SDT System file rw C:\tau34\Q1228\Inap\inap.sdt

Source directory rw C:\tau34\Q1228\Inap\

Text readme −− TCAP\readme.txt

—— Chapter ASN.1 Files

ASN ASN.1 Text INCS2datatypes rw ASN1\incs2−dt.asn

ASN ASN.1 Text INCS2SSFSCFopsargs rw ASN1\ssf−scf.asn

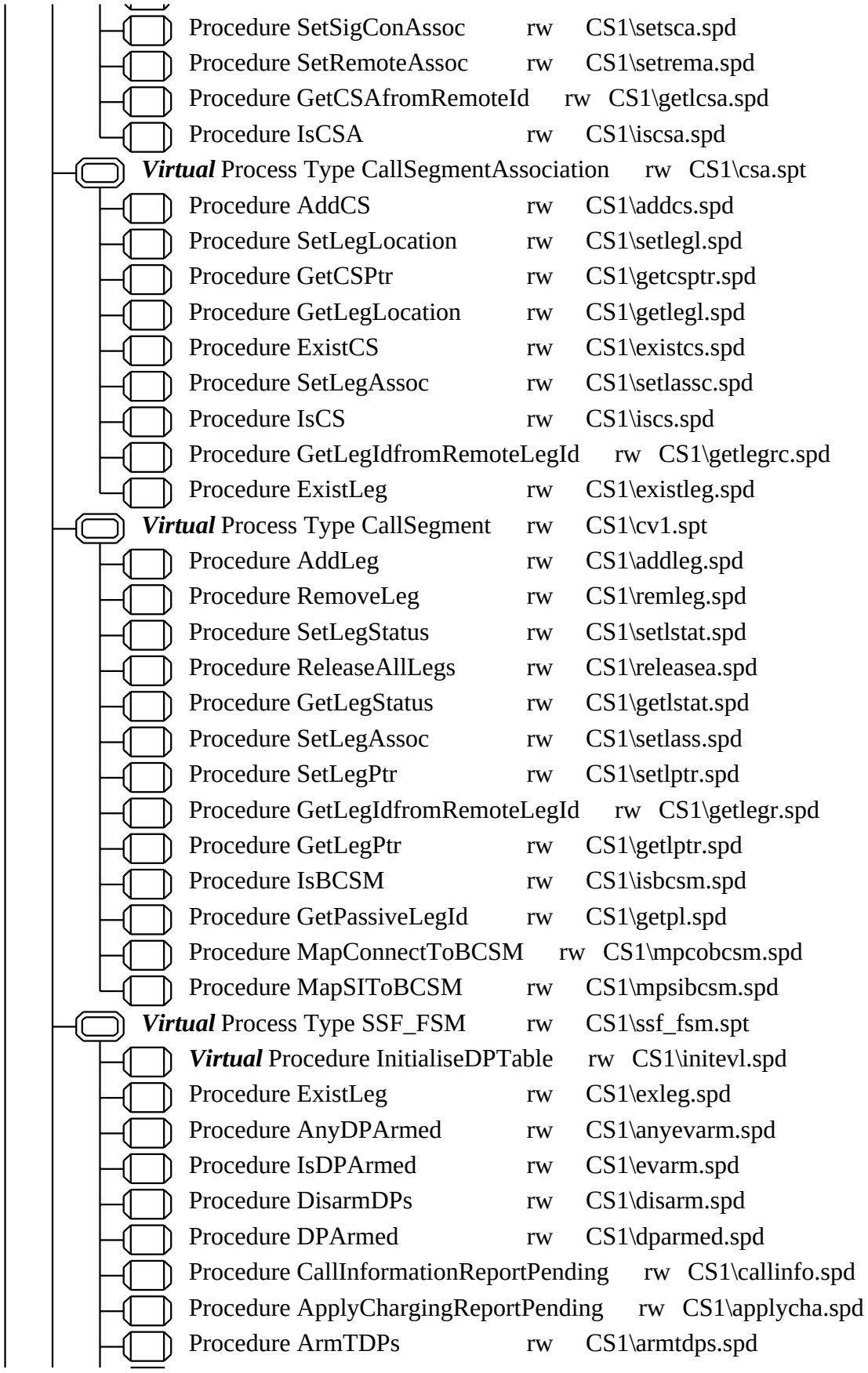
ASN ASN.1 Text INCS2BundleArg rw ASN1\incs2b.asn

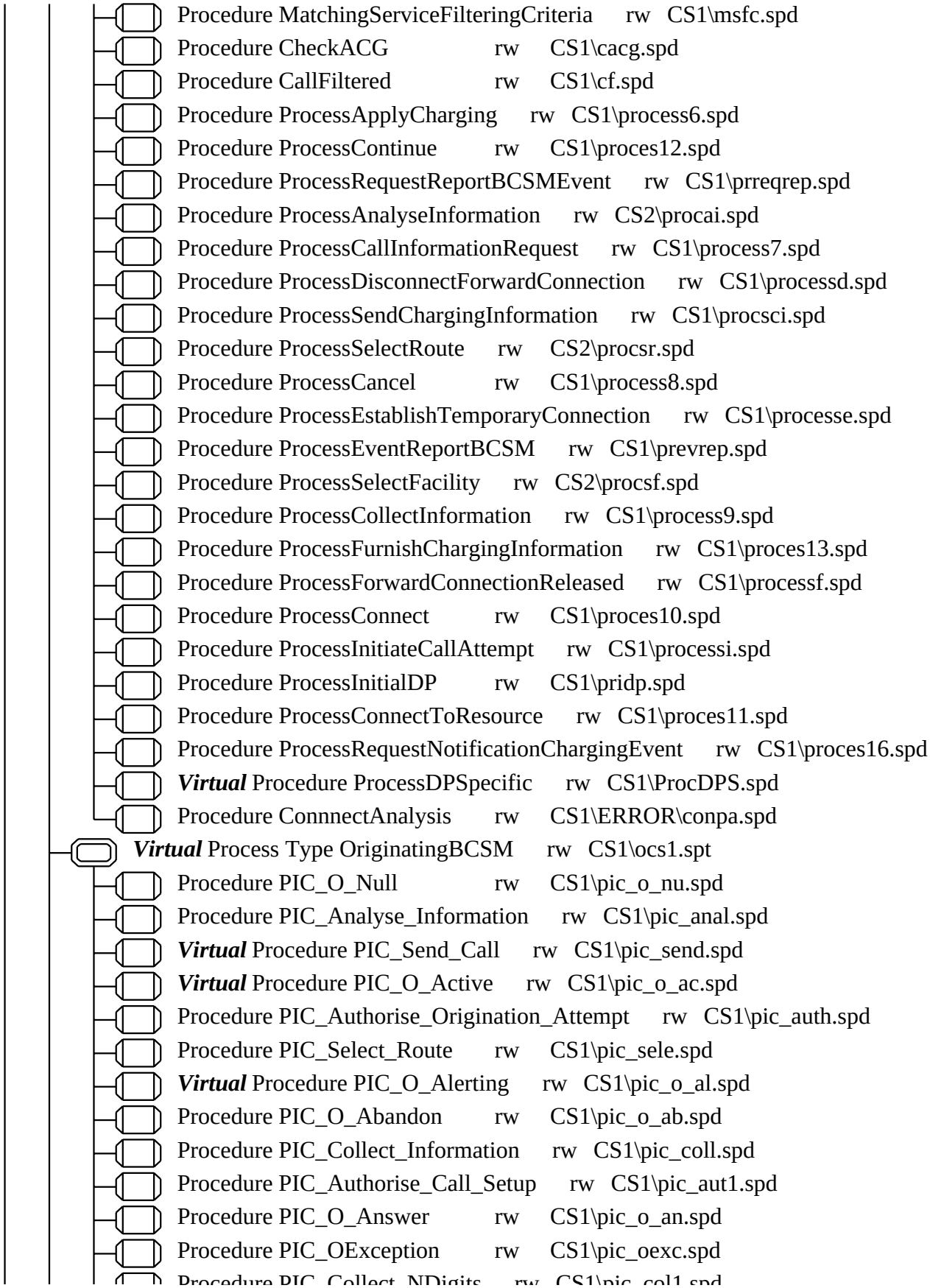
ASN ASN.1 Text INCS2Internals rw ASN1\cs2i.asn

