OrganizerView

SDT	System file *inap*	rw rw	/export/home/stf/stf154/ultan/CS3_SDL_work/inap.sdt inap.scu
	Source directory	rw	/export/home/stf/stf154/ultan/CS3_SDL_work/
	Target directory	rw	/export/home/stf/stf154/ultan/CS3_SDL_work/Target/
	Header	rw	/home/mulligan/INAP/CoreINAPCS32/INAP_Header
	Text readme	9	rw readme.txt
	Chapter ASN.1 Files *ASN1Files*	10	rw asn1/ASN1Files.scu
ASN	ASN.1 Text INCS3opsargs	10	rw asn1/ssf−scf.asn
ASN	ASN.1 Text INCS3BundleArg	30	rw asn1/incs3b.asn
ASN	ASN.1 Text INCS3datatypes	33	rw asn1/incs3−dt.asn
ASN	ASN.1 Text INCS3Internals	72	rw asn1/cs3i.asn
	Chapter IN CS−3 Specification	77	
	Package CS3_INAP *CS3_INAP*	77	rw sdl/cs3_inap/CS3_INAP.sun rw sdl/cs3_inap/CS3_INAP.scu
	Block Type SSF_CCF	85	rw sdl/cs3_inap/ssf1.sbt
	Process Type SSME_Control *SSME_Control*	98	rw sdl/ssme_c/ssme_control.spt rw sdl/ssme_c/SSME_Control.scu
	Procedure AllocateCSAId	119	rw sdl/ssme_c/alldid.spd
	Procedure AddCSAId	120	rw sdl/ssme_c/adddid.spd
	Procedure GetCSAIdfromCSA	121	rw sdl/ssme_c/getdid.spd
	Procedure GetCSAfromCSAId	122	rw sdl/ssme_c/getcvd.spd
	Procedure IsCSA	123	rw sdl/ssme_c/iscsa.spd
	Procedure ProcessMoveCallSegments	124	rw sdl/ssme_c/processm.spd
	Procedure GetMasterTDPTable	125	rw sdl/ssme_c/GetMasterTDPTable.spd
	Procedure ProcessActivateServiceFiltering	126	rw sdl/ssme_c/procasf.spd
	Procedure InitialiseTDPTable	127	rw sdl/ssme_c/inittdps.spd
	Procedure ProcessCallGap	128	rw sdl/ssme_c/proccg.spd
	Procedure AddTDPEntry	129	rw sdl/ssme_c/AddTDPEntry.spd
	Procedure Mgt_SetTriggerTable	130	rw sdl/ssme_c/mgt_stt.spd
	Procedure CheckMatchingServiceFilteringCriteria	131	rw sdl/ssme_c/matching.spd
	Procedure ProcessManageTriggerData	132	rw sdl/ssme_c/procmtd.spd
	Procedure ManageOneTrigger	134	rw sdl/ssme_c/ManOneTrigger.spd
	Procedure CheckACG	135	rw sdl/ssme_c/checkacg.spd
	Procedure ProcessCreateOrRemoveTriggerData	136	rw sdl/ssme_c/CreateOrRemove.spd
	Procedure CheckCallFiltered	138	rw sdl/ssme_c/callfilt.spd
	Procedure CountCall	139	rw sdl/ssme_c/CountCall.spd
	Procedure FindTDP	140	rw sdl/ssme_c/FindTDP.spd
	Procedure SendSFR	141	rw sdl/ssme_c/SendSFR.spd
	Procedure RemoveTDP	142	rw sdl/ssme_c/RemoveTDP.spd
	Procedure SFGetCause	143	rw sdl/ssme_c/SFGetCause.spd
	Procedure CGGetCause	144	rw sdl/ssme_c/CGGetCause.spd

OrganizerView

Procedure SFInfoToSend	145	rw	sdl/ssme_c/SFInfoToSend.spd
Procedure CGInfoToSend	146	rw	sdl/ssme_c/CGInfoToSend.spd
Process Type CallSegmentAssociation	147	rw	sdl/csa/csa.spt
CallSegmentAssociation		rw	sdl/csa/CallSegmentAssociation.scu
Procedure AddCS	176	rw	sdl/csa/addcs.spd
Procedure SetLegLocation	177	rw	sdl/csa/setlegl.spd
Procedure GetCSPtr	178	rw	sdl/csa/getcsptr.spd
Procedure GetLegLocation	179	rw	sdl/csa/getlegl.spd
Procedure GetSSFPtr	180	rw	sdl/csa/getssfptr.spd
Procedure SetLegAssoc	181	rw	sdl/csa/setlassc.spd
Procedure ExistCS	182	rw	sdl/csa/existcs.spd
Procedure GetLegIdfromRemoteLegId	183	rw	sdl/csa/getlegrc.spd
Procedure IsCS	184	rw	sdl/csa/iscs.spd
Procedure ExistLeg	185	rw	sdl/csa/existleg.spd
Procedure NoOfSegments	186	rw	sdl/csa/noofsegm.spd
Procedure ExportLeg	187	rw	sdl/csa/exportl1.spd
Procedure ExportCS	188	rw	sdl/csa/exportcs.spd
Procedure ImportLeg	189	rw	sdl/csa/importl1.spd
Procedure ImportCS	190	rw	sdl/csa/importcs.spd
Procedure RemoveCS	191	rw	sdl/csa/removecs.spd
Procedure MoveLegs	192	rw	sdl/csa/movelegs.spd
Procedure ExportControllingLeg	193	rw	sdl/csa/expcl.spd
Procedure SendQueuedOperations	194	rw	sdl/csa/SendQueuedOperations.spd
Procedure QueueOperation	195	rw	sdl/csa/QueueOperation.spd
Procedure ProcessRRBE	196	rw	sdl/csa/processr.spd
Procedure ProcessFCI	197	rw	sdl/csa/ProcessFCI.spd
Procedure ReleaseCall	198	rw	sdl/csa/release1.spd
Procedure ProcessAC	199	rw	sdl/csa/ProcessAC.spd
Procedure ProcessCIRQ	200	rw	sdl/csa/ProcessCIRQ.spd
Procedure ReleaseAllSegments	201	rw	sdl/csa/ReleaseAllSegments.spd
Procedure ProcessRNCE	202	rw	sdl/csa/ProcessRNCE.spd
Procedure ProcessCancel	203	rw	sdl/csa/ProcessCancel.spd
Procedure BroadcastToCSs	204	rw	sdl/csa/BroadcastToCSs.spd
Procedure RemoveLegCSA	206	rw	sdl/csa/removele.spd
Process Type CallSegment	207	rw	sdl/cs/cv1.spt
CallSegment		rw	sdl/cs/CallSegment.scu
Procedure AddLeg	271	rw	sdl/cs/addleg.spd
Procedure RemoveLeg	272	rw	sdl/cs/remleg.spd
Procedure SetLegStatus	273	rw	sdl/cs/setlstat.spd
Procedure GetLegStatus	274	rw	sdl/cs/getlstat.spd
Procedure SetLegAssoc	275	rw	sdl/cs/setlass.spd
Procedure SetLegPtr	276	rw	sdl/cs/setlptr.spd
Procedure GetLegIdfromRemoteLegId	277	rw	sdl/cs/getlegr.spd
Procedure CheckNoPendingLegs	278	rw	sdl/cs/CheckNoPendingLegs.spd
Procedure NoOfJoinedLegs	279	rw	sdl/cs/NoOfJoinedLegs.spd
Procedure AddBCSM	280	rw	sdl/cs/AddBCSM.spd
Procedure DeletePendingLeg	281	rw	sdl/cs/DeletePendingLeg.spd
Procedure GetPassiveLegId	282	rw	sdl/cs/getpl.spd
Procedure GetPendingLegID	283	rw	sdl/cs/GetPassiveLegId.spd
Procedure MapConnectToBCSM	284	rw	sdl/cs/mpcobcsm.spd

Procedure MapSIToBCSM	285	rw	sdl/cs/mpsibcsm.spd
Procedure SetNoAnswerTime	286	rw	sdl/cs/SetNoAnswerTime.spd
Procedure ImportLeg	287	rw	sdl/cs/importle.spd
Procedure BroadcastToLegs	288	rw	sdl/cs/broadcas.spd
Procedure ExistLeg	290	rw	sdl/cs/ExistLeg.spd
Procedure ExportLeg	291	rw	sdl/cs/exportle.spd
Procedure NoOfLegs	292	rw	sdl/cs/nooflegs.spd
Procedure GetFAPtr	293	rw	sdl/cs/GetFAPtr.spd
Procedure ValidSourceCSforML	294	rw	sdl/cs/ValidSourceCSforML.spd
Procedure ValidSourceCSforMCS	295	rw	sdl/cs/ValidCSforMCS.spd
Procedure ValidSourceCSforSL	296	rw	sdl/cs/ValidSourceCSforSL.spd
Procedure ValidTargetCSforMLMCS	297	rw	sdl/cs/ValidTargetCSforML.spd
Procedure DPFiltering	298	rw	sdl/cs/dpfil.spd
Procedure MapCIToBCSM	302	rw	sdl/cs/mcitbcsm.spd
Procedure DPFilteringUTSI	303	rw	sdl/cs/dpfilter.spd
Process Type SSF_FSM	304	rw	sdl/ssf/ssf_fsm.spt
SSF_FSM		rw	sdl/ssf/SSF_FSM.scu
Procedure DPArmed	354	rw	sdl/ssf/dparmed.spd
Procedure ExistLeg	355	rw	sdl/ssf/exleg.spd
Procedure InitializeImplicitDisarmingTable	356	rw	sdl/ssf/InitializeImplicitDisarmingTable.spd
Procedure RemoveDP	358	rw	sdl/ssf/removedp.spd
Procedure PerformImplicitDisarming	359	rw	sdl/ssf/PerformImplicitDisarming.spd
Procedure CallInformationReportPending	360	rw	sdl/ssf/callinfo.spd
Procedure IncrementOutstandingRequestsLegID	361	rw	sdl/ssf/ior.spd
Procedure ApplyChargingReportPending	362	rw	sdl/ssf/applycha.spd
Procedure OustandingRequests	363	rw	sdl/ssf/or.spd
Procedure ProcessApplyCharging	364	rw	sdl/ssf/process6.spd
Procedure ProcessContinue	365	rw	sdl/ssf/proces12.spd
Procedure ProcessRequestReportBCSMEvent	366	rw	sdl/ssf/prreqrep.spd
Procedure ProcessCallInformationRequest	367	rw	sdl/ssf/process7.spd
Procedure ProcessDisconnectForwardConnection	368	rw	sdl/ssf/processd.spd
Procedure ProcessSendChargingInformation	369	rw	sdl/ssf/procsci.spd
Procedure ProcessCancel	370	rw	sdl/ssf/process8.spd
Procedure ProcessEstablishTemporaryConnection	371	rw	sdl/ssf/processe.spd
Procedure ProcessEventReportBCSM	372	rw	sdl/ssf/prevrep.spd
Procedure ProcessCollectInformation	373	rw	sdl/ssf/process9.spd
Procedure ProcessFurnishChargingInformation	374	rw	sdl/ssf/proces13.spd
Procedure ReleaseAllLegs	375	rw	sdl/ssf/releasea.spd
Procedure ProcessConnect	376	rw	sdl/ssf/proces10.spd
Procedure ProcessInitiateCallAttempt	378	rw	sdl/ssf/processi.spd
Procedure ProcessInitialDP	379	rw	sdl/ssf/pridp.spd
Procedure ProcessConnectToResource	381	rw	sdl/ssf/proces11.spd
Procedure ProcessRequestNotificationChargingEvent	382	rw	sdl/ssf/proces16.spd
Procedure ProcessSelectFacility	383	rw	sdl/ssf/procsf.spd
Procedure ExportEventRecord	384	rw	sdl/ssf/expevl.spd
Procedure ProcessContinueWithArgument	385	rw	sdl/ssf/pcwa.spd
Procedure ProcessRequestReportUTSI	387	rw	sdl/ssf/process3.spd
Procedure ImportEventRecord	388	rw	sdl/ssf/impevl.spd
Procedure ProcessDisconnectLeg	389	rw	sdl/ssf/processdl.spd

Procedure ProcessSendSTUI	390	rw	sdl/ssf/process4.spd
Procedure ProcessReportUTSI	391	rw	sdl/ssf/process5.spd
Procedure IsUTSIArmed	392	rw	sdl/ssf/isutsiar.spd
Procedure IncrementOutstandingRequestsCSId	393	rw	sdl/ssf/ir.spd
Procedure DecrementRequests	394	rw	sdl/ssf/dr.spd
Procedure ResumePassiveLegs	395	rw	sdl/ssf/ResumePassiveLegs.spd
Procedure TreatCall	396	rw	sdl/ssf/TreatCall.spd
Process Type FIM_Controller	397	rw	sdl/fim/fim.spt
FIM_Controller		rw	sdl/fim/FIM_Controller.scu
Procedure CreateFIMAgent	399	rw	sdl/fim/CreateFIMAgent.spd
Procedure FIMAgentStopped	400	rw	sdl/fim/FAStopped.spd
Procedure UpdateFAInfo	401	rw	sdl/fim/UpdateFAInfo.spd
Procedure WhichFIMAgent	402	rw	sdl/fim/WhichFIMAgent.spd
Process Type FIMAgent	403	rw	sdl/fa/fimAgent.spt
FIMAgent		rw	sdl/fa/FIMAgent.scu
Procedure Initialize	416	rw	sdl/fa/Initialize.spd
Procedure CreateBCSMChain	417	rw	sdl/fa/CreateBCSMChain.spd
Procedure DoTDPProcessing	418	rw	sdl/fa/DoTDPProcessing.spd
Procedure WhichLegID	420	rw	sdl/fa/fasettdpleg.spd
Procedure SplitChain	421	rw	sdl/fa/SplitChain.spd
Procedure AddNewCS	422	rw	sdl/fa/AddNewCS.spd
Procedure UpdateCSInfo	423	rw	sdl/fa/UpdateCSInformation.spd
Procedure RemoveChain	424	rw	sdl/fa/RemoveChain.spd
Procedure CheckChainTermination	425	rw	sdl/fa/CheckChainTermination.spd
Procedure CreateCSA	426	rw	sdl/fa/faCreateCSA.spd
Procedure CheckMiddleRelease	427	rw	sdl/fa/CheckMiddleRelease.spd
Procedure GetArmedTDPs	428	rw	sdl/fa/GetTriggeredTDPs.spd
Procedure SelectService	429	rw	sdl/fa/SelectService.spd
Procedure BCSMInfo	430	rw	sdl/fa/BCSMInfo.spd
Procedure FindBCSM	431	rw	sdl/fa/FindBCSM.spd
Procedure SetOneServiceProfile	432	rw	sdl/fa/SetOneSP.spd
Procedure CSToCSA	433	rw	sdl/fa/CSToCSA.spd
Procedure SetNoAnswerTime	434	rw	sdl/fa/SetNoAnswerTime1.spd
Procedure MoveCallSegment	435	rw	sdl/fa/MoveCallSegment.spd
Procedure ExportBCSM	436	rw	sdl/fa/ExportBCSM.spd
Procedure ImportBCSM	437	rw	sdl/fa/ImportBCSM.spd
Procedure UpdateServiceInfo	438	rw	sdl/fa/UpdateServiceInfo.spd
Process Type BCSMControllerProcess	439	rw	sdl/bcp/bcp.spt
BCSMControllerProcess		rw	sdl/bcp/BCSMControllerProcess.scu
Procedure AddPassiveBCSM	471	rw	sdl/bcp/AddPassiveBCSM.spd
Procedure BCSMChainCreated	472	rw	sdl/bcp/BCSMChainCreated.spd
Procedure BCSMStartChain	473	rw	sdl/bcp/BCSMStartChain.spd
Procedure CreateBCSM	474	rw	sdl/bcp/CreateBCSM.spd
Procedure CreateControllingBCSM	475	rw	sdl/bcp/CreateControllingBCSM.spd
Procedure CutBCSMChain	476	rw	sdl/bcp/CutBCSMChain.spd
Procedure DestinationFromCallRef	477	rw	sdl/bcp/DestinationFromCallRef.spd
Procedure DestinationFromPId	478	rw	sdl/bcp/DestinationFromPId.spd
Procedure DestinationFromSigCon	479	rw	sdl/bcp/DestinationFromSigCon.spd
Procedure DestinationsFromPId	480	rw	sdl/bcp/DestinationsFromPId.spd
Procedure DestinationsFromSigCon	481	rw	sdl/bcp/DestinationsFromSigCon.spd

Procedure GetBCSMState	482	rw	sdl/bcp/GetBCSMState.spd
Procedure GetOtherPassiveLeg	483	rw	sdl/bcp/GetOtherPassiveLeg.spd
Procedure GetSigState	484	rw	sdl/bcp/GetSigState.spd
Procedure IsBCSM	485	rw	sdl/bcp/bcpIsBCSM.spd
Procedure MapCallRefToChain	486	rw	sdl/bcp/MapCallRefToChain.spd
Procedure MapSigConToChain	487	rw	sdl/bcp/MapSigConToChain.spd
Procedure MapSIToBCSM	488	rw	sdl/bcp/bcpmpsibcsm.spd
Procedure MoveControllingBCSMChain	489	rw	sdl/bcp/MoveControllingBCSMChain.spd
Procedure MovePassiveBCSMChain	490	rw	sdl/bcp/MovePassiveBCSMChain.spd
Procedure RemoveBCSM	491	rw	sdl/bcp/RemoveBCSM.spd
Procedure RemoveBCSMChain	492	rw	sdl/bcp/RemoveBCSMChain.spd
Procedure RemoveBCSMs	493	rw	sdl/bcp/RemoveBCSMs.spd
Procedure SetBCSMState	494	rw	sdl/bcp/bcpSetBCSMState.spd
Procedure SetCallProgressState	495	rw	sdl/bcp/SetCallProgressState.spd
Procedure SetPassiveToIBI	496	rw	sdl/bcp/SetPassiveToIBI.spd
Procedure SetSigState	497	rw	sdl/bcp/bcpSetSigState.spd
Procedure SetSurrogateChainStart	498	rw	sdl/bcp/SetSurrogateChainStart.spd
Procedure SetSurrogate	499	rw	sdl/bcp/SetSurrogate.spd
Process Type OriginatingBCSM	500	rw	sdl/o_bcsm/ocs1.spt
OriginatingBCSM		rw	sdl/o_bcsm/OriginatingBCSM.scu
Procedure PIC_O_Null	519	rw	sdl/o_bcsm/pic_o_nu.spd
Procedure PIC_Analyse_Information	521	rw	sdl/o_bcsm/pic_anal.spd
Procedure PIC_Send_Call	526	rw	sdl/o_bcsm/pic_send.spd
Procedure PIC_O_Active	532	rw	sdl/o_bcsm/pic_o_ac.spd
Procedure PIC_Authorise_Origination_Attempt	534	rw	sdl/o_bcsm/pic_auth.spd
Procedure PIC_Select_Route	535	rw	sdl/o_bcsm/pic_sele.spd
Procedure PIC_O_Alerting	536	rw	sdl/o_bcsm/pic_o_al.spd
Procedure PIC_O_Abandon	541	rw	sdl/o_bcsm/pic_o_ab.spd
Procedure PIC_Collect_Information	542	rw	sdl/o_bcsm/pic_coll.spd
Procedure PIC_Authorise_Call_Setup	545	rw	sdl/o_bcsm/pic_aut1.spd
Procedure PIC_O_Answer	546	rw	sdl/o_bcsm/pic_o_an.spd
Procedure PIC_OException	547	rw	sdl/o_bcsm/pic_oexc.spd
Procedure PIC_Collect_NDigits	548	rw	sdl/o_bcsm/pic_col1.spd
Procedure PIC_O_Suspended	551	rw	sdl/o_bcsm/pic_o_su.spd
Procedure PIC_O_Retention	554	rw	sdl/o_bcsm/pic_o_re.spd
Procedure PIC_Disconnect	556	rw	sdl/o_bcsm/pic_disc.spd
Procedure PIC_SCFRelease	557	rw	sdl/o_bcsm/pic_scfrel.spd
Procedure DP_origAttemptAuthorised	558	rw	sdl/o_bcsm/dp_origa.spd
Procedure DP_Route_Select_Failure	562	rw	sdl/o_bcsm/dp_route.spd
Procedure DP_oCalledPartyBusy	565	rw	sdl/o_bcsm/dp_ocall.spd
Procedure DP_oDisconnect	567	rw	sdl/o_bcsm/dpdiscon.spd
Procedure DP_Collected_Info	573	rw	sdl/o_bcsm/dp_colle.spd
Procedure DP_oAnswer	577	rw	sdl/o_bcsm/dp_o_ans.spd
Procedure DP_oMidCall	580	rw	sdl/o_bcsm/dp_omidc.spd
Procedure DP_Analysed_Information	584	rw	sdl/o_bcsm/dp_analy.spd
Procedure DP_oNoAnswer	587	rw	sdl/o_bcsm/dp_onoan.spd
Procedure DP_oAbandon	589	rw	sdl/o_bcsm/dp_oaban.spd
Procedure DP_origAttempt	590	rw	sdl/o_bcsm/dp_orig1.spd
Procedure DP_oSuspend	592	rw	sdl/o_bcsm/dp_osusp.spd

Procedure DP_oReanswer	594	rw	sdl/o_bcsm/dp_orean.spd
Procedure DP_oTermSeized	596	rw	sdl/o_bcsm/dp_oterm.spd
Procedure DP_Authorize_Route_Failure	598	rw	sdl/o_bcsm/dp_arf.spd
Procedure DP_origDenied	599	rw	sdl/o_bcsm/dp_odenied.spd
Procedure MapToSIRArg	600	rw	sdl/o_bcsm/maptosir.spd
Procedure MapToDP	601	rw	sdl/o_bcsm/maptodp.spd
Procedure MapFromPIC	602	rw	sdl/o_bcsm/mapfpic.spd
Procedure AnalyzeDigits	603	rw	sdl/o_bcsm/AnalyzeDigits.spd
Procedure SetNoAnswerTime	604	rw	sdl/o_bcsm/SetNoAnswerTime.spd
Procedure SetFIMAgent	605	rw	sdl/o_bcsm/SetFIMAgent1.spd
Process Type TerminatingBCSM	606	rw	sdl/t_bcsm/tcs1.spt
TerminatingBCSM		rw	sdl/t_bcsm/TerminatingBCSM.scu
Procedure PIC_T_Null	619	rw	sdl/t_bcsm/pic_t_nu.spd
Procedure PIC_Select_Facility	621	rw	sdl/t_bcsm/pic_sel1.spd
Procedure PIC_Disconnect	622	rw	sdl/t_bcsm/pic_dis2.spd
Procedure PIC_T_Suspended	623	rw	sdl/t_bcsm/picts.spd
Procedure PIC_Authorize_Termination_Attempt	626	rw	sdl/t_bcsm/pic_aut2.spd
Procedure PIC_Present_Call	627	rw	sdl/t_bcsm/pic_pres.spd
Procedure PIC_tActive	630	rw	sdl/t_bcsm/pic_tact.spd
Procedure PIC_TException	633	rw	sdl/t_bcsm/pic_texc.spd
Procedure PIC_tAlerting	634	rw	sdl/t_bcsm/pic_tale.spd
Procedure DP_termAttemptAuthorized	637	rw	sdl/t_bcsm/dp_terma.spd
Procedure DP_tNoAnswer	641	rw	sdl/t_bcsm/dp_tnoan.spd
Procedure DP_termAttempt	645	rw	sdl/t_bcsm/dpta.spd
Procedure DP_tReanswer	648	rw	sdl/t_bcsm/dptr.spd
Procedure DP_tBusy	651	rw	sdl/t_bcsm/dp_tbusy.spd
Procedure DP_tMidCall	655	rw	sdl/t_bcsm/dp_tmidc.spd
Procedure DP_Facility_Selected_and_Available	657	rw	sdl/t_bcsm/dpfsaa.spd
Procedure DP_termDenied	660	rw	sdl/t_bcsm/dp_tdenied.spd
Procedure DP_tAnswer	662	rw	sdl/t_bcsm/dp_tansw.spd
Procedure DP_tAbandon	665	rw	sdl/t_bcsm/dp_taban.spd
Procedure DP_tSuspend	666	rw	sdl/t_bcsm/dpts.spd
Procedure DP_tCall_Accepted	669	rw	sdl/t_bcsm/dp_tcall.spd
Procedure DP_tDisconnect	671	rw	sdl/t_bcsm/dp_disco.spd
Procedure SetNoAnswerTime	679	rw	sdl/t_bcsm/SetNoAnswerTime.spd
Procedure SetFIMAgent	680	rw	sdl/t_bcsm/SetFIMAgent.spd
Package TCAP_Simulator	681	rw	sdl/tcap/TCAP_Simulator.sun
TCAP_Simulator		rw	sdl/tcap/TCAP_Simulator.scu
Block Type TCAP_Simulator	682	rw	sdl/tcap/tcaps.sbt
Process Type TCAP_InterfaceHandler	684	rw	sdl/tcap/tcapi.spt
Operator newDialog	702	rw	sdl/tcap/newd.sop
Operator initDialogs	703	rw	sdl/tcap/initd.sop
Operator getDialogID	704	rw	sdl/tcap/getd.sop
Operator nextFreeDialogID	705	rw	sdl/tcap/nextfd.sop
Process Type TCAP_Dialog	706	rw	sdl/tcap/tcapd.spt
Operator newTCmessage	726	rw	sdl/tcap/newtcm.sop
Package CS3_TCAP	727	rw	sdl/tcap/CS3_TCAP.sun
CS3_TCAP		rw	sdl/tcap/CS3_TCAP.scu

OrganizerView

Chapter System Instantiations	730		
System CS3_INAP	730	rw	sdl/CS3_INAP.ssy

LinkView

Information from:
/home/mulligan/INAP/CoreINAPCS32/inap.sli

State Idle, CS2/cv1.spt

State Idle, CS2/csa.spt

Unknown DP (Disconnect) is not se, MSCs/FollowOnScenario1.msc

Statistics:
3 endpoints
0 links

Text readme

ETSI CoreINAP CS-3 SDLs

Version 6.0, 13 July 2000.
Version as approved for PE at the SPAN 3 Plenary meeting, 26-30 June 2000.

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.

ASN.1 Text INCS3opsargs

```
INCS3opsargs
--IN-CS3-SSF-SCF-ops-args {itu-t(0) identified-organization(4) etsi(0) inDomain(
1) in-network(1) cs3(30) modules(0) in-cs3-ssf-scf-ops-args(8) version1(0)}


DEFINITIONS IMPLICIT TAGS ::=

BEGIN

IMPORTS

--     common-classes,
--     common-datatypes,
--     errortypes,
--     scf-srf-classes,
--     scf-srf-datatypes,
--     ssf-scf-classes,
--     ssf-scf-datatypes,
--     operationcodes,
--     ros-InformationObjects ,
--     tc-Messages
--FROM IN-CS3-object-identifiers
--     {itu-t(0) identified-organization(4) etsi(0) inDomain(1) in-network(1) cs3(30
) modules(0) in-cs3-object-identifiers(0) version1(0)}

--     OPERATION

--FROM Remote-Operations-Information-Objects ros-InformationObjects


     AddOnChargingInformation,
     ChargingTariffInformation,
     ChargingMessageType,
--FROM Tariffing-DataTypes {itu-t(0) identified-organization etsi (0) 1296  vers
ion2(3)}

     ISDN-AddressString,
     IMSI,
--     Ext-BasicServiceCode,
-- Refer to 3G TS 29.002 for encoding.
--FROM MAP-CommonDataTypes {ccitt(0) identified-organization(4) etsi(0) mobileDo
main(0)
--gsm-Network(1) modules(3) map-CommonDataTypes(18) version6(6)}

--     CUG-Index,
--     CUG-Interlock,
--     LocationInformation,
-- Refer to 3G TS 29.002 for encoding.
--     SubscriberState,
--FROM MAP-MS-DataTypes {ccitt(0) identified-organization(4) etsi(0) mobileDomai
n(0)
--gsm-Network(1) modules(3) map-MS-DataTypes(11) version6(6)}



--     CallReferenceNumber,
-- Refer to 3G TS 29.002 for encoding.
--     SuppressionOfAnnouncement,
--FROM MAP-CH-DataTypes {ccitt(0) identified-organization(4) etsi(0) mobileDomai
n(0)
--gsm-Network(1) modules(3) map-CH-DataTypes(13) version6(6)}


--     COMMON-BOUNDS
--FROM IN-CS3-Common-Classes common-classes

--     SCF-SSF-BOUNDS
--FROM IN-CS3-SSF-SCF-Classes ssf-scf-classes

--     SCF-SRF-BOUNDS
```

```
--FROM IN-CS3-SCF-SRF-Classes scf-srf-classes


--     opcode-activateServiceFiltering,
--     opcode-activityTest,
--     opcode-applyCharging,
--     opcode-applyChargingReport,
--     opcode-assistRequestInstructions,
--     opcode-callGap,
--     opcode-callInformationReport,
--     opcode-callInformationRequest,
--     opcode-cancel,
--     opcode-cancelStatusReportRequest,
--     opcode-collectInformation,
--     opcode-connect,
--     opcode-connectToResource,
--     opcode-continue,
--     opcode-continueWithArgument,
--     opcode-createCallSegmentAssociation,
--     opcode-createOrRemoveTriggerData,
--     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-requestCurrentStatusReport,
--     opcode-requestEveryStatusChangeReport,
--     opcode-requestFirstStatusMatchReport,
--     opcode-requestNotificationChargingEvent,
--     opcode-requestReportBCSMEvent,
--     opcode-requestReportUTSI,
--     opcode-resetTimer,
--     opcode-selectFacility,
--     opcode-sendChargingInformation,
--     opcode-sendSTUI,
--     opcode-serviceFilteringResponse,
--     opcode-setServiceProfile,
--     opcode-splitLeg,
--     opcode-statusReport

--FROM IN-CS3-operationcodes operationcodes

     Extensions,
     Integer4,
     InvokeID,

--FROM IN-CS3-common-datatypes common-datatypes



                ActionIndicator,
                ActionPerformed,
                AChBillingChargingCharacteristics ,
                AChChargingAddress ,
                AdditionalCallingPartyNumber ,
                AlertingPattern,
                ApplicationTimer,
                AssistingSSPIPRoutingAddress ,
                BackwardGVNS ,
                BCSMEvent  ,
                BearerCapability ,
                CalledDirectoryNumber ,
```

```
              CalledPartyNumber ,
              CalledPartyBCDNumber ,
              CallingGeodeticLocation ,
              CallingPartyBusinessGroupID,
              CallingPartyNumber ,
              CallingPartysCategory,
              CallReference ,
              CallResult ,
--              CallSegmentID ,
              CallSupervision ,
              Carrier ,
              Cause ,
              CCSS,
              CGEncountered,
              ChargeNumber ,
              ChargingControlType,
              ChargingEvent ,
              CNInfo ,
              ControlType,
              CorrelationID ,
              CountersValue,
              CreateOrRemoveIndicator,
              CSAID ,
              CutAndPaste,
              DateAndTime,
              DefaultFaultHandling ,
              DestinationRoutingAddress ,
              Digits ,
              DisplayInformation ,
              XDuration,
              EventSpecificInformationBCSM ,
              EventSpecificInformationCharging ,
              EventTypeBCSM,
              EventTypeCharging ,
              EventTypeTariff,
              ExportLegsType,     -- for SDL model only
              FCIBillingChargingCharacteristics ,
              FilteredCallTreatment ,
              FilteringCharacteristics,
              FilteringCriteria ,
              FilteringTimeOut,
              ForwardCallIndicators,
              ForwardGVNS ,
              GapCriteria ,
              GapIndicators,
              GapTreatment ,
              GenericName ,
              GenericNumbers ,
              GlobalCallReference ,
              HighLayerCompatibility,
--              initialCallSegment,
              InitialDPArgExtension,
              INprofile ,
              INServiceCompatibilityIndication ,
              INServiceCompatibilityResponse,
              IPAvailable ,
              IPRoutingAddress ,
              IPSSPCapabilities ,
              ISDNAccessRelatedInformation ,
--              LegID,
--              leg1,
              LocationNumber ,
--              MiscCallInfo,
              MonitorMode,
              NAOliInfo,
              OCSIApplicable,
              OriginalCalledPartyID ,
              ProfileIdentifier ,
              Reason ,
              RedirectingPartyID ,
              RedirectionInformation,
              RegistratorIdentifier,
              ReportCondition,
```

```
               RouteingNumber ,
               RequestedInformationList ,
               RequestedInformationTypeList,
               RequestedUTSIList ,
               ResourceID ,
               ResourceStatus,
               ResponseCondition,
               RouteList ,
               ScfID ,
               SCIBillingChargingCharacteristics ,
               SDSSinformation ,
               ServiceInteractionIndicators ,
               ServiceInteractionIndicatorsTwo,
               ServiceKey,
               TDPIdentifier,
               TerminalType,
               TimeAndTimezone ,
--               TimerID,
               TimerValue,
               TriggerData,
               TriggerDataIdentifier ,
               TriggerDPType,
               Triggers  ,
               TriggerStatus,
               TriggerResults ,
               USIInformation ,
               USIServiceIndicator ,
               VPNIndicator,

--SRF Datatypes
     InformationToSend,
     CollectedInfo ,
     MailBoxID ,
     InformationToRecord ,
     Media,
     ReceivedStatus,
     RecordedMessageID,
     SRFGapCriteria ,

     Code,
     GenericNumber,

               numOfINProfile,
               numOfBCSMEvents,
               numOfChargingEvents,
               numOfLegs,
               numOfCSs,
               maxRecordedMessageUnits

FROM INCS3datatypes--IN-CS3-SSF-SCF-datatypes ssf-scf-datatypes

--     InformationToSend
--FROM IN-CS3-scf-srf-datatypes scf-srf-datatypes

--     cancelFailed,
--     eTCFailed,
--     improperCallerResponse,
--     missingCustomerRecord,
--     missingParameter,
--     parameterOutOfRange,
--     requestedInfoError,
--     systemFailure,
--     taskRefused,
--     unavailableResource,
--     unexpectedComponentSequence,
--     unexpectedDataValue,
--     unexpectedParameter,
--     unknownLegID,
--     unknownResource
--FROM IN-CS3-errortypes errortypes
;
-- The following three definitions are local short-hand notation for convenience
.
```

```
--B1 ::= COMMON-BOUNDS           defined in Part 1.1
--B2 ::= SCF-SSF-BOUNDS           defined in this part
--B3 ::= SCF-SRF-BOUNDS           defined in Part 1.3


-- Definitions of MAP and ISUP types imported above.

CallReferenceNumber ::= INTEGER


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


SubscriberState ::= CHOICE {
     assumedIdle     [0] NULL,
     camelBusy     [1] NULL,
     netDetNotReachable     NotReachableReason,
     notProvidedFromVLR     [2] NULL}

NotReachableReason ::= ENUMERATED {
     msPurged (0),
     imsiDetached (1),
     restrictedArea (2),
     notRegistered (3)}

SuppressionOfAnnouncement ::= NULL

LocationInformation ::= OCTET STRING

CUG-Index ::= INTEGER (0..32767)
     -- The internal structure is defined in ETS 300 138.

CUG-Interlock ::= OCTET STRING (SIZE (4))

-- *****
-- TYPE DEFINITIONS FOR IN CS-3 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.

UnavailableNetworkResource ::= ENUMERATED {
     unavailableResources(0),
     componentFailure(1),
     basicCallProcessingException(2),
     resourceStatusFailure(3),
     endUserFailure(4),
     screening(5)
     }
-- Indicates the network resource that failed
-- Note that in IN CS3 the screening value can only be used by the SCF.

-- *****

-- *****
-- Due to a restriction in SDT it is not possible to denote values of types defi
ned in an inmported
-- module (for example in DEFAULT statements). Therefore the types for which DEF
AULT 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)
                         }
          }
requested MiscCallInfo ::= {messageType request}
notified MiscCallInfo ::= {messageType notification}

TimerID ::= ENUMERATED {
     tssf(0)
     }

-- *****



-- Operations and Arguments:

-- Direction: SCF -> SSF, Timer: Tasf
-- When receiving this operation, the SSF handles calls to destination in a spec
ified manner
-- without  sending queries for every detected call. It is used for example for
providing
-- televoting or mass calling services. Simple registration functionality (count
ers) and
-- announcement control may be  located at the SSF. The operation initializes th
e 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] Extensions            OPTIONAL
     }

-- Direction: SCF -> SSF or SSF-> SCF, 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 wi
ll take the
-- appropriate action..  As an option, this operation may be used in the reverse
 direction by the
-- SSF to check for the continued existence of a relationship with the SCF.


-- 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.
-- This operation is can also be used to instruct the SSF to release the call re
garding
-- some condition.

ApplyChargingArg  ::= SEQUENCE {
     aChBillingChargingCharacteristics     [0] AChBillingChargingCharacteristics  OPTION
AL,
     -- parameter is made optional to allow backward compatibility and to avoid a ma
ndatory use
     -- of a CS1 defined operator specific Octet string.
     sendCalculationToSCPIndication          [1]     BOOLEAN OPTIONAL,
     --     parameter must be TRUE when given, otherwise the operation is rejected.
     aChChargingAddress                         [2] AChChargingAddress,--  DEFAULT legID:s
eg1,
     extensions                                   [3] Extensions       OPTIONAL,
     callSupervision                              [52] CallSupervision      OPTIONAL
     }


-- 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

-- Direction: SSF  -> SCF or SRF  -> SCF, Timer: Tari
-- This operation is used when there is an assist or a hand-off procedure and ma
y be sent by the SSF
-- or SRF to the SCF. This operation is sent by the assisting SSF to SCF, when t
he 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.

AssistRequestInstructionsArg  ::= SEQUENCE {
     correlationID          [0] CorrelationID ,
     iPAvailable               [1] IPAvailable           OPTIONAL,
     iPSSPCapabilities     [2] IPSSPCapabilities      OPTIONAL,
     extensions               [3] Extensions            OPTIONAL
     }
-- OPTIONAL denotes network operator specific use. The value of the correlationI
D 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 specif
ic
-- service requests are sent to the  SCF.
```

```
-- Use of this operation by the SCF to gap queries and updates at the SDF is out
side
-- the scope of this capability set .

CallGapArg  ::= SEQUENCE {
     gapCriteria                        [0] GapCriteria ,
     gapIndicators                   [1] GapIndicators,
     controlType                        [2] ControlType     OPTIONAL,
     gapTreatment                    [3] GapTreatment       OPTIONAL,
     extensions                         [4] Extensions                 OPTIONAL
     }
-- OPTIONAL denotes network operator optional. If gapTreatment is not present, t
he SSF will use
-- a default treatment depending on network operator implementation.


-- Direction: SSF ->  SCF, Timer: Tcirp
-- 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] Extensions       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 abo
ut a
-- single call and report it to
-- the SCF (with a CallInformationReport operation).

CallInformationRequestArg  ::= SEQUENCE {
     requestedInformationTypeList      [0] RequestedInformationTypeList,
     extensions                              [2] Extensions       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 requ
ests
-- This operation can also be used to cancel all outstanding requests and enable
 the
-- state machine (SSF)to go to idle.
-- In  this case the Cancel operation  does not specify any specific operation t
o be cancelled.
-- For the SCF-SRF operations that can be cancelled, refer to Part 3 of Q.1238

CancelArg  ::= CHOICE {
     invokeID                                  [0] InvokeID,
     allRequests                                  [1] NULL,
     callSegmentToCancel                        [2] SEQUENCE {
               invokeID                             [0] InvokeID,
               callSegmentID                        [1] CallSegmentID
               },
     allRequestsForCallSegment               [3] CallSegmentID
     }
-- The InvokeID has the same value as that which was used for the SCF-SRF  opera
tion,
-- i.e. is used to identify the correlated previous SCF-SRF operation to be canc
celled.

-- Direction: SCF ->  SSF, Timer:  Tcsr
-- This operation cancels the following processes: RequestFirstStatusMatchReport
 and
-- RequestEveryStatusChangeReport.

CancelStatusReportRequestArg  ::= SEQUENCE {
     resourceID     [0] ResourceID      OPTIONAL,
```

```
     extensions     [1] Extensions      OPTIONAL
     }

-- Direction: SCF ->  SSF, Timer: Tci
-- This operation is used to request the SSF to perform the originating basic ca
ll
-- 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] Extensions            OPTIONAL
     }

-- Direction: SCF ->  SSF, Timer: Tcon
-- This operation is used to request the SSF to perform the call processing acti
ons to route
-- or forward a call to a specified destination. To do so, the SSF may or may no
t use
-- destination information from the calling party (e.g. dialed digits) and exist
ing 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 proc
essing
-- resumes at PIC Analyse_Information in the O-BCSM.
--  When address information and routing information is included, call processin
g resumes at PIC
--  Select_Route.

ConnectArg  ::= SEQUENCE {
     destinationRoutingAddress                [0] DestinationRoutingAddress ,
     alertingPattern                      [1] AlertingPattern
     correlationID                          [2] CorrelationID                        OPTI
     cutAndPaste                              [3] CutAndPaste
     iSDNAccessRelatedInformation           [5] ISDNAccessRelatedInformation      OPTIONAL
     originalCalledPartyID                    [6] OriginalCalledPartyID                OP
     routeList                                  [7] RouteList
     scfID                                       [8] ScfID
     extensions                                  [10] Extensions
     carrier                                       [11] Carrier
     serviceInteractionIndicators           [26] ServiceInteractionIndicators      OPTIONAL
     callingPartyNumber                          [27] CallingPartyNumber                OPT
     callingPartysCategory                    [28] CallingPartysCategory
     redirectingPartyID                          [29] RedirectingPartyID                OPT
     redirectionInformation                    [30] RedirectionInformation
     displayInformation                          [12] DisplayInformation                OPT
     forwardCallIndicators                    [13] ForwardCallIndicators
     genericNumbers                              [14] GenericNumbers                     O
     serviceInteractionIndicatorsTwo          [15] ServiceInteractionIndicatorsTwo     OPT

     iNServiceCompatibilityResponse          [16] INServiceCompatibilityResponse
     forwardGVNS                                   [17] ForwardGVNS
     backwardGVNS                              [18] BackwardGVNS
     chargeNumber                              [19] ChargeNumber
     callSegmentID                              [20] CallSegmentID
     legToBeCreated                              [21] LegID
     sDSSinformation                              [22] SDSSinformation
     calledDirectoryNumber                    [23] CalledDirectoryNumber            OPTION
     bearerCapability                         [24] BearerCapability                     O
     cug-Interlock                              [31] CUG-Interlock
     cug-OutgoingAccess                         [32] NULL
     suppressionOfAnnouncement               [55] SuppressionOfAnnouncement
     oCSIApplicable                              [56] OCSIApplicable
     na-OliInfo                              [57] NAOliInfo
     }
--     na-OliInfo is included at the discretion of the gsmSCF operator.
-- OPTIONAL parameters are only provided if modifications desired to basic call

-- processing values
-- TAG 4, 9, 50 and 51 are reserved and shall not be used.
```

```
-- Direction: SCF ->  SSF, Timer: Tctr
-- This operation is used to connect a call from the SSP to the physical entity
containing the SRF.

ConnectToResourceArg  ::= SEQUENCE {
     resourceAddress     CHOICE {
          ipRoutingAddress               [0] IPRoutingAddress ,
          legID                            [1] LegID,
          ipAddressAndLegID               [2] SEQUENCE {
                ipRoutingAddress                    [0] IPRoutingAddress ,
                legID                                   [1] LegID
                },
          nnone                            [3] NULL,
          callSegmentID                   [5] CallSegmentID  ,
          ipAddressAndCallSegment         [6] SEQUENCE {
                ipRoutingAddress                    [0] IPRoutingAddress ,
                callSegmentID                         [1] CallSegmentID
                }
          },
     extensions                            [4]  Extensions
     serviceInteractionIndicators     [30] ServiceInteractionIndicators           OPTIONAL
     serviceInteractionIndicatorsTwo     [7]  ServiceInteractionIndicatorsTwo          OPT

     uSIServiceIndicator                    [35] USIServiceIndicator                     O
     uSIInformation                         [36] USIInformation                          O
     }

-- 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 le
gs 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 at
-- 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:sendingSid
     alertingPattern                       [1] AlertingPattern
     genericName                            [2] GenericName
     iNServiceCompatibilityResponse     [3] INServiceCompatibilityResponse              O
     forwardGVNS                     [4] ForwardGVNS                                OPTIONA
     backwardGVNS                    [5] BackwardGVNS                                OPTION
     extensions                            [6]  Extensions
     serviceInteractionIndicatorsTwo [7] ServiceInteractionIndicatorsTwo              OPTI
AL,
     sDSSinformation                       [8] SDSSinformation
     iSDNAccessRelatedInformation     [19] ISDNAccessRelatedInformation          OPTIONAL,
     originalCalledPartyID               [10] OriginalCalledPartyID                     OP
     callingPartyNumber                   [11] CallingPartyNumber                      OPT
     callingPartysCategory               [12] CallingPartysCategory
     redirectingPartyID                   [13] RedirectingPartyID                      OPT
     redirectionInformation               [14] RedirectionInformation
     forwardCallIndicators               [15] ForwardCallIndicators
     genericNumbers                       [16] GenericNumbers                            O
     cug-Interlock                        [17] CUG-Interlock
     cug-OutgoingAccess                   [18] NULL
     chargeNumber                        [50] ChargeNumber
     carrier                                  [52] Carrier
     suppressionOfAnnouncement         [55] SuppressionOfAnnouncement                   O
 na-OliInfo                       [56] NAOliInfo                                       OP
```

```
     }
-- OPTIONAL parameters are only provided if modifications desired to basic call
processing values

-- Tag value 51 is reserved.

-- 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 CS
A which is
-- unique within the SSF.

CreateCallSegmentAssociationArg  ::= SEQUENCE {
     extensions                        [0] Extensions  OPTIONAL
     }

CreateCallSegmentAssociationResultArg  ::= SEQUENCE {
     newCallSegmentAssociation     [0] CSAID ,
     extensions                        [1] Extensions  OPTIONAL
     }

-- Direction: SCF  ->  SSF,  Class 1, Timer: Tcrt
--  This trigger management operation is used by the SCF outside the context of
a call
--  to create a new trigger detection point in the CCF/SSF by downloading trigge
r data
-- (e.g. triggering criteria, ServiceKey, SCF address,....)
-- or to remove an existing trigger.


CreateOrRemoveTriggerDataArg   ::= SEQUENCE {
     createOrRemove               [0] CreateOrRemoveIndicator,-- DEFAULT crreate,
     dPName                      [1] EventTypeBCSM                              OPTIONAL,
     triggerDPType               [2] TriggerDPType,--      DEFAULT tdp-r,
     serviceKey              [3]  ServiceKey                                OPTIONAL,
     profile                   [4] ProfileIdentifier                     OPTIONAL,
     triggerData                 [5] TriggerData                            OPTIONAL,
     defaultFaultHandling      [6] DefaultFaultHandling           OPTIONAL,
     tDPIdentifier               [7] TDPIdentifier                        OPTIONAL,
     extensions              [30] Extensions                              OPTIONAL
     }

CreateOrRemoveTriggerDataResultArg   ::= SEQUENCE {
     triggerStatus           [0] TriggerStatus,
     tDPIdentifier           [1] TDPIdentifier ,

     registratorIdentifier      [2] RegistratorIdentifier                    OPTIONAL,
     extensions              [30]  Extensions                             OPTIONAL
     }

-- Direction: SCF ->  SSF, Timer: Tdfc
-- This operation is used to disconnect a forward temporary connection or a
-- connection to a resource.


-- Direction: SCF ->  SSF, Timer: Tdfcwa
-- This operation is used to disconnect a forward temporary connection or a
-- connection to a resource.

DisconnectForwardConnectionWithArgumentArg  ::= SEQUENCE {
     partyToDisconnect     CHOICE {
               legID                        [0] LegID,
               callSegmentID               [1] CallSegmentID
               },
     extensions                             [2] Extensions                      OPTIONAL,
     uSIServiceIndicator                   [3] USIServiceIndicator           OPTIONAL,
     uSIInformation                  [4] USIInformation                OPTIONAL
     }

--     Direction: SCF -> SSF. Timer: T dl
--     This operation  is issued by the SCF  to release a specific leg associated wi
```

```
th 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.

DisconnectLegArg  ::= SEQUENCE {
     legToBeReleased          [0] LegID,
     releaseCause          [1] Cause                    OPTIONAL,
     extensions               [2] Extensions               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 per
iod of time
-- (e.g. to play an announcement, to collect user information); it implies the u
se of the assist
-- procedure.

EstablishTemporaryConnectionArg  ::= SEQUENCE {
     assistingSSPIPRoutingAddress     [0] AssistingSSPIPRoutingAddress ,
     correlationID                         [1] CorrelationID                         OPTI
     partyToConnect     CHOICE {
                         legID                    [2] LegID,
                         callSegmentID          [7] CallSegmentID
                                             }
     scfID                                   [3] ScfID
     extensions                              [4] Extensions                           OPTI
     carrier                                   [5] Carrier                               O
     serviceInteractionIndicators     [30] ServiceInteractionIndicators      OPTIONAL,
     serviceInteractionIndicatorsTwo     [6] ServiceInteractionIndicatorsTwo          OPTI
     na-OliInfo                              [50] NAOliInfo
     chargeNumber                          [51] ChargeNumber                         OPTIO
     }
-- OPTIONAL parameters are only provided if modifications desired to basic call
processing values

-- 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      OPTIONAL,
     eventSpecificInformationCharging     [1] EventSpecificInformationCharging   OPTIONA
L,
     legID                                        [2] LegID     OPTIONAL,
     extensions                                   [3] Extensions
     monitorMode                                   [30] MonitorMode, --    DEFAULT notifyA
     eventTypeTariff                              [50] EventTypeTariff
     eventSpecificInformationTariff          [51] ChargingMessageType
     }
-- 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 e
vents such as
-- busy or no answer) previously requested by the SCF in a RequestReportBCSMEven
t operation.

EventReportBCSMArg  ::= SEQUENCE {
     eventTypeBCSM                              [0] EventTypeBCSM,
          eventSpecificInformationBCSM[2] EventSpecificInformationBCSM       OPTIONAL,
     legID                                        [3] LegID
     miscCallInfo                             [4] MiscCallInfo     DEFAULT     {messageType re
s                              [5]  Extensions                            OPTIONAL
     }

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

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                      O
     callingPartyNumber                            [3] CallingPartyNumber                 OPTI
     callingPartyBusinessGroupID                  [4] CallingPartyBusinessGroupID
     callingPartysCategory                       [5] CallingPartysCategory
     cGEncountered                                 [7] CGEncountered
     iPSSPCapabilities                            [8] IPSSPCapabilities                      O
     iPAvailable                                      [9] IPAvailable
     locationNumber                                 [10] LocationNumber                      O
     originalCalledPartyID                       [12] OriginalCalledPartyID                 OP
     terminalType                                  [14] TerminalType
     extensions                                       [15] Extensions
     highLayerCompatibility                       [23] HighLayerCompatibility
     serviceInteractionIndicators              [24] ServiceInteractionIndicators  OPTIONAL,
     additionalCallingPartyNumber              [25] AdditionalCallingPartyNumber      OPTIONAL
     forwardCallIndicators                        [26] ForwardCallIndicators
     bearerCapability                             [27] BearerCapability                      OP
     eventTypeBCSM                                 [28] EventTypeBCSM
     redirectingPartyID                            [29] RedirectingPartyID                OPT
     redirectionInformation                       [30] RedirectionInformation
     cause                                            [17] Cause
     iSDNAccessRelatedInformation              [21] ISDNAccessRelatedInformation      OPTIONAL
     iNServiceCompatibilityIndication         [22] INServiceCompatibilityIndication
ONAL,
     genericNumbers                                 [31] GenericNumbers                      O
     serviceInteractionIndicatorsTwo             [32] ServiceInteractionIndicatorsTwo     OPT

     forwardGVNS                                      [33] ForwardGVNS
     createdCallSegmentAssociation              [34] CSAID                              OPTIO
     uSIServiceIndicator                            [35] USIServiceIndicator                 O
     uSIInformation                                 [36] USIInformation                      O
     carrier                                          [37] Carrier
     cCSS                                           [38] CCSS
     vPNIndicator                                 [39] VPNIndicator
     cNInfo                                           [40] CNInfo
     callReference                                 [41] CallReference
     routeingNumber                                 [42] RouteingNumber                      O
     callingGeodeticLocation                       [43] CallingGeodeticLocation           OPT
     globalCallReference                            [44] GlobalCallReference                 O
     cug-Index                                      [45] CUG-Index
     cug-Interlock                                 [46] CUG-Interlock
     cug-OutgoingAccess                            [47] NULL
     iMSI                                           [50] IMSI
     subscriberState                                [51] SubscriberState
     locationInformation                            [52] LocationInformation
     ext-basicServiceCode                        [53] Ext-BasicServiceCode
     callReferenceNumber                            [54] CallReferenceNumber
```

```
     mscAddress                                  [55] ISDN-AddressString
     calledPartyBCDNumber                    [56] CalledPartyBCDNumber              OPTIO
     timeAndTimezone                             [57] TimeAndTimezone
     gsm-ForwardingPending                    [58] NULL
     initialDPArgExtension                    [59] InitialDPArgExtension
     }


-- for mscAddress for encoding see 3G TS 29.002
-- OPTIONAL for iPSSPCapabilities, iPAvailable, cGEncountered, and miscCallInfo
denotes network
-- operator specific use.
-- OPTIONAL for terminalType indicates that this parameter applies only at origi
nating
-- or terminating
-- local exchanges if the SSF has this information.
-- Tag values 1, 6, 11, 13, 16, 18, 19, 20 are reserved and shall not be used


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

InitiateCallAttemptArg  ::= SEQUENCE {
     destinationRoutingAddress          [0] DestinationRoutingAddress ,
     alertingPattern                         [1] AlertingPattern
     iSDNAccessRelatedInformation     [2] ISDNAccessRelatedInformation           OPTIONAL,
     extensions                              [4]  Extensions
     serviceInteractionIndicators     [29] ServiceInteractionIndicators           OPTIONAL
     callingPartyNumber                    [30] CallingPartyNumber                    OPT
     legToBeCreated                         [5] LegID      DEFAULT       sendingSideID:leg
     newCallSegment                         [6] CallSegmentID  DEFAULT initialCallSegment,
     iNServiceCompatibilityResponse     [7] INServiceCompatibilityResponse               O
     serviceInteractionIndicatorsTwo     [8] ServiceInteractionIndicatorsTwo

     carrier                                     [9] Carrier
     correlationID                         [10] CorrelationID                         OPT
     scfID                                   [11] ScfID
     callReference                         [12] CallReference                          OP
     calledDirectoryNumber                [13] CalledDirectoryNumber                 OPTION
     originalCalledPartyID                [14] OriginalCalledPartyID                    OP
     callingPartysCategory                [15] CallingPartysCategory
     redirectingPartyID                    [16] RedirectingPartyID                    OPT
     redirectionInformation                [17] RedirectionInformation
     displayInformation                    [18] DisplayInformation                    OPT
     forwardCallIndicators                [19] ForwardCallIndicators
     genericNumbers                         [20] GenericNumbers                          O
     forwardGVNS                              [21] ForwardGVNS
     bearerCapability                     [22] BearerCapability                       OP
     globalCallReference                    [23] GlobalCallReference                   O
     cug-Interlock                         [24] CUG-Interlock
     cug-OutgoingAccess                    [25] NULL
     }
-- Tag values 3, 50, 51, 52 are reserved and shall not be used
-- OPTIONAL parameters are only provided if modifications desired to basic call
processing values

-- Direction: SCF  ->  SSF,  Class 1, Timer: Tmtd
-- This trigger management operation is used outside the context of a call to ac
tivate,
-- deactivate or retrieve
-- the status of one or several trigger detection point linked to a subscriber p
rofile known at
-- the switch, e.g. related to an access line ( i.e. an individual trigger).

ManageTriggerDataArg   ::= SEQUENCE {
     actionIndicator              [0] ActionIndicator,
     triggerDataIdentifier     CHOICE {
            profileAndDP                     [1] TriggerDataIdentifier ,
            -- one trigger
            profile                       [5] ProfileIdentifier
          },
```

```
     registratorIdentifier     [2] RegistratorIdentifier                    OPTIONAL,
     extensions                   [3] Extensions                               OPTIONAL,
     tDPIdentifier               [4] TDPIdentifier                           OPTIONAL
     }

ManageTriggerDataResultArg   ::= CHOICE
     {
     oneTriggerResult                 SEQUENCE {
                    actionPerformed              [0]  ActionPerformed,
                    extensions                   [1] Extensions            OPTIONAL  },
     severalTriggerResult      [1] SEQUENCE {
                      results                         [0] TriggerResults ,
                     extensions                 [1] Extensions           OPTIONAL }
     }

--     Direction: SCF -> SSF. Timer: T mc
-- This operation  is issued by the SCF  to merge two associated CSs , into one
CS .

MergeCallSegmentsArg  ::= SEQUENCE {
     sourceCallSegment               [0] CallSegmentID ,
     targetCallSegment               [1] CallSegmentID  DEFAULT initialCallSegment,
     extensions                         [2] Extensions                     OPTIONAL
     }

--  Direction: SCF -> SSF, Timer Tmcs
-- This operation moves a CS from the source CSA to the the target CSA

MoveCallSegmentsArg  ::= SEQUENCE {
     targetCallSegmentAssociation     [0] CSAID ,
     callSegments                         [1] SEQUENCE SIZE (1..numOfCSs)
 OF SEQUENCE {
          sourceCallSegment                     [0] CallSegmentID       DEFAULT initialCall
          newCallSegment                          [1] CallSegmentID
          },
     legs                                  [2] ExportLegsType, --changed for SDL model
     extensions                             [3]  Extensions                OPTIONAL
     }

-- 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] Extensions       OPTIONAL
     }


-- Direction: SCF ->  SSF, Timer: Trc
-- This operation is used by the SCF to tear down an existing call segment at an
y phase of
-- the call for all parties involved in the call segment or to tear down all exi
sting
-- call segments within a Call Segment Association.

ReleaseCallArg  ::= CHOICE {
     initialCallSegment                          Cause ,
     callSegmentToRelease          [1] SEQUENCE {
          callSegment                       [0] INTEGER (1..numOfCSs),
          releaseCause                   [1] Cause                      OPTIONAL,
          forcedRelease                   [2] BOOLEAN  DEFAULT  FALSE
          },
     allCallSegments                    [2] SEQUENCE {
           releaseCause                   [0] Cause                      OPTIONAL,
          timeToRelease                   [1] TimerValue                      OPTIONAL,
          forcedRelease                   [2] BOOLEAN  DEFAULT  FALSE
          }
     }
-- A default cause value of decimal 31 (normal unspecified) should be coded appr
opriately.
```

```
-- If timeToRelease parameter is omitted, the default  shall be no timed disconn
ect  requested
-- If forcedRelease parameter is omitted  (default value  "FALSE") the default s
hall be no
-- forced release requested.

-- 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 (USI)  to t
he SCF.

ReportUTSIArg  ::= SEQUENCE {
     uSIServiceIndicator     [0] USIServiceIndicator ,
     legID                    [1] LegID      DEFAULT receivingSideID:leg1,
     uSIInformation          [2] USIInformation ,
     extensions               [3] Extensions                     OPTIONAL
     }

-- Direction: SCF ->  SSF, Timer: Trcs
-- This operation is used to request the SSF to report immediately the busy/idle
 status
-- of a physical termination resource.

RequestCurrentStatusReportArg  ::= ResourceID

RequestCurrentStatusReportResultArg  ::= SEQUENCE {
     resourceStatus     [0] ResourceStatus,
     resourceID          [1] ResourceID      OPTIONAL,
     extensions          [2] Extensions       OPTIONAL
     }

-- Direction: SCF ->  SSF, Timer: Tres
-- This operation is used to request the SSF to report every change of busy/idle
 status
-- of a physical termination resource.

RequestEveryStatusChangeReportArg  ::= SEQUENCE {
     resourceID               [0] ResourceID ,
     correlationID          [1] CorrelationID                OPTIONAL,
     monitorDuration          [2] XDuration                         OPTIONAL,
     extensions               [3] Extensions                  OPTIONAL
     }
-- For correlationID OPTIONAL denotes network operator optional.
-- monitorDuration is required if outside the context of a call.
-- It is not expected if we are in the context of a call, because in that case
-- the end of the call implicitly means the end of the monitoring.

-- Direction: SCF ->  SSF, Timer: Trfs
-- This operation is used to request the SSF to report the first change busy/idl
e to
-- the specified status of a physical termination resource.

RequestFirstStatusMatchReportArg  ::= SEQUENCE {
     resourceID               [0] ResourceID                OPTIONAL,
     resourceStatus          [1] ResourceStatus     OPTIONAL,
     correlationID          [2] CorrelationID                OPTIONAL,
     monitorDuration          [3] XDuration                         OPTIONAL,
     extensions               [4] Extensions                  OPTIONAL,
     bearerCapability     [5] BearerCapability           OPTIONAL
     }
-- For correlationID OPTIONAL denotes network operator optional.
-- monitorDuration is required if outside the context of a call. It is not expec
ted if we are in
-- the context of a call, because in that case the end of the call implicitly me
ans the end of
-- the monitoring.

-- Direction: SCF ->  SSF, Timer: Trnc
-- This operation  is used by the SCF to instruct the SSF on how to manage the c
harging events
-- which are received  from other FE's and not under control of the service logi
c instance.
```

```
RequestNotificationChargingEventArg  ::= SEQUENCE SIZE(1..numOfChargingEvents) O
F
           ChargingEvent

-- Direction: SCF ->  SSF, Timer: Trrb


RequestReportBCSMEventArg  ::= SEQUENCE {
     bcsmEvents                    [0] SEQUENCE SIZE(1..numOfBCSMEvents)  OF
                                                BCSMEvent ,
          extensions               [2]   Extensions       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 requ
est 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] Extensions                    OPTIONAL,
     legID                    [2] LegID     DEFAULT sendingSideID:leg1
     }

-- 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] Extensions                 OPTIONAL,
     callSegmentID     [3] CallSegmentID                OPTIONAL
     }

-- Direction: SCF ->  SSF, Timer: Tsf
-- This operation is used to request the SSF to perform the terminating basic ca
ll processing
-- actions to select the terminating line if it is idle,. If no idle line, the S
SF
-- determines that the terminating facility is busy.


SelectFacilityArg  ::= SEQUENCE {
     alertingPattern                         [0] AlertingPattern
     extensions                              [6]  Extensions
     serviceInteractionIndicatorsTwo     [12] ServiceInteractionIndicatorsTwo          OPT

     legID                                   [16] LegID
     }
-- OPTIONAL parameters are only provided if modifications desired to basic call
processing values.

-- Direction: SCF ->  SSF, Timer: Tsci
-- This operation is used to  instruct the SSF on the charging information to se
nd by the SSF.
-- The charging information can either be sent back by means of signalling or in
ternal
-- 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 standa
rd call record.

SendChargingInformationArg ::= SEQUENCE {
     sCIBillingChargingCharacteristics     [0] SCIBillingChargingCharacteristics  OPTION
AL,
     partyToCharge                                [1] LegID,
     extensions                                     [2] Extensions
     tariffMessage                                [50] ChargingMessageType          OPTIONAL
```

```
     chargingControl                              [51] ChargingControlType --DEFAULT {tari
angeSCI { }}
     }

-- 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 (USI  inf
ormation)
-- data element to the indicated user.

SendSTUIArg  ::= SEQUENCE {
     uSIServiceIndicator     [0] USIServiceIndicator ,
     legID                    [1] LegID     DEFAULT sendingSideID:leg1,
     uSIInformation          [2] USIInformation ,
     extensions               [3] Extensions                    OPTIONAL
     }

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

ServiceFilteringResponseArg  ::= SEQUENCE {
     countersValue          [0] CountersValue,
     filteringCriteria     [1] FilteringCriteria ,
     extensions               [2] Extensions                OPTIONAL,
     responseCondition     [3] ResponseCondition              OPTIONAL
     }

-- Direction SCF -> SSF, Timer Tsep
-- This operation is used within the context of a call to request the SSF to
-- activate/de-activate  a list of trigger for one of the parties in the call.

SetServiceProfileArg  ::= SEQUENCE {
          iNprofiles      [0]  SEQUENCE SIZE (1..numOfINProfile)
                              OF INprofile ,
          extensions     [30] Extensions           OPTIONAL
          }

--  Direction: SCF -> SSF. Timer: T sl
--  This operation  is issued by the SCF  to separate one joined leg from a mult
i-way connection
--  or  a single 2 party Call segment.

SplitLegArg  ::= SEQUENCE {
     legToBeSplit          [0] LegID,
     newCallSegment          [1] INTEGER (2..numOfCSs),
     extensions               [2] Extensions                    OPTIONAL
     }

-- Direction: SSF ->  SCF, Timer: Tsrp
-- This operation is used as a response to RequestFirstStatusMatchReport or
-- RequestEveryStatusChangeReport operations.

StatusReportArg  ::= SEQUENCE {
     resourceStatus          [0] ResourceStatus                   OPTIONAL,
     correlationID          [1] CorrelationID               OPTIONAL,
     resourceID               [2] ResourceID                OPTIONAL,
     extensions               [3] Extensions                OPTIONAL,
     reportCondition          [4] ReportCondition                  OPTIONAL
     }
-- For correlationID, OPTIONAL denotes network operator optional.
-- resourceID is required when the SSF sends a report as an answer to a previous
 request when the
-- correlationID was present.




-- Direction: SCF -> SRF, Timer: Tpa
-- This operation is to be used after Establish Temporary Connection (assist pro
cedure with a second SSP)
```

```
-- or a Connect to Resource (no assist) operation. It may be used for inband int
eraction 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 b
e correctly correlated.

PlayAnnouncementArg  ::= SEQUENCE {
     informationToSend     [0] InformationToSend ,
     disconnectFromIPForbidden     [1] BOOLEAN     DEFAULT TRUE,
     requestAnnouncementComplete     [2] BOOLEAN     DEFAULT TRUE,
     extensions     [3] Extensions      OPTIONAL,
     connectedPartyCHOICE {
               legID     [4] LegID,
               callSegmentID     [5] CallSegmentID
               }      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] Extensions       OPTIONAL,
     callSegmentID     [4] CallSegmentID       OPTIONAL
     }

ReceivedInformationArg  ::= CHOICE {
     digitsResponse     [0] Digits ,
     iA5Response     [1] IA5String,
     modemdetected      [2] BOOLEAN
}

--  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] Extensions      OPTIONAL,
     subscriberID     [4] GenericNumber      OPTIONAL,
     mailBoxID     [5] MailBoxID      OPTIONAL,
     informationToRecord     [6] InformationToRecord ,
     media     [7] Media , --     DEFAULT voiceMail,
     callSegmentID     [8] CallSegmentID      OPTIONAL
     }

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


-- Direction:SCF-> SRF. Timer :TCl
-- This operation is issued by the SCF to deallocate the resources used to perfo
rm the
-- instance of the "User Interaction" script : the context is released.

ScriptCloseArg  ::= SEQUENCE {
     uIScriptId     Code, --UISCRIPT.&id({SupportedUIScripts }),
     uIScriptSpecificInfo     [0] OCTET STRING OPTIONAL,
     extensions     [1] Extensions      OPTIONAL,
     callSegmentID     [2] CallSegmentID      OPTIONAL
     }
```

```
-- Direction:SRF-> SCF. Timer :TRe
-- This operation is issued by the SRF to return information to the SCF on the r
esults of the
-- execution of the instance of User Interaction script.

ScriptEventArg  ::= SEQUENCE {
     uIScriptId     Code, --UISCRIPT.&id({SupportedUIScripts }),
     uIScriptResult     [0] OCTET STRING OPTIONAL,
     extensions     [1] Extensions      OPTIONAL,
     callSegmentID     [2] CallSegmentID      OPTIONAL,
     lastEventIndicator     [3] BOOLEAN      DEFAULT  FALSE
     }

-- Direction:SCF-> SRF. Timer :Tinf

ScriptInformationArg ::= SEQUENCE {
     uIScriptId     Code, --UISCRIPT.&id({SupportedUIScripts }),
     uIScriptSpecificInfo     [0] OCTET STRING OPTIONAL,
     extensions     [1] Extensions      OPTIONAL,
     callSegmentID     [2] CallSegmentID      OPTIONAL
     }

-- Direction:SCF-> SRF. Timer :Tru
-- This operation is issued by the SCF to allocate the necessary resources to pe
rform the
-- instance of the "User Interaction" script and then to activate this "User Int
eraction" script
-- instance. A context is partially defined for it if necessary.

ScriptRunArg  ::= SEQUENCE {
     uIScriptId     Code, --UISCRIPT.&id({SupportedUIScripts }),
     uIScriptSpecificInfo     [0] OCTET STRING OPTIONAL,
     extensions     [1] Extensions      OPTIONAL,
     disconnectFromIPForbidden     [2] BOOLEAN     DEFAULT TRUE,
     callSegmentID     [3] CallSegmentID      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

-- Direction: SRF ->  SCF, Timer: Tcg
-- This operation is used to request the SCF  to reduce the rate at which specif
ic service requests are sent to
-- the  SRF.

SRFCallGapArg  ::= SEQUENCE {
     sRFgapCriteria     [0] SRFGapCriteria ,
     gapIndicators     [1] GapIndicators,
     controlType     [2] ControlType     OPTIONAL,
extensions     [30] Extensions       OPTIONAL
     }


END
```

ASN.1 Text INCS3BundleArg

```
INCS3BundleArg
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 ,
     CancelStatusReportRequestArg,
     CollectInformationArg ,
     ConnectArg ,
     ConnectToResourceArg,
     ContinueWithArgumentArg ,
     CreateCallSegmentAssociationArg ,
     CreateCallSegmentAssociationResultArg ,
     CreateOrRemoveTriggerDataArg,
     CreateOrRemoveTriggerDataResultArg,
     DisconnectForwardConnectionWithArgumentArg ,
     DisconnectLegArg ,
     EntityReleasedArg ,
     EstablishTemporaryConnectionArg ,
     EventNotificationChargingArg ,
     EventReportBCSMArg ,
     FurnishChargingInformationArg ,
     InitialDPArg ,
     InitiateCallAttemptArg ,
     ManageTriggerDataArg ,
     ManageTriggerDataResultArg  ,
     MergeCallSegmentsArg ,
     MoveCallSegmentsArg ,
     MoveLegArg ,
     ReleaseCallArg ,
     ReportUTSIArg ,
     RequestCurrentStatusReportArg,
     RequestCurrentStatusReportResultArg, -- 3.6
     RequestEveryStatusChangeReportArg,
     RequestFirstStatusMatchReportArg,
     RequestNotificationChargingEventArg,
     RequestReportBCSMEventArg ,
     RequestReportUTSIArg ,
     ResetTimerArg ,
     SelectFacilityArg,
     SendChargingInformationArg ,
     SendSTUIArg ,
     ServiceFilteringResponseArg ,
     SetServiceProfileArg,
     SplitLegArg ,
     StatusReportArg,

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

```
     ScriptRunArg ,
     SpecializedResourceReportArg


FROM INCS3opsargs

;

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 ,
           cAArg          [10] CancelArg ,
           cIArg          [11] CollectInformationArg ,
           cOArg          [12] ConnectArg ,
           cTRArg           [13] ConnectToResourceArg,
           cWAArg           [14] ContinueWithArgumentArg ,
           cCSAArg           [15] CreateCallSegmentAssociationArg ,
           cCSARArg     [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,
           meCSArg           [30] MergeCallSegmentsArg,
           meCSRArg     [31] Empty,
           moCSArg           [32] MoveCallSegmentsArg,
           moCSRArg     [33] Empty,
           mLArg          [34] MoveLegArg,
           mLRArg           [35] Empty,
           rCArg          [36] ReleaseCallArg ,
           rUTSIArg     [37] ReportUTSIArg ,
           rNCEArg           [38] RequestNotificationChargingEventArg ,
           rRBEArg           [39] RequestReportBCSMEventArg ,
           rRUTSIArg     [40] RequestReportUTSIArg ,
           rTArg          [41] ResetTimerArg ,
           sCIArg           [42] SendChargingInformationArg ,
           sSTUIArg     [43] SendSTUIArg ,
           sFRArg           [44] ServiceFilteringResponseArg ,
           sLArg          [45] SplitLegArg ,
           sLRArg           [46] Empty ,
            -- SRF
           pAArg          [47] PlayAnnouncementArg ,
           mRArg          [48] MessageReceivedArg ,
           pCUIArg           [49] PromptAndCollectUserInformationArg ,
           pCUIRArg           [103] ReceivedInformationArg, -- Tag 103 3.6
           pRMArg           [50] PromptAndReceiveMessageArg ,
           pRMRArg           [100] MessageReceivedArg, --Tag 100 3.6
           rIArg          [51] ReceivedInformationArg ,
           sCArg          [52] ScriptCloseArg ,
           sEArg          [53] ScriptEventArg ,
           sIArg          [54] ScriptInformationArg ,
           scRArg           [55] ScriptRunArg ,
           sRRArg           [56] SpecializedResourceReportArg,
           -- ITU CS-3
```

```
          cRTDArg          [94] CreateOrRemoveTriggerDataArg,
          cRTDRArg     [93] CreateOrRemoveTriggerDataResultArg,
          cSRArg     [101] CancelStatusReportRequestArg, --Tag 101 3.6
          rCSArg          [80] RequestCurrentStatusReportArg,
          rCSRArg           [102] RequestCurrentStatusReportResultArg, --Tag 102 3.6
          rESArg          [79] RequestEveryStatusChangeReportArg,
          rESRArg              [104] Empty,
          rFSArg          [78] RequestFirstStatusMatchReportArg,
          rFSRArg              [105] Empty,
          sFArg          [75] SelectFacilityArg,
          sSPArg          [72] SetServiceProfileArg,
          sRPArg          [71] StatusReportArg
     }

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
```

ASN.1 Text INCS3datatypes

```
INCS3datatypes
-- The Definition of SSF - SCF  Data Types Follows

--IN-CS3-SSF-SCF-datatypes {itu-t(0) identified-organization(4) etsi(0) inDomain
(1) in-network(1) cs3(30) modules(0) in-cs3-ssf-scf-datatypes(6) version1(0)}


DEFINITIONS IMPLICIT TAGS ::=

BEGIN
--IMPORTS
--     common-classes,
--     common-datatypes,
--     ssf-scf-classes,
--     scf-srf-classes,
--     scf-srf-datatypes,
--     tc-Messages

--FROM IN-CS3-object-identifiers {itu-t(0) identified-organization(4) etsi(0) in
Domain(1) in-network(1) cs3(30) modules(0) in-cs3-object-identifiers(0) version1
(0)}

--     COMMON-BOUNDS
--FROM IN-CS3-Common-Classes common-classes

--     TRIGGER,
--     SCF-SSF-BOUNDS
--FROM IN-CS3-SSF-SCF-Classes ssf-scf-classes

--     SCF-SRF-BOUNDS
--FROM IN-CS3-SCF-SRF-Classes scf-srf-classes

--     Extensions{},
--     Integer4
--FROM IN-CS3-common-datatypes common-datatypes

--               InformationToSend {}

--FROM IN-CS3-scf-srf-datatypes scf-srf-datatypes


--     AddOnChargingInformation,
--     ChargingTariffInformation,
--     ChargingMessageType
--FROM Tariffing-DataTypes {itu-t(0) identified-organization etsi (0) 1296 versi
on2(3)}

--     ISDN-AddressString
--FROM MAP-CommonDataTypes {ccitt(0) identified-organization(4) etsi(0) mobileDo
main(0) gsm-Network(1) modules(3) map-CommonDataTypes(18) version6(6)}
--;
-- Moved to the top of this document because Tau 3.6 didn't like them at the end
!
-- The Definition of range of constants Follows
minAChBillingChargingLength     INTEGER  ::=     1 -- example value
maxAChBillingChargingLength     INTEGER  ::=     5 -- example value
minBackwardGVNSLength           INTEGER  ::=     1 -- example value
maxBackwardGVNSLength           INTEGER  ::=     5 -- example value
maxBearerCapabilityLength       INTEGER  ::=     5 -- example value
minCalledDirectoryNumberLength  INTEGER  ::=     1 -- example value
maxCalledDirectoryNumberLength  INTEGER  ::=     5 -- example value
minCalledPartyBCDNumberLength           INTEGER  ::=     1 -- example value
maxCalledPartyBCDNumberLength           INTEGER  ::=     5 -- example value
minCalledPartyNumberLength              INTEGER          ::=     0 -- example va
lue
maxCalledPartyNumberLength      INTEGER  ::=     16 -- example value
minCallingGeodeticLocationLength        INTEGER  ::=     1 -- example value
maxCallingGeodeticLocationLength        INTEGER  ::=      5 -- example value
minCallingPartyNumberLength     INTEGER  ::=     0 -- example value
maxCallingPartyNumberLength     INTEGER  ::=     16 -- example value
```

```
minCallResultcs1Length          INTEGER  ::=     1 -- example value
maxCallResultcs1Length          INTEGER  ::=     5 -- example value
maxCallReferenceLength          INTEGER  ::=     5   -- example value
minCarrierLength        INTEGER  ::=     3   -- example value
maxCarrierLength        INTEGER  ::=     10  -- example value
minChargedPartyNumberLength     INTEGER  ::=     1 -- example value
maxChargedPartyNumberLength             INTEGER  ::=     14 -- example value
maxCauseLength  INTEGER  ::=     4 -- example value
minCommunicationTariffNum       INTEGER  ::=     1 -- example value
maxCommunicationTariffNum               INTEGER  ::=     4 -- example value
maxCNInfoLength         INTEGER  ::=     5 -- example value
minDigitsLength         INTEGER  ::=     1 -- example value
maxDigitsLength         INTEGER  ::=      5 -- example value
minDisplayInformationLength     INTEGER  ::=      1 -- example value
maxDisplayInformationLength     INTEGER  ::=     5 -- example value
minEventSpecificInformationChargingLength       INTEGER  ::=     1 -- example va
lue
maxEventSpecificInformationChargingLength       INTEGER  ::=     5 -- example va
lue
minEventTypeChargingLength      INTEGER  ::=      1 -- example value
maxEventTypeChargingLength      INTEGER  ::=     5 -- example value
minFCIBCCcs1Length      INTEGER  ::=     1 -- example value
maxFCIBCCcs1Length      INTEGER  ::=     5 -- example value
minFCIBCCcs2Length      INTEGER  ::=     1 -- example value
maxFCIBCCcs2Length      INTEGER  ::=      5 -- example value
minFCIBillingChargingDataLength         INTEGER  ::=     1 -- example value
maxFCIBillingChargingDataLength         INTEGER  ::=     160 -- example value
minCamelFCIBillingChargingDataLength    INTEGER  ::=     1 -- example value
maxCamelFCIBillingChargingDataLength    INTEGER  ::=     5 -- example value
minFCIBillingChargingLength     INTEGER  ::=     5 -- example value
maxFCIBillingChargingLength     INTEGER  ::=     172 -- example value
minFormatDataLength             INTEGER  ::=     1 -- example value
maxFormatDataLength             INTEGER  ::=     5 -- example value
minForwardGVNSLength            INTEGER  ::=     1 -- example value
maxForwardGVNSLength            INTEGER  ::=     5 -- example value
minGenericNameLength            INTEGER  ::=     1 -- example value
maxGenericNameLength            INTEGER  ::=     5 -- example value
minGenericNumberLength          INTEGER  ::=     1 -- example value
maxGenericNumberLength  INTEGER  ::=     5 -- example value
maxGlobalCallReferenceLength    INTEGER  ::=     10  -- example value
maxInitialTimeInterval          INTEGER  ::=     5 -- example value
maxINServiceCompatibilityIndLength      INTEGER  ::=     5  -- example value
minIPAvailableLength    INTEGER  ::=     1 -- example value
maxIPAvailableLength    INTEGER  ::=     5 -- example value
minIPSSPCapabilitiesLength      INTEGER  ::=     1 -- example value
maxIPSSPCapabilitiesLength      INTEGER  ::=     5 -- example value
minISDNAccessRelatedInfoLength          INTEGER  ::=     1   -- example value
maxISDNAccessRelatedInfoLength          INTEGER  ::=     10  -- example value
minLocationNumberLength         INTEGER  ::=     1 -- example value
maxLocationNumberLength         INTEGER  ::=     5 -- example value
minMidCallControlInfoNum        INTEGER  ::=     1 -- example value
maxMidCallControlInfoNum        INTEGER  ::=     5 -- example value
minOriginalCalledPartyIDLength          INTEGER  ::=     1 -- example value
maxOriginalCalledPartyIDLength          INTEGER  ::=     5 -- example value
minReasonLength         INTEGER  ::=      1 -- example value
maxReasonLength         INTEGER  ::=      5 -- example value
minRedirectingPartyIDLength     INTEGER  ::=      1 -- example value
maxRedirectingPartyIDLength     INTEGER  ::=      5 -- example value
minRequestedUTSINum             INTEGER  ::=     1 -- example value
maxRequestedUTSINum             INTEGER  ::=      5 -- example value
minRouteListLength      INTEGER  ::=     1 -- example value
maxRouteListLength      INTEGER  ::=     5 -- example value
minRouteingNumberLength         INTEGER  ::=      1 -- example value
maxRouteingNumberLength         INTEGER  ::=      5 -- example value
minScfIDLength          INTEGER  ::=      1 -- example value
maxScfIDLength          INTEGER  ::=     5 -- example value
minSCIBillingChargingLength     INTEGER  ::=     1 -- example value
maxSCIBillingChargingLength     INTEGER  ::=     5 -- example value
minSDSSinformationLength        INTEGER  ::=     1   -- example value
maxSDSSinformationLength        INTEGER  ::=     5   -- example value
minServiceInteractionIndicatorsLength           INTEGER  ::=      1 -- example v
alue
maxServiceInteractionIndicatorsLength           INTEGER  ::=     5 -- example va
```

```
lue
minSFBillingChargingLength      INTEGER  ::=     1 -- example value
maxSFBillingChargingLength      INTEGER  ::=     5 -- example value
minSubTariffControlLen          INTEGER  ::=     1 -- example value
maxSubTariffControlLen          INTEGER  ::=     5 -- example value
minTimeAndTimezoneLength        INTEGER  ::=     1   -- example value
maxTimeAndTimezoneLength        INTEGER  ::=     8   -- example value
minTariffIndicatorsLen          INTEGER  ::=     1 -- example value
maxTariffIndicatorsLen          INTEGER  ::=     8 -- example value
minUSIInformationLength         INTEGER  ::=     1 -- example value
maxUSIInformationLength         INTEGER  ::=     5 -- example value
minUSIServiceIndicatorLength            INTEGER  ::=     1 -- example value
maxUSIServiceIndicatorLength            INTEGER  ::=     5 -- example value
numOfBCSMEvents         INTEGER  ::=     4 -- example value
numOfBCUSMEvents                INTEGER  ::=     4 -- example value
numOfChargingEvents             INTEGER  ::=     4 -- example value
numOfCSAs       INTEGER  ::=     50 -- example value
numOfCSs        INTEGER  ::=     10 -- example value
numOfGenericNumbers             INTEGER  ::=      2 -- example value
numOfINProfile          INTEGER  ::=     1 -- example value
numOfTriggers           INTEGER  ::=      5 -- example value
numOfInServiceCompatibilityIndLength    INTEGER  ::=      1  -- example value
numOfLegs               INTEGER  ::=     10  -- example value
--numOfMessageIDs       INTEGER  ::=     2  example value
maxAmount       INTEGER  ::=      2 -- example value
maxInitialUnitIncrement         INTEGER  ::=      2 -- example value
maxScalingFactor        INTEGER  ::=     2 -- example value
maxSegmentsPerDataInterval      INTEGER  ::=     5 -- example value
ub-nbCall       INTEGER  ::=     5 -- example value
numOfAddresses          INTEGER  ::=     5 -- example value

     minAttributesLength     INTEGER ::= 0 --example value
     maxAttributesLength     INTEGER ::= 5 --example value
     minMailBoxIDLength     INTEGER ::= 1 --example value
     maxMailBoxIDLength     INTEGER ::= 5 --example value
     minMessageContentLength     INTEGER ::= 0 --example value
     maxMessageContentLength     INTEGER ::= 50 --example value
     minReceivedInformationLength     INTEGER ::= 1 --example value
     maxReceivedInformationLength      INTEGER ::= 5 --example value
     maxRecordingTime     INTEGER ::= 5 --example value
     numOfMessageIDs     INTEGER ::= 5 --example value
     maxRecordedMessageUnits     INTEGER ::= 5 --example value
     maxVariableParts     INTEGER ::= 5 --example value

-- Note: Definition of the cause values (informative)
noIndication Cause ::= '00'H
originationAttemptDenied 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
routeFailure2 Cause ::= '0C'H
bPtyBusyUDUB Cause ::= '0D'H
bPtyBusyNDUB Cause ::= '0E'H
aPtyAbandon Cause ::= '0F'H
aPtyDisc Cause ::= '10'H
bPtyDisc Cause ::= '11'H
terminationAttemptDenied 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


-- DEFINITIONS OF TYPES WHICH ARE IMPORTED OR THE DEFINITION IS MISSING (Informa
tive information).
TriggerID ::= INTEGER
TriggerData ::= SEQUENCE OF TriggerID
Extensions ::= SEQUENCE SIZE (1..numOfExtensions) OF ExtensionField
ExtensionField ::= OCTET STRING

ExportLegsType ::= SEQUENCE SIZE (1..numOfLegs) OF SEQUENCE {
  sourceLeg [0] DTLegID,
  newLeg    [1] DTLegID
}

Integer4 ::= INTEGER  -- (0..2147483647) Bounds removed for Tau 3.6
InvokeID ::= INTEGER (-128..127)


-- From MAP Common Datatypes
AddressString ::= OCTET STRING (SIZE (1..maxAddressLength))
     -- This type is used to represent a number for addressing
     -- purposes. It is composed of
     --     a)     one octet for nature of address, and numbering plan
     --          indicator.
     --     b)     digits of an address encoded as TBCD-String.

     -- a)     The first octet includes a one bit extension indicator, a
     --          3 bits nature of address indicator and a 4 bits numbering
     --          plan indicator, encoded as follows:

     -- bit 8: 1  (no extension)

     -- bits 765: nature of address indicator
     --     000  unknown
     --     001  international number
     --     010  national significant number
     --     011  network specific number
     --     100  subscriber number
     --     101  reserved
     --     110  abbreviated number
     --     111  reserved for extension

     -- bits 4321: numbering plan indicator
     --     0000  unknown
     --     0001  ISDN/Telephony Numbering Plan (Rec CCITT E.164)
     --     0010  spare
     --     0011  data numbering plan (CCITT Rec X.121)
     --     0100  telex numbering plan (CCITT Rec F.69)
     --     0101  spare
     --     0110  land mobile numbering plan (CCITT Rec E.212)
     --     0111  spare
     --     1000  national numbering plan
     --     1001  private numbering plan
     --     1111  reserved for extension

     --     all other values are reserved.

     -- b)     The following octets representing digits of an address
     --          encoded as a TBCD-STRING.

maxAddressLength  INTEGER ::= 20

ISDN-AddressString ::=
               AddressString (SIZE (1..maxISDN-AddressLength))
     -- This type is used to represent ISDN numbers.

maxISDN-AddressLength  INTEGER ::= 9
```

```
TBCD-STRING ::= OCTET STRING
     -- This type (Telephony Binary Coded Decimal String) is used to
     -- represent several digits from 0 through 9, *, #, a, b, c, two
     -- digits per octet, each digit encoded 0000 to 1001 (0 to 9),
     -- 1010 (*), 1011 (#), 1100 (a), 1101 (b) or 1110 (c); 1111 used
     -- as filler when there is an odd number of digits.

     -- bits 8765 of octet n encoding digit 2n
     -- bits 4321 of octet n encoding digit 2(n-1) +1

IMSI ::= TBCD-STRING (SIZE (3..8))
     -- digits of MCC, MNC, MSIN are concatenated in this order.


-- From Remote-Operations-Information-Objects
Code ::= CHOICE
{
     local     INTEGER,
     global     OBJECT IDENTIFIER
}



--Types which are defined in the SCF-SRF part
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,
     interruptableAnnInd     [8] BOOLEAN     DEFAULT TRUE,
     voiceInformation     [9] BOOLEAN     DEFAULT FALSE,
     voiceBack     [10] BOOLEAN     DEFAULT FALSE,
     detectModem     [11] BOOLEAN     DEFAULT FALSE
}
-- The use of voiceBack is network operator specific.
-- The endOfReplyDigit, cancelDigit, and startDigit parameters have been designa
ted 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 depende
nt.
-- firstDigitTimeOut and interDigitTimeOut are measured in seconds.

CollectedInfo ::= CHOICE {
     collectedDigits     [0] CollectedDigits,
     iA5Information     [1] BOOLEAN,
     detectModem     [2] BOOLEAN
     }


ElementaryMessageID ::= Integer4

ErrorTreatment ::= ENUMERATED {
          reportErrorToScf(0),
          help(1),
          repeatPrompt(2)
          }
-- reportErrorToScf means returning the "ImproperCallerResponse" error in the ev
ent of an error
-- condition during collection of user info.

GapOnResource ::= Code


InbandInfo   ::= SEQUENCE {
     messageID     [0] MessageID ,
     numberOfRepetitions     [1] INTEGER (1..127)     OPTIONAL,
     duration     [2] INTEGER (0..32767)     OPTIONAL,
     interval     [3] INTEGER (0.. 32767)     OPTIONAL,
     preferredLanguage     [4] Language     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 ,
     sDSSinformation      [3] SDSSinformation
     }

Language ::= PrintableString (SIZE (3) ) -- ISO 639 codes only;



MailBoxID  ::= OCTET STRING (SIZE(
                    minMailBoxIDLength..maxMailBoxIDLength))

Media ::= ENUMERATED {
     voiceMail (0),
     faxGroup3 (1),
     faxGroup4 (2)
     }

MessageID  ::= CHOICE {
     elementaryMessageID     [0] Integer4,
     text     [1] SEQUENCE {
               messageContent     [0] IA5String (SIZE (minMessageContentLength..maxMessage
tLength)),
               attributes     [1] OCTET STRING (SIZE (minAttributesLength..maxAttributesLe

                                   OPTIONAL
                },
     elementaryMessageIDs     [29] SEQUENCE SIZE (1..numOfMessageIDs) OF Integer4,
     variableMessage     [30] SEQUENCE {
                elementaryMessageID[0] Integer4,
                variableParts [1] SEQUENCE SIZE (1..maxVariableParts)  OF VariablePart
                }
}
-- OPTIONAL denotes network operator specific use.


ReceivedStatus ::=ENUMERATED {
     messageComplete (0),
     messageInterrupted (1),
     messageTimeOut (2)
     }

RecordedMessageID ::= Integer4

SRFGapCriteria  ::= CHOICE {
     iPAddressValue     [1] Digits ,
     gapOnResource     [2] GapOnResource,
```

```
     iPAddressAndresource     [3] SEQUENCE {
               iPAddressValue     [1] Digits ,
               gapOnResource     [2] GapOnResource
               }
     }

--Tone ::= SEQUENCE {
--     toneID     [0] Integer4,
--     duration     [1] Integer4     OPTIONAL,
--     ...
--     }
-- The duration specifies the length of the tone in seconds, value 0 indicates i
nfinite duration.

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..4)),
     -- size 3: YYMMDD, BCD coded.
     -- size 4: YYYYMMDD, 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.
--  The most significant hours digit occupies bits 0-3 of the first octet, and t
he least
--  significant digit occupies bits 4-7 of the first octet. The most significant
 minutes digit
--  occupies bits 0-3 of the second octet, and the least significant digit occup
ies bits 4-7
--  of the second octet.
-- For example, time = 12:15 would be encoded as:
--     Bits                         HGFE           DCBA
--      leading octet     2     1
--               5     1
-- OPTION 1 (date: Size 3):
-- date = 1993 September 30th would be encoded as:
--     Bits                         HGFE          DCBA
--     leading octet      3     9
--               9     0
--               0     3
-- For a system operating when or after this Recommendation  is released, the 2
digit value
-- representing a Year shall be interpreted as follows :
--        - If the two-digits value is 00 through 49 inclusive, it shall be inte
rpreted as representing
--          year  2000  through 2049.
--       -  If the two-digits value is 50 through 99 inclusive, it shall be inte
rpreted as representing
--         year 1950 through 1999.
-- OPTION 2 (date: Size 4):
--  Date is BCD encoded. The year digit indicating millenium occupies bits 0-3 o
f the first octet,
--  and the year digit indicating century occupies bits 4-7 of the first octet.
The year digit
--  indicating decade occupies bits 0-3 of the second octet, whilst the digit in
dicating the year
--  within the decade occupies bits 4-7 of the second octet.
--  The most significant month digit occupies bits 0-3 of the third octet, and t
he least
--  significant month digit occupies bits 4-7 of the third octet. The most signi
ficant day digit
--  occupies bits 0-3 of the fourth octet, and the least significant day digit o
ccupies bits 4-7
--  of the fourth octet.
--  Price is BCD encoded. The digit indicating hundreds of thousands occupies bi
ts 0-3 of the
--  first octet, and the digit indicating tens of thousands occupies bits 4-7 of
 the first octet.
--  The digit indicating thousands occupies bits 0-3 of the second octet, whilst
 the digit
--  indicating hundreds occupies bits 4-7 of the second octet. The digit indicat
```

```
ing tens occupies
--  bits 0-3 of the third octet, and the digit indicating 0 to 9 occupies bits 4
-7 of the third
--  octet. The tenths digit occupies bits 0-3 of the fourth octet, and the hundr
edths digit
--  occupies bits 4-7 of the fourth octet.
--
--  For the encoding of digits in an octet, refer to the timeAndtimezone paramet
er




--CHARGING RELATED TYPES IMPORTED FROM ISUP CHARGING SPEC.(EN 301 296)

--ChargingMessageType Type
ChargingMessageType ::= CHOICE {
     crgt          [0] ChargingTariffInformation ,
     aocrg          [1] AddOnChargingInformation ,
     crga          [2] ChargingAcknowledgementInformation,
     startt          [3] StartCharging ,
     stopp          [4] StopCharging
     }


--ChargingAcknowledgementInformation Type
ChargingAcknowledgementInformation ::= SEQUENCE {
     acknowledgementIndicators     [0] BIT STRING { accepted(0) } (SIZE (minAcknowledgem
entIndicatorsLen..
                              maxAcknowledgementIndicatorsLen)),
     -- extensions               [1] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField O
L,
     originationIdentification     [2] ChargingReferenceIdentification,
     destinationIdentification     [3] ChargingReferenceIdentification
     }
--Coding of "accepted":
-- 0 : not accepted.
-- 1 : accepted.
--The "extensions" is a SEQUENCE for possible extensions.

--ChargingControlIndicators Type
ChargingControlIndicators ::= BIT STRING{
          subscriberCharge(0),
          immediateChangeOfActuallyAppliedTariff(1),
          delayUntilStart(2) }
     (SIZE(minChargingControlIndicatorsLen..maxChargingControlIndicatorsLen))
--Coding of "subscriberCharge":
-- 0 - advice-of-charge : charging information only to be used by the adviceofch
arge service.
-- 1 - subscriber-charge : charging information to be used by the subscriberchar
ging program.
-- Coding of "immediateChangeOfActuallyAppliedTariff":
-- 0 - immediate tariff change without restart
-- 1 - immediate tariff change with restart
-- It is only used to change the actually applied tariff.
--Coding of ˜delayUntilStart˜:
-- 0 - start tariffing, if it is not already started, without waiting for the 's
tart' signal
-- 1 ˘ delay start of tariffing up to the receipt of the 'start' signal

-- AddOnChargingInformation Type
AddOnChargingInformation ::= SEQUENCE {
     chargingControlIndicators     [0] ChargingControlIndicators ,
     addOnCharge               [1] CHOICE {
               addOnChargeCurrency          [0] CurrencyFactorScale ,
               addOnChargePulse          [1] PulseUnits
               },
     --extensions               [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField OP
,
     originationIdentification     [3] ChargingReferenceIdentification,
     destinationIdentification     [4] ChargingReferenceIdentification OPTIONAL,
     currency               [5] Currency
     }
```

```
-- This message is used to add an amount of charge for the call and does not alt
er the current tariff.
-- The destinationIdentification is not available in an initial AddOnChargingInf
ormation, in all subsequent ones it is included, see
-- "Handling of Identifiers".
-- In the message the
-- add-on charge has either the pulse or currency format.

-- ChargingTariffInformation Type
ChargingTariffInformation ::= SEQUENCE {
          chargingControlIndicators     [0] ChargingControlIndicators,
          chargingTariff               [1] CHOICE {
                    tariffCurrency          [0] TariffCurrency,
                    tariffPulse          [1] TariffPulse
                    },
          --extensions               [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionFie
L,
          originationIdentification     [3] ChargingReferenceIdentification,
          destinationIdentification     [4] ChargingReferenceIdentification OPTIONAL,
          currency               [5] Currency
          }
--This message is used
-- to transfer explicit tariff data to the originating subscriber exchange and t
he charge registration exchange during call
-- set-up and also in the active phase of a call.
-- The destinationIdentification is not available in an initial ChargingTariffIn
formation, in all subsequent ones it is included, see
-- "Handling of Identifiers".


-- Criticality Type
CriticalityType ::= ENUMERATED {
          ignore(0),
          abort(1)
          }





-- ExtensionField Type
--ExtensionField ::= SEQUENCE {
          --type EXTENSION.&id ({SupportedExtensions}),
-- shall identify the value of an EXTENSION type
          --criticality CriticalityType DEFAULT ignore,
          --value [1] EXTENSION.&ExtensionType
                    --({SupportedExtensions}{@type})
          --}
-- This parameter indicates an extension of an argument data type. Its contents
is network operator specific.


-- StartCharging Type
StartCharging ::= SEQUENCE {
          networkOperators     [0] SEQUENCE SIZE (1..maxNetworkOperators) OF NetworkIdenti
cation OPTIONAL,
          -- extensions          [1] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionField O
L,
          originationIdentification [2] ChargingReferenceIdentification
          }
--This message is used
-- to start charge generation/registration for one that belongs to one network o
perator or
-- to start charge generation/registration for more tariffs that belong to sever
al network
-- operators.
-- The corresponding network operator(s) are listed within sub-parameter ~networ
kOperators~.
-- The sub-parameter ~networkOperators~ is mandatory in configurations of multip
le charge
-- determination points.
-- If sub-parameter ~networkOperators~ is not present in a configuration with on
ly one charge
```

```
-- determination point, the corresponding - tariff is started.
-- The ^extensions~ is a SEQUENCE for possible extensions.

-- StopCharging Type
StopCharging ::= SEQUENCE {
          stopIndicators               [0] BIT STRING {
                    callAttemptChargesApplicable     (0) } (SIZE(minStopIndicatorsLen.. ma
atorsLen)),
          networkOperators          [1] SEQUENCE SIZE (1..maxNetworkOperators) OF NetworkI
ication OPTIONAL,
          --extensions               [2] SEQUENCE SIZE(1..numOfExtensions) OF ExtensionFie
L,
          originationIdentification     [3] ChargingReferenceIdentification
          }
--This message is used to stop charge generation/registration
-- for one that belongs to one network operator or
-- to stop charge generation/registration for more tariffs that belong to severa
l network operators.
-- The corresponding network operator(s) are listed within sub-parameter ~networ
kOperators~.
-- The sub-parameter ~networkOperators~ is mandatory in configurations of multip
le charge
-- determination points.
-- If sub-parameter ~networkOperators~ is not present in a configuration with on
ly one charge
-- determination point, the corresponding - tariff is stopped.
-- Coding of ~callAttemptChargesApplicable~:
-- 0 - stop tariffing, call attempt charges are not applicable
-- 1 - stop tariffing, call attempt charges are applicable
-- The ^extensions~ is a SEQUENCE for possible extensions.




minAcknowledgementIndicatorsLen INTEGER ::= 1
maxAcknowledgementIndicatorsLen INTEGER ::= 8
minChargingControlIndicatorsLen INTEGER ::= 1
maxChargingControlIndicatorsLen INTEGER ::= 8
maxNetworkOperators INTEGER ::= 6
minStopIndicatorsLen INTEGER ::= 1
maxStopIndicatorsLen INTEGER ::= 8
noCharge INTEGER ::= 0
noScale INTEGER ::= 0
numOfExtensions INTEGER ::= 1--network specific

-- ChargingReferenceIdentification Type
ChargingReferenceIdentification ::= SEQUENCE {
          networkIdentification          [0] NetworkIdentification,
          referenceID               [1] ReferenceID
          }

-- NetworkIdentification Type
NetworkIdentification ::= OBJECT IDENTIFIER
-- Following structure of the networkIdentification value shall be used:
-- {itu-t (0) administration (2) <national regulation authority> (x) network (y)
 node identification (z)}
-- The value for x is the value of the national regulation authority, the value
for y is under the control of the national
-- regulation authority concerned, the value for z is under the control of the n
etwork concerned.

-- ReferenceID Type
ReferenceID ::= INTEGER --(0..4294967295)
-- maximum value 2 32 -1


-- End of added definitions.
-- *****


-- The following three definitions are local short-hand notation for convenience
.
```

```
--B1 ::= COMMON-BOUNDS           defined in Part 1.1
--B2' ::= SCF-SSF-BOUNDS           defined herein.
--B3 ::= SCF-SRF-BOUNDS           defined in Part 1.3.



AChBillingChargingCharacteristics   ::= OCTET STRING (SIZE
          (minAChBillingChargingLength..maxAChBillingChargingLength))
-- The AChBillingChargingCharacteristics parameter specifies charging related in
formation.
-- Its content is network operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.
-- CAMEL:
-- AChBillingChargingCharacteristics {PARAMETERS-BOUND : bound} ::= OCTET STRING
 (SIZE
--     (bound.&minAChBillingChargingLength..bound.&maxAChBillingChargingLength))
--     (CONSTRAINED BY {
-- shall be the result of the BER-encoded value of the type --
--     CAMEL-AchBillingChargingCharacteristics })
-- The AChBillingChargingCharacteristics parameter specifies the charging relate
d information
-- to be provided by the gsmSSF and the conditions on which this information has
 to be reported
-- back to the gsmSCF with the ApplyChargingReport operation. The value of the
-- AchBillingChargingCharacteristics of type OCTET STRING carries a value of
-- the ASN.1 data type: CAMEL-AchBillingChargingCharacteristics.
-- The normal encoding rules are used to encode this value.
-- The violation of the UserDefinedConstraint shall be handled as an ASN.1 synta
x error.


AChChargingAddress          ::= CHOICE {
               legID               [2]  DTLegID,
               srfCallSegment     [50] DTCallSegmentID ,
               bNCF               [51] DTLegID
}

ActionIndicator  ::= ENUMERATED {
     activate           (1) ,
     deactivate          (2) ,
     retrieve          (3)
     }
-- indicates the action to be performed by the ManageTriggerData operation.


ActionOnProfile ::= ENUMERATED {
     activate          (0),
     deactivate          (1)
     }
-- indicates the action to be performed by the SetServiceProfile operation.

ActionPerformed  ::= ENUMERATED {
     activated          (1) ,
     deactivated          (2) ,
     alreadyActive     (3) ,
     alreadyInactive     (4) ,
     isActive          (5) ,
     isInactive          (6),
     tDPunknown          (7)
     }
-- indicates the result of the operation ManageTriggerData

AdditionalCallingPartyNumber   ::= Digits
-- Indicates the Additional Calling Party Number.  Refer to ITU-T Recommendation
 Q.763
-- Generic Number for encoding.
```

```
AddOnCharge          ::= CHOICE {
     addOnChargeCurrency     [0] CurrencyFactorScale ,
     addOnChargePulse     [1] PulseUnits
     }

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.
-- Only applies if SSF is the terminating local exchange for the subscriber.
-- Refer to the ITU-T Recommendation Q.931  Signal parameter
-- respective 3G TS 29.002 for encoding.

AOCBeforeAnswer                              ::= SEQUENCE {
     aOCInitial                                   [0] CAI-GSM0224,
     aOCSubsequent                               [1] AOCSubsequent     OPTIONAL
     }
-- support for CAMEL

AOCSubsequent                              ::= SEQUENCE {
     cAI-GSM0224                                   [0] CAI-GSM0224     ,
     tariffSwitchInterval                    [1] INTEGER (1..86400)    OPTIONAL
     }
--   tariffSwitchInterval is measured in 1 second units
-- support for CAMEL

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.
-- Refer to ITU-T Recommendation Q.763 Generic Number for encoding.

BackwardGVNS   ::= OCTET STRING (SIZE(
minBackwardGVNSLength..maxBackwardGVNSLength))
-- Indicates the GVNS Backward information. Refer to Q.735 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

     holdTreatmentIndicator                    [3] OCTET STRING (SIZE(1))     OPTIONAL,
     -- acceptHoldRequest     'xxxx xx01'B
     -- rejectHoldRequest     'xxxx xx10'B
     -- network default is accept hold request

     ectTreatmentIndicator                    [4]     OCTET STRING (SIZE(1))     OPTIONAL
     -- acceptEctRequest     'xxxx xx01'B
     -- rejectEctRequest     'xxxx xx10'B
     -- network default is accept ect request
     }

BasicGapCriteria   ::= 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 encoding of the digits, refer to ITU-T Recommendation Q.763 Generic Numbe
r.


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

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 narrowband bearerCapability, either
-- DSS 1 (ES 300 403-1) or the ISUP User Service Information (ITU-T Recommendati
on Q.763)
-- encoding can be used. Refer
-- to the ITU-T Recommendation Q.763 Transmission Medium Requirement parameter f
or tmr encoding.


BothwayThroughConnectionInd ::= ENUMERATED {
     bothwayPathRequired          (0),
     bothwayPathNotRequired     (1)
     }
-- The default is bothwayPathRequired.
CAI-GSM0224                                       ::= SEQUENCE {
     e1                                                [0] INTEGER (0..8191)
     e2                                                [1] INTEGER (0..8191)
     e3                                                [2] INTEGER (0..8191)
     e4                                                [3] INTEGER (0..8191)
     e5                                                [4] INTEGER (0..8191)
     e6                                                [5] INTEGER (0..8191)
     e7                                                [6] INTEGER (0..8191)
     }
--  Support for CAMEL
--  Indicates Charge Advice Information to the Mobile Station. For information r
egarding
--  parameter usage, refer to 3G TS 22.024.

CalledDirectoryNumber   ::= OCTET STRING (SIZE
                              (minCalledDirectoryNumberLength..
                         maxCalledDirectoryNumberLength))
-- Indicates the Called Directory Number.
-- Refer to ITU-T Recommendation Q.763  'Called Directory Number' for encoding.


CalledPartyBCDNumber   ::= OCTET STRING (SIZE
          (minCalledPartyBCDNumberLength .. maxCalledPartyBCDNumberLength))
-- Indicates the Called Party Number, including service selection information.
-- Refer to 3G TS 24.008  for encoding.
-- This data type carries only the "type of number", "numbering plan identificat
ion" and
-- "number digit" fields defined in 3G TS 24.008;
-- 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 ITU-T Recommendation Q.763 for en
coding.
```

```
CallingGeodeticLocation  ::= OCTET STRING ( SIZE (minCallingGeodeticLocationLeng
th..maxCallingGeodeticLocationLength))
-- The coding of this parameter is based on the appropriate mapping with the
-- ISUP parameter Calling Geodetic Location.
-- Refer to Q.763 for encoding.

CallingPartyBusinessGroupID ::= OCTET STRING
-- Indicates the business group of the calling party. The value of this octet st
ring is network
-- operator specific.
-- Its content is network operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this parameter
-- (after the first tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

CallingPartyNumber   ::= OCTET STRING (SIZE (
                            minCallingPartyNumberLength..
                                 maxCallingPartyNumberLength))
-- Indicates the Calling Party Number.  Refer to Q.763 for encoding.


CallingPartysCategory ::= OCTET STRING (SIZE(1))
-- Indicates the type of calling party (e.g. operator, payphone, ordinary subscr
iber).
-- Refer to Q.763 for encoding

CallReference ::= OCTET STRING (SIZE(1..maxCallReferenceLength))
-- The coding of this parameter is network specific.
-- A possible coding is the Q.763 call reference
-- but other encoding schemes are possible.

CAMEL-AChBillingChargingCharacteristics  ::= CHOICE {
     timeDurationCharging                        [0] SEQUENCE {
               maxCallPeriodDuration                   [0] INTEGER (1..864000),
               releaseIfdurationExceeded               [1] BOOLEAN DEFAULT FALSE,
               tariffSwitchInterval                    [2] INTEGER (1..86400)
              tone                                       [3] BOOLEAN DEFAULT FALSE,
              extensions                                  [4] Extensions      OPTIONAL
          }
     }
--  tariffSwitchInterval is measured in 1 second units.
--  maxCallPeriodDuration is measured in100 millisecond units

CAMEL-CallResult           ::= CHOICE {
     timeDurationChargingResult                    [0] SEQUENCE {
          partyToCharge                                  [0] ReceivingSideID,
          timeInformation                                  [1] TimeInformation,
          callActive                                       [2] BOOLEAN
          callReleasedAtTcpExpiry                        [3] NULL
          extensions                                       [4] Extensions           OPTIO
          }
     }

CAMEL-FCIBillingChargingCharacteristics  ::= CHOICE{
     fCIBCCCAMELsequence1               [0] SEQUENCE {
          freeFormatData                        [0] OCTET STRING (SIZE ( minCamelFCIBilli
th..                                                                          maxCamelFCI
          partyToCharge                        [1] SendingSideID
                                                      DEFAULT sendingSideID : dtleg1
          }
     }


CAMEL-SCIBillingChargingCharacteristics ::= CHOICE {
      aOCBeforeAnswer                                  [0] AOCBeforeAnswer,
      aOCAfterAnswer                                  [1] AOCSubsequent
     }
```

```
GlobalCallReference ::= OCTET STRING (SIZE(1..maxGlobalCallReferenceLength))
-- The coding of this parameter is defined in Annex D.


CallResult            ::= CHOICE {
     callResult     OCTET STRING (SIZE (minCallResultcs1Length ..maxCallResultcs1Length)
),
-- This parameter provides the SCF with the charging related information previou
sly requested
-- using the ApplyCharging operation.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter
-- (after the first tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.
     callSupervisionResult     [51]     SEQUENCE {
          callResult                          [0] OCTET STRING (SIZE (minCallResultcs1Lengt
                                                   maxCallResultcs1Length))     OPTIONAL,
          reportConditionInformation     [1] ReportConditionInformation,
          aChChargingAddress               [2] AChChargingAddress      DEFAULT legID:recei
eg1,
          supervisionResult                [3] CHOICE {
                              cost               [1] UsedCost,
                              time               [2] UsedTime,
                              pulses               [3] UsedPulses
               },
          minLimitNeeded                     [4] CHOICE {
                              cost               [1] NeededCost,
                              pulses               [2] NeededPulses
               },
          extensions                          [5] Extensions   OPTIONAL
          }
     }

-- This parameter provides the SCF with the charging related information previou
sly requested
-- using the ApplyCharging operation.
-- Examples of charging related information to be provided by the SSF may be: bu
lk 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 re
ported may be:
-- threshold value reached, timer expiration, tariff change, end of connection c
onfiguration, etc.
-- CAMEL:
-- CallResult {PARAMETERS-BOUND : bound} ::= OCTET STRING (SIZE (minCallResultLe
ngth ..
--     maxCallResultLength))
--     (CONSTRAINED BY {
-- shall be the result of the BER-encoded value of type --
--     CAMEL-CallResult })
-- The violation of the UserDefinedConstraint shall be handled as an ASN.1 synta
x error.
-- This parameter provides the gsmSCF with the charging related information prev
iously requested
-- using the ApplyCharging operation. This shall include the partyToCharge param
eter as
-- received in the related ApplyCharging operation to correlate the result to th
e request
-- The value of the CallResult of type OCTET STRING carries
-- a value of the ASN.1 data type: CAMEL-CallResult.
-- The normal encoding rules are used to encode this value.

DTCallSegmentID   ::= INTEGER (1..numOfCSs)

CallSupervision          ::= SEQUENCE {
     supervisionMethod               [1] SupervisionMethod,
     warningBeforeLimitReached     [2] WarningBeforeLimitReached               OPTIONAL,
     releaseWhenLimitReached          [3] ReleaseWhenLimitReached               OPTIONAL,
     reportConditions               [4] ReportConditions                     OPTIONAL
     }
```

```
Carrier  ::= OCTET STRING (SIZE (minCarrierLength..
                       maxCarrierLength))

-- Contains the carrier selection field (first octet) followed by either  Carrie
r ID information
-- (option 1 (e.g. North America (na)) ) , or the Transit
-- Network selection information (option 2 (e.g. Europa)), depending on the netw
ork
-- in which the switch is located..

-- In both cases, the Carrier selection is one octet and is encoded as:
-- 00000000          No indication
-- 00000001          Selected carrier identification code (CIC) pre subscribed and not
--                    input by calling party
-- 00000010          Selected carrier identification code (CIC) pre subscribed and input
 by calling party
-- 00000011          Selected carrier identification code (CIC) pre subscribed, no indic
ation of
--                    whether input by calling party (undetermined)
-- 00000100      Selected carrier identification code (CIC) not pre subscribed and i
nput by
--                    calling party
-- 00000101
--      to               Spare
-- 11111110
-- 11111111          Reserved

-- Refer to ITU-T Recommendation Q.763 for encoding of Transit Network Selection
.

-- Refer to ANSI ISUP T.113 for encoding of na carrier ID information


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.

CCSS::=BOOLEAN
-- Used by the SSF to indicate CCSS (Call Completion on Service Set-up) if set t
o "True"
-- to the SCF, i.e.
-- that the current call is due a special procedure  (CCBS or CCNR).
-- The value TRUE corresponds with the ITU-T Recommendation Q.763 CCSS parameter

-- "CCSS call indicator" value 1 (CCSS call).
-- The value FALSE corresponds with the ITU-T Recommendation Q.763 CCSS paramete
r
-- "CCSS call indicator" value 0 (no indication).

CGEncountered ::= ENUMERATED {
     noCGencountered          (0),
     manualCGencountered     (1),
     sCPOverload               (2)
     }
-- Indicates the type of automatic call gapping encountered, if any.

ChargedParty                          ::= ENUMERATED {
     callingPartyNumber     (0),
     calledINNumber          (1),
     connectedNumber          (2),
     translatedNumber     (3),
     specialINNumber          (4)
     }
-- The TranslatedNumber is on the ISUP named CalledPartyNumber
-- (e.g. translated number after Connect operation).

ChargedPartyNumber   ::= OCTET STRING (SIZE(minChargedPartyNumberLength..
                                 maxChargedPartyNumberLength) )
-- Indicates the number that identifies the entity or party to be charged for th
e call.
```

```
-- The coding of this parameter is network specific.
-- A possible coding is the Q.763 called party number
-- but other encoding schemes are possible.

ChargeNumber   ::= LocationNumber
-- This parameter is only used for CAMEL. It uniquely identifies the chargeable
number
-- of a call sent into a North American
-- long distance carrier. It transports the ChargeNumber Parameter Field
-- as defined in ANSI ISUP T1.113. This provides 1 octet for the nature of addre
ss indicator field,
-- plus 1 octet for a numbering plan field, plus up to 5 octets for the address
signal
-- (up to 10  digits). It is used in the context of CAMEL.
--     The Charge Number in ANSI T1.113 normally contains a 10 digit national number
 within the North
--     American Numbering Plan (NANP); longer (e.g. international) charge numbers ar
e not supported in
--     T1.113-
-- Information indicating the chargeable number for the call and consisting of
the
-- odd/even indicator, nature of address indicator, numbering plan indicator,
-- and address signals.
-- Uses the LocationNumber format for encoding which is based on the Q.763 Locat
ion Number format

--ChargeUnitTimeIntervalType
ChargeUnitTimeInterval ::= OCTET STRING(SIZE(2))
--The Charge Unit Time Interval is binary coded and has the value range from 0 t
o 35997.
--It begins with 200 milliseconds and then in steps of 50 milliseconds.
--the LSB it is the least significant bit of the first octet
-- the MSB it is the most significant bit of the last octet
-- the coding of the Charge Unit Time Interval is the following:
--0 : no periodic metering
--1 : 200msec
--2 : 250msec
--..
--35997 : 30minutes
--All other values are spare.


ChargingAddress                    ::= CHOICE {
     leg                              [2] DTLegID,
     srfCallSegment                [50] DTCallSegmentID
     }

ChargingControlType                          ::= SEQUENCE {
     tariffInfoFromSuccExchangeSCI           [0]          TariffInfoFromSuccExchangeSCI,
     sSFdetermination                           [1]          SSFdetermination -- DEFAULT no
     }

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 mo
de and LedID

ChargingTariff               ::= CHOICE {
                    tariffCurrency          [0] TariffCurrency ,
                    tariffPulse               [1] TariffPulse
     }

CNInfo ::= OCTET STRING (SIZE(1.. maxCNInfoLength))
-- Refer to ITU-T Recommendation Q.765.1 NNI specific information to be transpor
ted in the
-- Application Transport Parameter (APP).

-- The NNI specific information for the VPN application is carried within the AP
P.
```

```
-- Only the CNID (Corporate Telecommunications Network Identifier) indicator is
significant
-- for INAP (bits 5 and 6).
-- The CNID indicator is followed by optionally (if included) the CNID length an
d CNID value.
-- CommunicationChargeCurrency Type
CommunicationChargeCurrency  ::= SEQUENCE {
               currencyFactorScale      [0] CurrencyFactorScale ,
               tariffDuration               [1] TariffDuration ,
               subTariffControl           [2] SubTariffControl
               }

-- CommunicationChargePulse Type
CommunicationChargePulse ::= SEQUENCE {
               pulseUnits                     [0] PulseUnits ,
               chargeUnitTimeInterval      [1] ChargeUnitTimeInterval ,
               tariffDuration                [2] TariffDuration
               }

ConnectedNumberTreatmentInd ::= ENUMERATED {
               noINImpact                                (0),
               presentationRestricted                (1),
               presentCalledINNumber                (2),
               presentCalledINNumberRestricted     (3)
               }
-- The default is presentCalledINNumber.

ControlType ::= ENUMERATED {
     sCPOverloaded          (0),
     manuallyInitiated     (1),
     destinationOverload     (2)
     }

CompoundCriteria      ::= SEQUENCE {
          basicGapCriteria            [0] BasicGapCriteria ,
          scfID                          [1] ScfID  OPTIONAL
                                       }
-- If scfID parameter is not available the call gapping is not dedicated to a sp
ecific SCF.

CorrelationID  ::= Digits
-- used by SCF for correlation with a previous operation.
-- refer to Q.763 Generic Digits for encoding from SCF to SSF.
-- refer to Q.763 Generic Number for encoding from SRF/SSF to SCF.


CounterAndValue ::= SEQUENCE {
          counterID          [0] CounterID,
          counterValue     [1] Integer4
          }

CounterID ::= INTEGER (0..99)
-- Indicates the counters to be incremented.
-- The counterIDs are addressed by using the last digits of the dialed number.

CountersValue ::= SEQUENCE SIZE(0..numOfCounters) OF CounterAndValue

CreateOrRemoveIndicator ::= ENUMERATED {
               crreate      (0),
               remove      (1)
               }

-- Indicates the creation or removal of a TDP-R.

CSAID  ::= INTEGER (1..numOfCSAs)
-- Indicates the SSF CSA identifier


-- Currency Type

Currency ::= ENUMERATED {
     noIndication          (0),
     australianDollar     (1),
```

```
     austrianSchilling     (2),
     belgianFranc          (3),
     britishPound          (4),
     czechKoruna               (5),
     danishKrone               (6),
     dutchGuilder          (7),
     euro                    (8),
     finnishMarkka          (9),
     frenchFranc               (10),
     germanMark               (11),
     greekDrachma          (12),
     hungarianForint          (13),
     irishPunt               (14),
     italianLira               (15),
     japaneseYen               (16),
     luxembourgian-Franc     (17),
     norwegianKrone          (18),
     polishZloty               (19),
     portugeseEscudo          (20),
     russianRouble          (21),
     slovakKoruna          (22),
     spanishPeseta          (23),
     swedishKrone          (24),
     swissFranc               (25),
     turkishLira               (26),
     uSDollar               (27)}

-- CurrencyFactor Type

CurrencyFactor ::= INTEGER (0..999999)

-- Value 0 indicates "no charge".

-- CurrencyFactorScale Type

CurrencyFactorScale ::= SEQUENCE {
     currencyFactor     [0] CurrencyFactor DEFAULT noCharge     ,
     currencyScale     [1] CurrencyScale DEFAULT noScale
     }
-- The charge amount is indicated by the currency factor multiplied with the cur
rency scale.
-- "no charge" indicates CurrencyFactorScale has the value 0.

-- CurrencyScale Type

CurrencyScale ::= INTEGER (-7..3)
-- The actual value for currency scale is given by 10x, where x is the value of
the CurrencyScale.
-- The coding of CurrencyScale is as follows, all other values are spare:
-- -7 (249): 0,0000001
-- -6 (250): 0,000001
-- -5 (251): 0,00001
-- -4 (252): 0,0001
-- -3 (253): 0,001
-- -2 (254): 0,01
-- -1 (255): 0,1
--  0      : 1
--  1      : 10
--  2      : 100
--  3      : 1000

CutAndPaste ::= INTEGER (0..22)
-- Indicates the number of leading digits to be deleted (cut) and to paste remai
ning dialed digits.

DateAndTime ::= OCTET STRING (SIZE(6..7))
-- Indicates, amongst others, the start time and stop time for activate service
filtering.
-- Coded as YYMMDDHHMMSS (option 1) or YYYYMMDDHHSS (option 2) with each digit c
oded BCD

-- OPTION 1 (Size 6):
-- The first octet contains YY and the remaining items are sequenced following
```

```
-- For example, 1998 September 30th, 12:15:01 would be encoded as:
-- Bits                     HGFE     DCBA
-- leading octet          8          9
--                                9          0
--                                0          3
--                                2          1
--                                5          1
--                               1          0
-- The 2 digit value
--  representing a Year shall be interpreted as follows
-- If the two-digits value is 00 through 49 inclusive, it shall be interpreted a
s representing
-- year 2000  through 2049.
-- If the two-digits value is 50 through 99 inclusive, it shall be interpreted a
s representng
-- year 1950 through 1999.

-- Option 2 (Size 7):
--  Support for CAMEL.
--  The year digit indicating millenium occupies bits
--  0-3 of the first octet, and the year digit indicating century occupies bits
--  4-7 of the first octet.
--  The year digit indicating decade occupies bits 0-3 of the second octet,
--  whilst the digit indicating the year within the decade occupies bits 4-7 of
--  the second octet.
--  The most significant month digit occupies bits 0-3 of the third octet,
--  and the least  significant month digit occupies bits 4-7 of the third octet.

--  The most significant day digit occupies bits 0-3 of the fourth octet,
--  and the least significant day digit occupies bits 4-7 of the fourth octet.
--  The most significant hours digit occupies bits 0-3 of the fifth octet,
--  and the least significant digit occupies bits 4-7 of the fifth octet.
--  The most significant minutes digit occupies bits 0-3 of the sixth octet,
--  and the least significant digit occupies bits 4-7 of the sixth octet.
--  The most significant seconds digit occupies bits 0-3 of the seventh octet,
--  and the least seconds significant digit occupies bits 4-7 of the seventh oct
et.
--  For the encoding of digits in an octet, refer to the timeAndtimezone paramet
er.


DefaultFaultHandling  ::= SEQUENCE{
      action          [0]               ENUMERATED  {
          resumeCallProcessing       (0),
          releaseCall                (1)
          },
     treatment     [1] GapTreatment   OPTIONAL
     }
-- The data type GapTreatment was reused from the existing types to avoid the de
finition
-- of a new one.

DestinationRoutingAddress  ::=
     SEQUENCE SIZE(1) OF CalledPartyNumber
-- Indicates the destination address for routeing.

Digits  ::=
     OCTET STRING (SIZE (minDigitsLength..maxDigitsLength))

-- Indicates the address signalling digits.
--  Refer to the Q.763 Generic Number and Generic Digits parameter for encoding.

-- The digits may also include the '#', '*', a, b and c digits.
-- The coding of the subfield's 'NumberQualifier' in Generic Number and 'TypeOfD
igits' 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 valu
e
-- is network operator specific.
-- The following parameters should use Generic Number
-- Additional Calling Party Number, CorrelationID for AssistRequestInstructions,
```

```
-- AssistingSSPIPRoutingAddress for EstablishTemporaryConnection,
-- calledAddressValue for all occurrences, callingAddressValue for all occurrenc
es
-- The following parameters should use Generic Digits: all
-- other CorrelationID occurrences, , lineID for ResourceID type, digitResponse
for
-- ReceivedInformationArg, iNServiceControlLow / iNServiceControlHigh for
-- MidCallInfoType, iNServiceControlCode for MidCallInfo.

DisplayInformation  ::= IA5String (SIZE (minDisplayInformationLength..
xDisplayInformationLength))
-- Indicates the display information

DpSpecificCriteria  ::= CHOICE {
          numberOfDigits               [0] NumberOfDigits,
          applicationTimer          [1] ApplicationTimer,
          midCallControlInfo          [2] MidCallControlInfo ,
          numberOfDigitsTwo          [3] SEQUENCE {
               requestedNumberOfDigits     [0] NumberOfDigits,
               minNumberOfDigits          [1] NumberOfDigits OPTIONAL
               }
          }
-- The SCF may specify the number of digits to be collected by the SSF for the C
ollectedInfo 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
-- The SCF may specify the number of digits to be collected by the SSF for the
-- CollecteInfo event and hereby specify a minimum number of digits to be collec
ted in case
-- the exact number of digits is unknown to the SCF, but a  report is desired in
 case
-- of complete number is determined before the requested number of digits has be
en collected.

XDuration ::= INTEGER (-2..86400)
-- Values are seconds
-- special meaning applies for -2, -1 and 0 values as described for the concerne
d
-- operation procedures.

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

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 {
               oNoAnswerCause                    [0] Cause  OPTIONAL
               },
     oAnswerSpecificInfo                    [5] SEQUENCE {

               backwardGVNS               [0] BackwardGVNS  OPTIONAL,
               destinationAddress          [50]     CalledPartyNumber  OPTIONAL,
-- CAMEL support
               or-Call                          [51]     NULL                    OPTIONAL,
-- CAMEL support
               forwardedCall               [52]      NULL                    OPTIONAL
-- CAMEL support
               },
```

```
     oMidCallSpecificInfo              [6] SEQUENCE {
               connectTime                        [0] Integer4 OPTIONAL,
               oMidCallInfo                   [1] MidCallInfo  OPTIONAL
               },
     oDisconnectSpecificInfo              [7] SEQUENCE {
               releaseCause                   [0] Cause  OPTIONAL,
               connectTime                        [1] Integer4 OPTIONAL
               },
     tBusySpecificInfo                   [8] SEQUENCE {
               busyCause                          [0] Cause  OPTIONAL,
               callForwarded               [50] NULL                        OPTIONAL
-- CAMEL suport
               },
     tNoAnswerSpecificInfo               [9] SEQUENCE {
               tNoAnswerCause                     [0] Cause  OPTIONAL,
               callForwarded                  [50] NULL          OPTIONAL
-- CAMEL support
               },
     tAnswerSpecificInfo                    [10] SEQUENCE {

               destinationAddress              [50]     CalledPartyNumber      OPTIONAL,
-- CAMEL support
               or-Call                             [51]     NULL                    OPTIO
-- CAMEL support
               forwardedCall                   [52]     NULL                     OPTIONAL
-- CAMEL support
               },
     tMidCallSpecificInfo              [11] SEQUENCE {
               connectTime                        [0] Integer4 OPTIONAL,
               tMidCallInfo                   [1] MidCallInfo  OPTIONAL
               },
     tDisconnectSpecificInfo              [12] SEQUENCE {
               releaseCause                   [0] Cause  OPTIONAL,
               connectTime                        [1] Integer4 OPTIONAL
               },
     oTermSeizedSpecificInfo              [13] NULL, --SEQUENCE {
               -- no specific info defined

               --},
     oSuspend                             [14] NULL, --SEQUENCE {
               -- no specific info defined
               --},
     tSuspend                             [15] NULL, --SEQUENCE {
               -- no specific info defined
               --},
     origAttemptAuthorized               [16] NULL, --SEQUENCE {
               -- no specific info defined
               --},
     oReAnswer                            [17] NULL, --SEQUENCE {
               -- no specific info defined
               --},
     tReAnswer                            [18] NULL, --SEQUENCE {
               -- no specific info defined
               --},
     facilitySelectedAndAvailable     [19] NULL, --SEQUENCE {
               -- no specific info defined
               --},
     callAccepted                        [20] NULL, --SEQUENCE {
               -- no specific info defined
               --},
     oAbandon                             [21] SEQUENCE {
               abandonCause                   [0] Cause  OPTIONAL
               },
     tAbandon                             [22] SEQUENCE {
               abandonCause                   [0] Cause  OPTIONAL
               },
     authorizeRouteFailure               [23] SEQUENCE {
               authoriseRouteFailureCause     [0] Cause  OPTIONAL
               },
     terminationAttemptAuthorized     [24] NULL, --SEQUENCE {
               -- no specific info defined

               --},
```

```
     orininationAttemptDenied          [25] SEQUENCE {
               originationDeniedCause          [0] Cause  OPTIONAL
               },
     terminationAttemptDenied          [26] SEQUENCE {
               terminationDeniedCause          [0] Cause  OPTIONAL
               }

     }
-- Indicates the call related information specific to the event.
-- The unit for the connectTime is 100 milliseconds

EventSpecificInformationCharging  ::= OCTET STRING (SIZE (minEventSpecificInform
ationChargingLength..maxEventSpecificInformationChargingLength))
-- defined by network operator.
-- Its content is network signalling/operator specific.
-- Indicates the charging related information specific to the event.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

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),
     oSuspend                         (20),
     tSuspend                         (21),
     origAttempt                         (22),
     termAttempt                         (23),
     oReAnswer                         (24),
     tReAnswer                         (25),
     facilitySelectedAndAvailable(26),
     callAccepted                    (27),
     authorizeRouteFailure          (28),
     originationAttemptDenied     (29),
     terminationAttemptDenied     (30)
     }

-- Indicates the name of the BCSM detection point event.
-- The value range 100- 127 is reserved.

EventTypeCharging  ::= OCTET STRING (SIZE
                                  (minEventTypeChargingLength..
                                   maxEventTypeChargingLength))
-- This parameter indicates the charging event type. .
-- Its content is network signalling / operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

EventTypeTariff                ::= ENUMERATED {
     chargingTariffInformation              (0),
     addOnchargingInformation              (1),
```

```
     chargingAcknowledgementInformation     (2),
     chargingAcknowledgeTimerExpired          (3),
     startCharging                              (4),
     stopCharging                              (5)
     }


FacilityGroup ::= CHOICE {
     trunkGroupID          [0] INTEGER,
     privateFacilityID     [1] INTEGER,
     huntGroup               [2] OCTET STRING,
     routeIndex               [3] OCTET STRING
     }
-- Indicates the particular group of facilities to route the call.
-- huntGroup and routeIndex are encoded as
-- network operator specific.
-- The internal structure of huntGroup and routeIndex parameters 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 information) is the BER encoding of the defin
ed ASN.1
-- internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

FCIBillingChargingCharacteristics  ::= CHOICE {
     fCIBCCcs1     OCTET STRING (SIZE (minFCIBCCcs1Length..
                                          maxFCIBCCcs1Length)),
-- Its content is network operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and
--length information) is the BER encoding of the defined ASN.1 internal structur
e.
-- The tag of this parameter as defined by ETSI is never replaced.

     fCIBCCsequence          [51] SEQUENCE {

          fCIBCC                              [0] OCTET STRING (SIZE (minFCIBCCcs2Length .
OPTIONAL,
-- Its content is network operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag
-- and length information) is the BER encoding of the defined ASN.1 internal str
ucture.
-- The tag of this parameter as defined by ETSI is never replaced.

          chargeMessageFromSCF          [1] CHOICE {
               sCFChargingTariff                    [0] ChargingTariff ,
               sCFAddOnCharge                        [1] AddOnCharge
               }


          tariffFromSuccExchange          [2] TariffFromSuccExchange, -- DEFAULT notTakenI
ount,
                    freeFormatData                    [3] OCTET STRING (SIZE (minFormatDat
aLength))     OPTIONAL,
-- Its content is network operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter
-- (after the first tag and lengthinformation) is the BER encoding of the
-- defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

          chargingAddress                    [4] ChargingAddress      DEFAULT leg:sendingS
          partyToChargeIdentifier          [5] PartyToCharge      OPTIONAL,
          iNRecordIdentifier               [6] INCallRecordIndicator,
          extensions                         [7] Extensions   OPTIONAL
          }
```

```
     }
-- This FCIBillingChargingCharacteristics parameter indicates the billing and/or

-- charging characteristics.
-- Its content is partly network operator specific.
-- CAMEL:
-- FCIBillingChargingCharacteristics {PARAMETERS-BOUND : bound} ::= OCTET STRING
 (SIZE
--     (bound.&minFCIBillingChargingLength..bound.&maxFCIBillingChargingLength))
--     (CONSTRAINED BY {
-- shall be the result of the BER-encoded value of type --
--     CAMEL-FCIBillingChargingCharacteristics })
-- This parameter indicates the billing and/or charging characteristics.
-- The violation of the UserDefinedConstraint shall be handled as an ASN.1 synta
x error.
-- The value of the FCIBillingChargingCharacteristics of type OCTET STRING carri
es
-- a value of the ASN.1 data type: CAMEL-FCIBillingChargingCharacteristics.
-- The normal encoding rules are used to encode this value.



FilteredCallTreatment  ::= SEQUENCE {
     sFBillingChargingCharacteristics     [0] SFBillingChargingCharacteristics      OPTION
,
     informationToSend                         [1] InformationToSend      OPTIONAL,
     maximumNumberOfCounters                    [2] MaximumNumberOfCounters     OPTIONAL,
     releaseCause                              [3] Cause      OPTIONAL,
     sFTariffMessage                              [50] CHOICE {
          crgt                                           [0] ChargingTariffInformation}
     }
-- If releaseCause is not present, the default value is the same as the ISUP cau
se value decimal 31.
-- If informationToSend is present, the call will be released after the end of t
he announcement
-- with the indicated or default releaseCause.
-- If maximumNumberOfCounters is not present, ServiceFilteringResponse will be s
ent with
-- CountersValue::= SEQUENCE SIZE (0) OF CountersAndValue.

FilteringCharacteristics::= CHOICE {
     interval               [0] INTEGER (-1..32000),
     numberOfCalls          [1] Integer4
     }
-- The interval is specified in seconds.

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-maximumNumberOfC
ounters.
-- For encoding of the digits, refer to ITU-T Recommendation Q.763 for Generic N
umber.

FilteringTimeOut ::= CHOICE {
     duration          [0] XDuration,
     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 ITU-T Recommendation Q.763 fo
```

```
r encoding

ForwardGVNS  ::= OCTET STRING (SIZE(
               minForwardGVNSLength..
               maxForwardGVNSLength))
-- Indicates the GVNS Forward information. Refer to ITU-T Recommendation Q.735 f
or 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
     -- callOfferingNoINImpact                    'xxxx x100'B
     -- indicates if call offering is "allowed", "not allowed" or "no impact by IN".

     -- 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

     callWaitingTreatmentIndicator          [5] OCTET STRING (SIZE(1))     OPTIONAL,
     -- callWaitingAllowed                          'xxxx xx01'B,
     -- callWaitingNotAllowed                         'xxxx xx10'B
     -- network default is Call Waiting allowed

     holdTreatmentIndicator                    [6] OCTET STRING (SIZE(1))     OPTIONAL,
     -- acceptHoldRequest     'xxxx xx01'B
     -- rejectHoldRequest     'xxxx xx10'B
     -- network default is accept hold request

     ectTreatmentIndicator                    [7]     OCTET STRING (SIZE(1))     OPTIONAL
     -- acceptEctRequest     'xxxx xx01'B
     -- rejectEctRequest     'xxxx xx10'B
     -- network default is accept ect request
}

--The forwardServiceInteractionInd parameter is applicable to IDP, CON, CWA and
ICA operations.

GapCriteria ::= CHOICE {
     basicGapCriteria          BasicGapCriteria ,
     compoundGapCriteria          CompoundCriteria
     }


GapOnService ::= SEQUENCE {
     serviceKey     [0] ServiceKey
     }
GapIndicators ::= SEQUENCE {
     duration     [0] XDuration,
     gapInterval     [1] Interval
     }
-- Indicates the gapping characteristics.

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))
-- Refer to ITU-T Recommendation Q.931 Display Information parameter for encodin
g.


GenericNumber   ::= OCTET STRING (SIZE(
                    minGenericNumberLength..
                    maxGenericNumberLength))
-- Refer to ITU-T Recommendation Q.763 Generic Number for encoding.

GenericNumbers  ::=
     SET SIZE(1..numOfGenericNumbers) OF GenericNumber

HighLayerCompatibility ::= OCTET STRING (SIZE (highLayerCompatibilityLength))
-- Indicates the teleservice. For encoding, DSS1 (Q.931) is used.


INCallRecordIndicator               ::= SEQUENCE {
     iNCallRecordType               [0] INCallRecordType, -- DEFAULT generateCallRecord,
     hotBillingRequired               [1] BOOLEAN OPTIONAL
     }

INCallRecordType                        ::= ENUMERATED {
     generateCallRecord               (1),
     appendDataToCallRecord          (2),
     overwriteDataForCallRecord     (3),
     noDataForCallRecord               (4)
     }

initialCallSegment INTEGER ::= 1

InitialDPArgExtension                   ::= SEQUENCE {
     gmscAddress                                [1] ISDN-AddressString          OPTIONA
}

INprofile   ::=SEQUENCE {
          actionOnProfile      [0] ActionOnProfile,
          tDPIdentifier          [1] TDPIdentifier ,
          dPName                    [2] EventTypeBCSM  OPTIONAL,
          extensions               [3] Extensions   OPTIONAL
          }

INServiceCompatibilityIndication  ::=
     SEQUENCE SIZE (1..numOfInServiceCompatibilityIndLength) OF Entry

INServiceCompatibilityResponse      ::= Entry

Interval ::= INTEGER (-1..60000)
-- Units are in milliseconds. A -1 value denotes infinite.

IPAvailable  ::= OCTET STRING (SIZE (
               minIPAvailableLength..
               maxIPAvailableLength))
-- defined by network operator.
-- Indicates that the resource is available.
-- Its content is network operator specific

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.
-- Its content is network operator specific

ISDNAccessRelatedInformation  ::= OCTET STRING (SIZE
```

```
               (minISDNAccessRelatedInfoLength..
                 maxISDNAccessRelatedInfoLength))
-- Indicates the destination user network interface related information. Refer t
o the Q.763 Access
-- Transport parameter for encoding.

DTLegID ::= CHOICE {
     sendingSideID     [0] DTLegType,
     receivingSideID     [1] DTLegType
     }
-- sendingSideID is used where legID is sent from the SCF  to the SSF and
-- receivingSideID is used  where SCF receives legID from the SSF.

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 ITU-T Recommend
ation Q.763
--  for encoding.

MaximumNumberOfCounters ::= INTEGER (1..numOfCounters)

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
     }

MiscCallInfo ::= 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.

NAOliInfo          ::= OCTET STRING (SIZE (1))
--  NA Oli information is only used by Camel for NA (North America).
--  It takes the same value as defined in ANSI ISUP T1.113
--  e.g.      '3D'H  - Decimal value 61 - Cellular Service (Type 1)
--                '3E'H  - Decimal value 62 - Cellular Service (Type 2)
--                '3F'H  - Decimal value 63 - Cellular Service (roaming)

NeededCost          ::= SEQUENCE {
     usedCurrencyFactor     [1] CurrencyFactor,
     usedScale               [2]     CurrencyScale     OPTIONAL,
```

```
     reportedCurrency     [3] Currency     --DEFAULT noIndication
     }

NeededPulses     ::= SEQUENCE {
     usedPulseUnits     [1] OCTET STRING (SIZE (2))
     }


normalRelease OCTET STRING ::= '839F'H

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

OCSIApplicable           ::= NULL
--  Indicates that the Originating CAMEL Subscription Information shall be
--  applied on the outgoing call leg created with a Connect operation. For the u
se of this
--  parameter see 3G TS 23.078.

OriginalCalledPartyID  ::= OCTET STRING (SIZE
               (minOriginalCalledPartyIDLength..
               maxOriginalCalledPartyIDLength))
-- Indicates the original called number.
-- Refer to the Q.763 Original Called Number for encoding.

PartyToCharge                          ::= SEQUENCE {
     chargedParty                     [0] ChargedParty, --DEFAULT callingPartyNumber,
     specialINNumber                     [1] ChargedPartyNumber      OPTIONAL
     }

ProfileIdentifier   ::= CHOICE {
     access          [0]  CalledPartyNumber ,
     group          [1]  FacilityGroup
      }
-- Please note that 'CalledPartyNumber' is used to address a subscriber access l
ine.
-- The data type was reused from the existing types to avoid the definition of a
 new one.
-- PulseUnits Type
PulseUnits ::= OCTET STRING (SIZE(1))
-- the PulseUnits is binary coded and has the value range from 0 to 255

Reason  ::= OCTET STRING (SIZE(
     minReasonLength..
     maxReasonLength))
-- Its content is network operator specific
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter
-- (after the first tag and length information) is the BER encoding of the defin
ed ASN.1
-- internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

ReceivingSideID               ::= CHOICE {receivingSideID [1] DTLegType}
-- used to identify DTLegID  in operations sent from gsmSSF to gsmSCF

RedirectingPartyID  ::= OCTET STRING (SIZE (
               minRedirectingPartyIDLength..
               maxRedirectingPartyIDLength))
-- Indicates redirecting number. Refer to the Q.763 Redirecting number for encod
ing.

RedirectionInformation ::= OCTET STRING (SIZE(2))
-- Indicates redirection information. Refer to the ITU-T Recommendation Q.763
-- Redirection Information for encoding.

RegistratorIdentifier  ::= OCTET STRING
-- Its content is network operator specific

RelayChargesFromDestination               ::= ENUMERATED {
     noIndication                     (0),
     relay                               (1),
```

```
     noRelay                              (2)
     }

ReleaseWhenLimitReached          ::= SEQUENCE {
     releaseCause          [1] Cause      DEFAULT normalRelease
     }

ReportCondition ::= ENUMERATED {
     statusReport     (0),
     timerExpired     (1),
     canceled          (2)
     }
-- ReportCondition specifies the cause of sending "StatusReport" operation to th
e SCF.

ReportConditions ::= SEQUENCE {
     periodicReportinterval          [1] INTEGER     (1..32767)
     -- Values are in units of seconds.
     }

ReportConditionInformation     ::= ENUMERATED {
     intermediateReport               (0),
     finalReportCallReleased      (1),
     finalReportCallActive          (2)
     }

RequestedInformationList  ::=
     SEQUENCE SIZE (1..numOfInfoItems) OF RequestedInformation

RequestedInformationTypeList ::= SEQUENCE SIZE (1..numOfInfoItems) OF RequestedI
nformationType

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

RequestedUTSI  ::= SEQUENCE {
     uSIServiceIndicator     [0] USIServiceIndicator ,
     uSImonitorMode          [1] USIMonitorMode
     }

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

ResourceID  ::= CHOICE {
     lineID                          [0] Digits ,
     facilityGroupID               [1] FacilityGroup,
     facilityGroupMemberID     [2] INTEGER,
     trunkGroupID                [3] INTEGER
     }
-- Indicates a logical identifier for the physical termination resource.

ResourceStatus ::= ENUMERATED {
```

```
     busy     (0),
     idle     (1)
     }

ResponseCondition ::= ENUMERATED {
     intermediateResponse     (0),
     lastResponse               (1)
     }
-- ResponseCondition is used to identify the reason why ServiceFilteringResponse
 operation is sent.

RouteList  ::=
     SEQUENCE SIZE(1..3) OF OCTET STRING (SIZE
               (minRouteListLength..
                maxRouteListLength))
-- Indicates a list of trunk groups or a route index. .
-- Its content is network operator specific

RouteingNumber   ::= OCTET STRING (SIZE
               (minRouteingNumberLength..
               maxRouteingNumberLength))
-- Indicates the Routeing Number.
-- Refer to ITU-T Recommendation Q.763  parameter Network Routeing Number for en
coding.


ScfID  ::= OCTET STRING (SIZE
                                    (minScfIDLength..
                                     maxScfIDLength))
-- defined by network operator.
-- Indicates the SCF identity.
-- Refer to Q.713 "calling party address" parameter  for encoding.
-- Other encoding schemes are also possible as a network specific option.

SCIBillingChargingCharacteristics  ::= OCTET STRING (SIZE
                                    (minSCIBillingChargingLength..
                                     maxSCIBillingChargingLength))
-- This parameter indicates the billing and/or charging characteristics.
-- Its content is network signalling / operator specific
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.
-- CAMEL:
--  SCIBillingChargingCharacteristics {PARAMETERS-BOUND : bound} ::= OCTET STRIN
G (SIZE (
--     minSCIBillingChargingLength..maxSCIBillingChargingLength))
--     (CONSTRAINED BY {
-- shall be the result of the BER-encoded value of type --
--     CAMEL-SCIBillingChargingCharacteristics})
-- Indicates AOC information to be sent to a Mobile Station
-- The violation of the UserDefinedConstraint shall be handled as an ASN.1 synta
x error.
-- The value of the SCIBillingChargingCharacteristics of type OCTET STRING carri
es
-- a value of the ASN.1 data type: CAMEL-SCIBillingChargingCharacteristics.
-- The normal encoding rules are used to encode this value.


SDSSinformation  ::= OCTET STRING (SIZE
                                    (minSDSSinformationLength..
                                     maxSDSSinformationLength))

-- Its content is network signalling/operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.

SendingSideID                              ::= CHOICE {sendingSideID  [0] DTLegType}
```

```
-- used to identify DTLegID in operations sent from gsmSCF to gsmSSF

ServiceInteractionIndicators  ::= OCTET STRING (SIZE (
                                     minServiceInteractionIndicatorsLength..
                                     maxServiceInteractionIndicatorsLength))

-- 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.

-- Its content is network signalling/operator specific

-- Note this parameter is kept ifor backward compatibility to IN CS-1,
-- for this ENsee parameter ServiceInteractionIndicatorsTwo
-- Its content is network signalling/operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

ServiceInteractionIndicatorsTwo ::=      SEQUENCE {
-- ServiceInteractionIndicatorsTwo contains Indicators which are exchanged betwe
en SSP and SCP
-- to resolve interactions between IN based services  and network based services
,
-- respectively between different IN based services.

     forwardServiceInteractionInd          [0] ForwardServiceInteractionInd     OPTIONAL,
-- applicable to operations IDP, CON, ICA, CWA.

     backwardServiceInteractionInd          [1] BackwardServiceInteractionInd     OPTIONAL
--applicable to operations IDP, CON, CTR, ETC, CWA.

     bothwayThroughConnectionInd               [2] BothwayThroughConnectionInd          OP
-- applicable to operations CTR, ETC.

     suspendTimer                              [3] SuspendTimer     OPTIONAL,
-- applicable to operations CON, ICA CWA.

     connectedNumberTreatmentInd               [4] ConnectedNumberTreatmentInd     OPTIONA
--applicable to operations CON, CTR, ETC, CWA .

     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 Address Presentation restricted indicator of the ISUP
-- "called IN number" shall be set to presentation allowed (TRUE)
--  or presentation restricted (FALSE). Refer to Rec. Q.1601.

     userDialogueDurationInd                    [8] BOOLEAN     DEFAULT TRUE,
-- applicable to operations CTR, ETC.
-- applicable when interaction with the user is required during call set-up
-- The interaction TRUE means the user interaction may last longer than 90 secon
ds.
-- Otherwise the indicator should be set to FALSE. Used  for delaying ISUP T9 ti
mer.

     overrideLineRestrictions               [9] BOOLEAN      DEFAULT FALSE,
-- only applicable to operations  (e.g. Connect) which lead to a transition to a
 PIC before
--  the AuthorizeCallSetup PIC.
-- When set to TRUE, this parameter indicates that some facility restrictions
-- should not be checked when the authority to place a call is verified in the
-- Authorize_Call_Setup PIC.
-- Which restrictions are actually overwriden is network specific.
```

```
     suppressVPNAPP                              [10] BOOLEAN DEFAULT FALSE,
-- applicable to CWA, CON, ICA.
-- indicates whether to allow or stop (suppress) the forward transmission of the

-- VPN PSS1 capability.
-- When set to TRUE, the  exchange, on receipt of this parameter, will not trans
mit for this call
-- any ISUP Application transport parameter with Application Context Identifier
set to
-- « PSS1 ASE (VPN) »
-- This indicator  is populated by the SCF, where the SCF and SSF in conjunction
 have provided the
-- outgoing gateway PINX functionality as required by PSS1.

calledINNumberOverriding                    [11] BOOLEAN     OPTIONAL,
-- applicable to CON and CWA
-- indicates whether the generation/override of  the ISUP
-- "called IN number"  is allowed (TRUE)   or  not allowed (FALSE)
-- If set to FALSE, the ISUP shall not generate a "called IN number"or override

-- an already existing "called IN number".
-- if absent, the default will be "generation/overriding allowed" (TRUE).

-- tag 12 IS RESERVED

nonCUGCall                                   [13] BOOLEAN     DEFAULT FALSE
-- applicable to CON and CWA
-- indicates whether no parameters for CUG shall be used for the call (i.e. the
call shall be
-- a non-CUG call)(TRUE) .
-- If set to TRUE, then neither CUG Interlock Code nor Outgoing
-- Access Indicator shall be present; if any of these parameters are present
-- then an error is returned.
-- If set to FALSE or not present, it indicates one of two things:
--     a) continue with modified CUG information (when one or more of either CUG Int
erlock Code
--     and Outgoing Access Indicator are present), or
--  b) continue with original CUG information (when neither CUG Interlock Code n
or Outgoing
--      Access Indicator are present).

}

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 filt
ered calls.
-- Its content is network signalling/operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced

SSFdetermination ::= ENUMERATED{
          noDetermination                (0),
          destinationRoutingAddress      (1),
          calledInNumber                    (2)
          }
-- The calledInNumber is the calledPartyNumber in the InitialDP operation.

-- SubTariffControl Type
SubTariffControl ::= BIT STRING {oneTimeCharge (0)}
(SIZE(minSubTariffControlLen..maxSubTariffControlLen))
-- The coding of the oneTimeCharge is as follows:
-- 0 - Periodic charge
-- 1 - One time charge
```

```
SupervisionMethod      ::= CHOICE {
     maximumTariffCurrency     [1] SEQUENCE {
          currencyFactor               [0] INTEGER (1..999999),
          -- Values are in currency units.
          currencyScale               [1] CurrencyScale DEFAULT noScale,
          -- Imported from ES 201 296.
          currency                    [2] Currency     --DEFAULT noIndication,
-- Imported from ES 201 296.

          },
     maximumTimeDuration          [2] SEQUENCE {
          maximumDurationAllowed      [0] INTEGER (1..864000),
          -- Values are in units of 100 msec.
          switchOverTime               [1]     INTEGER (1..86400)     OPTIONAL
          -- Values are in units of seconds.
          },
     maximumPulseUnits          [3] OCTET STRING (SIZE (2))
     }

--SupportedTriggers   TRIGGER ::= {...}

SuspendTimer ::= INTEGER (-1..120)
-- value in seconds
-- The default is as specified in EN 301 070-1
-- The value -1 indicates the network specific suspend timer (T6) is to be used.


-- TariffCurrency Type
TariffCurrency  ::= SEQUENCE {
          currentTariffCurrency     [0] TariffCurrencyFormat  OPTIONAL ,
          tariffSwitchCurrency     [1] TariffSwitchCurrency  OPTIONAL
          }
-- TariffSwitchCurrency Type
TariffSwitchCurrency  ::= SEQUENCE {
          nextTariffCurrency     [0] TariffCurrencyFormat ,
          tariffSwitchoverTime     [1] TariffSwitchoverTime
          }

TariffControlIndicators  ::= BIT STRING {non-cyclicTariff (0) }
          (SIZE(minTariffIndicatorsLen..maxTariffIndicatorsLen))

-- TariffCurrencyFormat Type
TariffCurrencyFormat  ::= SEQUENCE {
          communicationChargeSequenceCurrency          [0] SEQUENCE SIZE(minCommunicationT
um..maxCommunicationTariffNum)
                                  OF CommunicationChargeCurrency  OPTIONAL ,
          tariffControlIndicators                    [1] TariffControlIndicators ,
          callAttemptChargeCurrency               [2] CurrencyFactorScale OPTIONAL ,
          callSetupChargeCurrency                    [3] CurrencyFactorScale OPTIONAL
          }
-- The communication charge sequence currency is a direct charge in currency per
 time unit.
-- Only one fixed time unit is used.
-- This time unit has to be agreed between all cooperating networks, e.g. one se
cond.
-- Being fixed, the time unit is not transferred overISUP/INAP.
-- The call attempt charge is a direct charge, to be charged only for unsuccessf
ul calls.
-- The call set-up charge is a direct charge, to be charged once at start of cha
rging.
--
-- The coding of the non-cyclicTariff is as follows:
-- 0 - Cyclic tariff ( at expiration of the tariff duration of the last
-- communication tariff of the communication charge sequence, the communication
charge
-- sequence is re-applied.
-- 1 - Non-cyclic tariff ( at expiration of the tariff duration of the last
-- communication tariff of the communication charge sequence,
-- do not re-apply the communication charge sequence)

-- TariffDuration Type
TariffDuration ::= INTEGER (0..36000)
```

```
-- TariffDuration identifies with 0 unlimited duration and else in seconds unit.

-- 0 = unlimited
--1 =1second
--2 =2seconds
-- ...
-- 36000 = 10 hours
--
-- The duration indicates for how long time the communication charge component i
s valid.
-- Expiration of the tariff duration timer leads to the activation of the next
-- communication charge (if present).
-- In the case where there is no next communication charge in the communication
charge
-- sequence, the action to be performed is indicated by the tariffControlIndicat
ors.

-- TariffPulse Type
TariffPulse  ::= SEQUENCE {
          currentTariffPulse      [0] TariffPulseFormat  OPTIONAL ,
          tariffSwitchPulse      [1] TariffSwitchPulse  OPTIONAL
}

-- TariffSwitchPulse Type
TariffSwitchPulse  ::= SEQUENCE {
          nextTariffPulse           [0] TariffPulseFormat ,
          tariffSwitchoverTime      [1] TariffSwitchoverTime
          }

-- TariffPulseFormat Type
TariffPulseFormat  ::= SEQUENCE {
          communicationChargeSequencePulse      [0] SEQUENCE SIZE(minCommunicationTariffNu
.
                                     maxCommunicationTariffNum) OF CommunicationChargePulse
          tariffControlIndicators               [1] TariffControlIndicators ,
          callAttemptChargePulse                    [2] PulseUnits OPTIONAL ,
          callSetupChargePulse                    [3] PulseUnits OPTIONAL
          }
-- The communication charges are meter-pulse units, which are to be applied per
charge unit time interval.
-- The call attempt pulse units are to be charged only for unsuccessful calls.
-- The call set-up pulse units are to be charged once at start of charging..Draf
t ETSI ES 201 296 V1.2.1 (1999-11) 34

-- TariffSwitchoverTime Type
TariffSwitchoverTime ::= OCTET STRING (SIZE(1))
-- This time is the absolute time at which the next tariff has to become active.

-- It is represented in steps of 15 minutes.
-- The coding is the following:
-- 0 : spare
--1 :0hour 15 minutes
--2 :0hour 30 minutes
--3 :0hour 45 minutes
--4 :1hour 0 minutes
-- ..
-- 96 : 24 hours 0 minutes
-- 97-255 : spare


TariffFromSuccExchange               ::= ENUMERATED {
     notTakenIntoAccount                    (0),
     takeIntoAccountNoAOC                (1),
     takeIntoAccountTranslateIntoAOC      (2)
     }

TariffInfoFromSuccExchangeSCI           ::= SEQUENCE {
     relayChargesFromDestination          [0] RelayChargesFromDestination          DEFAULT
     relayAOCfromDestination               [1] BOOLEAN          DEFAULT FALSE
     }

TDPIdentifier   ::=  CHOICE {
```

```
     oneTrigger           INTEGER,
     triggers      [1] Triggers
     }

TerminalType ::= ENUMERATED {
     unknown          (0),
     dialPulse     (1),
     dtmf          (2),
     isdn          (3),
     isdnNoDtmf     (4),
     spare          (16)
     }

TimeAndTimezone ::= OCTET STRING (SIZE(minTimeAndTimezoneLength..
                                                                    maxTimeAndTimez
--  Support for CAMEL
--  Indicates the time and timezone, relative to GMT. This parameter BCD encoded
.
--  The year digit indicating millenium occupies bits 0-3 of the first octet, an
d the year
--  digit indicating century occupies bits 4-7 of the first octet.
--  The year digit indicating decade occupies bits 0-3 of the second octet, whil
st the digit
--  indicating the year within the decade occupies bits 4-7 of the second octet.

--  The most significant month digit occupies bits 0-3 of the third octet, and t
he least
--  significant month digit occupies bits 4-7 of the third octet.
--  The most significant day digit occupies bits 0-3 of the fourth octet, and th
e least
--  significant day digit occupies bits 4-7 of the fourth octet.
--  The most significant hours digit occupies bits 0-3 of the fifth octet, and t
he least
--  significant hours digit occupies bits 4-7 of the fifth octet.
--  The most significant minutes digit occupies bits 0-3 of the sixth octet, and
 the least
--  significant minutes digit occupies bits 4-7 of the sixth octet.
--  The most significant seconds digit occupies bits 0-3 of the seventh octet, a
nd the least
--  significant seconds digit occupies bits 4-7 of the seventh octet.
--
--  The timezone information occupies the eigth octet. For the encoding of Timez
one refer to
--  Reference [29], 3G TS 23.040.
--
--  The BCD digits are packed and encoded as follows:
--
--  Bit      7  6  5   4  |  3  2  1  0
--       2nd digit    | 1st digit          Octet 1
--       3rd digit    | 4th digit          Octet 2
--                          ..                   ..
--       nth digit    | n-1th digit        Octet m
--
--       0000       digit 0
--       0001       digit 1
--       0010       digit 2
--       0011       digit 3
--       0100       digit 4
--       0101       digit 5
--       0110       digit 6
--       0111       digit 7
--       1000       digit 8
--       1001       digit 9
--       1010       spare
--       1011       spare
--       1100       spare
--       1101       spare
--       1110       spare
--       1101       spare
--
-- where the leftmost bit of the digit is either bit 7 or bit 3 of the octet.

TimeInformation                                  ::= CHOICE {
```

```
      timeIfNoTariffSwitch                    [0] TimeIfNoTariffSwitch,
      timeIfTariffSwitch                         [1] TimeIfTariffSwitch
     }
-- Indicates call duration information
-- Support for CAMEL

TimeIfNoTariffSwitch                    ::= INTEGER(0..864000)
-- TimeIfNoTariffSwitch is measured in 100 millisecond intervals
-- Support for CAMEL

TimeIfTariffSwitch                           ::= SEQUENCE {
      timeSinceTariffSwitch                    [0] INTEGER(0..864000),
      tariffSwitchInterval                    [1] INTEGER(1..864000)          OPTIONAL
     }
-- timeSinceTariffSwitch and tariffSwitchInterval are measured in 100 millisecon
d intervals
-- Support for CAMEL

TimerID ::= ENUMERATED {
     tssf(0)
     }
-- Indicates the timer to be reset.

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 i
nfinite duration.

--TriggerData ::= SEQUENCE {
--     triggerId           [0]      TRIGGER.&id          ({SupportedTriggers}),
--     triggerPar           [1]          TRIGGER.&Parameter ({SupportedTriggers}{@triggerI
--     ...
--     }

TriggerDataIdentifier  ::= SEQUENCE {
     triggerID          [0] EventTypeBCSM,
     profile               [1] ProfileIdentifier ,
     extensions          [2] Extensions       OPTIONAL
     }
-- It is outside the scope of this capability set  whether all TDP types really
apply

TriggerDPType ::= ENUMERATED {tdp-r(0),
                    tdp-n(1) --added for SDL model, was in ITU-T Q.1238
                    }

TriggerResults  ::=
     SEQUENCE SIZE (1..numOfTriggers) OF TriggerResult

TriggerResult  ::=SEQUENCE {
     tDPIdentifer          [0] INTEGER,
     actionPerformed         [1] ActionPerformed,
     dPName                    [2] EventTypeBCSM OPTIONAL
     }

Triggers  ::=SEQUENCE SIZE(1..numOfTriggers) OF Trigger

Trigger  ::=SEQUENCE {
     tDPIdentifier            [0] INTEGER,
     dpName                    [1] EventTypeBCSM OPTIONAL
     }

TriggerStatus ::= ENUMERATED {
               created           (0),
               alreadyExist      (1),
               deleted           (2),
               unknownTrigger     (3)
               }
```

```
UsedCost          ::= SEQUENCE {
     usedCurrencyFactor     [1] CurrencyFactor,
-- imported from ES 201 296
     usedScale               [2]     CurrencyScale     OPTIONAL,
-- imported from ES 201 296
     reportedCurrency     [3] Currency     --DEFAULT noIndication,
-- imported from ES 201 296

     }

UsedTime     ::= SEQUENCE {
     usedDuration                         [1] INTEGER (0..864000),
     -- Units are in 100 msec.
     usedDurationAfterSwitchOverTime     [2] INTEGER (0..864000)     OPTIONAL
     -- Units are in 100 msec.
     }

UsedPulses     ::= SEQUENCE {
     usedPulseUnits     [1] OCTET STRING (SIZE (2))
     }

USIInformation  ::= OCTET STRING (SIZE
               (minUSIInformationLength..
                maxUSIInformationLength))
-- Its content is network signalling/operator specific
-- Indicates the length of the USIInformation element, maxUSIInformationlength w
ill depend on
-- the constraints imposed by the network signalling used to transport the USI i
nformation.
-- Its content is network signalling/operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

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))
}
-- In case of local its content is network signalling/operator specific
-- Its content is network signalling/operator specific.
-- The internal structure of this parameter can be defined using ASN.1 and the r
elated Basic
-- Encoding Rules (BER). In such a case the value of this paramter (after the fi
rst tag and length
-- information) is the BER encoding of the defined ASN.1 internal structure.
-- The tag of this parameter as defined by ETSI is never replaced.

VPNIndicator ::= BOOLEAN
-- This parameter is set to TRUE if the originating call part supports
-- VPN with PSS1 information flows

WarningBeforeLimitReached     ::= SEQUENCE {
     durationBeforeLimitReached     [1] INTEGER (1..255),
     -- The values are in units of 1 second.
     warningToSend                    [2] Tone          OPTIONAL,
     warningDirection               [3] DTLegID                OPTIONAL
     }

-- The Definition of range of constants Follows
```

```
highLayerCompatibilityLength     INTEGER ::= 2
minCauseLength                        INTEGER ::= 1
numOfCounters                        INTEGER ::= 100
numOfInfoItems                        INTEGER ::= 5


END
```

ASN.1 Text INCS3Internals

```
INCS3Internals

DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
IMPORTS
        CalledPartyNumber,
        CallingPartyNumber,
        CallingPartysCategory,
        Cause,
     DefaultFaultHandling, -- 3.6
        DestinationRoutingAddress,
        Digits,
        DTLegType,
        EventTypeBCSM,
     EventTypeCharging,
     FacilityGroup,
        ForwardCallIndicators,
     GenericNumbers,
        DTLegID,
        MonitorMode,
     numOfLegs,
     OriginalCalledPartyID,
     ProfileIdentifier,
     RedirectingPartyID,
     RedirectionInformation,
     RouteList,
        ServiceKey,
     TDPIdentifier,
     TriggerDPType,
        USIServiceIndicator,
        USIInformation
FROM INCS3datatypes
;
-- 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 {
  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,
  positionInChain [4] INTEGER
}



-- DEFINITION OF TYPES RELATED TO THE DEFINITION OF TDP TABLES
-- (INFORMATIVE)

TDPEntry ::= SEQUENCE {
  tdpIdentifier          INTEGER,
  dpName                 EventTypeBCSM,
  legID                DTLegType,
  triggerDPType          TriggerDPType,
  serviceKey             ServiceKey,
  profile              ProfileIdentifier OPTIONAL,
--  triggerData          TriggerData,
  defaultFaultHandling DefaultFaultHandling,
  isArmed                BOOLEAN,            -- TRUE if the trigger is active
  positionInChain        INTEGER OPTIONAL    -- (ETSI SDL model specific) positi
on
                                             -- in the BCSM chain to which the T
DP entry applies
}

TDPTableType ::= SEQUENCE OF TDPEntry
```

```
-- DEFINITION OF TYPES RELATED TO THE DEFINITION OF EDP TABLES
-- (INFORMATIVE)

EDPEntry ::= SEQUENCE {
  dpName                 EventTypeBCSM,
  legID                DTLegType,
  eDPType              EDPType
}

EDPType ::= ENUMERATED {edp-r(0), edp-n(1)}

EDPTableType ::= SEQUENCE OF EDPEntry


 -- 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, AnalyseInformation
    waitMidCall(3),                -- Used with Connect
    selectRoute (4),               -- Used with Connect, SelectRoute
    origAttempt (5),                   -- Used with Connect
    collectInfo (6)            -- Used with CollectInformation
}

DPFacilityArg ::= SEQUENCE {
  legID               DTLegType,
  party               PartyType
}



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

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
}

emptyString OCTET STRING ::= ''H

oct0 OCTET STRING ::= '00'H
oct1 OCTET STRING ::= '01'H
oct2 OCTET STRING ::= '02'H
oct3 OCTET STRING ::= '03'H
oct4 OCTET STRING ::= '04'H
```

```
oct5 OCTET STRING ::= '05'H
oct6 OCTET STRING ::= '06'H
oct7 OCTET STRING ::= '07'H
oct8 OCTET STRING ::= '08'H
oct9 OCTET STRING ::= '09'H
octa OCTET STRING ::= '0A'H
octb OCTET STRING ::= '0B'H


BCSMPars ::= 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,
  callRef                     [15] CallRef                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),
  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
  reconnect(21),             -- Note: IN CS-3
  midCall(22)             -- Used with DisconnectLeg operation
}


SignallingState ::= ENUMERATED {   --  Real signalling state of legs in Call
  idle (0),
  requested (1),
  progressed (2),
  acknowledged (3),
  confirmed (4)
}

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,
  facilityGroup              [18] FacilityGroup           OPTIONAL,
  cause                      [17] Cause                   OPTIONAL,
  stopBCSM                   [19] BOOLEAN                 OPTIONAL
  }

-- maxActiveServices determines how many services can be simultaneously active
-- on the same leg.
-- This constant thereby also governs the length of the fixed chain of BCSMs

maxActiveServices INTEGER ::= 3

-- Return type for the digit analysis in the O-BCSM
DigitAnalysisResultType ::= ENUMERATED {
  complete(0),
  incomplete(1) ,
  invalid(2)
}

END
```

```
USE INCS3Internals;
USE INCS3datatypes;
USE INCS3opsargs;
USE INCS3BundleArg;
USE CS3_TCAP;
```

package CS3_INAP 1(8)

```
/* This package contains
 * the SDL model of the
 * SSF/CCF of ETSI Core INAP CS−3
 */
```

SSF_CCF

The role of the SSF/CCF block type is to:
− Model the half call view as defined in Q.1238.
− Perform call processing.

package CS3_INAP 2(8)

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

/* Note: − Connect is defined as Connnect and Continue as Continnue
         because these are reserved words in SDL.
*/
```

```
SIGNAL
 /* SCF − SSF operations */
 ActivateServiceFiltering(InvokeID,CSAID, ActivateServiceFilteringArg),
 ActivateServiceFilteringResult(InvokeID, CSAID),
 ActivityTest(InvokeID, CSAID),
 ActivityTestResult(InvokeID, CSAID),
 ApplyCharging(InvokeID,CSAID,ApplyChargingArg),
 ApplyChargingReport(CSAID,ApplyChargingReportArg, Boolean),
 AssistRequestInstructions(CSAID,AssistRequestInstructionsArg),
 CallGap(InvokeID,CSAID,CallGapArg),
 CallInformationReport(CSAID,CallInformationReportArg,Boolean),
 CallInformationRequest(InvokeID,CSAID,CallInformationRequestArg),
 Cancel(InvokeID,CSAID,CancelArg),
 CollectInformation(InvokeID,CSAID,CollectInformationArg),
 Connnect(InvokeID,CSAID,ConnectArg,Boolean),
 ConnectToResource(InvokeID,CSAID,ConnectToResourceArg),
 Continnue(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);
```

package CS3_INAP 3(8)

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

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

/* Informative signal from SRF */
SIGNAL
  SRFReleased;

/* Informative operations */
SIGNAL
  Mgt_SetTriggerTable(MGTSetTriggerTableArg); /* Informative signal, used to model the management of the trigger table. */

/* No Operations, but additional Signals exchanged between TCAP Adapter, SSF and SCF */
SIGNAL
  CSaIDreq(DialogIDtype),
  CSaIDresp(DialogIDtype,CSAID),
  RegisterSSFreq(SSME_CroleType),
  RegisterSSFresp(SSME_CroleType);
```

```
/* INAP operations added after CS−1 */
SIGNAL
  CancelStatusReportRequest(InvokeID,DialogIDtype,CancelStatusReportRequestArg),
  ContinueWithArgument(InvokeID,CSAID,ContinueWithArgumentArg),
  CreateCallSegmentAssociation(InvokeID,CSAID,CreateCallSegmentAssociationArg),
  CreateCallSegmentAssociationResult(InvokeID,CSAID,CreateCallSegmentAssociationResultArg),
  CreateOrRemoveTriggerData(InvokeID,CSAID,CreateOrRemoveTriggerDataArg),
  CreateOrRemoveTriggerDataResult(InvokeID,CSAID,CreateOrRemoveTriggerDataResultArg),
  DisconnectLeg(InvokeID,CSAID,DisconnectLegArg),
  DisconnectLegResult(InvokeID,CSAID),
  DisconnectForwardConnectionWithArgument(InvokeID,CSAID,DisconnectForwardConnectionWithArgumentArg),
  EntityReleased(CSAID,EntityReleasedArg),
  ManageTriggerData(InvokeID,CSAID,ManageTriggerDataArg),
  ManageTriggerDataResult(InvokeID,CSAID,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),
  RequestCurrentStatusReport(InvokeID,DialogIDtype,RequestCurrentStatusReportArg),
  RequestCurrentStatusReportResult(InvokeID,DialogIDtype,RequestCurrentStatusReportResultArg),
  RequestEveryStatusChangeReport(InvokeID,DialogIDtype,RequestEveryStatusChangeReportArg),
  RequestEveryStatusChangeReportResult(InvokeID,DialogIDtype),
  RequestFirstStatusMatchReport(InvokeID,DialogIDtype,RequestFirstStatusMatchReportArg),
  RequestFirstStatusMatchReportResult(InvokeID,DialogIDtype),
  RequestReportUTSI(InvokeID,CSAID,RequestReportUTSIArg),
  SelectFacility(InvokeID,CSAID,SelectFacilityArg),
  SetServiceProfile(InvokeID,CSAID,SetServiceProfileArg),
  SendSTUI(InvokeID,CSAID,SendSTUIArg),
  SplitLeg(InvokeID,CSAID,SplitLegArg),
  SplitLegResult(InvokeID,CSAID),
  StatusReport(DialogIDtype,StatusReportArg),

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

package CS3_INAP 4(8)

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

```
/* SSF −> SCF */

SIGNALLIST CS1_INAP_To_SCF =
  ActivateServiceFilteringResult,
  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,
  Continnue,
  DisconnectForwardConnection,
  EstablishTemporaryConnection,
  FurnishChargingInformation,
  InitiateCallAttempt,
  ReleaseCall,
  RequestNotificationChargingEvent,
  RequestReportBCSMEvent,
  ResetTimer,
  SendChargingInformation,

  PlayAnnouncement,
  PromptAndCollectUserInformation,

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


  ApplicationBegin,
  ApplicationContinue,
  ApplicationAbort,
  ApplicationEnd,

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

package CS3_INAP 5(8)

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

```
/* SSF −> SCF */
SIGNALLIST CS3_INAP_To_SCF =
  (CS1_INAP_To_SCF),
  ActivityTest,
  CreateCallSegmentAssociationResult,
  CreateOrRemoveTriggerDataResult,
  DisconnectLegResult,
  EntityReleased,
  ManageTriggerDataResult,
  MergeCallSegmentsResult,
  MoveCallSegmentsResult,
  MoveLegResult,
  RequestCurrentStatusReportResult,
  RequestEveryStatusChangeReportResult,
  RequestFirstStatusMatchReportResult,
  ReportUTSI,
  SplitLegResult,
  StatusReport,

  PromptAndReceiveMessageResult,
  ScriptEvent;

/* SCF −> SSF */
SIGNALLIST CS3_INAP_From_SCF =
  (CS1_INAP_From_SCF),
  ActivityTestResult,
  CancelStatusReportRequest,
  ContinueWithArgument,
  CreateCallSegmentAssociation,
  CreateOrRemoveTriggerData,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  ManageTriggerData,
  MergeCallSegments,
  MoveCallSegments,
  MoveLeg,
  RequestCurrentStatusReport,
  RequestEveryStatusChangeReport,
  RequestFirstStatusMatchReport,
  RequestReportUTSI,
  SelectFacility,
  SetServiceProfile,
  SendSTUI,
  SplitLeg,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun;
```

```
/* SSF −> SRF */
SIGNALLIST CS3_SRF_Out =
   PlayAnnouncement,
   PromptAndCollectUserInformation,
   PromptAndReceiveMessage,
   Cancel,
   ScriptRun,
   ScriptInformation,
   ScriptClose;

/* SRF −> SSF */
SIGNALLIST CS3_SRF_In =
  SpecializedResourceReport,
  PromptAndCollectUserInformationResult,
  PromptAndReceiveMessageResult,
  ScriptEvent,
  SRFReleased;
```

```
USE INCS3datatypes;
USE INCS3opsargs;
USE INCS3BundleArg;
USE CS3_TCAP;
```

package CS3_INAP 6(8)

```
/**** SIGNAL DEFINITIONS FOR THE ERROR AND DIALOGUE HANDLING ****/

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

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

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

```
/* DummyType is just there to add the IntToDuration operator. */
NEWTYPE DummyType
OPERATORS
  IntToDuration : Integer −> Duration;

  OPERATOR IntToDuration; FPAR Seconds Integer;
    RETURNS Duration;
  START;
    RETURN 1.0 * Float (Seconds);
  ENDOPERATOR;
ENDNEWTYPE;
```

package CS3_INAP 7(8)

```
/**** 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 CS3 extension to
the SigCon interfaces */

SIGNAL
  NetworkSuspendInd(NetworkSRType),
  NetworkSuspendReq(NetworkSRType),
  NetworkResumeInd(NetworkSRType),
  NetworkResumeReq(NetworkSRType),
  DataReq(UserDataType),
  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),
  NetworkResumeReqInd(NetworkSRType),
  DataReqInd(UserDataType);

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

package CS3_INAP 8(8)

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

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

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

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

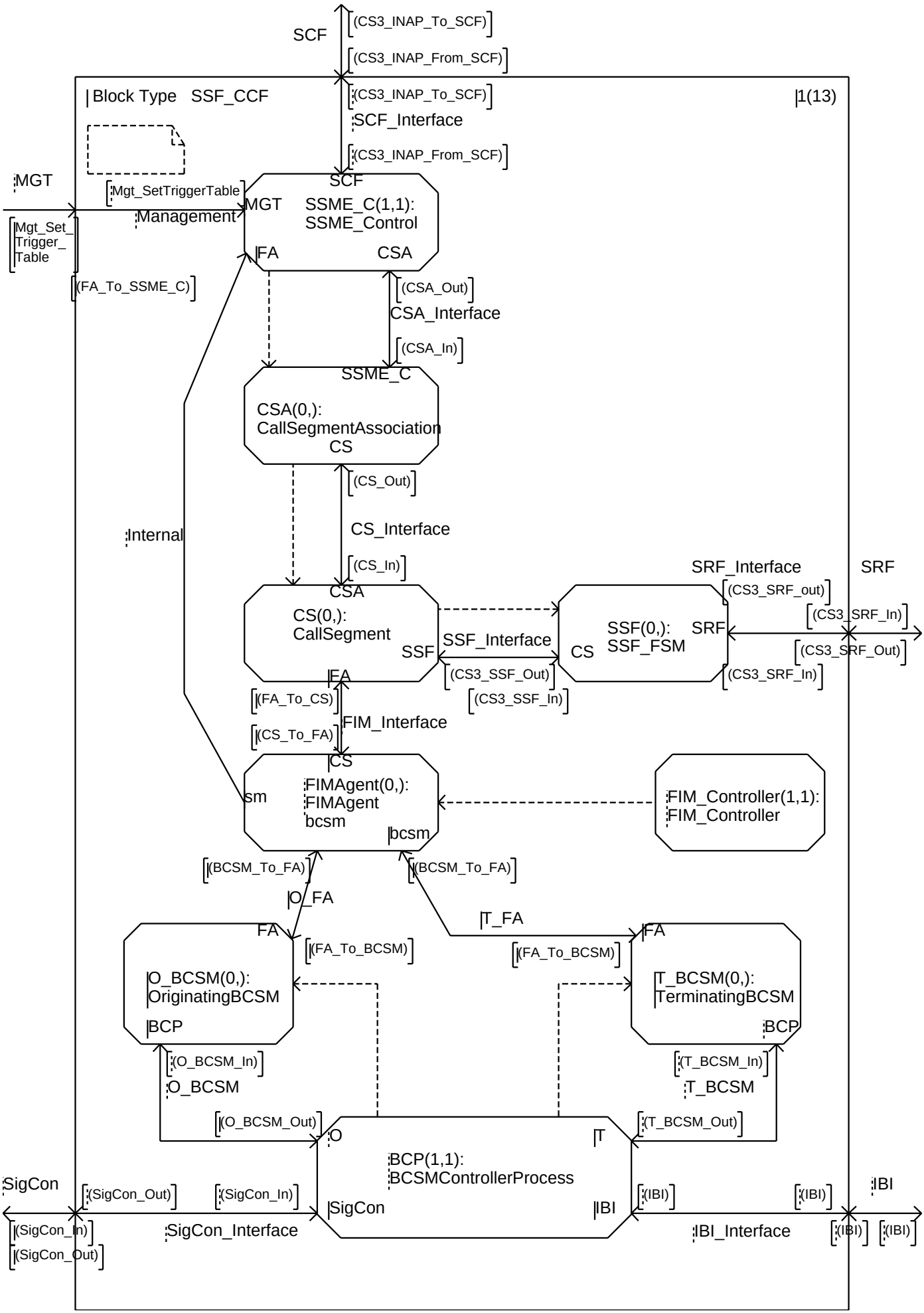
```
/* From SigCon to O−BCSM */

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

/* from O−BCSM to SigCon */

SIGNALLIST O_SigCon_Out =
  CallProgressReq,
  ReleaseReq,
  SetupResp,
  SubsequentAddressReq,
  ServiceFeatureIndReq;

/* from T−BCSM to SigCon */

SIGNALLIST T_SigCon_Out =
  ReleaseReq,
  SetupReq,
  SubsequentAddressReq,
  ServiceFeatureIndReq;

/* from called agent to T−BCSM */

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

```
SIGNALLIST SigCon_In =
  (O_SigCon_In),
  (T_SigCon_In),
  (SigCon2_In);

SIGNALLIST SigCon_Out =
  (O_SigCon_Out),
  (T_SigCon_Out),
  (SigCon2_Out);
```

```
/* Terminating to Originating BCSM */

SIGNALLIST IBI_In =
  CallProgressReqInd,
  ReleaseReqInd,
  ServiceFeatureReqInd,
  SetupRespConf;

/* Originating to Terminating BCSM */

SIGNALLIST IBI_Out =
  ReleaseReqInd,
  ServiceFeatureReqInd,
  SetupReqInd,
  SubsequentAddressReqInd;

SIGNALLIST IBI =
(IBI_In),
(IBI_Out),
(IBI2);
```

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

SIGNAL
  CallProgressReqInd(CallProgressType),
  ReleaseReqInd(ReleaseType),
  ServiceFeatureReqInd(ServiceFeatureType),
  SetupReqInd(SetupIRType),
  SetupRespConf(SetupCRType),
  SubsequentAddressReqInd(SubsequentAddressType);
```

SSF_CCF


SCF
(CS3_INAP_To_SCF)
(CS3_INAP_From_SCF)
Block Type SSF_CCF
1(13)
(CS3_INAP_To_SCF)
SCF_Interface
(CS3_INAP_From_SCF)
MGT
Mgt_SetTriggerTable
Management
Mgt_Set_Trigger_Table
SCF
MGT
SSME_C(1,1):
SSME_Control
FA
CSA
(FA_To_SSME_C)
(CSA_Out)
CSA_Interface
(CSA_In)
SSME_C
CSA(0,):
CallSegmentAssociation
CS
(CS_Out)
Internal
CS_Interface
(CS_In)
SRF_Interface
SRF
(CS3_SRF_out)
(CS3_SRF_In)
CSA
CS(0,):
CallSegment
SSF
SSF_Interface
SSF(0,):
SSF_FSM
SRF
CS
(CS3_SRF_Out)
(CS3_SSF_Out)
(CS3_SRF_In)
(CS3_SSF_In)
FA
(FA_To_CS)
FIM_Interface
(CS_To_FA)
CS
sm
FIMAgent(0,):
FIMAgent
bcsm
bcsm
FIM_Controller(1,1):
FIM_Controller
(BCSM_To_FA)
(BCSM_To_FA)
O_FA
T_FA
FA
(FA_To_BCSM)
FA
(FA_To_BCSM)
O_BCSM(0,):
OriginatingBCSM
BCP
T_BCSM(0,):
TerminatingBCSM
BCP
(O_BCSM_In)
O_BCSM
(T_BCSM_In)
T_BCSM
(O_BCSM_Out)
(T_BCSM_Out)
O
T
BCP(1,1):
BCSMControllerProcess
SigCon
(SigCon_Out)
(SigCon_In)
SigCon
IBI
(IBI)
(IBI)
IBI
(SigCon_In)
(SigCon_Out)
SigCon_Interface
IBI_Interface
(IBI)
(IBI)

Block Type SSF_CCF 2(13)

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

SSME_
_Control

The SSME_Control (SSME_C) process type manages:
− Processing of the operations MoveCallSegments and CreateCSA.
− The dialogue with the SCF.
− The passing of primitives between the SSF−SCF Interface and the CSA.
− The processing of IN CS−3 management operations
− The handling of static arming of TDPs.
− The processing of ManageTriggerData

CallSegment
Association

The CallSegmentAssociation (CSA) process type manages:
− The creation of CallSegments
− Processing of call segment related operations.
− The passing of primitives between the SSME_Control and the CallSegment.

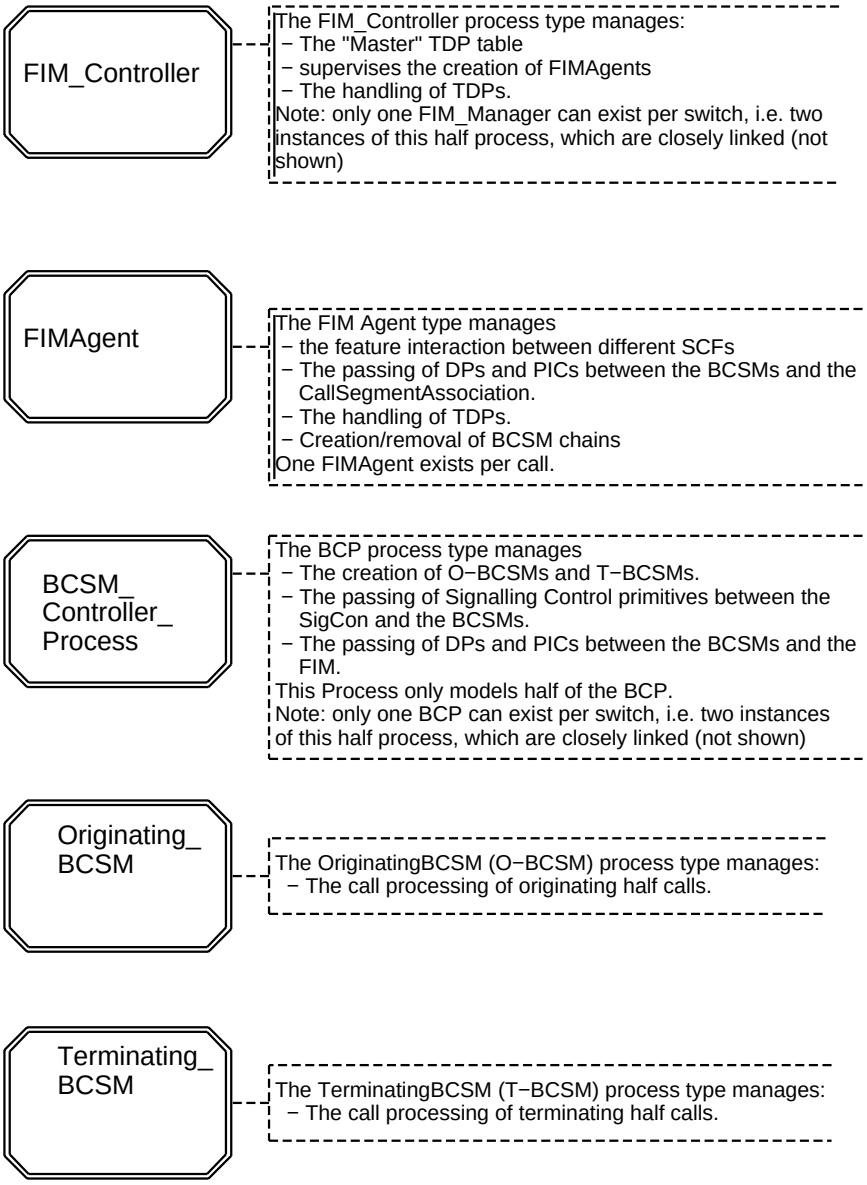
CallSegment

The CallSegment (CS) process type manages:
− The connection view (in terms of legs) of the call segment.
− The creation of the SSF−FSM.
− Processing of connection view oriented operations.
− Filtering of detection points
− The passing of primitives between the CallSegmentAssociation and the SSF−FSM.
− The passing of DPs and PICs between the FIM_Manager and the SSF−FSM.

SSF_FSM

The SSF−FSM process type manages:
− The processing of IN operations.
− The handling of EDPs.
Note: Only one SSF−FSM/CS can exist.

Block Type SSF_CCF 3(13)



Block Type SSF_CCF 4(13)

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

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

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

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

/* Signals from T−BCSM to SSF */
SIGNALLIST tDPs =
  DP,
  DPAnswer,
  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, the FIM and the BCP. */
```

```
SIGNAL
  PIC(PICArg),
  PICResume(LegType, Boolean);
/* The second parameter indicates if the PICResume is a result of the last armed EDP. */

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

SIGNALLIST tPICs =
  PIC,
  PICResume;

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

Block Type SSF_CCF 5(13)

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

```
/* Definition of Leg */

NEWTYPE LegStatusType
  LITERALS joined, pending, surrogate;
ENDNEWTYPE;

NEWTYPE BCSMType
  LITERALS Originating, Terminating;
ENDNEWTYPE;

NEWTYPE LegInfo STRUCT
  Used Boolean;            /* Flag indicating whether the leg is in use or not. */
  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;
```

```
NEWTYPE LegOperators
/* Placeholder for any leg related operators */
  OPERATORS
     IntToLegID: Integer −> LegType;

OPERATOR IntToLegID;
 FPAR i Integer;
 RETURNS res LegType;
START ;
 DECISION i;
  (1):
    RETURN leg1;
  (2):
    RETURN leg2;
  (3):
    RETURN oct3;
  (4):
    RETURN oct4;
  (5):
    RETURN oct5;
  (6):
    RETURN oct6;
  (7):
    RETURN oct7;
  (8):
    RETURN oct8;
  (9):
    RETURN oct9;
  (10):
    RETURN octa;
  (11):
    RETURN octb;
 ENDDECISION;
ENDOPERATOR IntToLegID;
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 1:30
ENDSYNTYPE;

SYNONYM numOfBCSMEvents NATURAL = 30;

NEWTYPE EventListType
  ARRAY(NumberOfBCSMEvents,BCSMEvent)
ENDNEWTYPE;

SYNTYPE MaxNumOfLegs = NATURAL
  CONSTANTS 1:numOfLegs
ENDSYNTYPE;

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

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


NEWTYPE EventTableType
  ARRAY(LegType,EventRecordType)
ENDNEWTYPE;

SYNTYPE EventRecordType = EDPTableType
ENDSYNTYPE;

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

Block Type SSF_CCF 6(13)

```
/**** DEFINITIONS OF INTERNAL SIGNALS BETWEEN PROCESSES. */
```

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

SIGNAL
  BCSMException(EntityReleasedArg, Boolean),
    /* CS −> CSA : indication that a BCSM has stopped. The first parameter contains the argument to
       the EntityReleased operation that may be sent by the CSA to the SCF. The second parameter
       indicates if the stopped BCSM caused the CS to stop. */
  BCSMStop(PartyType,Boolean),
    /* BCSM −> FIMAgent : indication that the BCSM has stopped. The second parameter indicates if the
       BCSM stopped due to an exception. */
  BCSMStopped(LegType, PartyType, Boolean),
    /* FIMAgent −> CS : indication that the BCSM has stopped. The third parameter indicates if the
       BCSM stopped due to an exception. */
  StopBCSM(Boolean),
    /* FIMAgent−>BCSM, BCP −> BCSM : instruction to stop the BCSM. If the parameter is true, the BCSM will
       send release signals, otherwise it will just stop. */
  CSStop(CallSegmentID,Cause),  /* CS −> CSA, CS −> FIMAgent: indication that the CS has stopped. */
  SSFStop,                      /* CS −> SSF−FSM: instruction to stop the SSF−FSM. */
  SSF_PId (PId),                /* CS −> CSA: indication that a new SSF_FSM is created. */
  DeleteLeg(LegType),           /* CS −> SSF: indication that a BCSM has stopped. */
  ReleaseSSF,                   /* FA −> CS: request to release the SSF/CS when all armed EDPs are reported. */
  CSAStop(CSAID);               /* CSA −> SSME_C: indiaction that a CSA has stopped */

/*  Signals used for passing CSA and CS creation requests between the FIM, SSME_Control, CSA and CS */

SIGNAL
  NewCSReq (BCSMType, ServiceKey, PId),
  NewCSResp (CSAID, CallSegmentID, LegType);

SIGNALLIST FA_To_SSME_C =
  NewCSReq;
```

```
/* signals between CS and FIMAgent */
SIGNAL
FAAddBCSM (CSAID, /* CS */ PId, LegType, /* isControlling */ Boolean, BCSMType, BCSMPars),
/* Used in Connect/MoveLeg/SplitLeg. Creates 1 new BCSM. If isControlling is false, the new
   BCSM will be automatically added as a passive leg to the connection point. If isControlling is true,
   the BCSM will be added to the controlling */
FADelControllingBCSM (/* CS */ PId, LegType),
/* Used in transitions from CSCV state 1−Party. Deletes the BCSM attached to a controlling leg.
   The BCSM process will actually be stopped and deleted from internal data structures. */
FADiscardLeg (/* CS */ PId, LegType),
/* Used in MoveLeg/MergeCallSegments for discarding of pending legs in the target CS. Will
   remove the leg from the connection point and stop the BCSM connected to the leg. The BCSM
   chain will not yet be deleted (but this will probably happen when the release propagates through the
   chain, causing all BCSMs to stop). */
FAMoveControllingLeg (/* SourceCS */ PId, LegType, /* TargetCS */ Pid),
/* used in SplitLeg, MoveLeg, and MergeCallSegment operations */
FAMovePassiveLeg (/* SourceCS */ PId, LegType, /* TargetCS */ Pid),
/* used in SplitLeg, MoveLeg, and MergeCallSegment operations */
FASetLegToSurrogate (/* CS */ PId, LegType);
/* Will notify the BCP that the connection point\022s controlling leg has become surrogate. The BCP
   needs to know this for interworking. This signal should be sent from the SetLegStatus procedure in
   CS whenever the controlling leg becomes surrogate */
```

Block Type SSF_CCF 7(13)

```
/*******    REMOTE PROCEDURE DEFINITIONS BETWEEN FIM_Manager,
        FIM AGENTS AND BCSM CONTROLLER   *****/

/* BCSMStartChain is exported by BCP, and called by FIMAgent
 * It indicates that the FIMAgent will create a BCSM chain
 */
REMOTE PROCEDURE BCSMStartChain;

/* CreateBCSM is exported by BCP, and called by FIMAgent.
 * It will request the creation of 1 BCSM that should be linked into
 * the chain. It returns the PId of the created BCSM
 */
REMOTE PROCEDURE CreateBCSM;
 FPAR BCSMType, BCSMPars, PId;
 RETURNS PId;

/* BCSMChainCreated is exported by BCP, and called by FIMAgent
 * It indicates that the FIMAgent has finished with the creation of the BCSM chain.
 * The two parameters are the PIds of the first and last BCSM in the chain.
 */
REMOTE PROCEDURE BCSMChainCreated;
 FPAR PId, PId;

/* RemoveBCSM is exported by BCP, and called by FIMAgent.
 * This requests the BCP to remove all processes in the chain.
 * The PId parameter points to the first BCSM in the chain.
 */
REMOTE PROCEDURE RemoveBCSMChain;
 FPAR PId;

/* SetFIMAgent is exported by O/T_BCSM and called by a FIMAgent.
 * It lets the BCSM know that it is handled by another FIMAgent.
 */
REMOTE PROCEDURE SetFIMAgent;
 FPAR PId;

/* CreateFIMAgent is exported by FIM_Controller, and called by BCP and CS.
 * It requests the creation of a new FIMAgent + a chain of BCSMs.
 * Returns the PId of the created FIMAgent.
 */
REMOTE PROCEDURE CreateFIMAgent;
 FPAR BCSMType, BCSMPars, CSAID, PId, LegType;
 RETURNS PId;

/* AddNewCS is exported by FIMAgent and called by FIM_Controller.
 * It is called when CS wants to create a new FIMAgent, but CS's CSA
 * that is already handled by some FIMAgent.
 * In this case, the existing FIMAgent is reused.
 * The call of this RPC will result in a new BCSM chain being created.
 */
REMOTE PROCEDURE AddNewCS;
 FPAR BCSMType, BCSMPars, CSAID, PId, LegType;

/* UpdateFAInfo is exported by FIM_Controller and called by FIMAgent
 * to tell FIM_Controller that it takes care of the given CSA
 */
REMOTE PROCEDURE UpdateFAInfo;
 FPAR CSAID;

/* WhichFIMAgent is exported by FIM_Controller and called by FIMAgent
 * to ask which FIMAgent takes care of the given CSA
 */
REMOTE PROCEDURE WhichFIMAgent;
  FPAR CSAID;
  RETURNS PId;
```

Block Type SSF_CCF 8(13)

```
/*******   REMOTE PROCEDURE DEFINITIONS BETWEEN
          FIM AGENTS AND BCSM CONTROLLER   *****/

/* all below remote procedures are exported by BCP and
   called by FIMAgent */

/* creates a new BCSM attached to a controlling leg.
   If the PId param is not null, the bcsm is created
   in the same connection point (CS) as the specified BCSM */
REMOTE PROCEDURE CreateControllingBCSM;
 FPAR BCSMType, BCSMPars, PId;
 RETURNS Pid;

/* creates a new passive leg with a BCSM in the same
   connection point as the specified BCSM. */
REMOTE PROCEDURE AddPassiveBCSM;
 FPAR BCSMType, BCSMPars, PId;
 RETURNS Pid;

/* cuts the chain in two. Used in middle−release */
REMOTE PROCEDURE CutBCSMChain;
 FPAR Pid;

/* moves a controlling leg to another connection point */
REMOTE PROCEDURE MoveControllingBCSMChain;
 FPAR/* some BCSM in source CP */ PId, /* target BCSM */ PId;

/* moves a passive leg to another connection point */
REMOTE PROCEDURE MovePassiveBCSMChain;
 FPAR /* source BCSM */ PId, /* target BCSM */ PId;

/* deletes all information about the BCSM */
REMOTE PROCEDURE RemoveBCSM;
 FPAR PId;
```

```
/*******   REMOTE PROCEDURE DEFINITIONS
          BETWEEN CSA AND CS   *****/

/* all remote procedures below are exported by CS and
   called by CSA */

/* returns the PId of the associated FIM Agent */
REMOTE PROCEDURE GetFAPtr;
 RETURNS Pid;

/* is the MergeCallSegments operation applicable if the
   called CS is the source CS */
REMOTE PROCEDURE ValidSourceCSforMCS;
 RETURNS Boolean;

/* is the MoveLeg operation applicable if the
   called CS is the source CS */
REMOTE PROCEDURE ValidSourceCSforML;
 FPAR MoveLegArg;
 RETURNS Boolean;

/* is the MoveLeg operation applicable if the
   called CS is the target CS */
REMOTE PROCEDURE ValidTargetCSforMLMCS;
 RETURNS Boolean;

/* is the SplitLeg operation applicable if the
   called CS is the source CS */
REMOTE PROCEDURE ValidSourceCSforSL;
 FPAR SplitLegArg;
 RETURNS Boolean;

/* Delete all pending legs in CS (called in Source CS
before performing MoveCallSegments) */
REMOTE PROCEDURE DeletePendingLeg;

/* RemoveLegCSA is exported by the CSA, and permits
CS to update CSA whenever
it removes a leg, and that removal is not visible to CSA */
REMOTE PROCEDURE RemoveLegCSA;
FPAR LegType;

/* ExportEventRecord is exported by SSF,
and permits CS to update SSF
whenever it removes a leg and that
removal is not visible to SSF */
REMOTE PROCEDURE ExportEventRecord;
 FPAR IN LegType, IN/OUT EventRecordType, IN/OUT Integer;
```

Block Type SSF_CCF 9(13)

```
/**** DATA TYPE DEFINITIONS, SIGNALS, AND REMOTE PROCEDURES
     COMMON TO THE SSME_C, 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,Integer),
  ImportLegReq(LegType,EventRecordType,Integer),
  ImportLegResp,

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

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

  /* Signal to change the value of the CallSegmentId in the CSs, when the CS has
     been moved to another CSA. */
  SetCSId(CallSegmentId),

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

  /* Signal to change the value of the FA pointer in the CS when a new
     CS has been created by a CSA due to a SplitLeg, or the CS is moved to a
     new CSA due to a MoveCallSegments. */
  SetFA(PId),

  /* Signals for importing and exporting call segments between SSME_C 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),
  DPFacility(DPFacilityArg);
```

```
/* Remote procedure exported by SSME_C
 * and called by SSF_FSM
 */

REMOTE PROCEDURE CheckMatchingServiceFilteringCrit
FPAR
   IN DPArg,
   IN ServiceKey,
   IN/OUT FilteringCriteria;
RETURNS Boolean;
*/
REMOTE PROCEDURE CheckACG;
FPAR
   IN DPArg,
   IN ServiceKey,
   IN/OUT GapCriteria;
RETURNS Boolean;

REMOTE PROCEDURE CheckCallFiltered;
FPAR IN FilteringCriteria;
RETURNS Boolean;

REMOTE PROCEDURE SendSFR;
FPAR IN FilteringCriteria,
        IN Boolean;

REMOTE PROCEDURE CountCall;
FPAR
 IN DPArg,
 IN FilteringCriteria;

REMOTE PROCEDURE SFGetCause;
FPAR IN FilteringCriteria;
RETURNS Cause;

REMOTE PROCEDURE SFInfoToSend;
FPAR IN FilteringCriteria;
RETURNS InformationToSend;

REMOTE PROCEDURE CGGetCause;
FPAR IN GapCriteria;
RETURNS Cause;

REMOTE PROCEDURE CGInfoToSend;
FPAR IN GapCriteria;
RETURNS InformationToSend;

/* GetMasterTDPTable is exported by SSME_C,
 * and called by FIMAgent to obtain the "master" TDP table.
 */

REMOTE PROCEDURE GetMasterTDPTable;
 RETURNS TDPTableType;

/* Remote procedure exported by O−BCSM, T−BCSM,
CS and FA. Called by SSF_FSM */
REMOTE PROCEDURE SetNoAnswerTime
; FPAR LegType, Integer;
```

Block Type SSF_CCF 10(13)

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

/* INAP OPERATIONS ON THE CSA/SSME_C INTERFACE */

```
SIGNALLIST CS1_INAP_CSA_In =
  ApplyCharging,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continnue,
  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 CS3_INAP_CSA_IN =
  (CS1_INAP_CSA_IN),
  ActivityTestResult,
  ContinueWithArgument,
  CreateCallSegmentAssociation,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  MergeCallSegments,
  MoveLeg,
  RequestReportUTSI,
  SendSTUI,
  SplitLeg,
  SelectFacility,
  SetServiceProfile,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun;

SIGNALLIST CS3_INAP_CSA_OUT =
  (CS1_INAP_CSA_OUT),
  ActivityTest,
  CreateCallSegmentAssociationResult,
  DisconnectLegResult,
  EntityReleased,
  MergeCallSegmentsResult,
  MoveLegResult,
  ReportUTSI,
  SplitLegResult,

  ScriptEvent,
  PromptAndReceiveMessageResult;
```

Block Type SSF_CCF 11(13)

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

/*  INAP OPERATIONS ON THE CS/CSA INTERFACE */
```

```
SIGNALLIST CS1_INAP_CS_In =
  ApplyCharging,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continnue,
  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 CS3_INAP_CS_IN =
  (CS1_INAP_CS_IN),
  ContinueWithArgument,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  RequestReportUTSI,
  SelectFacility,
  SendSTUI,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun,

  SetServiceProfile;

SIGNALLIST CS3_INAP_CS_OUT =
  (CS1_INAP_CS_OUT),
  DisconnectLegResult,
  EntityReleased,
  ReportUTSI,

  ScriptEvent,
  PromptAndReceiveMessageResult;
```

Block Type SSF_CCF 12(13)

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

/*  INAP OPERATIONS ON THE CS/SSF_FSM INTERFACE */
```

```
SIGNALLIST CS1_INAP_SSF_In =
  ApplyCharging,
  CallInformationRequest,
  Cancel,
  CollectInformation,
  Connnect,
  ConnectToResource,
  Continnue,
  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 CS3_INAP_SSF_IN =
  (CS1_INAP_SSF_IN),
  ContinueWithArgument,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  RequestReportUTSI,
  SelectFacility,
  SendSTUI,

  PromptAndReceiveMessage,
  ScriptClose,
  ScriptInformation,
  ScriptRun;

SIGNALLIST CS3_INAP_SSF_OUT =
  (CS1_INAP_SSF_OUT),
  DisconnectLegResult,
  EntityReleased,
  ReportUTSI,

  ScriptEvent,
  PromptAndReceiveMessageResult;
```

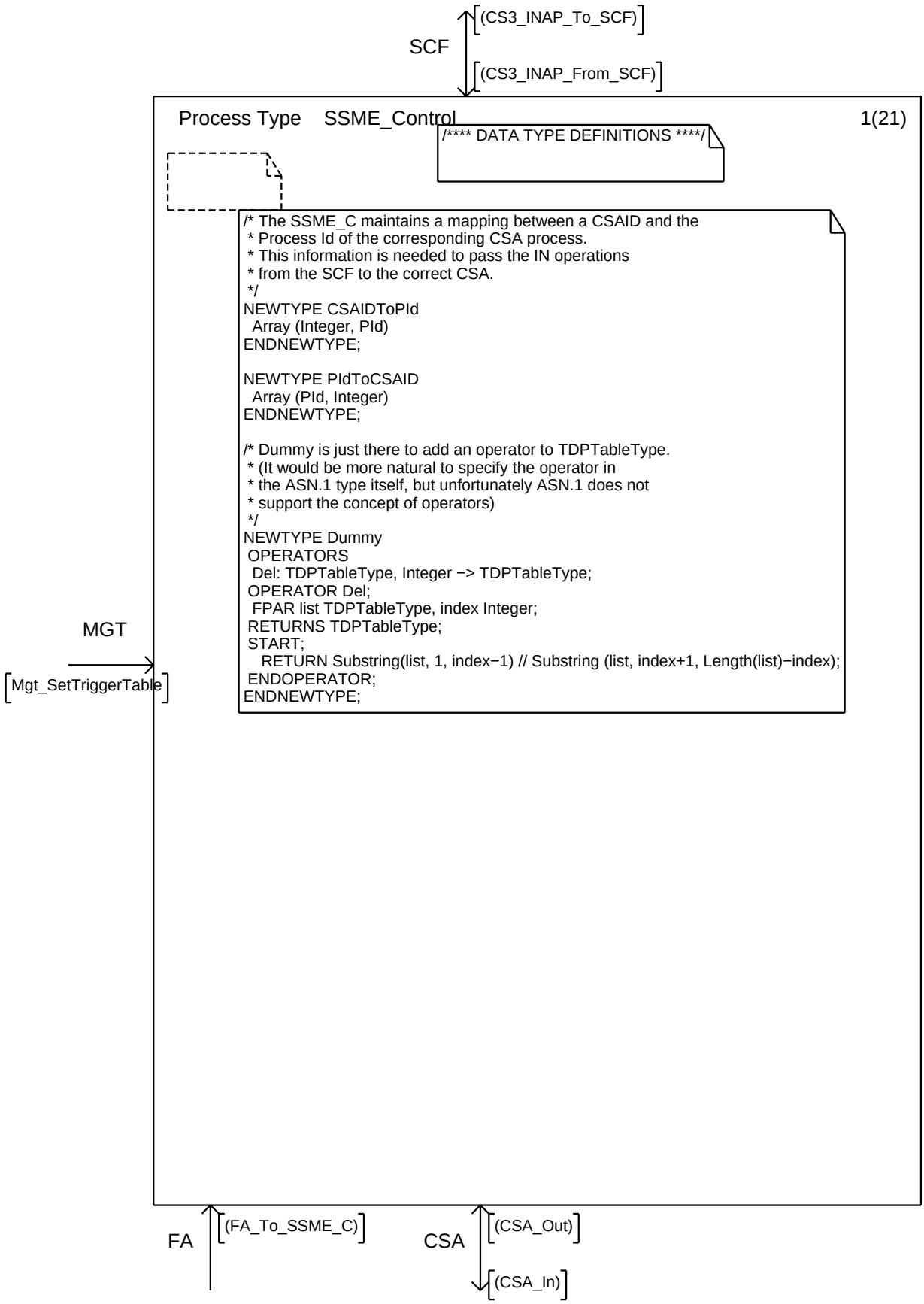
| Block Type SSF_CCF |13(13)

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

```
SIGNALLIST CSA_In =
  (CS3_INAP_CSA_In),
  ExportCSReq,
  ImportCSReq,
  NewCSReq;

SIGNALLIST CSA_Out =
  (CS3_INAP_CSA_Out),
  ExportCSResp,
  ImportCSResp,
  NewCSReq,
  CSAStop;

SIGNALLIST CS_In =
  (CS3_INAP_CS_In),
  CSStop,
  CSExportLegReq,
  CSImportLegReq,
  DataReqInd,
  DataInd,
  FAMoveControllingLeg,
  FADiscardLeg,
  FAMovePassiveLeg,
  SetCSA,
  SetCSId,
  SetFA,
  SetLegID,
  NewCSReq,
  DPUTSI;

SIGNALLIST CS_Out =
  (CS3_INAP_CS_Out),
  BCSMException,
  CSStop,
  CSExportLegResp,
  CSImportLegResp,
  NewCSResp,
  SSF_PId;

SIGNALLIST CS3_SSF_In =
  (DPs),
  (CS3_INAP_SSF_In),
  DeleteLeg,
  SSFStop,
  ExportLegReq,
  ImportLegReq,
  SetLegID,
  DPUTSI,
  ReleaseSSF;

SIGNALLIST CS3_SSF_Out =
  (PICs),
  (CS3_INAP_SSF_Out),
  ExportLegResp,
  ImportLegResp;
```

```
SIGNALLIST O_BCSM_In =
  (oPICs),
  (O_SigCon_In),
  (IBI_In),
  StopBCSM,
  NetworkSuspendReqInd,
  NetworkResumeReqInd,
  DataReqInd,
  DataInd,
  SetCS;

SIGNALLIST O_BCSM_Out =
  (oDPs),
  (O_SigCon_Out),
  (IBI_Out),
  BCSMStop,
  NetworkSuspendReq,
  NetworkResumeReq,
  DataReqInd,
  DataReq,
  DPUTSI;

SIGNALLIST T_BCSM_In =
  (tPICs),
  (T_SigCon_In),
  (IBI_Out),
  StopBCSM,
  NetworkSuspendInd,
  NetworkResumeInd,
  DataInd,
  DataReqInd,
  SetCS;

SIGNALLIST T_BCSM_Out =
  (tDPs),
  (T_SigCon_Out),
  (IBI_In),
  BCSMStop,
  NetworkSuspendReqInd,
  NetworkResumeReqInd,
  DataReq,
  DataReqInd,
  DPUTSI;
```

```
/* signals CS −> FIMAgent */
SIGNALLIST CS_To_FA =
  (PICs),
  CSStop,
  FAAddBCSM,
  FADelControllingBCSM,
  FADiscardLeg,
  FAMoveControllingLeg,
  FAMovePassiveLeg,
  FASetLegToSurrogate,
  NewCSResp,
  SetCSA,
  SetLegID,
  SetServiceProfile;

/* signals FIMAgent −> CS */
SIGNALLIST FA_To_CS =
  (DPs),
  BCSMStopped,
  SetFA,
  ReleaseSSF;

/* signallists FA <−> O− and T−BCSM */
SIGNALLIST FA_To_BCSM =
  (PICs),
  StopBCSM,
  DPUTSI;

SIGNALLIST BCSM_To_FA =
  (DPs),
  BCSMStop,
  DPUTSI;
```


SCF
(CS3_INAP_To_SCF)
(CS3_INAP_From_SCF)
Process Type SSME_Control
1(21)
/**** DATA TYPE DEFINITIONS ****/
/* The SSME_C maintains a mapping between a CSAID and the
* Process Id of the corresponding CSA process.
* This information is needed to pass the IN operations
* from the SCF to the correct CSA.
*/
NEWTYPE CSAIDToPId
Array (Integer, PId)
ENDNEWTYPE;
NEWTYPE PIdToCSAID
Array (PId, Integer)
ENDNEWTYPE;
/* Dummy is just there to add an operator to TDPTableType.
* (It would be more natural to specify the operator in
* the ASN.1 type itself, but unfortunately ASN.1 does not
* support the concept of operators)
*/
NEWTYPE Dummy
OPERATORS
Del: TDPTableType, Integer −> TDPTableType;
OPERATOR Del;
FPAR list TDPTableType, index Integer;
RETURNS TDPTableType;
START;
RETURN Substring(list, 1, index−1) // Substring (list, index+1, Length(list)−index);
ENDOPERATOR;
ENDNEWTYPE;
MGT
Mgt_SetTriggerTable
FA
(FA_To_SSME_C)
CSA
(CSA_Out)
(CSA_In)

Process Type SSME_Control 2(21)

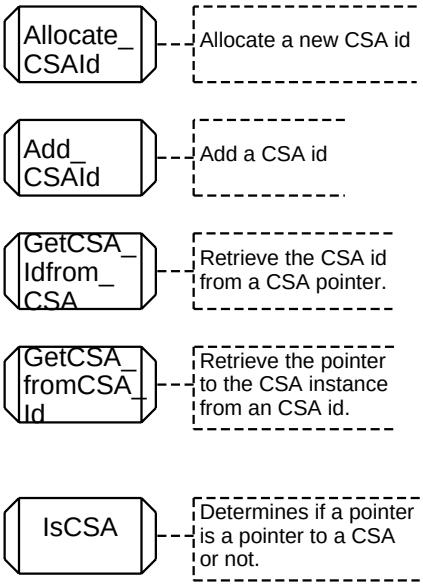
```
/**** VARIABLE DECLARATIONS ****/
```

```
DCL
  /* Mapping CSAID <−> process id of corresponding CSA. */
  CSAIDToCSA CSAIDToPId,
  CSAToCSAID PIdToCSAID,
  /* global variable that keeps track
     of the highest CSAID */
  HighestCSAIDInUse Integer,

  /* "master" TDP Table */
  TDPTable TDPTableType,

  /* highest assigned TDP identifier,
     used in entries in TDP table */
  HighestTDPIdentifier Integer,

  /* Other variables */
  csa PId,
  csaID, newCSAID CSAID,
  invokeID InvokeID,
  i Integer,
  termination Boolean,
  csID CallSegmentID,
  halfCallSide SSME_CRoleType,
  dialogID DialogIDtype,
  r Boolean,
  dID DialogIDtype,
  FA PId,
  serviceKey ServiceKey,
  bcsm BCSMType,
  mgt_STTArg MGTSetTriggerTableArg;
```

```
DCL
  /* IN 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,
  mrArg MessageReceivedArg,
  paArg PlayAnnouncementArg,
  pacuiArg PromptAndCollectUserInformationArg,
  parmArg PromptAndReceiveMessageArg,
  riArg ReceivedInformationArg,
  rcArg ReleaseCallArg,
  rnceARg RequestNotificationChargingEventArg,
  rrBCSMEArg RequestReportBCSMEventArg,
  rtARg ResetTimerArg,
  scArg ScriptCloseArg,
  seArg ScriptEventArg,
  sfArg SelectFacilityArg,
  siArg ScriptInformationArg,
  scrArg ScriptRunArg,
  sciArg SendChargingInformationArg,
  sfrArg ServiceFilteringResponseArg,
  srrArg SpecializedResourceReportArg,
  srepArg StatusReportArg,

  ieArg ErrorArg;

DCL
  /* IN CS−3 operation arguments. */
  ccsaArg CreateCallSegmentAssociationArg,
  ccsarArg CreateCallSegmentAssociationResultArg,
  csrArg CancelStatusReportRequestArg,
  ctwaArg ContinueWithArgumentArg,
  ctdArg CreateOrRemoveTriggerDataArg,
  ctdrArg CreateOrRemoveTriggerDataResultArg,
  dfcwaArg DisconnectForwardConnectionWithArgumentArg,
  dlArg DisconnectLegArg,
  erArg EntityReleasedArg,
  mtdArg ManageTriggerDataArg,
  mtdrArg ManageTriggerDataResultArg,
  mcsArg MergeCallSegmentsArg,
  mocsArg MoveCallSegmentsArg,
  mlArg MoveLegArg,
  rcsArg RequestCurrentStatusReportArg,
  rcsrArg RequestCurrentStatusReportResultArg,
  resArg RequestEveryStatusChangeReportArg,
  rfsArg RequestFirstStatusMatchReportArg,
  rutsiArg ReportUTSIArg,
  rrutsiArg RequestReportUTSIArg,
  srpArg StatusReportArg,
  sstuiArg SendSTUIArg,
  sspArg SetServiceProfileArg,
  slArg SplitLegArg;
```

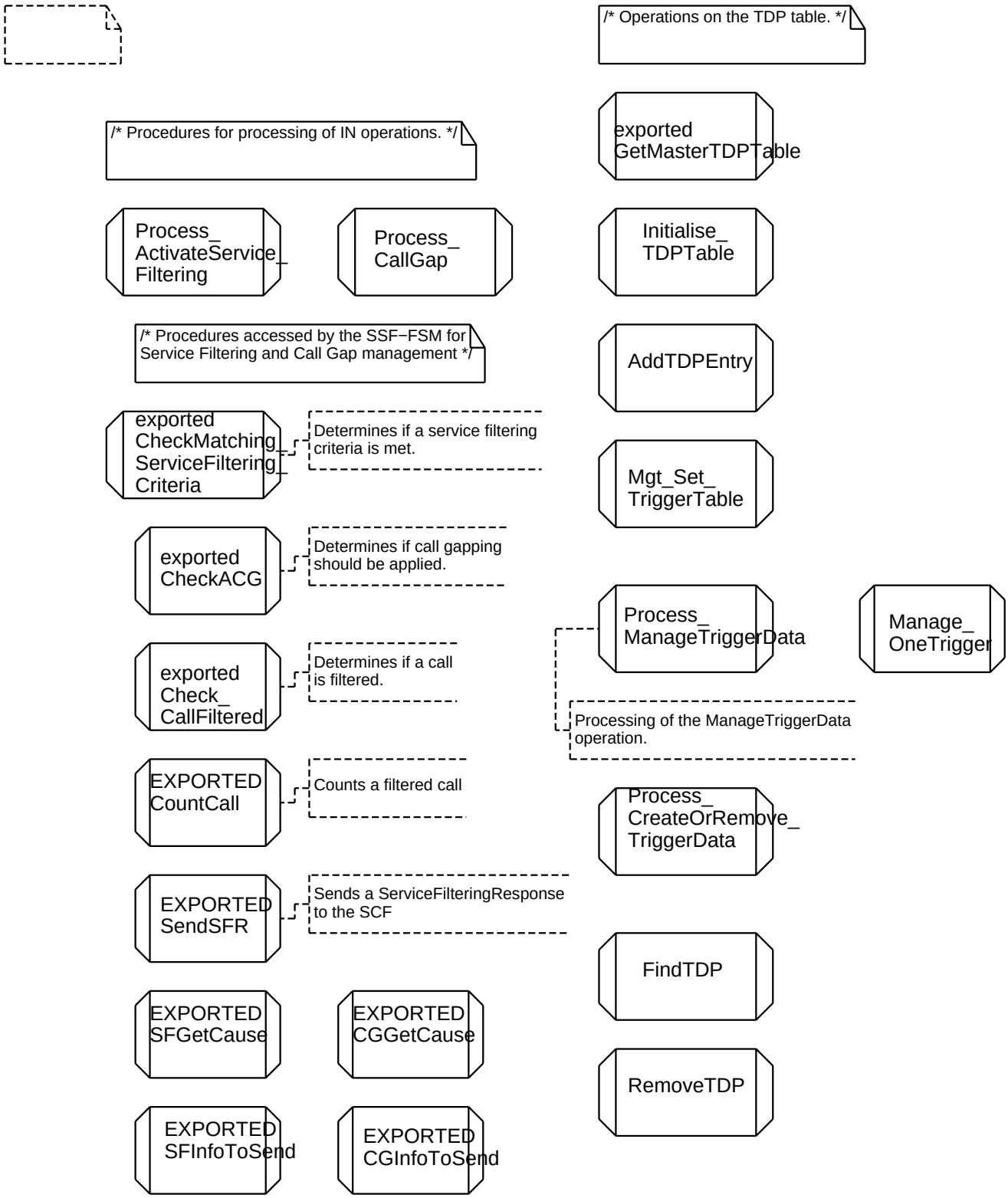
Process Type SSME_Control 3(21)

/* DECLARATION OF OPERATIONS ON THE ROUTING TABLES */

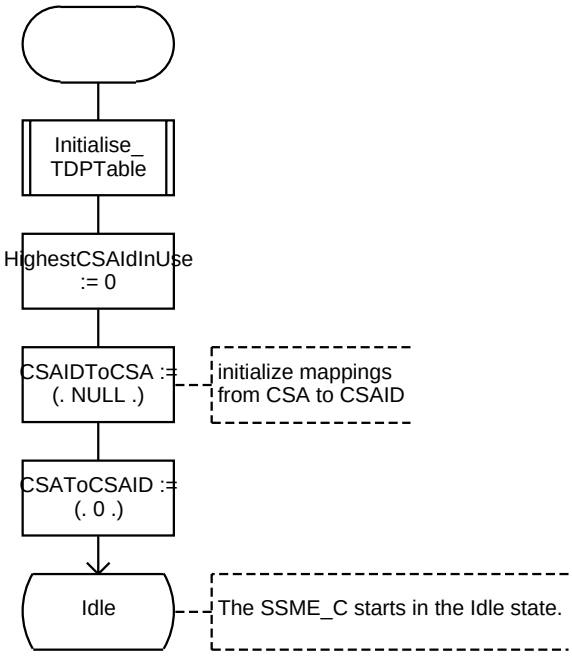


/* Procedure for processing of the operation MoveCallSegments. */

Process Type SSME_Control 4(21)



Process Type SSME_Control 5(21)



Process Type SSME_Control 6(21)

/* Handling of the opening and closing of dialouges with the TCAP. */

Idle

Application_
Begin
(csaID)

call
IsCSA(sender)

true
/* From CSA */

false
/* 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)

true
/* From CSA */

false
/* From SCF */

csa := call
GetCSAfrom_
CSAId(csaID)

Retrieve the CSA ID.

csaID := call
GetCSAIdfrom_
CSA(sender)

Retrieve the CSA ID.

Application_
End(termination,csaID)

to csa

/* Pass on to CSA. */

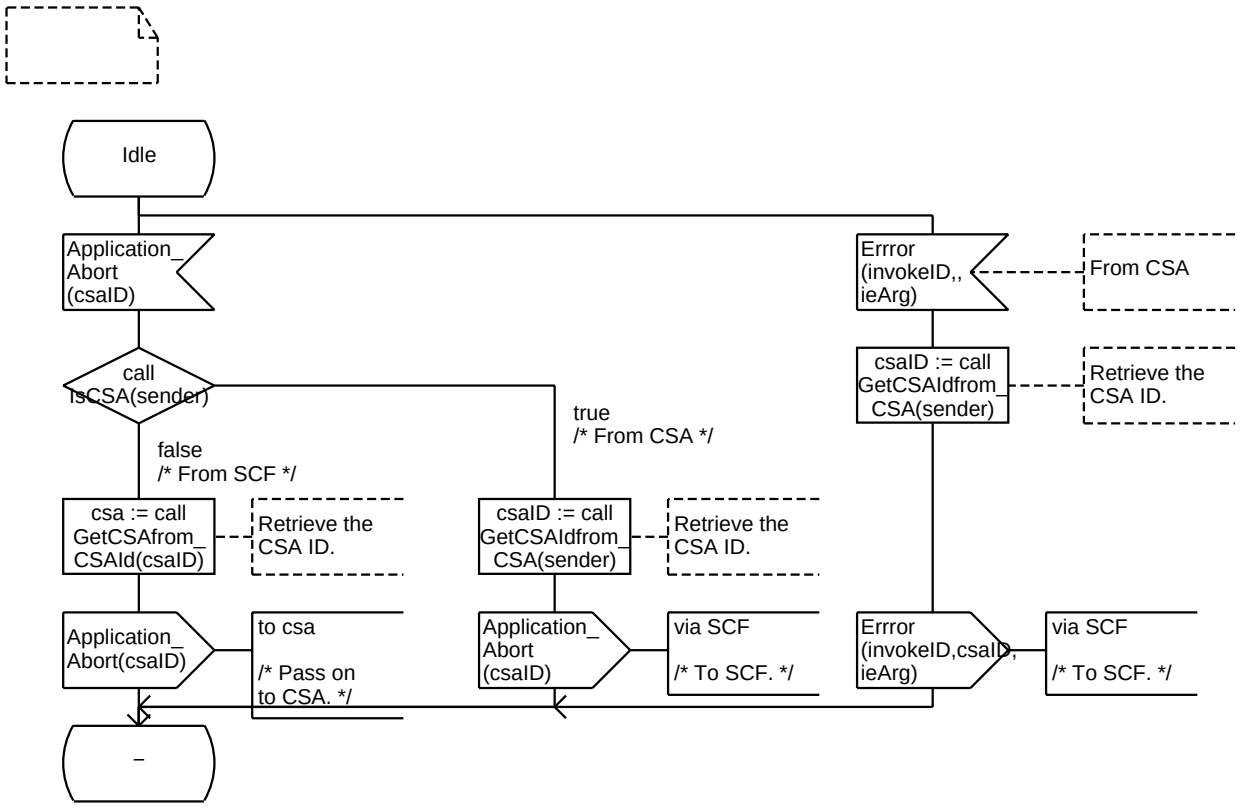
Application_
End(termination,csaID)

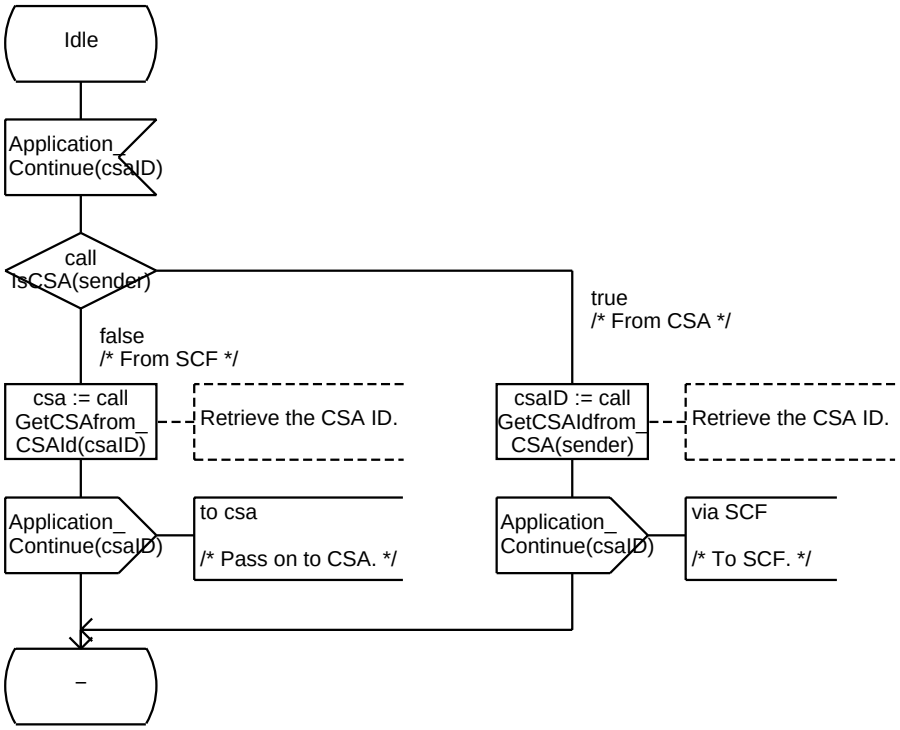
via SCF

/* To SCF. */

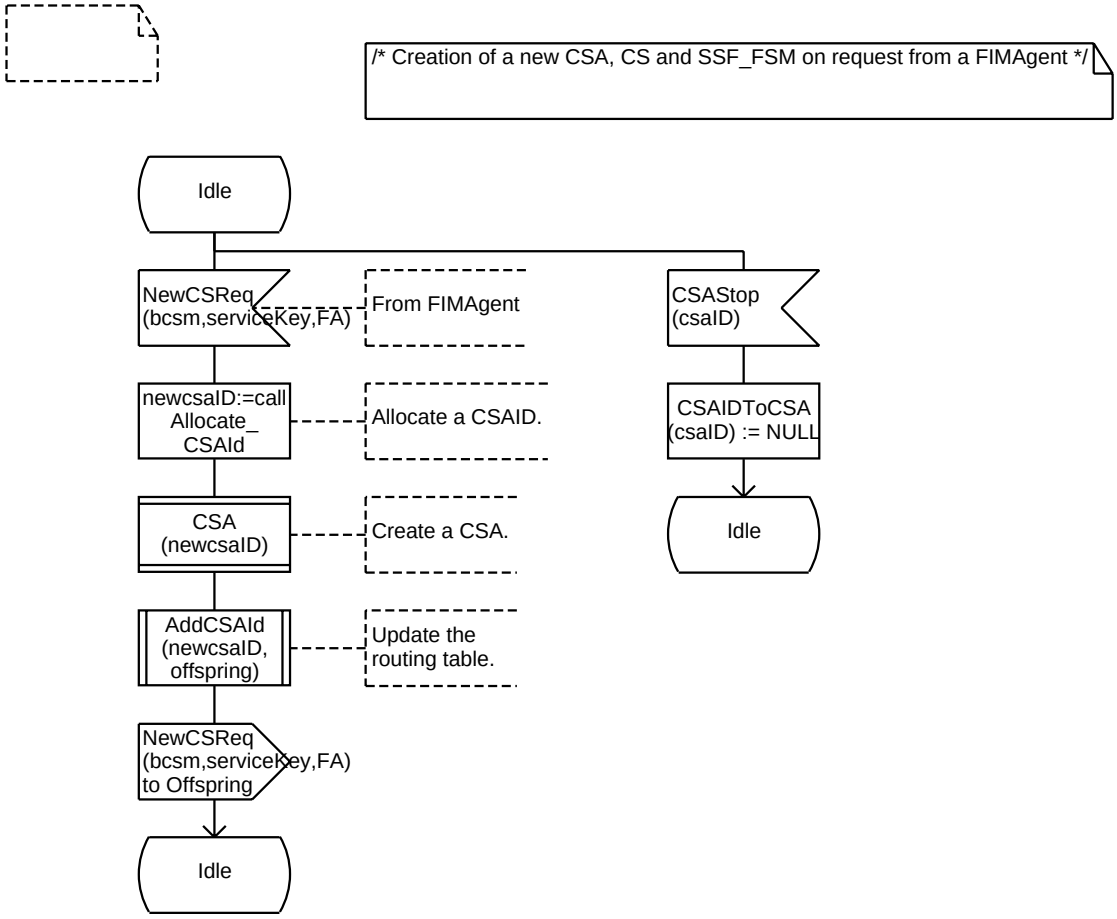
–

Process Type SSME_Control 7(21)

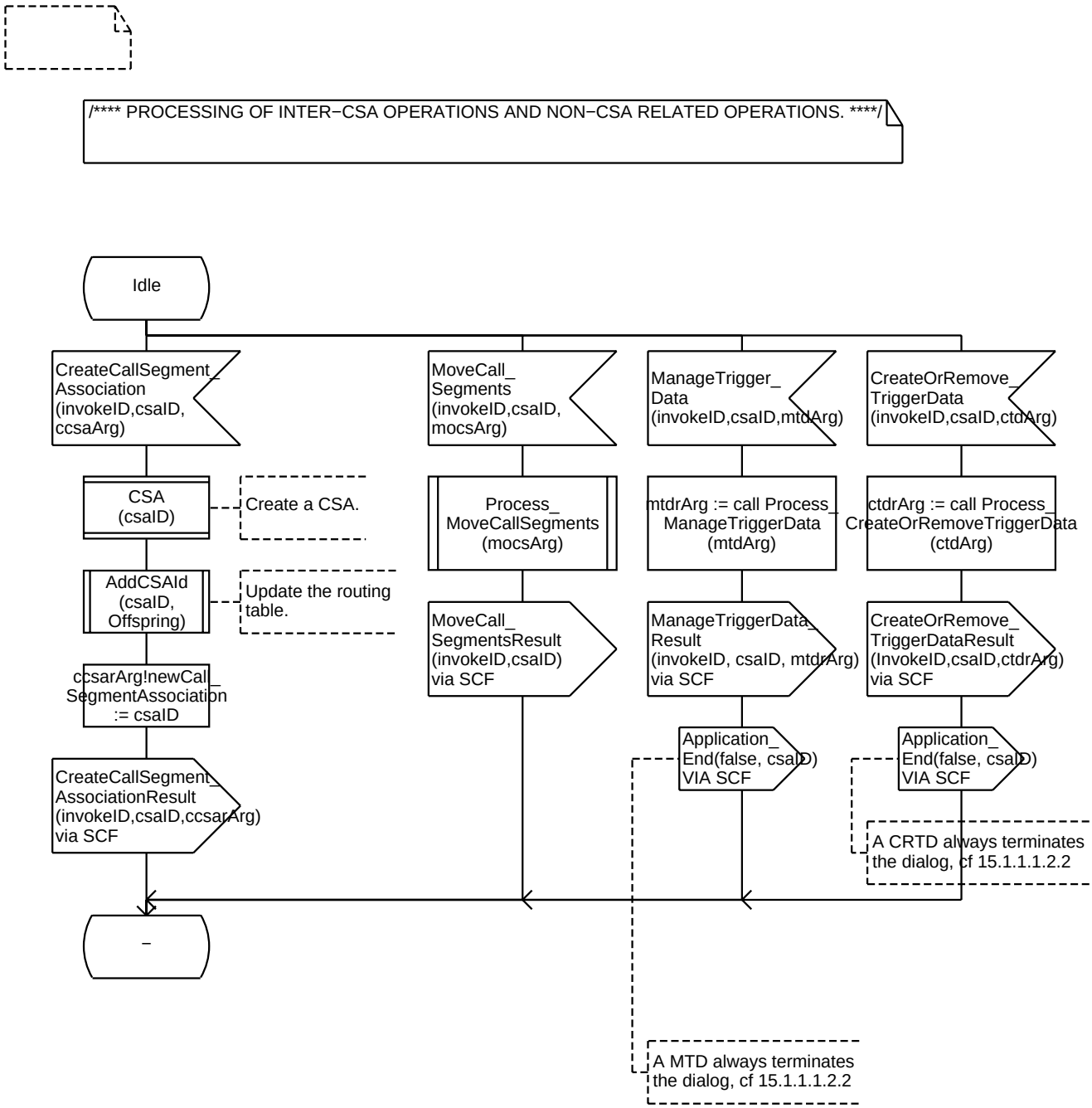




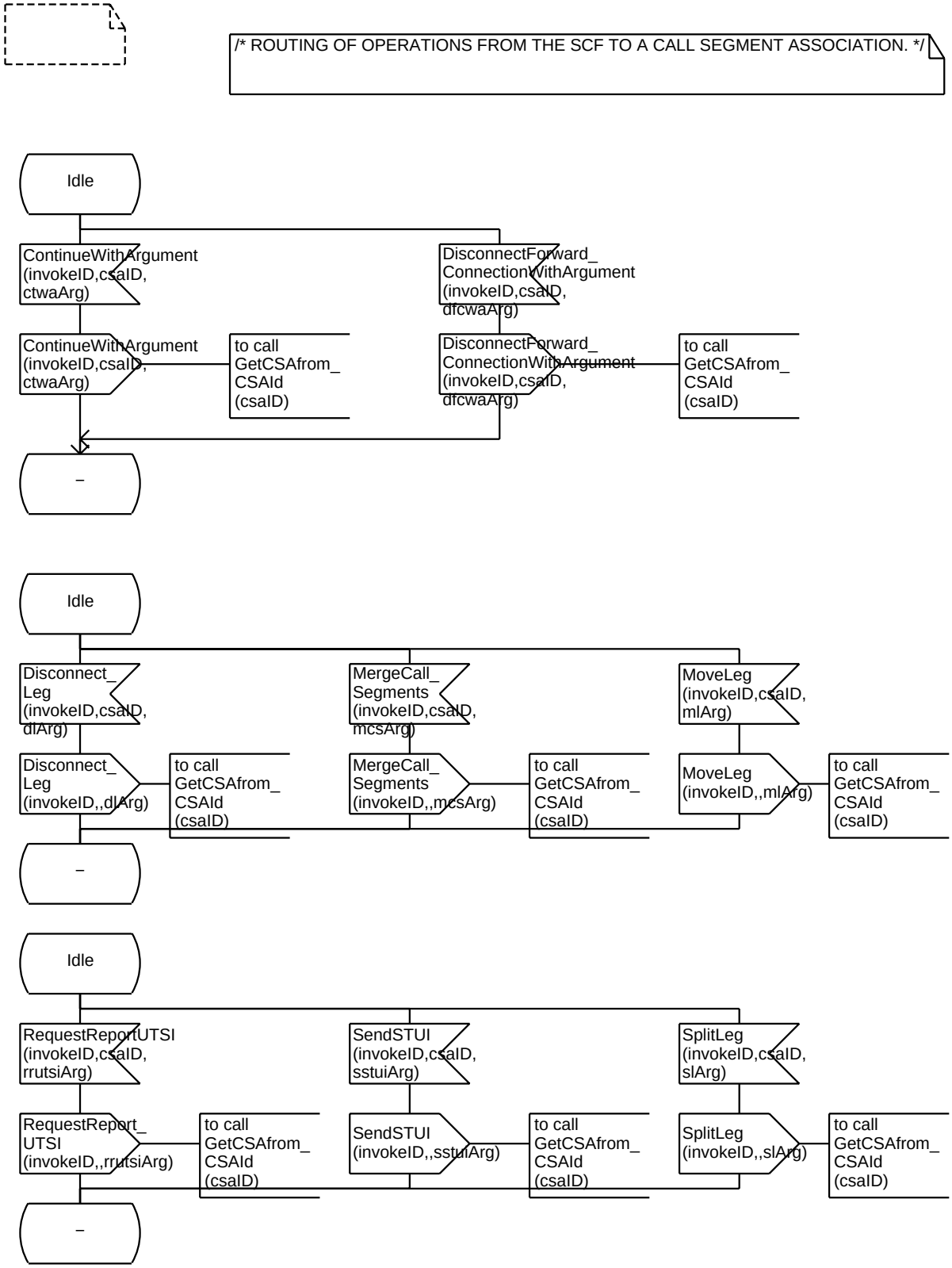
Process Type SSME_Control 8(21)



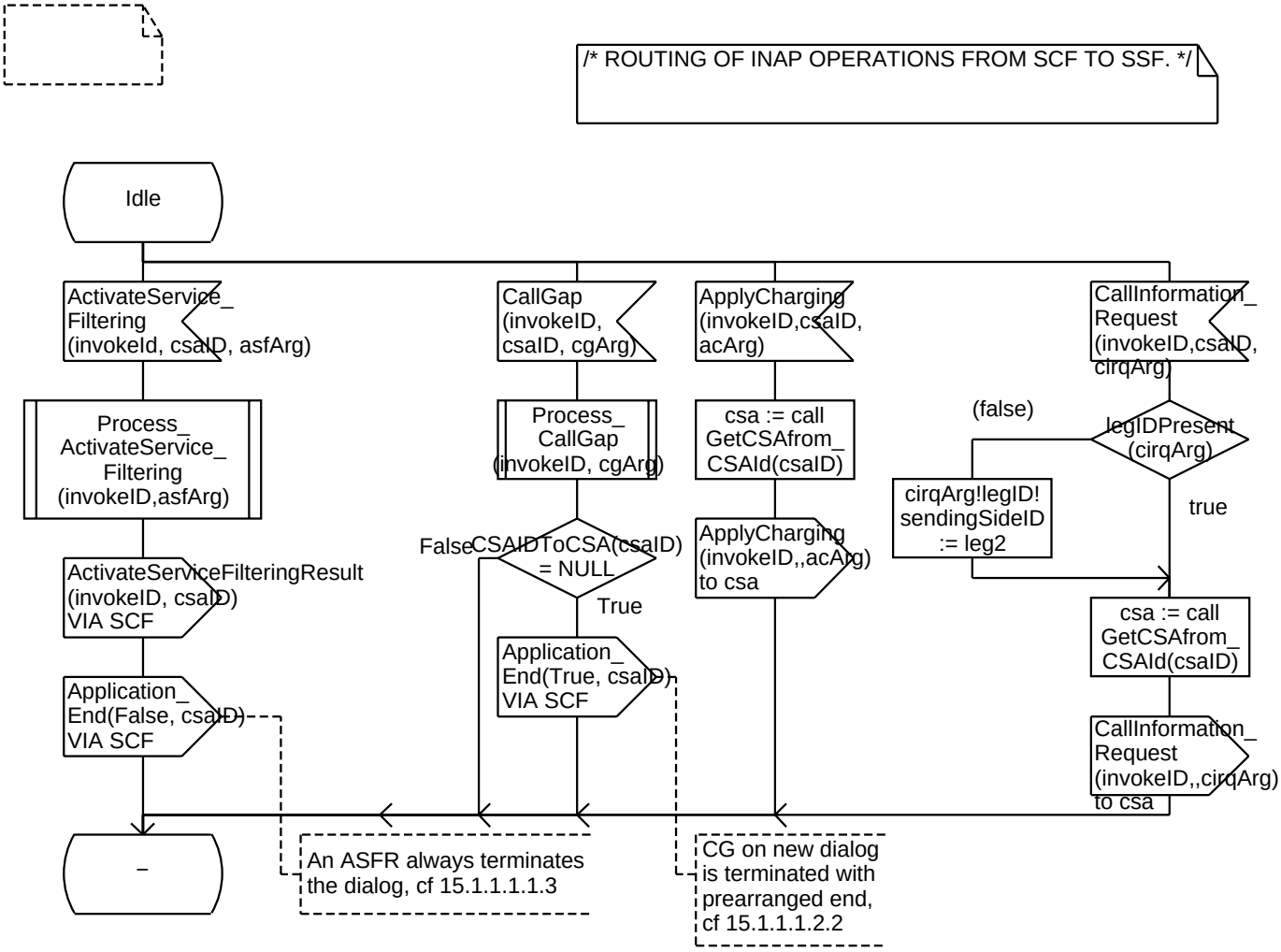
Process Type SSME_Control 9(21)

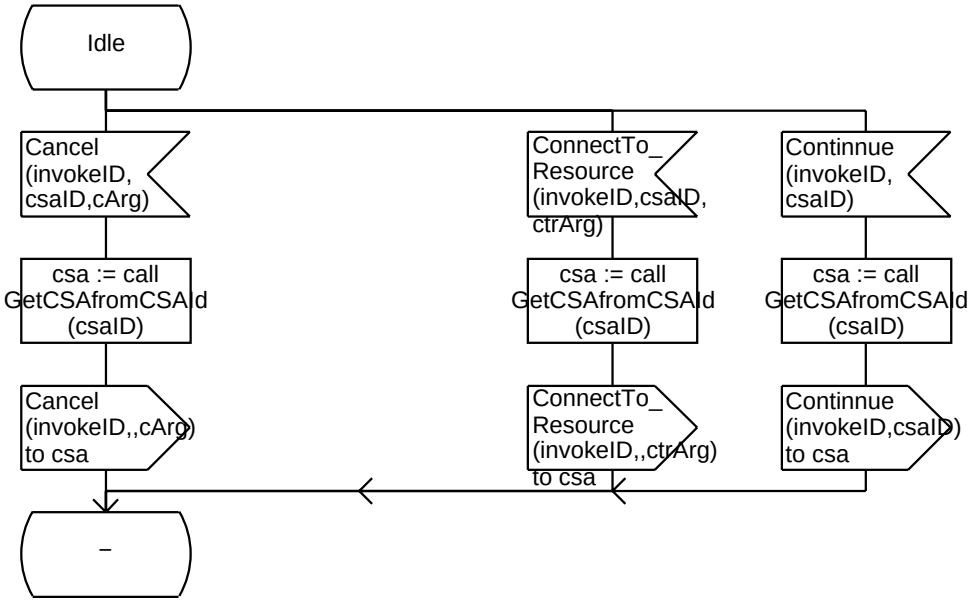


Process Type SSME_Control 10(21)

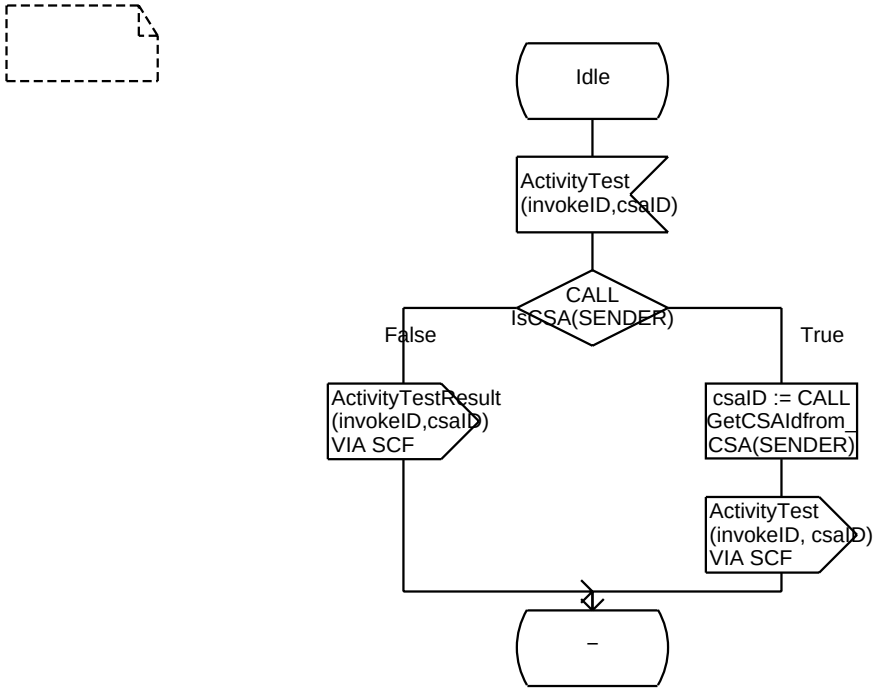


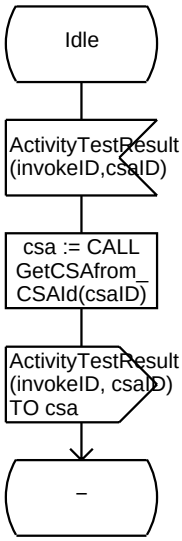
Process Type SSME_Control 11(21)



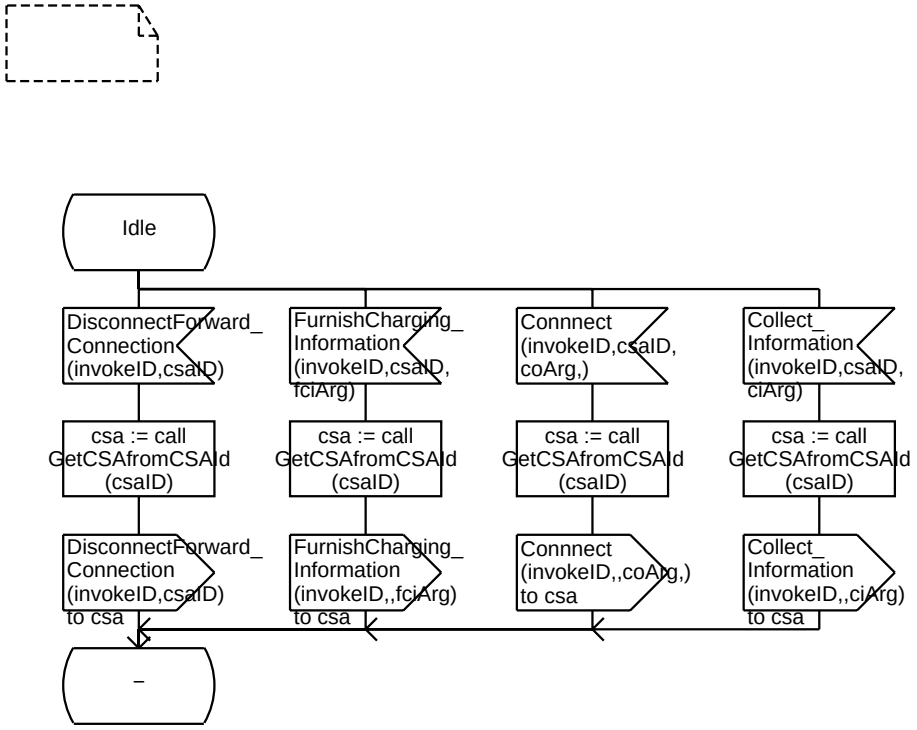


Process Type SSME_Control 12(21)

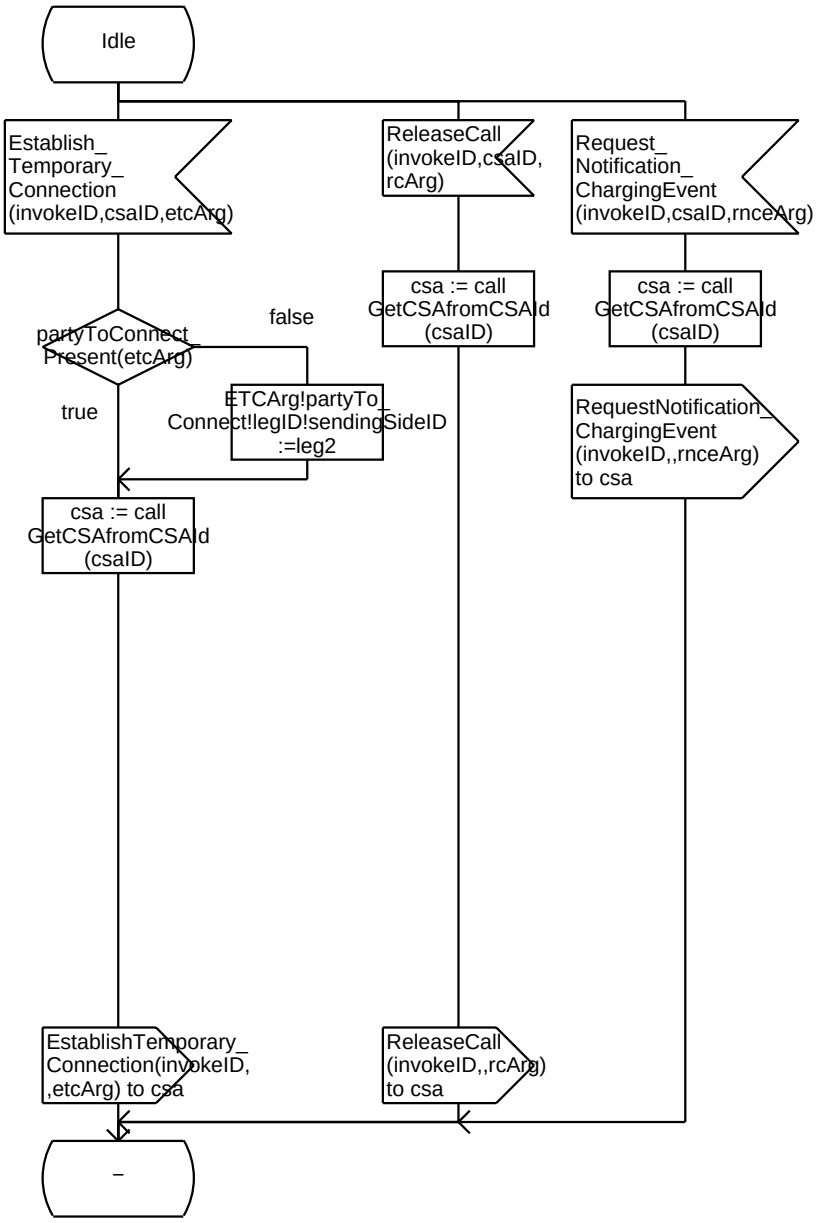




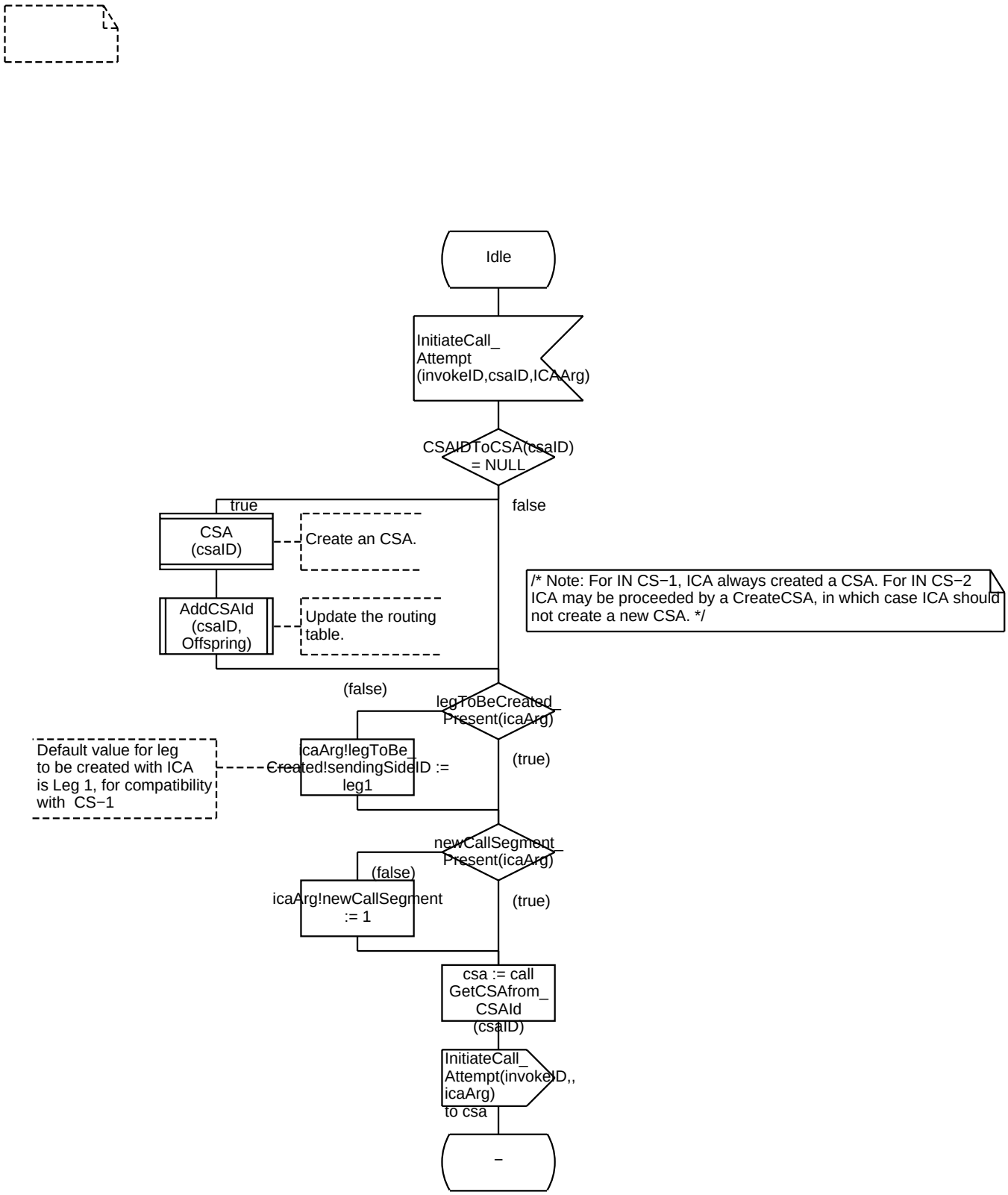
Process Type SSME_Control 13(21)

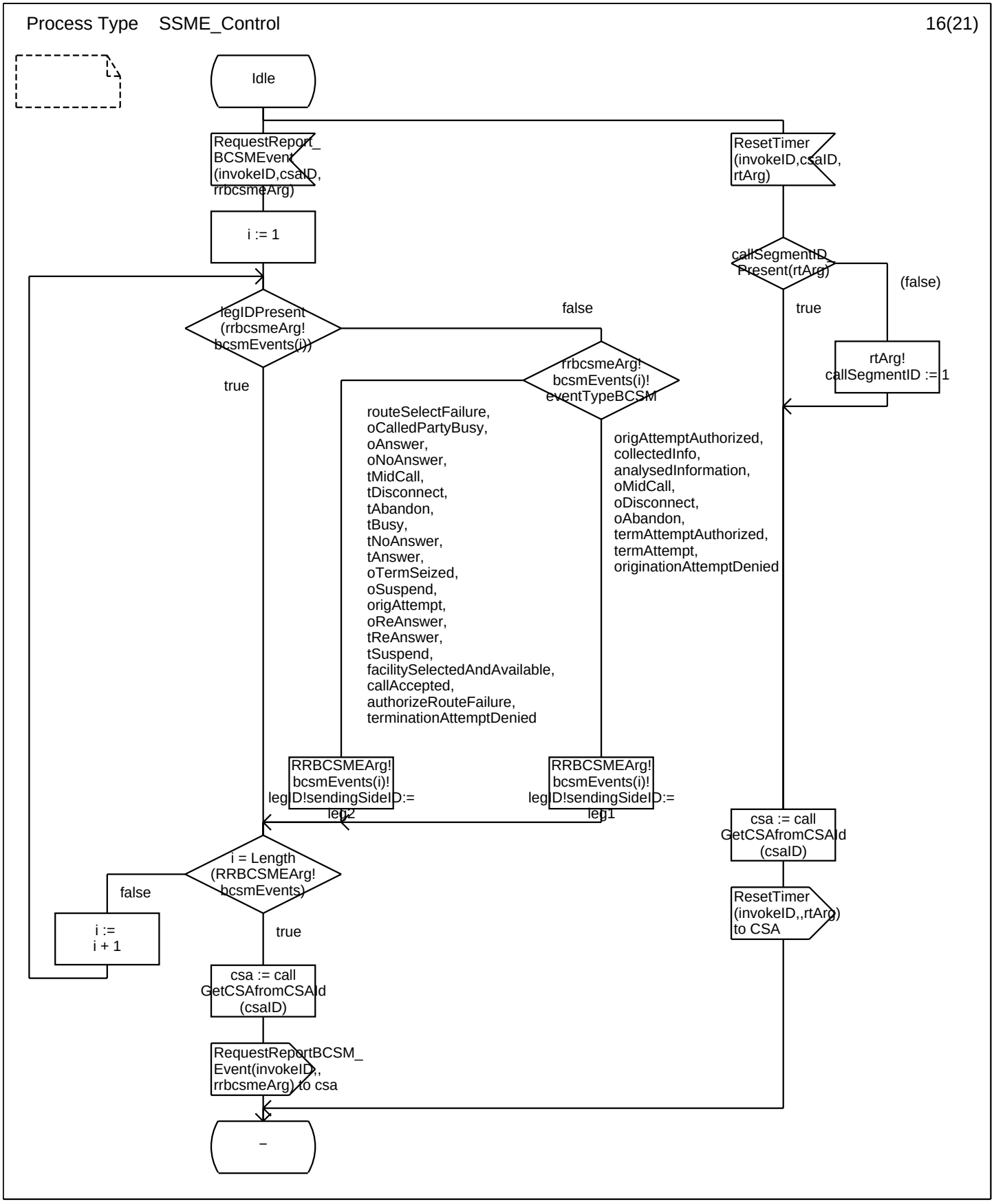


Process Type SSME_Control 14(21)

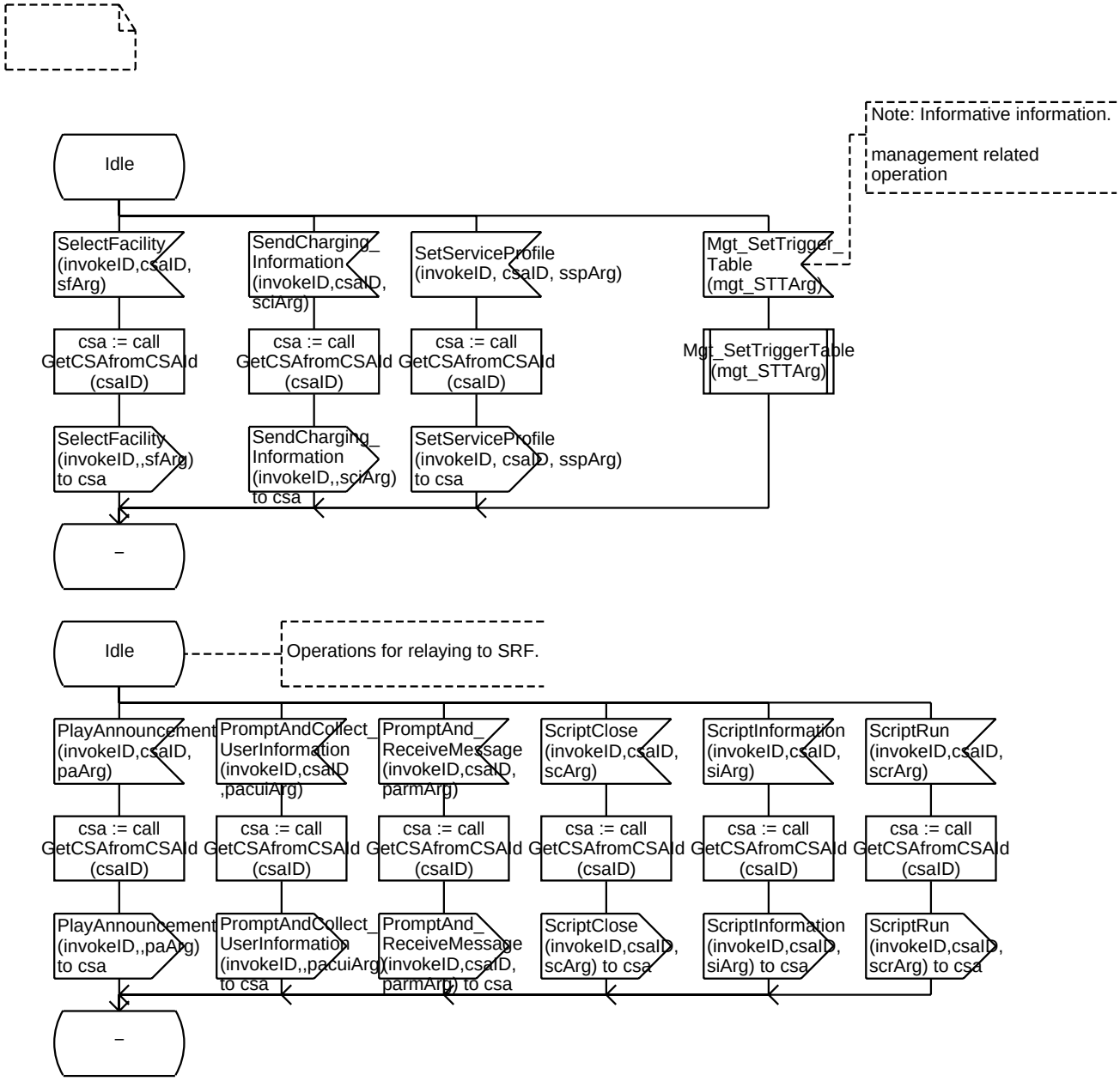


Process Type SSME_Control 15(21)

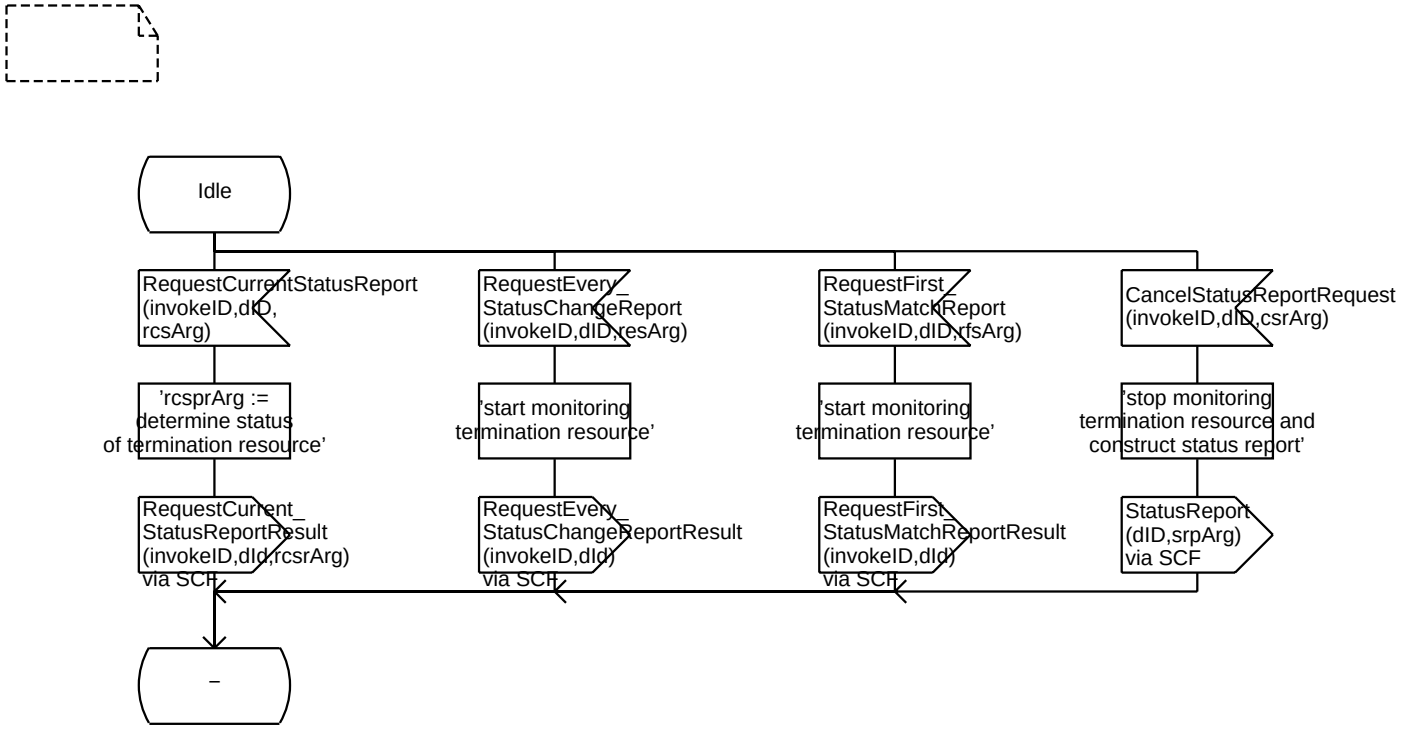



Process Type SSME_Control
16(21)
Idle
RequestReport_
BCSMEvent
(invokeID,csaID,
rrbcsmeArg)
ResetTimer
(invokeID,csaID,
rtArg)
i := 1
callSegmentID
Present(rtArg)
(false)
true
legIDPresent
(rrbcsmeArg!
bcsmEvents(i))
false
rtArg!
callSegmentID := 1
true
rrbcsmeArg!
bcsmEvents(i)!
eventTypeBCSM
routeSelectFailure,
oCalledPartyBusy,
oAnswer,
oNoAnswer,
tMidCall,
tDisconnect,
tAbandon,
tBusy,
tNoAnswer,
tAnswer,
oTermSeized,
oSuspend,
origAttempt,
oReAnswer,
tReAnswer,
tSuspend,
facilitySelectedAndAvailable,
callAccepted,
authorizeRouteFailure,
terminationAttemptDenied
origAttemptAuthorized,
collectedInfo,
analysedInformation,
oMidCall,
oDisconnect,
oAbandon,
termAttemptAuthorized,
termAttempt,
originationAttemptDenied
RRBCSMEArg!
bcsmEvents(i)!
legID!sendingSideID:=
leg2
RRBCSMEArg!
bcsmEvents(i)!
legID!sendingSideID:=
leg1
csa := call
GetCSAfromCSAId
(csaID)
i = Length
(RRBCSMEArg!
bcsmEvents)
false
ResetTimer
(invokeID,,rtArg)
to CSA
i :=
i + 1
true
csa := call
GetCSAfromCSAId
(csaID)
RequestReportBCSM_
Event(invokeID,,
rrbcsmeArg) to csa
–

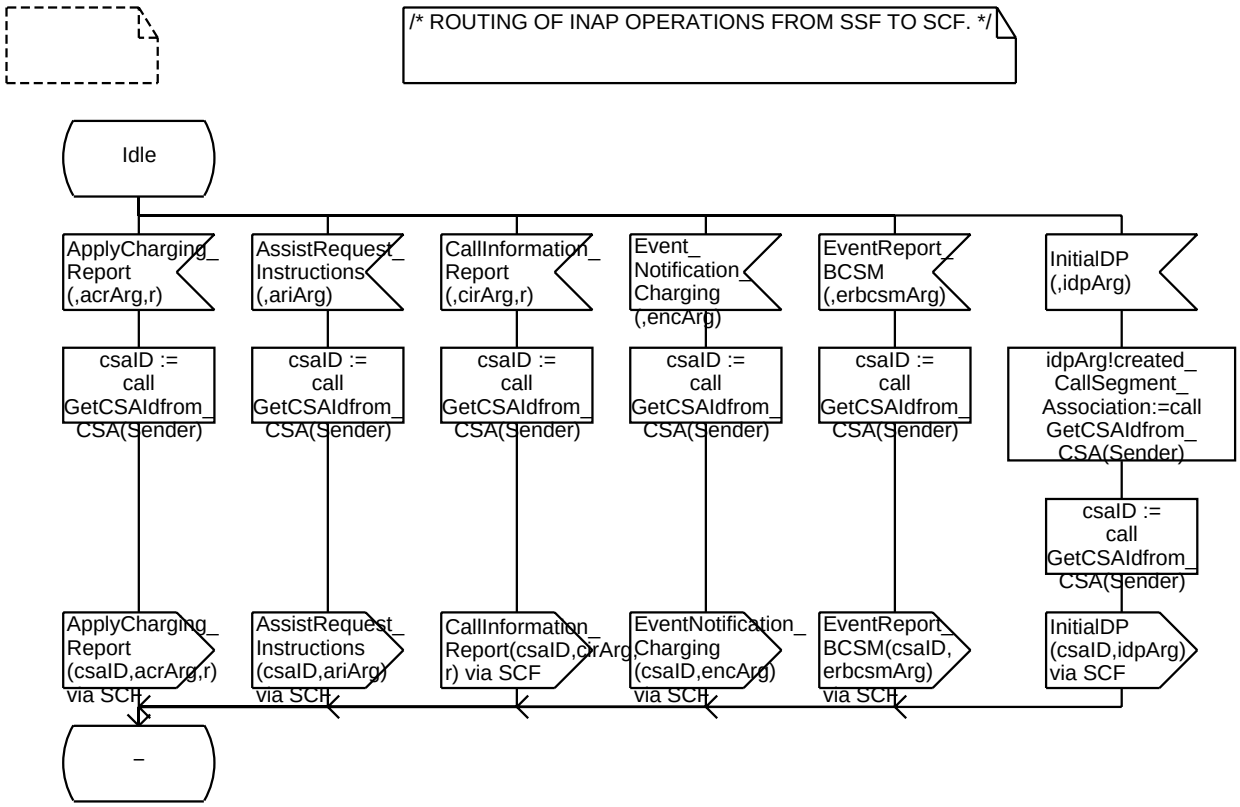
Process Type SSME_Control 17(21)



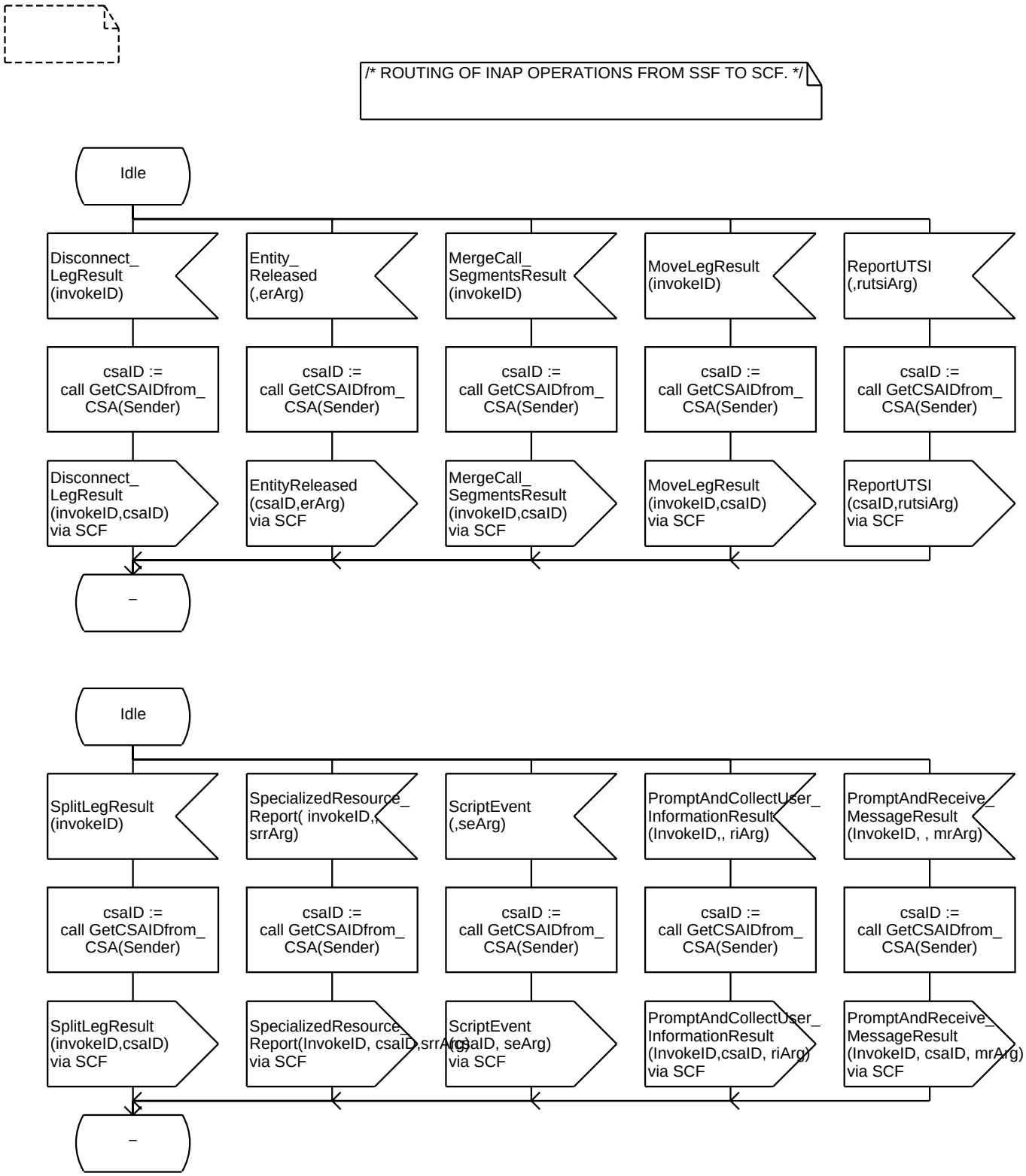
Process Type SSME_Control 18(21)



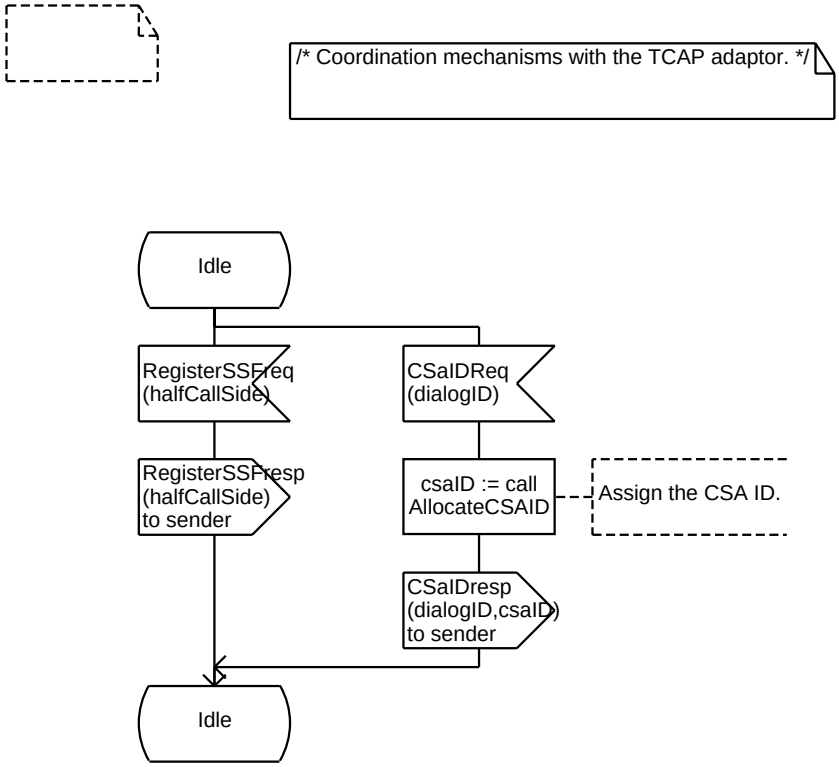
Process Type SSME_Control 19(21)



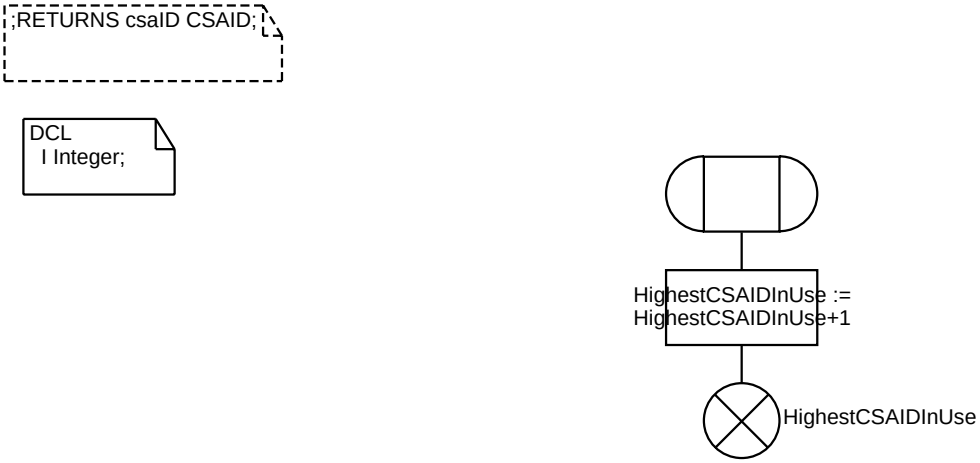
Process Type SSME_Control 20(21)



Process Type SSME_Control 21(21)

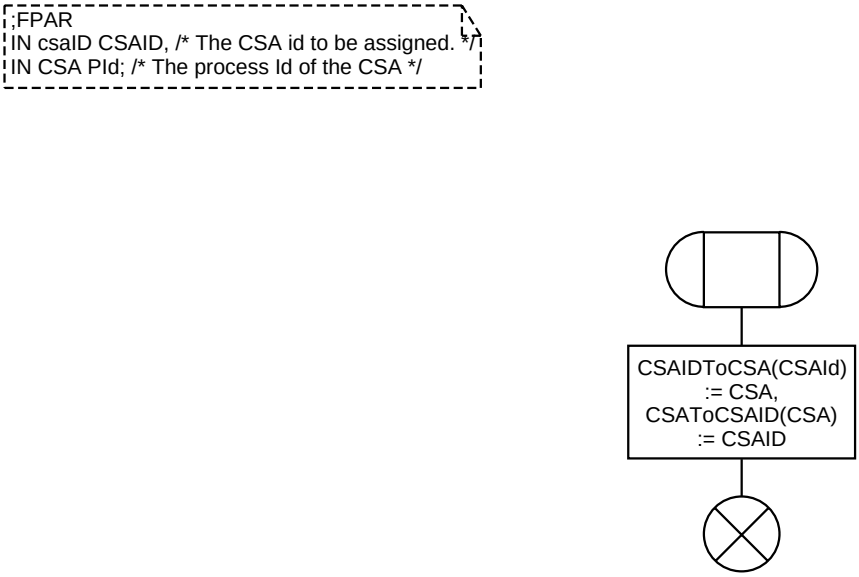


Procedure AllocateCSAId 1(1)



AddCSAId

Procedure AddCSAId 1(1)

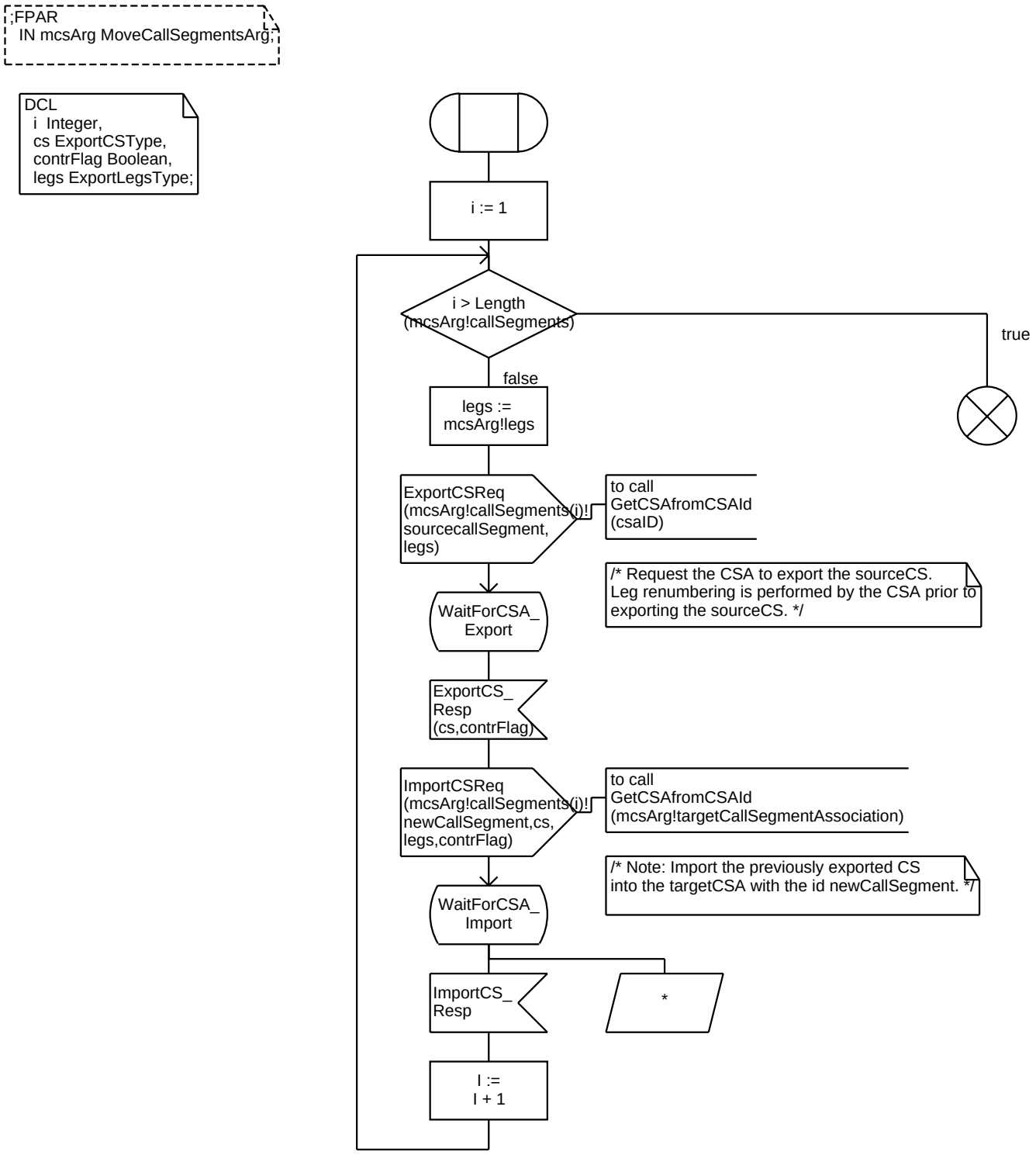


Procedure GetCSAIdfromCSA 1(1)

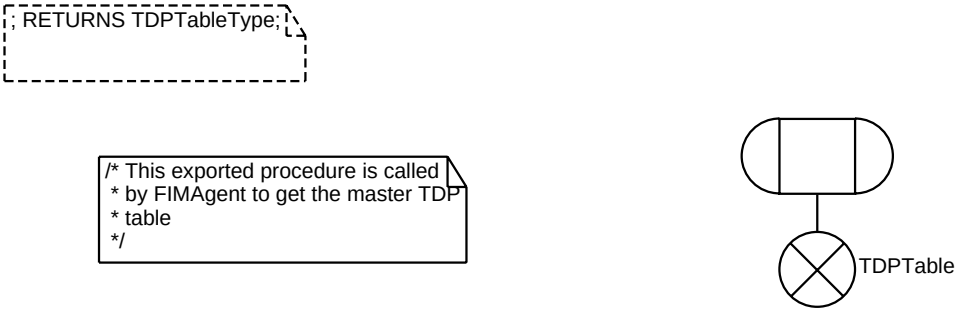
Procedure GetCSAfromCSAId 1(1)

Procedure IsCSA 1(1)

Procedure ProcessMoveCallSegments 1(1)



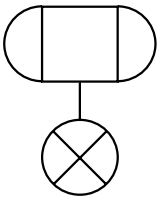
Procedure GetMasterTDPTable 1(1)



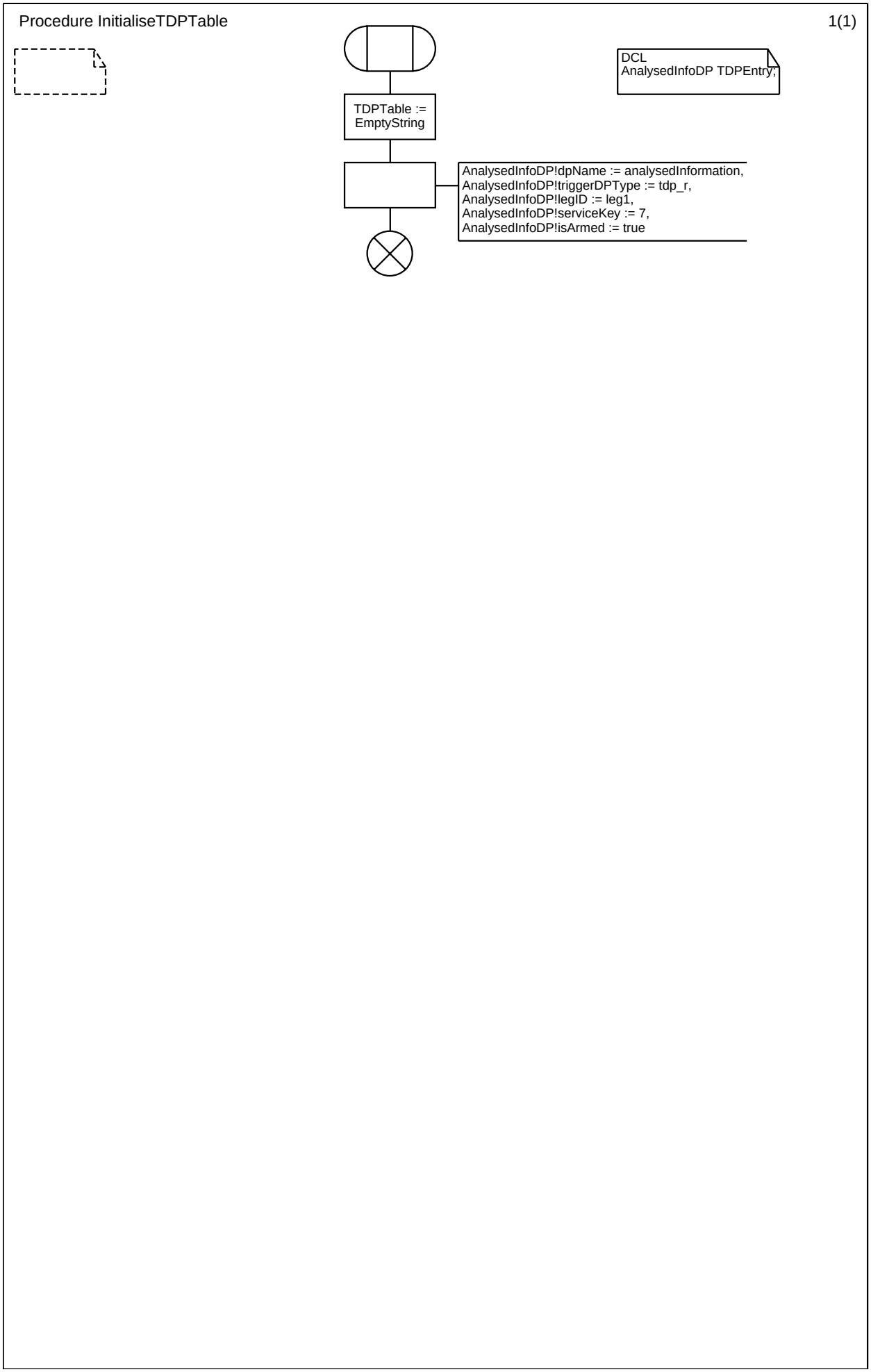
ProcessActivateServiceFiltering

Procedure ProcessActivateServiceFiltering 1(1)

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

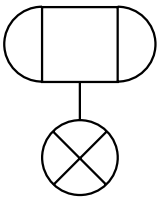


InitialiseTDPTable



Procedure ProcessCallGap 1(1)

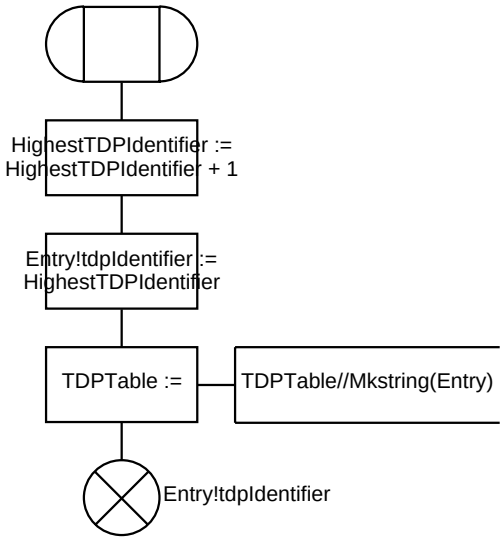
```
;FPAR
  IN invokeID Integer,
  IN cgArg    CallGapArg;
```



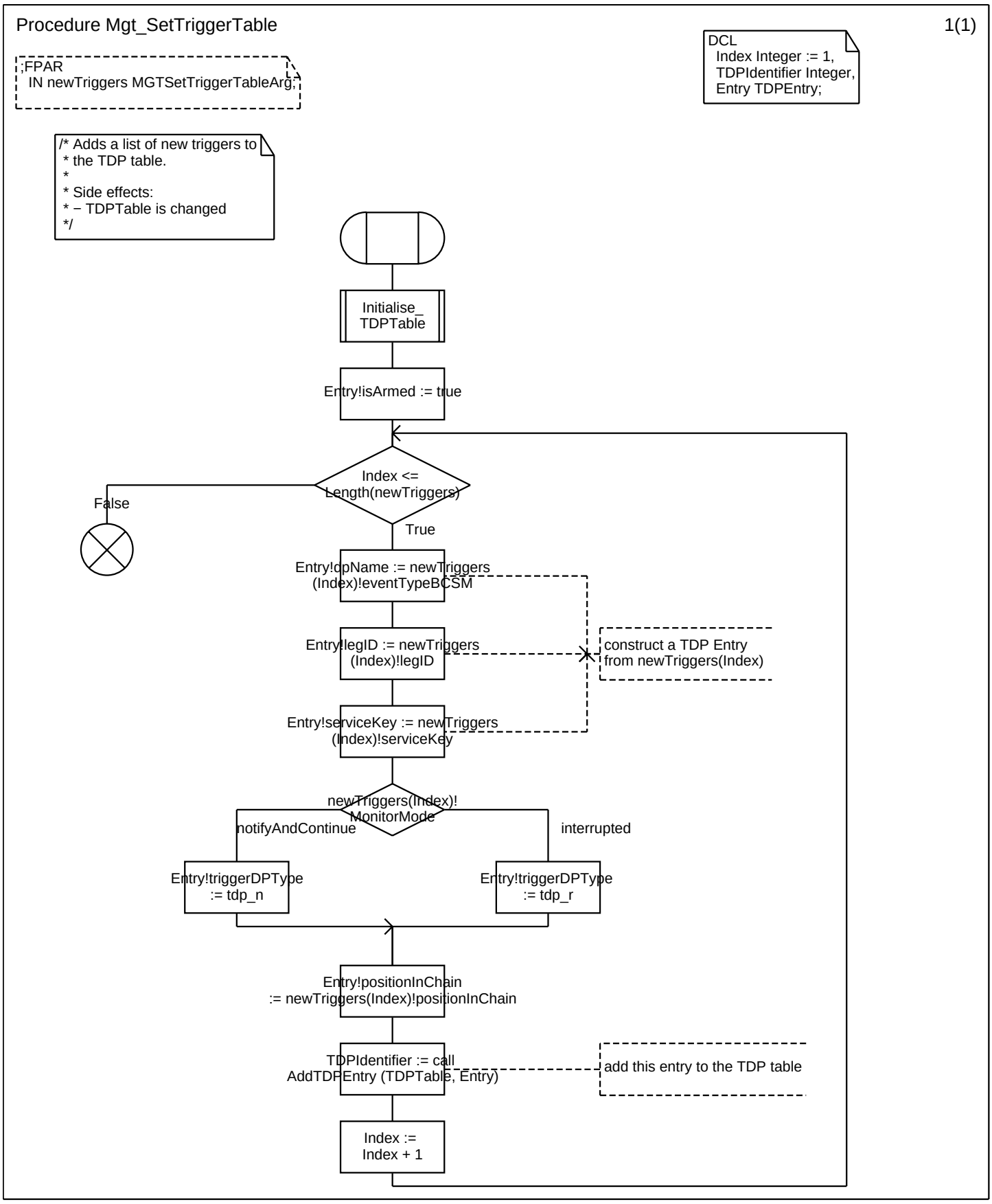
Procedure AddTDPEntry 1(1)

; FPAR IN/OUT TDPTable TDPTableType, Entry TDPEntry;
RETURNS Integer;

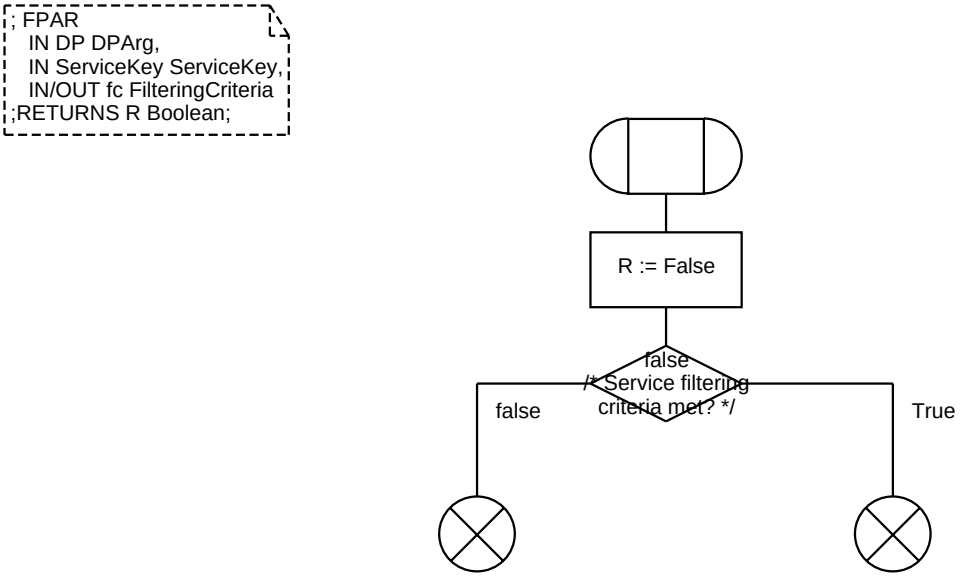
```
/* Adds a new entry to the TDP table.
 * Returns the TDP Identifier for
 * the newly created entry.
 *
 * Side effects:
 * − will increase HighestTDPIdentifier
 */
```



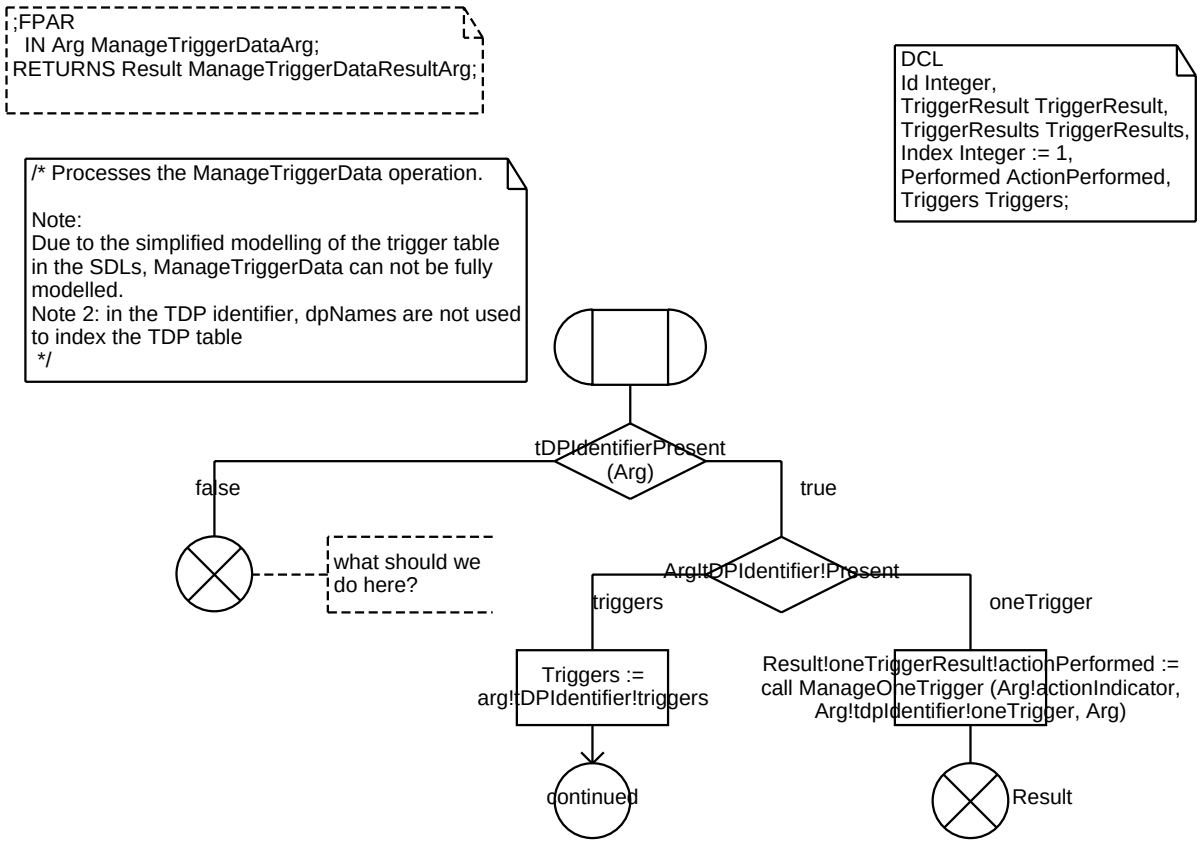
Mgt_SetTriggerTable



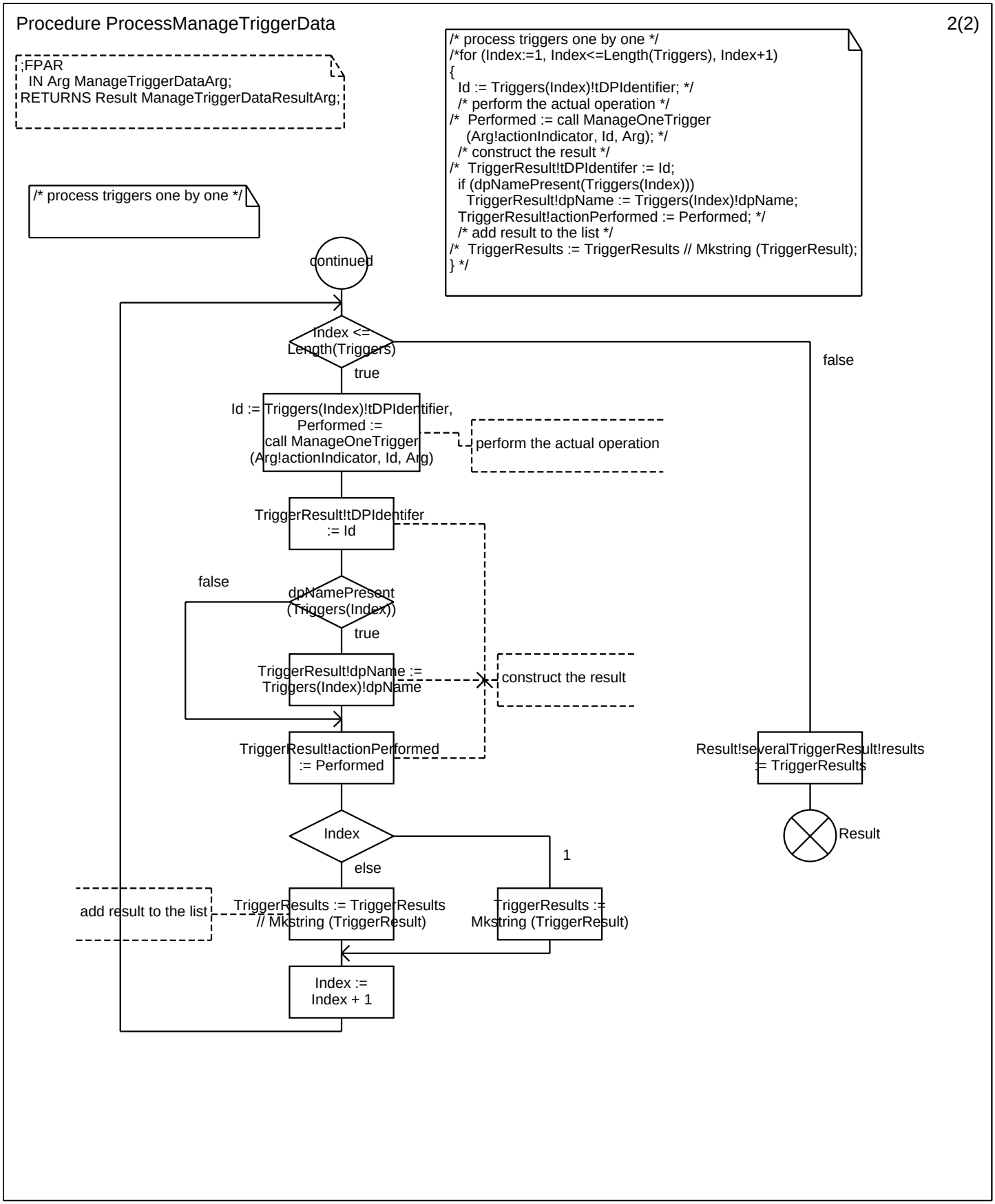
Procedure CheckMatchingServiceFilteringCriteria 1(1)



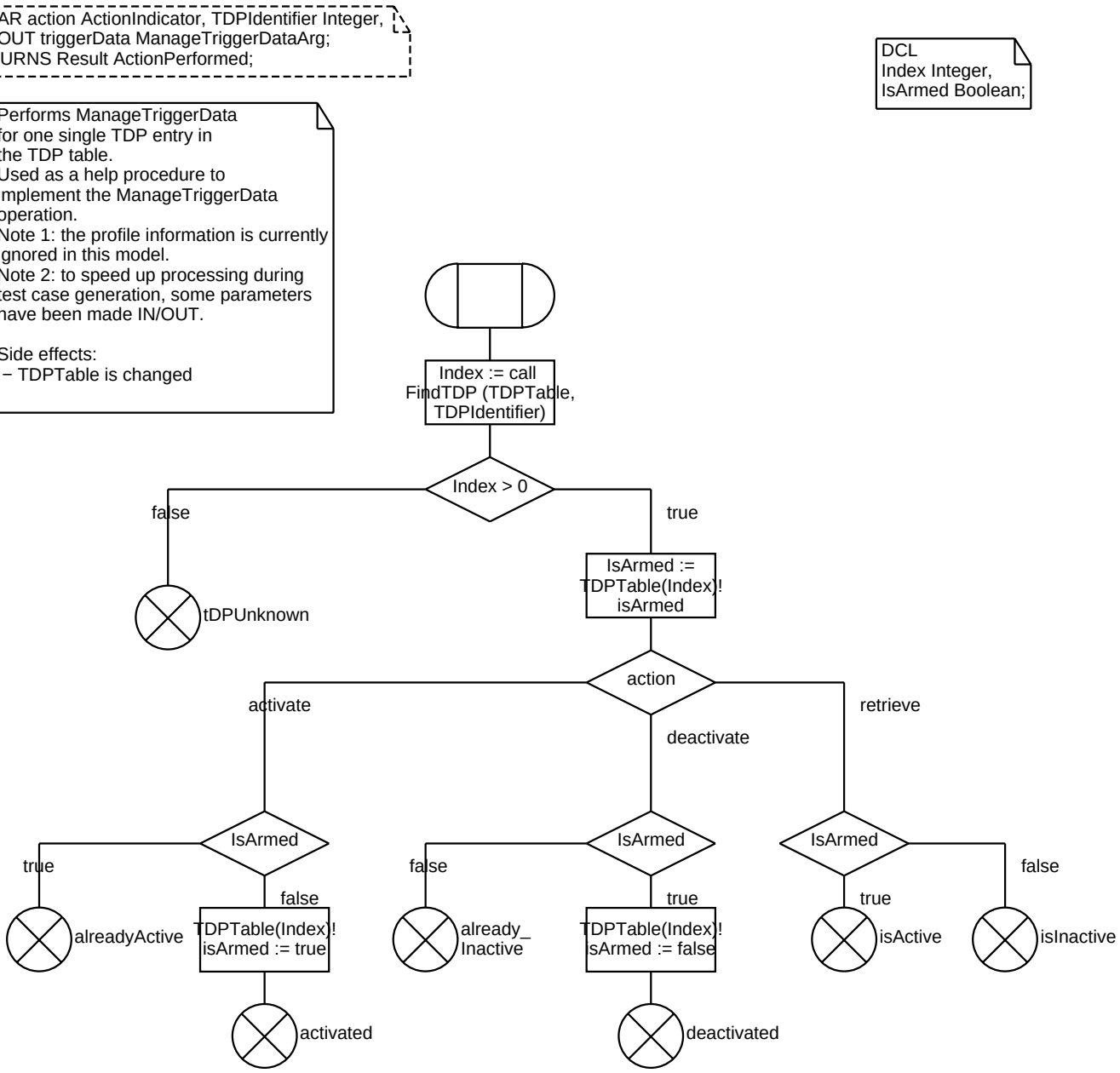
Procedure ProcessManageTriggerData 1(2)



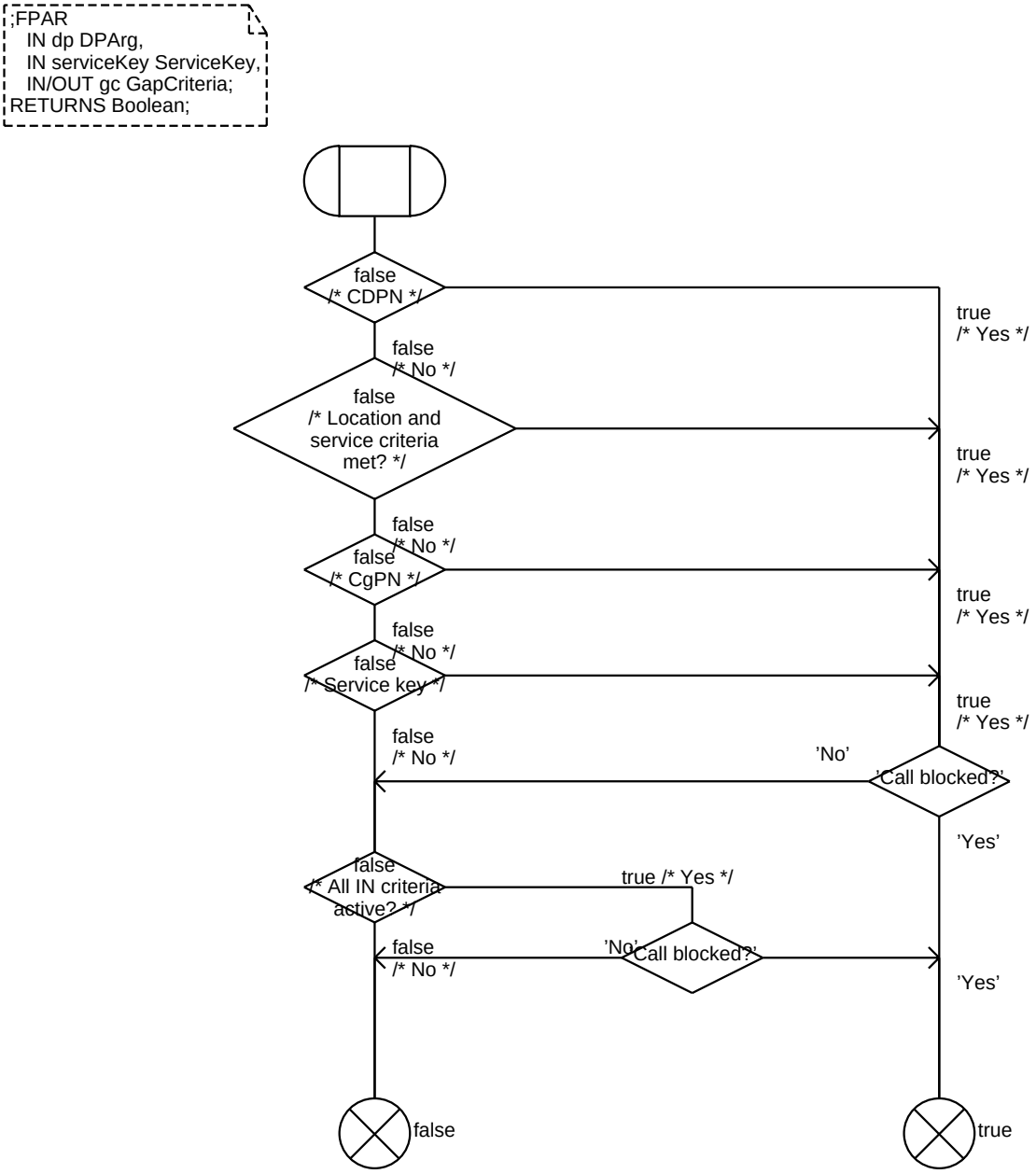
Procedure ProcessManageTriggerData 2(2)



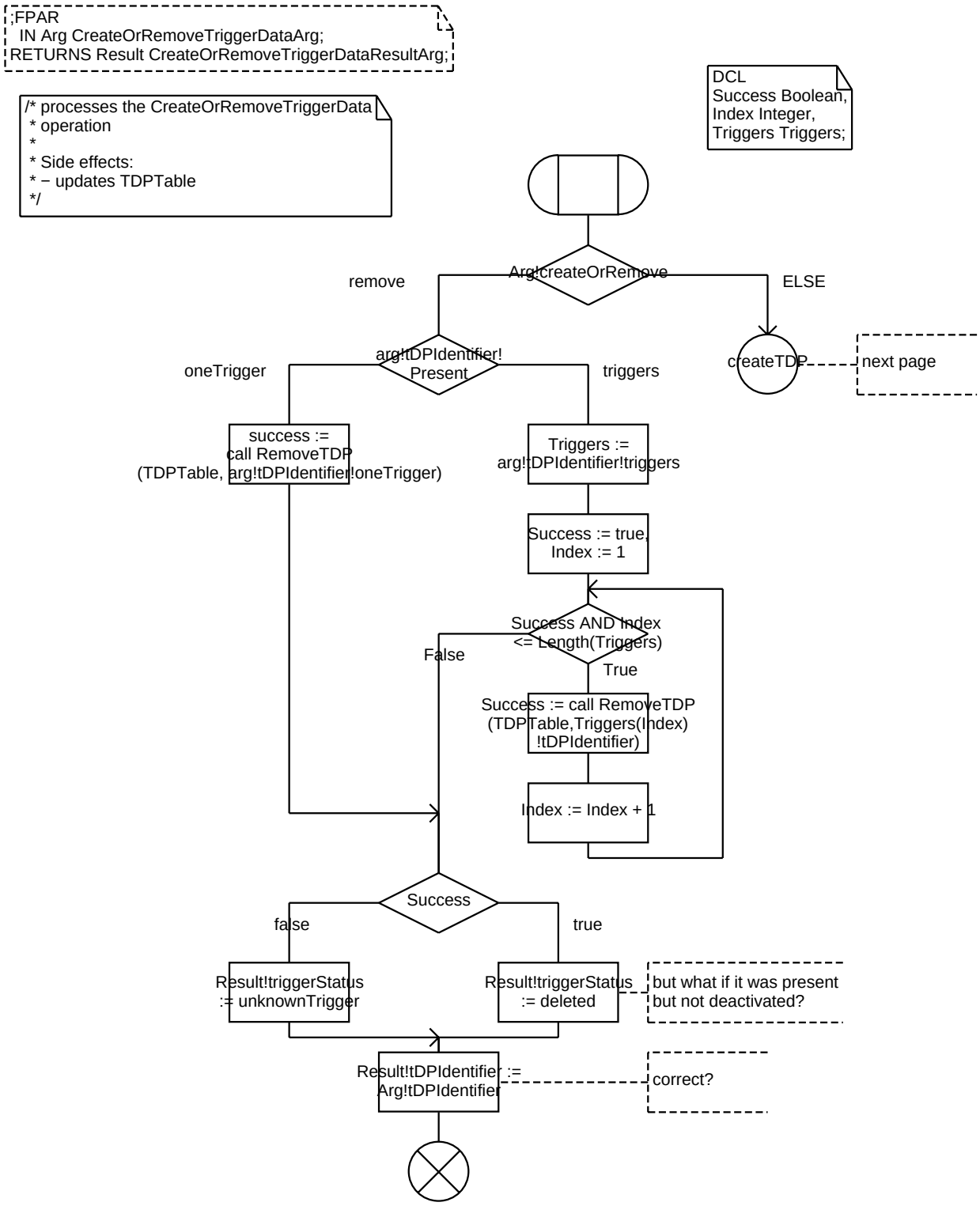
Procedure ManageOneTrigger 1(1)



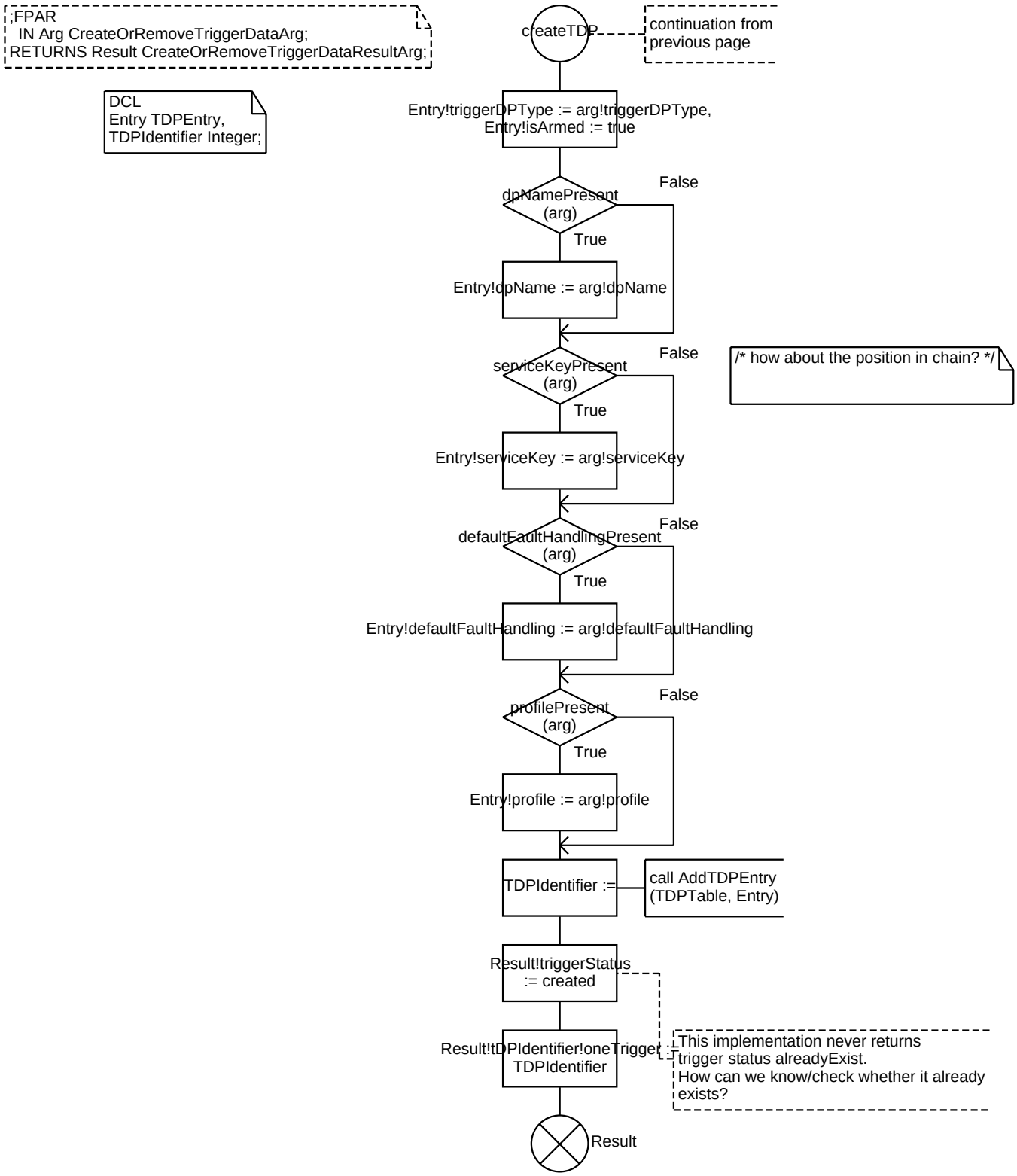
Procedure <<Process Type SSME_Control>> CheckACG 1(1)



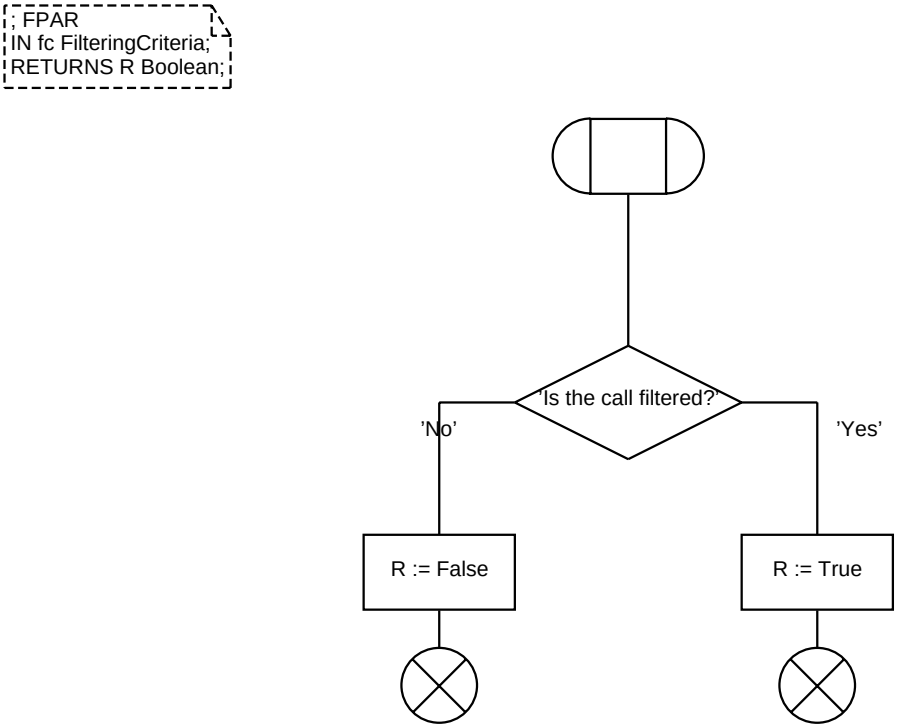
Procedure ProcessCreateOrRemoveTriggerData 1(2)



Procedure ProcessCreateOrRemoveTriggerData 2(2)

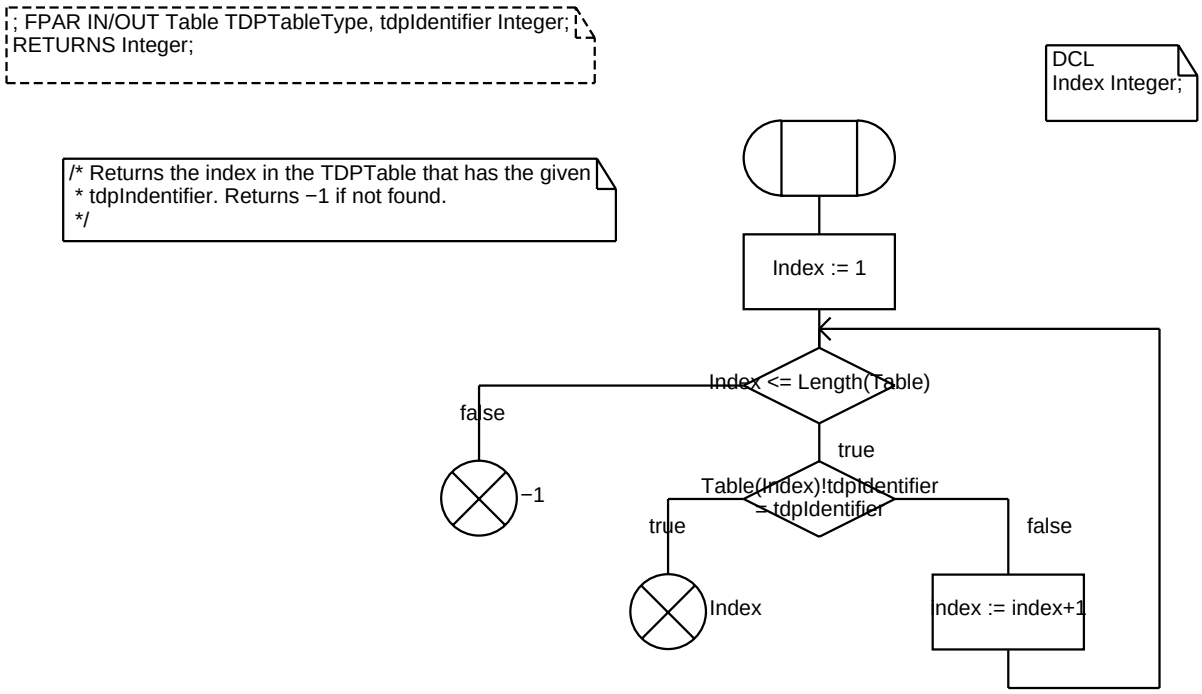


Procedure CheckCallFiltered 1(1)

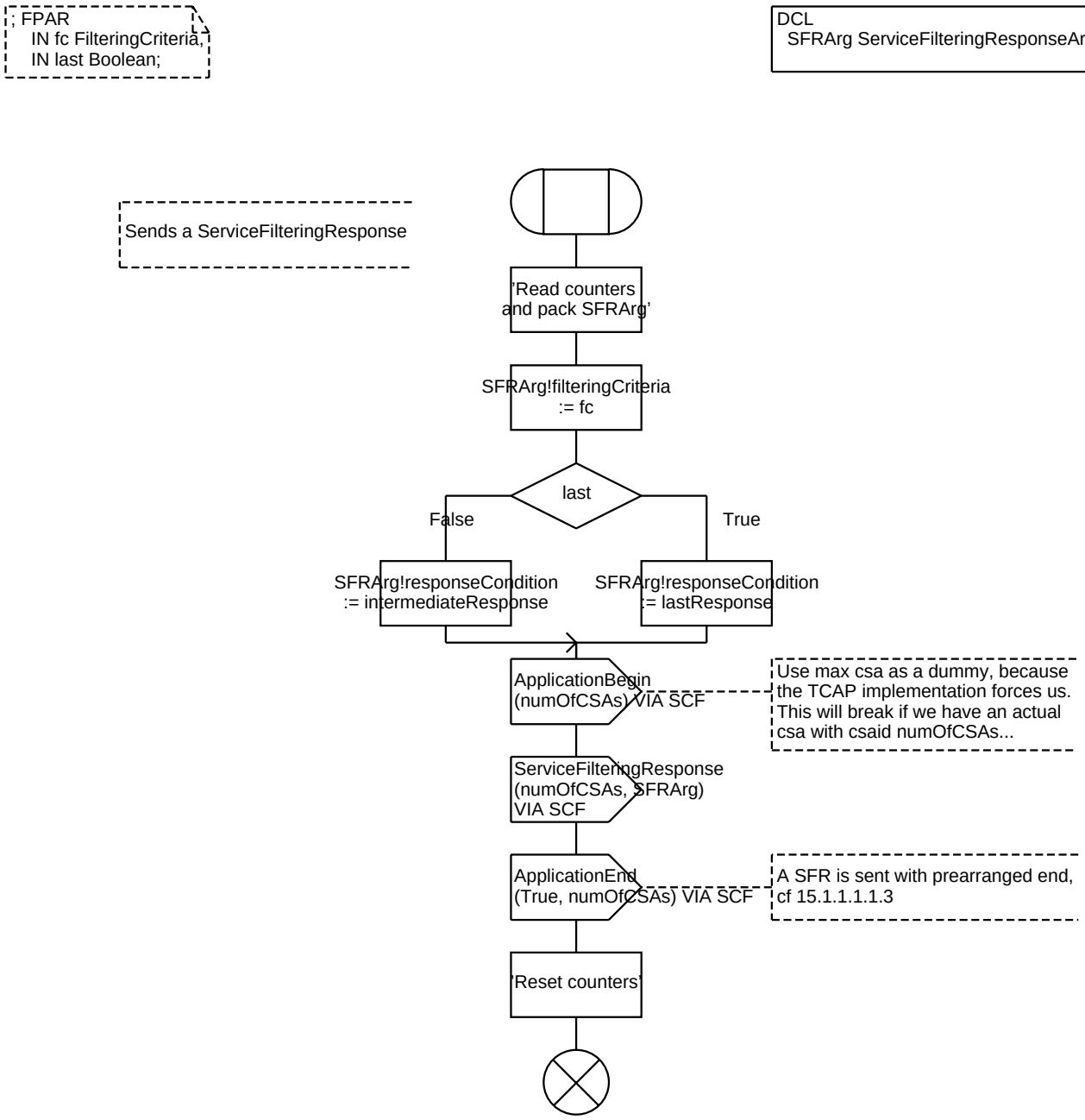


Procedure CountCall 1(1)

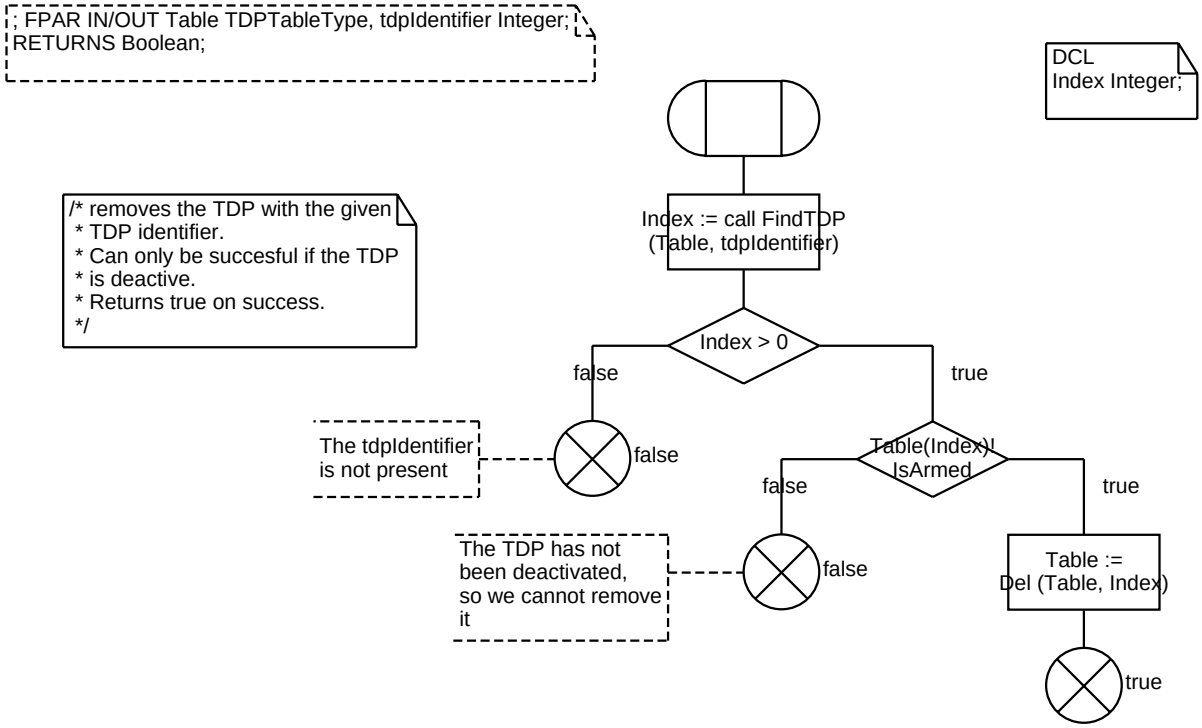
Procedure FindTDP 1(1)



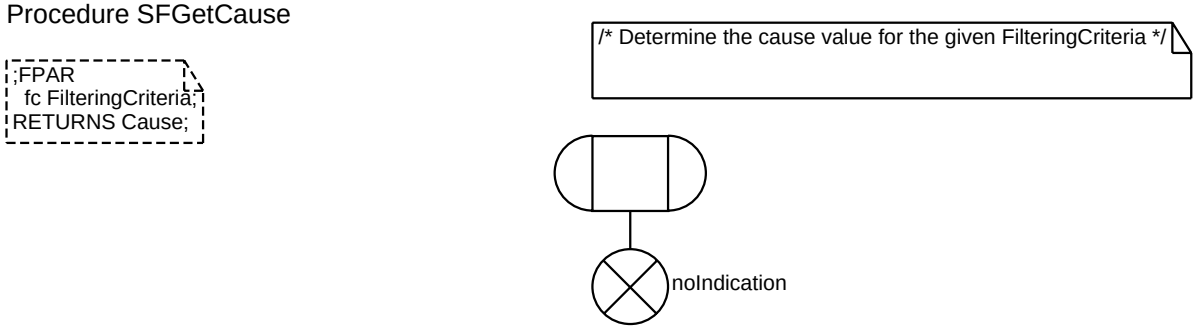
Procedure SendSFR 1(1)



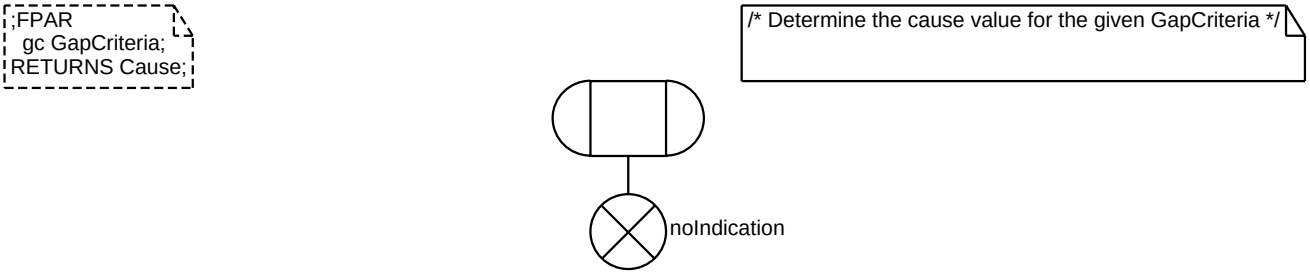
Procedure RemoveTDP 1(1)



Procedure SFGetCause 1(1)



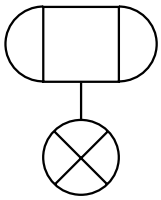
Procedure CGGetCause 1(1)



Procedure SFInfoToSend 1(1)

;FPAR IN fc FilteringCriteria;
RETURNS infoToSend InformationToSend;

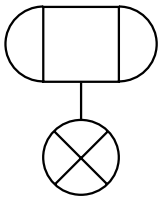
/* Determine the infoToSend for the given FilteringCriteria */



Procedure CGInfoToSend 1(1)

;FPAR IN gc GapCriteria;
RETURNS infoToSend InformationToSend;

/* Determine the infoToSend for the given GapCriteria */



Process Type CallSegmentAssociation 1(29)

