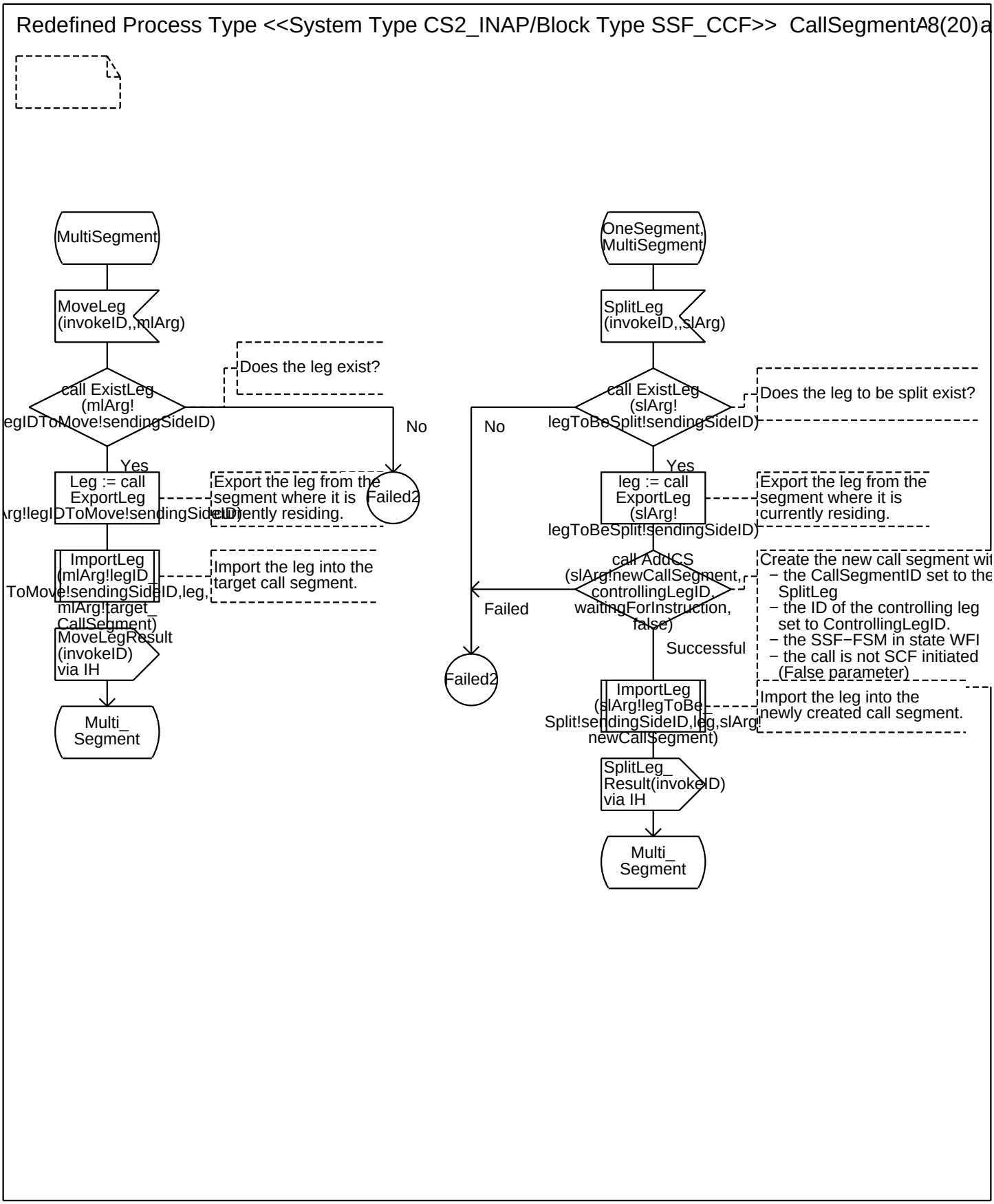
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA6(20)a



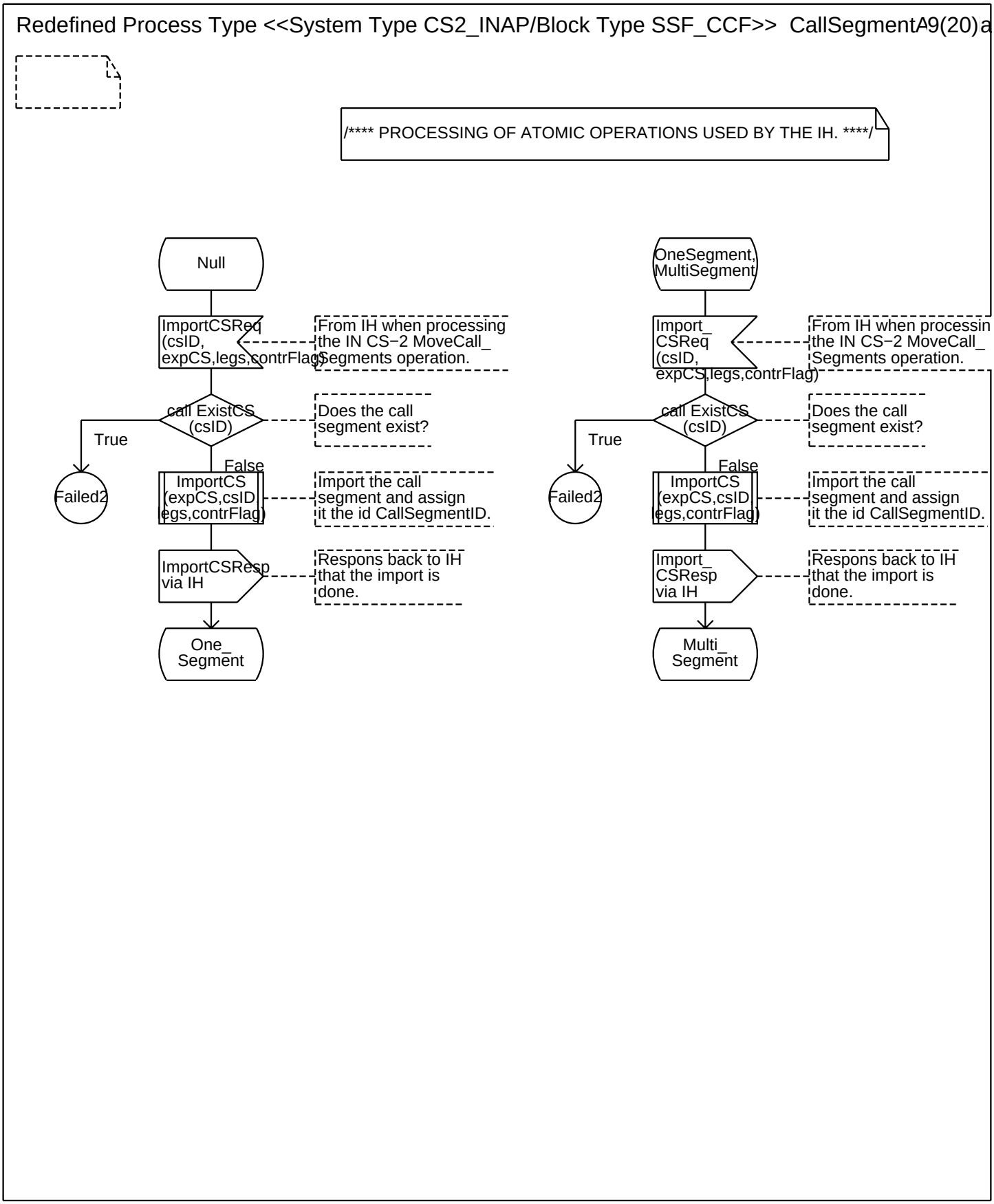
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA7(20)a
/**** PROCESSING OF IN CS−2 CS−OPERATIONS. ****/
OneSegment, MultiSegment
DisconnectLeg (invokeID,,dlArg)
dlArg!legToBe_ Released!sendingSideID = controllingLegID
false
true
csID := call GetLegLocation (dlArg!legToBe_ Released!sendingSideID)
RemoveLeg (dlArg!legToBe_ Released!sendingSideID)
DisconnectLeg (invokeID,,dlArg)
to call GetCSPtr (csID)
/* Pass DisconnectLeg on to the call segment for further processing. */
–
DisconnectControlling_ Leg
n := 0
WaitDLReturn_ _Result
DisconnectLeg_ Result (invokeID)
*
/* Every call segment will return DLReturnResult, but only the last DLReturnResult should be passed to the SCF. */
n := n+1
n = (callNoOfSegments)
true
false
DisconnectLeg_ Result (invokeID) via IH
–
n > 1
true
false
One_ Segment
Multi_ Segment
Multi_ Segment
MergeCall_ Segments (invokeID,,mcsArg)
/* Note: The removal of t will be when the signal C from the source CS. This 13. */
MoveLegs (mcsArg!source_ CallSegment, mcsArg!target_ CallSegment)
Move all legs from the source CS to the target CS.
MergeCallSegments_ Result(invokeID) via IH
Multi_ Segment

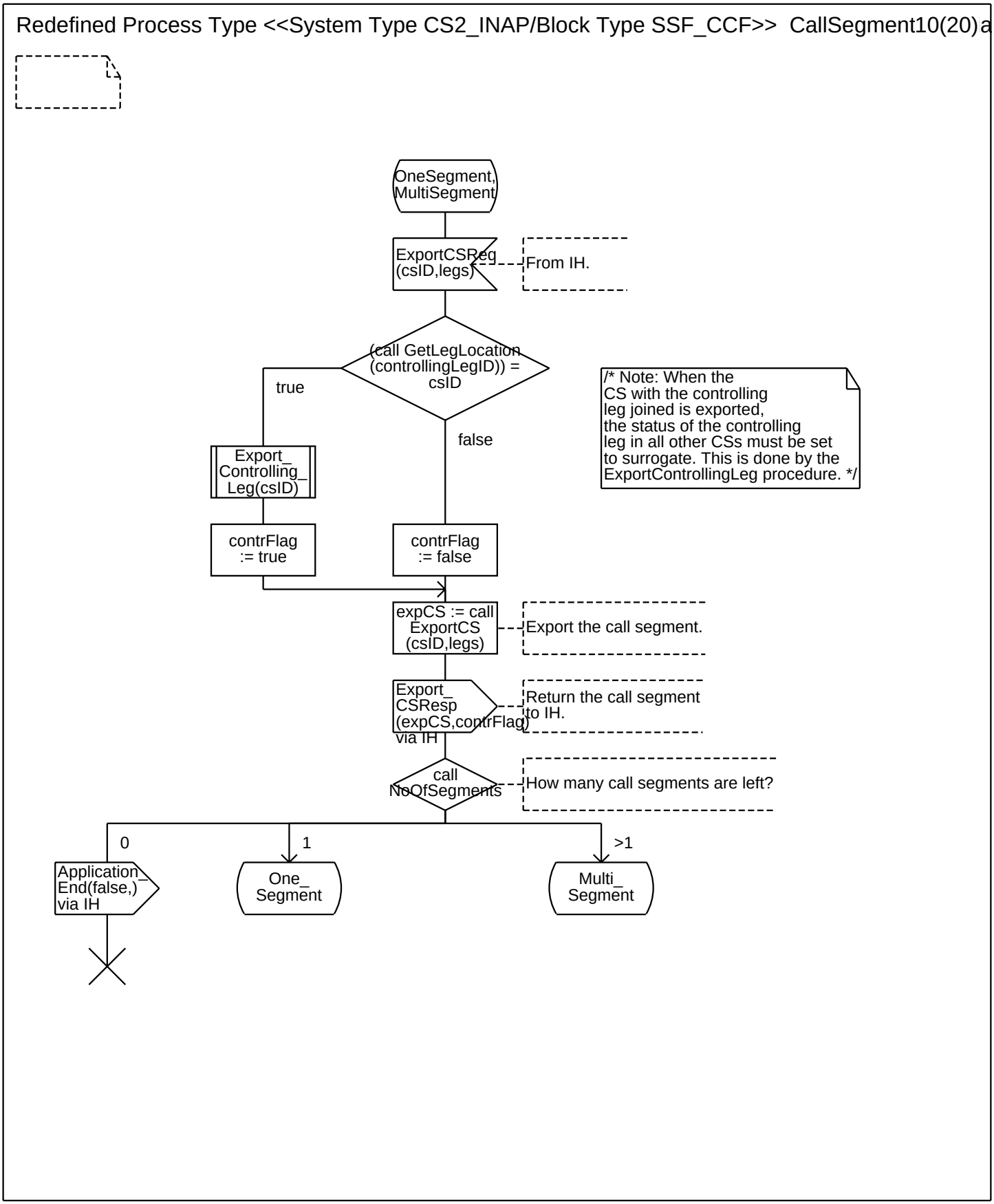
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA8(20)a
MultiSegment
MoveLeg (invokeID,,mlArg)
Does the leg exist?
call ExistLeg (mlArg! legIDToMove!sendingSideID)
No
Yes
Leg := call ExportLeg (mlArg!legIDToMove!sendingSideID)
Export the leg from the segment where it is currently residing.
Failed2
ImportLeg (mlArg!legID_ ToMove!sendingSideID,leg, mlArg!target_ CallSegment)
Import the leg into the target call segment.
MoveLegResult (invokeID) via IH
Multi_ Segment
OneSegment, MultiSegment
SplitLeg (invokeID,,slArg)
call ExistLeg (slArg! legToBeSplit!sendingSideID)
Does the leg to be split exist?
No
Yes
leg := call ExportLeg (slArg! legToBeSplit!sendingSideID)
Export the leg from the segment where it is currently residing.
call AddCS (slArg!newCallSegment, controllingLegID, waitingForInstruction, false)
Create the new call segment wit
− the CallSegmentID set to the SplitLeg
− the ID of the controlling leg set to ControllingLegID.
− the SSF−FSM in state WFI
− the call is not SCF initiated (False parameter)
Failed
Successful
Failed2
ImportLeg (slArg!legToBe_ Split!sendingSideID,leg,slArg! newCallSegment)
Import the leg into the newly created call segment.
SplitLeg_ Result(invokeID) via IH
Multi_ Segment

Annex A:

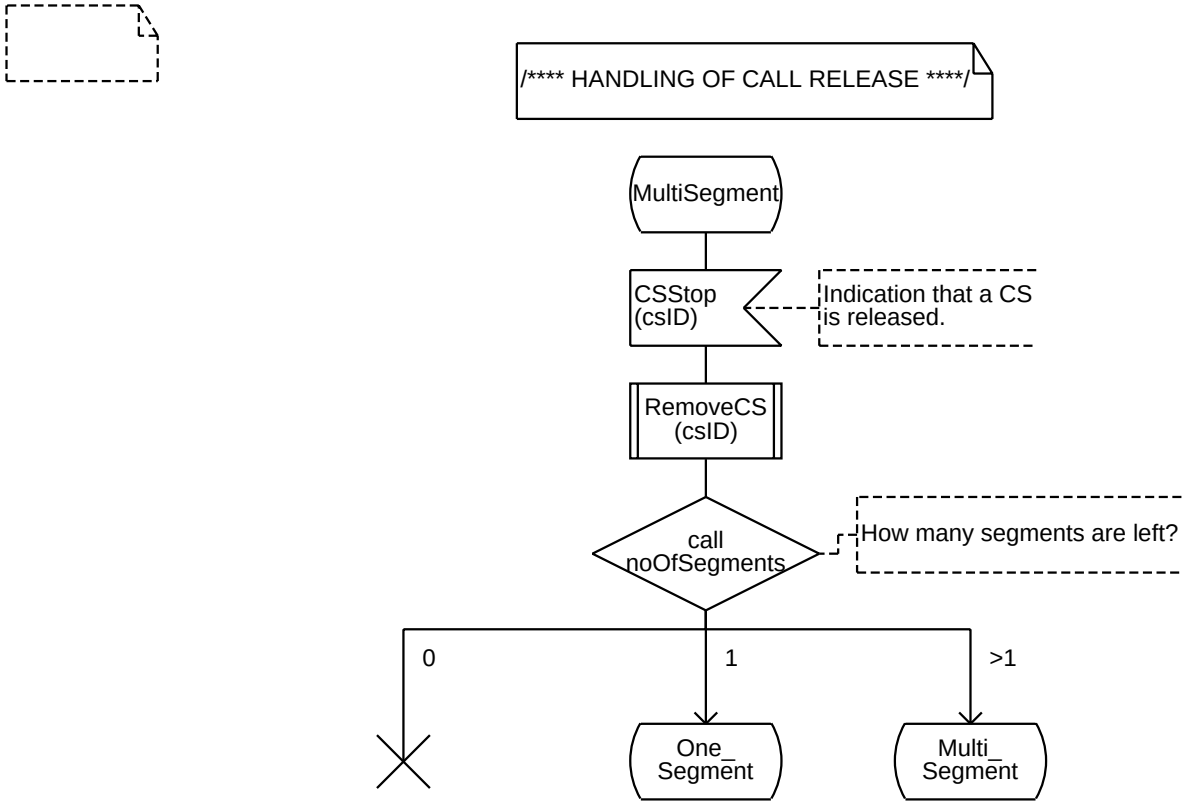

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentA9(20)a
/**** PROCESSING OF ATOMIC OPERATIONS USED BY THE IH. ****/
Null
ImportCSReq (csID, expCS,legs,contrFlag)
From IH when processing the IN CS−2 MoveCall_ Segments operation.
call ExistCS (csID)
Does the call segment exist?
True
Failed2
False
ImportCS (expCS,csID legs,contrFlag)
Import the call segment and assign it the id CallSegmentID.
ImportCSResp via IH
Respons back to IH that the import is done.
One_ Segment
OneSegment, MultiSegment
Import_ CSReq (csID, expCS,legs,contrFlag)
From IH when processin the IN CS−2 MoveCall_ Segments operation.
call ExistCS (csID)
Does the call segment exist?
True
Failed2
False
ImportCS (expCS,csID legs,contrFlag)
Import the call segment and assign it the id CallSegmentID.
Import_ CSResp via IH
Respons back to IH that the import is done.
Multi_ Segment

Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment10(20)a
OneSegment, MultiSegment
ExportCSReq (csID,legs)
From IH.
(call GetLegLocation (controllingLegID)) = csID
true
false
/* Note: When the CS with the controlling leg joined is exported, the status of the controlling leg in all other CSs must be set to surrogate. This is done by the ExportControllingLeg procedure. */
Export_ Controlling_ Leg(csID)
contrFlag := true
contrFlag := false
expCS := call ExportCS (csID,legs)
Export the call segment.
Export_ CSResp (expCS,contrFlag) via IH
Return the call segment to IH.
call NoOfSegments
How many call segments are left?
0
1
>1
Application_ End(false,) via IH
One_ Segment
Multi_ Segment

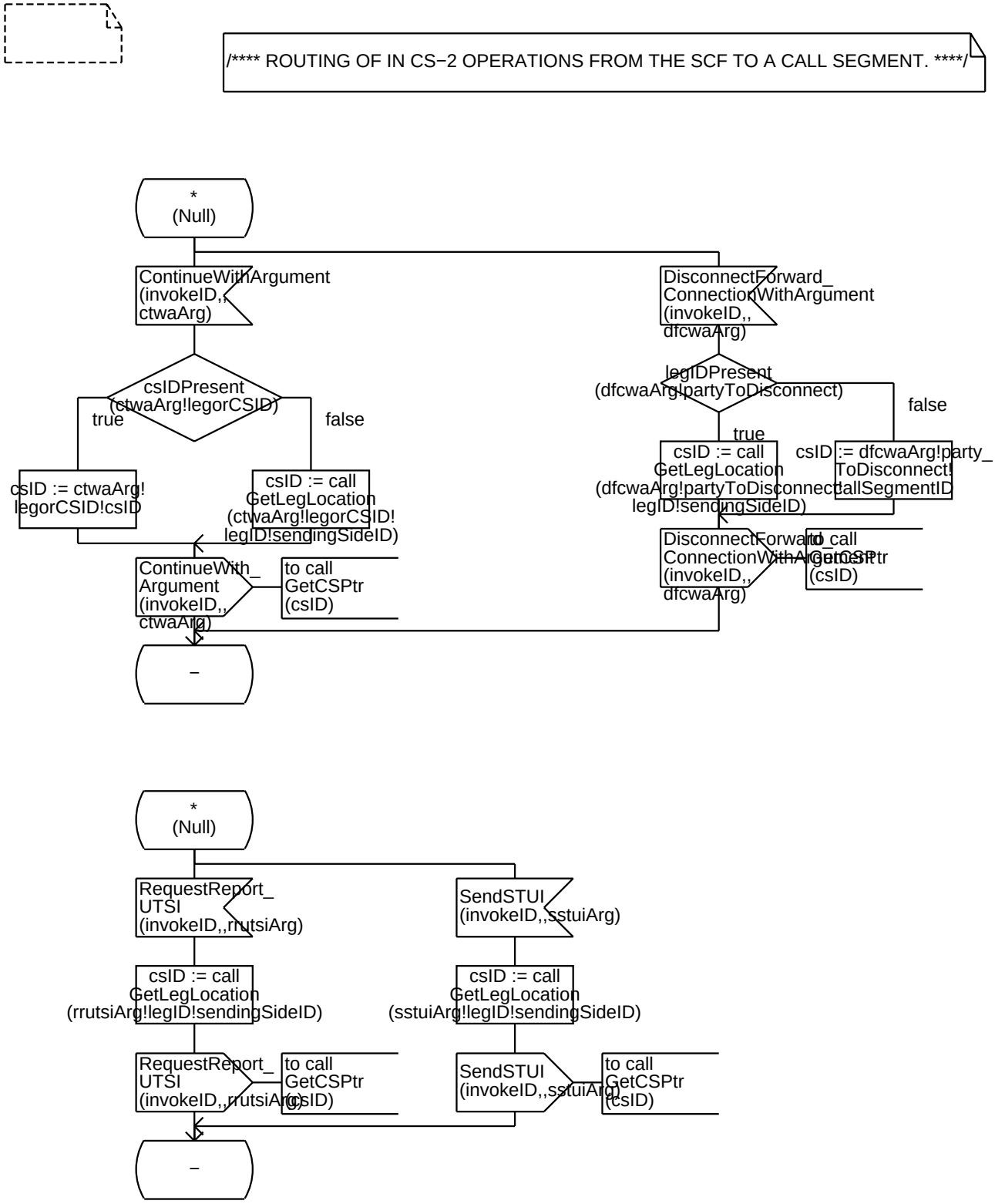
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment11(20)a



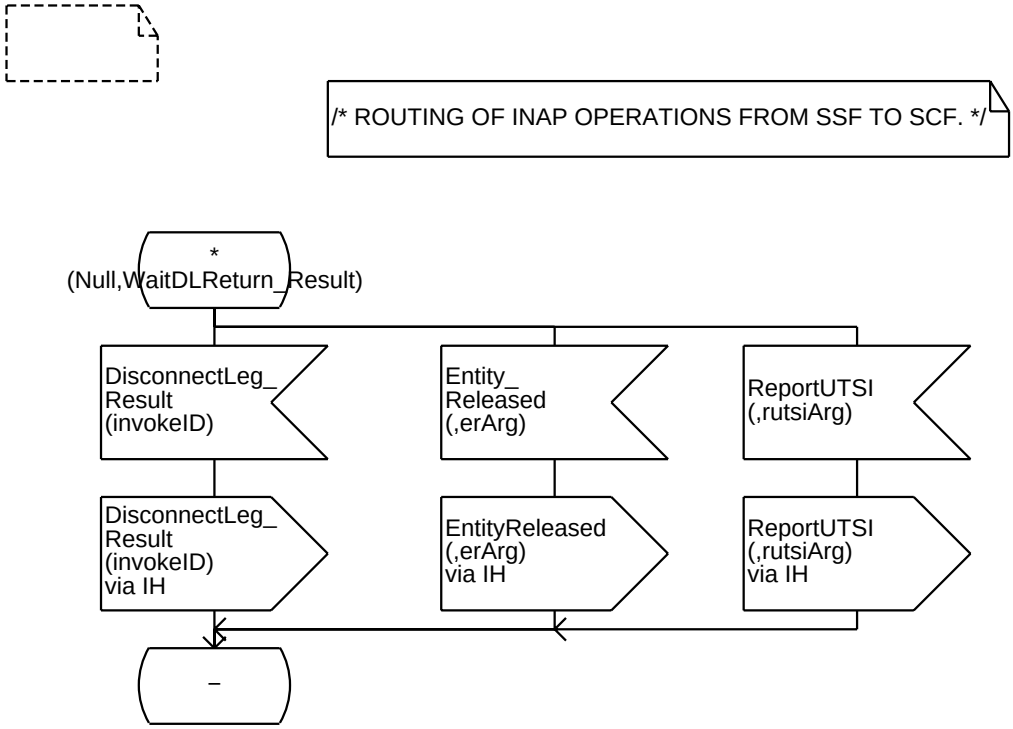
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment12(20)a



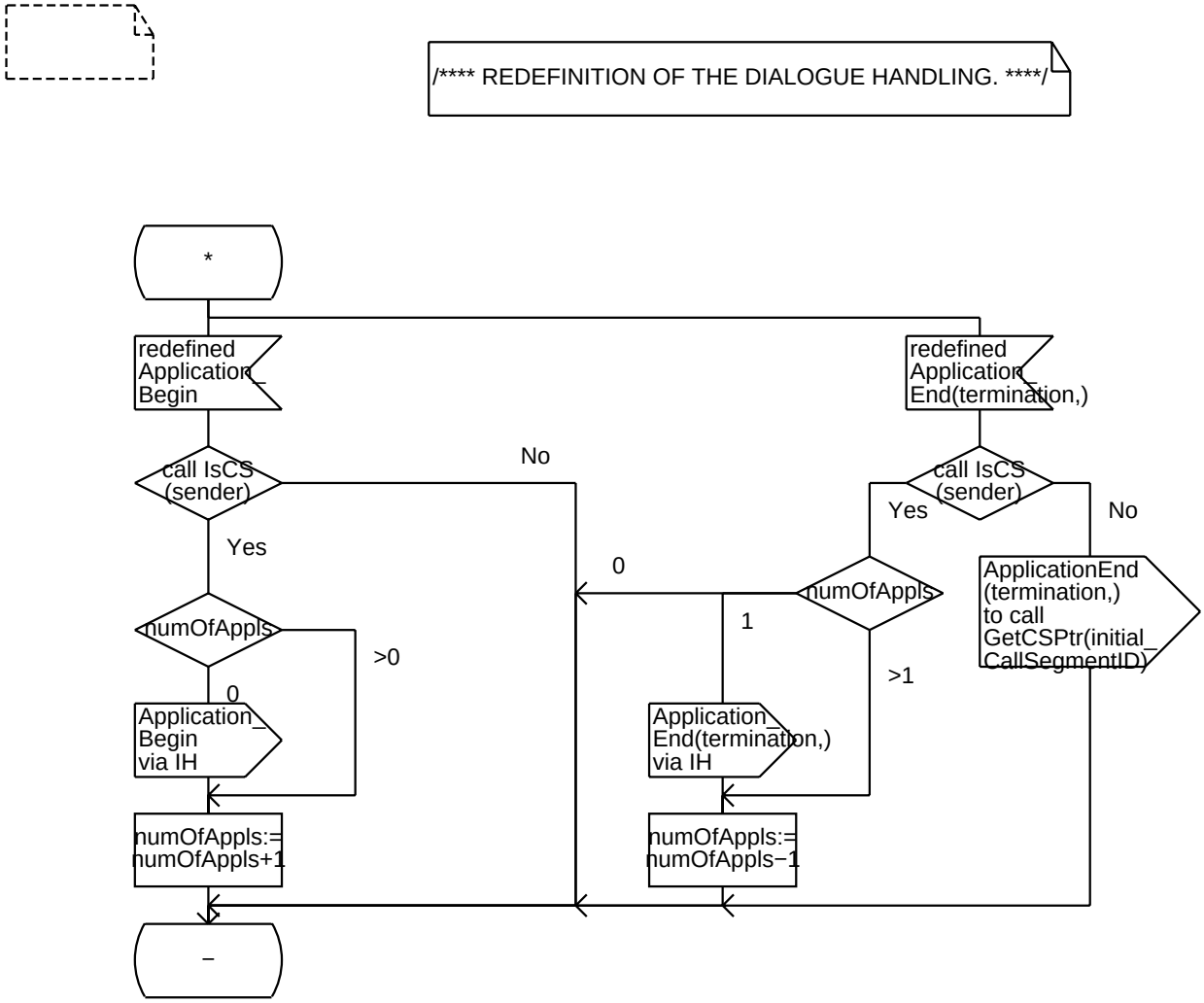
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment13(20)a



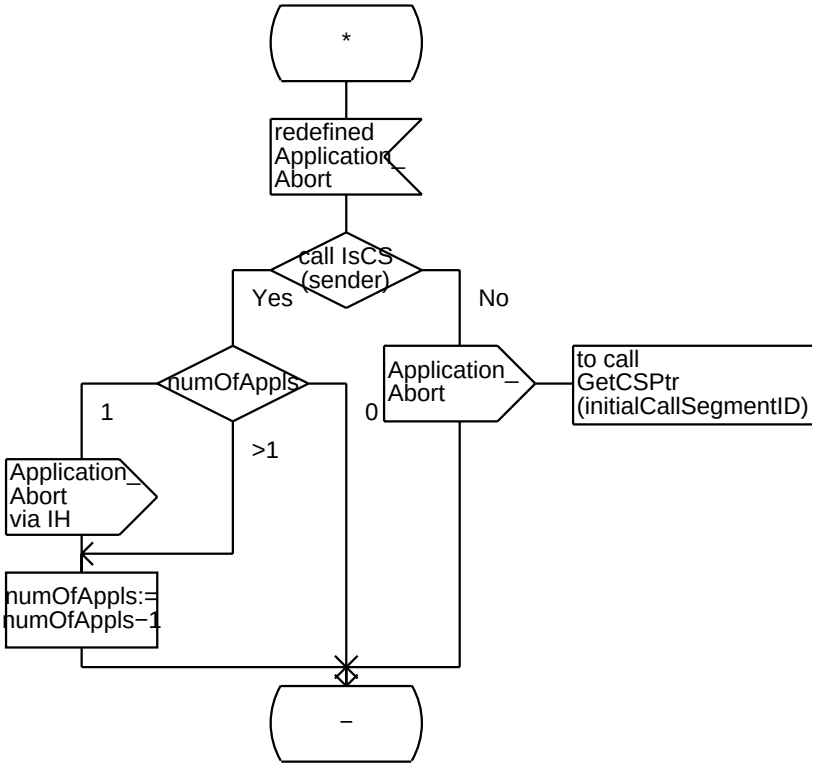
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment14(20)a

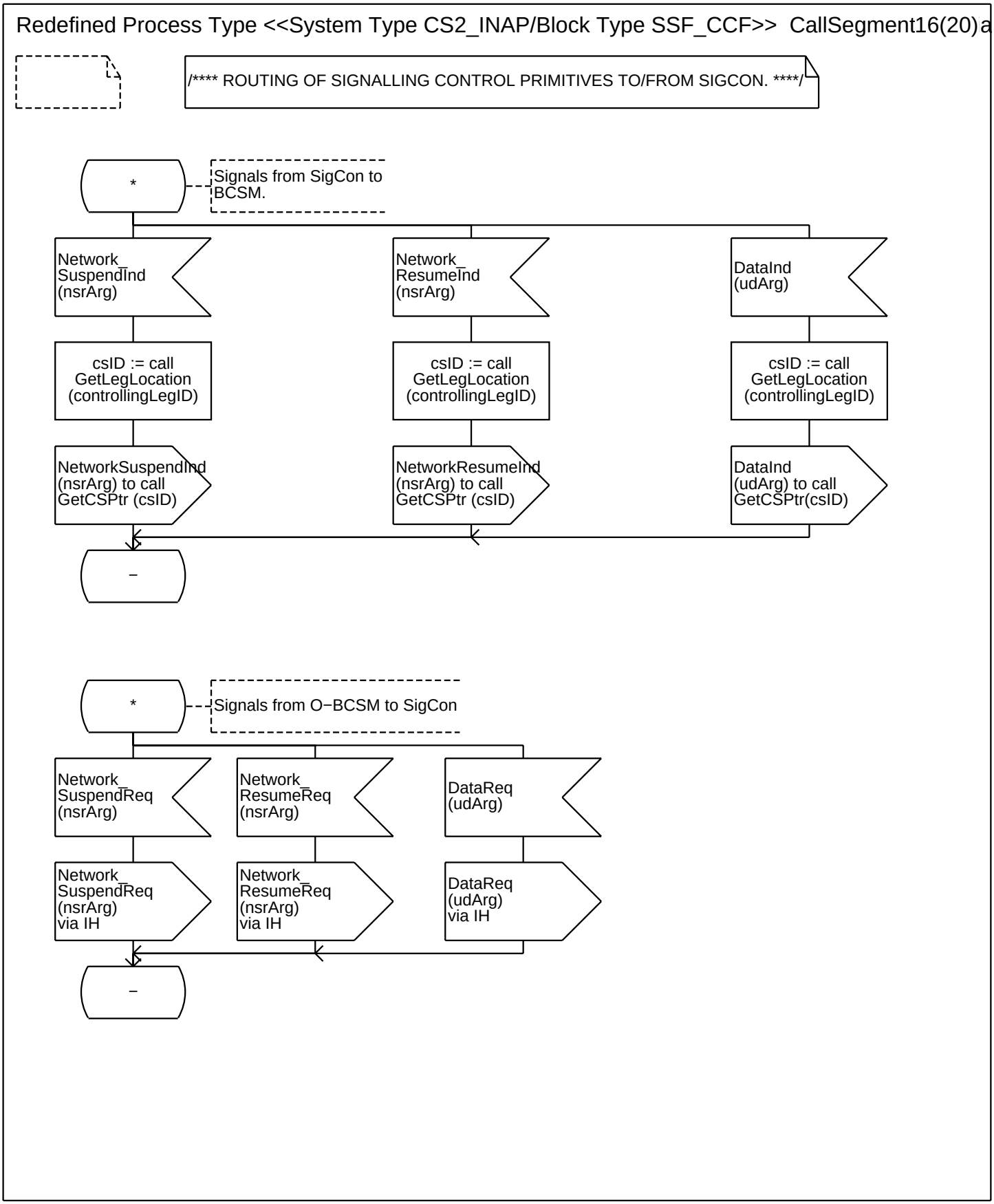


Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment15(20)a

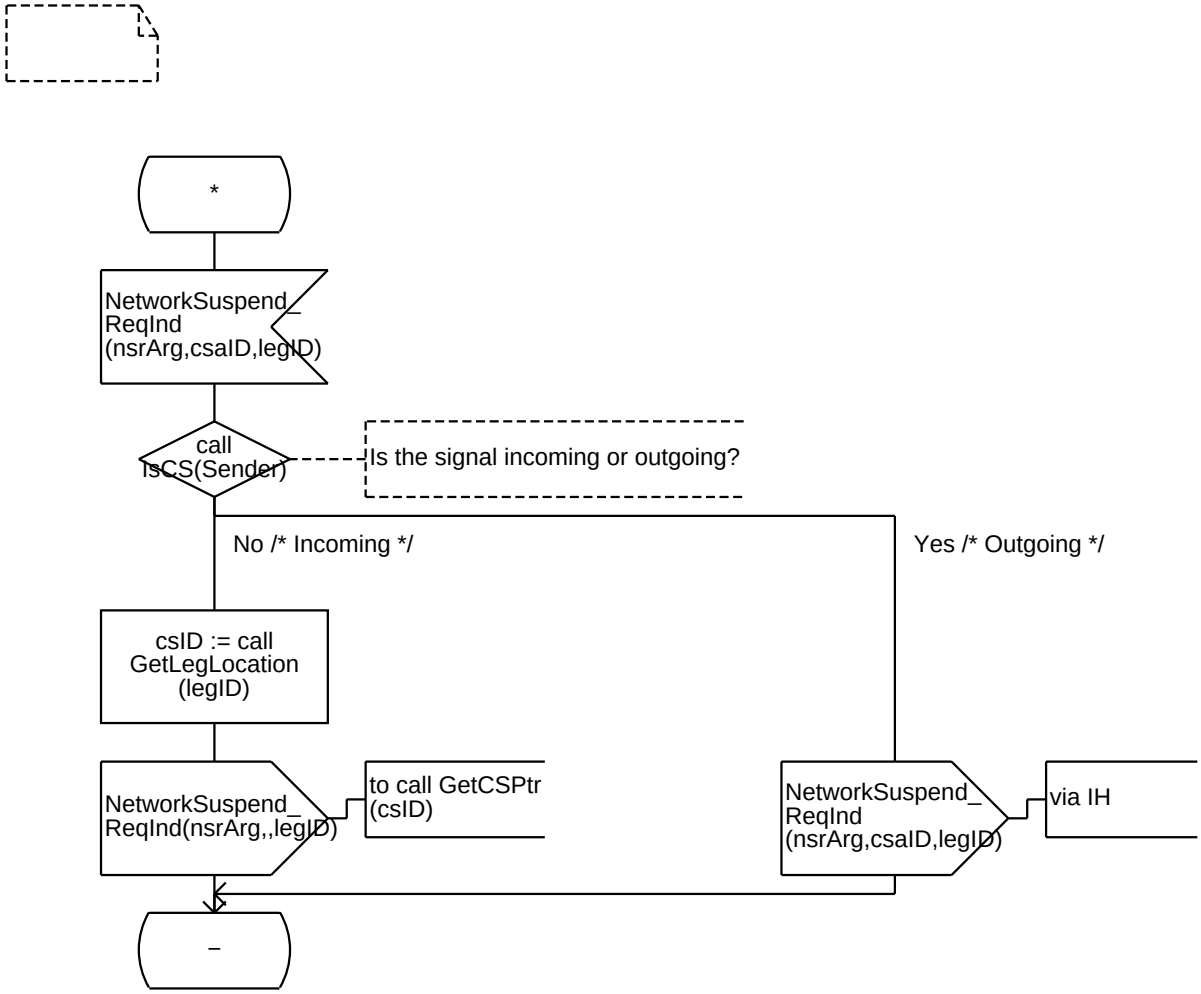


Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment16(20)a
/**** ROUTING OF SIGNALLING CONTROL PRIMITIVES TO/FROM SIGCON. ****/
*
Signals from SigCon to BCSM.
Network_ SuspendInd (nsrArg)
Network_ ResumeInd (nsrArg)
DataInd (udArg)
csID := call GetLegLocation (controllingLegID)
csID := call GetLegLocation (controllingLegID)
csID := call GetLegLocation (controllingLegID)
NetworkSuspendInd (nsrArg) to call GetCSPtr (csID)
NetworkResumeInd (nsrArg) to call GetCSPtr (csID)
DataInd (udArg) to call GetCSPtr(csID)
−
*
Signals from O−BCSM to SigCon
Network_ SuspendReq (nsrArg)
Network_ ResumeReq (nsrArg)
DataReq (udArg)
Network_ SuspendReq (nsrArg) via IH
Network_ ResumeReq (nsrArg) via IH
DataReq (udArg) via IH
−

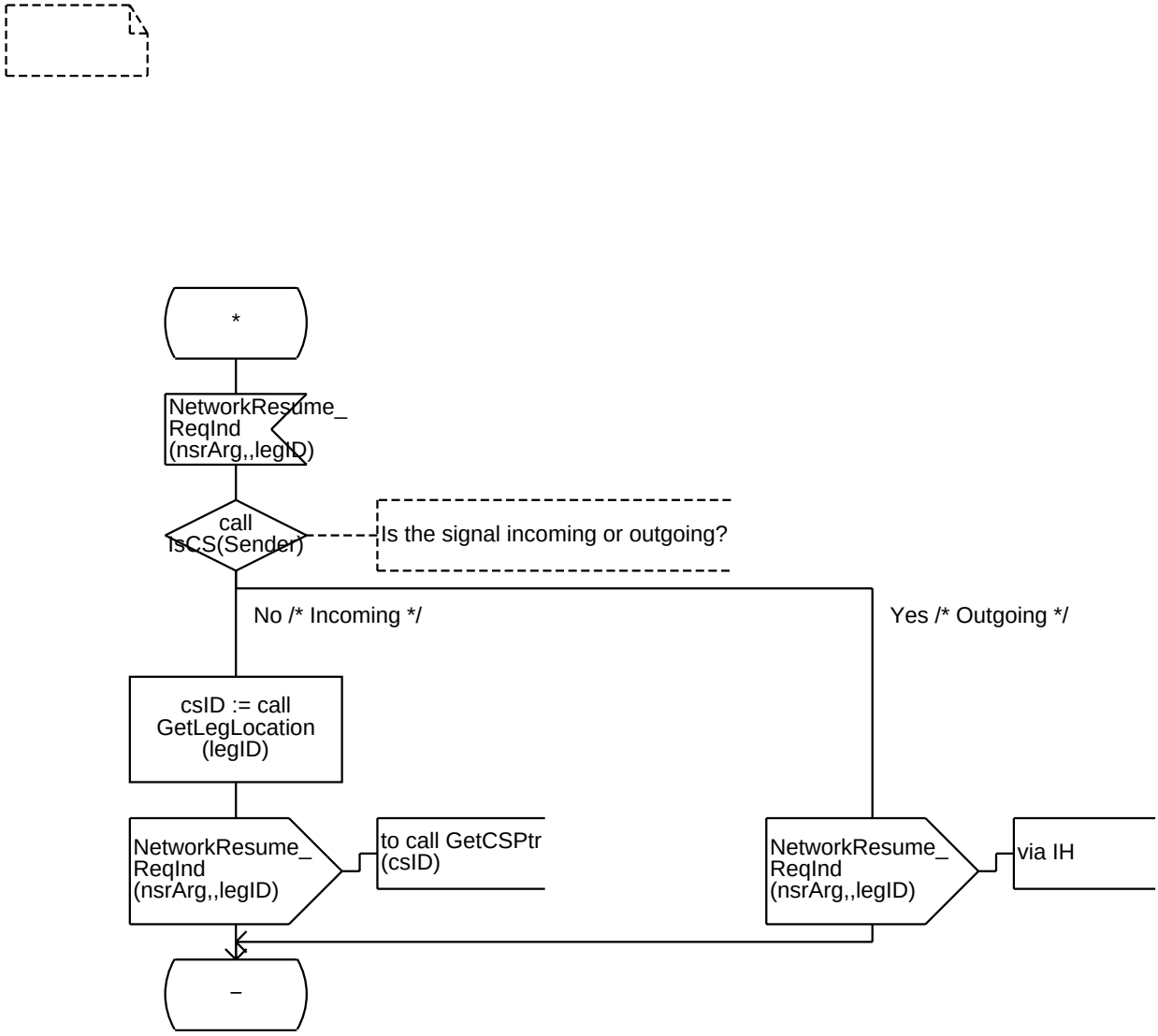
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment17(20)a

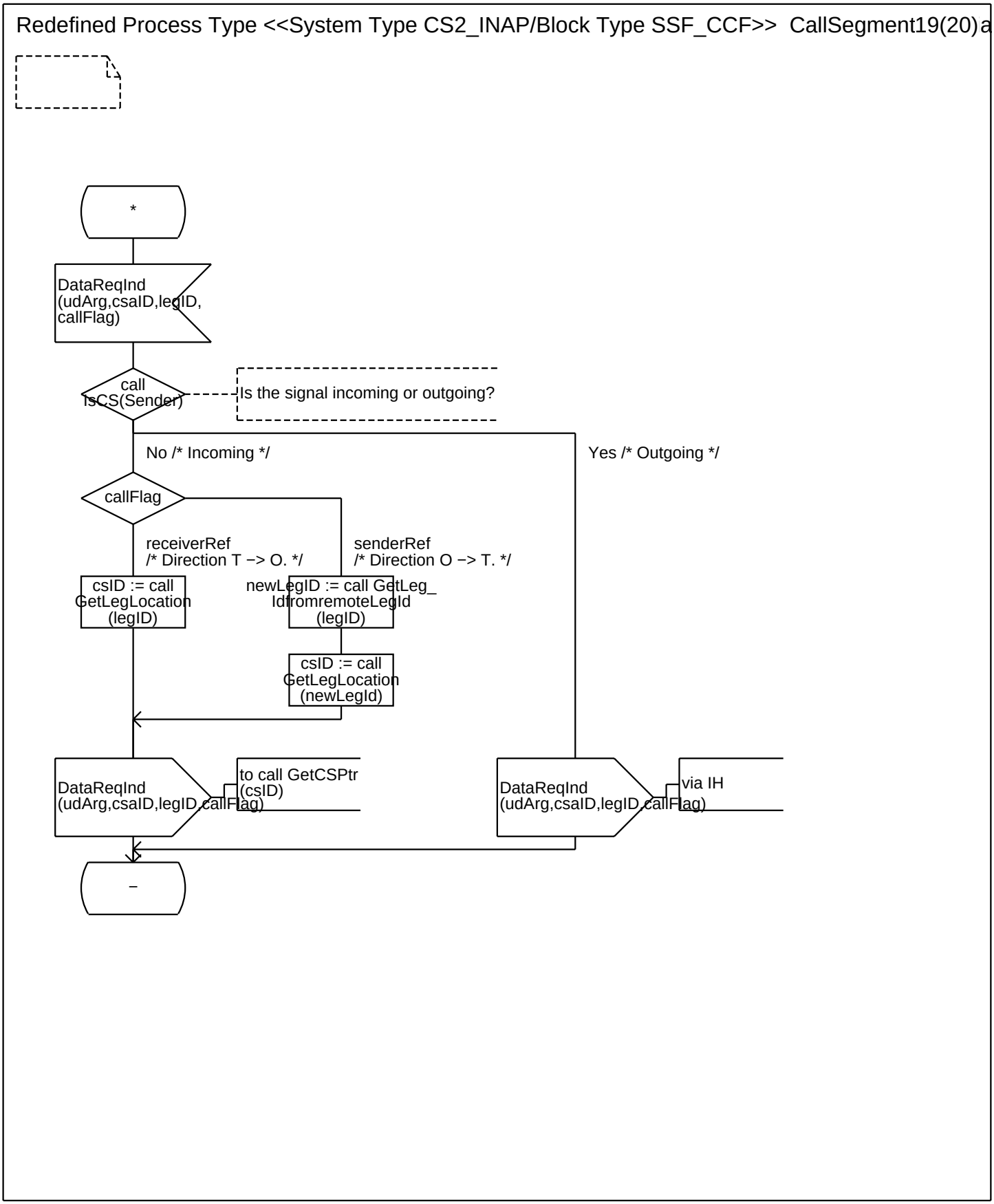


Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment18(20)a

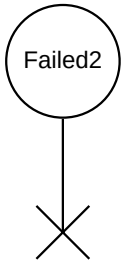


Annex A:

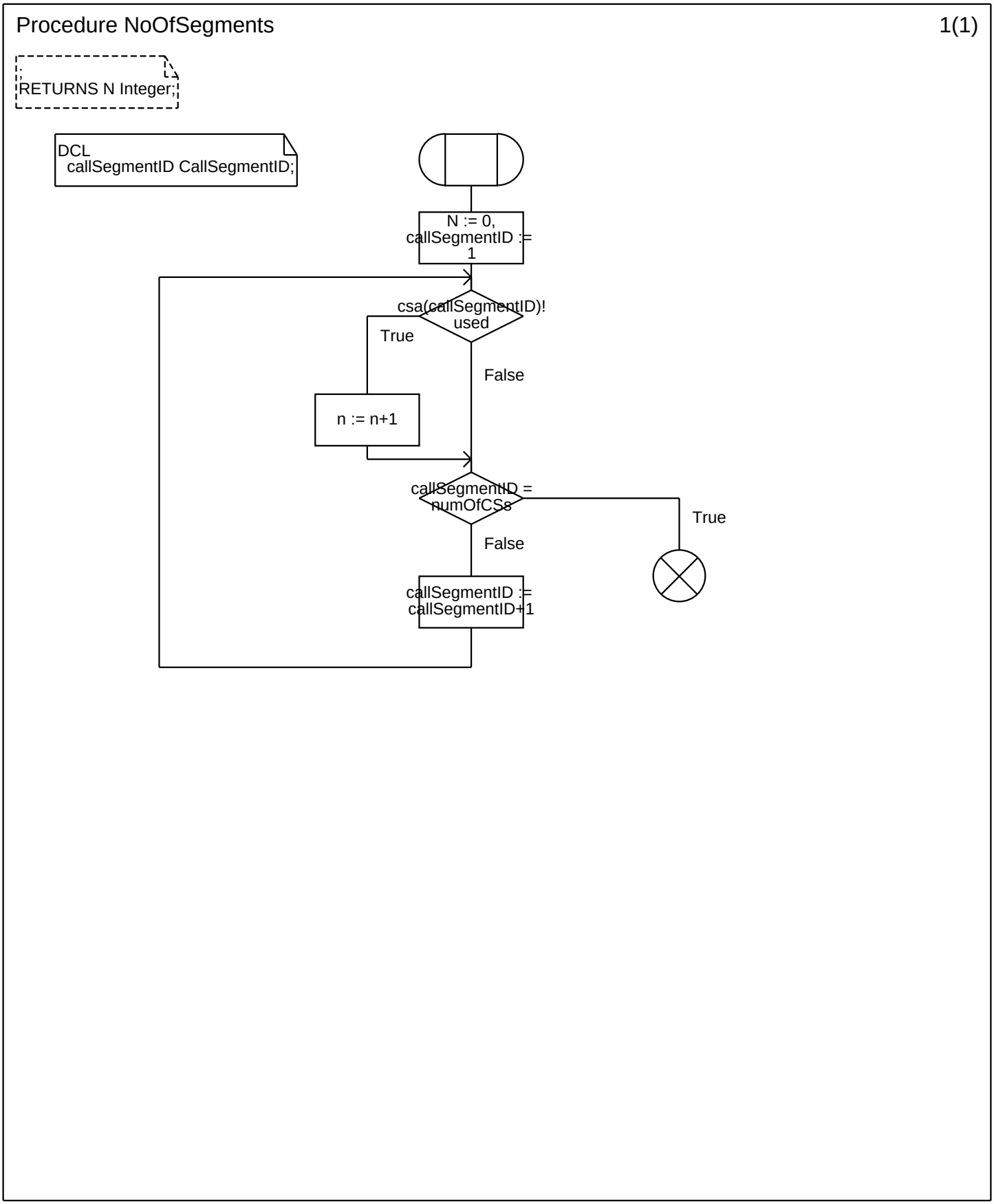

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment19(20)a
*
DataReqInd (udArg,csaID,legID, callFlag)
call IsCS(Sender)
Is the signal incoming or outgoing?
No /* Incoming */
Yes /* Outgoing */
callFlag
receiverRef /* Direction T −> O. */
senderRef /* Direction O −> T. */
csID := call GetLegLocation (legID)
newLegID := call GetLeg_ IdfromremoteLegId (legID)
csID := call GetLegLocation (newLegId)
DataReqInd (udArg,csaID,legID,callFlag)
to call GetCSPtr (csID)
DataReqInd (udArg,csaID,legID,callFlag)
via IH
−

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment20(20)a

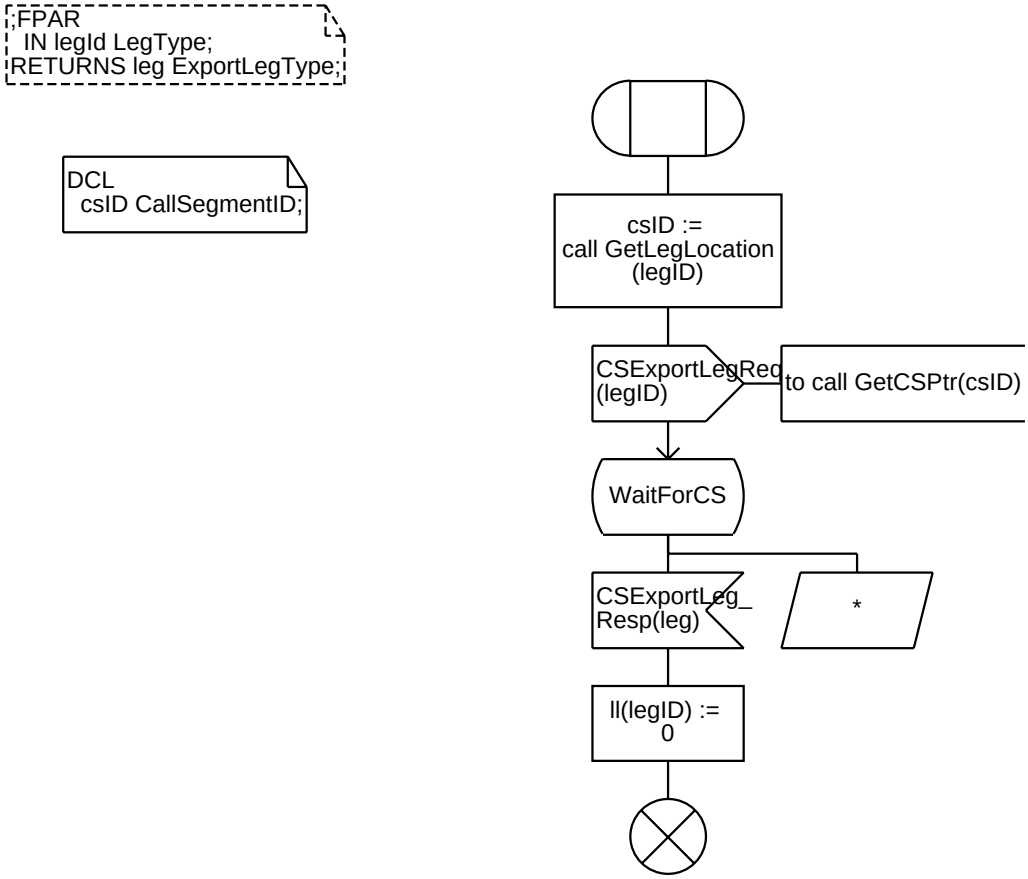


Annex A:

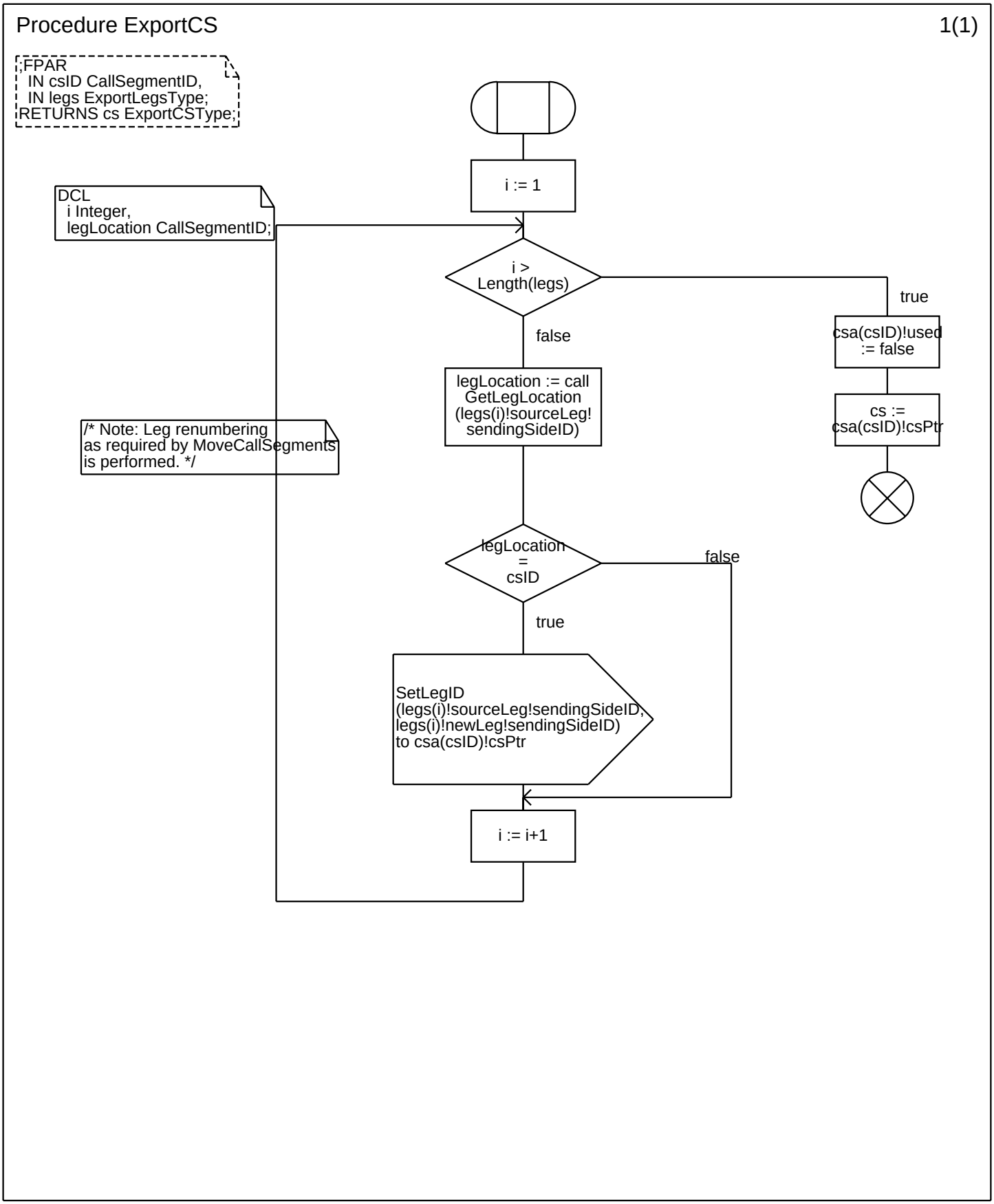

Procedure NoOfSegments
1(1)
;
RETURNS N Integer;
DCL
callSegmentID CallSegmentID;
N := 0,
callSegmentID :=
1
csa(callSegmentID)!
used
True
False
n := n+1
callSegmentID =
numOfCSs
True
False
callSegmentID :=
callSegmentID+1

Annex A:

Procedure <<Process Type CallSegmentAssociation>> ExportLeg 1(1)

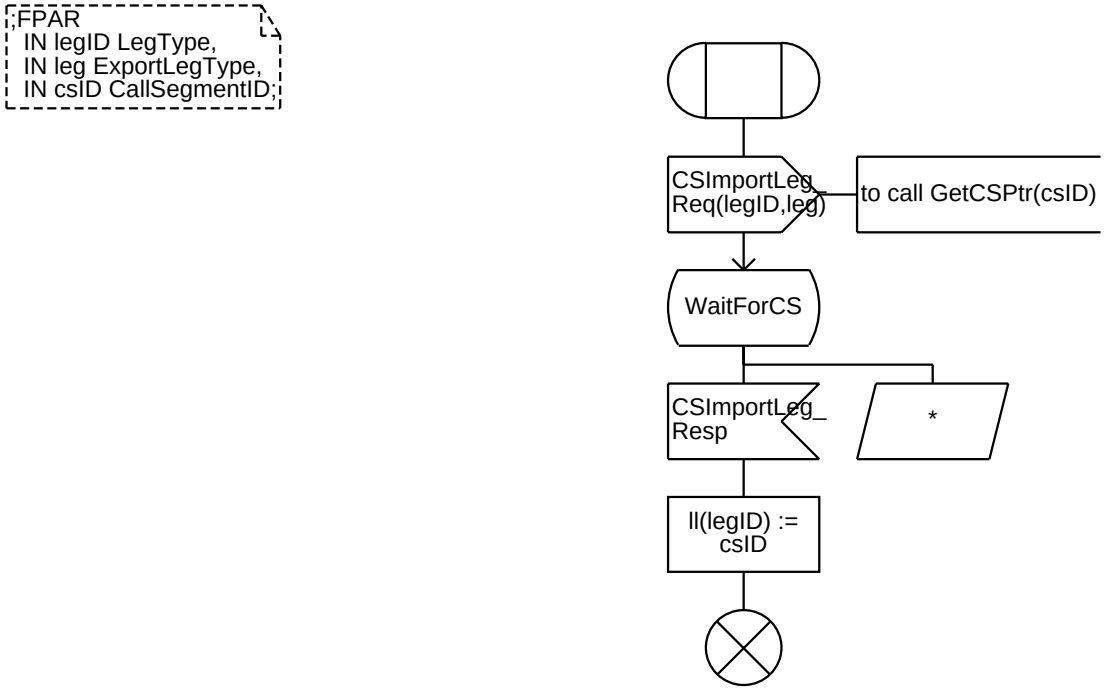


Annex A:

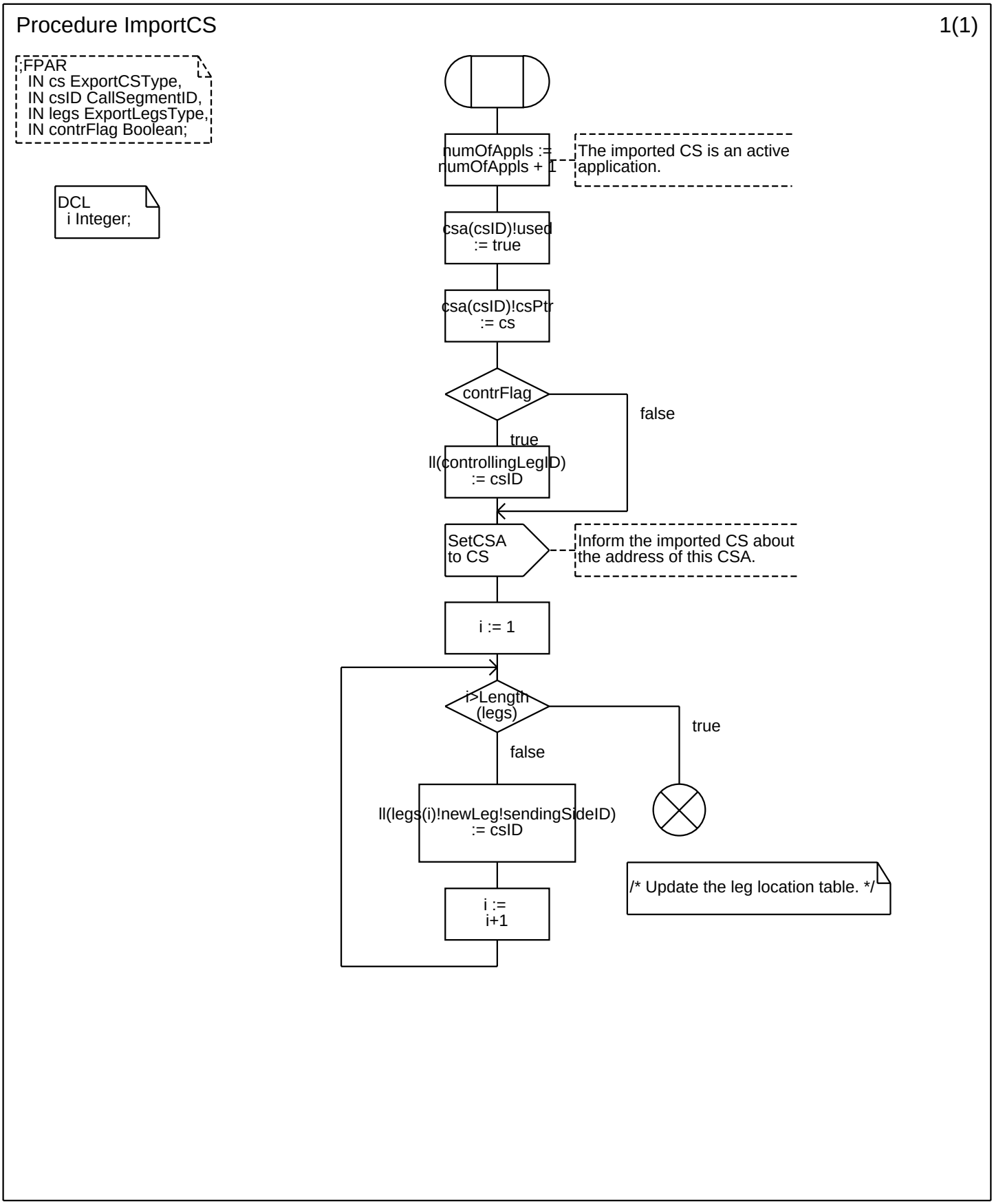

Procedure ExportCS
1(1)
;FPAR
IN csID CallSegmentID,
IN legs ExportLegsType;
RETURNS cs ExportCSType;
DCL
i Integer,
legLocation CallSegmentID;
/* Note: Leg renumbering
as required by MoveCallSegments
is performed. */
i := 1
i >
Length(legs)
true
false
csa(csID)!used
:= false
cs :=
csa(csID)!csPtr
legLocation := call
GetLegLocation
(legs(i)!sourceLeg!
sendingSideID)
legLocation
=
csID
false
true
SetLegID
(legs(i)!sourceLeg!sendingSideID,
legs(i)!newLeg!sendingSideID)
to csa(csID)!csPtr
i := i+1

Annex A:

Procedure <<Process Type CallSegmentAssociation>> ImportLeg 1(1)

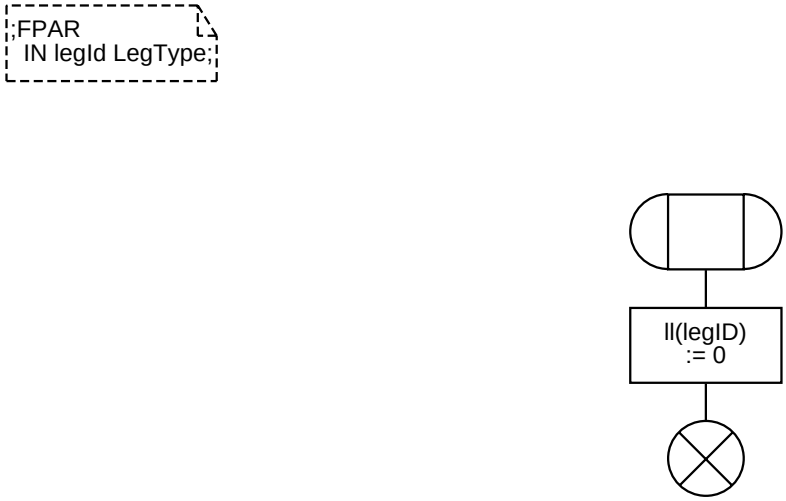


Annex A:


Procedure ImportCS
1(1)
;FPAR
IN cs ExportCSType,
IN csID CallSegmentID,
IN legs ExportLegsType,
IN contrFlag Boolean;
DCL
i Integer;
numOfAppls := numOfAppls + 1
The imported CS is an active application.
csa(csID)!used := true
csa(csID)!csPtr := cs
contrFlag
false
true
ll(controllingLegID) := csID
SetCSA to CS
Inform the imported CS about the address of this CSA.
i := 1
i>Length (legs)
true
false
ll(legs(i)!newLeg!sendingSideID) := csID
/* Update the leg location table. */
i := i+1

Annex A:

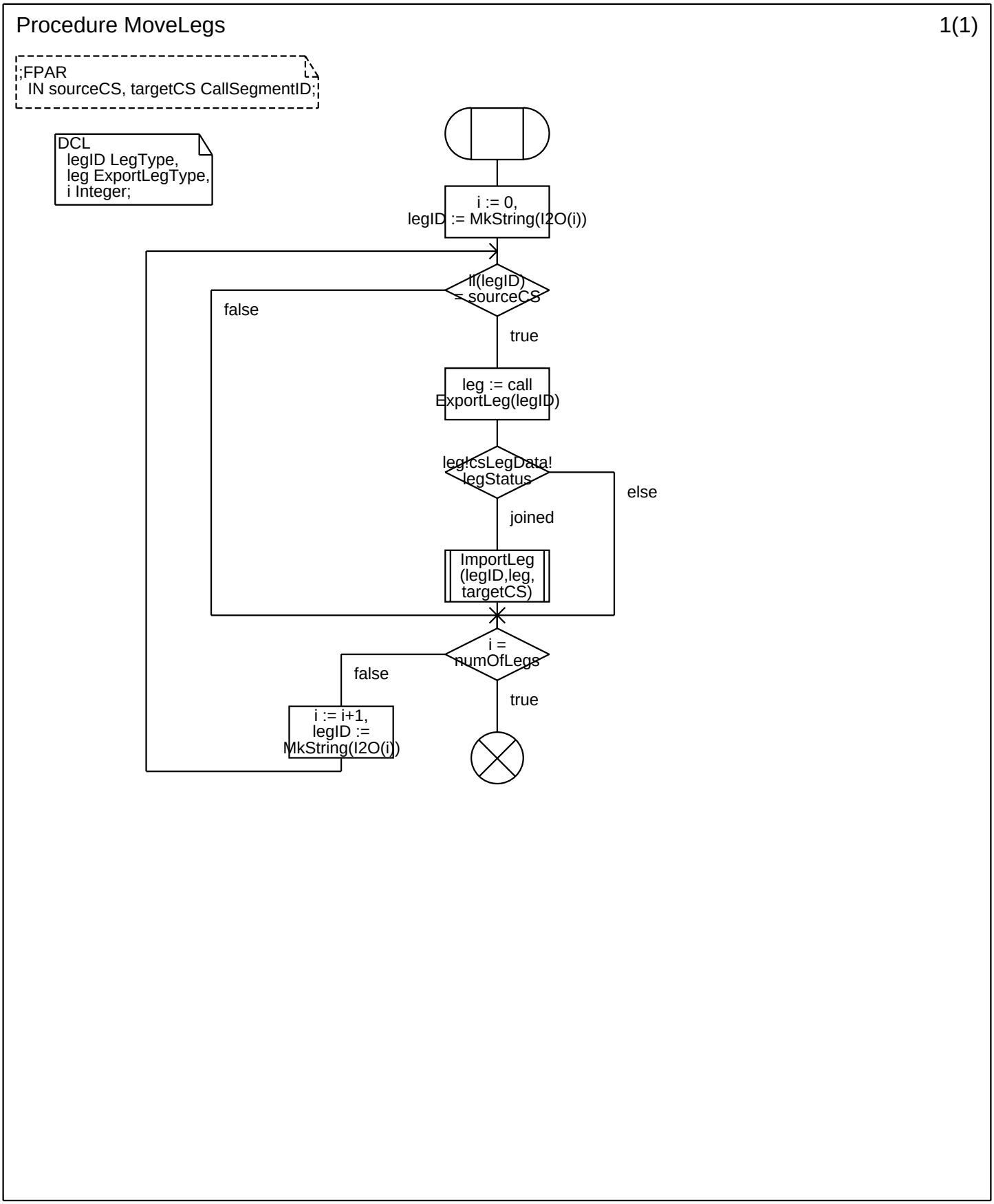
Procedure <<Process Type CallSegmentAssociation>> RemoveLeg 1(1)



Annex A:

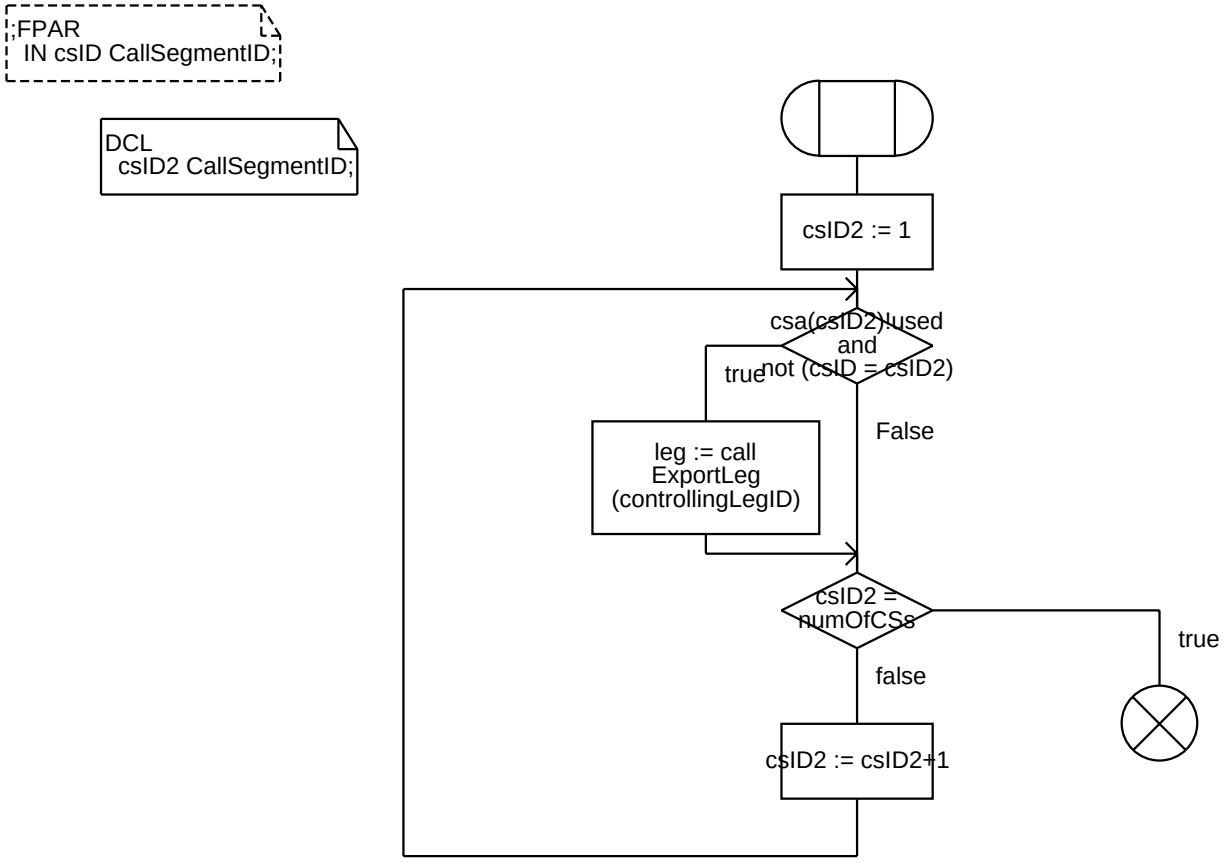
Procedure RemoveCS 1(1)

Annex A:

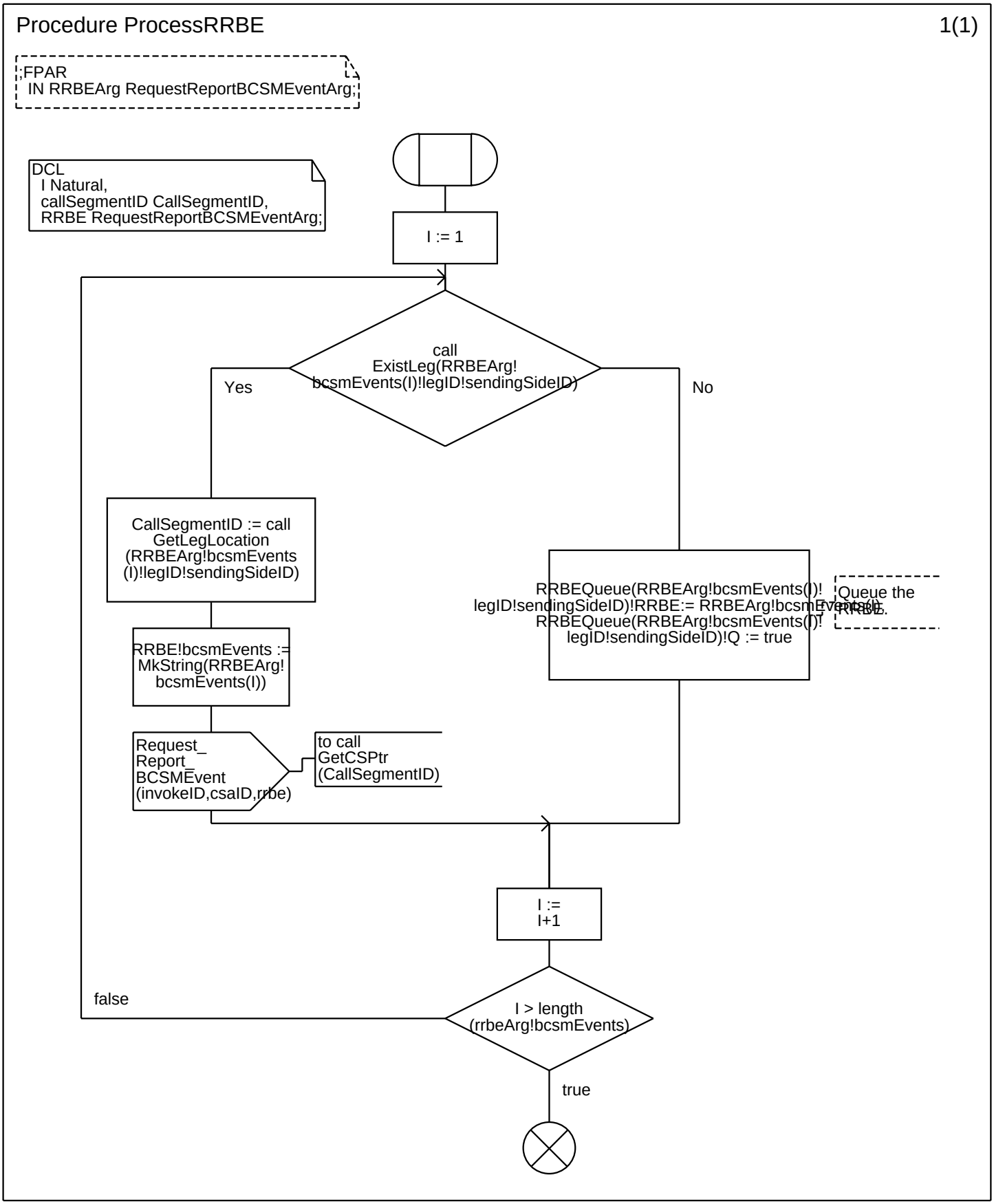

Procedure MoveLegs
1(1)
;FPAR
IN sourceCS, targetCS CallSegmentID;
DCL
legID LegType,
leg ExportLegType,
i Integer;
i := 0,
legID := MkString(I2O(i))
l(legID)
= sourceCS
false
true
leg := call
ExportLeg(legID)
leg!csLegData!
legStatus
else
joined
ImportLeg
(legID,leg,
targetCS)
i =
numOfLegs
false
true
i := i+1,
legID :=
MkString(I2O(i))

Annex A:

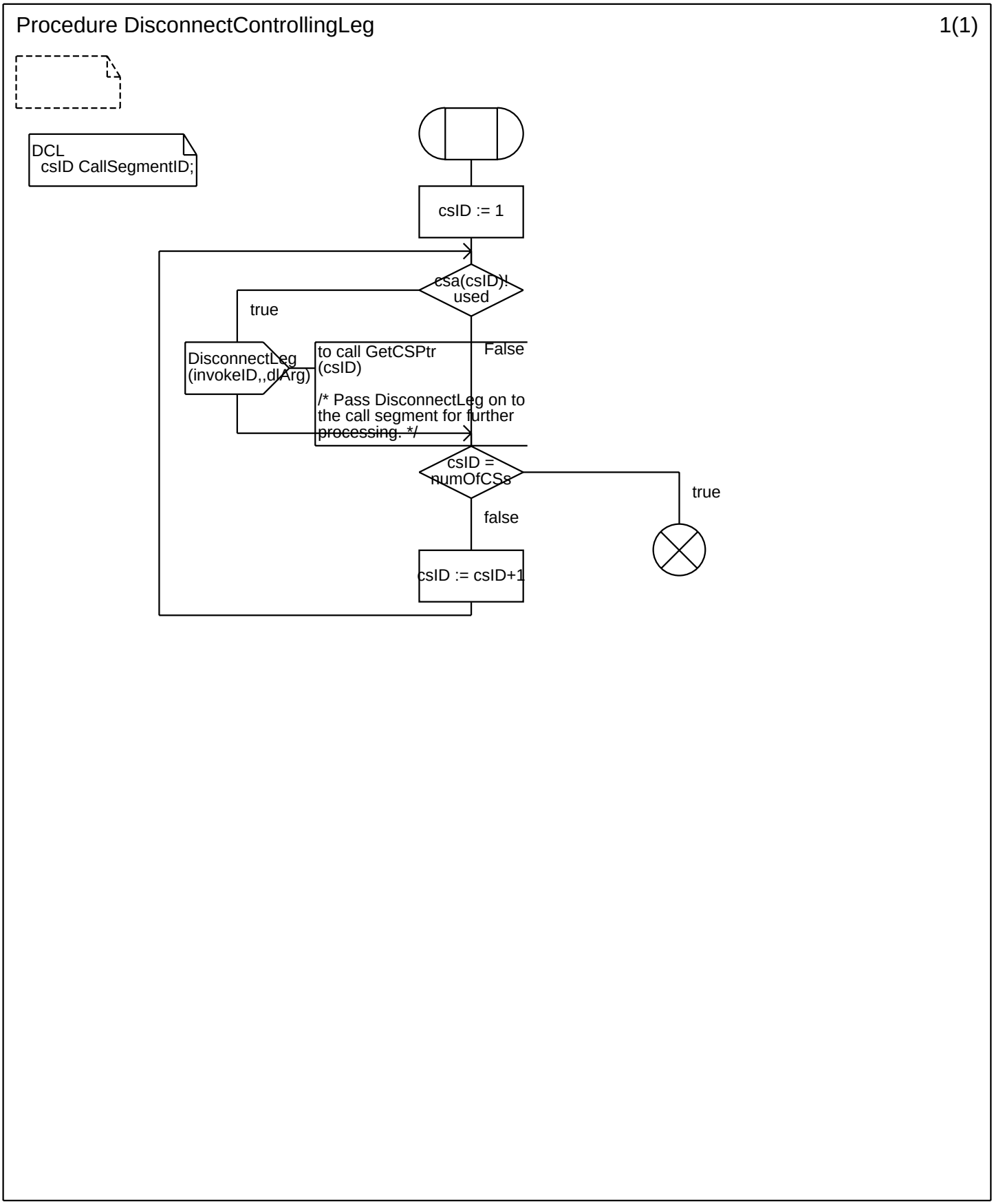
Procedure ExportControllingLeg 1(1)



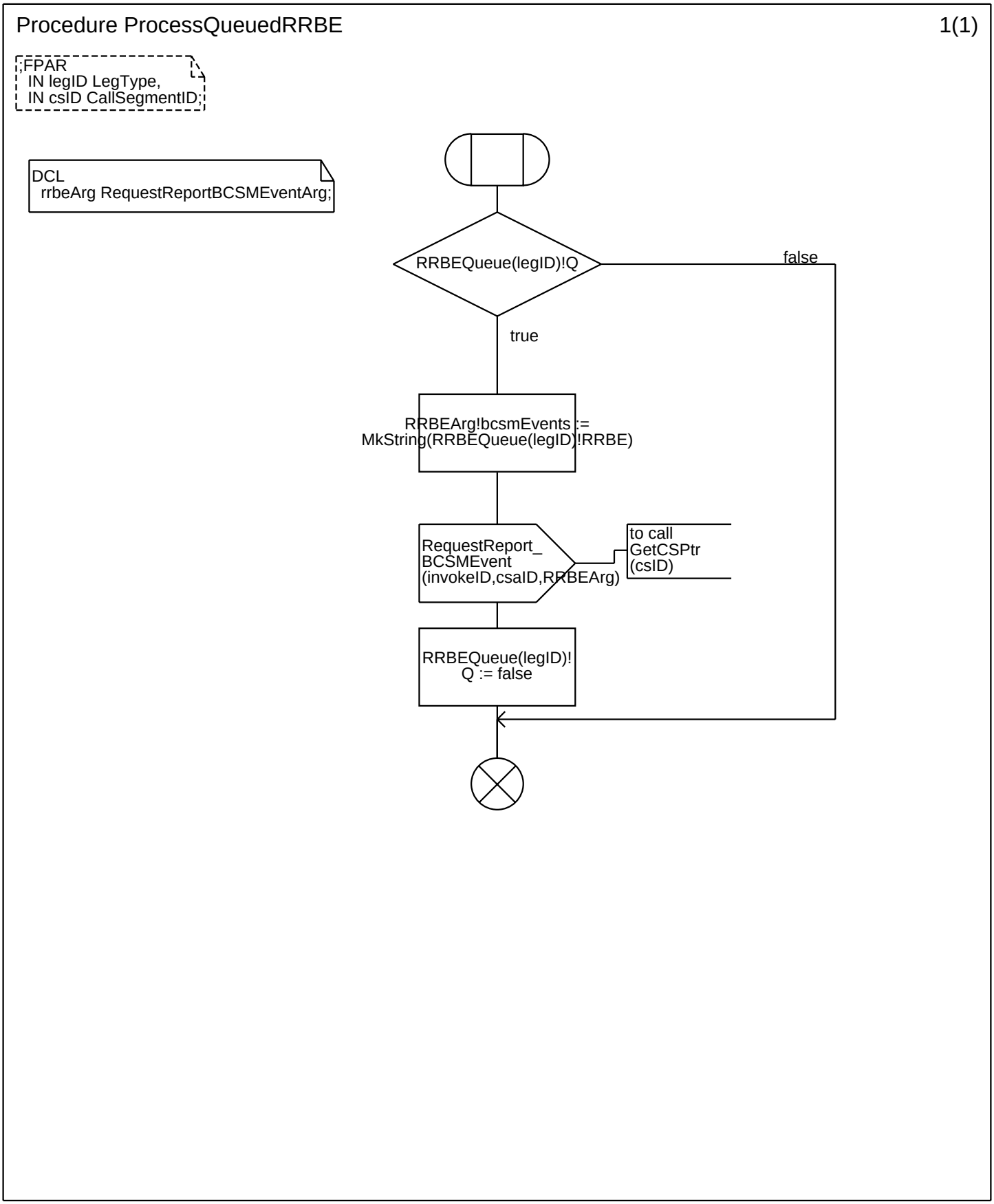
Annex A:


Procedure ProcessRRBE
1(1)
;FPAR
IN RRBEArg RequestReportBCSMEventArg;
DCL
I Natural,
callSegmentID CallSegmentID,
RRBE RequestReportBCSMEventArg;
I := 1
call
ExistLeg(RRBEArg!
bcsmEvents(I)!legID!sendingSideID)
Yes
No
CallSegmentID := call
GetLegLocation
(RRBEArg!bcsmEvents
(I)!legID!sendingSideID)
RRBEQueue(RRBEArg!bcsmEvents(I)!
legID!sendingSideID)!RRBE:= RRBEArg!bcsmEvents(I)
RRBEQueue(RRBEArg!bcsmEvents(I)!
legID!sendingSideID)!Q := true
Queue the
RRBE.
RRBE!bcsmEvents :=
MkString(RRBEArg!
bcsmEvents(I))
Request_
Report_
BCSMEvent
(invokeID,csaID,rrbe)
to call
GetCSPtr
(CallSegmentID)
I :=
I+1
false
I > length
(rrbeArg!bcsmEvents)
true

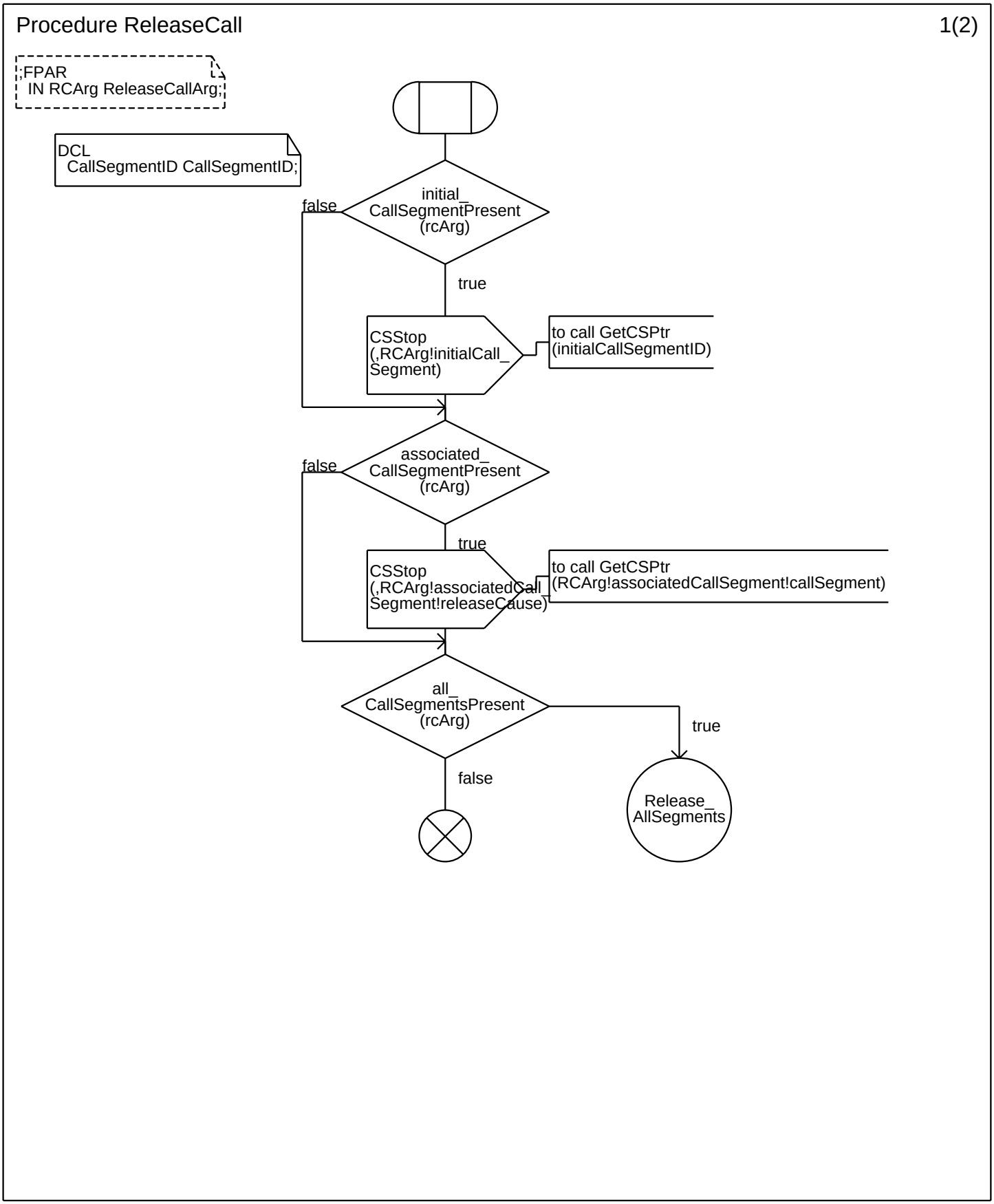
Annex A:


Procedure DisconnectControllingLeg
1(1)
DCL
csID CallSegmentID;
csID := 1
csa(csID)!
used
true
DisconnectLeg
(invokeID,,dlArg)
to call GetCSPtr
(csID)
/* Pass DisconnectLeg on to
the call segment for further
processing. */
False
csID =
numOfCSs
true
false
csID := csID+1

Annex A:

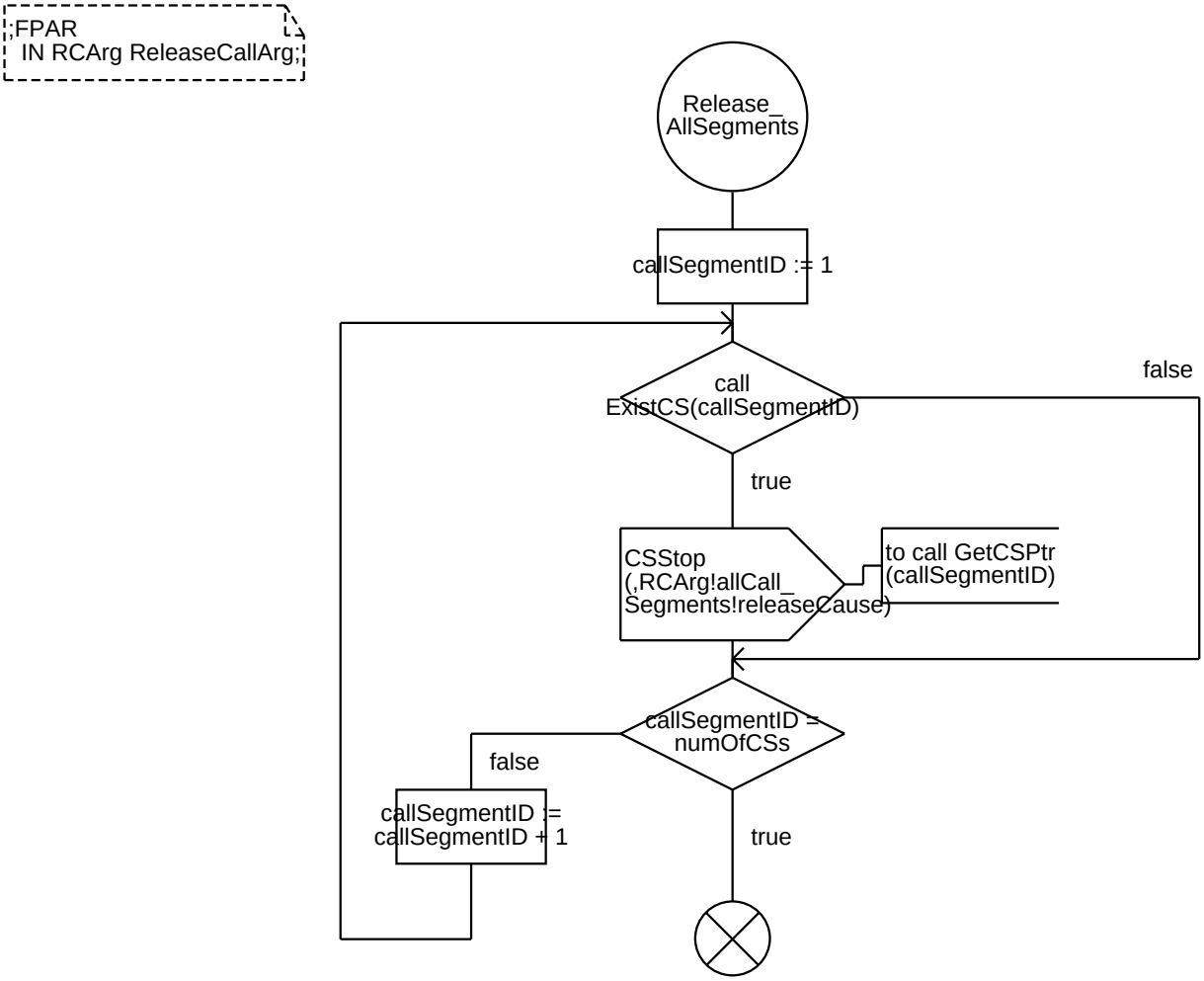

Procedure ProcessQueuedRRBE
1(1)
;FPAR
IN legID LegType,
IN csID CallSegmentID;
DCL
rrbeArg RequestReportBCSMEventArg;
RRBEQueue(legID)!Q
false
true
RRBEArg!bcsmEvents :=
MkString(RRBEQueue(legID)!RRBE)
RequestReport_
BCSMEvent
(invokeID,csaID,RRBEArg)
to call
GetCSPtr
(csID)
RRBEQueue(legID)!
Q := false

Annex A:

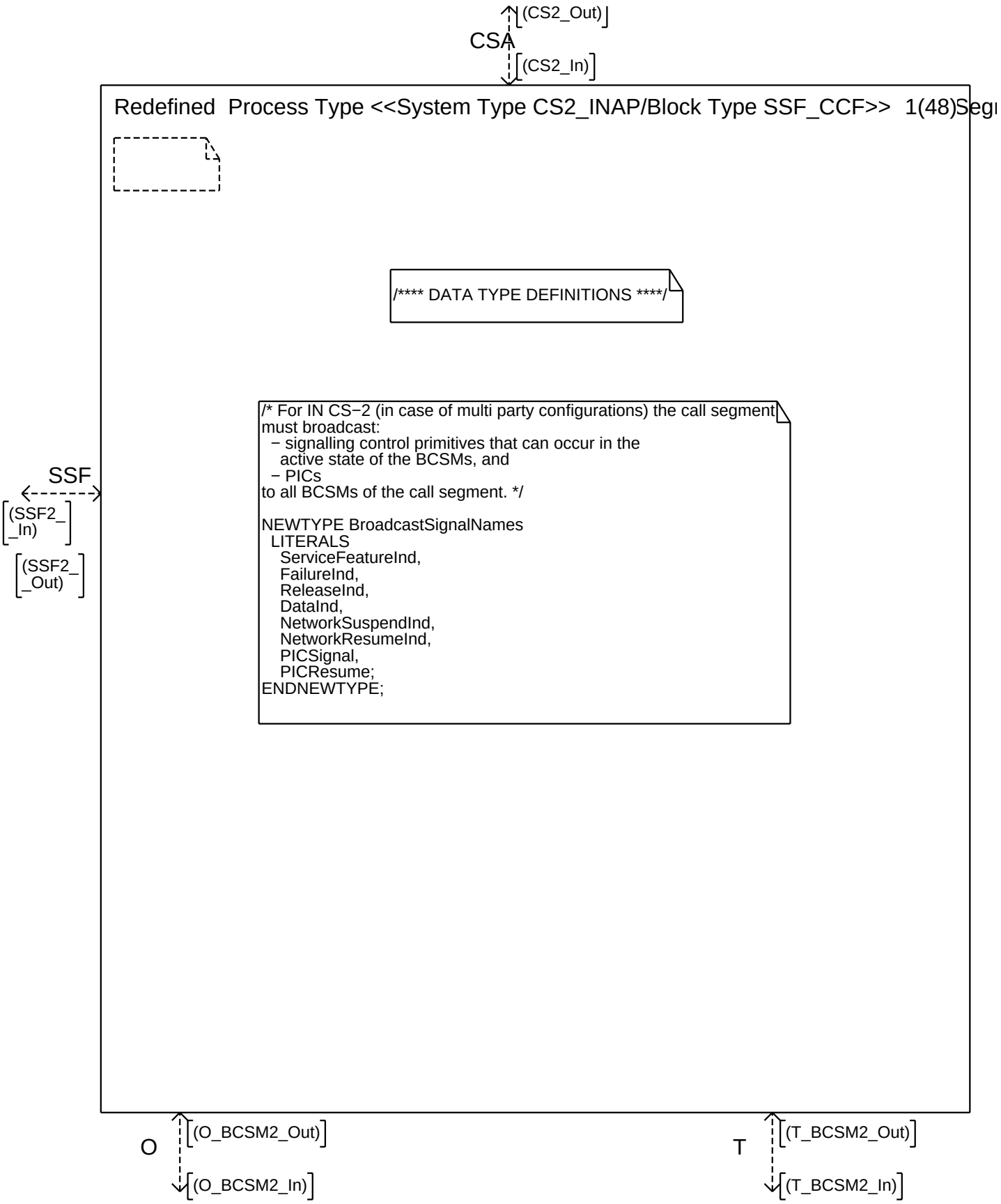

Procedure ReleaseCall
1(2)
;FPAR
IN RCArg ReleaseCallArg;
DCL
CallSegmentID CallSegmentID;
initial_
CallSegmentPresent
(rcArg)
false
true
CSStop
(,RCArg!initialCall_
Segment)
to call GetCSPtr
(initialCallSegmentID)
associated_
CallSegmentPresent
(rcArg)
false
true
CSStop
(,RCArg!associatedCall_
Segment!releaseCause)
to call GetCSPtr
(RCArg!associatedCallSegment!callSegment)
all_
CallSegmentsPresent
(rcArg)
true
false
Release_
AllSegments

Annex A:

Procedure ReleaseCall 2(2)



Annex A:


(CS2_Out)
CSA
(CS2_In)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> 1(48)Segm
/**** DATA TYPE DEFINITIONS ****/
/* For IN CS−2 (in case of multi party configurations) the call segment must broadcast:
− signalling control primitives that can occur in the active state of the BCSMs, and
− PICs
to all BCSMs of the call segment. */
NEWTYPE BroadcastSignalNames
LITERALS
ServiceFeatureInd,
FailureInd,
ReleaseInd,
DataInd,
NetworkSuspendInd,
NetworkResumeInd,
PICSignal,
PICResume;
ENDNEWTYPE;
SSF
(SSF2_ _In)
(SSF2_ _Out)
O
(O_BCSM2_Out)
(O_BCSM2_In)
T
(T_BCSM2_Out)
(T_BCSM2_In)

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen2(48)

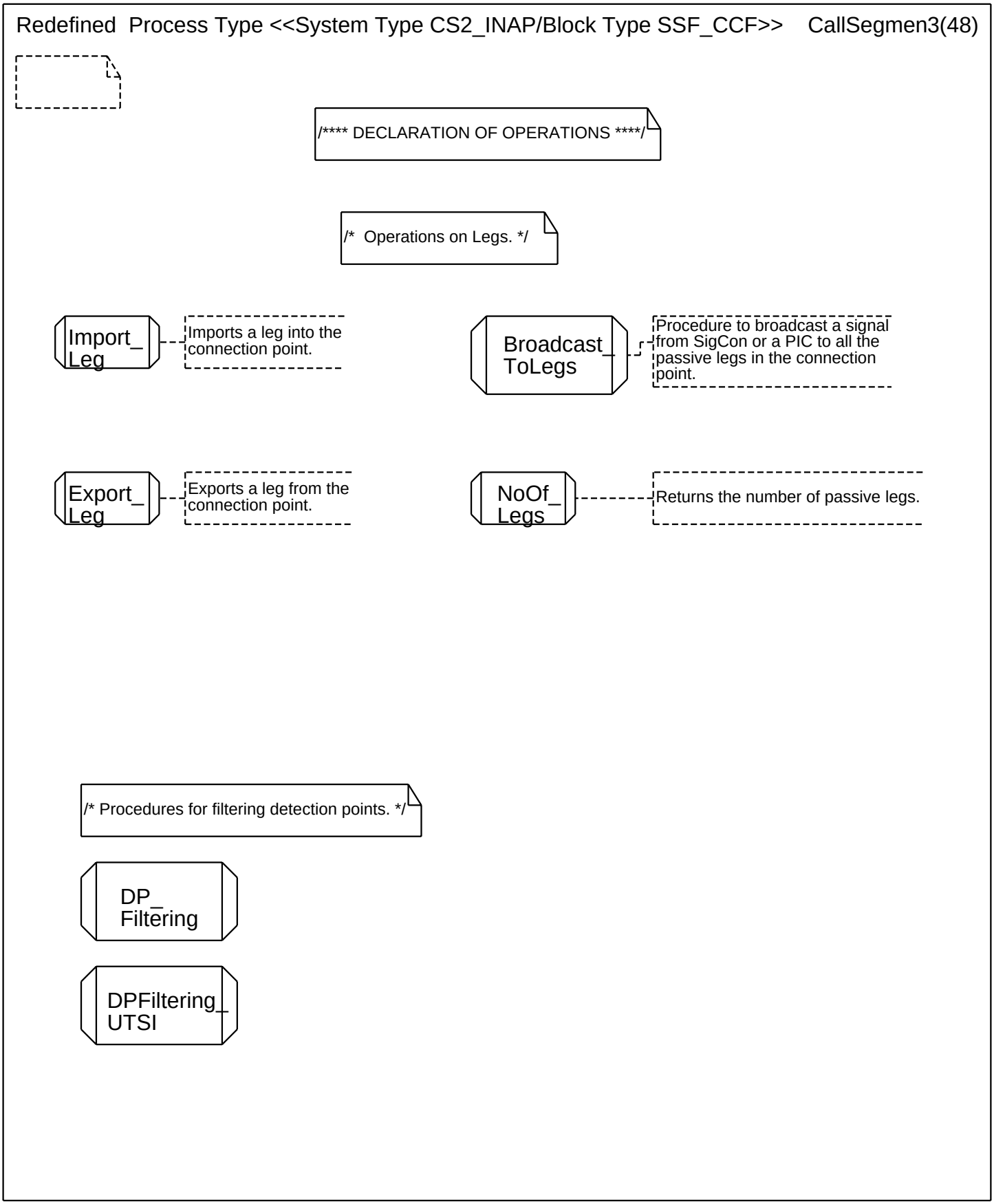
```
/**** VARIABLE DECLARATIONS ****/
```

```
DCL
  /* IN CS−2 operation arguments. */
  ctwaArg ContinueWithArgumentArg,
  dfcwaArg DisconnectForwardConnectionWithArgumentArg,
  dlArg DisconnectLegArg,
  erArg EntityReleasedArg,
  rutsiArg ReportUTSIArg,
  rrutsiArg RequestReportUTSIArg,
  sstuiArg SendSTUIArg;
```

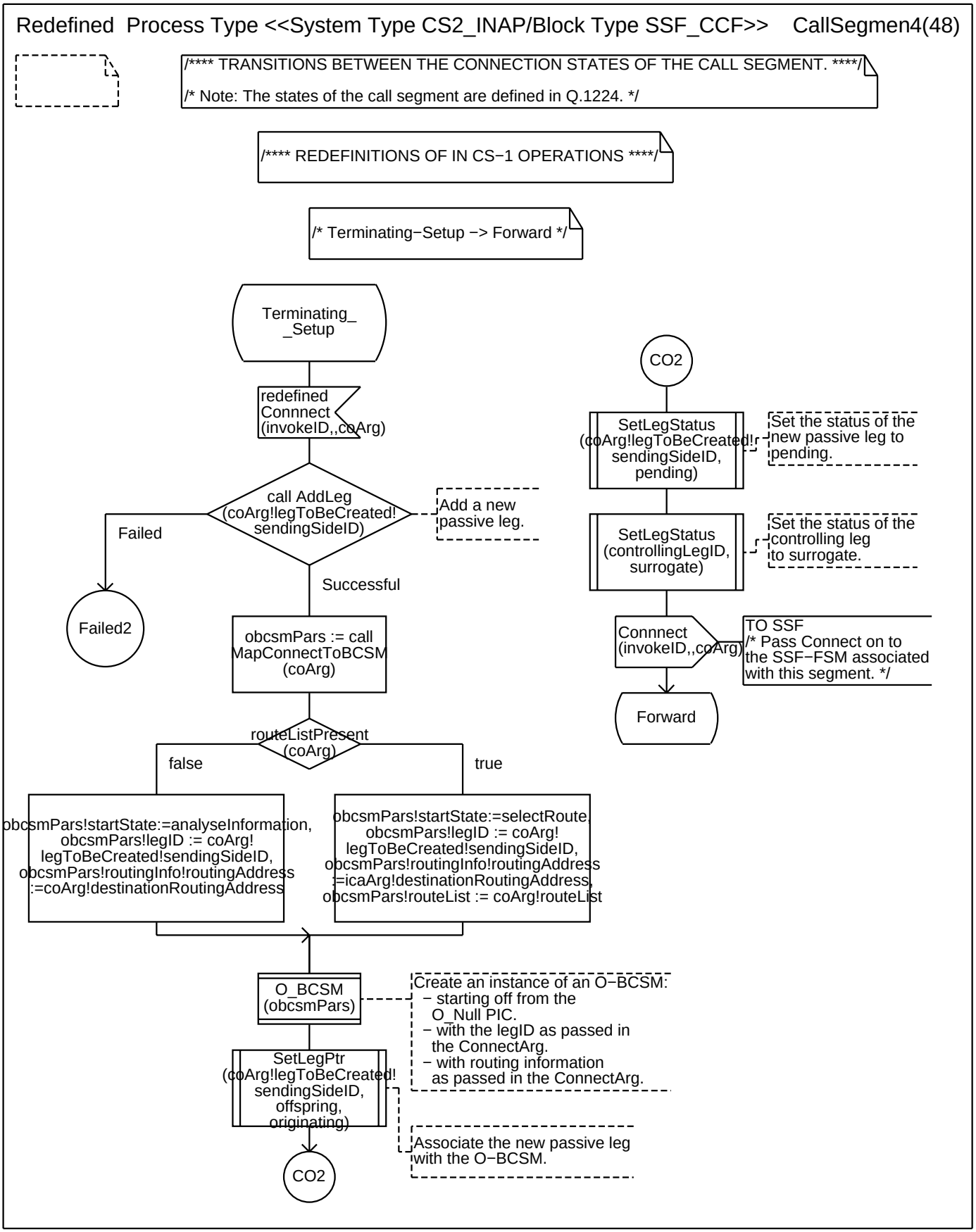
```
DCL
  /* Parameters at the signalling control interface. */
  udArg UserDataType,
  nsrArg NetworkSRType;
```

```
DCL
  /* Other variables. */
  currentLegID LegType,
  expLeg ExportLegType,
  dpUTSIArg DPUTSIArg,
  noOfMidCallDPs Natural := 0,
  noOfDisconnectDPs Natural := 0,
  noOfAbandonDPs Natural := 0,
  noOfUTSIDPs Natural := 0,
  party PartyType,
  cwaFlag Boolean := false,
  /* Used to indicate the status of the call processing. */
  setupSent boolean := false;    /* Indicates if SetupReq is sent to the called party */
```

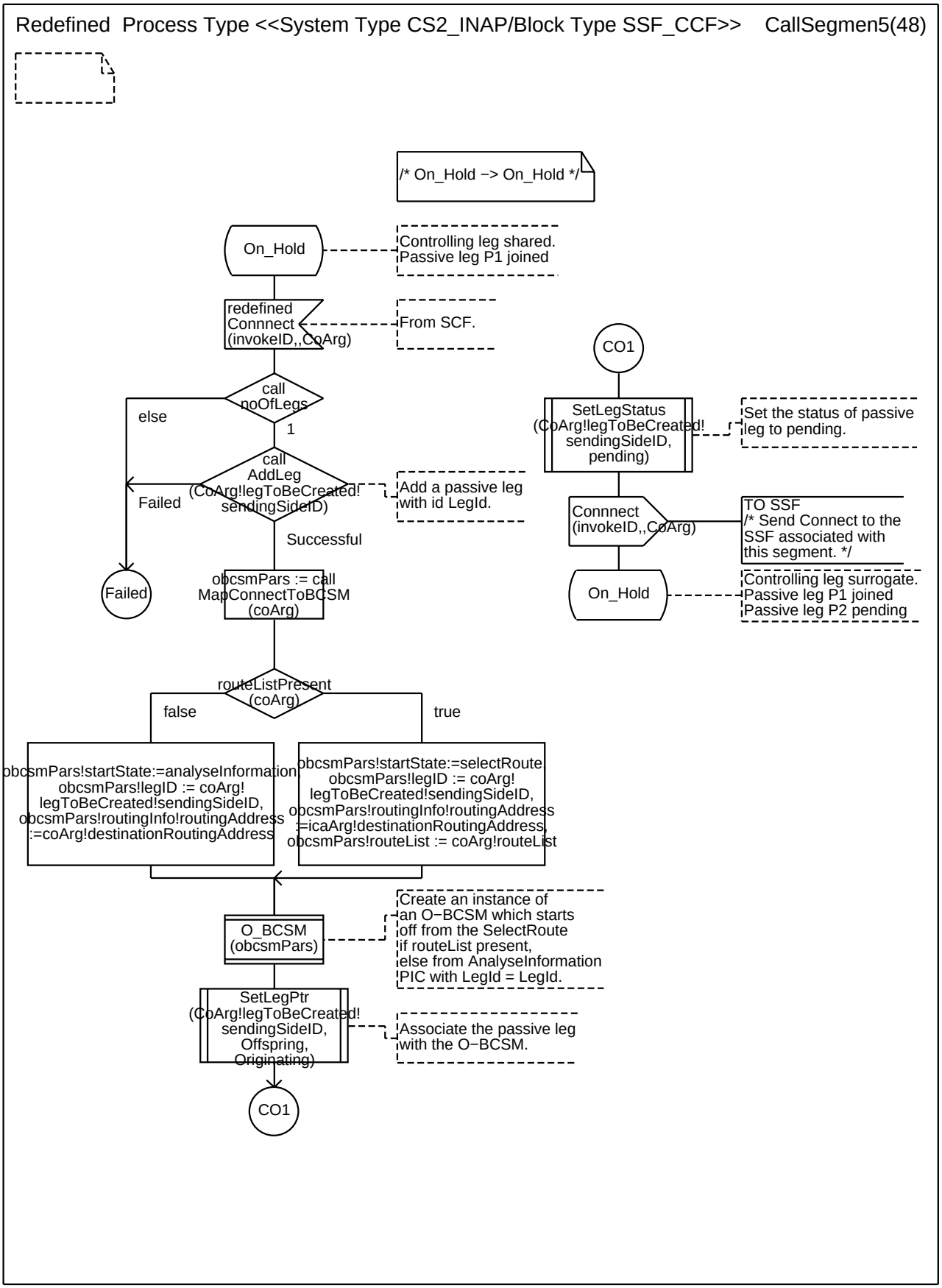
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen3(48)
/**** DECLARATION OF OPERATIONS ****/
/* Operations on Legs. */
Import_ Leg
Imports a leg into the connection point.
Broadcast_ ToLegs
Procedure to broadcast a signal from SigCon or a PIC to all the passive legs in the connection point.
Export_ Leg
Exports a leg from the connection point.
NoOf_ Legs
Returns the number of passive legs.
/* Procedures for filtering detection points. */
DP_ Filtering
DPFiltering_ UTSI

Annex A:

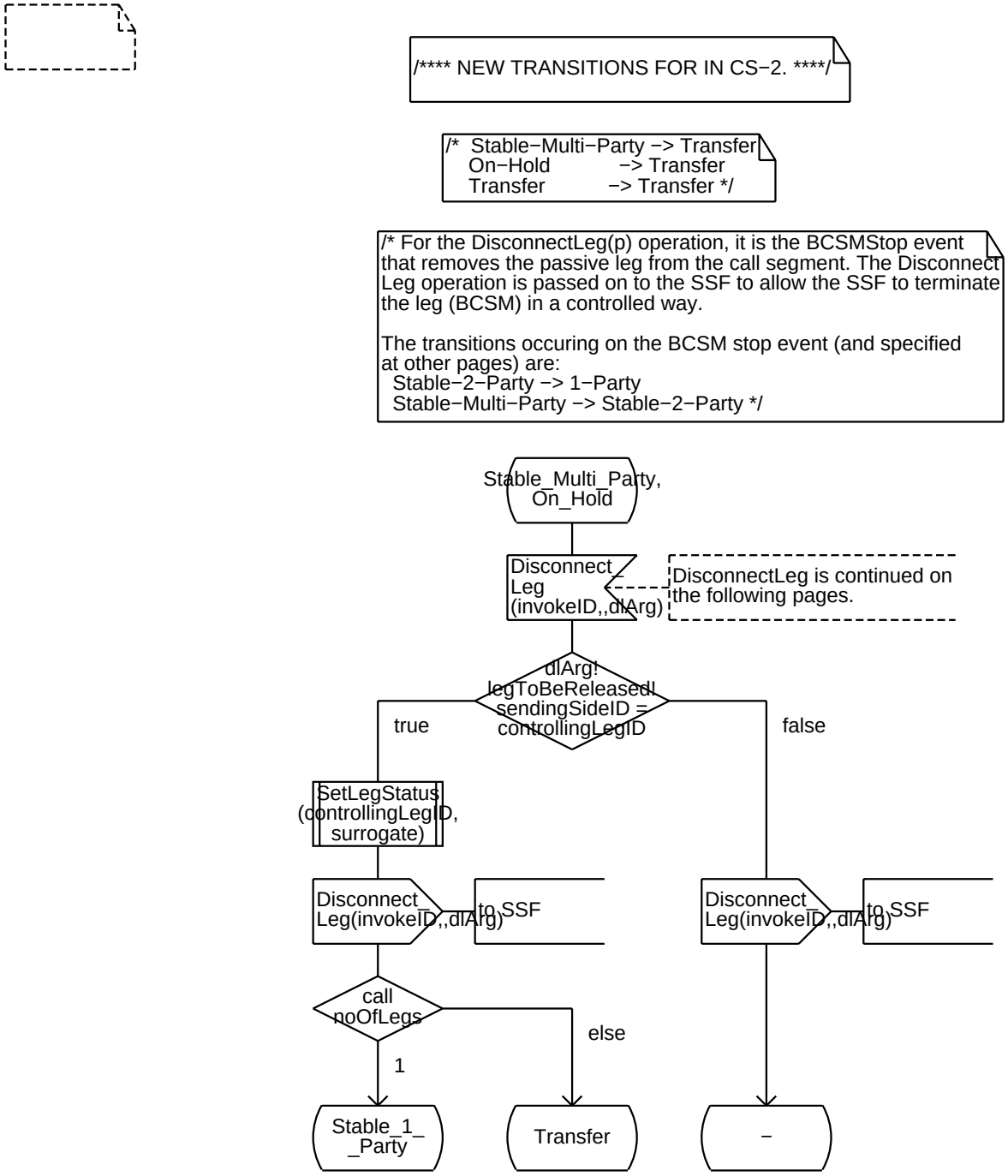

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen4(48)
/**** TRANSITIONS BETWEEN THE CONNECTION STATES OF THE CALL SEGMENT. ****/
/* Note: The states of the call segment are defined in Q.1224. */
/**** REDEFINITIONS OF IN CS−1 OPERATIONS ****/
/* Terminating−Setup −> Forward */
Terminating_ _Setup
redefined Connnect (invokeID,,coArg)
call AddLeg (coArg!legToBeCreated! sendingSideID)
Add a new passive leg.
Failed
Failed2
Successful
obcsmPars := call MapConnectToBCSM (coArg)
routeListPresent (coArg)
false
true
obcsmPars!startState:=analyseInformation, obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=coArg!destinationRoutingAddress
obcsmPars!startState:=selectRoute, obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=icaArg!destinationRoutingAddress, obcsmPars!routeList := coArg!routeList
O_BCSM (obcsmPars)
Create an instance of an O−BCSM:
− starting off from the O_Null PIC.
− with the legID as passed in the ConnectArg.
− with routing information as passed in the ConnectArg.
SetLegPtr (coArg!legToBeCreated! sendingSideID, offspring, originating)
Associate the new passive leg with the O−BCSM.
CO2
CO2
SetLegStatus (coArg!legToBeCreated! sendingSideID, pending)
Set the status of the new passive leg to pending.
SetLegStatus (controllingLegID, surrogate)
Set the status of the controlling leg to surrogate.
Connnect (invokeID,,coArg)
TO SSF
/* Pass Connect on to the SSF−FSM associated with this segment. */
Forward

Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen5(48)
/* On_Hold −> On_Hold */
On_Hold
Controlling leg shared.
Passive leg P1 joined
redefined
Connnect
(invokeID,,CoArg)
From SCF.
call
noOfLegs
else
1
call
AddLeg
(CoArg!legToBeCreated!
sendingSideID)
Failed
Add a passive leg
with id LegId.
Successful
Failed
obcsmPars := call
MapConnectToBCSM
(coArg)
routeListPresent
(coArg)
false
true
obcsmPars!startState:=analyseInformation,
obcsmPars!legID := coArg!
legToBeCreated!sendingSideID,
obcsmPars!routingInfo!routingAddress
:=coArg!destinationRoutingAddress
obcsmPars!startState:=selectRoute
obcsmPars!legID := coArg!
legToBeCreated!sendingSideID,
obcsmPars!routingInfo!routingAddress
:=icaArg!destinationRoutingAddress,
obcsmPars!routeList := coArg!routeList
O_BCSM
(obcsmPars)
Create an instance of
an O−BCSM which starts
off from the SelectRoute
if routeList present,
else from AnalyseInformation
PIC with LegId = LegId.
SetLegPtr
(CoArg!legToBeCreated!
sendingSideID,
Offspring,
Originating)
Associate the passive leg
with the O−BCSM.
CO1
CO1
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. */
On_Hold
Controlling leg surrogate.
Passive leg P1 joined
Passive leg P2 pending

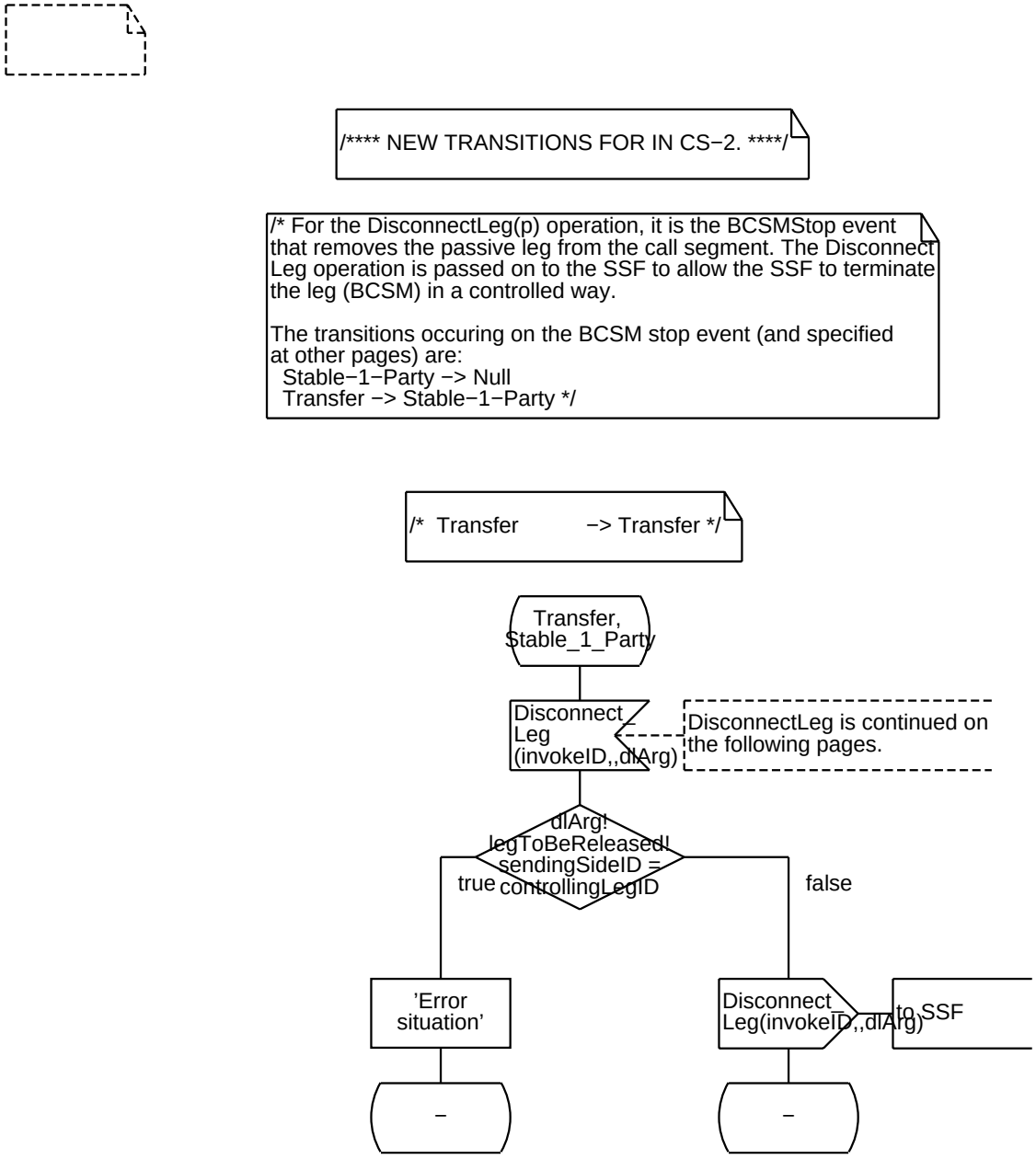
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen6(48)



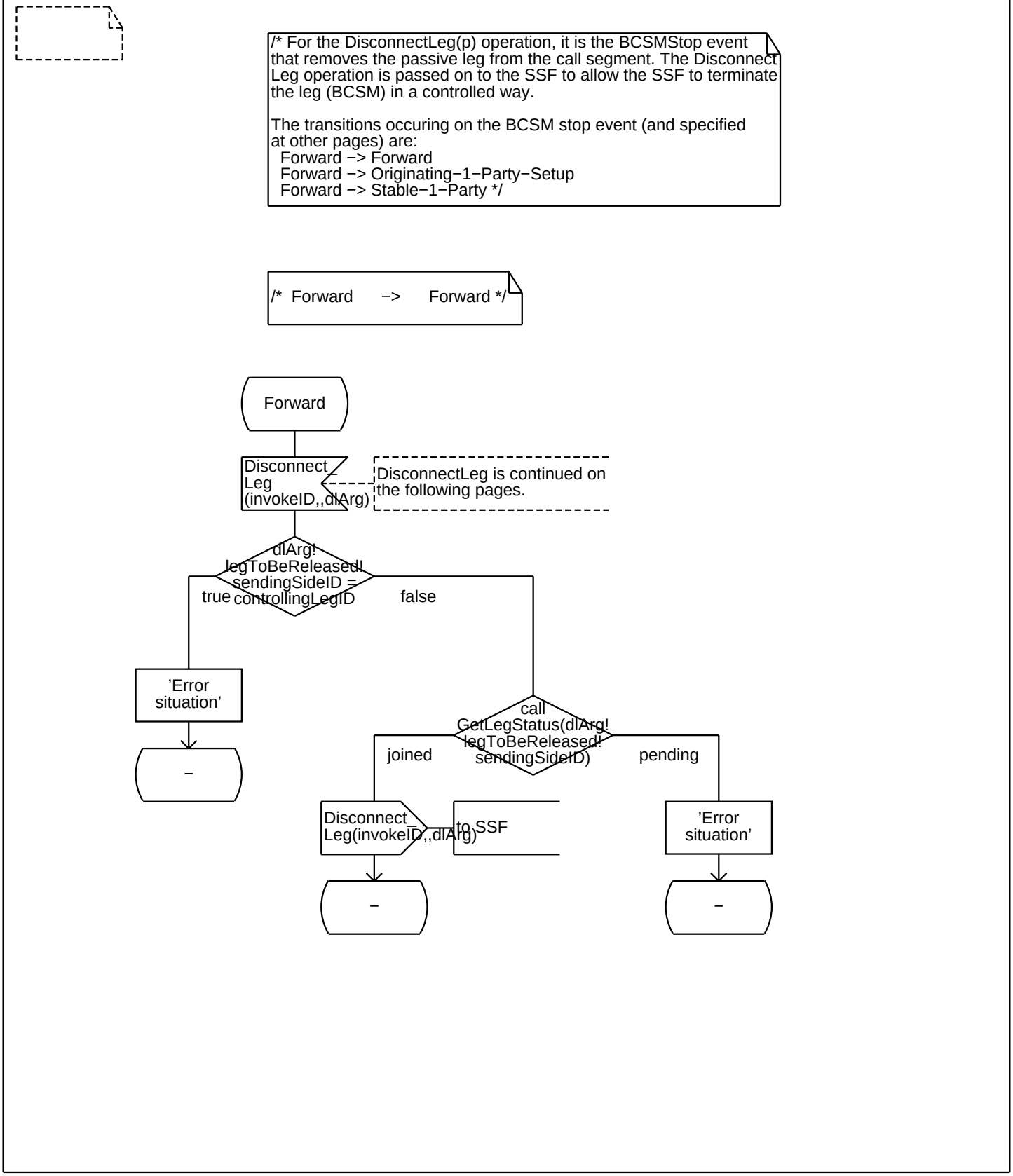
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen7(48)

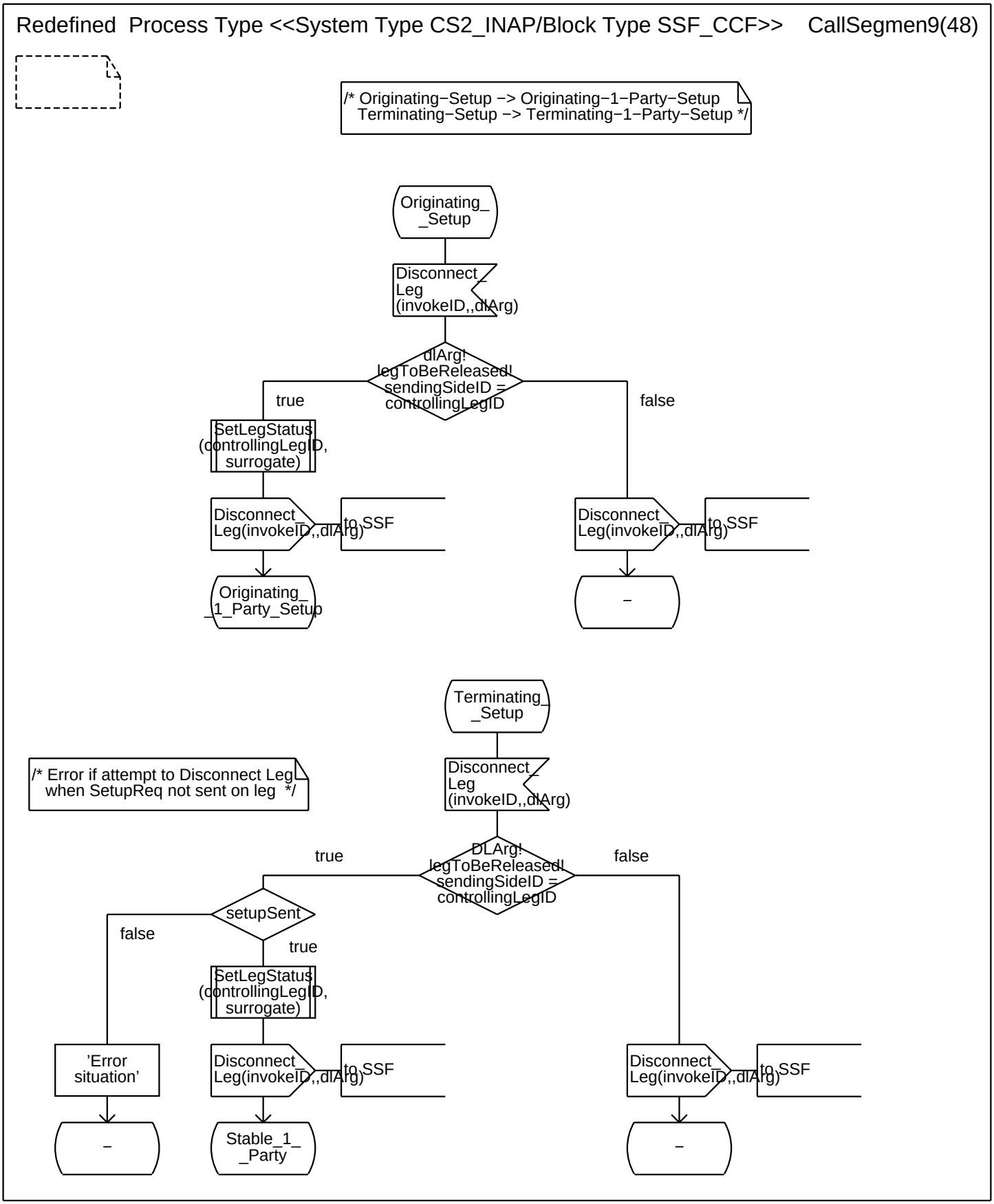


Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen8(48)

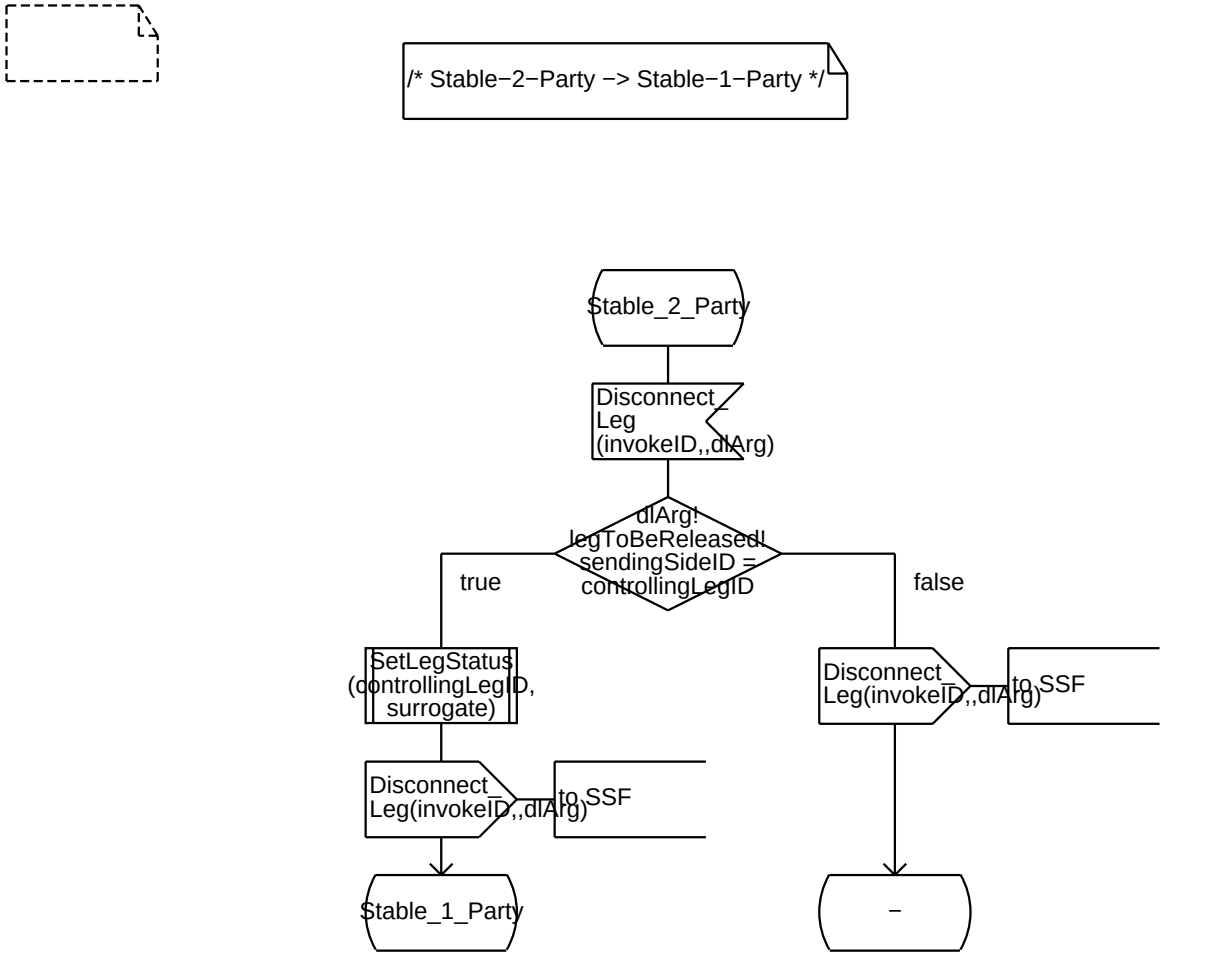


Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmen9(48)
/* Originating−Setup −> Originating−1−Party−Setup
Terminating−Setup −> Terminating−1−Party−Setup */
Originating_
_Setup
Disconnect_
Leg
(invokeID,,dlArg)
dlArg!
legToBeReleased!
sendingSideID =
controllingLegID
true
false
SetLegStatus
(controllingLegID,
surrogate)
Disconnect_
Leg(invokeID,,dlArg)
to SSF
Disconnect_
Leg(invokeID,,dlArg)
to SSF
Originating_
_1_Party_Setup
–
Terminating_
_Setup
/* Error if attempt to Disconnect Leg
when SetupReq not sent on leg */
Disconnect_
Leg
(invokeID,,dlArg)
DLArg!
legToBeReleased!
sendingSideID =
controllingLegID
true
false
setupSent
false
true
SetLegStatus
(controllingLegID,
surrogate)
'Error
situation'
Disconnect_
Leg(invokeID,,dlArg)
to SSF
Disconnect_
Leg(invokeID,,dlArg)
to SSF
–
Stable_1_
_Party
–

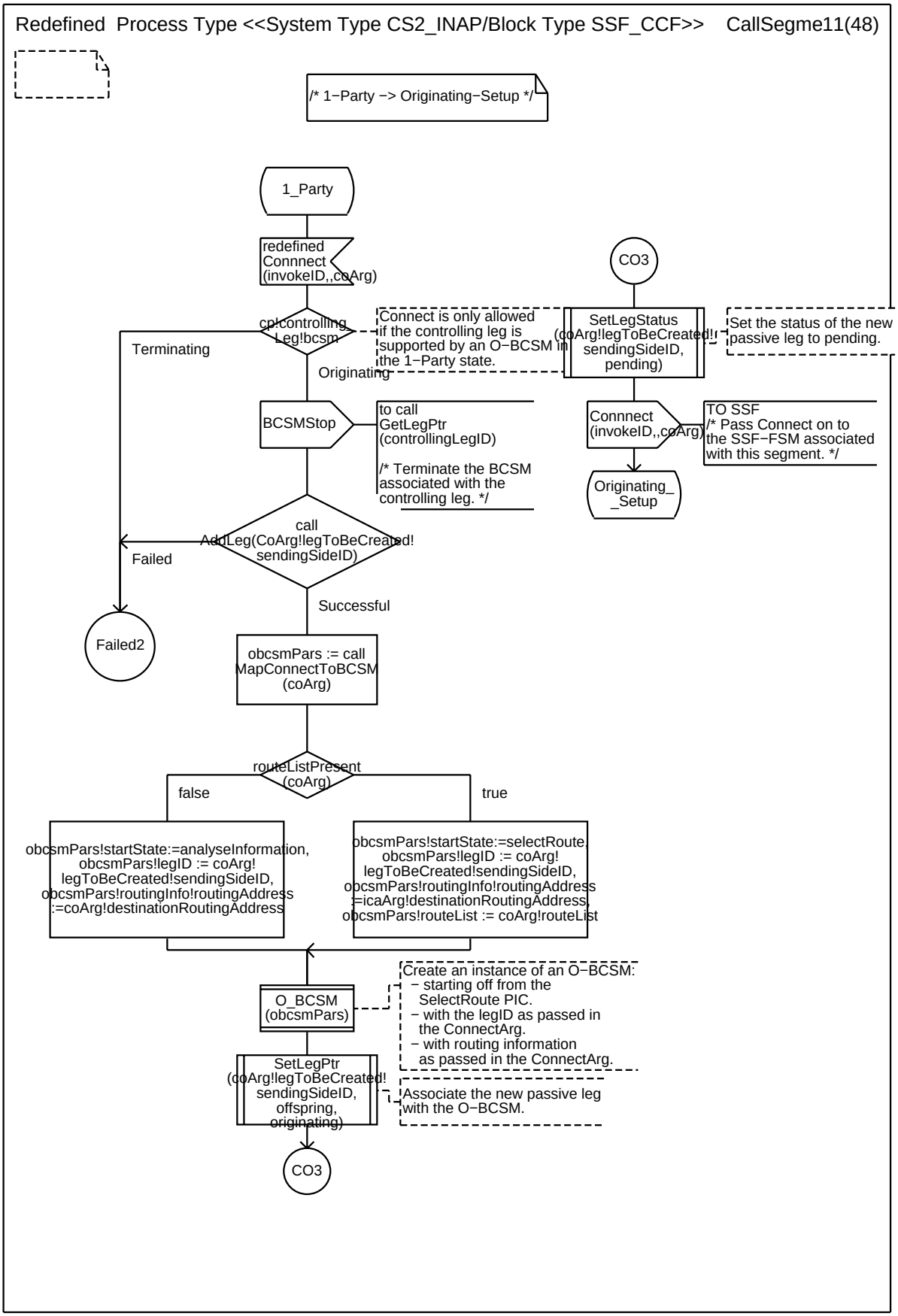
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme10(48)

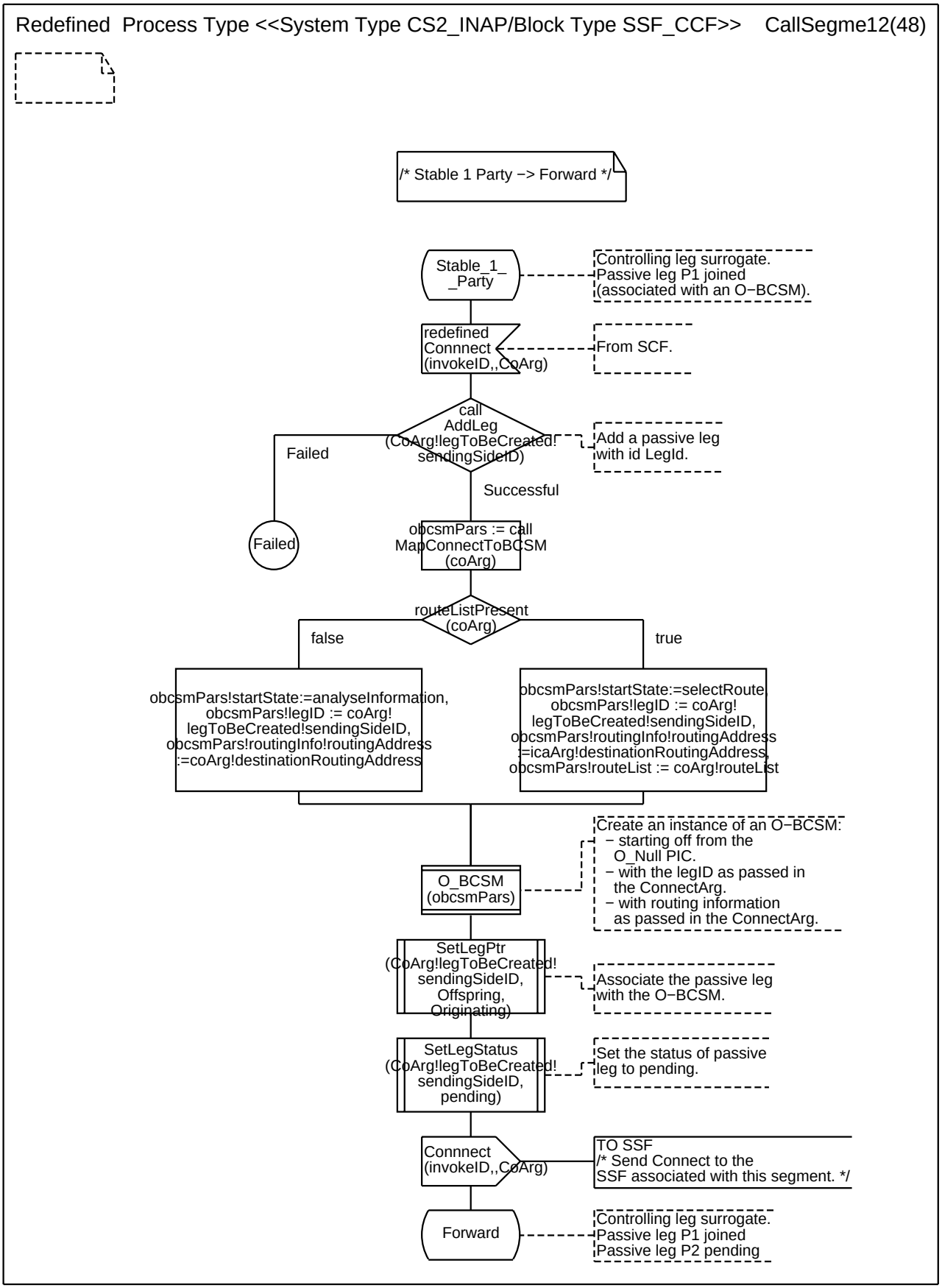


/* Notes:
− DisconnectLeg is not defined for the following states:
Originating−1−Party−Setup, Terminating−1−Party−Setup,
Stable−1−Party.
− DisconnectLeg of the controlling leg in any multi party state
will result in that all BCSMs will generate a ReleaseReq towards
the signalling control. The signalling control should filter these
ReleaseReqs and pass on only one Release message to the signalling. */

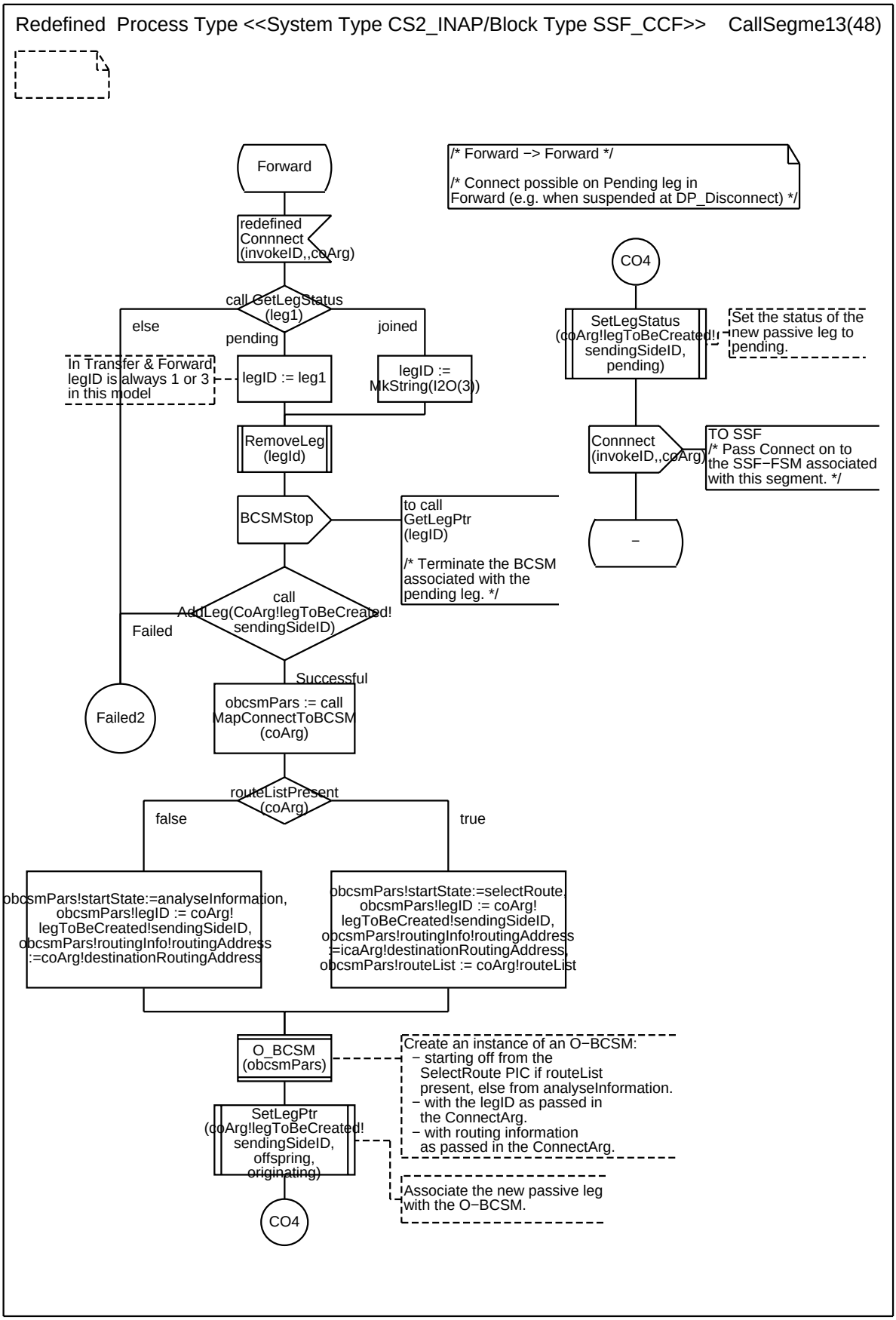
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme11(48)
/* 1−Party −> Originating−Setup */
1_Party
redefined Connnect (invokeID,,coArg)
cp!controlling Leg!bcsm
Terminating
Originating
Connect is only allowed if the controlling leg is supported by an O−BCSM in the 1−Party state.
CO3
SetLegStatus (coArg!legToBeCreated!! sendingSideID, pending)
Set the status of the new passive leg to pending.
BCSMStop
to call GetLegPtr (controllingLegID)
/* Terminate the BCSM associated with the controlling leg. */
Connnect (invokeID,,coArg)
TO SSF
/* Pass Connect on to the SSF−FSM associated with this segment. */
Originating_ _Setup
call AddLeg(CoArg!legToBeCreated! sendingSideID)
Failed
Successful
Failed2
obcsmPars := call MapConnectToBCSM (coArg)
routeListPresent (coArg)
false
true
obcsmPars!startState:=analyseInformation, obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=coArg!destinationRoutingAddress
obcsmPars!startState:=selectRoute, obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=icaArg!destinationRoutingAddress, obcsmPars!routeList := coArg!routeList
O_BCSM (obcsmPars)
Create an instance of an O−BCSM:
− starting off from the SelectRoute PIC.
− with the legID as passed in the ConnectArg.
− with routing information as passed in the ConnectArg.
SetLegPtr (coArg!legToBeCreated! sendingSideID, offspring, originating)
Associate the new passive leg with the O−BCSM.
CO3

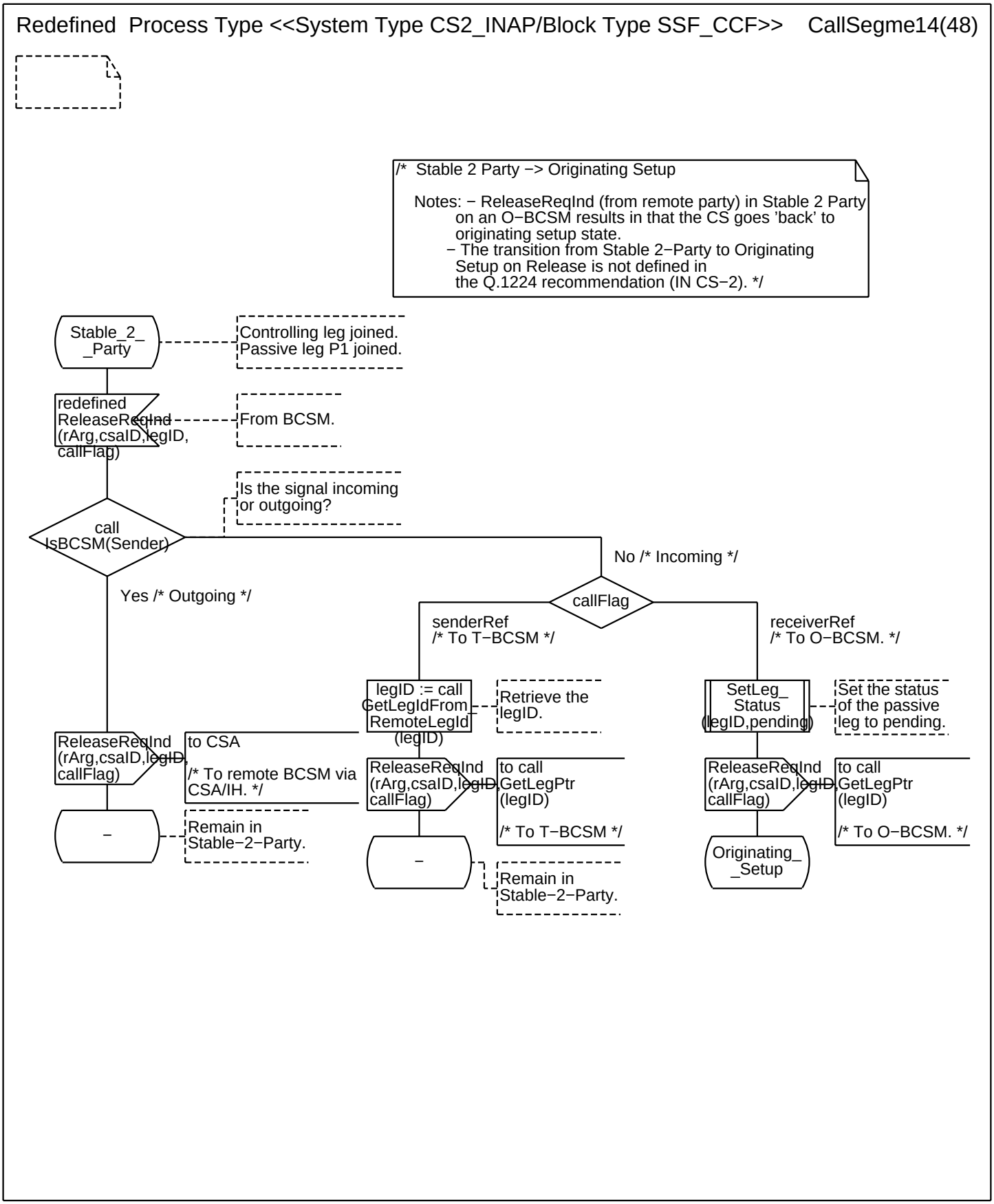
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme12(48)
/* Stable 1 Party −> Forward */
Stable_1_ _Party
Controlling leg surrogate. Passive leg P1 joined (associated with an O−BCSM).
redefined Connnect (invokeID,,CoArg)
From SCF.
call AddLeg (CoArg!legToBeCreated! sendingSideID)
Add a passive leg with id LegId.
Failed
Successful
Failed
obcsmPars := call MapConnectToBCSM (coArg)
routeListPresent (coArg)
false
true
obcsmPars!startState:=analyseInformation, obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=coArg!destinationRoutingAddress
obcsmPars!startState:=selectRoute obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=icaArg!destinationRoutingAddress, obcsmPars!routeList := coArg!routeList
Create an instance of an O−BCSM:
− starting off from the O_Null PIC.
− with the legID as passed in the ConnectArg.
− with routing information as passed in the ConnectArg.
O_BCSM (obcsmPars)
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