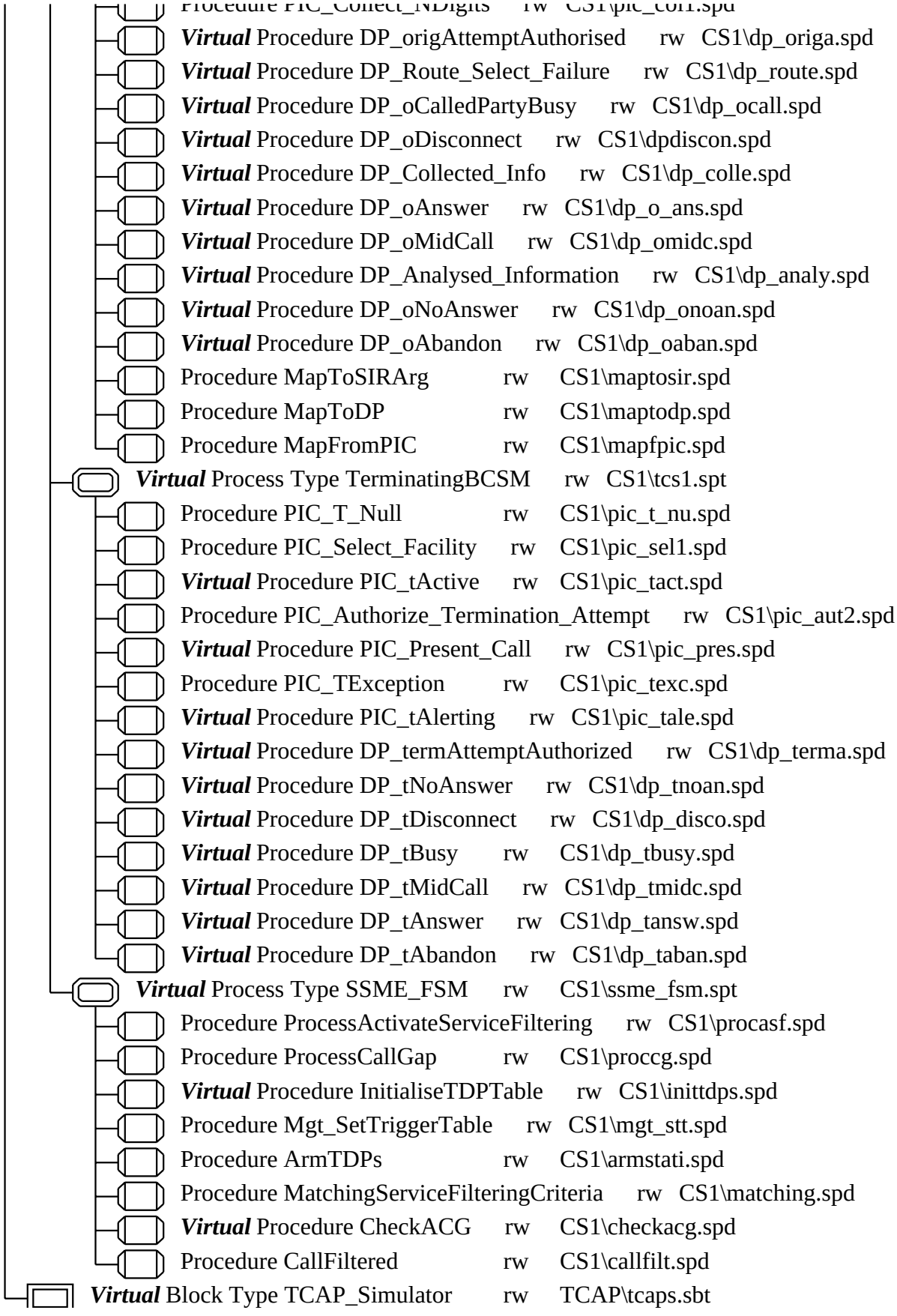
—— Chapter IN CS−1 Specification

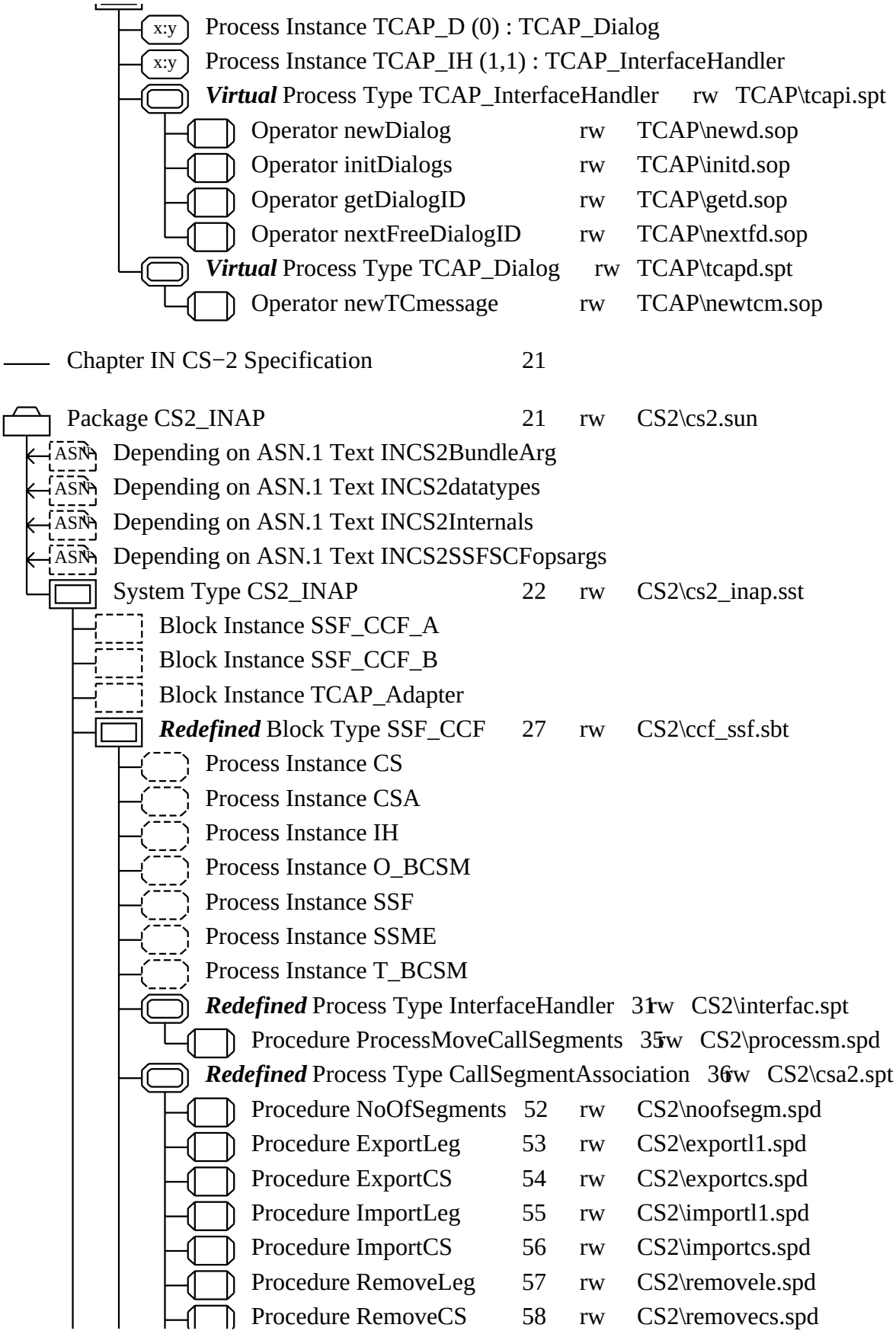
Package CS1_INAP rw CS1\cs1_inap.sun

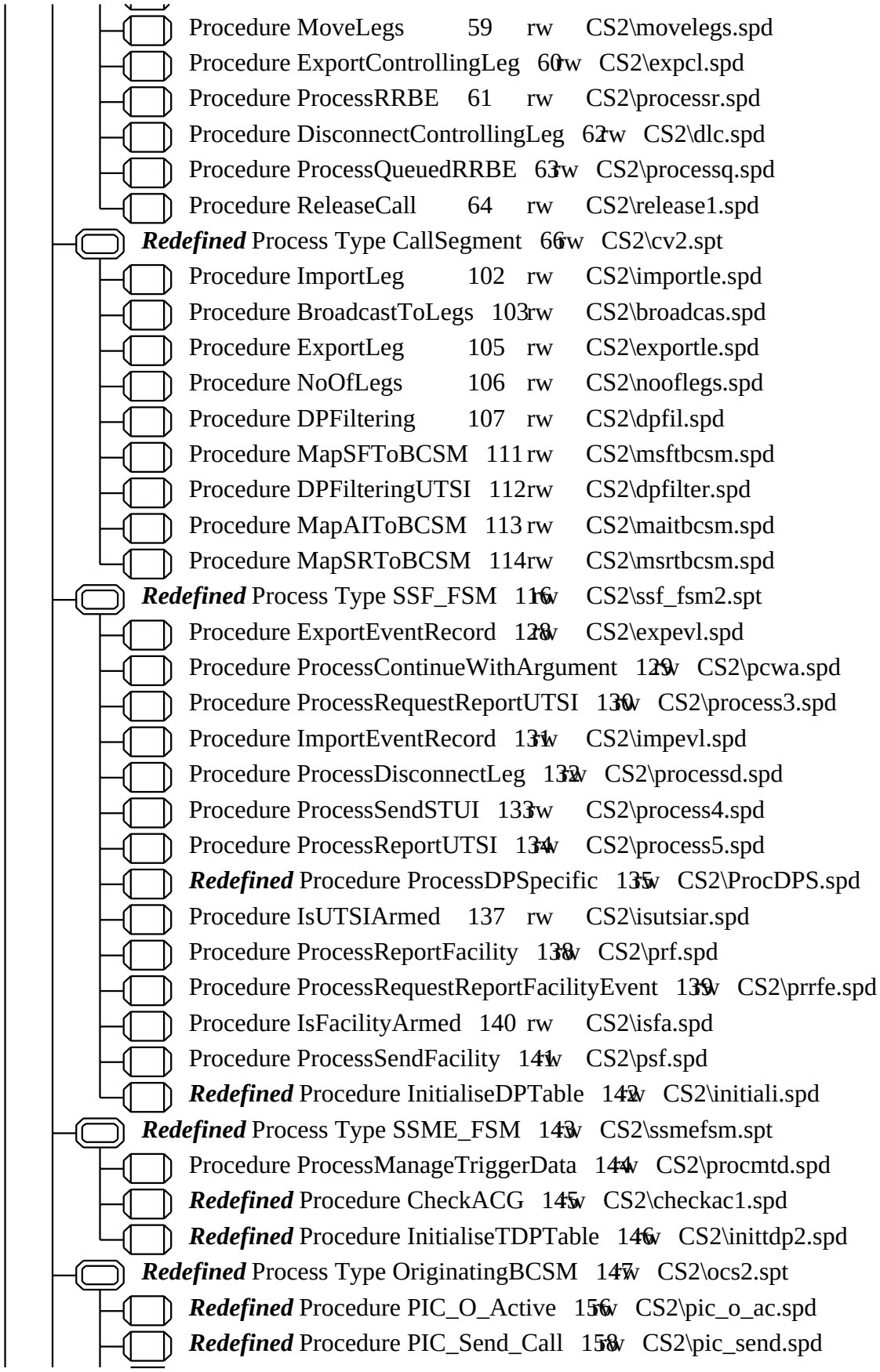
- ASN Depending on ASN.1 Text INCS2BundleArg
- ASN Depending on ASN.1 Text INCS2datatypes
- ASN Depending on ASN.1 Text INCS2Internals
- ASN Depending on ASN.1 Text INCS2SSFSCFopsargs
- System Type CS1_INAP rw CS1\cs1_inap.sst
  - x:y Block Instance SSF_CCF_A : SSF_CCF
  - x:y Block Instance SSF_CCF_B : SSF_CCF
  - x:y Block Instance TCAP_Adapter : TCAP_Simulator
  - ***Virtual*** Block Type SSF_CCF rw CS1\ssf1.sbt
    - x:y Process Instance CS (0) : CallSegment
    - x:y Process Instance CSA (0) : CallSegmentAssociation
    - x:y Process Instance IH (1,1) : InterfaceHandler
    - x:y Process Instance O_BCSM (0) : OriginatingBCSM
    - x:y Process Instance SSF (0) : SSF_FSM
    - x:y Process Instance SSME (1,1) : SSME_FSM
    - x:y Process Instance T_BCSM (0) : TerminatingBCSM
    - ***Virtual*** Process Type InterfaceHandler rw CS1\ih.spt
      - Procedure AllocateCSAId rw CS1\alldid.spd
      - Procedure AddCSAId rw CS1\adddid.spd
      - Procedure GetCSAIdfromCSA rw CS1\getdid.spd
      - Procedure GetCSAfromCSAId rw CS1\getcvd.spd
      - Procedure GetCSAfromSigConId rw CS1\getcvu.spd