```
;FPAR
 csaID CSAID;
```

```
/**** 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. */
  SSFPtr PId;   /*Pointer to SSF which corresponds to CS (1−1 relationship) */
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,Integer)
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 ResultType
  LITERALS Successful, Failed;
ENDNEWTYPE;

/* Definition of dialogue primitives that can be broadcast to the CSs. */
NEWTYPE BroadcastPrimitiveNames
  LITERALS ApplicationAbort, ApplicationContinue, ApplicationEnd;
ENDNEWTYPE;
```

```
/* 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;
```

```
/*  IMPORTED REMOTE PROCEDURES */

/* all remote procedures below are exported by CS and
   called by CSA */

/* returns the PId of the associated FIM Agent */
IMPORTED PROCEDURE GetFAPtr;
 RETURNS Pid;

/* is the MergeCallSegments operation applicable for the
   called CS (applies to both Source and Target CS */
IMPORTED PROCEDURE ValidSourceCSforMCS;
 RETURNS Boolean;

/* is the MoveLeg operation applicable if the
   called CS is the source CS */
IMPORTED PROCEDURE ValidSourceCSforML;
 FPAR MoveLegArg;
 RETURNS Boolean;

/* is the MoveLeg operation applicable if the
   called CS is the target CS */
IMPORTED PROCEDURE ValidTargetCSforMLMCS;
 RETURNS Boolean;

/* is the SplitLeg operation applicable if the
   called CS is the source CS */
IMPORTED PROCEDURE ValidSourceCSforSL;
 FPAR SplitLegArg;
 RETURNS Boolean;

/* Delete all pending legs in CS (called in Source CS
before performing MoveCallSegments) */
IMPORTED PROCEDURE DeletePendingLeg;
```

```
/* Codes of the operations that can be queued: RequestReportBCSM, ApplyCharging,
   FurnishChargingInformation,RequestNotificationChargingEvent and
CallInformationRequest */
NEWTYPE OperationType
  LITERALS RRB,AC,FCI,RNCE,CIRQ;
ENDNEWTYPE;

/* QueuedOperation is used to represent an operation and its arguments
   that should be queued */
NEWTYPE QueuedOperation STRUCT
  operation OperationType;
  invokeID InvokeID;
  opArg ArgType;
ENDNEWTYPE;

/* OperationQueue is used to queue certain (e.g. RRBE amd ApplyCharging) operations
   sent for a leg not yet created */
NEWTYPE OperationQueue
  String (QueuedOperation, emptystring);
 OPERATORS
  Add: OperationQueue, QueuedOperation −> OperationQueue;
  /* adds a new operation to the queue */
  Del: OperationQueue, Integer −> OperationQueue;
  /* removes 1 operation (indicated by the index) from the queue */
 OPERATOR Add;
  FPAR opQueue OperationQueue, queuedOp QueuedOperation;
 RETURNS OperationQueue;
 START;
   RETURN opQueue // Mkstring (queuedOp);
 ENDOPERATOR;
 OPERATOR Del;
  FPAR opQueue OperationQueue, index Integer;
 RETURNS OperationQueue;
 START;
   RETURN Substring(opQueue, 1, index−1) // Substring (opQueue, index+1,
      Length(opQueue)−index);
 ENDOPERATOR;
ENDNEWTYPE;
```

Process Type CallSegmentAssociation 2(29)

```
FPAR
  csaID CSAID;
```

```
/**** 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. */
  csID, newcsID CallSegmentID,
  CSStopped Boolean,
  invokeID InvokeID,
  termination Boolean,
  initialPassiveLegID, legID, newLegID LegType,
  eventTable EventTableType,
  r Boolean,
  bcsm BCSMType,
  FA PId,
  serviceKey ServiceKey,
  sourceCSPId, targetCSPId PId,
  requests Integer,
  maxLeg LegType;
```

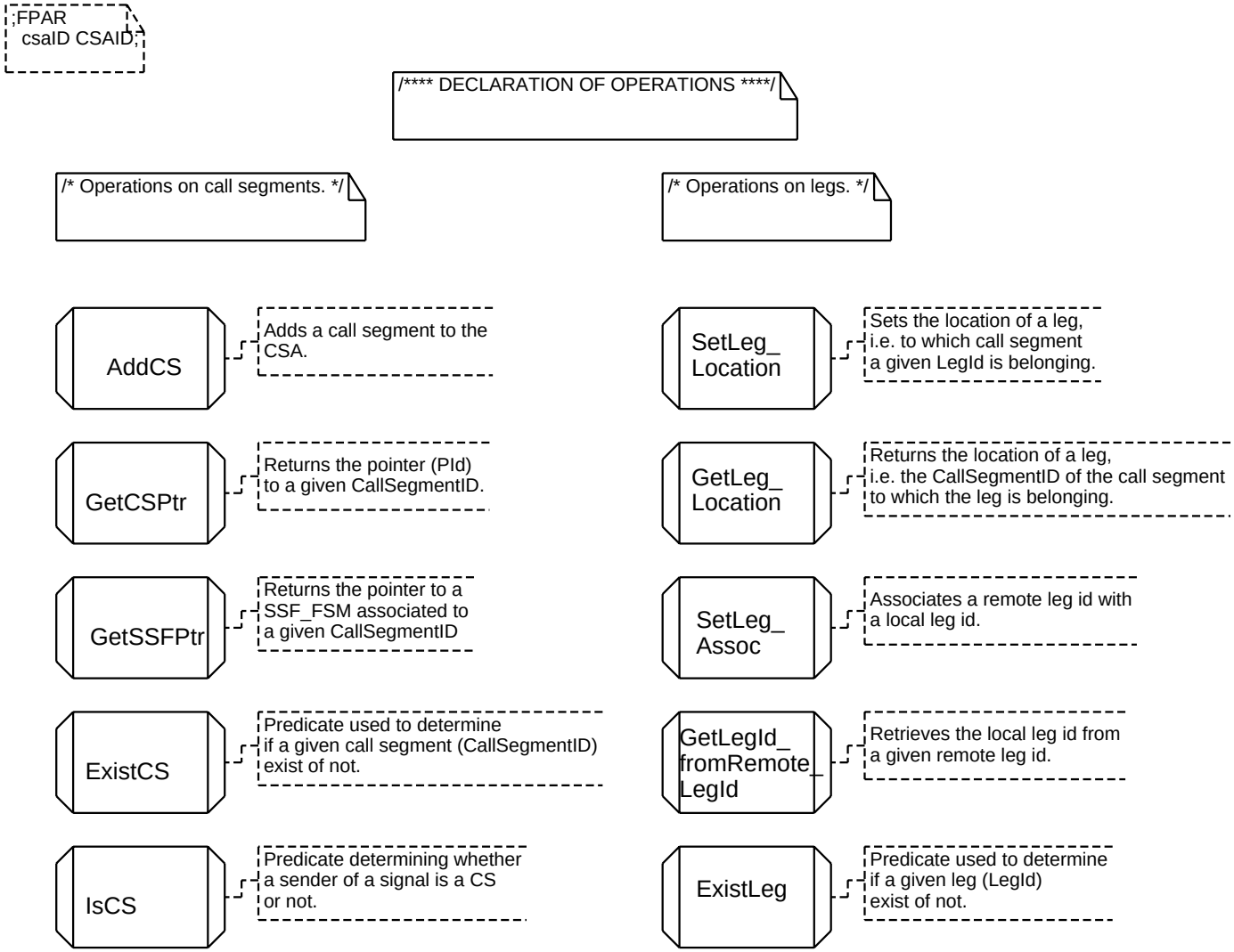
```
DCL
  rrbeQueue RRBEQueueType,     /* The queued RRBEs. */

  opQueue OperationQueue,
  qopArg ArgType,
  queueActive Boolean := false,
  queuedLeg LegType,
  contrFlag Boolean,
  leg   ExportLegType,
  legs  ExportLegsType,
  expCS ExportCSType,
  n Integer,
  timedRelease Boolean := False;
```

```
TIMER
TDisconnect,
Tati,
Tat := 10;
```

```
DCL
  /* 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,
  MRArg MessageReceivedArg,
  PAArg PlayAnnouncementArg,
  PACUIArg PromptAndCollectUserInformationArg,
  PARMArg PromptAndReceiveMessageArg,
  RIArg ReceivedInformationArg,
  RCArg ReleaseCallArg,
  RNCEARg RequestNotificationChargingEventArg,
  RRBCSMEArg RequestReportBCSMEventArg,
  RTARg ResetTimerArg,
  SCArg ScriptCloseArg,
  SEArg ScriptEventArg,
  SFArg SelectFacilityArg,
  SIArg ScriptInformationArg,
  SCRArg ScriptRunArg,
  SCIArg SendChargingInformationArg,
  SRRArg SpecializedResourceReportArg,
  IEArg ErrorArg;
```

```
DCL
  /* IN operation arguments. */
  ctwaArg ContinueWithArgumentArg,
  dfcwaArg DisconnectForwardConnectionWithArgumentArg,
  dlArg DisconnectLegArg,
  erArg EntityReleasedArg,
  mtdArg ManageTriggerDataArg,
  mtdrArg ManageTriggerDataResultArg,
  mcsArg MergeCallSegmentsArg,
  mlArg MoveLegArg,
  rutsiArg ReportUTSIArg,
  rrutsiArg RequestReportUTSIArg,
  sstuiArg SendSTUIArg,
  sspArg SetServiceProfileArg,
  slArg SplitLegArg;
```

Process Type CallSegmentAssociation 3(29)



Process Type CallSegmentAssociation 4(29)

FPAR
csaID CSAID;

/**** 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.

EXPORTED RemoveLegCSA — 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 operation queue. */

SendQueued_Operations

QueueOperation

Process_RRBE

ProcessFCI

ProcessAC

ProcessCIRQ

Process_RNCE

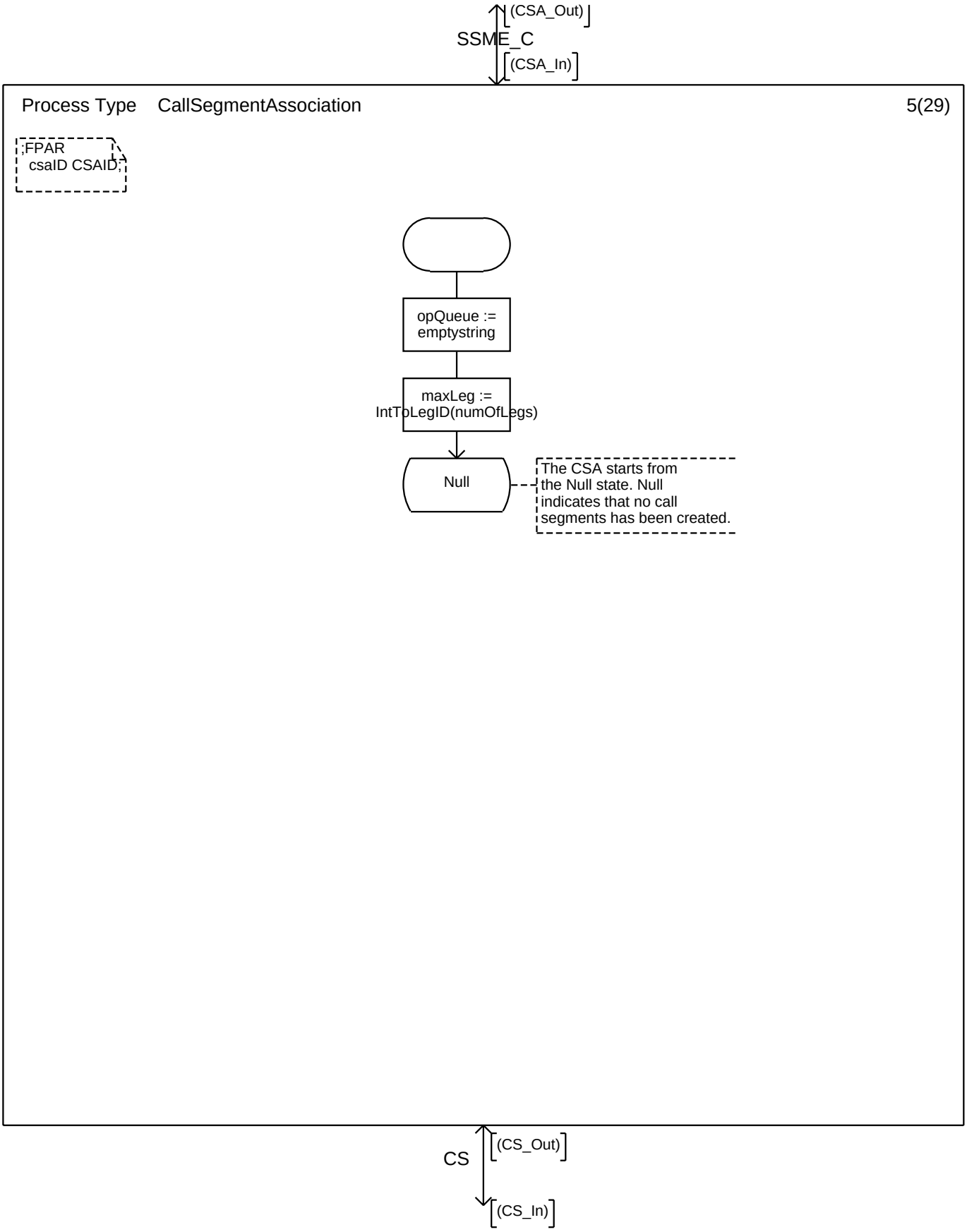
ProcessCancel

Release_Call

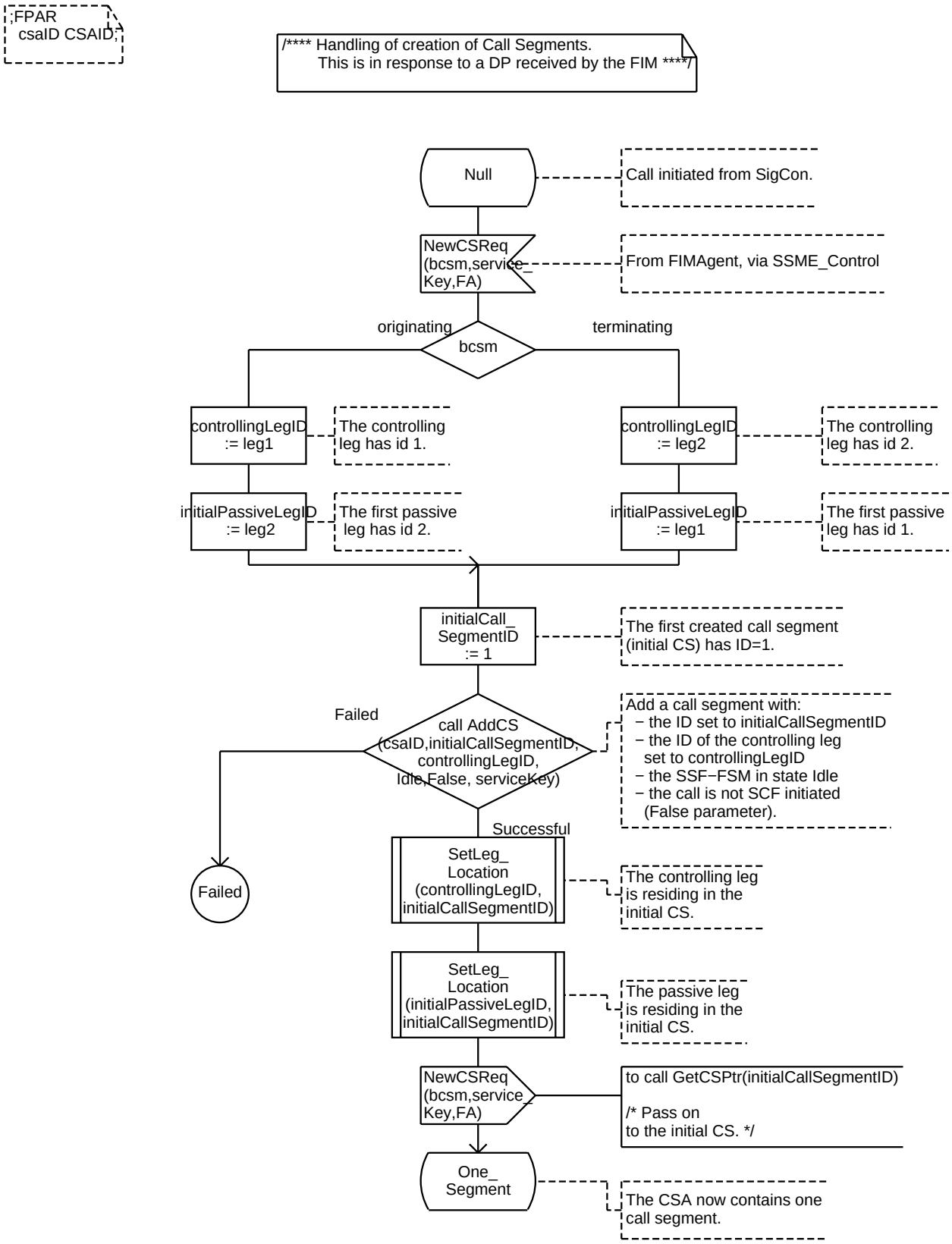
ReleaseAll_Segments

/* Sending of dialogue primitives. */

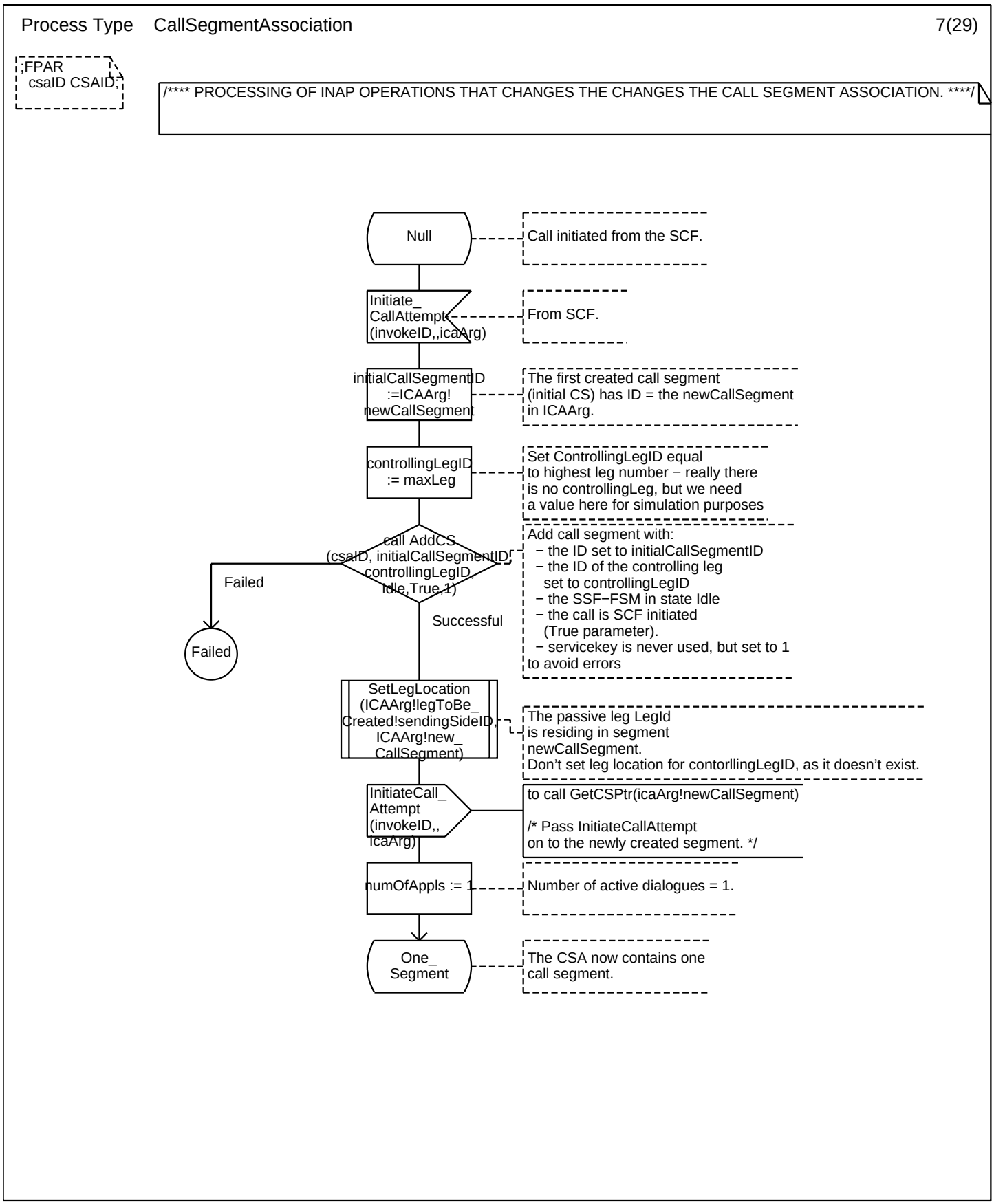
Broadcast_ToCSs


(CSA_Out)
SSME_C
(CSA_In)
Process Type CallSegmentAssociation
5(29)
FPAR
csaID CSAID;
opQueue :=
emptystring
maxLeg :=
IntToLegID(numOfLegs)
Null
The CSA starts from
the Null state. Null
indicates that no call
segments has been created.
CS
(CS_Out)
(CS_In)

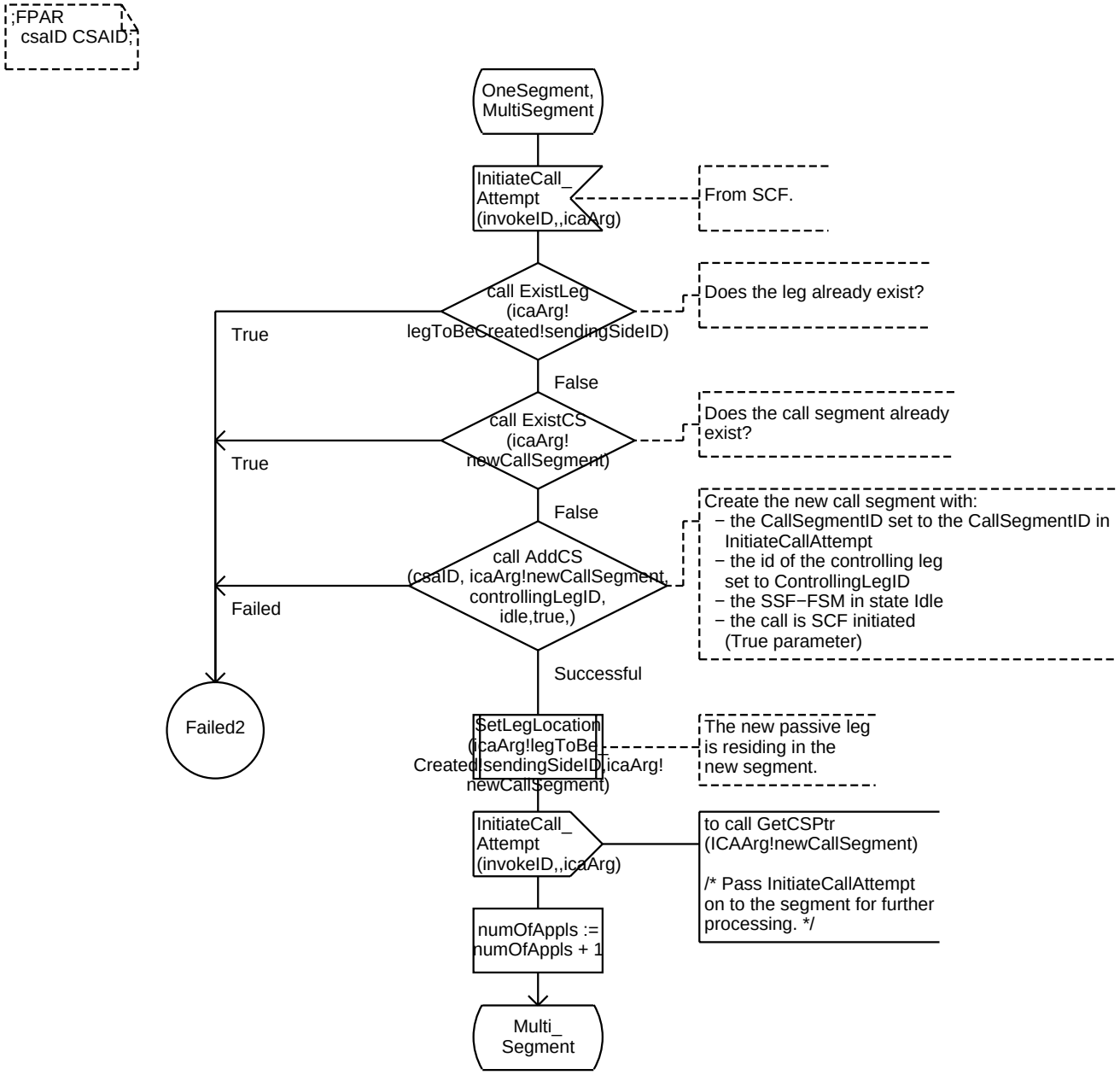
Process Type CallSegmentAssociation 6(29)



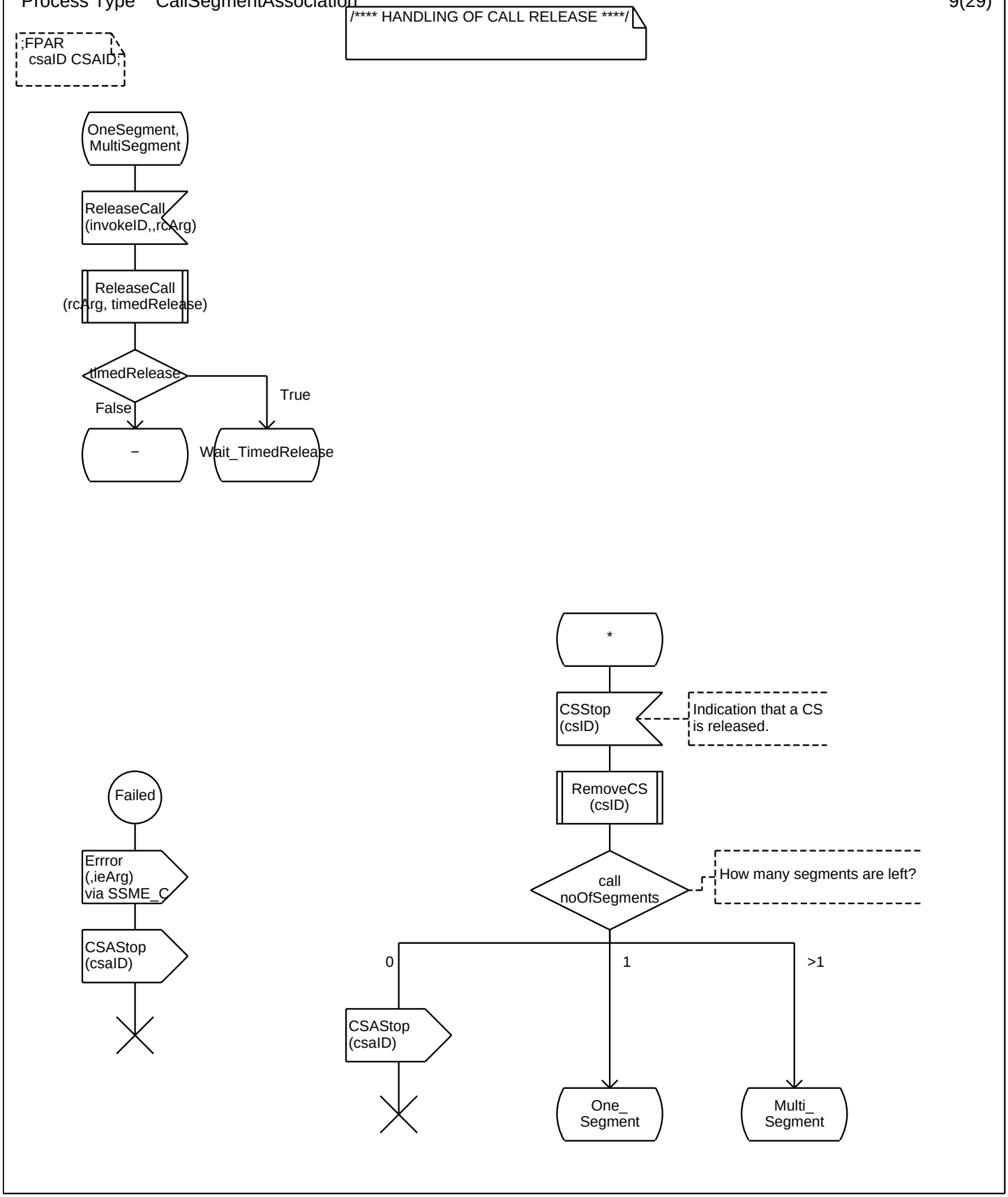
Process Type CallSegmentAssociation 7(29)



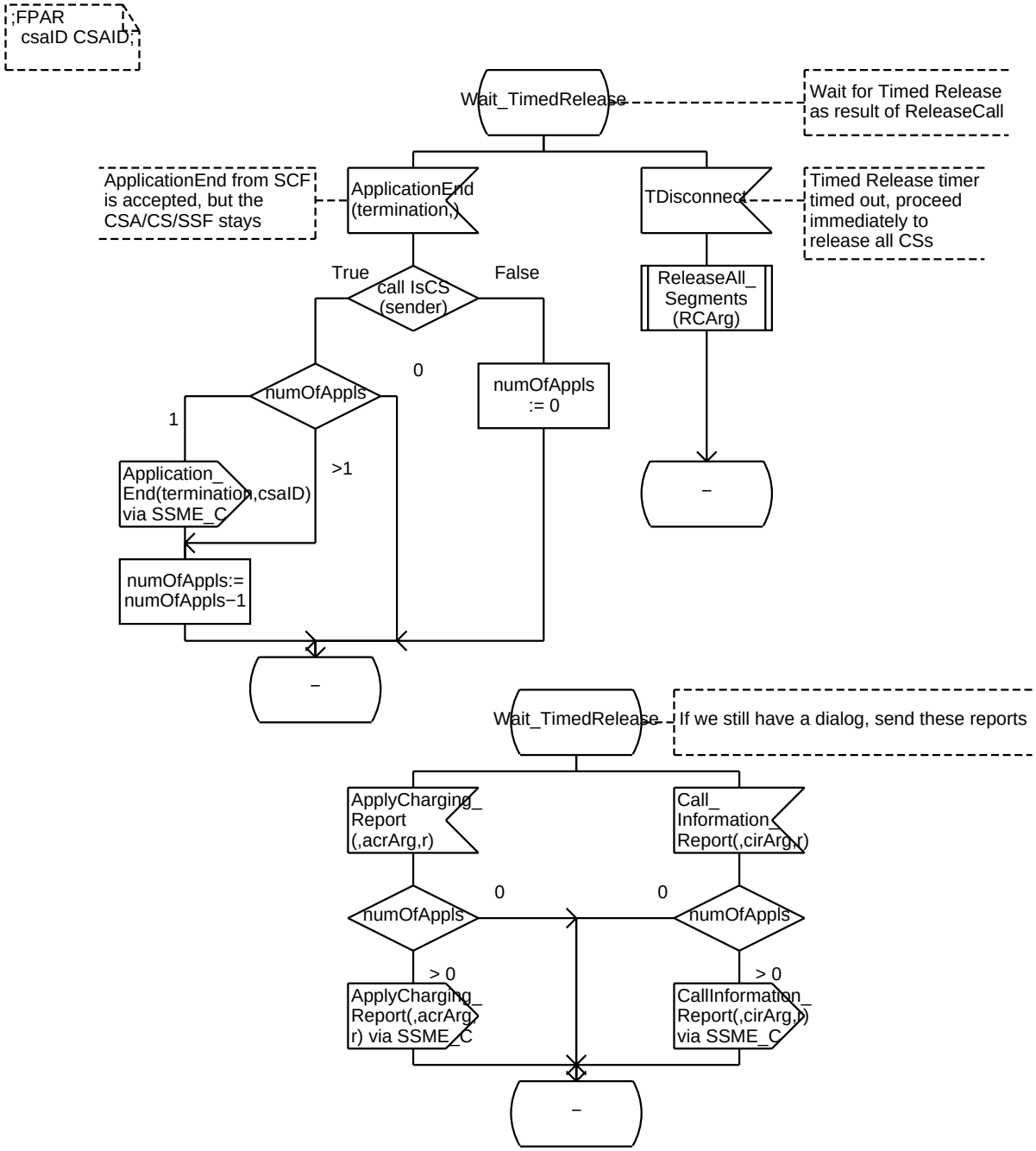
Process Type CallSegmentAssociation 8(29)



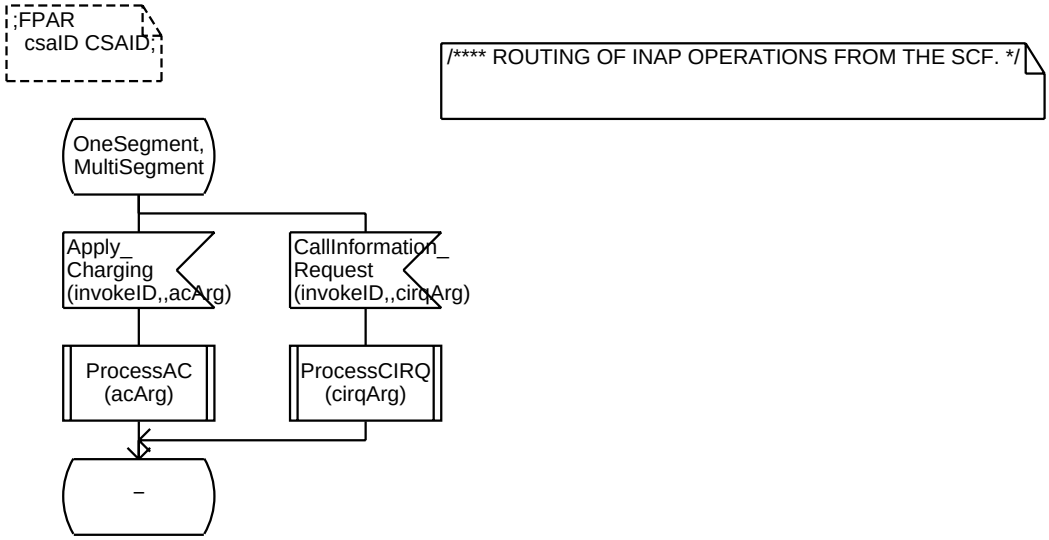
Process Type CallSegmentAssociation 9(29)

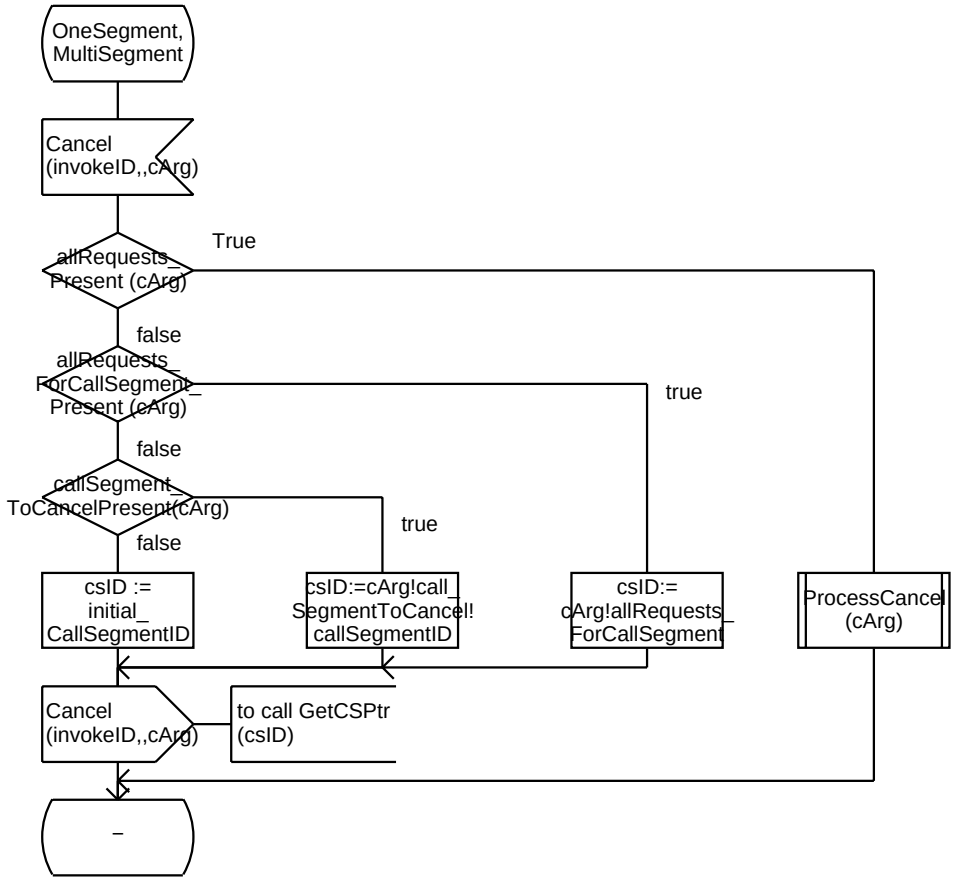


Process Type CallSegmentAssociation 10(29)

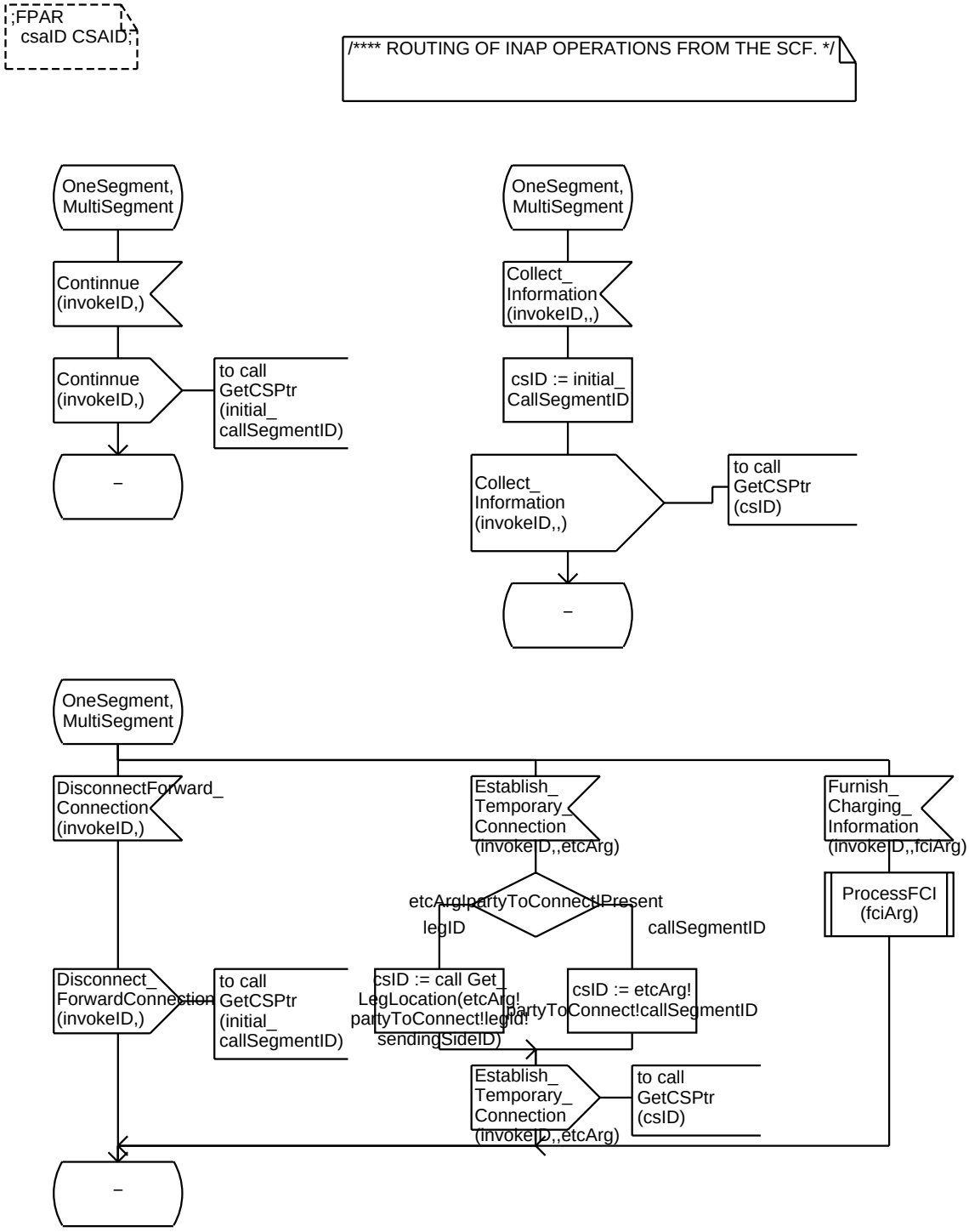


Process Type CallSegmentAssociation 11(29)

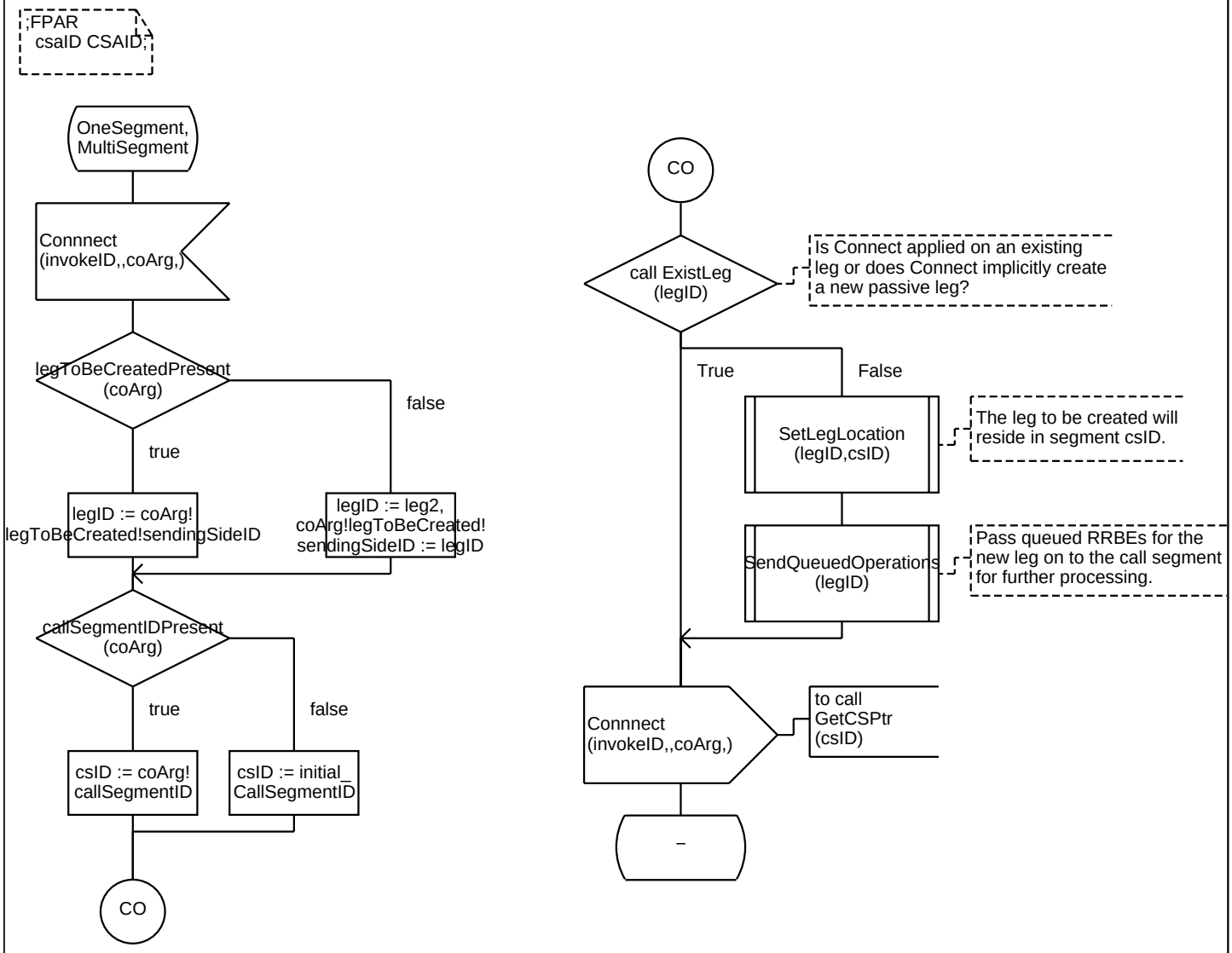




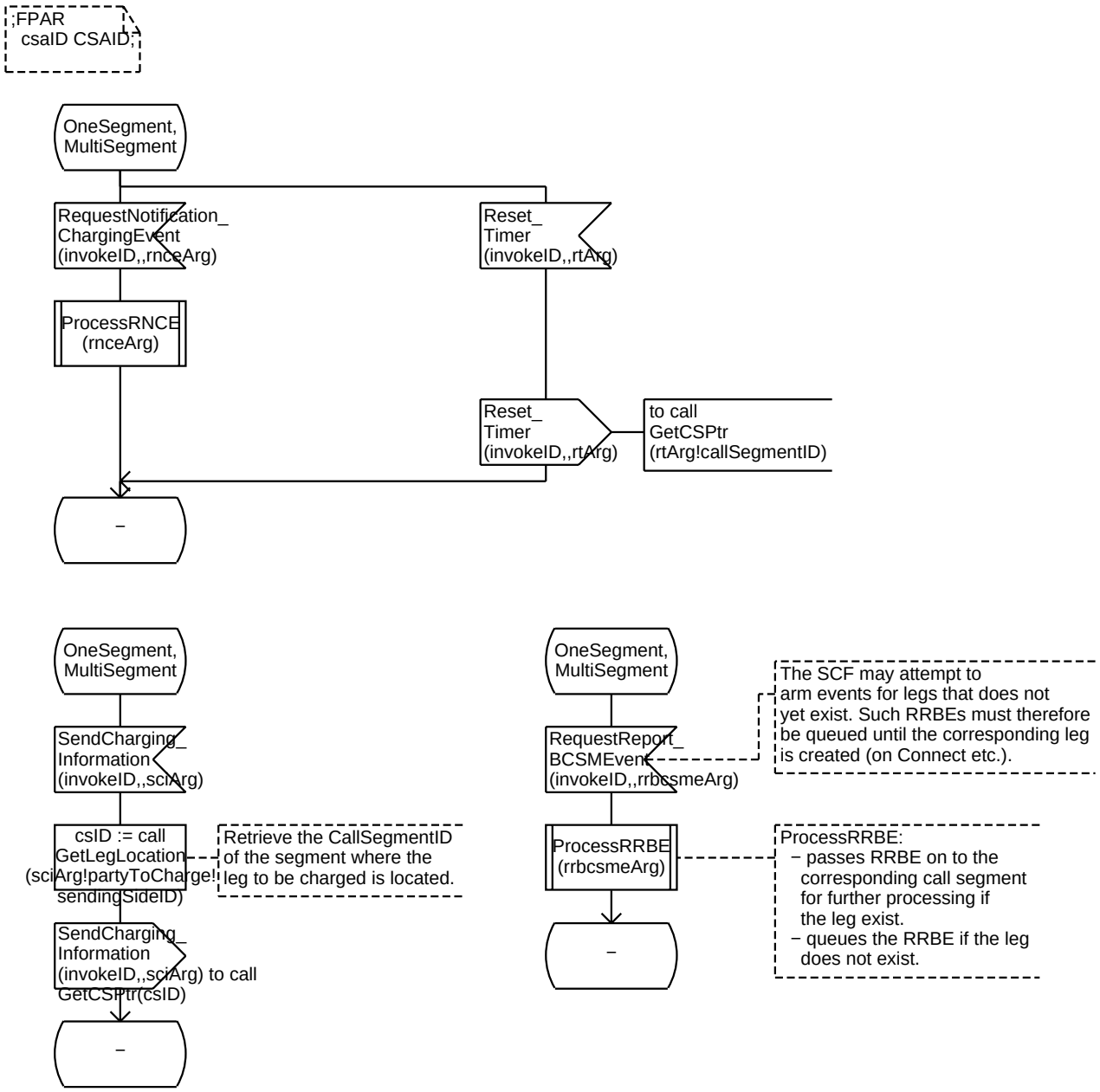
Process Type CallSegmentAssociation 12(29)



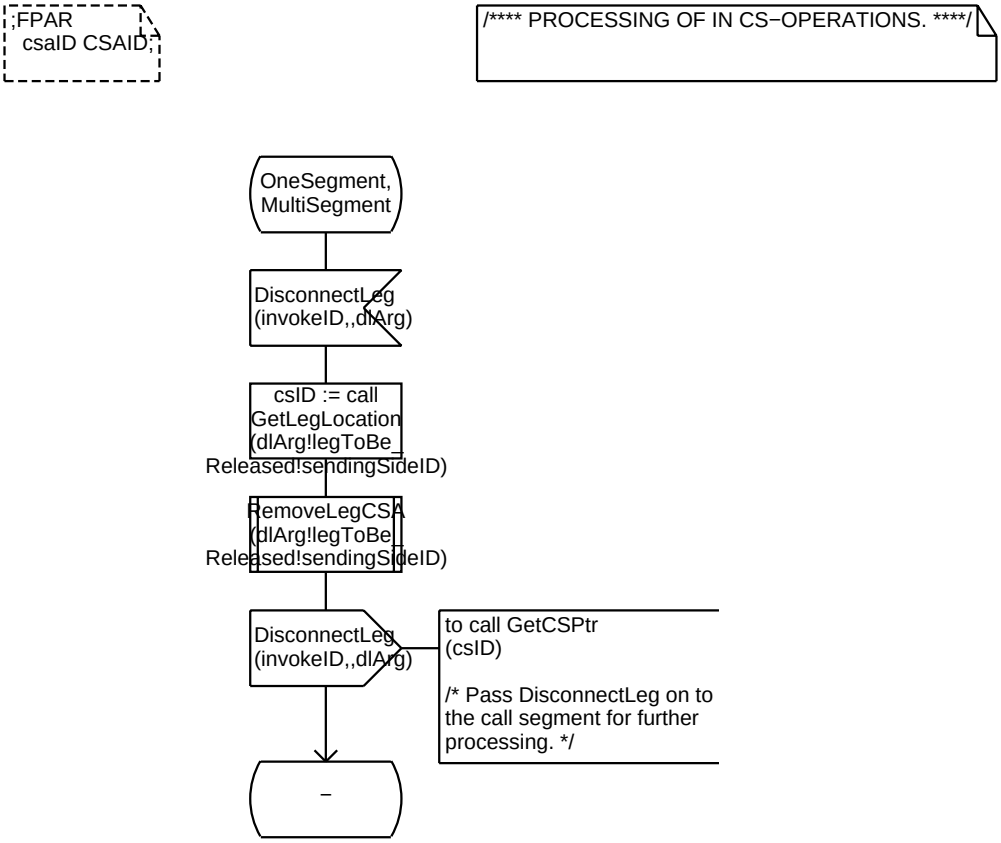
Process Type CallSegmentAssociation 13(29)



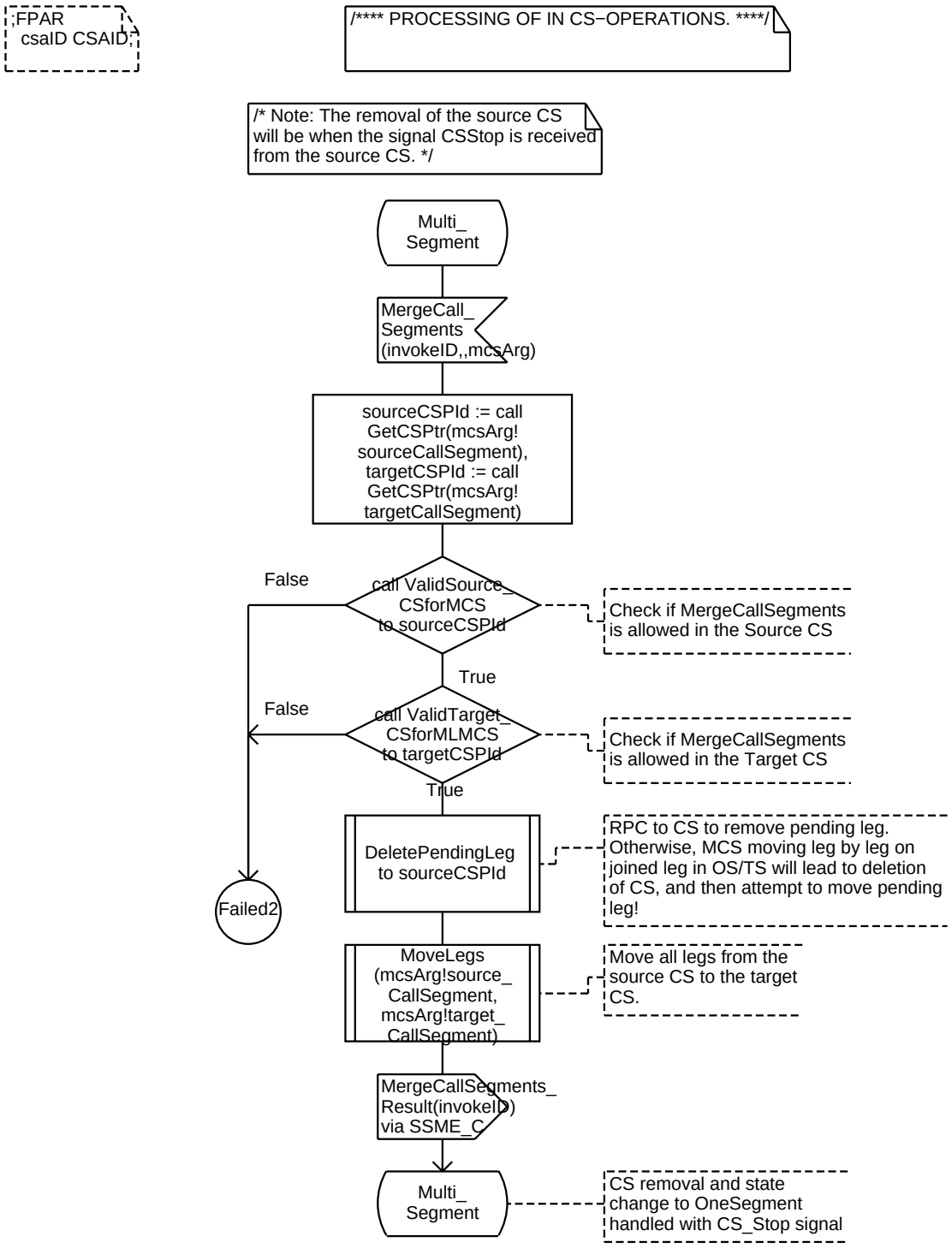
Process Type CallSegmentAssociation 14(29)



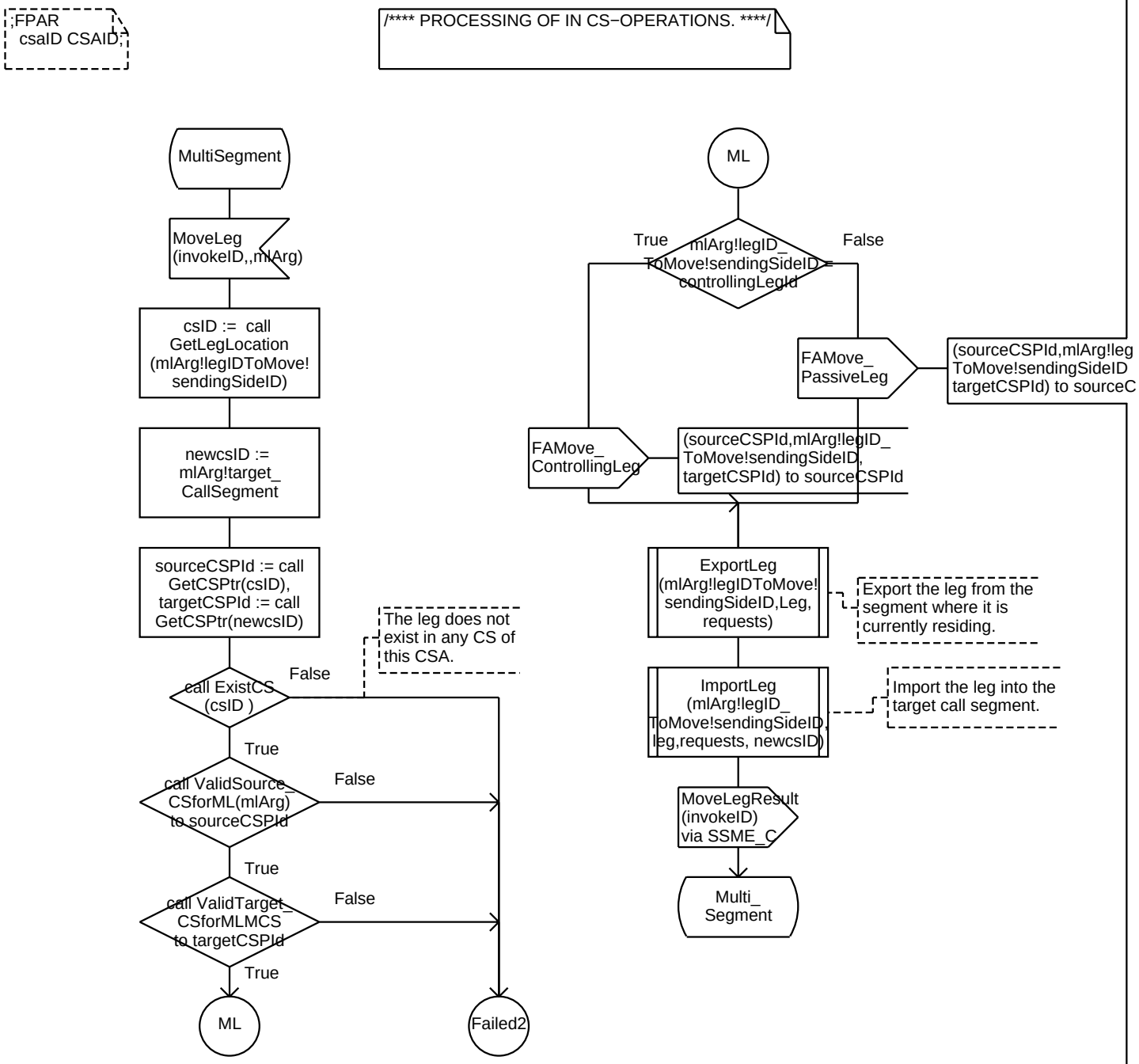
Process Type CallSegmentAssociation 15(29)



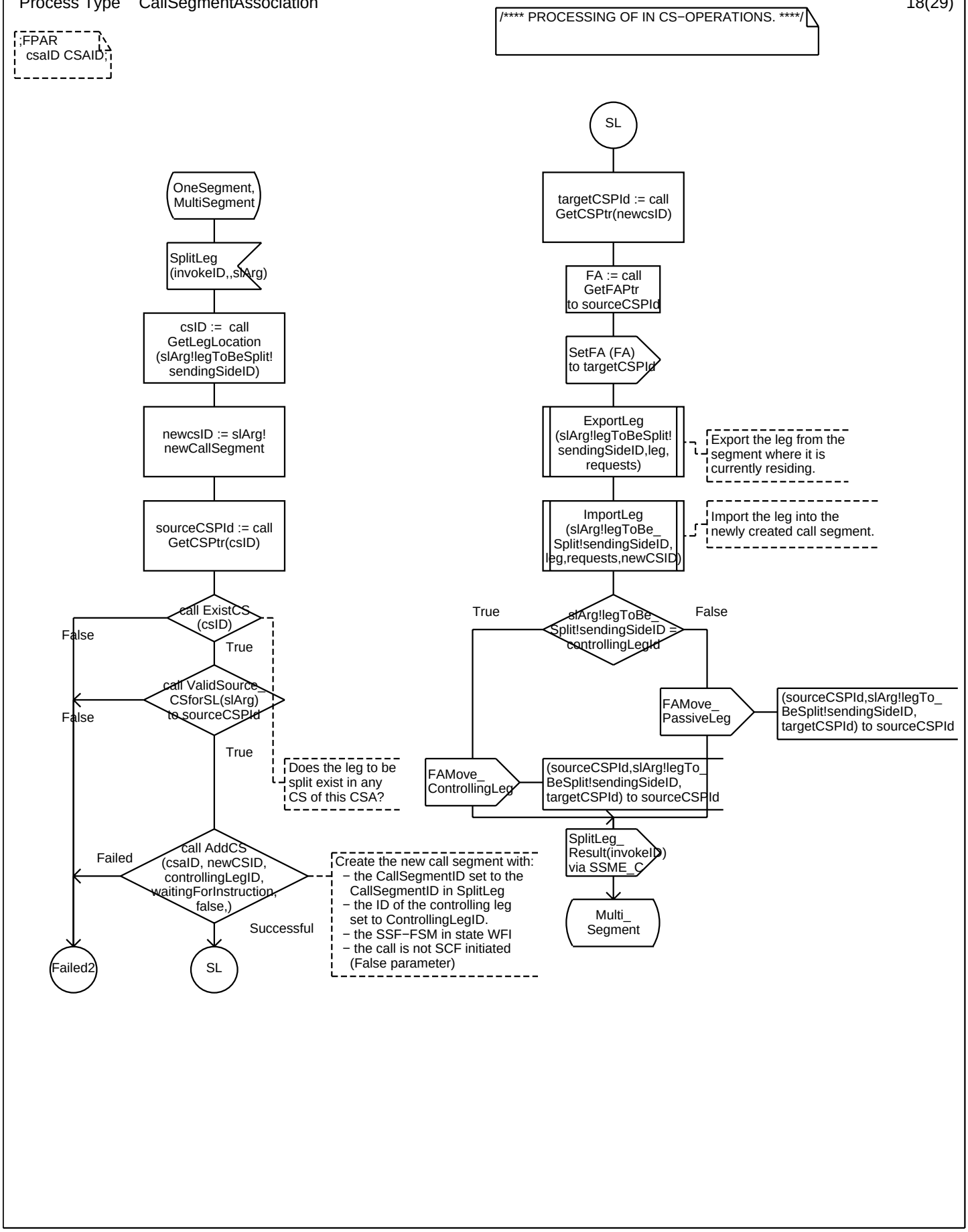
Process Type CallSegmentAssociation 16(29)



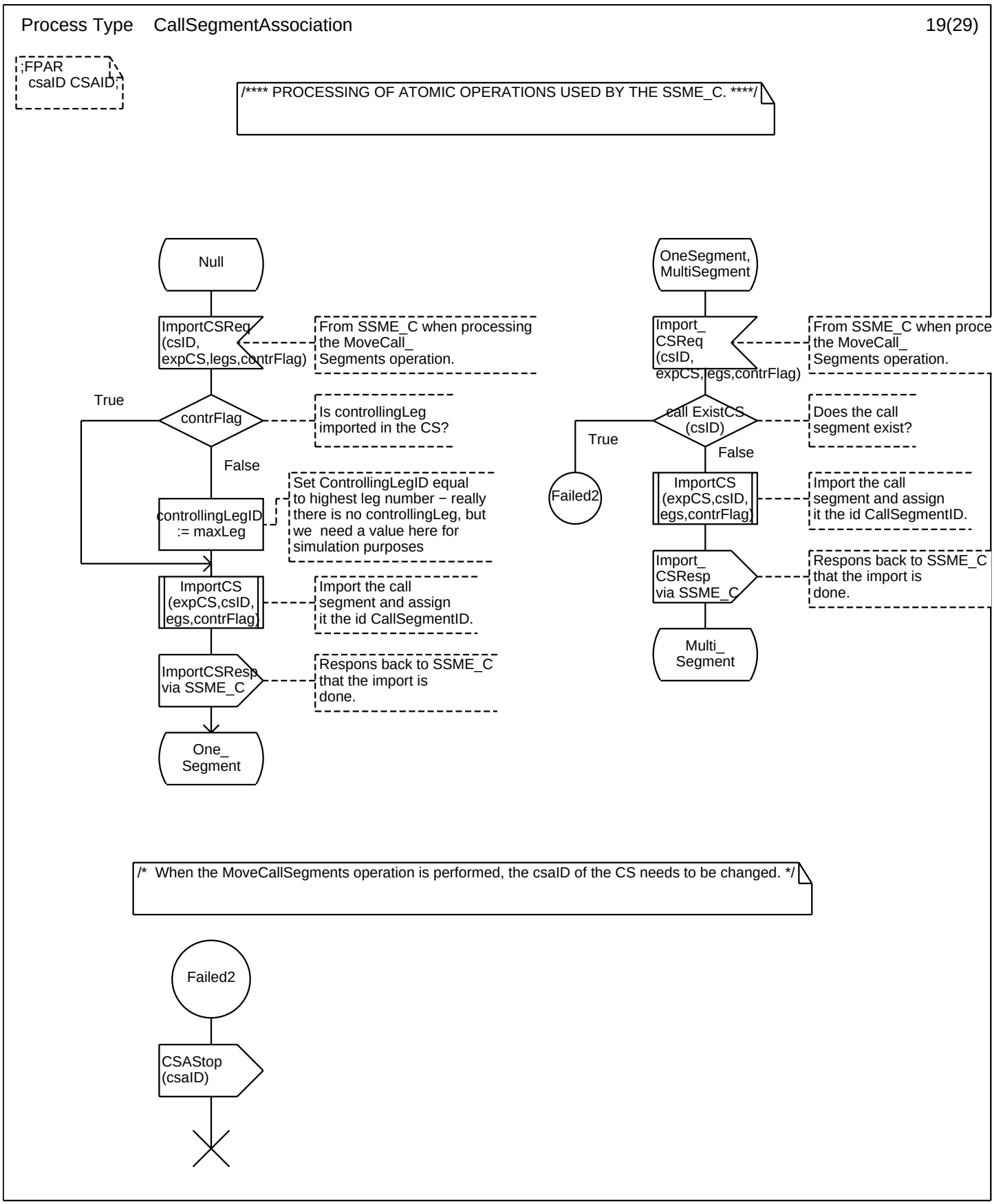
Process Type CallSegmentAssociation 17(29)



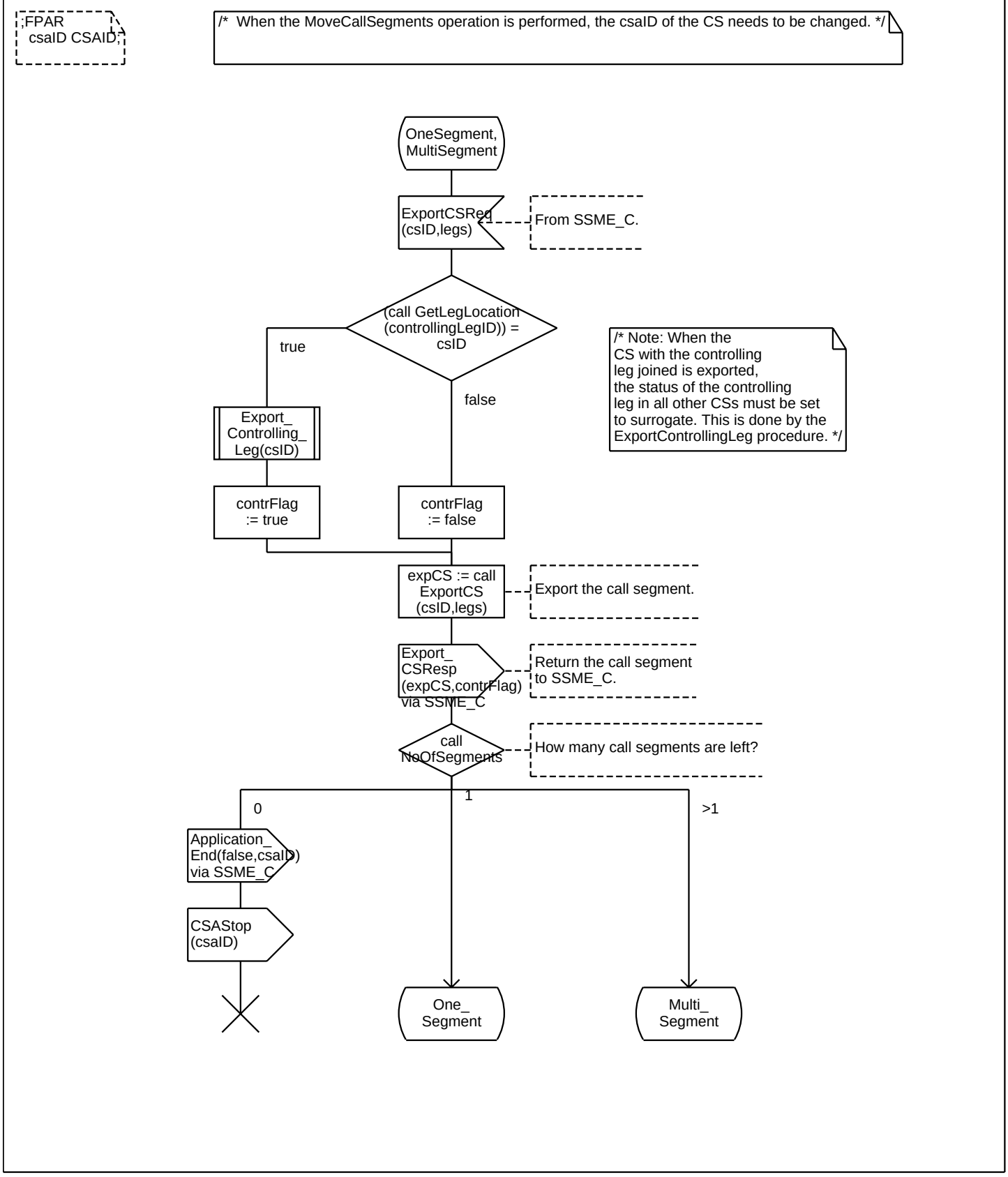
Process Type CallSegmentAssociation 18(29)



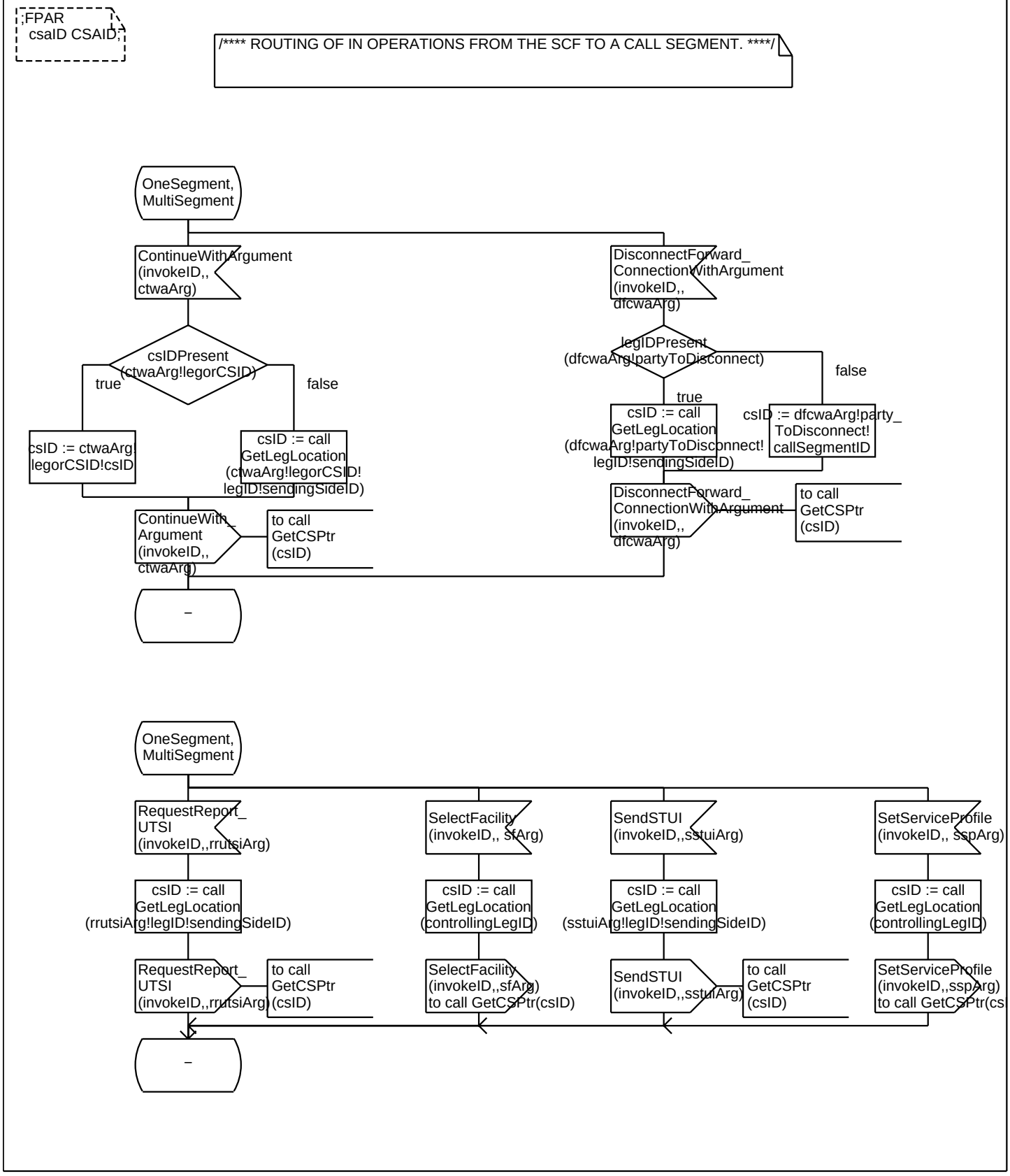
Process Type CallSegmentAssociation 19(29)



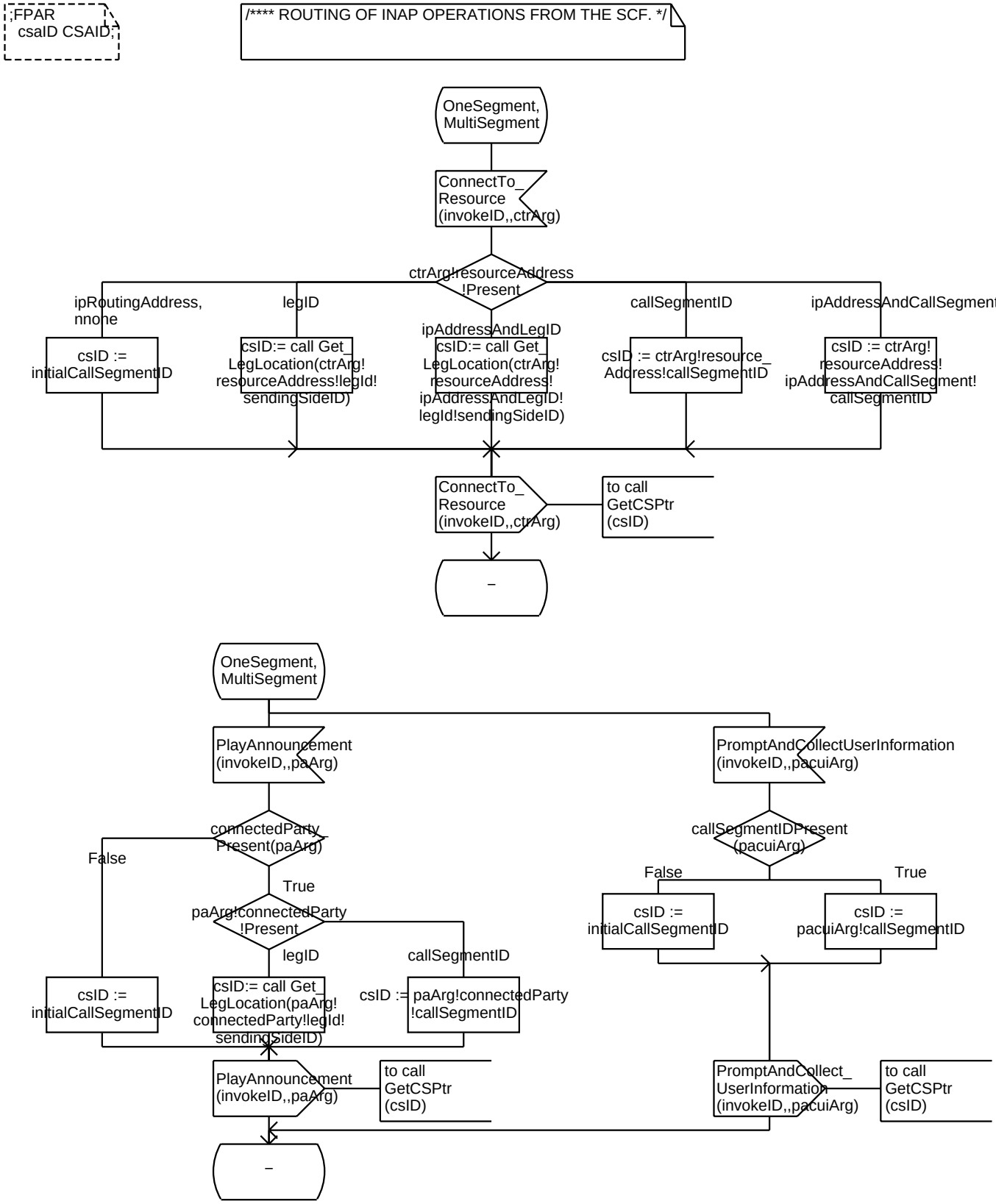
Process Type CallSegmentAssociation 20(29)



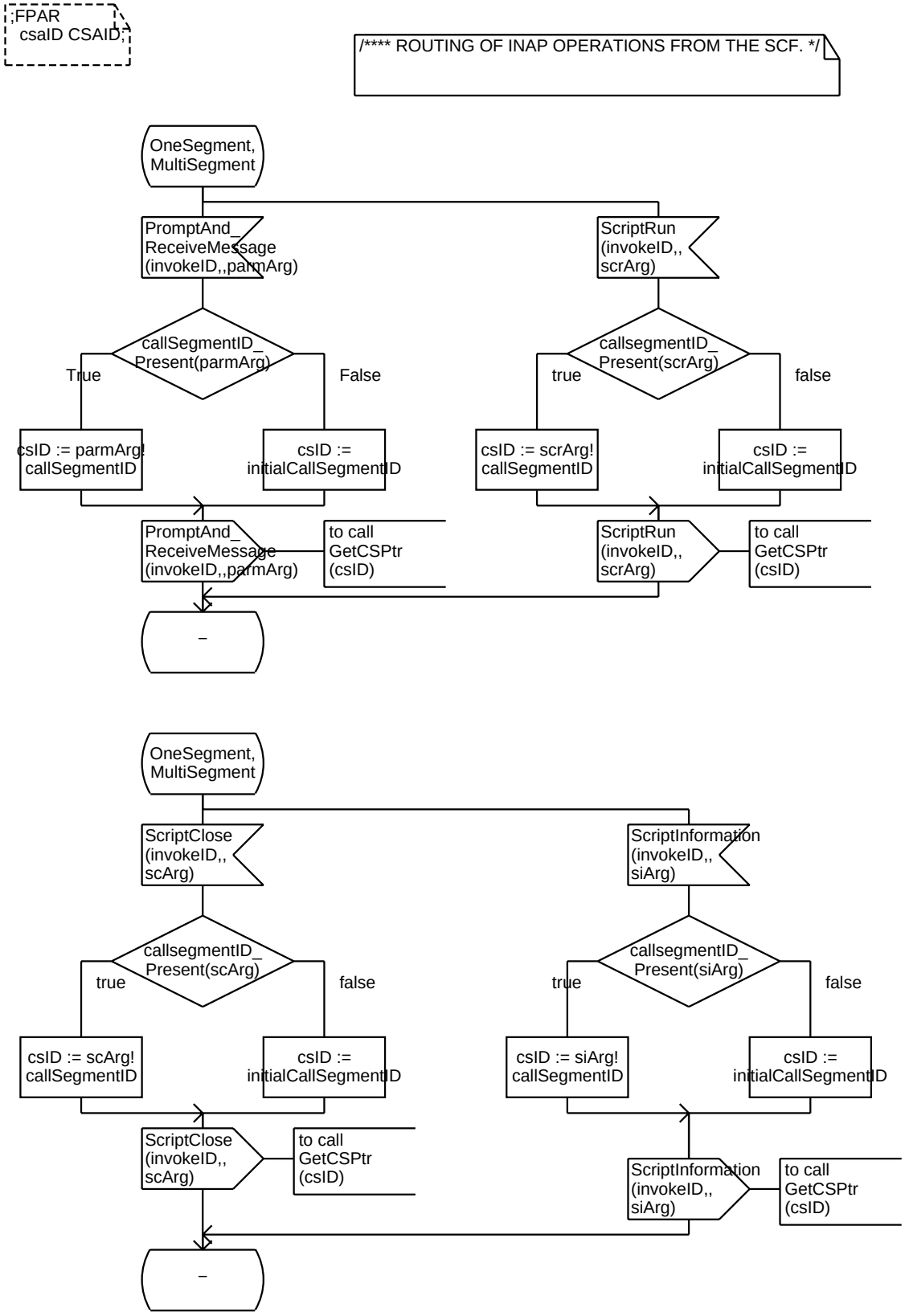
Process Type CallSegmentAssociation 21(29)



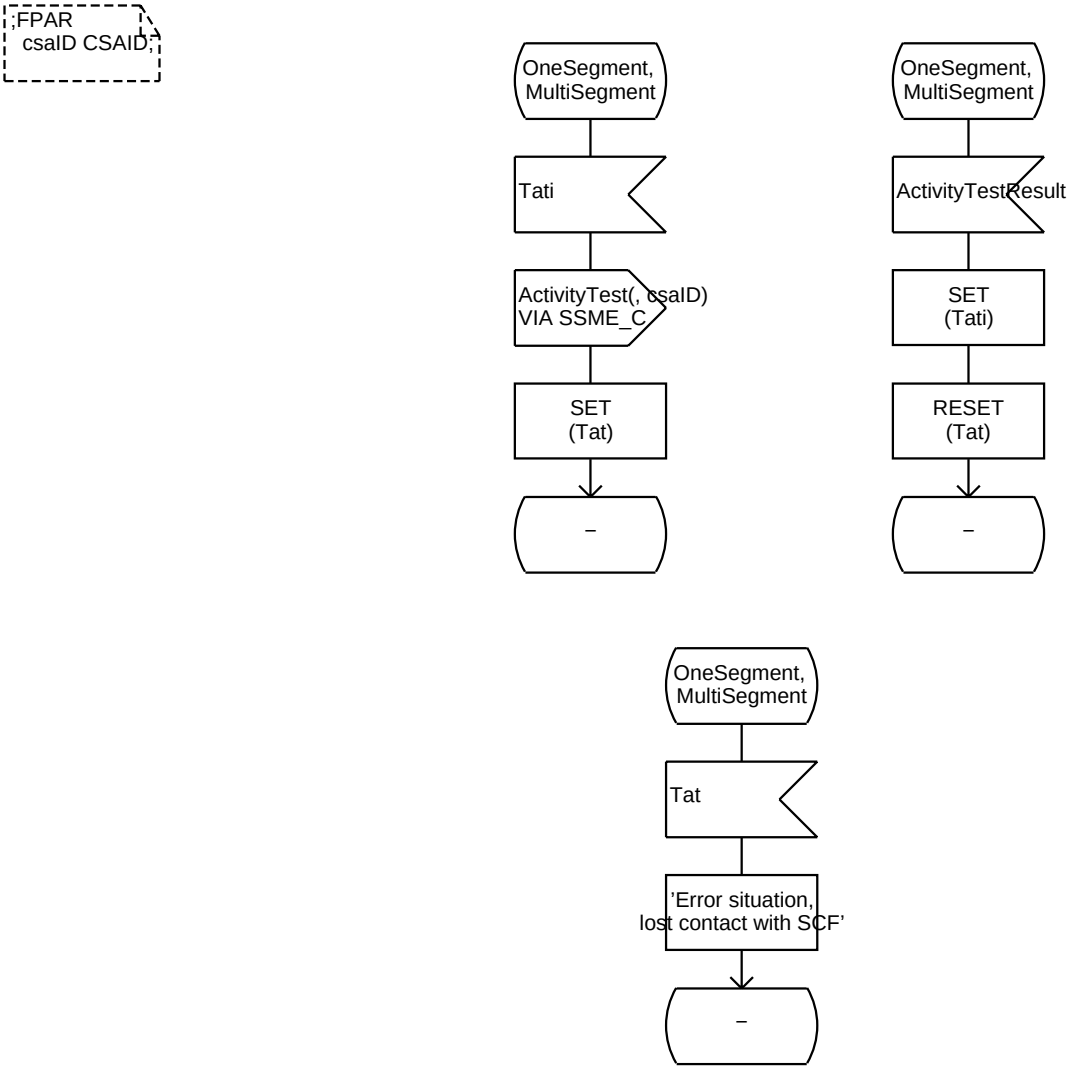
Process Type CallSegmentAssociation 22(29)



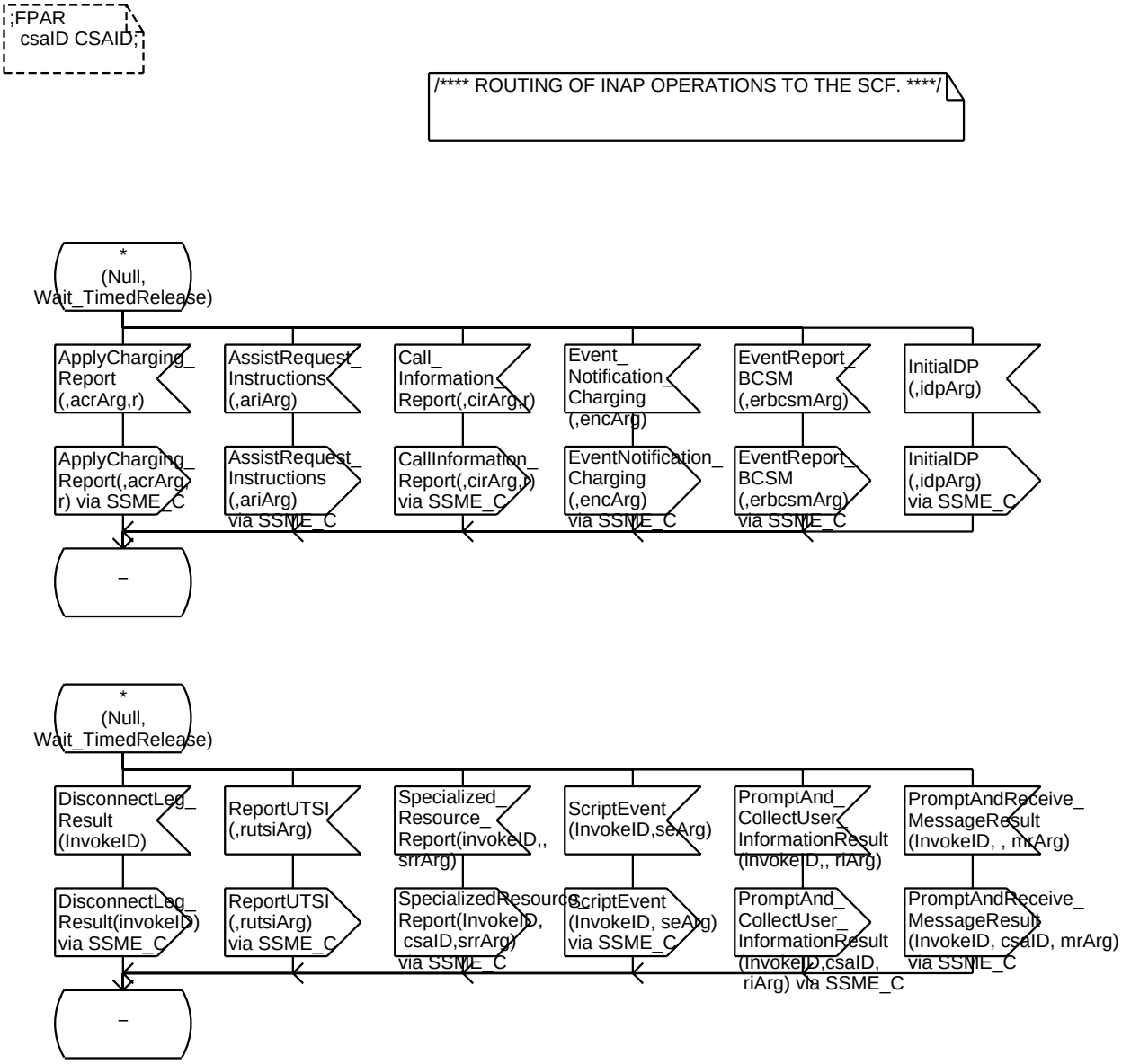
Process Type CallSegmentAssociation 23(29)



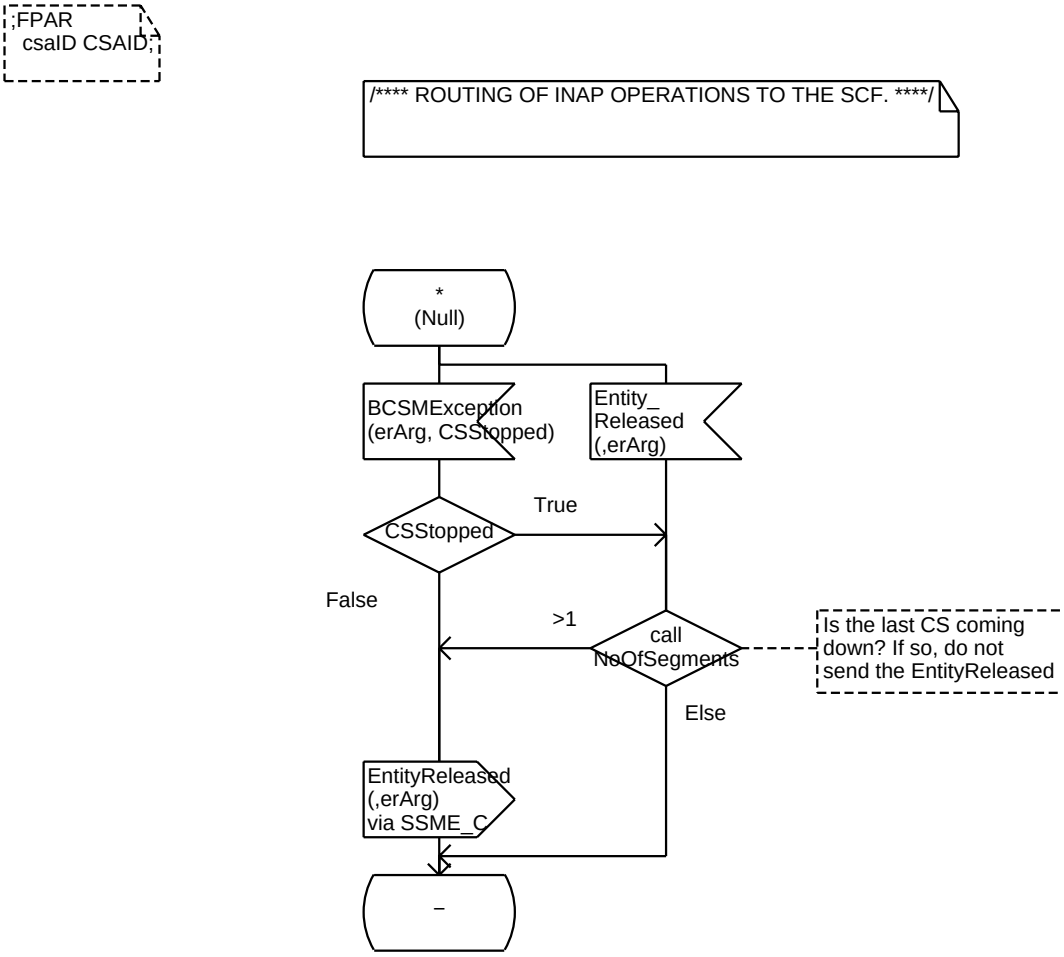
Process Type CallSegmentAssociation 24(29)



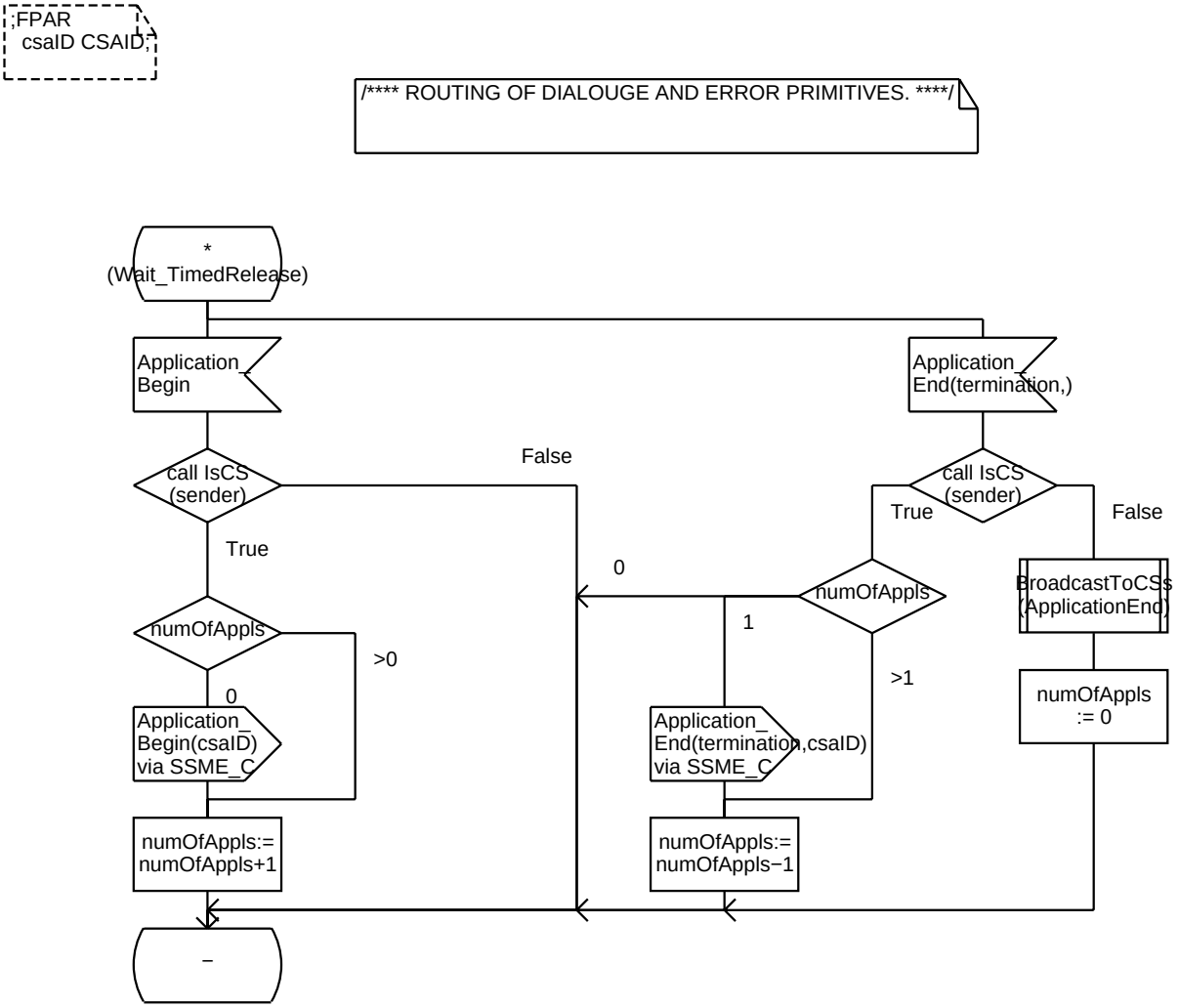
Process Type CallSegmentAssociation 25(29)



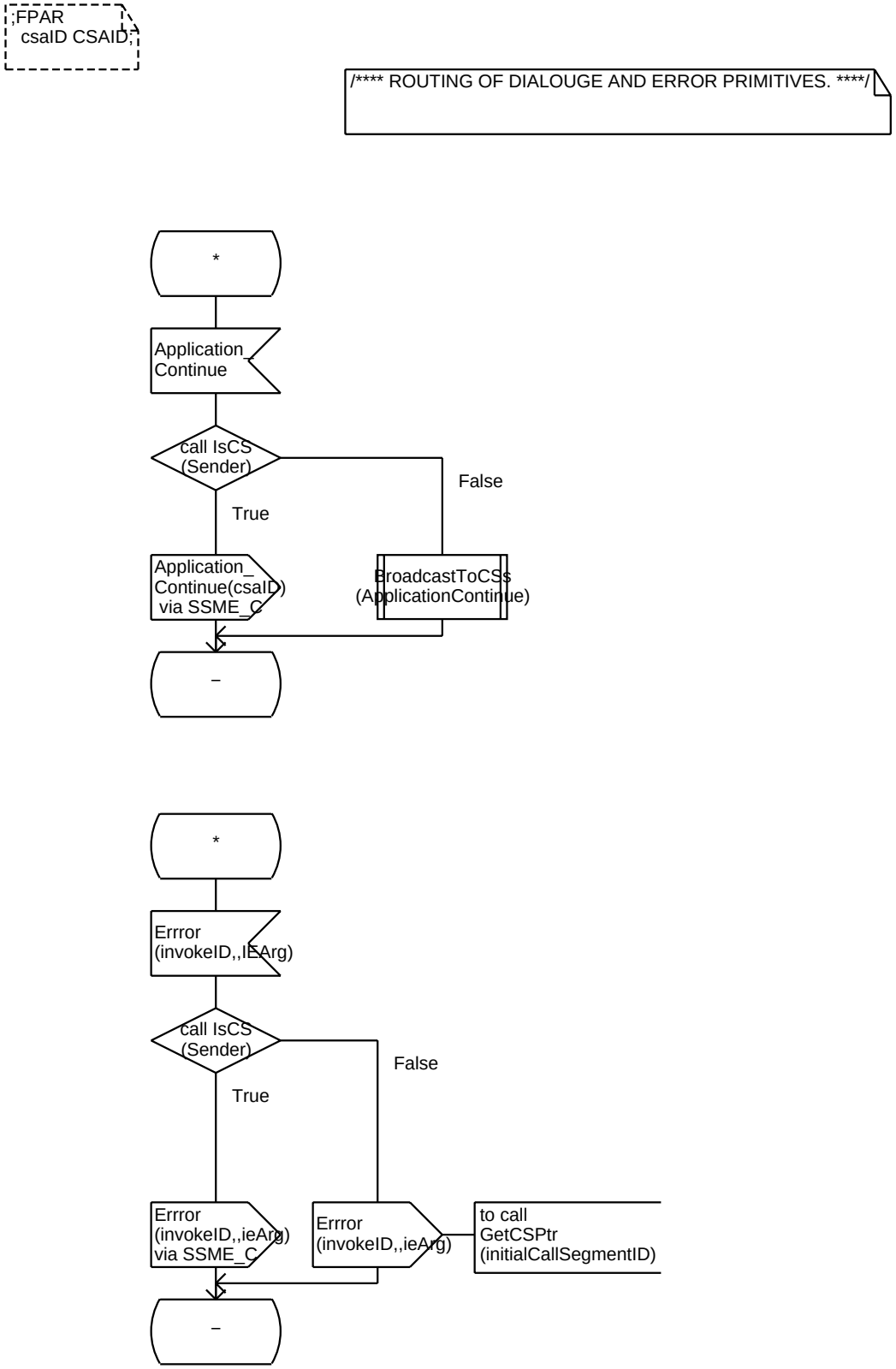
Process Type CallSegmentAssociation 26(29)



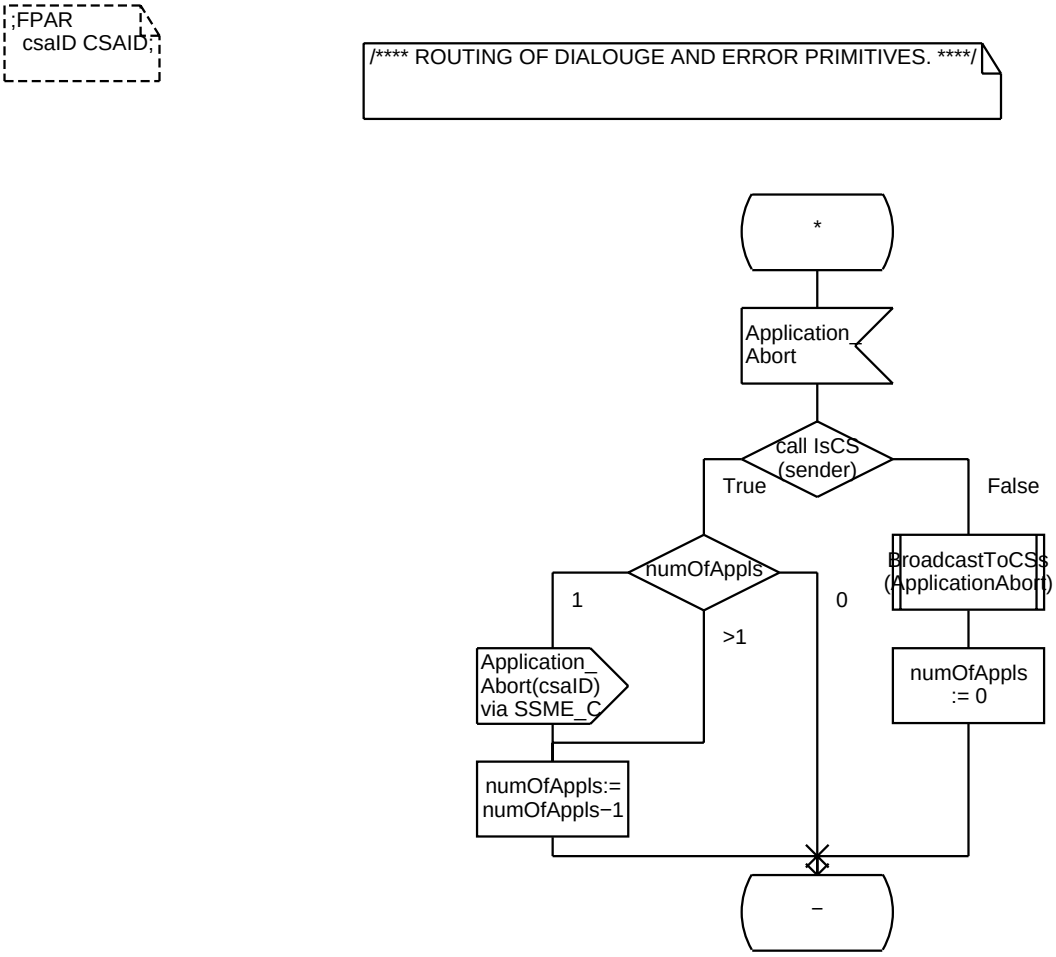
Process Type CallSegmentAssociation 27(29)



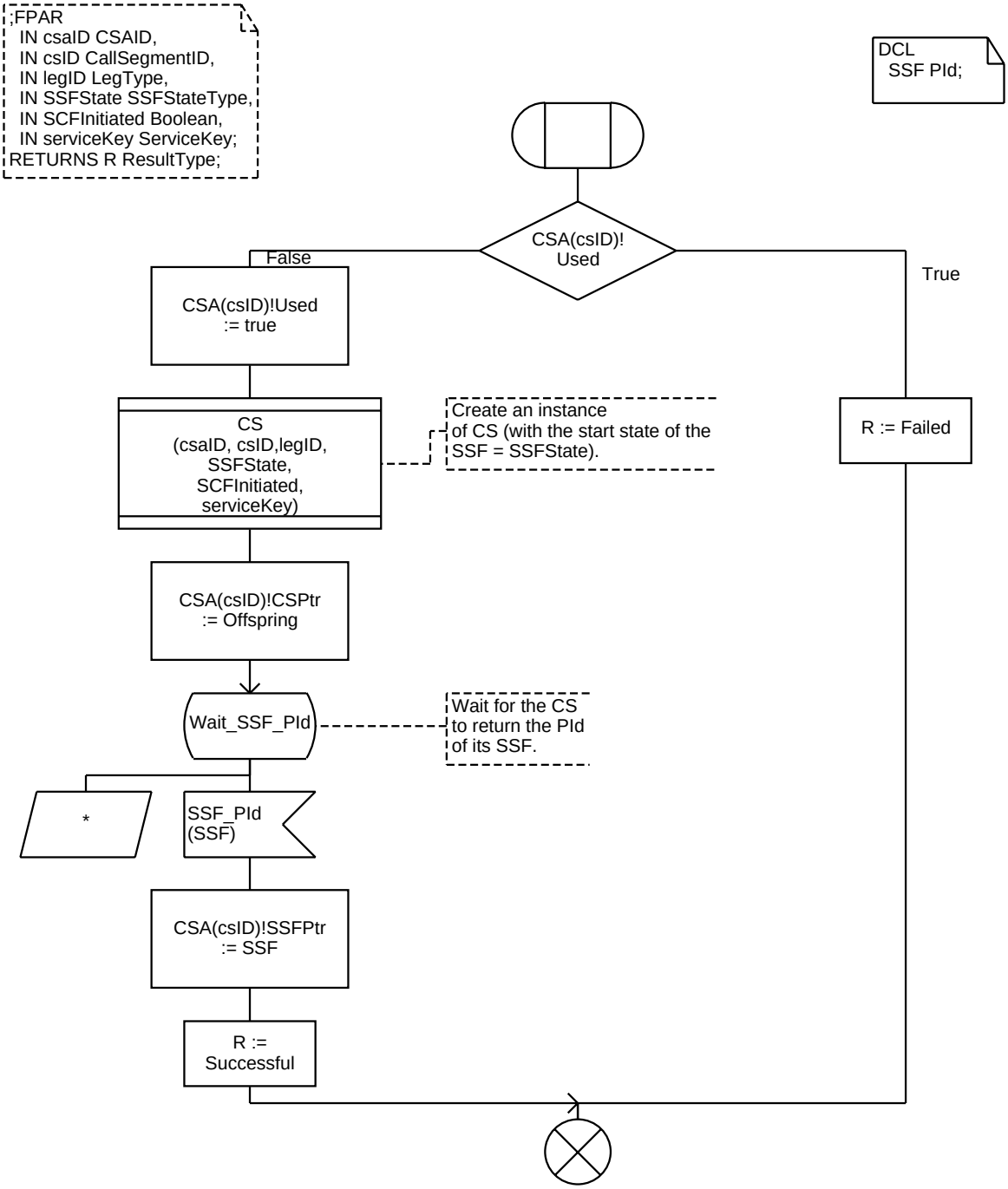
Process Type CallSegmentAssociation 28(29)



Process Type CallSegmentAssociation 29(29)

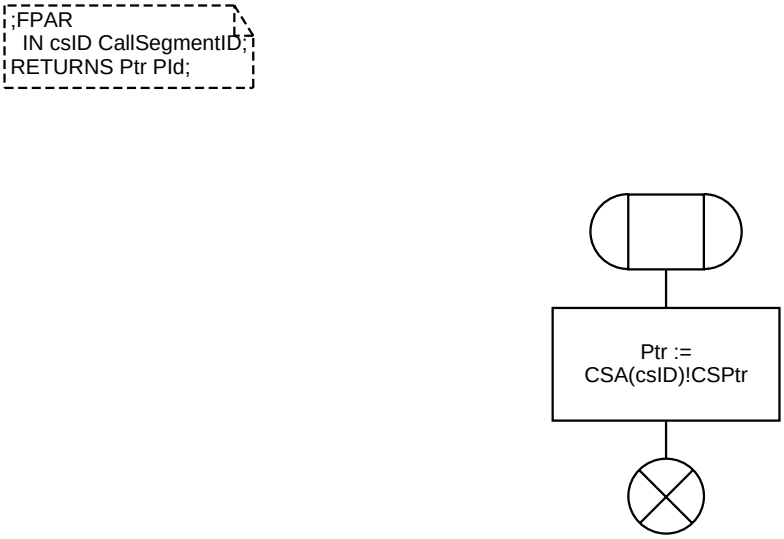


Procedure <<Block Type SSF_CCF/Process Type CallSegmentAssociation>> AddCS 1(1)



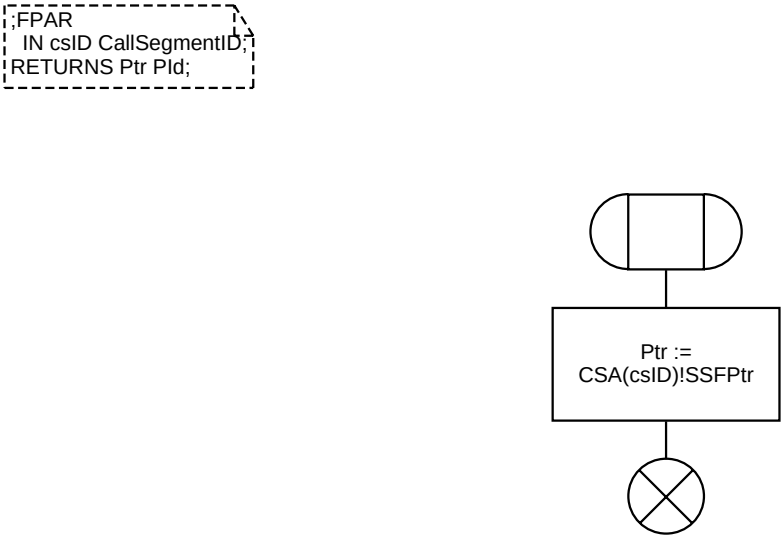
Procedure SetLegLocation 1(1)

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

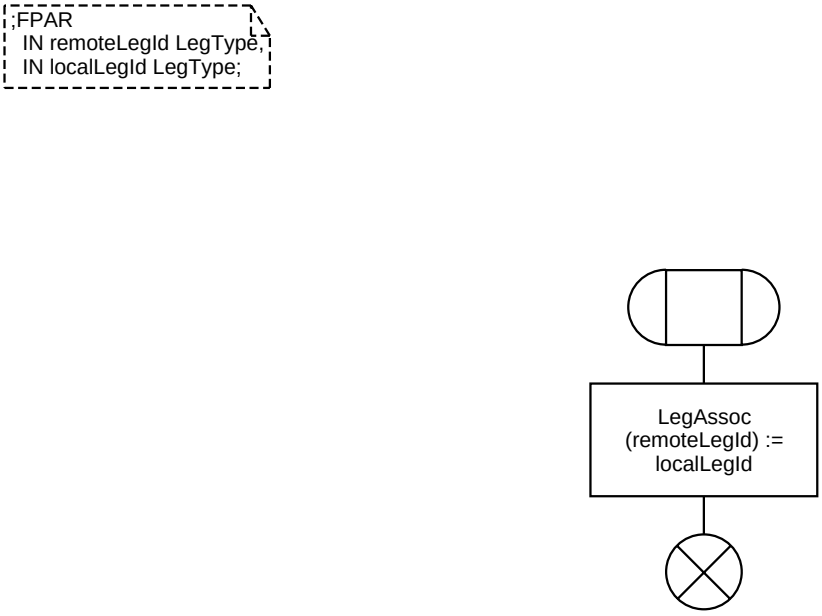


Procedure GetLegLocation 1(1)

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

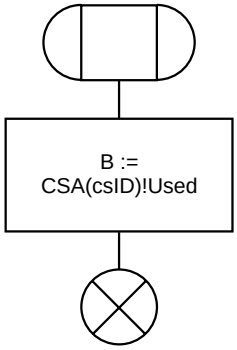


Procedure <<Process Type CallSegmentAssociation>> SetLegAssoc 1(1)



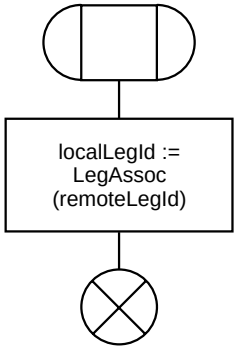
Procedure <<Block Type SSF_CCF/Process Type CallSegmentAssociation>> ExistCS 1(1)

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

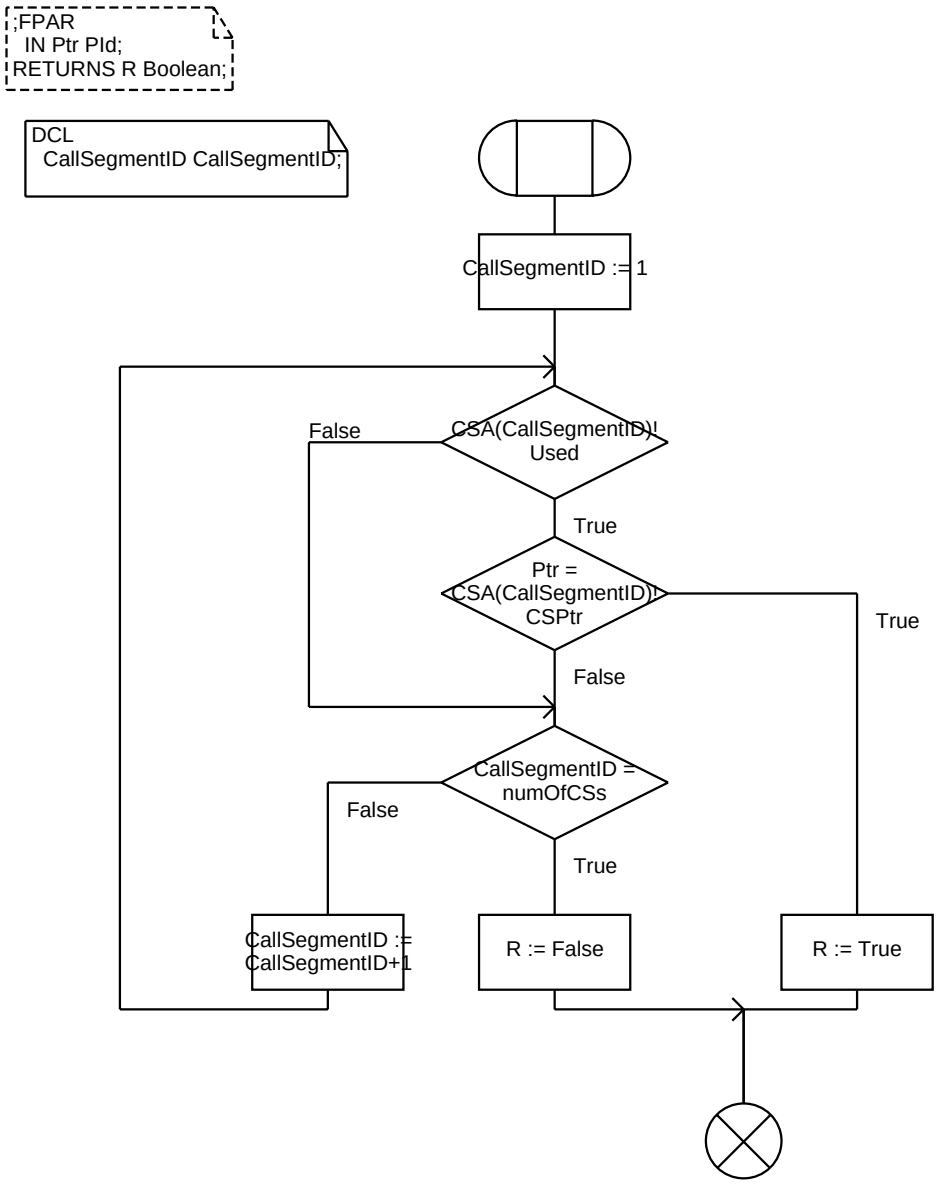


Procedure <<Process Type CallSegmentAssociation>> GetLegIdfromRemoteLegId 1(1)

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

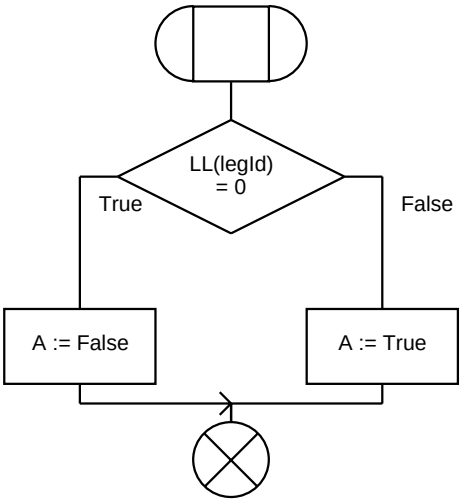


Procedure IsCS 1(1)

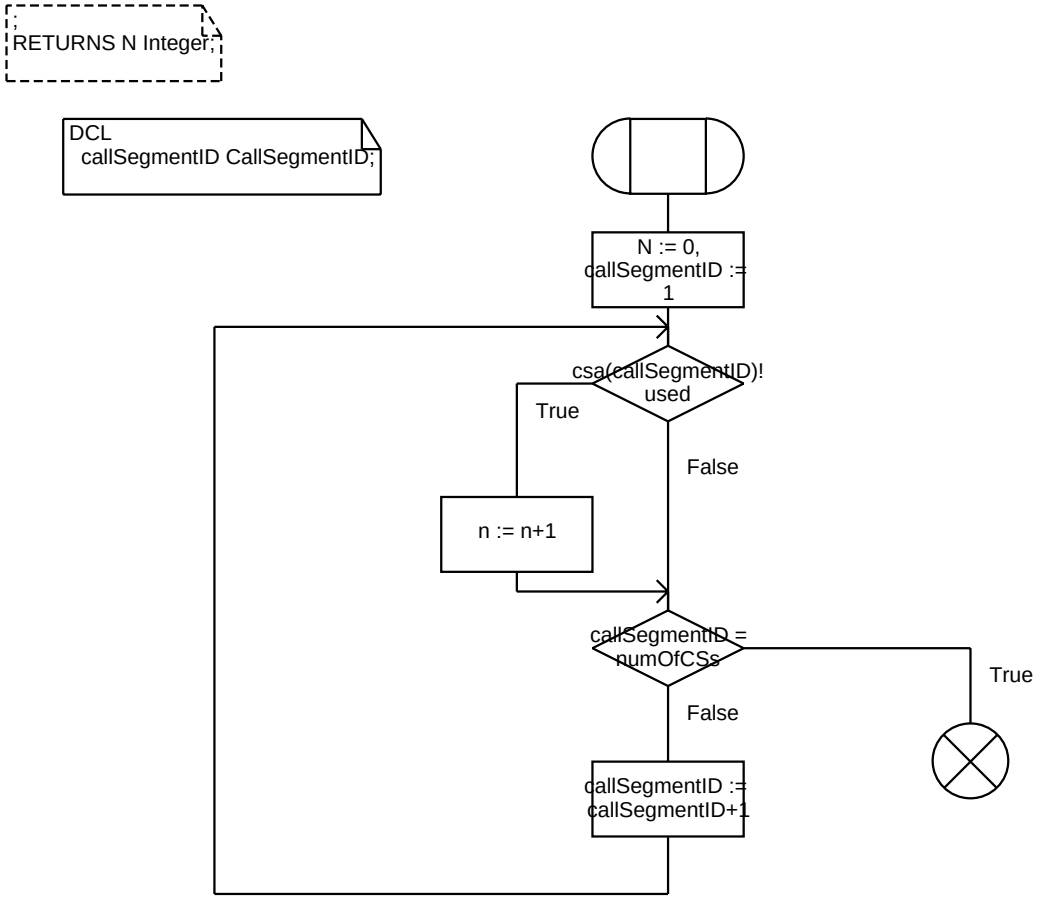


Procedure <<Process Type CallSegmentAssociation>> ExistLeg 1(1)

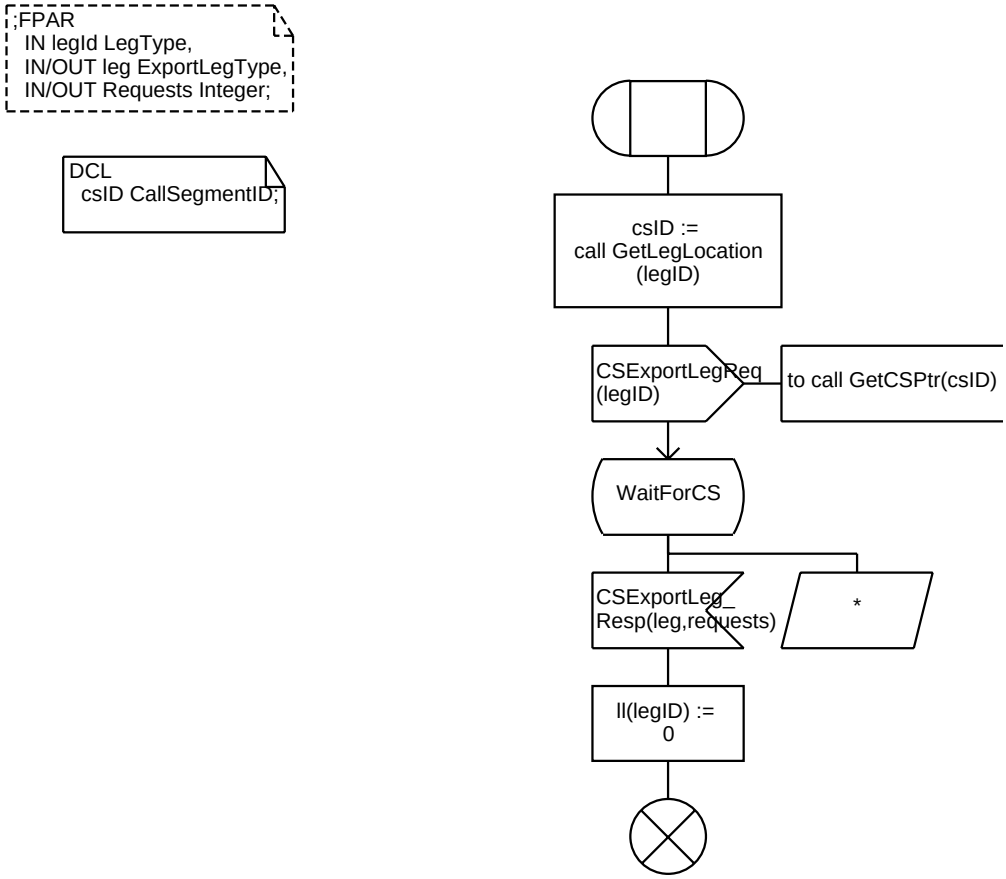
```
;FPAR
  IN legId LegType;
RETURNS A Boolean;
```



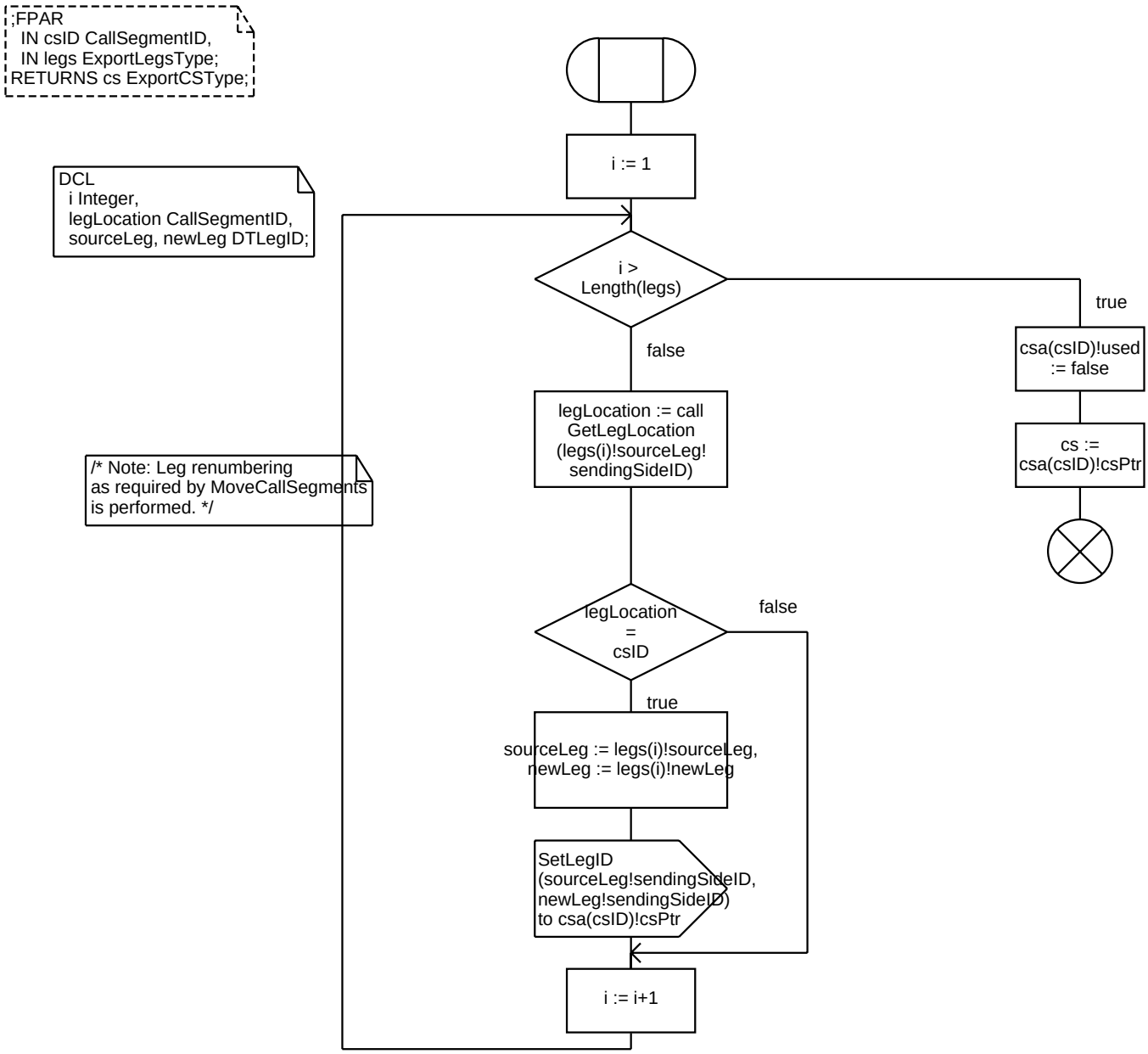
Procedure NoOfSegments 1(1)



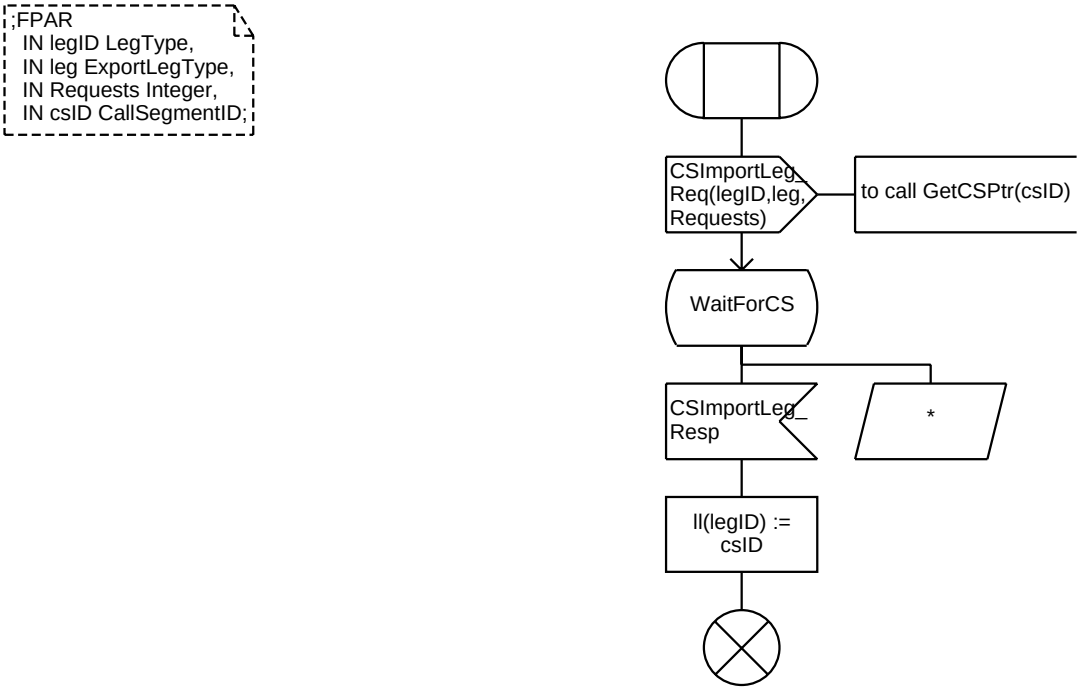
Procedure <<Process Type CallSegmentAssociation>> ExportLeg 1(1)



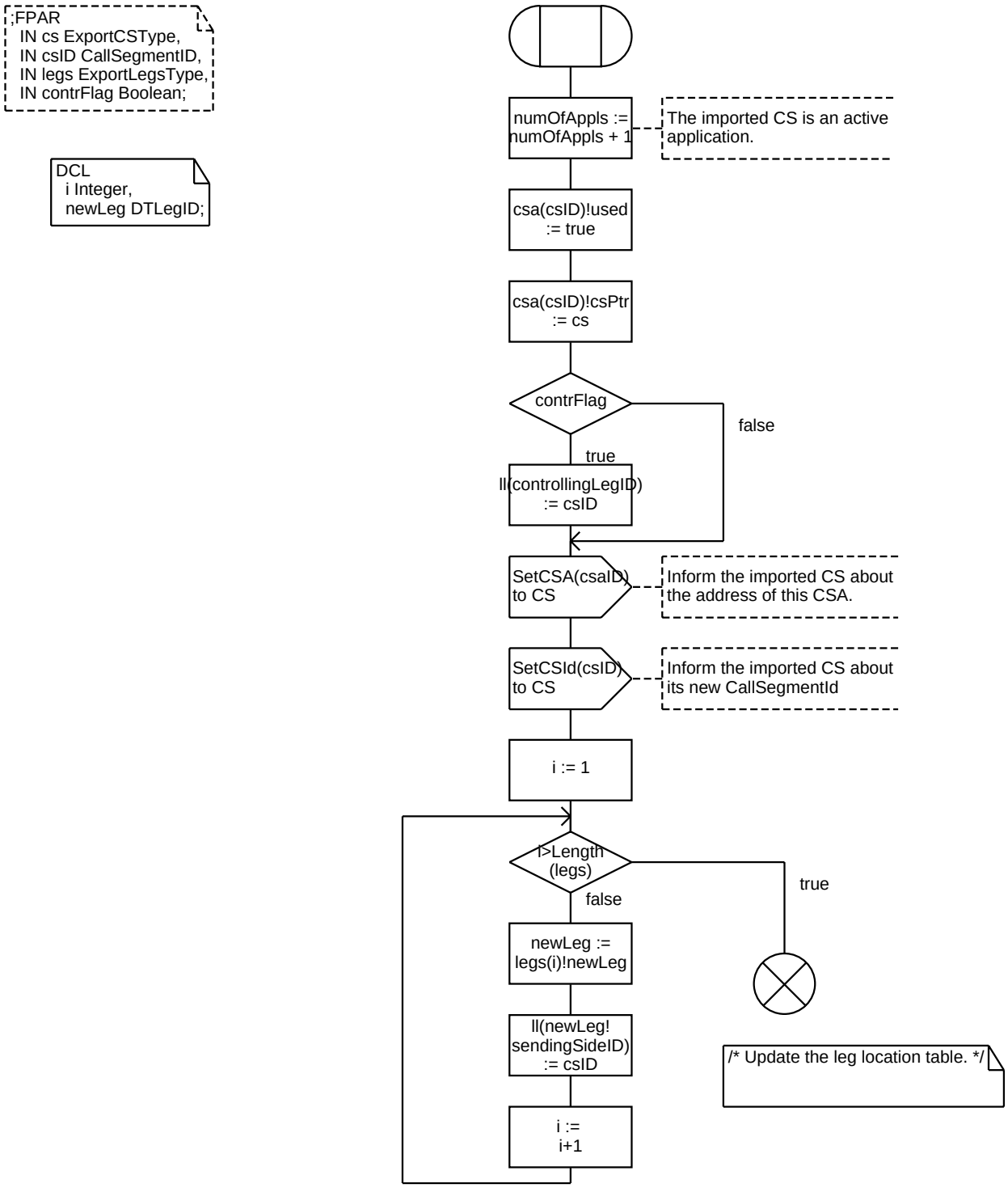
Procedure ExportCS 1(1)



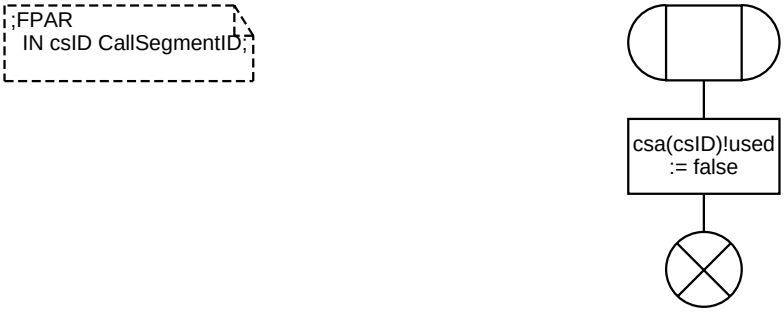
Procedure <<Process Type CallSegmentAssociation>> ImportLeg 1(1)



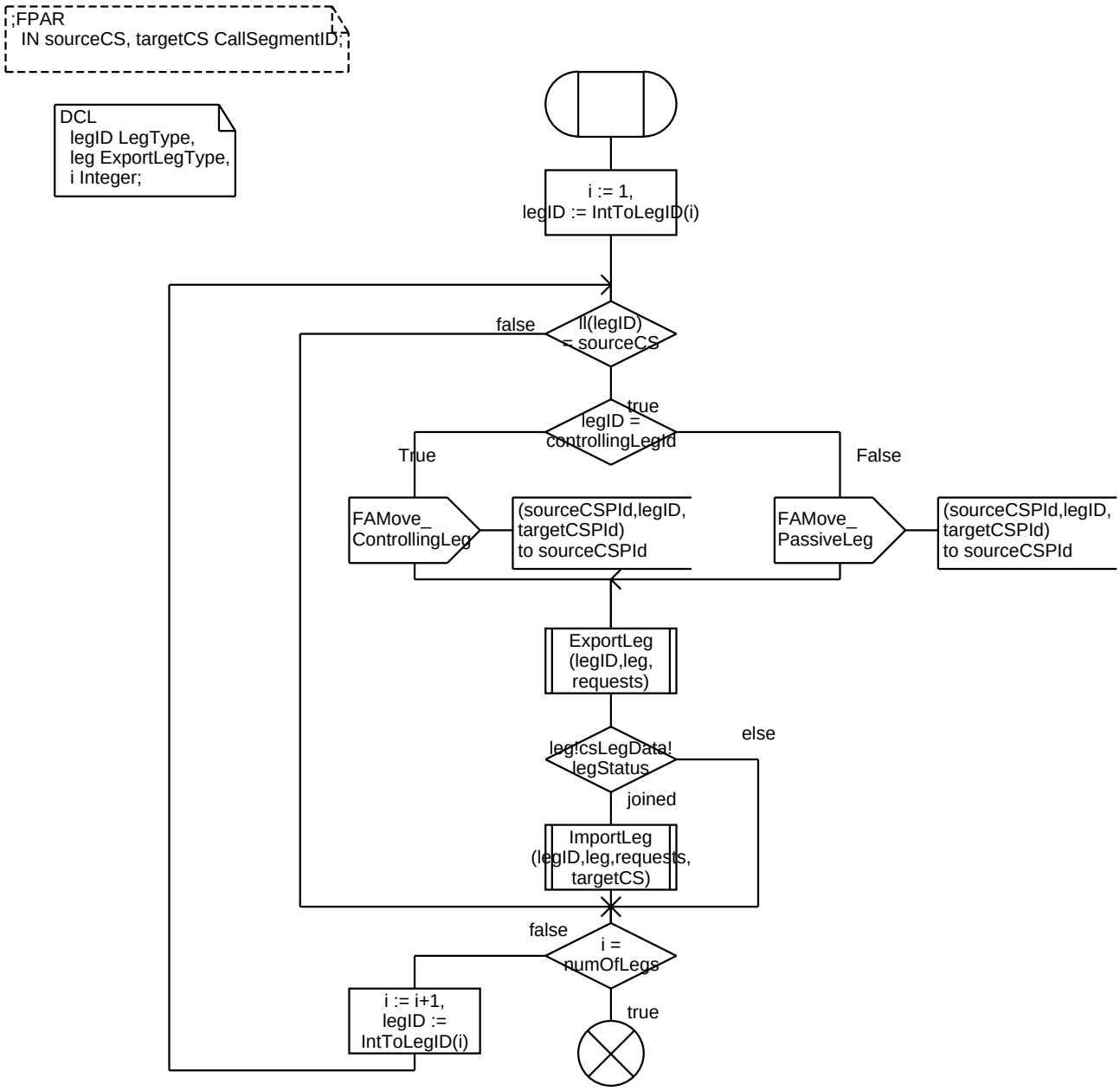
Procedure ImportCS 1(1)



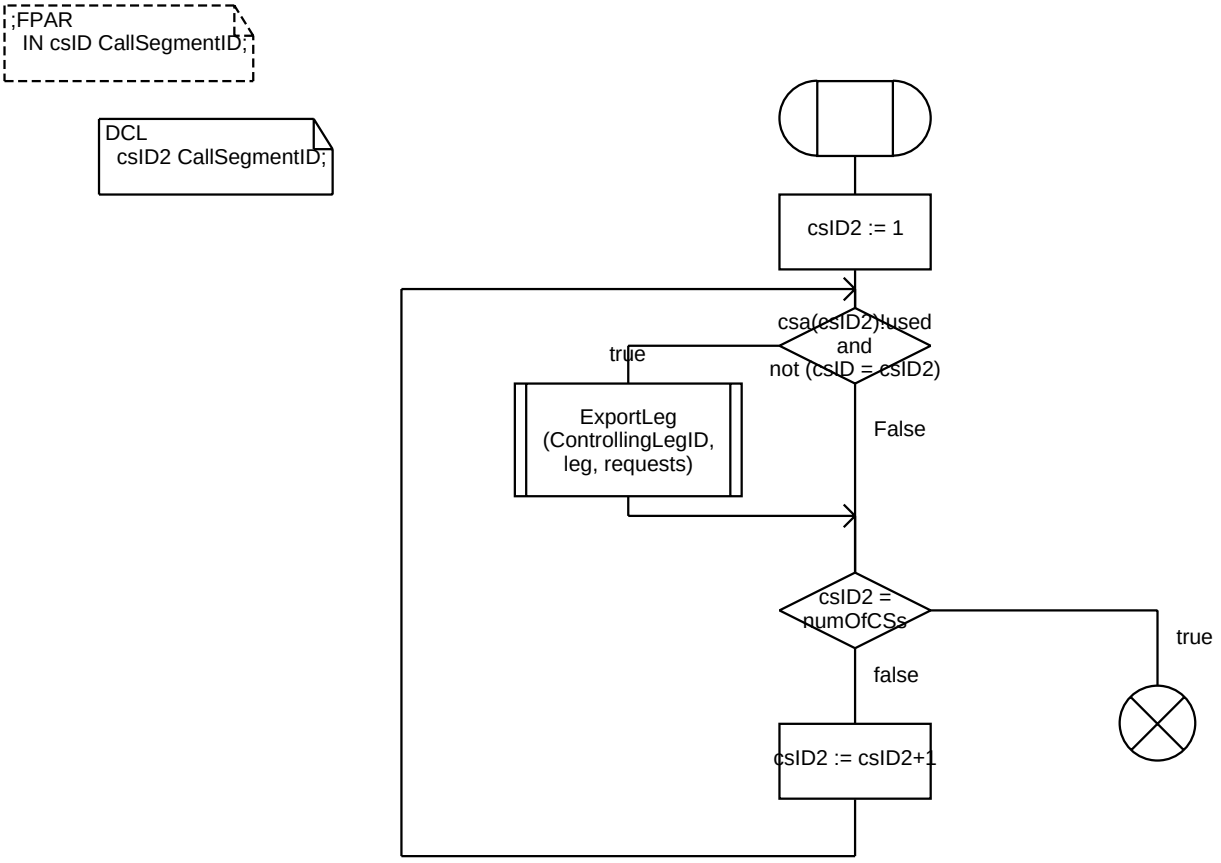
Procedure RemoveCS 1(1)



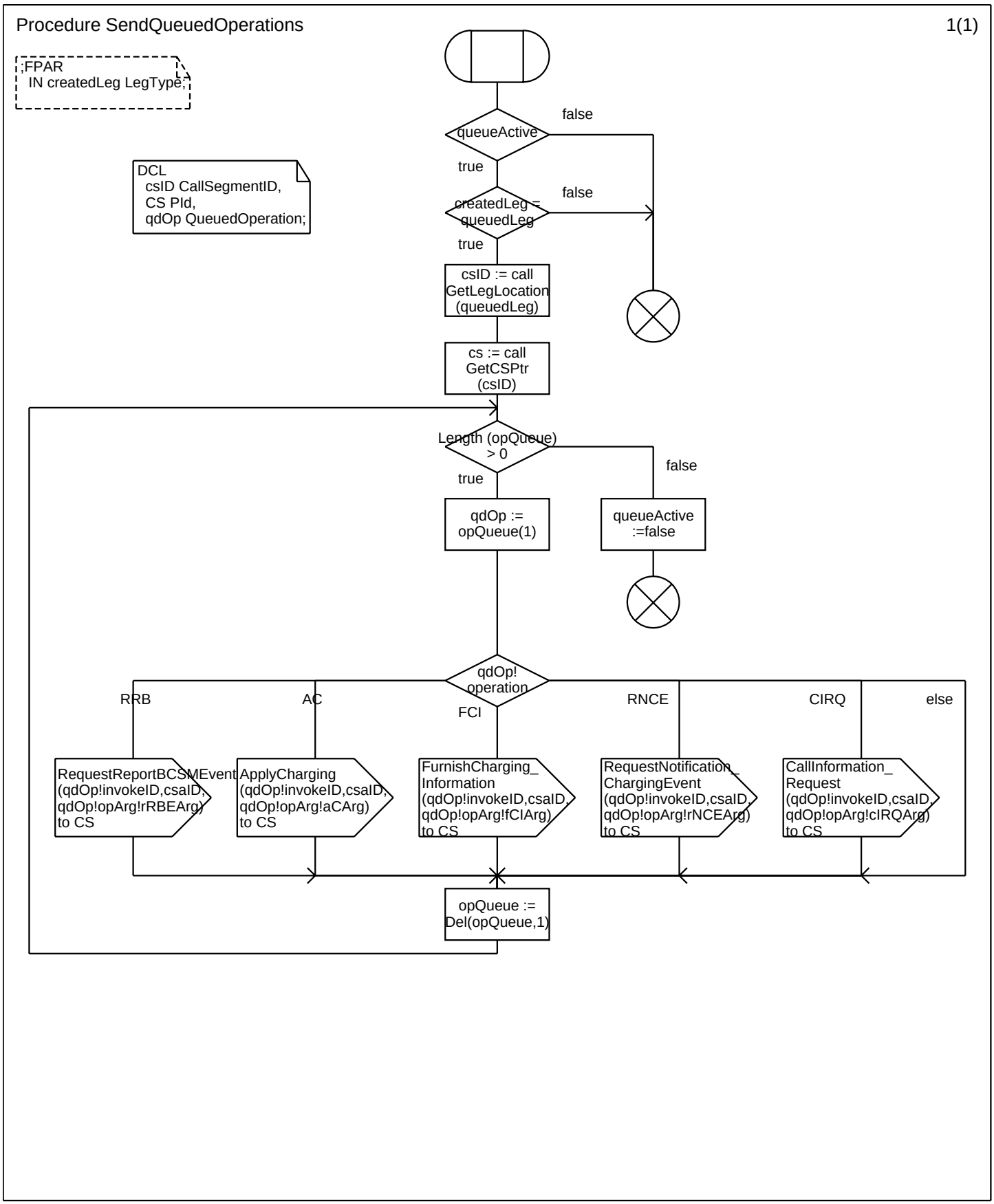
Procedure MoveLegs 1(1)



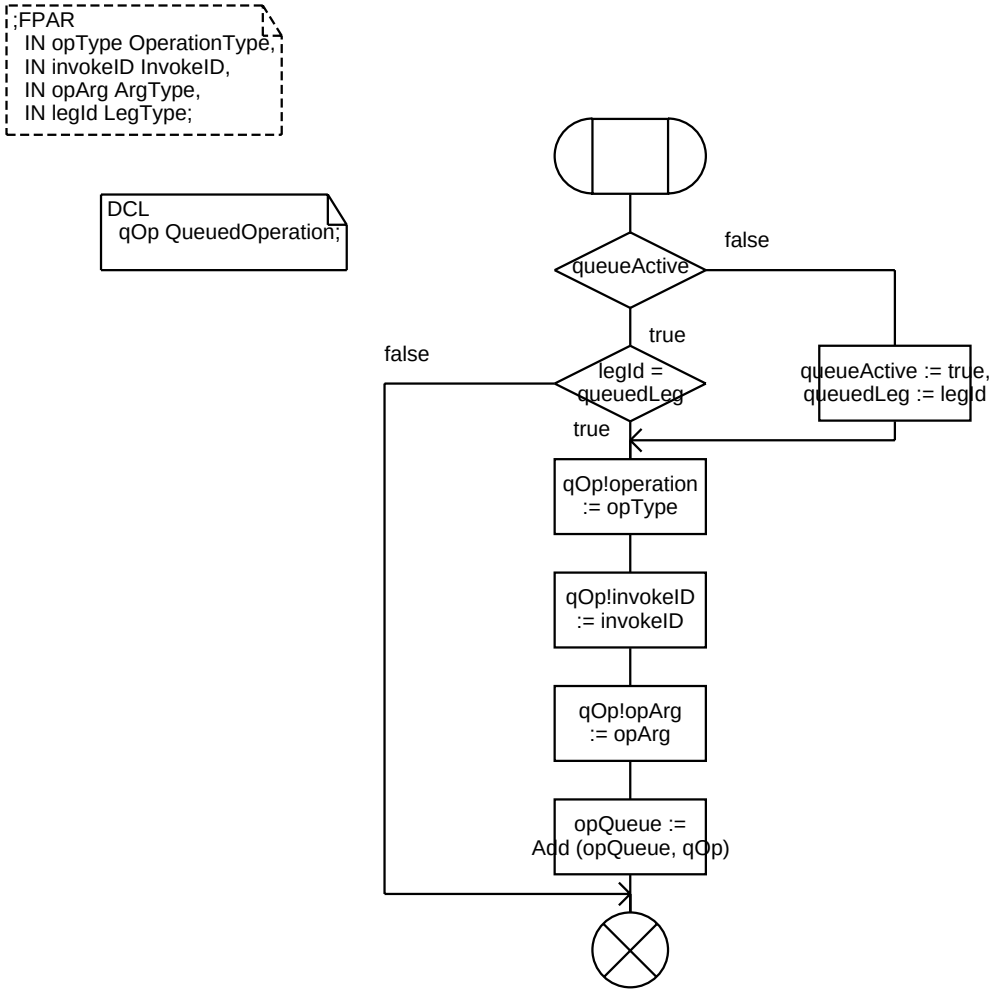
Procedure ExportControllingLeg 1(1)



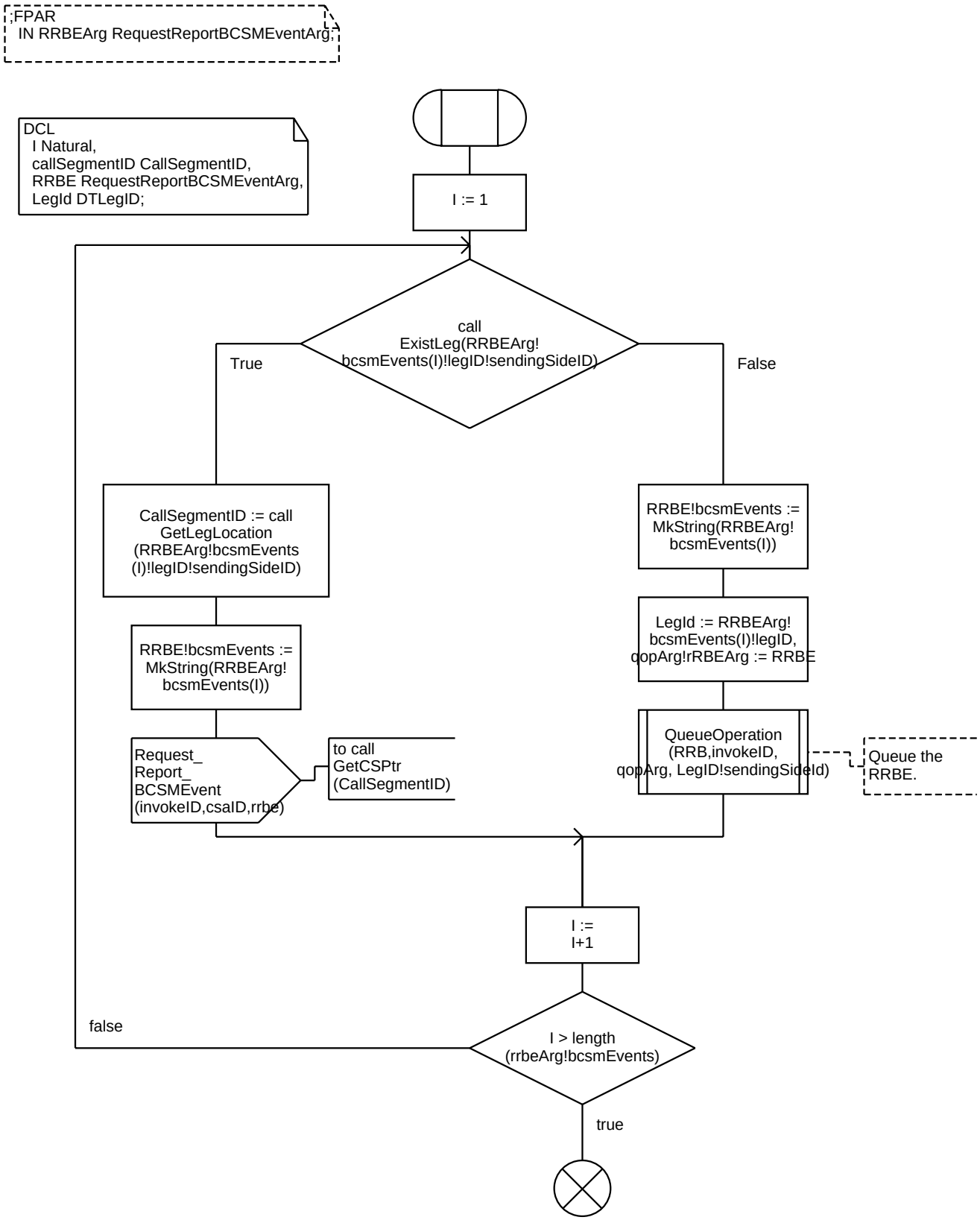
SendQueuedOperations

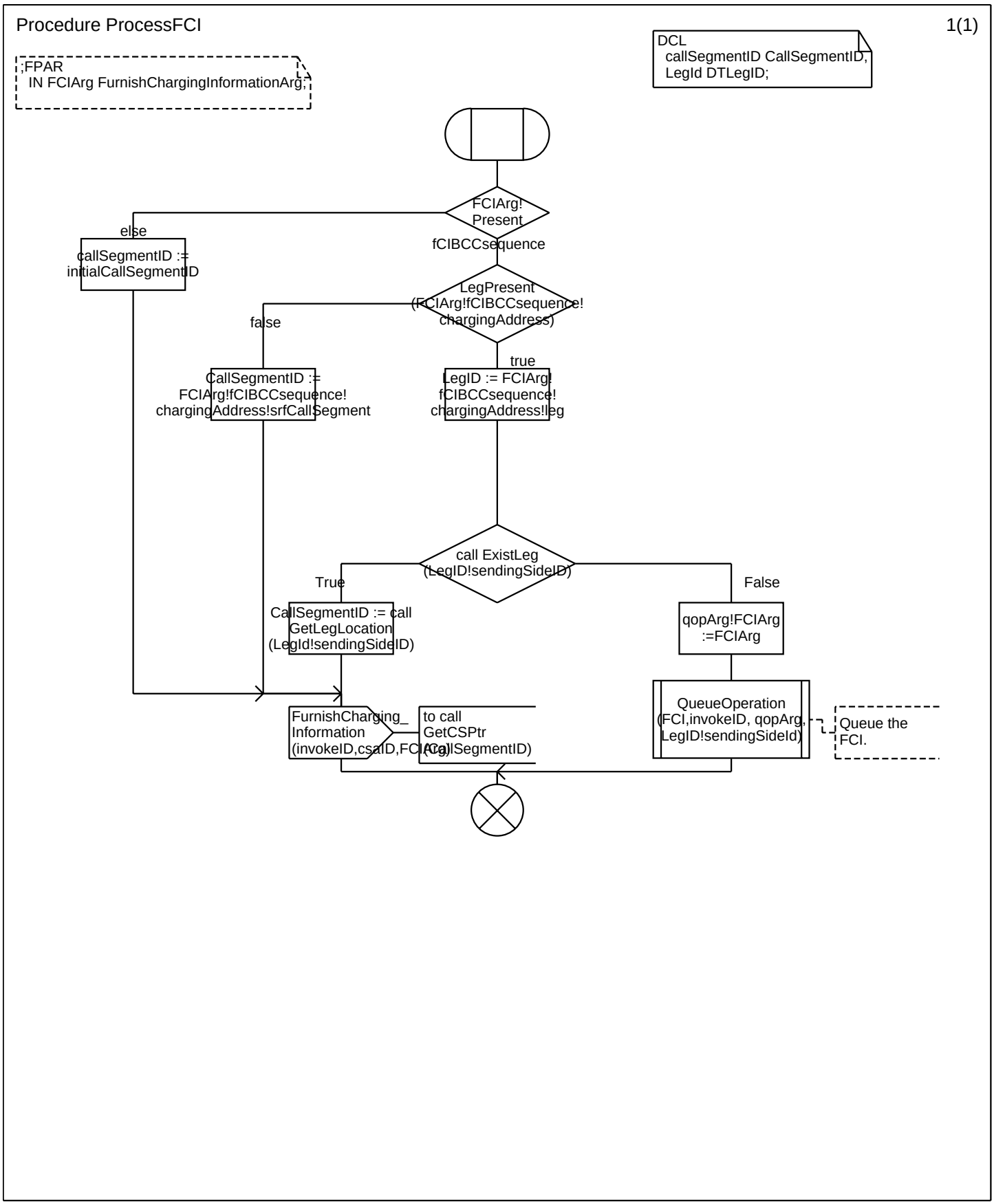


Procedure QueueOperation 1(1)

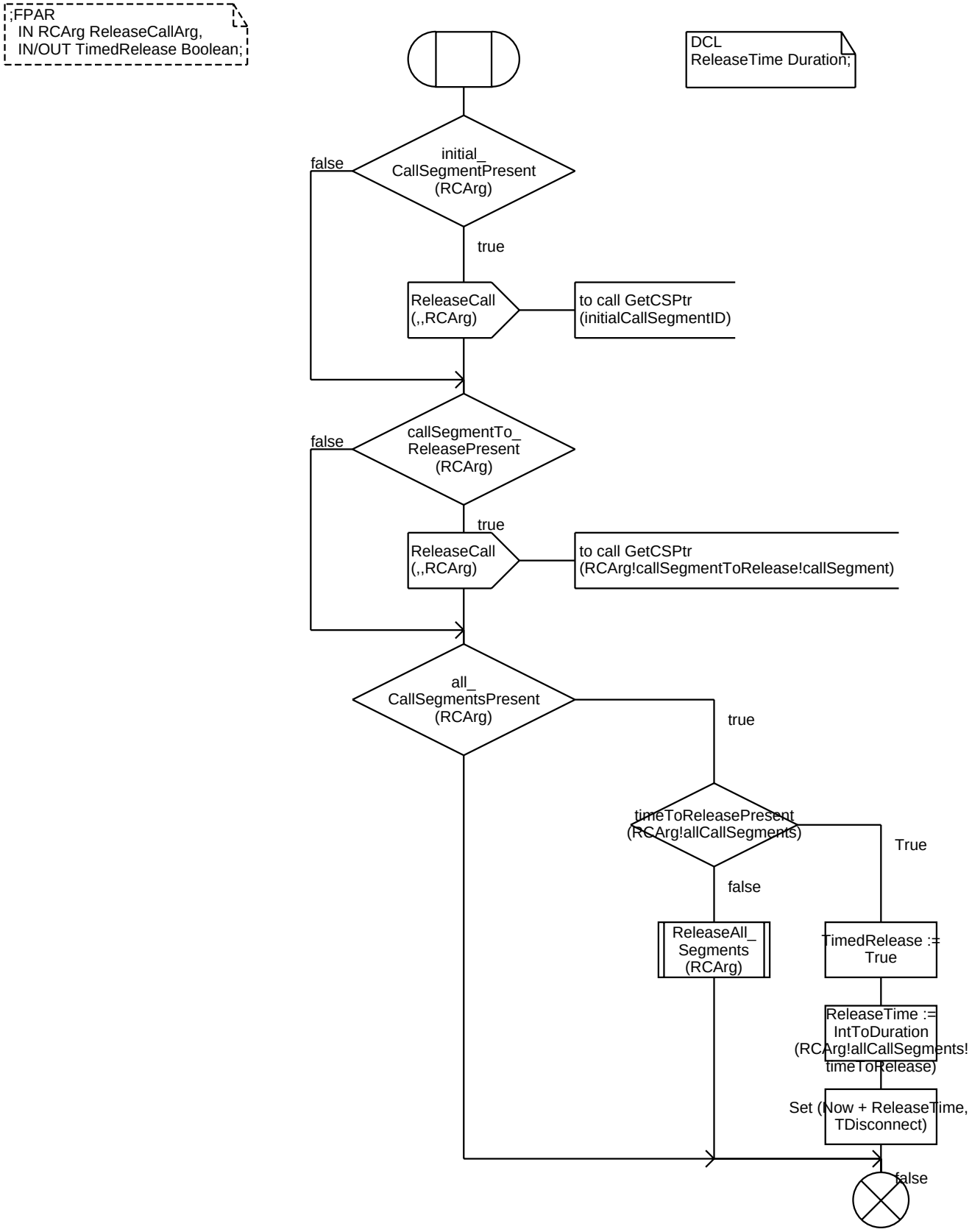


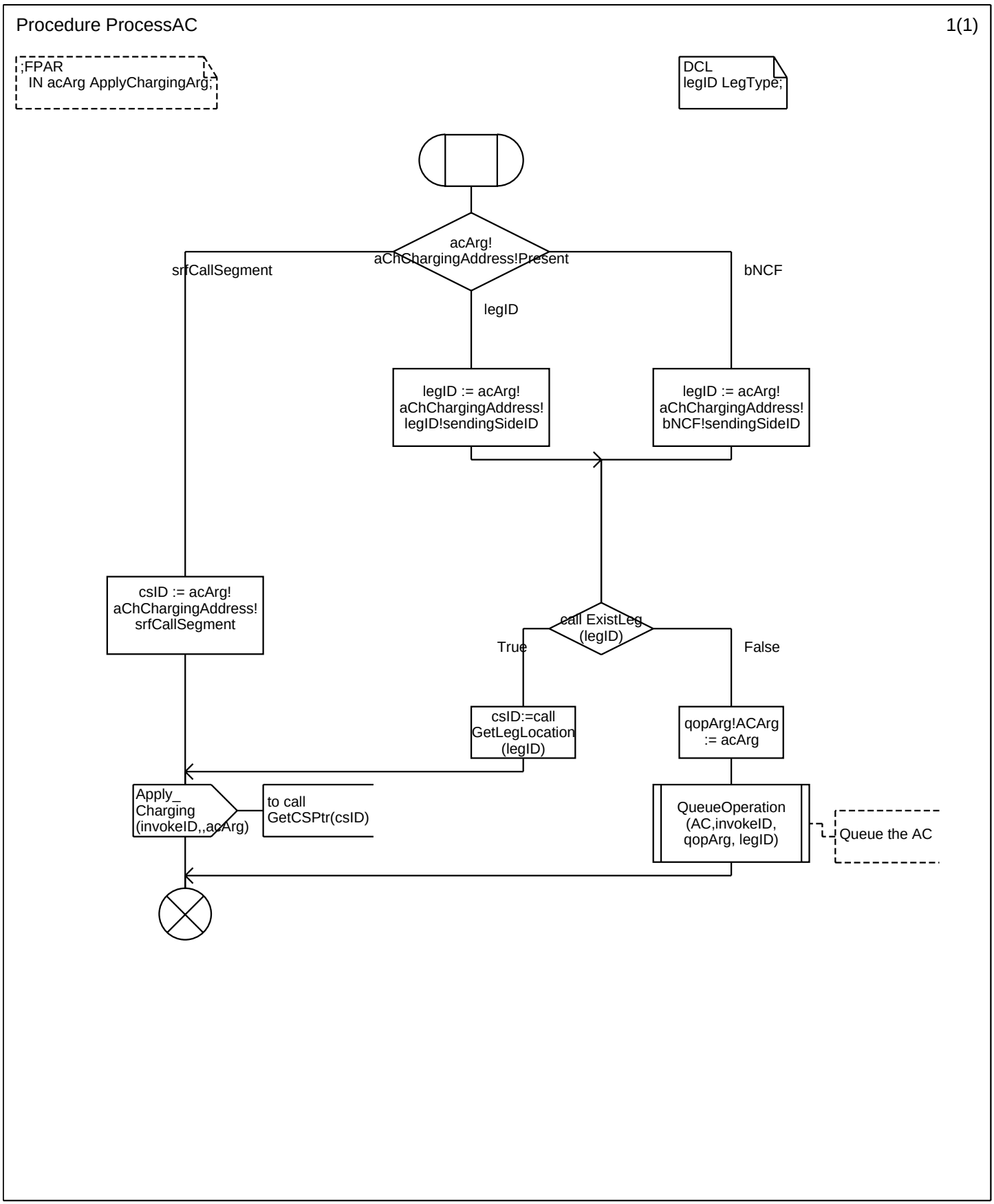
Procedure ProcessRRBE 1(1)



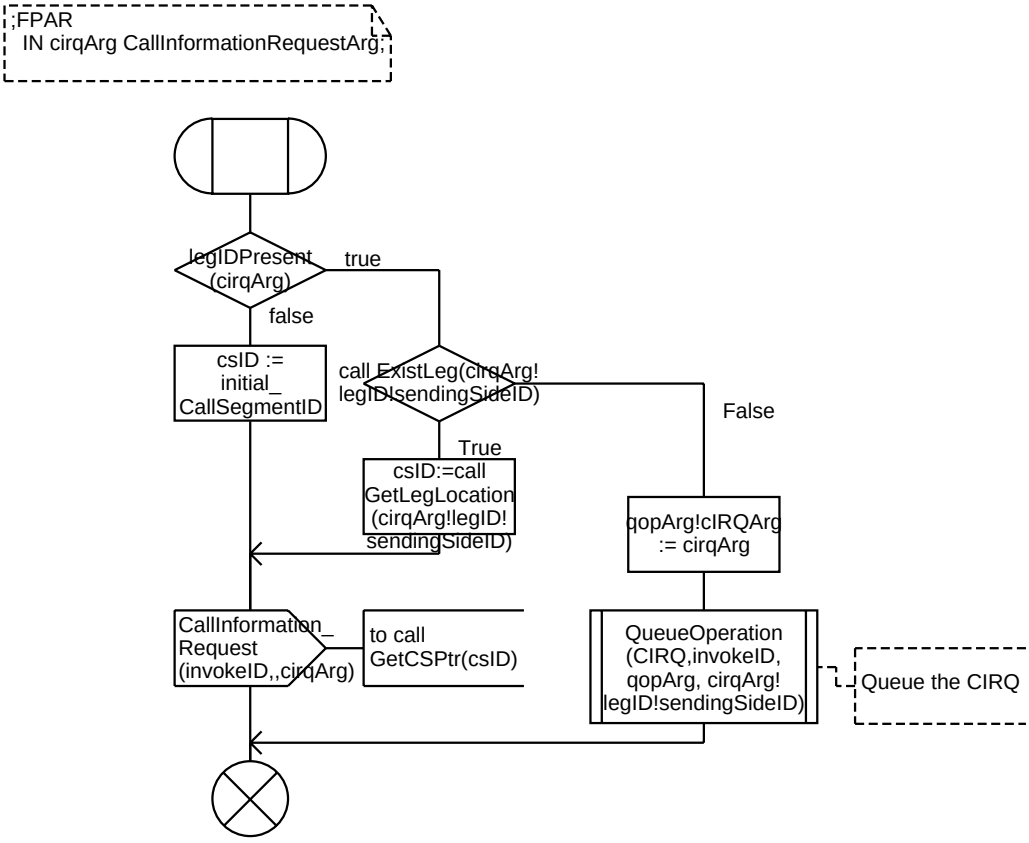

Procedure ProcessFCI
1(1)
DCL
callSegmentID CallSegmentID;
LegId DTLegID;
;FPAR
IN FCIArg FurnishChargingInformationArg;
FCIArg!
Present
else
fCIBCCsequence
callSegmentID :=
initialCallSegmentID
LegPresent
(FCIArg!fCIBCCsequence!
chargingAddress)
false
true
CallSegmentID :=
FCIArg!fCIBCCsequence!
chargingAddress!srfCallSegment
LegID := FCIArg!
fCIBCCsequence!
chargingAddress!leg
call ExistLeg
(LegID!sendingSideID)
True
False
CallSegmentID := call
GetLegLocation
(LegId!sendingSideID)
qopArg!FCIArg
:=FCIArg
FurnishCharging_
Information
(invokeID,csaID,FCIArg)
to call
GetCSPtr
(CallSegmentID)
QueueOperation
(FCI,invokeID, qopArg,
LegID!sendingSideId)
Queue the
FCI.

Procedure ReleaseCall 1(1)

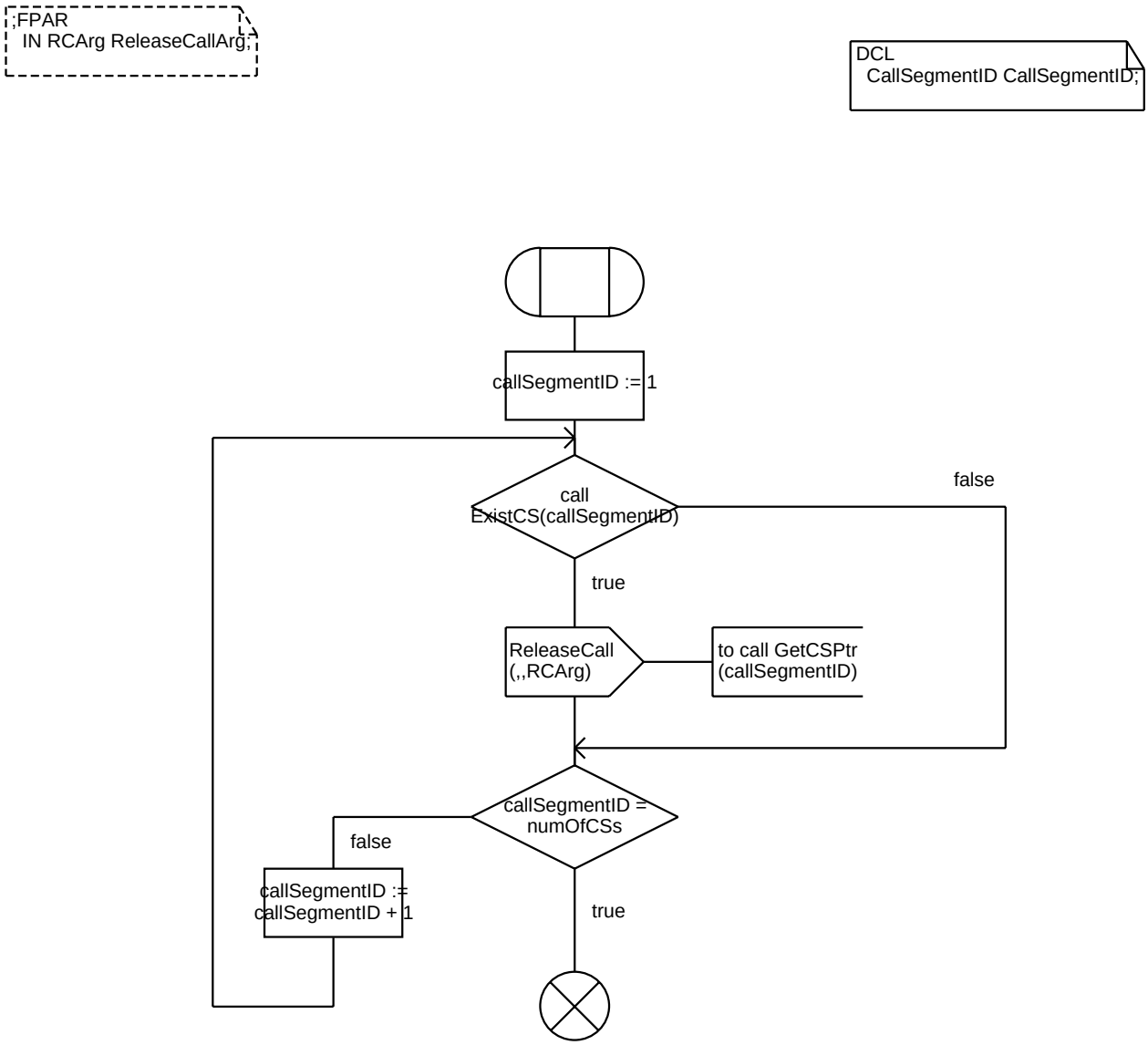



Procedure ProcessAC
1(1)
;FPAR
IN acArg ApplyChargingArg;
DCL
legID LegType;
acArg!
aChChargingAddress!Present
srfCallSegment
bNCF
legID
legID := acArg!
aChChargingAddress!
legID!sendingSideID
legID := acArg!
aChChargingAddress!
bNCF!sendingSideID
csID := acArg!
aChChargingAddress!
srfCallSegment
call ExistLeg
(legID)
True
False
csID:=call
GetLegLocation
(legID)
qopArg!ACArg
:= acArg
Apply_
Charging
(invokeID,,acArg)
to call
GetCSPtr(csID)
QueueOperation
(AC,invokeID,
qopArg, legID)
Queue the AC

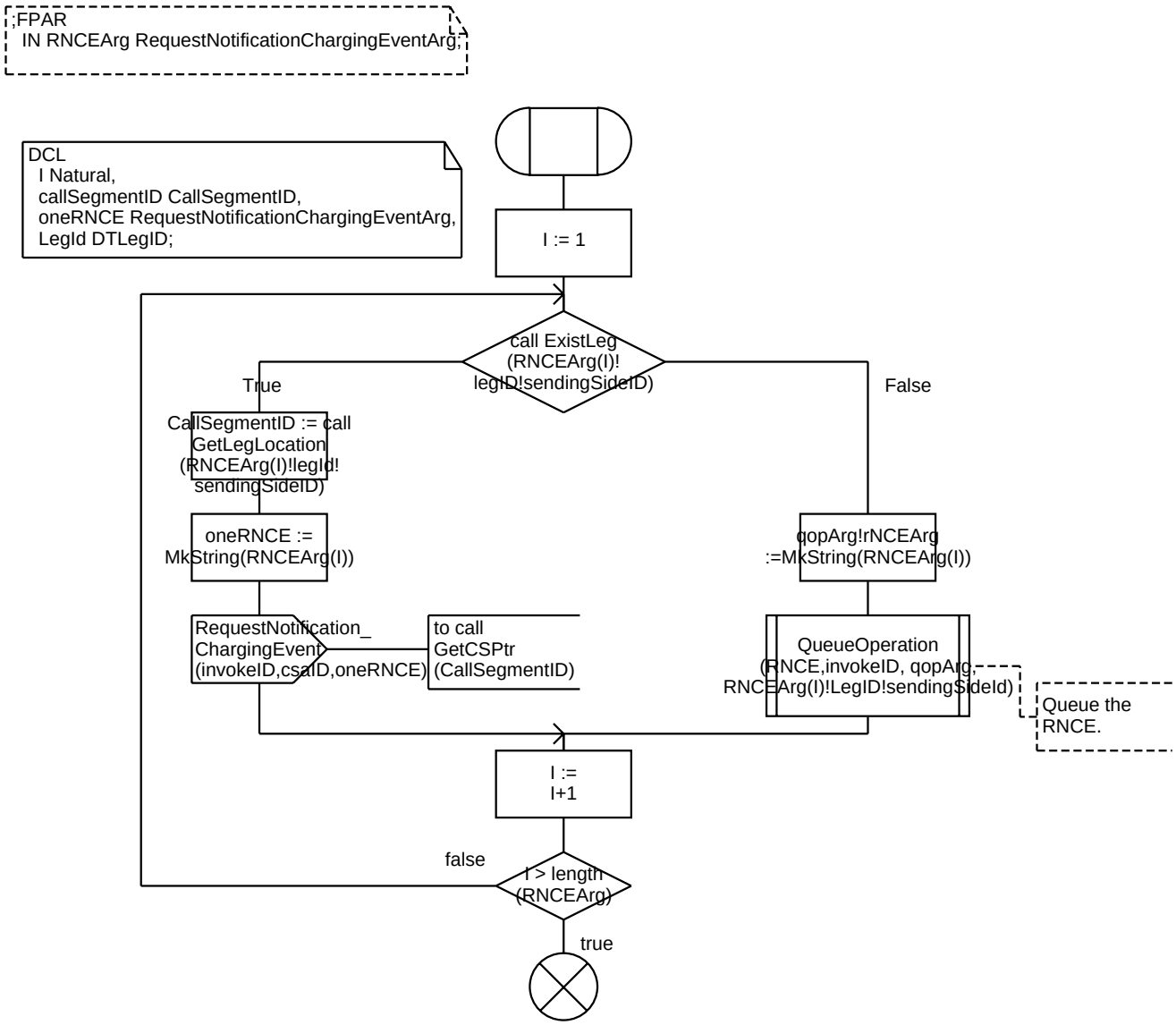
Procedure ProcessCIRQ 1(1)



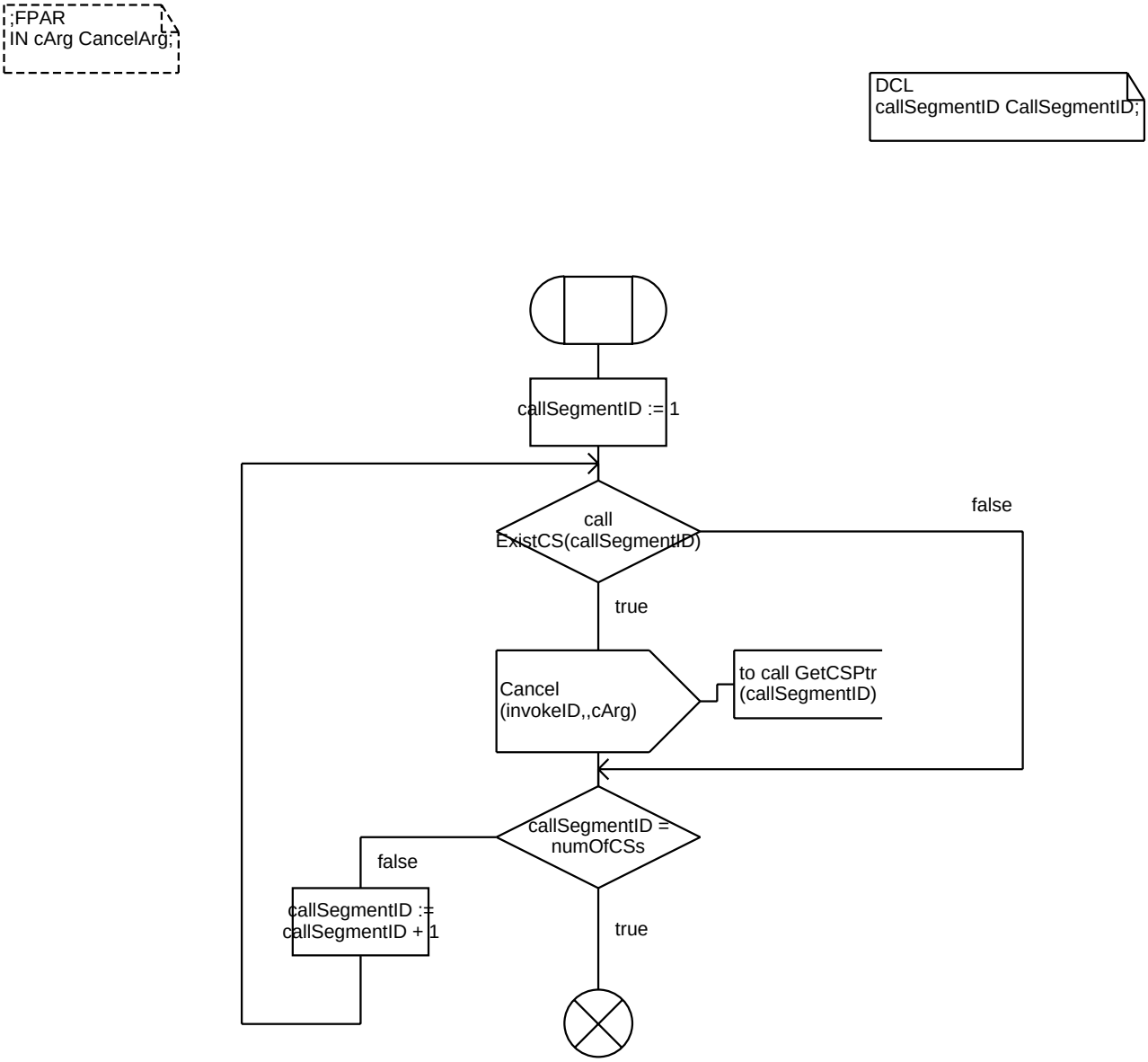
Procedure ReleaseAllSegments 1(1)



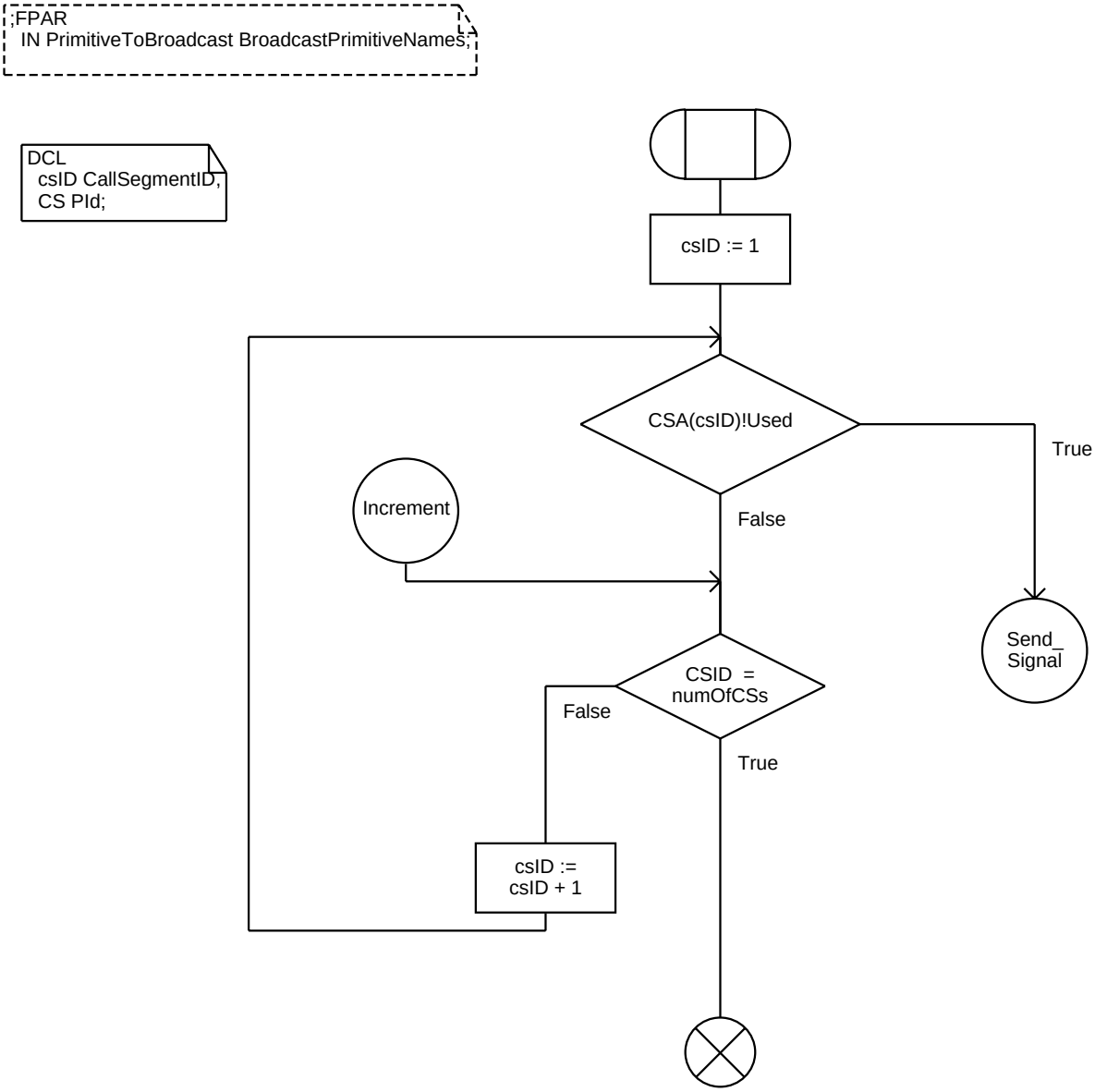
Procedure ProcessRNCE 1(1)



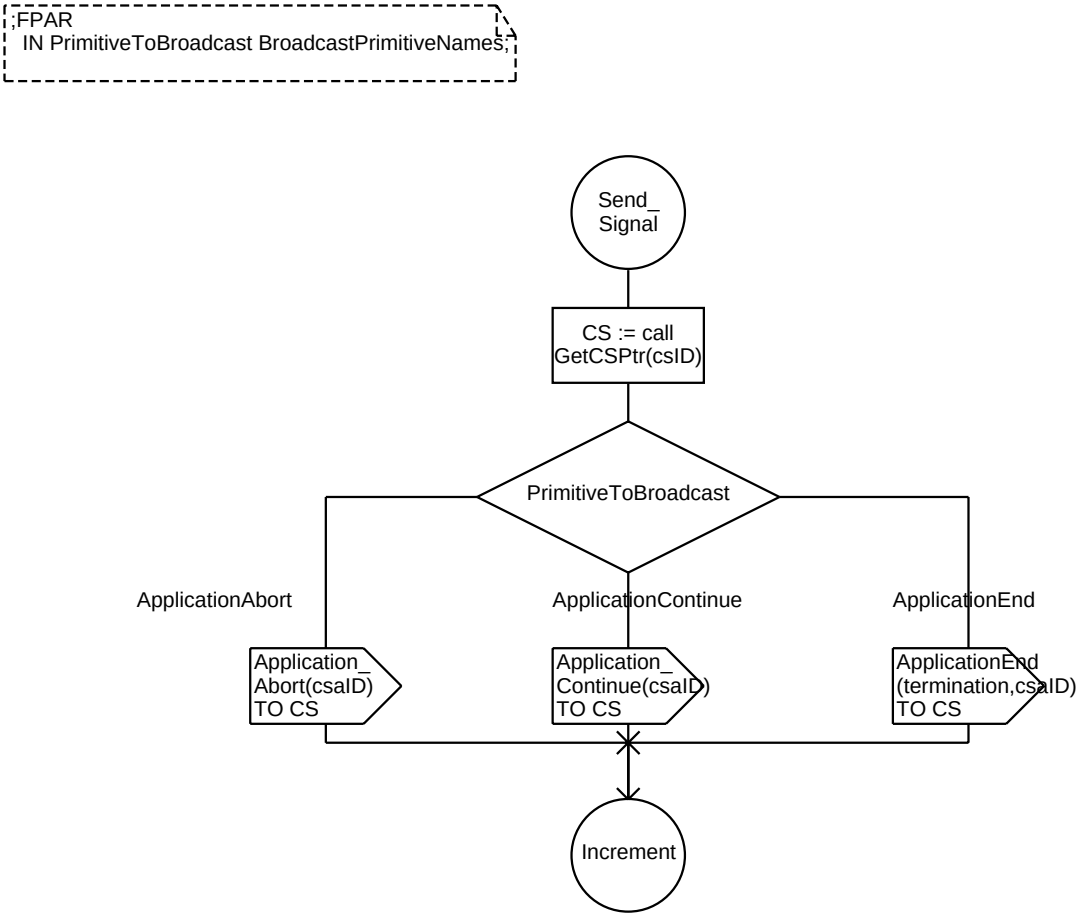
Procedure <<Process Type CallSegmentAssociation>> ProcessCancel 1(1)



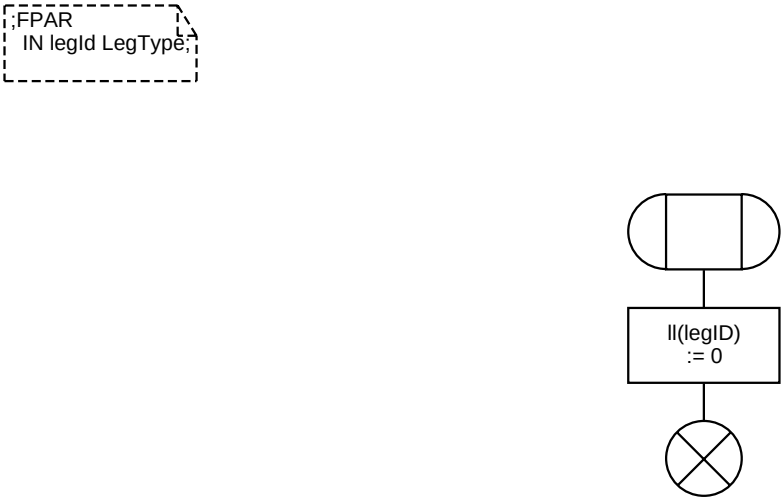
Procedure BroadcastToCSs 1(2)

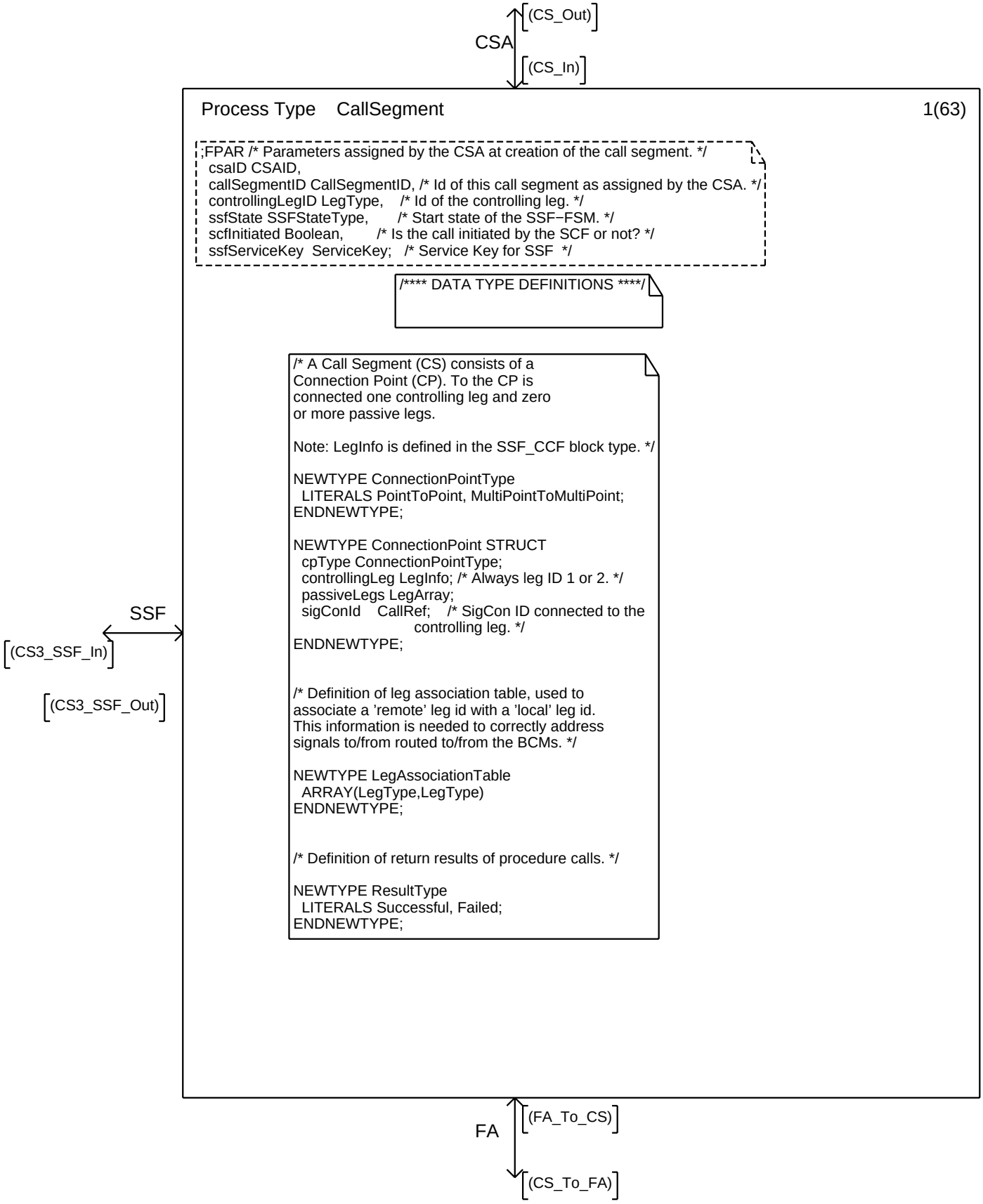


Procedure BroadcastToCSs 2(2)



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




CSA
(CS_Out)
(CS_In)
Process Type CallSegment
1(63)
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
csaID CSAID,
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? */
ssfServiceKey ServiceKey; /* Service Key for SSF */
/**** 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;
SSF
(CS3_SSF_In)
(CS3_SSF_Out)
FA
(FA_To_CS)
(CS_To_FA)

Process Type CallSegment 2(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

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

```
/* (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;
```

```
/*   DECLARATION OF IMPORTED REMOTE PROCEDURES  */

/* CreateFIMAgent is exported by FIM_Controller, and called by BCP and CS.
 * It requests the creation of a new FIMAgent + a chain of BCSMs.
 * Returns the PId of the created FIMAgent.
 */
IMPORTED PROCEDURE CreateFIMAgent;
 FPAR BCSMType, BCSMPars, CSAID, PId, LegType;
 RETURNS PId;

/* RemoveLegCSA is exported by the CSA, and permits CS to update CSA whenever
it removes a leg, and that removal is not visible to CSA */
IMPORTED PROCEDURE RemoveLegCSA;
 FPAR LegType;

/* ExportEventRecord is exported by SSF, and permits CS to update SSF
whenever it removes a leg and that removal is not visible to SSF */
IMPORTED PROCEDURE ExportEventRecord;
 FPAR IN LegType, IN/OUT EventRecordType, IN/OUT Integer;
```

Process Type CallSegment 3(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

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

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

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

  /* Pointer to the FIMAgent associated
     with this segment. */
  FA 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 */
  invokeID InvokeID,
  termination Boolean,
  bcsmPars BCSMPars,
  bcsmStopped Boolean := false,
  legId LegType,
  newLegID LegType,
  pic PICArg,
  dp DPArg,
  eventTable EventTableType,
  r Boolean,
  bcsm BCSMType,
  typeOfBCSM BCSMType,
  oldcs PID,
  newcs PID,
 requests Integer,
  Exception Boolean;
```

```
DCL
  /* IN 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,
  mrArg MessageReceivedArg,
  paArg PlayAnnouncementArg,
  pacuiArg PromptAndCollectUserInformationArg,
  parmArg PromptAndReceiveMessageArg,
  riArg ReceivedInformationArg,
  rcArg ReleaseCallArg,
  rnceARg RequestNotificationChargingEventArg,
  rrBCSMEArg RequestReportBCSMEventArg,
  rtARg ResetTimerArg,
  scArg ScriptCloseArg,
  seArg ScriptEventArg,
  siArg ScriptInformationArg,
  scrArg ScriptRunArg,
  sfArg SelectFacilityArg,
  sciArg SendChargingInformationArg,
  srrArg SpecializedResourceReportArg,

  ieArg ErrorArg;
```

Process Type CallSegment 4(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

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

```
DCL
  /* IN operation arguments. */
  ctwaArg ContinueWithArgumentArg,
  dfcwaArg DisconnectForwardConnectionWithArgumentArg,
  dlArg DisconnectLegArg,
  erArg EntityReleasedArg,
  rutsiArg ReportUTSIArg,
  rrutsiArg RequestReportUTSIArg,
  sspArg SetServiceProfileArg,
  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,
  lastEDPFlag Boolean,
  /* Used to indicate the status of the call processing. */
  setupSent boolean := false;    /* Indicates if SetupReq is sent to the called party */
```

Process Type CallSegment 5(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF. */
```

/**** 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.

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.

CheckNo_PendingLegs

NoOfJoined_Legs

AddBCSM — Requests FIM and BCP to add a BCSM to CP, for leg added to CS

Exported Delete_PendingLeg

Get_Passive_LegId — Returns the id of the passive leg in the connection point.

GetPending_LegID

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

MapConnect_ToBCSM

MapSI_ToBCSM

exported SetNoAnswerTime

Process Type CallSegment 6(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF. */
```

/*** 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.

Exist_ Leg

Export_ Leg — Exports a leg from the connection point.

NoOf_ Legs — Returns the number of passive legs.