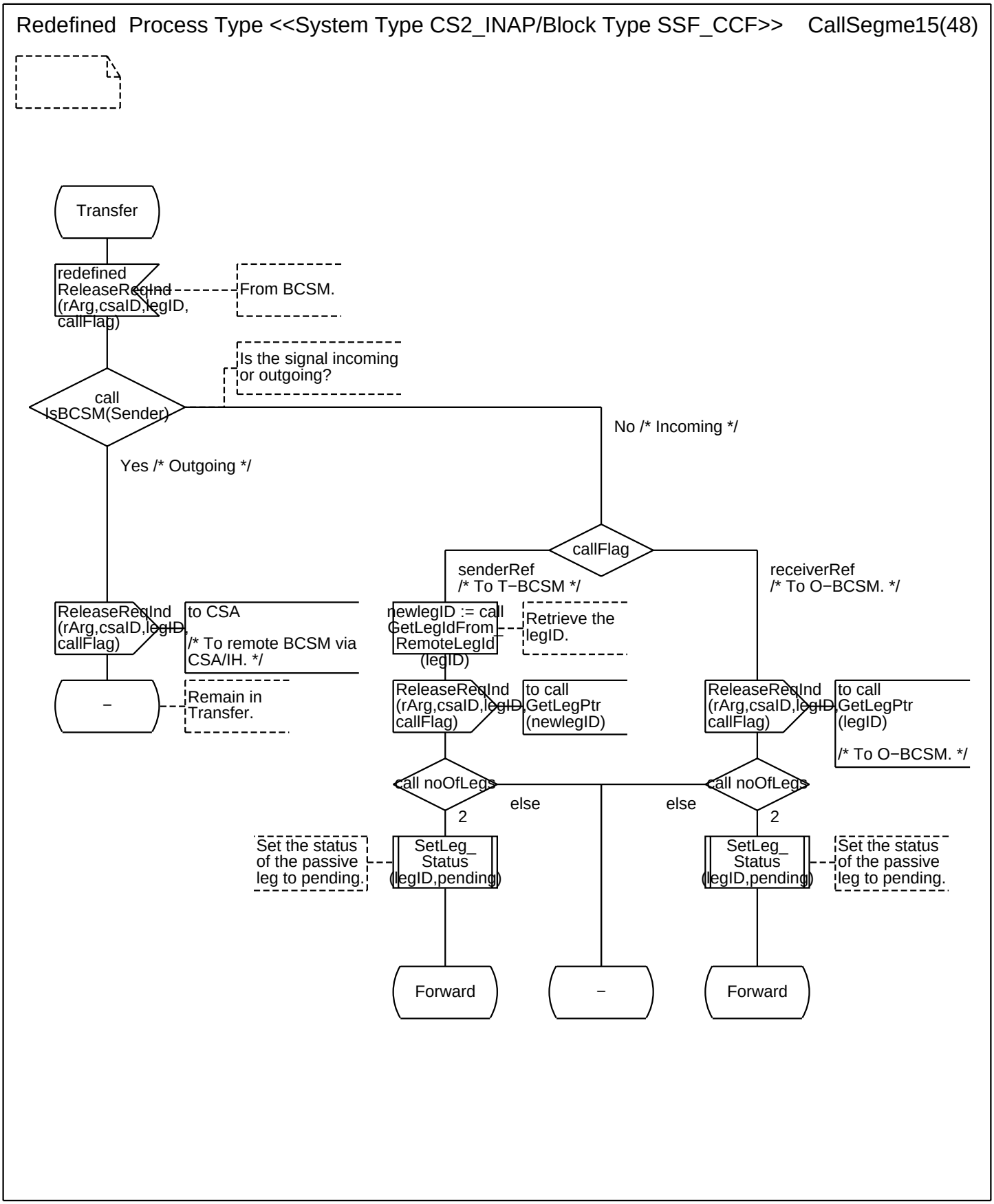
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme13(48)
Forward
/* Forward −> Forward */
/* Connect possible on Pending leg in Forward (e.g. when suspended at DP_Disconnect) */
redefined Connnect (invokeID,,coArg)
call GetLegStatus (leg1)
else
pending
joined
In Transfer & Forward legID is always 1 or 3 in this model
legID := leg1
legID := MkString(I2O(3))
RemoveLeg (legId)
BCSMStop
to call GetLegPtr (legID)
/* Terminate the BCSM associated with the pending leg. */
call AddLeg(CoArg!legToBeCreated! sendingSideID)
Failed
Failed2
Successful
obcsmPars := call MapConnectToBCSM (coArg)
routeListPresent (coArg)
false
true
obcsmPars!startState:=analyseInformation, obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=coArg!destinationRoutingAddress
obcsmPars!startState:=selectRoute, obcsmPars!legID := coArg! legToBeCreated!sendingSideID, obcsmPars!routingInfo!routingAddress :=icaArg!destinationRoutingAddress, obcsmPars!routeList := coArg!routeList
O_BCSM (obcsmPars)
Create an instance of an O−BCSM:
− starting off from the SelectRoute PIC if routeList present, else from analyseInformation.
− with the legID as passed in the ConnectArg.
− with routing information as passed in the ConnectArg.
SetLegPtr (coArg!legToBeCreated! sendingSideID, offspring, originating)
Associate the new passive leg with the O−BCSM.
CO4
CO4
SetLegStatus (coArg!legToBeCreated!! sendingSideID, pending)
Set the status of the new passive leg to pending.
Connnect (invokeID,,coArg)
TO SSF
/* Pass Connect on to the SSF−FSM associated with this segment. */
−

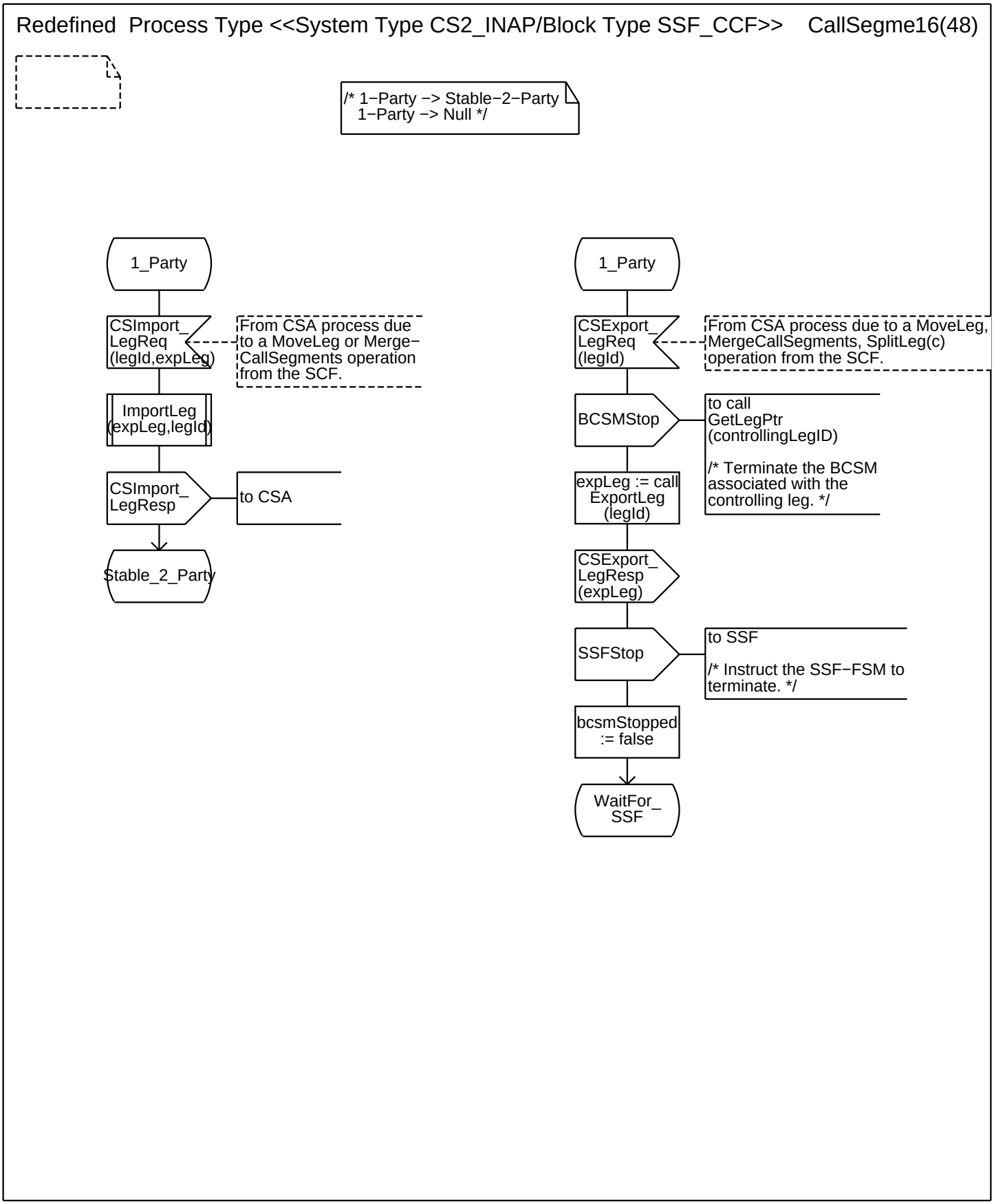
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme14(48)
/* Stable 2 Party −> Originating Setup
Notes: − ReleaseReqInd (from remote party) in Stable 2 Party on an O−BCSM results in that the CS goes 'back' to originating setup state.
− The transition from Stable 2−Party to Originating Setup on Release is not defined in the Q.1224 recommendation (IN CS−2). */
Stable_2_ _Party
Controlling leg joined. Passive leg P1 joined.
redefined 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) /* To T−BCSM */
ReleaseReqInd (rArg,csaID,legID, callFlag)
to call GetLegPtr (legID) /* To O−BCSM. */
−
Remain in Stable−2−Party.
−
Remain in Stable−2−Party.
Originating_ _Setup

Annex A:

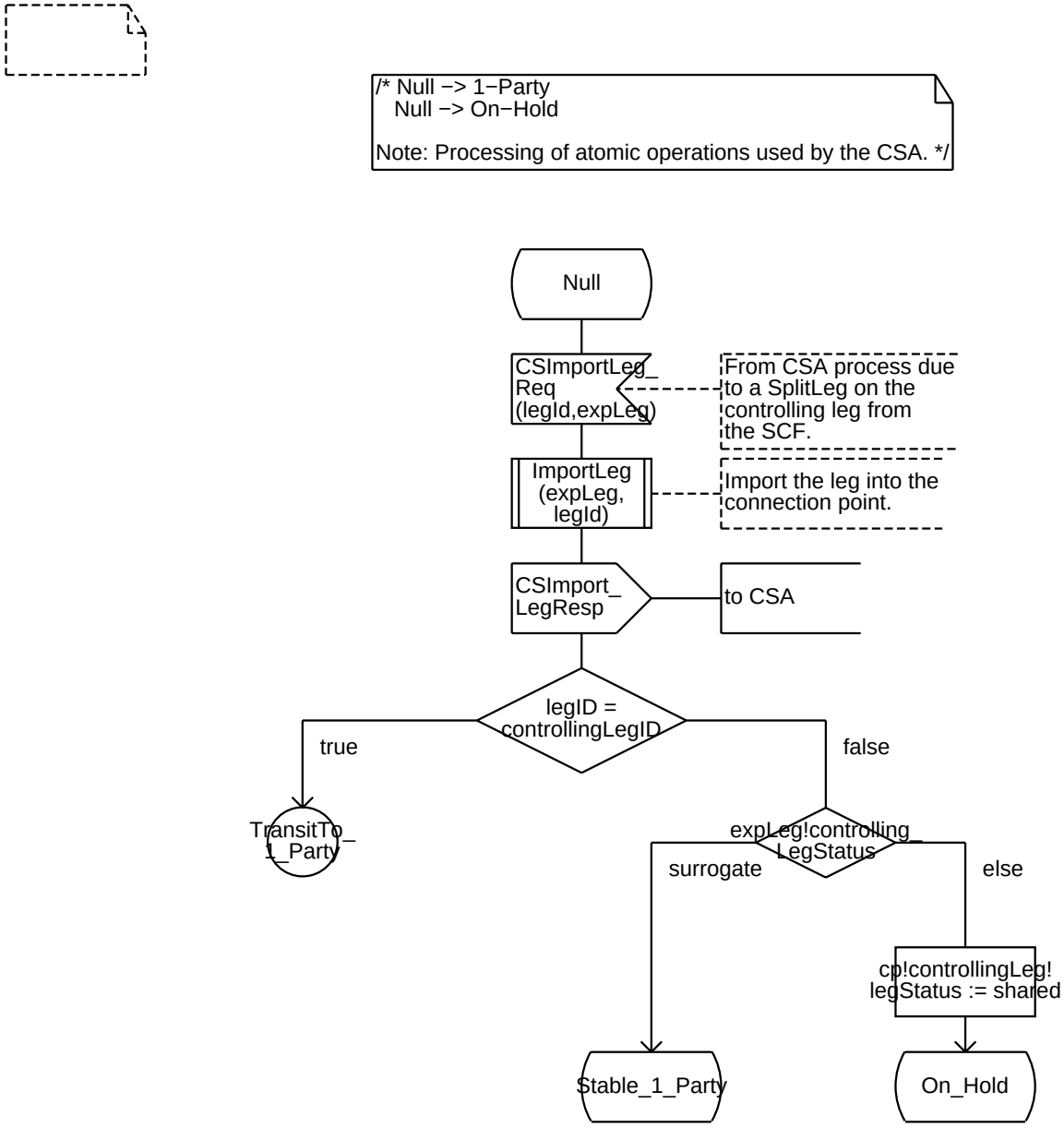

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme15(48)
Transfer
redefined ReleaseReqInd (rArg,csaID,legID, callFlag)
From BCSM.
Is the signal incoming or outgoing?
call IsBCSM(Sender)
No /* Incoming */
Yes /* Outgoing */
callFlag
senderRef /* To T−BCSM */
receiverRef /* To O−BCSM. */
ReleaseReqInd (rArg,csaID,legID, callFlag)
to CSA /* To remote BCSM via CSA/IH. */
newlegID := call GetLegIdFromRemoteLegId (legID)
Retrieve the legID.
−
Remain in Transfer.
ReleaseReqInd (rArg,csaID,legID, callFlag)
to call GetLegPtr (newlegID)
ReleaseReqInd (rArg,csaID,legID, callFlag)
to call GetLegPtr (legID)
/* To O−BCSM. */
call noOfLegs
else
2
else
call noOfLegs
2
Set the status of the passive leg to pending.
SetLeg_ Status (legID,pending)
SetLeg_ Status (legID,pending)
Set the status of the passive leg to pending.
Forward
−
Forward

Annex A:

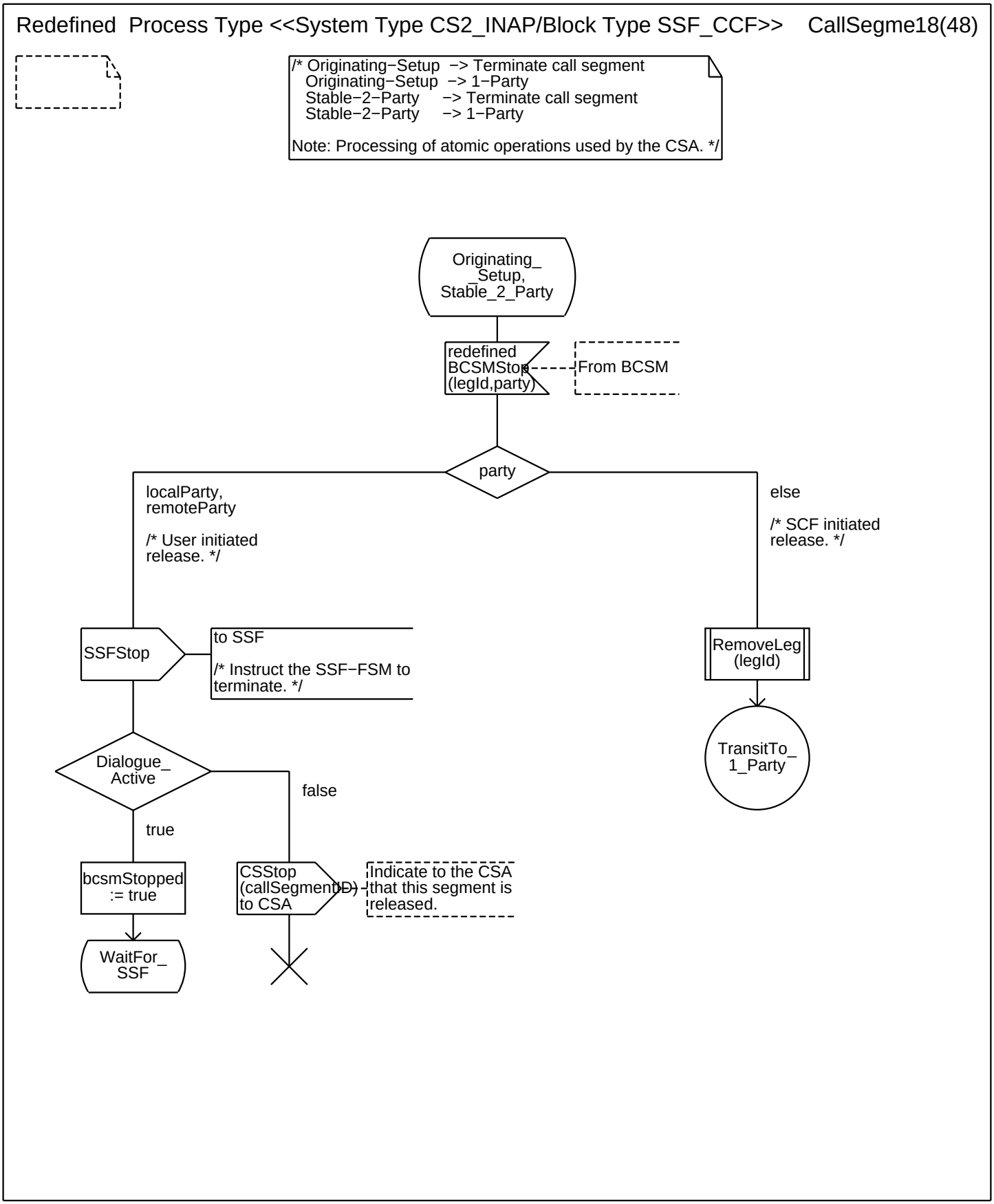

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme16(48)
/* 1−Party −> Stable−2−Party
1−Party −> Null */
1_Party
CSImport_
LegReq
(legId,expLeg)
From CSA process due to a MoveLeg or Merge−CallSegments operation from the SCF.
ImportLeg
(expLeg,legId)
CSImport_
LegResp
to CSA
Stable_2_Party
1_Party
CSExport_
LegReq
(legId)
From CSA process due to a MoveLeg, MergeCallSegments, SplitLeg(c) operation from the SCF.
BCSMStop
to call
GetLegPtr
(controllingLegID)
/* Terminate the BCSM associated with the controlling leg. */
expLeg := call
ExportLeg
(legId)
CSExport_
LegResp
(expLeg)
SSFStop
to SSF
/* Instruct the SSF−FSM to terminate. */
bcsmStopped
:= false
WaitFor_
SSF

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme17(48)

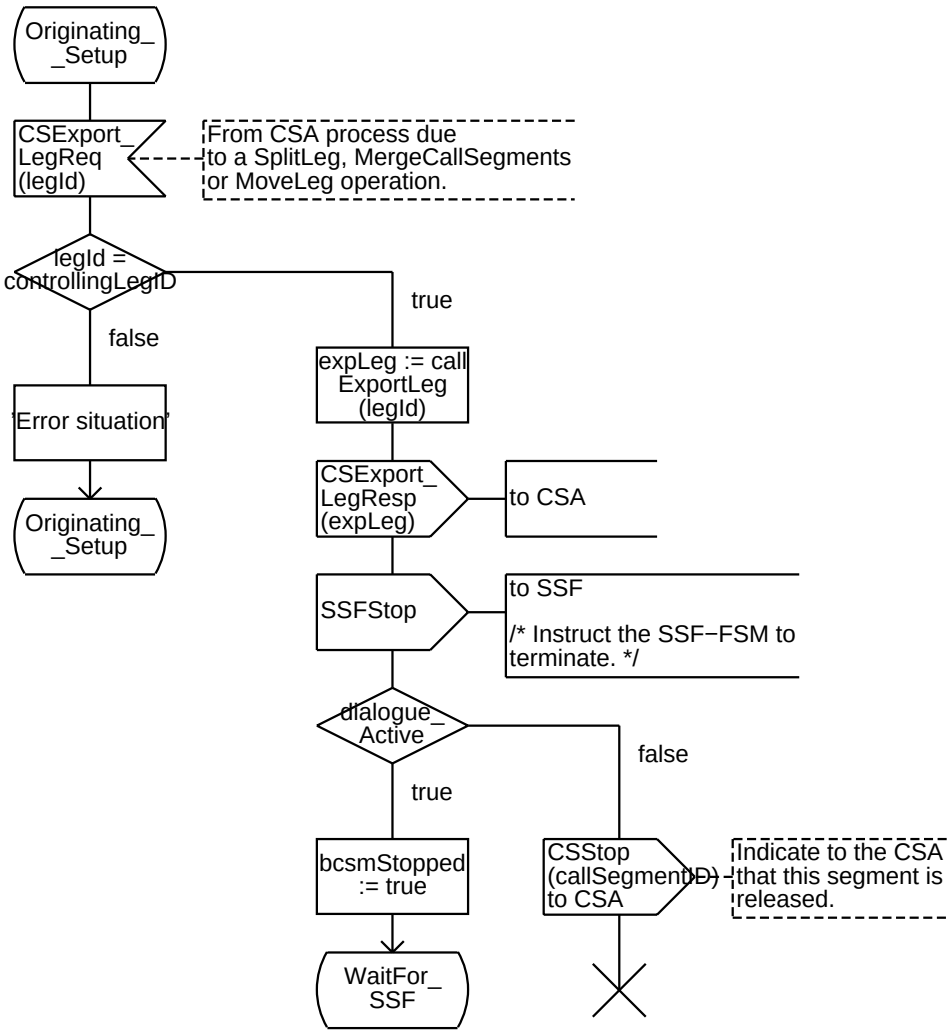


Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme18(48)
/* Originating−Setup −> Terminate call segment
Originating−Setup −> 1−Party
Stable−2−Party −> Terminate call segment
Stable−2−Party −> 1−Party
Note: Processing of atomic operations used by the CSA. */
Originating_ _Setup, Stable_2_Party
redefined BCSMStop (legId,party)
From BCSM
party
localParty, remoteParty
/* User initiated release. */
else
/* SCF initiated release. */
SSFStop
to SSF
/* Instruct the SSF−FSM to terminate. */
RemoveLeg (legId)
TransitTo_ 1_Party
Dialogue_ Active
false
true
bcsmStopped := true
CSStop (callSegmentID) to CSA
Indicate to the CSA that this segment is released.
WaitFor_ SSF

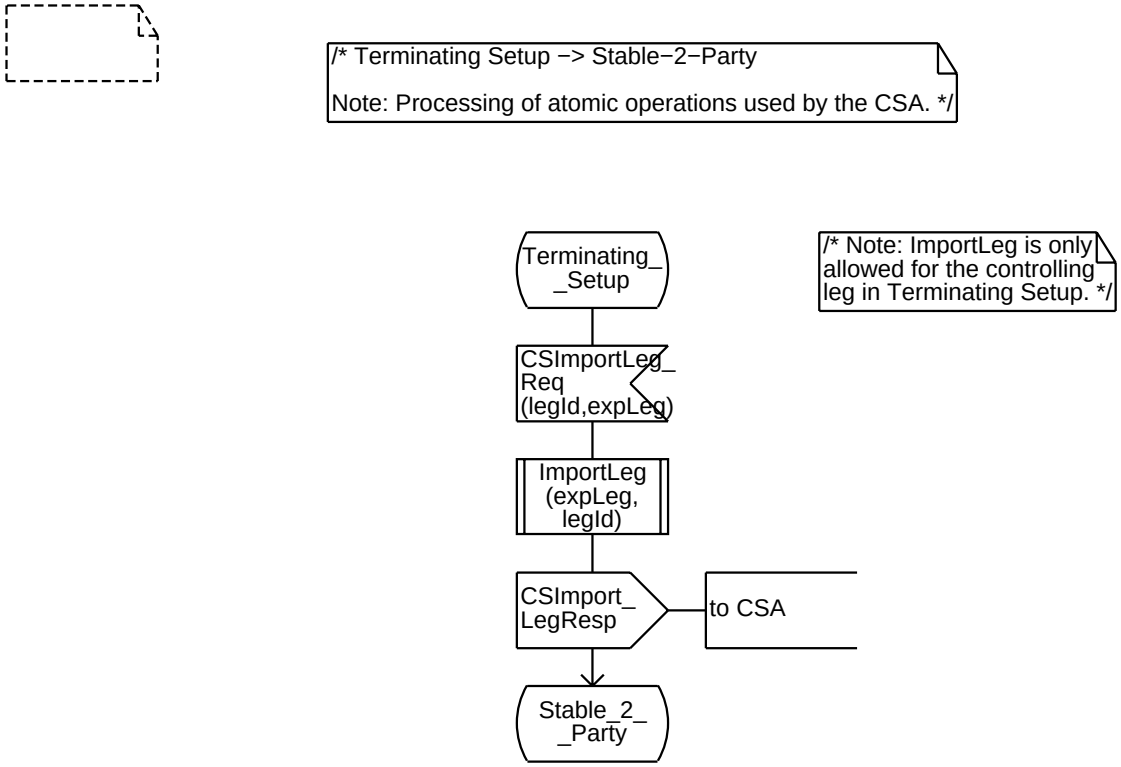
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme19(48)



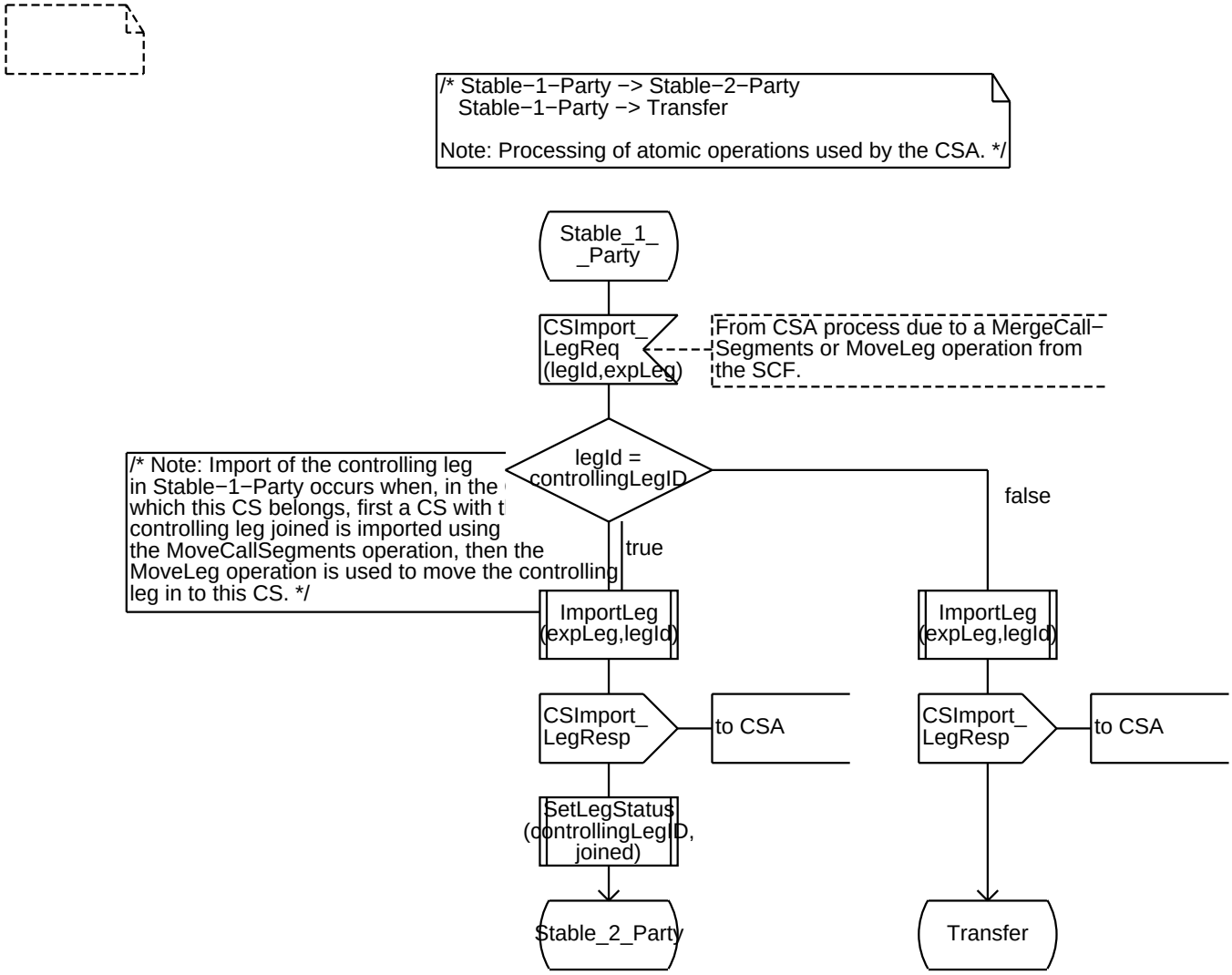
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme20(48)



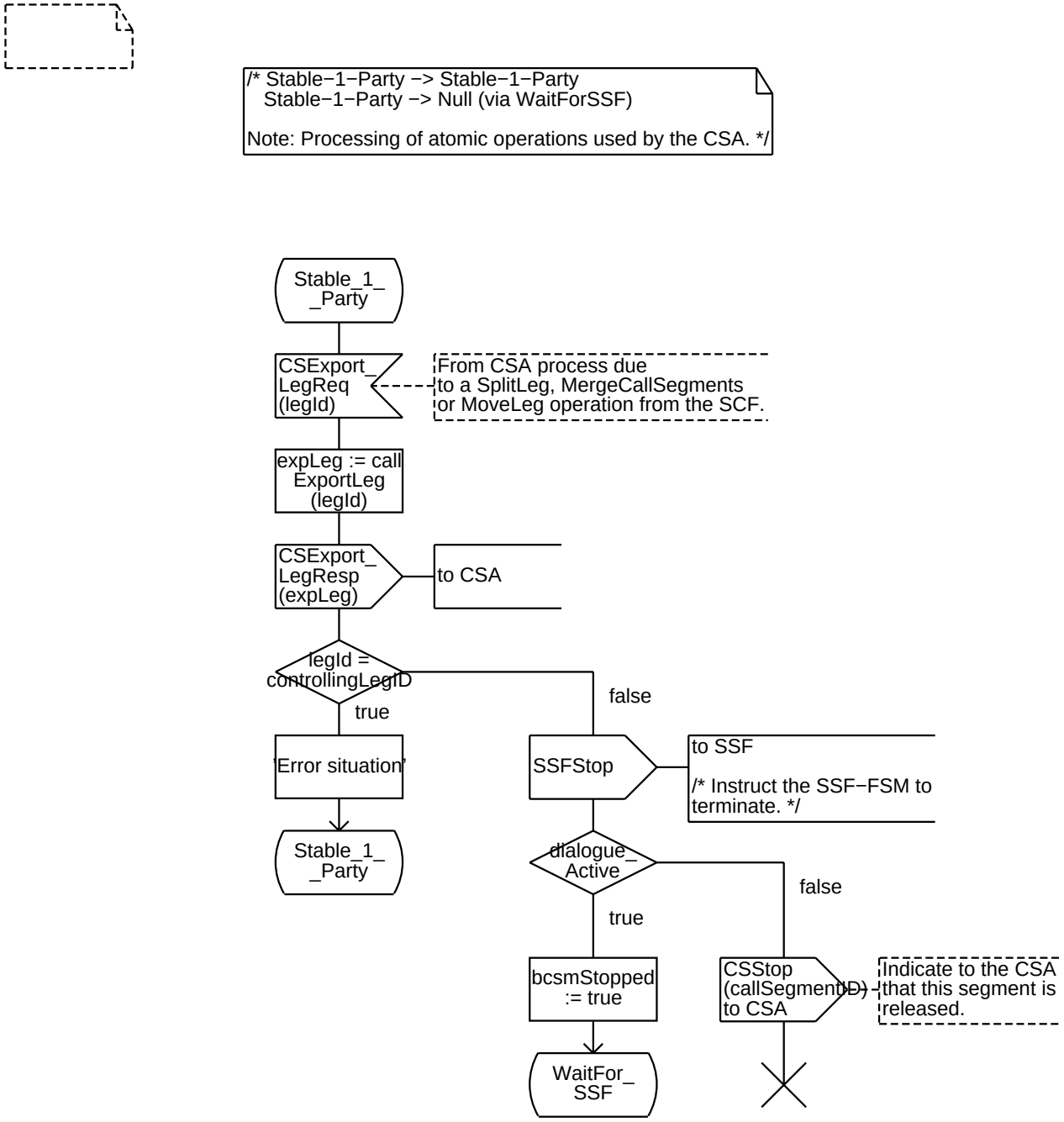
Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme21(48)

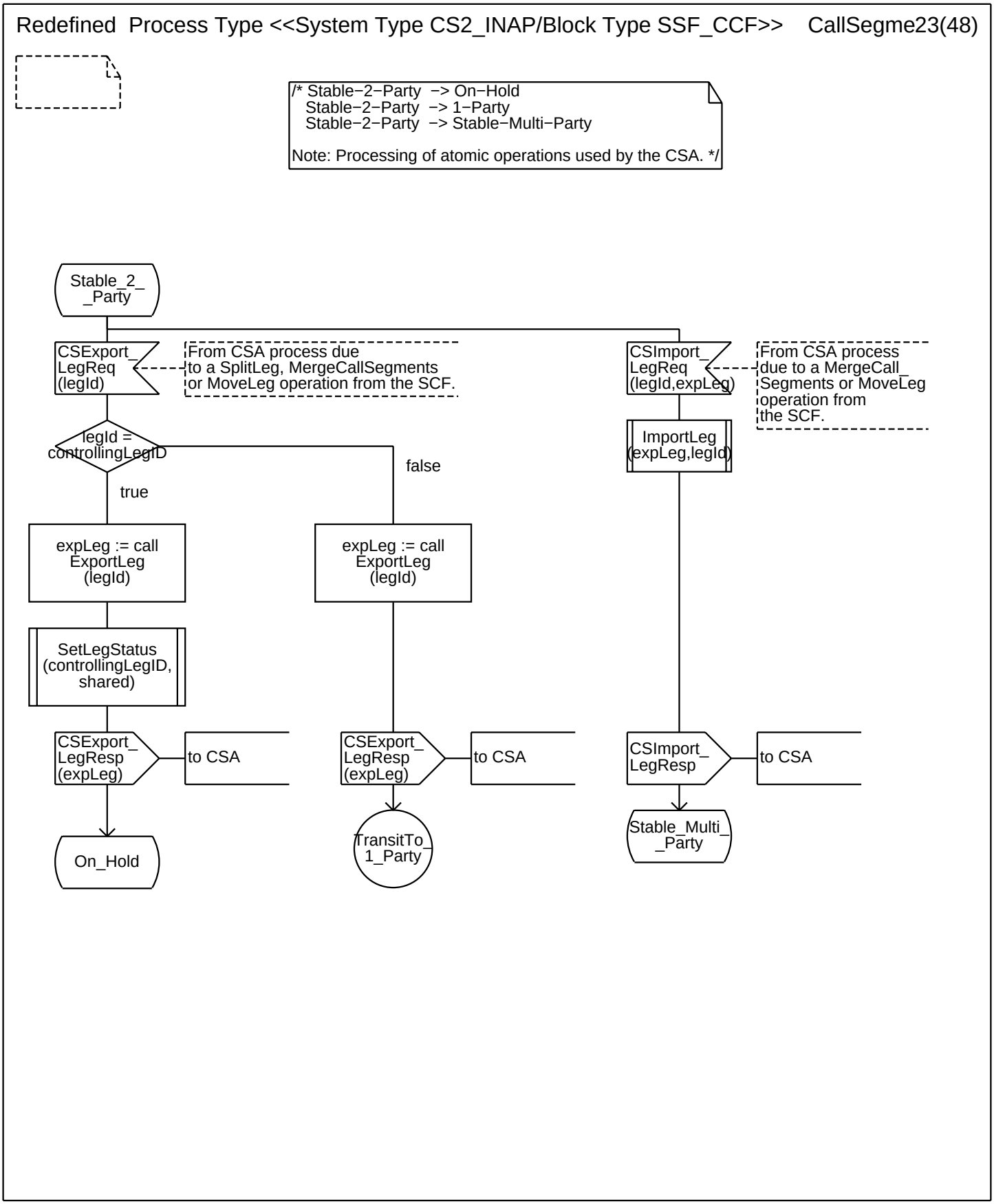


Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme22(48)

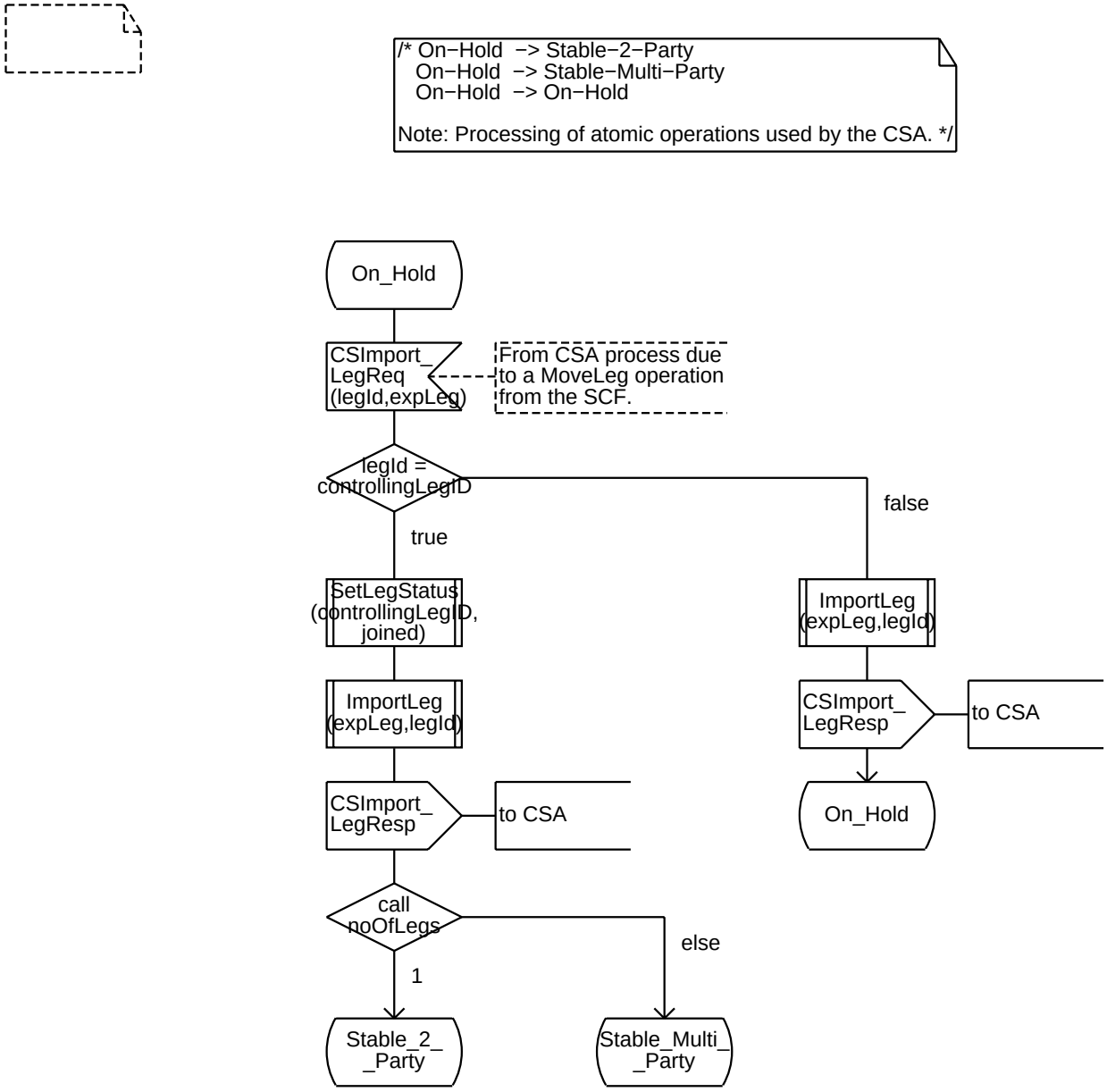


Annex A:

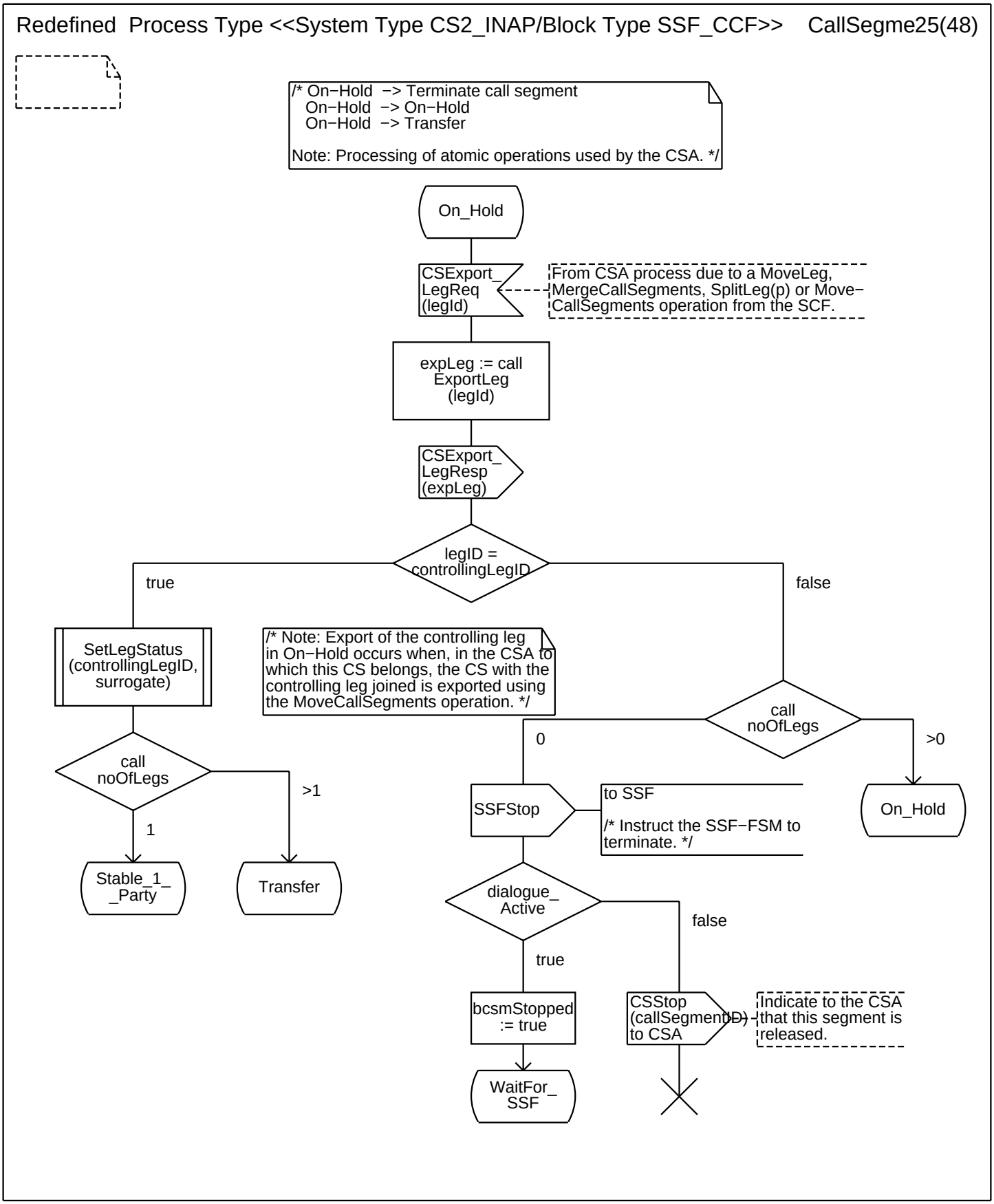

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme23(48)
/* Stable−2−Party −> On−Hold
Stable−2−Party −> 1−Party
Stable−2−Party −> Stable−Multi−Party
Note: Processing of atomic operations used by the CSA. */
Stable_2_ _Party
CSExport_ LegReq (legId)
From CSA process due to a SplitLeg, MergeCallSegments or MoveLeg operation from the SCF.
CSImport_ LegReq (legId,expLeg)
From CSA process due to a MergeCall_ Segments or MoveLeg operation from the SCF.
legId = controllingLegID
false
true
ImportLeg (expLeg,legId)
expLeg := call ExportLeg (legId)
expLeg := call ExportLeg (legId)
SetLegStatus (controllingLegID, shared)
CSExport_ LegResp (expLeg)
to CSA
CSExport_ LegResp (expLeg)
to CSA
CSImport_ LegResp
to CSA
On_Hold
TransitTo_ 1_Party
Stable_Multi_ _Party

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme24(48)

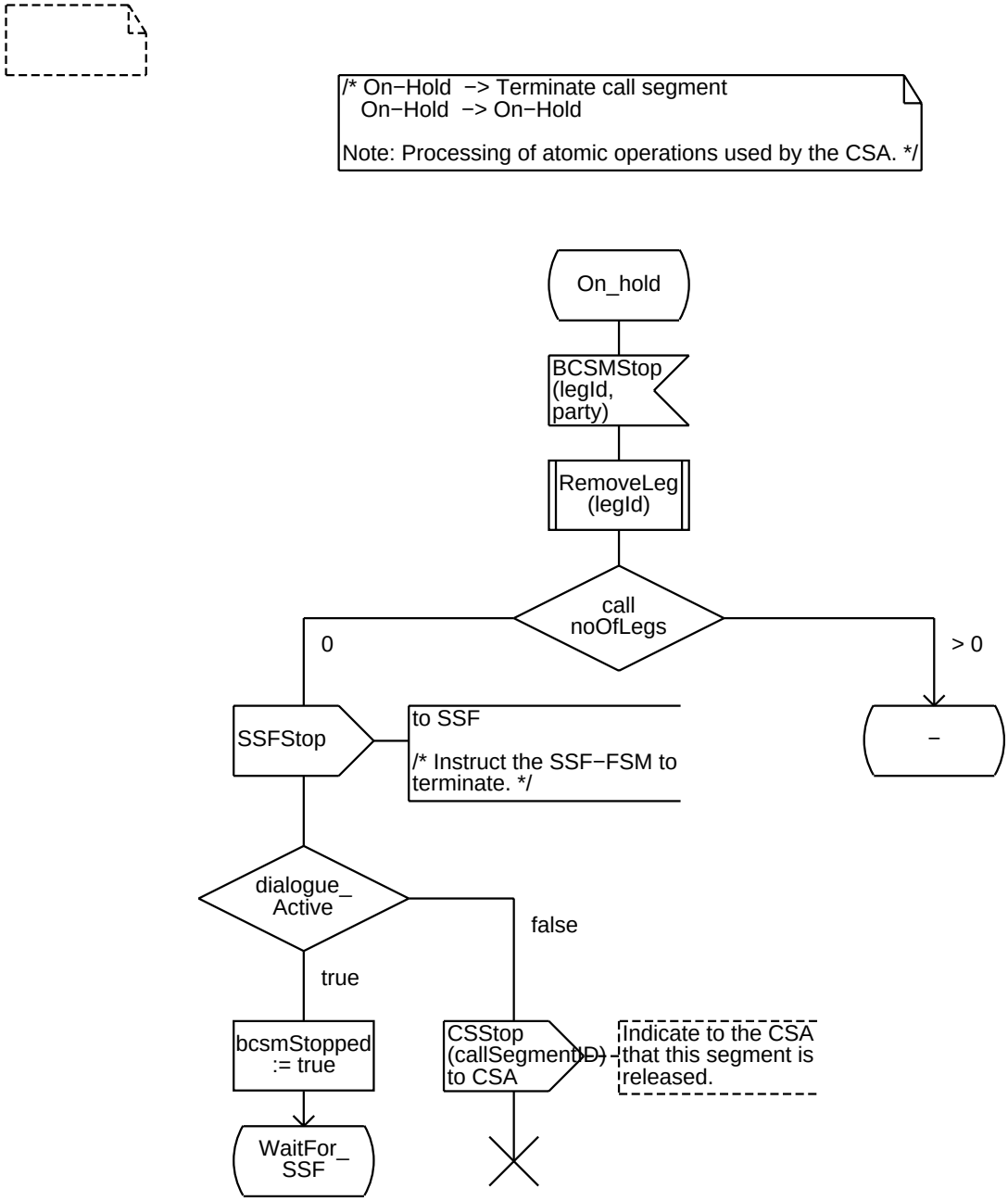


Annex A:

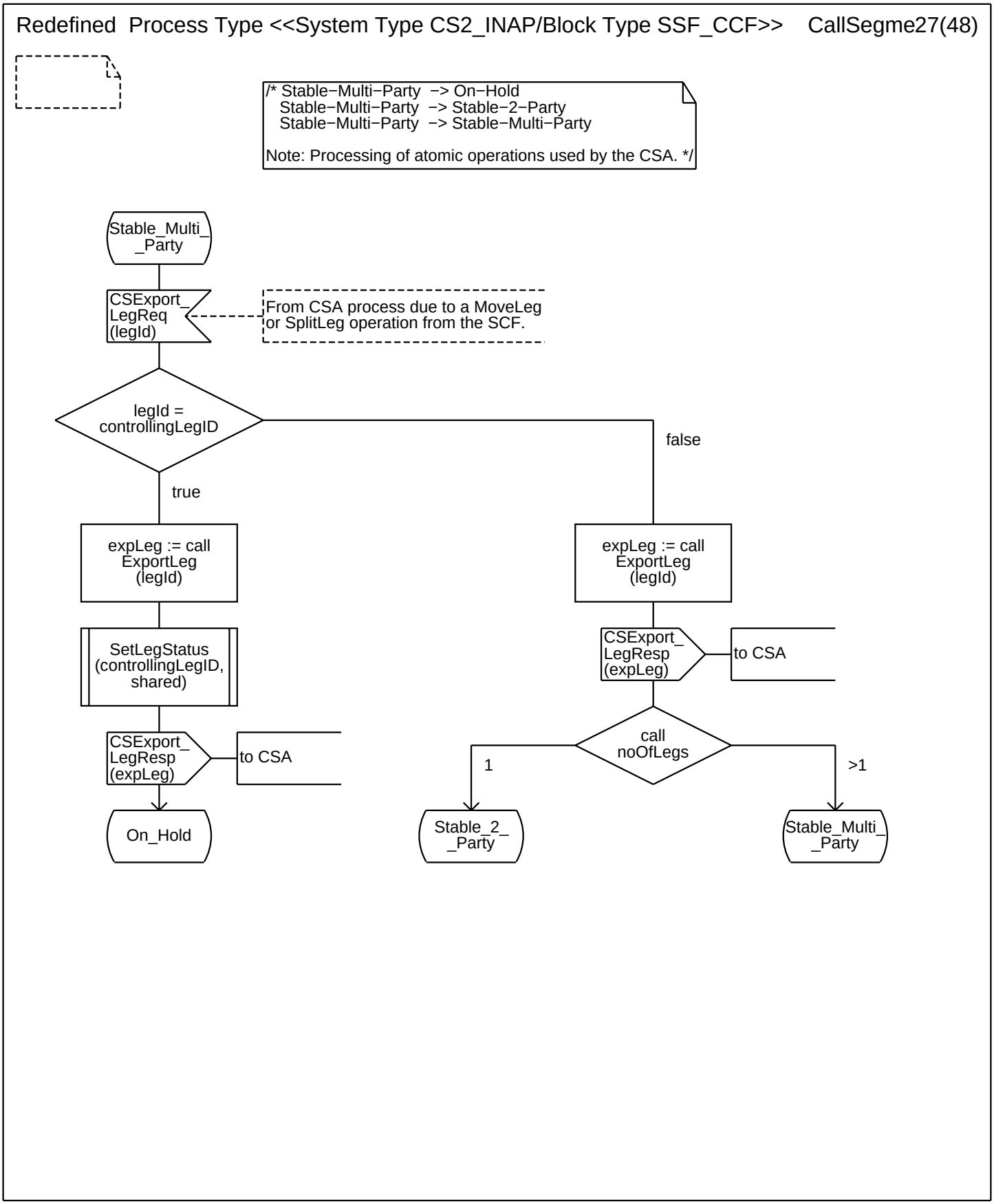

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme25(48)
/* On−Hold −> Terminate call segment
On−Hold −> On−Hold
On−Hold −> Transfer
Note: Processing of atomic operations used by the CSA. */
On_Hold
CSExport_ LegReq (legId)
From CSA process due to a MoveLeg, MergeCallSegments, SplitLeg(p) or Move−CallSegments operation from the SCF.
expLeg := call ExportLeg (legId)
CSExport_ LegResp (expLeg)
legID = controllingLegID
true
false
SetLegStatus (controllingLegID, surrogate)
/* Note: Export of the controlling leg in On−Hold occurs when, in the CSA to which this CS belongs, the CS with the controlling leg joined is exported using the MoveCallSegments operation. */
call noOfLegs
0
>0
call noOfLegs
>1
1
Stable_1_ _Party
Transfer
SSFStop
to SSF
/* Instruct the SSF−FSM to terminate. */
On_Hold
dialogue_ Active
false
true
bcsmStopped := true
CSStop (callSegmentID) to CSA
Indicate to the CSA that this segment is released.
WaitFor_ SSF

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme26(48)

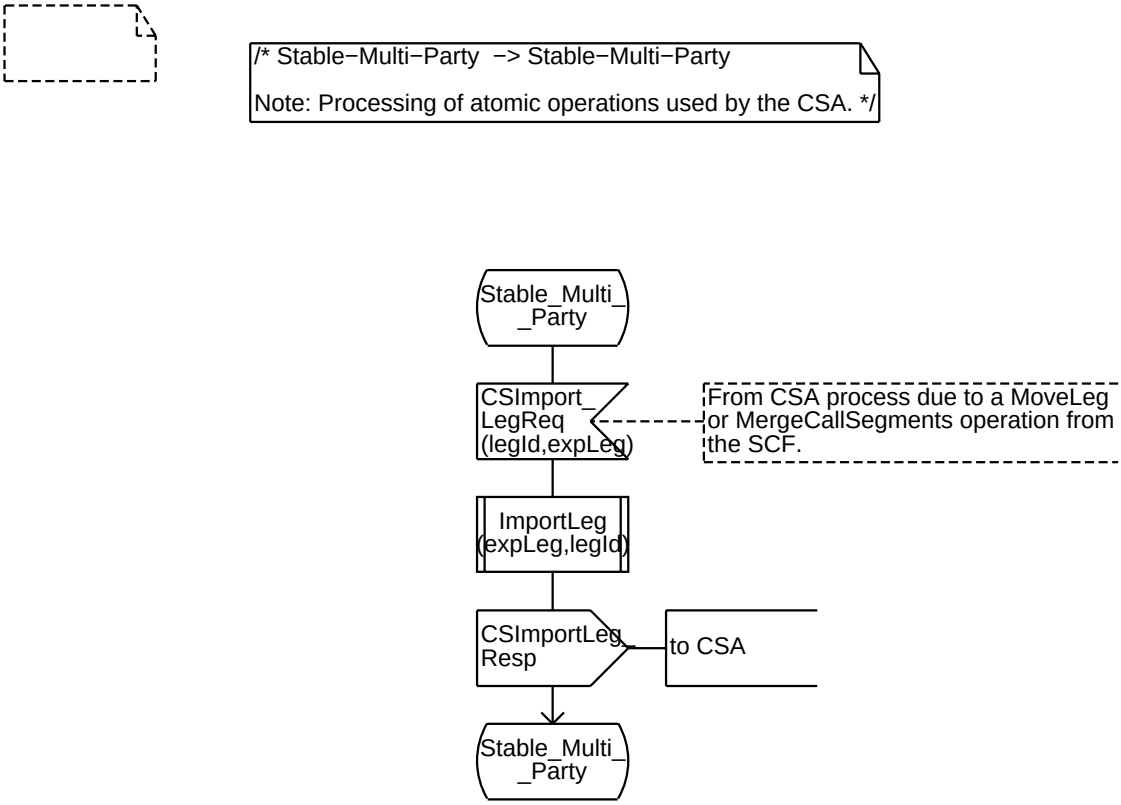


Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme27(48)
/* Stable−Multi−Party −> On−Hold
Stable−Multi−Party −> Stable−2−Party
Stable−Multi−Party −> Stable−Multi−Party
Note: Processing of atomic operations used by the CSA. */
Stable_Multi_ _Party
CSExport_ LegReq (legId)
From CSA process due to a MoveLeg or SplitLeg operation from the SCF.
legId = controllingLegID
false
true
expLeg := call ExportLeg (legId)
SetLegStatus (controllingLegID, shared)
CSExport_ LegResp (expLeg)
to CSA
On_Hold
expLeg := call ExportLeg (legId)
CSExport_ LegResp (expLeg)
to CSA
call noOfLegs
1
>1
Stable_2_ _Party
Stable_Multi_ _Party

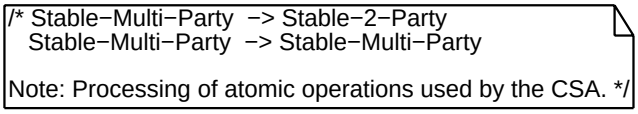
Annex A:

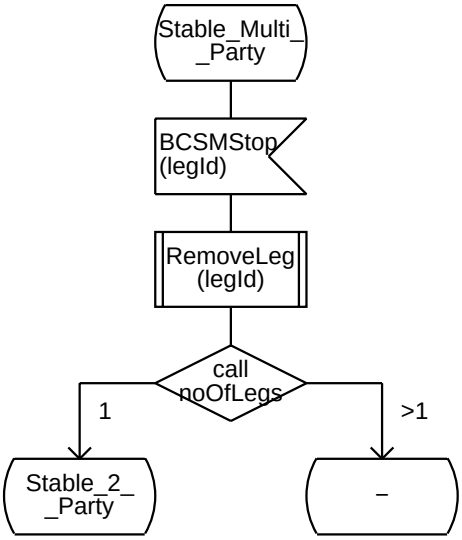
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme28(48)



Annex A:

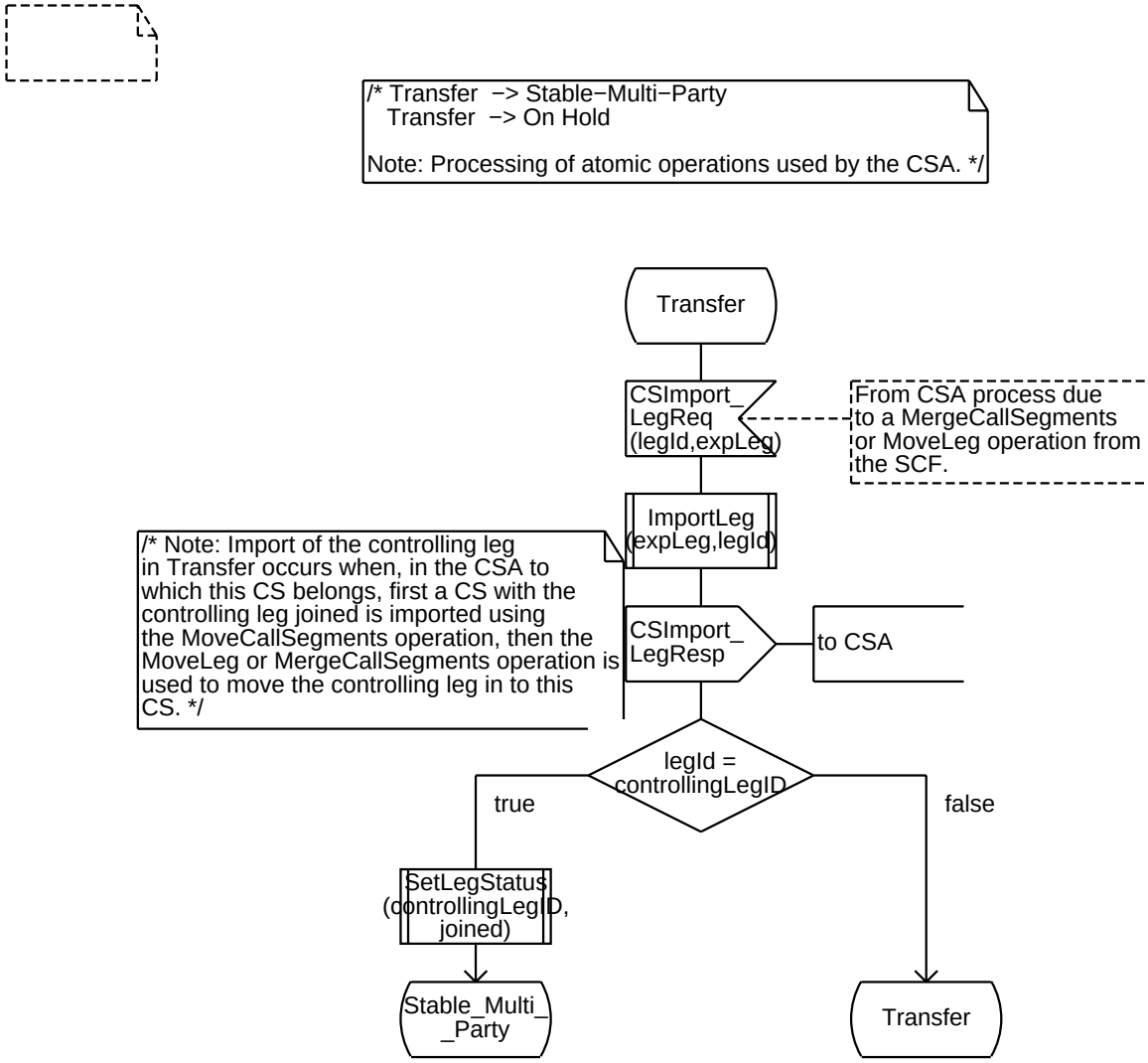
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme29(48)



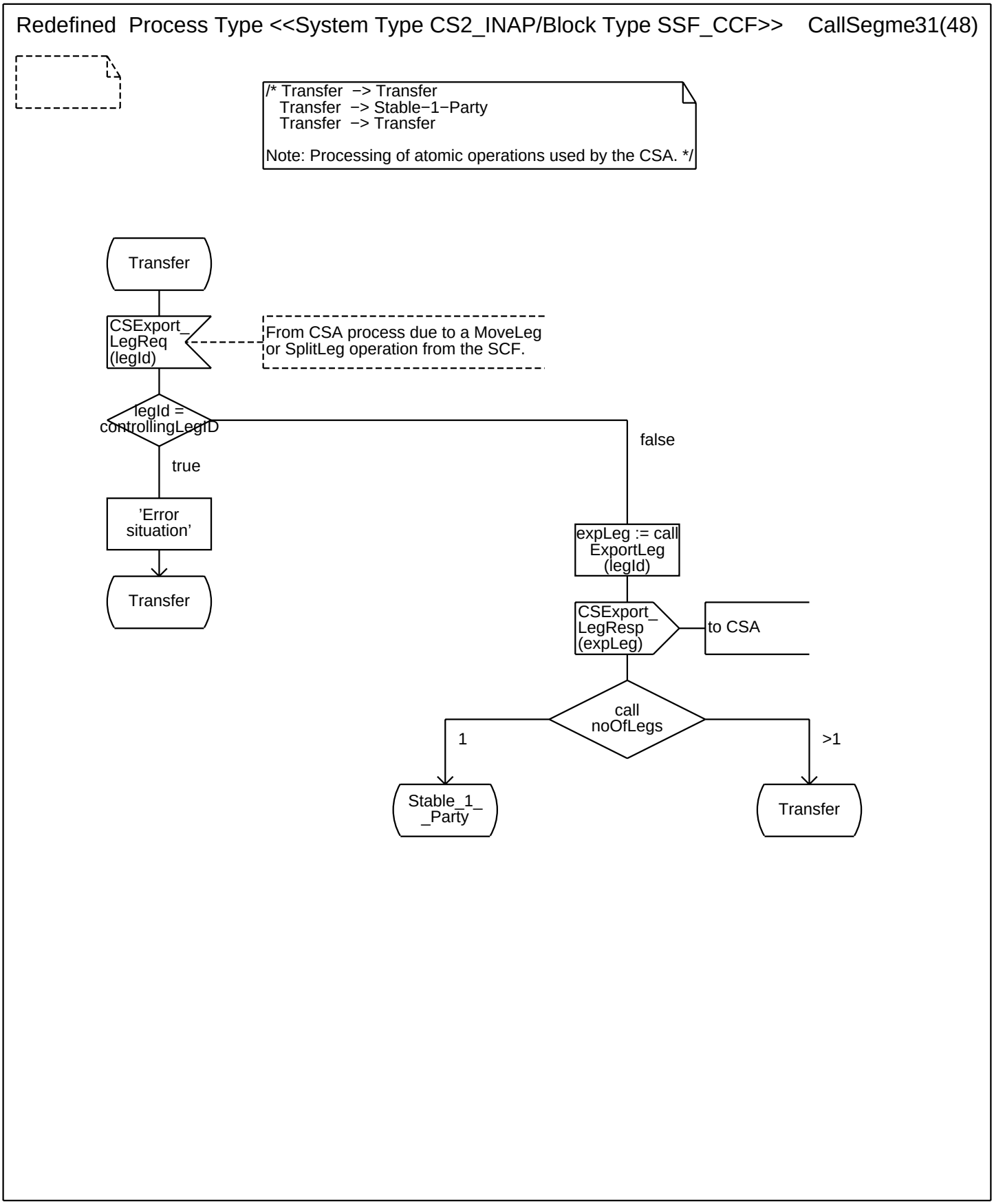


Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme30(48)

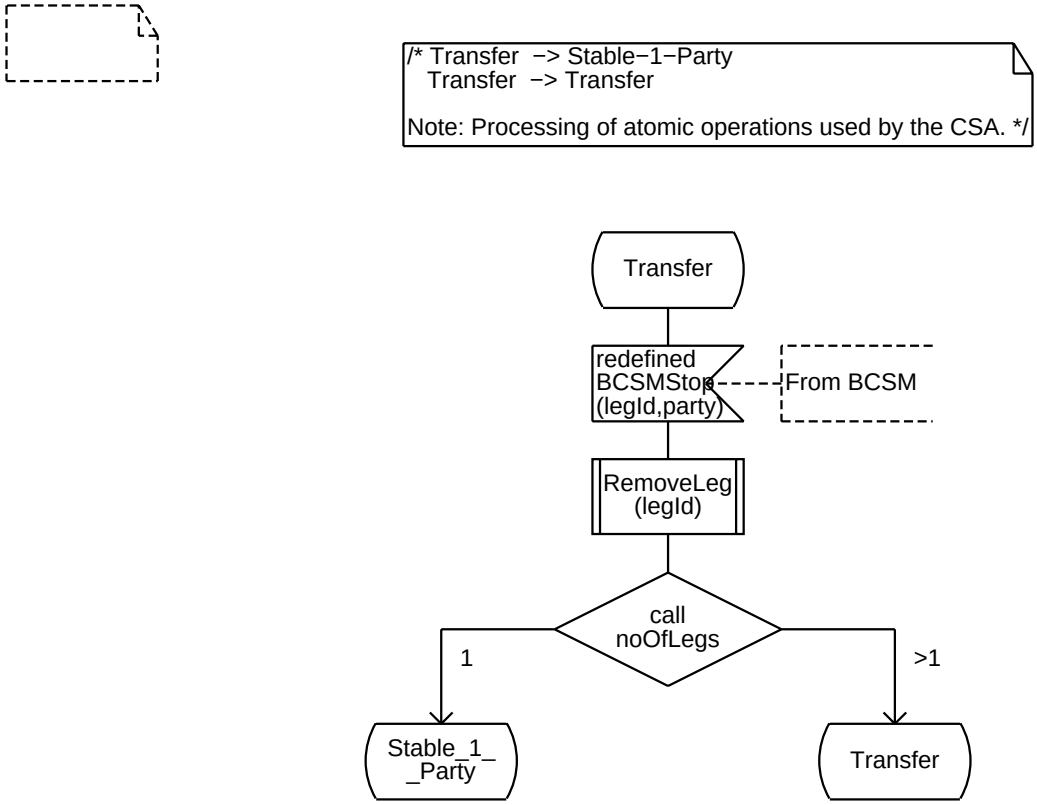


Annex A:

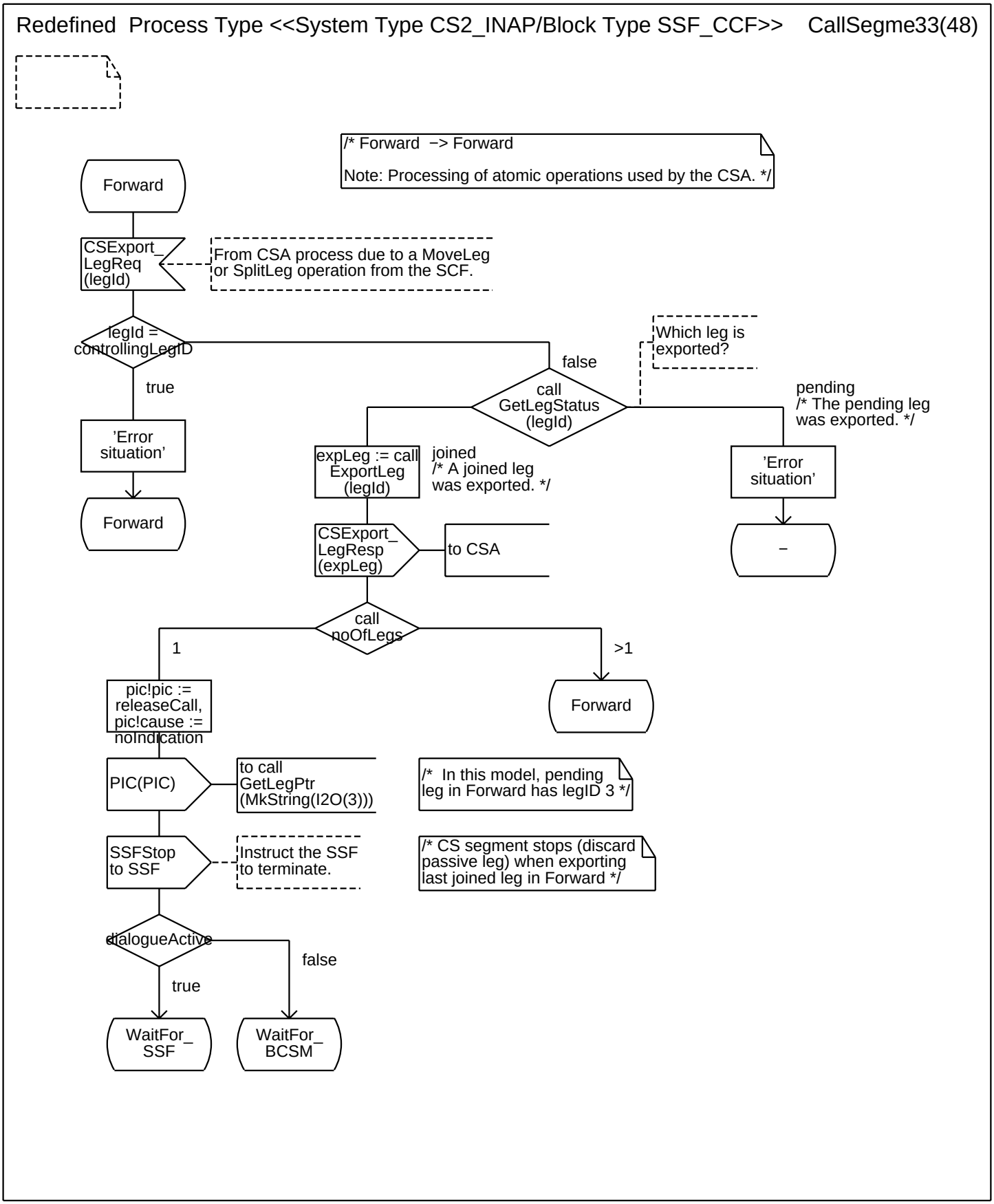

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme31(48)
/* Transfer −> Transfer
Transfer −> Stable−1−Party
Transfer −> Transfer
Note: Processing of atomic operations used by the CSA. */
Transfer
CSExport_ LegReq (legId)
From CSA process due to a MoveLeg or SplitLeg operation from the SCF.
legId = controllingLegID
true
false
'Error situation'
Transfer
expLeg := call ExportLeg (legId)
CSExport_ LegResp (expLeg)
to CSA
call noOfLegs
1
>1
Stable_1_ _Party
Transfer

Annex A:

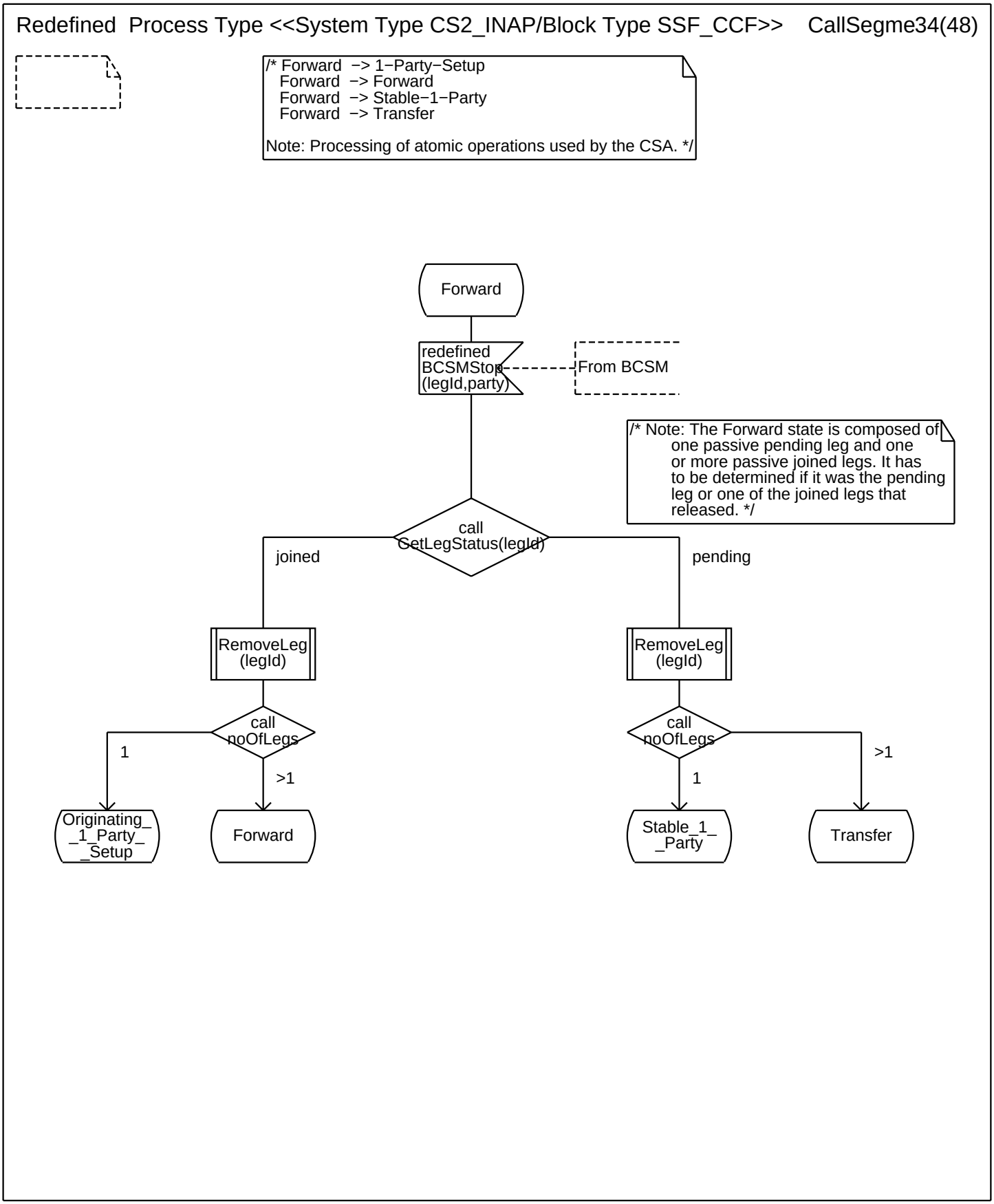
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme32(48)



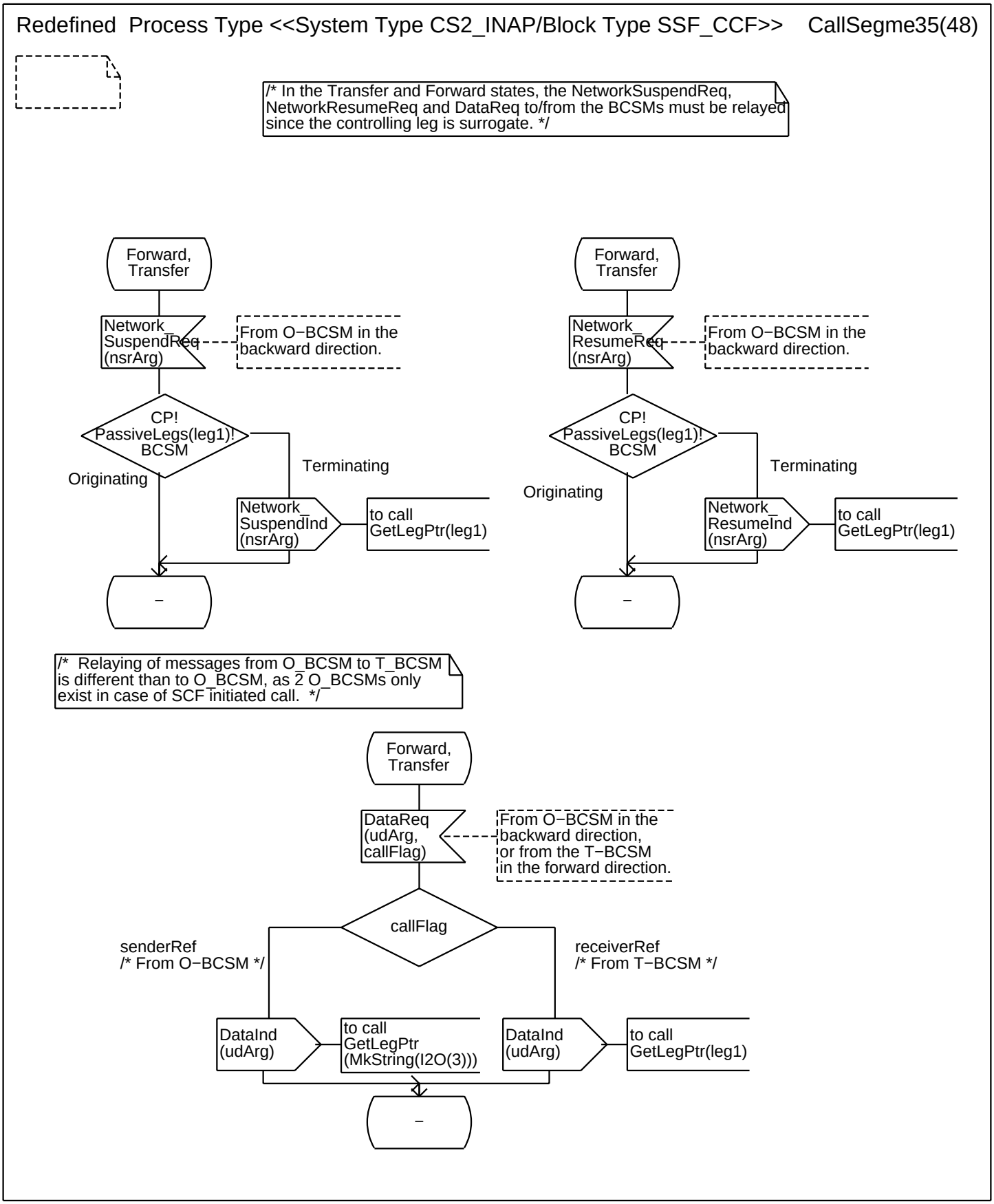
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme33(48)
/* Forward −> Forward
Note: Processing of atomic operations used by the CSA. */
Forward
CSExport_ LegReq (legId)
From CSA process due to a MoveLeg or SplitLeg operation from the SCF.
legId = controllingLegID
true
'Error situation'
Forward
false
call GetLegStatus (legId)
Which leg is exported?
pending
/* The pending leg was exported. */
'Error situation'
–
expLeg := call ExportLeg (legId)
joined
/* A joined leg was exported. */
CSExport_ LegResp (expLeg)
to CSA
call noOfLegs
1
>1
Forward
pic!pic := releaseCall, pic!cause := noIndication
PIC(PIC)
to call GetLegPtr (MkString(I2O(3)))
/* In this model, pending leg in Forward has legID 3 */
SSFStop to SSF
Instruct the SSF to terminate.
/* CS segment stops (discard passive leg) when exporting last joined leg in Forward */
dialogueActive
true
false
WaitFor_ SSF
WaitFor_ BCSM

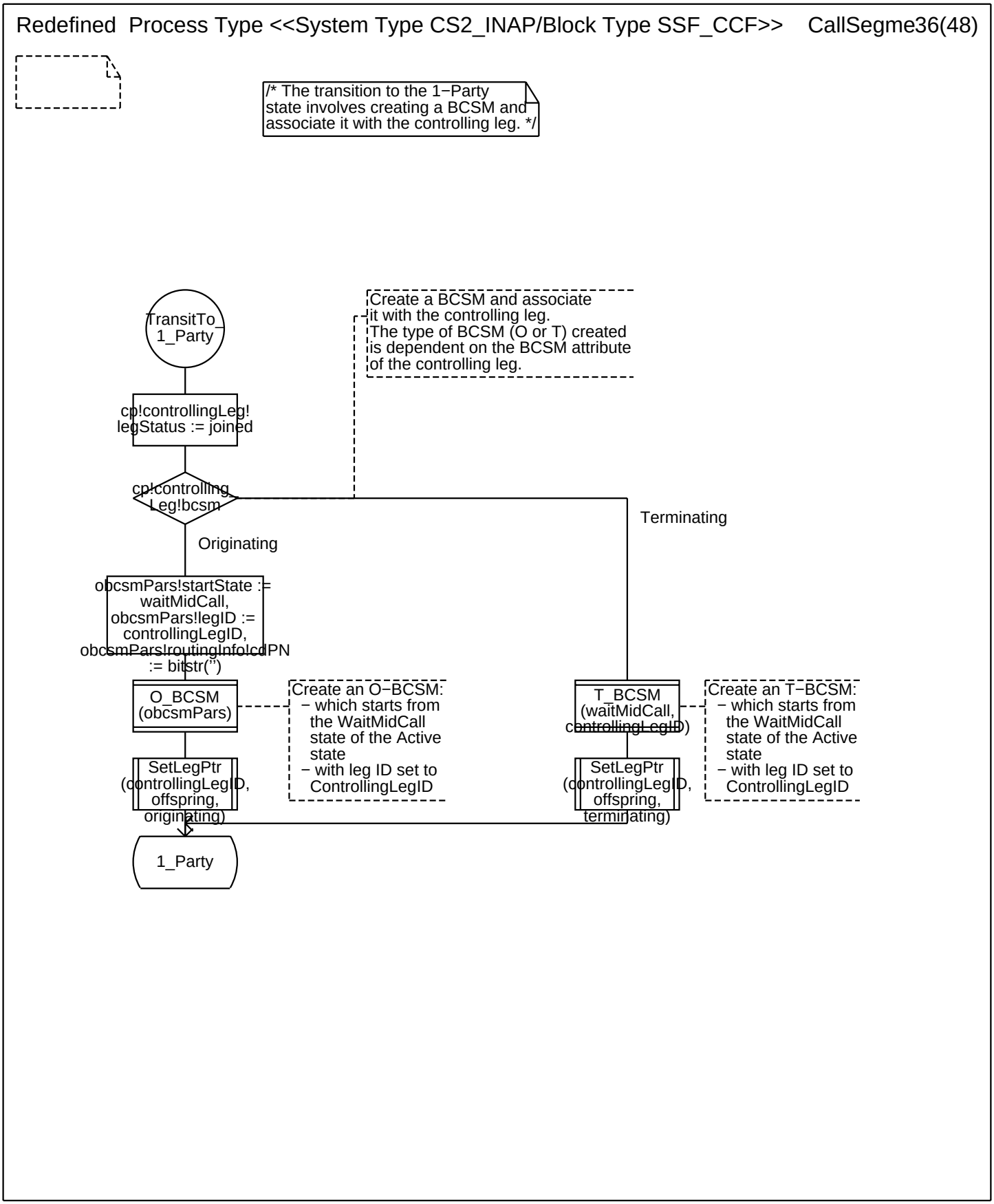
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme34(48)
/* Forward −> 1−Party−Setup
Forward −> Forward
Forward −> Stable−1−Party
Forward −> Transfer
Note: Processing of atomic operations used by the CSA. */
Forward
redefined BCSMStop (legId,party)
From BCSM
/* Note: The Forward state is composed of one passive pending leg and one or more passive joined legs. It has to be determined if it was the pending leg or one of the joined legs that released. */
call GetLegStatus(legId)
joined
pending
RemoveLeg (legId)
RemoveLeg (legId)
call noOfLegs
call noOfLegs
1
>1
1
>1
Originating_ _1_Party_ _Setup
Forward
Stable_1_ _Party
Transfer

Annex A:

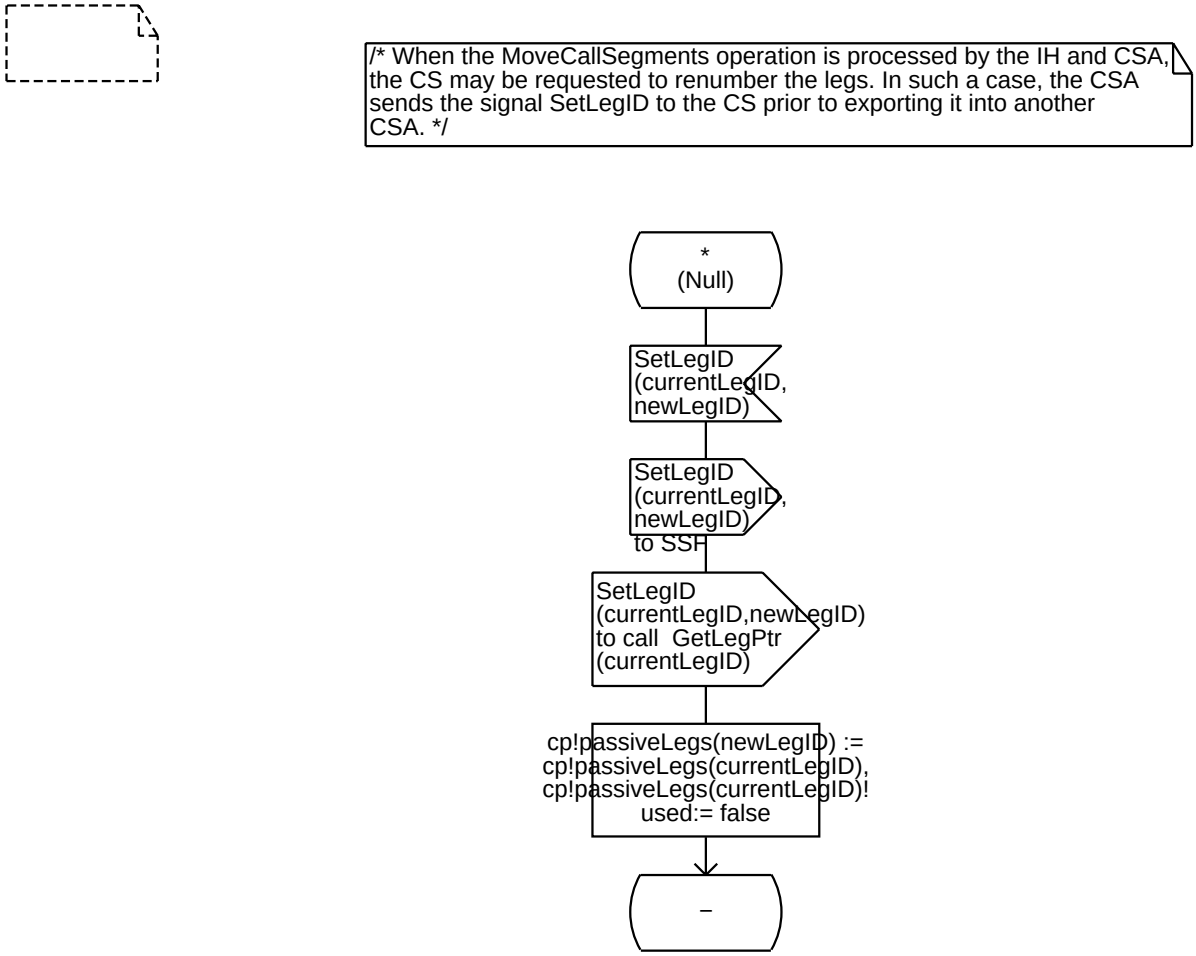

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme35(48)
/* In the Transfer and Forward states, the NetworkSuspendReq, NetworkResumeReq and DataReq to/from the BCSMs must be relayed since the controlling leg is surrogate. */
Forward, Transfer
Network_ SuspendReq (nsrArg)
From O−BCSM in the backward direction.
CP! PassiveLegs(leg1)! BCSM
Originating
Terminating
Network_ SuspendInd (nsrArg)
to call GetLegPtr(leg1)
−
Forward, Transfer
Network_ ResumeReq (nsrArg)
From O−BCSM in the backward direction.
CP! PassiveLegs(leg1)! BCSM
Originating
Terminating
Network_ ResumeInd (nsrArg)
to call GetLegPtr(leg1)
−
/* Relaying of messages from O_BCSM to T_BCSM is different than to O_BCSM, as 2 O_BCSMs only exist in case of SCF initiated call. */
Forward, Transfer
DataReq (udArg, callFlag)
From O−BCSM in the backward direction, or from the T−BCSM in the forward direction.
callFlag
senderRef /* From O−BCSM */
receiverRef /* From T−BCSM */
DataInd (udArg)
to call GetLegPtr (MkString(I2O(3)))
DataInd (udArg)
to call GetLegPtr(leg1)
−

Annex A:

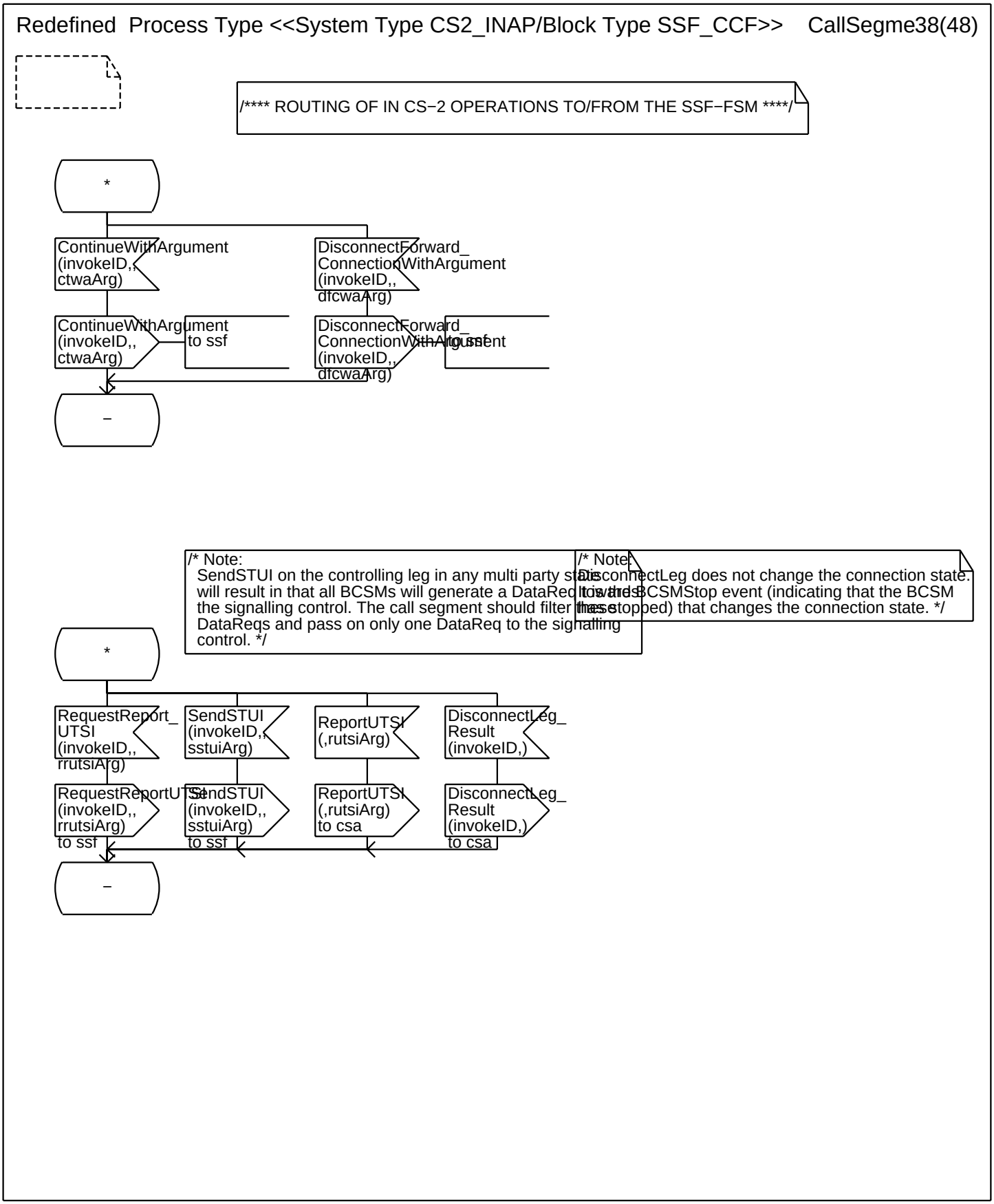

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme36(48)
/* The transition to the 1−Party state involves creating a BCSM and associate it with the controlling leg. */
TransitTo_1_Party
Create a BCSM and associate it with the controlling leg. The type of BCSM (O or T) created is dependent on the BCSM attribute of the controlling leg.
cp!controllingLeg!legStatus := joined
cp!controllingLeg!bcsm
Terminating
Originating
obcsmPars!startState := waitMidCall, obcsmPars!legID := controllingLegID, obcsmPars!routingInfo!cdPN := bitstr(")
O_BCSM (obcsmPars)
Create an O−BCSM: − which starts from the WaitMidCall state of the Active state − with leg ID set to ControllingLegID
T_BCSM (waitMidCall, controllingLegID)
Create an T−BCSM: − which starts from the WaitMidCall state of the Active state − with leg ID set to ControllingLegID
SetLegPtr (controllingLegID, offspring, originating)
SetLegPtr (controllingLegID, offspring, terminating)
1_Party

Annex A:

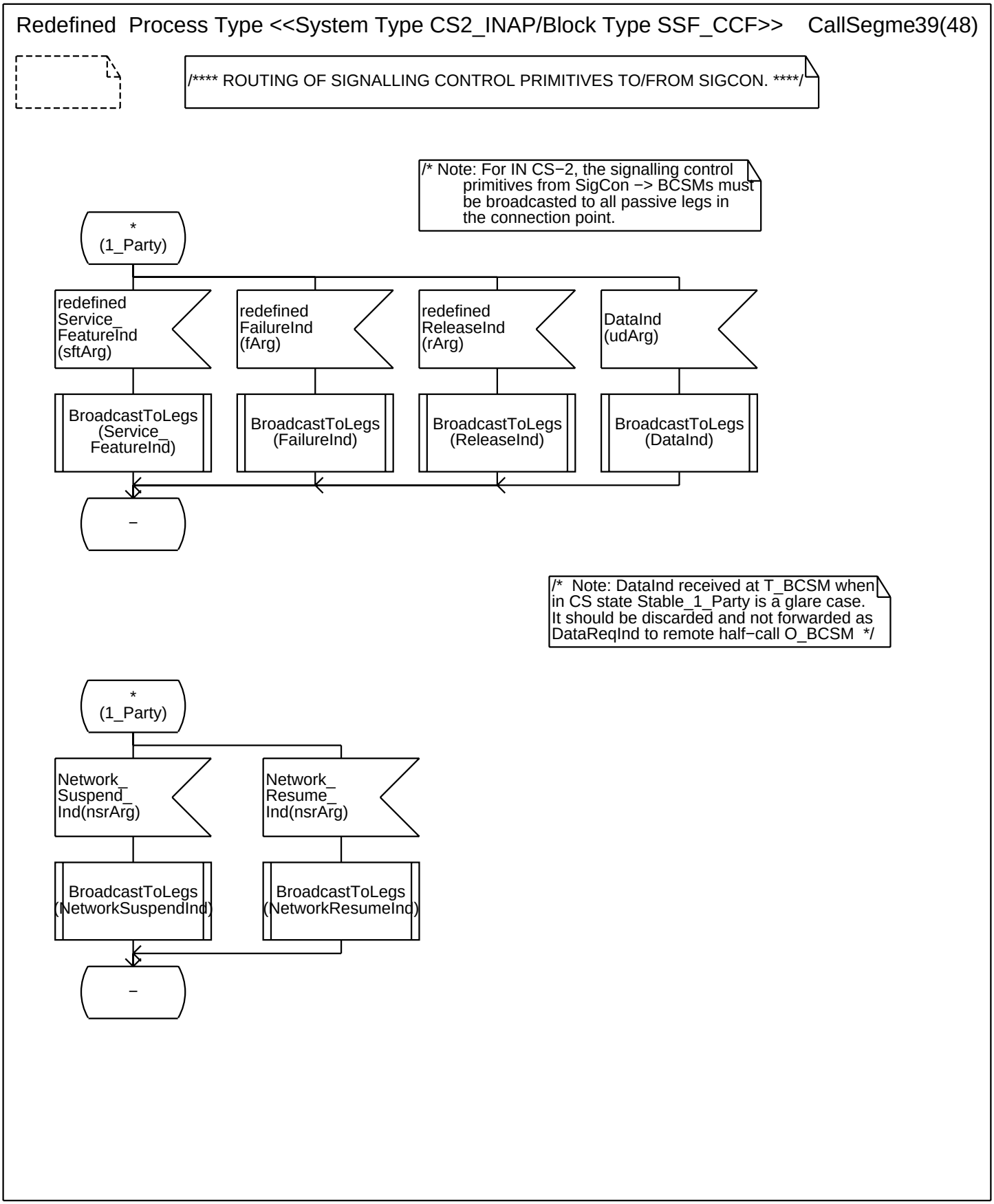
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme37(48)



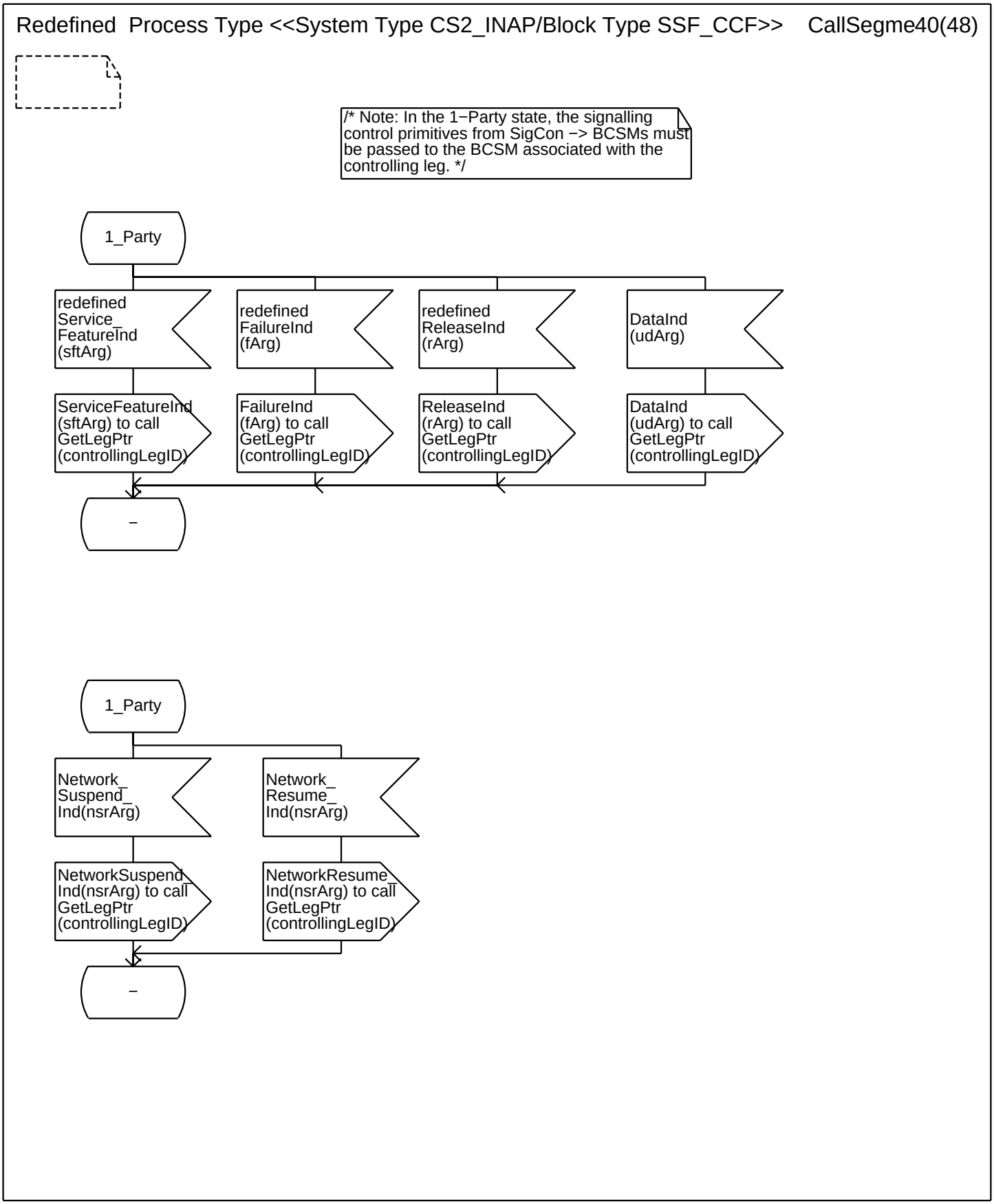
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme38(48)
/**** ROUTING OF IN CS−2 OPERATIONS TO/FROM THE SSF−FSM ****/
*
ContinueWithArgument (invokeID,, ctwaArg)
DisconnectForward_ ConnectionWithArgument (invokeID,, dfcwaArg)
ContinueWithArgument (invokeID,, ctwaArg)
to ssf
DisconnectForward_ ConnectionWithArgument (invokeID,, dfcwaArg)
to ssf
−
/* Note:
SendSTUI on the controlling leg in any multi party state will result in that all BCSMs will generate a DataReq towards the signalling control. The call segment should filter these DataReqs and pass on only one DataReq to the signalling control. */
/* Note:
DisconnectLeg does not change the connection state. It will result in a BCSMStop event (indicating that the BCSM has stopped) that changes the connection state. */
*
RequestReport_ UTSI (invokeID,, rrutsiArg)
SendSTUI (invokeID,, sstuiArg)
ReportUTSI (,rutsiArg)
DisconnectLeg_ Result (invokeID,)
RequestReportUTSI (invokeID,, rrutsiArg)
to ssf
SendSTUI (invokeID,, sstuiArg)
to ssf
ReportUTSI (,rutsiArg)
to csa
DisconnectLeg_ Result (invokeID,)
to csa
−

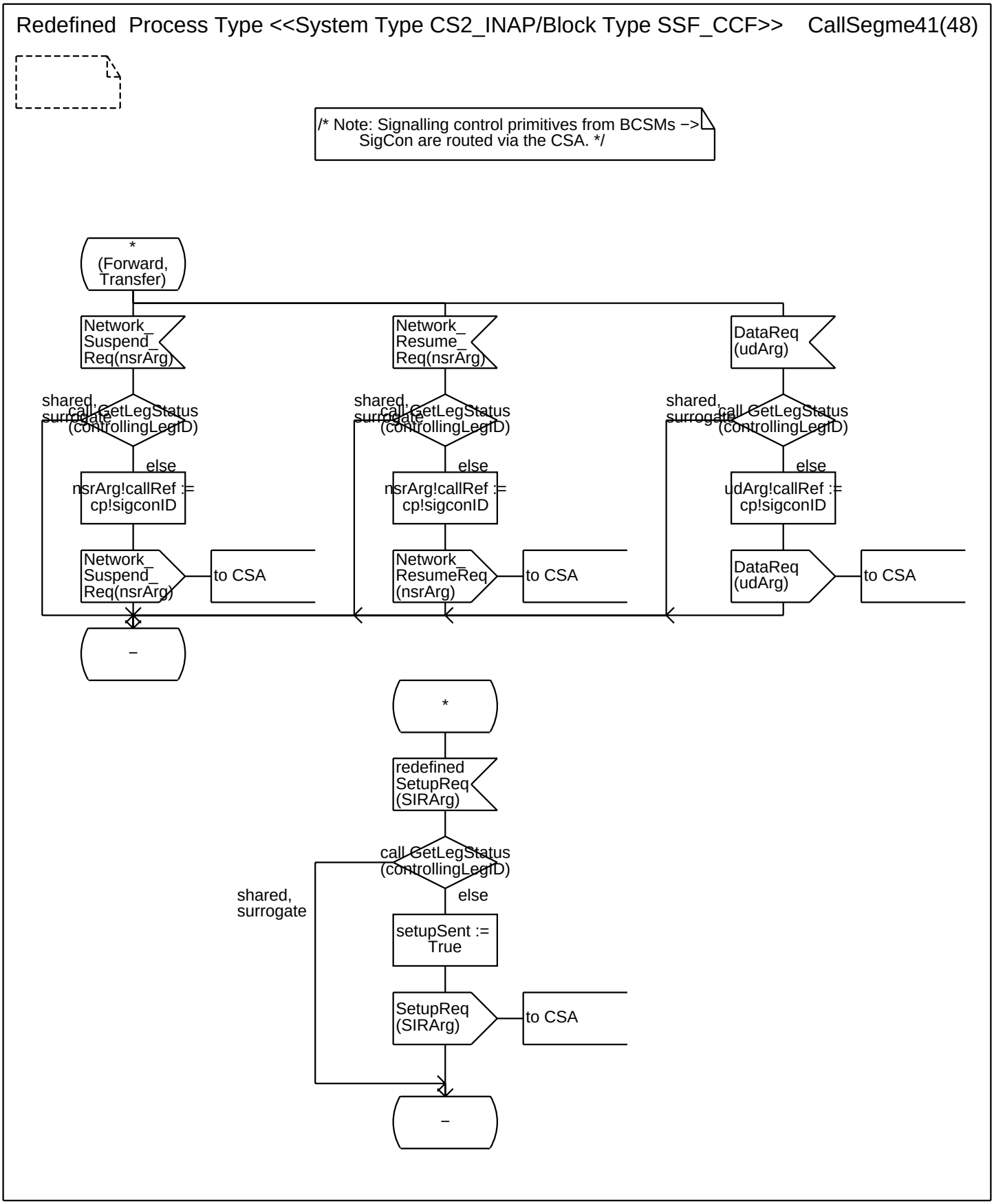
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme39(48)
/**** ROUTING OF SIGNALLING CONTROL PRIMITIVES TO/FROM SIGCON. ****/
/* Note: For IN CS−2, the signalling control primitives from SigCon −> BCSMs must be broadcasted to all passive legs in the connection point.
*
(1_Party)
redefined Service_ FeatureInd (sftArg)
redefined FailureInd (fArg)
redefined ReleaseInd (rArg)
DataInd (udArg)
BroadcastToLegs (Service_ FeatureInd)
BroadcastToLegs (FailureInd)
BroadcastToLegs (ReleaseInd)
BroadcastToLegs (DataInd)
−
/* Note: DataInd received at T_BCSM when in CS state Stable_1_Party is a glare case. It should be discarded and not forwarded as DataReqInd to remote half−call O_BCSM */
*
(1_Party)
Network_ Suspend_ Ind(nsrArg)
Network_ Resume_ Ind(nsrArg)
BroadcastToLegs (NetworkSuspendInd)
BroadcastToLegs (NetworkResumeInd)
−

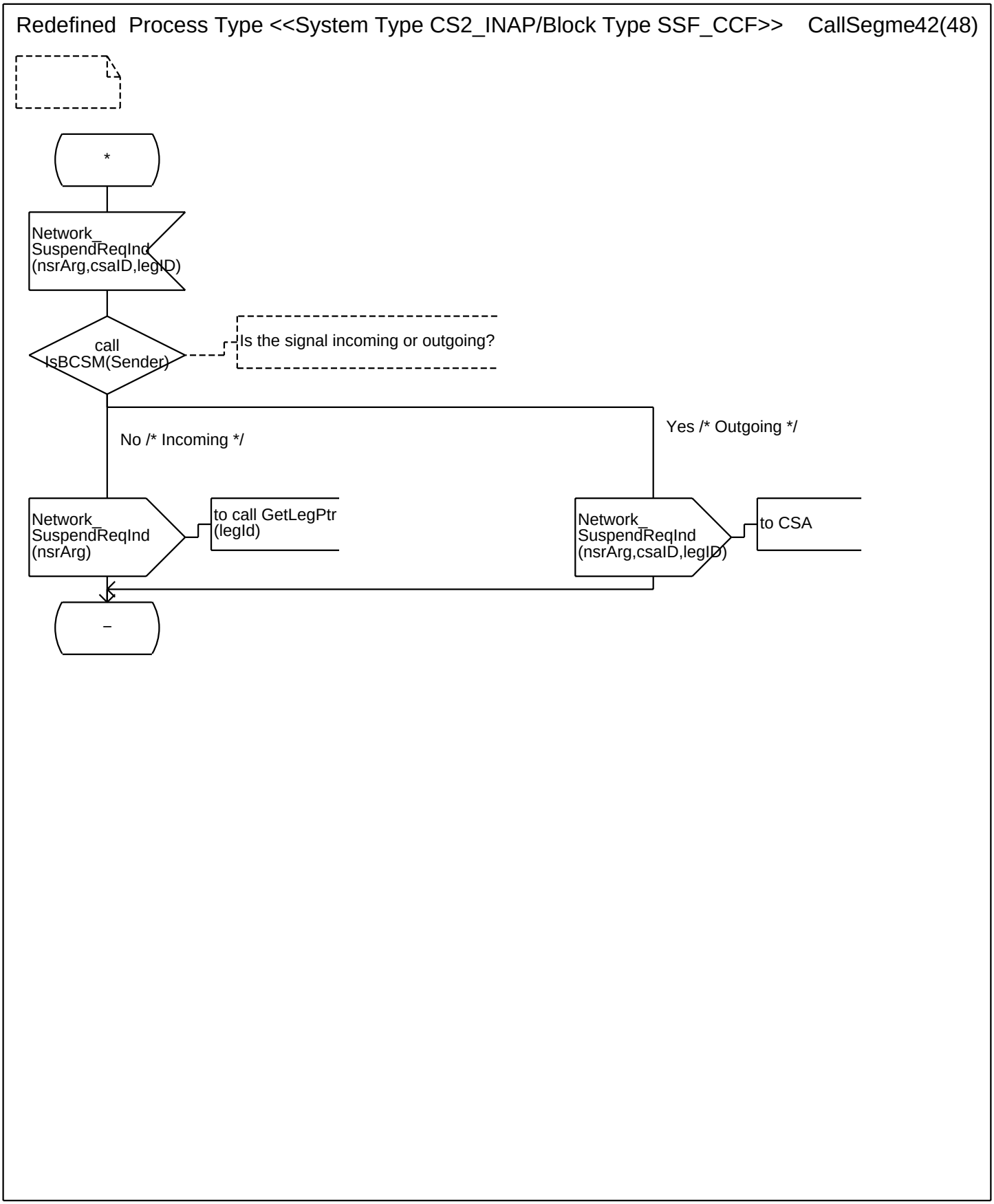
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme40(48)
/* Note: In the 1−Party state, the signalling control primitives from SigCon −> BCSMs must be passed to the BCSM associated with the controlling leg. */
1_Party
redefined Service_ FeatureInd (sftArg)
redefined FailureInd (fArg)
redefined ReleaseInd (rArg)
DataInd (udArg)
ServiceFeatureInd (sftArg) to call GetLegPtr (controllingLegID)
FailureInd (fArg) to call GetLegPtr (controllingLegID)
ReleaseInd (rArg) to call GetLegPtr (controllingLegID)
DataInd (udArg) to call GetLegPtr (controllingLegID)
–
1_Party
Network_ Suspend_ Ind(nsrArg)
Network_ Resume_ Ind(nsrArg)
NetworkSuspend_ Ind(nsrArg) to call GetLegPtr (controllingLegID)
NetworkResume_ Ind(nsrArg) to call GetLegPtr (controllingLegID)
–

Annex A:

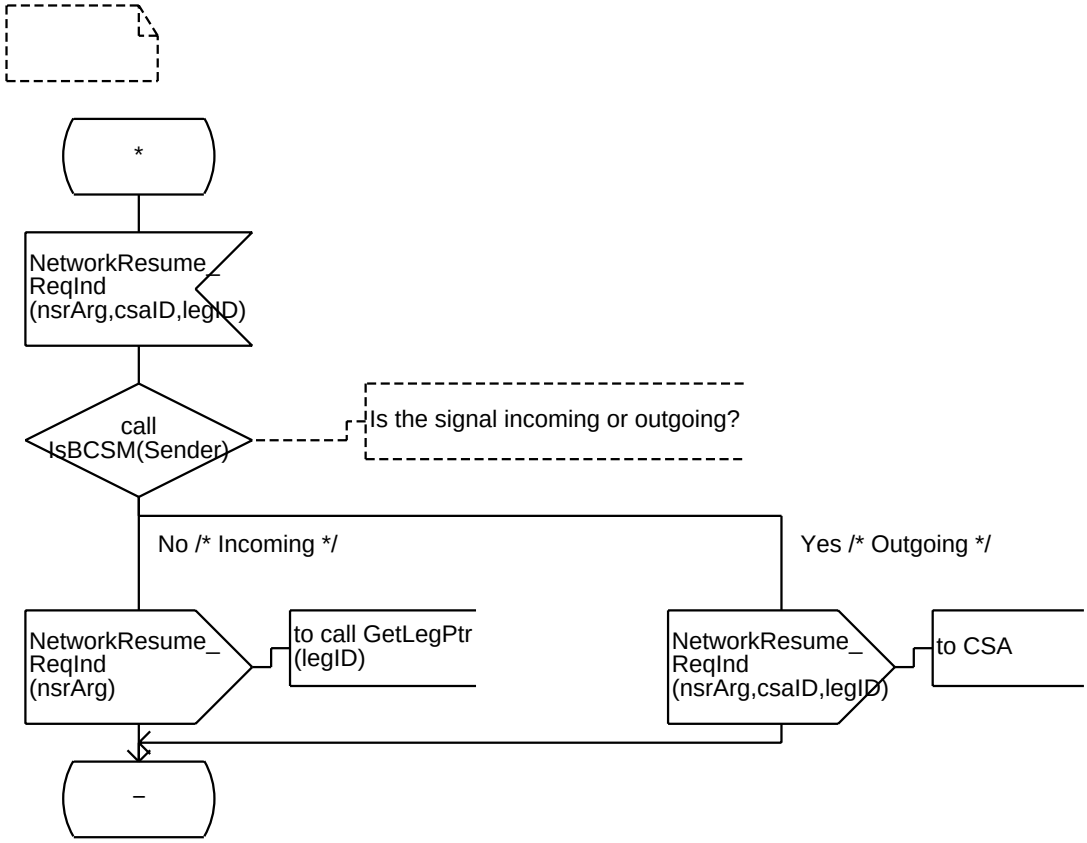

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme41(48)
/* Note: Signalling control primitives from BCSMs −> SigCon are routed via the CSA. */
*
(Forward, Transfer)
Network_ Suspend_ Req(nsrArg)
Network_ Resume_ Req(nsrArg)
DataReq (udArg)
shared, surrogate
call GetLegStatus (controllingLegID)
else
nsrArg!callRef := cp!sigconID
udArg!callRef := cp!sigconID
Network_ Suspend_ Req(nsrArg)
Network_ ResumeReq (nsrArg)
DataReq (udArg)
to CSA
−
*
redefined SetupReq (SIRArg)
call GetLegStatus (controllingLegID)
shared, surrogate
else
setupSent := True
SetupReq (SIRArg)
to CSA
−

Annex A:

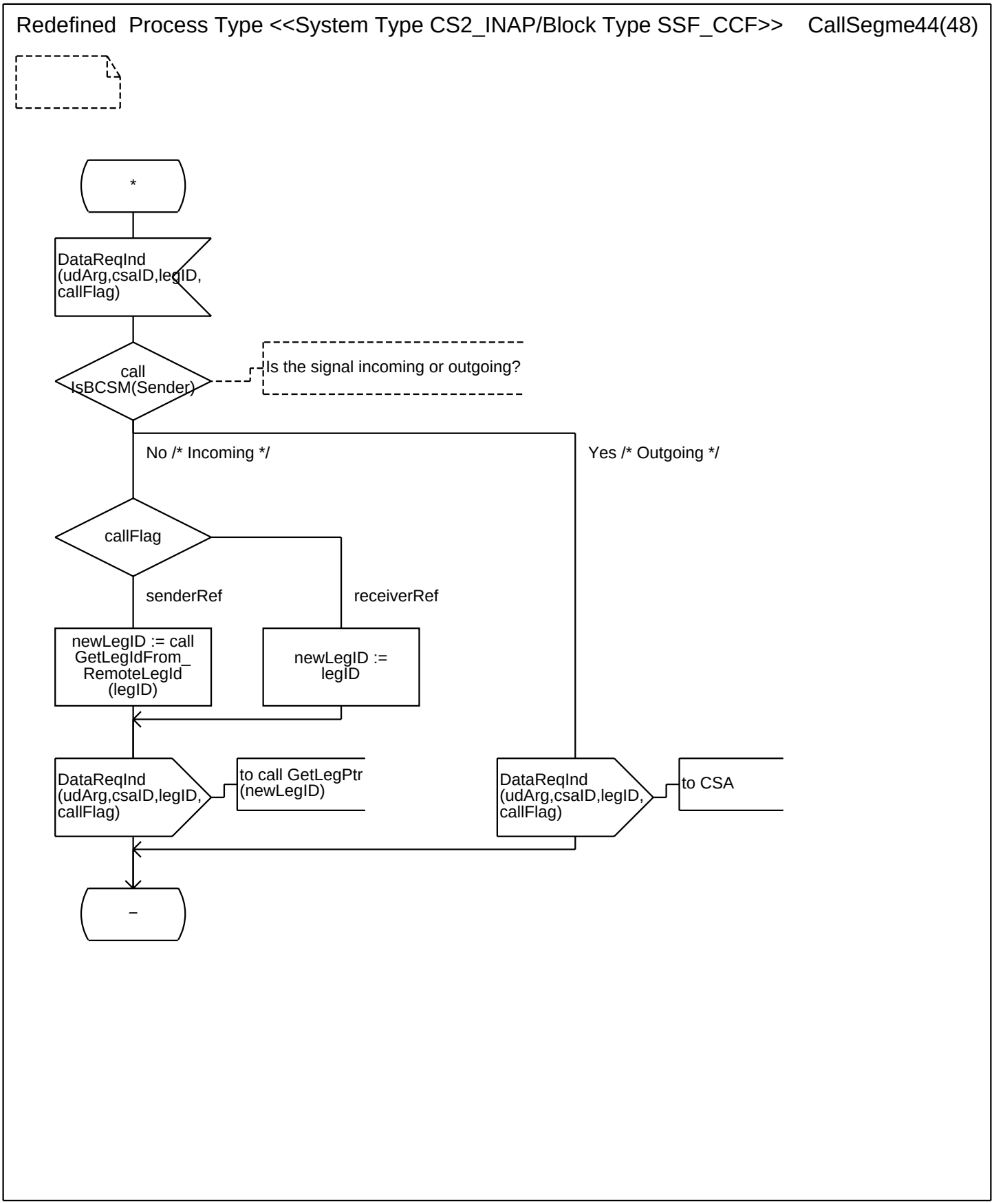

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme42(48)
*
Network_
SuspendReqInd
(nsrArg,csaID,legID)
call
IsBCSM(Sender)
Is the signal incoming or outgoing?
No /* Incoming */
Yes /* Outgoing */
Network_
SuspendReqInd
(nsrArg)
to call GetLegPtr
(legId)
Network_
SuspendReqInd
(nsrArg,csaID,legID)
to CSA
–

Annex A:

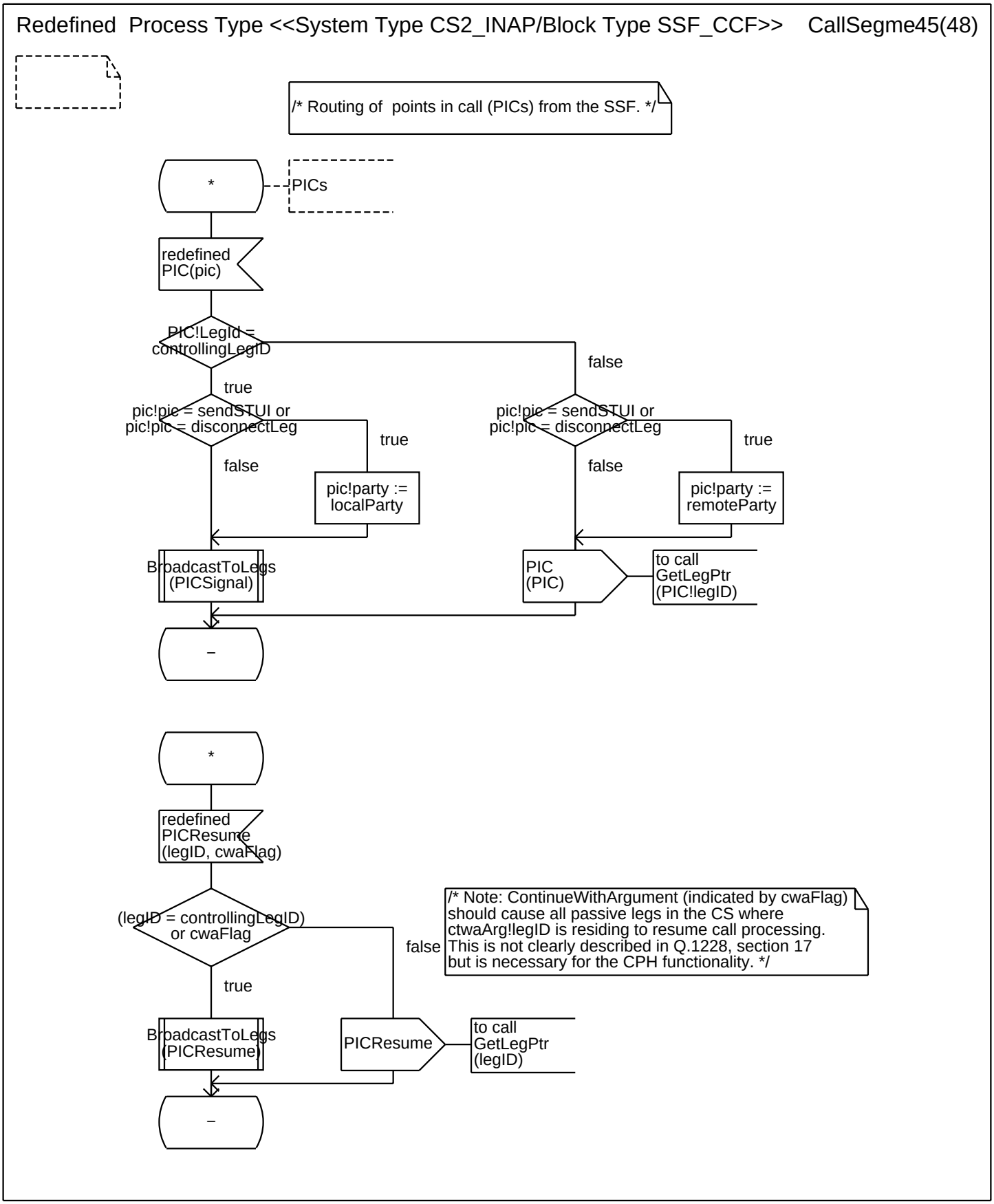
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme43(48)



Annex A:

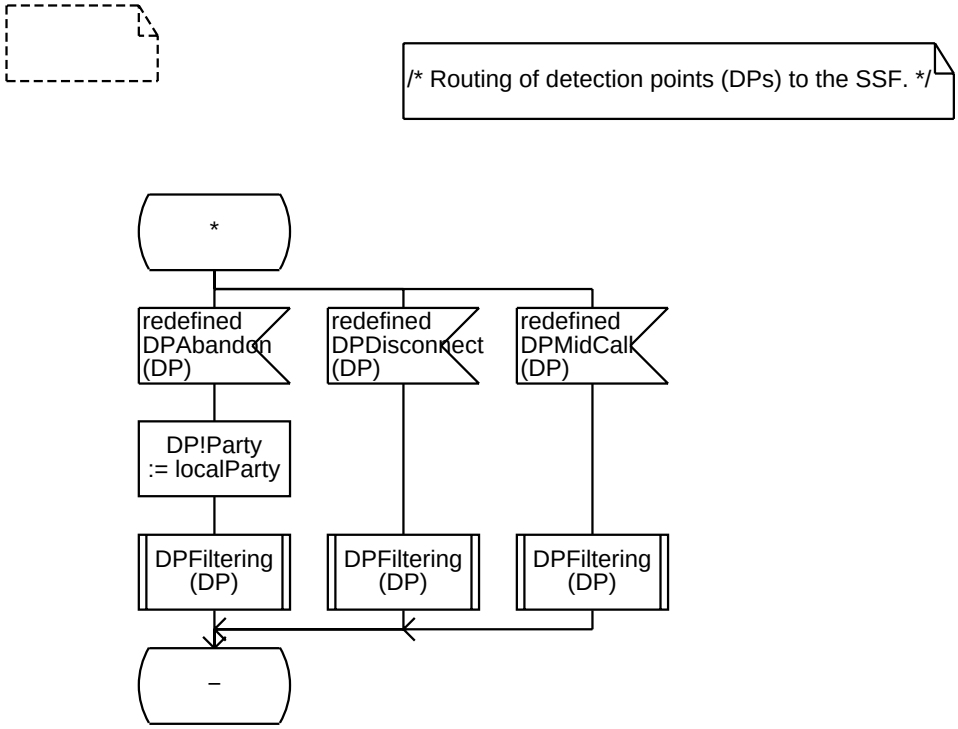

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme44(48)
*
DataReqInd (udArg,csaID,legID, callFlag)
call IsBCSM(Sender)
Is the signal incoming or outgoing?
No /* Incoming */
Yes /* Outgoing */
callFlag
senderRef
receiverRef
newLegID := call GetLegIdFrom_ RemoteLegId (legID)
newLegID := legID
DataReqInd (udArg,csaID,legID, callFlag)
to call GetLegPtr (newLegID)
DataReqInd (udArg,csaID,legID, callFlag)
to CSA
–

Annex A:

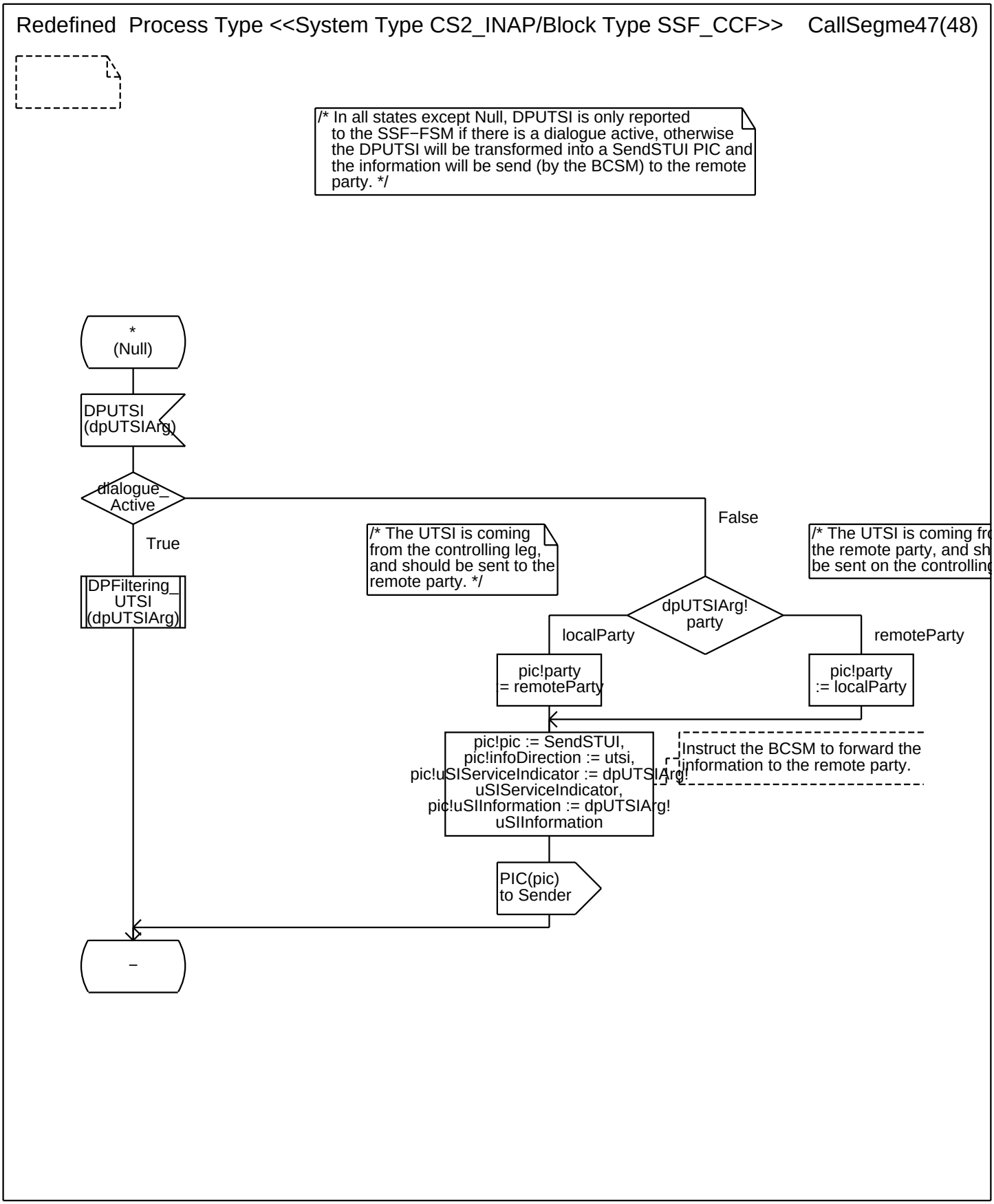

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme45(48)
/* Routing of points in call (PICs) from the SSF. */
*
PICs
redefined PIC(pic)
PIC!LegId = controllingLegID
false
true
pic!pic = sendSTUI or pic!pic = disconnectLeg
true
false
pic!party := localParty
pic!pic = sendSTUI or pic!pic = disconnectLeg
true
false
pic!party := remoteParty
BroadcastToLegs (PICSignal)
PIC (PIC)
to call GetLegPtr (PIC!legID)
–
*
redefined PICResume (legID, cwaFlag)
(legID = controllingLegID) or cwaFlag
false
/* Note: ContinueWithArgument (indicated by cwaFlag) should cause all passive legs in the CS where ctwaArg!legID is residing to resume call processing. This is not clearly described in Q.1228, section 17 but is necessary for the CPH functionality. */
true
BroadcastToLegs (PICResume)
PICResume
to call GetLegPtr (legID)
–

Annex A:

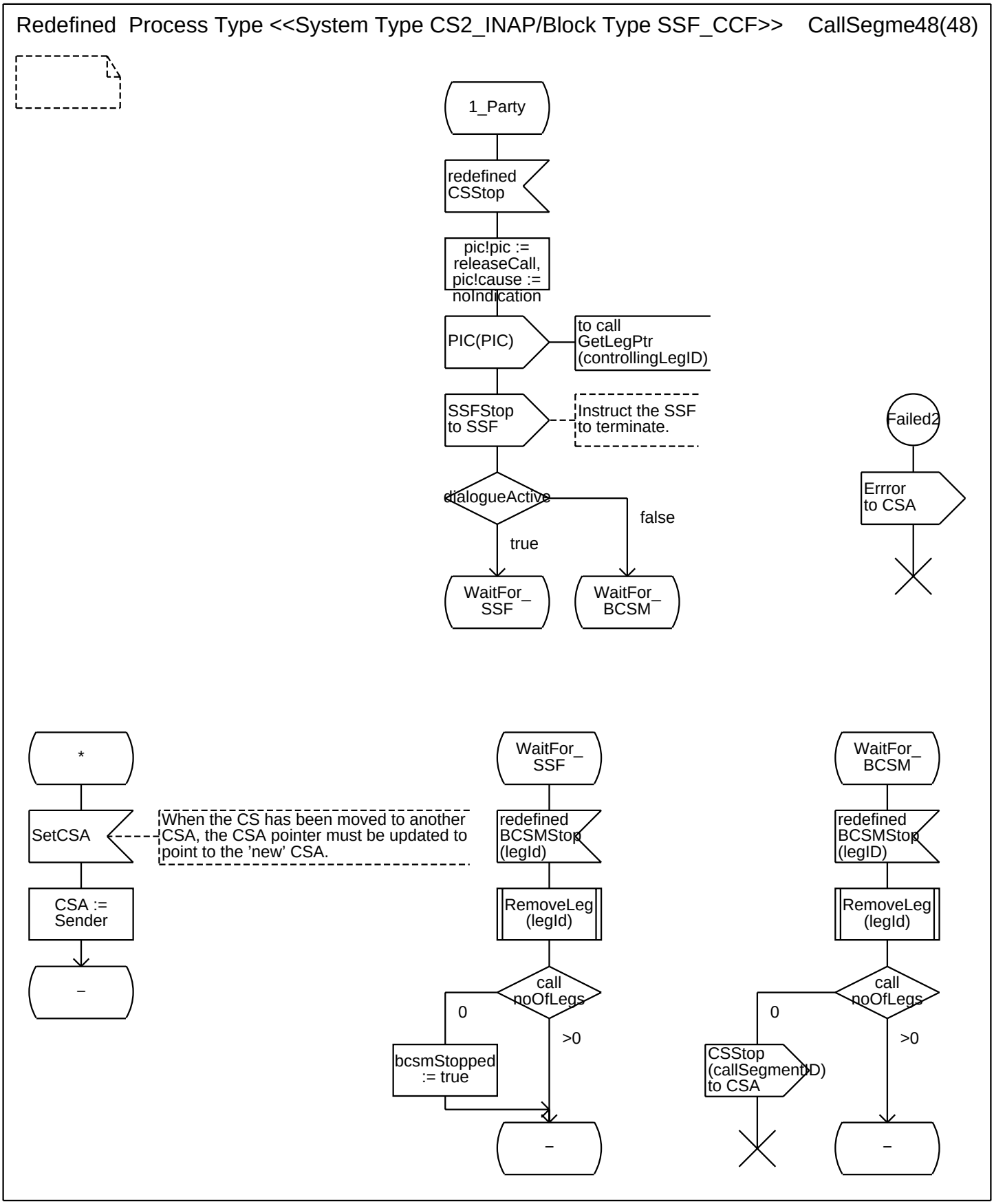
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme46(48)



Annex A:

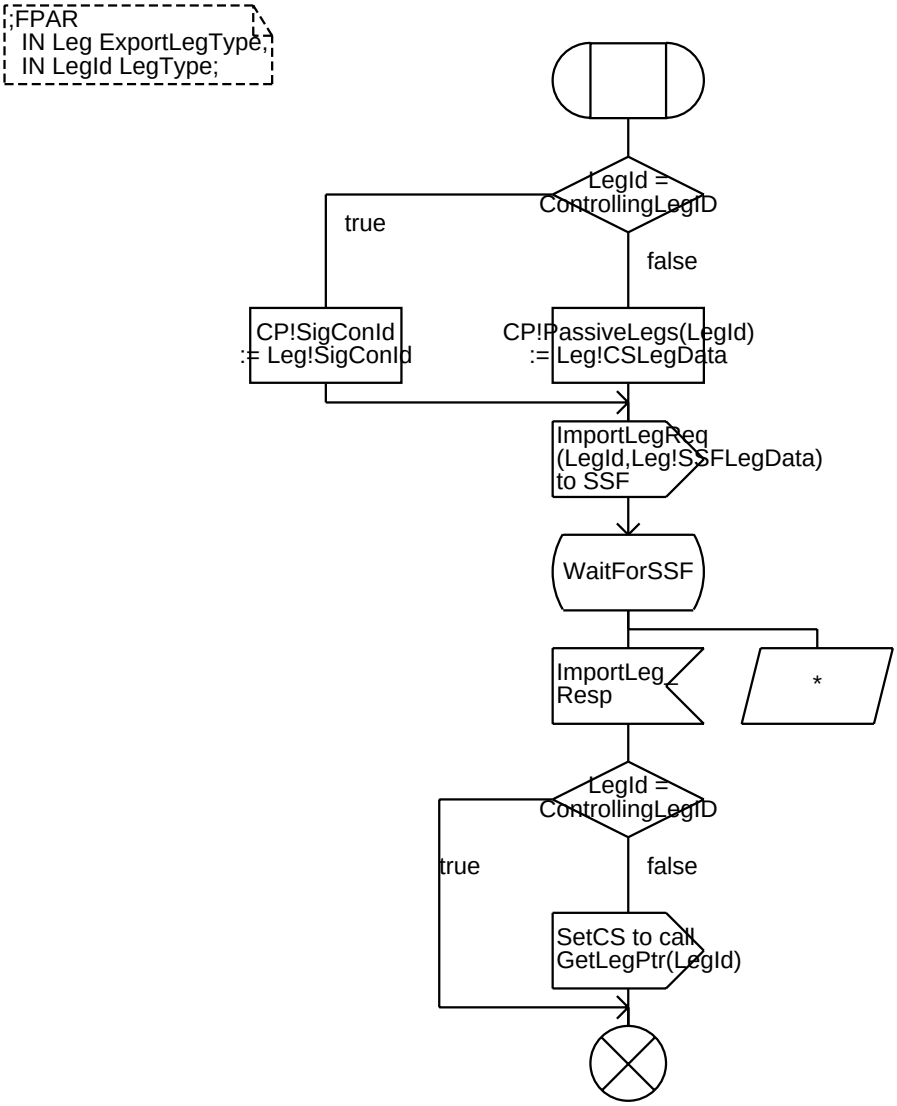

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme47(48)
/* In all states except Null, DPUTSI is only reported to the SSF−FSM if there is a dialogue active, otherwise the DPUTSI will be transformed into a SendSTUI PIC and the information will be send (by the BCSM) to the remote party. */
*
(Null)
DPUTSI
(dpUTSIArg)
dialogue_
Active
True
False
DPFiltering_
UTSI
(dpUTSIArg)
/* The UTSI is coming from the controlling leg, and should be sent to the remote party. */
/* The UTSI is coming fr
the remote party, and sh
be sent on the controllin
dpUTSIArg!
party
localParty
remoteParty
pic!party
= remoteParty
pic!party
:= localParty
pic!pic := SendSTUI,
pic!infoDirection := utsi,
pic!uSIServiceIndicator := dpUTSIArg!
uSIServiceIndicator,
pic!uSIInformation := dpUTSIArg!
uSIInformation
Instruct the BCSM to forward the information to the remote party.
PIC(pic)
to Sender
−

Annex A:

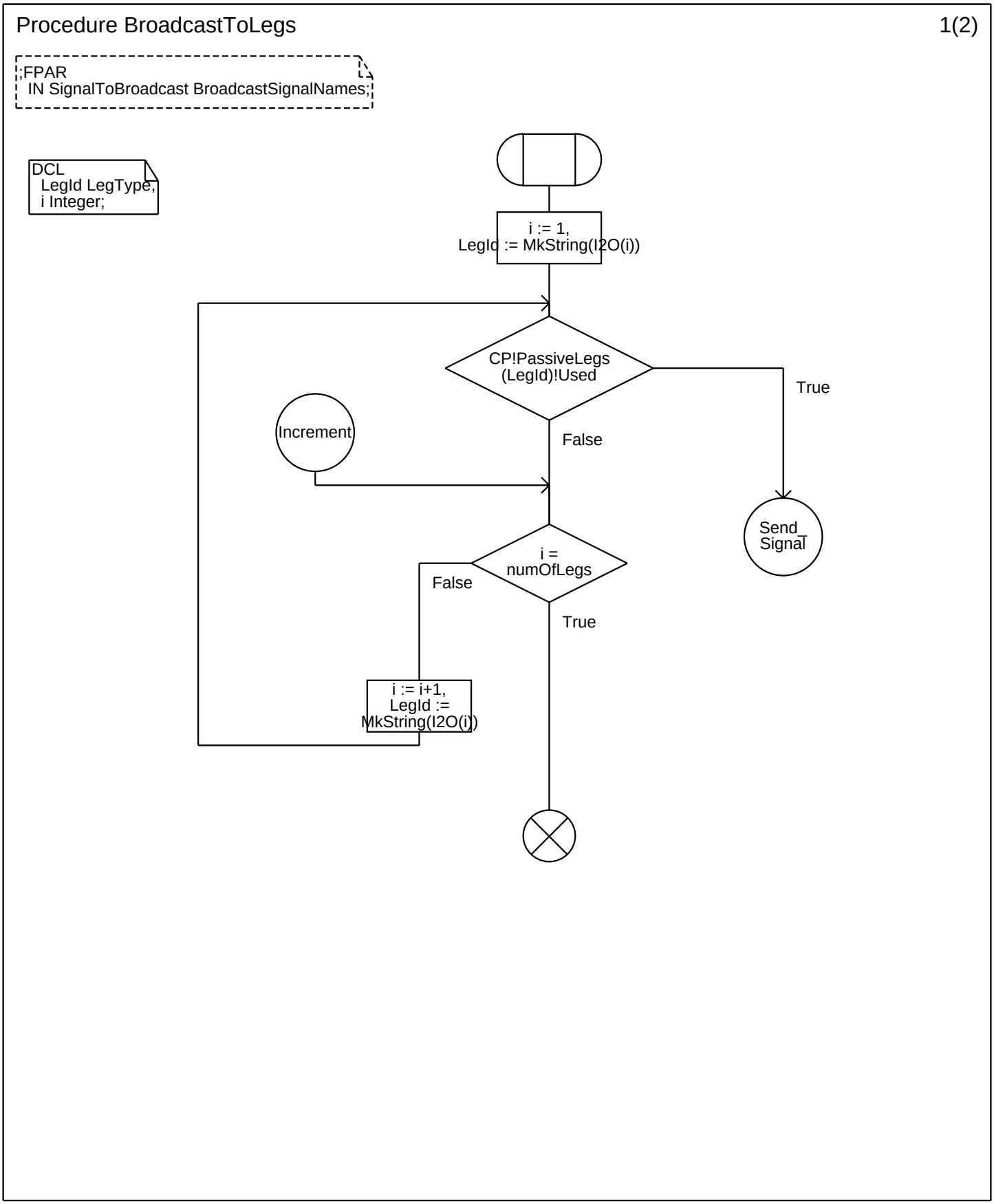

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegme48(48)
1_Party
redefined CSStop
pic!pic := releaseCall, pic!cause := noIndication
PIC(PIC)
to call GetLegPtr (controllingLegID)
SSFStop to SSF
Instruct the SSF to terminate.
dialogueActive
true
false
WaitFor_ SSF
WaitFor_ BCSM
Failed2
Errror to CSA
*
SetCSA
When the CS has been moved to another CSA, the CSA pointer must be updated to point to the 'new' CSA.
CSA := Sender
–
WaitFor_ SSF
redefined BCSMStop (legId)
RemoveLeg (legId)
call noOfLegs
0
>0
bcsmStopped := true
–
WaitFor_ BCSM
redefined BCSMStop (legID)
RemoveLeg (legId)
call noOfLegs
0
>0
CSStop (callSegmentID) to CSA
–

Annex A:

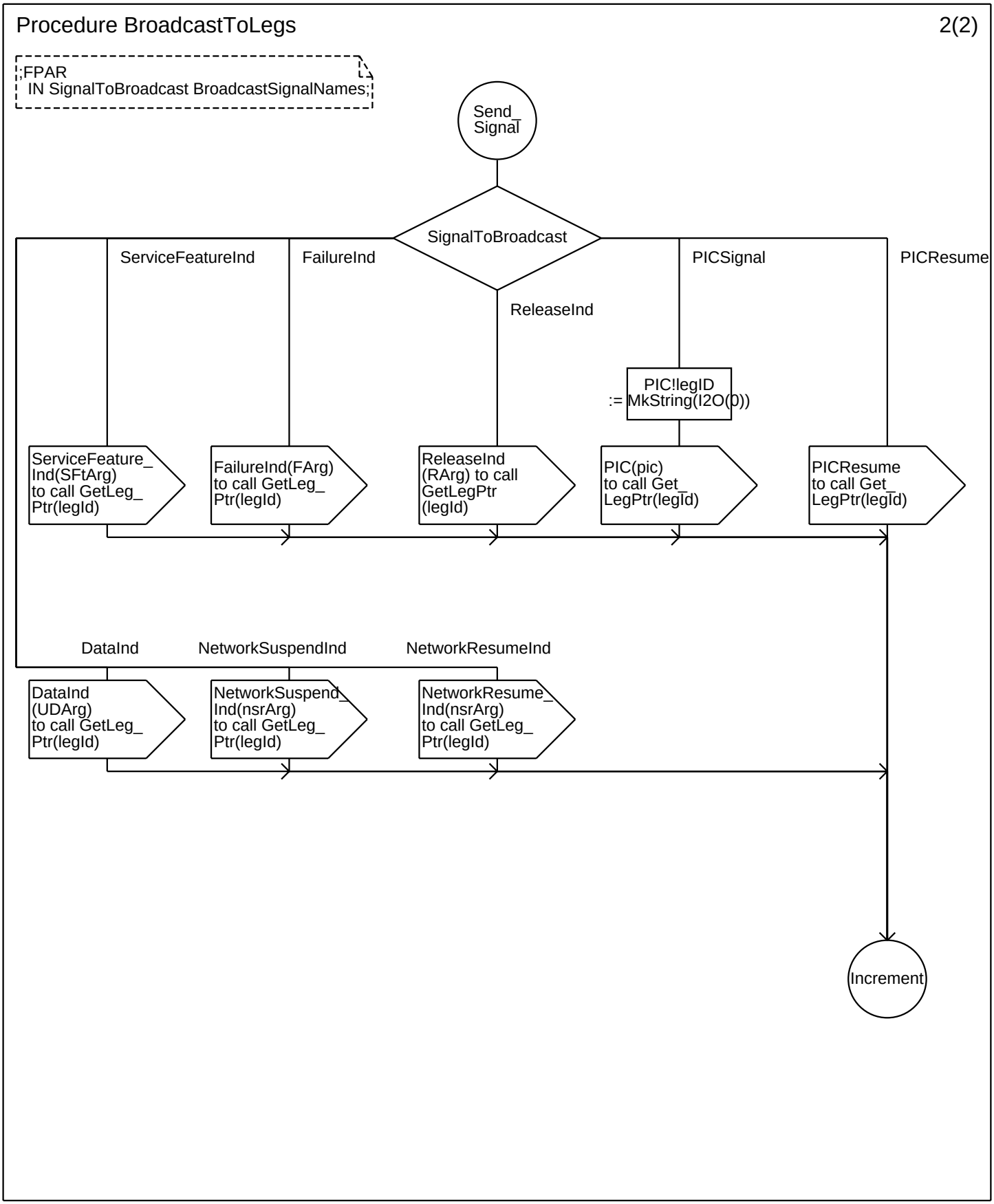
Procedure <<Process Type CallSegment>> ImportLeg 1(1)



Annex A:

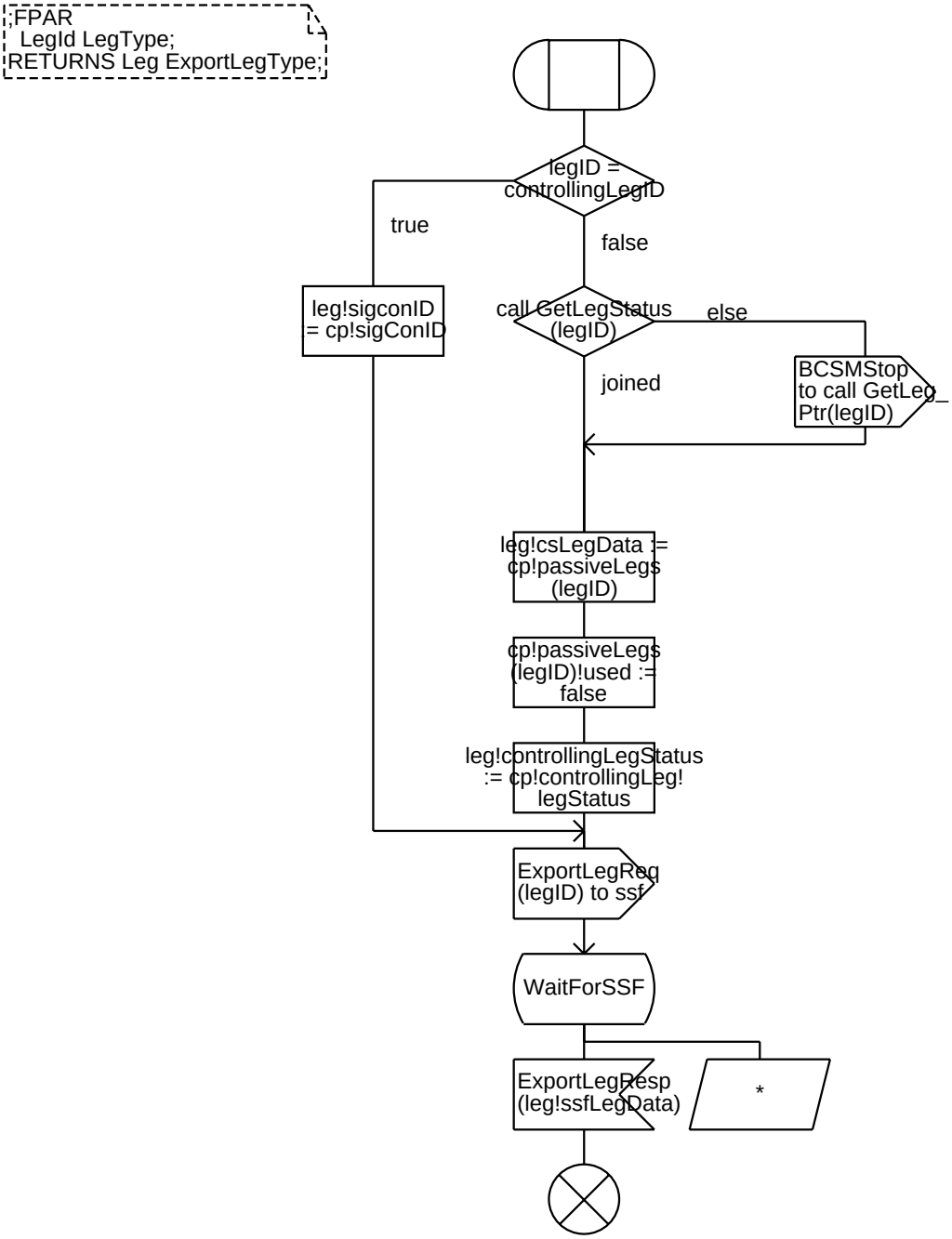

Procedure BroadcastToLegs
1(2)
;FPAR
IN SignalToBroadcast BroadcastSignalNames;
DCL
LegId LegType,
i Integer;
i := 1,
LegId := MkString(I2O(i))
CP!PassiveLegs
(LegId)!Used
True
Increment
False
Send_
Signal
i =
numOfLegs
False
True
i := i+1,
LegId :=
MkString(I2O(i))

Annex A:

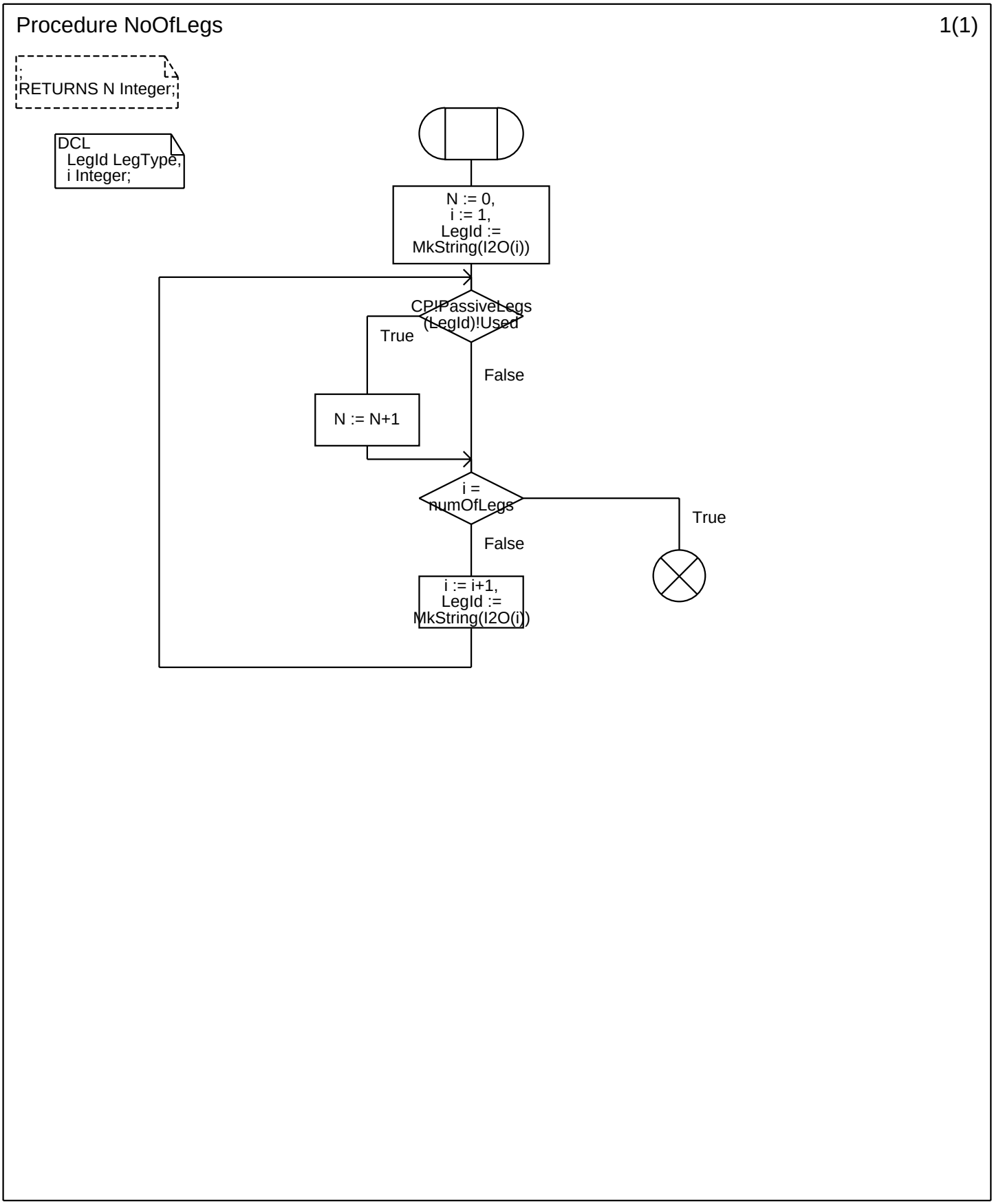

Procedure BroadcastToLegs
2(2)
;FPAR
IN SignalToBroadcast BroadcastSignalNames;
Send_
Signal
SignalToBroadcast
ServiceFeatureInd
FailureInd
PICSignal
PICResume
ReleaseInd
PIC!legID
:= MkString(I2O(0))
ServiceFeature_
Ind(SFtArg)
to call GetLeg_
Ptr(legId)
FailureInd(FArg)
to call GetLeg_
Ptr(legId)
ReleaseInd
(RArg) to call
GetLegPtr
(legId)
PIC(pic)
to call Get_
LegPtr(legId)
PICResume
to call Get_
LegPtr(legId)
DataInd
NetworkSuspendInd
NetworkResumeInd
DataInd
(UDArg)
to call GetLeg_
Ptr(legId)
NetworkSuspend_
Ind(nsrArg)
to call GetLeg_
Ptr(legId)
NetworkResume_
Ind(nsrArg)
to call GetLeg_
Ptr(legId)
Increment

Annex A:

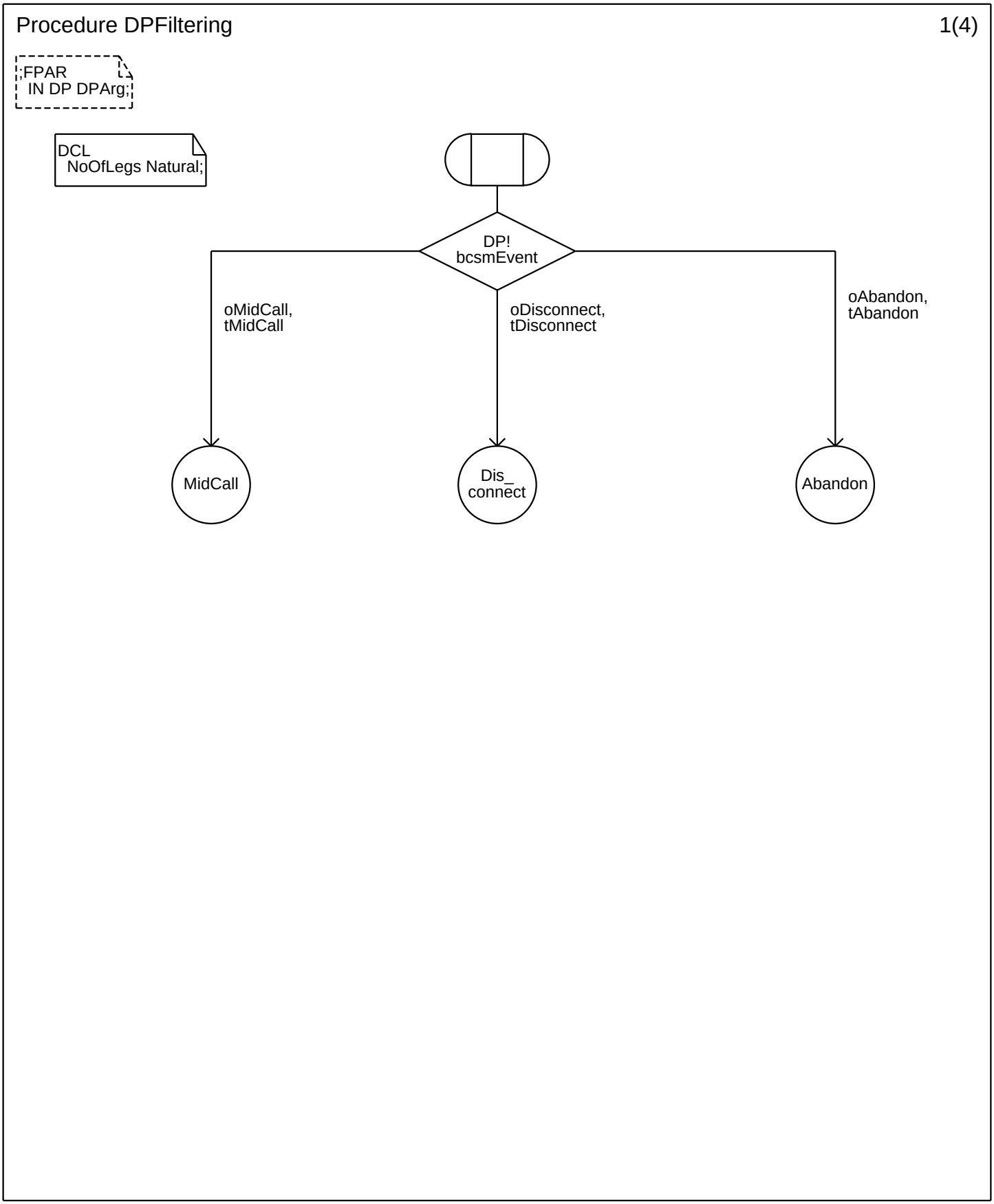
Procedure <<Process Type CallSegment>> ExportLeg 1(1)



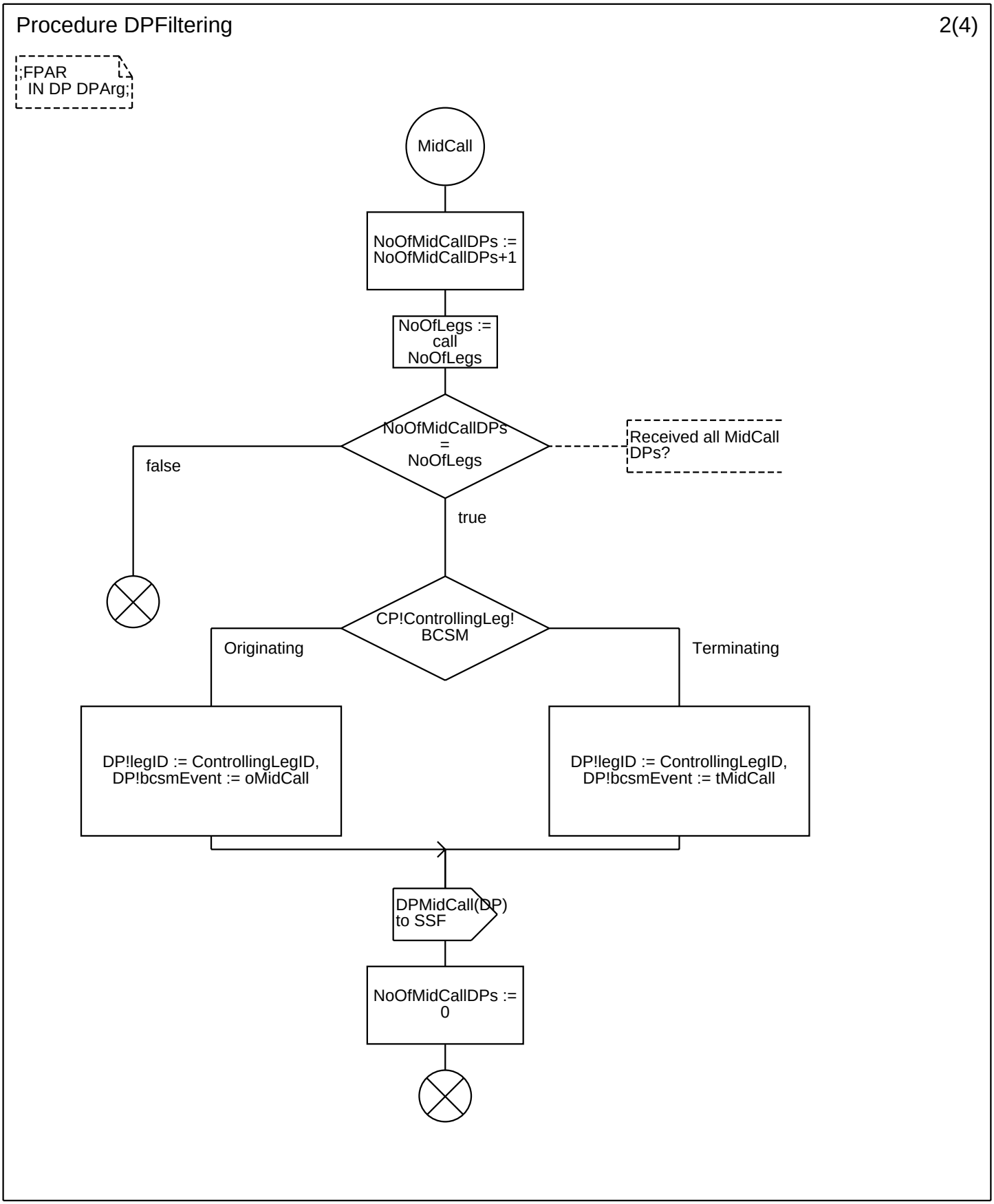
Annex A:


Procedure NoOfLegs
1(1)
;
RETURNS N Integer;
DCL
LegId LegType,
i Integer;
N := 0,
i := 1,
LegId :=
MkString(I2O(i))
CP!PassiveLegs
(LegId)!Used
True
False
N := N+1
i =
numOfLegs
True
False
i := i+1,
LegId :=
MkString(I2O(i))

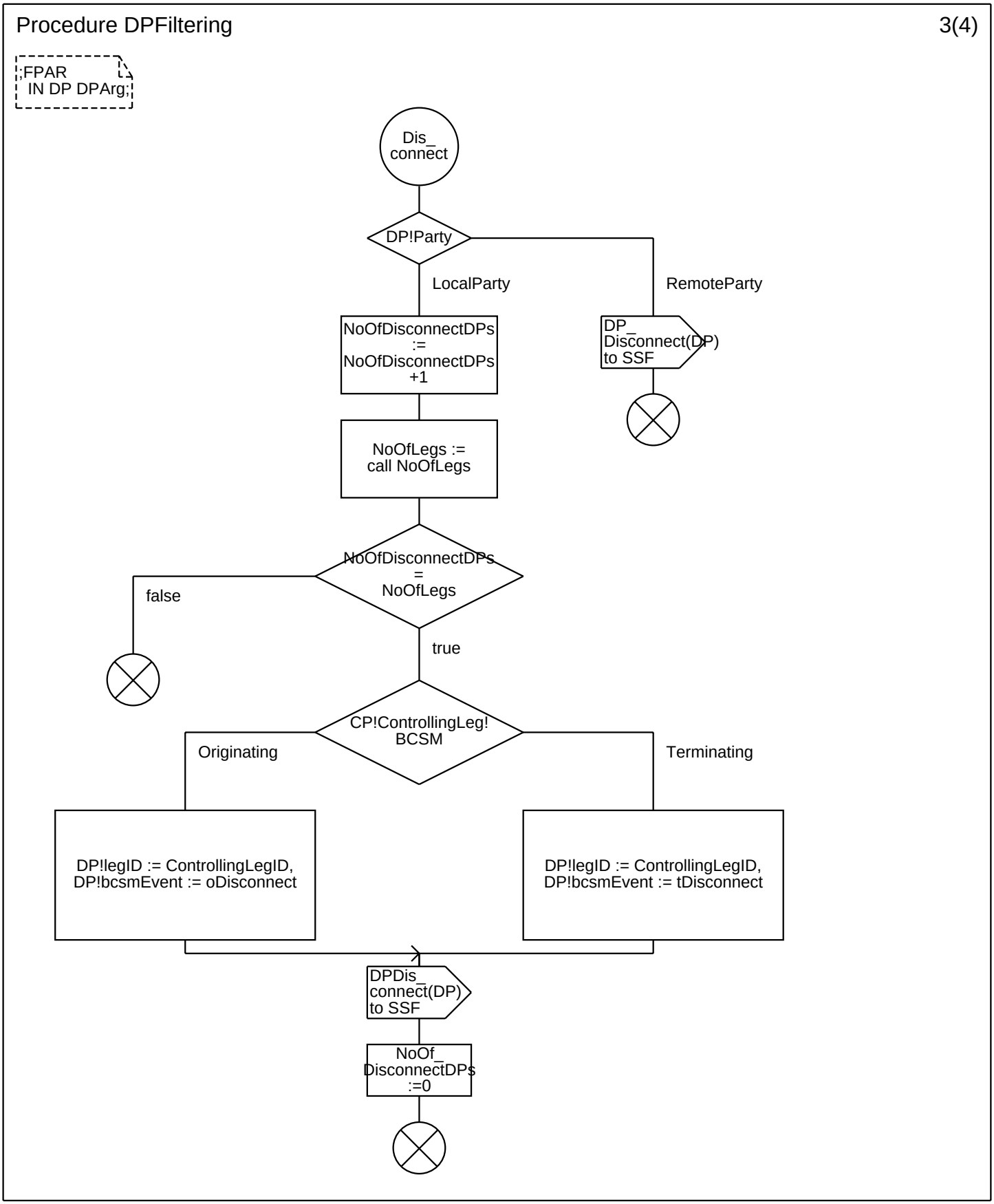
Annex A:


Procedure DPFiltering
1(4)
;FPAR
IN DP DPArg;
DCL
NoOfLegs Natural;
DP!
bcsmEvent
oMidCall,
tMidCall
oDisconnect,
tDisconnect
oAbandon,
tAbandon
MidCall
Dis_
connect
Abandon

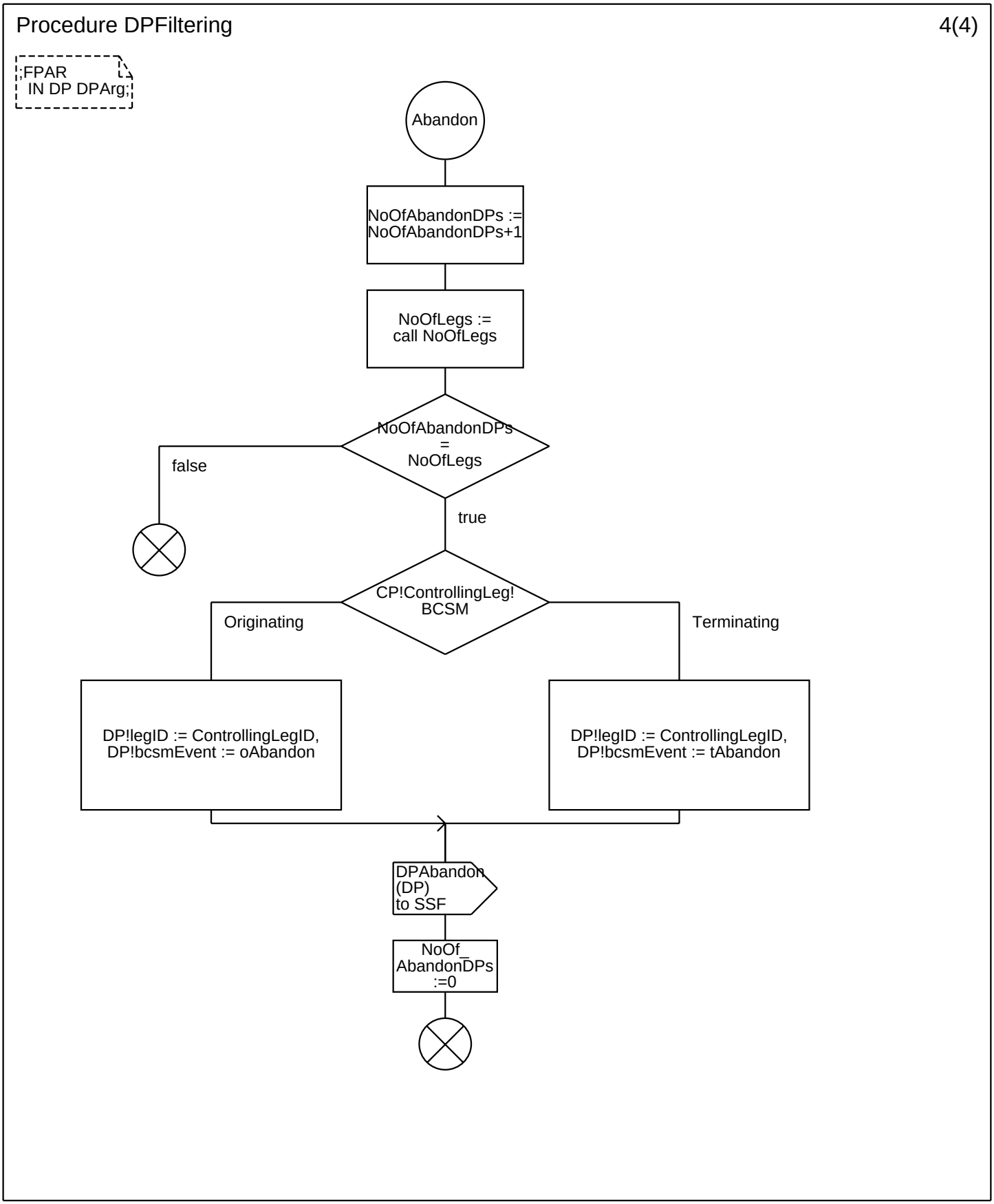
Annex A:


Procedure DPFiltering
2(4)
;FPAR
IN DP DPArg;
MidCall
NoOfMidCallDPs :=
NoOfMidCallDPs+1
NoOfLegs :=
call
NoOfLegs
NoOfMidCallDPs
=
NoOfLegs
Received all MidCall
DPs?
false
true
CP!ControllingLeg!
BCSM
Originating
Terminating
DP!legID := ControllingLegID,
DP!bcsmEvent := oMidCall
DP!legID := ControllingLegID,
DP!bcsmEvent := tMidCall
DPMidCall(DP)
to SSF
NoOfMidCallDPs :=
0

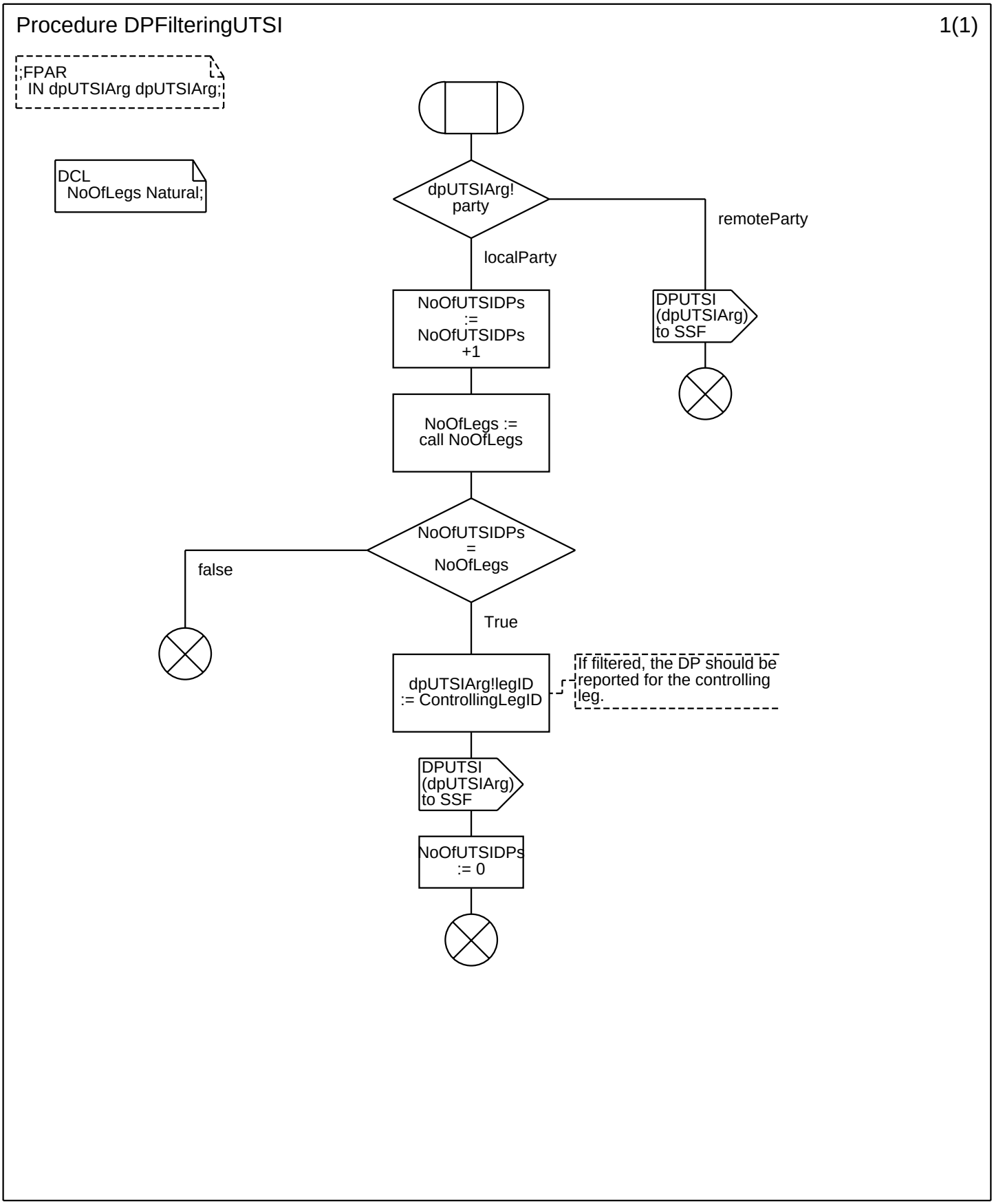
Annex A:


Procedure DPFiltering
3(4)
;FPAR
IN DP DPArg;
Dis_
connect
DP!Party
LocalParty
RemoteParty
NoOfDisconnectDPs
:=
NoOfDisconnectDPs
+1
DP_
Disconnect(DP)
to SSF
NoOfLegs :=
call NoOfLegs
NoOfDisconnectDPs
=
NoOfLegs
false
true
CP!ControllingLeg!
BCSM
Originating
Terminating
DP!legID := ControllingLegID,
DP!bcsmEvent := oDisconnect
DP!legID := ControllingLegID,
DP!bcsmEvent := tDisconnect
DPDis_
connect(DP)
to SSF
NoOf_
DisconnectDPs
:=0

Annex A:


Procedure DPFiltering
4(4)
;FPAR
IN DP DPArg;
Abandon
NoOfAbandonDPs :=
NoOfAbandonDPs+1
NoOfLegs :=
call NoOfLegs
NoOfAbandonDPs
=
NoOfLegs
false
true
CP!ControllingLeg!
BCSM
Originating
Terminating
DP!legID := ControllingLegID,
DP!bcsmEvent := oAbandon
DP!legID := ControllingLegID,
DP!bcsmEvent := tAbandon
DPAbandon
(DP)
to SSF
NoOf_
AbandonDPs
:=0

Annex A:


Procedure DPFilteringUTSI
1(1)
;FPAR
IN dpUTSIArg dpUTSIArg;
DCL
NoOfLegs Natural;
dpUTSIArg!
party
remoteParty
localParty
NoOfUTSIDPs
:=
NoOfUTSIDPs
+1
DPUTSI
(dpUTSIArg)
to SSF
NoOfLegs :=
call NoOfLegs
NoOfUTSIDPs
=
NoOfLegs
false
True
dpUTSIArg!legID
:= ControllingLegID
If filtered, the DP should be reported for the controlling leg.
DPUTSI
(dpUTSIArg)
to SSF
NoOfUTSIDPs
:= 0

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM1(16)

```
/* Note: The SSF−FSM for IN CS−2 only shows the extentions required to the
IN CS−1 SSF−FSM. Therefore, to understand the complete SSF−FSM behaviour it is
necessary to read the two SSF−FSM descriptions together. */
```

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

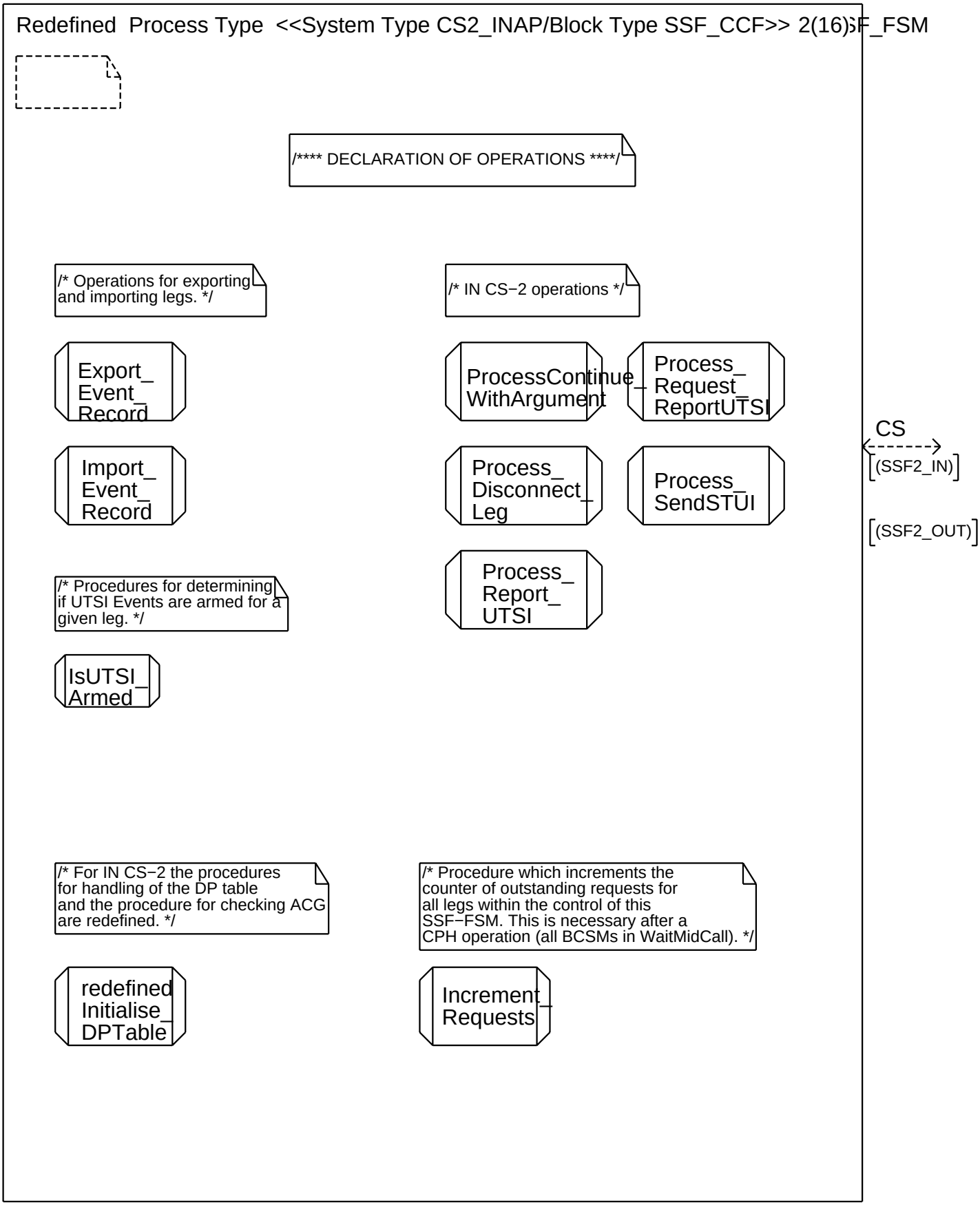
```
/* The SSF−FSM must maintain the arming and
disarming of UTSI events for each leg. */

NEWTYPE UTSIArmedType
  ARRAY(LegType,Boolean)
ENDNEWTYPE;
```

```
/**** DECLARATIONS OF VARIABLES. ****/
```

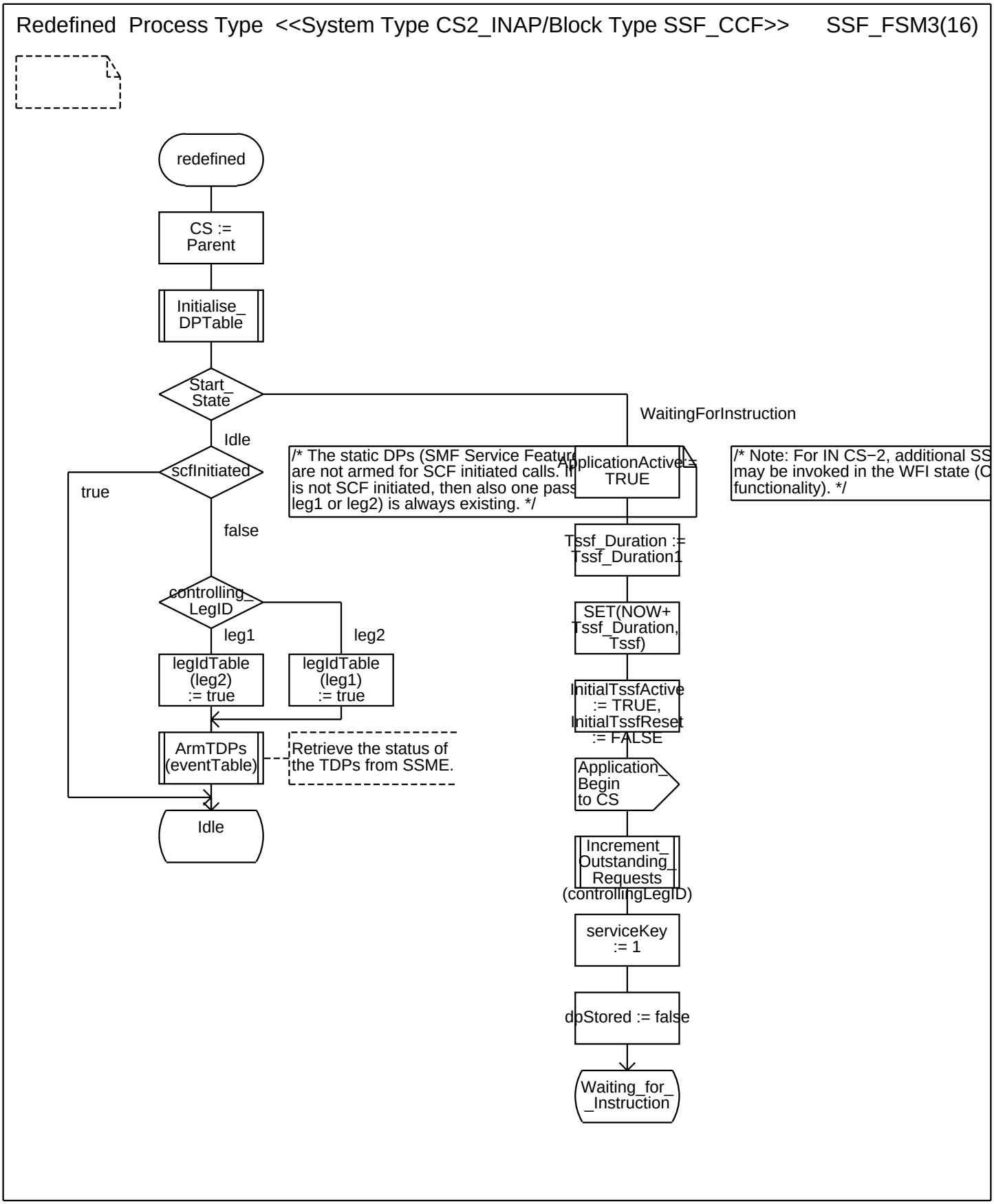
```
DCL
  currentLegID, newLegID LegType,
  utsiArmed UTSIArmedType,
  exportEventRecord EventRecordType,
  importEventRecord EventRecordType,
  dpUTSIArg DPUTSIArg,
  legID LegType;
```

```
DCL
  /* IN CS−2 operation arguments. */
  ctwaArg ContinueWithArgumentArg,
  dfcwaArg DisconnectForwardConnectionWithArgumentArg,
  dlArg DisconnectLegArg,
  rutsiArg ReportUTSIArg,
  rrutsiArg RequestReportUTSIArg,
  sstuiArg SendSTUIArg,

  parmArg PromptAndReceiveMessageArg,
  scArg ScriptCloseArg,
  siArg ScriptInformationArg,
  srArg ScriptRunArg;
```

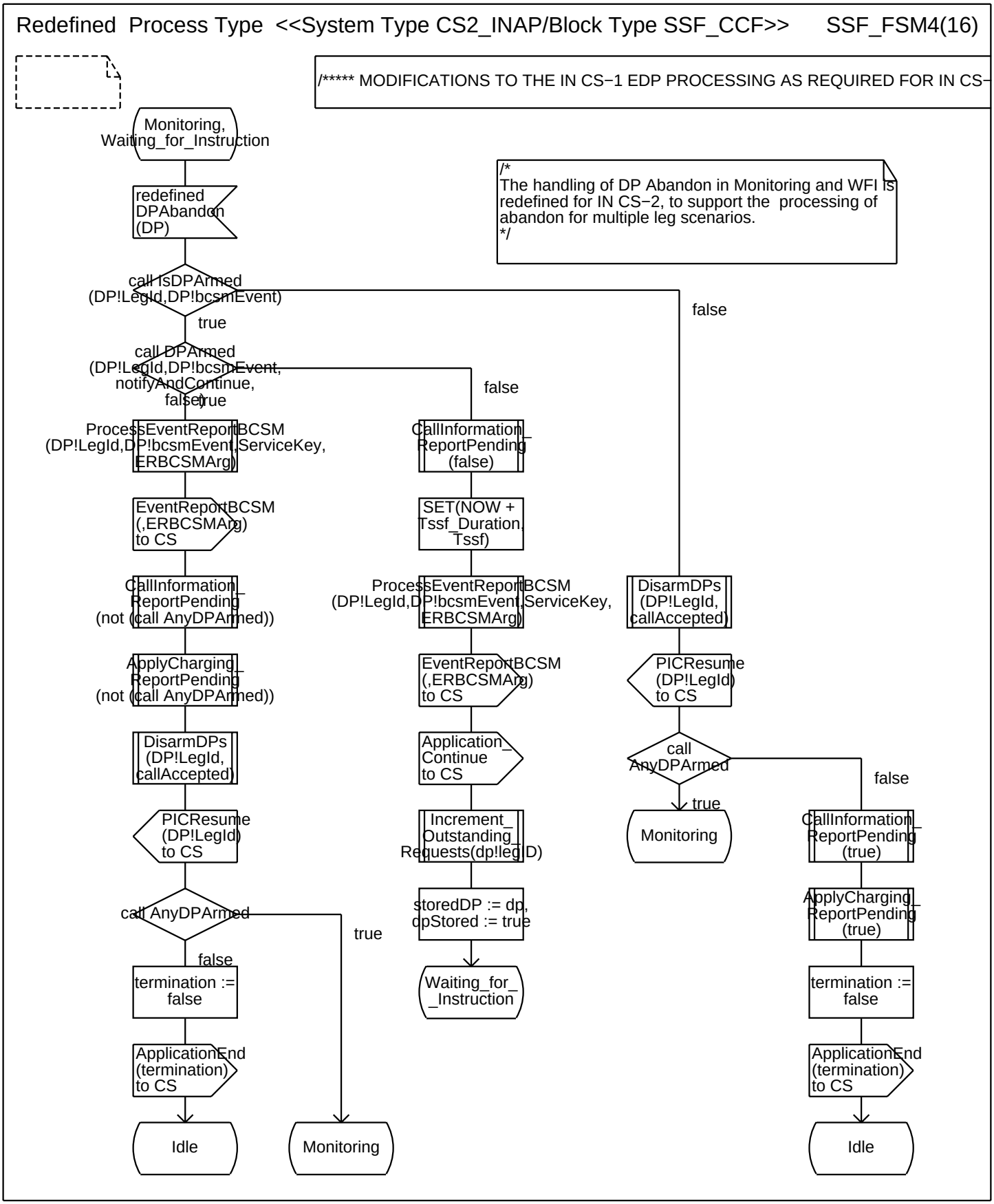
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> 2(16)SSF_FSM
/**** DECLARATION OF OPERATIONS ****/
/* Operations for exporting and importing legs. */
Export_ Event_ Record
Import_ Event_ Record
/* Procedures for determining if UTSI Events are armed for a given leg. */
IsUTSI_ Armed
/* IN CS−2 operations */
ProcessContinue_ WithArgument
Process_ Request_ ReportUTSI
Process_ Disconnect_ Leg
Process_ SendSTUI
Process_ Report_ UTSI
CS
(SSF2_IN)
(SSF2_OUT)
/* For IN CS−2 the procedures for handling of the DP table and the procedure for checking ACG are redefined. */
redefined Initialise_ DPTable
/* Procedure which increments the counter of outstanding requests for all legs within the control of this SSF−FSM. This is necessary after a CPH operation (all BCSMs in WaitMidCall). */
Increment_ Requests

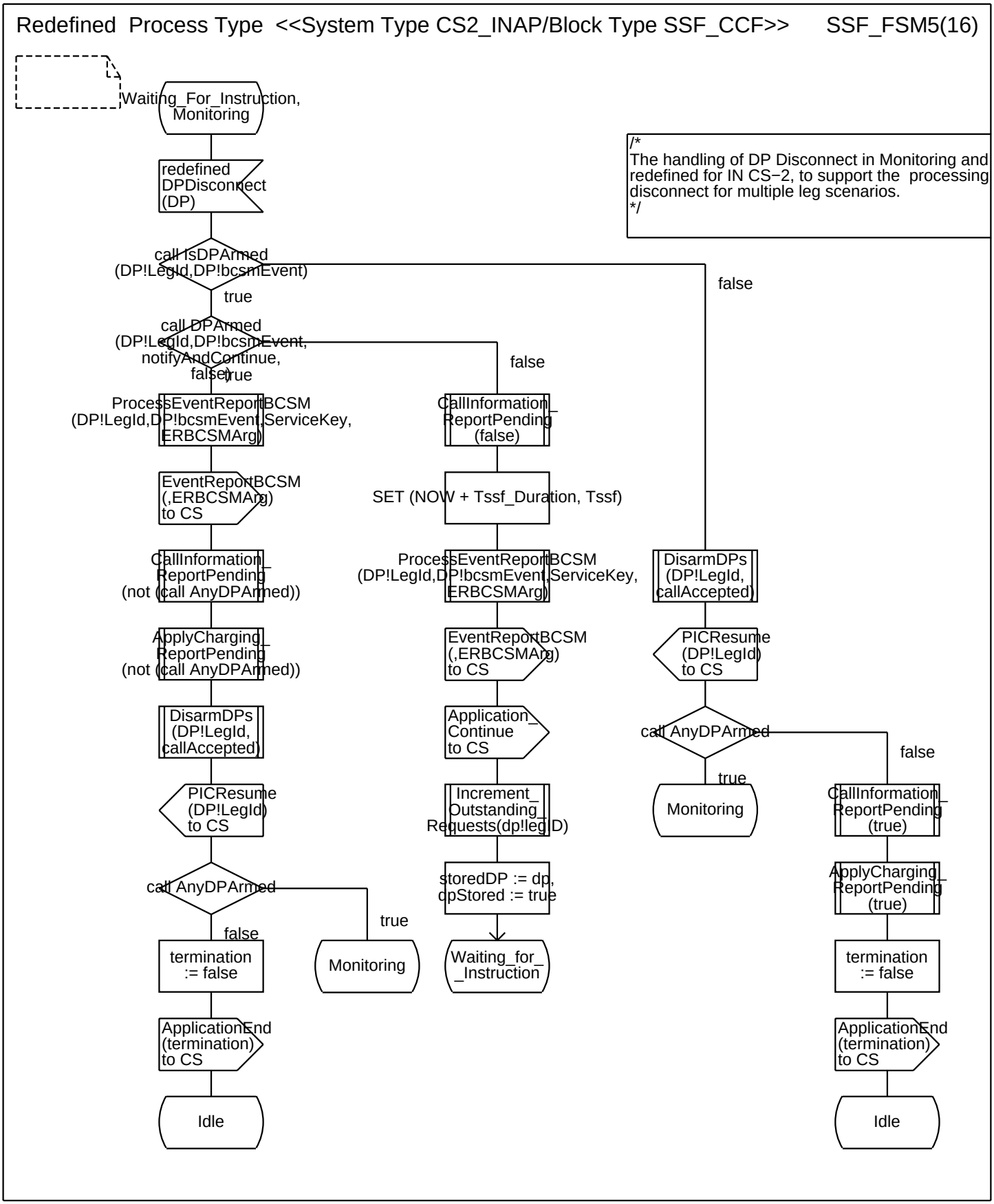
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM3(16)
redefined
CS := Parent
Initialise_ DPTable
Start_ State
WaitingForInstruction
Idle
scfInitiated
true
false
/* The static DPs (SMF Service Feature
are not armed for SCF initiated calls. If
is not SCF initiated, then also one pass
leg1 or leg2) is always existing. */
ApplicationActive := TRUE
/* Note: For IN CS−2, additional SS
may be invoked in the WFI state (C
functionality). */
Tssf_Duration := Tssf_Duration1
controlling_ LegID
leg1
leg2
legIdTable (leg2) := true
legIdTable (leg1) := true
SET(NOW+ Tssf_Duration, Tssf)
InitialTssfActive := TRUE, InitialTssfReset := FALSE
ArmTDPs (eventTable)
Retrieve the status of the TDPs from SSME.
Application_ Begin to CS
Idle
Increment_ Outstanding_ Requests (controllingLegID)
serviceKey := 1
dpStored := false
Waiting_for_ _Instruction

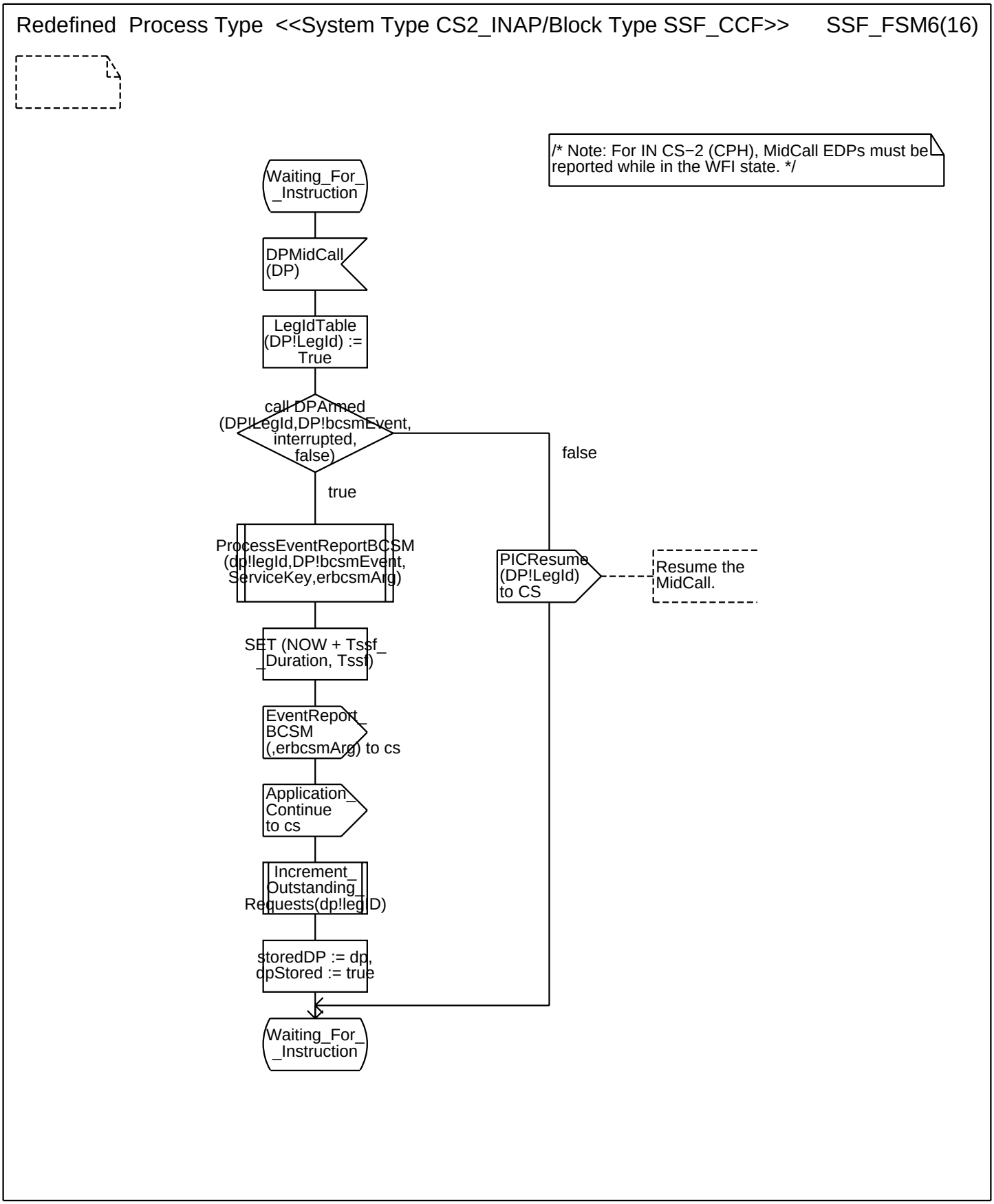
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM4(16)
/***** MODIFICATIONS TO THE IN CS−1 EDP PROCESSING AS REQUIRED FOR IN CS−
Monitoring, Waiting_for_Instruction
redefined DPAbandon (DP)
/* The handling of DP Abandon in Monitoring and WFI is redefined for IN CS−2, to support the processing of abandon for multiple leg scenarios. */
call IsDPArmed (DP!LegId,DP!bcsmEvent)
true
false
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue, false)
true
false
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent,ServiceKey, ERBCSMArg)
EventReportBCSM (,ERBCSMArg) to CS
CallInformation_ ReportPending (not (call AnyDPArmed))
ApplyCharging_ ReportPending (not (call AnyDPArmed))
DisarmDPs (DP!LegId, callAccepted)
PICResume (DP!LegId) to CS
call AnyDPArmed
true
false
termination := false
ApplicationEnd (termination) to CS
Idle
Monitoring
CallInformation_ ReportPending (false)
SET(NOW + Tssf_Duration, Tssf)
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent,ServiceKey, ERBCSMArg)
EventReportBCSM (,ERBCSMArg) to CS
Application_ Continue to CS
Increment_ Outstanding_ Requests(dp!legID)
storedDP := dp, dpStored := true
Waiting_for_ _Instruction
DisarmDPs (DP!LegId, callAccepted)
PICResume (DP!LegId) to CS
call AnyDPArmed
true
false
Monitoring
CallInformation_ ReportPending (true)
ApplyCharging_ ReportPending (true)
termination := false
ApplicationEnd (termination) to CS
Idle

Annex A:

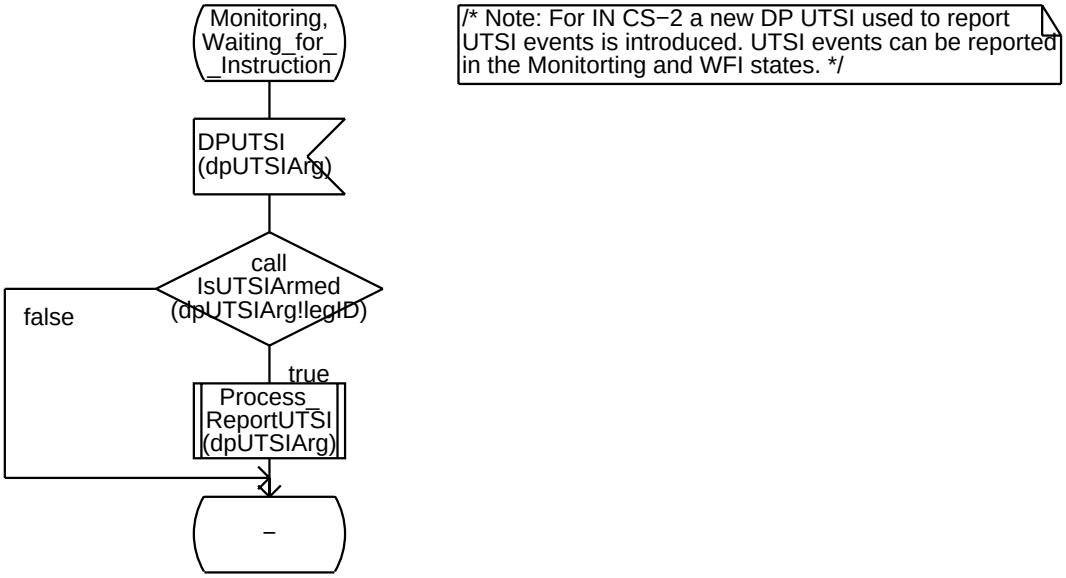

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM5(16)
Waiting_For_Instruction, Monitoring
redefined DPDisconnect (DP)
/*
The handling of DP Disconnect in Monitoring and redefined for IN CS−2, to support the processing disconnect for multiple leg scenarios.
*/
call IsDPArmed (DP!LegId,DP!bcsmEvent)
true
false
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue, false)
true
false
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent,ServiceKey, ERBCSMArg)
CallInformation_ ReportPending (false)
EventReportBCSM (,ERBCSMArg) to CS
SET (NOW + Tssf_Duration, Tssf)
CallInformation_ ReportPending (not (call AnyDPArmed))
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent,ServiceKey, ERBCSMArg)
DisarmDPs (DP!LegId, callAccepted)
ApplyCharging_ ReportPending (not (call AnyDPArmed))
EventReportBCSM (,ERBCSMArg) to CS
PICResume (DP!LegId) to CS
DisarmDPs (DP!LegId, callAccepted)
Application_ Continue to CS
call AnyDPArmed
false
true
PICResume (DP!LegId) to CS
Increment_ Outstanding_ Requests(dp!legID)
Monitoring
CallInformation_ ReportPending (true)
call AnyDPArmed
storedDP := dp, dpStored := true
ApplyCharging_ ReportPending (true)
true
false
termination := false
Monitoring
Waiting_for_ _Instruction
termination := false
ApplicationEnd (termination) to CS
ApplicationEnd (termination) to CS
Idle
Idle

Annex A:

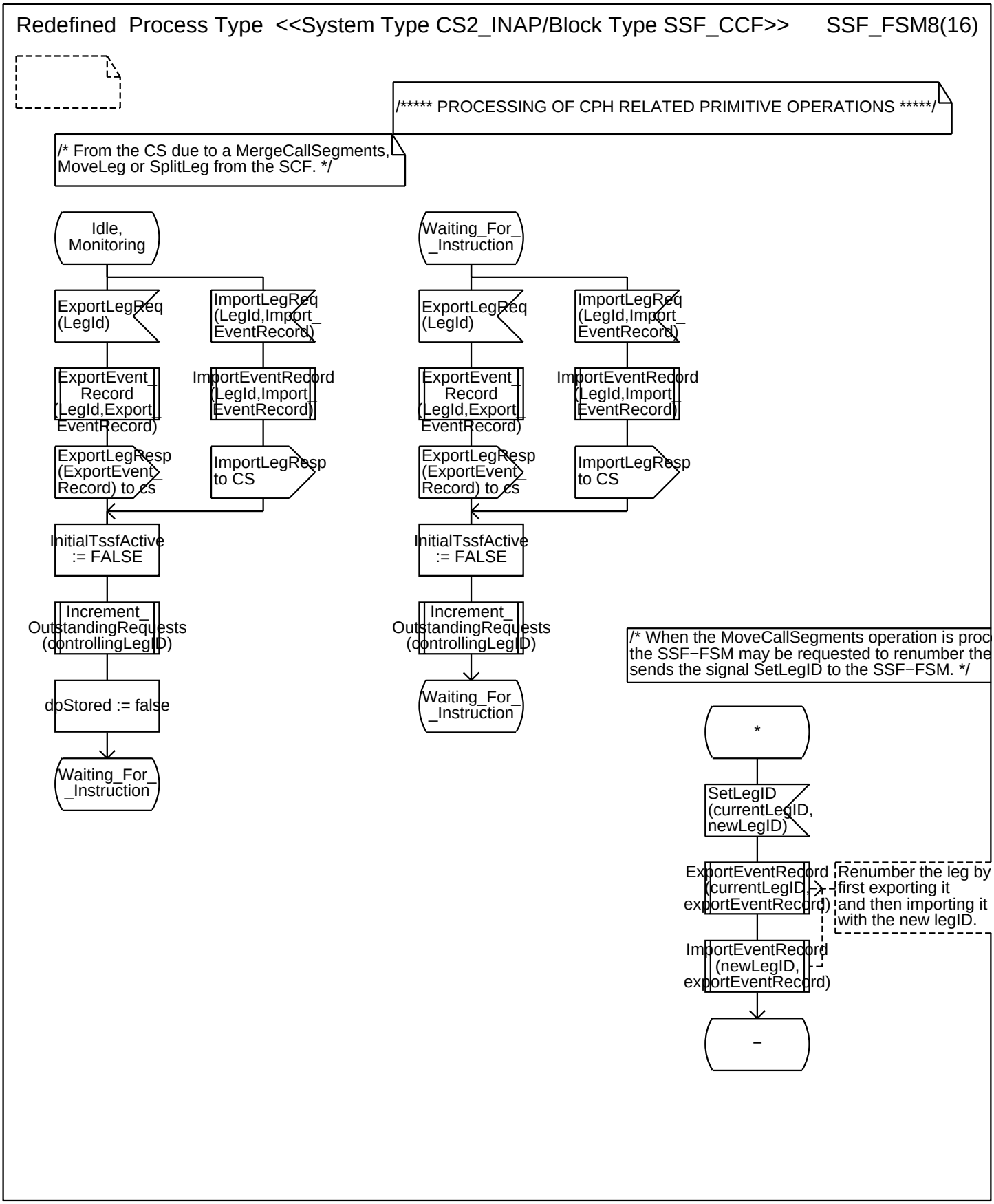

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM6(16)
/* Note: For IN CS−2 (CPH), MidCall EDPs must be reported while in the WFI state. */
Waiting_For_ _Instruction
DPMidCall (DP)
LegIdTable (DP!LegId) := True
call DPArmed (DP!LegId,DP!bcsmEvent, interrupted, false)
false
true
ProcessEventReportBCSM (dp!legId,DP!bcsmEvent, ServiceKey,erbcsmArg)
PICResume (DP!LegId) to CS
Resume the MidCall.
SET (NOW + Tssf_ _Duration, Tssf)
EventReport_ BCSM (,erbcsmArg) to cs
Application_ Continue to cs
Increment_ Outstanding_ Requests(dp!legID)
storedDP := dp, dpStored := true
Waiting_For_ _Instruction

Annex A:

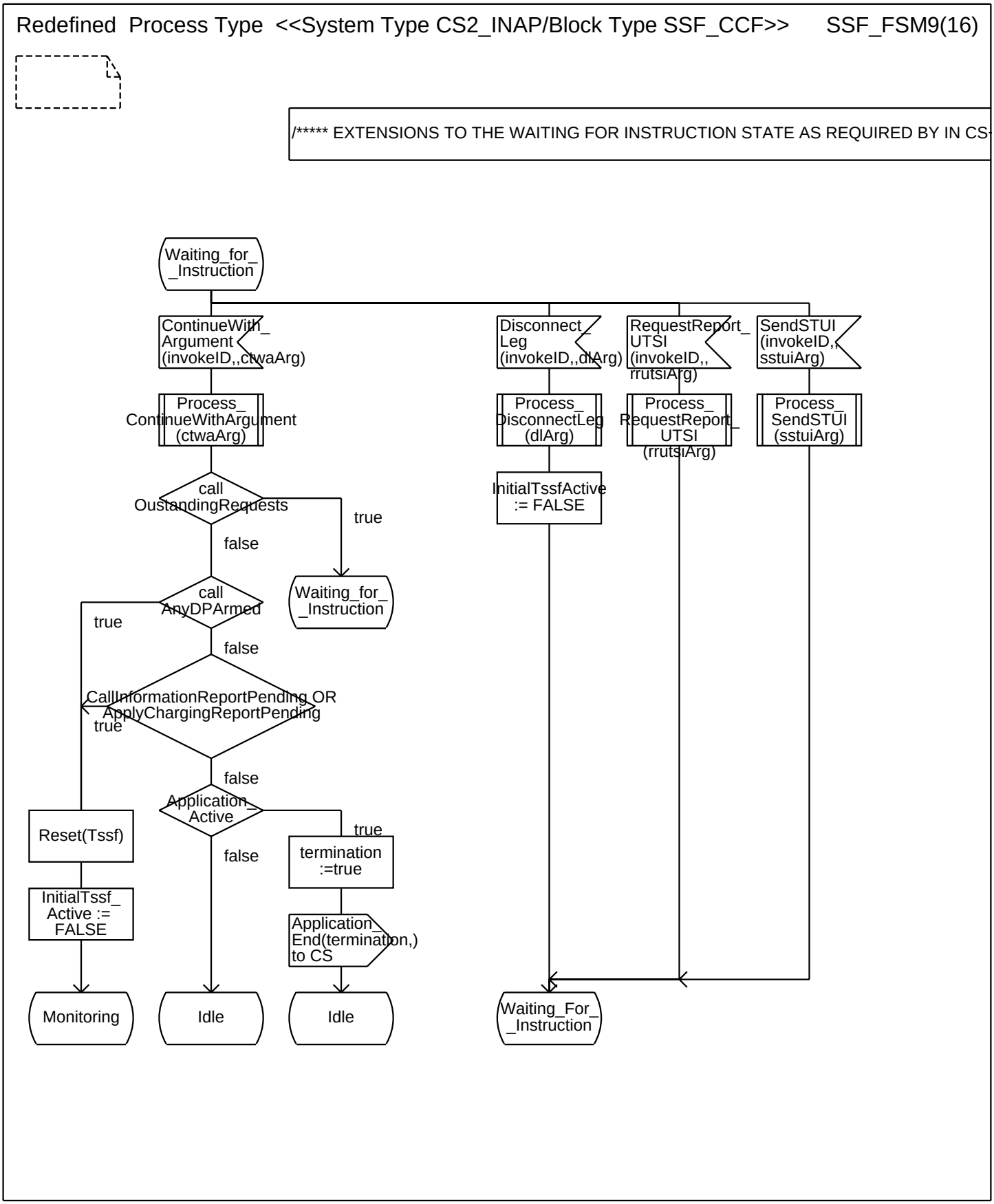
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM7(16)



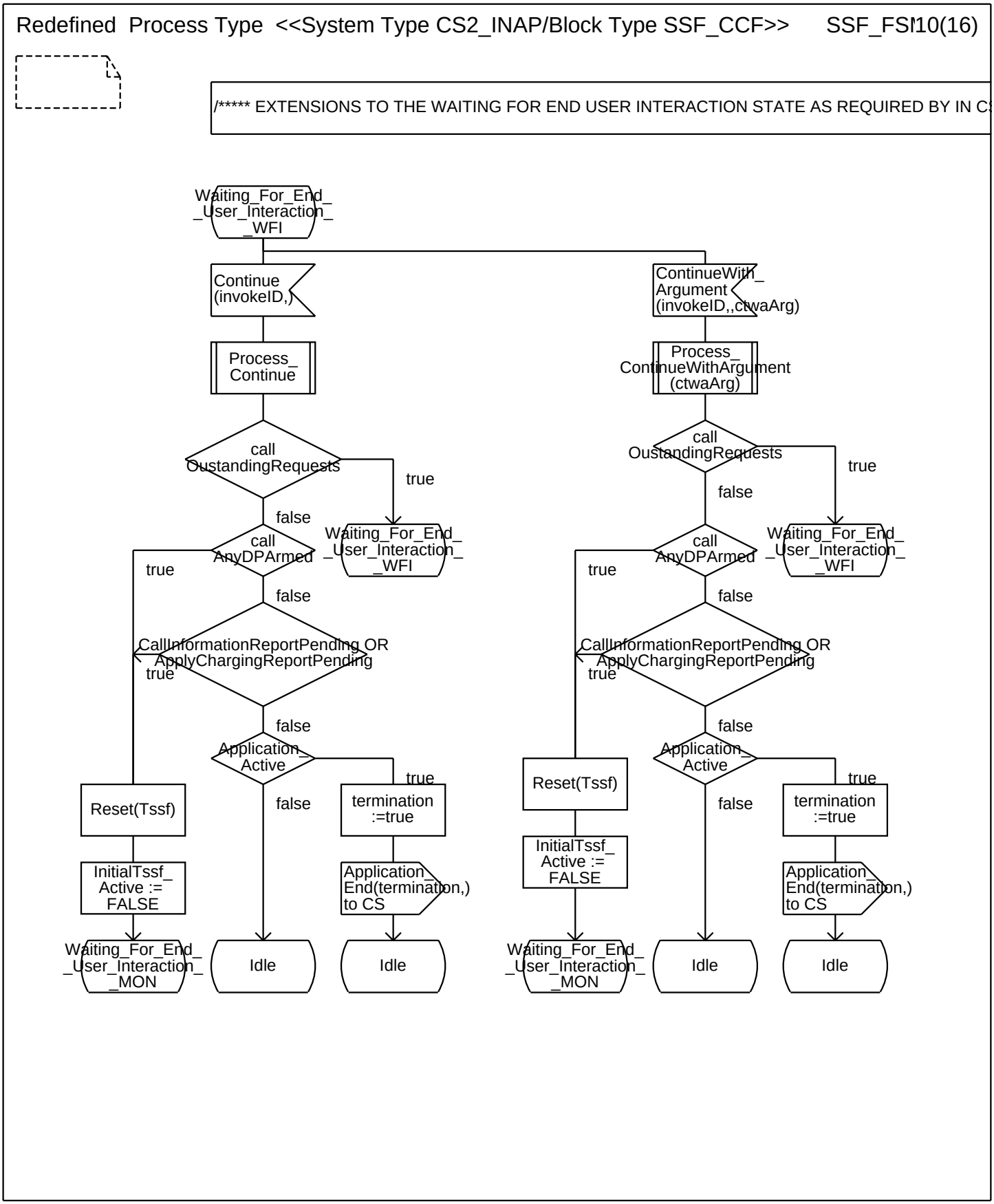
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM8(16)
/***** PROCESSING OF CPH RELATED PRIMITIVE OPERATIONS *****/
/* From the CS due to a MergeCallSegments, MoveLeg or SplitLeg from the SCF. */
Idle, Monitoring
ExportLegReq (LegId)
ImportLegReq (LegId,Import_EventRecord)
ExportEvent_Record (LegId,Export_EventRecord)
ImportEventRecord (LegId,Import_EventRecord)
ExportLegResp (ExportEvent_Record) to cs
ImportLegResp to CS
InitialTssfActive := FALSE
Increment_OutstandingRequests (controllingLegID)
dpStored := false
Waiting_For_Instruction
Waiting_For_Instruction
ExportLegReq (LegId)
ImportLegReq (LegId,Import_EventRecord)
ExportEvent_Record (LegId,Export_EventRecord)
ImportEventRecord (LegId,Import_EventRecord)
ExportLegResp (ExportEvent_Record) to cs
ImportLegResp to CS
InitialTssfActive := FALSE
Increment_OutstandingRequests (controllingLegID)
Waiting_For_Instruction
/* When the MoveCallSegments operation is proc the SSF−FSM may be requested to renumber the sends the signal SetLegID to the SSF−FSM. */
*
SetLegID (currentLegID, newLegID)
ExportEventRecord (currentLegID, exportEventRecord)
Renumber the leg by first exporting it and then importing it with the new legID.
ImportEventRecord (newLegID, exportEventRecord)
−

Annex A:

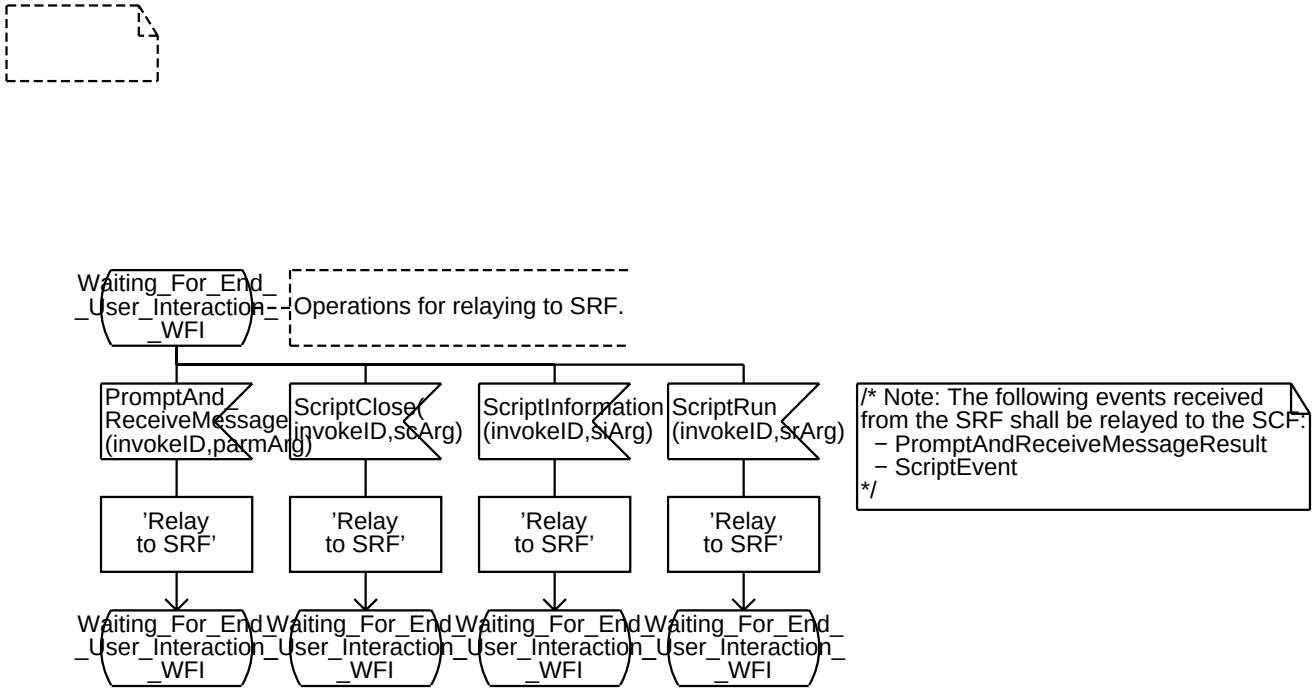

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM9(16)
/***** EXTENSIONS TO THE WAITING FOR INSTRUCTION STATE AS REQUIRED BY IN CS-
Waiting_for_ _Instruction
ContinueWith_ Argument (invokeID,,ctwaArg)
Disconnect_ Leg (invokeID,,dlArg)
RequestReport_ UTSI (invokeID,, rrutsiArg)
SendSTUI (invokeID,, sstuiArg)
Process_ ContinueWithArgument (ctwaArg)
Process_ DisconnectLeg (dlArg)
Process_ RequestReport_ UTSI (rrutsiArg)
Process_ SendSTUI (sstuiArg)
call OustandingRequests
true
false
Waiting_for_ _Instruction
InitialTssfActive := FALSE
call AnyDPArmed
true
false
CallInformationReportPending OR ApplyChargingReportPending
true
false
Application_ Active
true
false
Reset(Tssf)
termination :=true
InitialTssf_ Active := FALSE
Application_ End(termination,) to CS
Monitoring
Idle
Idle
Waiting_For_ _Instruction

Annex A:

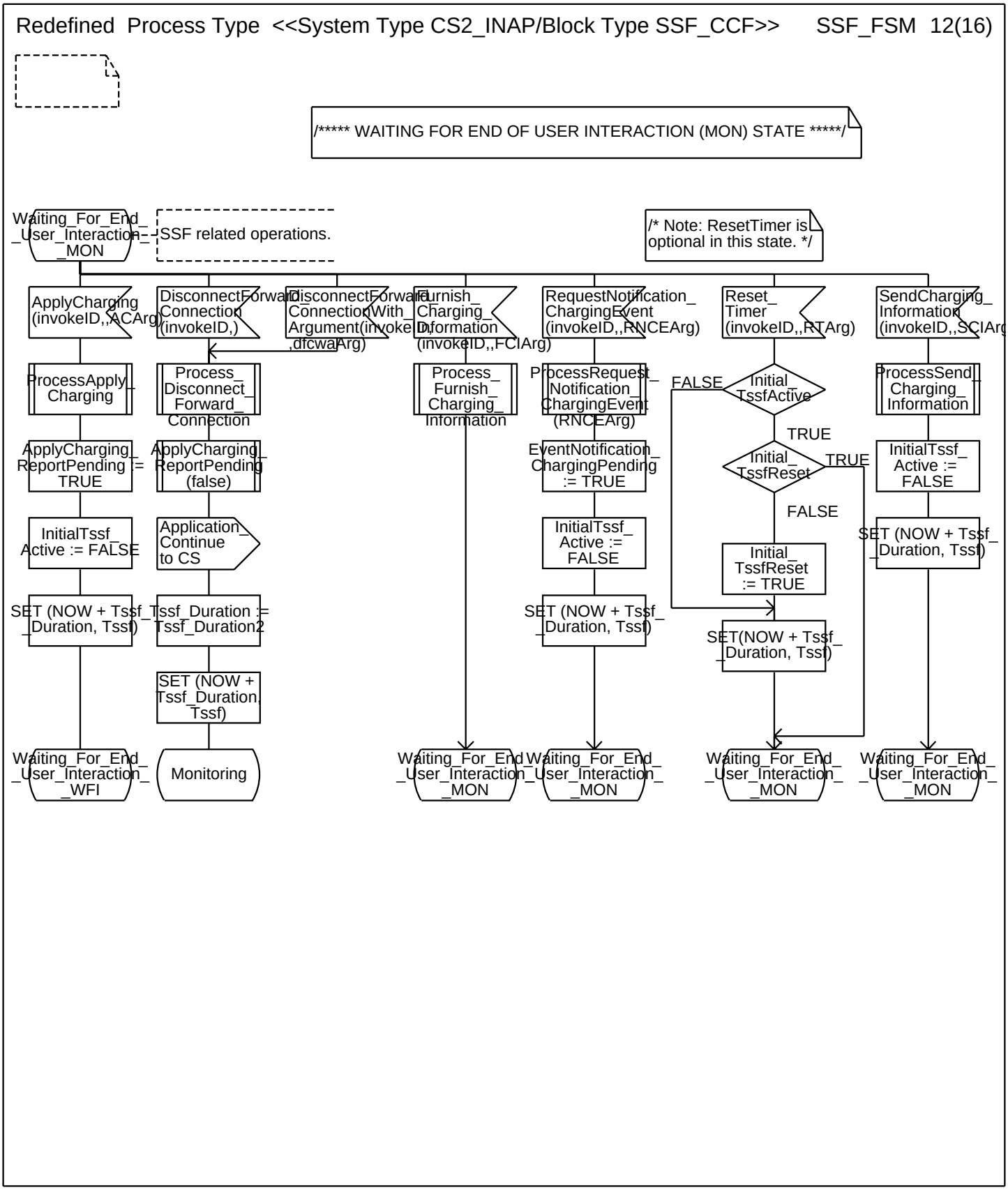

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSI10(16)
/***** EXTENSIONS TO THE WAITING FOR END USER INTERACTION STATE AS REQUIRED BY IN C
Waiting_For_End_ _User_Interaction_ _WFI
Continue (invokeID,)
ContinueWith_ Argument (invokeID,,ctwaArg)
Process_ Continue
Process_ ContinueWithArgument (ctwaArg)
call OustandingRequests
true
false
call AnyDPArmed
Waiting_For_End_ _User_Interaction_ _WFI
true
false
CallInformationReportPending OR ApplyChargingReportPending
true
false
Application_ Active
true
false
Reset(Tssf)
termination :=true
InitialTssf_ Active := FALSE
Application_ End(termination,) to CS
Waiting_For_End_ _User_Interaction_ _MON
Idle
Idle

Annex A:

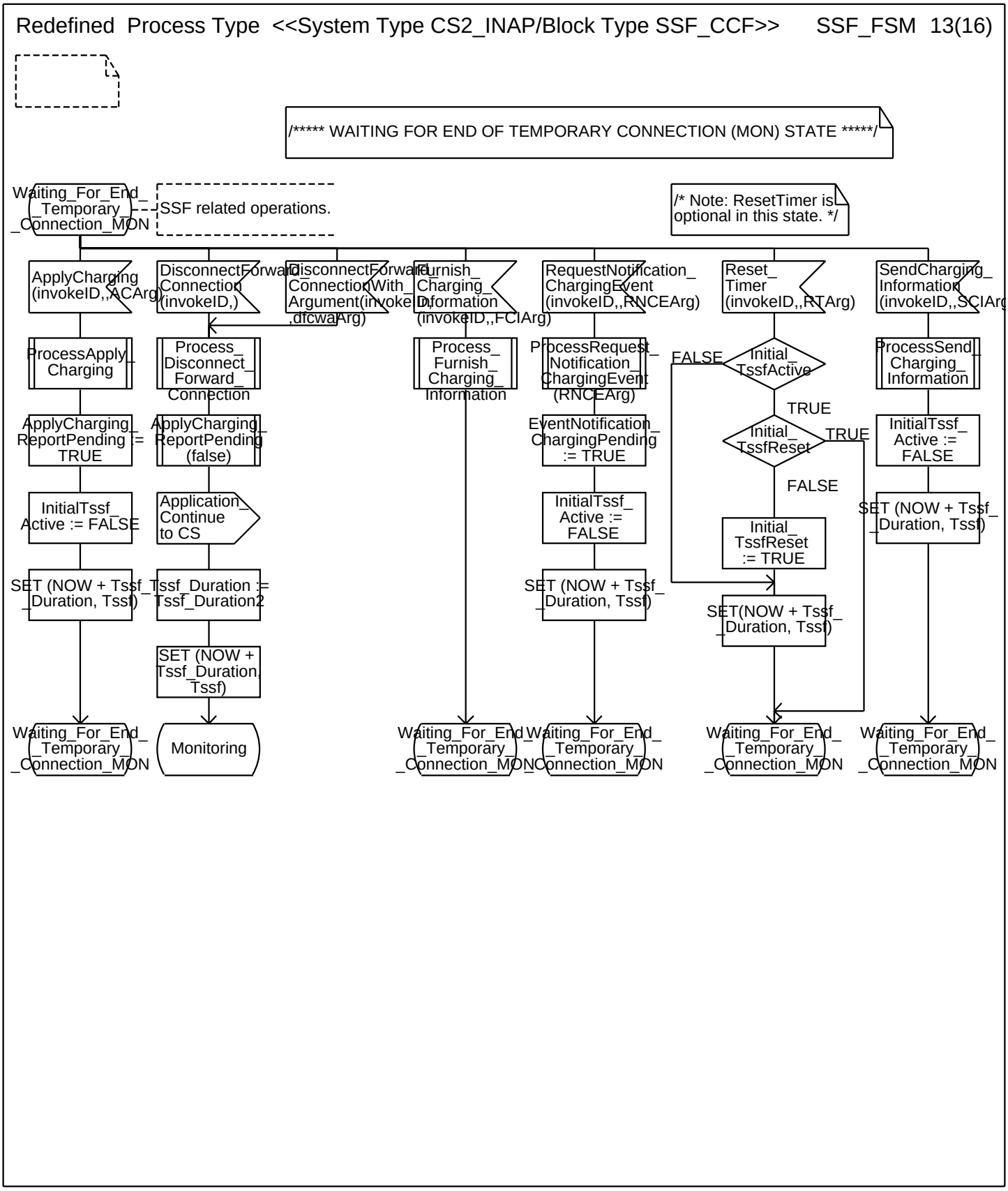
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSI11(16)



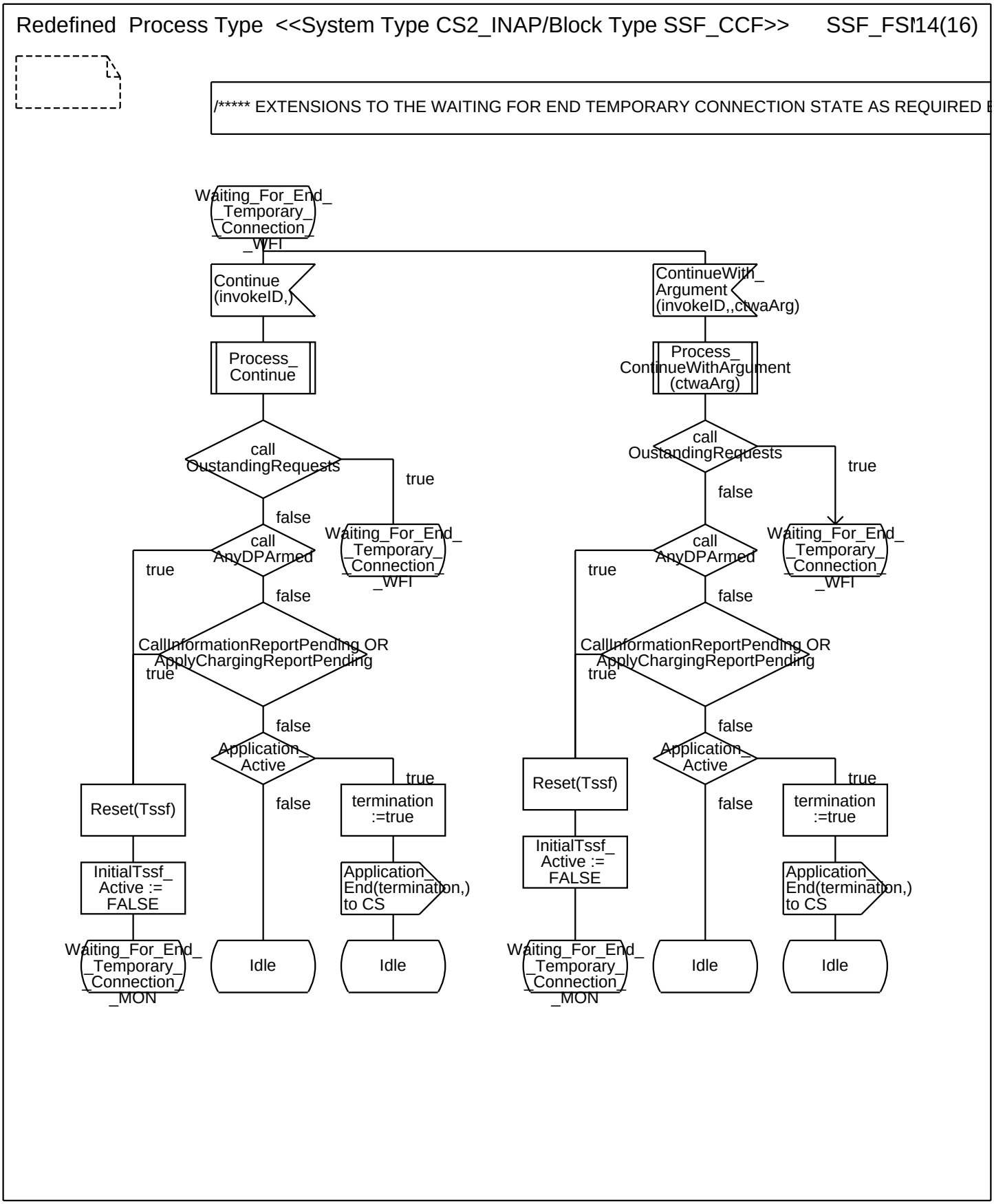
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM 12(16)
/***** WAITING FOR END OF USER INTERACTION (MON) STATE *****/
Waiting_For_End_User_Interaction_MON
SSF related operations.
/* Note: ResetTimer is optional in this state. */
ApplyCharging (invokeID,,ACArg)
DisconnectForward_Connection (invokeID,)
DisconnectForward_ConnectionWith_Argument(invokeID,,dfcwaArg)
Furnish_Charging_Information (invokeID,,FCIArg)
RequestNotification_ChargingEvent (invokeID,,RNCEArg)
Reset_Timer (invokeID,,RTArg)
SendCharging_Information (invokeID,,SCIArg
ProcessApply_Charging
Process_Disconnect_Forward_Connection
Process_Furnish_Charging_Information
ProcessRequest_Notification_ChargingEvent (RNCEArg)
FALSE
Initial_TssfActive
ProcessSend_Charging_Information
TRUE
ApplyCharging_ReportPending := TRUE
ApplyCharging_ReportPending (false)
EventNotification_ChargingPending := TRUE
Initial_TssfReset
TRUE
InitialTssf_Active := FALSE
FALSE
InitialTssf_Active := FALSE
Application_Continue to CS
Initial_TssfReset := TRUE
SET (NOW + Tssf_Duration, Tssf)
SET (NOW + Tssf_Duration, Tssf)
Tssf_Duration := Tssf_Duration2
SET (NOW + Tssf_Duration, Tssf)
SET(NOW + Tssf_Duration, Tssf)
SET (NOW + Tssf_Duration, Tssf)
Waiting_For_End_User_Interaction_WFI
Monitoring
Waiting_For_End_User_Interaction_MON
Waiting_For_End_User_Interaction_MON
Waiting_For_End_User_Interaction_MON
Waiting_For_End_User_Interaction_MON

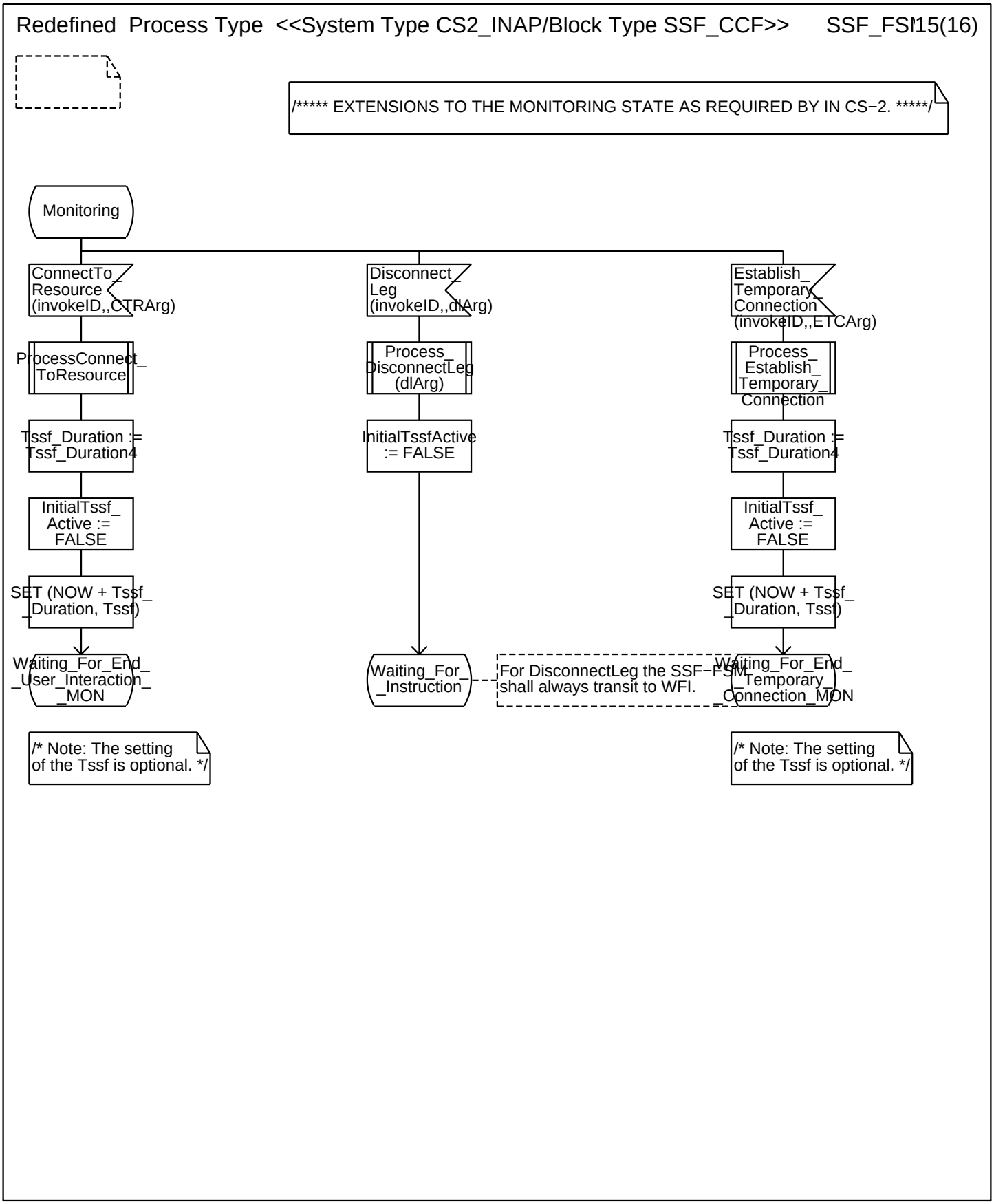
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSM 13(16)
/***** WAITING FOR END OF TEMPORARY CONNECTION (MON) STATE *****/
Waiting_For_End_ _Temporary_ _Connection_MON
SSF related operations.
/* Note: ResetTimer is optional in this state. */
ApplyCharging (invokeID,,ACArg)
DisconnectForward_ Connection (invokeID,)
DisconnectForward_ ConnectionWith_ Argument(invokeID, ,dfcwaArg)
Furnish_ Charging_ Information (invokeID,,FCIArg)
RequestNotification_ ChargingEvent (invokeID,,RNCEArg)
Reset_ Timer (invokeID,,RTArg)
SendCharging_ Information (invokeID,,SCIArg
ProcessApply_ Charging
Process_ Disconnect_ Forward_ Connection
Process_ Furnish_ Charging_ Information
ProcessRequest_ Notification_ ChargingEvent (RNCEArg)
FALSE
Initial_ TssfActive
ProcessSend_ Charging_ Information
TRUE
ApplyCharging_ ReportPending := TRUE
ApplyCharging_ ReportPending (false)
EventNotification_ ChargingPending := TRUE
Initial_ TssfReset
TRUE
InitialTssf_ Active := FALSE
FALSE
InitialTssf_ Active := FALSE
InitialTssf_ Active := FALSE
Application_ Continue to CS
Initial_ TssfReset := TRUE
SET (NOW + Tssf_ _Duration, Tssf)
SET (NOW + Tssf_ _Duration, Tssf)
Tssf_Duration := Tssf_Duration2
SET (NOW + Tssf_ _Duration, Tssf)
SET(NOW + Tssf_ _Duration, Tssf)
SET (NOW + Tssf_Duration, Tssf)
Waiting_For_End_ _Temporary_ _Connection_MON
Monitoring
Waiting_For_End_ _Temporary_ _Connection_MON
Waiting_For_End_ _Temporary_ _Connection_MON
Waiting_For_End_ _Temporary_ _Connection_MON
Waiting_For_End_ _Temporary_ _Connection_MON

Annex A:

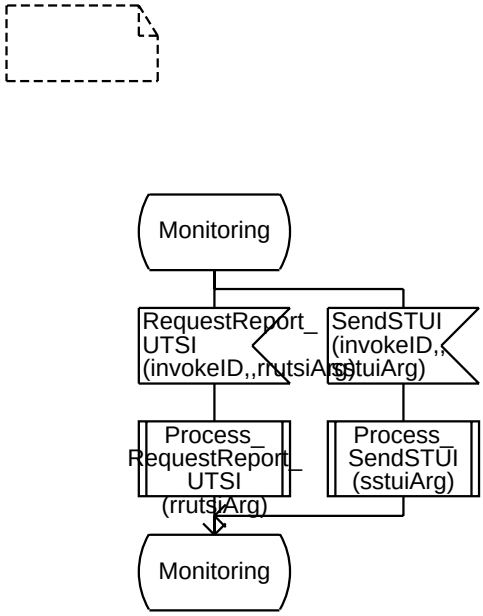

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSI14(16)
/***** EXTENSIONS TO THE WAITING FOR END TEMPORARY CONNECTION STATE AS REQUIRED
Waiting_For_End_ _Temporary_ _Connection_ _WFI
Continue (invokeID,)
Process_ Continue
call OustandingRequests
true
false
Waiting_For_End_ _Temporary_ _Connection_ _WFI
call AnyDPArmed
true
false
CallInformationReportPending OR ApplyChargingReportPending
true
false
Application_ Active
true
false
Reset(Tssf)
termination :=true
InitialTssf_ Active := FALSE
Application_ End(termination,) to CS
Waiting_For_End_ _Temporary_ _Connection_ _MON
Idle
Idle
ContinueWith_ Argument (invokeID,,ctwaArg)
Process_ ContinueWithArgument (ctwaArg)
call OustandingRequests
true
false
Waiting_For_End_ _Temporary_ _Connection_ _WFI
call AnyDPArmed
true
false
CallInformationReportPending OR ApplyChargingReportPending
true
false
Application_ Active
true
false
Reset(Tssf)
termination :=true
InitialTssf_ Active := FALSE
Application_ End(termination,) to CS
Waiting_For_End_ _Temporary_ _Connection_ _MON
Idle
Idle

Annex A:

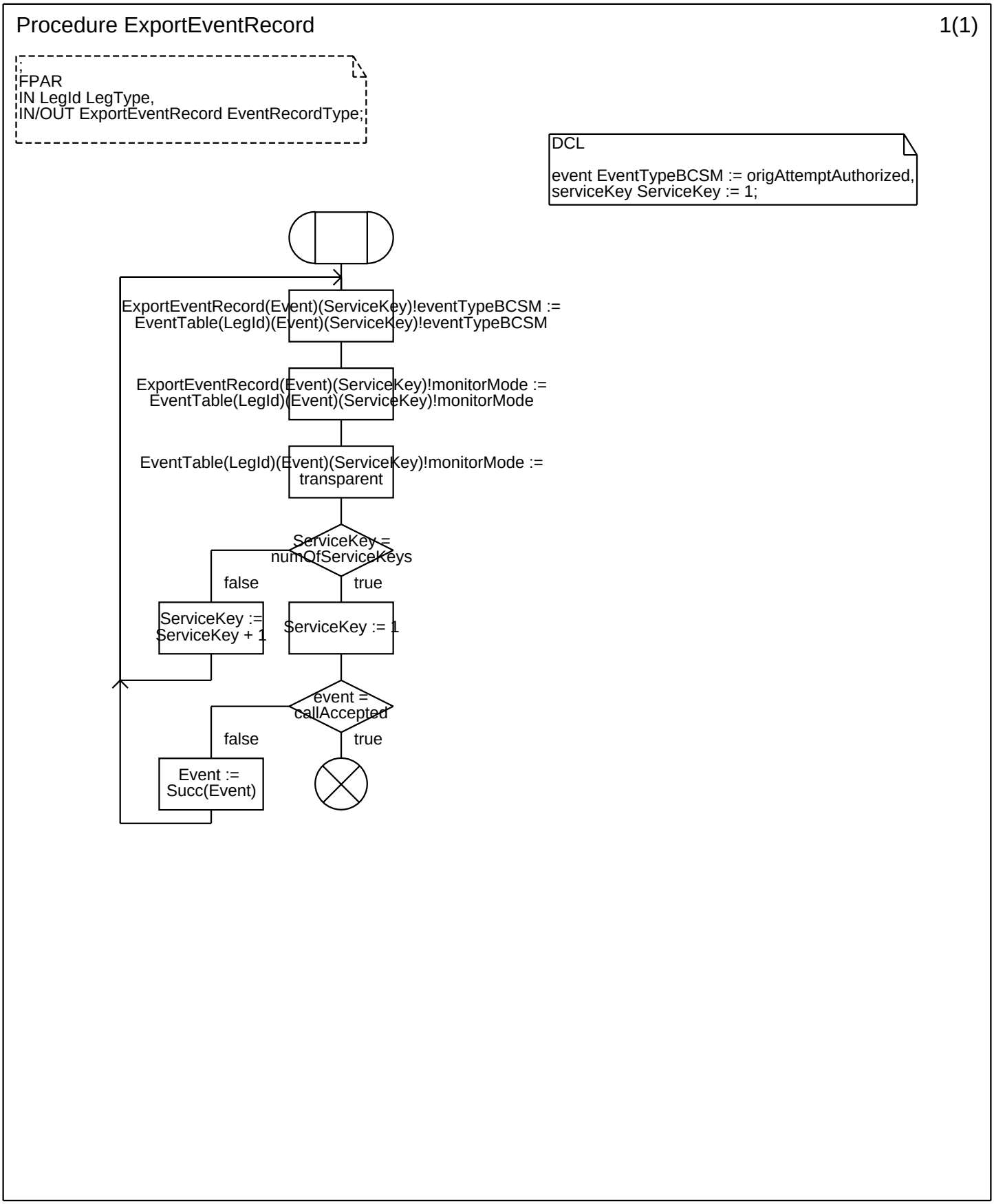

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSI15(16)
/***** EXTENSIONS TO THE MONITORING STATE AS REQUIRED BY IN CS−2. *****/
Monitoring
ConnectTo_ Resource (invokeID,,CTRArg)
Disconnect_ Leg (invokeID,,dlArg)
Establish_ Temporary_ Connection (invokeID,,ETCArg)
ProcessConnect_ ToResource
Process_ DisconnectLeg (dlArg)
Process_ Establish_ Temporary_ Connection
Tssf_Duration := Tssf_Duration4
InitialTssfActive := FALSE
Tssf_Duration := Tssf_Duration4
InitialTssf_ Active := FALSE
InitialTssf_ Active := FALSE
SET (NOW + Tssf_ _Duration, Tssf)
SET (NOW + Tssf_ _Duration, Tssf)
Waiting_For_End_ _User_Interaction_ _MON
Waiting_For_ _Instruction
For DisconnectLeg the SSF−FSM shall always transit to WFI.
Waiting_For_End_ _Temporary_ _Connection_MON
/* Note: The setting of the Tssf is optional. */
/* Note: The setting of the Tssf is optional. */

Annex A:

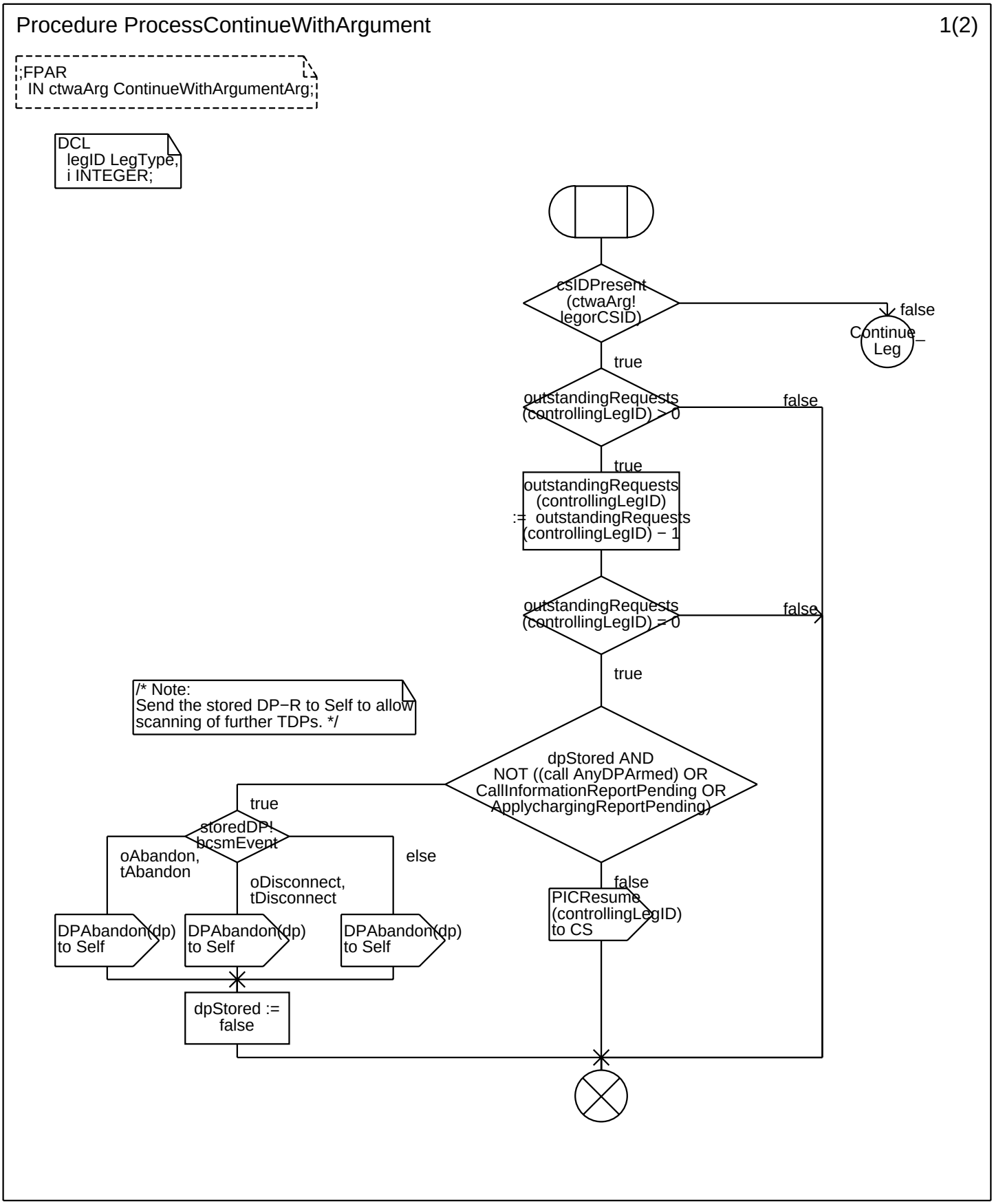
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> SSF_FSI16(16)



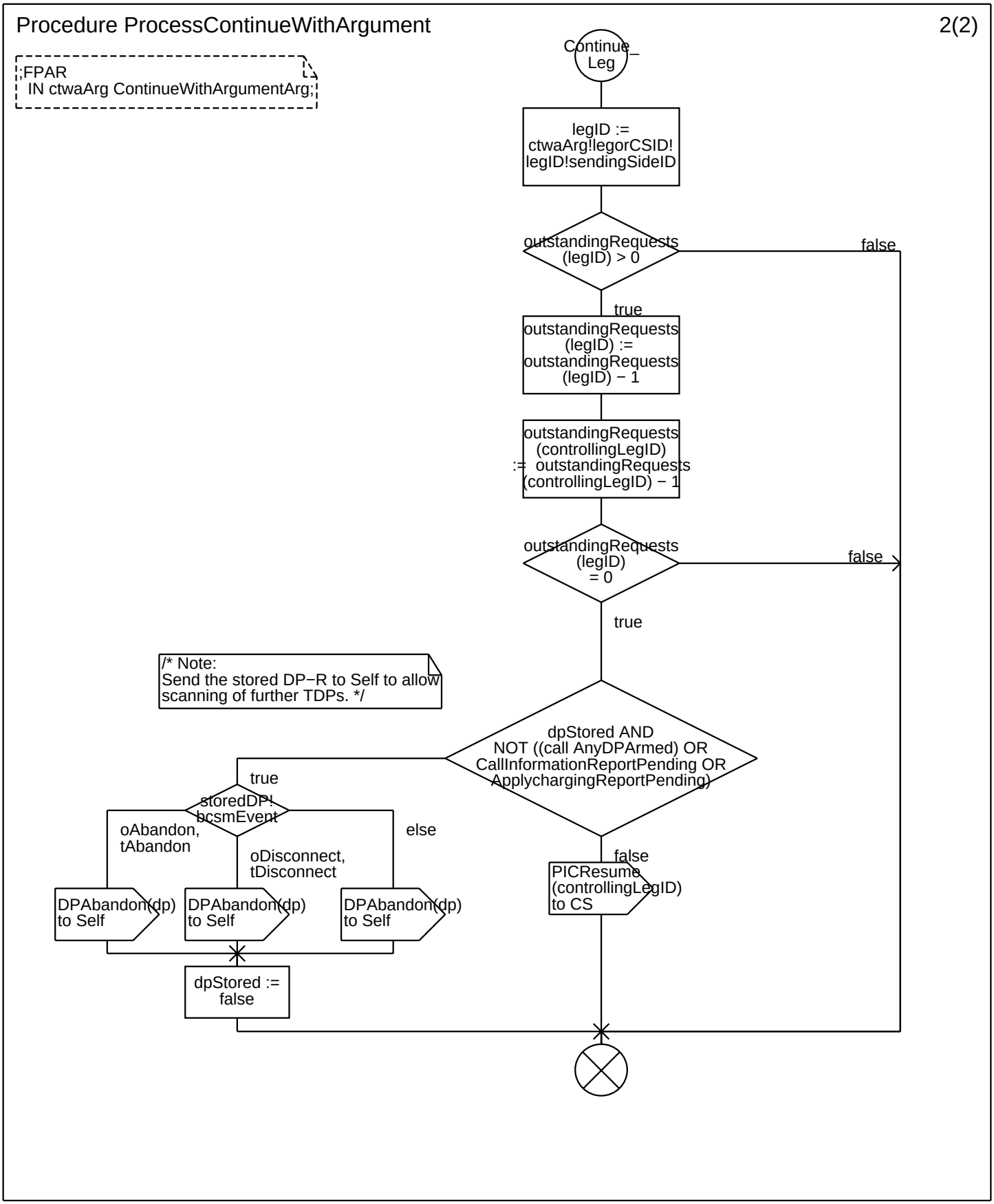
Annex A:


Procedure ExportEventRecord
1(1)
;
FPAR
IN LegId LegType,
IN/OUT ExportEventRecord EventRecordType;
DCL
event EventTypeBCSM := origAttemptAuthorized,
serviceKey ServiceKey := 1;
ExportEventRecord(Event)(ServiceKey)!eventTypeBCSM :=
EventTable(LegId)(Event)(ServiceKey)!eventTypeBCSM
ExportEventRecord(Event)(ServiceKey)!monitorMode :=
EventTable(LegId)(Event)(ServiceKey)!monitorMode
EventTable(LegId)(Event)(ServiceKey)!monitorMode :=
transparent
ServiceKey =
numOfServiceKeys
false
true
ServiceKey :=
ServiceKey + 1
ServiceKey := 1
event =
callAccepted
false
true
Event :=
Succ(Event)

Annex A:

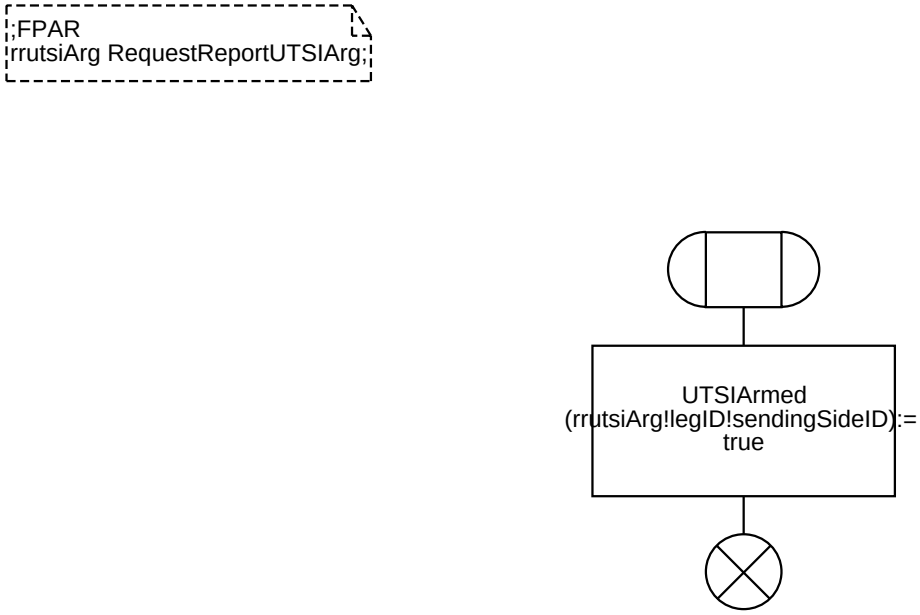

Procedure ProcessContinueWithArgument
1(2)
;FPAR
IN ctwaArg ContinueWithArgumentArg;
DCL
legID LegType;
i INTEGER;
csIDPresent (ctwaArg! legorCSID)
false
Continue_ Leg
true
outstandingRequests (controllingLegID) > 0
false
true
outstandingRequests (controllingLegID) := outstandingRequests (controllingLegID) − 1
outstandingRequests (controllingLegID) = 0
false
true
/* Note:
Send the stored DP−R to Self to allow scanning of further TDPs. */
dpStored AND NOT ((call AnyDPArmed) OR CallInformationReportPending OR ApplychargingReportPending)
true
storedDP! bcsmEvent
oAbandon, tAbandon
oDisconnect, tDisconnect
else
false
PICResume (controllingLegID) to CS
DPAbandon(dp) to Self
DPAbandon(dp) to Self
DPAbandon(dp) to Self
dpStored := false

Annex A:


Procedure ProcessContinueWithArgument
2(2)
;FPAR
IN ctwaArg ContinueWithArgumentArg;
Continue Leg
legID := ctwaArg!legorCSID! legID!sendingSideID
outstandingRequests (legID) > 0
false
true
outstandingRequests (legID) := outstandingRequests (legID) − 1
outstandingRequests (controllingLegID) := outstandingRequests (controllingLegID) − 1
outstandingRequests (legID) = 0
false
true
/* Note:
Send the stored DP−R to Self to allow scanning of further TDPs. */
dpStored AND NOT ((call AnyDPArmed) OR CallInformationReportPending OR ApplychargingReportPending)
true
storedDP! bcsmEvent
oAbandon, tAbandon
oDisconnect, tDisconnect
else
false
PICResume (controllingLegID) to CS
DPAbandon(dp) to Self
DPAbandon(dp) to Self
DPAbandon(dp) to Self
dpStored := false

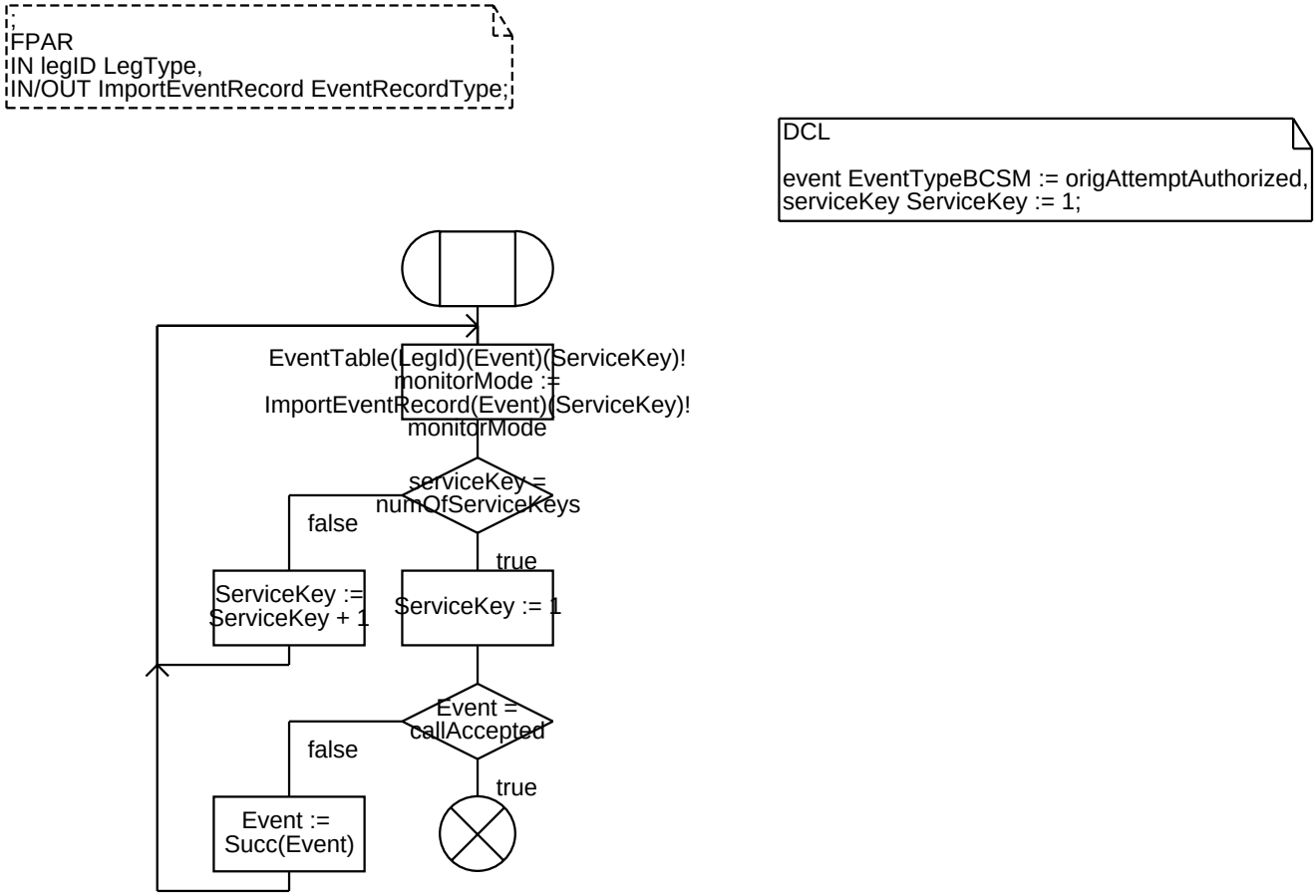
Annex A:

Procedure ProcessRequestReportUTSI 1(1)

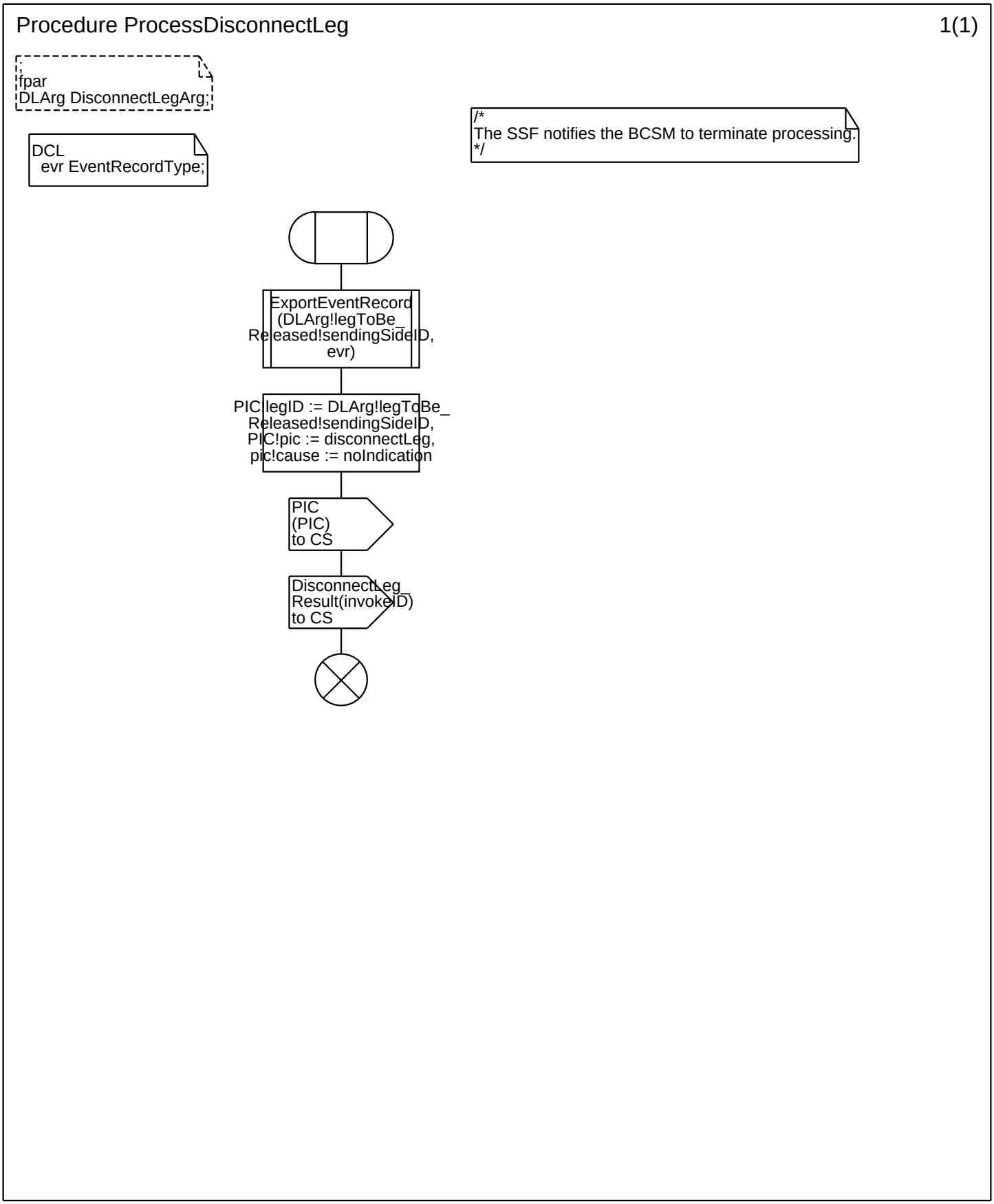


Annex A:

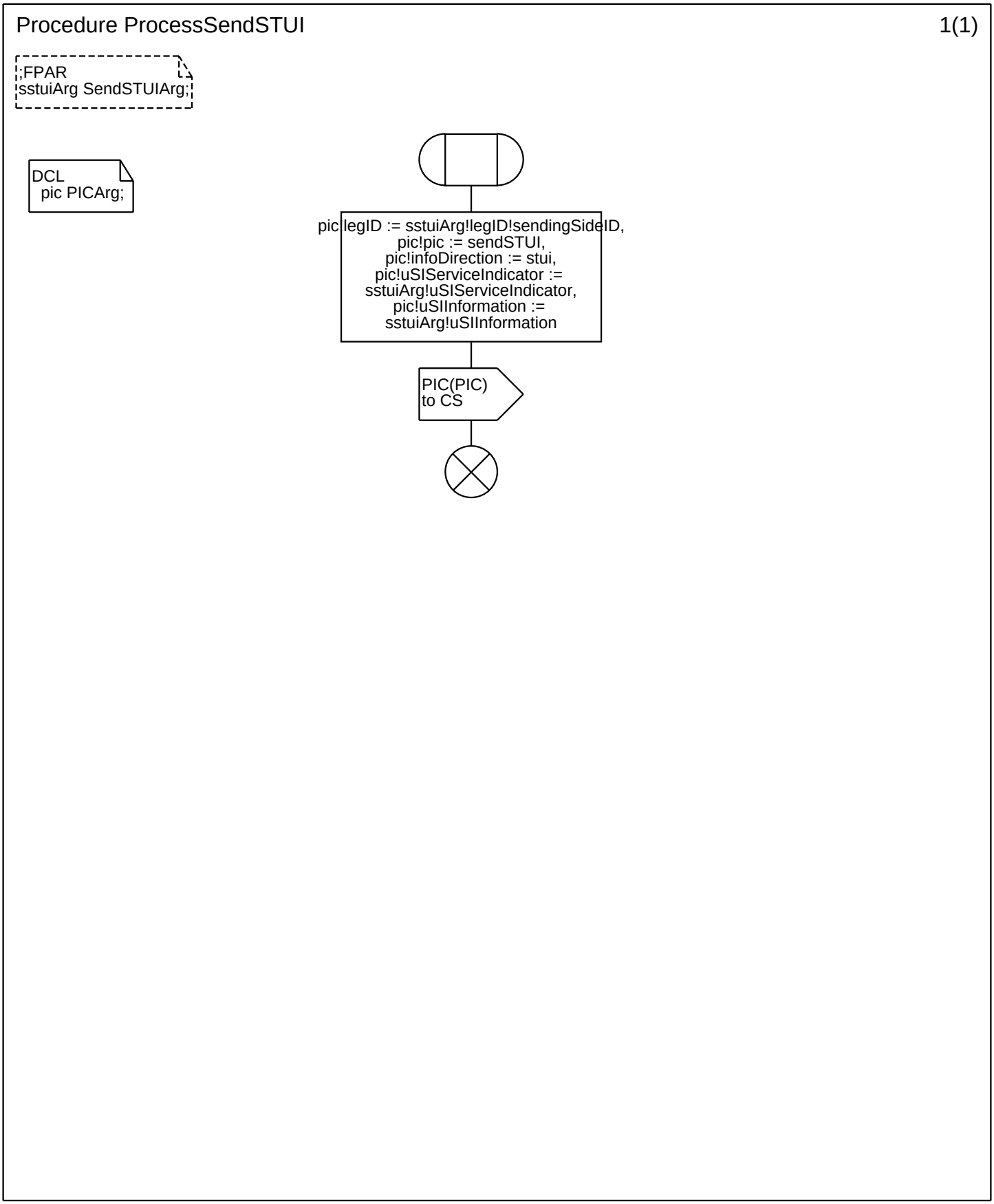
Procedure ImportEventRecord 1(1)



Annex A:

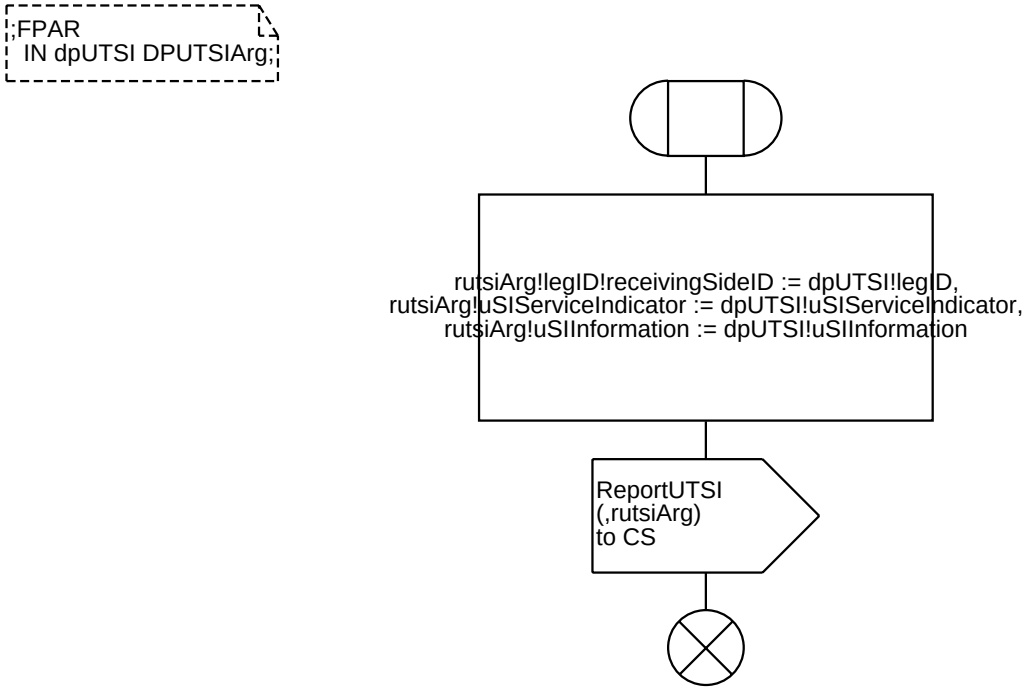

Procedure ProcessDisconnectLeg
1(1)
;
fpar
DLArg DisconnectLegArg;
DCL
evr EventRecordType;
/*
The SSF notifies the BCSM to terminate processing.
*/
ExportEventRecord
(DLArg!legToBe_
Released!sendingSideID,
evr)
PIC legID := DLArg!legToBe_
Released!sendingSideID,
PIC!pic := disconnectLeg,
pic!cause := noIndication
PIC
(PIC)
to CS
DisconnectLeg_
Result(invokeID)
to CS

Annex A:


Procedure ProcessSendSTUI
1(1)
;FPAR
sstuiArg SendSTUIArg;
DCL
pic PICArg;
pic!legID := sstuiArg!legID!sendingSideID,
pic!pic := sendSTUI,
pic!infoDirection := stui,
pic!uSIServiceIndicator :=
sstuiArg!uSIServiceIndicator,
pic!uSIInformation :=
sstuiArg!uSIInformation
PIC(PIC)
to CS

Annex A:

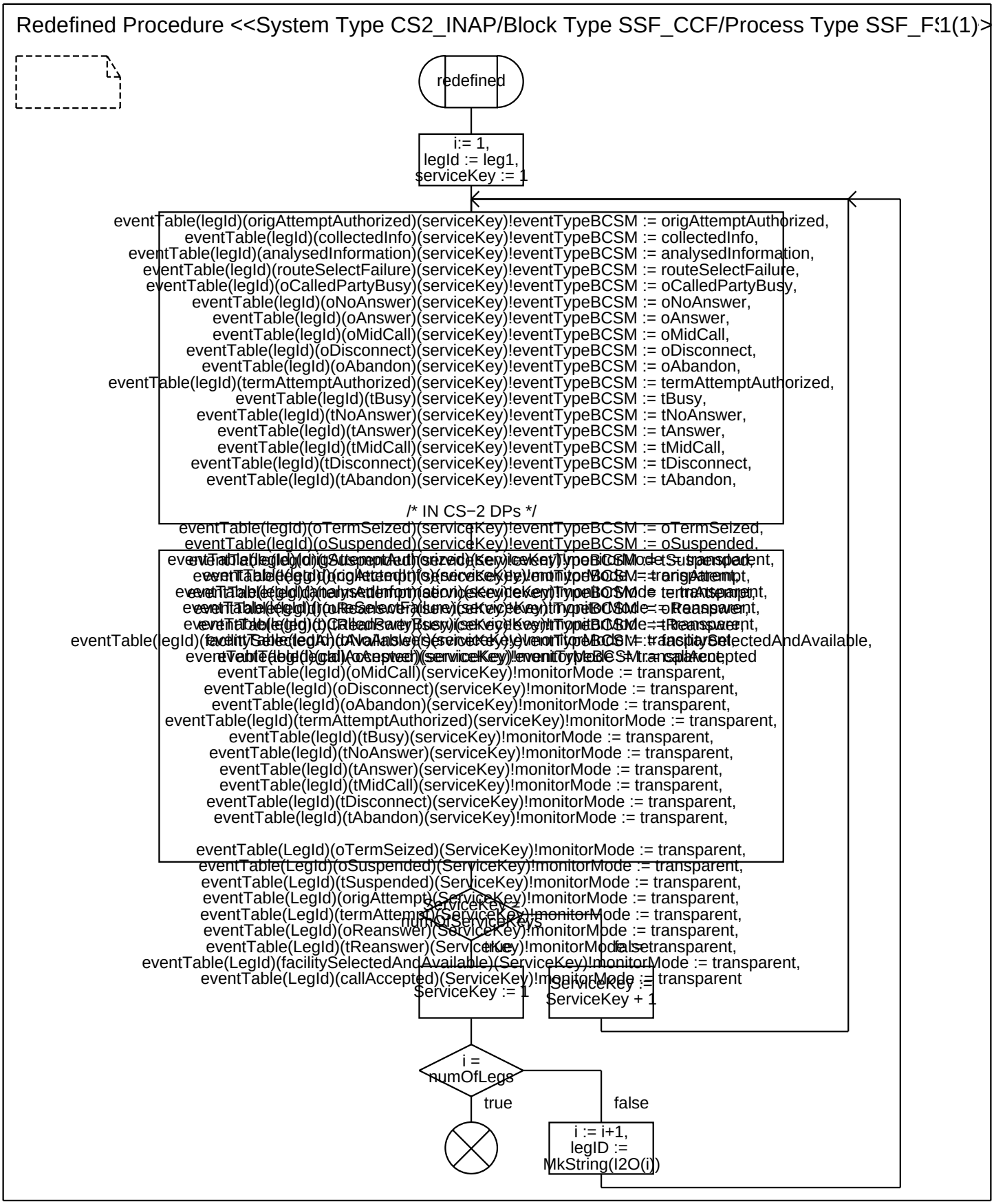
Procedure ProcessReportUTSI 1(1)



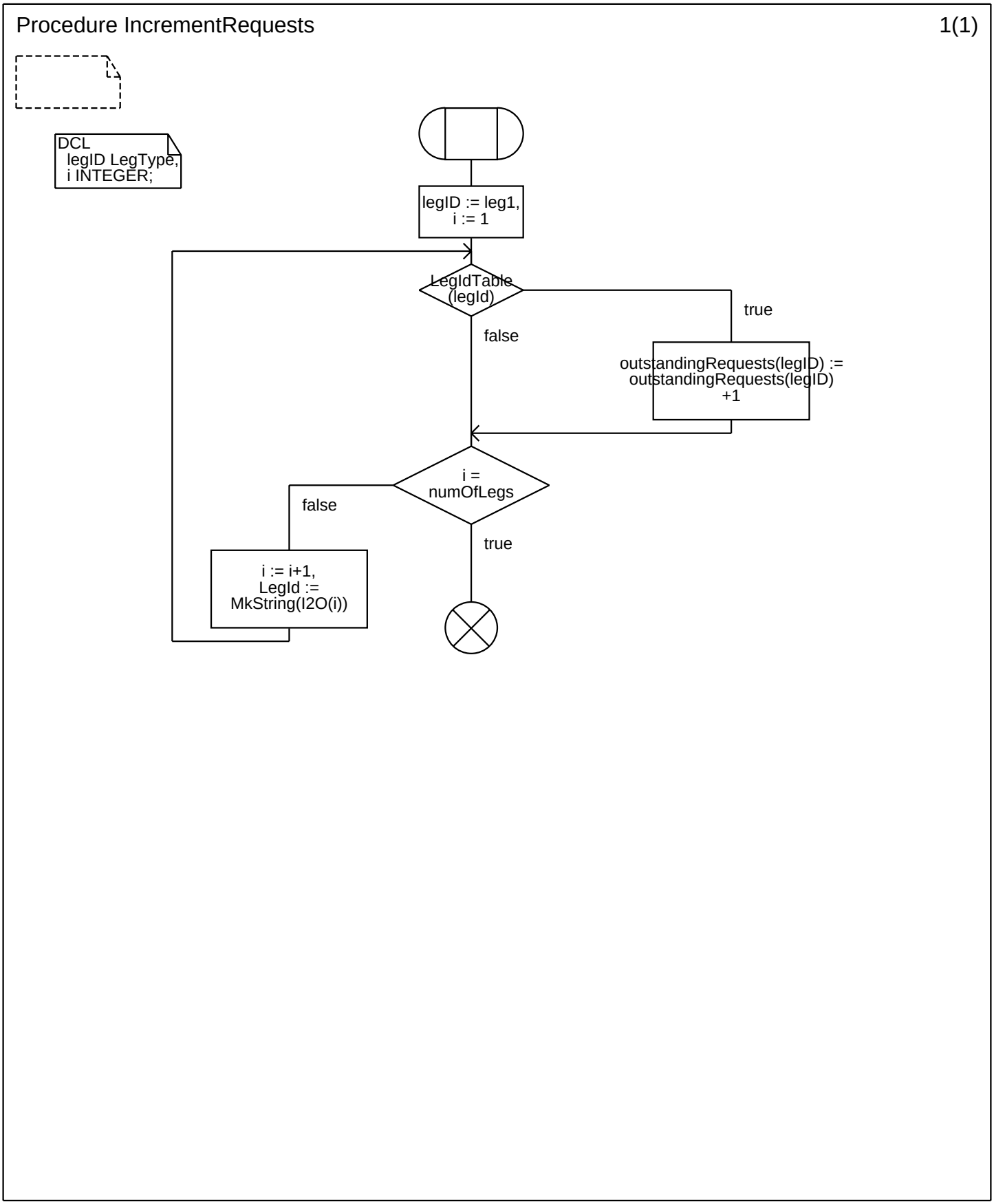
Annex A:

Procedure IsUTSIArmed 1(1)

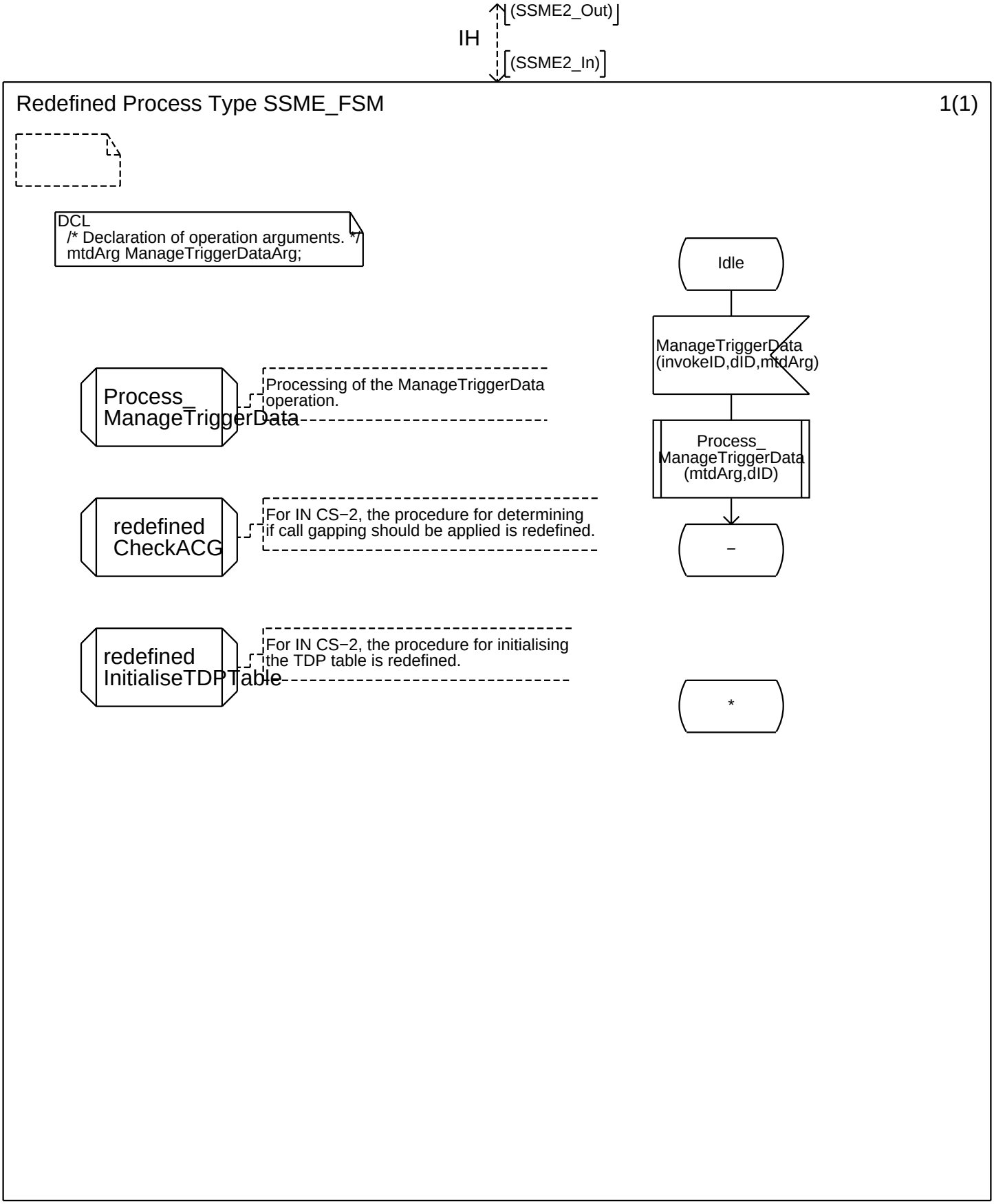
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type SSF_F
redefined
i:= 1,
legId := leg1,
serviceKey := 1
eventTable(legId)(origAttemptAuthorized)(serviceKey)!eventTypeBCSM := origAttemptAuthorized,
eventTable(legId)(collectedInfo)(serviceKey)!eventTypeBCSM := collectedInfo,
eventTable(legId)(analysedInformation)(serviceKey)!eventTypeBCSM := analysedInformation,
eventTable(legId)(routeSelectFailure)(serviceKey)!eventTypeBCSM := routeSelectFailure,
eventTable(legId)(oCalledPartyBusy)(serviceKey)!eventTypeBCSM := oCalledPartyBusy,
eventTable(legId)(oNoAnswer)(serviceKey)!eventTypeBCSM := oNoAnswer,
eventTable(legId)(oAnswer)(serviceKey)!eventTypeBCSM := oAnswer,
eventTable(legId)(oMidCall)(serviceKey)!eventTypeBCSM := oMidCall,
eventTable(legId)(oDisconnect)(serviceKey)!eventTypeBCSM := oDisconnect,
eventTable(legId)(oAbandon)(serviceKey)!eventTypeBCSM := oAbandon,
eventTable(legId)(termAttemptAuthorized)(serviceKey)!eventTypeBCSM := termAttemptAuthorized,
eventTable(legId)(tBusy)(serviceKey)!eventTypeBCSM := tBusy,
eventTable(legId)(tNoAnswer)(serviceKey)!eventTypeBCSM := tNoAnswer,
eventTable(legId)(tAnswer)(serviceKey)!eventTypeBCSM := tAnswer,
eventTable(legId)(tMidCall)(serviceKey)!eventTypeBCSM := tMidCall,
eventTable(legId)(tDisconnect)(serviceKey)!eventTypeBCSM := tDisconnect,
eventTable(legId)(tAbandon)(serviceKey)!eventTypeBCSM := tAbandon,
/* IN CS−2 DPs */
eventTable(legId)(oTermSeized)(serviceKey)!eventTypeBCSM := oTermSeized,
eventTable(legId)(oSuspended)(serviceKey)!eventTypeBCSM := oSuspended,
eventTable(legId)(tSuspended)(serviceKey)!eventTypeBCSM := tSuspended,
eventTable(legId)(origAttempt)(serviceKey)!eventTypeBCSM := origAttempt,
eventTable(legId)(termAttempt)(serviceKey)!eventTypeBCSM := termAttempt,
eventTable(legId)(oReanswer)(serviceKey)!eventTypeBCSM := oReanswer,
eventTable(legId)(tReanswer)(serviceKey)!eventTypeBCSM := tReanswer,
eventTable(legId)(facilitySelectedAndAvailable)(serviceKey)!eventTypeBCSM := facilitySelectedAndAvailable,
eventTable(legId)(callAccepted)(serviceKey)!eventTypeBCSM := callAccepted
eventTable(legId)(oMidCall)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oDisconnect)(serviceKey)!monitorMode := transparent,
eventTable(legId)(oAbandon)(serviceKey)!monitorMode := transparent,
eventTable(legId)(termAttemptAuthorized)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tBusy)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tNoAnswer)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tAnswer)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tMidCall)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tDisconnect)(serviceKey)!monitorMode := transparent,
eventTable(legId)(tAbandon)(serviceKey)!monitorMode := transparent,
eventTable(LegId)(oTermSeized)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(oSuspended)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(tSuspended)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(origAttempt)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(termAttempt)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(oReanswer)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(tReanswer)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(facilitySelectedAndAvailable)(ServiceKey)!monitorMode := transparent,
eventTable(LegId)(callAccepted)(ServiceKey)!monitorMode := transparent
ServiceKey := 1
ServiceKey <
numOfServiceKeys
true
false
ServiceKey :=
ServiceKey + 1
i =
numOfLegs
true
false
i := i+1,
legID :=
MkString(I2O(i))

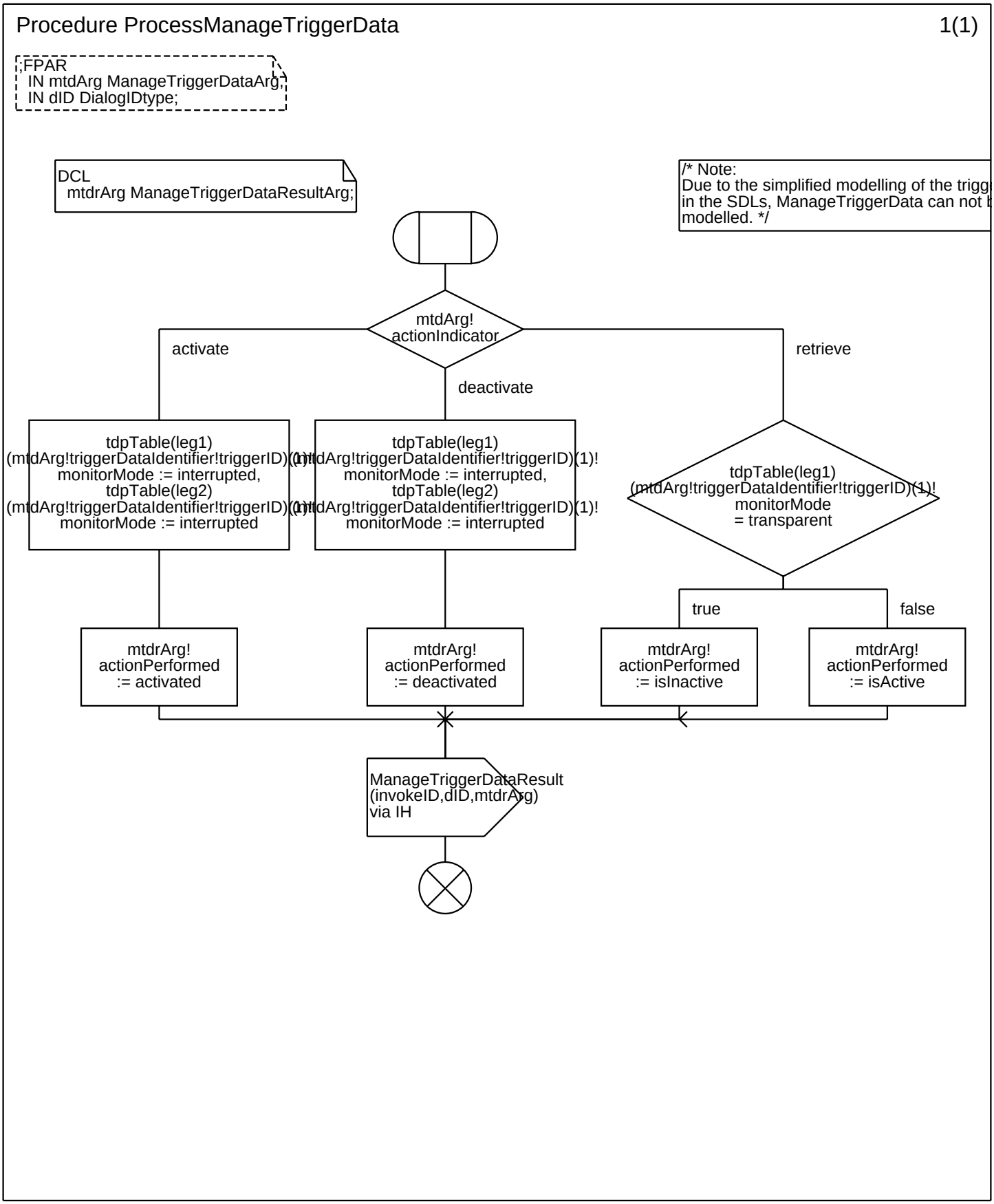
Annex A:


Procedure IncrementRequests
1(1)
DCL
legID LegType;
i INTEGER;
legID := leg1,
i := 1
LegIdTable
(legId)
true
false
outstandingRequests(legID) :=
outstandingRequests(legID)
+1
i =
numOfLegs
false
true
i := i+1,
LegId :=
MkString(I2O(i))

Annex A:

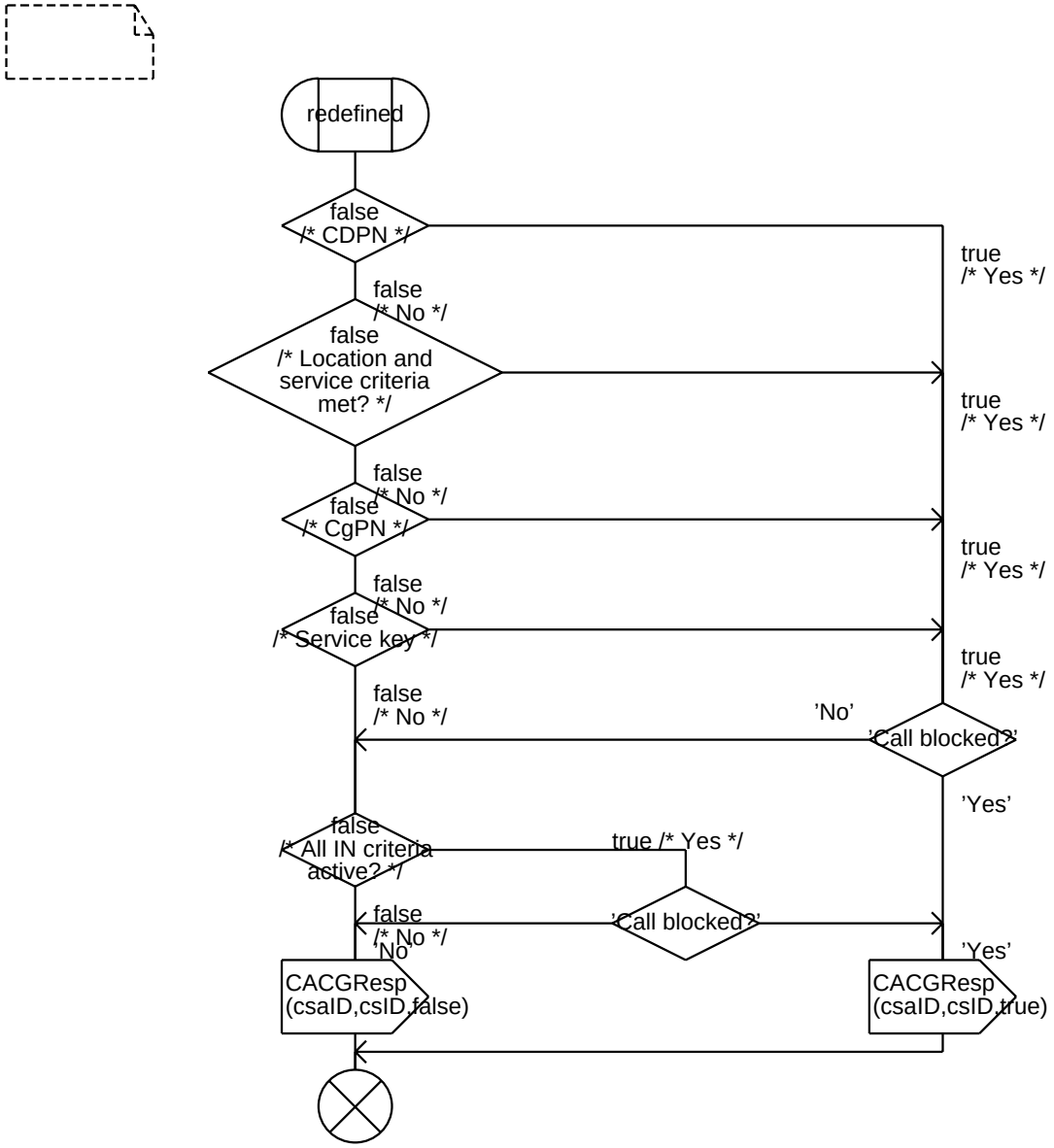

IH
(SSME2_Out)
(SSME2_In)
Redefined Process Type SSME_FSM
1(1)
DCL
/* Declaration of operation arguments. */
mtdArg ManageTriggerDataArg;
Idle
ManageTriggerData
(invokeID,dID,mtdArg)
Process_
ManageTriggerData
Processing of the ManageTriggerData operation.
Process_
ManageTriggerData
(mtdArg,dID)
redefined
CheckACG
For IN CS−2, the procedure for determining if call gapping should be applied is redefined.
−
redefined
InitialiseTDPTable
For IN CS−2, the procedure for initialising the TDP table is redefined.
*

Annex A:

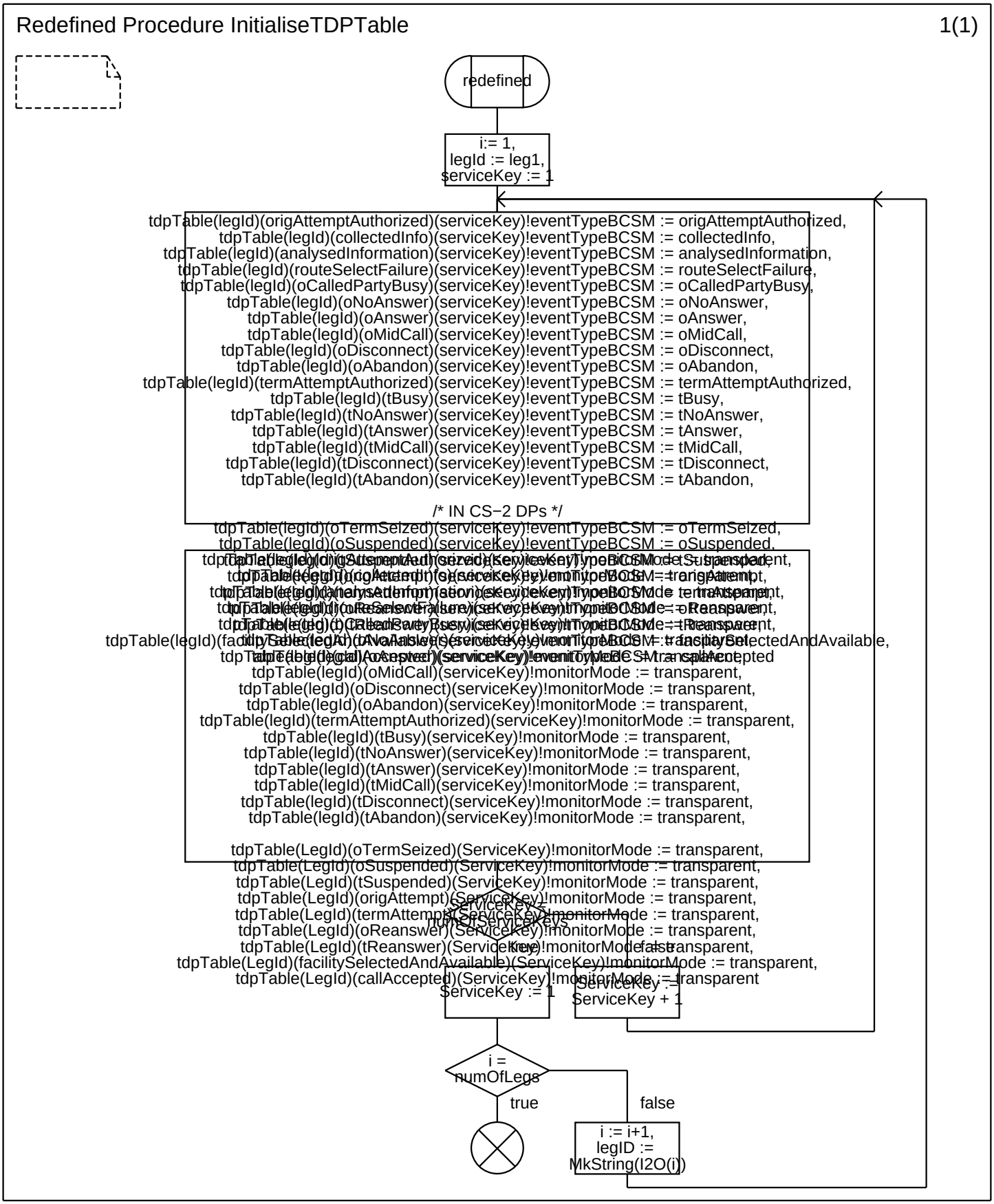

Procedure ProcessManageTriggerData
1(1)
;FPAR
IN mtdArg ManageTriggerDataArg;
IN dID DialogIDtype;
DCL
mtdrArg ManageTriggerDataResultArg;
/* Note:
Due to the simplified modelling of the trigg
in the SDLs, ManageTriggerData can not b
modelled. */
mtdArg!
actionIndicator
activate
deactivate
retrieve
tdpTable(leg1)
(mtdArg!triggerDataIdentifier!triggerID)(1)!
monitorMode := interrupted,
tdpTable(leg2)
(mtdArg!triggerDataIdentifier!triggerID)(1)!
monitorMode := interrupted
tdpTable(leg1)
(mtdArg!triggerDataIdentifier!triggerID)(1)!
monitorMode := interrupted,
tdpTable(leg2)
(mtdArg!triggerDataIdentifier!triggerID)(1)!
monitorMode := interrupted
tdpTable(leg1)
(mtdArg!triggerDataIdentifier!triggerID)(1)!
monitorMode
= transparent
true
false
mtdrArg!
actionPerformed
:= activated
mtdrArg!
actionPerformed
:= deactivated
mtdrArg!
actionPerformed
:= isInactive
mtdrArg!
actionPerformed
:= isActive
ManageTriggerDataResult
(invokeID,dID,mtdrArg)
via IH

Annex A:

Redefined Procedure CheckACG 1(1)



Annex A:


Redefined Procedure InitialiseTDPTable
1(1)
redefined
i:= 1,
legId := leg1,
serviceKey := 1
tdpTable(legId)(origAttemptAuthorized)(serviceKey)!eventTypeBCSM := origAttemptAuthorized,
tdpTable(legId)(collectedInfo)(serviceKey)!eventTypeBCSM := collectedInfo,
tdpTable(legId)(analysedInformation)(serviceKey)!eventTypeBCSM := analysedInformation,
tdpTable(legId)(routeSelectFailure)(serviceKey)!eventTypeBCSM := routeSelectFailure,
tdpTable(legId)(oCalledPartyBusy)(serviceKey)!eventTypeBCSM := oCalledPartyBusy,
tdpTable(legId)(oNoAnswer)(serviceKey)!eventTypeBCSM := oNoAnswer,
tdpTable(legId)(oAnswer)(serviceKey)!eventTypeBCSM := oAnswer,
tdpTable(legId)(oMidCall)(serviceKey)!eventTypeBCSM := oMidCall,
tdpTable(legId)(oDisconnect)(serviceKey)!eventTypeBCSM := oDisconnect,
tdpTable(legId)(oAbandon)(serviceKey)!eventTypeBCSM := oAbandon,
tdpTable(legId)(termAttemptAuthorized)(serviceKey)!eventTypeBCSM := termAttemptAuthorized,
tdpTable(legId)(tBusy)(serviceKey)!eventTypeBCSM := tBusy,
tdpTable(legId)(tNoAnswer)(serviceKey)!eventTypeBCSM := tNoAnswer,
tdpTable(legId)(tAnswer)(serviceKey)!eventTypeBCSM := tAnswer,
tdpTable(legId)(tMidCall)(serviceKey)!eventTypeBCSM := tMidCall,
tdpTable(legId)(tDisconnect)(serviceKey)!eventTypeBCSM := tDisconnect,
tdpTable(legId)(tAbandon)(serviceKey)!eventTypeBCSM := tAbandon,
/* IN CS−2 DPs */
tdpTable(legId)(oTermSeized)(serviceKey)!eventTypeBCSM := oTermSeized,
tdpTable(legId)(oSuspended)(serviceKey)!eventTypeBCSM := oSuspended,
tdpTable(legId)(origAttempt)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(termAttempt)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oReanswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tReanswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(facilitySelectedAndAvailable)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(callAccepted)(serviceKey)!monitorMode := transparent
tdpTable(legId)(oMidCall)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oDisconnect)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(oAbandon)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(termAttemptAuthorized)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tBusy)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tNoAnswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tAnswer)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tMidCall)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tDisconnect)(serviceKey)!monitorMode := transparent,
tdpTable(legId)(tAbandon)(serviceKey)!monitorMode := transparent,
tdpTable(LegId)(oTermSeized)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(oSuspended)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(tSuspended)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(origAttempt)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(termAttempt)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(oReanswer)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(tReanswer)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(facilitySelectedAndAvailable)(ServiceKey)!monitorMode := transparent,
tdpTable(LegId)(callAccepted)(ServiceKey)!monitorMode := transparent
ServiceKey := 1
ServiceKey := ServiceKey + 1
ServiceKey = numOfServiceKeys
false
i = numOfLegs
true
false
i := i+1,
legID := MkString(I2O(i))