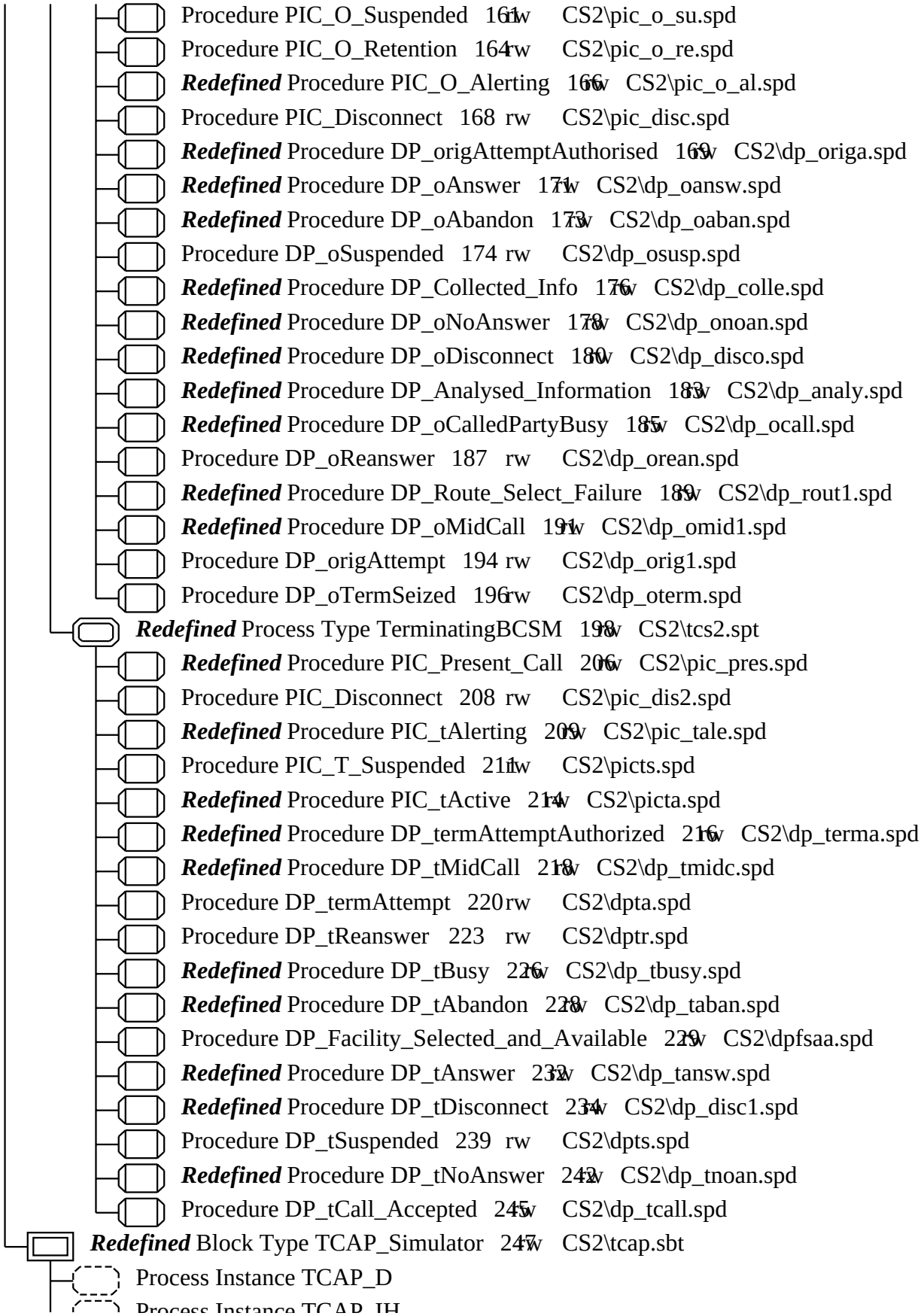

Procedure SetSigConAssoc rw CS1\setsca.spd
Procedure SetRemoteAssoc rw CS1\setrema.spd
Procedure GetCSAfromRemoteId rw CS1\getlcsa.spd
Procedure IsCSA rw CS1\iscsa.spd
Virtual Process Type CallSegmentAssociation rw CS1\csa.spt
Procedure AddCS rw CS1\addcs.spd
Procedure SetLegLocation rw CS1\setlegl.spd
Procedure GetCSPtr rw CS1\getcsptr.spd
Procedure GetLegLocation rw CS1\getlegl.spd
Procedure ExistCS rw CS1\existcs.spd
Procedure SetLegAssoc rw CS1\setlassc.spd
Procedure IsCS rw CS1\iscs.spd
Procedure GetLegIdfromRemoteLegId rw CS1\getlegrc.spd
Procedure ExistLeg rw CS1\existleg.spd
Virtual Process Type CallSegment rw CS1\cv1.spt
Procedure AddLeg rw CS1\addleg.spd
Procedure RemoveLeg rw CS1\remleg.spd
Procedure SetLegStatus rw CS1\setlstat.spd
Procedure ReleaseAllLegs rw CS1\releasea.spd
Procedure GetLegStatus rw CS1\getlstat.spd
Procedure SetLegAssoc rw CS1\setlass.spd
Procedure SetLegPtr rw CS1\setlptr.spd
Procedure GetLegIdfromRemoteLegId rw CS1\getlegr.spd
Procedure GetLegPtr rw CS1\getlptr.spd
Procedure IsBCSM rw CS1\isbcsm.spd
Procedure GetPassiveLegId rw CS1\getpl.spd
Procedure MapConnectToBCSM rw CS1\mpcobcsm.spd
Procedure MapSIToBCSM rw CS1\mpsibcsm.spd
Virtual Process Type SSF_FSM rw CS1\ssf_fsm.spt
Virtual Procedure InitialiseDPTable rw CS1\initevl.spd
Procedure ExistLeg rw CS1\exleg.spd
Procedure AnyDPArmed rw CS1\anyevarm.spd
Procedure IsDPArmed rw CS1\evarm.spd
Procedure DisarmDPs rw CS1\disarm.spd
Procedure DPArmed rw CS1\dparmed.spd
Procedure CallInformationReportPending rw CS1\callinfo.spd
Procedure ApplyChargingReportPending rw CS1\applycha.spd
Procedure ArmTDPs rw CS1\armtdps.spd

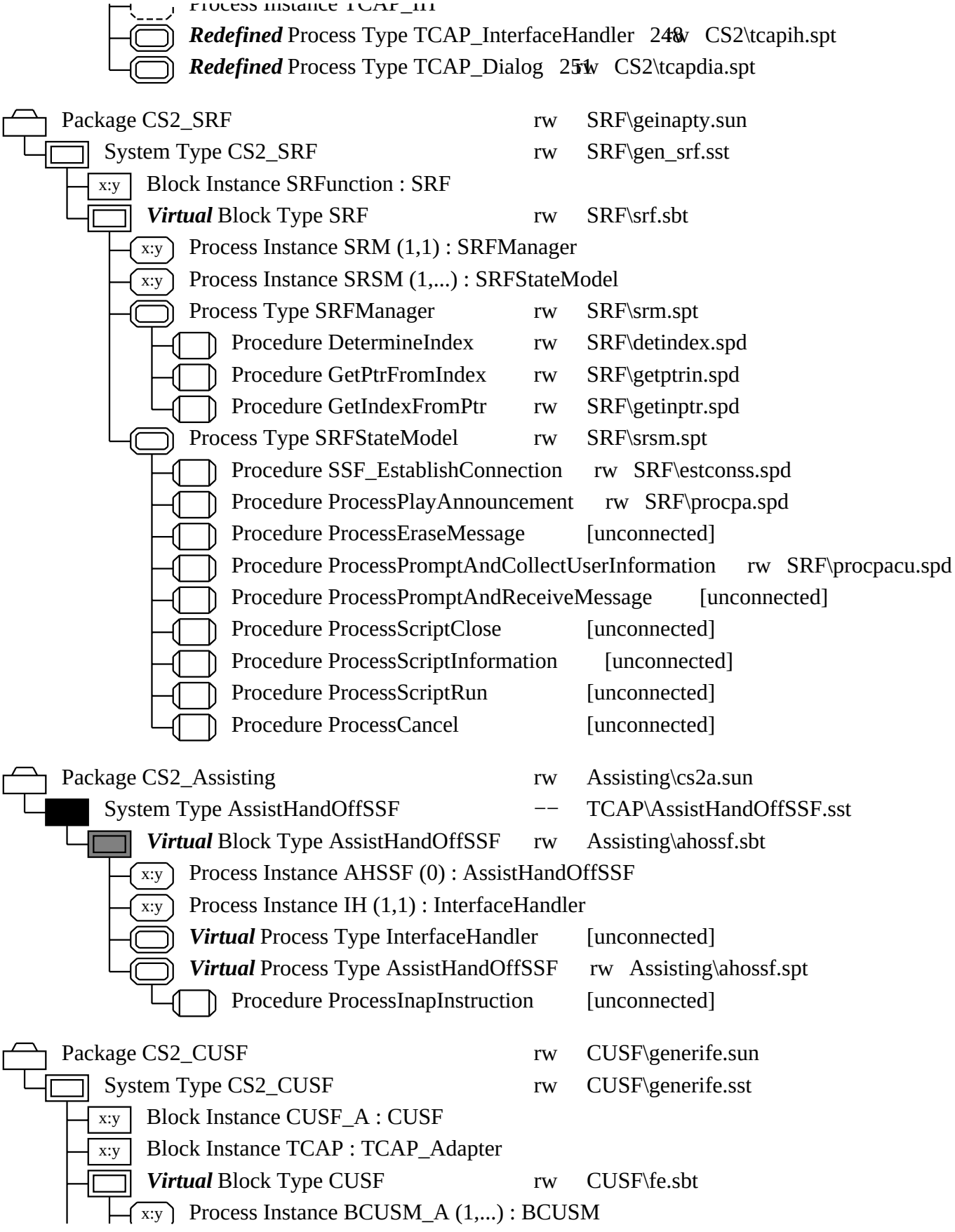

Procedure MatchingServiceFilteringCriteria rw CS1\msfc.spd
Procedure CheckACG rw CS1\cacg.spd
Procedure CallFiltered rw CS1\cf.spd
Procedure ProcessApplyCharging rw CS1\process6.spd
Procedure ProcessContinue rw CS1\proces12.spd
Procedure ProcessRequestReportBCSMEvent rw CS1\prreqrep.spd
Procedure ProcessAnalyseInformation rw CS2\procai.spd
Procedure ProcessCallInformationRequest rw CS1\process7.spd
Procedure ProcessDisconnectForwardConnection rw CS1\processd.spd
Procedure ProcessSendChargingInformation rw CS1\procsci.spd
Procedure ProcessSelectRoute rw CS2\procsr.spd
Procedure ProcessCancel rw CS1\process8.spd
Procedure ProcessEstablishTemporaryConnection rw CS1\processe.spd
Procedure ProcessEventReportBCSM rw CS1\prevrep.spd
Procedure ProcessSelectFacility rw CS2\procsf.spd
Procedure ProcessCollectInformation rw CS1\process9.spd
Procedure ProcessFurnishChargingInformation rw CS1\proces13.spd
Procedure ProcessForwardConnectionReleased rw CS1\processf.spd
Procedure ProcessConnect rw CS1\proces10.spd
Procedure ProcessInitiateCallAttempt rw CS1\processi.spd
Procedure ProcessInitialDP rw CS1\pridp.spd
Procedure ProcessConnectToResource rw CS1\proces11.spd
Procedure ProcessRequestNotificationChargingEvent rw CS1\proces16.spd
Virtual Procedure ProcessDPSpecific rw CS1\ProcDPS.spd
Procedure ConnnectAnalysis rw CS1\ERROR\conpa.spd
Virtual Process Type OriginatingBCSM rw CS1\ocs1.spt
Procedure PIC_O_Null rw CS1\pic_o_nu.spd
Procedure PIC_Analyse_Information rw CS1\pic_anal.spd
Virtual Procedure PIC_Send_Call rw CS1\pic_send.spd
Virtual Procedure PIC_O_Active rw CS1\pic_o_ac.spd
Procedure PIC_Authorise_Origination_Attempt rw CS1\pic_auth.spd
Procedure PIC_Select_Route rw CS1\pic_sele.spd
Virtual Procedure PIC_O_Alerting rw CS1\pic_o_al.spd
Procedure PIC_O_Abandon rw CS1\pic_o_ab.spd
Procedure PIC_Collect_Information rw CS1\pic_coll.spd
Procedure PIC_Authorise_Call_Setup rw CS1\pic_aut1.spd
Procedure PIC_O_Answer rw CS1\pic_o_an.spd
Procedure PIC_OException rw CS1\pic_oexc.spd