EXPORTED GetFAPtr

EXPORTED ValidSource_ CSforML

EXPORTED ValidSource_ CSforMCS

EXPORTED ValidSource_ CSforSL

EXPORTED ValidTarget_ CSforMLMCS

/* Procedures for filtering detection points. */

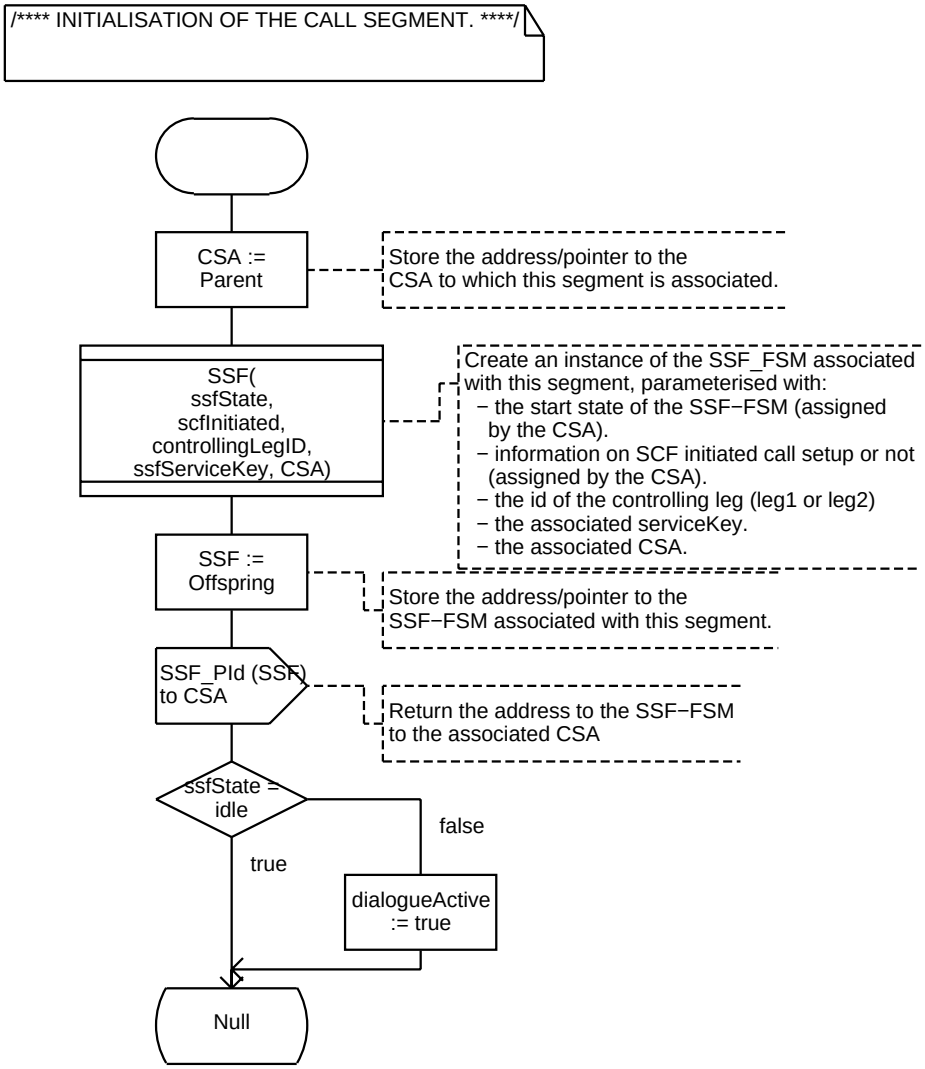
DP_ Filtering

MapCI_ ToBCSM

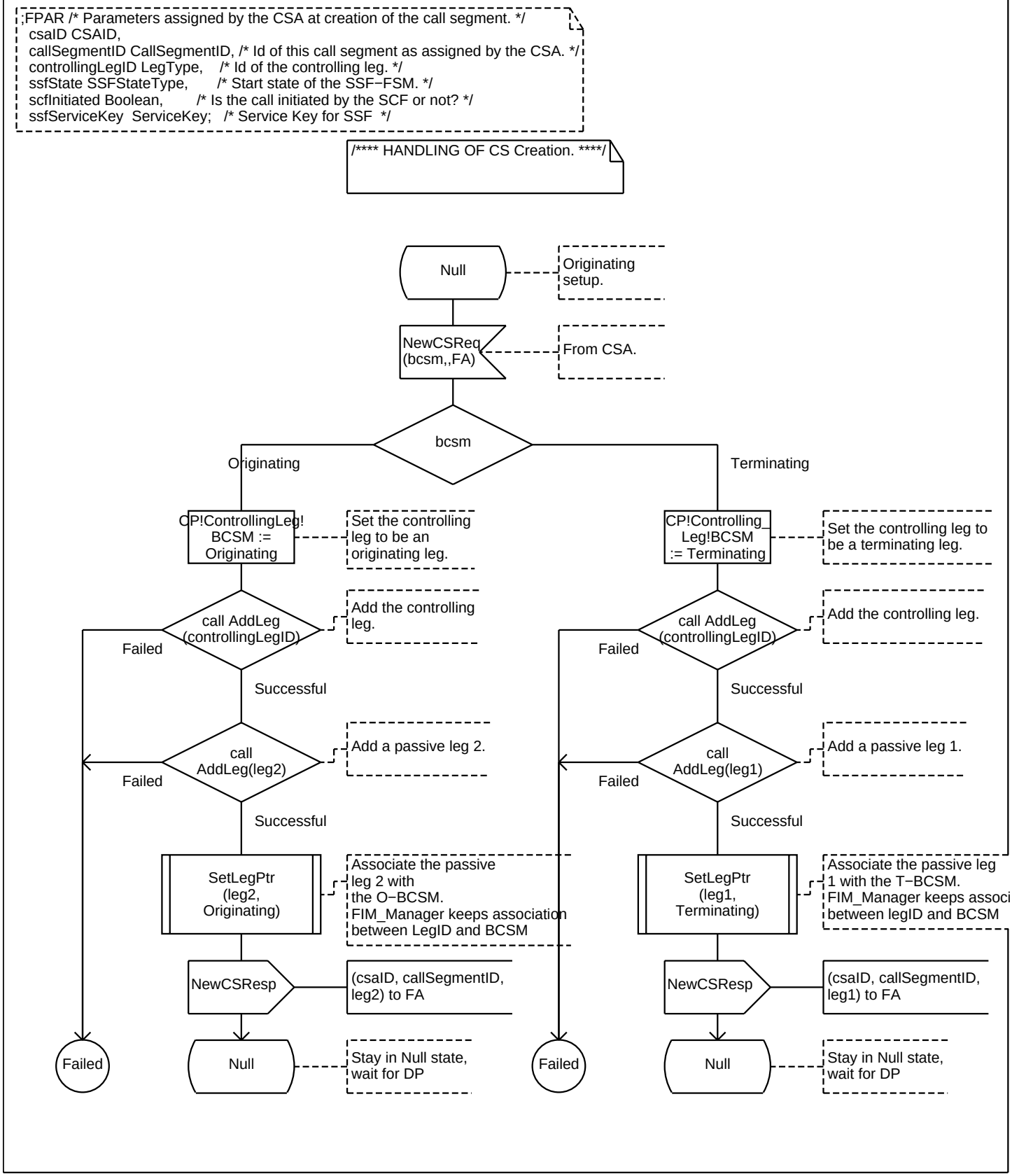
DPFiltering_ UTSI

Process Type CallSegment 7(63)

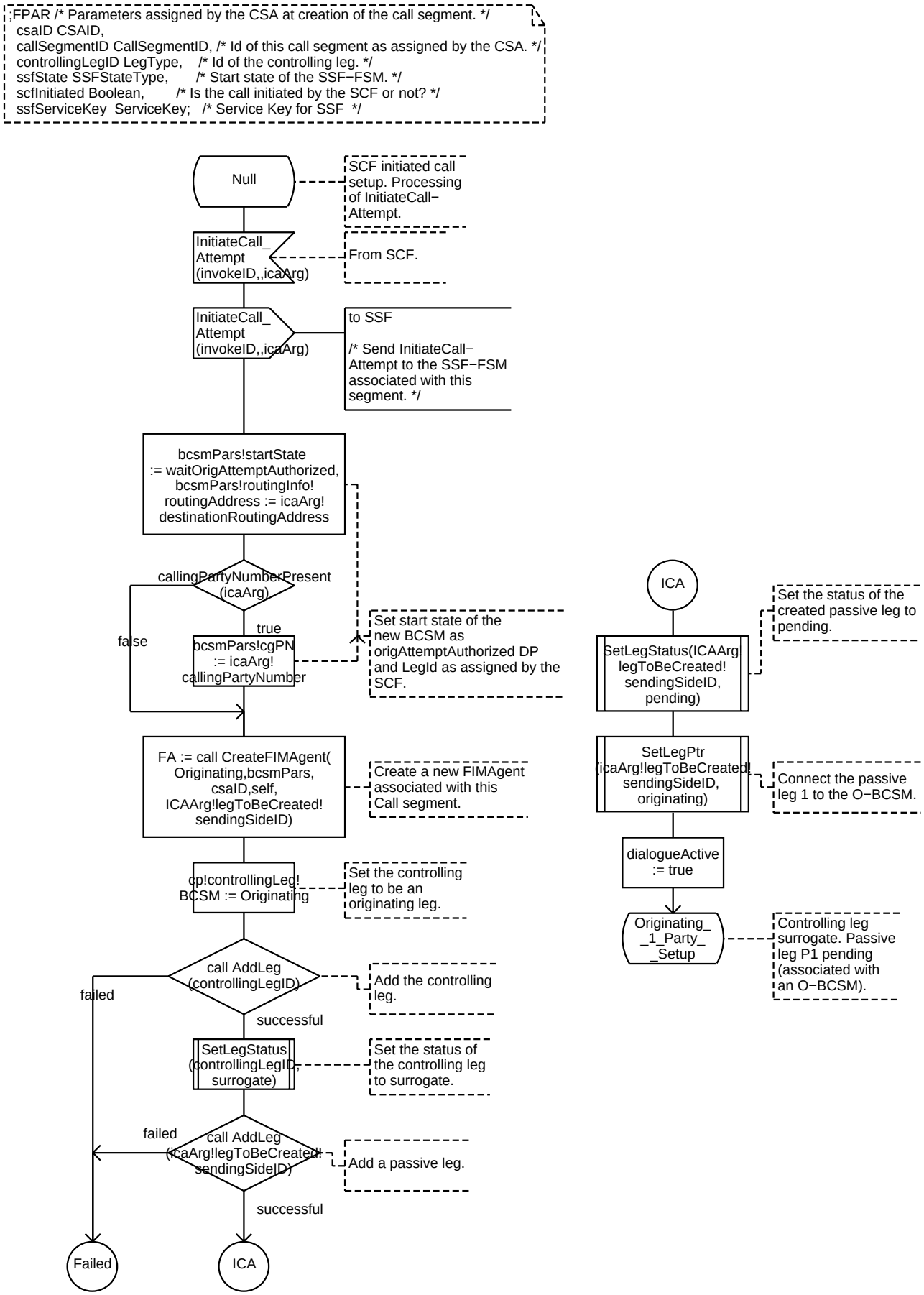
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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



Process Type CallSegment 8(63)



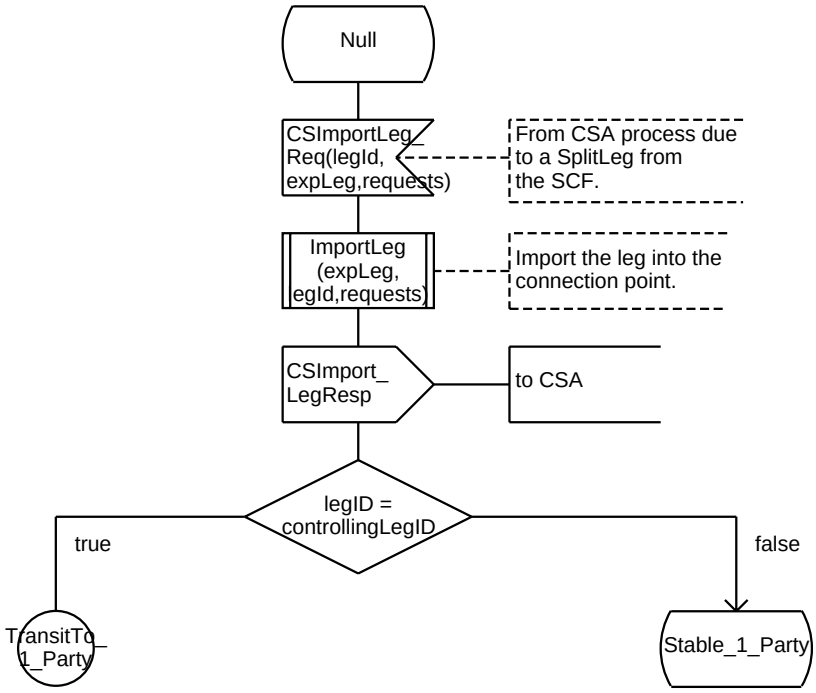
Process Type CallSegment 9(63)



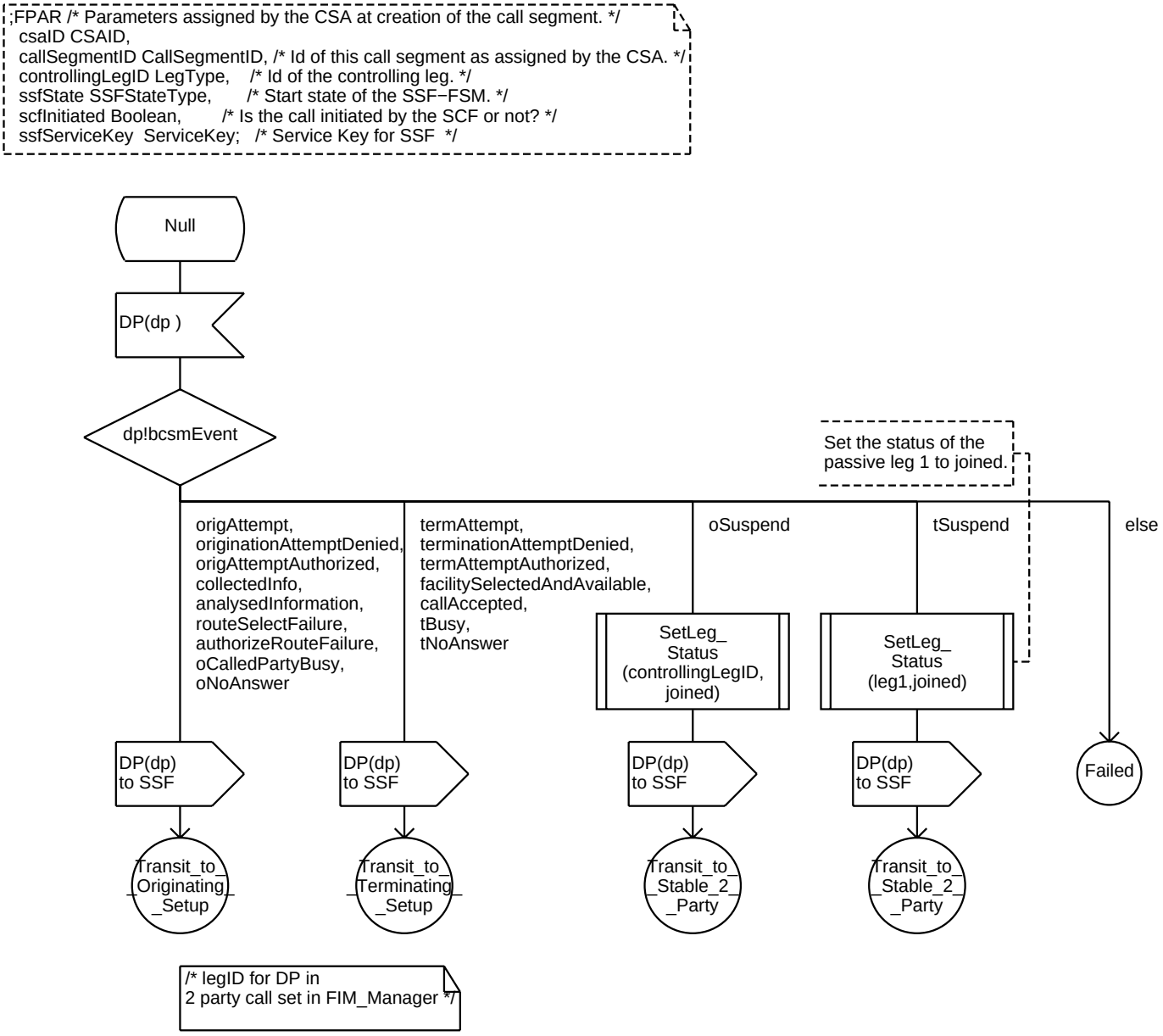
Process Type CallSegment 10(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

```
/* Null −> 1−Party
   Null −> On−Hold

As a result of a SplitLeg operation from SCF.
Note: Processing of atomic operations done by the CSA. */
```

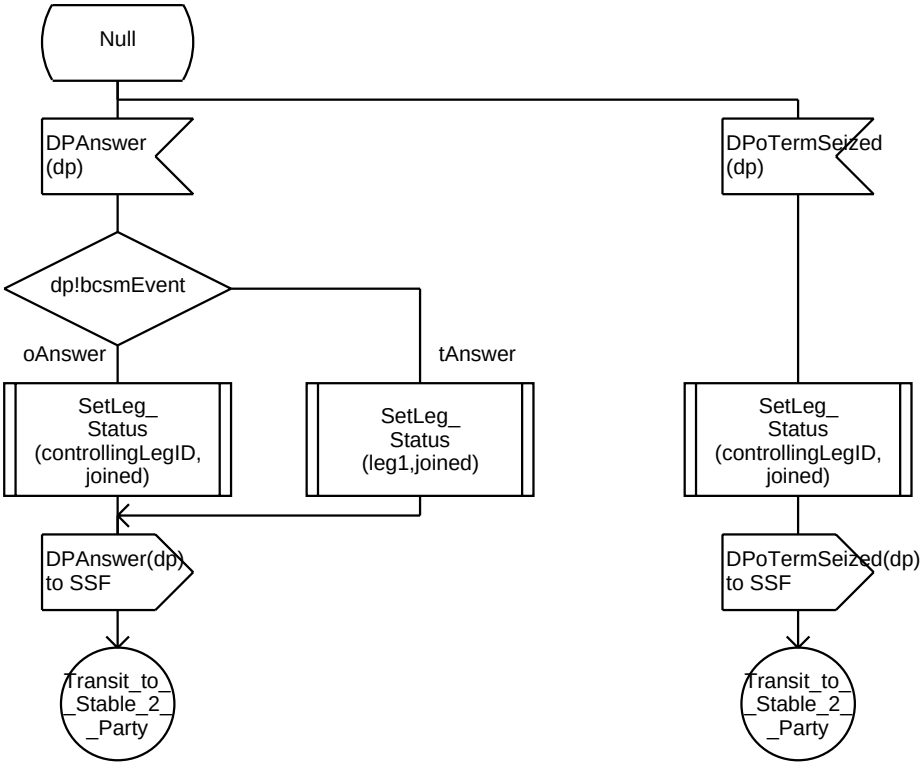


Process Type CallSegment 11(63)



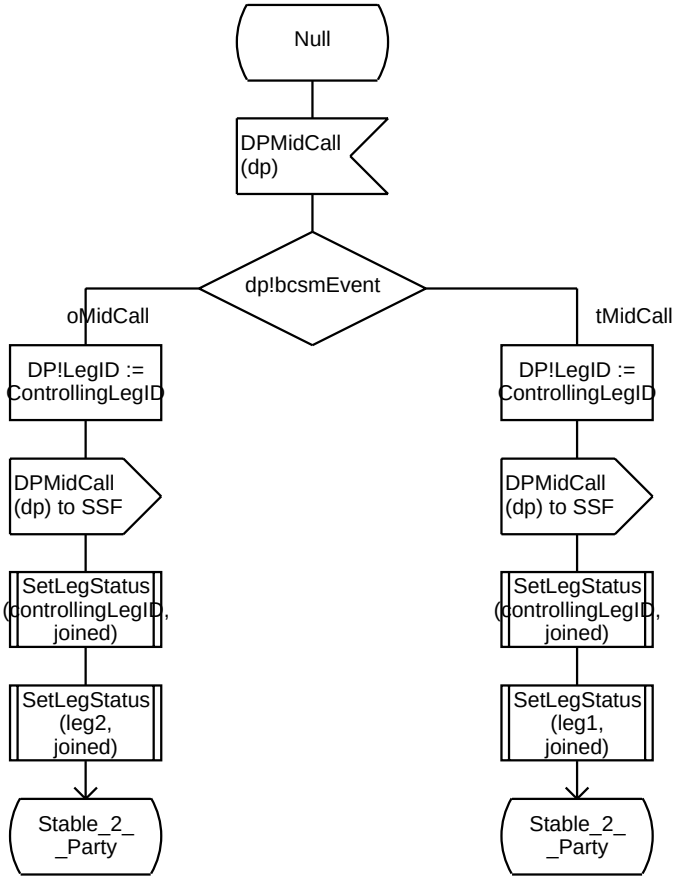
Process Type CallSegment 12(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



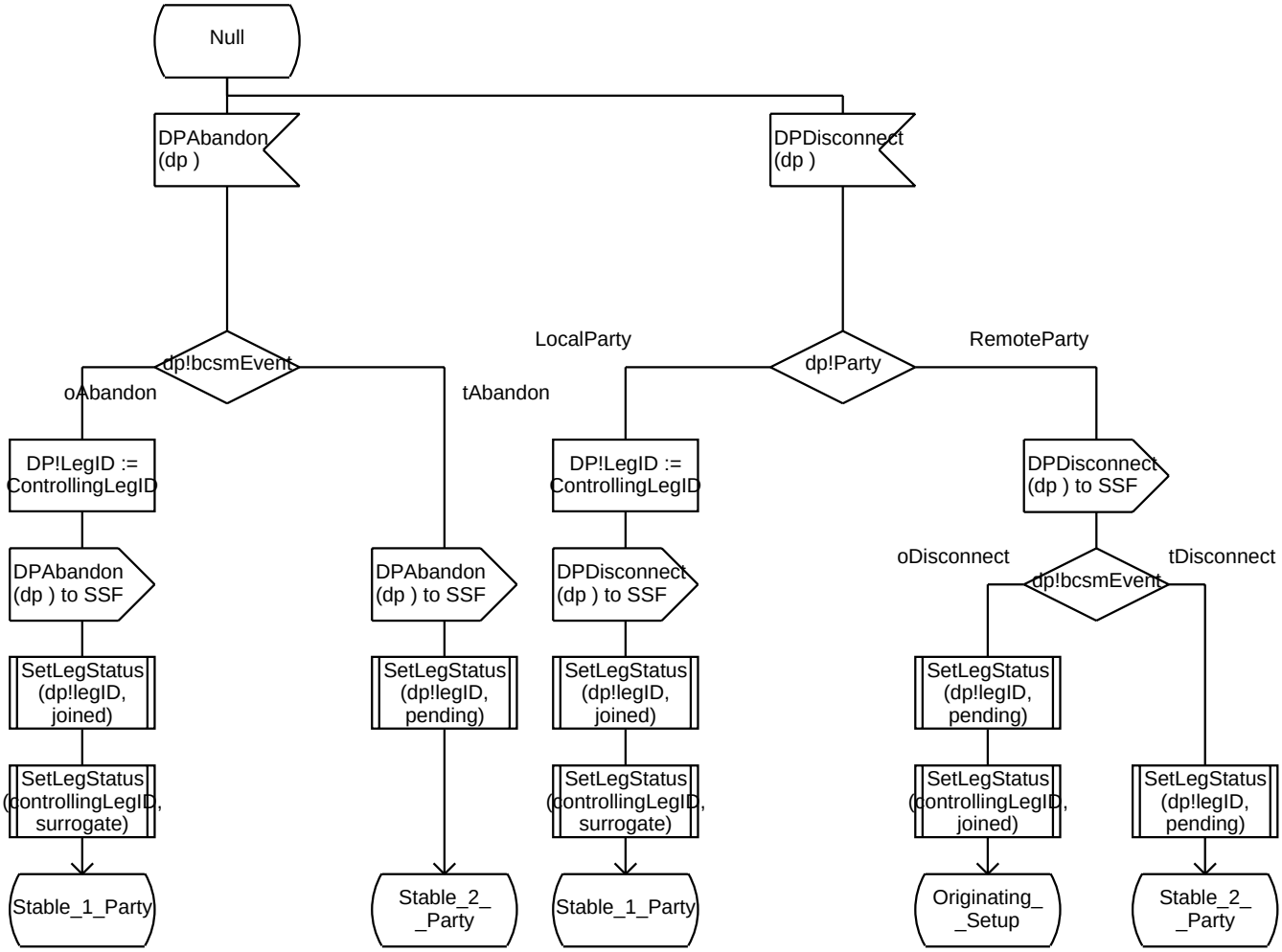
Process Type CallSegment 13(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



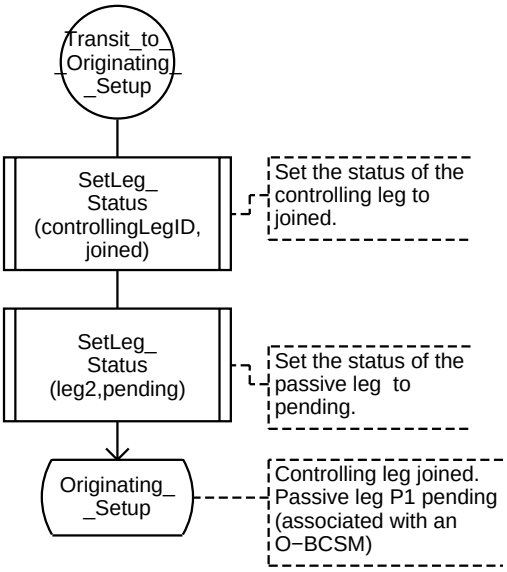
Process Type CallSegment 14(63)

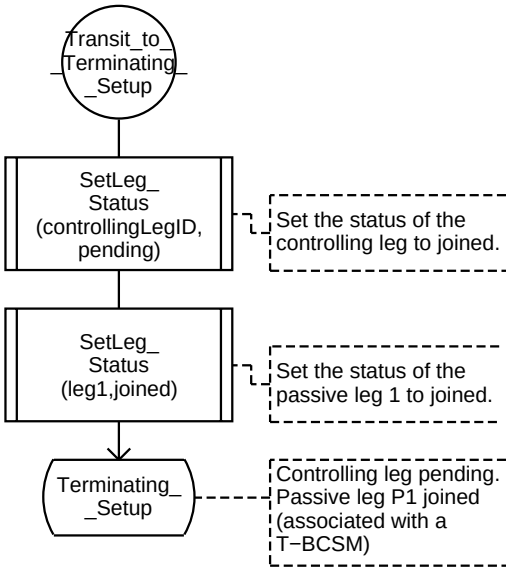
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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

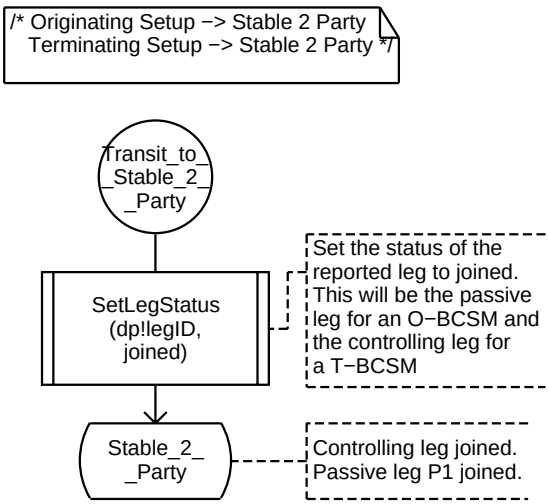


Process Type CallSegment 15(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



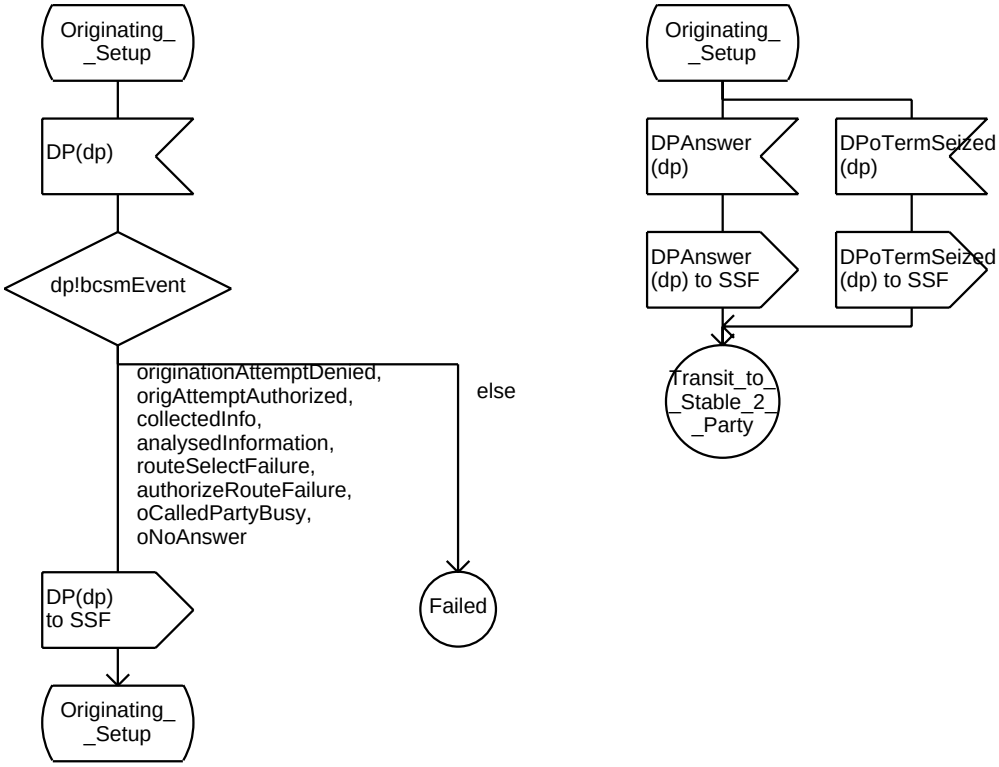




Process Type CallSegment 16(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

```
/*  DP Processing in state Originating_Setup */
```



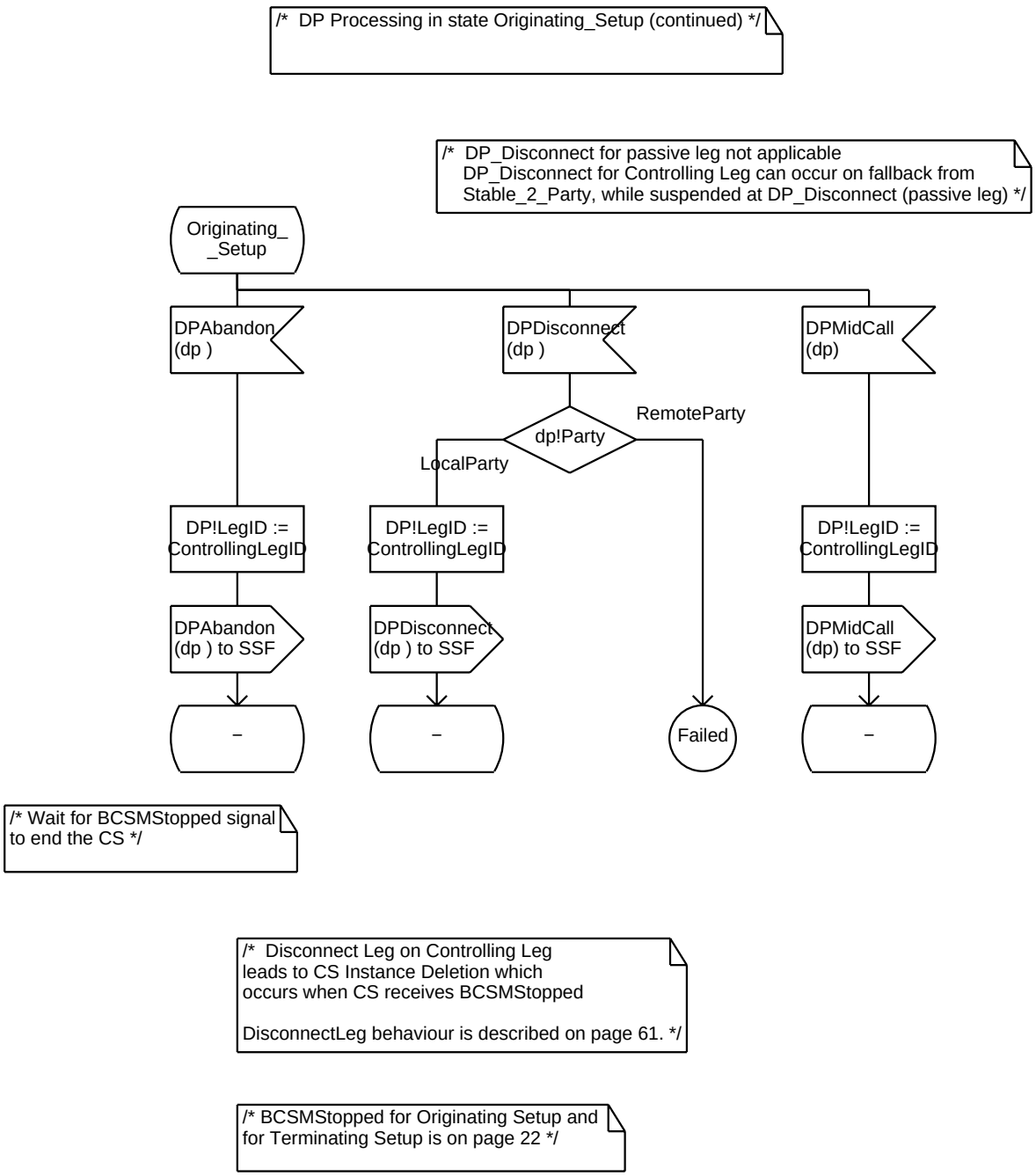

```
/* legID for DPs in
2 party call set in FIM_Manager */
```

```
/* Note:
Events occuring when the passive leg is pending
(in the case of originating setup), or when the
controlling leg is pending (in case of terminating
setup) should be reported for the controlling leg.

Note: This does not apply when the controlling leg
is surrogate, in Originating_1_Party_Setup, Stable_1_Party,
Forward or Stable_Multi_Passive_Party. */
```

```
/*  <−−−Is this always true, now that
we've changed the point at which the
pending leg becomes joined? */
```

Process Type CallSegment 17(63)

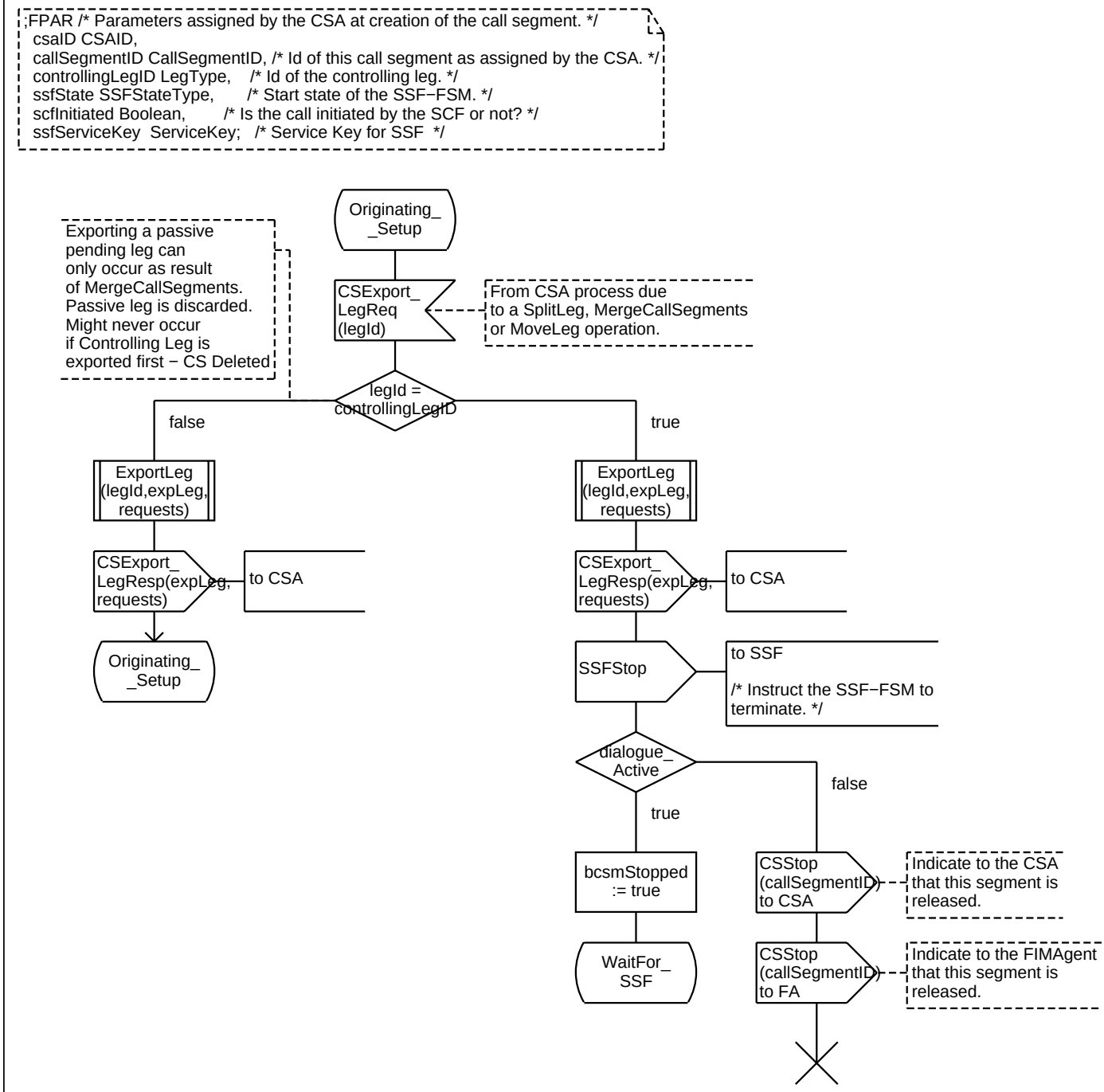
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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




```
/*  Disconnect Leg on Controlling Leg
leads to CS Instance Deletion which
occurs when CS receives BCSMStopped

DisconnectLeg behaviour is described on page 61. */
```

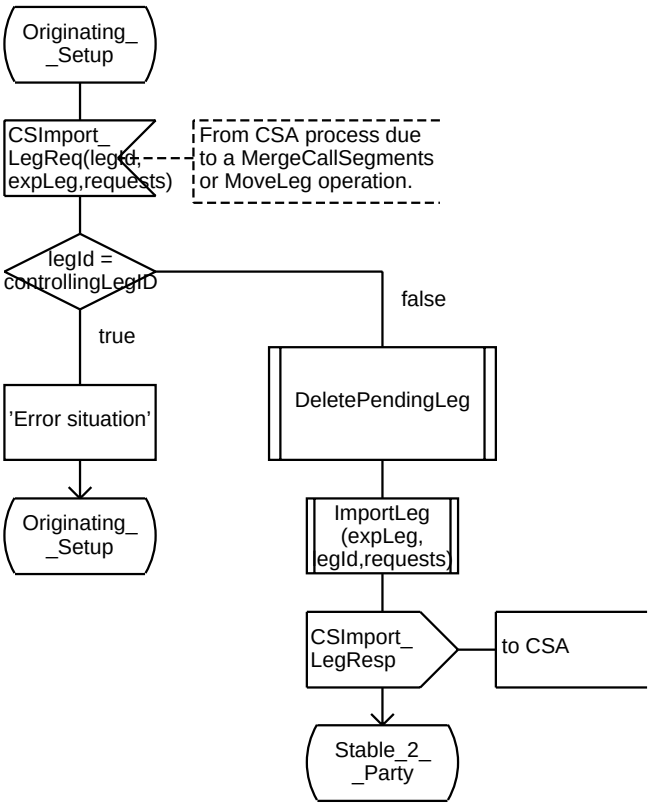
```
/* BCSMStopped for Originating Setup and
for Terminating Setup is on page 22 */
```

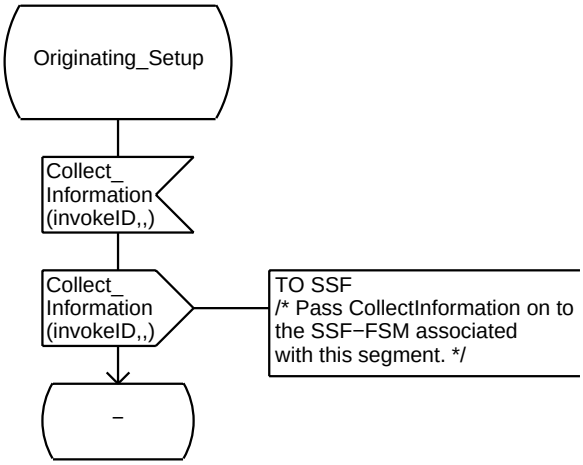
Process Type CallSegment 18(63)



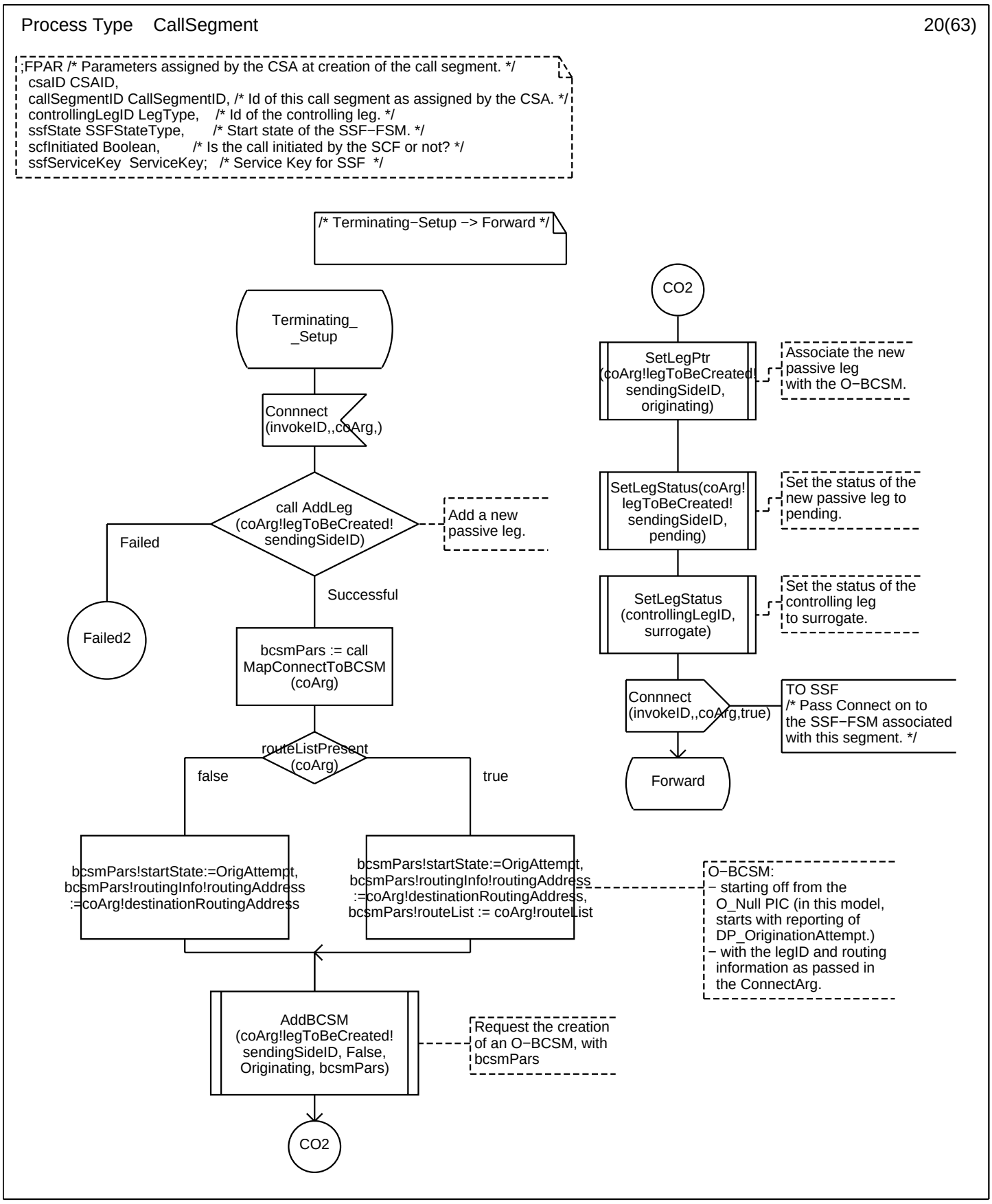
Process Type CallSegment 19(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



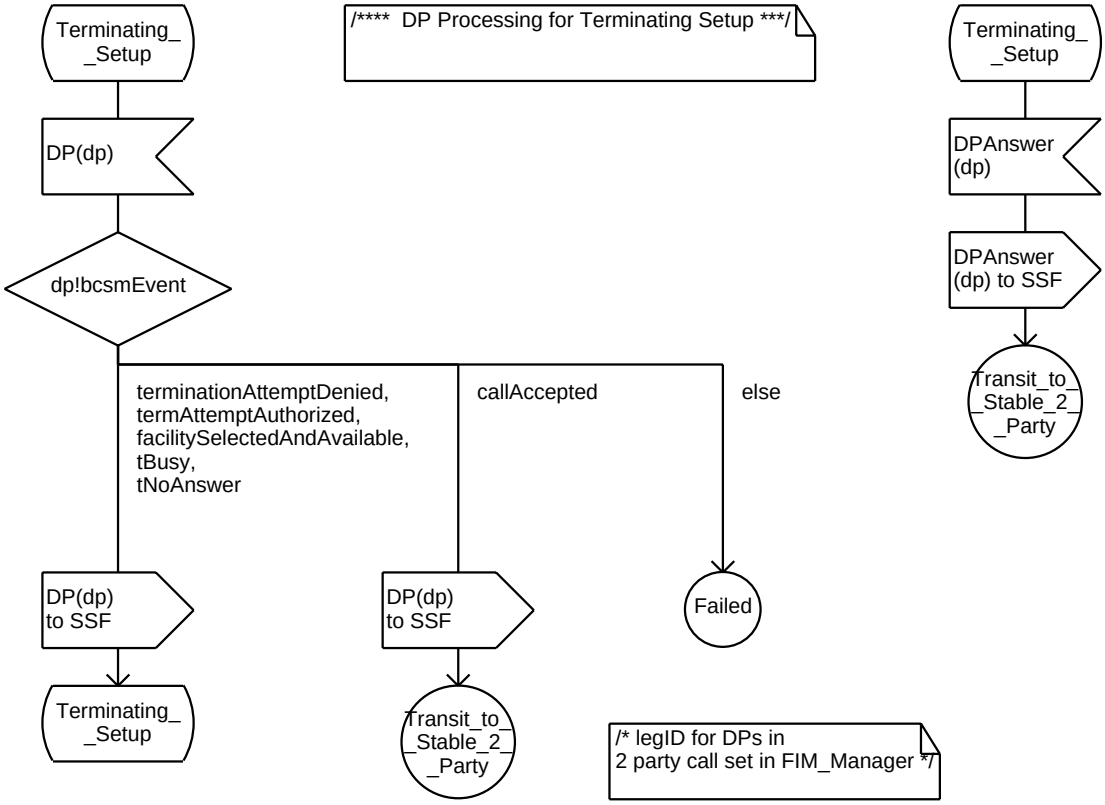


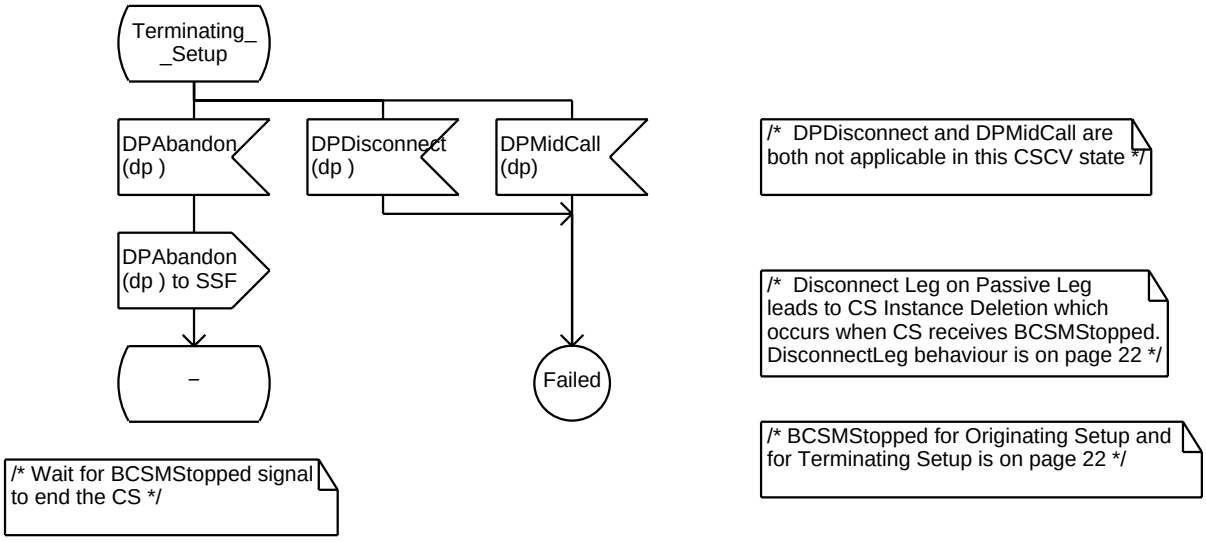
Process Type CallSegment 20(63)



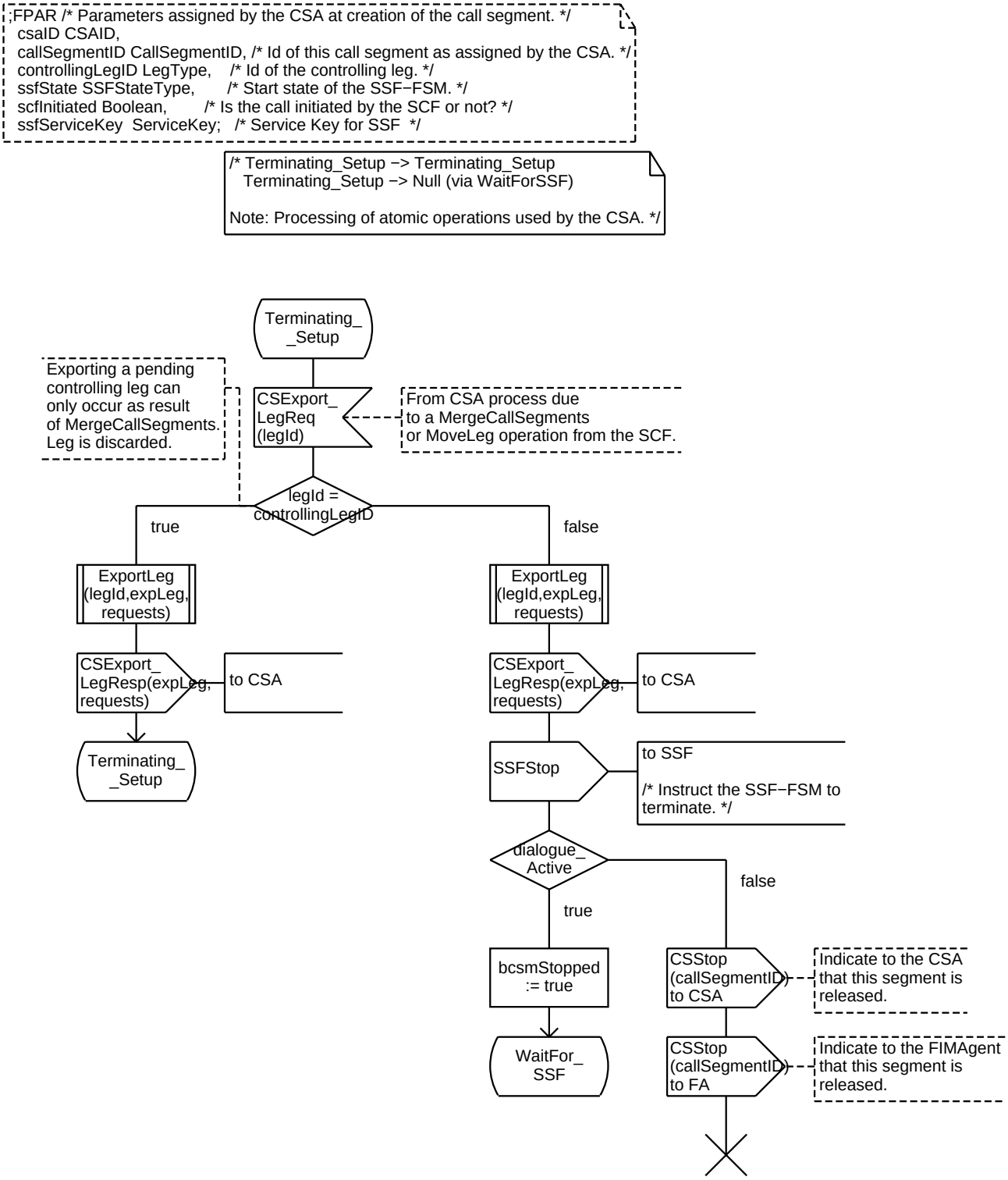
Process Type CallSegment 21(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

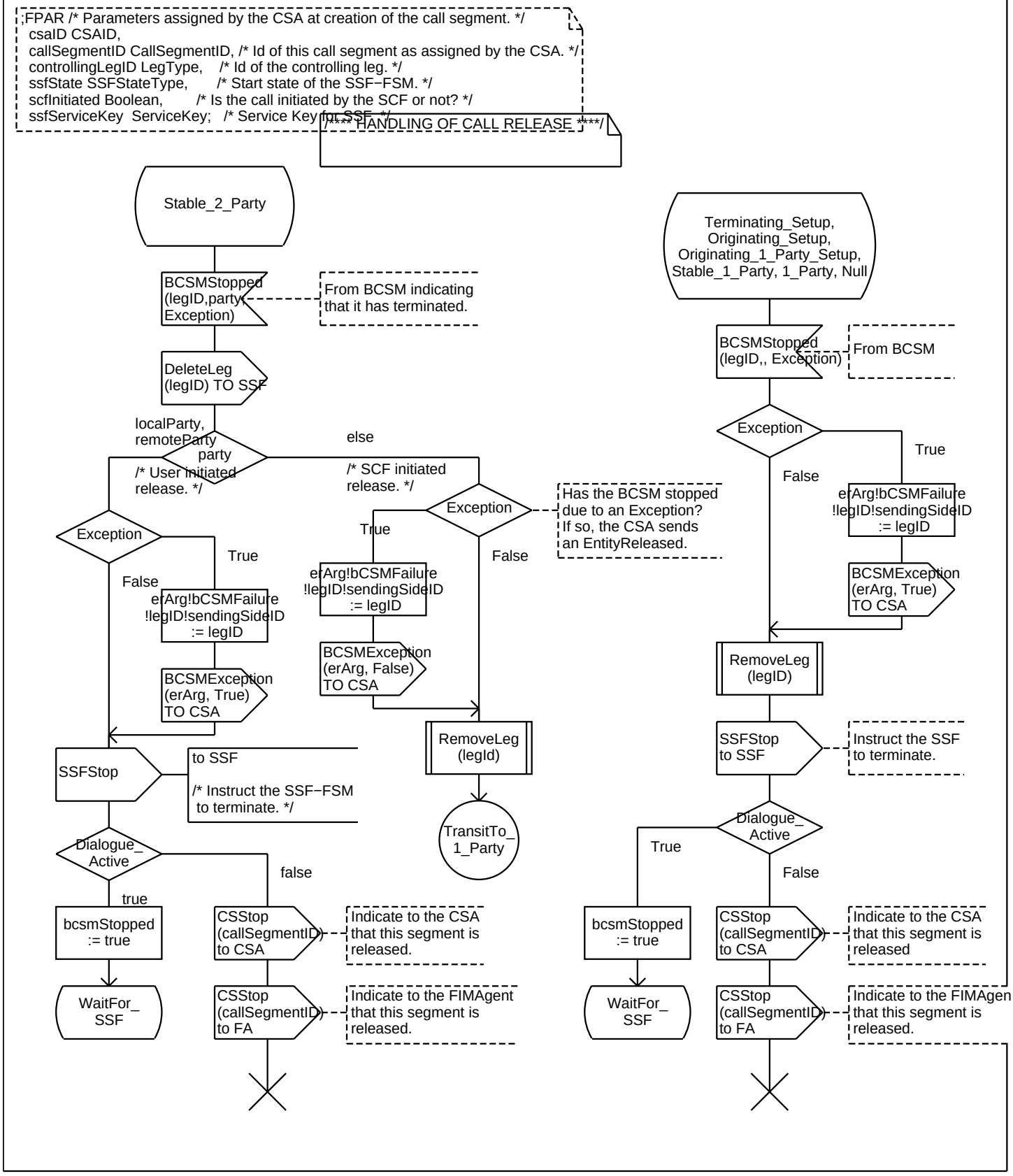




Process Type CallSegment 22(63)

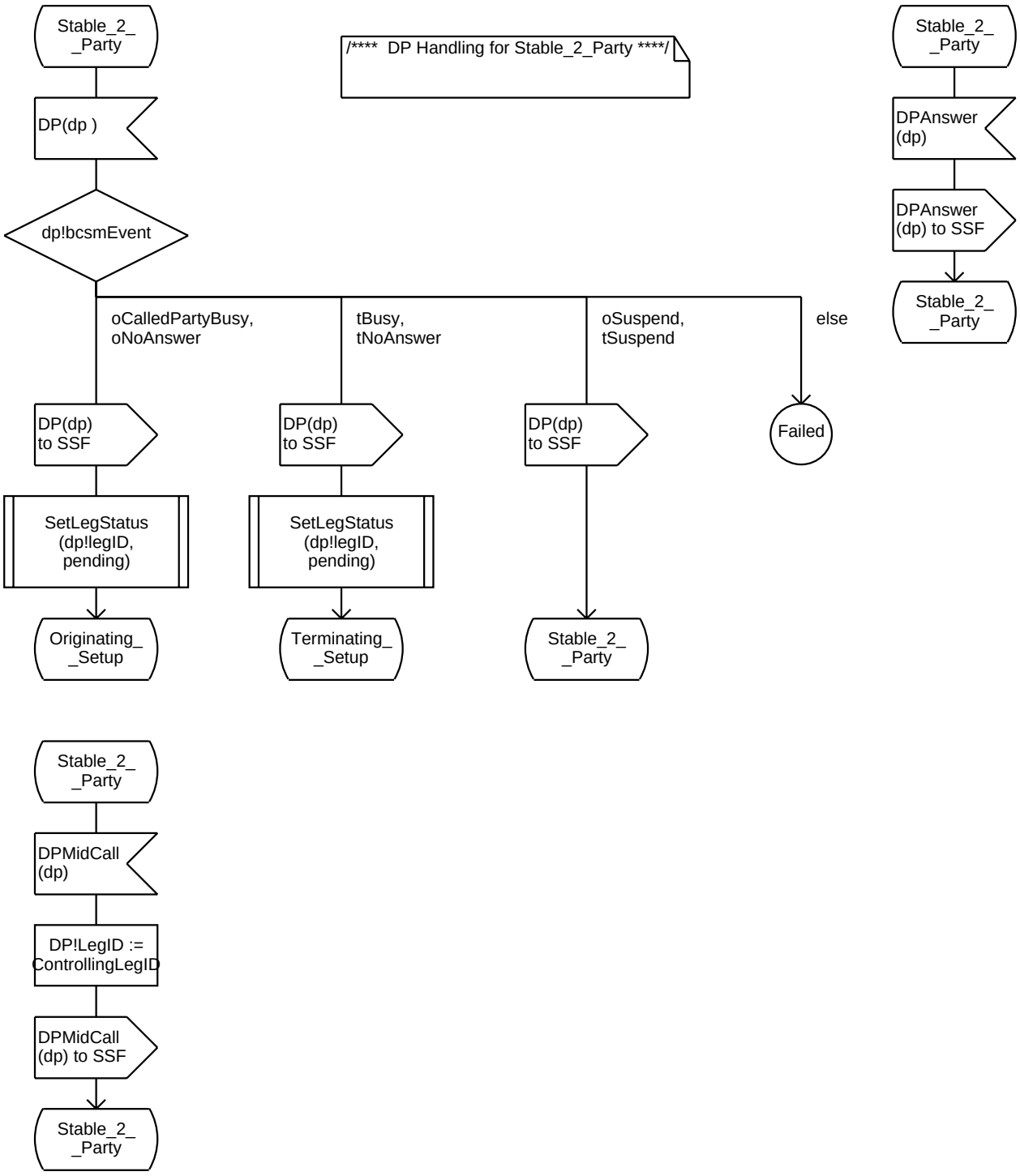


Process Type CallSegment 23(63)



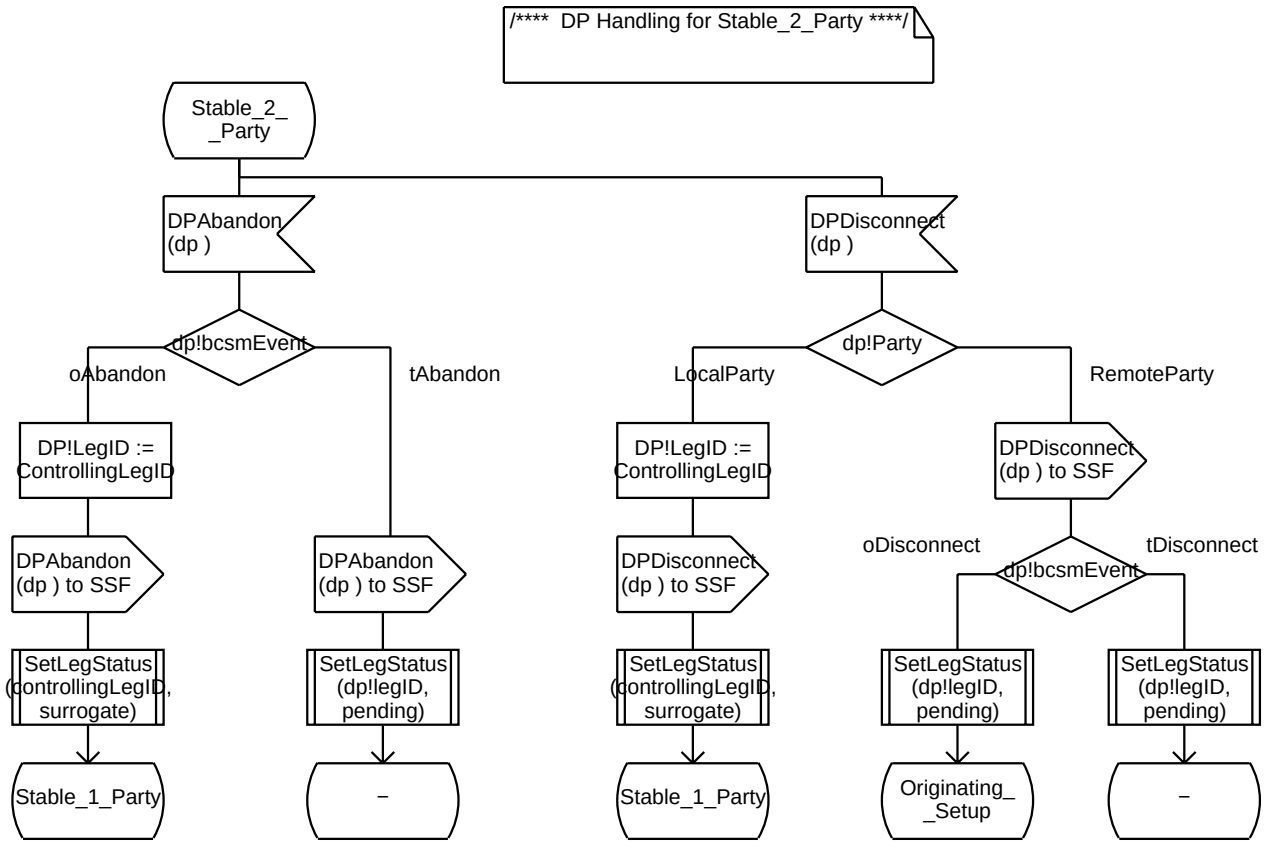
Process Type CallSegment 24(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



Process Type CallSegment 25(63)

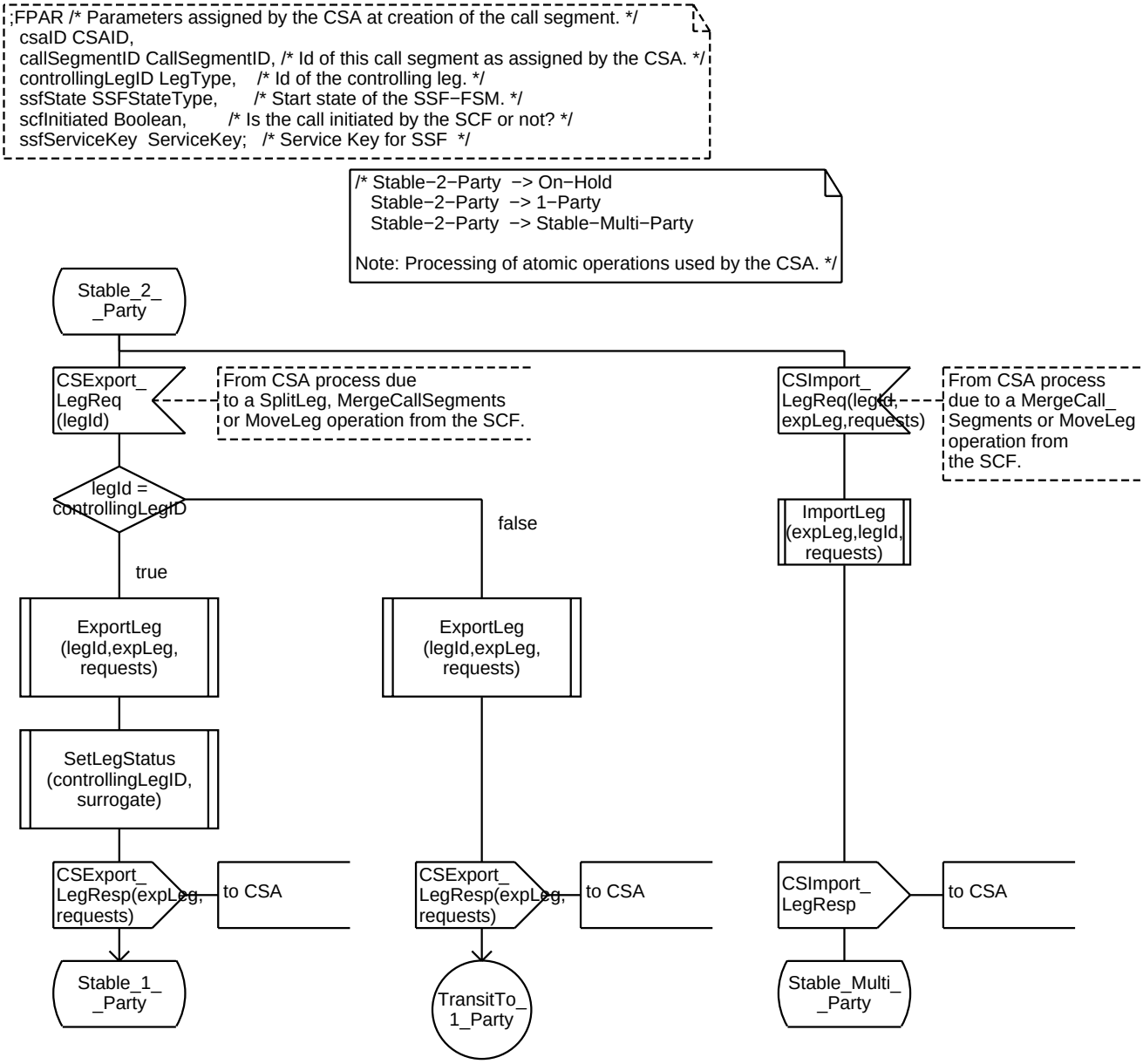
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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



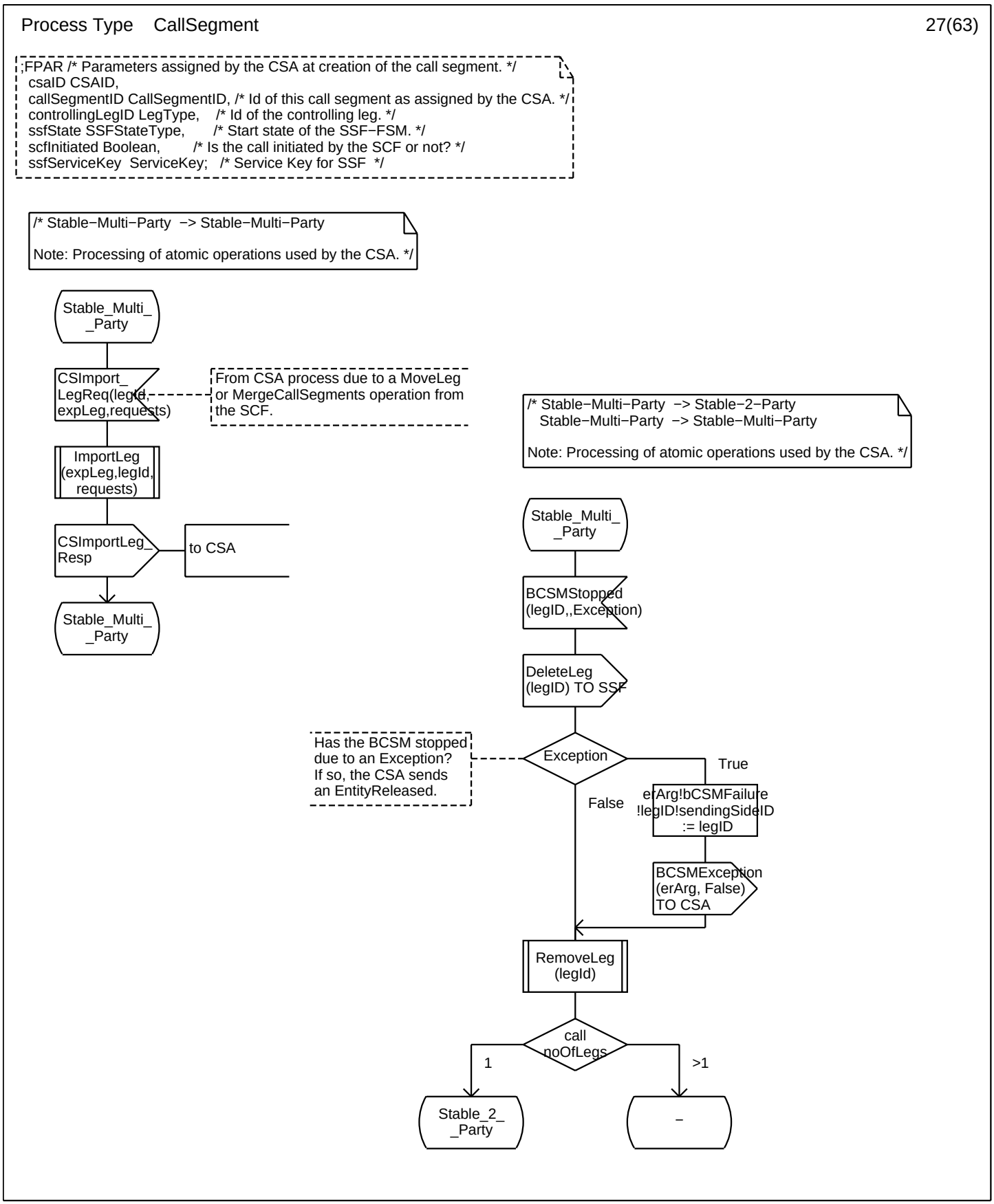
/* DisconnectLeg behaviour is on Page 61 */

/* BCSMStopped for Stable_2_Party
is on page 22 */

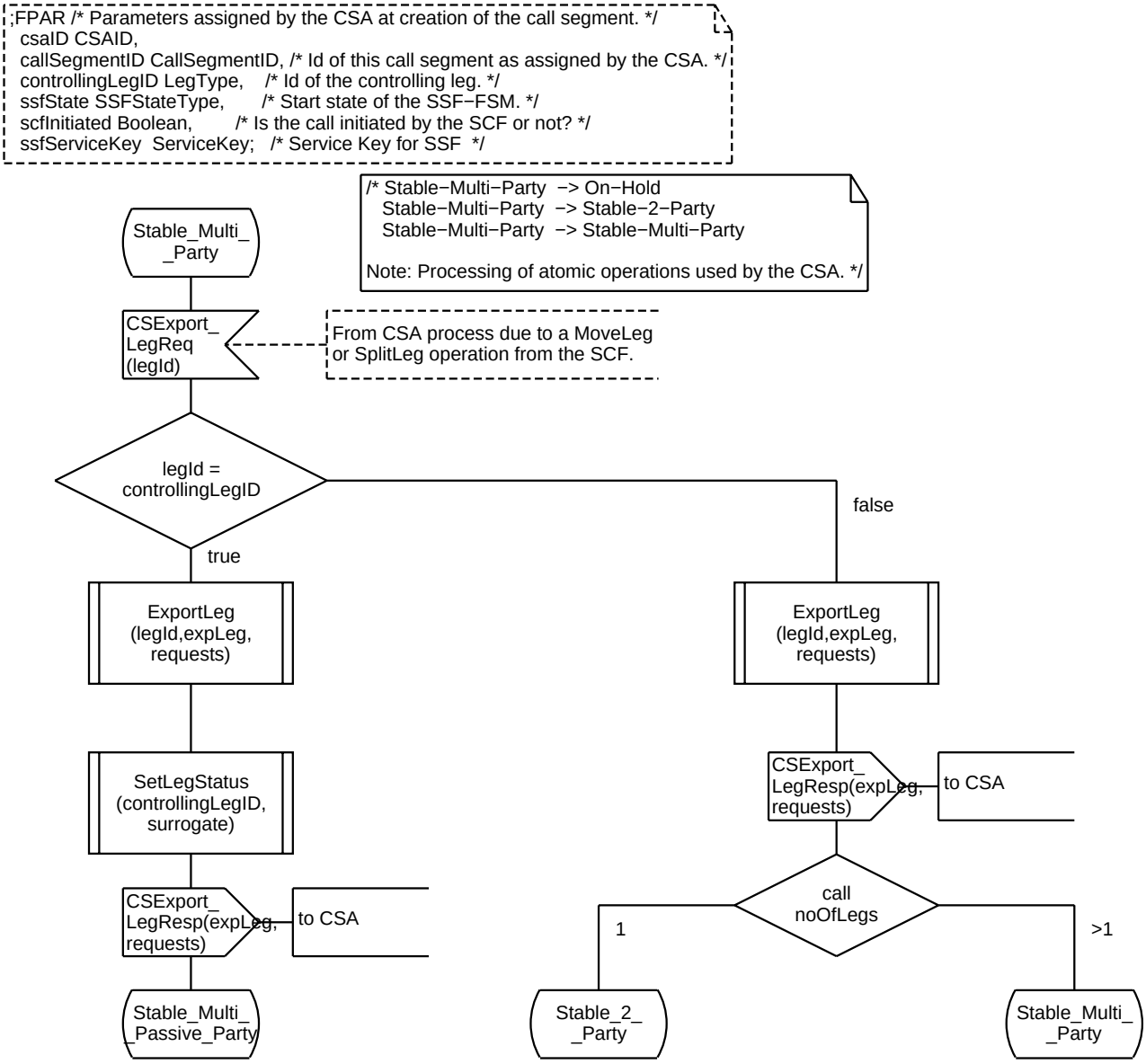
Process Type CallSegment 26(63)



Process Type CallSegment 27(63)

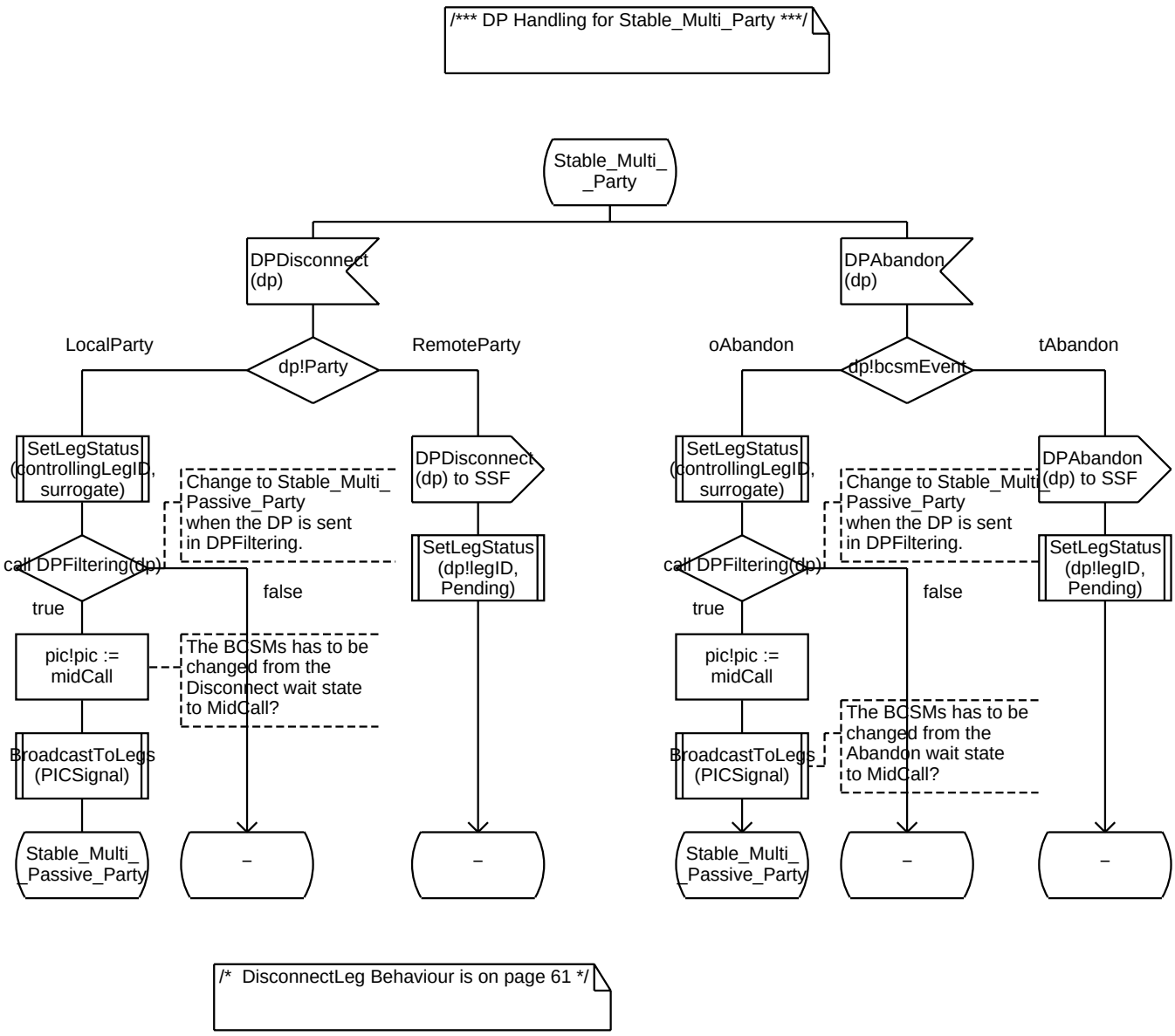


Process Type CallSegment 28(63)



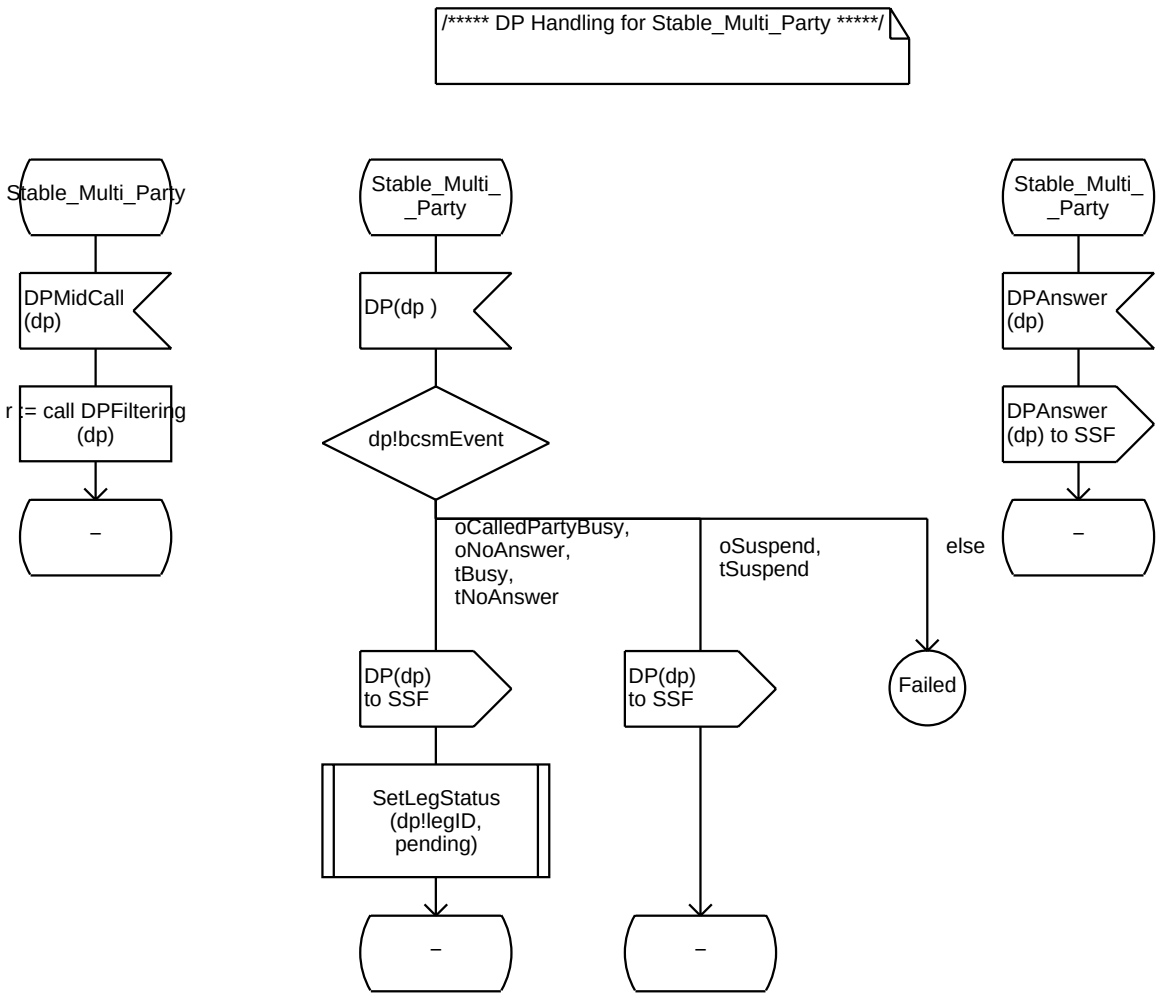
Process Type CallSegment 29(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



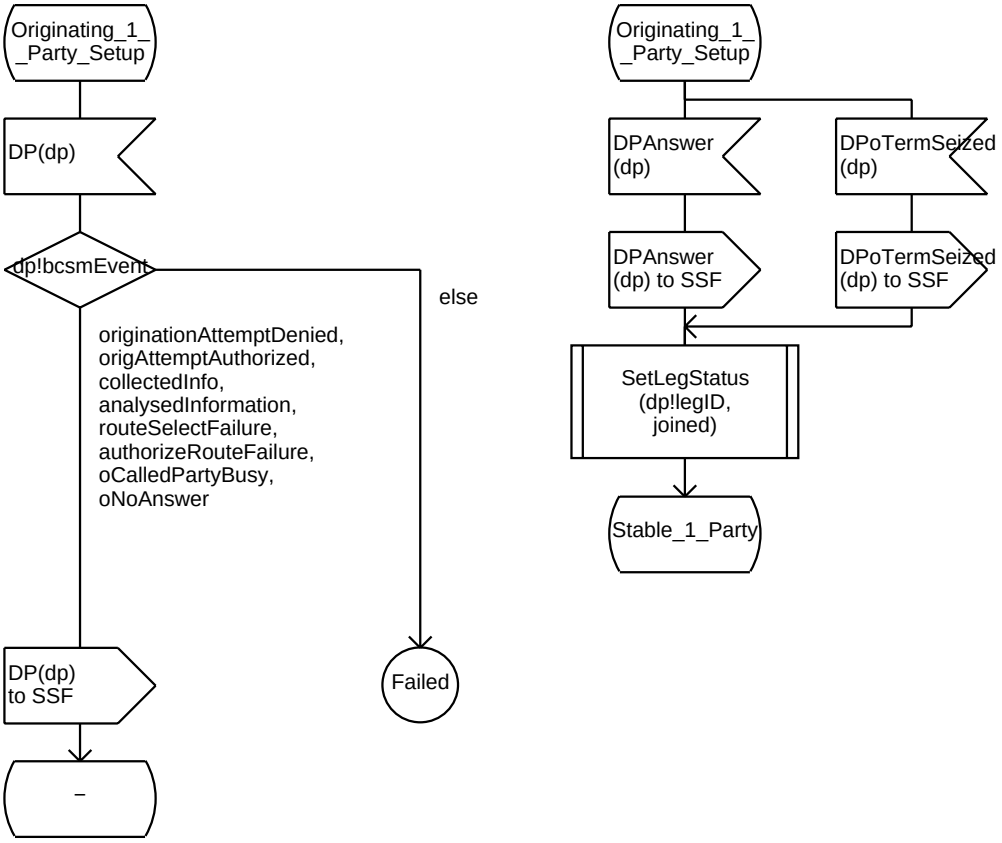
Process Type CallSegment 30(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

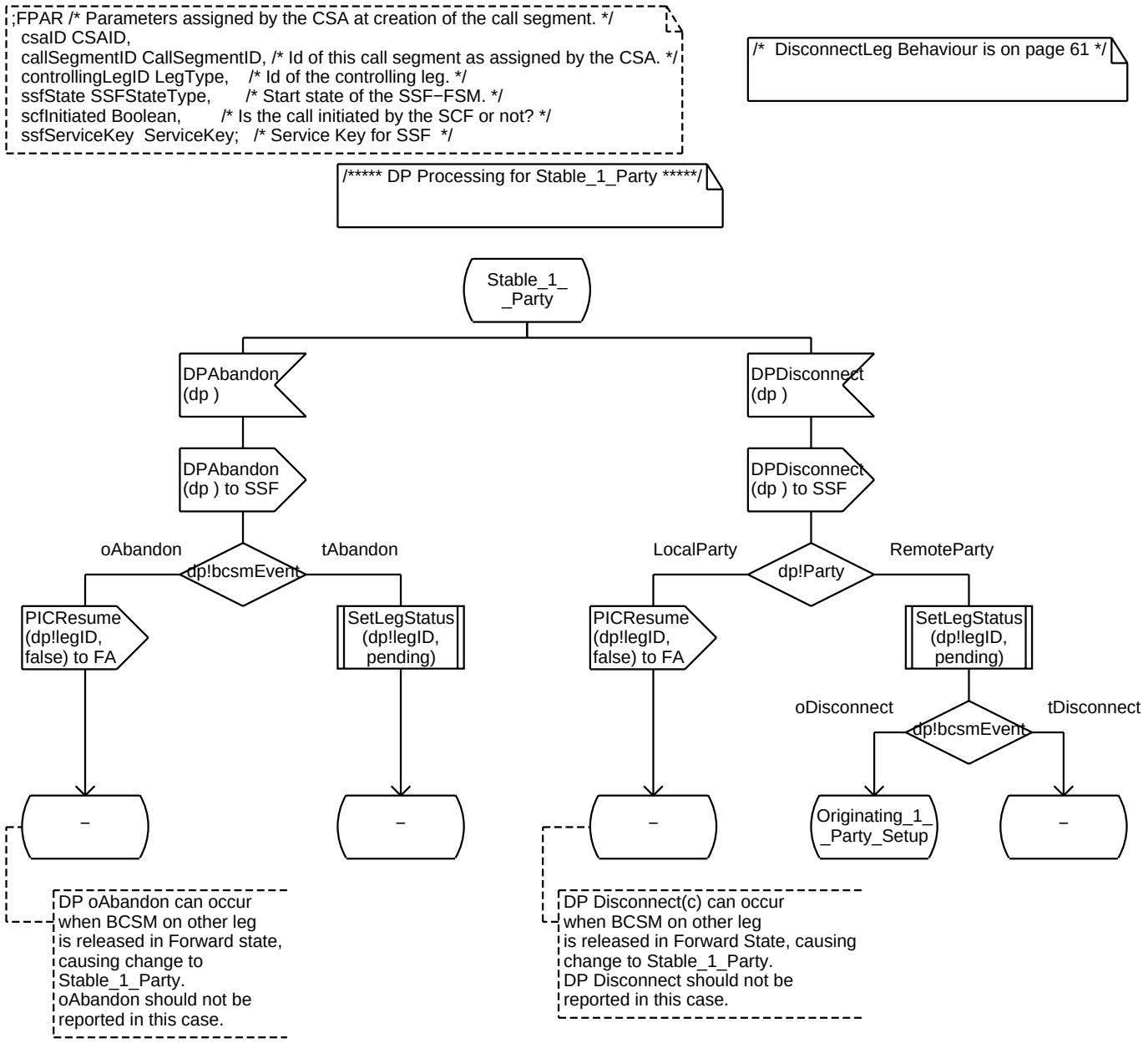


Process Type CallSegment 31(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

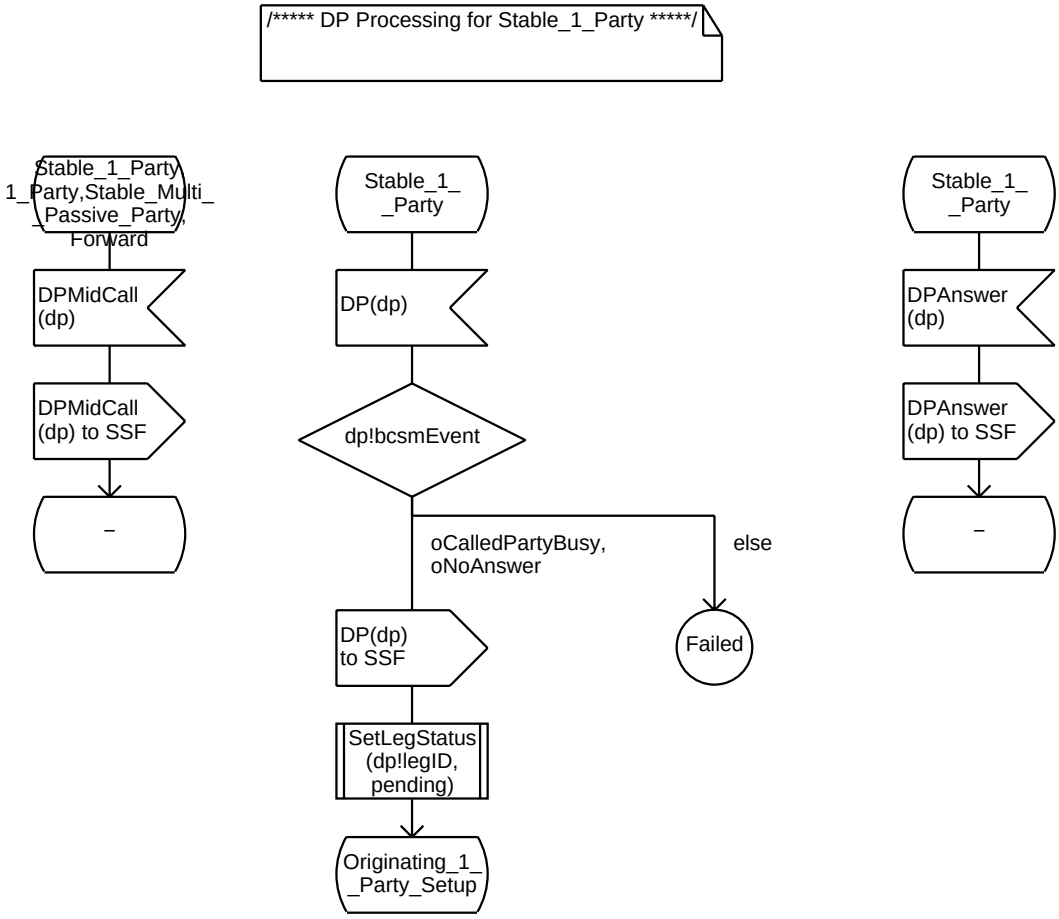


Process Type CallSegment 32(63)

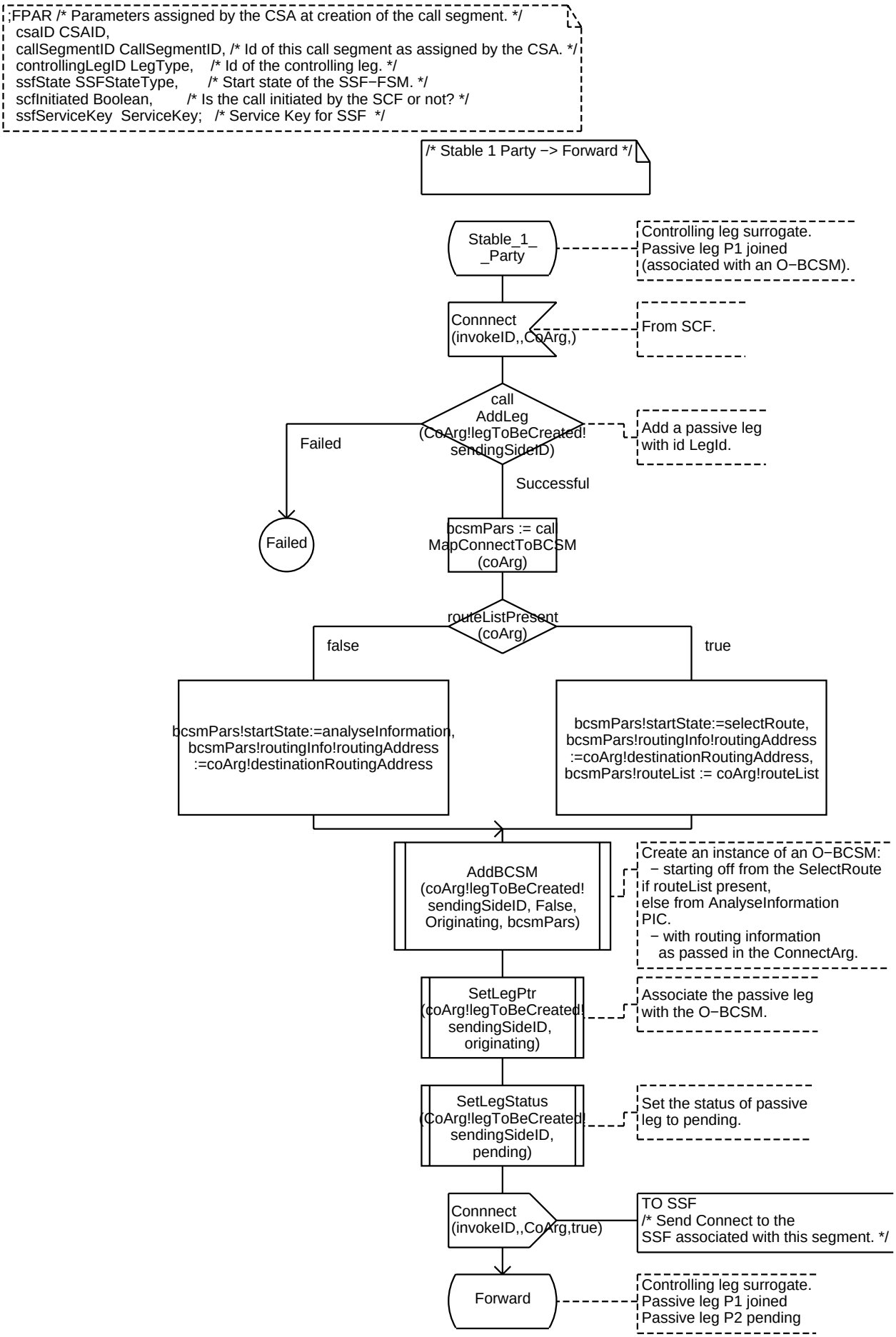


Process Type CallSegment 33(63)

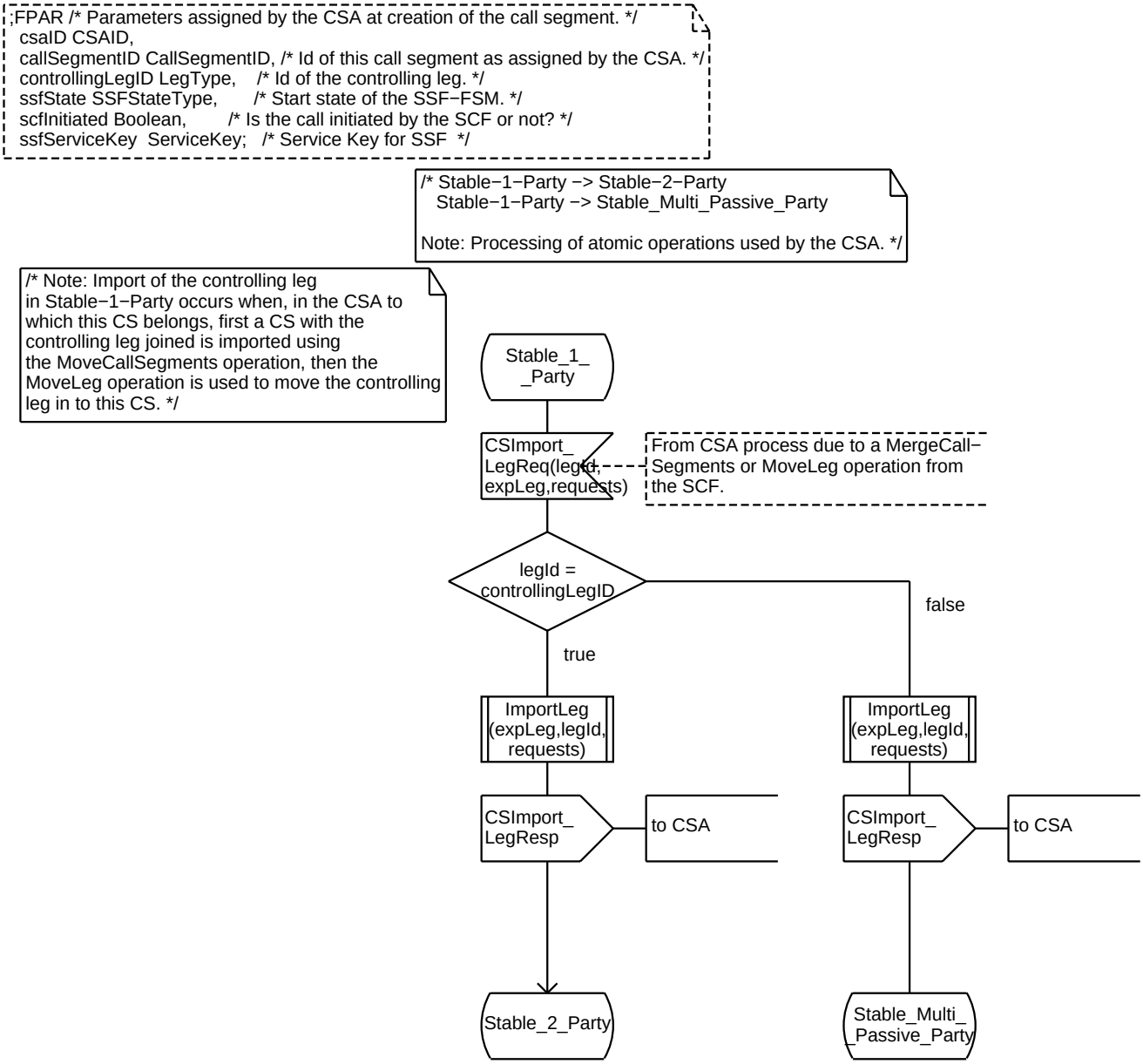
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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



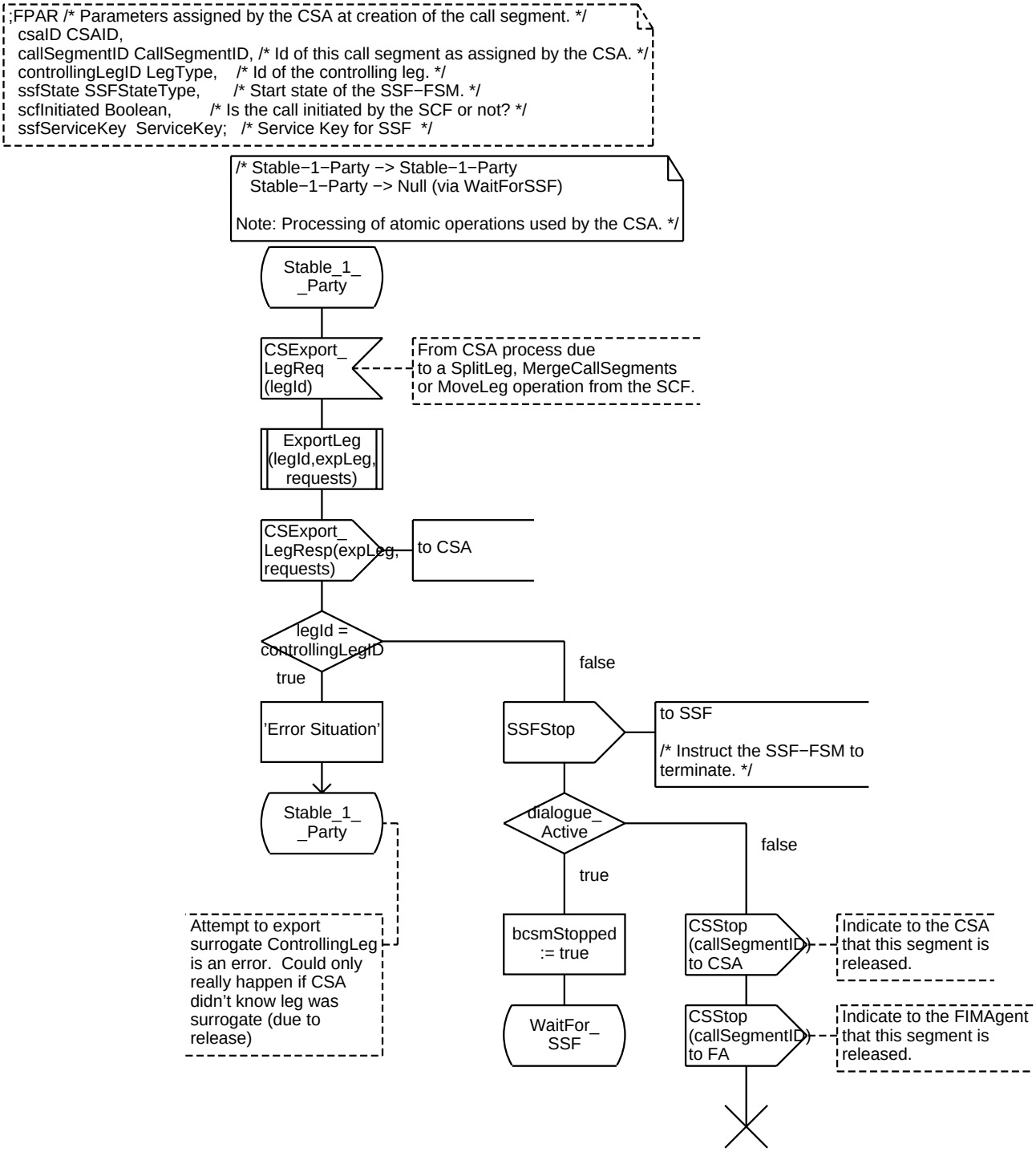
Process Type CallSegment 34(63)



Process Type CallSegment 35(63)

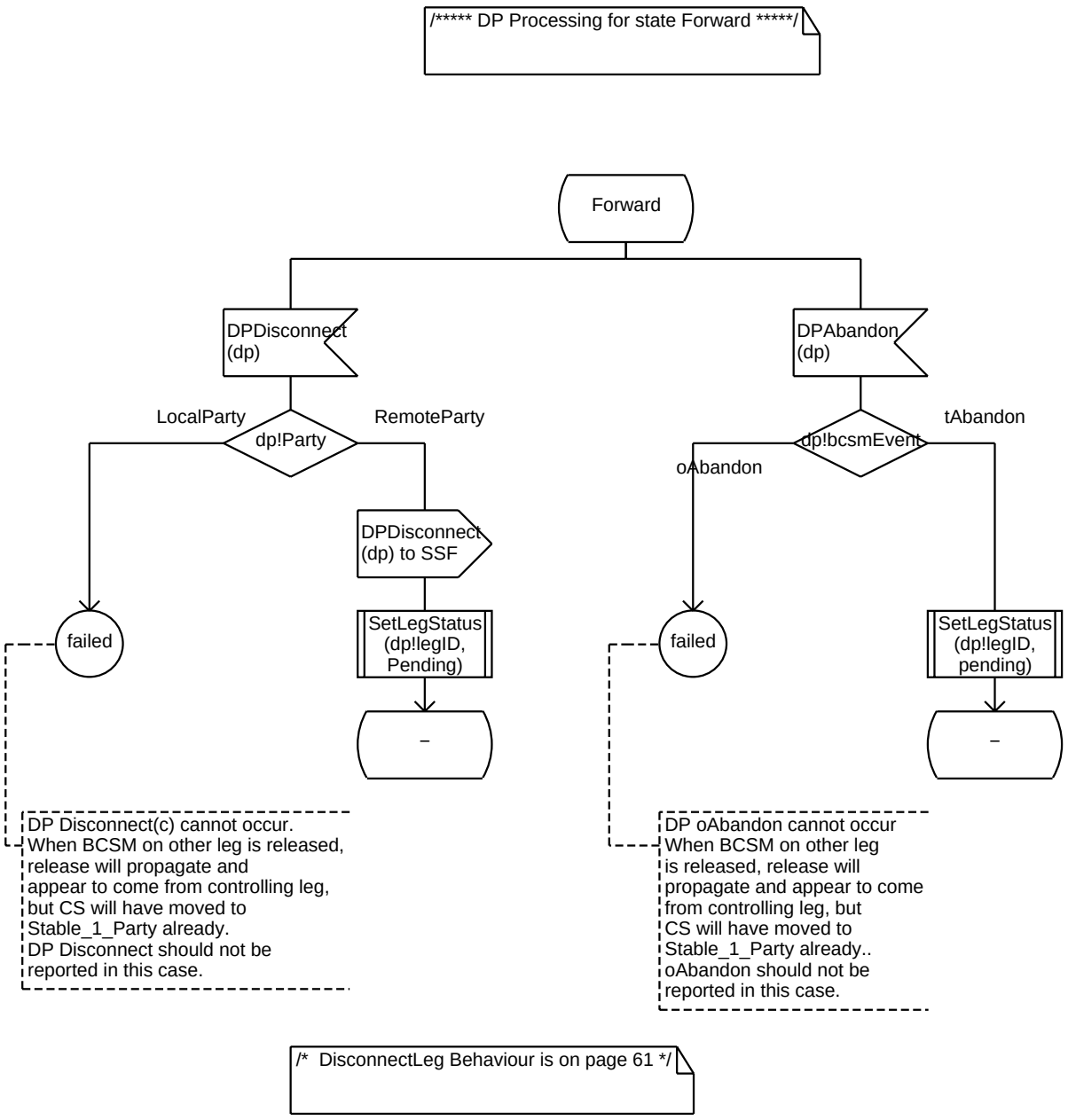


Process Type CallSegment 36(63)



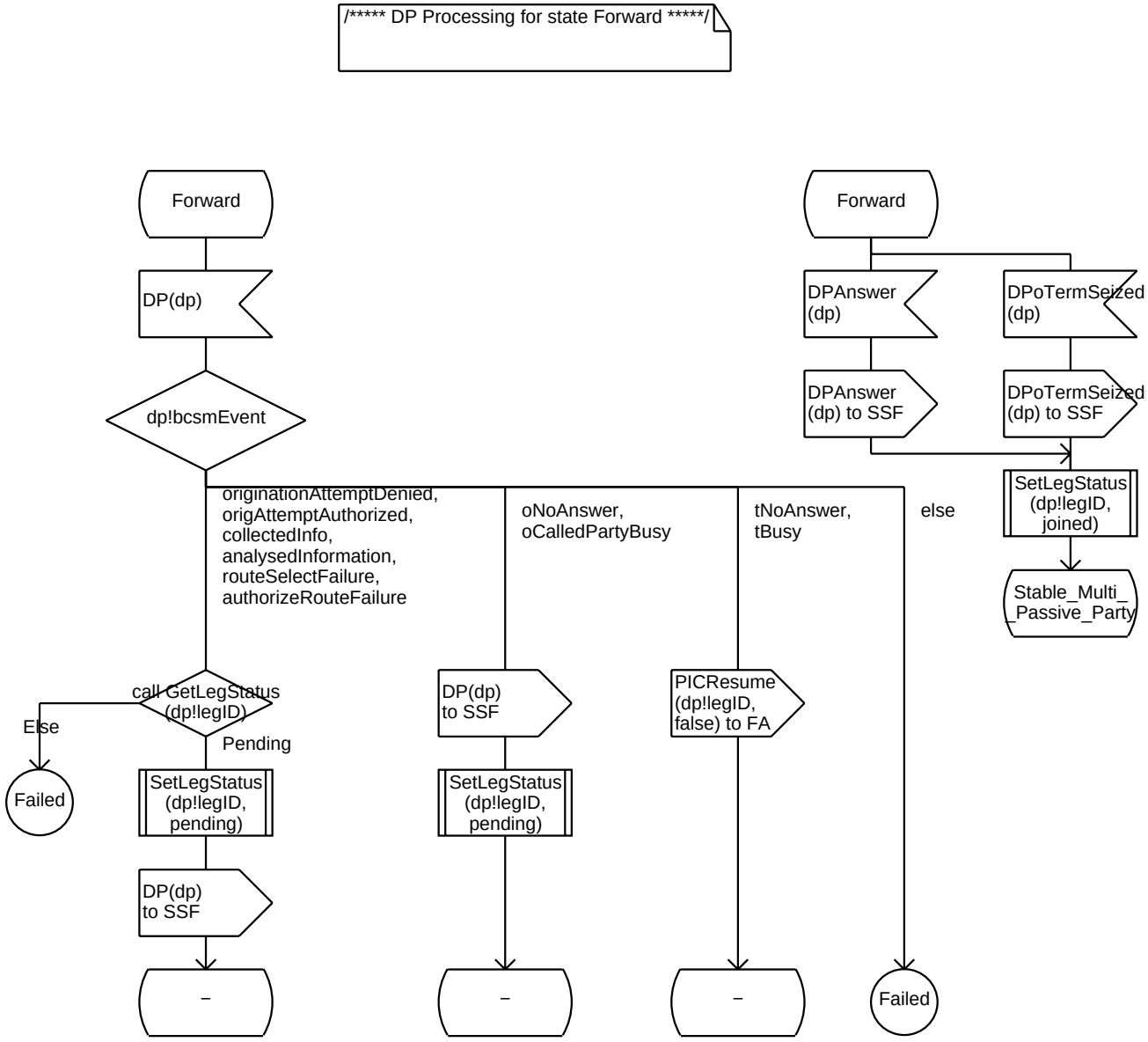
Process Type CallSegment 37(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



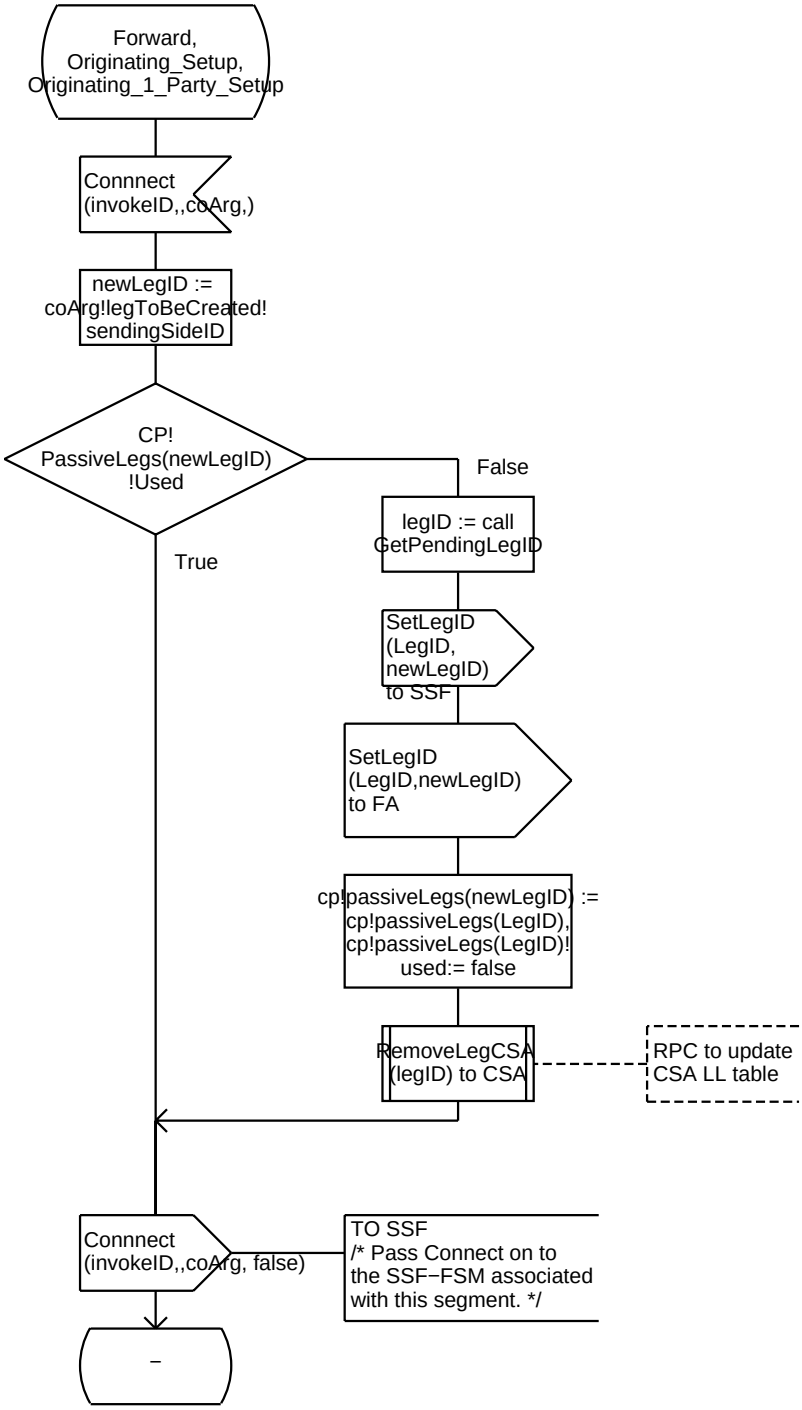
Process Type CallSegment 38(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

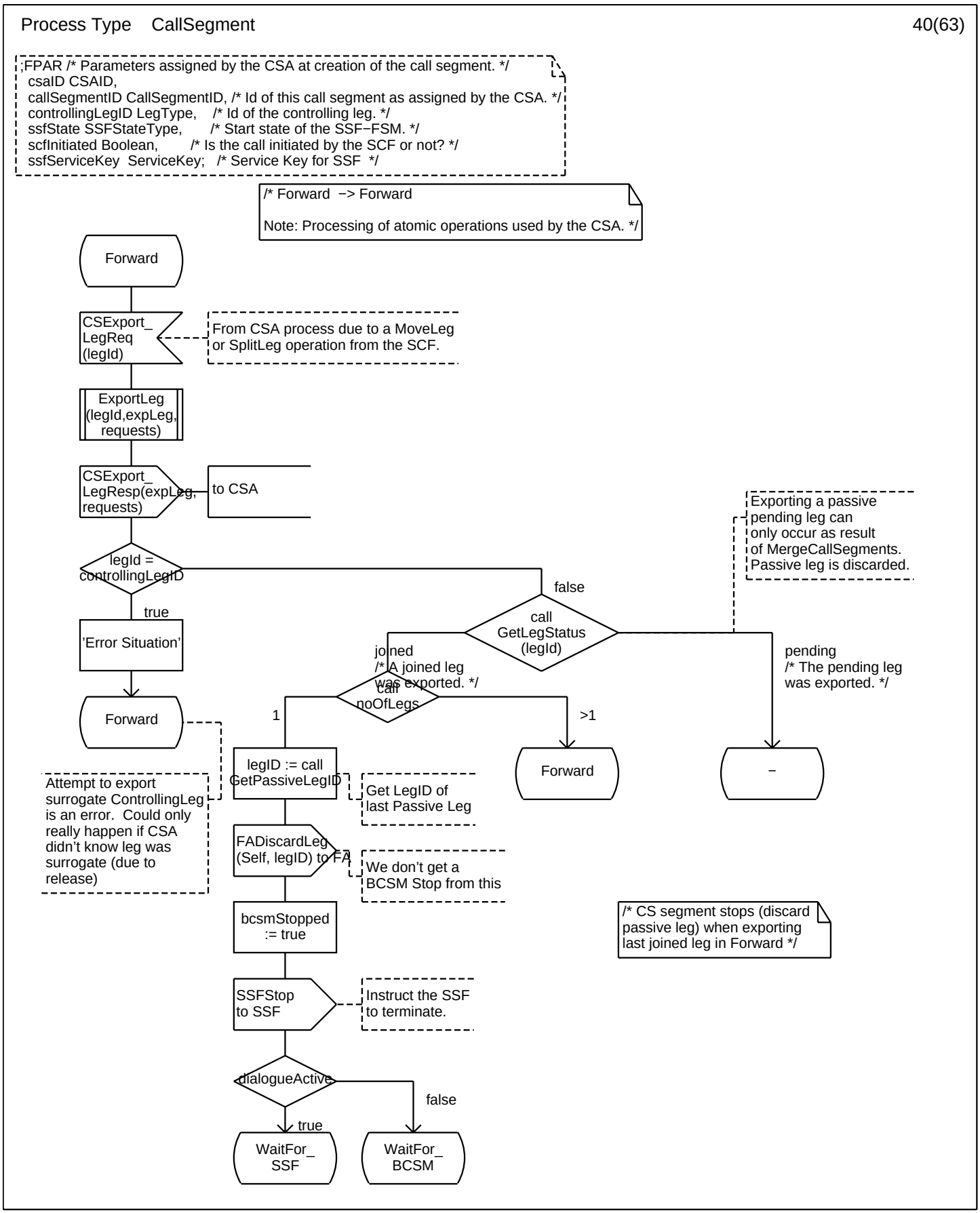


Process Type CallSegment 39(63)

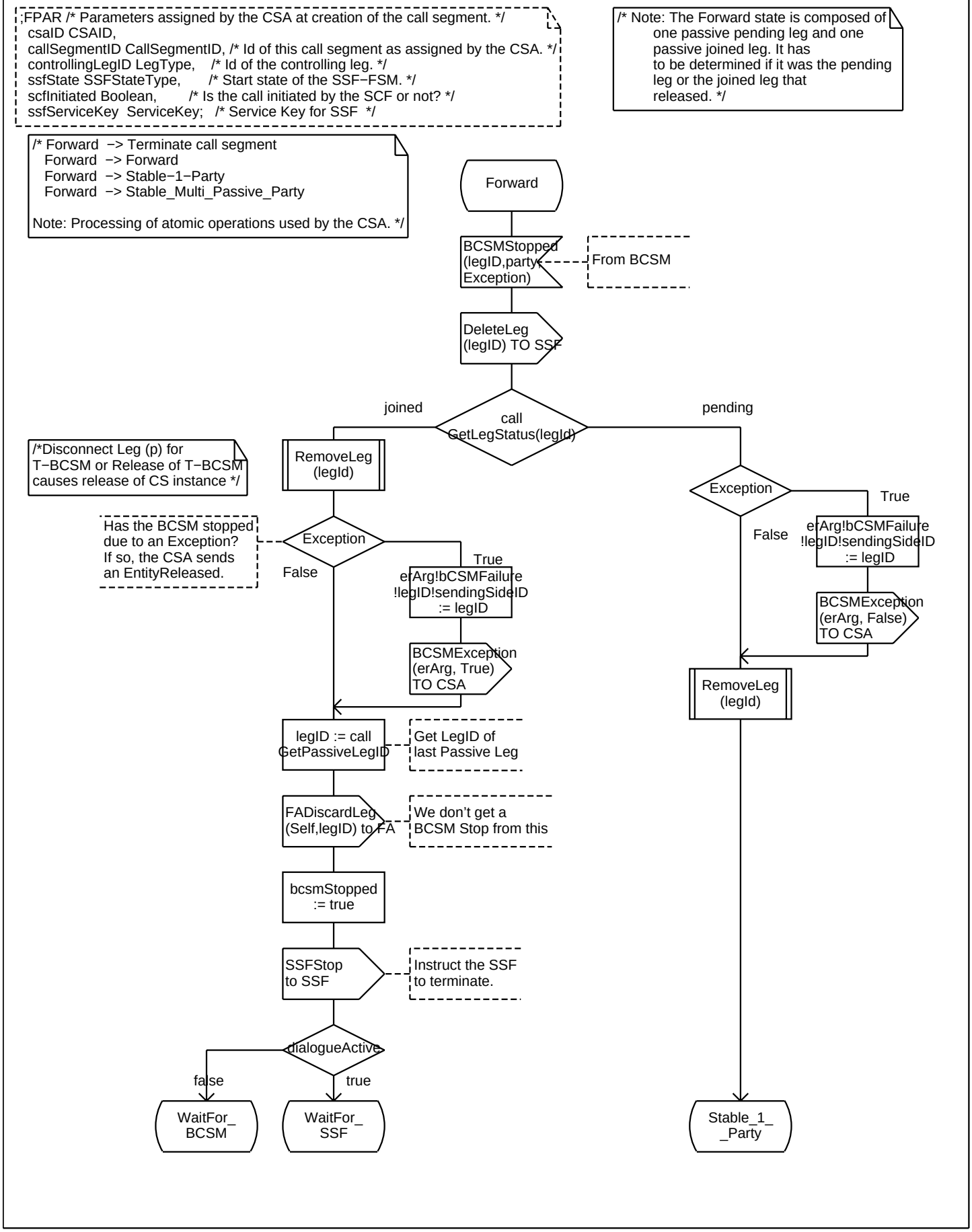
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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



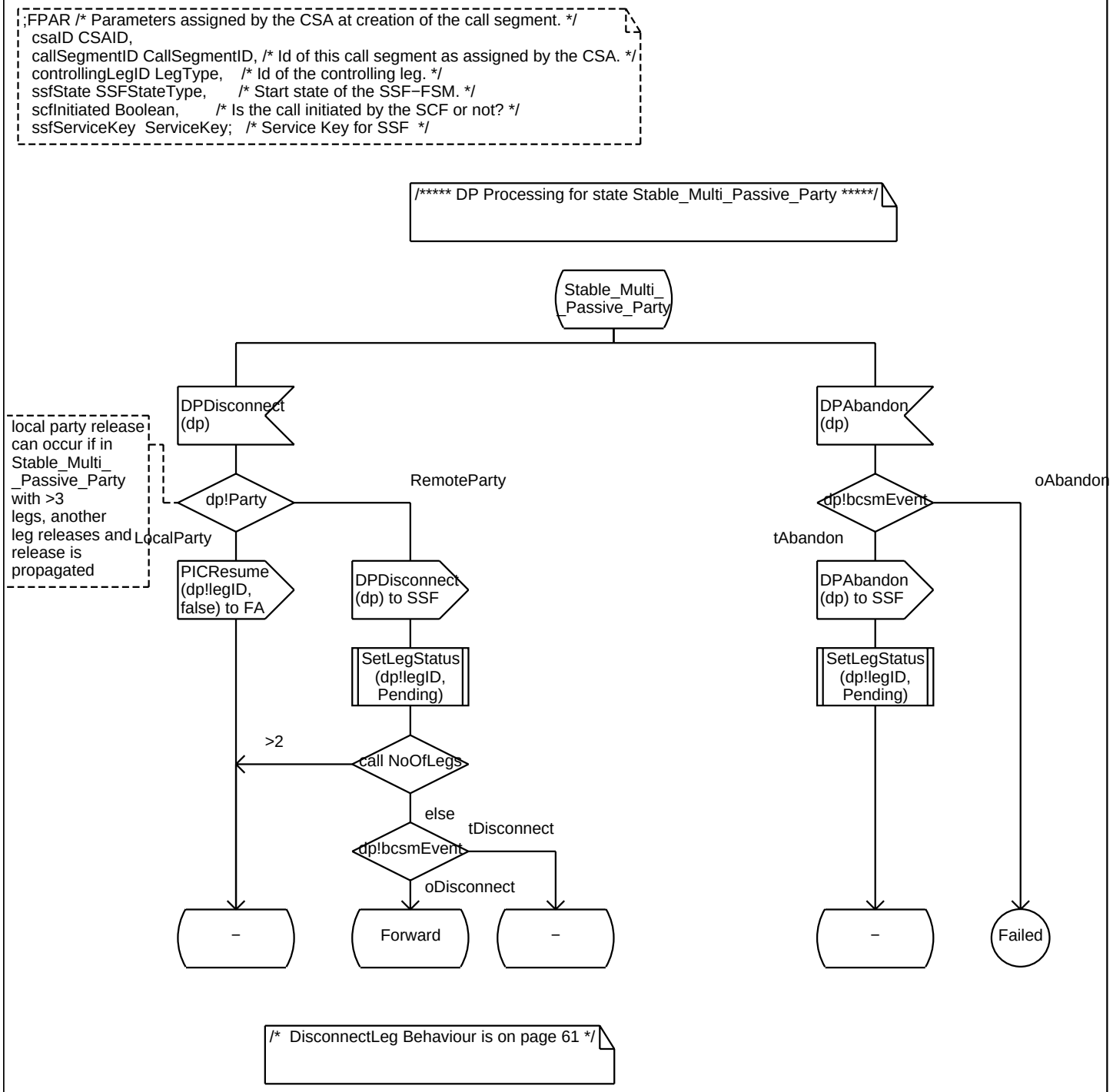
Process Type CallSegment 40(63)



Process Type CallSegment 41(63)

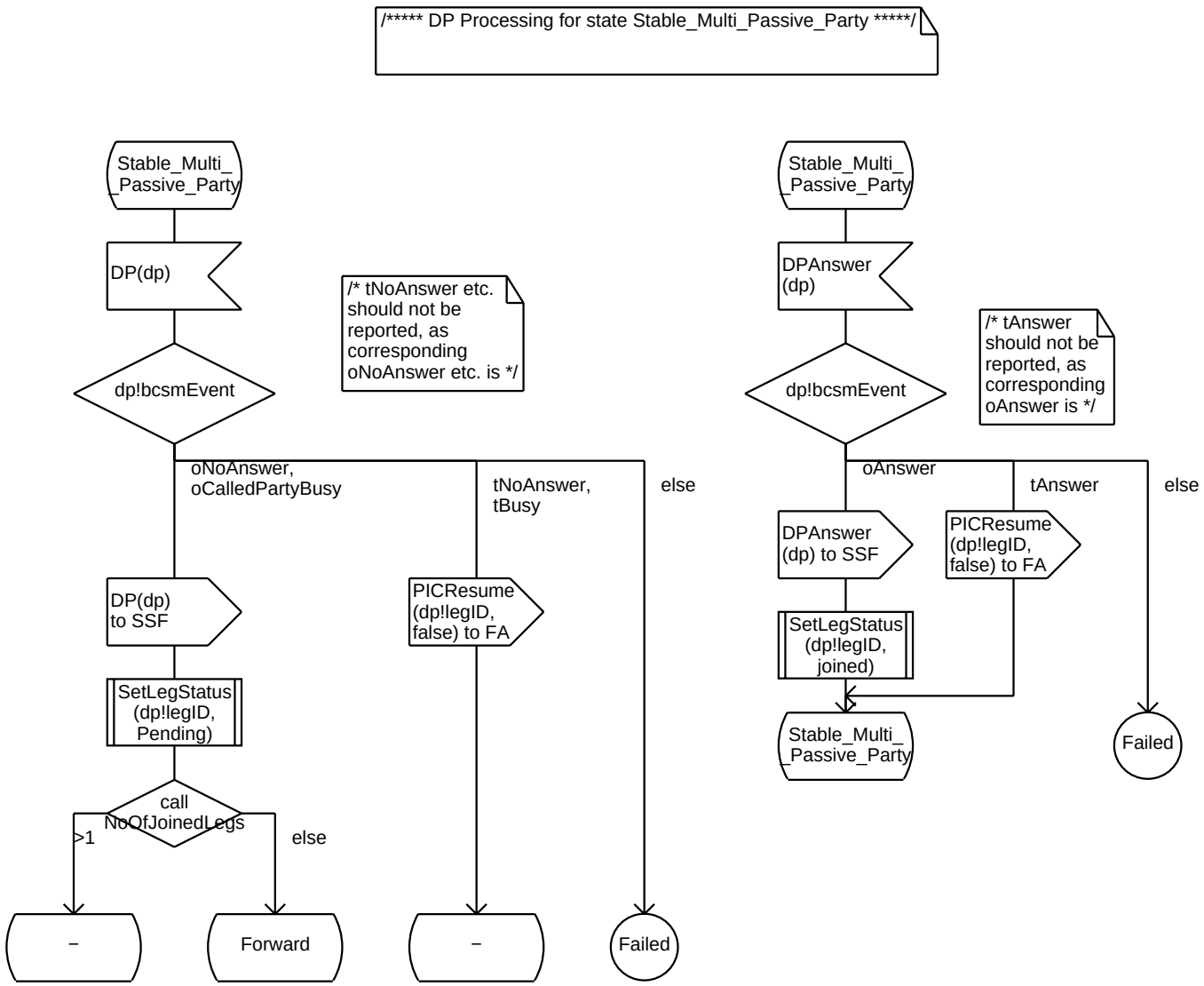


Process Type CallSegment 42(63)



Process Type CallSegment 43(63)

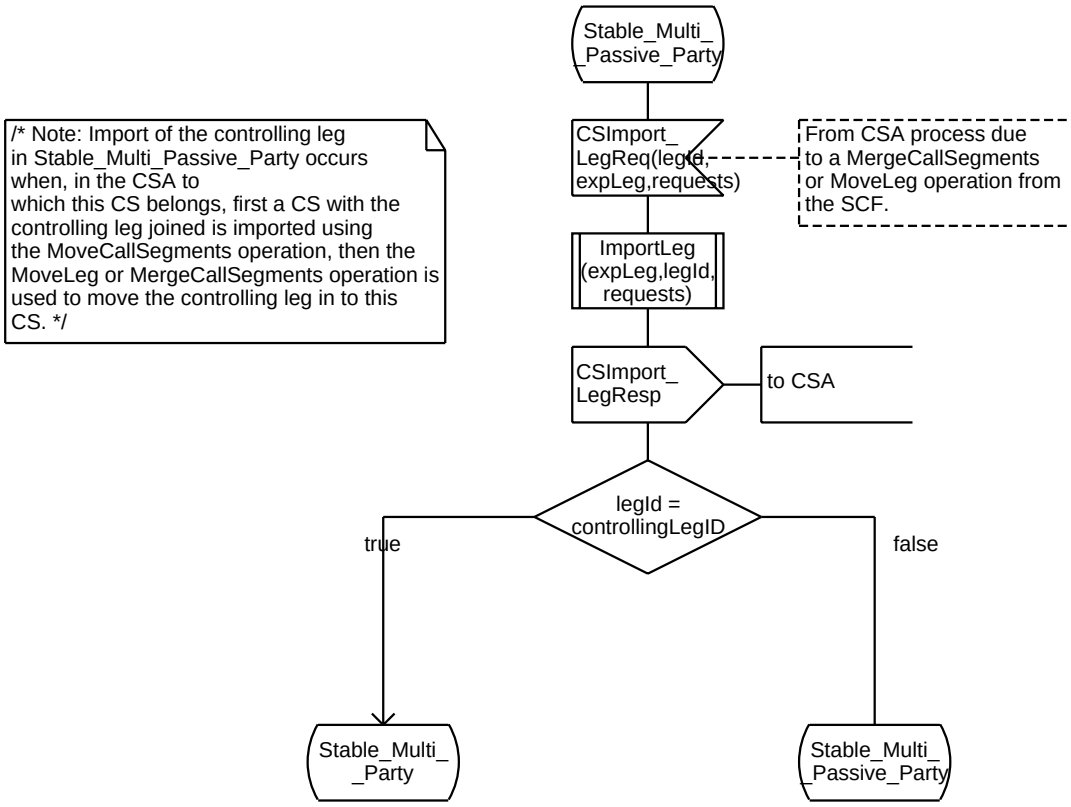
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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



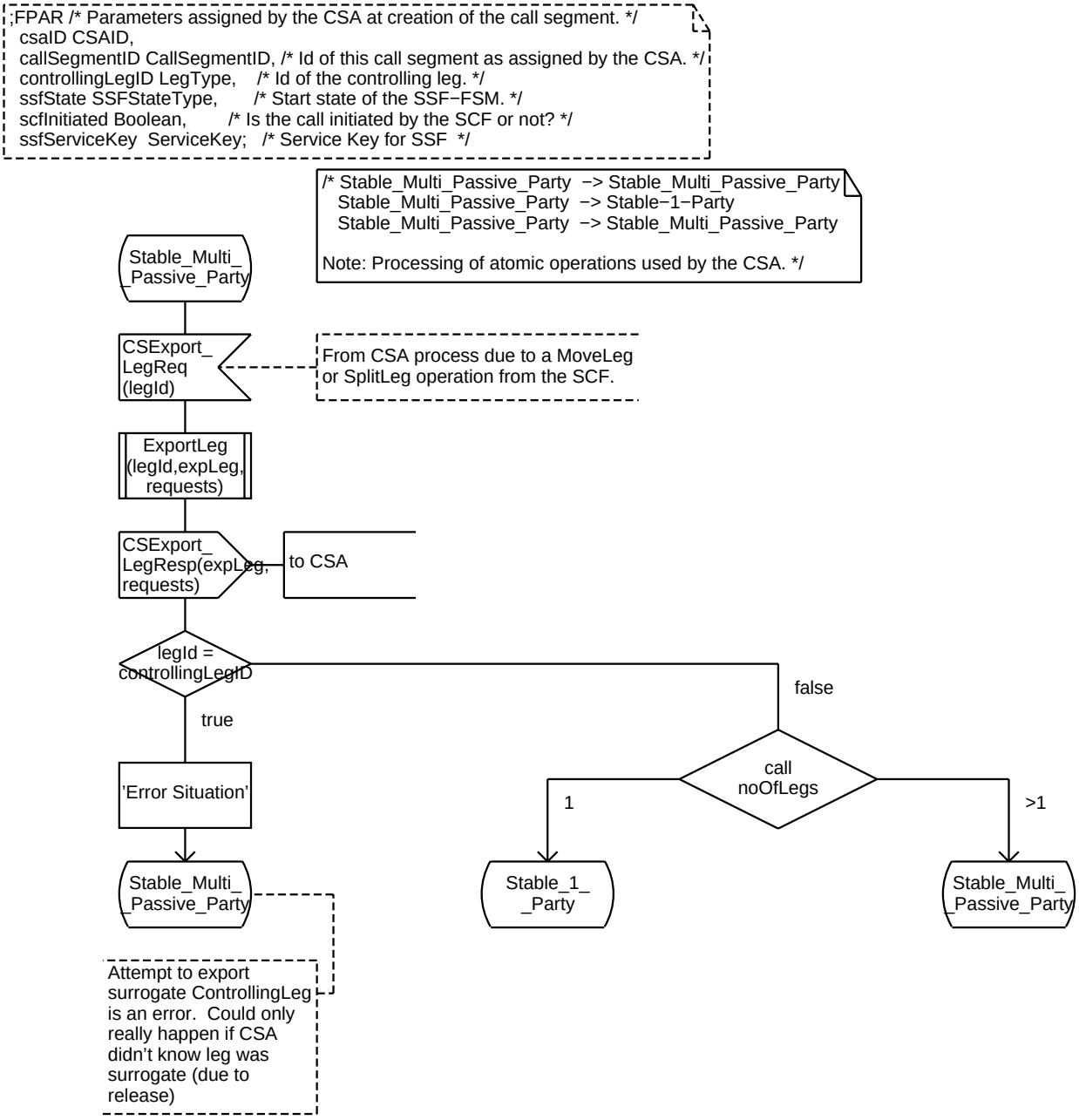
Process Type CallSegment 44(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

```
/* Stable_Multi_Passive_Party  −> Stable−Multi−Party
   Stable_Multi_Passive_Party  −> On Hold

Note: Processing of atomic operations used by the CSA. */
```



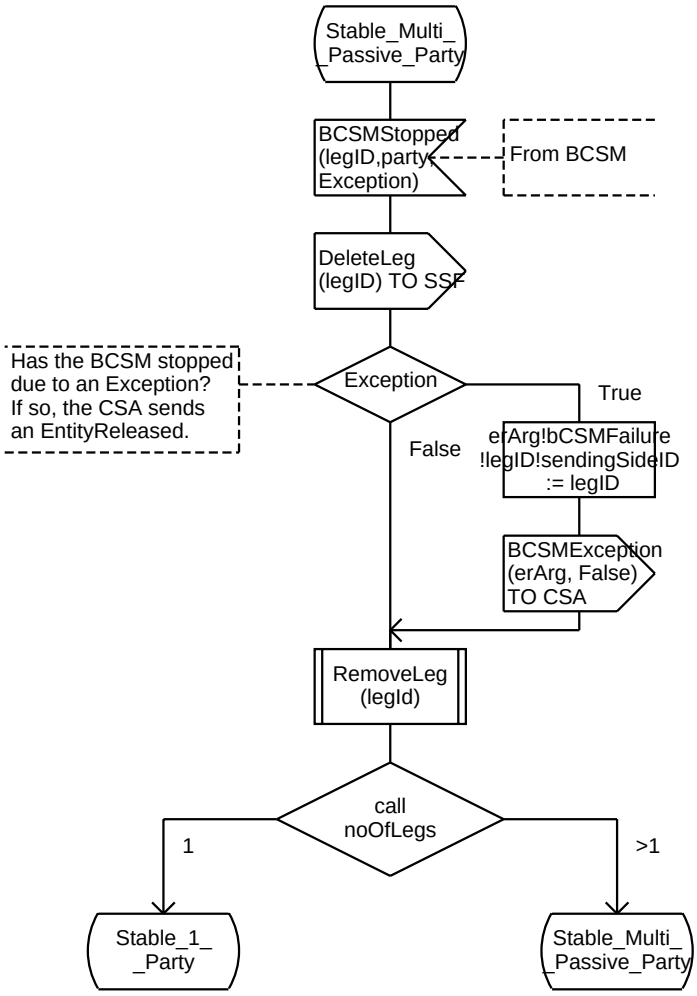
Process Type CallSegment 45(63)



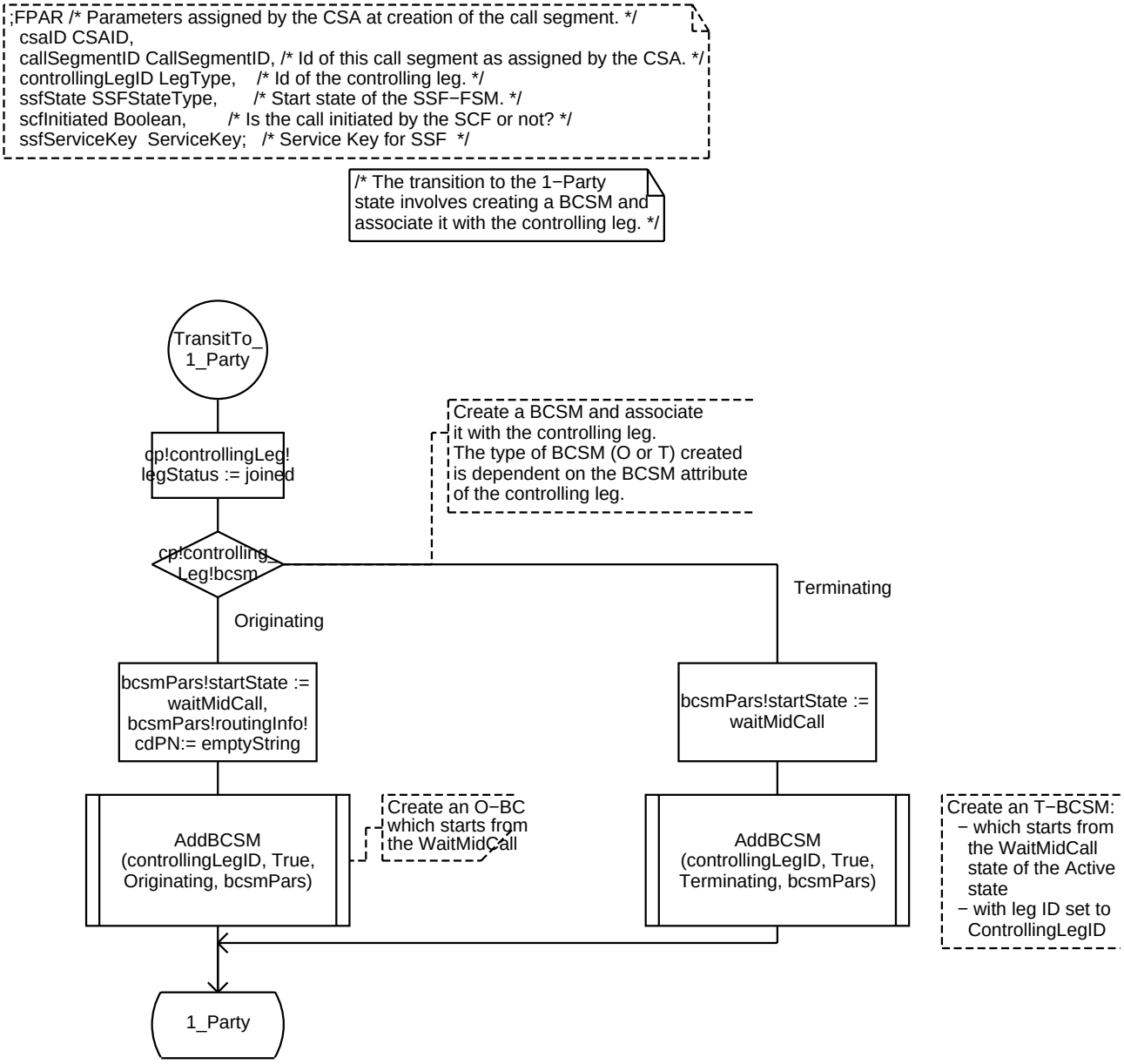
Process Type CallSegment 46(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

```
/* Stable_Multi_Passive_Party  −> Stable−1−Party
   Stable_Multi_Passive_Party  −> Stable_Multi_Passive_Party

Note: Processing of atomic operations used by the CSA. */
```



Process Type CallSegment 47(63)

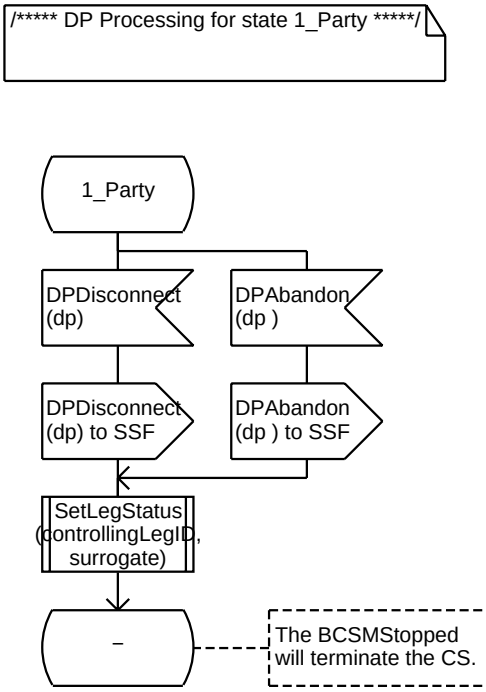


Process Type CallSegment

Create an O−BCSM:
which starts from
the WaitMidCall
state of the Active
state

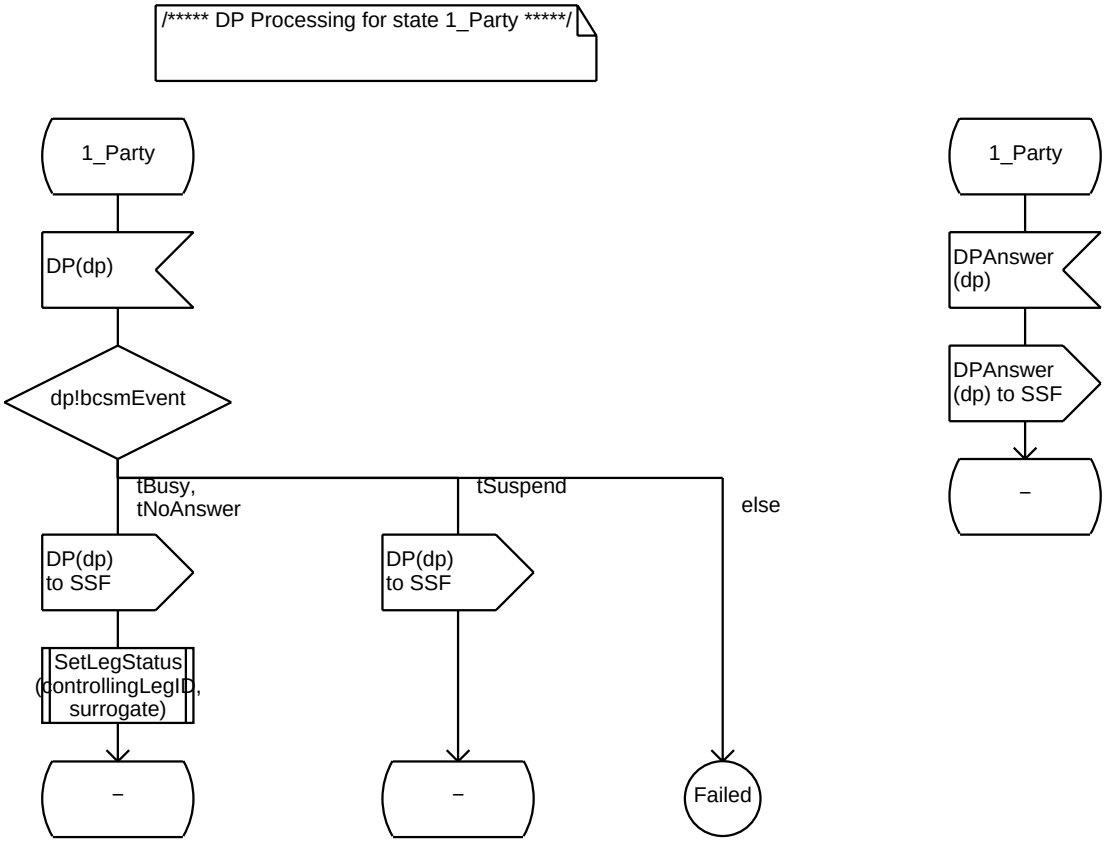
Process Type CallSegment 48(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

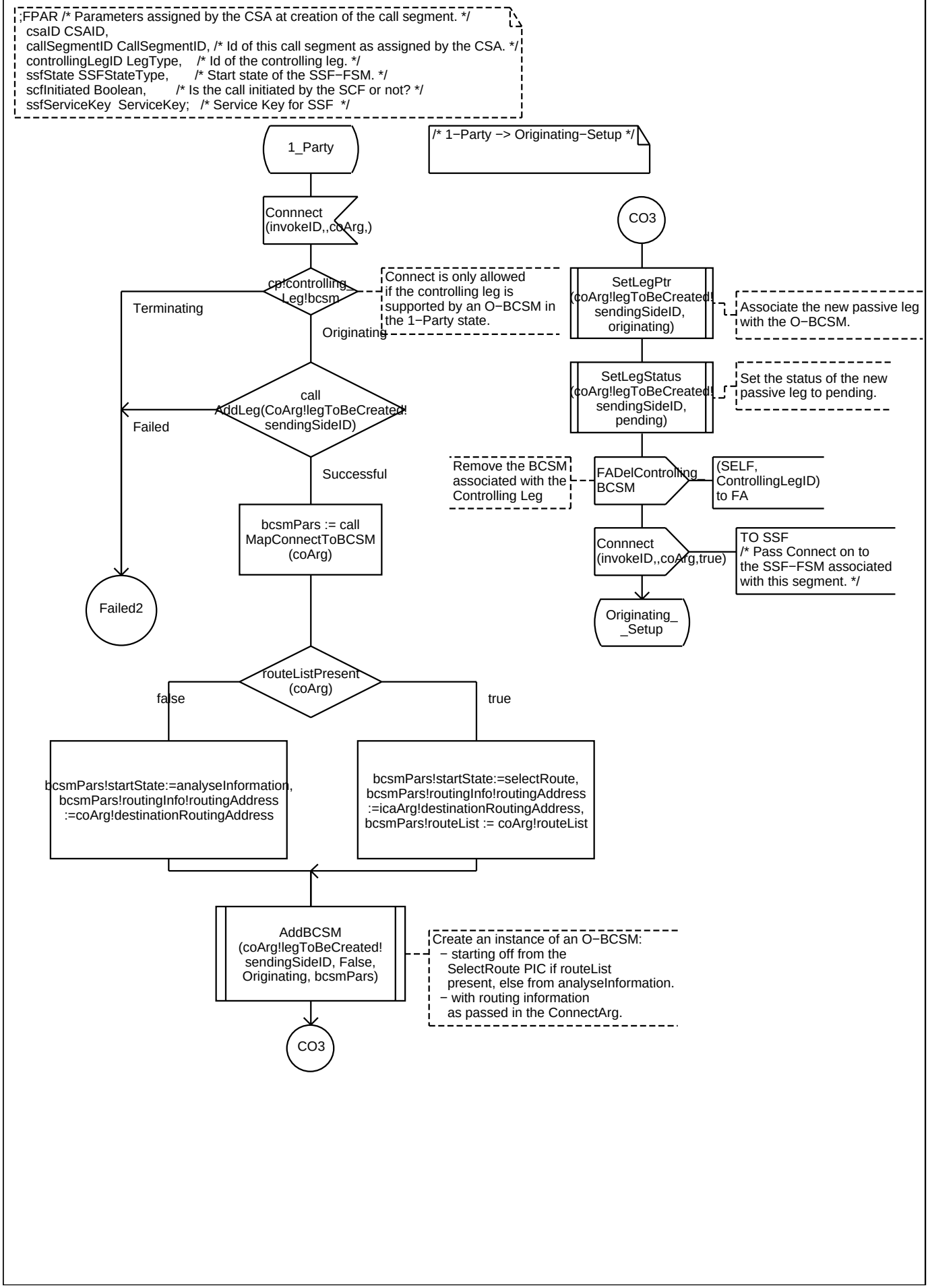


Process Type CallSegment 49(63)

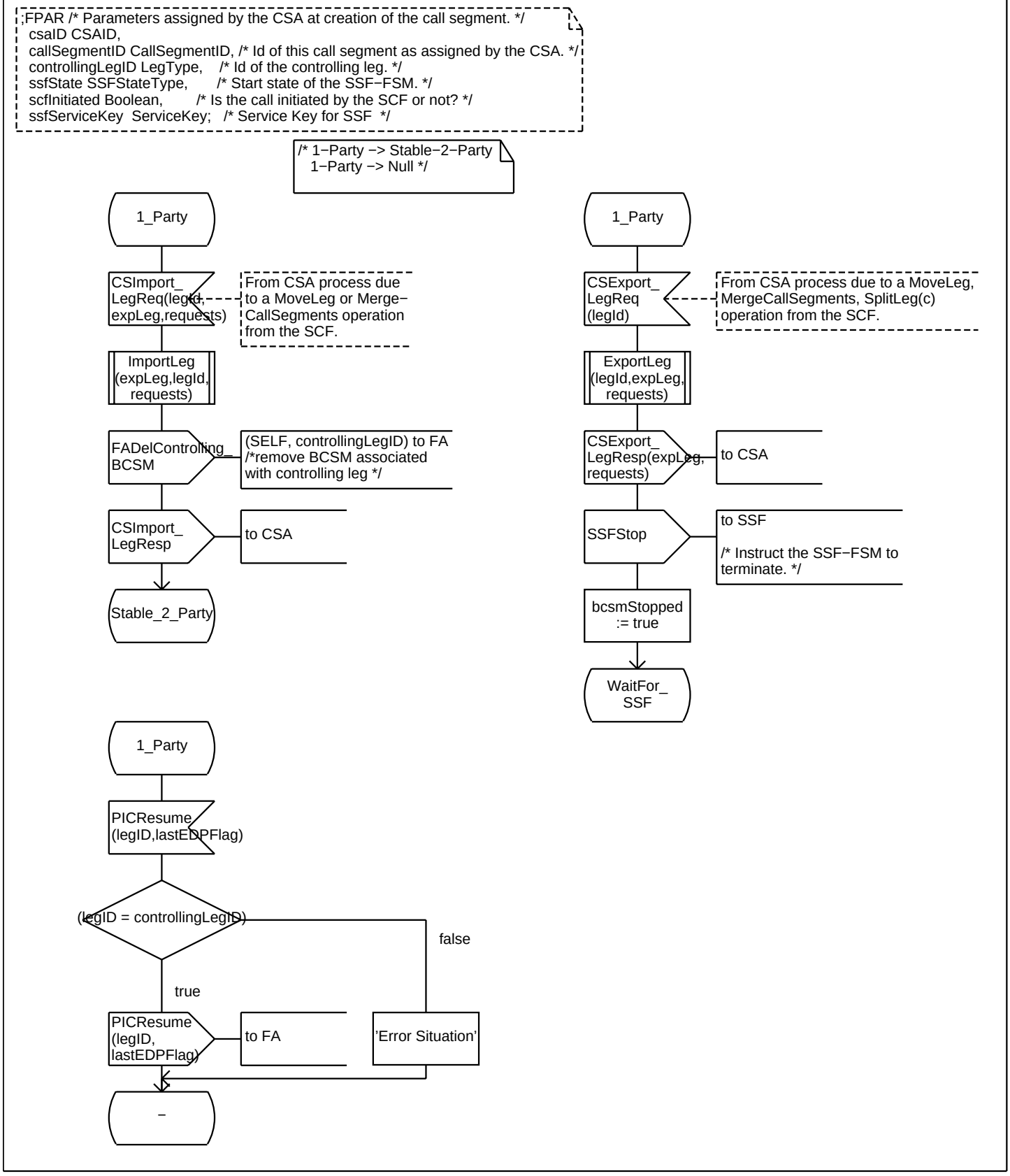
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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



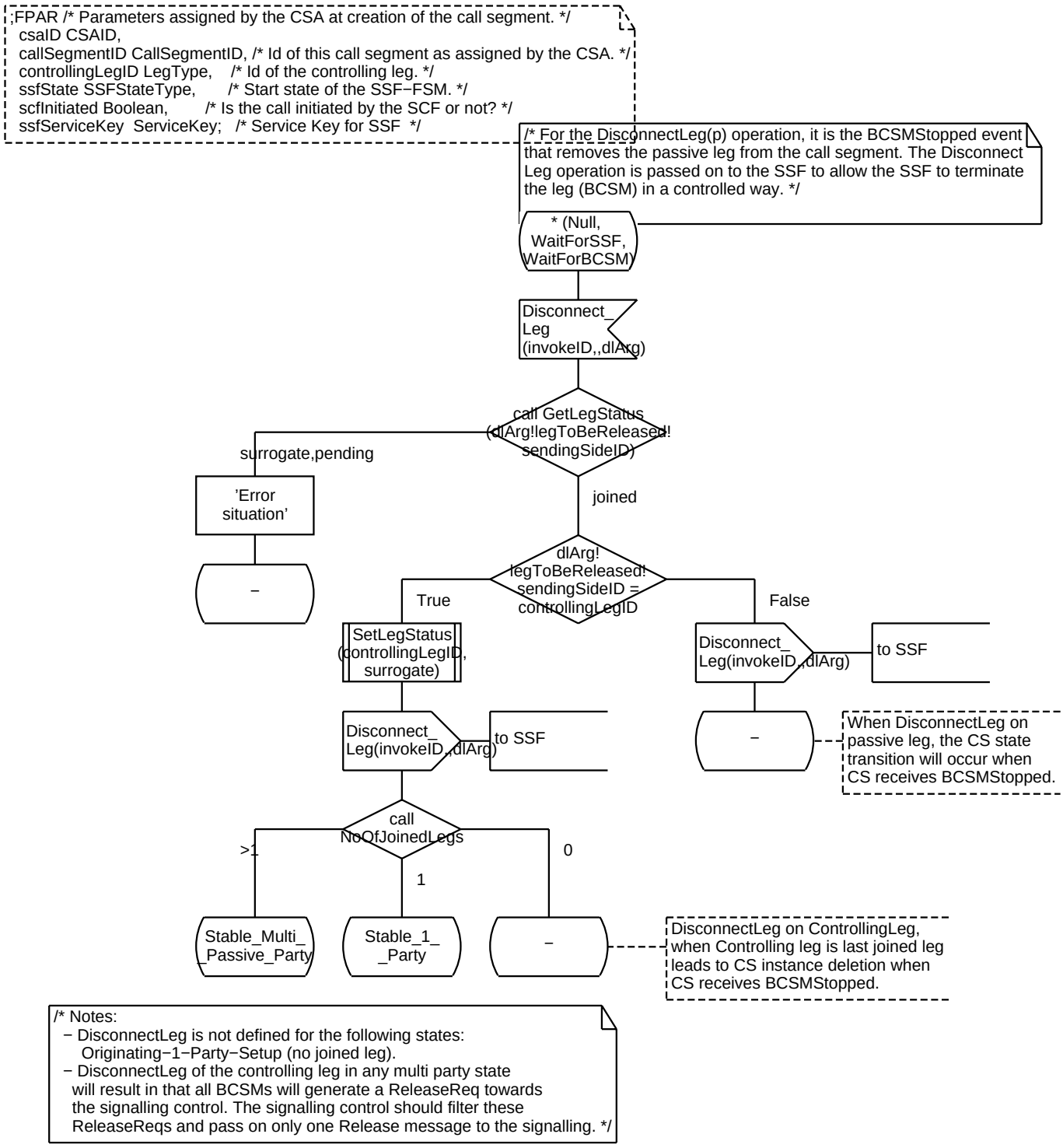
Process Type CallSegment 50(63)



Process Type CallSegment 51(63)



Process Type CallSegment 52(63)

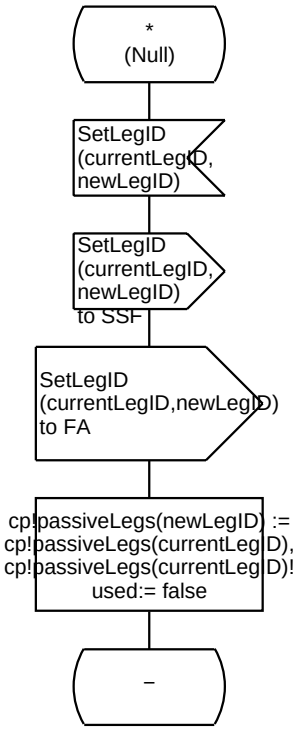


Process Type CallSegment 53(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

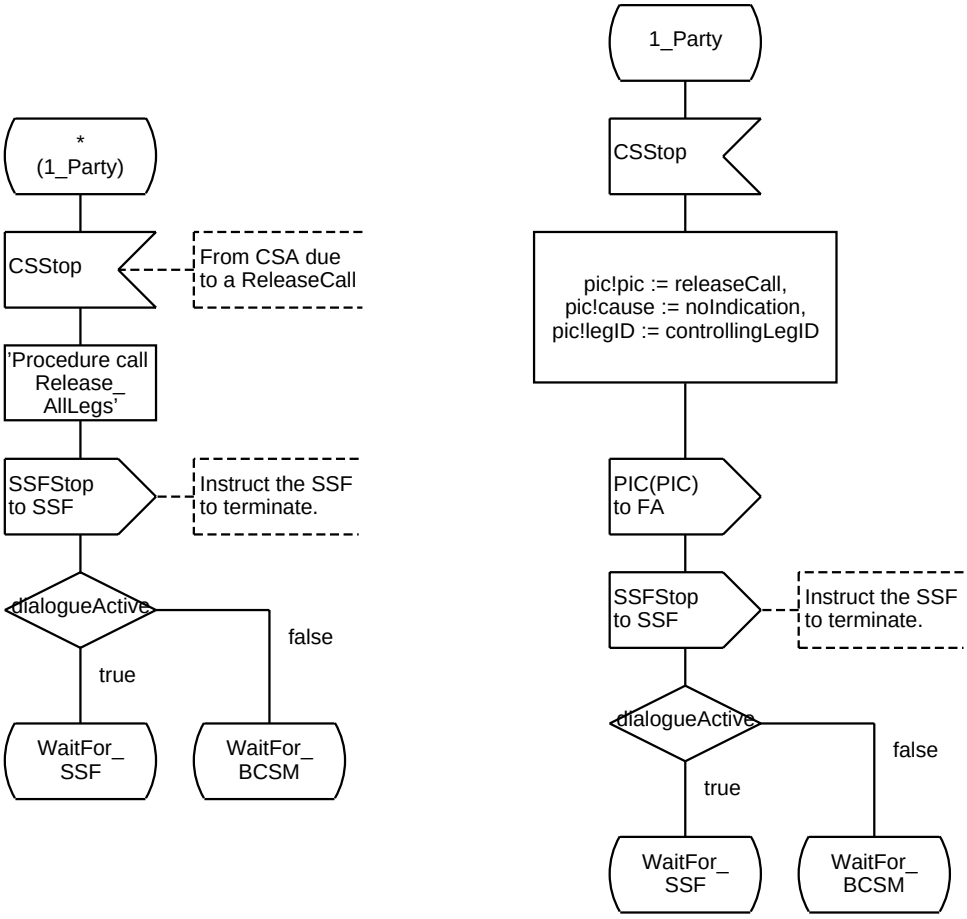
```
/* When the MoveCallSegments operation is processed by the SSME_C and CSA,
the CS may be requested to renumber the legs. In such a case, the CSA
sends the signal SetLegID to the CS prior to exporting it into another
CSA. */
```

```
/*  When the MoveCallSegments operation is performed, the csaID of the CS needs to be changed. */
/*  The new csaID should also be sent to the FIMAgent using the SetCSAId signal. */
```



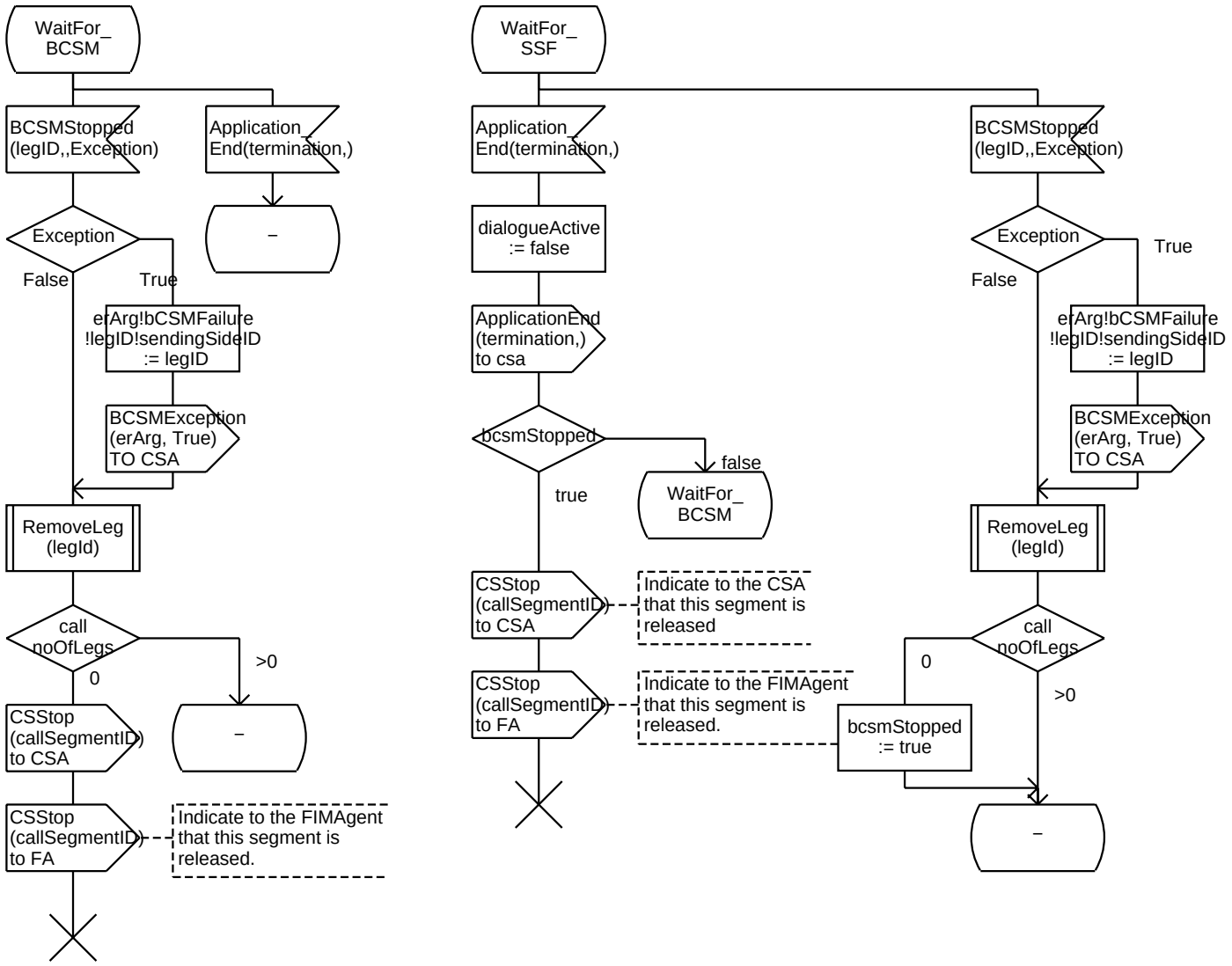
Process Type CallSegment 54(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



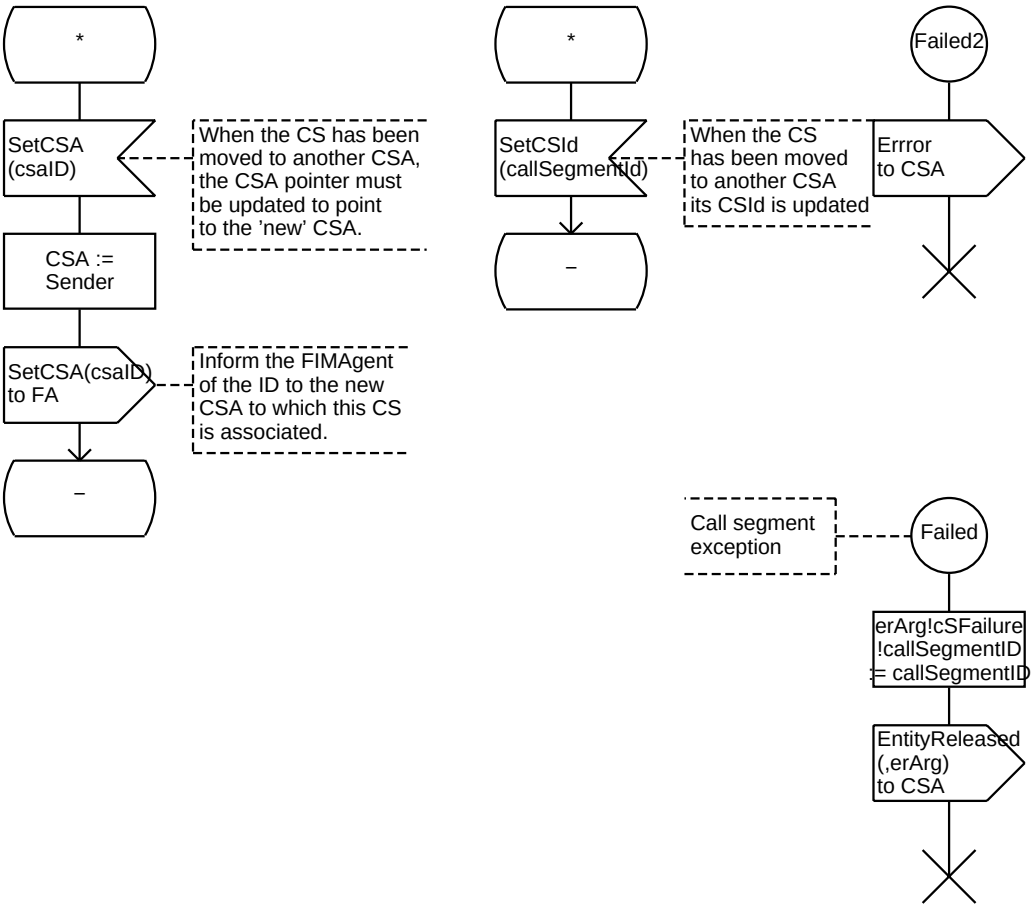
Process Type CallSegment 55(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



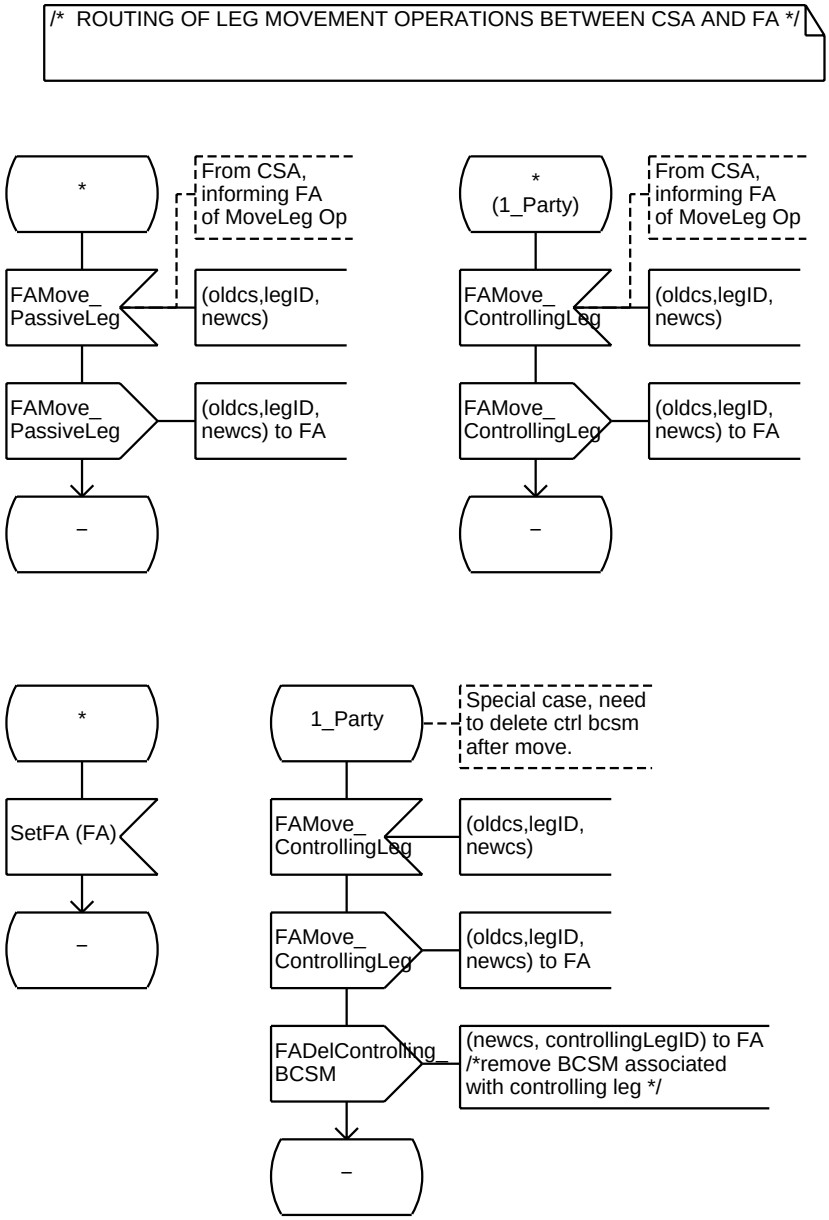
Process Type CallSegment 56(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

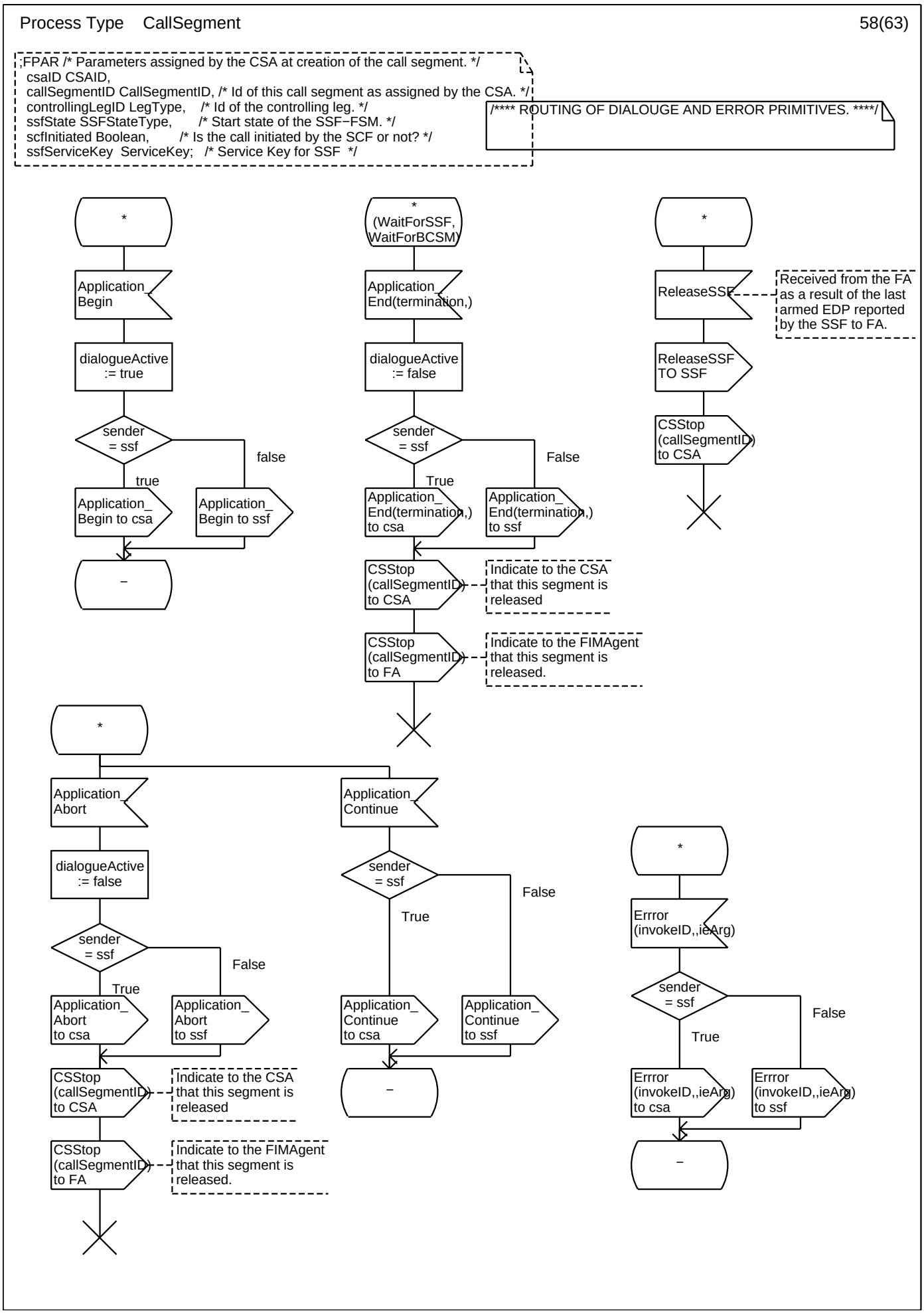


Process Type CallSegment 57(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

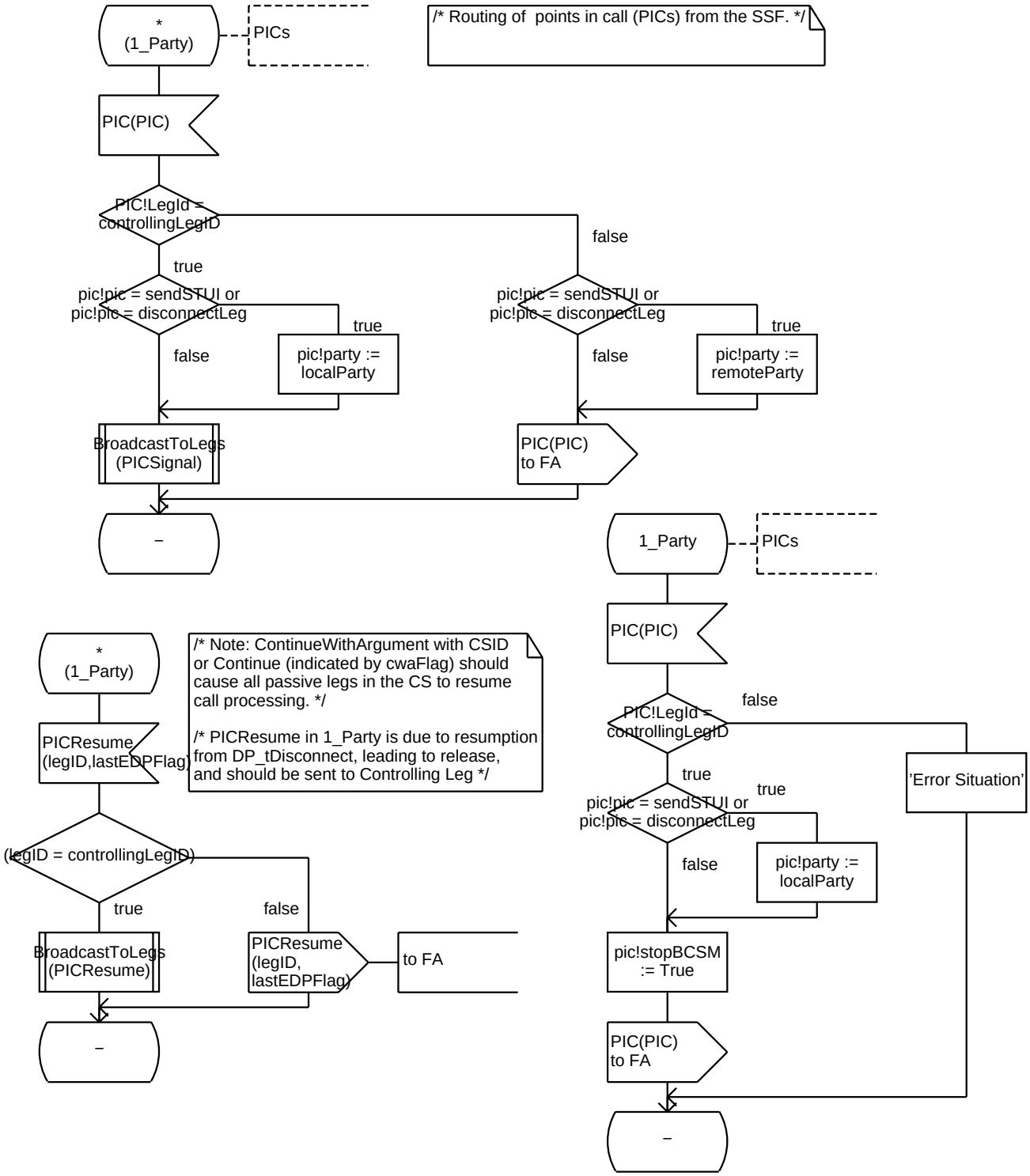


Process Type CallSegment 58(63)

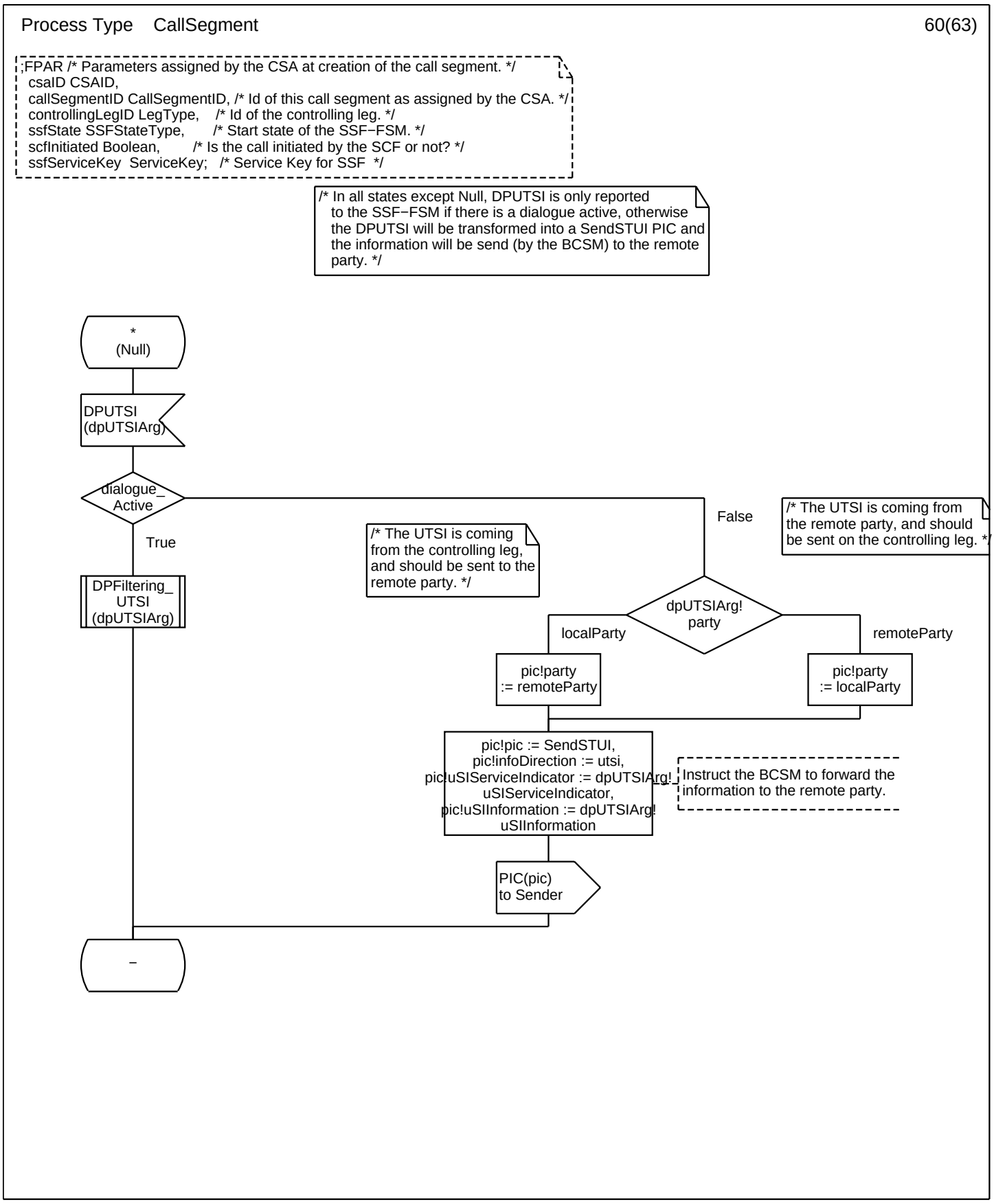


Process Type CallSegment 59(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```



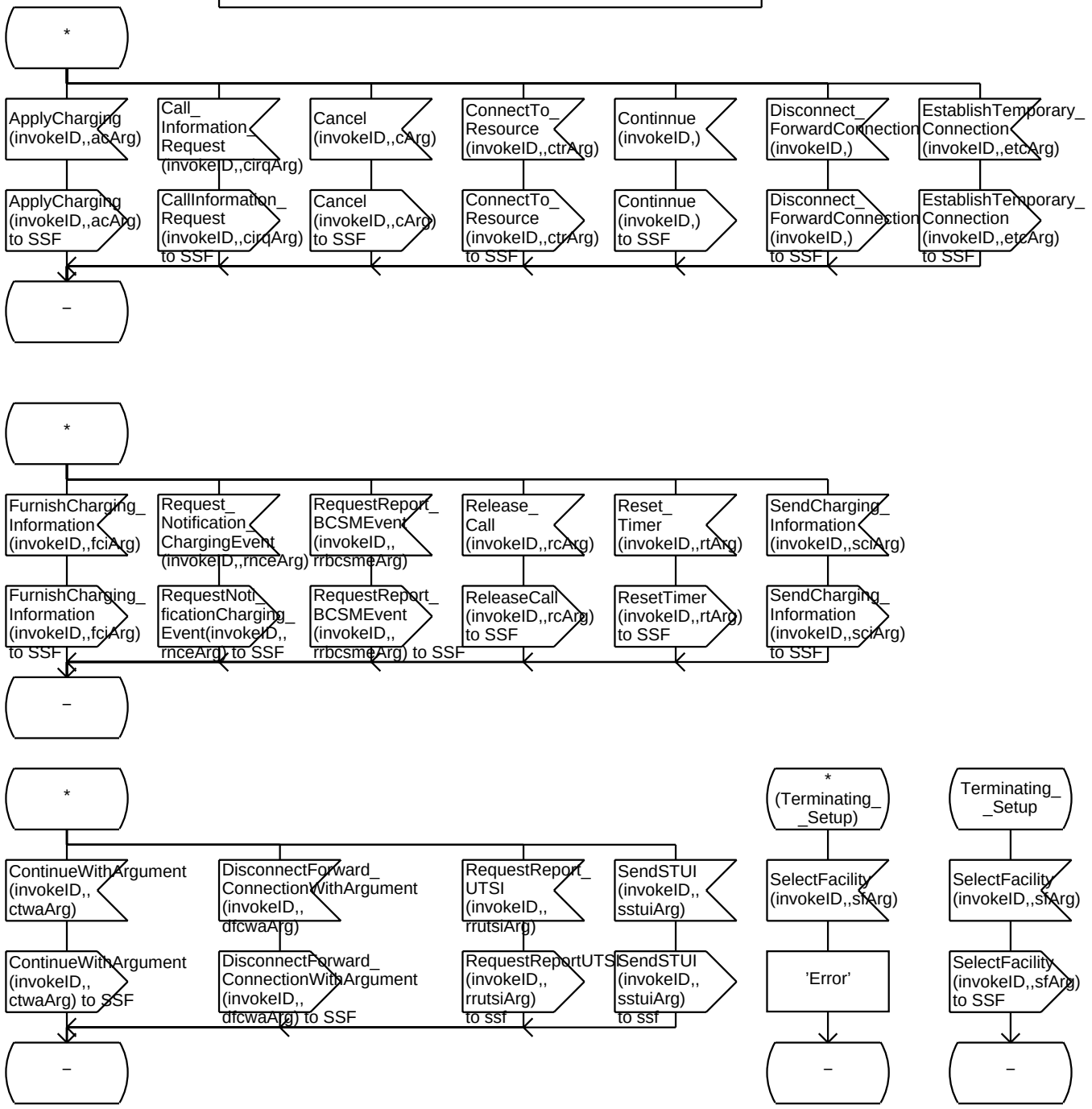
Process Type CallSegment 60(63)



Process Type CallSegment 61(63)

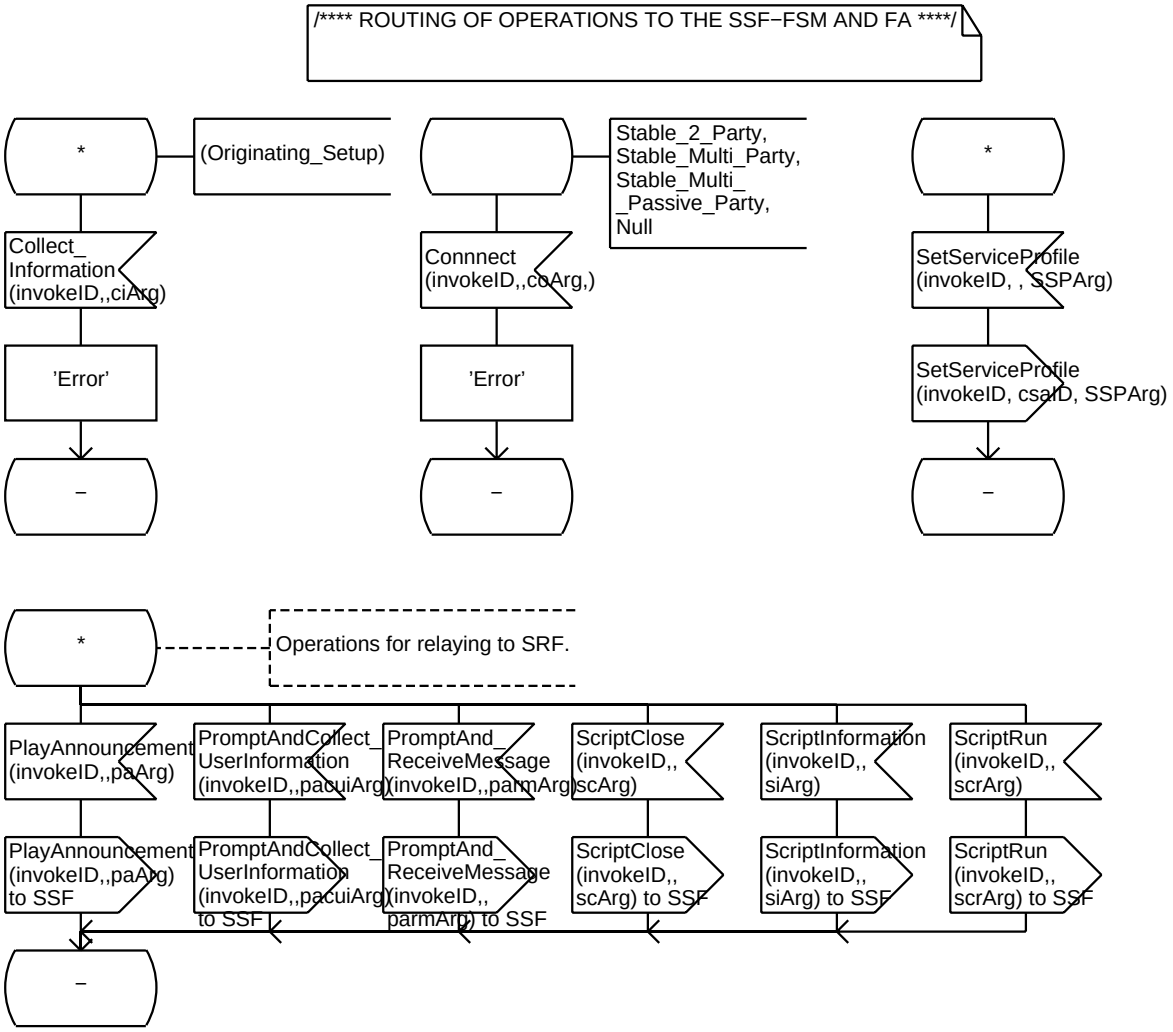
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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

```
/**** ROUTING OF OPERATIONS TO THE SSF−FSM AND FA ****/
```



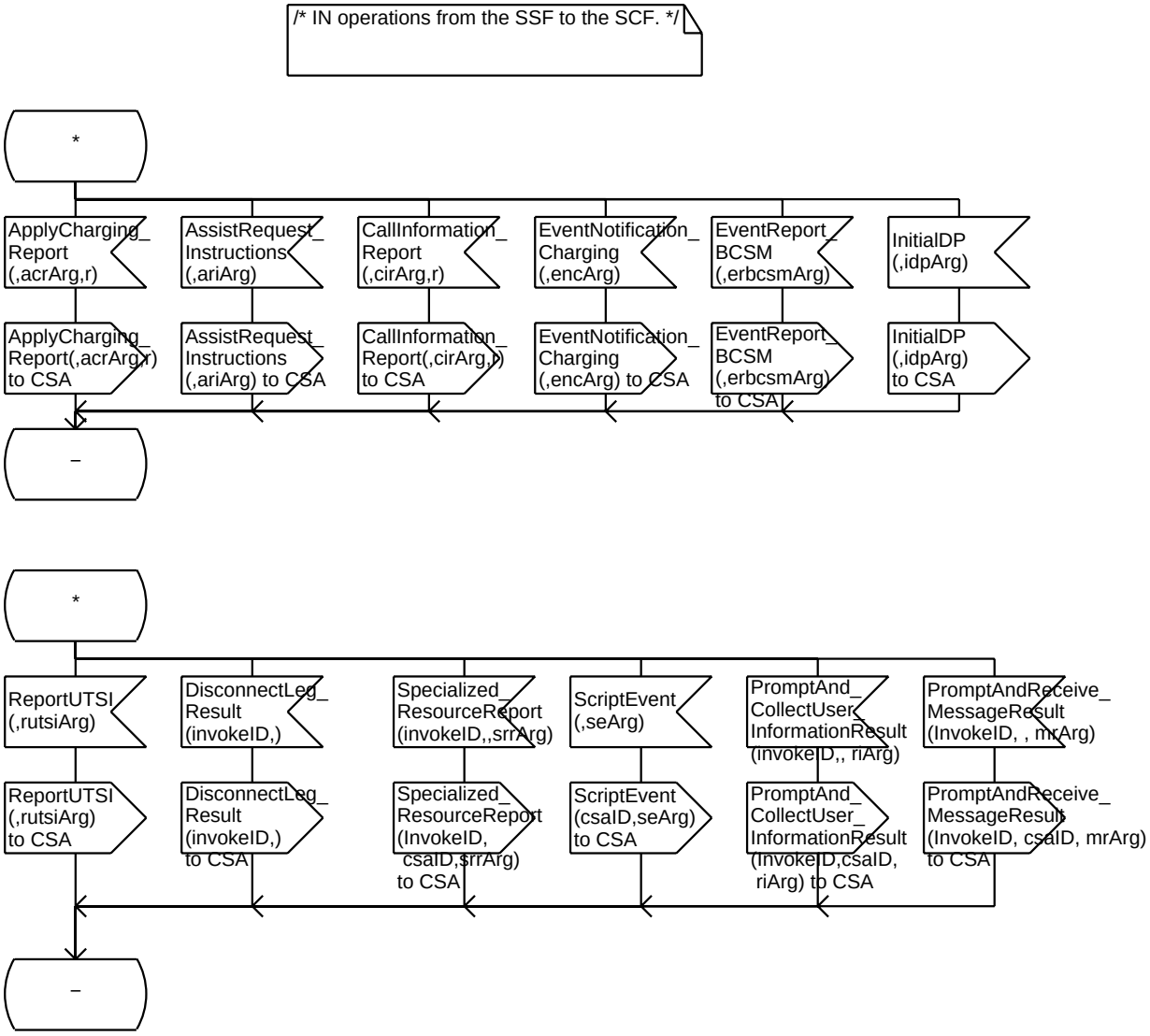
Process Type CallSegment 62(63)

```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  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? */
  ssfServiceKey  ServiceKey;   /* Service Key for SSF  */
```

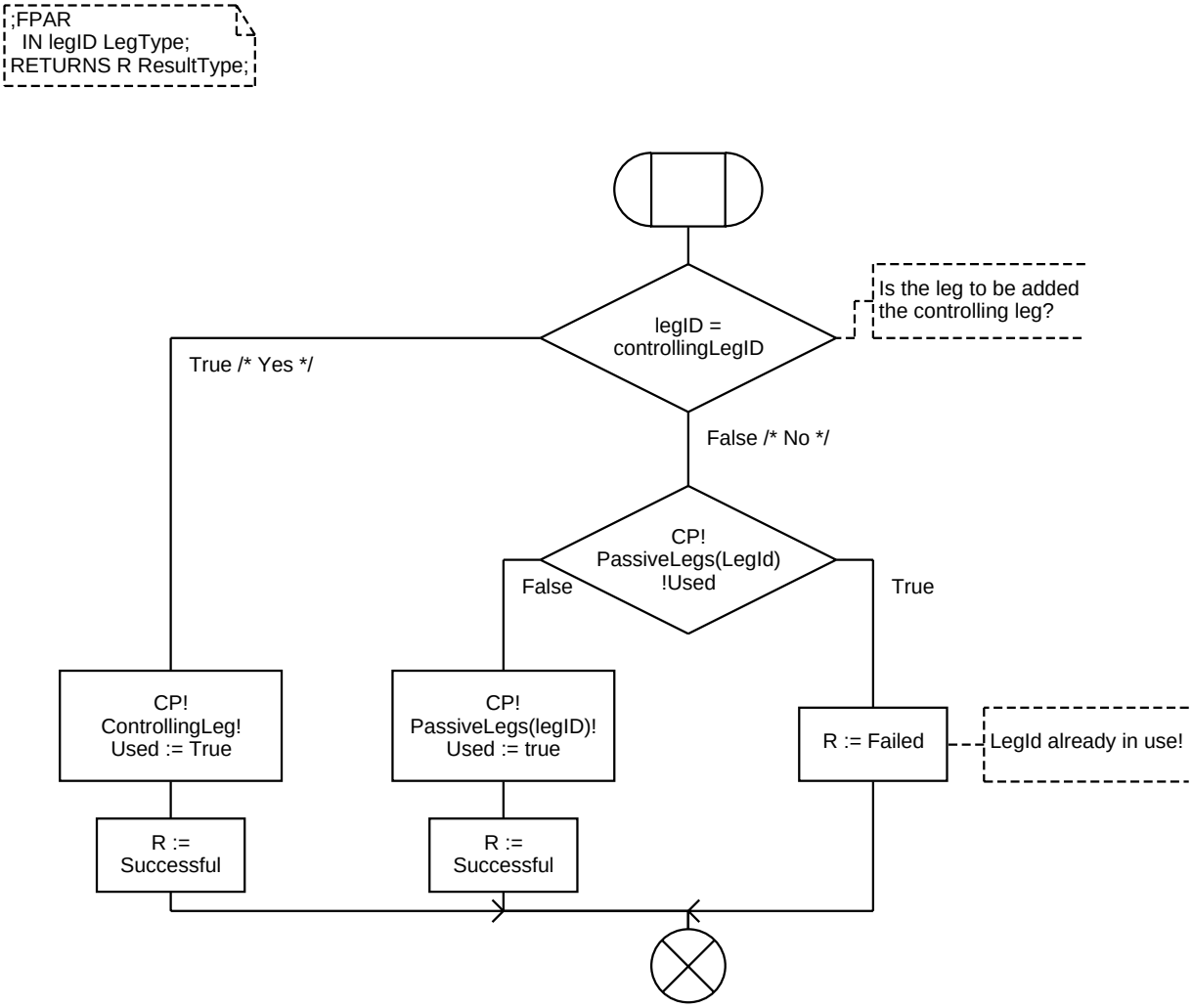


Process Type CallSegment 63(63)

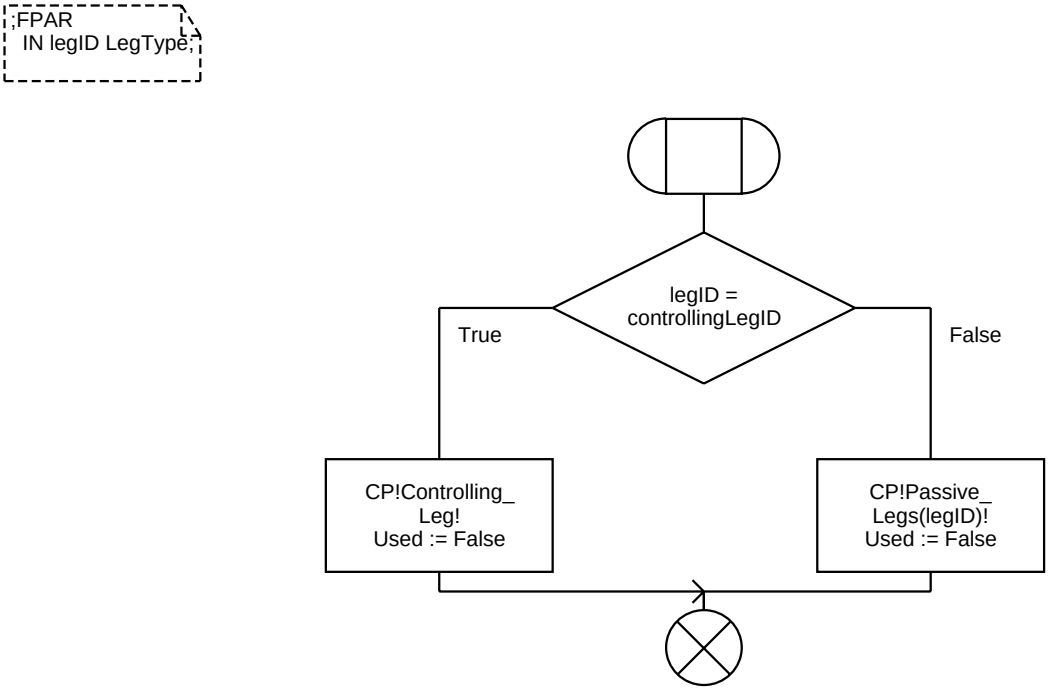
```
;FPAR /* Parameters assigned by the CSA at creation of the call segment. */
  csaID CSAID,
  callSegmentID CallSegmentID, /* Id of this call segment as assigned by the CSA. */
  controllingLegID LegType,    /* Id of the controlling leg. */
  ssfState SSFStateType,       /* Start state of the SSF−FSM.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



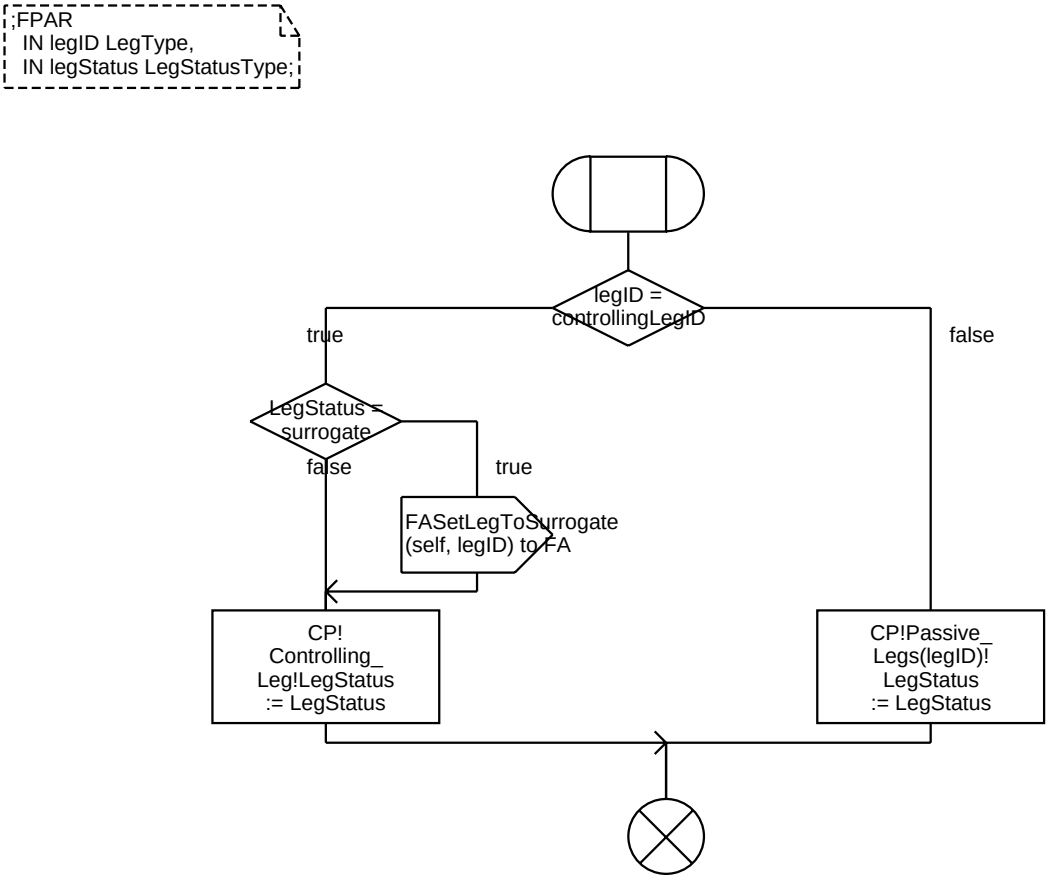
Procedure AddLeg 1(1)



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

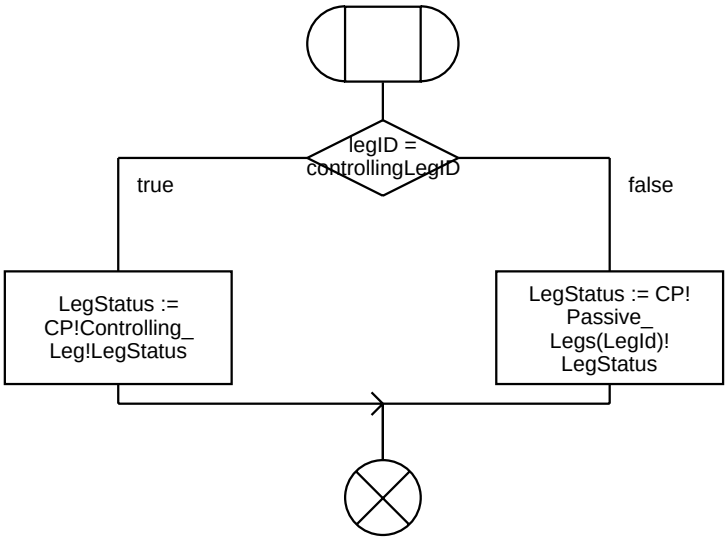


Procedure SetLegStatus 1(1)

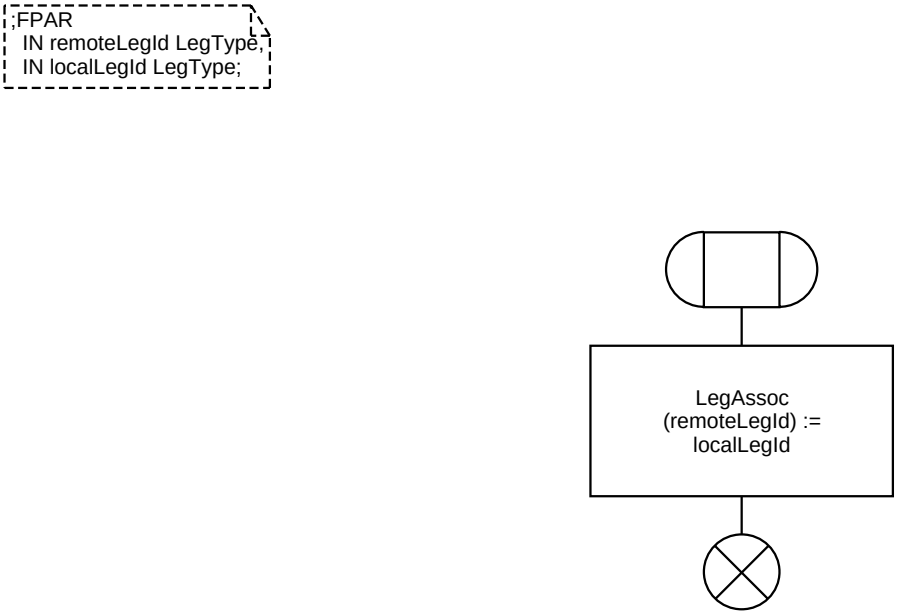


Procedure GetLegStatus 1(1)

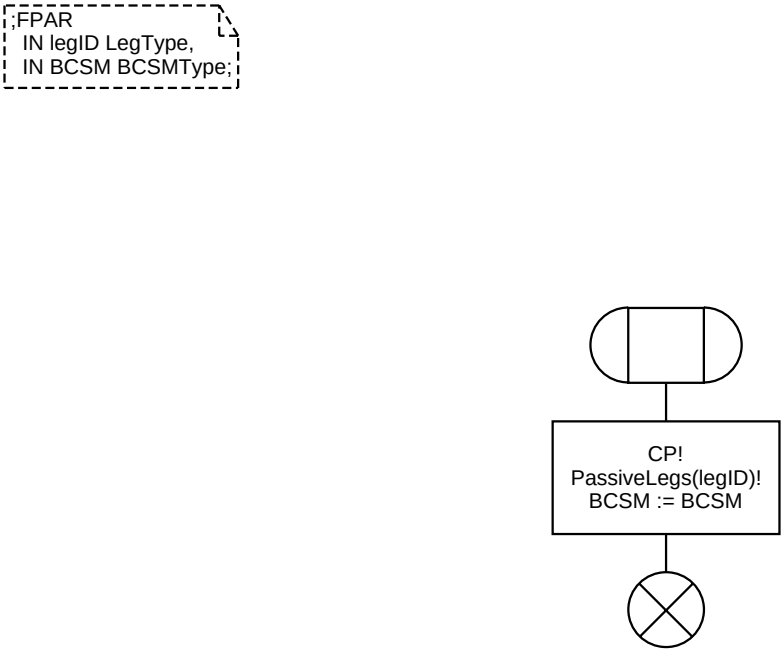
```
;FPAR
  IN legID LegType;
RETURNS LegStatus LegStatusType;
```



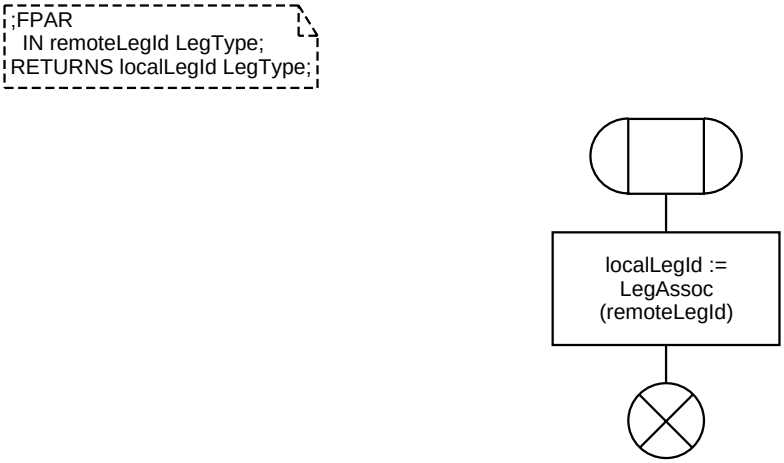
Procedure <<Process Type CallSegment>> SetLegAssoc 1(1)



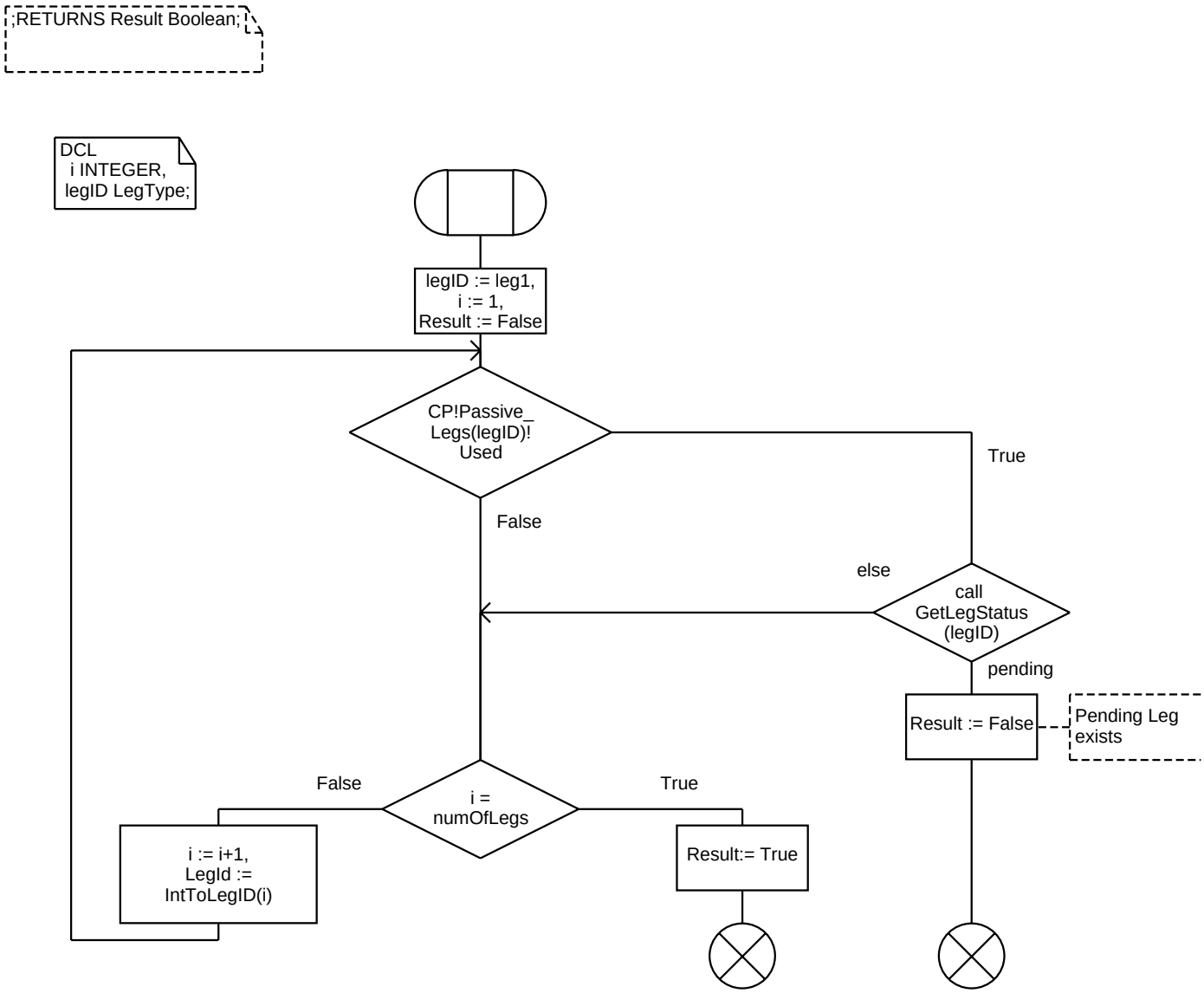
Procedure SetLegPtr 1(1)



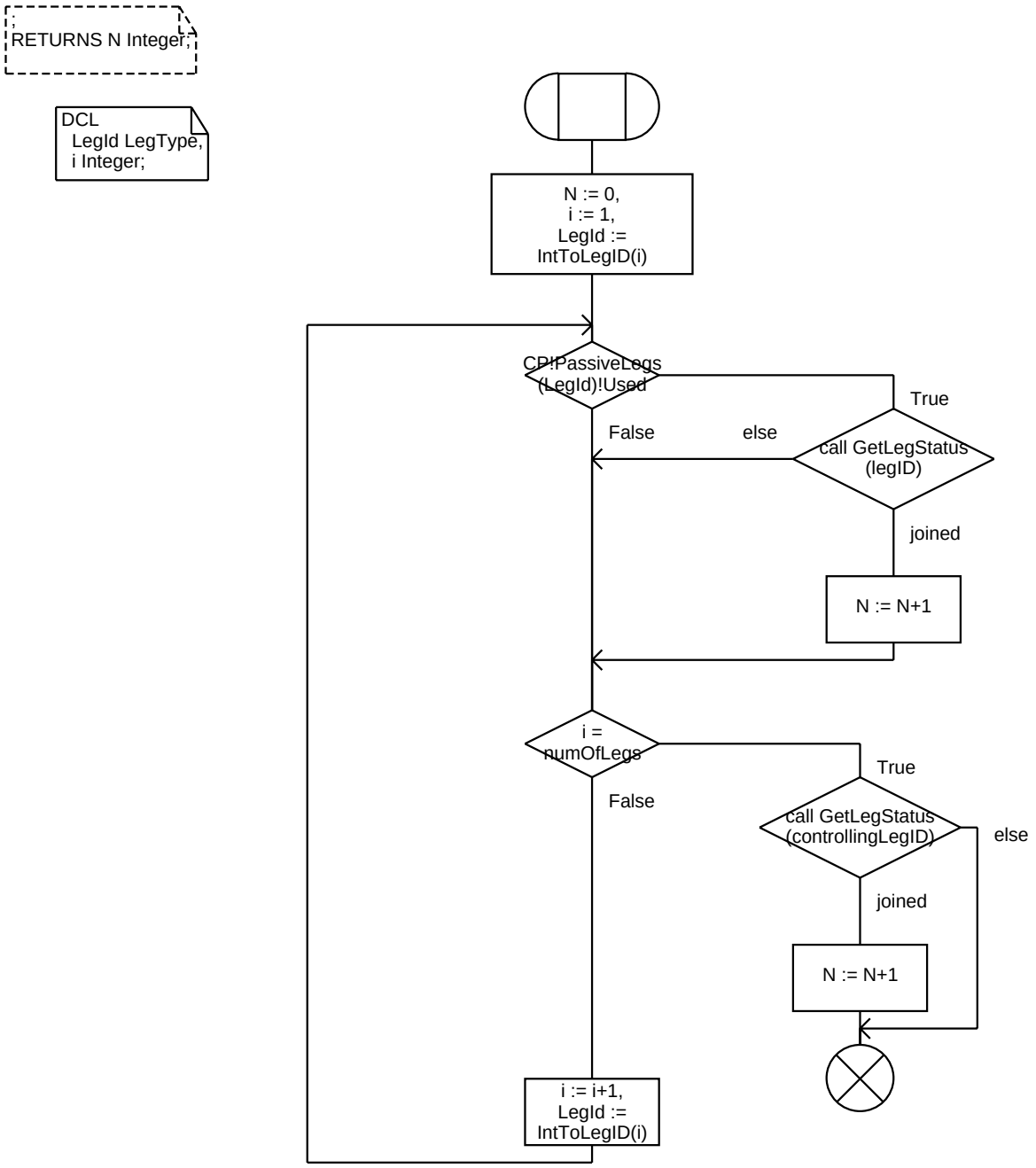
Procedure <<Process Type CallSegment>> GetLegIdfromRemoteLegId 1(1)



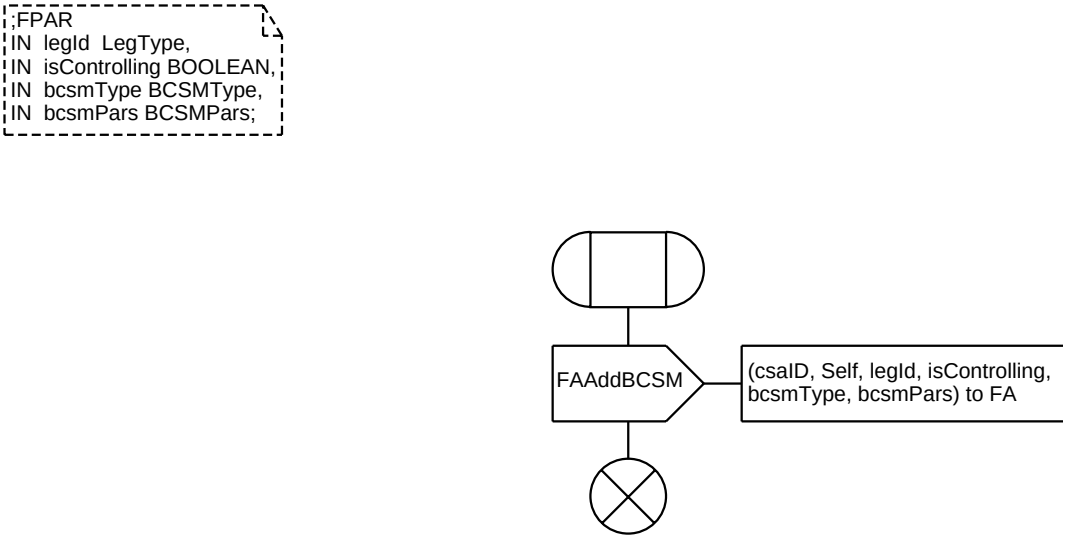
Procedure CheckNoPendingLegs 1(1)



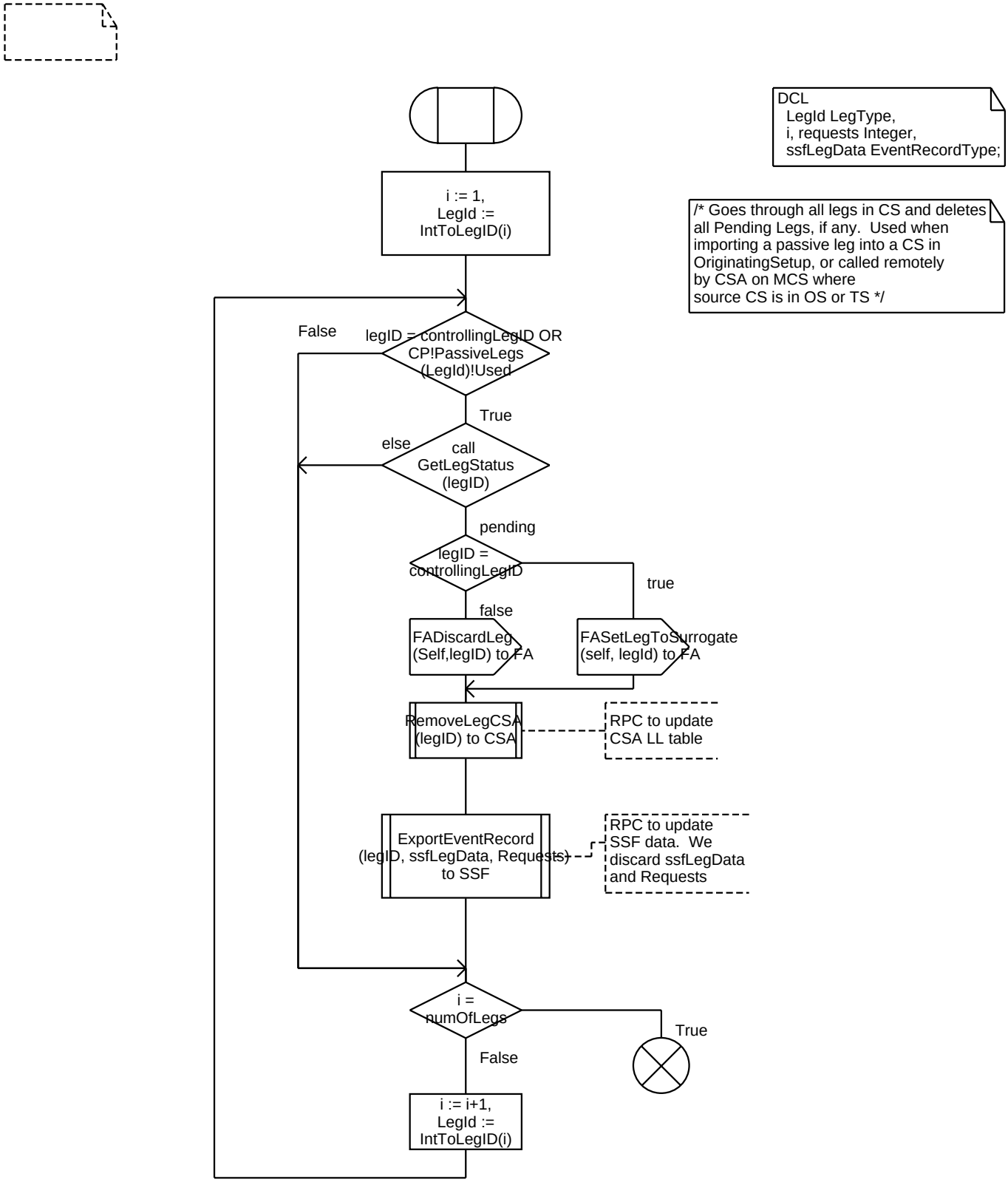
Procedure NoOfJoinedLegs 1(1)



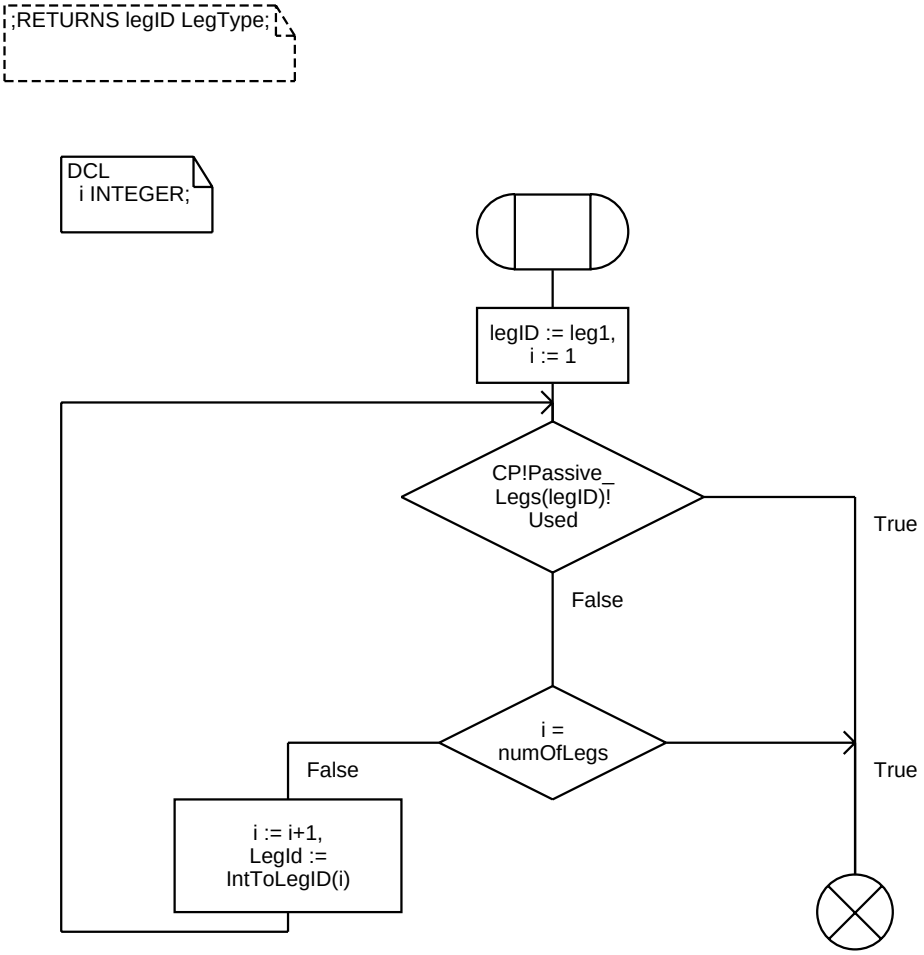
Procedure AddBCSM 1(1)



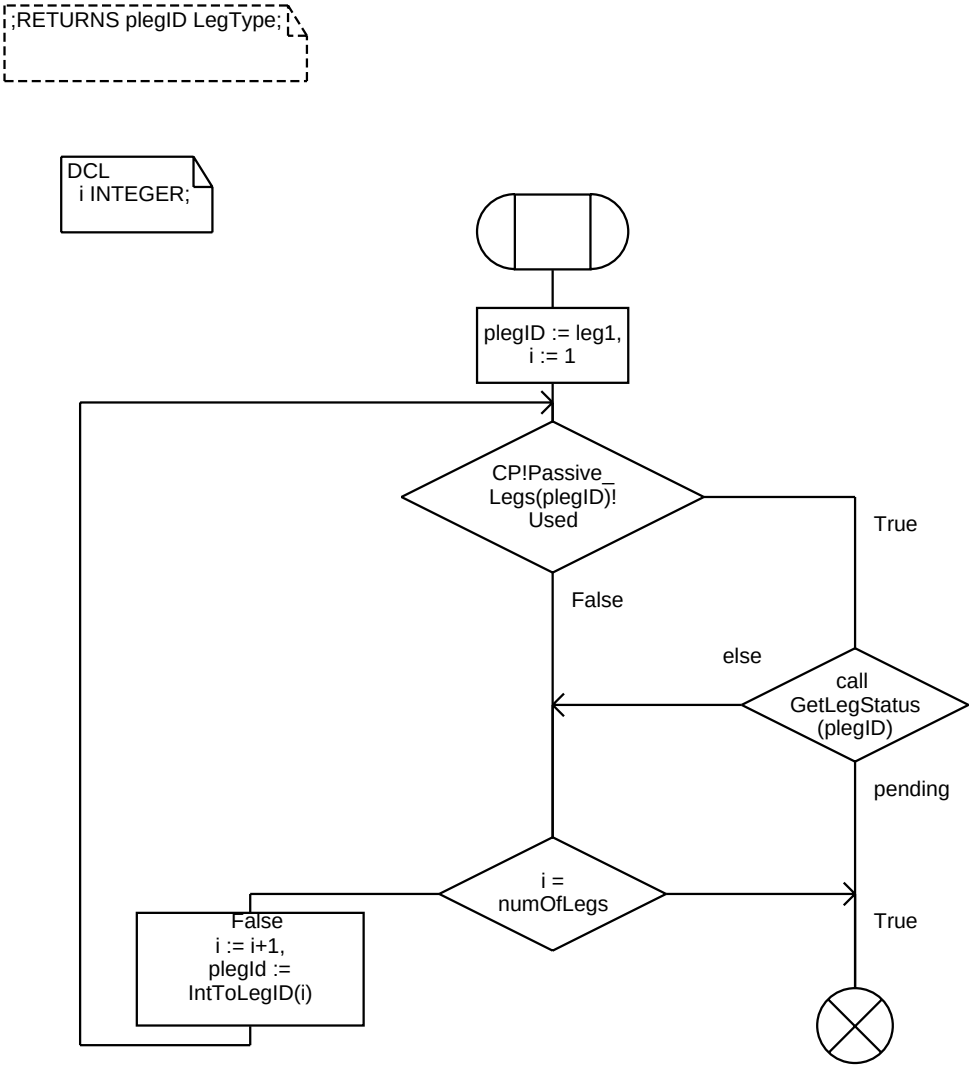
Procedure DeletePendingLeg 1(1)



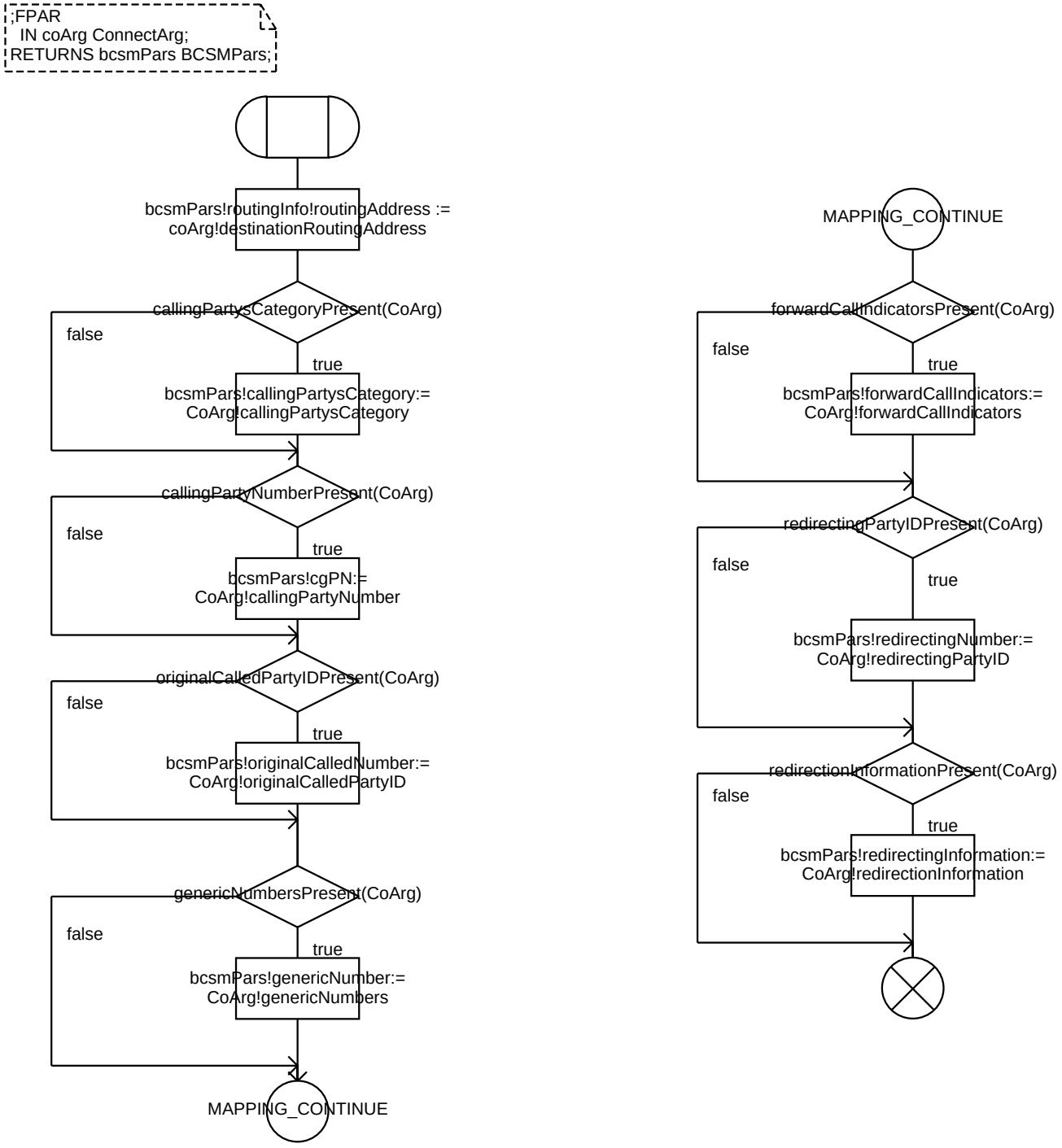
Procedure GetPassiveLegId 1(1)



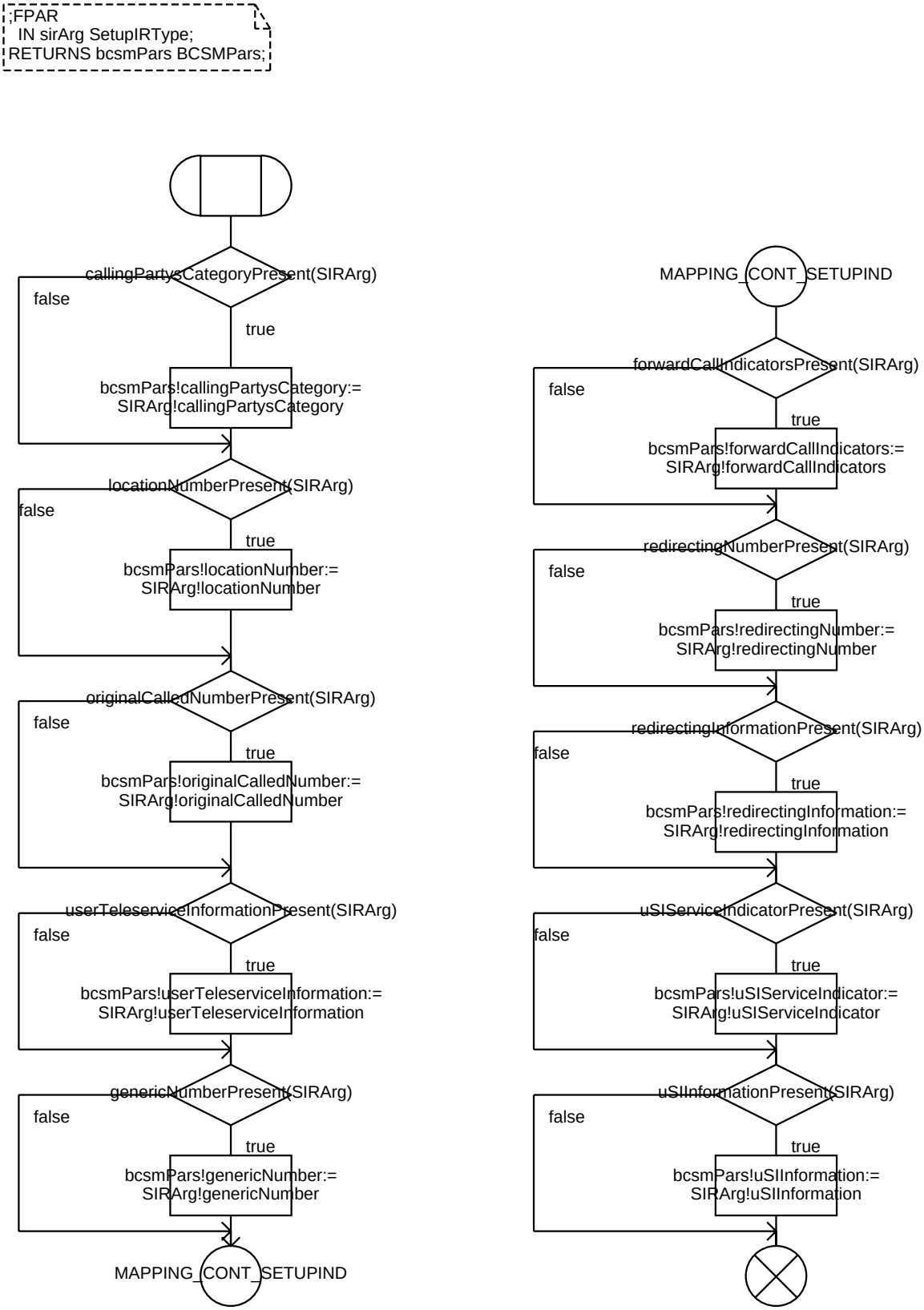
Procedure GetPendingLegID 1(1)



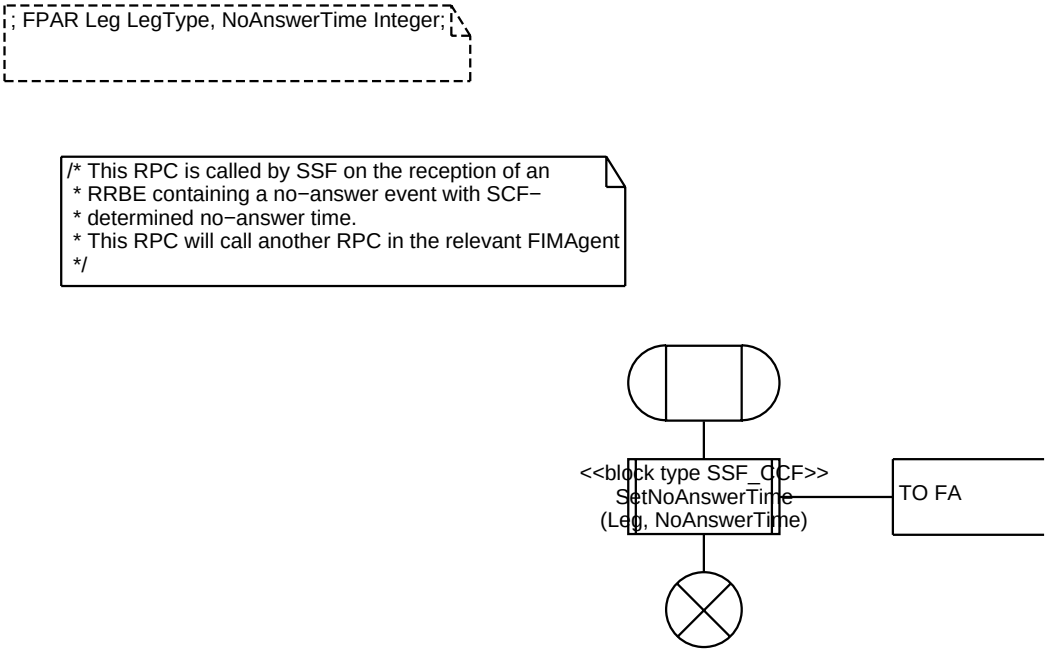
Procedure MapConnectToBCSM 1(1)



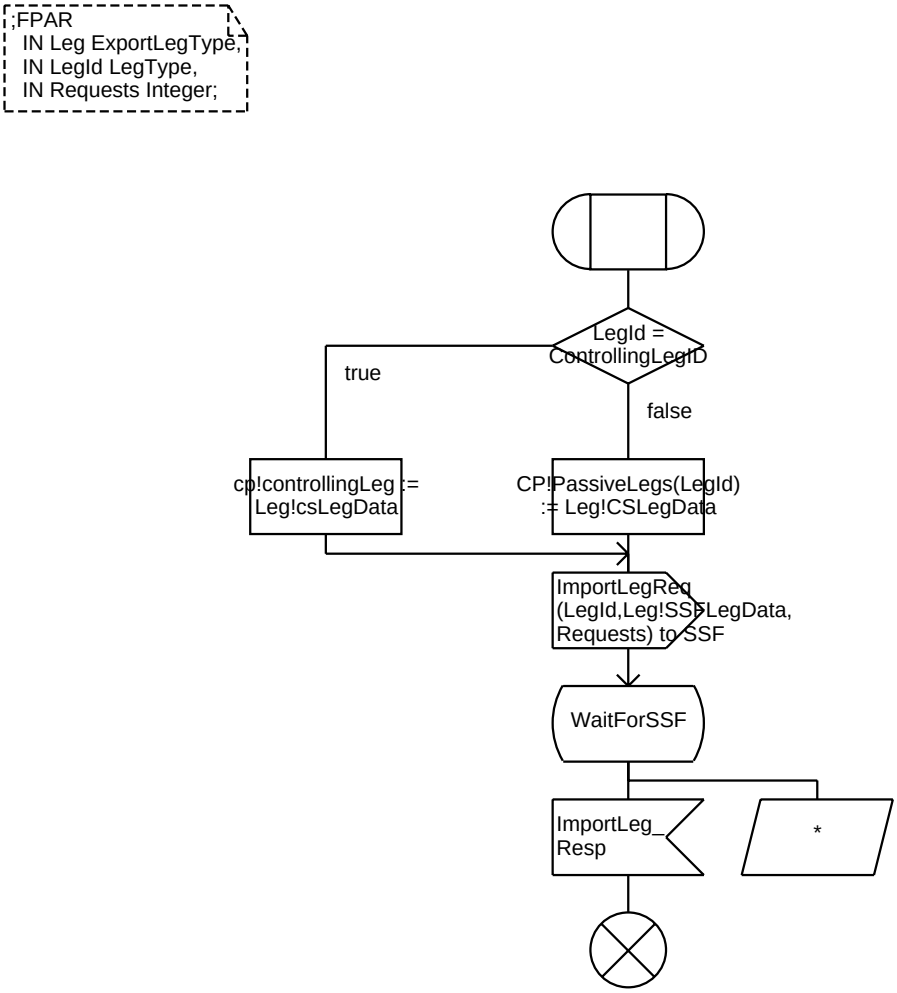
Procedure <<Process Type CallSegment>> MapSIToBCSM 1(1)



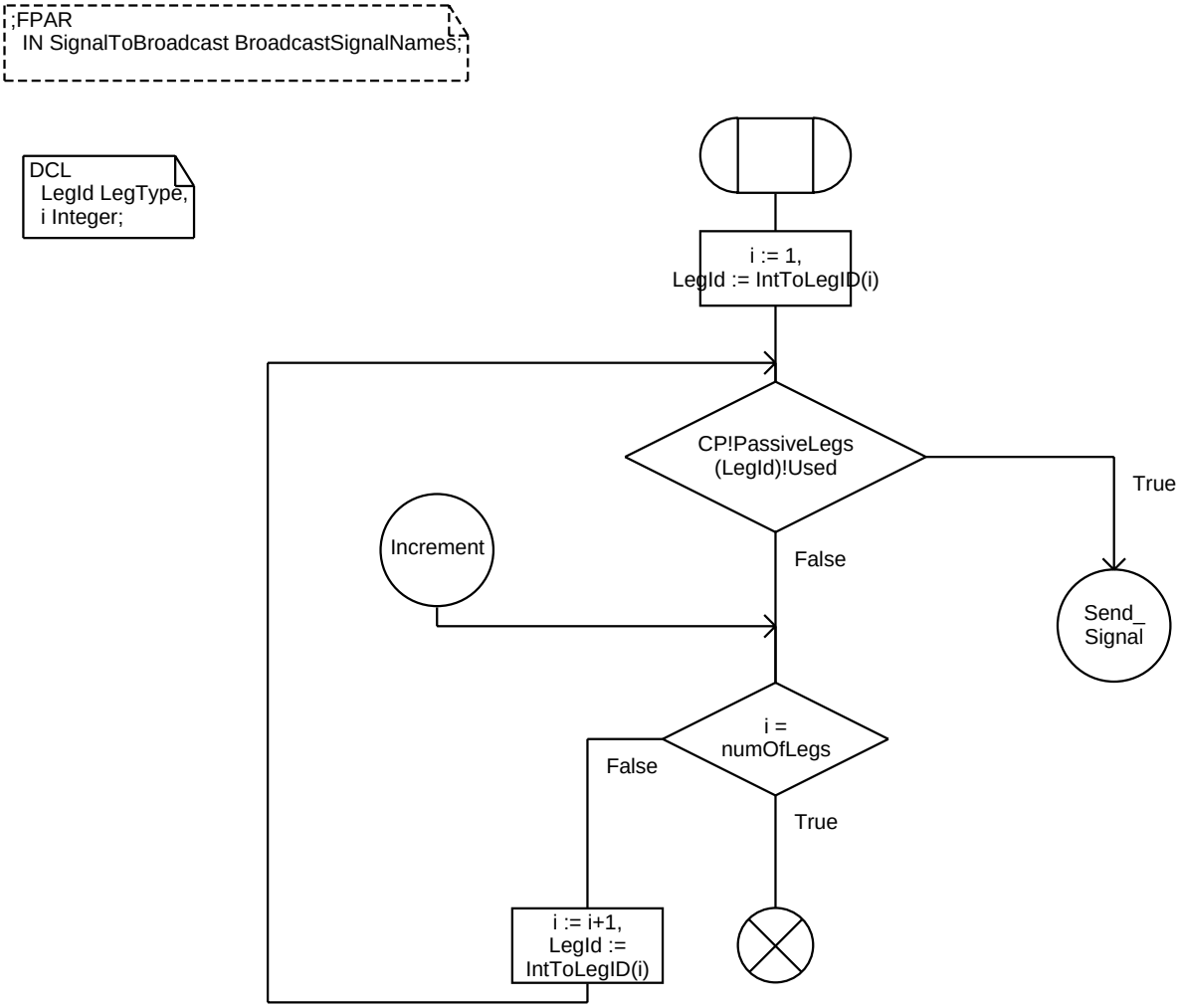
Procedure <<Process Type CallSegment>> SetNoAnswerTime 1(1)



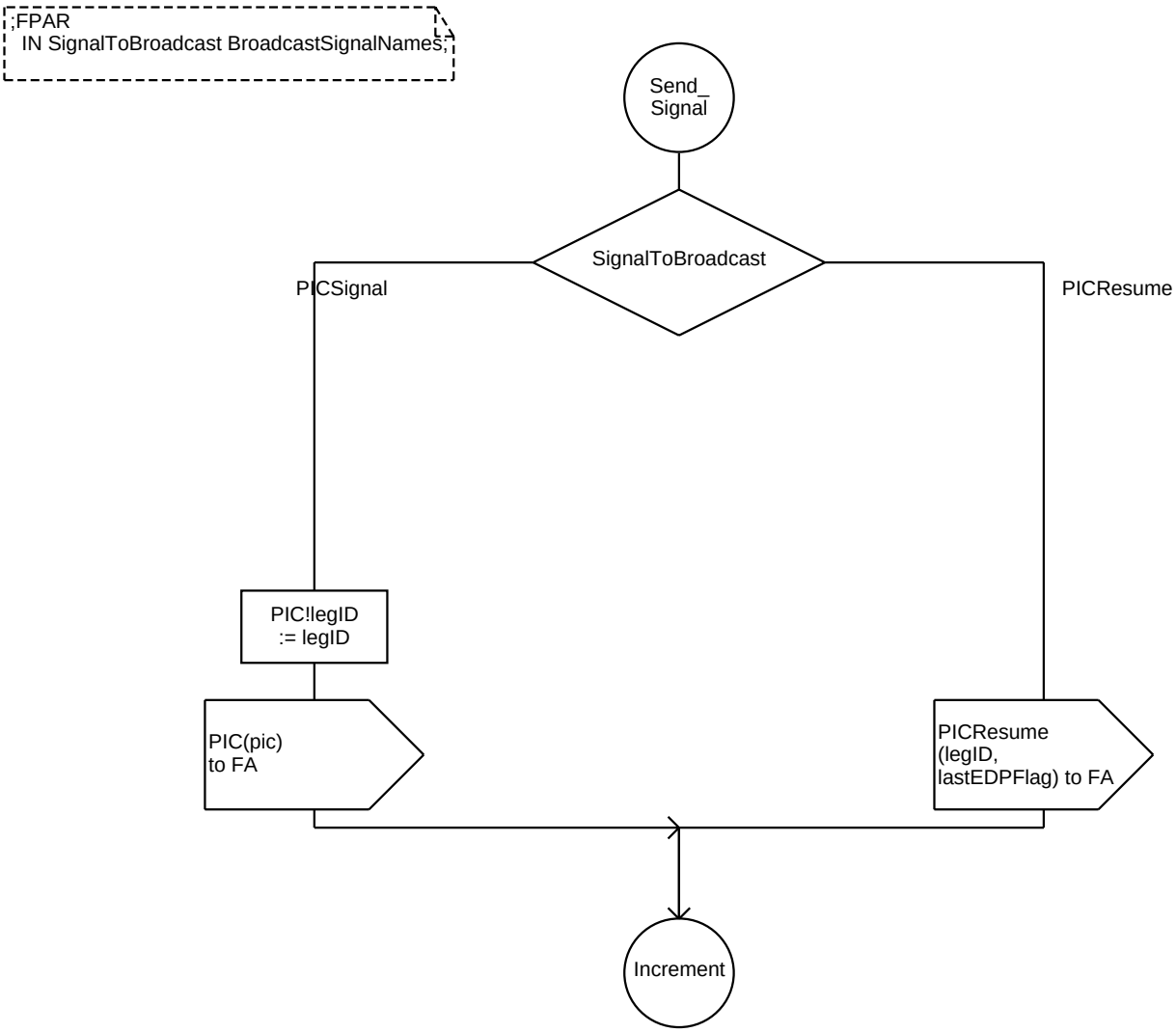
Procedure <<Process Type CallSegment>> ImportLeg 1(1)



Procedure BroadcastToLegs 1(2)

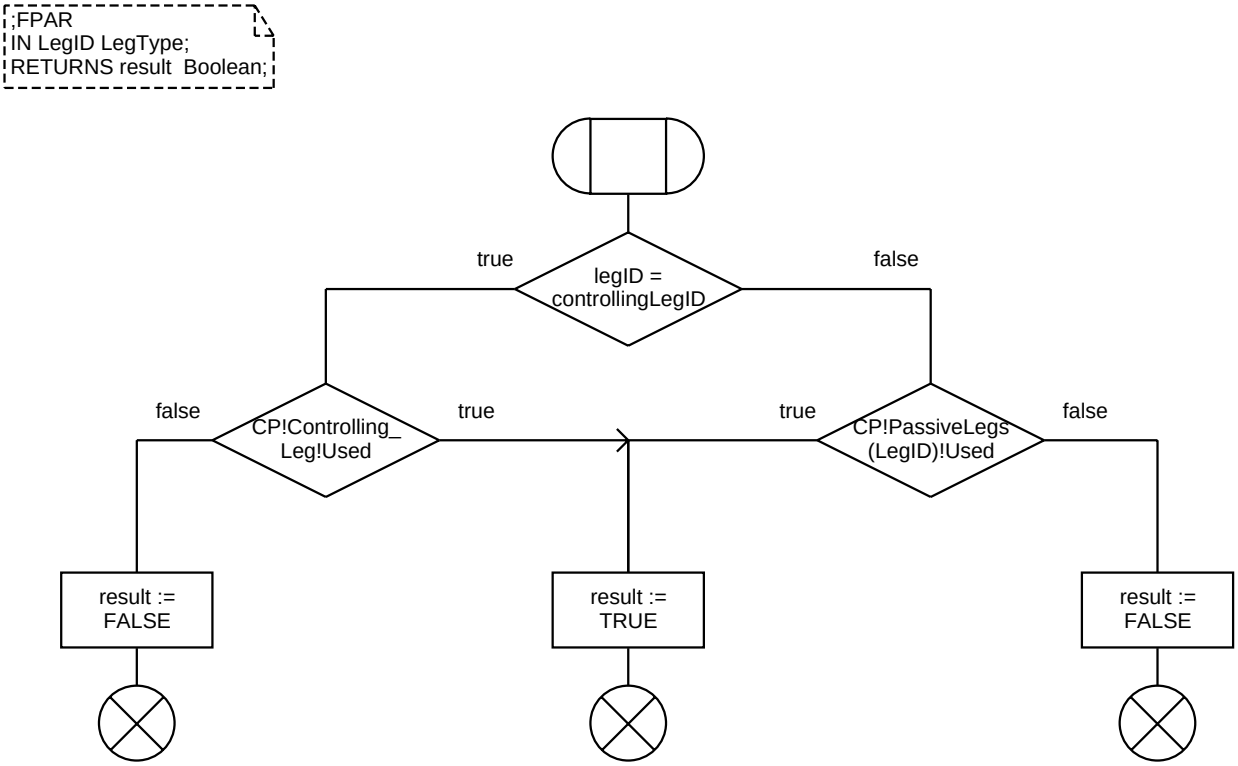


Procedure BroadcastToLegs 2(2)

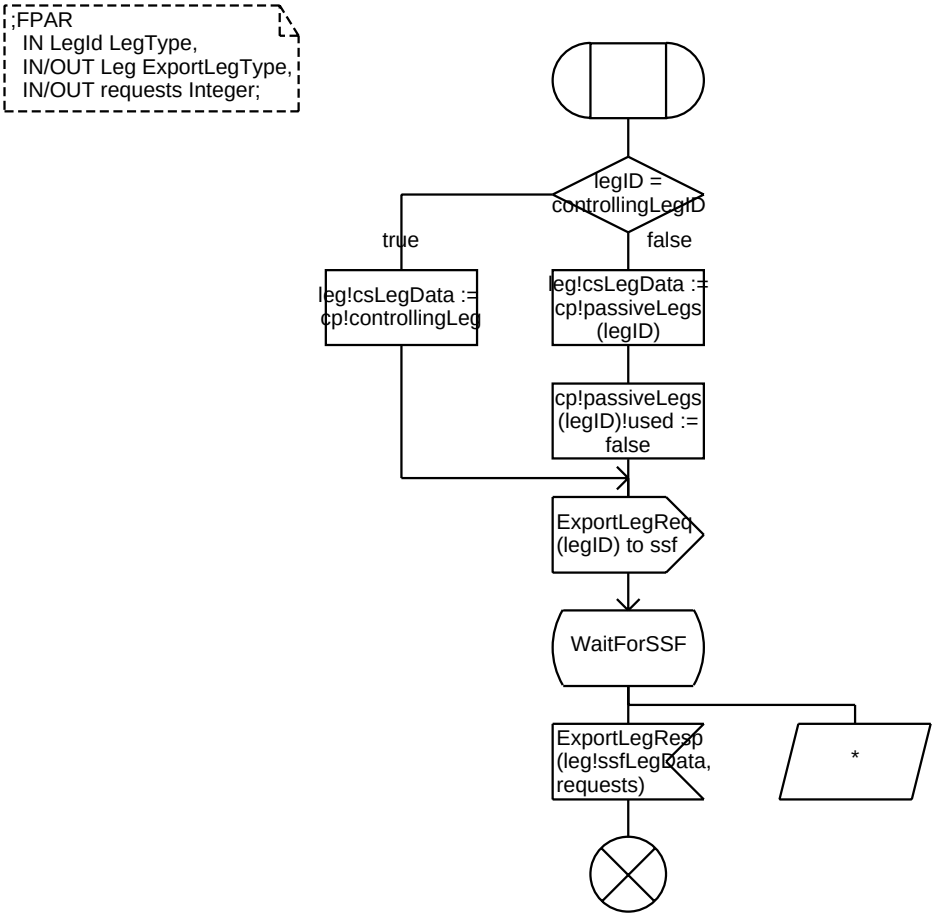


ExistLeg

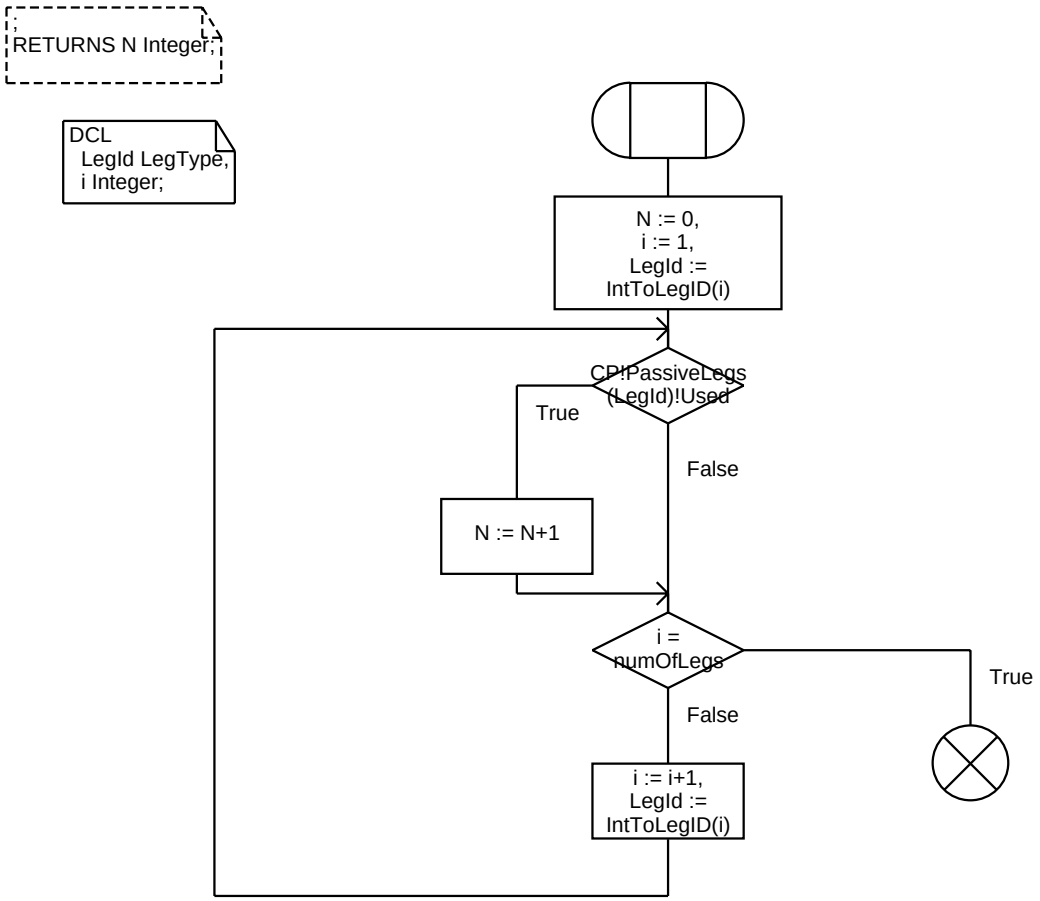
Procedure <<Process Type CallSegment>> ExistLeg 1(1)



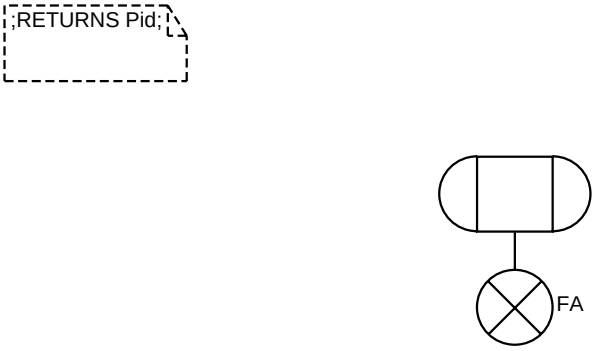
Procedure <<Process Type CallSegment>> ExportLeg 1(1)



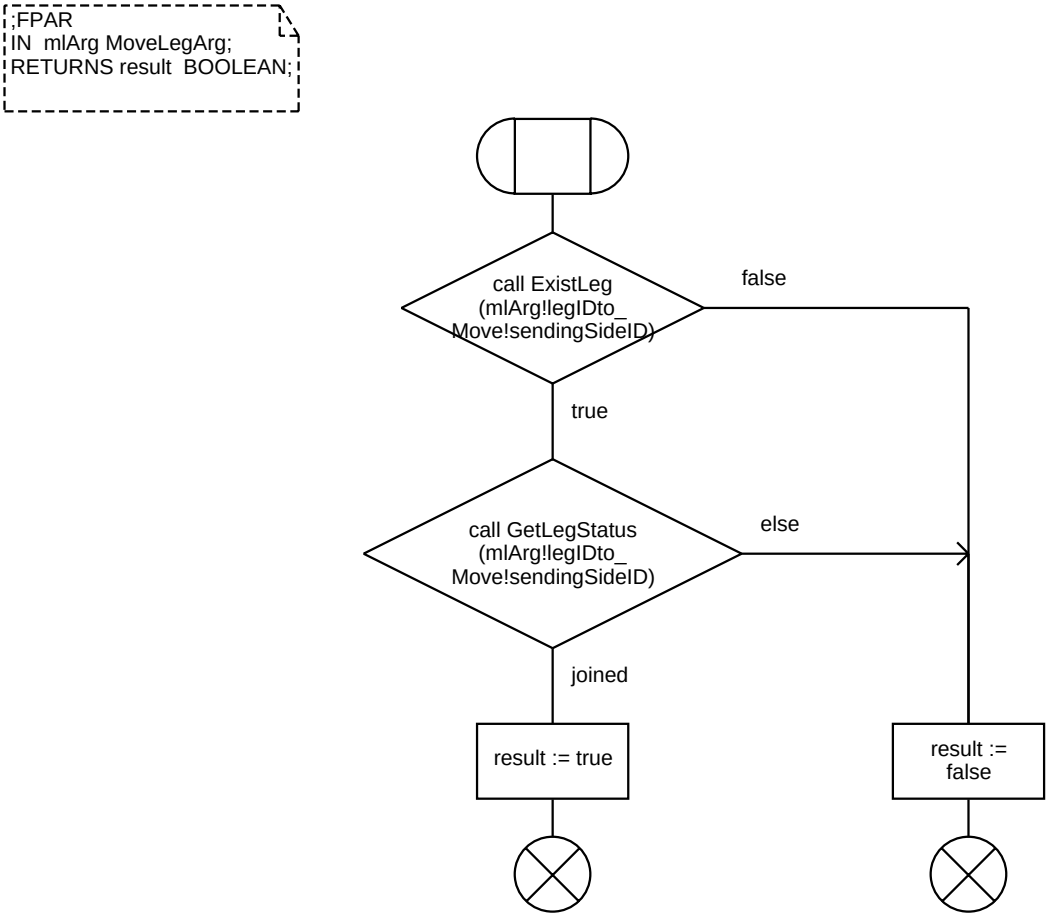
Procedure NoOfLegs 1(1)



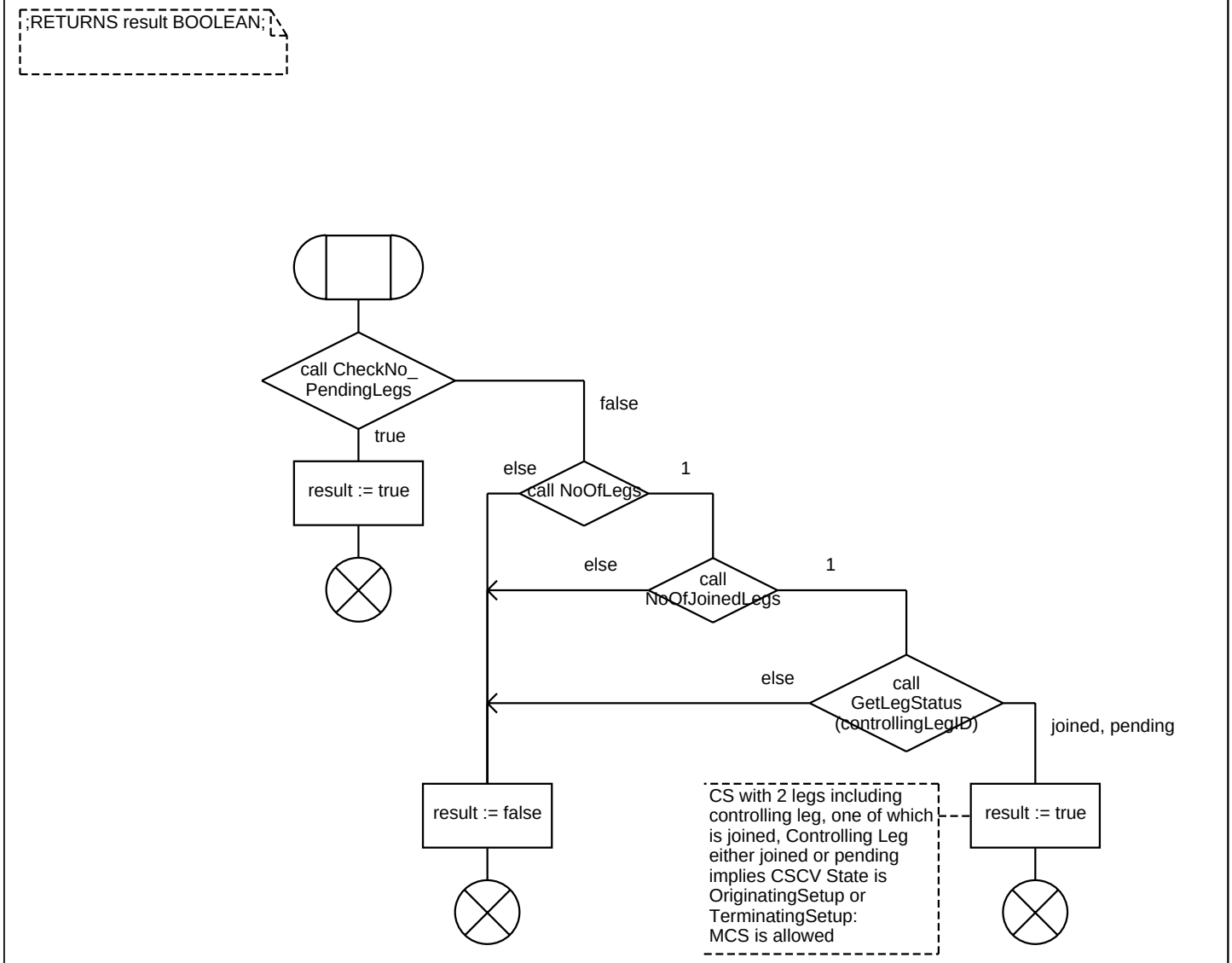
Procedure GetFAPtr 1(1)



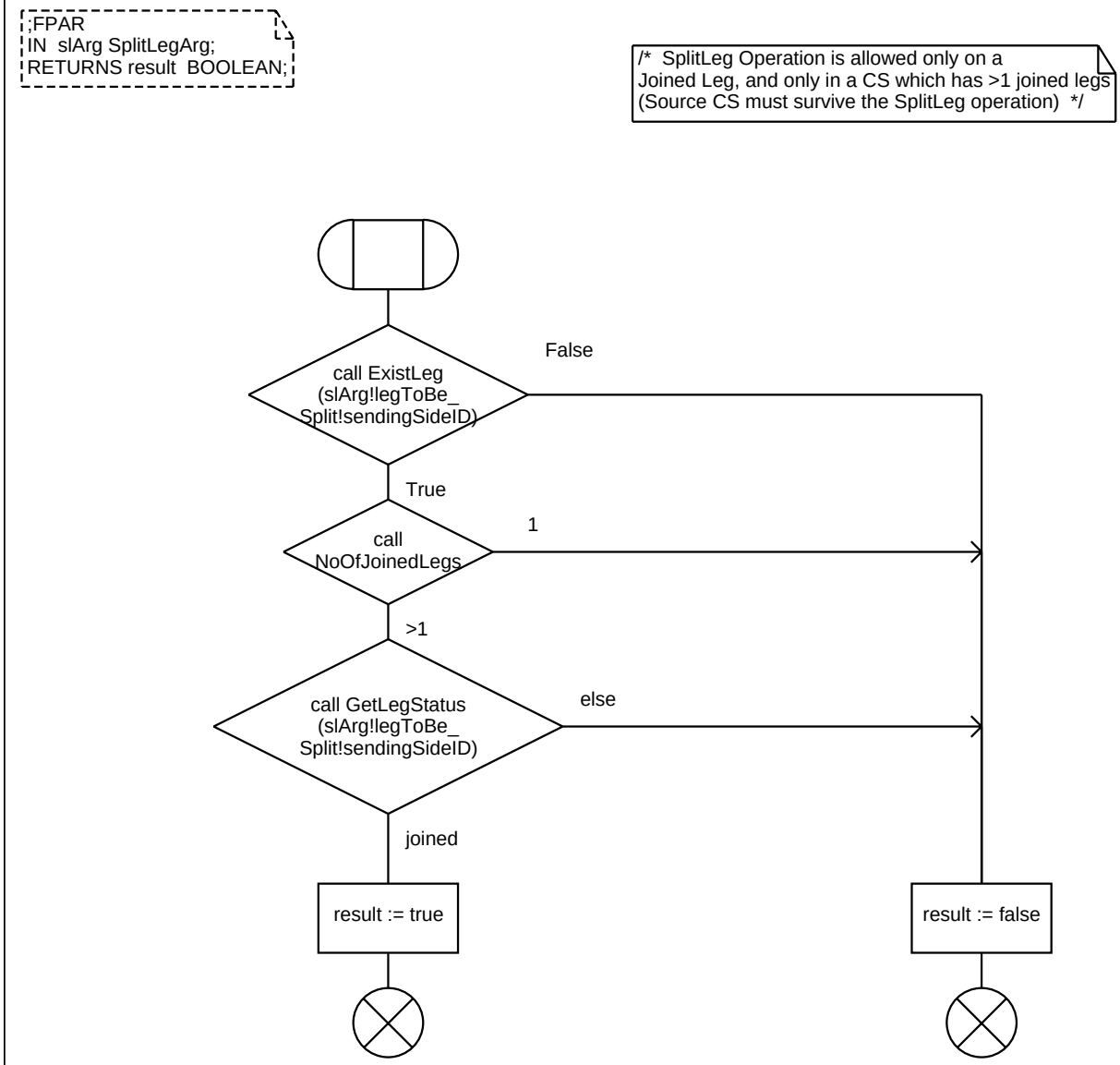
Procedure ValidSourceCSforML 1(1)



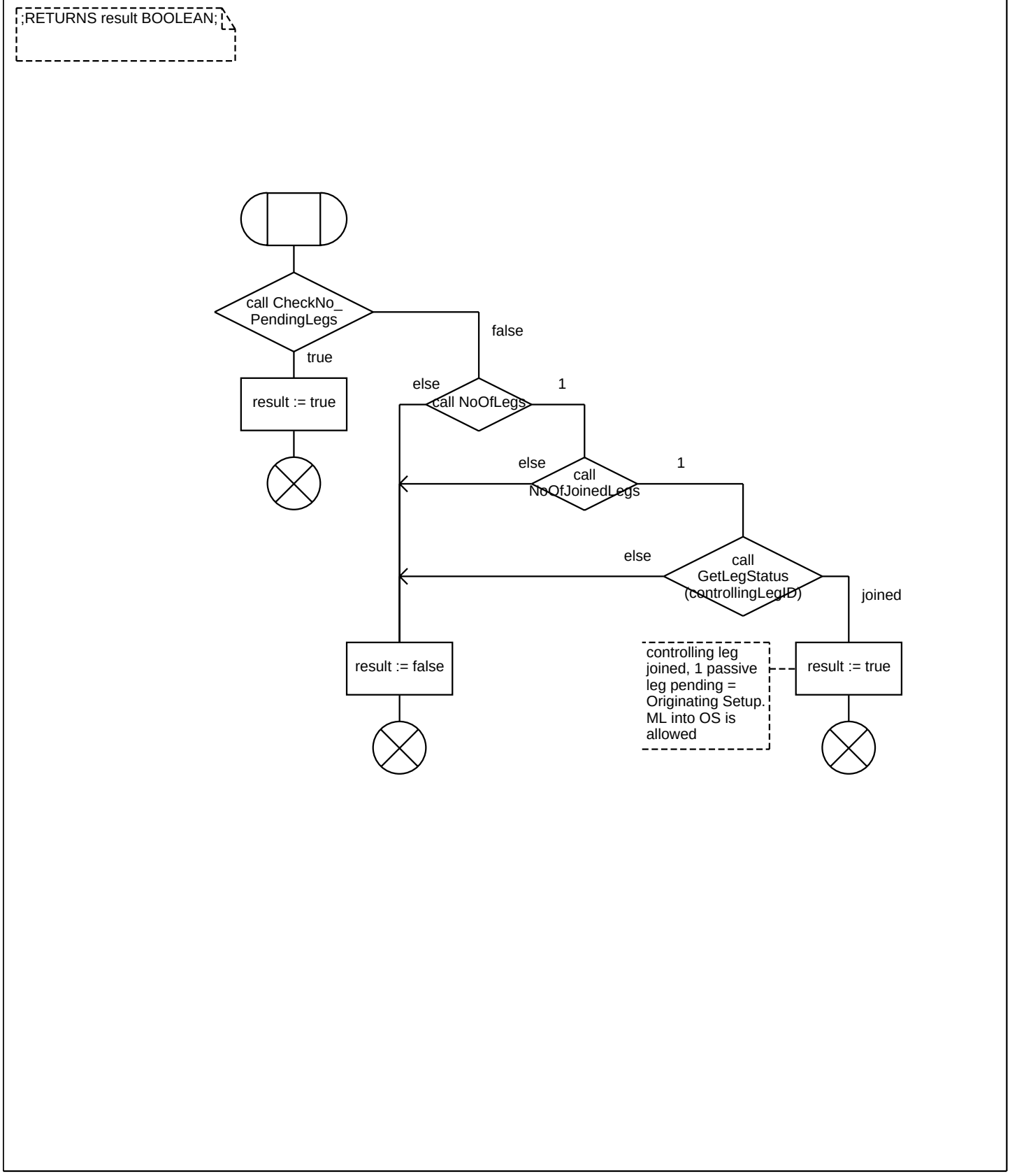
Procedure ValidSourceCSforMCS 1(1)



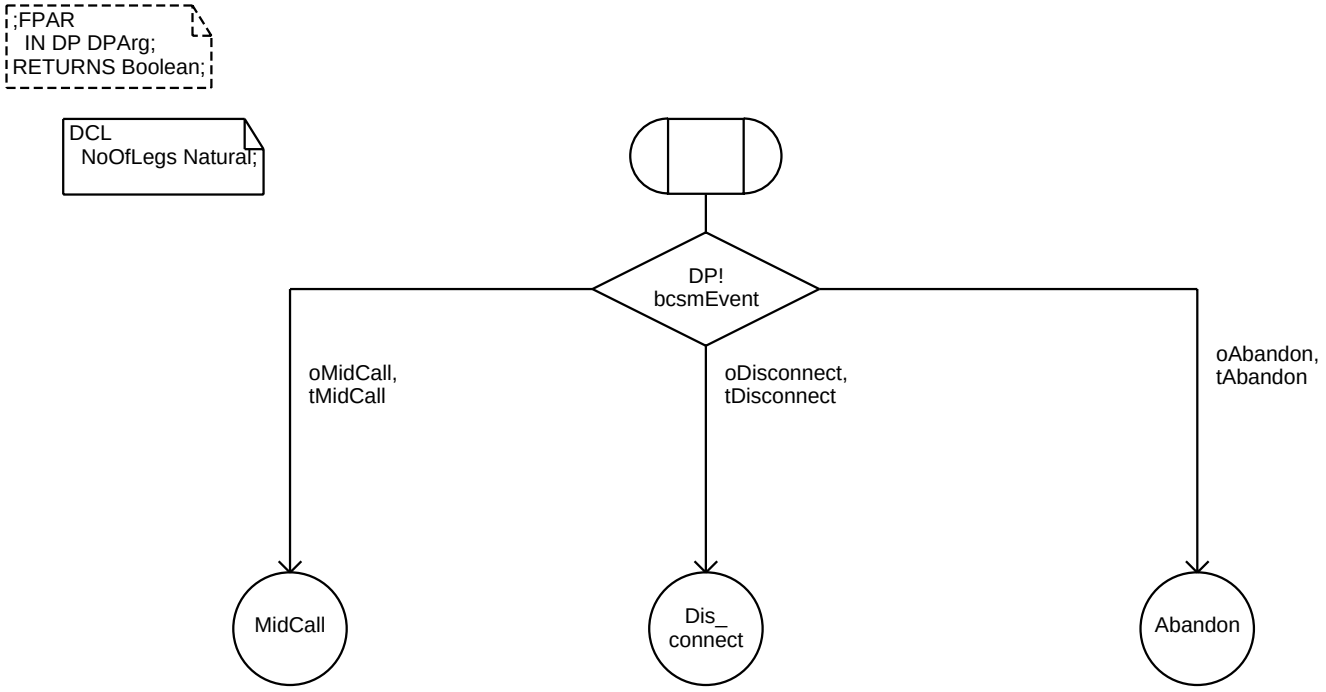
Procedure ValidSourceCSforSL 1(1)



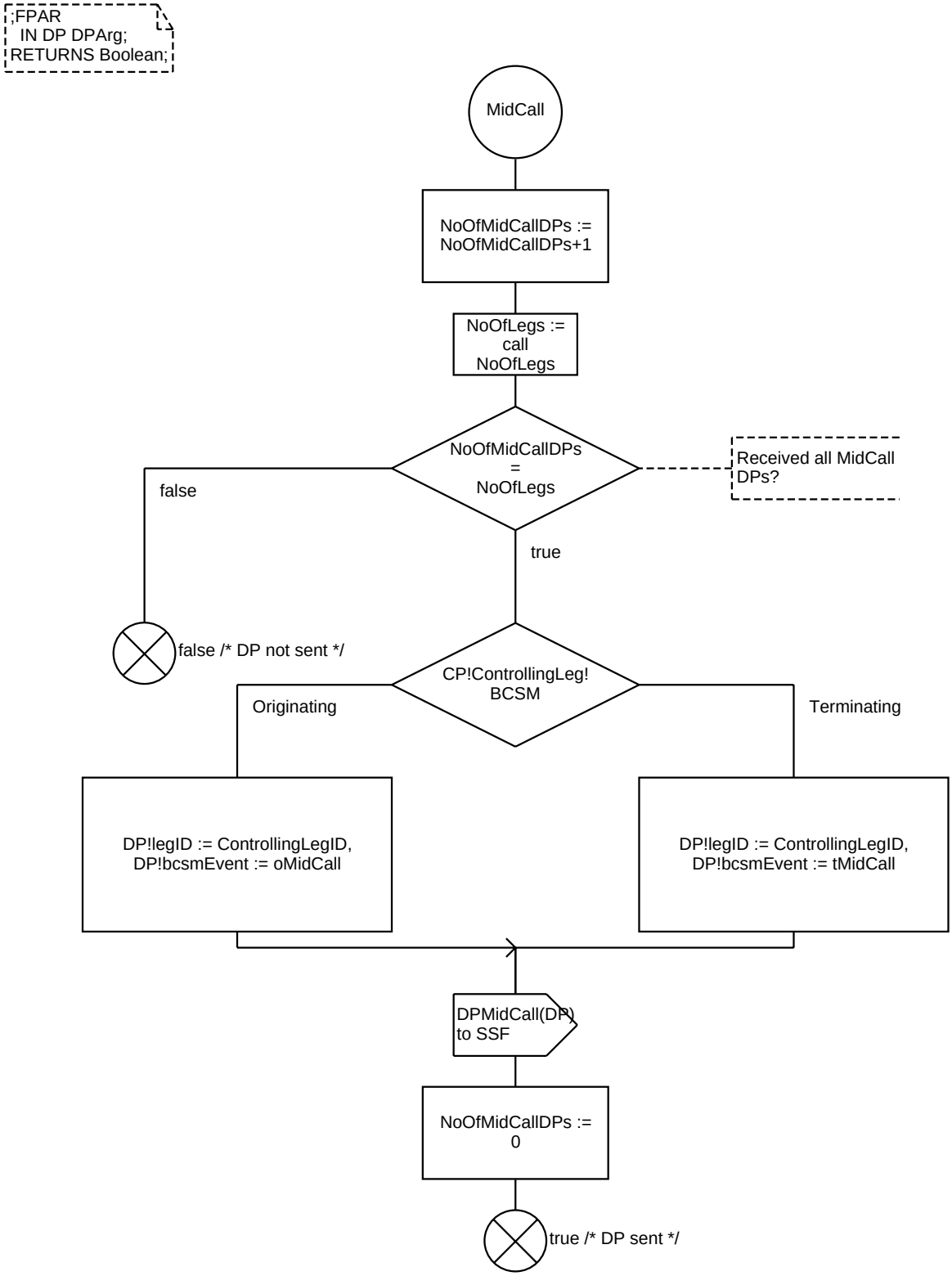
Procedure ValidTargetCSforMLMCS 1(1)



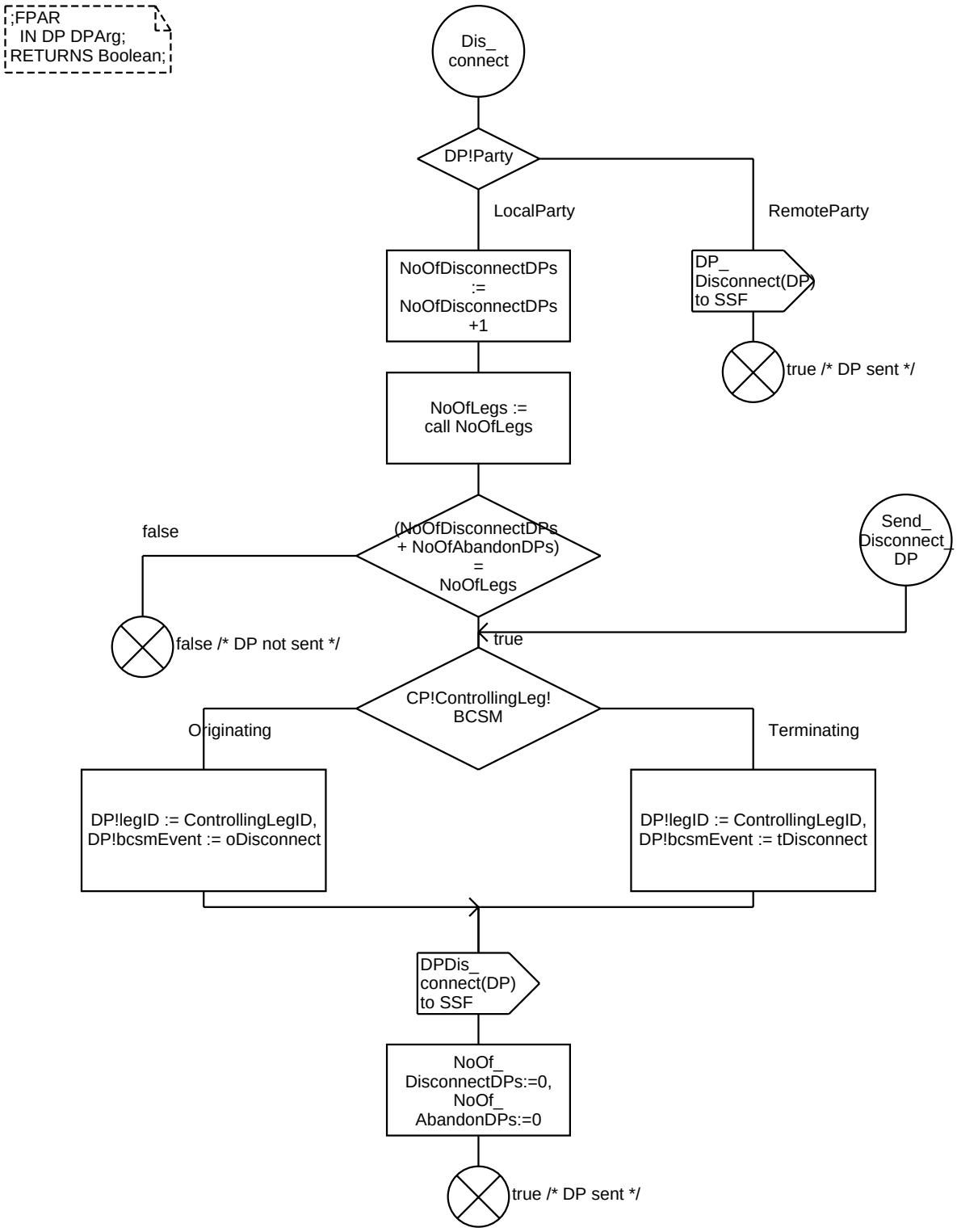
Procedure DPFiltering 1(4)



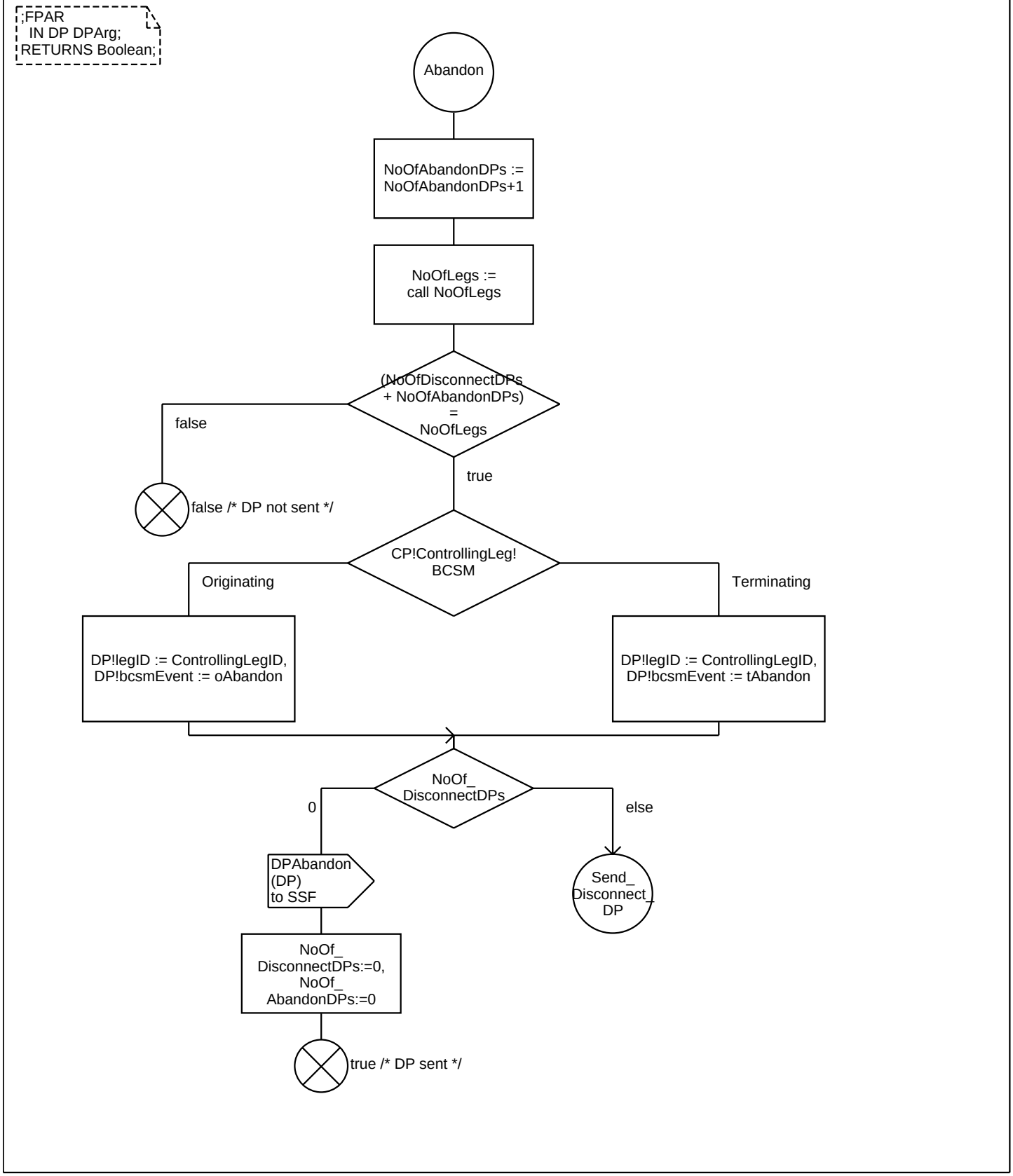
Procedure DPFiltering 2(4)



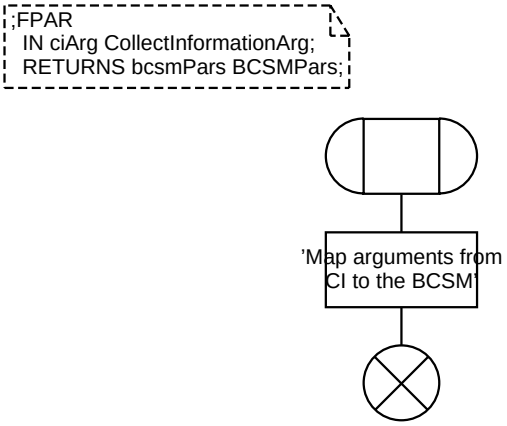
Procedure DPFiltering 3(4)



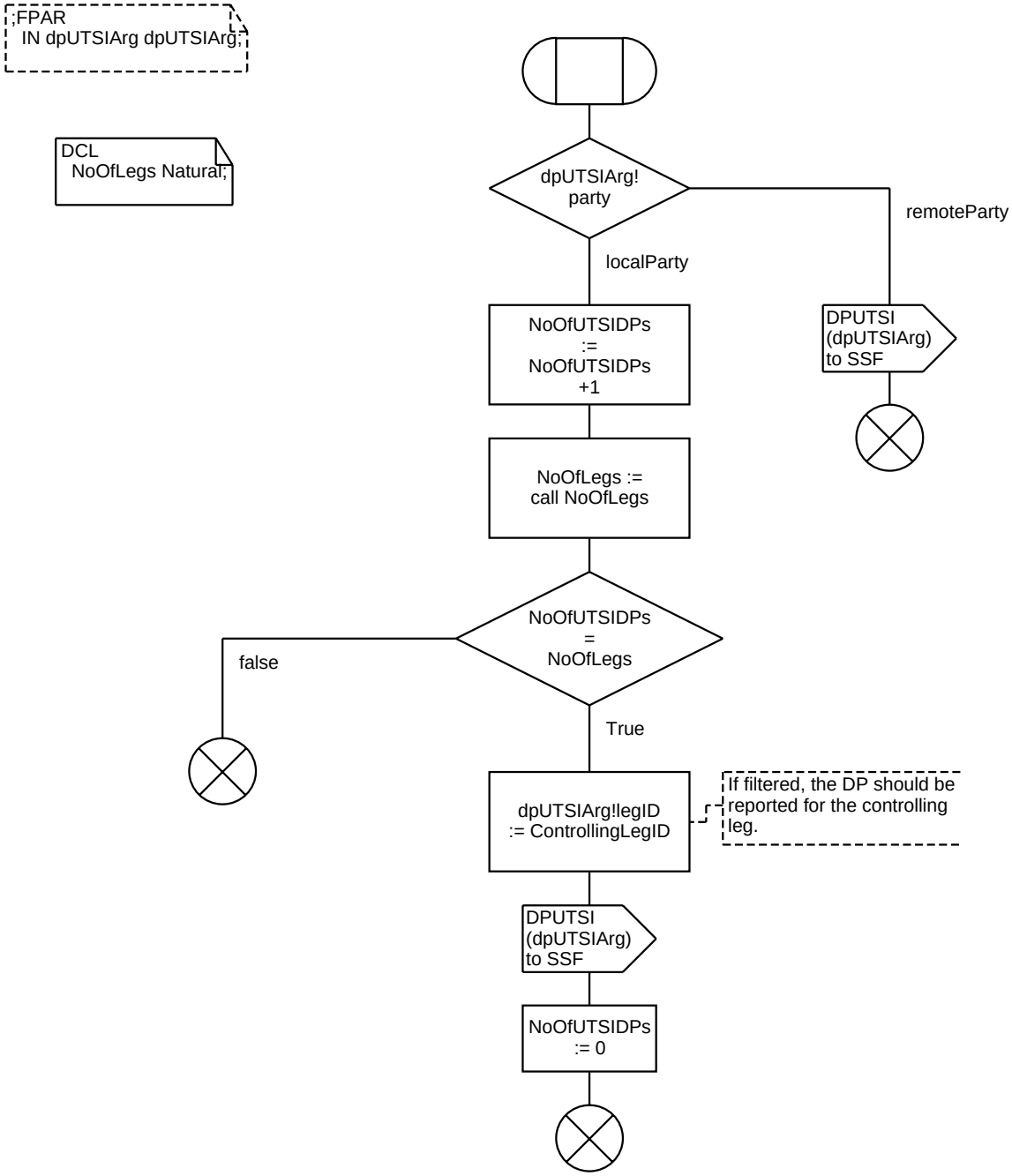
Procedure DPFiltering 4(4)



Procedure MapCIToBCSM 1(1)



Procedure DPFilteringUTSI 1(1)



Process Type SSF_FSM 1(50)

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType,
  serviceKey ServiceKey,
  CSA PId;
  SIGNALSET DPAbandon;
```

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

```
/* 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,
  continnue,
  primEnd,
  Abort;
ENDNEWTYPE;
```

CS

[(CS3_SSF_In)] [(CS3_SSF_Out)]

SRF

[(CS3_SRF_In)] [(CS3_SRF_Out)]

```
/* Note: The Dummy type is needed as a placeholder for the DisarmDP operator. */
NEWTYPE Dummy
 OPERATORS
  DisarmDP: EDPTableType, Integer −> EDPTableType;
  /* removes an entry in the EDPTable. */
 OPERATOR DisarmDP;
  FPAR list EDPTableType, index Integer;
 RETURNS EDPTableType;
 START;
    RETURN Substring(list, 1, index−1) // Substring (list, index+1, Length(list)−index);
 ENDOPERATOR;
ENDNEWTYPE;
```

```
/* The SSF−FSM maintains a table holding the implicit disarming rules */

NEWTYPE T3
  ARRAY (Boolean,Boolean)
ENDNEWTYPE;

NEWTYPE T2
  ARRAY (EventTypeBCSM,T3)
ENDNEWTYPE;

NEWTYPE T1
  ARRAY (Boolean,T2)
ENDNEWTYPE;

NEWTYPE ImplicitDisarmingTableType
  ARRAY (EventTypeBCSM,T1)
ENDNEWTYPE;
```

Process Type SSF_FSM 2(50)

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType,
  serviceKey ServiceKey,
  CSA PId;
  SIGNALSET DPAbandon;
```

```
/* Remote procedure exported by SSME_C
 * and called by SSF_FSM
 */

IMPORTED PROCEDURE CheckMatchingServiceFilteringCriteria;
FPAR
   IN DPArg,
   IN ServiceKey,
   IN/OUT FilteringCriteria;
RETURNS Boolean;

IMPORTED PROCEDURE CheckACG;
FPAR
   IN DPArg,
   IN ServiceKey,
   IN/OUT GapCriteria;
RETURNS Boolean;

IMPORTED PROCEDURE CheckCallFiltered;
FPAR IN FilteringCriteria;
RETURNS Boolean;

IMPORTED PROCEDURE SendSFR;
FPAR IN FilteringCriteria,
         IN Boolean;

IMPORTED PROCEDURE CountCall;
FPAR
 IN DPArg,
 IN FilteringCriteria;

IMPORTED PROCEDURE SFGetCause;
FPAR IN FilteringCriteria;
RETURNS Cause;

IMPORTED PROCEDURE SFInfoToSend;
FPAR IN FilteringCriteria;
RETURNS InformationToSend;


IMPORTED PROCEDURE CGGetCause;
FPAR IN GapCriteria;
RETURNS Cause;

IMPORTED PROCEDURE CGInfoToSend;
FPAR IN GapCriteria;
RETURNS InformationToSend;
```

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

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

Process Type SSF_FSM 3(50)

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType,
  serviceKey ServiceKey,
  CSA PId;
  SIGNALSET DPAbandon;
```

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

```
DCL
  /* IN 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,

/* SRF operations */
  paArg PlayAnnouncementArg,
  pacuiArg PromptAndCollectUserInformationArg,
  mrArg MessageReceivedArg,
  parmArg PromptAndReceiveMessageArg,
  riArg ReceivedInformationArg,
  scArg ScriptCloseArg,
  seArg ScriptEventArg,
  siArg ScriptInformationArg,
  scrArg ScriptRunArg,
  srrArg SpecializedResourceReportArg,

  ieArg ErrorArg;
```

```
DCL
  EDPTable EDPTableType,
  ImplicitDisarmingTable ImplicitDisarmingTableType,
  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,
  requests Integer,
  ReleaseCallBuffered Boolean := false,
  fc FilteringCriteria,
  gc GapCriteria,
  UserAbandon Boolean,
  AlwaysReportMidCall Boolean := False,

  /* Signal Parameter Declarations */
  legIdTable LegIdTableType;
```

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

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

Process Type SSF_FSM 4(50)

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType,
  serviceKey ServiceKey,
  CSA PId;
  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;
```

Process Type SSF_FSM 5(50)

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType,
  serviceKey ServiceKey,
  CSA PId;
  SIGNALSET DPAbandon;
```

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

```
/*
Event List Procedures
*/
```

DP_Armed — Generic procedure Check if the DP is armed

Exist_Leg — Returns true if a given LegID exist or not.

InitializeImplicit_DisarmingTable — Initialize the table holding the implicit disarming rules.

RemoveDP — Remove (if present) ian armed EDP

PerformImplicit_Disarming — Disarm EDPs according to the implicit disarming rules.

```
/* Management procedures. */
```

```
/* 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 2.4.2 Q.1238−2. */
```

Call_Information_Report_Pending — Send outstanding call information report

Increment_Outstanding_RequestsLegID — Increments the request counter for a given leg.

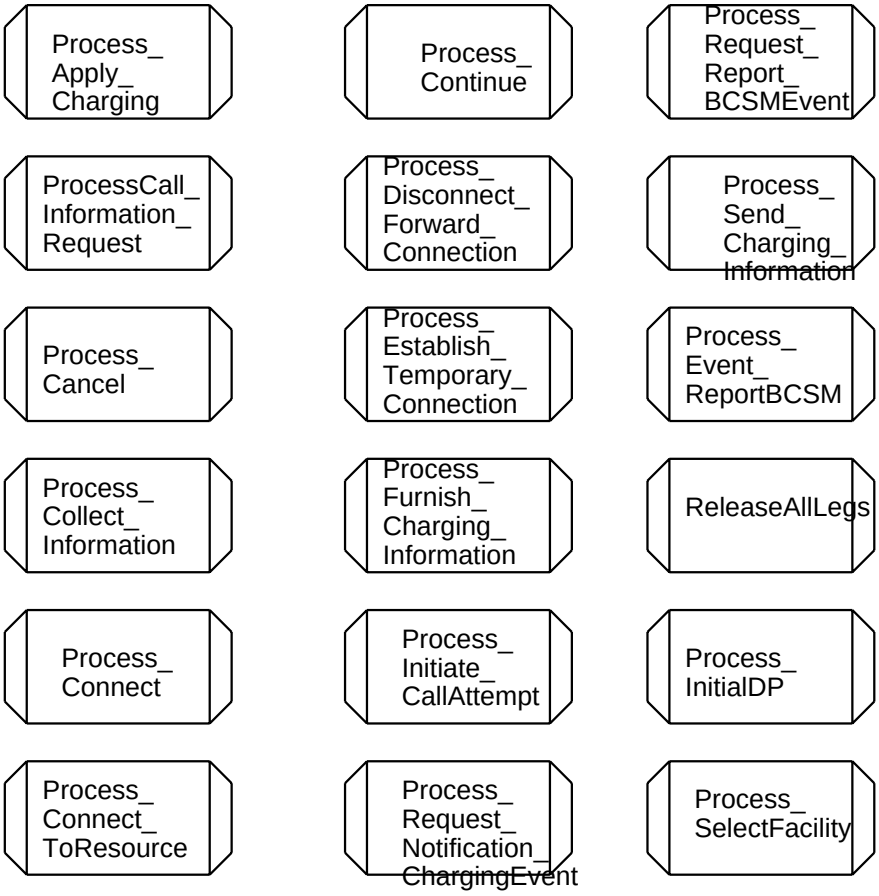
Apply_Charging_Report_Pending — Send outstanding call information report

Oustanding_Requests — Returns true if there are any outstanding requests.