Annex A:

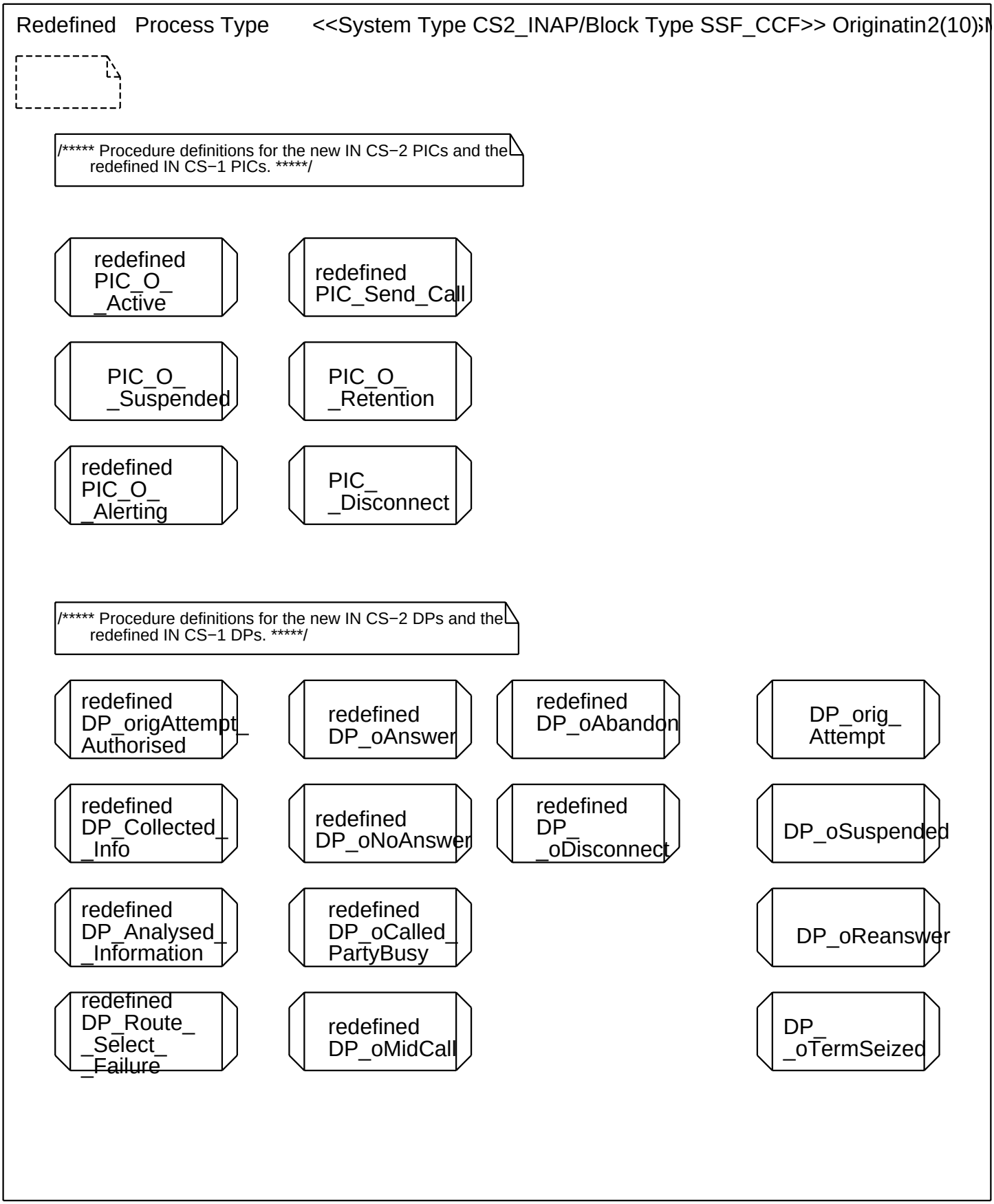
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> Originatin1(10);N

```
/***** O−BCSM FOR CORE INAP CS−2. *****/
```

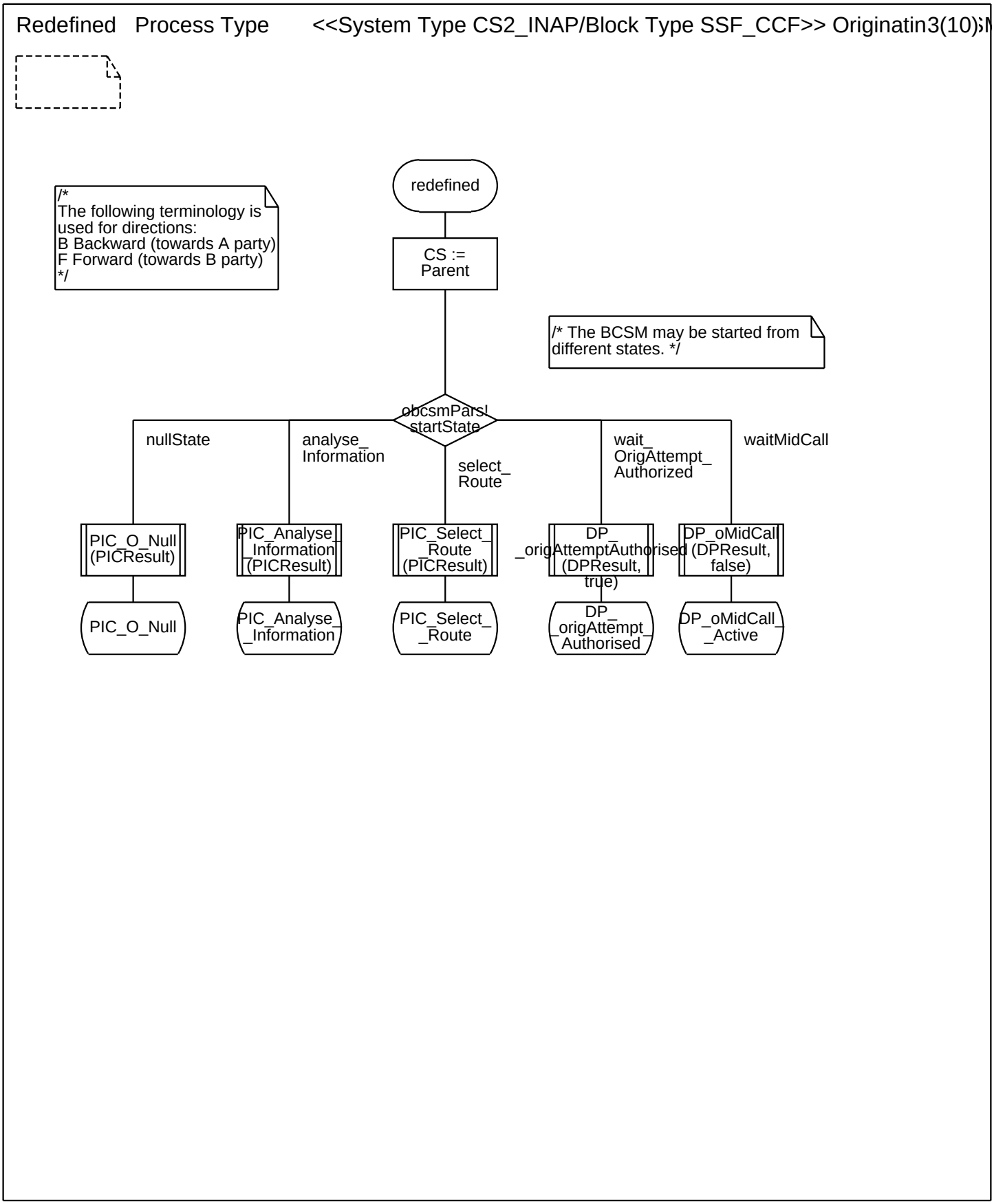
```
/* Note: The T−BCSM for IN CS−2 only shows the extentions required to the
IN CS−1 T−BCSM. Therefore, to understand the complete BCSM behaviour it is
necessary to read the two BCSM descriptions together. */
```

```
TIMER
  SuspendT := 100,
  RetentionO := 300;

DCL
  currentLegID, newLegID LegType,
  nsrArg NetworkSRType,
  udArg UserDataType,
  dpUTSIArg DPUTSIArg;
```

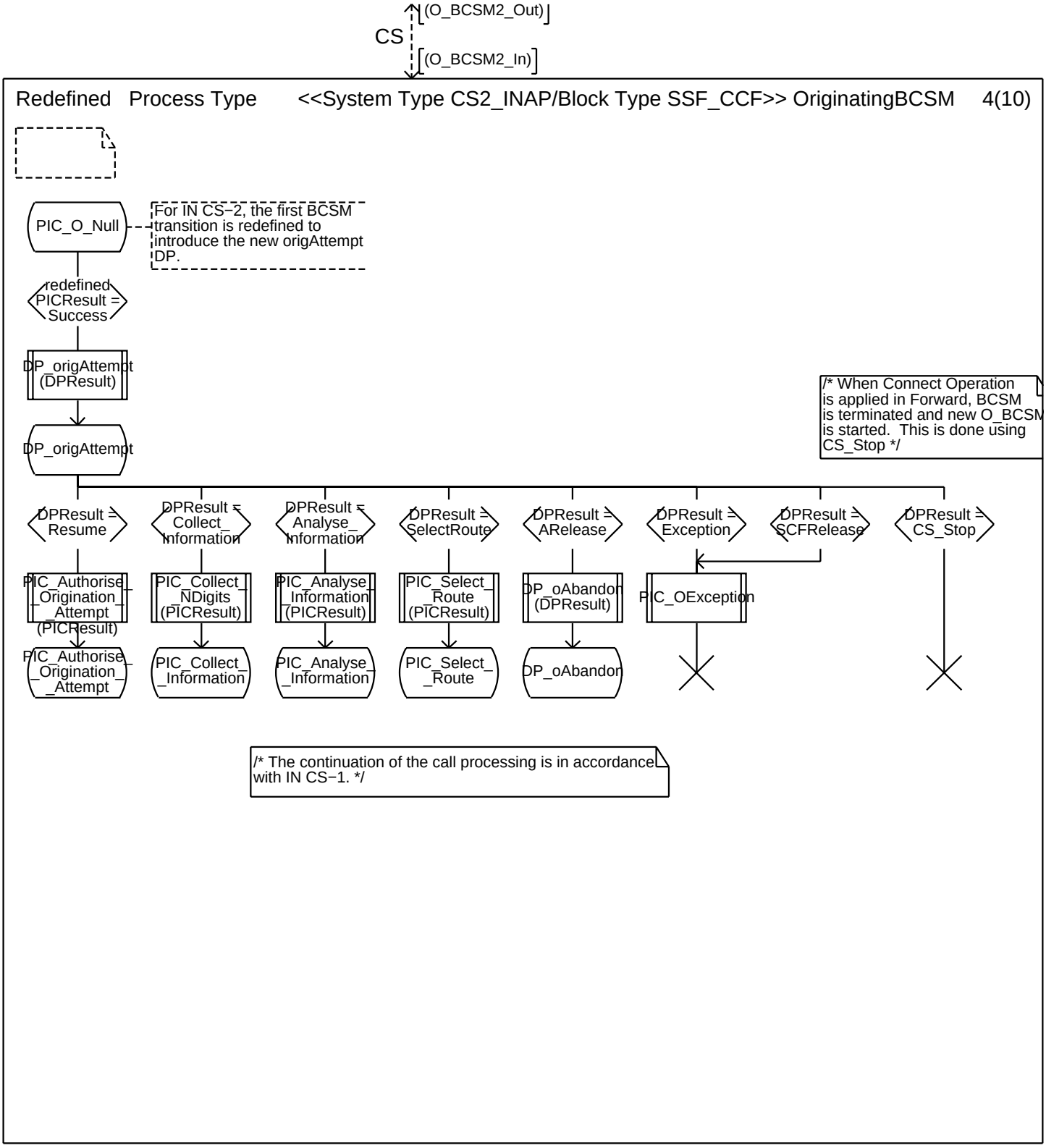
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> Originatin2(10);N
/***** Procedure definitions for the new IN CS−2 PICs and the redefined IN CS−1 PICs. *****/
redefined PIC_O_ _Active
redefined PIC_Send_Call
PIC_O_ _Suspended
PIC_O_ _Retention
redefined PIC_O_ _Alerting
PIC_ _Disconnect
/***** Procedure definitions for the new IN CS−2 DPs and the redefined IN CS−1 DPs. *****/
redefined DP_origAttempt_ Authorised
redefined DP_oAnswer
redefined DP_oAbandon
DP_orig_ Attempt
redefined DP_Collected_ _Info
redefined DP_oNoAnswer
redefined DP_ _oDisconnect
DP_oSuspended
redefined DP_Analysed_ _Information
redefined DP_oCalled_ PartyBusy
DP_oReanswer
redefined DP_Route_ _Select_ _Failure
redefined DP_oMidCall
DP_ _oTermSeized

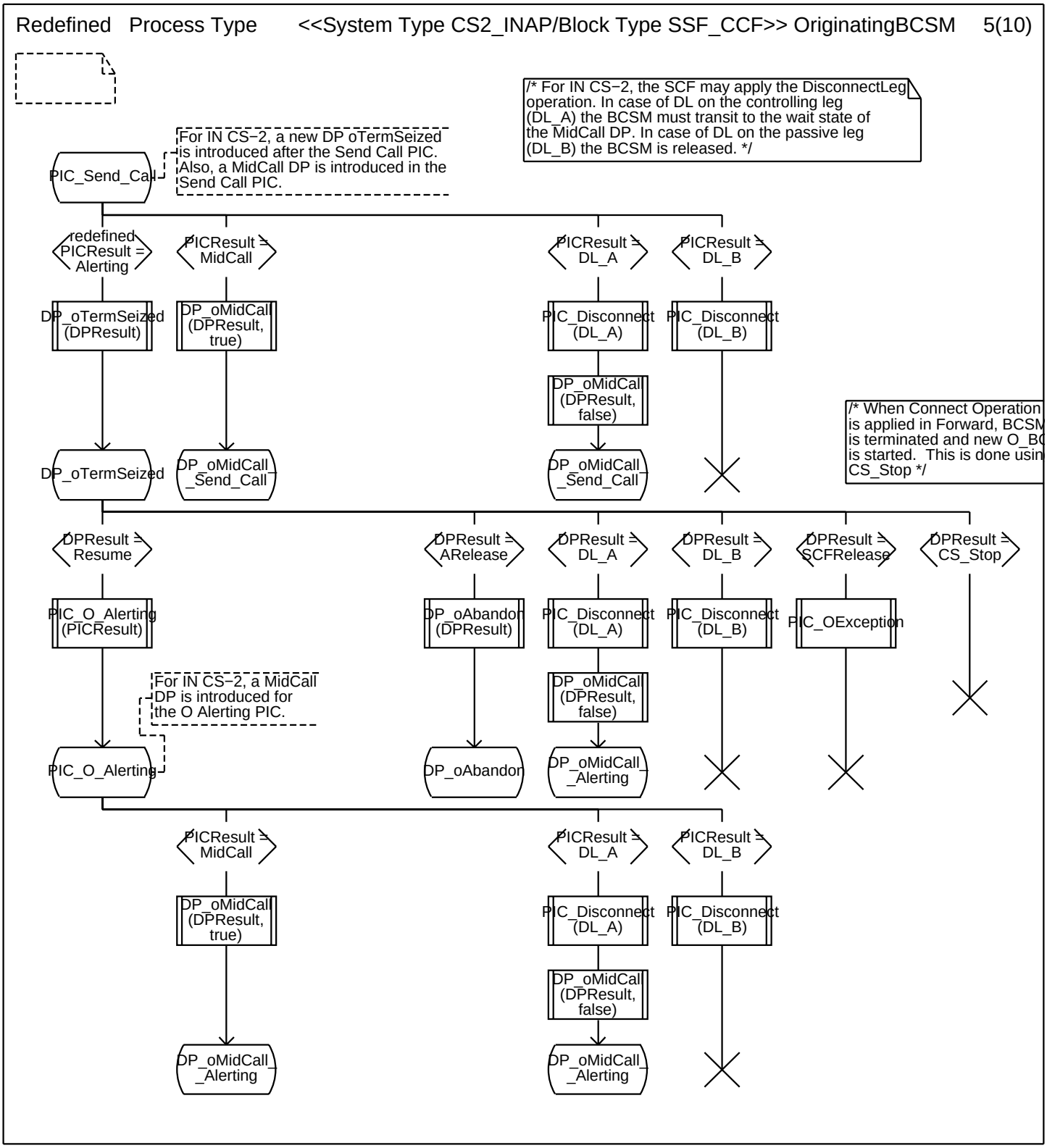
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> Originatin3(10);N
/*
The following terminology is used for directions:
B Backward (towards A party)
F Forward (towards B party)
*/
redefined
CS := Parent
/* The BCSM may be started from different states. */
obcsmParsl startState
nullState
analyse_ Information
select_ Route
wait_ OrigAttempt_ Authorized
waitMidCall
PIC_O_Null (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_ _origAttemptAuthorised (DPResult, true)
DP_oMidCall (DPResult, false)
PIC_O_Null
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_ _origAttempt_ Authorised
DP_oMidCall_ _Active

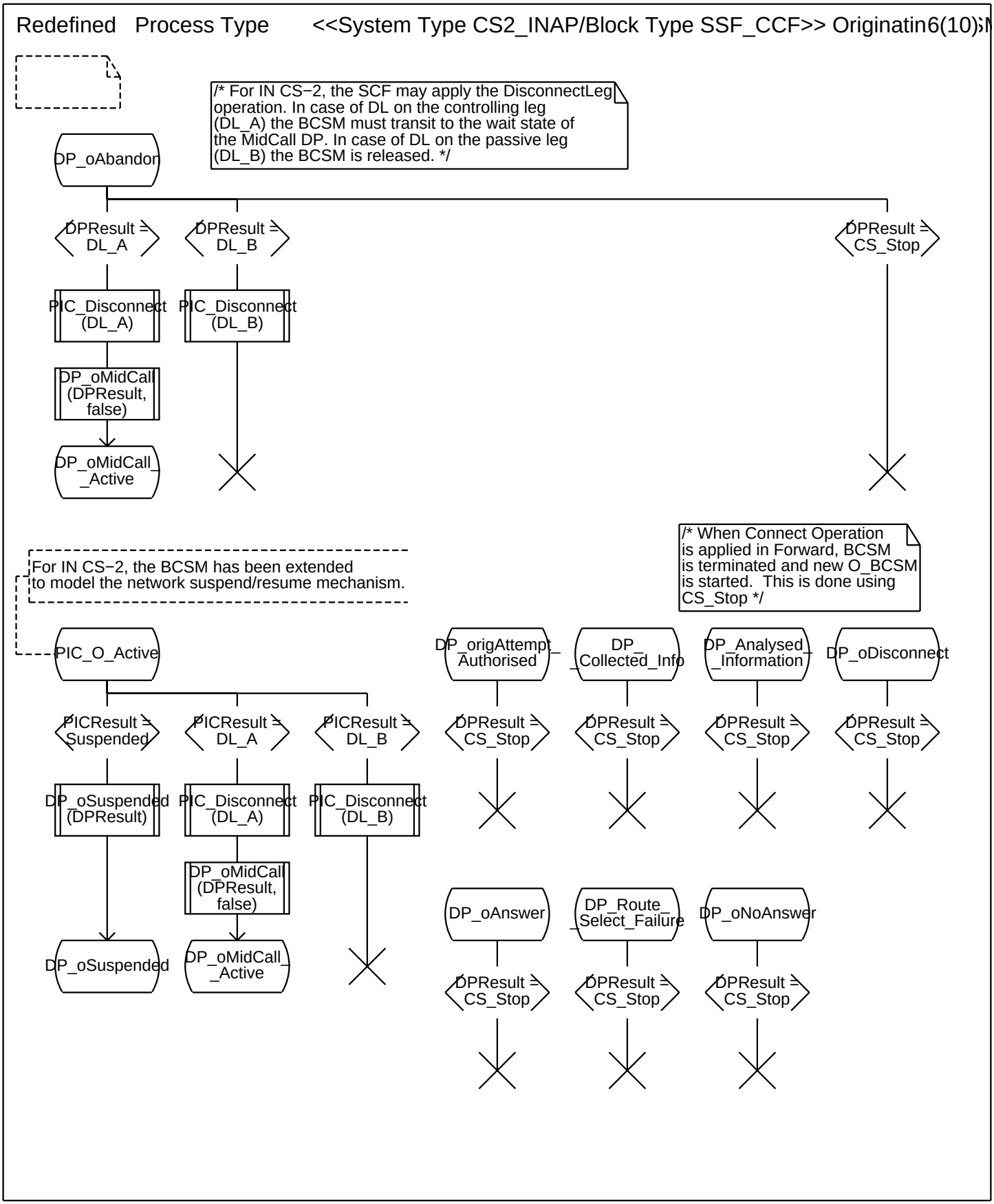
Annex A:


(O_BCSM2_Out)
CS
(O_BCSM2_In)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> OriginatingBCSM 4(10)
PIC_O_Null
For IN CS−2, the first BCSM transition is redefined to introduce the new origAttempt DP.
redefined PICResult = Success
DP_origAttempt (DPResult)
DP_origAttempt
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = SelectRoute
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
DPResult = CS_Stop
PIC_Authorise_ _Origination_ _Attempt (PICResult)
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_OException
PIC_Authorise_ _Origination_ _Attempt
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
/* The continuation of the call processing is in accordance with IN CS−1. */

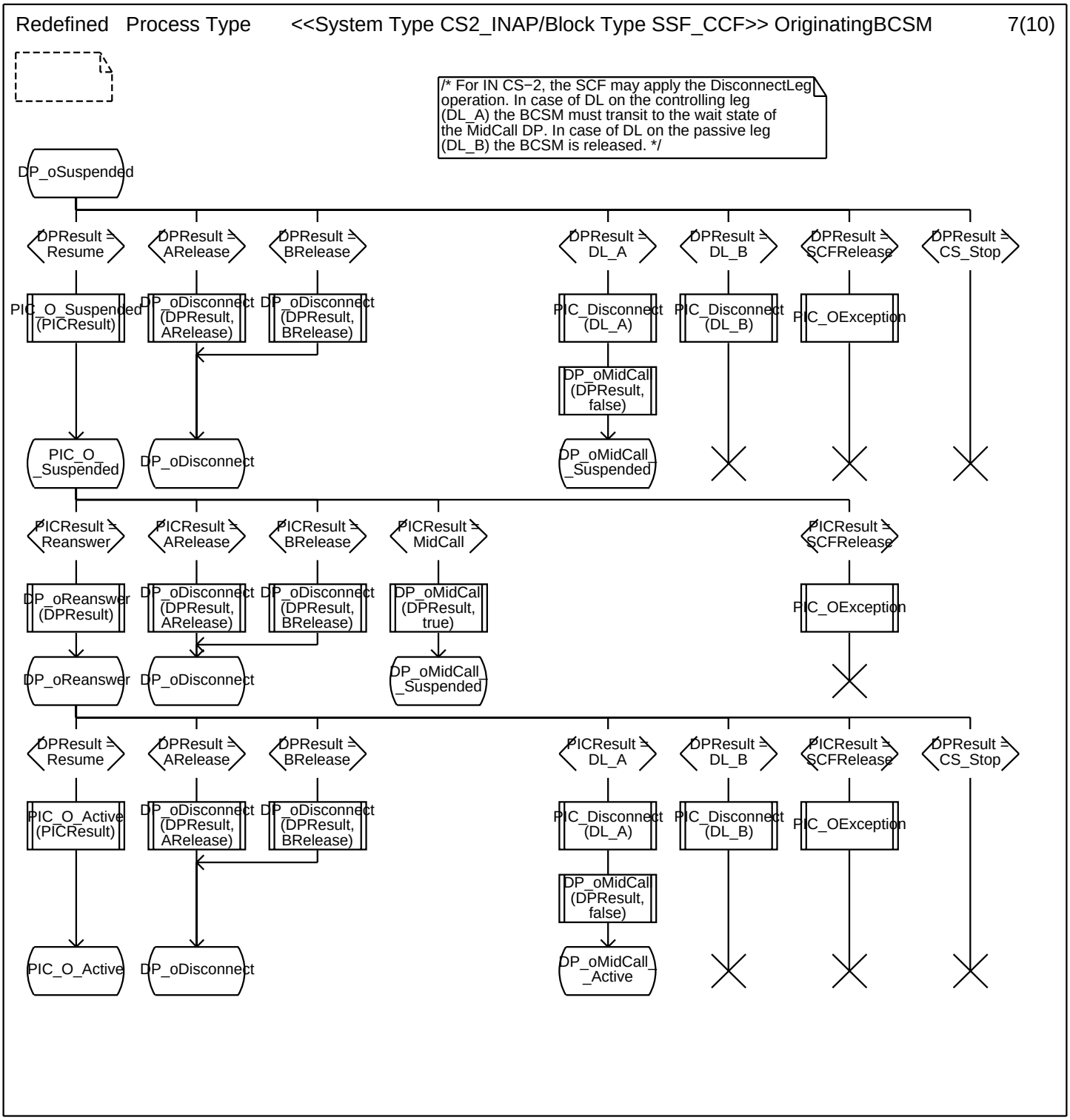
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> OriginatingBCSM 5(10)
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
For IN CS−2, a new DP oTermSeized is introduced after the Send Call PIC. Also, a MidCall DP is introduced in the Send Call PIC.
PIC_Send_Call
redefined PICResult = Alerting
PICResult = MidCall
PICResult = DL_A
PICResult = DL_B
DP_oTermSeized (DPResult)
DP_oMidCall (DPResult, true)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BC is started. This is done usin CS_Stop */
DP_oTermSeized
DP_oMidCall_Send_Call
DP_oMidCall_Send_Call
DPResult = Resume
DPResult = ARelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
DPResult = CS_Stop
PIC_O_Alerting (PICResult)
DP_oAbandon (DPResult)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oMidCall (DPResult, false)
For IN CS−2, a MidCall DP is introduced for the O Alerting PIC.
PIC_O_Alerting
DP_oAbandon
DP_oMidCall_Alerting
PICResult = MidCall
PICResult = DL_A
PICResult = DL_B
DP_oMidCall (DPResult, true)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oMidCall_Alerting
DP_oMidCall_Alerting

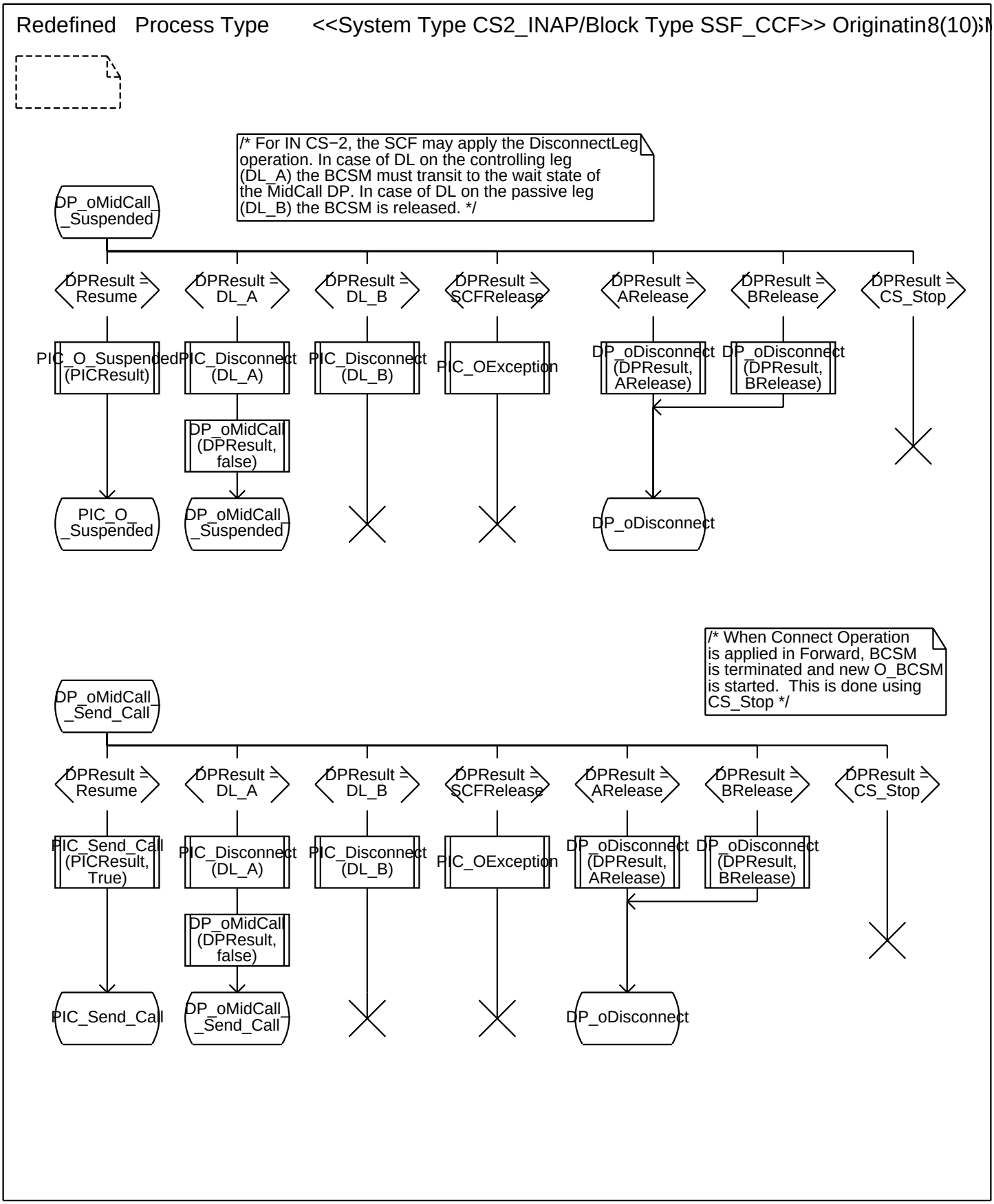
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> Originatin6(10);N
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oAbandon
DPResult = DL_A
DPResult = DL_B
DPResult = CS_Stop
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oMidCall_ _Active
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
For IN CS−2, the BCSM has been extended to model the network suspend/resume mechanism.
PIC_O_Active
DP_origAttempt_ _Authorised
DP_ _Collected_Info
DP_Analysed_ _Information
DP_oDisconnect
PICResult = Suspended
PICResult = DL_A
PICResult = DL_B
DPResult = CS_Stop
DP_oSuspended (DPResult)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oSuspended
DP_oMidCall_ _Active
DP_oAnswer
DP_Route_ _Select_Failure
DP_oNoAnswer

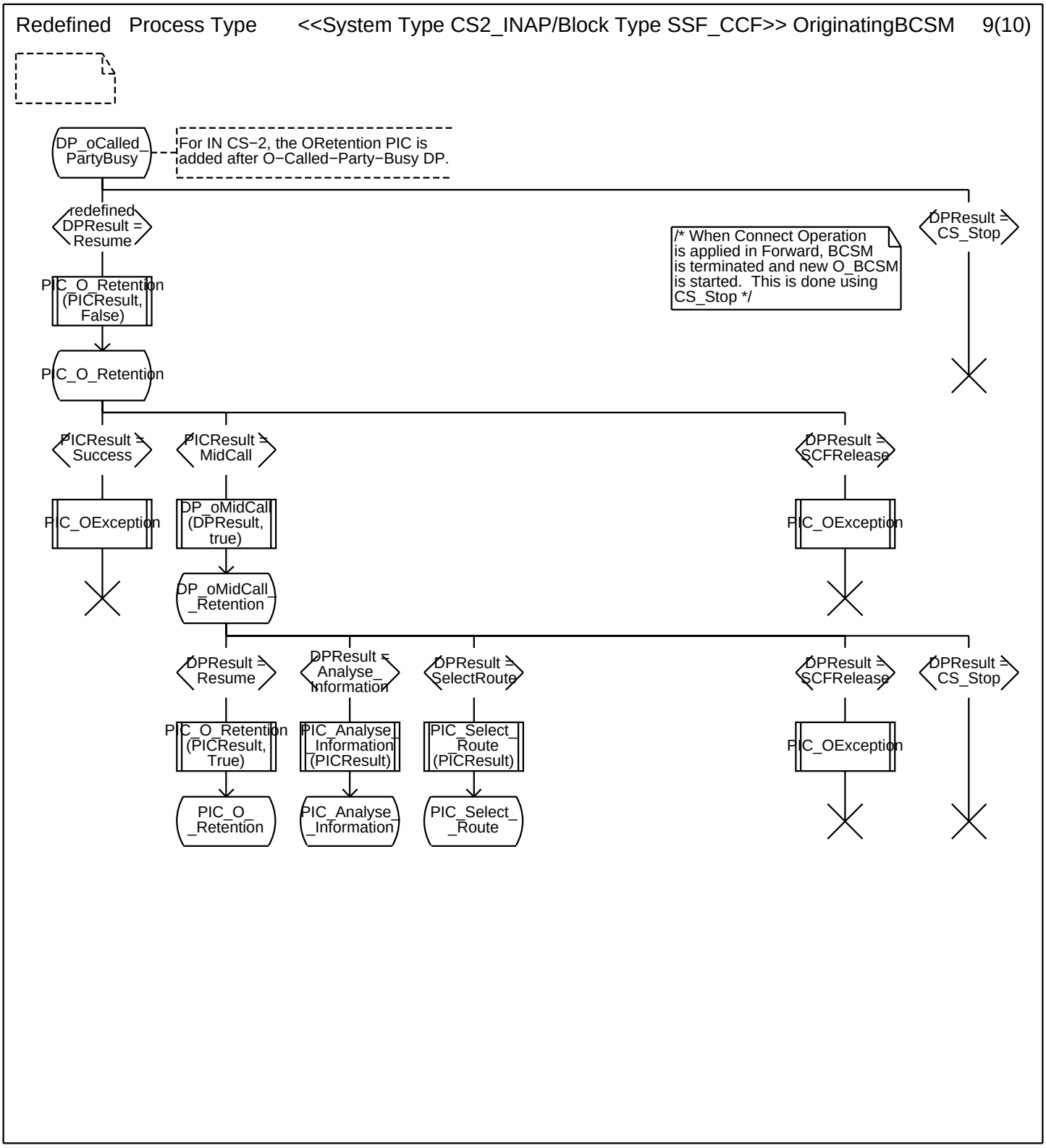
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> OriginatingBCSM 7(10)
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oSuspended
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
DPResult = CS_Stop
PIC_O_Suspended (PICResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oMidCall (DPResult, false)
PIC_O_ _Suspended
DP_oDisconnect
DP_oMidCall_ _Suspended
PICResult = Reanswer
PICResult = ARelease
PICResult = BRelease
PICResult = MidCall
PICResult = SCFRelease
DP_oReanswer (DPResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
DP_oMidCal (DPResult, true)
PIC_OException
DP_oReanswer
DP_oDisconnect
DP_oMidCall_ _Suspended
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
PICResult = DL_A
DPResult = DL_B
PICResult = SCFRelease
DPResult = CS_Stop
PIC_O_Active (PICResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oMidCall (DPResult, false)
PIC_O_Active
DP_oDisconnect
DP_oMidCall_ _Active

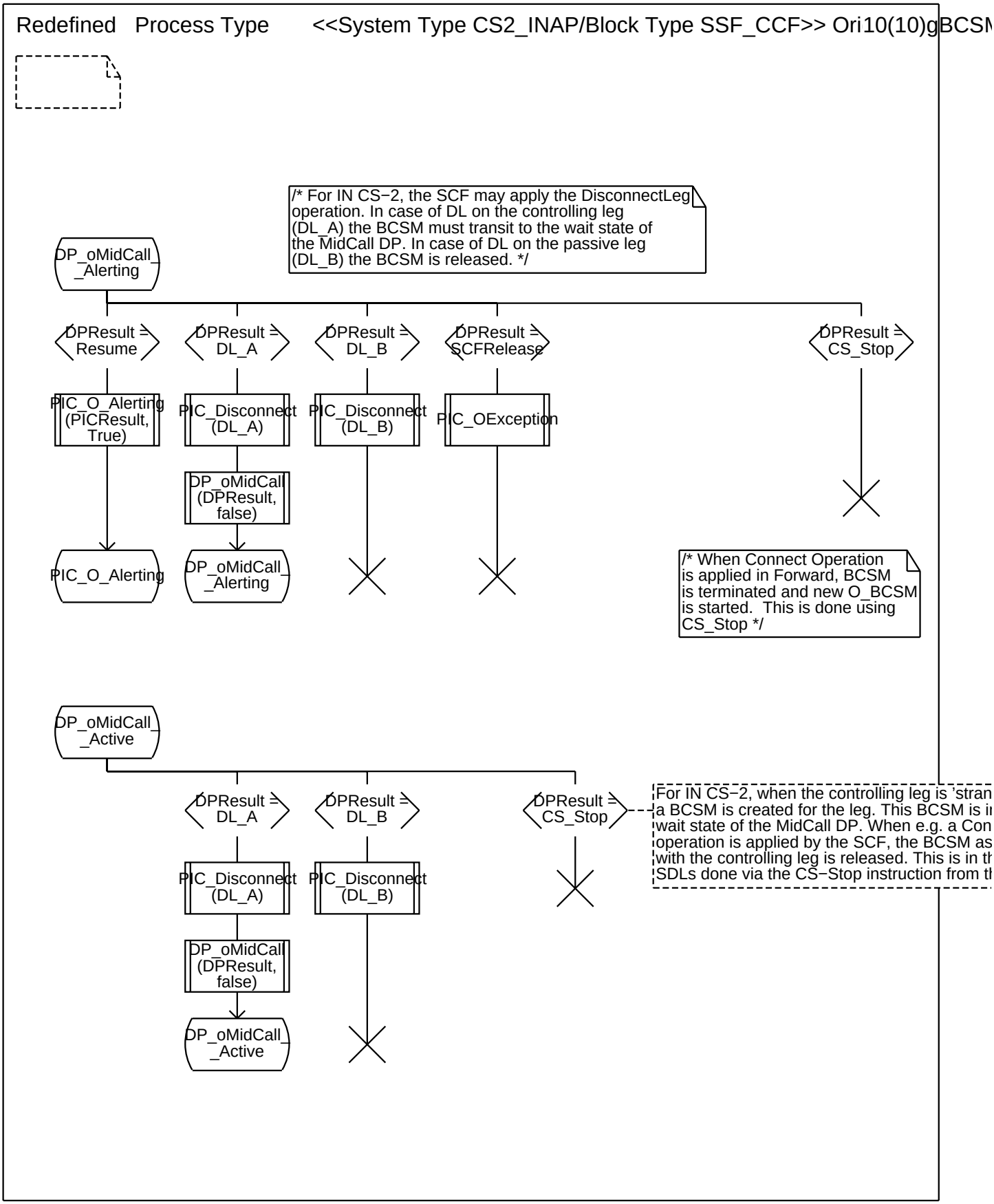
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> Originatin8(10);N
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oMidCall_ _Suspended
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
DPResult = ARelease
DPResult = BRelease
DPResult = CS_Stop
PIC_O_Suspended (PICResult)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
DP_oMidCall (DPResult, false)
PIC_O_ _Suspended
DP_oMidCall_ _Suspended
DP_oDisconnect
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
DP_oMidCall_ _Send_Call
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
DPResult = ARelease
DPResult = BRelease
DPResult = CS_Stop
PIC_Send_Call (PICResult, True)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
DP_oMidCall (DPResult, false)
PIC_Send_Call
DP_oMidCall_ _Send_Call
DP_oDisconnect

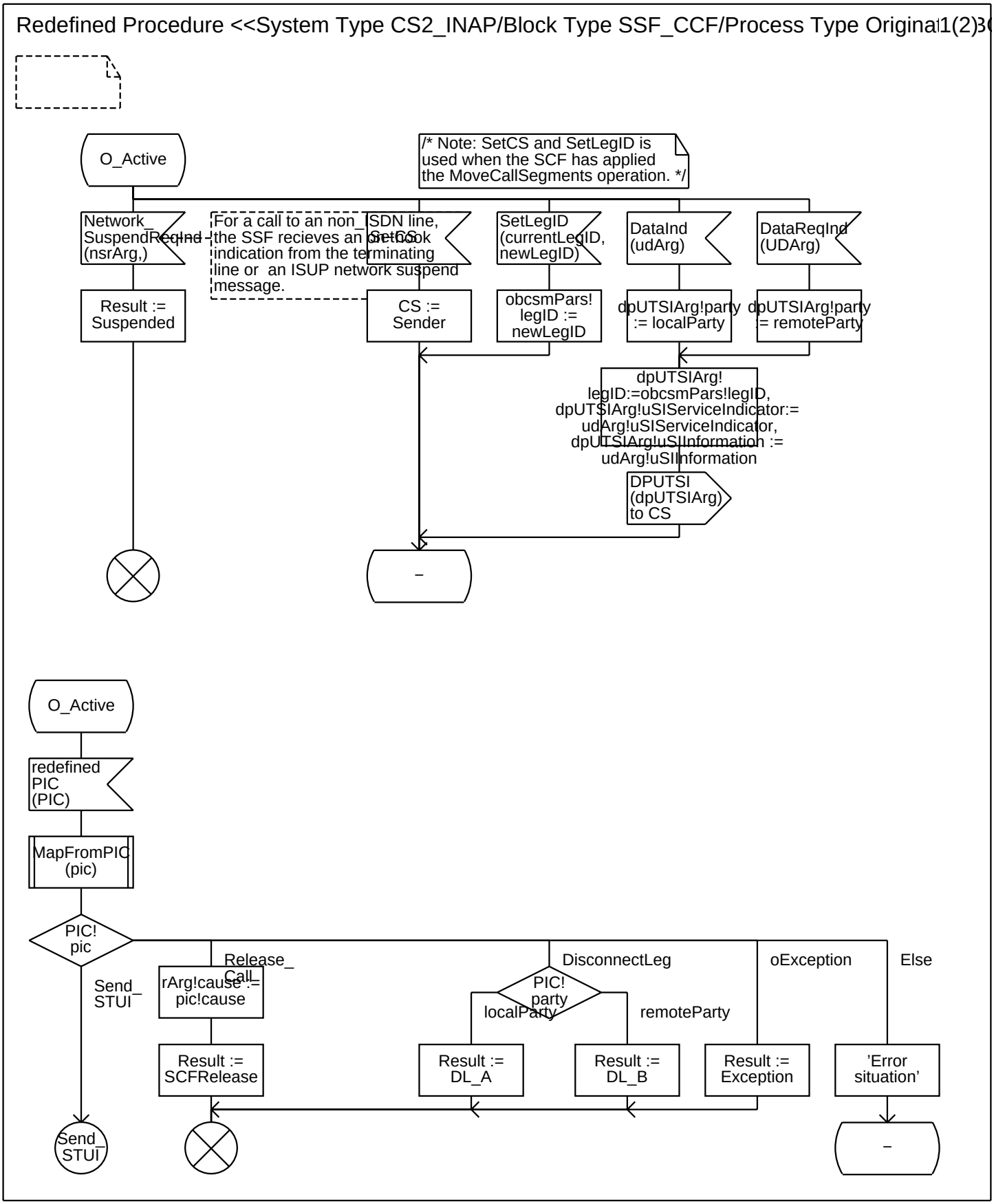
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> OriginatingBCSM 9(10)
DP_oCalled_PartyBusy
For IN CS−2, the ORetention PIC is added after O−Called−Party−Busy DP.
redefined DPResult = Resume
DPResult = CS_Stop
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
PIC_O_Retention (PICResult, False)
PIC_O_Retention
PICResult = Success
PICResult = MidCall
DPResult = SCFRelease
PIC_OException
DP_oMidCall (DPResult, true)
PIC_OException
DP_oMidCall_Retention
DPResult = Resume
DPResult = Analyse_Information
DPResult = SelectRoute
DPResult = SCFRelease
DPResult = CS_Stop
PIC_O_Retention (PICResult, True)
PIC_Analyse_Information (PICResult)
PIC_Select_Route (PICResult)
PIC_OException
PIC_O_Retention
PIC_Analyse_Information
PIC_Select_Route

Annex A:

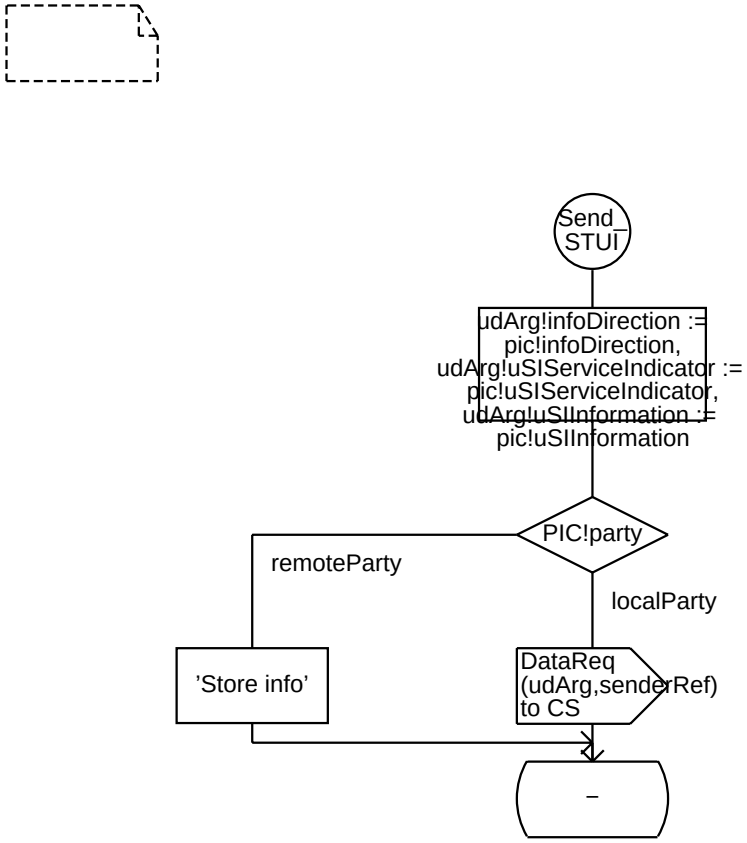

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> Ori10(10)gBCSM
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oMidCall_ _Alerting
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
DPResult = CS_Stop
PIC_O_Alerting (PICResult, True)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oMidCall (DPResult, false)
PIC_O_Alerting
DP_oMidCall_ _Alerting
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
DP_oMidCall_ _Active
DPResult = DL_A
DPResult = DL_B
DPResult = CS_Stop
For IN CS−2, when the controlling leg is 'stran
a BCSM is created for the leg. This BCSM is i
wait state of the MidCall DP. When e.g. a Con
operation is applied by the SCF, the BCSM as
with the controlling leg is released. This is in t
SDLs done via the CS−Stop instruction from t
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oMidCall_ _Active

Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
O_Active
/* Note: SetCS and SetLegID is used when the SCF has applied the MoveCallSegments operation. */
Network_ SuspendReqInd (nsrArg,)
For a call to an non_ISDN line, the SSF recieves an off-hook indication from the terminating line or an ISUP network suspend message.
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
DataReqInd (UDArg)
Result := Suspended
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
dpUTSIArg!party := remoteParty
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–
O_Active
redefined PIC (PIC)
MapFromPIC (pic)
PIC! pic
Send_ STUI
Release_ Call
rArg!cause := pic!cause
DisconnectLeg
PIC! party
localParty
remoteParty
oException
Else
Result := SCFRelease
Result := DL_A
Result := DL_B
Result := Exception
'Error situation'
Send_ STUI
–

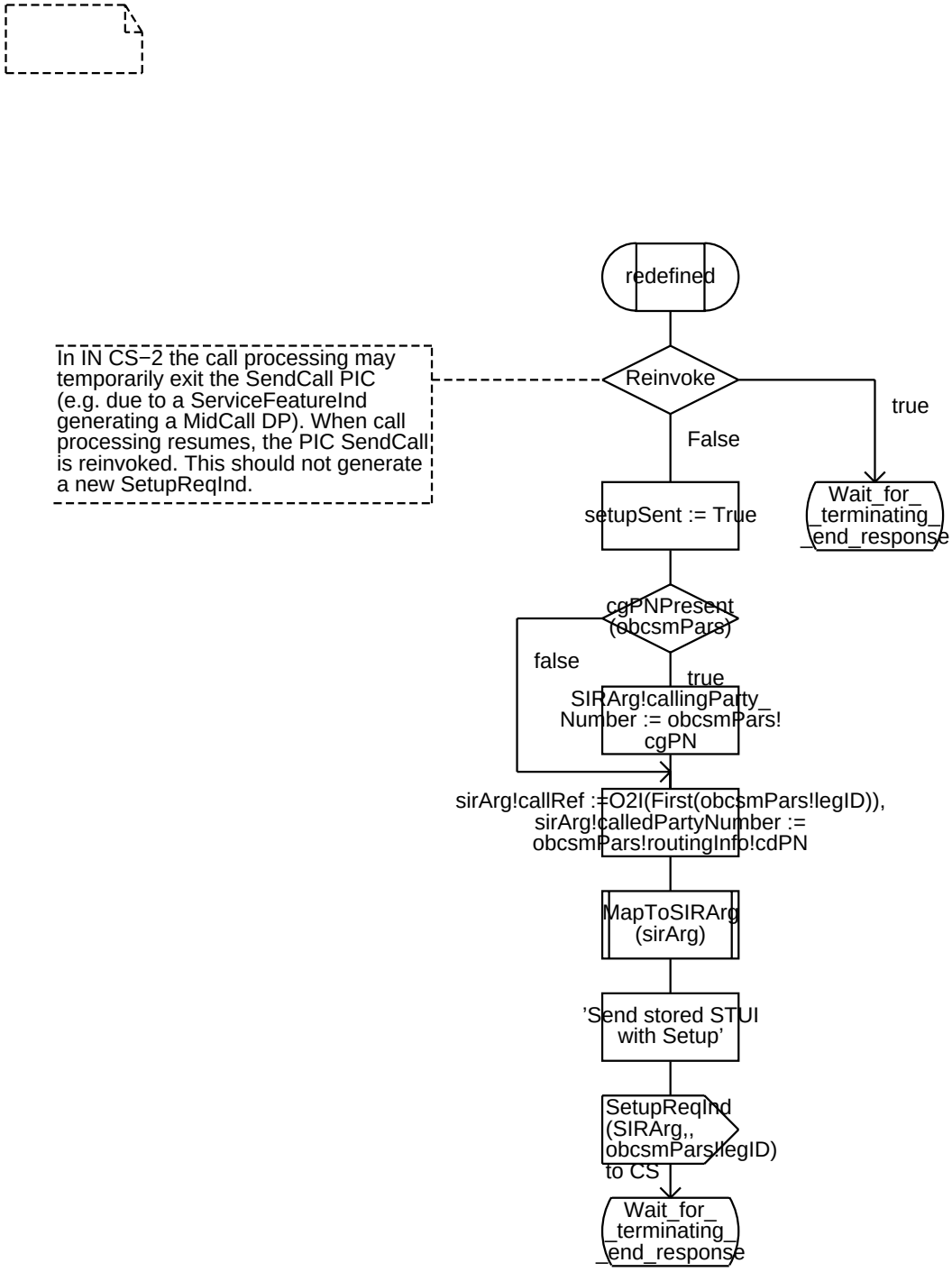
Annex A:

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original2(2)3

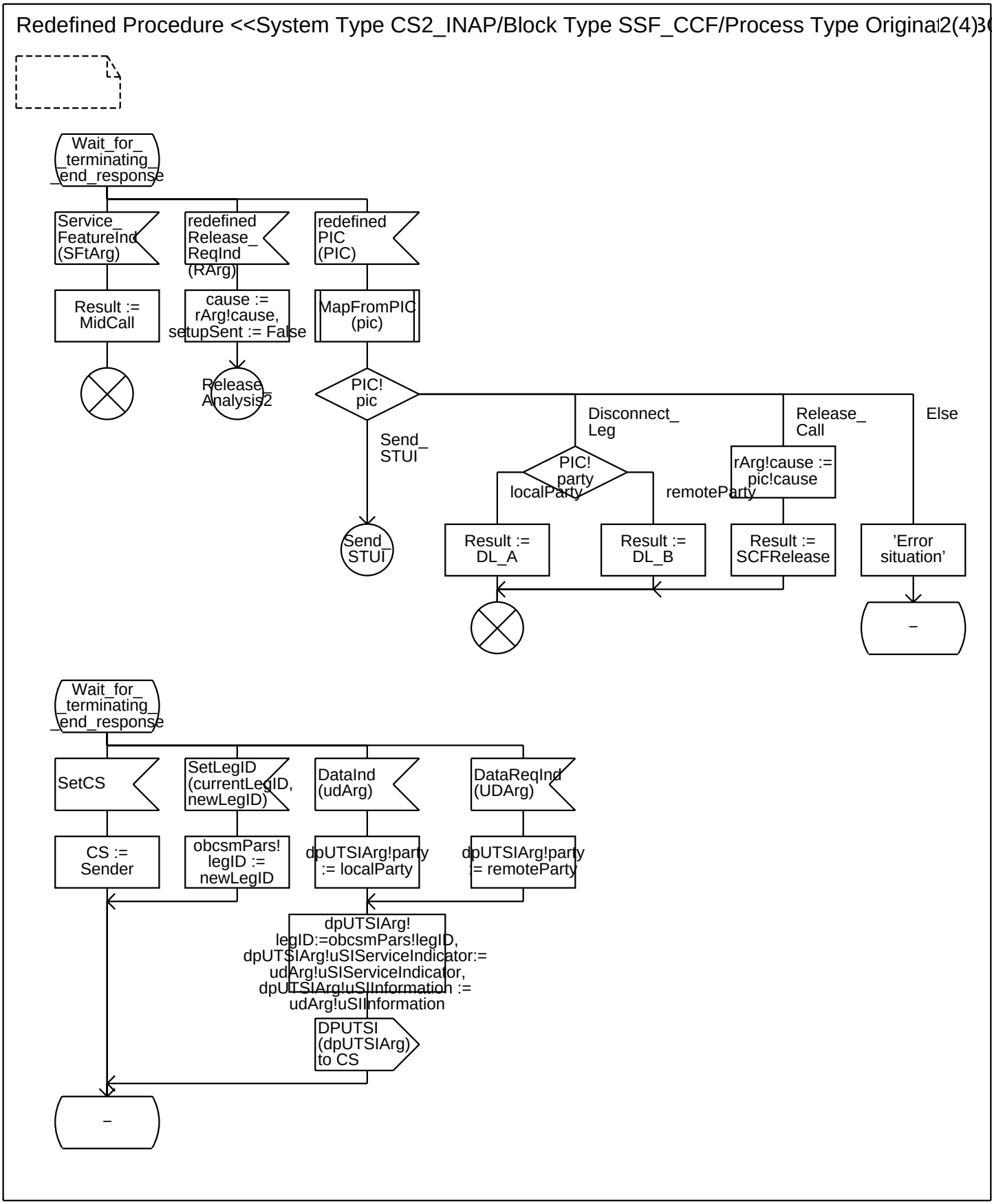


Annex A:

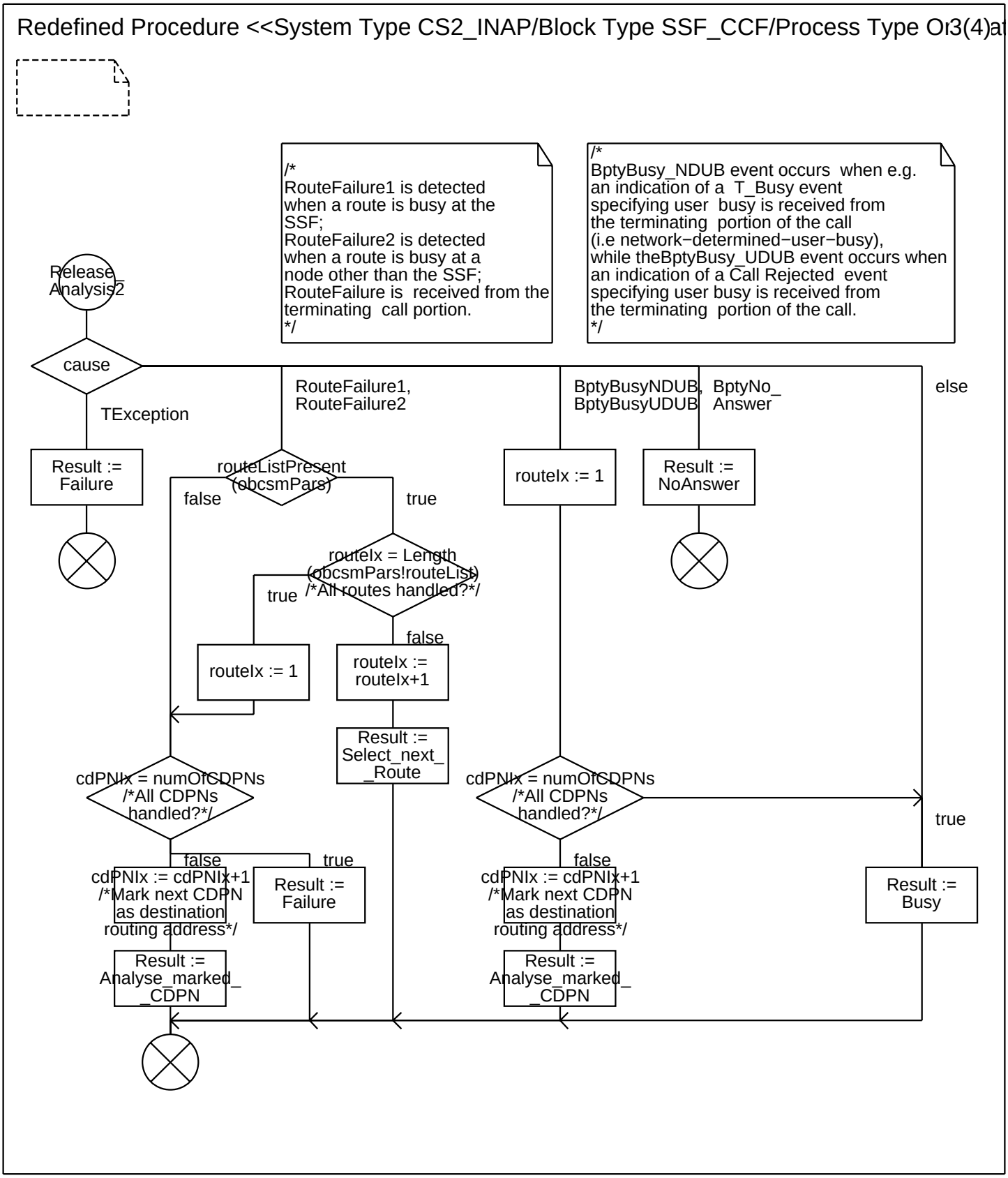
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(4)3



Annex A:

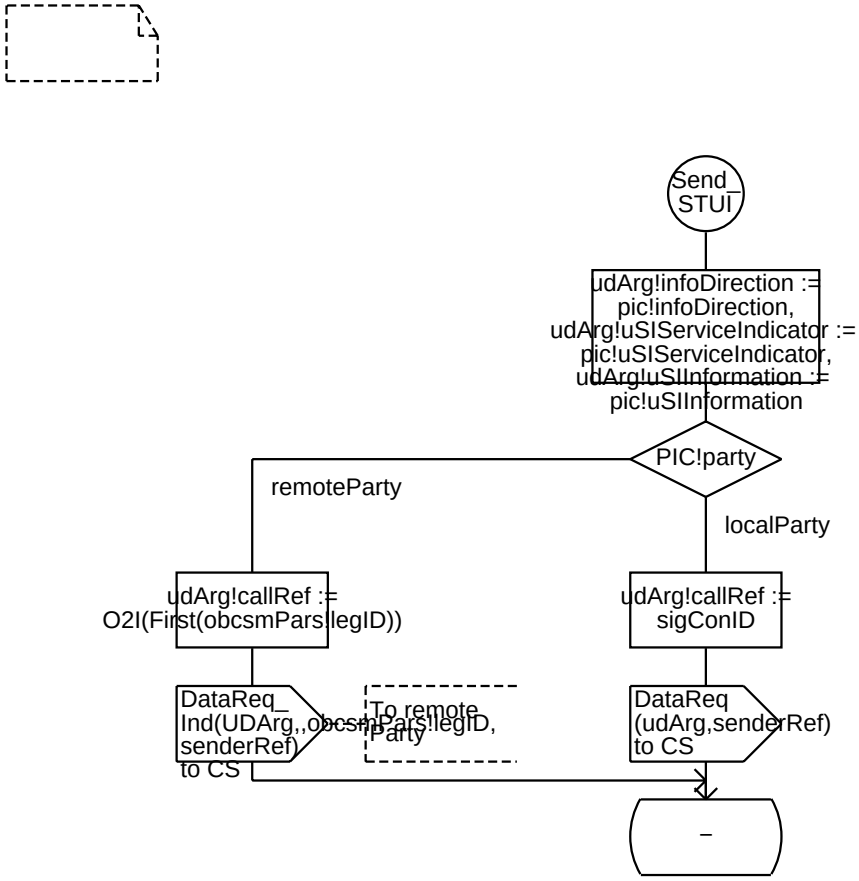

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(4)3
Wait_for_ _terminating_ _end_response
Service_ FeatureInd (SFtArg)
redefined Release_ ReqInd (RArg)
redefined PIC (PIC)
Result := MidCall
cause := rArg!cause, setupSent := False
MapFromPIC (pic)
Release_ Analysis2
PIC! pic
Send_ STUI
Disconnect_ Leg
Release_ Call
Else
PIC! party
localParty
remoteParty
rArg!cause := pic!cause
Send_ STUI
Result := DL_A
Result := DL_B
Result := SCFRelease
'Error situation'
–
Wait_for_ _terminating_ _end_response
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
DataReqInd (UDArg)
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
dpUTSIArg!party := remoteParty
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–

Annex A:

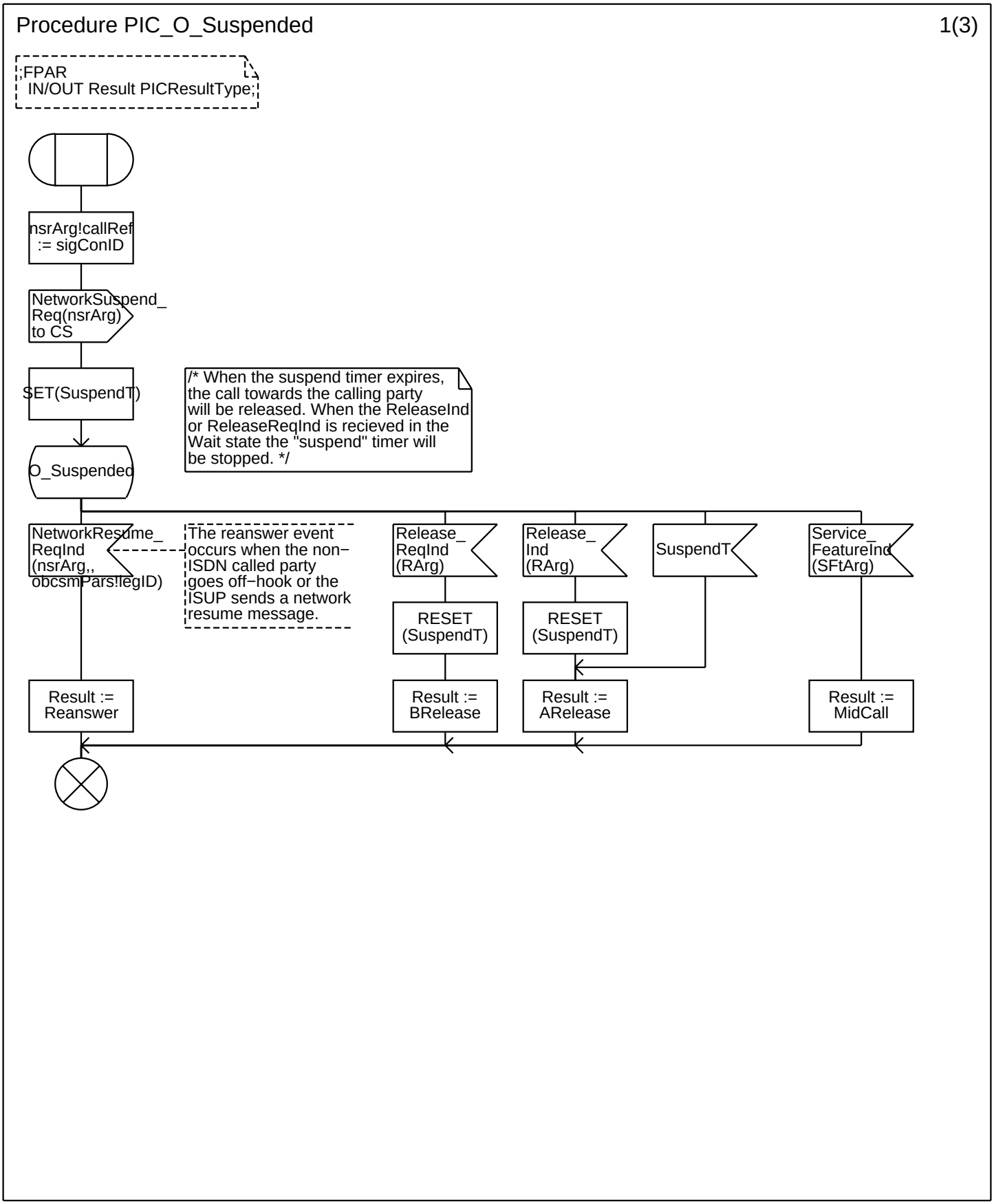

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Or3(4)at
/*
RouteFailure1 is detected
when a route is busy at the
SSF;
RouteFailure2 is detected
when a route is busy at a
node other than the SSF;
RouteFailure is received from the
terminating call portion.
*/
/*
BptyBusy_NDUB 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_
Analysis2
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
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Failure
false
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Busy
Result :=
Analyse_marked_
_CDPN
Result :=
Analyse_marked_
_CDPN

Annex A:

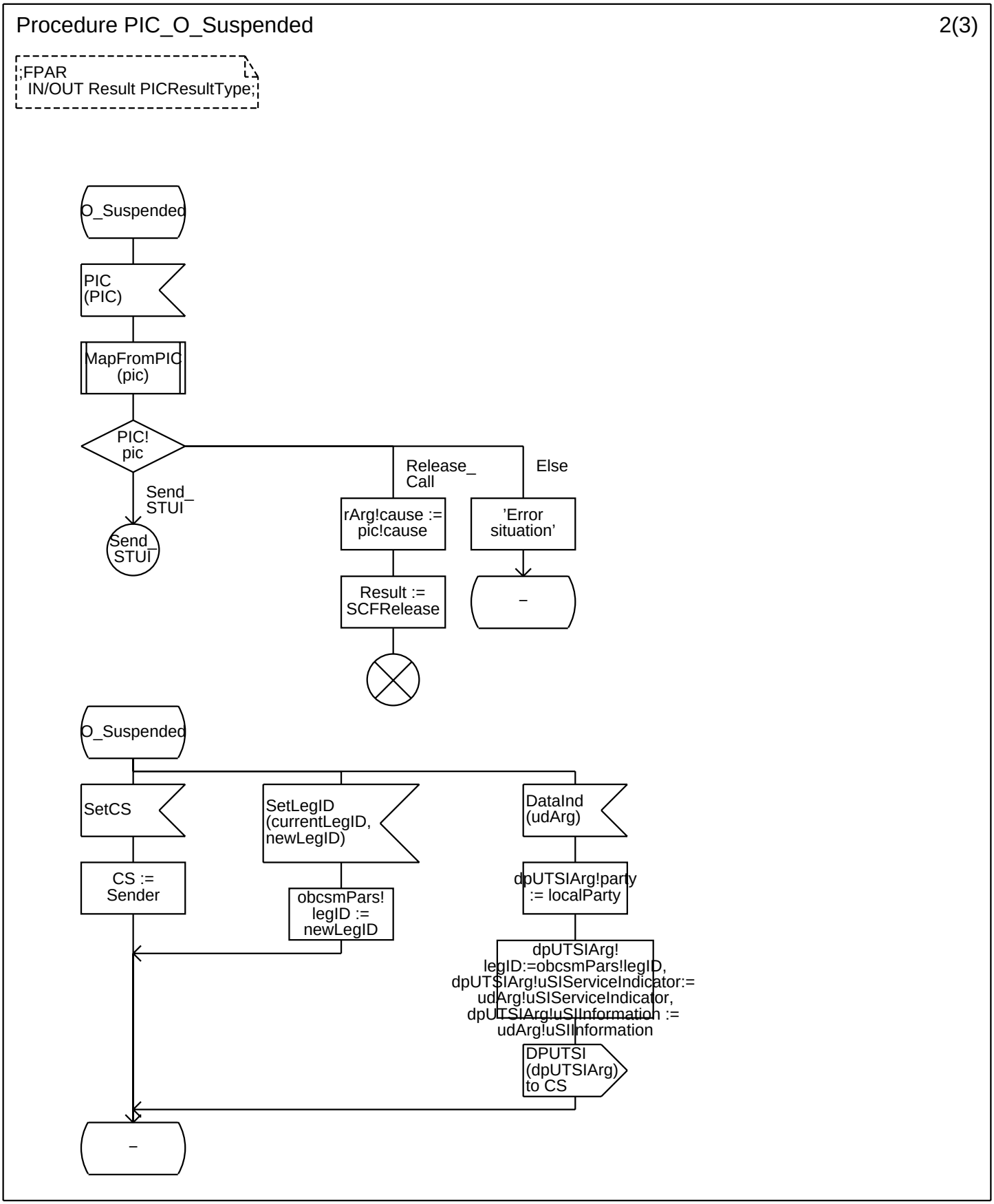
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original4(4)3



Annex A:

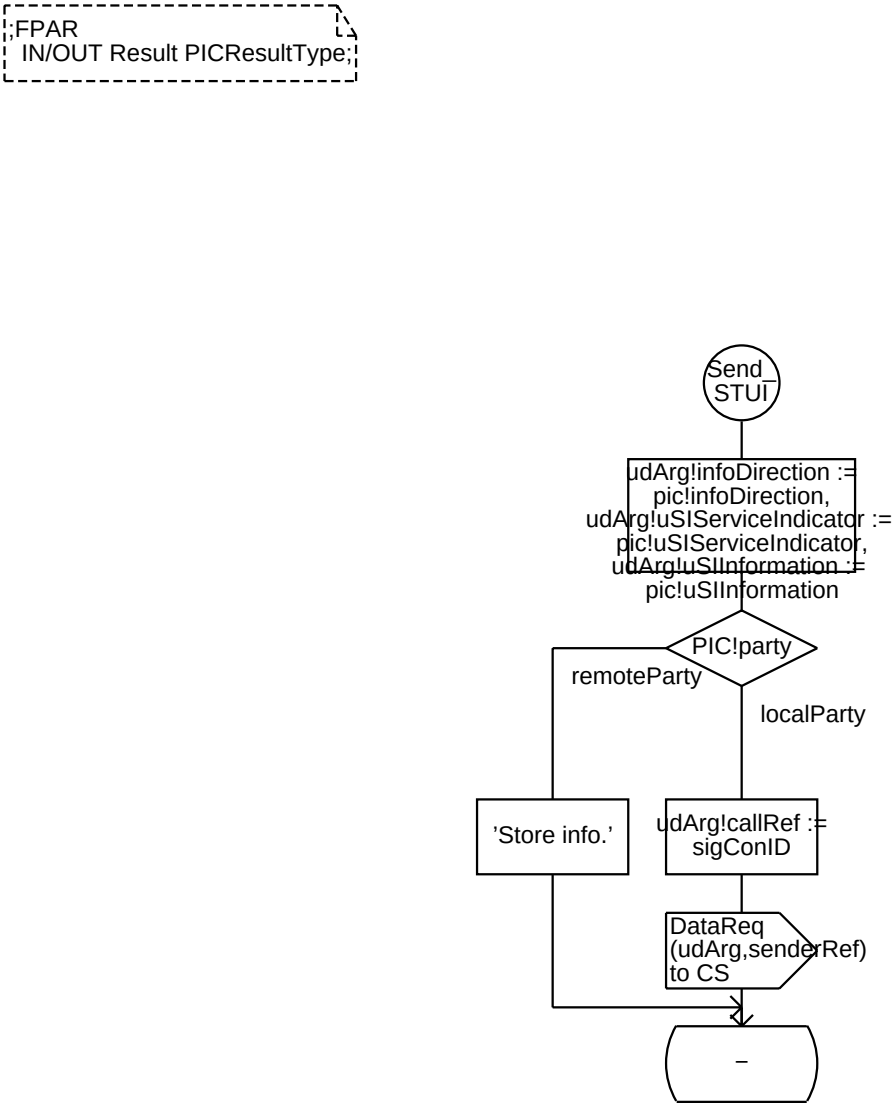

Procedure PIC_O_Suspended
1(3)
;FPAR
IN/OUT Result PICResultType;
nsrArg!callRef
:= sigConID
NetworkSuspend_
Req(nsrArg)
to CS
SET(SuspendT)
/* When the suspend timer expires, the call towards the calling party will be released. When the ReleaseInd or ReleaseReqInd is recieved in the Wait state the "suspend" timer will be stopped. */
O_Suspended
NetworkResume_
ReqInd
(nsrArg,,
obcsmPars!legID)
The reanswer event occurs when the non−ISDN called party goes off−hook or the ISUP sends a network resume message.
Release_
ReqInd
(RArg)
Release_
Ind
(RArg)
SuspendT
Service_
FeatureInd
(SFtArg)
RESET
(SuspendT)
RESET
(SuspendT)
Result :=
Reanswer
Result :=
BRelease
Result :=
ARelease
Result :=
MidCall

Annex A:


Procedure PIC_O_Suspended
2(3)
;FPAR
IN/OUT Result PICResultType;
O_Suspended
PIC
(PIC)
MapFromPIC
(pic)
PIC!
pic
Release_
Call
Else
Send_
STUI
rArg!cause :=
pic!cause
'Error
situation'
Send_
STUI
Result :=
SCFRelease
–
O_Suspended
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

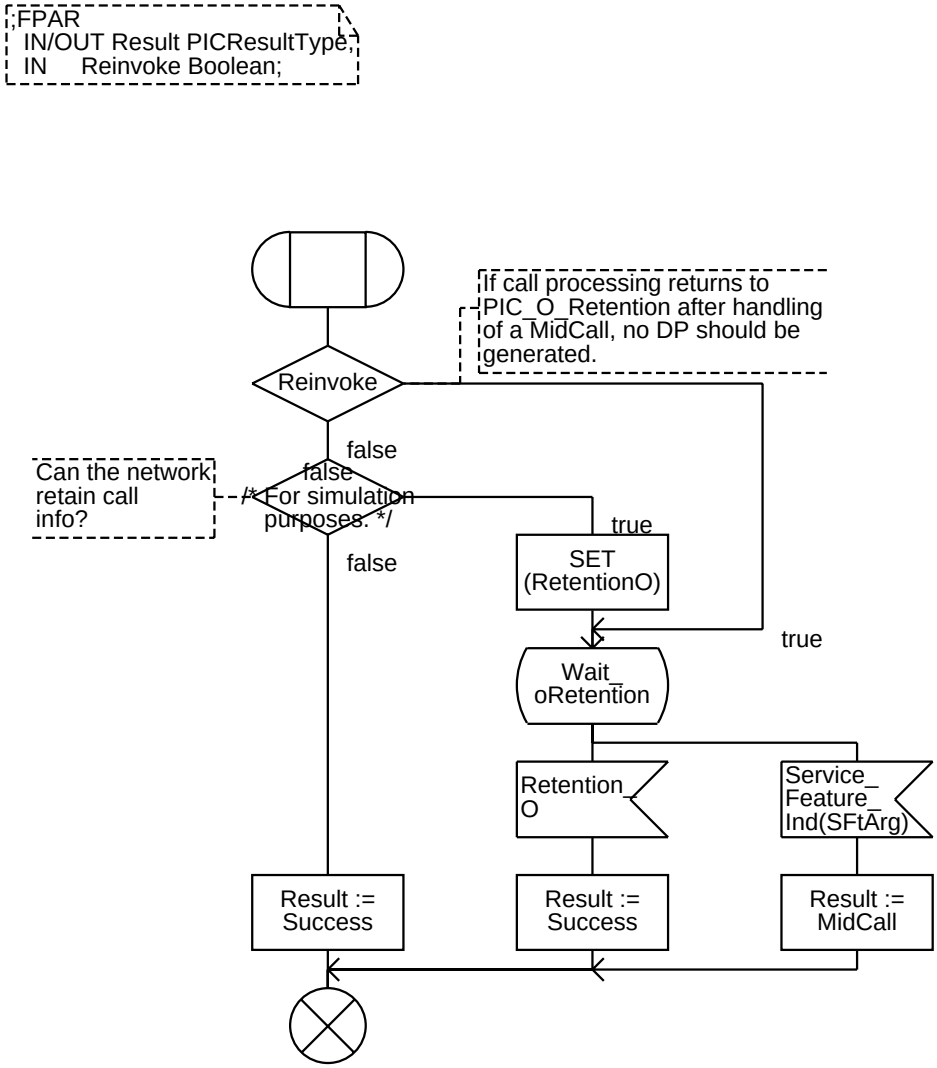
Annex A:

Procedure PIC_O_Suspended 3(3)

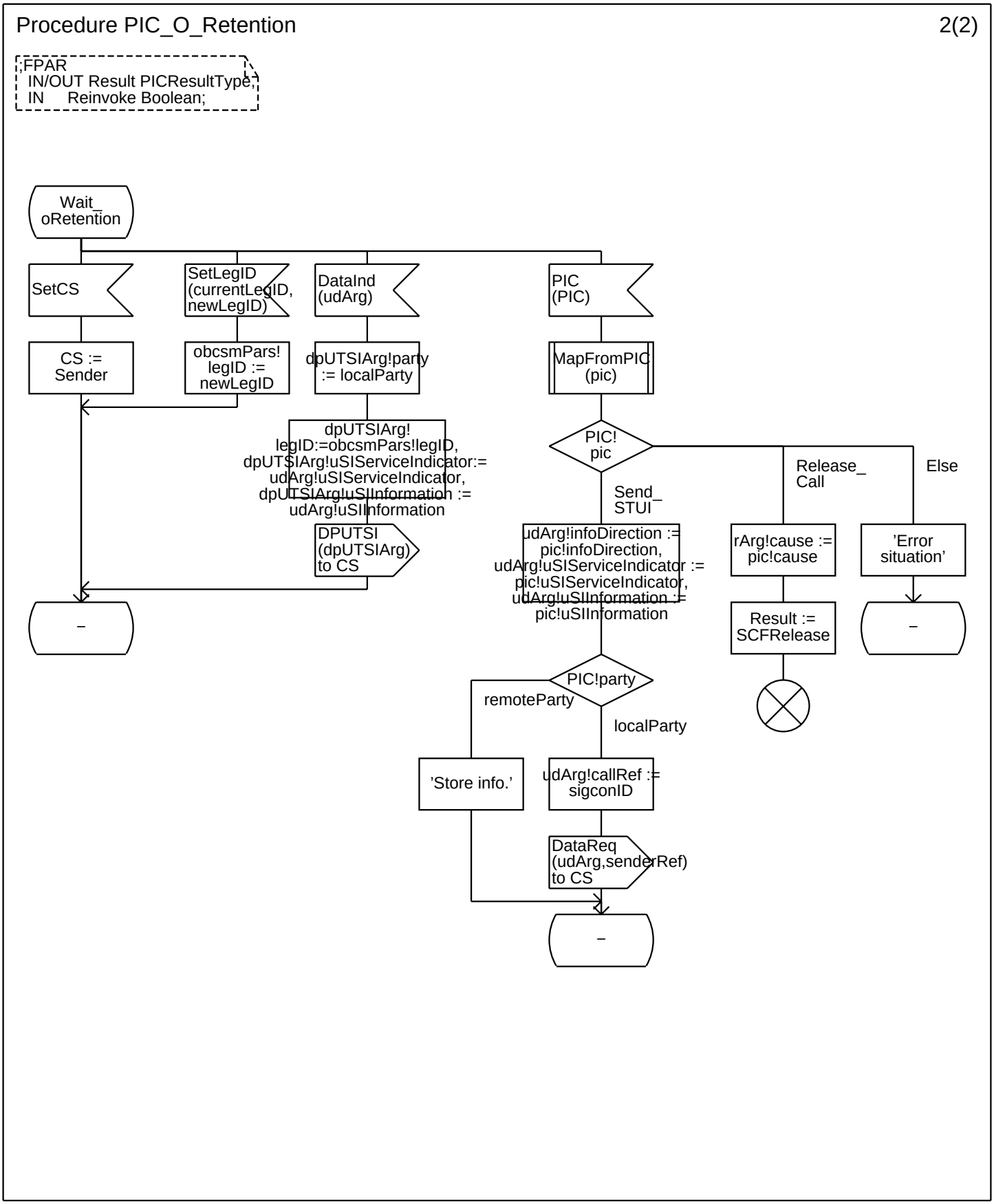


Annex A:

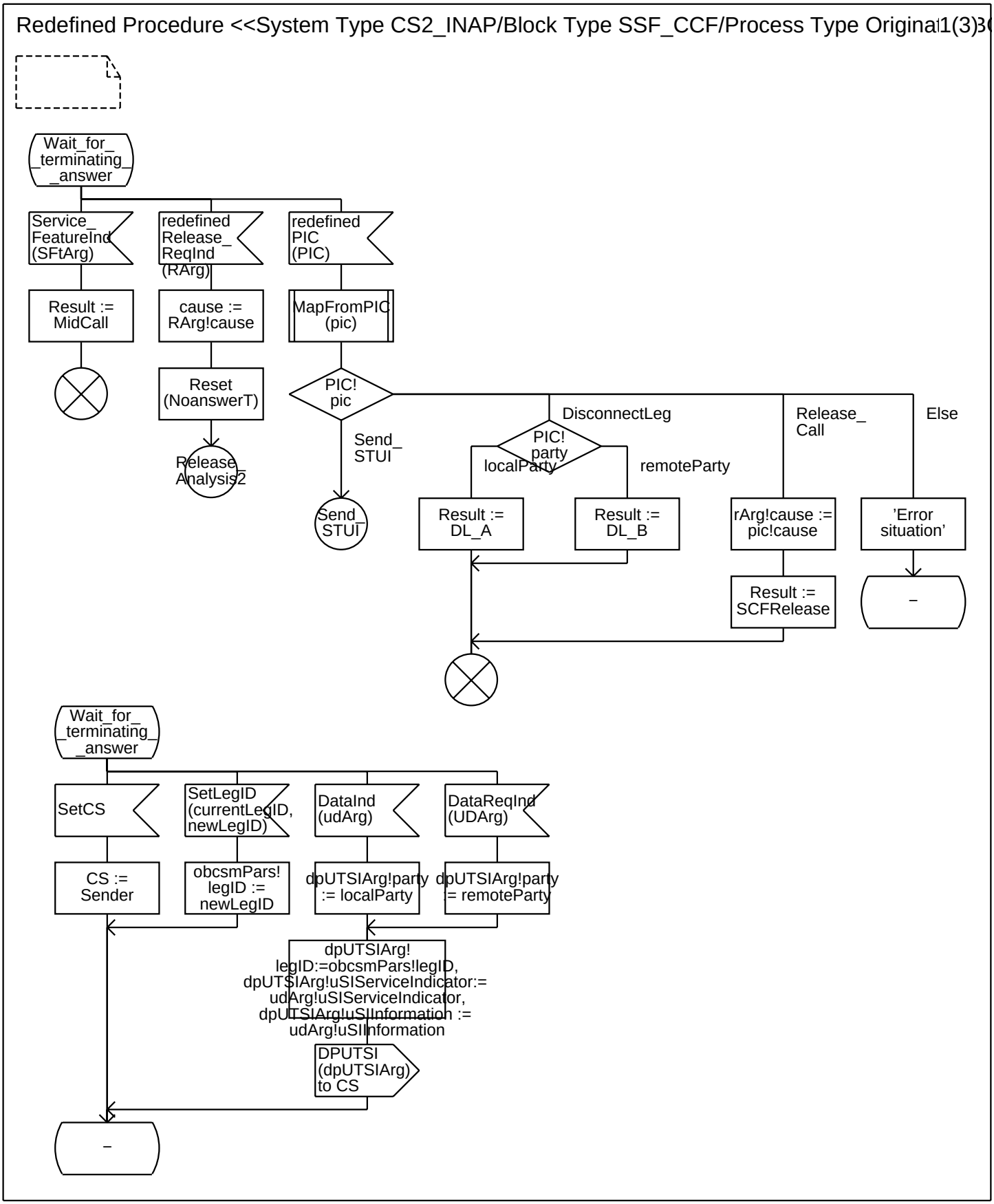
Procedure PIC_O_Retention 1(2)



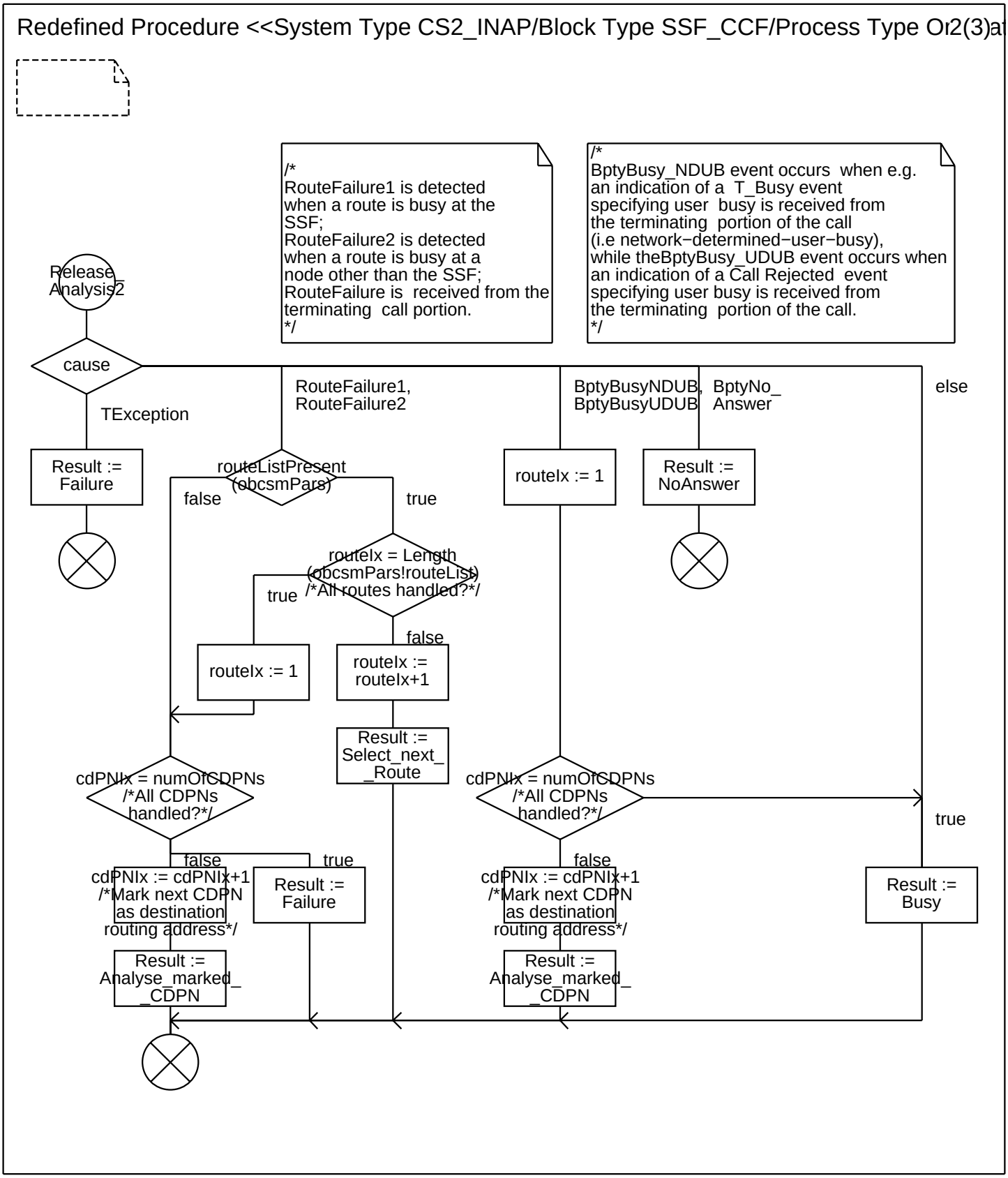
Annex A:


Procedure PIC_O_Retention
2(2)
;FPAR
IN/OUT Result PICResultType;
IN Reinvoke Boolean;
Wait_
oRetention
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
PIC
(PIC)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
MapFromPIC
(pic)
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
PIC!
pic
Release_
Call
Else
Send_
STUI
DPUTSI
(dpUTSIArg)
to CS
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
rArg!cause :=
pic!cause
'Error
situation'
–
Result :=
SCFRelease
–
PIC!party
remoteParty
localParty
'Store info.'
udArg!callRef :=
sigconID
DataReq
(udArg,senderRef)
to CS
–

Annex A:

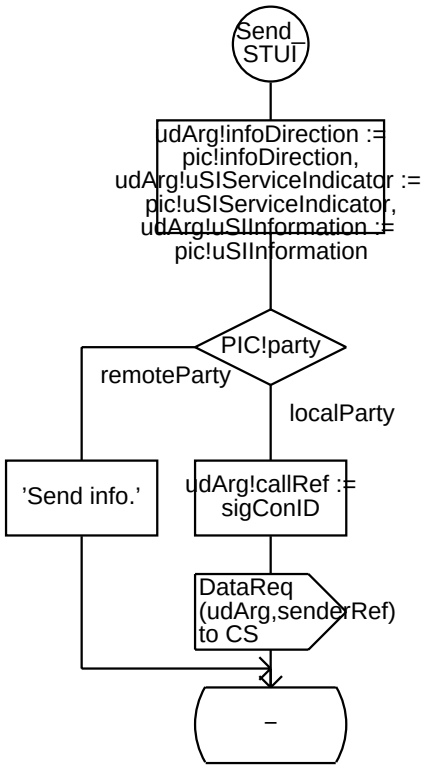

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(3)
Wait_for_ _terminating_ _answer
Service_ FeatureInd (SFtArg)
redefined Release_ ReqInd (RArg)
redefined PIC (PIC)
Result := MidCall
cause := RArg!cause
MapFromPIC (pic)
Reset (NoanswerT)
PIC! pic
Release_ Analysis2
Send_ STUI
Send_ STUI
DisconnectLeg
Release_ Call
Else
PIC! party
localParty
remoteParty
Result := DL_A
Result := DL_B
rArg!cause := pic!cause
'Error situation'
Result := SCFRelease
–
Wait_for_ _terminating_ _answer
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
DataReqInd (UDArg)
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
dpUTSIArg!party := remoteParty
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–

Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Or2(3)at
/*
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_
Analysis2
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
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Failure
false
cdPNIx := cdPNIx+1
/*Mark next CDPN
as destination
routing address*/
Result :=
Busy
Result :=
Analyse_marked_
_CDPN
Result :=
Analyse_marked_
_CDPN

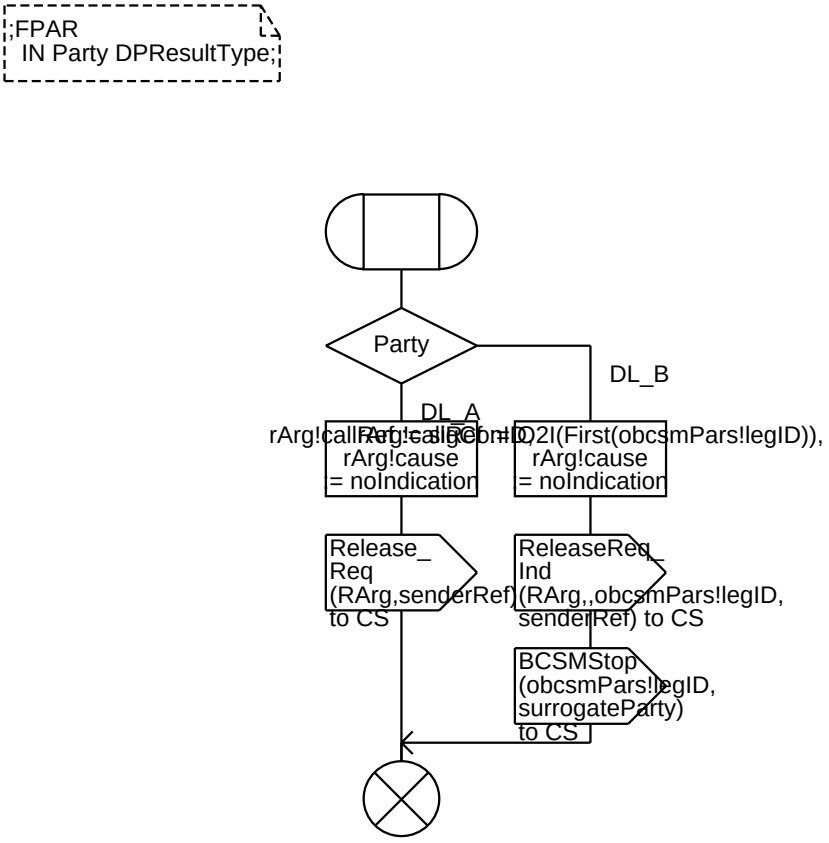
Annex A:

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 3(3)3

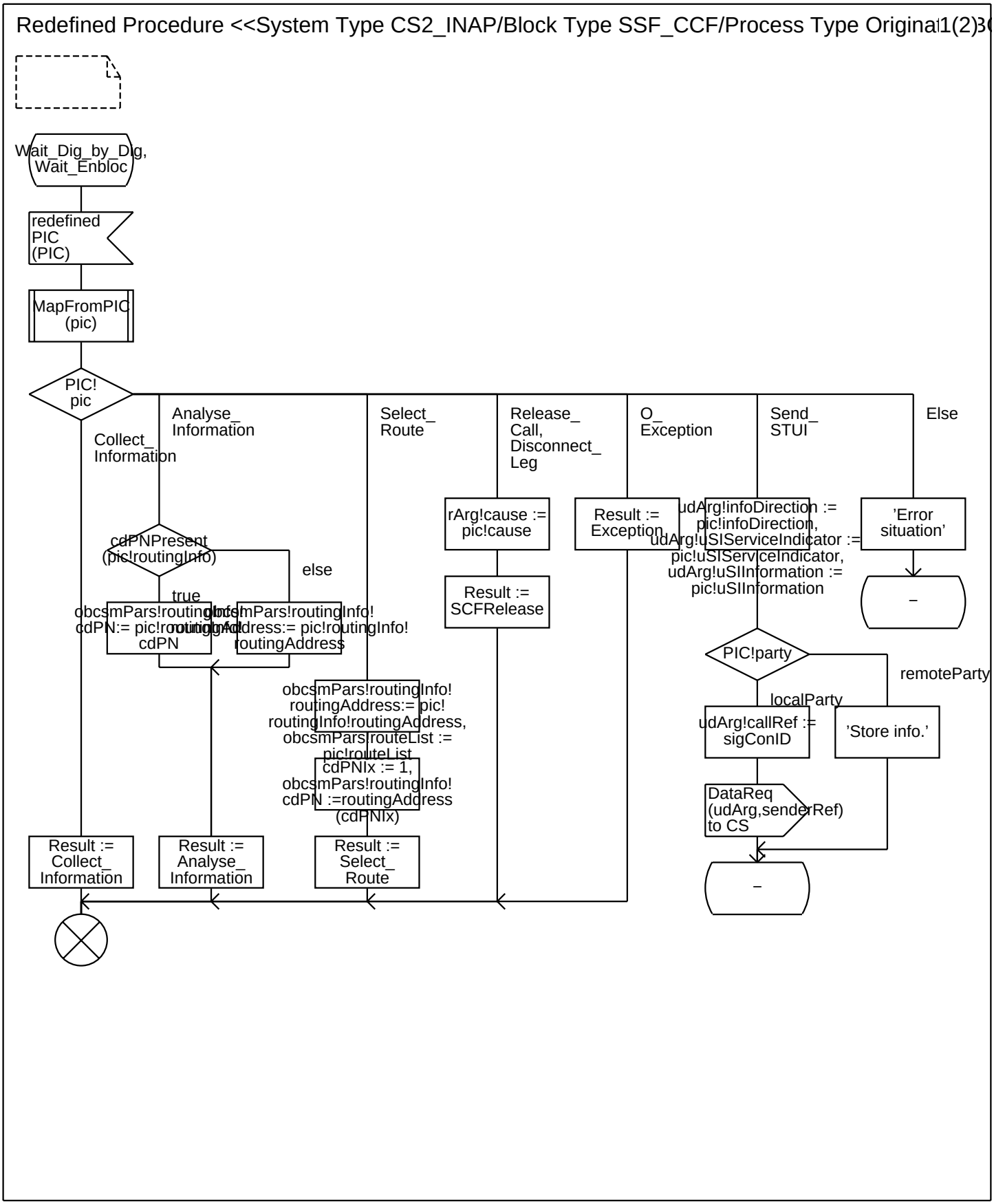


Annex A:

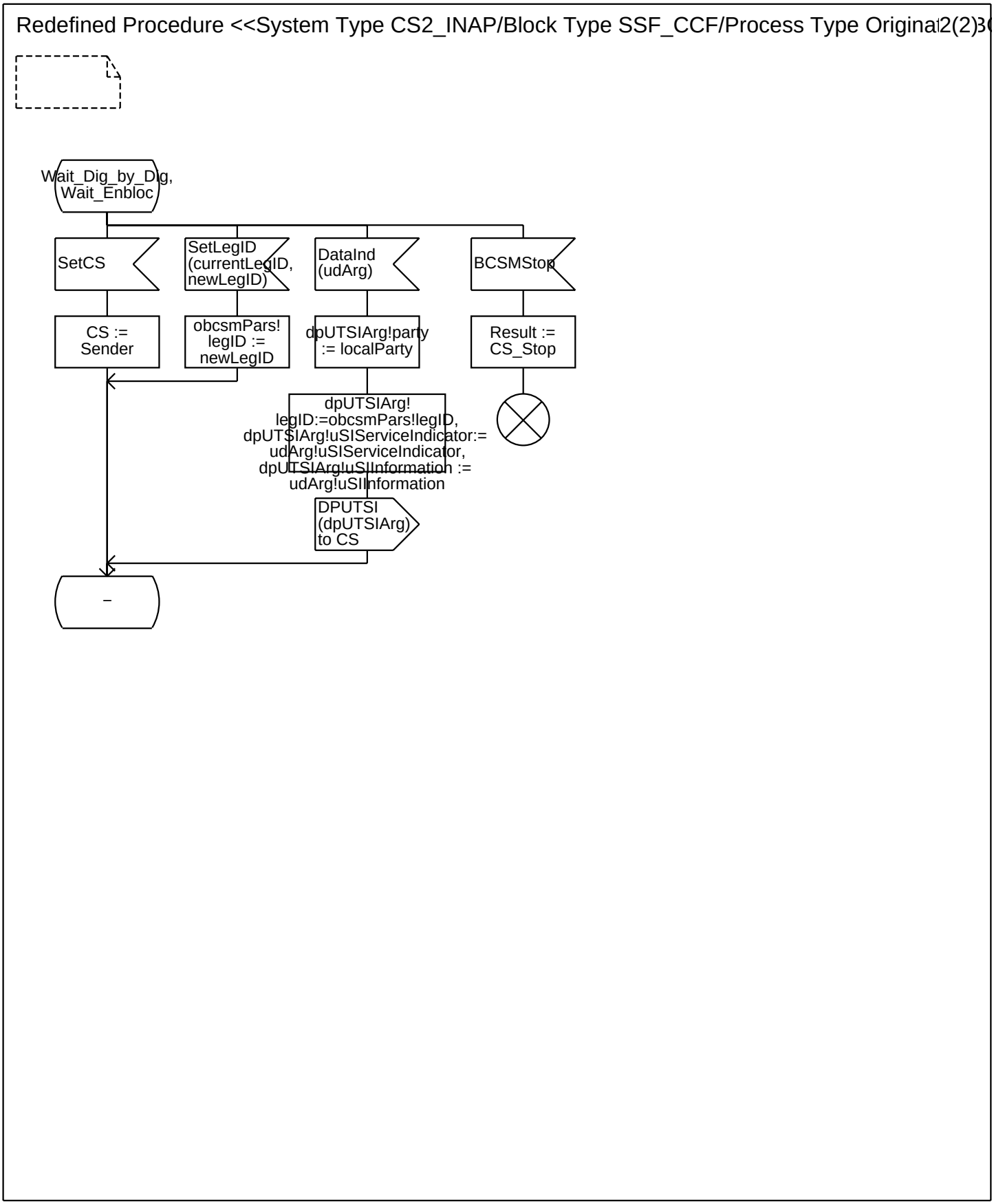
Procedure <<Process Type OriginatingBCSM>> PIC_Disconnect 1(1)



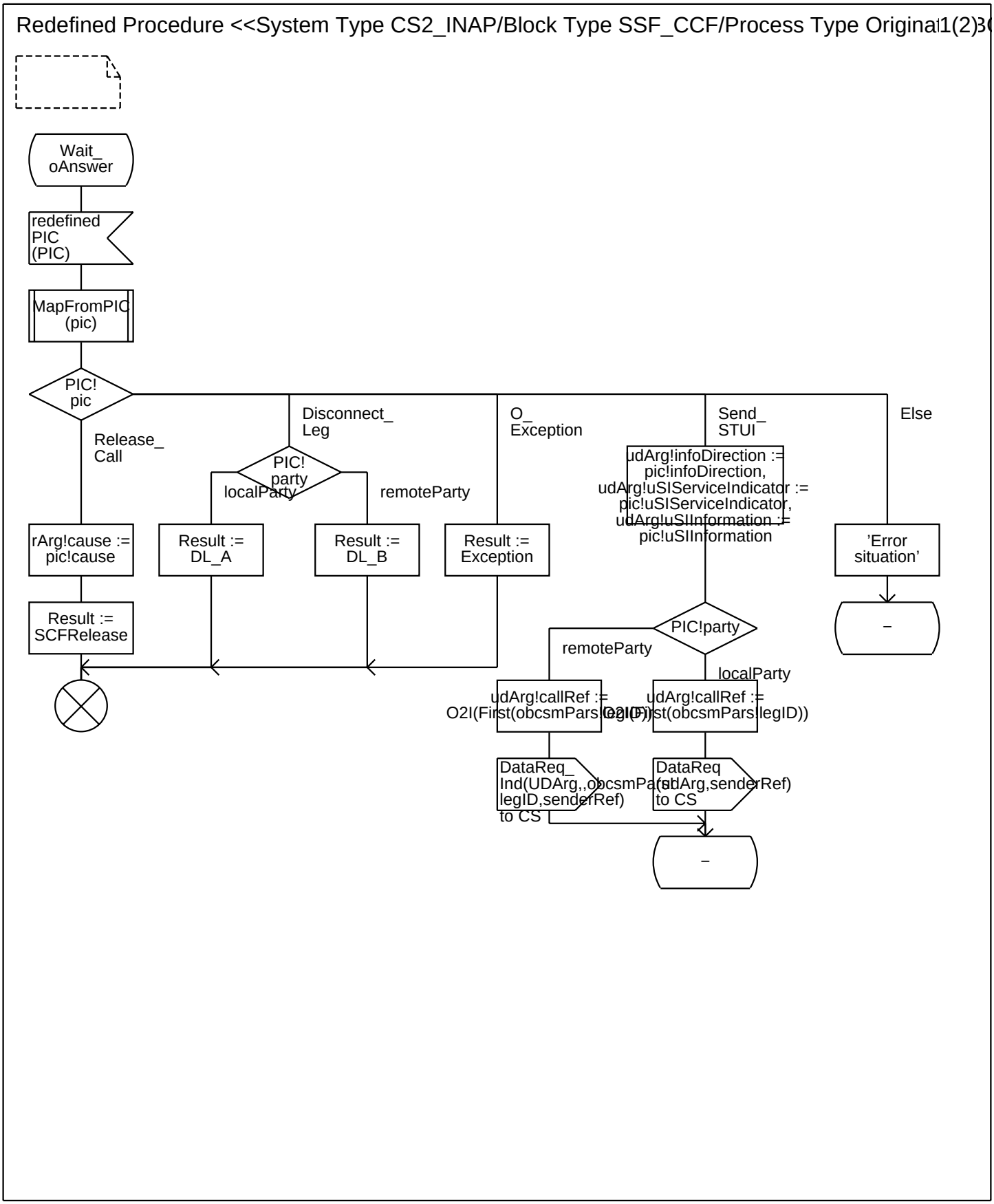
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
Wait_Dig_by_Dig, Wait_Enbloc
redefined PIC (PIC)
MapFromPIC (pic)
PIC! pic
Collect_ Information
Analyse_ Information
Select_ Route
Release_ Call, Disconnect_ Leg
O_ Exception
Send_ STUI
Else
cdPNPresent (pic!routingInfo)
true
else
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
obcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
obcsmPars!routingInfo! routingAddress:= pic! routingInfo!routingAddress, obcsmPars!routeList := pic!routeList
cdPNIx := 1, obcsmPars!routingInfo! cdPN :=routingAddress (cdPNIx)
rArg!cause := pic!cause
Result := SCFRelease
Result := Exception
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
'Error situation'
–
PIC!party
remoteParty
localParty
udArg!callRef := sigConID
'Store info.'
DataReq (udArg,senderRef) to CS
–
Result := Collect_ Information
Result := Analyse_ Information
Result := Select_ Route

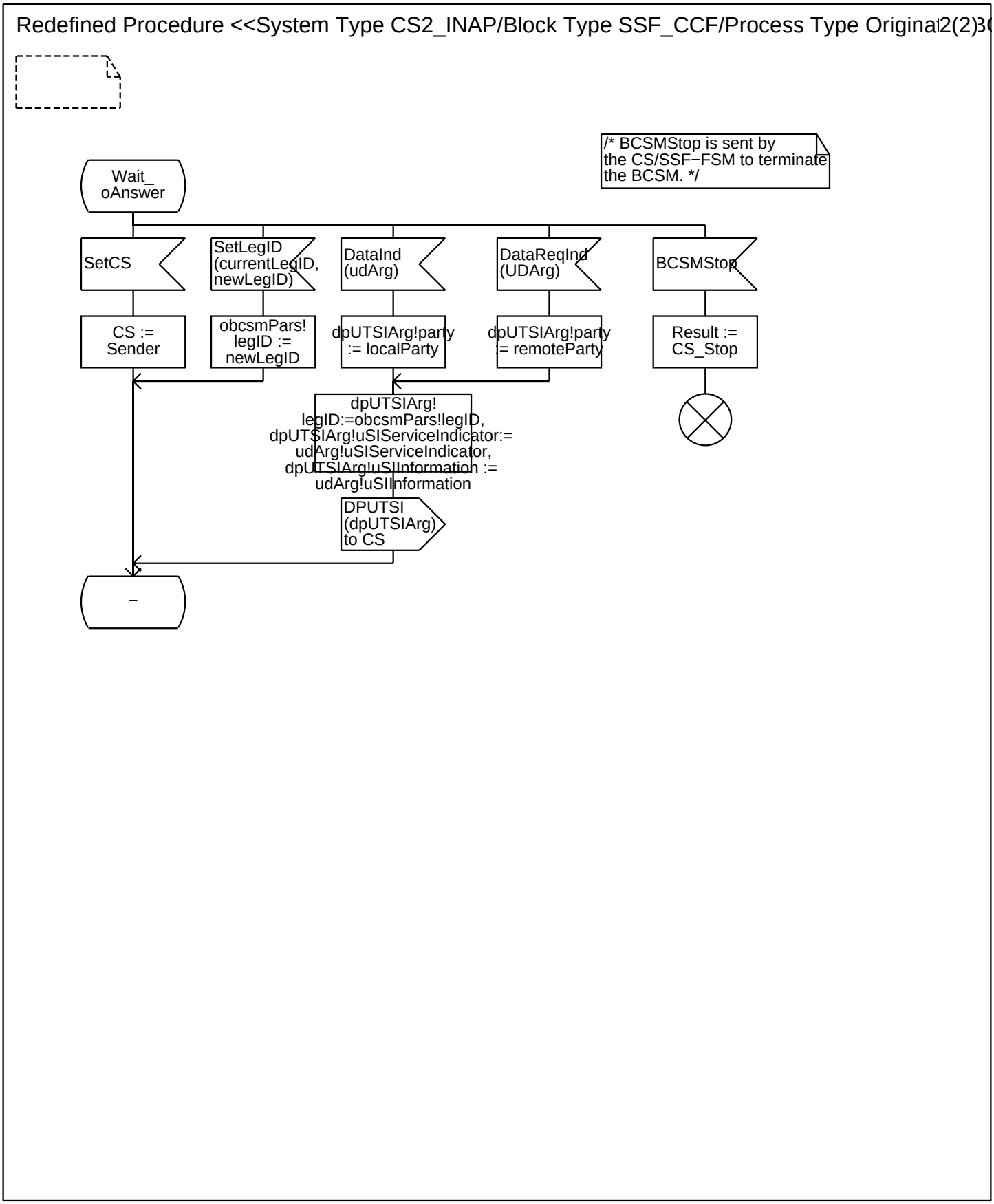
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(2)
Wait_Dig_by_Dig,
Wait_Enbloc
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
BCSMStop
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
Result :=
CS_Stop
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:

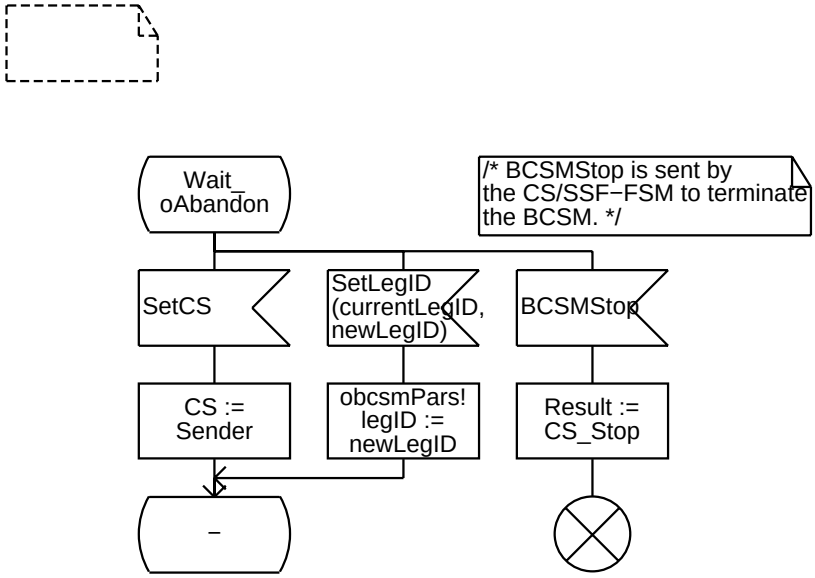

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
Wait_ oAnswer
redefined PIC (PIC)
MapFromPIC (pic)
PIC! pic
Release_ Call
Disconnect_ Leg
O_ Exception
Send_ STUI
Else
PIC! party
localParty
remoteParty
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
rArg!cause := pic!cause
Result := DL_A
Result := DL_B
Result := Exception
'Error situation'
Result := SCFRelease
PIC!party
remoteParty
localParty
udArg!callRef := O2I(First(obcsmParsIegID))
udArg!callRef := O2I(First(obcsmParsIegID))
DataReq_ Ind(UDArg,,obcsmParsIegID,senderRef) to CS
DataReq (udArg,senderRef) to CS

Annex A:

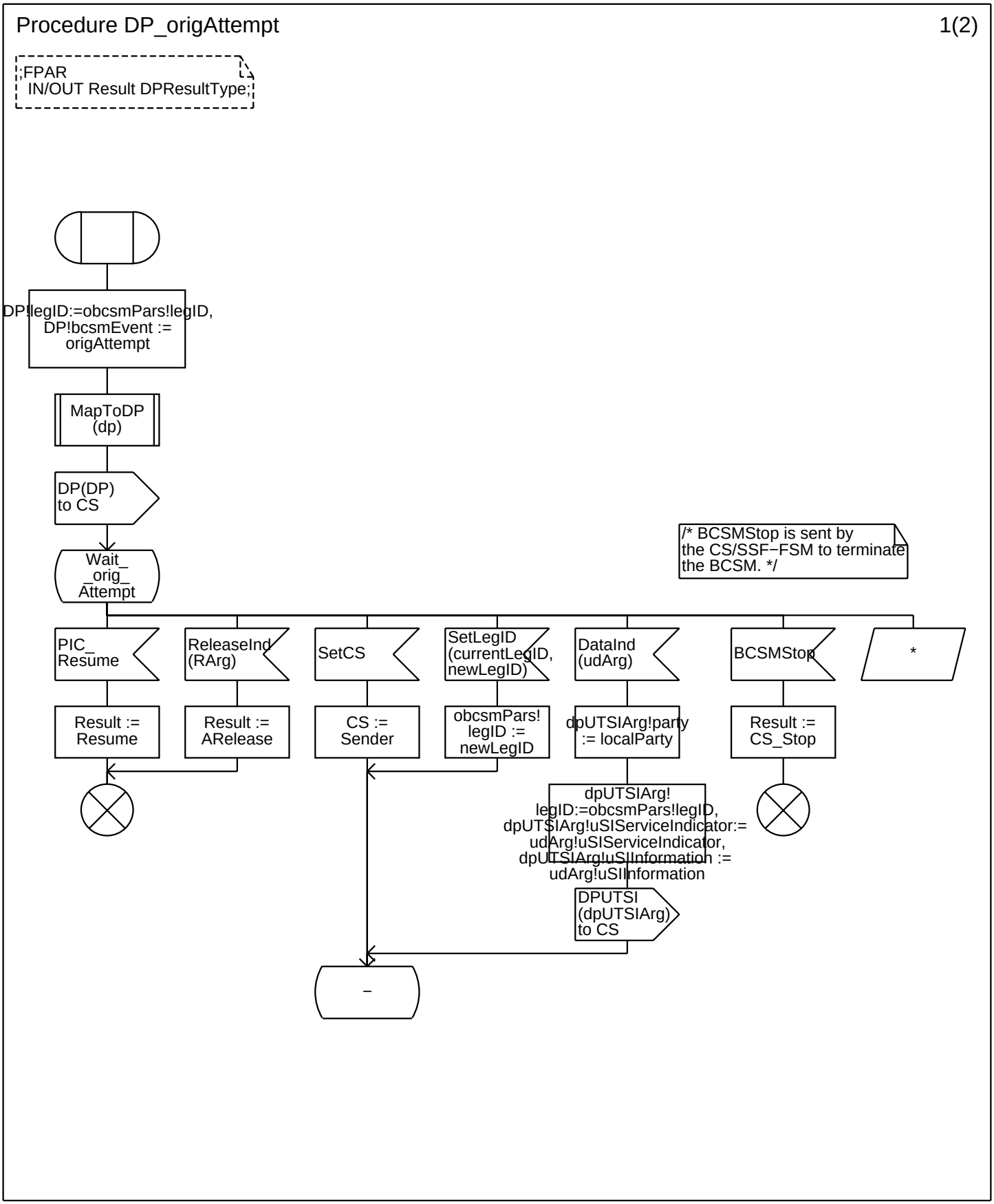

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original2(2)3
/* BCSMStop is sent by the CS/SSF−FSM to terminate the BCSM. */
Wait_ oAnswer
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
DataReqInd (UDArg)
BCSMStop
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
dpUTSIArg!party := remoteParty
Result := CS_Stop
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–

Annex A:

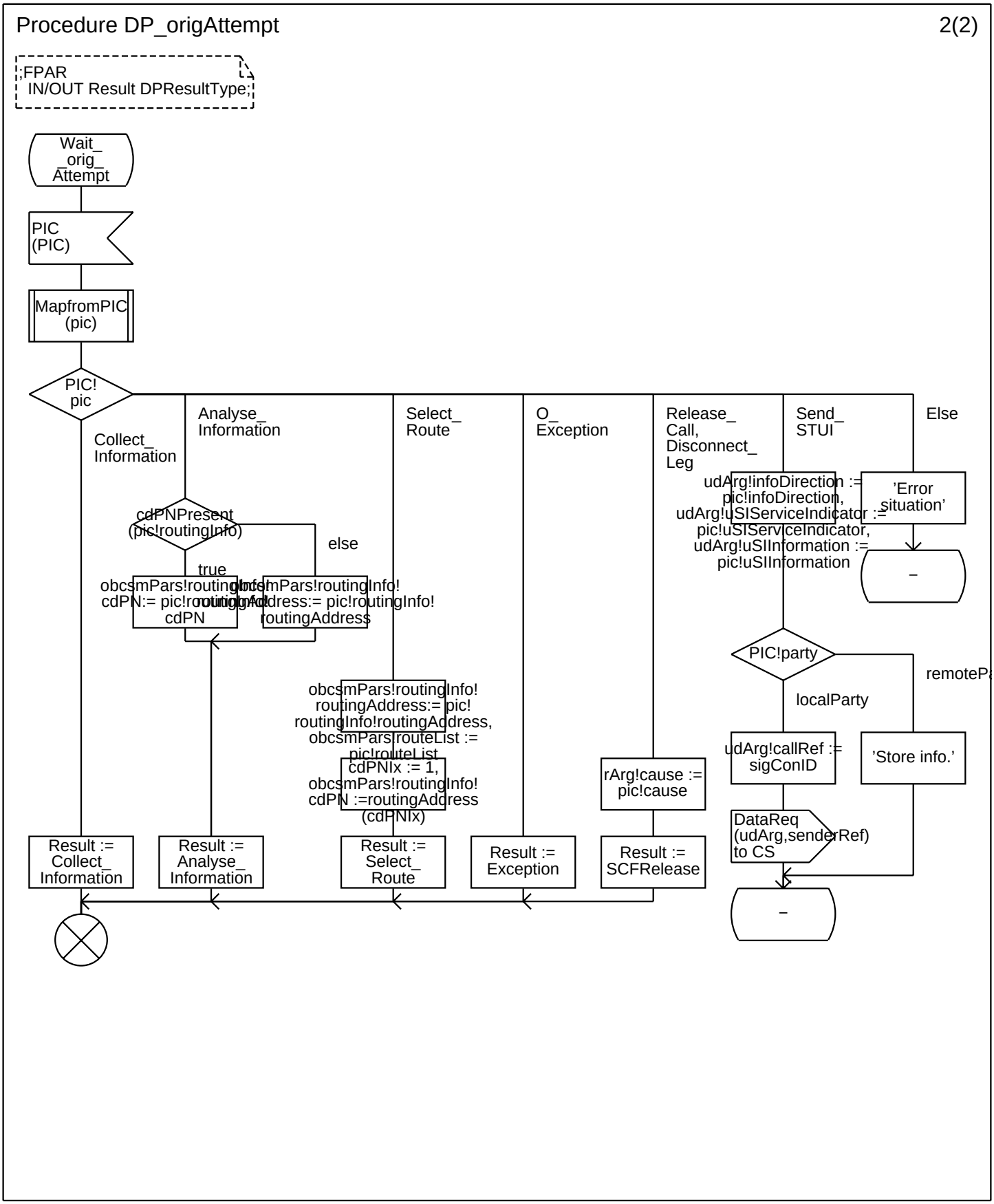
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(1)3



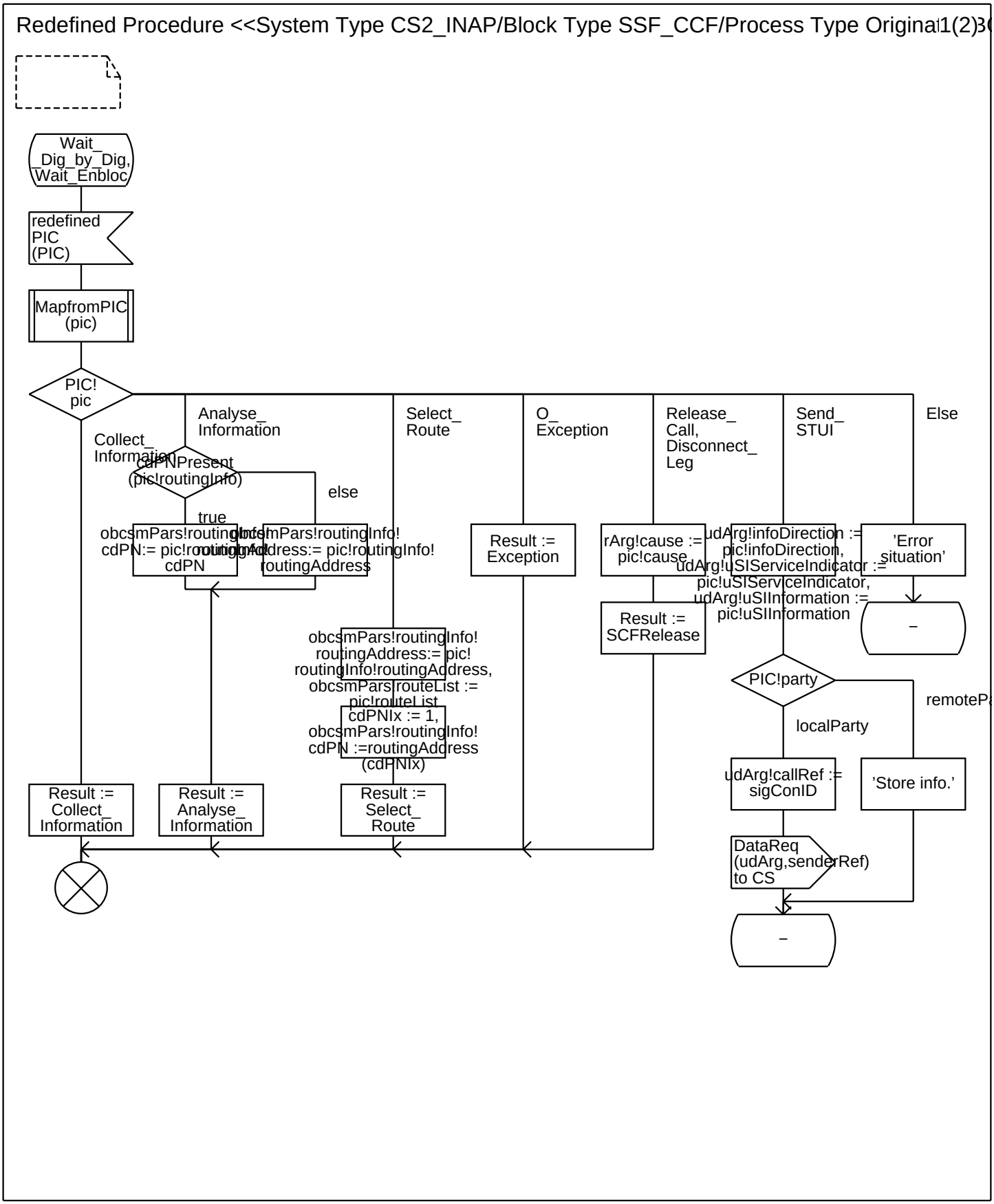
Annex A:


Procedure DP_origAttempt
1(2)
;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
origAttempt
MapToDP
(dp)
DP(DP)
to CS
Wait_
orig
Attempt
/* BCSMStop is sent by
the CS/SSF−FSM to terminate
the BCSM. */
PIC_
Resume
ReleaseInd
(RArg)
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
BCSMStop
*
Result :=
Resume
Result :=
ARelease
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
Result :=
CS_Stop
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

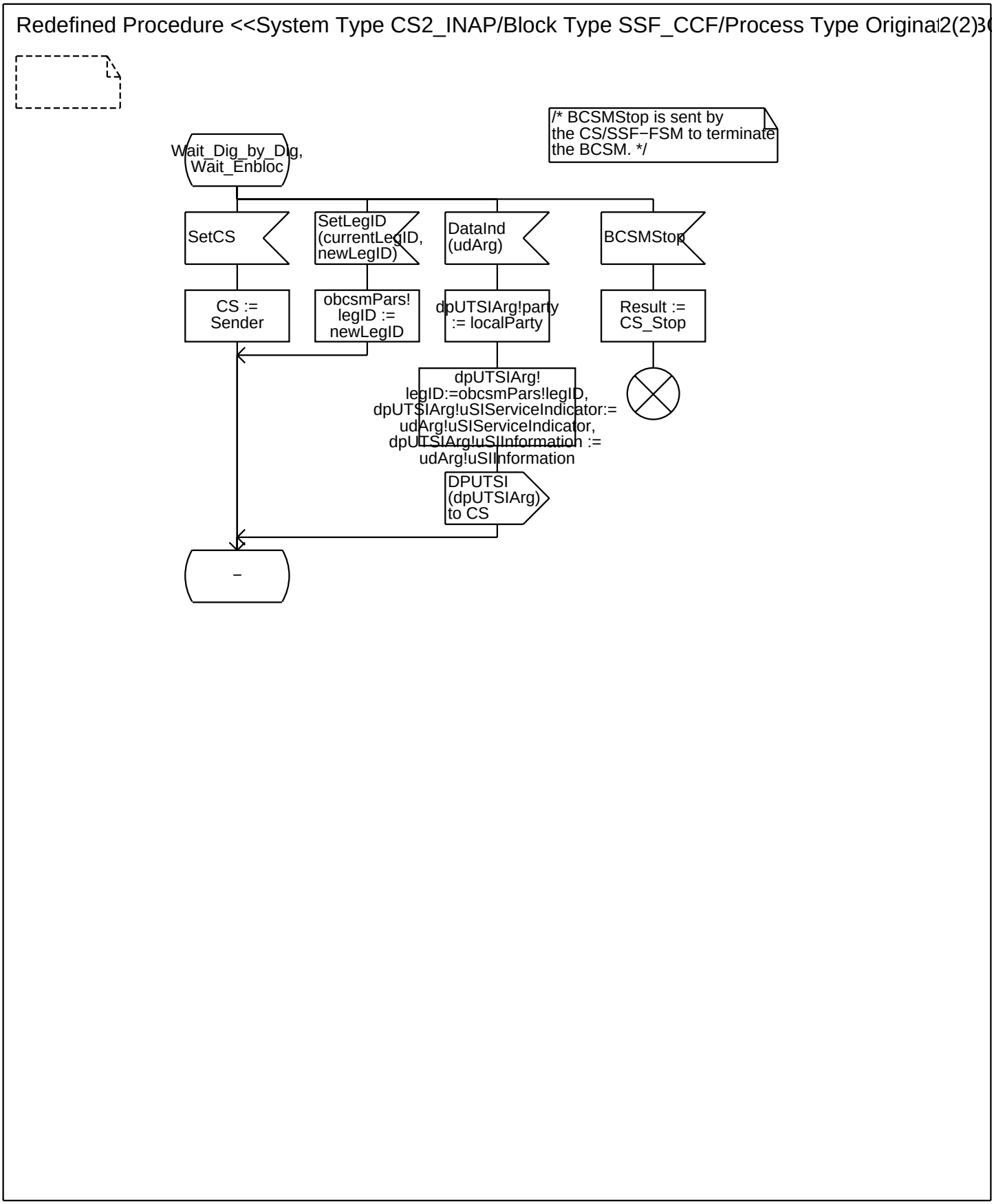
Annex A:


Procedure DP_origAttempt
2(2)
;FPAR
IN/OUT Result DPResultType;
Wait_
orig
Attempt
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Collect_
Information
Analyse_
Information
Select_
Route
O_
Exception
Release_
Call,
Disconnect_
Leg
Send_
STUI
Else
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList,
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
–
PIC!party
remoteP
localParty
udArg!callRef :=
sigConID
'Store info.'
DataReq
(udArg,senderRef)
to CS
–
Result :=
Collect_
Information
Result :=
Analyse_
Information
Result :=
Select_
Route
Result :=
Exception
Result :=
SCFRelease

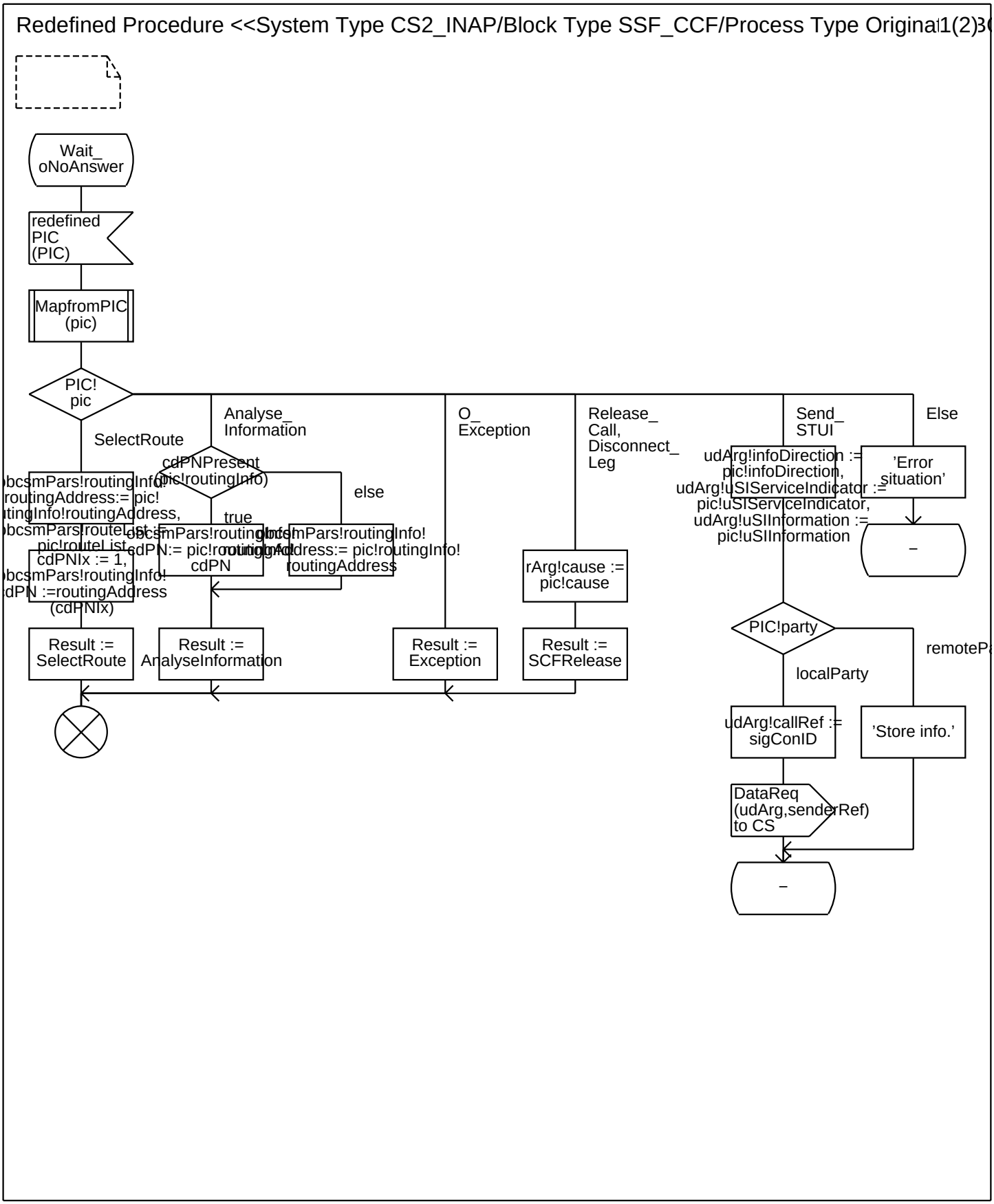
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
Wait_ _Dig_by_Dig, Wait_Enbloc
redefined PIC (PIC)
MapfromPIC (pic)
PIC! pic
Collect_ Information
Analyse_ Information
Select_ Route
O_ Exception
Release_ Call, Disconnect_ Leg
Send_ STUI
Else
cdPNPresent (pic!routingInfo)
else
true
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
obcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
Result := Exception
rArg!cause := pic!cause
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
'Error situation'
Result := SCFRelease
obcsmPars!routingInfo! routingAddress:= pic! routingInfo!routingAddress, obcsmPars!routeList := pic!routeList
cdPNIx := 1, obcsmPars!routingInfo! cdPN :=routingAddress (cdPNIx)
PIC!party
remoteP
localParty
udArg!callRef := sigConID
'Store info.'
DataReq (udArg,senderRef) to CS
Result := Collect_ Information
Result := Analyse_ Information
Result := Select_ Route

Annex A:

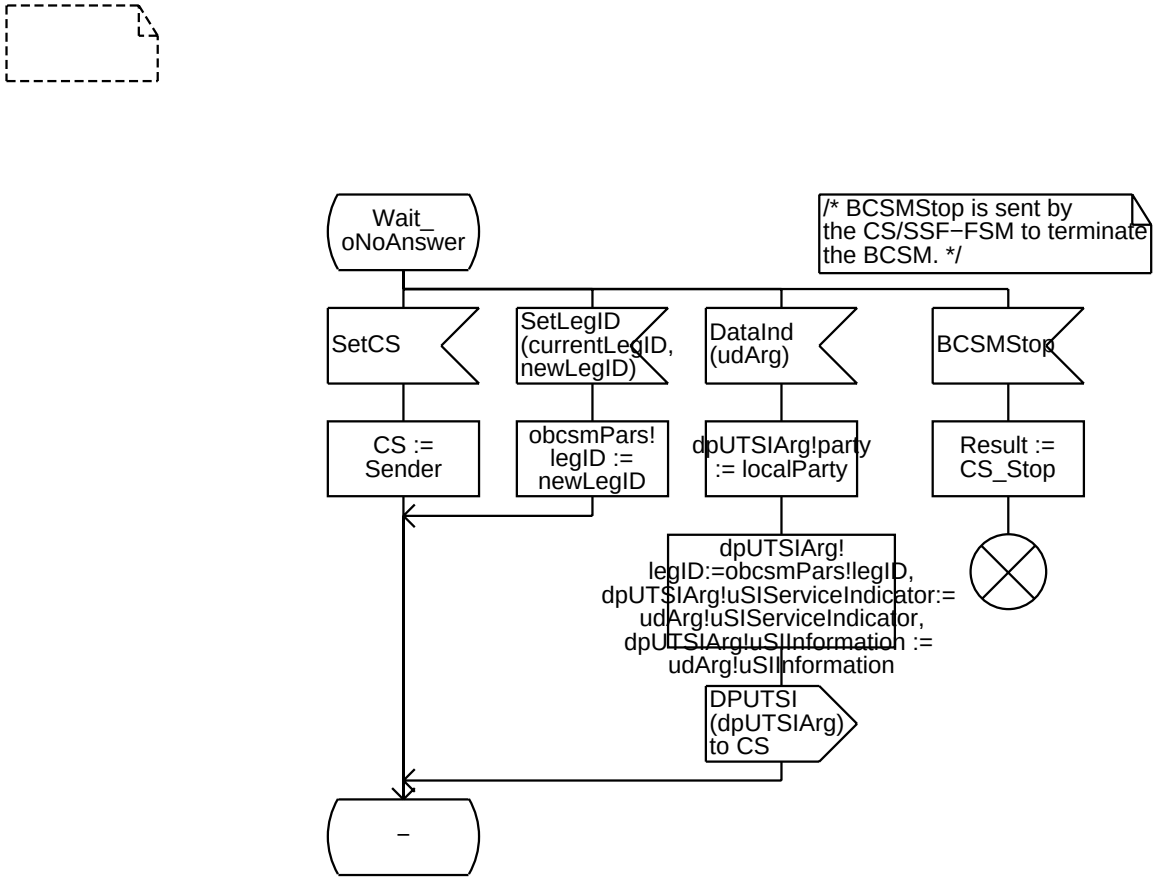

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(2)3
/* BCSMStop is sent by the CS/SSF−FSM to terminate the BCSM. */
Wait_Dig_by_Dig, Wait_Enbloc
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
BCSMStop
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
Result := CS_Stop
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–

Annex A:

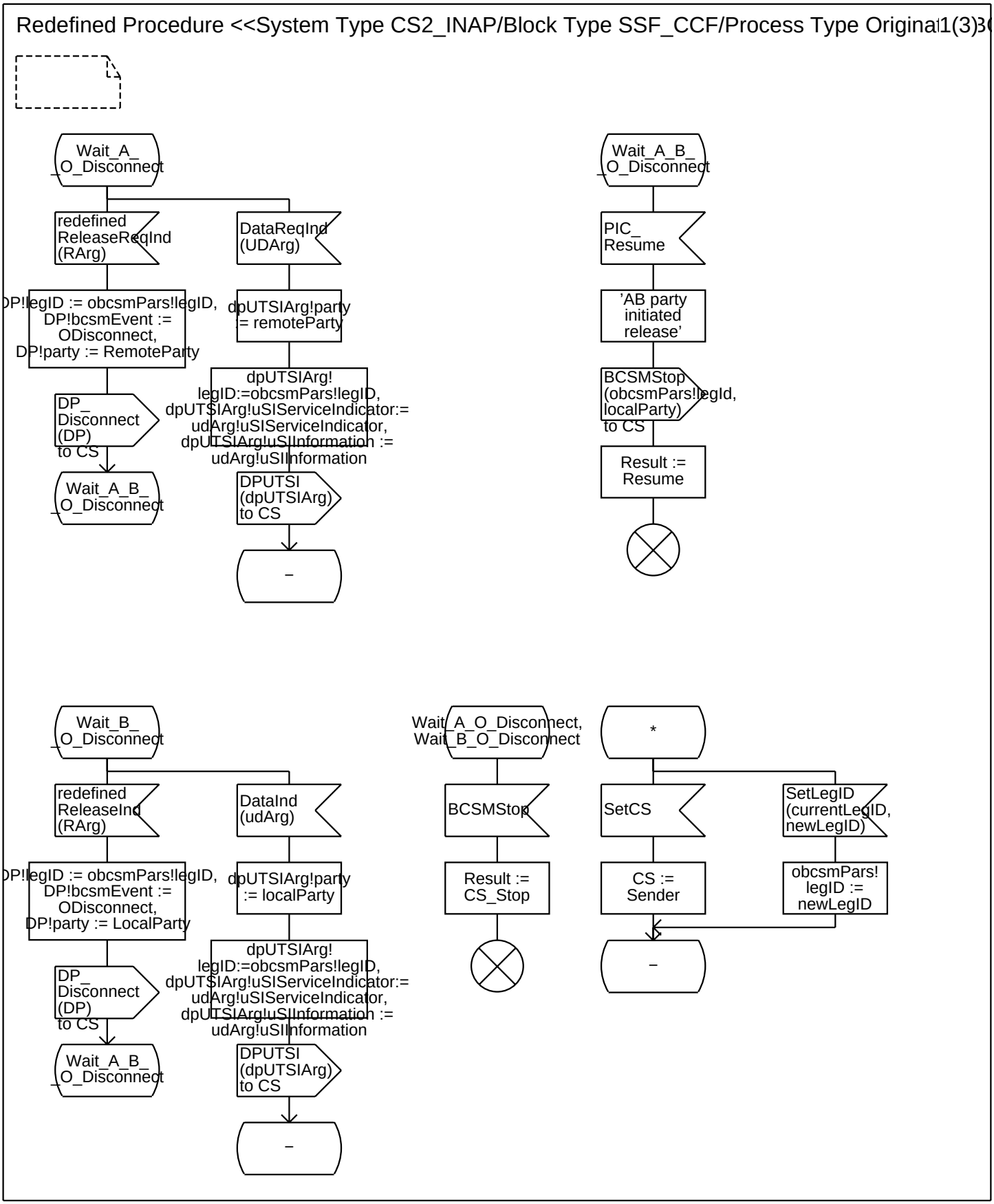

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
Wait_ oNoAnswer
redefined PIC (PIC)
MapfromPIC (pic)
PIC! pic
SelectRoute
Analyse_ Information
O_ Exception
Release_ Call, Disconnect_ Leg
Send_ STUI
Else
cdPNPresent (pic!routingInfo)
else
true
bcsmPars!routingInfo! routingAddress:= pic! tingInfo!routingAddress, bcsmPars!routeList := pic!routeList
cdPNIx := 1, bcsmPars!routingInfo! dPN :=routingAddress (cdPNIx)
bcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
bcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
rArg!cause := pic!cause
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
'Error situation'
–
Result := SelectRoute
Result := AnalyseInformation
Result := Exception
Result := SCFRelease
PIC!party
remoteP
localParty
udArg!callRef := sigConID
'Store info.'
DataReq (udArg,senderRef) to CS
–

Annex A:

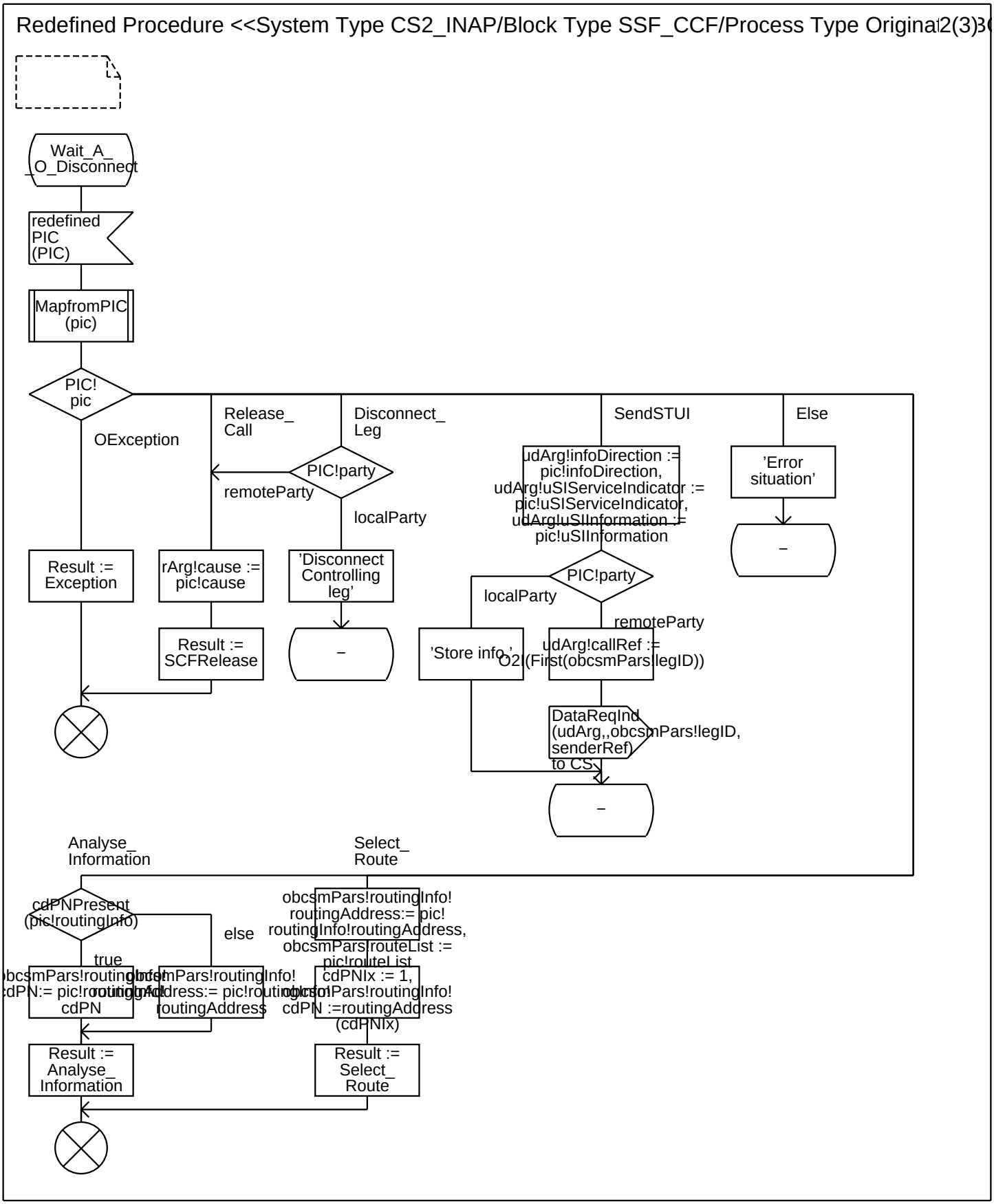
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(2)3



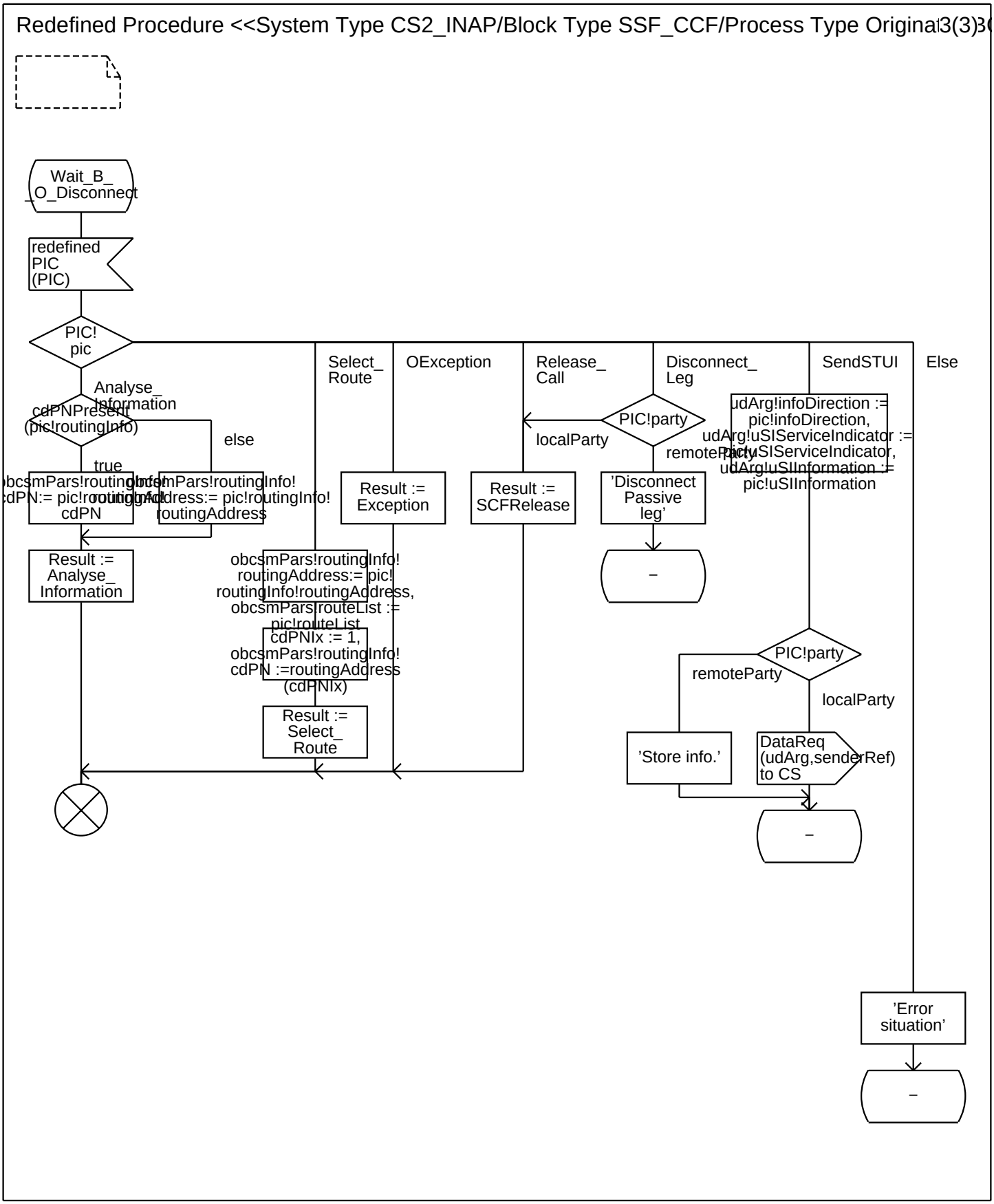
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(3)3
Wait_A_O_Disconnect
redefined ReleaseReqInd (RArg)
DP!legID := obcsmPars!legID, DP!bcsmEvent := ODisconnect, DP!party := RemoteParty
DP_Disconnect (DP) to CS
Wait_A_B_O_Disconnect
DataReqInd (UDArg)
dpUTSIArg!party := remoteParty
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–
Wait_A_B_O_Disconnect
PIC_Resume
'AB party initiated release'
BCSMStop (obcsmPars!legId, localParty) to CS
Result := Resume
Wait_B_O_Disconnect
redefined ReleaseInd (RArg)
DP!legID := obcsmPars!legID, DP!bcsmEvent := ODisconnect, DP!party := LocalParty
DP_Disconnect (DP) to CS
Wait_A_B_O_Disconnect
DataInd (udArg)
dpUTSIArg!party := localParty
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–
Wait_A_O_Disconnect, Wait_B_O_Disconnect
BCSMStop
Result := CS_Stop
*
SetCS
CS := Sender
SetLegID (currentLegID, newLegID)
obcsmPars! legID := newLegID
–

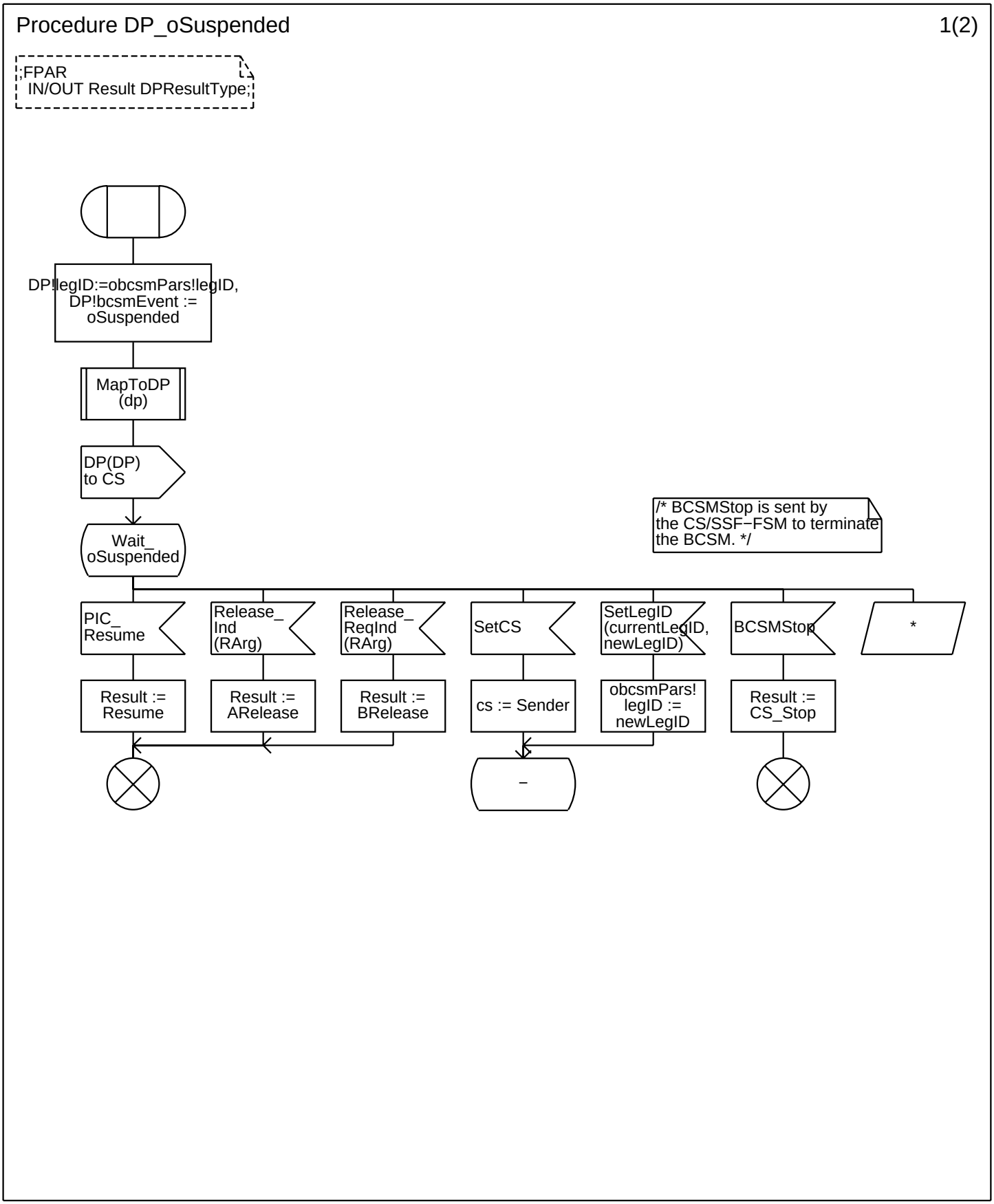
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(3)
Wait_A_
O_Disconnect
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
OException
Release_
Call
Disconnect_
Leg
SendSTUI
Else
PIC!party
remoteParty
localParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
Exception
rArg!cause :=
pic!cause
'Disconnect
Controlling
leg'
Result :=
SCFRelease
PIC!party
localParty
remoteParty
'Store info.'
udArg!callRef :=
O2I(First(obcsmPars!legID))
DataReqInd
(udArg,,obcsmPars!legID,
senderRef)
to CS
Analyse_
Information
Select_
Route
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
cdPN
Result :=
Analyse_
Information
Result :=
Select_
Route