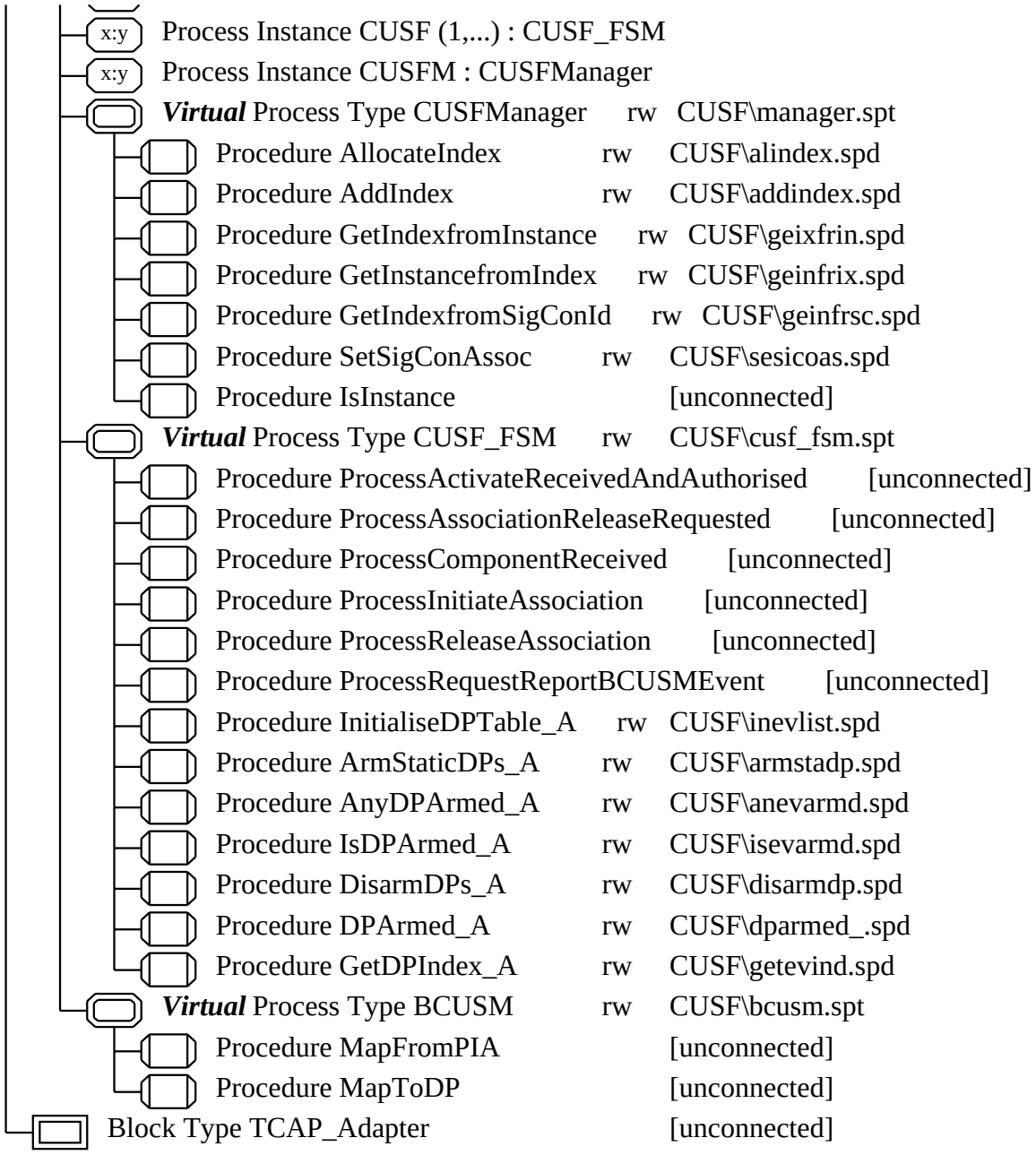

Procedure PIC_Collect_NDigits rw CS1\pic_col1.spd
Virtual Procedure DP_origAttemptAuthorised rw CS1\dp_origa.spd
Virtual Procedure DP_Route_Select_Failure rw CS1\dp_route.spd
Virtual Procedure DP_oCalledPartyBusy rw CS1\dp_ocall.spd
Virtual Procedure DP_oDisconnect rw CS1\dpdiscon.spd
Virtual Procedure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


x:y Process Instance TCAP_D (0) : TCAP_Dialog
x:y Process Instance TCAP_IH (1,1) : TCAP_InterfaceHandler
Virtual Process Type TCAP_InterfaceHandler rw TCAP\tcapi.spt
Operator newDialog rw TCAP\newd.sop
Operator initDialogs rw TCAP\initd.sop
Operator getDialogID rw TCAP\getd.sop
Operator nextFreeDialogID rw TCAP\nextfd.sop
Virtual Process Type TCAP_Dialog rw TCAP\tcapd.spt
Operator newTCmessage rw TCAP\newtcm.sop
Chapter IN CS−2 Specification 21
Package CS2_INAP 21 rw CS2\cs2.sun
ASN Depending on ASN.1 Text INCS2BundleArg
ASN Depending on ASN.1 Text INCS2datatypes
ASN Depending on ASN.1 Text INCS2Internals
ASN Depending on ASN.1 Text INCS2SSFSCFopsargs
System Type CS2_INAP 22 rw CS2\cs2_inap.sst
Block Instance SSF_CCF_A
Block Instance SSF_CCF_B
Block Instance TCAP_Adapter
Redefined Block Type SSF_CCF 27 rw CS2\ccf_ssf.sbt
Process Instance CS
Process Instance CSA
Process Instance IH
Process Instance O_BCSM
Process Instance SSF
Process Instance SSME
Process Instance T_BCSM
Redefined Process Type InterfaceHandler 31 rw CS2\interfac.spt
Procedure ProcessMoveCallSegments 35 rw CS2\processm.spd
Redefined Process Type CallSegmentAssociation 36 rw CS2\csa2.spt
Procedure NoOfSegments 52 rw CS2\noofsegm.spd
Procedure ExportLeg 53 rw CS2\exportl1.spd
Procedure ExportCS 54 rw CS2\exportcs.spd
Procedure ImportLeg 55 rw CS2\importl1.spd
Procedure ImportCS 56 rw CS2\importcs.spd
Procedure RemoveLeg 57 rw CS2\removele.spd
Procedure RemoveCS 58 rw CS2\removecs.spd


Procedure MoveLegs 59 rw CS2\movelegs.spd
Procedure ExportControllingLeg 60 rw CS2\expcl.spd
Procedure ProcessRRBE 61 rw CS2\processr.spd
Procedure DisconnectControllingLeg 62 rw CS2\dlc.spd
Procedure ProcessQueuedRRBE 63 rw CS2\processq.spd
Procedure ReleaseCall 64 rw CS2\release1.spd
Redefined Process Type CallSegment 66 rw CS2\cv2.spt
Procedure ImportLeg 102 rw CS2\importle.spd
Procedure BroadcastToLegs 103 rw CS2\broadcas.spd
Procedure ExportLeg 105 rw CS2\exportle.spd
Procedure NoOfLegs 106 rw CS2\nooflegs.spd
Procedure DPFiltering 107 rw CS2\dpfil.spd
Procedure MapSFToBCSM 111 rw CS2\msftbcsm.spd
Procedure DPFilteringUTSI 112 rw CS2\dpfilter.spd
Procedure MapAIToBCSM 113 rw CS2\maitbcsm.spd
Procedure MapSRToBCSM 114 rw CS2\msrtbcsm.spd
Redefined Process Type SSF_FSM 116 rw CS2\ssf_fsm2.spt
Procedure ExportEventRecord 128 rw CS2\expevl.spd
Procedure ProcessContinueWithArgument 129 rw CS2\pcwa.spd
Procedure ProcessRequestReportUTSI 130 rw CS2\process3.spd
Procedure ImportEventRecord 131 rw CS2\impevl.spd
Procedure ProcessDisconnectLeg 132 rw CS2\processd.spd
Procedure ProcessSendSTUI 133 rw CS2\process4.spd
Procedure ProcessReportUTSI 134 rw CS2\process5.spd
Redefined Procedure ProcessDPSpecific 135 rw CS2\ProcDPS.spd
Procedure IsUTSIArmed 137 rw CS2\isutsiar.spd
Procedure ProcessReportFacility 138 rw CS2\prf.spd
Procedure ProcessRequestReportFacilityEvent 139 rw CS2\prrfe.spd
Procedure IsFacilityArmed 140 rw CS2\isfa.spd
Procedure ProcessSendFacility 141 rw CS2\psf.spd
Redefined Procedure InitialiseDPTable 142 rw CS2\initiali.spd
Redefined Process Type SSME_FSM 143 rw CS2\ssmefsm.spt
Procedure ProcessManageTriggerData 144 rw CS2\procmtd.spd
Redefined Procedure CheckACG 145 rw CS2\checkac1.spd
Redefined Procedure InitialiseTDPTable 146 rw CS2\inittdp2.spd
Redefined Process Type OriginatingBCSM 147 rw CS2\ocs2.spt
Redefined Procedure PIC_O_Active 156 rw CS2\pic_o_ac.spd
Redefined Procedure PIC_Send_Call 158 rw CS2\pic_send.spd