Process Type SSF_FSM 6(50)

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType,
  serviceKey ServiceKey,
  CSA PId;
  SIGNALSET DPAbandon;
```

```
/*
IN Operation processing procedures
*/
```



Process Type SSF_FSM 7(50)

```
;FPAR
  startState SSFStateType,
  scfInitiated Boolean,
  controllingLegID LegType,
  serviceKey ServiceKey,
  CSA PId;
  SIGNALSET DPAbandon;
```

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

```
/* Operations for exporting
and importing legs. */
```

EXPORTED ExportEvent_Record

ProcessContinue_WithArgument

Process_Request_ReportUTSI

Import_Event_Record

Process_Disconnect_Leg

Process_SendSTUI

Process_Report_UTSI

```
/* Procedures for determining
if UTSI Events are armed for a
given leg. */
```

IsUTSI_Armed

```
/* Procedure for connecting
a filtered or gapped call to an SRF
and play the provided information */
```

TreatCall

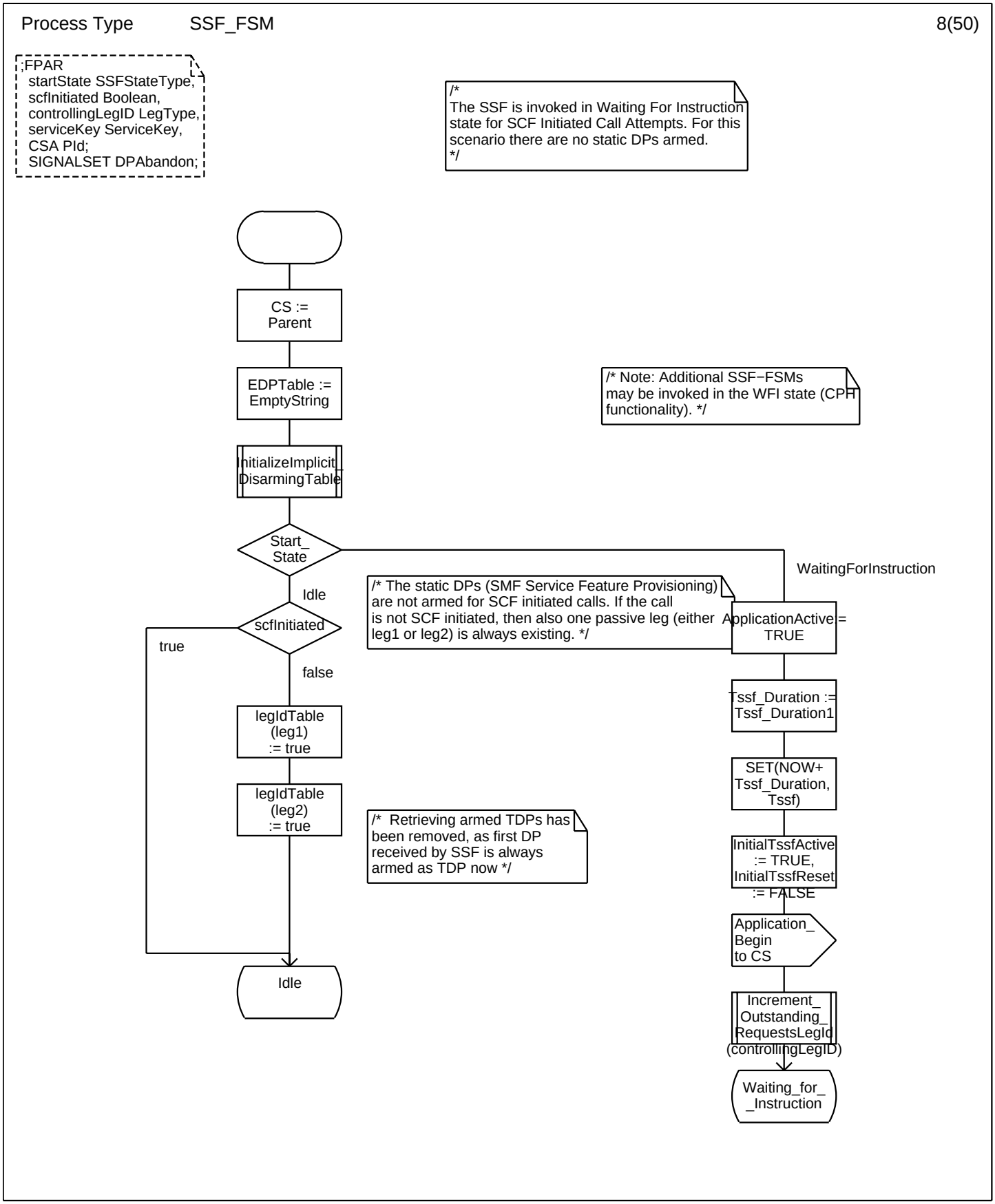
```
/* Procedure which increments the
counter of outstanding requests for
all legs within the control of this
SSF−FSM. This is necessary after a
CPH operation (all BCSMs in WaitMidCall).
See Q.1238−2 Section 2.4.2 */
```

Increment_Outstanding_RequestsCSId

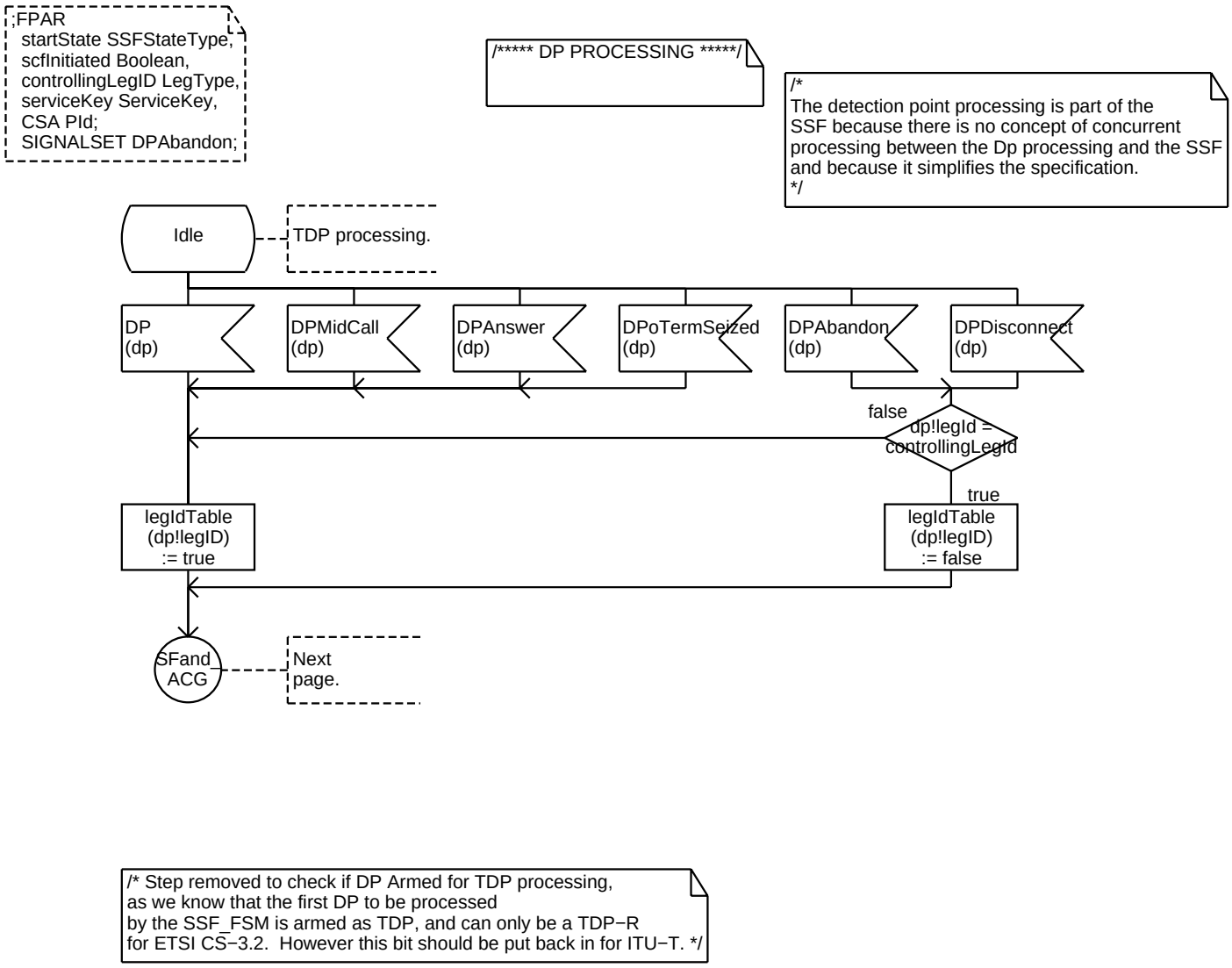
Decrement_Requests

Resume_PassiveLegs

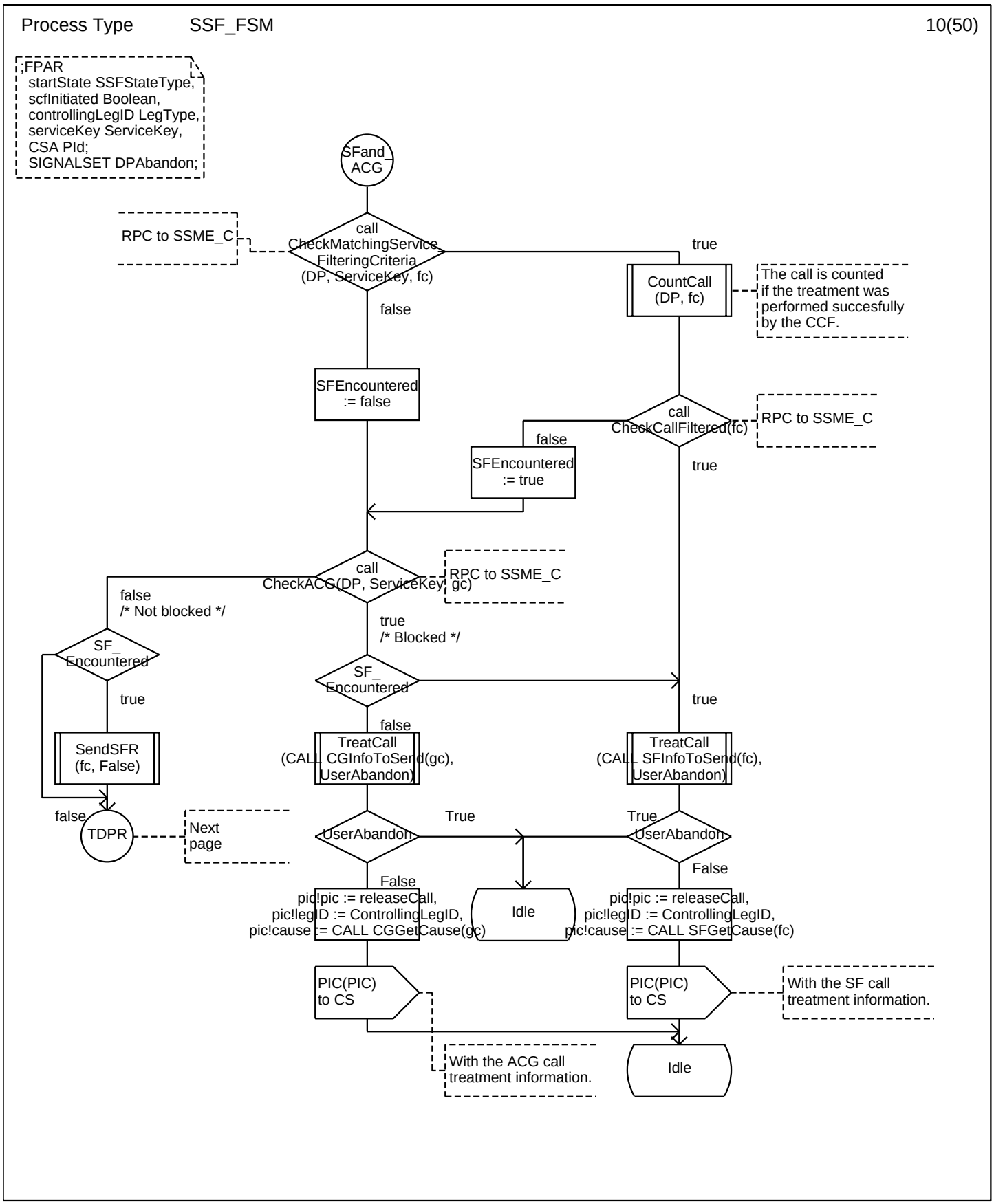
Process Type SSF_FSM 8(50)



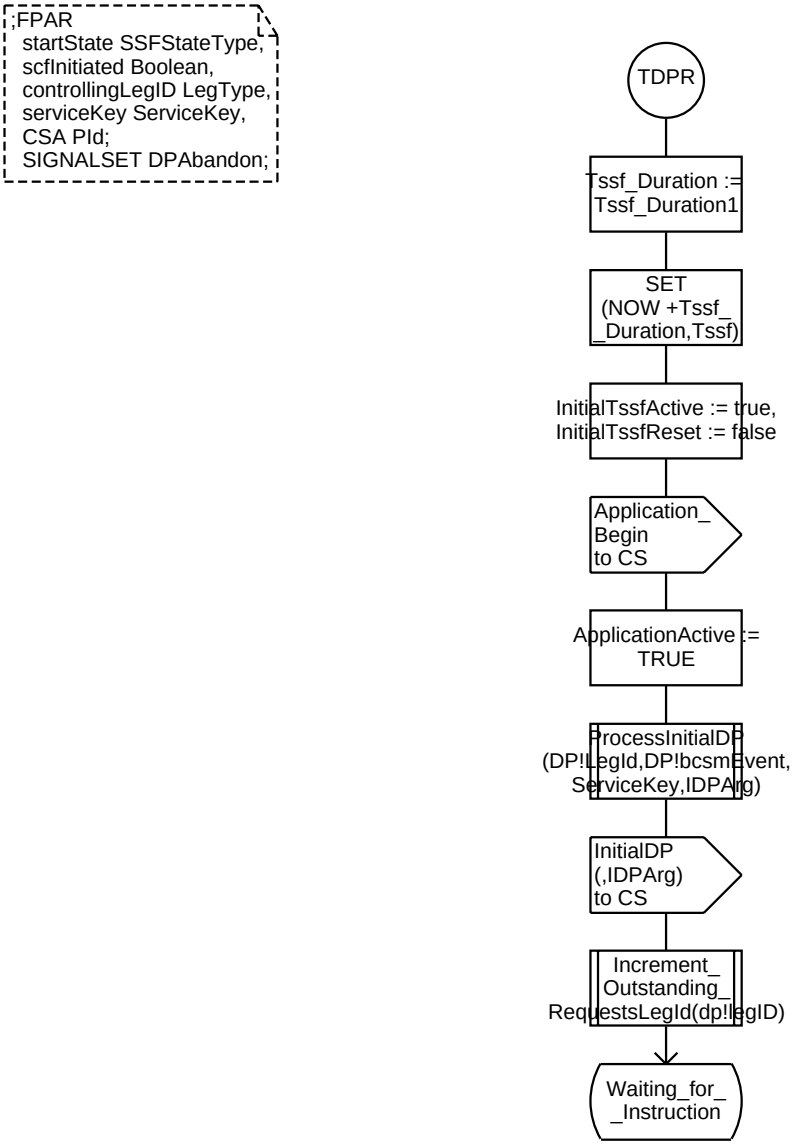
Process Type SSF_FSM 9(50)



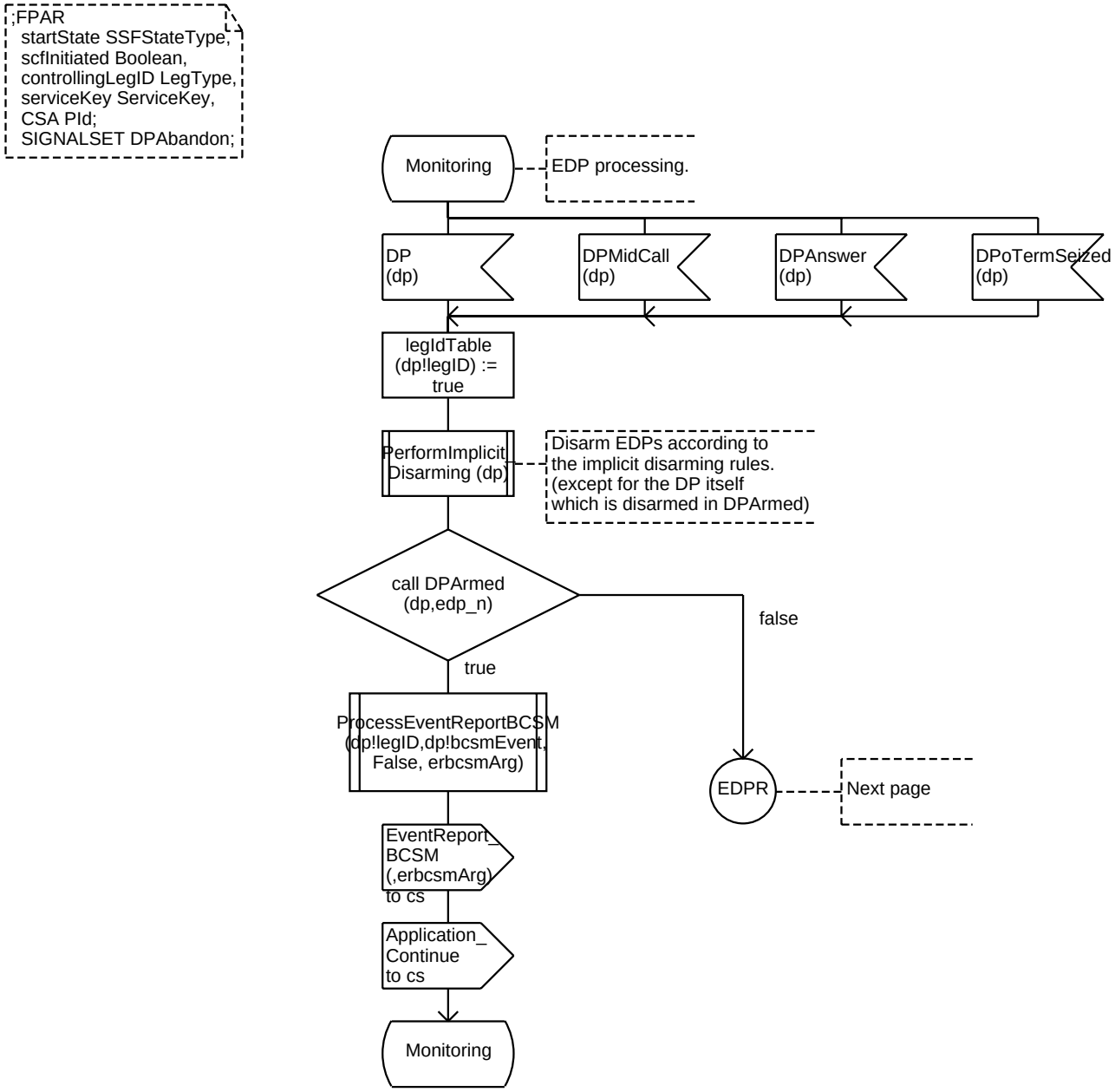
/* Step removed to check if DP Armed for TDP processing,
as we know that the first DP to be processed
by the SSF_FSM is armed as TDP, and can only be a TDP−R
for ETSI CS−3.2. However this bit should be put back in for ITU−T. */


Process Type SSF_FSM 10(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
SFand_ACG
RPC to SSME_C
call CheckMatchingServiceFilteringCriteria (DP, ServiceKey, fc)
true
CountCall (DP, fc)
The call is counted if the treatment was performed succesfully by the CCF.
false
SFEncountered := false
call CheckCallFiltered(fc)
RPC to SSME_C
false
SFEncountered := true
true
call CheckACG(DP, ServiceKey, gc)
RPC to SSME_C
false
/* Not blocked */
true
/* Blocked */
SF_ Encountered
SF_ Encountered
true
true
false
SendSFR (fc, False)
TreatCall (CALL CGInfoToSend(gc), UserAbandon)
TreatCall (CALL SFInfoToSend(fc), UserAbandon)
false
TDPR
Next page
UserAbandon
True
True
UserAbandon
False
False
pic!pic := releaseCall, pic!legID := ControllingLegID, pic!cause := CALL CGGetCause(gc)
Idle
pic!pic := releaseCall, pic!legID := ControllingLegID, pic!cause := CALL SFGetCause(fc)
PIC(PIC) to CS
PIC(PIC) to CS
With the SF call treatment information.
With the ACG call treatment information.
Idle

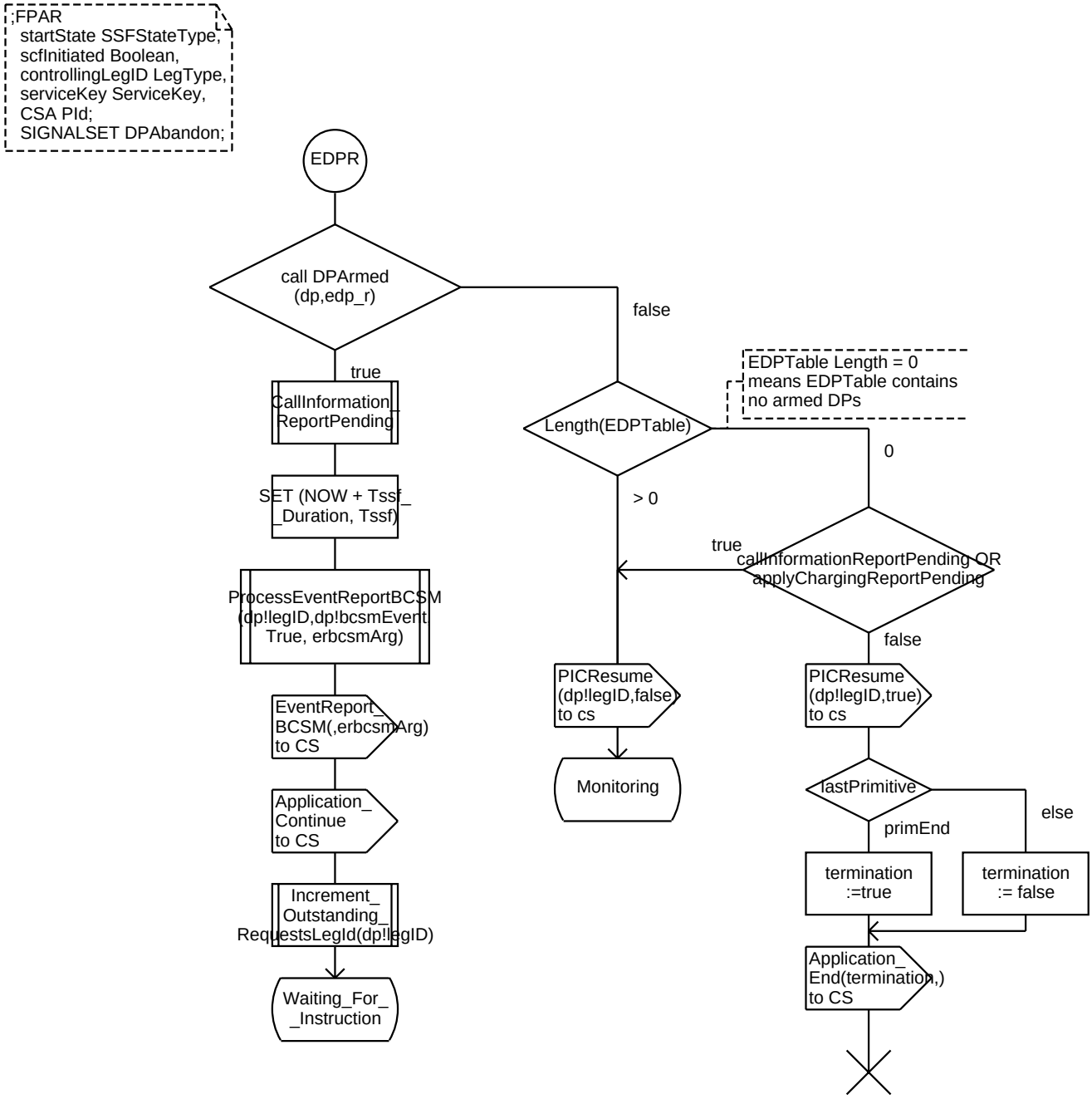
Process Type SSF_FSM 11(50)



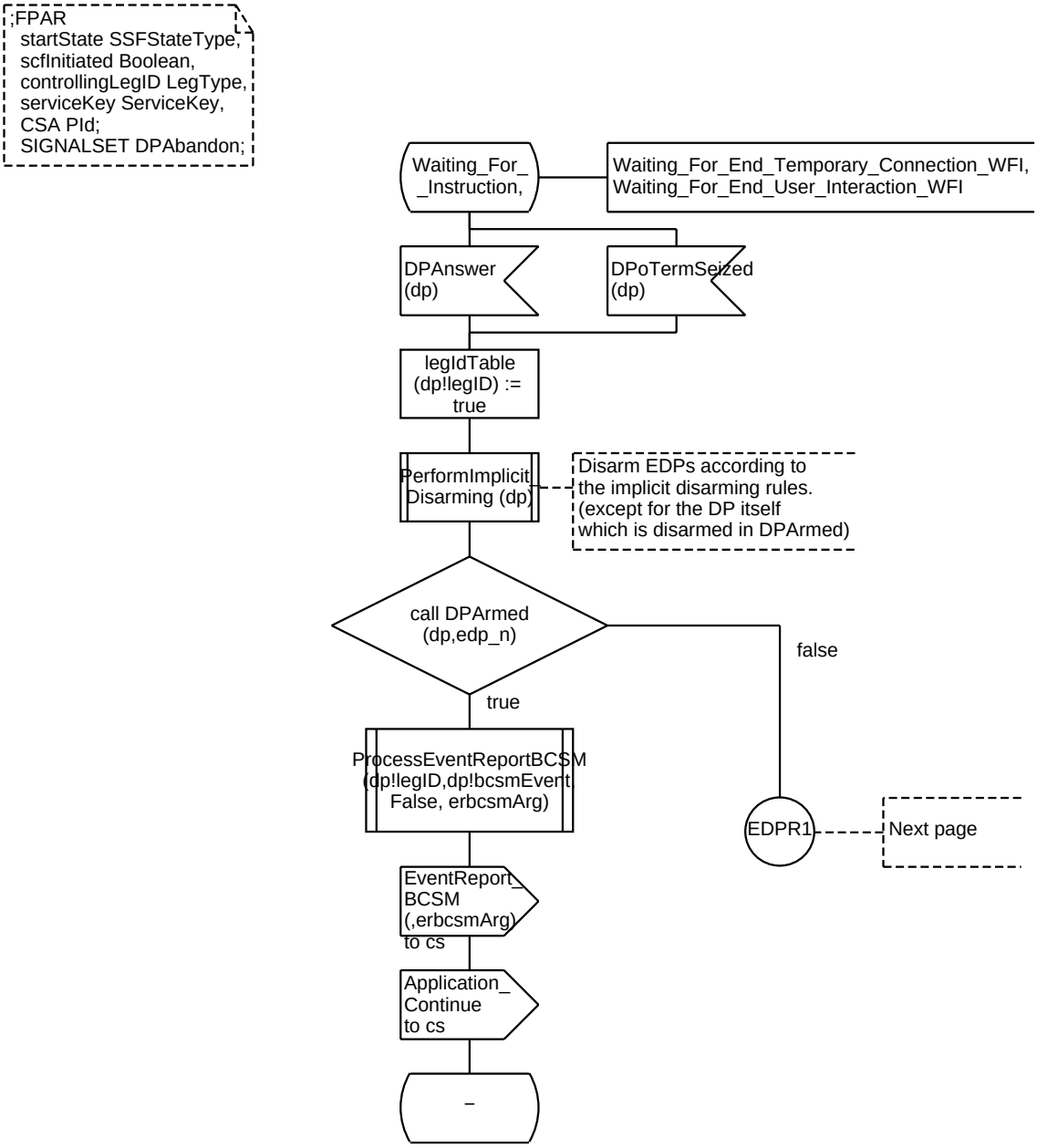
Process Type SSF_FSM 12(50)



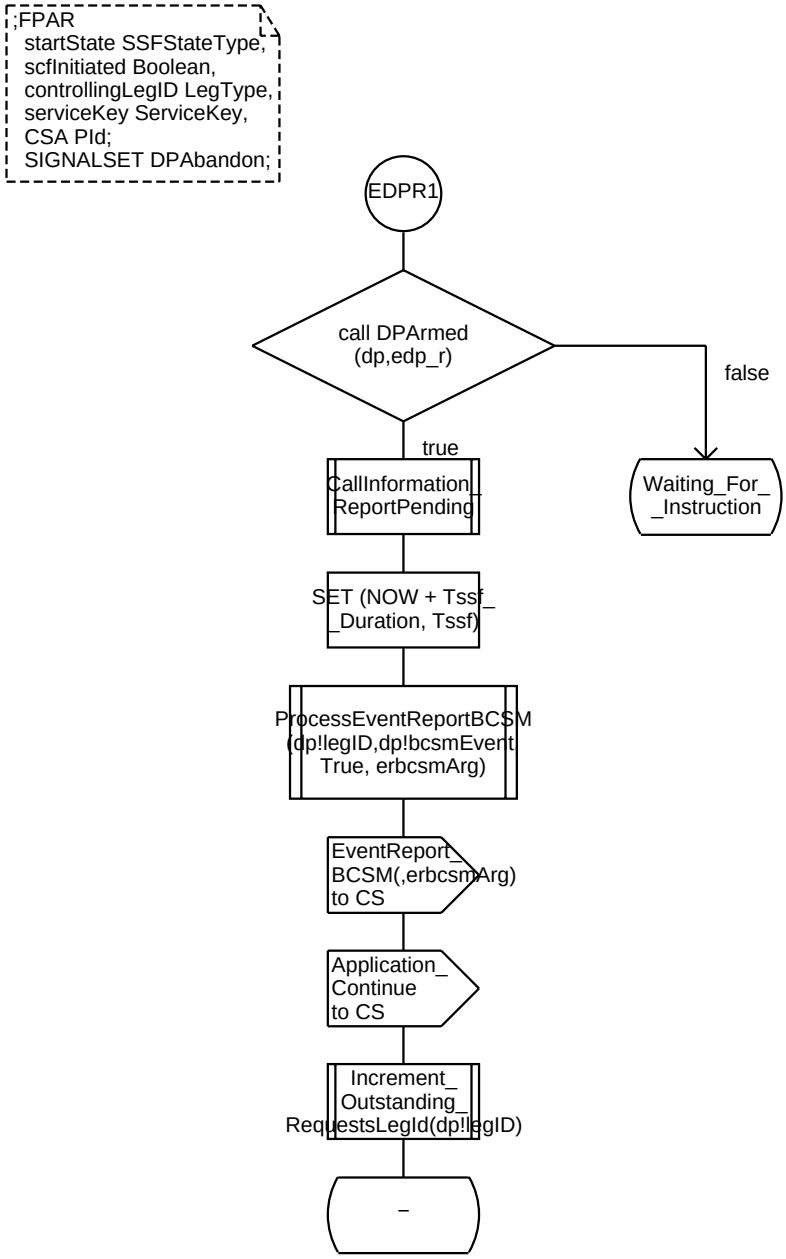
Process Type SSF_FSM 13(50)



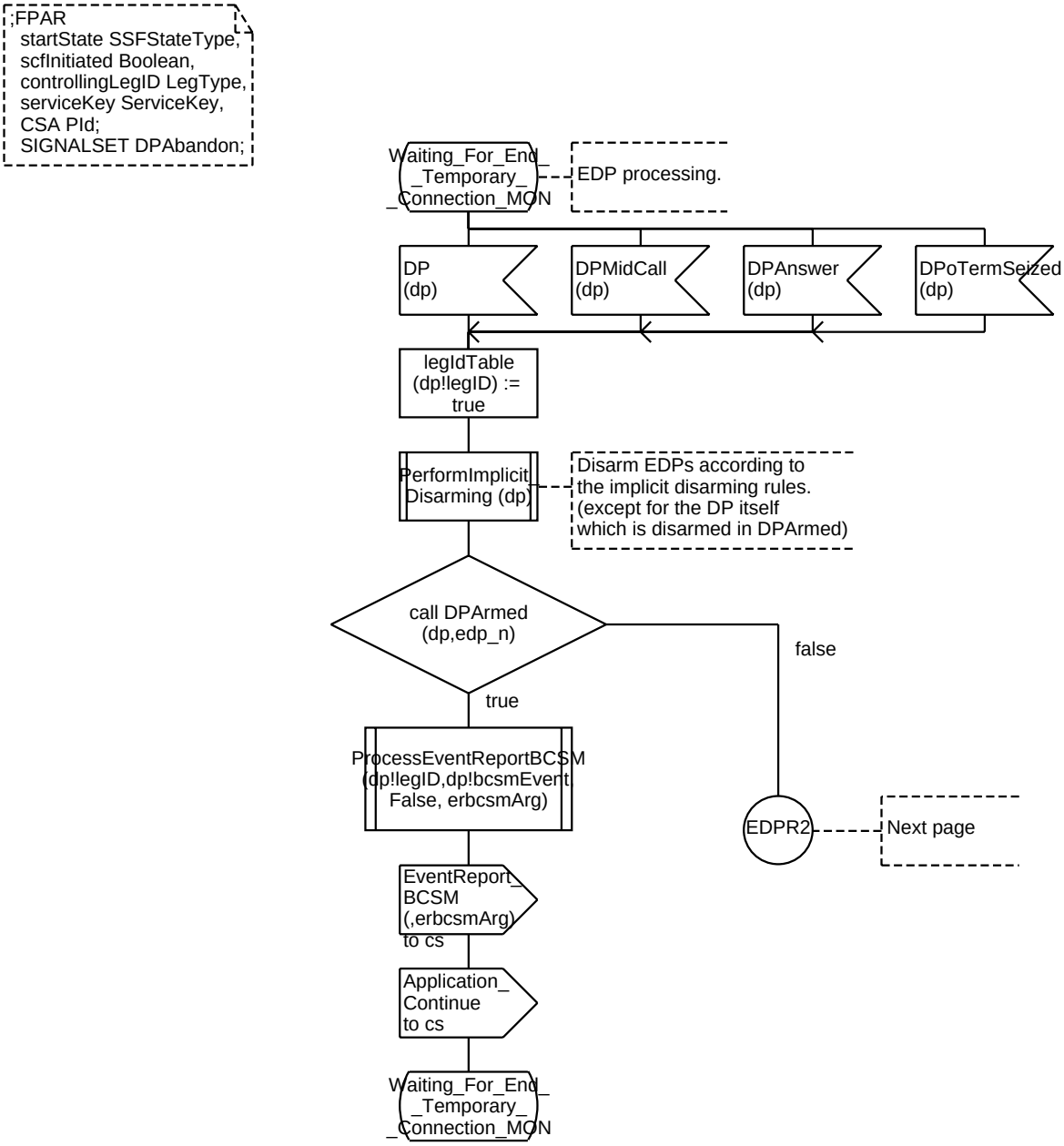
Process Type SSF_FSM 14(50)



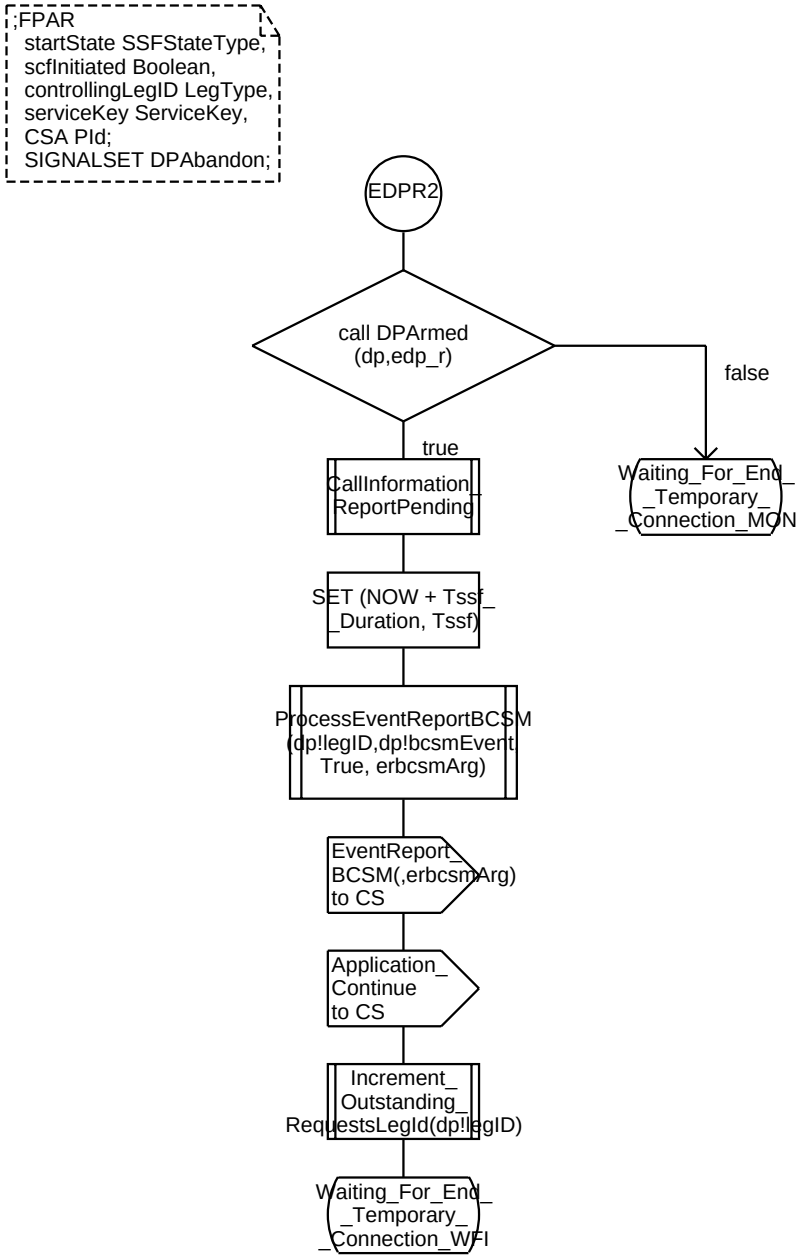
Process Type SSF_FSM 15(50)



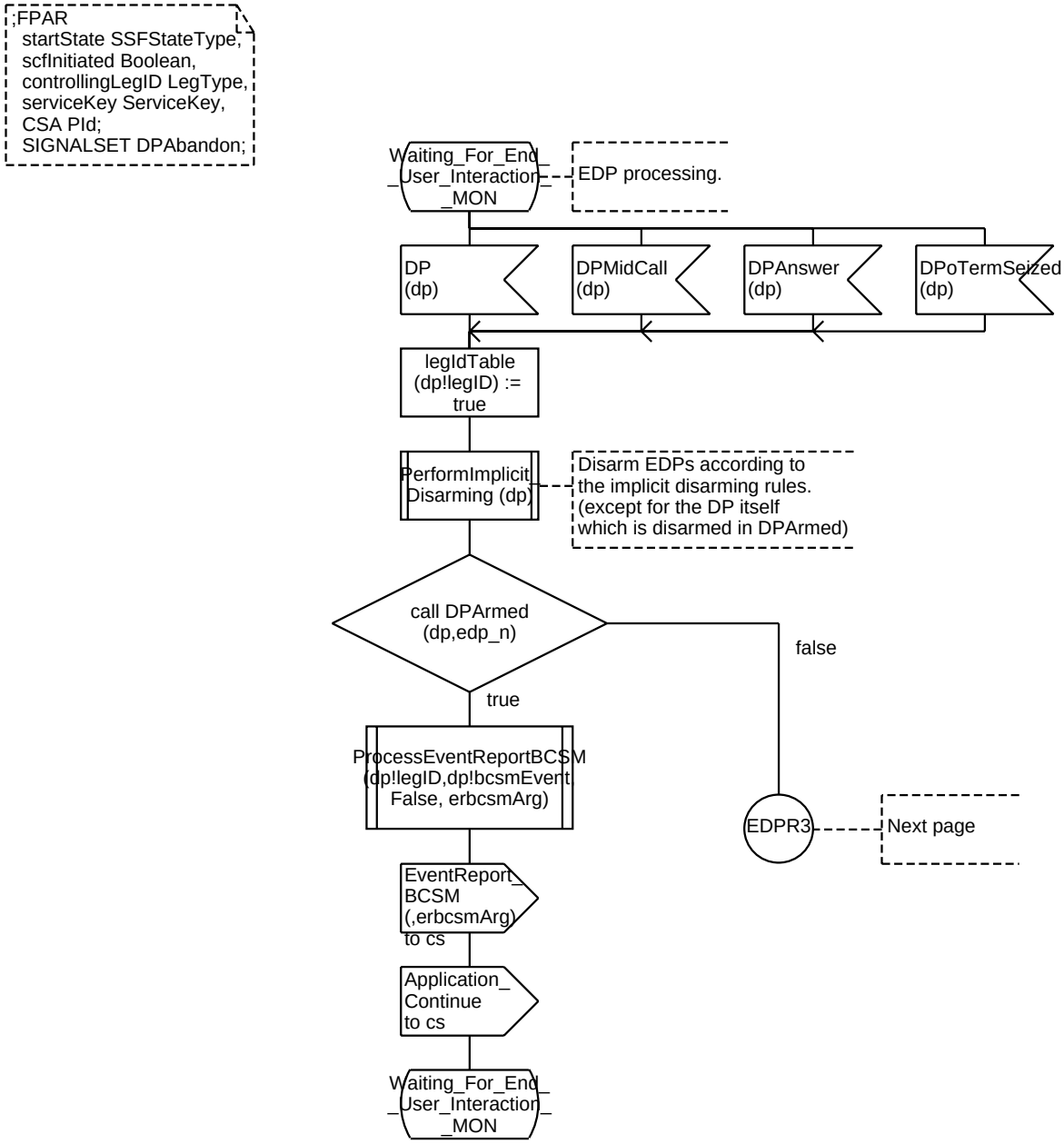
Process Type SSF_FSM 16(50)



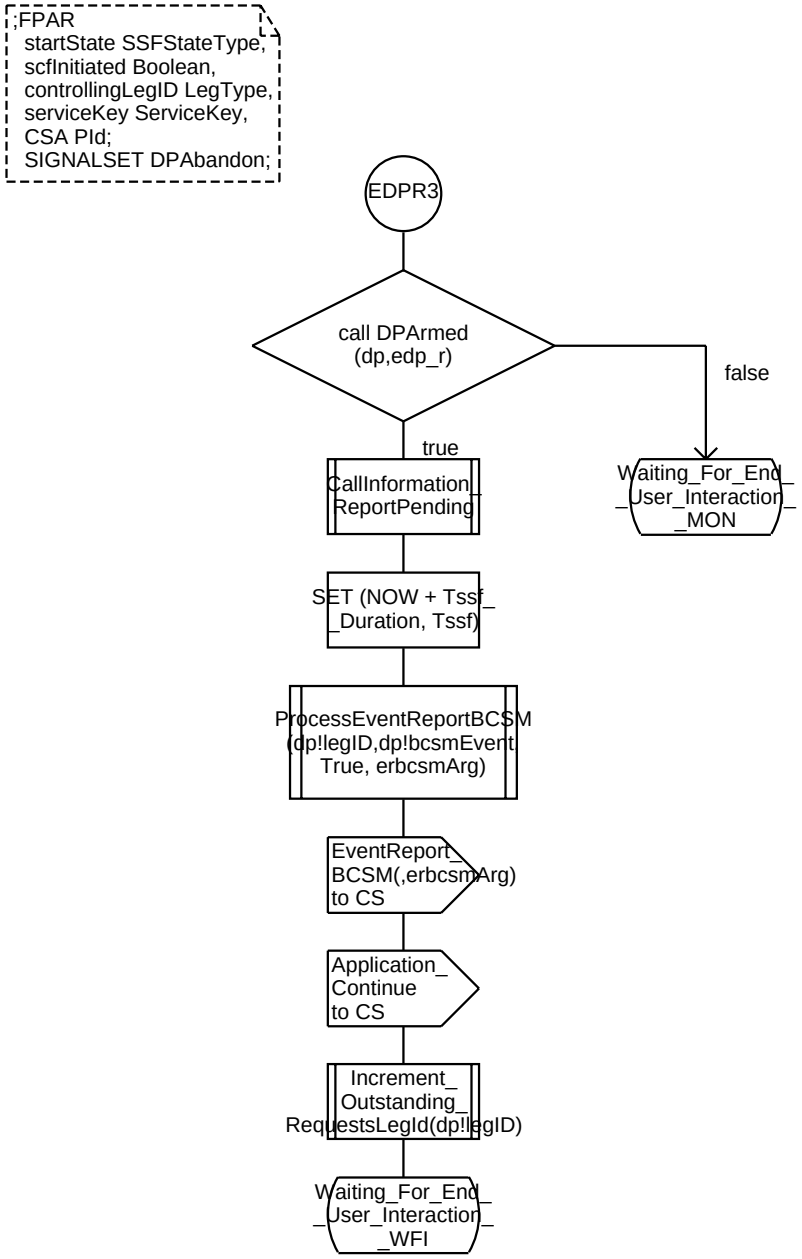
Process Type SSF_FSM 17(50)

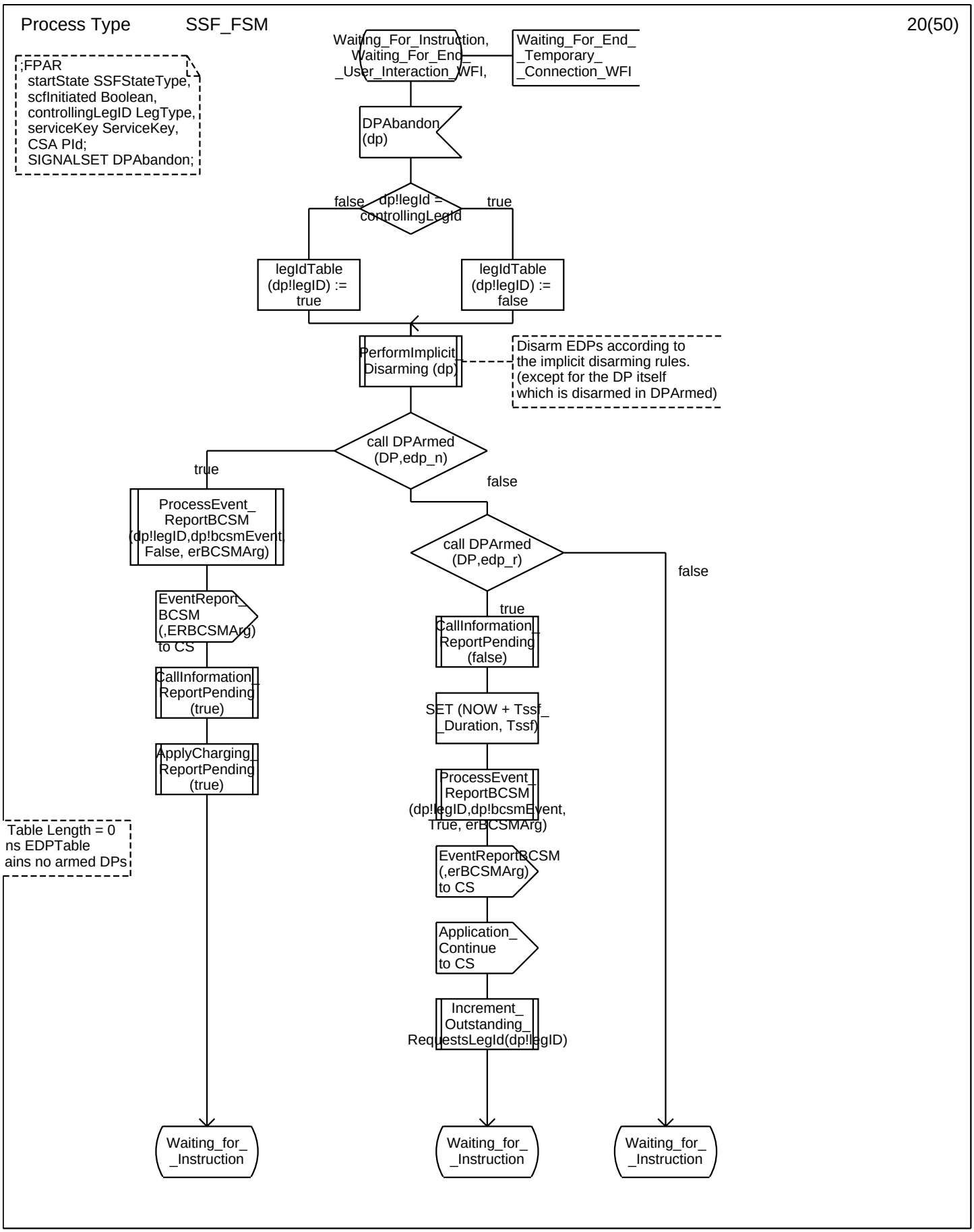


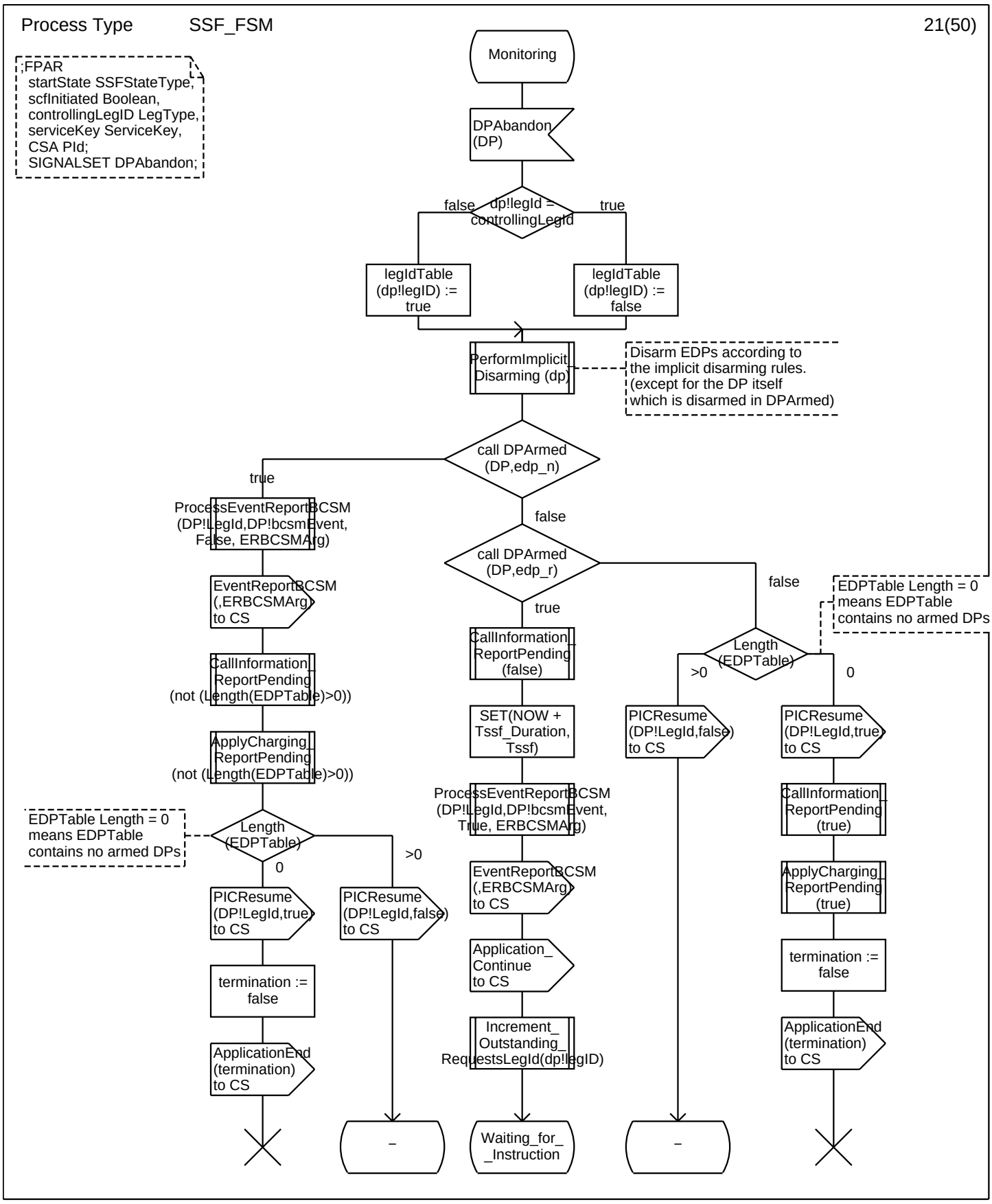
Process Type SSF_FSM 18(50)

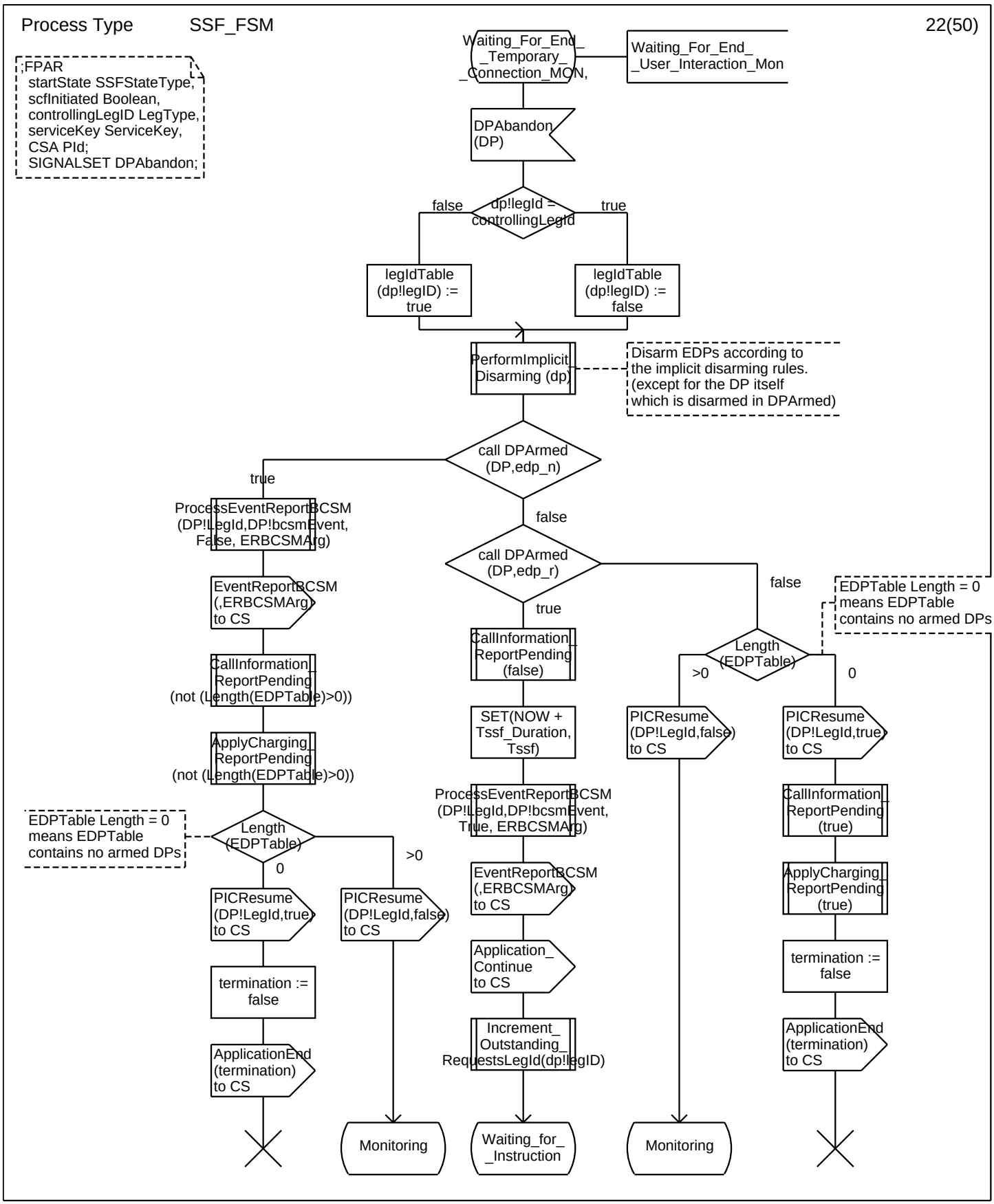


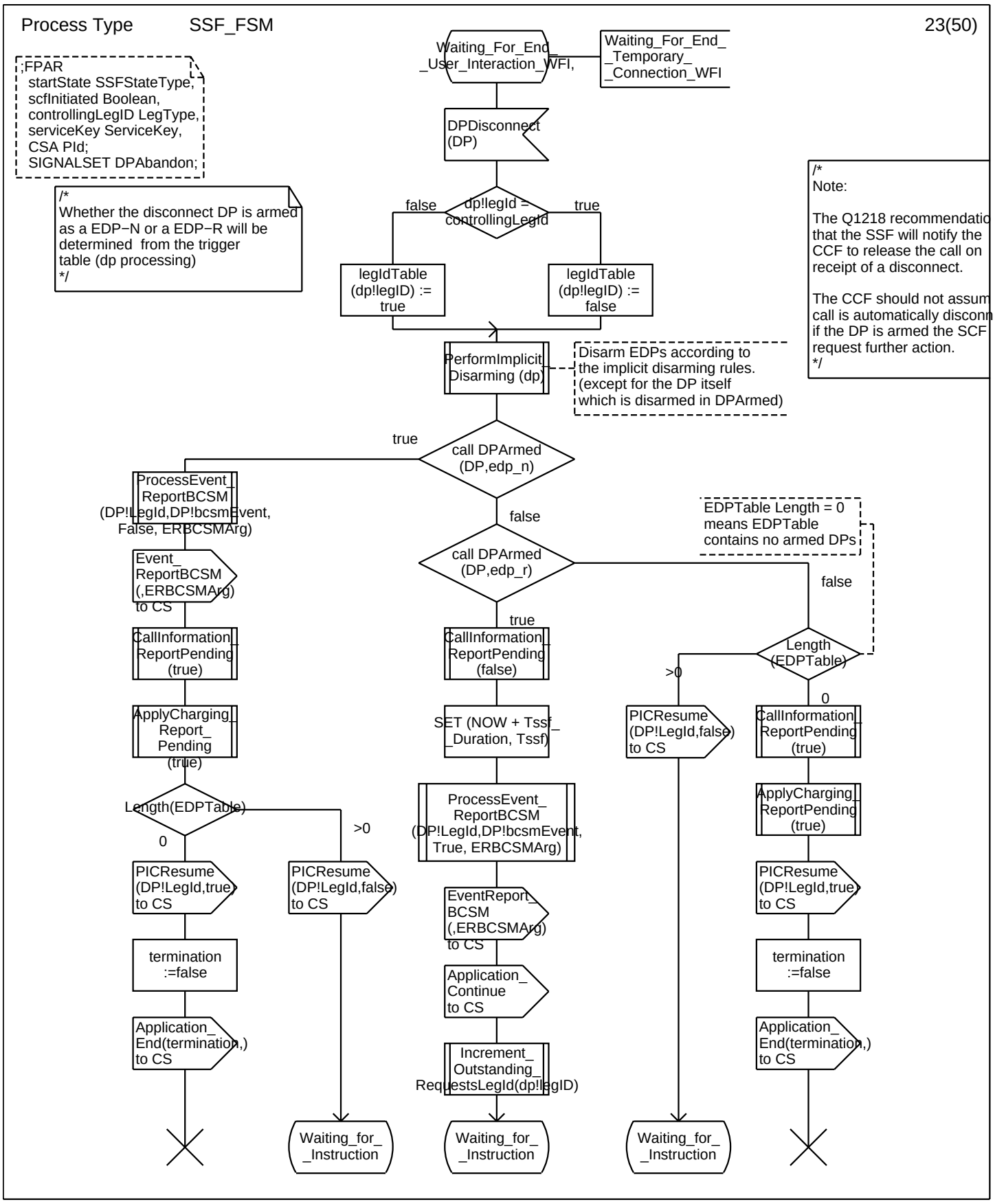
Process Type SSF_FSM 19(50)

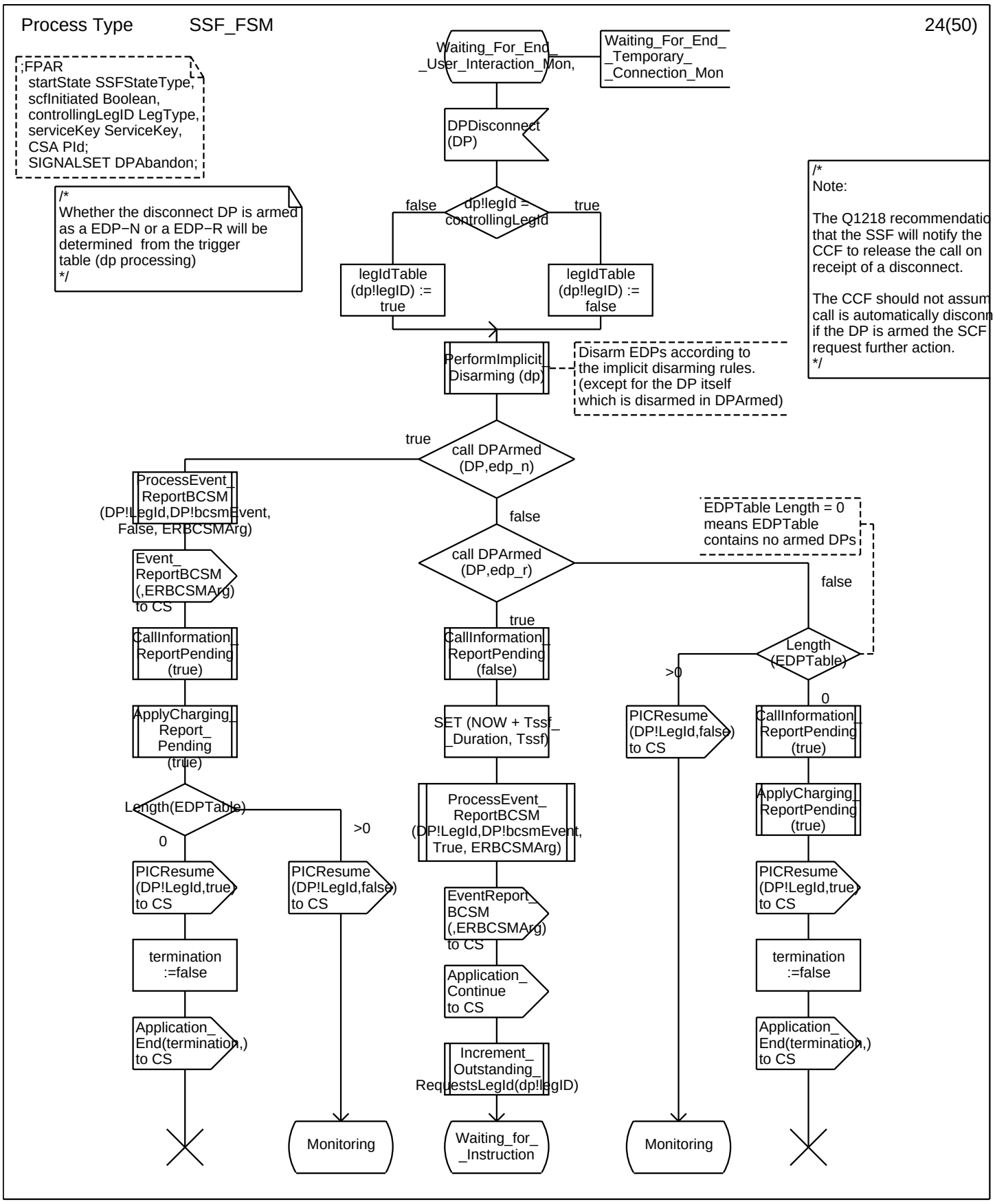


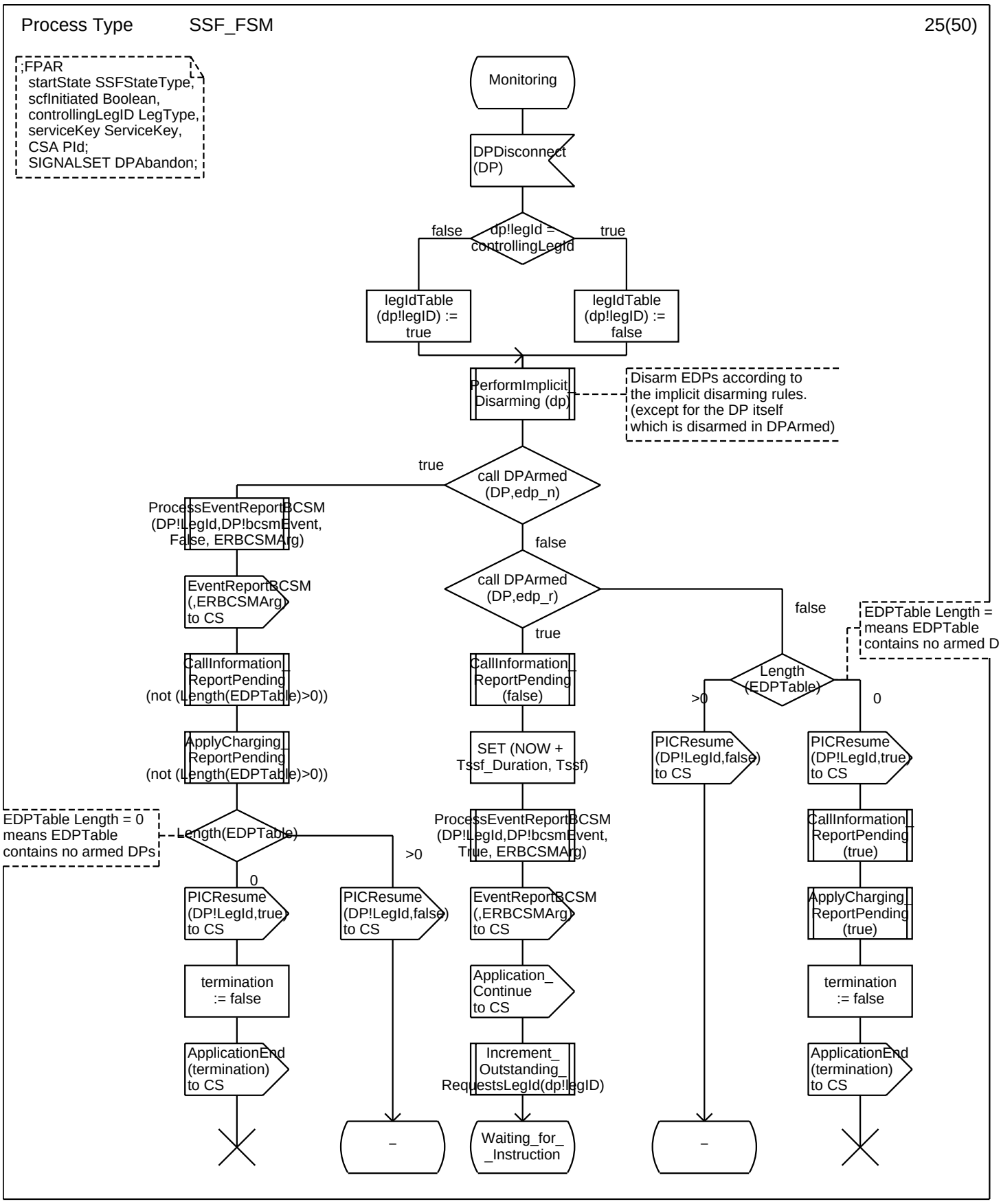

Process Type SSF_FSM
20(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
Waiting_For_Instruction,
Waiting_For_End_
_User_Interaction_WFI,
Waiting_For_End_
Temporary
_Connection_WFI
DPAbandon
(dp)
false
dp!legId =
controllingLegId
true
legIdTable
(dp!legID) :=
true
legIdTable
(dp!legID) :=
false
PerformImplicit_
Disarming (dp)
Disarm EDPs according to
the implicit disarming rules.
(except for the DP itself
which is disarmed in DPArmed)
call DPArmed
(DP,edp_n)
true
false
ProcessEvent_
ReportBCSM
(dp!legID,dp!bcsmEvent,
False, erBCSMArg)
call DPArmed
(DP,edp_r)
false
EventReport_
BCSM
(,ERBCSMArg)
to CS
true
CallInformation_
ReportPending
(false)
CallInformation_
ReportPending
(true)
SET (NOW + Tssf_
_Duration, Tssf)
ApplyCharging_
ReportPending
(true)
ProcessEvent_
ReportBCSM
(dp!legID,dp!bcsmEvent,
True, erBCSMArg)
Table Length = 0
ns EDPTable
ains no armed DPs
EventReportBCSM
(,erBCSMArg)
to CS
Application_
Continue
to CS
Increment_
Outstanding_
RequestsLegId(dp!legID)
Waiting_for_
_Instruction
Waiting_for_
_Instruction
Waiting_for_
_Instruction


Process Type SSF_FSM
21(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
Monitoring
DPAbandon (DP)
dp!legId = controllingLegId
false
true
legIdTable (dp!legID) := true
legIdTable (dp!legID) := false
PerformImplicit_Disarming (dp)
Disarm EDPs according to the implicit disarming rules. (except for the DP itself which is disarmed in DPArmed)
call DPArmed (DP,edp_n)
true
false
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent, False, ERBCSMArg)
EventReportBCSM (,ERBCSMArg) to CS
CallInformation_ReportPending (not (Length(EDPTable)>0))
ApplyCharging_ReportPending (not (Length(EDPTable)>0))
EDPTable Length = 0 means EDPTable contains no armed DPs
Length (EDPTable)
0
>0
PICResume (DP!LegId,true) to CS
PICResume (DP!LegId,false) to CS
termination := false
ApplicationEnd (termination) to CS
call DPArmed (DP,edp_r)
true
false
CallInformation_ReportPending (false)
SET(NOW + Tssf_Duration, Tssf)
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent, True, ERBCSMArg)
EventReportBCSM (,ERBCSMArg) to CS
Application_Continue to CS
Increment_Outstanding_RequestsLegId(dp!legID)
Waiting_for_Instruction
EDPTable Length = 0 means EDPTable contains no armed DPs
Length (EDPTable)
>0
0
PICResume (DP!LegId,false) to CS
PICResume (DP!LegId,true) to CS
CallInformation_ReportPending (true)
ApplyCharging_ReportPending (true)
termination := false
ApplicationEnd (termination) to CS

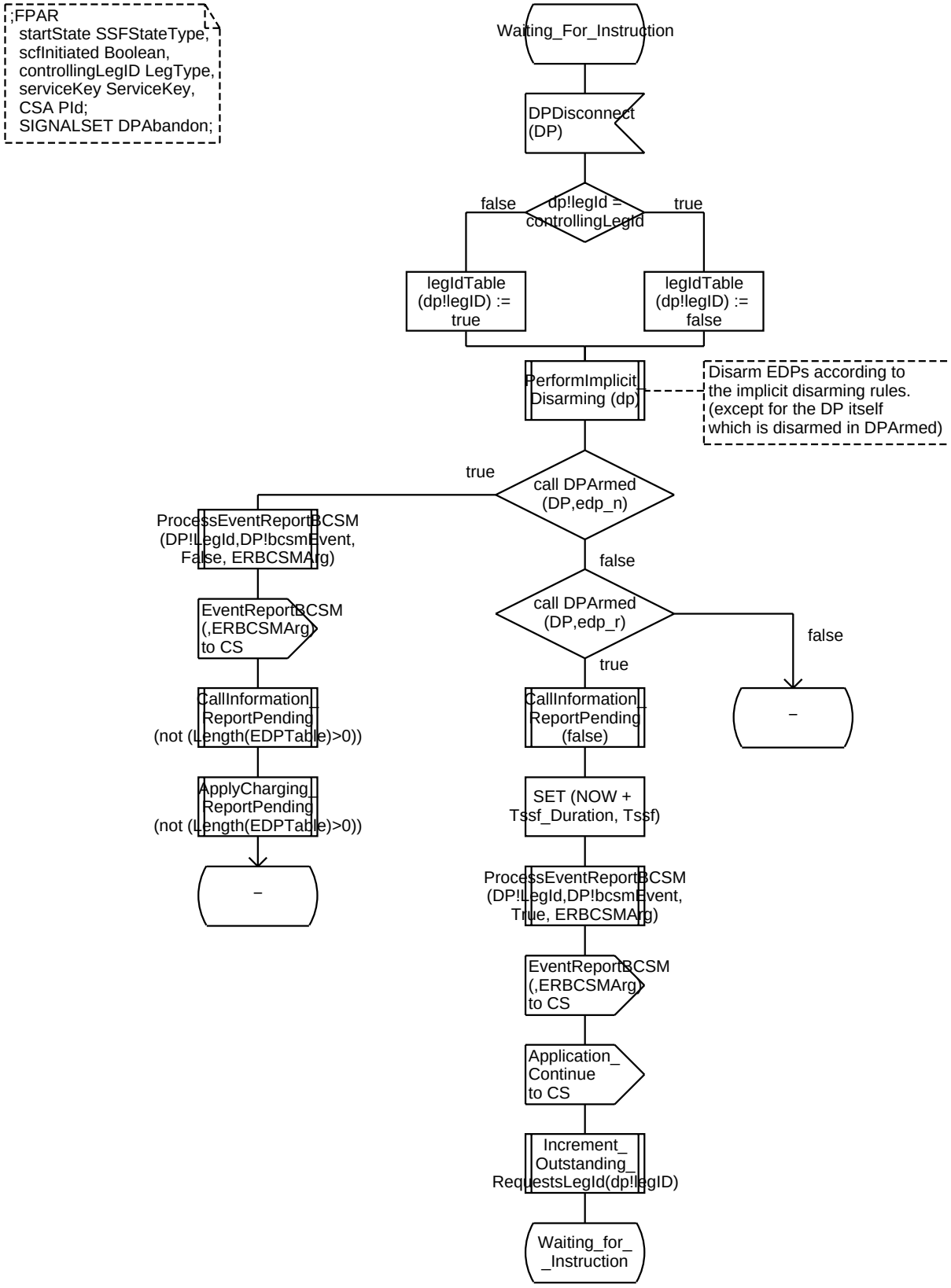

Process Type SSF_FSM
22(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
Waiting_For_End_
Temporary
_Connection_MON,
Waiting_For_End_
_User_Interaction_Mon
DPAbandon
(DP)
false
dp!legId =
controllingLegId
true
legIdTable
(dp!legID) :=
true
legIdTable
(dp!legID) :=
false
PerformImplicit_
Disarming (dp)
Disarm EDPs according to
the implicit disarming rules.
(except for the DP itself
which is disarmed in DPArmed)
call DPArmed
(DP,edp_n)
true
ProcessEventReportBCSM
(DP!LegId,DP!bcsmEvent,
False, ERBCSMArg)
EventReportBCSM
(,ERBCSMArg)
to CS
CallInformation_
ReportPending
(not (Length(EDPTable)>0))
ApplyCharging_
ReportPending
(not (Length(EDPTable)>0))
EDPTable Length = 0
means EDPTable
contains no armed DPs
Length
(EDPTable)
0
>0
PICResume
(DP!LegId,true)
to CS
PICResume
(DP!LegId,false)
to CS
termination :=
false
ApplicationEnd
(termination)
to CS
Monitoring
false
call DPArmed
(DP,edp_r)
true
CallInformation_
ReportPending
(false)
SET(NOW +
Tssf_Duration,
Tssf)
ProcessEventReportBCSM
(DP!LegId,DP!bcsmEvent,
True, ERBCSMArg)
EventReportBCSM
(,ERBCSMArg)
to CS
Application_
Continue
to CS
Increment_
Outstanding_
RequestsLegId(dp!legID)
Waiting_for_
_Instruction
false
EDPTable Length = 0
means EDPTable
contains no armed DPs
Length
(EDPTable)
>0
0
PICResume
(DP!LegId,false)
to CS
Monitoring
PICResume
(DP!LegId,true)
to CS
CallInformation_
ReportPending
(true)
ApplyCharging_
ReportPending
(true)
termination :=
false
ApplicationEnd
(termination)
to CS


Process Type SSF_FSM
23(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
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_End_User_Interaction_WFI,
Waiting_For_End_Temporary_Connection_WFI
DPDisconnect (DP)
dp!legId = controllingLegId
false
true
legIdTable (dp!legID) := true
legIdTable (dp!legID) := false
/*
Note:
The Q1218 recommendatio that the SSF will notify the CCF to release the call on receipt of a disconnect.
The CCF should not assum call is automatically disconn if the DP is armed the SCF request further action.
*/
PerformImplicit_Disarming (dp)
Disarm EDPs according to the implicit disarming rules. (except for the DP itself which is disarmed in DPArmed)
call DPArmed (DP,edp_n)
true
false
ProcessEvent_ReportBCSM (DP!LegId,DP!bcsmEvent, False, ERBCSMArg)
Event_ReportBCSM (,ERBCSMArg) to CS
CallInformation_ReportPending (true)
ApplyCharging_Report_Pending (true)
Length(EDPTable)
0
>0
PICResume (DP!LegId,true) to CS
PICResume (DP!LegId,false) to CS
termination :=false
Application_End(termination,) to CS
Waiting_for_Instruction
call DPArmed (DP,edp_r)
EDPTable Length = 0 means EDPTable contains no armed DPs
true
false
CallInformation_ReportPending (false)
SET (NOW + Tssf_Duration, Tssf)
ProcessEvent_ReportBCSM (DP!LegId,DP!bcsmEvent, True, ERBCSMArg)
EventReport_BCSM (,ERBCSMArg) to CS
Application_Continue to CS
Increment_Outstanding_RequestsLegId(dp!legID)
Waiting_for_Instruction
Length (EDPTable)
>0
0
PICResume (DP!LegId,false) to CS
CallInformation_ReportPending (true)
ApplyCharging_ReportPending (true)
PICResume (DP!LegId,true) to CS
termination :=false
Application_End(termination,) to CS
Waiting_for_Instruction

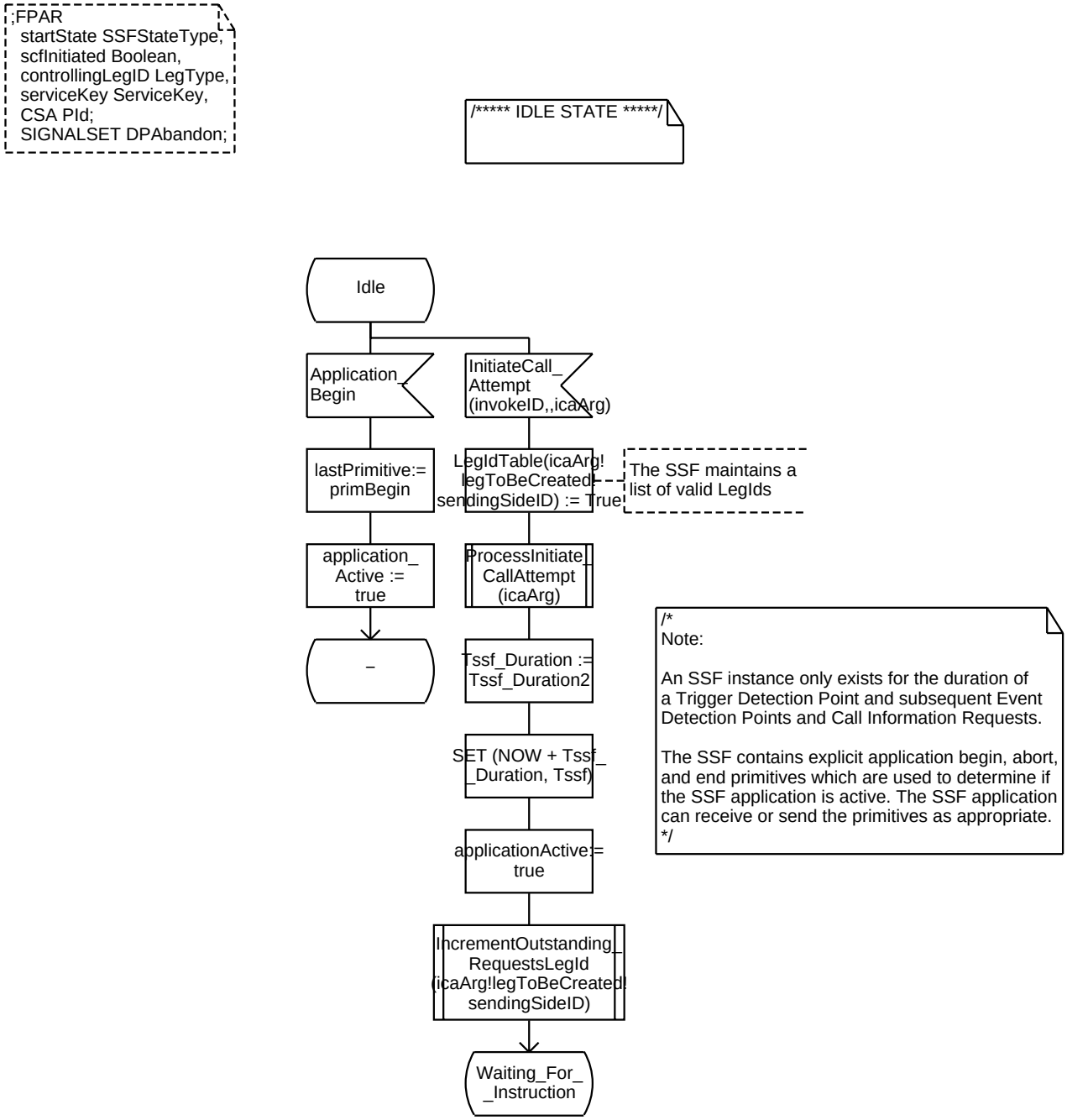

Process Type SSF_FSM
24(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
/*
Whether the disconnect DP is armed as a EDP−N or a EDP−R will be determined from the trigger table (dp processing)
*/
Waiting_For_End_User_Interaction_Mon,
Waiting_For_End_Temporary_Connection_Mon
DPDisconnect (DP)
dp!legId = controllingLegId
false
true
legIdTable (dp!legID) := true
legIdTable (dp!legID) := false
/*
Note:
The Q1218 recommendatio that the SSF will notify the CCF to release the call on receipt of a disconnect.
The CCF should not assum call is automatically disconn if the DP is armed the SCF request further action.
*/
PerformImplicit_Disarming (dp)
Disarm EDPs according to the implicit disarming rules. (except for the DP itself which is disarmed in DPArmed)
call DPArmed (DP,edp_n)
true
false
ProcessEvent_ReportBCSM (DP!LegId,DP!bcsmEvent, False, ERBCSMArg)
Event_ReportBCSM (,ERBCSMArg) to CS
CallInformation_ReportPending (true)
ApplyCharging_Report_Pending (true)
Length(EDPTable)
0
>0
PICResume (DP!LegId,true) to CS
PICResume (DP!LegId,false) to CS
termination :=false
Application_End(termination,) to CS
Monitoring
call DPArmed (DP,edp_r)
EDPTable Length = 0 means EDPTable contains no armed DPs
true
false
CallInformation_ReportPending (false)
SET (NOW + Tssf_Duration, Tssf)
ProcessEvent_ReportBCSM (DP!LegId,DP!bcsmEvent, True, ERBCSMArg)
EventReport_BCSM (,ERBCSMArg) to CS
Application_Continue to CS
Increment_Outstanding_RequestsLegId(dp!legID)
Waiting_for_Instruction
Length (EDPTable)
>0
0
PICResume (DP!LegId,false) to CS
CallInformation_ReportPending (true)
ApplyCharging_ReportPending (true)
PICResume (DP!LegId,true) to CS
termination :=false
Application_End(termination,) to CS
Monitoring

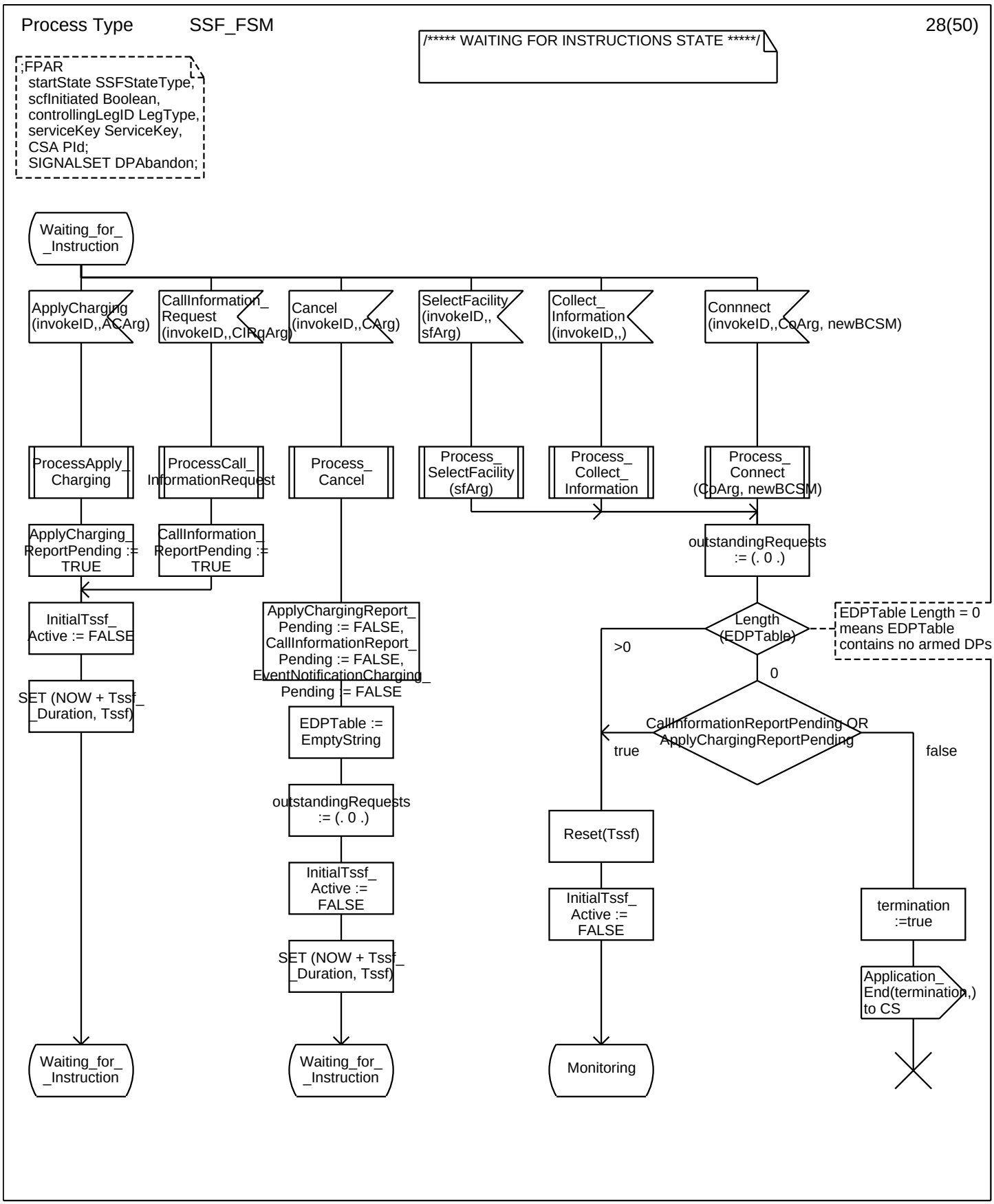

Process Type SSF_FSM 25(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
Monitoring
DPDisconnect (DP)
dp!legId = controllingLegId
false
true
legIdTable (dp!legID) := true
legIdTable (dp!legID) := false
PerformImplicit_Disarming (dp)
Disarm EDPs according to the implicit disarming rules. (except for the DP itself which is disarmed in DPArmed)
call DPArmed (DP,edp_n)
true
false
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent, False, ERBCSMArg)
EventReportBCSM (,ERBCSMArg) to CS
CallInformation_ReportPending (not (Length(EDPTable)>0))
ApplyCharging_ReportPending (not (Length(EDPTable)>0))
EDPTable Length = 0 means EDPTable contains no armed DPs
Length(EDPTable)
0
>0
PICResume (DP!LegId,true) to CS
PICResume (DP!LegId,false) to CS
termination := false
ApplicationEnd (termination) to CS
–
call DPArmed (DP,edp_r)
true
false
CallInformation_ReportPending (false)
SET (NOW + Tssf_Duration, Tssf)
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent, True, ERBCSMArg)
EventReportBCSM (,ERBCSMArg) to CS
Application_Continue to CS
Increment_Outstanding_RequestsLegId(dp!legID)
Waiting_for_Instruction
Length (EDPTable)
EDPTable Length = 0 means EDPTable contains no armed D
>0
0
PICResume (DP!LegId,false) to CS
PICResume (DP!LegId,true) to CS
CallInformation_ReportPending (true)
ApplyCharging_ReportPending (true)
termination := false
ApplicationEnd (termination) to CS
–

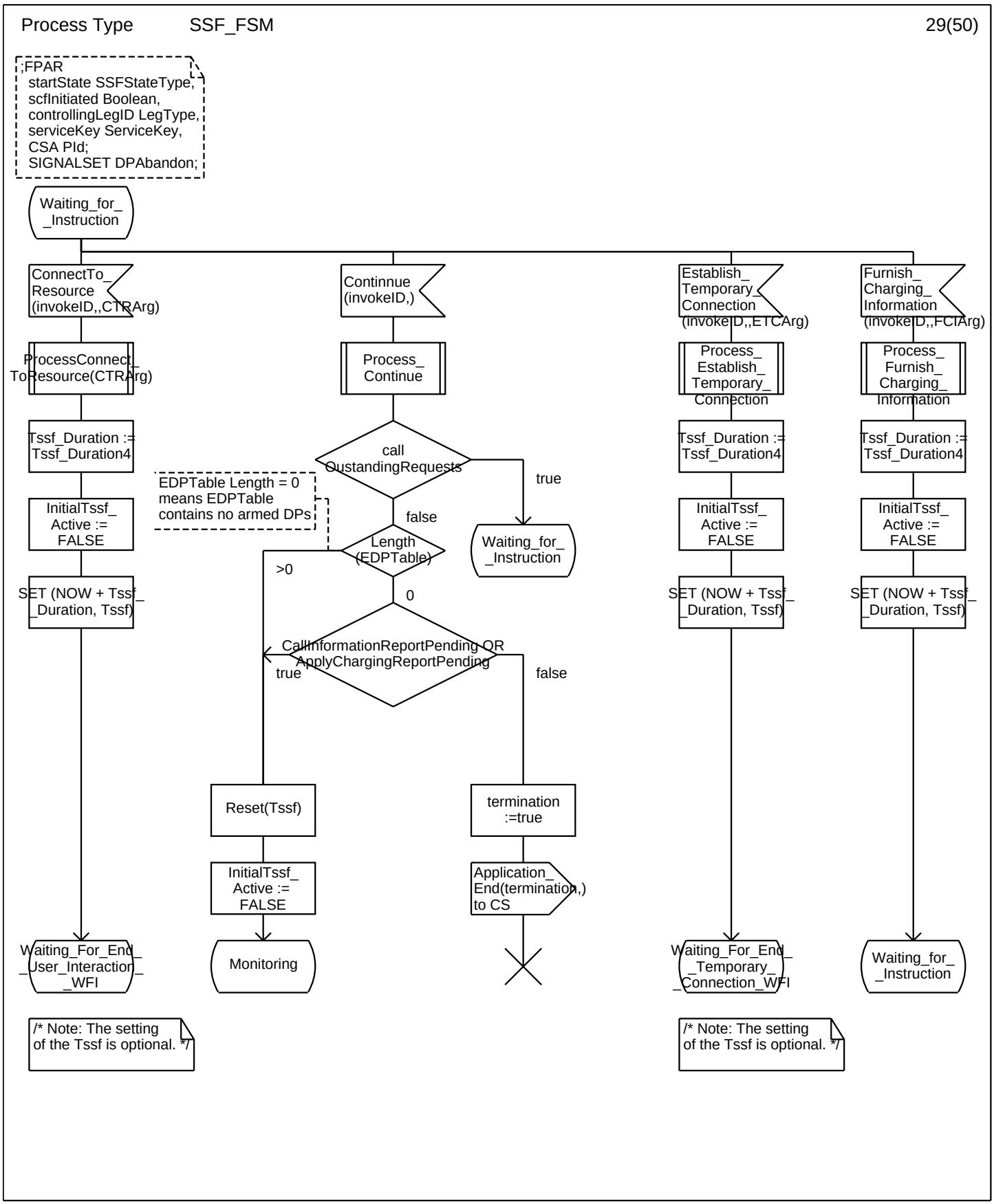
Process Type SSF_FSM 26(50)



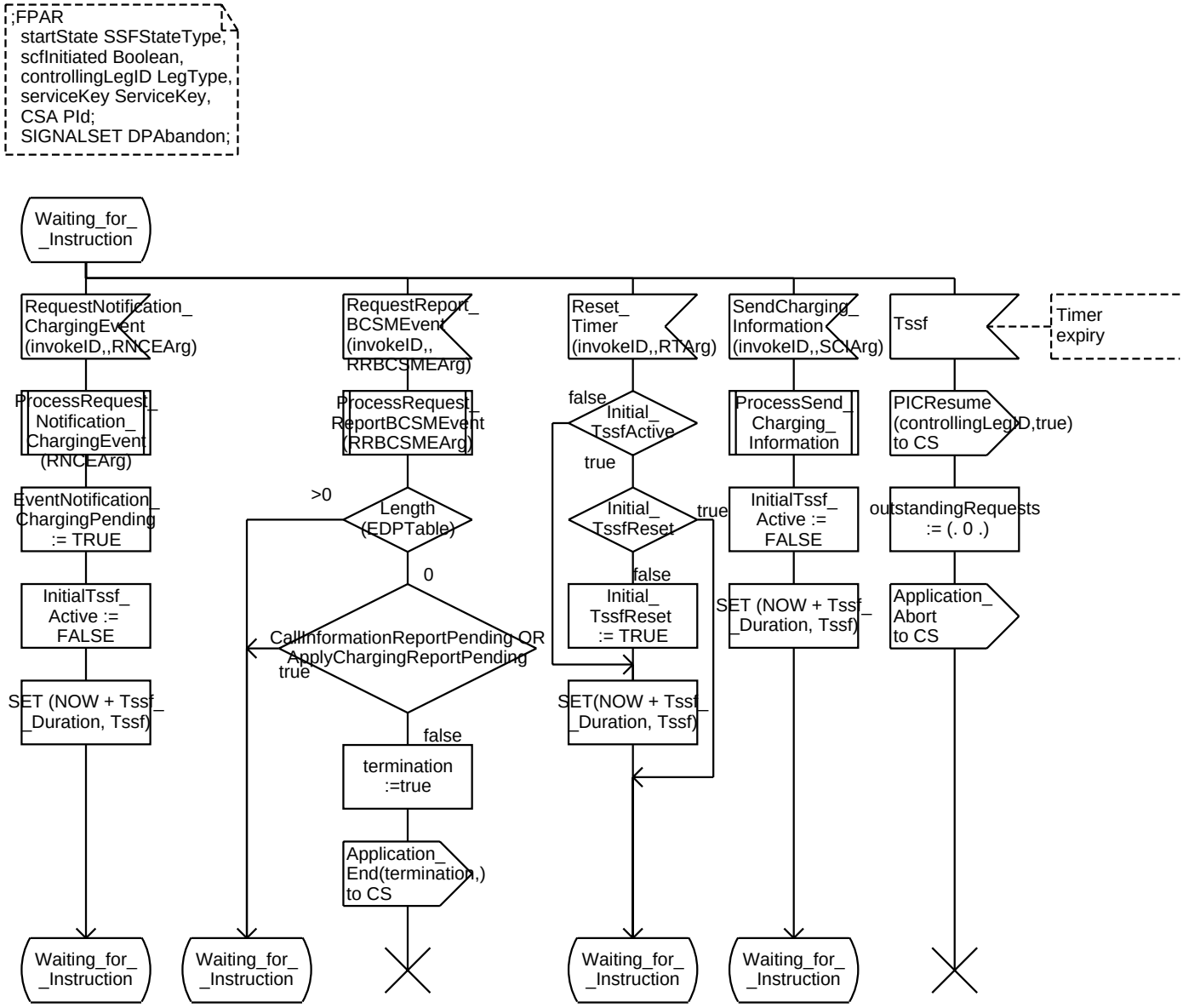
Process Type SSF_FSM 27(50)



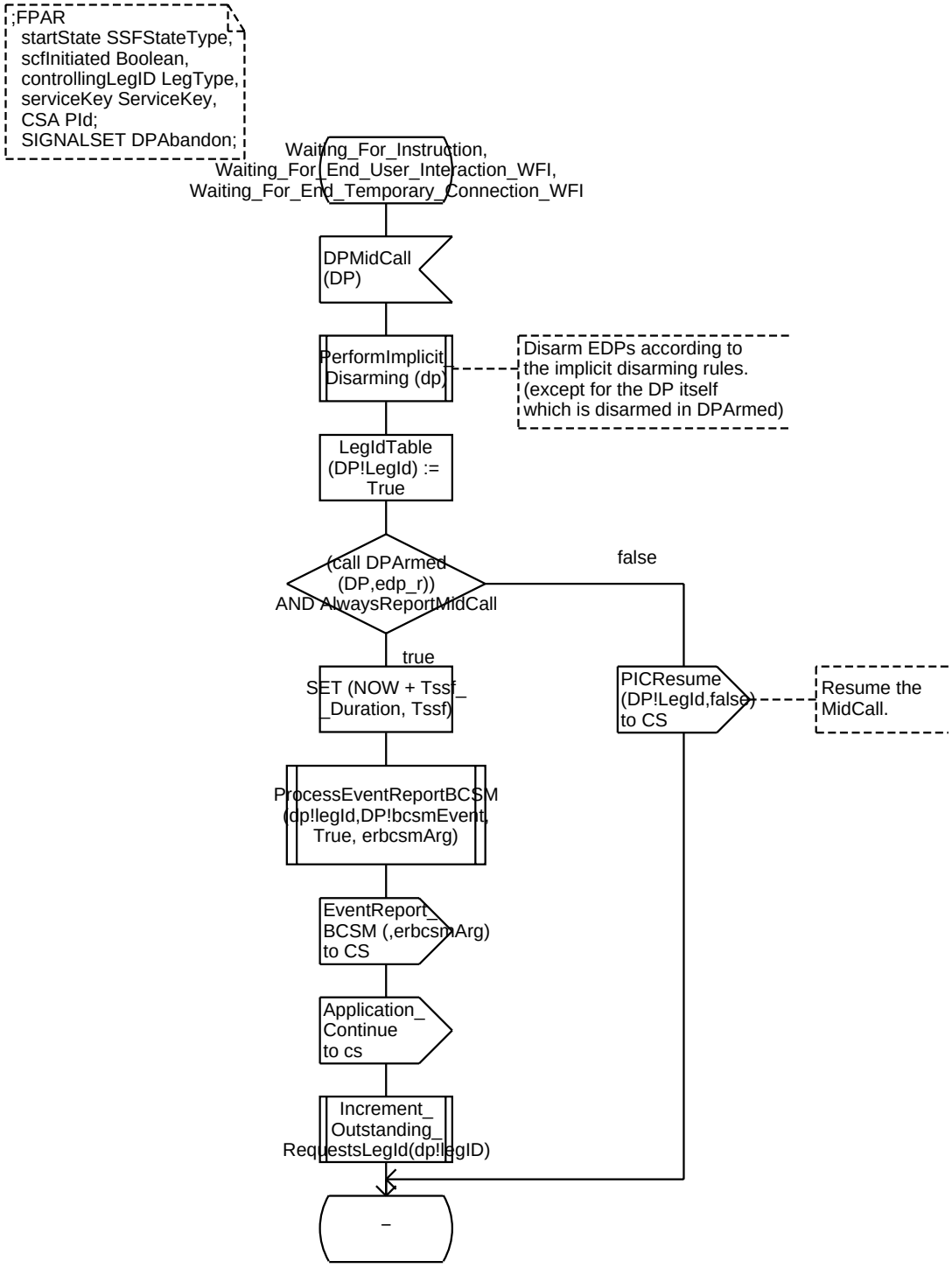

Process Type SSF_FSM
28(50)
/***** WAITING FOR INSTRUCTIONS STATE *****/
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
Waiting_for_ _Instruction
ApplyCharging (invokeID,,ACArg)
CallInformation_ Request (invokeID,,CIRqArg)
Cancel (invokeID,,CArg)
SelectFacility (invokeID,, sfArg)
Collect_ Information (invokeID,,)
Connnect (invokeID,,CoArg, newBCSM)
ProcessApply_ Charging
ProcessCall_ InformationRequest
Process_ Cancel
Process_ SelectFacility (sfArg)
Process_ Collect_ Information
Process_ Connect (CoArg, newBCSM)
ApplyCharging_ ReportPending := TRUE
CallInformation_ ReportPending := TRUE
outstandingRequests := (. 0 .)
InitialTssf_ Active := FALSE
ApplyChargingReport_ Pending := FALSE, CallInformationReport_ Pending := FALSE, EventNotificationCharging_ Pending := FALSE
Length (EDPTable)
EDPTable Length = 0 means EDPTable contains no armed DPs
>0
0
SET (NOW + Tssf_ _Duration, Tssf)
EDPTable := EmptyString
CallInformationReportPending OR ApplyChargingReportPending
true
false
outstandingRequests := (. 0 .)
Reset(Tssf)
InitialTssf_ Active := FALSE
termination :=true
InitialTssf_ Active := FALSE
SET (NOW + Tssf_ _Duration, Tssf)
Application_ End(termination,) to CS
Waiting_for_ _Instruction
Waiting_for_ _Instruction
Monitoring


Process Type SSF_FSM 29(50)
;FPAR
startState SSFStateType,
scfInitiated Boolean,
controllingLegID LegType,
serviceKey ServiceKey,
CSA PId;
SIGNALSET DPAbandon;
Waiting_for_ _Instruction
ConnectTo_ Resource (invokeID,,CTRArg)
Continnue (invokeID,)
Establish_ Temporary_ Connection (invokeID,,ETCArg)
Furnish_ Charging_ Information (invokeID,,FCIArg)
ProcessConnect_ ToResource(CTRArg)
Process_ Continue
Process_ Establish_ Temporary_ Connection
Process_ Furnish_ Charging_ Information
Tssf_Duration := Tssf_Duration4
call OustandingRequests
true
EDPTable Length = 0 means EDPTable contains no armed DPs
false
Length (EDPTable)
Waiting_for_ _Instruction
>0
0
InitialTssf_ Active := FALSE
SET (NOW + Tssf_ _Duration, Tssf)
CallInformationReportPending OR ApplyChargingReportPending
true
false
Reset(Tssf)
termination :=true
InitialTssf_ Active := FALSE
Application_ End(termination,) to CS
Waiting_For_End_ _User_Interaction_ _WFI
Monitoring
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. */

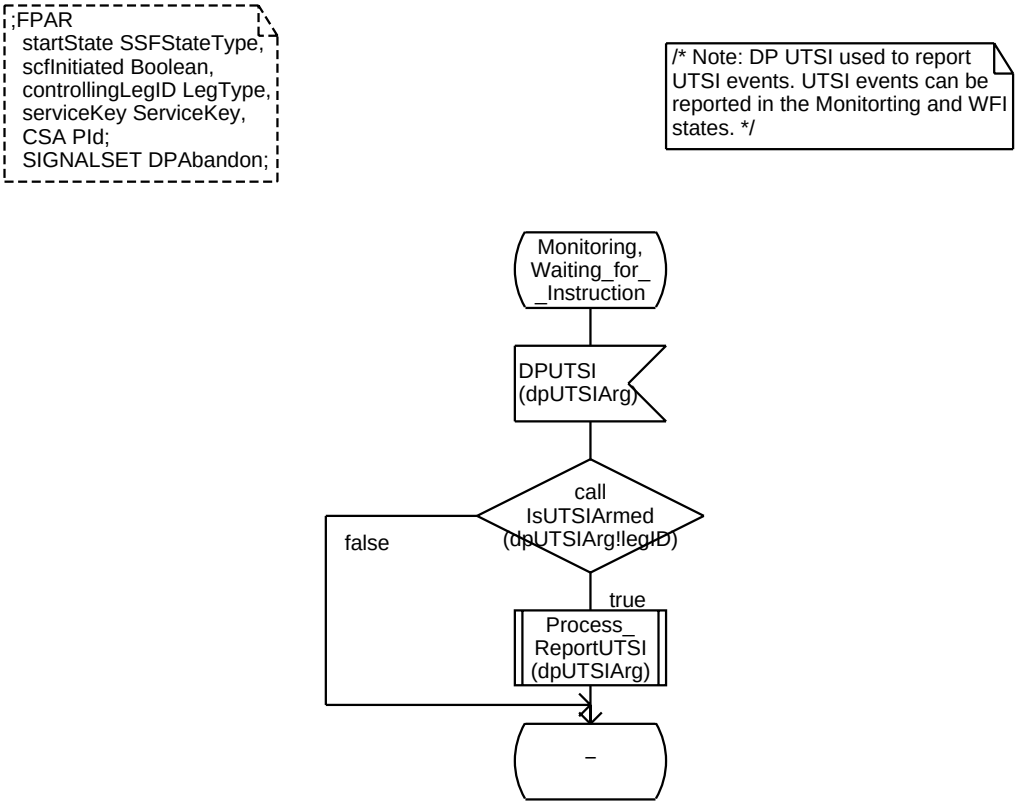
Process Type SSF_FSM 30(50)



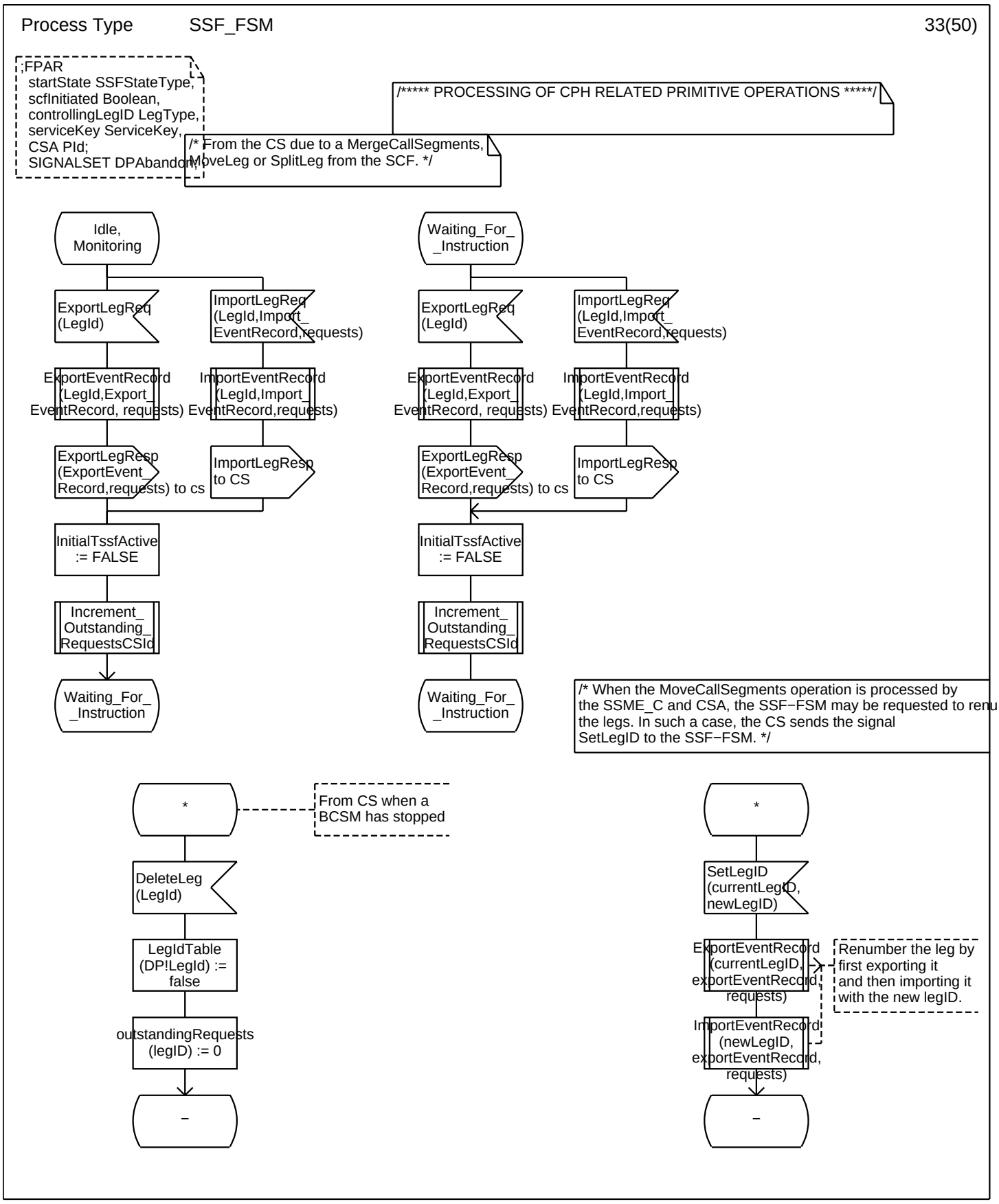
Process Type SSF_FSM 31(50)



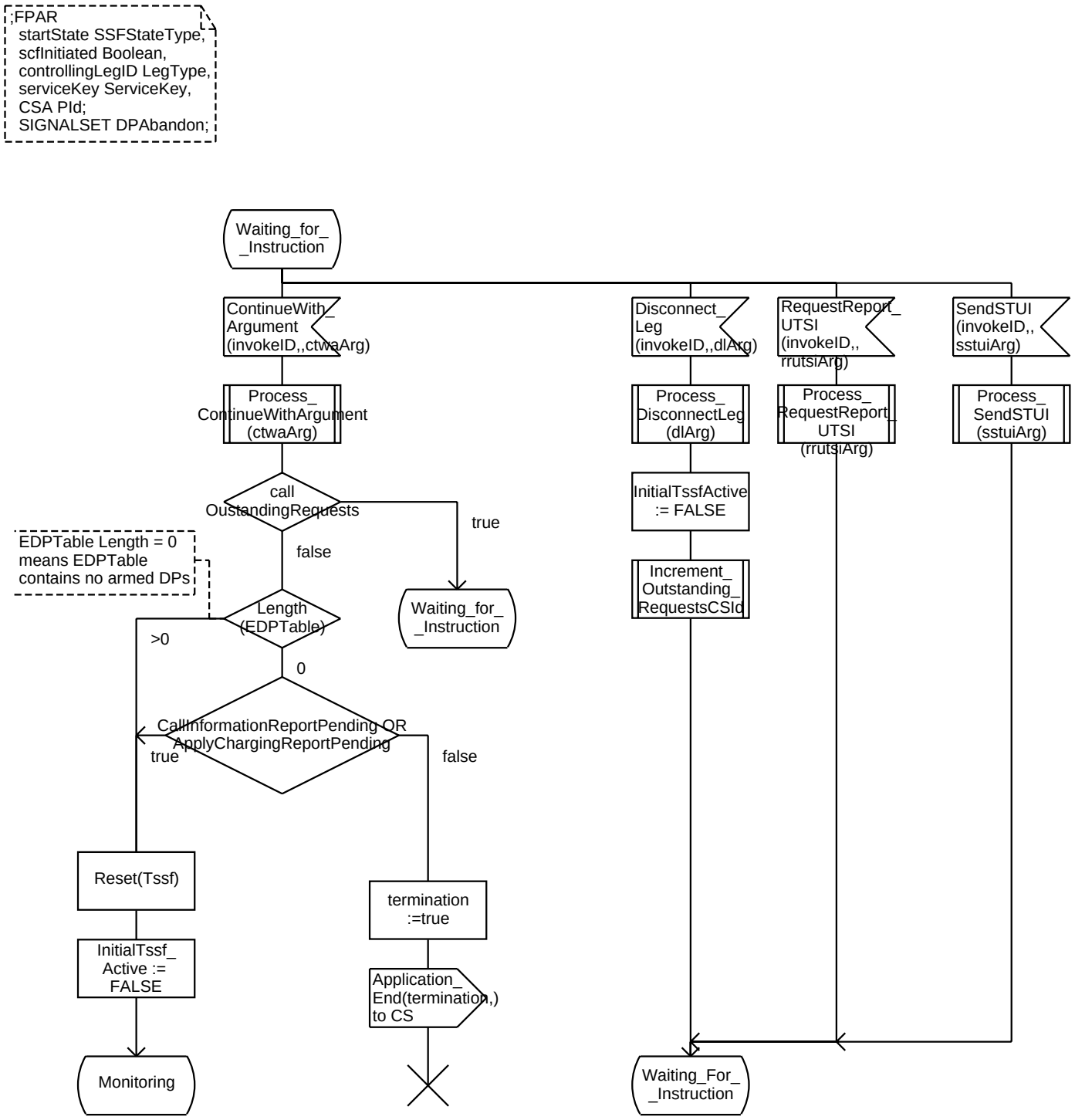
Process Type SSF_FSM 32(50)



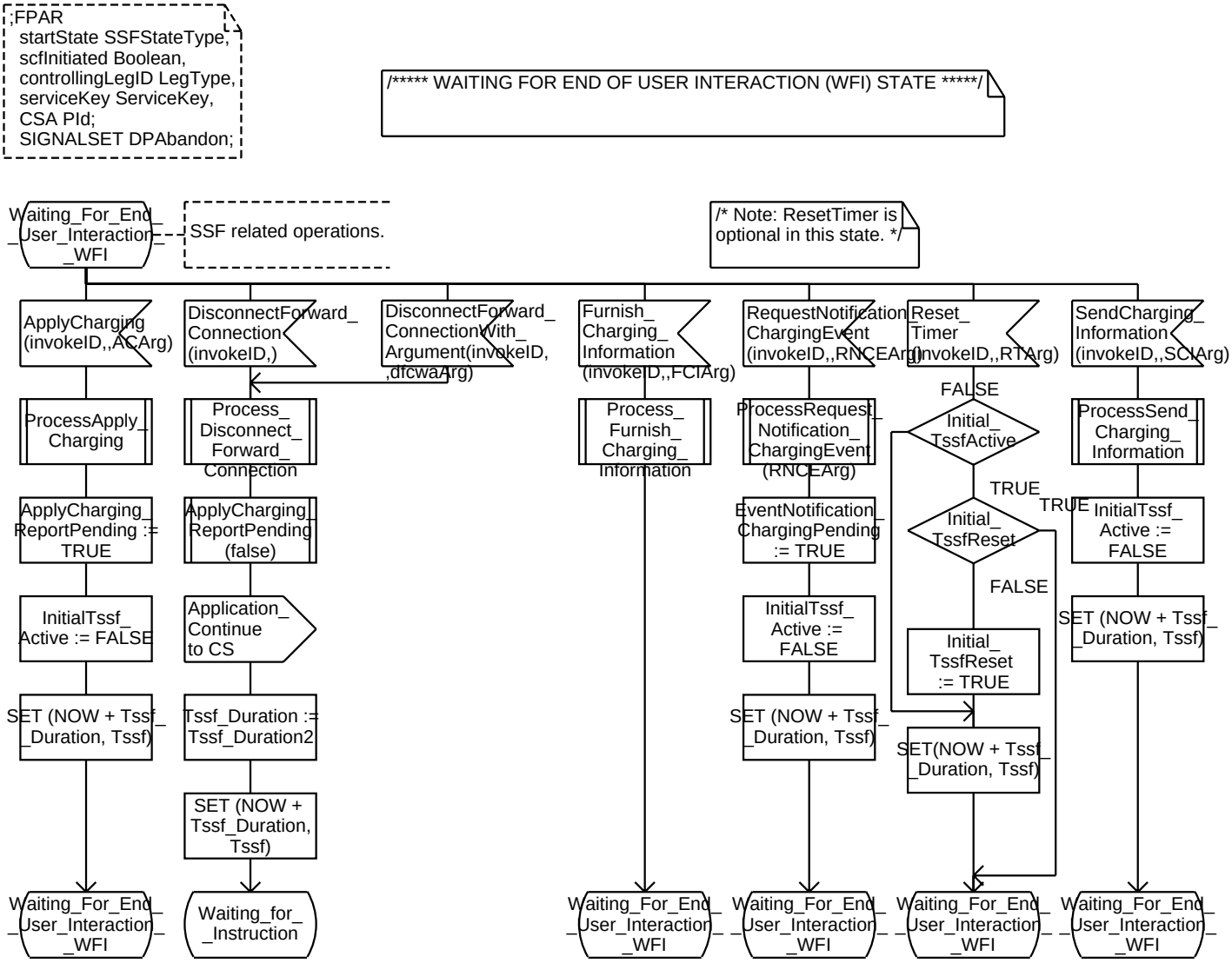
Process Type SSF_FSM 33(50)



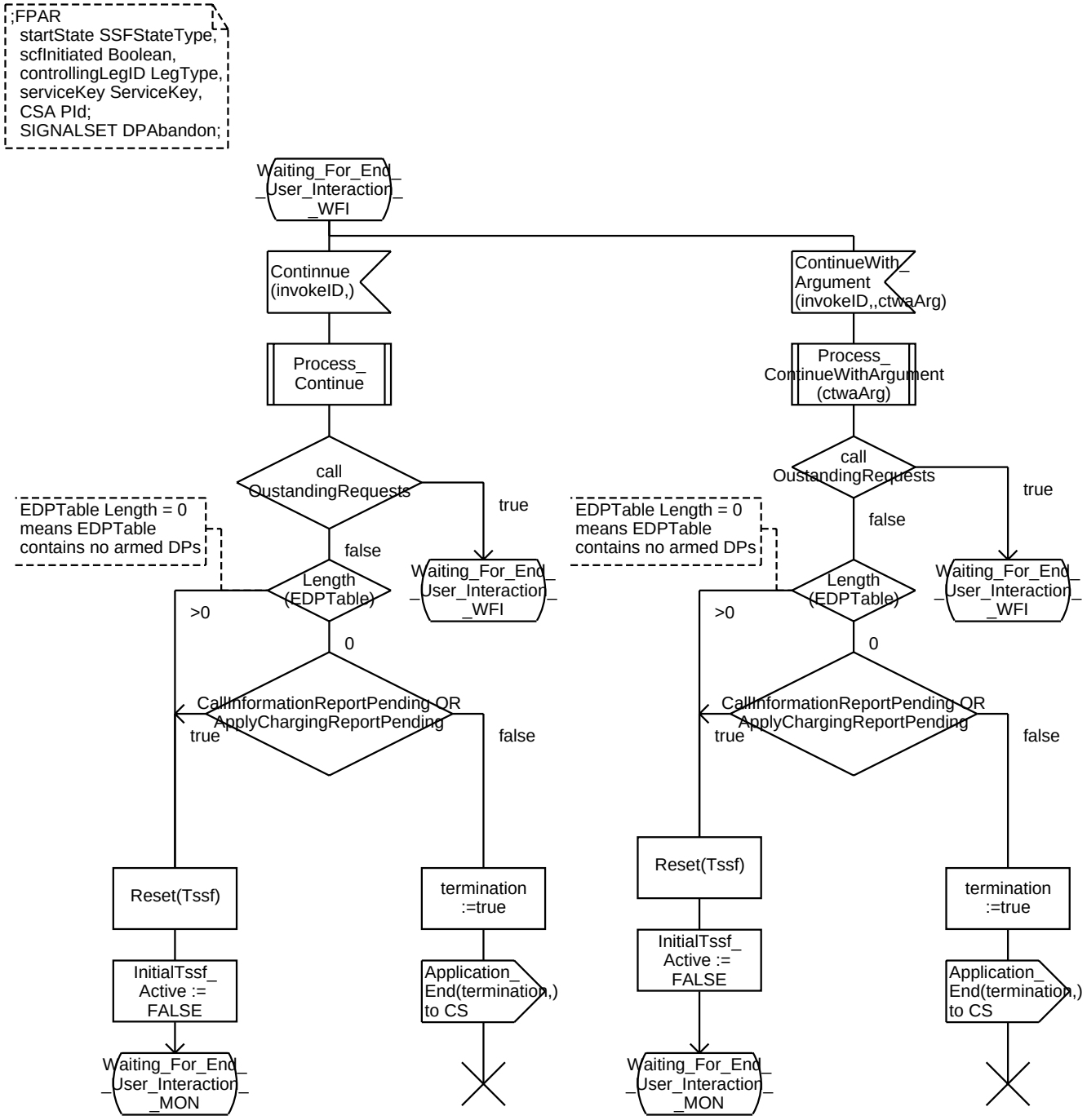
Process Type SSF_FSM 34(50)



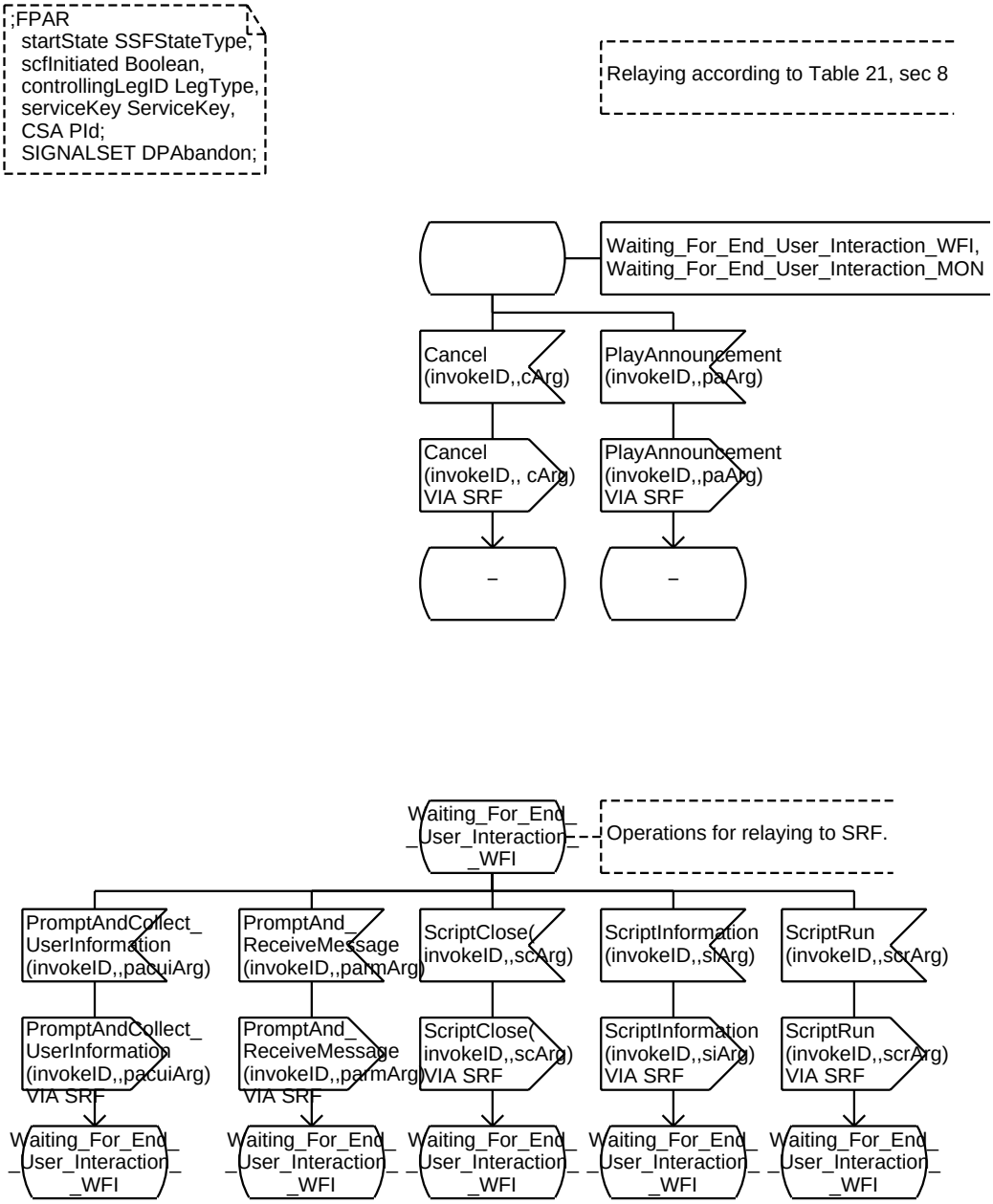
Process Type SSF_FSM 35(50)



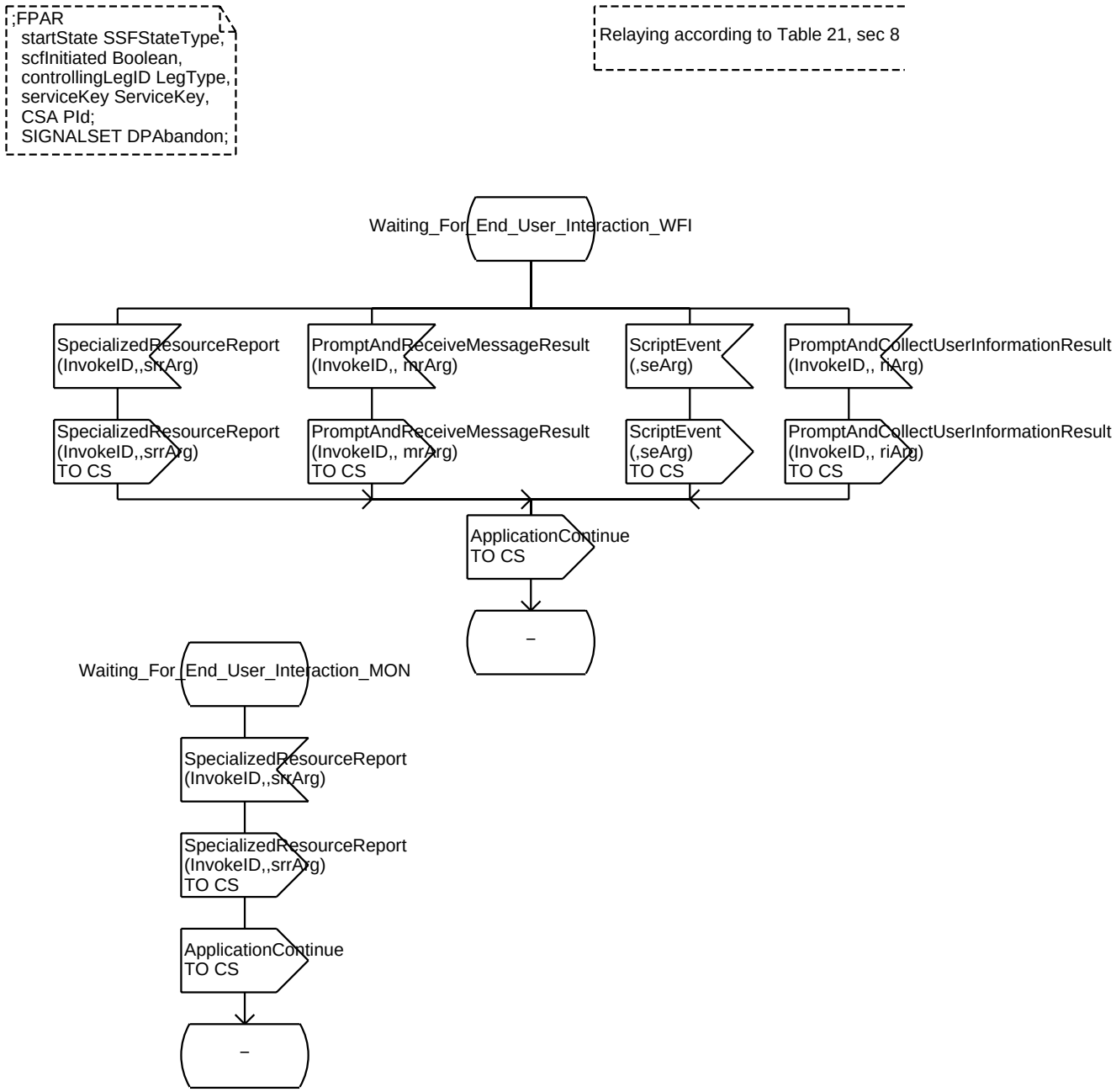
Process Type SSF_FSM 36(50)



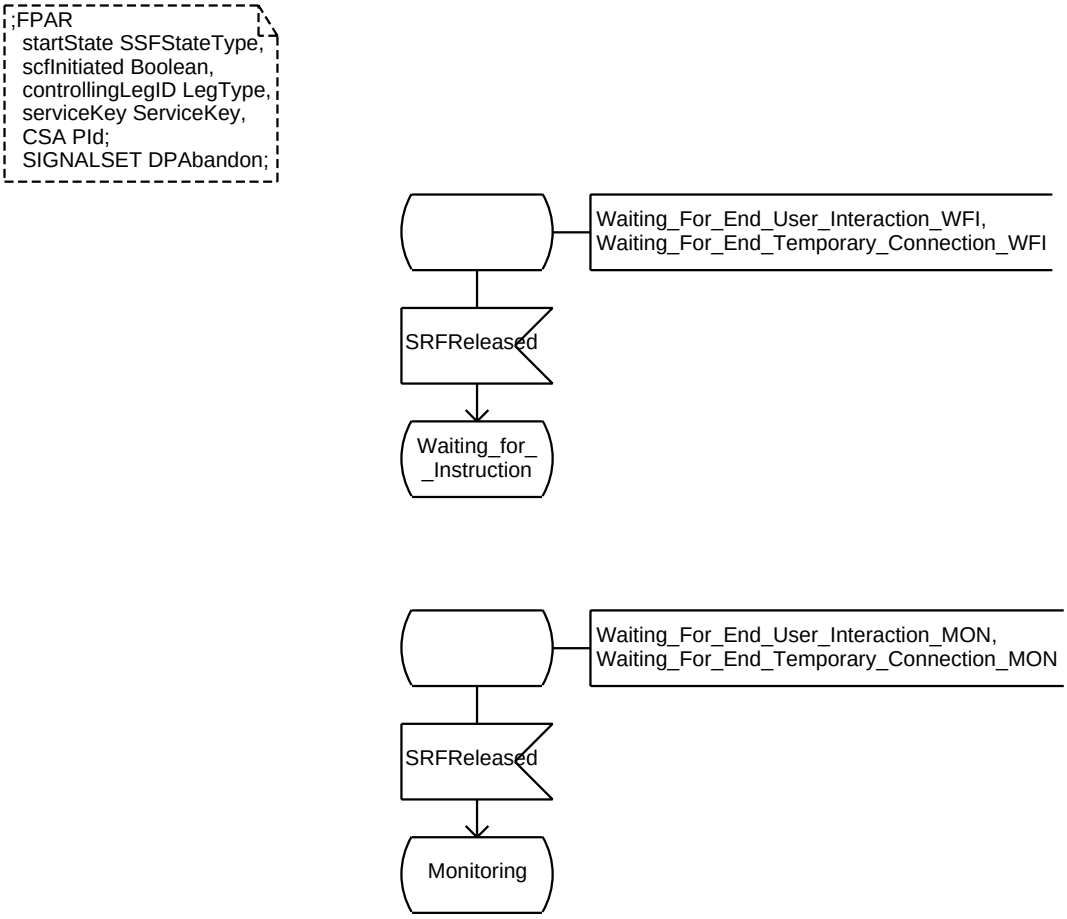
Process Type SSF_FSM 37(50)



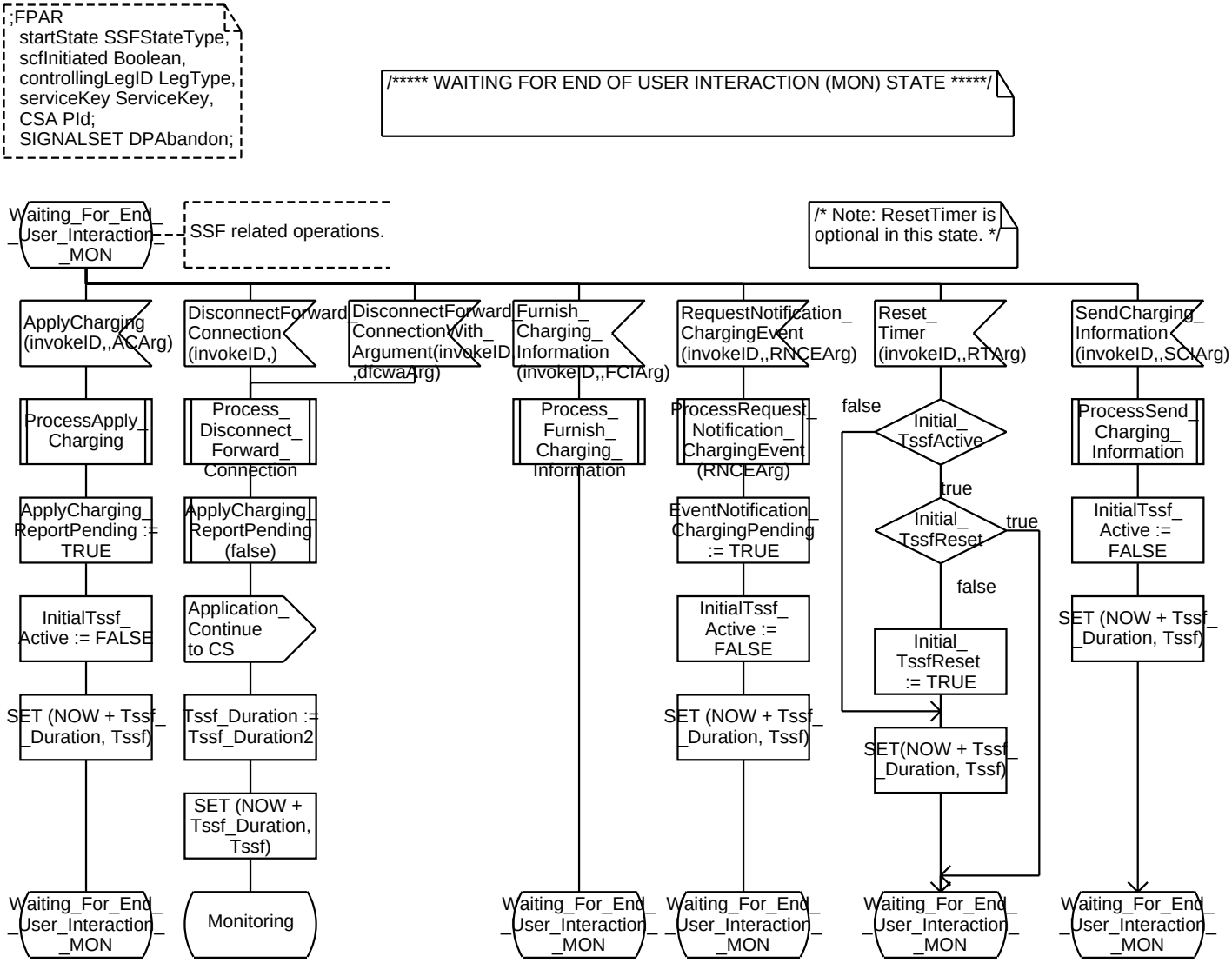
Process Type SSF_FSM 38(50)



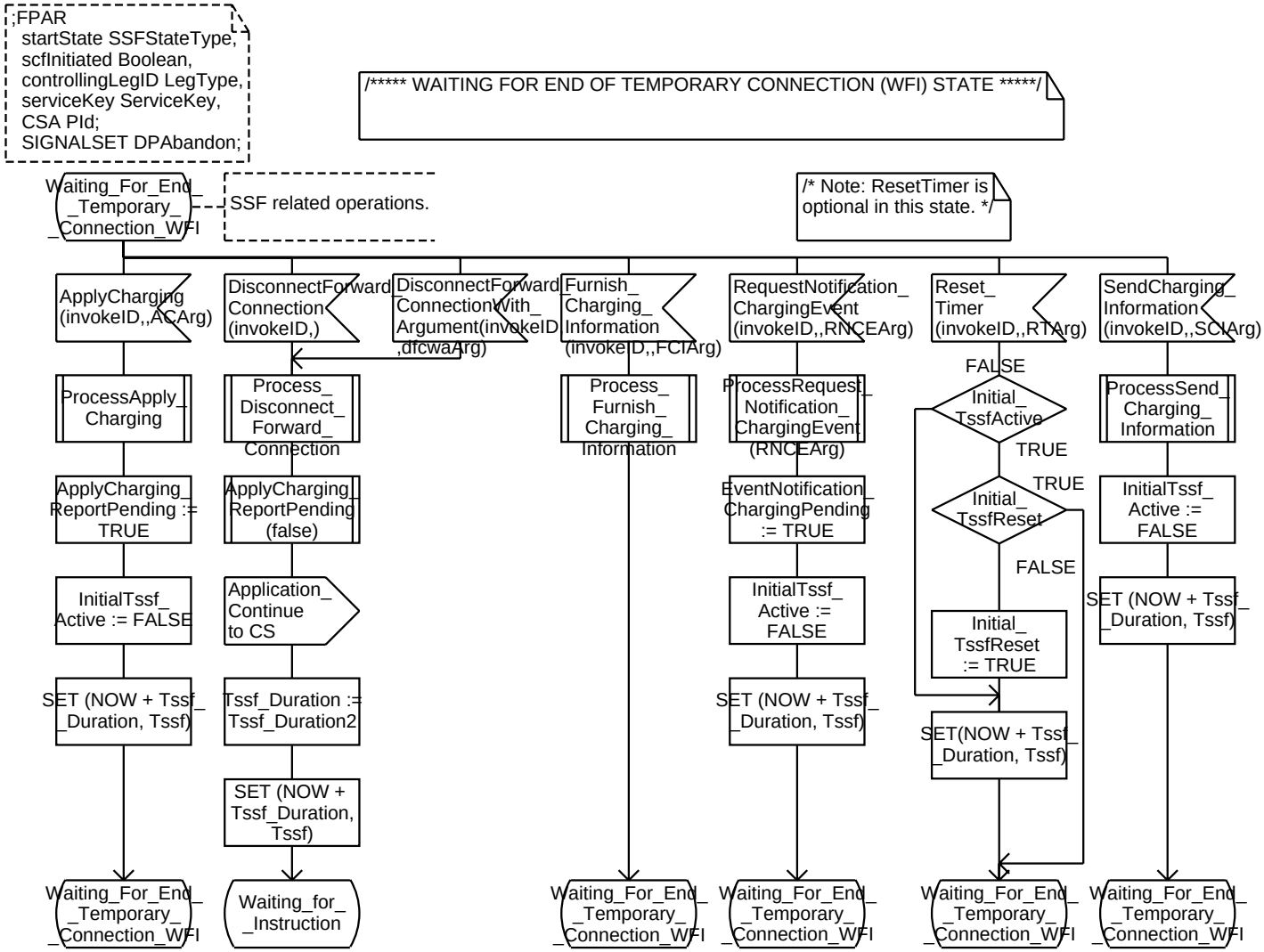
Process Type SSF_FSM 39(50)



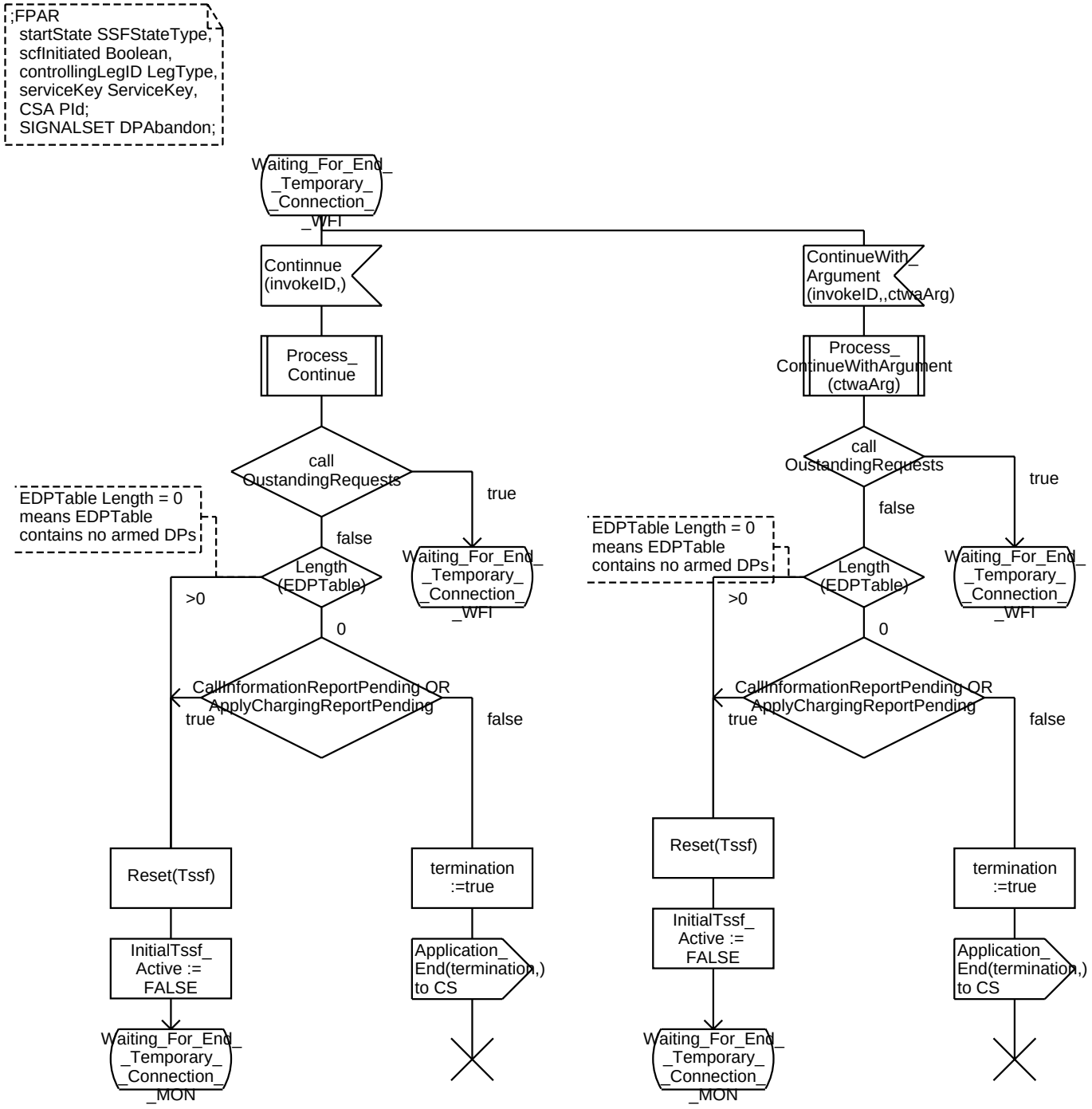
Process Type SSF_FSM 40(50)



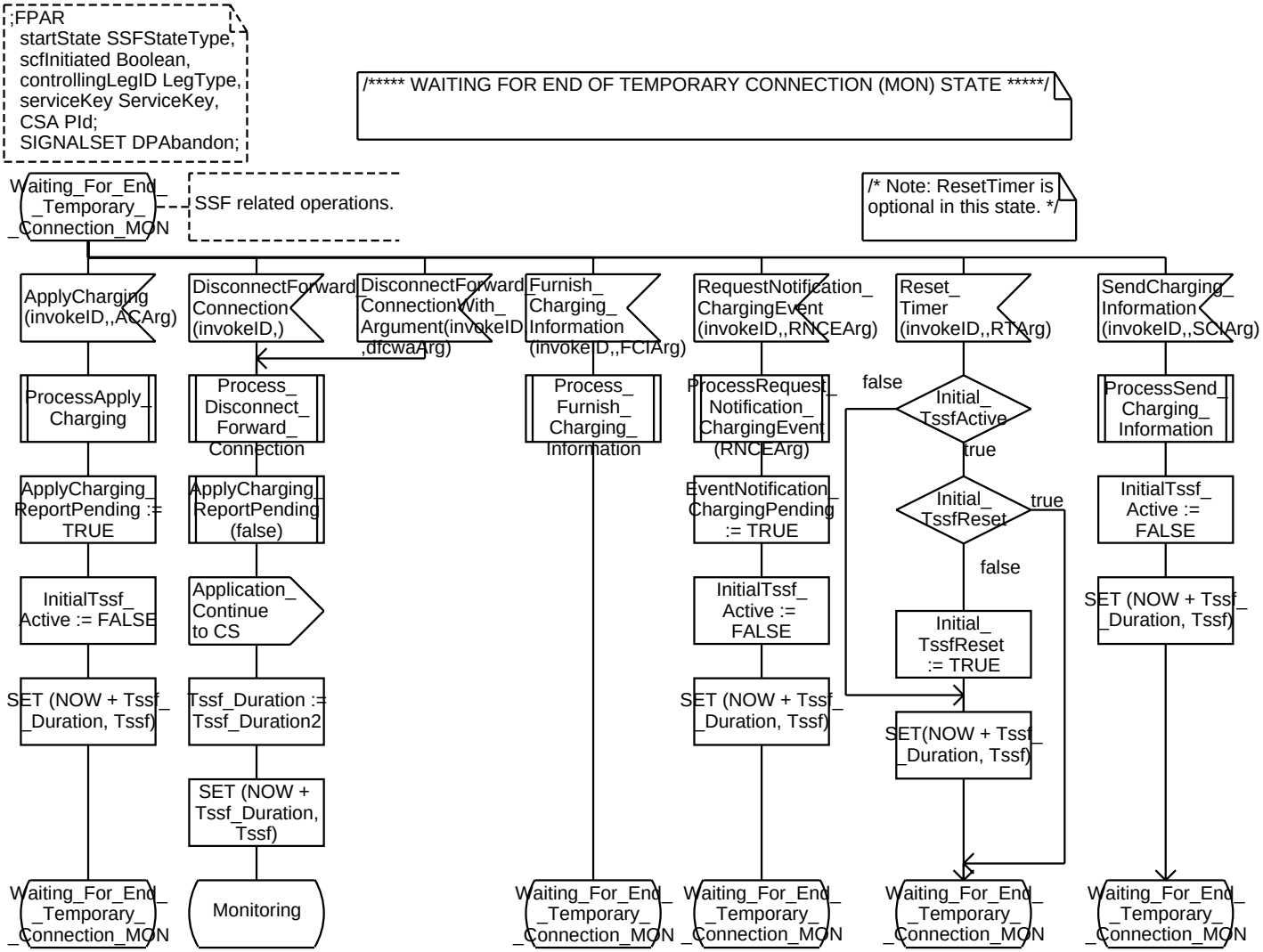
Process Type SSF_FSM 41(50)



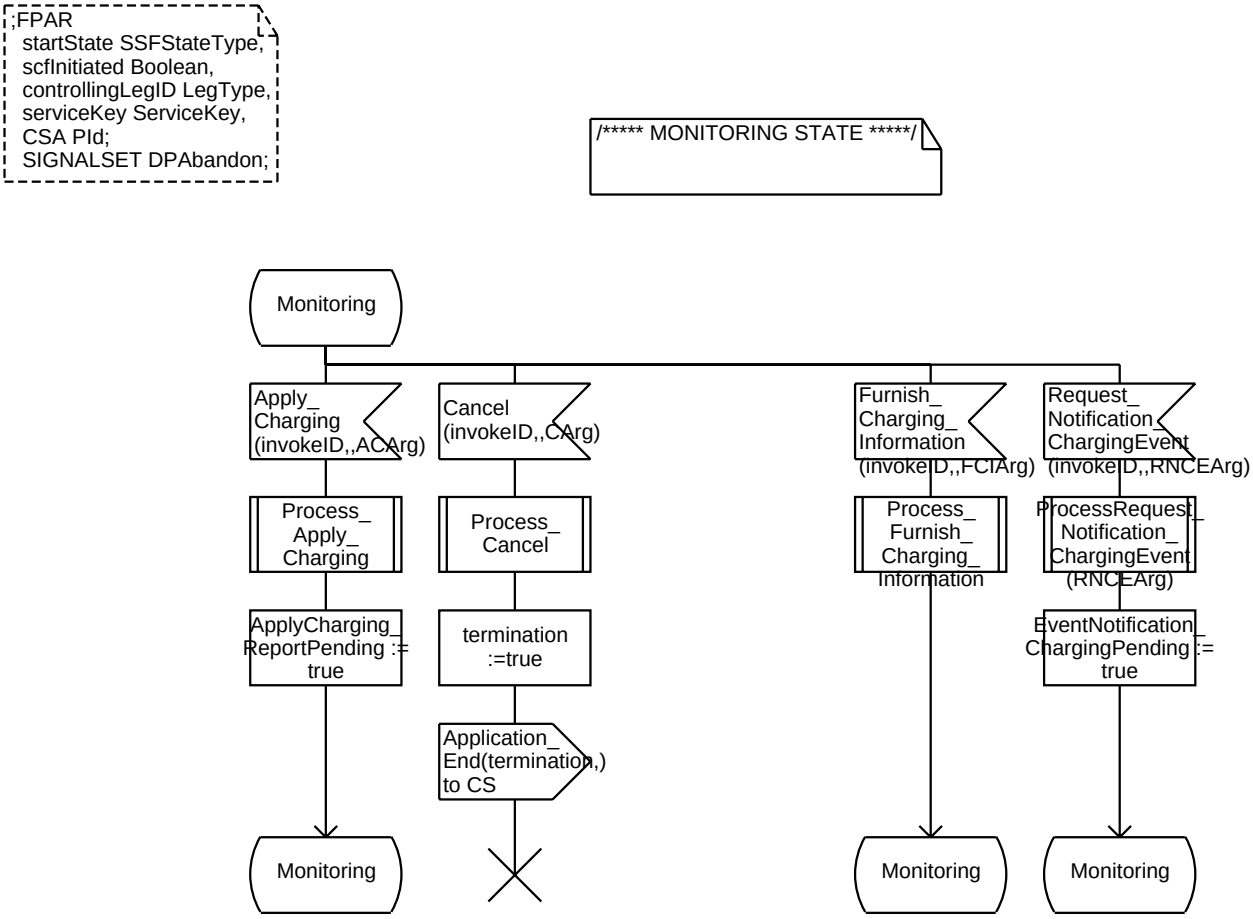
Process Type SSF_FSM 42(50)



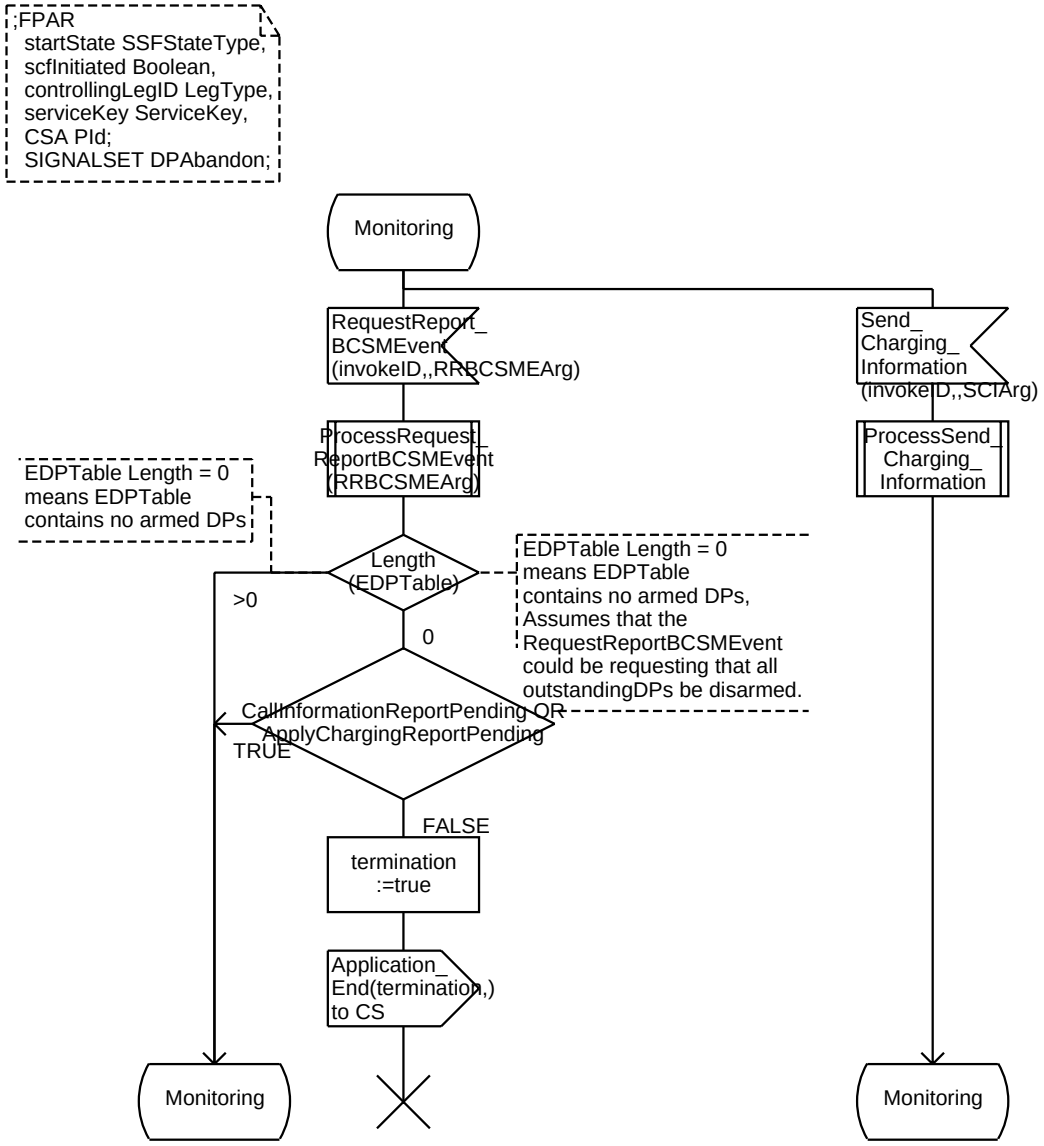
Process Type SSF_FSM 43(50)



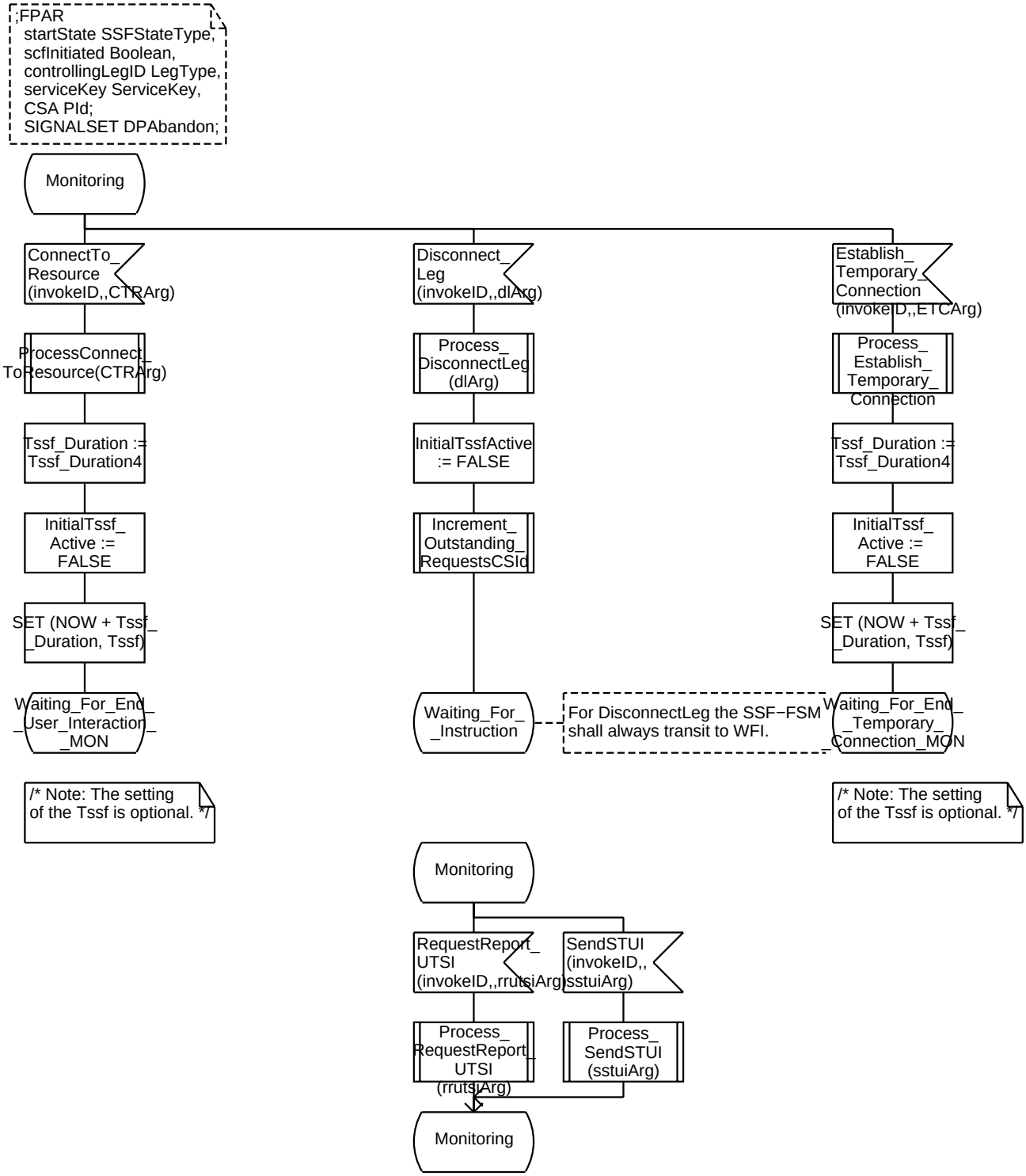
Process Type SSF_FSM 44(50)



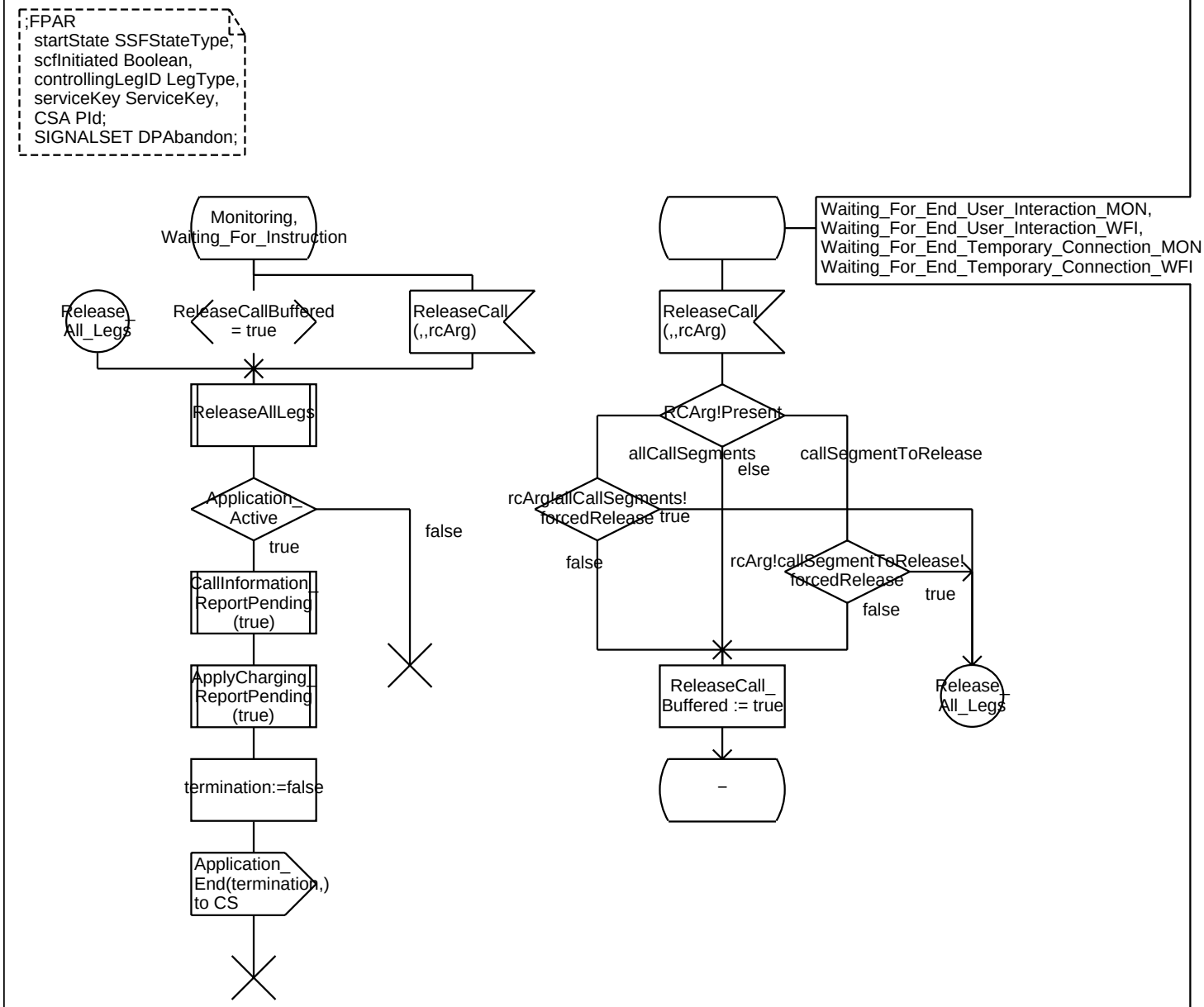
Process Type SSF_FSM 45(50)



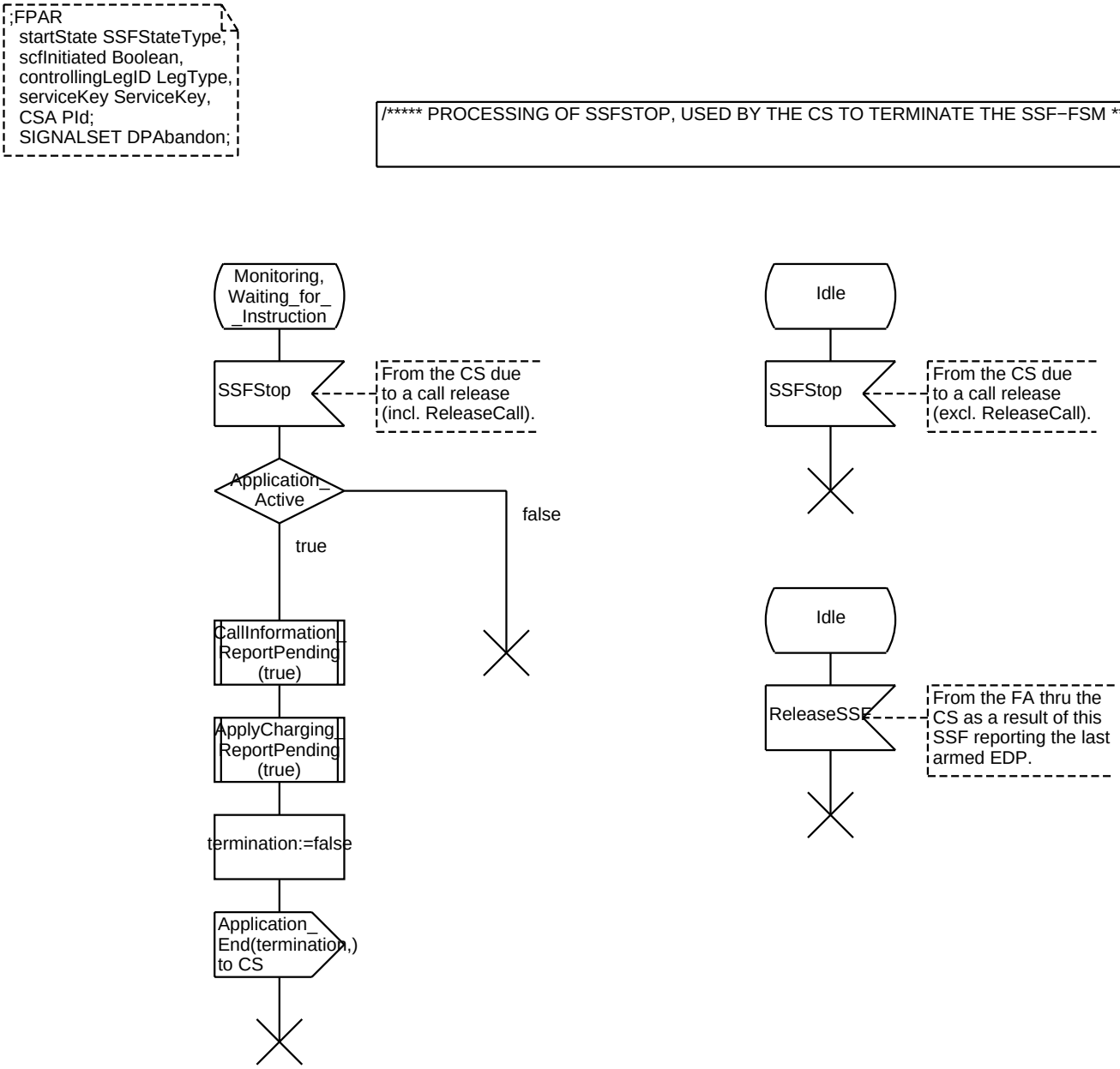
Process Type SSF_FSM 46(50)



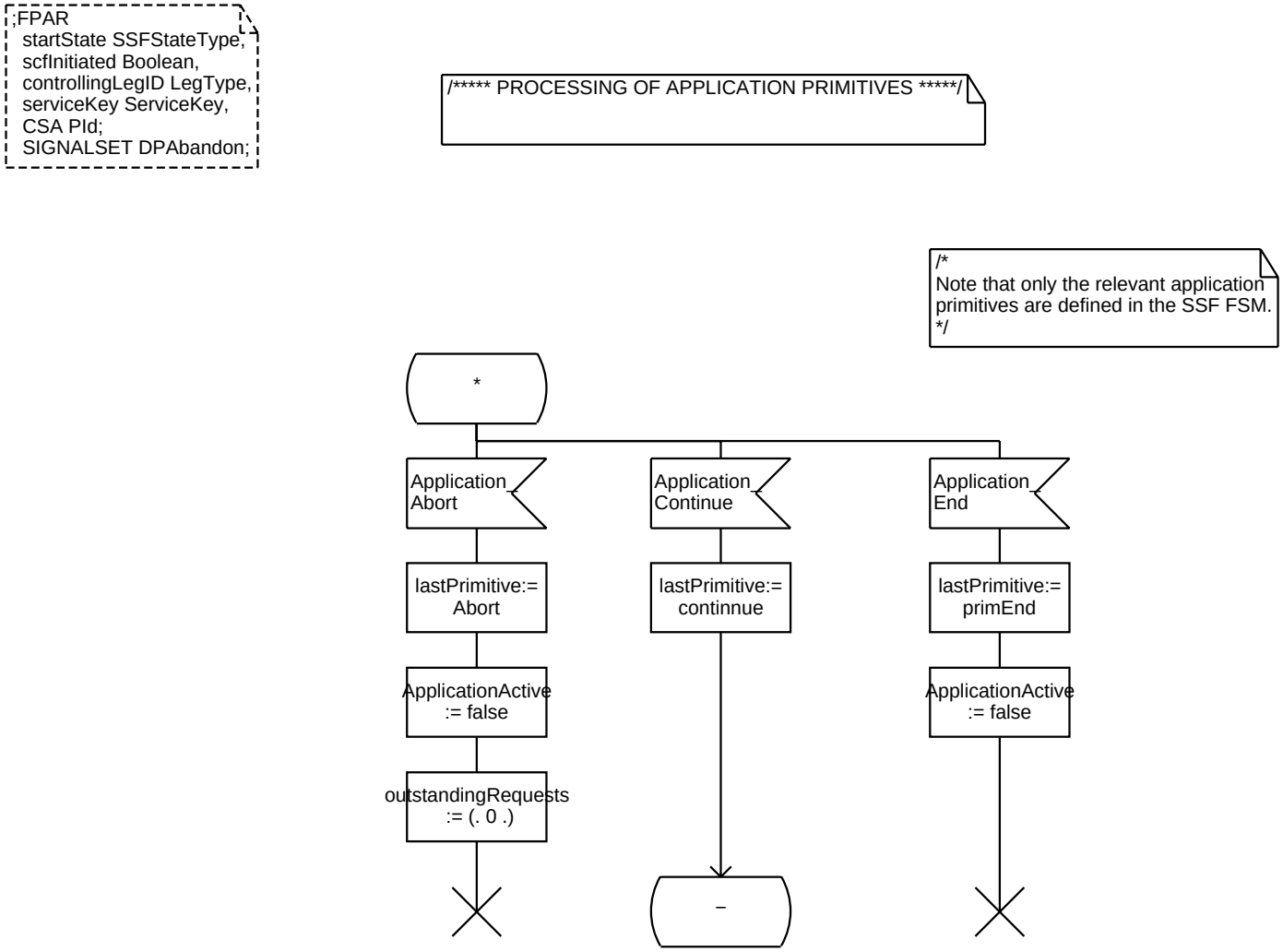
Process Type SSF_FSM 47(50)



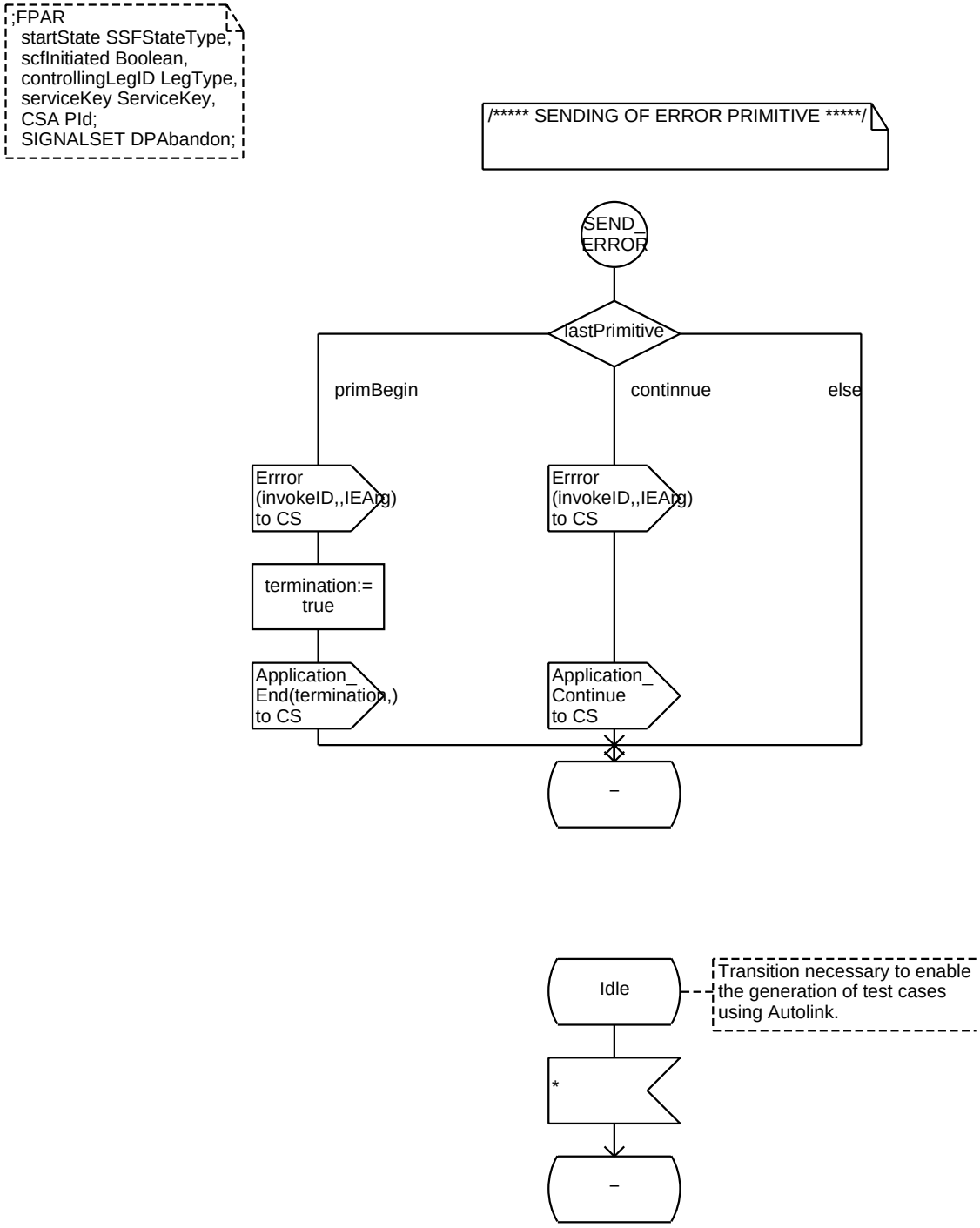
Process Type SSF_FSM 48(50)



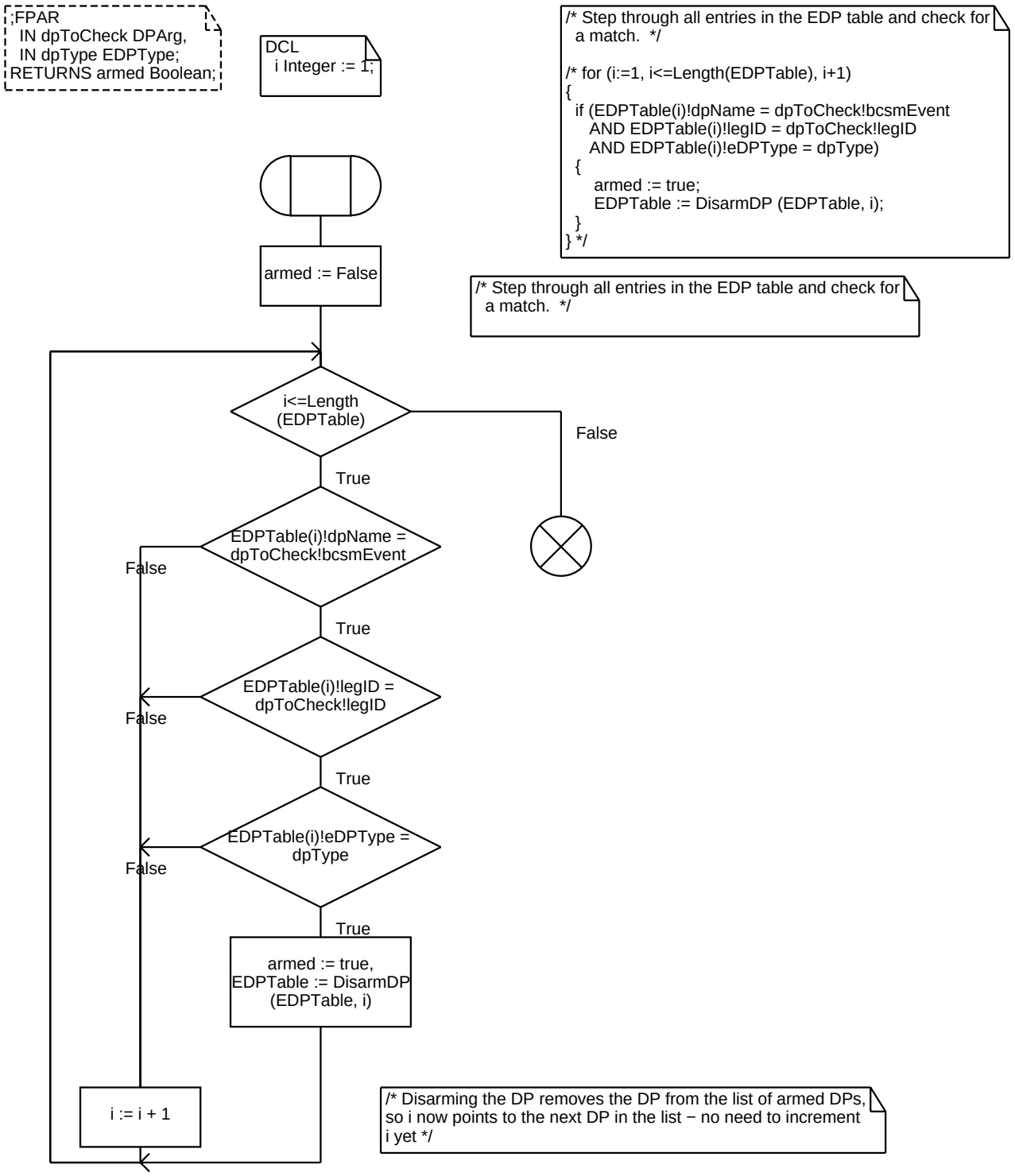
Process Type SSF_FSM 49(50)



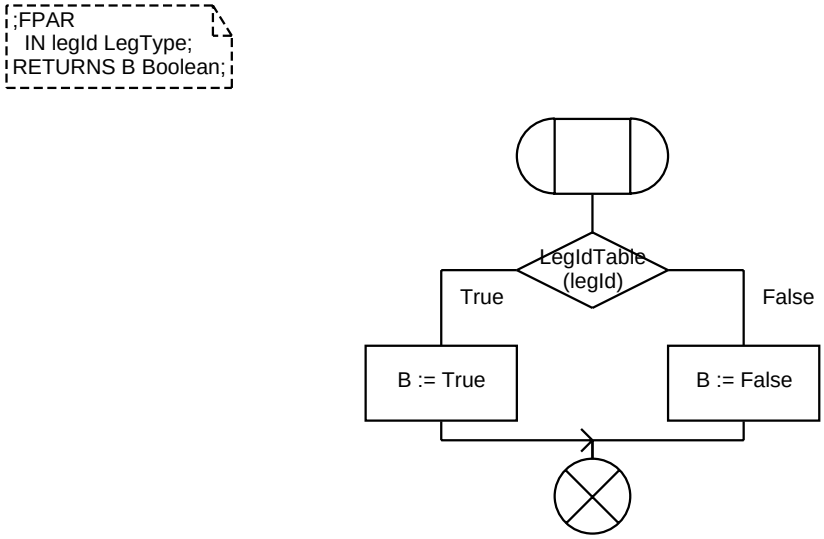
Process Type SSF_FSM 50(50)



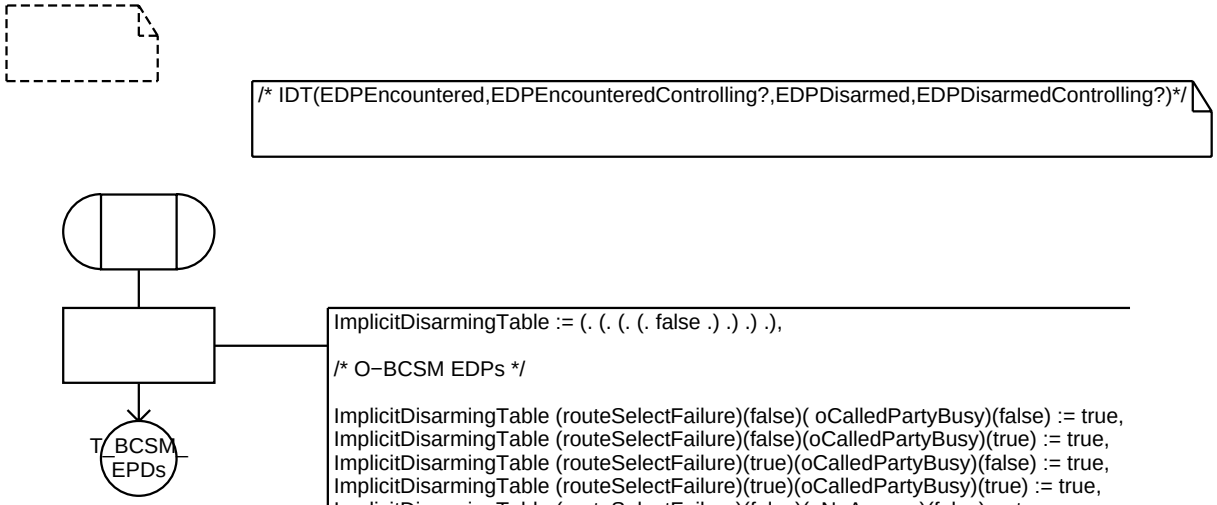
Procedure <<Process Type SSF_FSM>> DPArmed 1(1)



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

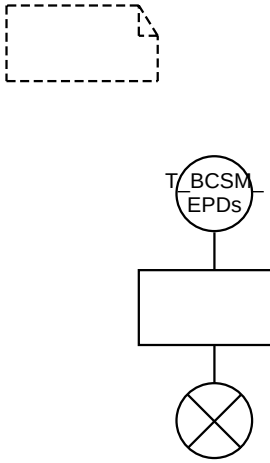


Procedure InitializeImplicitDisarmingTable 1(2)



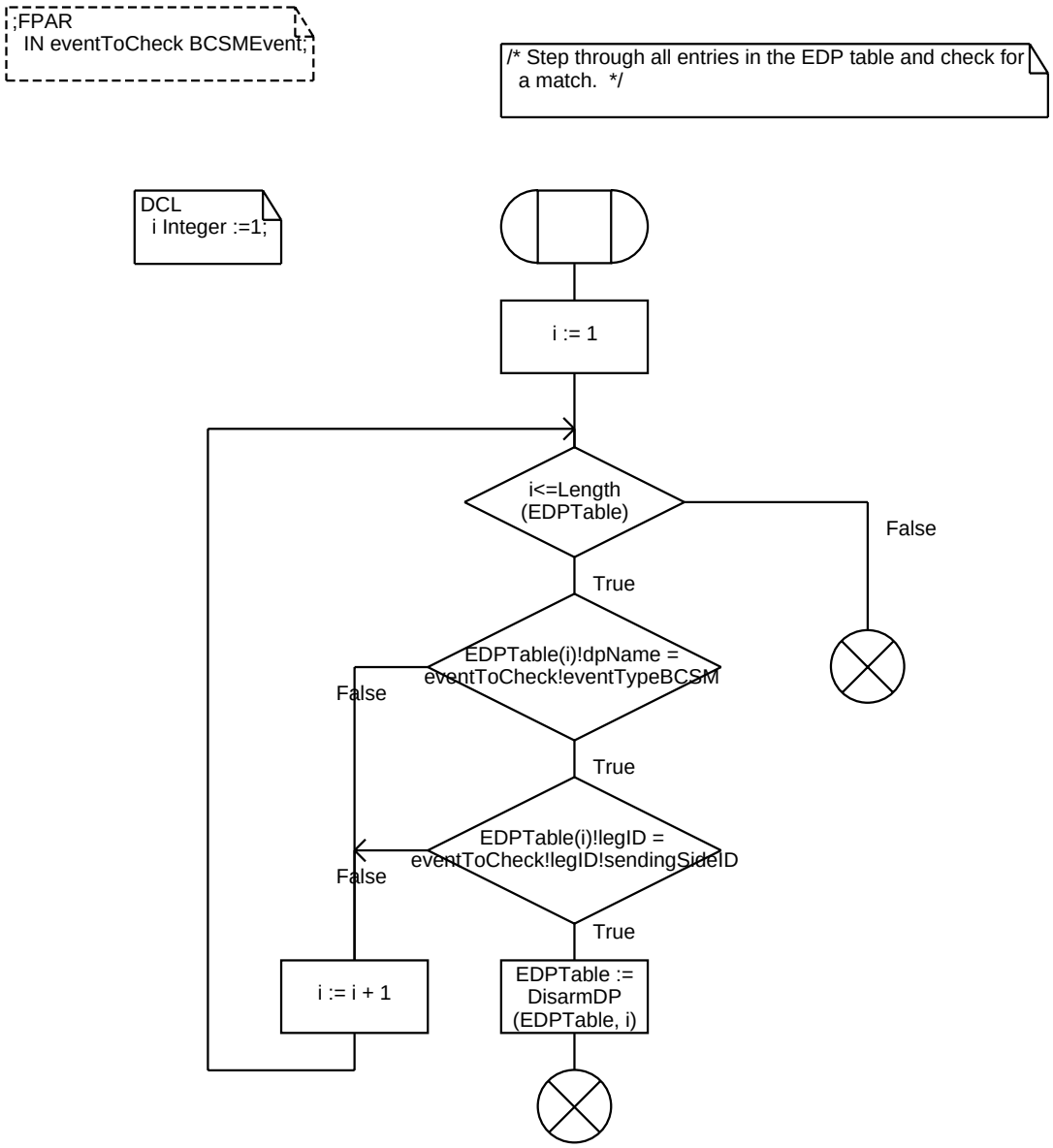

```
ImplicitDisarmingTable := (. (. (. (. false .) .) .) .),

/* O−BCSM EDPs */

ImplicitDisarmingTable (routeSelectFailure)(false)( oCalledPartyBusy)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oCalledPartyBusy)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oCalledPartyBusy)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oCalledPartyBusy)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oNoAnswer)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oNoAnswer)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oNoAnswer)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oNoAnswer)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oAnswer)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oAnswer)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oAnswer)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oAnswer)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oMidCall)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oMidCall)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oDisconnect)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oDisconnect)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oTermSeized)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oTermSeized)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oTermSeized)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oTermSeized)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oSuspend)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oSuspend)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oSuspend)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oSuspend)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oReAnswer)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(false)(oReAnswer)(true) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oReAnswer)(false) := true,
ImplicitDisarmingTable (routeSelectFailure)(true)(oReAnswer)(true) := true,

ImplicitDisarmingTable (oCalledPartyBusy)(false)(routeSelectFailure)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(routeSelectFailure)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(routeSelectFailure)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(routeSelectFailure)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oNoAnswer)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oNoAnswer)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oNoAnswer)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oNoAnswer)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oAnswer)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oAnswer)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oAnswer)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oAnswer)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oMidCall)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oMidCall)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oDisconnect)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oDisconnect)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oTermSeized)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oTermSeized)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oTermSeized)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oTermSeized)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oSuspend)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oSuspend)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oSuspend)(false) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(true)(oSuspend)(true) := true,
ImplicitDisarmingTable (oCalledPartyBusy)(false)(oReAnswer)(false) := true,
```

Procedure InitializeImplicitDisarmingTable 2(2)

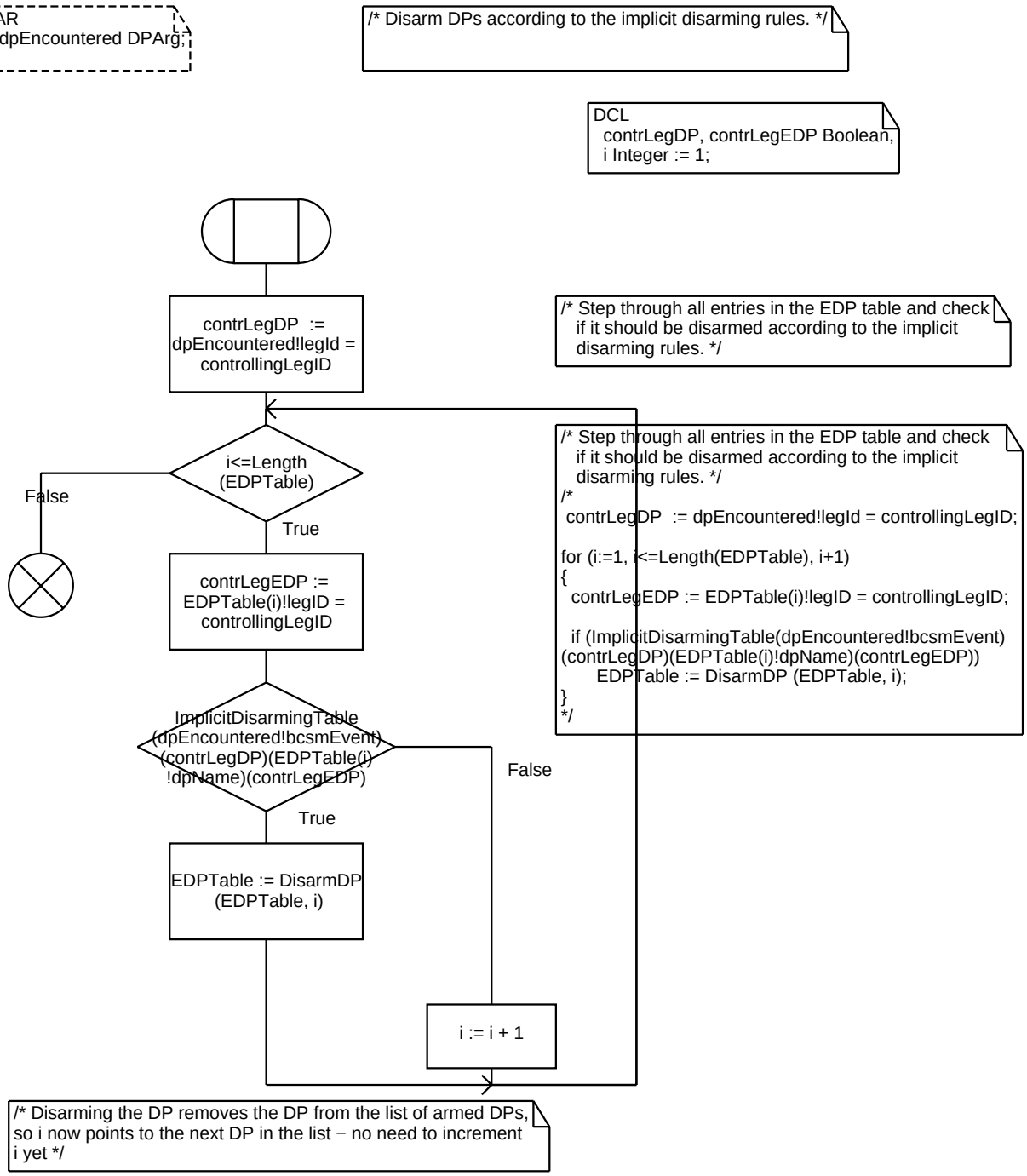



```
/* T−BCSM EDPs */

ImplicitDisarmingTable (tBusy)(false)(termAttemptAuthorized)(false) := true,
ImplicitDisarmingTable (tBusy)(false)(termAttemptAuthorized)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(termAttemptAuthorized)(false) := true,
ImplicitDisarmingTable (tBusy)(true)(termAttemptAuthorized)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(tNoAnswer)(false) := true,
ImplicitDisarmingTable (tBusy)(false)(tNoAnswer)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(tNoAnswer)(false) := true,
ImplicitDisarmingTable (tBusy)(true)(tNoAnswer)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(tAnswer)(false) := true,
ImplicitDisarmingTable (tBusy)(false)(tAnswer)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(tAnswer)(false) := true,
ImplicitDisarmingTable (tBusy)(true)(tAnswer)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(tMidCall)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(tMidCall)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(tDisconnect)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(tDisconnect)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(tSuspend)(false) := true,
ImplicitDisarmingTable (tBusy)(false)(tSuspend)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(tSuspend)(false) := true,
ImplicitDisarmingTable (tBusy)(true)(tSuspend)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(tReAnswer)(false) := true,
ImplicitDisarmingTable (tBusy)(false)(tReAnswer)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(tReAnswer)(false) := true,
ImplicitDisarmingTable (tBusy)(true)(tReAnswer)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(facilitySelectedAndAvailable)(false) := true,
ImplicitDisarmingTable (tBusy)(false)(facilitySelectedAndAvailable)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(facilitySelectedAndAvailable)(false) := true,
ImplicitDisarmingTable (tBusy)(true)(facilitySelectedAndAvailable)(true) := true,
ImplicitDisarmingTable (tBusy)(false)(callAccepted)(false) := true,
ImplicitDisarmingTable (tBusy)(false)(callAccepted)(true) := true,
ImplicitDisarmingTable (tBusy)(true)(callAccepted)(false) := true,
ImplicitDisarmingTable (tBusy)(true)(callAccepted)(true) := true,

ImplicitDisarmingTable (tNoAnswer)(false)(termAttemptAuthorized)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(termAttemptAuthorized)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(termAttemptAuthorized)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(termAttemptAuthorized)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tBusy)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tBusy)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tBusy)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tBusy)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tAnswer)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tAnswer)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tAnswer)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tAnswer)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tMidCall)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tMidCall)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tDisconnect)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tDisconnect)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tSuspend)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tSuspend)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tSuspend)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tSuspend)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tReAnswer)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(tReAnswer)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tReAnswer)(false) := true,
ImplicitDisarmingTable (tNoAnswer)(true)(tReAnswer)(true) := true,
ImplicitDisarmingTable (tNoAnswer)(false)(facilitySelectedAndAvailable)(false) := true,
```

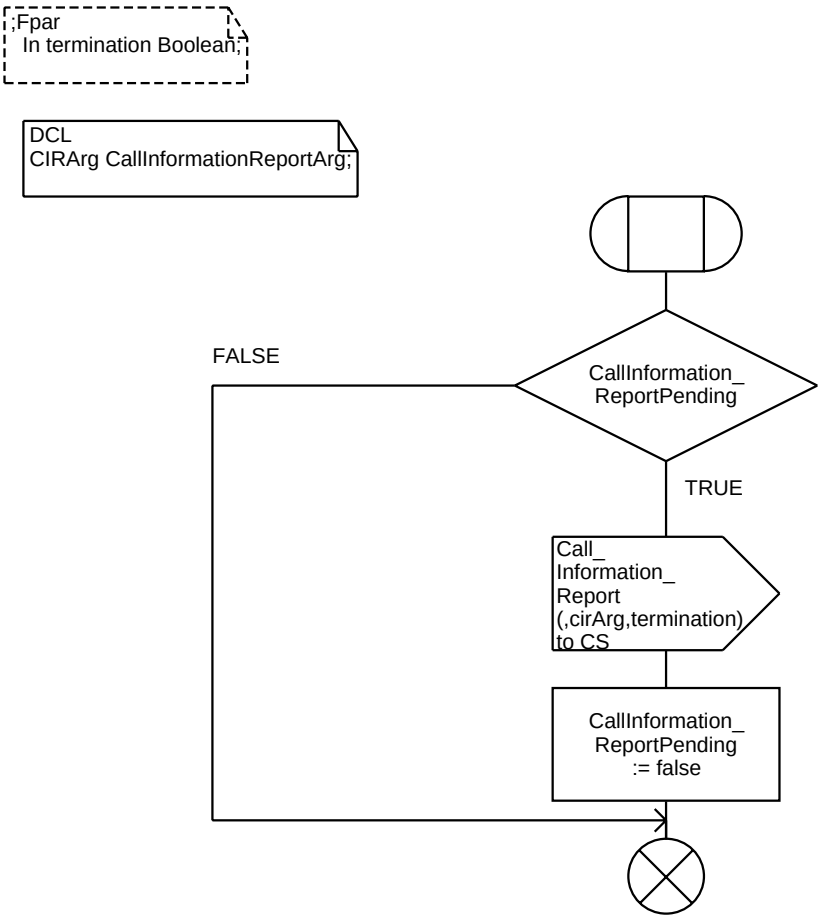
Procedure <<Process Type SSF_FSM>> RemoveDP 1(1)



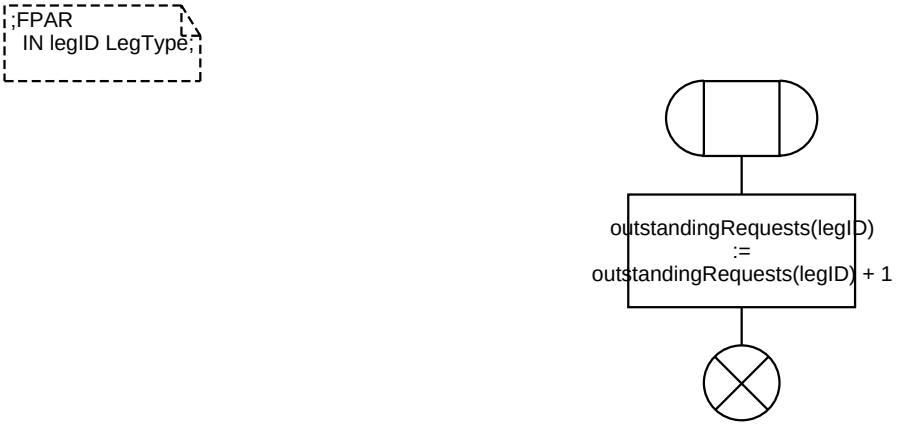
Procedure PerformImplicitDisarming 1(1)



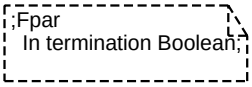
Procedure CallInformationReportPending 1(1)

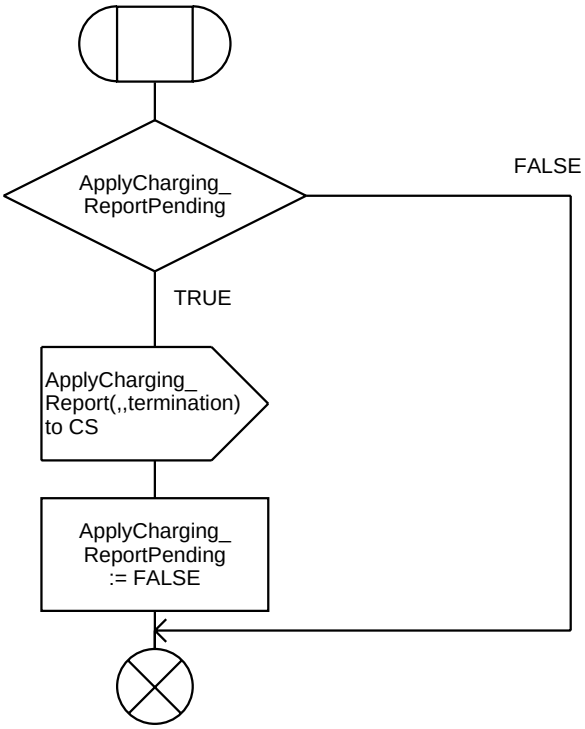


Procedure IncrementOutstandingRequestsLegId 1(1)

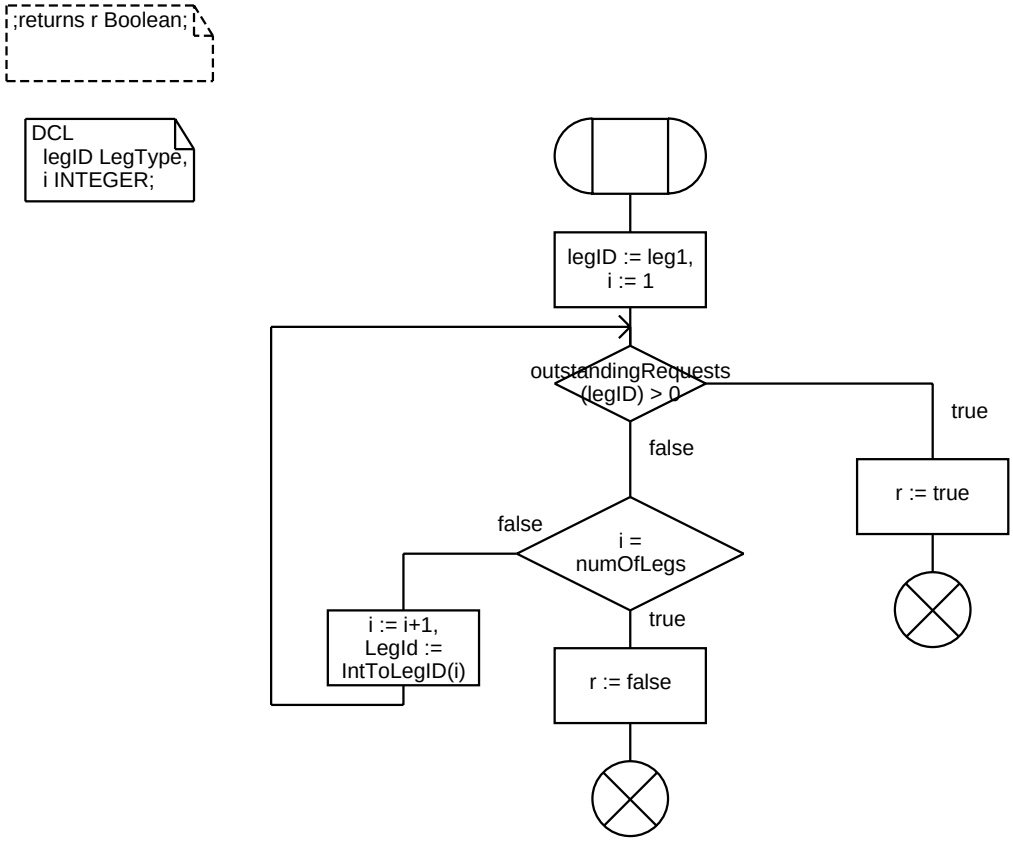


Procedure ApplyChargingReportPending 1(1)

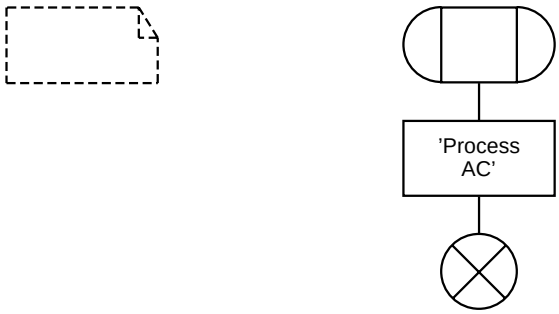




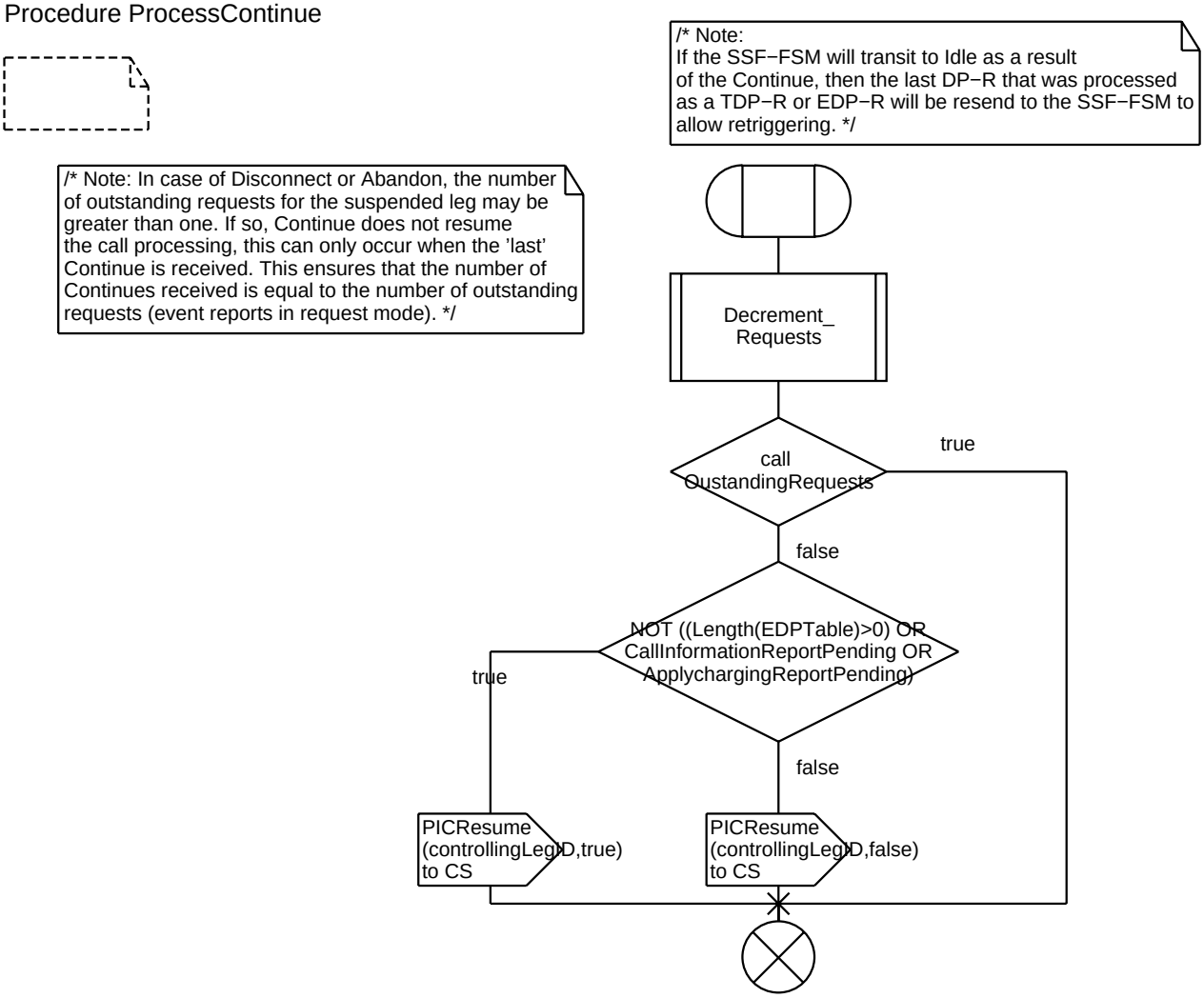
Procedure OustandingRequests 1(1)



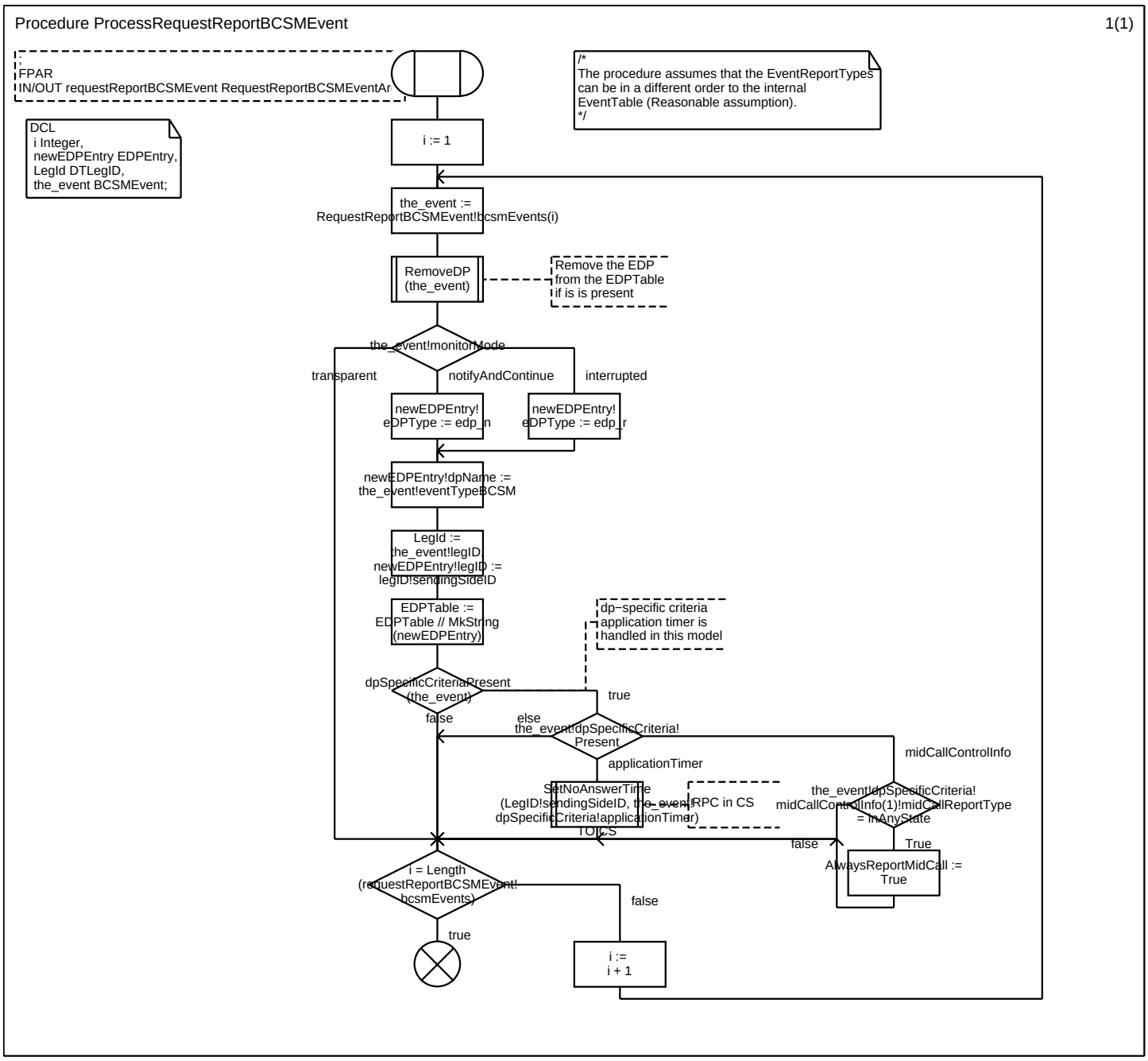
Procedure ProcessApplyCharging 1(1)



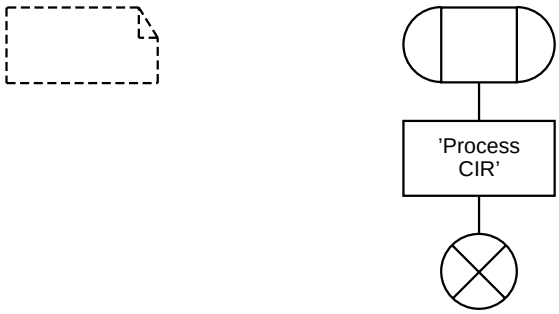
Procedure ProcessContinue 1(1)



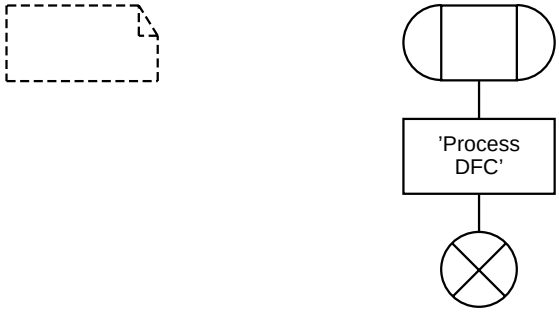
ProcessRequestReportBCSMEvent



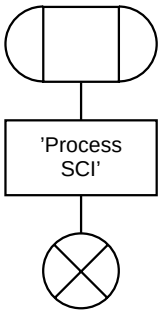
Procedure ProcessCallInformationRequest 1(1)



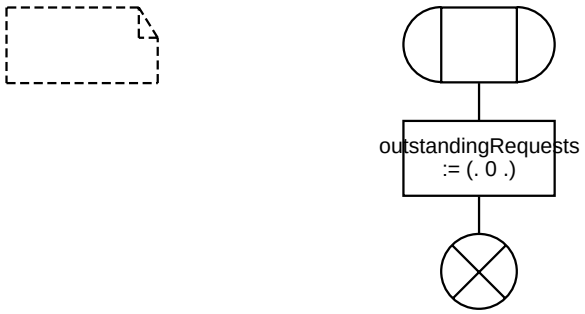
Procedure ProcessDisconnectForwardConnection 1(1)



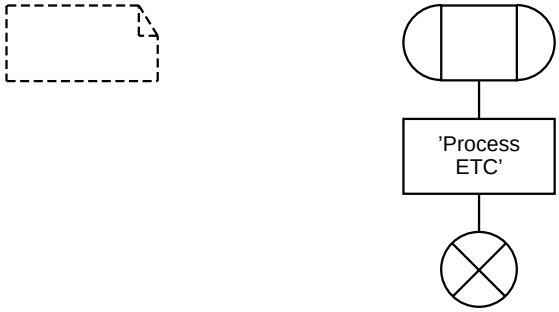
Procedure ProcessSendChargingInformation 1(1)



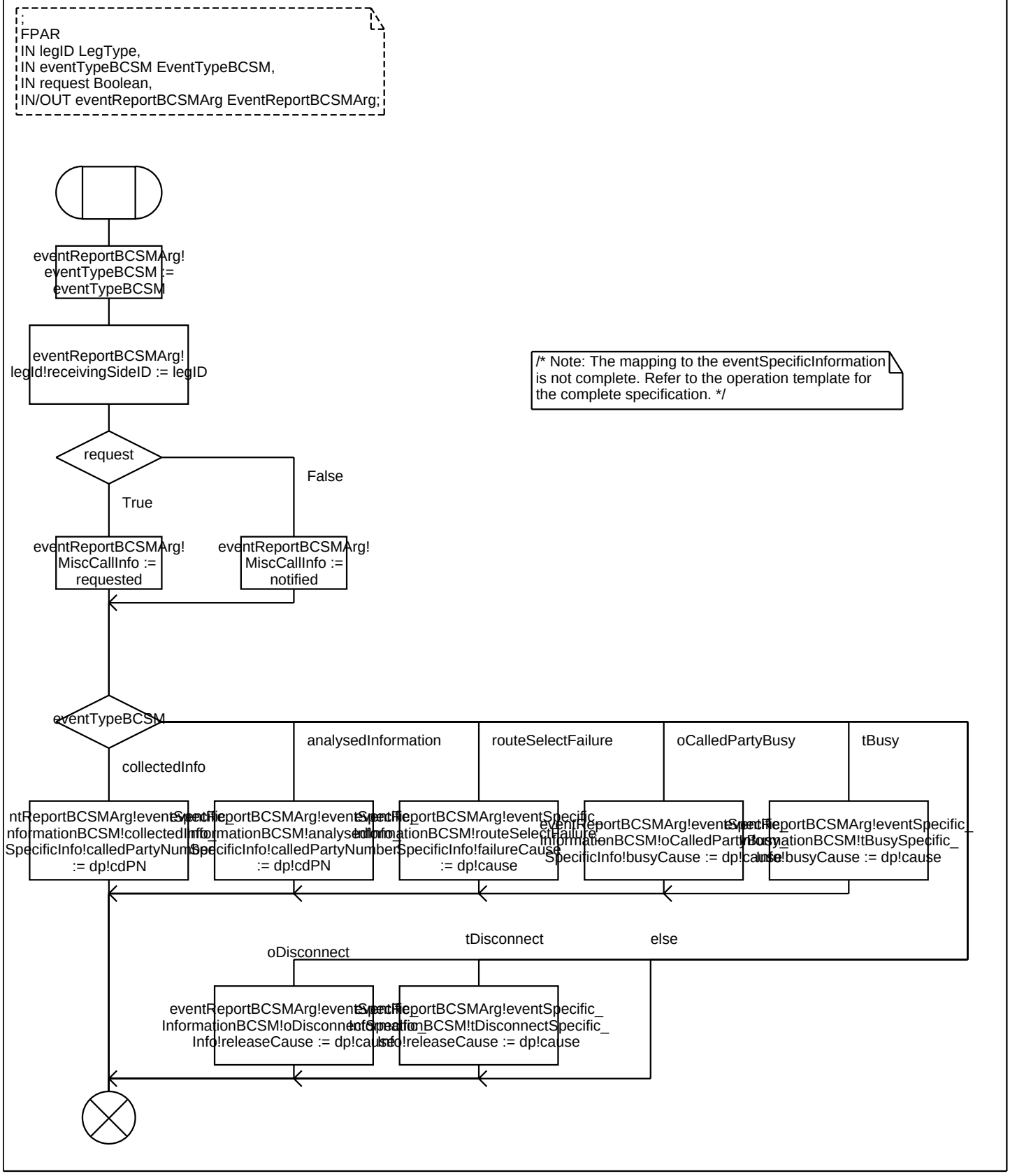
Procedure <<Process Type SSF_FSM>> ProcessCancel 1(1)



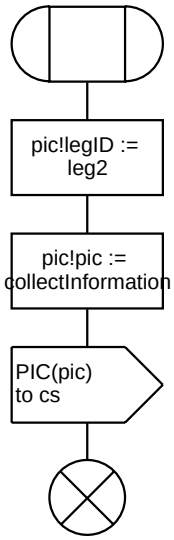
Procedure ProcessEstablishTemporaryConnection 1(1)



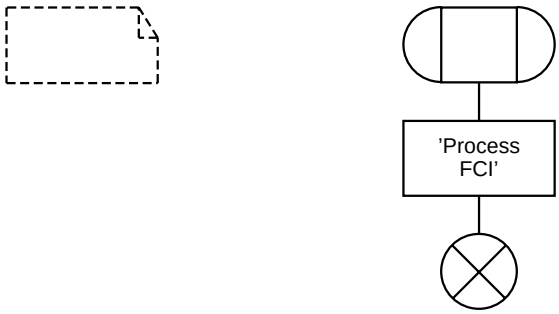
Procedure ProcessEventReportBCSM 1(1)



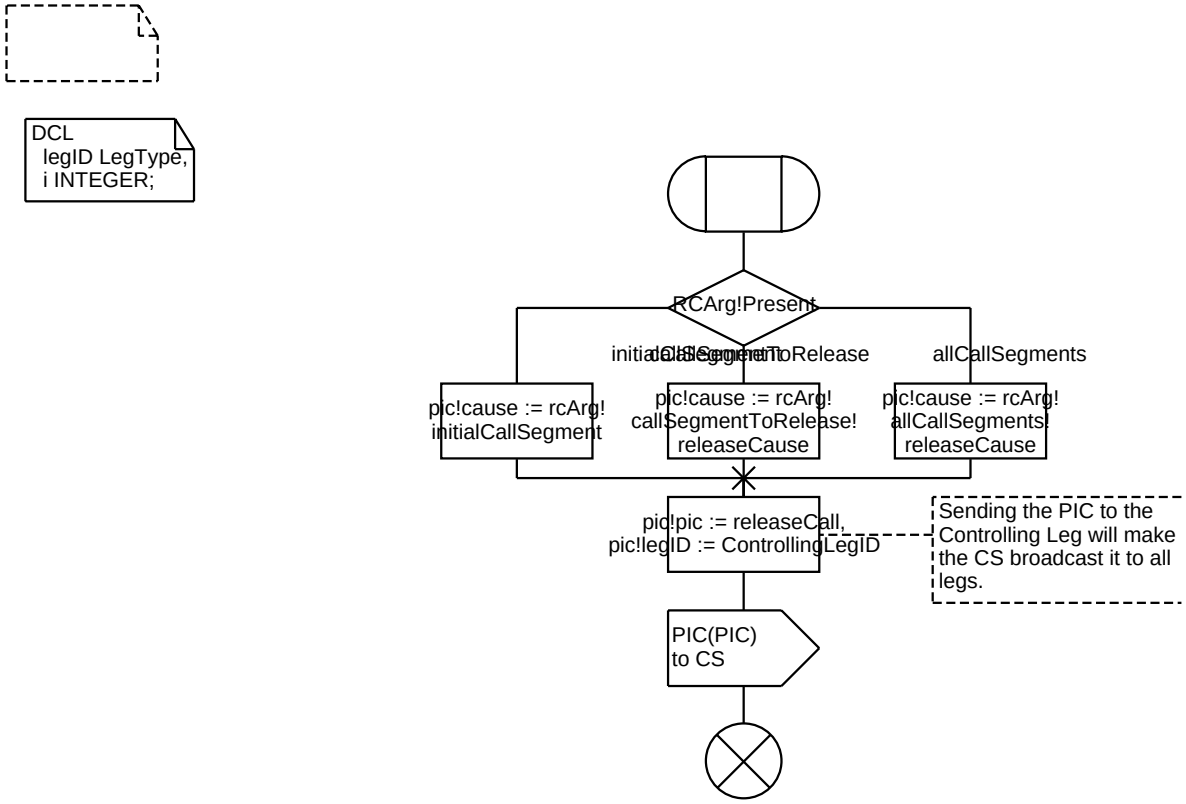
Procedure ProcessCollectInformation 1(1)



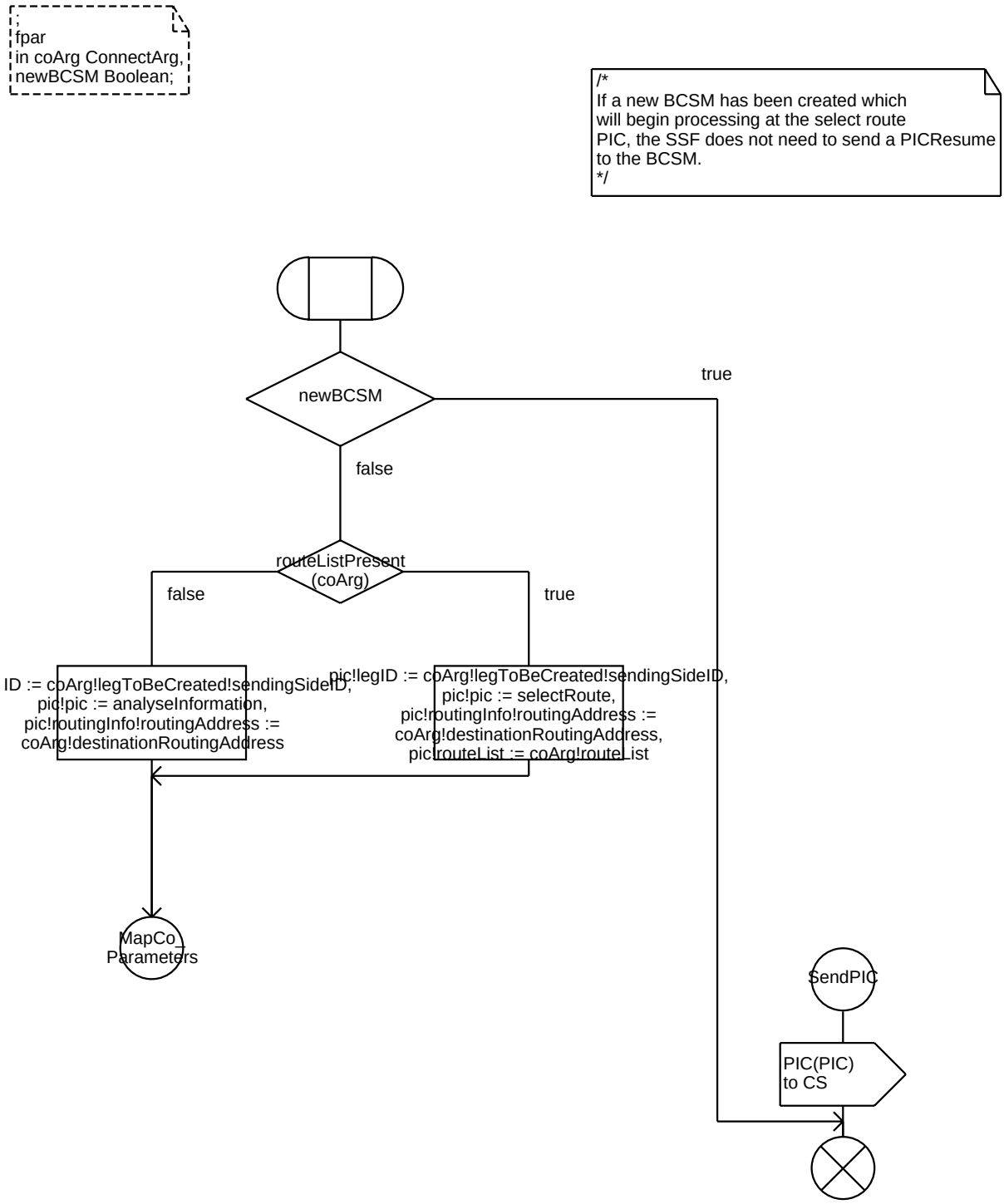
Procedure ProcessFurnishChargingInformation 1(1)



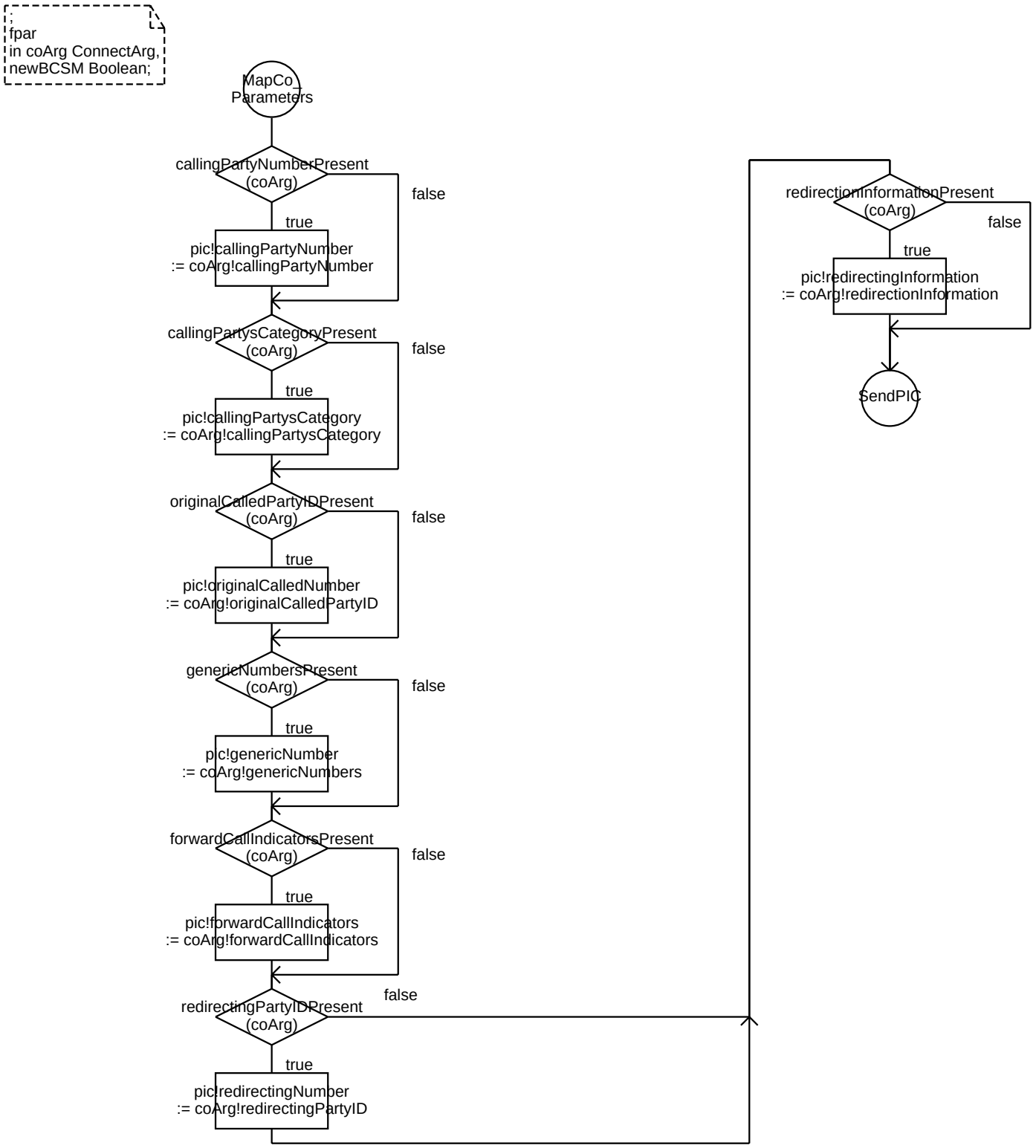
Procedure ReleaseAllLegs 1(1)



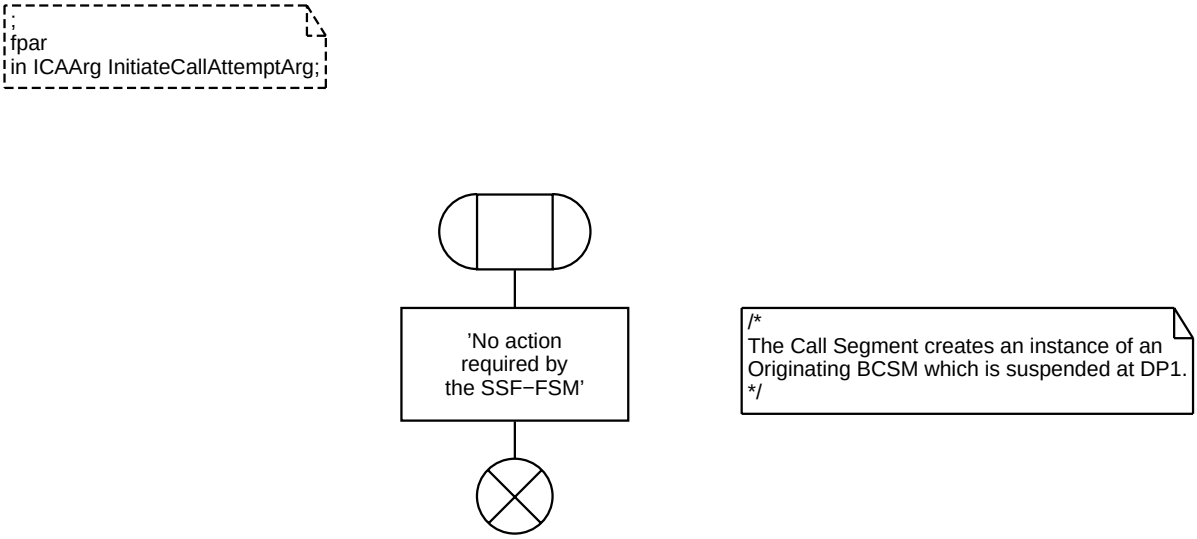
Procedure ProcessConnect 1(2)



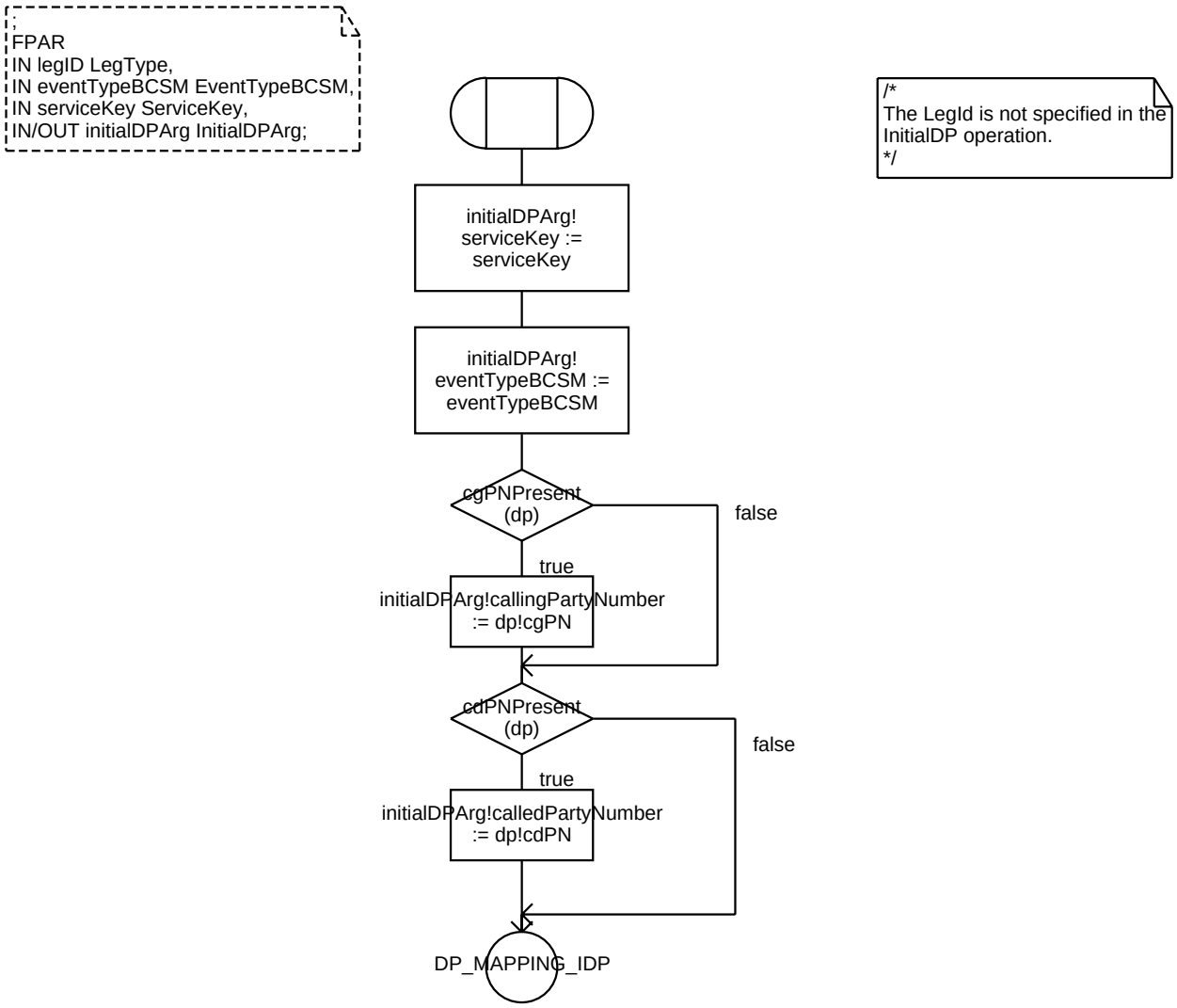
Procedure ProcessConnect 2(2)



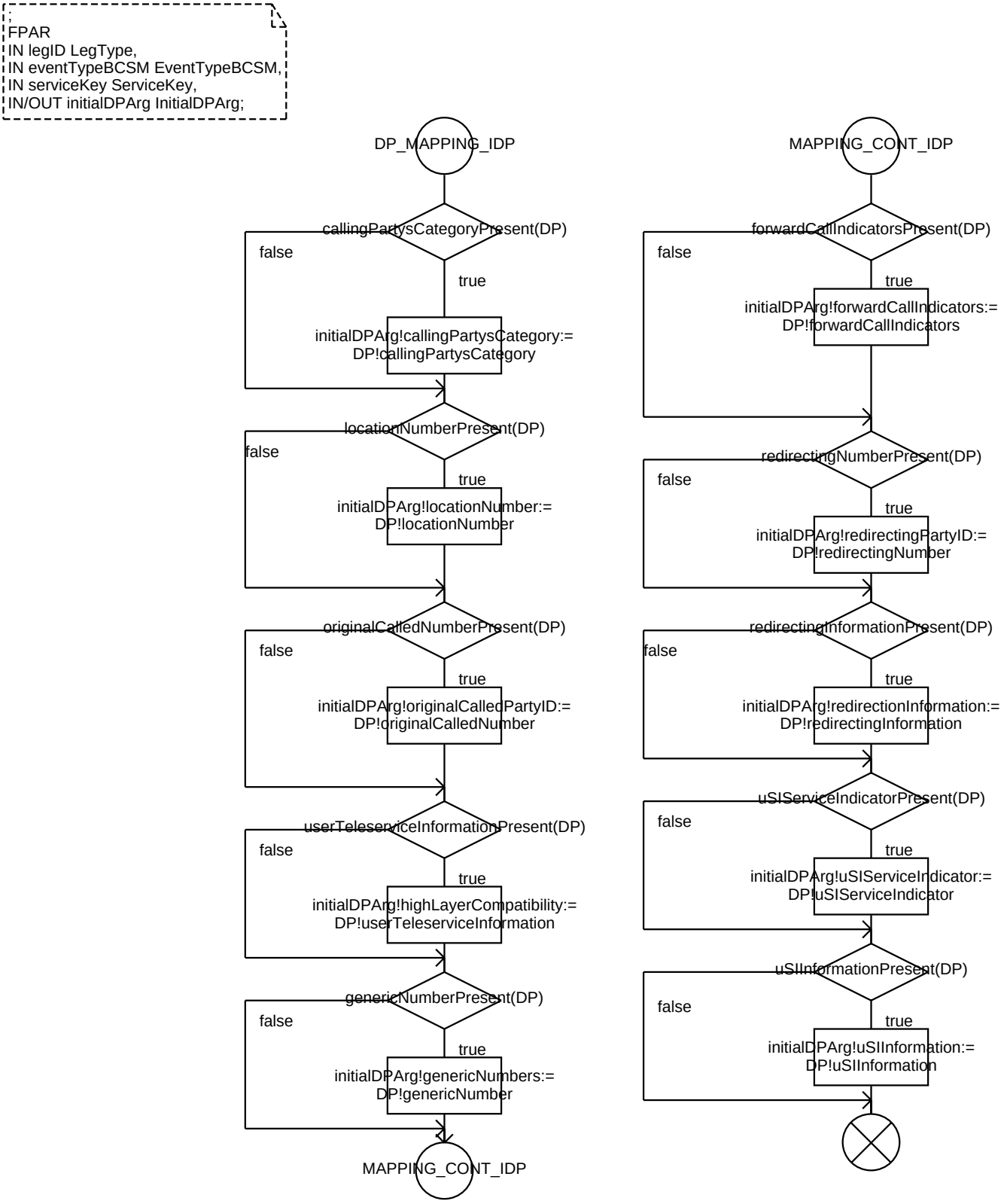
Procedure ProcessInitiateCallAttempt 1(1)



Procedure ProcessInitialDP 1(2)

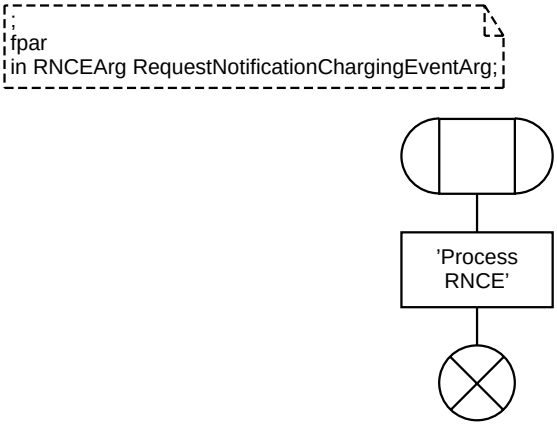


Procedure ProcessInitialDP 2(2)

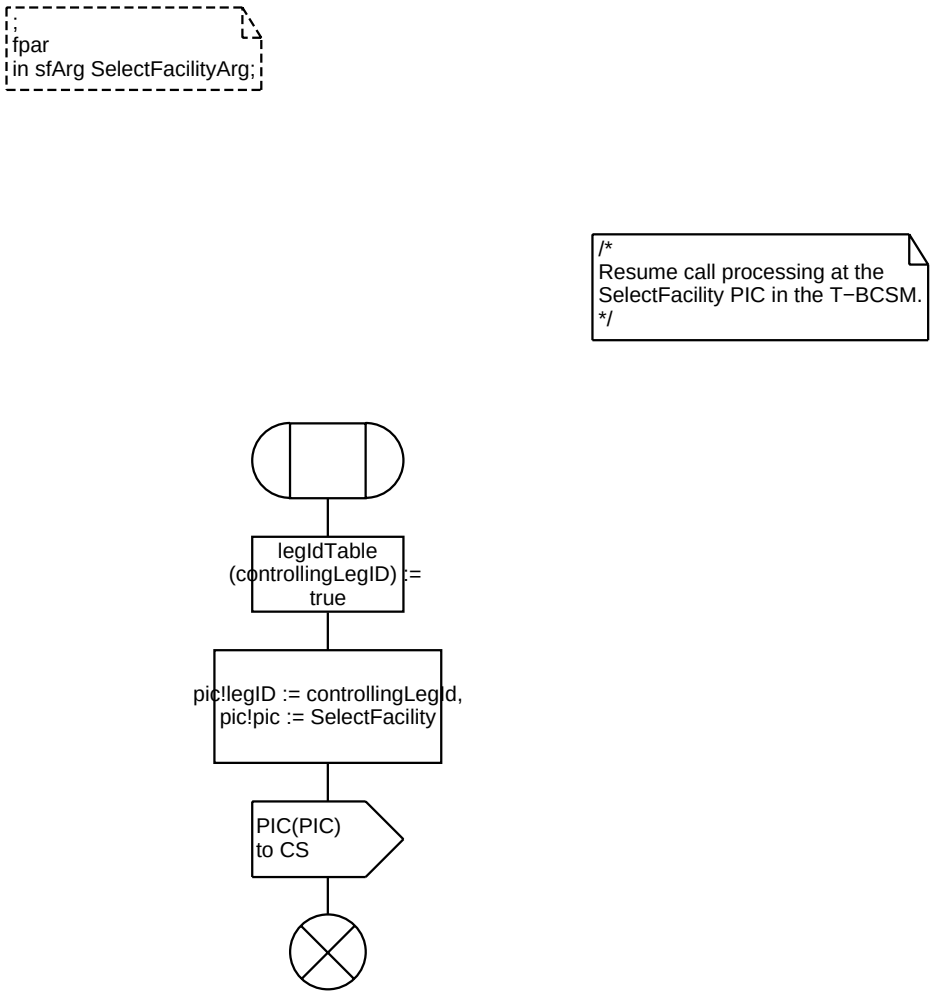


Procedure ProcessConnectToResource 1(1)

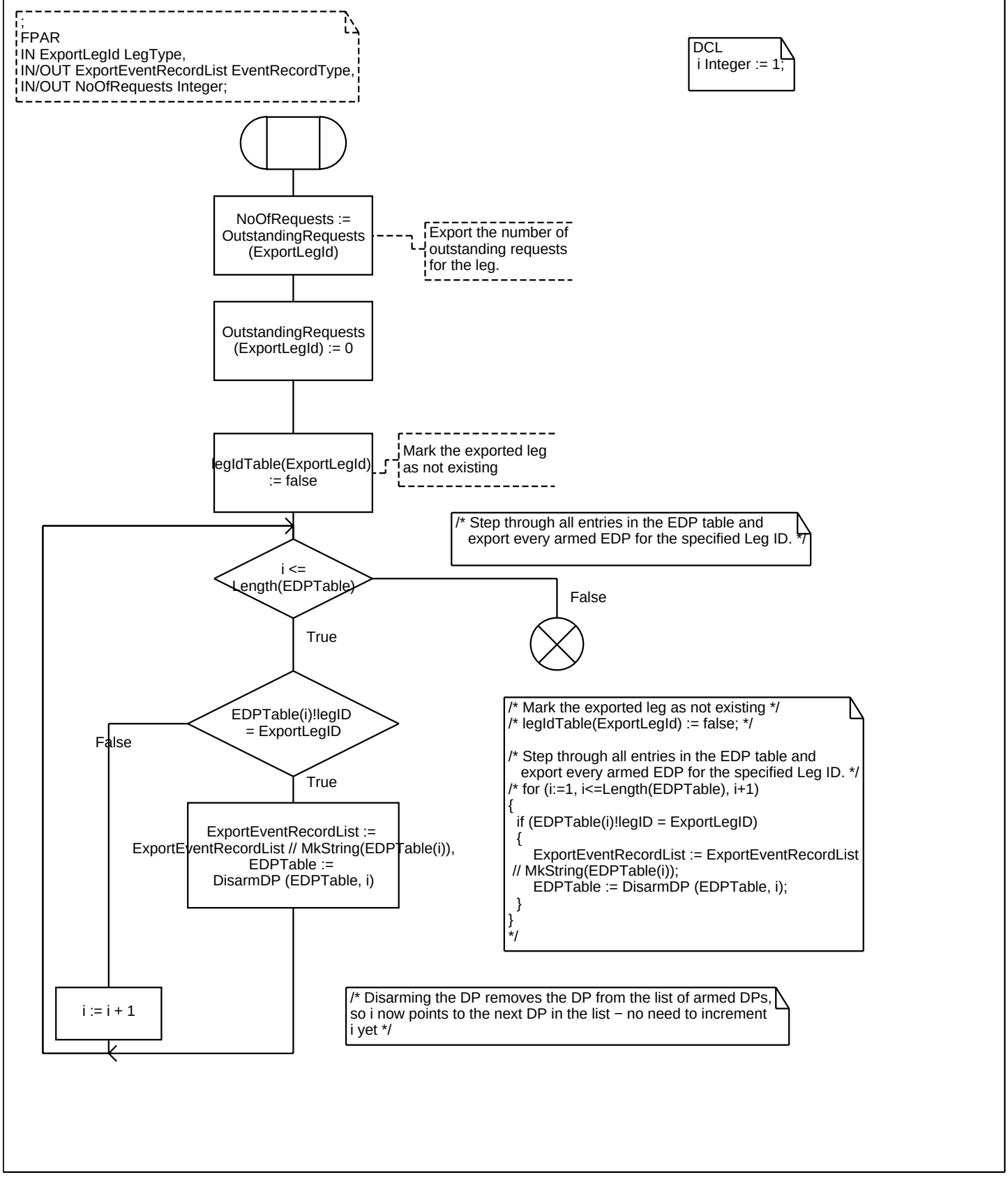
Procedure ProcessRequestNotificationChargingEvent 1(1)



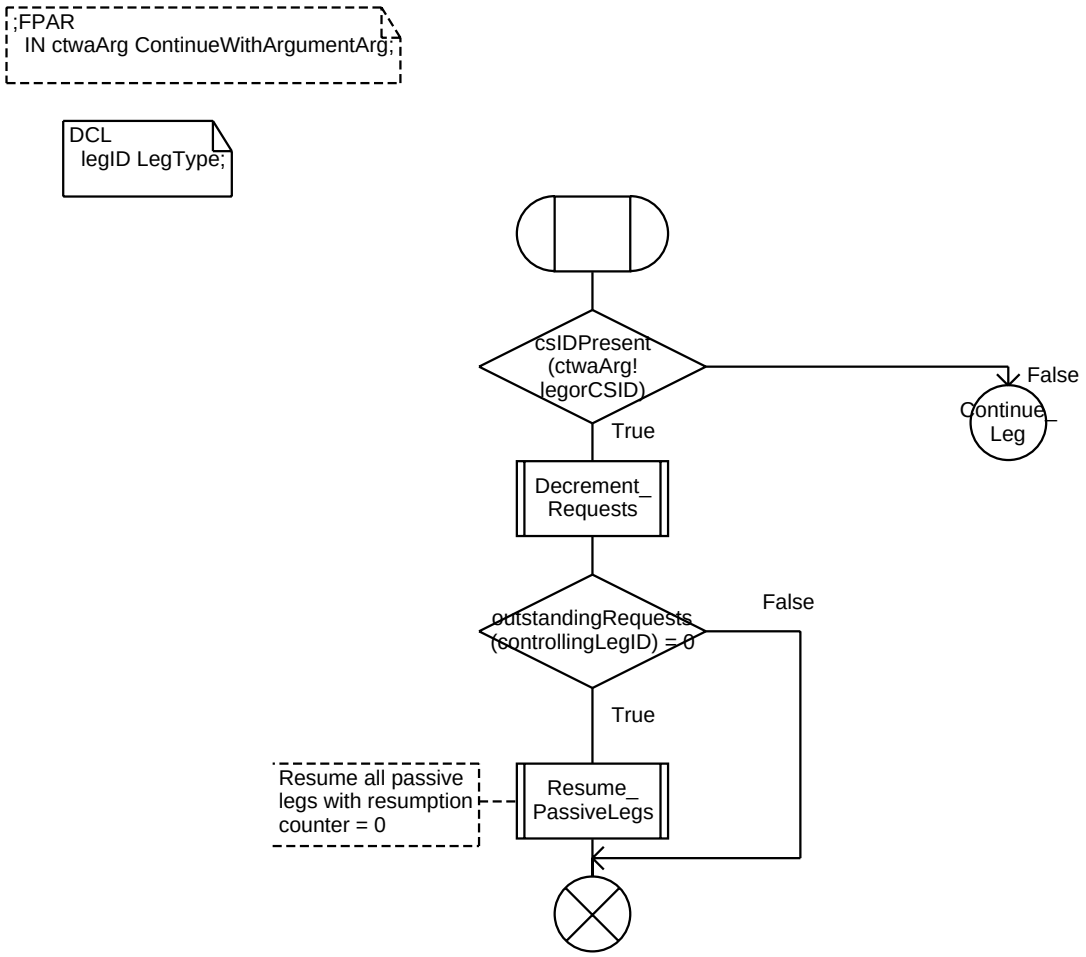
Procedure ProcessSelectFacility 1(1)



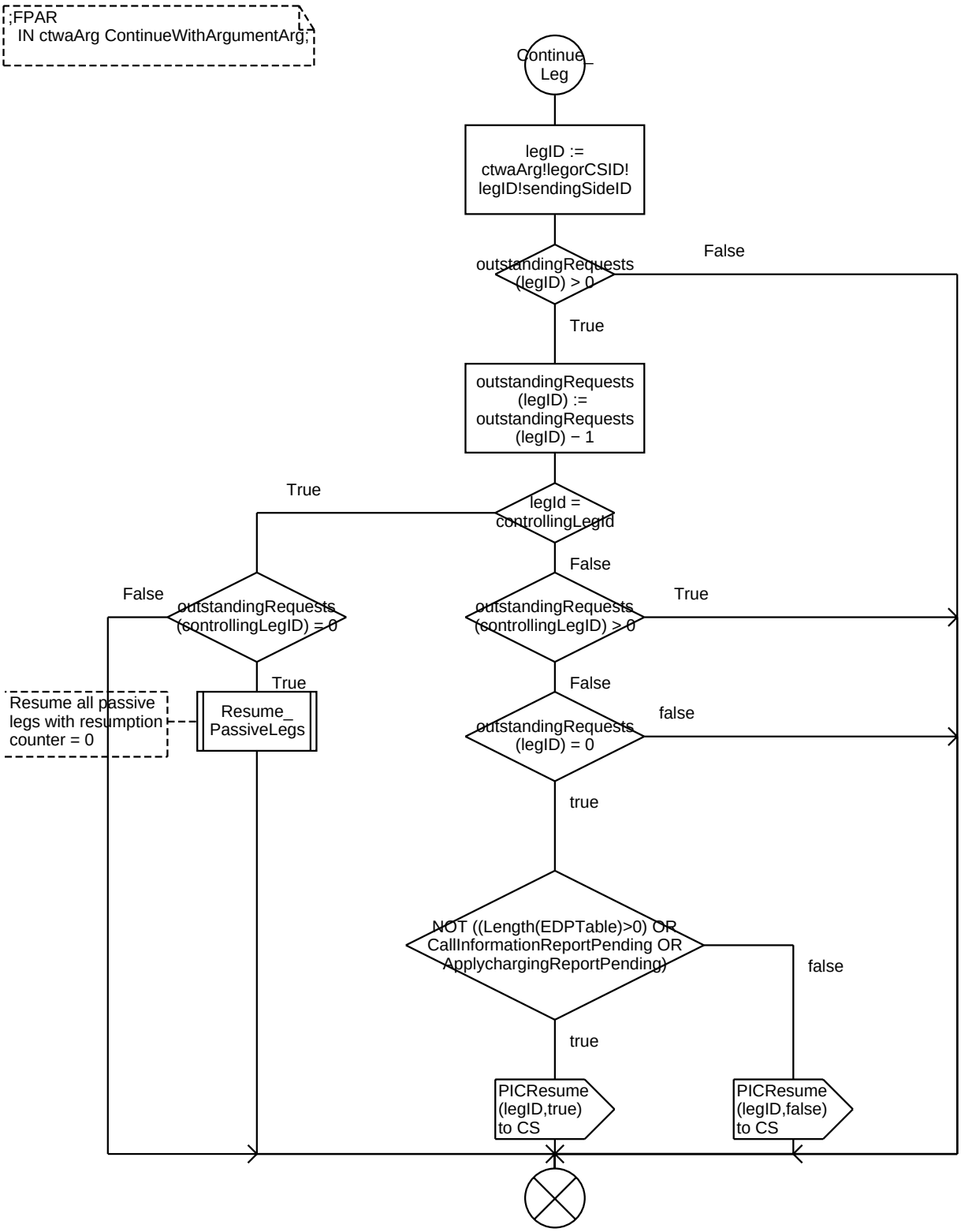
Procedure ExportEventRecord 1(1)



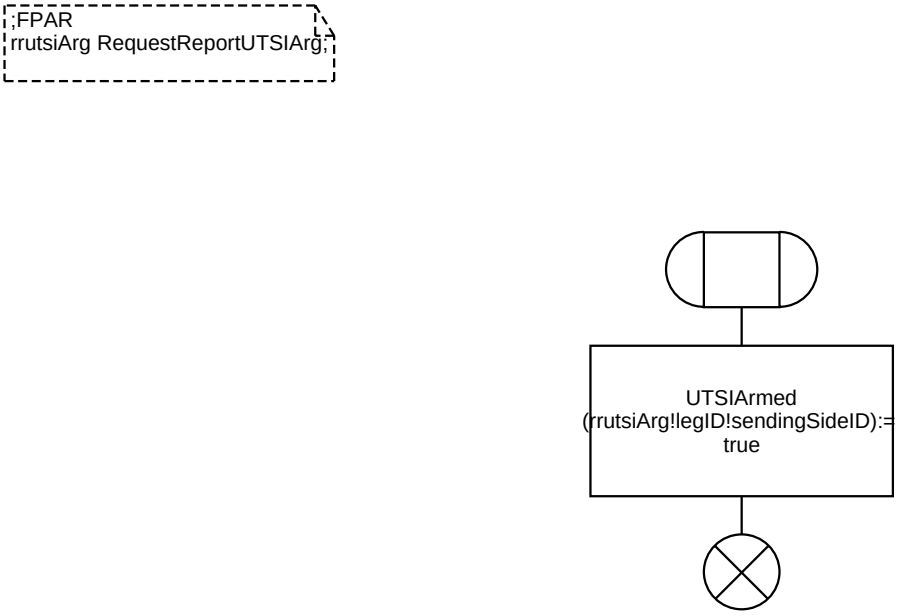
Procedure ProcessContinueWithArgument 1(2)



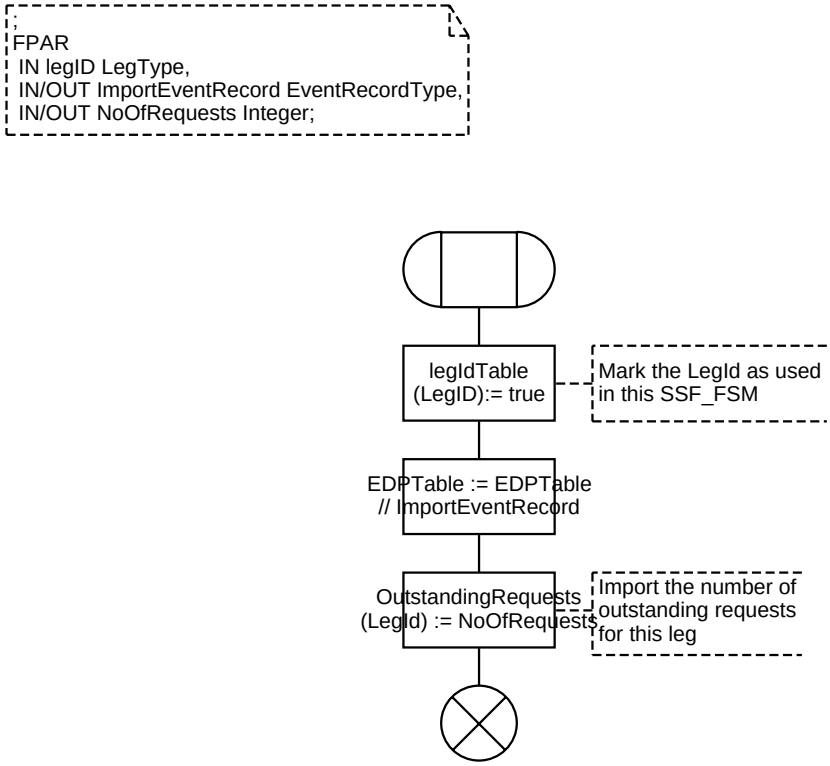
Procedure ProcessContinueWithArgument 2(2)



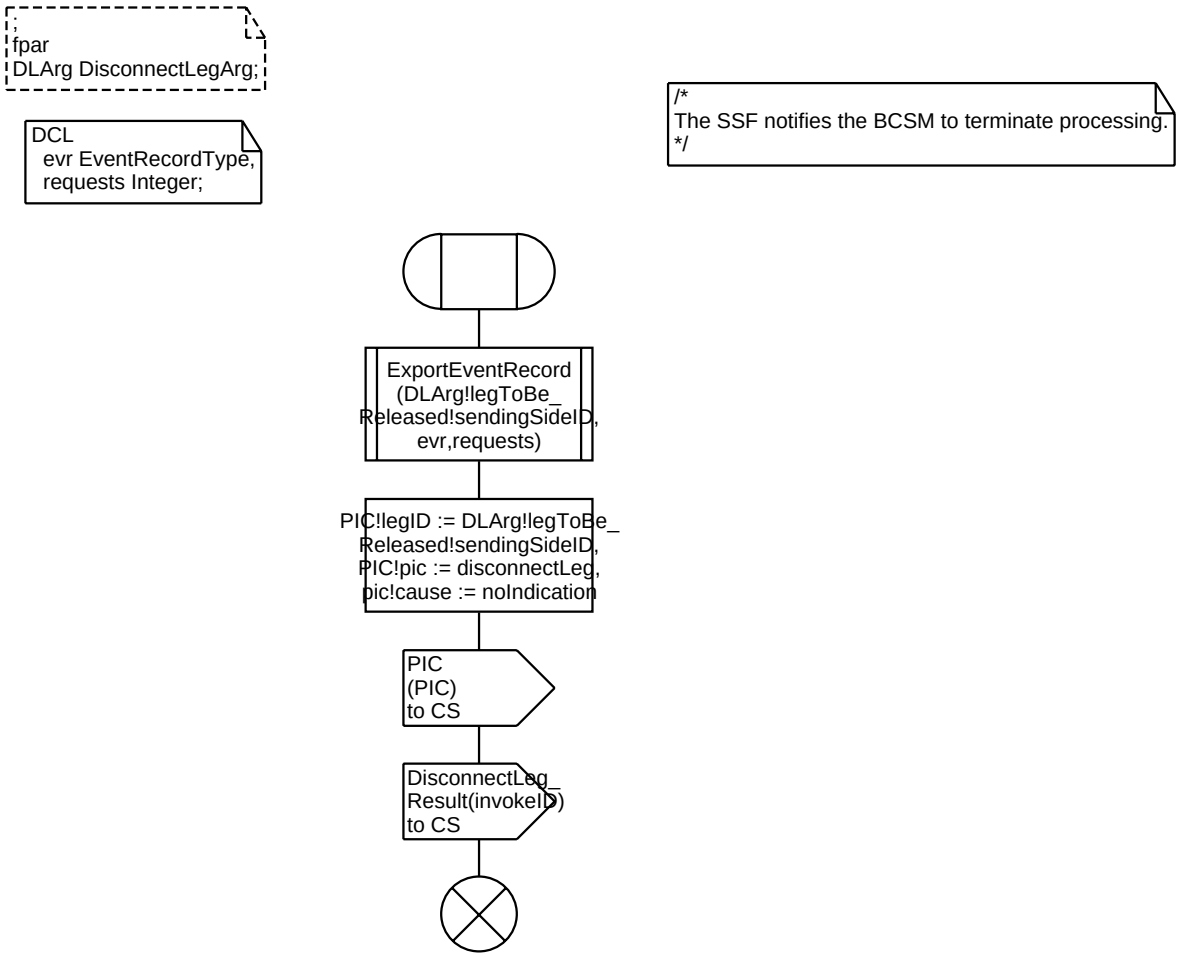
Procedure ProcessRequestReportUTSI 1(1)



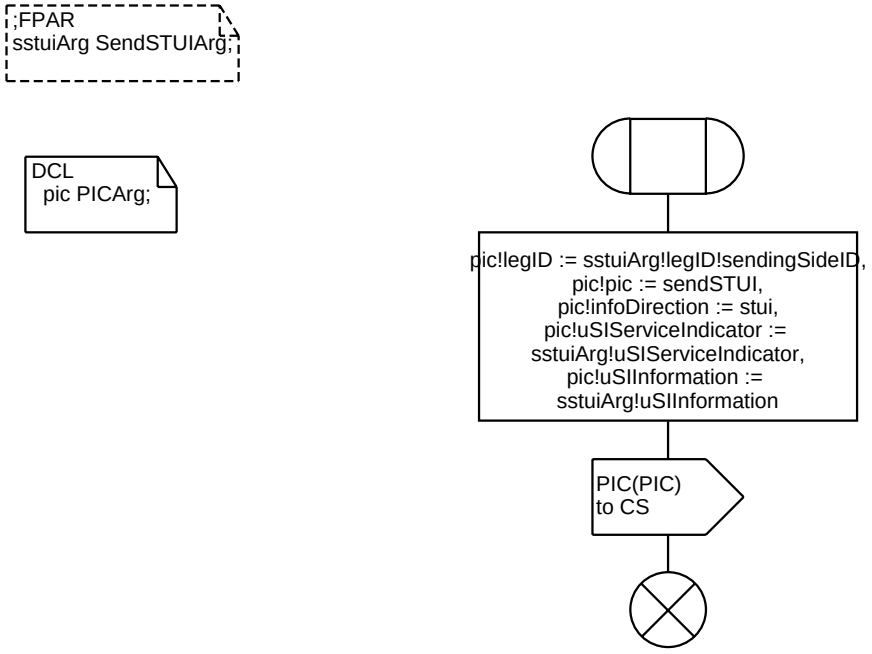
Procedure ImportEventRecord 1(1)



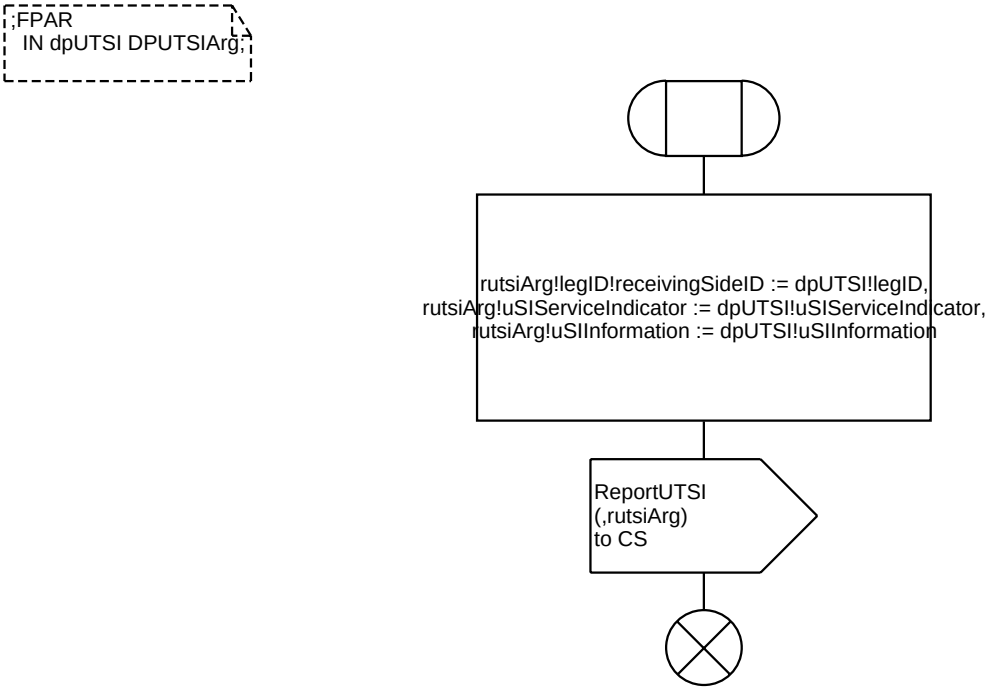
Procedure ProcessDisconnectLeg 1(1)



Procedure ProcessSendSTUI 1(1)

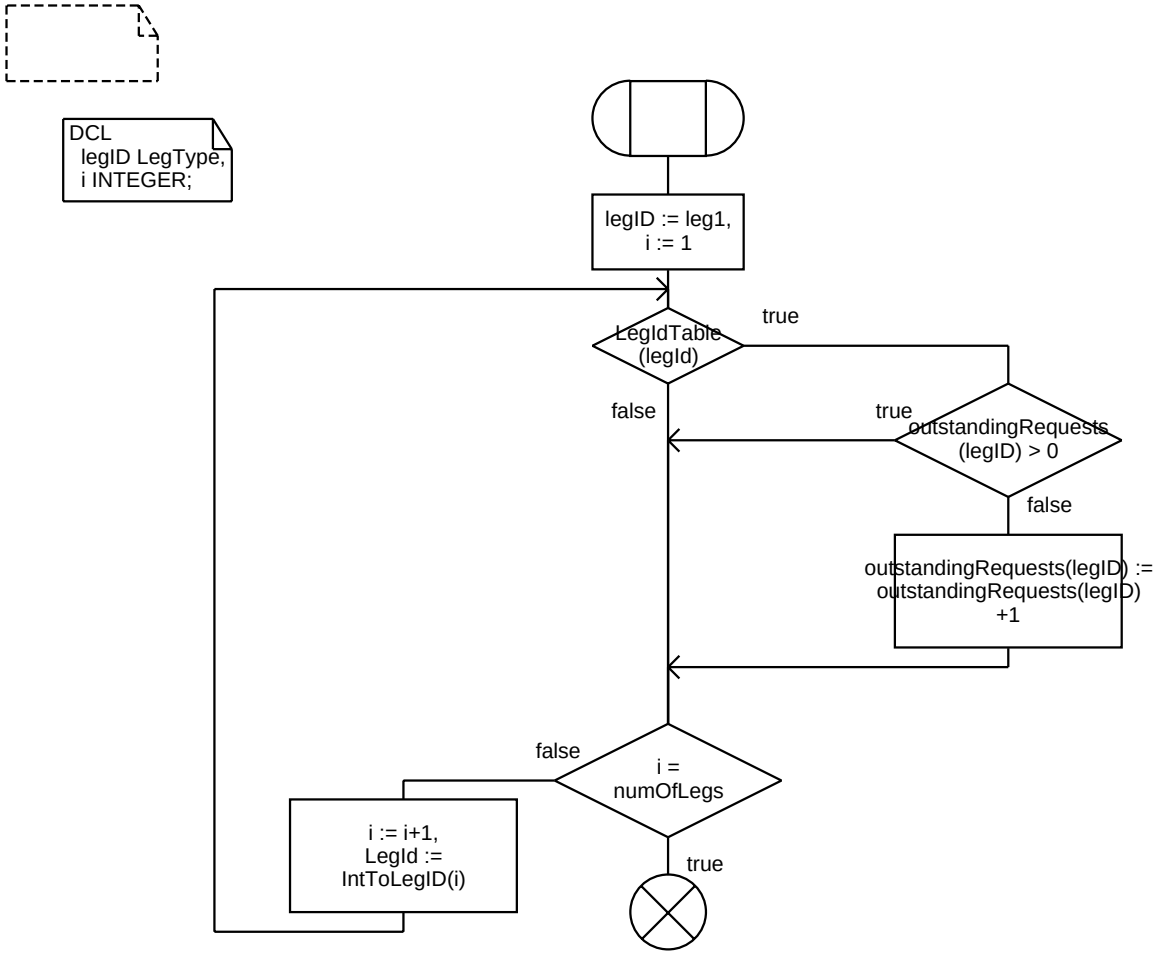


Procedure ProcessReportUTSI 1(1)

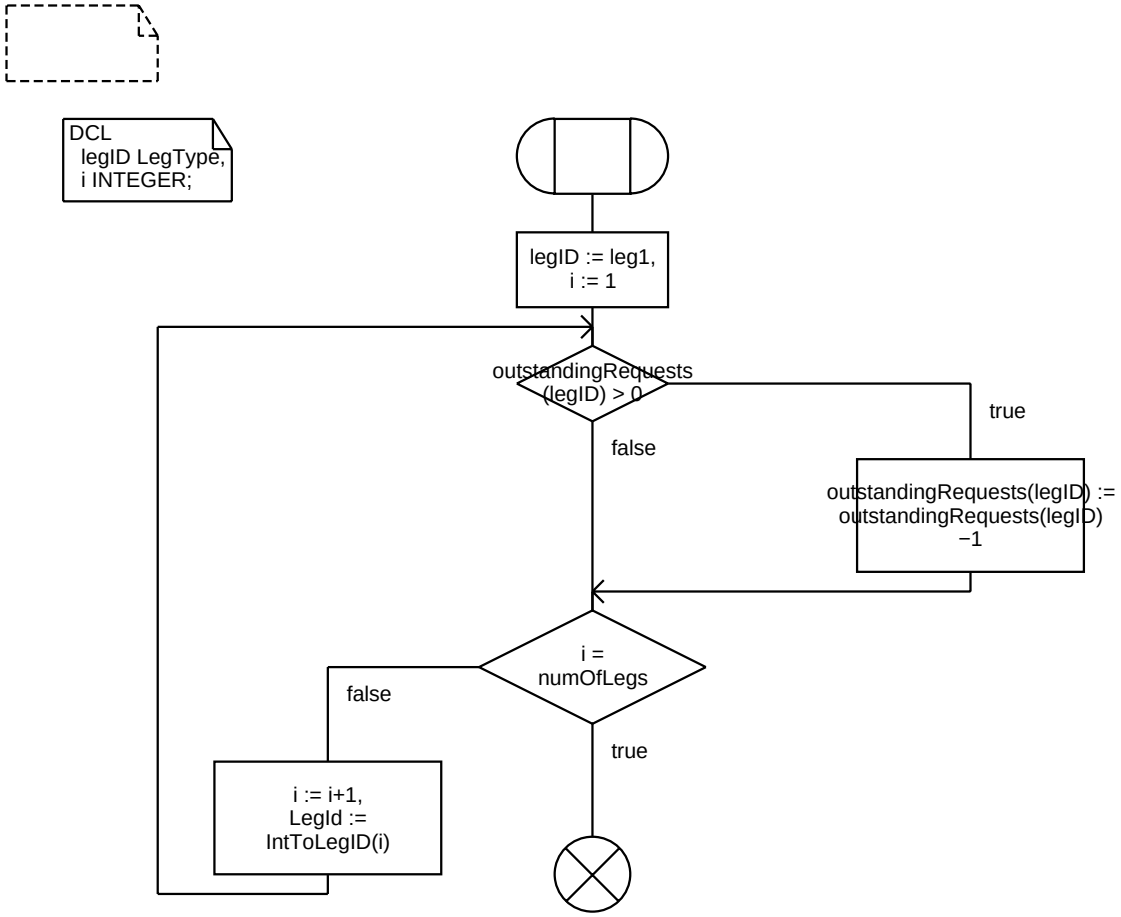


Procedure IsUTSIArmed 1(1)

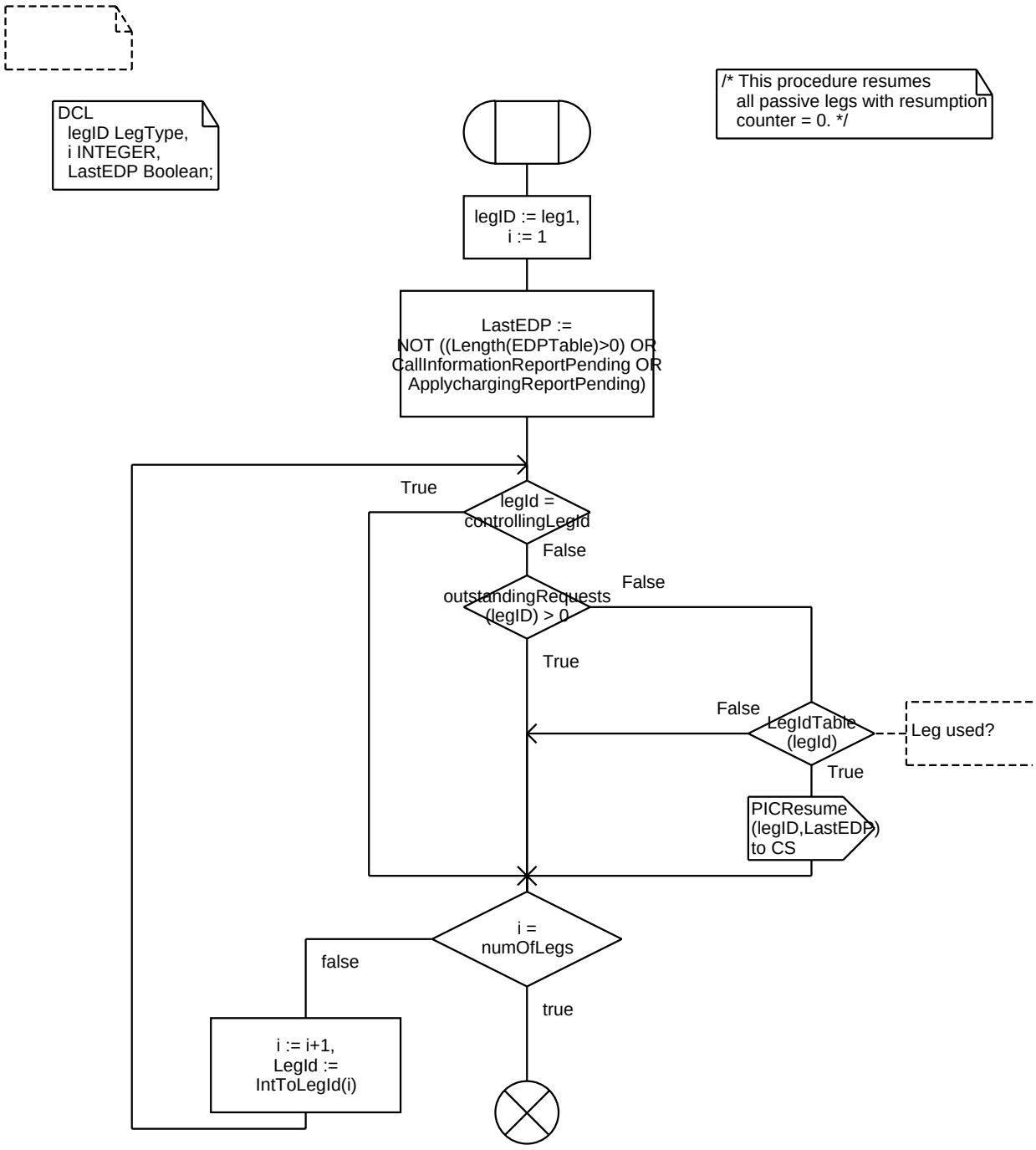
Procedure IncrementOutstandingRequestsCSId 1(1)



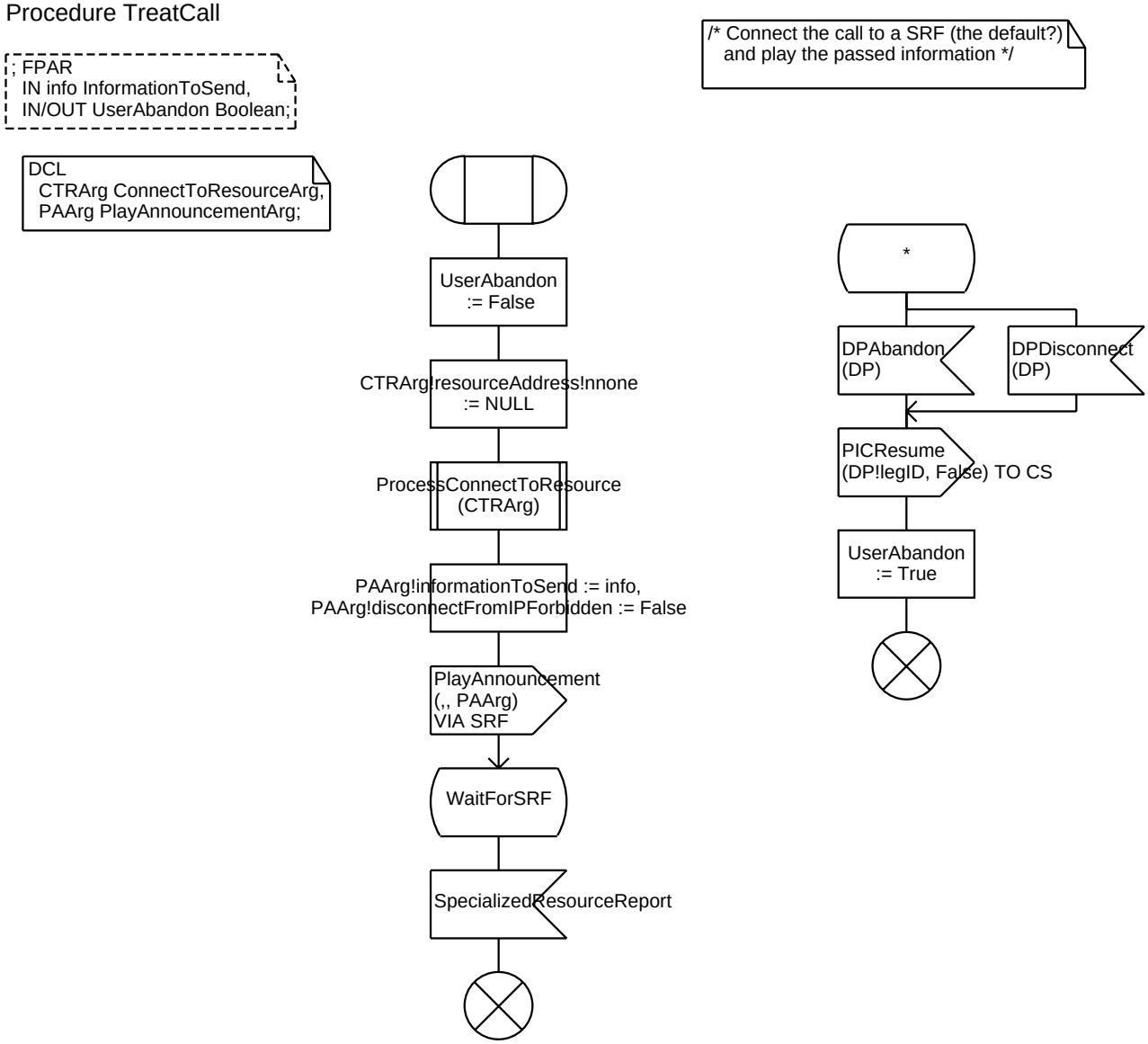
Procedure DecrementRequests 1(1)



Procedure ResumePassiveLegs 1(1)



Procedure TreatCall 1(1)



Process Type FIM_Controller 1(2)

```
/* This process is responsible
 * for the creation of FIMAgents
 */
```

```
/* Type definitions and variable declarations */

NEWTYPE CSAIDToPId
  Array (CSAID, PId)
ENDNEWTYPE;

DCL
  CSAIDToFIMAgent CSAIDToPId,
    /* CSAIDToFIMAgent is used to obtain the FIMAgent for
     * a given CSAID. One FIMAgent can handle several CSAIDs.
     * In this model, CSAIDs are unique, and even when a
     * call segment association ceases to exist, its CSAID will
     * never be reused. This fact is exploited in this process:
     * when a call segment association is released, CSAIDToFIMAgent
     * will not be updated, so this variable will grow infinitely
     * large. Obviously, real implementations should do something about this
     */
 MaxChainLength Integer;
    /* Max. Length of the BCSM chain, usually set to constant
     * MaxActiveServices, but by using simulator commands, this
     * length can be changed dynamically
     */
```

exported CreateFIMAgent

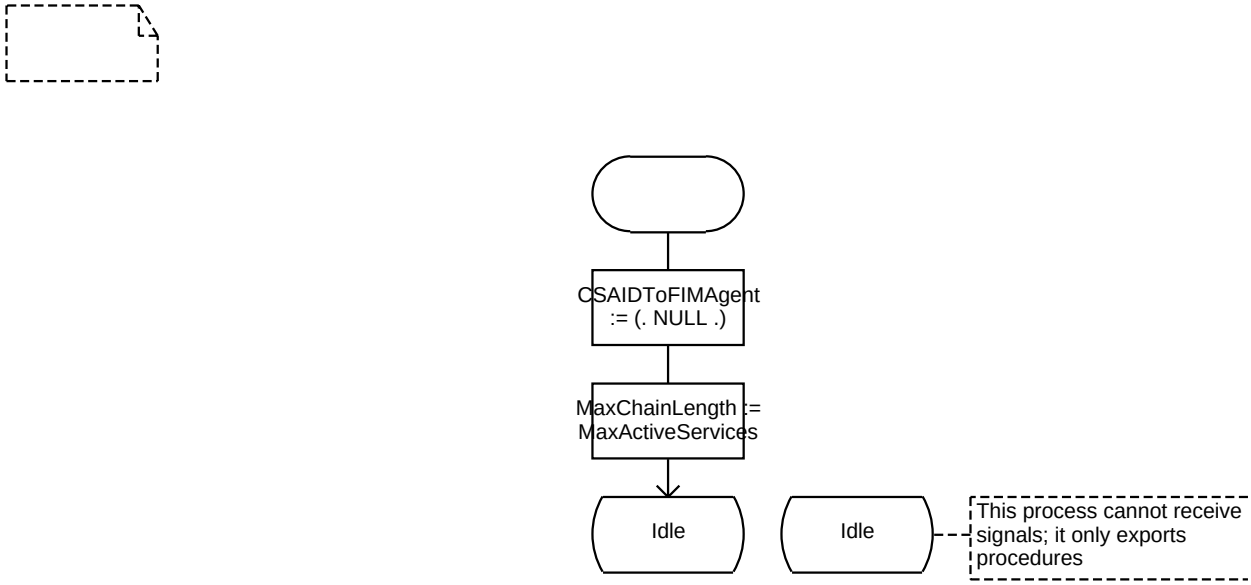
exported FIMAgentStopped

exported UpdateFAInfo

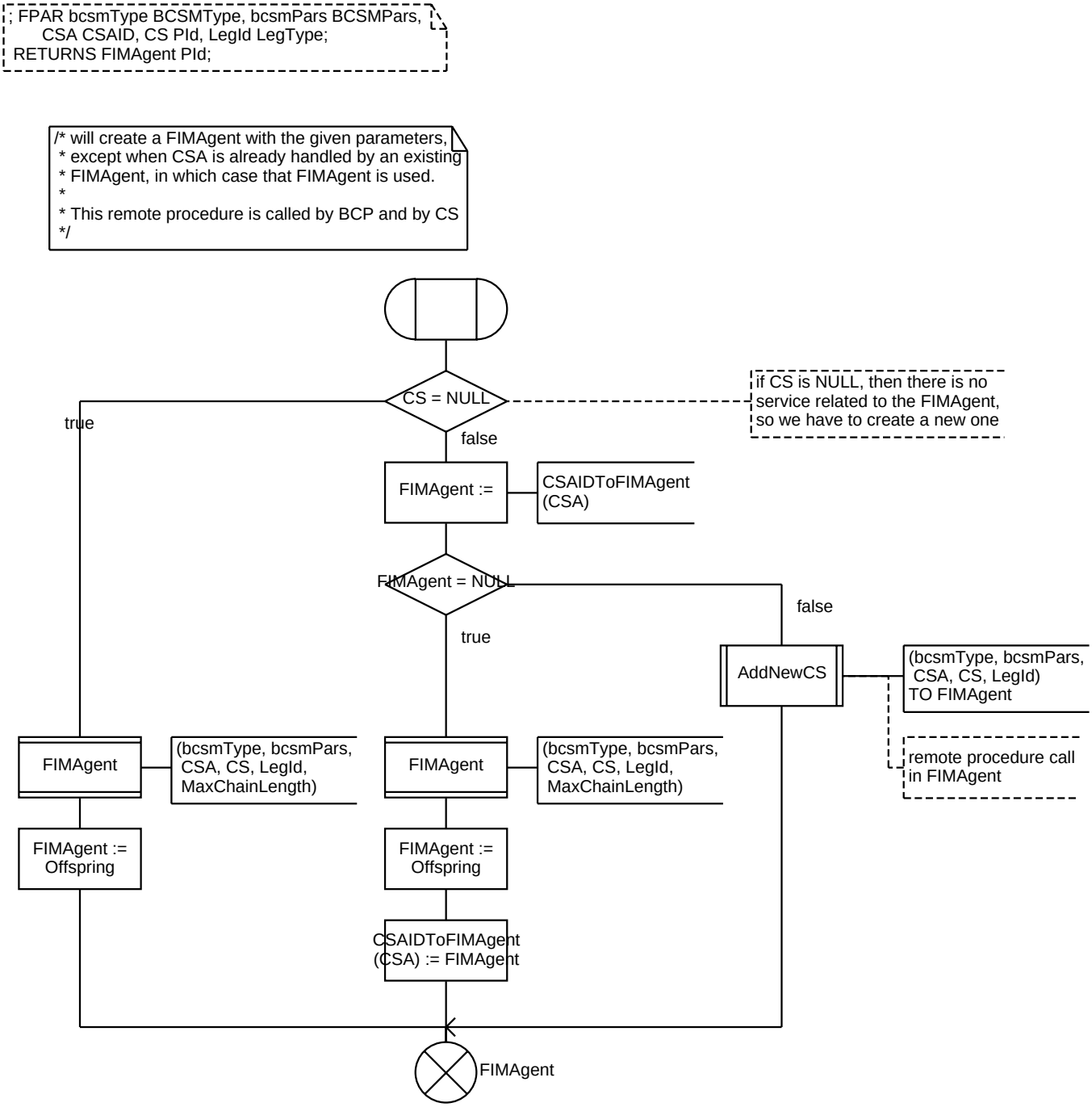
```
/*   IMPORTED REMOTE PROCEDURES   */

/* AddNewCS is exported by FIMAgent and called by FIM_Controller.
 * It is called when CS wants to create a new FIMAgent, but CS's CSA
 * that is already handled by some FIMAgent.
 * In this case, the existing FIMAgent is reused.
 * The call of this RPC will result in a new BCSM chain being created.
 */
IMPORTED PROCEDURE AddNewCS;
 FPAR BCSMType, BCSMPars, CSAID, PId, LegType;
```

exported WhichFIMAgent

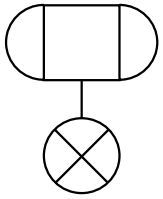
Process Type FIM_Controller 2(2)



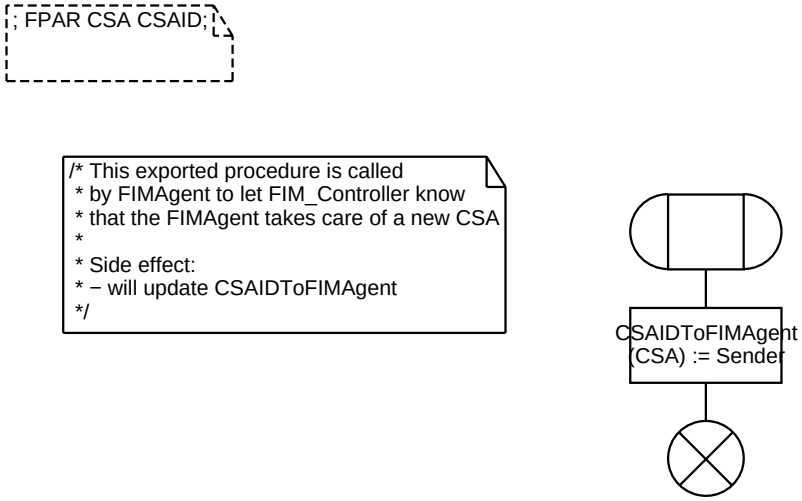
Procedure <<Process Type FIM_Controller>> CreateFIMAgent 1(1)



Procedure FIMAgentStopped 1(1)



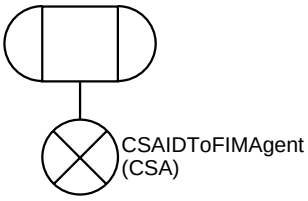
Procedure UpdateFAInfo 1(1)



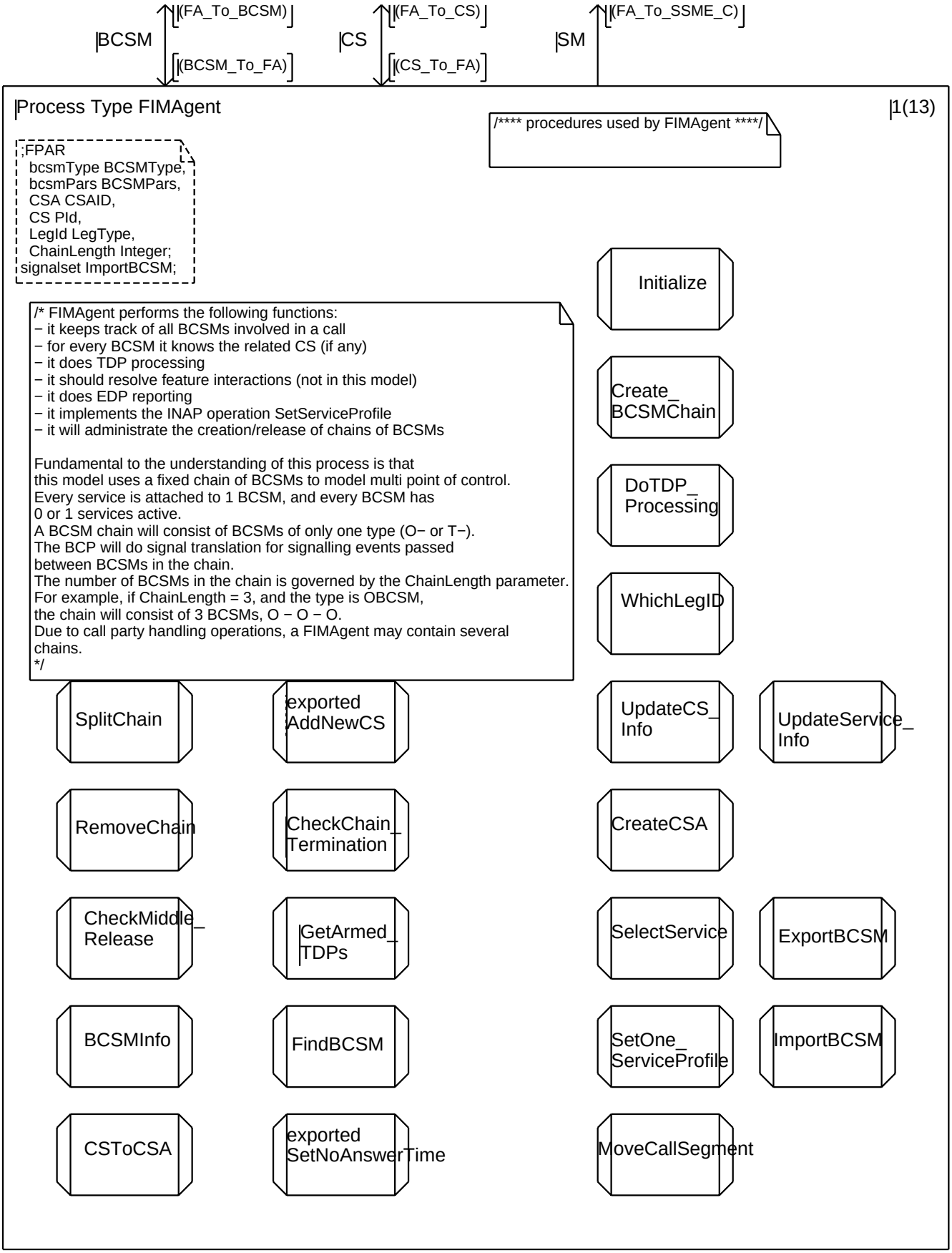
Procedure WhichFIMAgent 1(1)

; FPAR IN CSA CSAID;
RETURNS PId;