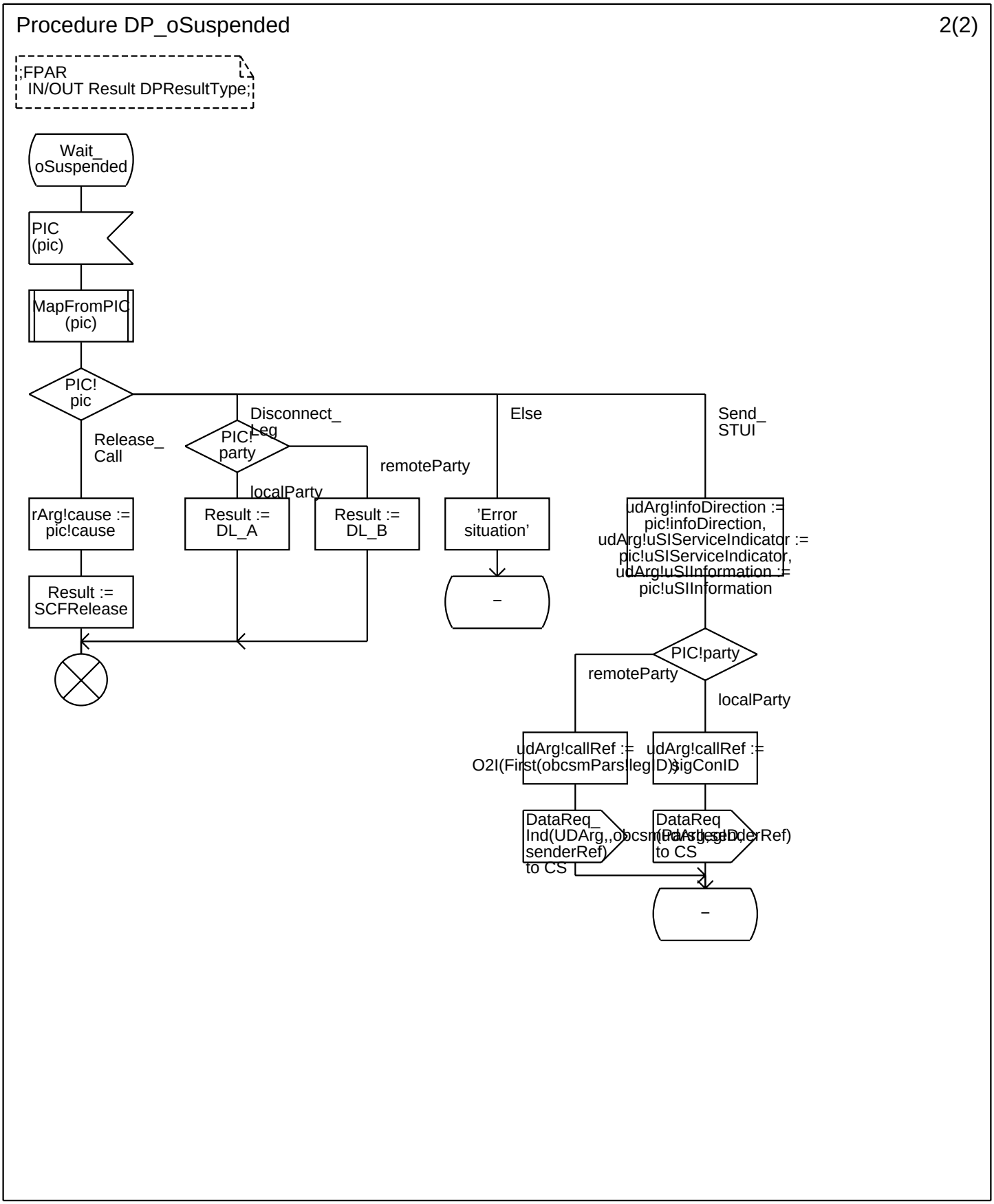
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 3(3)3
Wait_B_
_O_Disconnect
redefined
PIC
(PIC)
PIC!
pic
Analyse_
Information
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Analyse_
Information
Select_
Route
OException
Release_
Call
Disconnect_
Leg
SendSTUI
Else
Result :=
Exception
Result :=
SCFRelease
PIC!party
localParty
remoteParty
'Disconnect
Passive
leg'
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
PIC!party
remoteParty
localParty
'Store info.'
DataReq
(udArg,senderRef)
to CS
'Error
situation'

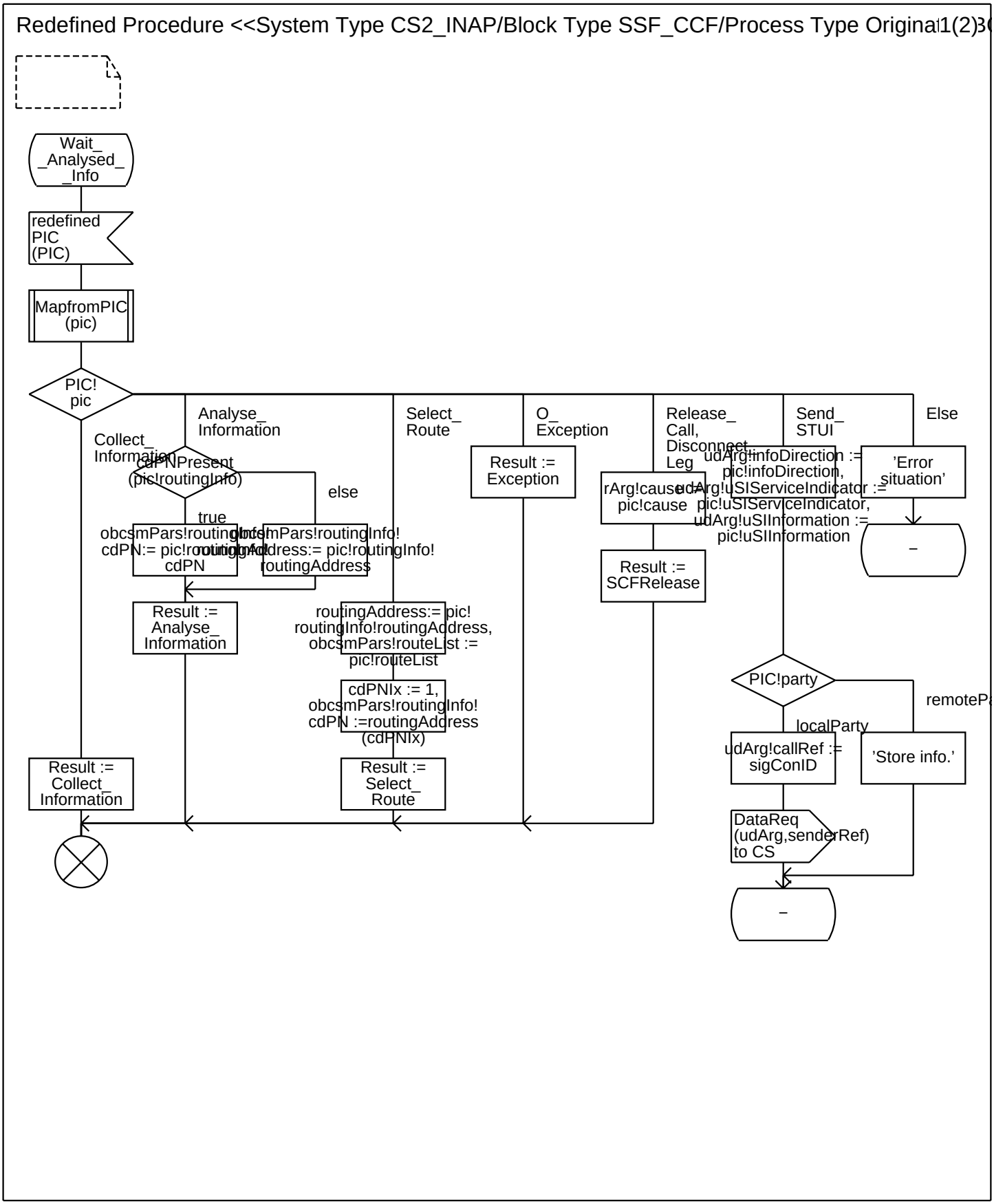
Annex A:


Procedure DP_oSuspended
1(2)
;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
oSuspended
MapToDP
(dp)
DP(DP)
to CS
Wait_
oSuspended
/* BCSMStop is sent by
the CS/SSF−FSM to terminate
the BCSM. */
PIC_
Resume
Release_
Ind
(RArg)
Release_
ReqInd
(RArg)
SetCS
SetLegID
(currentLegID,
newLegID)
BCSMStop
*
Result :=
Resume
Result :=
ARelease
Result :=
BRelease
cs := Sender
obcsmPars!
legID :=
newLegID
Result :=
CS_Stop
–

Annex A:

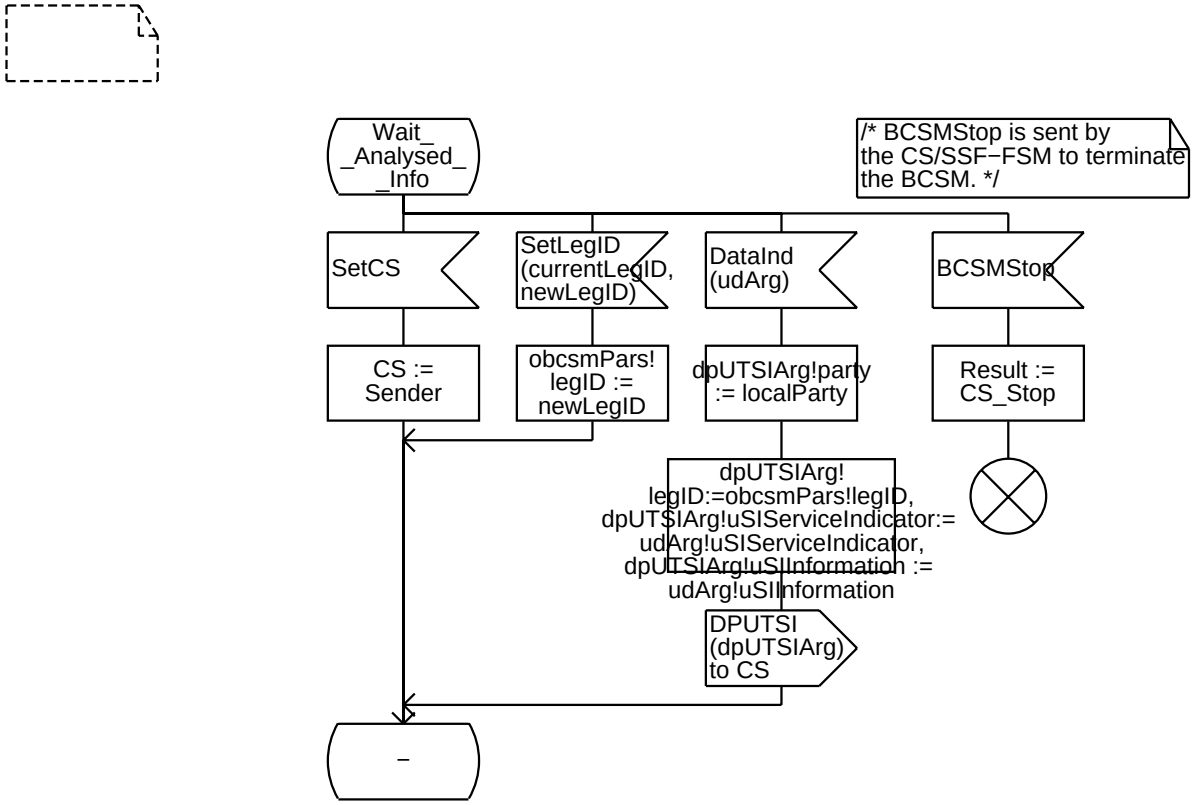

Procedure DP_oSuspended
2(2)
;FPAR
IN/OUT Result DPResultType;
Wait_
oSuspended
PIC
(pic)
MapFromPIC
(pic)
PIC!
pic
Release_
Call
Disconnect_
Leg
PIC!
party
remoteParty
localParty
Else
Send_
STUI
rArg!cause :=
pic!cause
Result :=
DL_A
Result :=
DL_B
'Error
situation'
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
Result :=
SCFRelease
PIC!party
remoteParty
localParty
udArg!callRef :=
O2I(First(obcsmParsleg D))
udArg!callRef :=
sigConID
DataReq_
Ind(UDArg,,obcsmParsleg,
senderRef)
to CS
DataReq
(UDArg,,sigConID,
senderRef)
to CS

Annex A:

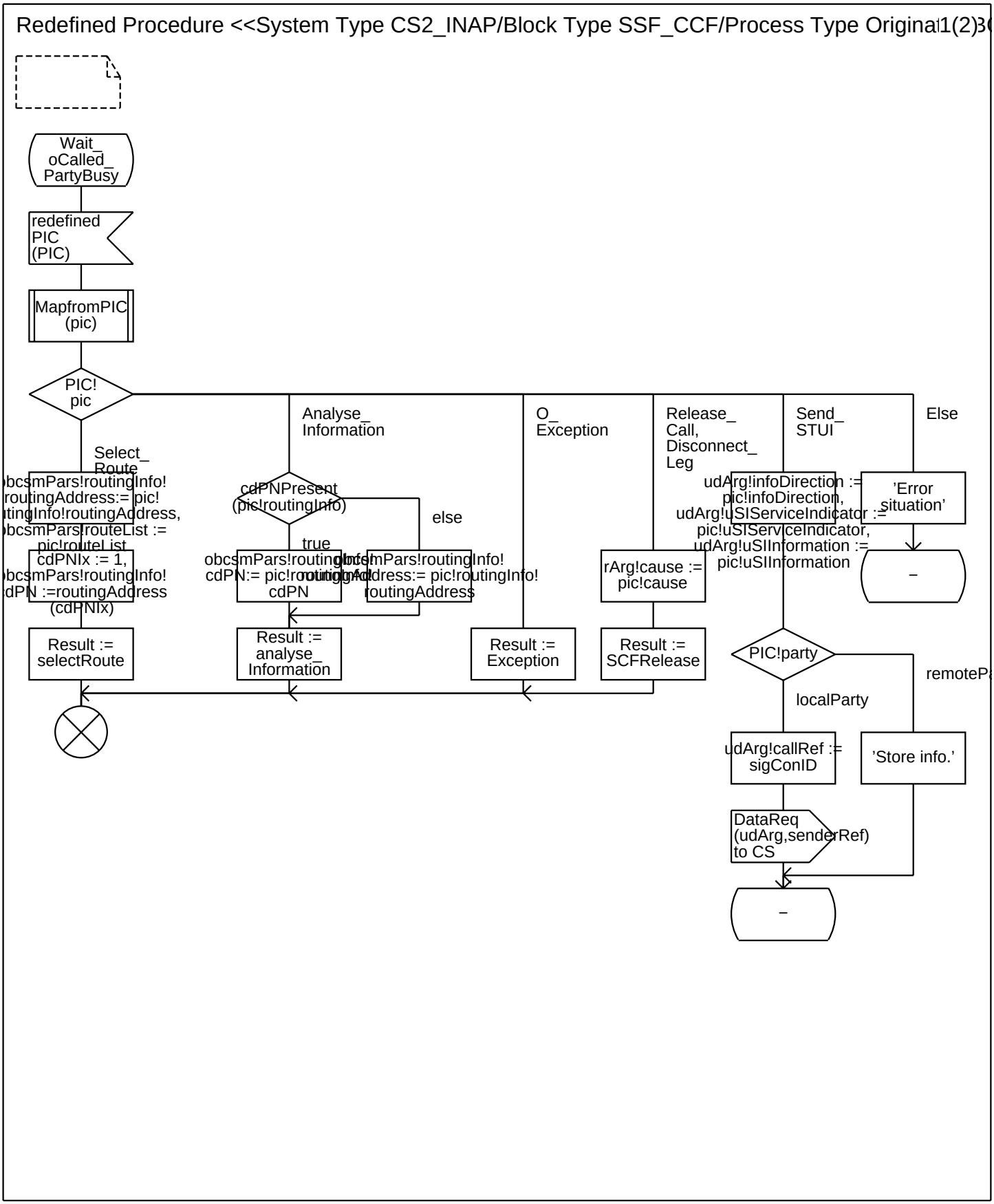

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
Wait_ _Analysed_ _Info
redefined PIC (PIC)
MapfromPIC (pic)
PIC! pic
Collect_ Information
Analyse_ Information
Select_ Route
O_ Exception
Release_ Call, Disconnect_ Leg
Send_ STUI
Else
cdPNPresent (pic!routingInfo)
else
true
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
obcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
Result := Analyse_ Information
routingAddress:= pic! routingInfo!routingAddress, obcsmPars!routeList := pic!routeList
cdPNIx := 1, obcsmPars!routingInfo! cdPN :=routingAddress (cdPNIx)
Result := Select_ Route
Result := Collect_ Information
Result := Exception
rArg!cause := pic!cause
Result := SCFRelease
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
'Error situation'
–
PIC!party
remoteP
localParty
udArg!callRef := sigConID
'Store info.'
DataReq (udArg,senderRef) to CS
–

Annex A:

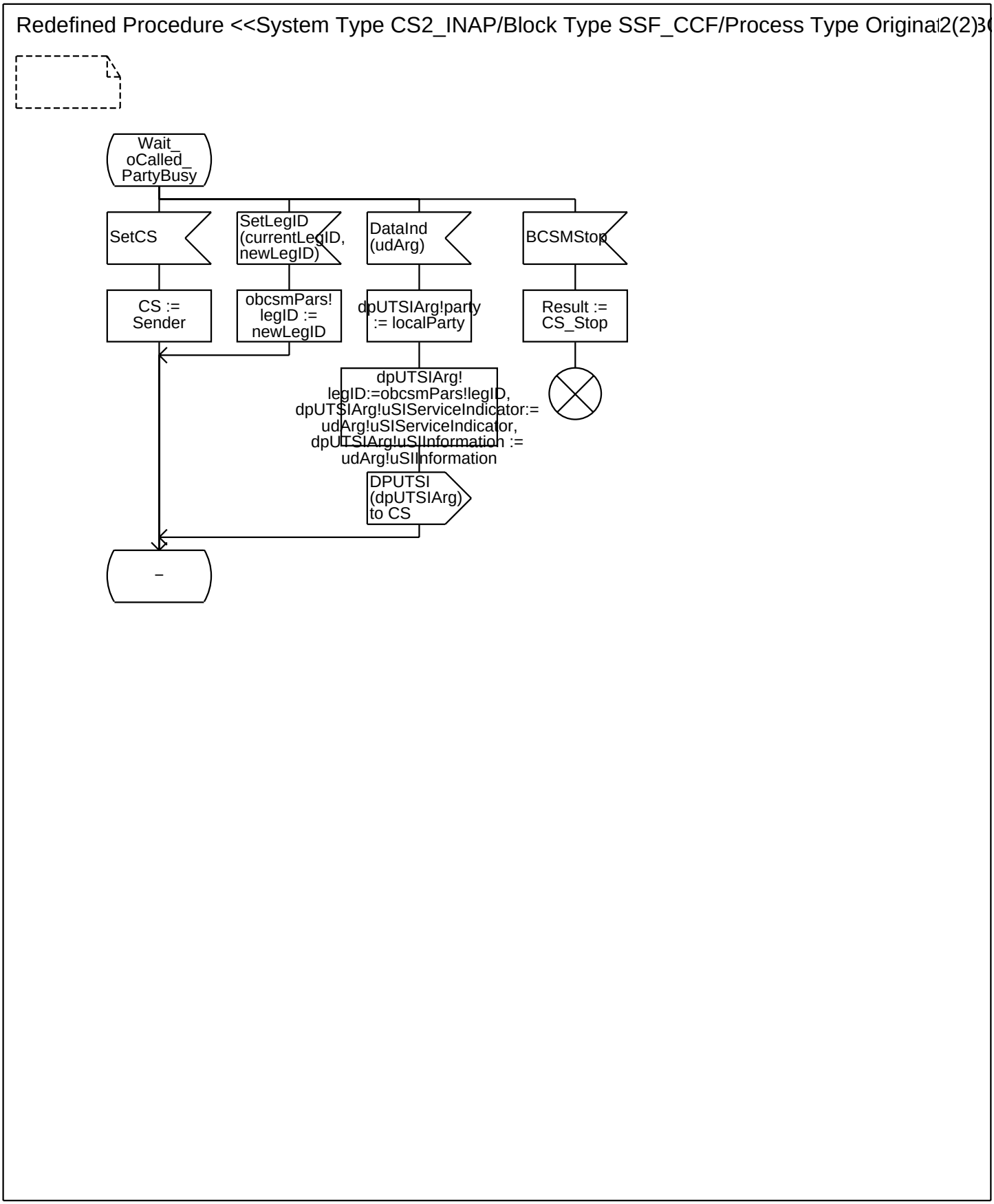
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(2)3



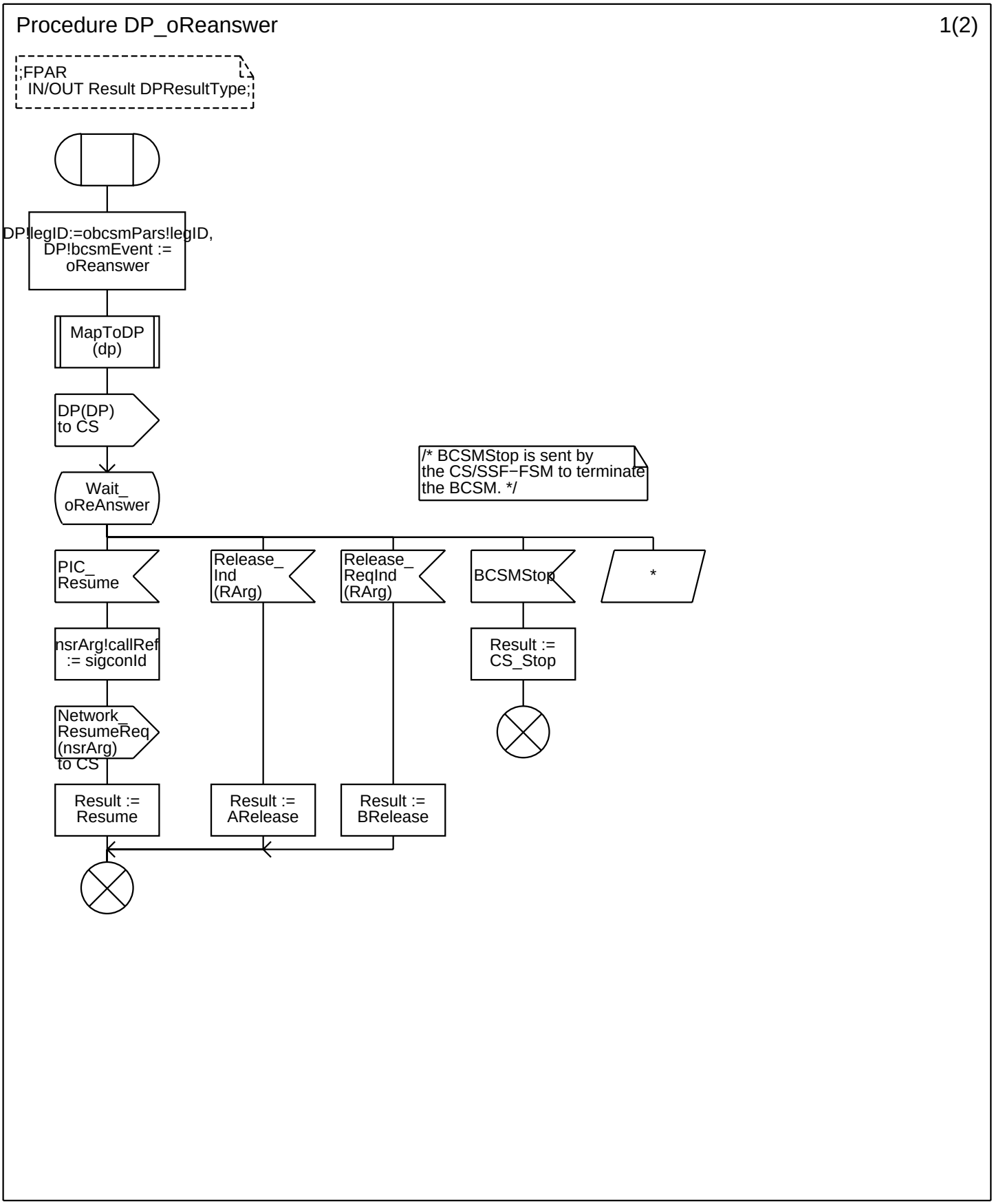
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
Wait_ oCalled_ PartyBusy
redefined PIC (PIC)
MapfromPIC (pic)
PIC! pic
Select_ Route
Analyse_ Information
O_ Exception
Release_ Call, Disconnect_ Leg
Send_ STUI
Else
bcsmPars!routingInfo! routingAddress:= pic! tingInfo!routingAddress, bcsmPars!routeList := pic!routeList
cdPNIx := 1, bcsmPars!routingInfo! dPN :=routingAddress (cdPNIx)
Result := selectRoute
cdPNPresent (pic!routingInfo)
true
else
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
bcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
Result := analyse_ Information
Result := Exception
rArg!cause := pic!cause
Result := SCFRelease
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
'Error situation'
–
PIC!party
localParty
remoteP
udArg!callRef := sigConID
'Store info.'
DataReq (udArg,senderRef) to CS
–

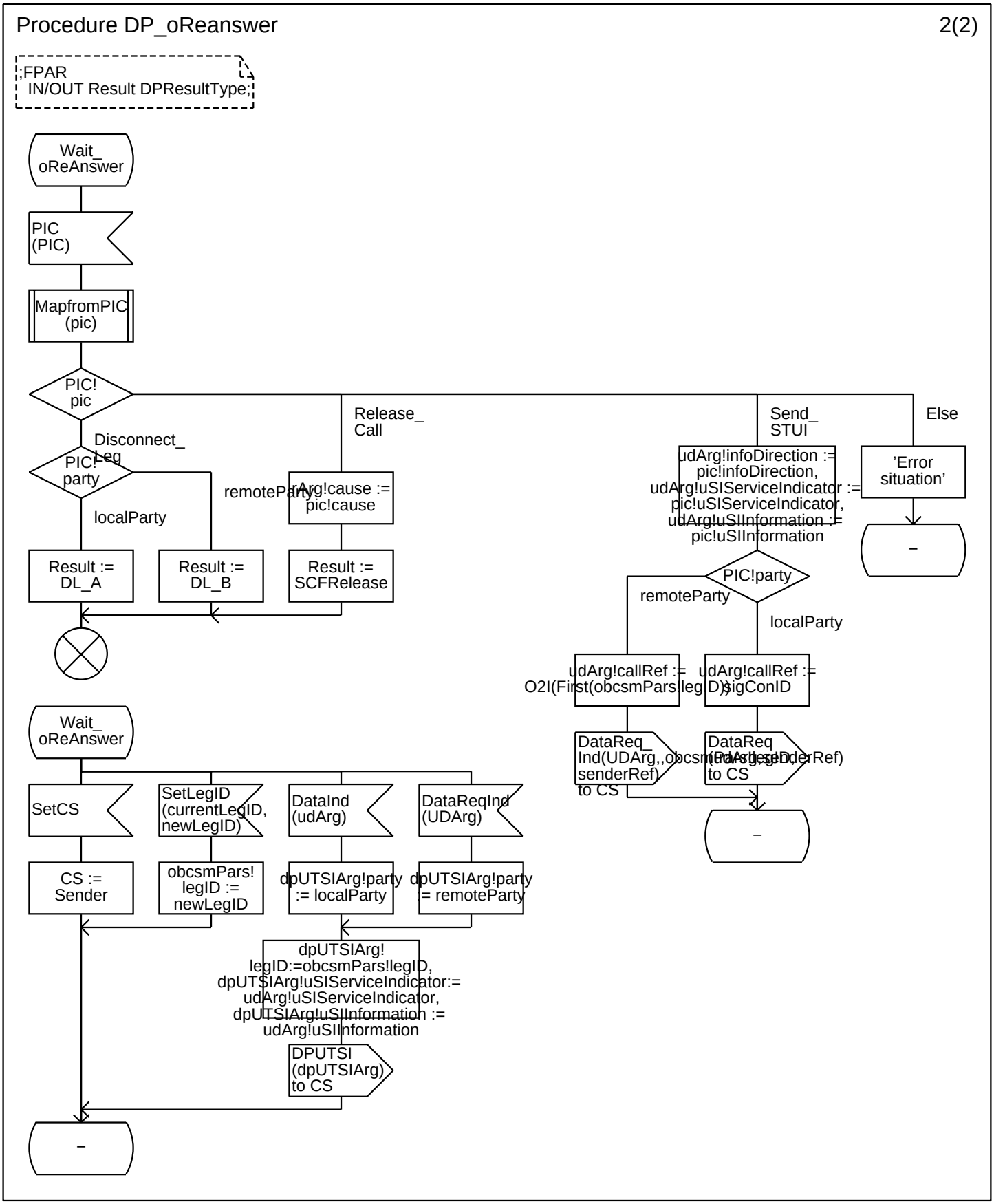
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(2)
Wait_ oCalled_ PartyBusy
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
BCSMStop
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
Result := CS_Stop
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–

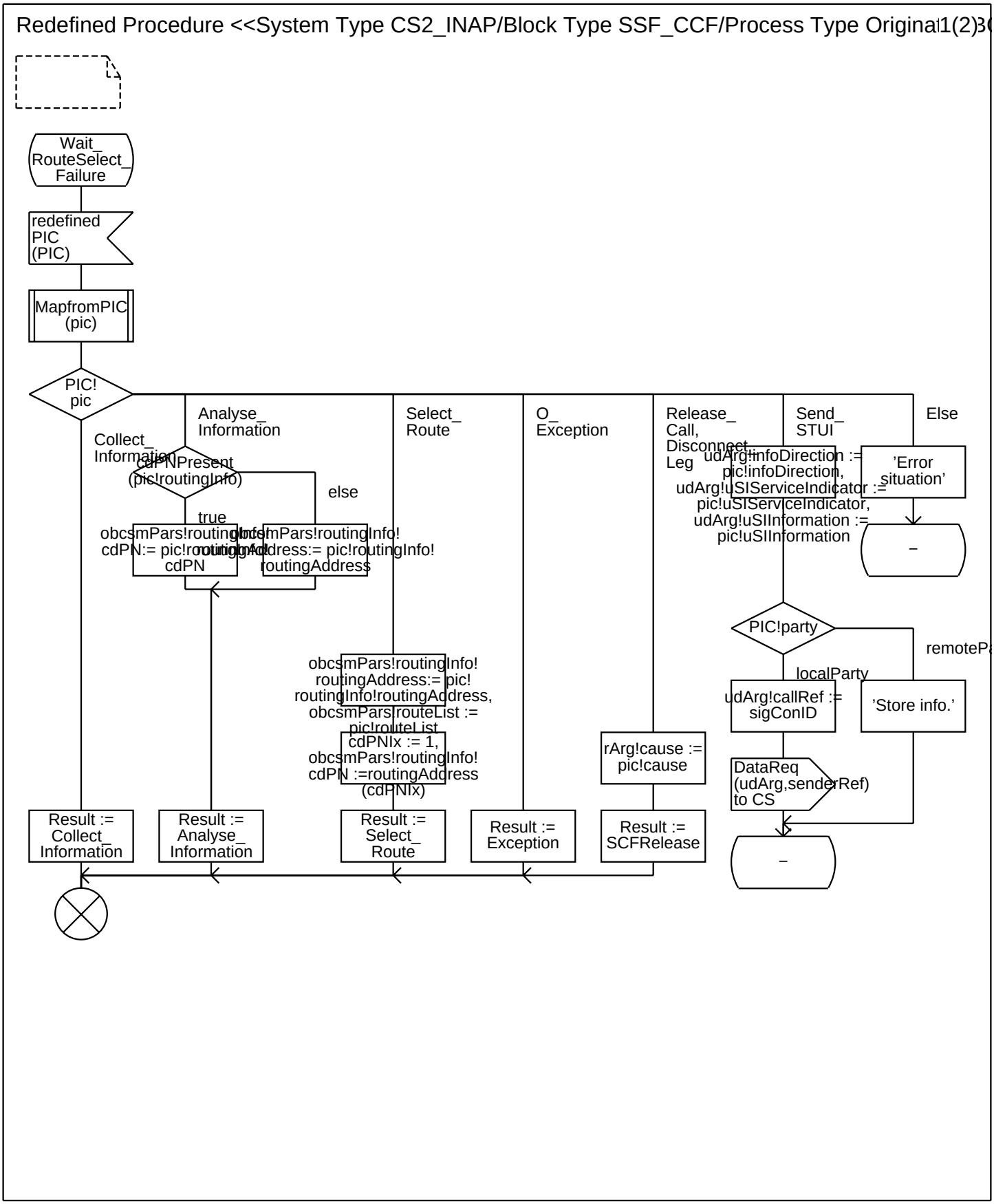
Annex A:


Procedure DP_oReanswer
1(2)
;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
oReanswer
MapToDP
(dp)
DP(DP)
to CS
Wait_
oReAnswer
/* BCSMStop is sent by
the CS/SSF−FSM to terminate
the BCSM. */
PIC_
Resume
Release_
Ind
(RArg)
Release_
ReqInd
(RArg)
BCSMStop
*
nsrArg!callRef
:= sigconId
Result :=
CS_Stop
Network_
ResumeReq
(nsrArg)
to CS
Result :=
Resume
Result :=
ARelease
Result :=
BRelease

Annex A:

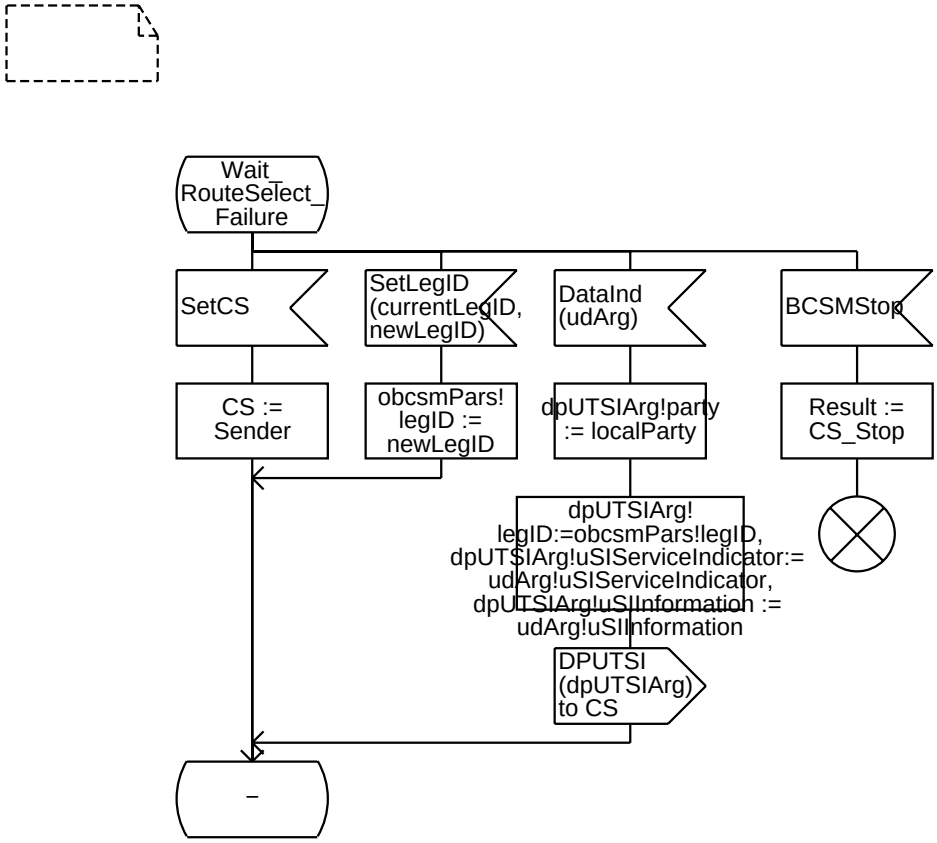

Procedure DP_oReanswer
2(2)
;FPAR
IN/OUT Result DPResultType;
Wait_ oReAnswer
PIC (PIC)
MapfromPIC (pic)
PIC! pic
Release_ Call
Send_ STUI
Else
Disconnect_ Leg
PIC! party
remoteParty
rArg!cause := pic!cause
localParty
Result := DL_A
Result := DL_B
Result := SCFRelease
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
'Error situation'
–
PIC!party
remoteParty
localParty
udArg!callRef := O2I(First(obcsmPars!legID))
udArg!callRef := sigConID
DataReq_ Ind(UDArg,, senderRef) to CS
DataReq (UDArg, senderRef) to CS
–
Wait_ oReAnswer
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
DataReqInd (UDArg)
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
dpUTSIArg!party := remoteParty
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
–

Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(2)3
Wait_ RouteSelect_ Failure
redefined PIC (PIC)
MapfromPIC (pic)
PIC! pic
Collect_ Information
Analyse_ Information
Select_ Route
O_ Exception
Release_ Call, Disconnect_ Leg
Send_ STUI
Else
cdPNPresent (pic!routingInfo)
else
true
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
obcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
'Error situation'
–
PIC!party
remoteP
localParty
udArg!callRef := sigConID
'Store info.'
DataReq (udArg,senderRef) to CS
–
obcsmPars!routingInfo! routingAddress:= pic! routingInfo!routingAddress, obcsmPars!routeList := pic!routeList
cdPNIx := 1, obcsmPars!routingInfo! cdPN :=routingAddress (cdPNIx)
rArg!cause := pic!cause
Result := Collect_ Information
Result := Analyse_ Information
Result := Select_ Route
Result := Exception
Result := SCFRelease

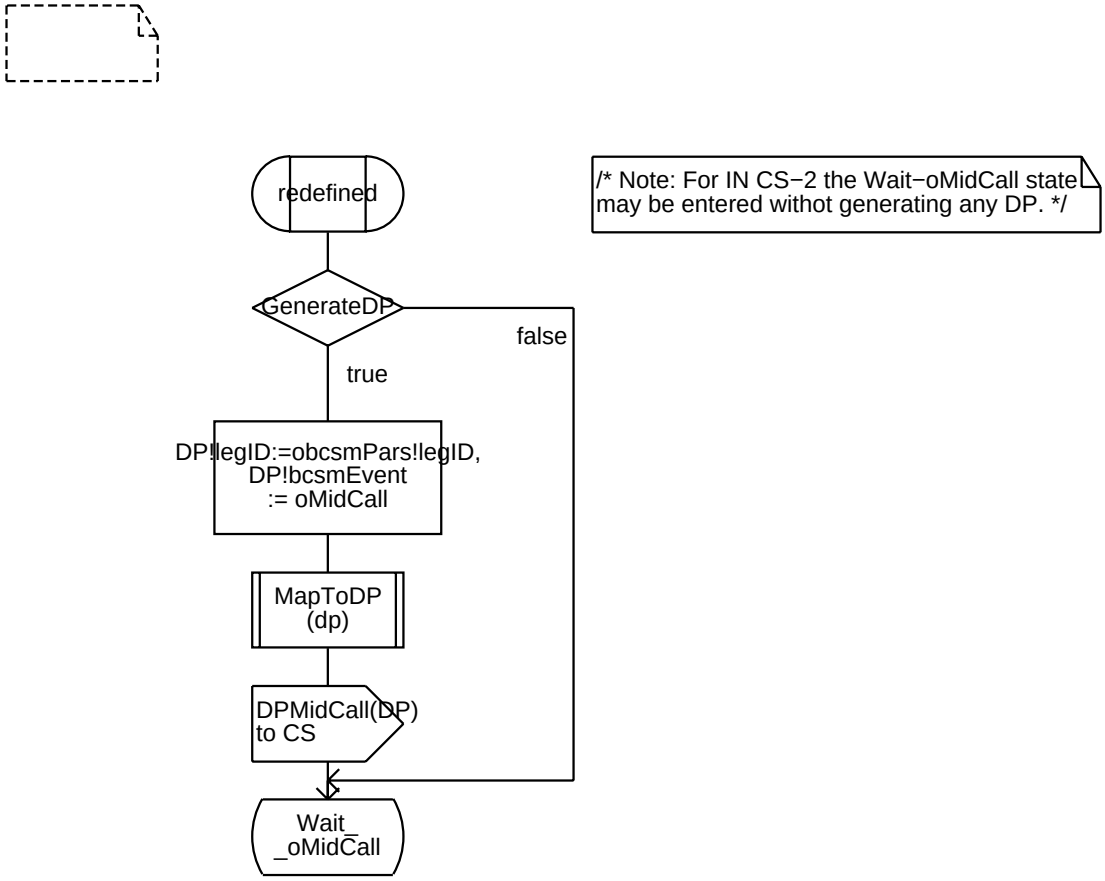
Annex A:

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(2)3

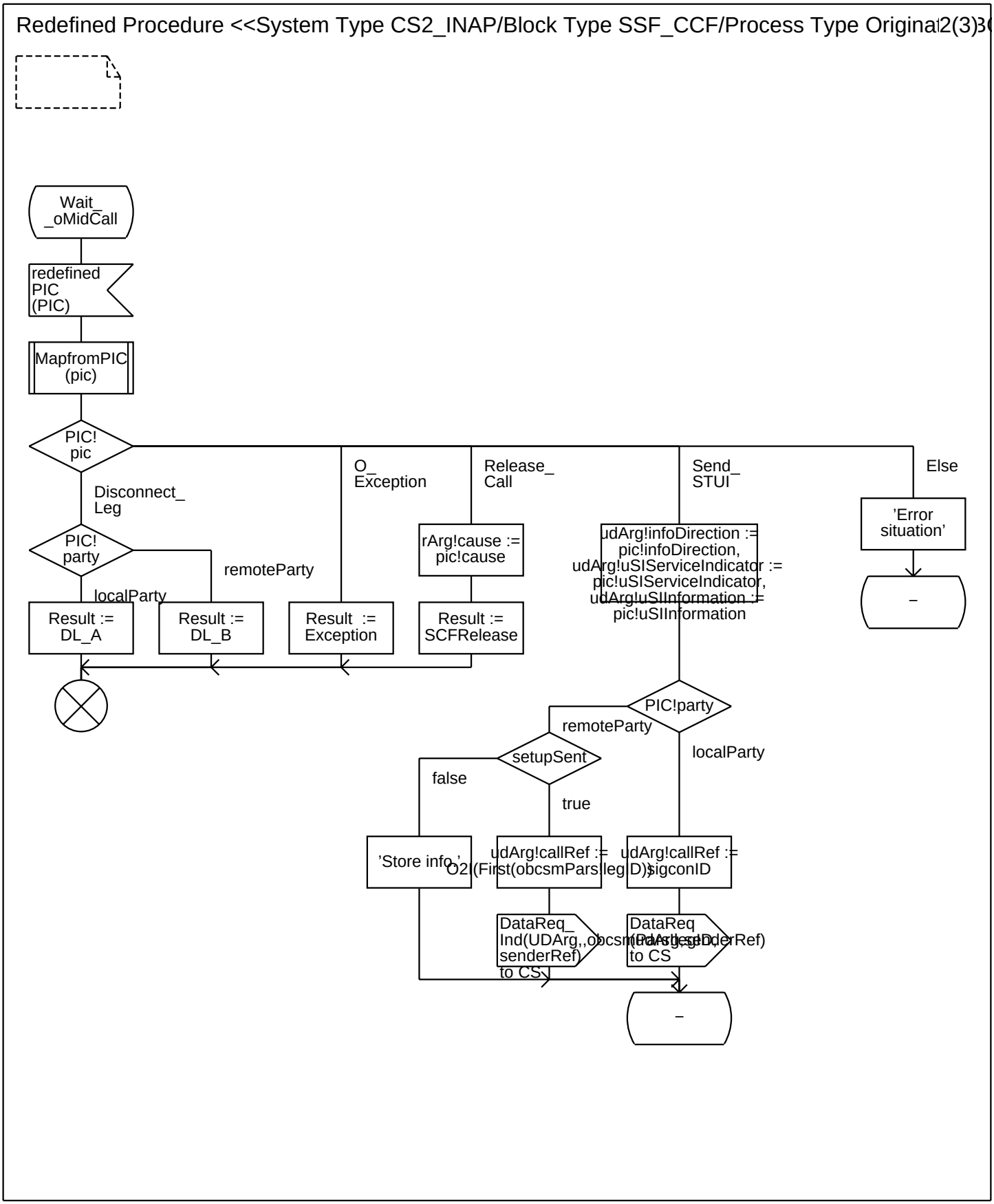


Annex A:

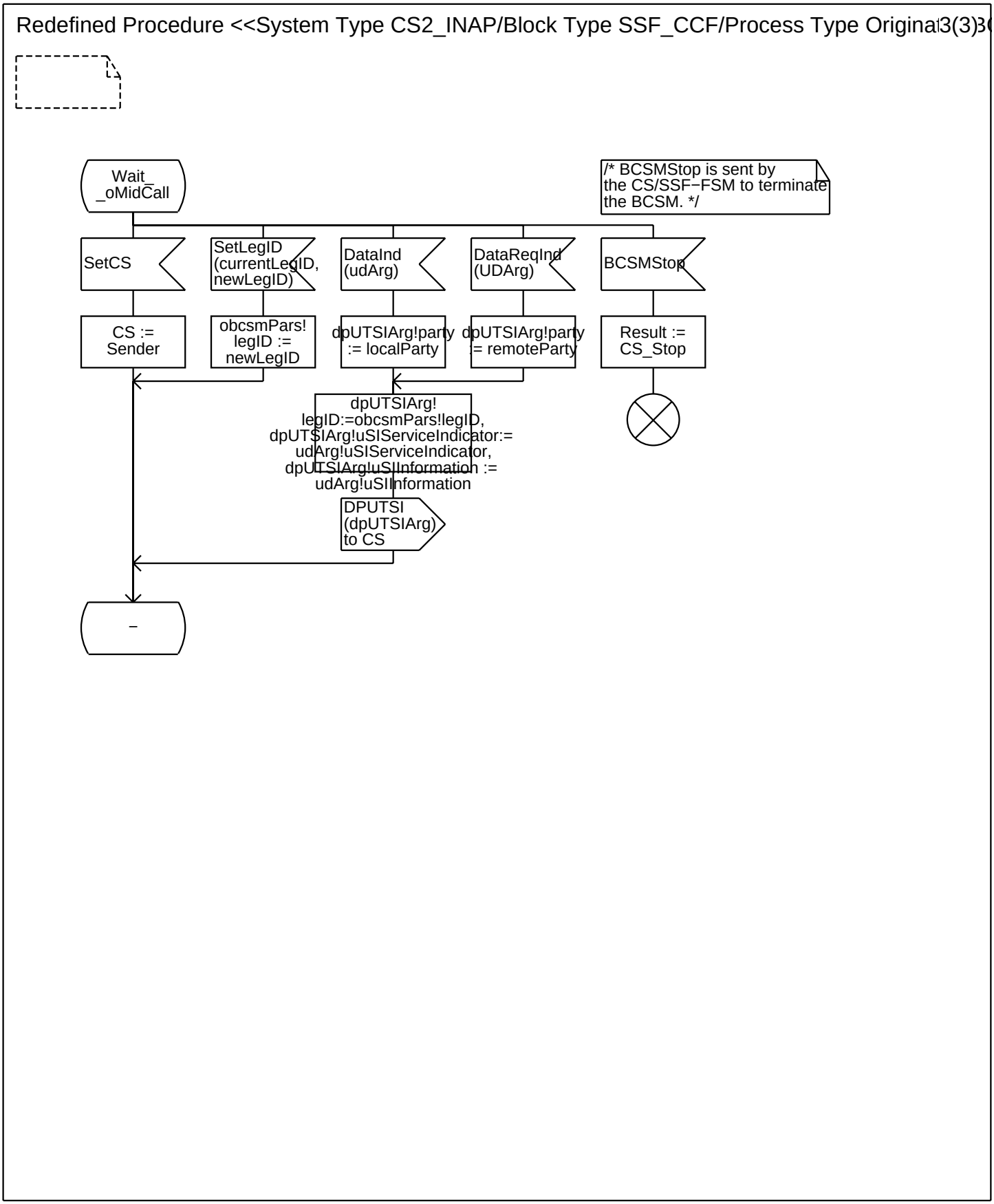
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 1(3)3



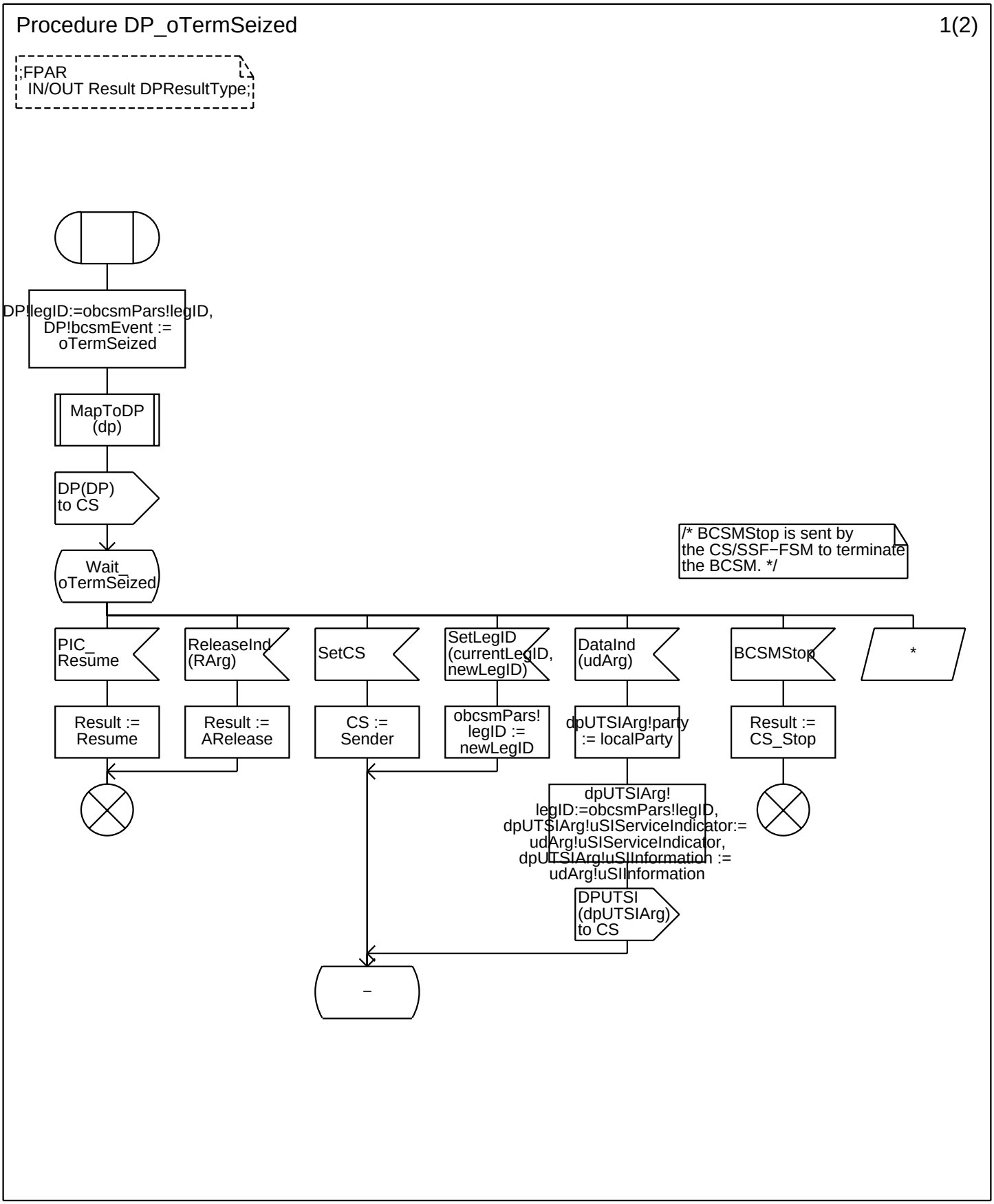
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 2(3)
Wait_
_oMidCall
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Disconnect_
Leg
O_
Exception
Release_
Call
Send_
STUI
Else
'Error
situation'
PIC!
party
remoteParty
localParty
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
Result :=
DL_A
Result :=
DL_B
Result :=
Exception
Result :=
SCFRelease
PIC!party
remoteParty
localParty
setupSent
false
true
'Store info'
udArg!callRef :=
O2(First(obcsmParsleg
udArg!callRef :=
D)); sigconID
DataReq_
Ind(UDArg,,obcsmParsleg,
senderRef)
to CS
DataReq
(UDArg,sigconID,
senderRef)
to CS

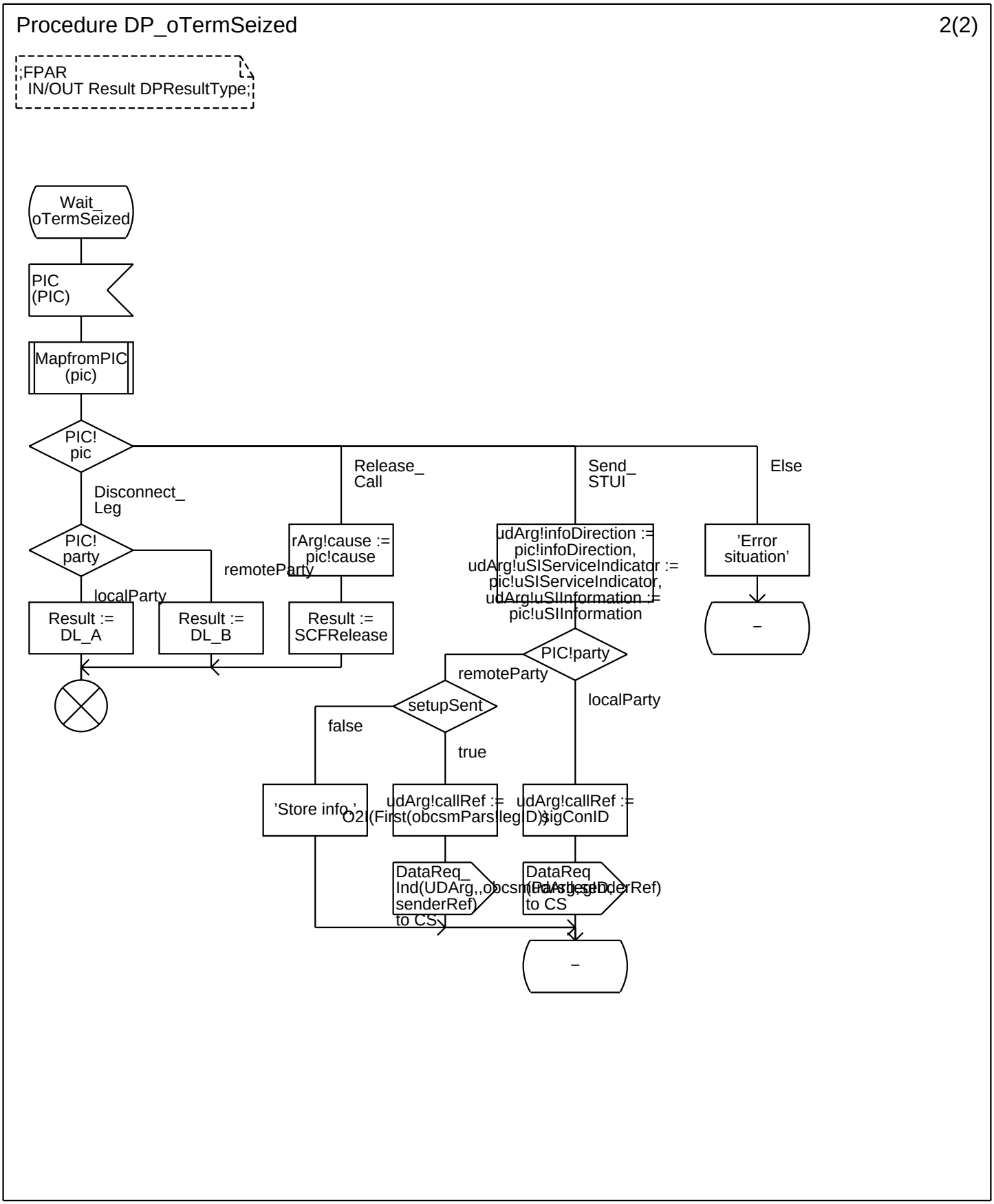
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Original 3(3)
Wait_
_oMidCall
/* BCSMStop is sent by
the CS/SSF−FSM to terminate
the BCSM. */
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
DataReqInd
(UDArg)
BCSMStop
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!party
:= remoteParty
Result :=
CS_Stop
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:


Procedure DP_oTermSeized
1(2)
;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
oTermSeized
MapToDP
(dp)
DP(DP)
to CS
Wait_
oTermSeized
/* BCSMStop is sent by
the CS/SSF−FSM to terminate
the BCSM. */
PIC_
Resume
ReleaseInd
(RArg)
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
BCSMStop
*
Result :=
Resume
Result :=
ARelease
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
Result :=
CS_Stop
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:


Procedure DP_oTermSeized
2(2)
;FPAR
IN/OUT Result DPResultType;
Wait_
oTermSeized
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Disconnect_
Leg
Release_
Call
Send_
STUI
Else
PIC!
party
remoteParty
localParty
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
DL_A
Result :=
DL_B
Result :=
SCFRelease
–
PIC!party
remoteParty
localParty
setupSent
false
true
'Store info.'
udArg!callRef :=
O2I(First(obcsmParsLegID))
udArg!callRef :=
sigConID
DataReq_
Ind(UDArg,,obcsmParsLegID,
senderRef)
to CS
DataReq
(UDArg,,sigConID,
senderRef)
to CS
–

Annex A:

Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE1(9)M

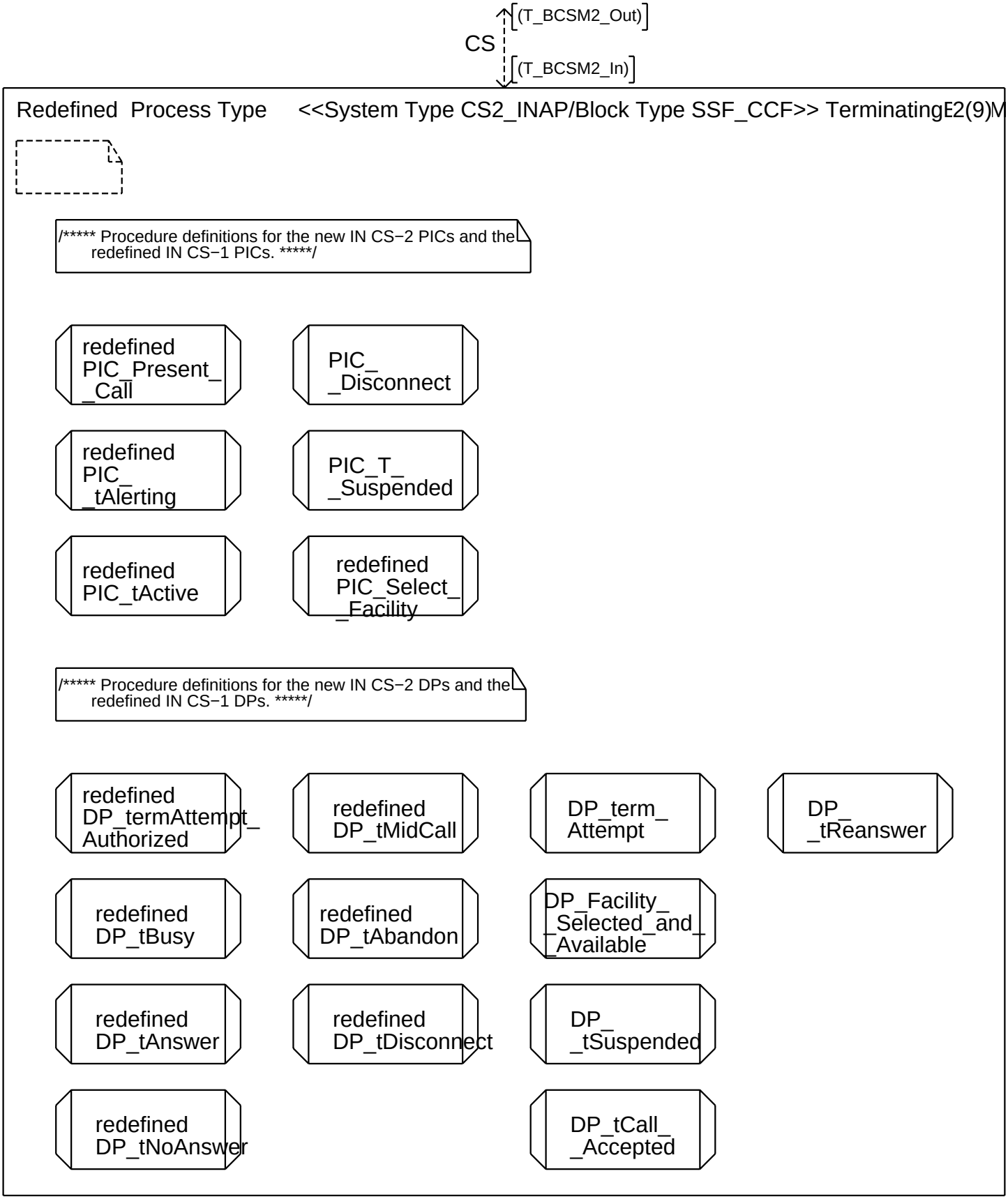
```
/***** T−BCSM FOR CORE INAP CS−2. *****/
```

```
/* Note: The T−BCSM for IN CS−2 only shows the extentions required to the
IN CS−1 T−BCSM. Therefore, to understand the complete BCSM behaviour it is
necessary to read the two BCSM descriptions together. */
```

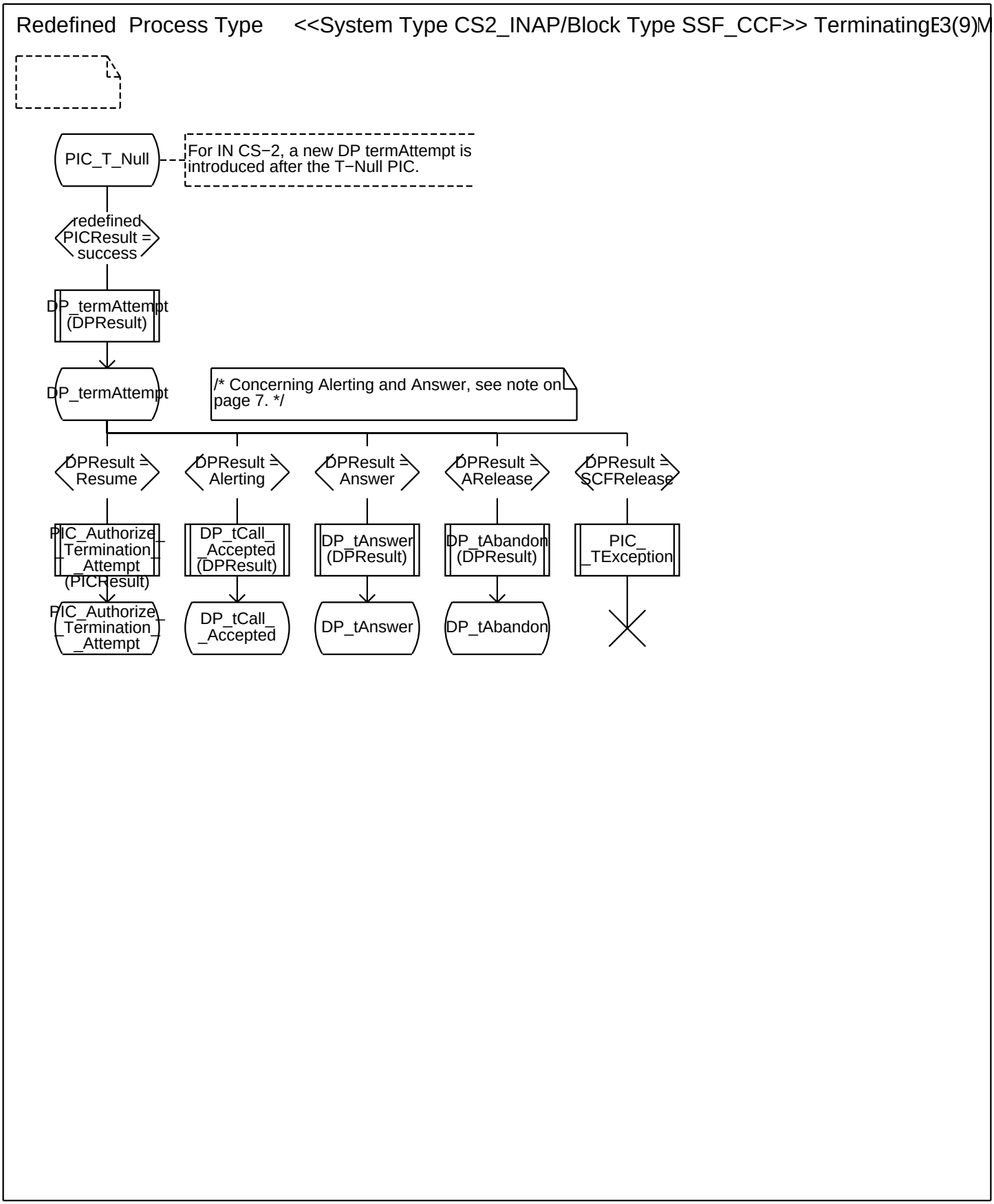
```
/***** VARIABLE AND TIMER DECLARATIONS. *****/
```

```
DCL
  currentLegID, newLegID LegType,
  udArg UserDataType,
  dpUTSIArg DPUTSIArg,
  nsrArg NetworkSRType,

  /* Used to indicate the status of the call processing. */
  setupSent boolean := false;    /* Indicates if SetupReq is sent to the called party */
```

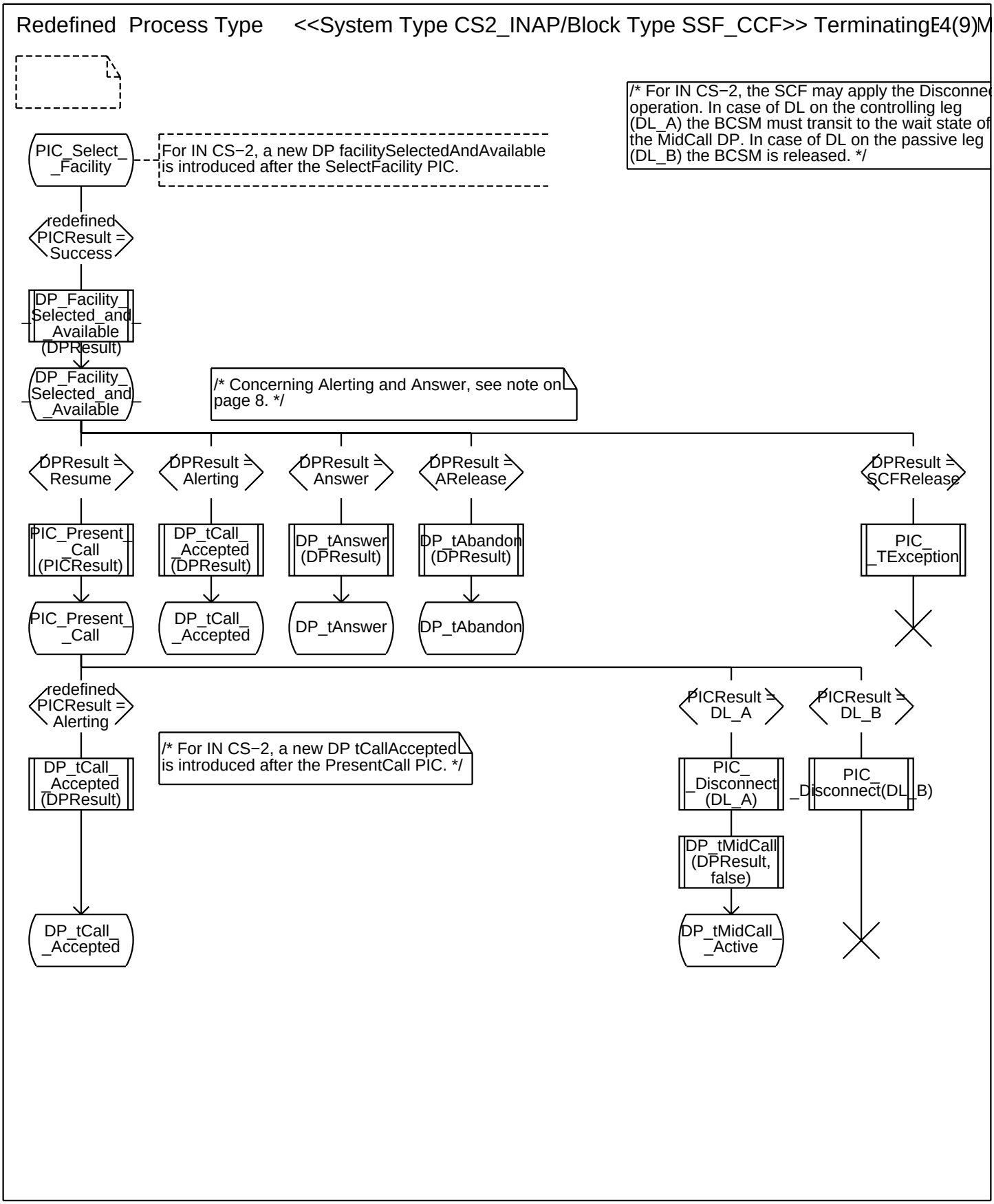
Annex A:


CS
(T_BCSM2_Out)
(T_BCSM2_In)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE2(9)M
/***** Procedure definitions for the new IN CS−2 PICs and the redefined IN CS−1 PICs. *****/
redefined PIC_Present_ _Call
PIC_ _Disconnect
redefined PIC_ _tAlerting
PIC_T_ _Suspended
redefined PIC_tActive
redefined PIC_Select_ _Facility
/***** Procedure definitions for the new IN CS−2 DPs and the redefined IN CS−1 DPs. *****/
redefined DP_termAttempt_ Authorized
redefined DP_tMidCall
DP_term_ Attempt
DP_ _tReanswer
redefined DP_tBusy
redefined DP_tAbandon
DP_Facility_ _Selected_and_ _Available
redefined DP_tAnswer
redefined DP_tDisconnect
DP_ _tSuspended
redefined DP_tNoAnswer
DP_tCall_ _Accepted

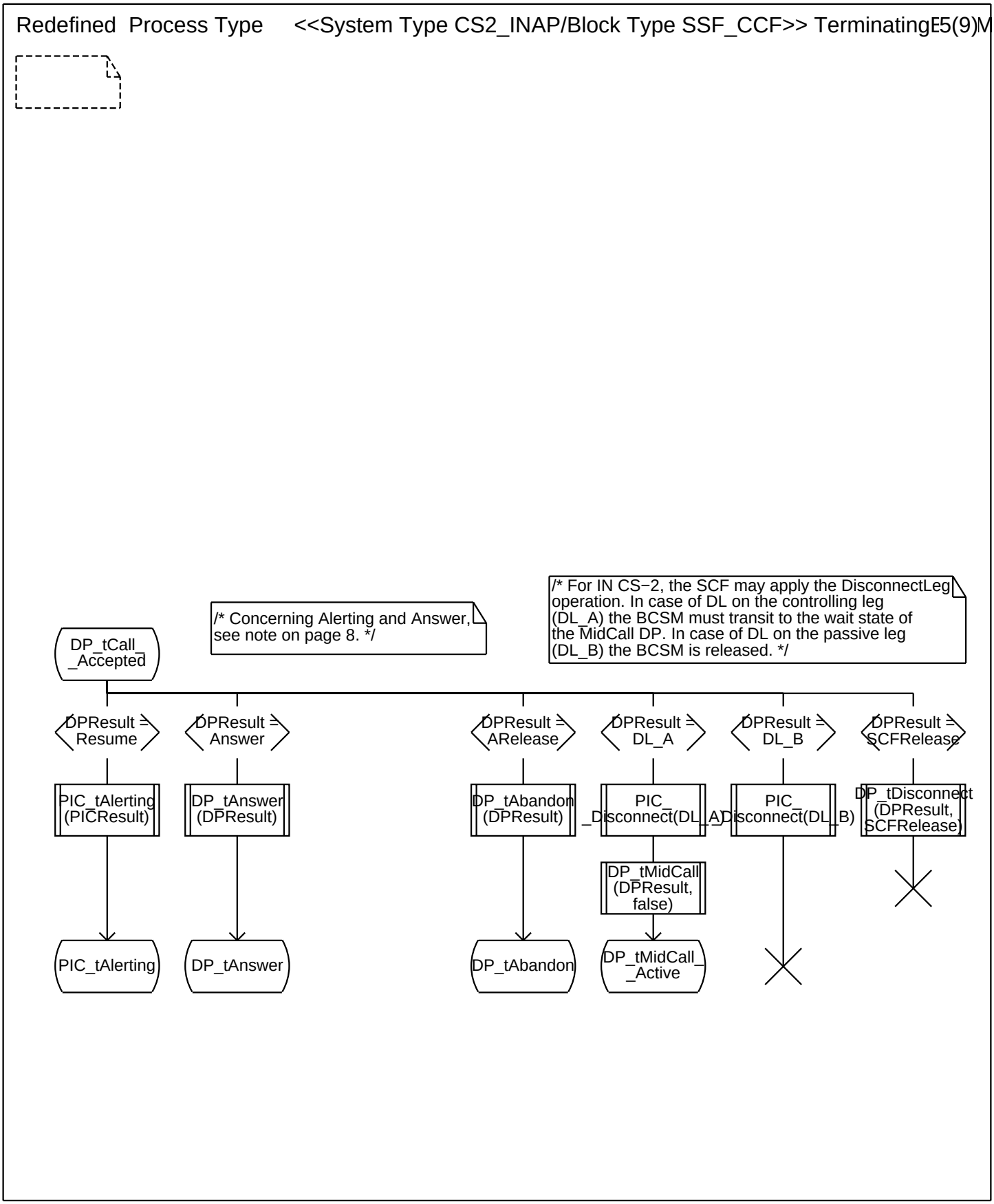
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE3(9)M
PIC_T_Null
For IN CS−2, a new DP termAttempt is introduced after the T−Null PIC.
redefined PICResult = success
DP_termAttempt (DPResult)
DP_termAttempt
/* Concerning Alerting and Answer, see note on page 7. */
DPResult = Resume
DPResult = Alerting
DPResult = Answer
DPResult = ARelease
DPResult = SCFRelease
PIC_Authorize_Termination_Attempt (PICResult)
DP_tCall_Accepted (DPResult)
DP_tAnswer (DPResult)
DP_tAbandon (DPResult)
PIC_TException
PIC_Authorize_Termination_Attempt
DP_tCall_Accepted
DP_tAnswer
DP_tAbandon

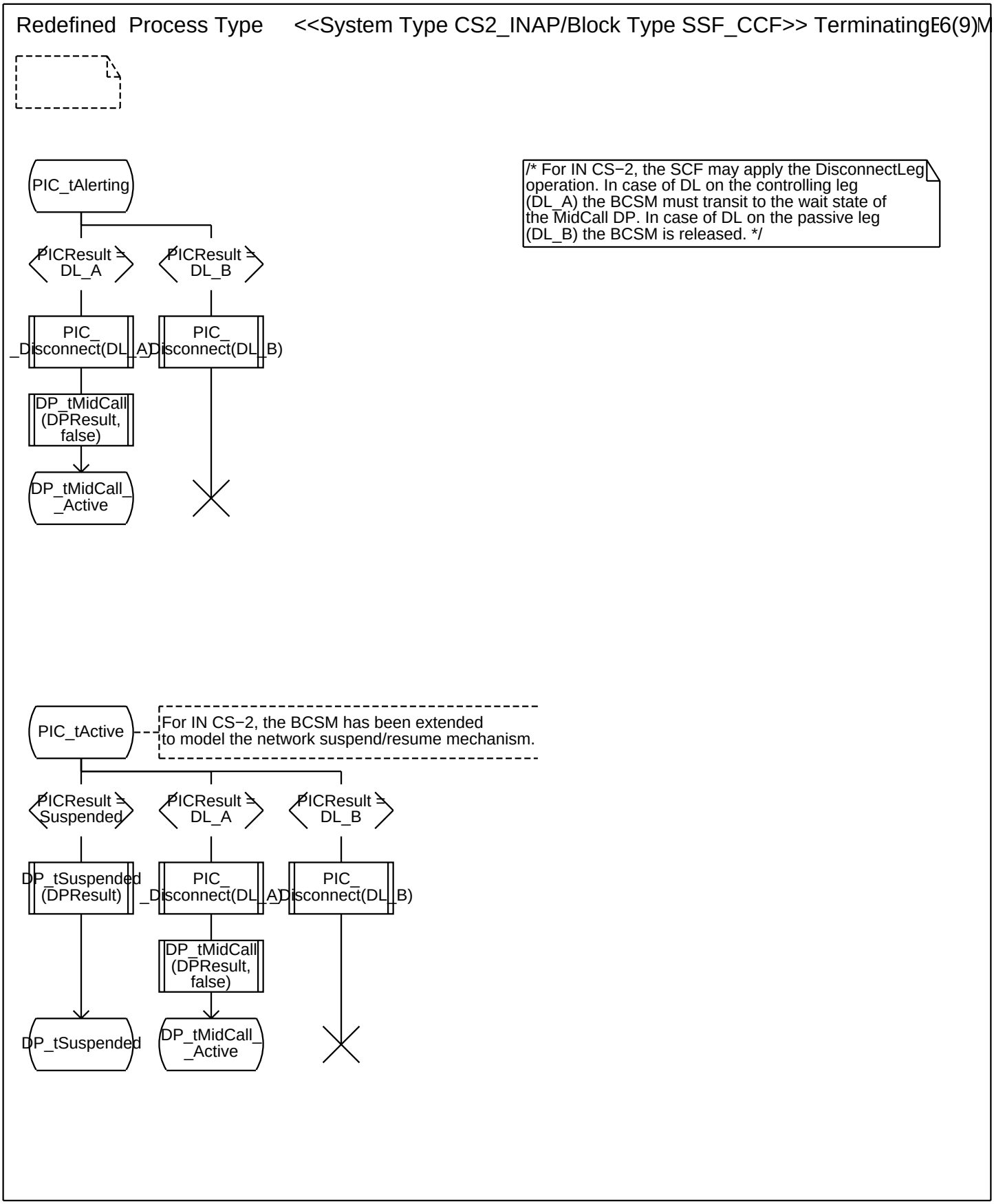
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE4(9)M
/* For IN CS−2, the SCF may apply the Disconne operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
PIC_Select_ _Facility
For IN CS−2, a new DP facilitySelectedAndAvailable is introduced after the SelectFacility PIC.
redefined PICResult = Success
DP_Facility_ _Selected_and_ _Available (DPResult)
DP_Facility_ _Selected_and_ _Available
/* Concerning Alerting and Answer, see note on page 8. */
DPResult = Resume
DPResult = Alerting
DPResult = Answer
DPResult = ARelease
DPResult = SCFRelease
PIC_Present_ _Call (PICResult)
DP_tCall_ _Accepted (DPResult)
DP_tAnswer (DPResult)
DP_tAbandon (DPResult)
PIC_ _TException
PIC_Present_ _Call
DP_tCall_ _Accepted
DP_tAnswer
DP_tAbandon
redefined PICResult = Alerting
PICResult = DL_A
PICResult = DL_B
/* For IN CS−2, a new DP tCallAccepted is introduced after the PresentCall PIC. */
DP_tCall_ _Accepted (DPResult)
PIC_ _Disconnect (DL_A)
PIC_ _Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tCall_ _Accepted
DP_tMidCall_ _Active

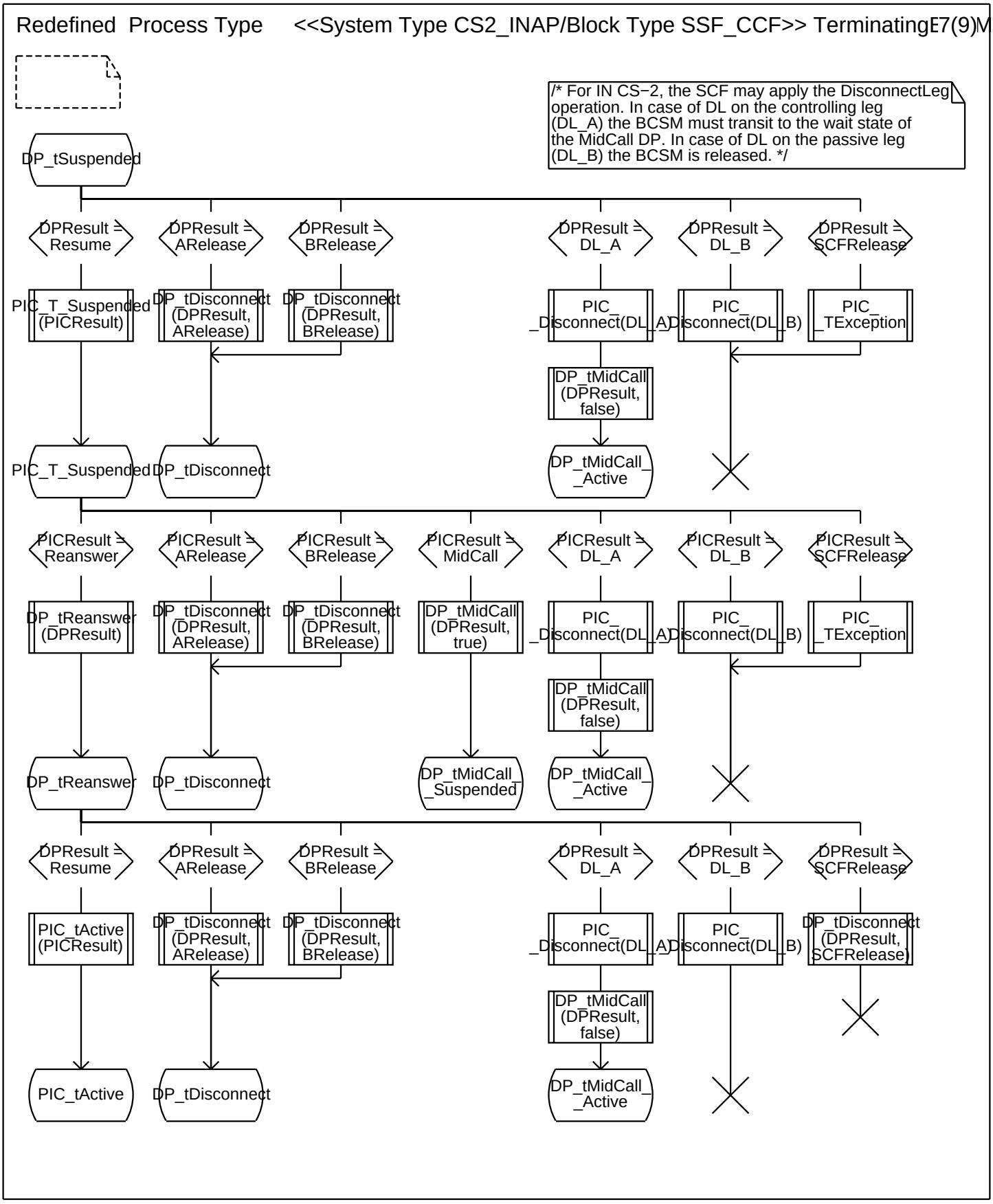
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE5(9)M
/* Concerning Alerting and Answer, see note on page 8. */
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_tCall_ _Accepted
DPResult = Resume
DPResult = Answer
DPResult = ARelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_tAlerting (PICResult)
DP_tAnswer (DPResult)
DP_tAbandon (DPResult)
PIC_ _Disconnect(DL_A)
PIC_ Disconnect(DL_B)
DP_tDisconnect (DPResult, SCFRelease)
DP_tMidCall (DPResult, false)
PIC_tAlerting
DP_tAnswer
DP_tAbandon
DP_tMidCall_ _Active

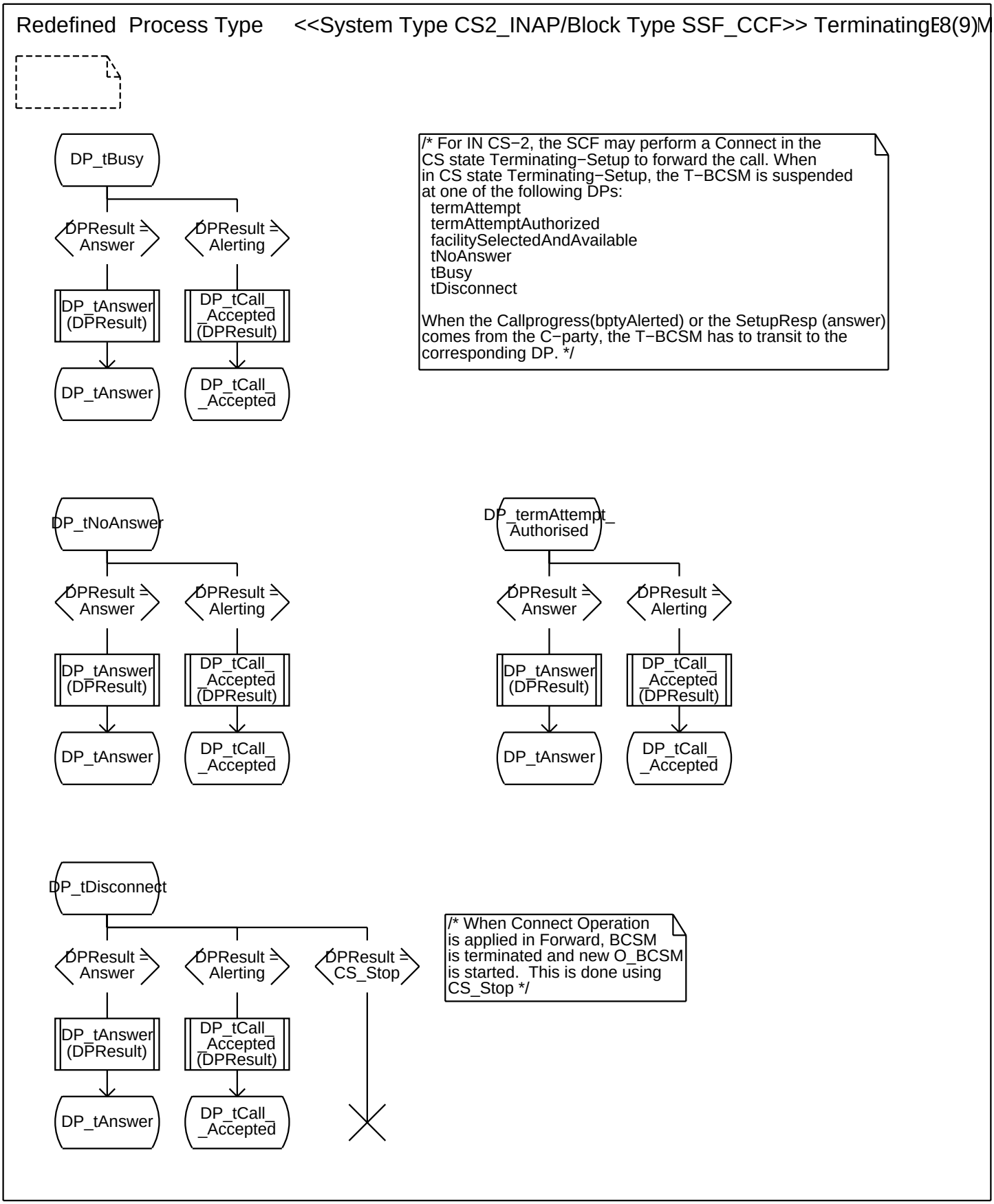
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE6(9)M
PIC_tAlerting
PICResult = DL_A
PICResult = DL_B
PIC_ _Disconnect(DL_A)
PIC_ Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tMidCall_ _Active
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
PIC_tActive
For IN CS−2, the BCSM has been extended to model the network suspend/resume mechanism.
PICResult = Suspended
PICResult = DL_A
PICResult = DL_B
DP_tSuspended (DPResult)
PIC_ _Disconnect(DL_A)
PIC_ Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tSuspended
DP_tMidCall_ _Active

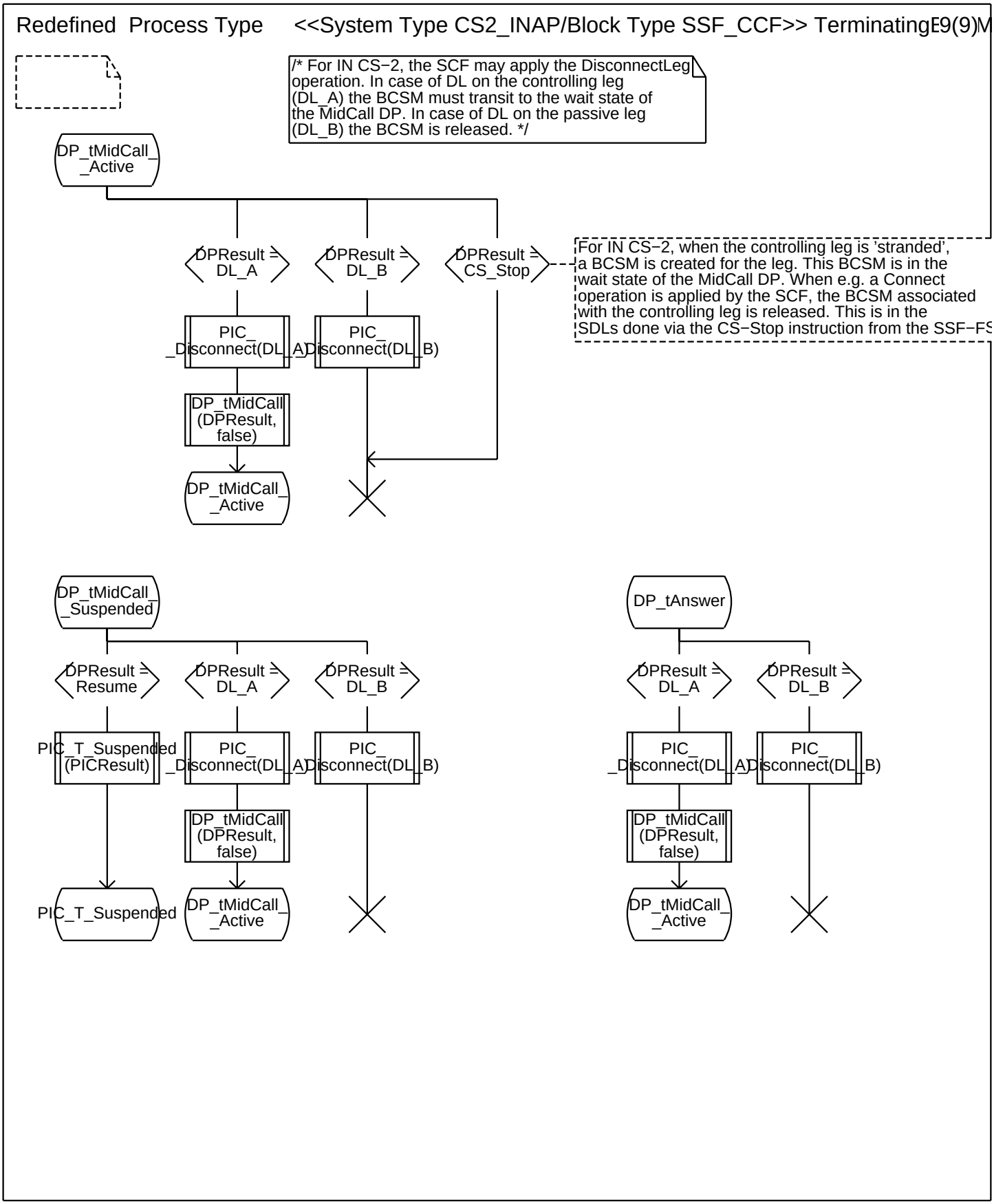
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE7(9)M
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_tSuspended
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_T_Suspended (PICResult)
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
PIC_ _TException
DP_tMidCall (DPResult, false)
PIC_T_Suspended
DP_tDisconnect
DP_tMidCall_ _Active
PICResult = Reanswer
PICResult = ARelease
PICResult = BRelease
PICResult = MidCall
PICResult = DL_A
PICResult = DL_B
PICResult = SCFRelease
DP_tReanswer (DPResult)
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
DP_tMidCall (DPResult, true)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
PIC_ _TException
DP_tMidCall (DPResult, false)
DP_tReanswer
DP_tDisconnect
DP_tMidCall_ _Suspended
DP_tMidCall_ _Active
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_tActive (PICResult)
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
DP_tDisconnect (DPResult, SCFRelease)
DP_tMidCall (DPResult, false)
PIC_tActive
DP_tDisconnect
DP_tMidCall_ _Active

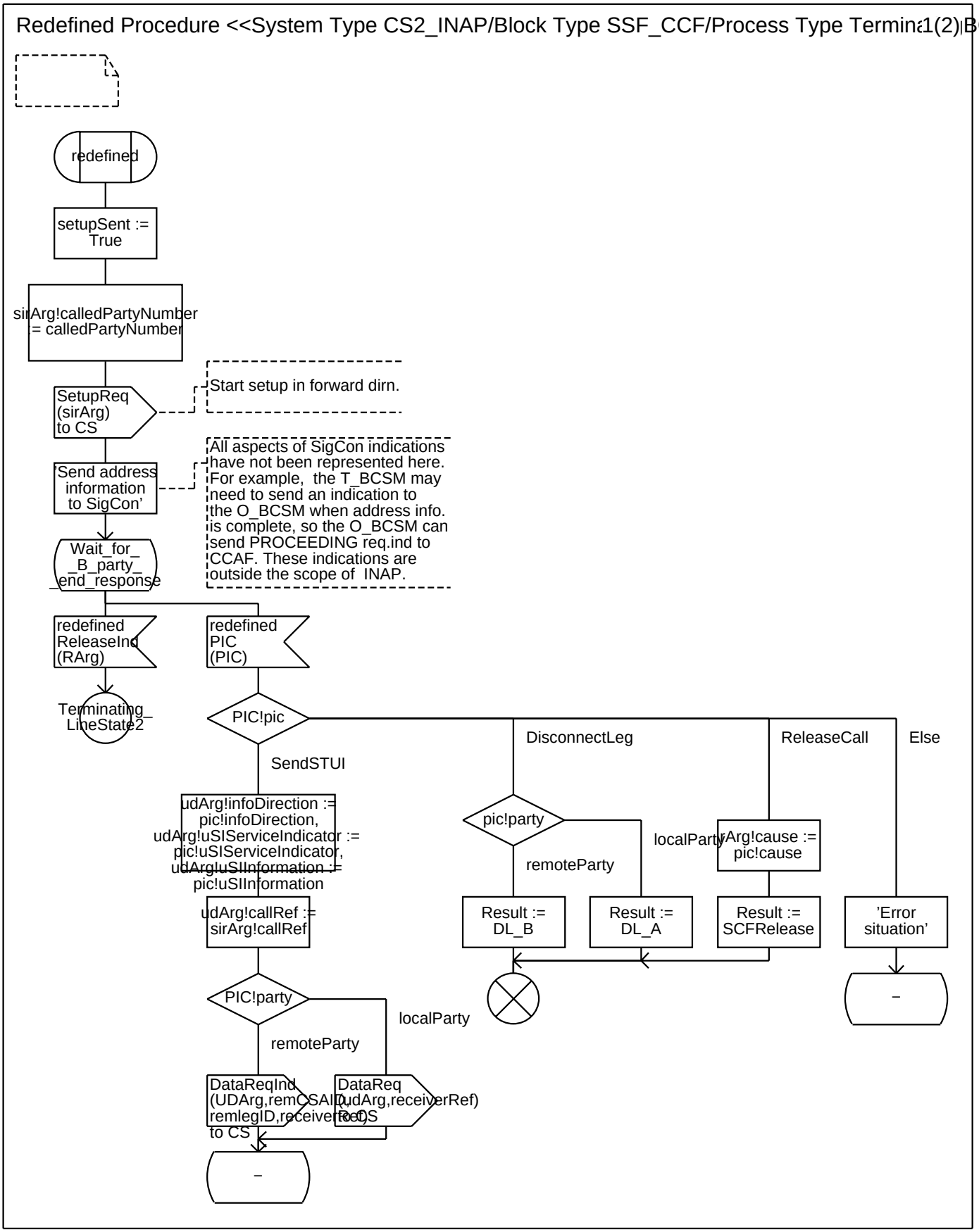
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE8(9)M
DP_tBusy
DPResult = Answer
DPResult = Alerting
DP_tAnswer (DPResult)
DP_tCall_ _Accepted (DPResult)
DP_tAnswer
DP_tCall_ _Accepted
/* For IN CS−2, the SCF may perform a Connect in the CS state Terminating−Setup to forward the call. When in CS state Terminating−Setup, the T−BCSM is suspended at one of the following DPs:
termAttempt
termAttemptAuthorized
facilitySelectedAndAvailable
tNoAnswer
tBusy
tDisconnect
When the Callprogress(bptyAlerted) or the SetupResp (answer) comes from the C−party, the T−BCSM has to transit to the corresponding DP. */
DP_tNoAnswer
DP_termAttempt_ Authorised
DP_tDisconnect
DPResult = CS_Stop
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */

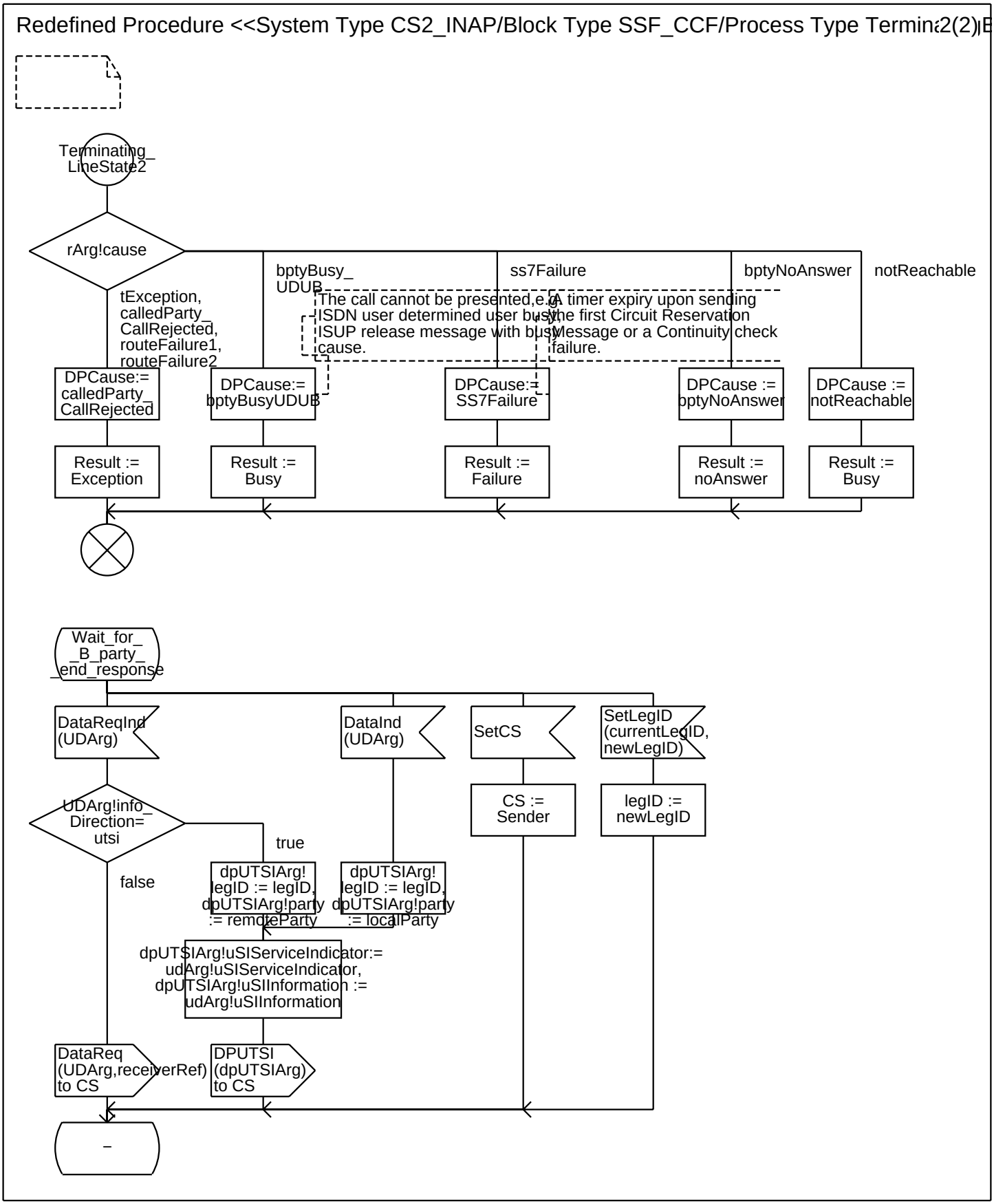
Annex A:


Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingE9(9)M
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_tMidCall_ _Active
DPResult = DL_A
DPResult = DL_B
DPResult = CS_Stop
For IN CS−2, when the controlling leg is 'stranded', a BCSM is created for the leg. This BCSM is in the wait state of the MidCall DP. When e.g. a Connect operation is applied by the SCF, the BCSM associated with the controlling leg is released. This is in the SDLs done via the CS−Stop instruction from the SSF−FS
PIC_ _Disconnect(DL_A)
PIC_ Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tMidCall_ _Active
DP_tMidCall_ _Suspended
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
PIC_T_Suspended (PICResult)
PIC_ _Disconnect(DL_A)
PIC_ Disconnect(DL_B)
DP_tMidCall (DPResult, false)
PIC_T_Suspended
DP_tMidCall_ _Active
DP_tAnswer
DPResult = DL_A
DPResult = DL_B
PIC_ _Disconnect(DL_A)
PIC_ Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tMidCall_ _Active

Annex A:

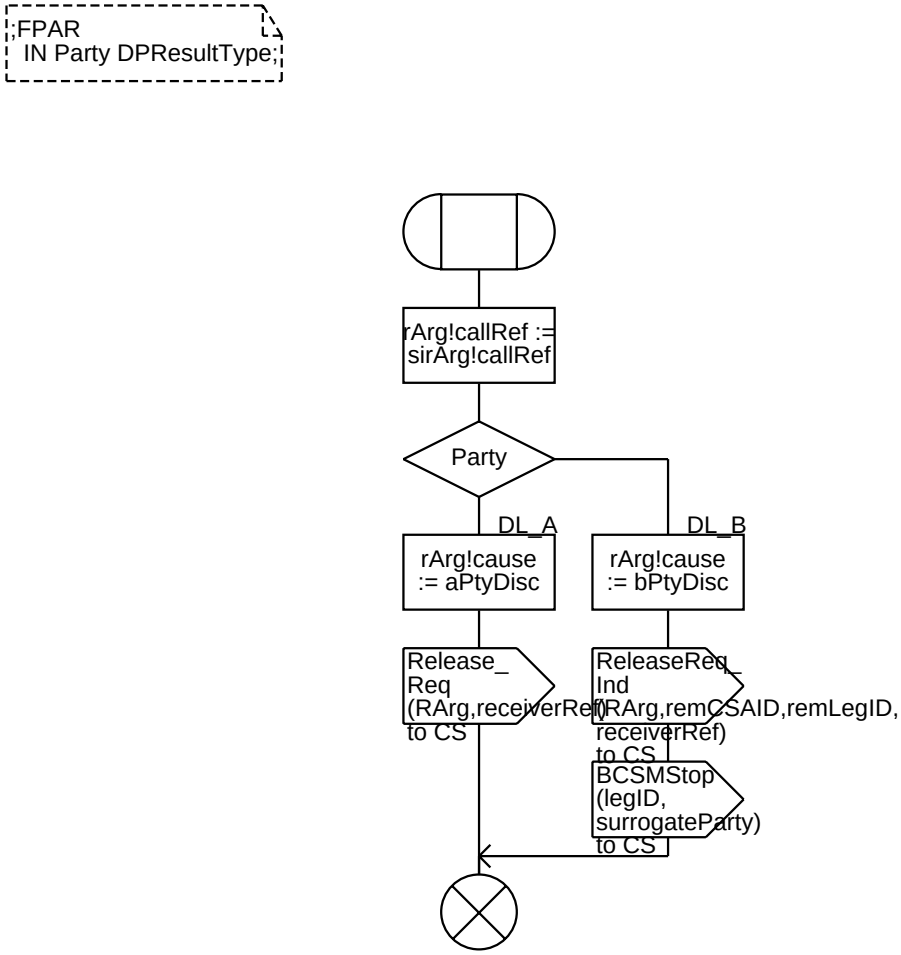

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina 1(2)|BCSM
redefined
setupSent := True
sirArg!calledPartyNumber = calledPartyNumber
SetupReq (sirArg) to CS
Start setup in forward dirn.
Send address information to SigCon'
All aspects of SigCon indications have not been 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
redefined ReleaseInd (RArg)
redefined PIC (PIC)
Terminating_ LineState2
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
udArg!callRef := sirArg!callRef
pic!party
localParty
remoteParty
rArg!cause := pic!cause
Result := DL_B
Result := DL_A
Result := SCFRelease
'Error situation'
PIC!party
localParty
remoteParty
DataReqInd (UDArg,remCSAID, remlegID,receiverRef) to CS
DataReq (UDArg,receiverRef) to CS

Annex A:

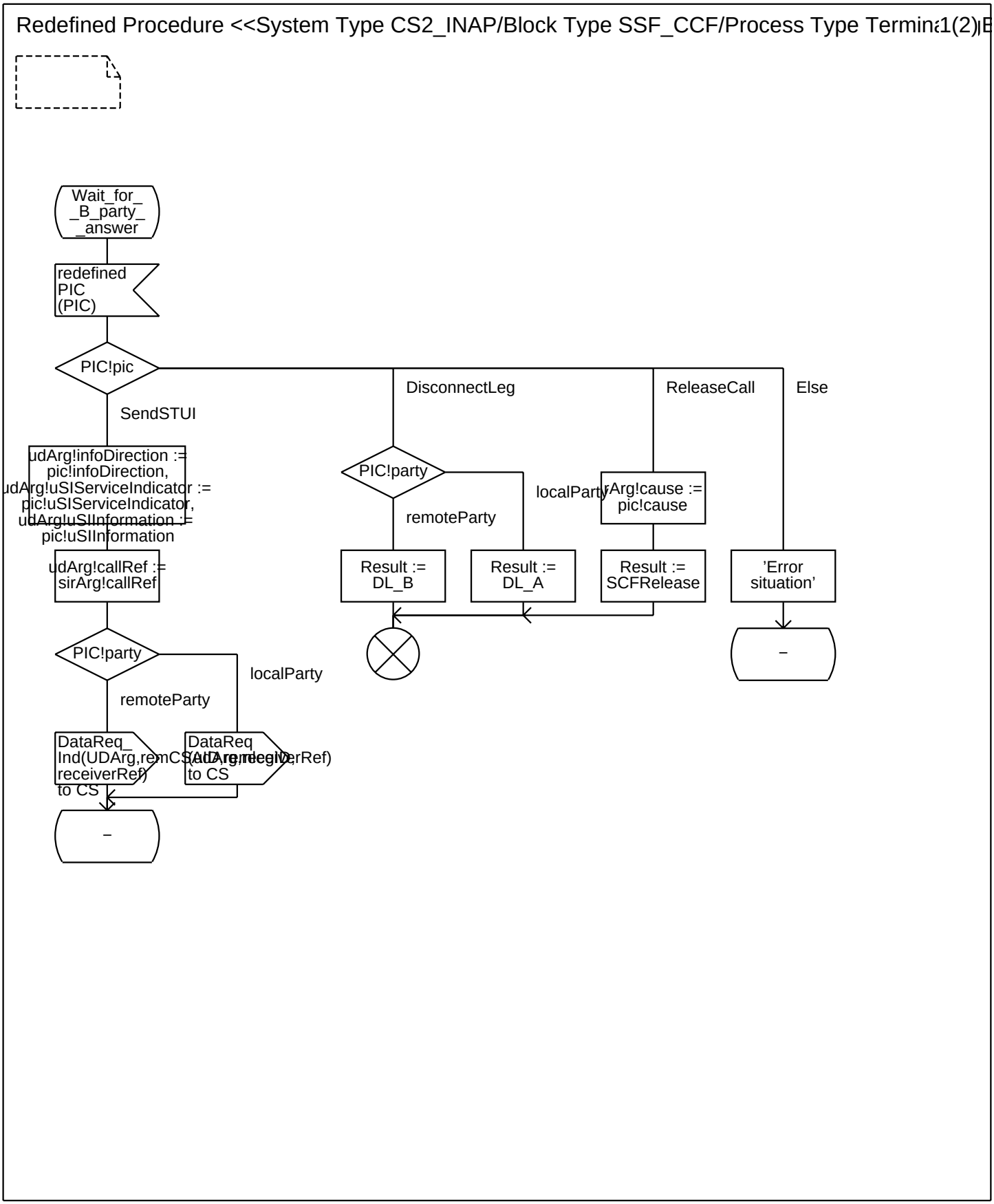

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
2(2)
Terminating_
LineState2
rArg!cause
tException,
calledParty_
CallRejected,
routeFailure1,
routeFailure2
bptyBusy_
UDUB
The call cannot be presented,e.g.
ISDN user determined user busy,
ISUP release message with busy
cause.
ss7Failure
A timer expiry upon sending
the first Circuit Reservation
Message or a Continuity check
failure.
bptyNoAnswer
notReachable
DPCause:=
calledParty_
CallRejected
DPCause:=
bptyBusyUDUB
DPCause:=
SS7Failure
DPCause :=
bptyNoAnswer
DPCause :=
notReachable
Result :=
Exception
Result :=
Busy
Result :=
Failure
Result :=
noAnswer
Result :=
Busy
Wait_for_
_B_party_
_end_response
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
CS :=
Sender
legID :=
newLegID
UDArg!info_
Direction=
utsi
true
false
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:

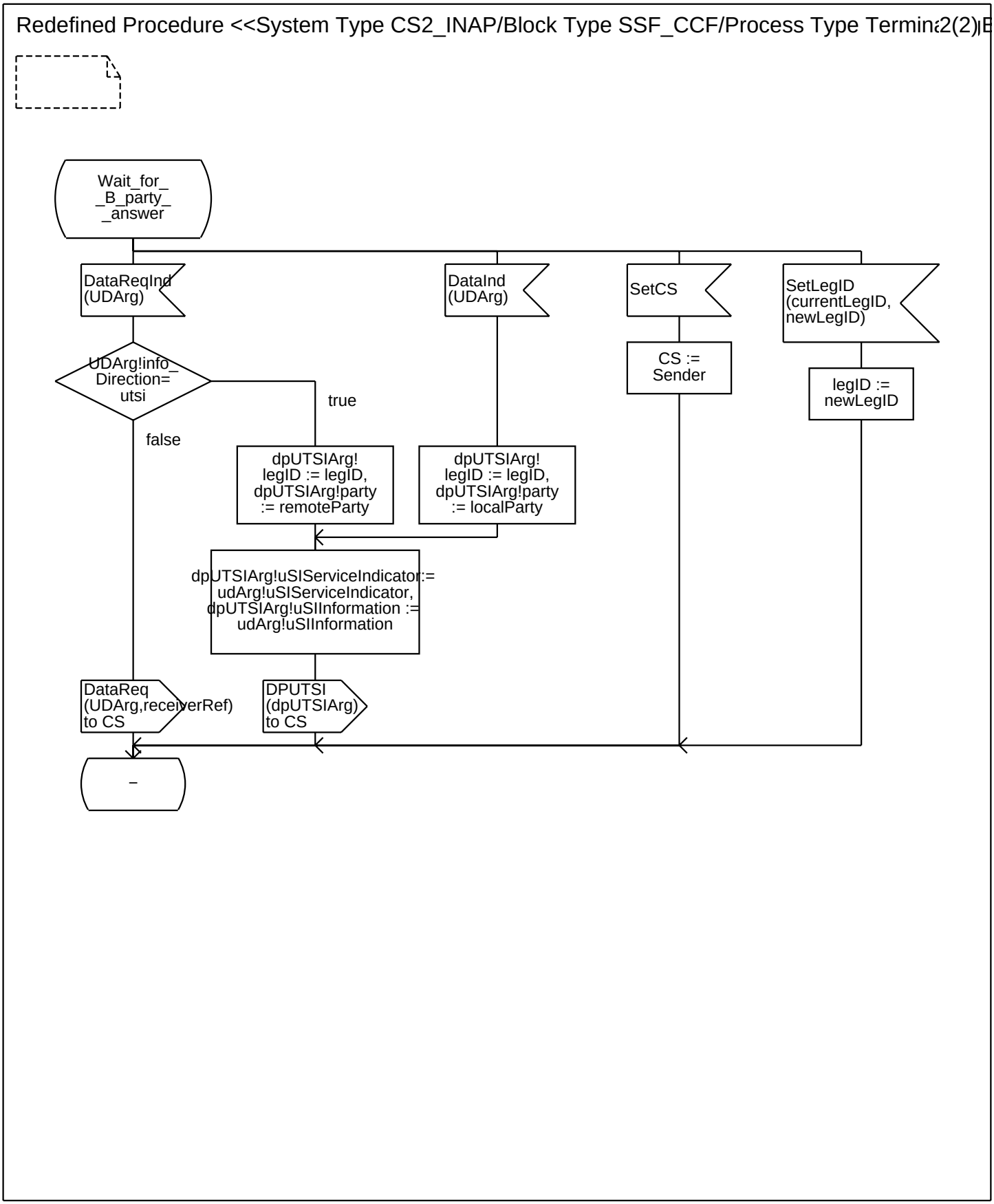
Procedure <<Process Type TerminatingBCSM>> PIC_Disconnect 1(1)



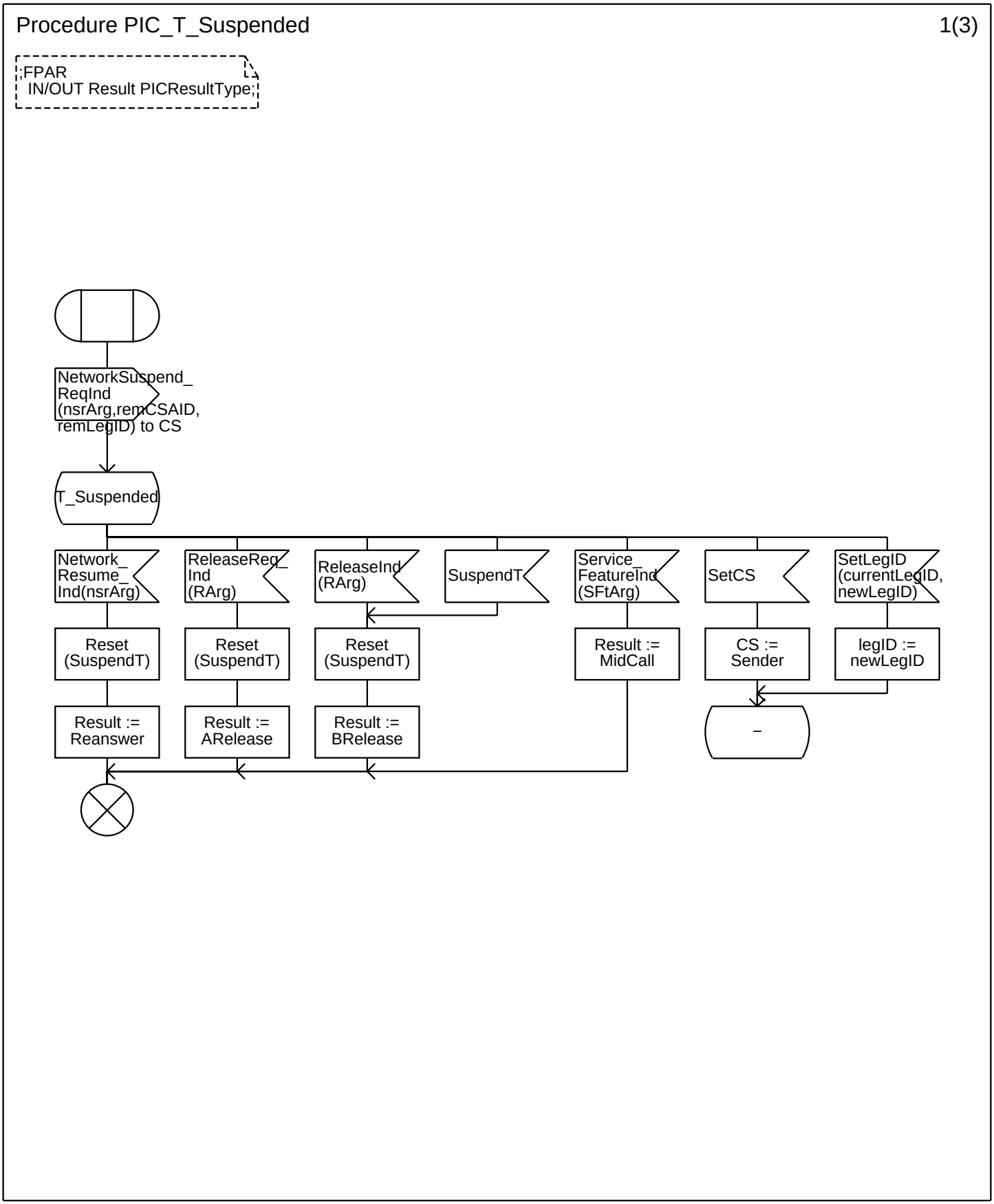
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
1(2)
Wait_for_
_B_party_
_answer
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAddressID,
receiverRef)
to CS
DataReq
(UDArg,remCSAddressID,
receiverRef)
to CS
–
PIC!party
localParty
remoteParty
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
Result :=
SCFRelease
'Error
situation'
–

Annex A:

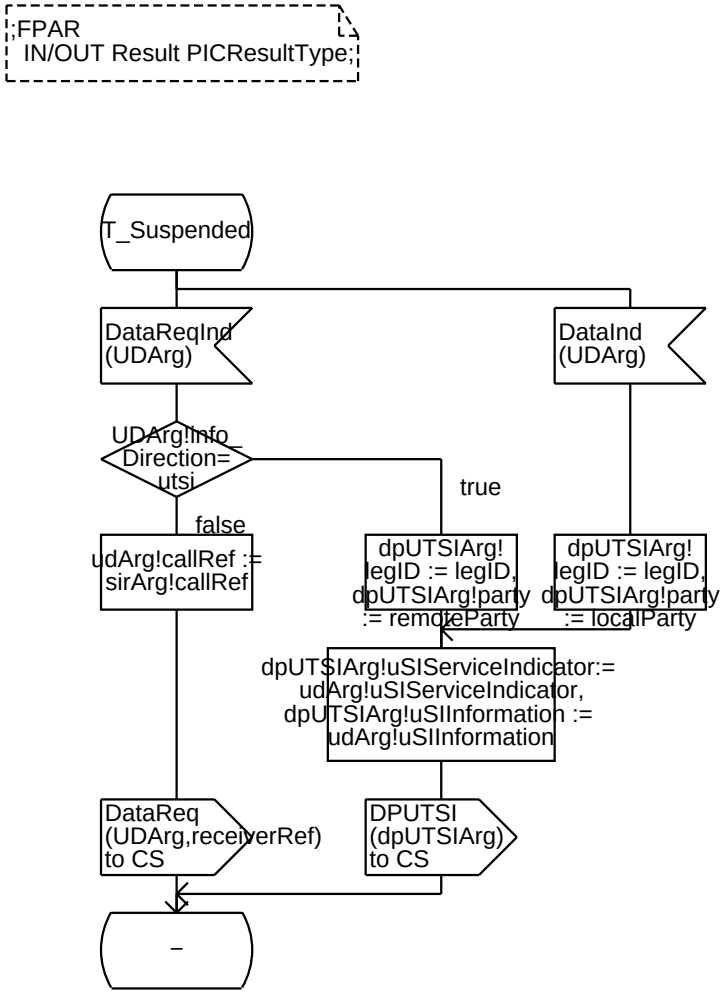

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina2(2)
Wait_for_
_B_party_
_answer
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator :=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:

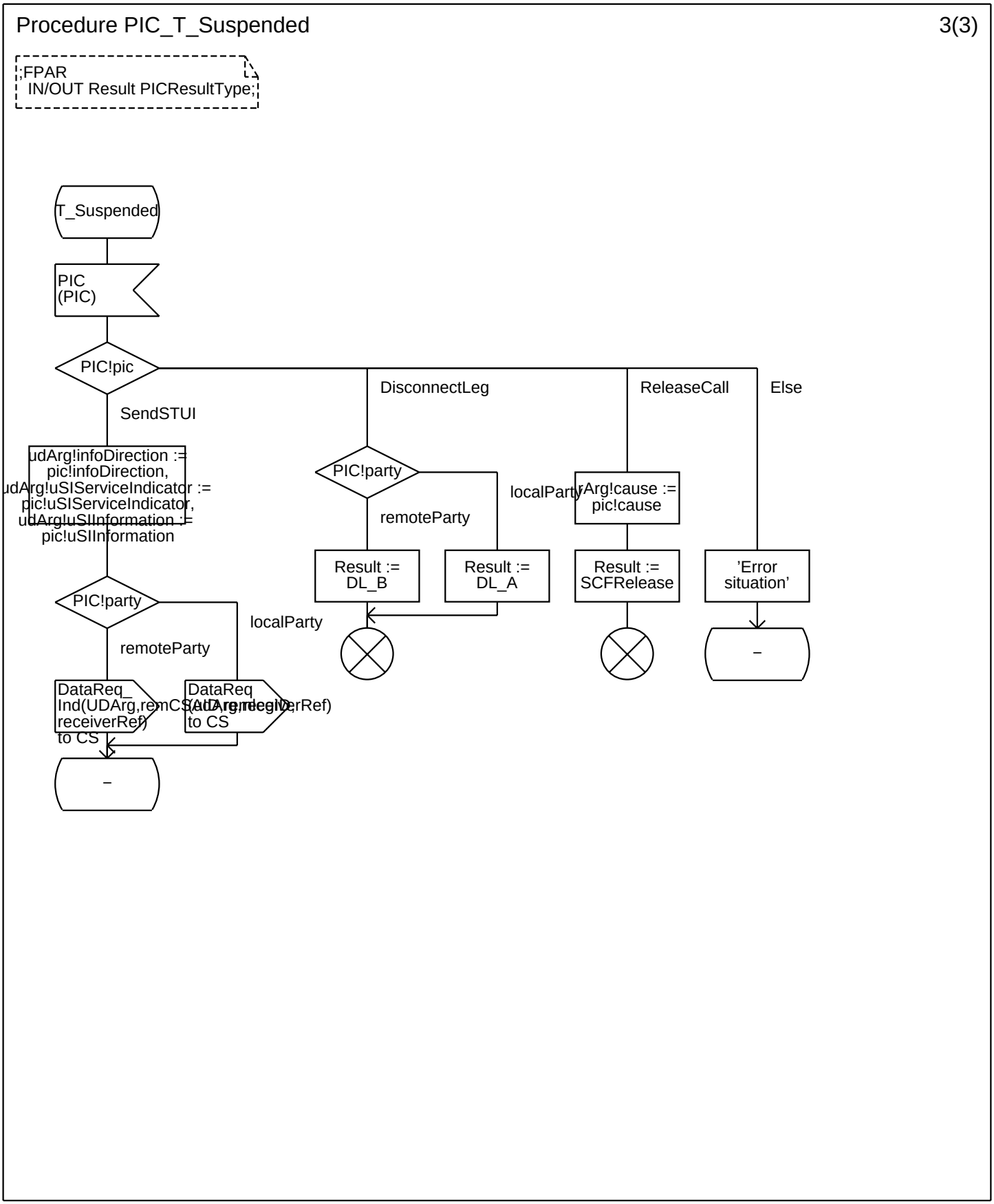

Procedure PIC_T_Suspended
1(3)
;FPAR
IN/OUT Result PICResultType;
NetworkSuspend_ ReqInd (nsrArg,remCSAID, remLegID) to CS
T_Suspended
Network_ Resume_ Ind(nsrArg)
ReleaseReq_ Ind (RArg)
ReleaseInd (RArg)
SuspendT
Service_ FeatureInd (SFtArg)
SetCS
SetLegID (currentLegID, newLegID)
Reset (SuspendT)
Reset (SuspendT)
Reset (SuspendT)
Result := MidCall
CS := Sender
legID := newLegID
Result := Reanswer
Result := ARelease
Result := BRelease
–

Annex A:

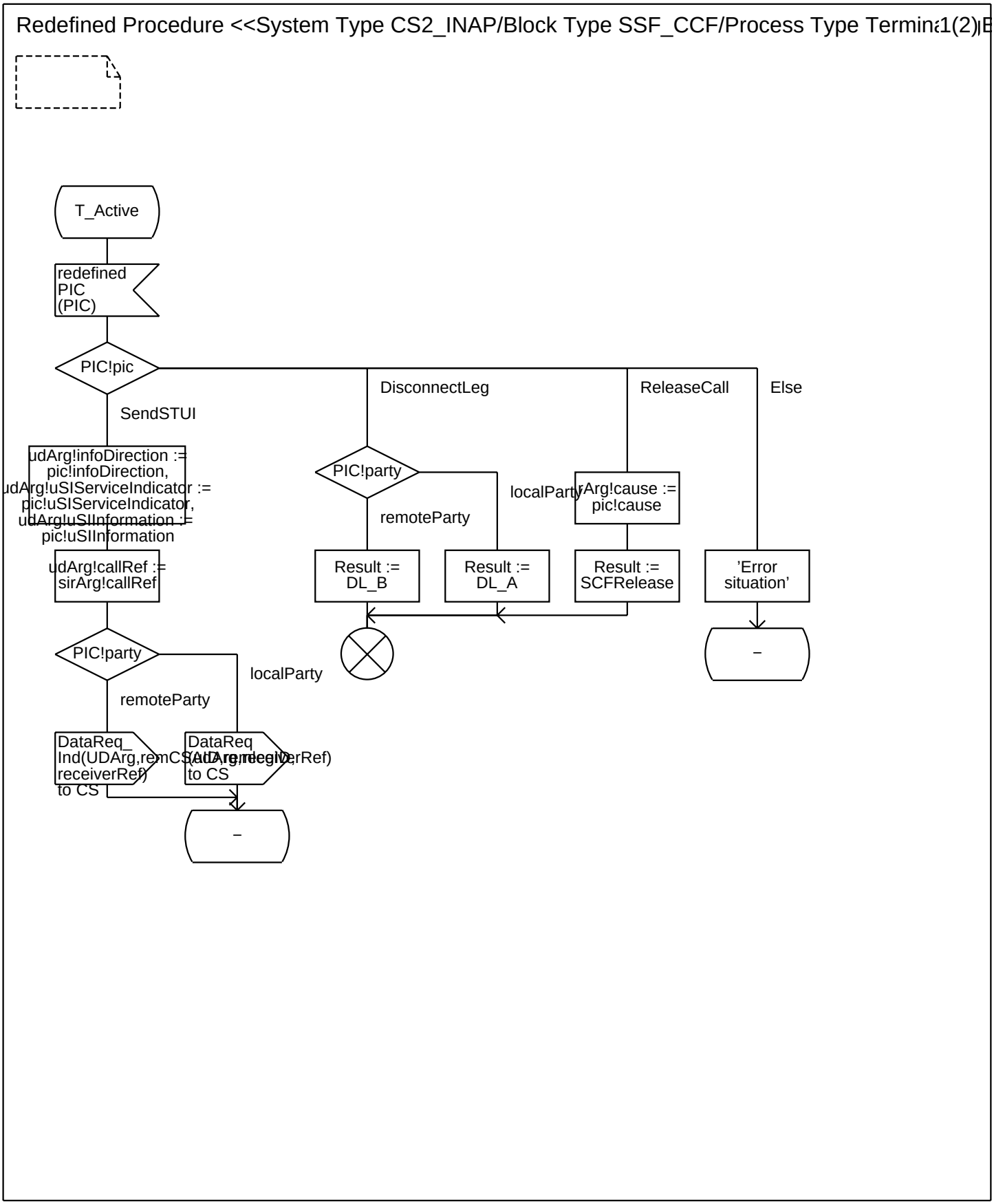
Procedure PIC_T_Suspended 2(3)



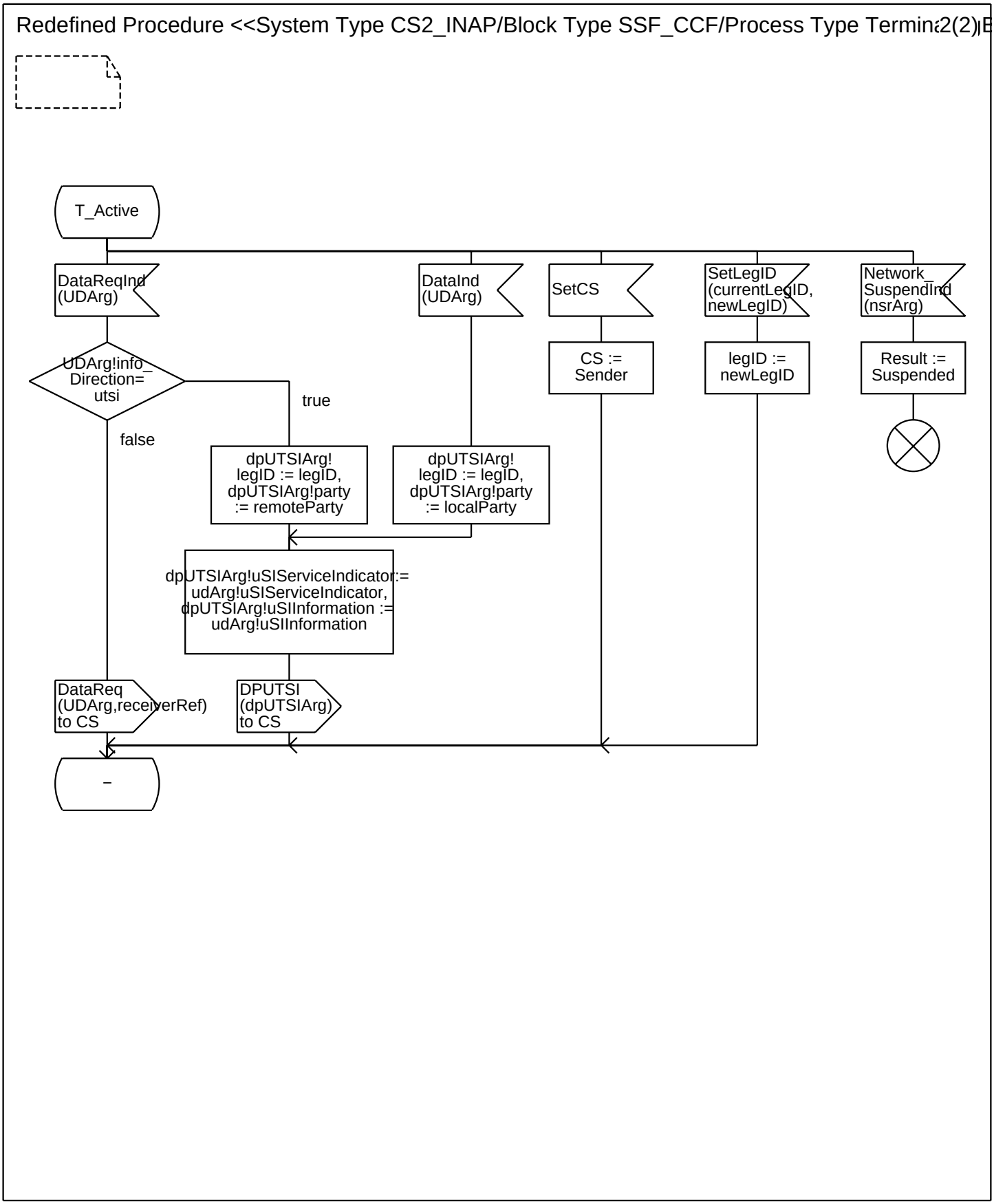
Annex A:


Procedure PIC_T_Suspended
3(3)
;FPAR
IN/OUT Result PICResultType;
T_Suspended
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
Result :=
DL_B
Result :=
DL_A
Result :=
SCFRelease
'Error
situation'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSUDArgReceiverRef)
receiverRef)
to CS
DataReq
(UDArg,receiverRef)
to CS
–
–

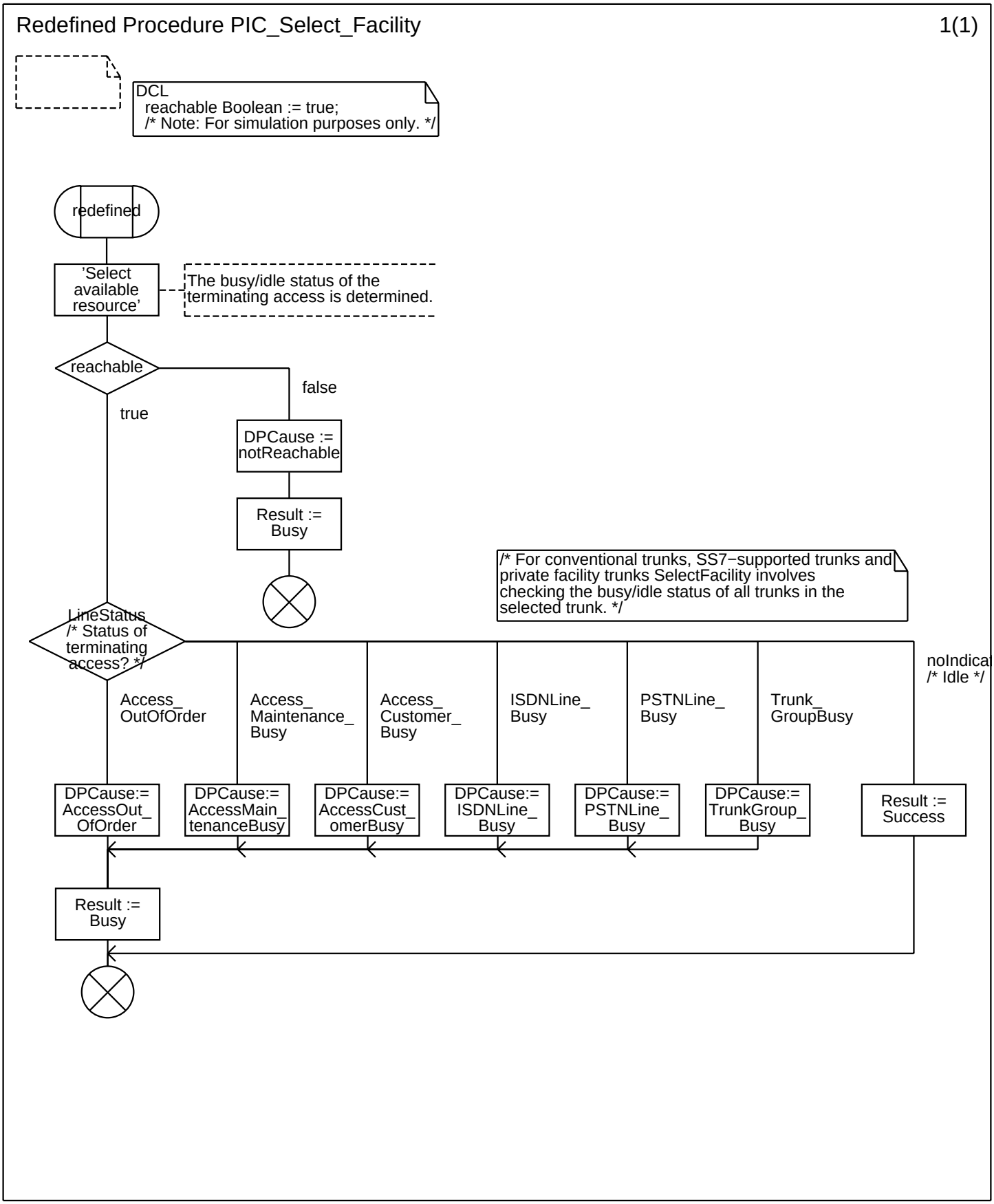
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
1(2)
T_Active
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAddr,receiverRef)
to CS
DataReq
(UDArg,remCSAddr,receiverRef)
to CS
–
PIC!party
remoteParty
localParty
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
Result :=
SCFRelease
'Error
situation'
–

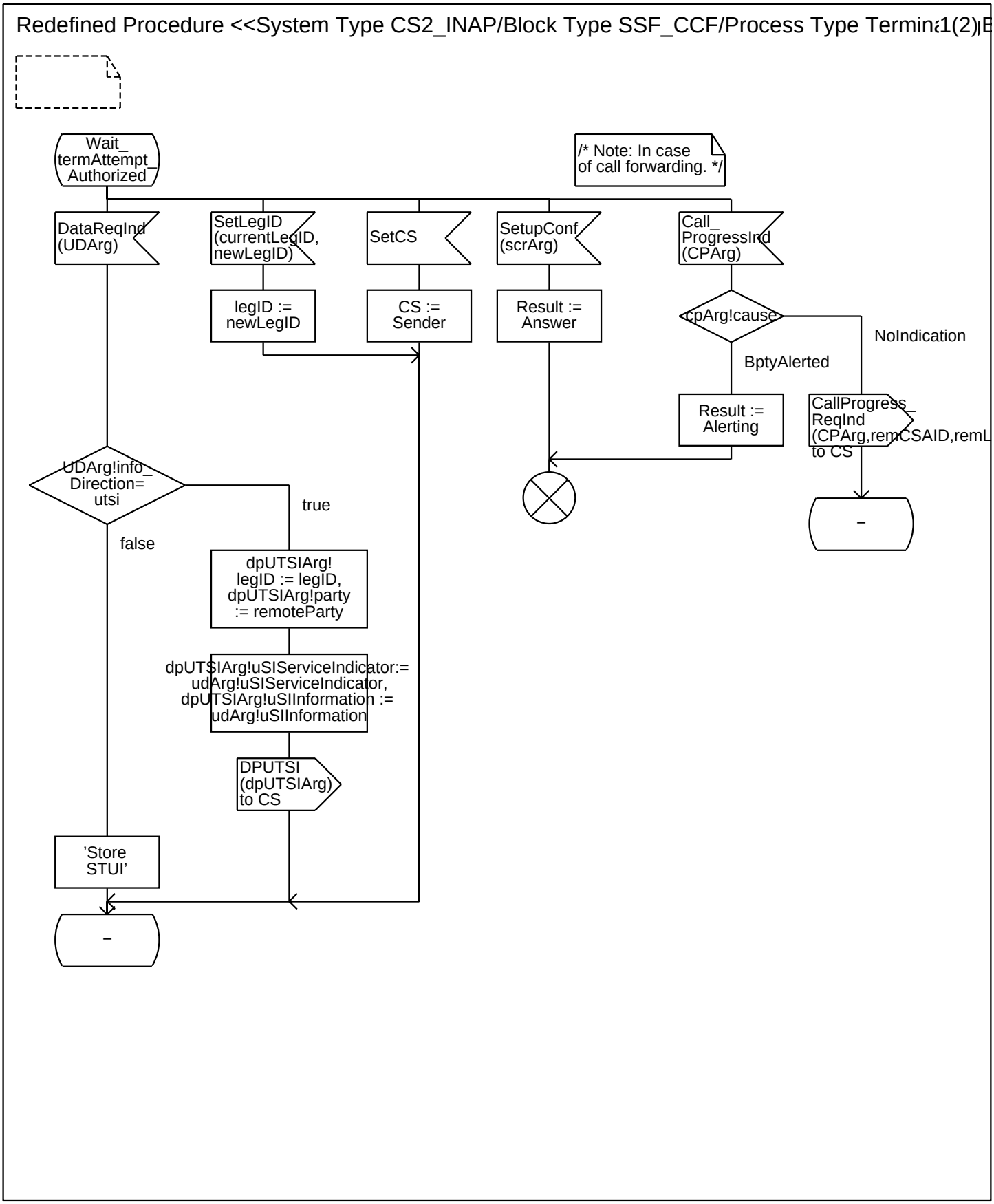
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina2(2)
T_Active
DataReqInd (UDArg)
DataInd (UDArg)
SetCS
SetLegID (currentLegID, newLegID)
Network_ SuspendInd (nsrArg)
UDArg!info_ Direction= utsi
true
false
CS := Sender
legID := newLegID
Result := Suspended
dpUTSIArg! legID := legID, dpUTSIArg!party := remoteParty
dpUTSIArg! legID := legID, dpUTSIArg!party := localParty
dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DataReq (UDArg,receiverRef) to CS
DPUTSI (dpUTSIArg) to CS
–

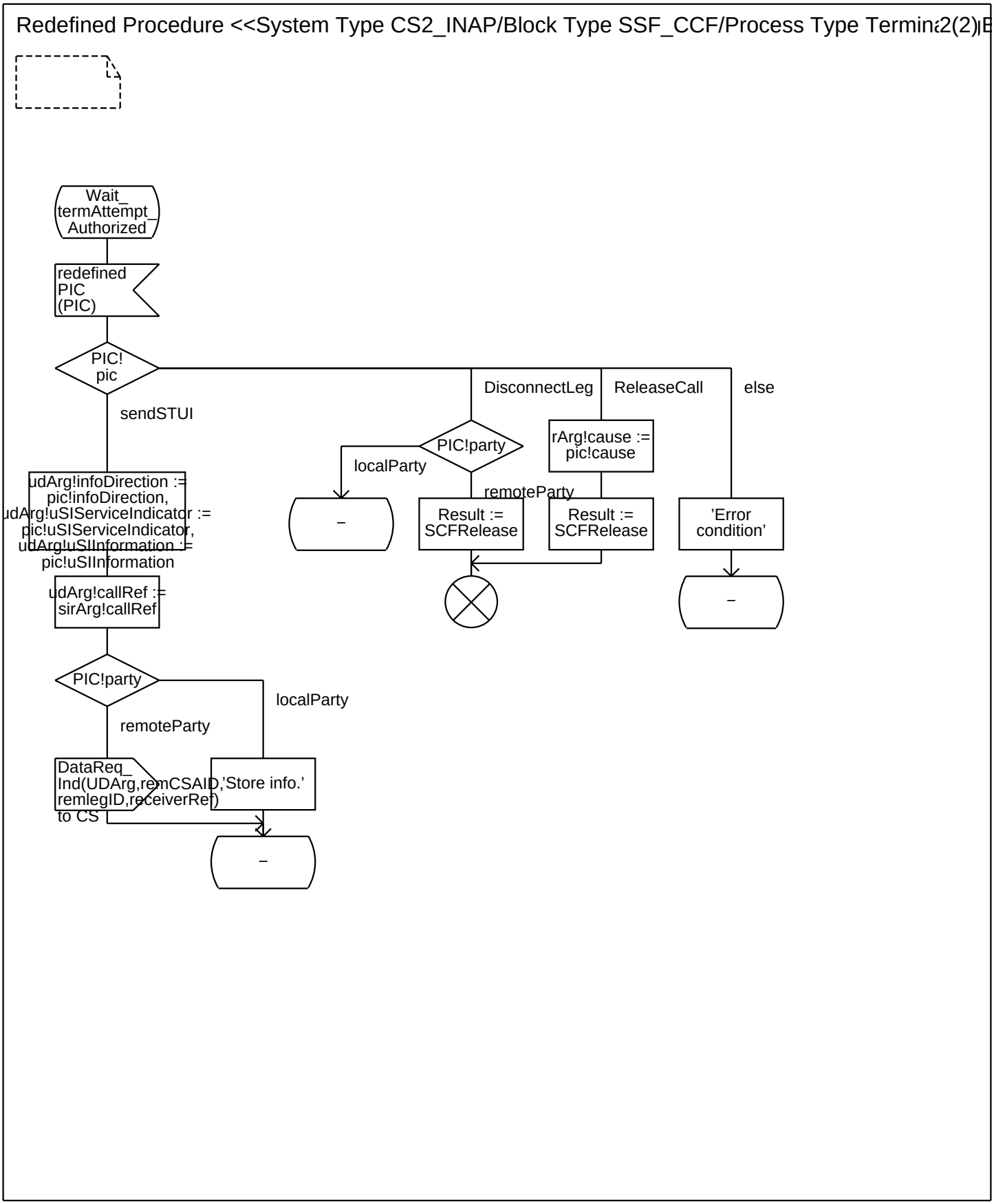
Annex A:


Redefined Procedure PIC_Select_Facility
1(1)
DCL
reachable Boolean := true;
/* Note: For simulation purposes only. */
redefined
'Select available resource'
The busy/idle status of the terminating access is determined.
reachable
false
true
DPCause := notReachable
Result := Busy
/* 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? */
noIndicat /* Idle */
Access_ OutOfOrder
Access_ Maintenance_ Busy
Access_ Customer_ Busy
ISDNLine_ Busy
PSTNLine_ Busy
Trunk_ GroupBusy
DPCause:= AccessOut_ OfOrder
DPCause:= AccessMain_ tenanceBusy
DPCause:= AccessCust_ omerBusy
DPCause:= ISDNLine_ Busy
DPCause:= PSTNLine_ Busy
DPCause:= TrunkGroup_ Busy
Result := Success
Result := Busy

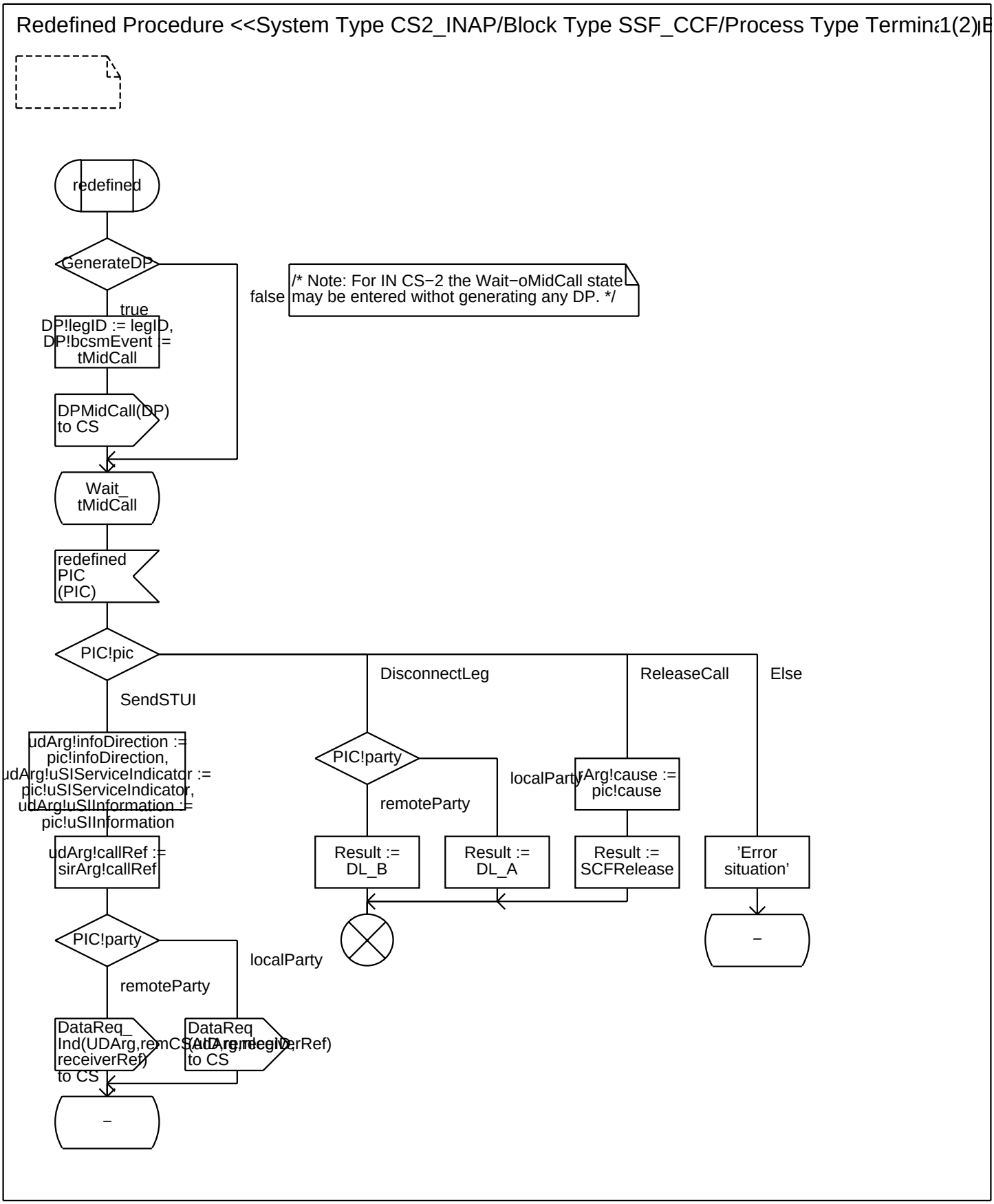
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina 1(2)
Wait_ termAttempt_ Authorized
/* Note: In case of call forwarding. */
DataReqInd (UDArg)
SetLegID (currentLegID, newLegID)
SetCS
SetupConf (scrArg)
Call_ ProgressInd (CPArg)
legID := newLegID
CS := Sender
Result := Answer
cpArg!cause
NoIndication
BptyAlerted
Result := Alerting
CallProgress_ ReqInd (CPArg,remCSAID,remL
to CS
UDArg!info_ Direction= utsi
true
false
dpUTSIArg! legID := legID, dpUTSIArg!party := remoteParty
dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
'Store STUI'

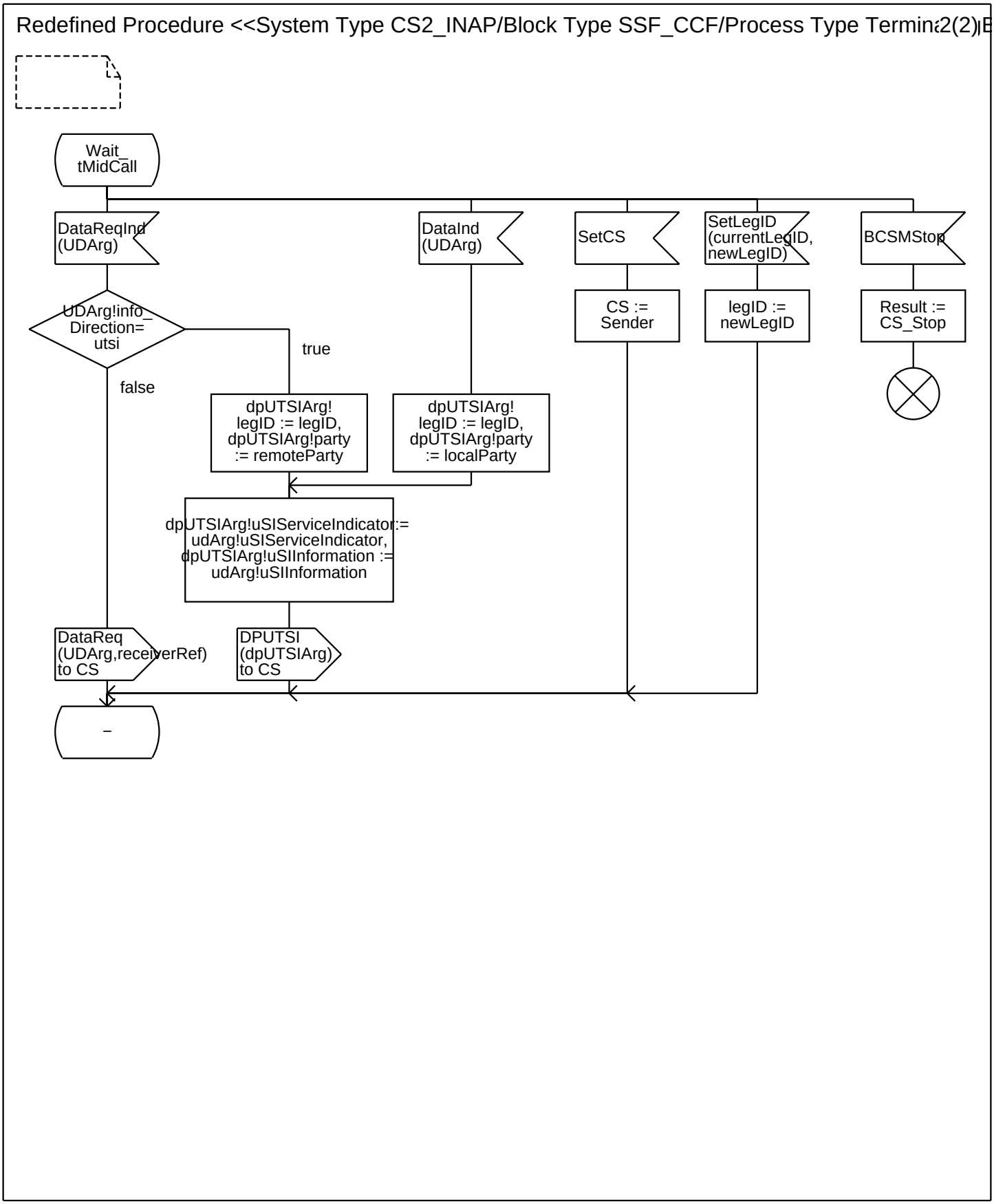
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina2(2)
Wait_ termAttempt_ Authorized
redefined PIC (PIC)
PIC! pic
sendSTUI
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
udArg!callRef := sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_ Ind(UDArg,remCSAID,remlegID,receiverRef) to CS
'Store info.'
–
DisconnectLeg
ReleaseCall
else
PIC!party
localParty
remoteParty
rArg!cause := pic!cause
Result := SCFRelease
Result := SCFRelease
'Error condition'
–