Procedure PIC_O_Suspended 161 rw CS2\pic_o_su.spd
Procedure PIC_O_Retention 164 rw CS2\pic_o_re.spd
Redefined Procedure PIC_O_Alerting 166 rw CS2\pic_o_al.spd
Procedure PIC_Disconnect 168 rw CS2\pic_disc.spd
Redefined Procedure DP_origAttemptAuthorised 169 rw CS2\dp_origa.spd
Redefined Procedure DP_oAnswer 171 rw CS2\dp_oansw.spd
Redefined Procedure DP_oAbandon 173 rw CS2\dp_oaban.spd
Procedure DP_oSuspended 174 rw CS2\dp_osusp.spd
Redefined Procedure DP_Collected_Info 176 rw CS2\dp_colle.spd
Redefined Procedure DP_oNoAnswer 178 rw CS2\dp_onoan.spd
Redefined Procedure DP_oDisconnect 180 rw CS2\dp_disco.spd
Redefined Procedure DP_Analysed_Information 183 rw CS2\dp_analy.spd
Redefined Procedure DP_oCalledPartyBusy 185 rw CS2\dp_ocall.spd
Procedure DP_oReanswer 187 rw CS2\dp_orean.spd
Redefined Procedure DP_Route_Select_Failure 189 rw CS2\dp_rout1.spd
Redefined Procedure DP_oMidCall 191 rw CS2\dp_omid1.spd
Procedure DP_origAttempt 194 rw CS2\dp_orig1.spd
Procedure DP_oTermSeized 196 rw CS2\dp_oterm.spd
Redefined Process Type TerminatingBCSM 198 rw CS2\tcs2.spt
Redefined Procedure PIC_Present_Call 206 rw CS2\pic_pres.spd
Procedure PIC_Disconnect 208 rw CS2\pic_dis2.spd
Redefined Procedure PIC_tAlerting 209 rw CS2\pic_tale.spd
Procedure PIC_T_Suspended 211 rw CS2\picts.spd
Redefined Procedure PIC_tActive 214 rw CS2\picta.spd
Redefined Procedure DP_termAttemptAuthorized 216 rw CS2\dp_terma.spd
Redefined Procedure DP_tMidCall 218 rw CS2\dp_tmidc.spd
Procedure DP_termAttempt 220 rw CS2\dpta.spd
Procedure DP_tReanswer 223 rw CS2\dptr.spd
Redefined Procedure DP_tBusy 226 rw CS2\dp_tbusy.spd
Redefined Procedure DP_tAbandon 228 rw CS2\dp_taban.spd
Procedure DP_Facility_Selected_and_Available 229 rw CS2\dpfsaa.spd
Redefined Procedure DP_tAnswer 232 rw CS2\dp_tansw.spd
Redefined Procedure DP_tDisconnect 234 rw CS2\dp_disc1.spd
Procedure DP_tSuspended 239 rw CS2\dpts.spd
Redefined Procedure DP_tNoAnswer 242 rw CS2\dp_tnoan.spd
Procedure DP_tCall_Accepted 245 rw CS2\dp_tcall.spd
Redefined Block Type TCAP_Simulator 247 rw CS2\tcap.sbt
Process Instance TCAP_D
Process Instance TCAP_IH


Process Instance TCAP_IH
Redefined Process Type TCAP_InterfaceHandler 248 rw CS2\tcapih.spt
Redefined Process Type TCAP_Dialog 251 rw CS2\tcapdia.spt
Package CS2_SRF rw SRF\geinapty.sun
System Type CS2_SRF rw SRF\gen_srf.sst
x:y Block Instance SRFunction : SRF
Virtual Block Type SRF rw SRF\srf.sbt
x:y Process Instance SRM (1,1) : SRFManager
x:y Process Instance SRSM (1,...) : SRFStateModel
Process Type SRFManager rw SRF\srm.spt
Procedure DetermineIndex rw SRF\detindex.spd
Procedure GetPtrFromIndex rw SRF\getptrin.spd
Procedure GetIndexFromPtr rw SRF\getinptr.spd
Process Type SRFStateModel rw SRF\srsm.spt
Procedure SSF_EstablishConnection rw SRF\estconss.spd
Procedure ProcessPlayAnnouncement rw SRF\procpa.spd
Procedure ProcessEraseMessage [unconnected]
Procedure ProcessPromptAndCollectUserInformation rw SRF\procpacu.spd
Procedure ProcessPromptAndReceiveMessage [unconnected]
Procedure ProcessScriptClose [unconnected]
Procedure ProcessScriptInformation [unconnected]
Procedure ProcessScriptRun [unconnected]
Procedure ProcessCancel [unconnected]
Package CS2_Assisting rw Assisting\cs2a.sun
System Type AssistHandOffSSF −− TCAP\AssistHandOffSSF.sst
Virtual Block Type AssistHandOffSSF rw Assisting\ahossf.sbt
x:y Process Instance AHSSF (0) : AssistHandOffSSF
x:y Process Instance IH (1,1) : InterfaceHandler
Virtual Process Type InterfaceHandler [unconnected]
Virtual Process Type AssistHandOffSSF rw Assisting\ahossf.spt
Procedure ProcessInapInstruction [unconnected]
Package CS2_CUSF rw CUSF\generife.sun
System Type CS2_CUSF rw CUSF\generife.sst
x:y Block Instance CUSF_A : CUSF
x:y Block Instance TCAP : TCAP_Adapter
Virtual Block Type CUSF rw CUSF\fe.sbt
x:y Process Instance BCUSM_A (1,...) : BCUSM



Chapter System Instantiations

System CS1_INAP rw CS1\cs1.ssy

System CS2_INAP rw CS2\cs2_inap.ssy

Chapter CS−1 Examples

MSC CI_Overview rw EXAMPLES\ciov.msc

MSC CI_Detailed rw EXAMPLES\cid.msc

 MSC CO_Overview rw EXAMPLES\coov.msc

 MSC CO_Detailed rw EXAMPLES\cod.msc

—— Chapter CPH Examples

 MSC 3Pty_Overview rw EXAMPLES\3ptyov.msc

 MSC 3Pty_Detailed rw EXAMPLES\3ptyd.msc

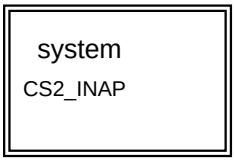
 MSC CF_Overview rw EXAMPLES\cfov.msc

 MSC CF_Detailed rw EXAMPLES\cfd.msc

—— Chapter CS−1 Test Purposes

USE INCS2SSFSCFopsargs;
USE INCS2BundleArg;
USE CS1_INAP;

Package CS2_INAP 1(1)