```
/* returns the PId of the FIMAgent
 * that takes care of this CSA
 * (or NULL if there is no such FIMAgent)
 */
```



FIMAgent


BCSM
(FA_To_BCSM)
(BCSM_To_FA)
CS
(FA_To_CS)
(CS_To_FA)
SM
(FA_To_SSME_C)
Process Type FIMAgent
1(13)
/**** procedures used by FIMAgent ****/
;FPAR
bcsmType BCSMType,
bcsmPars BCSMPars,
CSA CSAID,
CS PId,
LegId LegType,
ChainLength Integer;
signalset ImportBCSM;
/* FIMAgent performs the following functions:
− it keeps track of all BCSMs involved in a call
− for every BCSM it knows the related CS (if any)
− it does TDP processing
− it should resolve feature interactions (not in this model)
− it does EDP reporting
− it implements the INAP operation SetServiceProfile
− it will administrate the creation/release of chains of BCSMs
Fundamental to the understanding of this process is that
this model uses a fixed chain of BCSMs to model multi point of control.
Every service is attached to 1 BCSM, and every BCSM has
0 or 1 services active.
A BCSM chain will consist of BCSMs of only one type (O− or T−).
The BCP will do signal translation for signalling events passed
between BCSMs in the chain.
The number of BCSMs in the chain is governed by the ChainLength parameter.
For example, if ChainLength = 3, and the type is OBCSM,
the chain will consist of 3 BCSMs, O − O − O.
Due to call party handling operations, a FIMAgent may contain several
chains.
*/
Initialize
Create_ BCSMChain
DoTDP_ Processing
WhichLegID
SplitChain
exported AddNewCS
UpdateCS_ Info
UpdateService_ Info
RemoveChain
CheckChain_ Termination
CreateCSA
CheckMiddle_ Release
GetArmed_ TDPs
SelectService
ExportBCSM
BCSMInfo
FindBCSM
SetOne_ ServiceProfile
ImportBCSM
CSToCSA
exported SetNoAnswerTime
MoveCallSegment

Process Type FIMAgent 2(13)

```
;FPAR
  bcsmType BCSMType,
  bcsmPars BCSMPars,
  CSA CSAID,
  CS PId,
  LegId LegType,
  ChainLength Integer;
signalset ImportBCSM;
```

```
/*  DECLARATION OF REMOTE PROCEDURES IMPORTED
 BY THE FIM AGENT   */
```

```
IMPORTED PROCEDURE BCSMStartChain;

IMPORTED PROCEDURE SetFIMAgent;
  FPAR PId;


IMPORTED PROCEDURE CreateBCSM;
 FPAR BCSMType, BCSMPars, PId;
 RETURNS PId;


IMPORTED PROCEDURE BCSMChainCreated;
 FPAR PId, PId;


IMPORTED PROCEDURE RemoveBCSMChain;
 FPAR PId;


IMPORTED PROCEDURE UpdateFAInfo;
 FPAR CSAID;

IMPORTED PROCEDURE FIMAgentStopped;

IMPORTED PROCEDURE GetMasterTDPTable;
 RETURNS TDPTableType;
```

Process Type FIMAgent 3(13)

```
;FPAR
  bcsmType BCSMType,
  bcsmPars BCSMPars,
  CSA CSAID,
  CS PId,
  LegId LegType,
  ChainLength Integer;
signalset ImportBCSM;
```

```
/* signal between FIMAgents, used to implement MoveCallSegments */
SIGNAL ImportBCSM (PId, BCSMInfo, CSAInfo);
```

```
/*  MORE IMPORTED PROCEDURES   */


/* all below remote procedures are exported by BCP and
   called by FIMAgent */

/* creates a new BCSM attached to a controlling leg. */
IMPORTED PROCEDURE CreateControllingBCSM;
 FPAR BCSMType, BCSMPars, PId;
 RETURNS Pid;

/* creates a new passive leg with a BCSM in the same
   connection point as the specified BCSM. */
IMPORTED PROCEDURE AddPassiveBCSM;
 FPAR BCSMType, BCSMPars, PId;
 RETURNS Pid;

/* cuts the chain in two. Used in middle−release */
IMPORTED PROCEDURE CutBCSMChain;
 FPAR Pid;

/* moves a controlling leg to another connection point */
IMPORTED PROCEDURE MoveControllingBCSMChain;
 FPAR/* some BCSM in source CP */ PId, /* target BCSM */ PId;

/* moves a passive leg to another connection point */
IMPORTED PROCEDURE MovePassiveBCSMChain;
 FPAR /* source BCSM */ PId, /* target BCSM */ PId;

/* deletes all information about the BCSM */
IMPORTED PROCEDURE RemoveBCSM;
 FPAR PId;
```

Process Type FIMAgent 4(13)

```
;FPAR
  bcsmType BCSMType,
  bcsmPars BCSMPars,
  CSA CSAID,
  CS PId,
  LegId LegType,
  ChainLength Integer;
signalset ImportBCSM;
```

```
/* Data types used locally in FIMAgent */

/* BCSMInfo maintains information about 1 BCSM process */
NEWTYPE BCSMInfo STRUCT
  bcsmType BCSMType;
  isStopped Boolean;
    /* true if the BCSM has stopped */
  serviceActive Boolean;
    /* true if there is a service active */
  CS PId;
    /* call segment associated with the BCSM, NULL if no service active */
  legId LegType;
    /* passive leg associated with the BCSM */
  CSA CSAID;
  positionInChain Integer;
    /* position in the chain. */
  highestServiceKey Integer;
    /* highest service key of the services triggered on the same DP */
  lastReportedDP DPArg;
    /* DP that was last reported to CS (used when continuing TDP processing after resumption) */
  prev PId;
    /* points to the previous BCSM in the chain. Null if at the beginning of the chain */
  next PId;
    /* points to the next BCSM in the chain. Null if at the end of the chain */
ENDNEWTYPE;

/* BCSMInfoArray contains a mapping from a BCSM's PId to its info */
NEWTYPE BCSMInfoArray
  Array (PId, BCSMInfo)
ENDNEWTYPE;

/* BCSMChain is used to represent a chain of BCSMs */
NEWTYPE BCSMChain STRUCT
  firstBCSM PId;
    /* points to the first BCSM in the chain. From there on, we can step through the
       chain using the prev/next fields in BCSMInfo */
ENDNEWTYPE;

/* BCSMChainList is used to represent all chains maintained by a FIMAgent */
NEWTYPE BCSMChainList
  String (BCSMChain, emptystring);
 OPERATORS
  Add: BCSMChainList, BCSMChain −> BCSMChainList;
  /* adds a new chain to the list */
  Del: BCSMChainList, Integer −> BCSMChainList;
  /* removes 1 chain (indicated by the index) from the list */
 OPERATOR Add;
  FPAR list BCSMChainList, chain BCSMChain;
 RETURNS BCSMChainList;
 START;
    RETURN list // Mkstring (chain);
 ENDOPERATOR;
 OPERATOR Del;
  FPAR list BCSMChainList, index Integer;
 RETURNS BCSMChainList;
 START;
    RETURN Substring(list, 1, index−1) // Substring (list, index+1, Length(list)−index);
 ENDOPERATOR;
ENDNEWTYPE;

/* LegToPIdArray is used for mappings from Leg to PId. */
NEWTYPE LegToPIdArray
  Array (LegType, PId);
ENDNEWTYPE;

/* CSToLegArray is used to map CS to a LegToPidArray */
NEWTYPE CSToLegArray
  Array (PId, LegToPIdArray);
ENDNEWTYPE;

/* CSAToLegArray is used to contain the legId of the controlling leg of the CSA−s */
NEWTYPE CSAInfo STRUCT
  controllingLeg LegType; /* legId of the controlling leg */
  isSurrogate Boolean; /* true if the controlling leg is surrogate */
ENDNEWTYPE;

NEWTYPE CSAInfoArray
  Array (CSAID, CSAInfo)
ENDNEWTYPE;

/* IndexSet is used to represent a list of indices in the TDP table */
NEWTYPE IndexSet
  String (Integer, emptystring)
ENDNEWTYPE;

NEWTYPE ControllingOrPassive
  LITERALS
    controlling, passive
ENDNEWTYPE;
```

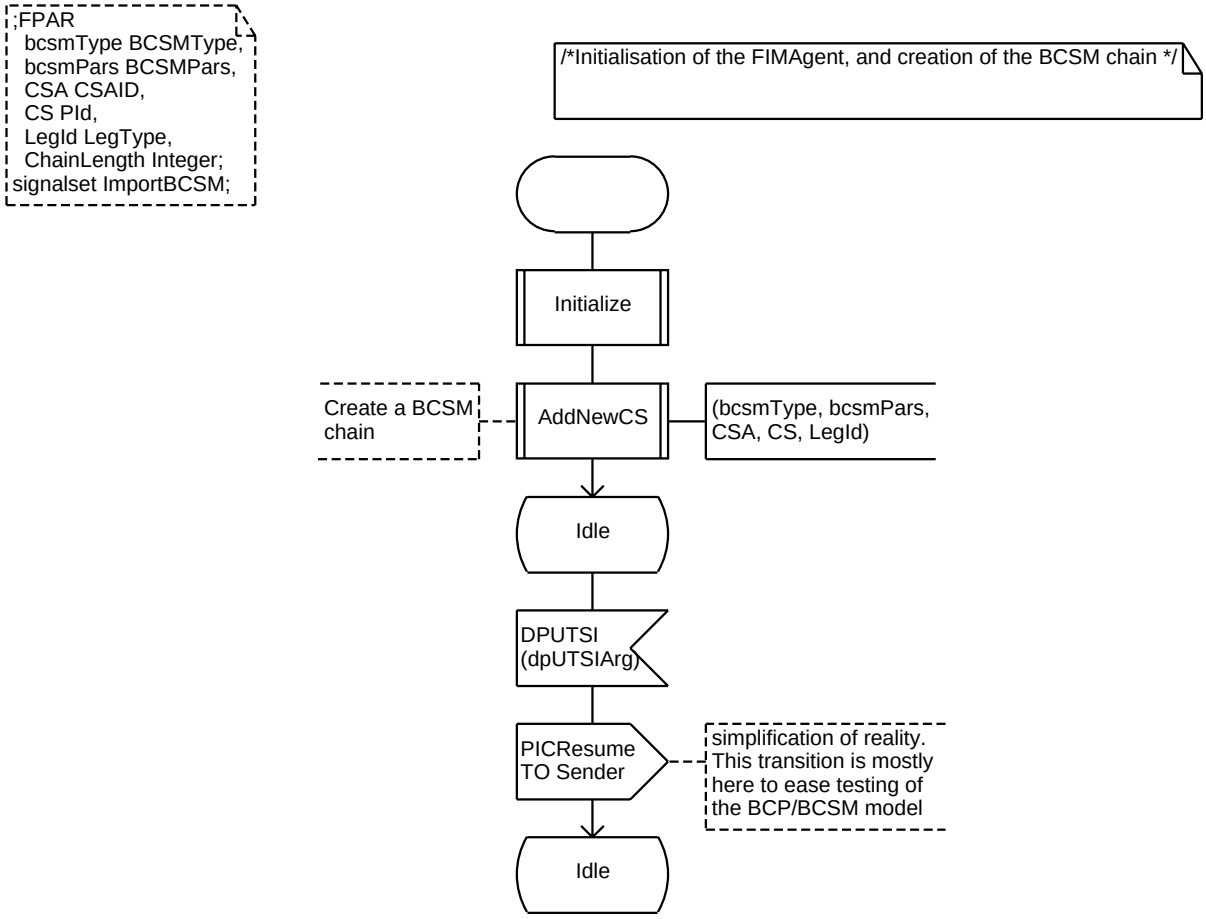
Process Type FIMAgent 5(13)

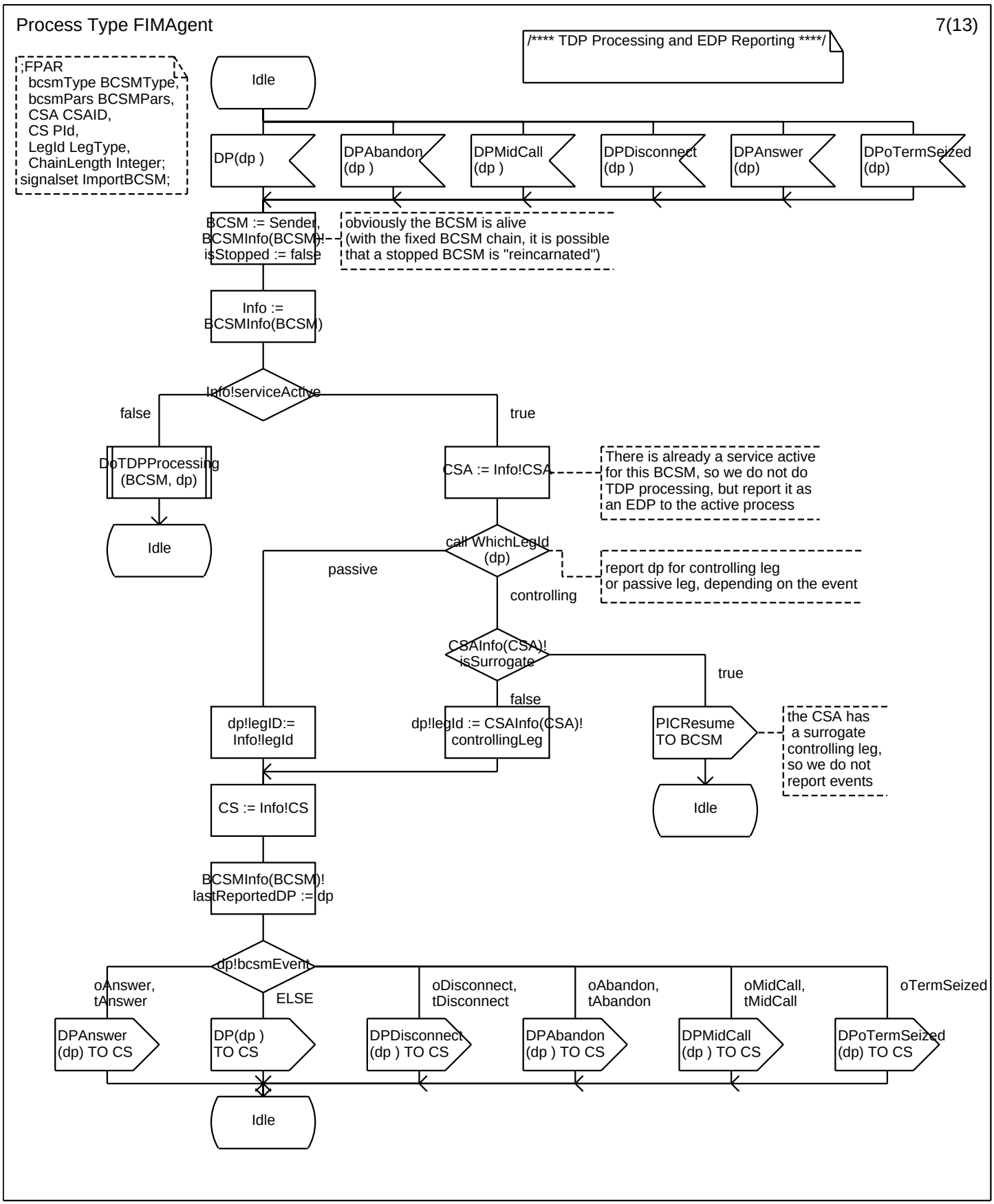
```
;FPAR
  bcsmType BCSMType,
  bcsmPars BCSMPars,
  CSA CSAID,
  CS PId,
  LegId LegType,
  ChainLength Integer;
signalset ImportBCSM;
```

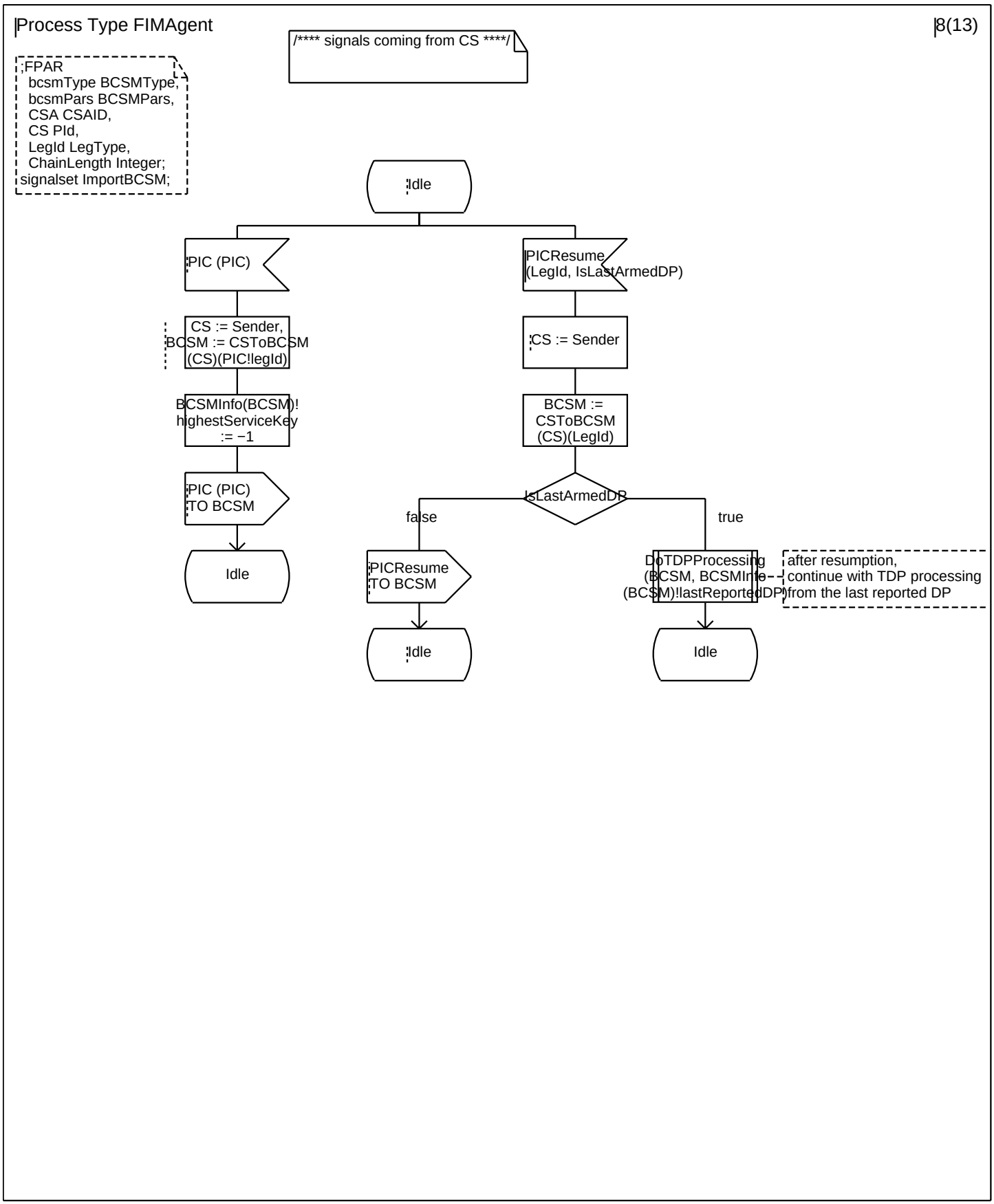
```
/* dummy CSA */
SYNONYM DummyCSA CSAID = 1;
```

```
/* Variable Declarations */

DCL
 /* 'state' variables (variables that are maintained throughout
    the lifetime of a FIMAgent */
 BCSMInfo BCSMInfoArray,
   /* keeps information about all BCSMs served by FIMAgent */
 CSToBCSM CSToLegArray,
   /* keeps mapping from CS+LegId to corresponding BCSM */
 CSAInfo CSAInfoArray,
   /* info about CSA−s */
 ChainList BCSMChainList,
   /* keeps all BCSM chains */
 TDPTable TDPTableType,
   /* keeps the TDP table per call context (= FIMAgent) */
 FIM_Controller PId,

 /* temporary variables (used in parameters, loops, etc.) */
 BCSM PId,
 Chain BCSMChain,
 ConnPointBCSM PId,
 CSAID CSAID,
 CSAInf CSAInfo,
 dp DPArg,
 dpUTSIArg DPUTSIArg,
 Exception Boolean,
 Info BCSMInfo,
 InvokeID InvokeID,
 IsControlling Boolean,
 IsLastArmedDP Boolean,
 LegNr Integer,
 NewCSAID CSAID,
 NewLegId LegType,
 OldLegId LegType,
 Party PartyType,
 pic PICArg,
 Profile Integer,
 highestServiceKey Integer,
 SourceBCSM PId,
 SourceCS PId,
 SSPArg SetServiceProfileArg,
 TargetBCSM PId,
 TargetCS PId;
```

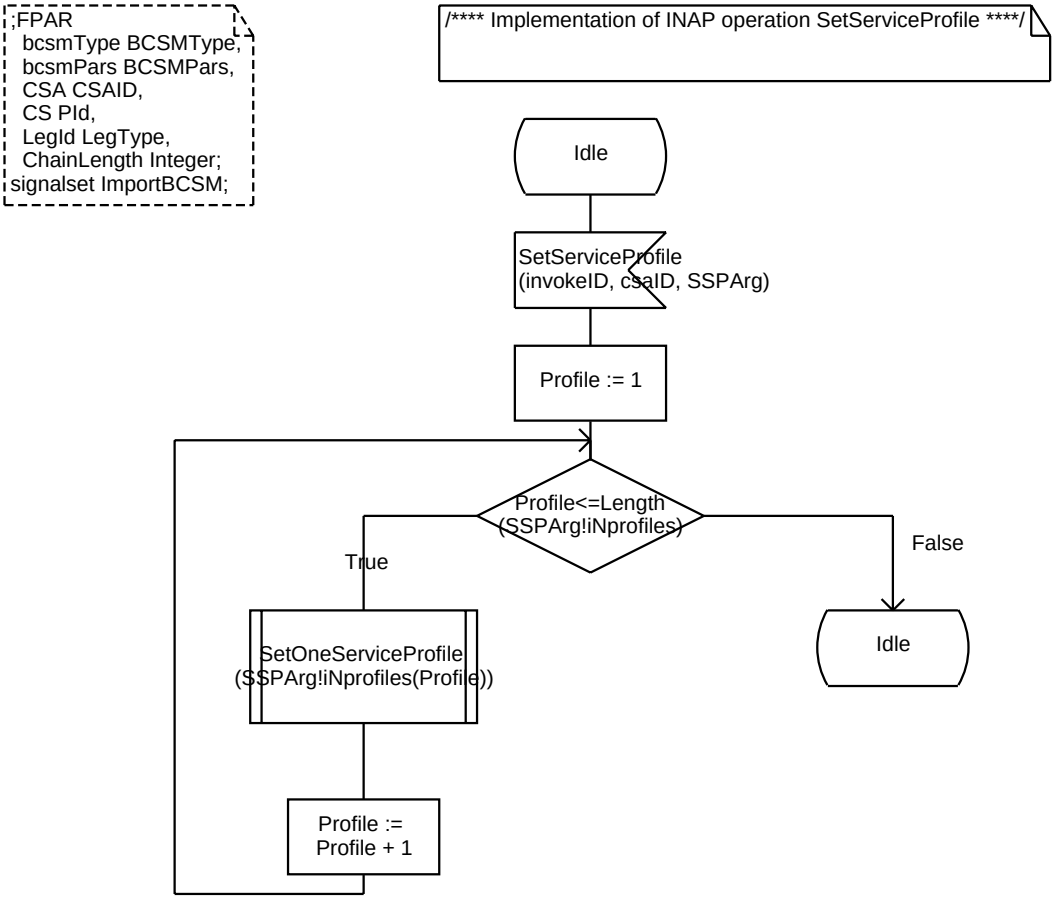
Process Type FIMAgent 6(13)



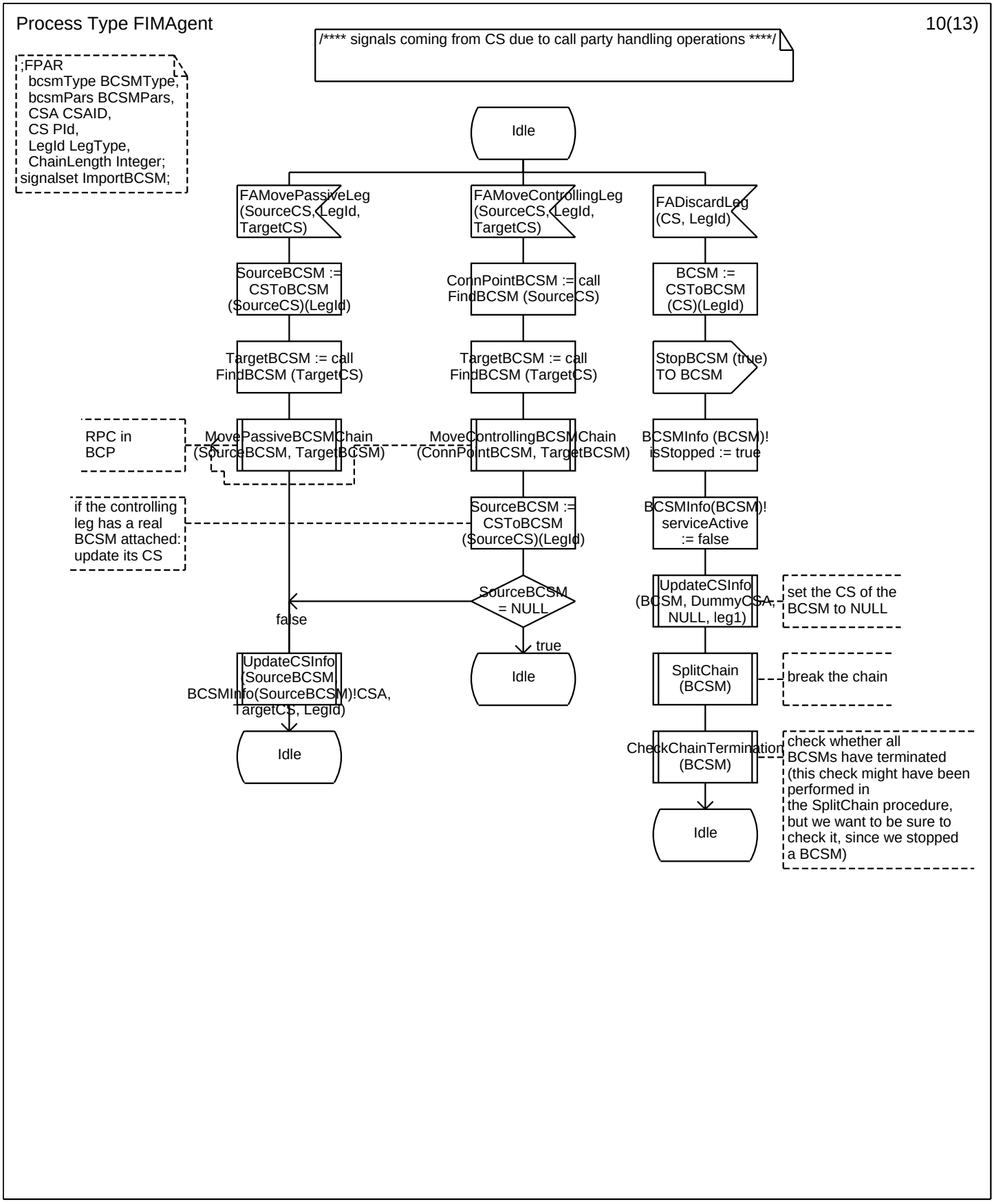

Process Type FIMAgent
7(13)
/**** TDP Processing and EDP Reporting ****/
;FPAR
bcsmType BCSMType,
bcsmPars BCSMPars,
CSA CSAID,
CS PId,
LegId LegType,
ChainLength Integer;
signalset ImportBCSM;
Idle
DP(dp)
DPAbandon (dp)
DPMidCall (dp)
DPDisconnect (dp)
DPAnswer (dp)
DPoTermSeized (dp)
BCSM := Sender, BCSMInfo(BCSM)! isStopped := false
obviously the BCSM is alive (with the fixed BCSM chain, it is possible that a stopped BCSM is "reincarnated")
Info := BCSMInfo(BCSM)
Info!serviceActive
false
true
DoTDPProcessing (BCSM, dp)
Idle
CSA := Info!CSA
There is already a service active for this BCSM, so we do not do TDP processing, but report it as an EDP to the active process
call WhichLegId (dp)
passive
controlling
report dp for controlling leg or passive leg, depending on the event
CSAInfo(CSA)! isSurrogate
true
false
dp!legID:= Info!legId
dp!legId := CSAInfo(CSA)! controllingLeg
PICResume TO BCSM
the CSA has a surrogate controlling leg, so we do not report events
Idle
CS := Info!CS
BCSMInfo(BCSM)! lastReportedDP := dp
dp!bcsmEvent
oAnswer, tAnswer
ELSE
oDisconnect, tDisconnect
oAbandon, tAbandon
oMidCall, tMidCall
oTermSeized
DPAnswer (dp) TO CS
DP(dp) TO CS
DPDisconnect (dp) TO CS
DPAbandon (dp) TO CS
DPMidCall (dp) TO CS
DPoTermSeized (dp) TO CS
Idle


Process Type FIMAgent
8(13)
/**** signals coming from CS ****/
;FPAR
bcsmType BCSMType,
bcsmPars BCSMPars,
CSA CSAID,
CS PId,
LegId LegType,
ChainLength Integer;
signalset ImportBCSM;
Idle
PIC (PIC)
PICResume
(LegId, IsLastArmedDP)
CS := Sender,
BCSM := CSToBCSM
(CS)(PIC!legId)
CS := Sender
BCSMInfo(BCSM)!
highestServiceKey
:= −1
BCSM :=
CSToBCSM
(CS)(LegId)
PIC (PIC)
TO BCSM
IsLastArmedDP
false
true
Idle
PICResume
TO BCSM
DoTDPProcessing
(BCSM, BCSMInfo
(BCSM)!lastReportedDP)
after resumption,
continue with TDP processing
from the last reported DP
Idle
Idle

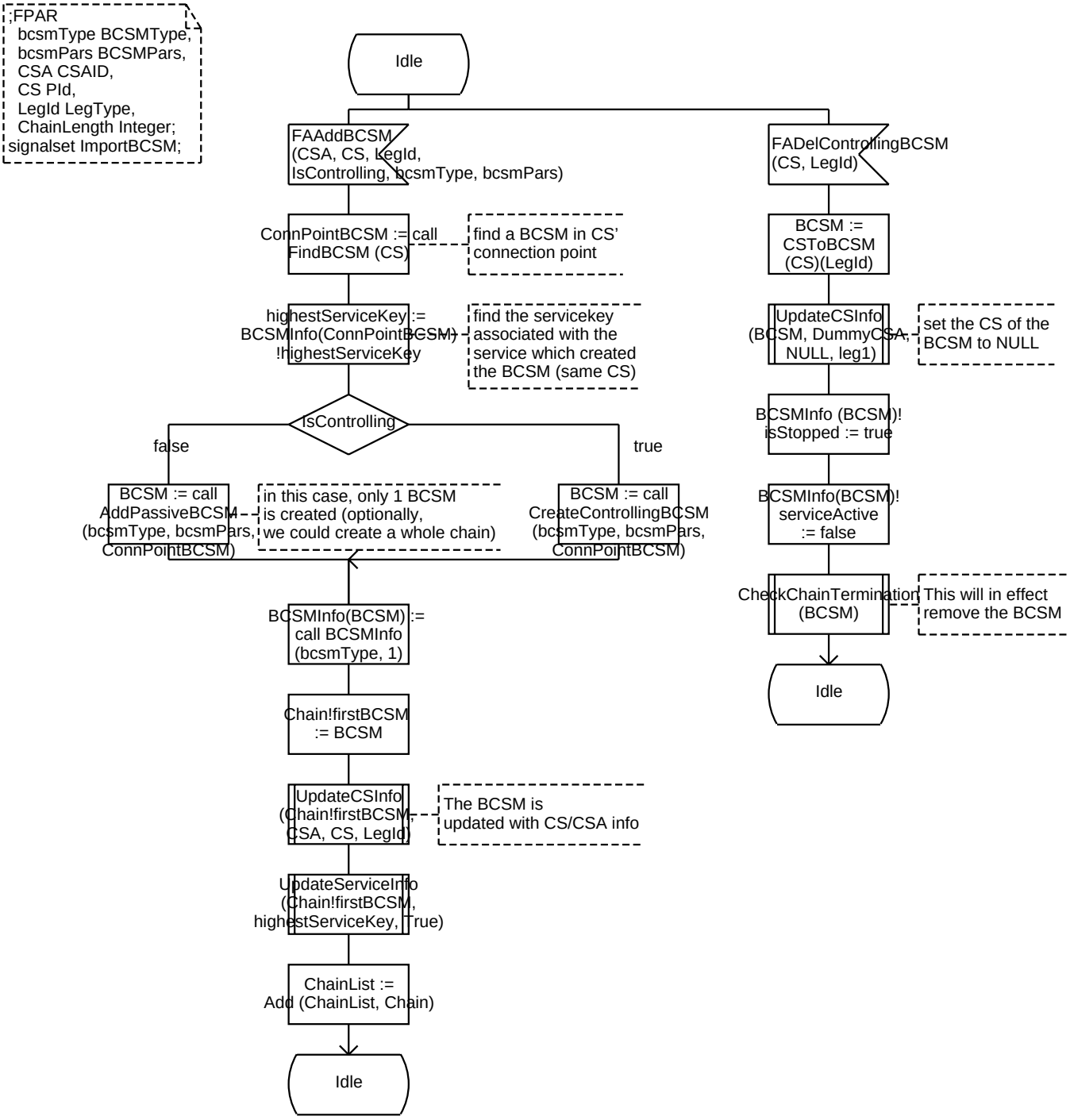
Process Type FIMAgent 9(13)



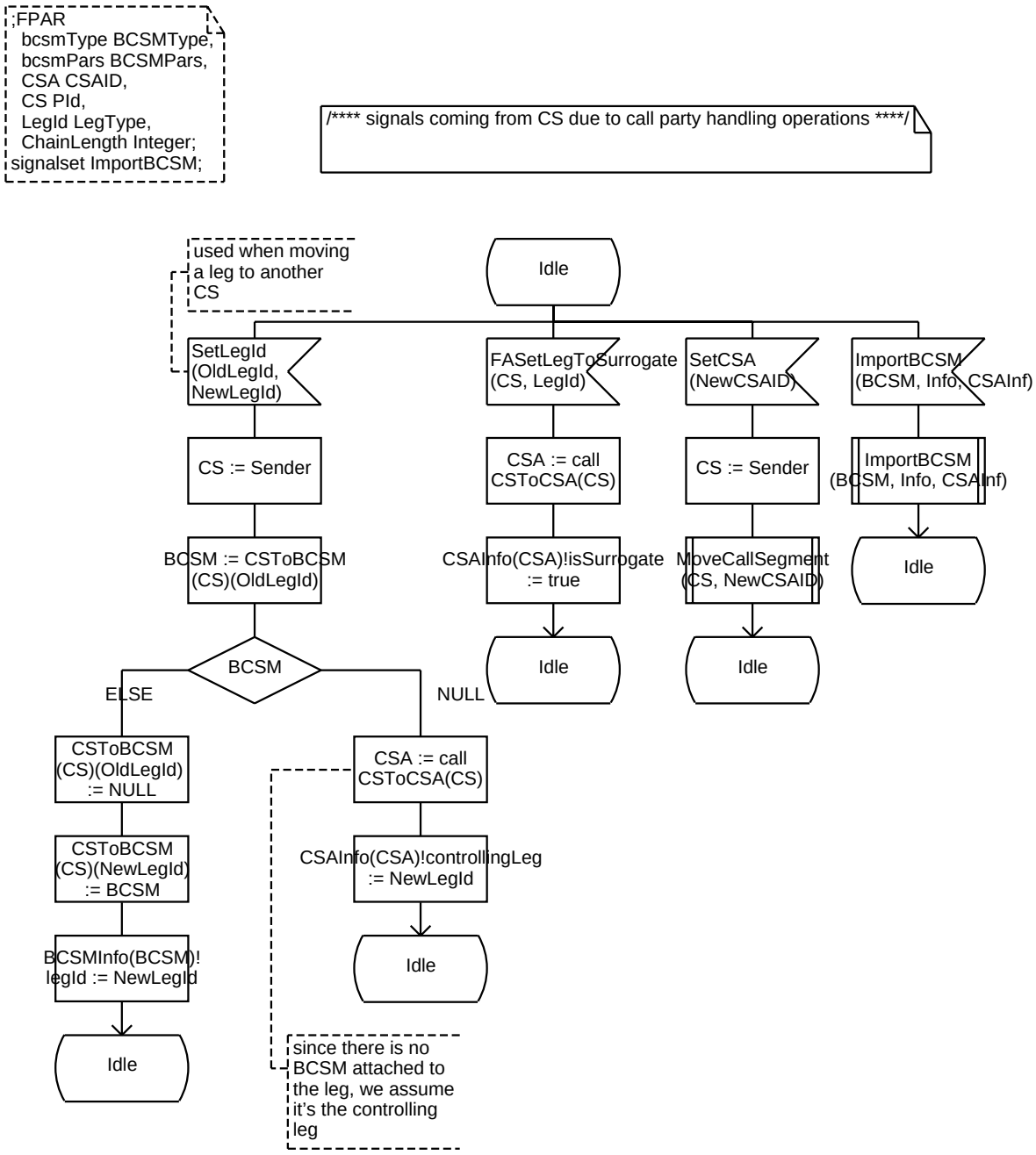
Process Type FIMAgent 10(13)

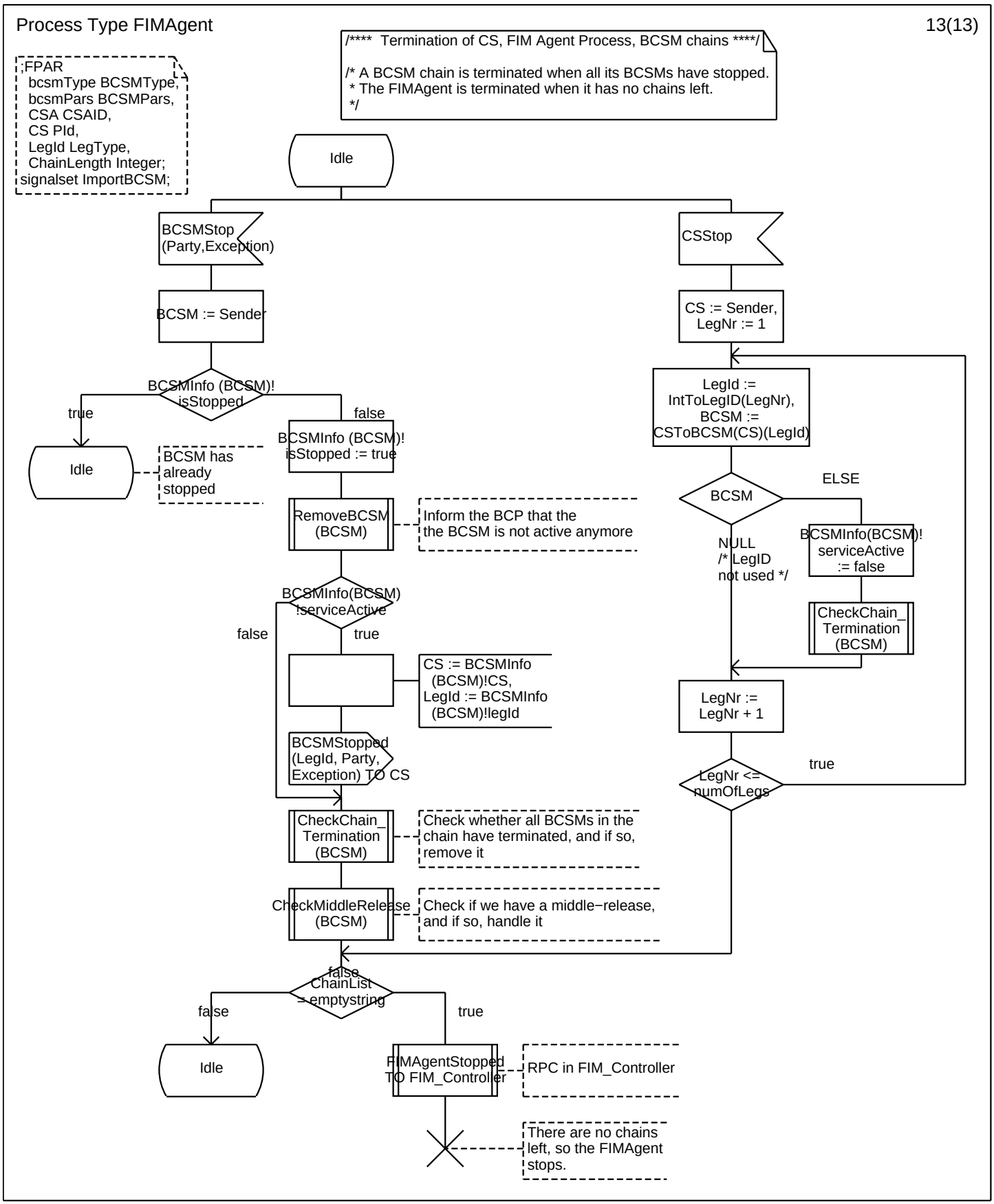


Process Type FIMAgent 11(13)

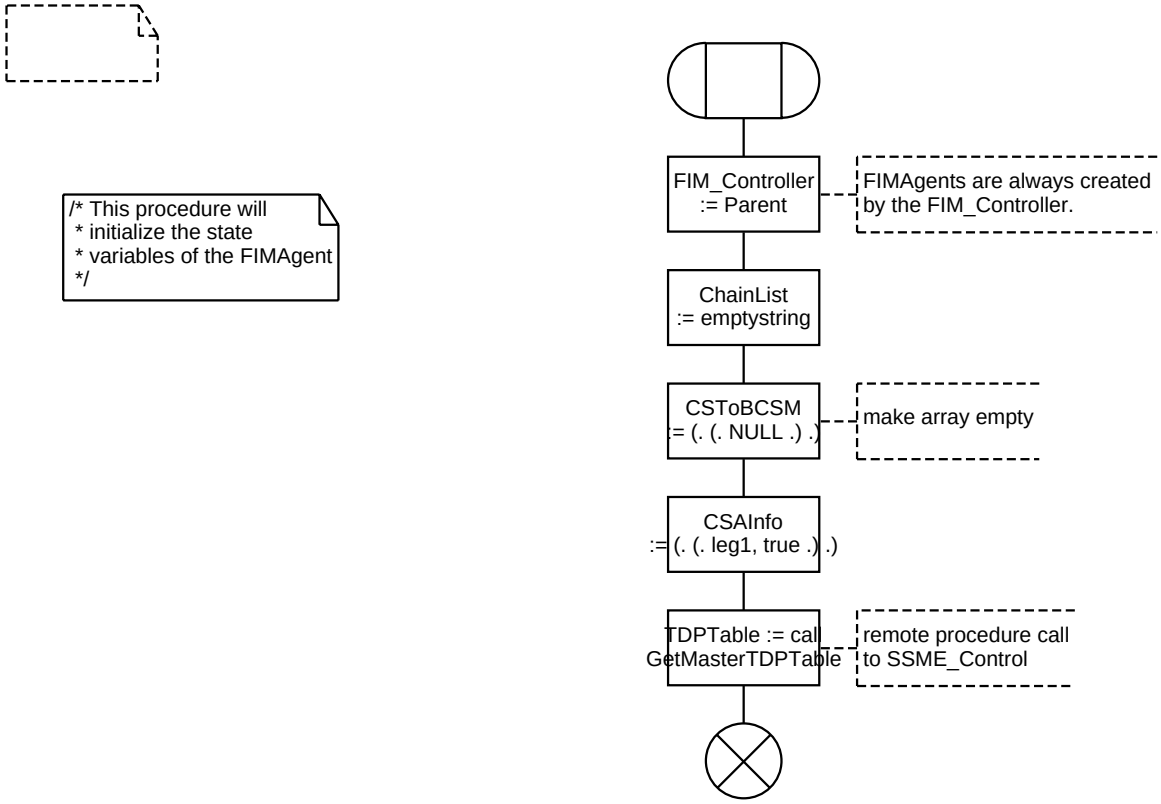


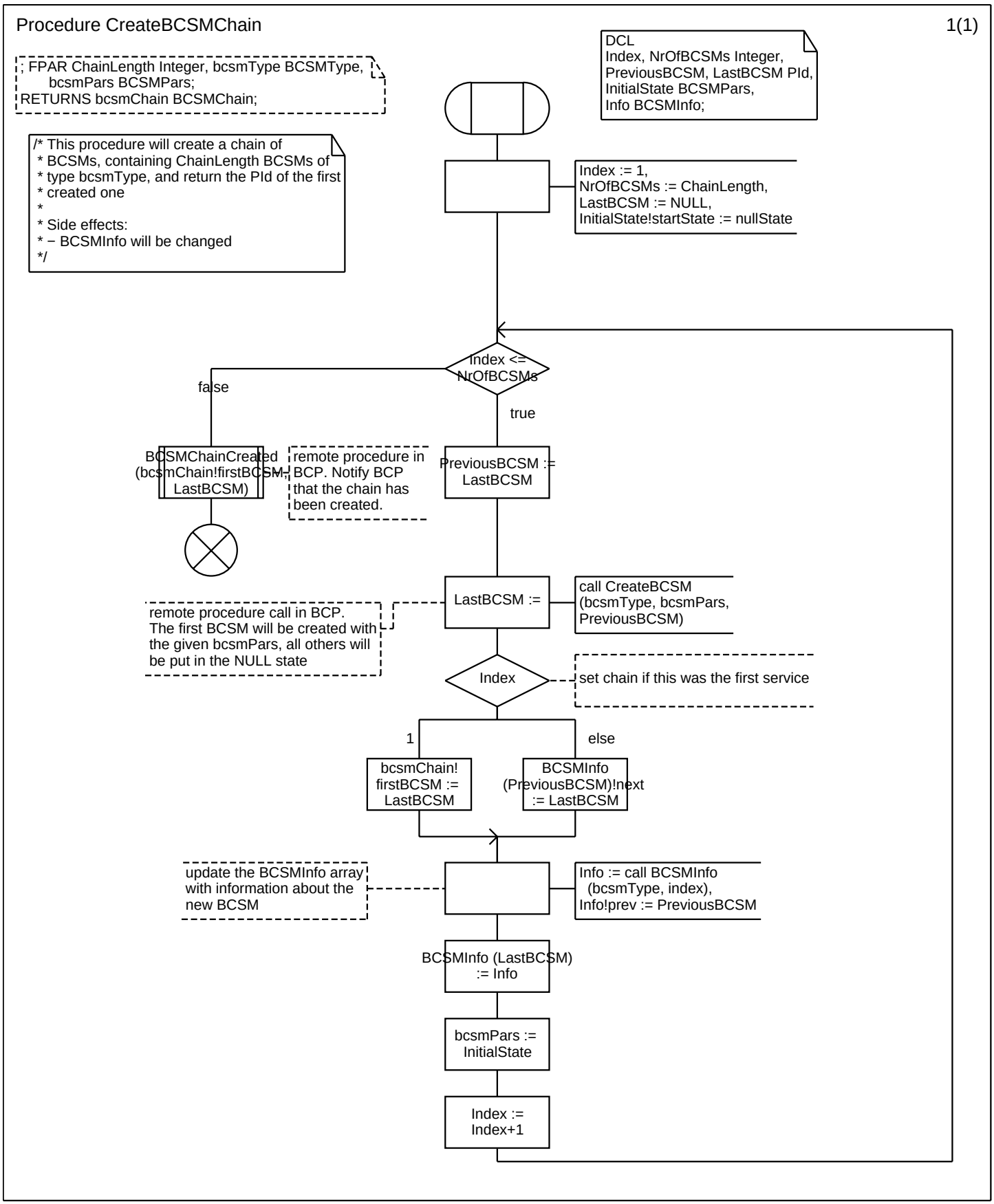
Process Type FIMAgent 12(13)



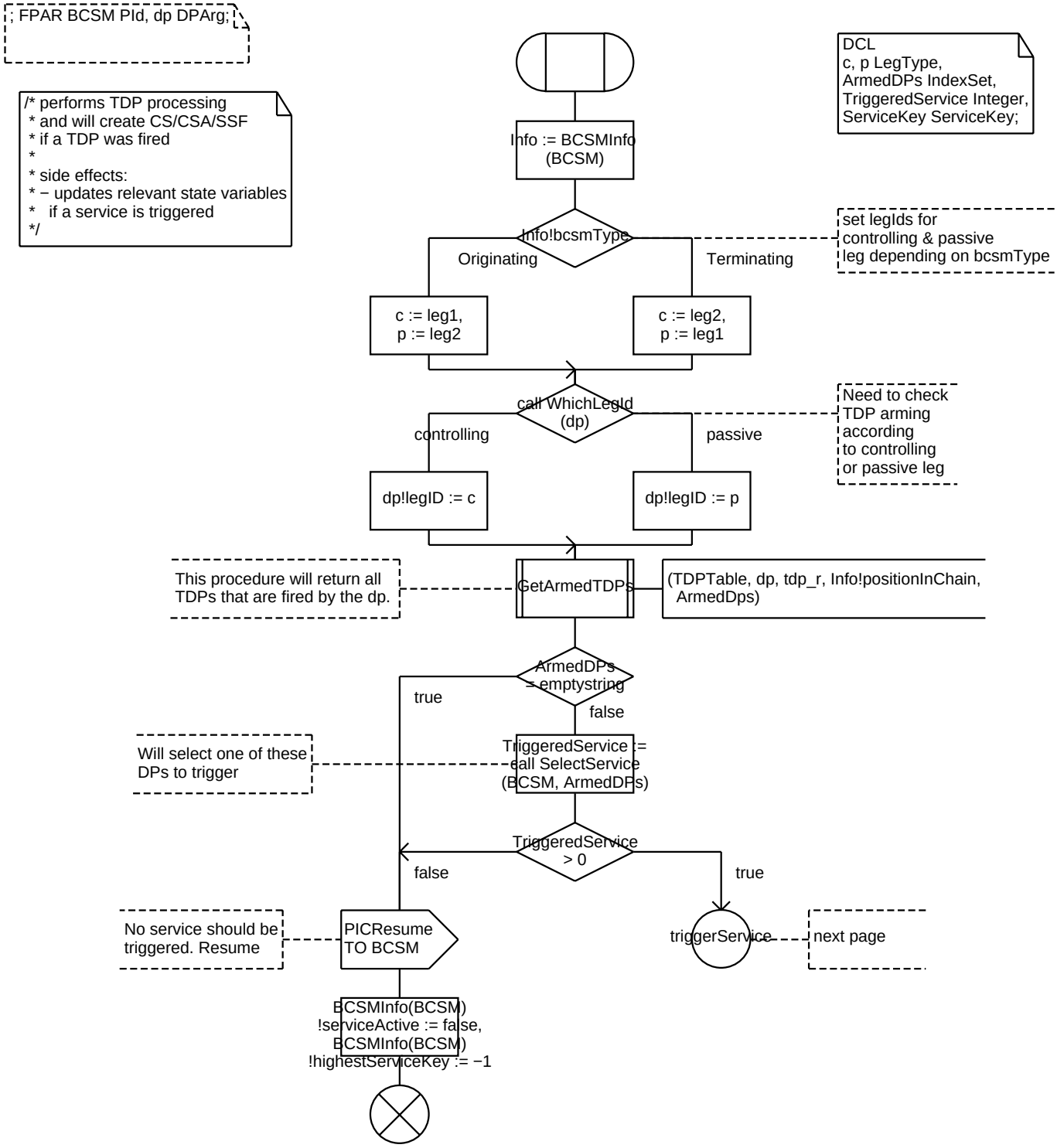

Process Type FIMAgent
13(13)
/**** Termination of CS, FIM Agent Process, BCSM chains ****/
/* A BCSM chain is terminated when all its BCSMs have stopped.
* The FIMAgent is terminated when it has no chains left.
*/
;FPAR
bcsmType BCSMType,
bcsmPars BCSMPars,
CSA CSAID,
CS PId,
LegId LegType,
ChainLength Integer;
signalset ImportBCSM;
Idle
BCSMStop (Party,Exception)
CSStop
BCSM := Sender
CS := Sender, LegNr := 1
BCSMInfo (BCSM)! isStopped
true
false
Idle
BCSM has already stopped
BCSMInfo (BCSM)! isStopped := true
RemoveBCSM (BCSM)
Inform the BCP that the the BCSM is not active anymore
BCSMInfo(BCSM) !serviceActive
false
true
CS := BCSMInfo (BCSM)!CS, LegId := BCSMInfo (BCSM)!legId
BCSMStopped (LegId, Party, Exception) TO CS
CheckChain_ Termination (BCSM)
Check whether all BCSMs in the chain have terminated, and if so, remove it
CheckMiddleRelease (BCSM)
Check if we have a middle−release, and if so, handle it
LegId := IntToLegID(LegNr), BCSM := CSToBCSM(CS)(LegId)
BCSM
ELSE
NULL /* LegID not used */
BCSMInfo(BCSM)! serviceActive := false
CheckChain_ Termination (BCSM)
LegNr := LegNr + 1
LegNr <= numOfLegs
true
false
ChainList = emptystring
false
true
Idle
FIMAgentStopped TO FIM_Controller
RPC in FIM_Controller
There are no chains left, so the FIMAgent stops.

Procedure Initialize 1(1)

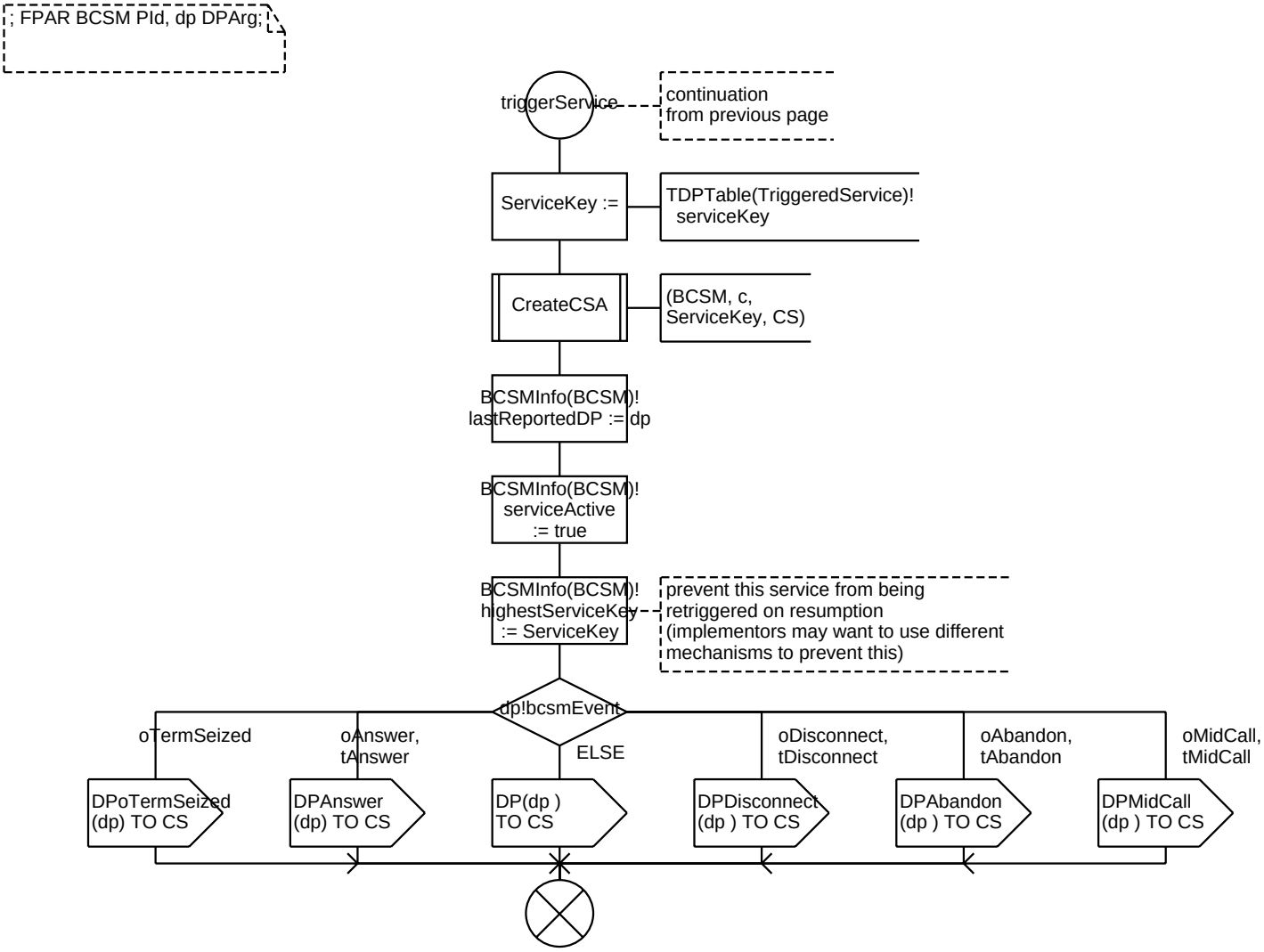



Procedure CreateBCSMChain
1(1)
; FPAR ChainLength Integer, bcsmType BCSMType,
bcsmPars BCSMPars;
RETURNS bcsmChain BCSMChain;
DCL
Index, NrOfBCSMs Integer,
PreviousBCSM, LastBCSM PId,
InitialState BCSMPars,
Info BCSMInfo;
/* This procedure will create a chain of
* BCSMs, containing ChainLength BCSMs of
* type bcsmType, and return the PId of the first
* created one
*
* Side effects:
* − BCSMInfo will be changed
*/
Index := 1,
NrOfBCSMs := ChainLength,
LastBCSM := NULL,
InitialState!startState := nullState
Index <= NrOfBCSMs
false
true
BCSMChainCreated (bcsmChain!firstBCSM, LastBCSM)
remote procedure in BCP. Notify BCP that the chain has been created.
PreviousBCSM := LastBCSM
LastBCSM :=
call CreateBCSM (bcsmType, bcsmPars, PreviousBCSM)
remote procedure call in BCP. The first BCSM will be created with the given bcsmPars, all others will be put in the NULL state
Index
set chain if this was the first service
1
else
bcsmChain! firstBCSM := LastBCSM
BCSMInfo (PreviousBCSM)!next := LastBCSM
update the BCSMInfo array with information about the new BCSM
Info := call BCSMInfo (bcsmType, index),
Info!prev := PreviousBCSM
BCSMInfo (LastBCSM) := Info
bcsmPars := InitialState
Index := Index+1

Procedure DoTDPProcessing 1(2)



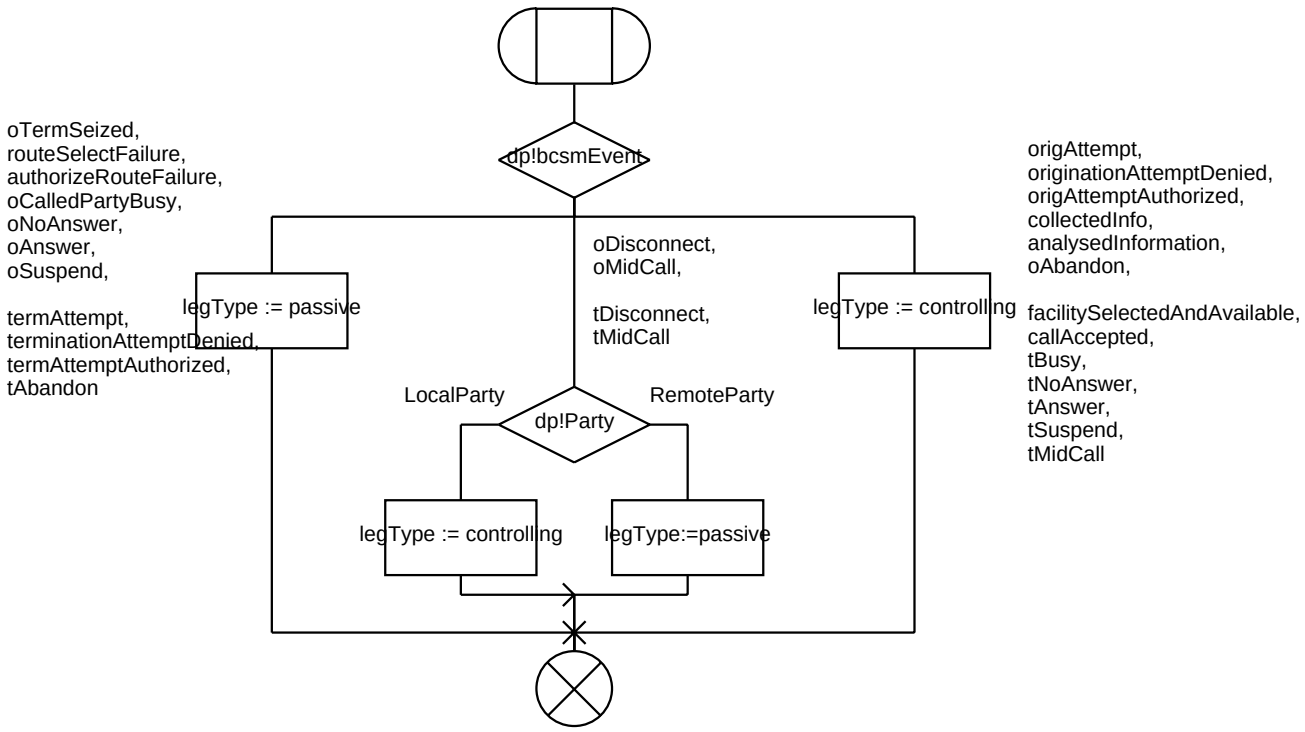
Procedure DoTDPProcessing 2(2)



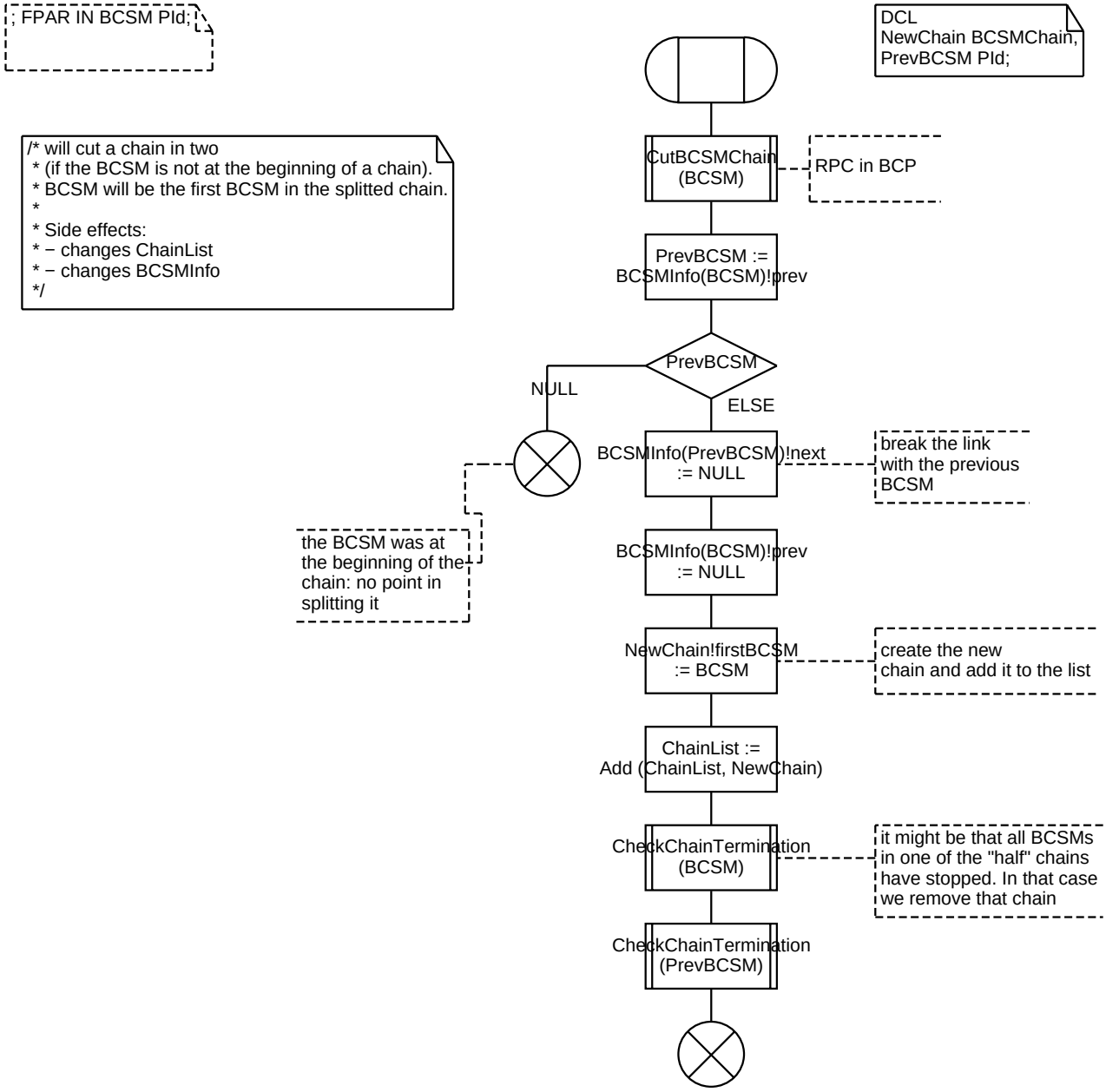
Procedure <<Process Type FIMAgent>> WhichLegID 1(1)

```
;FPAR
  IN dp DPArg;
  RETURNS legType ControllingOrPassive;
```

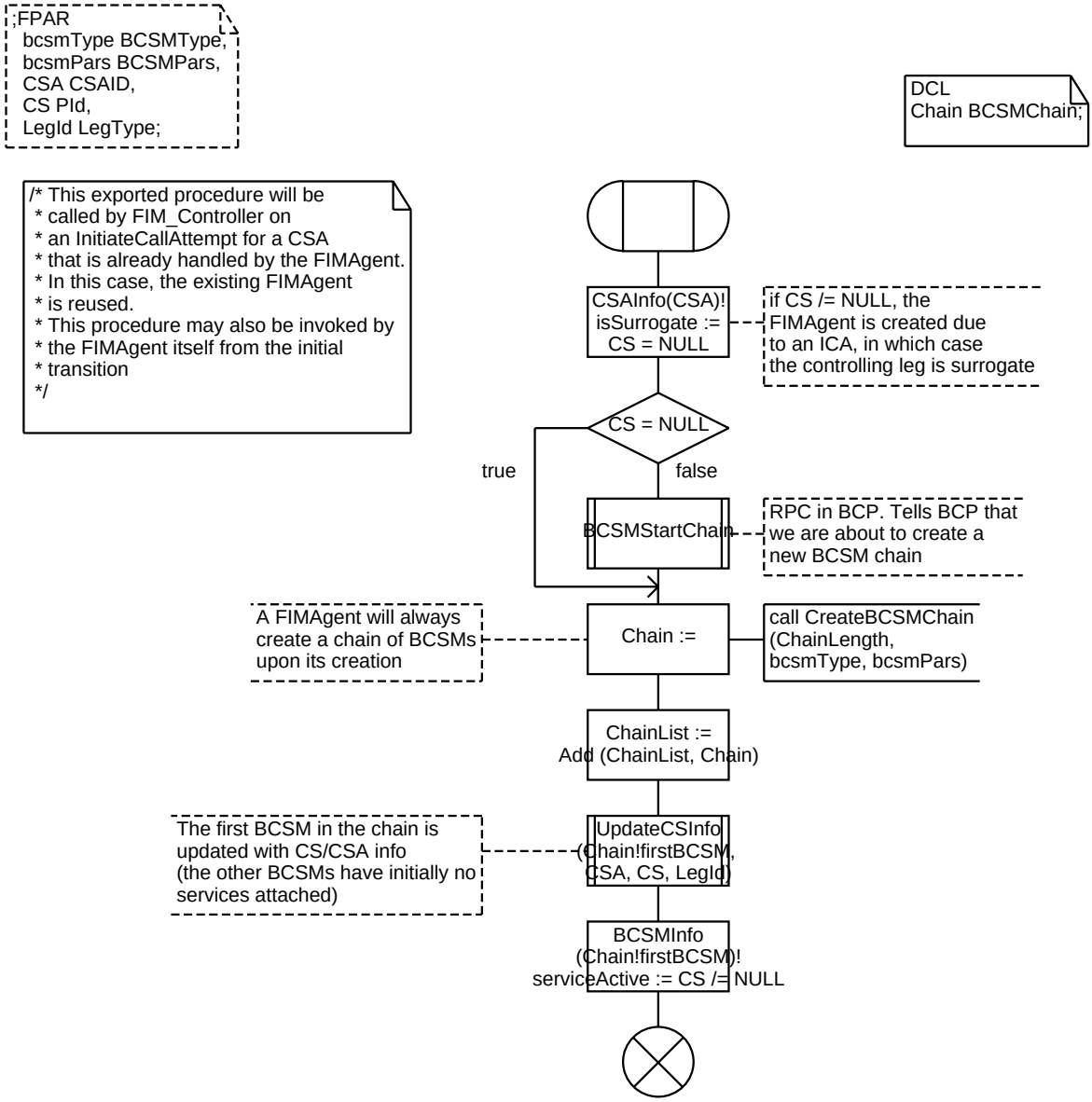
```
/* Determine the LegID  for which the event should be reported.
Depending on the event,
controlling or passive will be returned as legType.
Actual legID that is reported to SCF may in some
cases be different; will depend on CS */
```



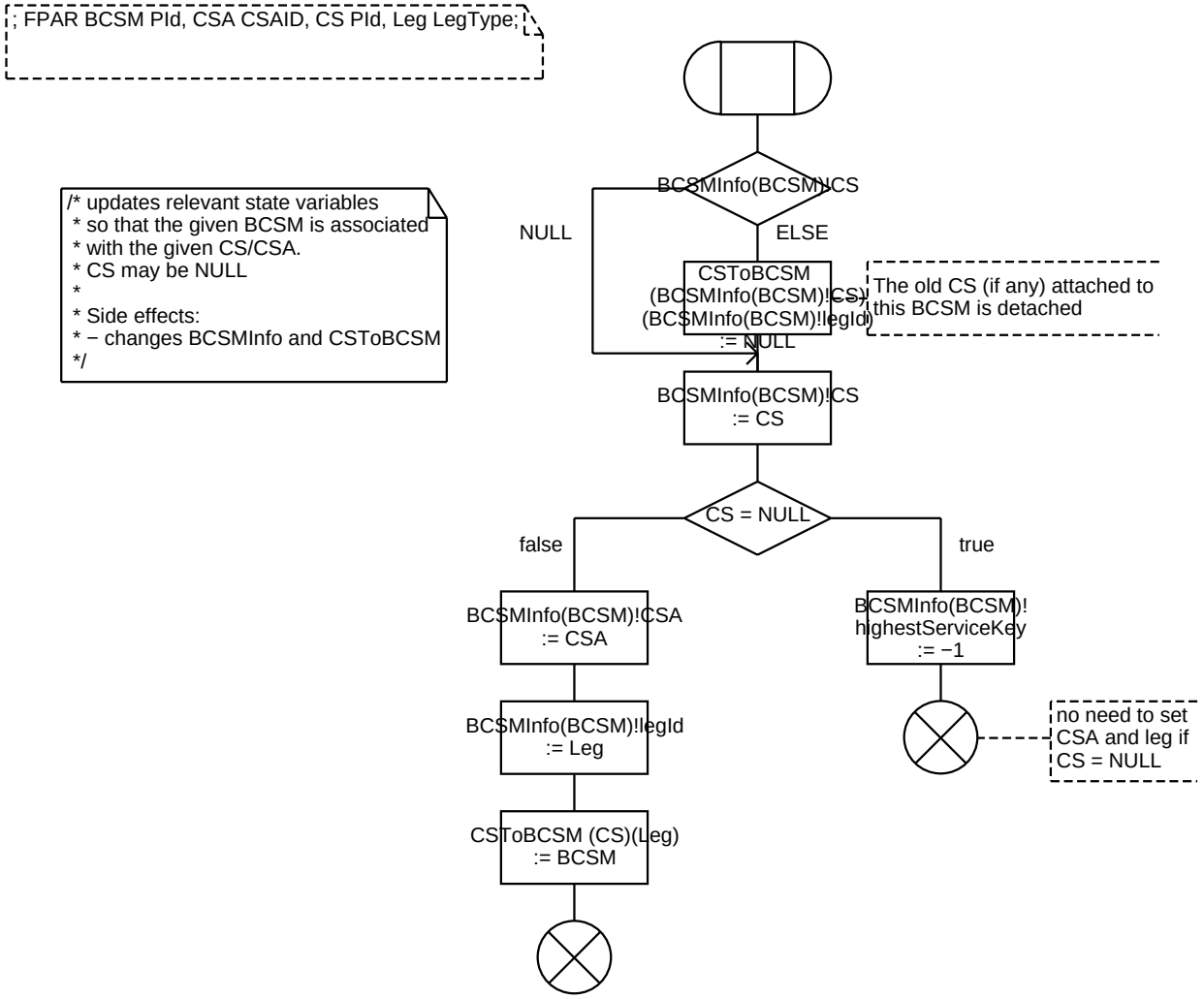
Procedure SplitChain 1(1)



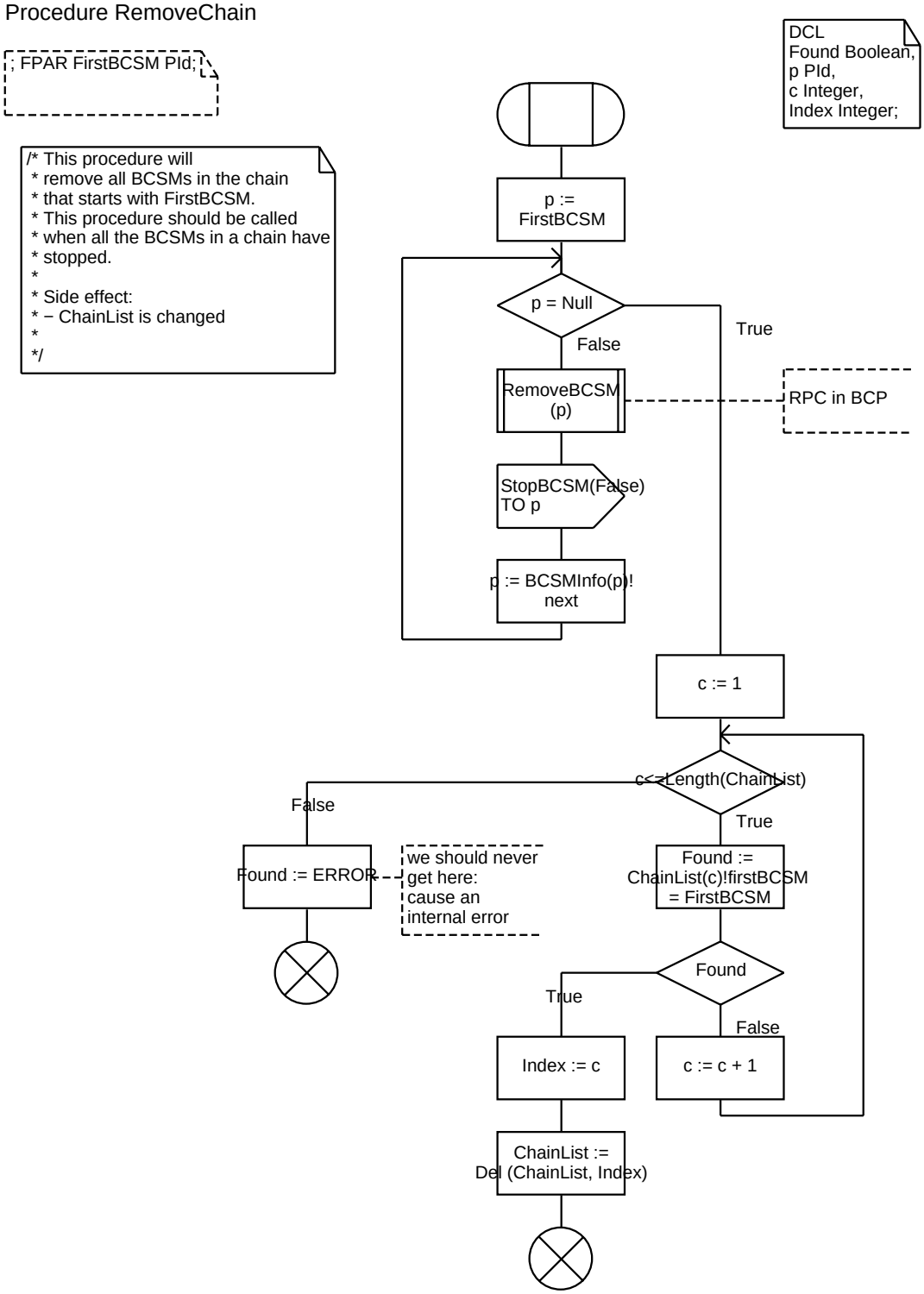
Procedure AddNewCS 1(1)

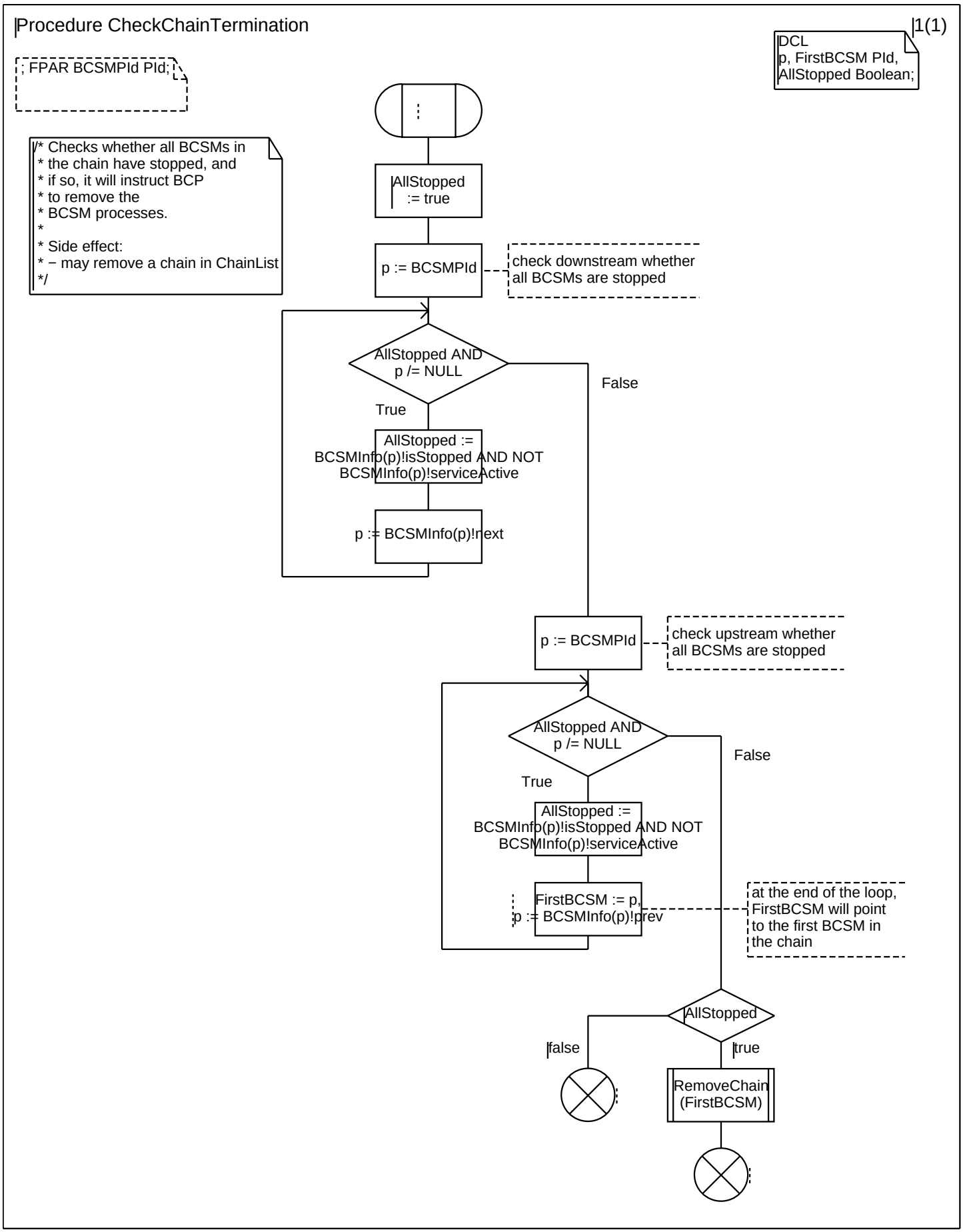


Procedure UpdateCSInfo 1(1)

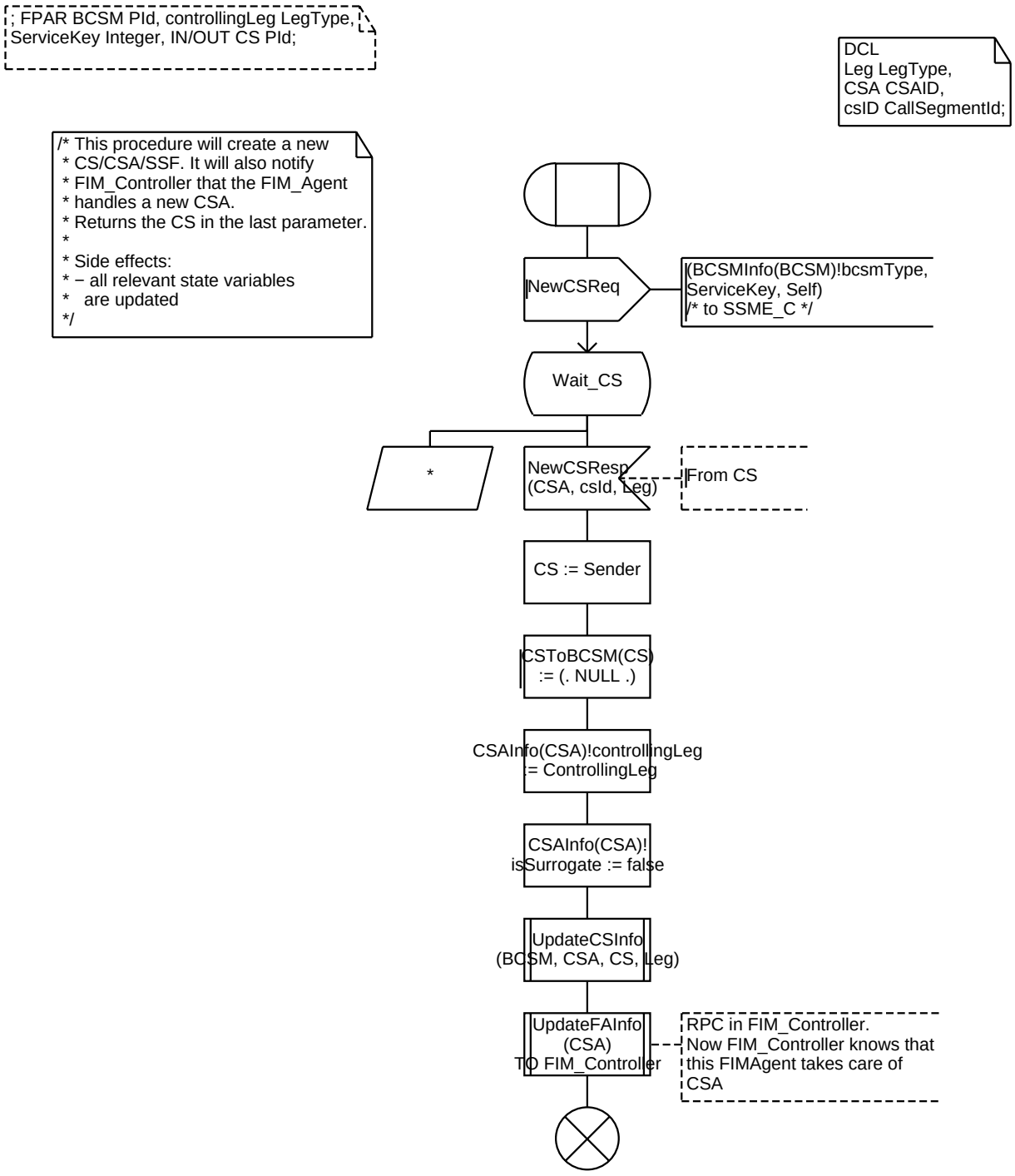


Procedure RemoveChain 1(1)

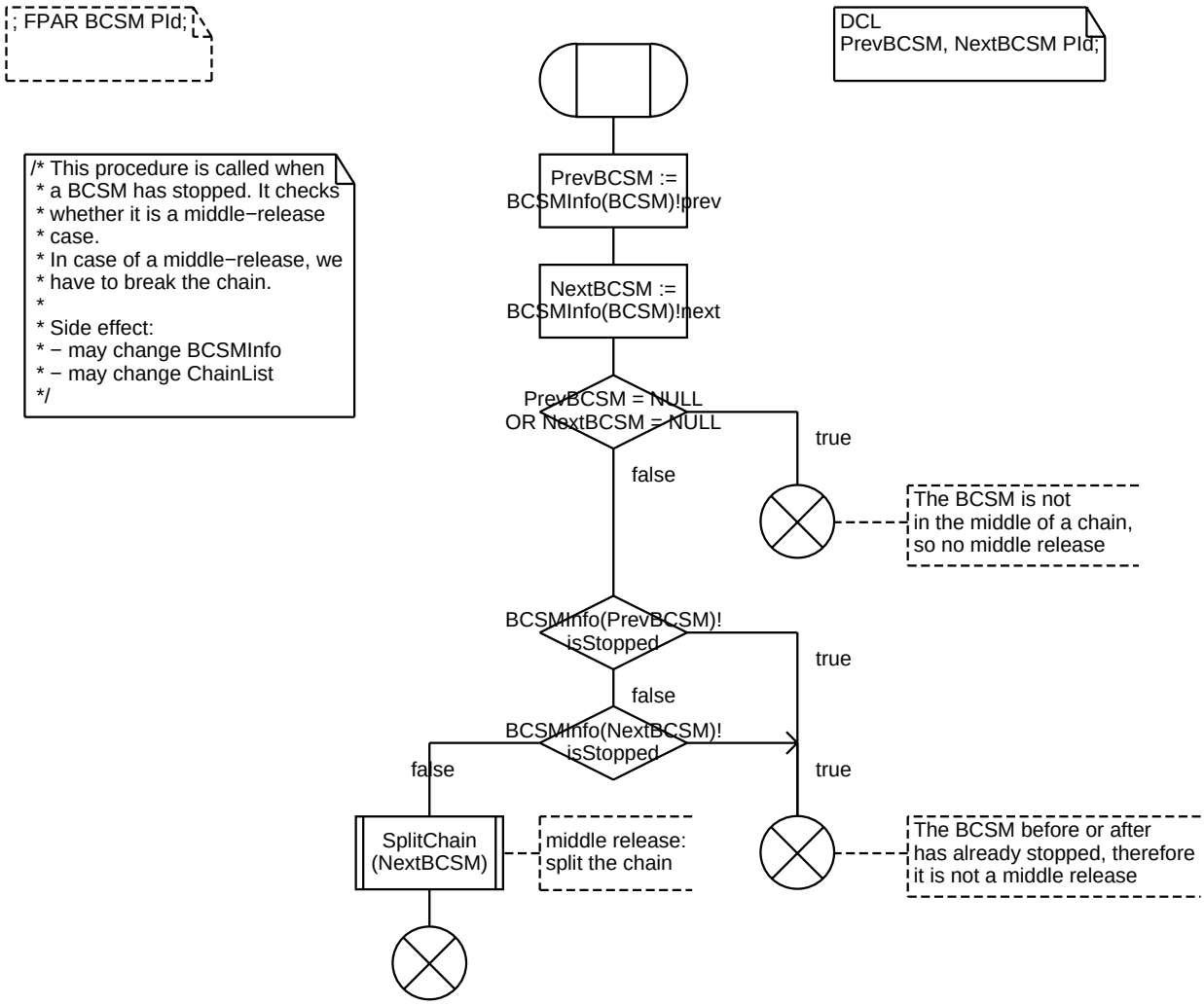



Procedure CheckChainTermination
1(1)
DCL
p, FirstBCSM PId,
AllStopped Boolean;
; FPAR BCSMPId PId;
/* Checks whether all BCSMs in
* the chain have stopped, and
* if so, it will instruct BCP
* to remove the
* BCSM processes.
*
* Side effect:
* − may remove a chain in ChainList
*/
AllStopped
:= true
p := BCSMPId
check downstream whether
all BCSMs are stopped
AllStopped AND
p /= NULL
False
True
AllStopped :=
BCSMInfo(p)!isStopped AND NOT
BCSMInfo(p)!serviceActive
p := BCSMInfo(p)!next
p := BCSMPId
check upstream whether
all BCSMs are stopped
AllStopped AND
p /= NULL
False
True
AllStopped :=
BCSMInfo(p)!isStopped AND NOT
BCSMInfo(p)!serviceActive
FirstBCSM := p,
p := BCSMInfo(p)!prev
at the end of the loop,
FirstBCSM will point
to the first BCSM in
the chain
AllStopped
false
true
RemoveChain
(FirstBCSM)

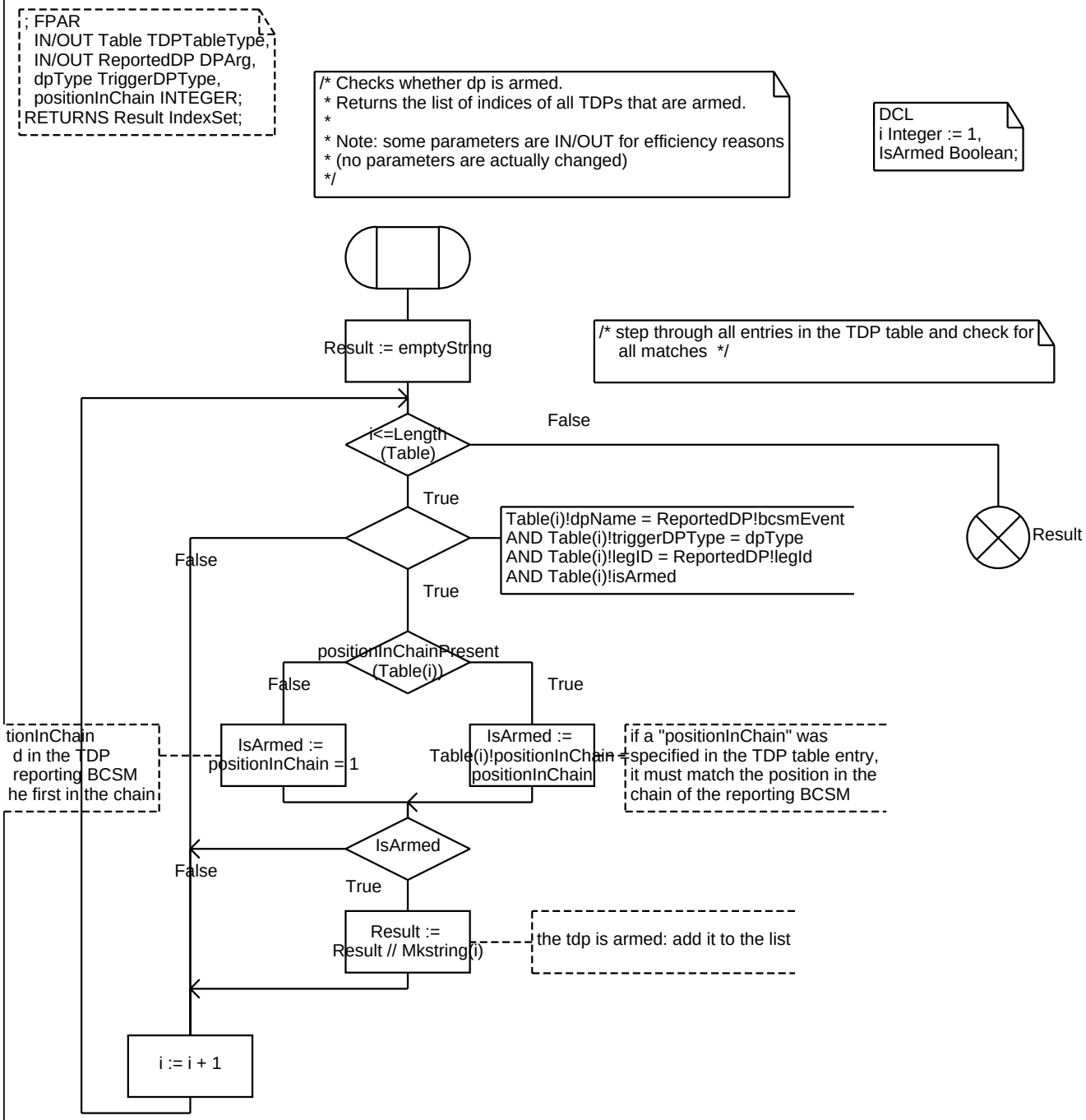
Procedure <<Process Type FIMAgent>> CreateCSA 1(1)



Procedure CheckMiddleRelease 1(1)



Procedure GetArmedTDPs 1(1)

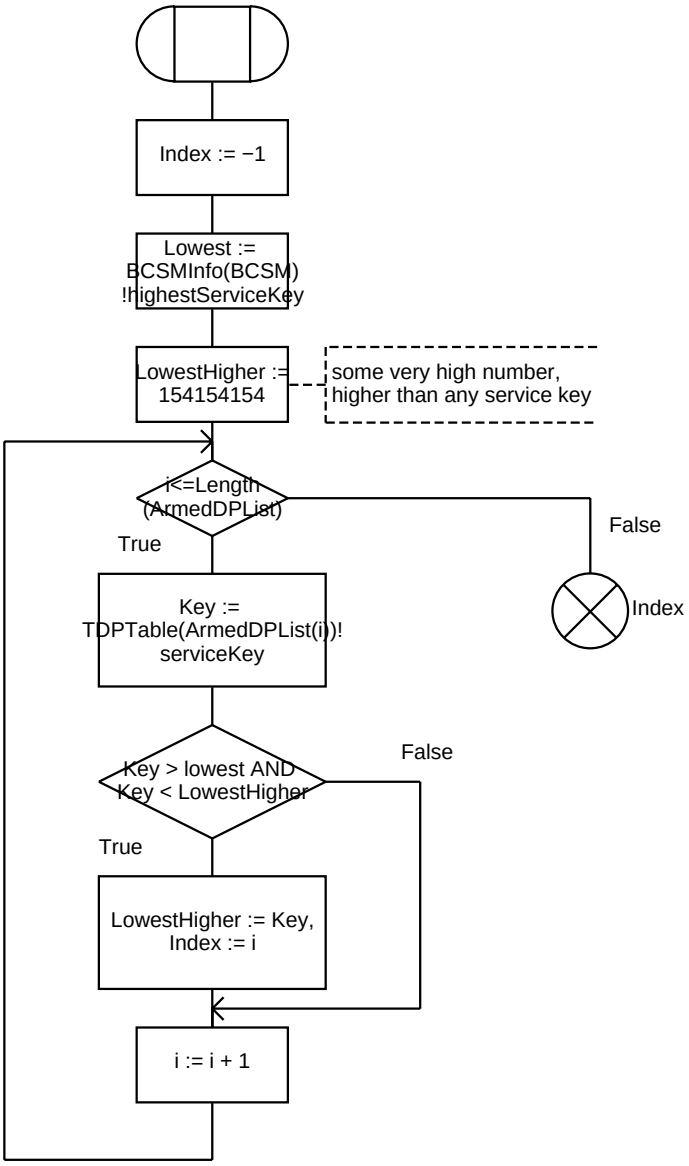


Procedure SelectService 1(1)

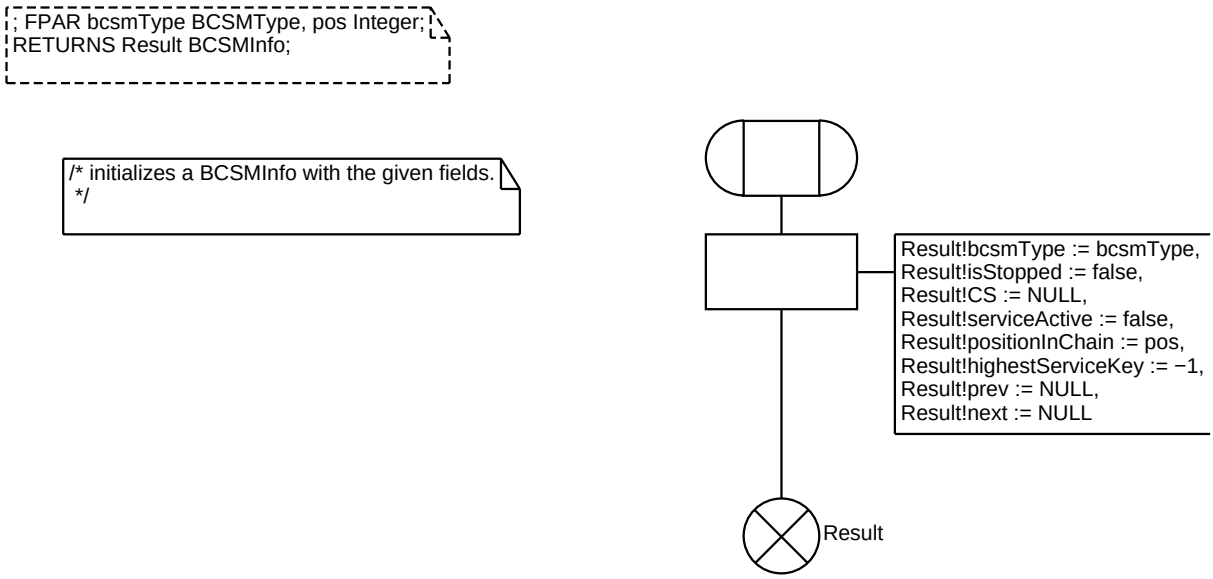
; FPAR BCSM PId, ArmedDPList IndexSet;
RETURNS Index Integer;

DCL
Key, Lowest, LowestHigher Integer,
i Integer := 1;

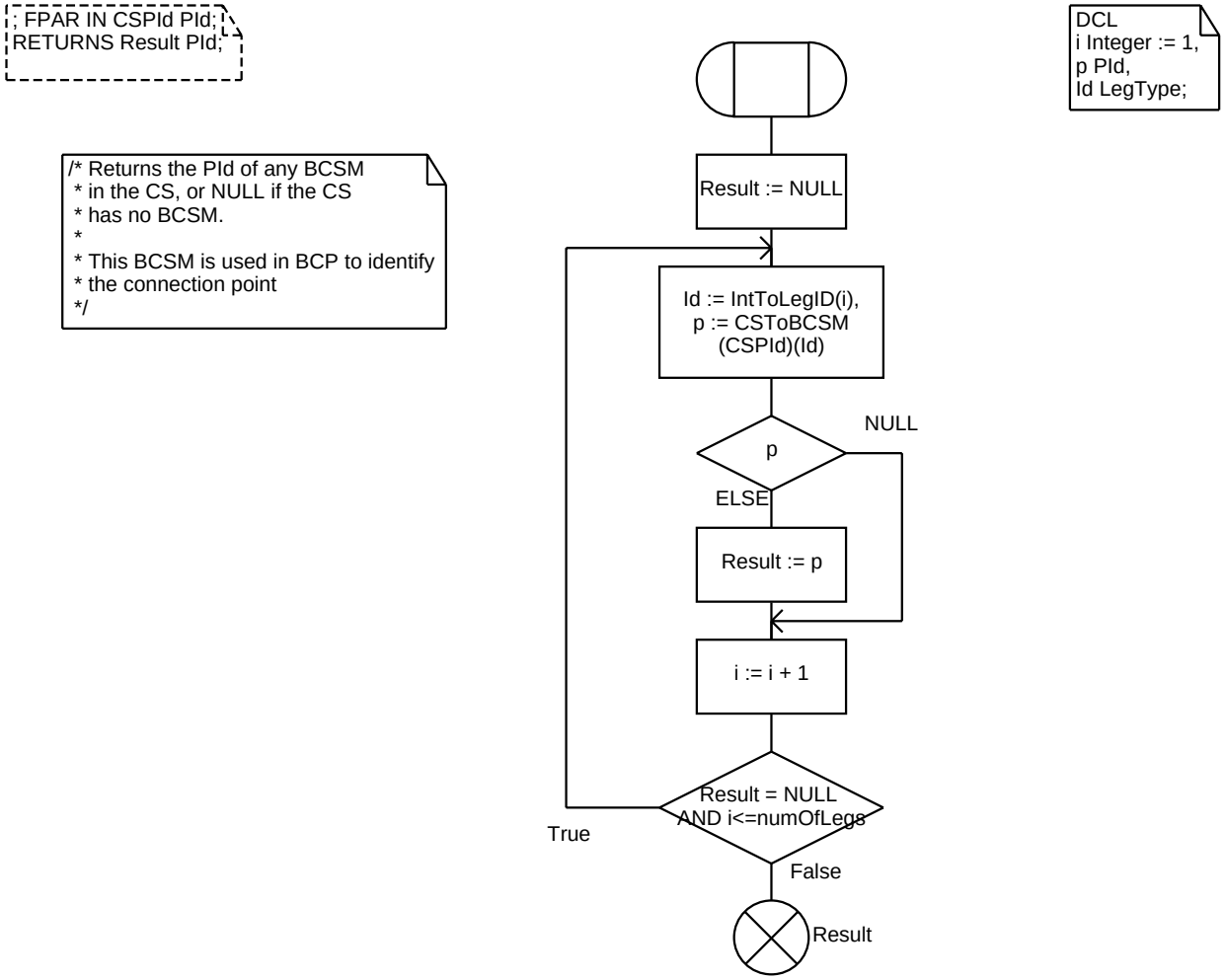
```
/*
This procedure should be called after
calling GetTriggeredTDPs.
This procedure will select
one service from ArmedDPList
and return the corresponding index in the TDP table,
or −1 if no service shall be triggered.
This selected service will be the
one that is really triggered.

In the ETSI model, we always select
the service in the list with the lowest
possible service key that is higher
than BCSM!highestServiceKey. This prevents
the same service being retriggered over and
over again.

In real implementations, this mechanism
should be based on more advanced
feature interaction mechanisms, like:
− using more advanced priority schemes
− taking into account which services are already
  active in the current call, and select a compatible one
*/
```



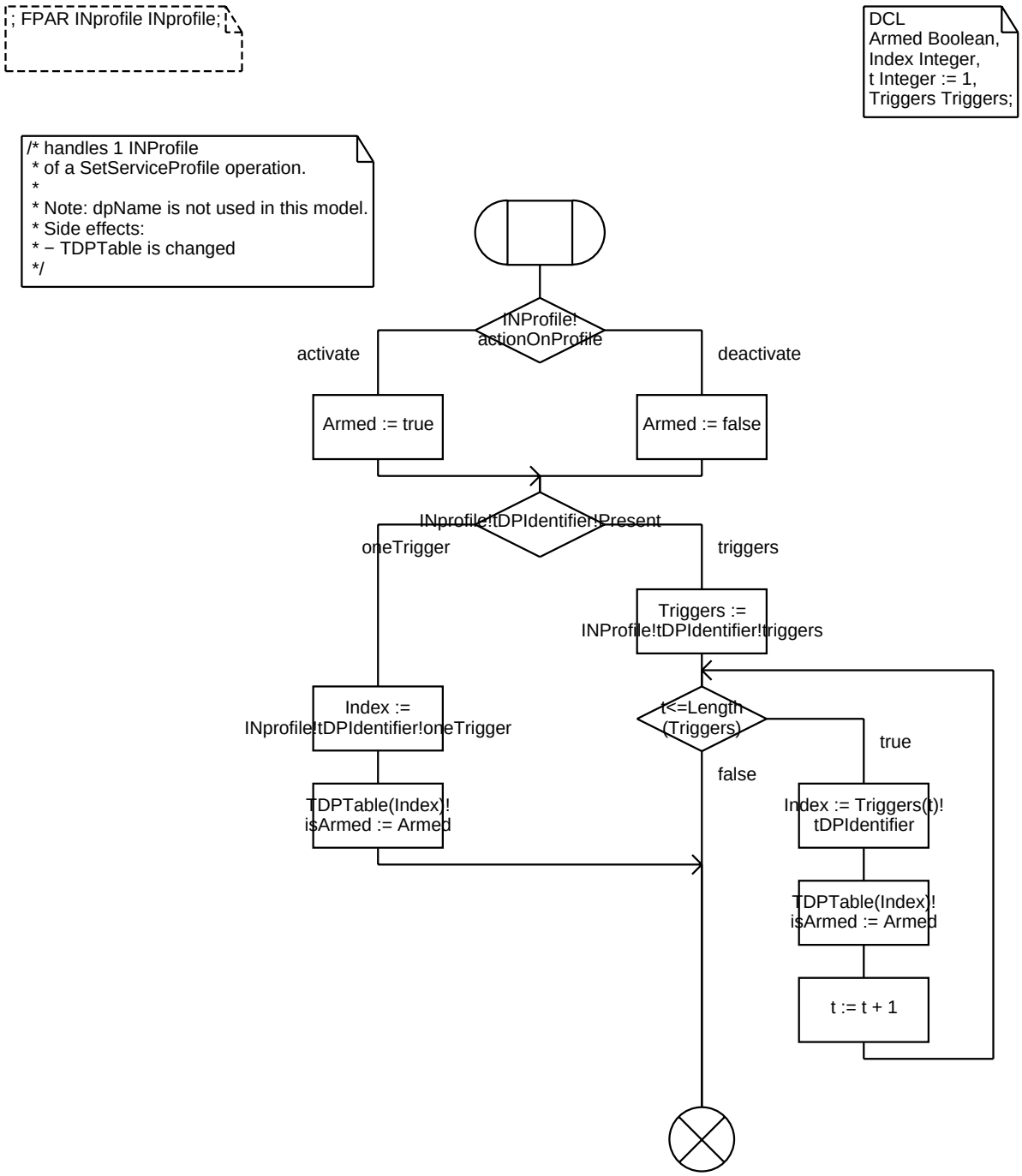
Procedure BCSMInfo 1(1)



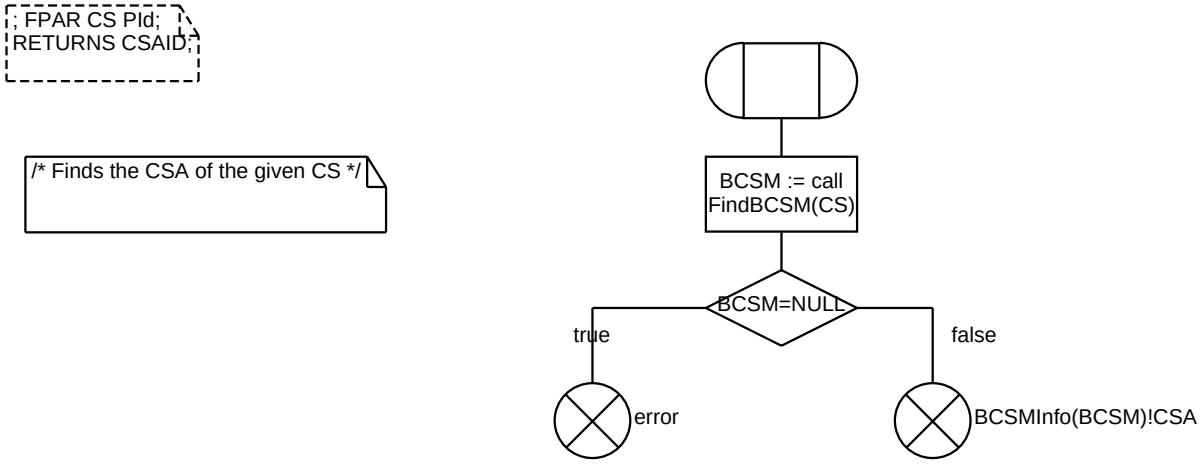
Procedure FindBCSM 1(1)



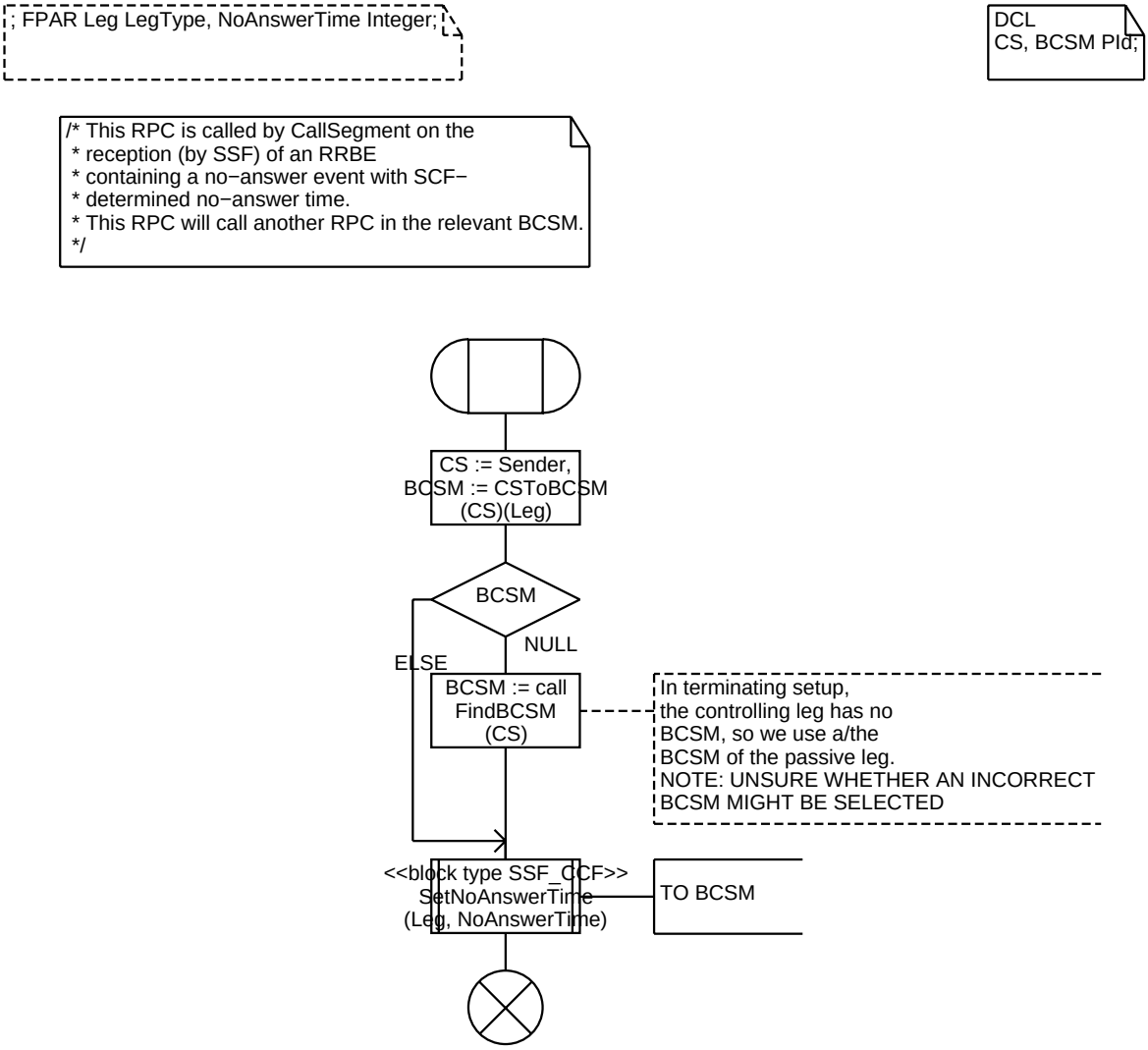
Procedure SetOneServiceProfile 1(1)



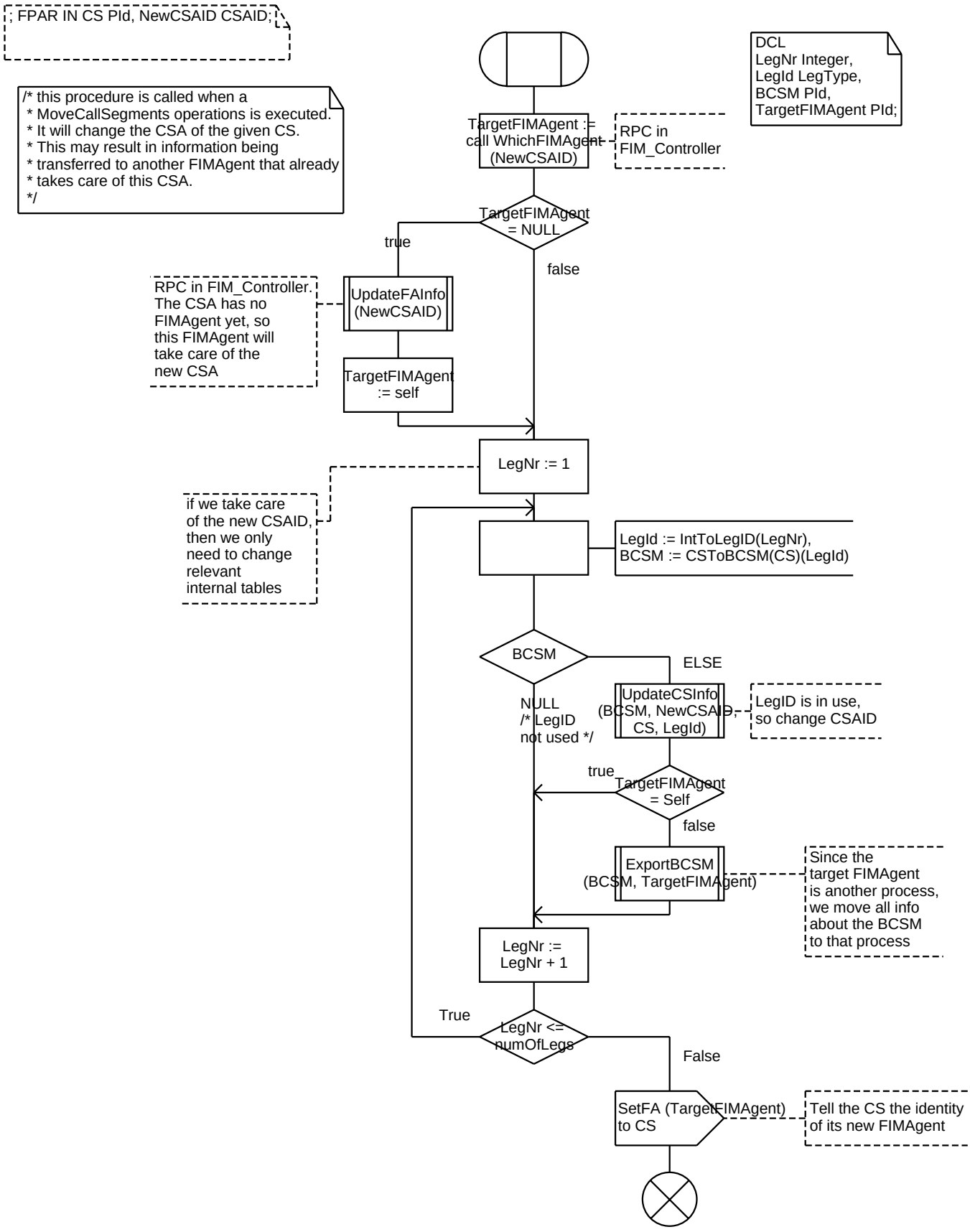
Procedure CSToCSA 1(1)



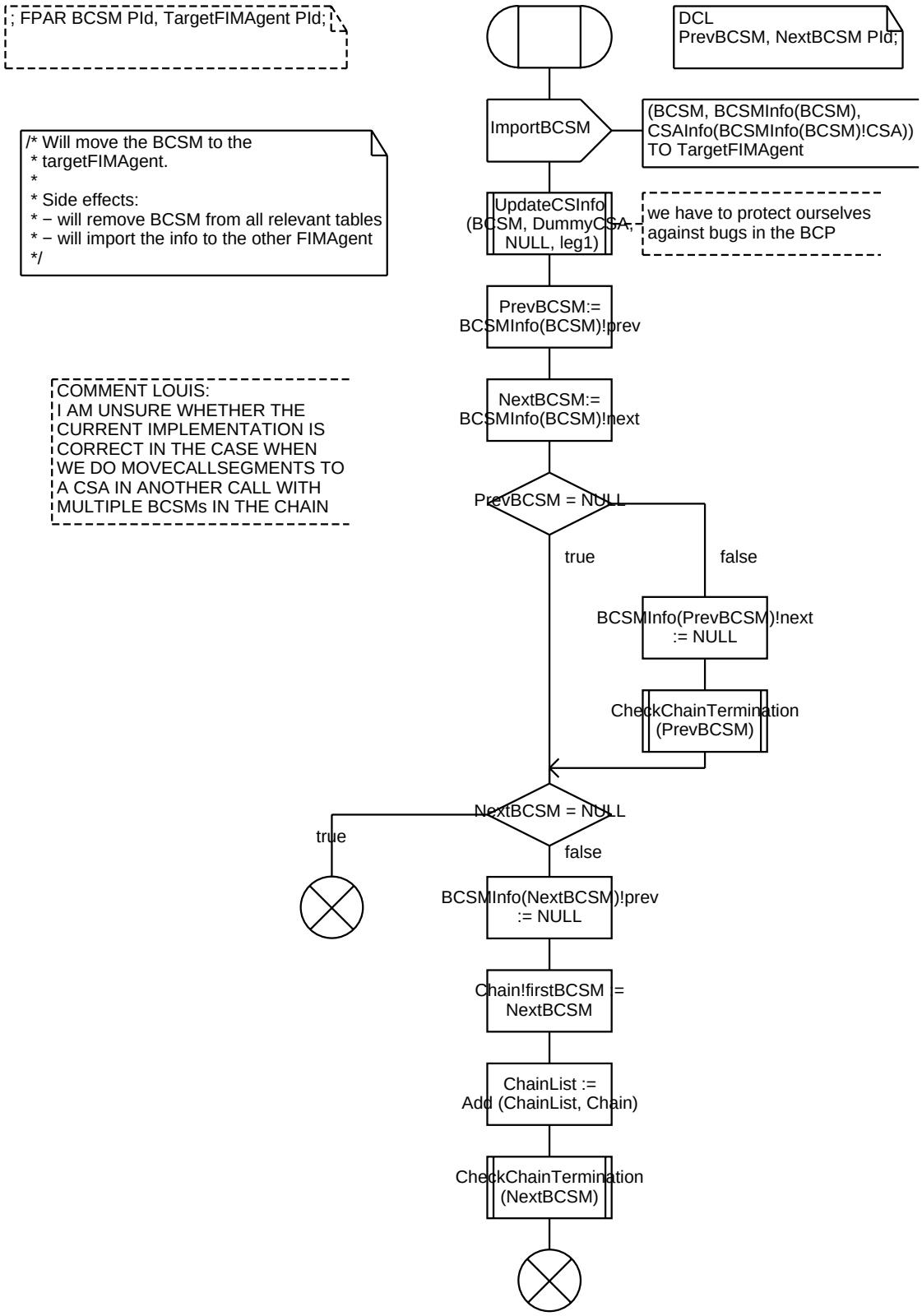
Procedure <<Process Type FIMAgent>> SetNoAnswerTime 1(1)



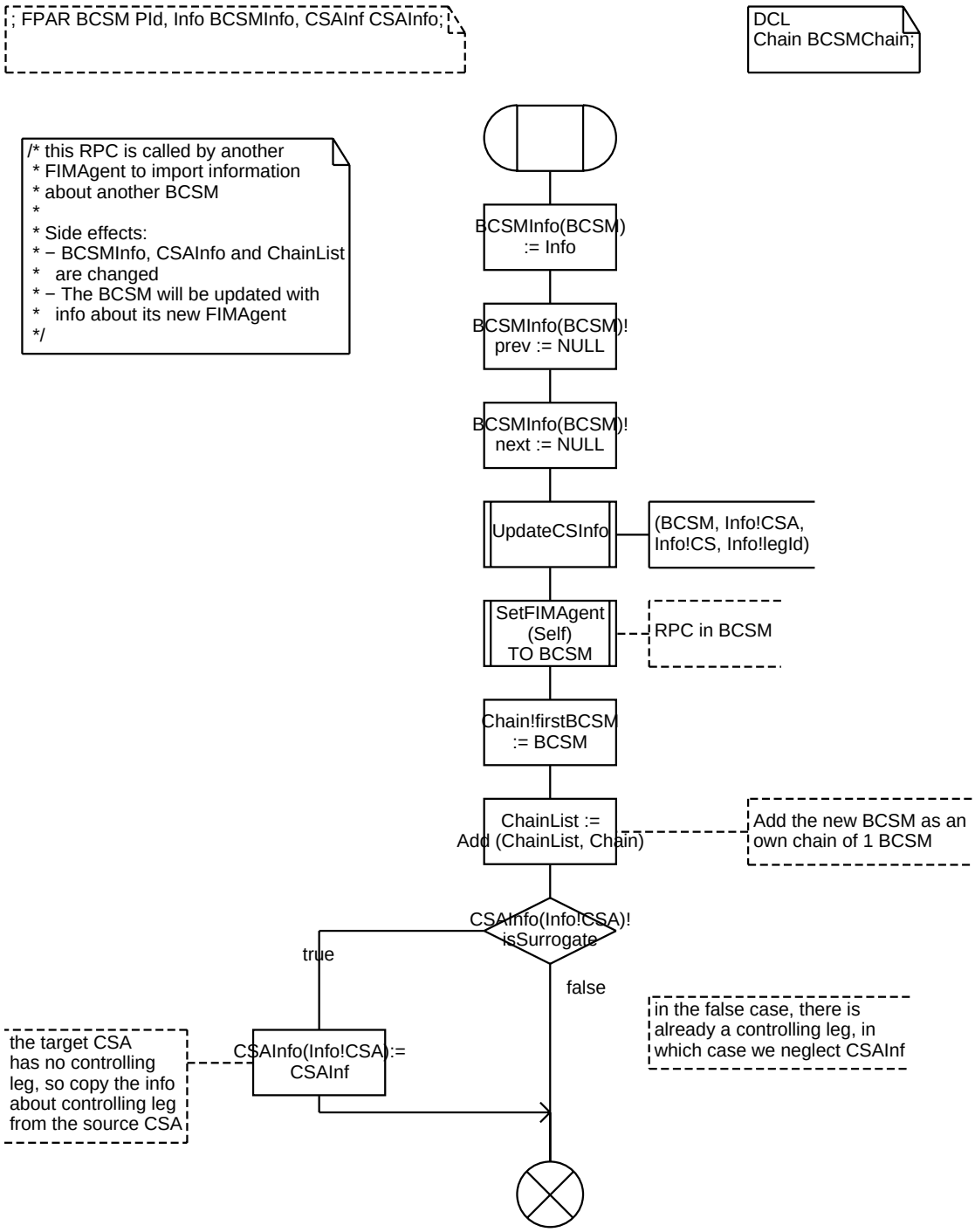
Procedure MoveCallSegment 1(1)



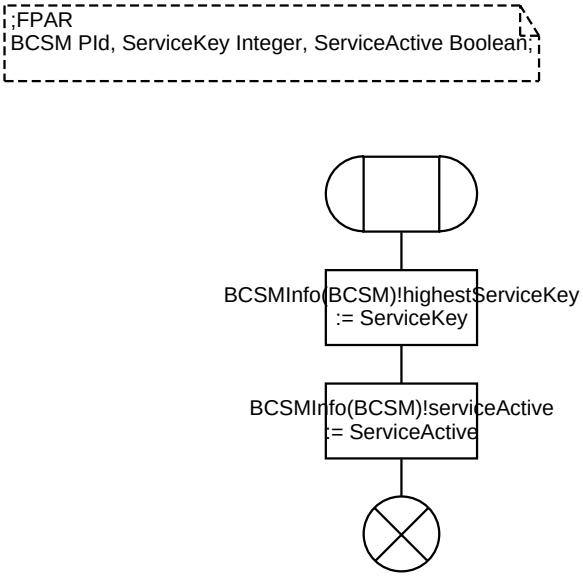
Procedure ExportBCSM 1(1)

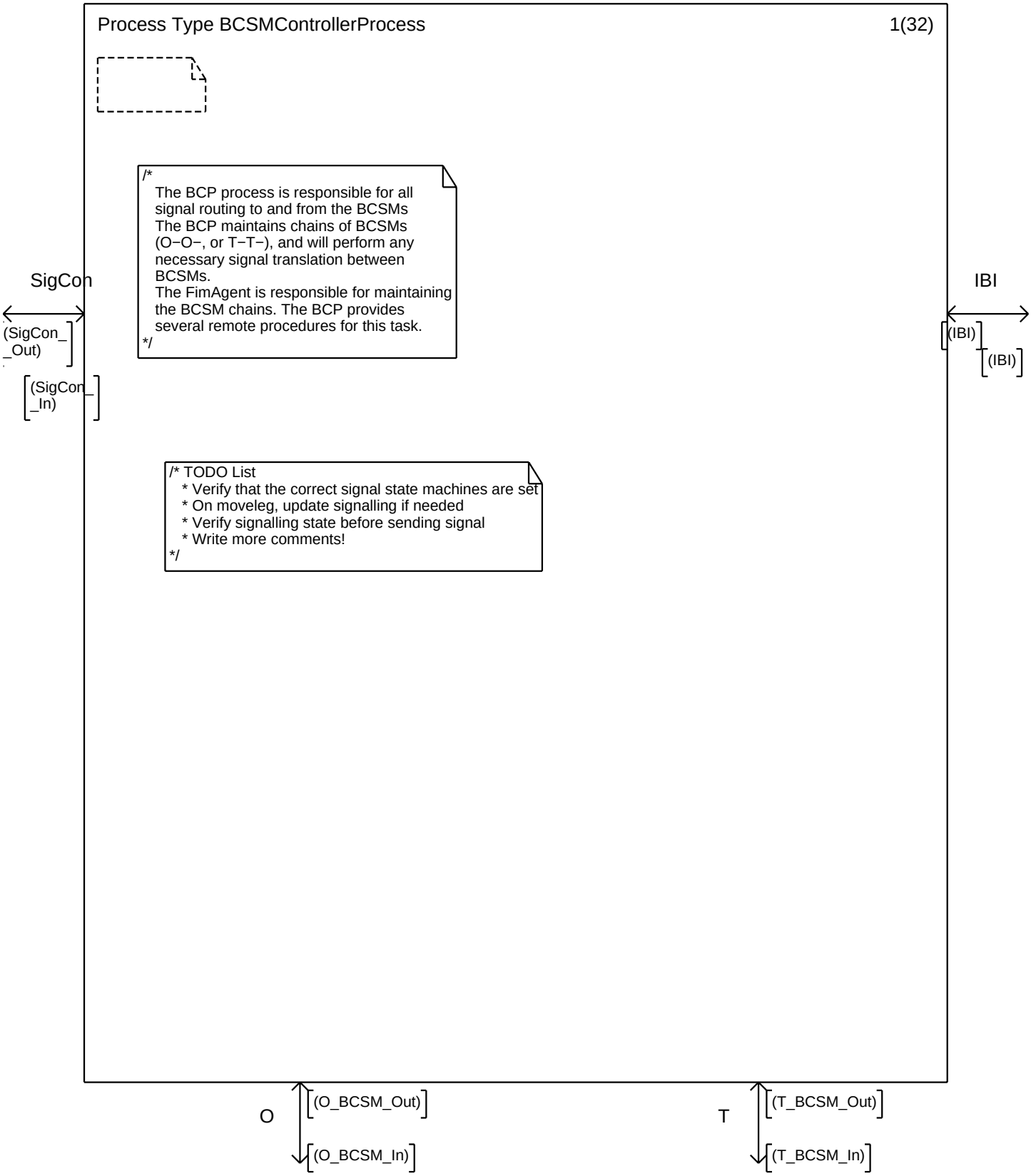


Procedure ImportBCSM 1(1)



Procedure UpdateServiceInfo 1(1)




Process Type BCSMControllerProcess
1(32)
/*
The BCP process is responsible for all signal routing to and from the BCSMs
The BCP maintains chains of BCSMs (O−O−, or T−T−), and will perform any necessary signal translation between BCSMs.
The FimAgent is responsible for maintaining the BCSM chains. The BCP provides several remote procedures for this task.
*/
SigCon
(SigCon_ _Out)
(SigCon_ _In)
IBI
(IBI)
(IBI)
/* TODO List
* Verify that the correct signal state machines are set
* On moveleg, update signalling if needed
* Verify signalling state before sending signal
* Write more comments!
*/
O
(O_BCSM_Out)
(O_BCSM_In)
T
(T_BCSM_Out)
(T_BCSM_In)

Process Type BCSMControllerProcess 2(32)

/* Types used for handling chain of BCSMs */

```
NEWTYPE ConnectType
/* What we are connected to */
  LITERALS
    CP, /* Anonter ConnectPoint */
    IBI, /* IBI Interface */
    SigCon, /* SigCon interface */
/* Nothing − surrogate leg, terminate signalling */
    Surrogate,
/* Nothing (on passive side), terminate signalling */
    Nothing;
ENDNEWTYPE;

NEWTYPE ControlInfo
/* Information on what the controlling leg is connected to */
  STRUCT
/* Used when we have no passive legs */
    bcsm PId;
/* The type of connection */
    connectedTo ConnectType;
/* The controlling leg signalling state */
   controlState SignallingState;
/* Index into the CPList array */
    nextcp Integer; /* Only used if connectedTo = CP */
/* Only used if connectedTo = SigCon */
    sigConID CallRef;
  DEFAULT (. null, SigCon, idle, 0, 0 .);
ENDNEWTYPE;

NEWTYPE PassiveInfo
/* Information on what the passive leg is connected to */
  STRUCT
    connectedTo ConnectType;
    bcsm PId;
    nextcp Integer; /* Only used if connectedTo = CP */
    callRef CallRef; /* Only used if connectedTo = IBI */
ENDNEWTYPE;

NEWTYPE ConnectPoint
/* Describes a point where a controlling and
  a passive leg is connected */
  STRUCT
   controlling ControlInfo;
   transfer Boolean; /* Is it a real transfer */
   passives PassiveInfoList;
  DEFAULT (. (. null, SigCon, idle, 0, 0 .), True, emptystring .);
ENDNEWTYPE;
```

Process Type BCSMControllerProcess 3(32)

```
/* PassiveInfoList is used to maintain a list of passive legs */
NEWTYPE PassiveInfoList
  String (PassiveInfo, emptystring);
 OPERATORS
  Add: PassiveInfoList, PassiveInfo −> PassiveInfoList;
  /* adds a new passiveInfo to the list */
  Del: PassiveInfoList, Integer −> PassiveInfoList;
  /* removes 1 passiveInfo (indicated by the index) from the list */
  FindBCSM: PassiveInfoList, PId −> Integer;
  /* Returns the index of the PassiveInfo containing the supplied BCSM */
  FindCP: PassiveInfoList, Integer −> Integer;
  /* Returns the index of the PassiveInfo pointing to the supplied CP */

 OPERATOR Add;
  FPAR list PassiveInfoList, pInfo PassiveInfo;
  RETURNS PassiveInfoList;
 START;
    RETURN list // Mkstring (pInfo);
 ENDOPERATOR;

 OPERATOR Del;
  FPAR list PassiveInfoList, index Integer;
  RETURNS PassiveInfoList;
 START;
    RETURN Substring(list, 1, index−1) //
      Substring (list, index+1, Length(list)−index);
 ENDOPERATOR;

OPERATOR FindBCSM;
    FPAR list PassiveInfoList, bcsm PId;
    RETURNS Integer;
    DCL pos Integer := −1;
    DCL i Integer;
    DCL n Integer;
START ;
TASK n := Length(list);
TASK i := 1;
repeat:
DECISION i <= n;
 (True):
  DECISION bcsm = list(i)!bcsm;
   (true):
    TASK pos := i;
    RETURN pos;
   ELSE:
    TASK i := i + 1;
  ENDDECISION;
 ELSE:
  RETURN −1;
ENDDECISION;
JOIN repeat;
ENDOPERATOR FindBCSM;

OPERATOR FindCP;
    FPAR list PassiveInfoList, cp Integer;
    RETURNS Integer;
    DCL pos Integer := −1;
    DCL i Integer;
    DCL n Integer;
START ;
TASK n := Length(list);
TASK i := 1;
repeat:
DECISION i <= n;
 (True):
  DECISION cp = list(i)!nextcp;
   (true):
    TASK pos := i;
    RETURN pos;
   ELSE:
    TASK i := i + 1;
  ENDDECISION;
 ELSE:
  RETURN −1;
ENDDECISION;
JOIN repeat;ENDOPERATOR FindCP;

ENDNEWTYPE;
```

Process Type BCSMControllerProcess 4(32)

```
NEWTYPE BCSMInfo
  STRUCT
     cpNum Integer; /* Index to CPList */
     bcsmType BCSMType; /* O or T */
/* current signalling status for this bcsm */
     sigState SignallingState;
  DEFAULT (. 0, originating, idle .)
ENDNEWTYPE;

NEWTYPE BCSMArray
/* This is where we store our BCSMs */
  Array(PId, BCSMInfo);
ENDNEWTYPE;

NEWTYPE ConnectPointArray
/* This is where we store our ConnectPoints */
  Array(Integer, ConnectPoint);
ENDNEWTYPE;

NEWTYPE CallRefArray
/* Maps from a callref to a Receiver */
  Array(CallRef, ReceiverInfo);
ENDNEWTYPE;

NEWTYPE SigConArray
/* Maps from SigConID to a ConnectPoint */
  Array(CallRef, Integer);
ENDNEWTYPE;

NEWTYPE SignalType /* What type of signal */
  LITERALS
     SigCon_Sig, /* Signal sent on SigCon interface */
     IBI_Sig; /* Signal sent on IBI interface */
ENDNEWTYPE;

NEWTYPE PIdSet
/* Set of PIds, used for storing active BCSMs */
  PowerSet(PId);
ENDNEWTYPE;
```

```
NEWTYPE ReceiverInfo
/* Describes a signal receiver */
   STRUCT
      connectedTo ConnectType;
      bcsm PId;
      sigconID CallRef;
      callRef CallRef;
ENDNEWTYPE;

/* ReceiverInfoList is used to maintain a list of signal receivers */
NEWTYPE ReceiverInfoList
  String (ReceiverInfo, emptystring);
 OPERATORS
  Add: ReceiverInfoList, ReceiverInfo −> ReceiverInfoList;
  /* adds a new ReceiverInfo to the list */

 OPERATOR Add;
  FPAR list ReceiverInfoList, rInfo ReceiverInfo;
  RETURNS ReceiverInfoList;
 START;
    RETURN list // Mkstring (rInfo);
 ENDOPERATOR;
ENDNEWTYPE;
```

```
/*    DECLARATION OF IMPORTED
   REMOTE PROCEDURES */

/* CreateFIMAgent is exported by FIM_Controller, and called by BCP and CS.
 * It requests the creation of a new FIMAgent + a chain of BCSMs.
 * Returns the PId of the created FIMAgent.
 */
IMPORTED PROCEDURE CreateFIMAgent;
 FPAR BCSMType, BCSMPars, CSAID, PId, LegType;
 RETURNS PId;
```

Process Type BCSMControllerProcess 5(32)

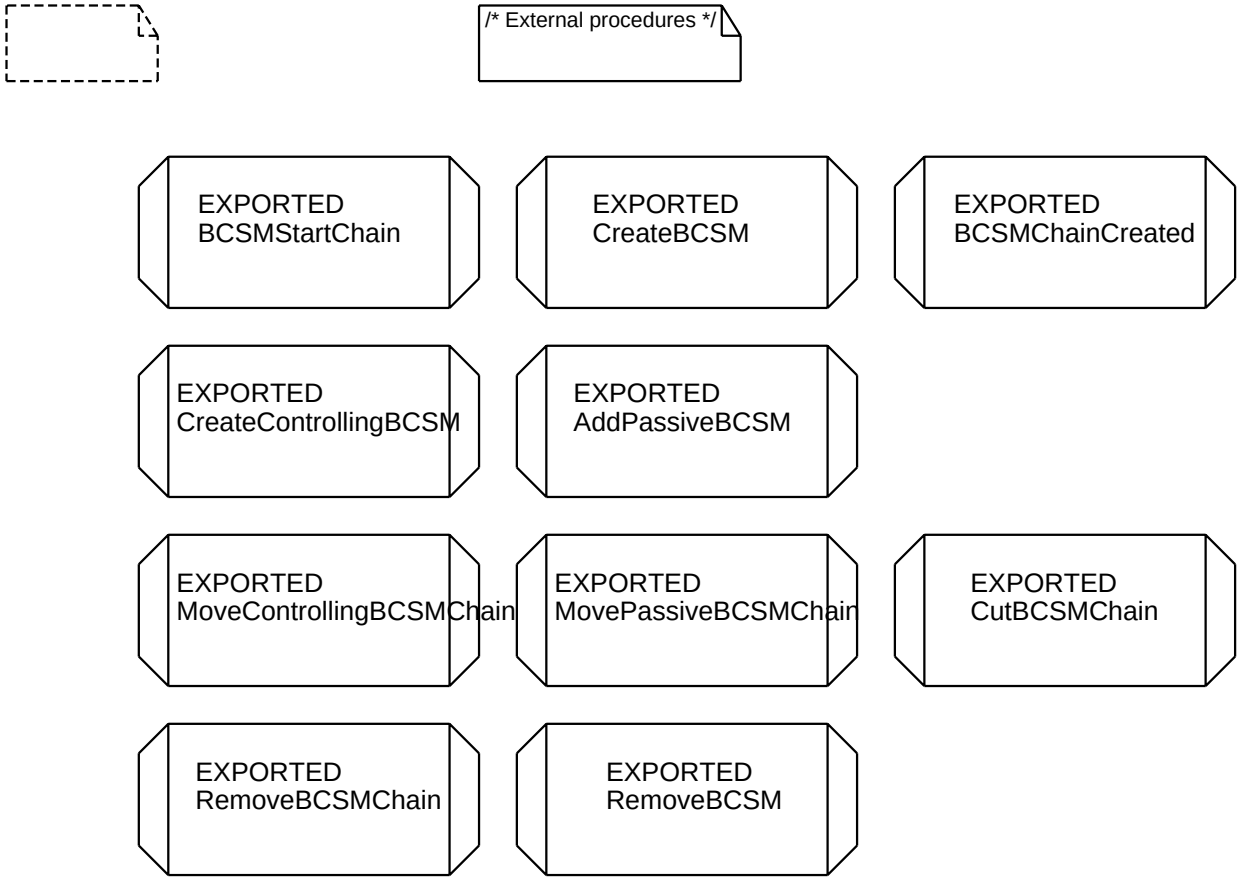
```
/* Variables used in the process */
```

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

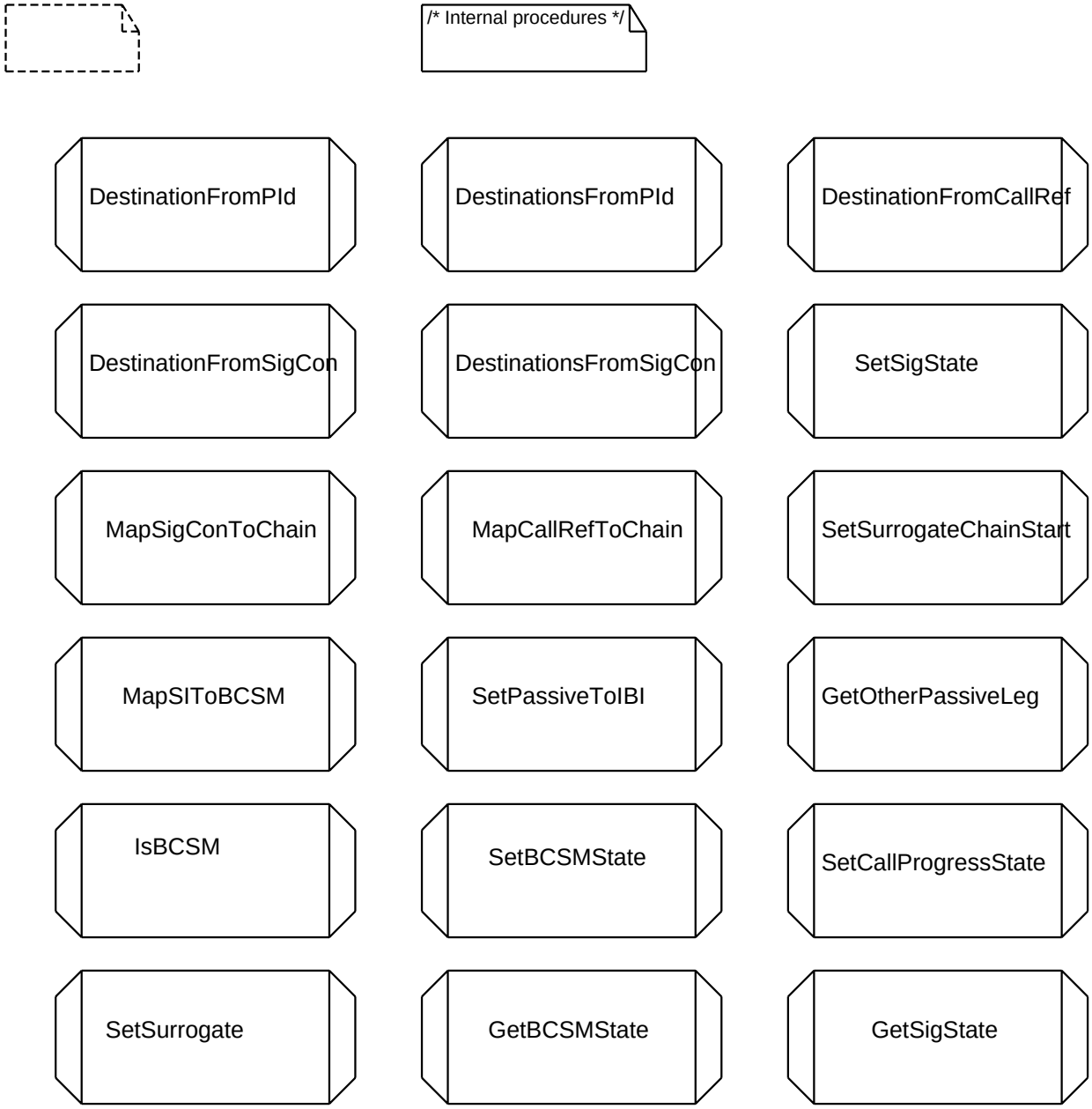
```
/* Transient variables */
DCL
  Receiver ReceiverInfo,
  Receivers ReceiverInfoList,
  SenderConnectedTo ConnectType,
  FirstBCSM PId,
  LastBCSM PId,
  otherLeg PId,
  bcsmType, bcsmType2 BCSMType,
  cPoint ConnectPoint,
  i Integer,
  N Integer,
  bcsmPars BCSMPars,
  scfInitiated Boolean,
  dummyPId PId;
```

```
/* Static variables related to the chain */
DCL
/* The BCSMs */
  BCSMs BCSMArray,
/* The connect points */
  CPList ConnectPointArray,
/* Map between call refs and receivers */
  CallRefs CallRefArray,
/* Map between sig con ids and connect points */
  SigCons SigConArray,
/* The set of active bcsms */
  activeBCSMs PIdSet,
/* Utility variables */
  HiCallRef CallRef,
  HiSigCon CallRef,
  HiConnectPoint Integer;
```

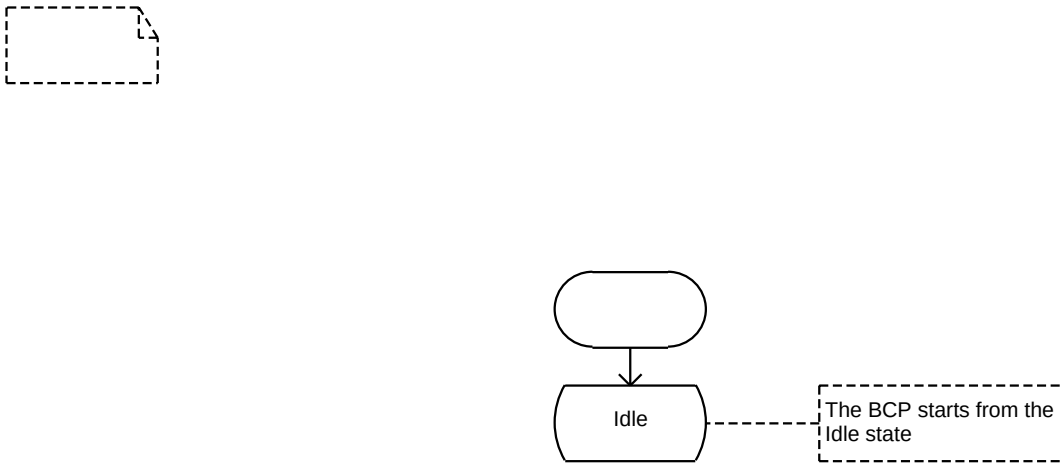
Process Type BCSMControllerProcess 6(32)



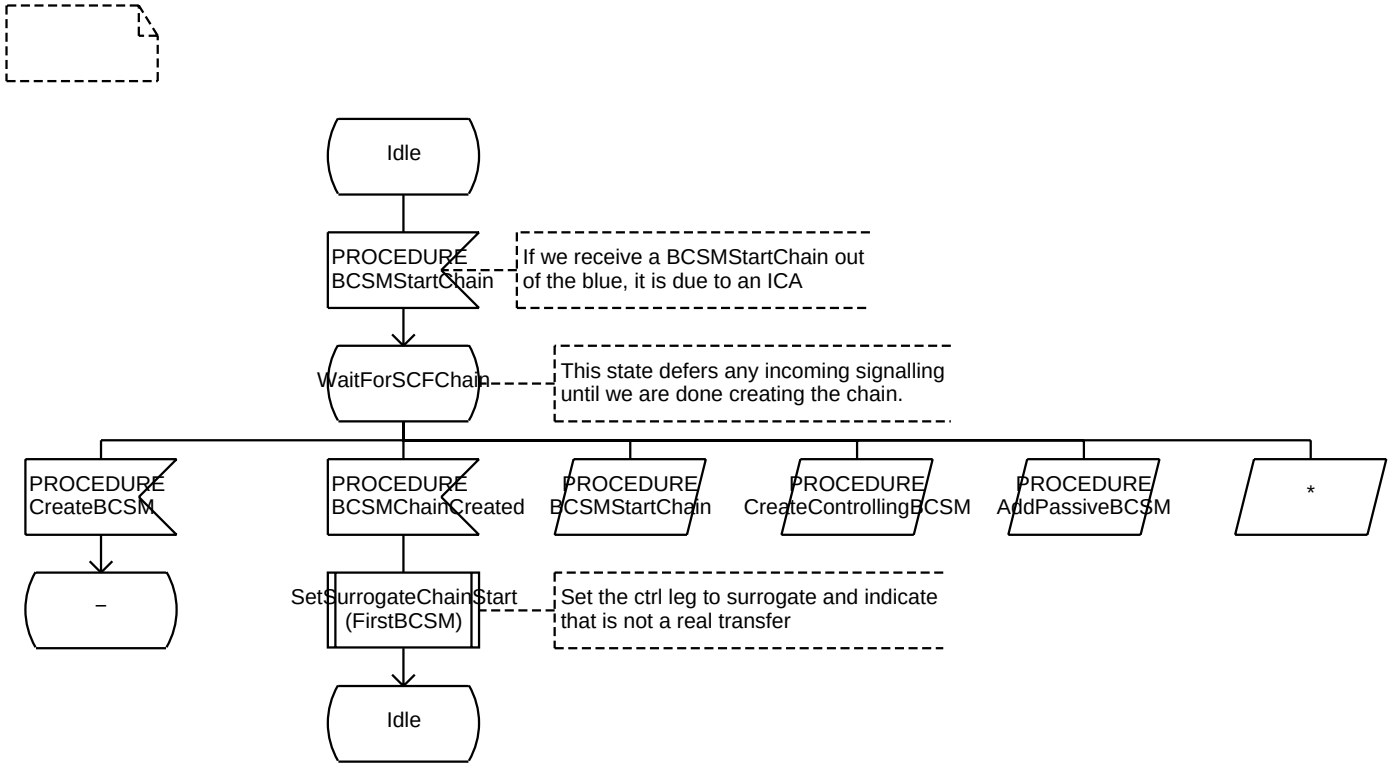
Process Type BCSMControllerProcess 7(32)

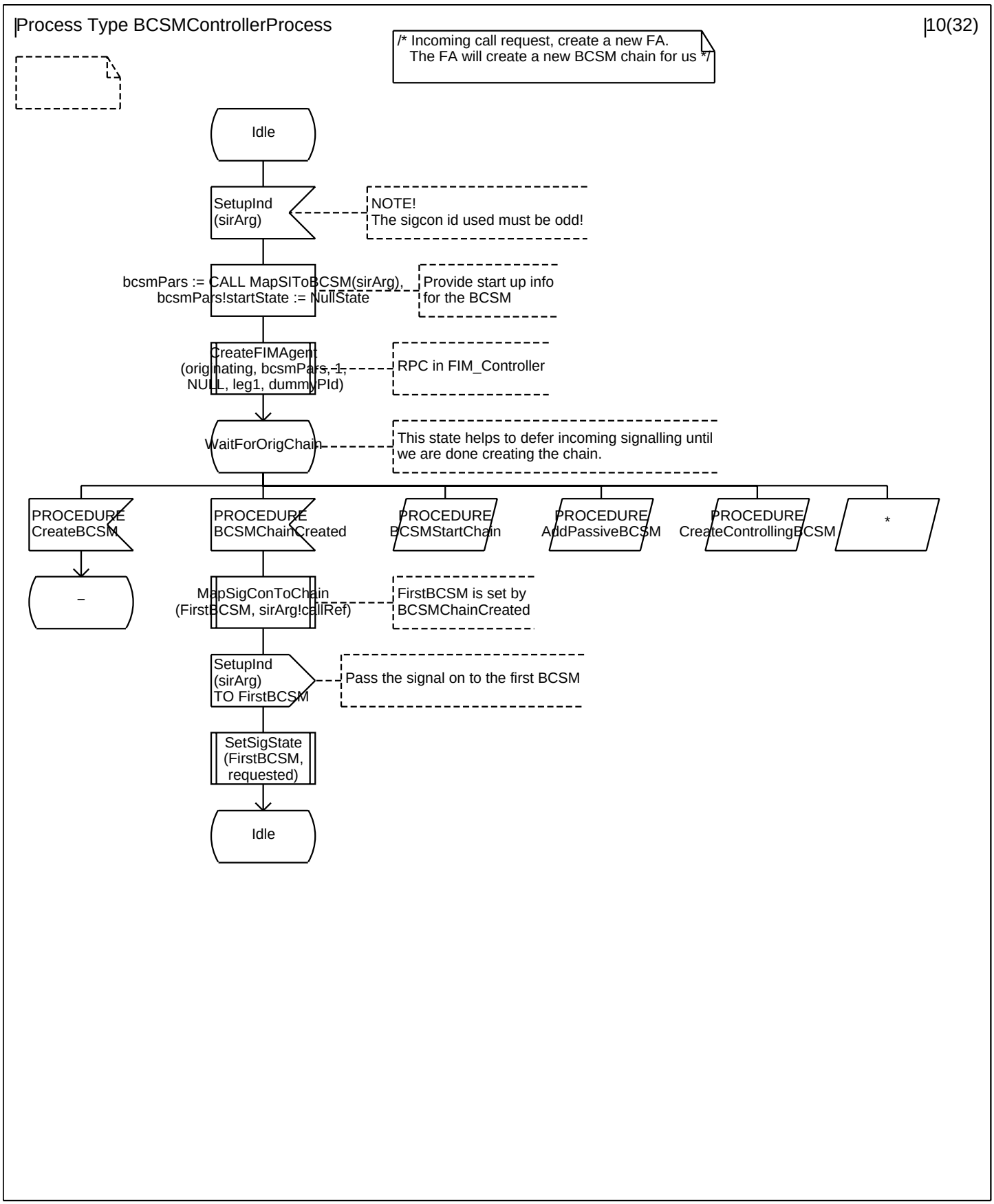


Process Type BCSMControllerProcess 8(32)

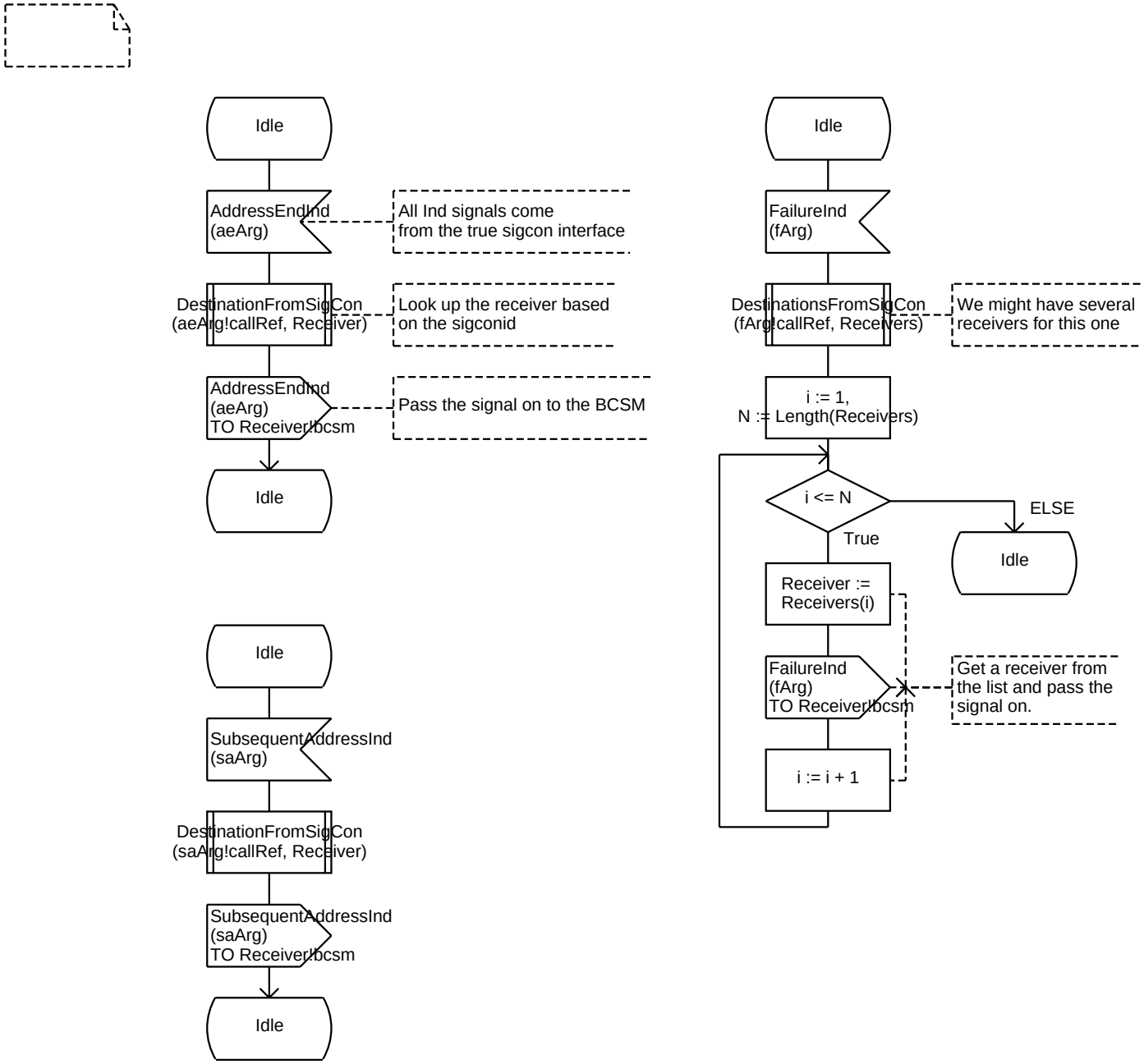


Process Type BCSMControllerProcess 9(32)

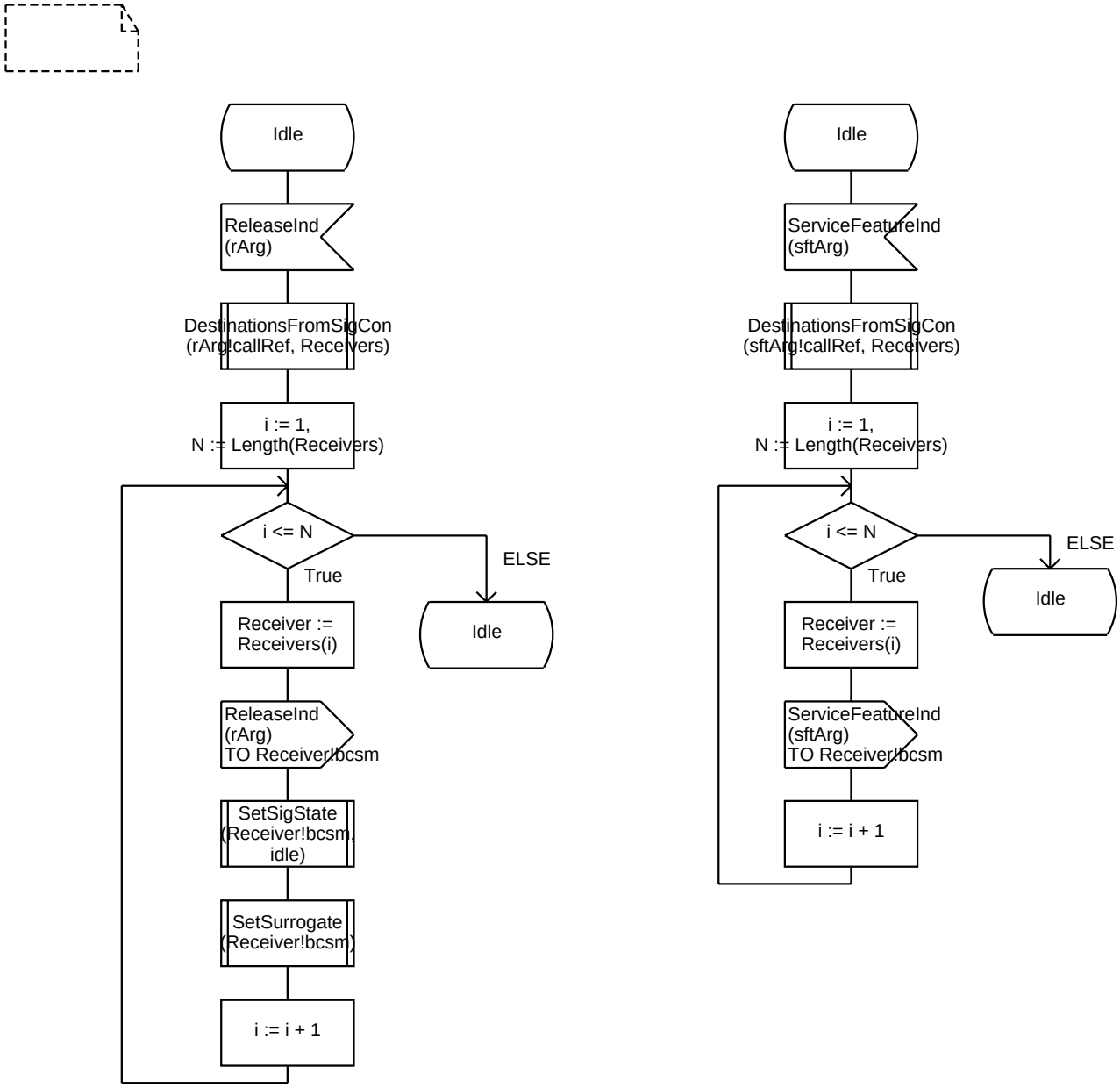



Process Type BCSMControllerProcess
10(32)
/* Incoming call request, create a new FA.
The FA will create a new BCSM chain for us */
Idle
SetupInd
(sirArg)
NOTE!
The sigcon id used must be odd!
bcsmPars := CALL MapSIToBCSM(sirArg),
bcsmPars!startState := NullState
Provide start up info
for the BCSM
CreateFIMAgent
(originating, bcsmPars, -1,
NULL, leg1, dummyPId)
RPC in FIM_Controller
WaitForOrigChain
This state helps to defer incoming signalling until
we are done creating the chain.
PROCEDURE
CreateBCSM
PROCEDURE
BCSMChainCreated
PROCEDURE
BCSMStartChain
PROCEDURE
AddPassiveBCSM
PROCEDURE
CreateControllingBCSM
*
–
MapSigConToChain
(FirstBCSM, sirArg!callRef)
FirstBCSM is set by
BCSMChainCreated
SetupInd
(sirArg)
TO FirstBCSM
Pass the signal on to the first BCSM
SetSigState
(FirstBCSM,
requested)
Idle

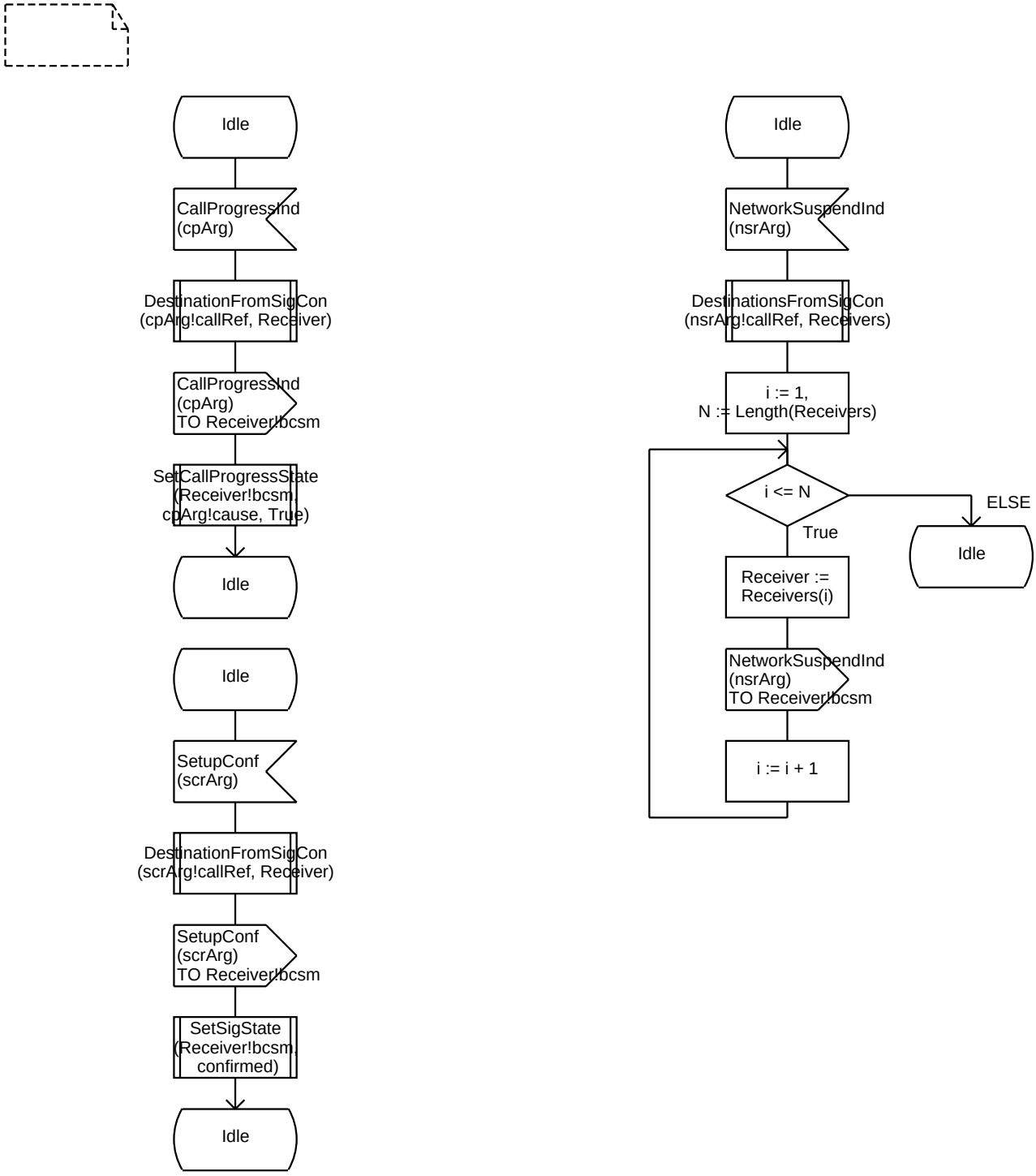
Process Type BCSMControllerProcess 11(32)



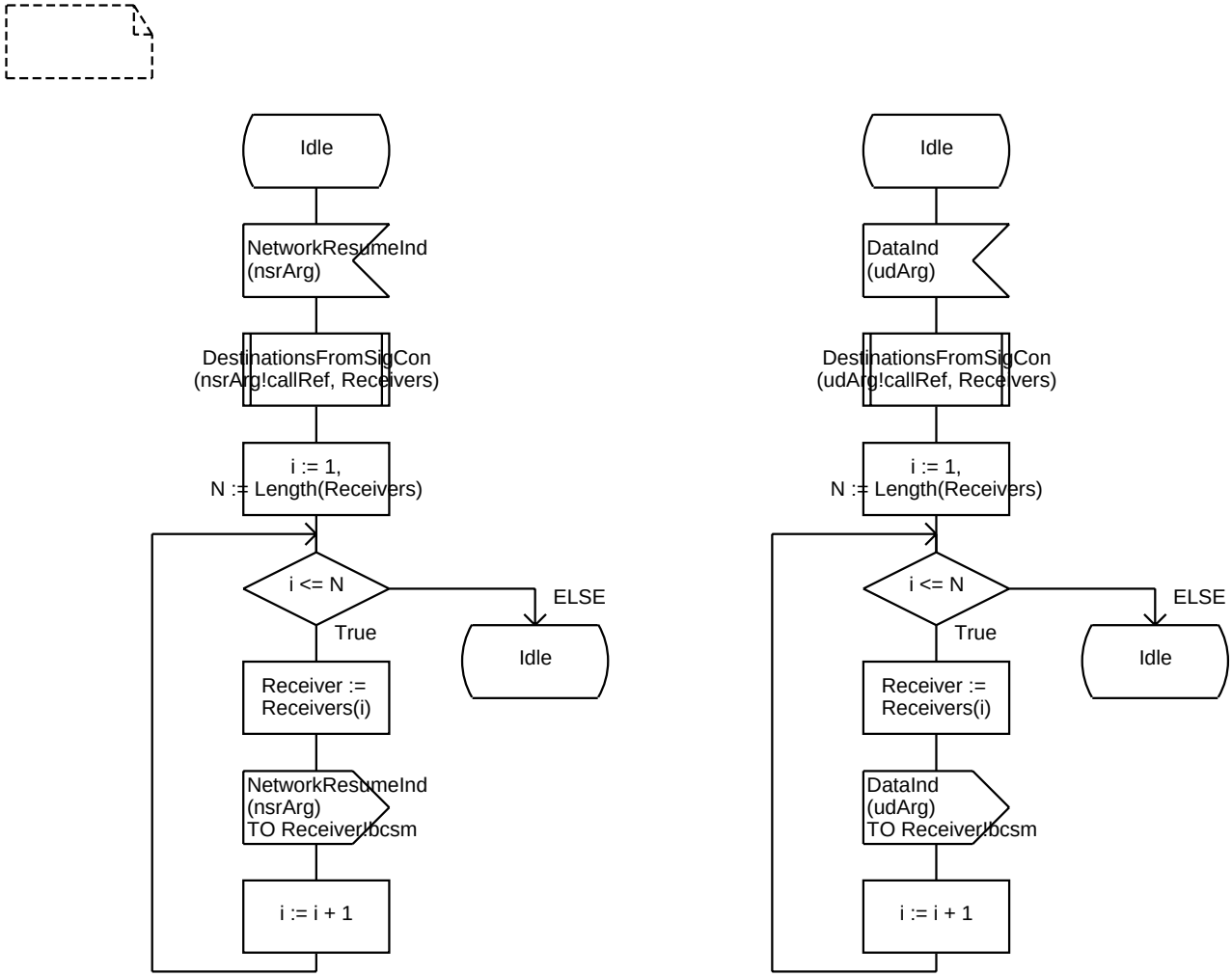
Process Type BCSMControllerProcess 12(32)



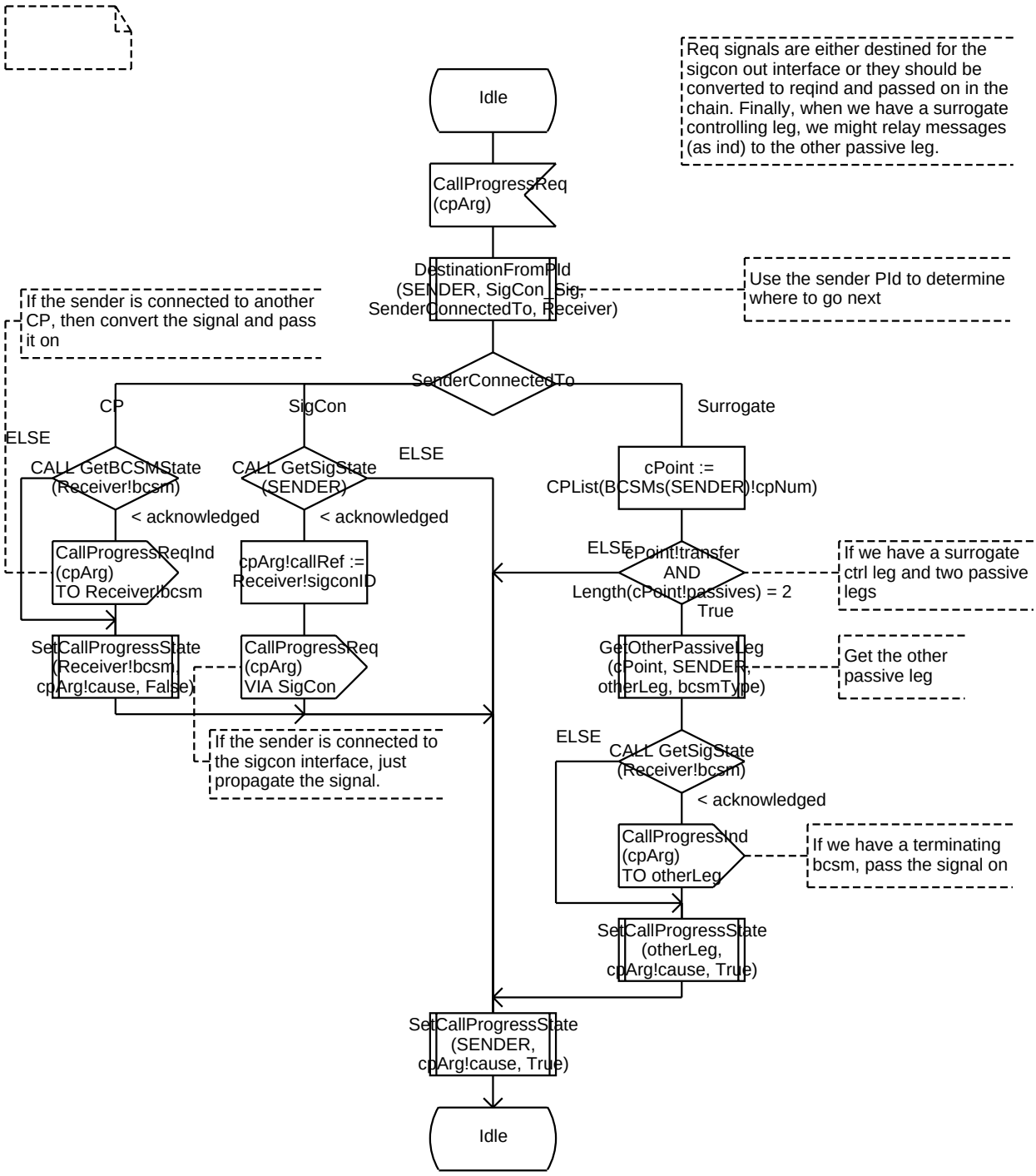
Process Type BCSMControllerProcess 13(32)



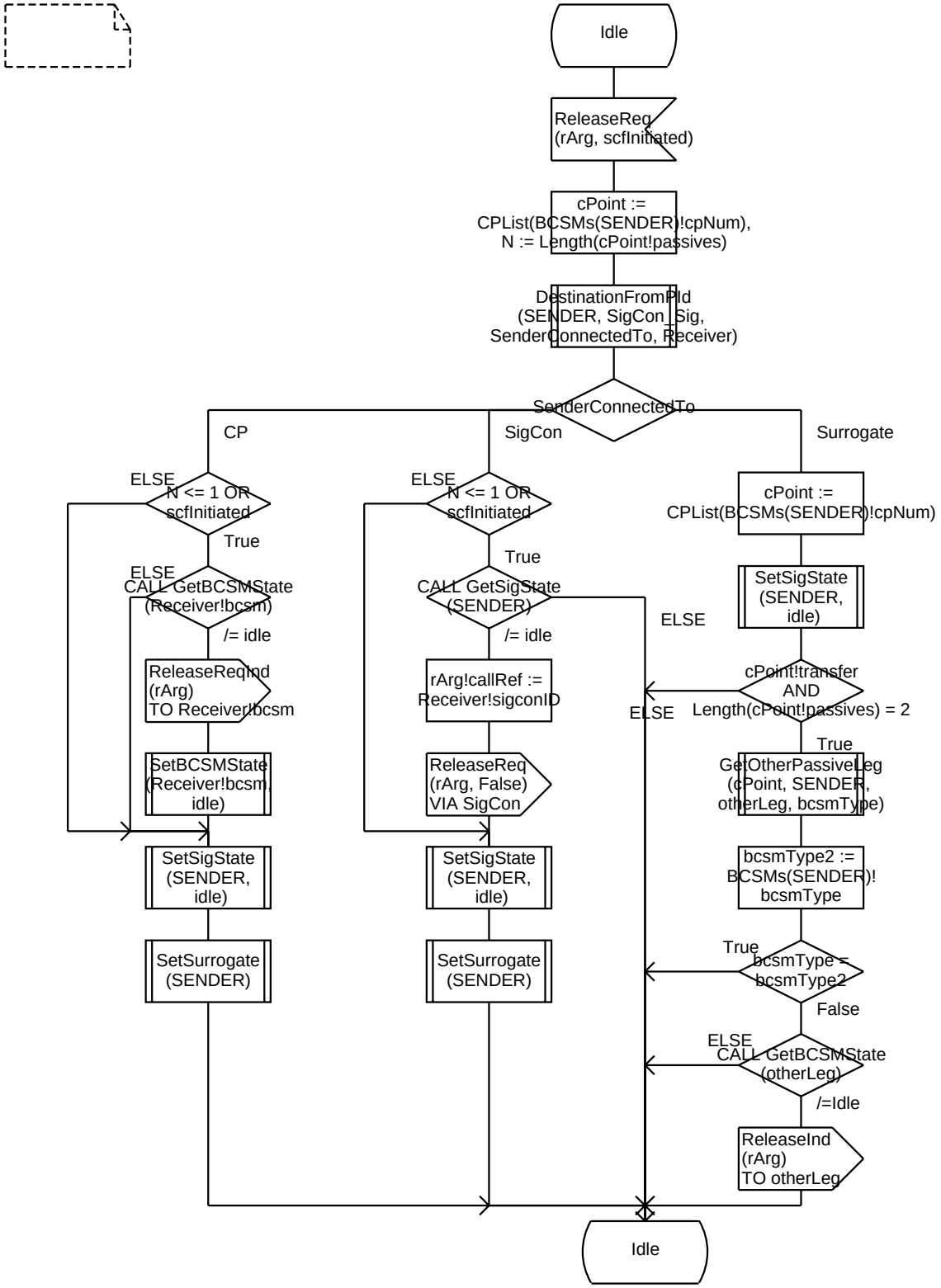
Process Type BCSMControllerProcess 14(32)



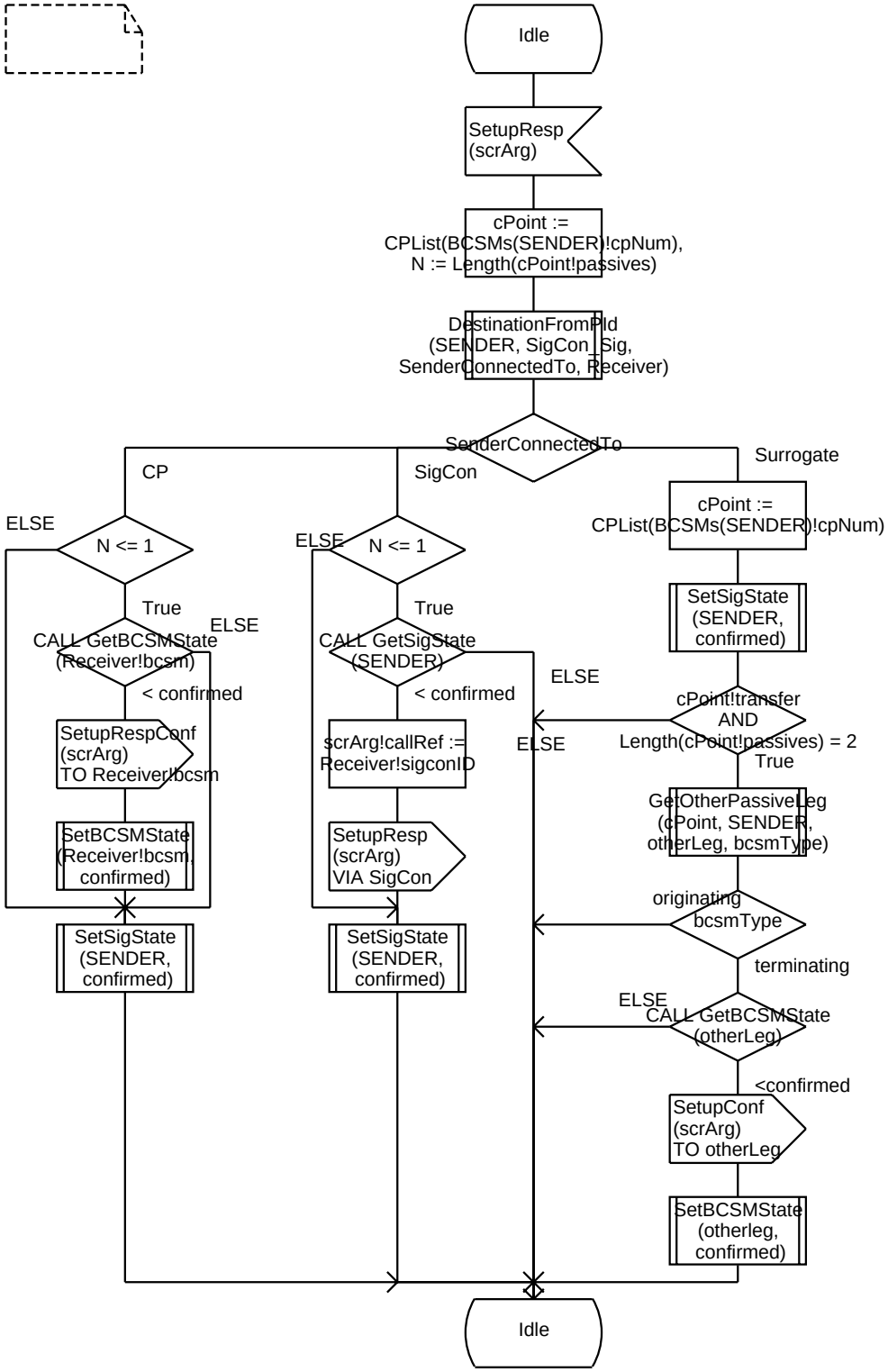
Process Type BCSMControllerProcess 15(32)



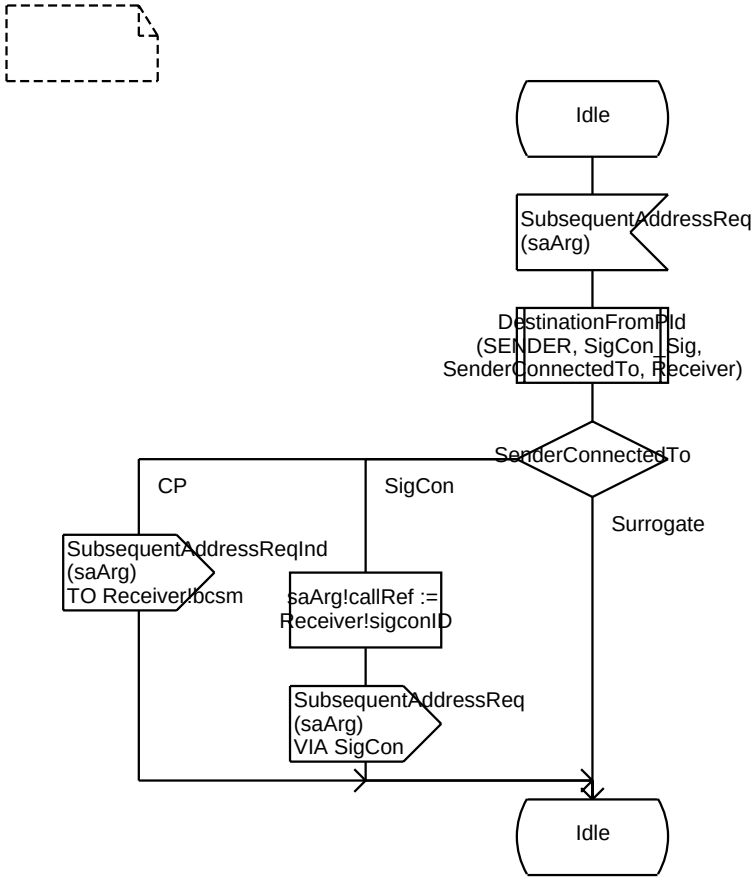
Process Type BCSMControllerProcess 16(32)



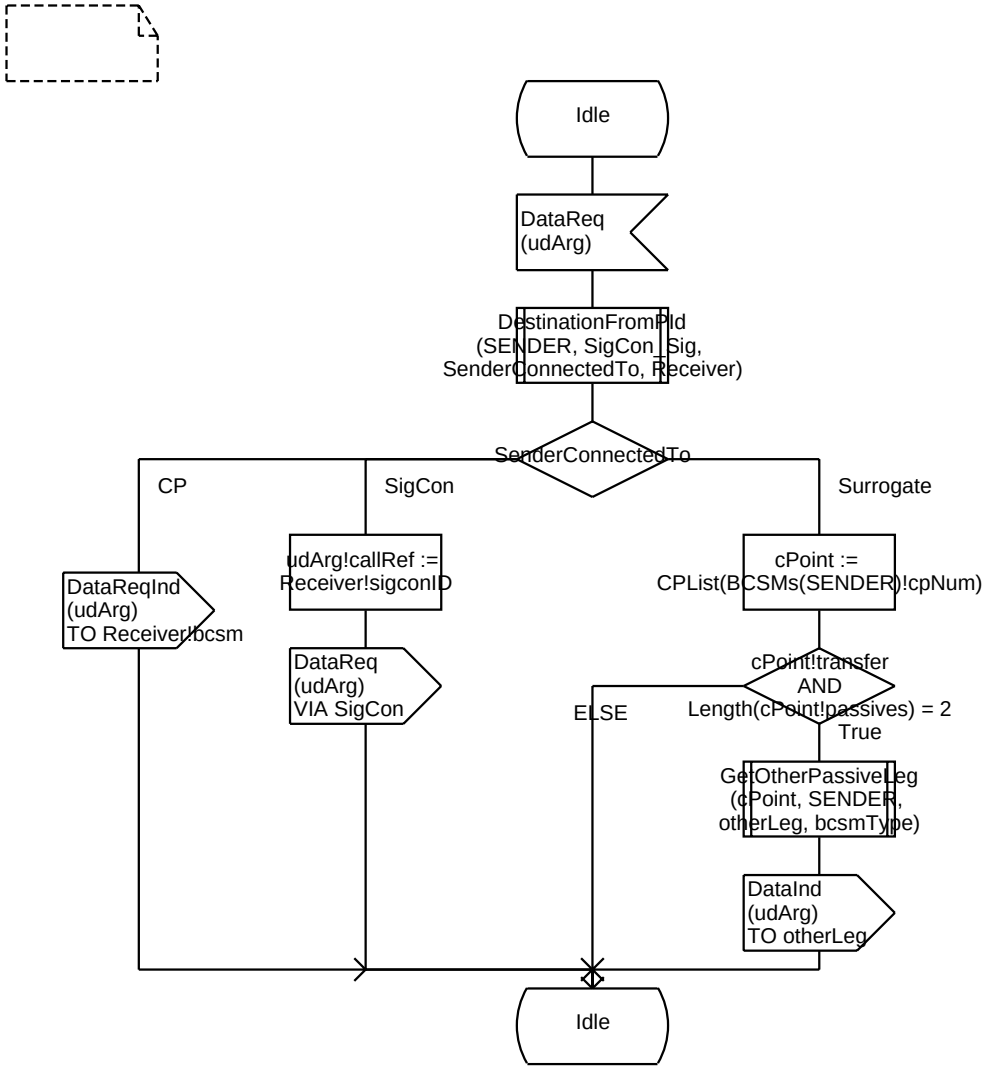
Process Type BCSMControllerProcess 17(32)



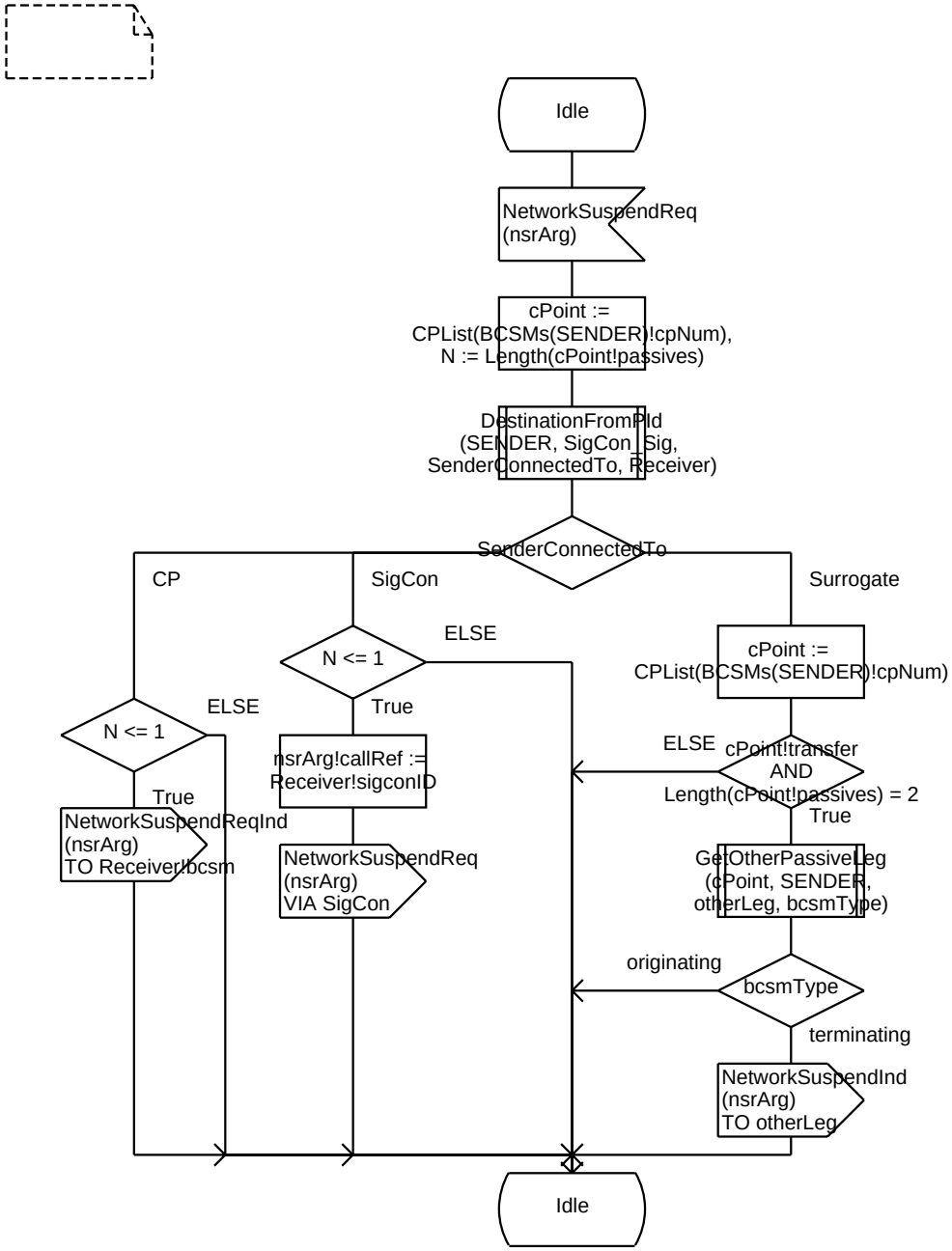
Process Type BCSMControllerProcess 18(32)



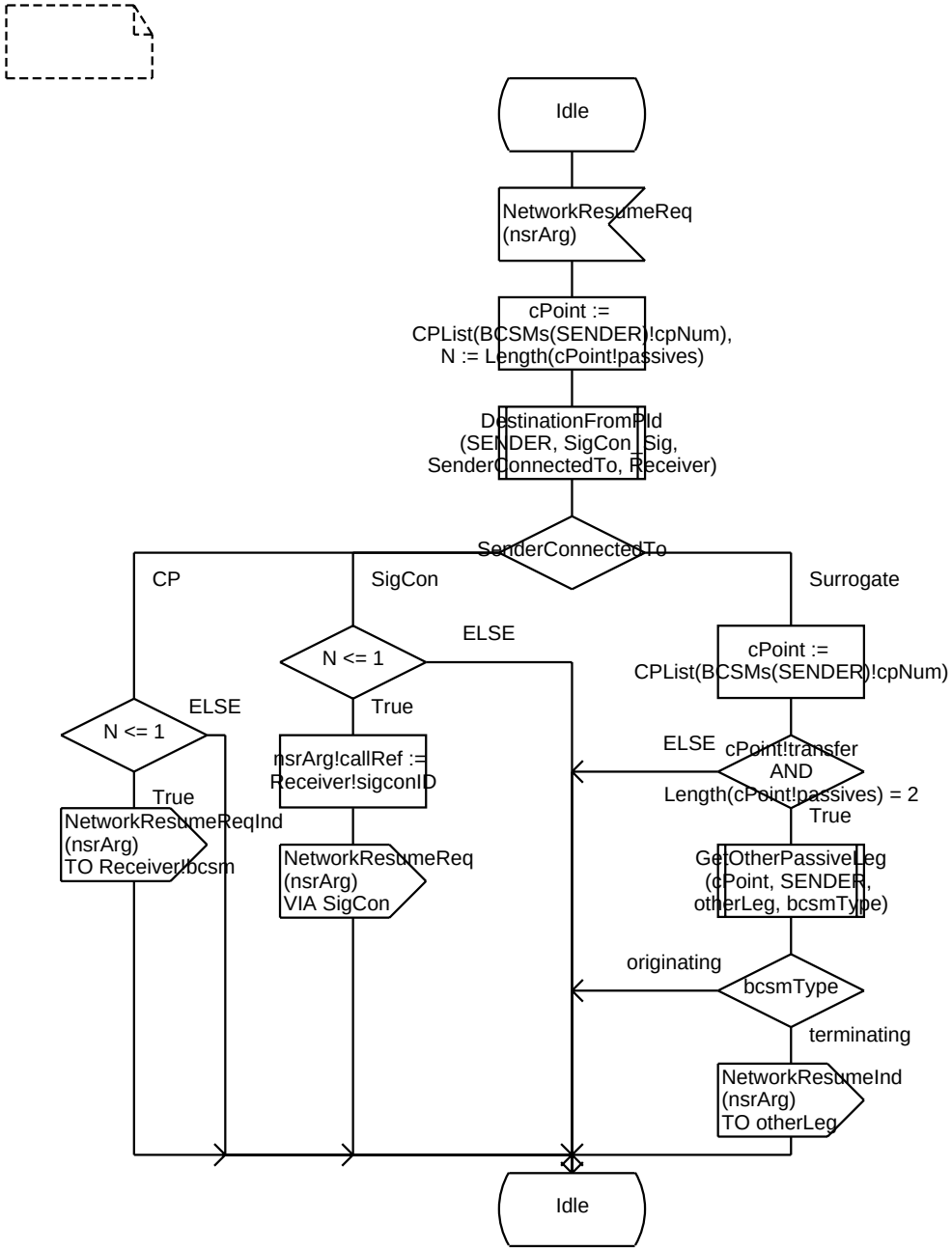
Process Type BCSMControllerProcess 19(32)



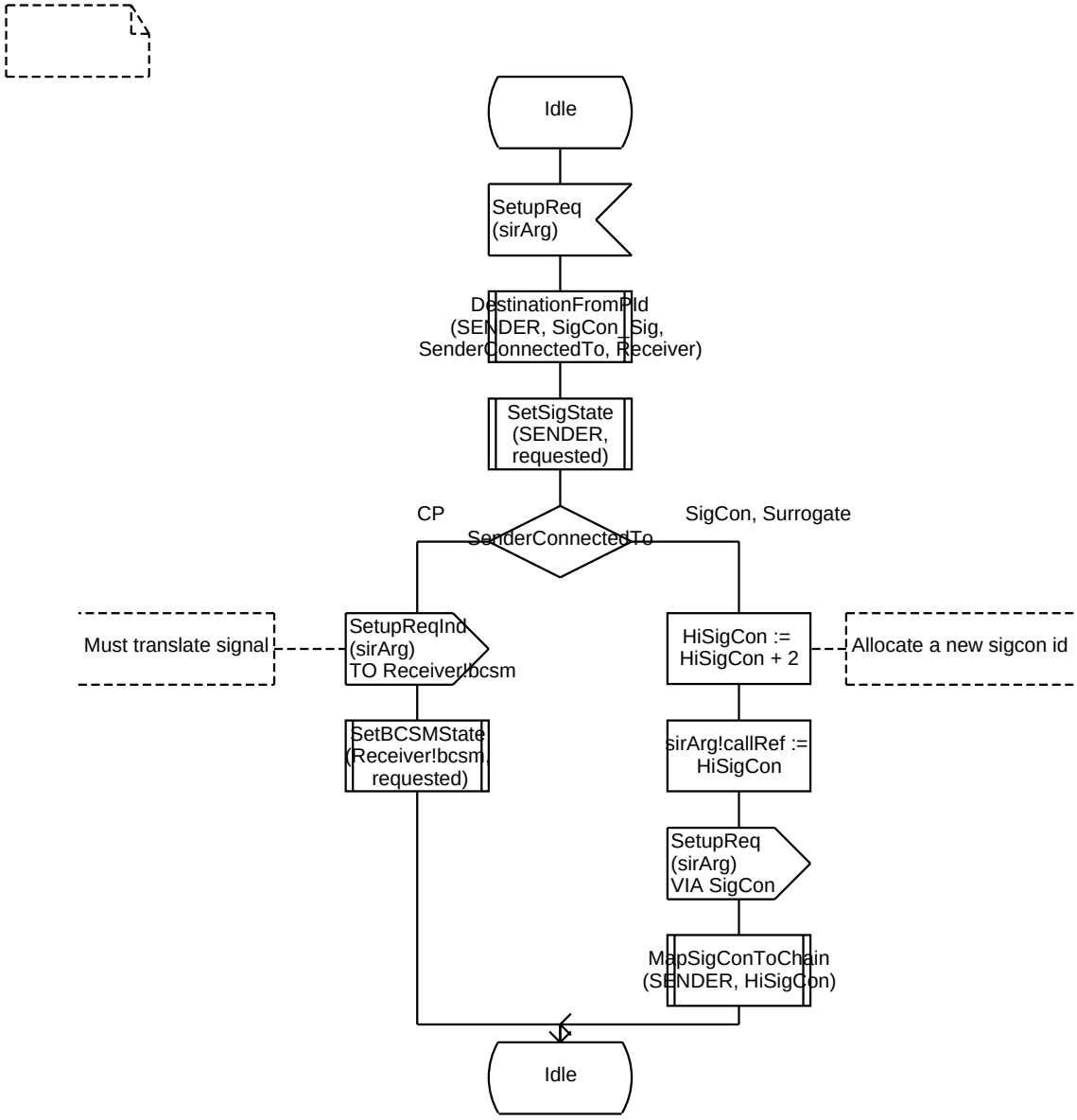
Process Type BCSMControllerProcess 20(32)



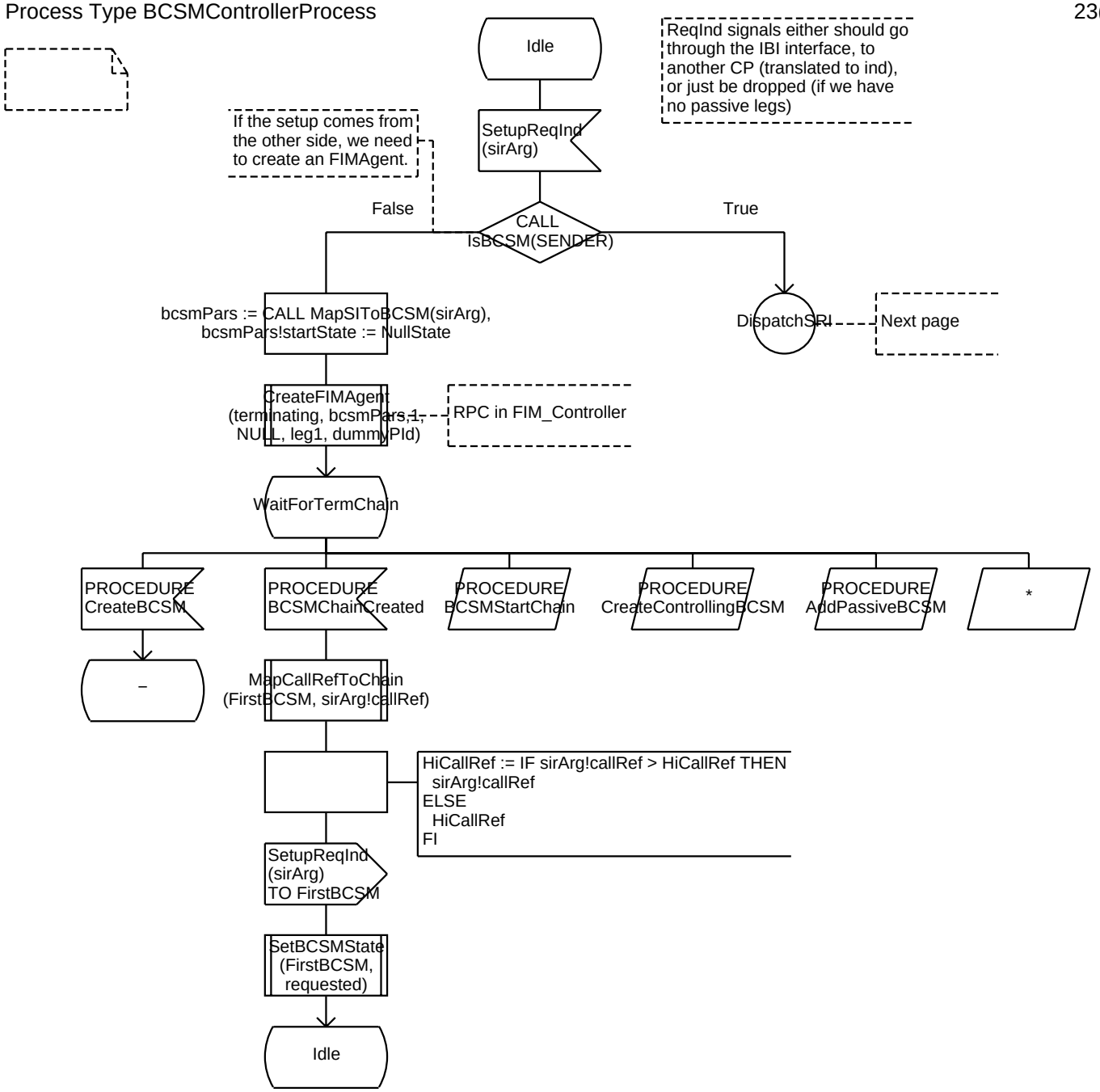
Process Type BCSMControllerProcess 21(32)



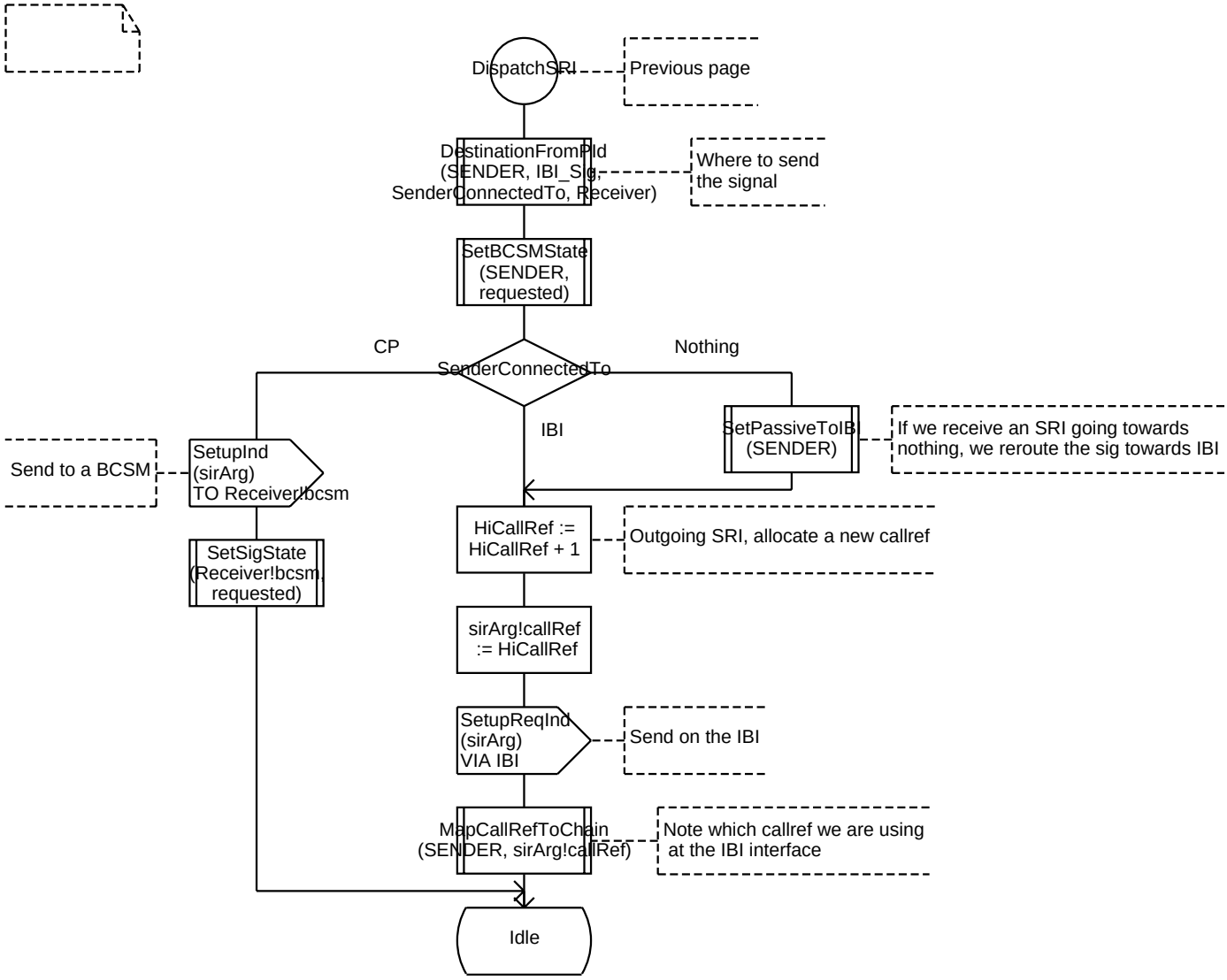
Process Type BCSMControllerProcess 22(32)



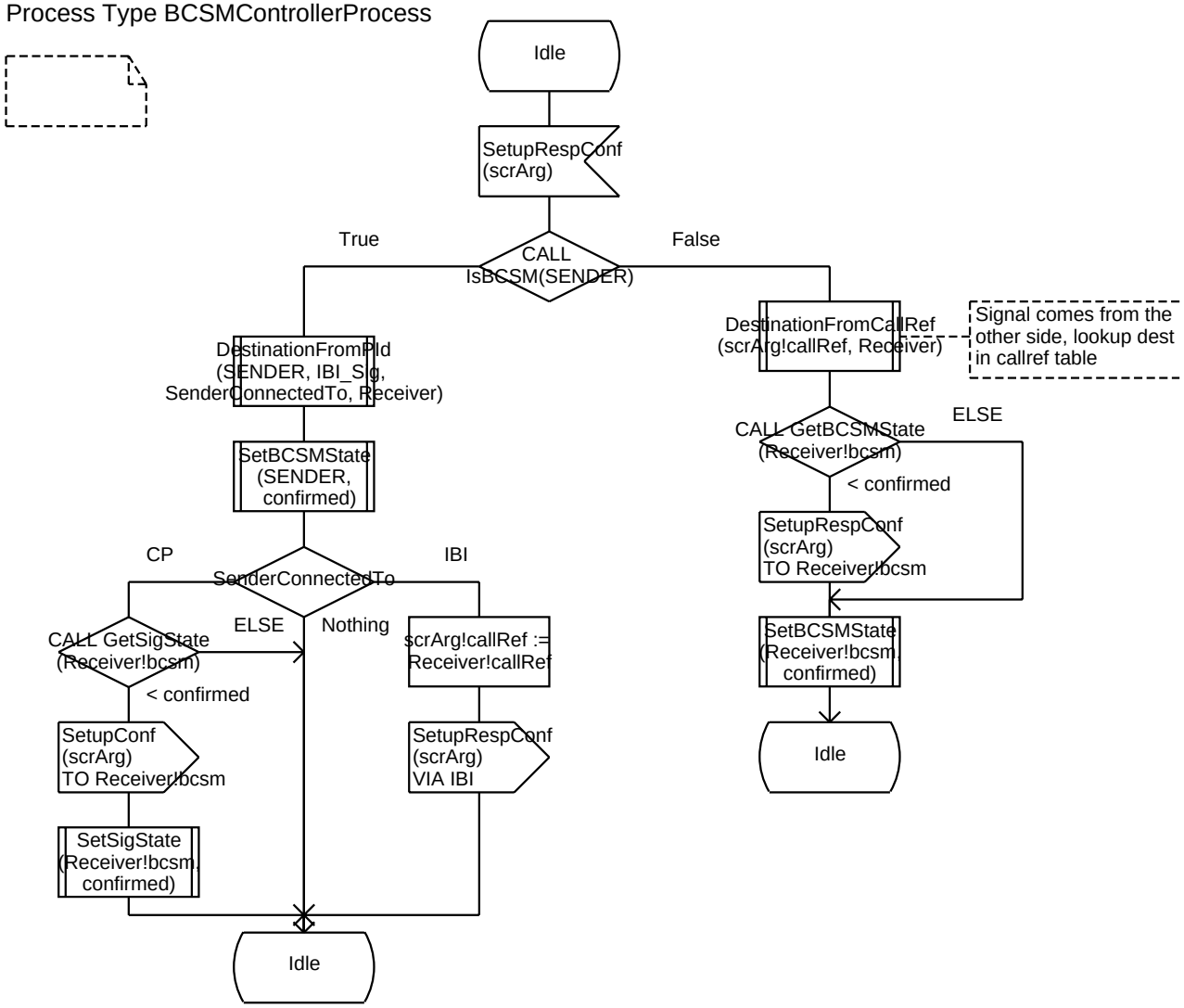
Process Type BCSMControllerProcess 23(32)



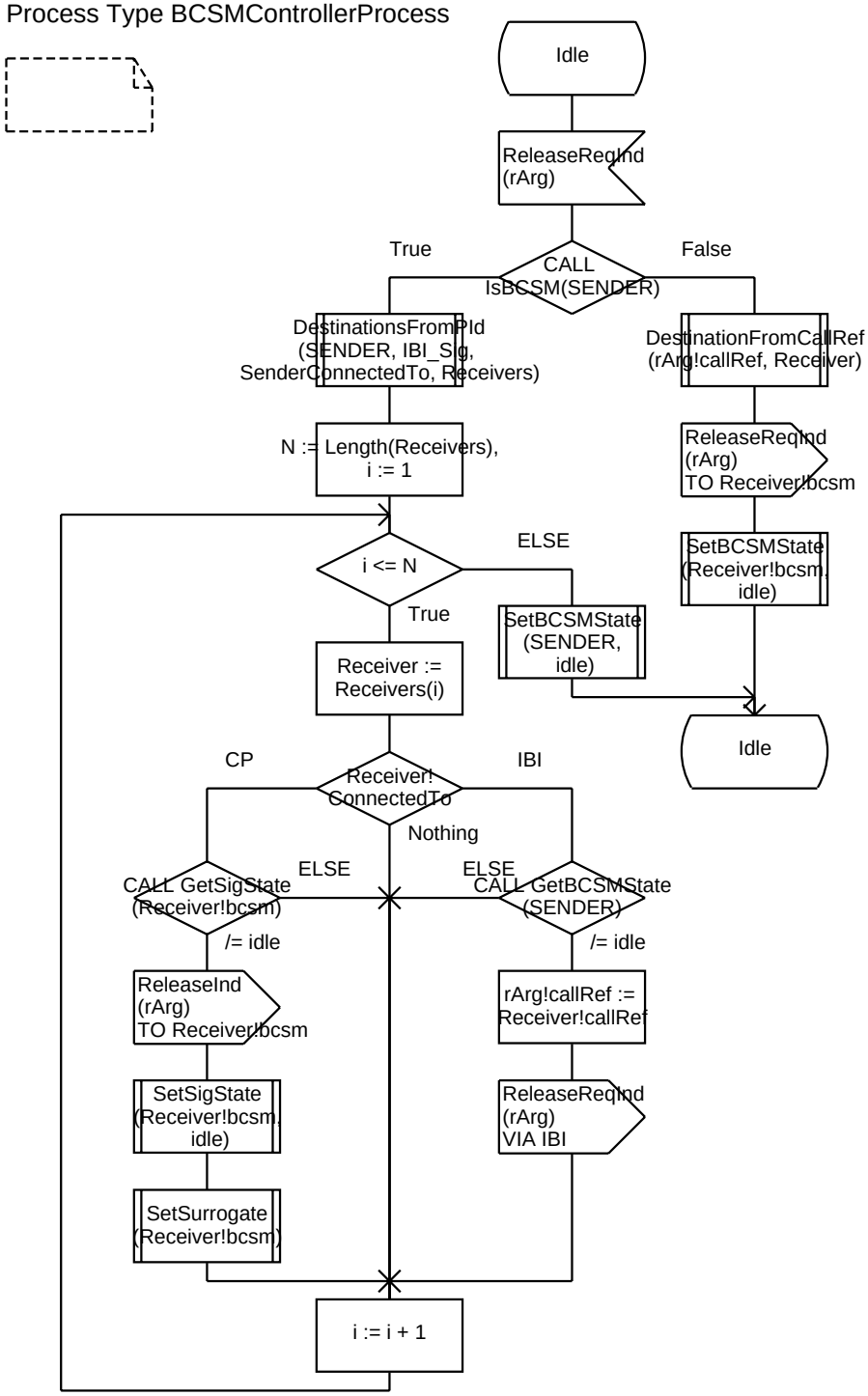
Process Type BCSMControllerProcess 24(32)



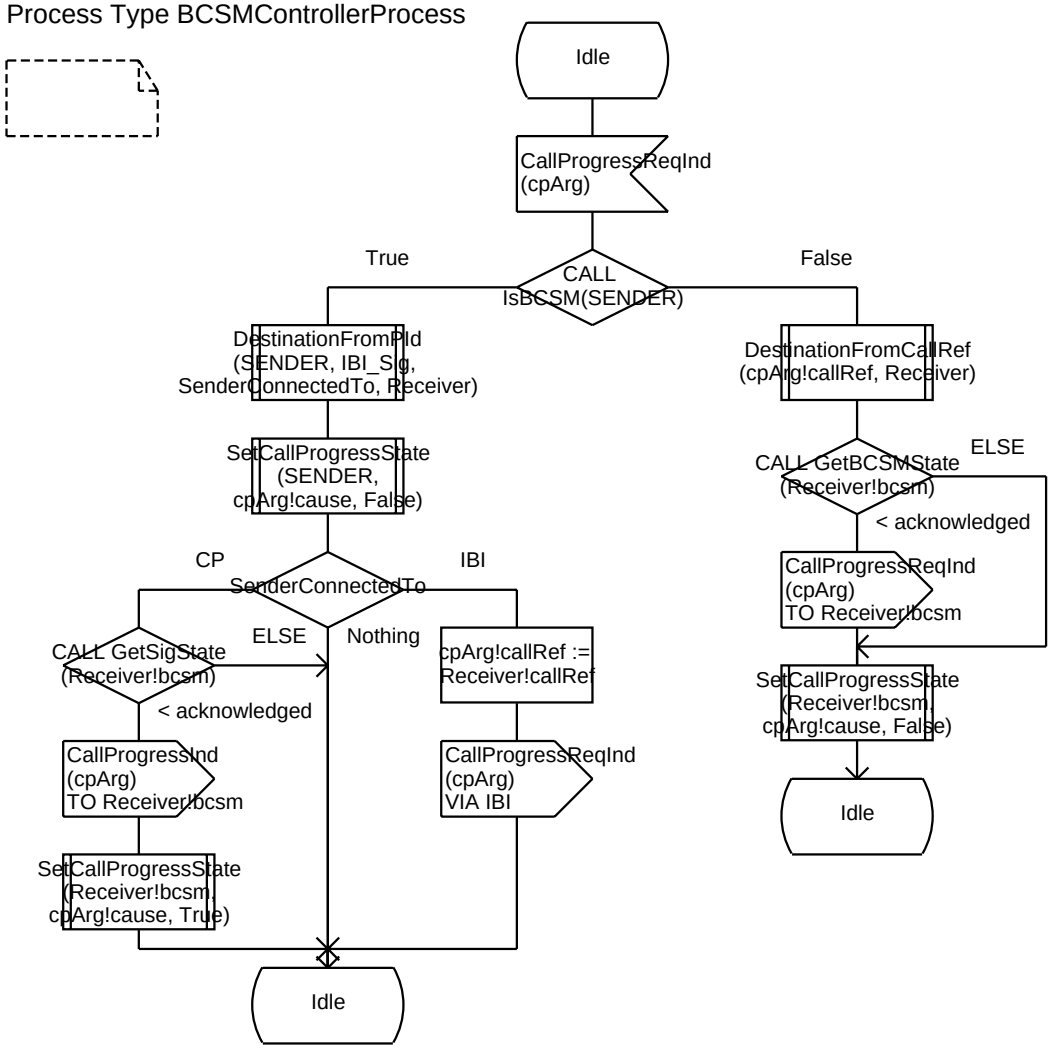
Process Type BCSMControllerProcess 25(32)



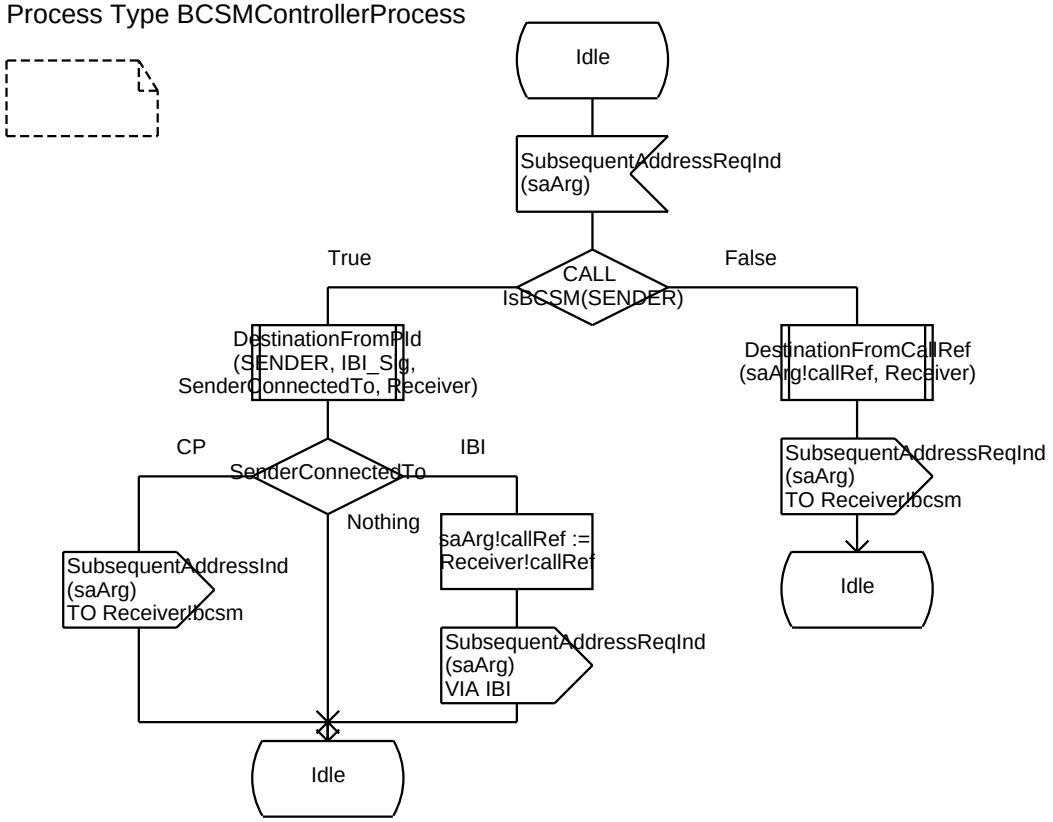
Process Type BCSMControllerProcess 26(32)



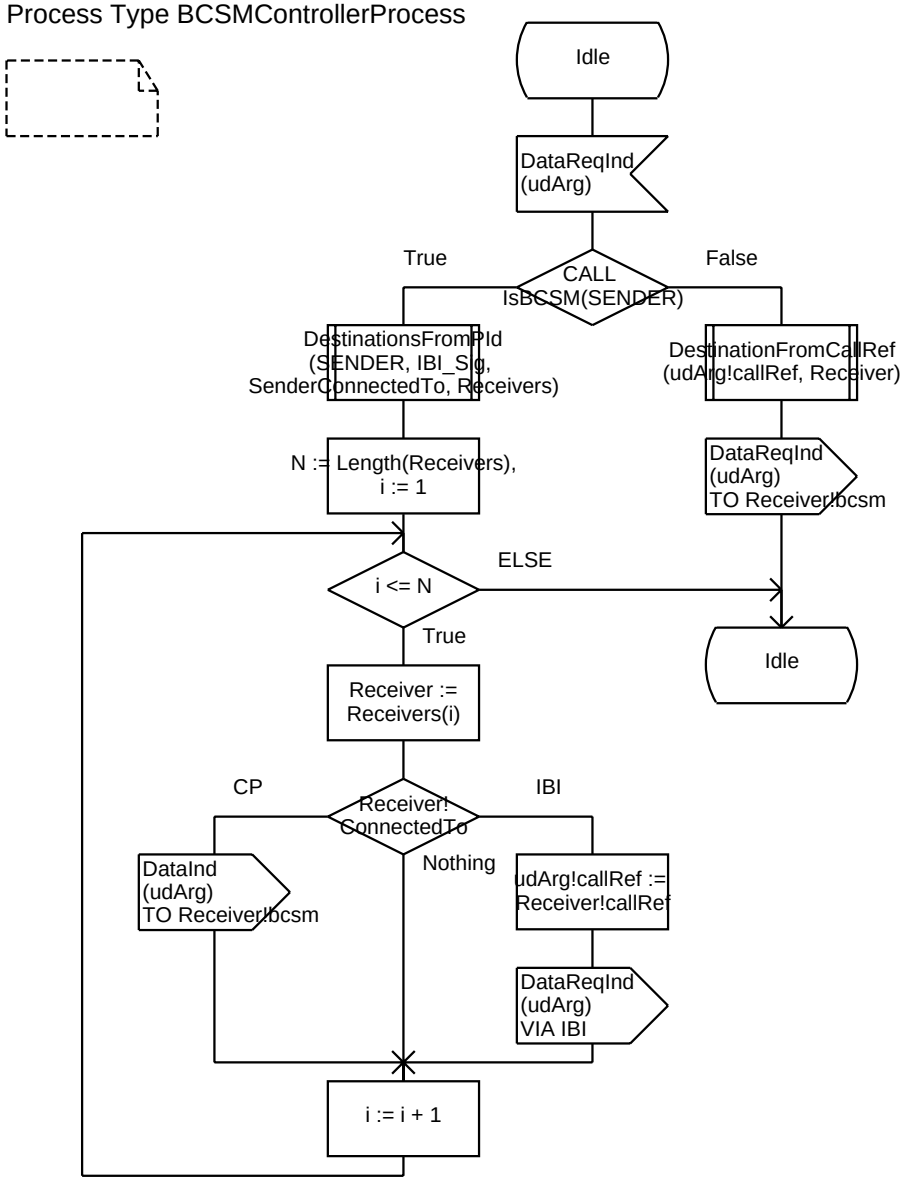
Process Type BCSMControllerProcess 27(32)



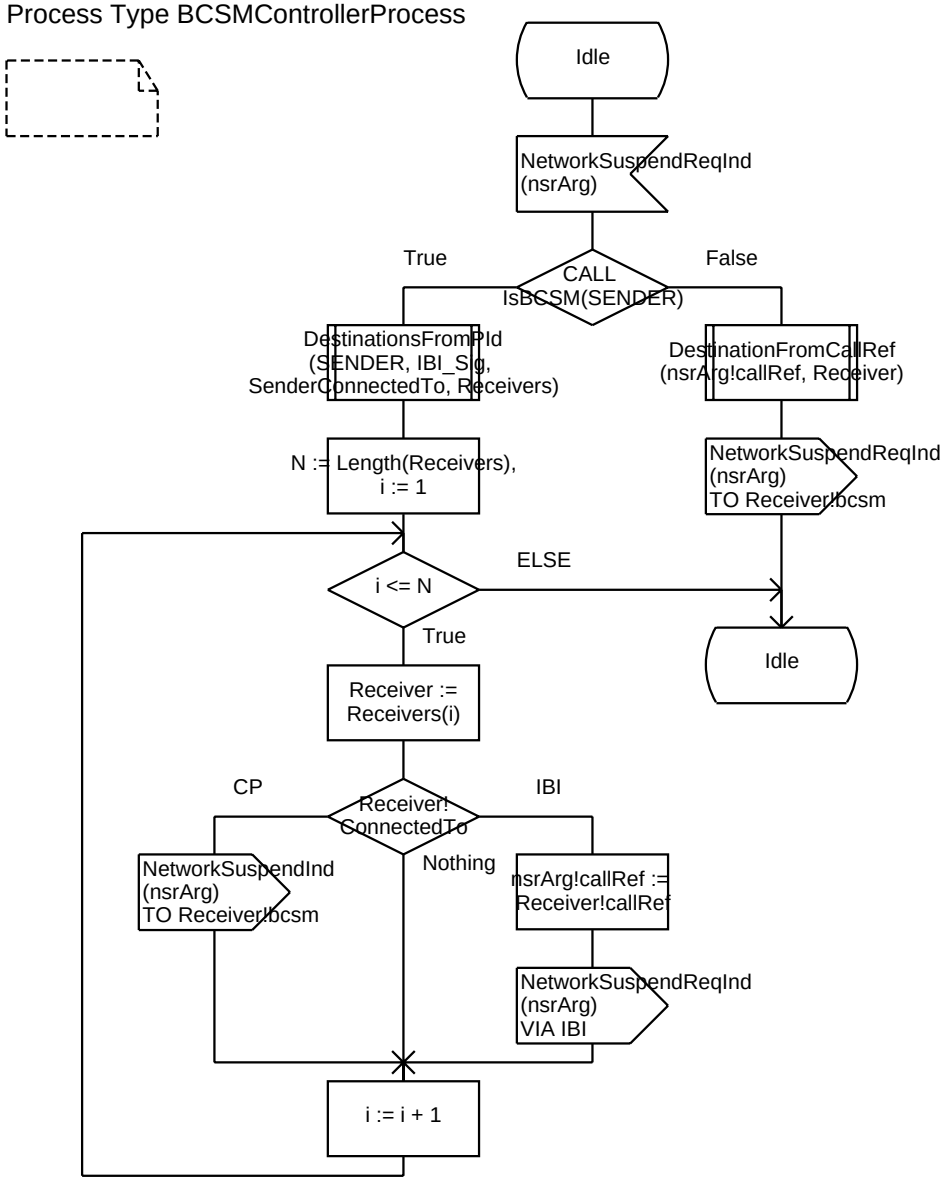
Process Type BCSMControllerProcess 28(32)



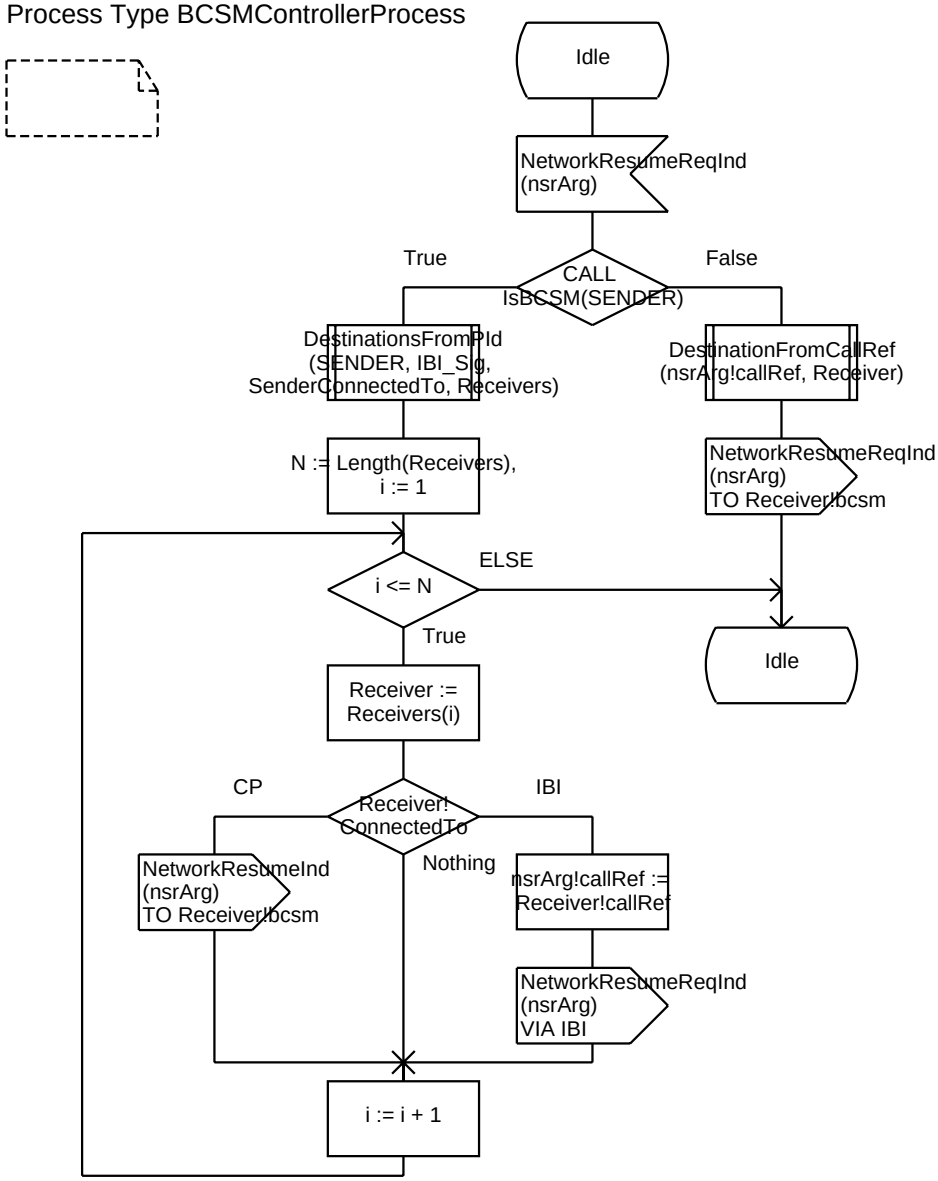
Process Type BCSMControllerProcess 29(32)



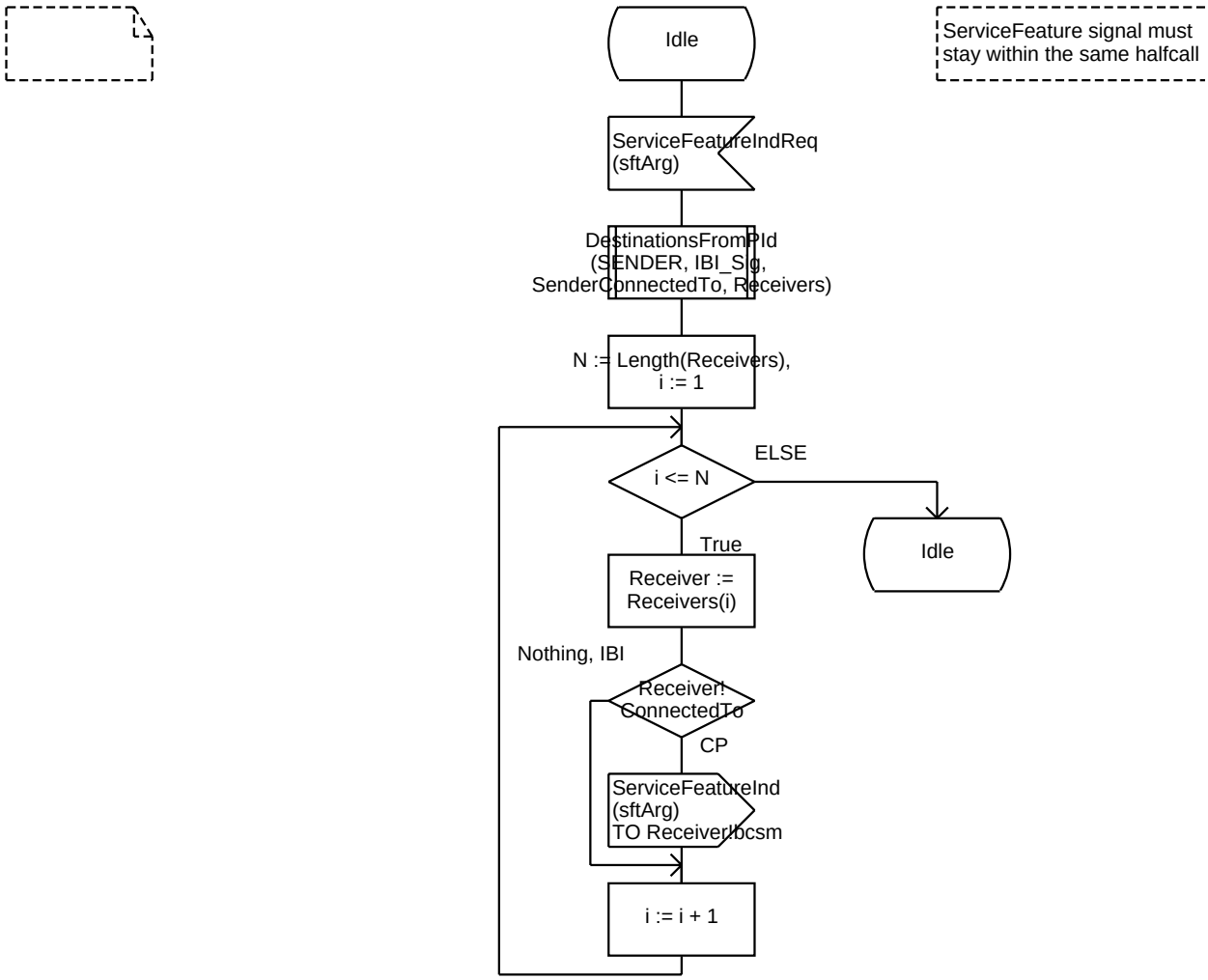
Process Type BCSMControllerProcess 30(32)

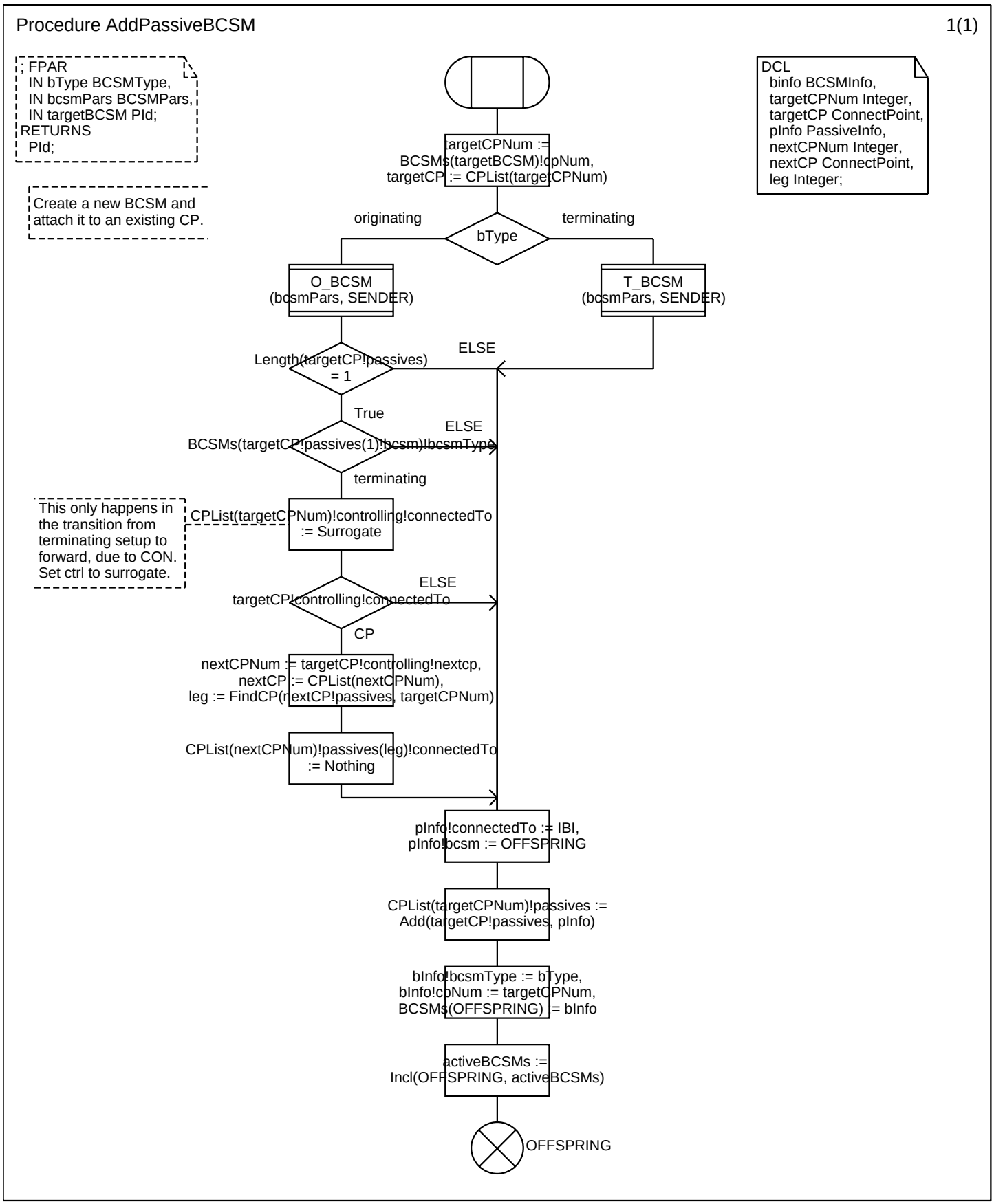


Process Type BCSMControllerProcess 31(32)

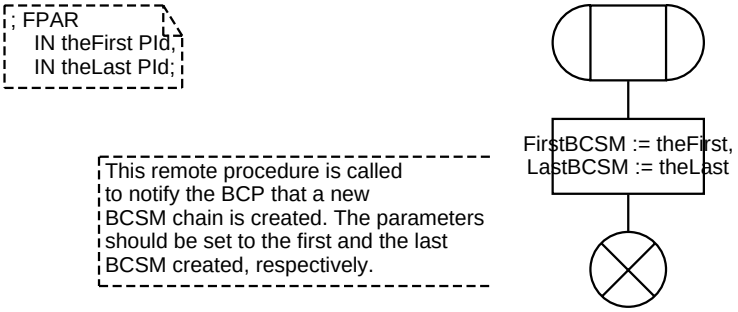


Process Type BCSMControllerProcess 32(32)

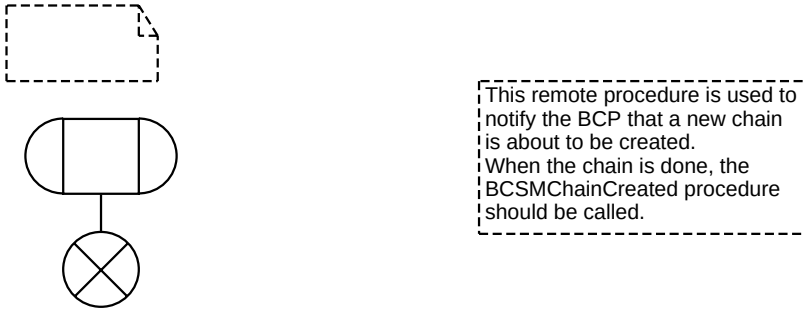


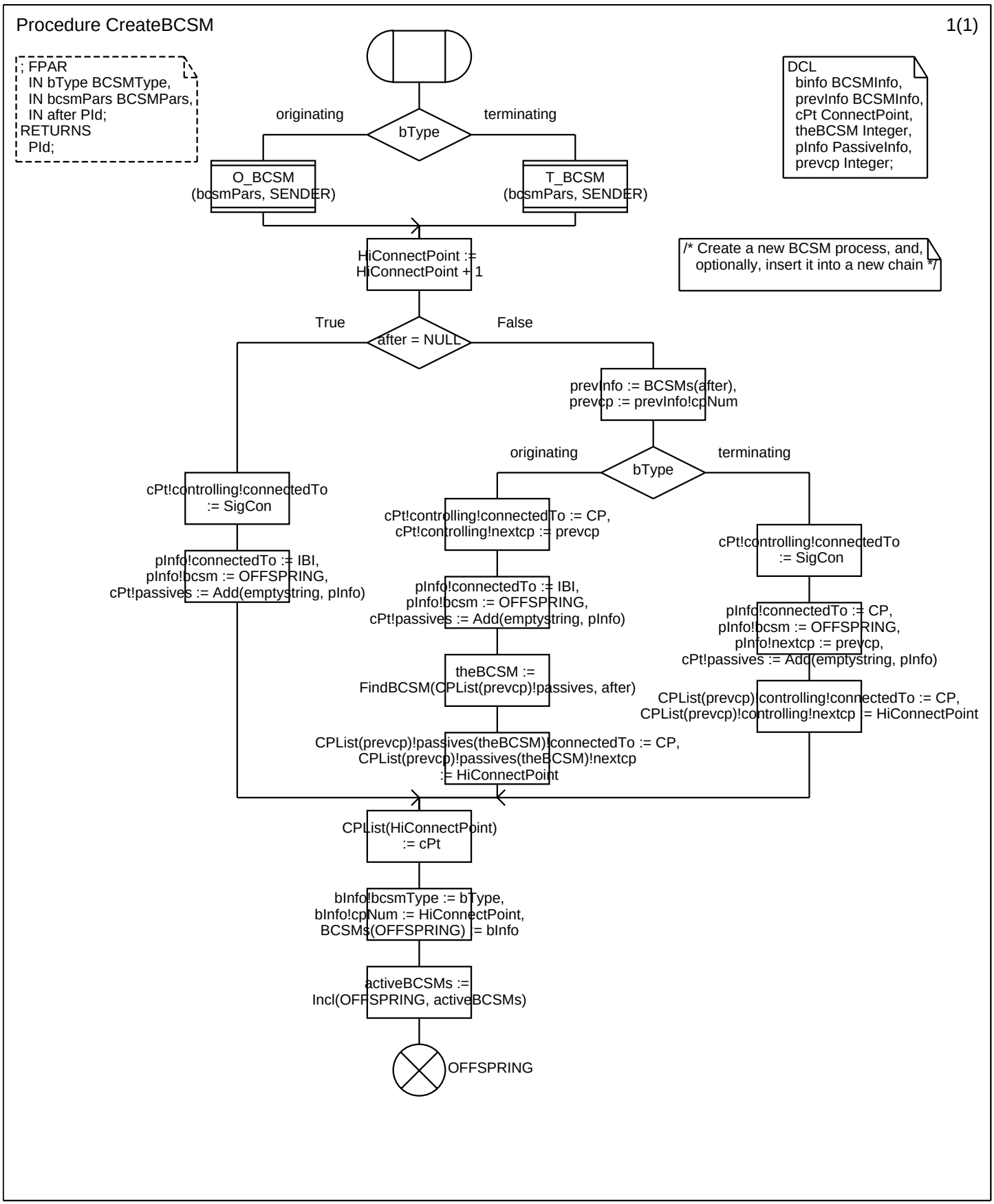

Procedure AddPassiveBCSM
1(1)
; FPAR
IN bType BCSMType,
IN bcsmPars BCSMPars,
IN targetBCSM PId;
RETURNS
PId;
Create a new BCSM and attach it to an existing CP.
DCL
binfo BCSMInfo,
targetCPNum Integer,
targetCP ConnectPoint,
pInfo PassiveInfo,
nextCPNum Integer,
nextCP ConnectPoint,
leg Integer;
targetCPNum := BCSMs(targetBCSM)!cpNum, targetCP := CPList(targetCPNum)
originating
bType
terminating
O_BCSM (bcsmPars, SENDER)
T_BCSM (bcsmPars, SENDER)
Length(targetCP!passives) = 1
ELSE
True
BCSMs(targetCP!passives(1)!bcsm)!bcsmType
ELSE
terminating
This only happens in the transition from terminating setup to forward, due to CON. Set ctrl to surrogate.
CPList(targetCPNum)!controlling!connectedTo := Surrogate
targetCP!controlling!connectedTo
ELSE
CP
nextCPNum := targetCP!controlling!nextcp, nextCP := CPList(nextCPNum), leg := FindCP(nextCP!passives, targetCPNum)
CPList(nextCPNum)!passives(leg)!connectedTo := Nothing
pInfo!connectedTo := IBI, pInfo!bcsm := OFFSPRING
CPList(targetCPNum)!passives := Add(targetCP!passives, pInfo)
bInfo!bcsmType := bType, bInfo!cpNum := targetCPNum, BCSMs(OFFSPRING) := bInfo
activeBCSMs := Incl(OFFSPRING, activeBCSMs)
OFFSPRING

Procedure BCSMChainCreated 1(1)

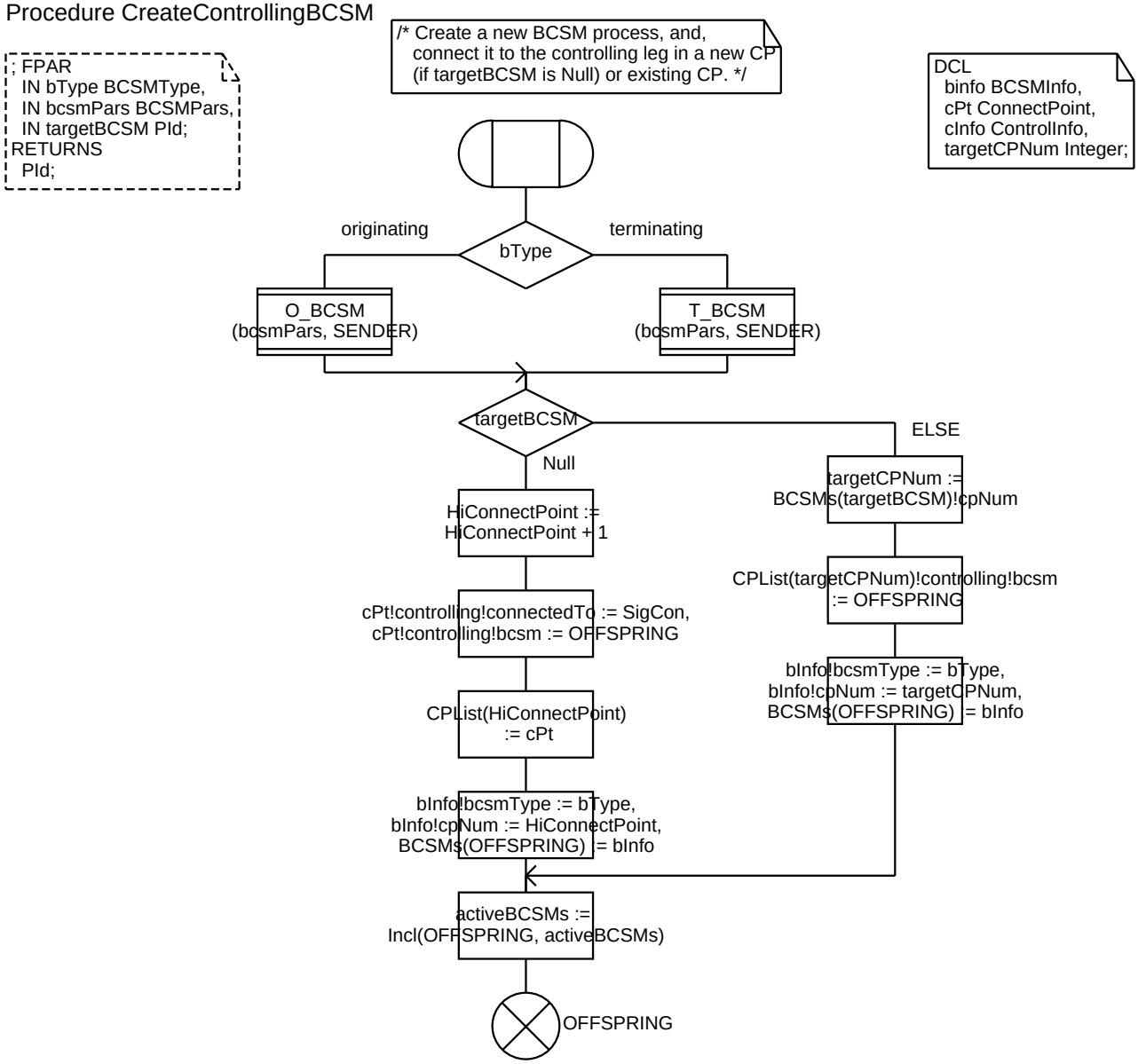


Procedure BCSMStartChain 1(1)

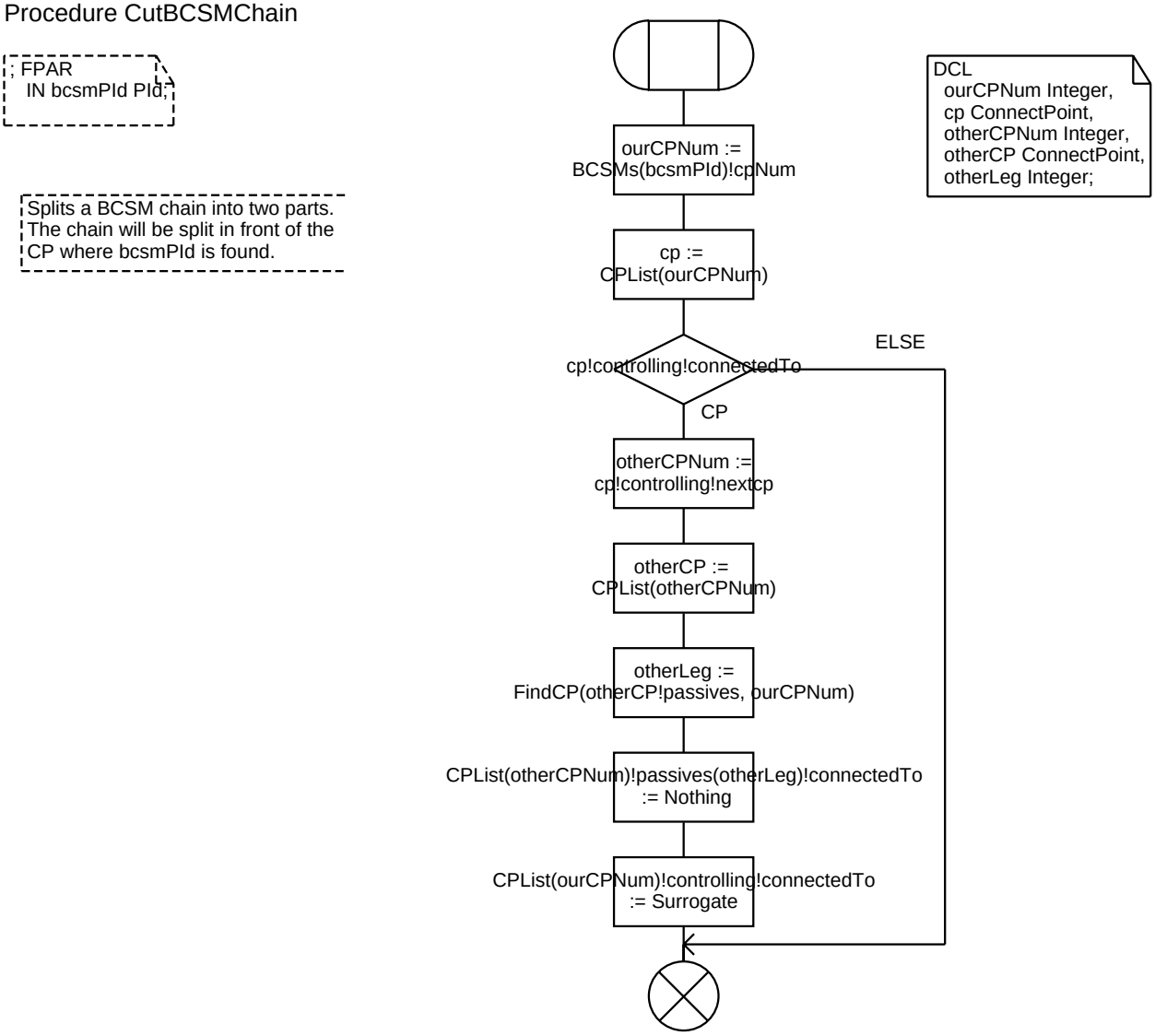



Procedure CreateBCSM
1(1)
; FPAR
IN bType BCSMType,
IN bcsmPars BCSMPars,
IN after PId;
RETURNS
PId;
DCL
binfo BCSMInfo,
prevInfo BCSMInfo,
cPt ConnectPoint,
theBCSM Integer,
pInfo PassiveInfo,
prevcp Integer;
originating
bType
terminating
O_BCSM
(bcsmPars, SENDER)
T_BCSM
(bcsmPars, SENDER)
HiConnectPoint :=
HiConnectPoint + 1
/* Create a new BCSM process, and,
optionally, insert it into a new chain */
True
after = NULL
False
prevInfo := BCSMs(after),
prevcp := prevInfo!cpNum
originating
bType
terminating
cPt!controlling!connectedTo
:= SigCon
cPt!controlling!connectedTo := CP,
cPt!controlling!nextcp := prevcp
cPt!controlling!connectedTo
:= SigCon
pInfo!connectedTo := IBI,
pInfo!bcsm := OFFSPRING,
cPt!passives := Add(emptystring, pInfo)
pInfo!connectedTo := IBI,
pInfo!bcsm := OFFSPRING,
cPt!passives := Add(emptystring, pInfo)
pInfo!connectedTo := CP,
pInfo!bcsm := OFFSPRING,
pInfo!nextcp := prevcp,
cPt!passives := Add(emptystring, pInfo)
theBCSM :=
FindBCSM(CPList(prevcp)!passives, after)
CPList(prevcp)!controlling!connectedTo := CP,
CPList(prevcp)!controlling!nextcp := HiConnectPoint
CPList(prevcp)!passives(theBCSM)!connectedTo := CP,
CPList(prevcp)!passives(theBCSM)!nextcp
:= HiConnectPoint
CPList(HiConnectPoint)
:= cPt
bInfo!bcsmType := bType,
bInfo!cpNum := HiConnectPoint,
BCSMs(OFFSPRING) := bInfo
activeBCSMs :=
Incl(OFFSPRING, activeBCSMs)
OFFSPRING

Procedure CreateControllingBCSM 1(1)



Procedure CutBCSMChain 1(1)

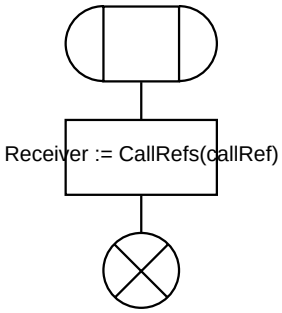


Procedure DestinationFromCallRef 1(1)

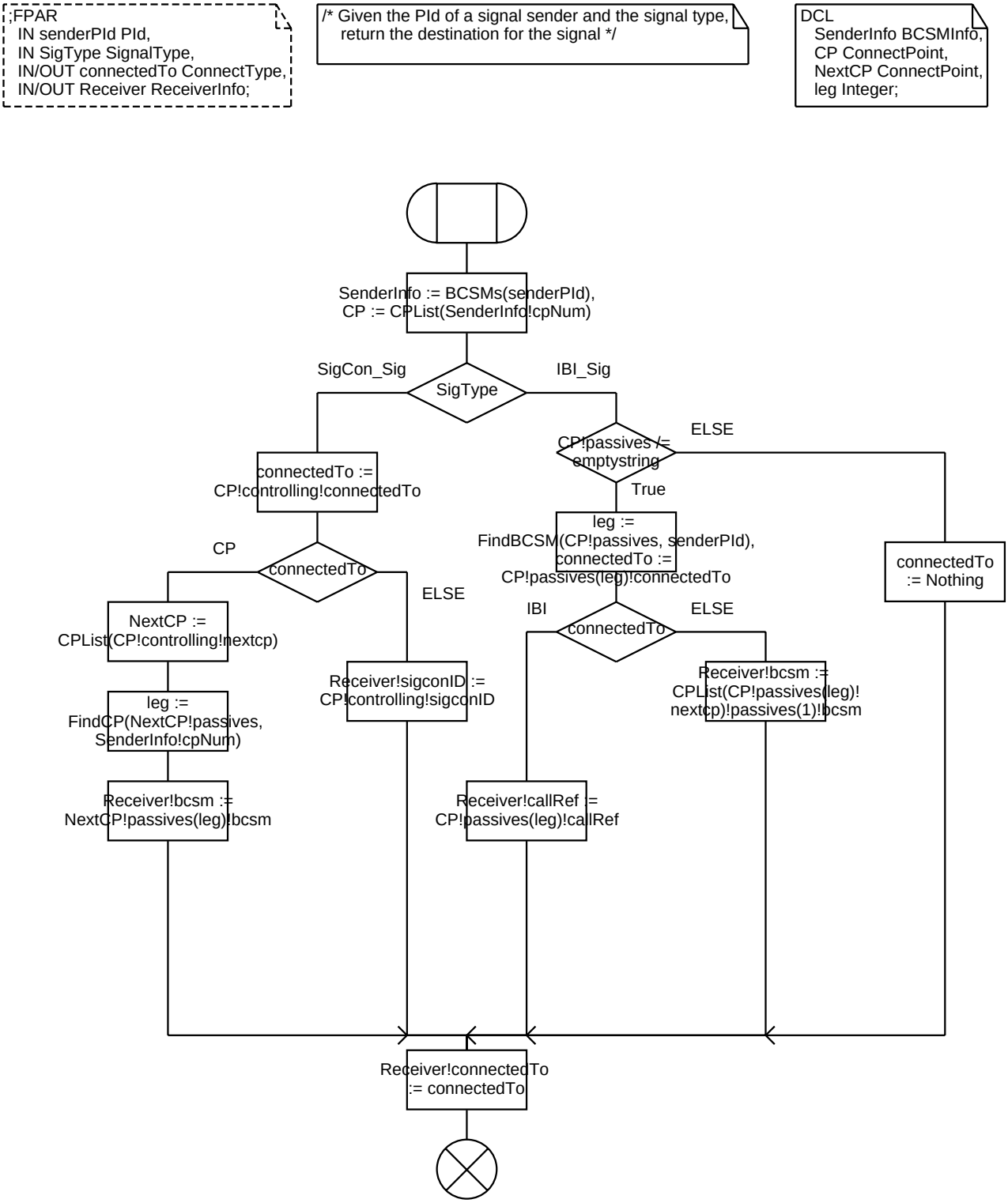
```
;FPAR
  IN callRef CallRef,
  IN/OUT Receiver ReceiverInfo;
```

```
/* Given a callRef, return the destination for the signal */
```

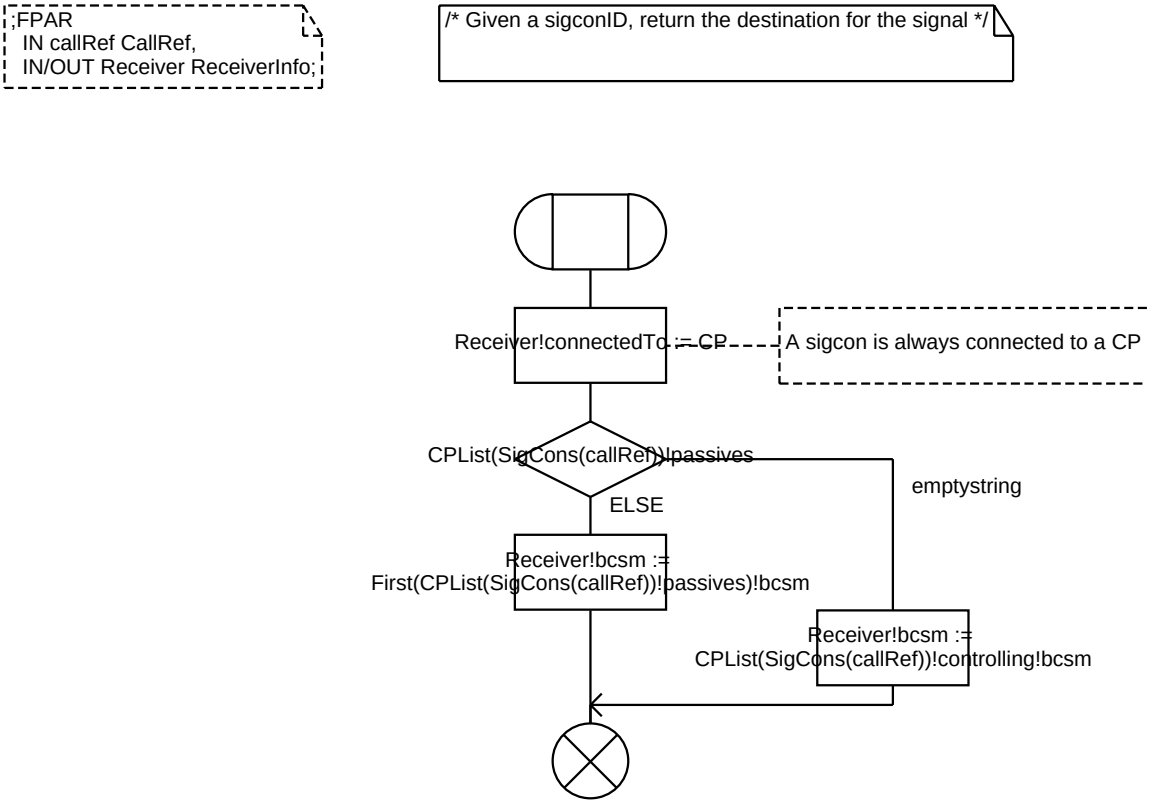
```
DCL
  CP ConnectPoint;
```



Procedure DestinationFromPId 1(1)

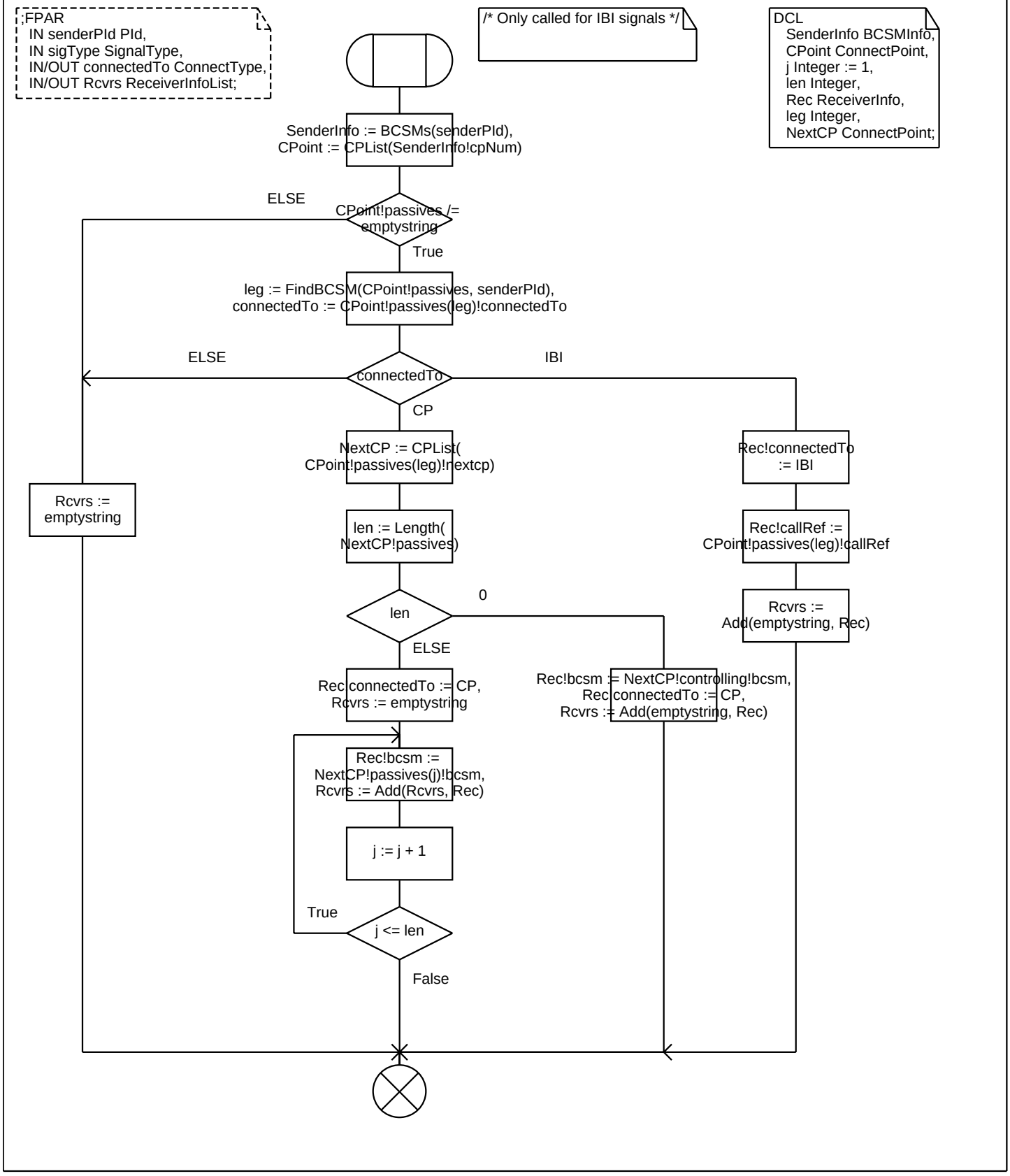


Procedure DestinationFromSigCon 1(1)

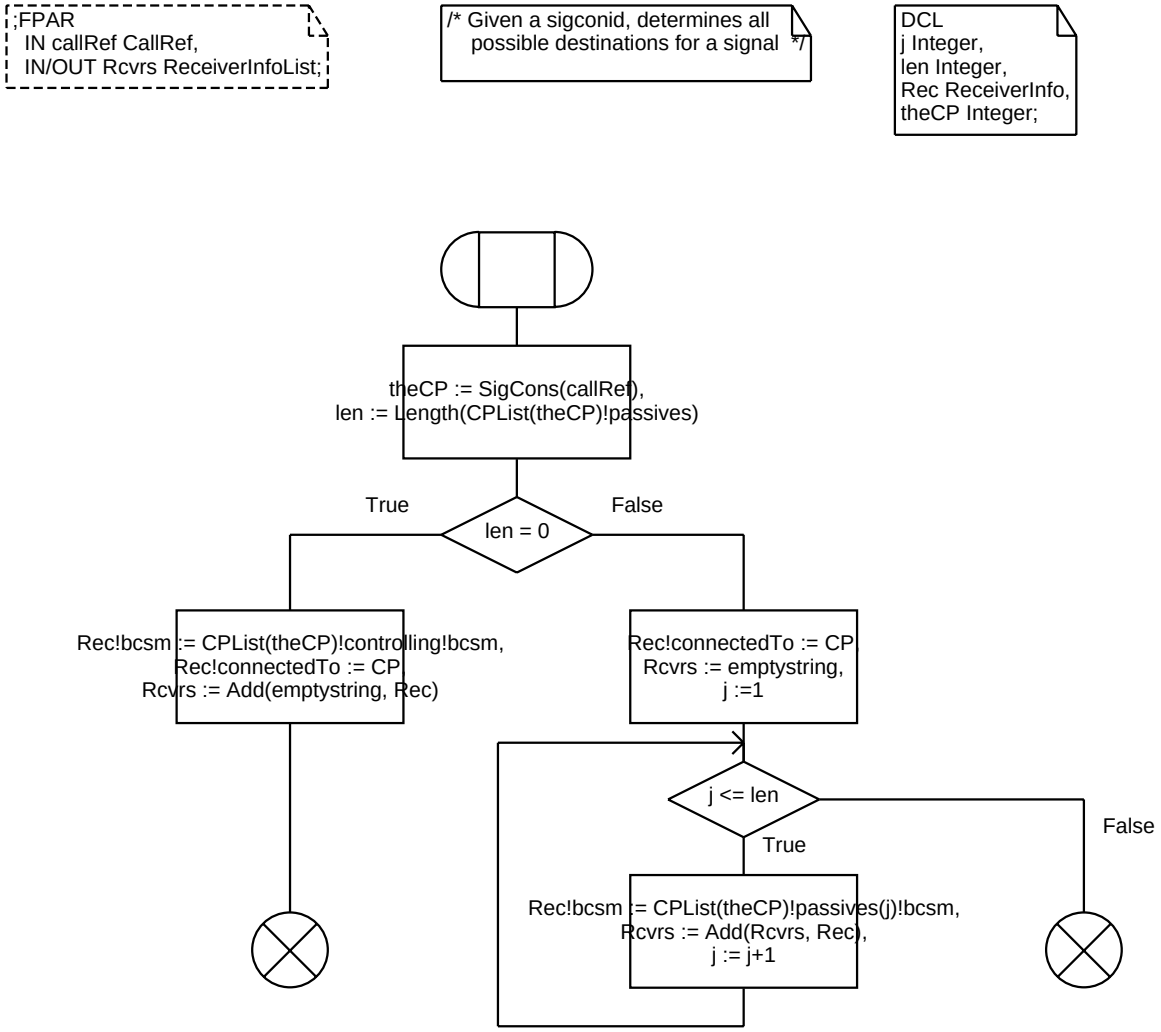


DestinationsFromPId

Procedure DestinationsFromPId 1(1)



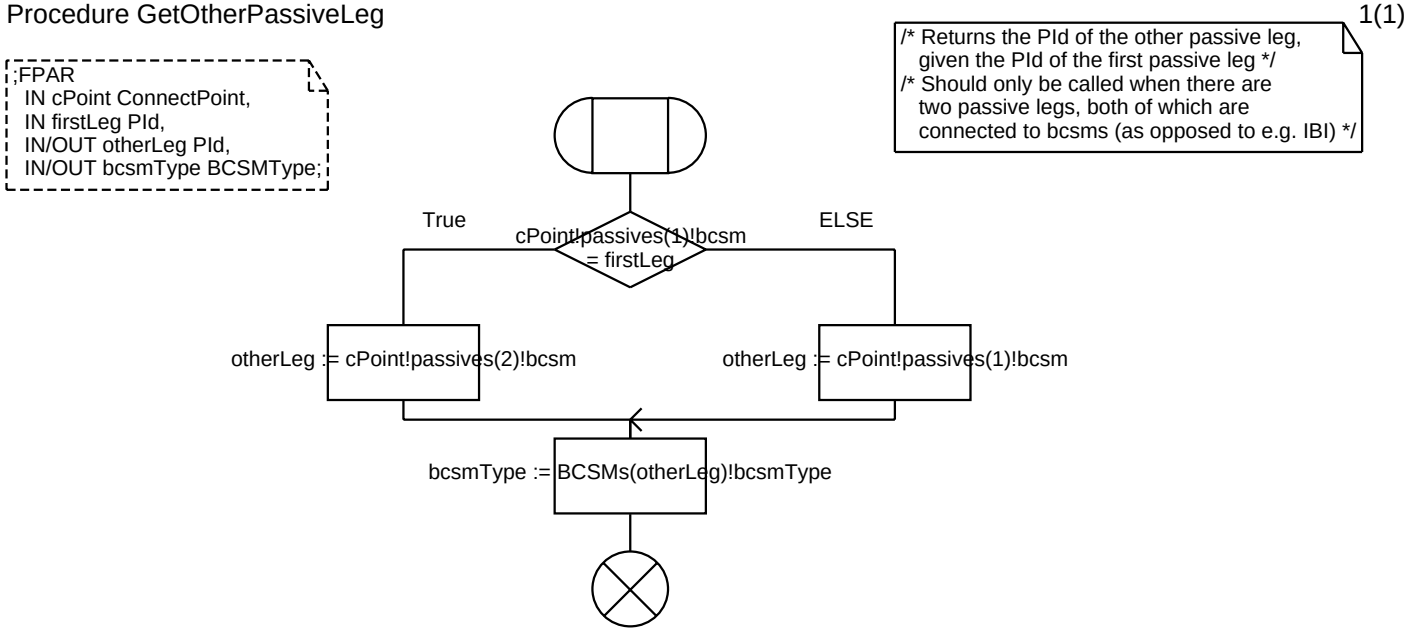
Procedure DestinationsFromSigCon 1(1)



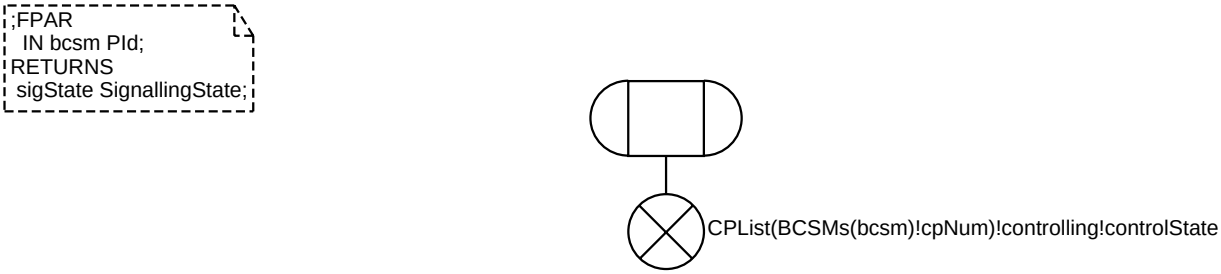