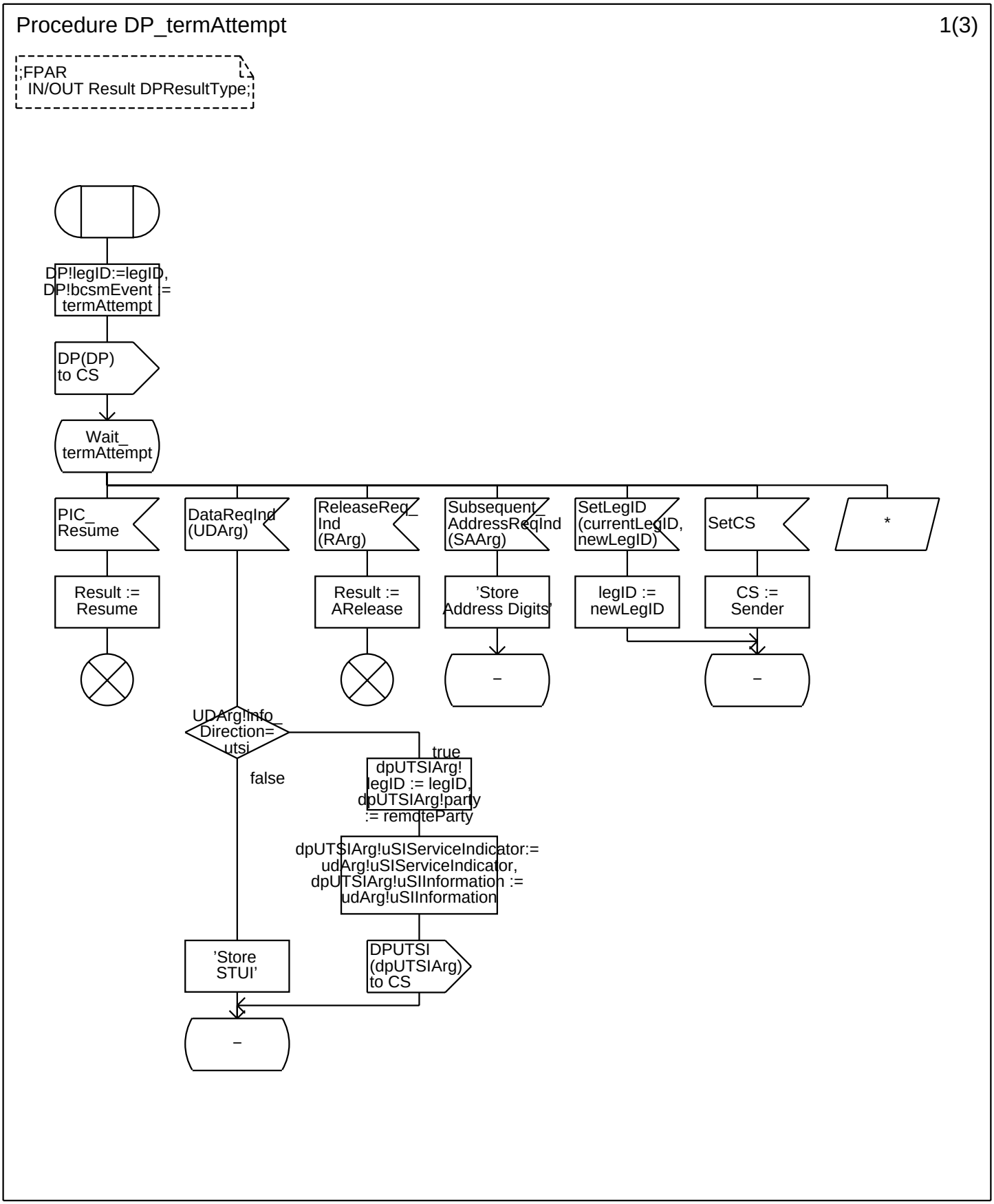
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
1(2)
redefined
GenerateDP
true
false
/* Note: For IN CS−2 the Wait−oMidCall state may be entered withot generating any DP. */
DP!legID := legID,
DP!bcsmEvent :=
tMidCall
DPMidCall(DP)
to CS
Wait_
tMidCall
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
Result :=
DL_B
Result :=
DL_A
Result :=
SCFRelease
'Error
situation'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAddressID,
receiverRef)
to CS
DataReq
(UDArg,remCSAddressID,
receiverRef)
to CS
–
–

Annex A:

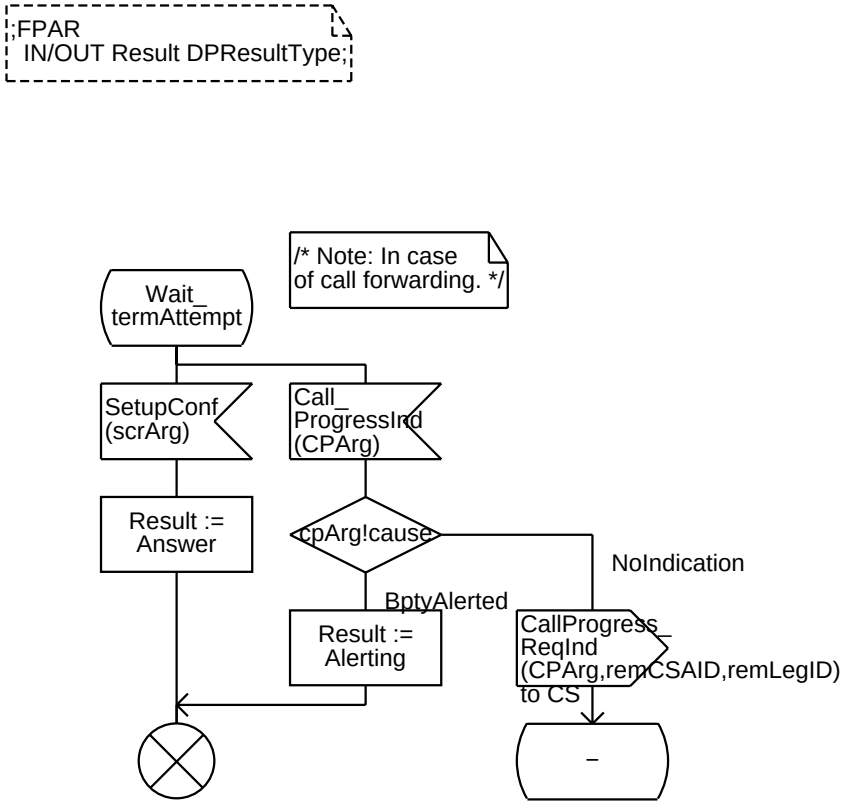

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
2(2)
Wait_
tMidCall
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
BCSMStop
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
Result :=
CS_Stop
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator :=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:

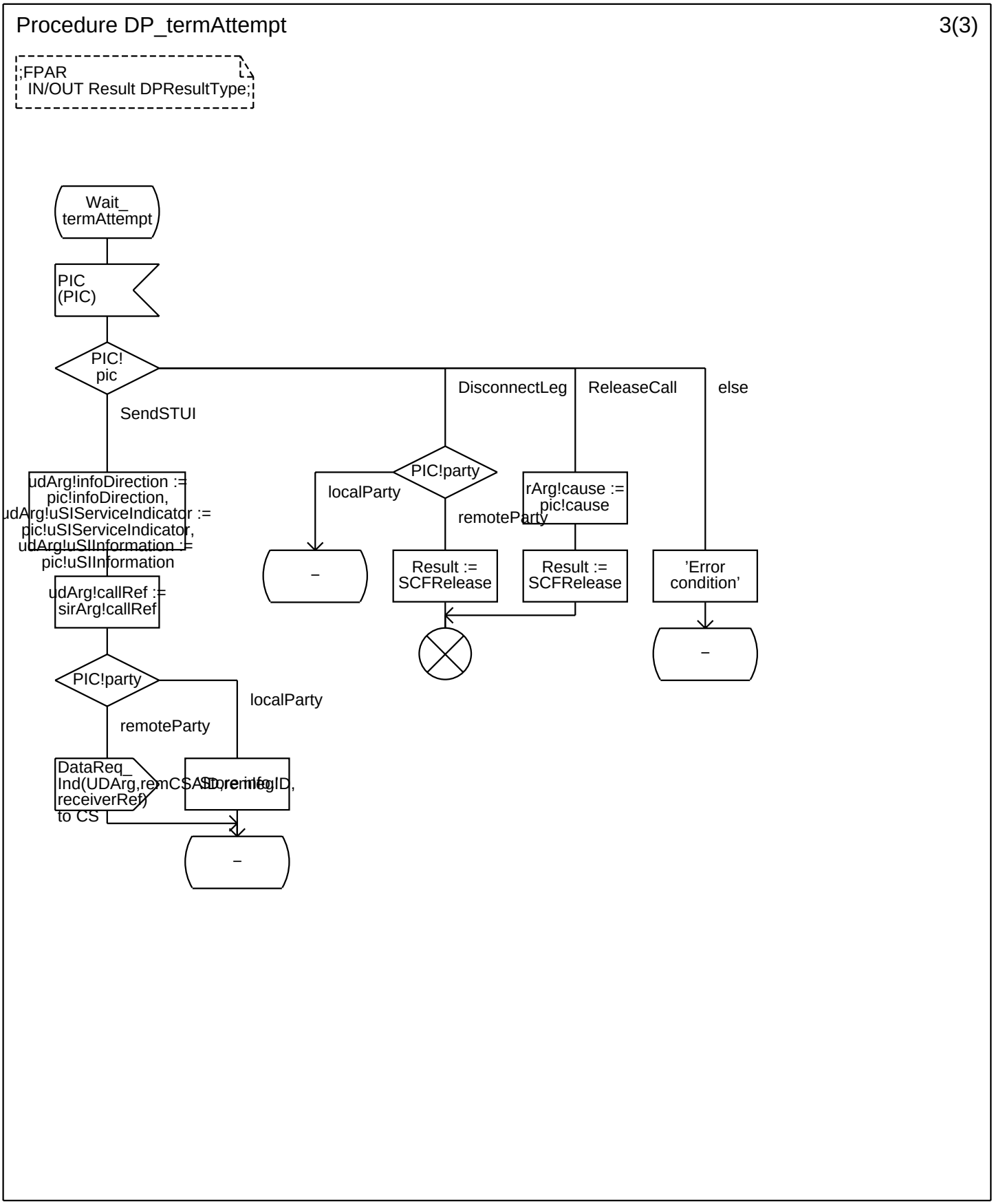

Procedure DP_termAttempt
1(3)
;FPAR
IN/OUT Result DPResultType;
DP!legID:=legID,
DP!bcsmEvent :=
termAttempt
DP(DP)
to CS
Wait_
termAttempt
PIC_
Resume
DataReqInd
(UDArg)
ReleaseReq_
Ind
(RArg)
Subsequent_
AddressReqInd
(SAArg)
SetLegID
(currentLegID,
newLegID)
SetCS
*
Result :=
Resume
Result :=
ARelease
'Store
Address Digits'
legID :=
newLegID
CS :=
Sender
–
–
UDArg!Info_
Direction=
utsi
true
false
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
'Store
STUI'
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:

Procedure DP_termAttempt 2(3)

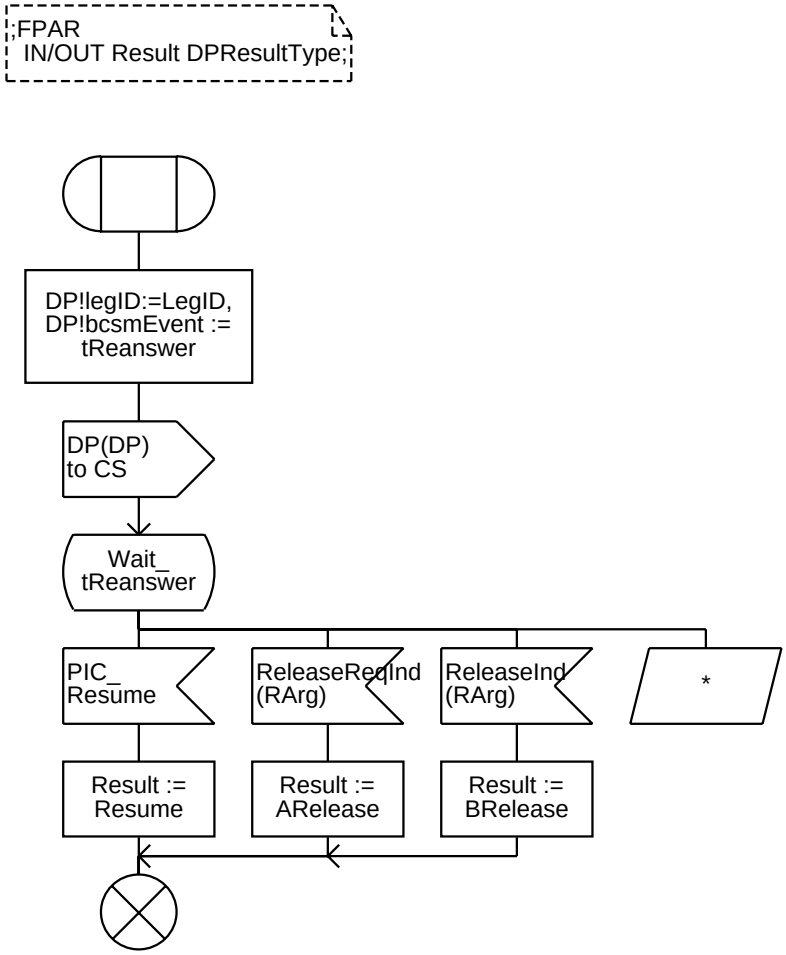


Annex A:

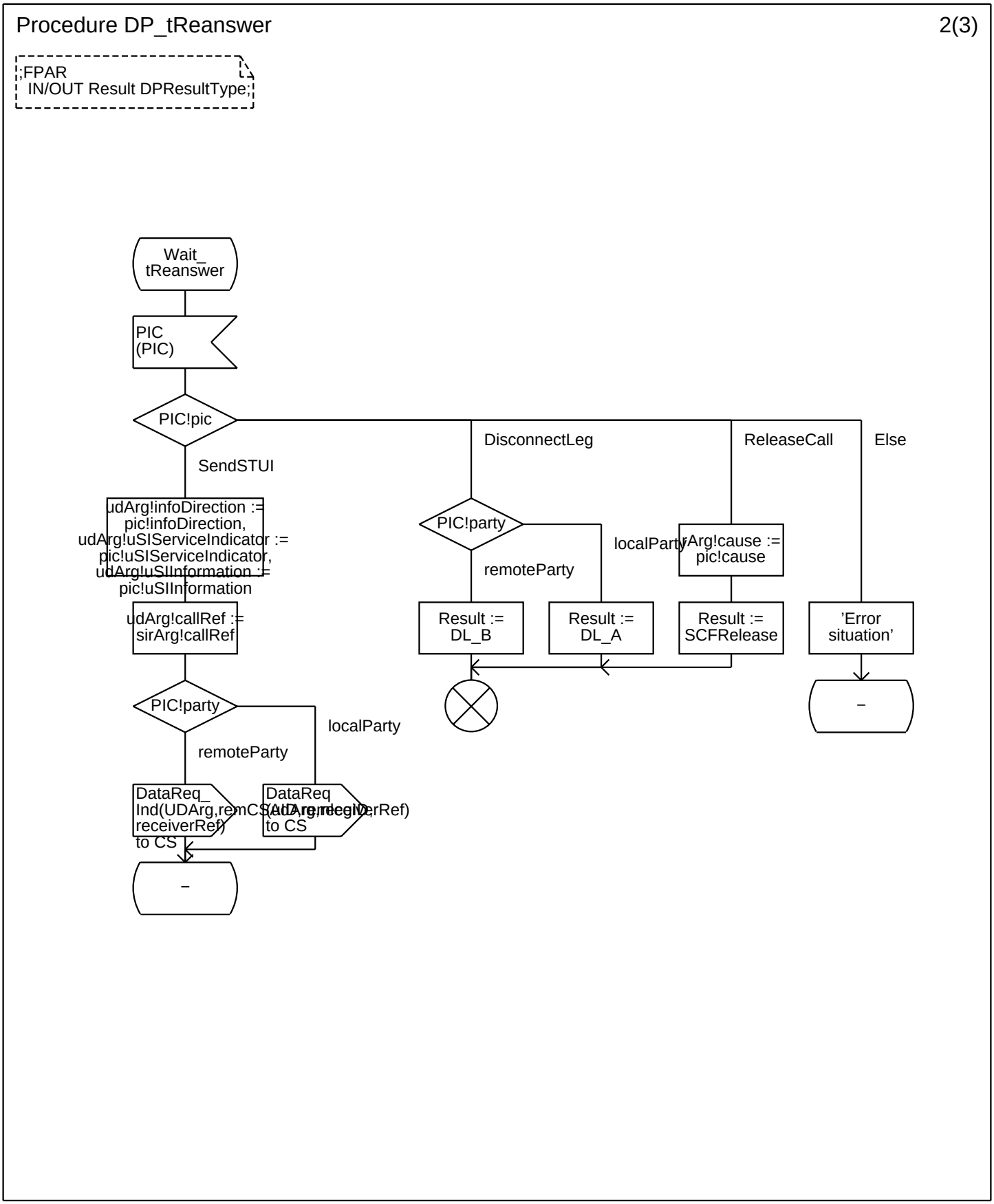

Procedure DP_termAttempt
3(3)
;FPAR
IN/OUT Result DPResultType;
Wait_
termAttempt
PIC
(PIC)
PIC!
pic
SendSTUI
DisconnectLeg
ReleaseCall
else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
–
Result :=
SCFRelease
Result :=
SCFRelease
'Error
condition'
–
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAddr,
receiverRef)
to CS
StoreInfo
–

Annex A:

Procedure DP_tReanswer 1(3)

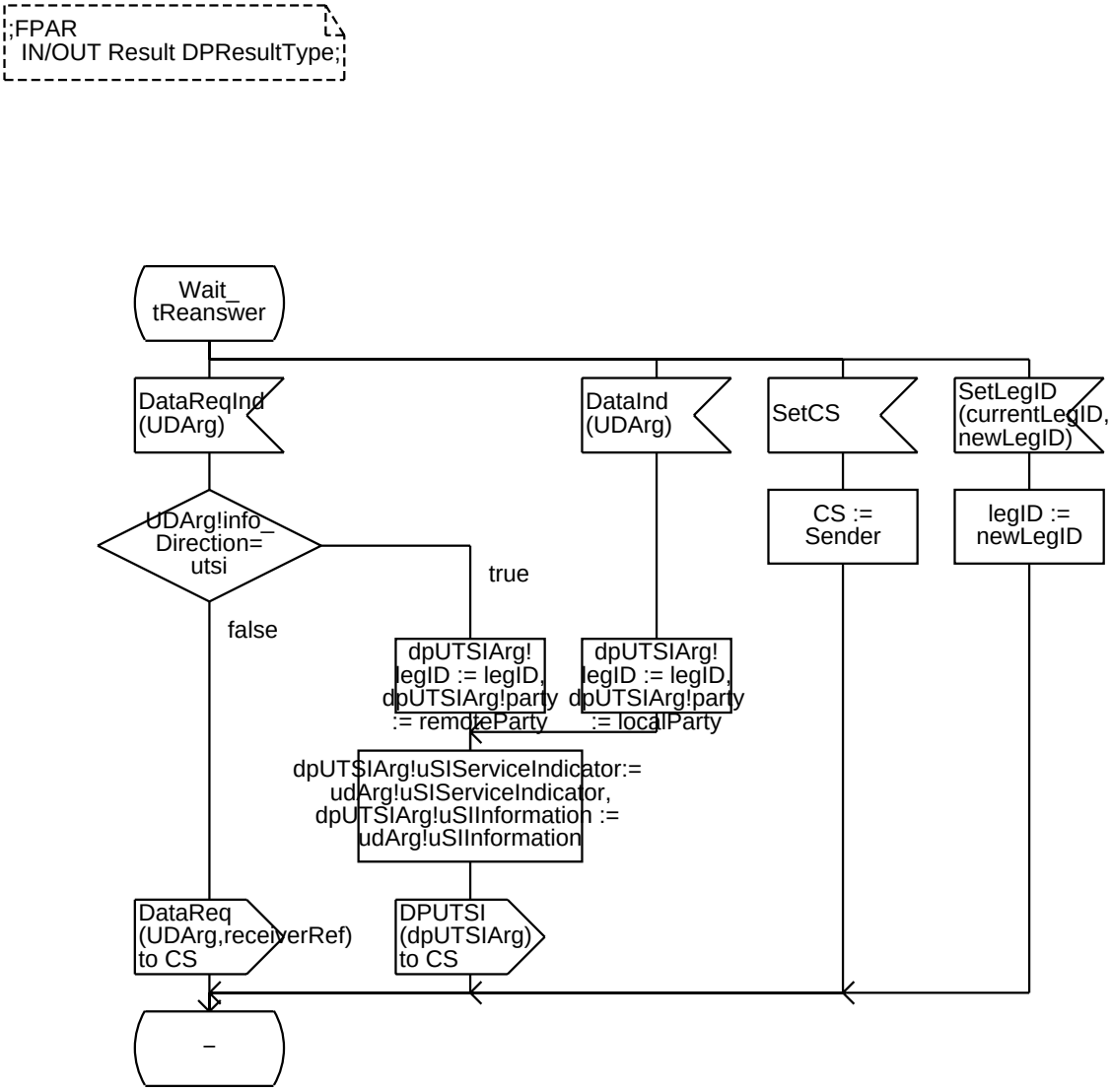


Annex A:

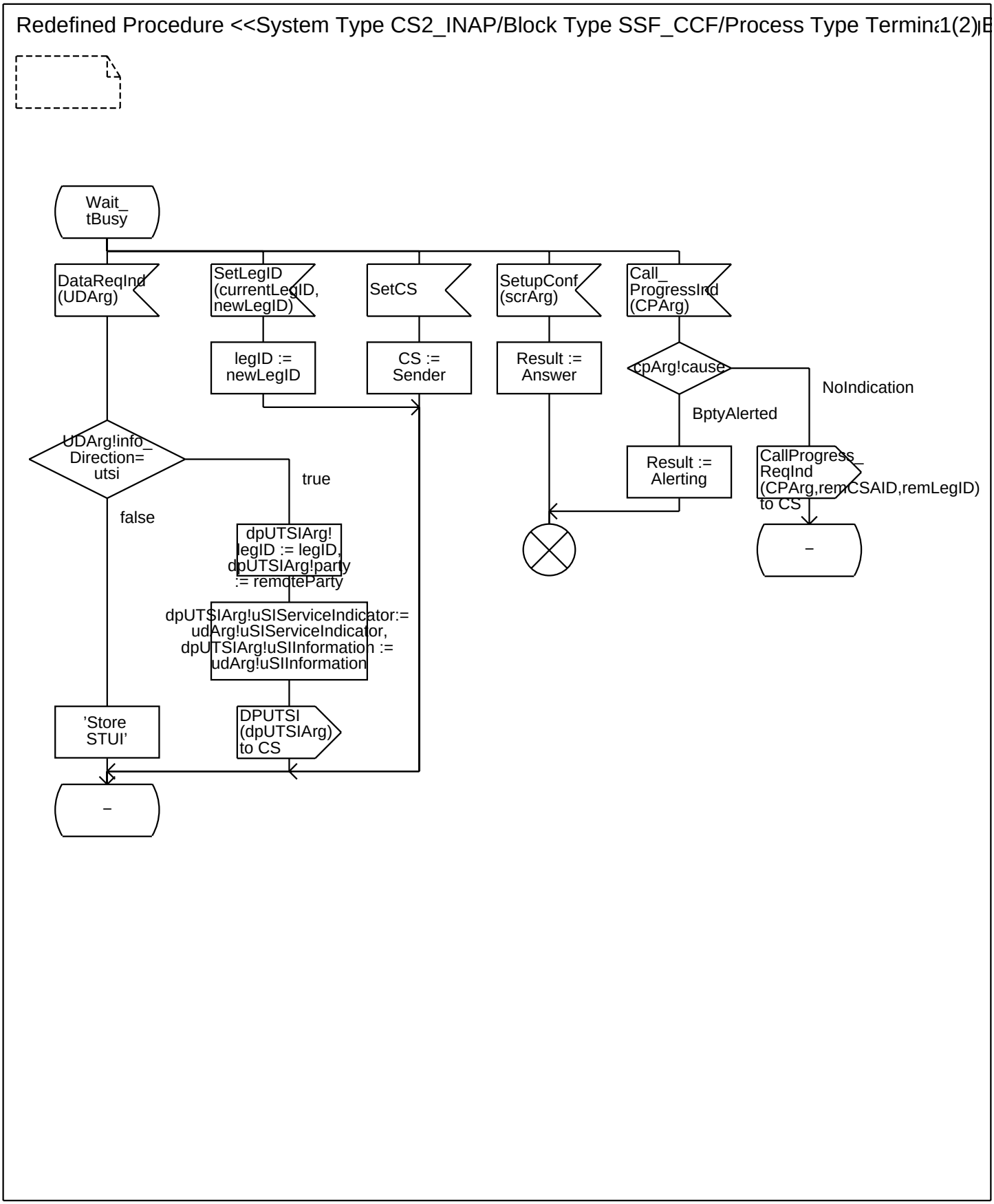

Procedure DP_tReanswer
2(3)
;FPAR
IN/OUT Result DPResultType;
Wait_
tReanswer
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAddr,
receiverRef)
to CS
DataReq
(UDArg,remCSAddr,
receiverRef)
to CS
–
PIC!party
remoteParty
localParty
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
Result :=
SCFRelease
'Error
situation'
–

Annex A:

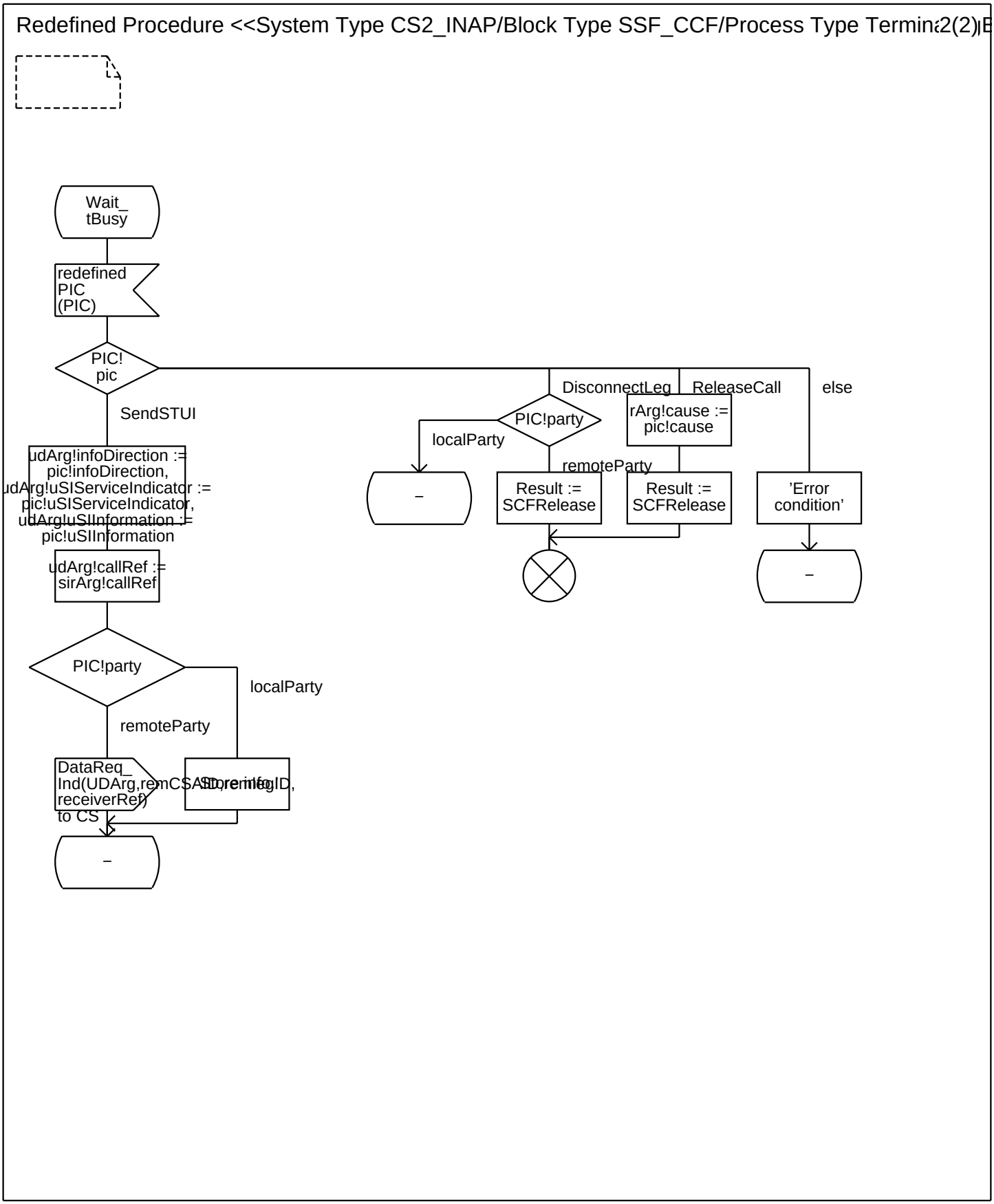
Procedure DP_tReanswer 3(3)



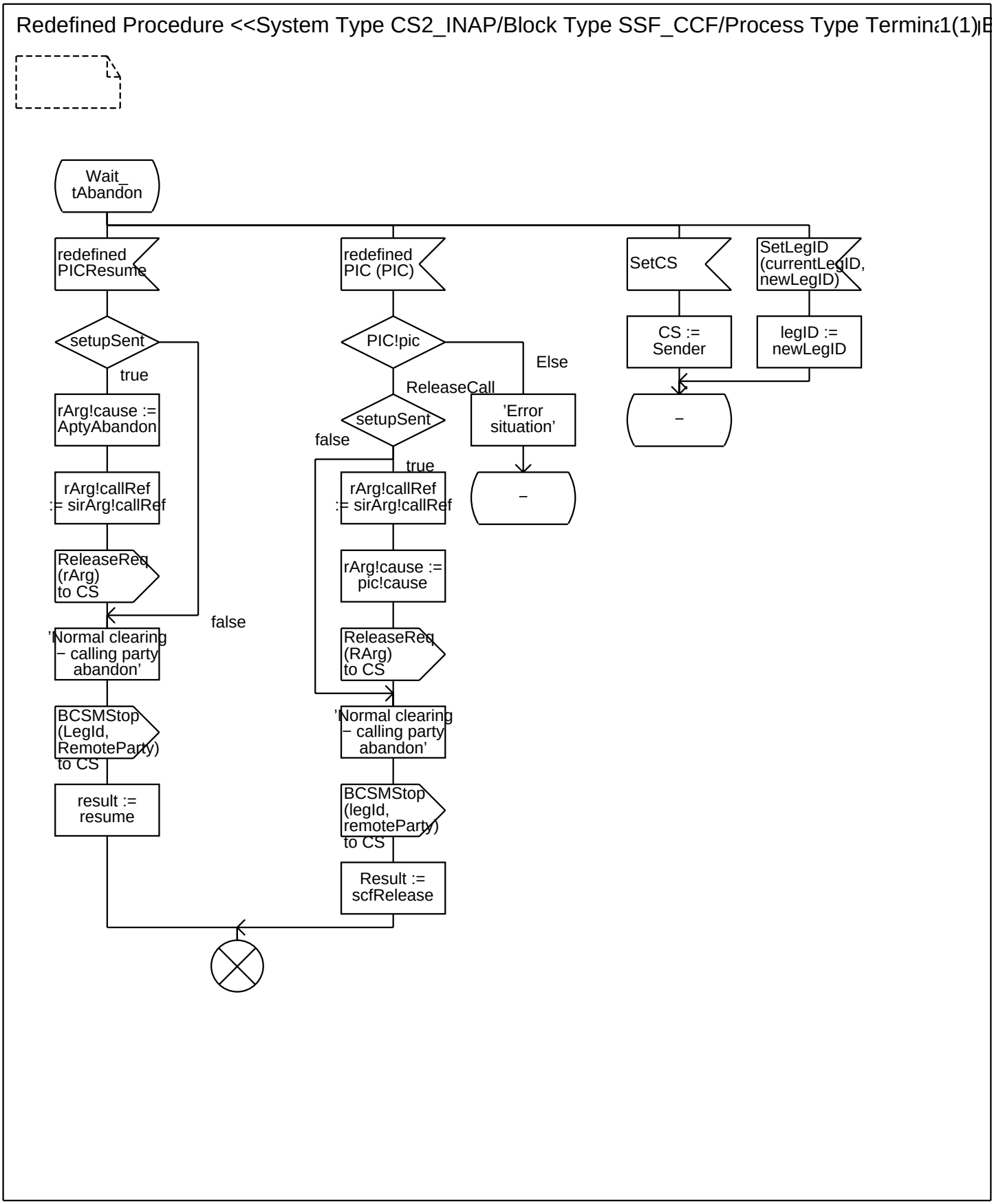
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina1(2)
Wait_ tBusy
DataReqInd (UDArg)
SetLegID (currentLegID, newLegID)
SetCS
SetupConf (scrArg)
Call_ ProgressInd (CPArg)
legID := newLegID
CS := Sender
Result := Answer
cpArg!cause
NoIndication
BptyAlerted
UDArg!info_ Direction= utsi
true
false
Result := Alerting
CallProgress_ ReqInd (CPArg,remCSAID,remLegID) to CS
dpUTSIArg! legID := legID, dpUTSIArg!party := remoteParty
dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
'Store STUI'
DPUTSI (dpUTSIArg) to CS
–
–

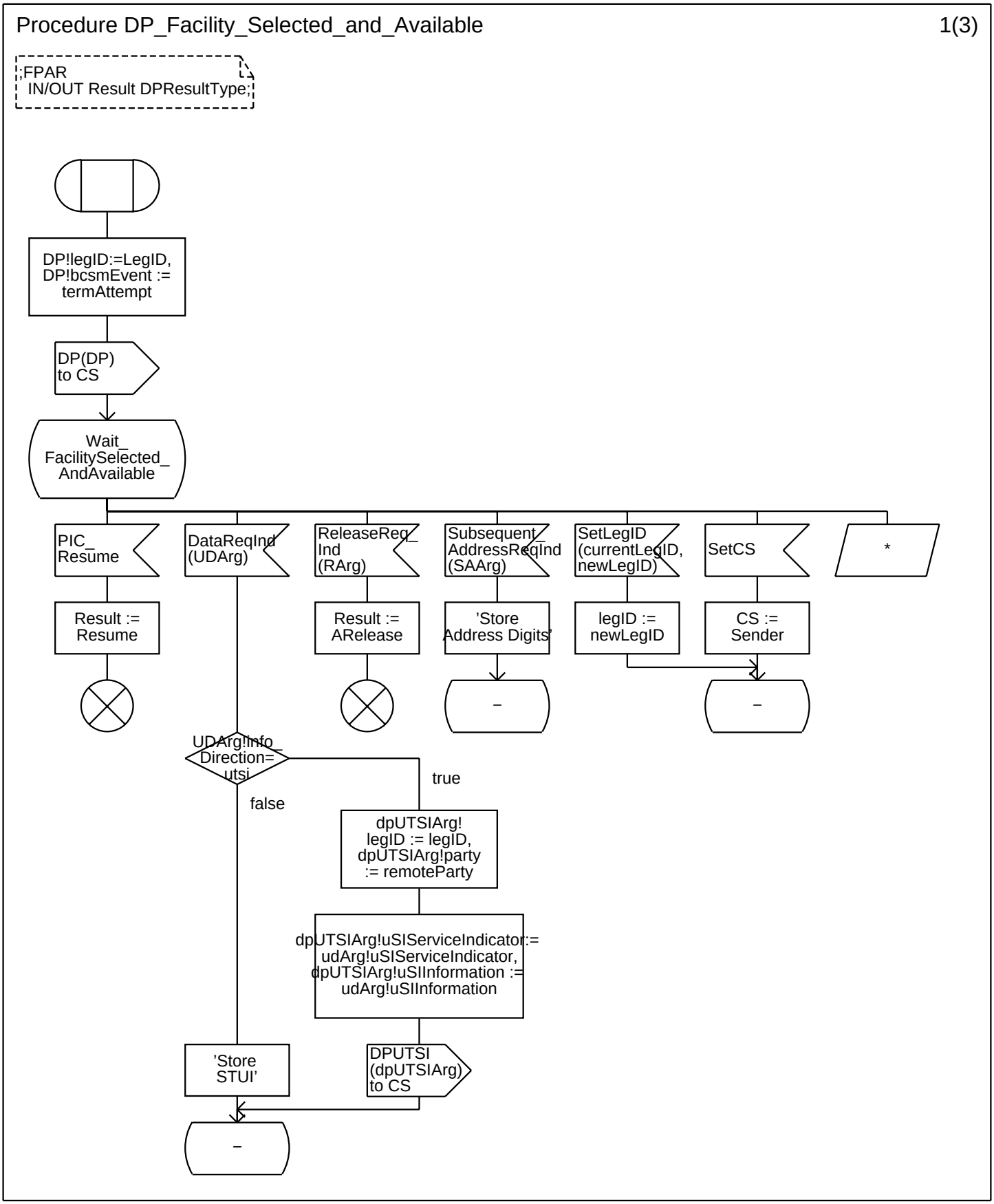
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
2(2)
Wait_
tBusy
redefined
PIC
(PIC)
PIC!
pic
SendSTUI
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAddr,remMsgID,
receiverRef)
to CS
DisconnectLeg
ReleaseCall
else
PIC!party
rArg!cause :=
pic!cause
localParty
remoteParty
Result :=
SCFRelease
Result :=
SCFRelease
'Error
condition'

Annex A:

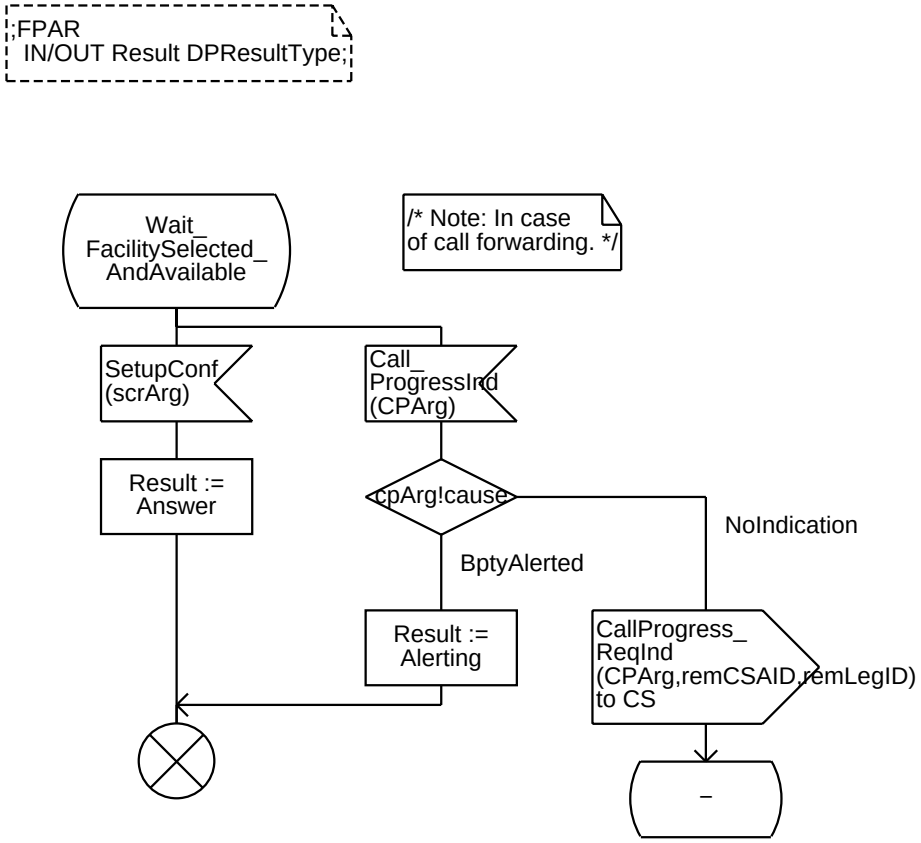

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
1(1)
Wait_
tAbandon
redefined
PICResume
setupSent
true
rArg!cause :=
AptyAbandon
rArg!callRef
:= sirArg!callRef
ReleaseReq
(rArg)
to CS
false
'Normal clearing
− calling party
abandon'
BCSMStop
(LegId,
RemoteParty)
to CS
result :=
resume
redefined
PIC (PIC)
PIC!pic
Else
ReleaseCall
setupSent
false
true
'Error
situation'
–
rArg!callRef
:= sirArg!callRef
rArg!cause :=
pic!cause
ReleaseReq
(RArg)
to CS
'Normal clearing
− calling party
abandon'
BCSMStop
(legId,
remoteParty)
to CS
Result :=
scfRelease
SetCS
CS :=
Sender
SetLegID
(currentLegID,
newLegID)
legID :=
newLegID
–

Annex A:

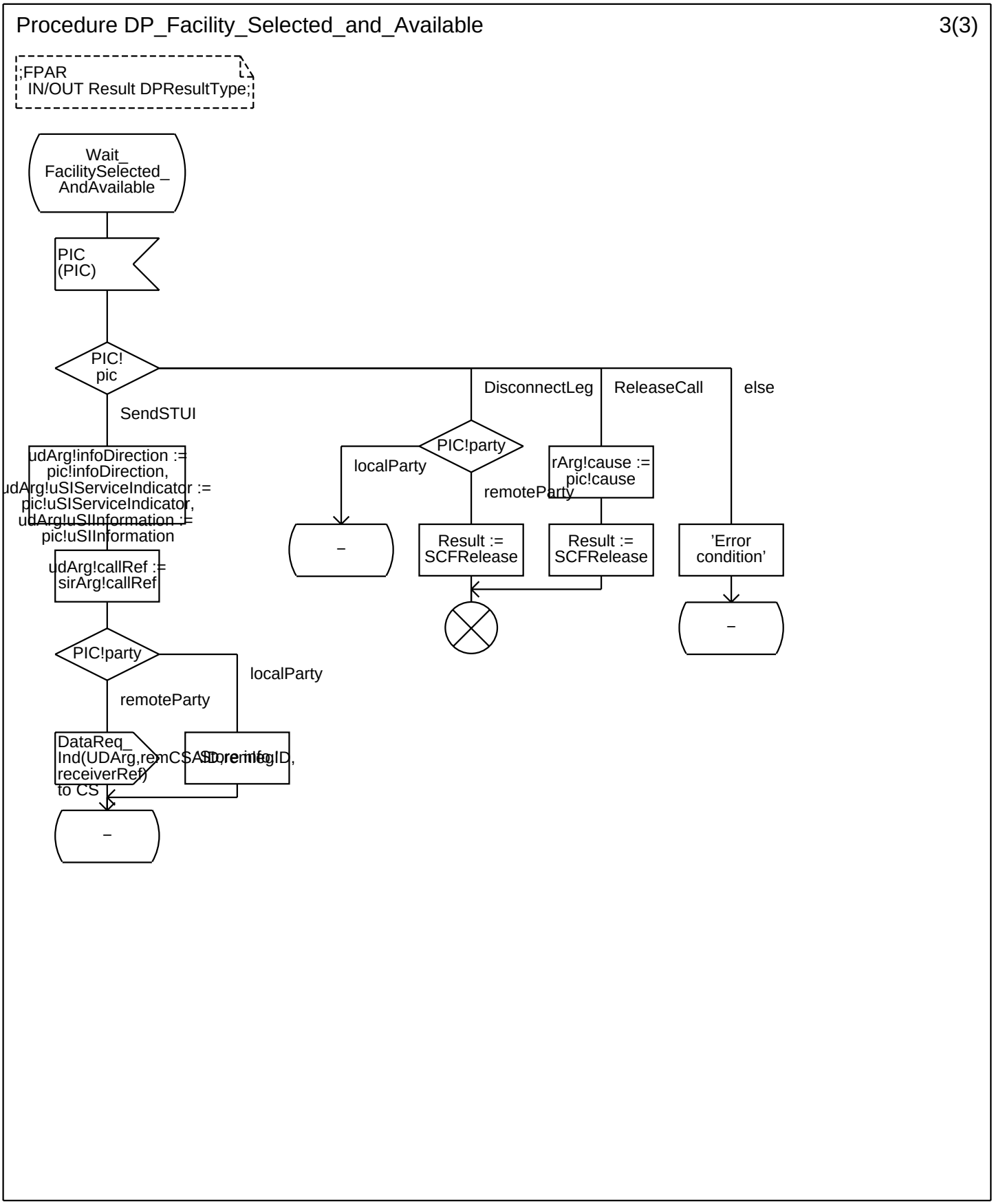

Procedure DP_Facility_Selected_and_Available
1(3)
;FPAR
IN/OUT Result DPResultType;
DP!legID:=LegID,
DP!bcsmEvent :=
termAttempt
DP(DP)
to CS
Wait_
FacilitySelected_
AndAvailable
PIC_
Resume
DataReqInd
(UDArg)
ReleaseReq_
Ind
(RArg)
Subsequent_
AddressReqInd
(SAArg)
SetLegID
(currentLegID,
newLegID)
SetCS
*
Result :=
Resume
Result :=
ARelease
'Store
Address Digits'
legID :=
newLegID
CS :=
Sender
–
–
UDArg!Info_
Direction=
utsi
true
false
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
'Store
STUI'
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:

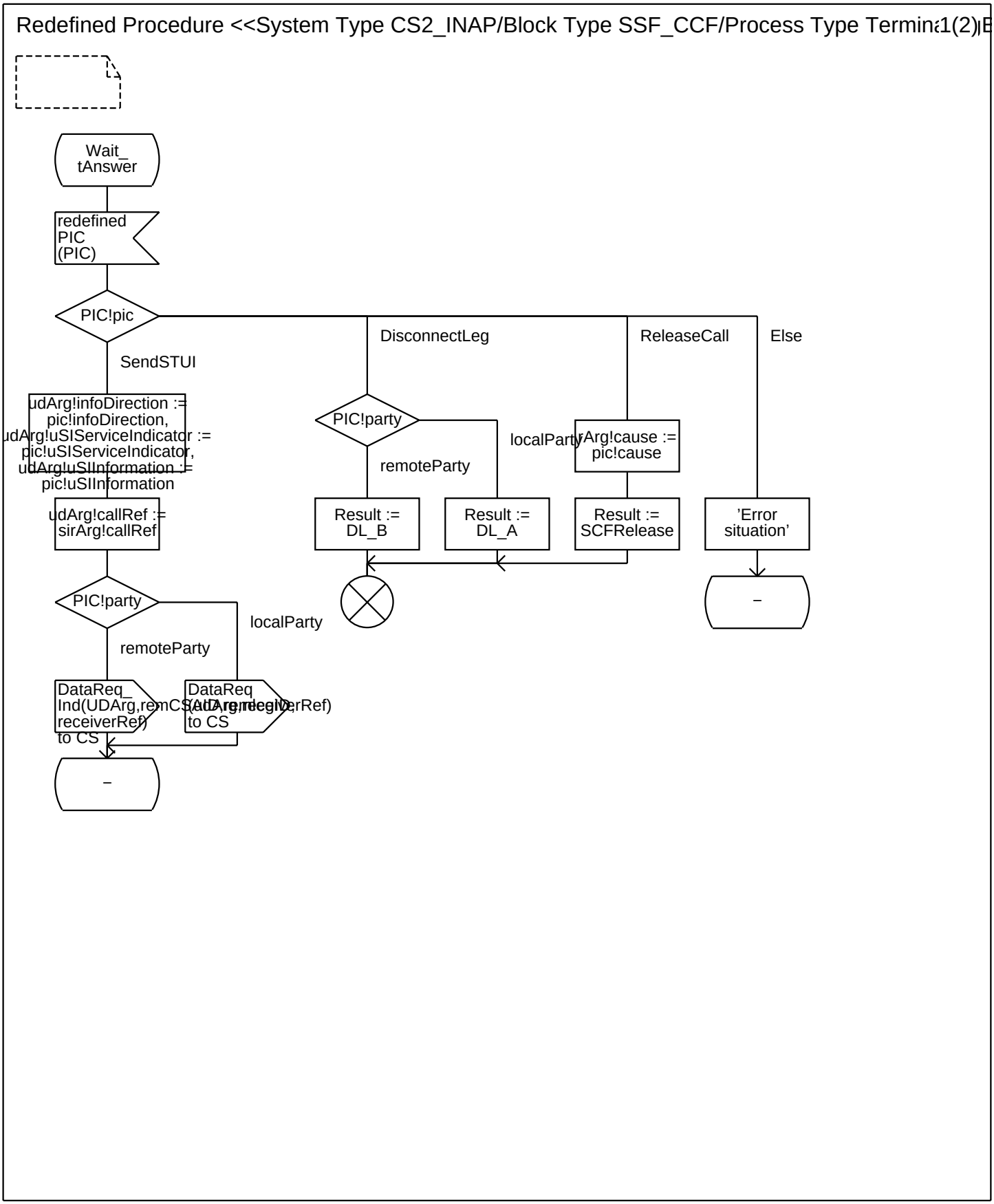
Procedure DP_Facility_Selected_and_Available 2(3)



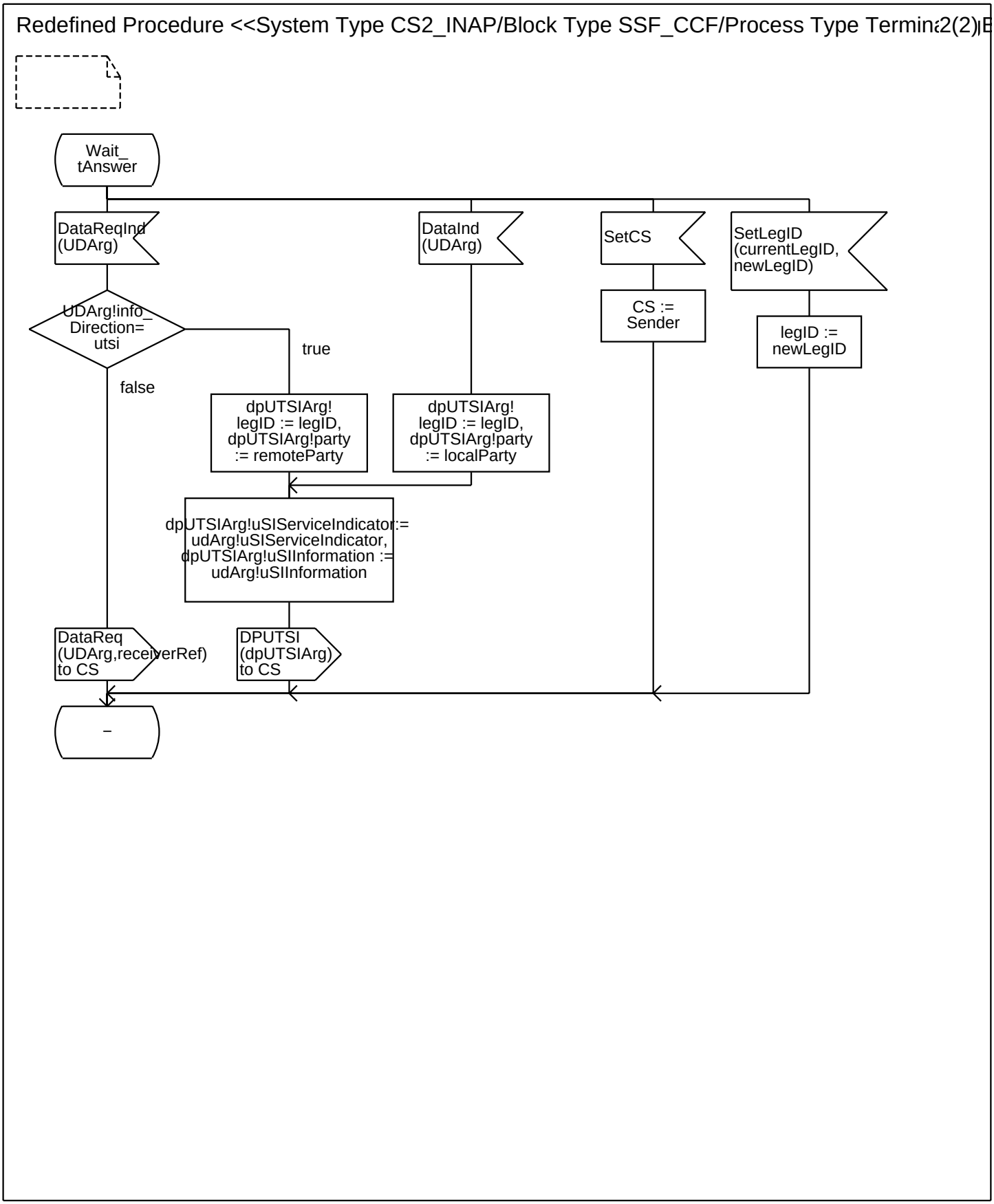
Annex A:


Procedure DP_Facility_Selected_and_Available
3(3)
;FPAR
IN/OUT Result DPResultType;
Wait_
FacilitySelected_
AndAvailable
PIC
(PIC)
PIC!
pic
SendSTUI
DisconnectLeg
ReleaseCall
else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
–
Result :=
SCFRelease
Result :=
SCFRelease
'Error
condition'
–
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,
receiverRef)
to CS
–

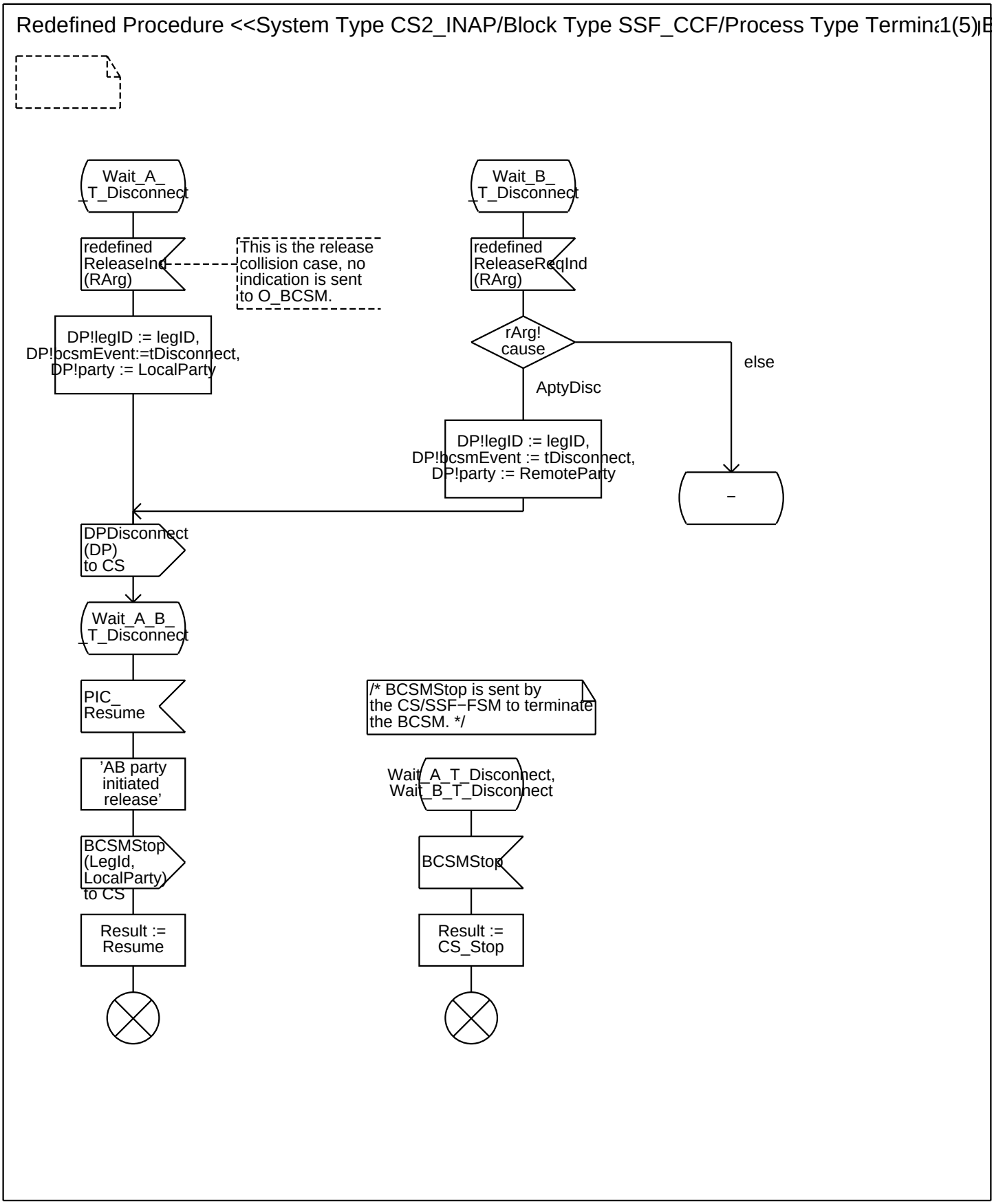
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
1(2)
Wait_
tAnswer
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
Result :=
DL_B
Result :=
DL_A
Result :=
SCFRelease
'Error
situation'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSCallRef,
receiverRef)
to CS
DataReq
(UDArg,remCSCallRef,
receiverRef)
to CS
–
–

Annex A:

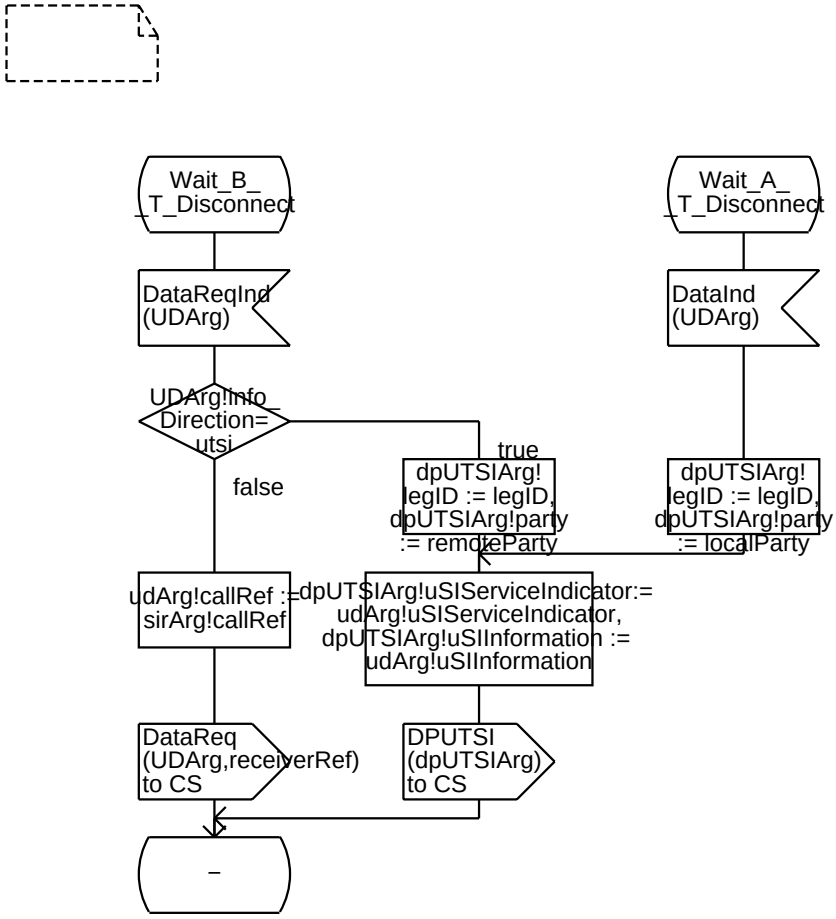

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina2(2)
Wait_
tAnswer
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
1(5)
Wait_A_
_T_Disconnect
redefined
ReleaseInd
(RArg)
This is the release
collision case, no
indication is sent
to O_BCSM.
DP!legID := legID,
DP!bcsmEvent:=tDisconnect,
DP!party := LocalParty
Wait_B_
_T_Disconnect
redefined
ReleaseReqInd
(RArg)
rArg!
cause
else
AptyDisc
DP!legID := legID,
DP!bcsmEvent := tDisconnect,
DP!party := RemoteParty
–
DPDisconnect
(DP)
to CS
Wait_A_B_
_T_Disconnect
PIC_
Resume
/* BCSMStop is sent by
the CS/SSF−FSM to terminate
the BCSM. */
'AB party
initiated
release'
Wait_A_T_Disconnect,
Wait_B_T_Disconnect
BCSMStop
(LegId,
LocalParty)
to CS
BCSMStop
Result :=
Resume
Result :=
CS_Stop

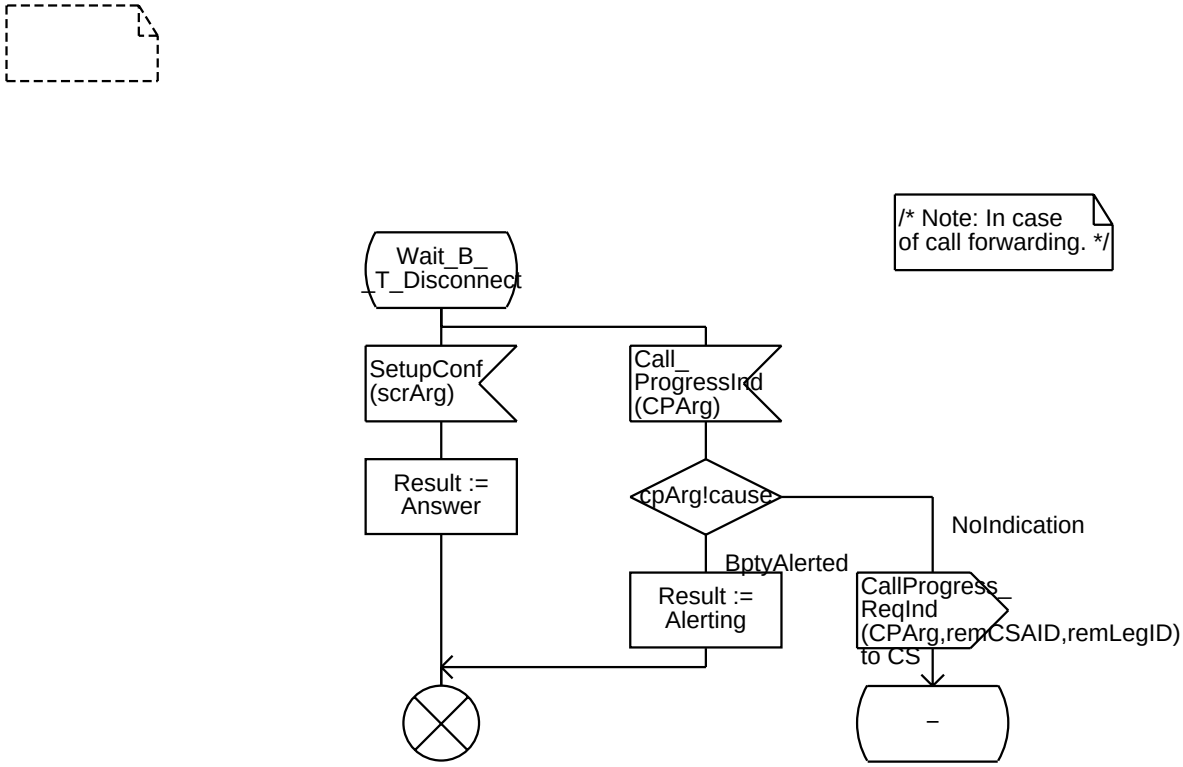
Annex A:

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina2(5)

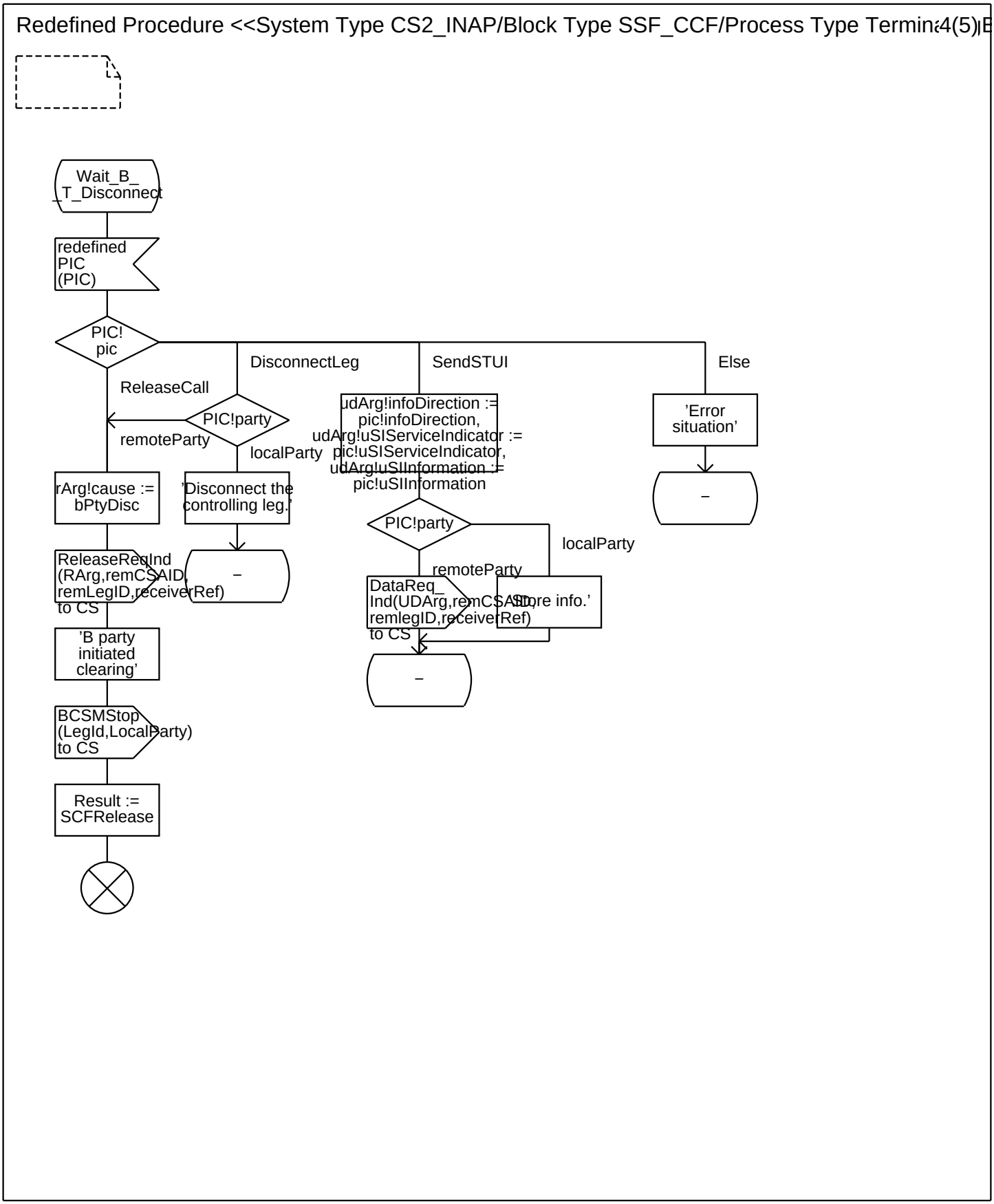


Annex A:

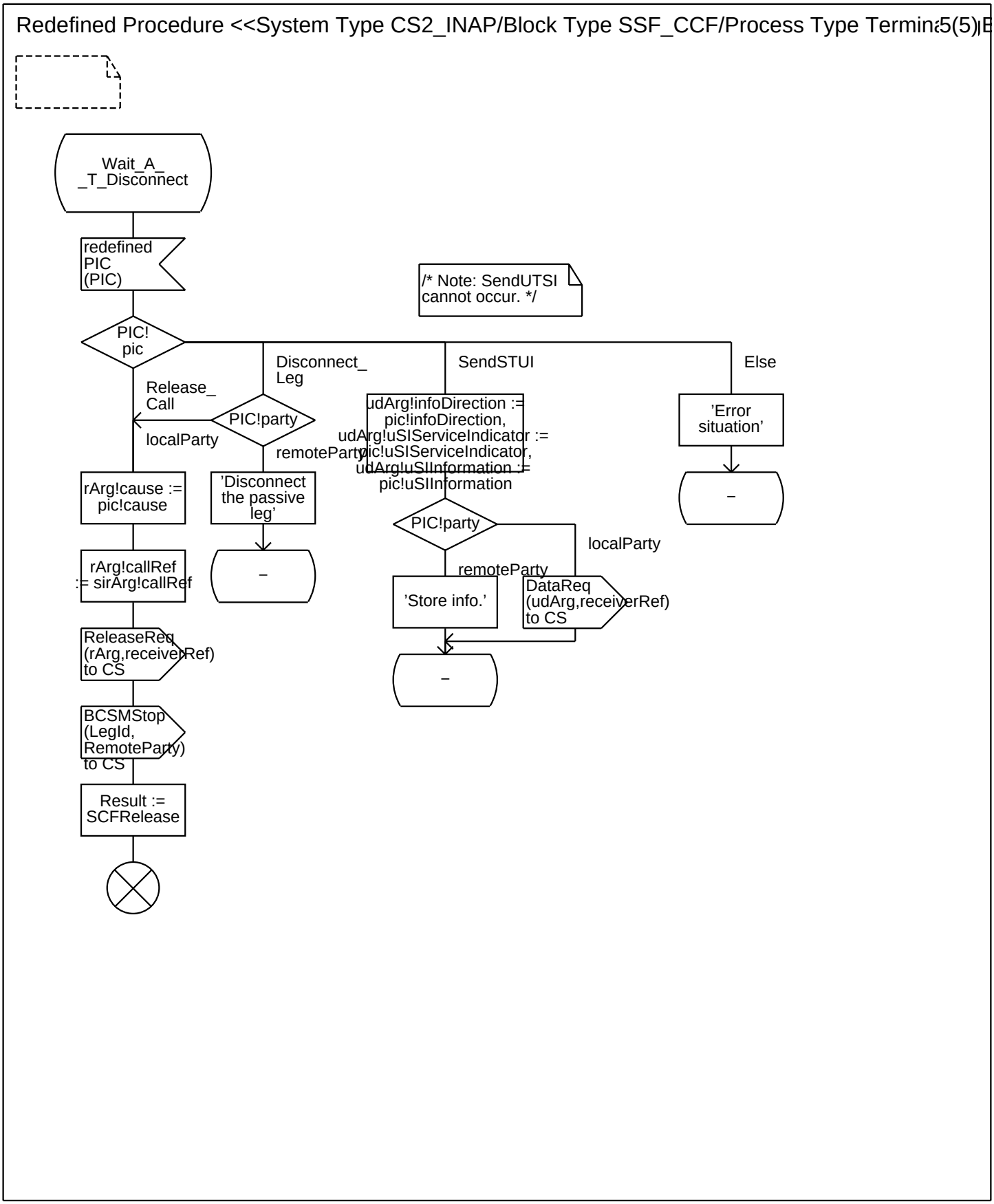
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina 3(5)



Annex A:

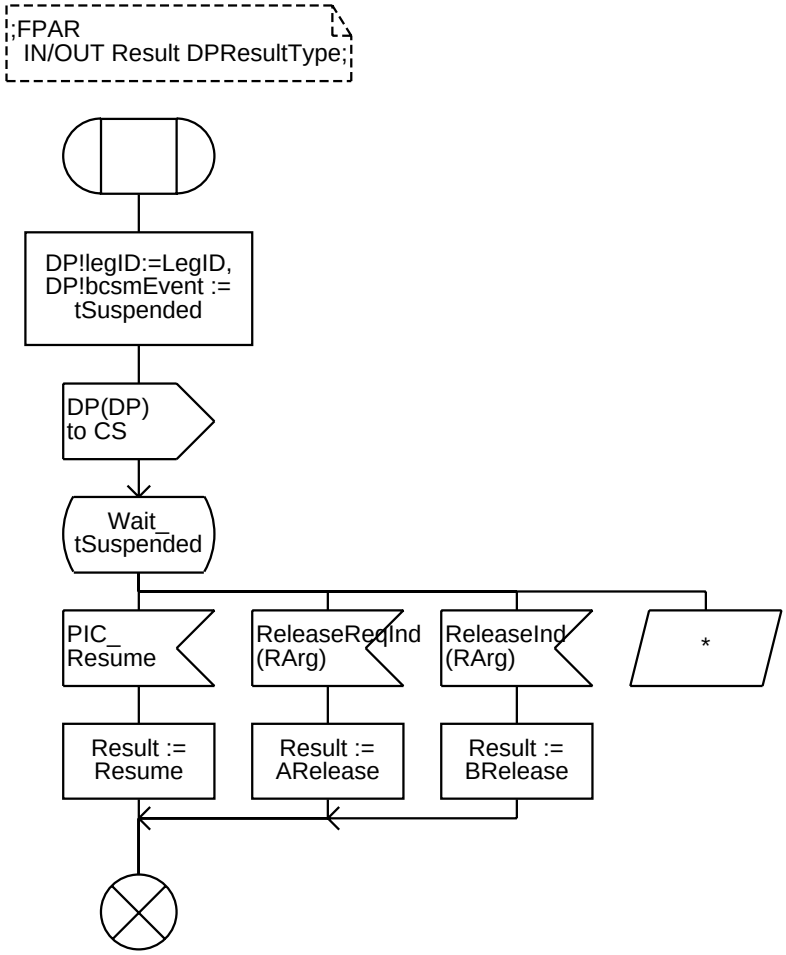

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
4(5)
Wait_B_
T_Disconnect
redefined
PIC
(PIC)
PIC!
pic
ReleaseCall
DisconnectLeg
SendSTUI
Else
PIC!party
remoteParty
localParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
rArg!cause :=
bPtyDisc
'Disconnect the
controlling leg.'
PIC!party
localParty
remoteParty
ReleaseReqInd
(RArg,remCSAID,
remLegID,receiverRef)
to CS
DataReq_
Ind(UDArg,remCSAID,
remlegID,receiverRef)
to CS
'Store info.'
'B party
initiated
clearing'
BCSMStop
(LegId,LocalParty)
to CS
Result :=
SCFRelease

Annex A:

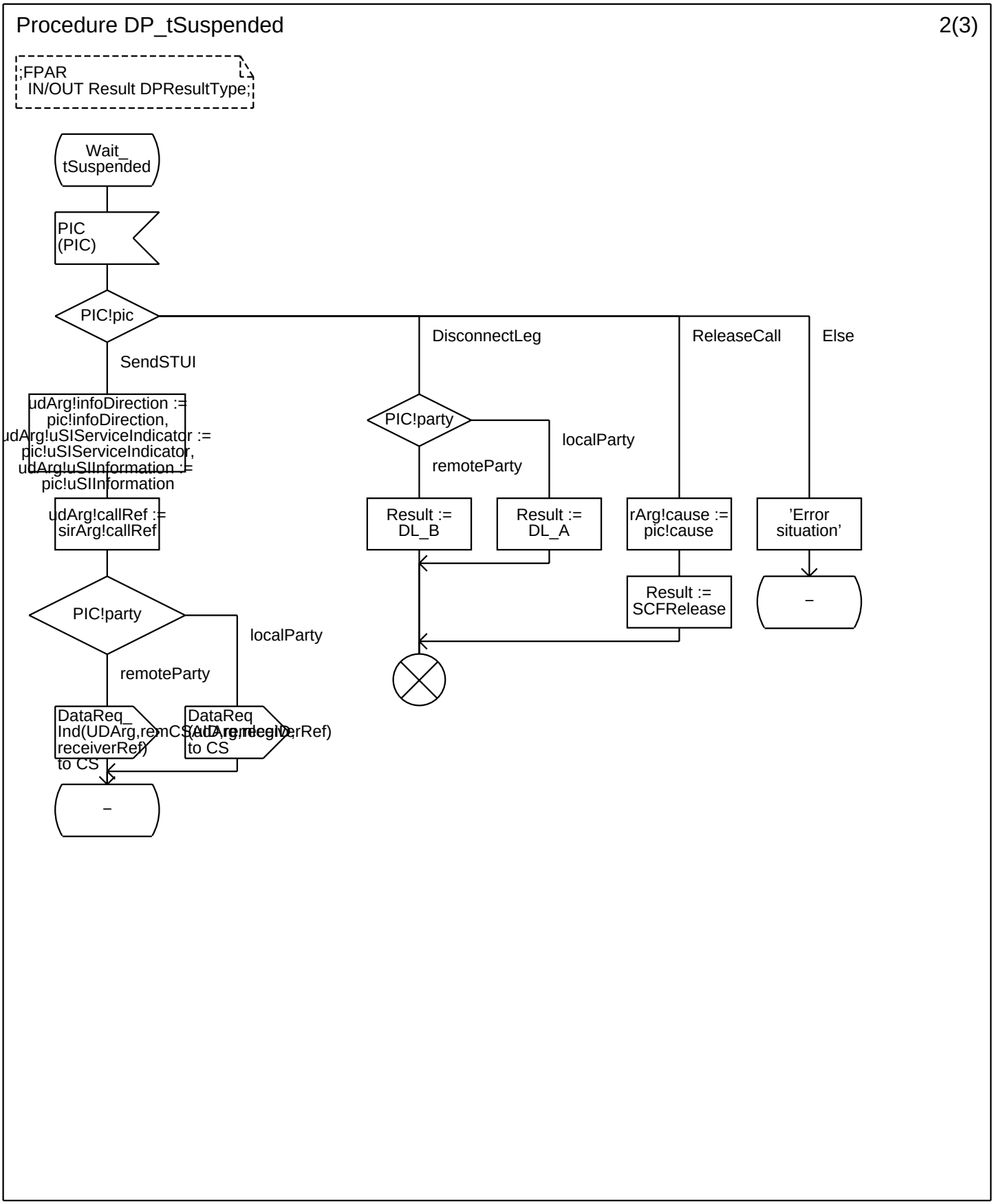

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
5(5)
Wait_A_
_T_Disconnect
redefined
PIC
(PIC)
/* Note: SendUTSI
cannot occur. */
PIC!
pic
Disconnect_
Leg
SendSTUI
Else
Release_
Call
PIC!party
localParty
remoteParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
rArg!cause :=
pic!cause
'Disconnect
the passive
leg'
–
PIC!party
localParty
remoteParty
rArg!callRef
:= sirArg!callRef
–
'Store info.'
DataReq
(udArg,receiverRef)
to CS
ReleaseReq
(rArg,receiverRef)
to CS
–
BCSMStop
(LegId,
RemoteParty)
to CS
Result :=
SCFRelease

Annex A:

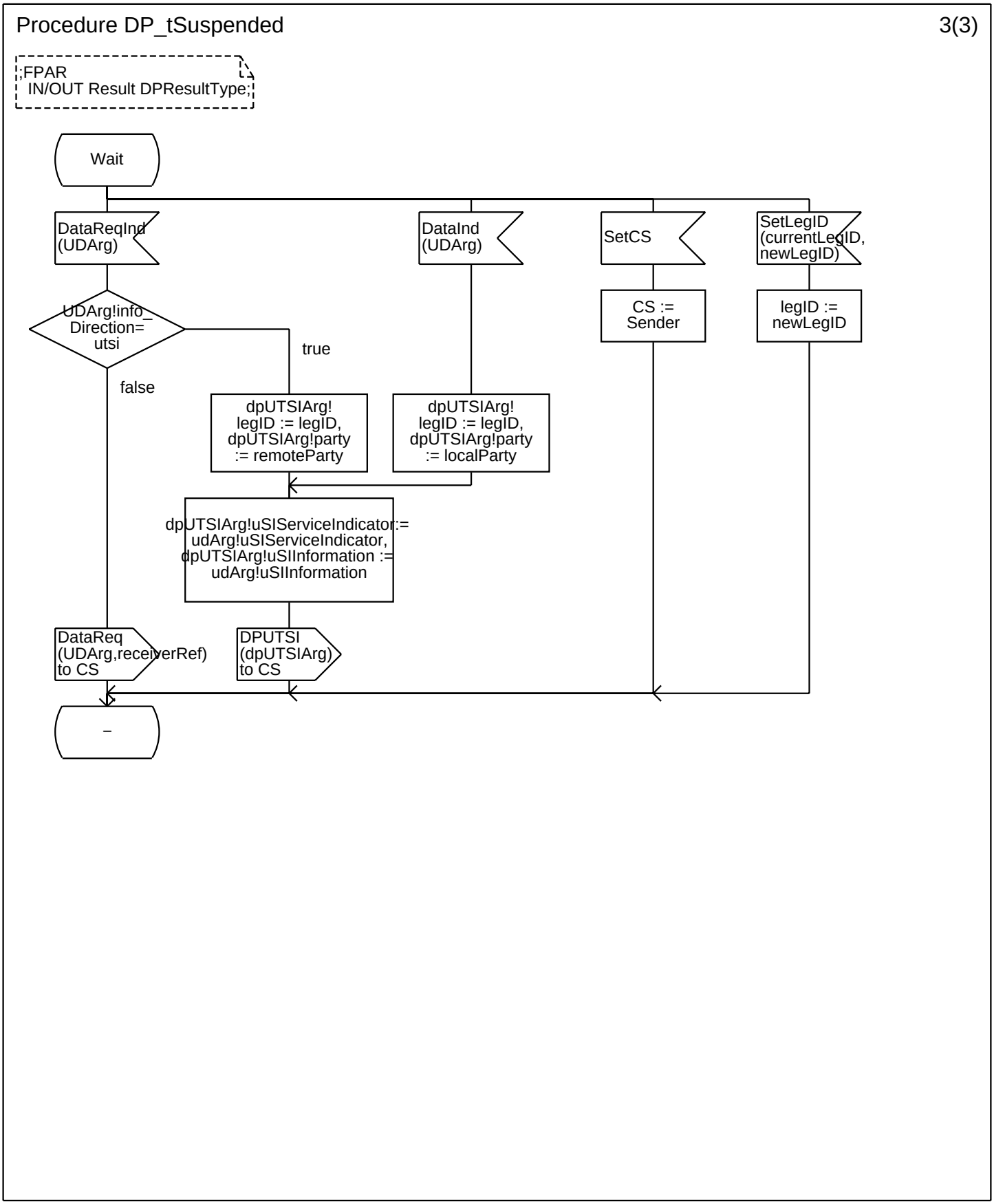
Procedure DP_tSuspended 1(3)



Annex A:

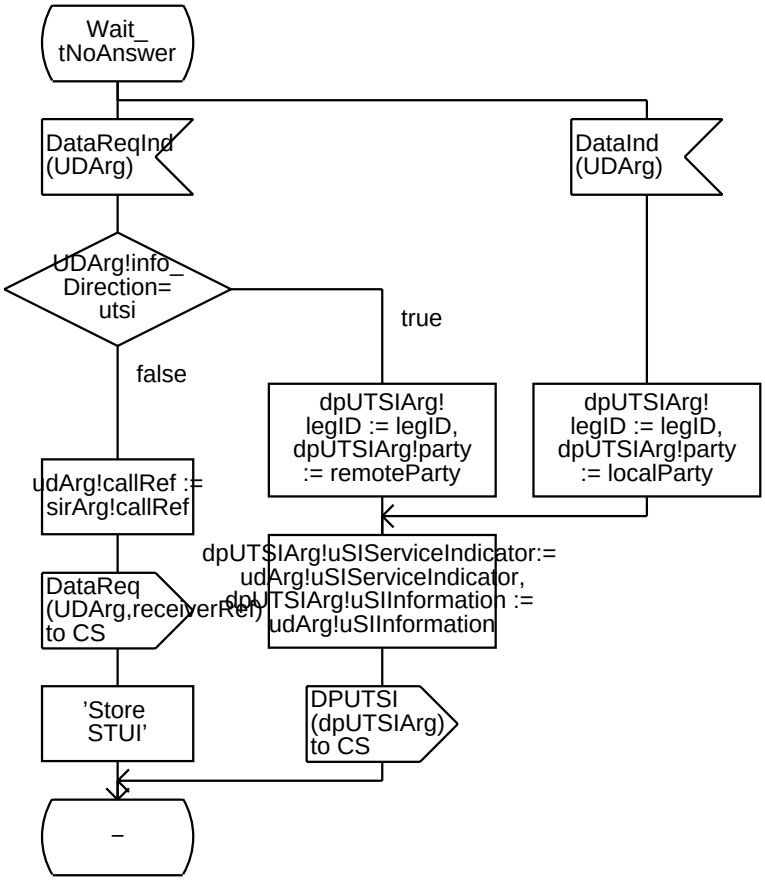

Procedure DP_tSuspended
2(3)
;FPAR
IN/OUT Result DPResultType;
Wait_
tSuspended
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
–
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAddressID,
receiverRef)
to CS
DataReq
(UDArg,remCSAddressID,
receiverRef)
to CS
–

Annex A:


Procedure DP_tSuspended
3(3)
;FPAR
IN/OUT Result DPResultType;
Wait
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

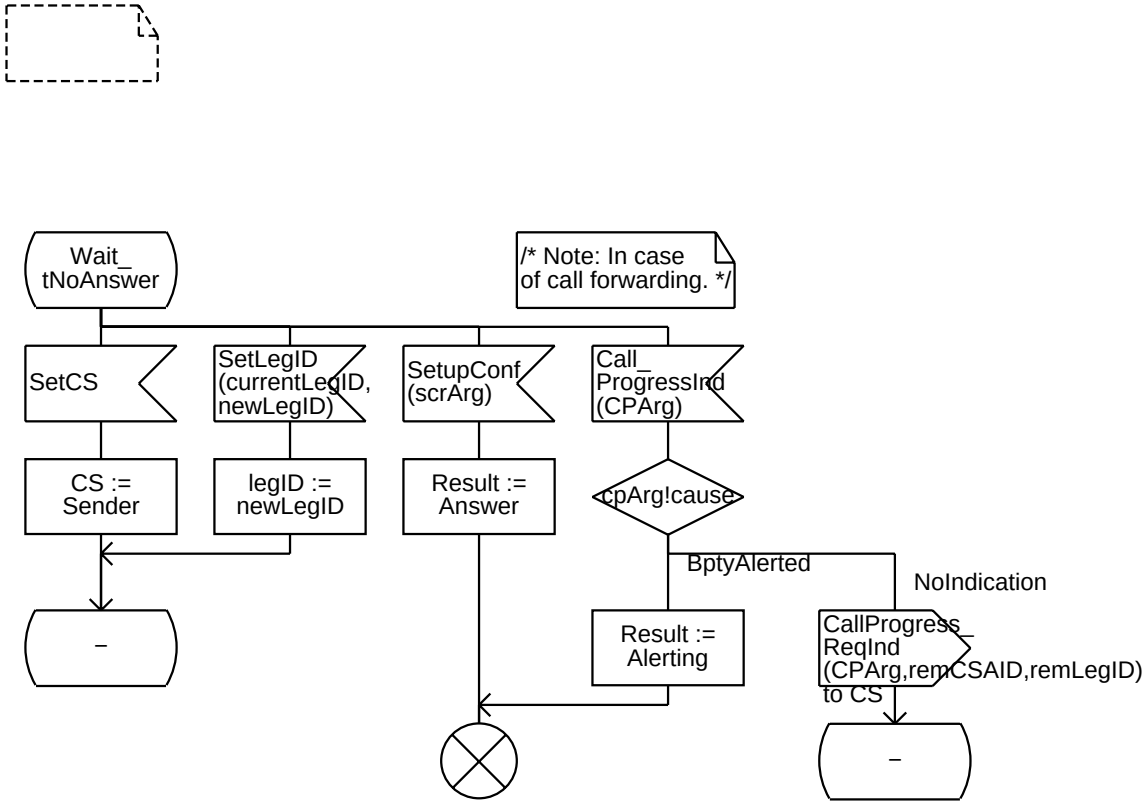
Annex A:

Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina 1(3)

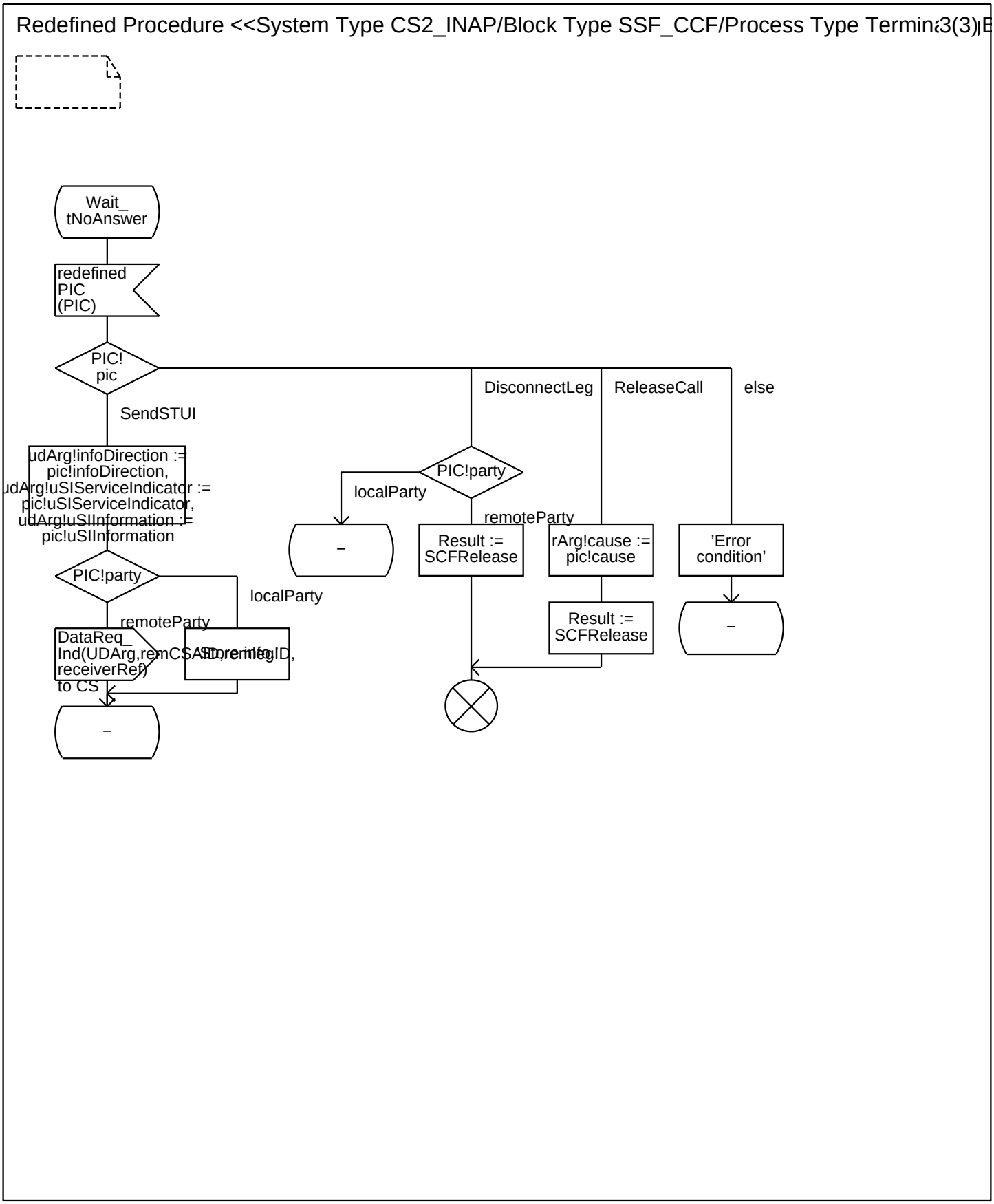


Annex A:

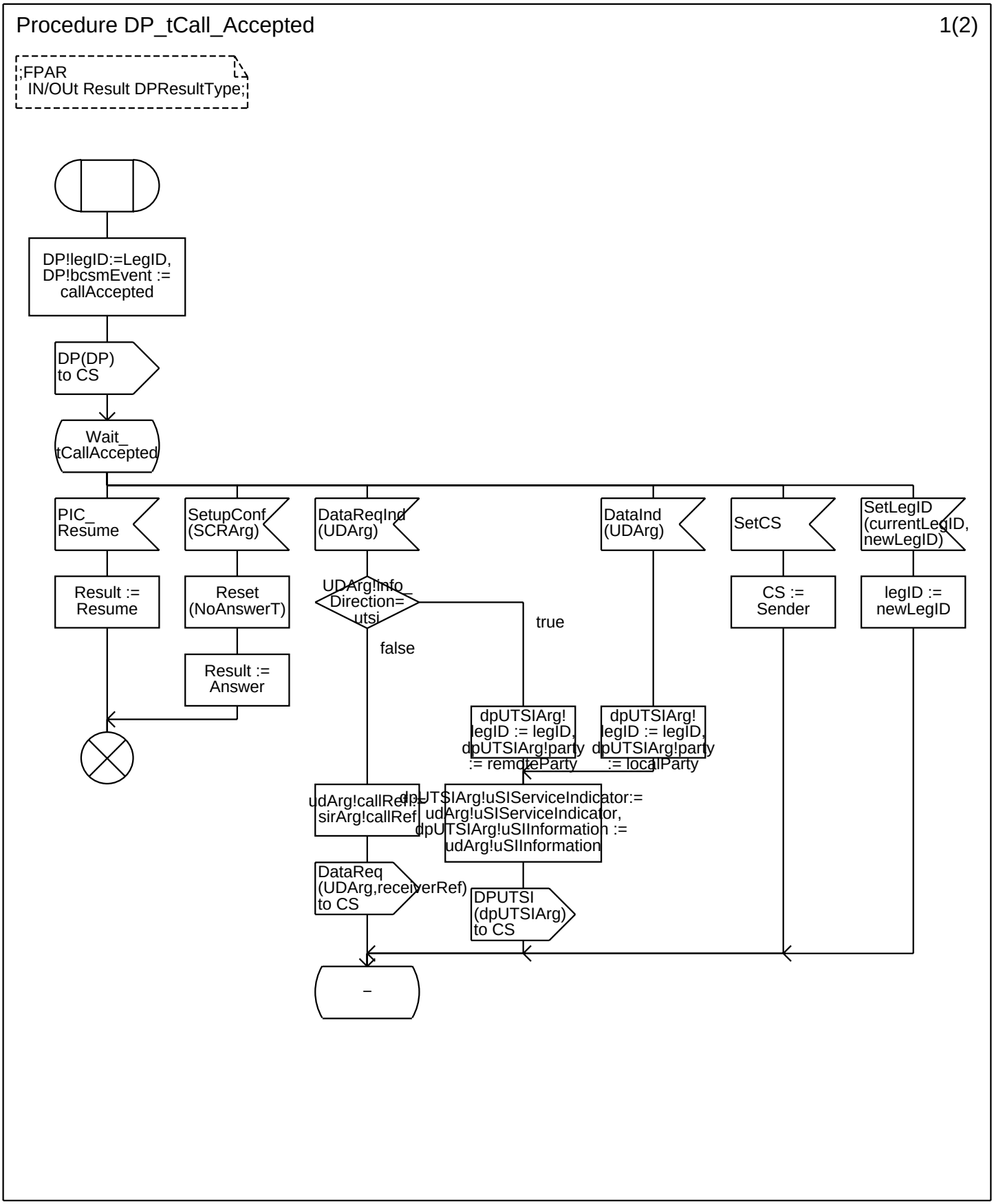
Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina2(3)E



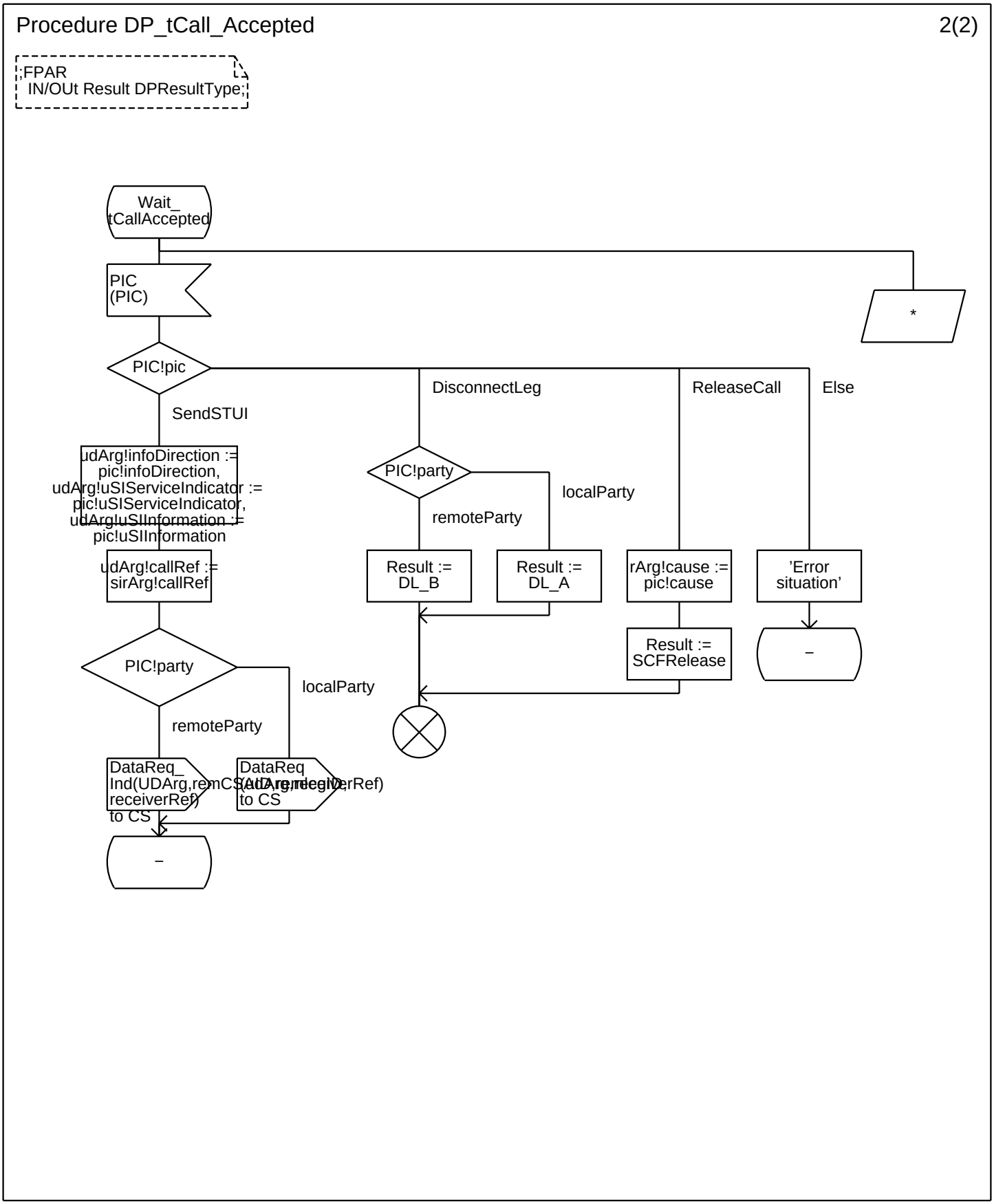
Annex A:


Redefined Procedure <<System Type CS2_INAP/Block Type SSF_CCF/Process Type Termina
3(3)
Wait_
tNoAnswer
redefined
PIC
(PIC)
PIC!
pic
SendSTUI
DisconnectLeg
ReleaseCall
else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
–
Result :=
SCFRelease
rArg!cause :=
pic!cause
'Error
condition'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remMsgID,
receiverRef)
to CS
Result :=
SCFRelease
–
–

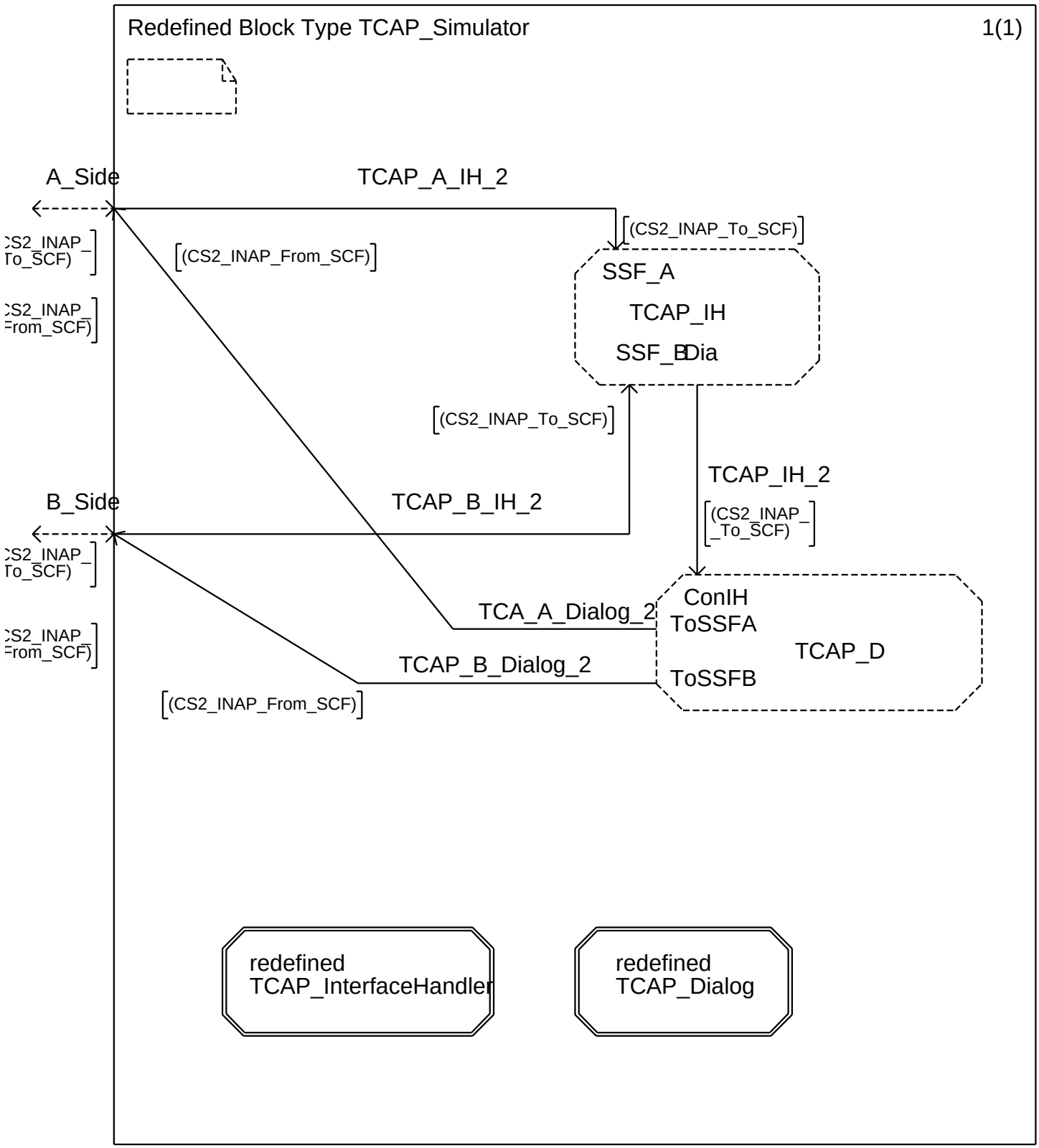
Annex A:


Procedure DP_tCall_Accepted
1(2)
;FPAR
IN/OUt Result DPResultType;
DP!legID:=LegID,
DP!bcsmEvent :=
callAccepted
DP(DP)
to CS
Wait_
tCallAccepted
PIC_
Resume
SetupConf
(SCRArg)
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
Result :=
Resume
Reset
(NoAnswerT)
UDArg!Info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
Result :=
Answer
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
udArg!callRef=
sirArg!callRef
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

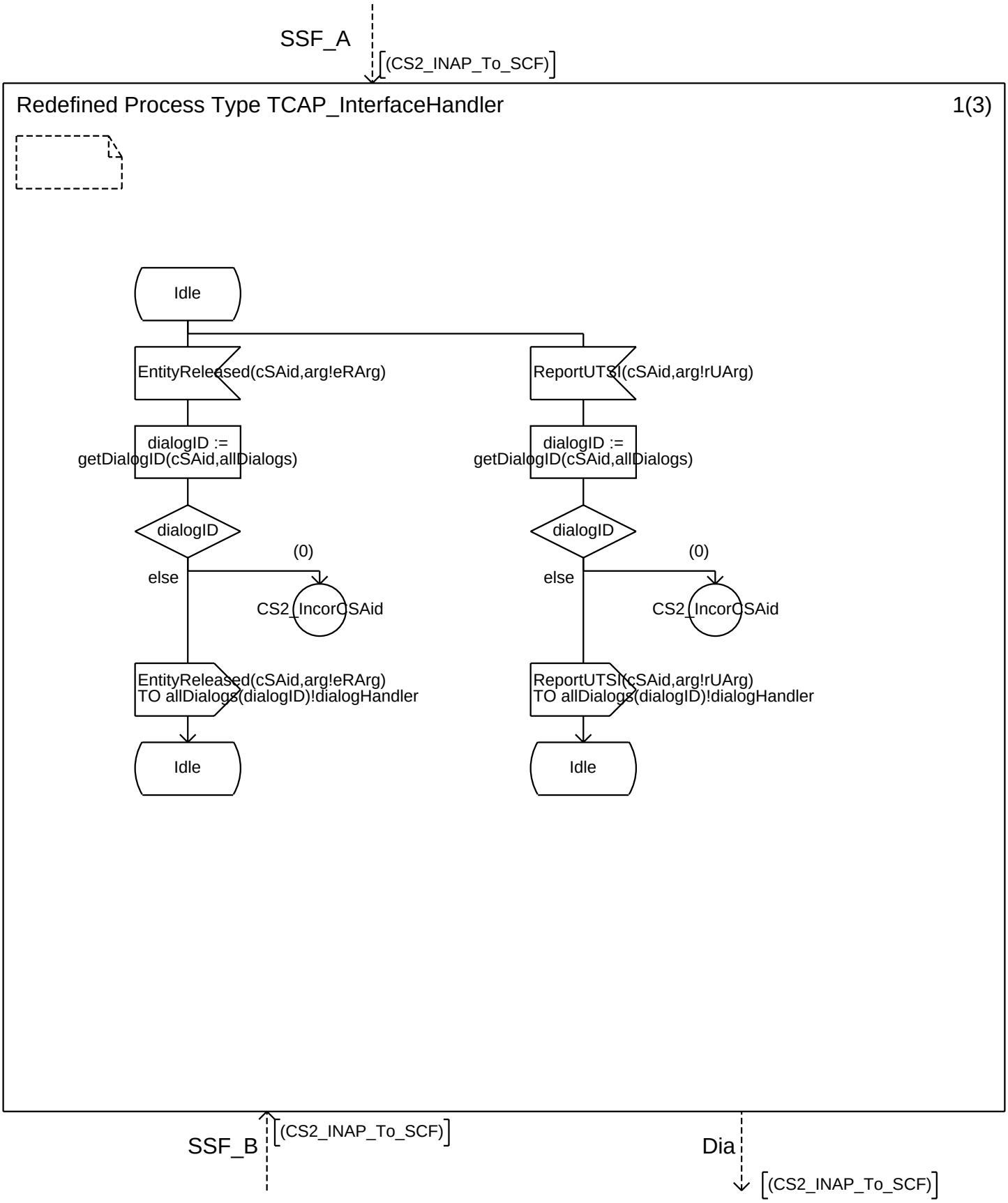
Annex A:


Procedure DP_tCall_Accepted
2(2)
;FPAR
IN/OUt Result DPResultType;
Wait_
tCallAccepted
PIC
(PIC)
*
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
–
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSReqId,
receiverRef)
to CS
DataReq
(UDArg,reqId,receiverRef)
to CS
–

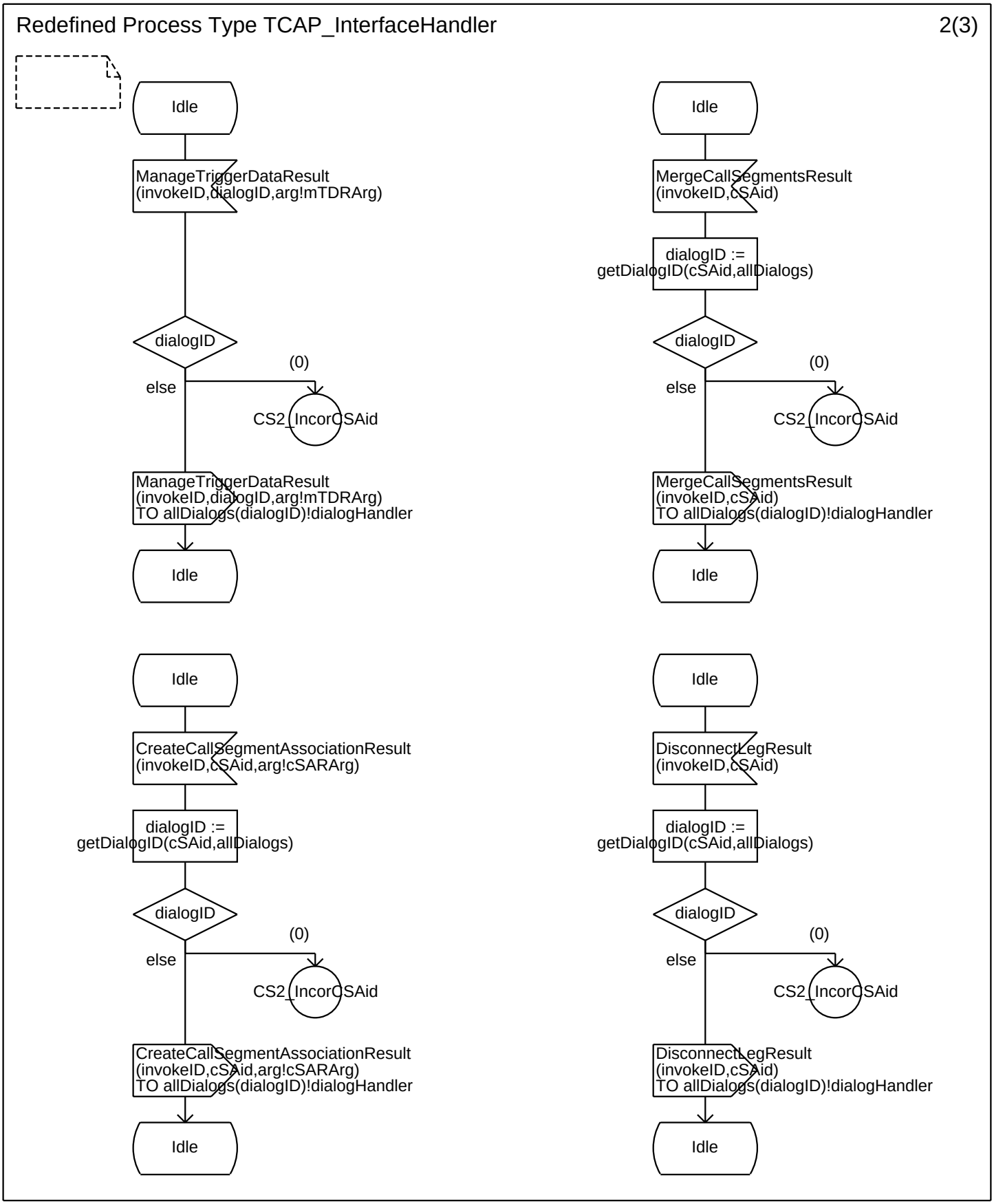
Annex A:


Redefined Block Type TCAP_Simulator
1(1)
A_Side
TCAP_A_IH_2
(CS2_INAP_To_SCF)
(CS2_INAP_From_SCF)
SSF_A
TCAP_IH
SSF_BDia
(CS2_INAP_To_SCF)
TCAP_IH_2
(CS2_INAP_To_SCF)
B_Side
TCAP_B_IH_2
ConIH
ToSSFA
TCA_A_Dialog_2
TCAP_D
TCAP_B_Dialog_2
ToSSFB
(CS2_INAP_From_SCF)
redefined
TCAP_InterfaceHandler
redefined
TCAP_Dialog

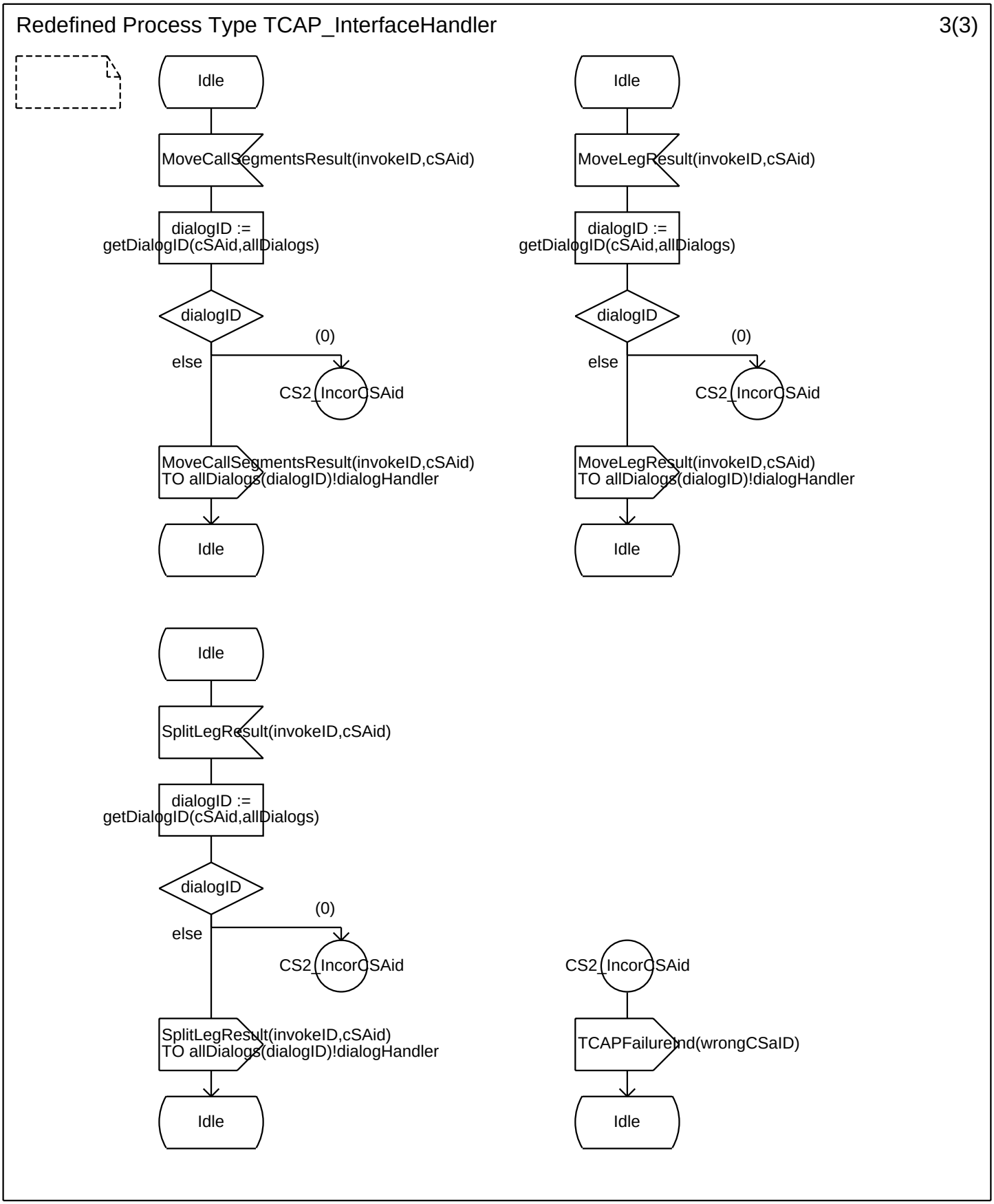
Annex A:


SSF_A
[(CS2_INAP_To_SCF)]
Redefined Process Type TCAP_InterfaceHandler
1(3)
Idle
EntityReleased(cSAid,arg!eRArg)
ReportUTSI(cSAid,arg!rUArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
dialogID
(0)
(0)
else
else
CS2_IncorCSAid
CS2_IncorCSAid
EntityReleased(cSAid,arg!eRArg)
TO allDialogs(dialogID)!dialogHandler
ReportUTSI(cSAid,arg!rUArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
SSF_B
[(CS2_INAP_To_SCF)]
Dia
[(CS2_INAP_To_SCF)]

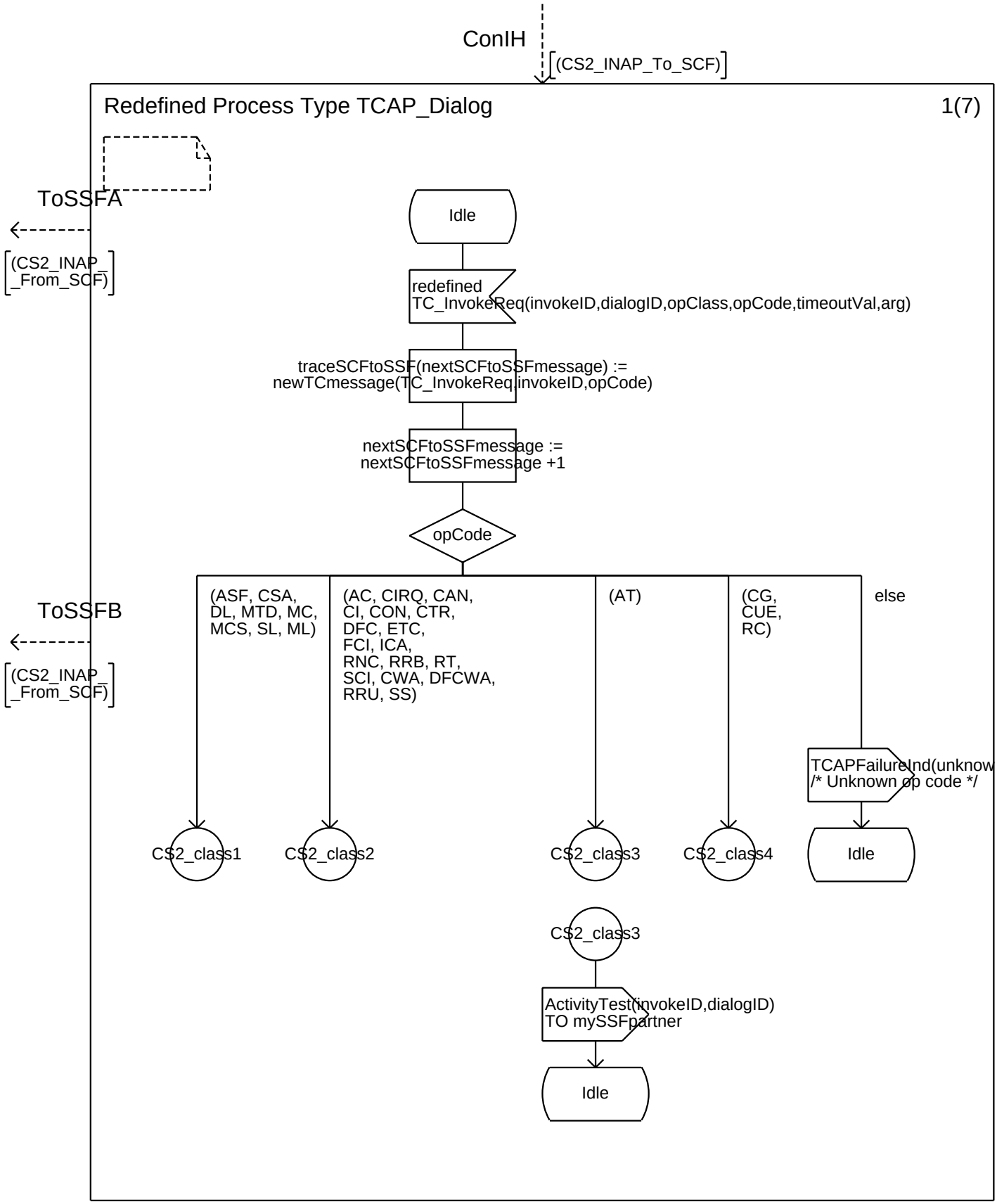
Annex A:


Redefined Process Type TCAP_InterfaceHandler
2(3)
Idle
ManageTriggerDataResult
(invokeID,dialogID,arg!mTDRArg)
dialogID
(0)
else
CS2_IncorCSAid
ManageTriggerDataResult
(invokeID,dialogID,arg!mTDRArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
MergeCallSegmentsResult
(invokeID,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
MergeCallSegmentsResult
(invokeID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
CreateCallSegmentAssociationResult
(invokeID,cSAid,arg!cSARArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
CreateCallSegmentAssociationResult
(invokeID,cSAid,arg!cSARArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
DisconnectLegResult
(invokeID,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
DisconnectLegResult
(invokeID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle

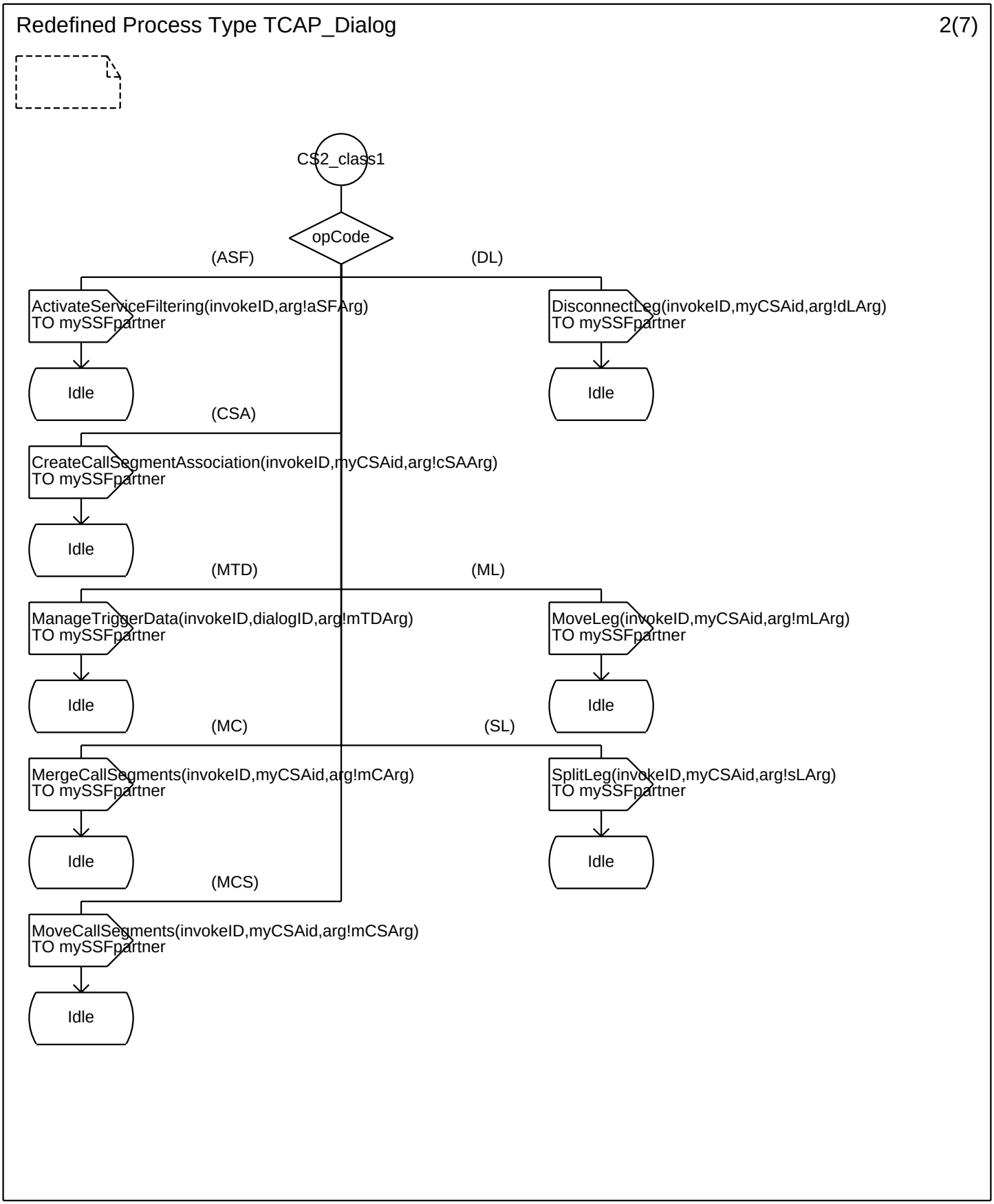
Annex A:


Redefined Process Type TCAP_InterfaceHandler
3(3)
Idle
MoveCallSegmentsResult(invokeID,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
MoveCallSegmentsResult(invokeID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
MoveLegResult(invokeID,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
MoveLegResult(invokeID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
SplitLegResult(invokeID,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
SplitLegResult(invokeID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle
CS2_IncorCSAid
TCAPFailureInd(wrongCSaID)
Idle

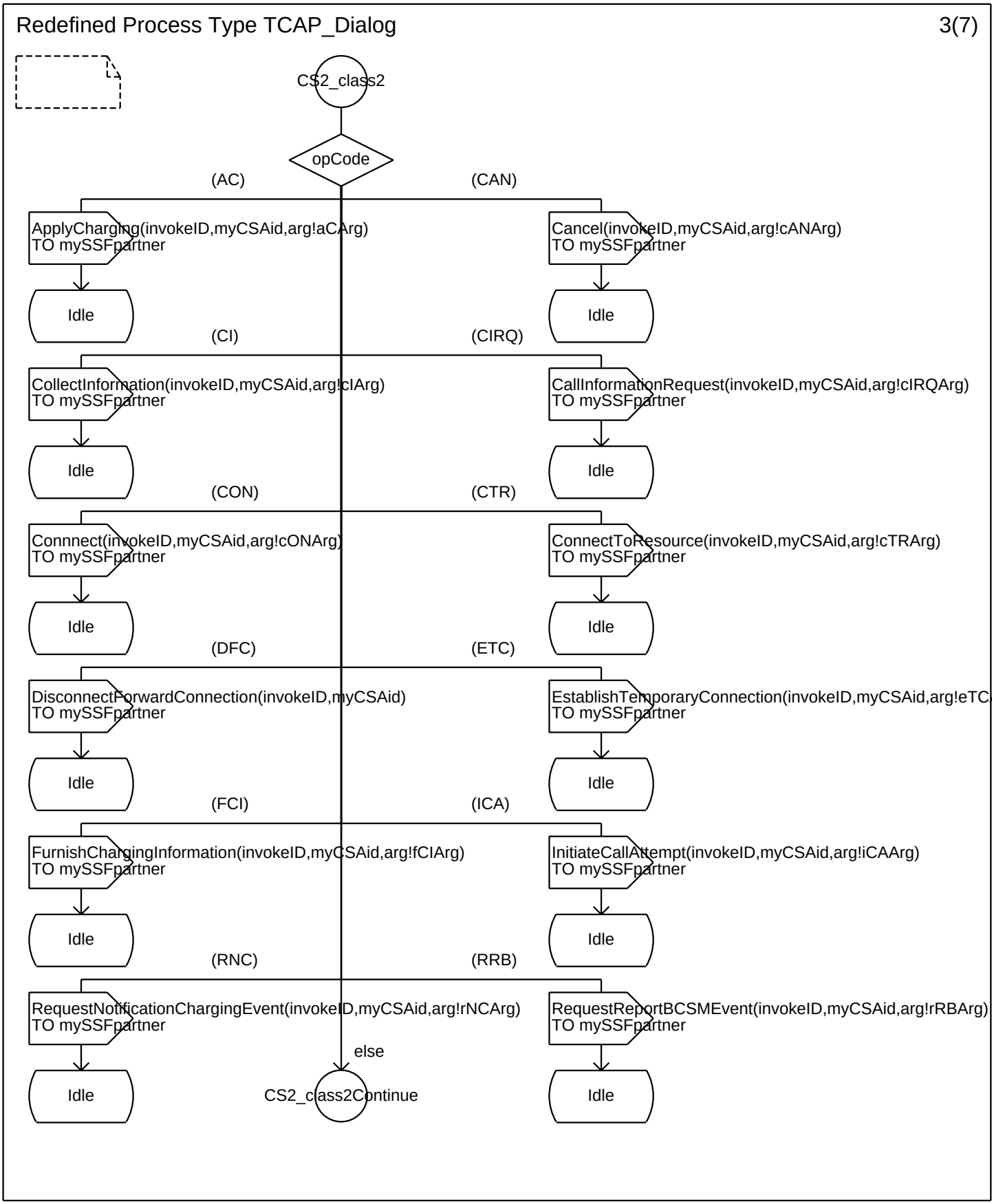
Annex A:


ConIH
(CS2_INAP_To_SCF)
Redefined Process Type TCAP_Dialog
1(7)
ToSSFA
(CS2_INAP_
_From_SCF)
Idle
redefined
TC_InvokeReq(invokeID,dialogID,opClass,opCode,timeoutVal,arg)
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_InvokeReq,invokeID,opCode)
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
opCode
ToSSFB
(CS2_INAP_
_From_SCF)
(ASF, CSA,
DL, MTD, MC,
MCS, SL, ML)
(AC, CIRQ, CAN,
CI, CON, CTR,
DFC, ETC,
FCI, ICA,
RNC, RRB, RT,
SCI, CWA, DFCWA,
RRU, SS)
(AT)
(CG,
CUE,
RC)
else
TCAPFailureInd(unknow
/* Unknown op code */
CS2_class1
CS2_class2
CS2_class3
CS2_class4
Idle
CS2_class3
ActivityTest(invokeID,dialogID)
TO mySSFpartner
Idle

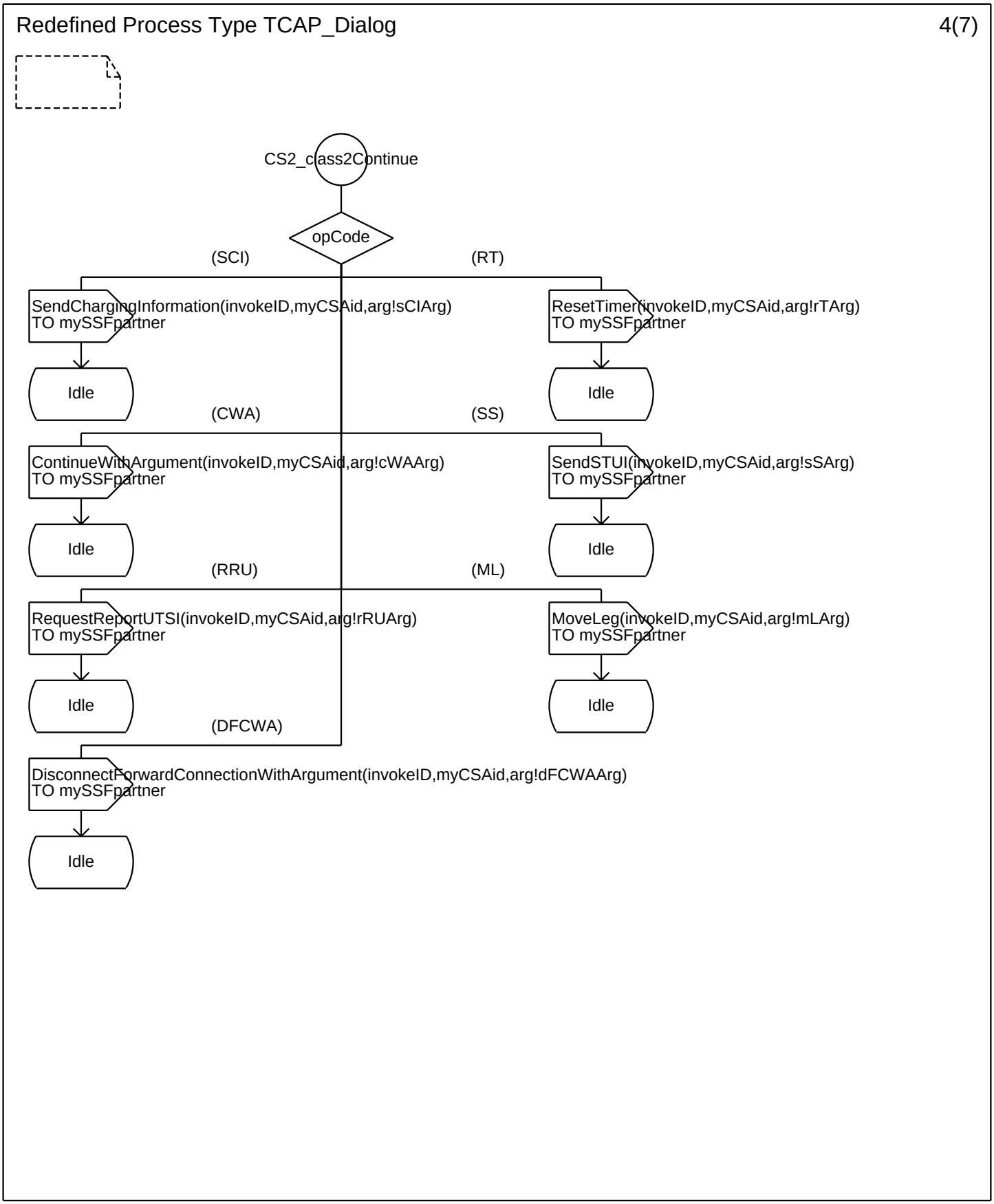
Annex A:


Redefined Process Type TCAP_Dialog
2(7)
CS2_class1
opCode
(ASF)
(DL)
ActivateServiceFiltering(invokeID,arg!aSFArg)
TO mySSFpartner
DisconnectLeg(invokeID,myCSAid,arg!dLArg)
TO mySSFpartner
Idle
Idle
(CSA)
CreateCallSegmentAssociation(invokeID,myCSAid,arg!cSAArg)
TO mySSFpartner
Idle
(MTD)
(ML)
ManageTriggerData(invokeID,dialogID,arg!mTDArg)
TO mySSFpartner
MoveLeg(invokeID,myCSAid,arg!mLArg)
TO mySSFpartner
Idle
Idle
(MC)
(SL)
MergeCallSegments(invokeID,myCSAid,arg!mCArg)
TO mySSFpartner
SplitLeg(invokeID,myCSAid,arg!sLArg)
TO mySSFpartner
Idle
Idle
(MCS)
MoveCallSegments(invokeID,myCSAid,arg!mCSArg)
TO mySSFpartner
Idle

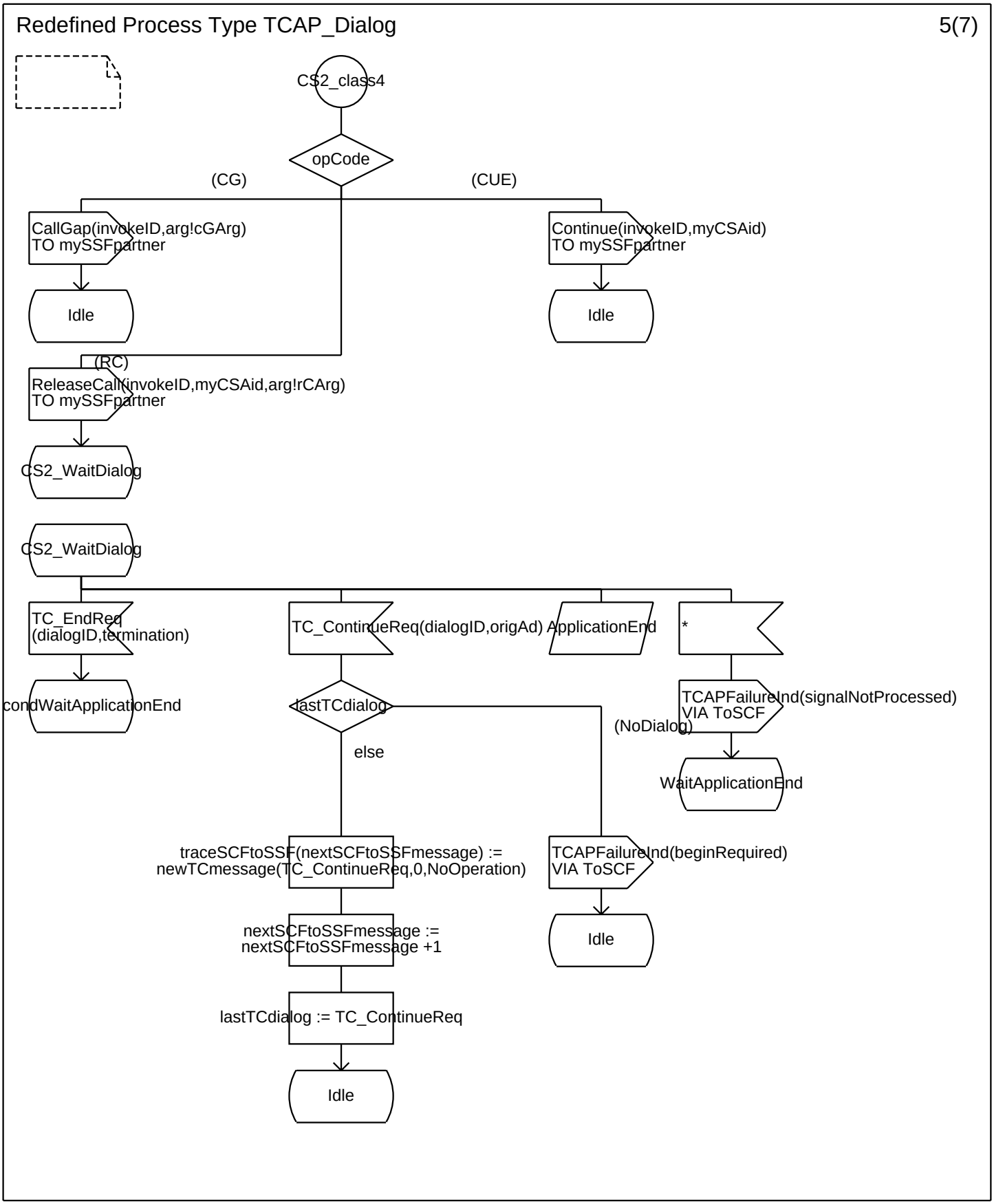
Annex A:


Redefined Process Type TCAP_Dialog
3(7)
CS2_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!eTC
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
CS2_class2Continue
Idle

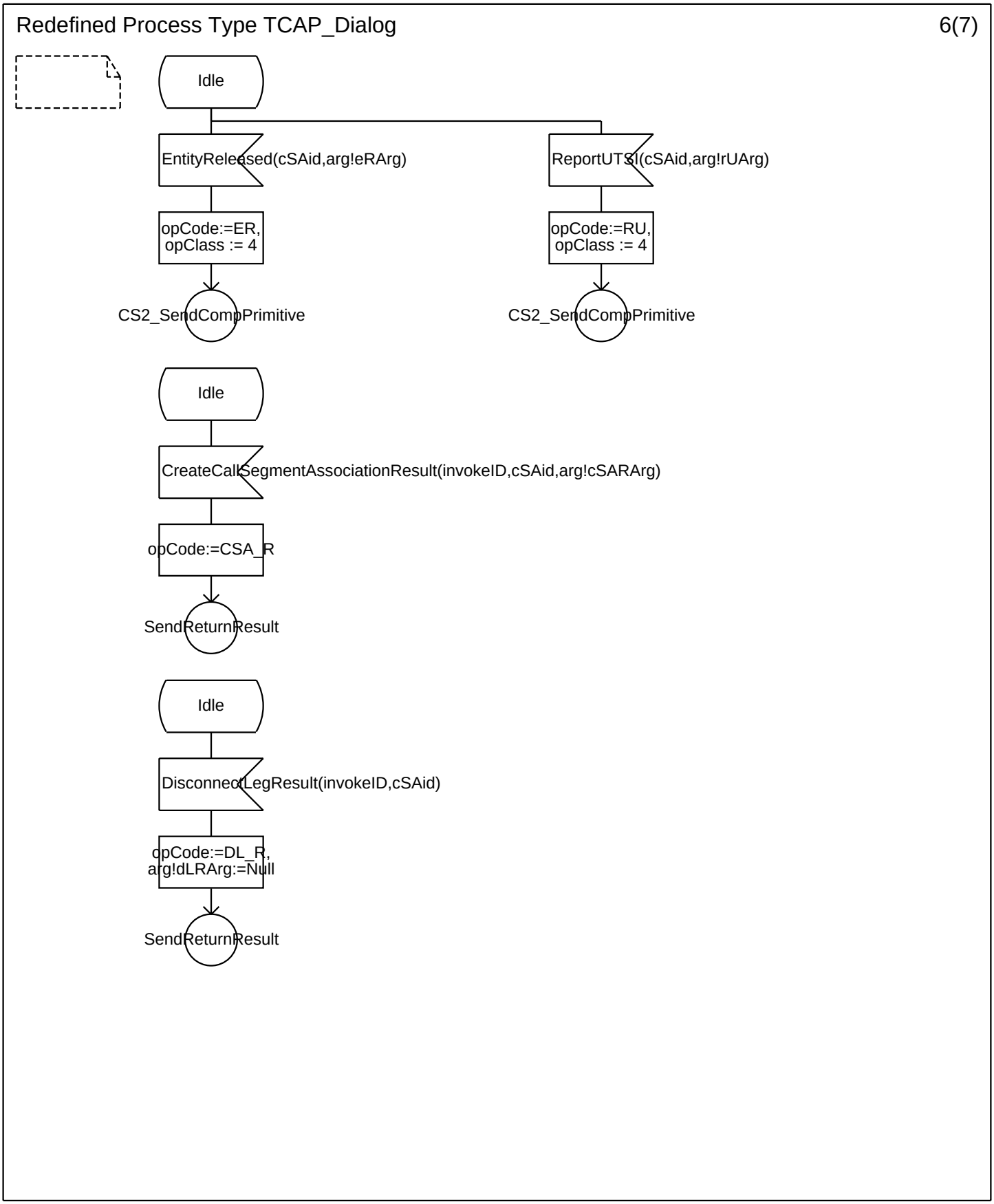
Annex A:


Redefined Process Type TCAP_Dialog
4(7)
CS2_class2Continue
opCode
(SCI)
(RT)
SendChargingInformation(invokeID,myCSAid,arg!sCIArg)
TO mySSFpartner
ResetTimer(invokeID,myCSAid,arg!rTArg)
TO mySSFpartner
Idle
Idle
(CWA)
(SS)
ContinueWithArgument(invokeID,myCSAid,arg!cWAArg)
TO mySSFpartner
SendSTUI(invokeID,myCSAid,arg!sSArg)
TO mySSFpartner
Idle
Idle
(RRU)
(ML)
RequestReportUTSI(invokeID,myCSAid,arg!rRUArg)
TO mySSFpartner
MoveLeg(invokeID,myCSAid,arg!mLArg)
TO mySSFpartner
Idle
Idle
(DFCWA)
DisconnectForwardConnectionWithArgument(invokeID,myCSAid,arg!dFCWAArg)
TO mySSFpartner
Idle

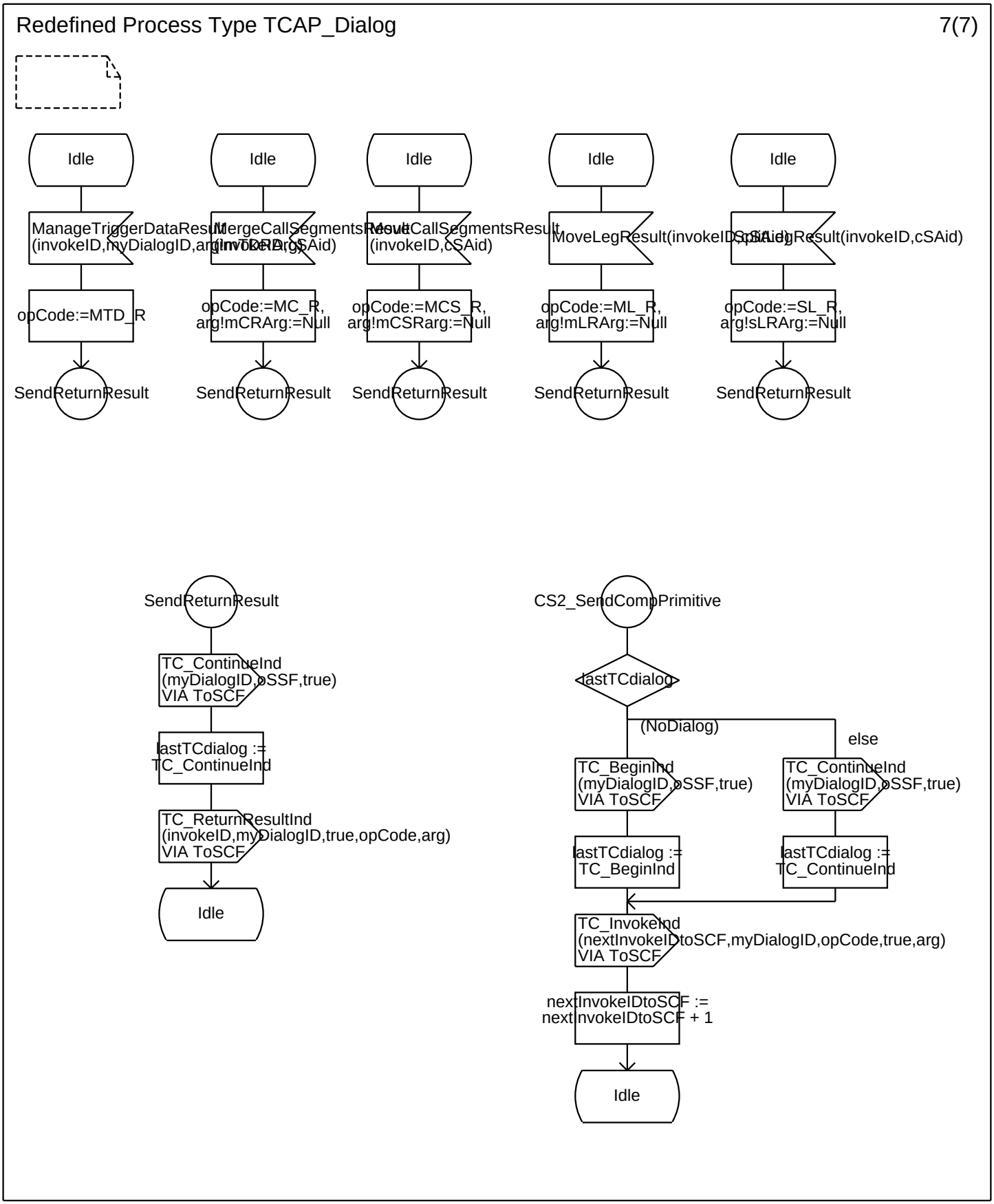
Annex A:


Redefined Process Type TCAP_Dialog
5(7)
CS2_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
CS2_WaitDialog
CS2_WaitDialog
TC_EndReq
(dialogID,termination)
TC_ContinueReq(dialogID,origAd)
ApplicationEnd
*
condWaitApplicationEnd
lastTCdialog
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
(NoDialog)
else
WaitApplicationEnd
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_ContinueReq,0,NoOperation)
TCAPFailureInd(beginRequired)
VIA ToSCF
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
Idle
lastTCdialog := TC_ContinueReq
Idle

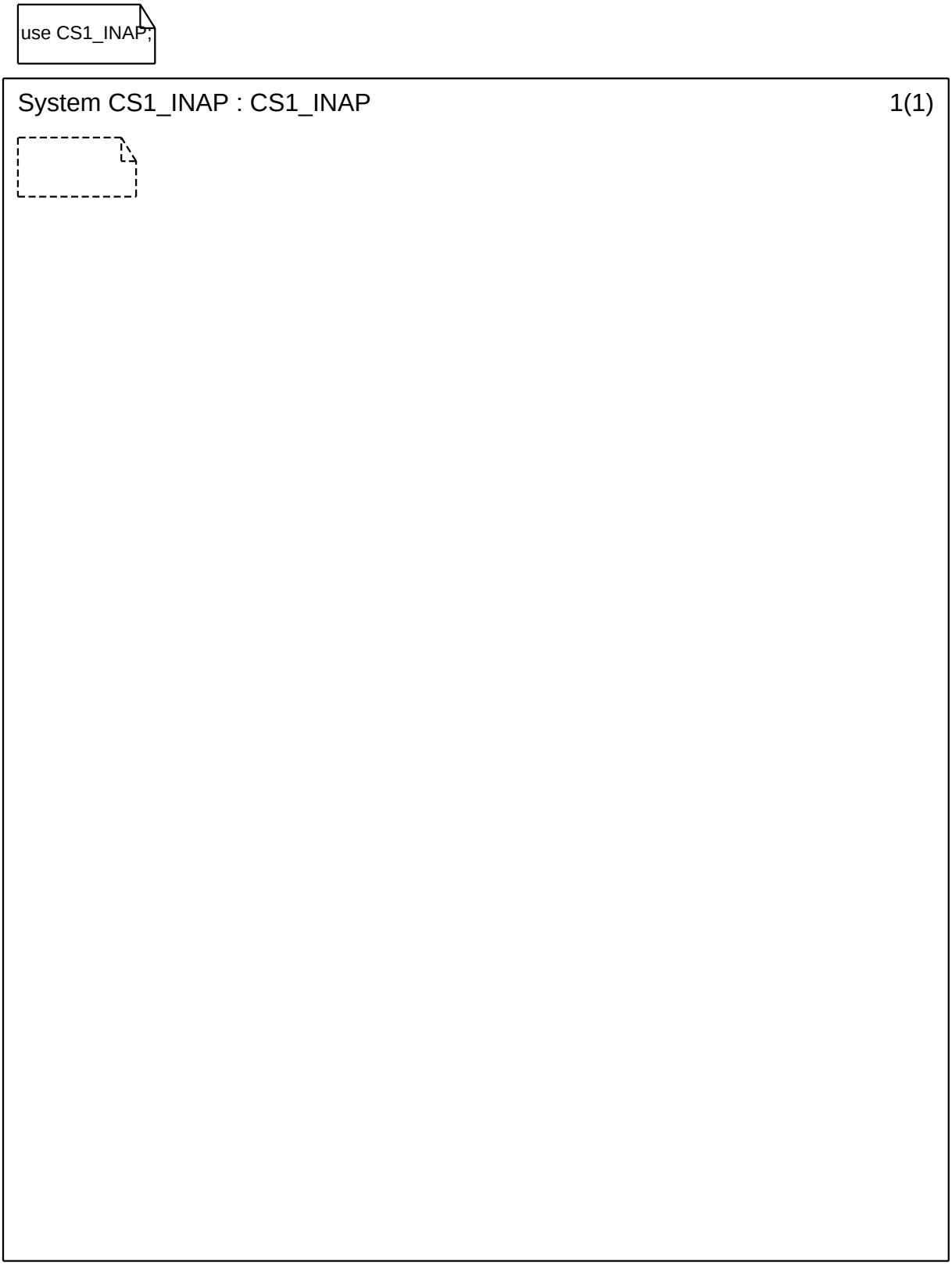
Annex A:


Redefined Process Type TCAP_Dialog
6(7)
Idle
EntityReleased(cSAid,arg!eRArg)
ReportUTSI(cSAid,arg!rUArg)
opCode:=ER,
opClass := 4
opCode:=RU,
opClass := 4
CS2_SendCompPrimitive
CS2_SendCompPrimitive
Idle
CreateCallSegmentAssociationResult(invokeID,cSAid,arg!cSARArg)
opCode:=CSA_R
SendReturnResult
Idle
DisconnectLegResult(invokeID,cSAid)
opCode:=DL_R,
arg!dLRArg:=Null
SendReturnResult