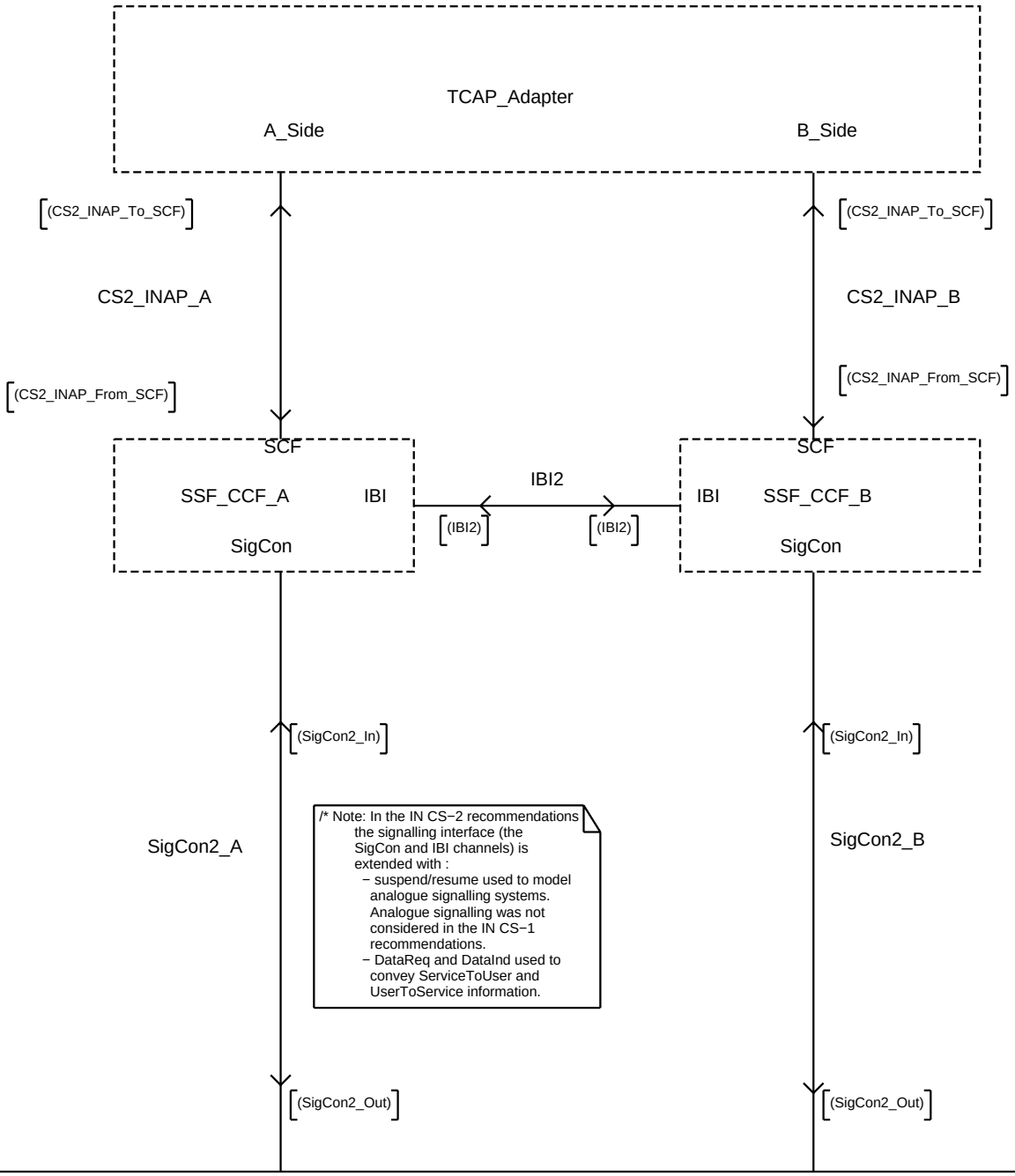


System Type CS2_INAP

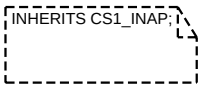
INHERITS CS1_INAP;

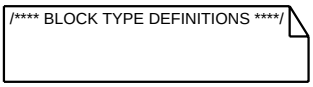
/* Note: This specification should be read in conjunction with the IN CS−1 SDL specification. */

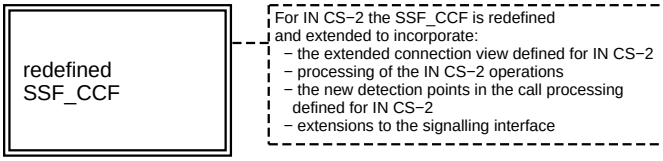
/* Notes: − Dashed symbols refer to entities defined in the IN CS−1 SDL specifications.
− For IN CS−2 the INAP interface (the CS2_INAP channels) is extended with the IN CS−2 operations. */

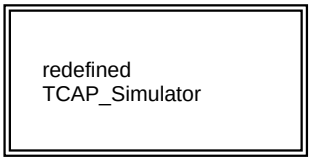


/* Note: In the IN CS−2 recommendations the signalling interface (the SigCon and IBI channels) is extended with :
− suspend/resume used to model analogue signalling systems. Analogue signalling was not considered in the IN CS−1 recommendations.
− DataReq and DataInd used to convey ServiceToUser and UserToService information.


INHERITS CS1_INAP;


/**** BLOCK TYPE DEFINITIONS ****/


redefined
SSF_CCF
For IN CS−2 the SSF_CCF is redefined
and extended to incorporate:
− the extended connection view defined for IN CS−2
− processing of the IN CS−2 operations
− the new detection points in the call processing
defined for IN CS−2
− extensions to the signalling interface


redefined
TCAP_Simulator

INHERITS CS1_INAP;

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

```
SIGNAL
  ContinueWithArgument(InvokeID,CSAID,continueWithArgumentArg),
  CreateCallSegmentAssociation(InvokeID,CSAID,CreateCallSegmentAssociationArg),
  CreateCallSegmentAssociationResult(InvokeID,CSAID,CreateCallSegmentAssociationResultArg),
  DisconnectLeg(InvokeID,CSAID,DisconnectLegArg),
  DisconnectLegResult(InvokeID,CSAID),
  DisconnectForwardConnectionWithArgument(InvokeID,CSAID,DisconnectForwardConnectionWithArgumentArg),
  EntityReleased(CSAID,EntityReleasedArg),
  EventReportFAcility(CSAID,EventReportFAcilityArg),
  FacilitySelectedAndAvailable(CSAID,FacilitySelectedAndAvailableArg),
  ManageTriggerData(InvokeID,DialogIDtype,ManageTriggerDataArg),
  ManageTriggerDataResult(InvokeID,DialogIDtype,ManageTriggerDataResultArg),
  MergeCallSegments(InvokeID,CSAID,MergeCallSegmentsArg),
  MergeCallSegmentsResult(InvokeID,CSAID),
  MoveCallSegments(InvokeID,CSAID,MoveCallSegmentsArg),
  MoveCallSegmentsResult(InvokeID,CSAID),
  MoveLeg(InvokeID,CSAID,MoveLegArg),
  MoveLegResult(InvokeID,CSAID),
  OriginationAttempt(CSAID,OriginationAttemptArg),
  OSuspended(CSAID,OSuspendedArg),
  ReportUTSI(CSAID,ReportUTSIArg),
  RequestReportFacilityEvent(InvokeID,CSAID,RequestReportFacilityEventArg),
  RequestReportUTSI(InvokeID,CSAID,RequestReportUTSIArg),
  SendFacilityInformation(InvokeID,CSAID,SendFacilityInformationArg),
  SendSTUI(InvokeID,CSAID,SendSTUIArg),
  SplitLeg(InvokeID,CSAID,SplitLegArg),
  SplitLegResult(InvokeID,CSAID),
  TerminationAttempt(CSAID,TerminationAttemptArg),
  TSuspended(CSAID,TSuspendedArg);
```

INHERITS CS1_INAP;