```
/*
DCL j Integer;
DCL len Integer;
DCL Rec ReceiverInfo;
DCL theCP Integer;
theCP := SigCons(callRef);
len := Length(CPList(theCP)!passives);
IF (len = 0) {
  Rec!bcsm := CPList(theCP)!controlling!bcsm;
  Rec!connectedTo := CP;
  Rcvrs := Add(emptystring, Rec);
} ELSE {
  Rec!connectedTo := CP;
  Rcvrs := emptystring;
  FOR (j := 1, j <= len, j + 1) {
    Rec!bcsm := CPList(theCP)!passives(j)!bcsm;
    Rcvrs := Add(Rcvrs, Rec);
  }
}
*/
```

Procedure GetBCSMState 1(1)

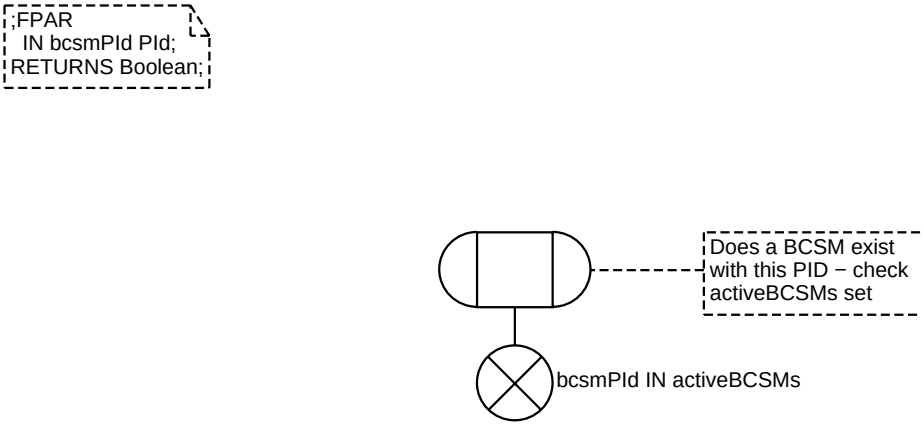
Procedure GetOtherPassiveLeg 1(1)



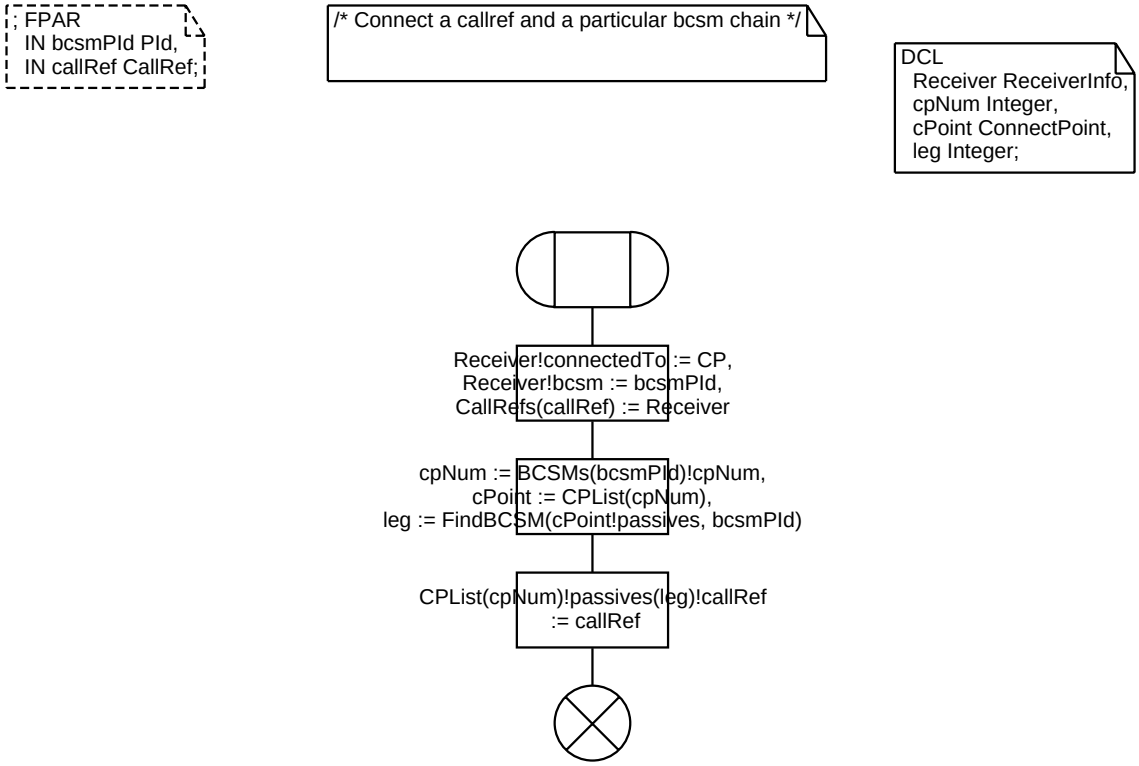
Procedure GetSigState 1(1)



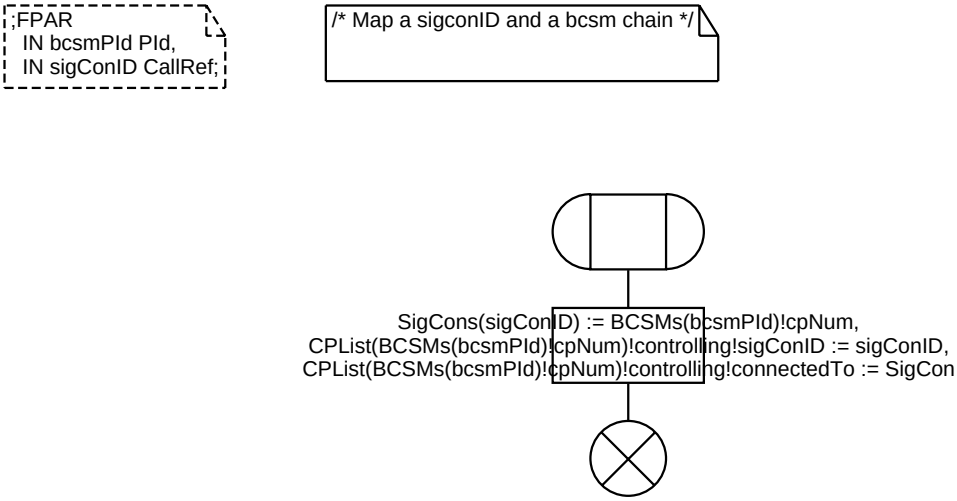
Procedure <<Process Type BCSMControllerProcess>> IsBCSM 1(1)



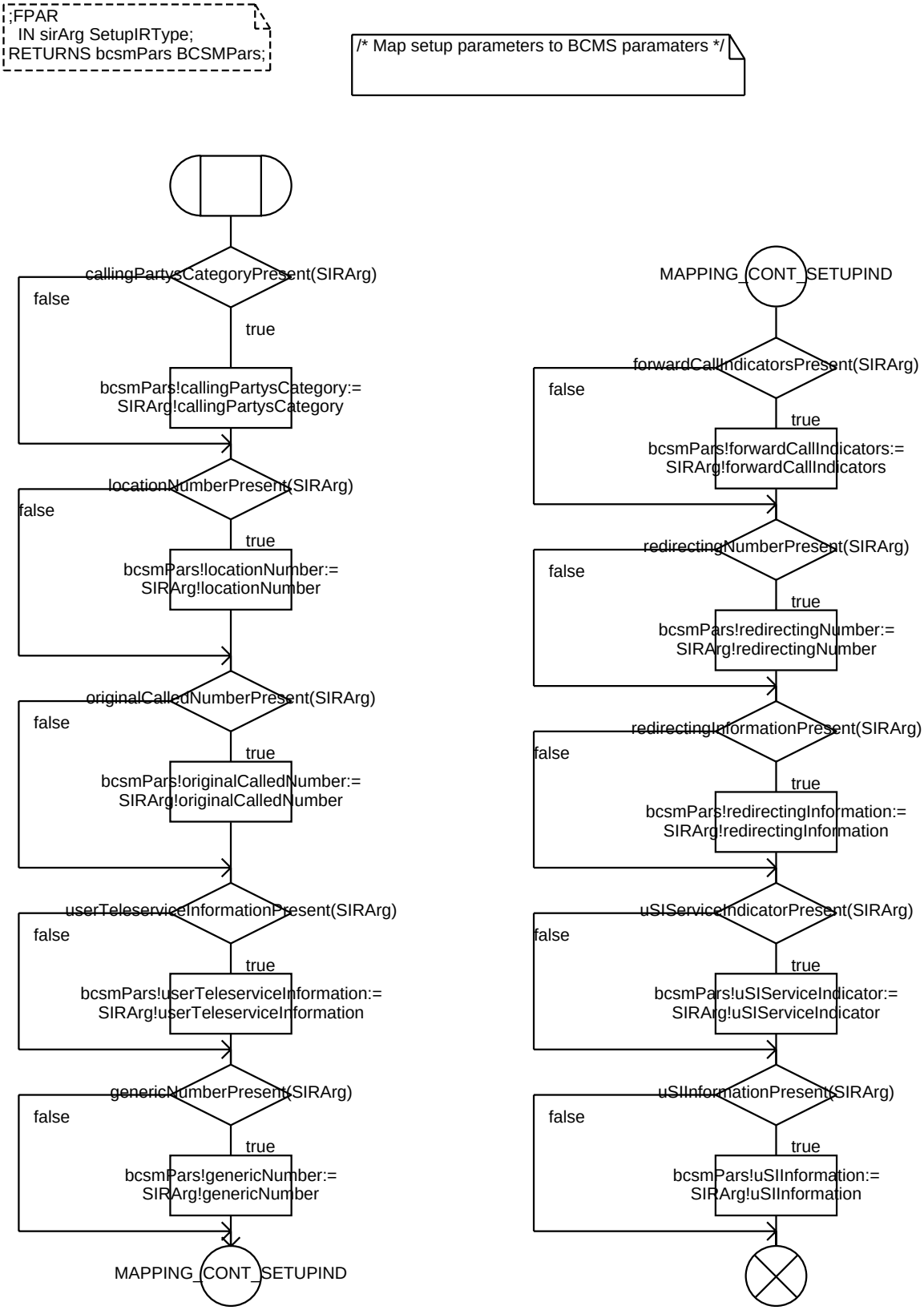
Procedure MapCallRefToChain 1(1)



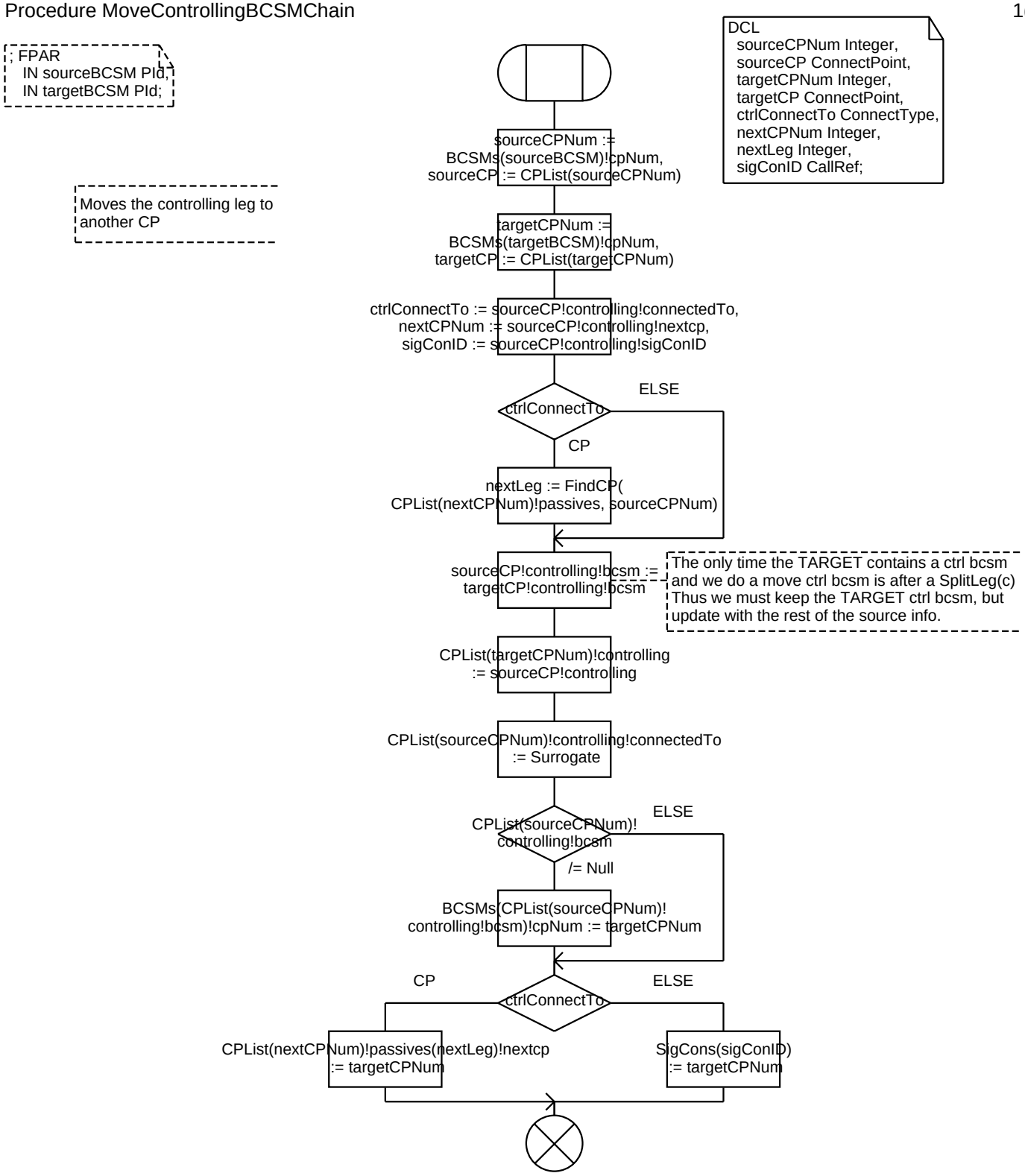
Procedure MapSigConToChain 1(1)



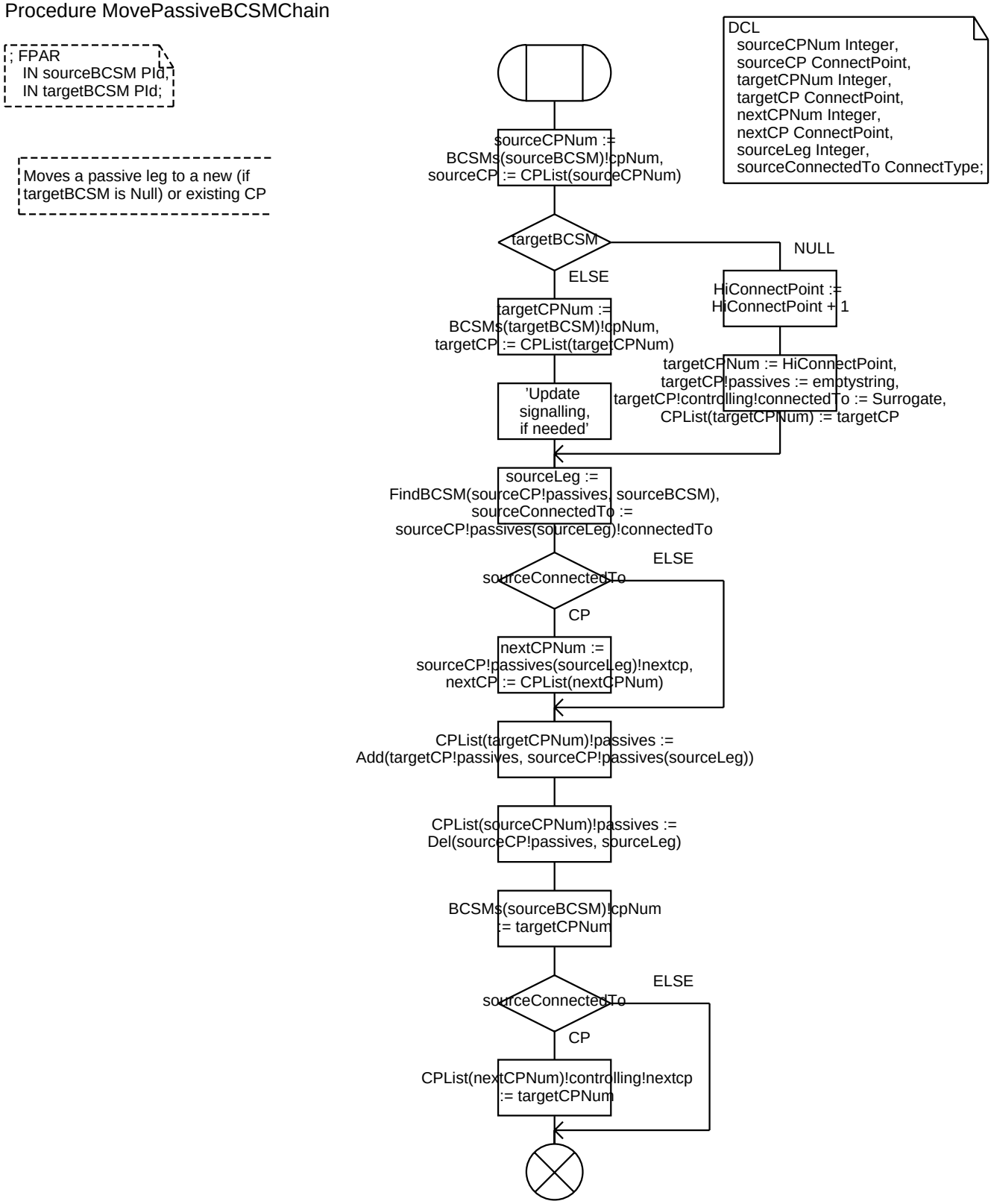
Procedure <<Process Type BCSMControllerProcess>> MapSIToBCSM 1(1)



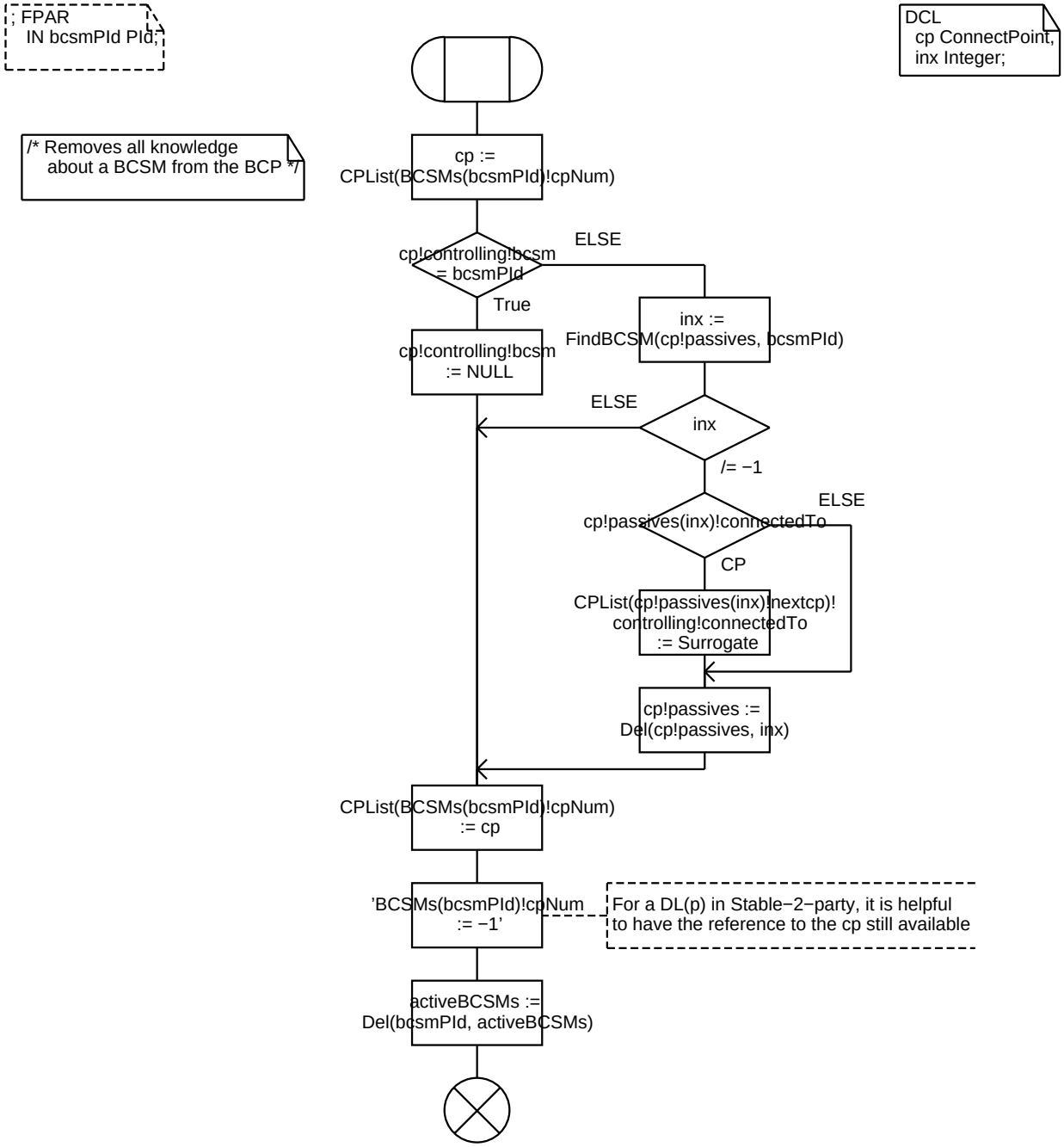
Procedure MoveControllingBCSMChain 1(1)



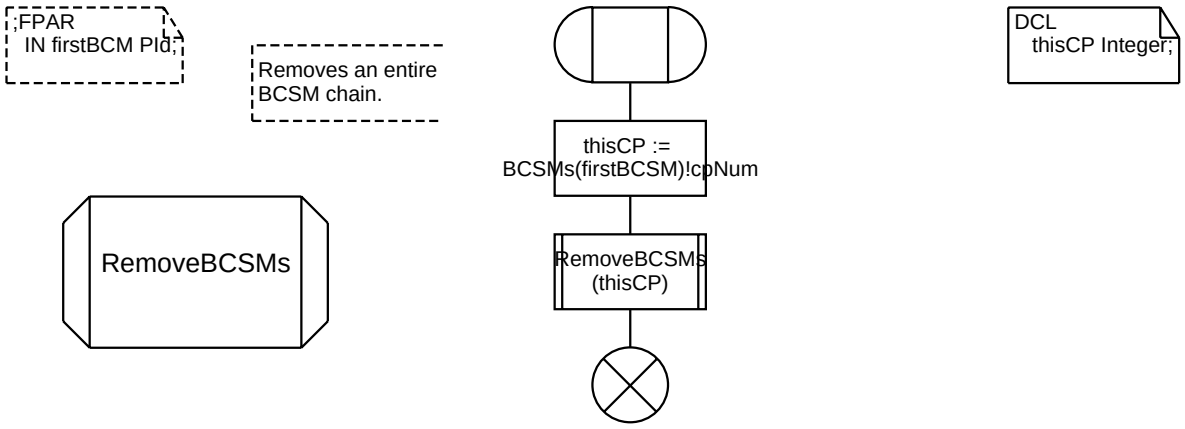
Procedure MovePassiveBCSMChain 1(1)



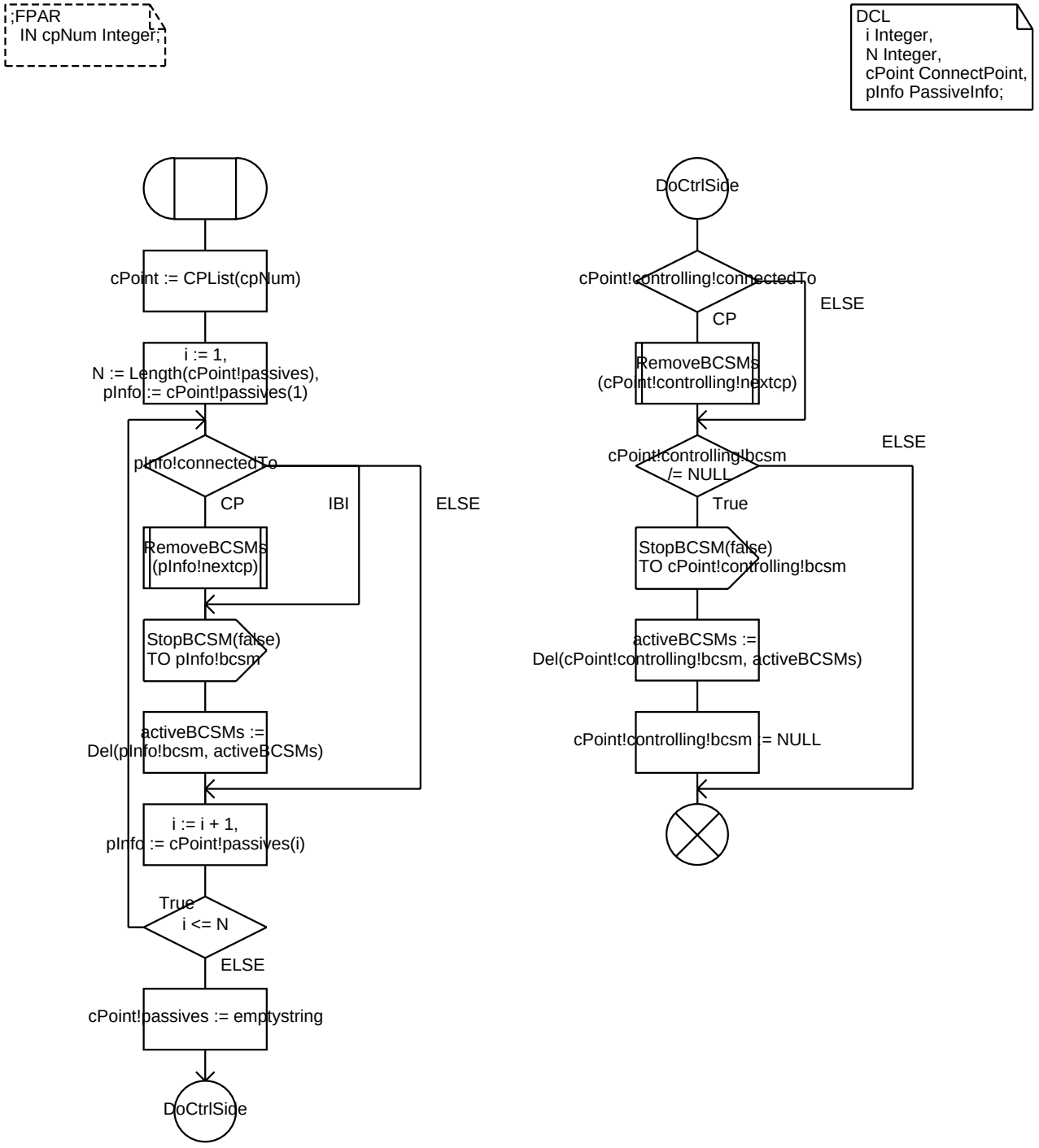
Procedure RemoveBCSM 1(1)



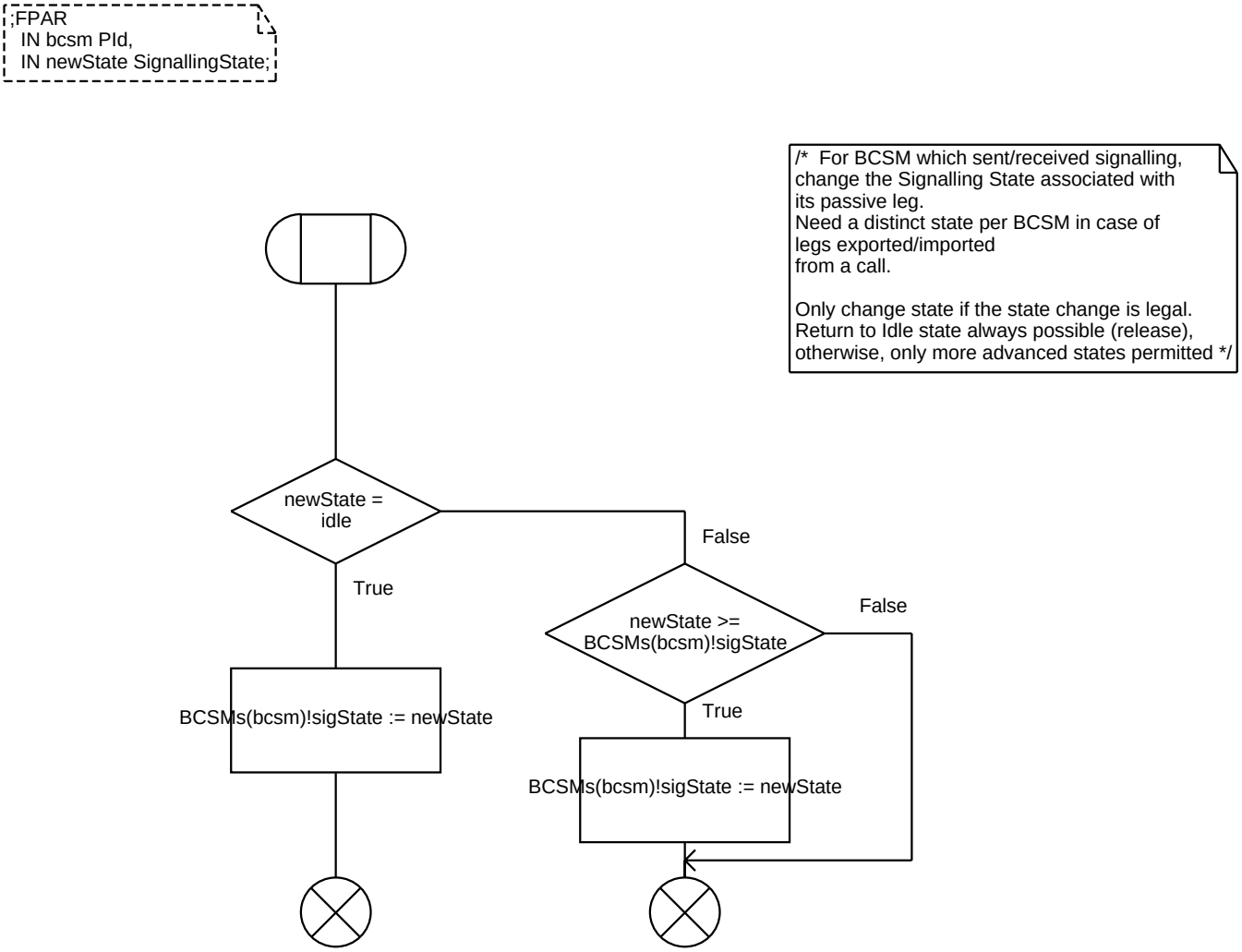
Procedure RemoveBCSMChain 1(1)



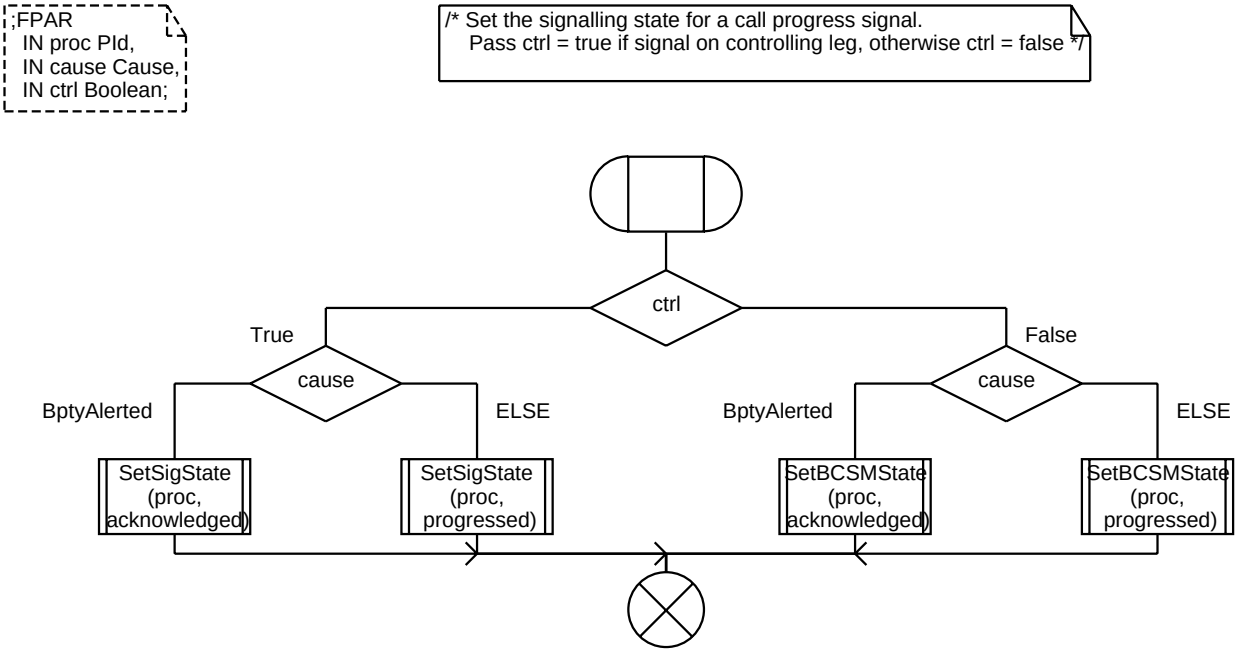
Procedure RemoveBCSMs 1(1)



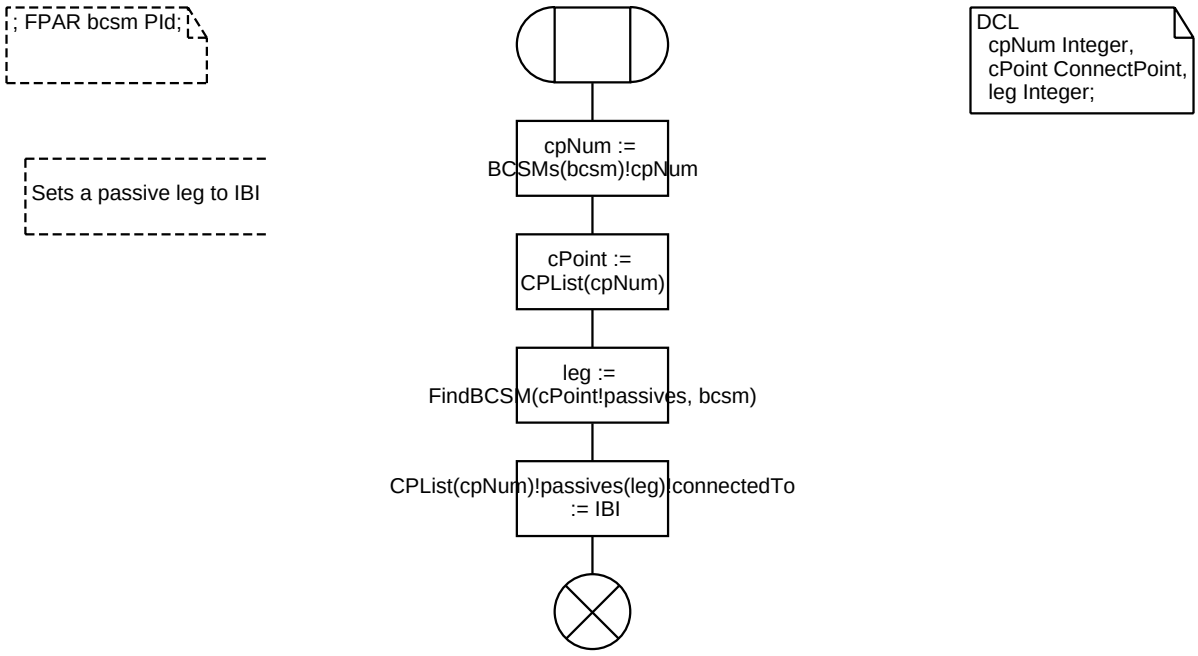
Procedure SetBCSMState 1(1)



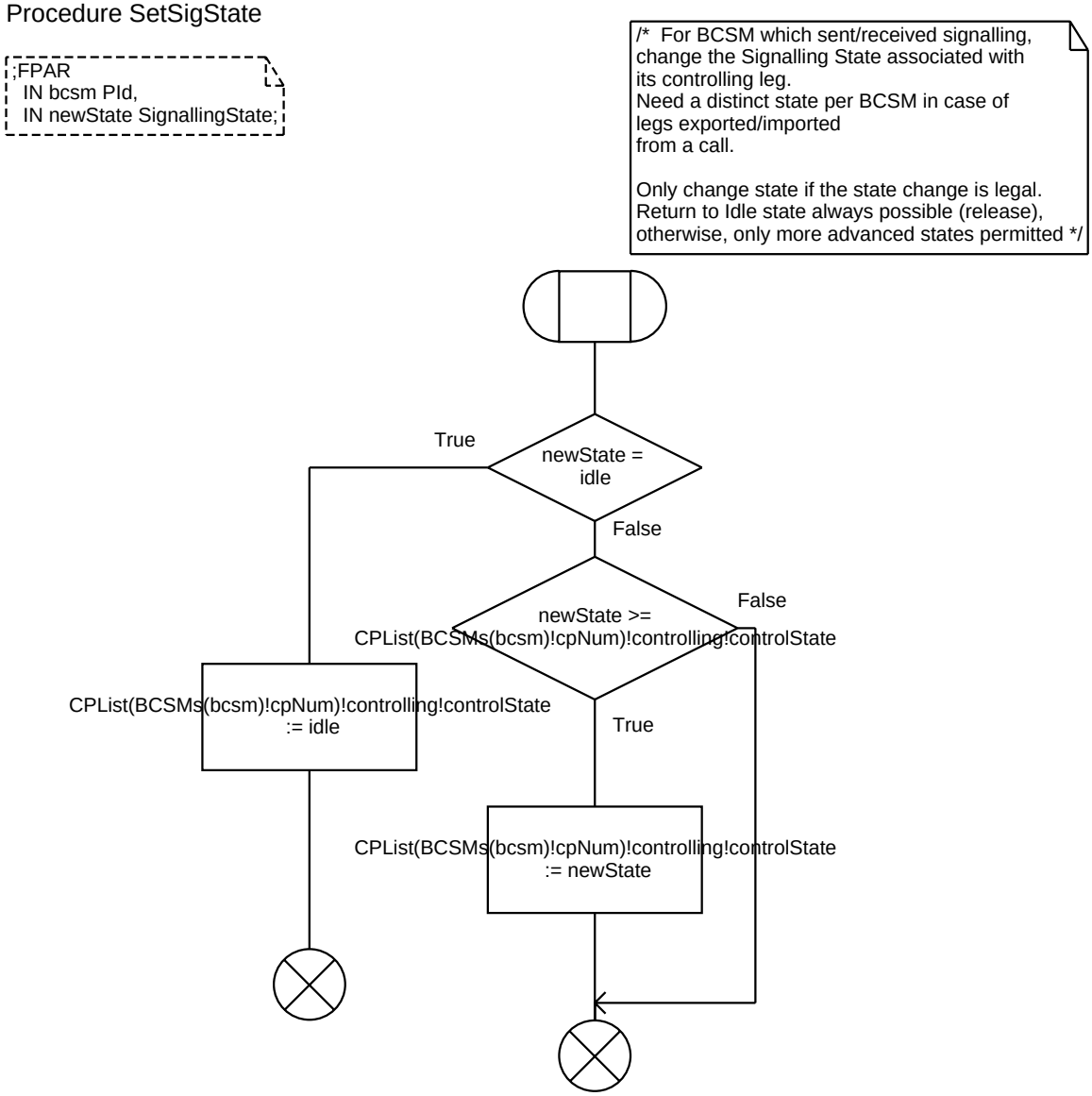
Procedure SetCallProgressState 1(1)



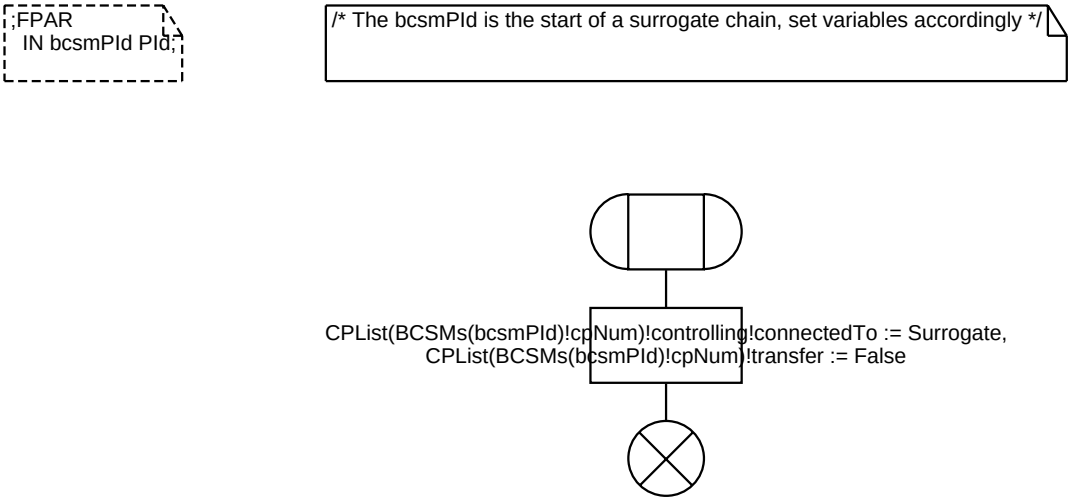
Procedure SetPassiveToIBI 1(1)



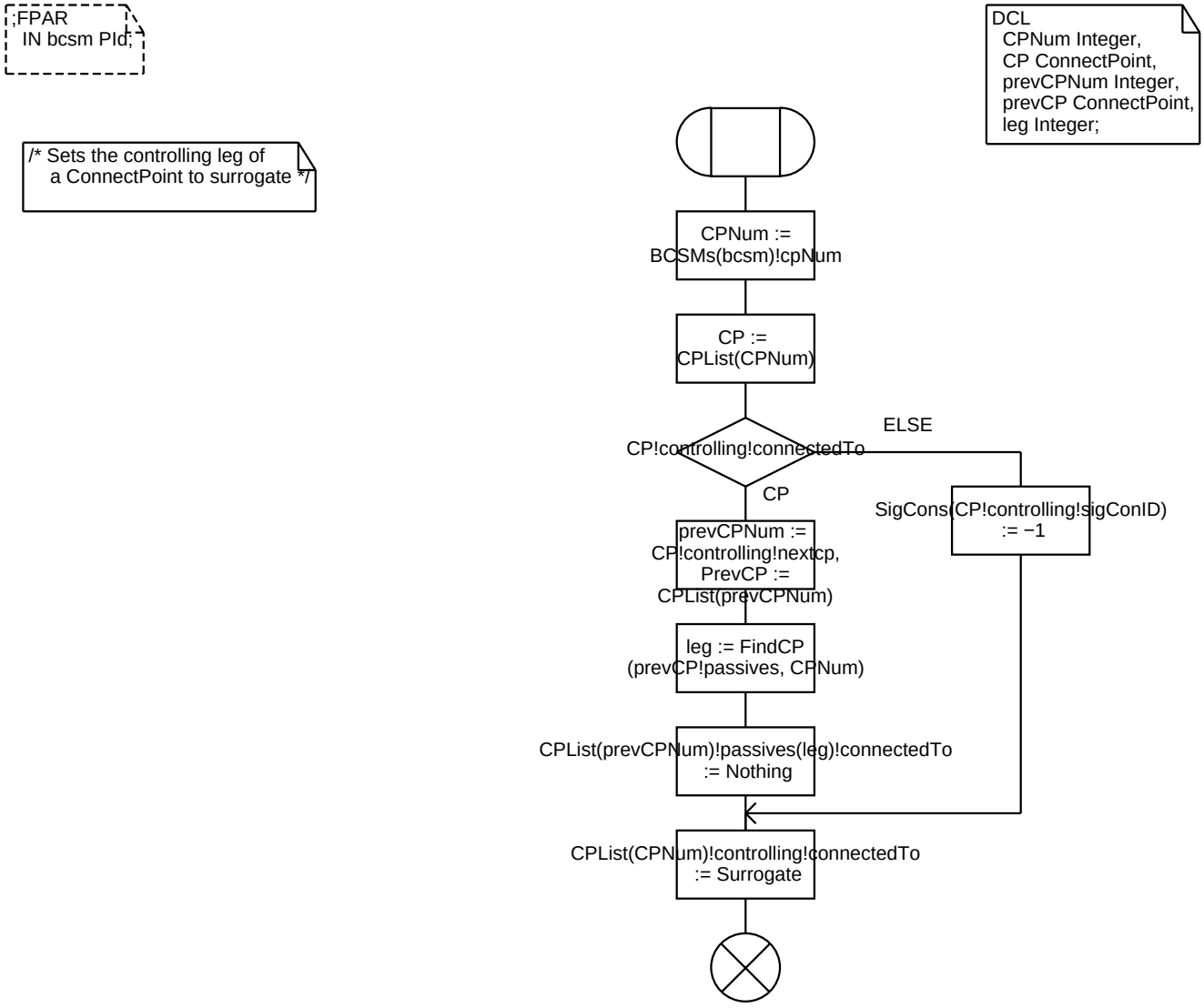
Procedure SetSigState 1(1)



Procedure SetSurrogateChainStart 1(1)



Procedure SetSurrogate 1(1)



Process Type OriginatingBCSM 1(19)

```
;FPAR  bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
```

```
/***** O−BCSM FOR INAP CS−3. *****/
```

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

```
NEWTYPE DPResultType
  LITERALS
    Resume,
    CollectInformation,
    AnalyseInformation,
    SelectRoute,
    Exception,
    ARelease,
    BRelease,
    SCFRelease, /* Used with ReleaseCall */
    Answer,
    Alerting,
    Reanswer,
    MidCall,
    DL_A, DL_B, /* Used with DisconnectLeg */
    CS_Stop,
    BCSM_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,
    CS_Stop,
    BCSM_Stop
ENDNEWTYPE;
```

BCP (O_BCSM_Out) (O_BCSM_In)

FA (BCSM_To_FA) (FA_To_BCSM)

Process Type OriginatingBCSM 2(19)

```
;FPAR  bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
```

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

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

DCL
  BCP 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 := noIndication,
  PIC  PICArg,
  DP DPArg,
  sendRelease Boolean,

  BufferedDigits Octet_String := EmptyString,

  SCFNoAnswerTimeSet Boolean := false,
  SCFNoAnswerTime Duration,

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

  digits Octet_String;
```

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

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


DCL
  nsrArg NetworkSRType,
  udArg UserDataType,
  dpUTSIArg DPUTSIArg;
```

Process Type OriginatingBCSM 3(19)

;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */

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

PIC_O_Null

PIC_Analyse_ _Information

PIC_Send_ _Call

PIC_O_ _Active

PIC_Authorise_ _Origination_ _Attempt

PIC_Select_ _Route

PIC_O_ _Alerting

PIC_O_ _Abandon

PIC_Collect_ _Information

PIC_Authorise_ _Call_Setup

PIC_O_Answer

PIC_ _OException

PIC_Collect_ _NDigits

PIC_O_ _Suspended

PIC_O_ _Retention

PIC_ _Disconnect

PIC_ _SCFRelease

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

DP_origAttempt_ Authorised

DP_Route_ _Select_Failure

DP_oCalled_ PartyBusy

DP_ _oDisconnect

DP_Collected_ _Info

DP_oAnswer

DP_oMidCall

DP_Analysed_ _Information

DP_oNoAnswer

DP_oAbandon

Process Type OriginatingBCSM 4(19)

;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */

DP_orig_ Attempt

DP_oSuspend

DP_oReanswer

DP_ _oTermSeized

DP_Authorize_ _Route_Failure

DP_origDenied

/* Procedures for mapping of parameters */

MapTo_ SIRArg

MapToDP

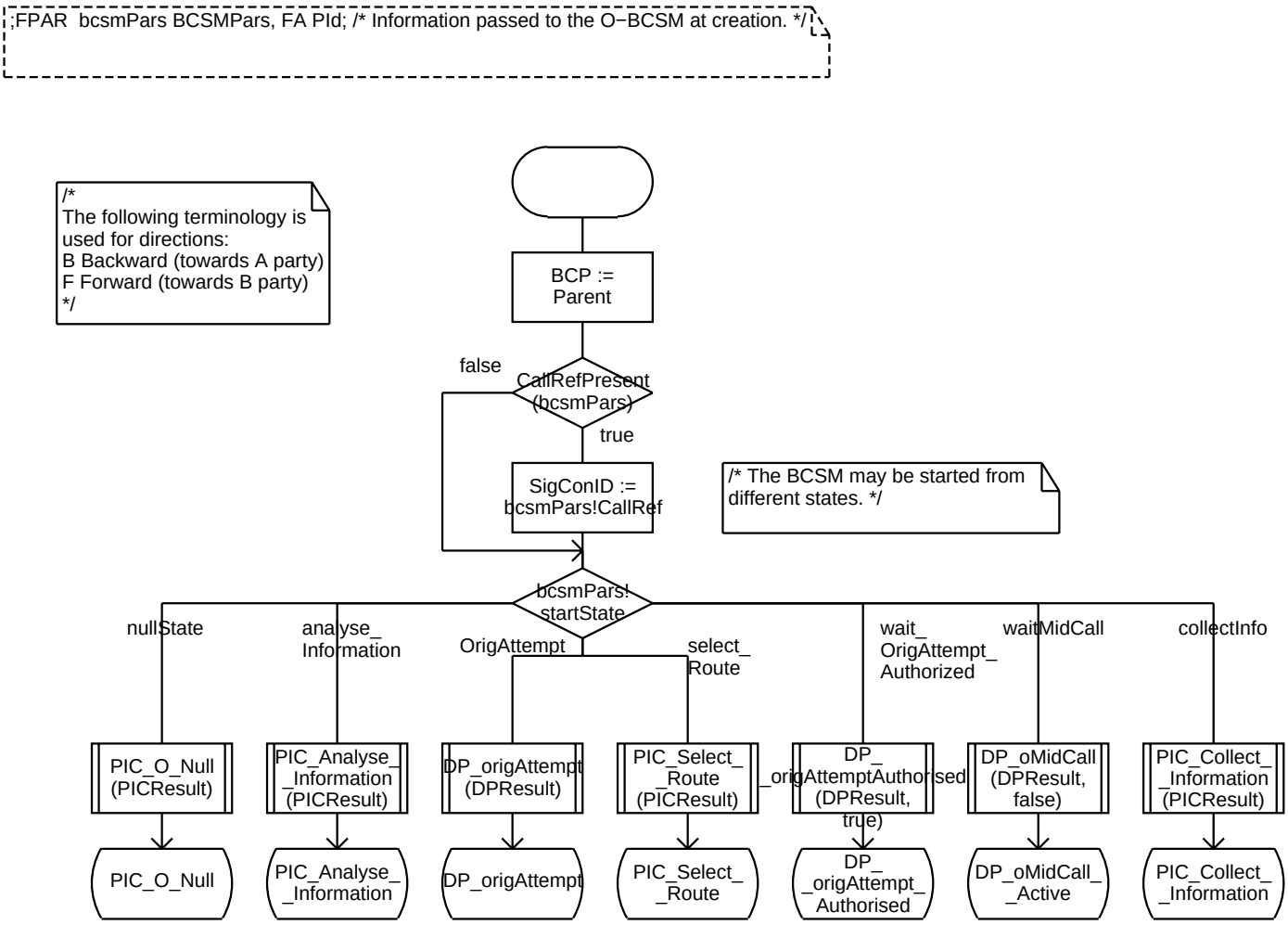
MapFromPIC

AnalyzeDigits

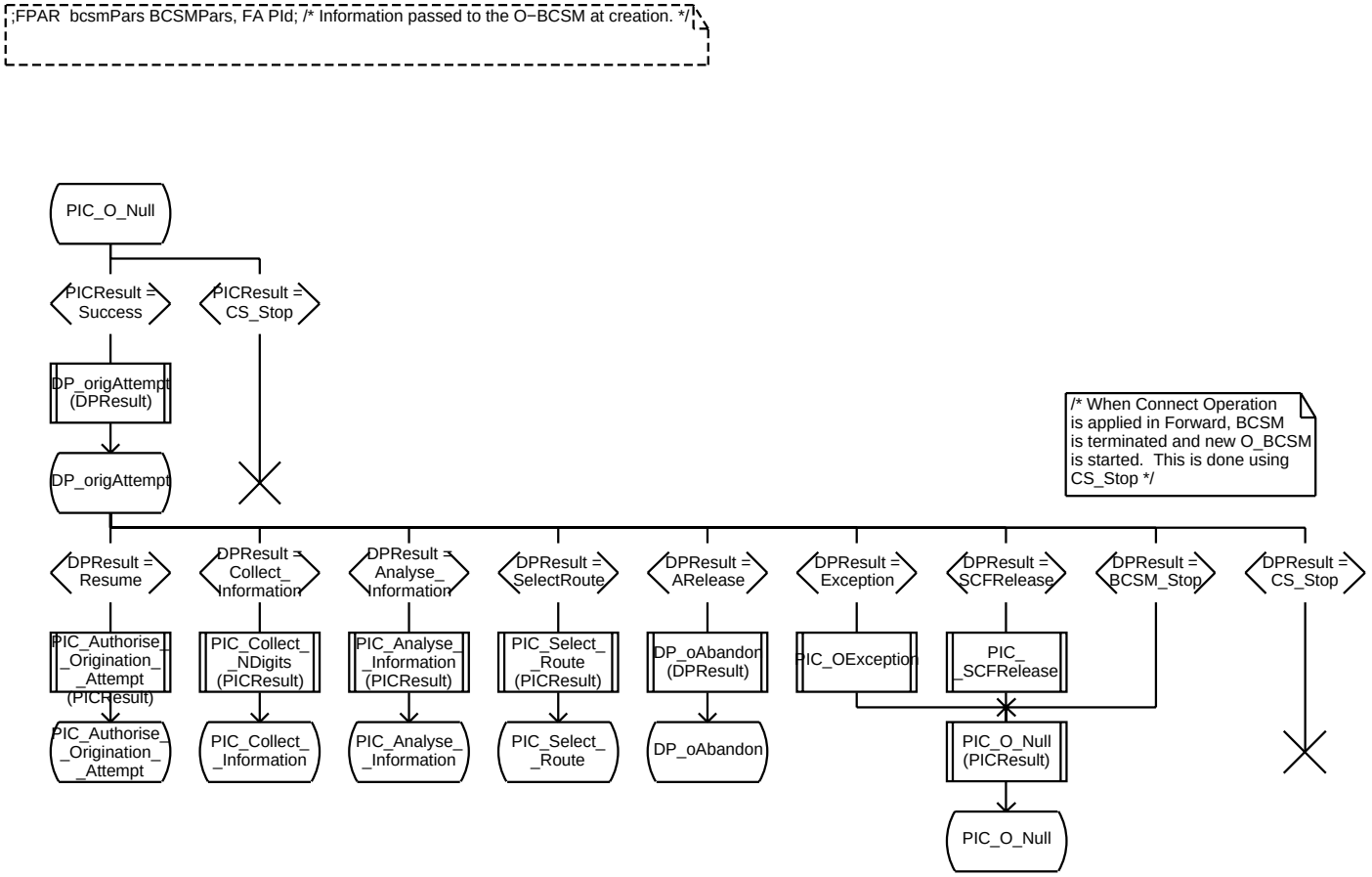
EXPORTED SetNoAnswerTime

EXPORTED SetFIMAgent

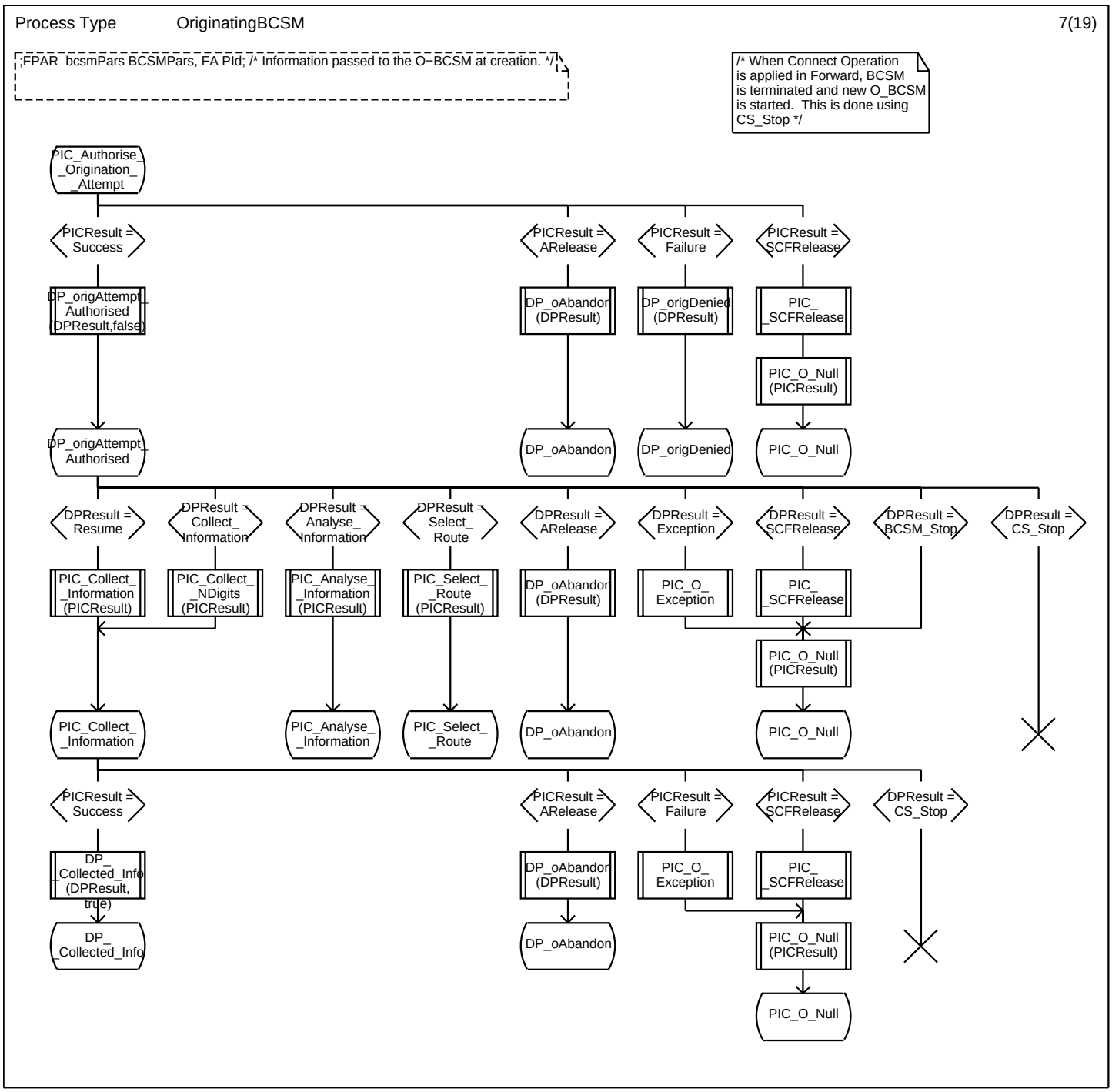
Process Type OriginatingBCSM 5(19)

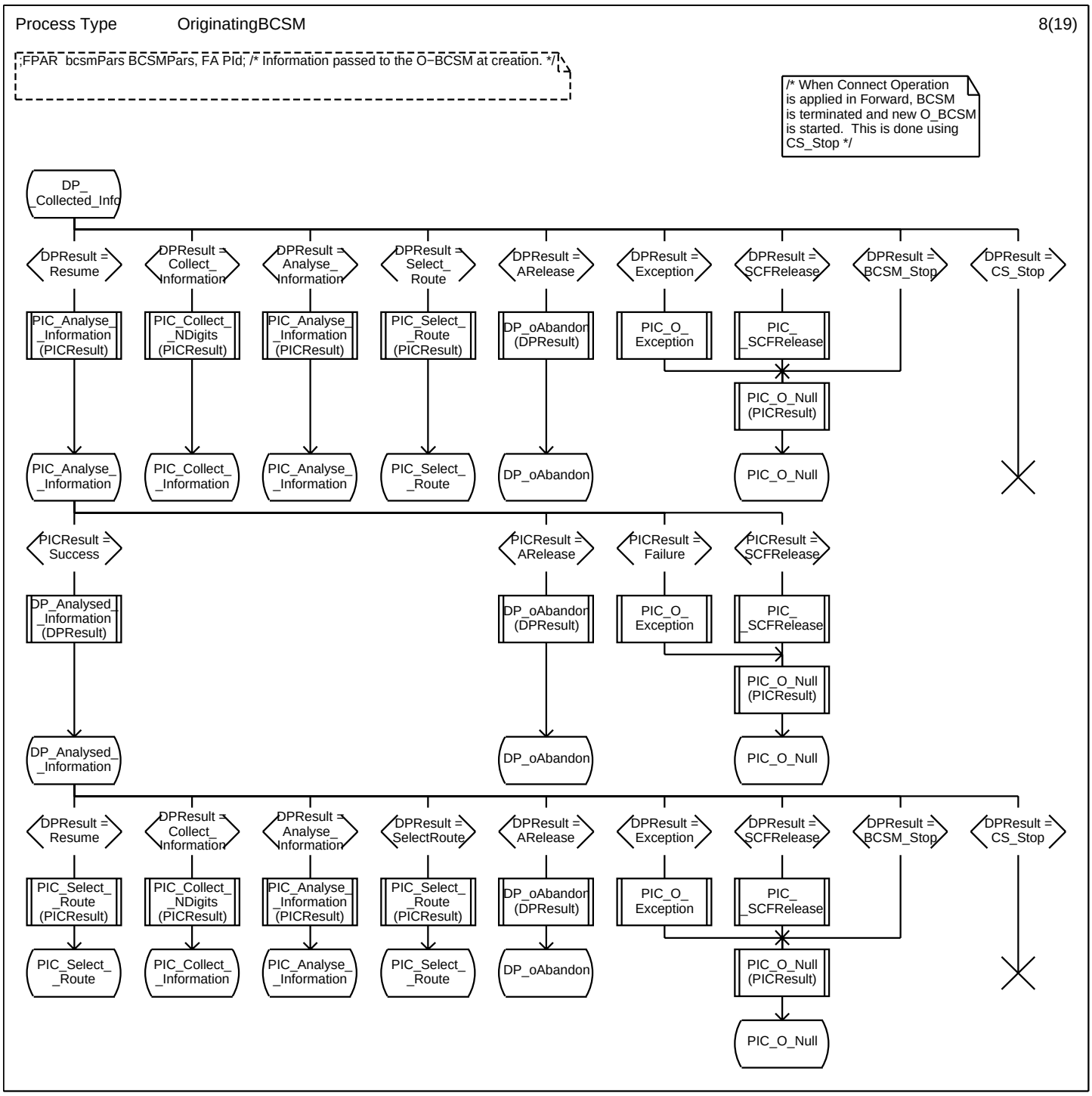


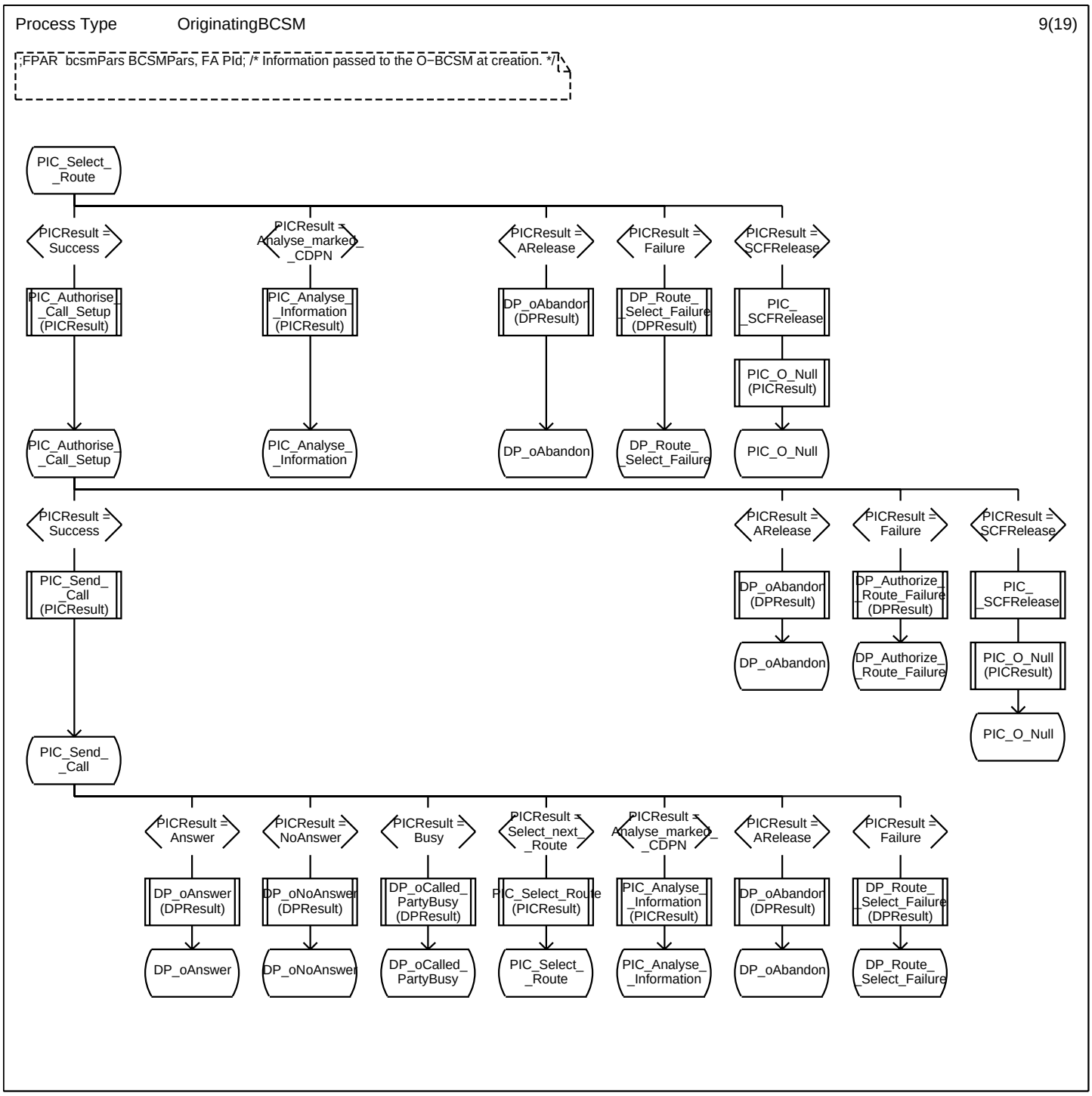
Process Type OriginatingBCSM 6(19)



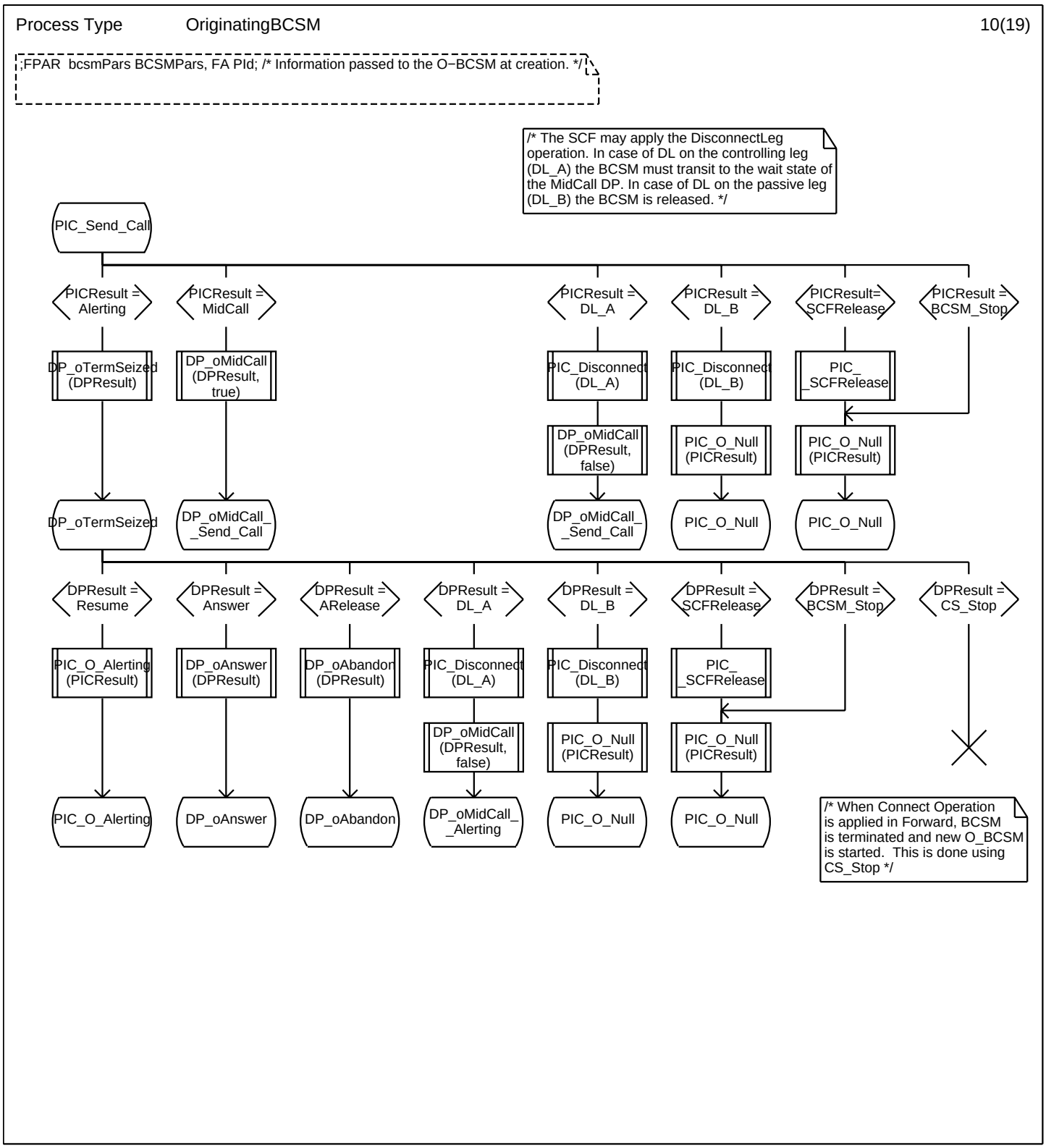
Process Type OriginatingBCSM 7(19)

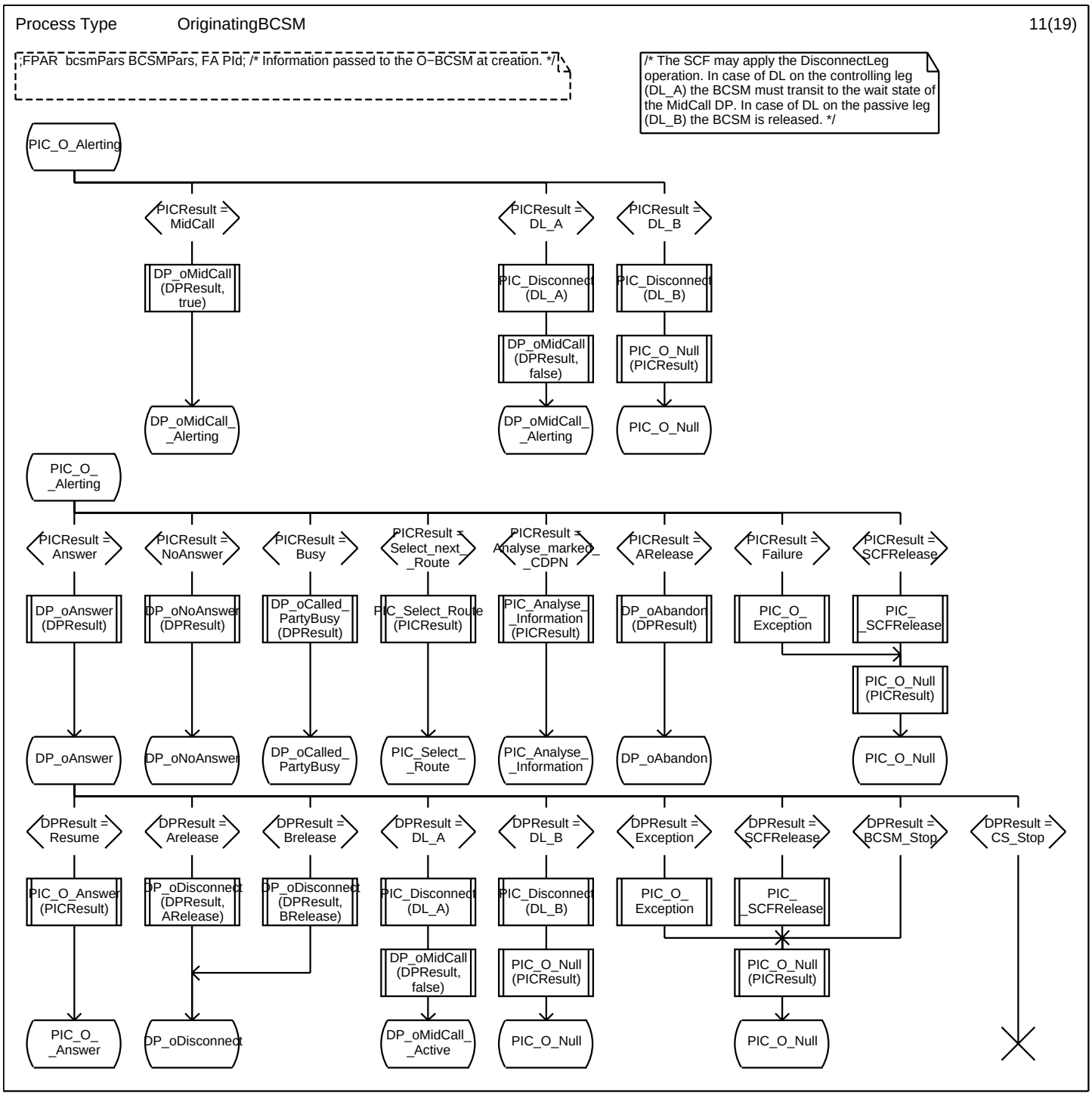



Process Type OriginatingBCSM 8(19)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
DP_ _Collected_Info
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_Analyse_ _Information (PICResult)
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_ _SCFRelease
PIC_O_Null (PICResult)
PIC_Analyse_ _Information
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
PIC_O_Null
PICResult = Success
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
DP_Analysed_ _Information (DPResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_ _SCFRelease
PIC_O_Null (PICResult)
DP_Analysed_ _Information
DP_oAbandon
PIC_O_Null
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = SelectRoute
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_Select_ _Route (PICResult)
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_ _SCFRelease
PIC_Select_ _Route
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
PIC_O_Null (PICResult)
PIC_O_Null

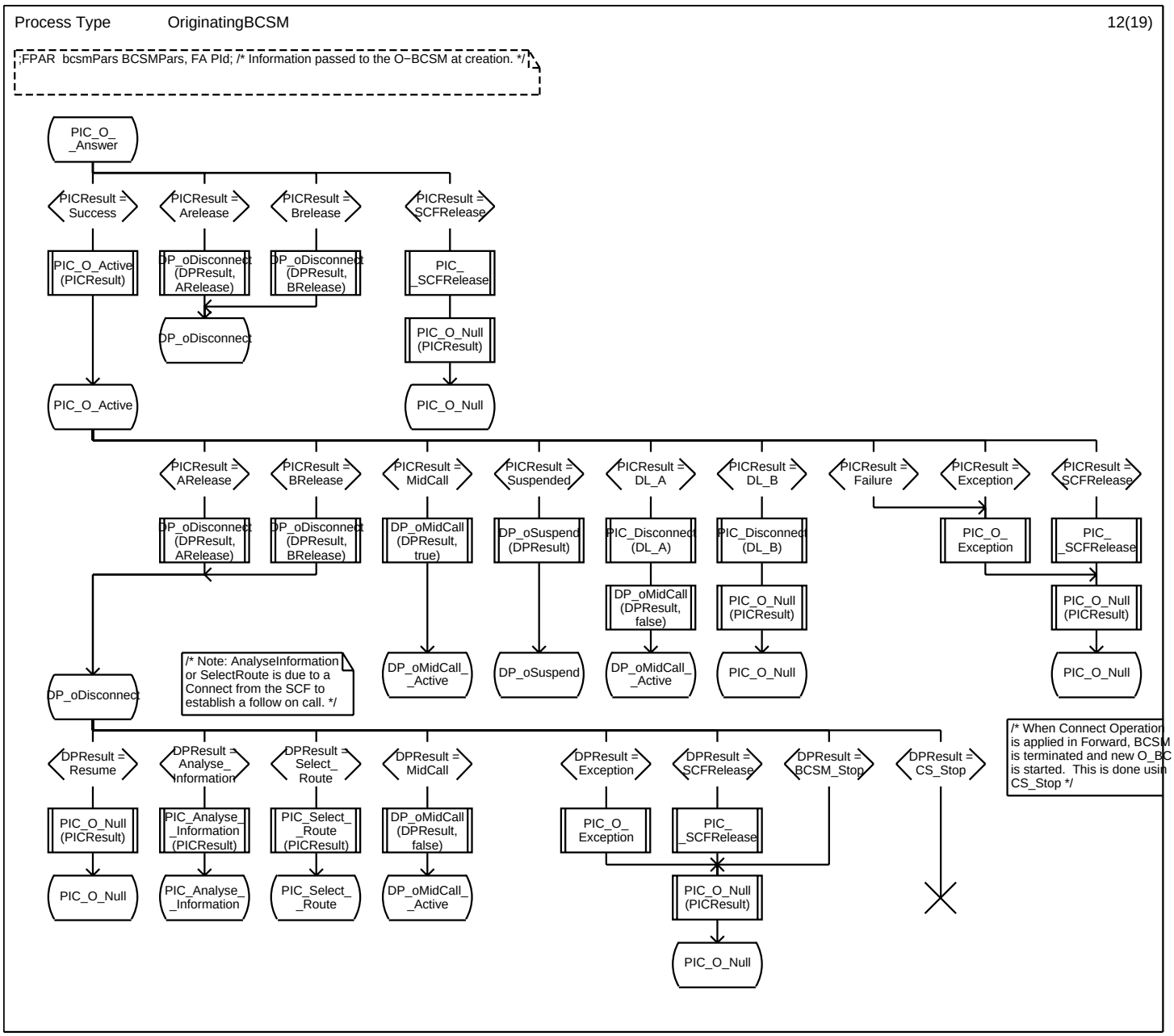

Process Type OriginatingBCSM 9(19)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
PIC_Select_ _Route
PICResult = Success
PICResult = Analyse_marked_ _CDPN
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
PIC_Authorise_ _Call_Setup (PICResult)
PIC_Analyse_ _Information (PICResult)
DP_oAbandon (DPResult)
DP_Route_ _Select_Failure (DPResult)
PIC_ _SCFRelease
PIC_O_Null (PICResult)
PIC_Authorise_ _Call_Setup
PIC_Analyse_ _Information
DP_oAbandon
DP_Route_ _Select_Failure
PIC_O_Null
PICResult = Success
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
PIC_Send_ _Call (PICResult)
DP_oAbandon (DPResult)
DP_Authorize_ _Route_Failure (DPResult)
PIC_ _SCFRelease
DP_oAbandon
DP_Authorize_ _Route_Failure
PIC_O_Null (PICResult)
PIC_O_Null
PIC_Send_ _Call
PICResult = Answer
PICResult = NoAnswer
PICResult = Busy
PICResult = Select_next_ _Route
PICResult = Analyse_marked_ _CDPN
PICResult = ARelease
PICResult = Failure
DP_oAnswer (DPResult)
DP_oNoAnswer (DPResult)
DP_oCalled_ PartyBusy (DPResult)
PIC_Select_Route (PICResult)
PIC_Analyse_ _Information (PICResult)
DP_oAbandon (DPResult)
DP_Route_ _Select_Failure (DPResult)
DP_oAnswer
DP_oNoAnswer
DP_oCalled_ PartyBusy
PIC_Select_ _Route
PIC_Analyse_ _Information
DP_oAbandon
DP_Route_ _Select_Failure

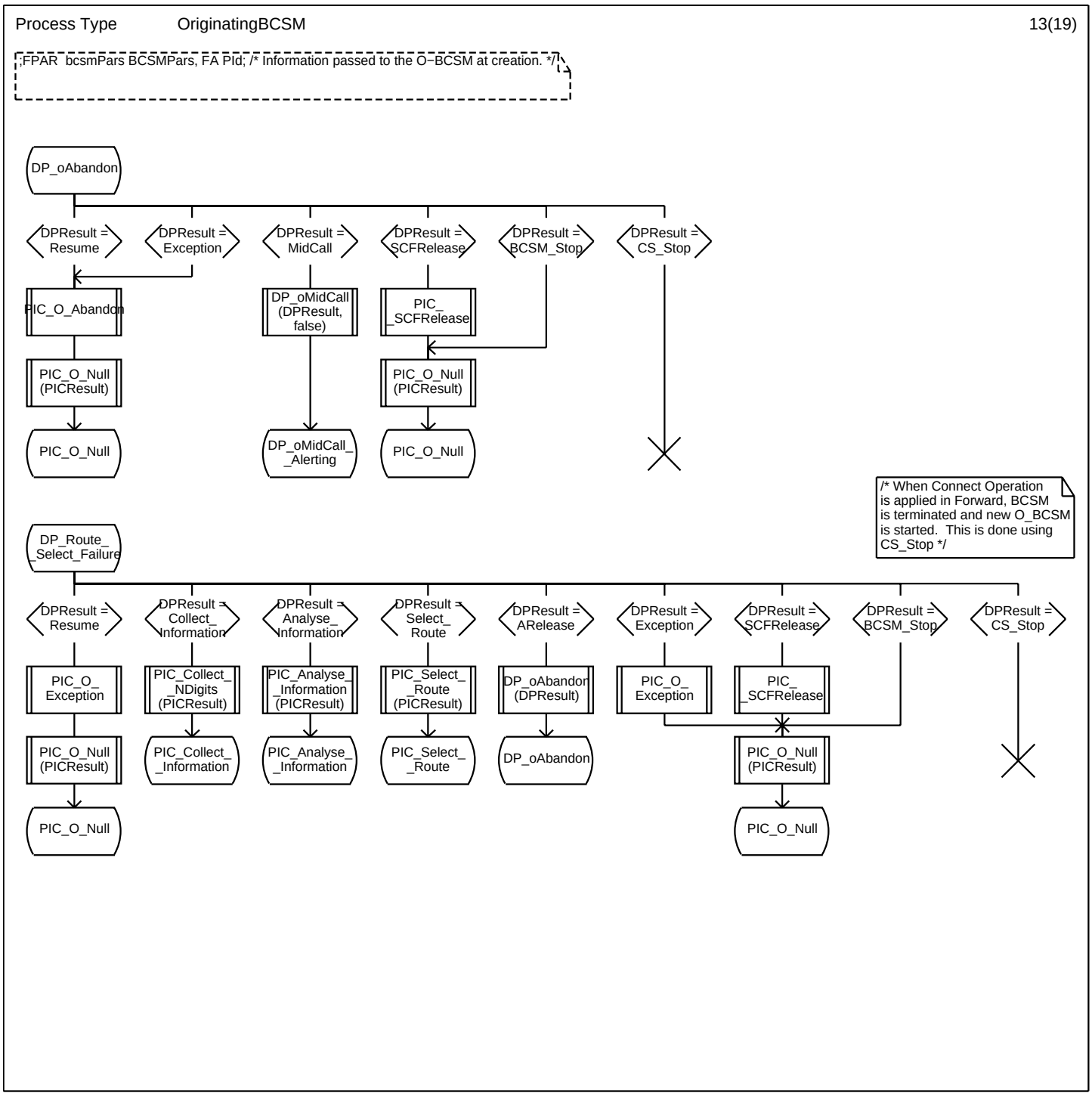
Process Type OriginatingBCSM 10(19)



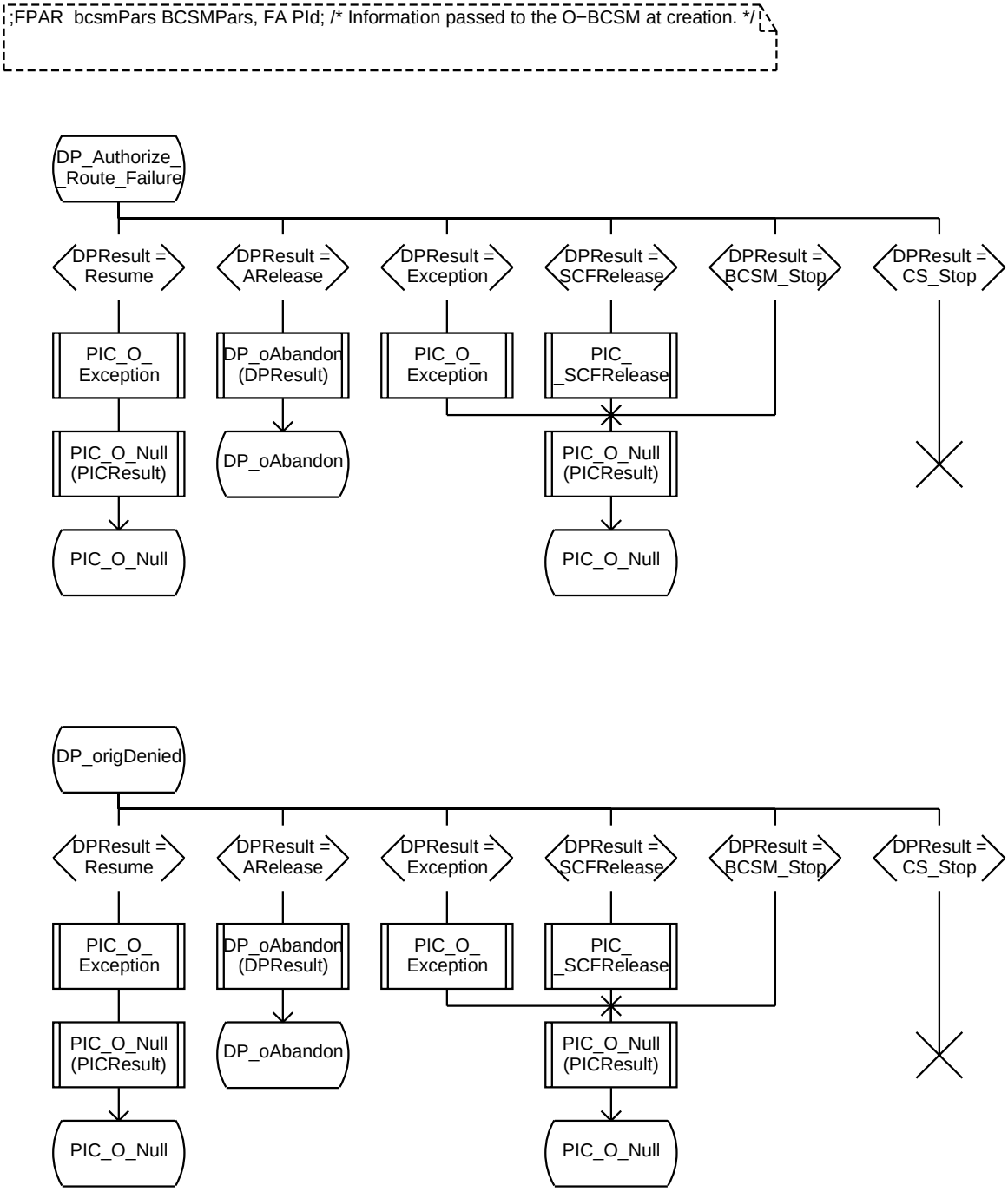

Process Type OriginatingBCSM 11(19)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
/* The SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
PIC_O_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)
PIC_O_Null (PICResult)
DP_oMidCall_ _Alerting
DP_oMidCall_ _Alerting
PIC_O_Null
PIC_O_ _Alerting
PICResult = Answer
PICResult = NoAnswer
PICResult = Busy
PICResult = Select_next_ _Route
PICResult = Analyse_marked_ _CDPN
PICResult = ARelease
PICResult = Failure
PICResult = SCFRelease
DP_oAnswer (DPResult)
DP_oNoAnswer (DPResult)
DP_oCalled_ PartyBusy (DPResult)
PIC_Select_Route (PICResult)
PIC_Analyse_ _Information (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_ _SCFRelease
PIC_O_Null (PICResult)
DP_oAnswer
DP_oNoAnswer
DP_oCalled_ PartyBusy
PIC_Select_ _Route
PIC_Analyse_ _Information
DP_oAbandon
PIC_O_Null
DPResult = Resume
DPResult = Arelease
DPResult = Brelease
DPResult = DL_A
DPResult = DL_B
DPResult = Exception
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_O_Answer (PICResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_O_ Exception
PIC_ _SCFRelease
DP_oMidCall (DPResult, false)
PIC_O_Null (PICResult)
PIC_O_Null (PICResult)
PIC_O_ _Answer
DP_oDisconnect
DP_oMidCall_ _Active
PIC_O_Null
PIC_O_Null

Process Type OriginatingBCSM 12(19)

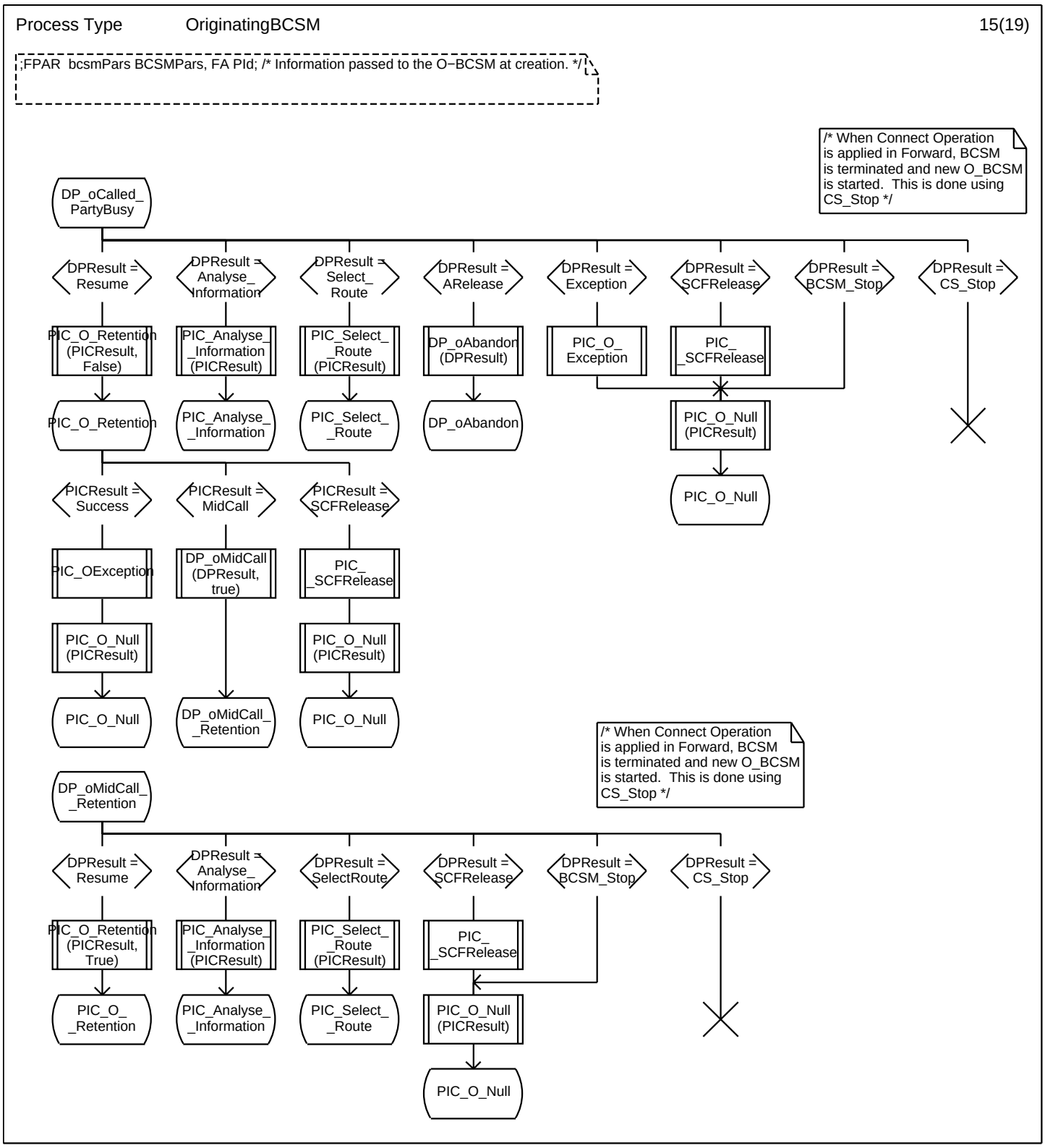


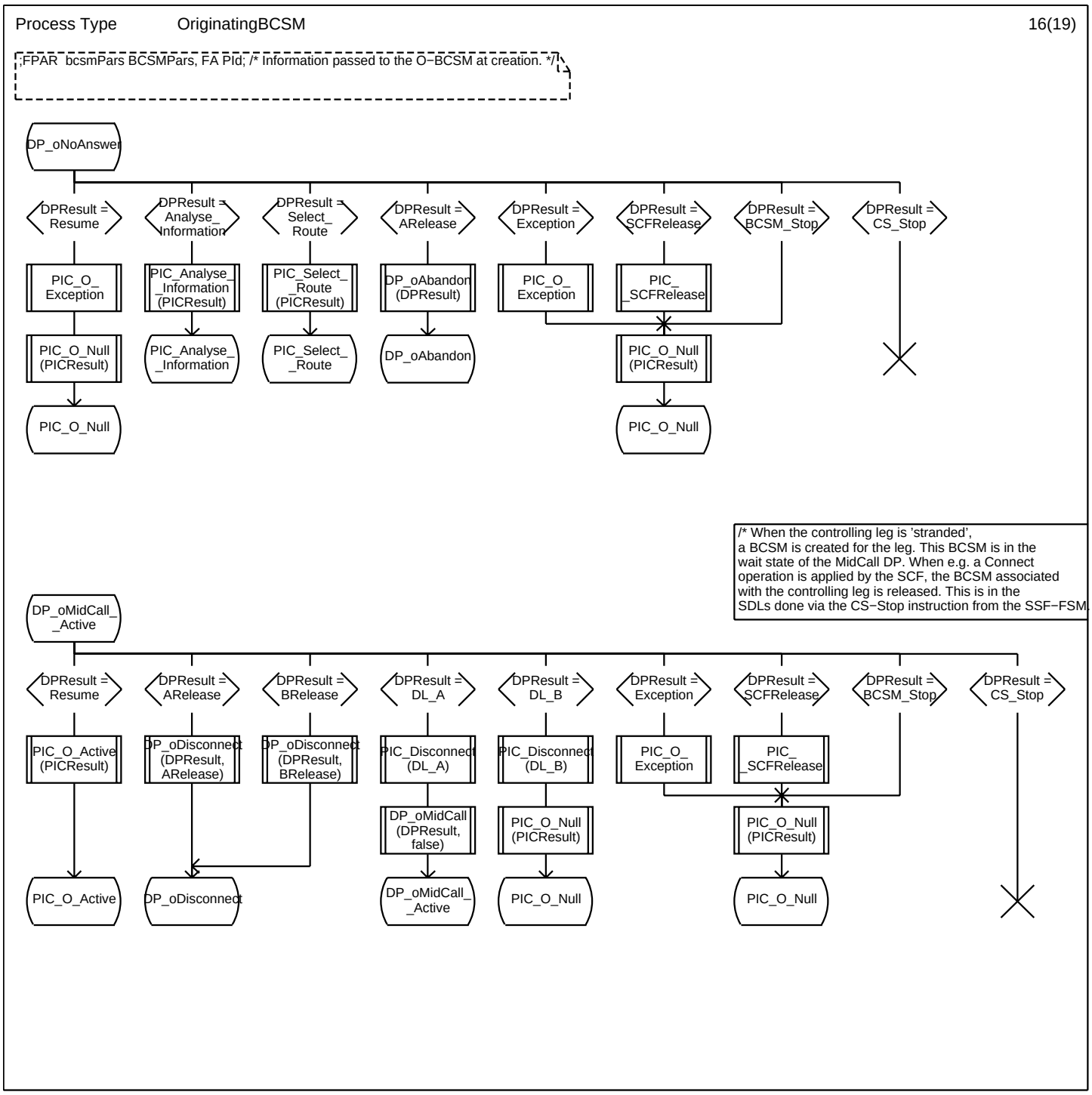

Process Type OriginatingBCSM 13(19)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
DP_oAbandon
DPResult = Resume
DPResult = Exception
DPResult = MidCall
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_O_Abandon
DP_oMidCall (DPResult, false)
PIC_ _SCFRelease
PIC_O_Null (PICResult)
PIC_O_Null (PICResult)
PIC_O_Null
DP_oMidCall_ _Alerting
PIC_O_Null
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
DP_Route_ _Select_Failure
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_O_ Exception
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_ _SCFRelease
PIC_O_Null (PICResult)
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
PIC_O_Null (PICResult)
PIC_O_Null
PIC_O_Null

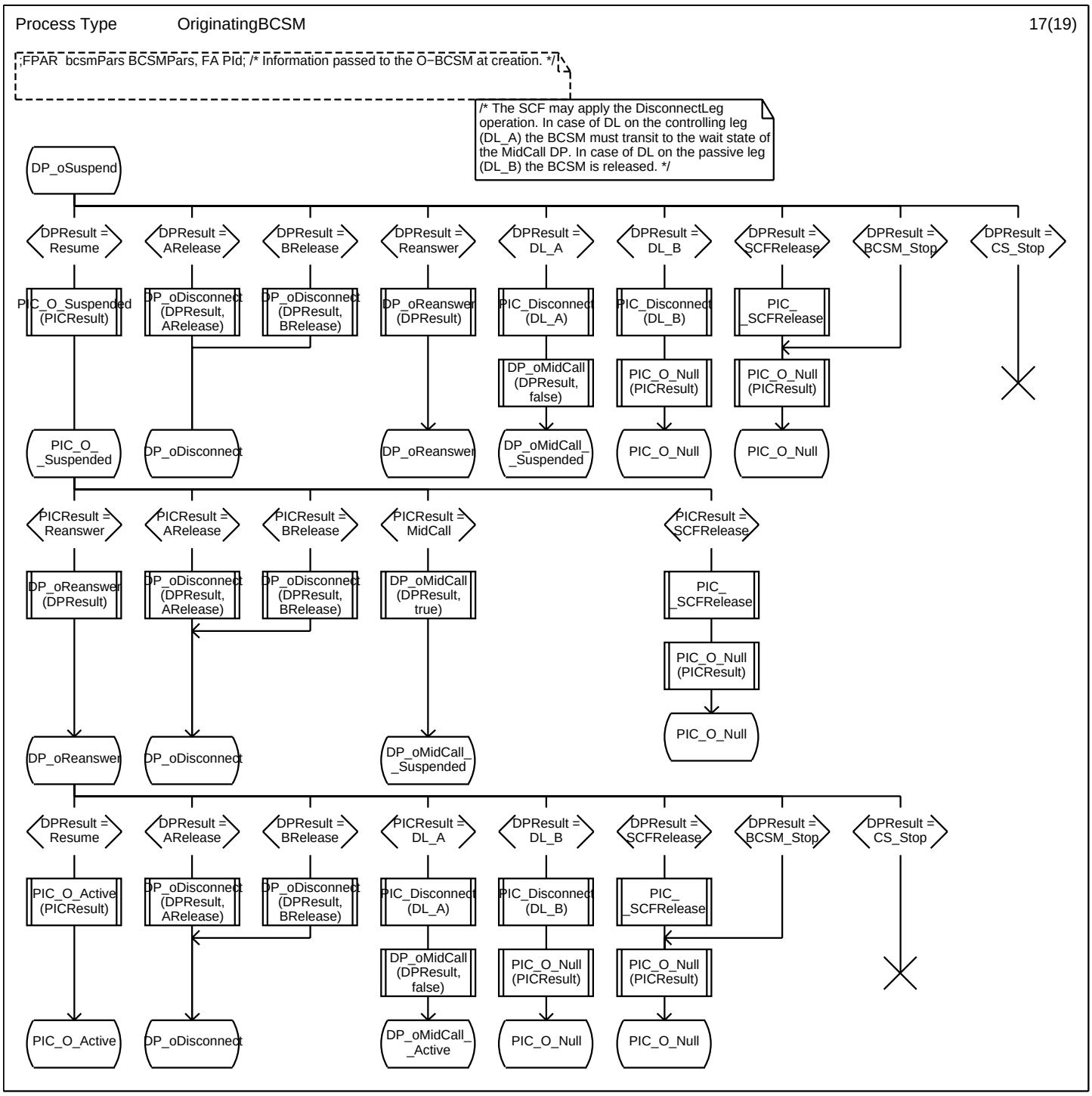
Process Type OriginatingBCSM 14(19)



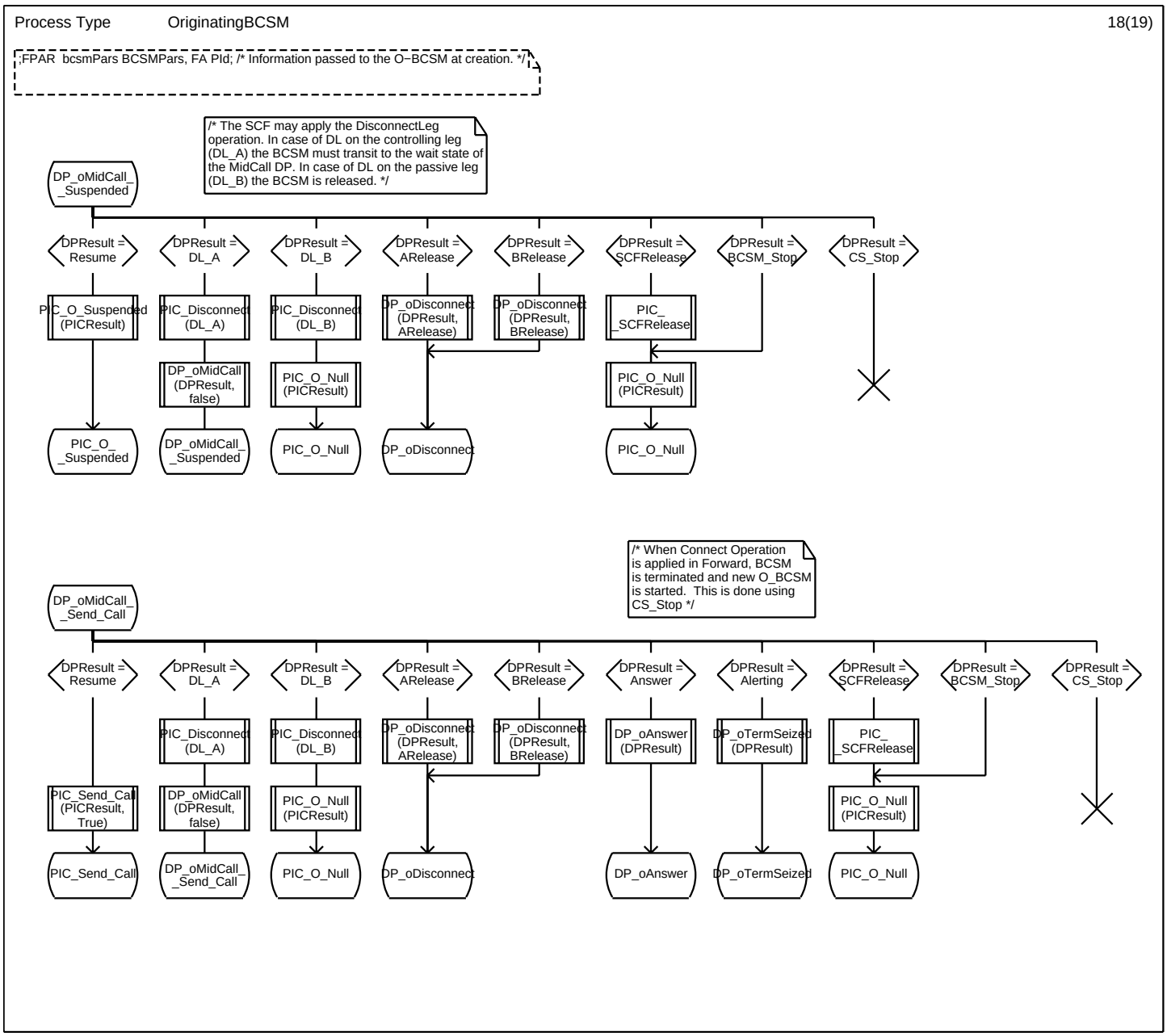
Process Type OriginatingBCSM 15(19)




Process Type OriginatingBCSM 16(19)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
DP_oNoAnswer
DPResult = Resume
DPResult = Analyse_ Information
DPResult = Select_ Route
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_O_ Exception
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_O_ Exception
PIC_ _SCFRelease
PIC_O_Null (PICResult)
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
PIC_O_Null (PICResult)
PIC_O_Null
PIC_O_Null
/* 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−FSM.
DP_oMidCall_ _Active
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = DL_A
DPResult = DL_B
DPResult = Exception
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_O_Active (PICResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_O_ Exception
PIC_ _SCFRelease
DP_oMidCall (DPResult, false)
PIC_O_Null (PICResult)
PIC_O_Null (PICResult)
PIC_O_Active
DP_oDisconnect
DP_oMidCall_ _Active
PIC_O_Null
PIC_O_Null

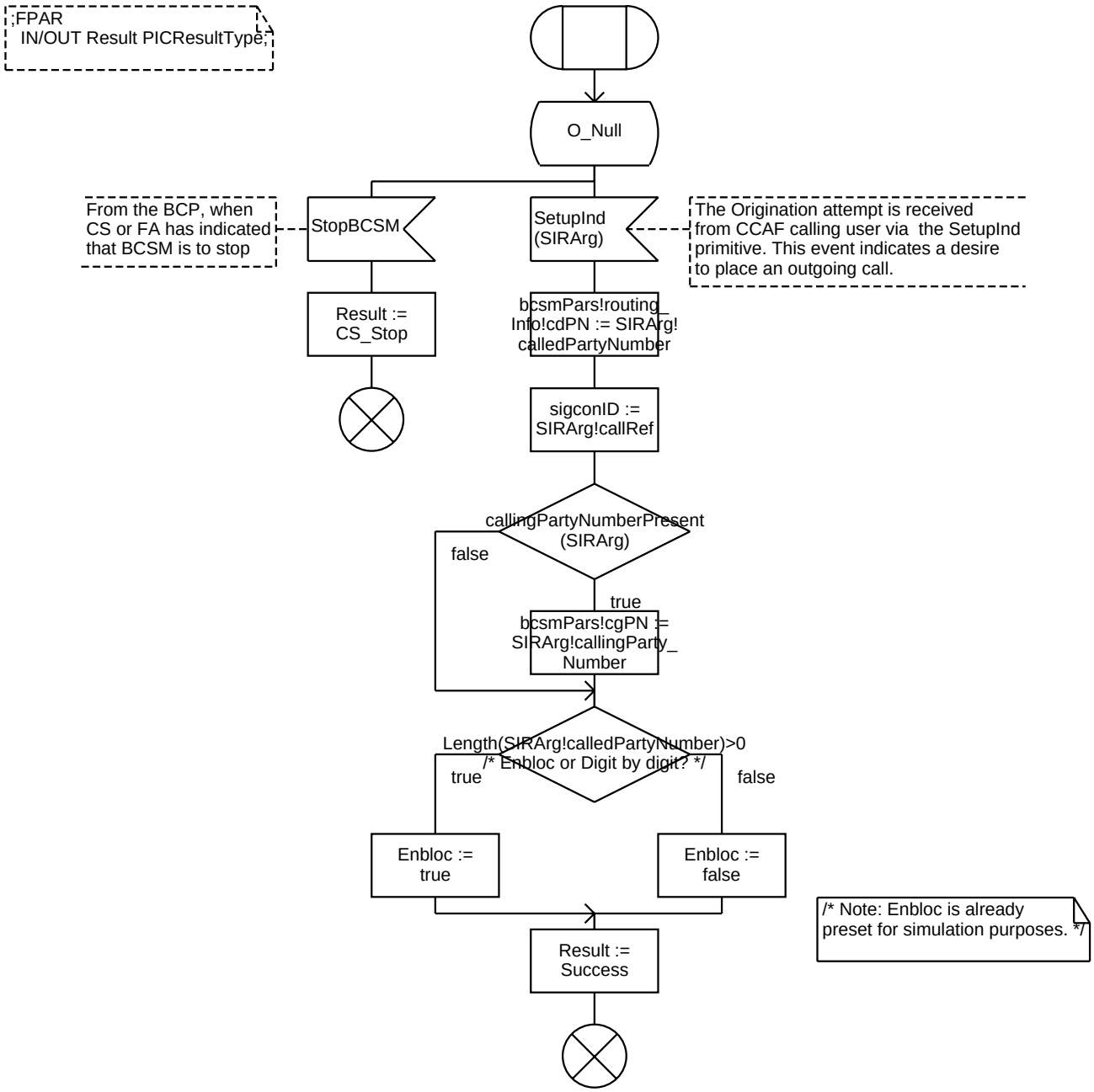

Process Type OriginatingBCSM 17(19)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the O−BCSM at creation. */
/* The SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oSuspend
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = Reanswer
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_O_Suspended (PICResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
DP_oReanswer (DPResult)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_ _SCFRelease
DP_oMidCall (DPResult, false)
PIC_O_Null (PICResult)
PIC_O_Null (PICResult)
PIC_O_ _Suspended
DP_oDisconnect
DP_oReanswer
DP_oMidCall_ _Suspended
PIC_O_Null
PIC_O_Null
PICResult = Reanswer
PICResult = ARelease
PICResult = BRelease
PICResult = MidCall
PICResult = SCFRelease
DP_oReanswer (DPResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
DP_oMidCall (DPResult, true)
PIC_ _SCFRelease
PIC_O_Null (PICResult)
PIC_O_Null
DP_oReanswer
DP_oDisconnect
DP_oMidCall_ _Suspended
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
PICResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_O_Active (PICResult)
DP_oDisconnect (DPResult, ARelease)
DP_oDisconnect (DPResult, BRelease)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_ _SCFRelease
DP_oMidCall (DPResult, false)
PIC_O_Null (PICResult)
PIC_O_Null (PICResult)
PIC_O_Active
DP_oDisconnect
DP_oMidCall_ _Active
PIC_O_Null
PIC_O_Null

Process Type OriginatingBCSM 18(19)

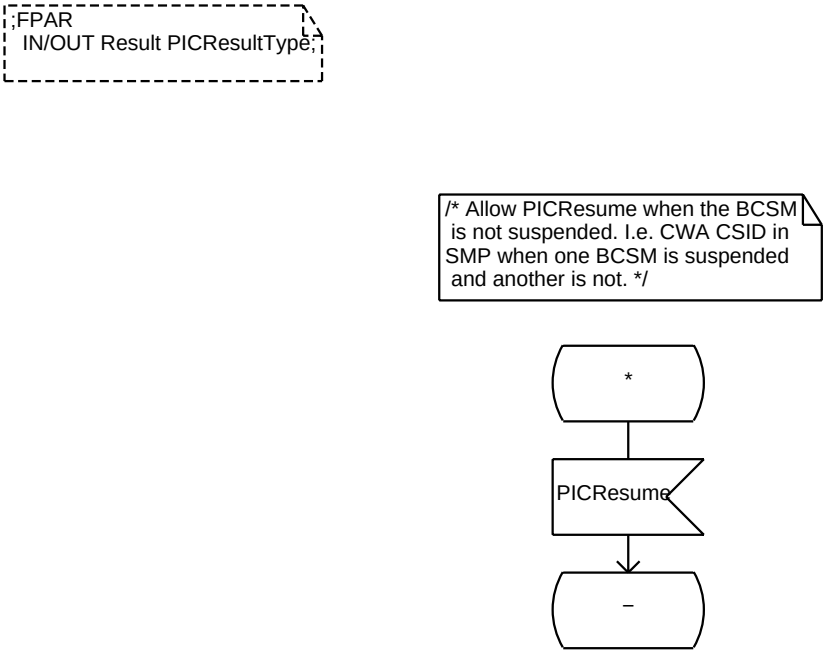


OriginatingBCSM

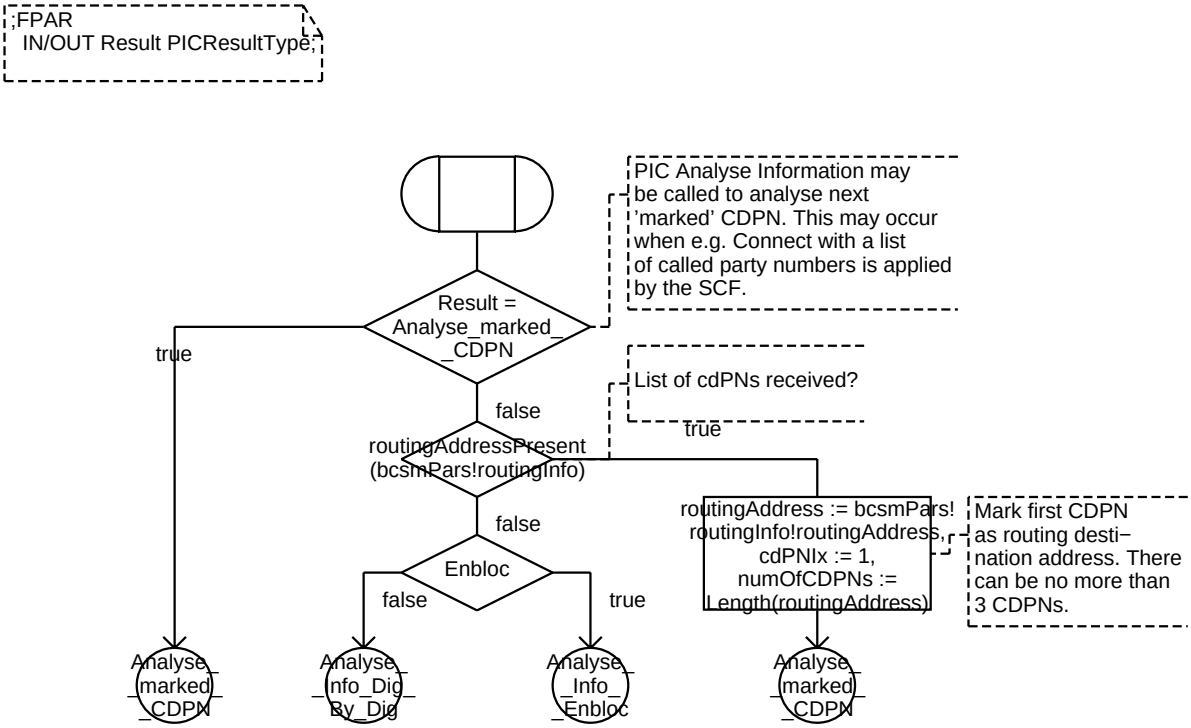
Procedure PIC_O_Null 1(2)



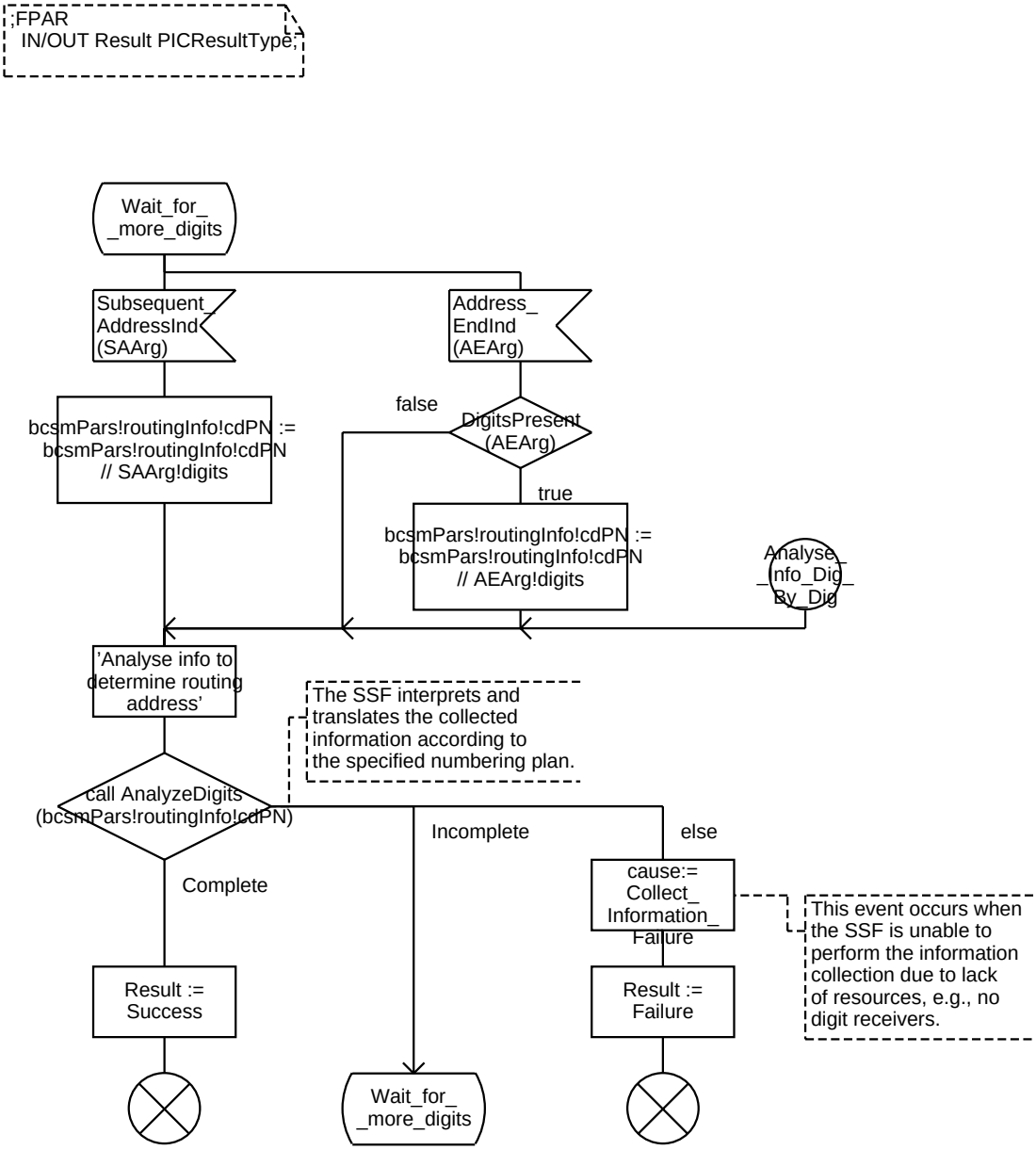
Procedure PIC_O_Null 2(2)



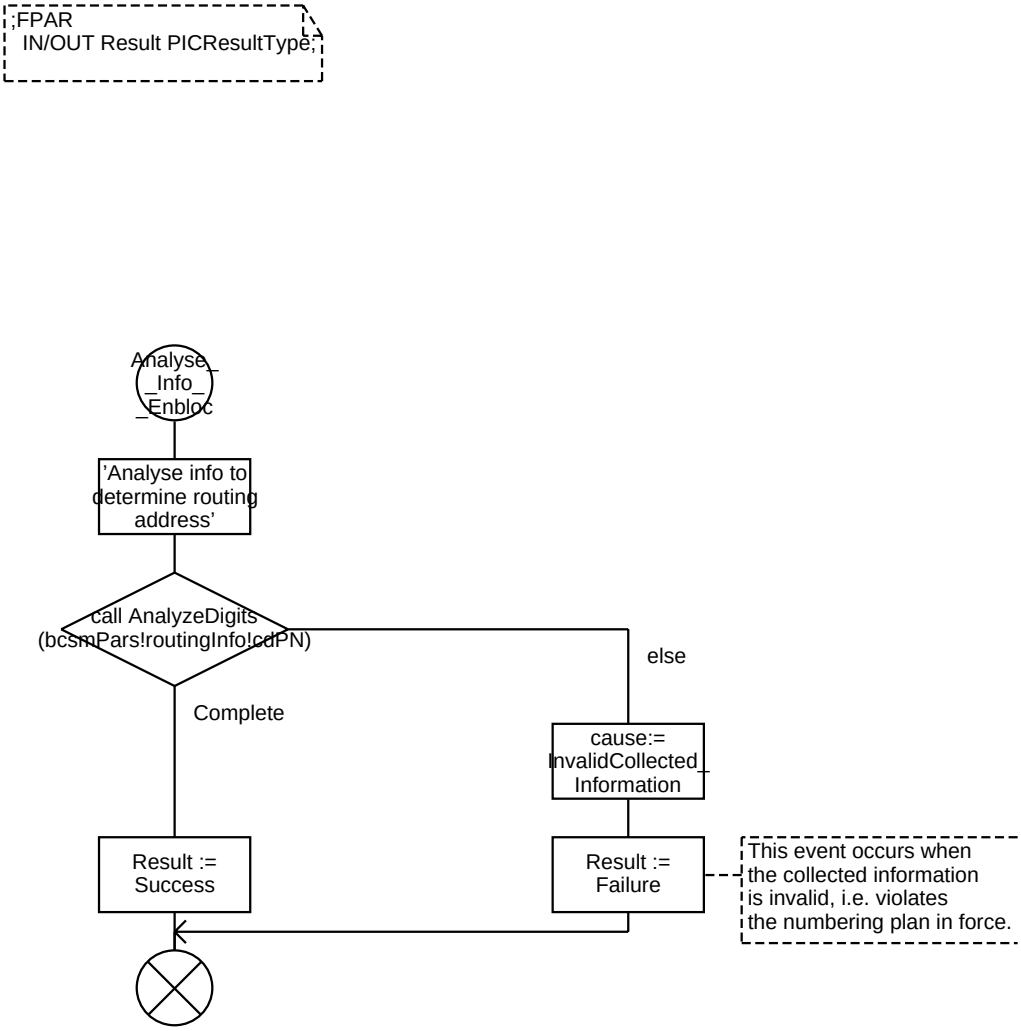
Procedure PIC_Analyse_Information 1(5)



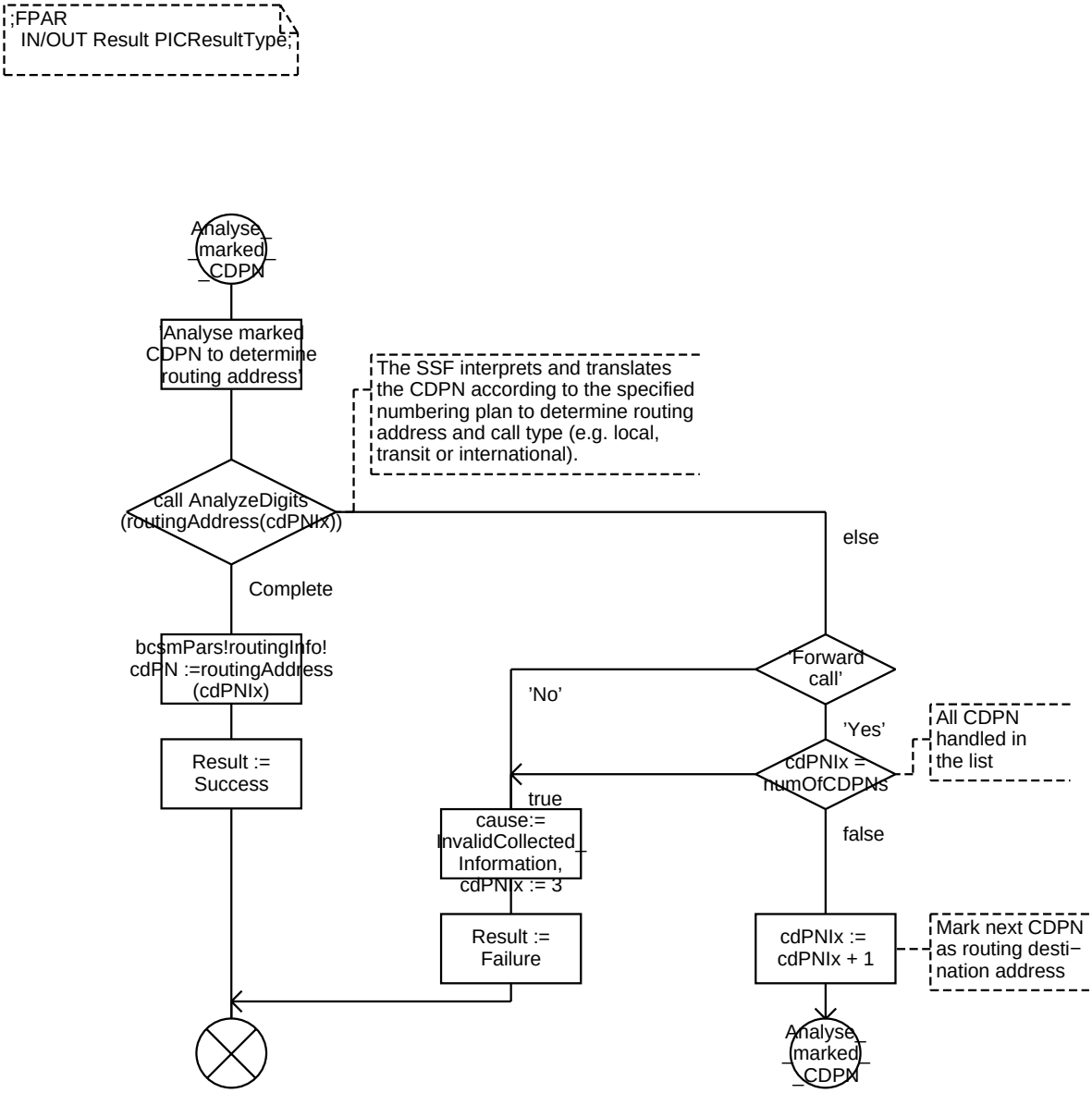
Procedure PIC_Analyse_Information 2(5)



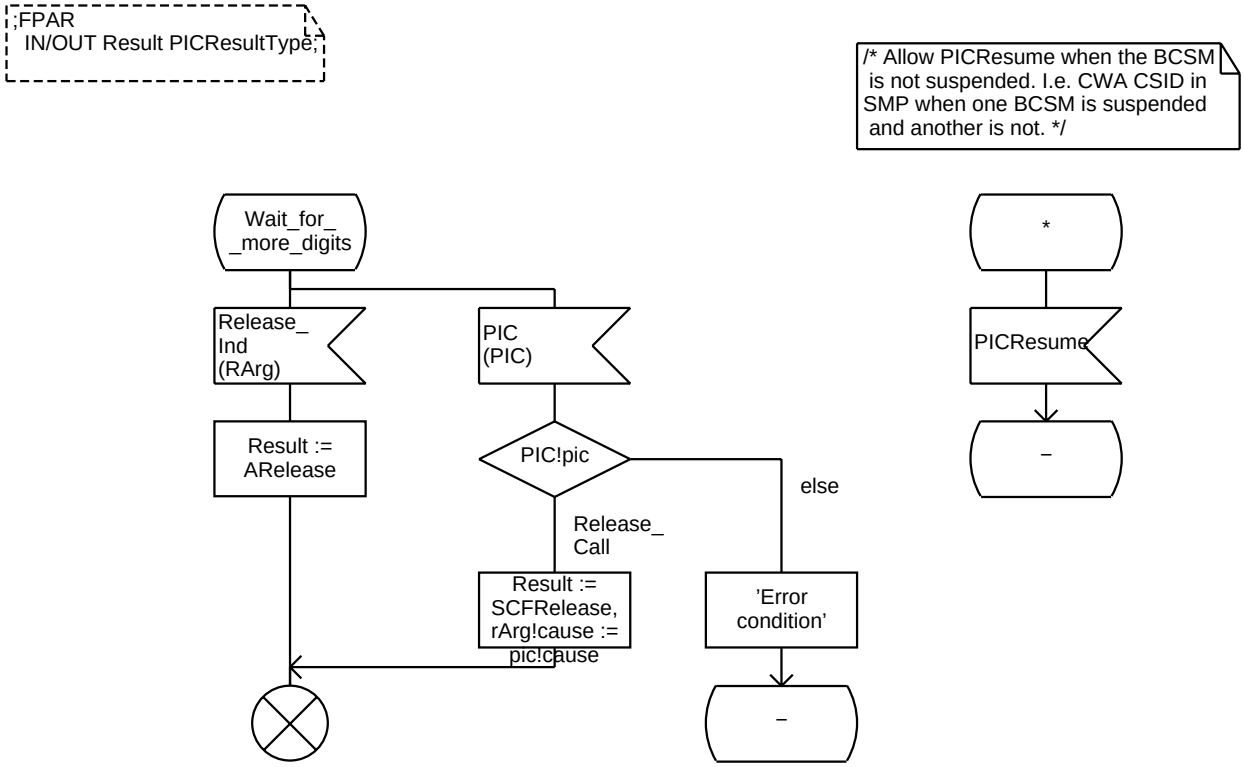
Procedure PIC_Analyse_Information 3(5)



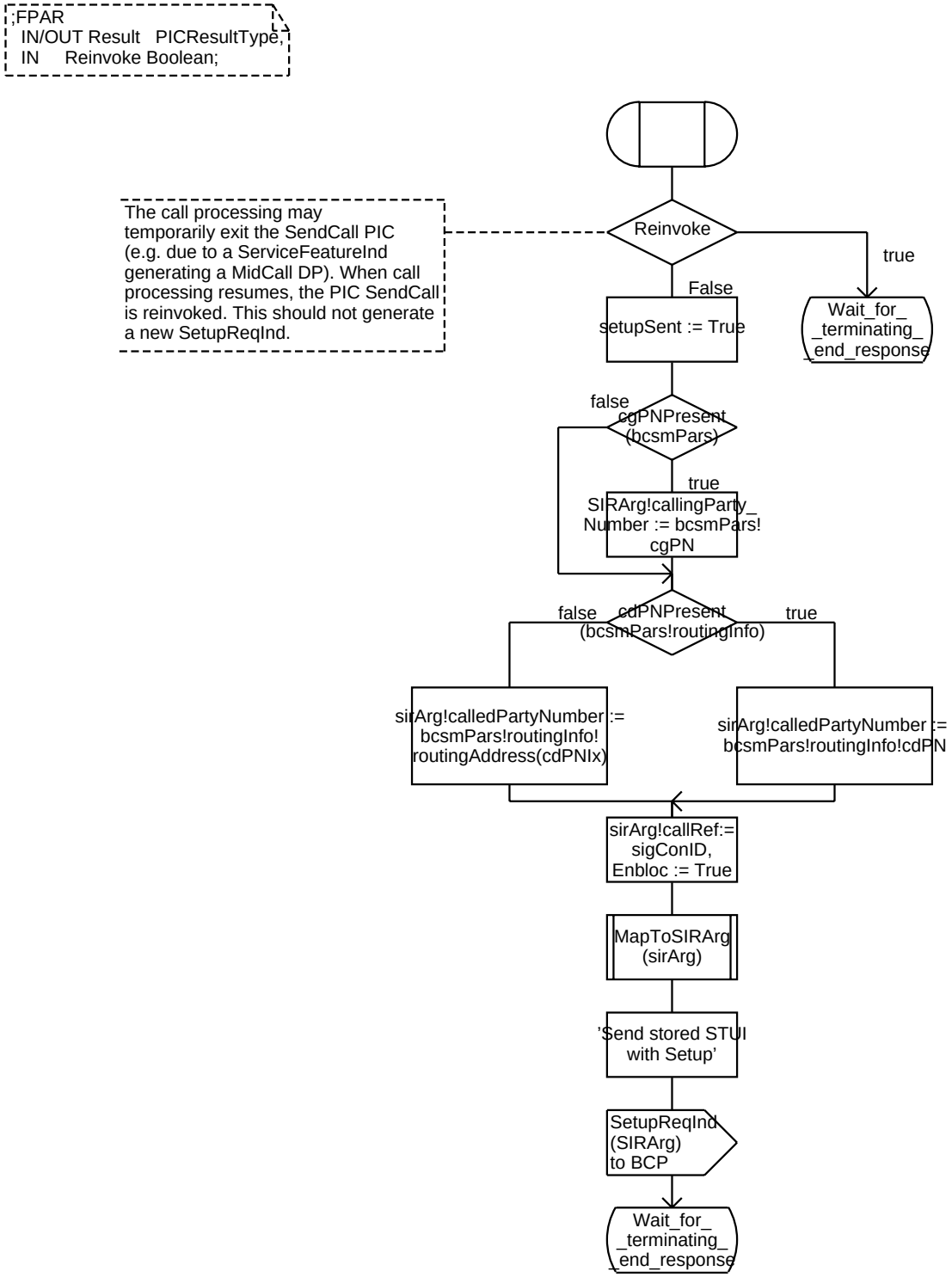
Procedure PIC_Analyse_Information 4(5)



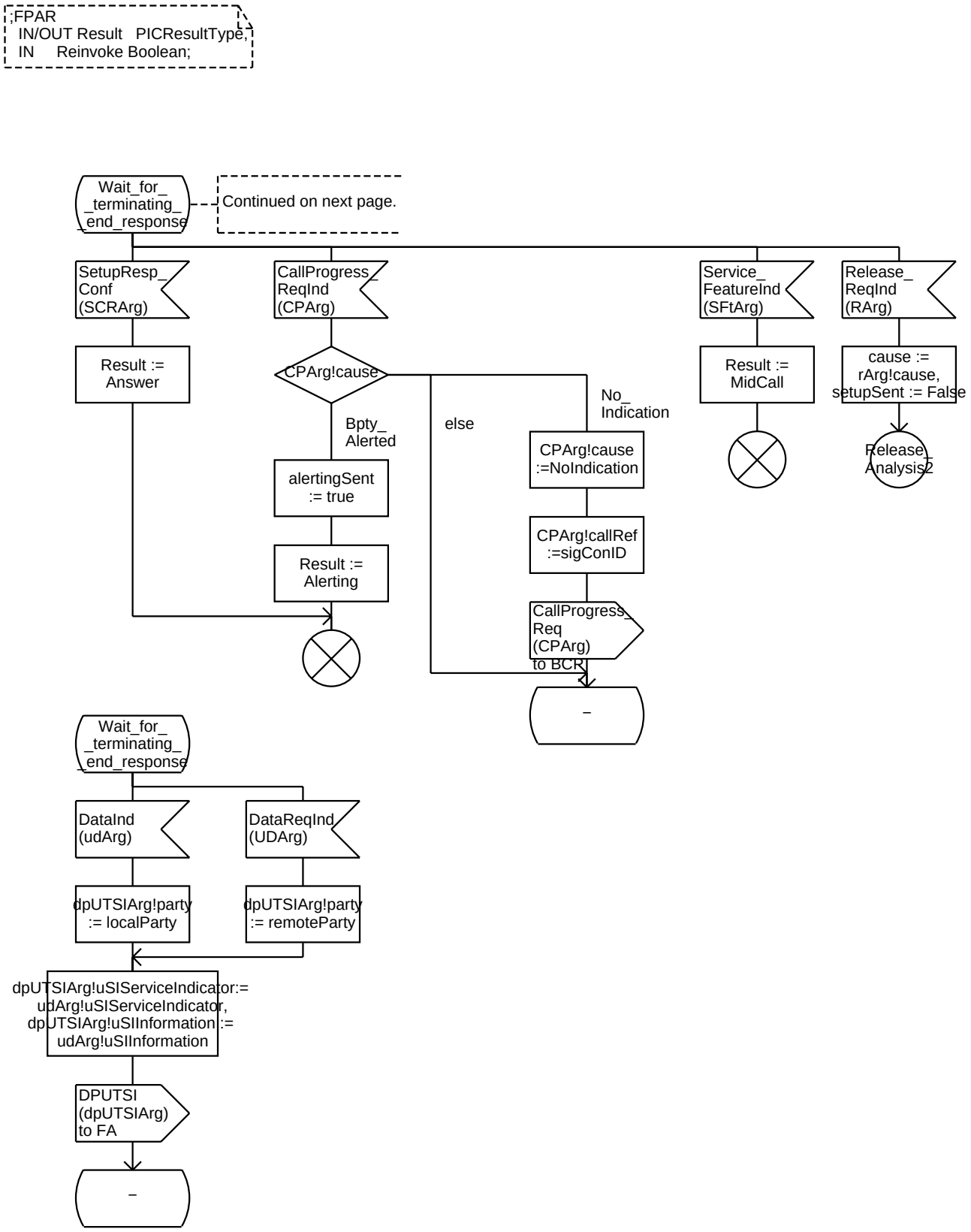
Procedure PIC_Analyse_Information 5(5)



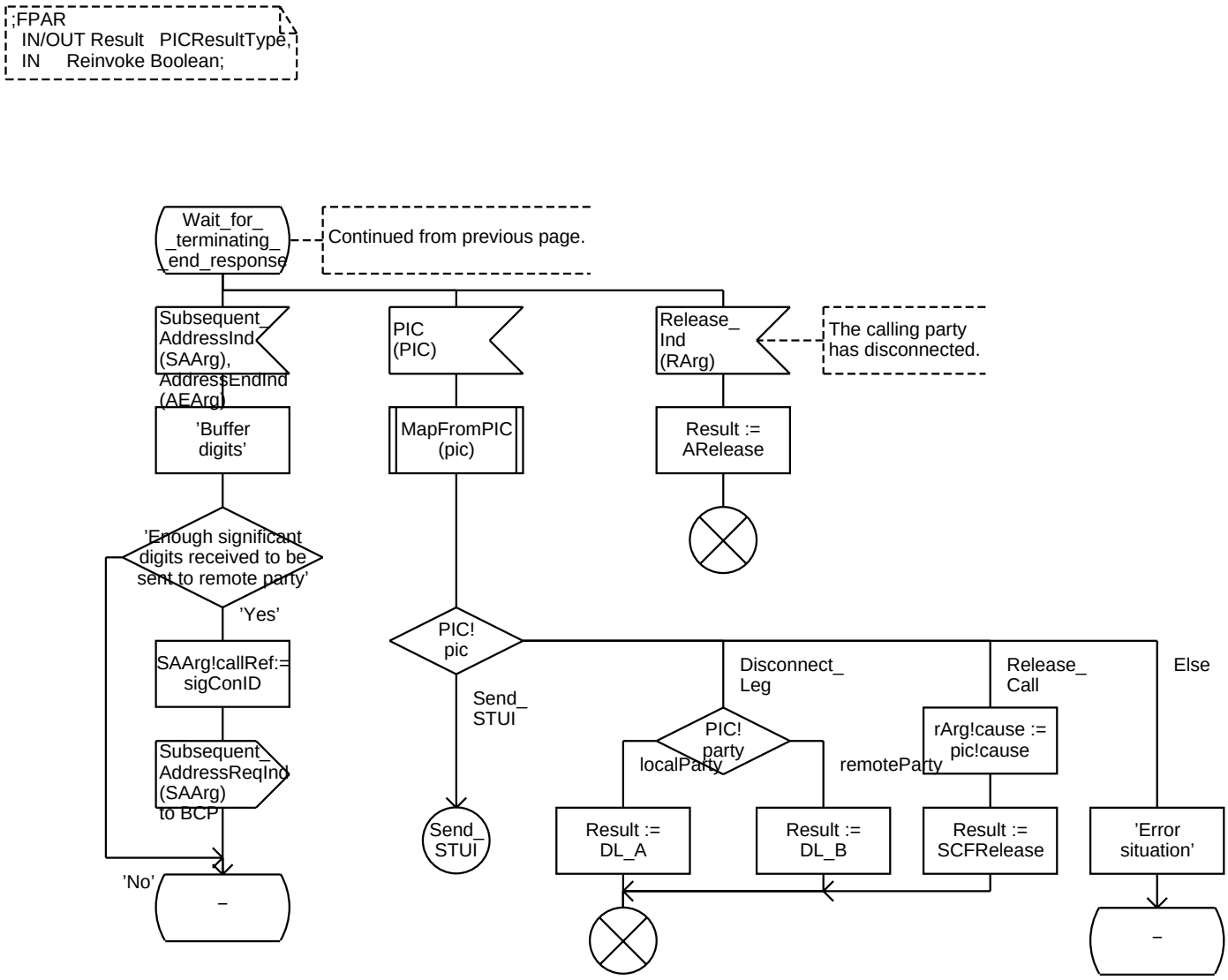
Procedure PIC_Send_Call 1(6)



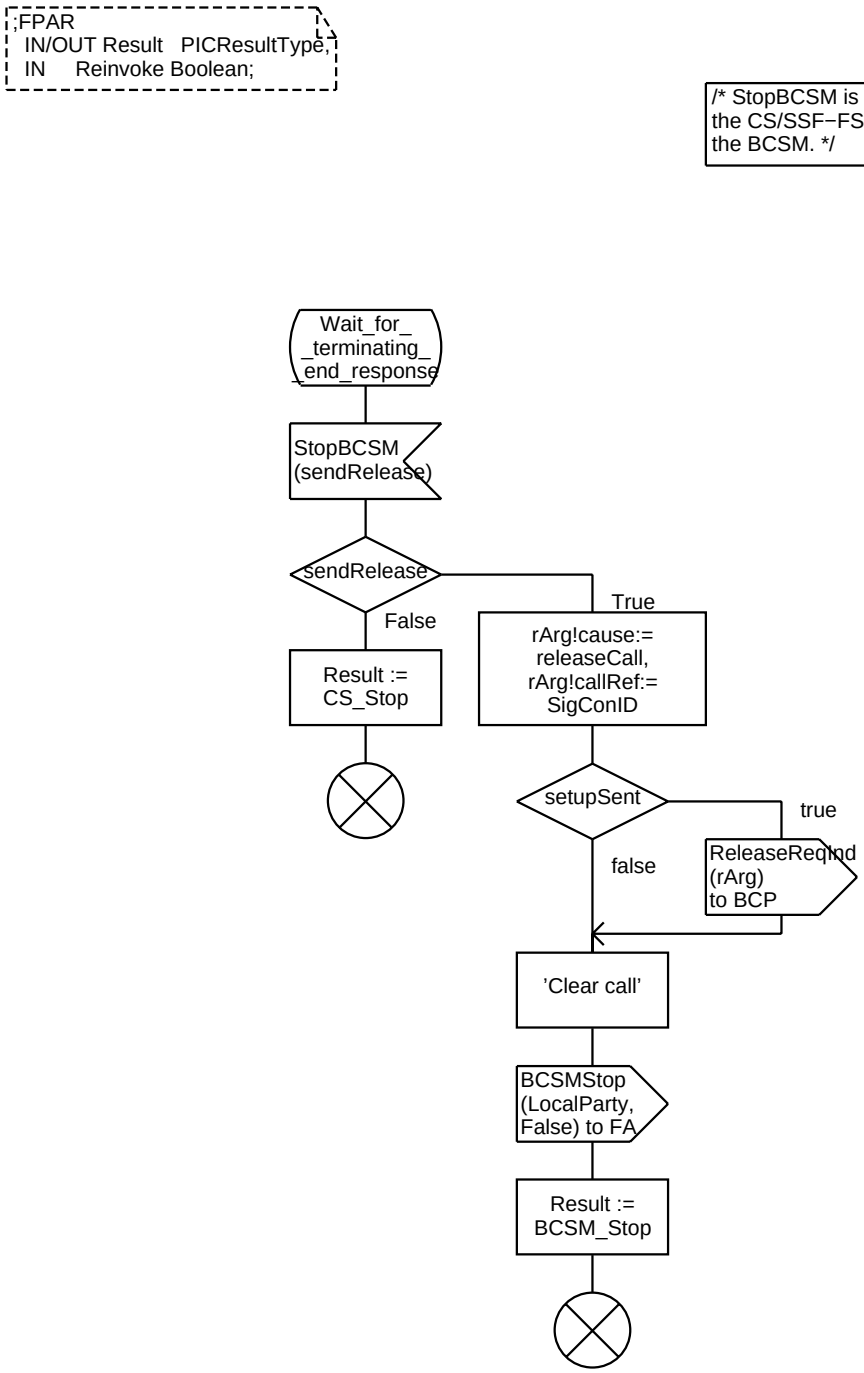
Procedure PIC_Send_Call 2(6)



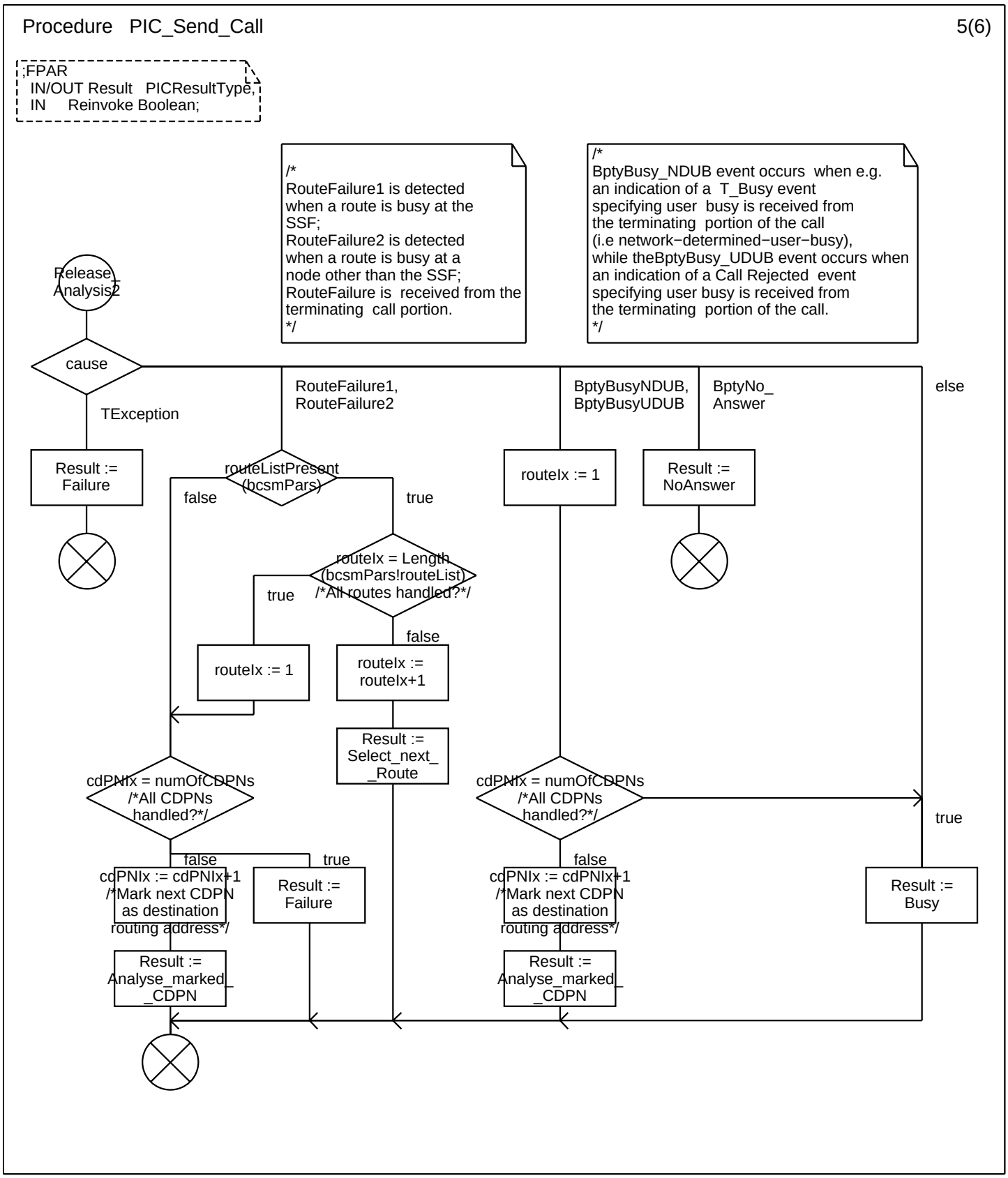
Procedure PIC_Send_Call 3(6)



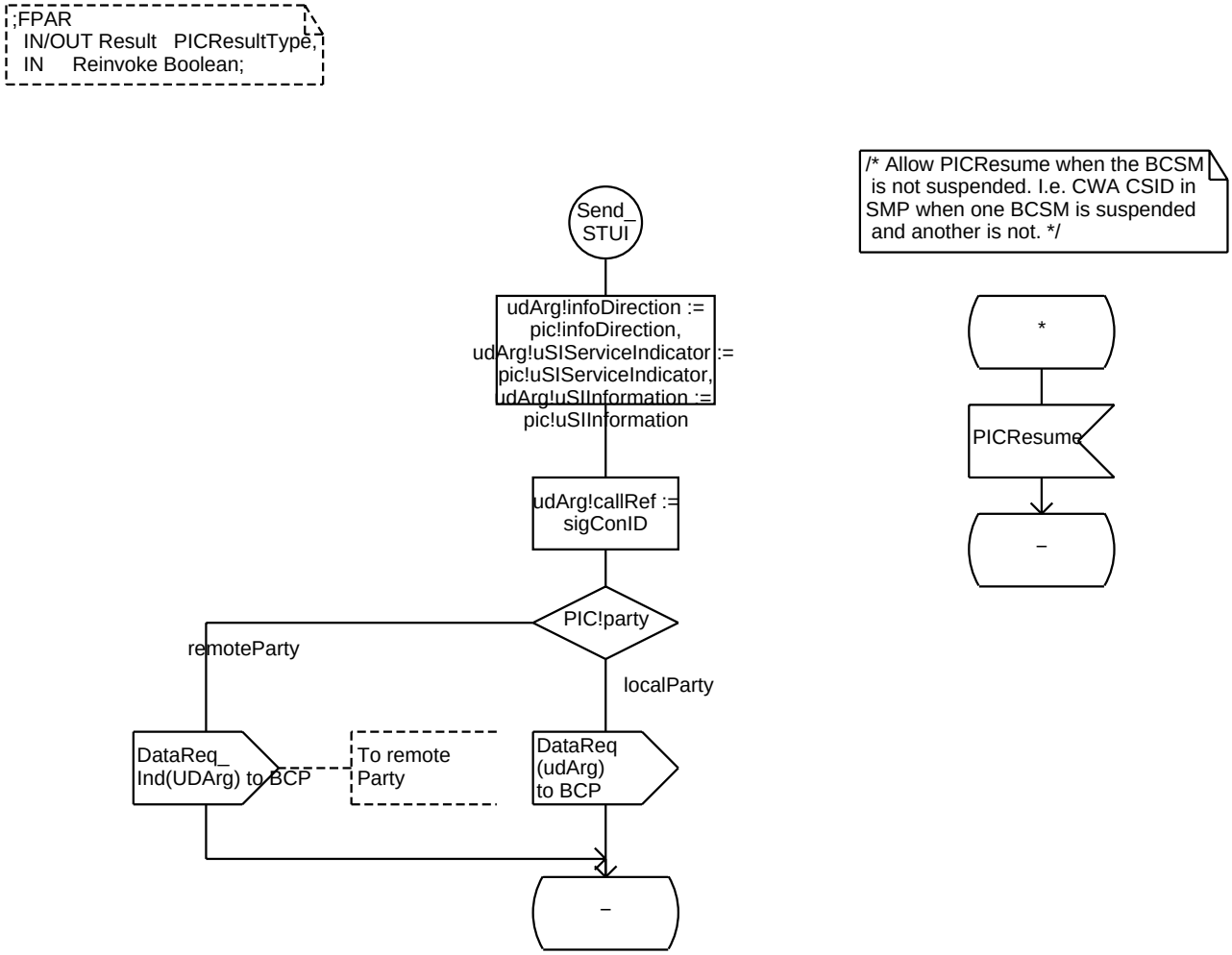
Procedure PIC_Send_Call 4(6)

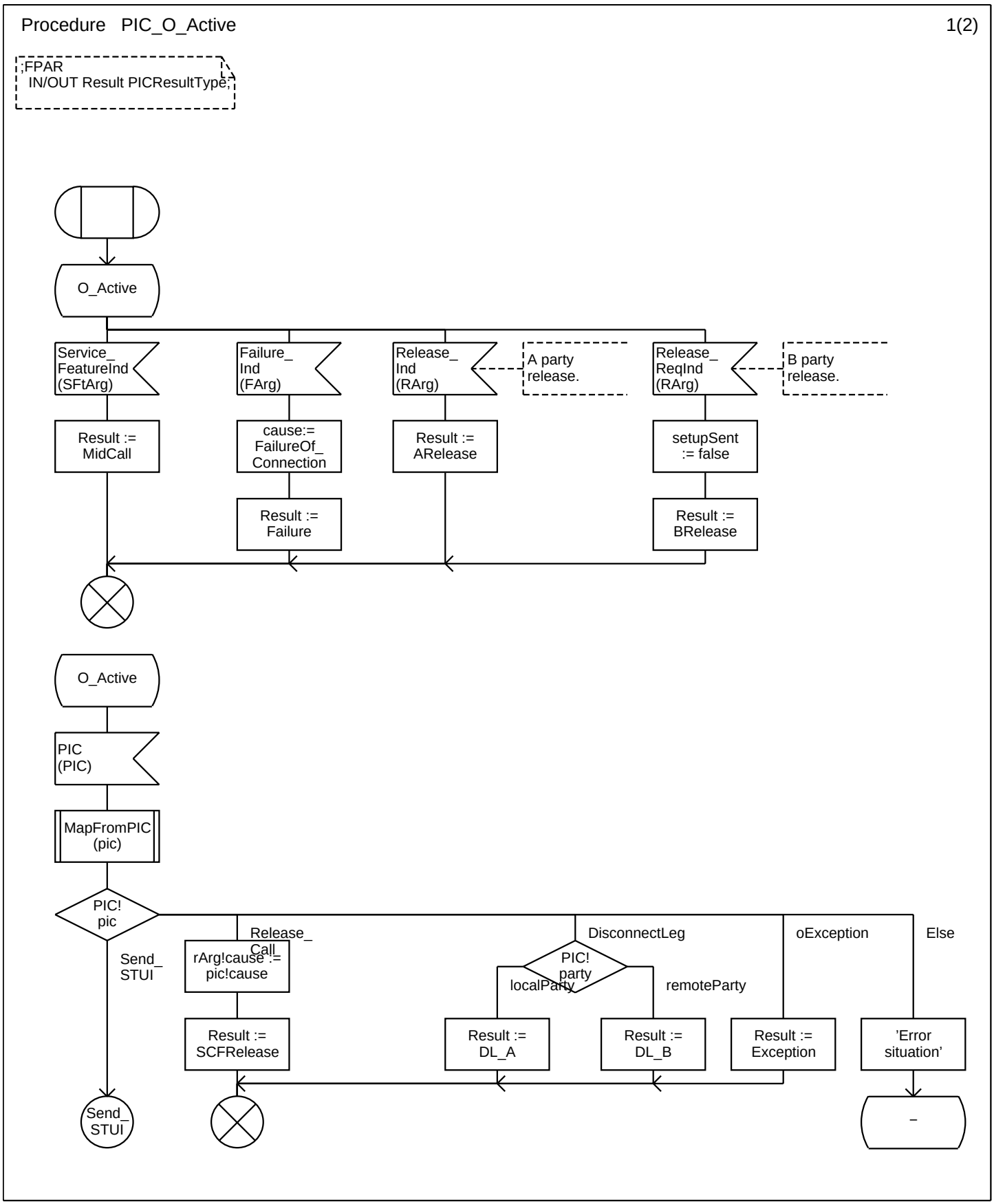


Procedure PIC_Send_Call 5(6)

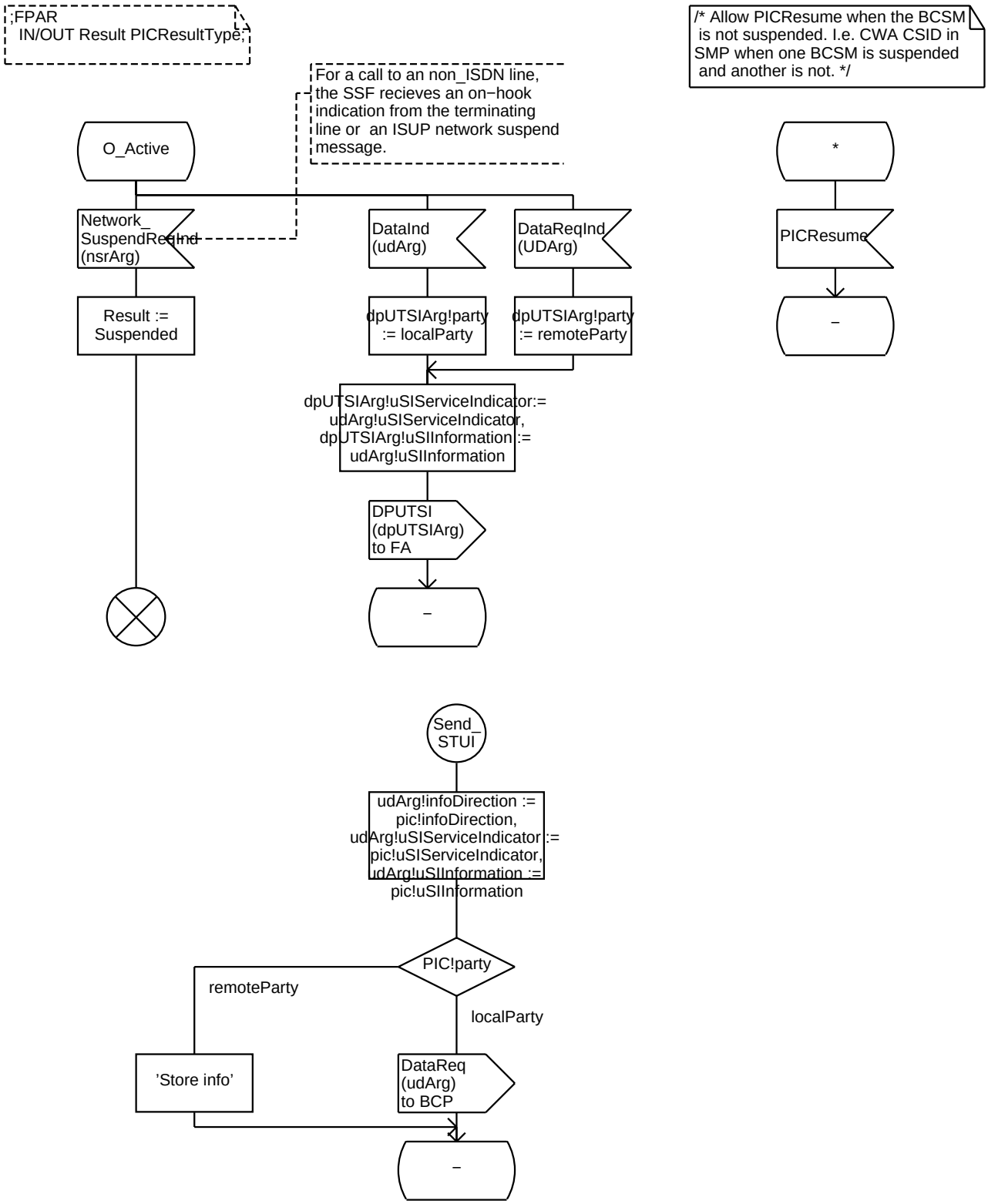


Procedure PIC_Send_Call 6(6)

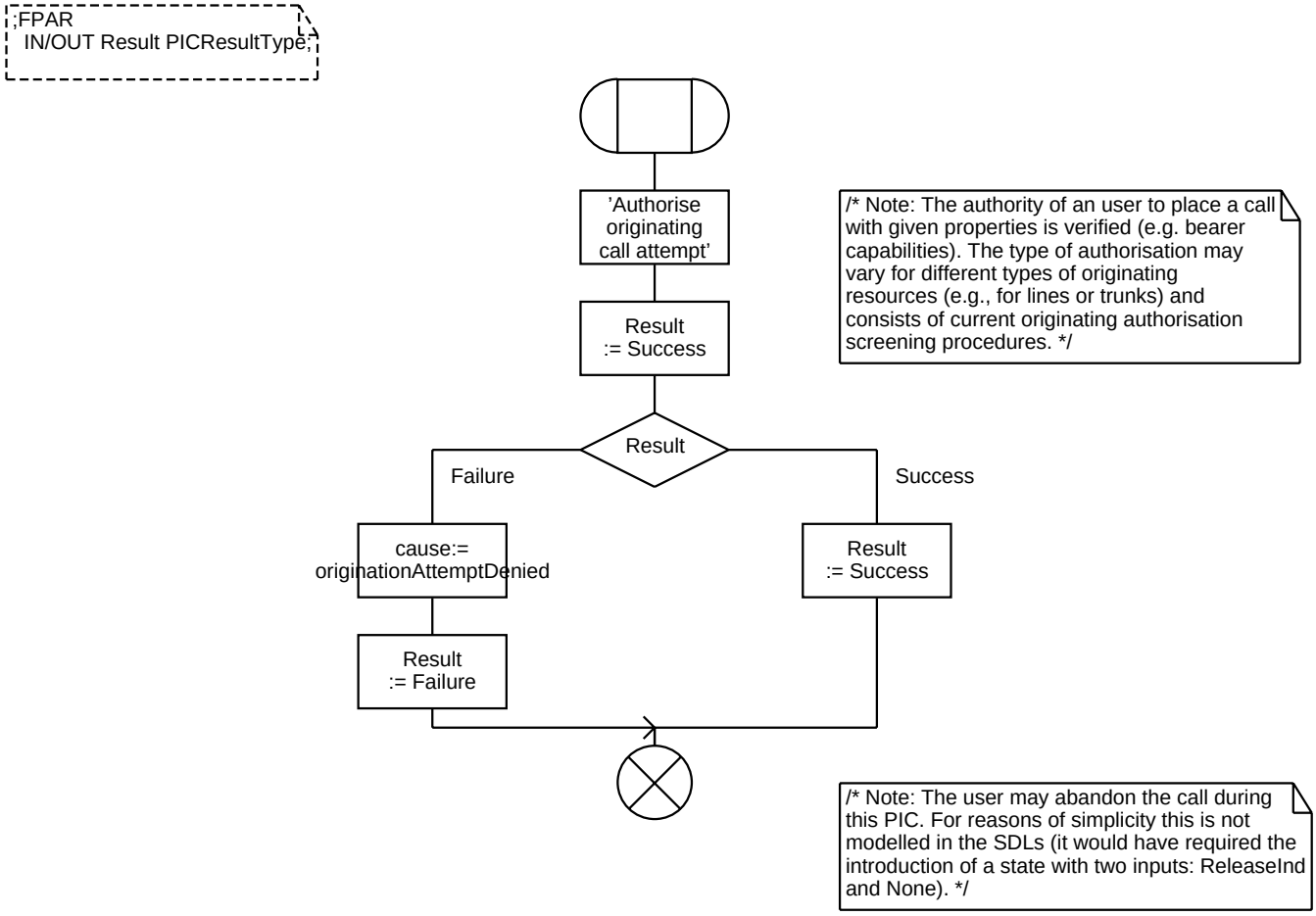



Procedure PIC_O_Active
1(2)
;FPAR
IN/OUT Result PICResultType;
O_Active
Service_ FeatureInd (SFtArg)
Failure_ Ind (FArg)
Release_ Ind (RArg)
A party release.
Release_ ReqInd (RArg)
B party release.
Result := MidCall
cause:= FailureOf_ Connection
Result := ARelease
setupSent := false
Result := Failure
Result := BRelease
O_Active
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
–

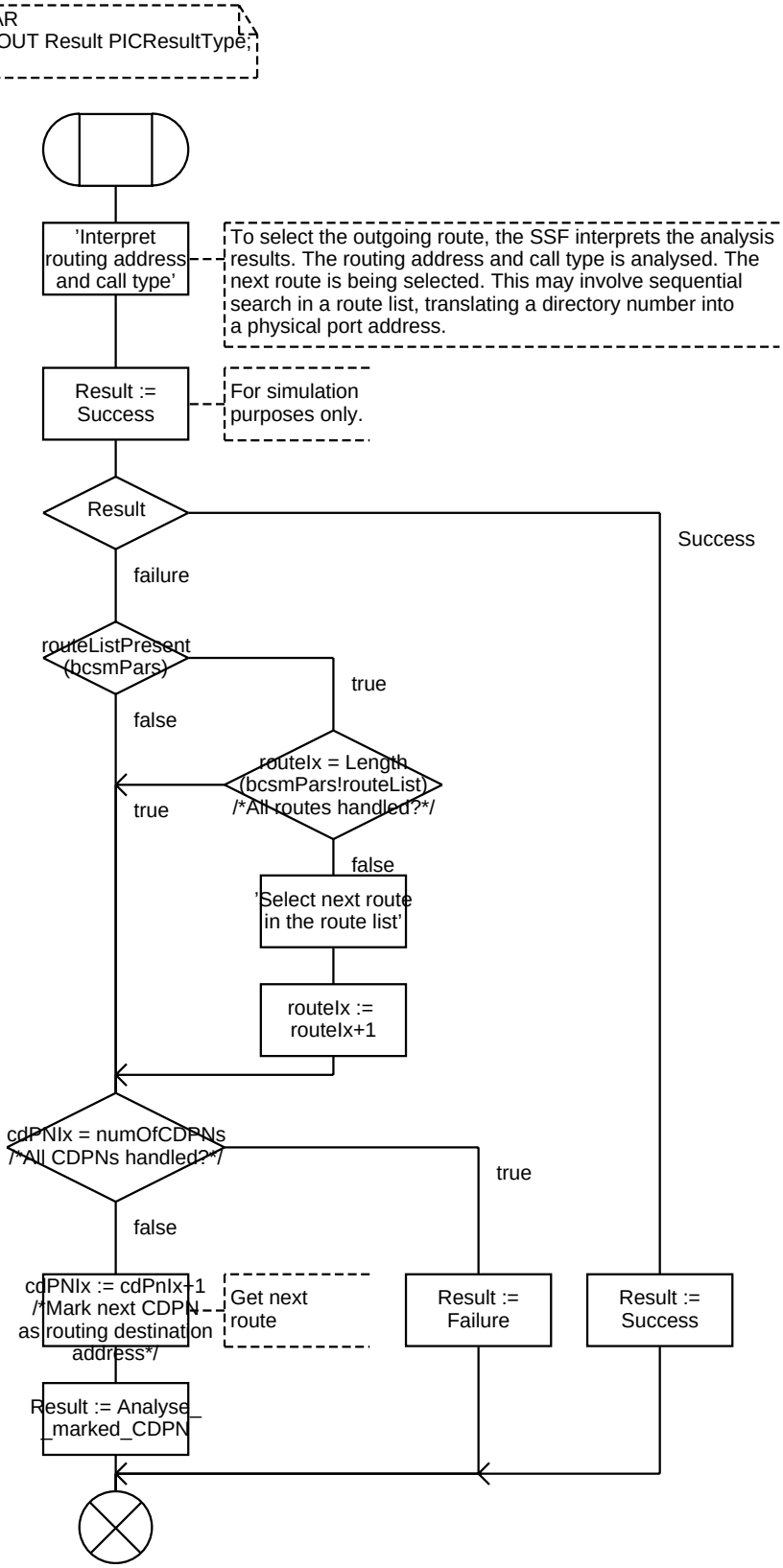
Procedure PIC_O_Active 2(2)



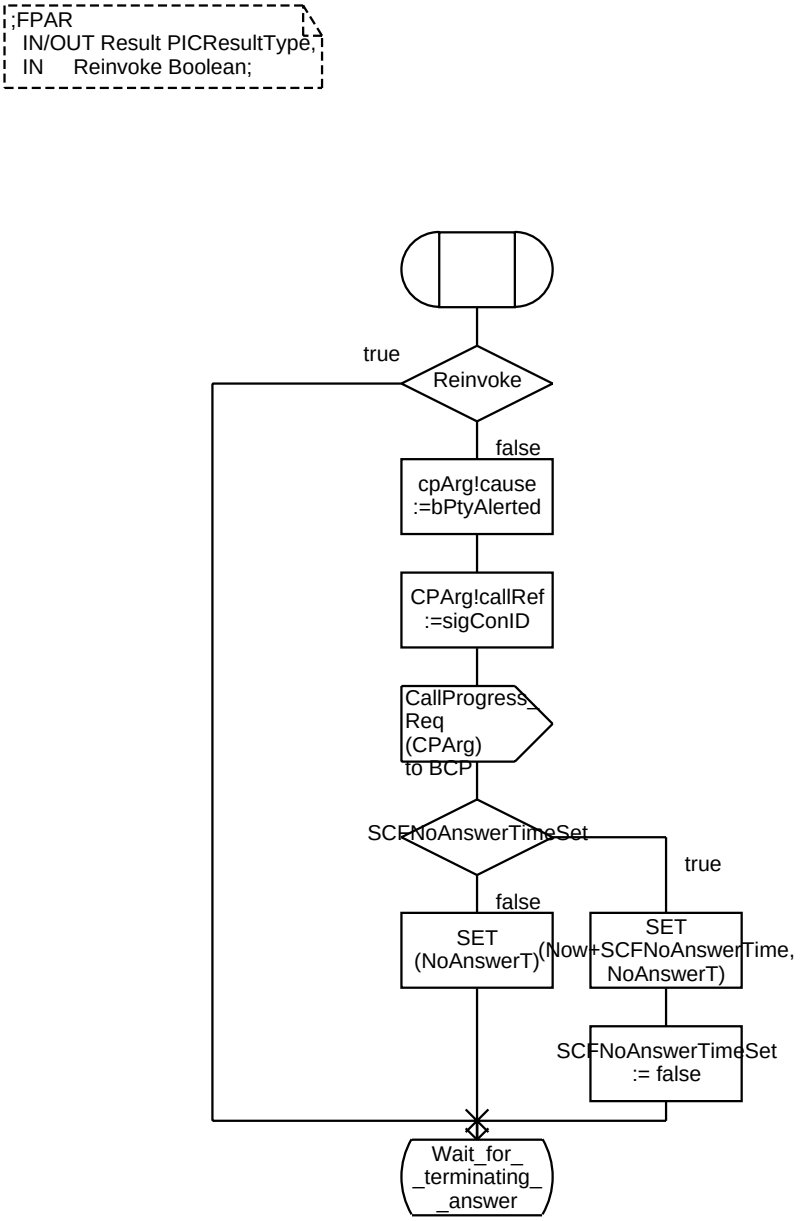
Procedure PIC_Authorise_Origination_Attempt 1(1)



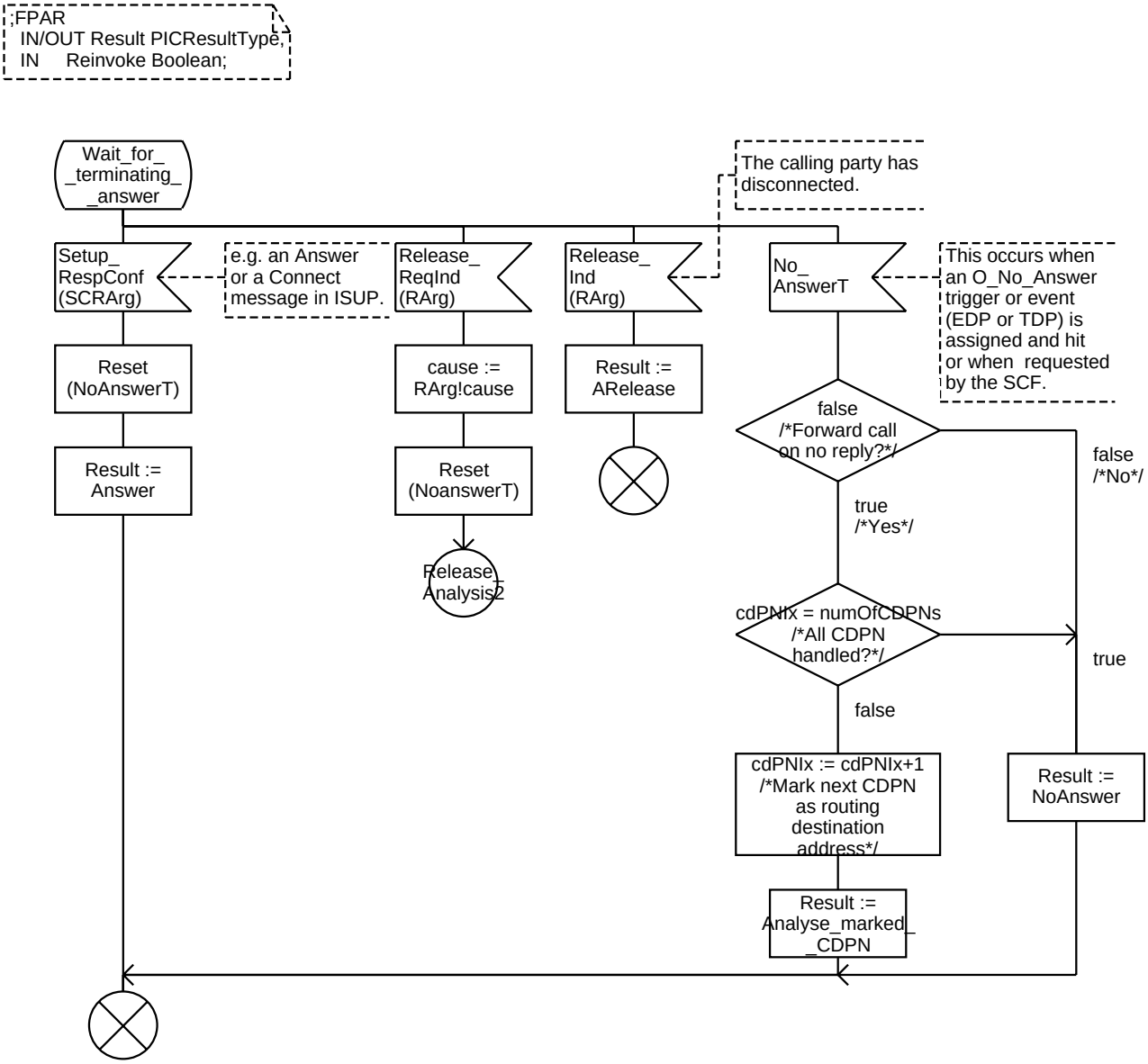
Procedure PIC_Select_Route 1(1)



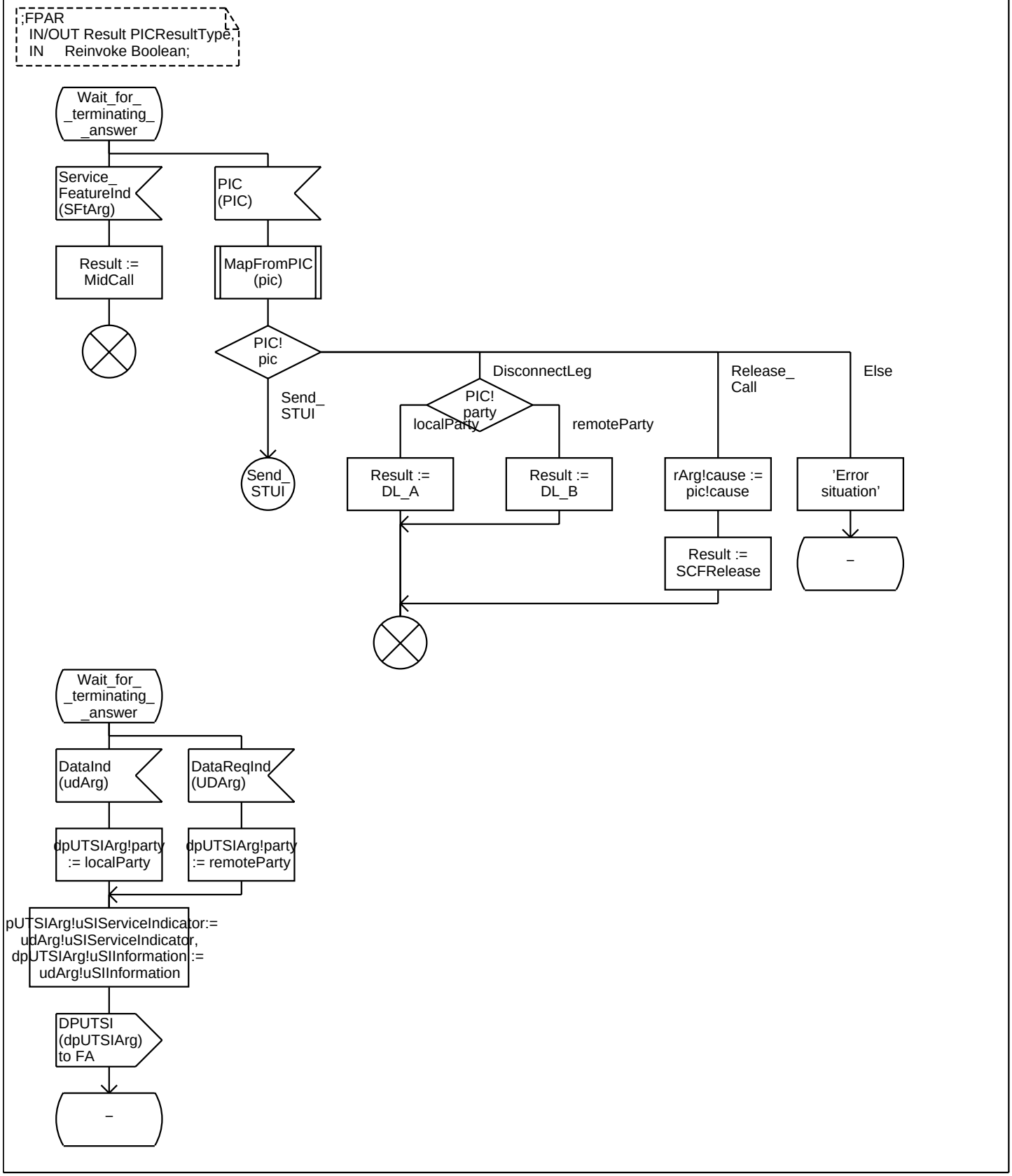
Procedure PIC_O_Alerting 1(5)



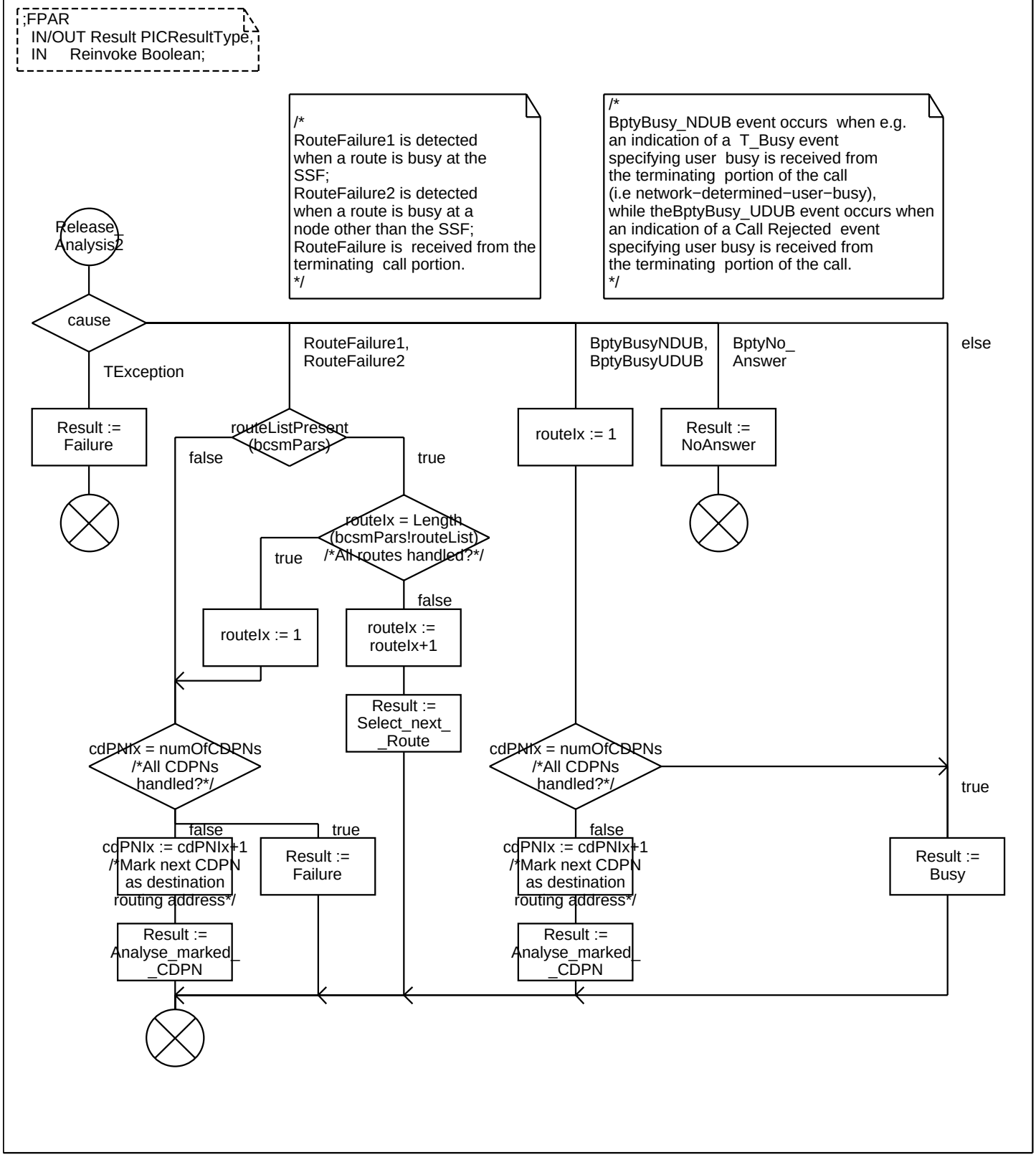
Procedure PIC_O_Alerting 2(5)



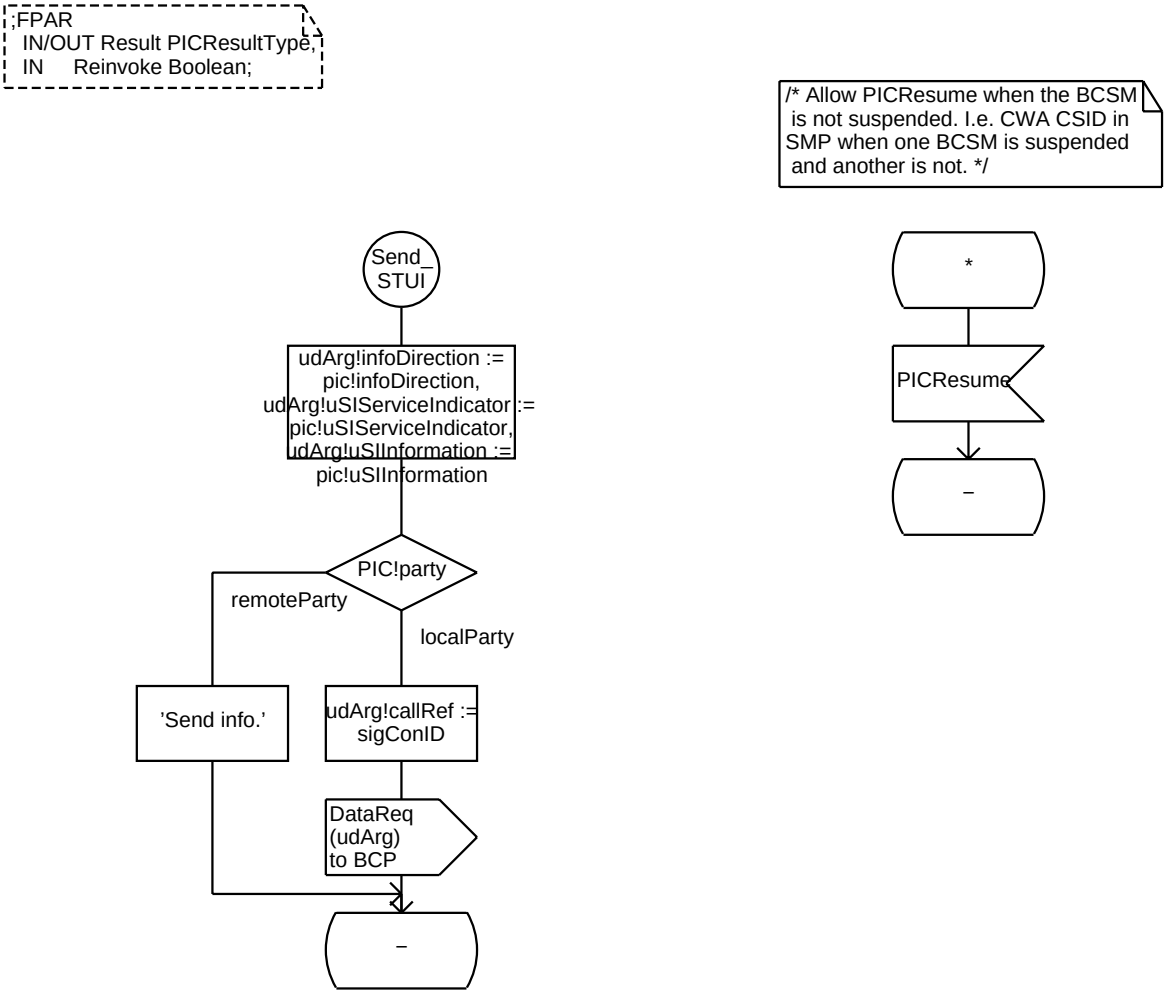
Procedure PIC_O_Alerting 3(5)



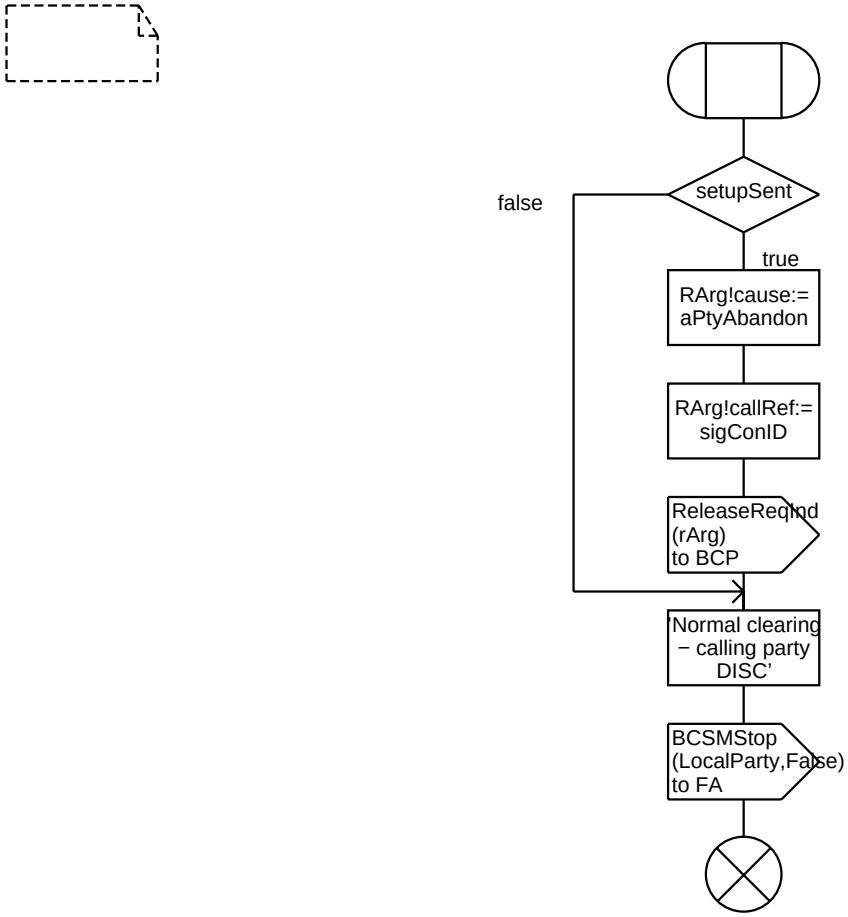
Procedure PIC_O_Alerting 4(5)



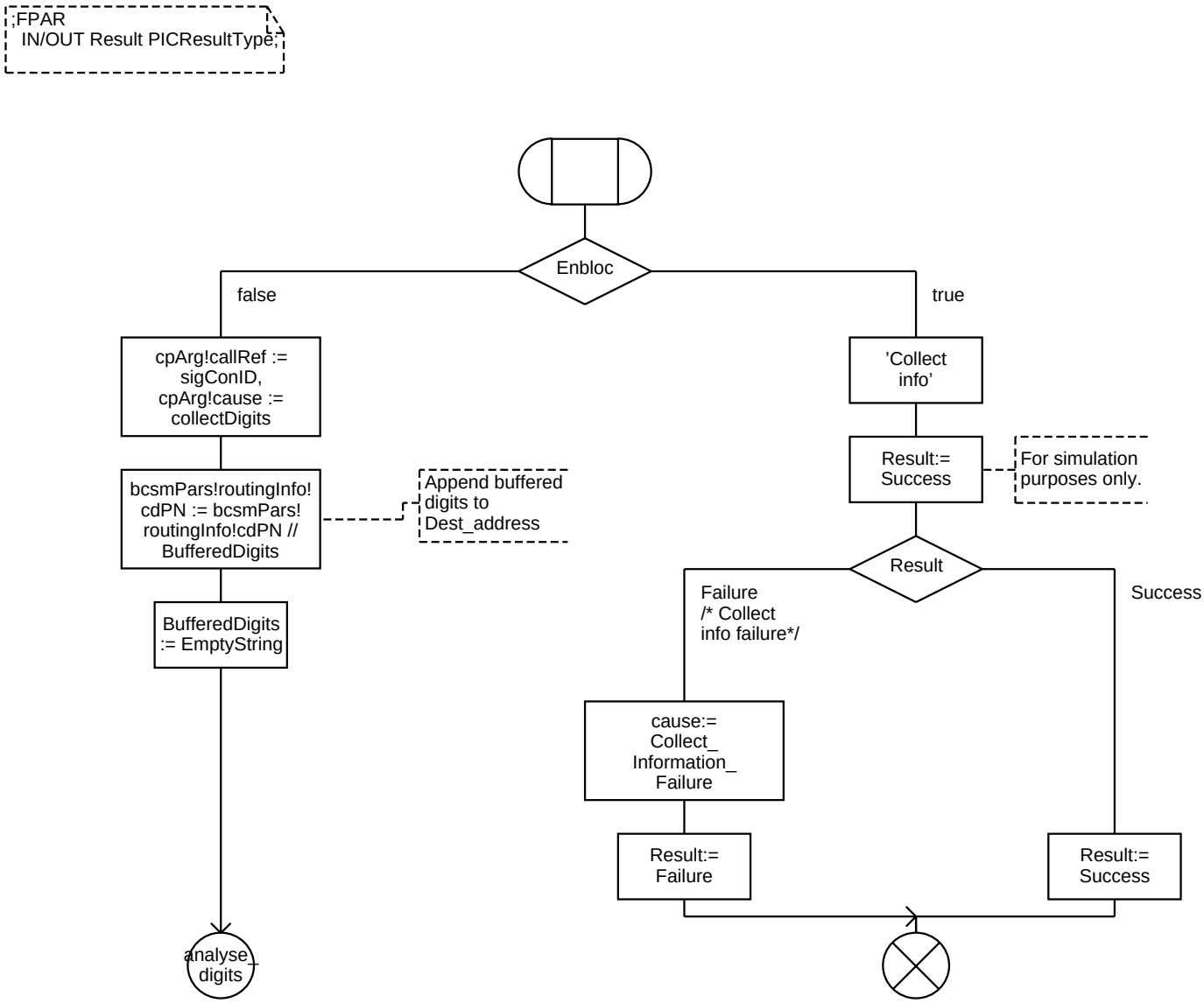
Procedure PIC_O_Alerting 5(5)



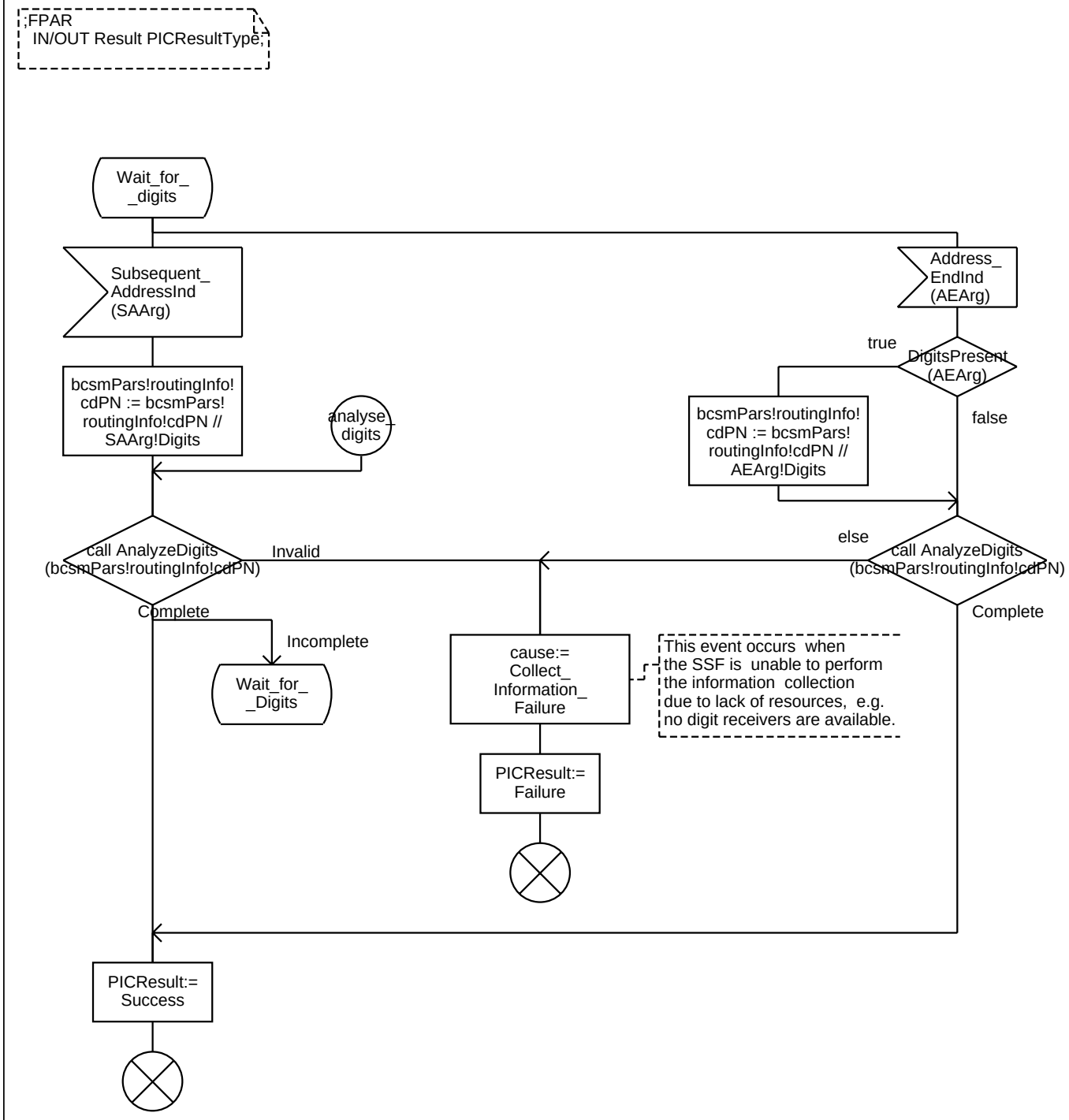
Procedure PIC_O_Abandon 1(1)



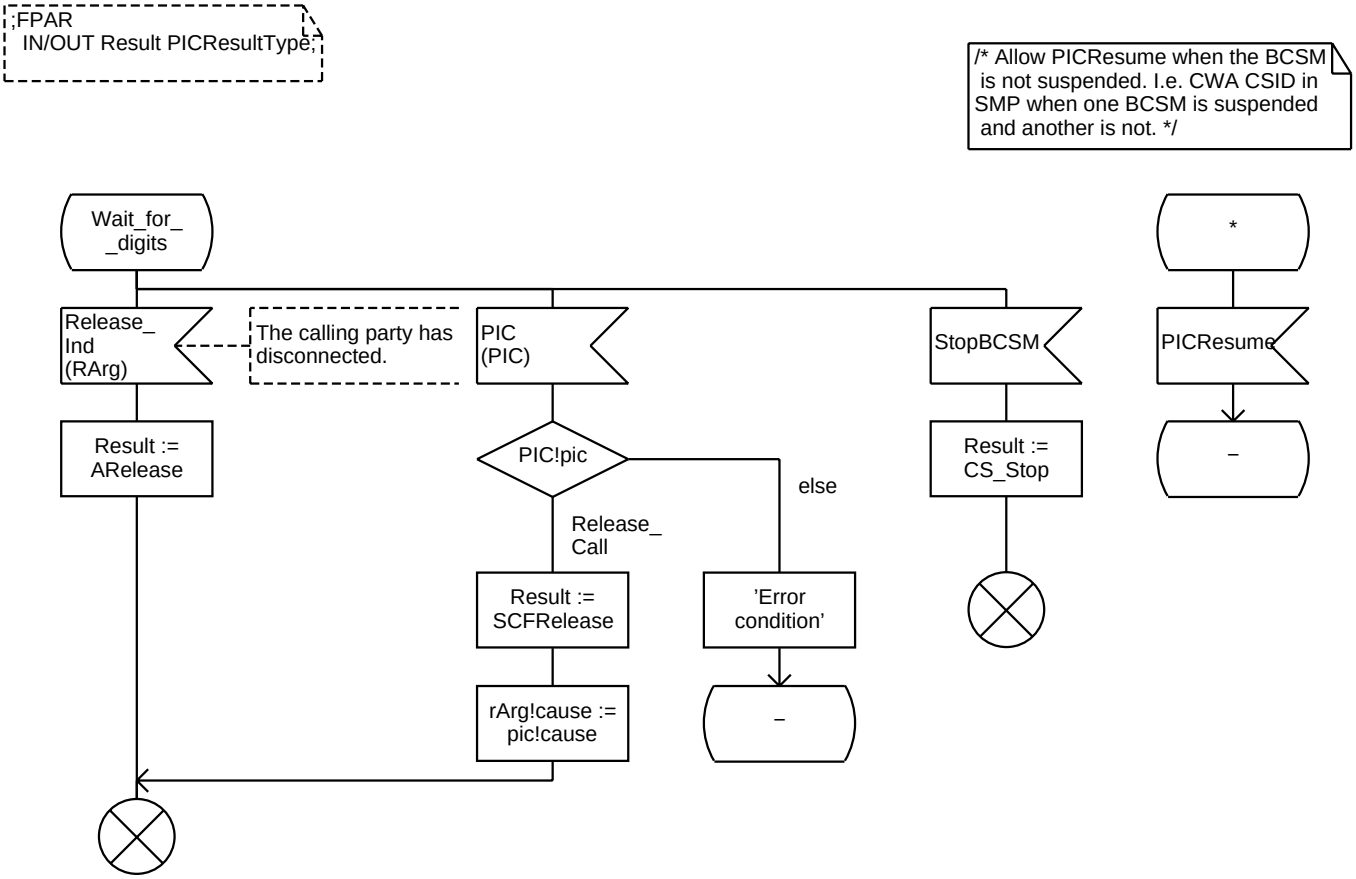
Procedure PIC_Collect_Information 1(3)



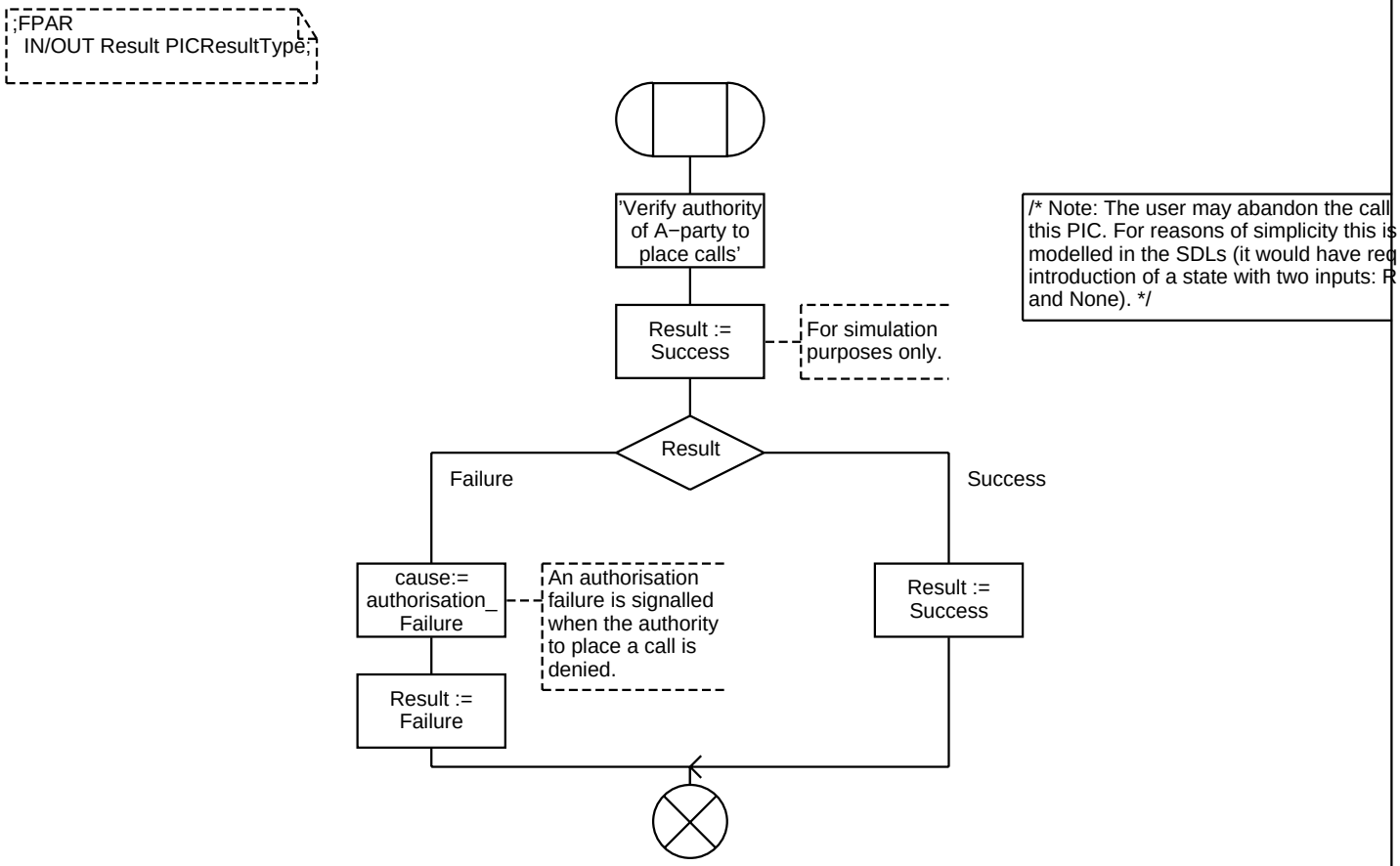
Procedure PIC_Collect_Information 2(3)



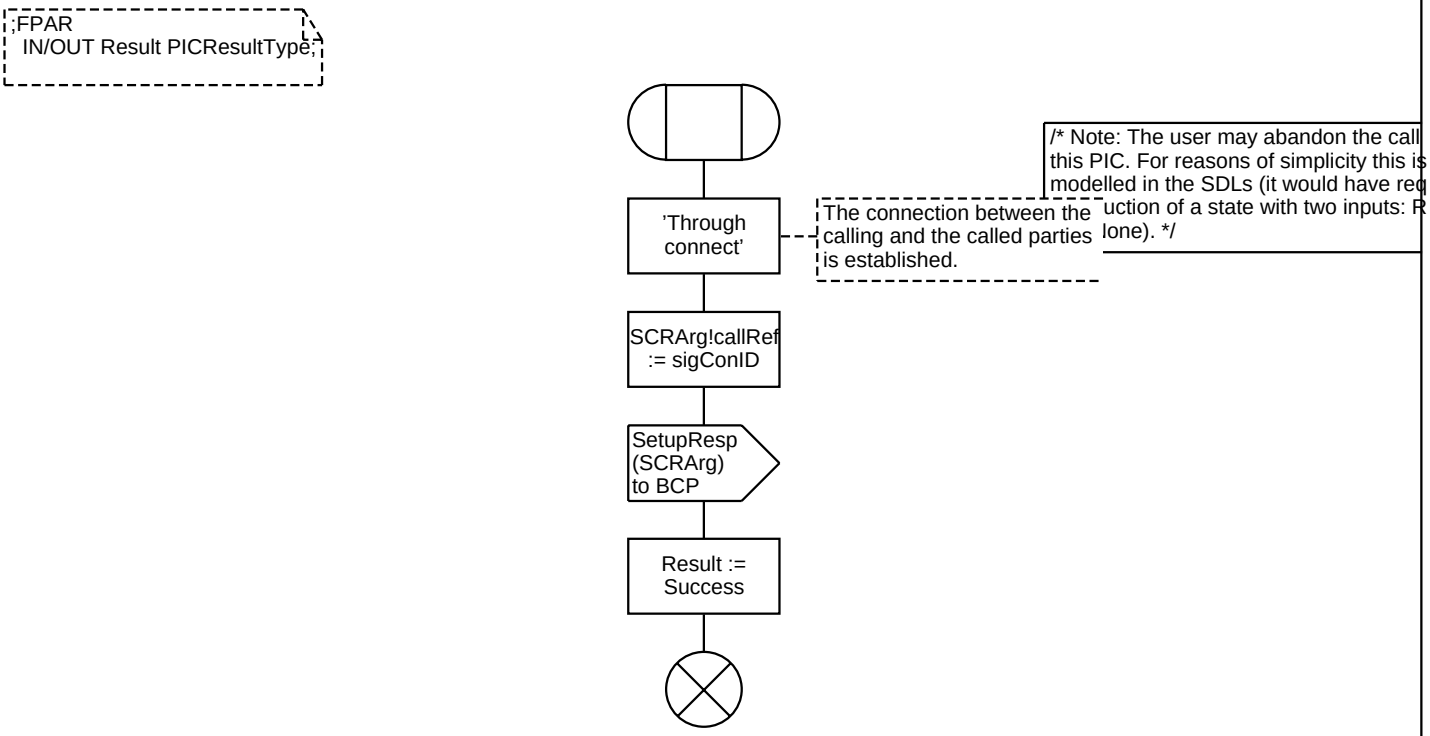
Procedure PIC_Collect_Information 3(3)



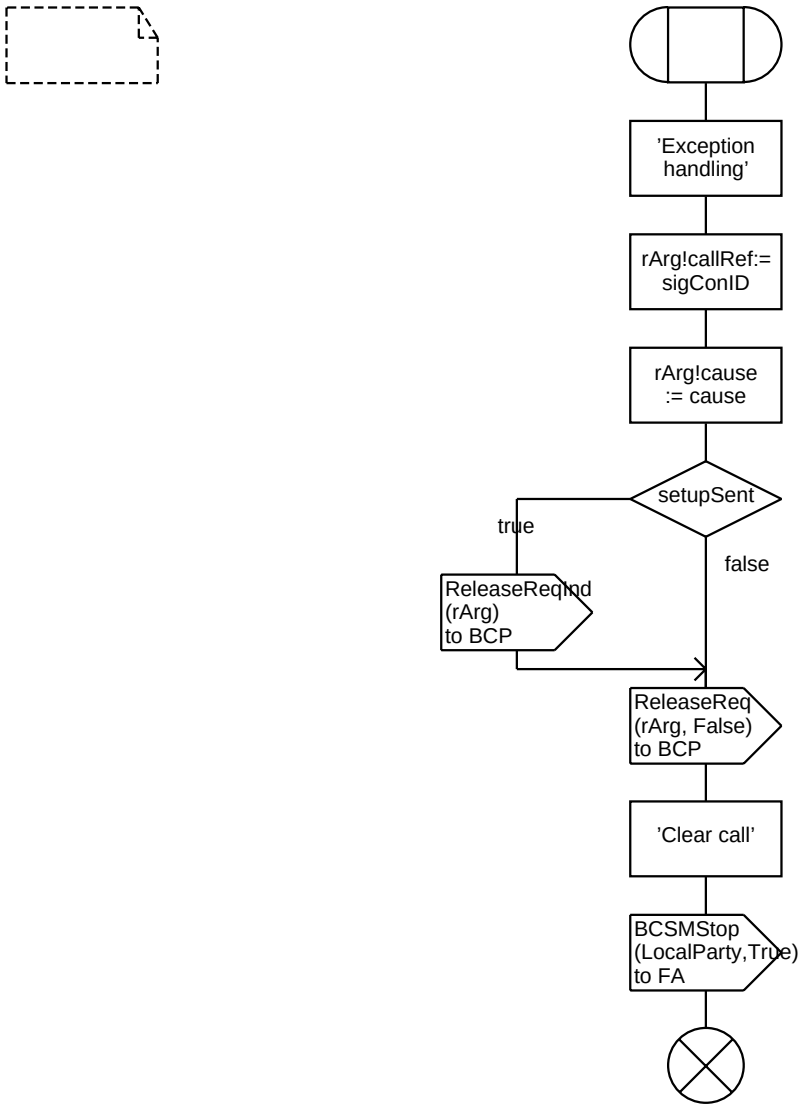
Procedure PIC_Authorise_Call_Setup 1(1)



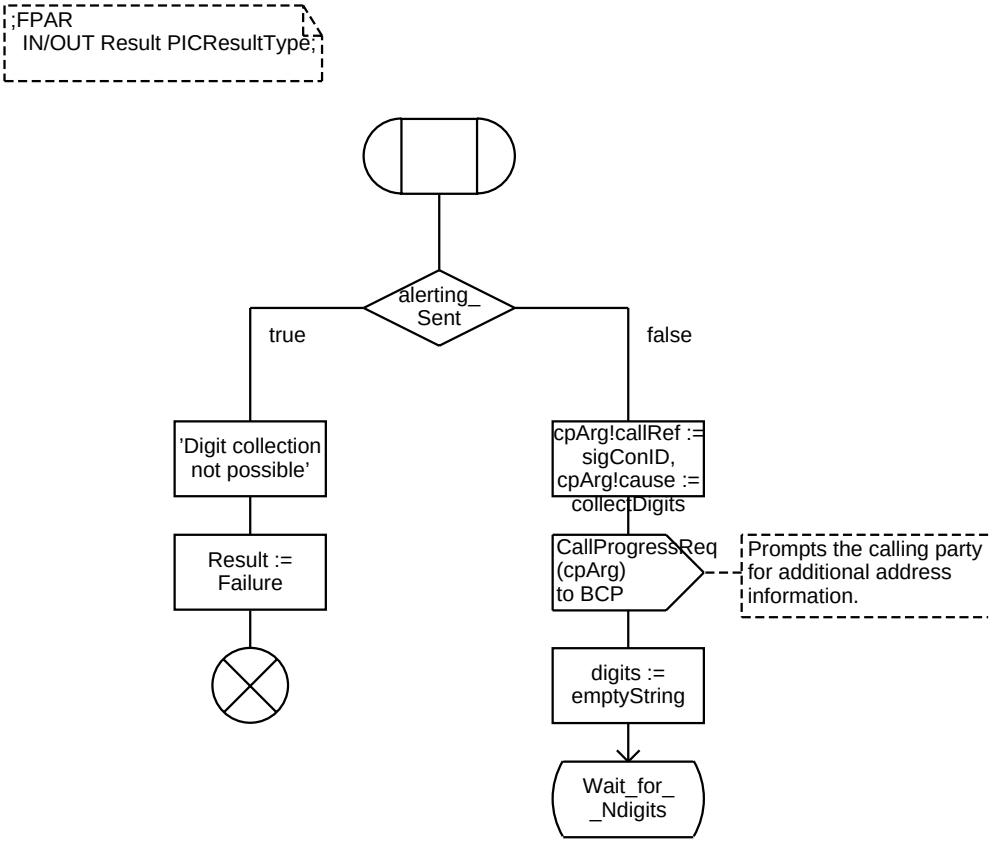
Procedure PIC_O_Answer 1(1)



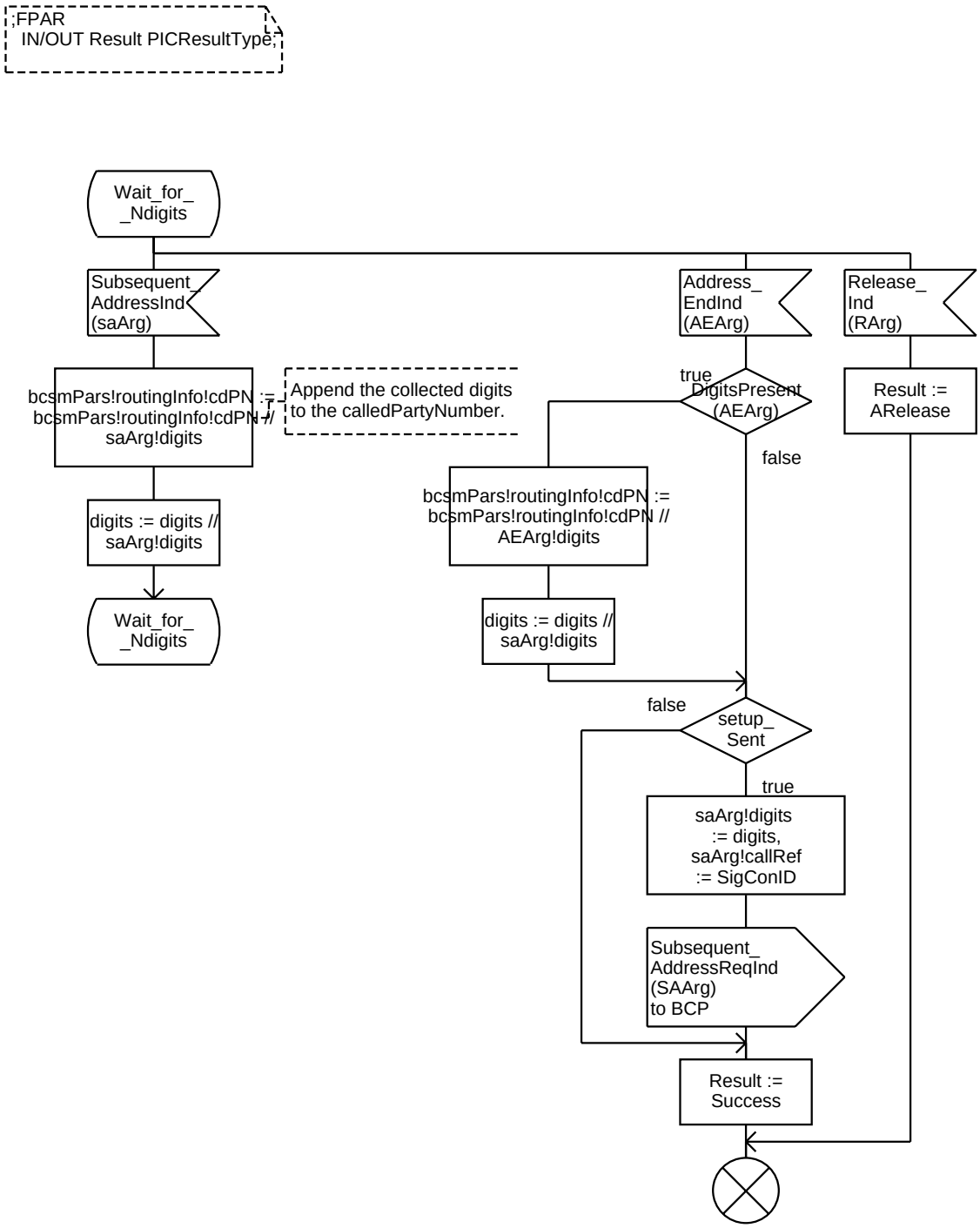
Procedure PIC_OException 1(1)



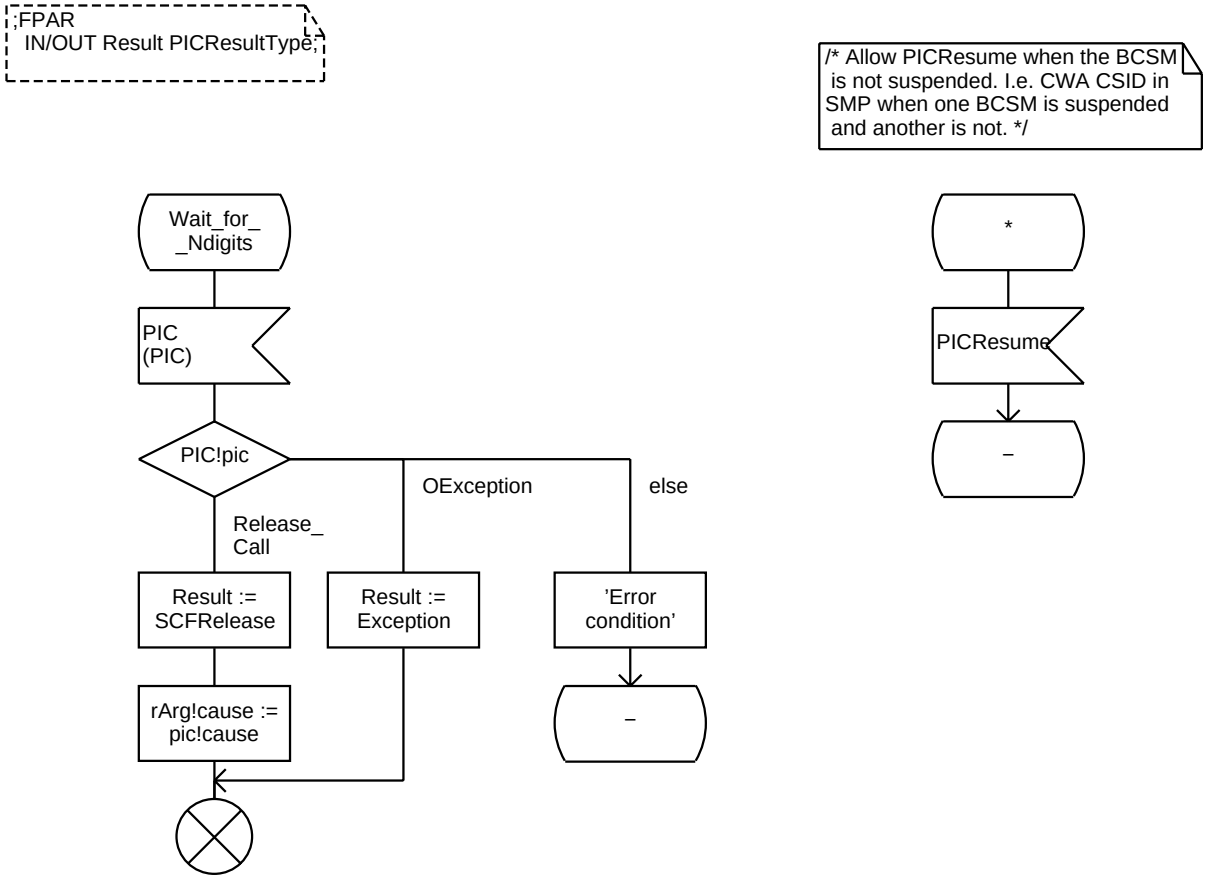
Procedure PIC_Collect_NDigits 1(3)



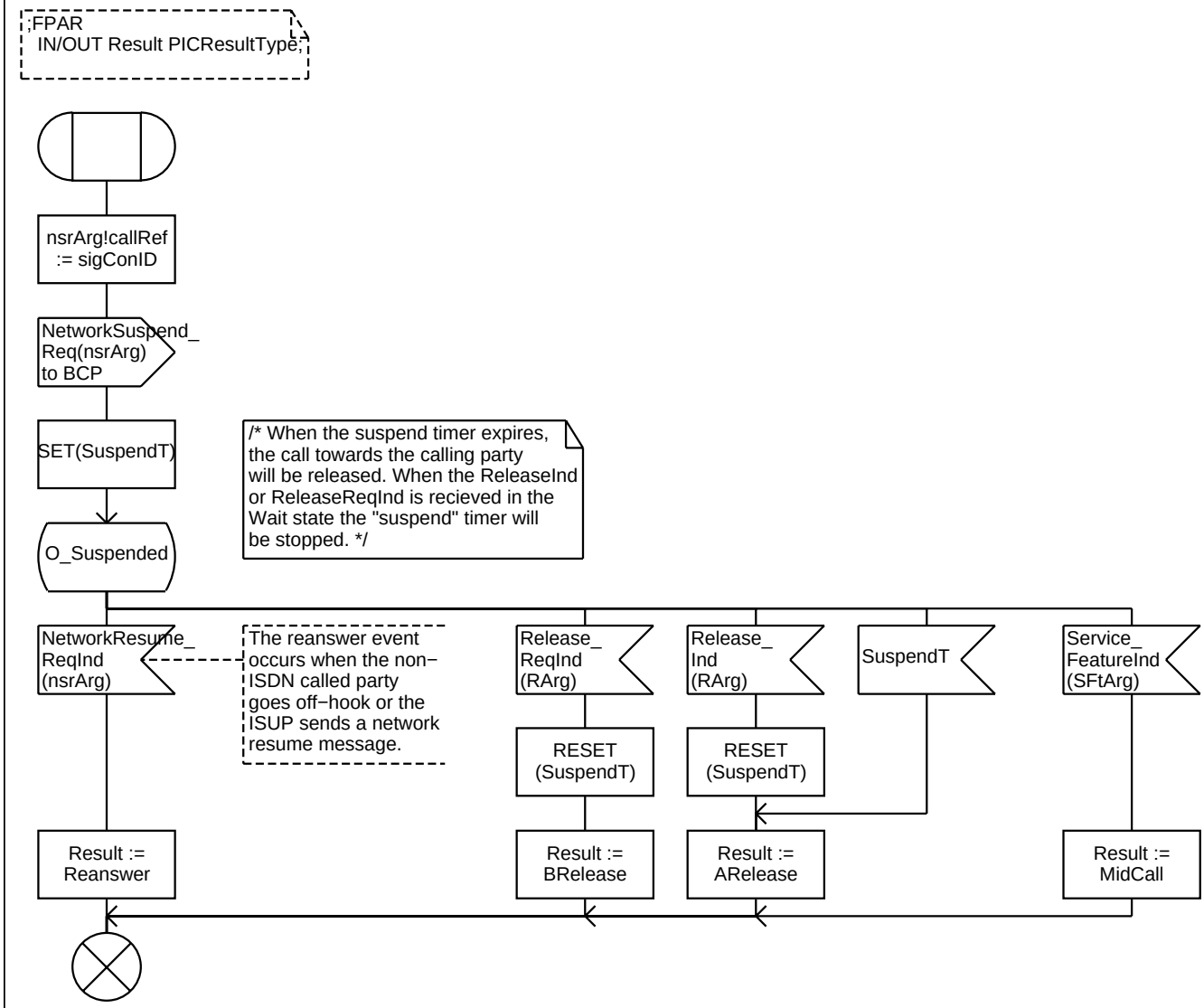
Procedure PIC_Collect_NDigits 2(3)



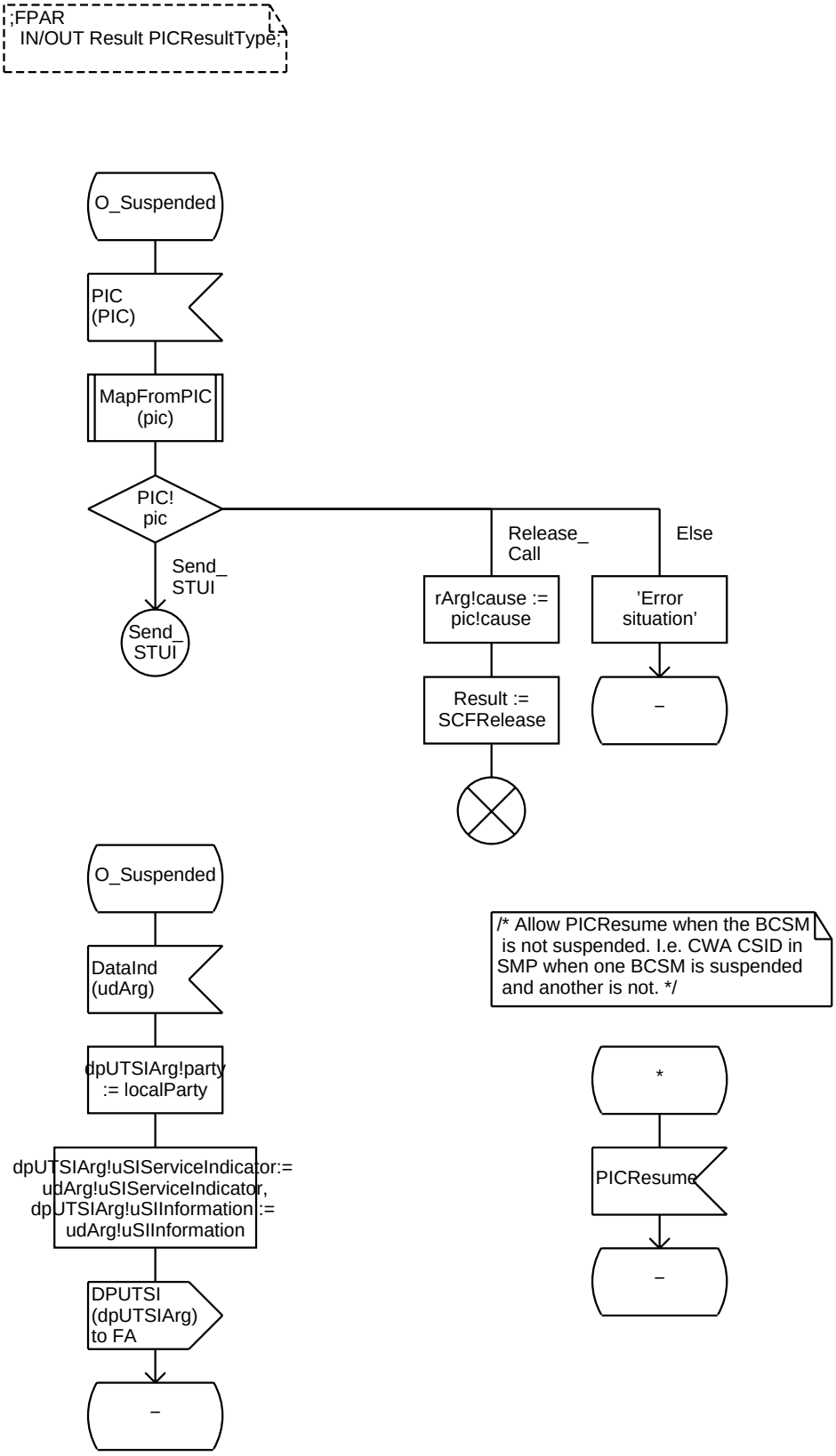
Procedure PIC_Collect_NDigits 3(3)



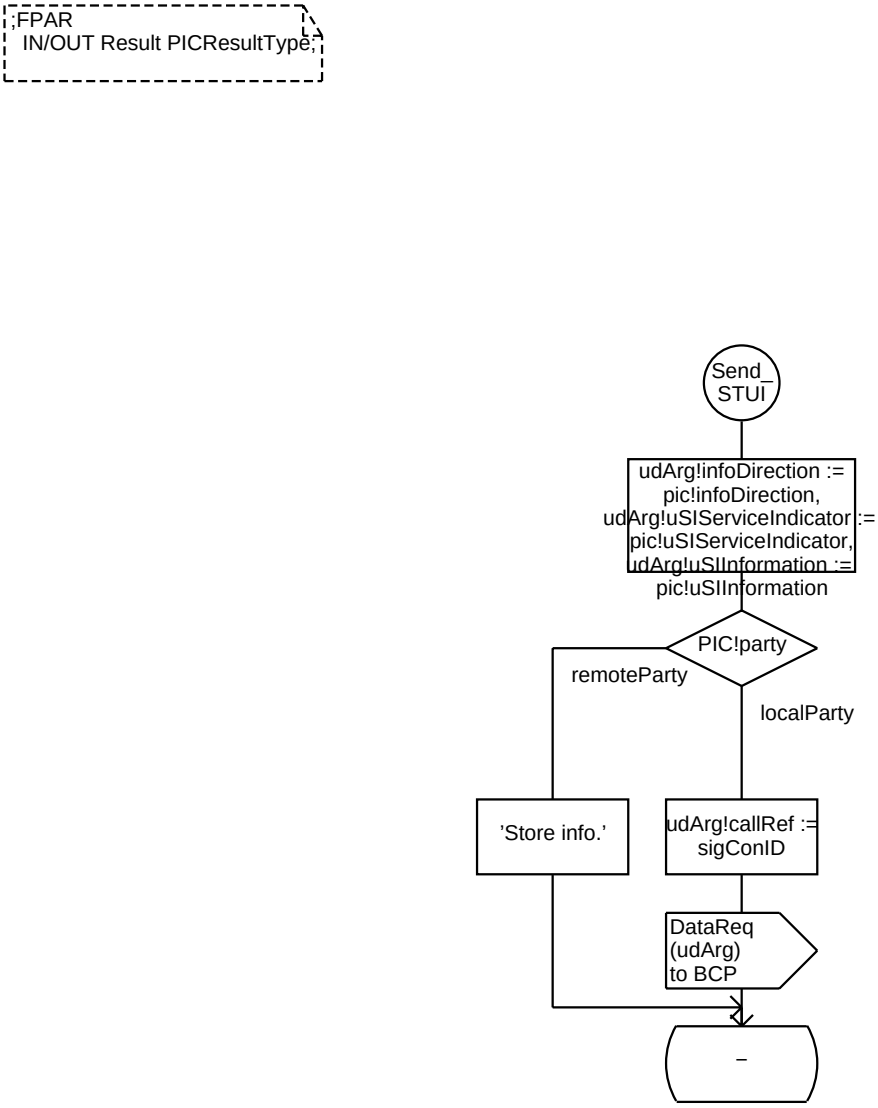
Procedure PIC_O_Suspended 1(3)



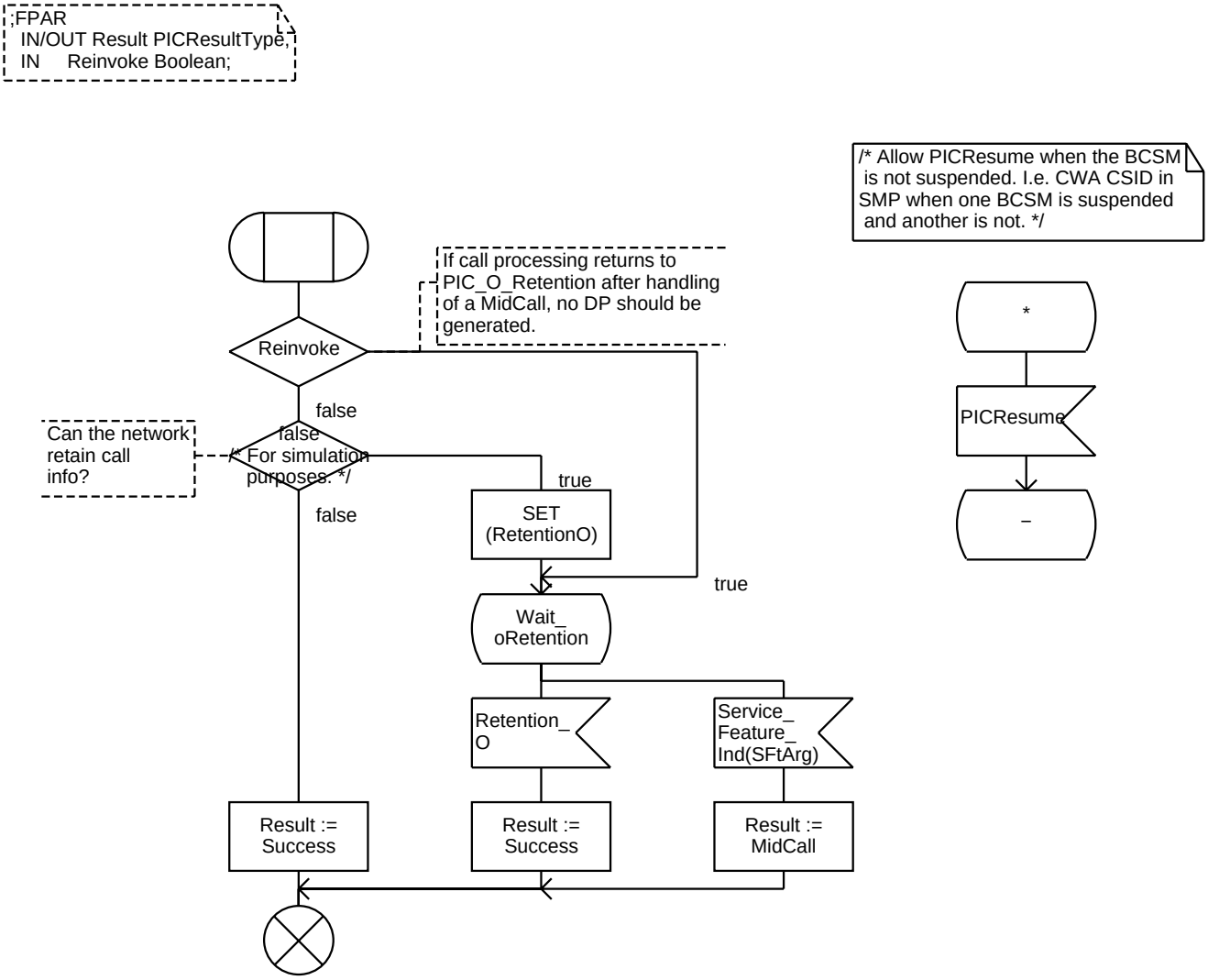
Procedure PIC_O_Suspended 2(3)



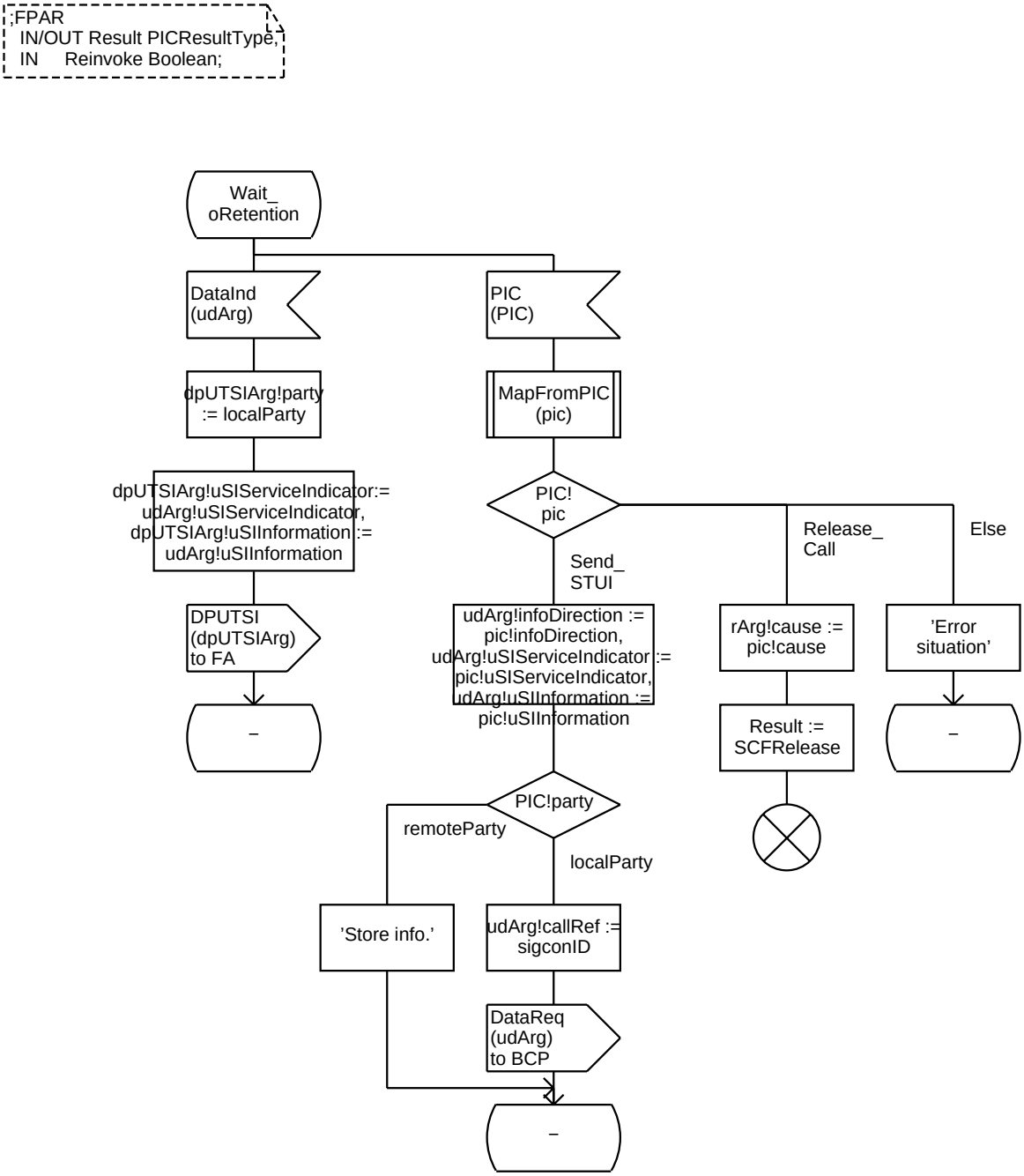
Procedure PIC_O_Suspended 3(3)



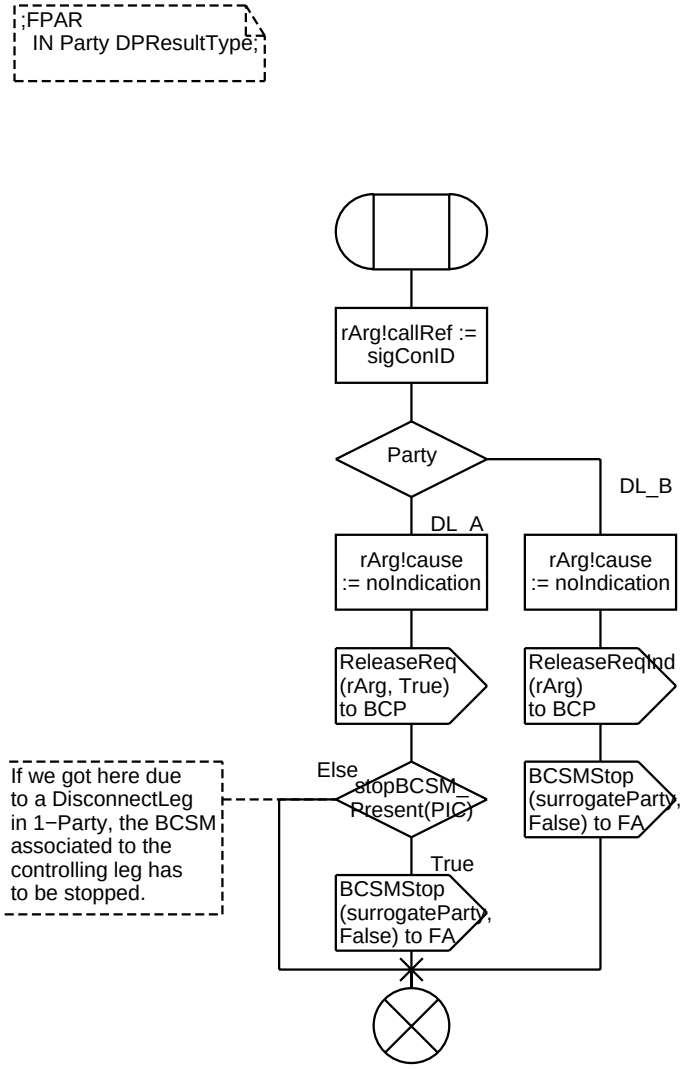
Procedure PIC_O_Retention 1(2)



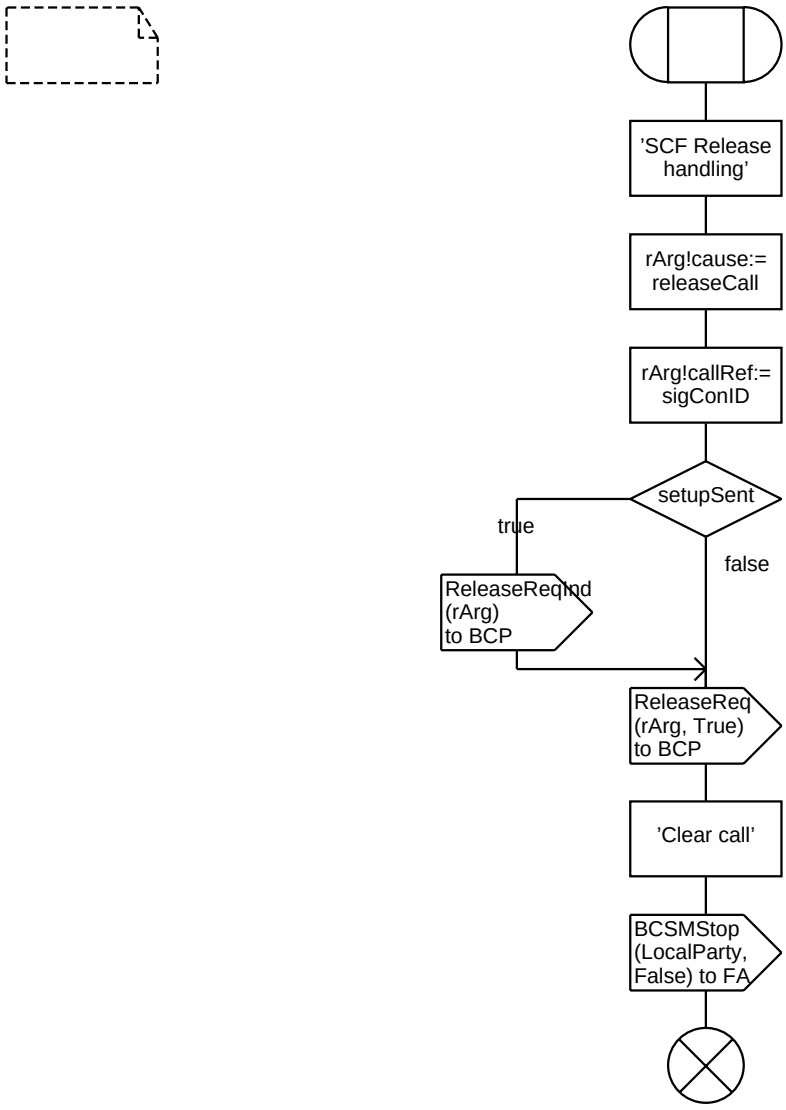
Procedure PIC_O_Retention 2(2)



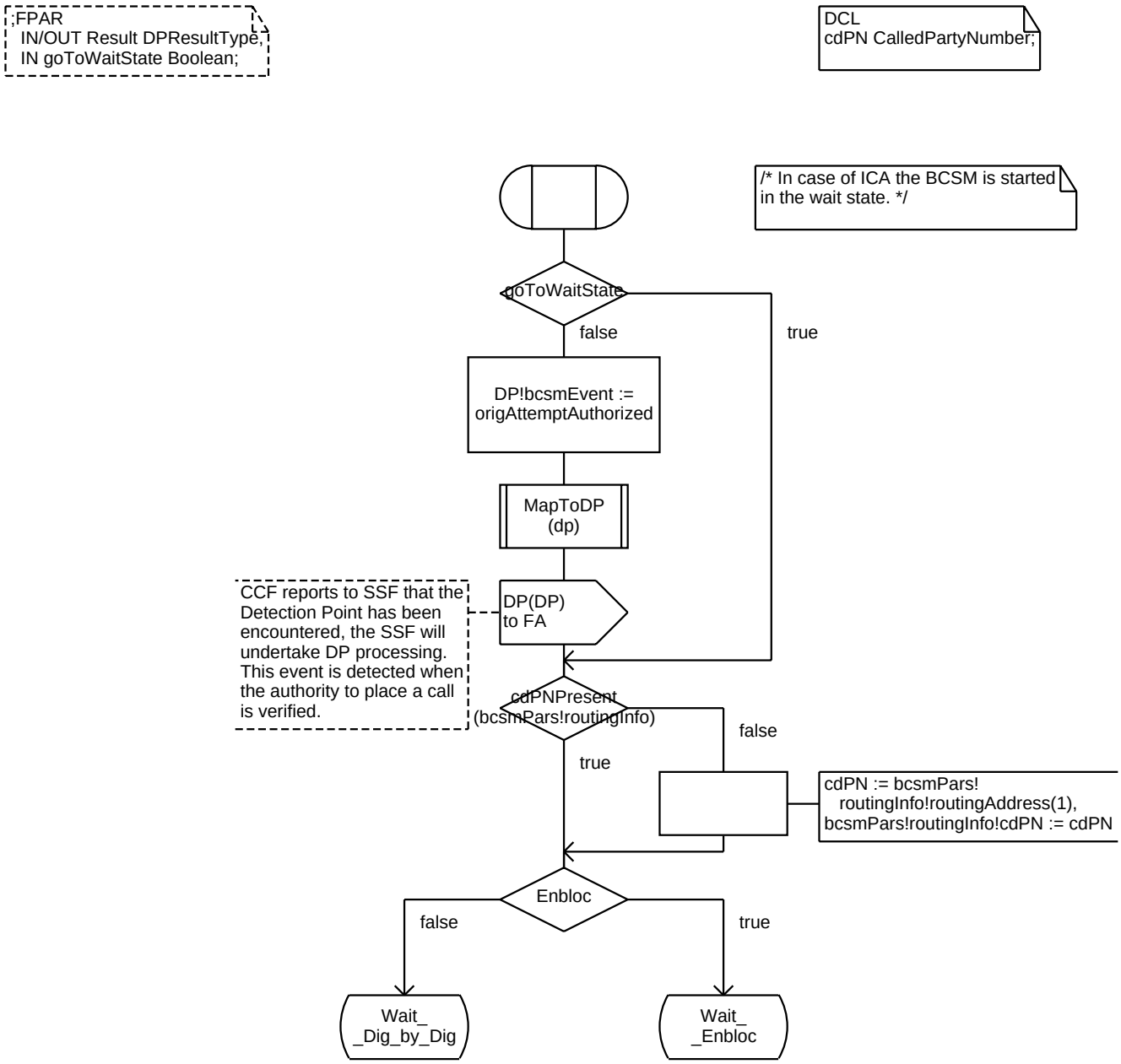
Procedure <<Process Type OriginatingBCSM>> PIC_Disconnect 1(1)



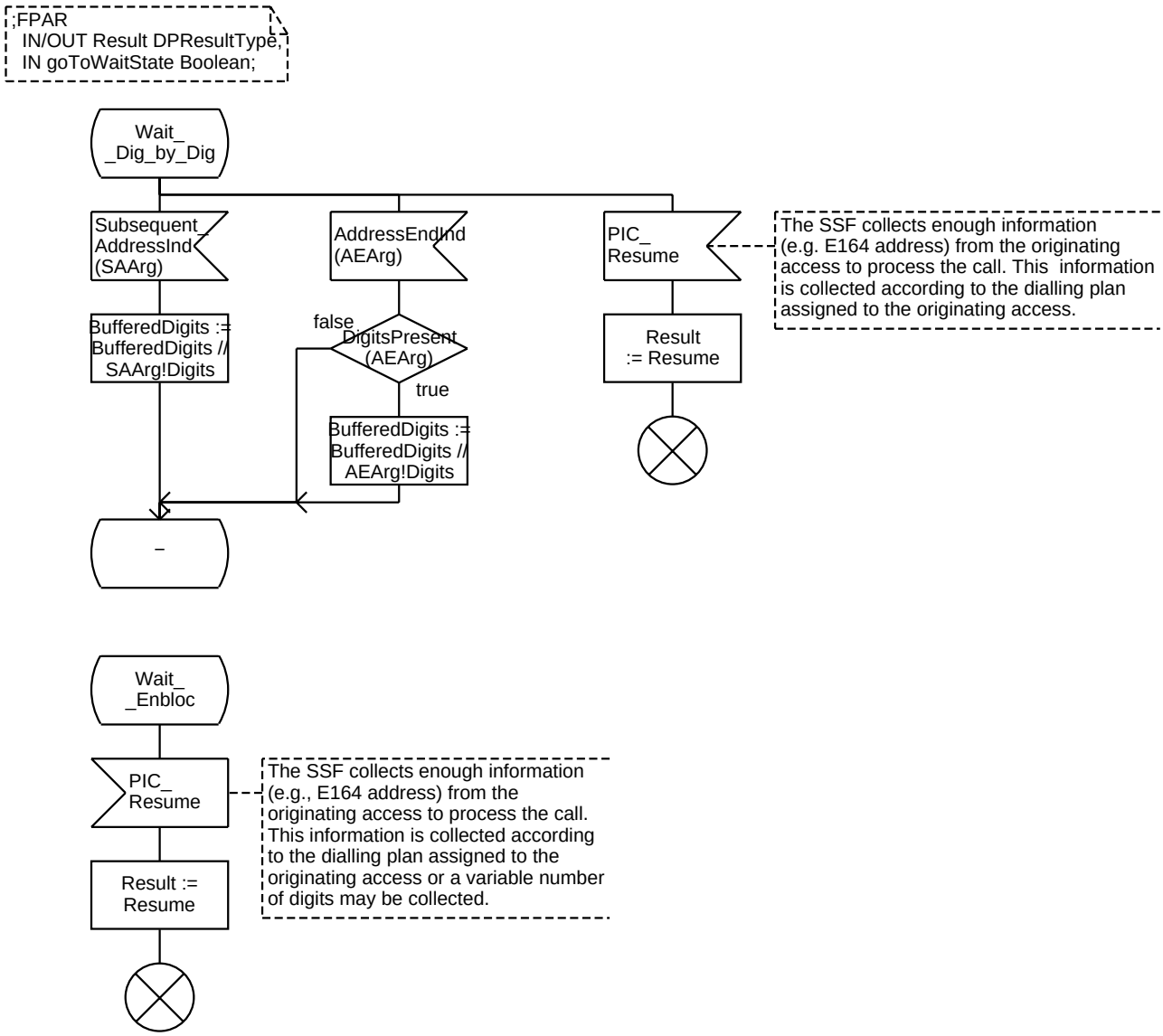
Procedure PIC_SCFRelease 1(1)



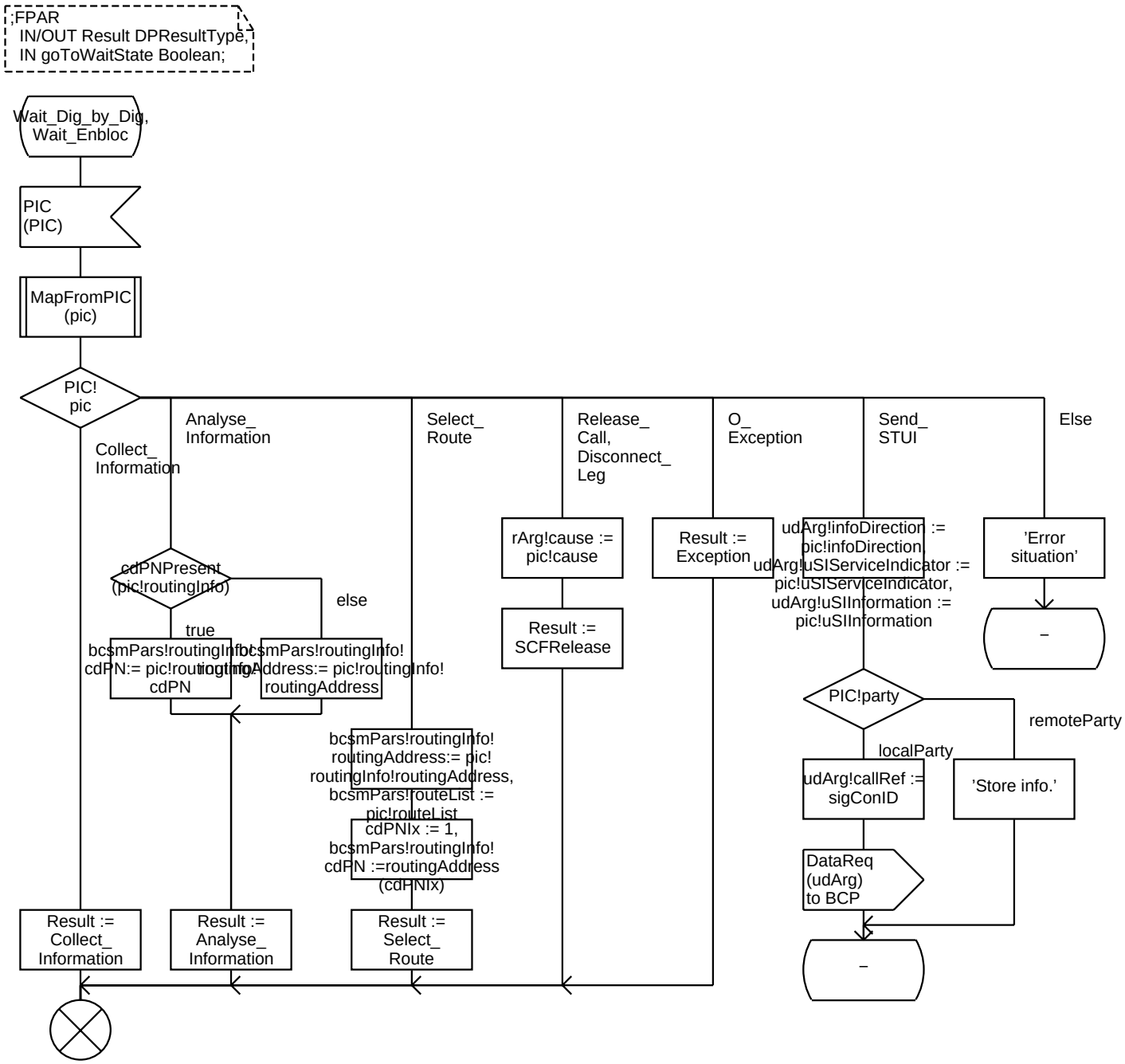
Procedure DP_origAttemptAuthorised 1(4)



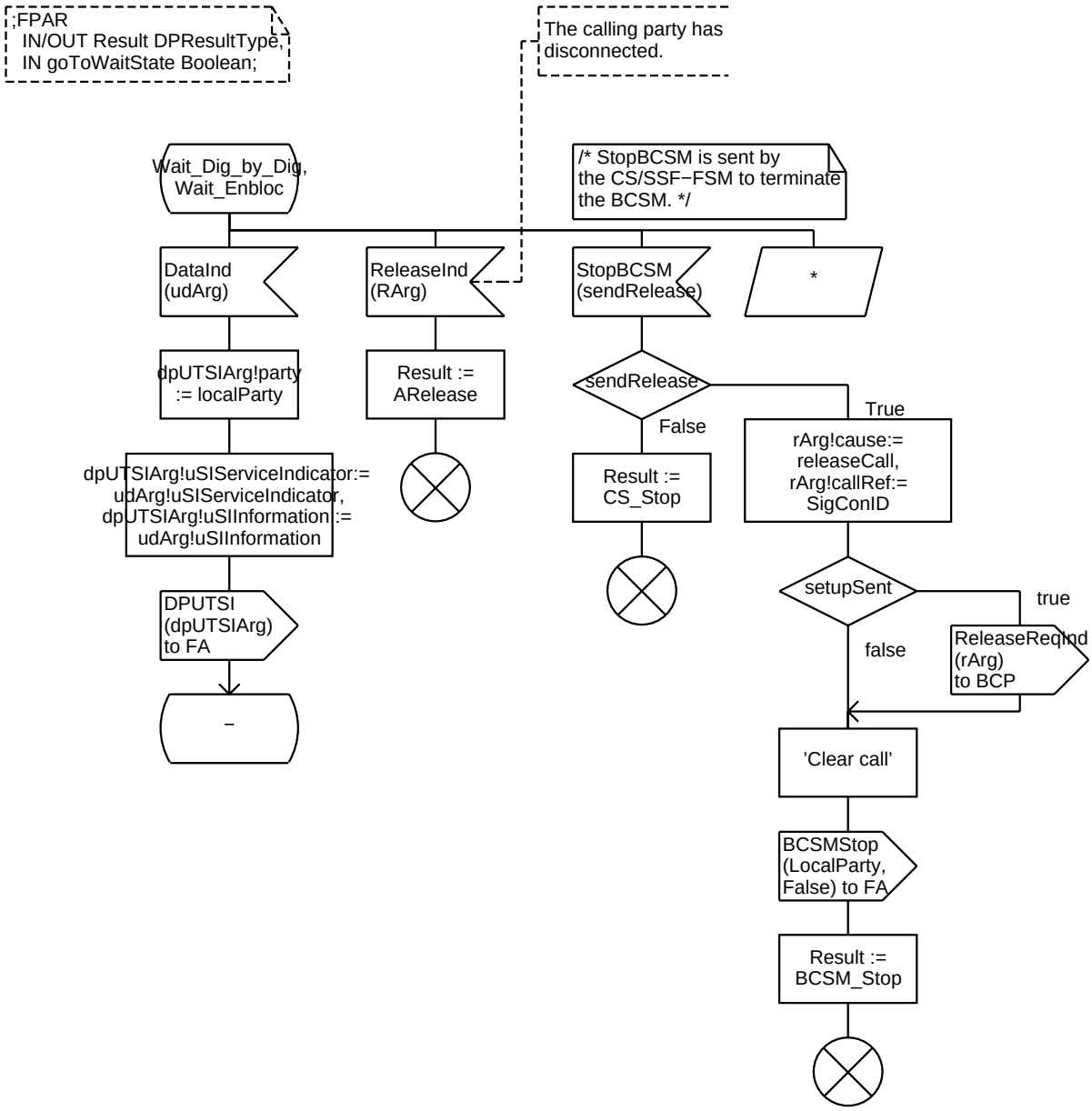
Procedure DP_origAttemptAuthorised 2(4)



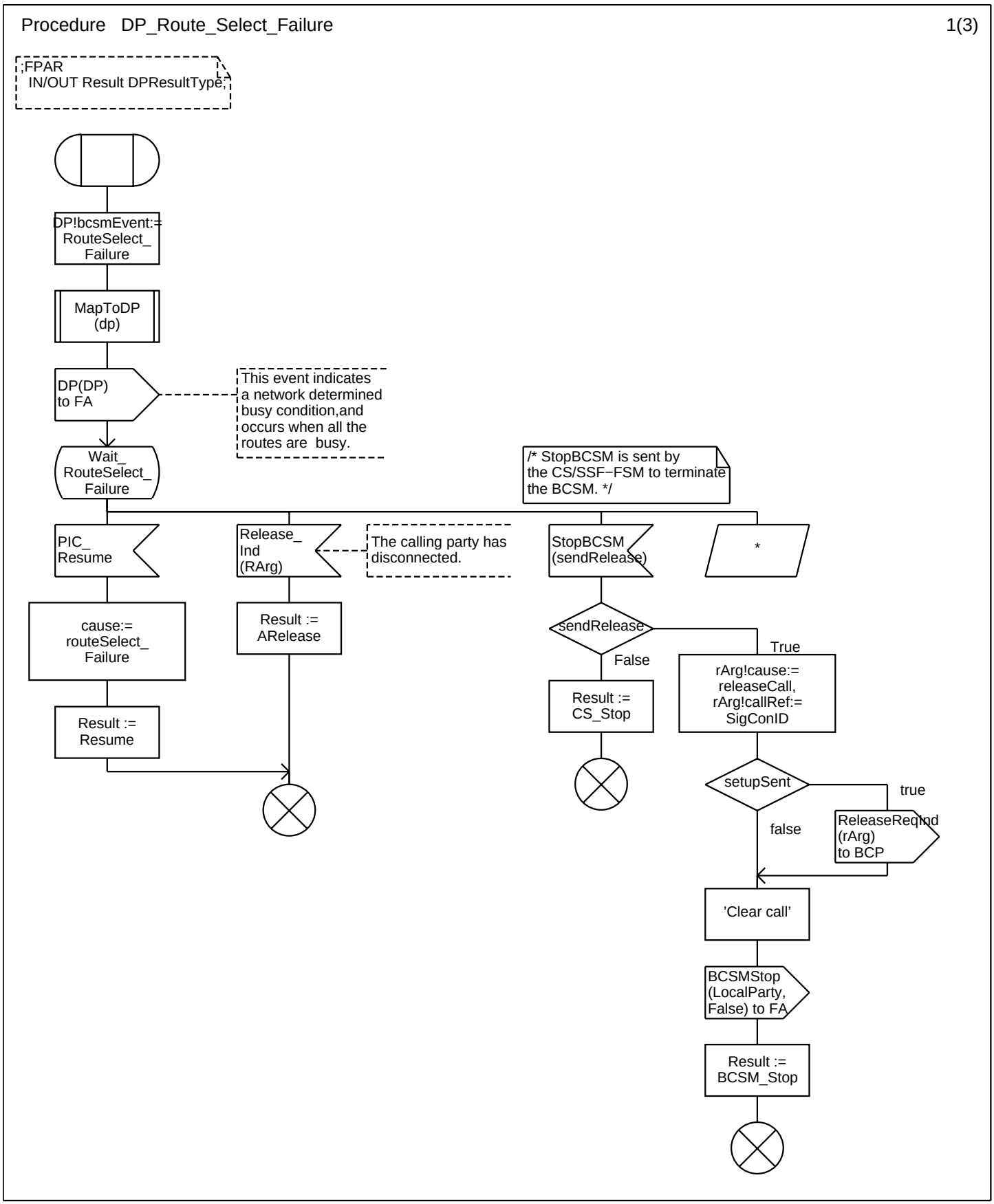
Procedure DP_origAttemptAuthorised 3(4)



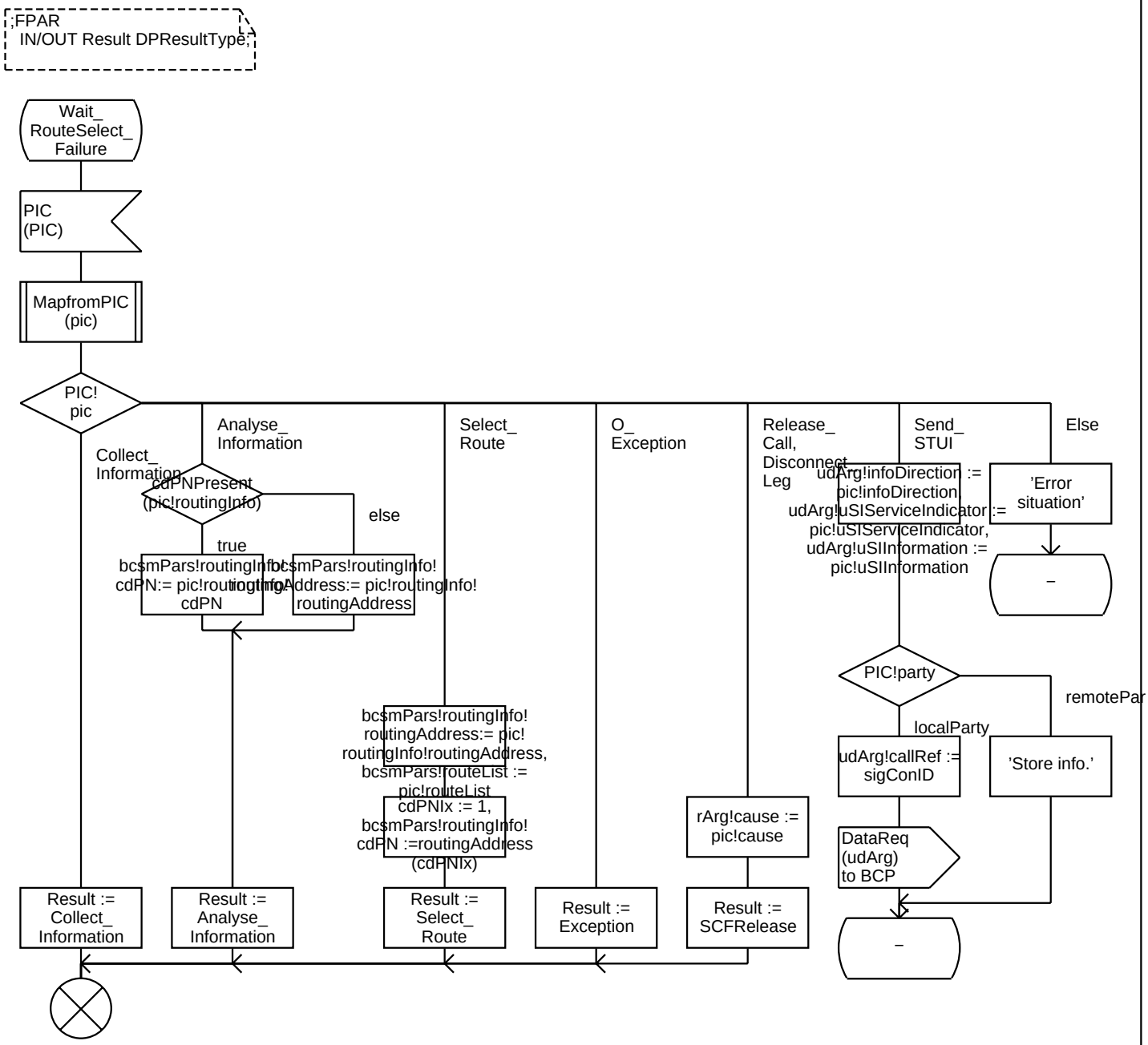
Procedure DP_origAttemptAuthorised 4(4)



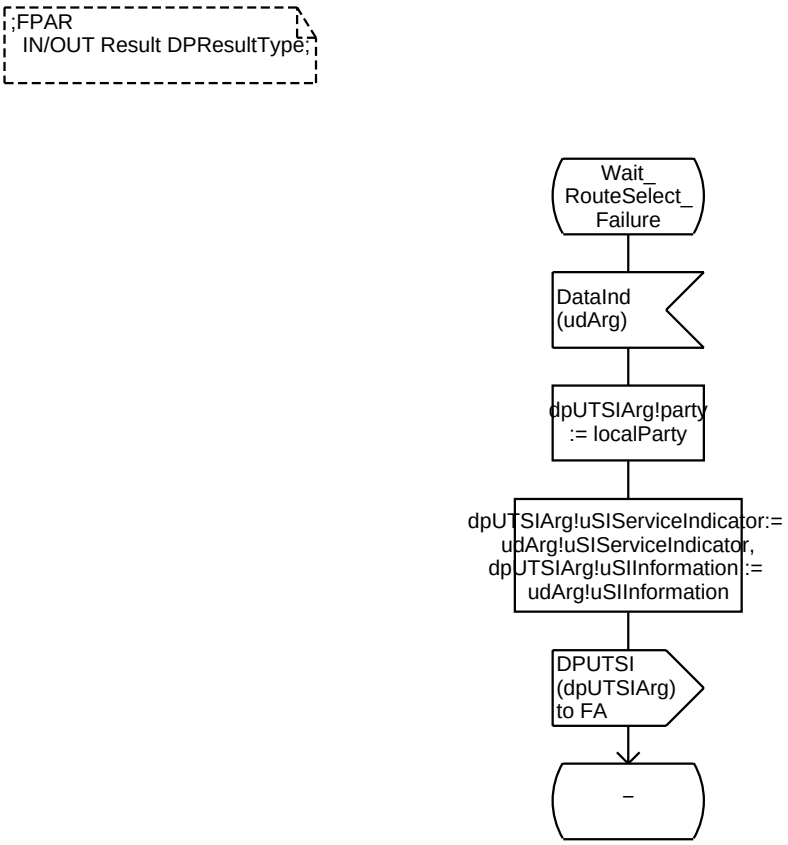
Procedure DP_Route_Select_Failure 1(3)

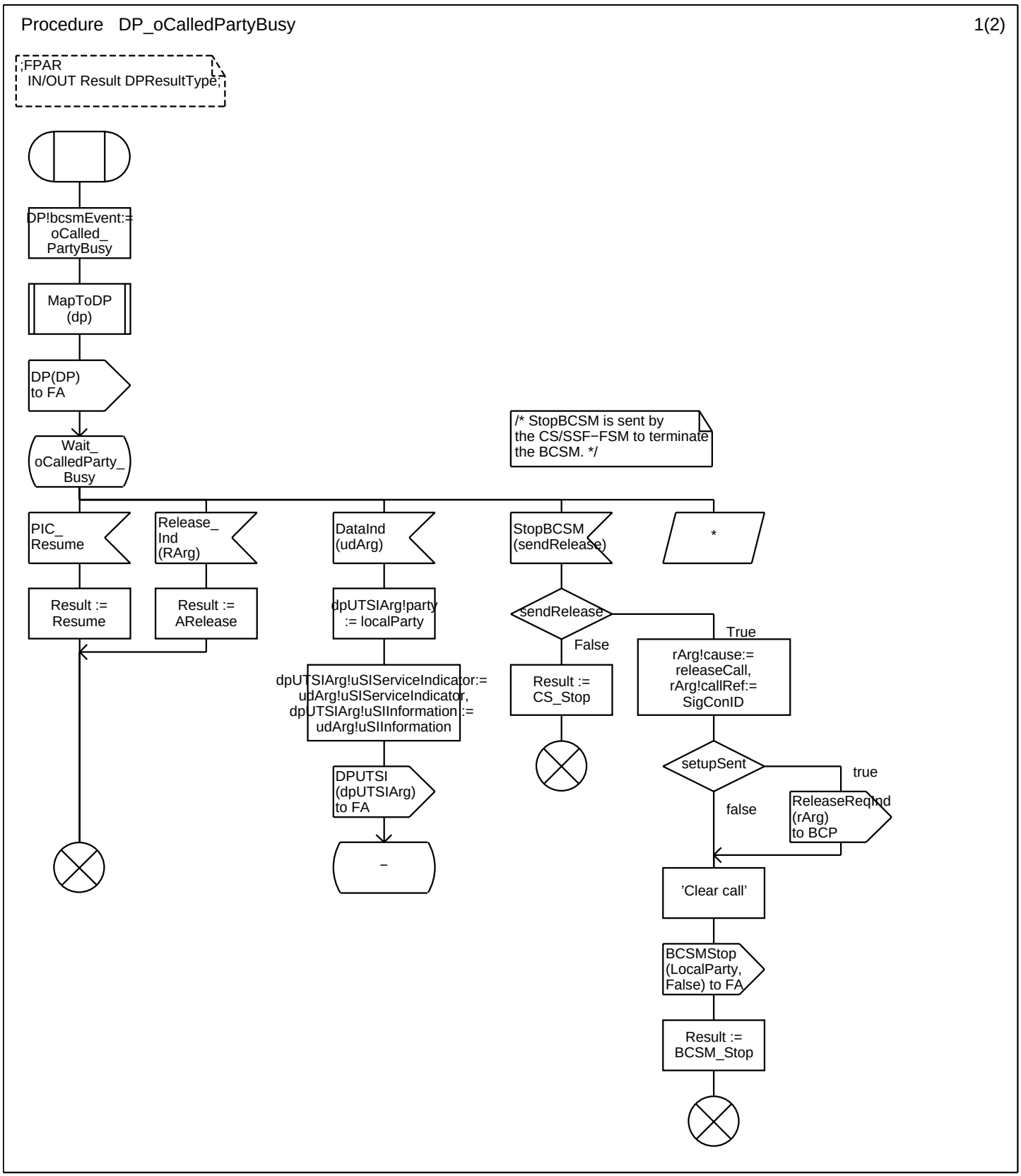


Procedure DP_Route_Select_Failure 2(3)

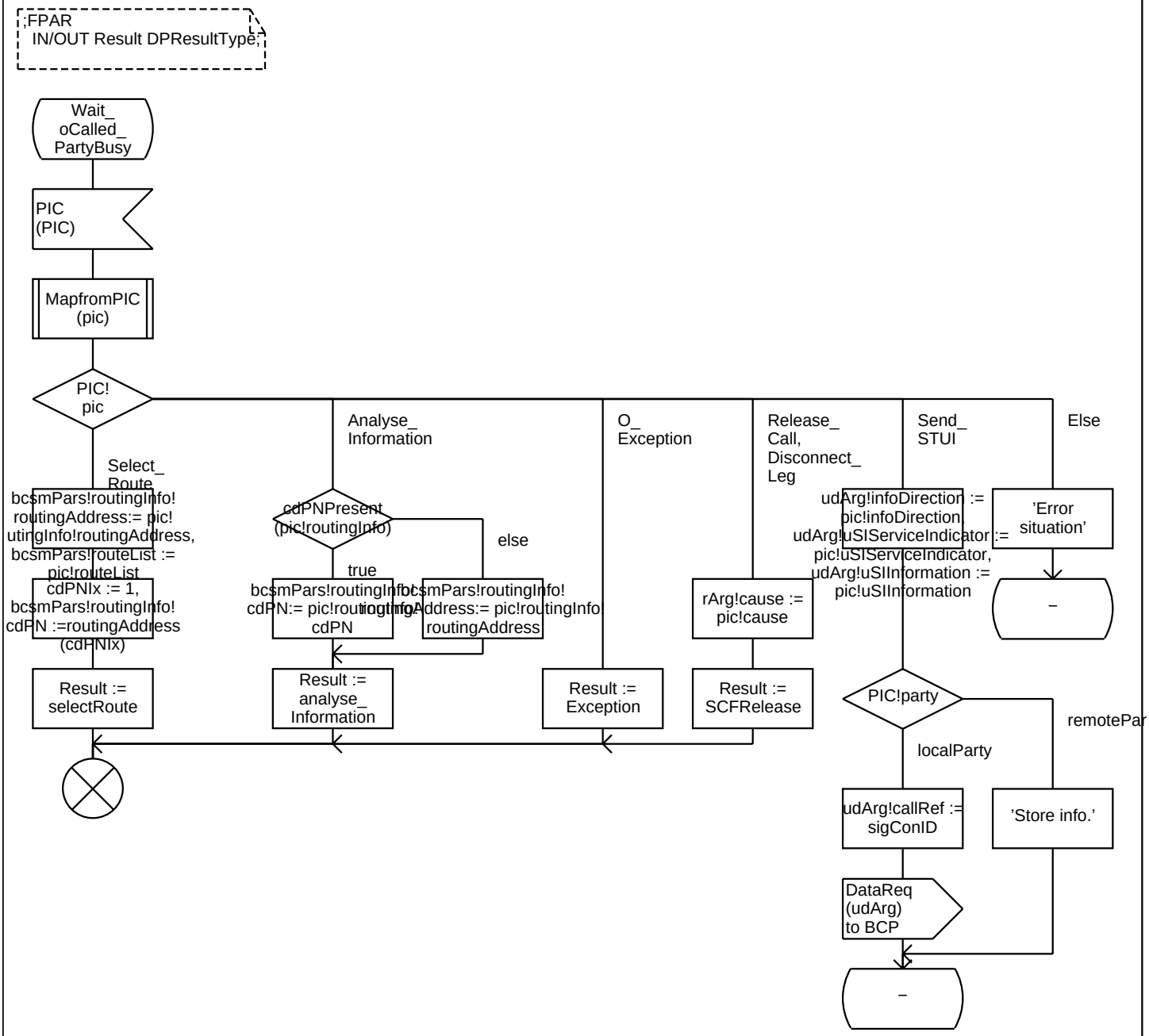


Procedure DP_Route_Select_Failure 3(3)

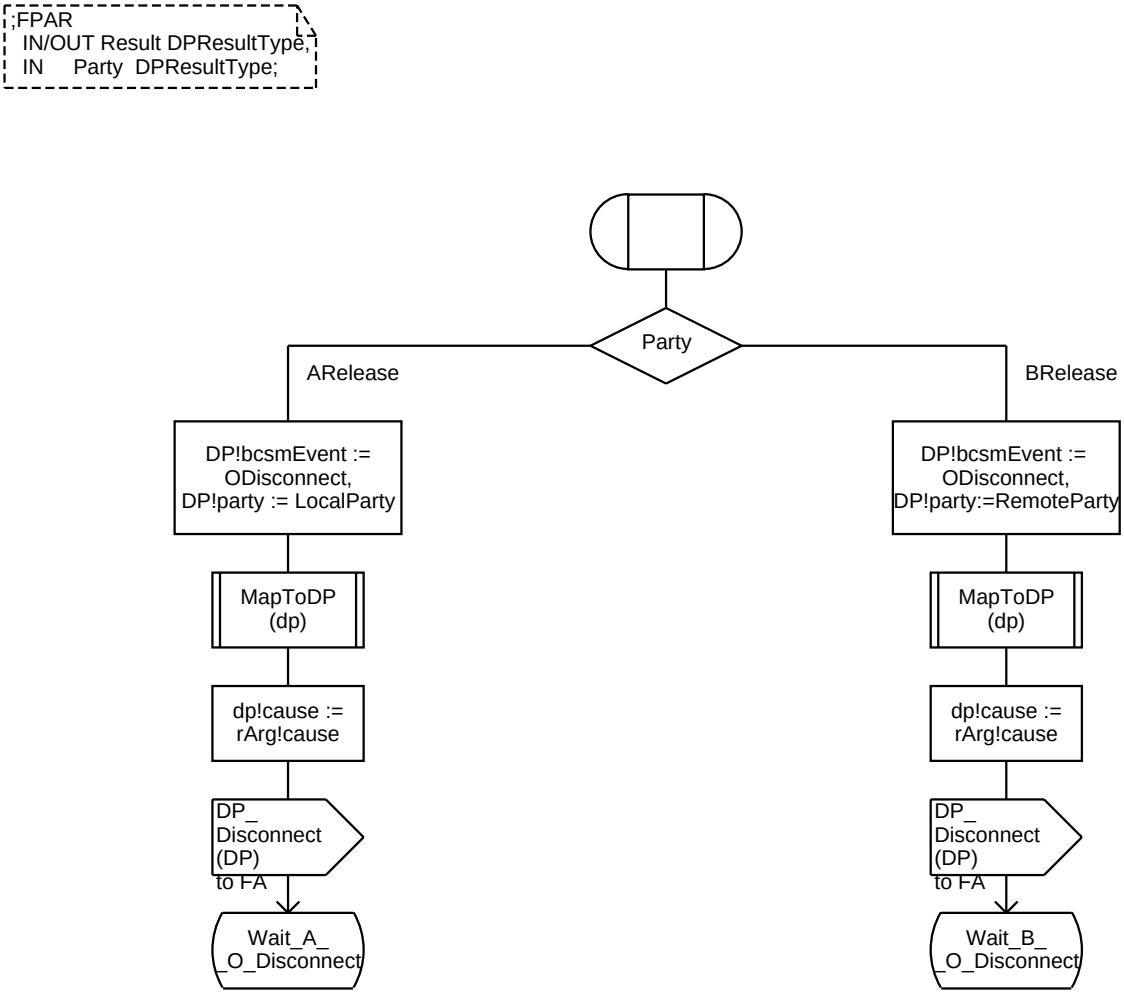



Procedure DP_oCalledPartyBusy
1(2)
;FPAR
IN/OUT Result DPResultType;
DP!bcsmEvent:=
oCalled_
PartyBusy
MapToDP
(dp)
DP(DP)
to FA
Wait_
oCalledPartyBusy
/* StopBCSM is sent by
the CS/SSF−FSM to terminate
the BCSM. */
PIC_
Resume
Release_
Ind
(RArg)
DataInd
(udArg)
StopBCSM
(sendRelease)
*
Result :=
Resume
Result :=
ARelease
dpUTSIArg!party
:= localParty
sendRelease
True
False
rArg!cause:=
releaseCall,
rArg!callRef:=
SigConID
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation:=
udArg!uSIInformation
Result :=
CS_Stop
setupSent
true
false
ReleaseReqInd
(rArg)
to BCP
DPUTSI
(dpUTSIArg)
to FA
–
'Clear call'
BCSMStop
(LocalParty,
False) to FA
Result :=
BCSM_Stop

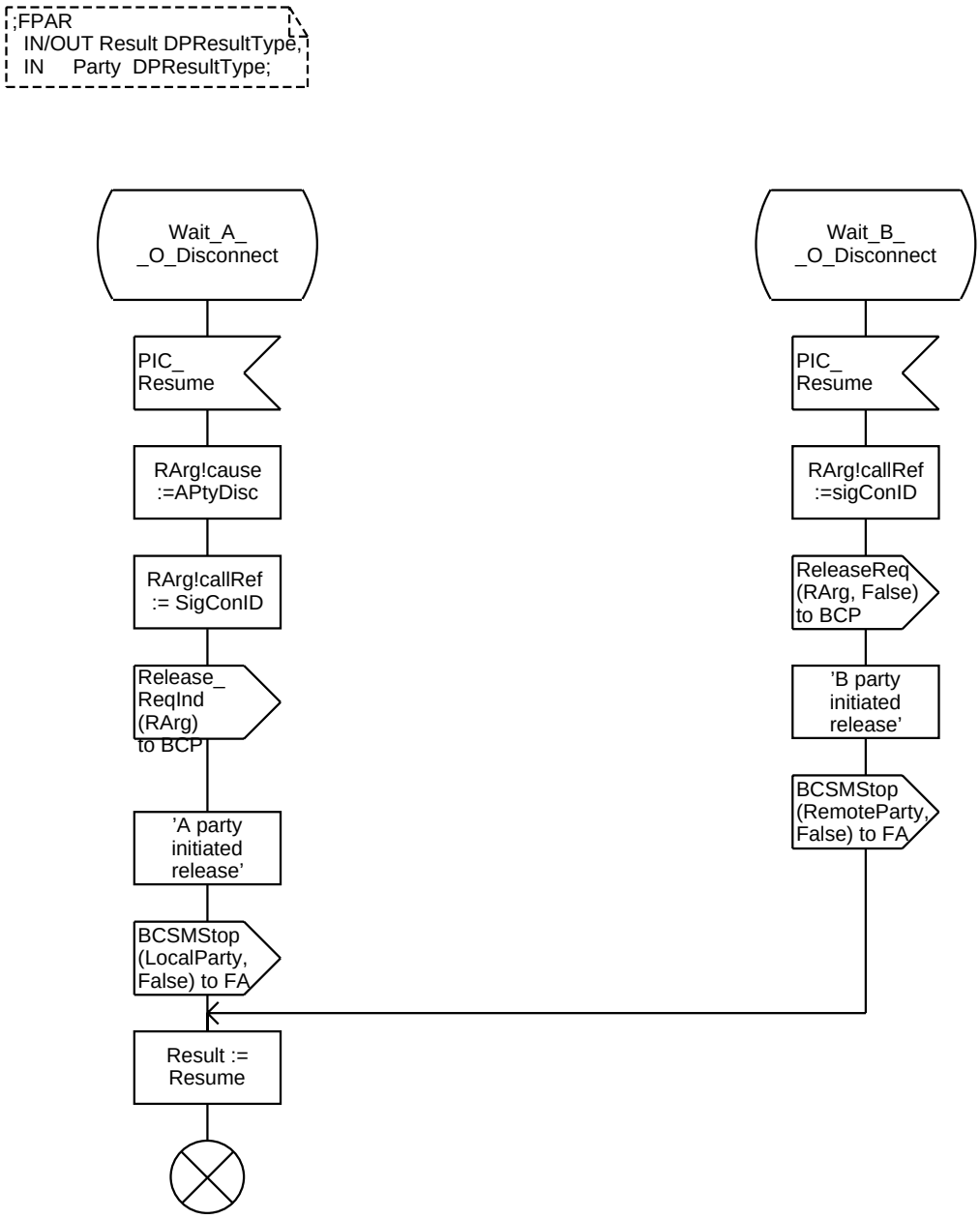
Procedure DP_oCalledPartyBusy 2(2)



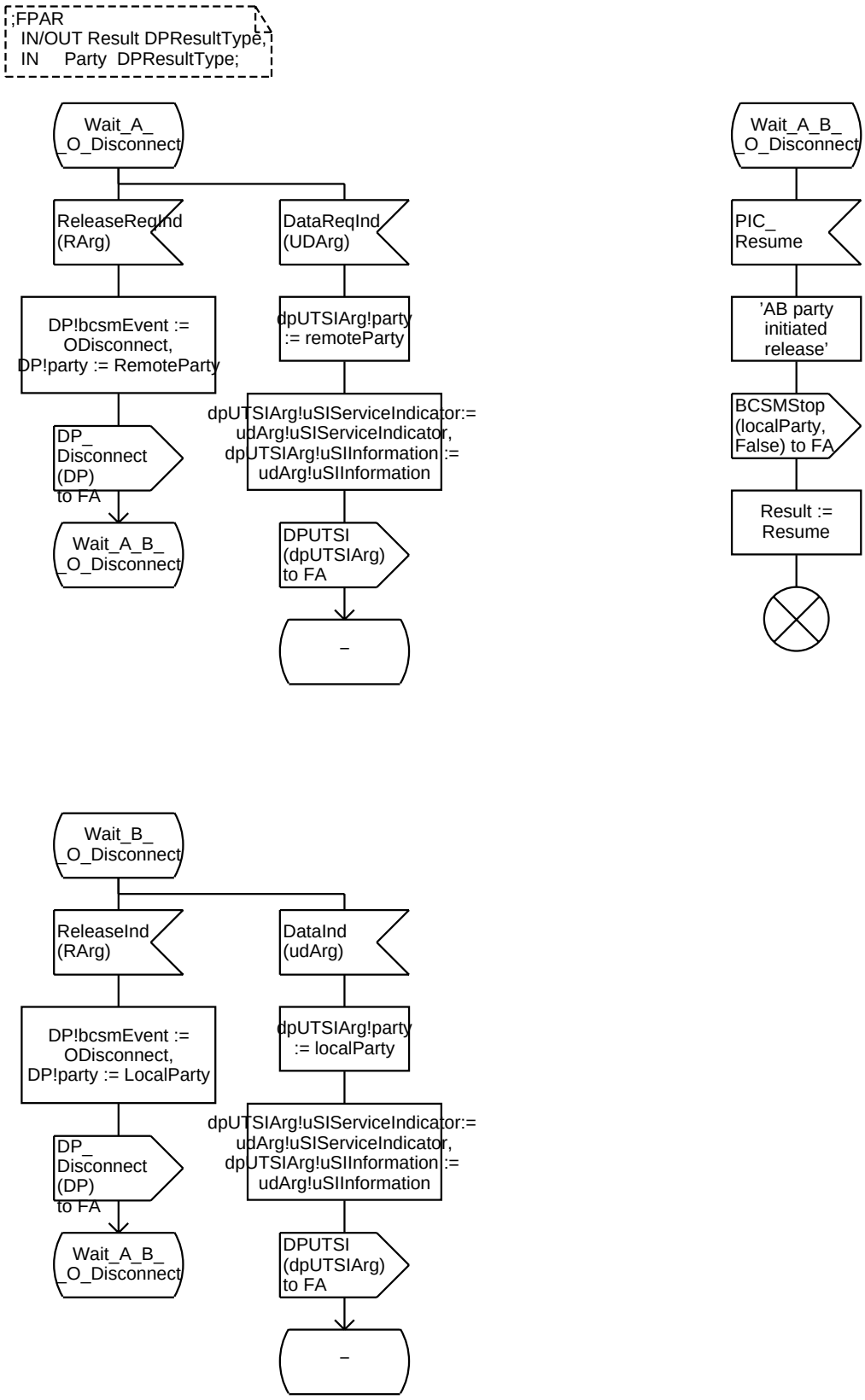
Procedure DP_oDisconnect 1(6)

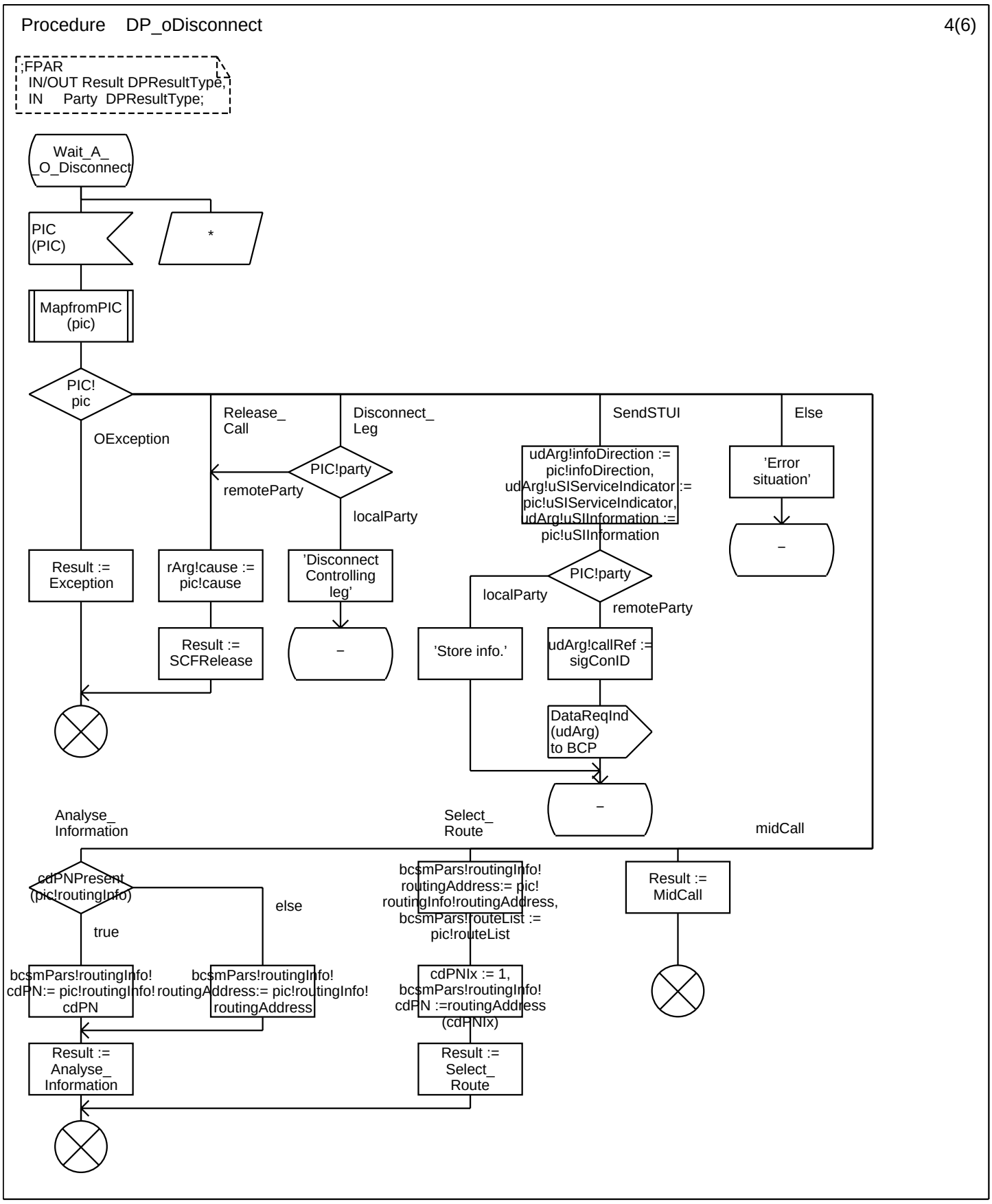


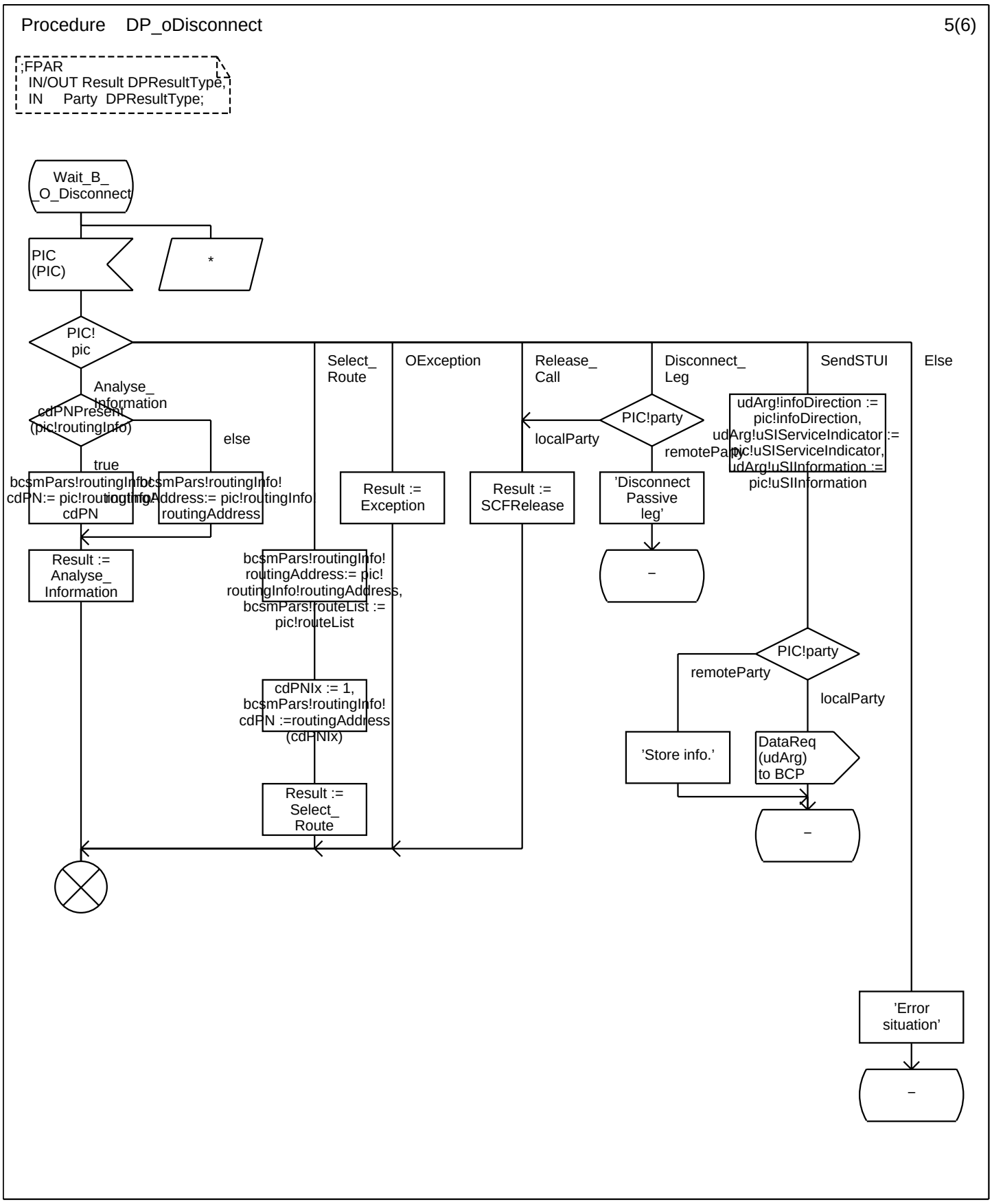
Procedure DP_oDisconnect 2(6)



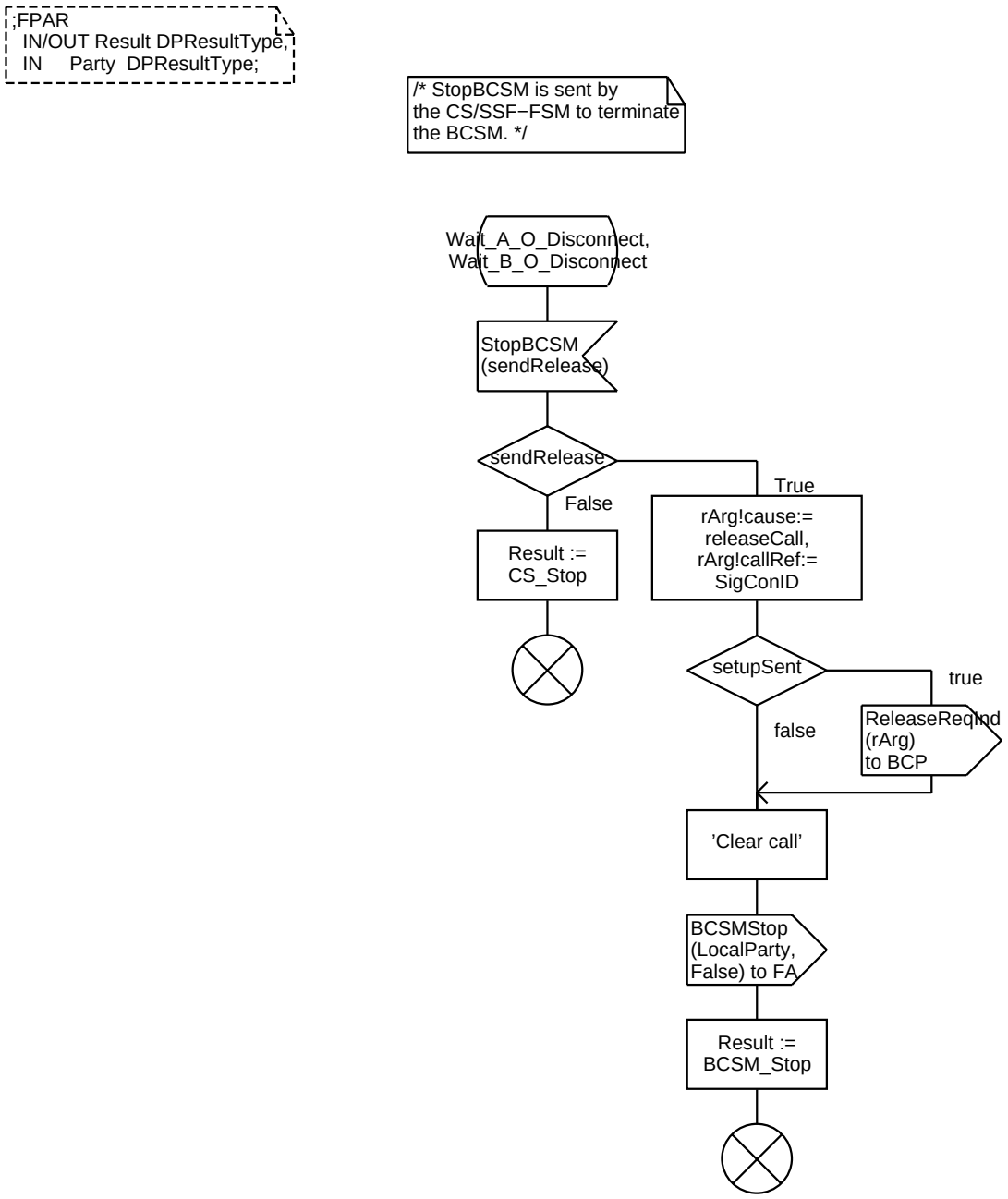
Procedure DP_oDisconnect 3(6)



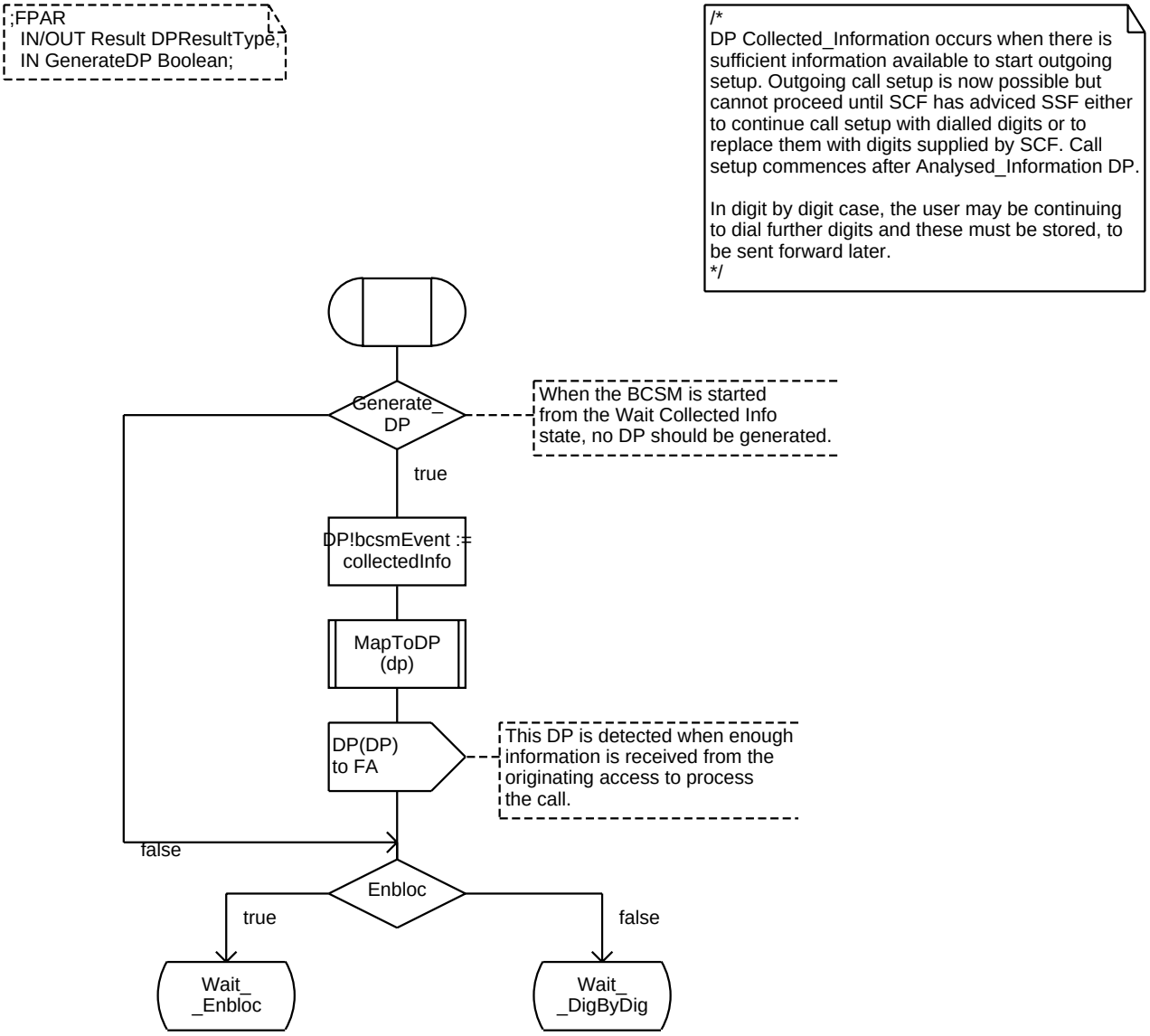

Procedure DP_oDisconnect
4(6)
;FPAR
IN/OUT Result DPResultType,
IN Party DPResultType;
Wait_A_
_O_Disconnect
PIC
(PIC)
*
MapfromPIC
(pic)
PIC!
pic
OException
Release_
Call
Disconnect_
Leg
SendSTUI
Else
PIC!party
remoteParty
localParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
–
Result :=
Exception
rArg!cause :=
pic!cause
'Disconnect
Controlling
leg'
PIC!party
localParty
remoteParty
Result :=
SCFRelease
–
'Store info.'
udArg!callRef :=
sigConID
DataReqInd
(udArg)
to BCP
–
Analyse_
Information
Select_
Route
midCall
cdPNPresent
(pic!routingInfo)
else
true
bcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
bcsmPars!routeList :=
pic!routeList
Result :=
MidCall
bcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
bcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
cdPNIx := 1,
bcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Analyse_
Information
Result :=
Select_
Route


Procedure DP_oDisconnect
5(6)
;FPAR
IN/OUT Result DPResultType,
IN Party DPResultType;
Wait_B_
_O_Disconnect
PIC
(PIC)
*
PIC!
pic
Select_
Route
OException
Release_
Call
Disconnect_
Leg
SendSTUI
Else
Analyse_
Information
cdPNPresent
(pic!routingInfo)
else
true
bcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
bcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Analyse_
Information
bcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
bcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
bcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
Result :=
Exception
Result :=
SCFRelease
PIC!party
localParty
remoteParty
'Disconnect
Passive
leg'
–
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
remoteParty
localParty
'Store info.'
DataReq
(udArg)
to BCP
–
'Error
situation'
–

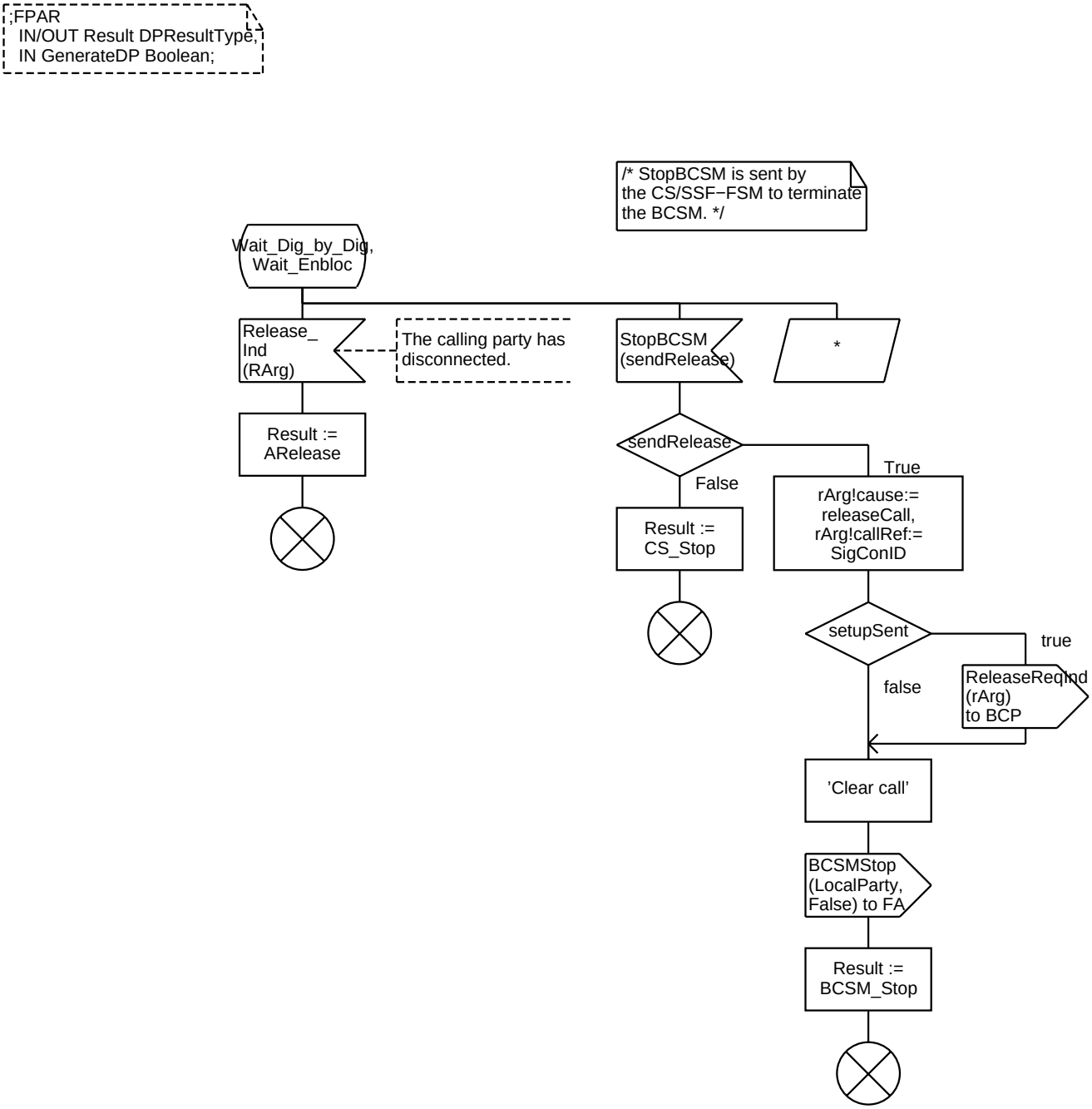
Procedure DP_oDisconnect 6(6)



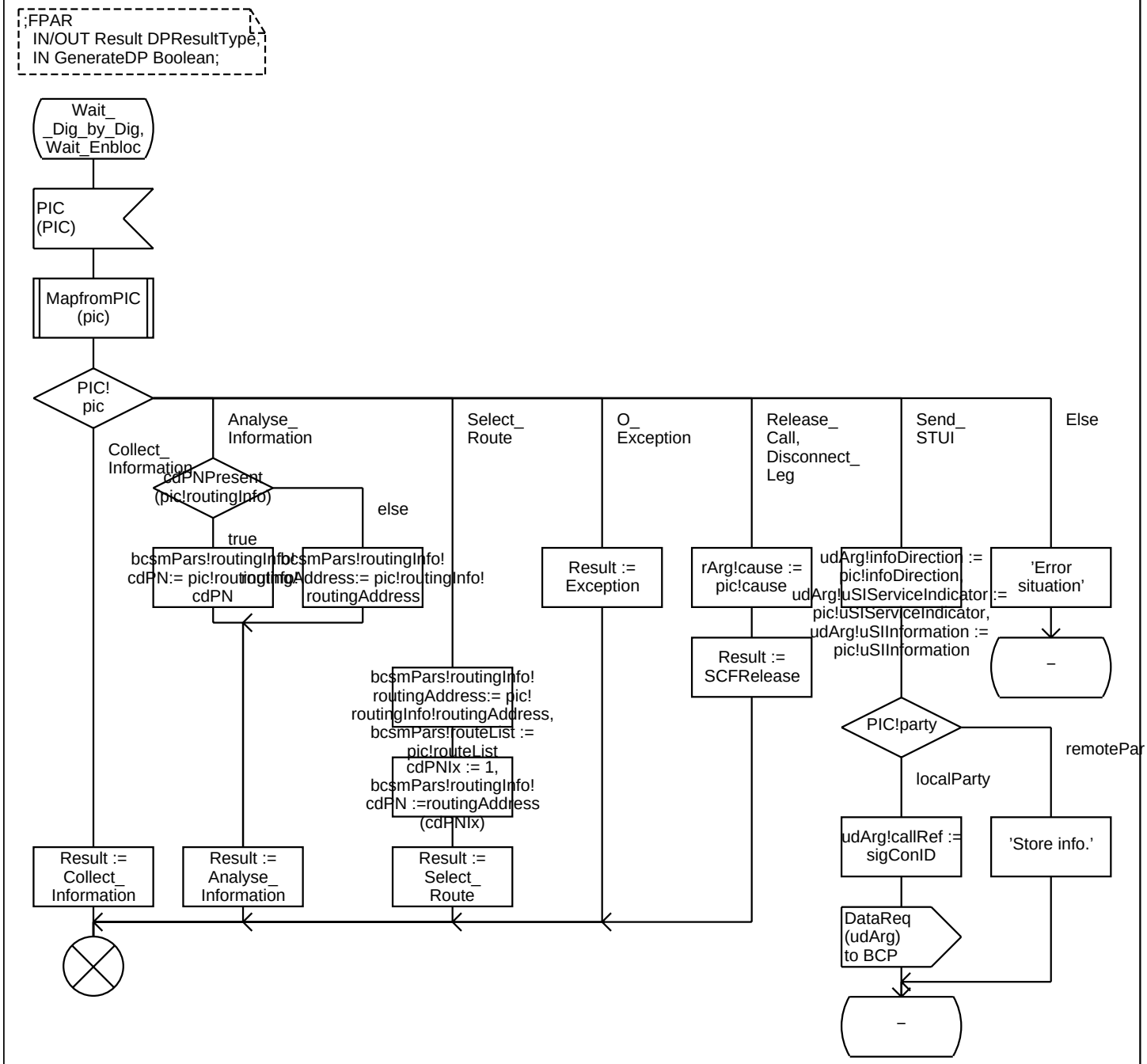
Procedure DP_Collected_Info 1(4)



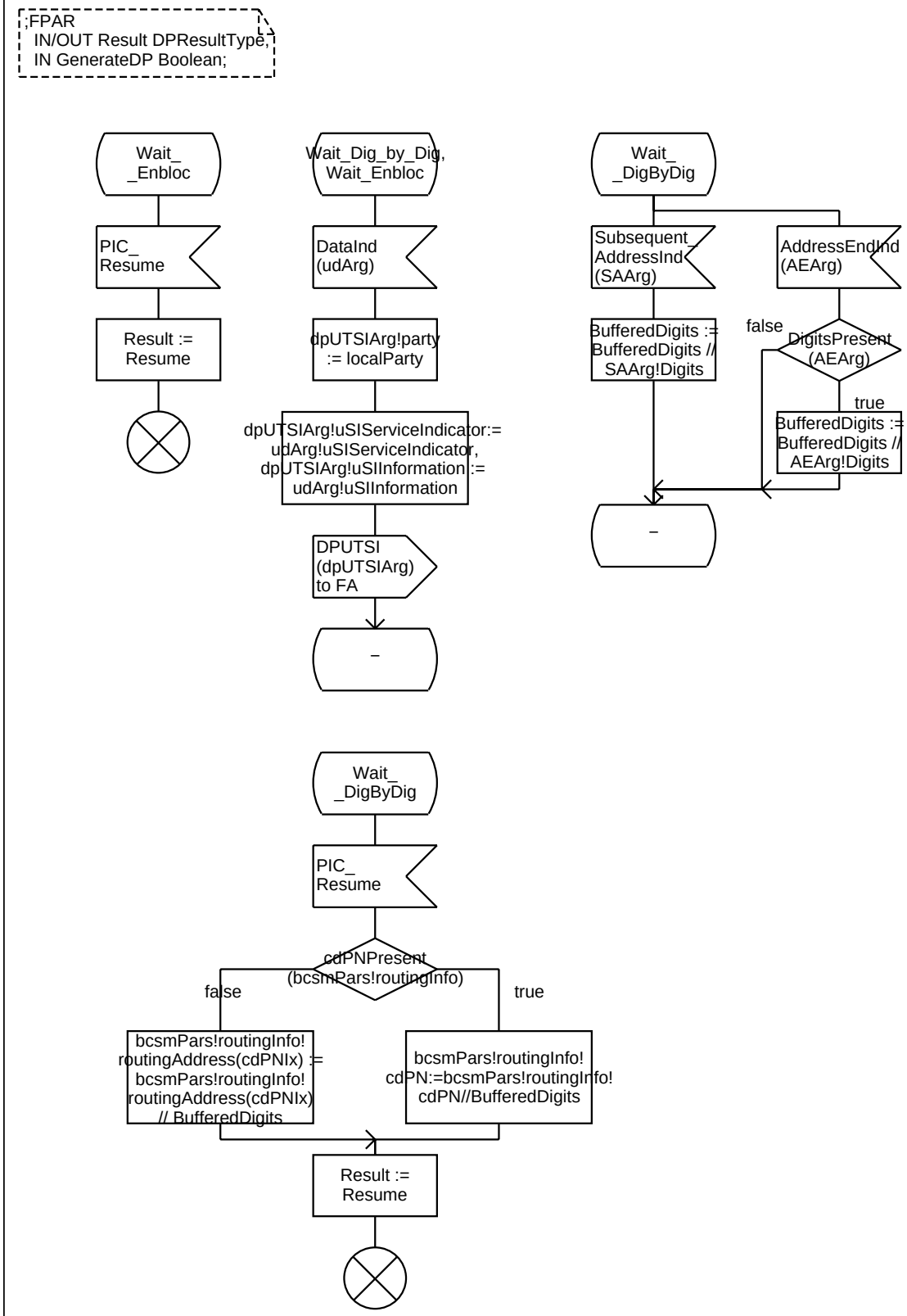
Procedure DP_Collected_Info 2(4)



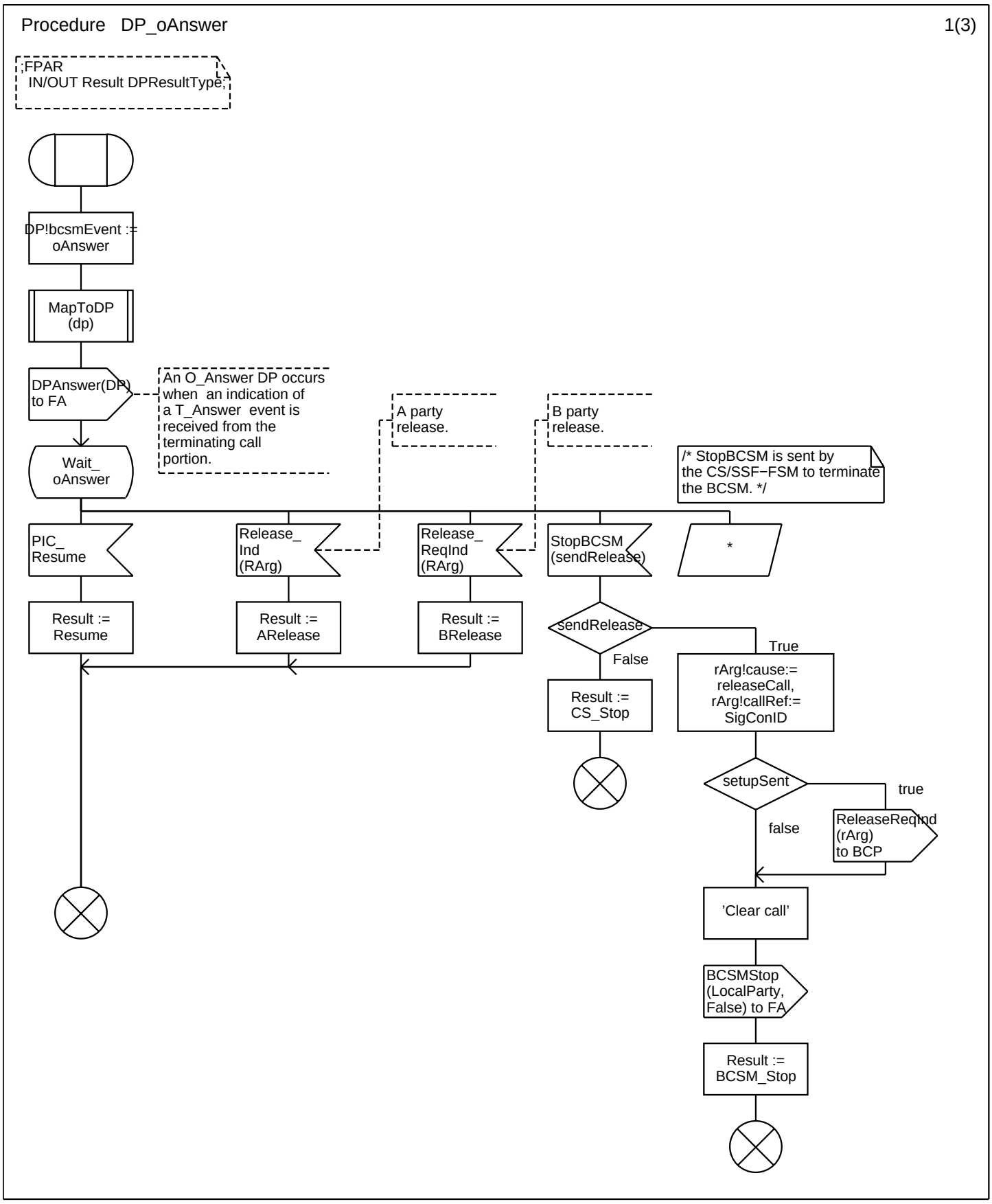
Procedure DP_Collected_Info 3(4)



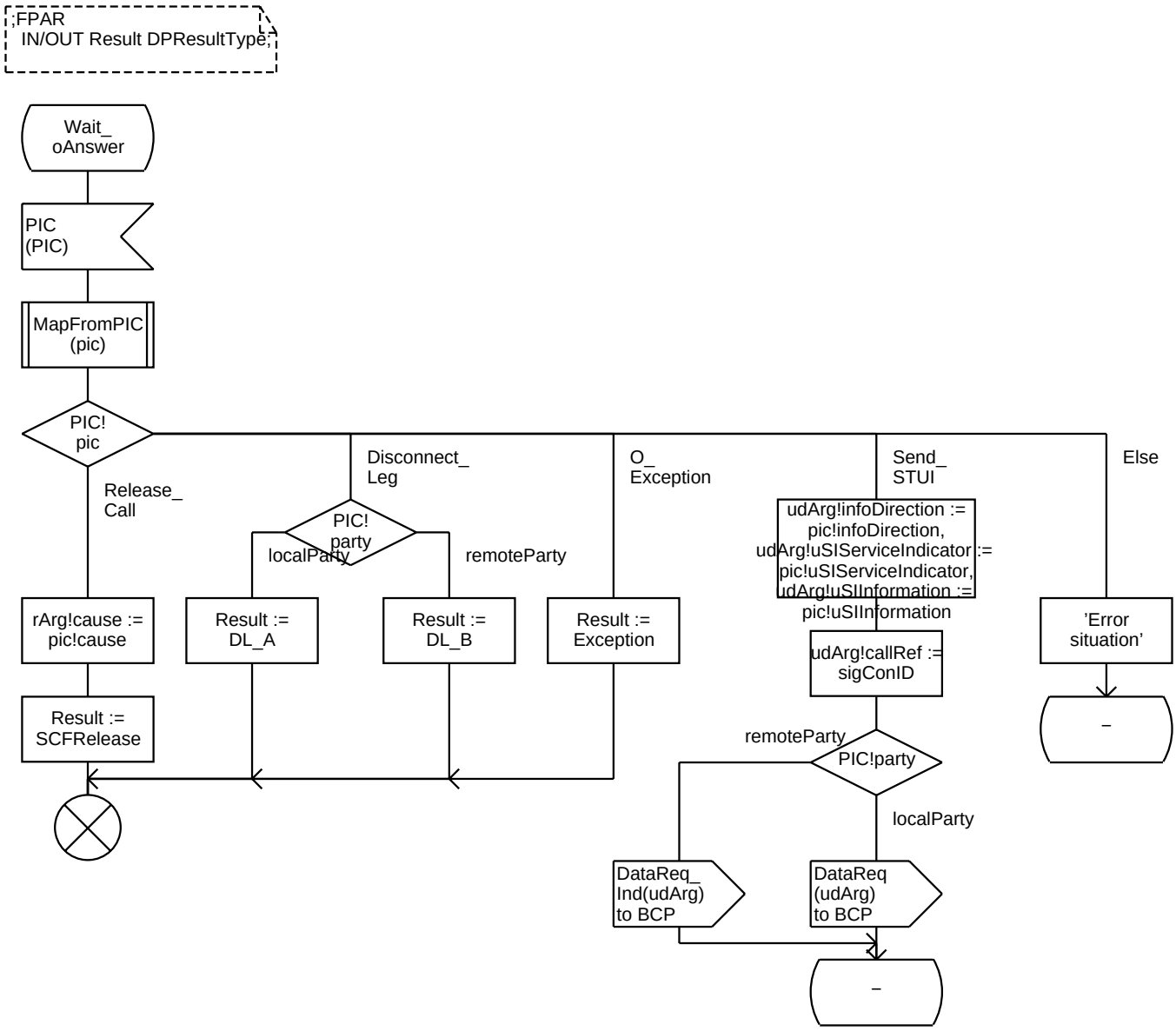
Procedure DP_Collected_Info 4(4)



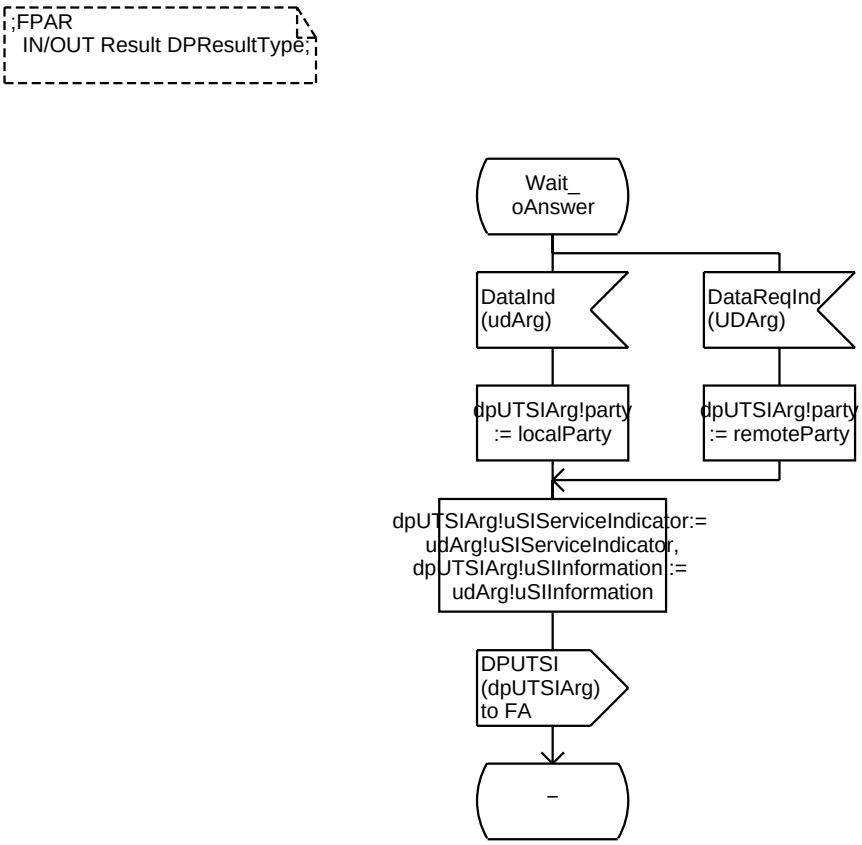
DP_oAnswer


Procedure DP_oAnswer
1(3)
;FPAR
IN/OUT Result DPResultType;
DP!bcsmEvent := oAnswer
MapToDP (dp)
DPAnswer(DP) to FA
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.
/* StopBCSM is sent by the CS/SSF−FSM to terminate the BCSM. */
Wait_ oAnswer
PIC_ Resume
Release_ Ind (RArg)
Release_ ReqInd (RArg)
StopBCSM (sendRelease)
*
Result := Resume
Result := ARelease
Result := BRelease
sendRelease
True
False
Result := CS_Stop
rArg!cause:= releaseCall, rArg!callRef:= SigConID
setupSent
true
false
ReleaseReqInd (rArg) to BCP
'Clear call'
BCSMStop (LocalParty, False) to FA
Result := BCSM_Stop

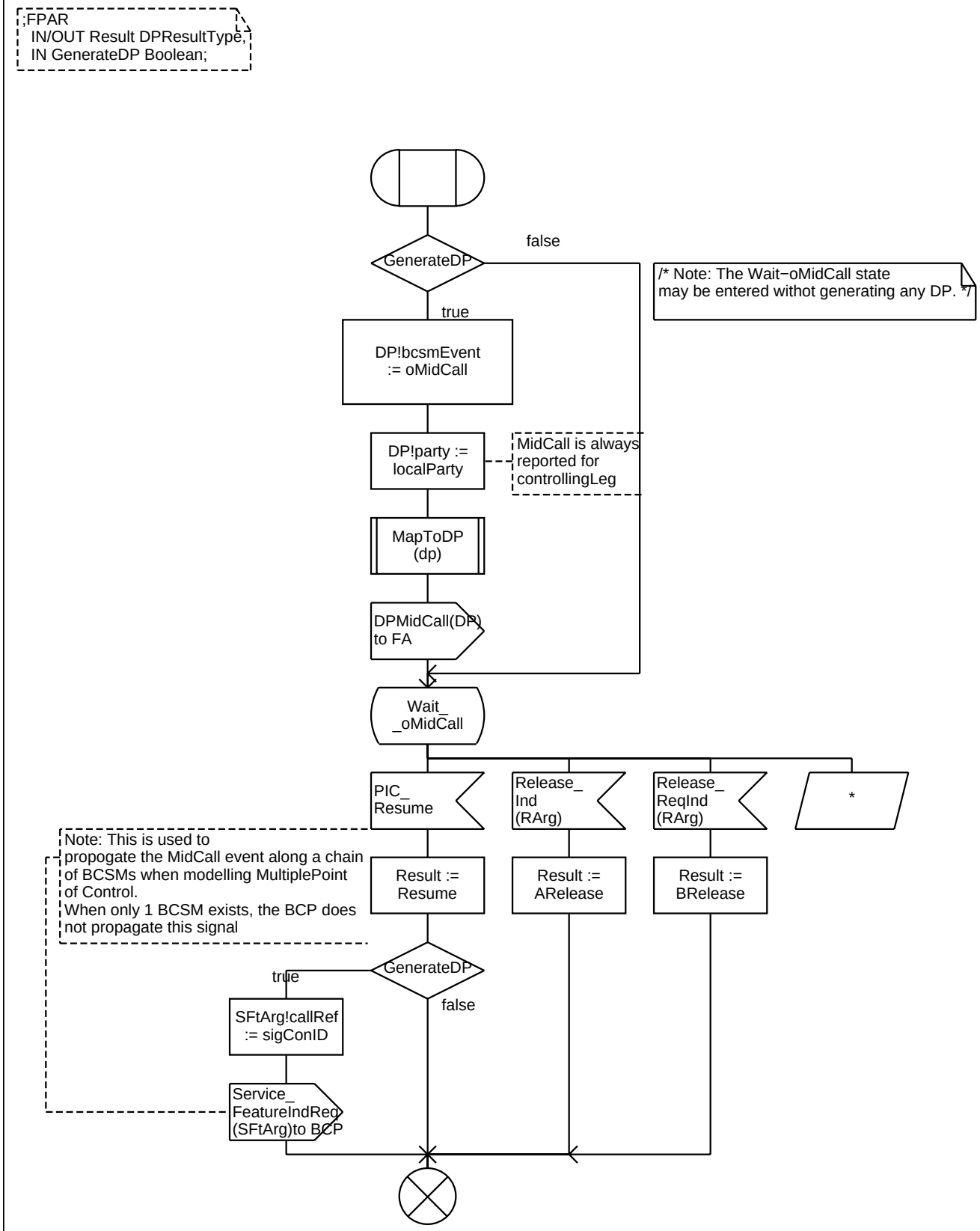
Procedure DP_oAnswer 2(3)



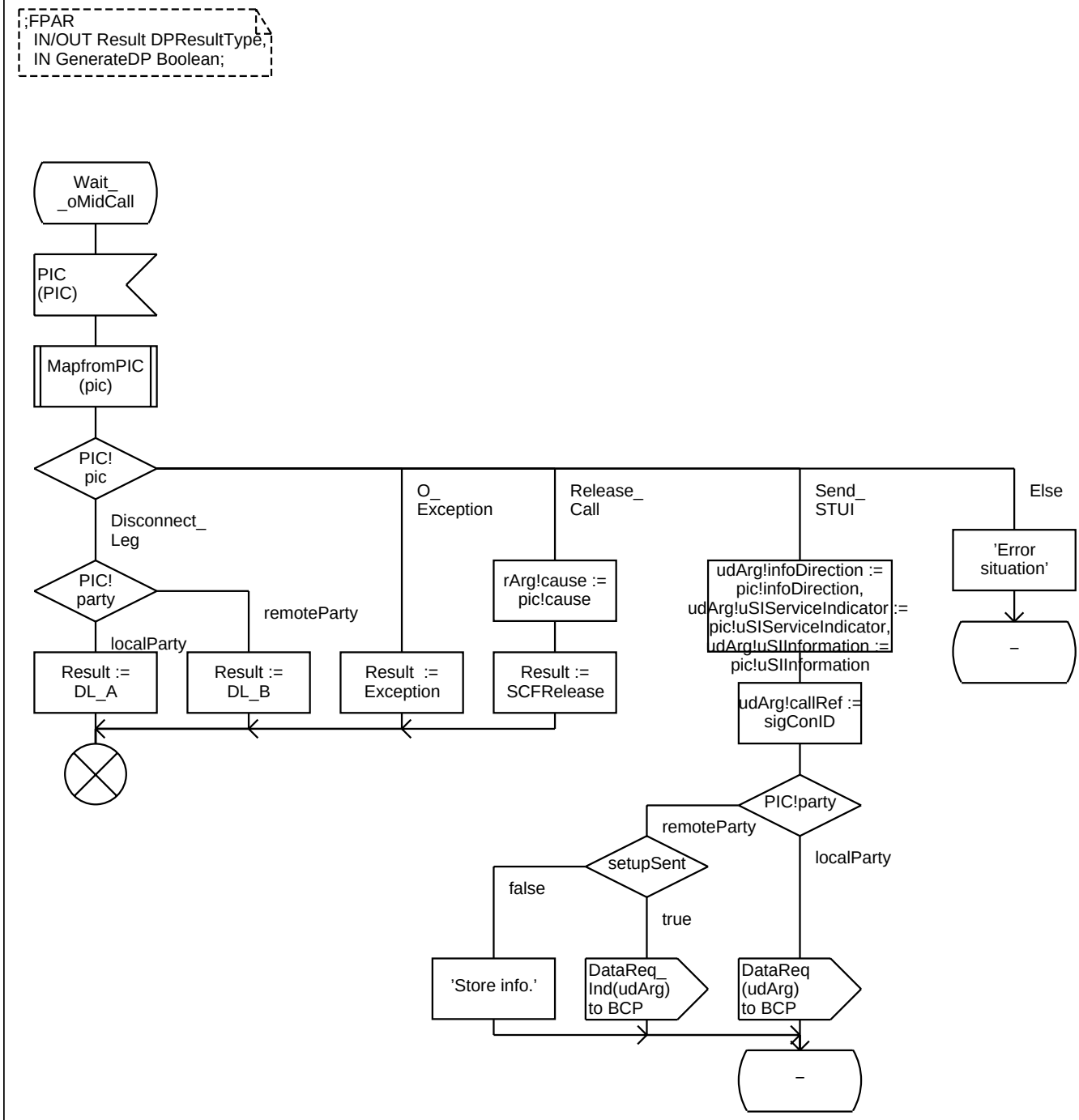
Procedure DP_oAnswer 3(3)



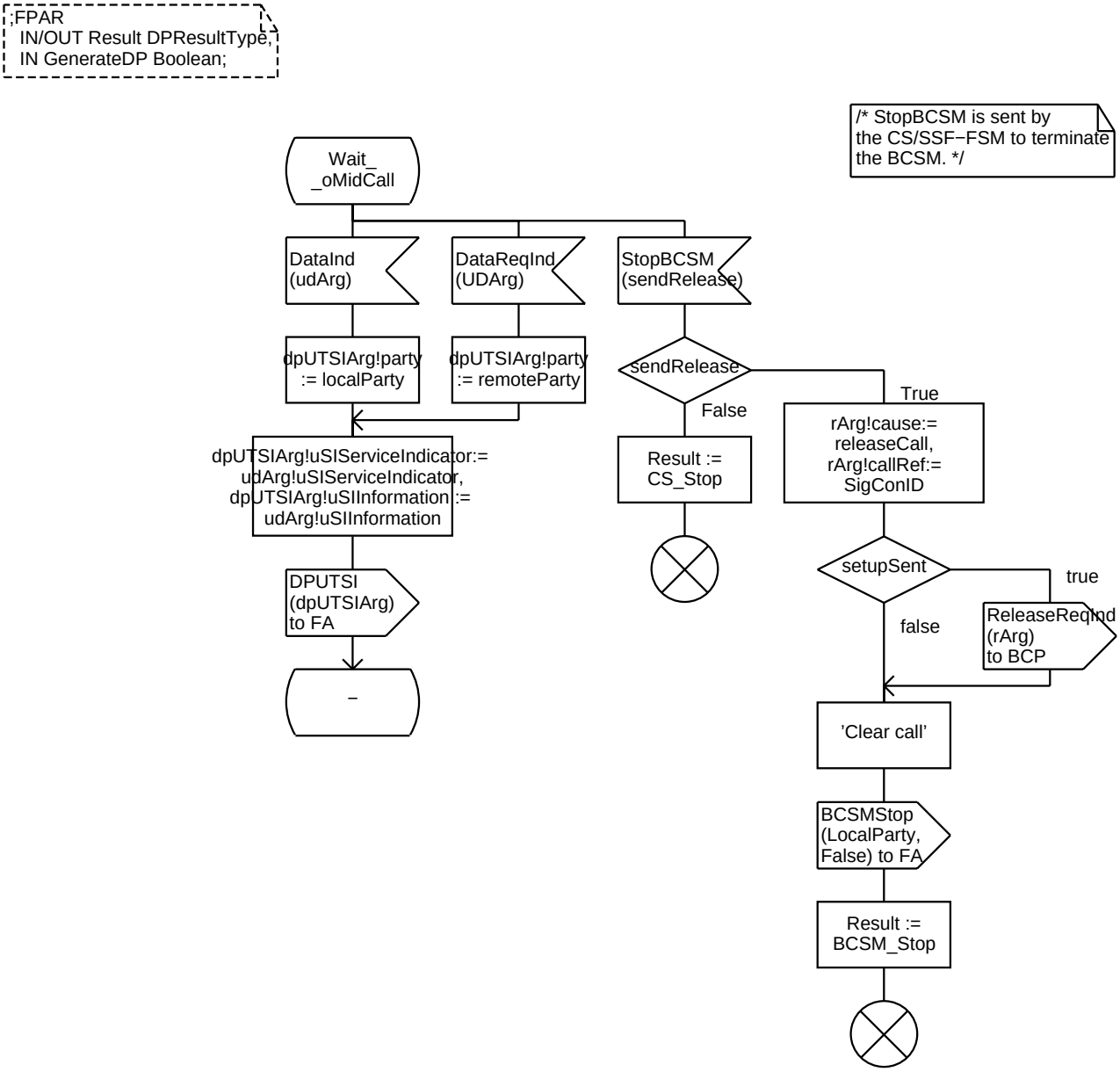
Procedure DP_oMidCall 1(4)



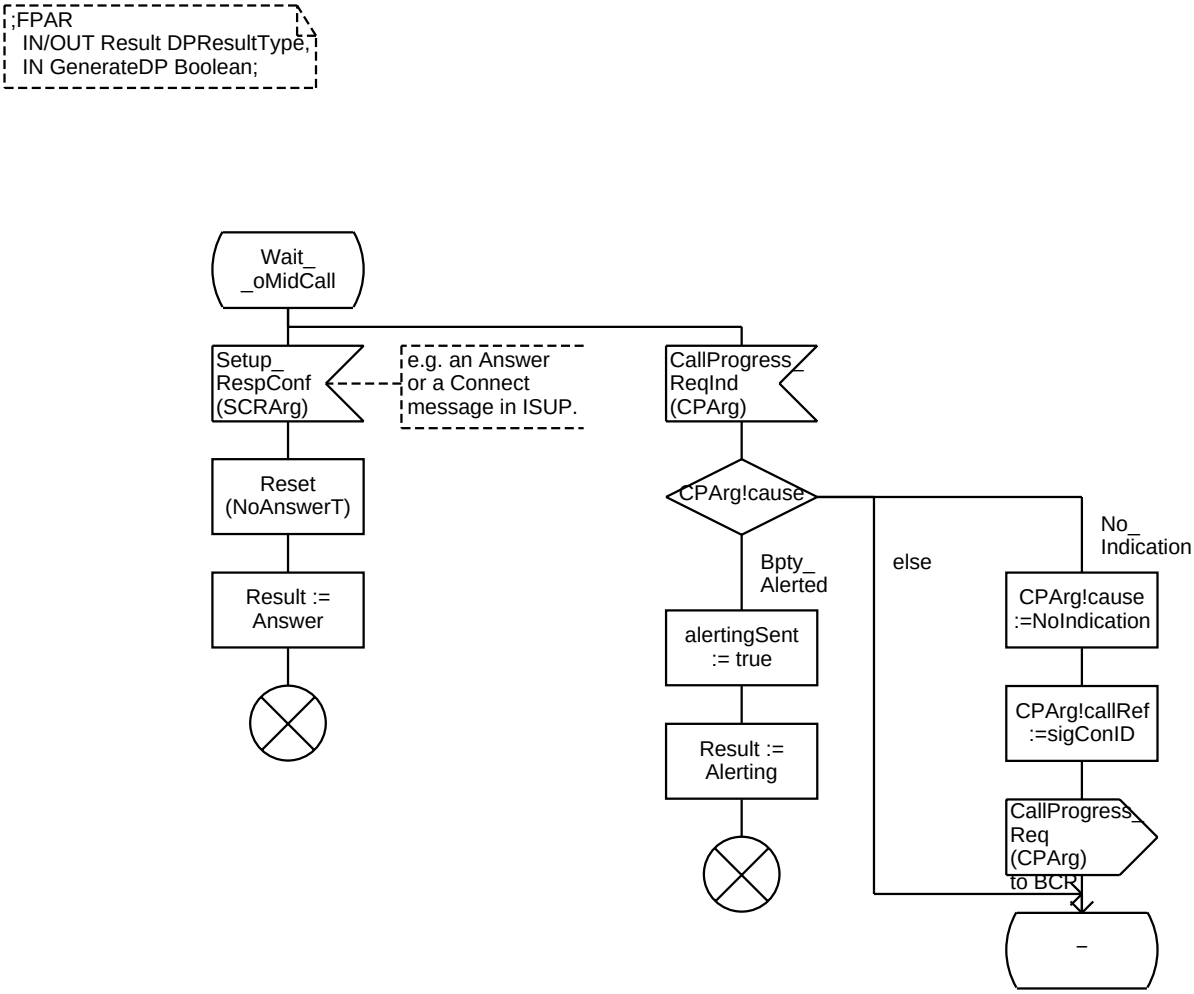
Procedure DP_oMidCall 2(4)



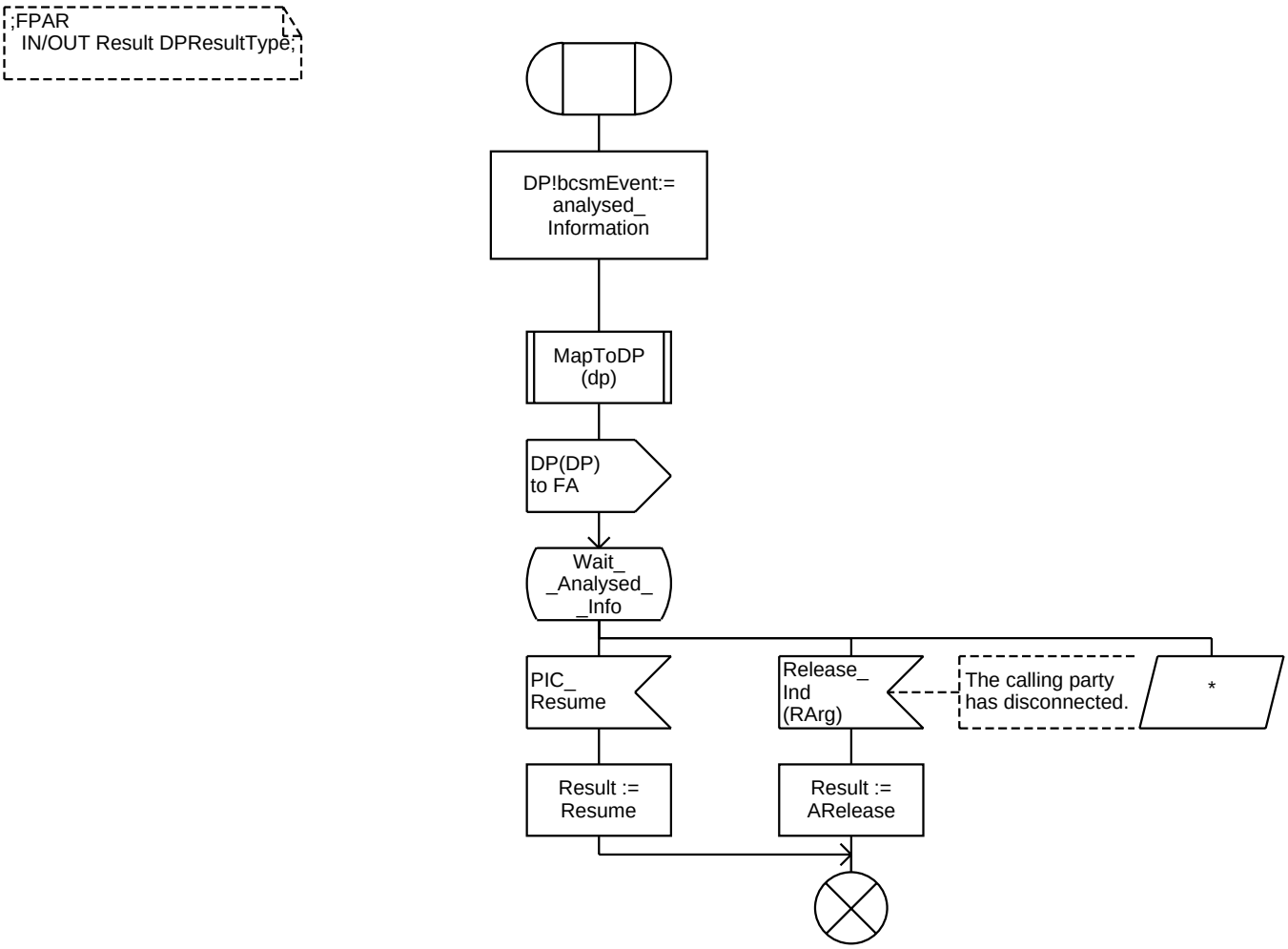
Procedure DP_oMidCall 3(4)



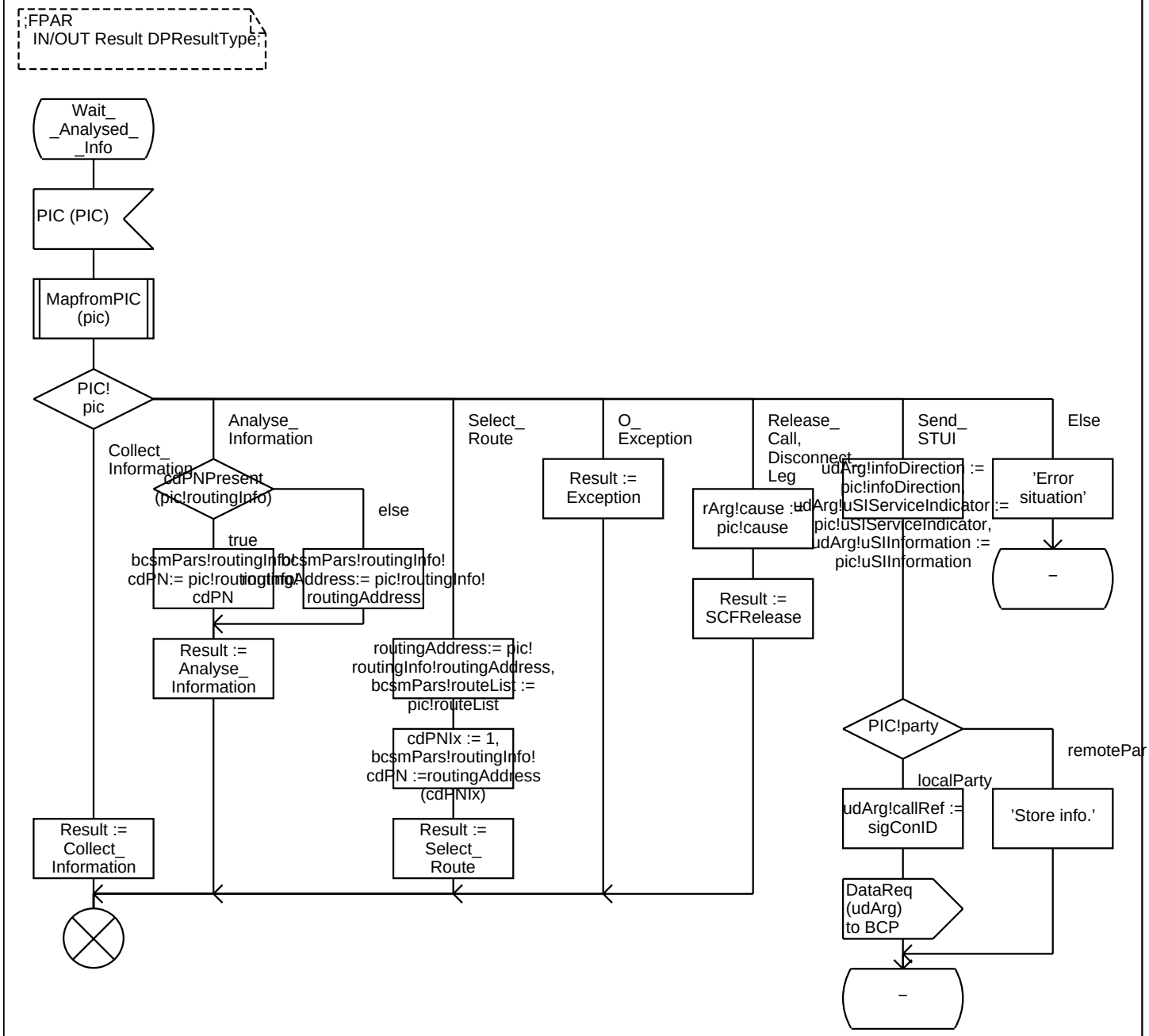
Procedure DP_oMidCall 4(4)



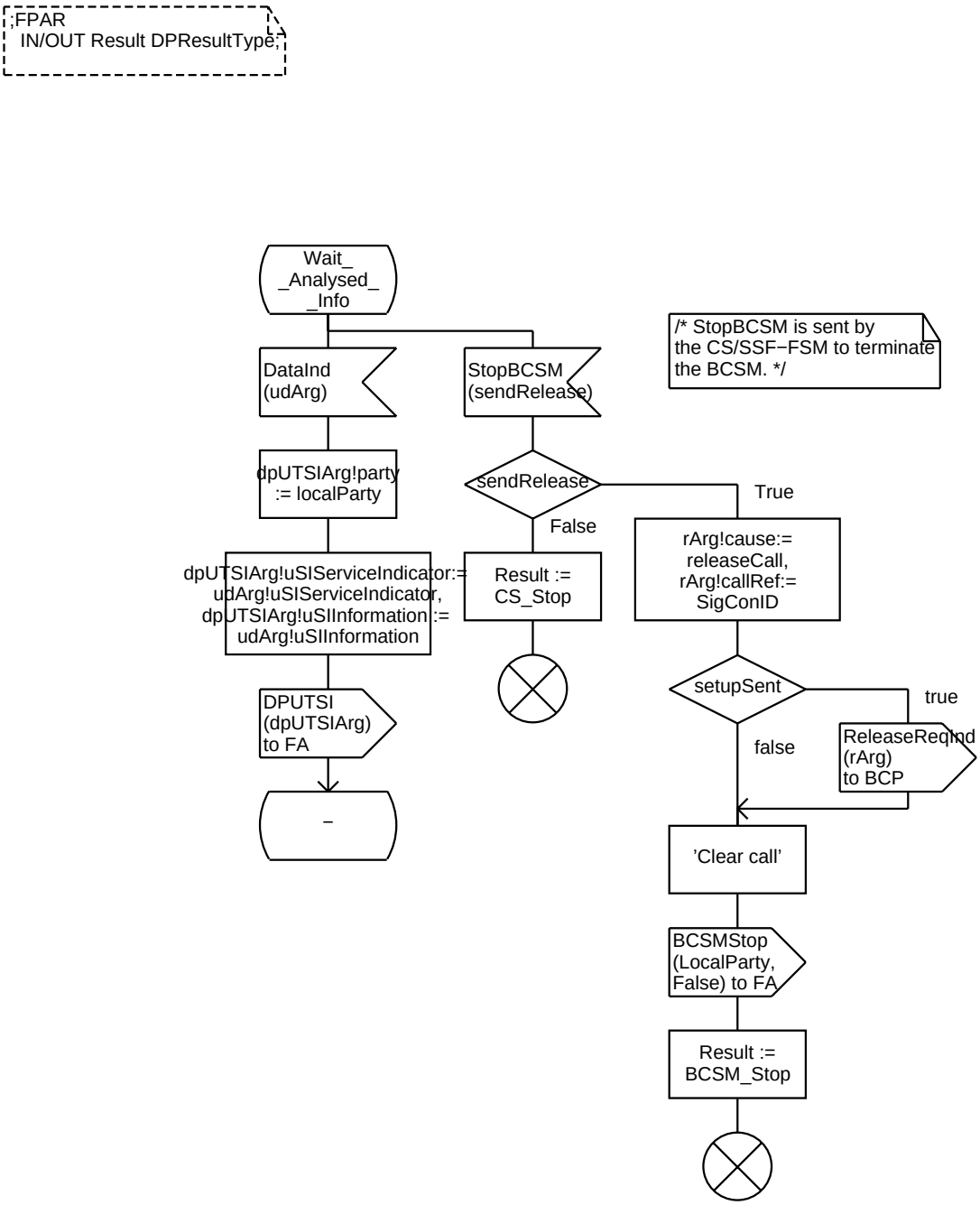
Procedure DP_Analysed_Information 1(3)



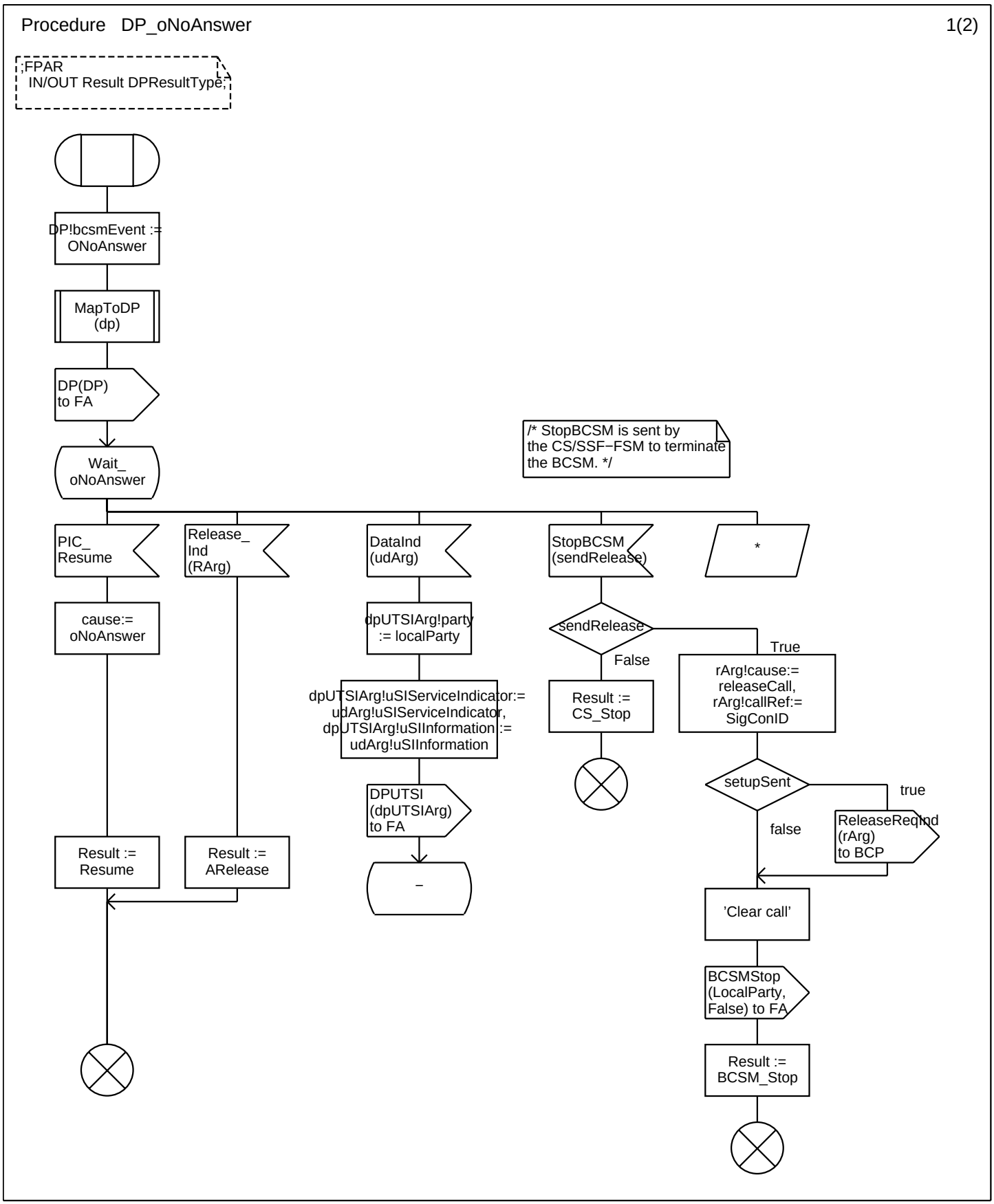
Procedure DP_Analysed_Information 2(3)



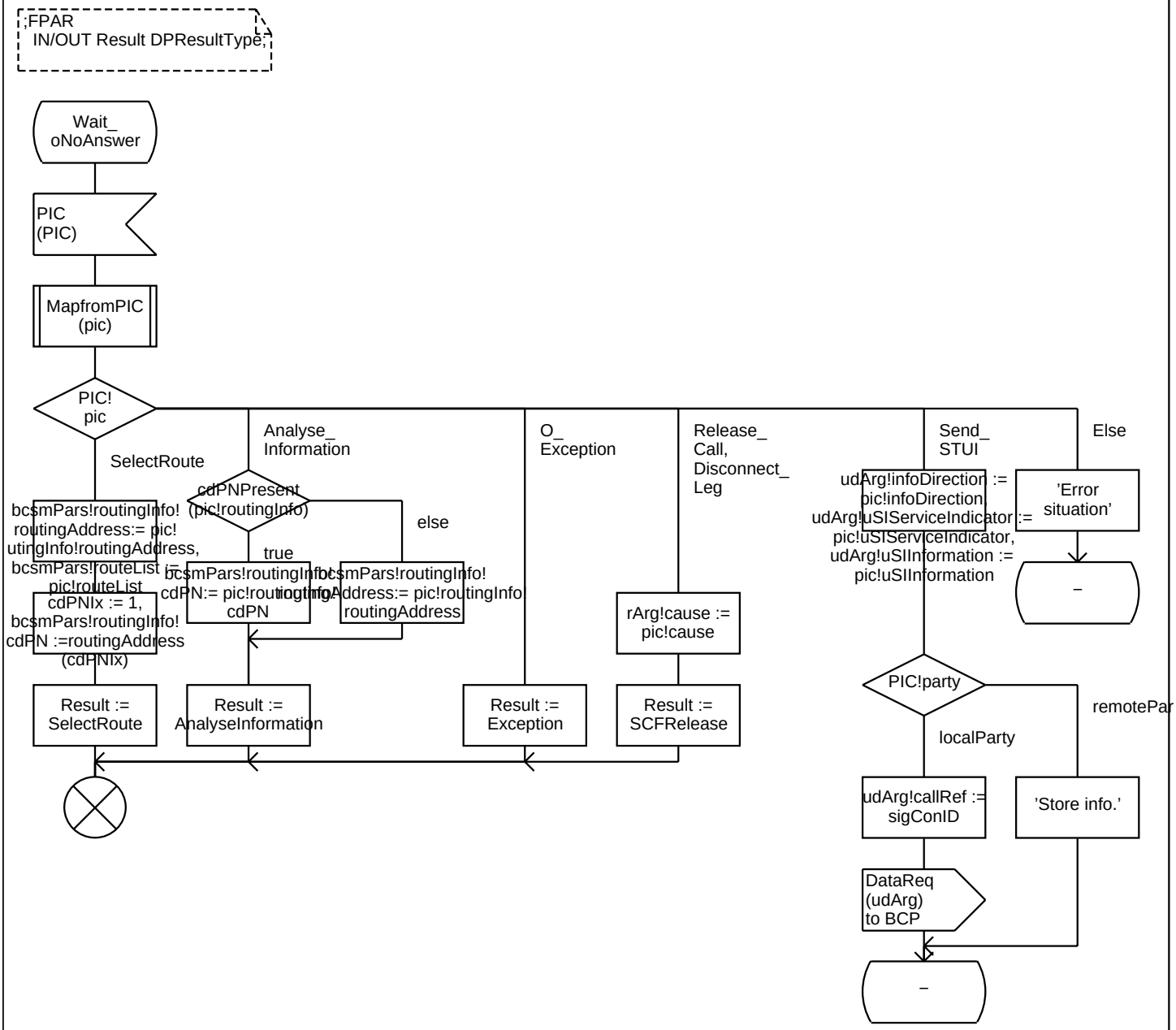
Procedure DP_Analysed_Information 3(3)



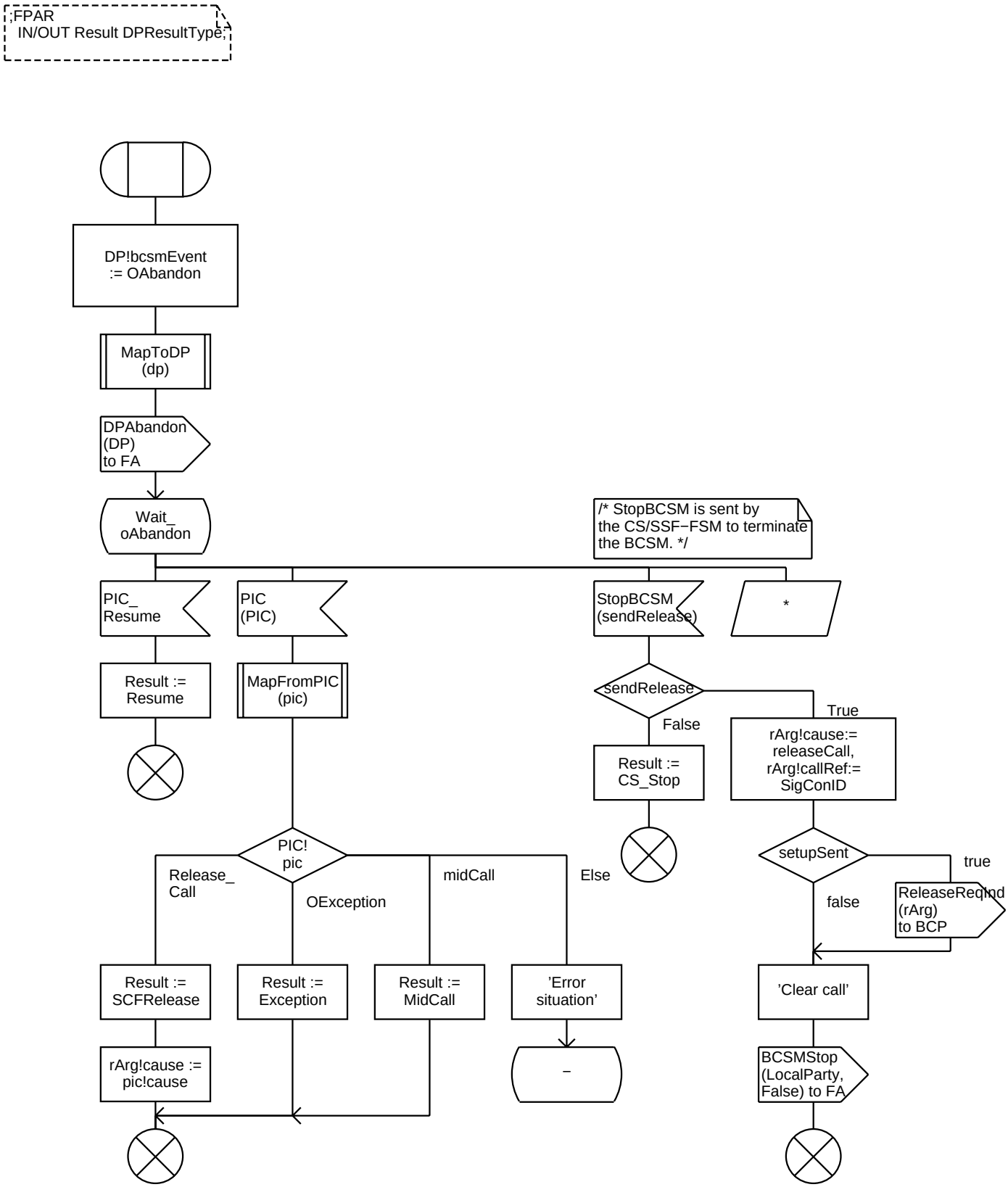
Procedure DP_oNoAnswer 1(2)



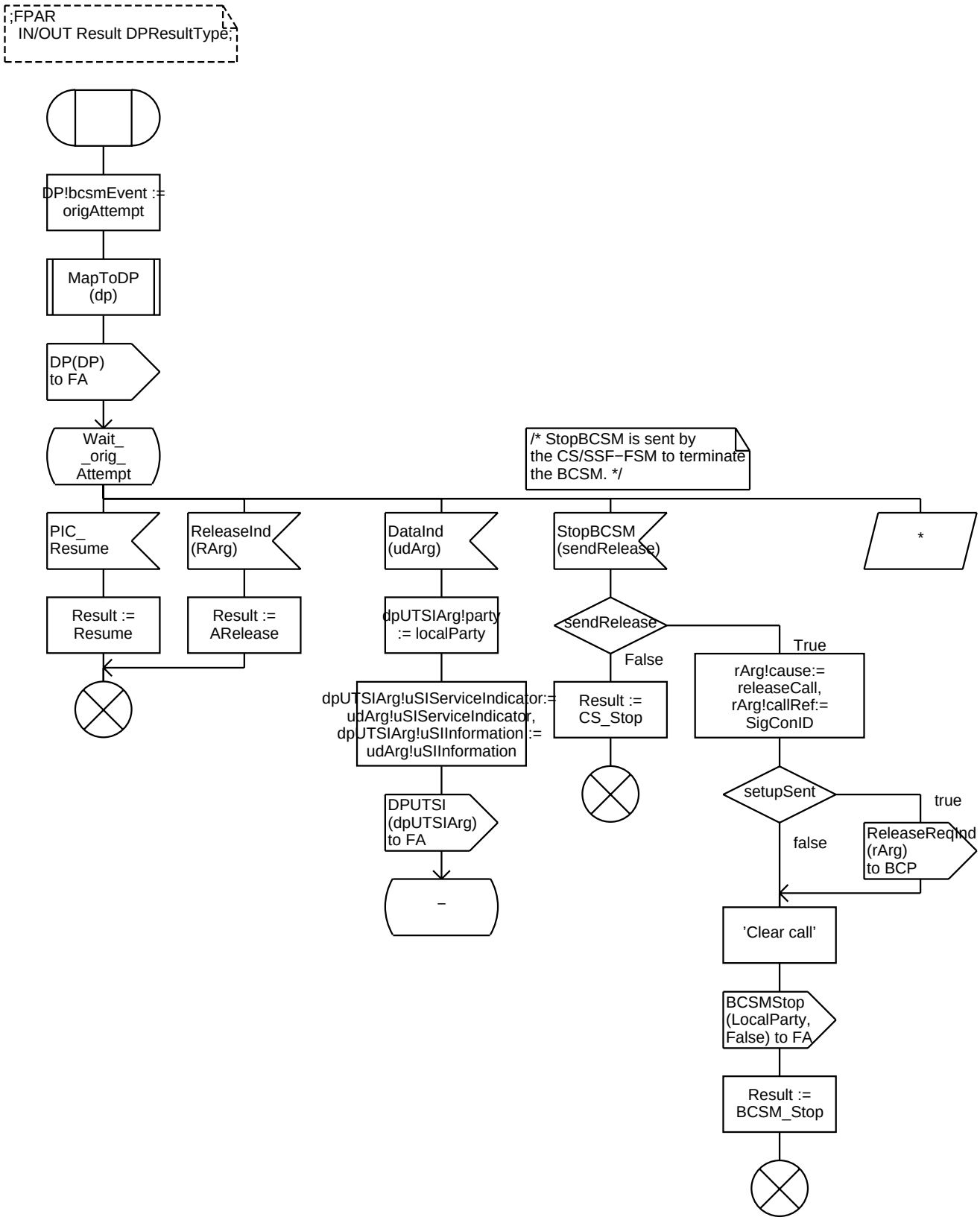
Procedure DP_oNoAnswer 2(2)



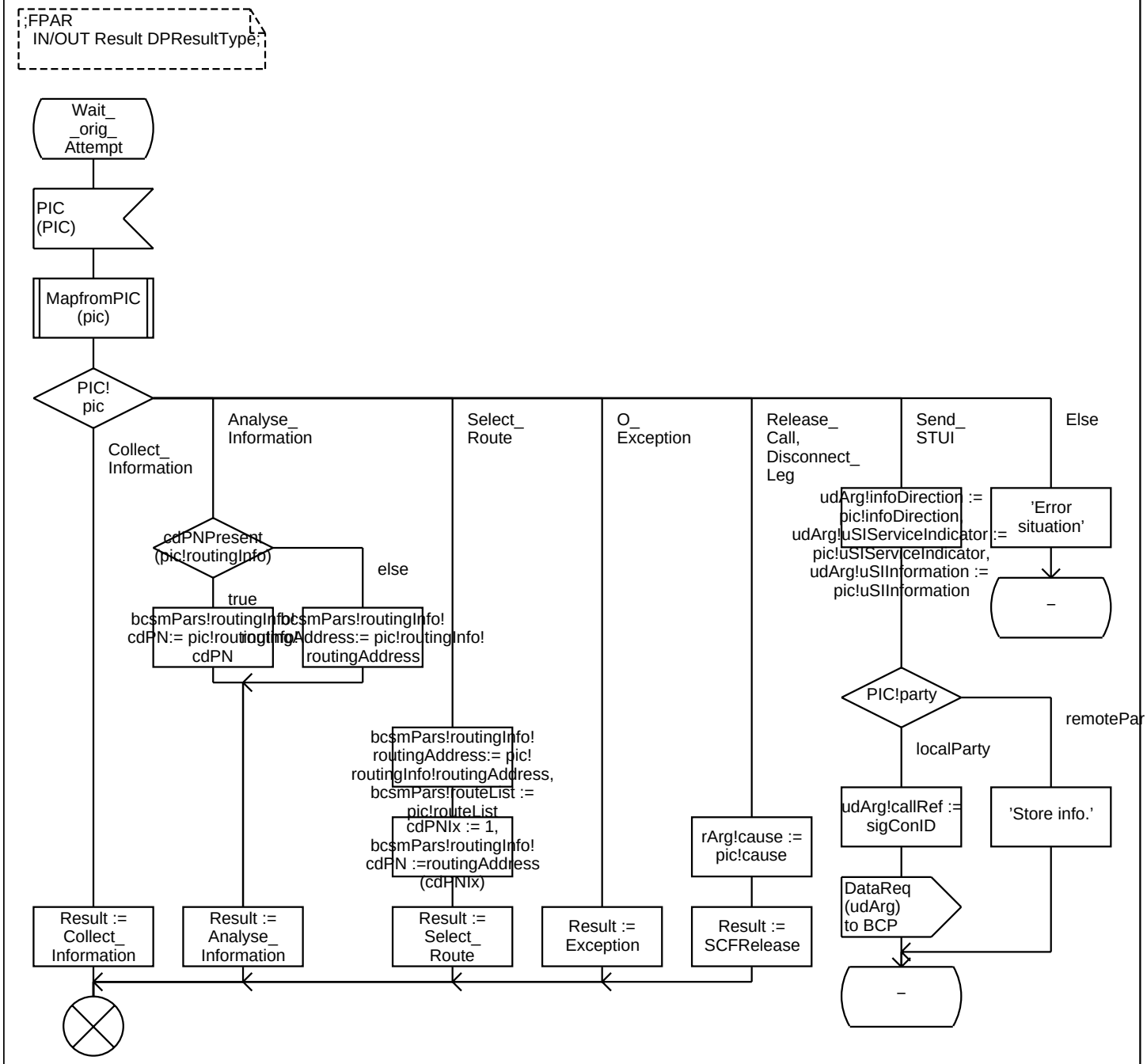
Procedure DP_oAbandon 1(1)



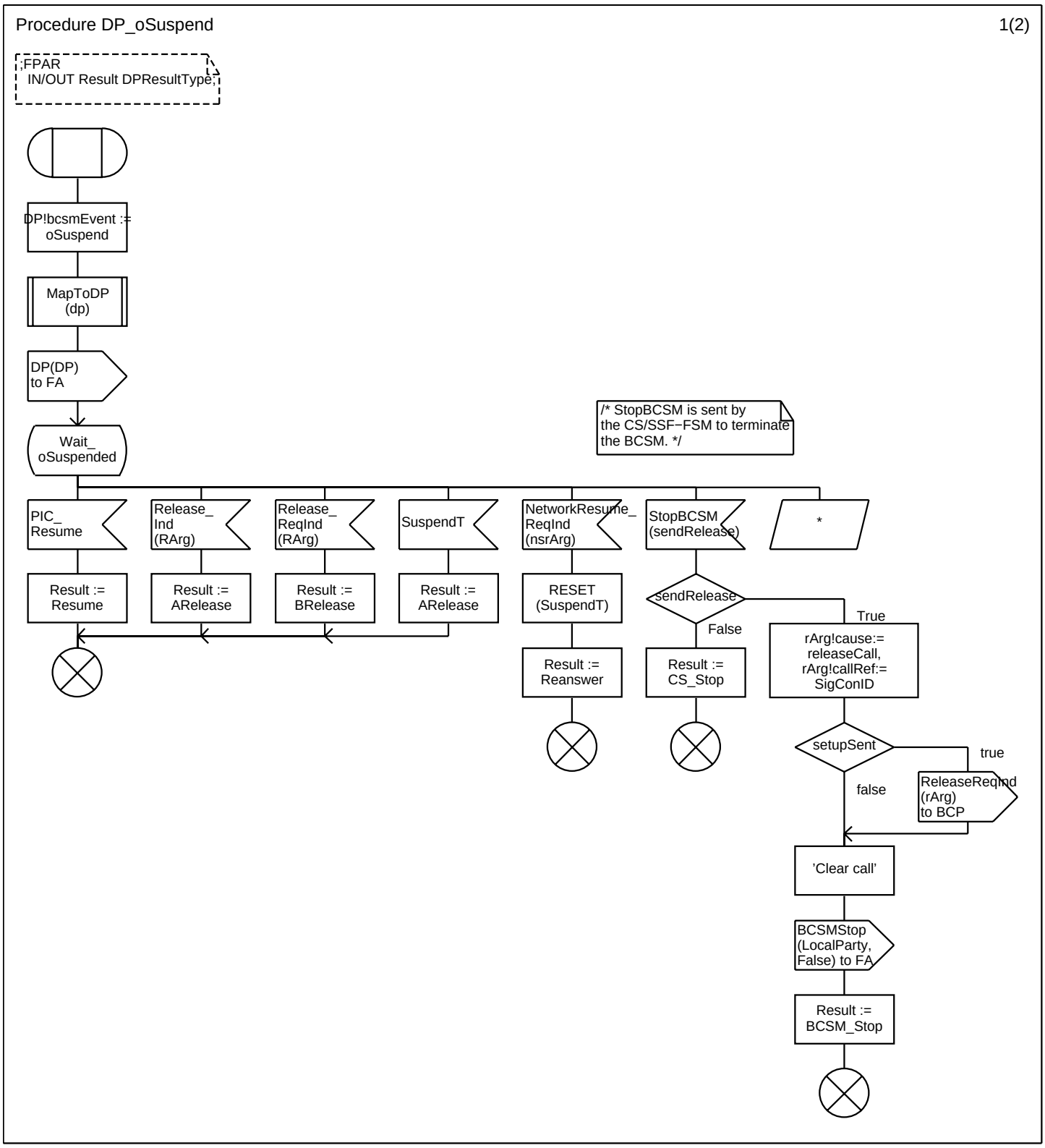
Procedure DP_origAttempt 1(2)



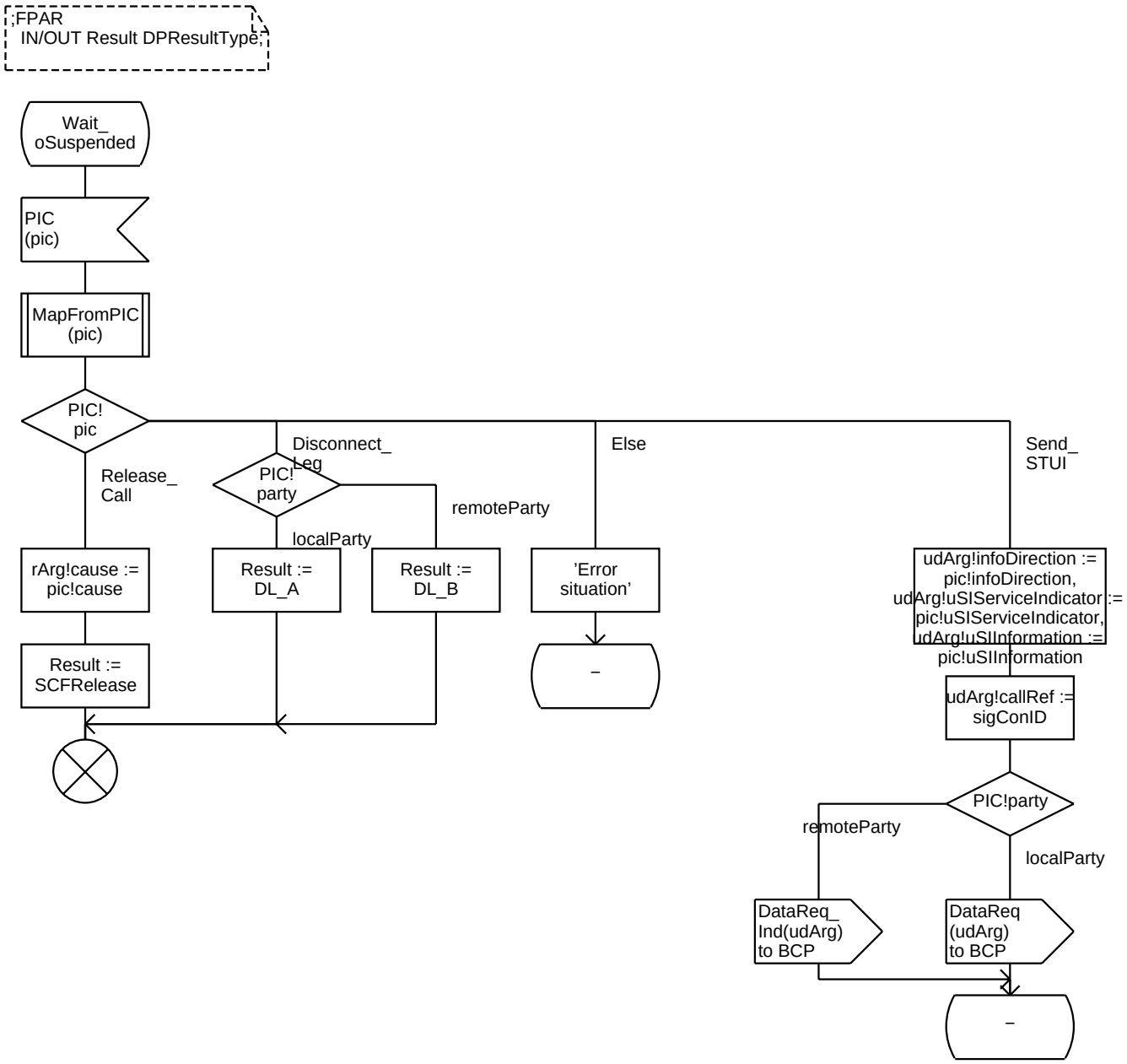
Procedure DP_origAttempt 2(2)



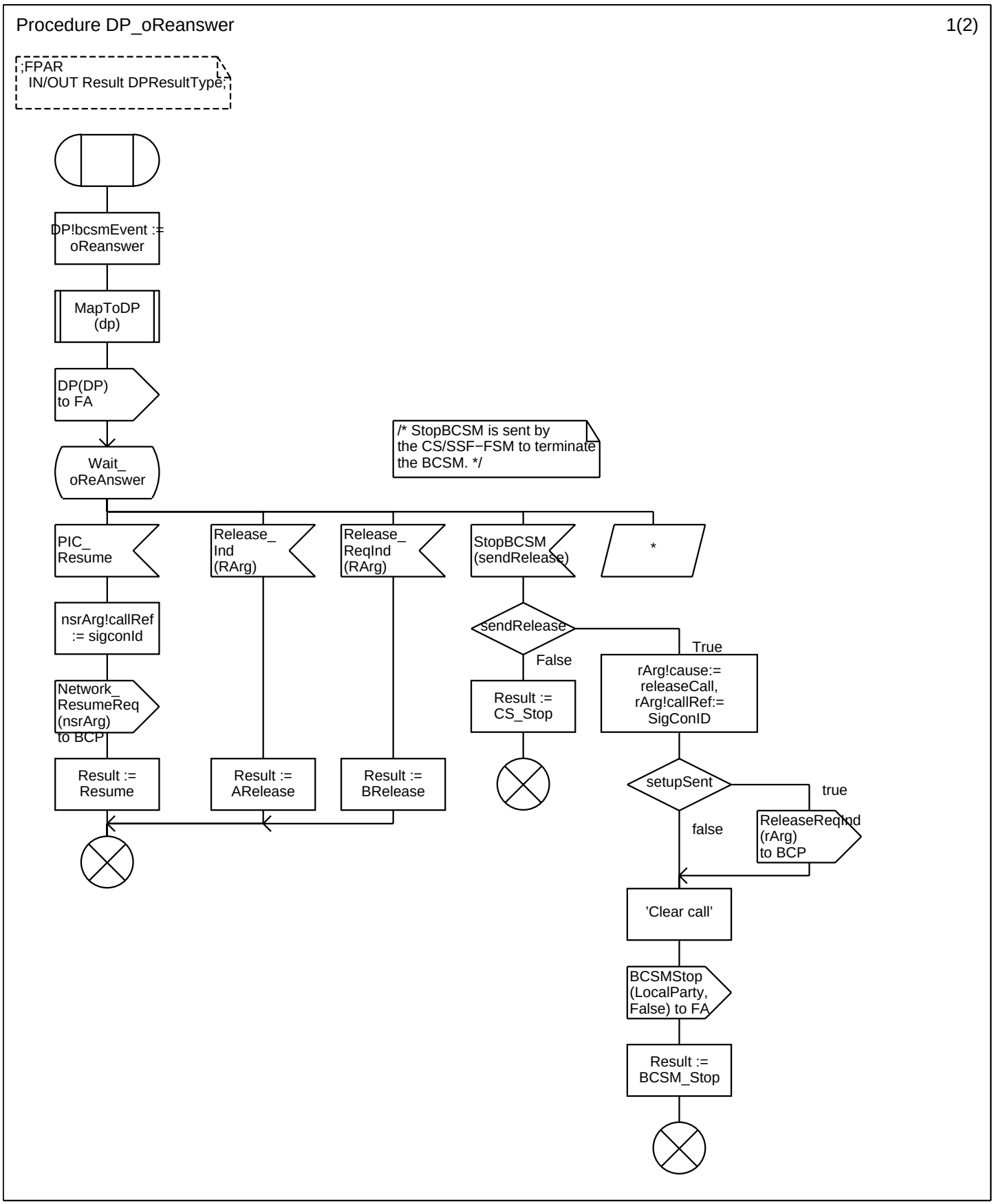
DP_oSuspend


Procedure DP_oSuspend
1(2)
;FPAR
IN/OUT Result DPResultType;
DP!bcsmEvent := oSuspend
MapToDP (dp)
DP(DP) to FA
Wait_ oSuspended
/* StopBCSM is sent by the CS/SSF−FSM to terminate the BCSM. */
PIC_ Resume
Release_ Ind (RArg)
Release_ ReqInd (RArg)
SuspendT
NetworkResume_ ReqInd (nsrArg)
StopBCSM (sendRelease)
*
Result := Resume
Result := ARelease
Result := BRelease
Result := ARelease
RESET (SuspendT)
sendRelease
True
False
Result := Reanswer
Result := CS_Stop
rArg!cause:= releaseCall, rArg!callRef:= SigConID
setupSent
true
false
ReleaseReqInd (rArg) to BCP
'Clear call'
BCSMStop (LocalParty, False) to FA
Result := BCSM_Stop

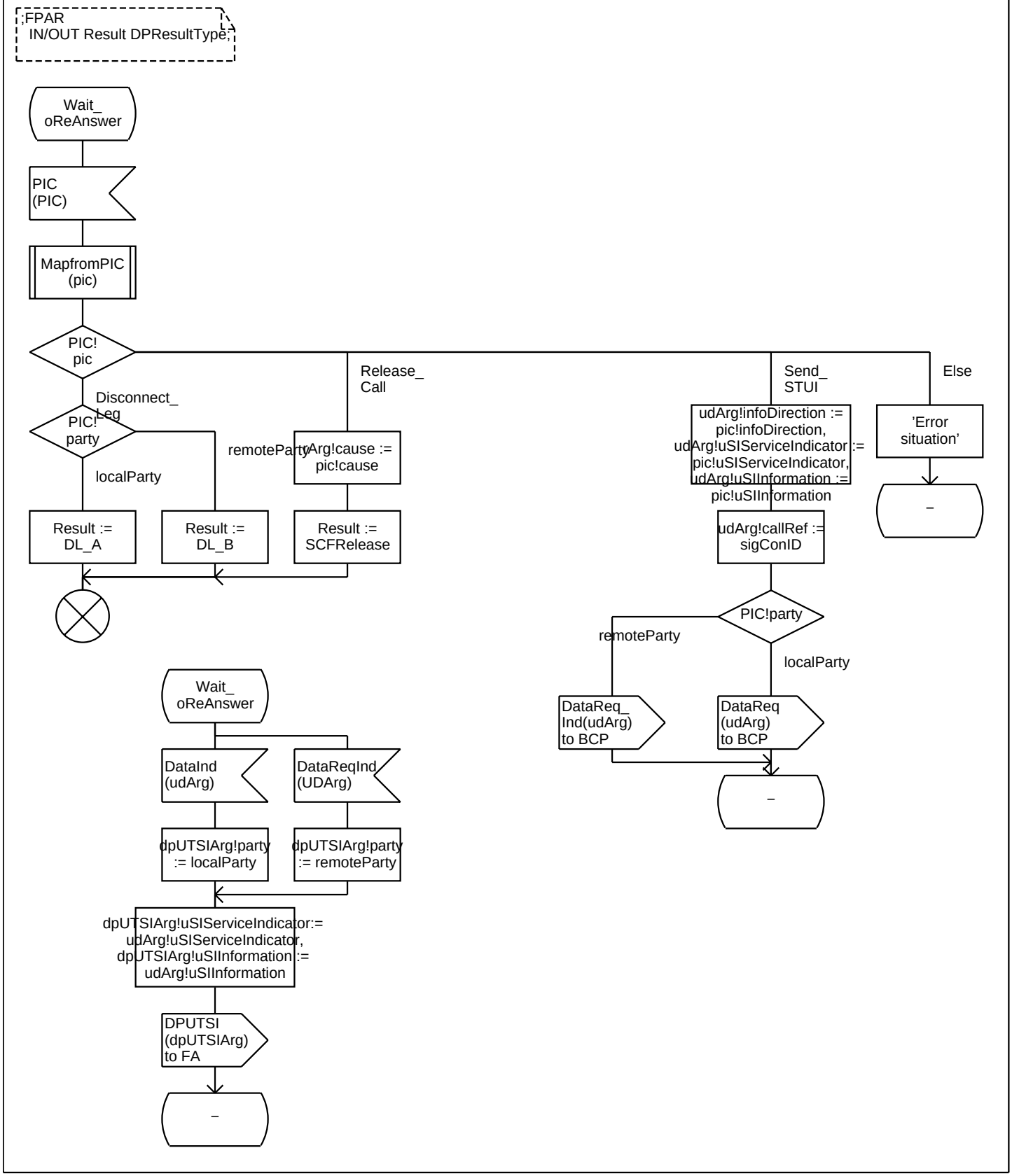
Procedure DP_oSuspend 2(2)



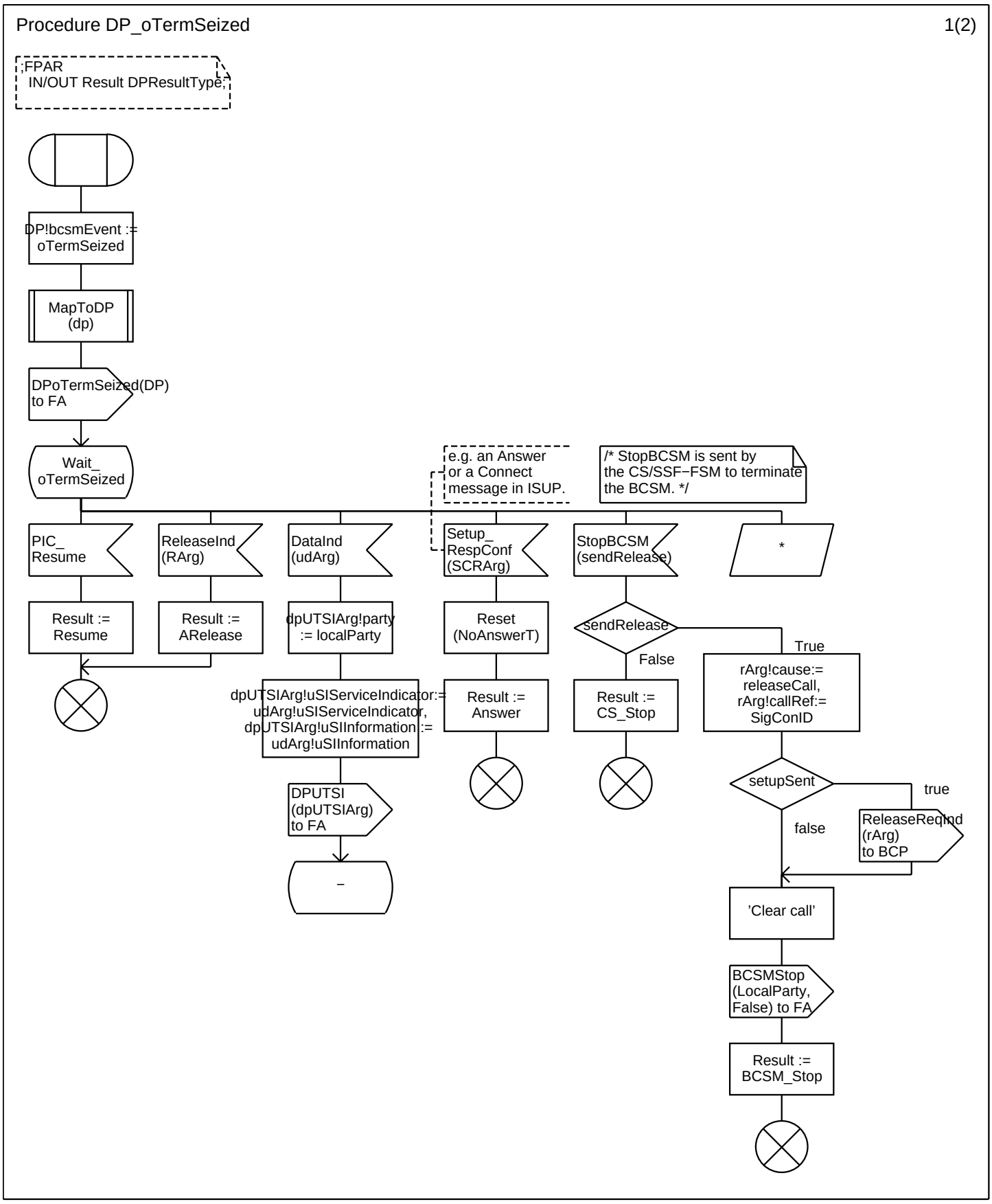
Procedure DP_oReanswer 1(2)



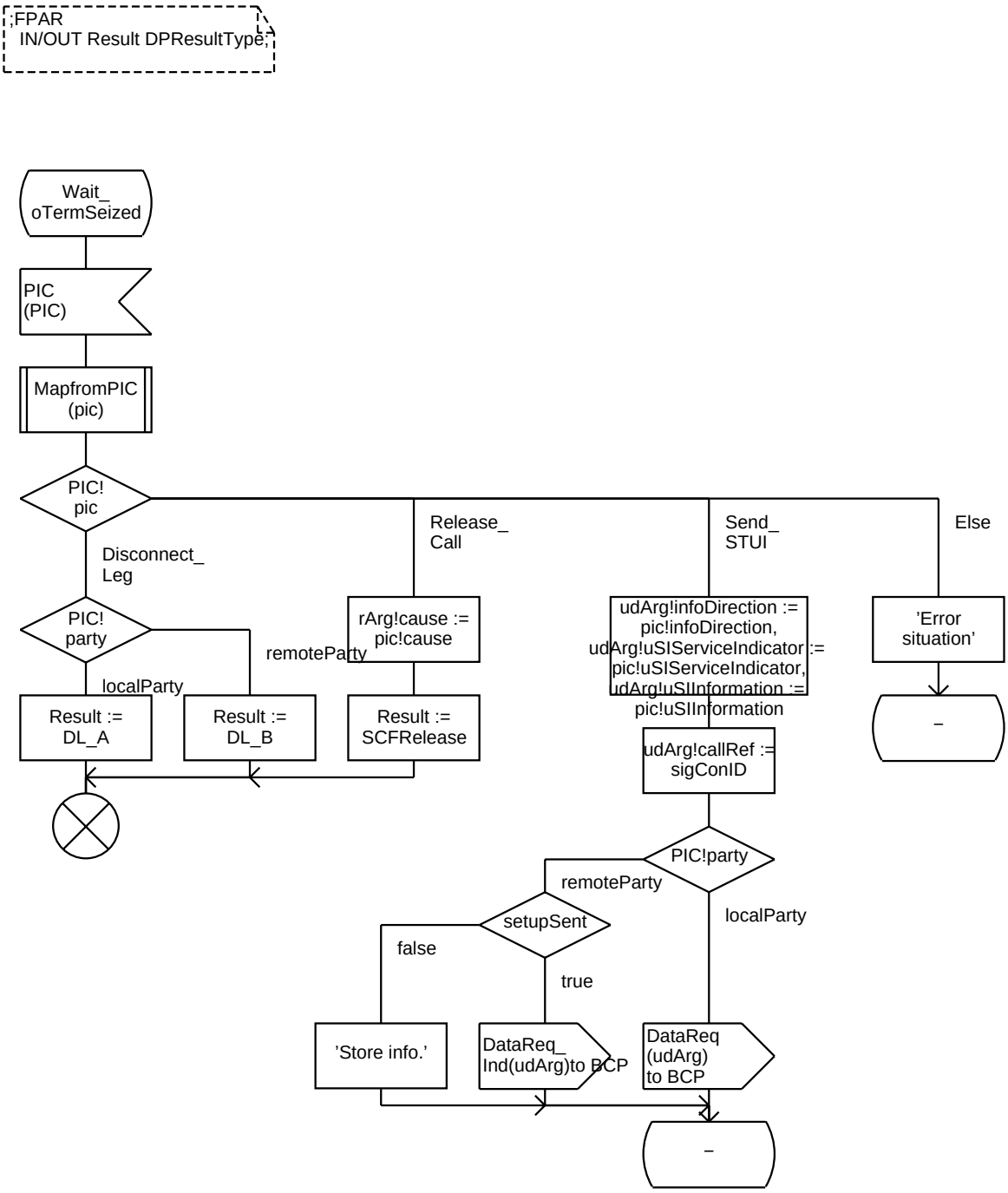
Procedure DP_oReanswer 2(2)



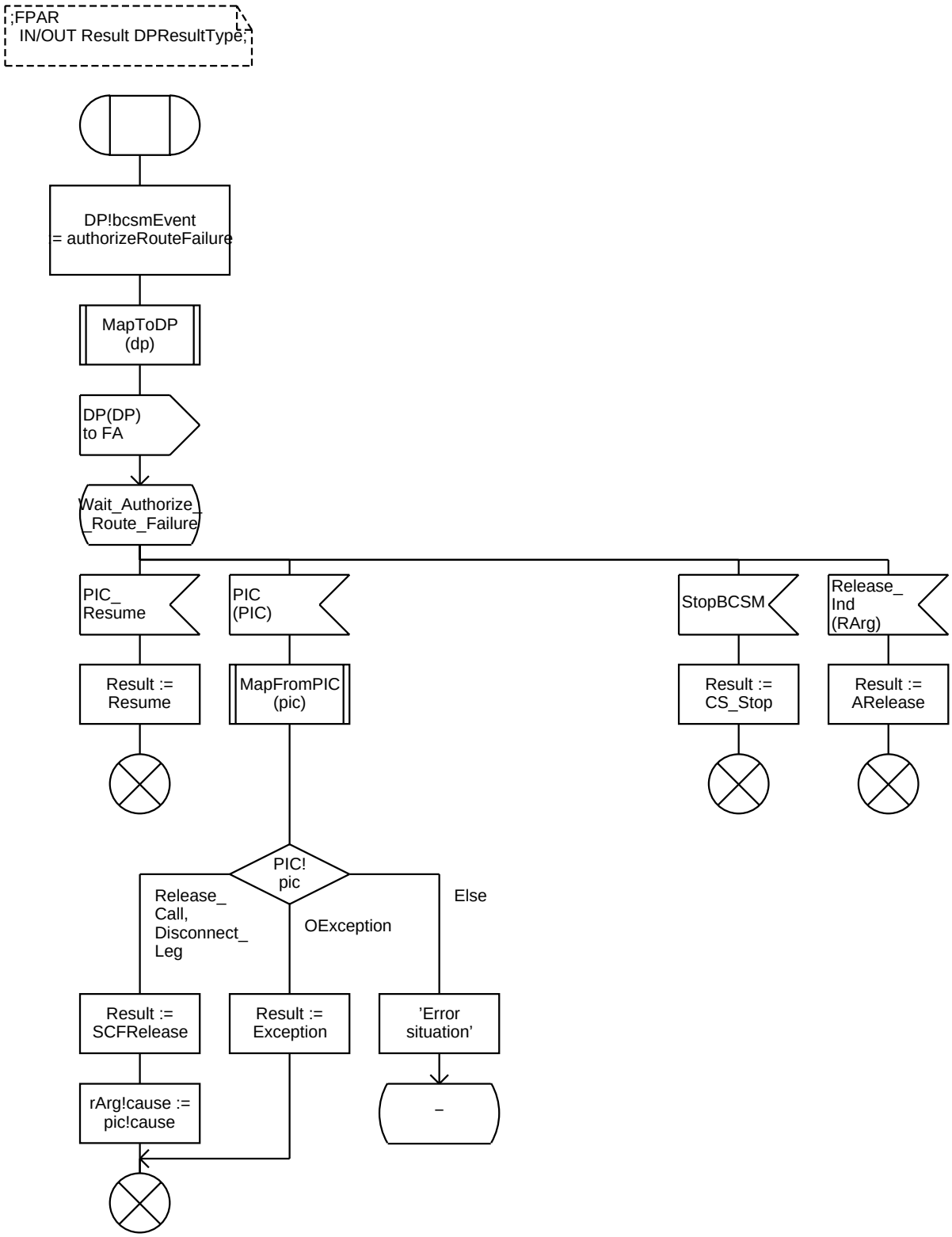
Procedure DP_oTermSeized 1(2)



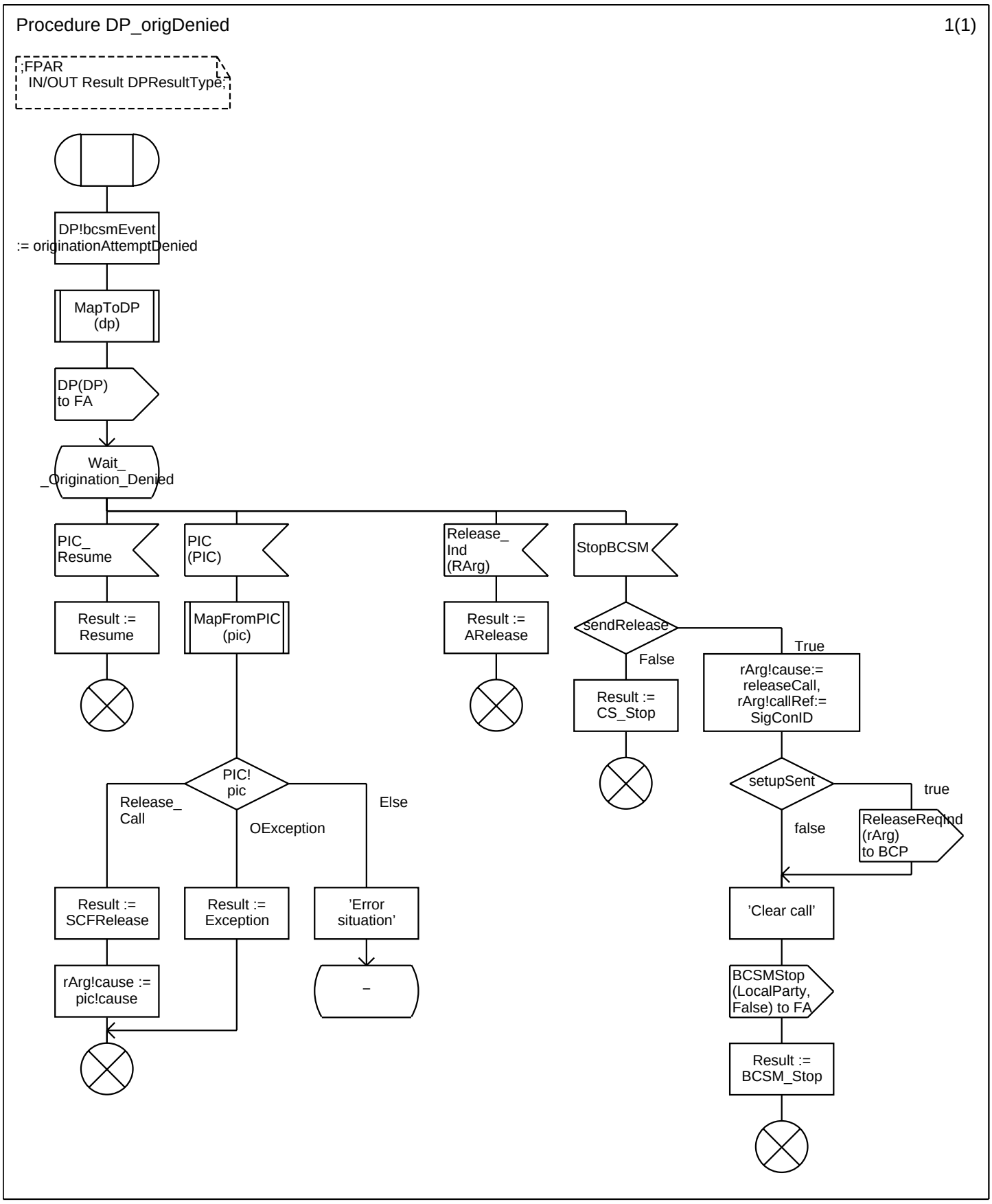
Procedure DP_oTermSeized 2(2)



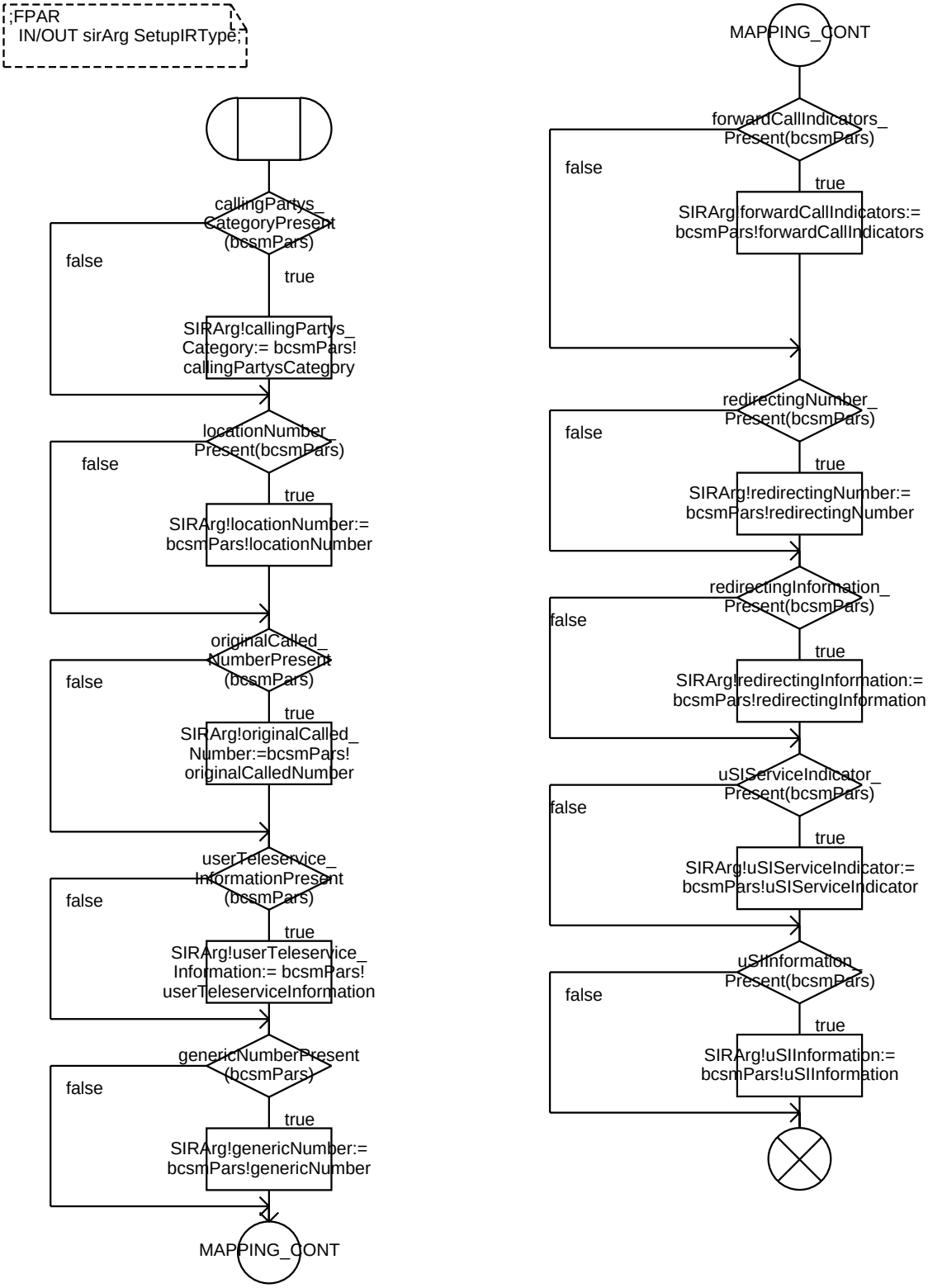
Procedure DP_Authorize_Route_Failure 1(1)

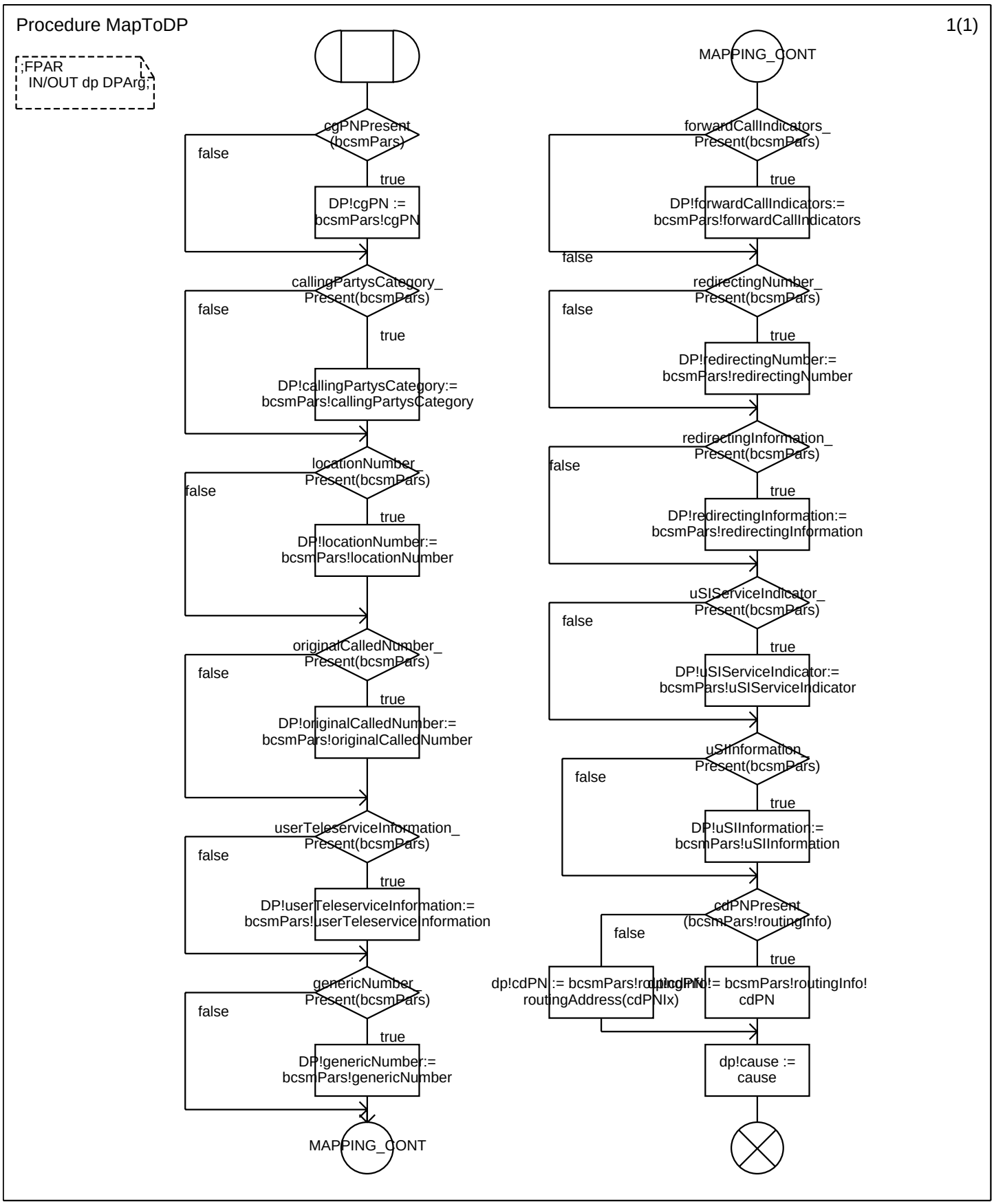


DP_origDenied

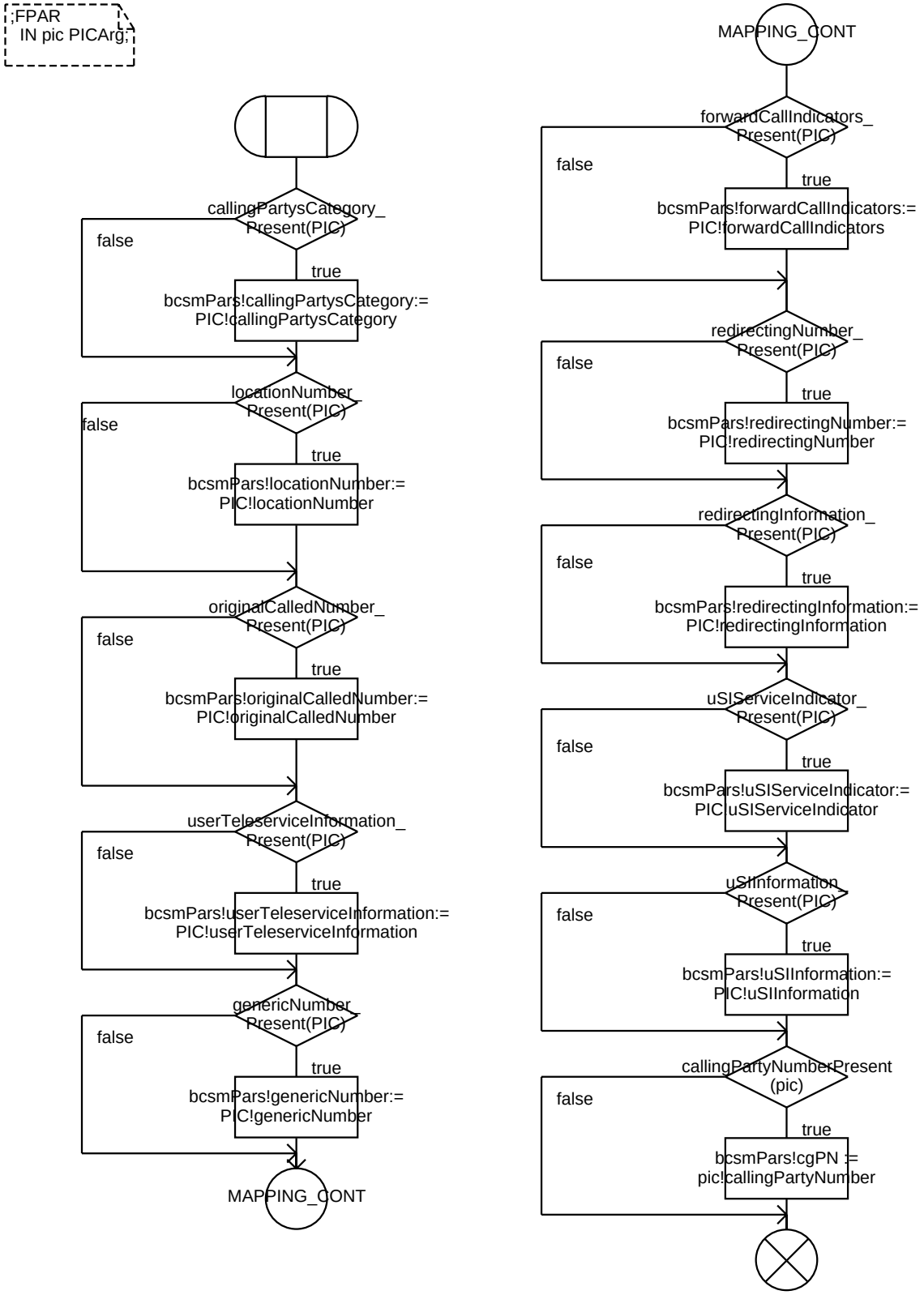


Procedure MapToSIRArg 1(1)




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

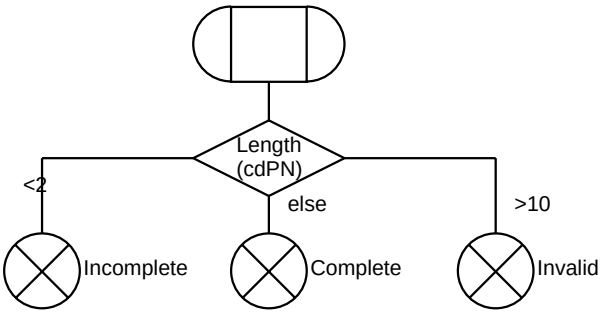
Procedure MapFromPIC 1(1)



Procedure AnalyzeDigits 1(1)

; FPAR IN cdPN CalledPartyNumber;
RETURNS DigitAnalysisResultType;

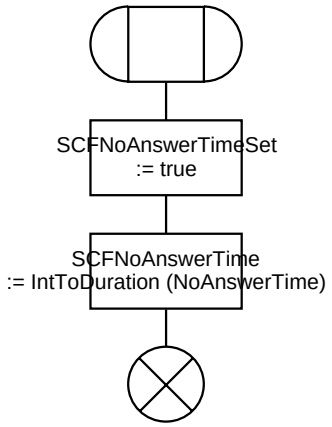
```
/* performs digit analysis
 * and returns whether the number
 * is complete, not complete,
 * or invalid
 *
 * The implementation of this procedure
 * is simplified for simulation
 * purposes.
 */
```



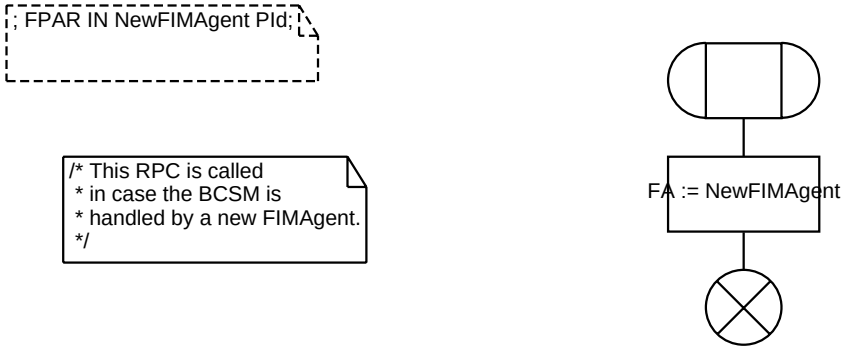
Procedure <<Process Type OriginatingBCSM>> SetNoAnswerTime 1(1)

; FPAR LegId LegType, NoAnswerTime Integer;

/* This RPC is called by CS
 * and will result in NoAnswerTime being
 * used as the value in the no−answer timer
 */



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



Process Type TerminatingBCSM 1(13)