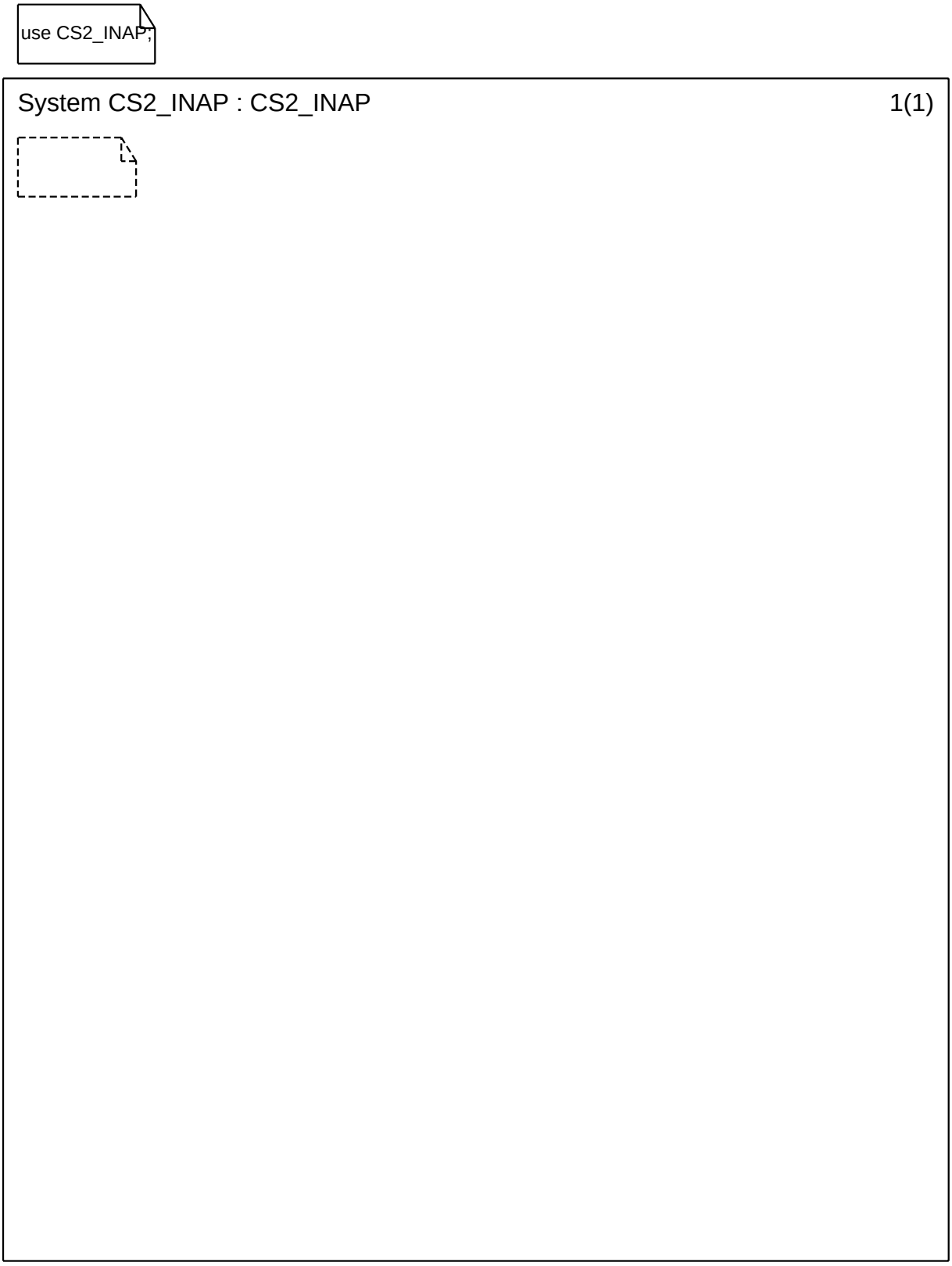
Annex A:


Redefined Process Type TCAP_Dialog
7(7)
Idle
ManageTriggerDataResult
(invokeID,myDialogID,arg,cSAid)
opCode:=MTD_R
SendReturnResult
Idle
MergeCallSegmentsResult
(invokeID,cSAid)
opCode:=MC_R,
arg!mCRArg:=Null
SendReturnResult
Idle
MoveCallSegmentsResult
(invokeID,cSAid)
opCode:=MCS_R,
arg!mCSRarg:=Null
SendReturnResult
Idle
MoveLegResult(invokeID,cSAid)
opCode:=ML_R,
arg!mLRArg:=Null
SendReturnResult
Idle
SplitLegResult(invokeID,cSAid)
opCode:=SL_R,
arg!sLRArg:=Null
SendReturnResult
SendReturnResult
TC_ContinueInd
(myDialogID,oSSF,true)
VIA ToSCF
lastTCdialog :=
TC_ContinueInd
TC_ReturnResultInd
(invokeID,myDialogID,true,opCode,arg)
VIA ToSCF
Idle
CS2_SendCompPrimitive
lastTCdialog
(NoDialog)
else
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

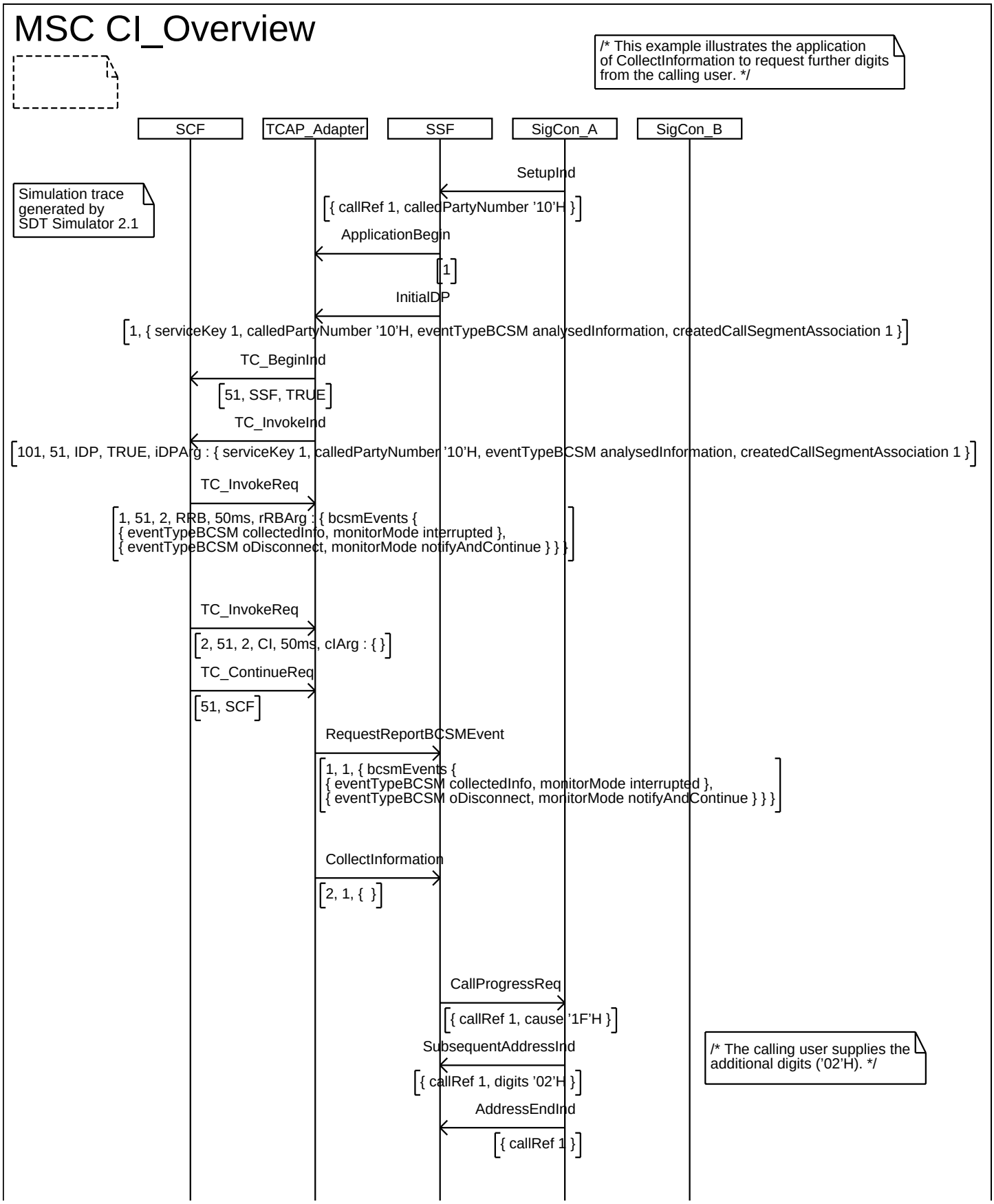
Annex A:



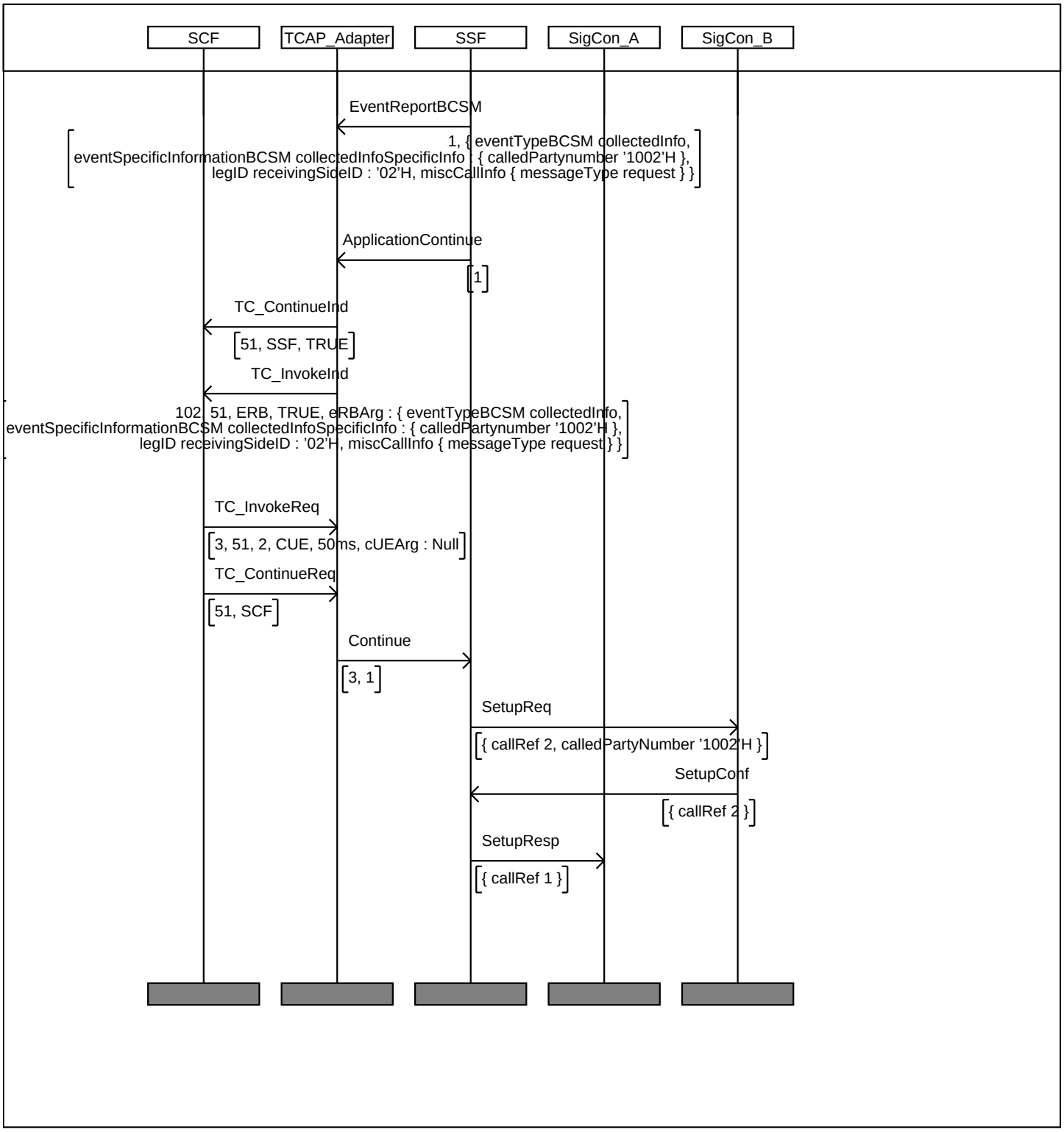
Annex A:



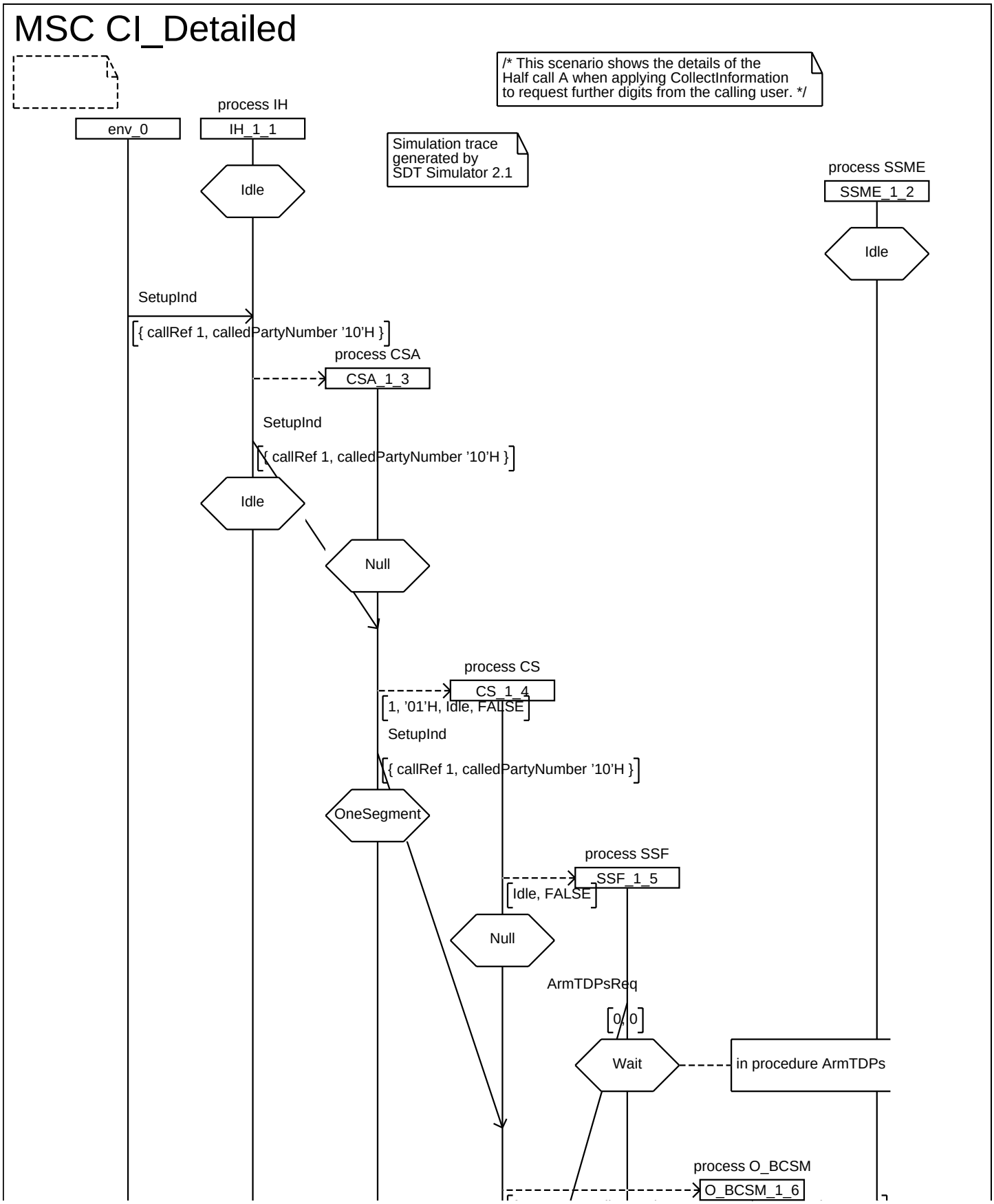
Annex A: CI_Overview



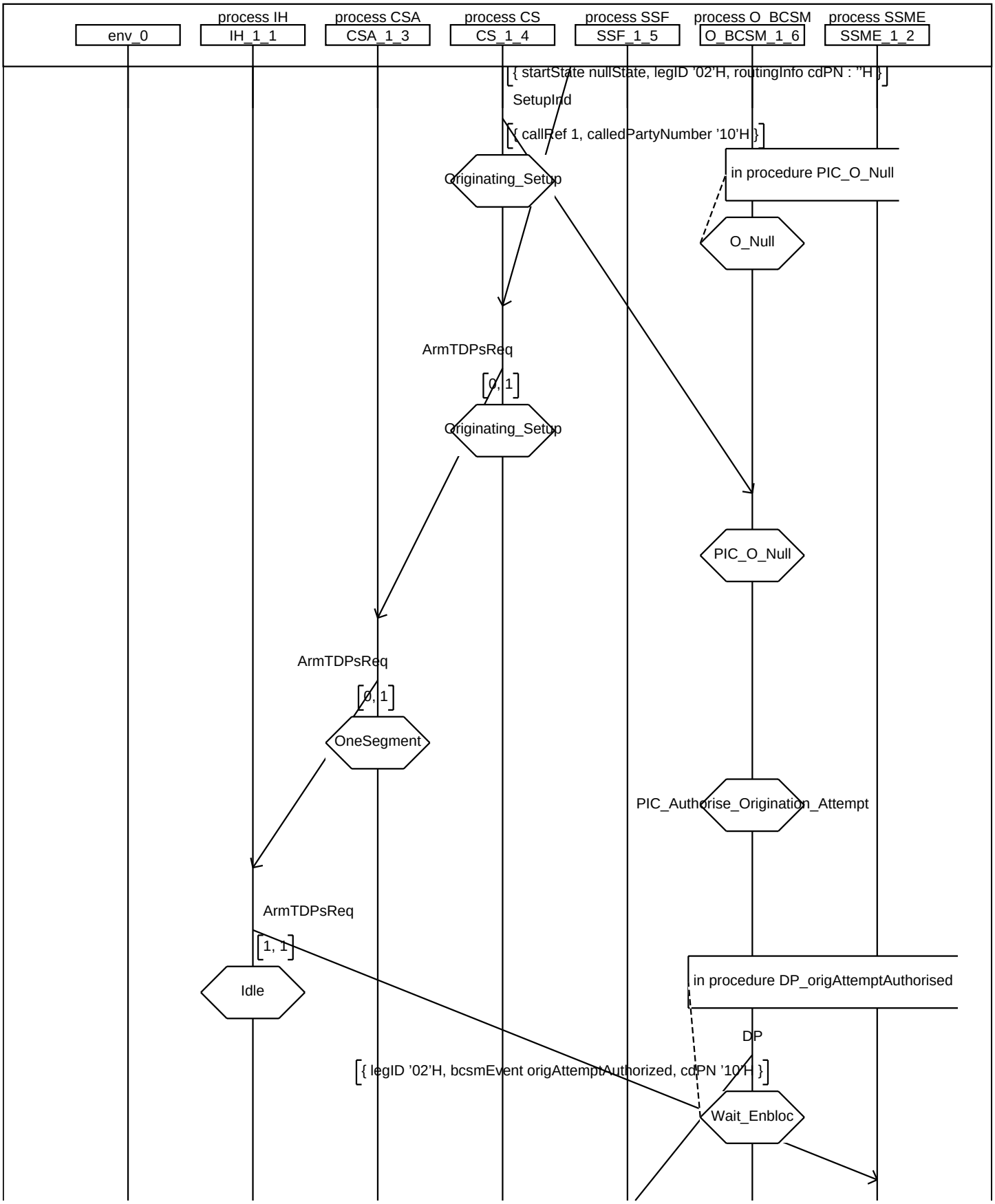
Annex A: CI_Overview



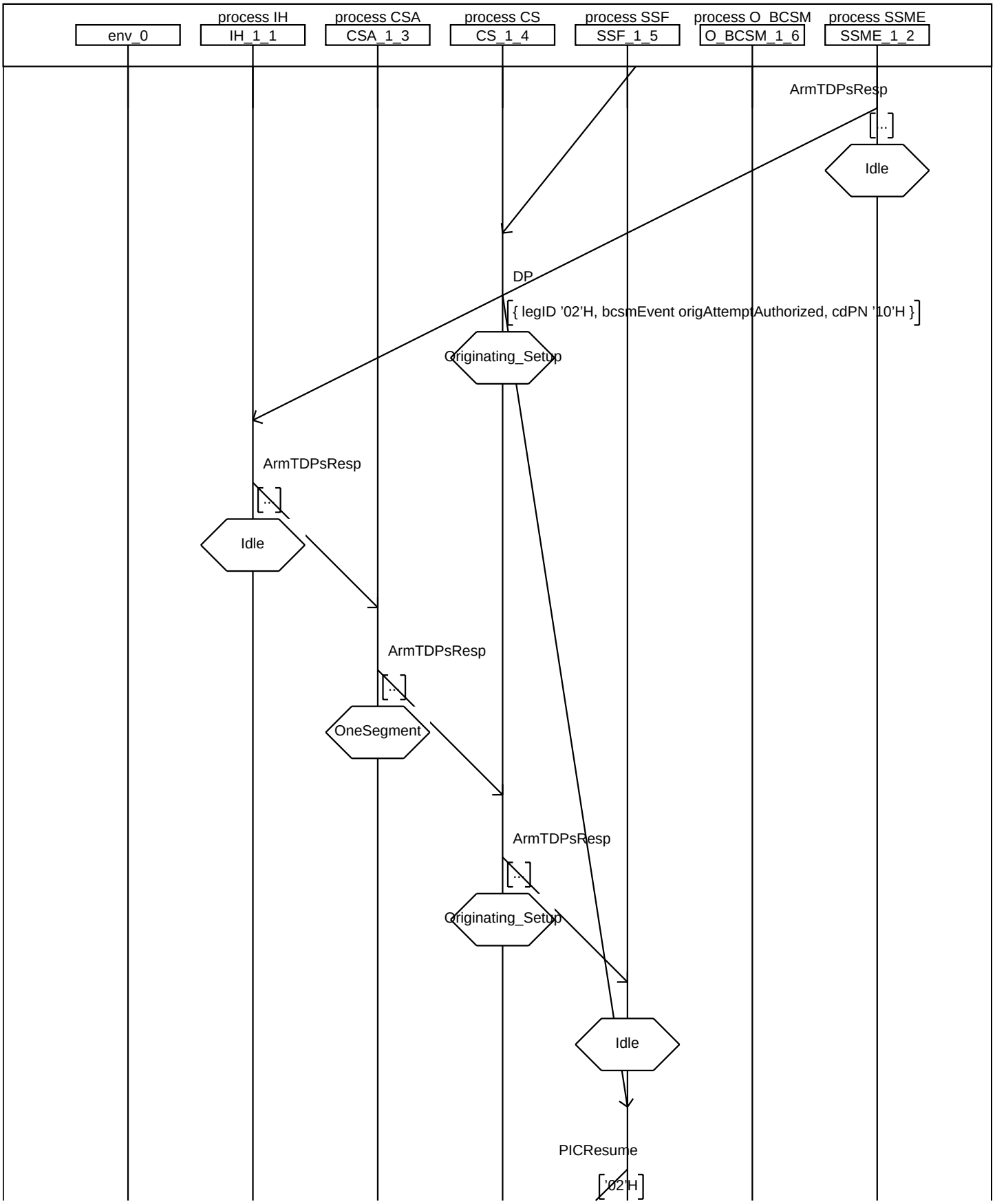
Annex A: CI_Detailed



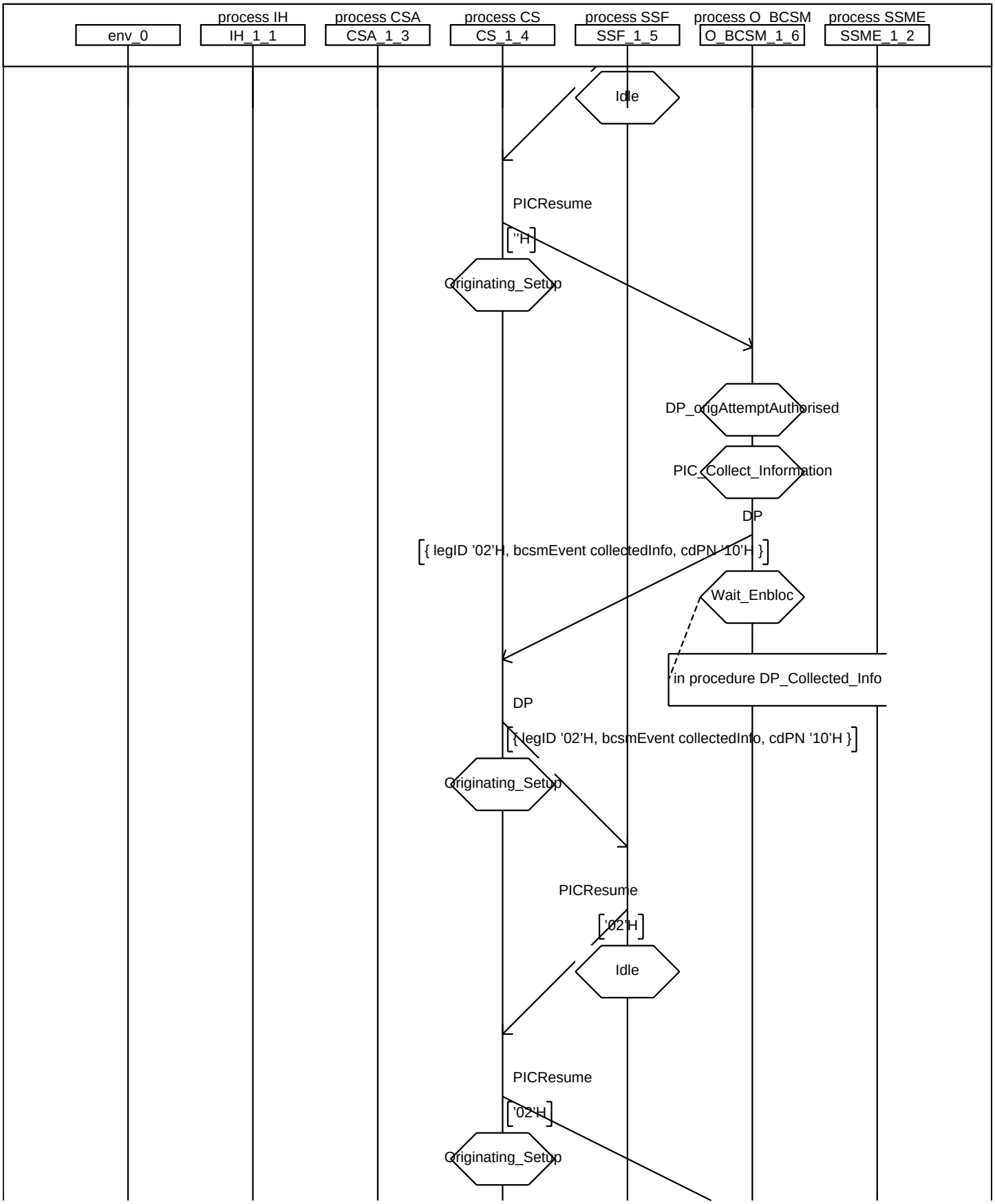
Annex A: CI_Detailed



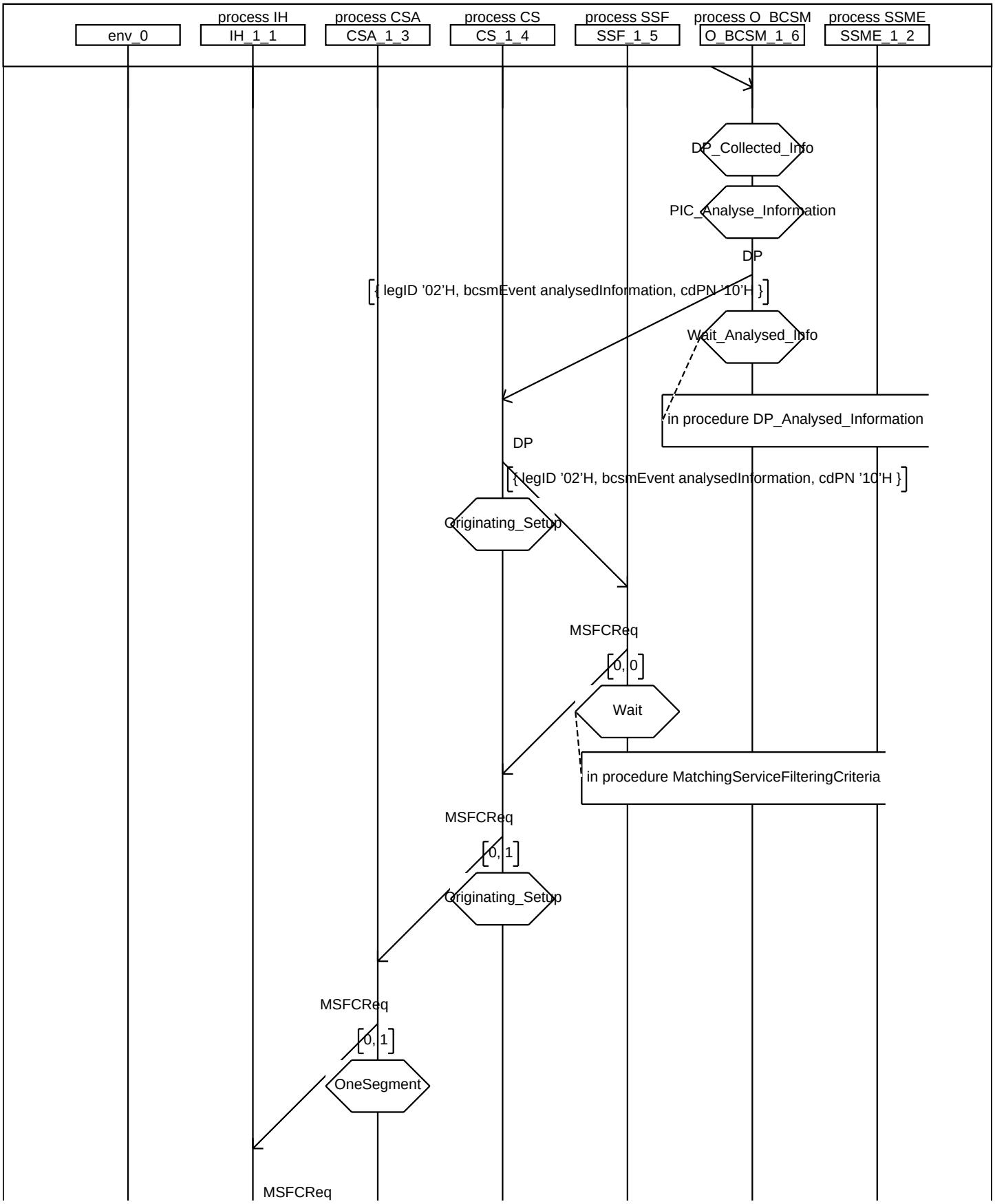
Annex A: CI_Detailed



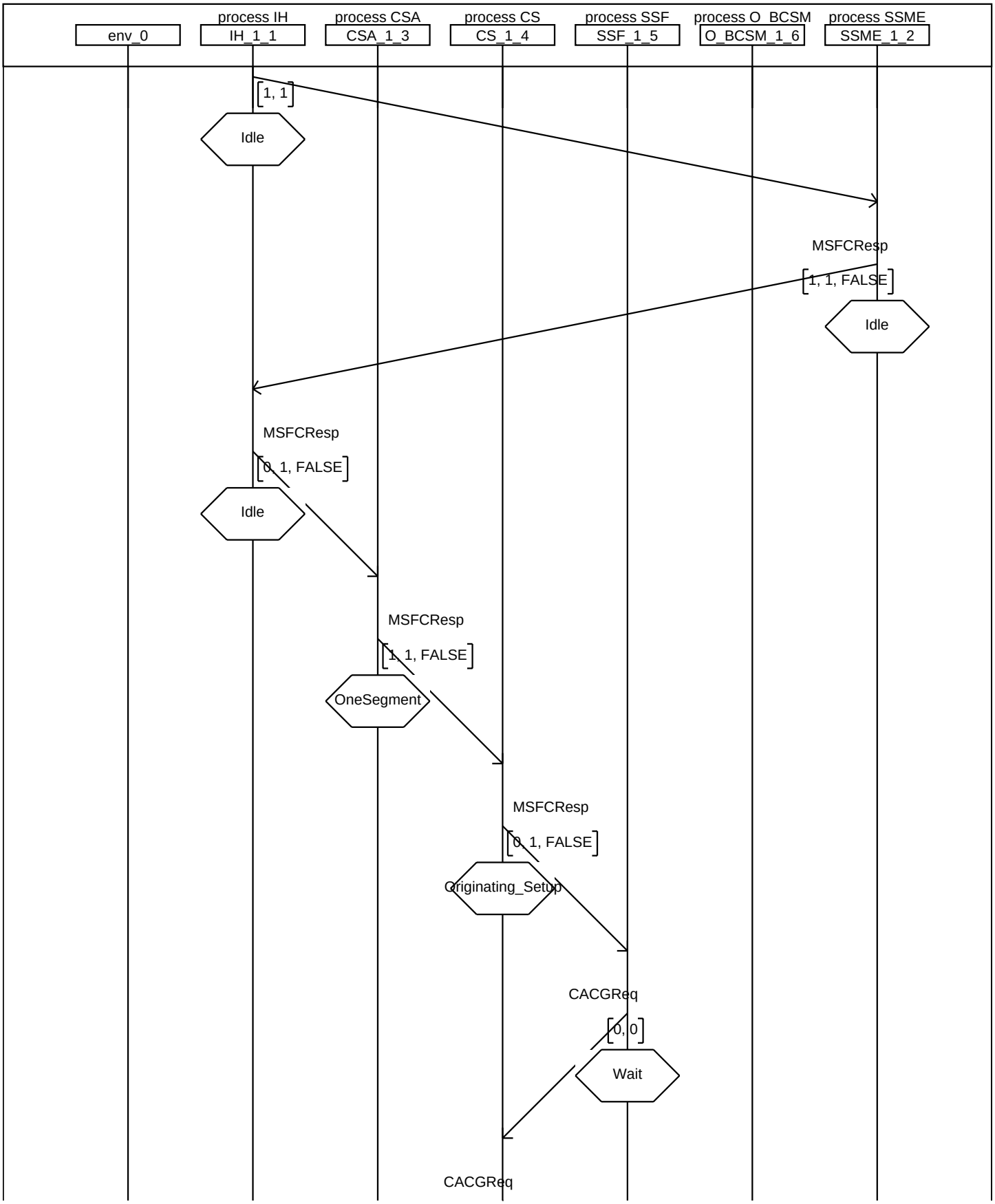
Annex A: CI_Detailed



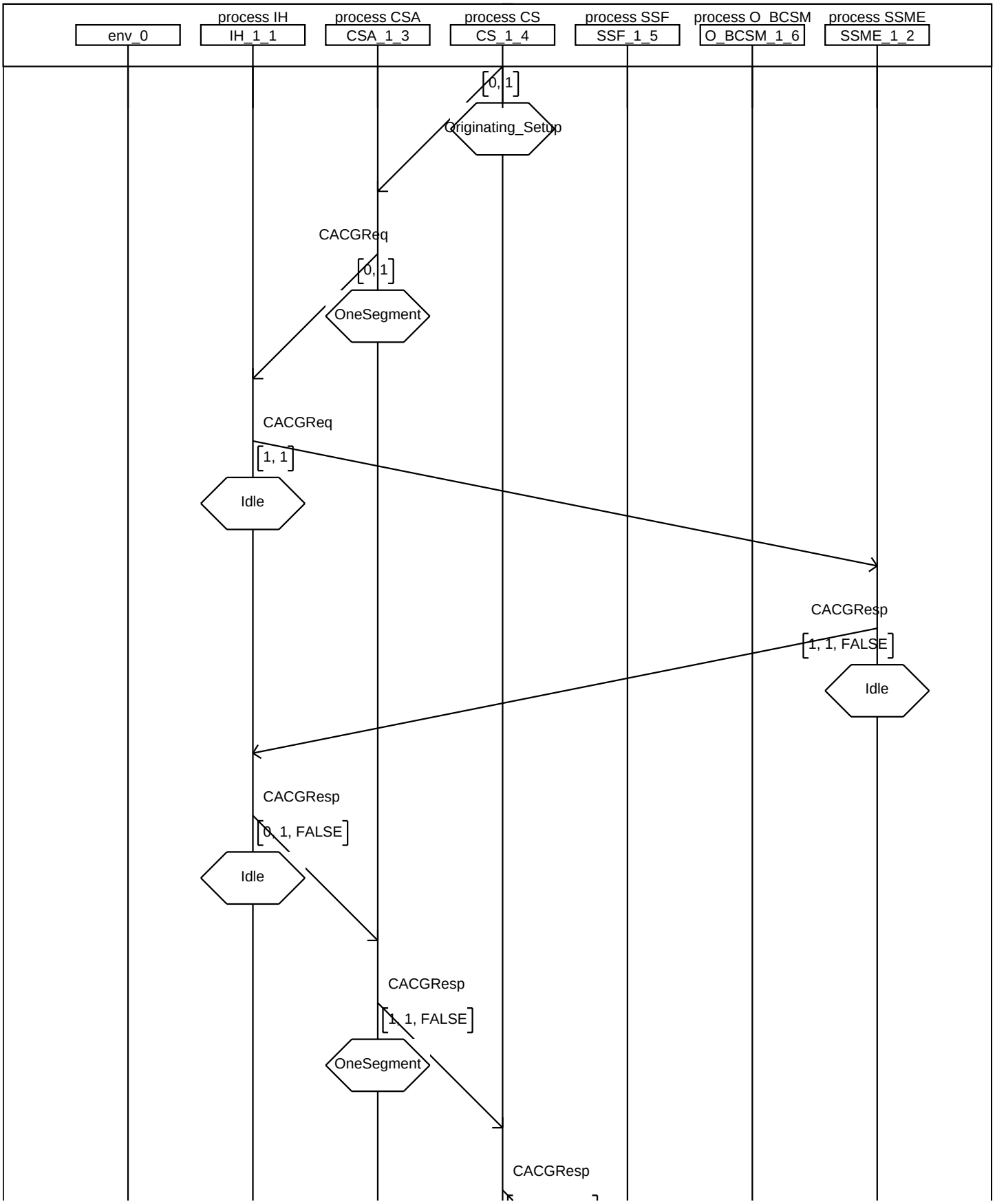
Annex A: CI_Detailed



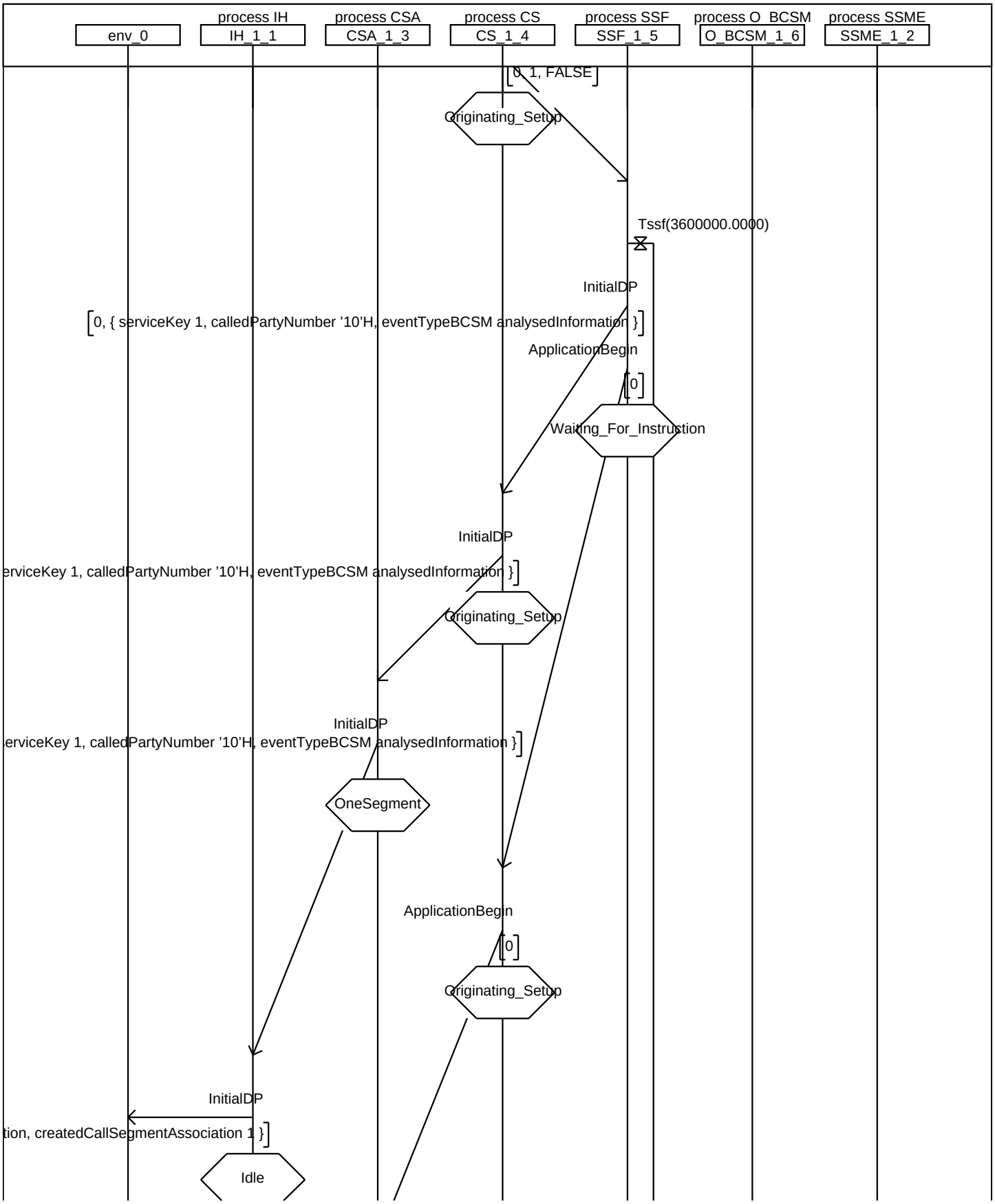
Annex A: CI_Detailed



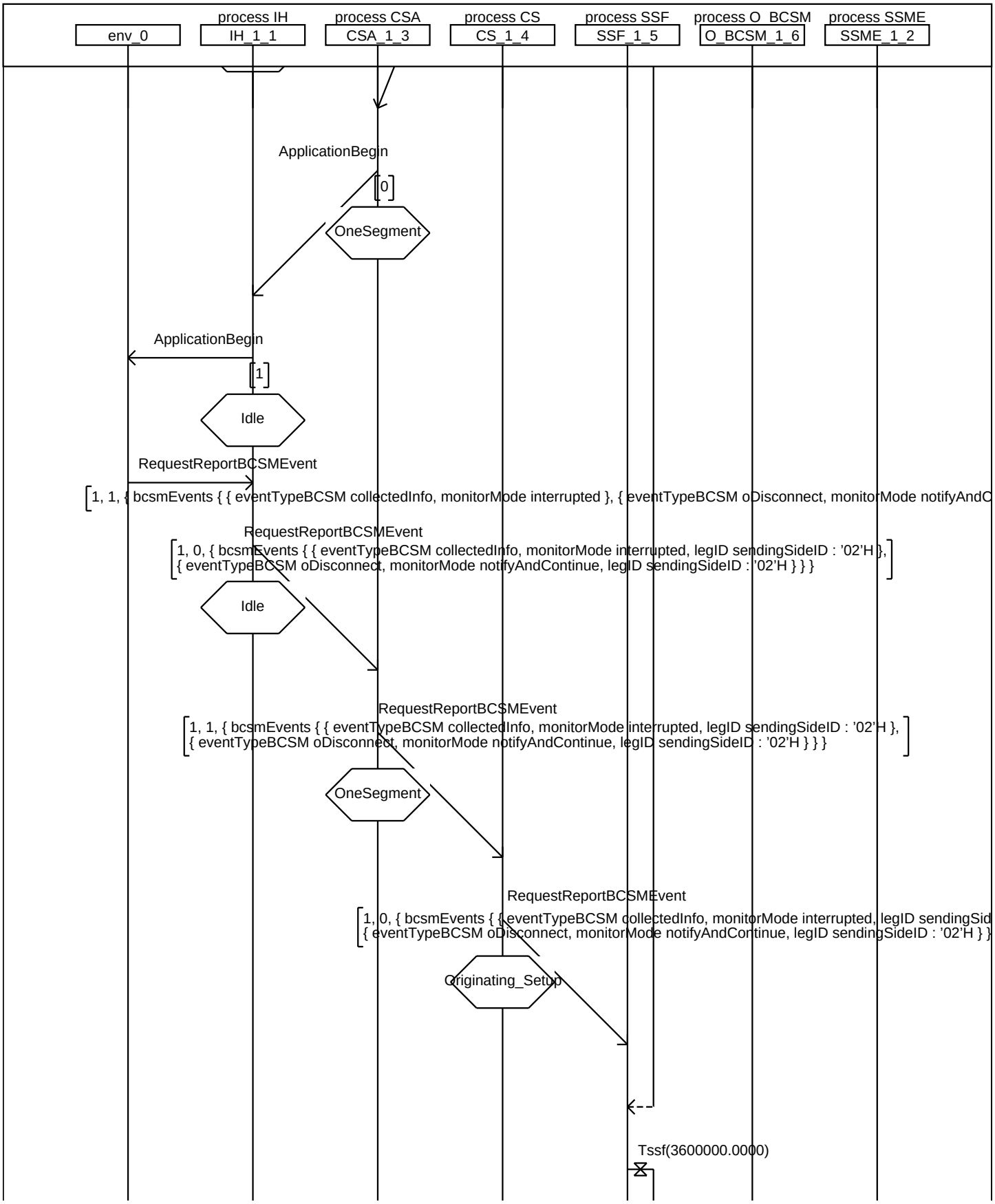
Annex A: CI_Detailed



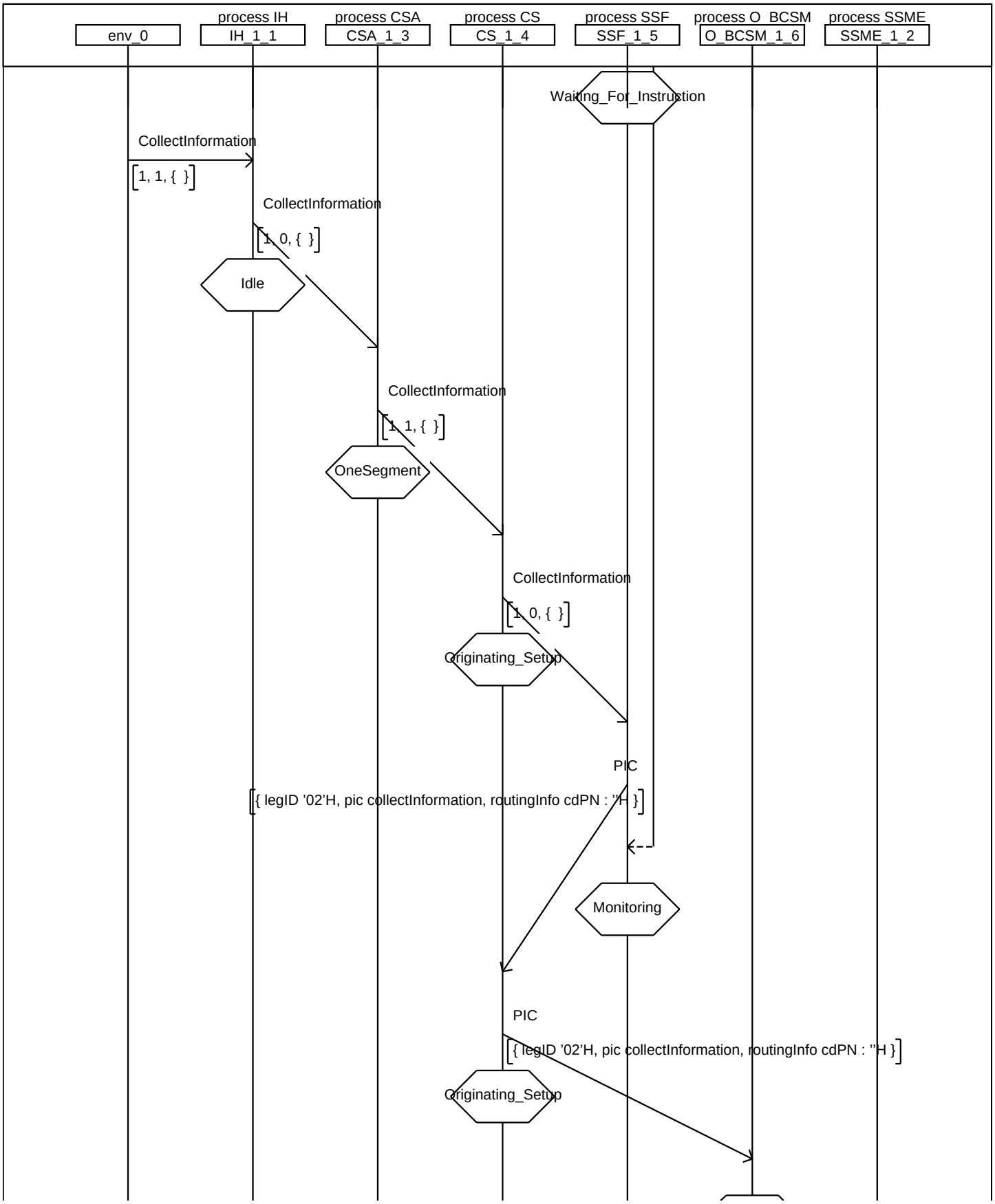
Annex A: CI_Detailed



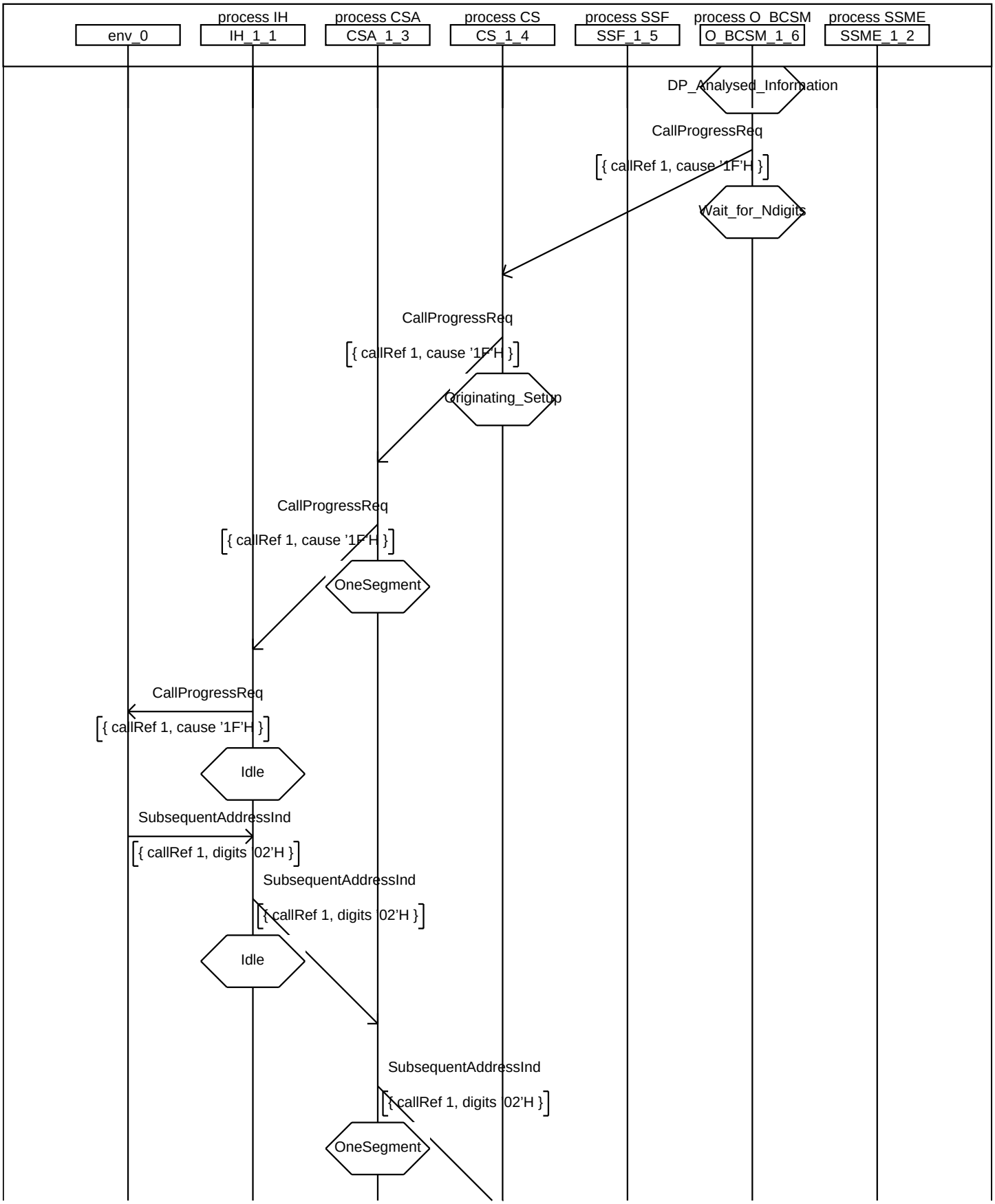
Annex A: CI_Detailed



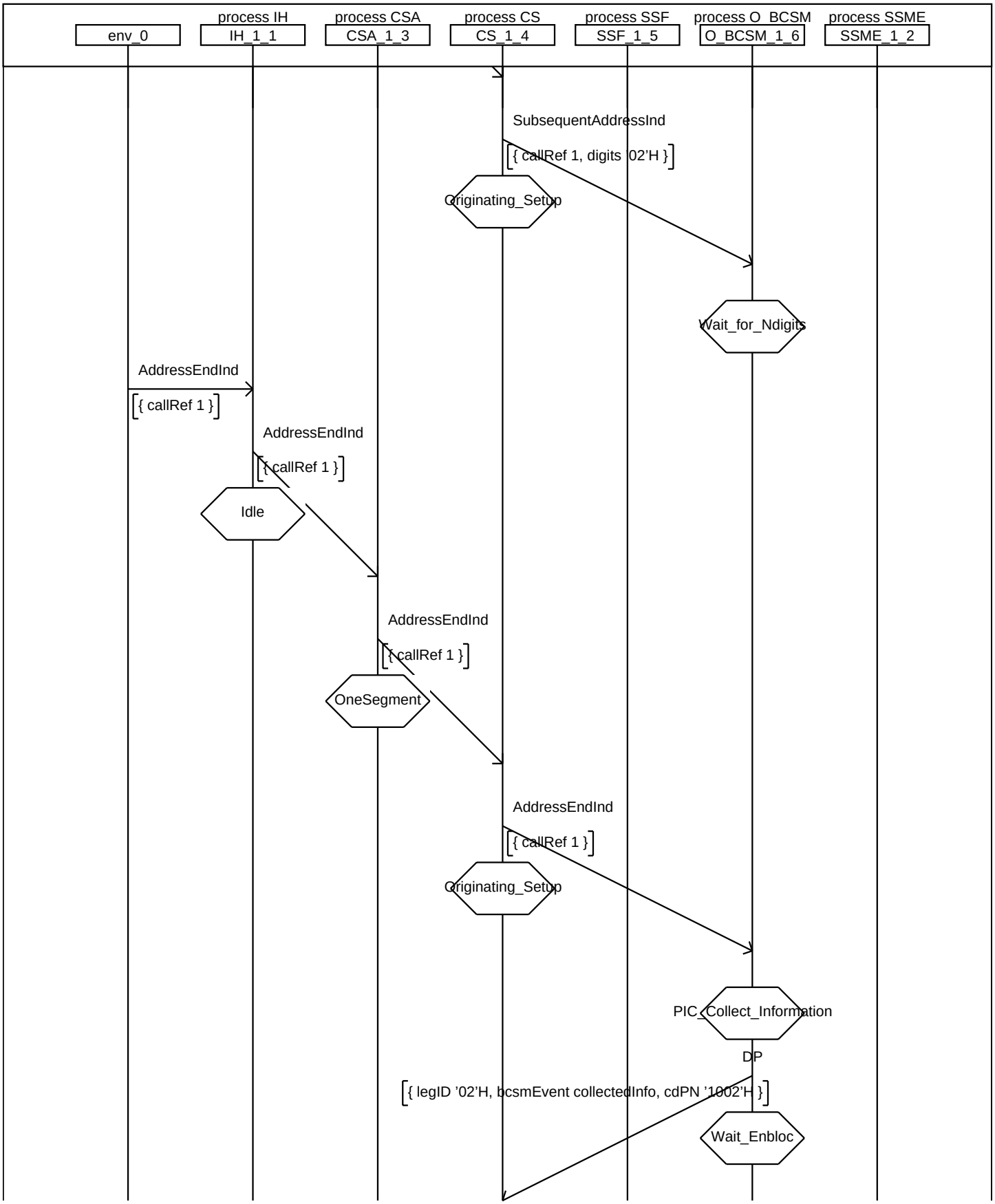
Annex A: CI_Detailed



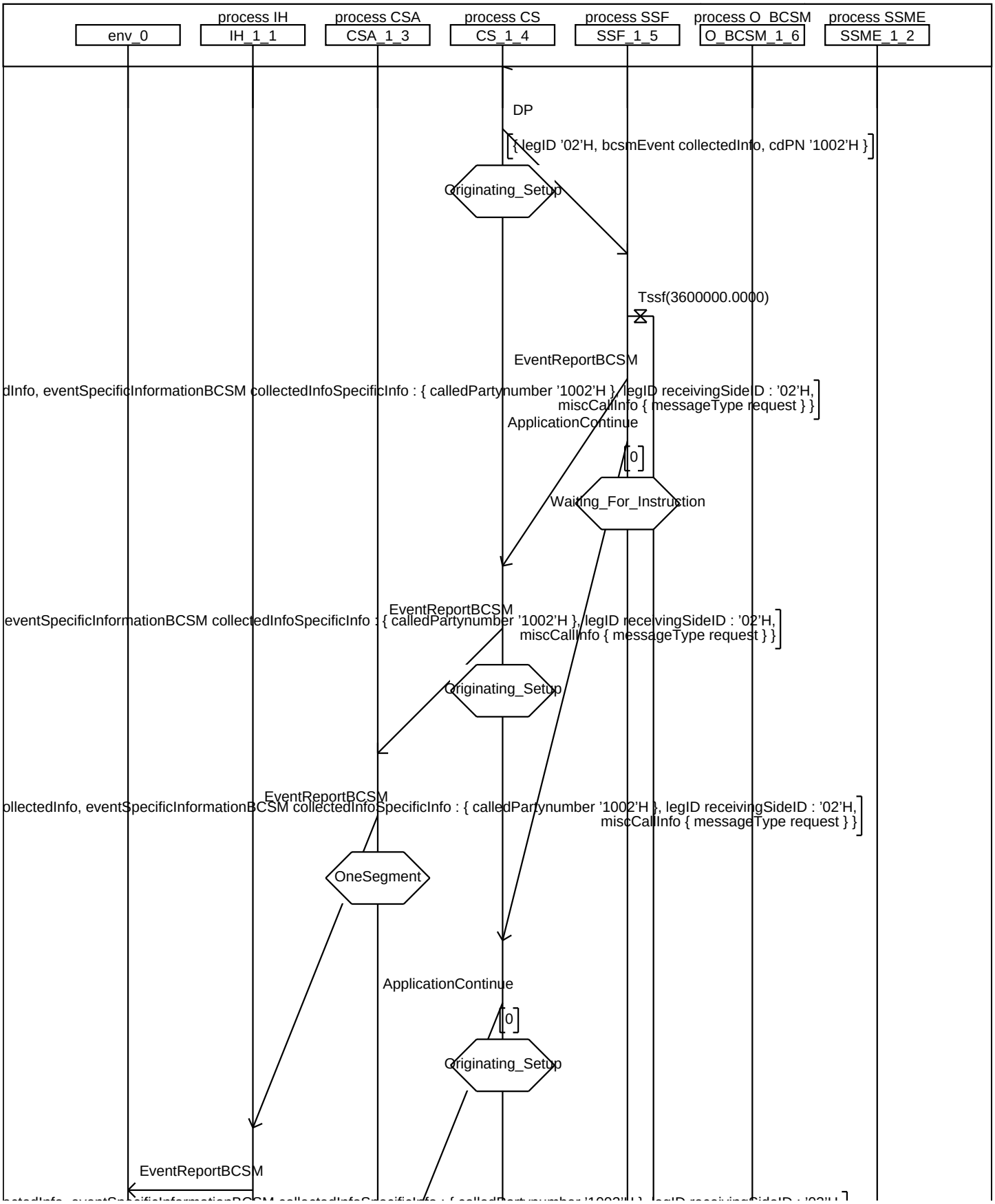
Annex A: CI_Detailed



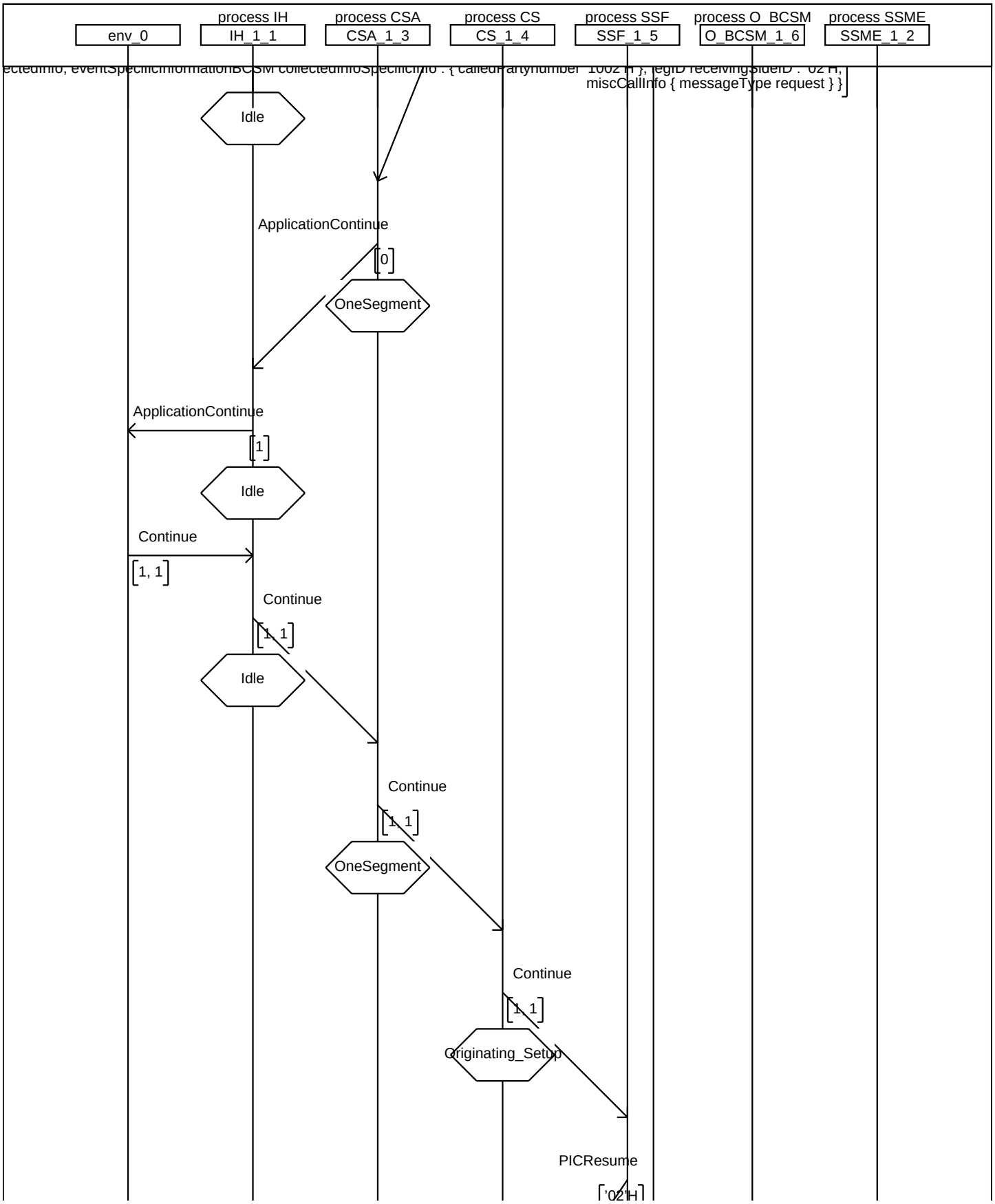
Annex A: CI_Detailed



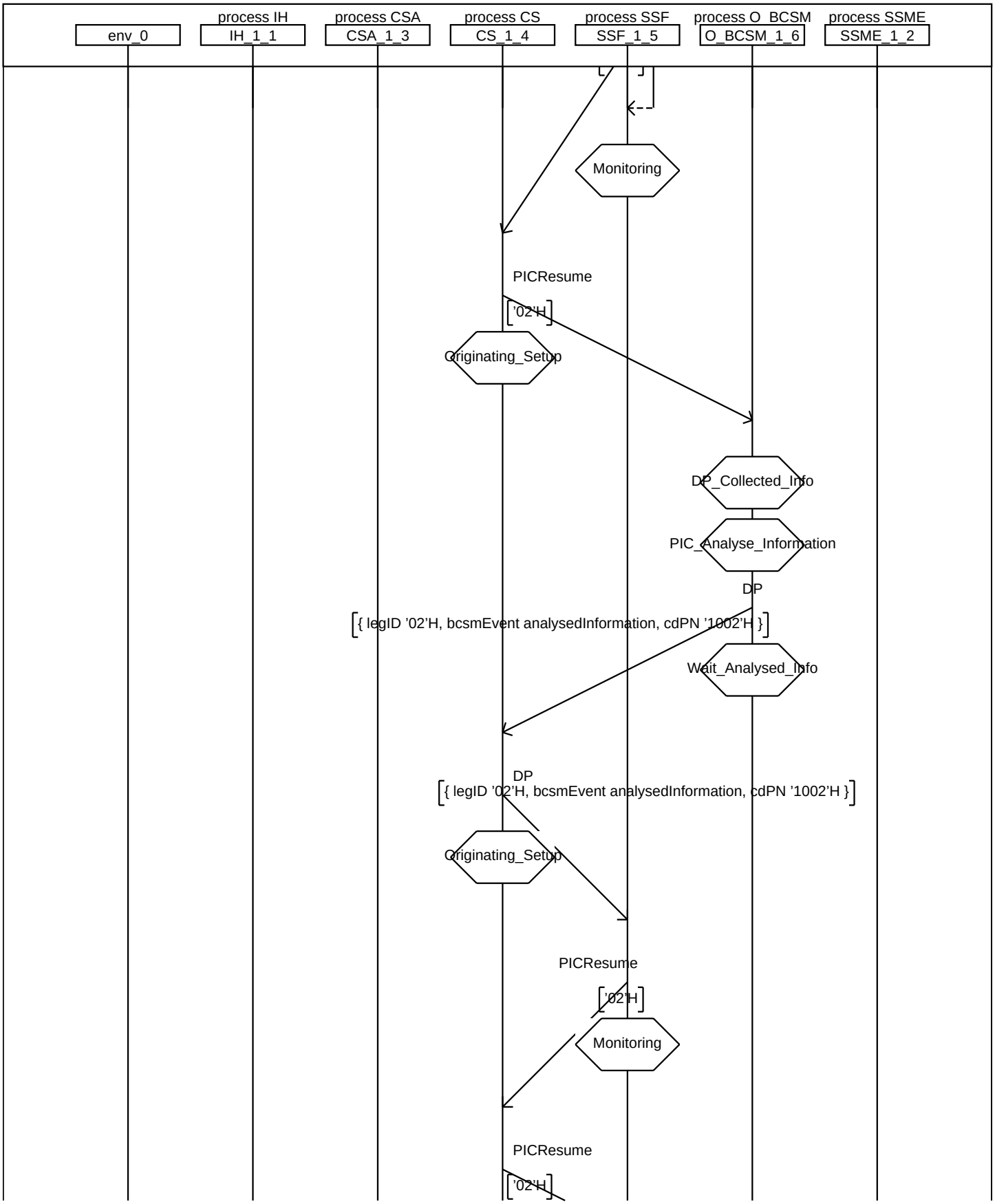
Annex A: CI_Detailed



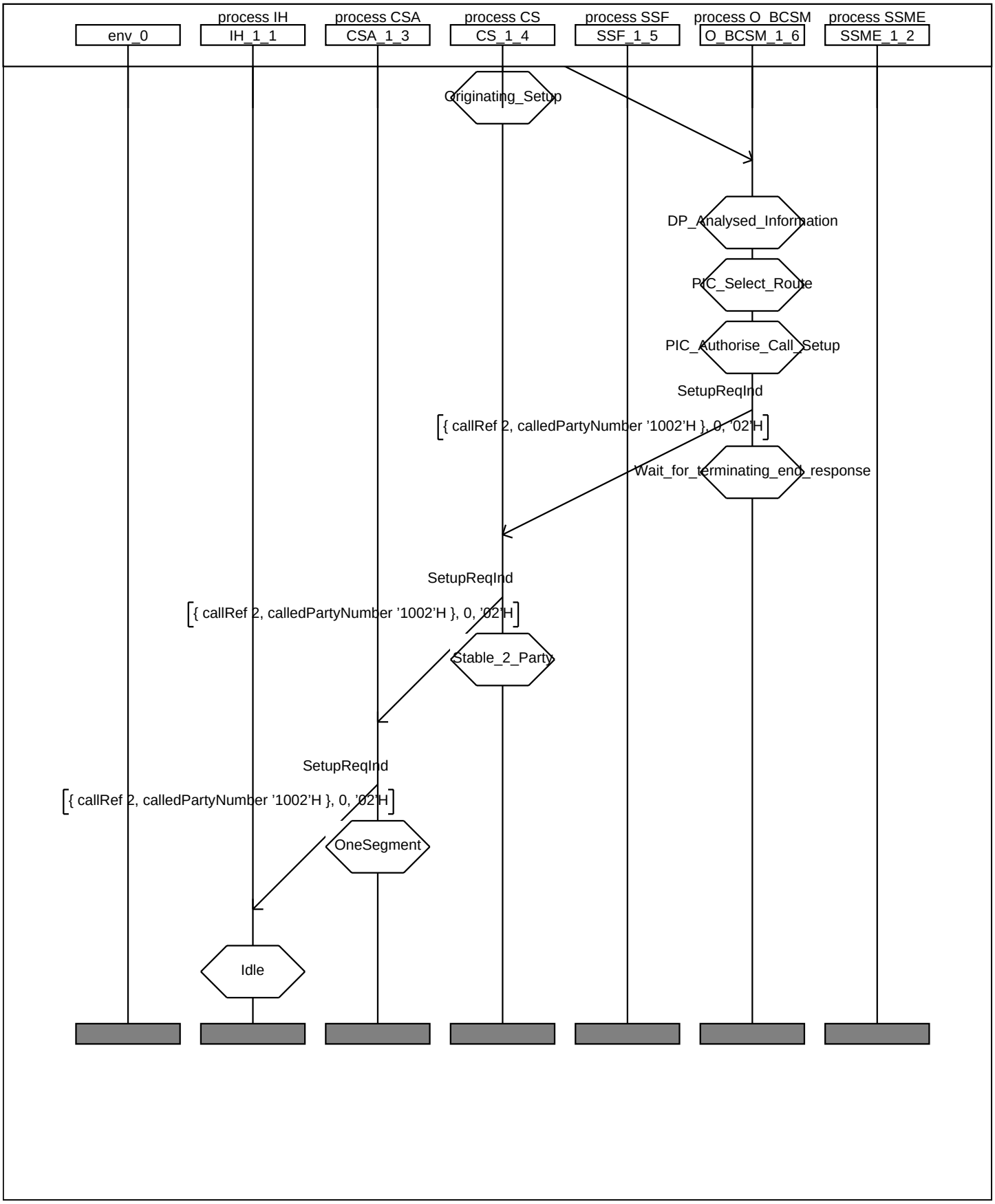
Annex A: CI_Detailed



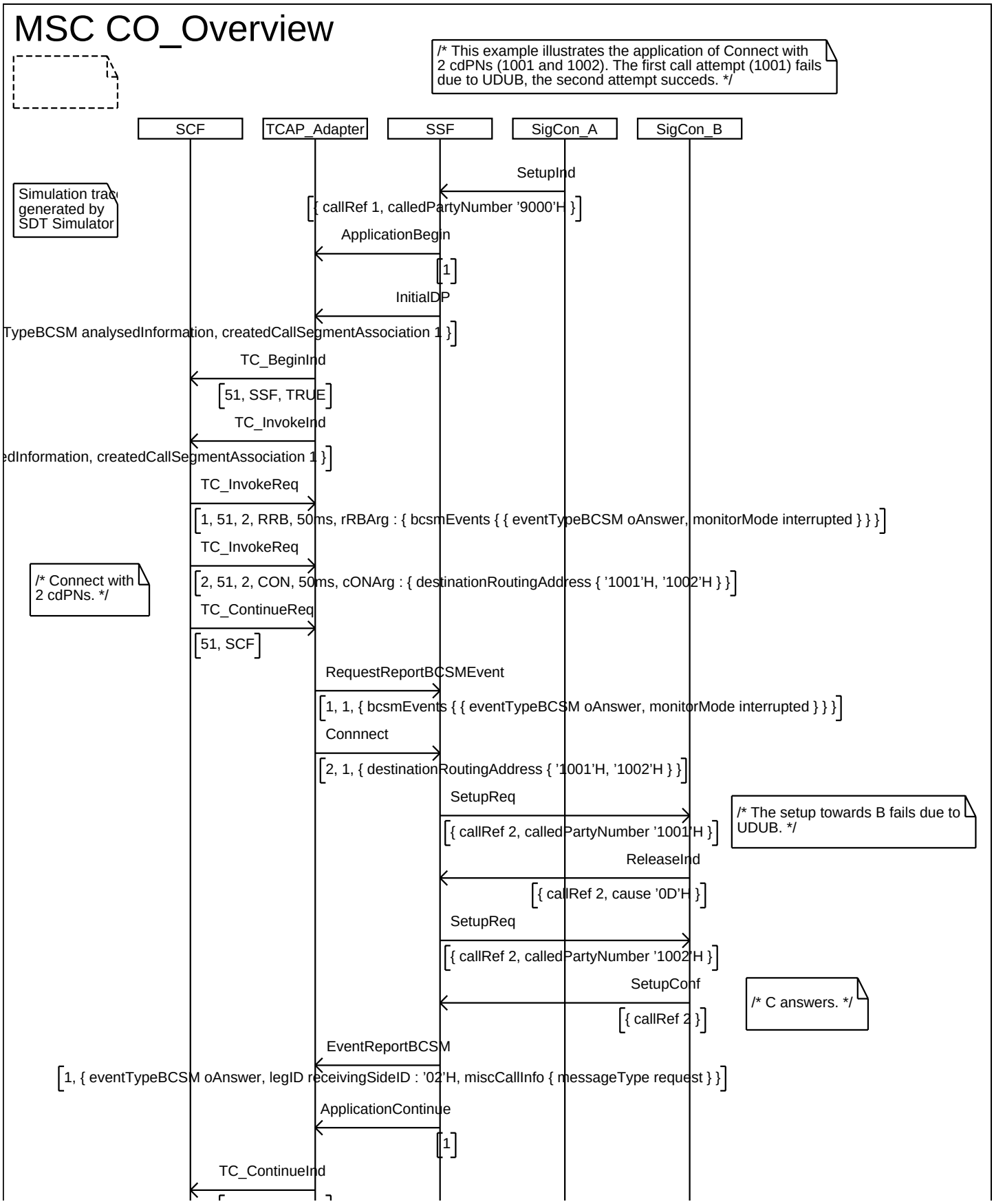
Annex A: CI_Detailed



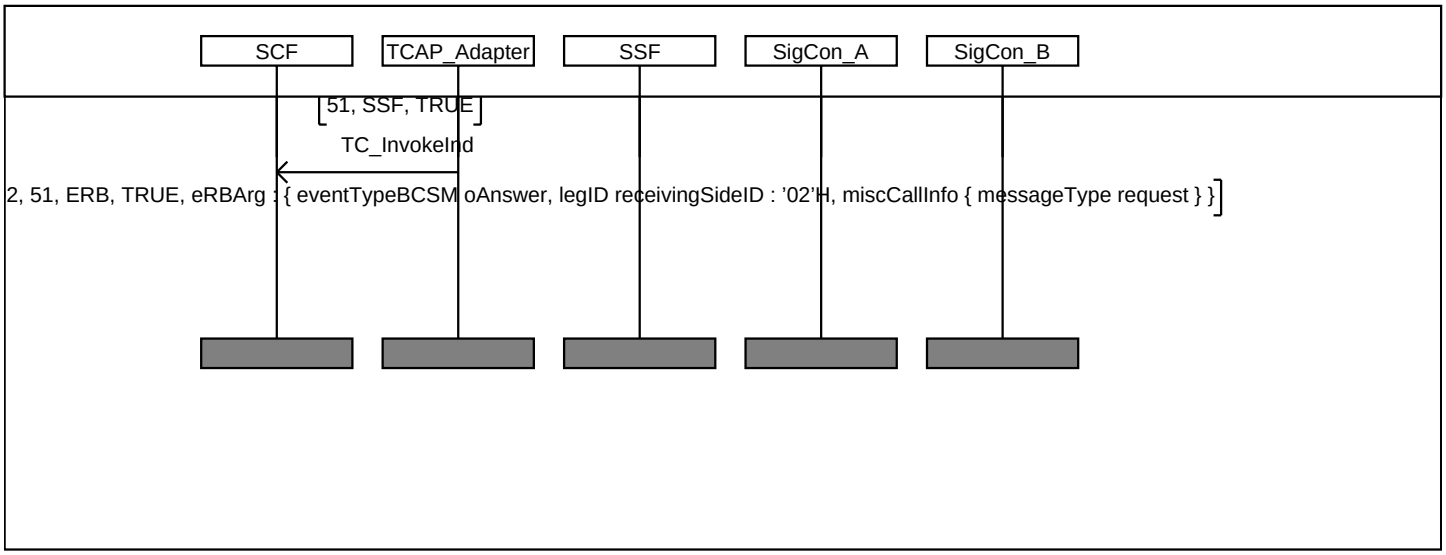
Annex A: CI_Detailed



Annex A: CO_Overview



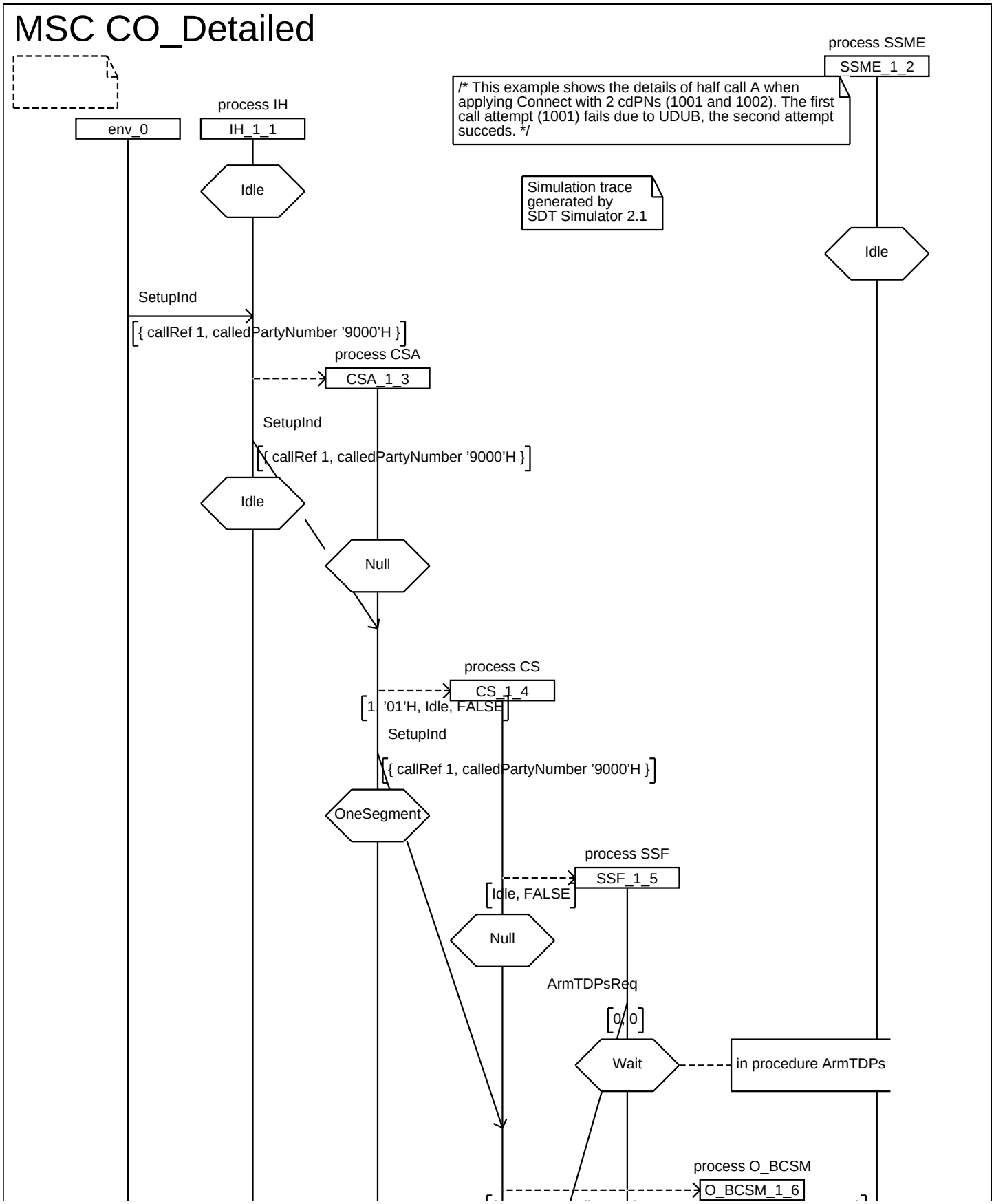
Annex A: CO_Overview



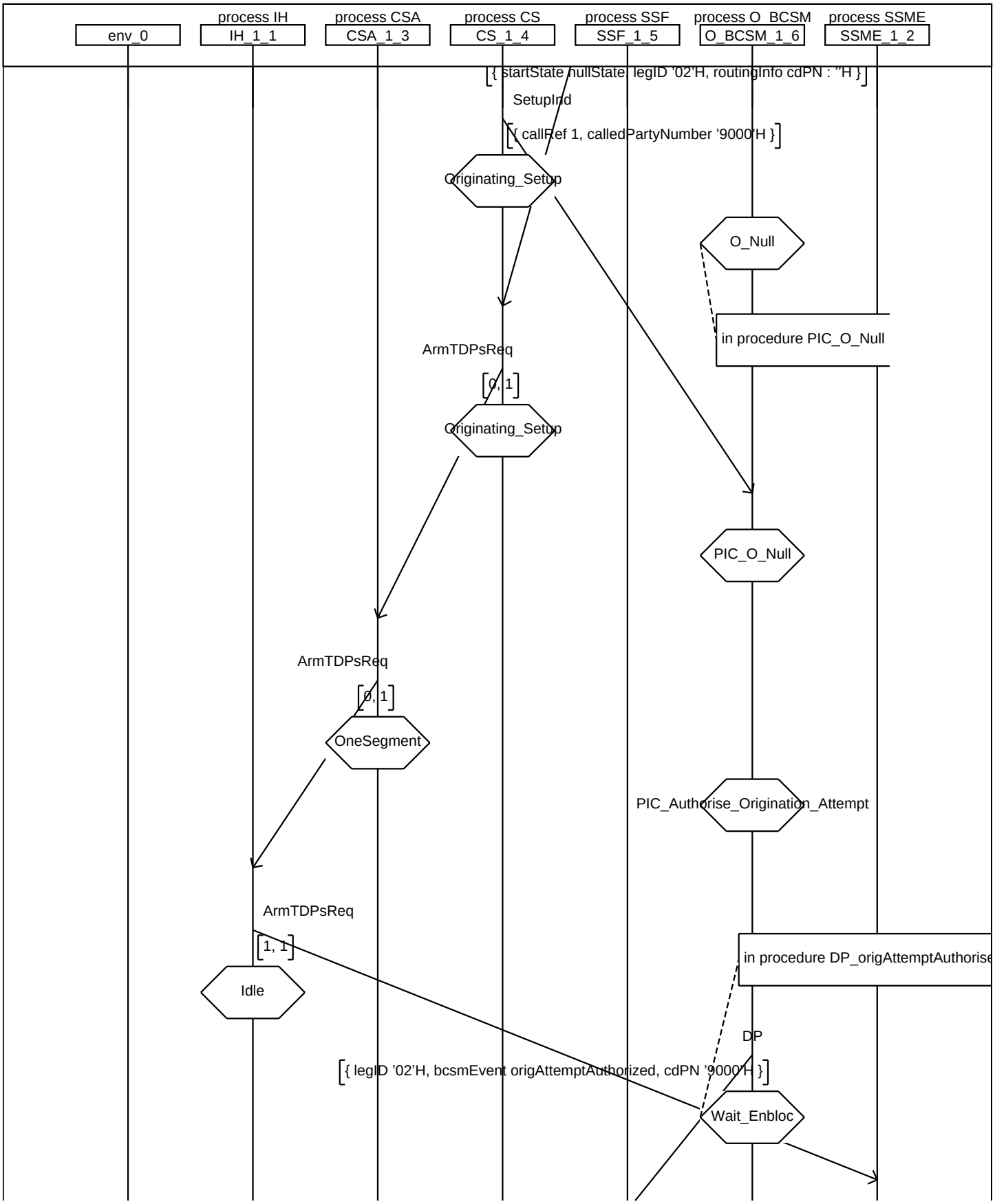
Annex A: CO_Overview

Simulation trace
generated by
SDT Simulator 2.1

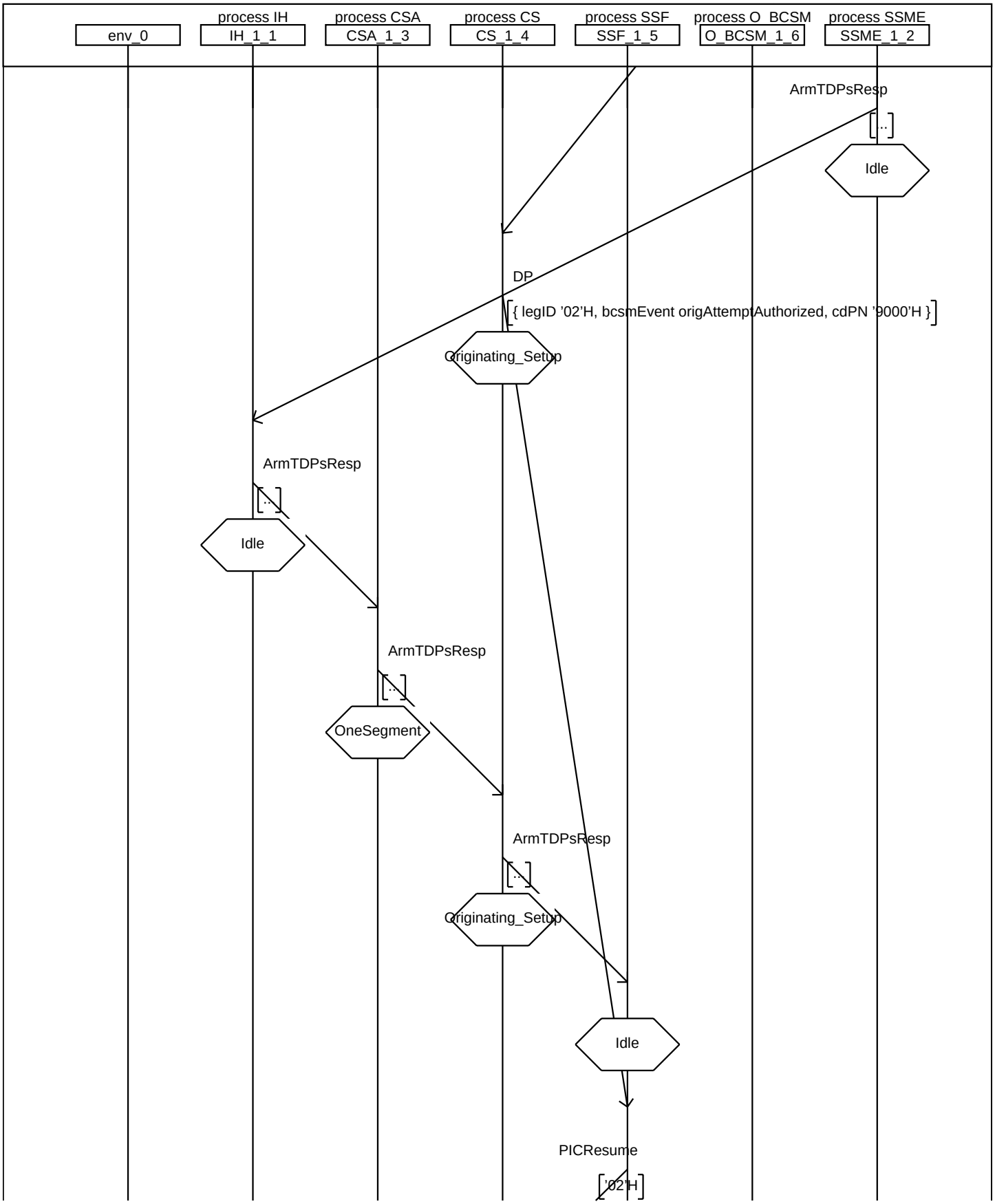
Annex A: CO_Detailed



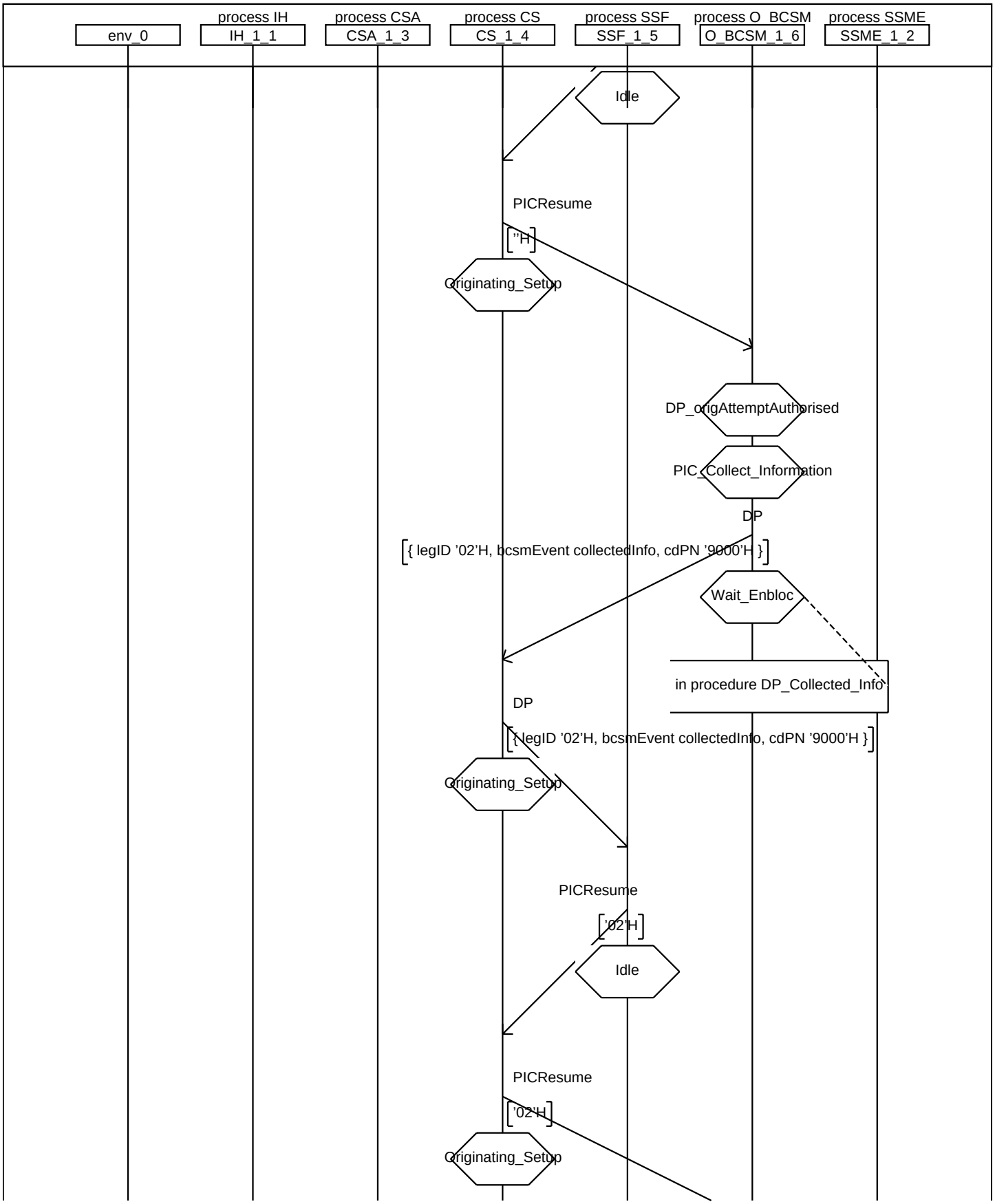
Annex A: CO_Detailed



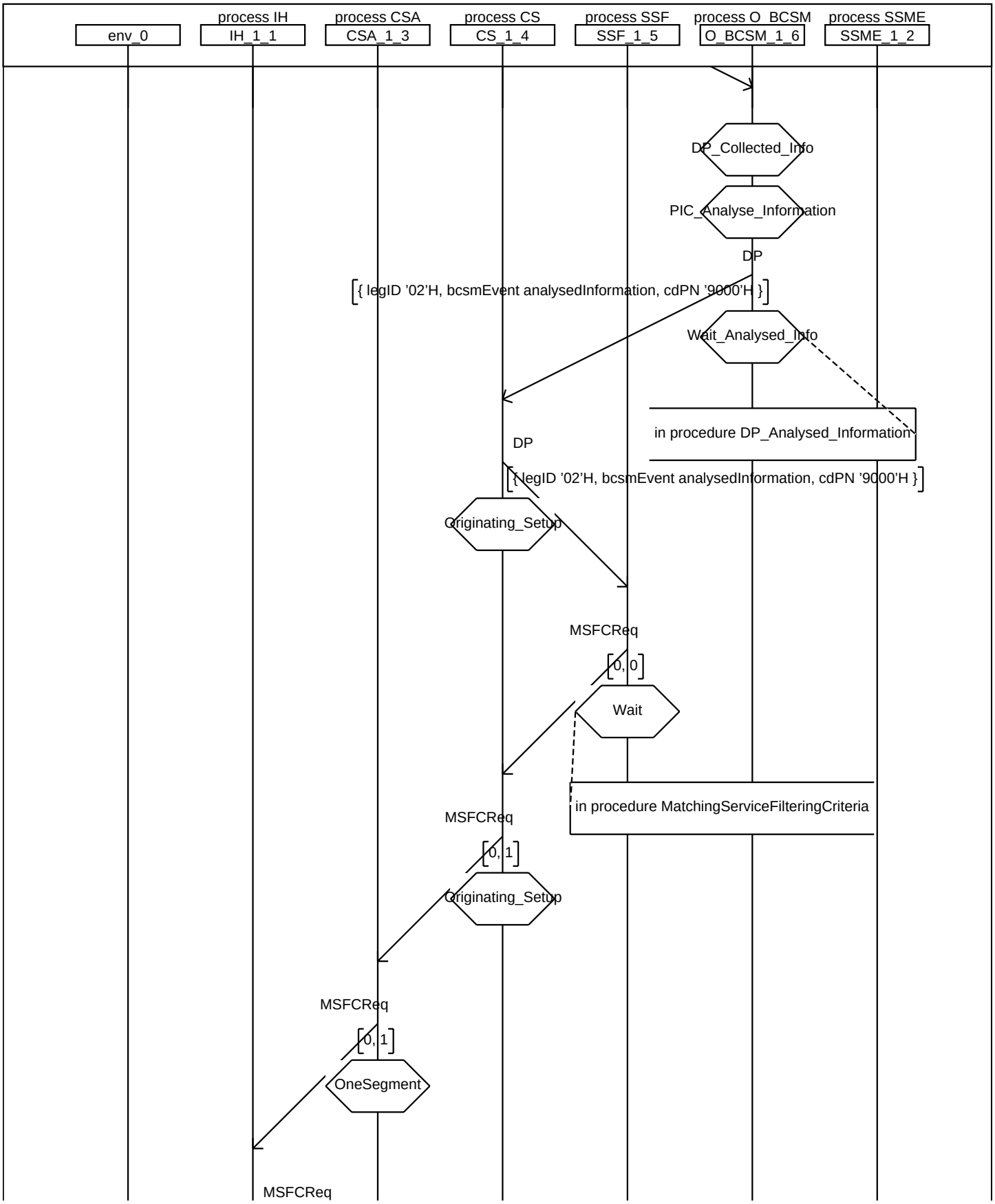
Annex A: CO_Detailed



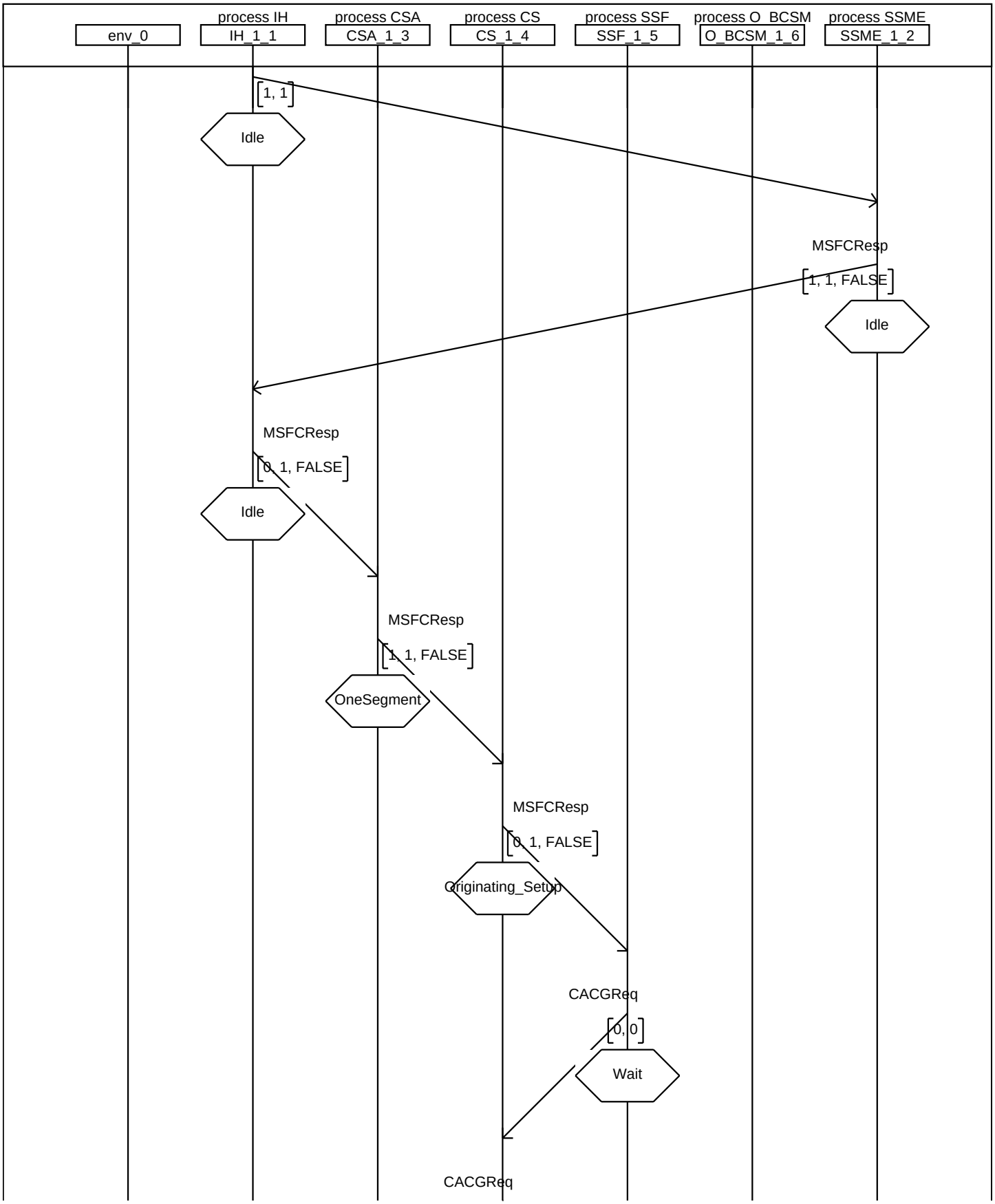
Annex A: CO_Detailed



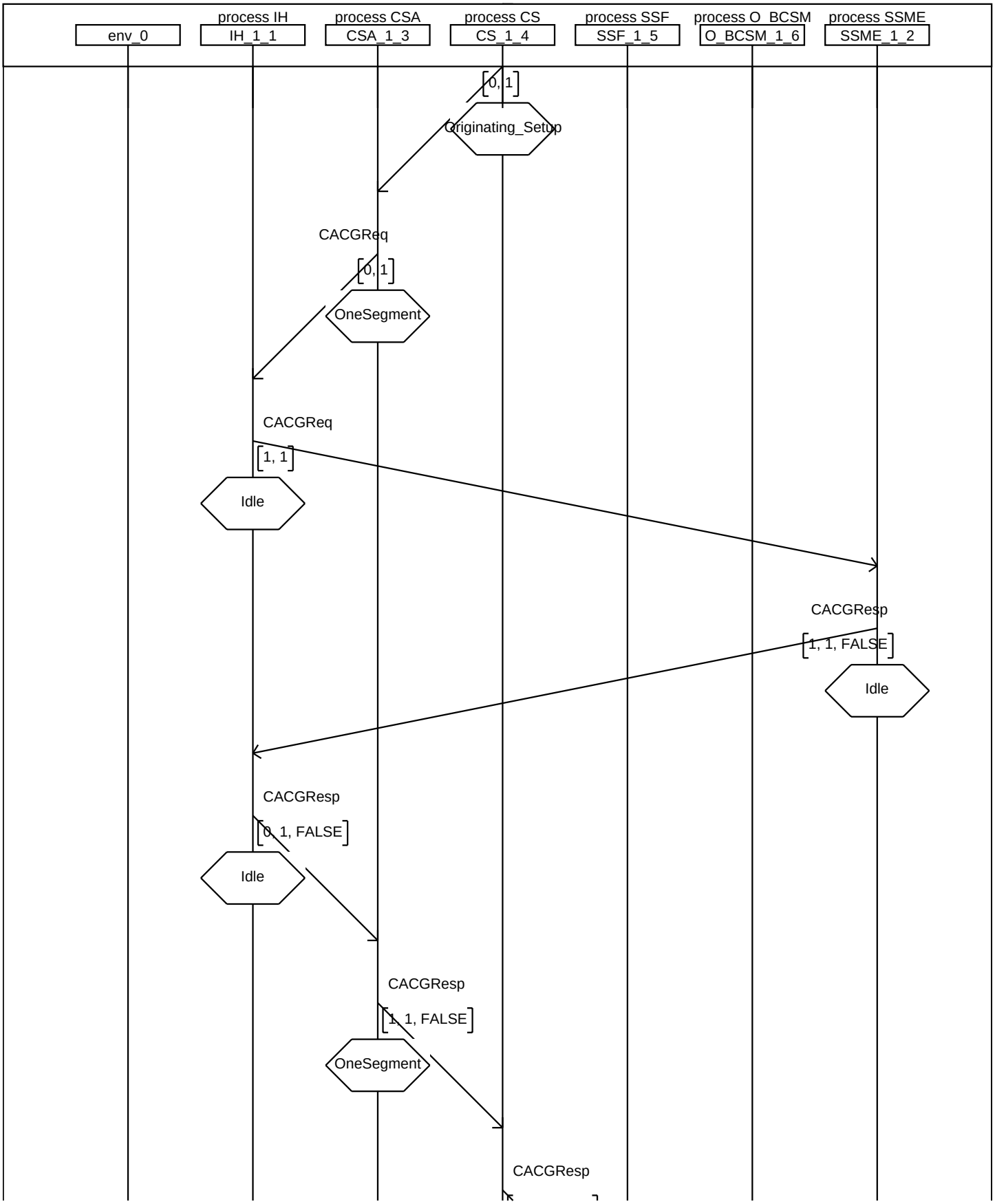
Annex A: CO_Detailed



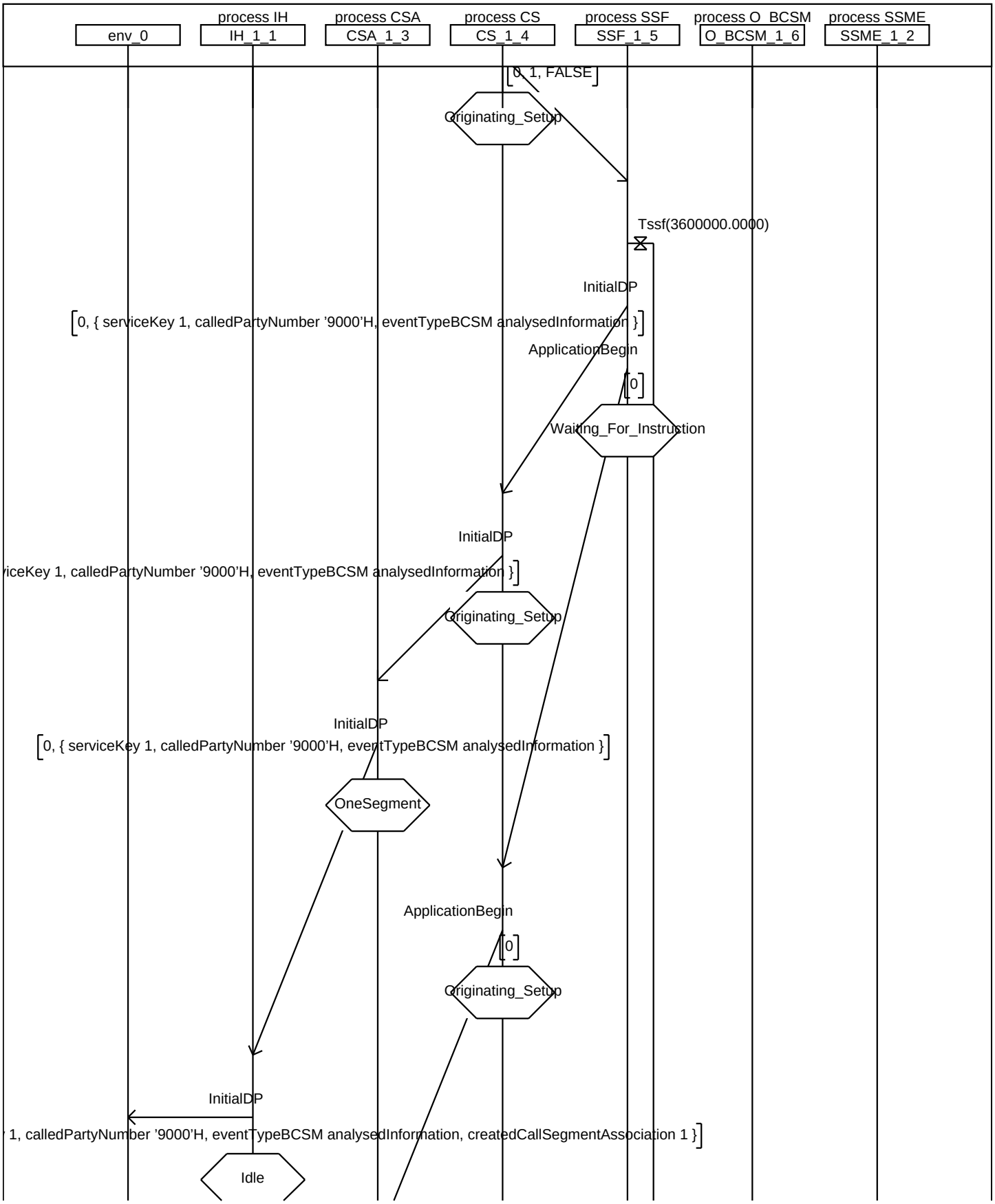
Annex A: CO_Detailed



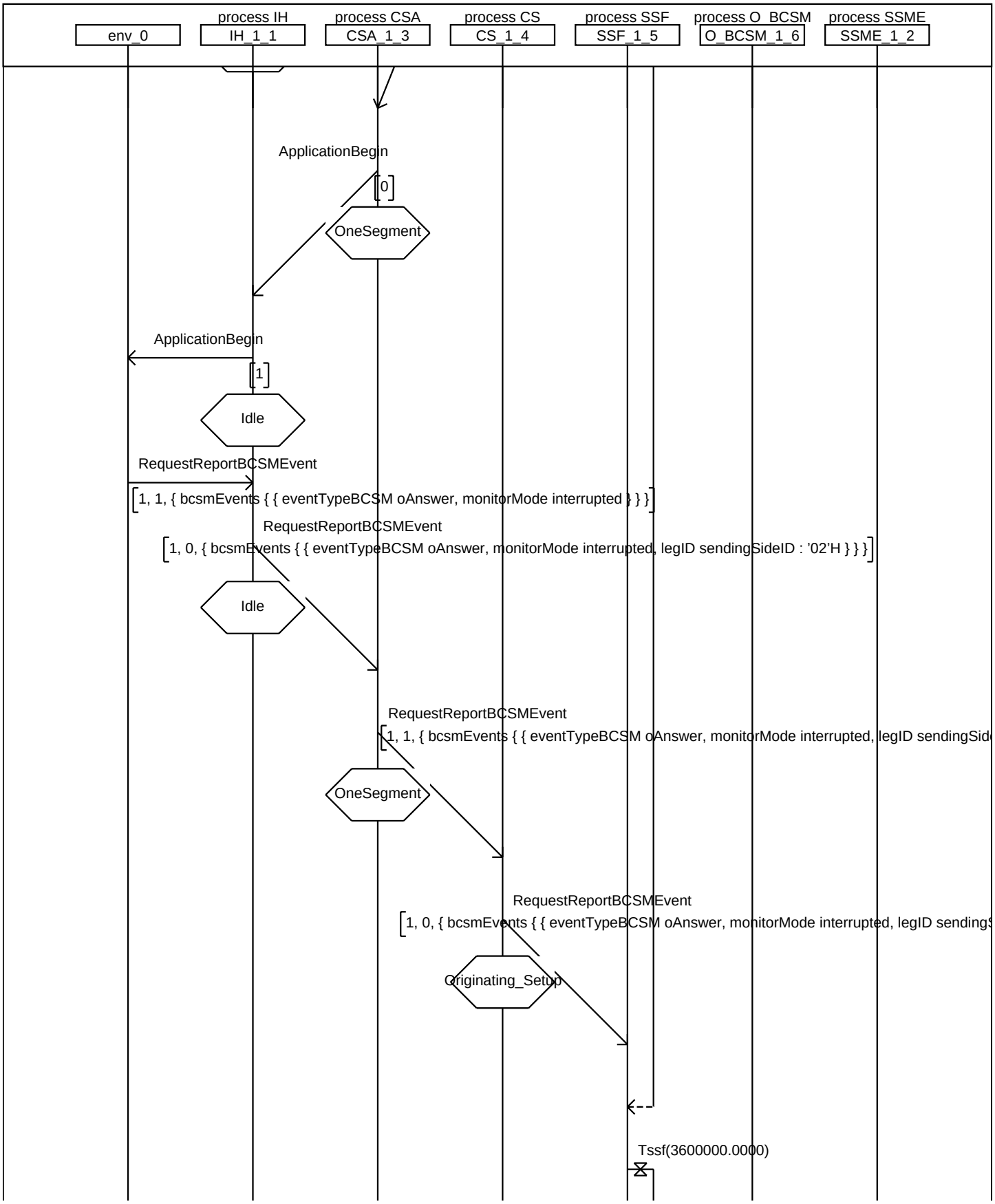
Annex A: CO_Detailed



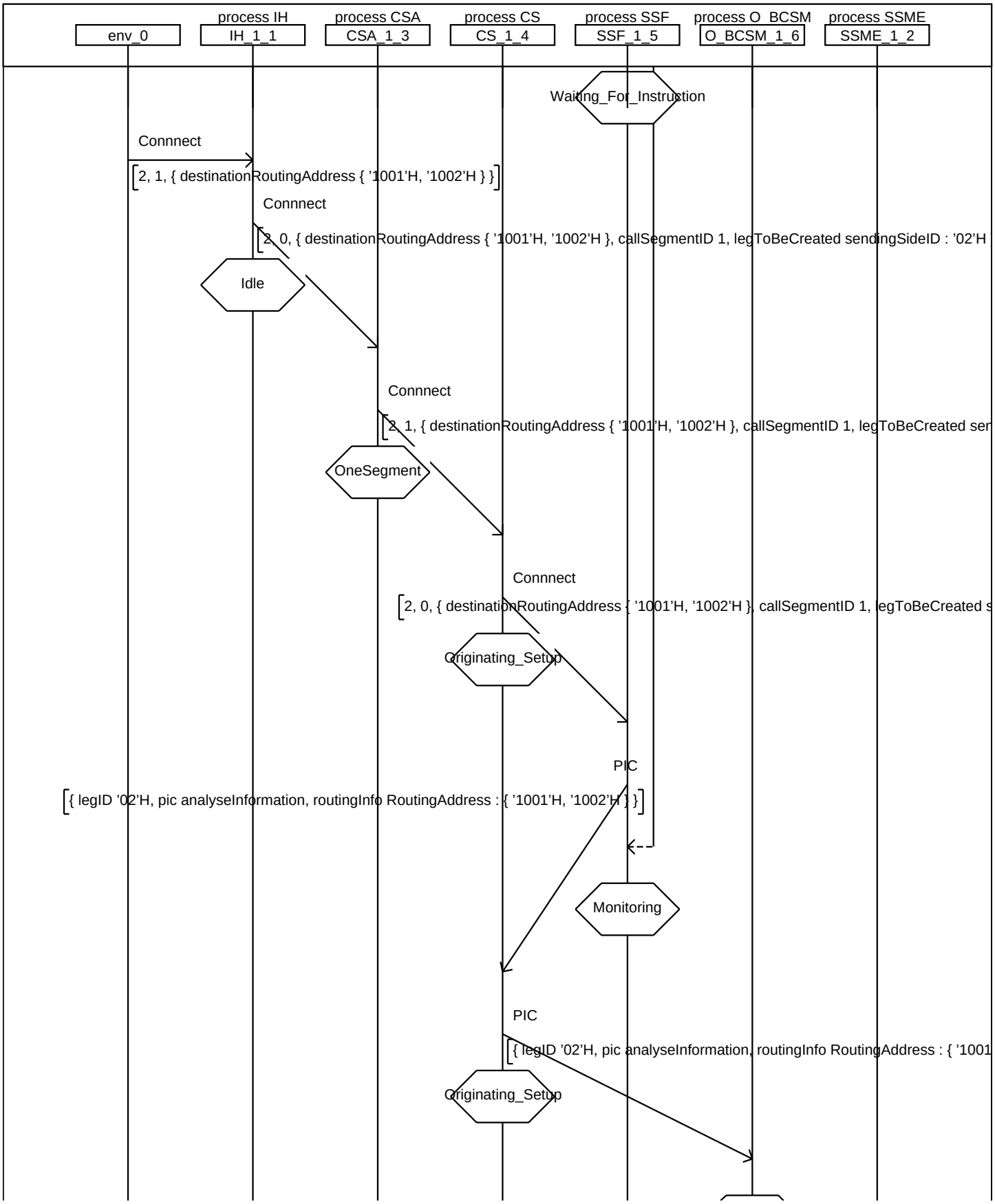
Annex A: CO_Detailed



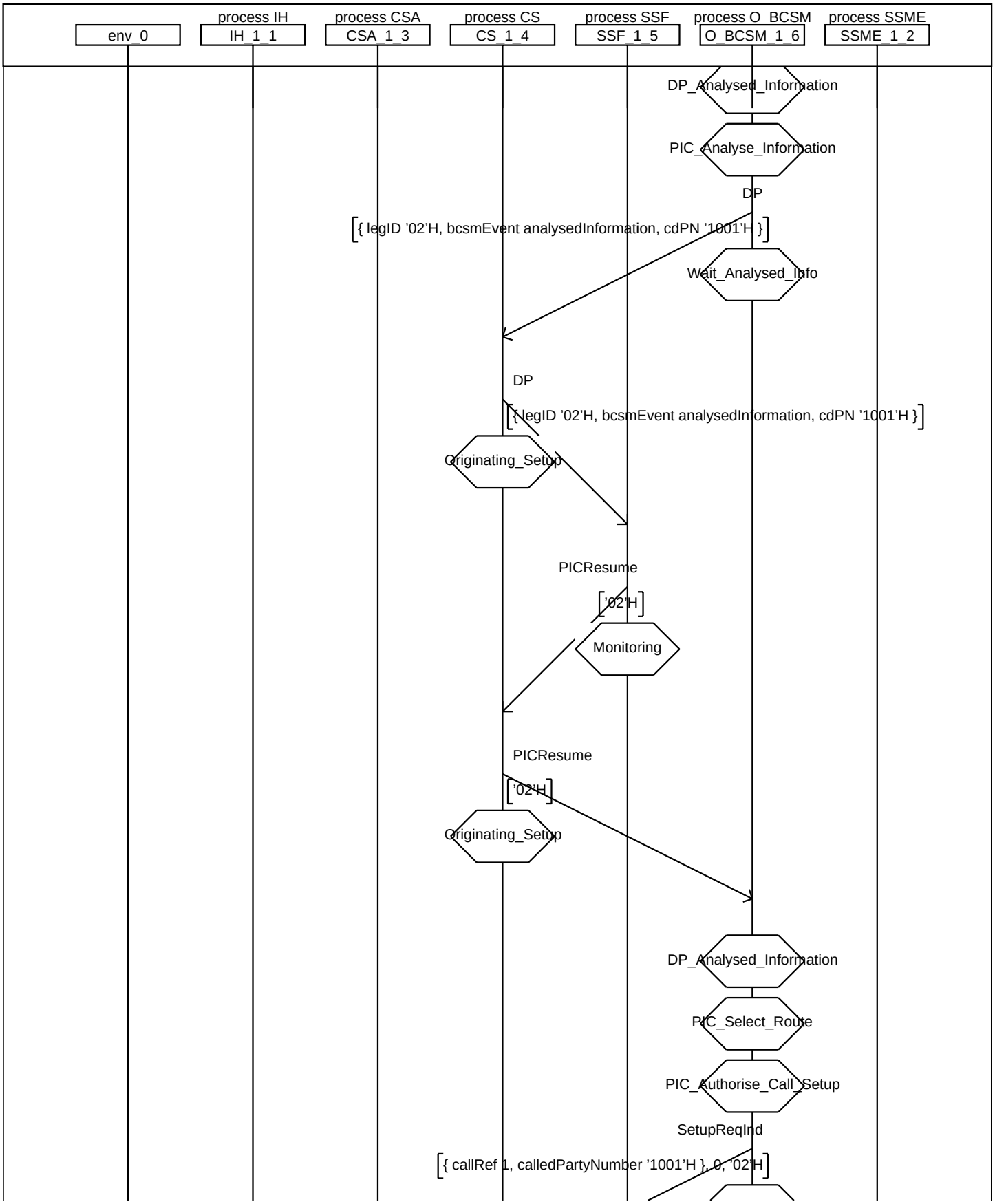
Annex A: CO_Detailed



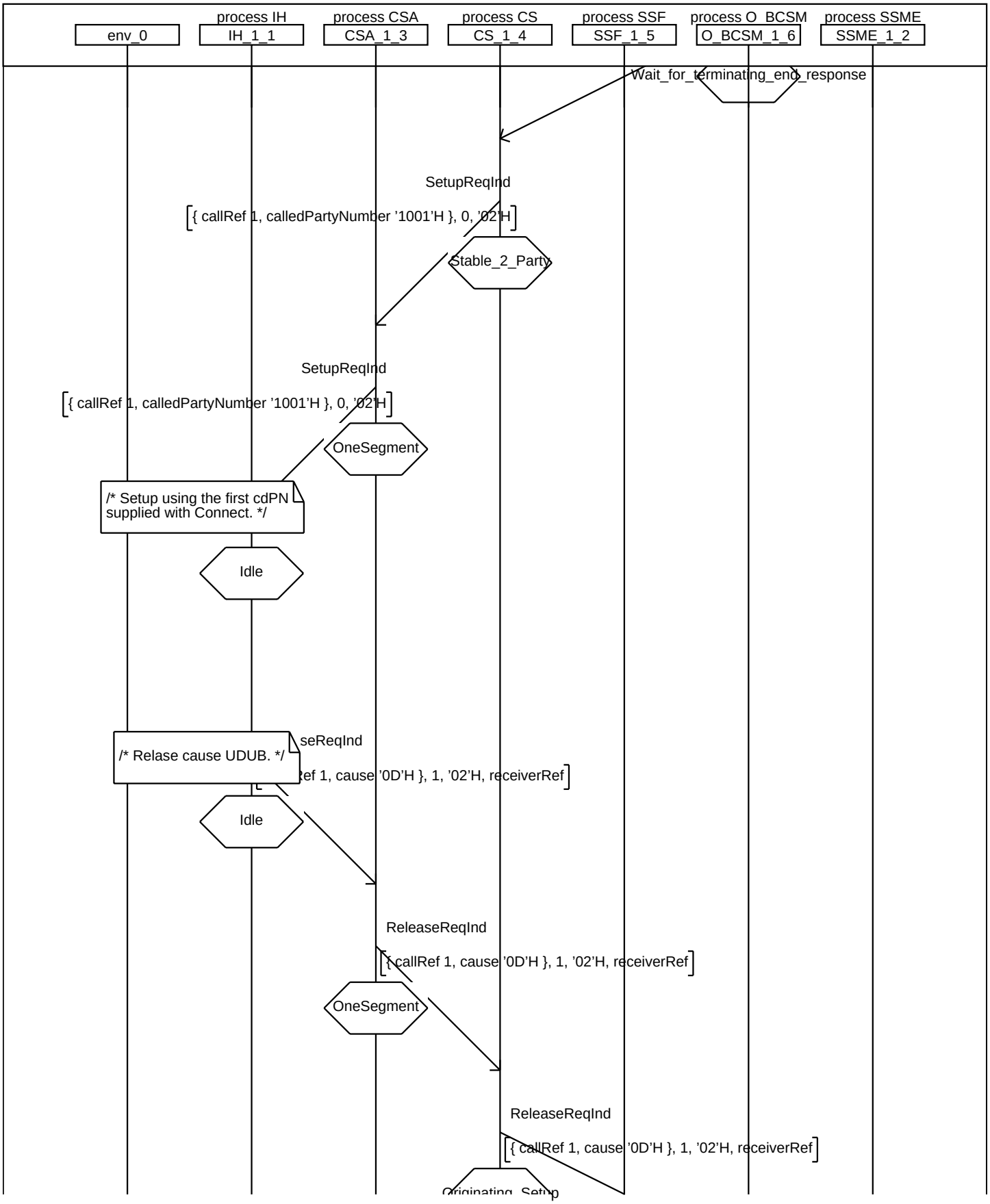
Annex A: CO_Detailed



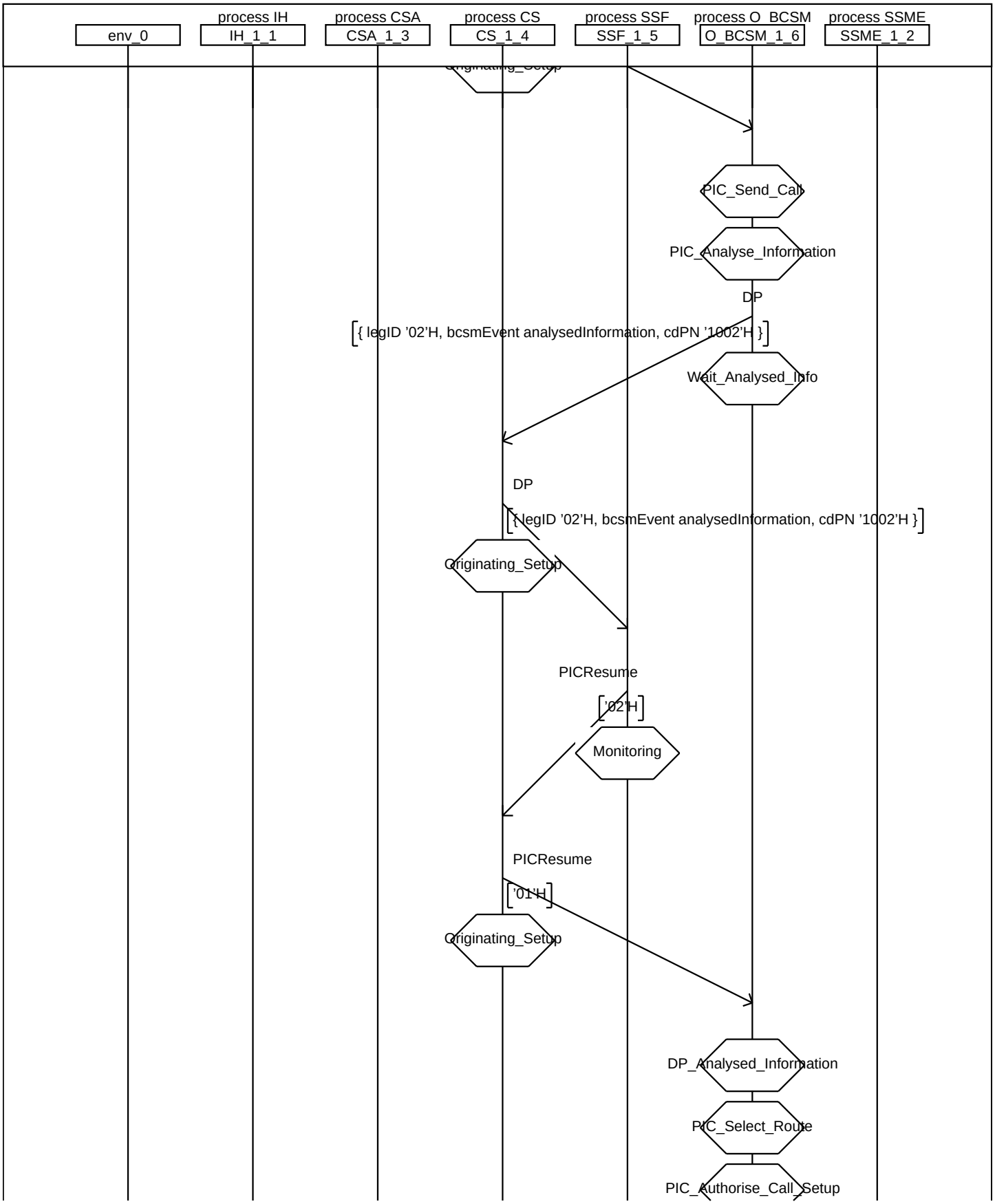
Annex A: CO_Detailed



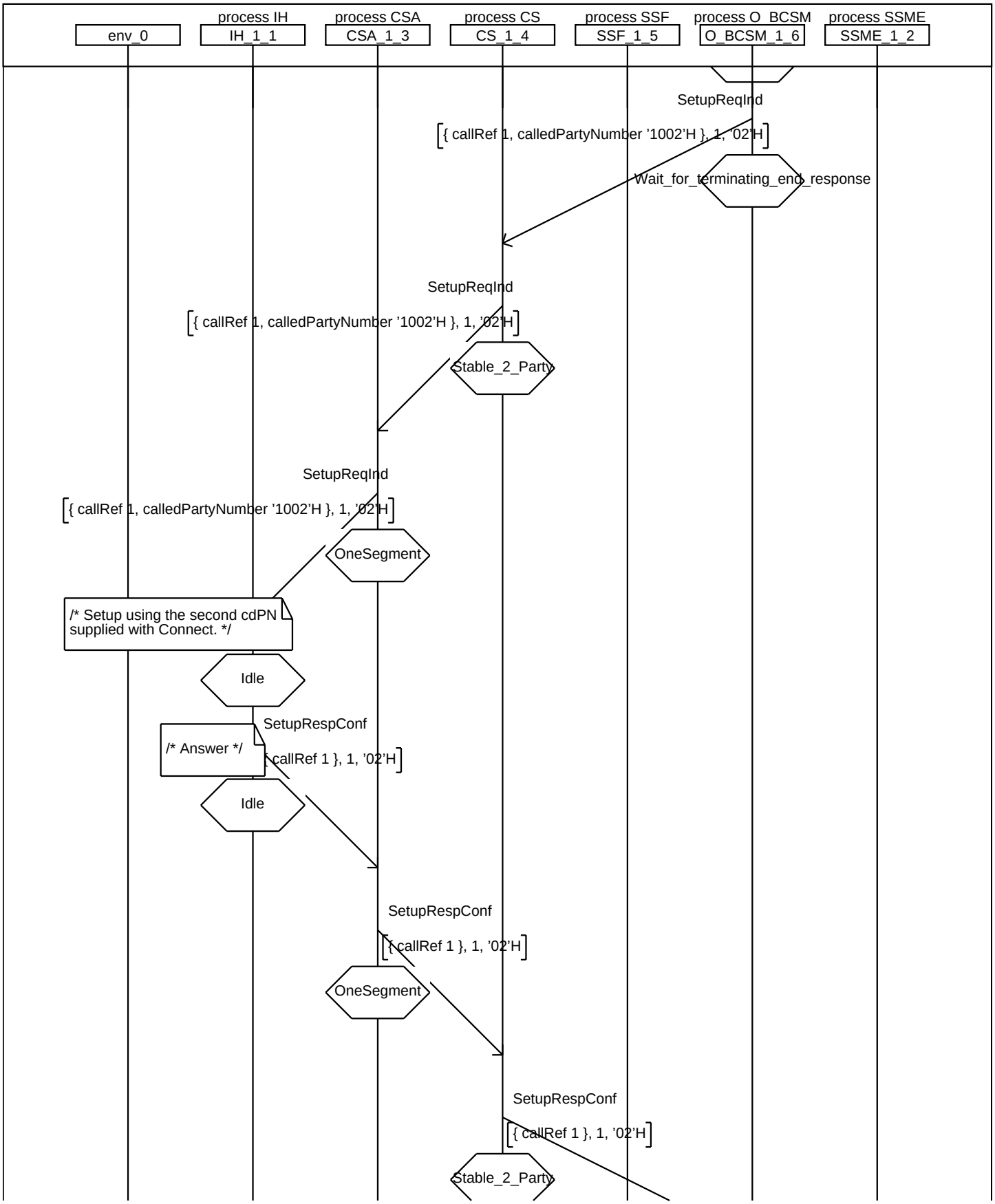
Annex A: CO_Detailed



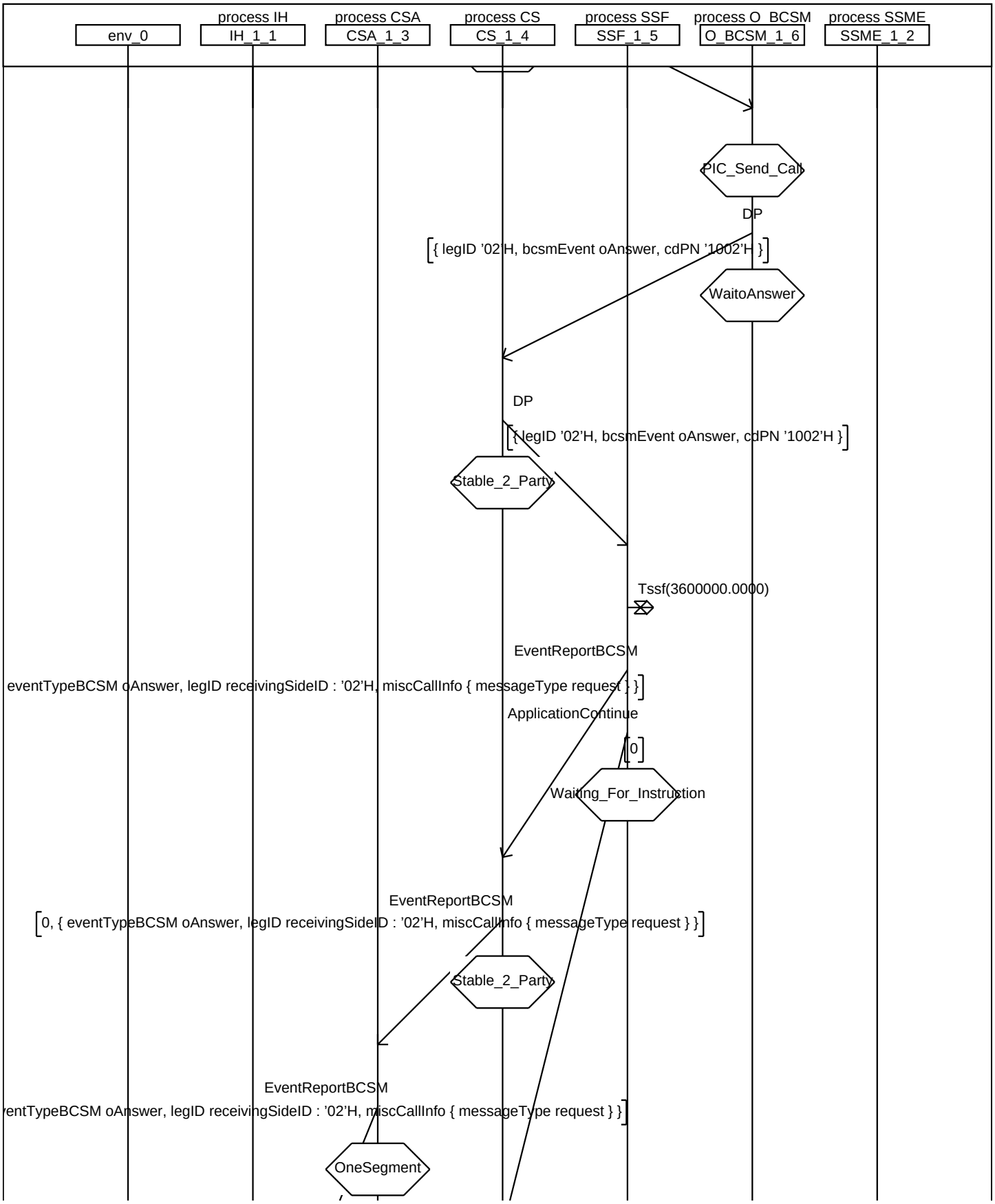
Annex A: CO_Detailed



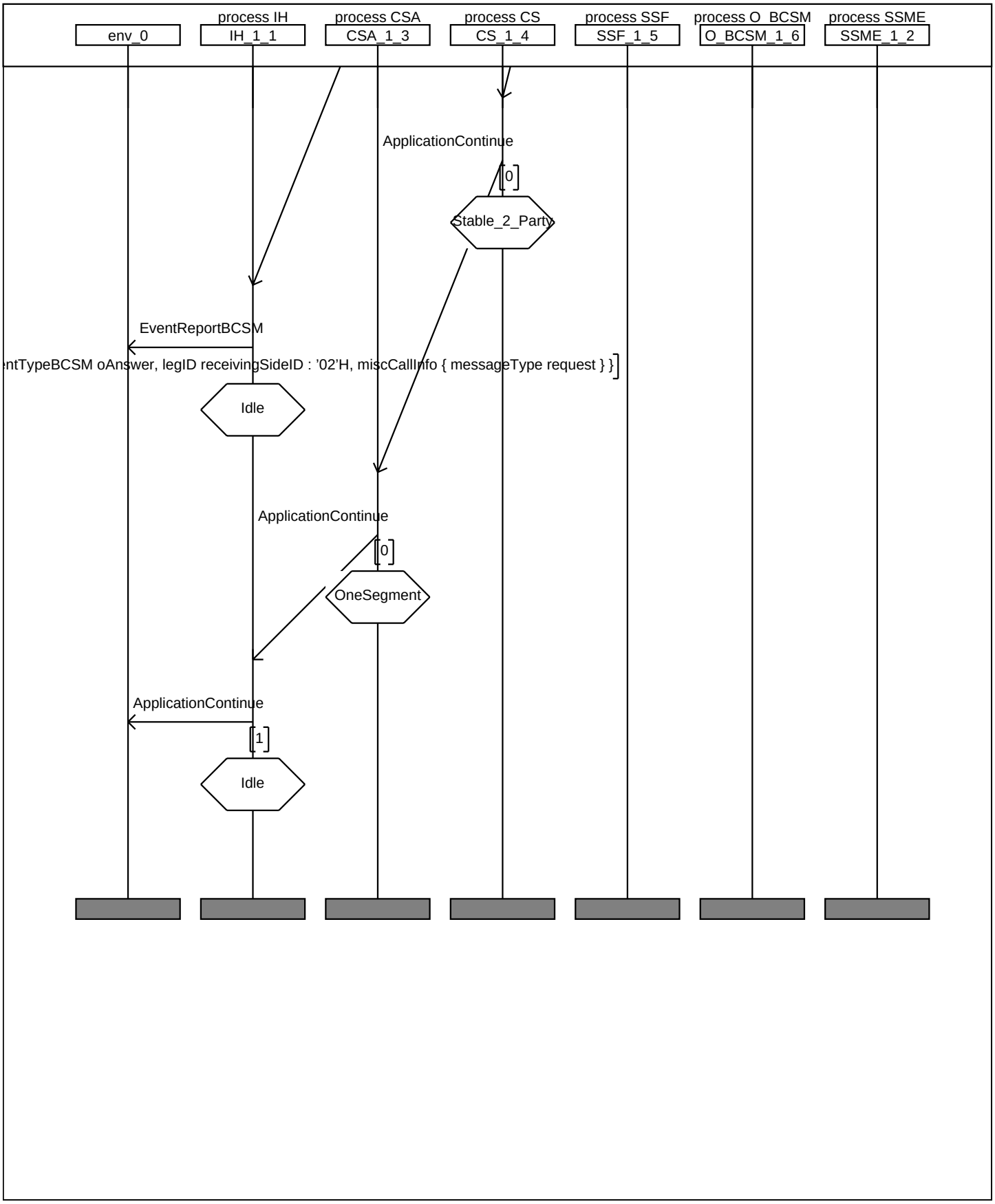
Annex A: CO_Detailed



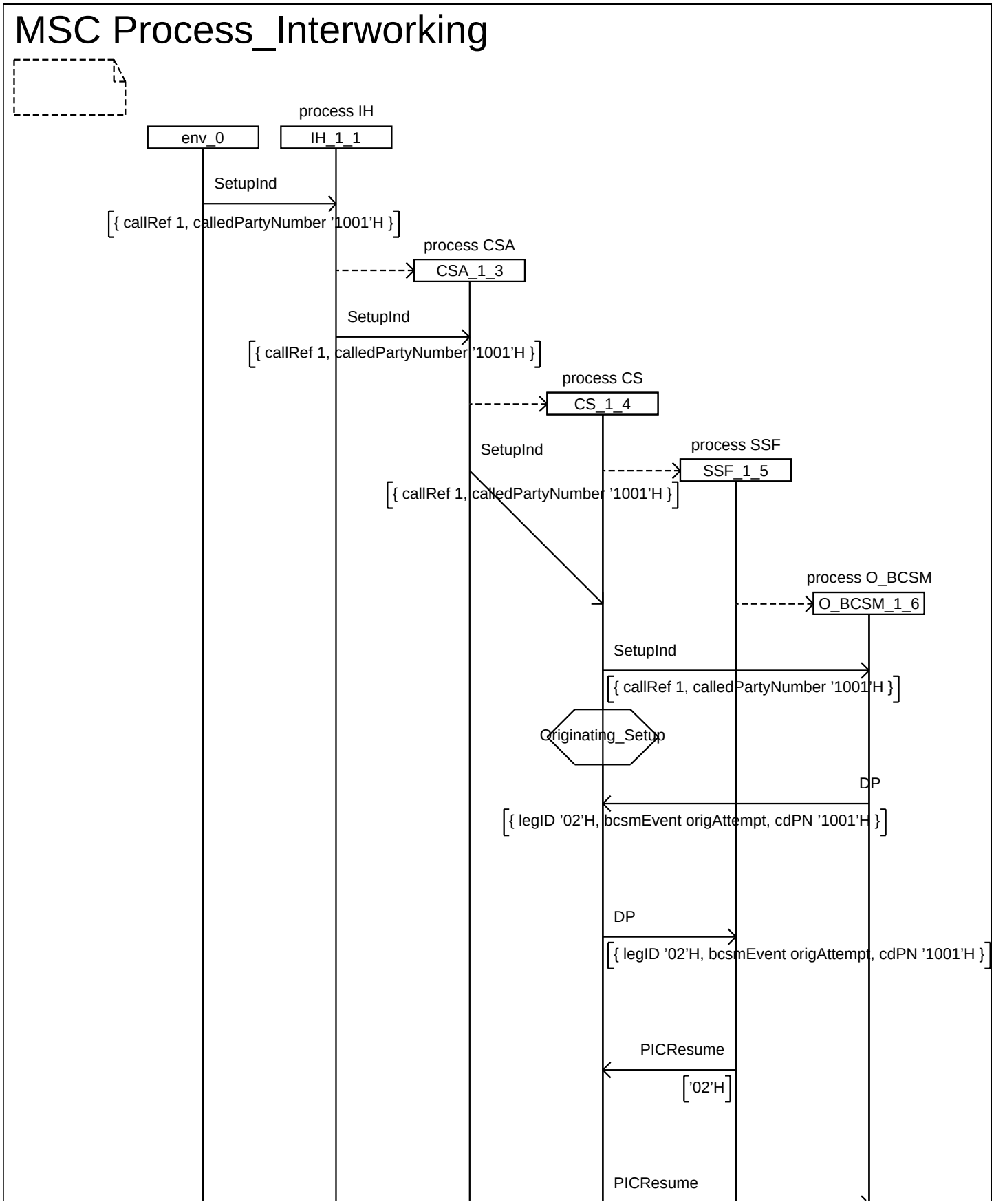
Annex A: CO_Detailed



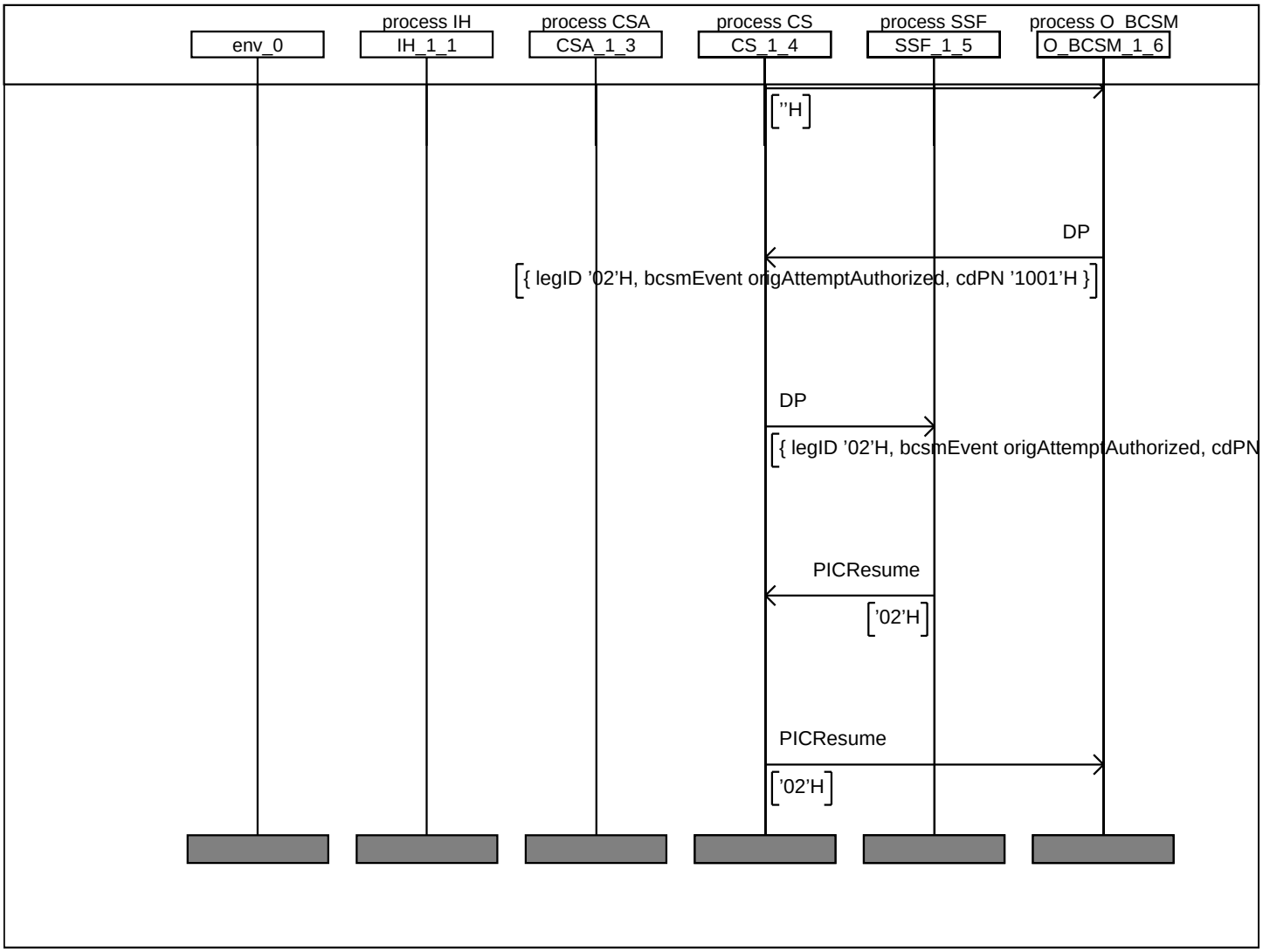
Annex A: CO_Detailed



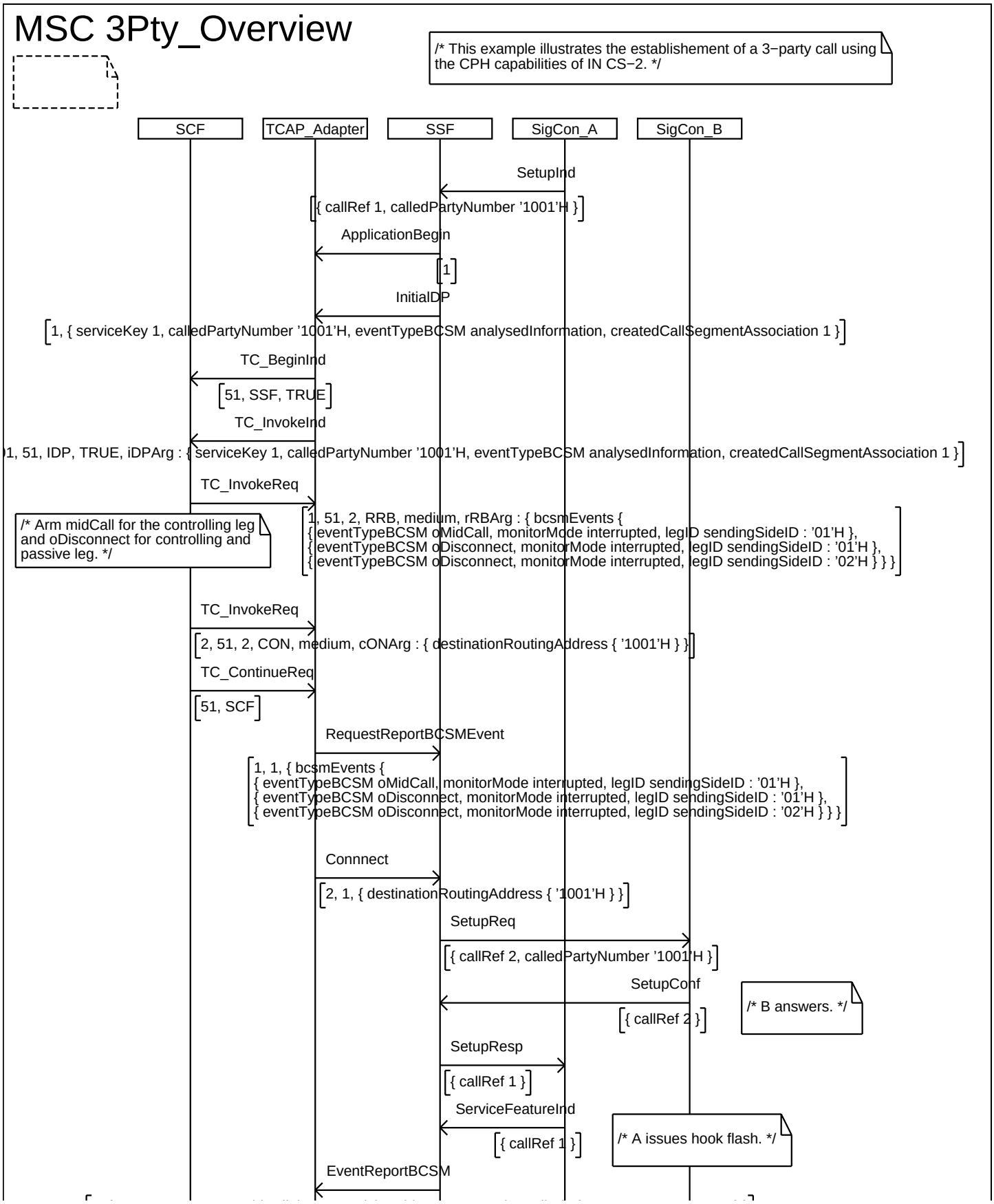
Annex A: Process_Interworking



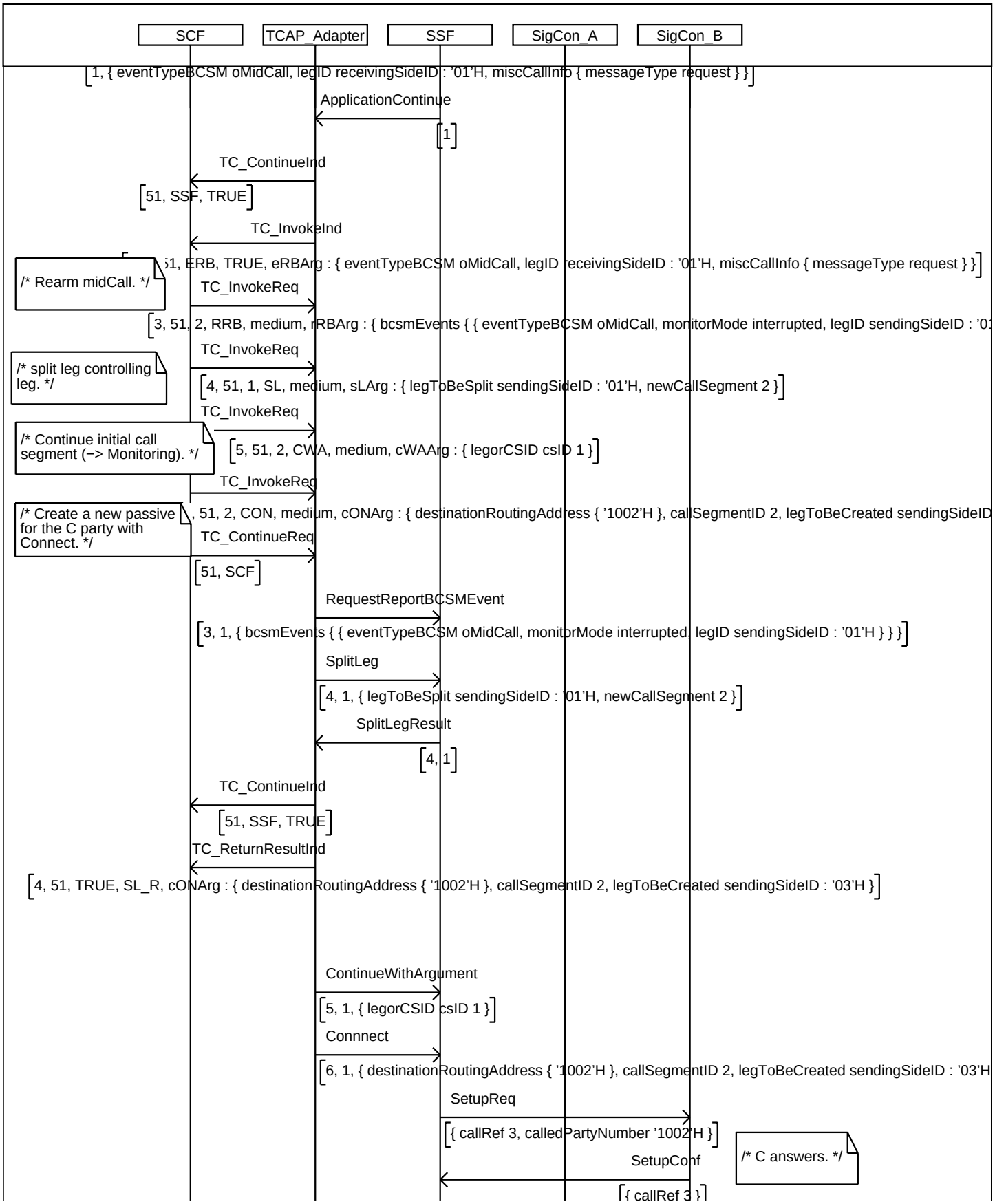
Annex A: Process_Interworking



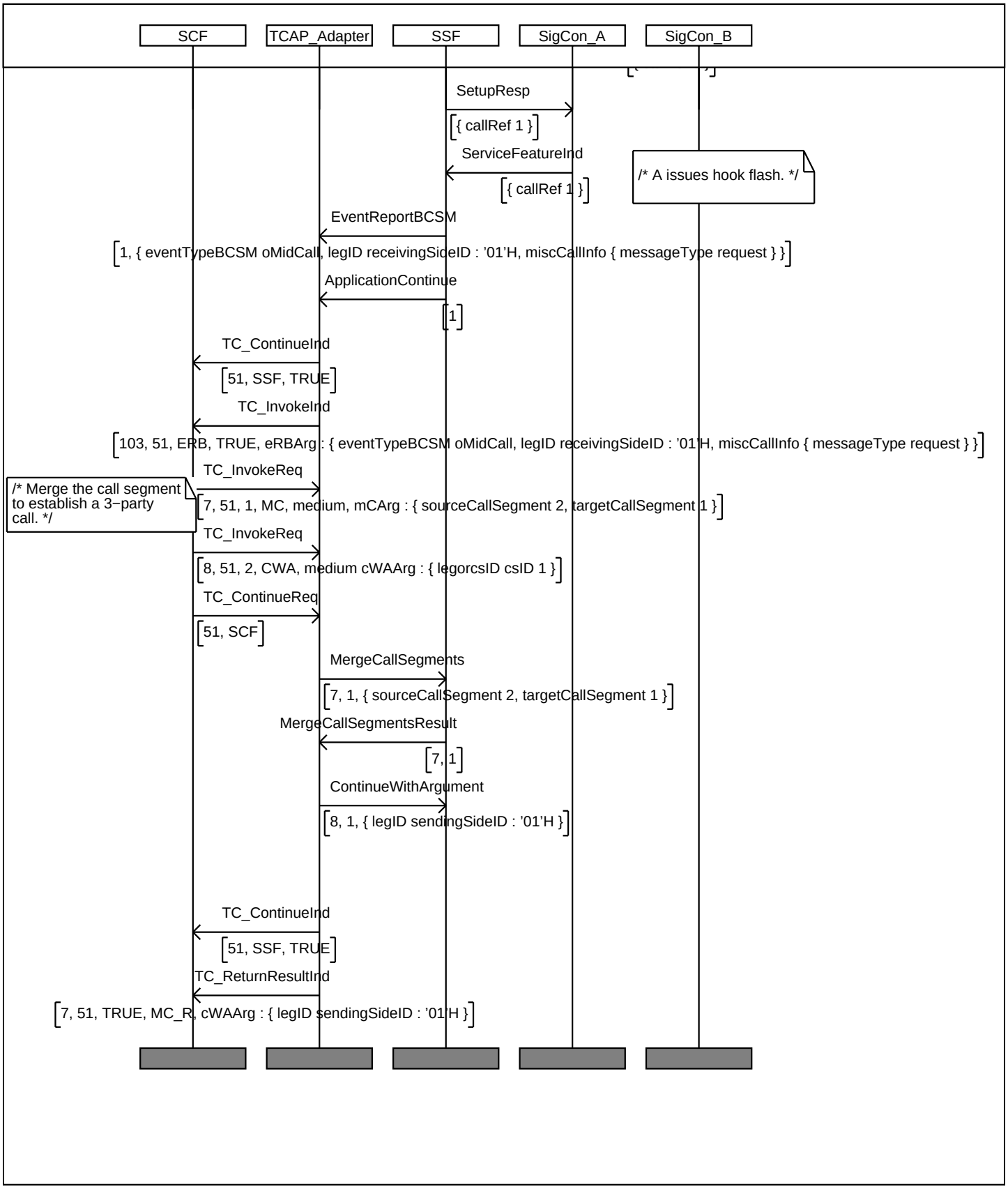
Annex A: 3Pty_Overview



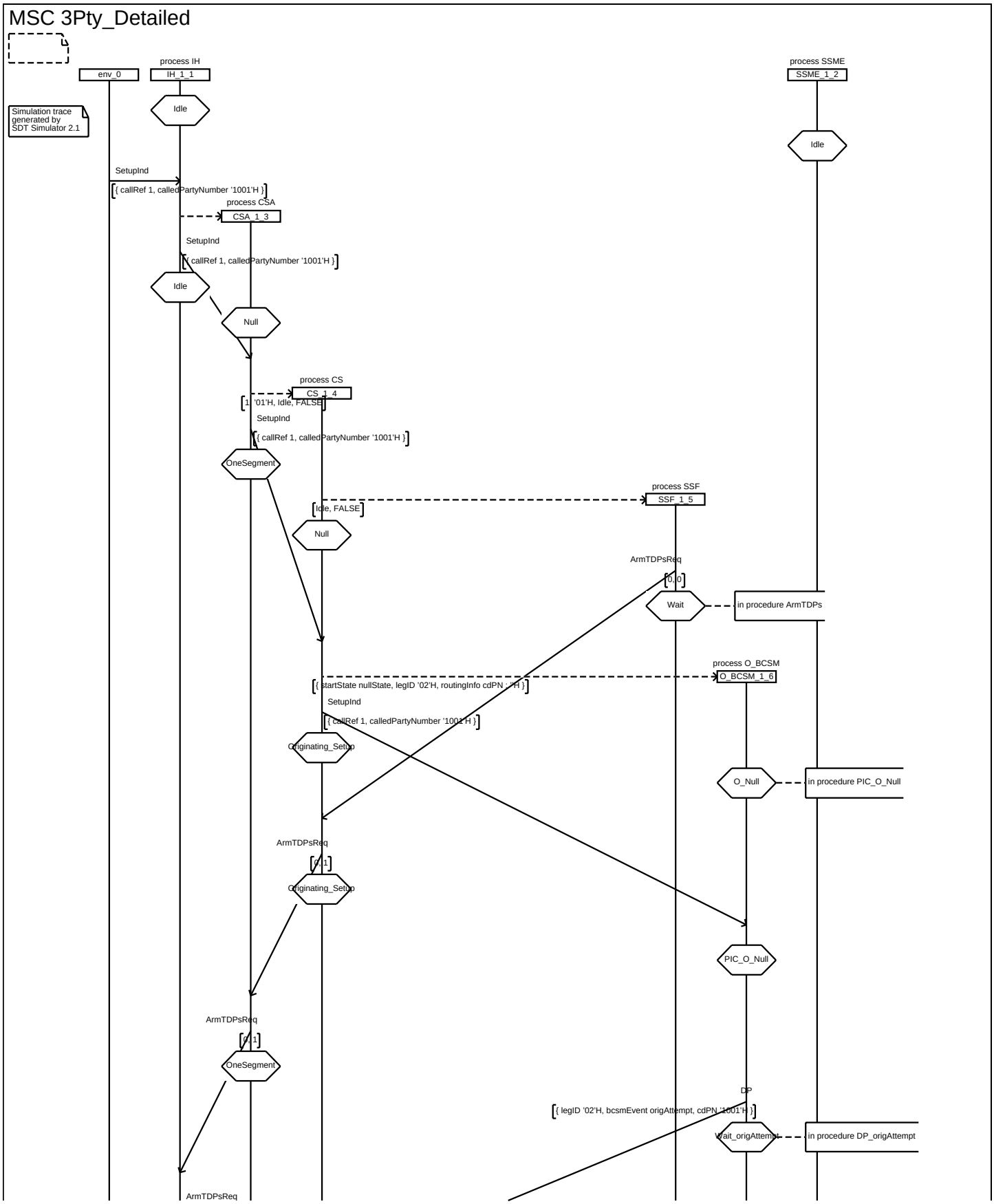
Annex A: 3Pty_Overview



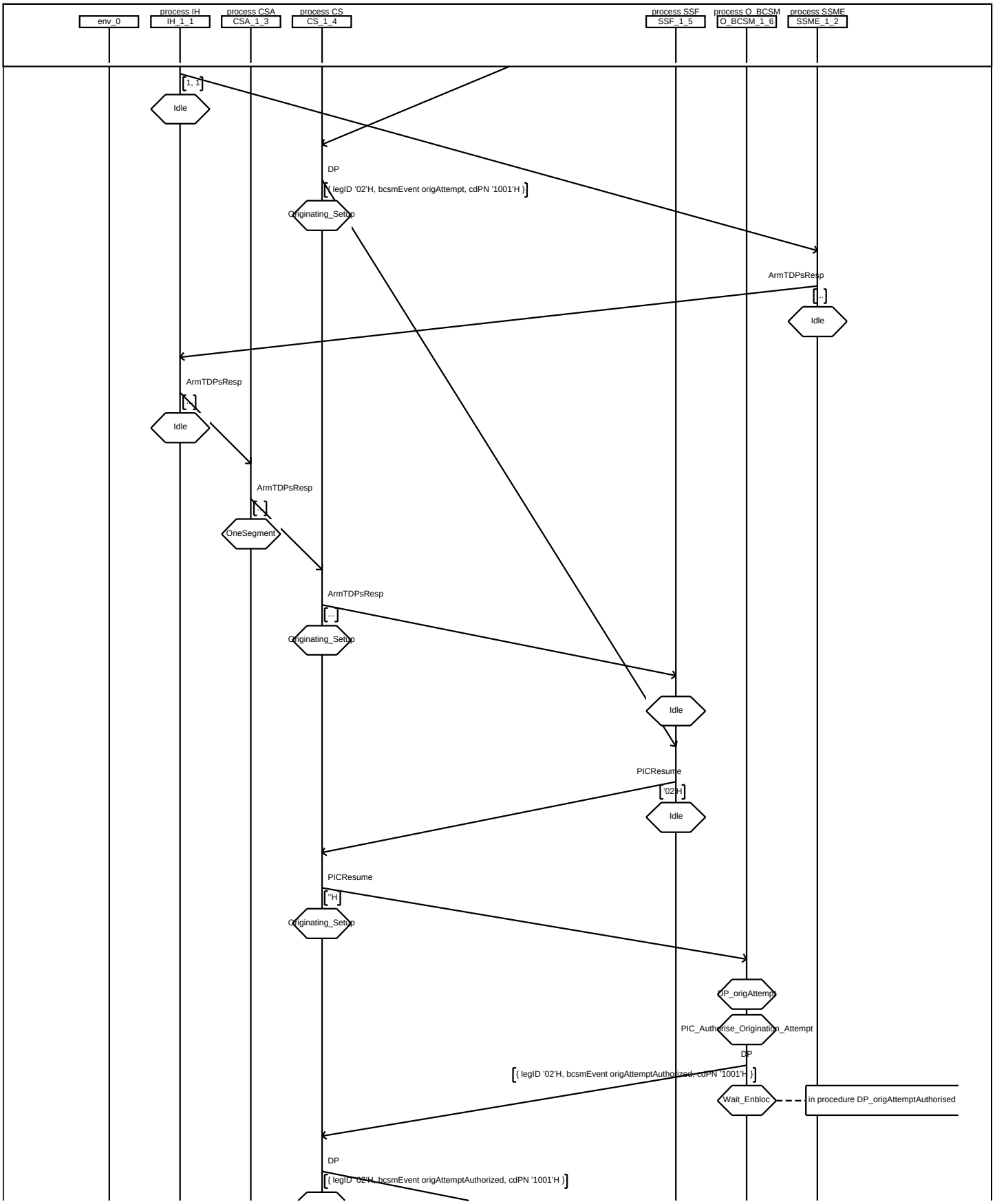
Annex A: 3Pty_Overview



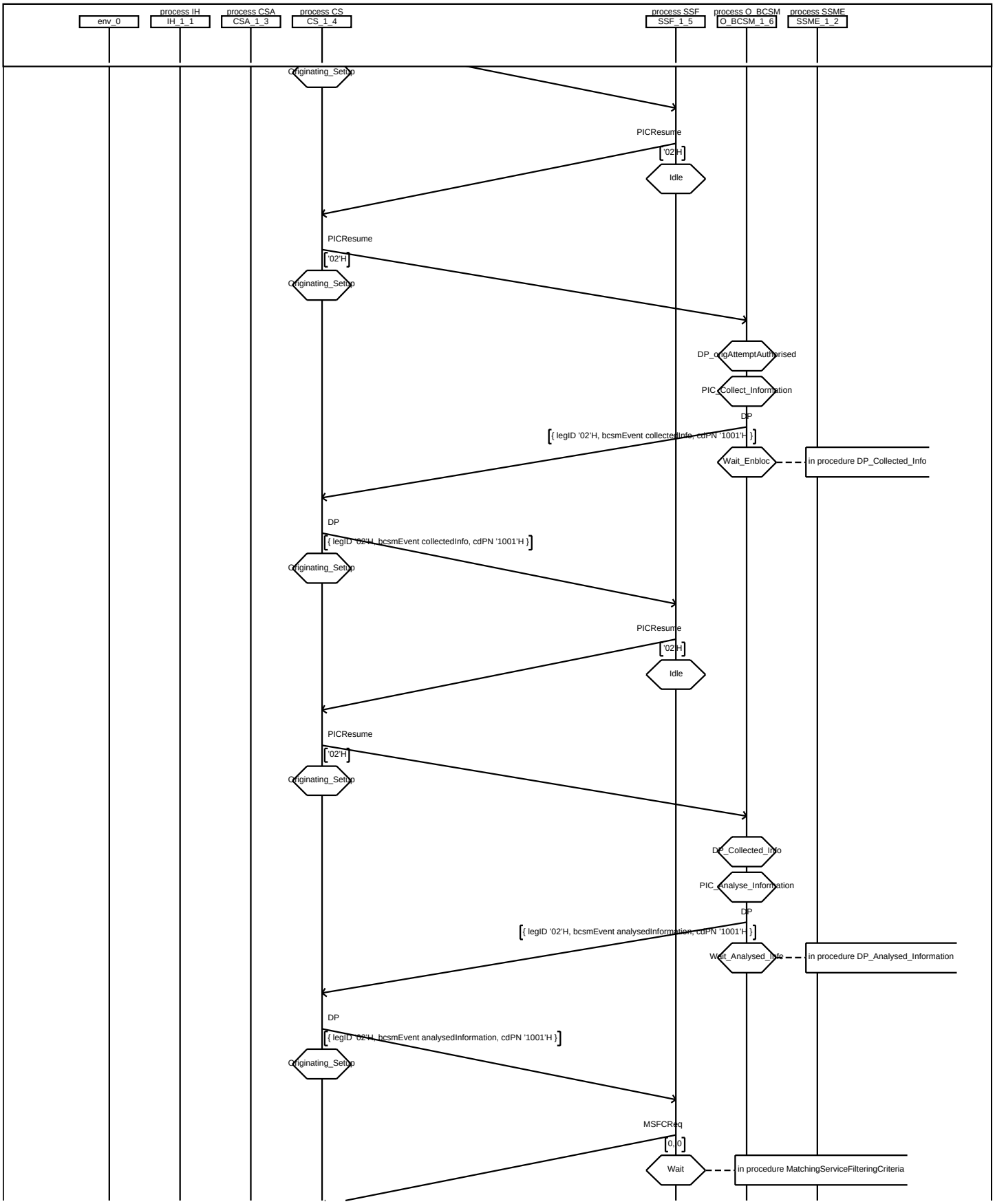
Annex A: 3Pty_Detailed



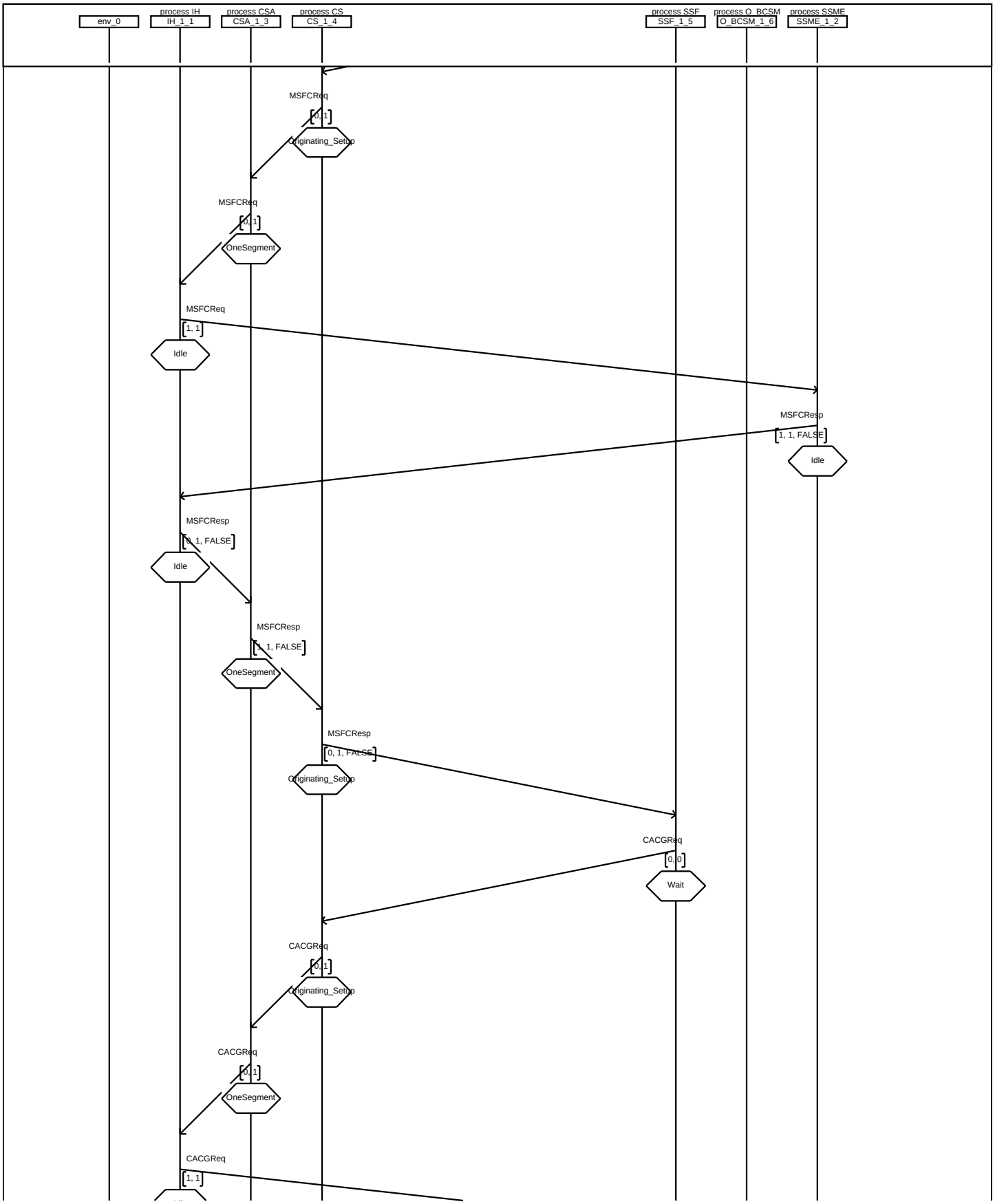
Annex A: 3Pty_Detailed



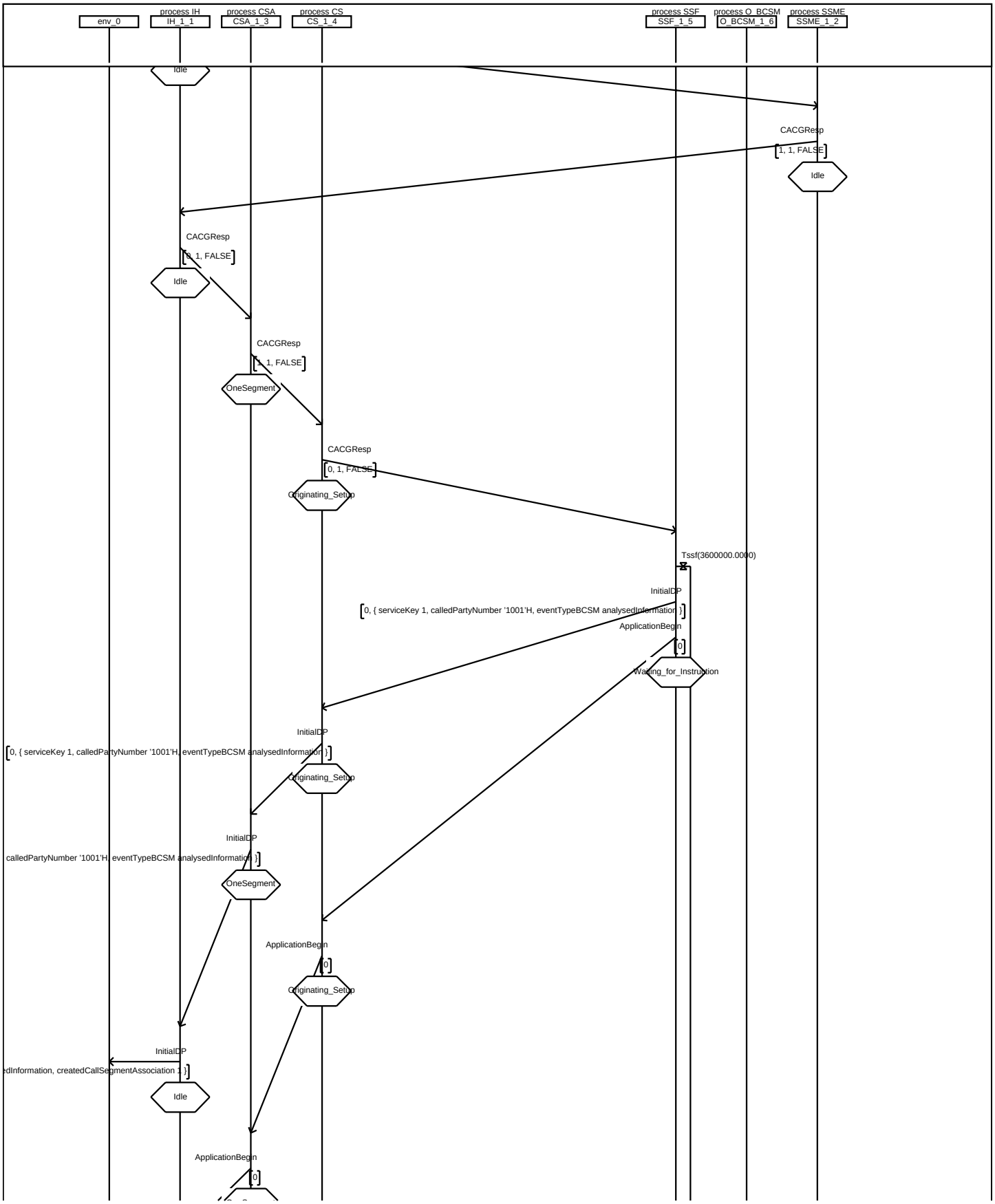
Annex A: 3Pty_Detailed



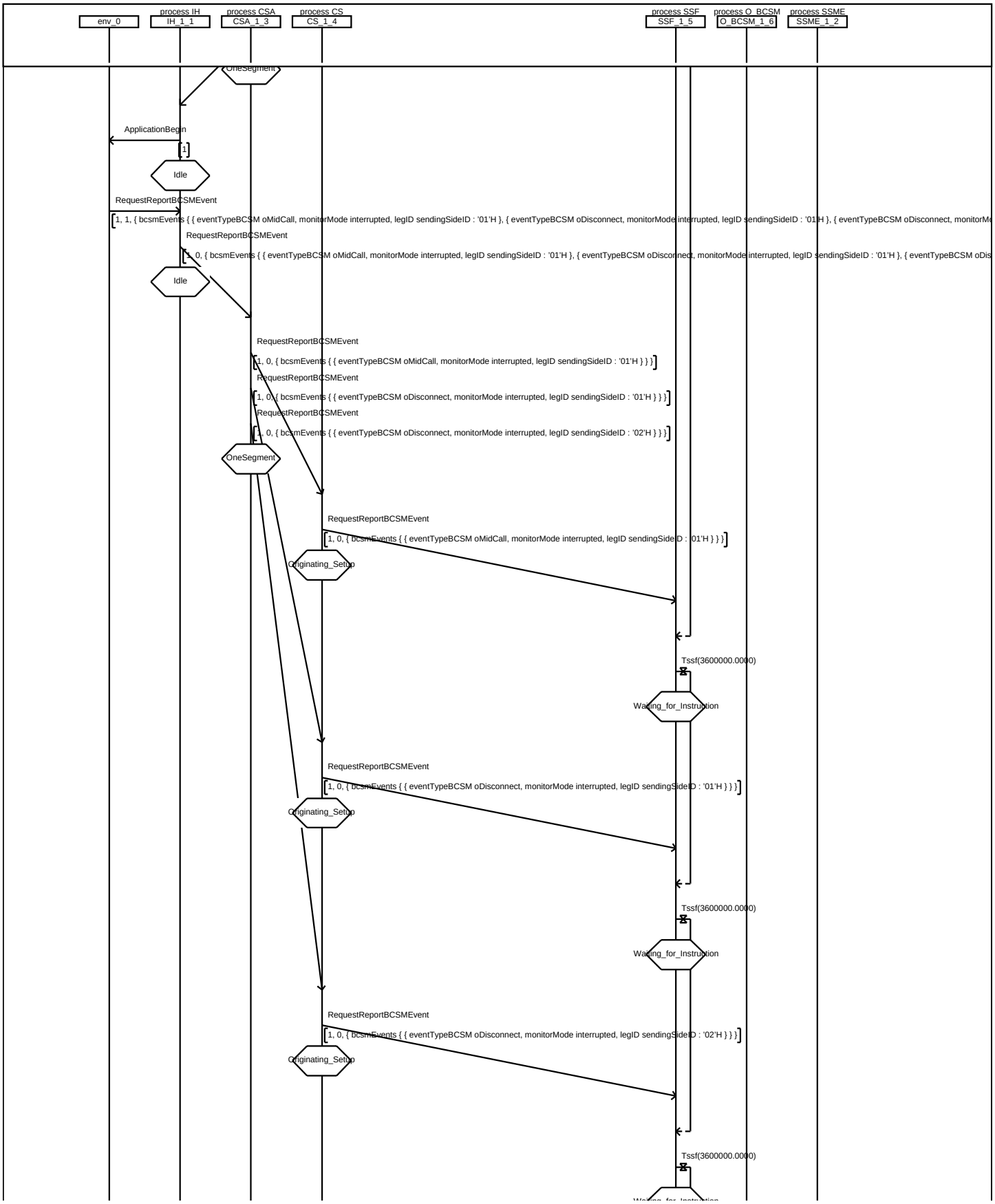
Annex A: 3Pty_Detailed



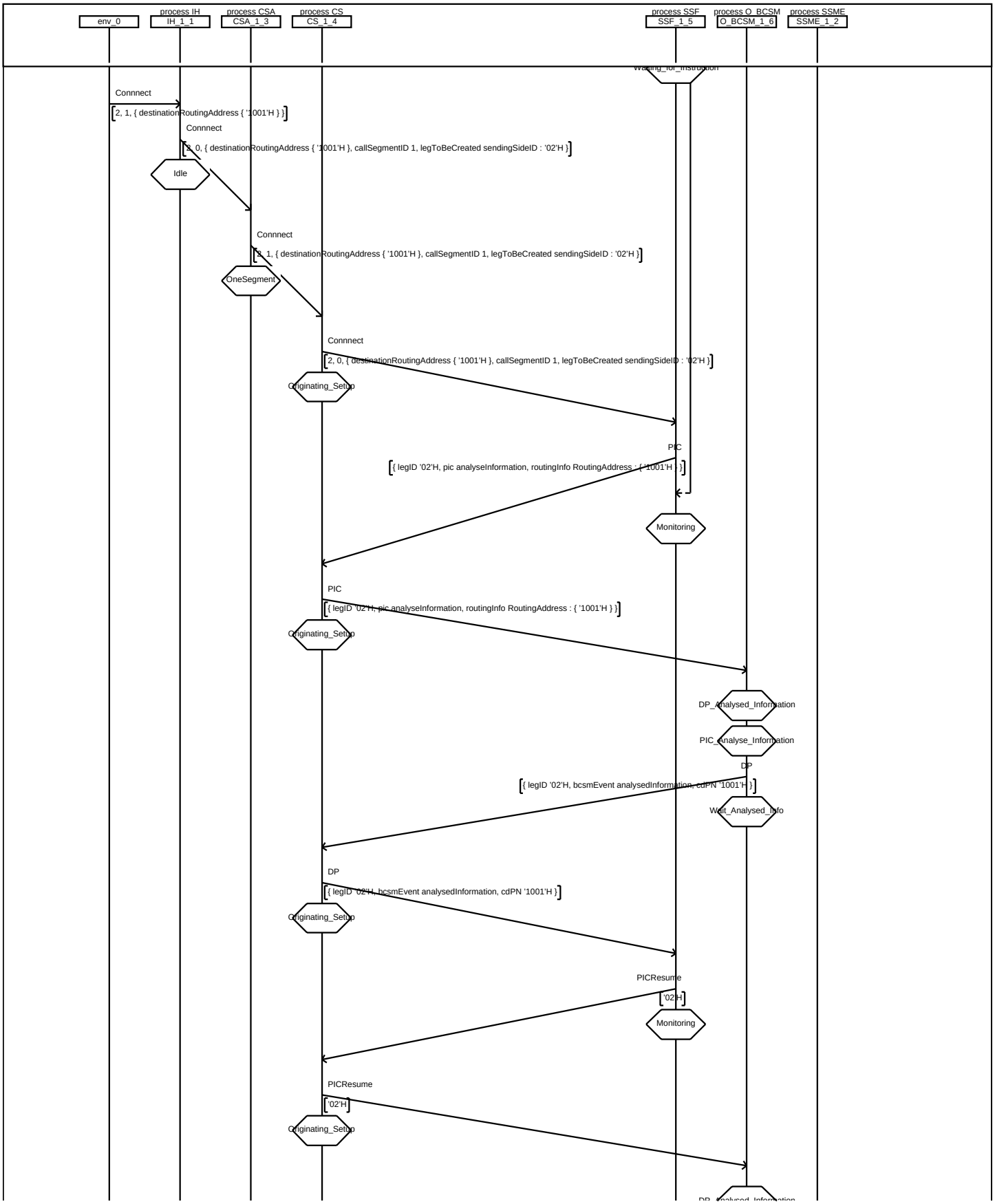
Annex A: 3Pty_Detailed



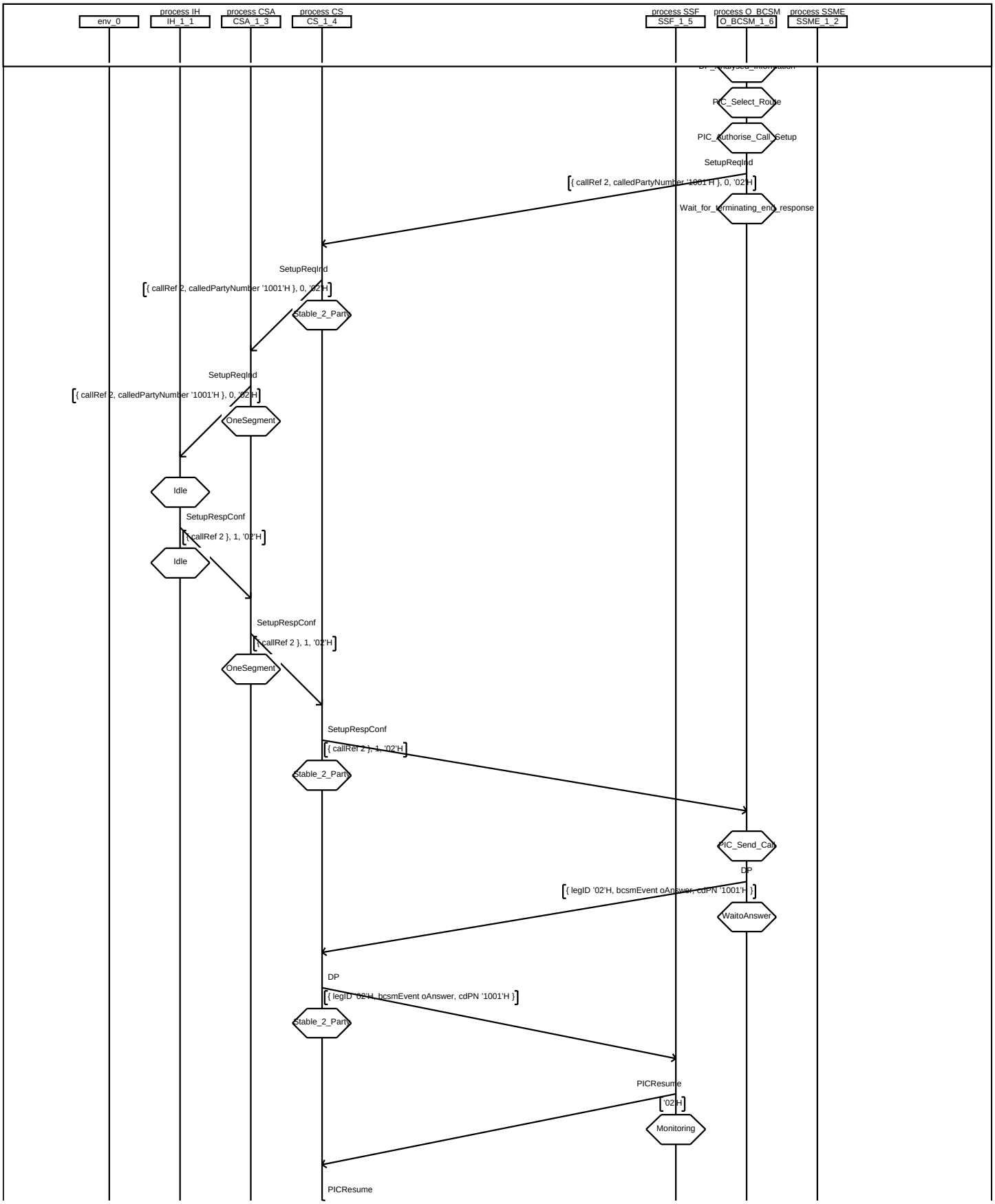
Annex A: 3Pty_Detailed



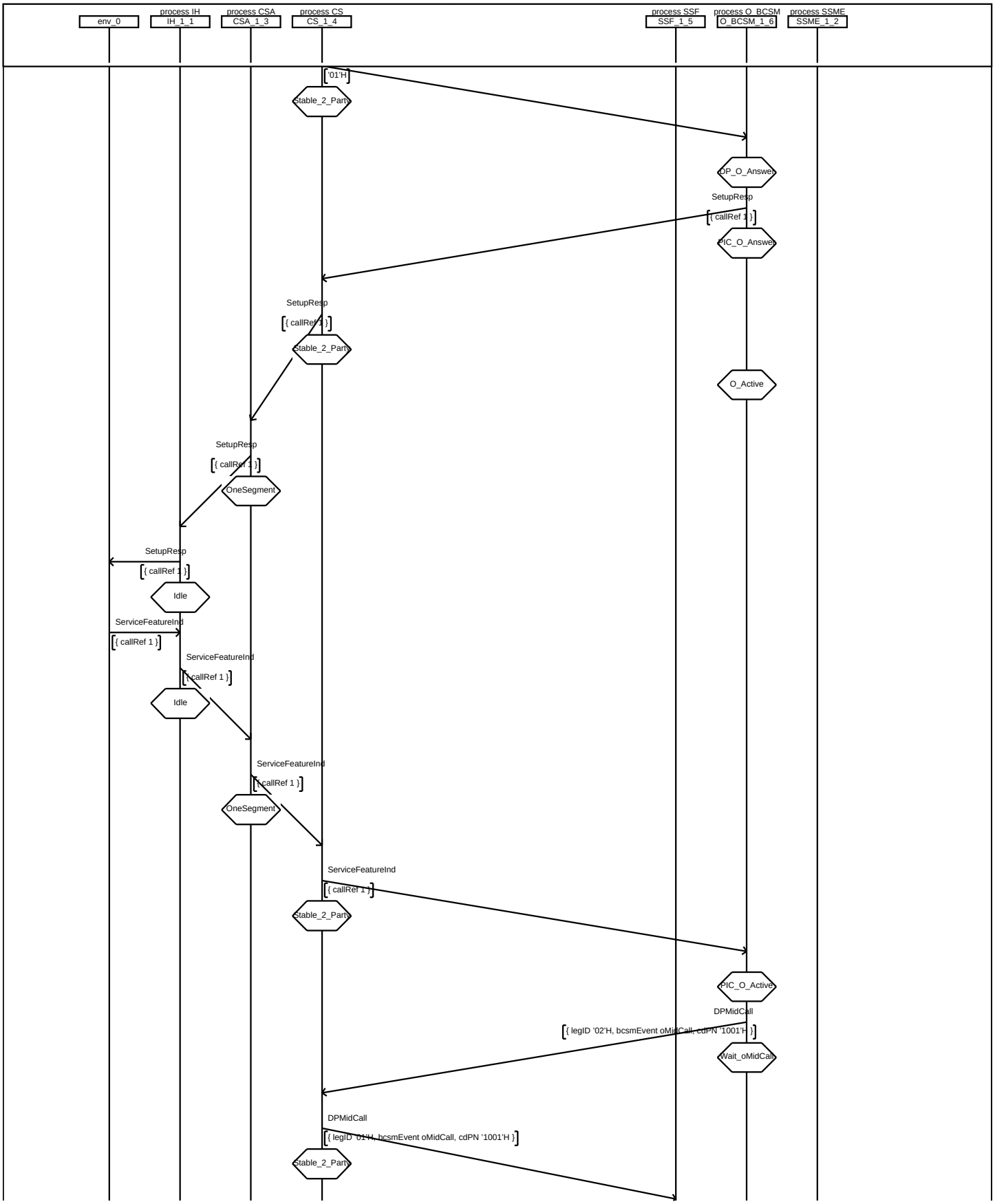
Annex A: 3Pty_Detailed



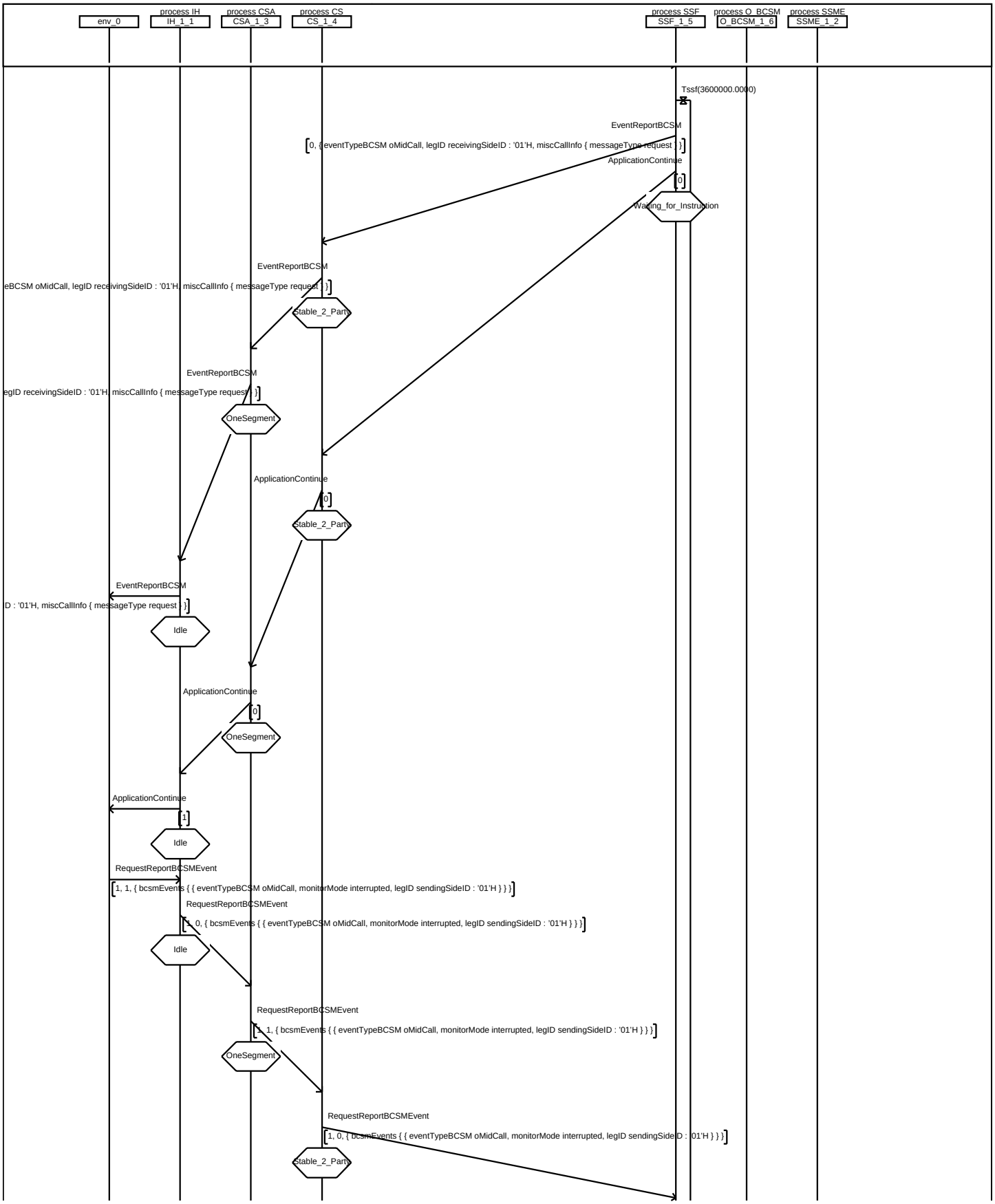
Annex A: 3Pty_Detailed



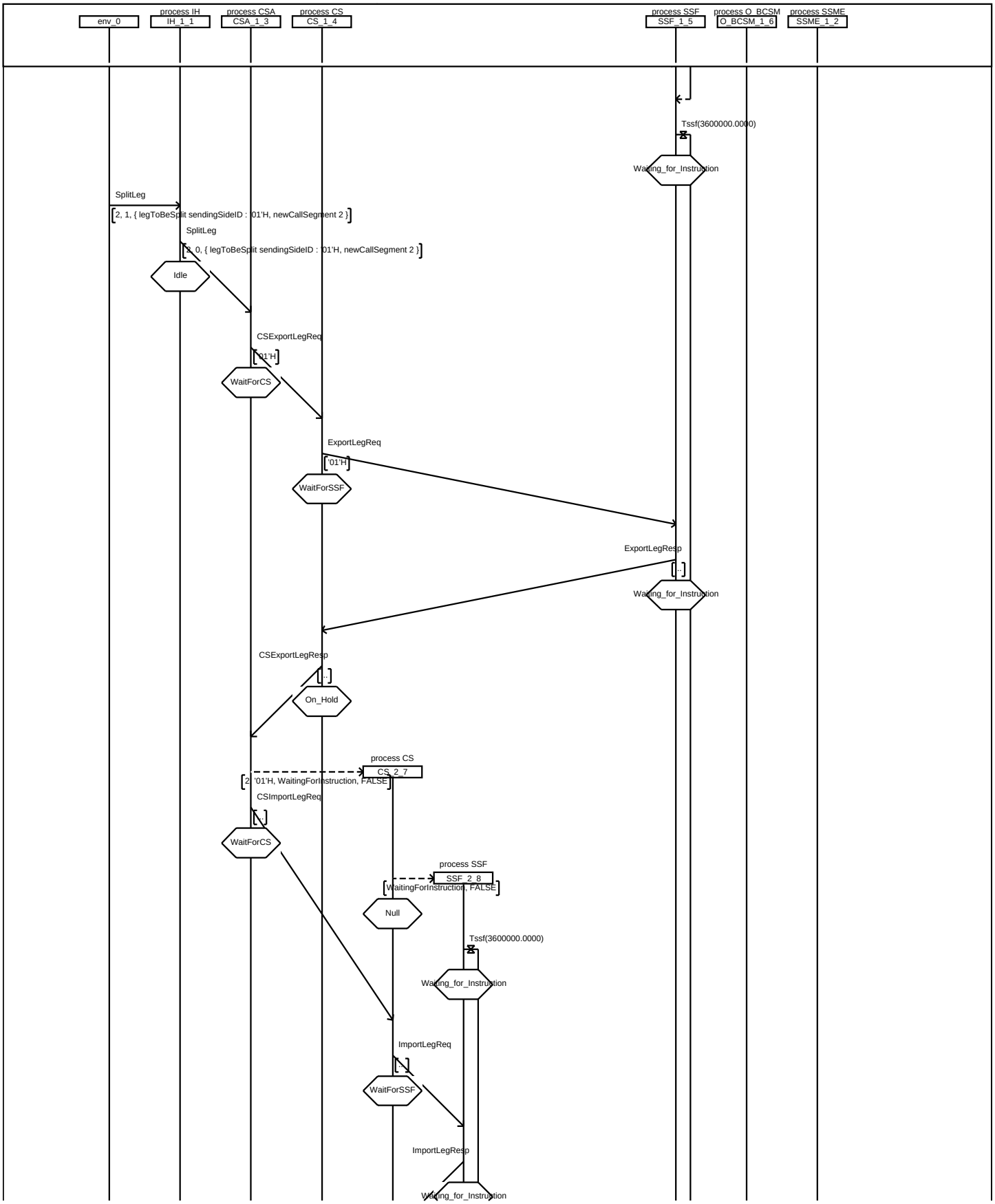
Annex A: 3Pty_Detailed



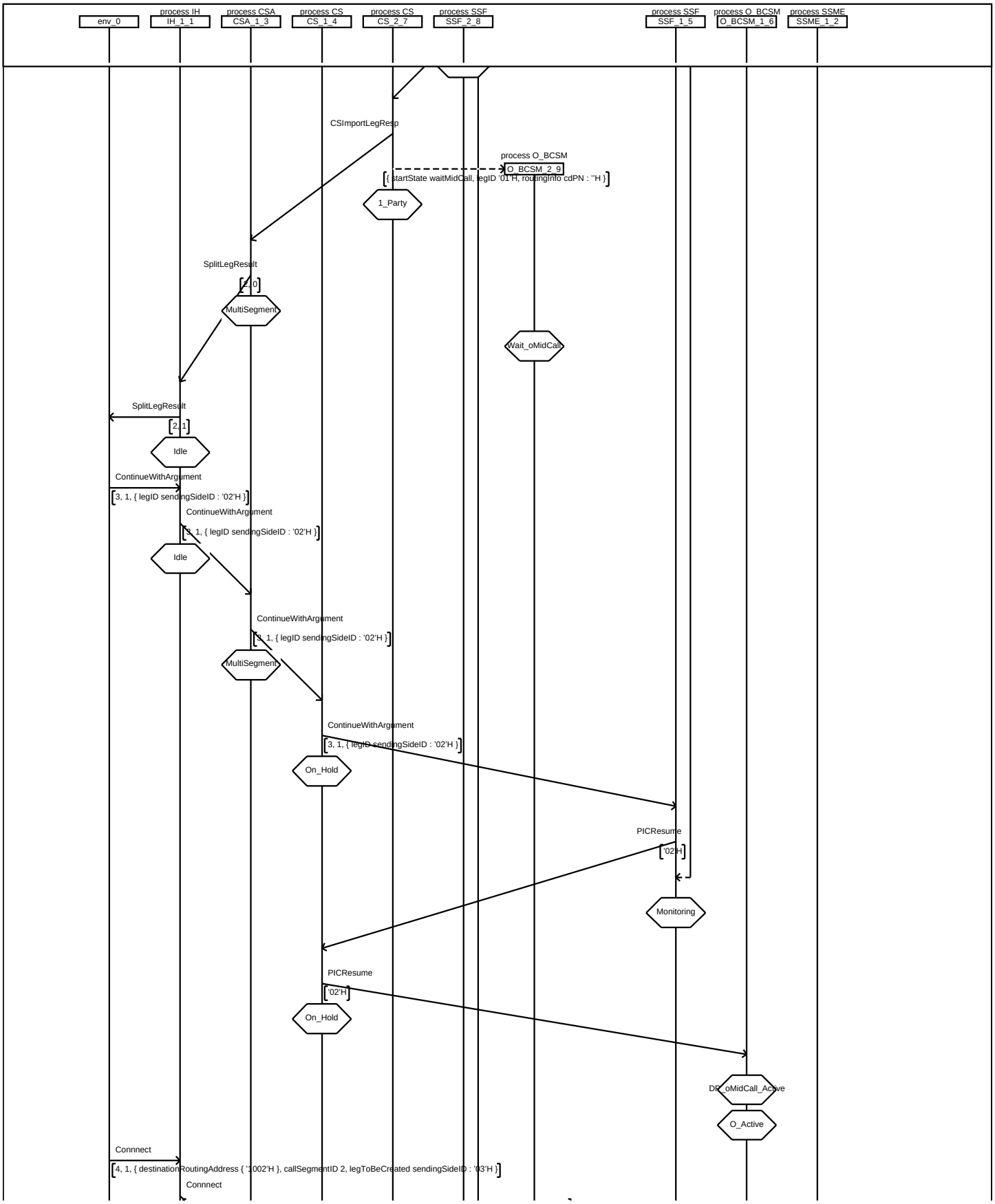
Annex A: 3Pty_Detailed



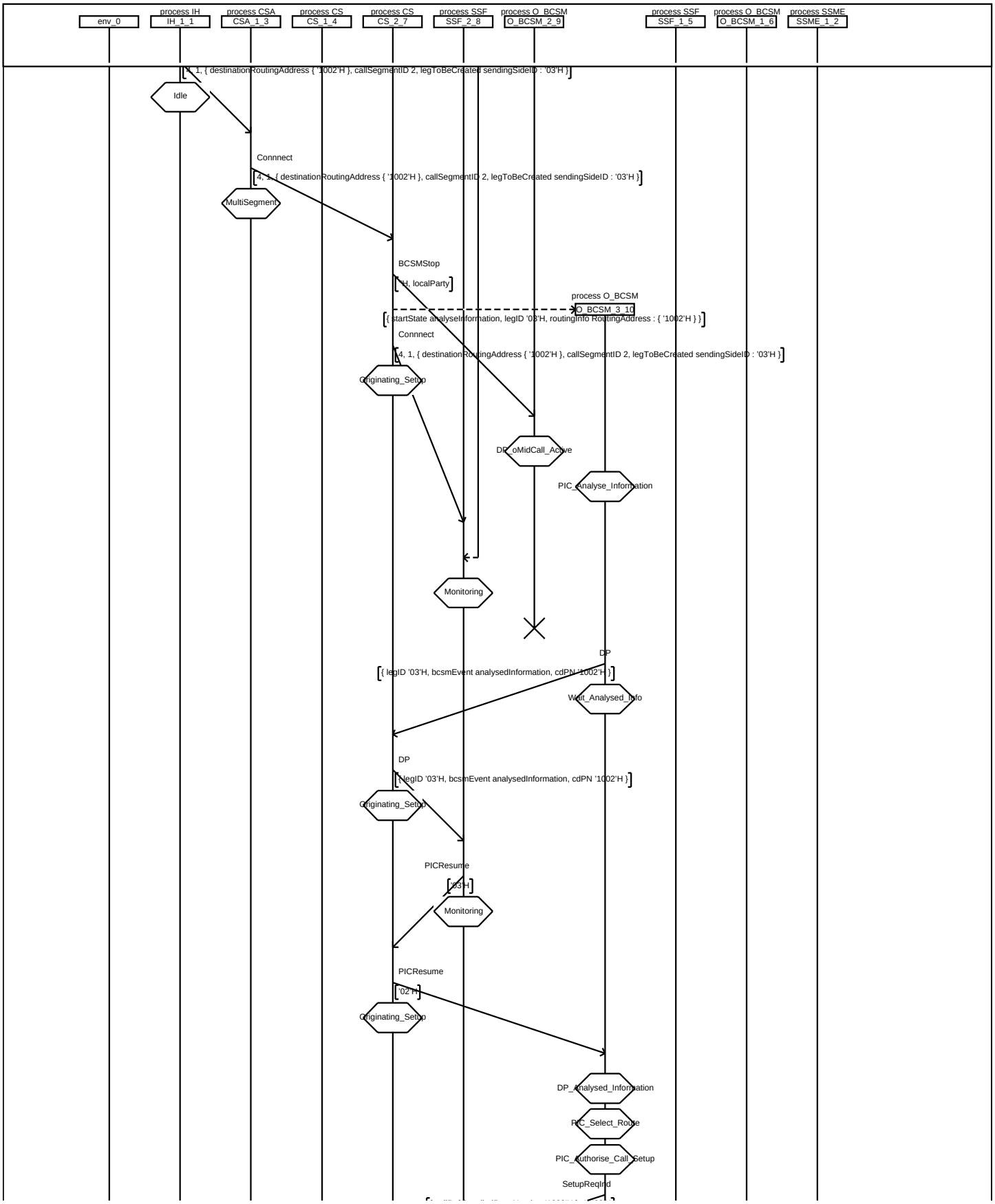
Annex A: 3Pty_Detailed



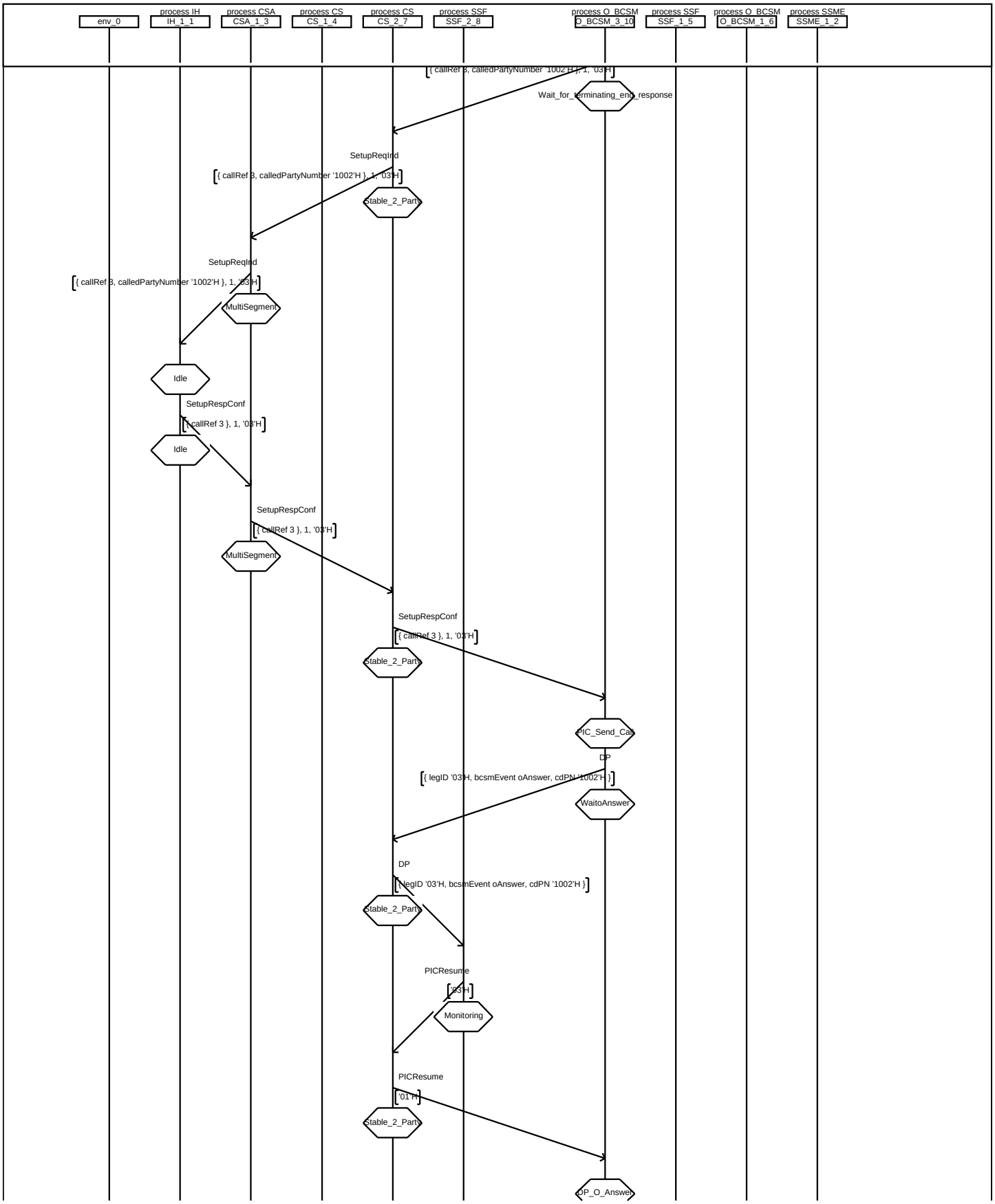
Annex A: 3Pty_Detailed



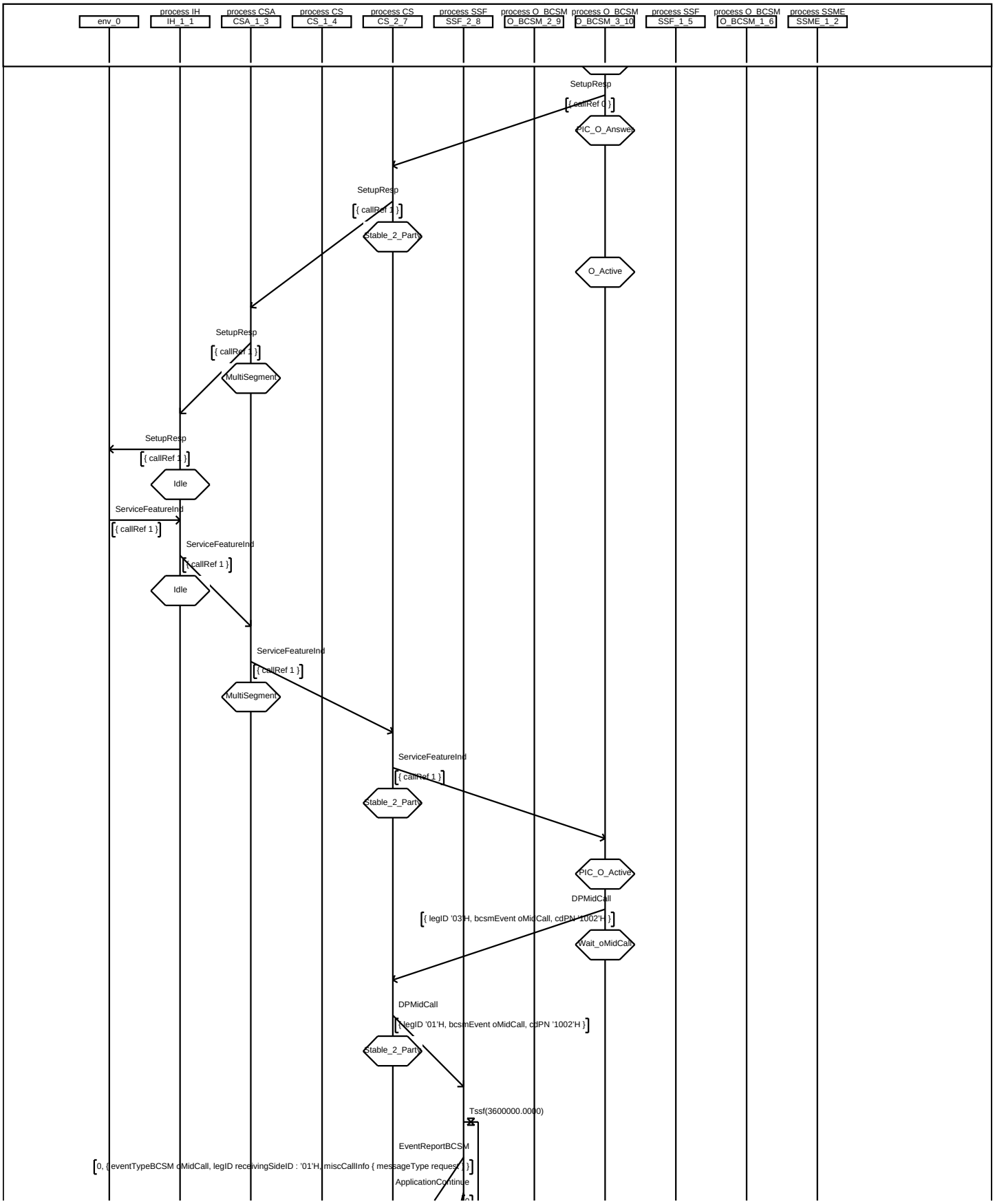
Annex A: 3Pty_Detailed



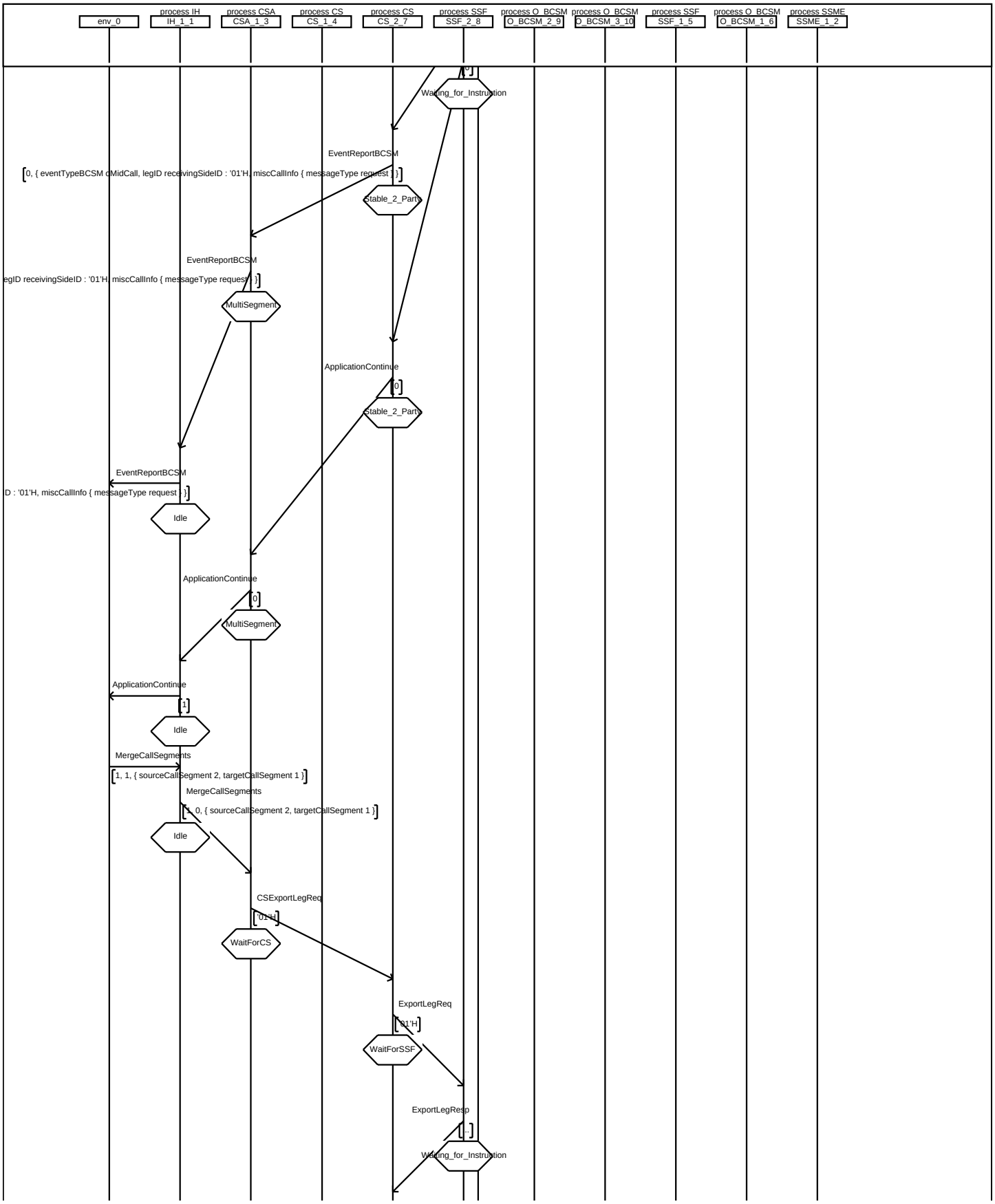
Annex A: 3Pty_Detailed



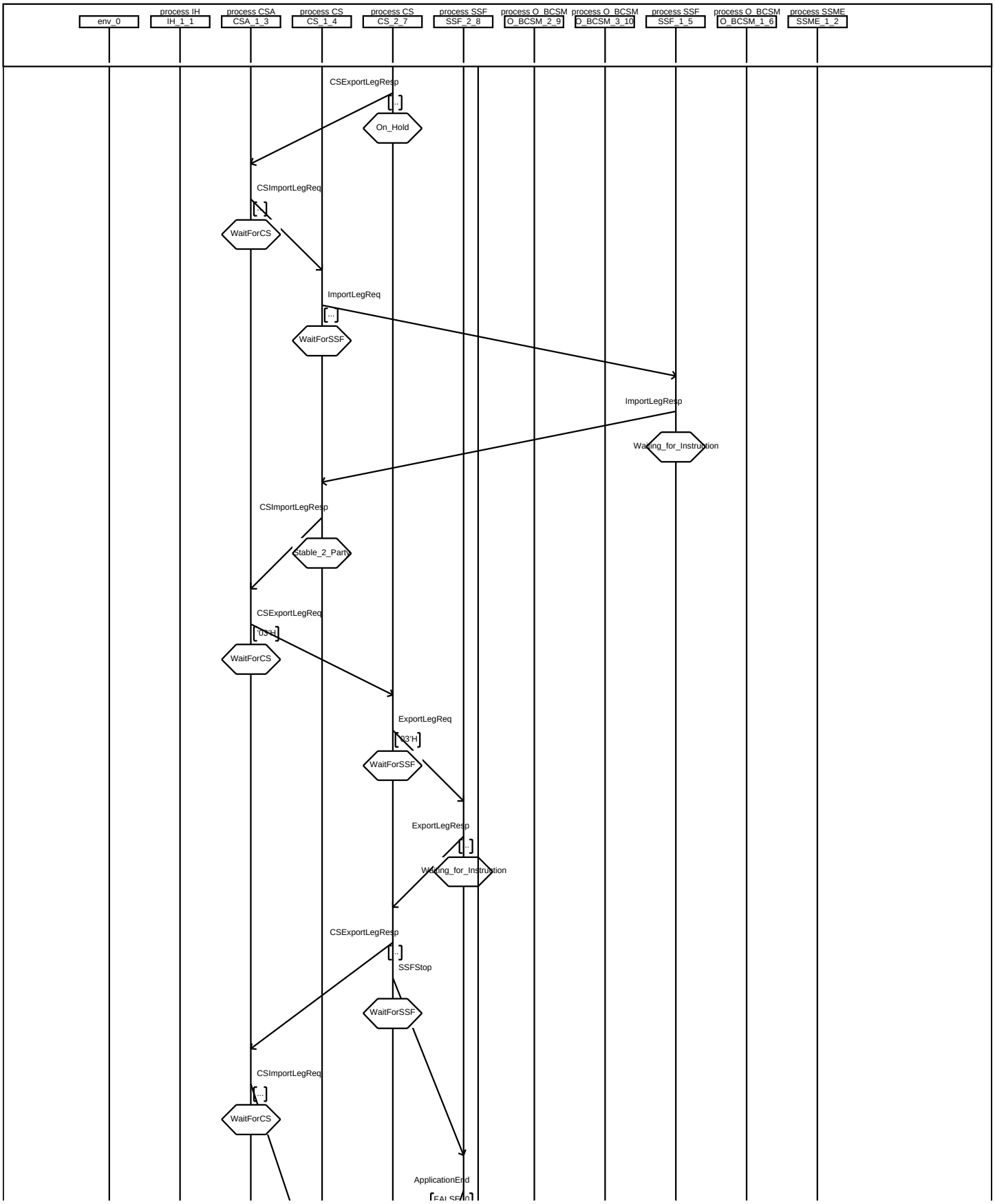
Annex A: 3Pty_Detailed



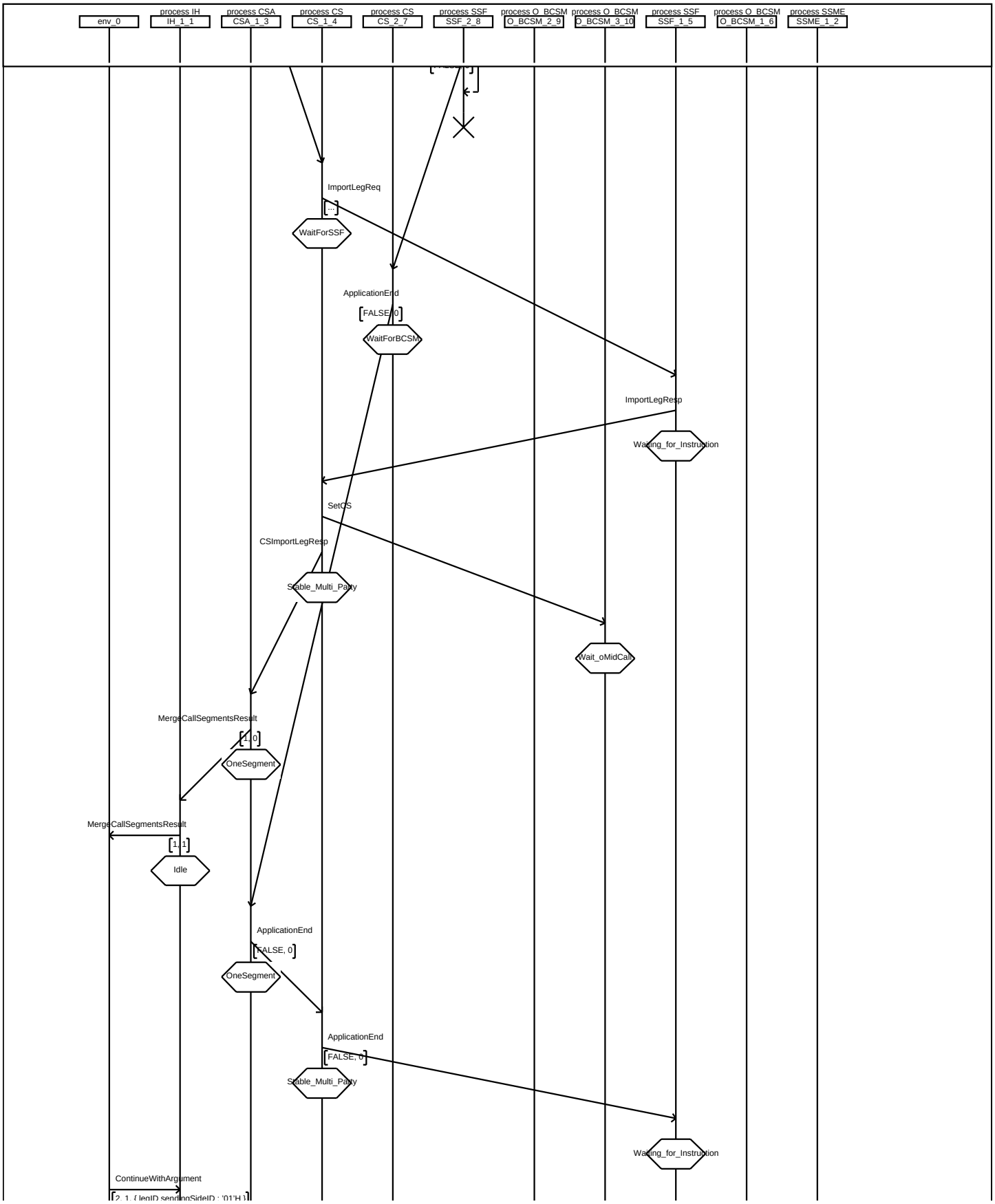
Annex A: 3Pty_Detailed



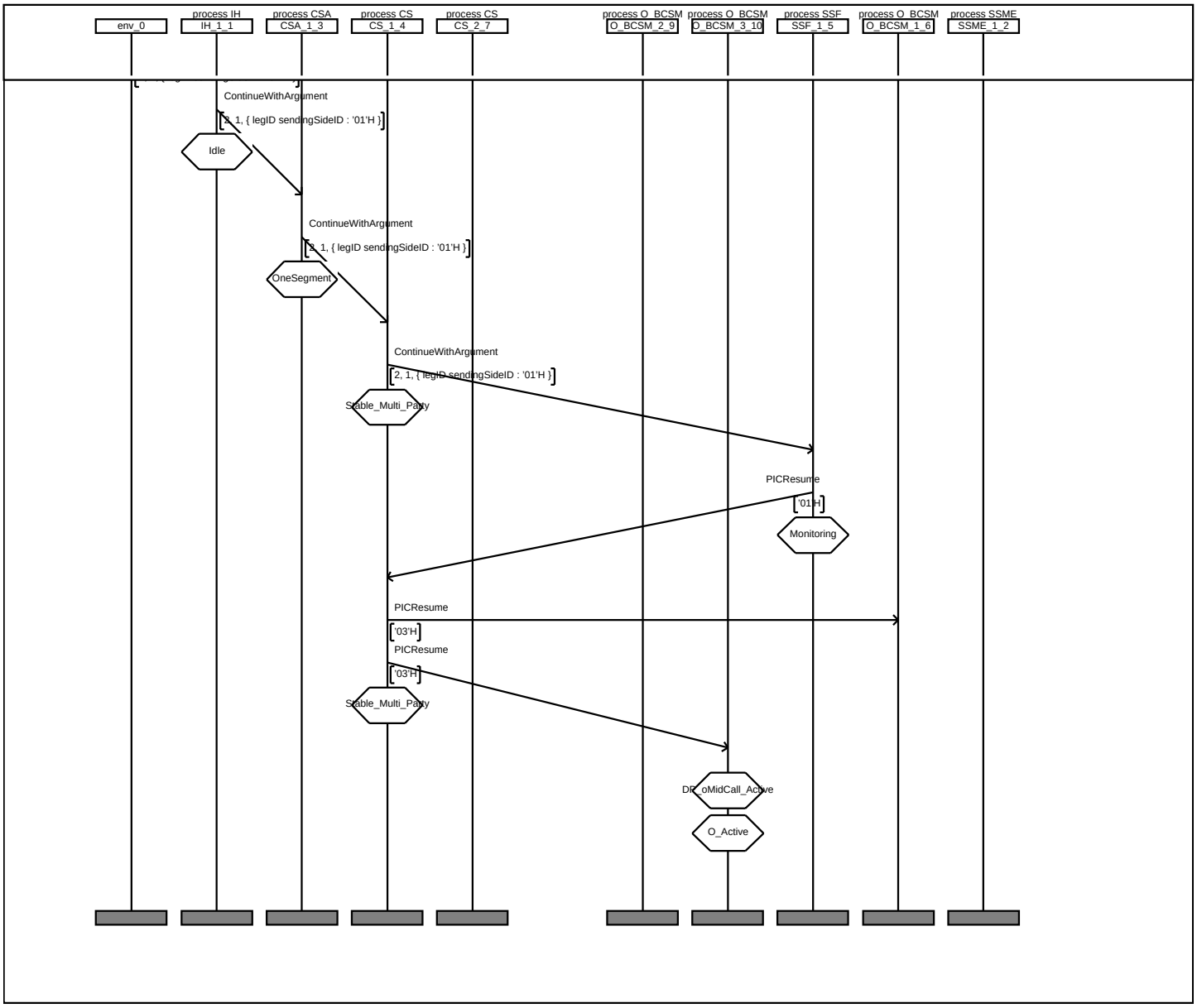
Annex A: 3Pty_Detailed



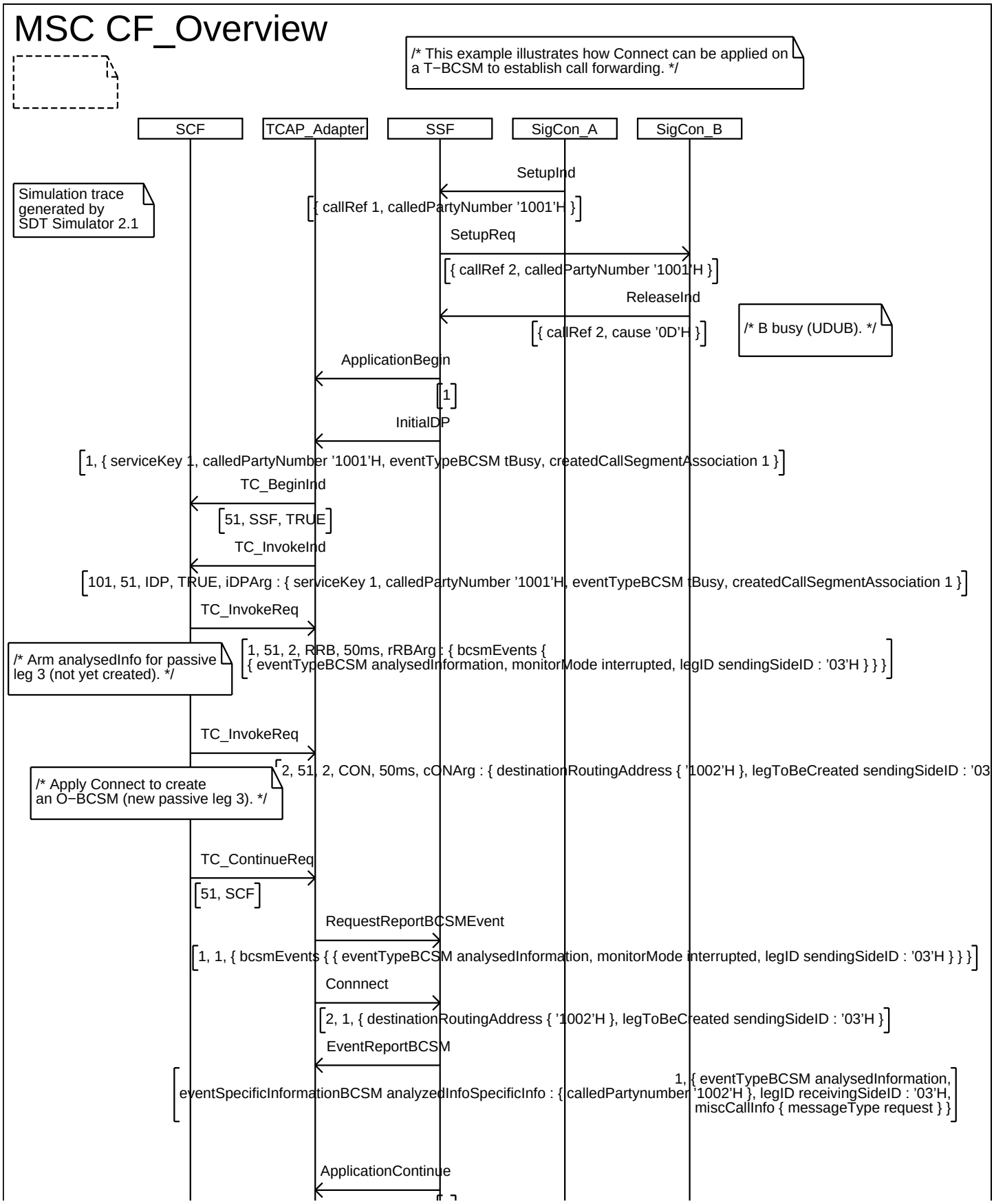
Annex A: 3Pty_Detailed



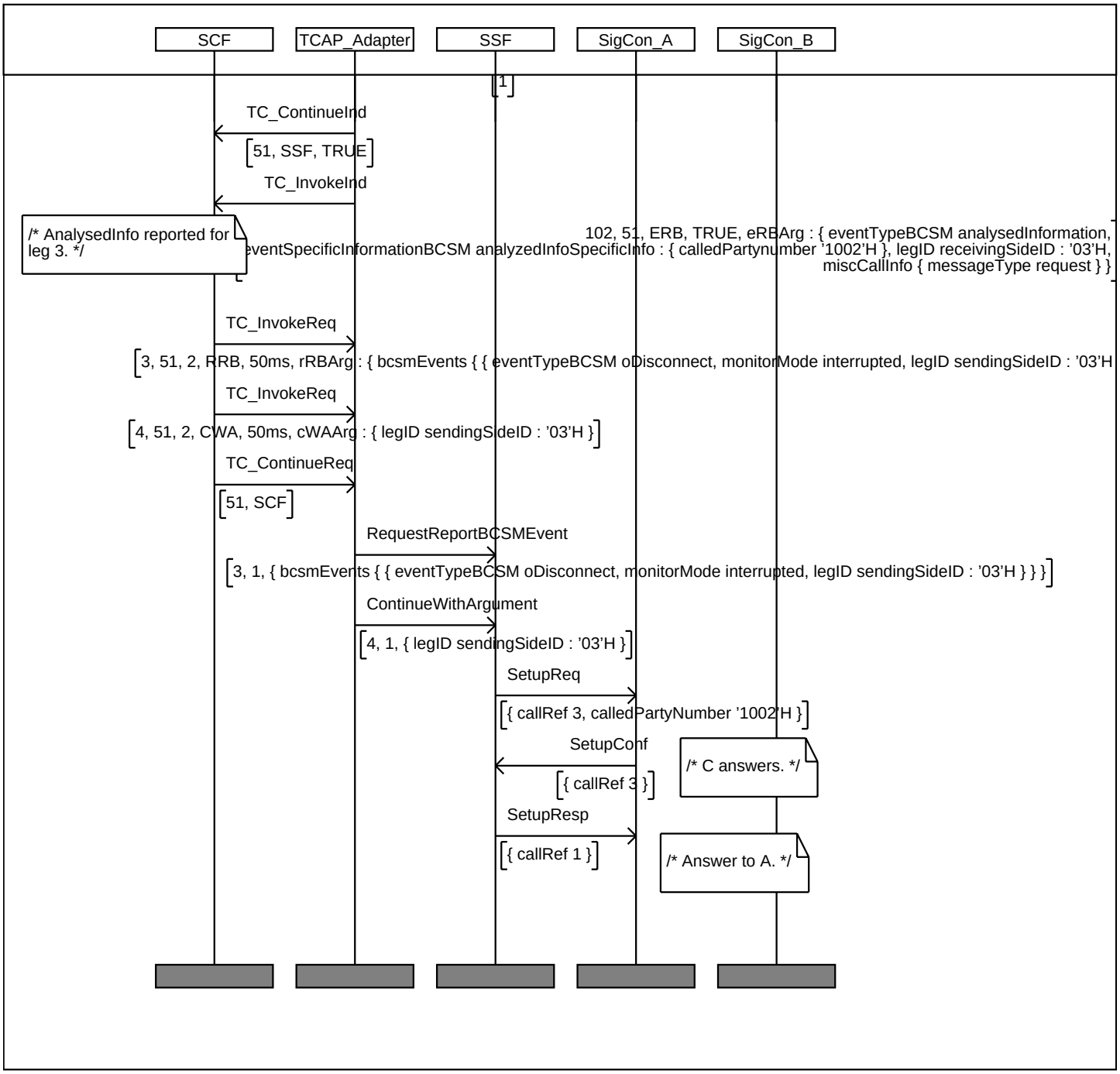
Annex A: 3Pty_Detailed



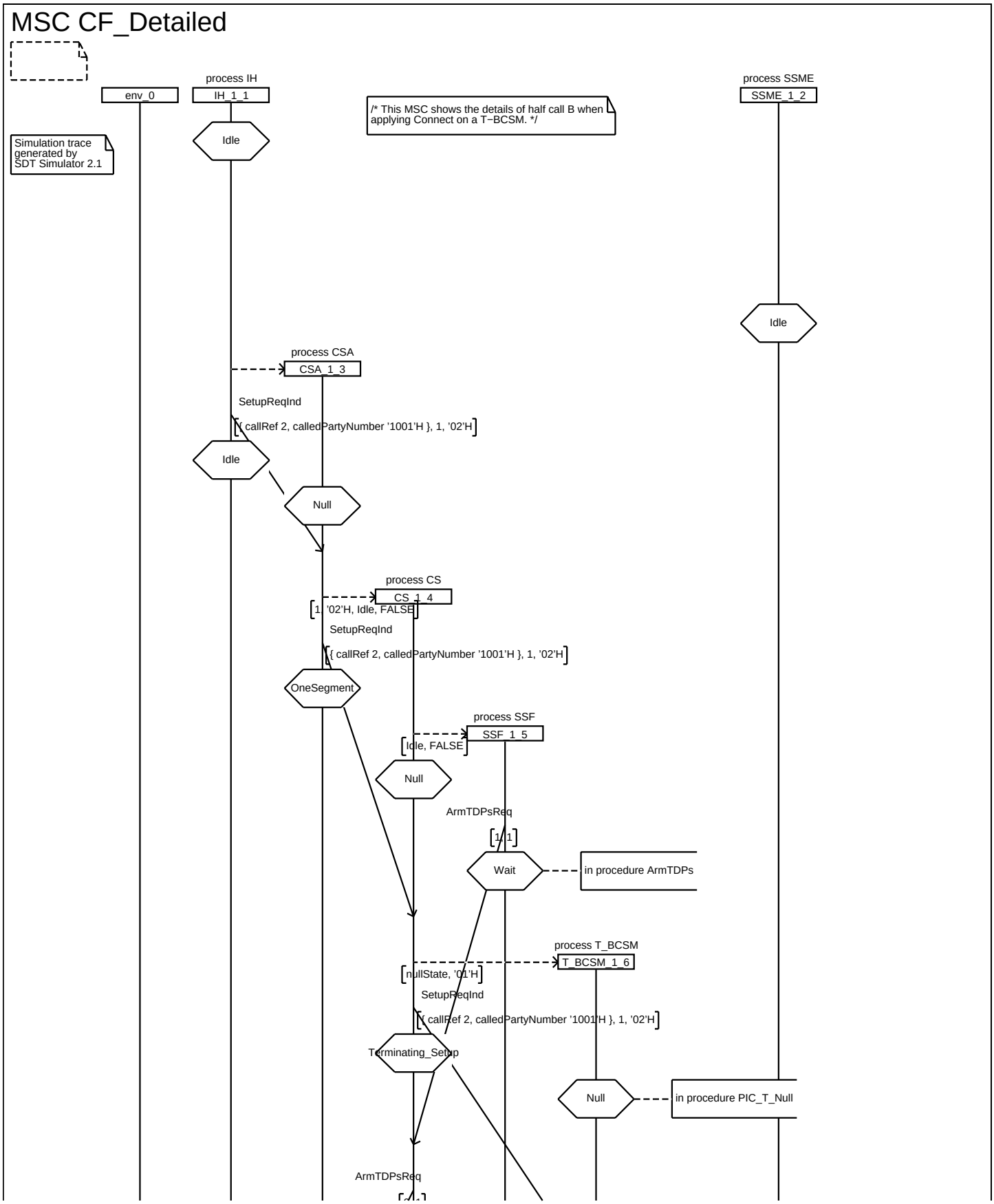
Annex A: CF_Overview



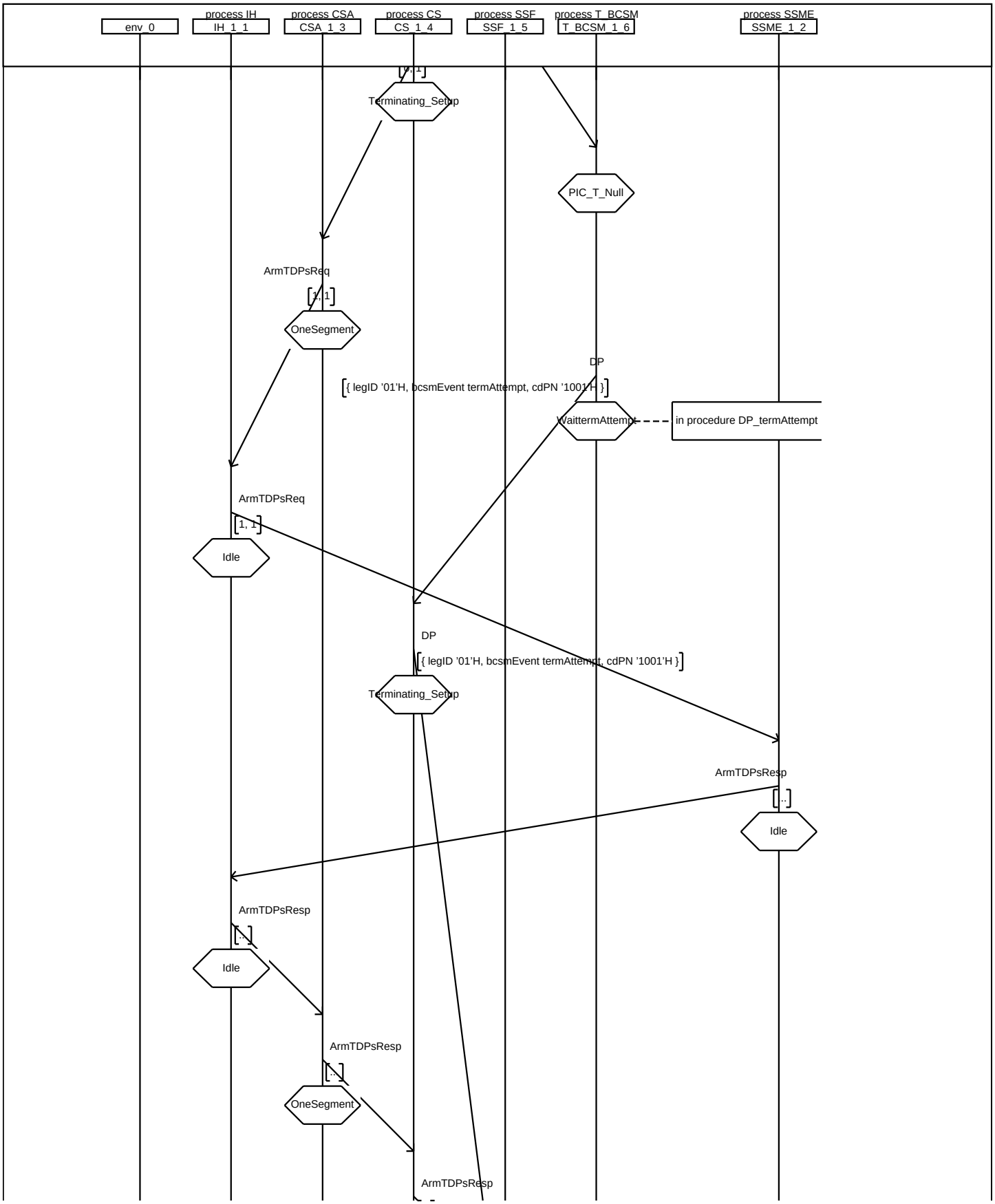
Annex A: CF_Overview



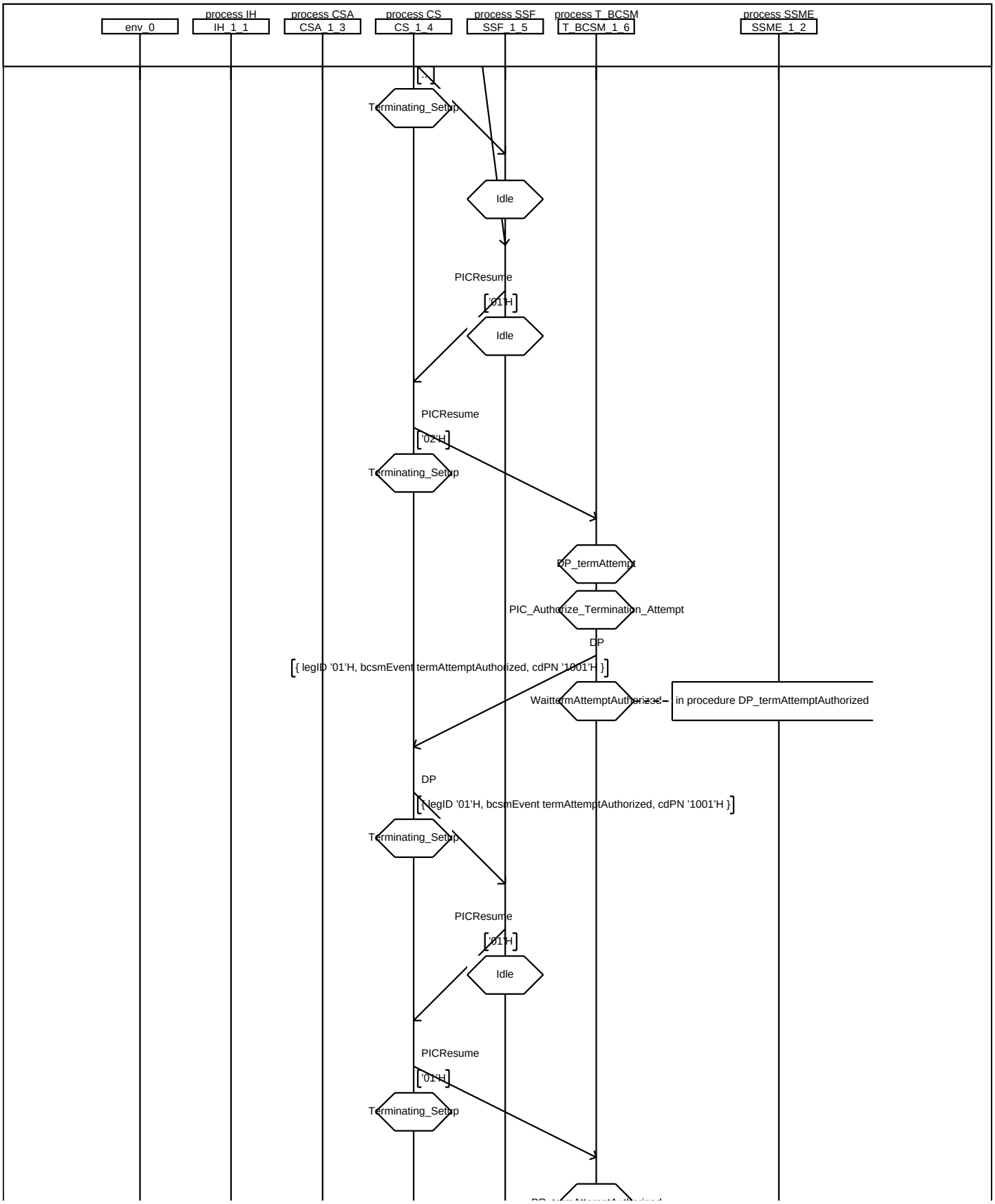
Annex A: CF_Detailed



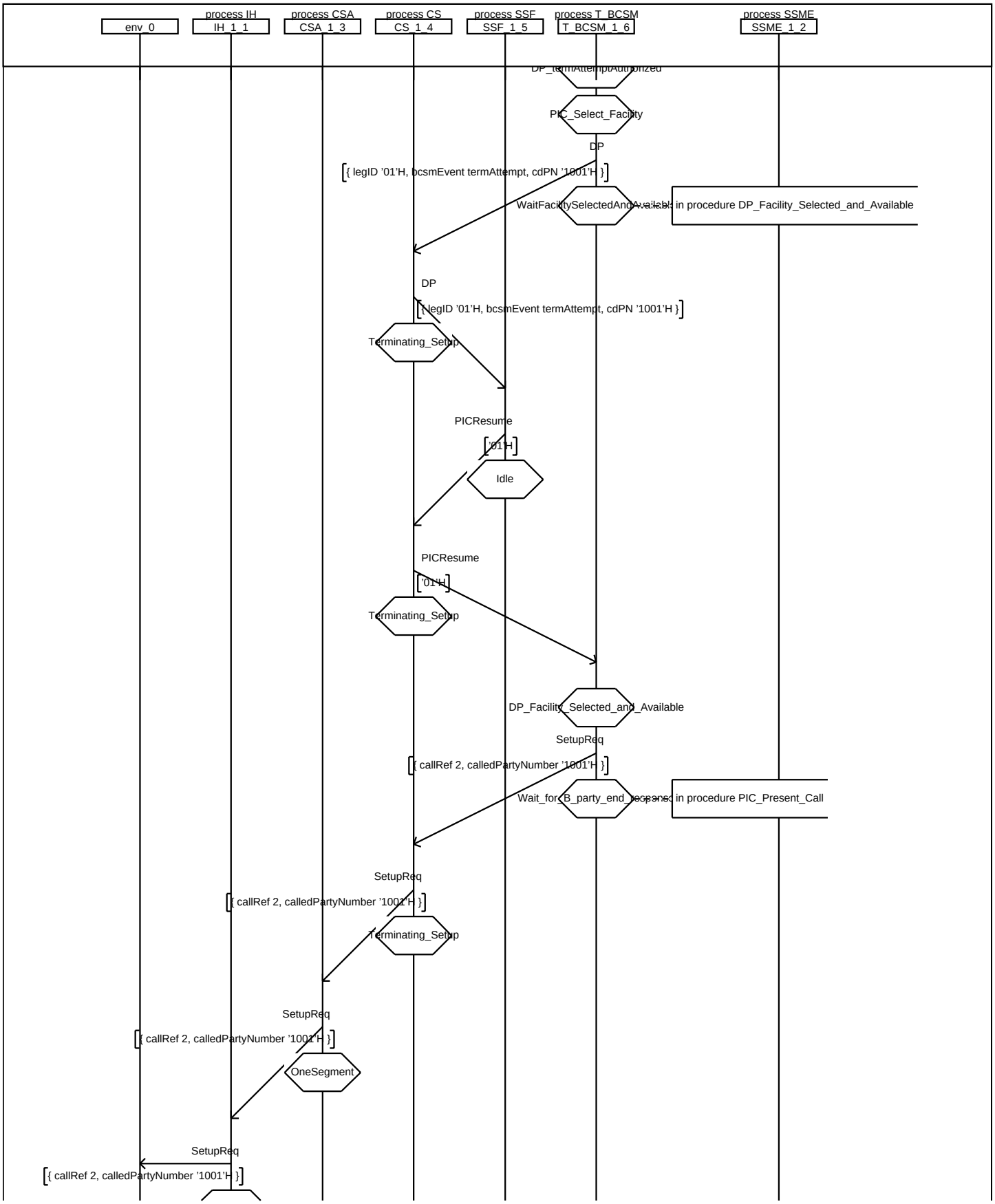
Annex A: CF_Detailed



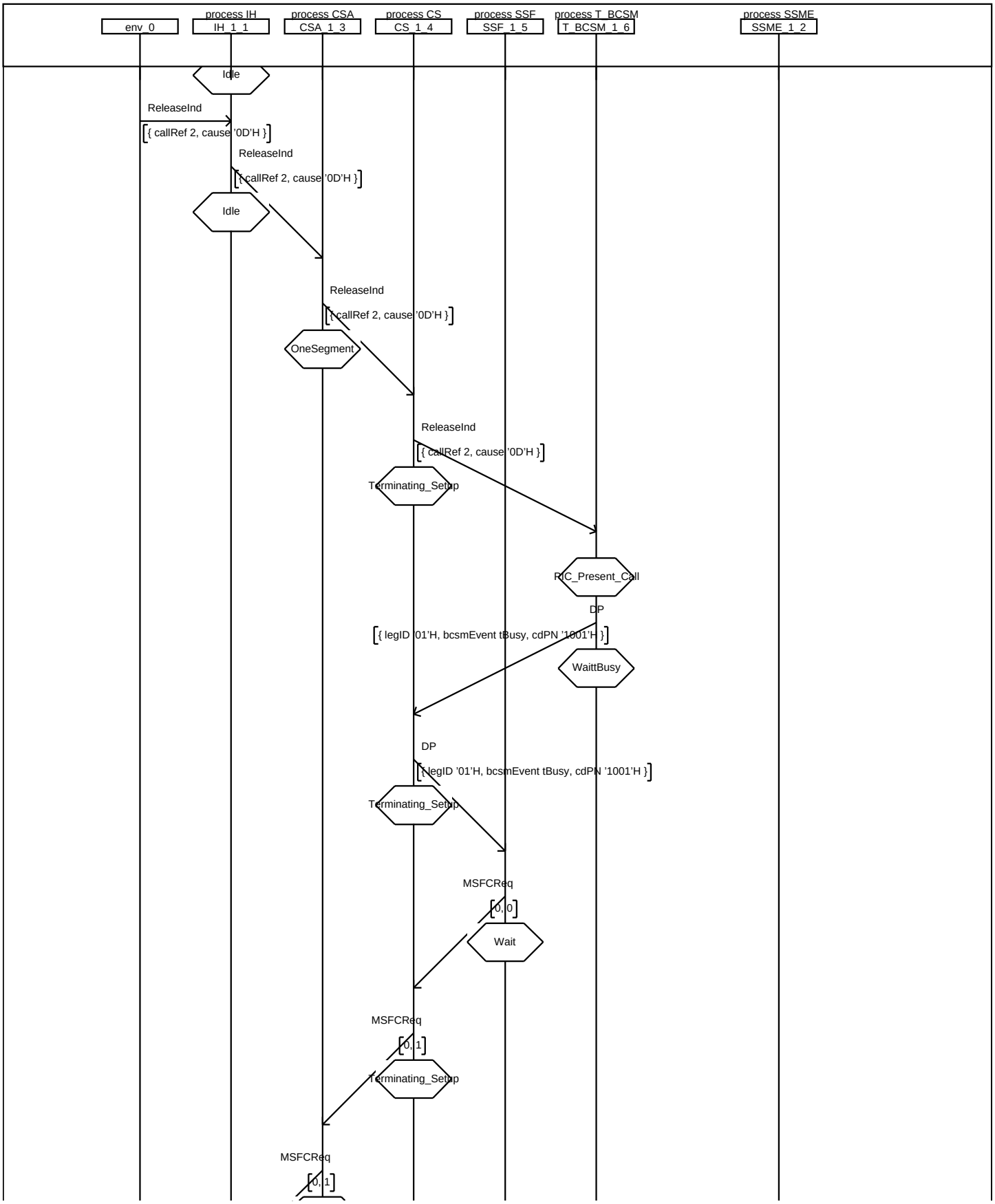
Annex A: CF_Detailed



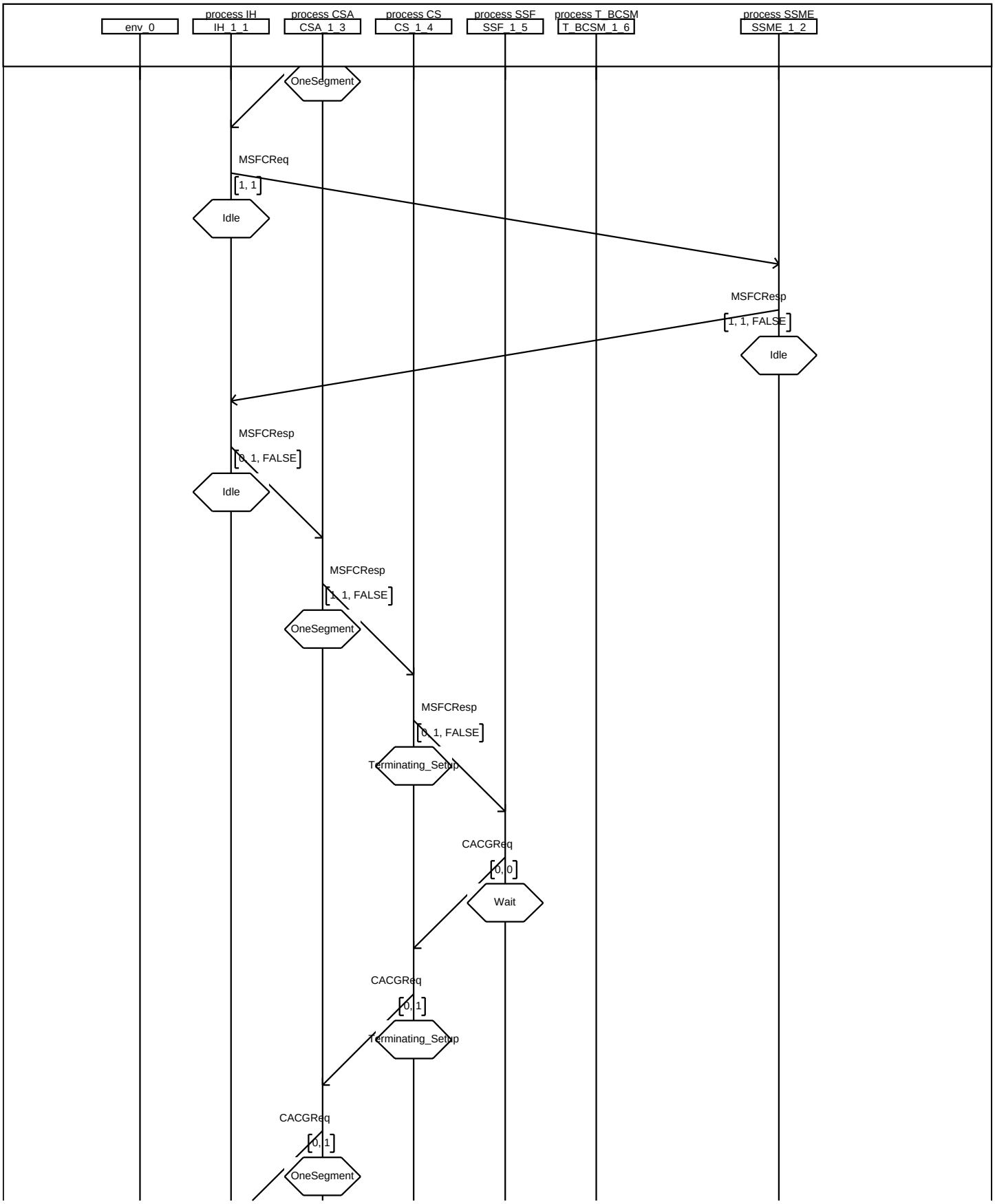
Annex A: CF_Detailed



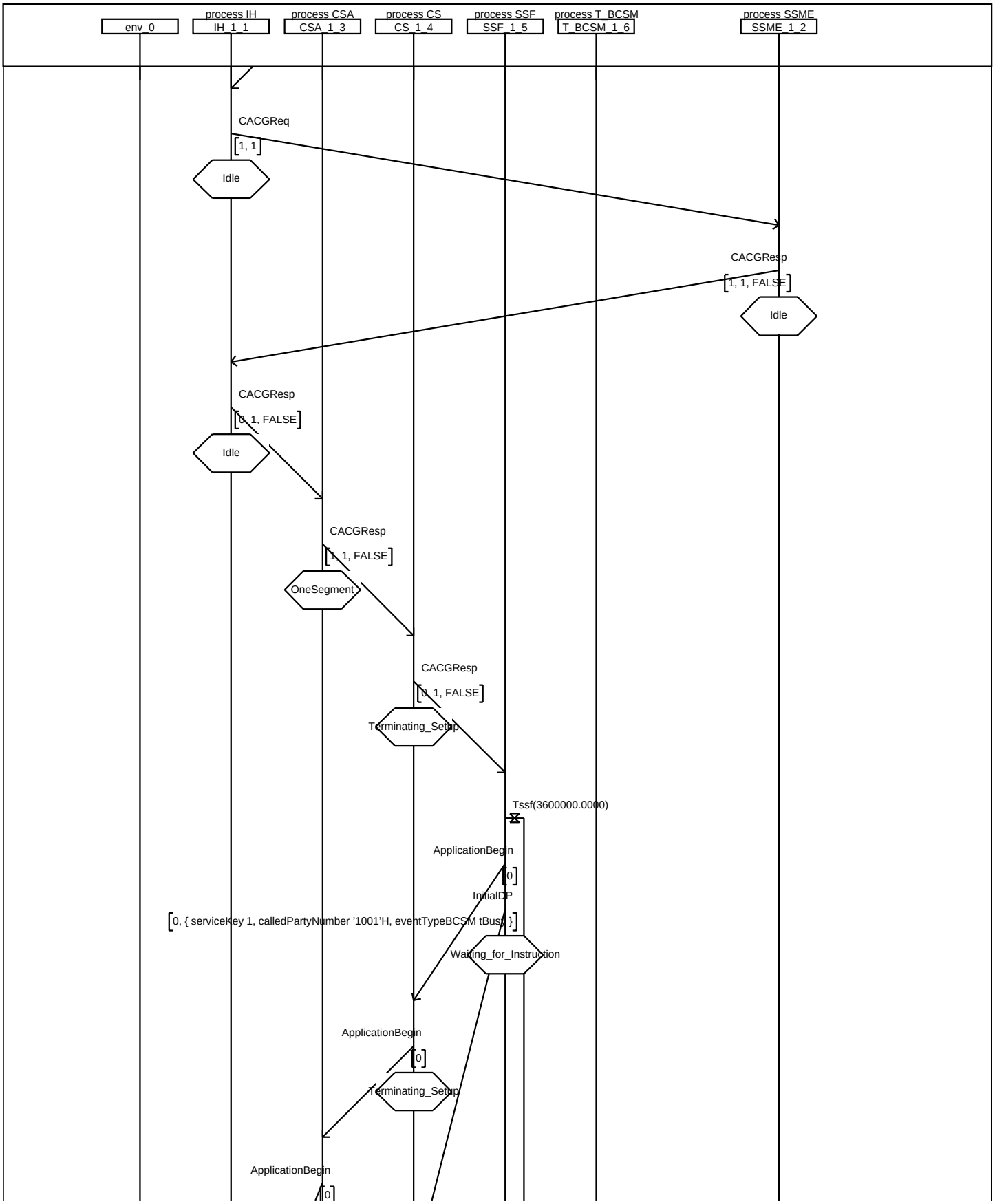
Annex A: CF_Detailed



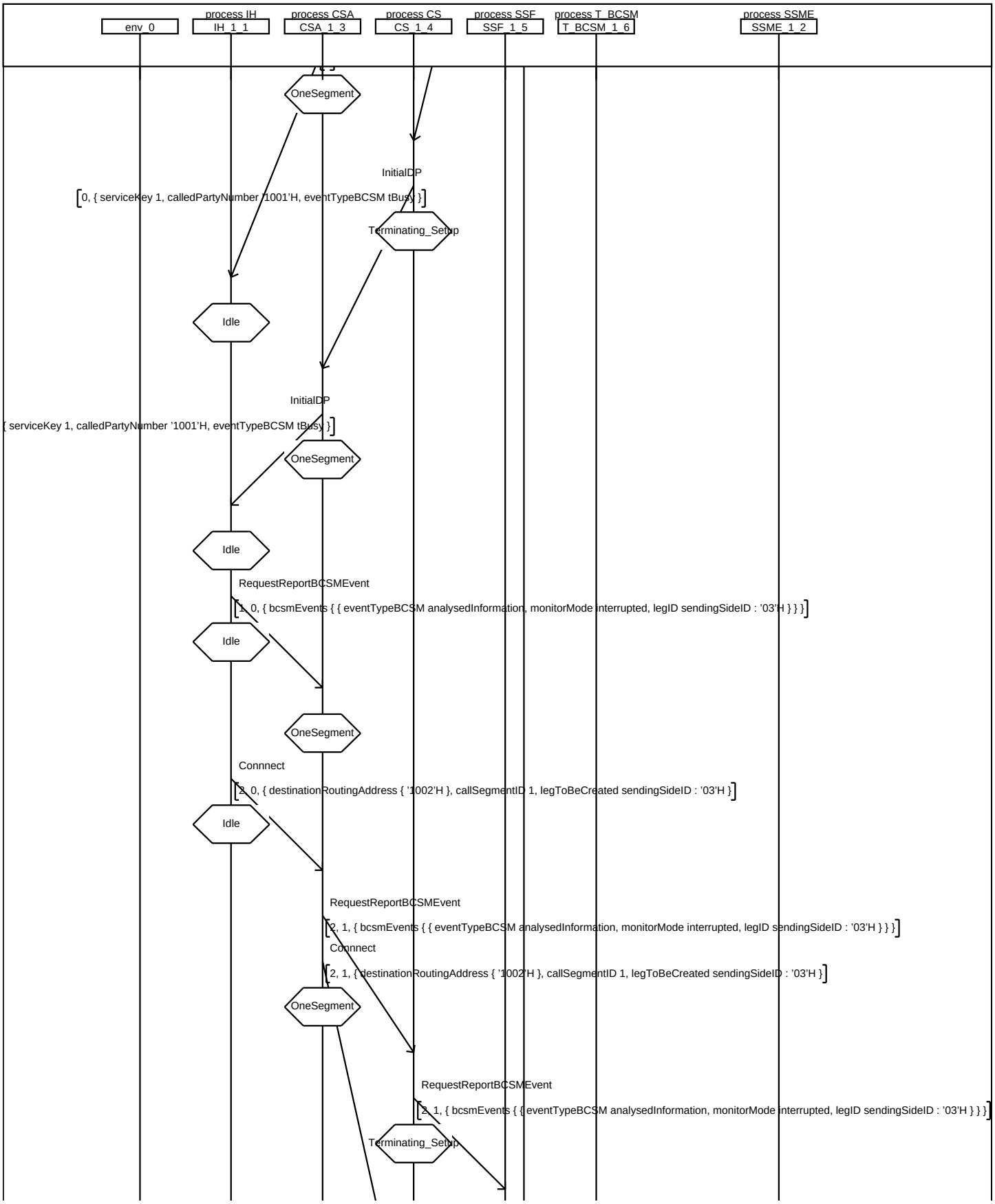
Annex A: CF_Detailed



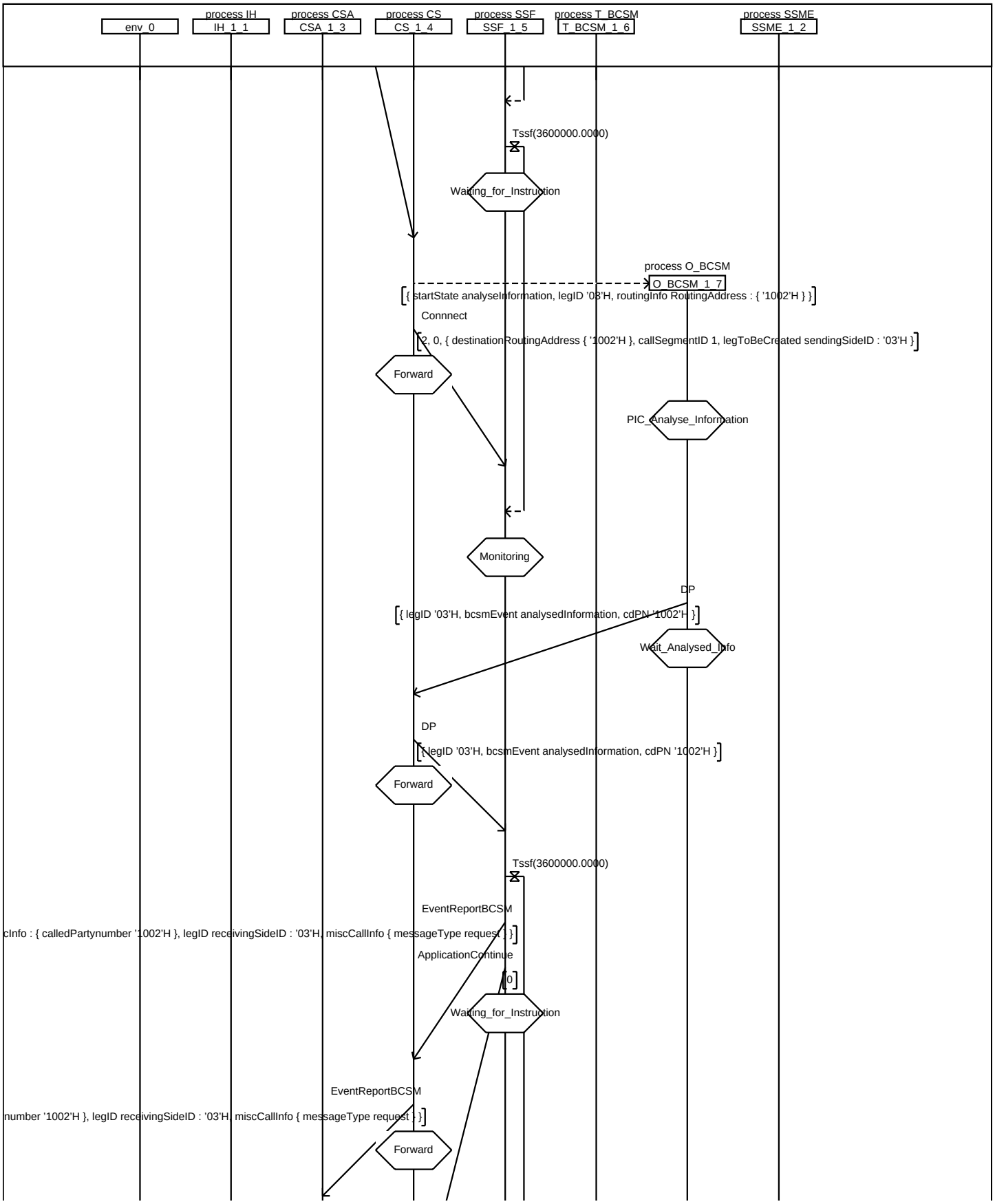
Annex A: CF_Detailed



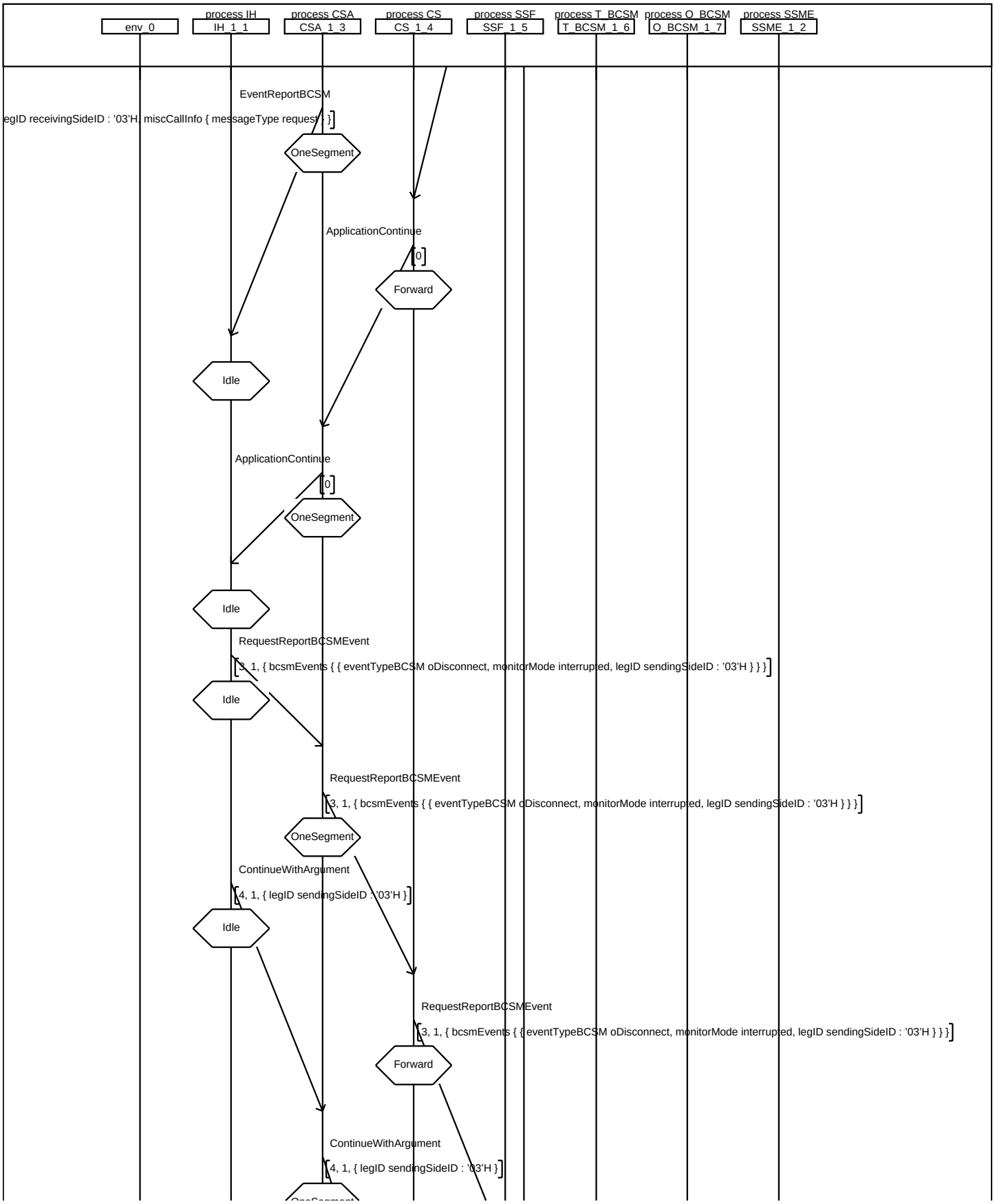
Annex A: CF_Detailed



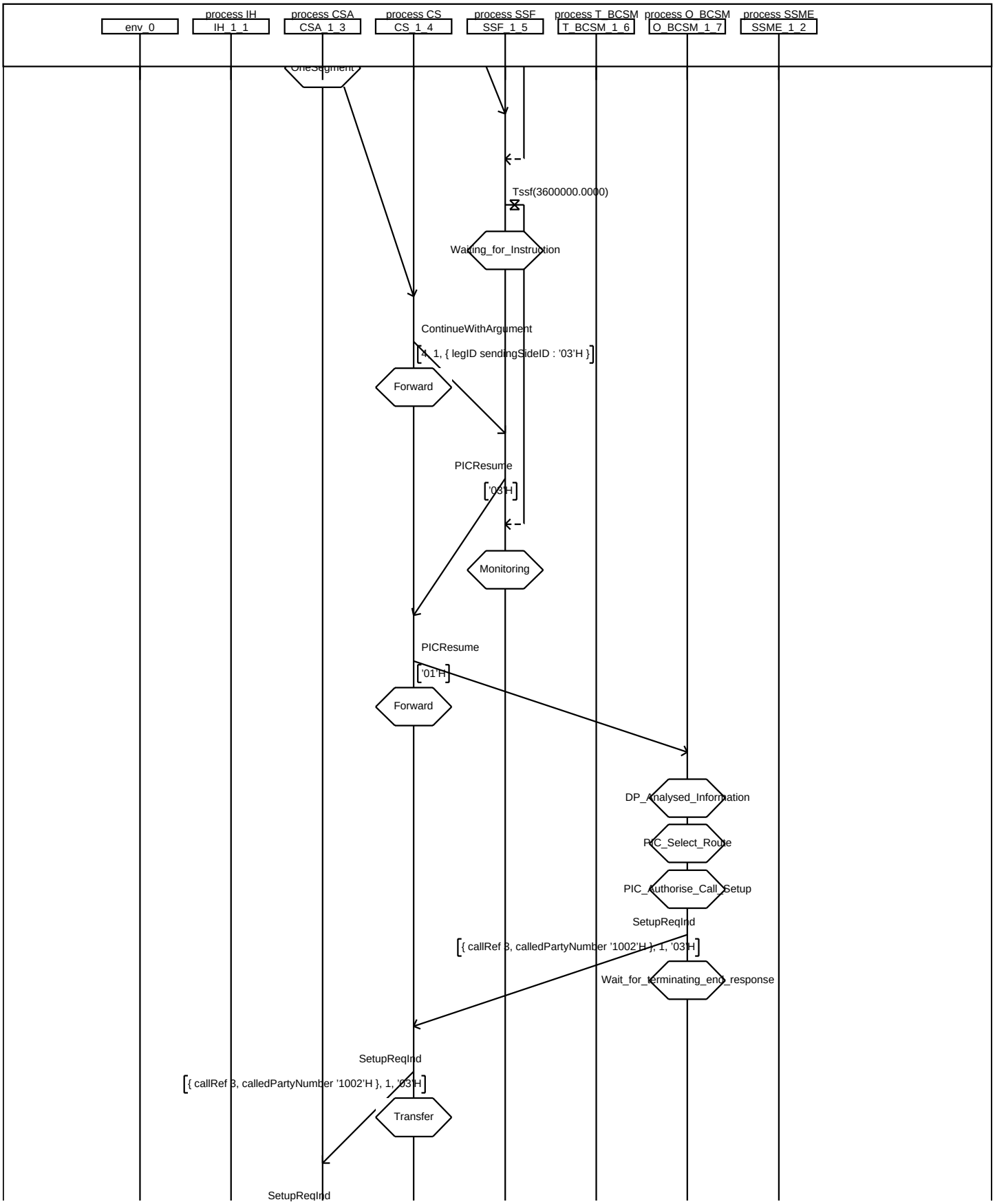
Annex A: CF_Detailed



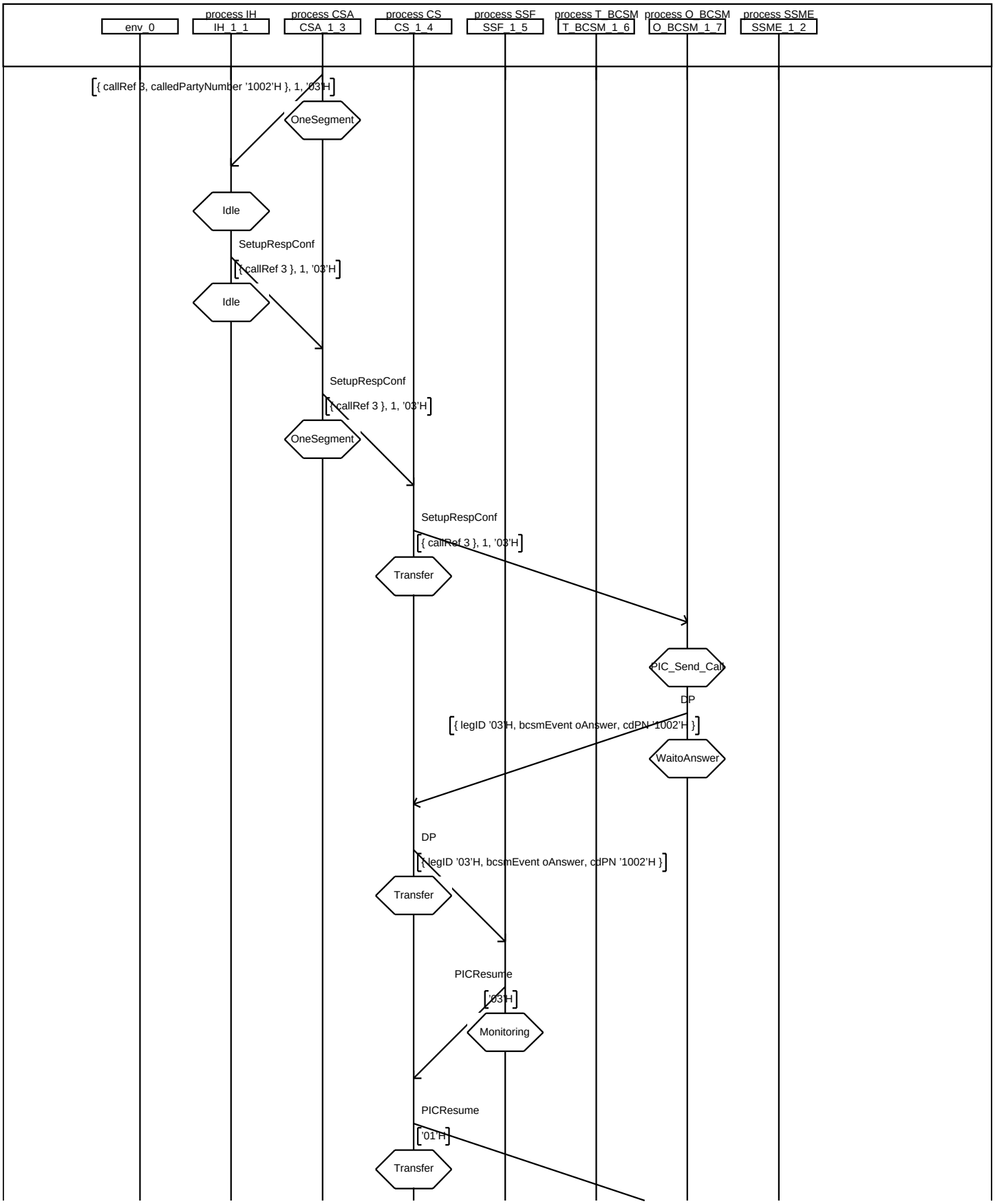
Annex A: CF_Detailed



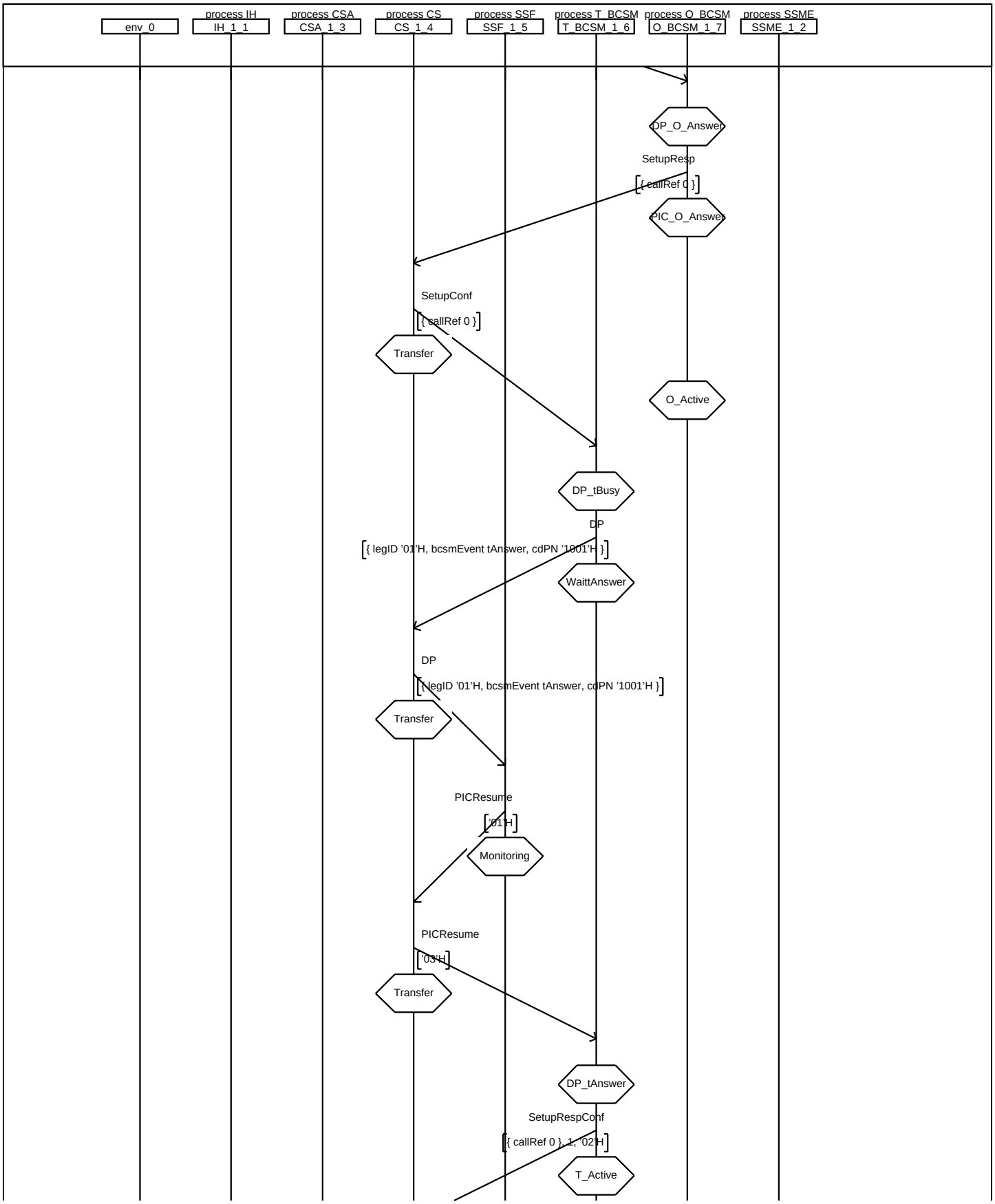
Annex A: CF_Detailed



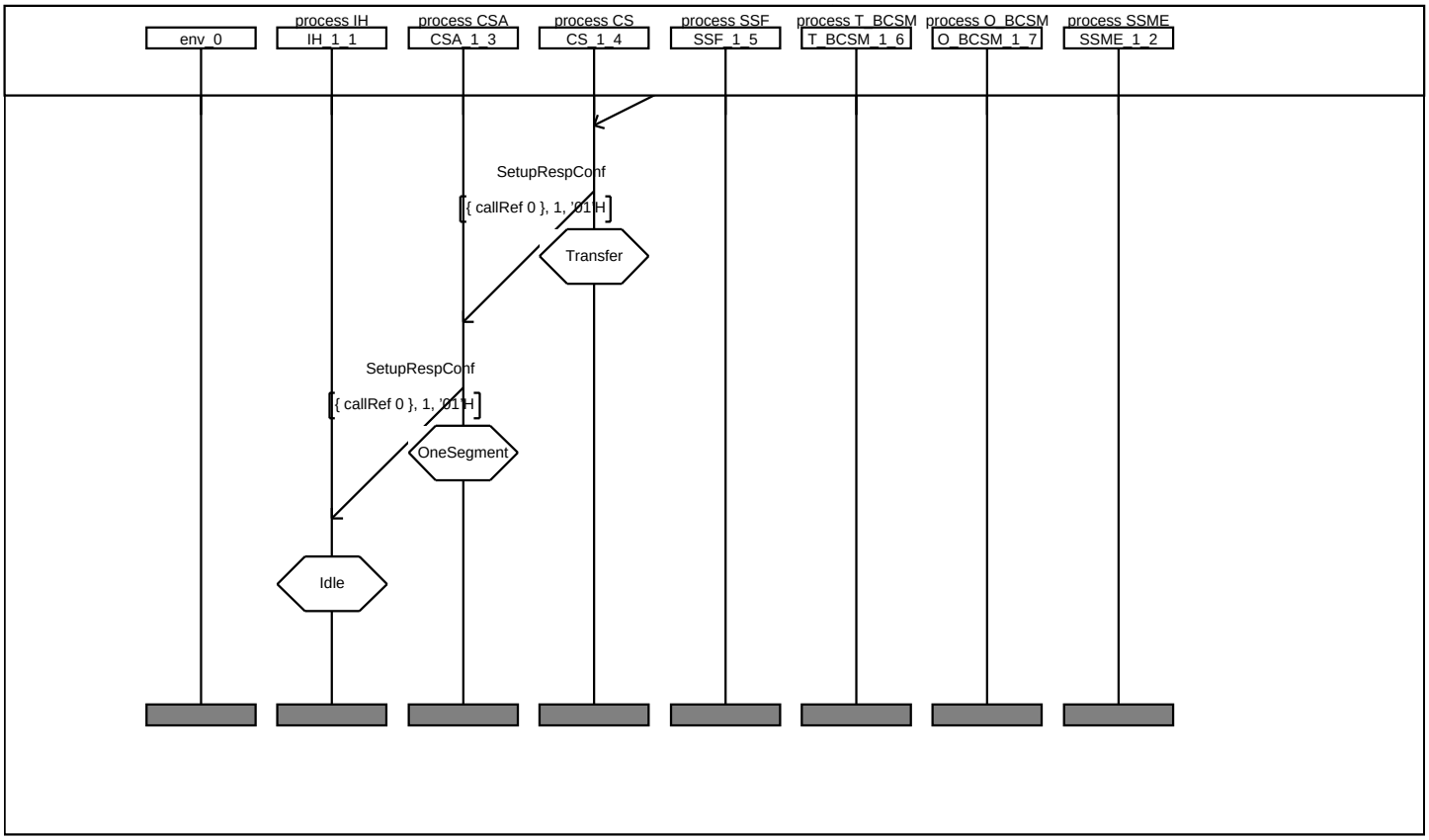
Annex A: CF_Detailed



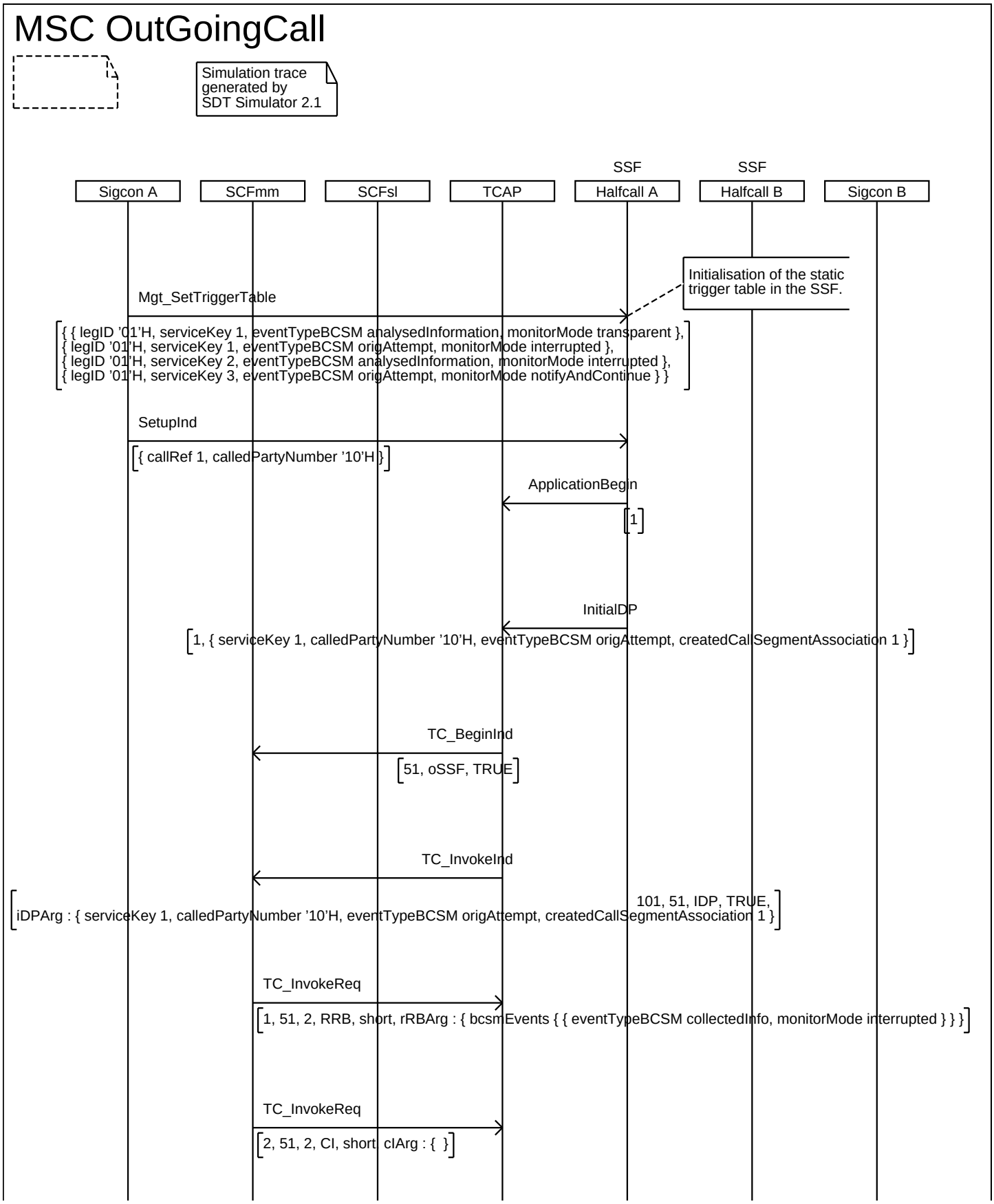
Annex A: CF_Detailed



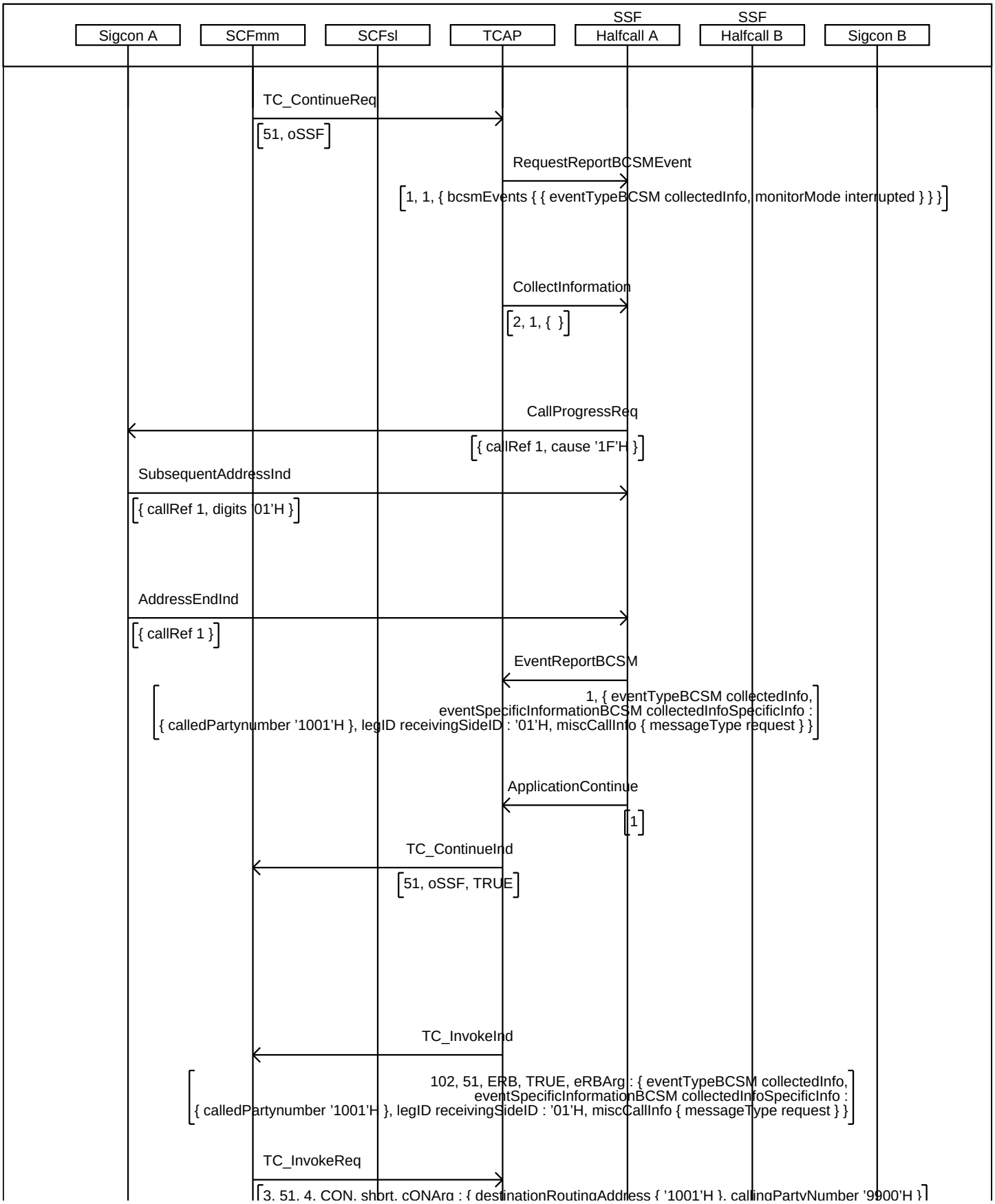
Annex A: CF_Detailed



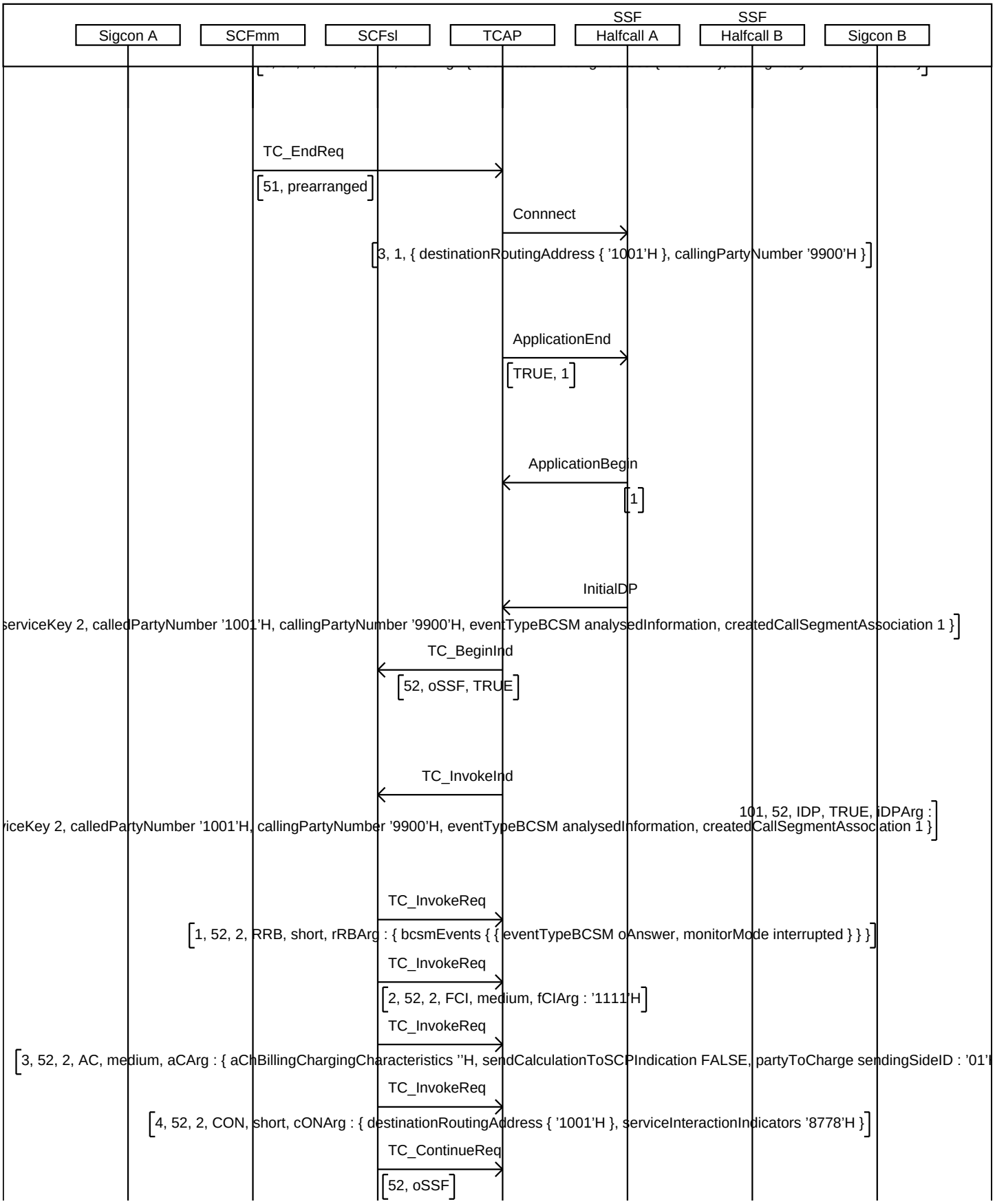
Annex A: OutGoingCall



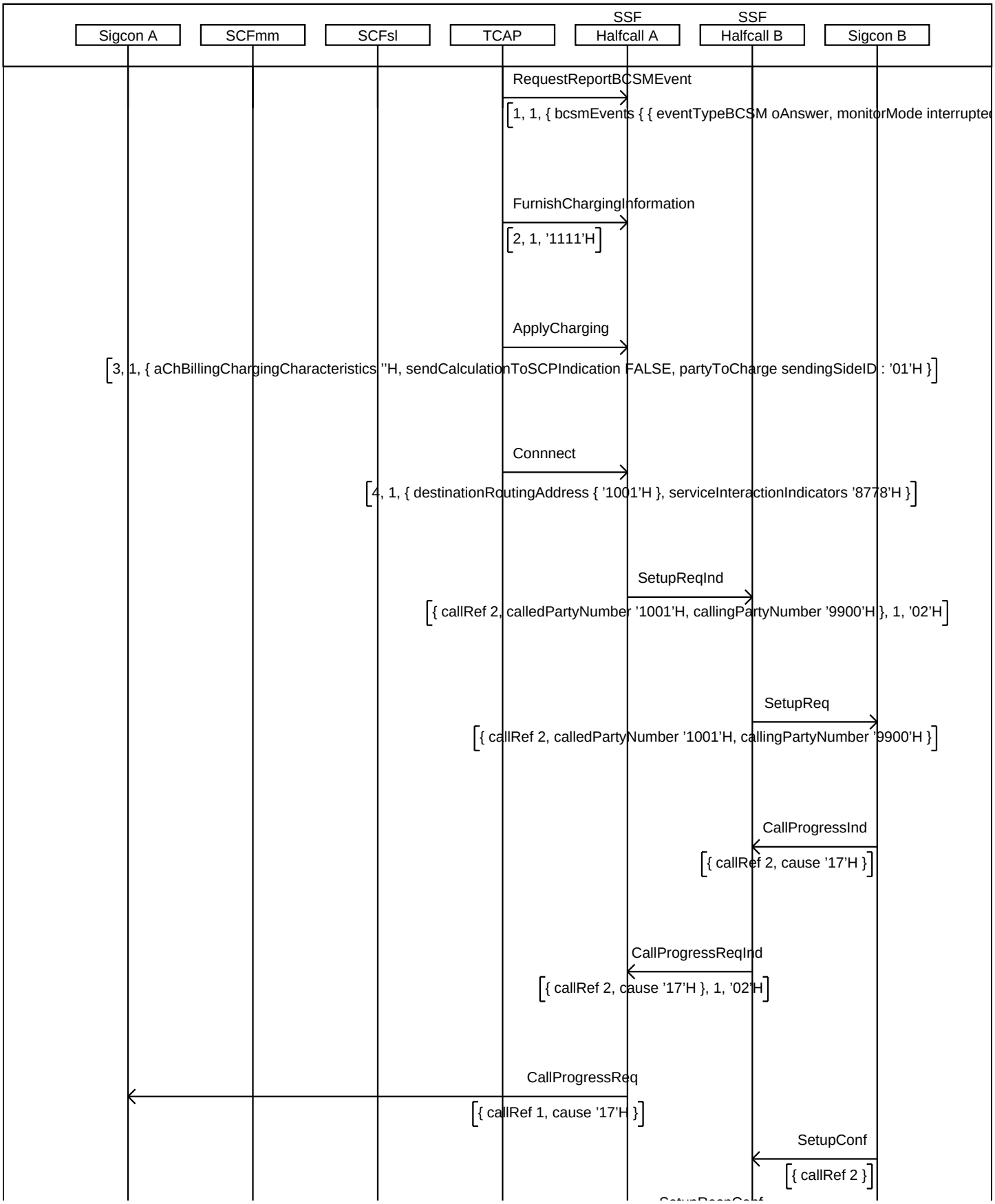
Annex A: OutGoingCall



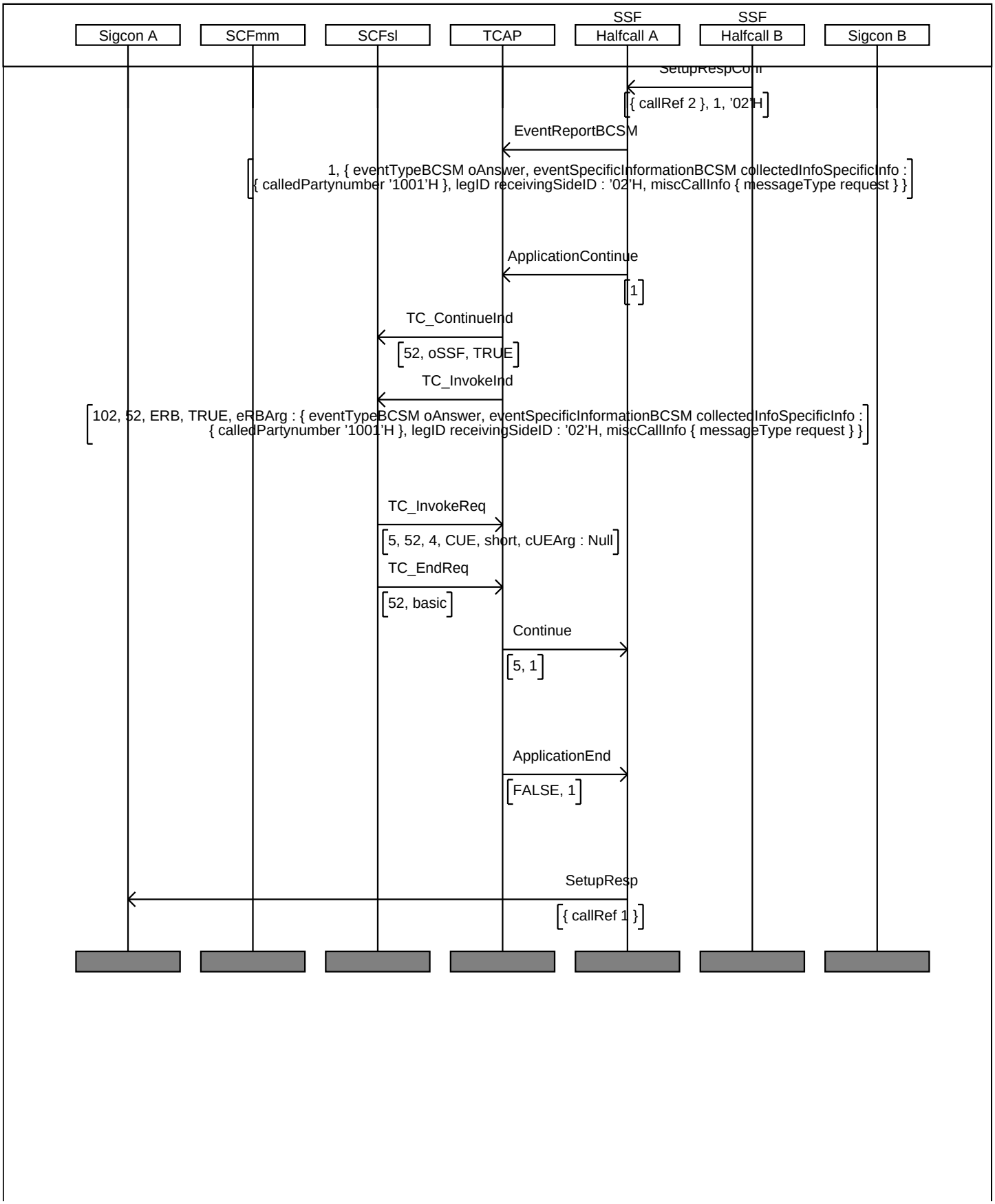
Annex A: OutGoingCall



Annex A: OutGoingCall



Annex A: OutGoingCall



Annex A: OutGoingCall