```
;FPAR  bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */
```

```
/***** T−BCSM FOR INAP CS−3. *****/
```

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

```
NEWTYPE DPResultType
  LITERALS
    Resume,
    SelectFacility,
    PresentCall,
    ARelease,
    BRelease,
    MidCall,
    SCFRelease, /* Used with ReleaseCall. */
    Answer,
    Alerting,
    AuthTermination,
    Exception,
    Busy,
    NoAnswer,
    Failure,
    Reanswer,
    DL_A, DL_B, /* Used with DisconnectLeg */
    CS_Stop,
    BCSM_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 */
    CS_Stop,
    BCSM_Stop
ENDNEWTYPE;
```

BCP (T_BCSM_Out) (T_BCSM_In)

FA (BCSM_To_FA) (FA_To_BCSM)

Process Type TerminatingBCSM 2(13)

```
;FPAR  bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */
```

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

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

DCL
  BCP PId,

  SigConID CallRef,

  calledPartyNumber CalledPartyNumber,

  /* Used to indicate the status of the call processing. */
  setupSent boolean := false,    /* Indicates if SetupReq is sent to the called party */

  StartState BCSMStateType,   /* The start state of the BCSM. */

  SCFNoAnswerTimeSet Boolean := false,
  SCFNoAnswerTime Duration,

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

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

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

Process Type TerminatingBCSM 3(13)

;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */

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

PIC_T_ _Null

PIC_Select_ _Facility

PIC_ _Disconnect

PIC_T_ _Suspended

PIC_Authorize_ _Termination_ _Attempt

PIC_Present_ _Call

PIC_tActive

PIC_ _TException

PIC_ _tAlerting

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

DP_termAttempt_ Authorized

DP_tNoAnswer

DP_term_ Attempt

DP_ _tReanswer

DP_tBusy

DP_tMidCall

DP_Facility_ _Selected_and_ _Available

DP_termDenied

DP_tAnswer

DP_tAbandon

DP_ _tSuspend

DP_tCall_ _Accepted

DP_tDisconnect

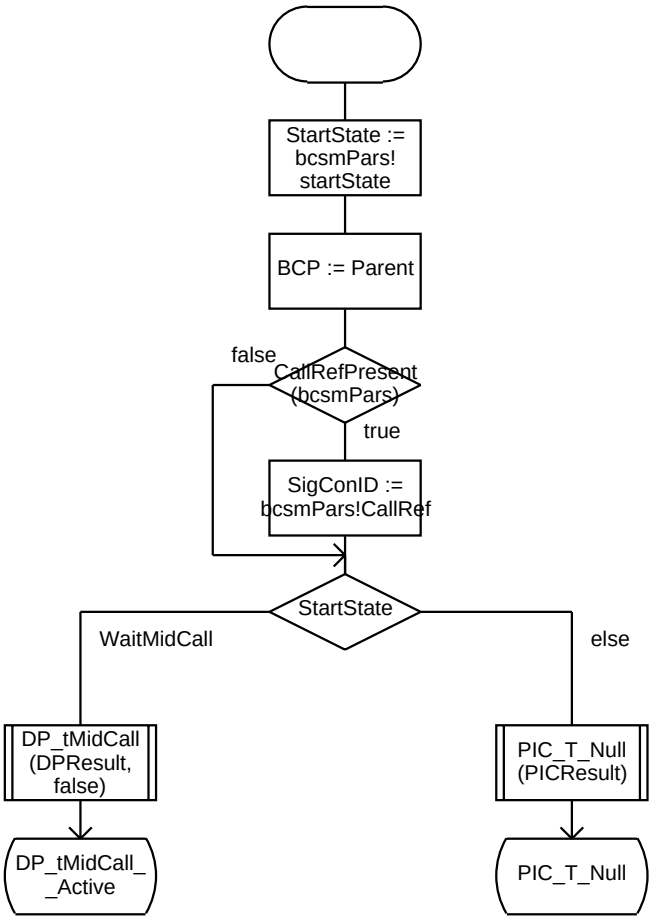
EXPORTED SetNoAnswerTime

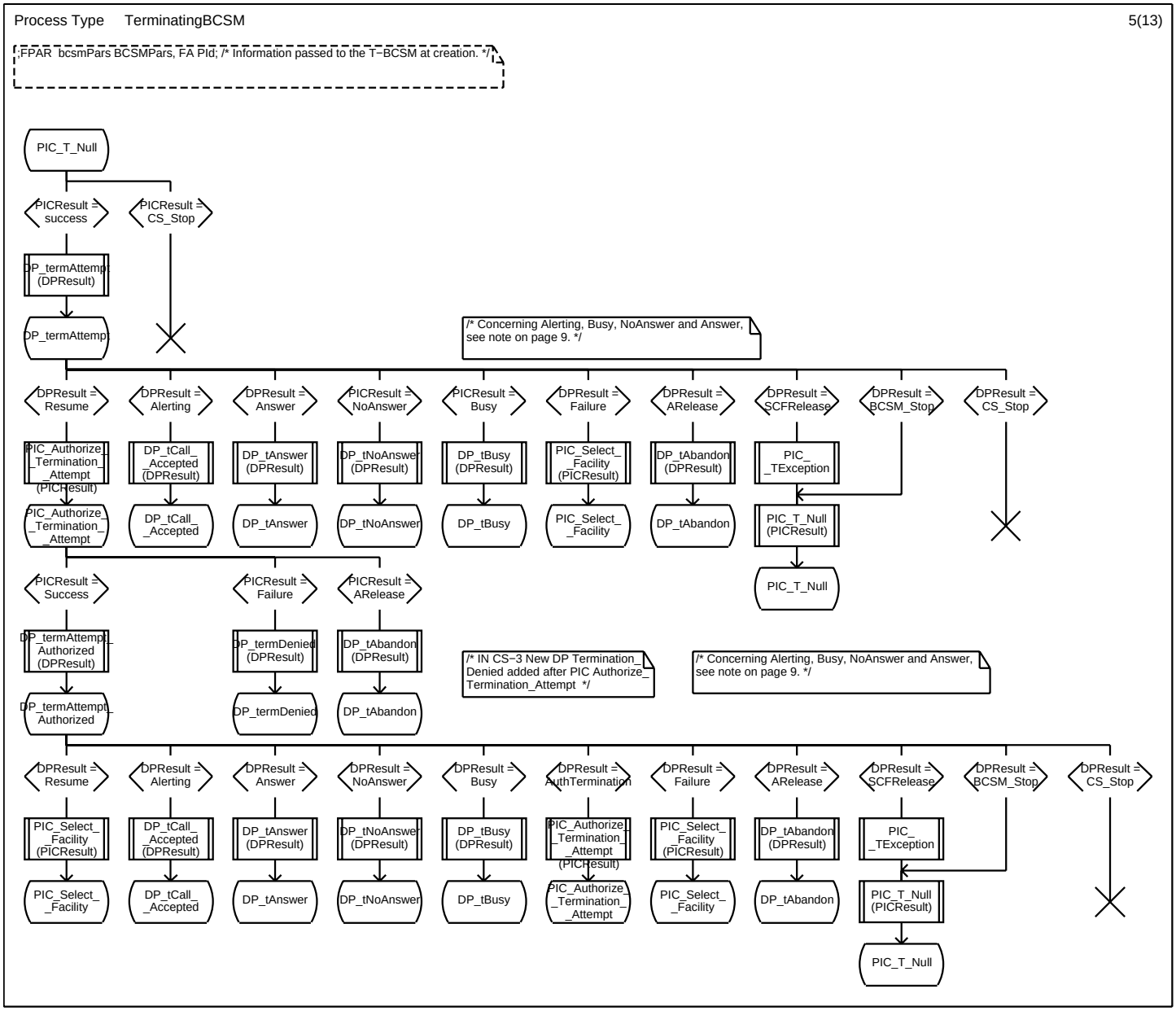
EXPORTED SetFIMAgent

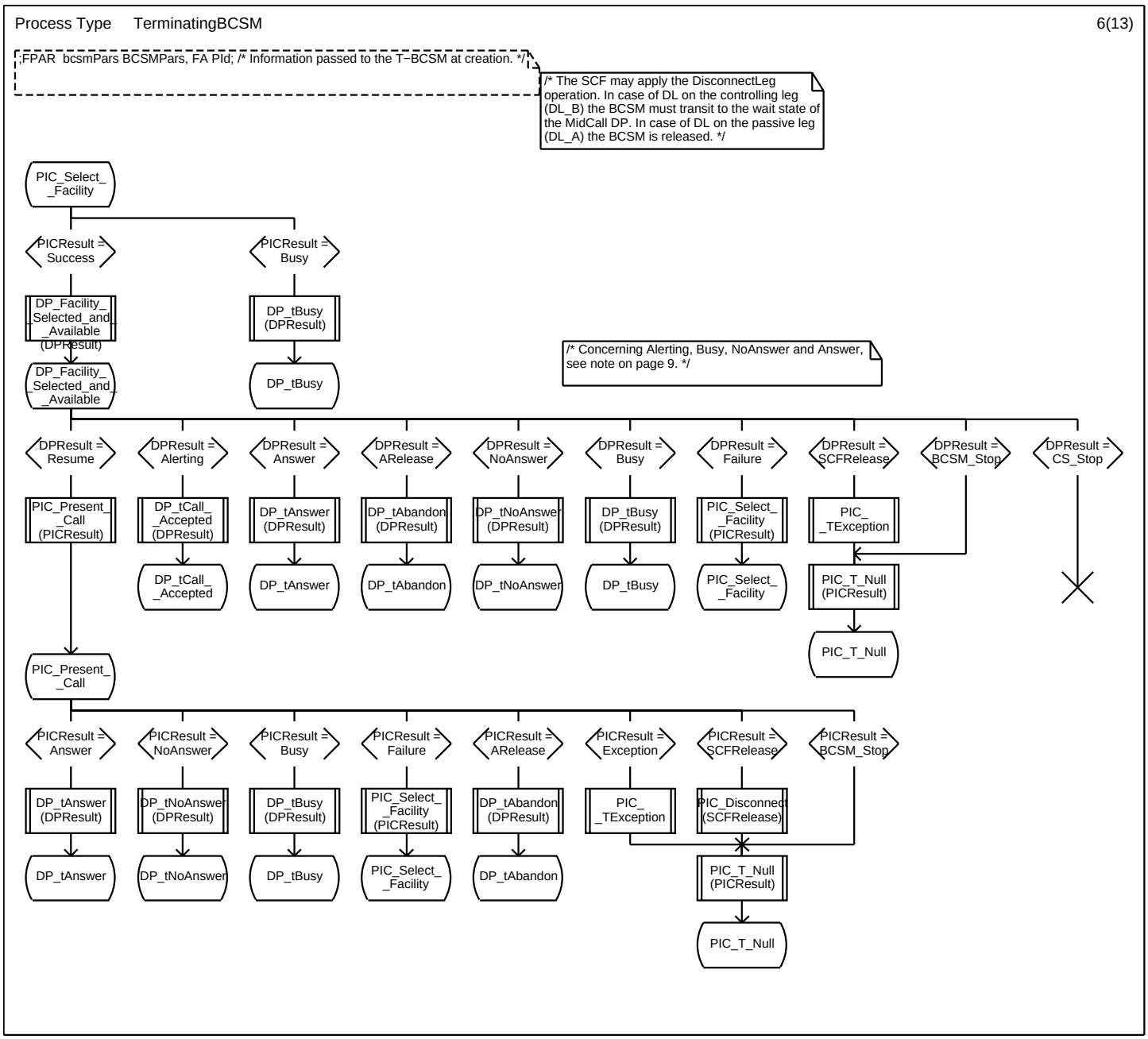
Process Type TerminatingBCSM 4(13)

;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */

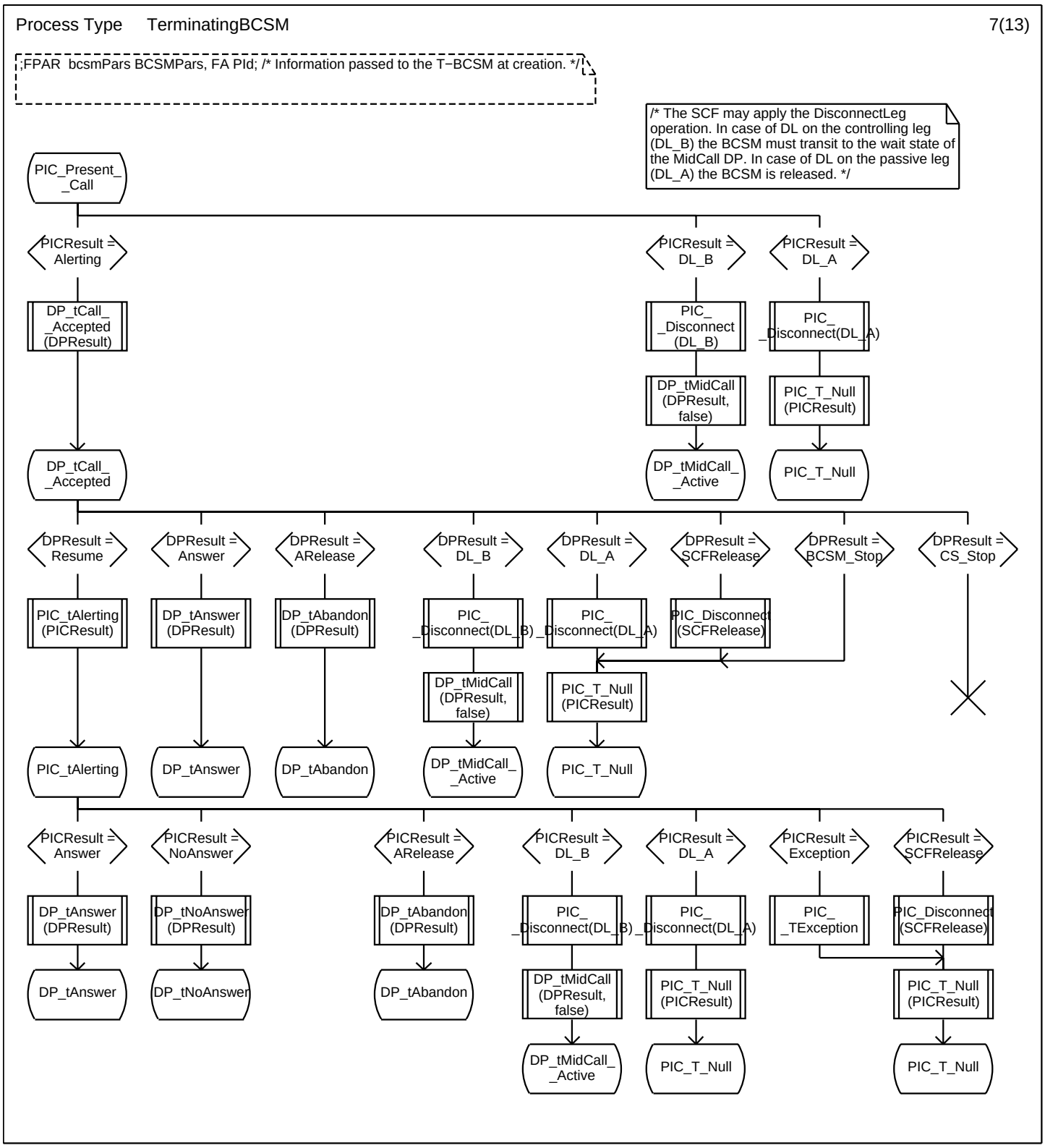
/* The BCSM may be started from different states. */



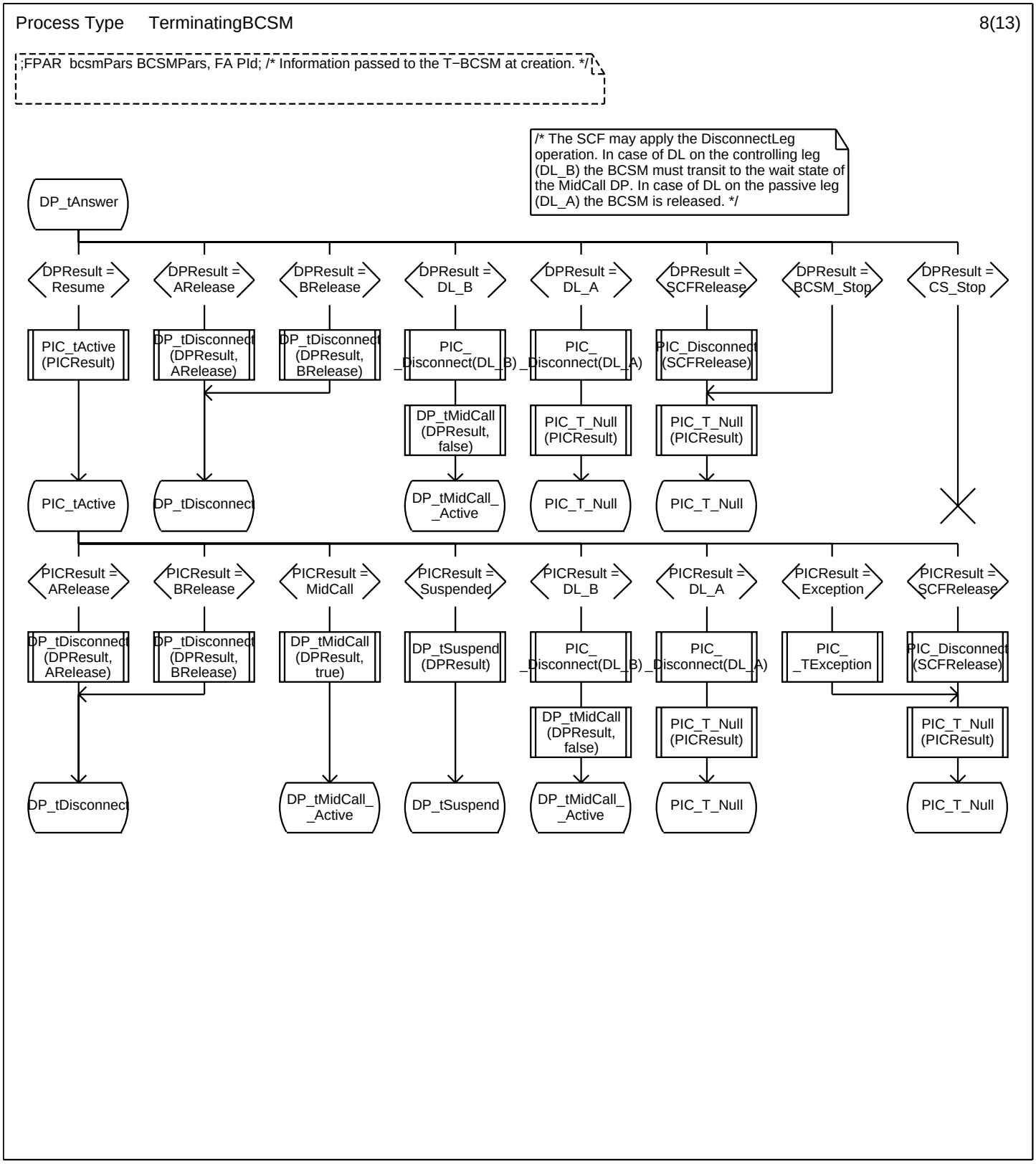

Process Type TerminatingBCSM
5(13)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */
PIC_T_Null
PICResult = success
PICResult = CS_Stop
DP_termAttempt (DPResult)
DP_termAttempt
/* Concerning Alerting, Busy, NoAnswer and Answer, see note on page 9. */
DPResult = Resume
DPResult = Alerting
DPResult = Answer
PICResult = NoAnswer
PICResult = Busy
DPResult = Failure
DPResult = ARelease
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_Authorize_ _Termination_ _Attempt (PICResult)
DP_tCall_ _Accepted (DPResult)
DP_tAnswer (DPResult)
DP_tNoAnswer (DPResult)
DP_tBusy (DPResult)
PIC_Select_ _Facility (PICResult)
DP_tAbandon (DPResult)
PIC_ _TException
PIC_Authorize_ _Termination_ _Attempt
DP_tCall_ _Accepted
DP_tAnswer
DP_tNoAnswer
DP_tBusy
PIC_Select_ _Facility
DP_tAbandon
PIC_T_Null (PICResult)
PIC_T_Null
PICResult = Success
PICResult = Failure
PICResult = ARelease
DP_termAttempt_ Authorized (DPResult)
DP_termDenied (DPResult)
DP_tAbandon (DPResult)
DP_termAttempt_ Authorized
DP_termDenied
DP_tAbandon
/* IN CS−3 New DP Termination_ Denied added after PIC Authorize_ Termination_Attempt */
/* Concerning Alerting, Busy, NoAnswer and Answer, see note on page 9. */
DPResult = Resume
DPResult = Alerting
DPResult = Answer
DPResult = NoAnswer
DPResult = Busy
DPResult = AuthTermination
DPResult = Failure
DPResult = ARelease
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_Select_ _Facility (PICResult)
DP_tCall_ _Accepted (DPResult)
DP_tAnswer (DPResult)
DP_tNoAnswer (DPResult)
DP_tBusy (DPResult)
PIC_Authorize_ _Termination_ _Attempt (PICResult)
PIC_Select_ _Facility (PICResult)
DP_tAbandon (DPResult)
PIC_ _TException
PIC_Select_ _Facility
DP_tCall_ _Accepted
DP_tAnswer
DP_tNoAnswer
DP_tBusy
PIC_Authorize_ _Termination_ _Attempt
PIC_Select_ _Facility
DP_tAbandon
PIC_T_Null (PICResult)
PIC_T_Null


Process Type TerminatingBCSM
6(13)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */
/* The SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_B) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_A) the BCSM is released. */
PIC_Select_ _Facility
PICResult = Success
PICResult = Busy
DP_Facility_ _Selected_and_ _Available (DPResult)
DP_tBusy (DPResult)
DP_Facility_ _Selected_and_ _Available
DP_tBusy
/* Concerning Alerting, Busy, NoAnswer and Answer, see note on page 9. */
DPResult = Resume
DPResult = Alerting
DPResult = Answer
DPResult = ARelease
DPResult = NoAnswer
DPResult = Busy
DPResult = Failure
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_Present_ _Call (PICResult)
DP_tCall_ _Accepted (DPResult)
DP_tAnswer (DPResult)
DP_tAbandon (DPResult)
DP_tNoAnswer (DPResult)
DP_tBusy (DPResult)
PIC_Select_ _Facility (PICResult)
PIC_ _TException
DP_tCall_ _Accepted
DP_tAnswer
DP_tAbandon
DP_tNoAnswer
DP_tBusy
PIC_Select_ _Facility
PIC_T_Null (PICResult)
PIC_T_Null
PIC_Present_ _Call
PICResult = Answer
PICResult = NoAnswer
PICResult = Busy
PICResult = Failure
PICResult = ARelease
PICResult = Exception
PICResult = SCFRelease
PICResult = BCSM_Stop
DP_tAnswer (DPResult)
DP_tNoAnswer (DPResult)
DP_tBusy (DPResult)
PIC_Select_ _Facility (PICResult)
DP_tAbandon (DPResult)
PIC_ _TException
PIC_Disconnect (SCFRelease)
DP_tAnswer
DP_tNoAnswer
DP_tBusy
PIC_Select_ _Facility
DP_tAbandon
PIC_T_Null (PICResult)
PIC_T_Null

Process Type TerminatingBCSM 7(13)



Process Type TerminatingBCSM 8(13)



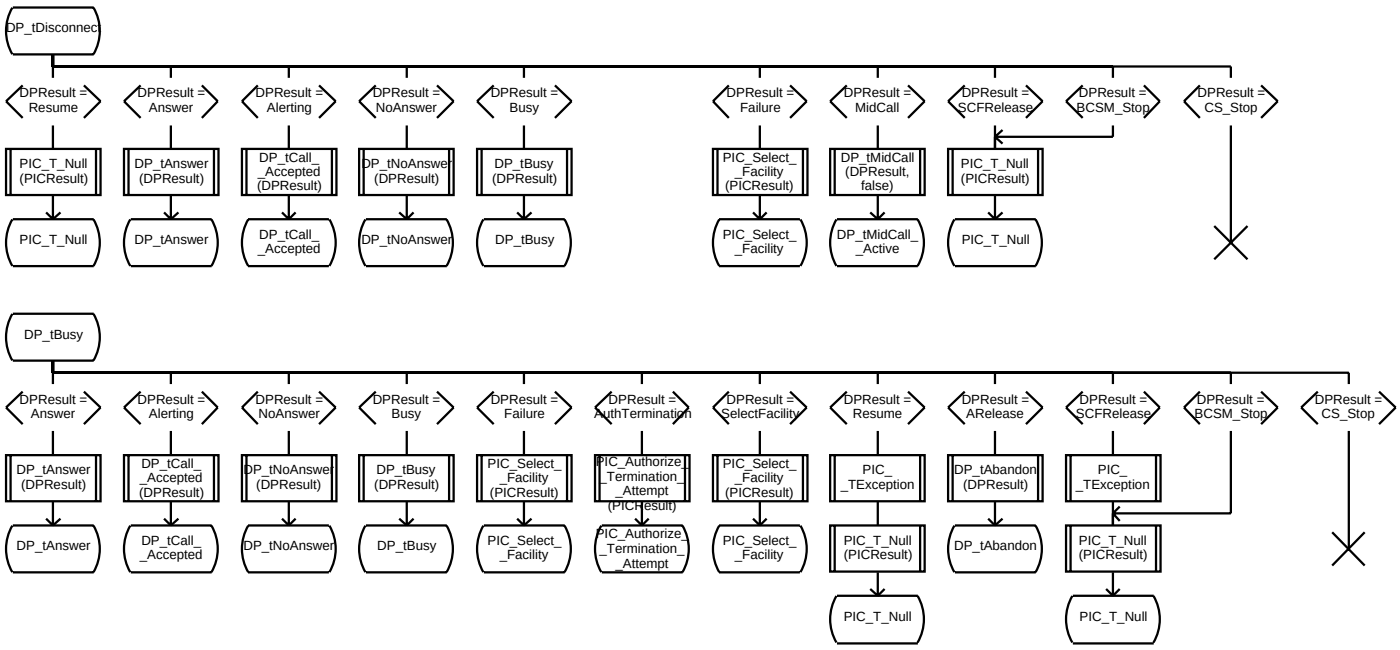
Process Type TerminatingBCSM 9(13)

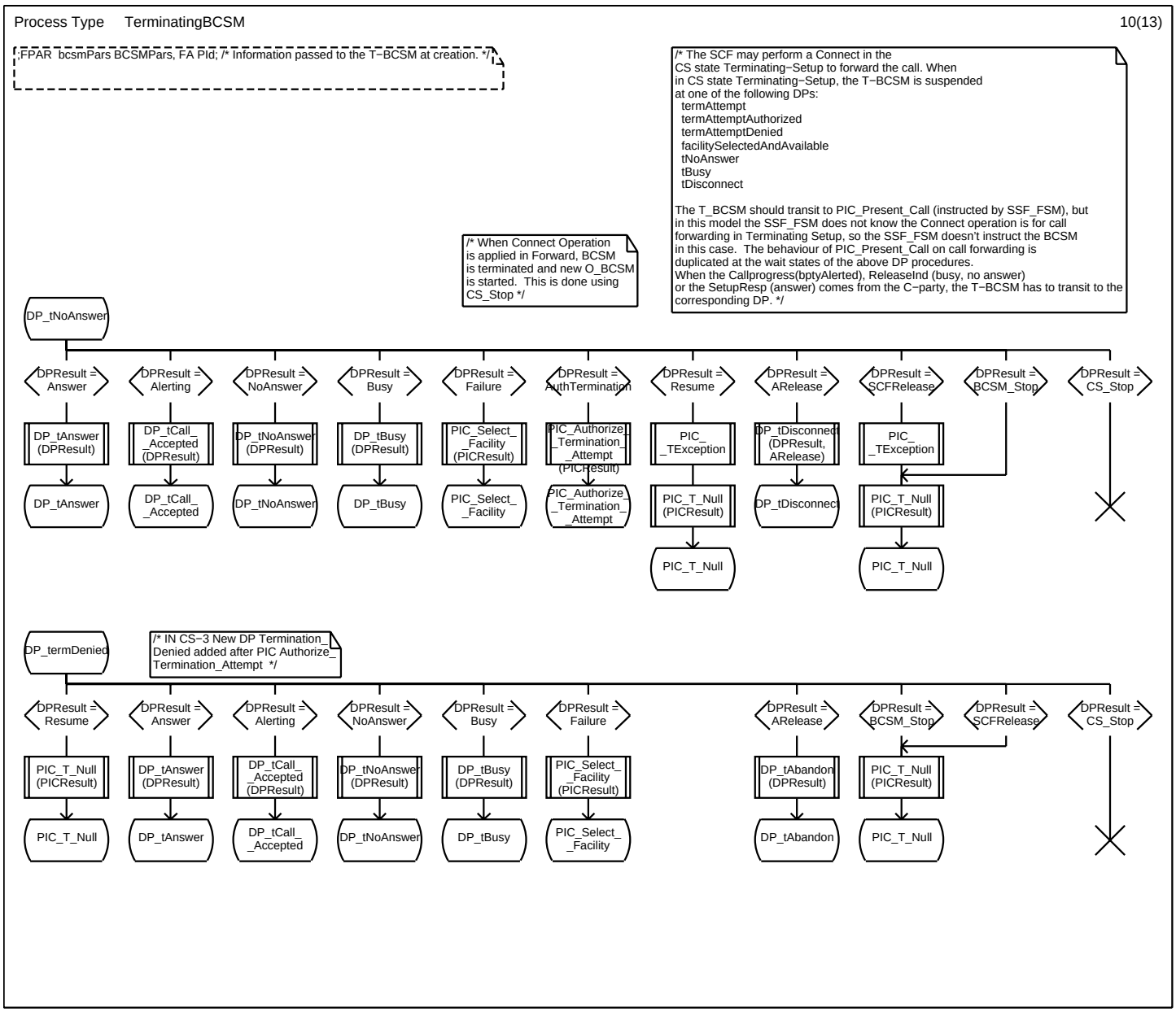
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */

/* 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
termAttemptDenied
facilitySelectedAndAvailable
tNoAnswer
tBusy
tDisconnect

The T_BCSM should transit to PIC_Present_Call (instructed by SSF_FSM), but
in this model the SSF_FSM does not know the Connect operation is for call
forwarding in Terminating Setup, so the SSF_FSM doesn't instruct the BCSM
in this case. The behaviour of PIC_Present_Call on call forwarding is
duplicated at the wait states of the above DP procedures.
When the Callprogress(bptyAlerted), ReleaseInd (busy, no answer)
or the SetupResp (answer) comes from the C−party, the T−BCSM has to transit to the
corresponding DP. */

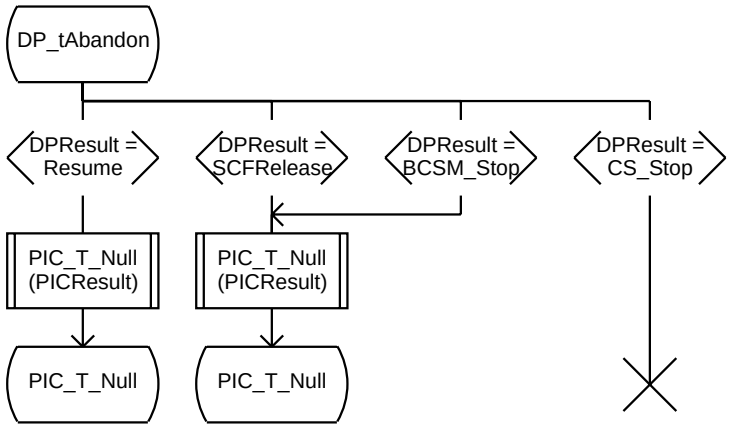
/* When Connect Operation
is applied in Forward, BCSM
is terminated and new O_BCSM
is started. This is done using
CS_Stop */

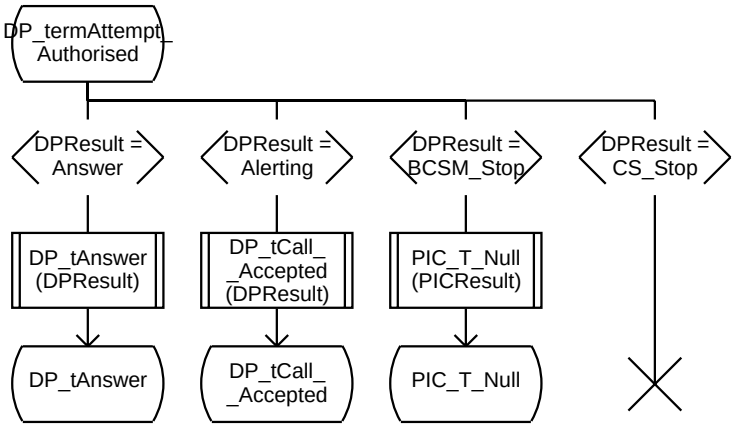


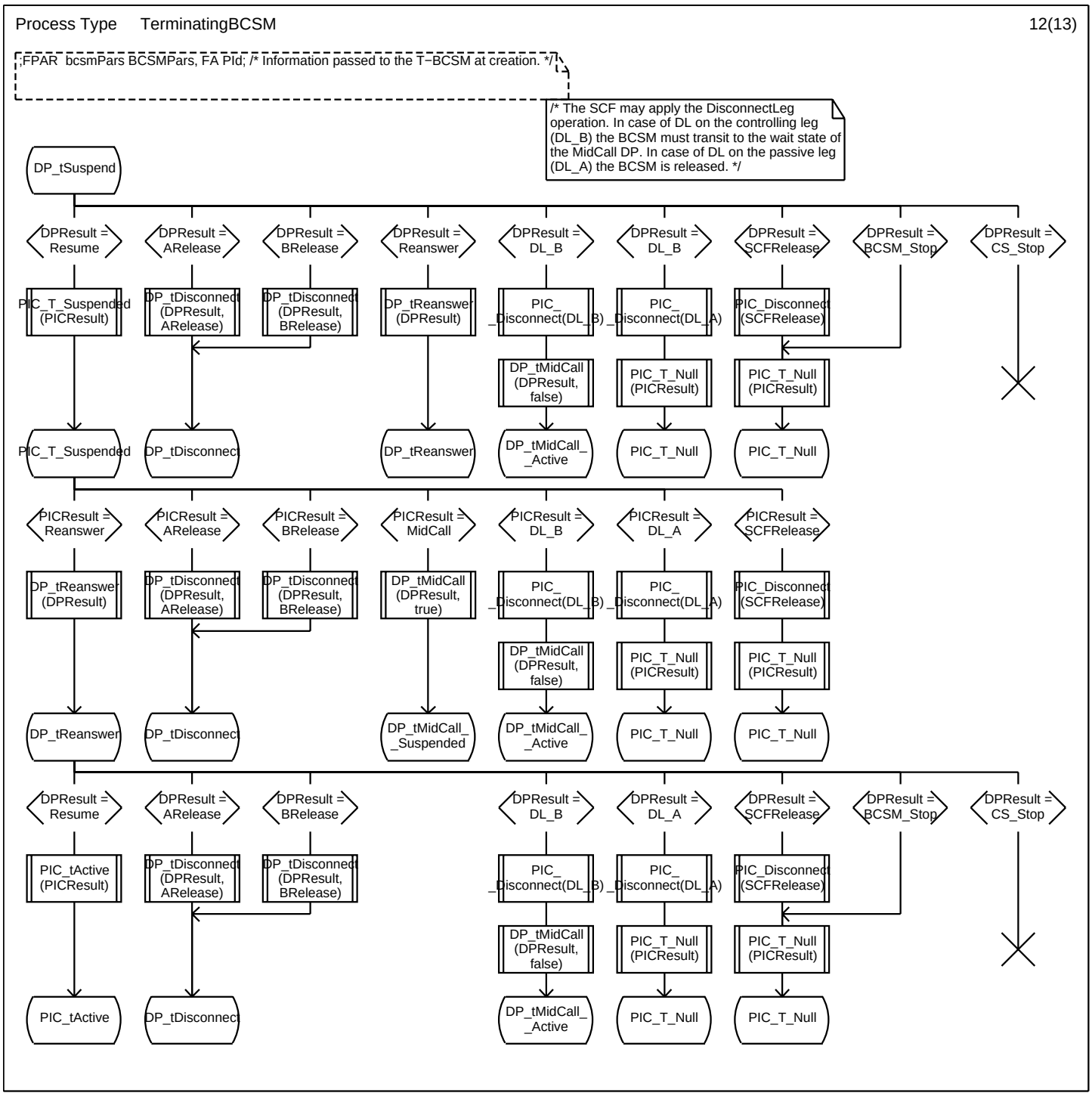

Process Type TerminatingBCSM
10(13)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */
/* The SCF may perform a Connect in the CS state Terminating−Setup to forward the call. When in CS state Terminating−Setup, the T−BCSM is suspended at one of the following DPs:
termAttempt
termAttemptAuthorized
termAttemptDenied
facilitySelectedAndAvailable
tNoAnswer
tBusy
tDisconnect
The T_BCSM should transit to PIC_Present_Call (instructed by SSF_FSM), but in this model the SSF_FSM does not know the Connect operation is for call forwarding in Terminating Setup, so the SSF_FSM doesn't instruct the BCSM in this case. The behaviour of PIC_Present_Call on call forwarding is duplicated at the wait states of the above DP procedures.
When the Callprogress(bptyAlerted), ReleaseInd (busy, no answer) or the SetupResp (answer) comes from the C−party, the T−BCSM has to transit to the corresponding DP. */
/* When Connect Operation is applied in Forward, BCSM is terminated and new O_BCSM is started. This is done using CS_Stop */
DP_tNoAnswer
DPResult = Answer
DPResult = Alerting
DPResult = NoAnswer
DPResult = Busy
DPResult = Failure
DPResult = AuthTermination
DPResult = Resume
DPResult = ARelease
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
DP_tAnswer (DPResult)
DP_tCall_ _Accepted (DPResult)
DP_tNoAnswer (DPResult)
DP_tBusy (DPResult)
PIC_Select_ _Facility (PICResult)
PIC_Authorize_ _Termination_ _Attempt (PICResult)
PIC_ _TException
DP_tDisconnect (DPResult, ARelease)
PIC_ _TException
DP_tAnswer
DP_tCall_ _Accepted
DP_tNoAnswer
DP_tBusy
PIC_Select_ _Facility
PIC_Authorize_ _Termination_ _Attempt
PIC_T_Null (PICResult)
DP_tDisconnect
PIC_T_Null (PICResult)
PIC_T_Null
PIC_T_Null
DP_termDenied
/* IN CS−3 New DP Termination_ Denied added after PIC Authorize_ Termination_Attempt */
DPResult = Resume
DPResult = Answer
DPResult = Alerting
DPResult = NoAnswer
DPResult = Busy
DPResult = Failure
DPResult = ARelease
DPResult = BCSM_Stop
DPResult = SCFRelease
DPResult = CS_Stop
PIC_T_Null (PICResult)
DP_tAnswer (DPResult)
DP_tCall_ _Accepted (DPResult)
DP_tNoAnswer (DPResult)
DP_tBusy (DPResult)
PIC_Select_ _Facility (PICResult)
DP_tAbandon (DPResult)
PIC_T_Null (PICResult)
PIC_T_Null
DP_tAnswer
DP_tCall_ _Accepted
DP_tNoAnswer
DP_tBusy
PIC_Select_ _Facility
DP_tAbandon
PIC_T_Null

Process Type TerminatingBCSM 11(13)

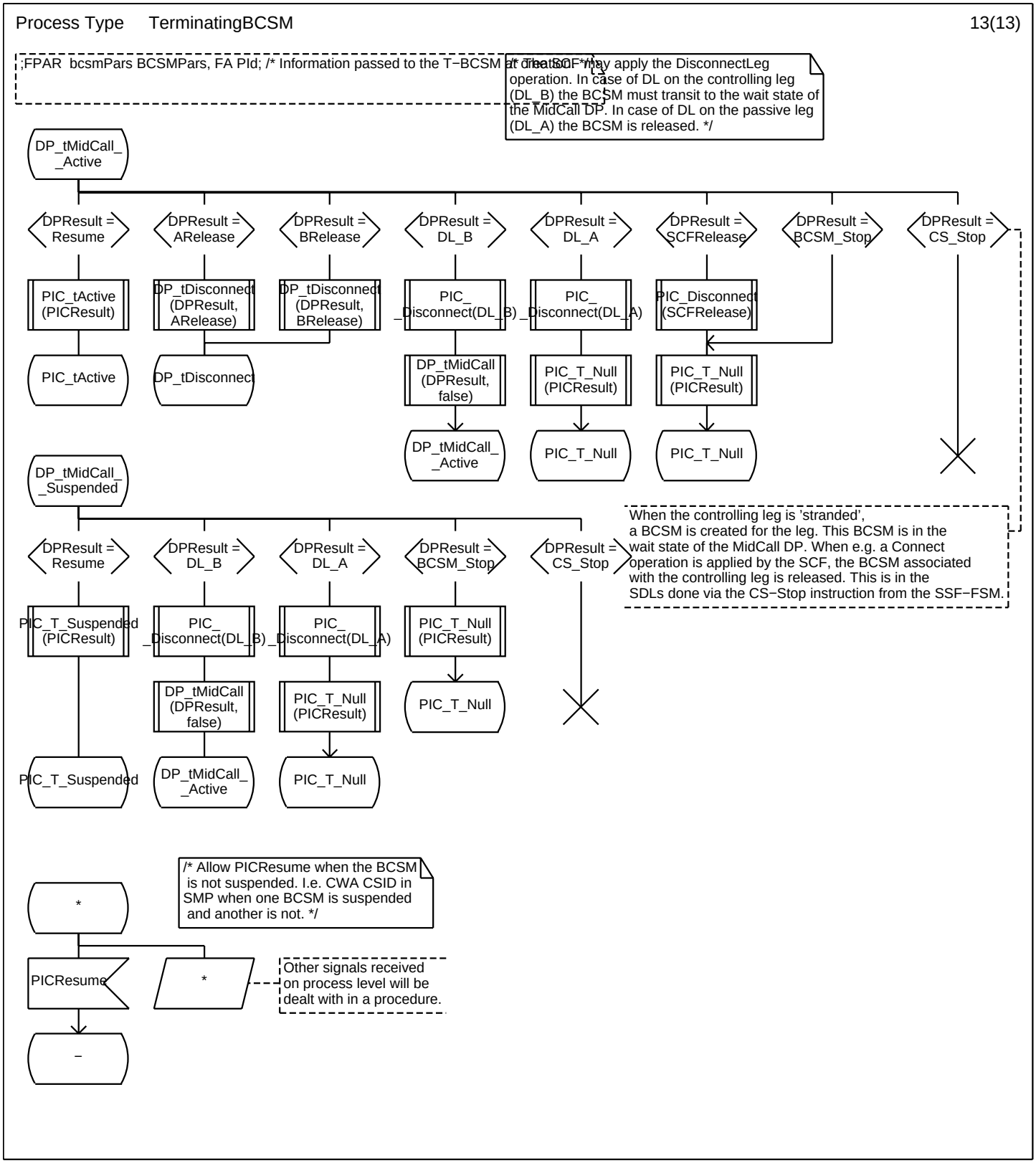
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */



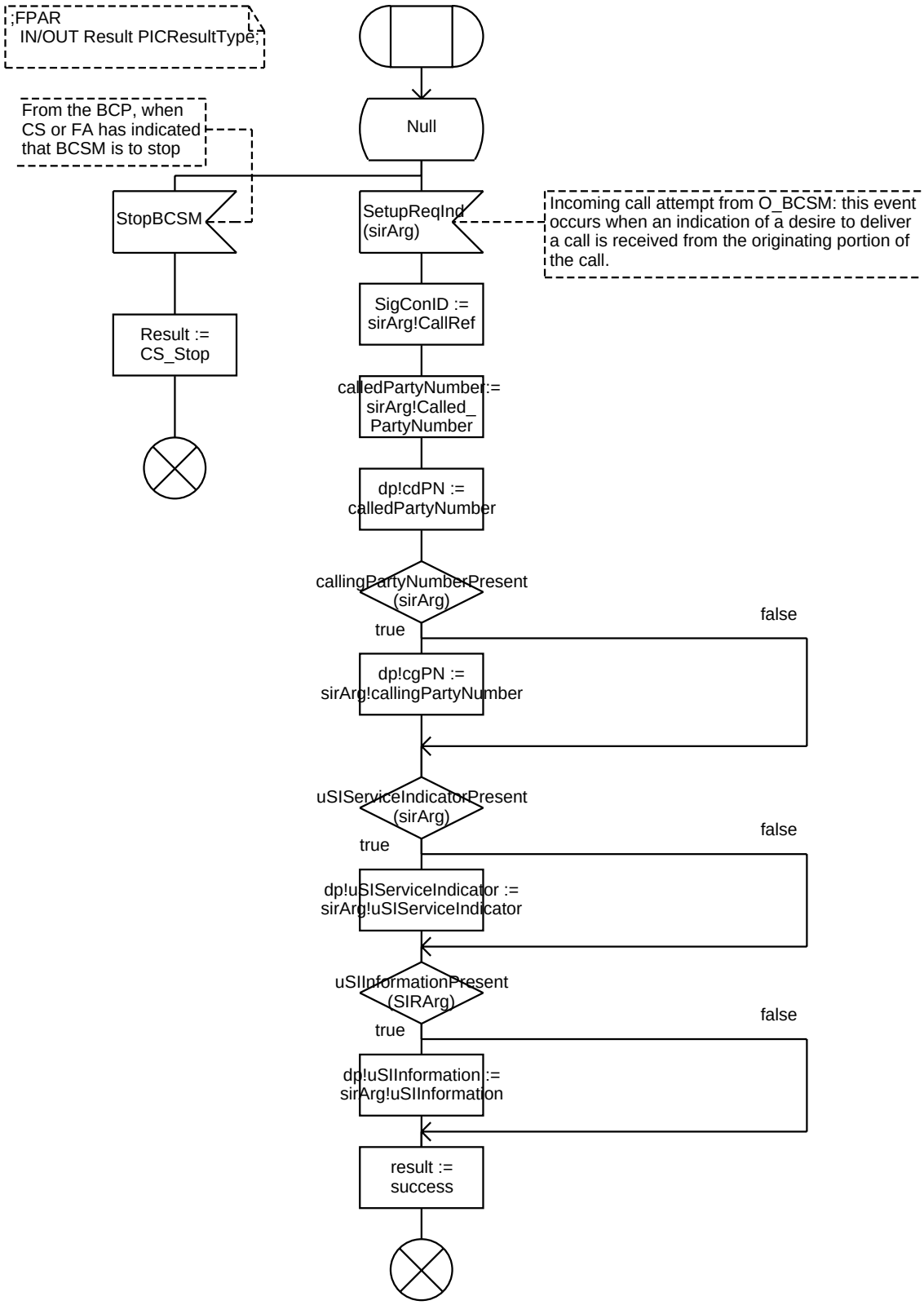



Process Type TerminatingBCSM
12(13)
;FPAR bcsmPars BCSMPars, FA PId; /* Information passed to the T−BCSM at creation. */
/* The SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_B) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_A) the BCSM is released. */
DP_tSuspend
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = Reanswer
DPResult = DL_B
DPResult = DL_B
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_T_Suspended (PICResult)
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
DP_tReanswer (DPResult)
PIC_Disconnect(DL_B)
PIC_Disconnect(DL_A)
PIC_Disconnect (SCFRelease)
DP_tMidCall (DPResult, false)
PIC_T_Null (PICResult)
PIC_T_Null (PICResult)
PIC_T_Suspended
DP_tDisconnect
DP_tReanswer
DP_tMidCall_Active
PIC_T_Null
PIC_T_Null
PICResult = Reanswer
PICResult = ARelease
PICResult = BRelease
PICResult = MidCall
PICResult = DL_B
PICResult = DL_A
PICResult = SCFRelease
DP_tReanswer (DPResult)
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
DP_tMidCall (DPResult, true)
PIC_Disconnect(DL_B)
PIC_Disconnect(DL_A)
PIC_Disconnect (SCFRelease)
DP_tMidCall (DPResult, false)
PIC_T_Null (PICResult)
PIC_T_Null (PICResult)
DP_tReanswer
DP_tDisconnect
DP_tMidCall_Suspended
DP_tMidCall_Active
PIC_T_Null
PIC_T_Null
DPResult = Resume
DPResult = ARelease
DPResult = BRelease
DPResult = DL_B
DPResult = DL_A
DPResult = SCFRelease
DPResult = BCSM_Stop
DPResult = CS_Stop
PIC_tActive (PICResult)
DP_tDisconnect (DPResult, ARelease)
DP_tDisconnect (DPResult, BRelease)
PIC_Disconnect(DL_B)
PIC_Disconnect(DL_A)
PIC_Disconnect (SCFRelease)
DP_tMidCall (DPResult, false)
PIC_T_Null (PICResult)
PIC_T_Null (PICResult)
PIC_tActive
DP_tDisconnect
DP_tMidCall_Active
PIC_T_Null
PIC_T_Null

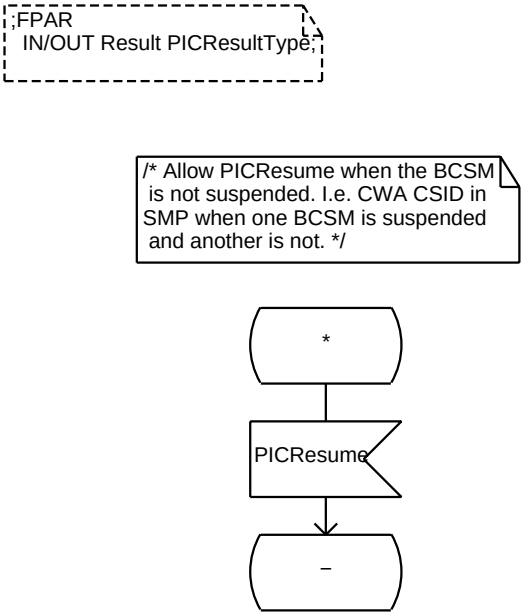
Process Type TerminatingBCSM 13(13)

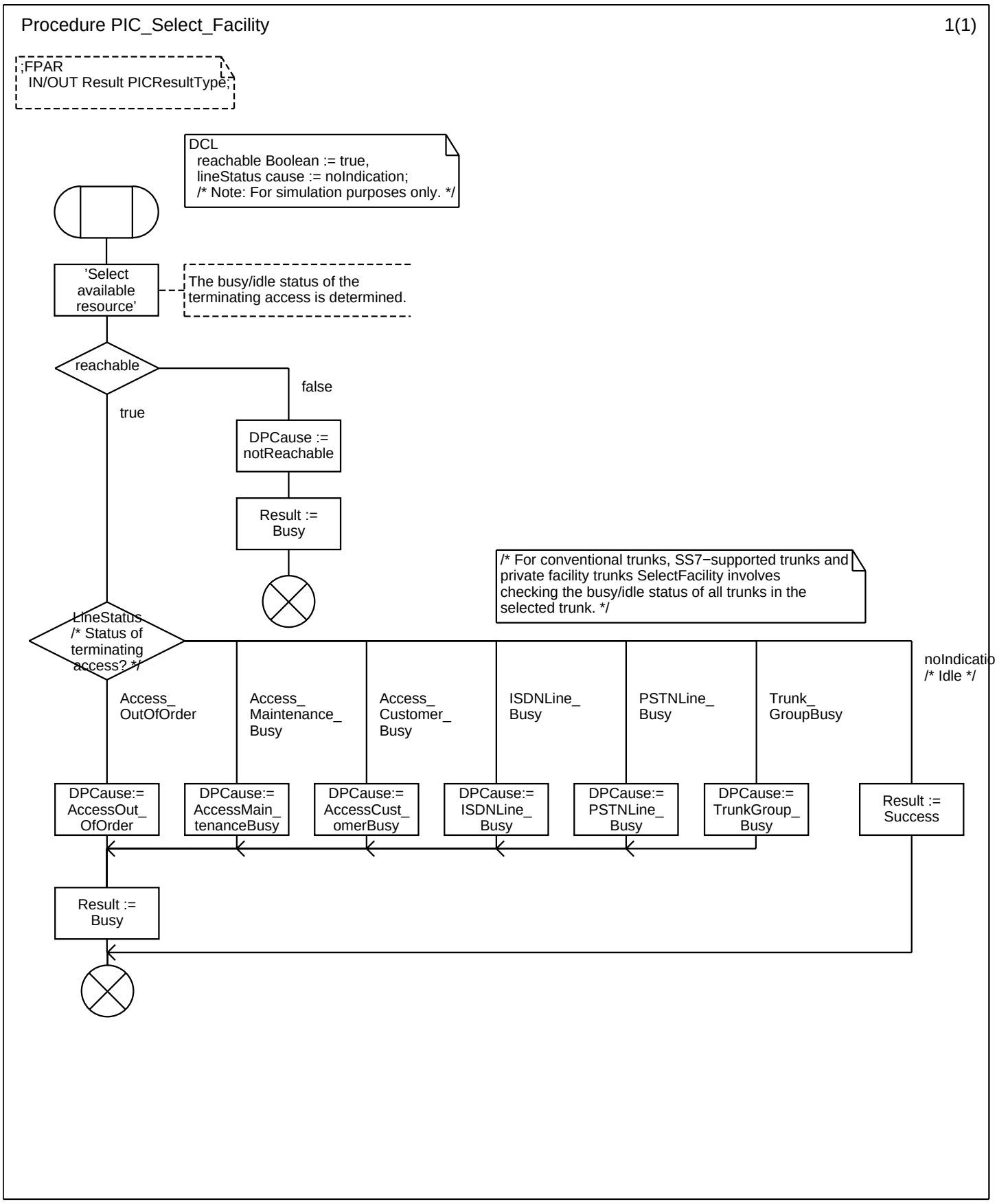


Procedure PIC_T_Null 1(2)

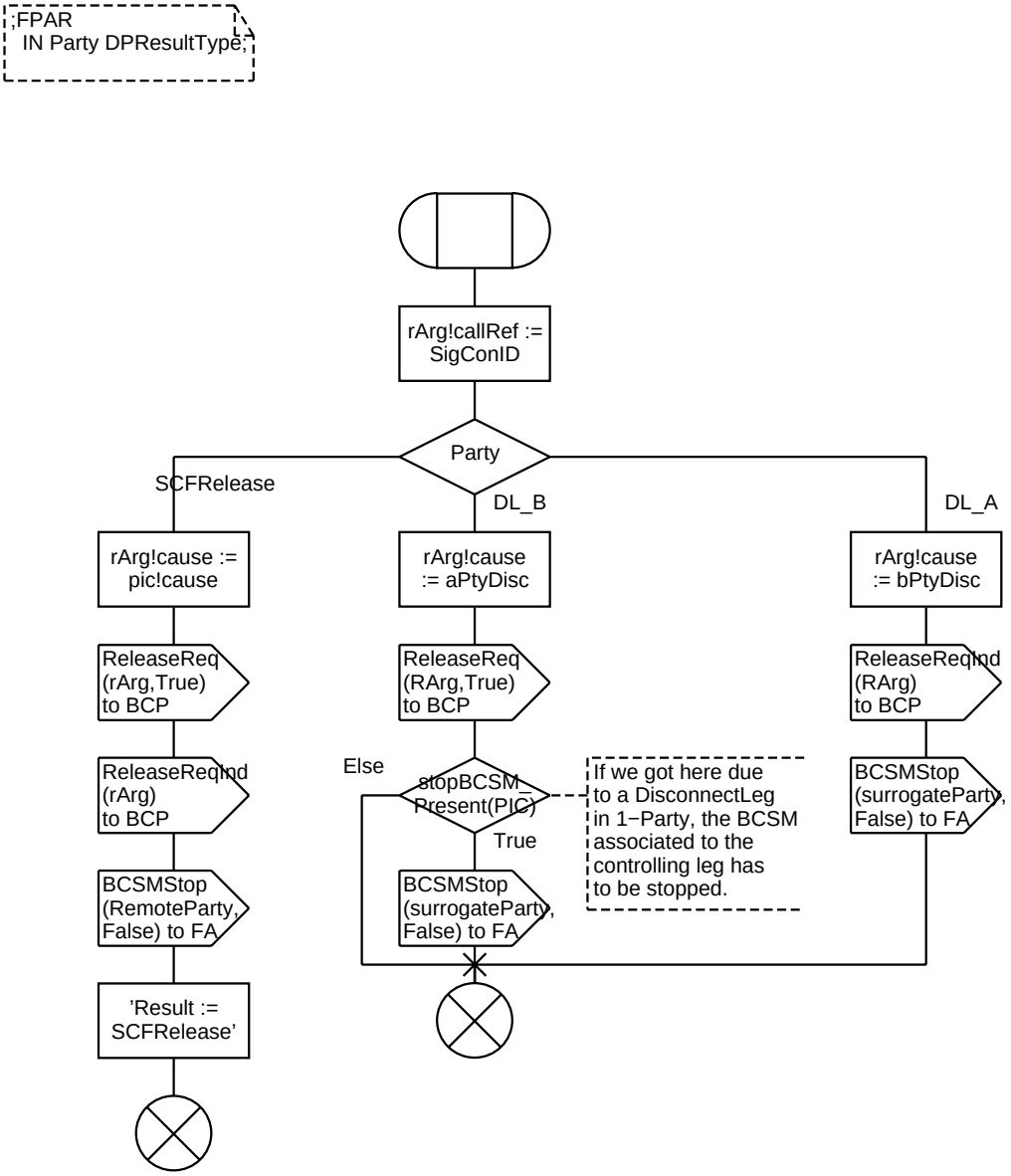


Procedure PIC_T_Null 2(2)

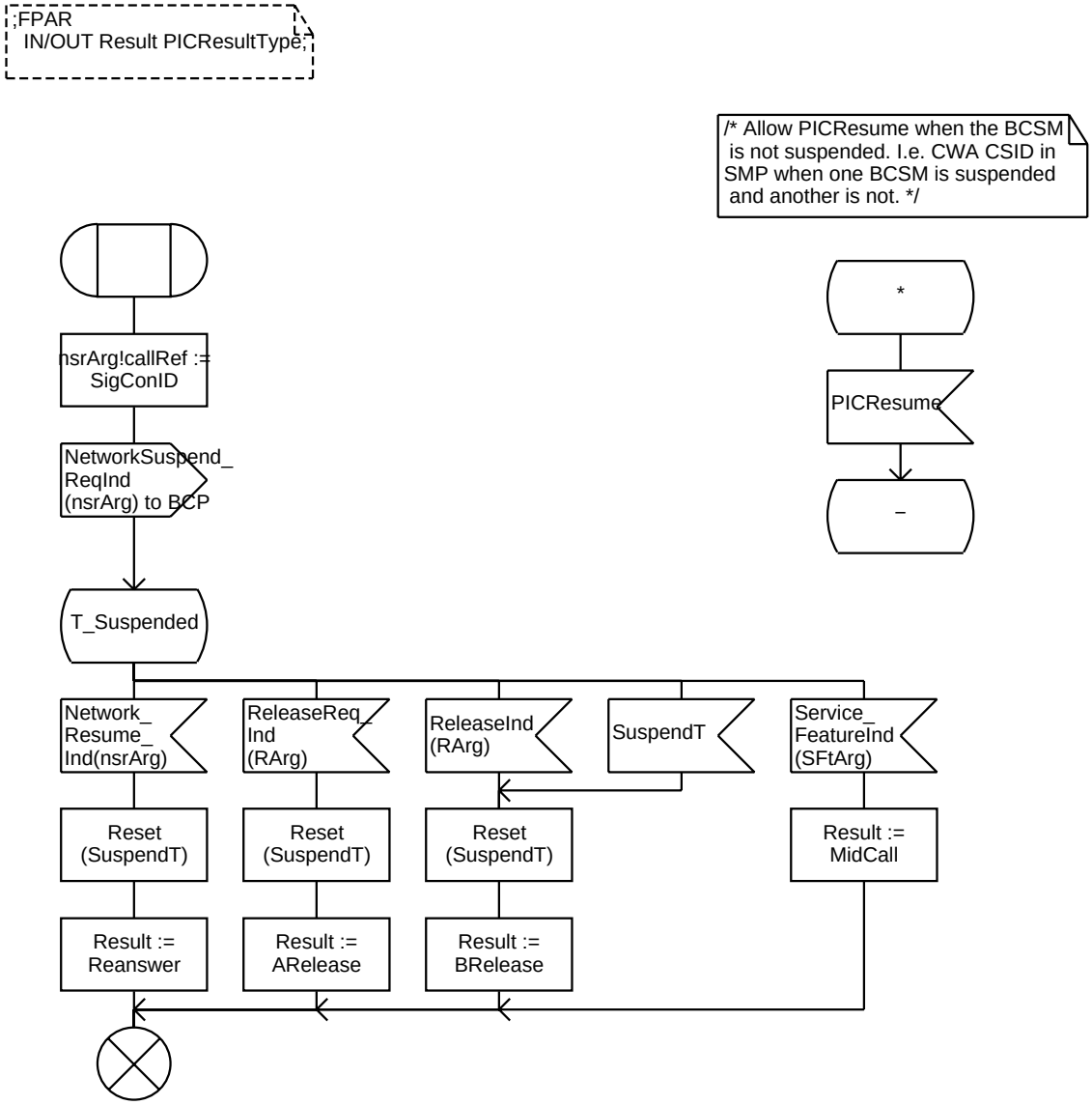



Procedure PIC_Select_Facility
1(1)
;FPAR
IN/OUT Result PICResultType;
DCL
reachable Boolean := true,
lineStatus cause := noIndication;
/* Note: For simulation purposes only. */
'Select available resource'
The busy/idle status of the terminating access is determined.
reachable
false
true
DPCause := notReachable
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? */
noIndicatio /* 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

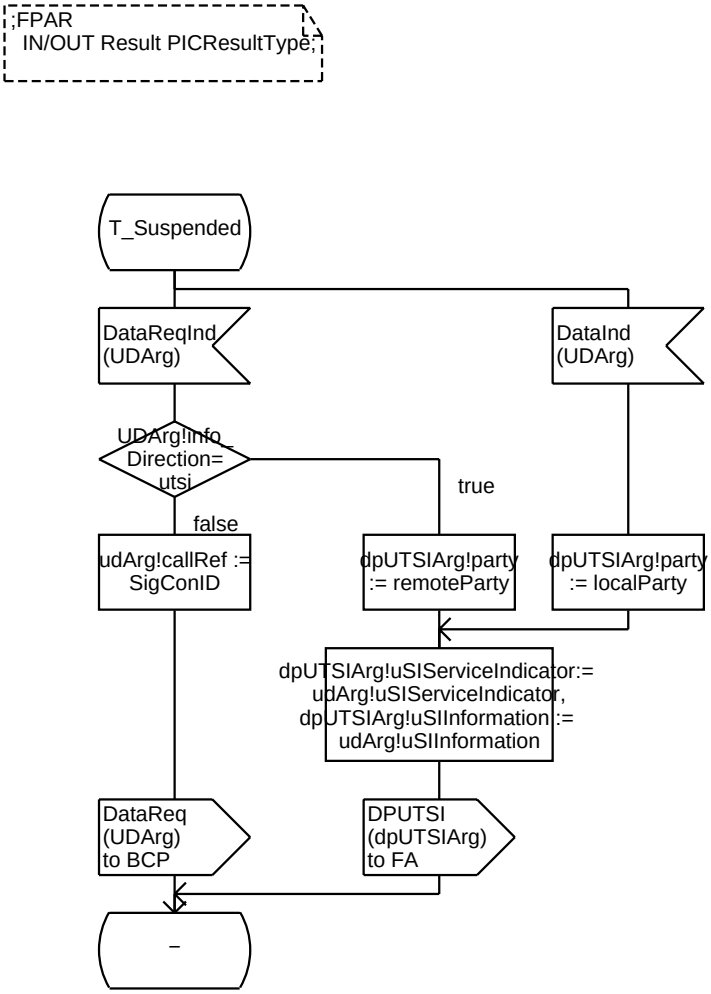
Procedure <<Process Type TerminatingBCSM>> PIC_Disconnect 1(1)



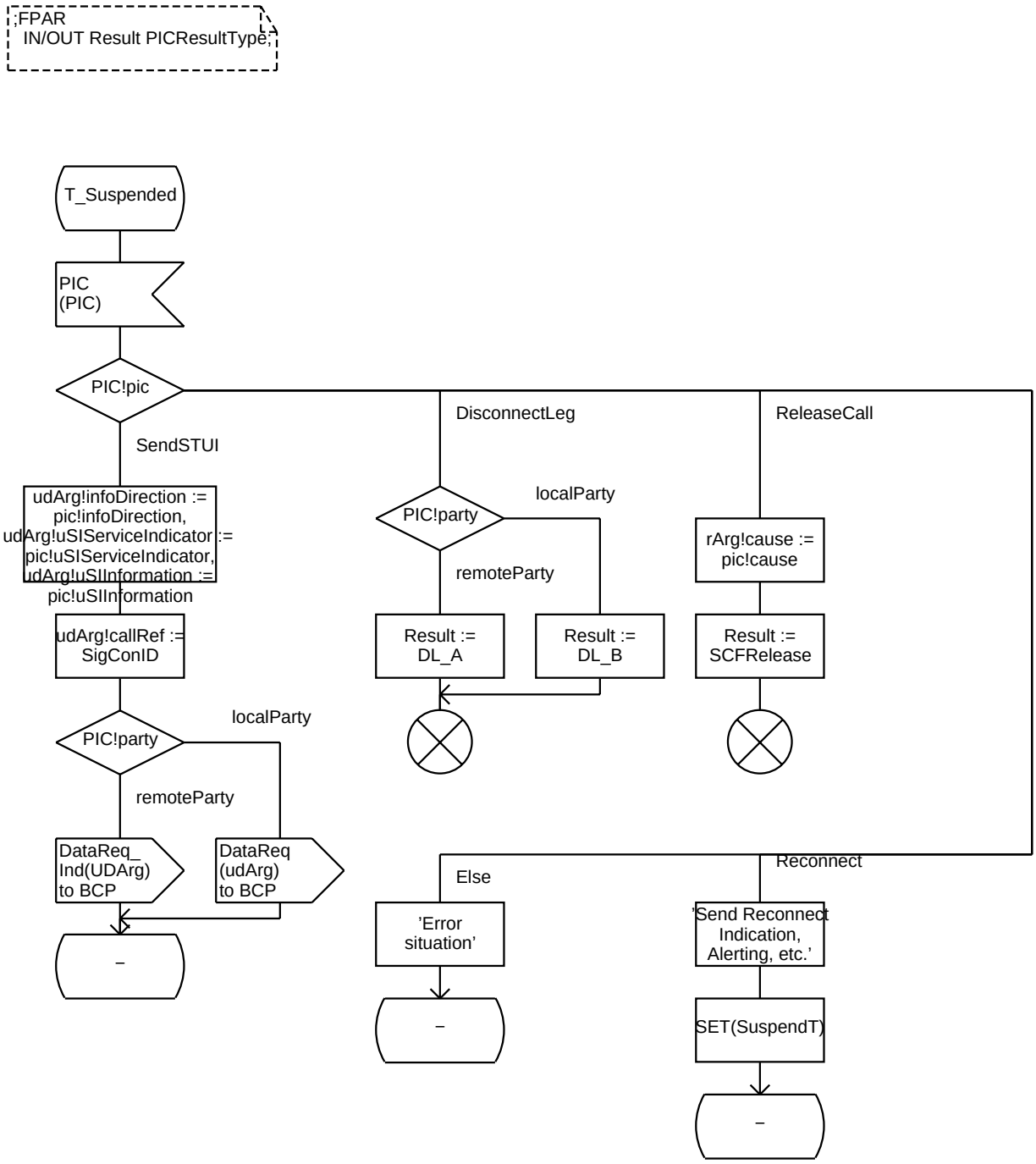
Procedure PIC_T_Suspended 1(3)



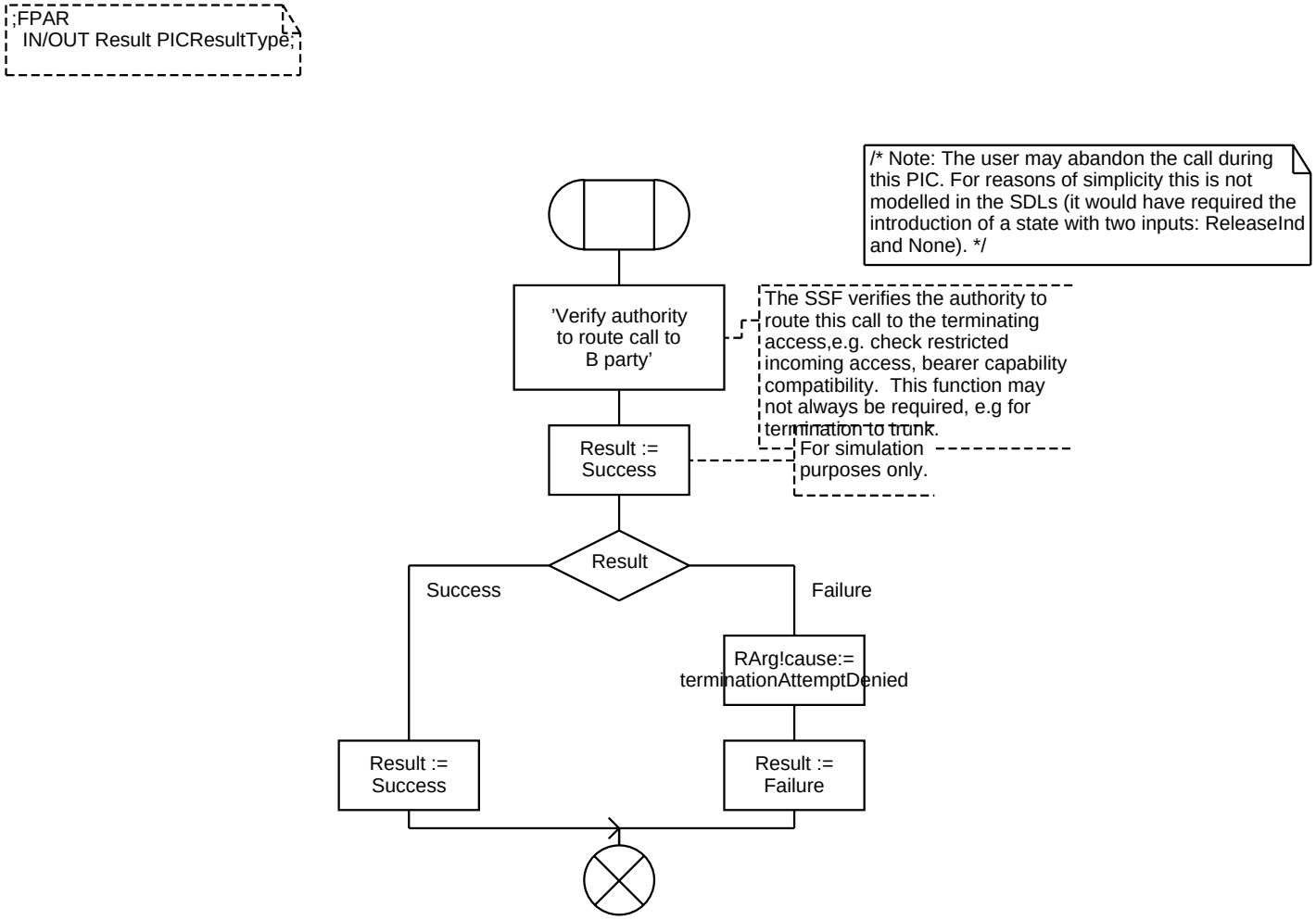
Procedure PIC_T_Suspended 2(3)



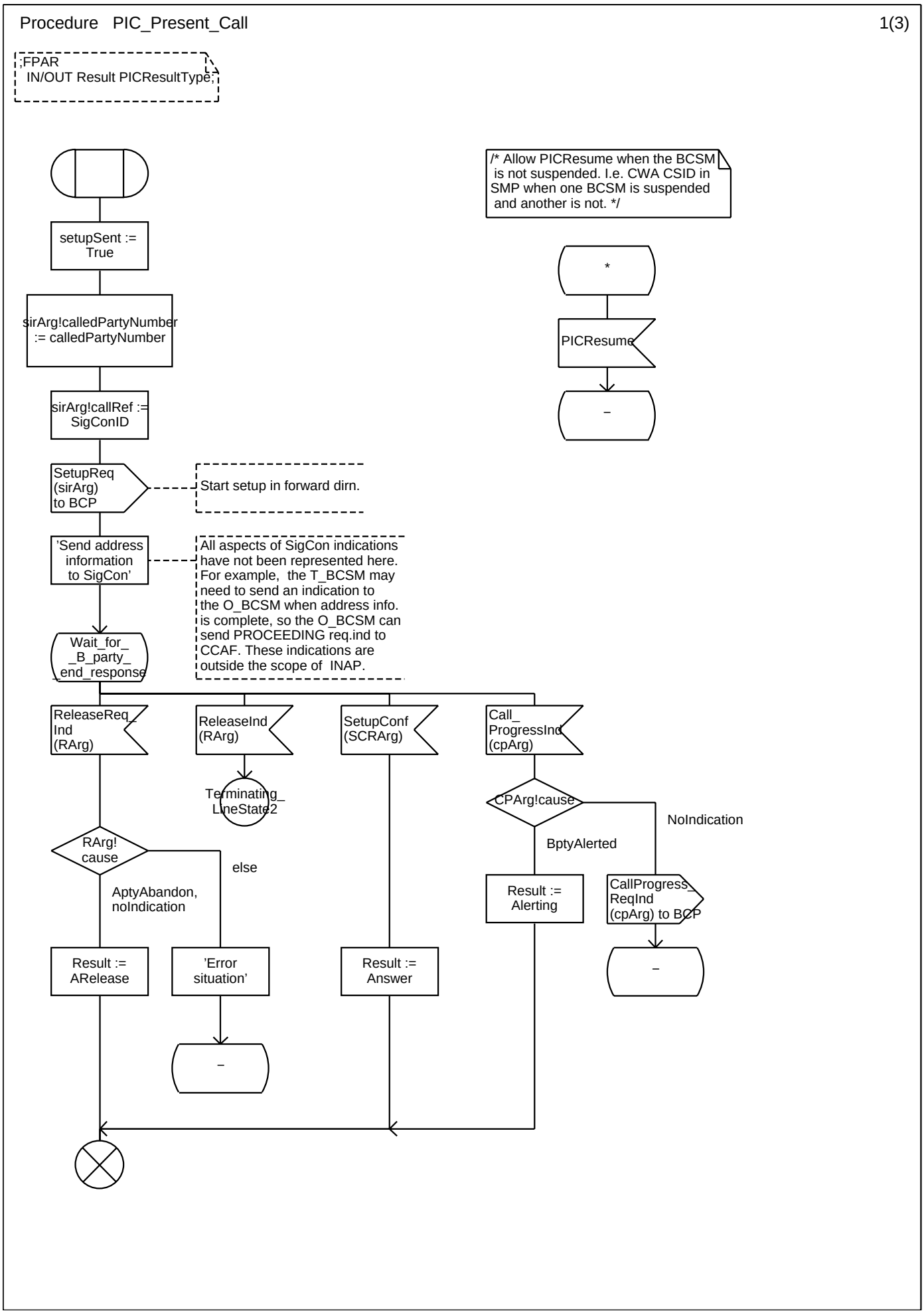
Procedure PIC_T_Suspended 3(3)



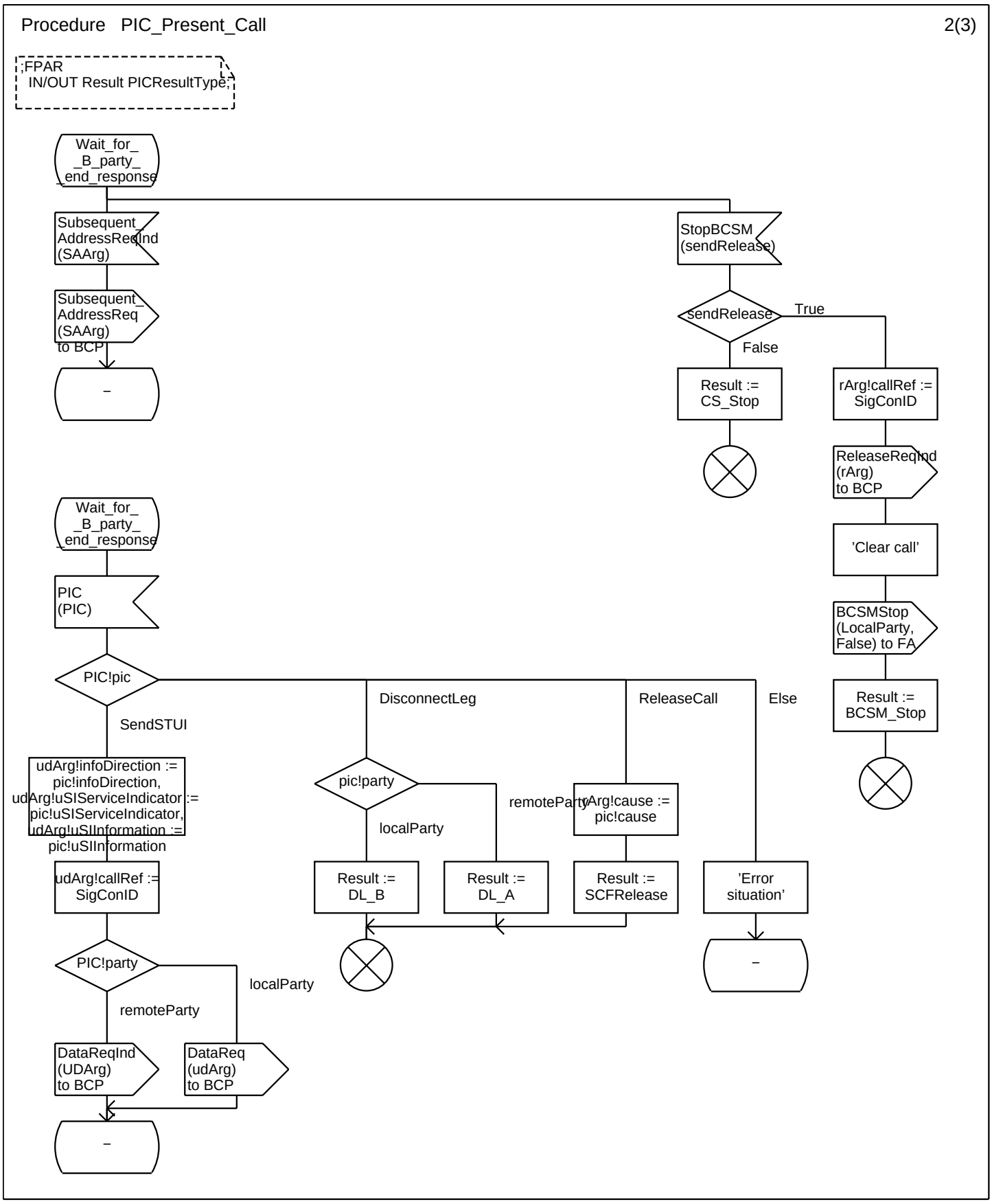
Procedure PIC_Authorize_Termination_Attempt 1(1)



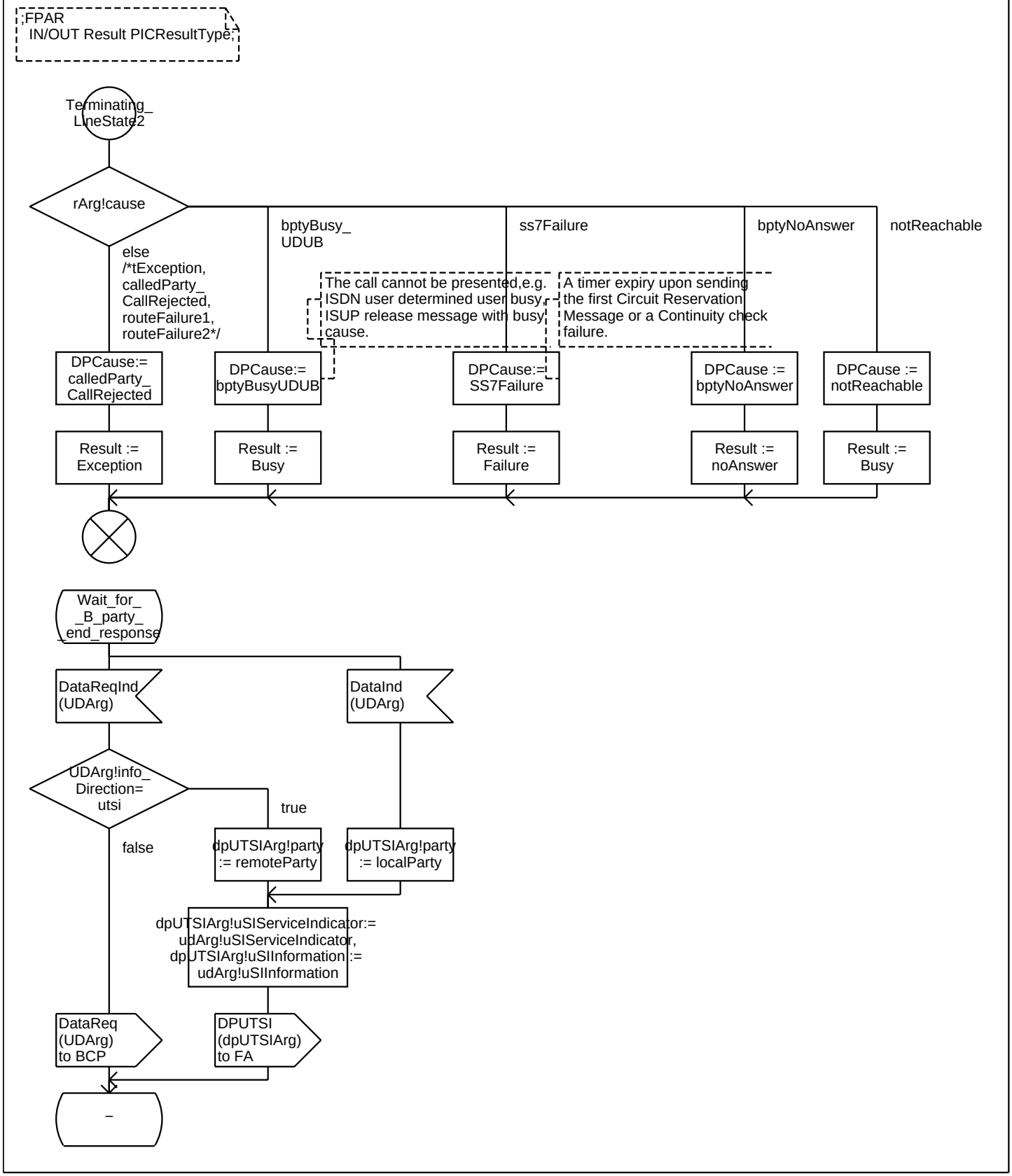
Procedure PIC_Present_Call 1(3)



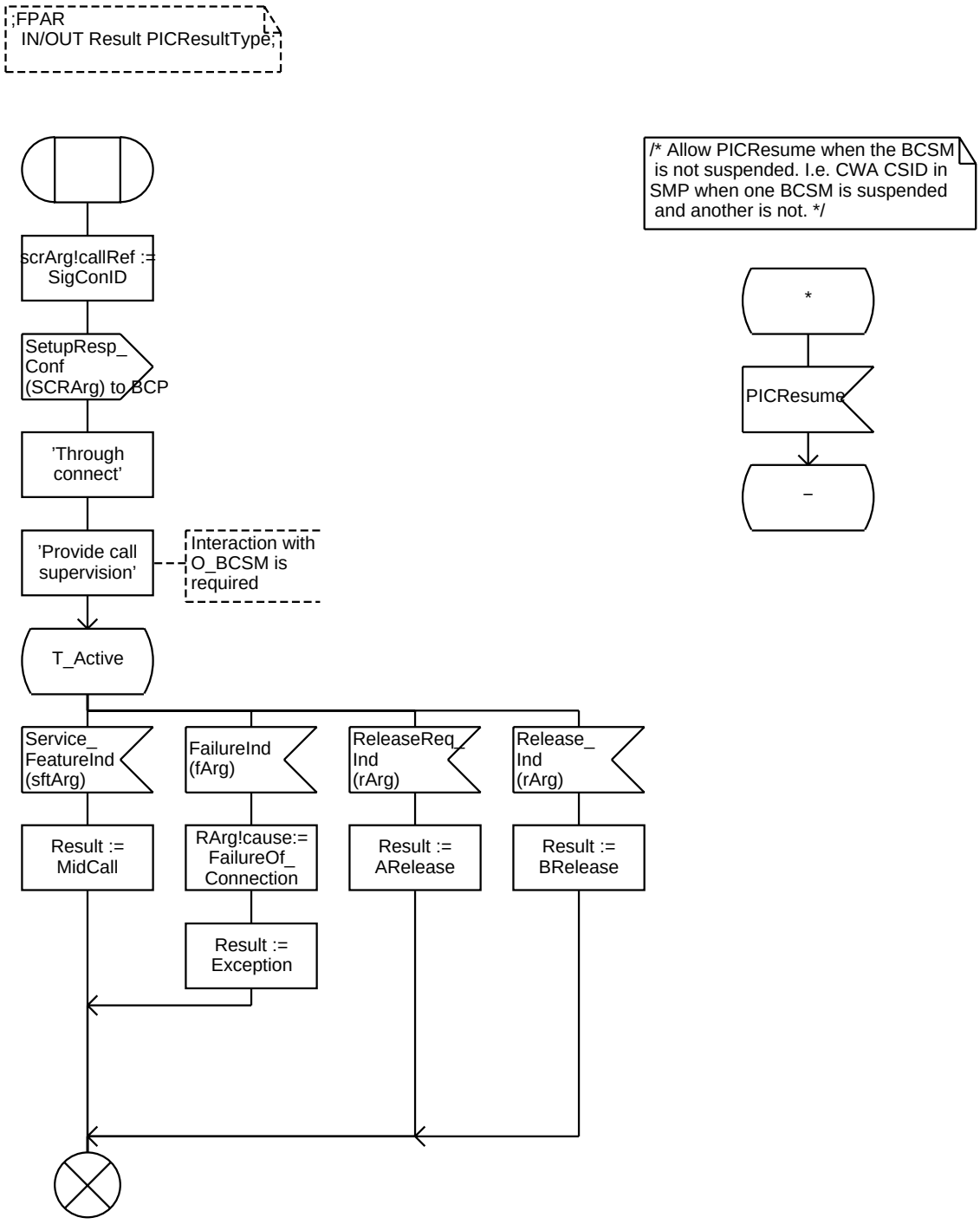
Procedure PIC_Present_Call 2(3)



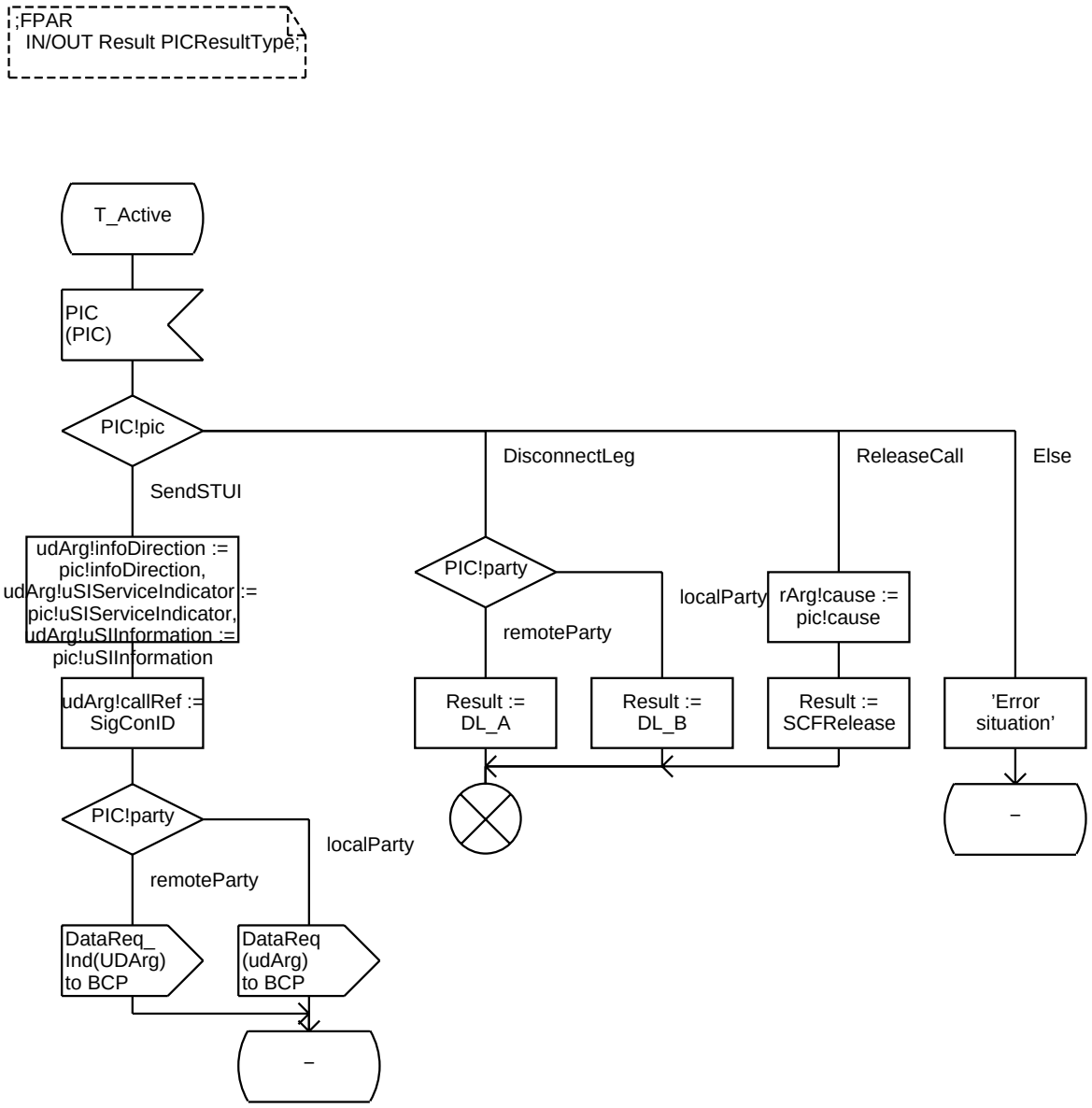
Procedure PIC_Present_Call 3(3)



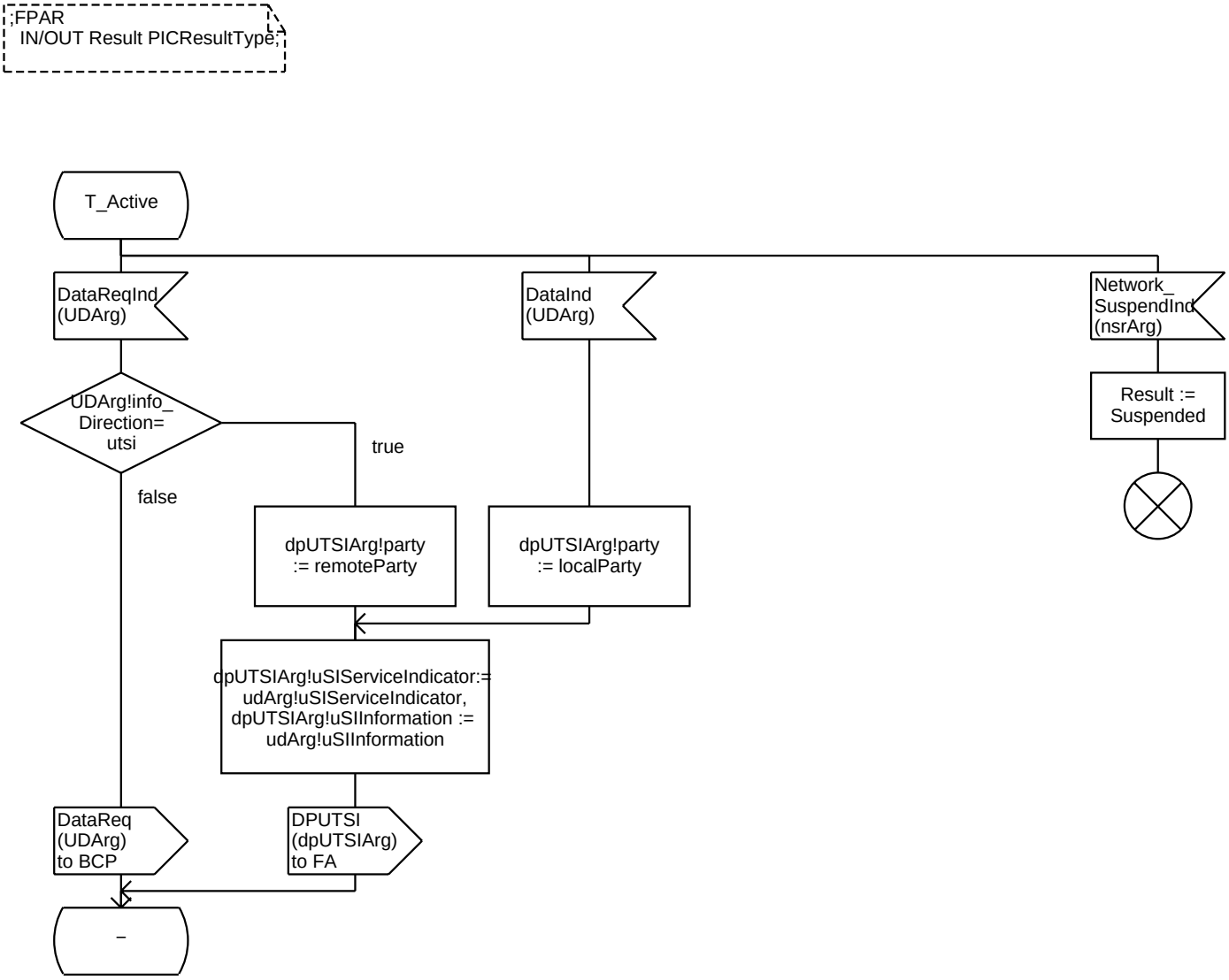
Procedure PIC_tActive 1(3)



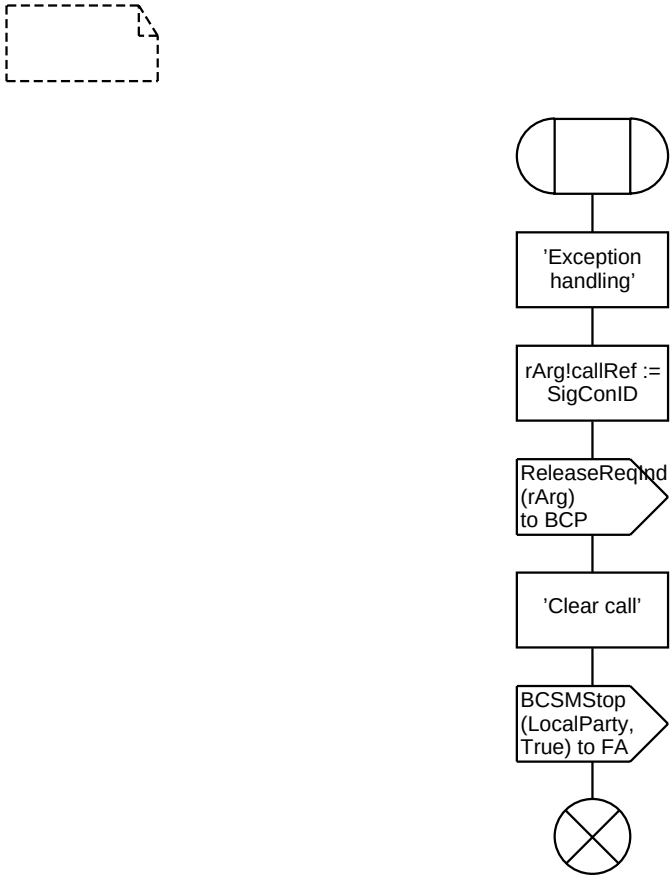
Procedure PIC_tActive 2(3)

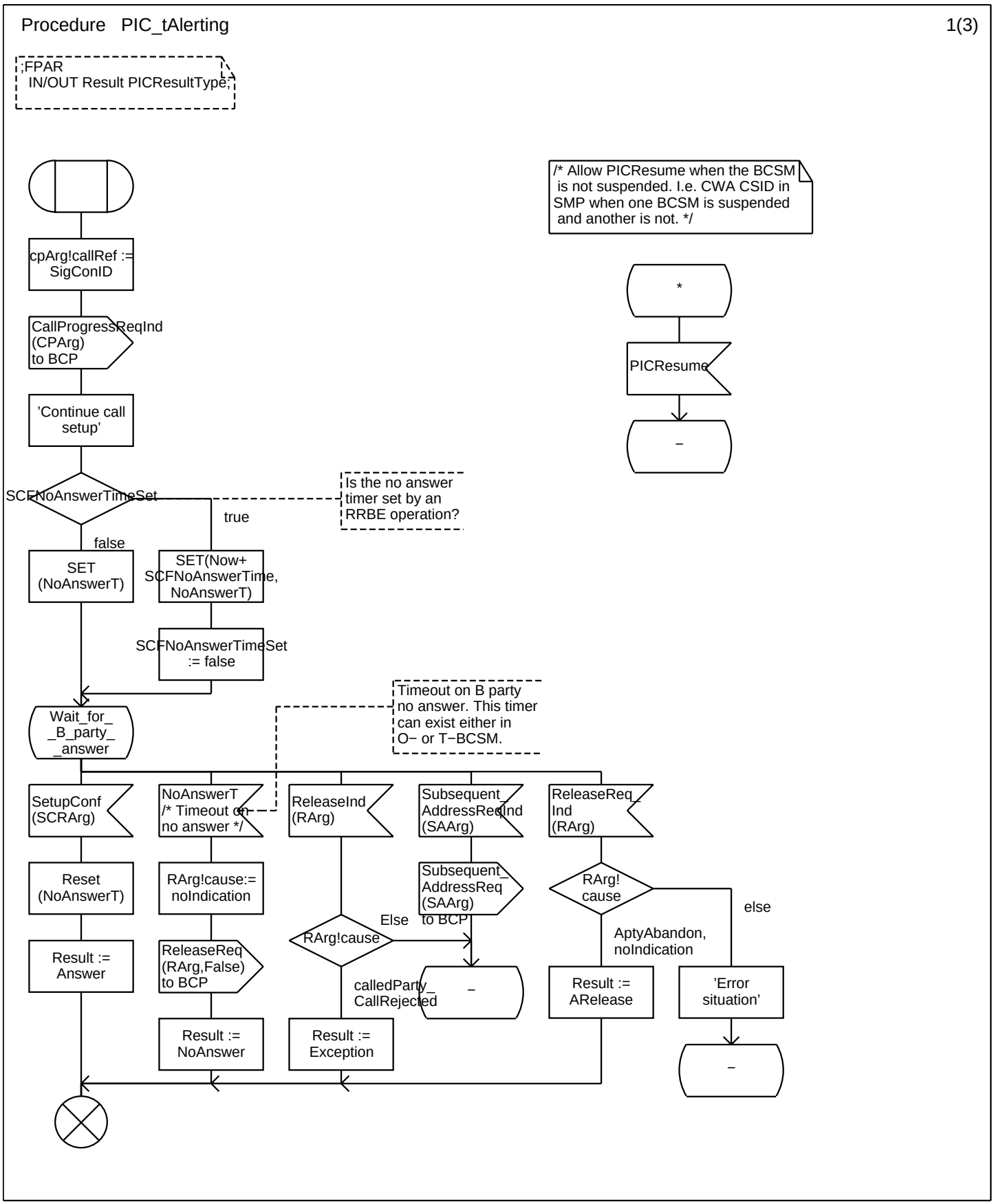


Procedure PIC_tActive 3(3)

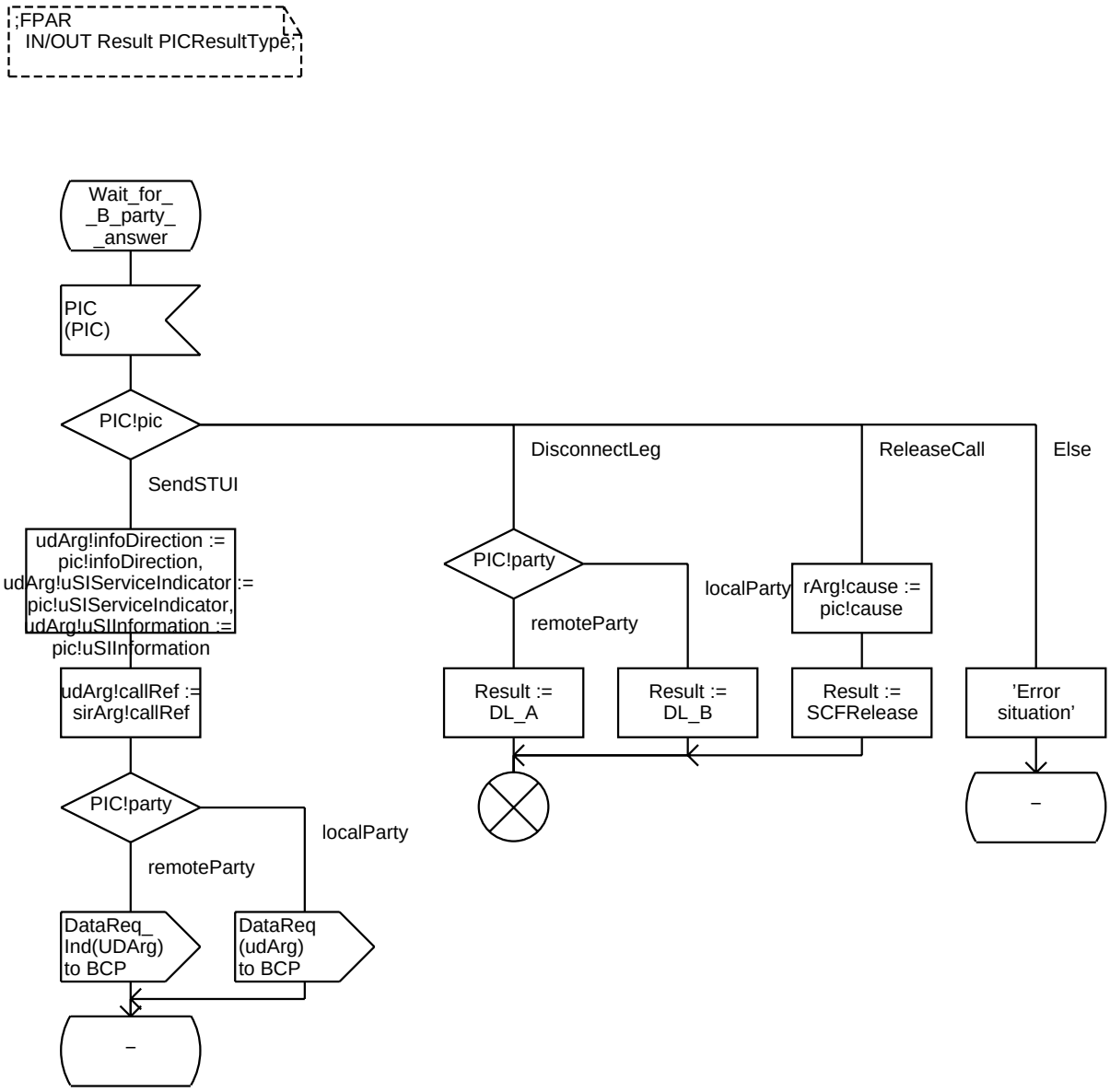


Procedure PIC_TException 1(1)

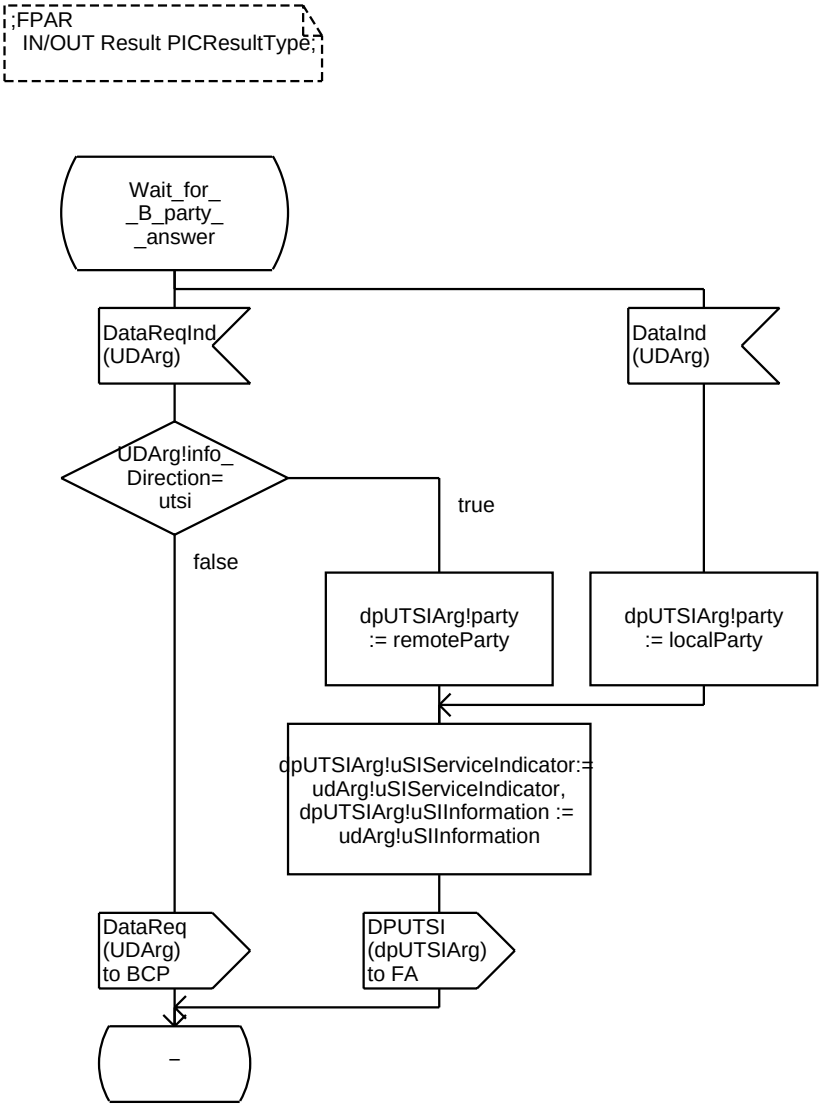



Procedure PIC_tAlerting
1(3)
;FPAR
IN/OUT Result PICResultType;
/* Allow PICResume when the BCSM is not suspended. I.e. CWA CSID in SMP when one BCSM is suspended and another is not. */
cpArg!callRef := SigConID
CallProgressReqInd (CPArg) to BCP
'Continue call setup'
SCFNoAnswerTimeSet
Is the no answer timer set by an RRBE operation?
true
false
SET (NoAnswerT)
SET(Now+ SCFNoAnswerTime, NoAnswerT)
SCFNoAnswerTimeSet := false
Wait_for_ _B_party_ _answer
Timeout on B party no answer. This timer can exist either in O− or T−BCSM.
SetupConf (SCRArg)
NoAnswerT /* Timeout on no answer */
ReleaseInd (RArg)
Subsequent_ AddressReqInd (SAArg)
ReleaseReq_ Ind (RArg)
Reset (NoAnswerT)
RArg!cause:= noIndication
Subsequent_ AddressReq (SAArg) to BCP
RArg! cause
else
Else
RArg!cause
AptyAbandon, noIndication
Result := Answer
ReleaseReq (RArg,False) to BCP
calledParty_ CallRejected
Result := ARelease
'Error situation'
Result := NoAnswer
Result := Exception
*
PICResume
–

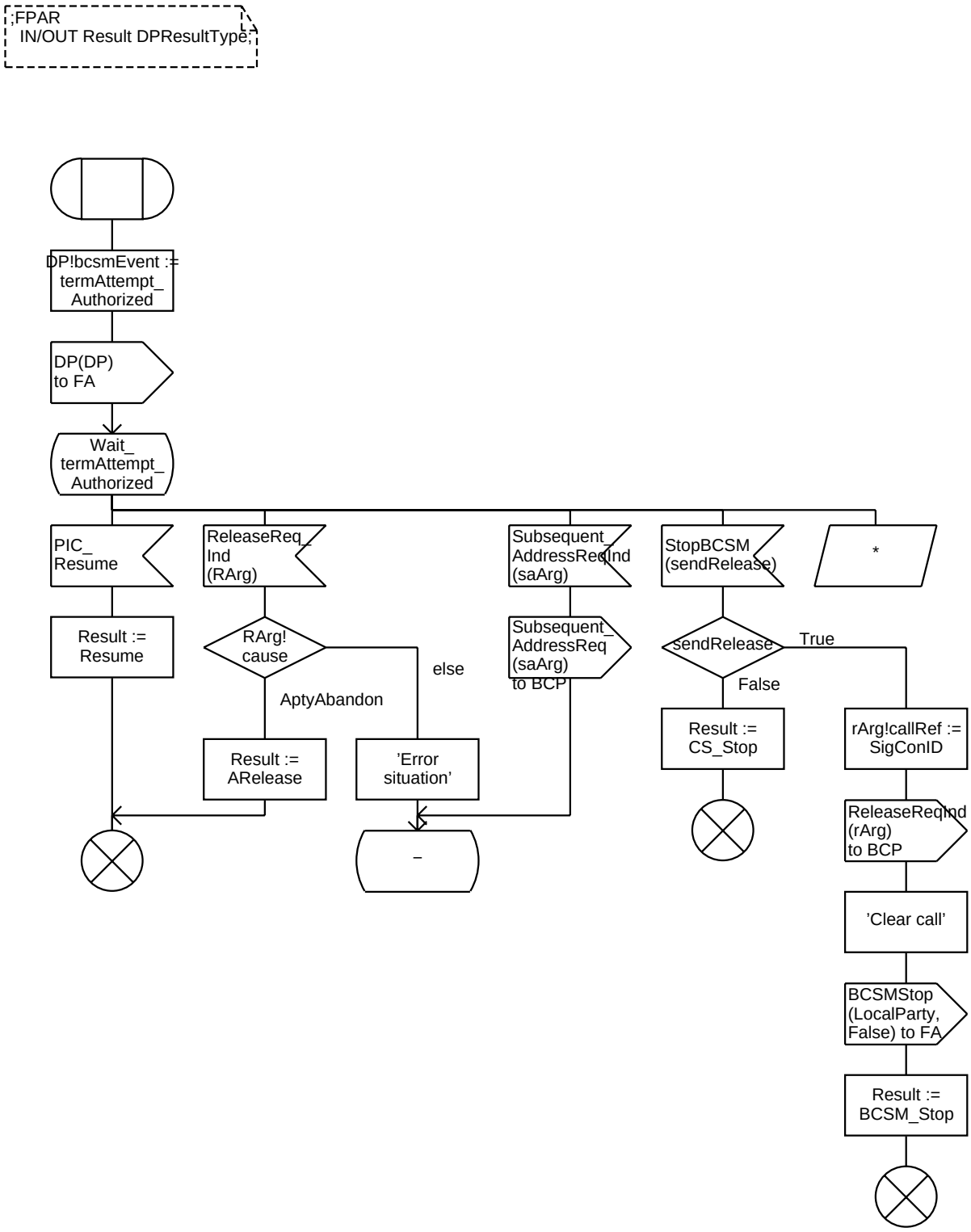
Procedure PIC_tAlerting 2(3)



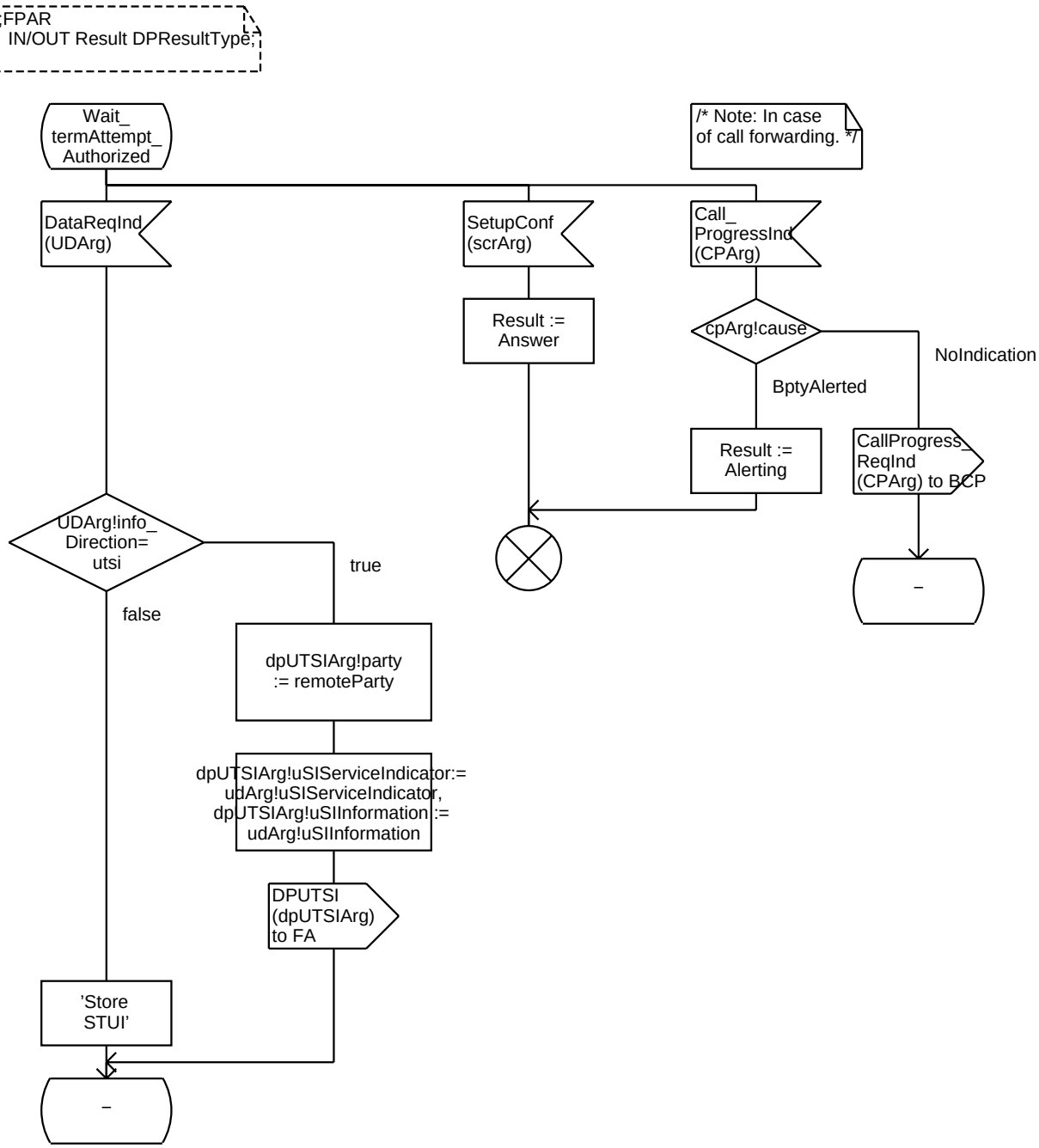
Procedure PIC_tAlerting 3(3)



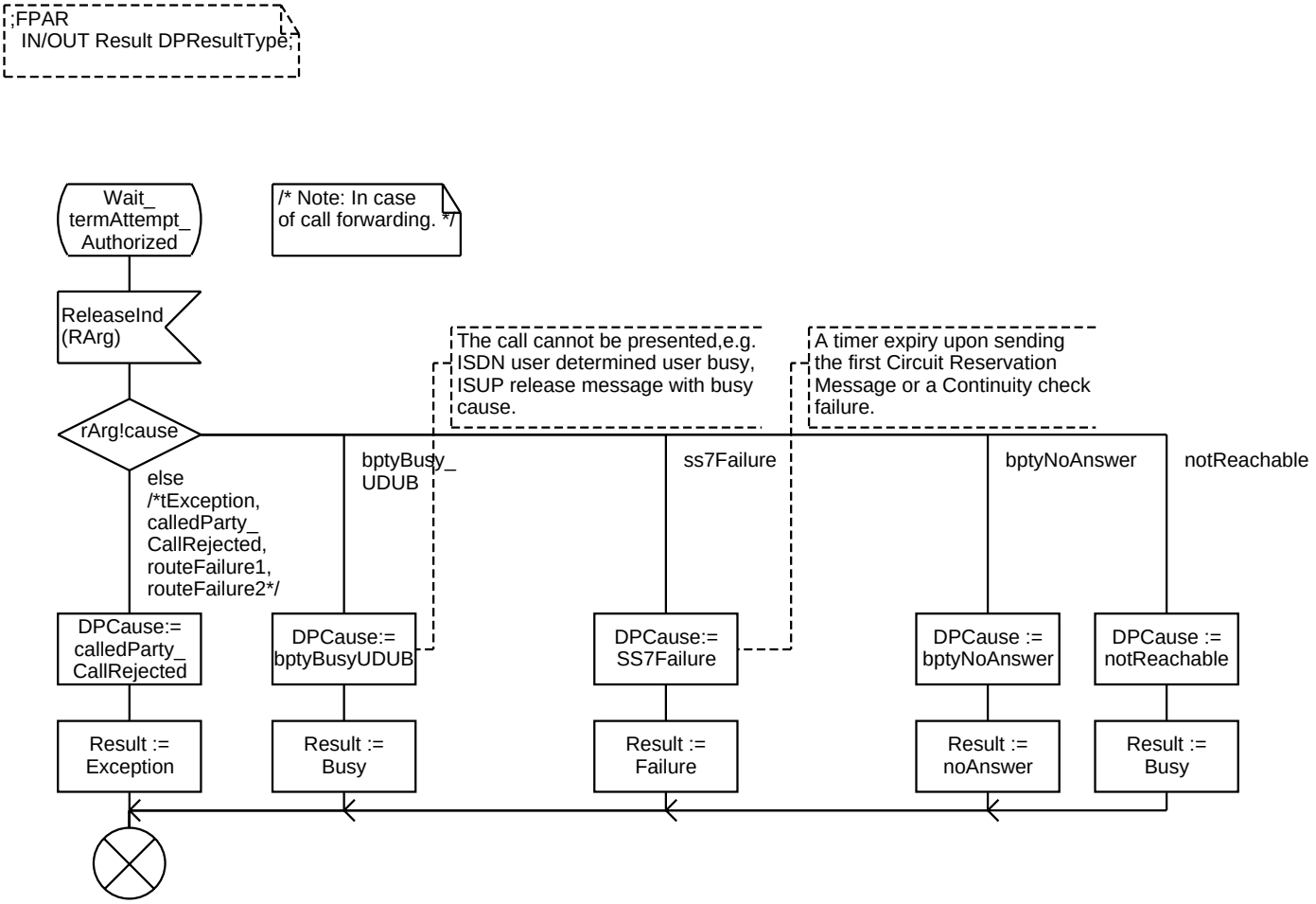
Procedure DP_termAttemptAuthorized 1(4)



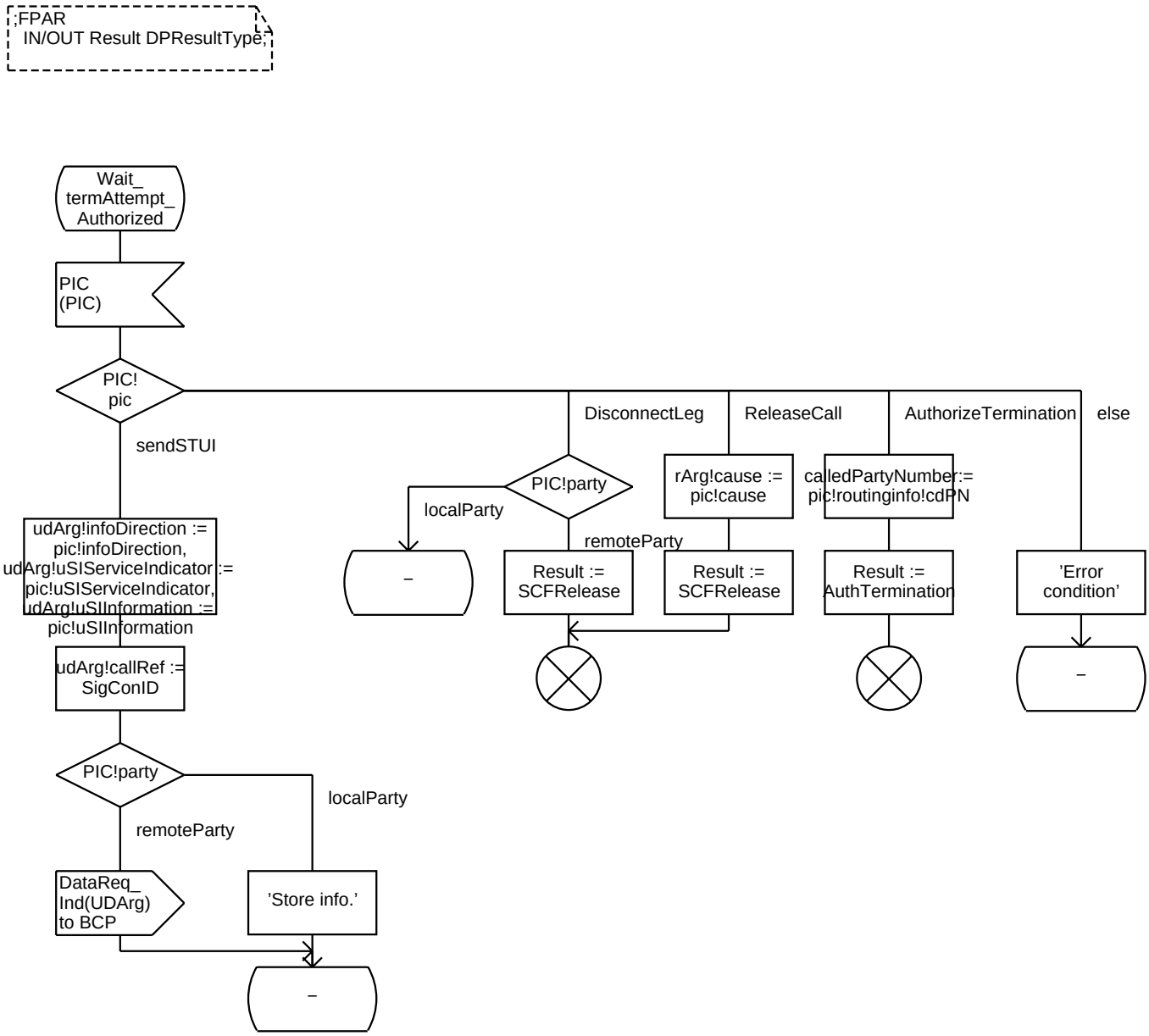
Procedure DP_termAttemptAuthorized 2(4)



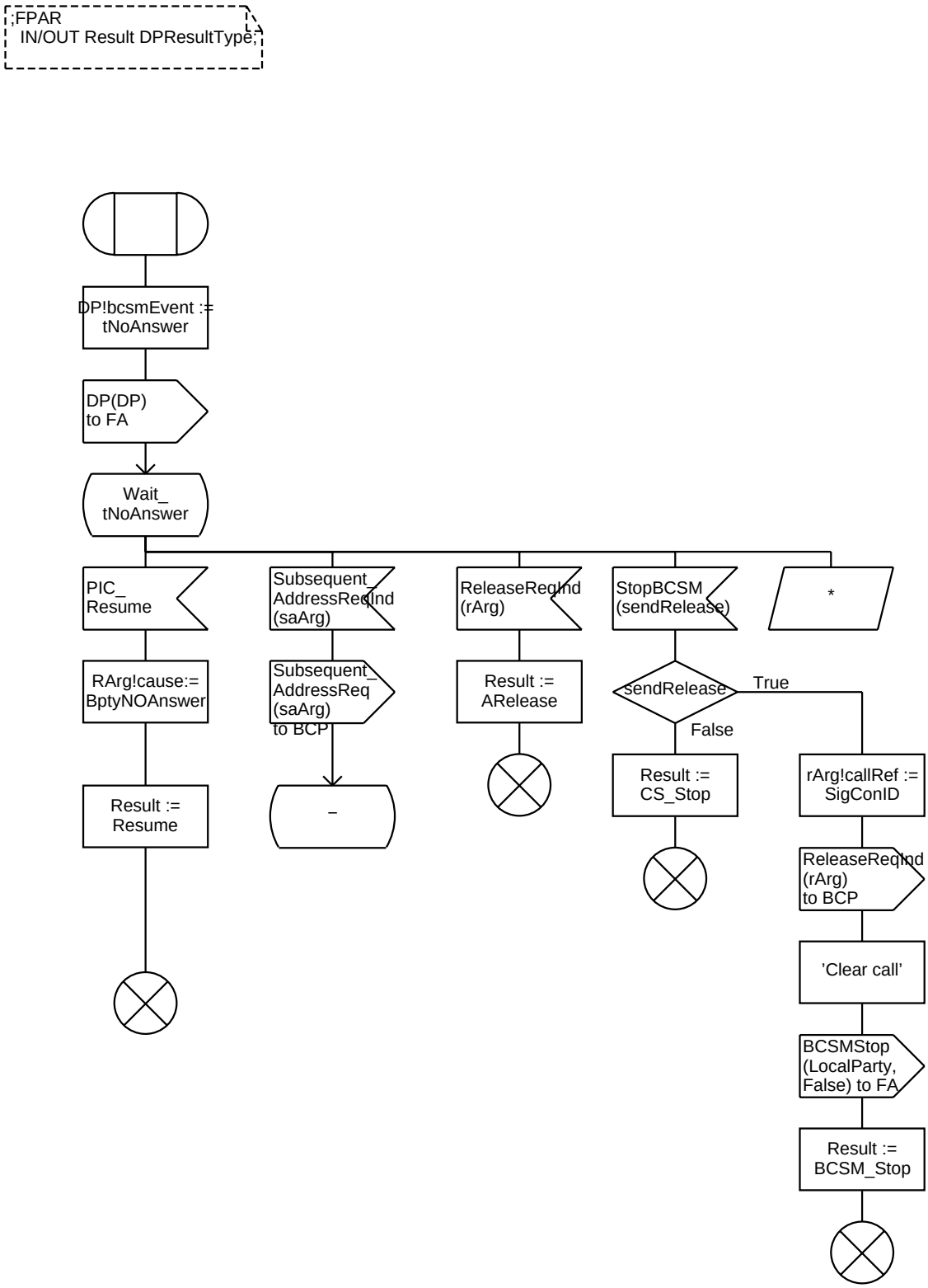
Procedure DP_termAttemptAuthorized 3(4)



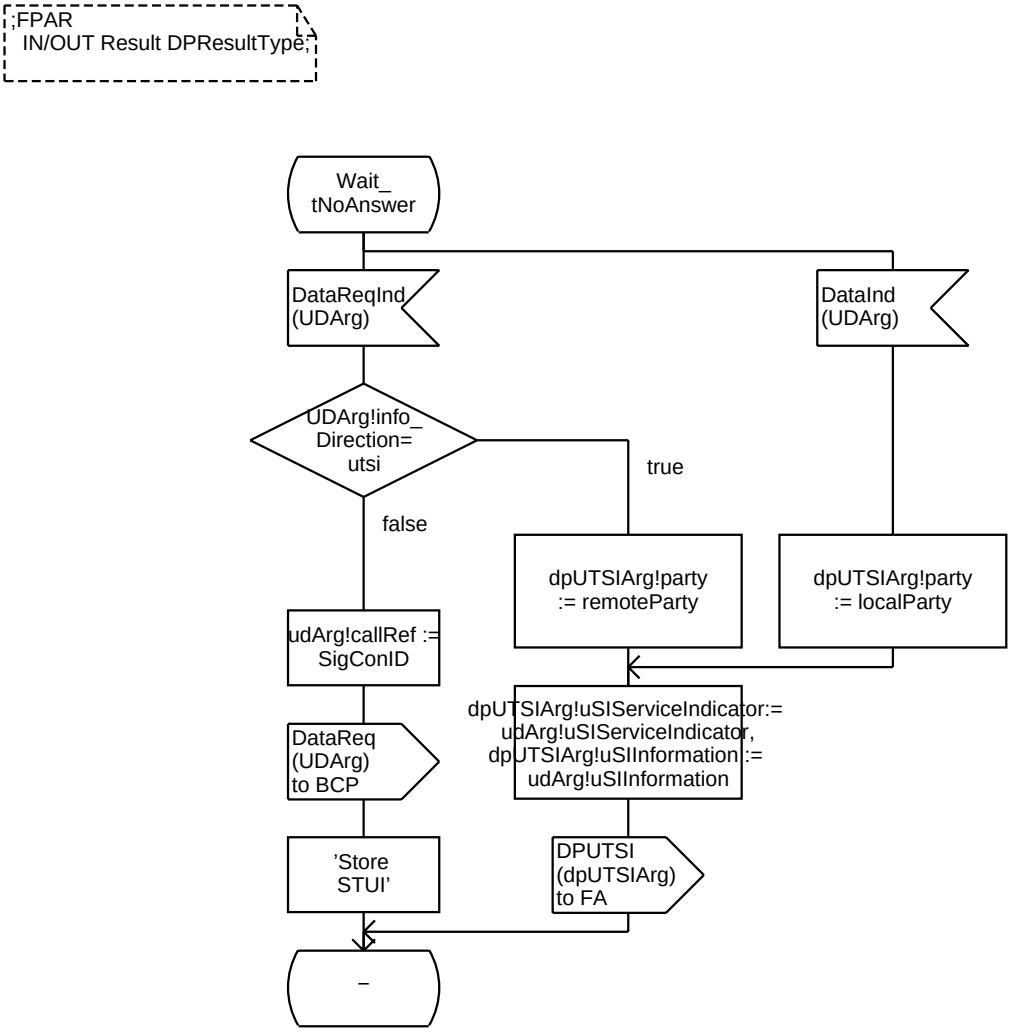
Procedure DP_termAttemptAuthorized 4(4)



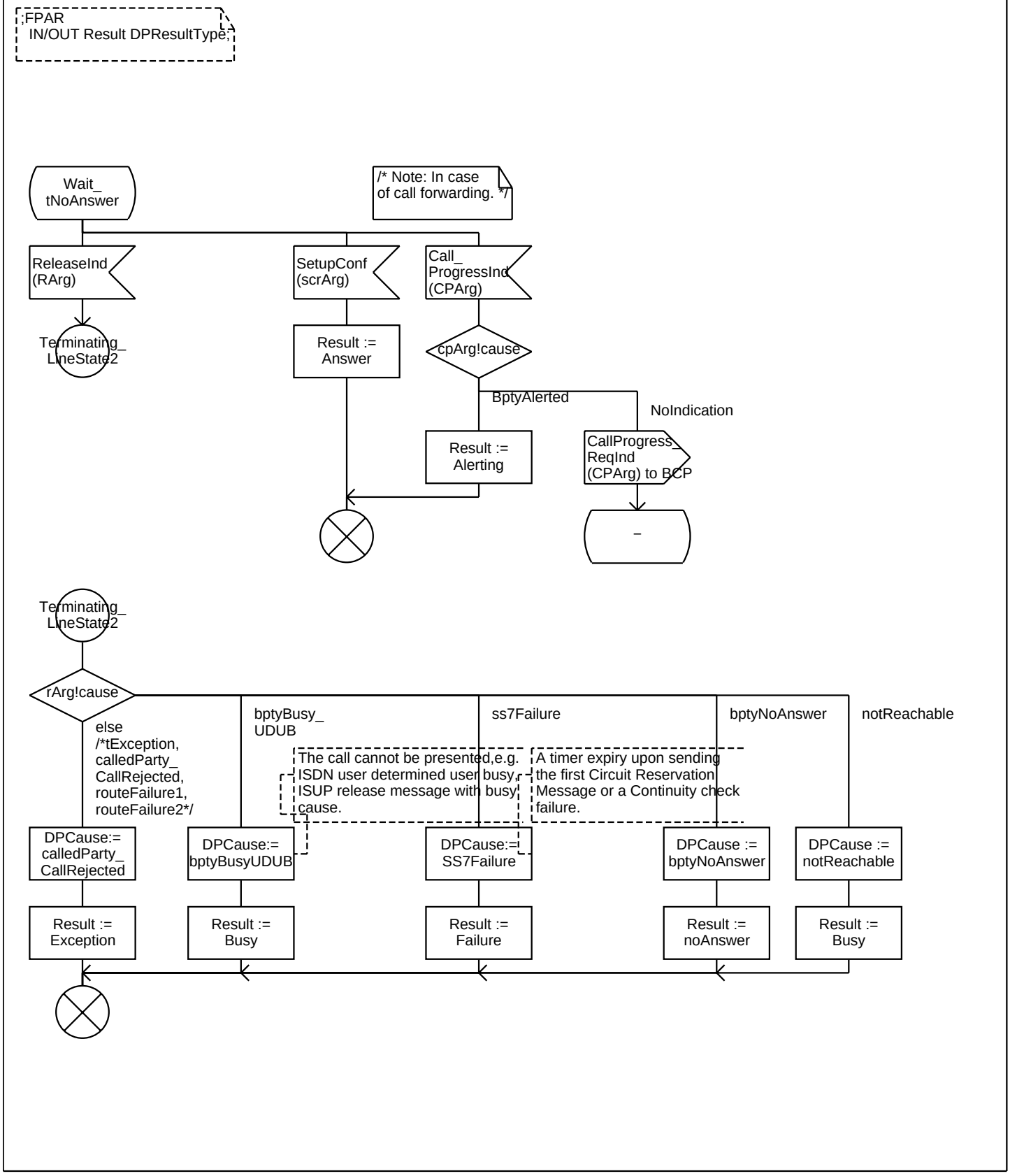
Procedure DP_tNoAnswer 1(4)



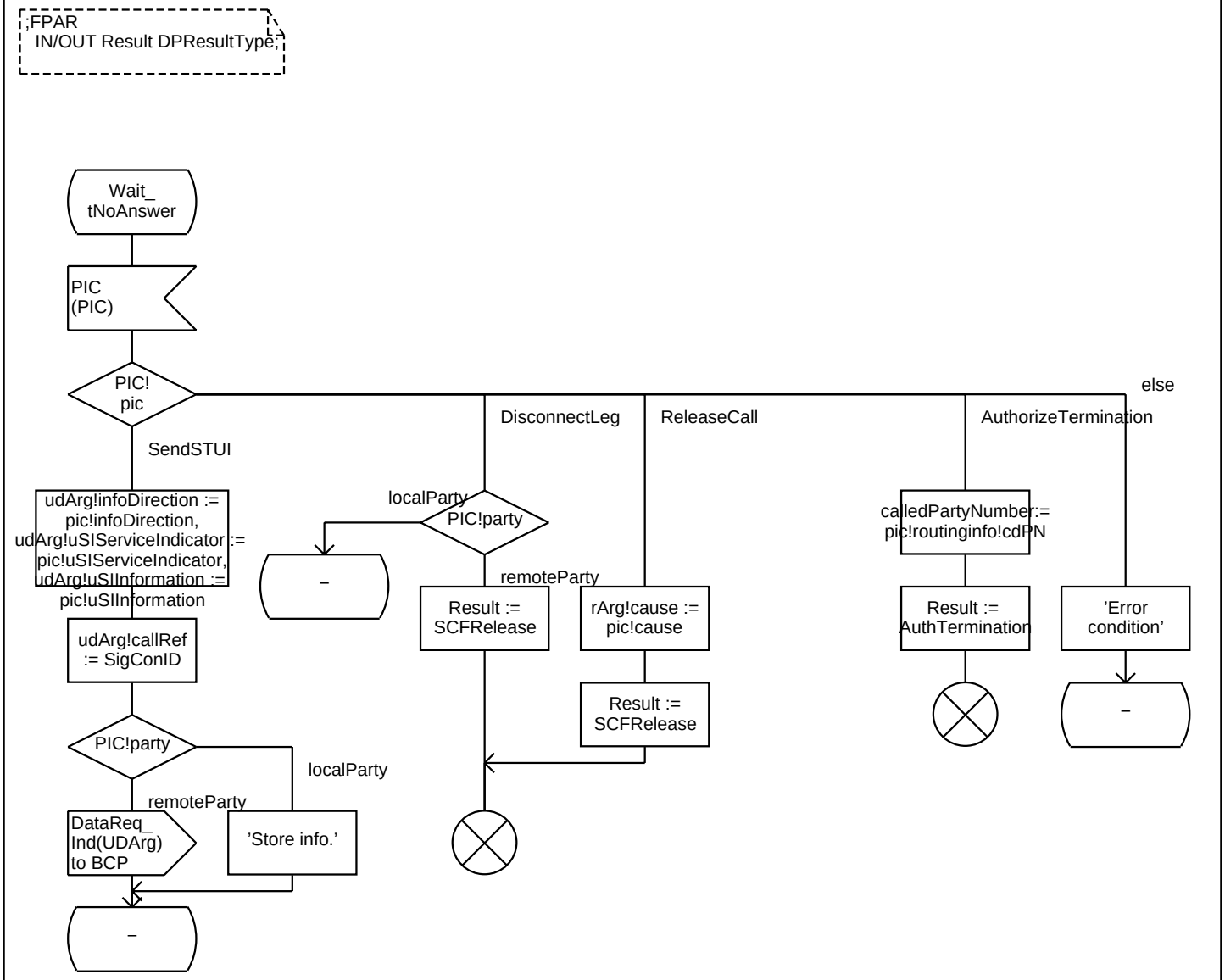
Procedure DP_tNoAnswer 2(4)



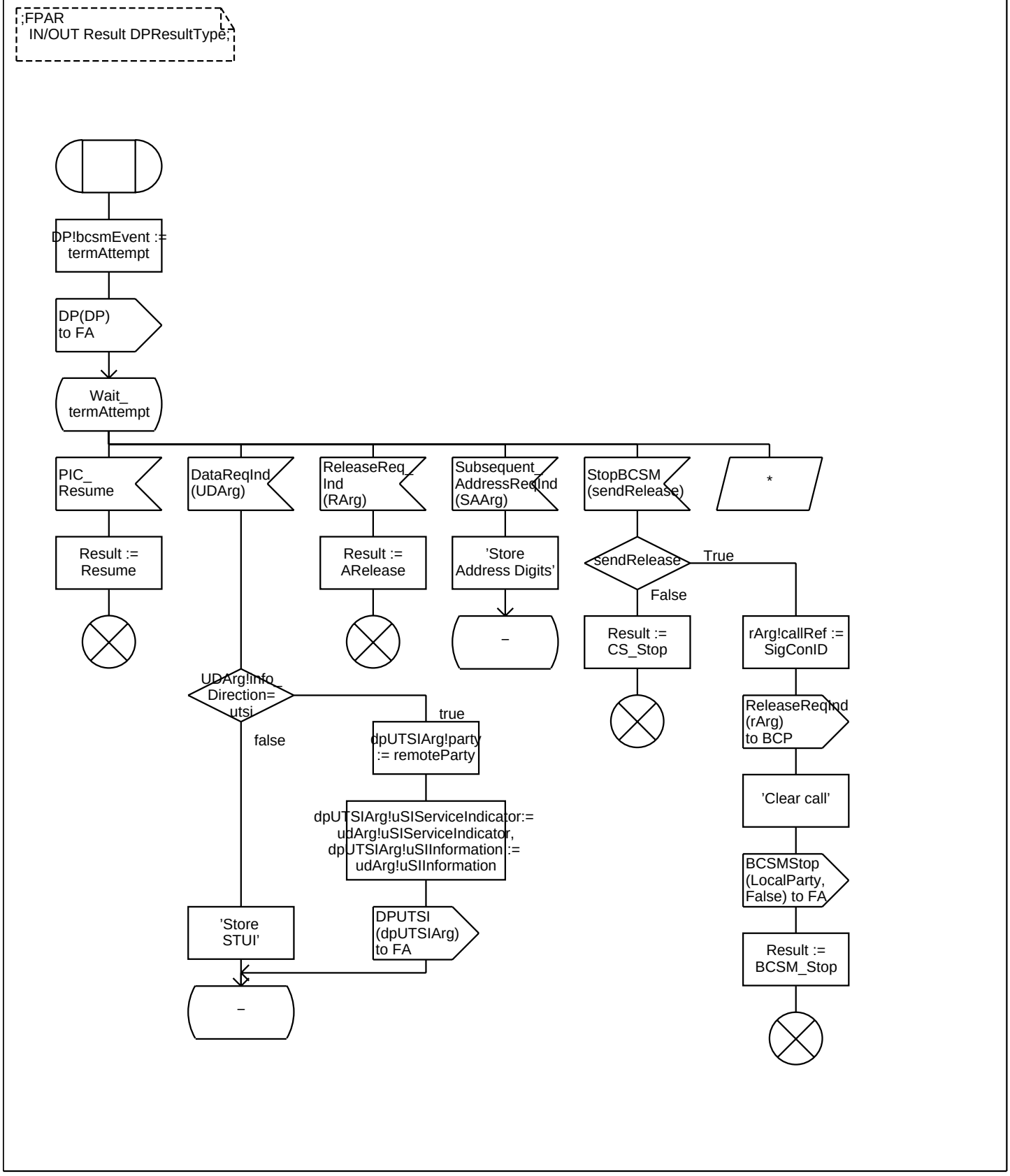
Procedure DP_tNoAnswer 3(4)



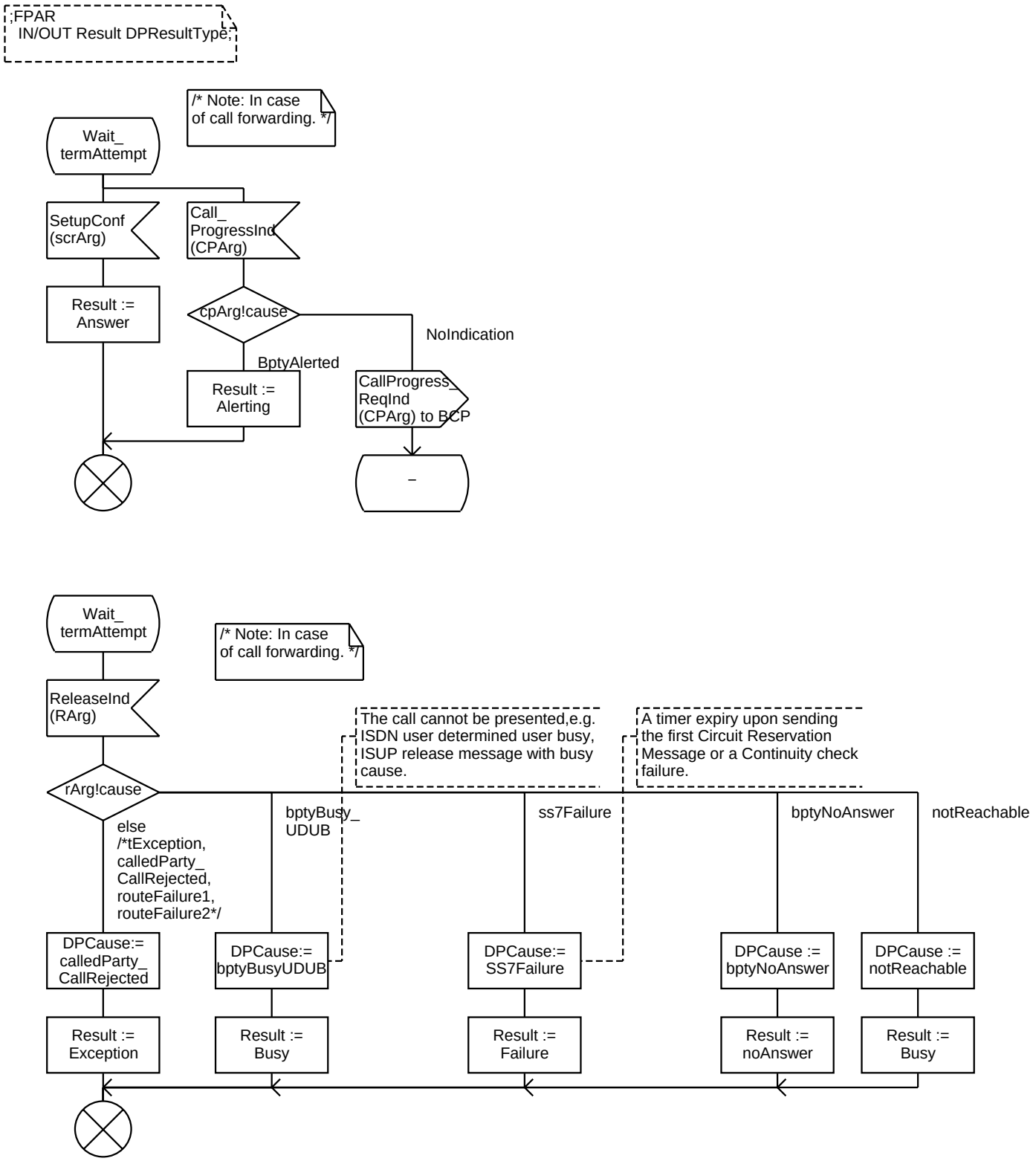
Procedure DP_tNoAnswer 4(4)



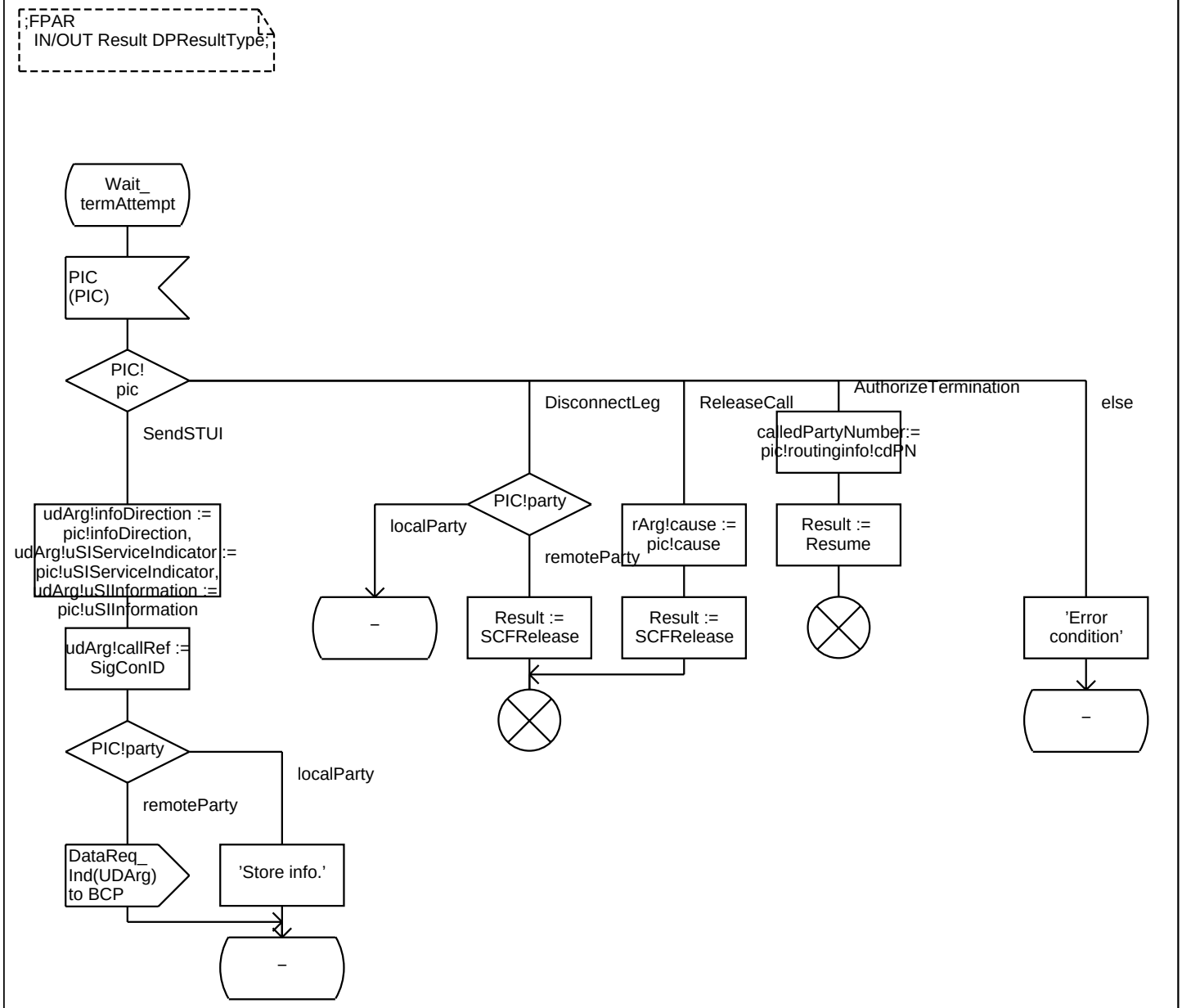
Procedure DP_termAttempt 1(3)



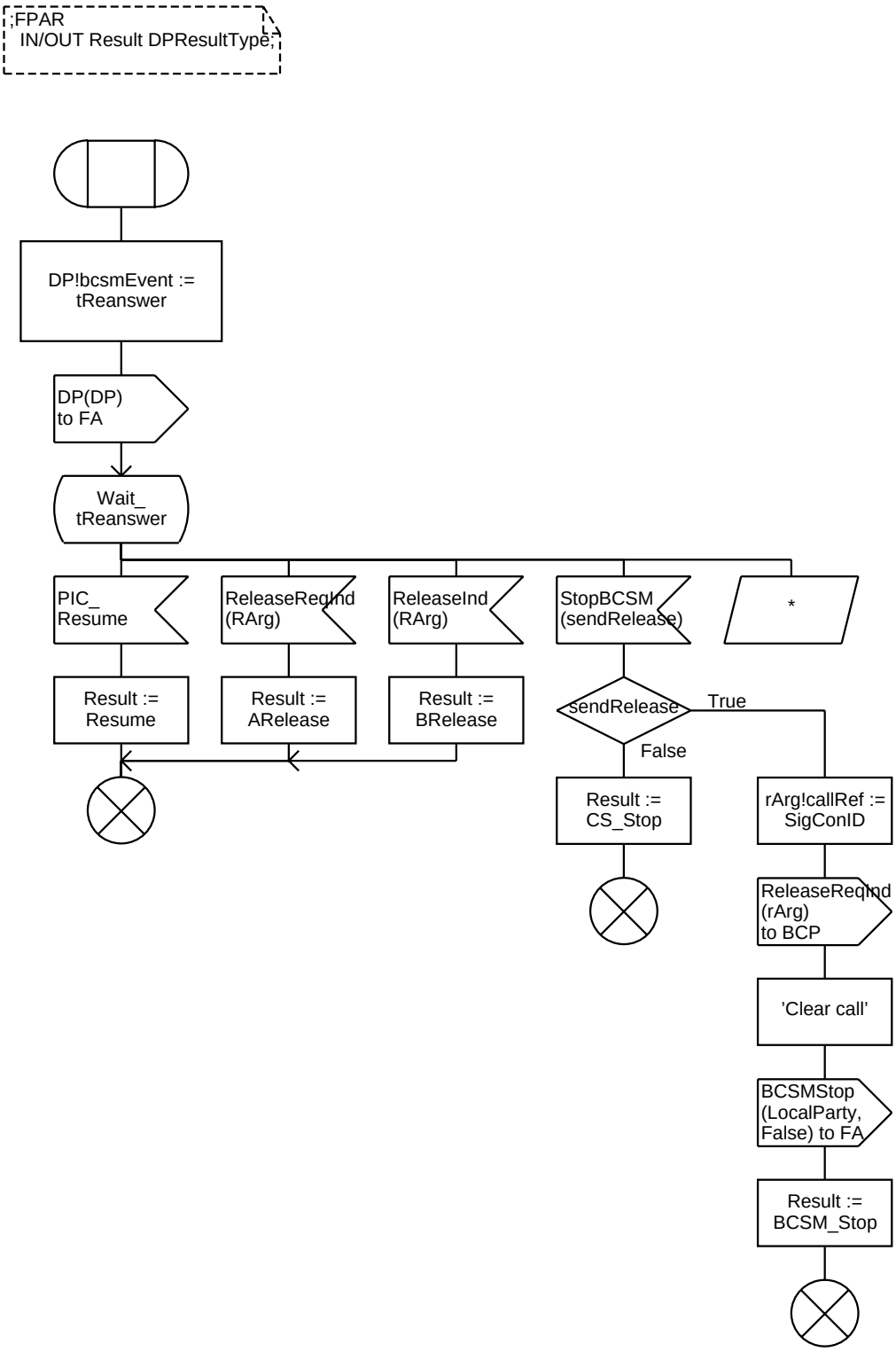
Procedure DP_termAttempt 2(3)



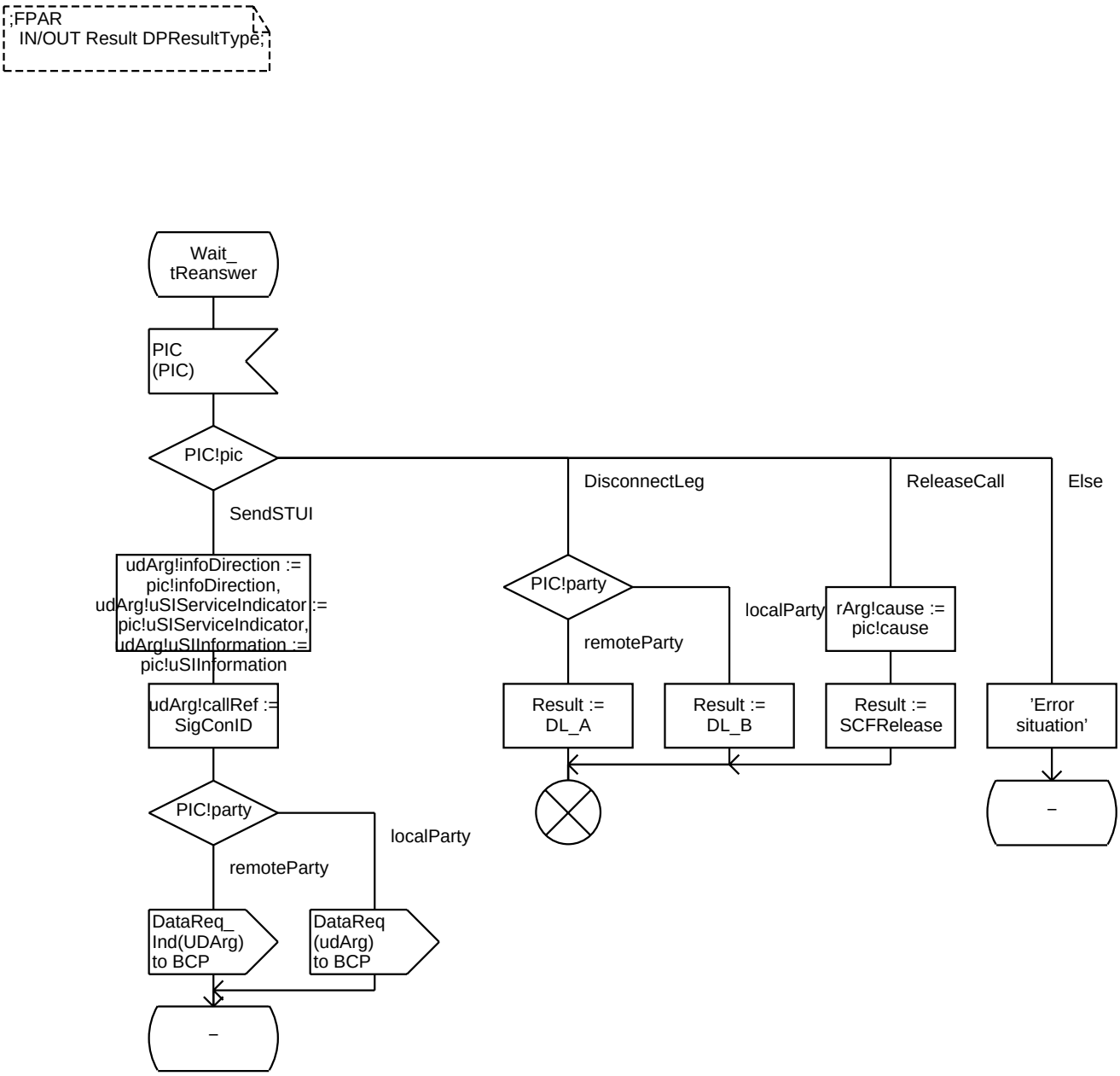
Procedure DP_termAttempt 3(3)



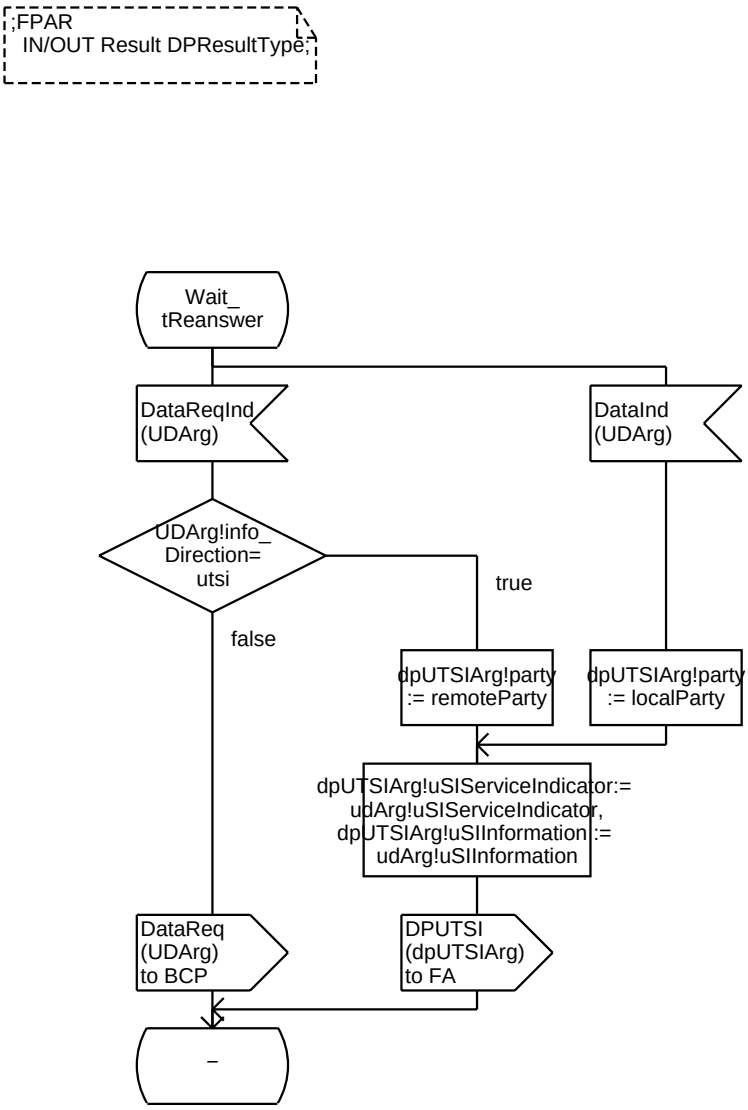
Procedure DP_tReanswer 1(3)



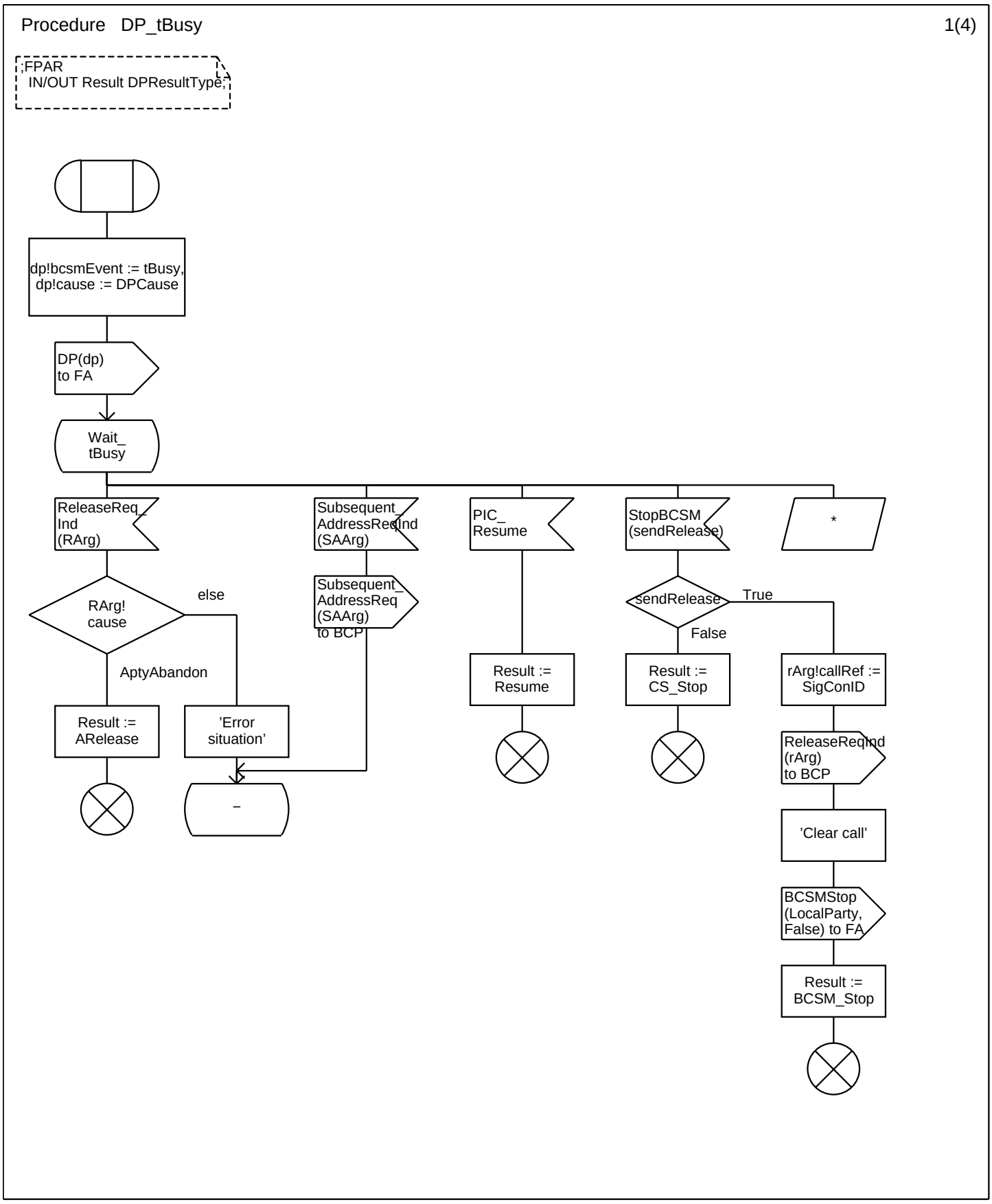
Procedure DP_tReanswer 2(3)



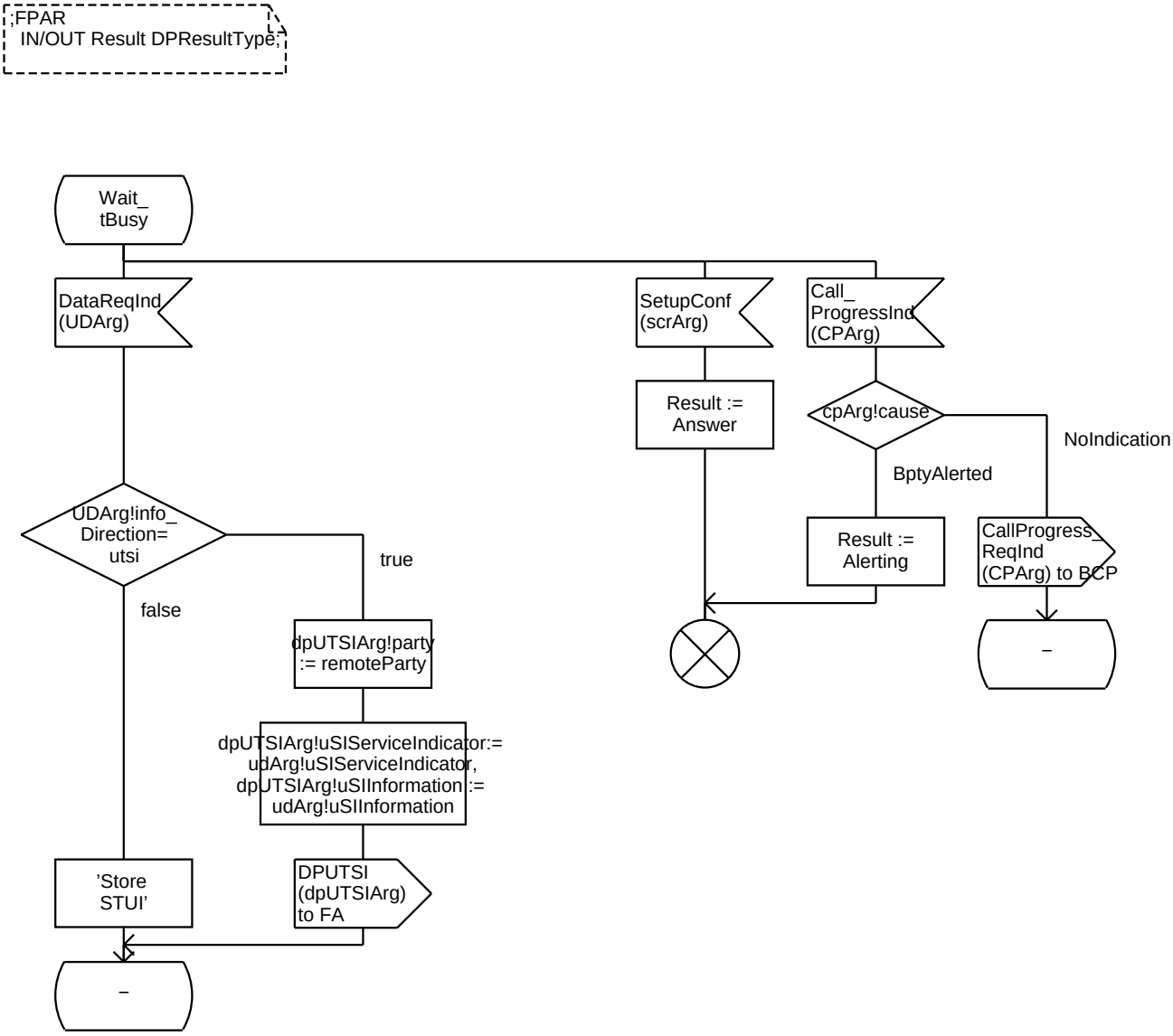
Procedure DP_tReanswer 3(3)



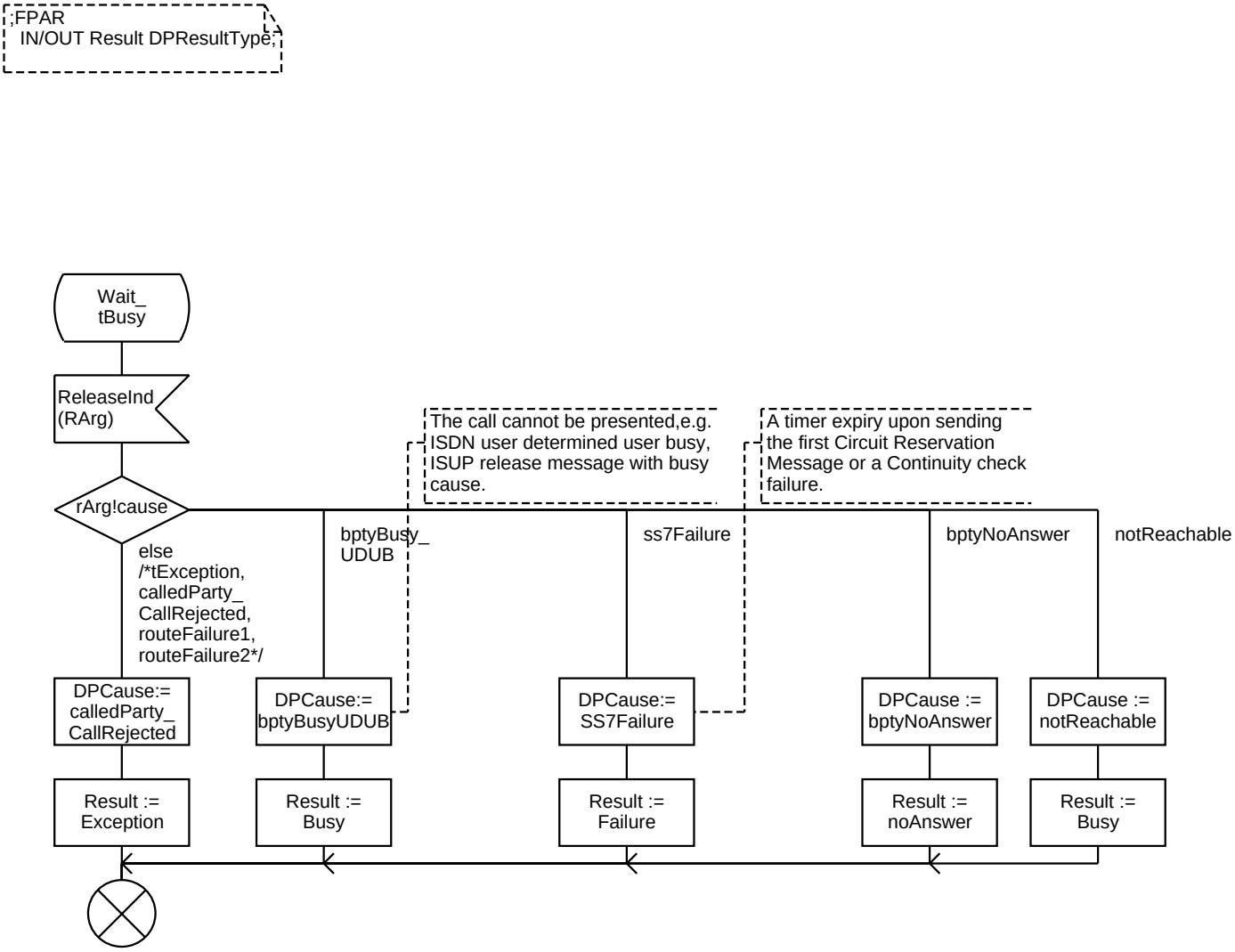
Procedure DP_tBusy 1(4)



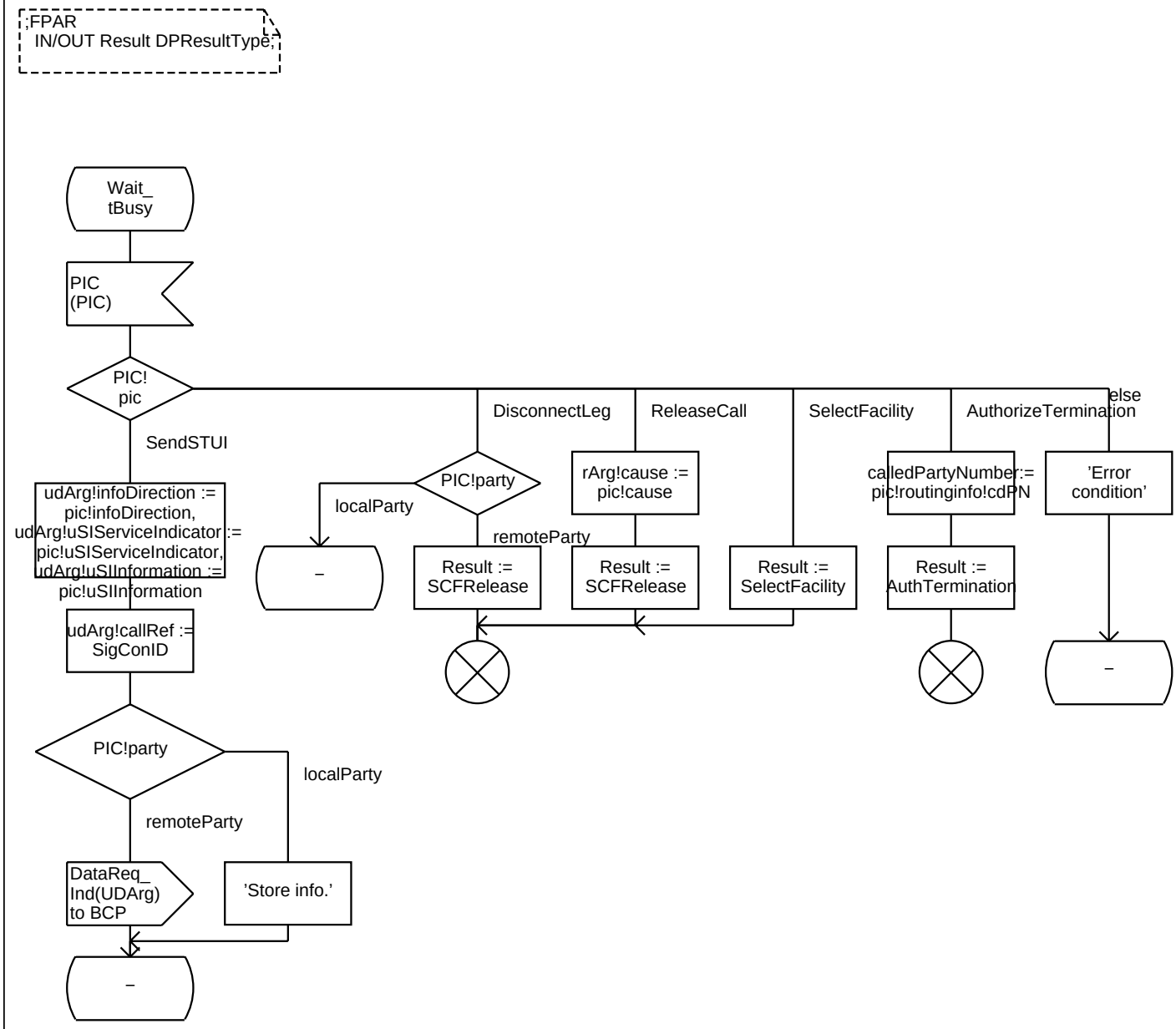
Procedure DP_tBusy 2(4)

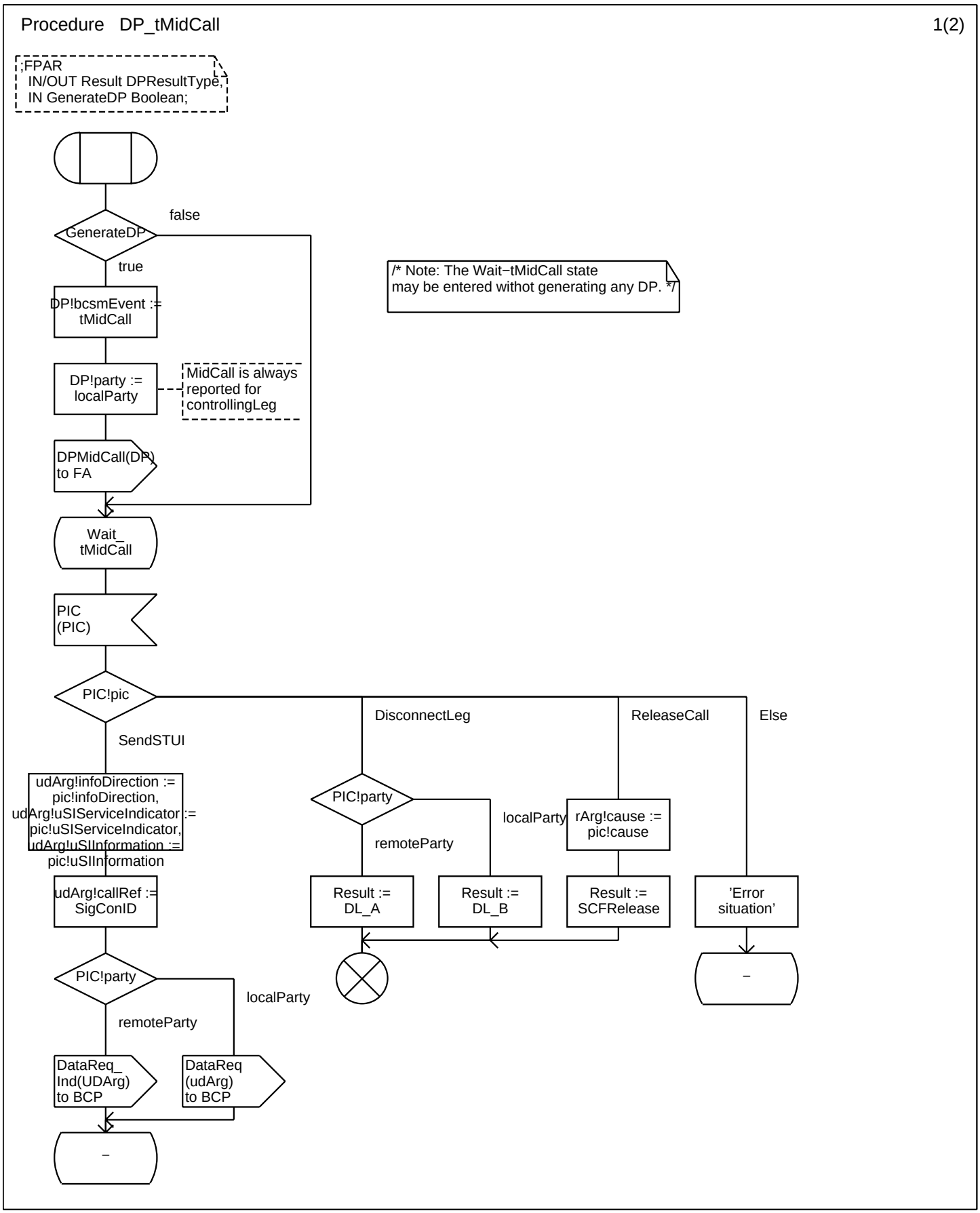


Procedure DP_tBusy 3(4)

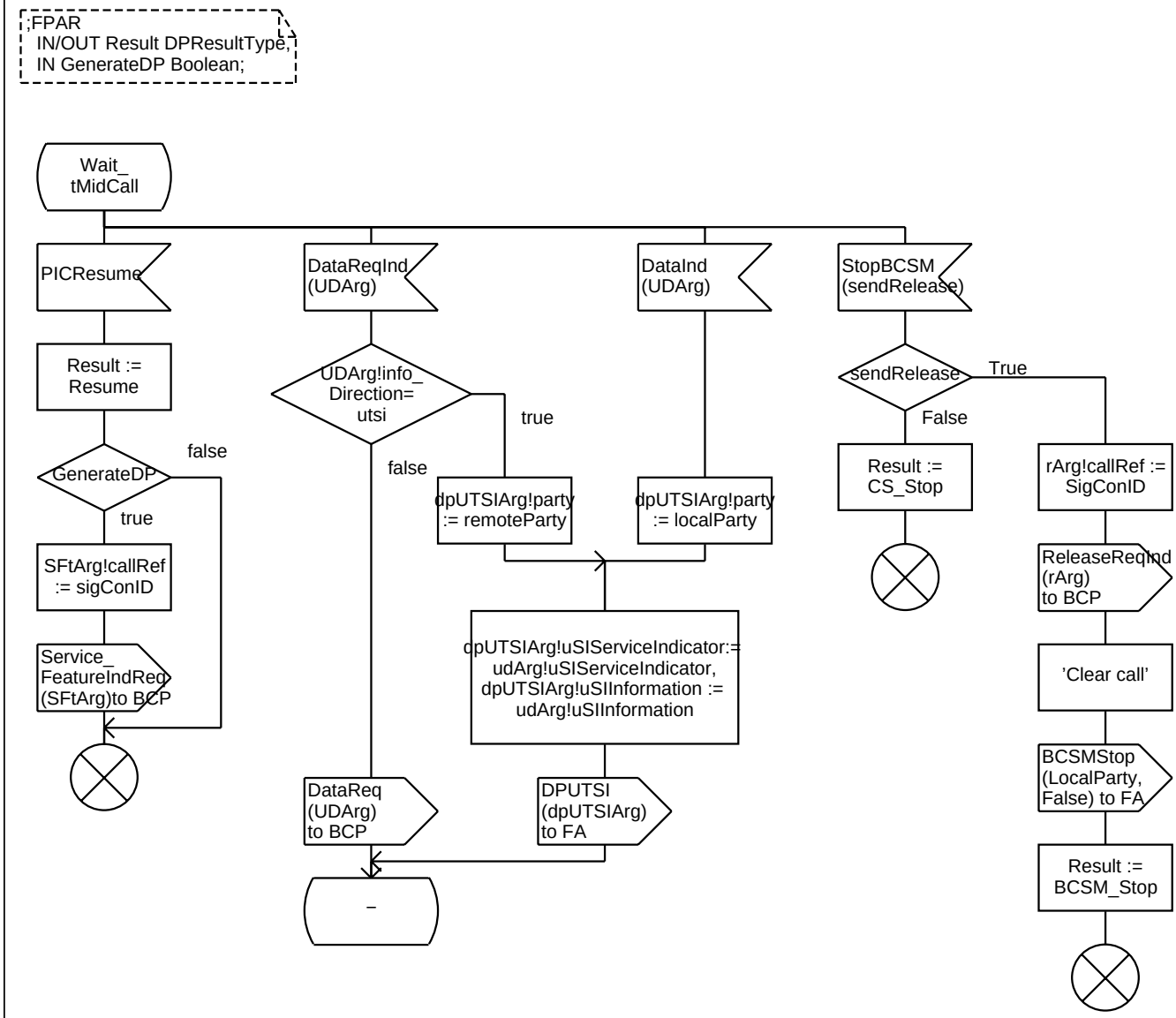


Procedure DP_tBusy 4(4)

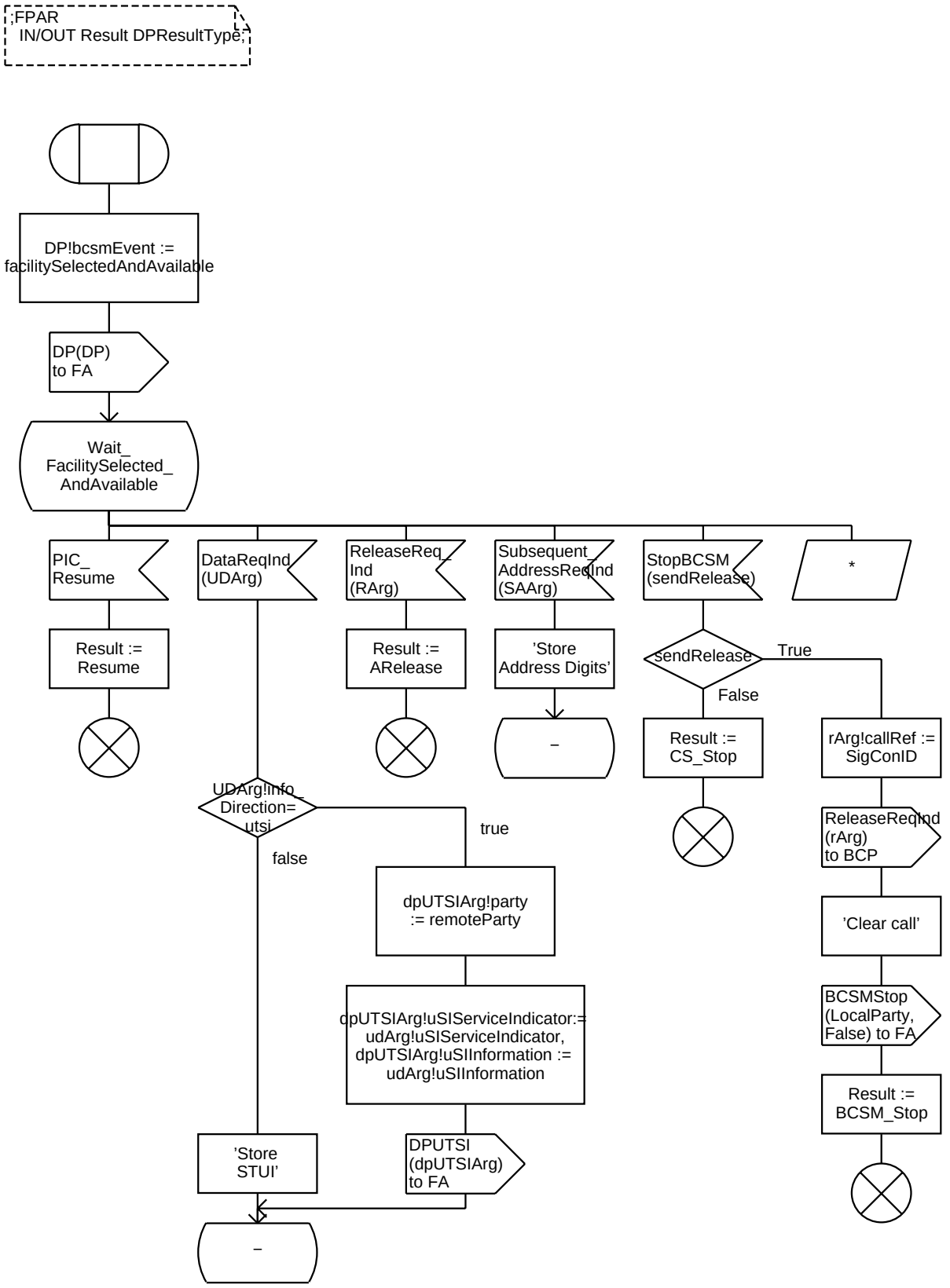



Procedure DP_tMidCall
1(2)
;FPAR
IN/OUT Result DPResultType,
IN GenerateDP Boolean;
GenerateDP
false
true
DP!bcsmEvent := tMidCall
DP!party := localParty
MidCall is always reported for controllingLeg
DPMidCall(DP) to FA
/* Note: The Wait−tMidCall state may be entered withot generating any DP. */
Wait_ tMidCall
PIC (PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection := pic!infoDirection,
udArg!uSIServiceIndicator := pic!uSIServiceIndicator,
udArg!uSIInformation := pic!uSIInformation
udArg!callRef := SigConID
PIC!party
localParty
remoteParty
DataReq_ Ind(UDArg) to BCP
DataReq (udArg) to BCP
−
PIC!party
localParty
remoteParty
Result := DL_A
Result := DL_B
rArg!cause := pic!cause
Result := SCFRelease
'Error situation'
−

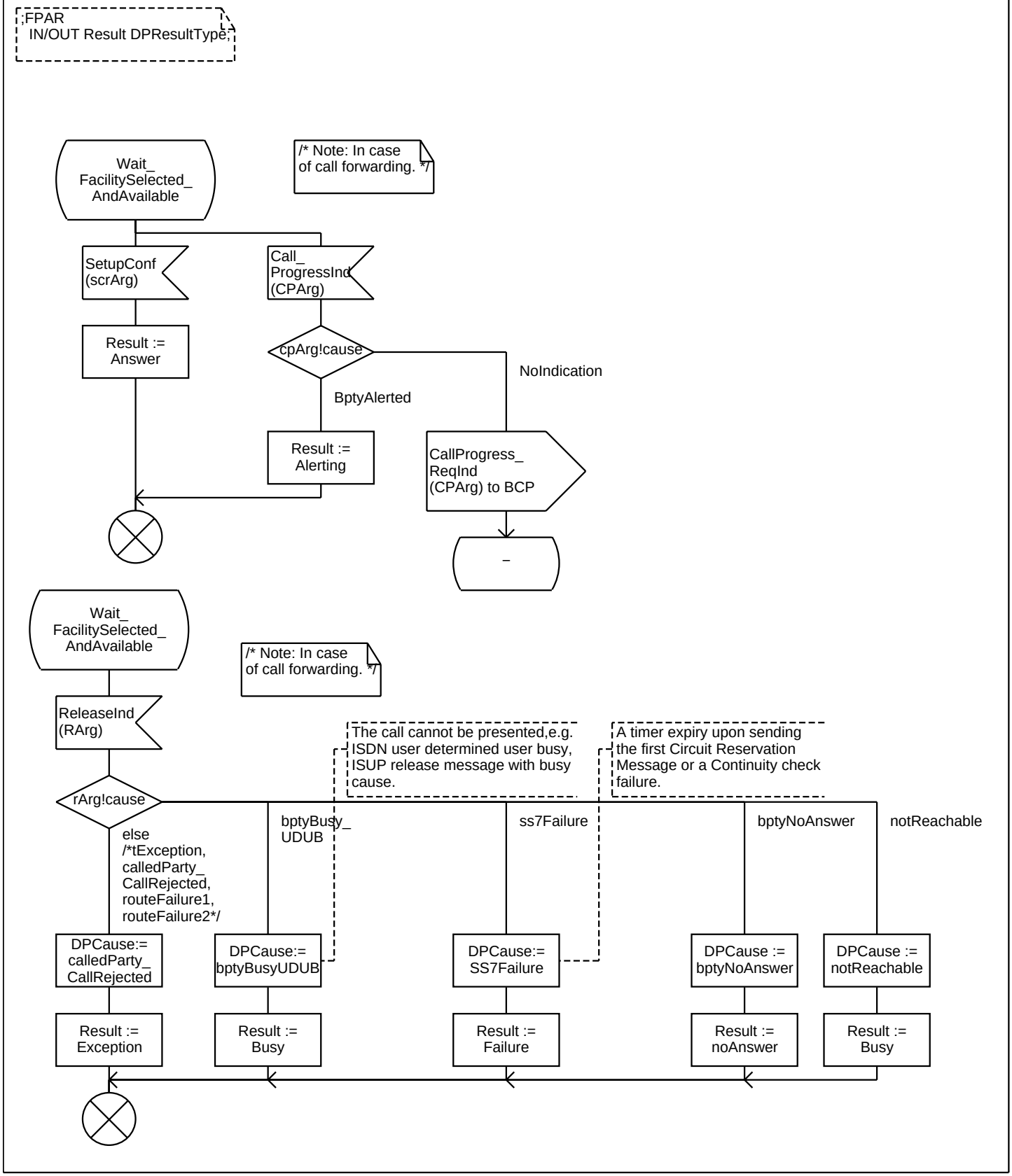
Procedure DP_tMidCall 2(2)



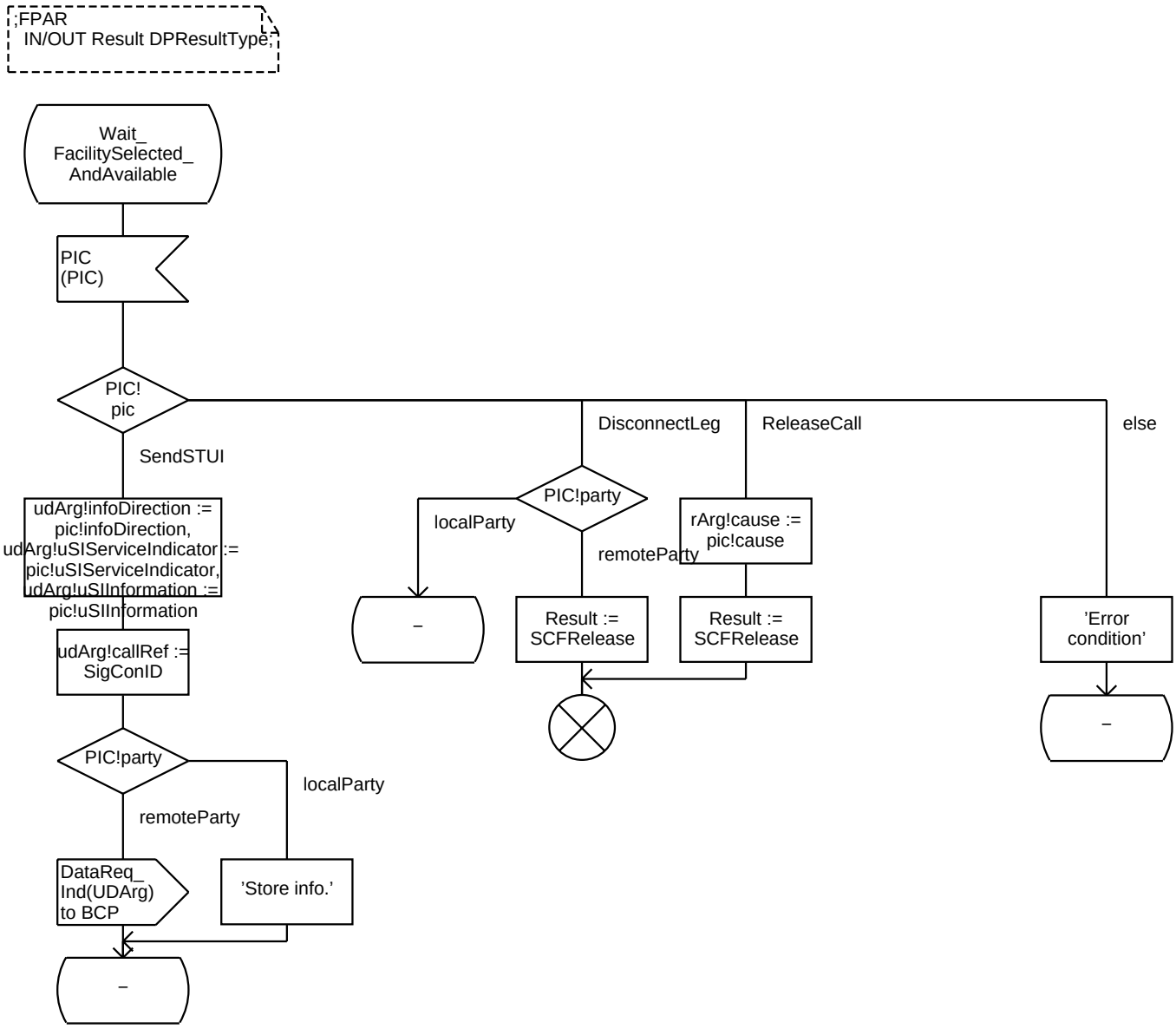
Procedure DP_Facility_Selected_and_Available 1(3)



Procedure DP_Facility_Selected_and_Available 2(3)

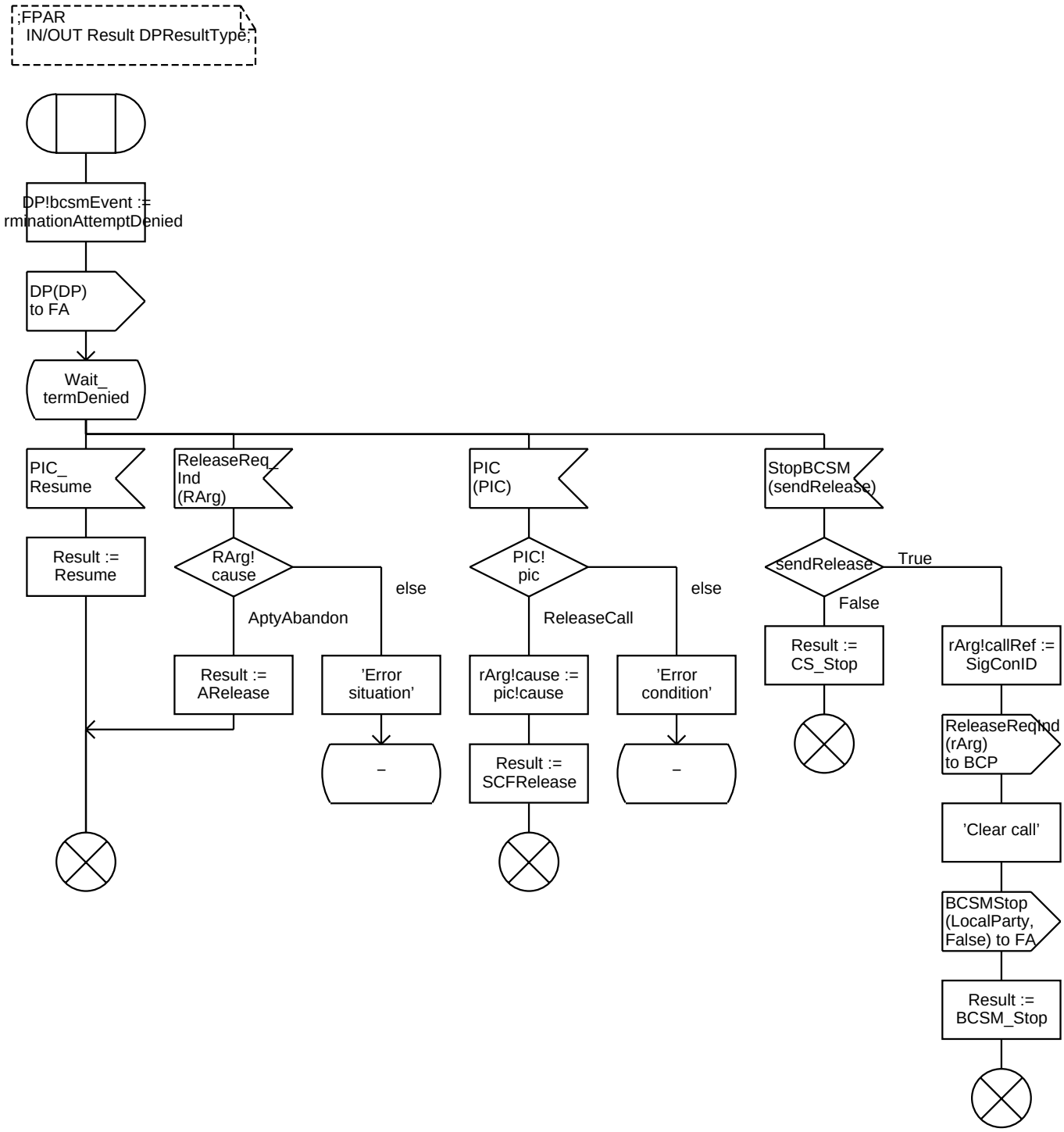


Procedure DP_Facility_Selected_and_Available 3(3)

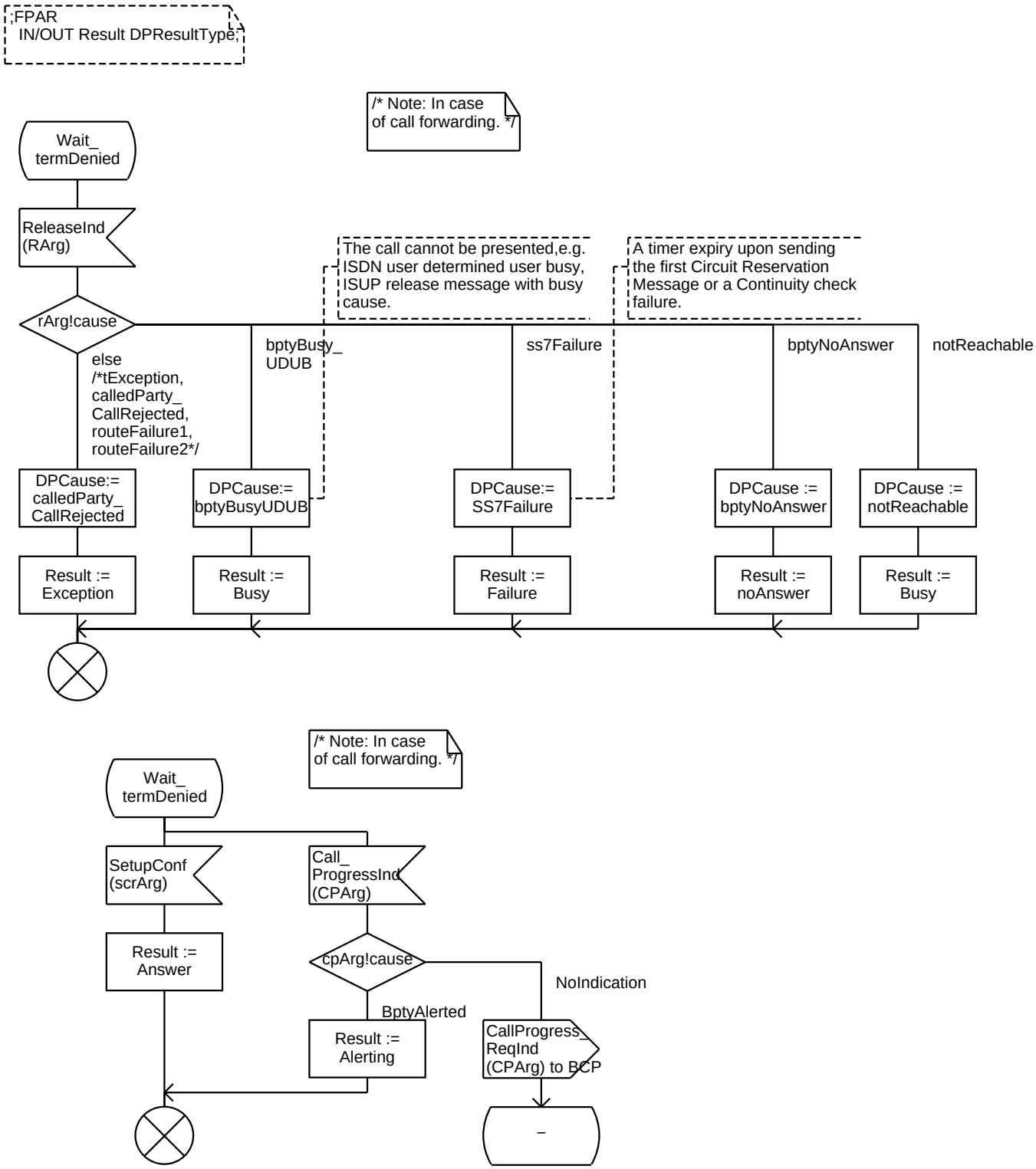


DP_termDenied

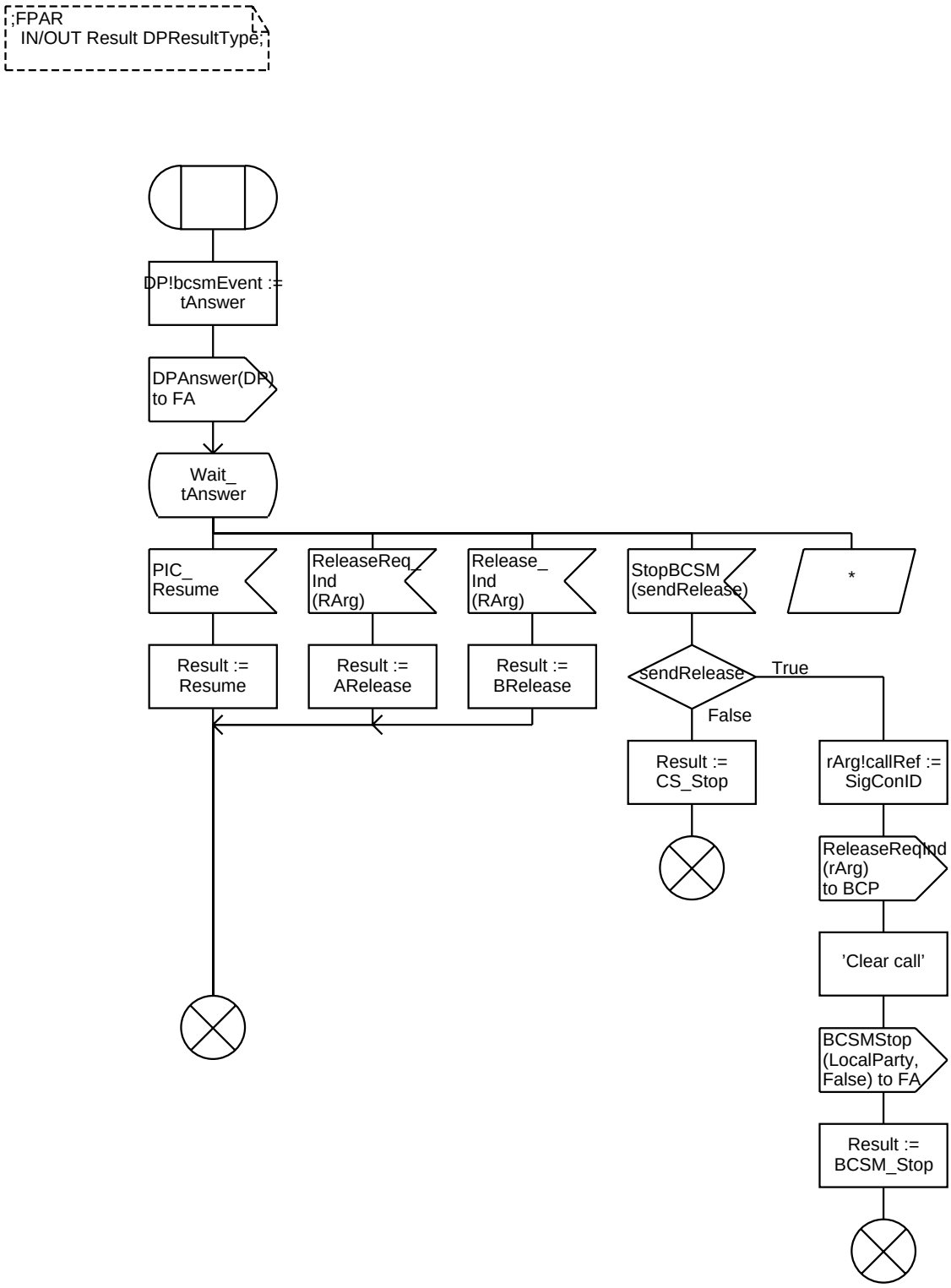
Procedure DP_termDenied 1(2)



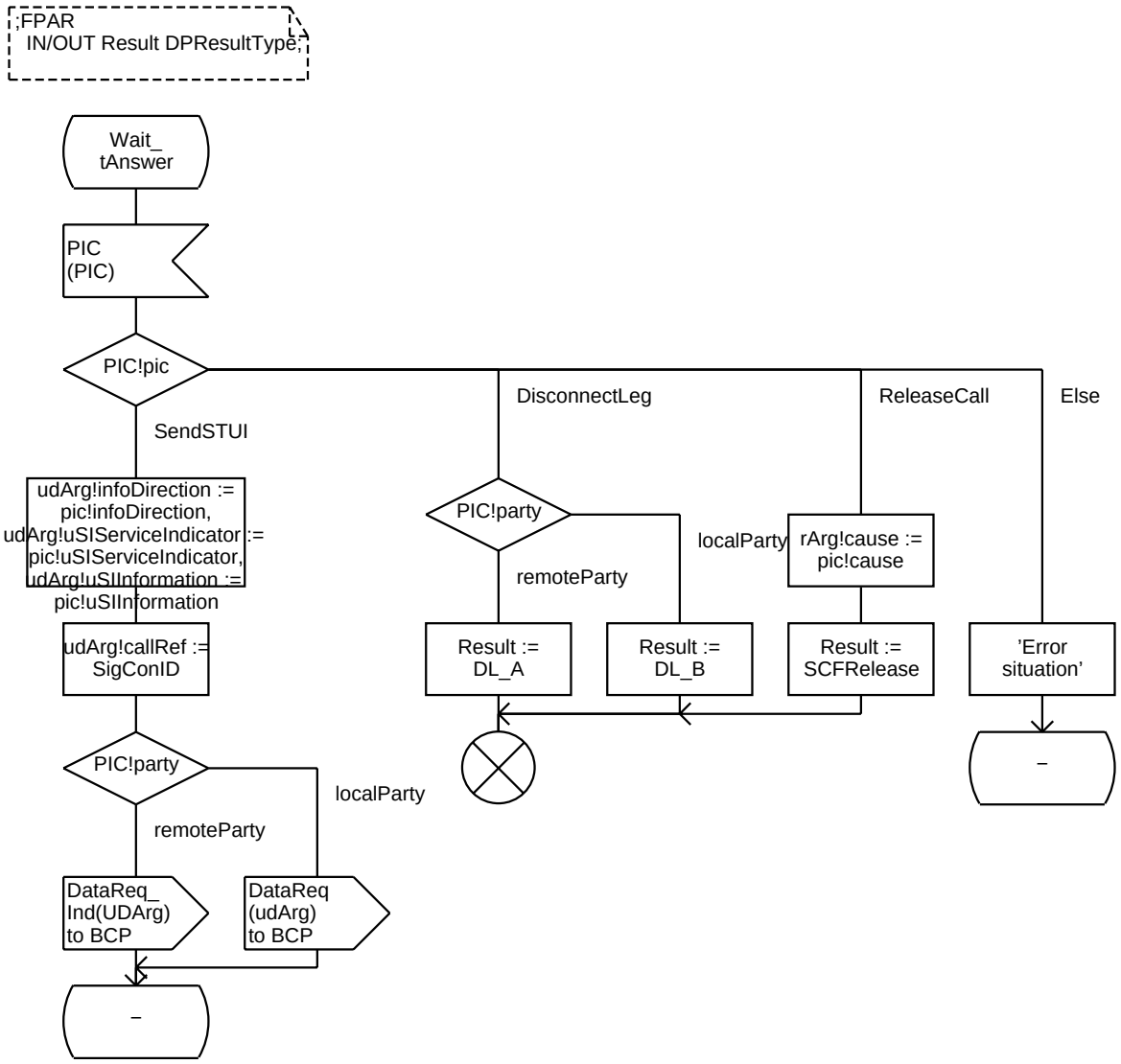
Procedure DP_termDenied 2(2)



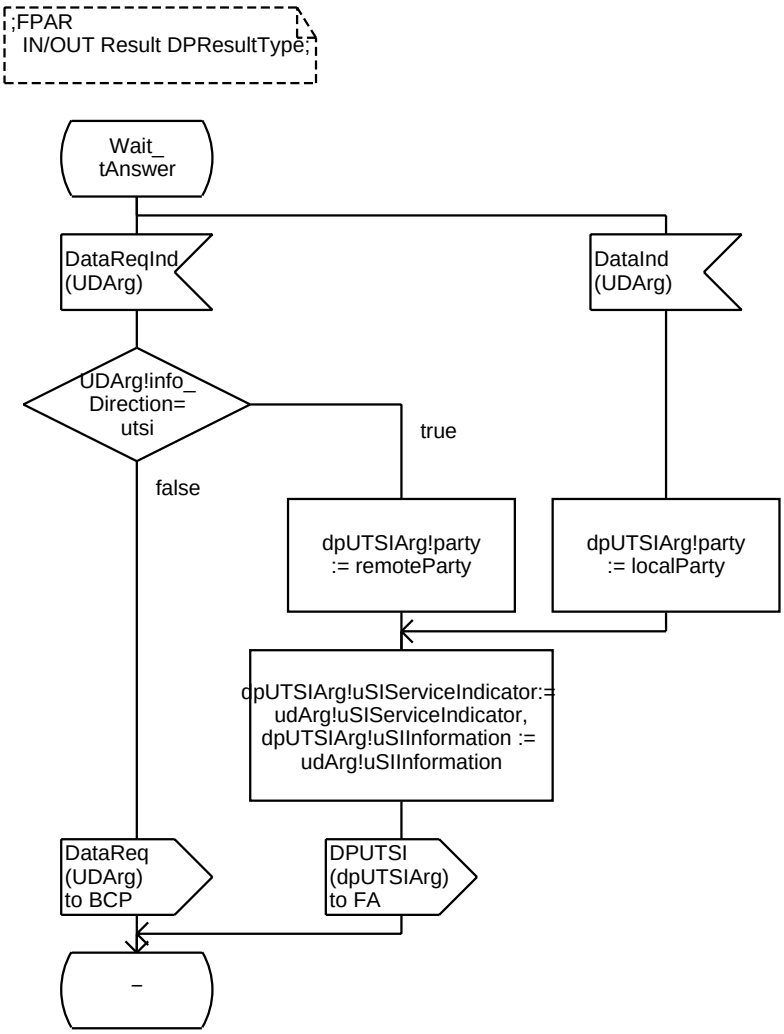
Procedure DP_tAnswer 1(3)



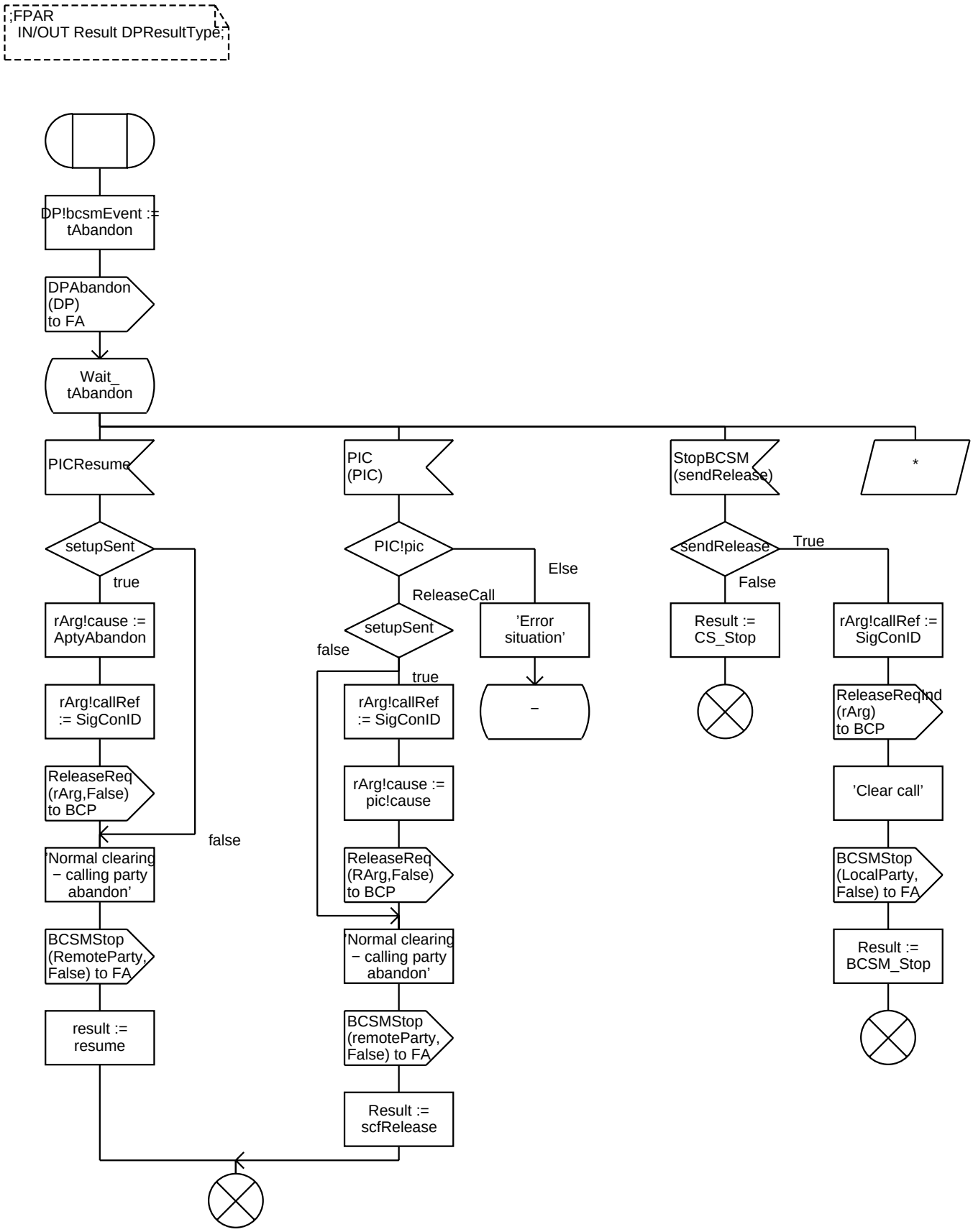
Procedure DP_tAnswer 2(3)



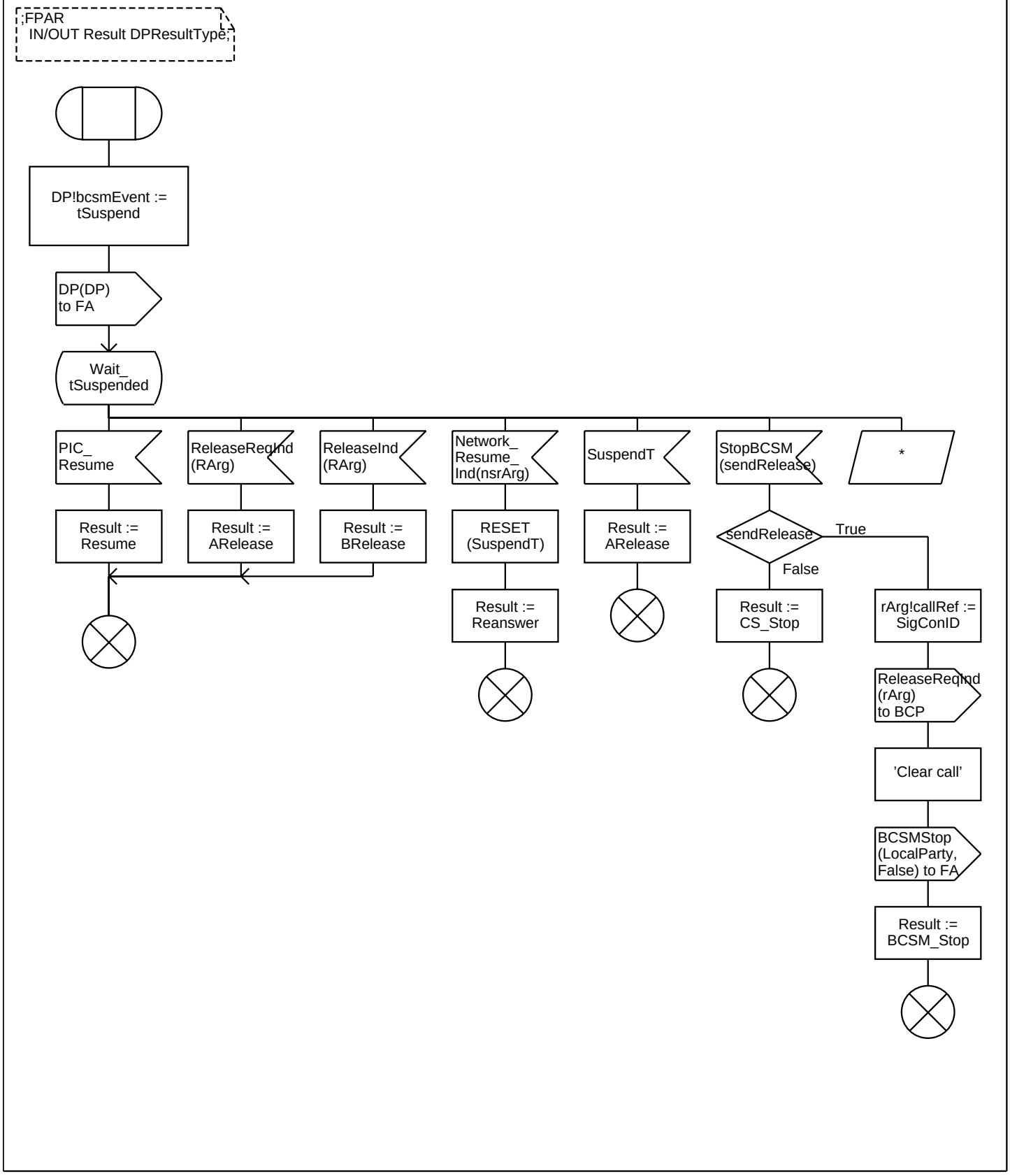
Procedure DP_tAnswer 3(3)



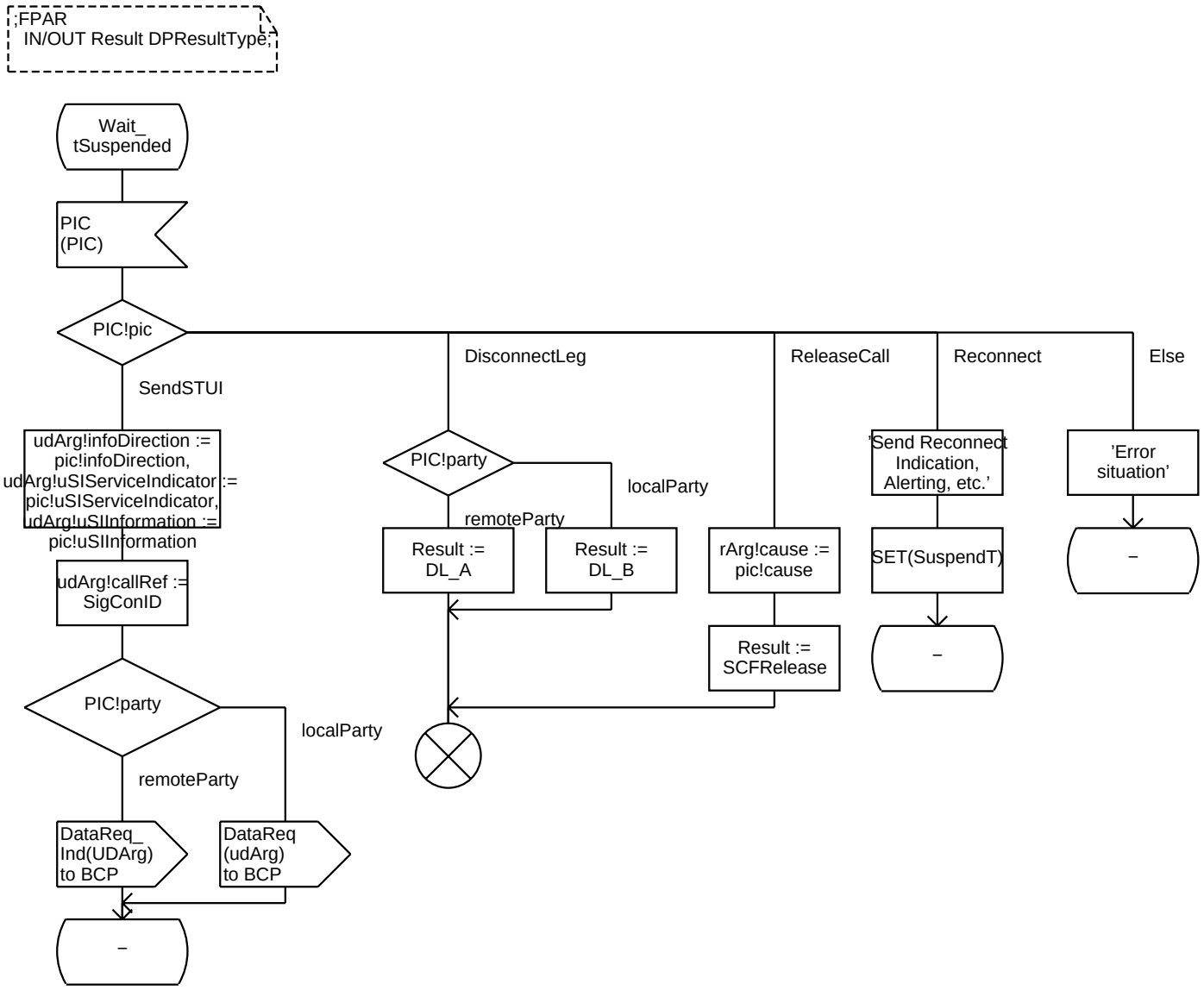
Procedure DP_tAbandon 1(1)



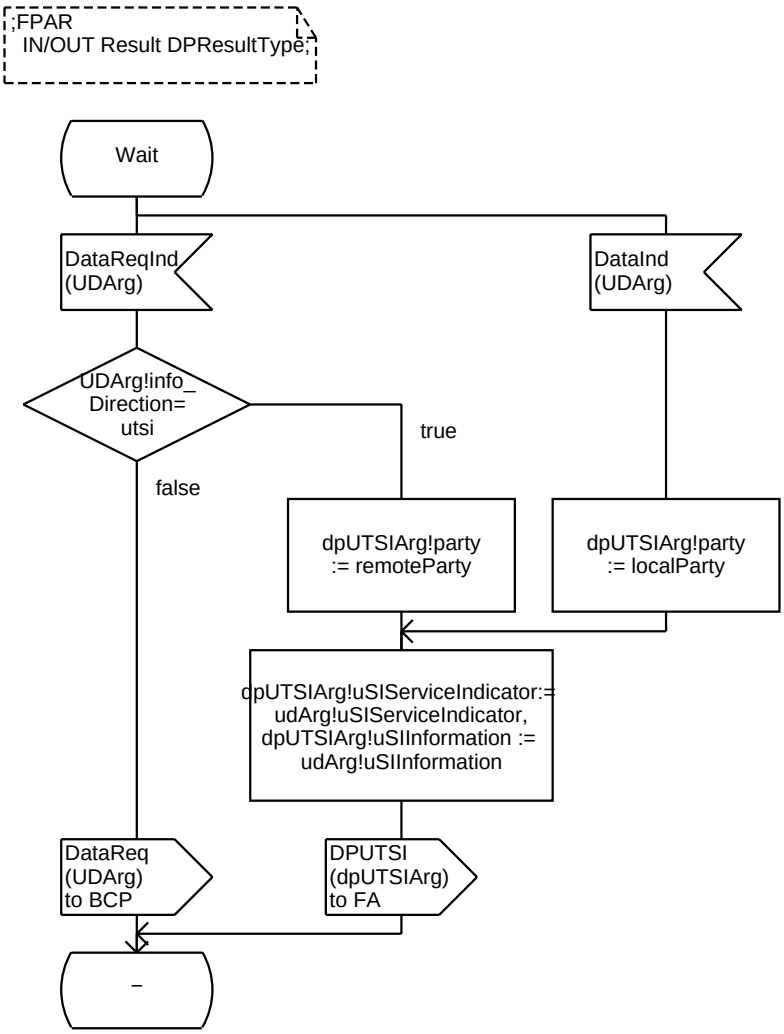
Procedure DP_tSuspend 1(3)



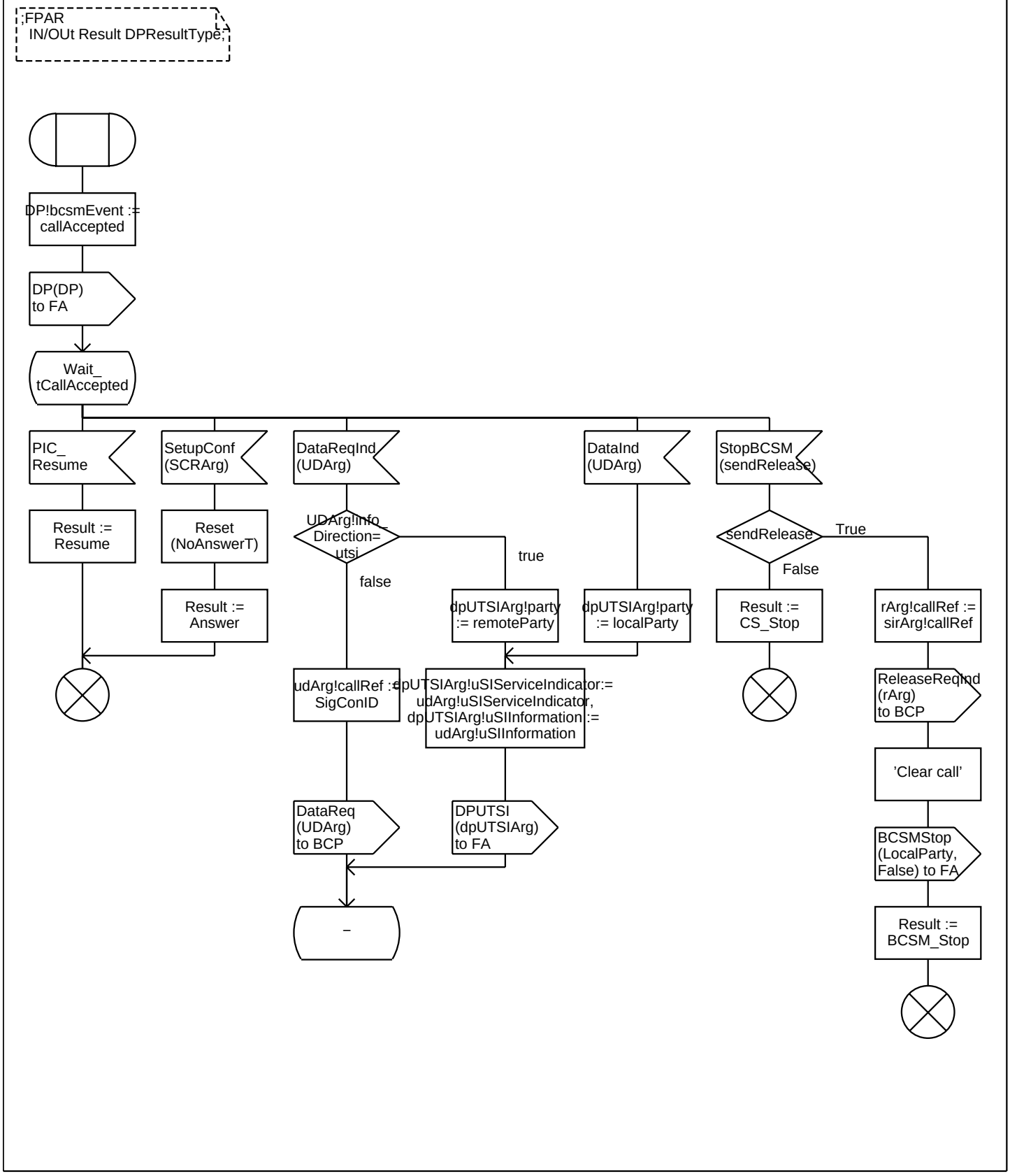
Procedure DP_tSuspend 2(3)



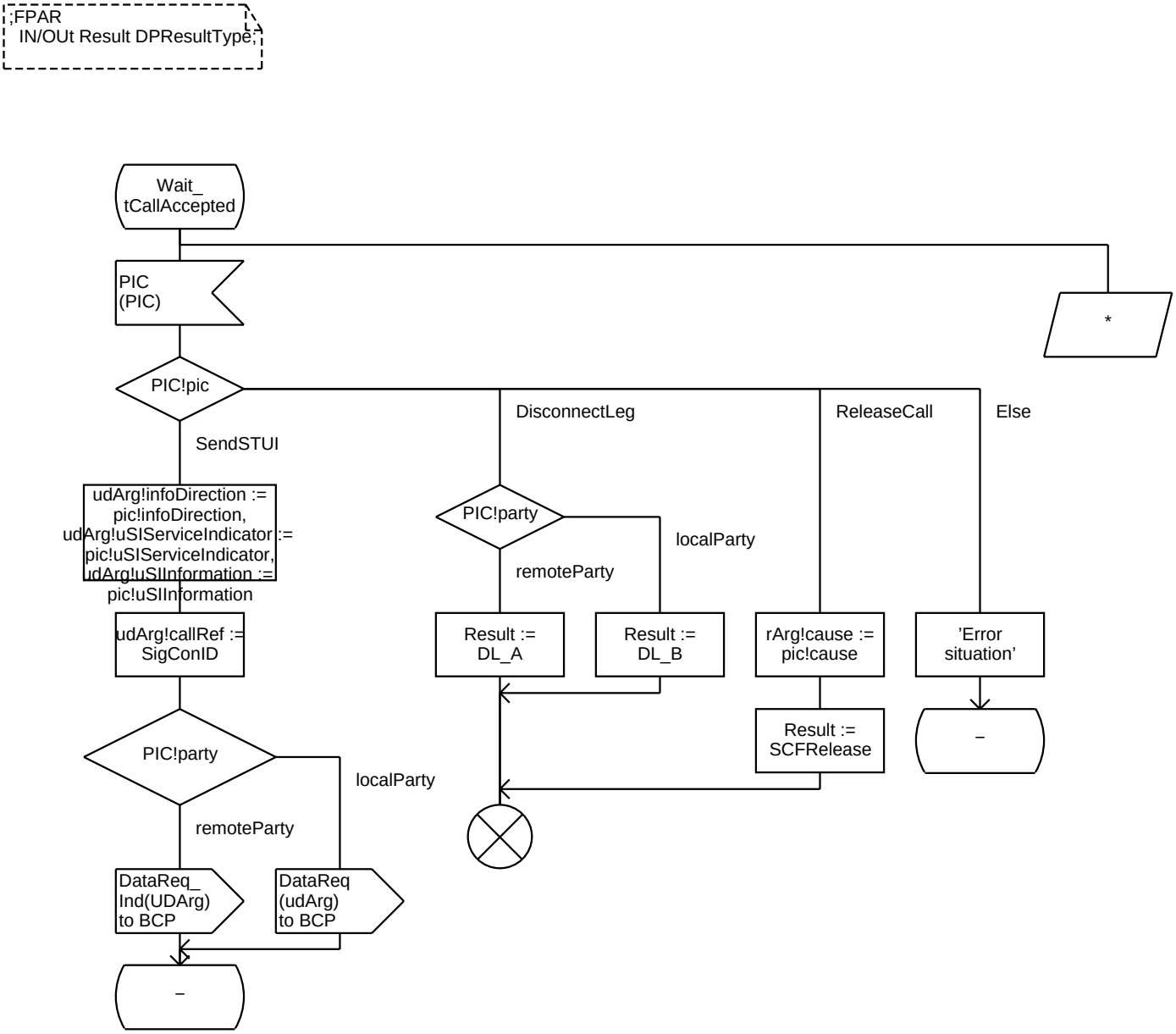
Procedure DP_tSuspend 3(3)



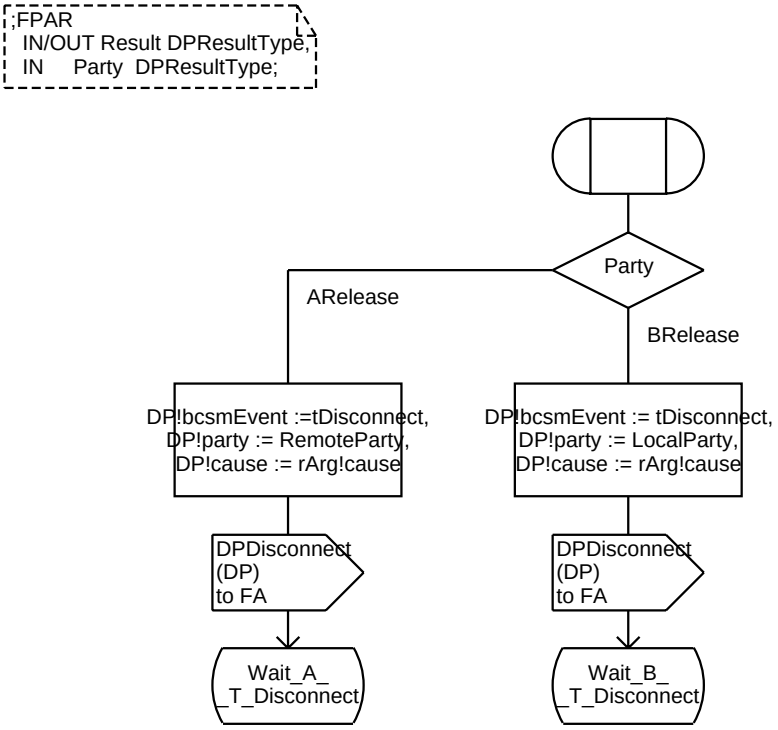
Procedure DP_tCall_Accepted 1(2)



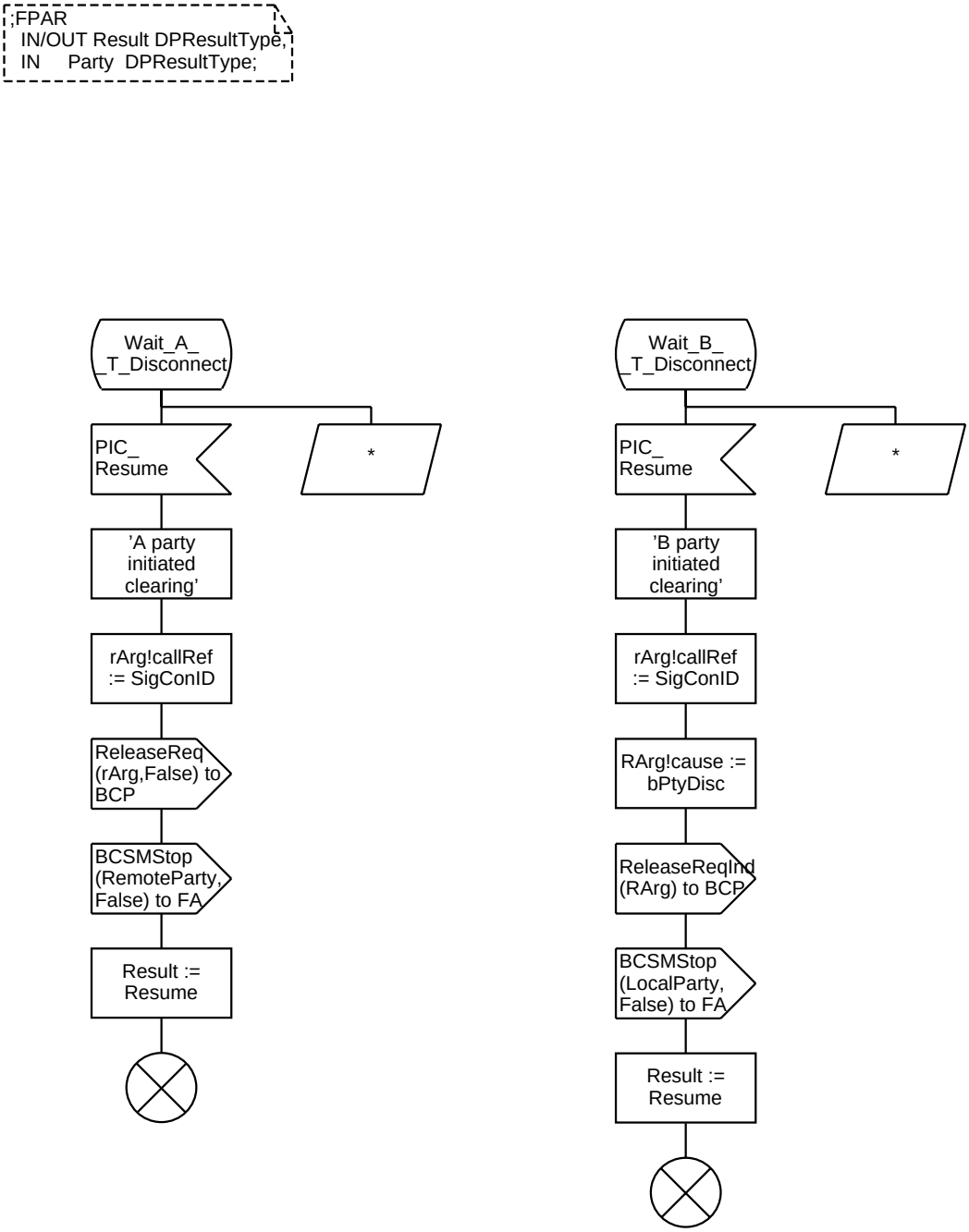
Procedure DP_tCall_Accepted 2(2)



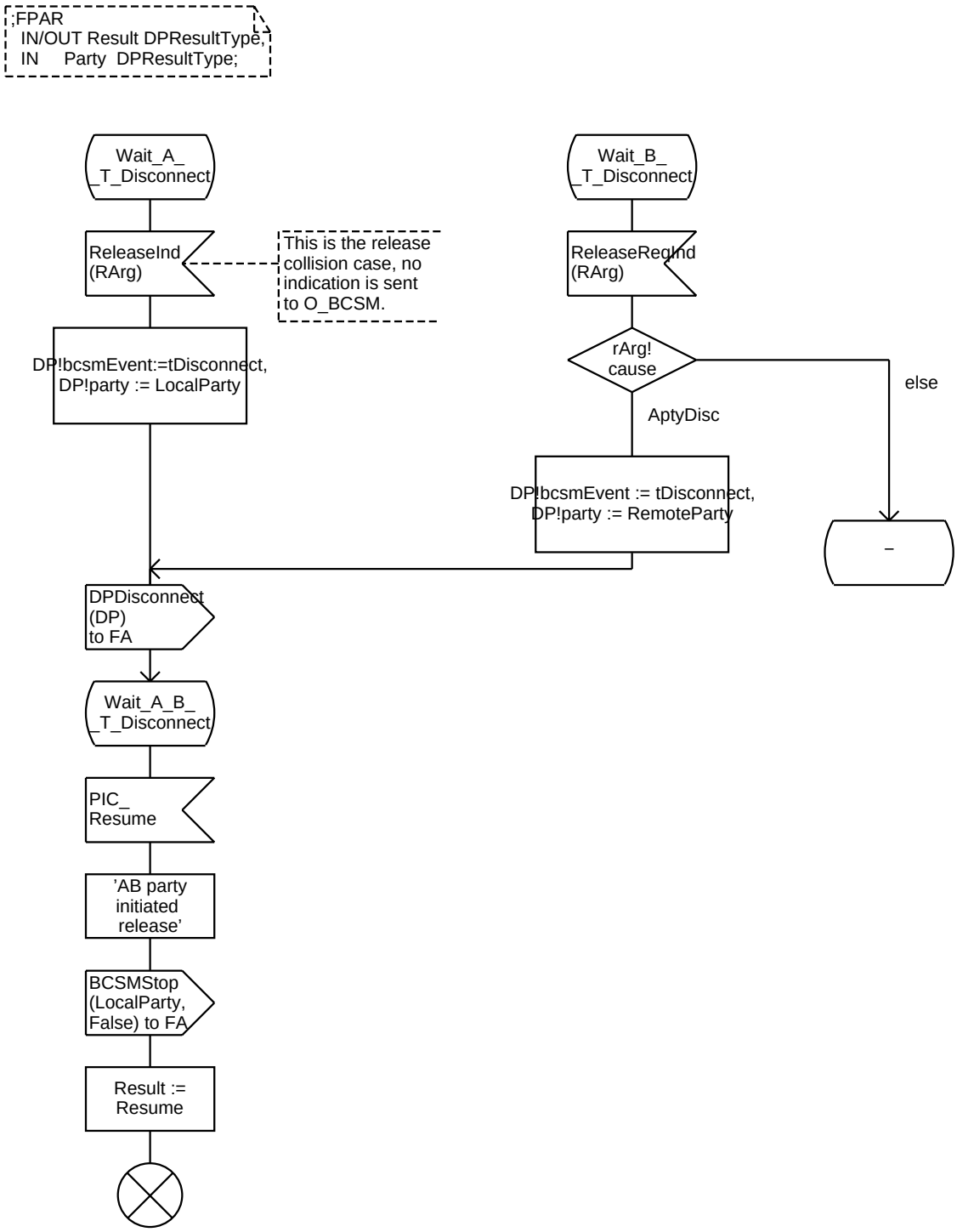
Procedure DP_tDisconnect 1(8)



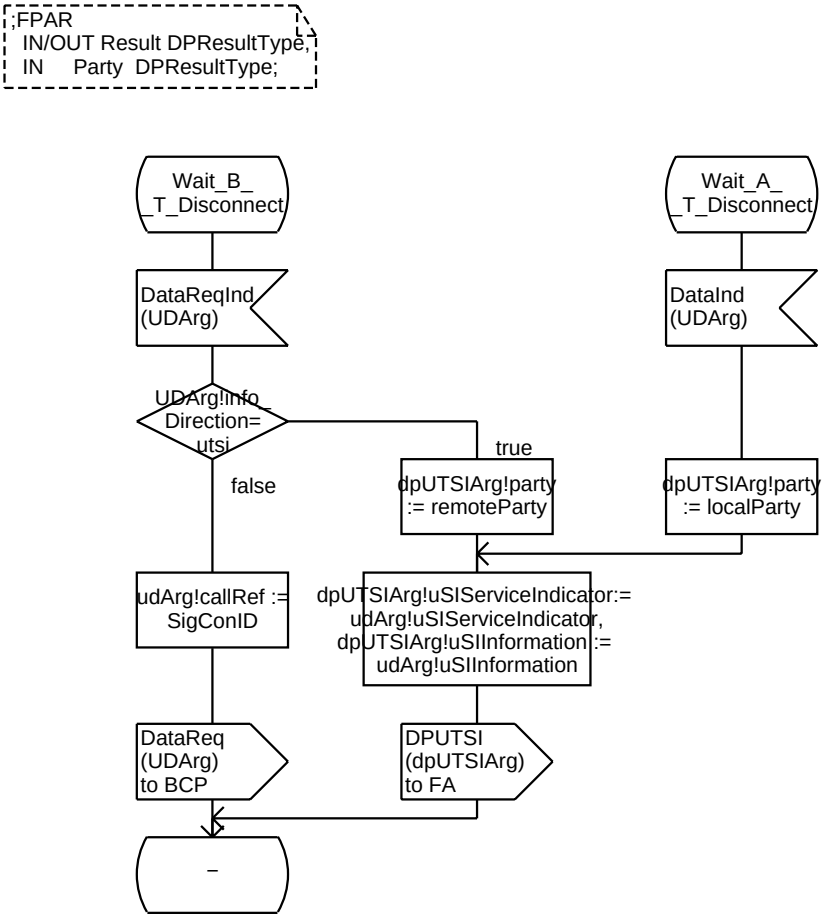
Procedure DP_tDisconnect 2(8)



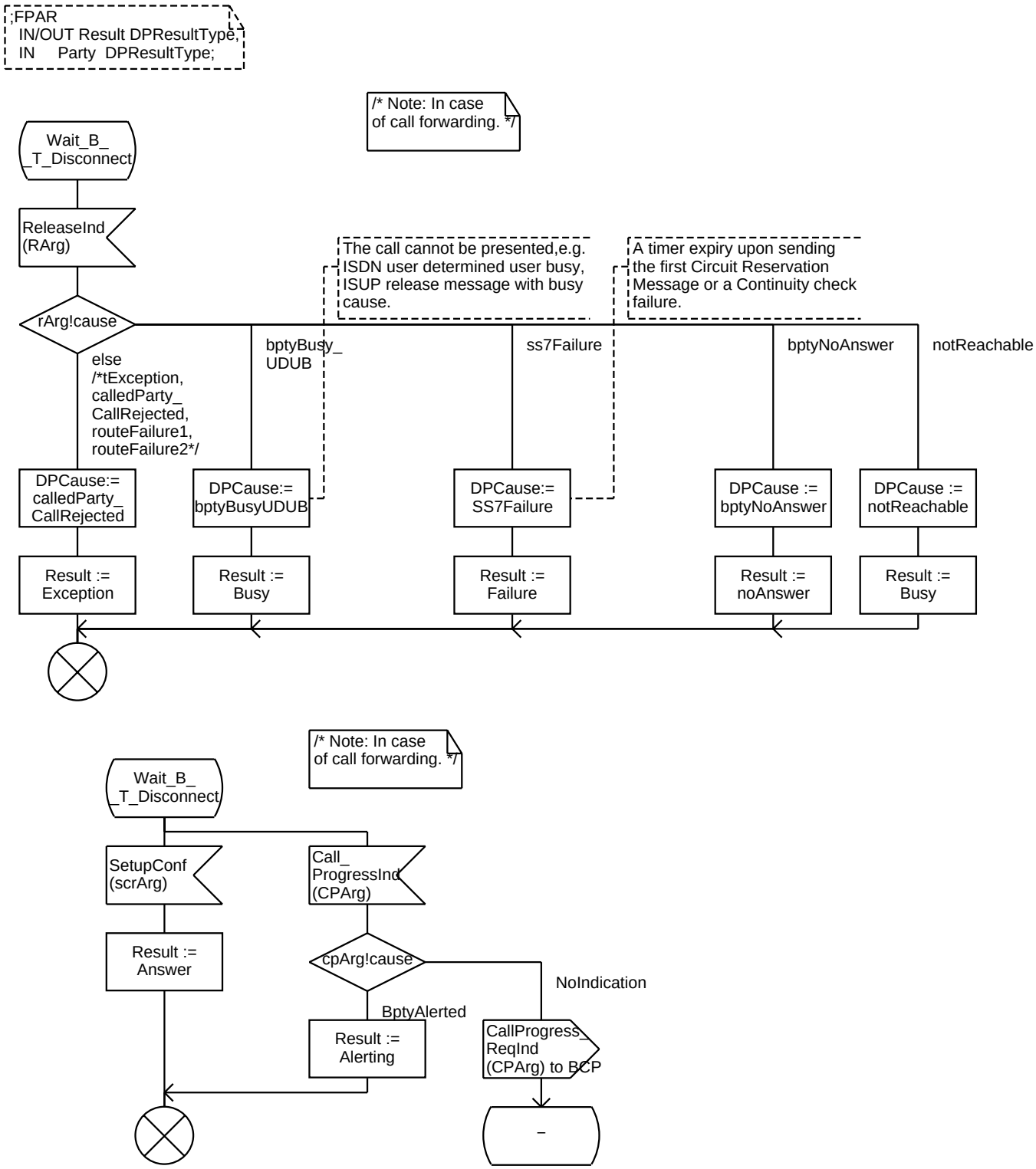
Procedure DP_tDisconnect 3(8)



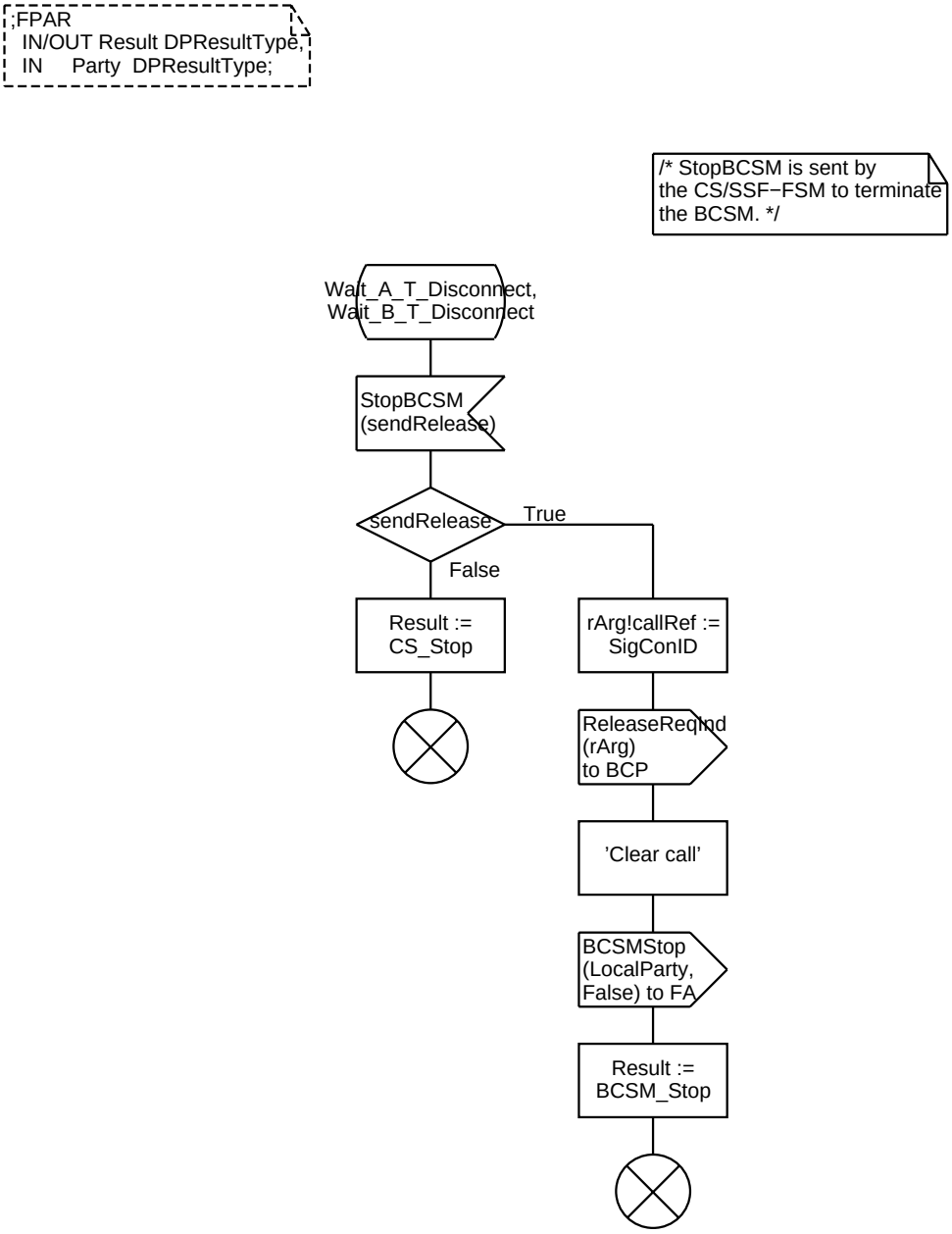
Procedure DP_tDisconnect 4(8)



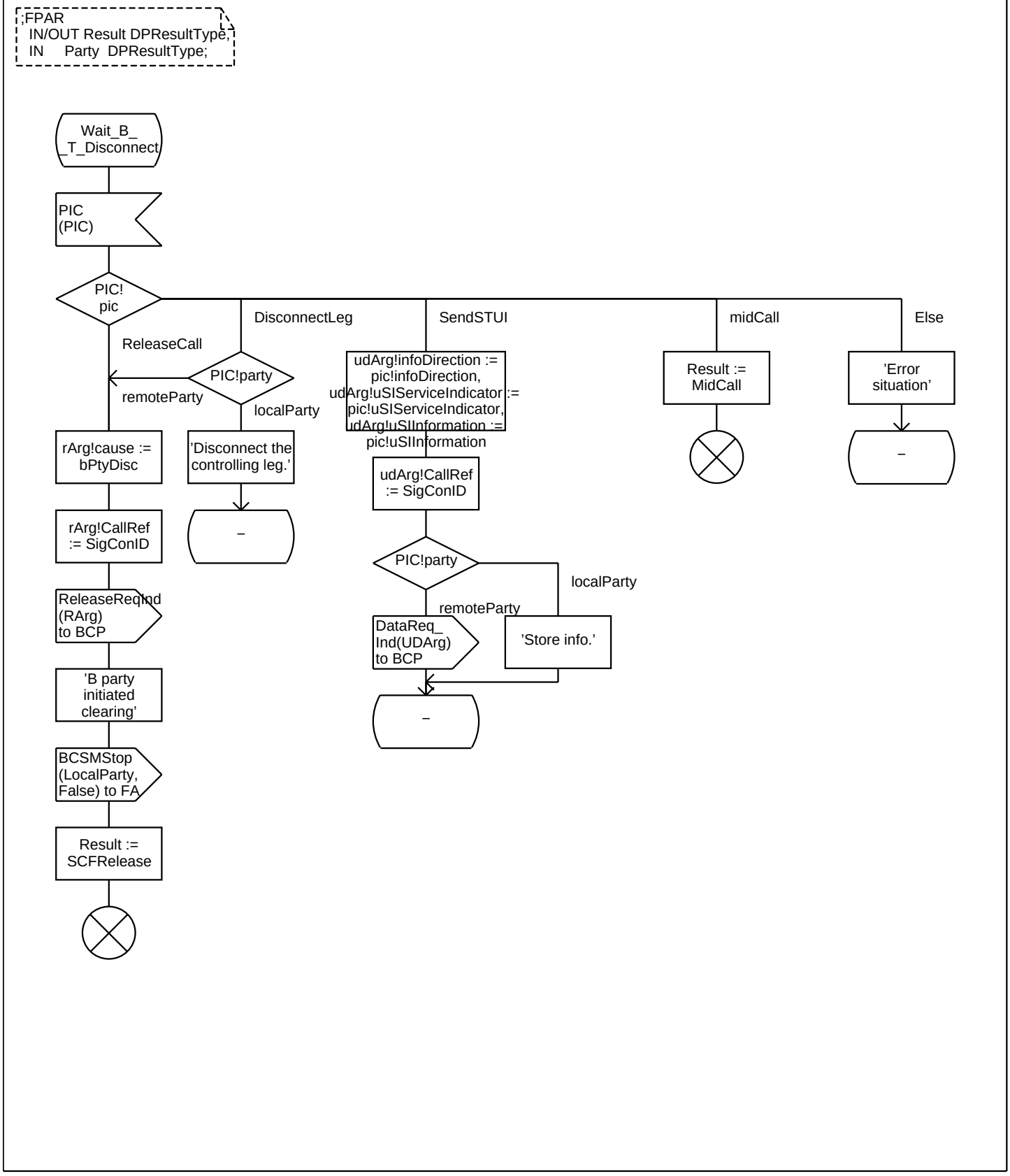
Procedure DP_tDisconnect 5(8)



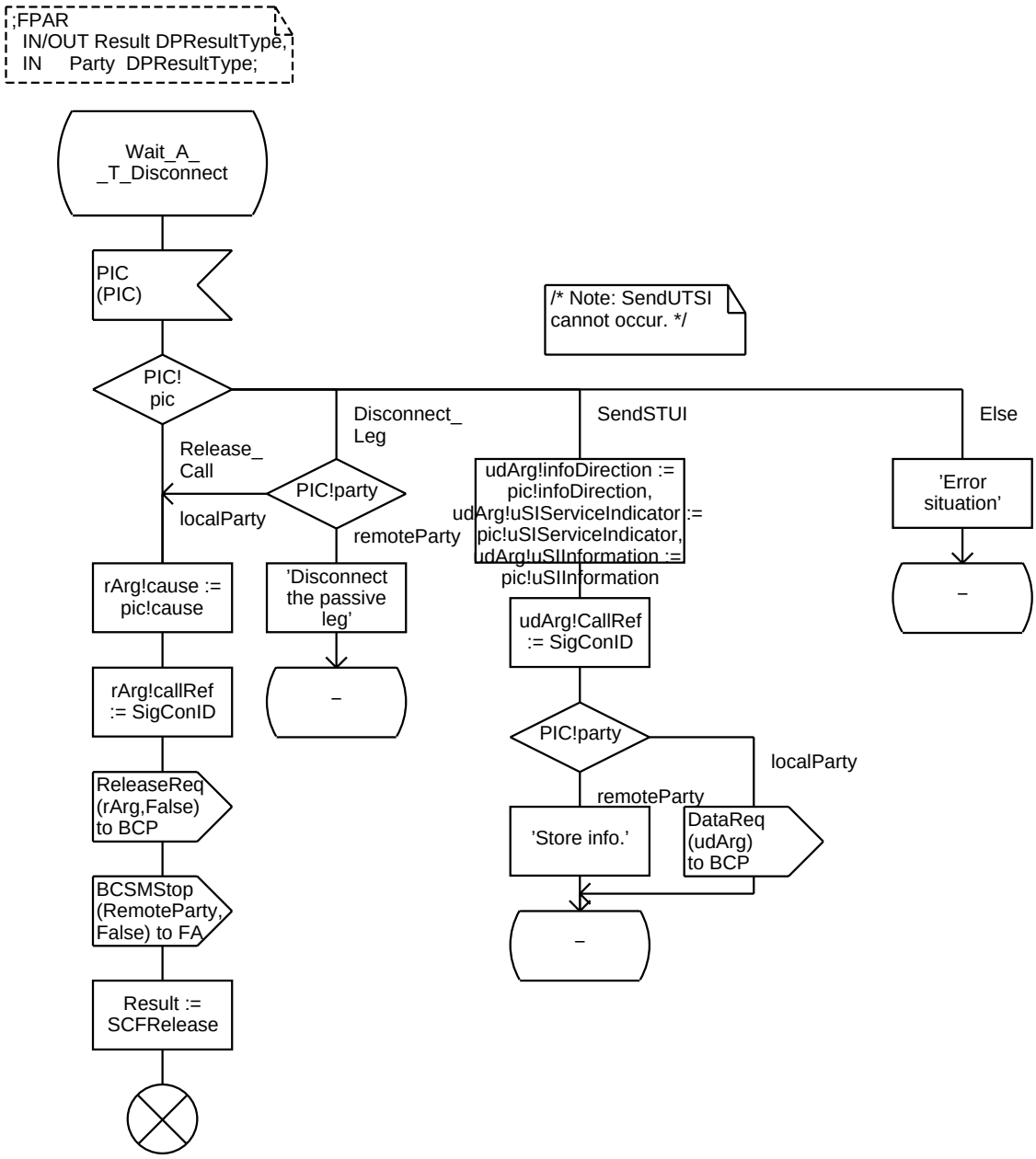
Procedure DP_tDisconnect 6(8)



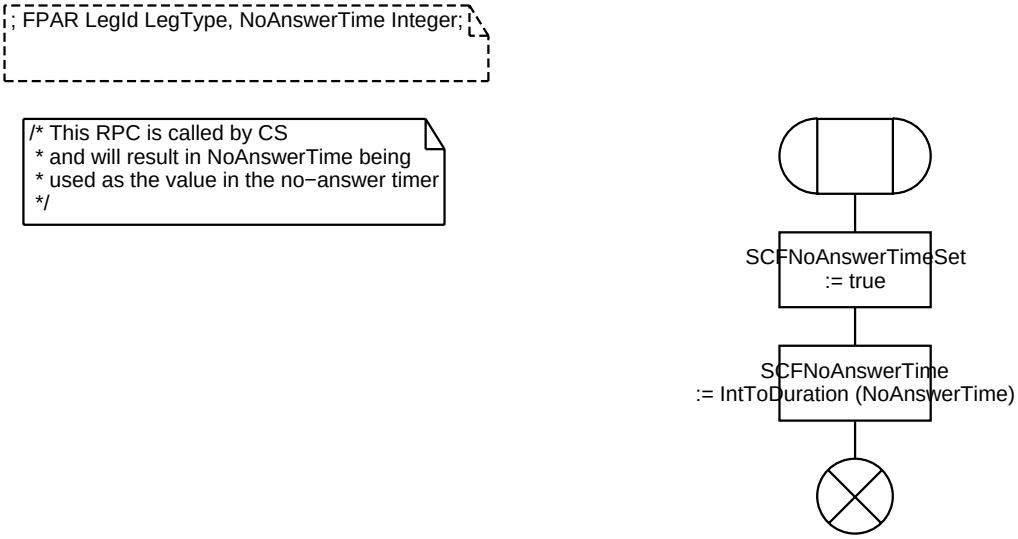
Procedure DP_tDisconnect 7(8)



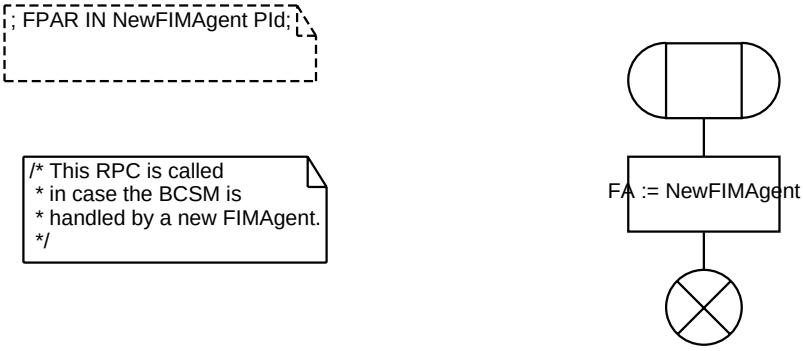
Procedure DP_tDisconnect 8(8)



Procedure <<Process Type TerminatingBCSM>> SetNoAnswerTime 1(1)



Procedure <<Process Type TerminatingBCSM>> SetFIMAgent 1(1)



```
USE INCS3Internals;
USE INCS3datatypes;
USE INCS3opsargs;
USE INCS3BundleArg;
USE CS3_TCAP;
USE CS3_INAP;
```

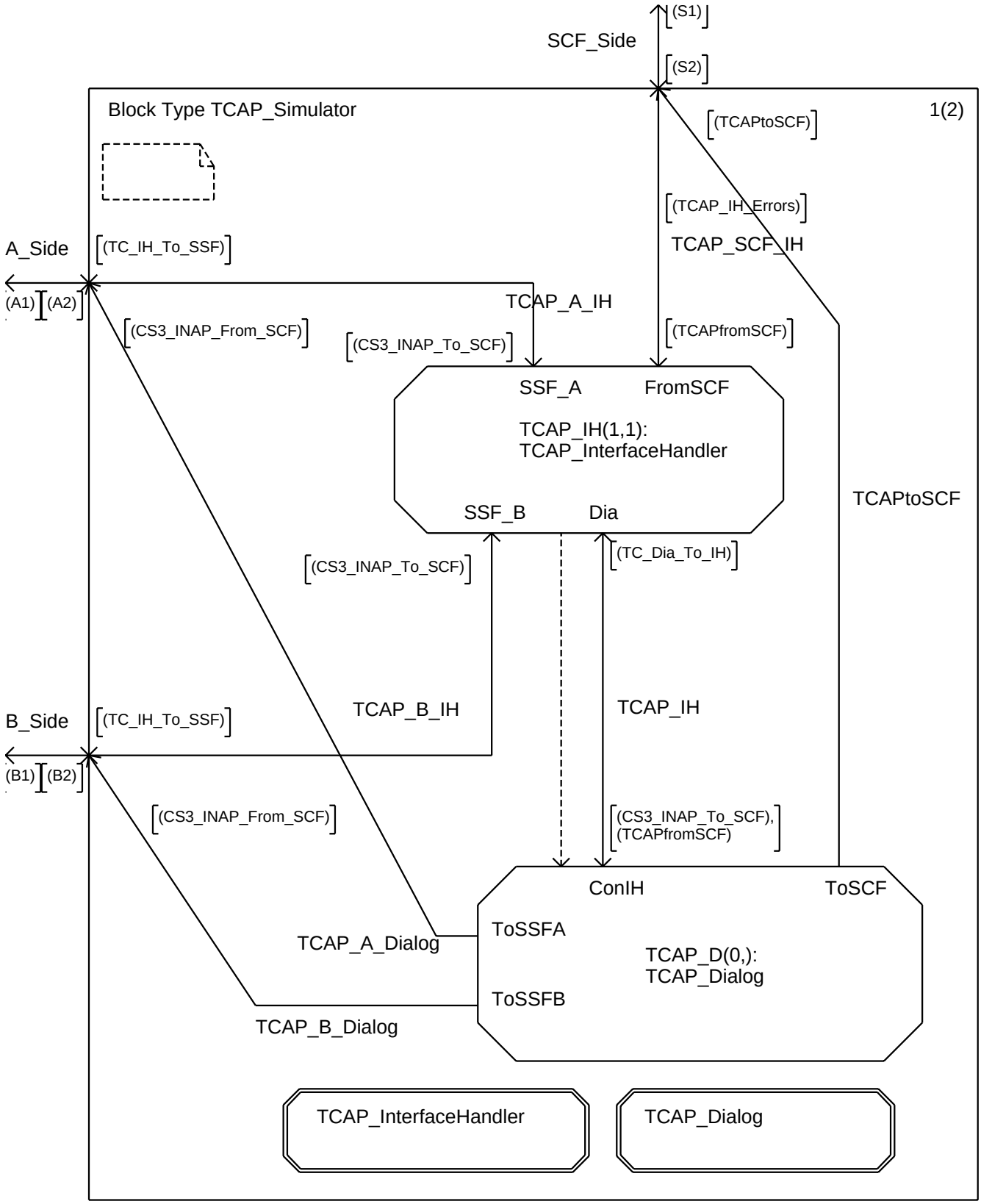
Package TCAP_Simulator 1(1)

```
/* This TCAP simulator models
 * the behaviour of the TCAP service as
 * seen at the interface between TCAP and the
 * SCF.
 *
 * This model does TCAP dialog handling and
 * some form of encoding INAP operations to TCAP.
 * This model of TCAP is very limited; for instance,
 * it supports at most 1 SCF.
 */
```

TCAP_Simulator

```
/* TCAP interface to SSF */

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

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

SIGNALLIST S1 = (TCAPtoSCF), (TCAP_IH_Errors);
SIGNALLIST S2 = (TCAPfromSCF);
SIGNALLIST A1 = (CS3_INAP_From_SCF), (TC_IH_To_SSF);
SIGNALLIST A2 = (CS3_INAP_To_SCF);
SIGNALLIST B1 = (A1);
SIGNALLIST B2 = (A2);
```


(S1)
SCF_Side
(S2)
Block Type TCAP_Simulator
1(2)
(TCAPtoSCF)
(TCAP_IH_Errors)
TCAP_SCF_IH
A_Side
(TC_IH_To_SSF)
(A1)
(A2)
TCAP_A_IH
(CS3_INAP_From_SCF)
(CS3_INAP_To_SCF)
(TCAPfromSCF)
SSF_A
FromSCF
TCAP_IH(1,1):
TCAP_InterfaceHandler
SSF_B
Dia
TCAPtoSCF
(CS3_INAP_To_SCF)
(TC_Dia_To_IH)
TCAP_B_IH
TCAP_IH
B_Side
(TC_IH_To_SSF)
(B1)
(B2)
(CS3_INAP_From_SCF)
(CS3_INAP_To_SCF),
(TCAPfromSCF)
ConIH
ToSCF
ToSSFA
TCAP_A_Dialog
TCAP_D(0,):
TCAP_Dialog
ToSSFB
TCAP_B_Dialog
TCAP_InterfaceHandler
TCAP_Dialog

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;
```

Process Type TCAP_InterfaceHandler 1(18)

```
NEWTYPE DialogType
  STRUCT
    status DialogStatusType;
    csa CSAID;
    dialogHandler PId;
  OPERATORS newDialog: DialogStatusType, CSAID, PId −> DialogType;
  OPERATOR newDialog REFERENCED;
ENDNEWTYPE;

/* Remark: Type CSAID (= Integer) is defined in ASN.1 file 'CS3i.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 REFERENCED;
  OPERATOR getDialogID REFERENCED;
  OPERATOR nextFreeDialogID REFERENCED;
ENDNEWTYPE;
```

operator
newDialog

operator
initDialogs

operator
getDialogID

operator
nextFreeDialogID

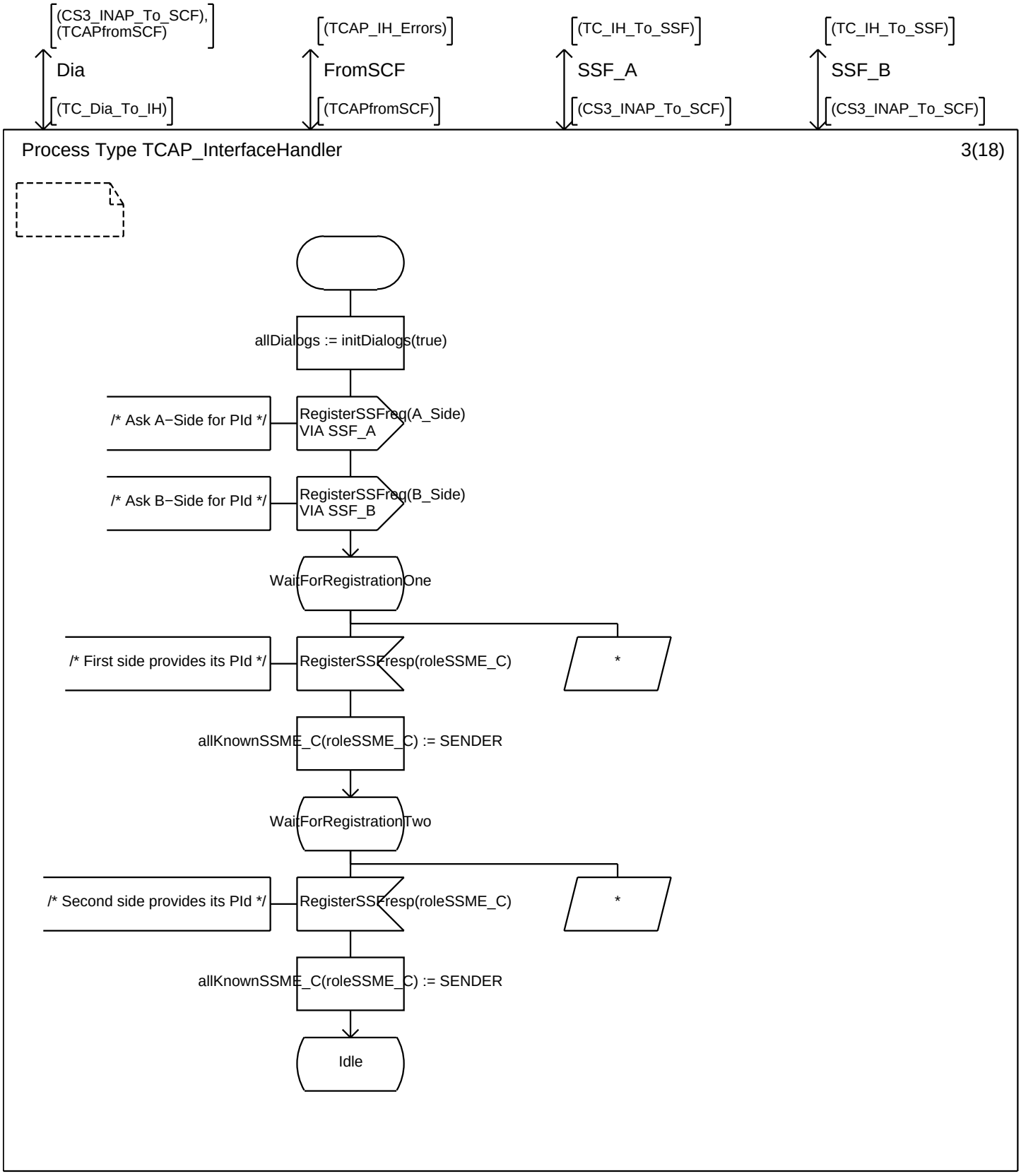
```
NEWTYPE SSME_CresourceType
  ARRAY (SSME_CroleType,PId); /* SSME_CroleType (= A_Side or B_Side) is defined on system level */
ENDNEWTYPE;
```

Process Type TCAP_InterfaceHandler 2(18)

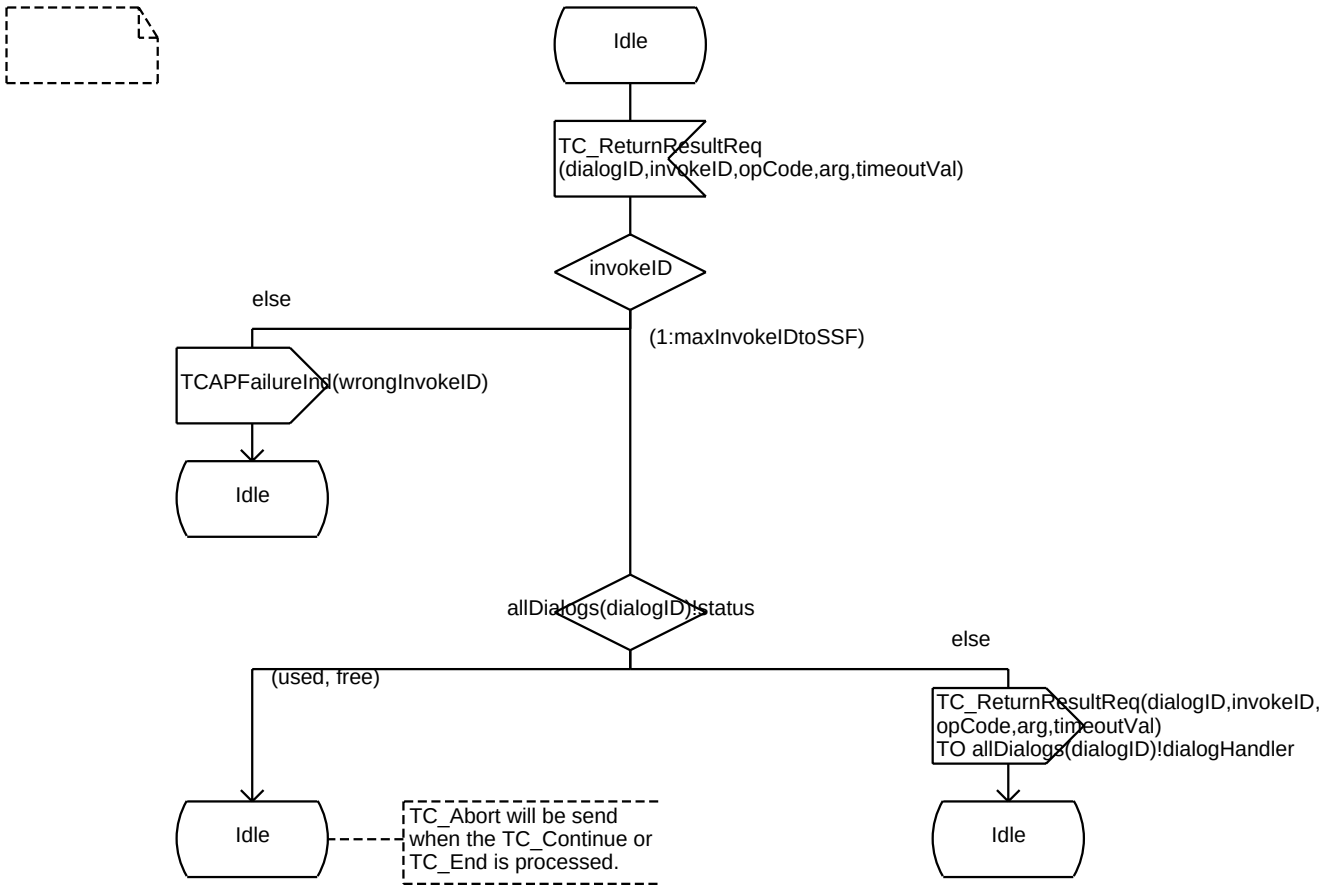
```
DCL allKnownSSME_C SSME_CresourceType;
DCL roleSSME_C SSME_CroleType;

DCL allDialogs AllDialogsType;
```

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

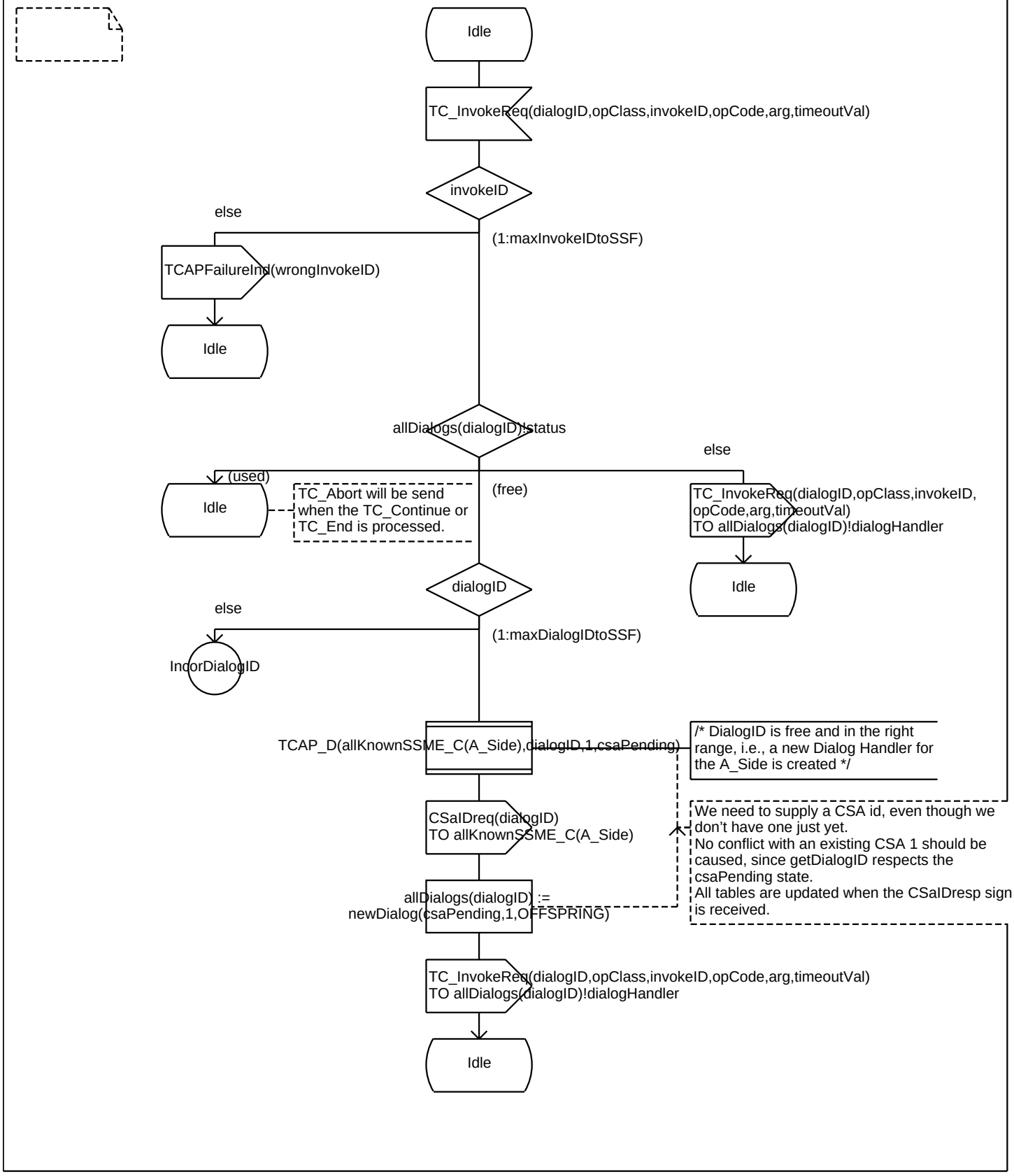
```
DCL arg ArgType; /* refers to ASN.1 Definition in INCS3BundleArg */
DCL errArg errorArg;
DCL compEnd Boolean;
```


(CS3_INAP_To_SCF), (TCAPfromSCF)
Dia
(TC_Dia_To_IH)
(TCAP_IH_Errors)
FromSCF
(TCAPfromSCF)
(TC_IH_To_SSF)
SSF_A
(CS3_INAP_To_SCF)
(TC_IH_To_SSF)
SSF_B
(CS3_INAP_To_SCF)
Process Type TCAP_InterfaceHandler
3(18)
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(roleSSME_C)
*
allKnownSSME_C(roleSSME_C) := SENDER
WaitForRegistrationTwo
/* Second side provides its PId */
RegisterSSFresp(roleSSME_C)
*
allKnownSSME_C(roleSSME_C) := SENDER
Idle

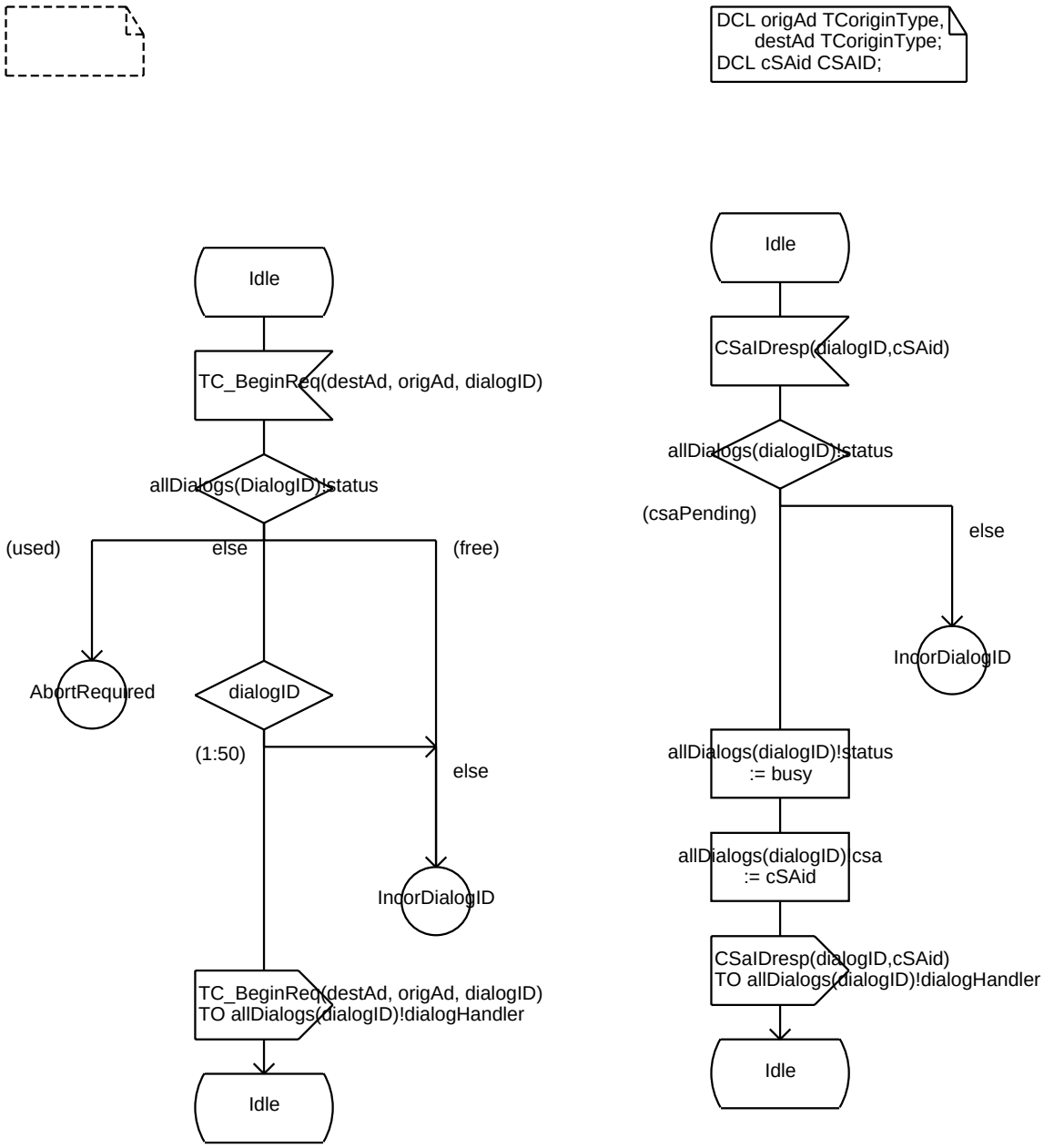
Process Type TCAP_InterfaceHandler 4(18)



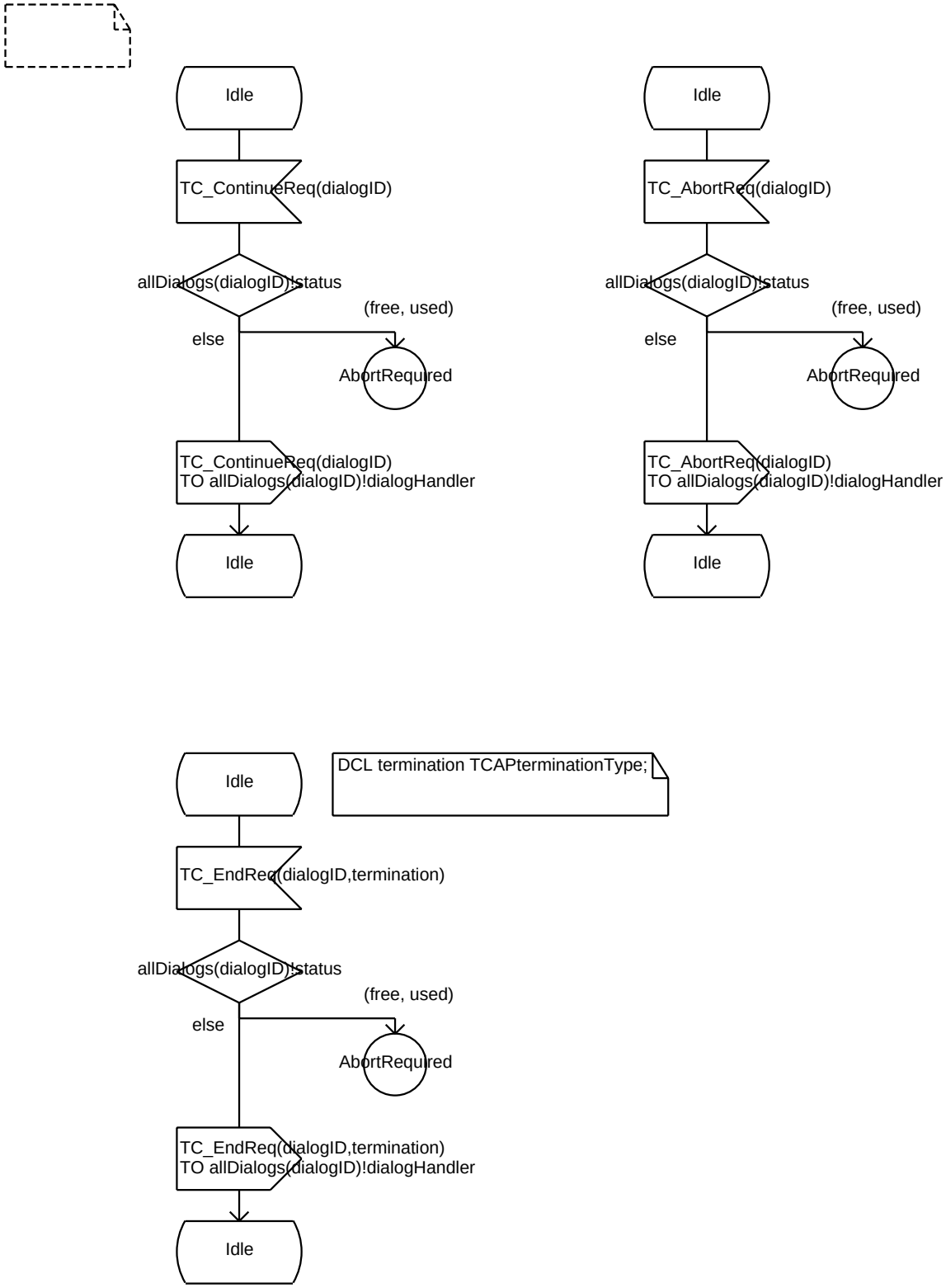
Process Type TCAP_InterfaceHandler 5(18)



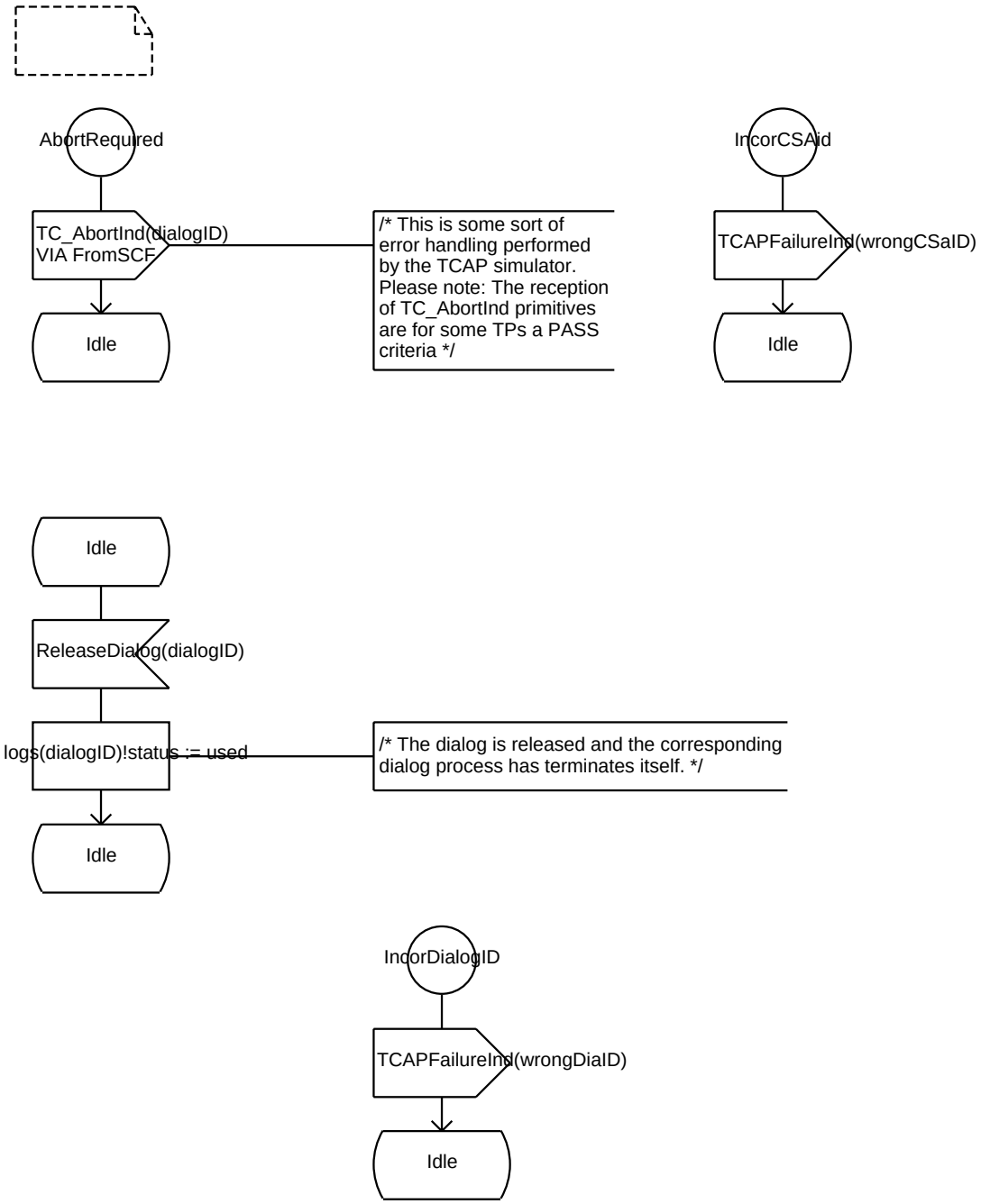
Process Type TCAP_InterfaceHandler 6(18)



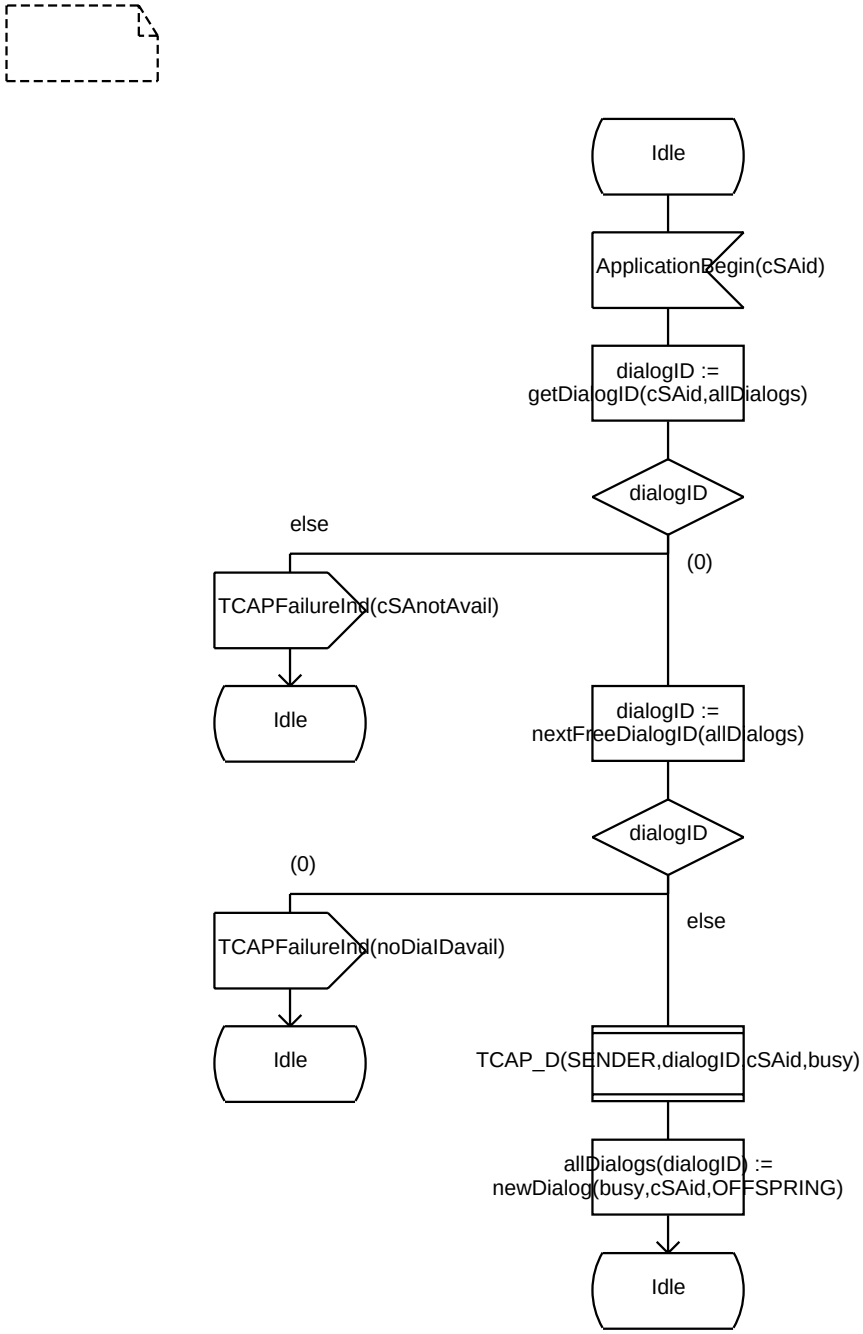
Process Type TCAP_InterfaceHandler 7(18)



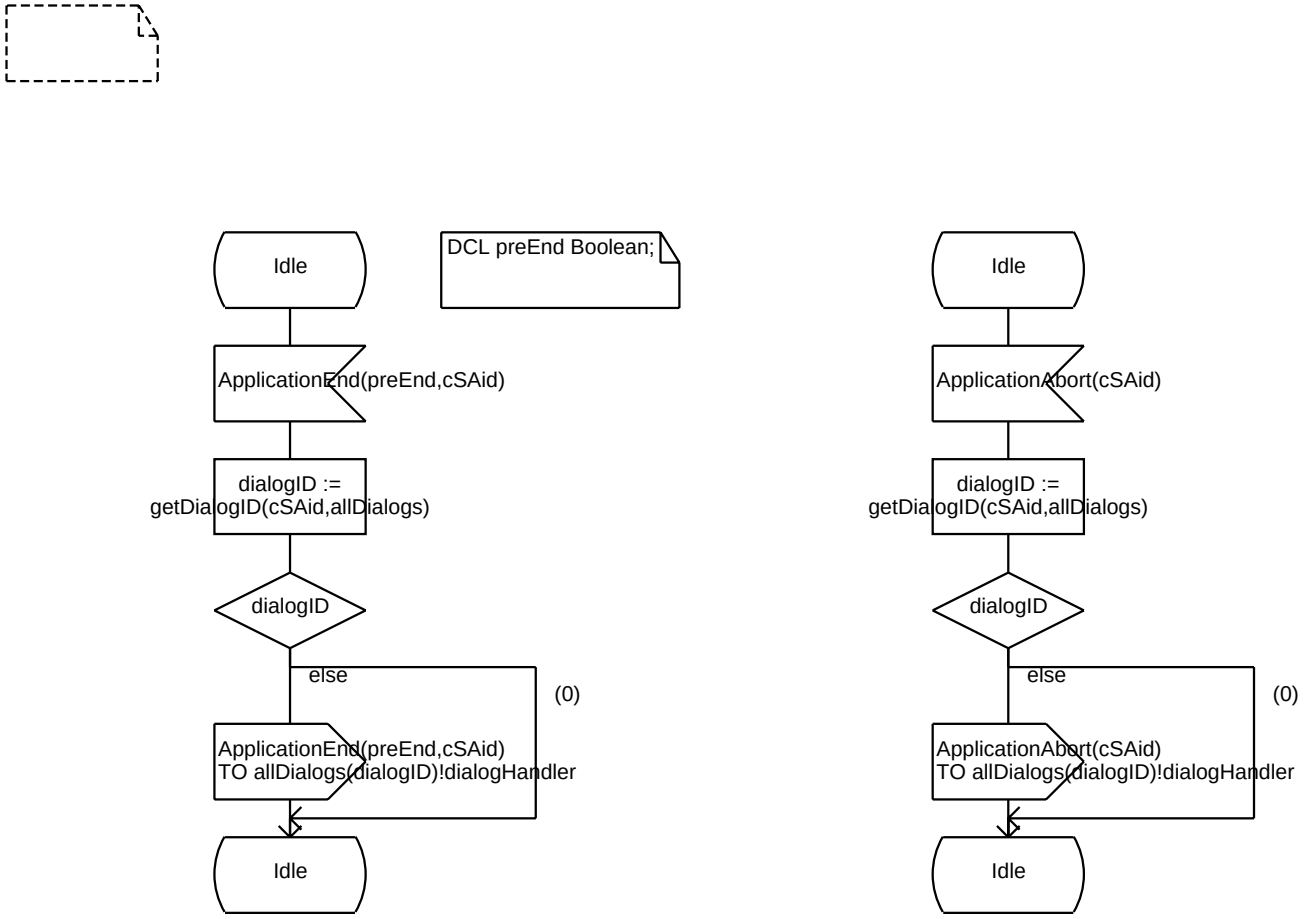
Process Type TCAP_InterfaceHandler 8(18)



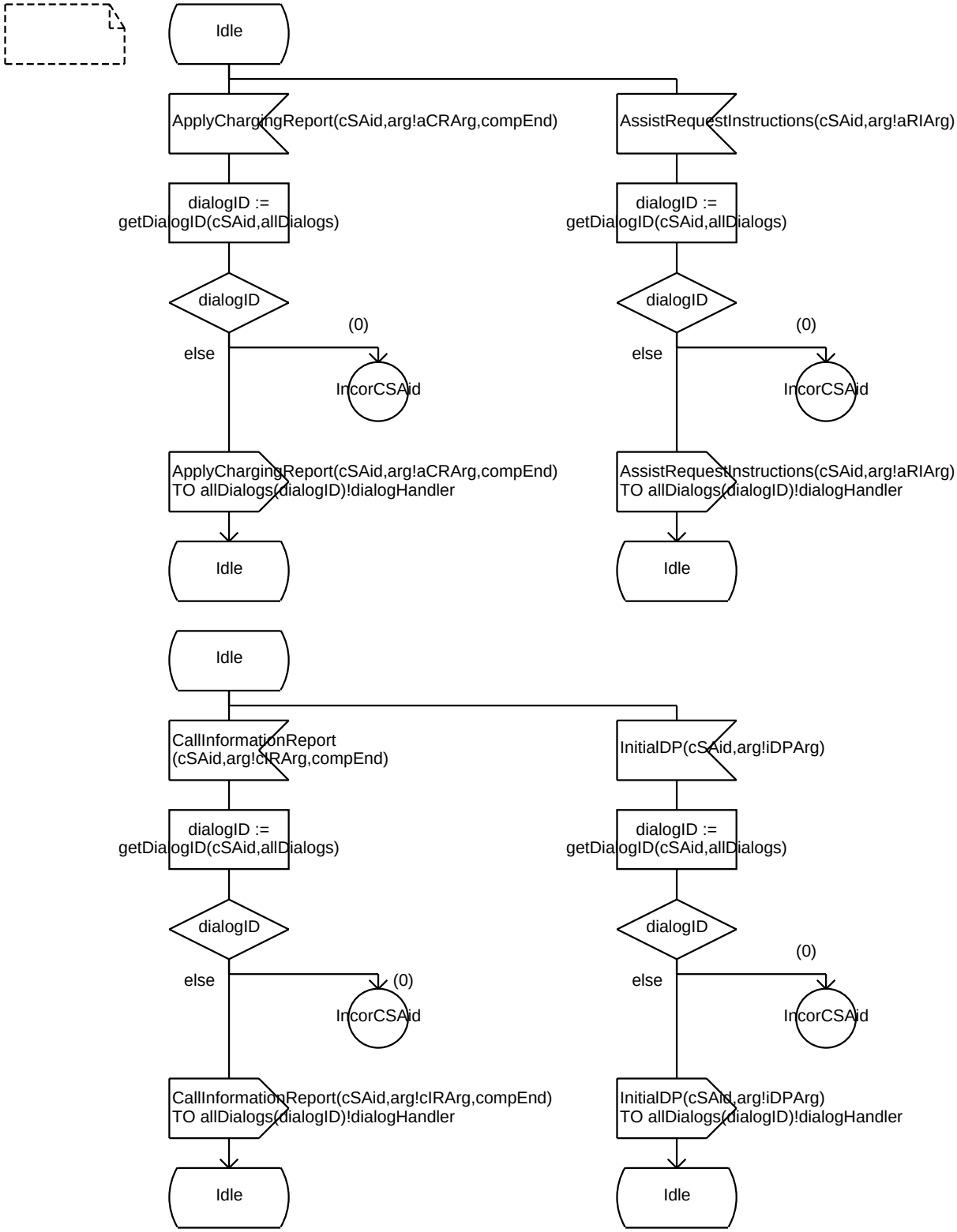
Process Type TCAP_InterfaceHandler 9(18)



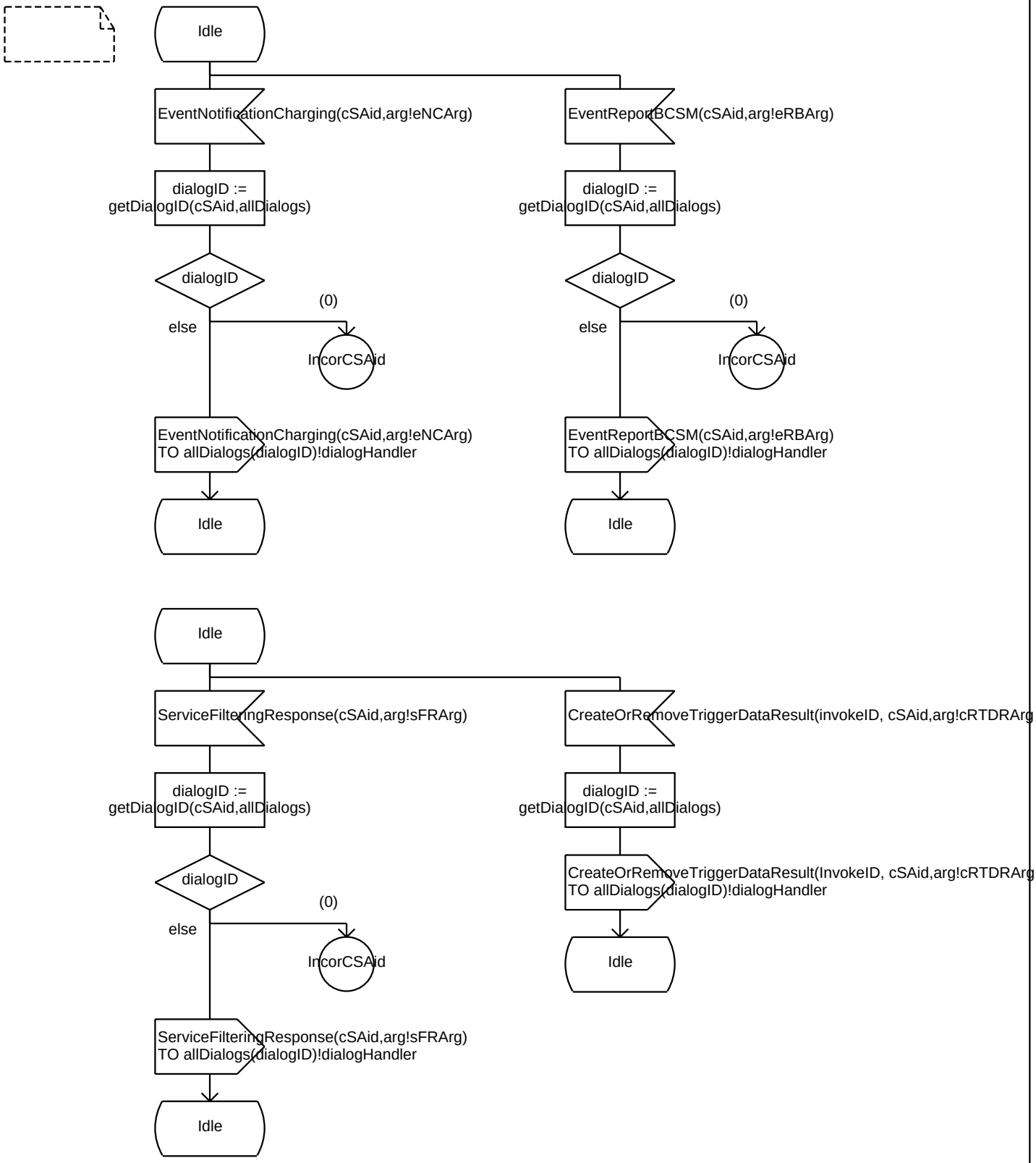
Process Type TCAP_InterfaceHandler 10(18)



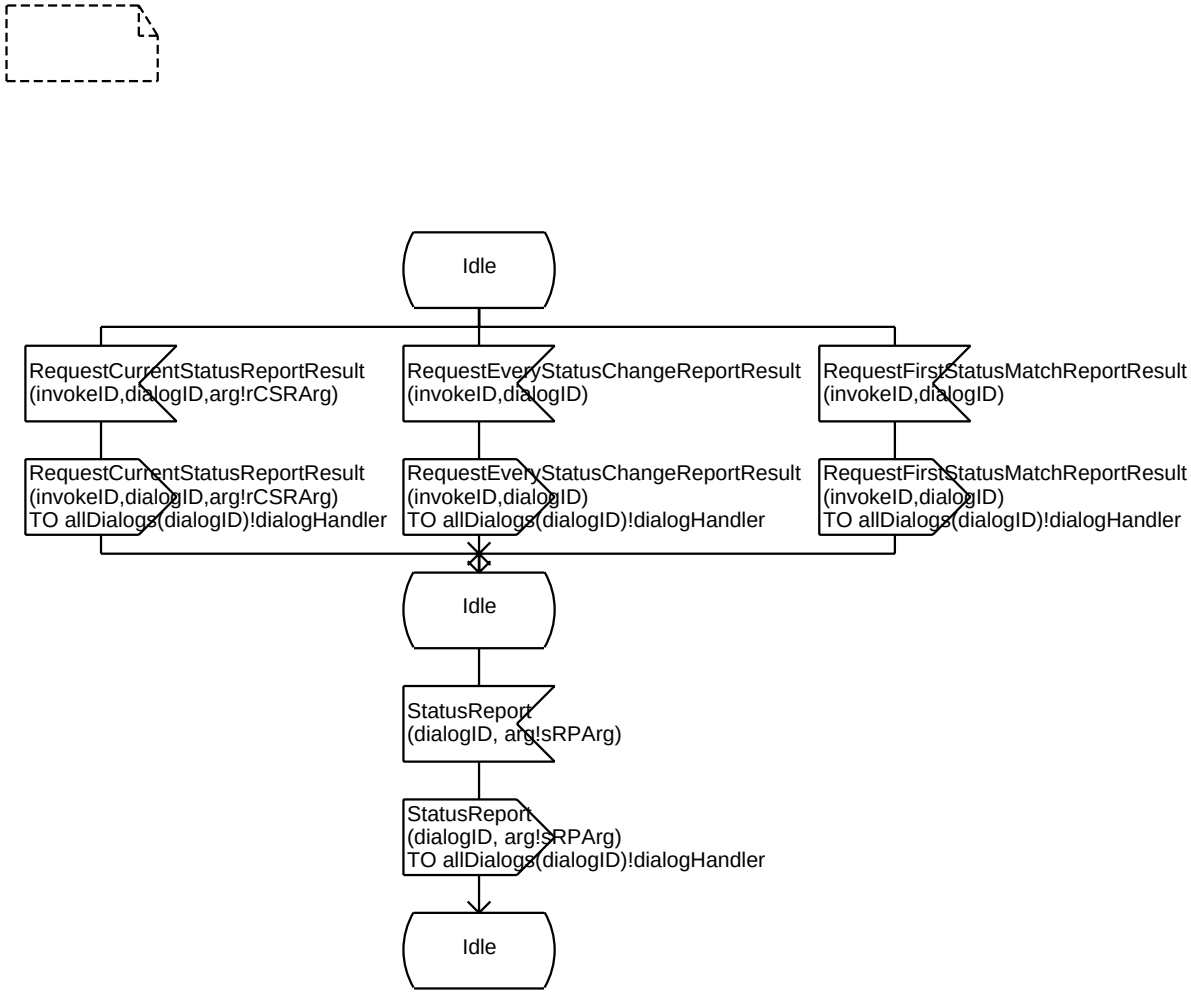
Process Type TCAP_InterfaceHandler 11(18)



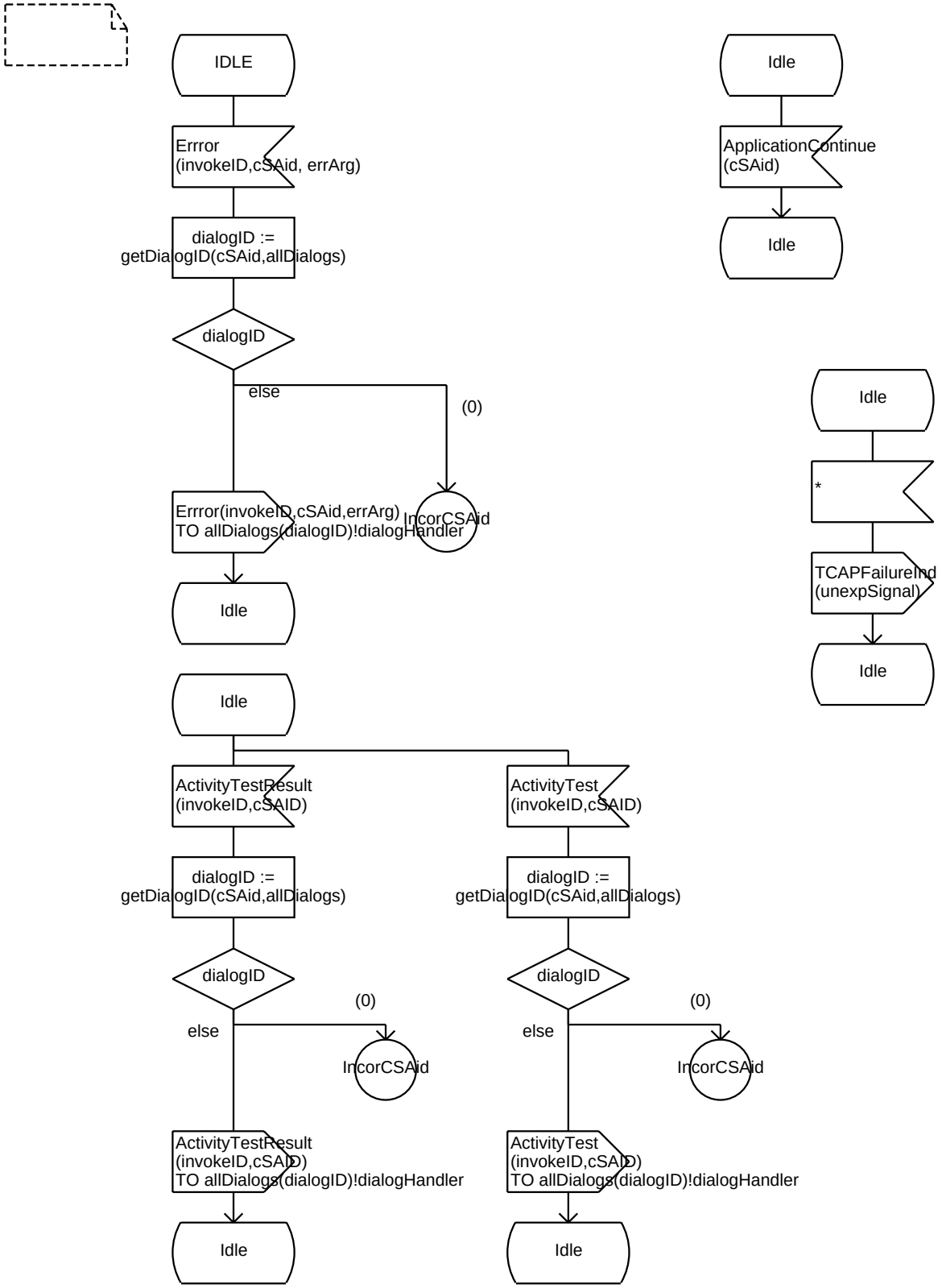
Process Type TCAP_InterfaceHandler 12(18)



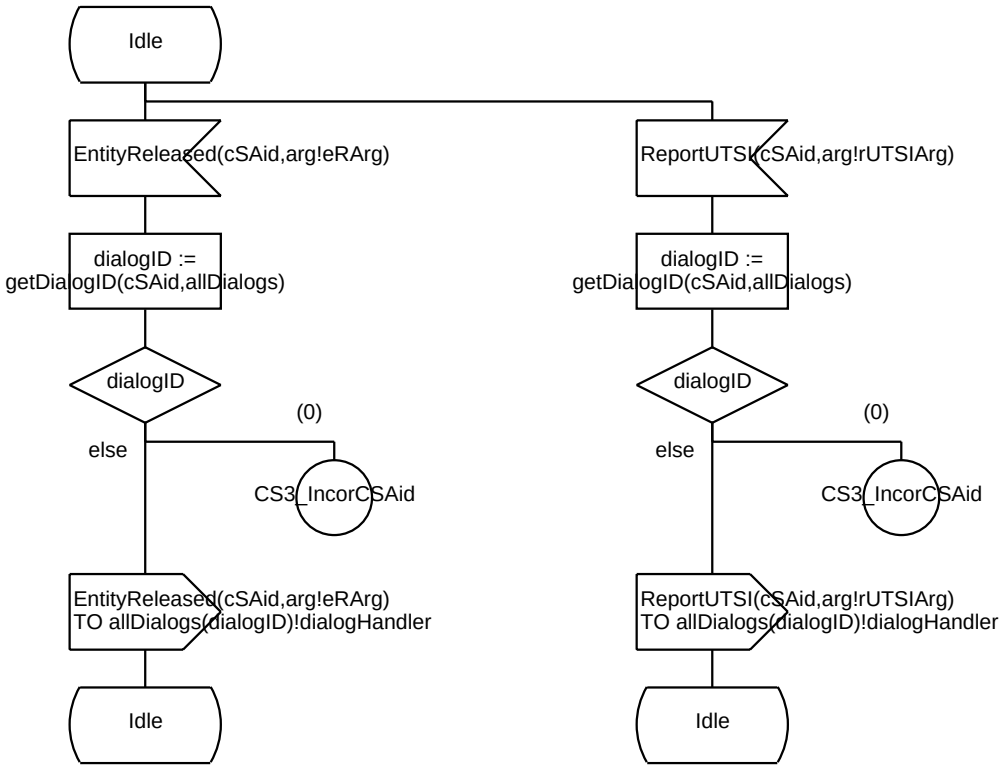
Process Type TCAP_InterfaceHandler 13(18)



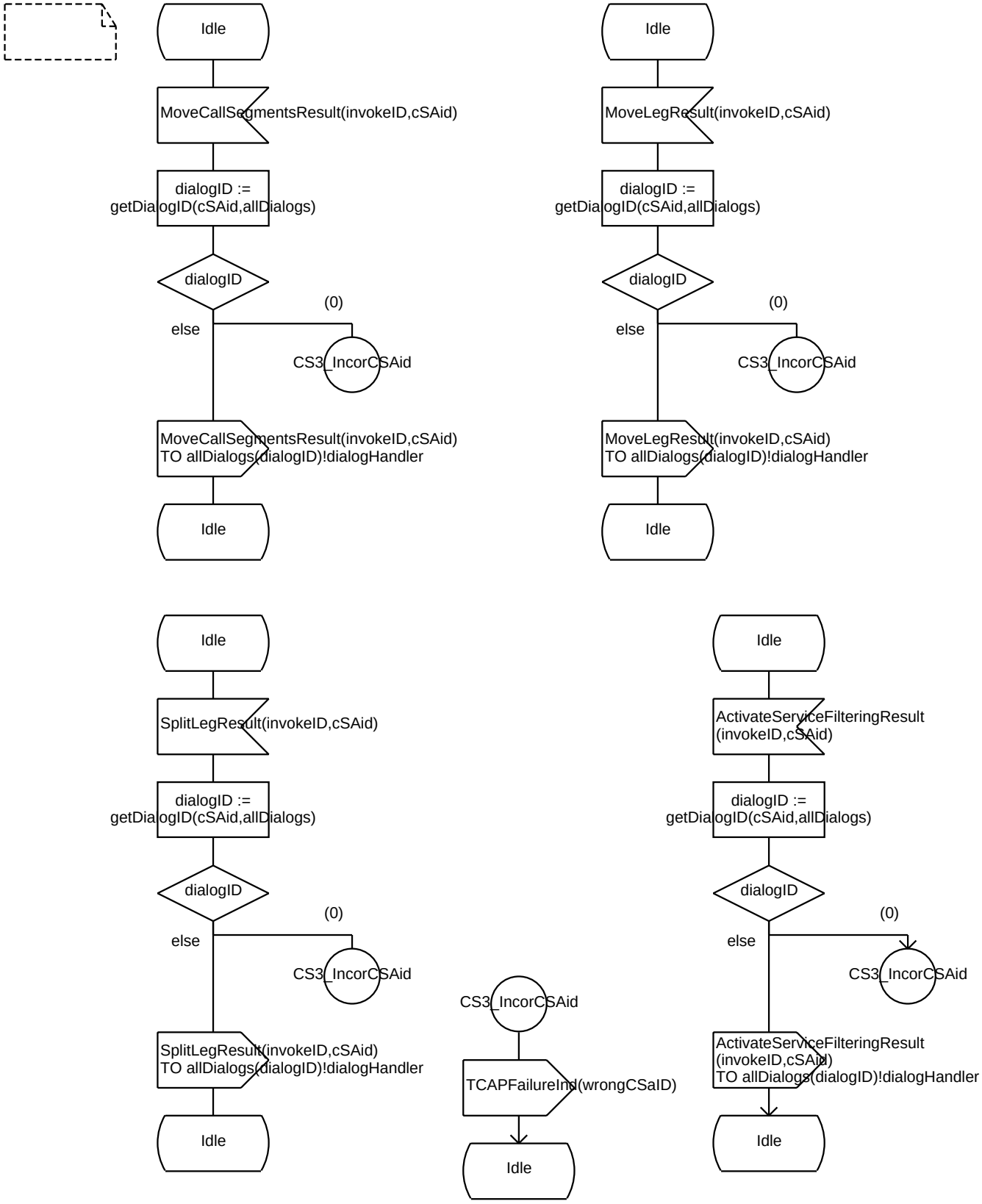
Process Type TCAP_InterfaceHandler 14(18)



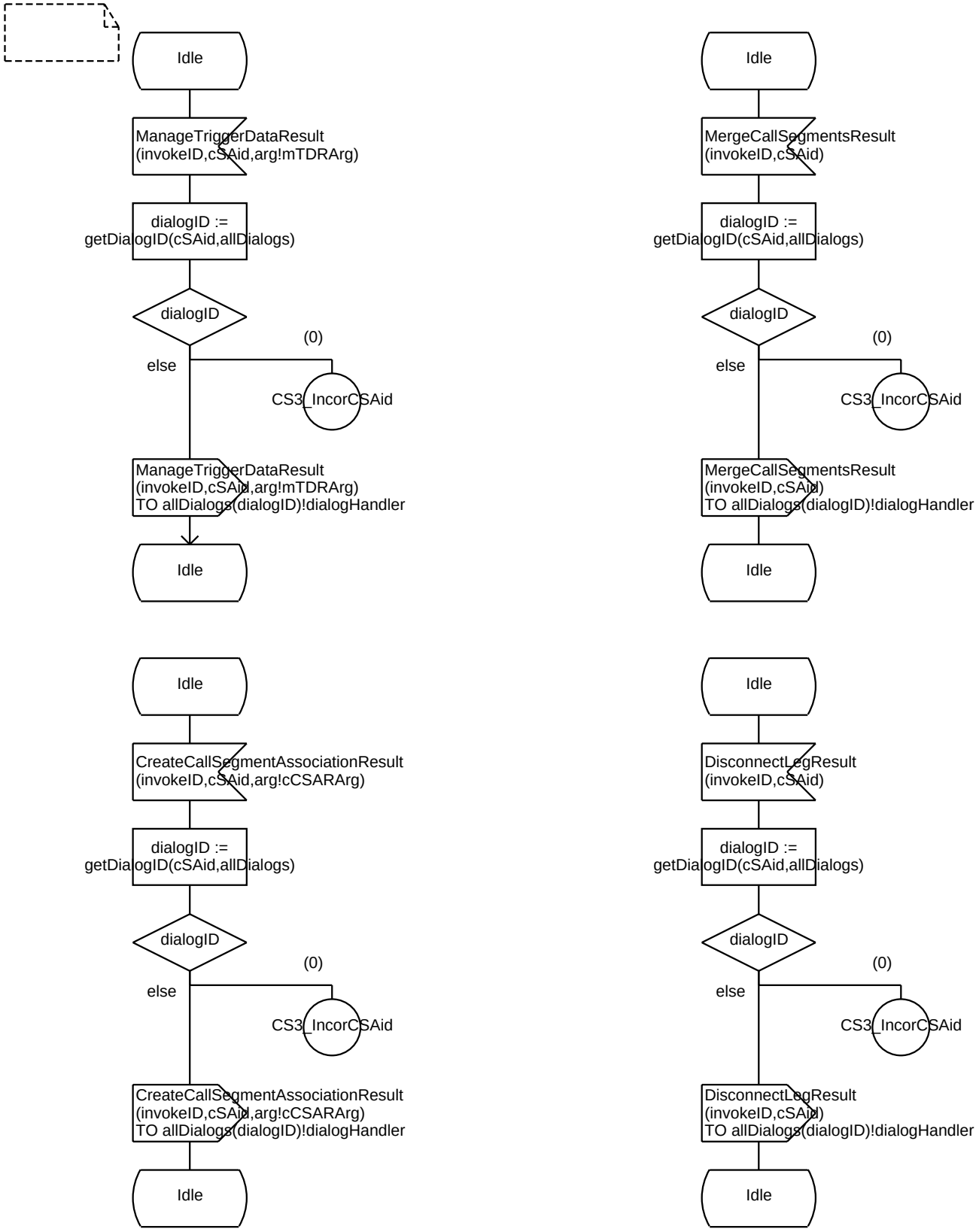
Process Type TCAP_InterfaceHandler 15(18)



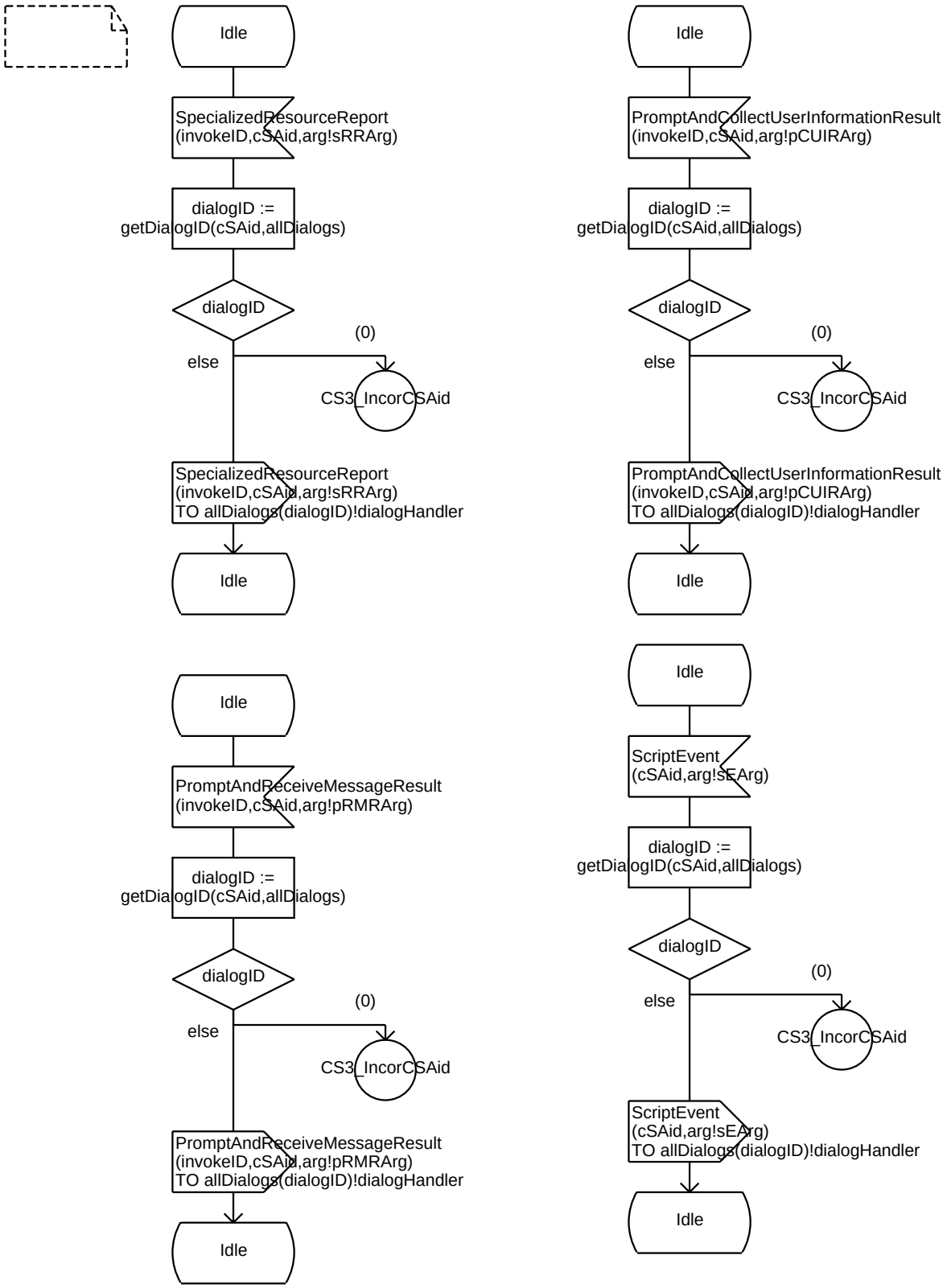
Process Type TCAP_InterfaceHandler 16(18)



Process Type TCAP_InterfaceHandler 17(18)

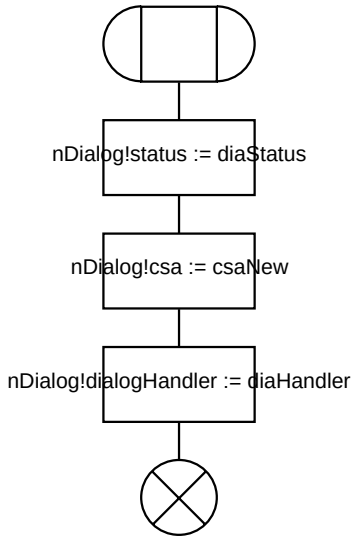


Process Type TCAP_InterfaceHandler 18(18)

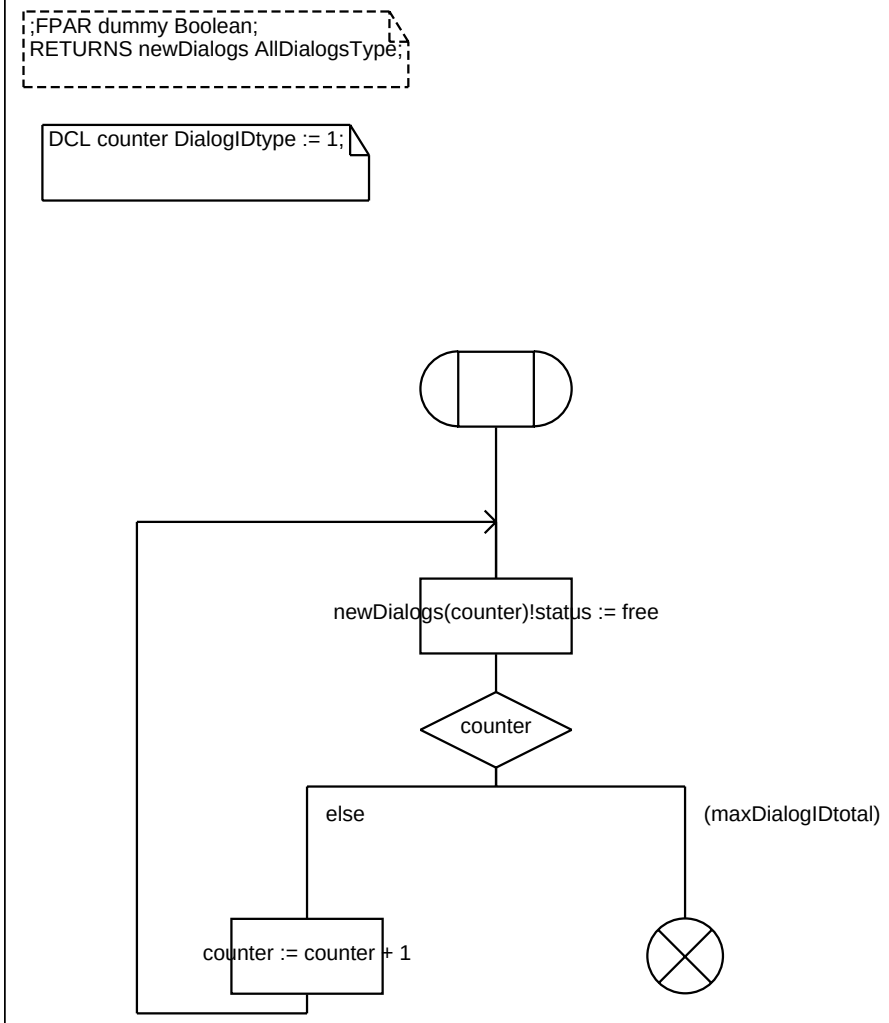


Operator newDialog 1(1)

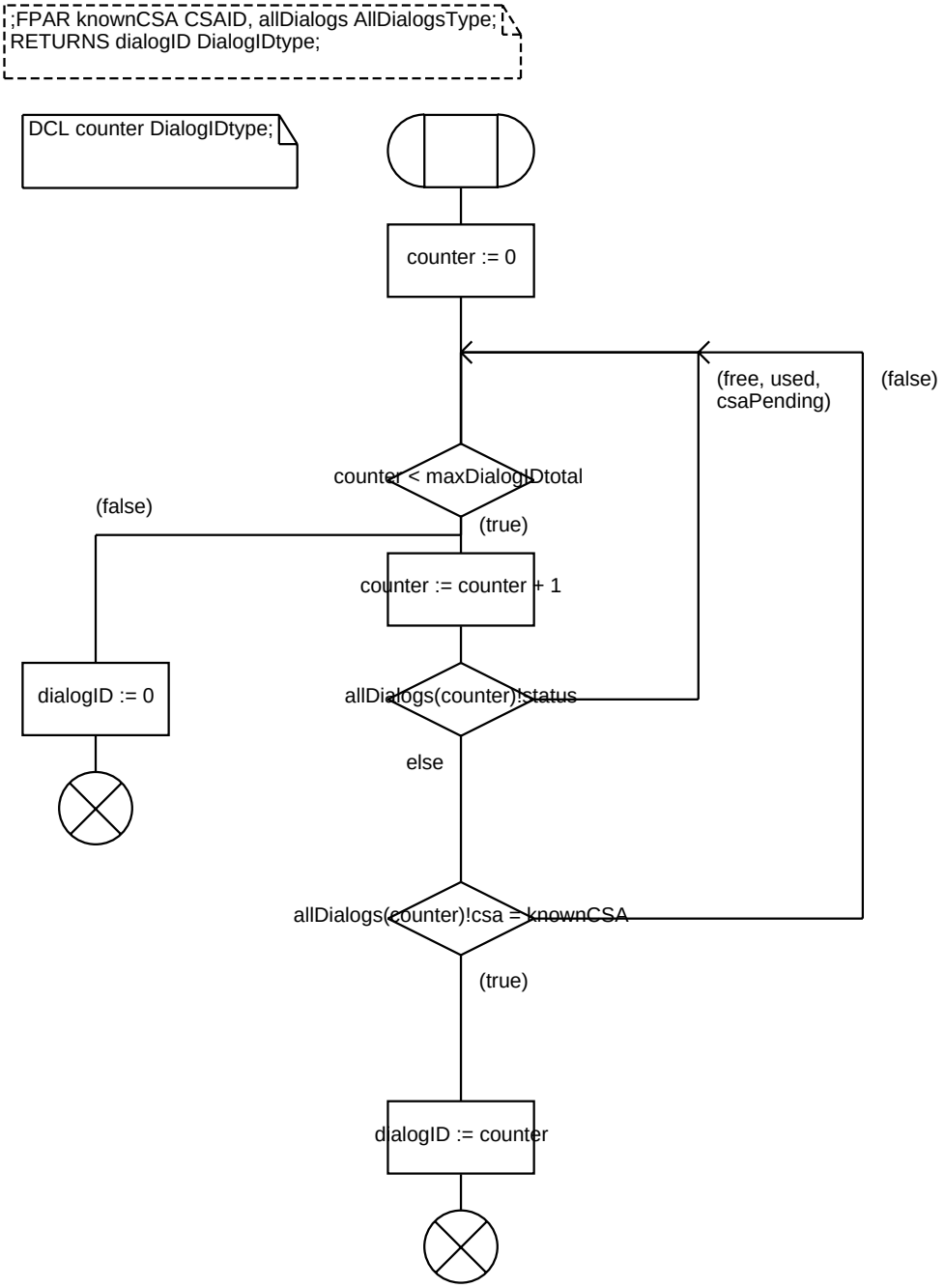
;FPAR diaStatus DialogStatusType, csaNew CSAID, diaHandler PId;
RETURNS nDialog DialogType;



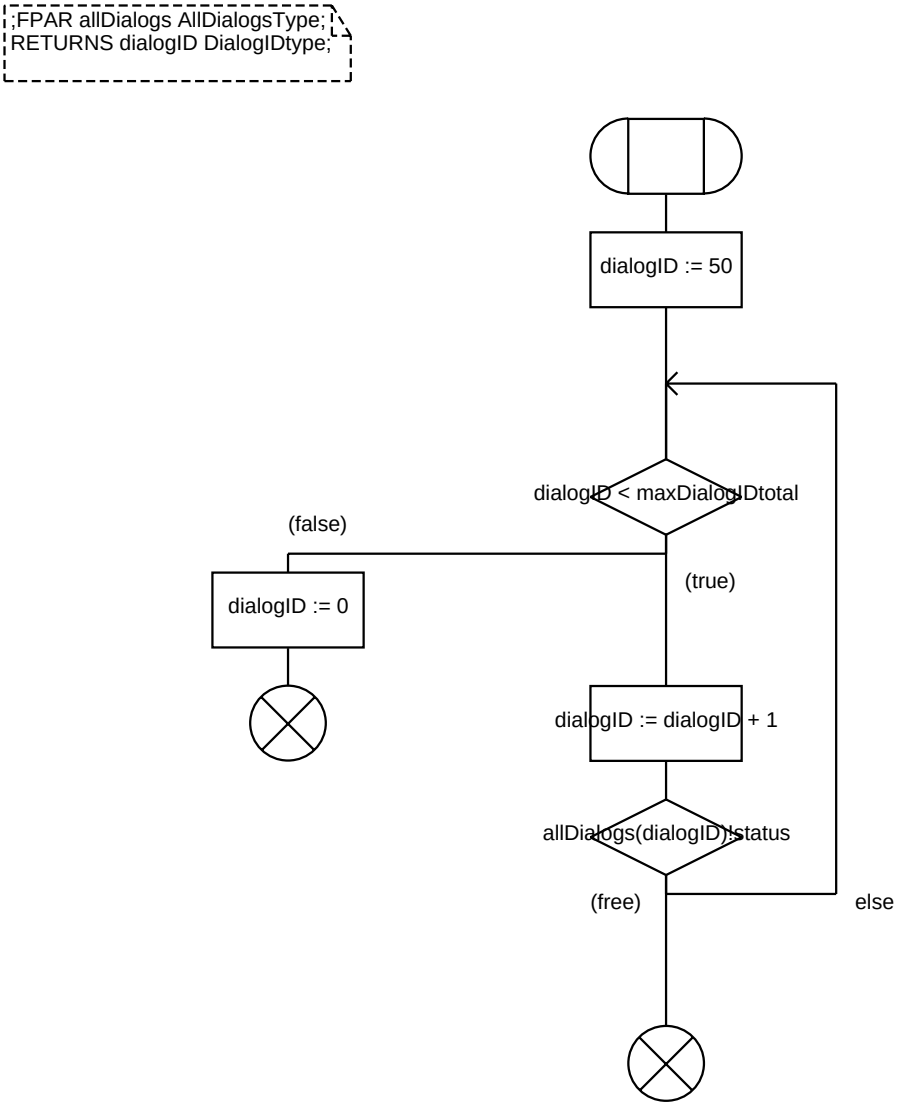
Operator initDialogs 1(1)

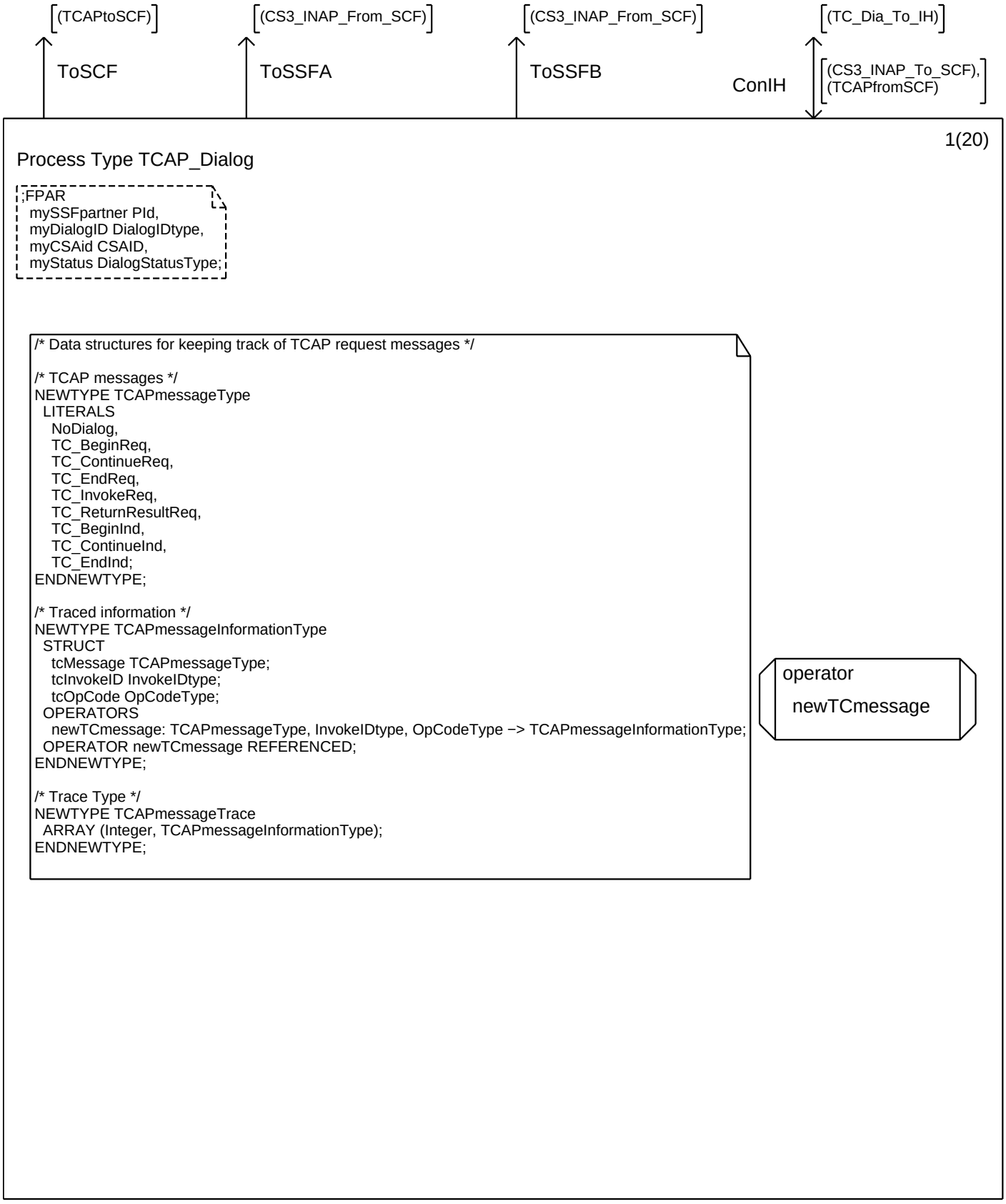


Operator getDialogID 1(1)



Operator nextFreeDialogID 1(1)




(TCAPtoSCF)
(CS3_INAP_From_SCF)
(CS3_INAP_From_SCF)
(TC_Dia_To_IH)
ToSCF
ToSSFA
ToSSFB
ConIH
(CS3_INAP_To_SCF),
(TCAPfromSCF)
1(20)
Process Type TCAP_Dialog
;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_ReturnResultReq,
TC_BeginInd,
TC_ContinueInd,
TC_EndInd;
ENDNEWTYPE;
/* Traced information */
NEWTYPE TCAPmessageInformationType
STRUCT
tcMessage TCAPmessageType;
tcInvokeID InvokeIDtype;
tcOpCode OpCodeType;
OPERATORS
newTCmessage: TCAPmessageType, InvokeIDtype, OpCodeType −> TCAPmessageInformationType;
OPERATOR newTCmessage REFERENCED;
ENDNEWTYPE;
/* Trace Type */
NEWTYPE TCAPmessageTrace
ARRAY (Integer, TCAPmessageInformationType);
ENDNEWTYPE;
operator
newTCmessage

2(20)

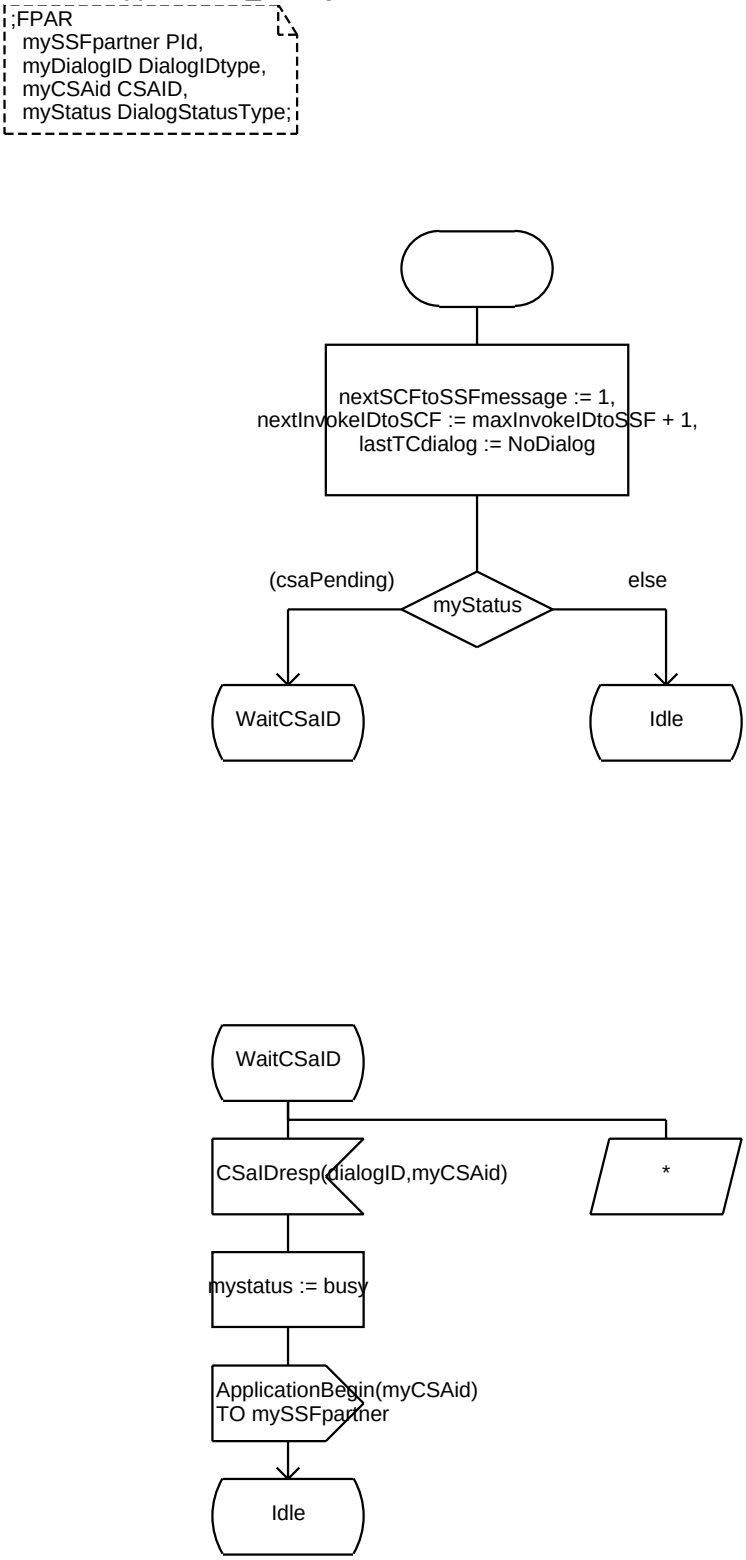
Process Type TCAP_Dialog

```
;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,
      destAd TCoriginType;
DCL termination TCAPterminationType;
DCL cSAid CSAID;
DCL compEnd Boolean := false;

DCL arg ArgType; /* refers to ASN.1 definition INCS3BundleArg */
DCL errArg ErrorArg;
```

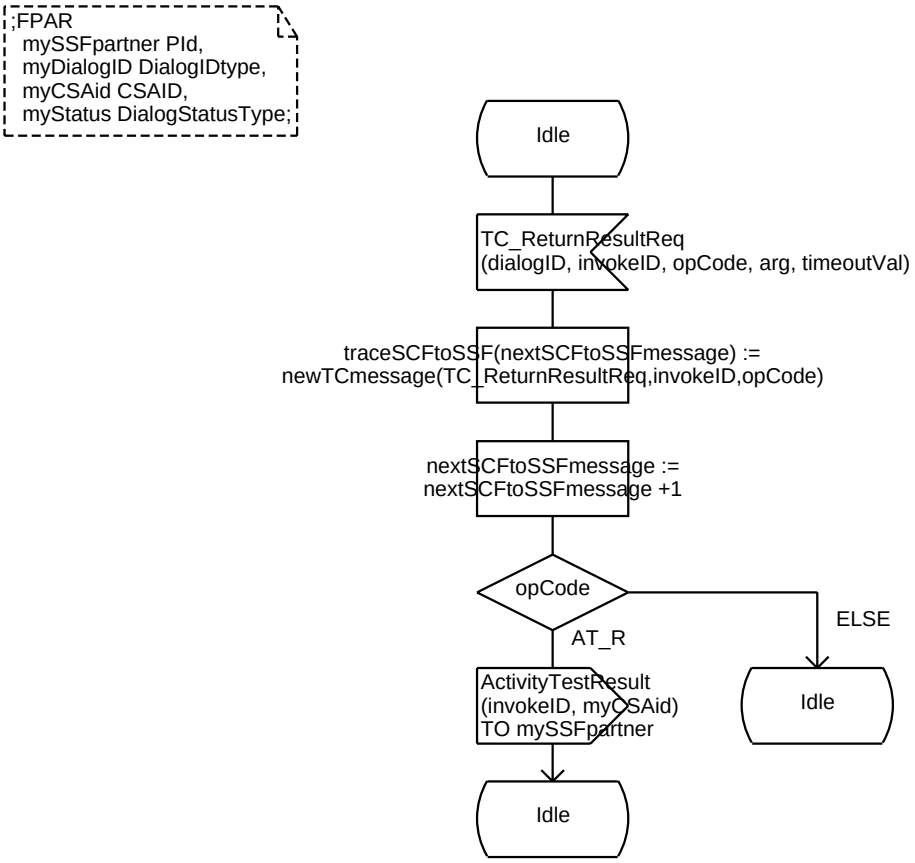
3(20)

Process Type TCAP_Dialog



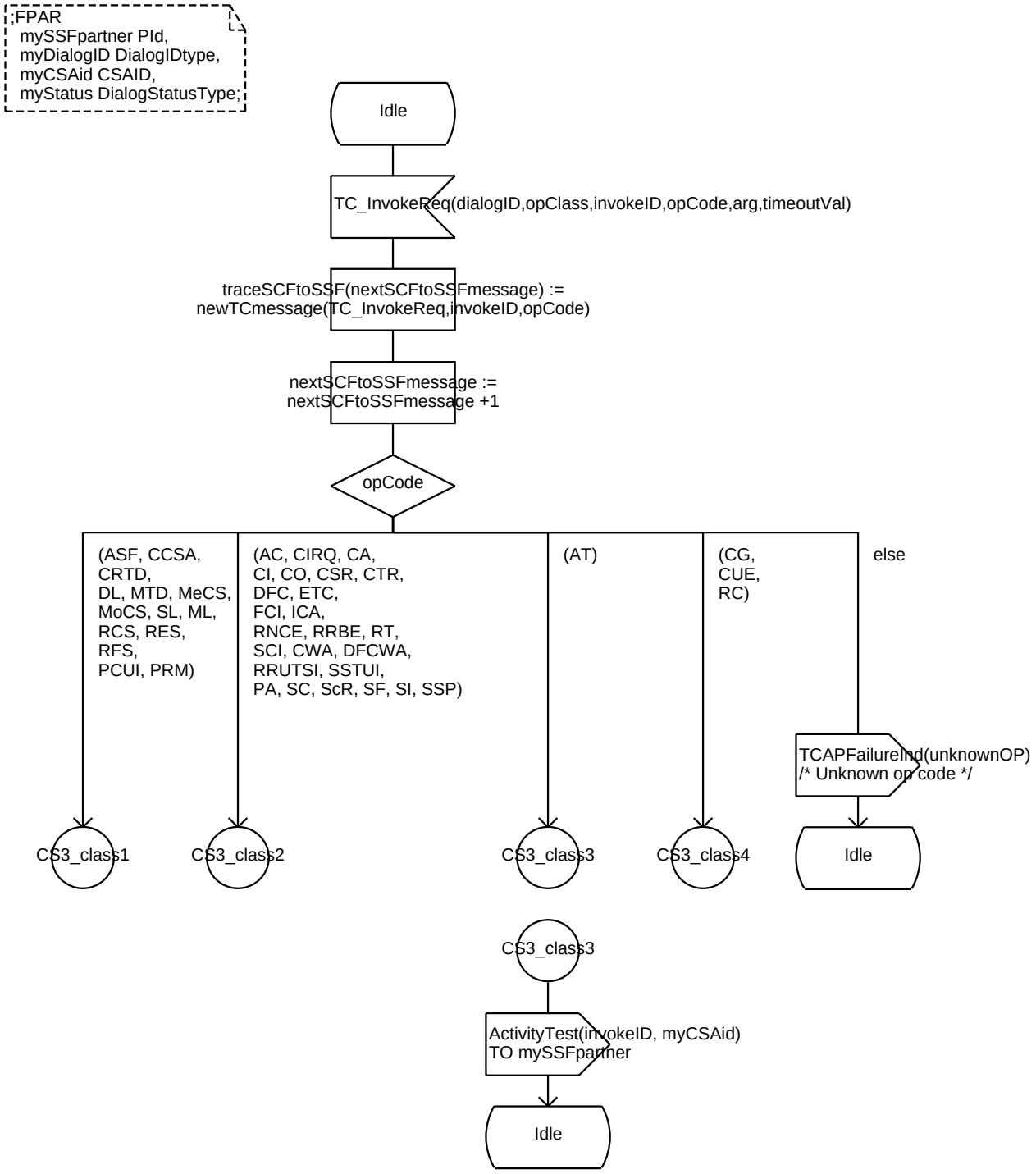
4(20)

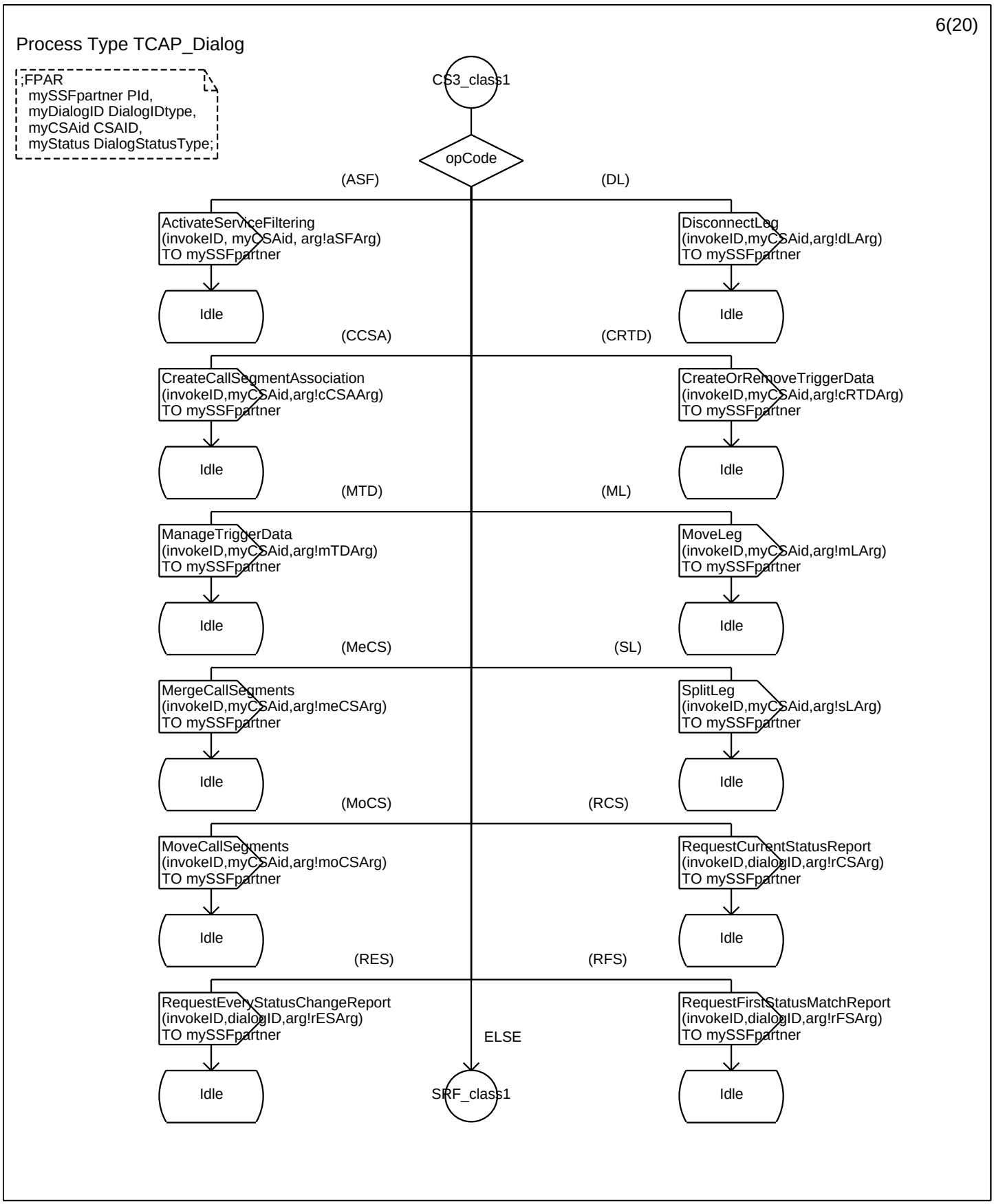
Process Type TCAP_Dialog

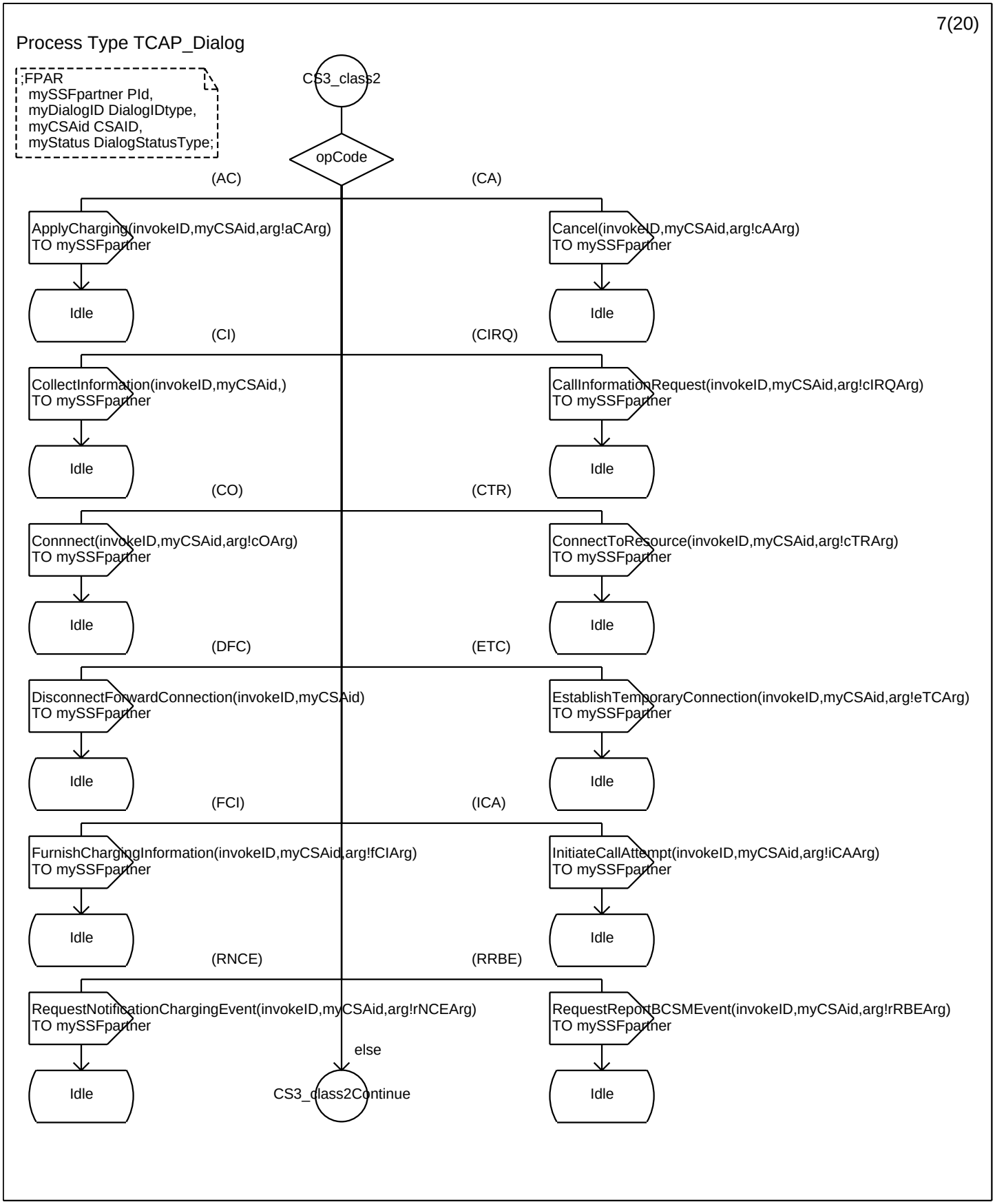


5(20)

Process Type TCAP_Dialog

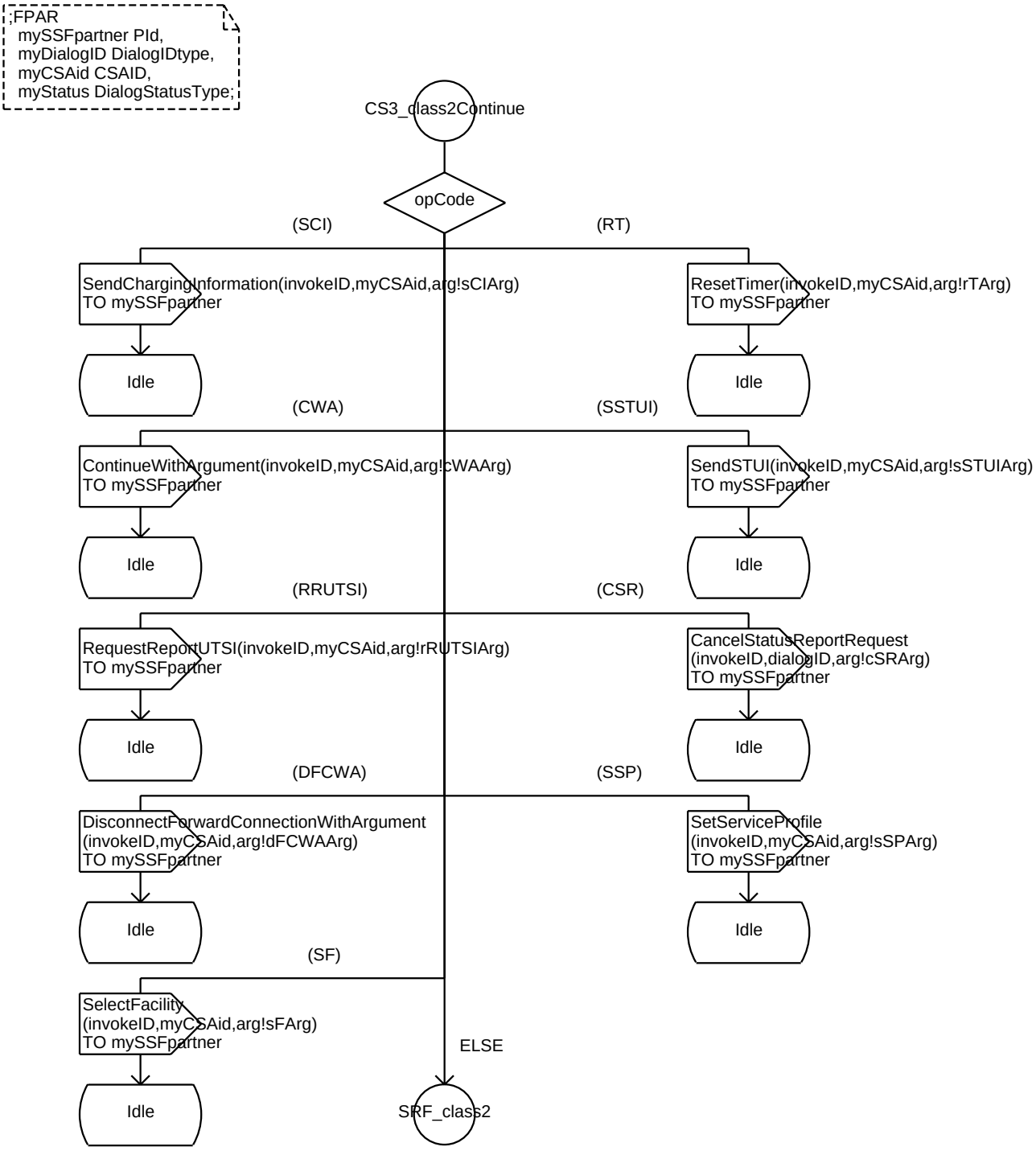



6(20)
Process Type TCAP_Dialog
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
CS3_class1
opCode
(ASF)
(DL)
ActivateServiceFiltering
(invokeID, myCSAid, arg!aSFArg)
TO mySSFpartner
DisconnectLeg
(invokeID,myCSAid,arg!dLArg)
TO mySSFpartner
Idle
Idle
(CCSA)
(CRTD)
CreateCallSegmentAssociation
(invokeID,myCSAid,arg!cCSAArg)
TO mySSFpartner
CreateOrRemoveTriggerData
(invokeID,myCSAid,arg!cRTDArg)
TO mySSFpartner
Idle
Idle
(MTD)
(ML)
ManageTriggerData
(invokeID,myCSAid,arg!mTDArg)
TO mySSFpartner
MoveLeg
(invokeID,myCSAid,arg!mLArg)
TO mySSFpartner
Idle
Idle
(MeCS)
(SL)
MergeCallSegments
(invokeID,myCSAid,arg!meCSArg)
TO mySSFpartner
SplitLeg
(invokeID,myCSAid,arg!sLArg)
TO mySSFpartner
Idle
Idle
(MoCS)
(RCS)
MoveCallSegments
(invokeID,myCSAid,arg!moCSArg)
TO mySSFpartner
RequestCurrentStatusReport
(invokeID,dialogID,arg!rCSArg)
TO mySSFpartner
Idle
Idle
(RES)
(RFS)
RequestEveryStatusChangeReport
(invokeID,dialogID,arg!rESArg)
TO mySSFpartner
RequestFirstStatusMatchReport
(invokeID,dialogID,arg!rFSArg)
TO mySSFpartner
ELSE
Idle
SRF_class1
Idle


7(20)
Process Type TCAP_Dialog
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
CS3_class2
opCode
(AC)
(CA)
ApplyCharging(invokeID,myCSAid,arg!aCArg)
TO mySSFpartner
Cancel(invokeID,myCSAid,arg!cAArg)
TO mySSFpartner
Idle
Idle
(CI)
(CIRQ)
CollectInformation(invokeID,myCSAid,)
TO mySSFpartner
CallInformationRequest(invokeID,myCSAid,arg!cIRQArg)
TO mySSFpartner
Idle
Idle
(CO)
(CTR)
Connnect(invokeID,myCSAid,arg!cOArg)
TO mySSFpartner
ConnectToResource(invokeID,myCSAid,arg!cTRArg)
TO mySSFpartner
Idle
Idle
(DFC)
(ETC)
DisconnectForwardConnection(invokeID,myCSAid)
TO mySSFpartner
EstablishTemporaryConnection(invokeID,myCSAid,arg!eTCArg)
TO mySSFpartner
Idle
Idle
(FCI)
(ICA)
FurnishChargingInformation(invokeID,myCSAid,arg!fCIArg)
TO mySSFpartner
InitiateCallAttempt(invokeID,myCSAid,arg!iCAArg)
TO mySSFpartner
Idle
Idle
(RNCE)
(RRBE)
RequestNotificationChargingEvent(invokeID,myCSAid,arg!rNCEArg)
TO mySSFpartner
RequestReportBCSMEvent(invokeID,myCSAid,arg!rRBEArg)
TO mySSFpartner
else
Idle
CS3_class2Continue
Idle

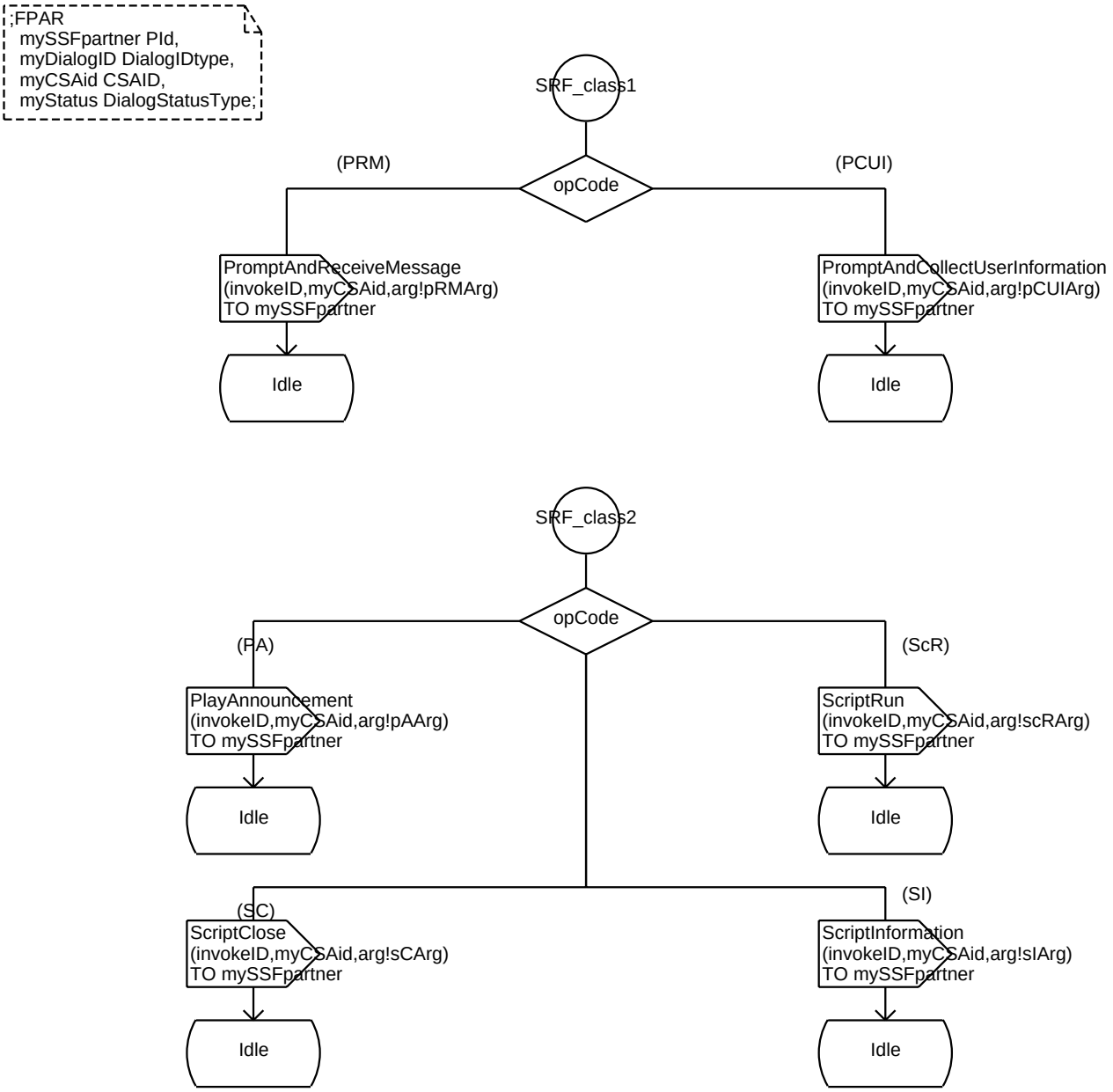
8(20)

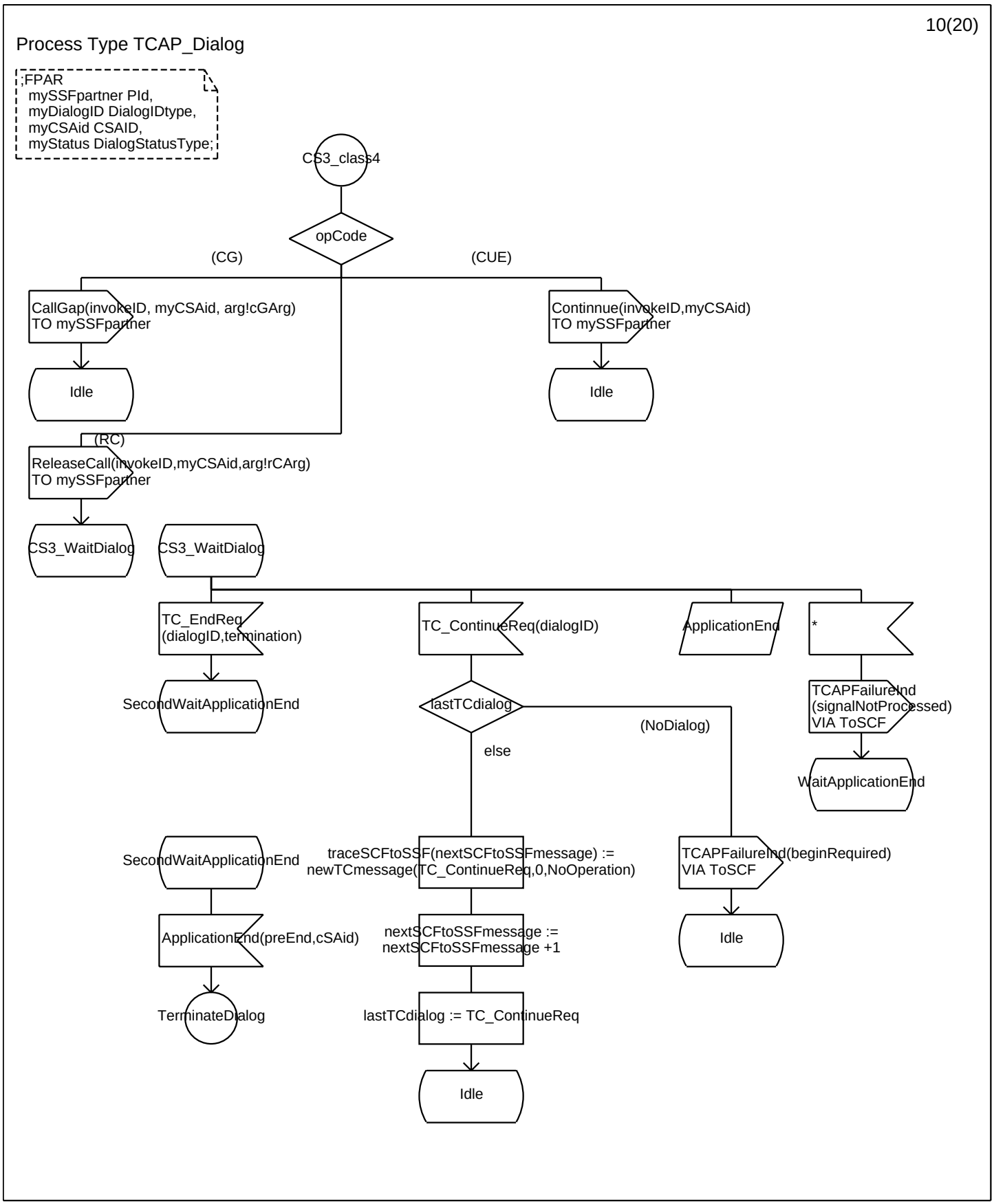
Process Type TCAP_Dialog



9(20)

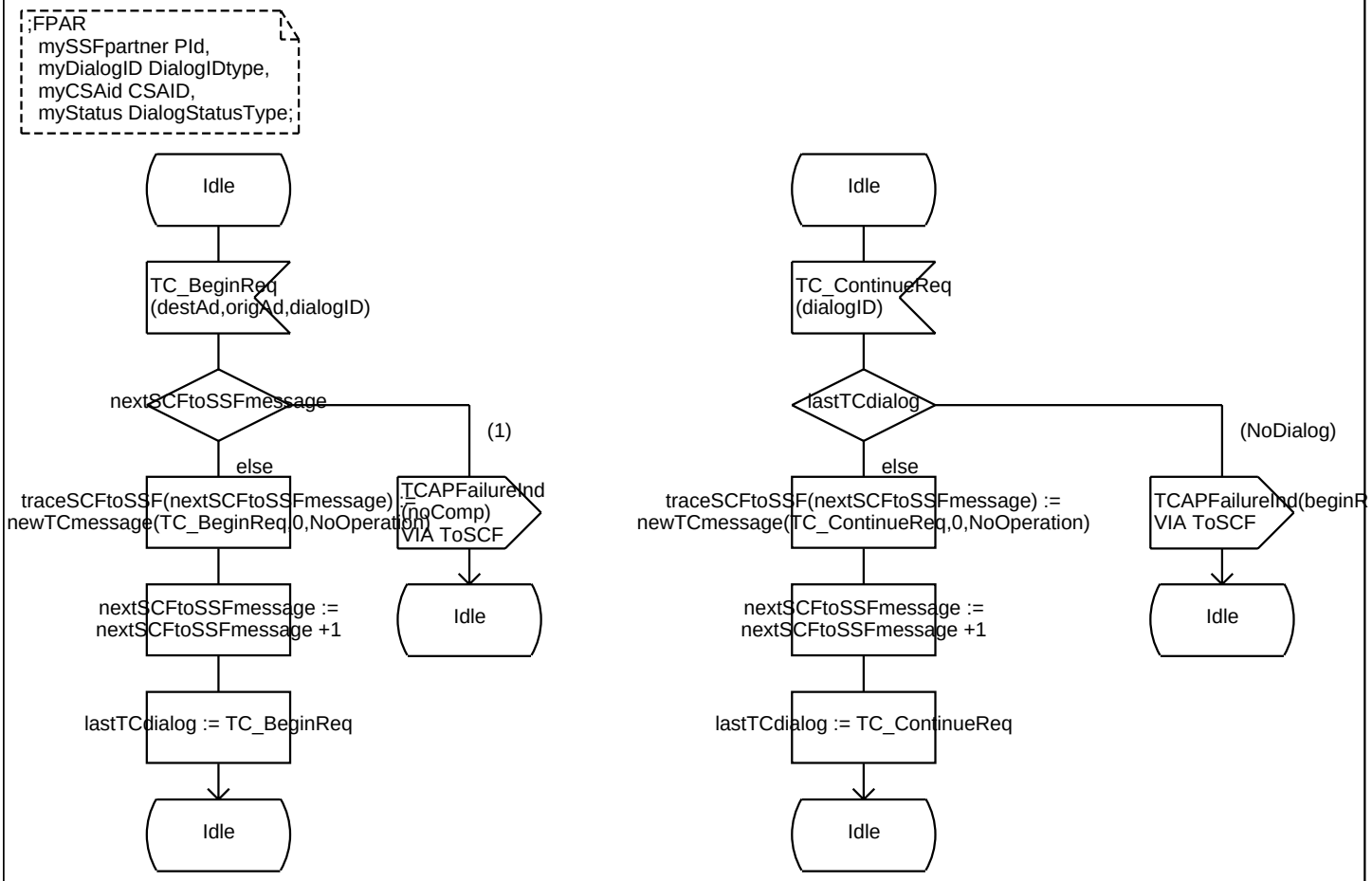
Process Type TCAP_Dialog




10(20)
Process Type TCAP_Dialog
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
CS3_class4
opCode
(CG)
(CUE)
CallGap(invokeID, myCSAid, arg!cGArg)
TO mySSFpartner
Continnue(invokeID,myCSAid)
TO mySSFpartner
Idle
Idle
(RC)
ReleaseCall(invokeID,myCSAid,arg!rCArg)
TO mySSFpartner
CS3_WaitDialog
CS3_WaitDialog
TC_EndReq
(dialogID,termination)
TC_ContinueReq(dialogID)
ApplicationEnd
*
SecondWaitApplicationEnd
lastTCdialog
(NoDialog)
else
TCAPFailureInd
(signalNotProcessed)
VIA ToSCF
WaitApplicationEnd
SecondWaitApplicationEnd
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_ContinueReq,0,NoOperation)
TCAPFailureInd(beginRequired)
VIA ToSCF
ApplicationEnd(preEnd,cSAid)
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
Idle
TerminateDialog
lastTCdialog := TC_ContinueReq
Idle

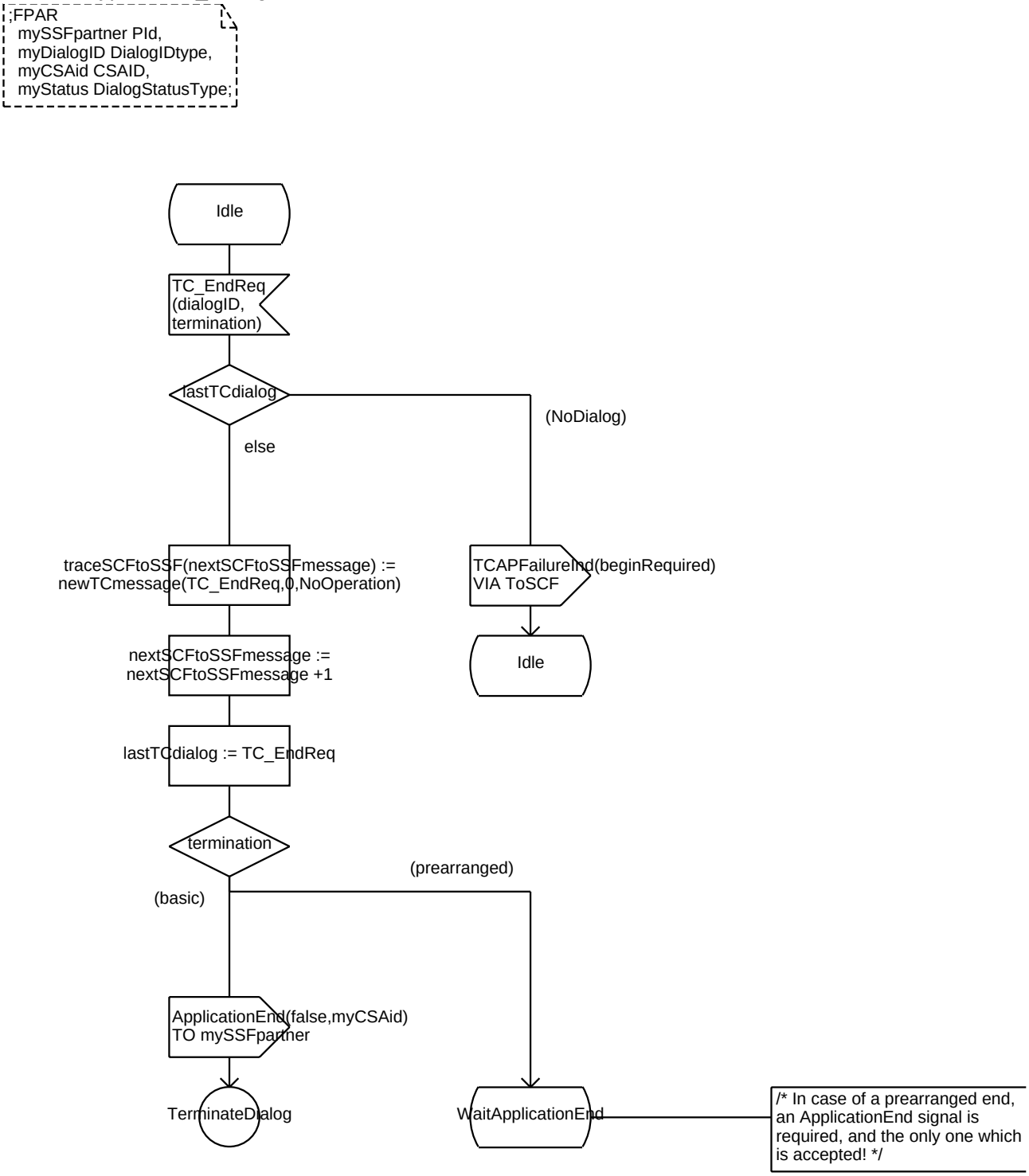
11(20)

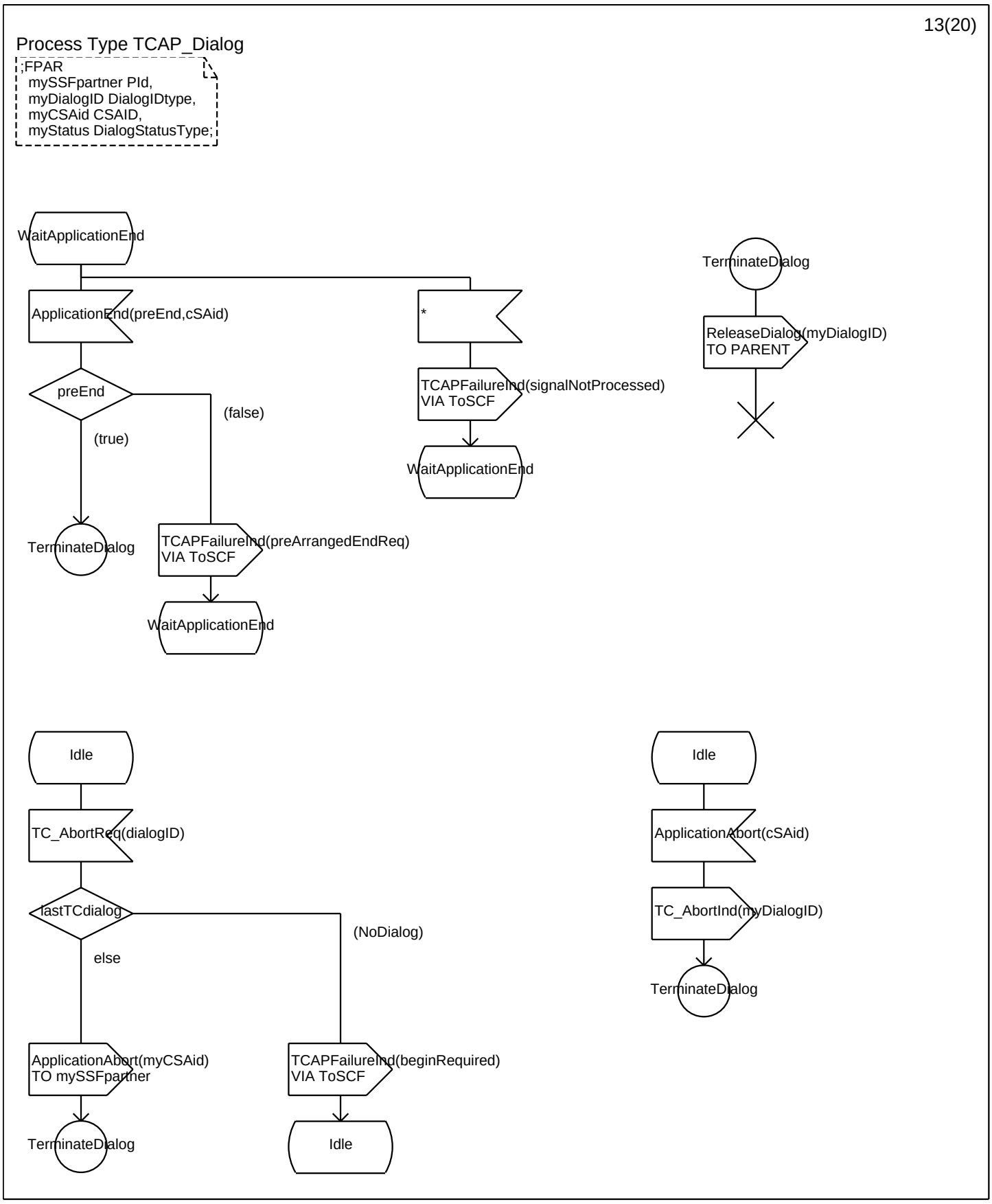
Process Type TCAP_Dialog



12(20)

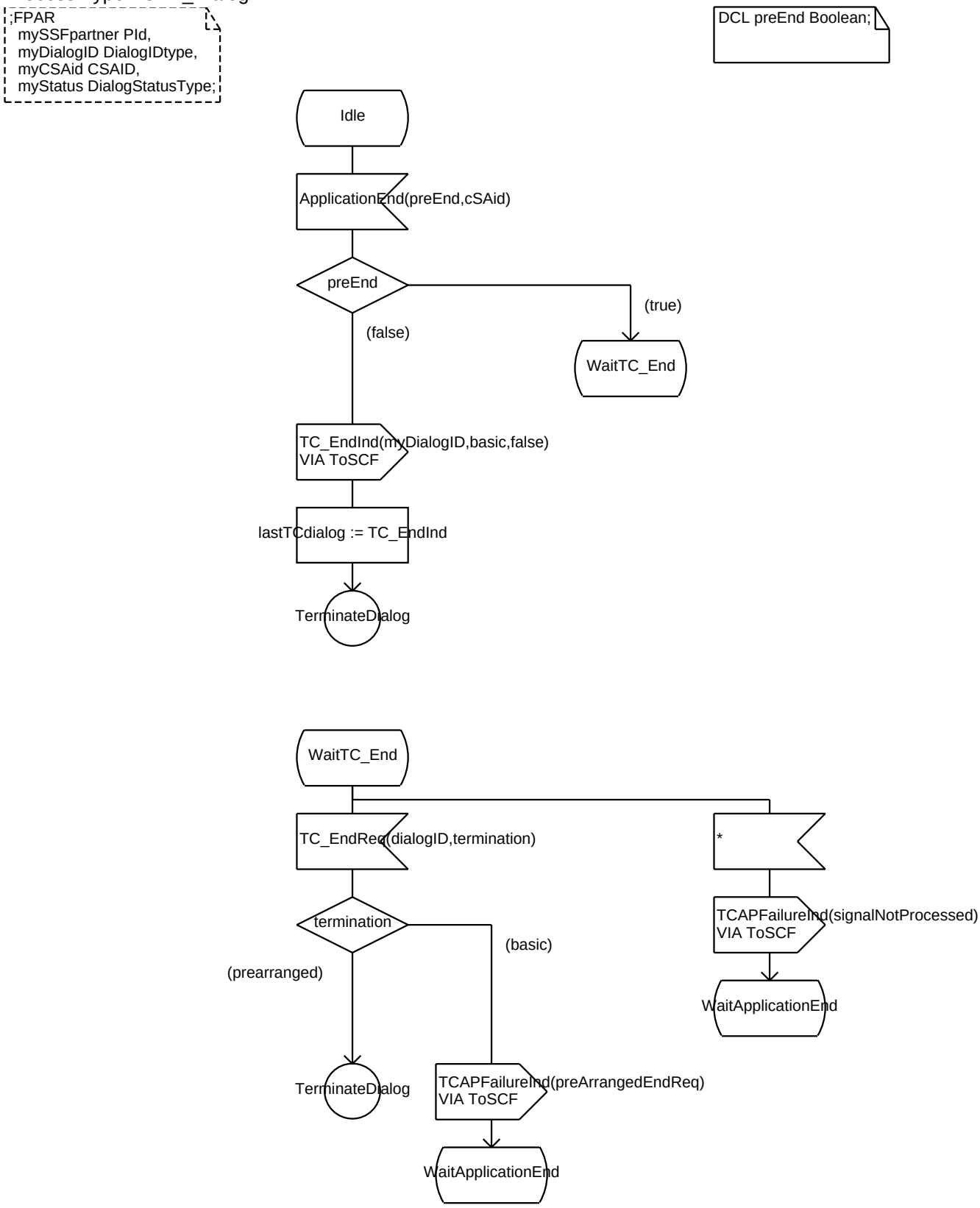
Process Type TCAP_Dialog




13(20)
Process Type TCAP_Dialog
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
WaitApplicationEnd
ApplicationEnd(preEnd,cSAid)
preEnd
(true)
(false)
TerminateDialog
TCAPFailureInd(preArrangedEndReq)
VIA ToSCF
WaitApplicationEnd
*
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
WaitApplicationEnd
TerminateDialog
ReleaseDialog(myDialogID)
TO PARENT
Idle
TC_AbortReq(dialogID)
lastTCdialog
else
(NoDialog)
ApplicationAbort(myCSAid)
TO mySSFpartner
TerminateDialog
TCAPFailureInd(beginRequired)
VIA ToSCF
Idle
Idle
ApplicationAbort(cSAid)
TC_AbortInd(myDialogID)
TerminateDialog

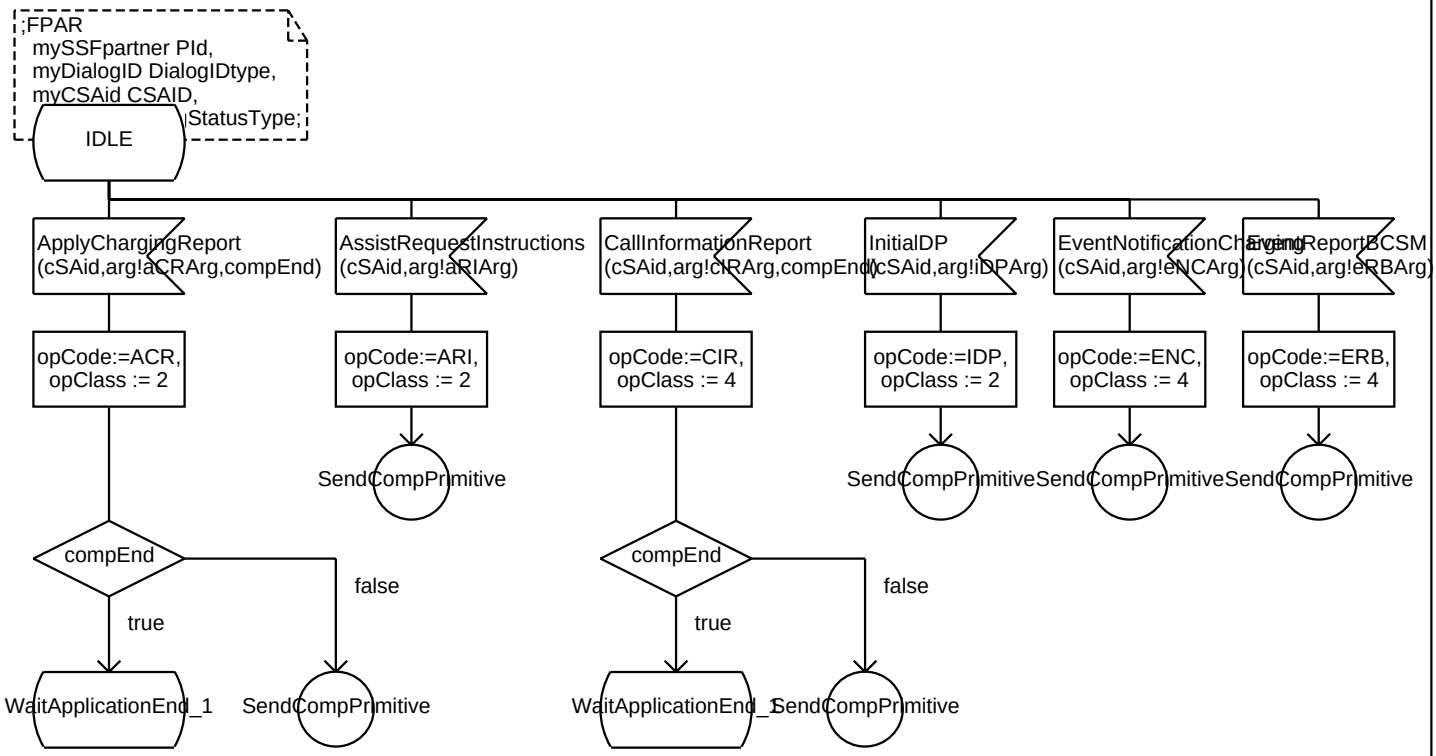
14(20)

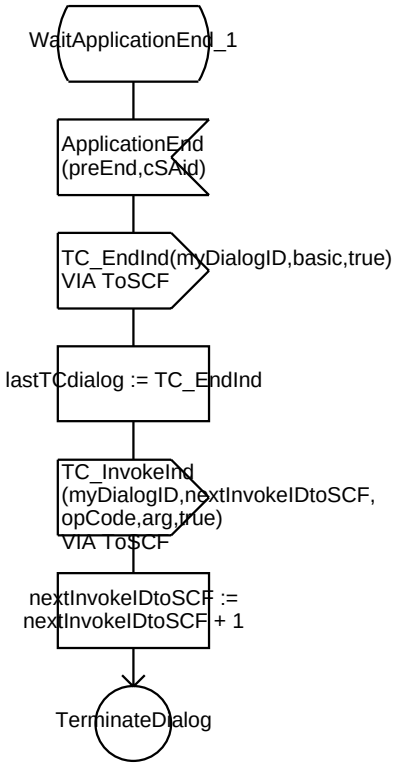
Process Type TCAP_Dialog



15(20)

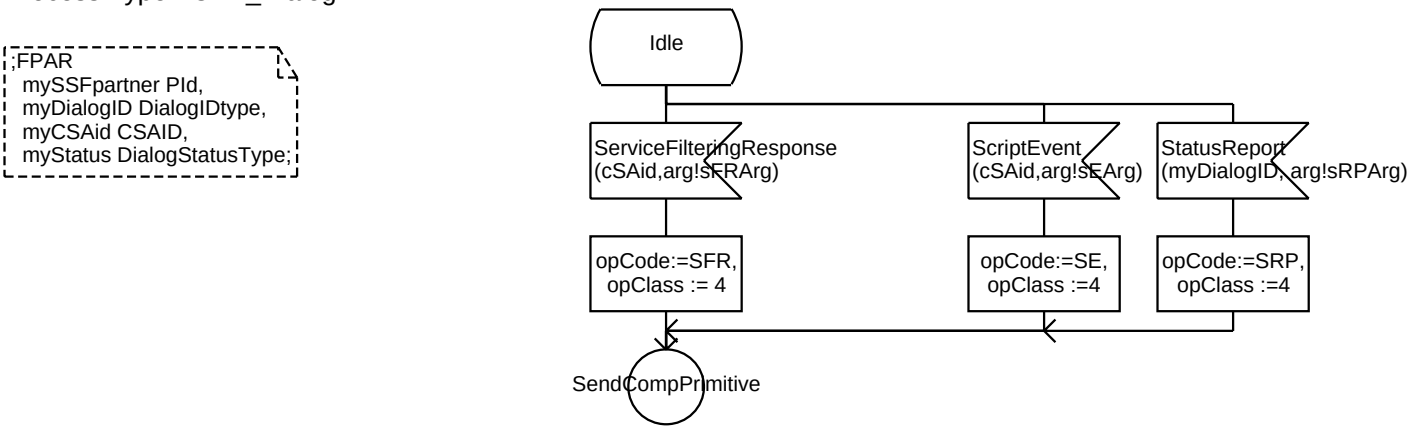
Process Type TCAP_Dialog

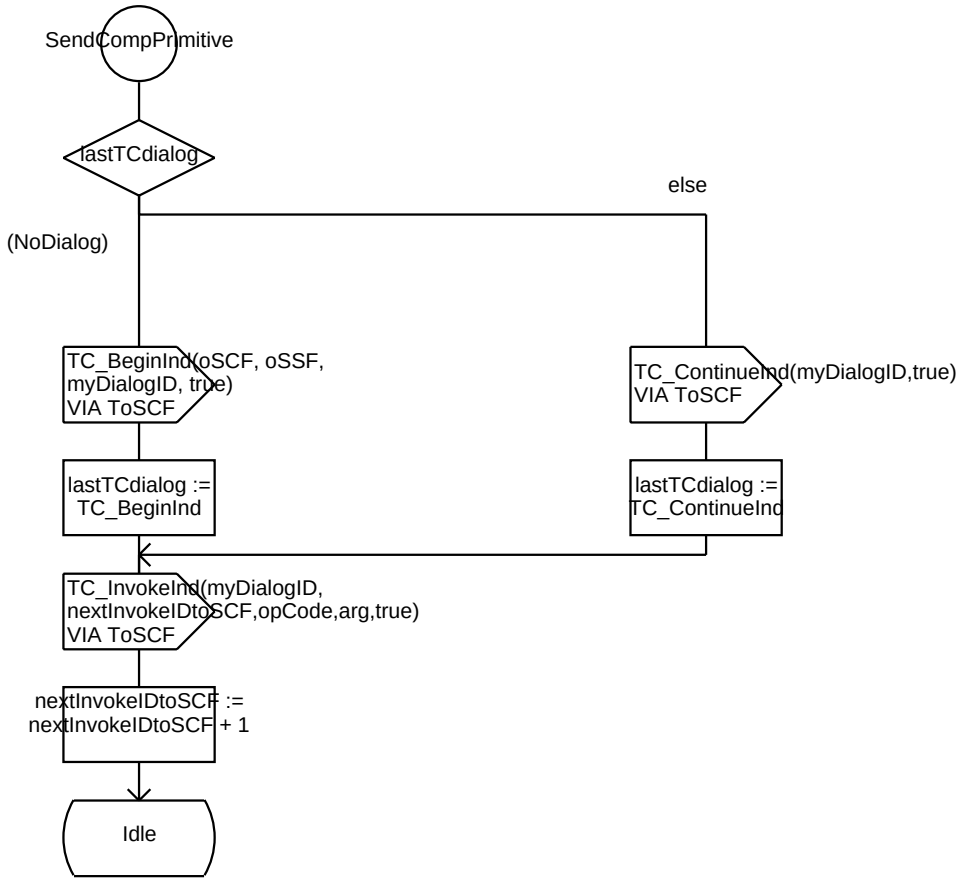




16(20)

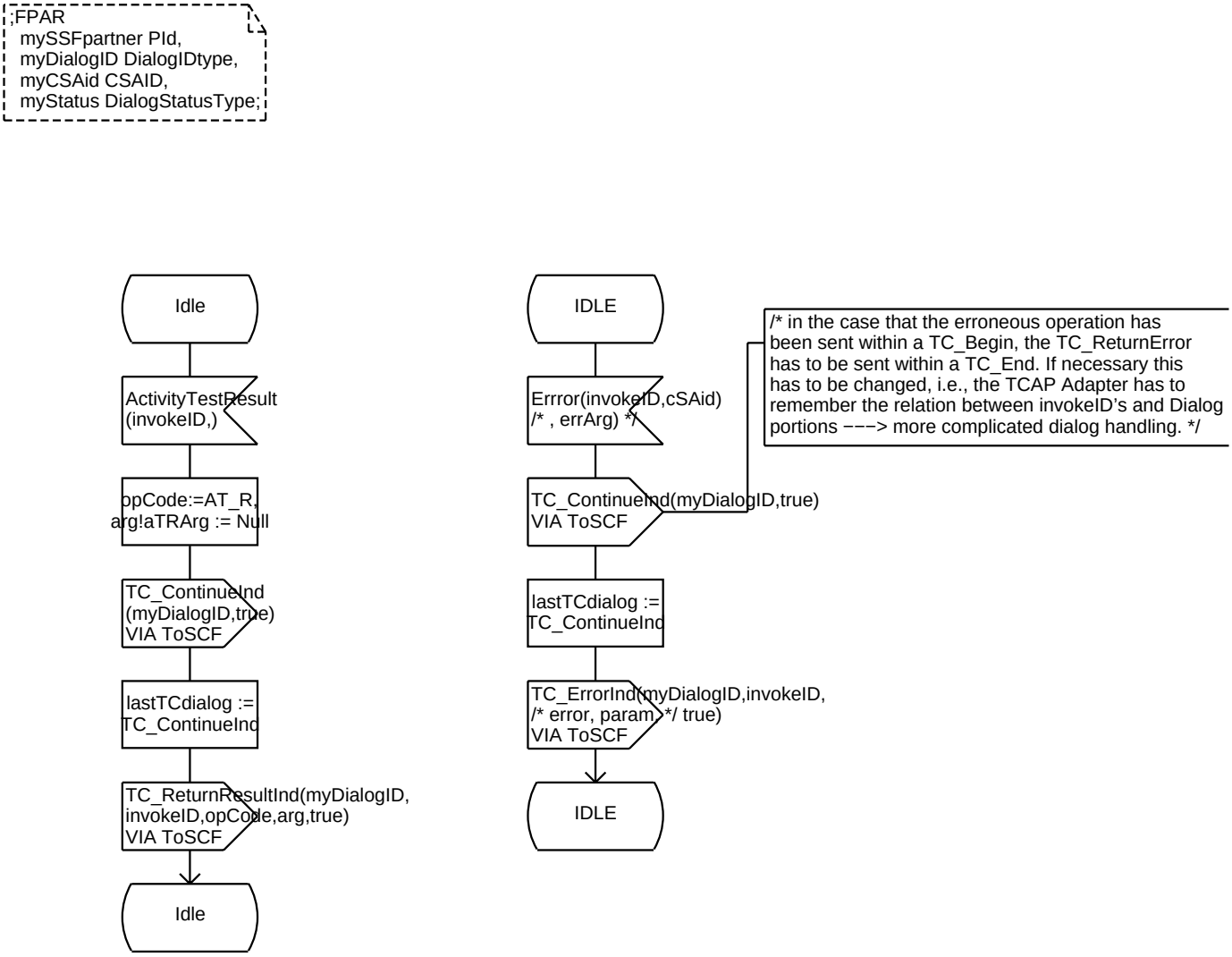
Process Type TCAP_Dialog





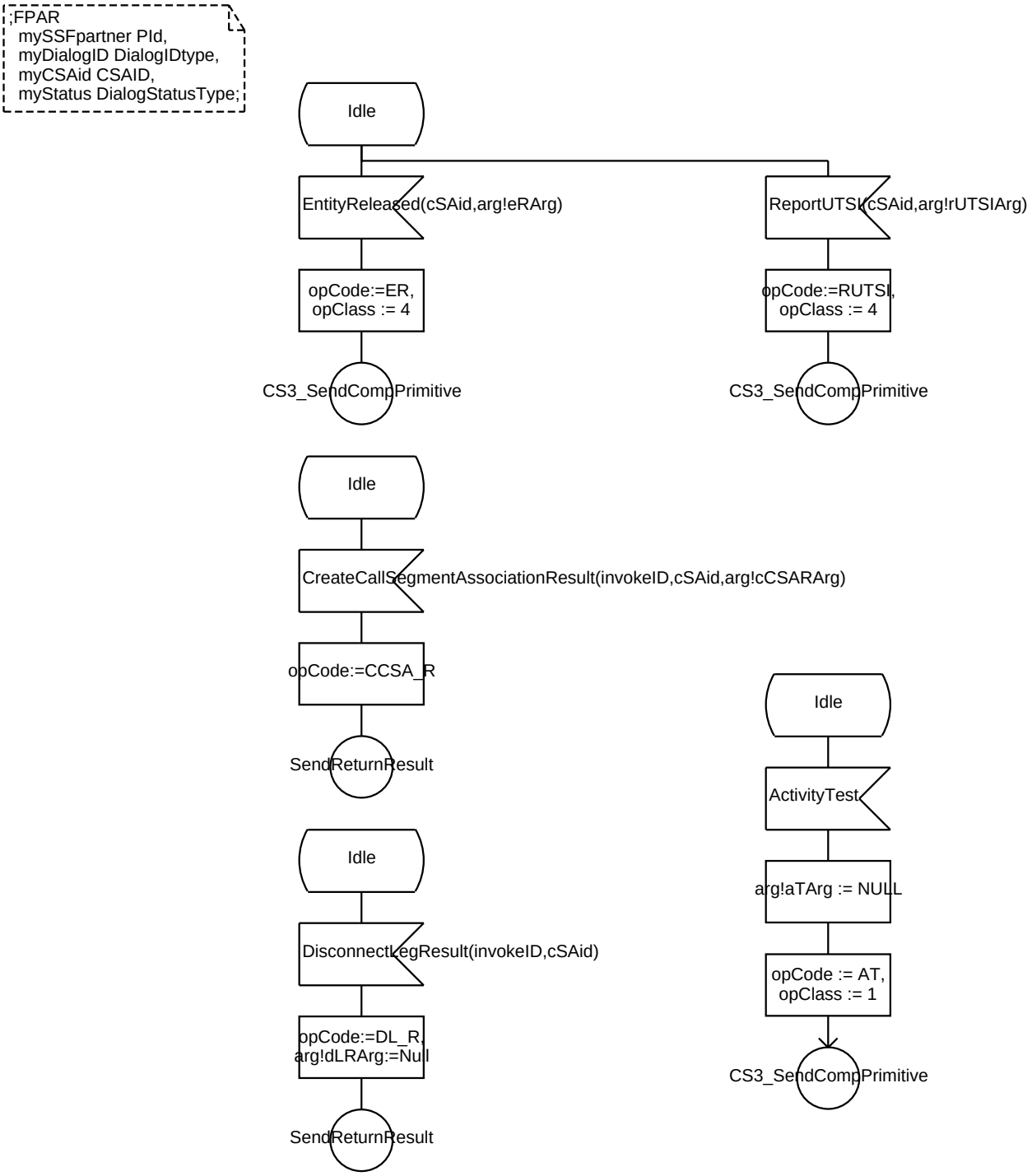
17(20)

Process Type TCAP_Dialog



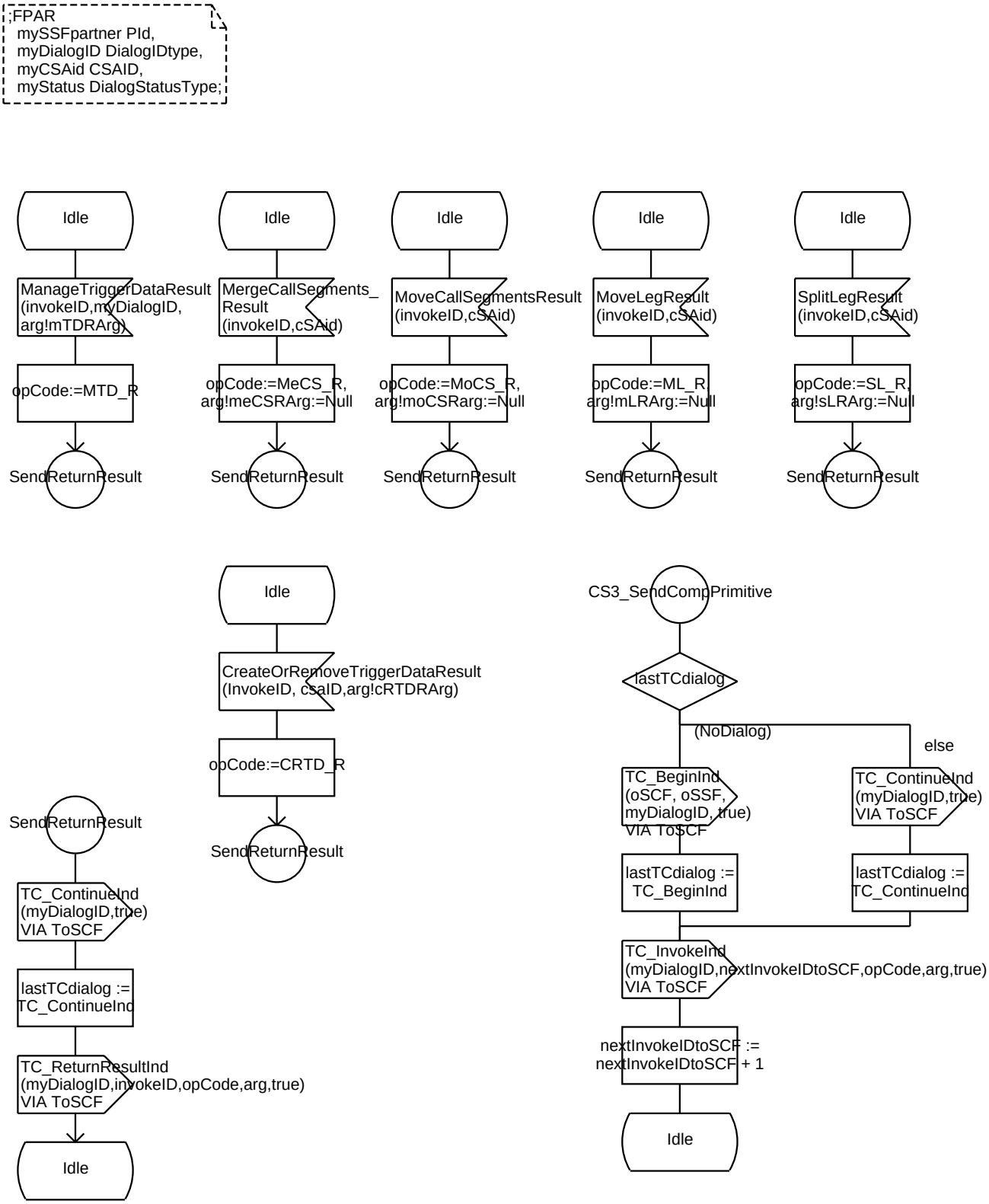
18(20)

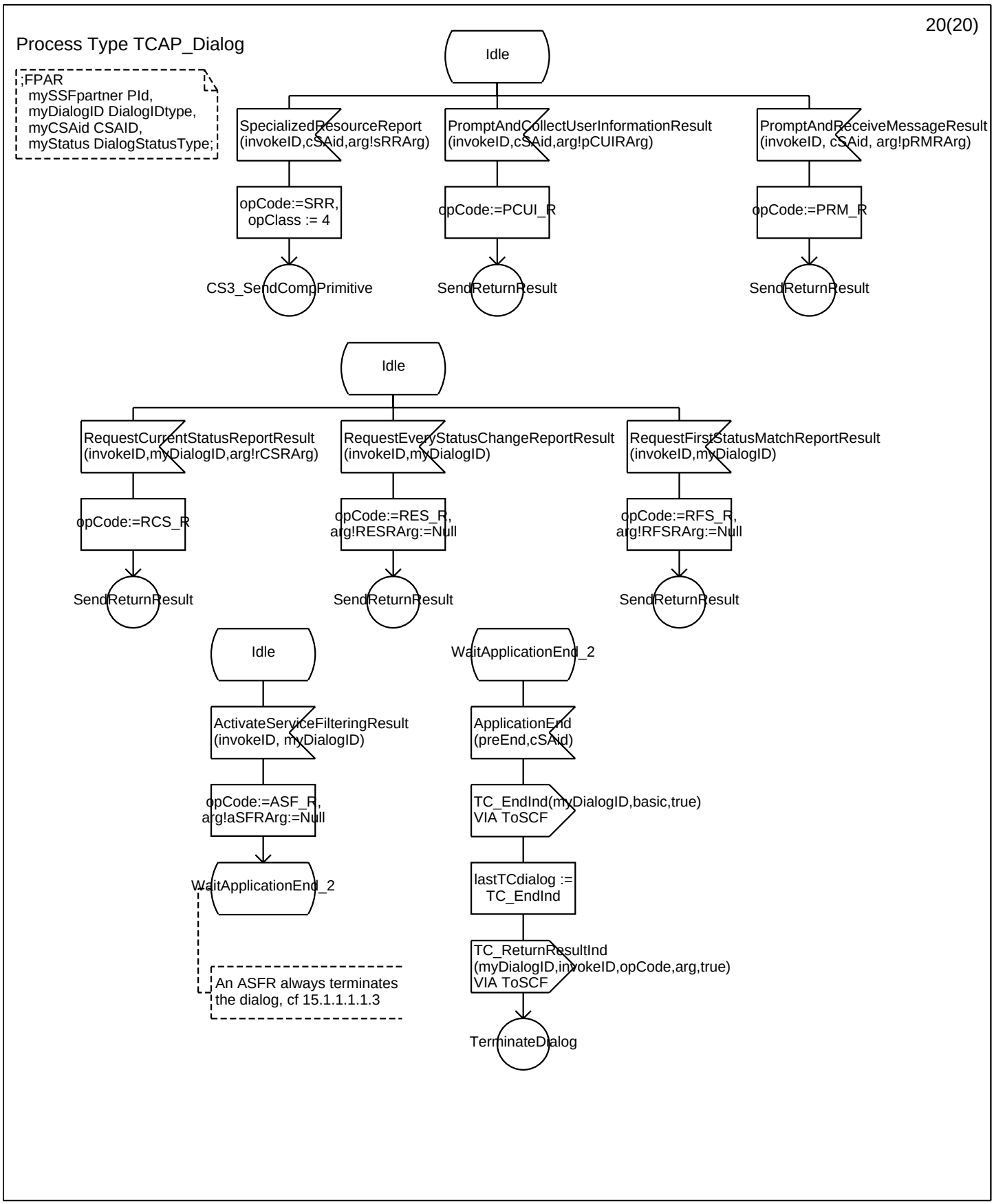
Process Type TCAP_Dialog



19(20)

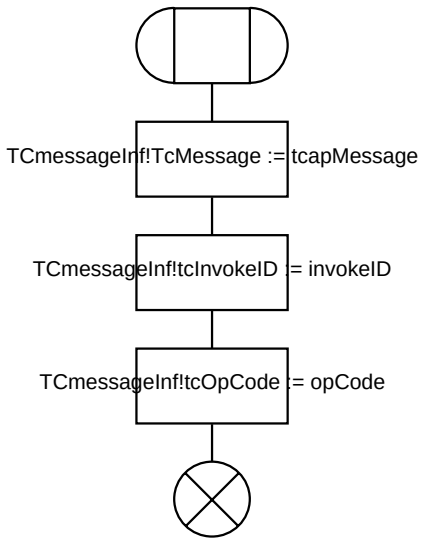
Process Type TCAP_Dialog




20(20)
Process Type TCAP_Dialog
;FPAR
mySSFpartner PId,
myDialogID DialogIDtype,
myCSAid CSAID,
myStatus DialogStatusType;
Idle
SpecializedResourceReport
(invokeID,cSAid,arg!sRRArg)
PromptAndCollectUserInformationResult
(invokeID,cSAid,arg!pCUIRArg)
PromptAndReceiveMessageResult
(invokeID, cSAid, arg!pRMRArg)
opCode:=SRR,
opClass := 4
opCode:=PCUI_R
opCode:=PRM_R
CS3_SendCompPrimitive
SendReturnResult
SendReturnResult
Idle
RequestCurrentStatusReportResult
(invokeID,myDialogID,arg!rCSRArg)
RequestEveryStatusChangeReportResult
(invokeID,myDialogID)
RequestFirstStatusMatchReportResult
(invokeID,myDialogID)
opCode:=RCS_R
opCode:=RES_R,
arg!RESRArg:=Null
opCode:=RFS_R,
arg!RFSRArg:=Null
SendReturnResult
SendReturnResult
SendReturnResult
Idle
ActivateServiceFilteringResult
(invokeID, myDialogID)
opCode:=ASF_R,
arg!aSFRArg:=Null
WaitApplicationEnd_2
An ASFR always terminates
the dialog, cf 15.1.1.1.1.3
WaitApplicationEnd_2
ApplicationEnd
(preEnd,cSAid)
TC_EndInd(myDialogID,basic,true)
VIA ToSCF
lastTCdialog :=
TC_EndInd
TC_ReturnResultInd
(myDialogID,invokeID,opCode,arg,true)
VIA ToSCF
TerminateDialog

Operator newTCmessage 1(1)

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



```
USE INCS3Internals;
USE INCS3datatypes;
USE INCS3opsargs;
USE INCS3BundleArg;
```

Package CS3_TCAP 1(3)

```
/* This package contains
 * definitions of TCAP signals
 * and data types as used on the
 * external interface of the system.
 */
```

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

/* Needed for the initialization of the TCAP Simulator */
NEWTYPE SSME_CroleType
  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;
```

Package CS3_TCAP 2(3)

```
/* Data type definitions used by the TCAP Simulator (Continued)*/
```

```
/* Reasons for a TCAP failure */
NEWTYPE TCAPfailReasonType
  LITERALS
    wrongDiaID, /* incorrect dialog ID: incorrect range or no dialog for ID present/available */
    wrongInvokeID, /* incorrect invoke ID */
    wrongCSaID, /* incorrect CSA ID */
    beginRequired, /* TC_BeginReq is required */
    noComp, /* TC_BeginReq without components */
    unknownOP, /* Unknown operation code (within TC Invoke) */
    noDiaIDavail, /* No dialogID available, 'maxDialogIDtoSCF' Diallogs are established */
    cSAnotAvail, /* the provide CSAid is not available */
    unexpSignal, /* the TCAP SSME_C 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,
  /* FROM SCF */
  /* CLASS 1 */ ASF, CCSA, CRTD, DL, MeCS, MoCS, ML, MTD, RCS, RES, RFS, SL, PCUI, PRM,
  /* CLASS 2 */ AC, CIRQ, CA, CI, CO, CSR, CTR, CWA,
              DFC, DFCWA, ETC, FCI, ICA, RECON, RNCE, RRBE, RRUTSI, SSTUI, RT, SCI,
              PA,  SC, ScR, SF, SI, SSP,
  /* CLASS 3 */ AT,
  /* CLASS 4 */ CG, CUE, RC,
  /* FROM SSF */
  /* CLASS 2 */ ACR, ARI, IDP,
  /* CLASS 4 */ CIR, ENC, ERB, SE, ER, RUTSI, SFR, SRP, SRR,
  /* RETURN RESULTS FROM SSF */ ASF_R, CCSA_R, CRTD_R, DL_R, MeCS_R, MoCS_R,
              ML_R, MTD_R, RCS_R, RES_R, RFS_R, SL_R, PCUI_R, PRM_R, AT_R;
ENDNEWTYPE;

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

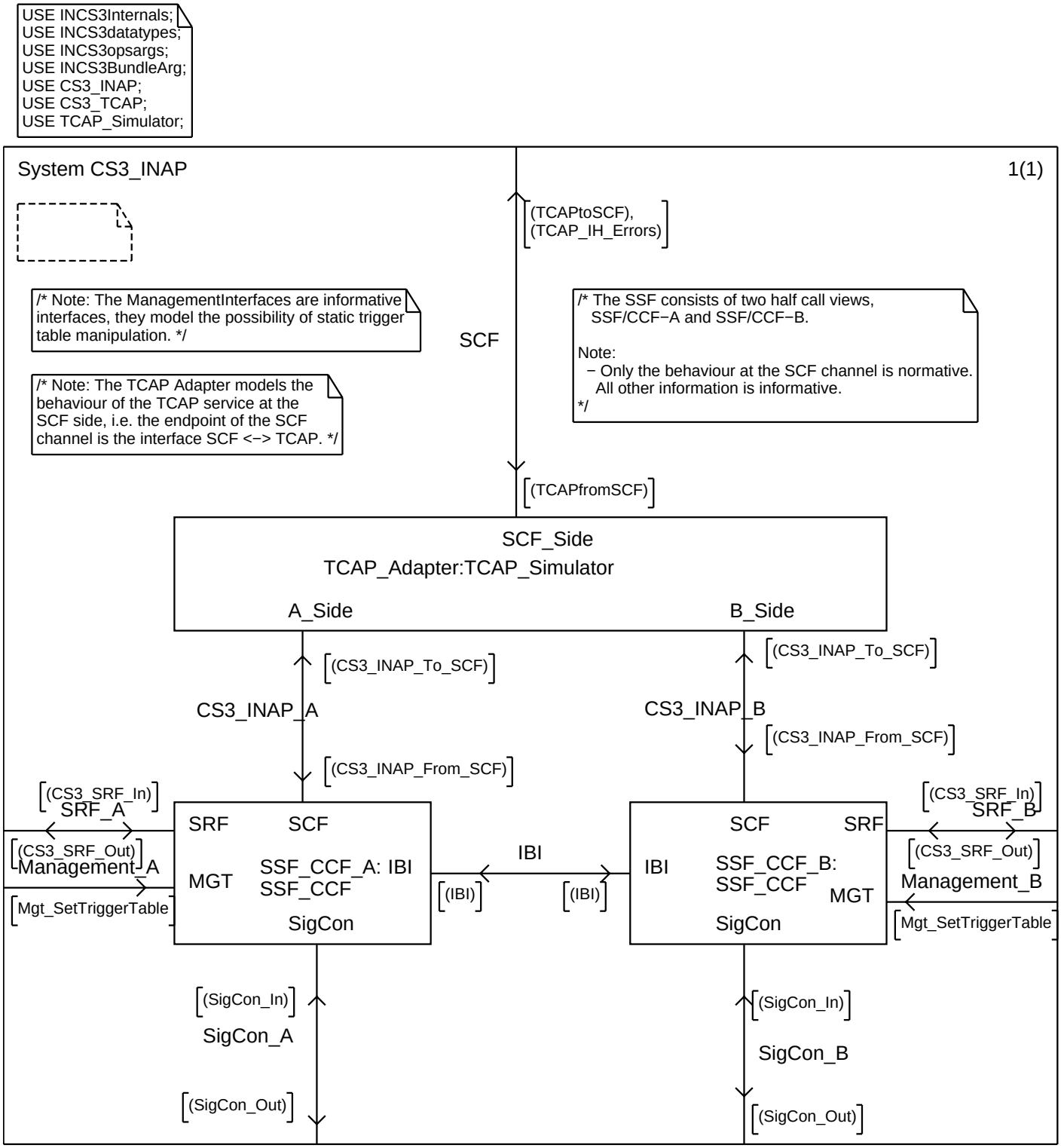
Package CS3_TCAP 3(3)

/* SIGNAL Definitions for the TCAP Adapter*/

```
/* TCAP Primitives */

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

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

```
/* SCF to TCAP */

SIGNALLIST TCAPfromSCF =
  TC_InvokeReq,
  TC_ReturnResultReq,
  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;

SIGNALLIST TCAP_IH_Errors =
  TCAPFailureInd,
  TC_AbortInd;
```


USE INCS3Internals;
USE INCS3datatypes;
USE INCS3opsargs;
USE INCS3BundleArg;
USE CS3_INAP;
USE CS3_TCAP;
USE TCAP_Simulator;
System CS3_INAP
1(1)
(TCAPtoSCF),
(TCAP_IH_Errors)
/* Note: The ManagementInterfaces are informative interfaces, they model the possibility of static trigger table manipulation. */
SCF
/* The SSF consists of two half call views, SSF/CCF−A and SSF/CCF−B.
Note:
− Only the behaviour at the SCF channel is normative. All other information is informative.
*/
/* Note: The TCAP Adapter models the behaviour of the TCAP service at the SCF side, i.e. the endpoint of the SCF channel is the interface SCF <−> TCAP. */
(TCAPfromSCF)
SCF_Side
TCAP_Adapter:TCAP_Simulator
A_Side
B_Side
(CS3_INAP_To_SCF)
CS3_INAP_A
(CS3_INAP_From_SCF)
(CS3_INAP_To_SCF)
CS3_INAP_B
(CS3_INAP_From_SCF)
(CS3_SRF_In)
SRF_A
(CS3_SRF_Out)
Management_A
Mgt_SetTriggerTable
SRF
SCF
MGT
SSF_CCF_A: IBI
SSF_CCF
SigCon
IBI
(IBI)
(IBI)
SCF
SRF
IBI
SSF_CCF_B:
SSF_CCF
MGT
SigCon
(CS3_SRF_In)
SRF_B
(CS3_SRF_Out)
Management_B
Mgt_SetTriggerTable
(SigCon_In)
SigCon_A
(SigCon_Out)
(SigCon_In)
SigCon_B
(SigCon_Out)