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

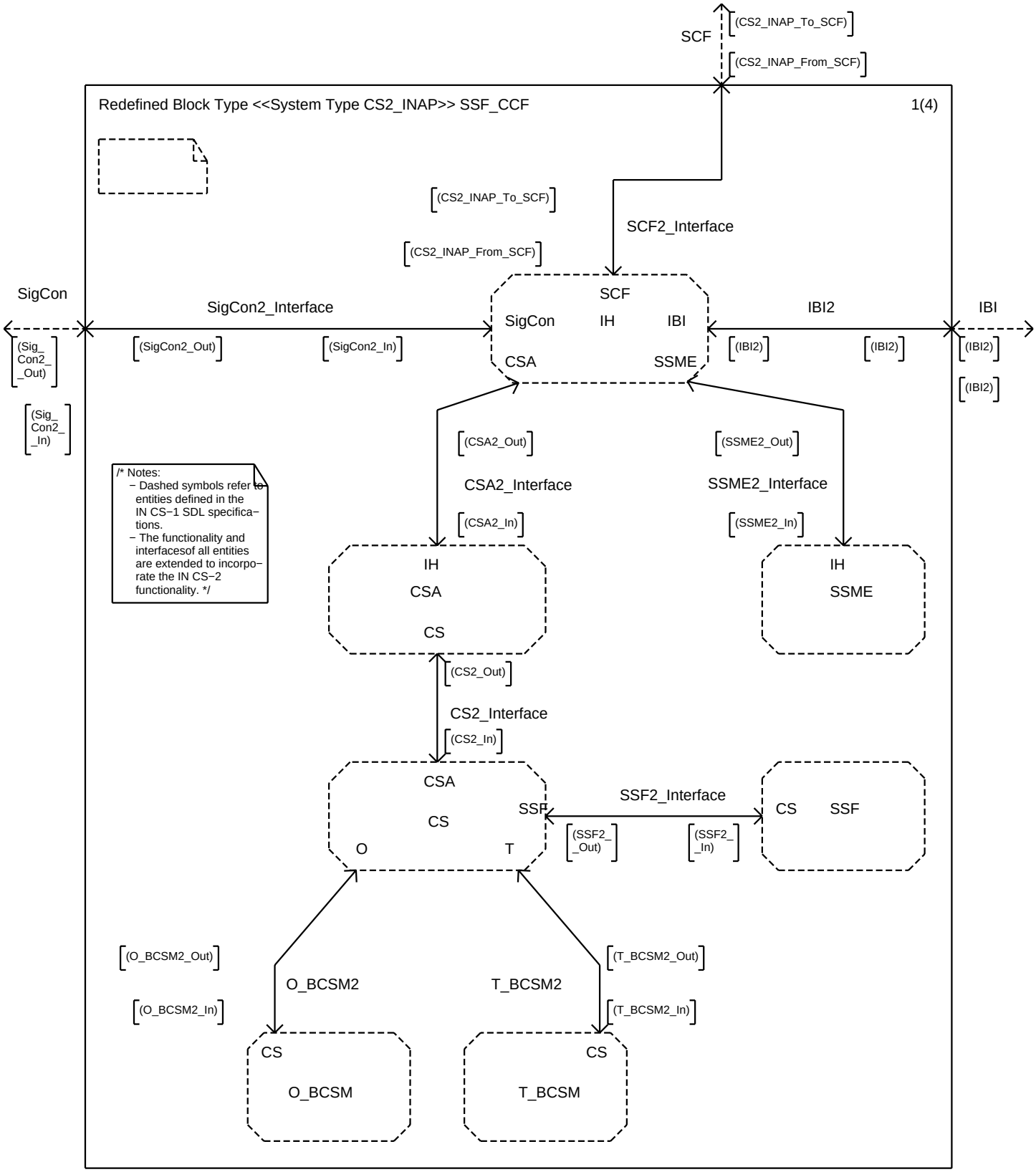
```
/* SSF −> SCF */
SIGNALLIST CS2_INAP_To_SCF =
  CreateCallSegmentAssociationResult,
  DisconnectLegResult,
  EntityReleased,
  ManageTriggerDataResult,
  MergeCallSegmentsResult,
  MoveCallSegmentsResult,
  MoveLegResult,
  ReportUTSI,
  SplitLegResult;

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

INHERITS CS1_INAP;

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

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

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

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

SIGNALLIST SigCon2_Out =
  NetworkSuspendReq,
  NetworkResumeReq,
  DataReq;

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

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

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


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

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

/* Note: All process type definitions are inherited from IN CS−1 and redefined to incorporate the new functionality required for IN CS−2.

redefined Interface_ Handler

The InterFaceHandler (IH) process type is extended to manage:
- Processing of the IN CS−2 operations MoveCallSegments and CreateCSA.
- The passing of new primitives between the Signalling Control Interface and the CSAs.
- The passing of new primitives between the IN CS−2 SSF−SCF Interface and the CSA.

redefined CallSegment_ Association

The CallSegmentAssociation (CSA) process type is extended to manage:
- Processing of IN CS−2 call segment related operations.
- The passing of new Signalling Control primitives between the InterfaceHandler and the CallSegment.
- The passing of IN CS−2 primitives between the InterfaceHandler and the CallSegment.

redefined CallSegment

The CallSegment (CS) process type is extended to manage:
- The extended connection view (in terms of legs) of the call segment.
- Processing of IN CS−2 leg related operations.
- Filtering of detection points
- The passing of new Signalling Control primitives between the CallSegmentAssociation and the BCSMs.
- The passing of IN CS−2 primitives between the CallSegmentAssociation and the SSF−FSM.

redefined SSF_FSM

The SSF−FSM process type is extended to manage:
- The processing of IN CS−2 operations.

redefined SSME_FSM

The SSME_FSM is extended to manage:
- The processing of ManageTriggerData

redefined Originating_ BCSM

The OriginatingBCSM (O−BCSM) process type is extended to manage:
- The extensions to the call processing of originating half calls.

redefined Terminating_ BCSM

The TerminatingBCSM (T−BCSM) process type is extended to manage:
- The extensions to the call processing of terminating half calls.

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

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

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


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

SYNTYPE ExportCSType = PId
ENDSYNTYPE;
```

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

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

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

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

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

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

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

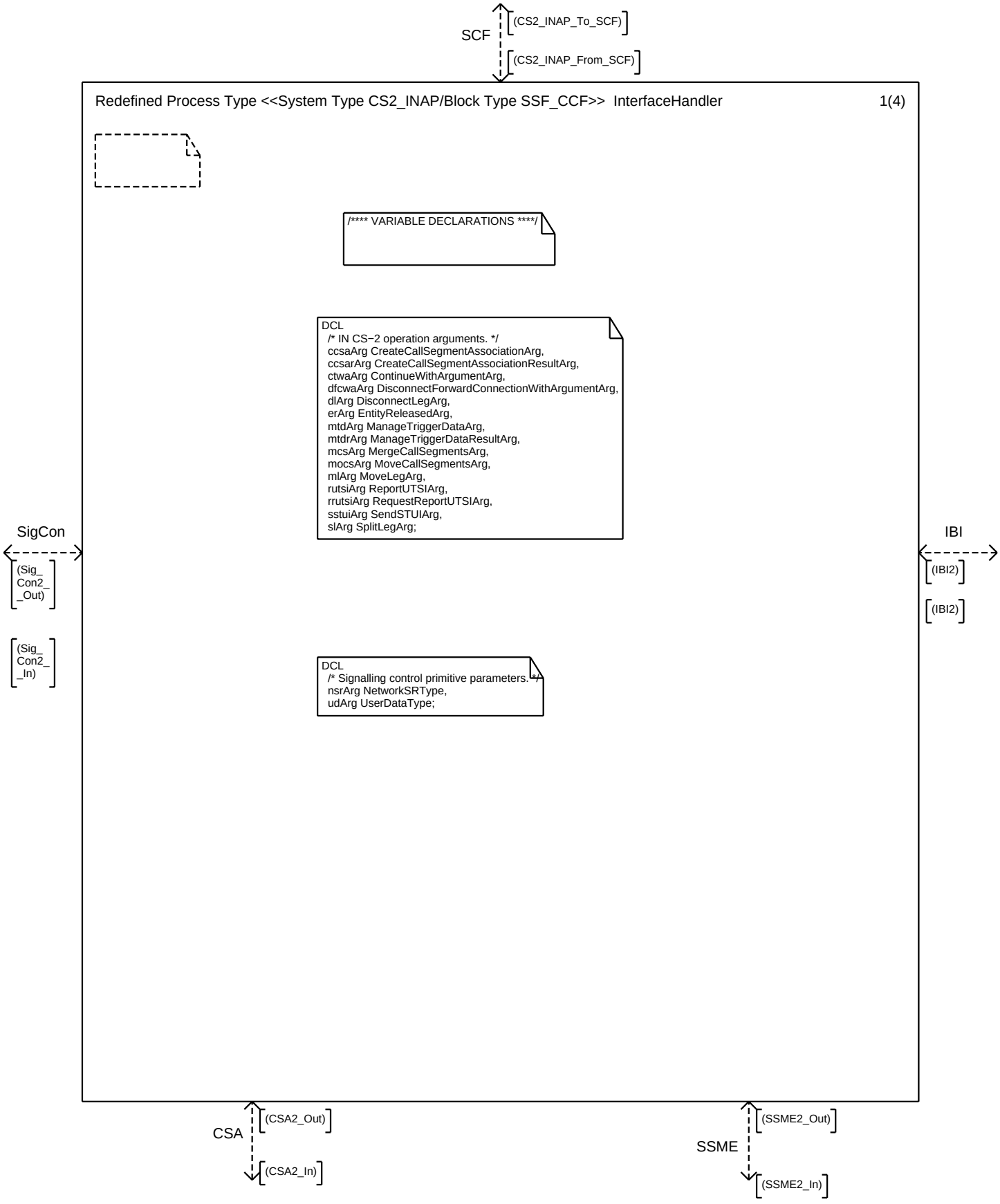
```
SIGNALLIST CS2_INAP_CS_IN =
  ContinueWithArgument,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  RequestReportFacilityEvent,
  RequestReportUTSI,
  SendFacilityInformation,
  SendSTUI,
  SplitLeg;

SIGNALLIST CS2_INAP_CS_OUT =
  DisconnectLegResult,
  EntityReleased,
  ReportUTSI,
  SplitLegResult;
```

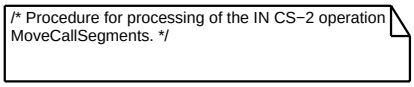
```
SIGNALLIST CS2_INAP_SSME_IN =
  ManageTriggerData;

SIGNALLIST CS2_INAP_SSME_OUT =
  ManageTriggerDataResult;
```

```
SIGNALLIST CS2_INAP_SSF_IN =
  ContinueWithArgument,
  DisconnectForwardConnectionWithArgument,
  DisconnectLeg,
  RequestReportFacilityEvent,
  RequestReportUTSI,
  SendFacilityInformation,
  SendSTUI;

SIGNALLIST CS2_INAP_SSF_OUT =
  DisconnectLegResult,
  ReportUTSI;
```

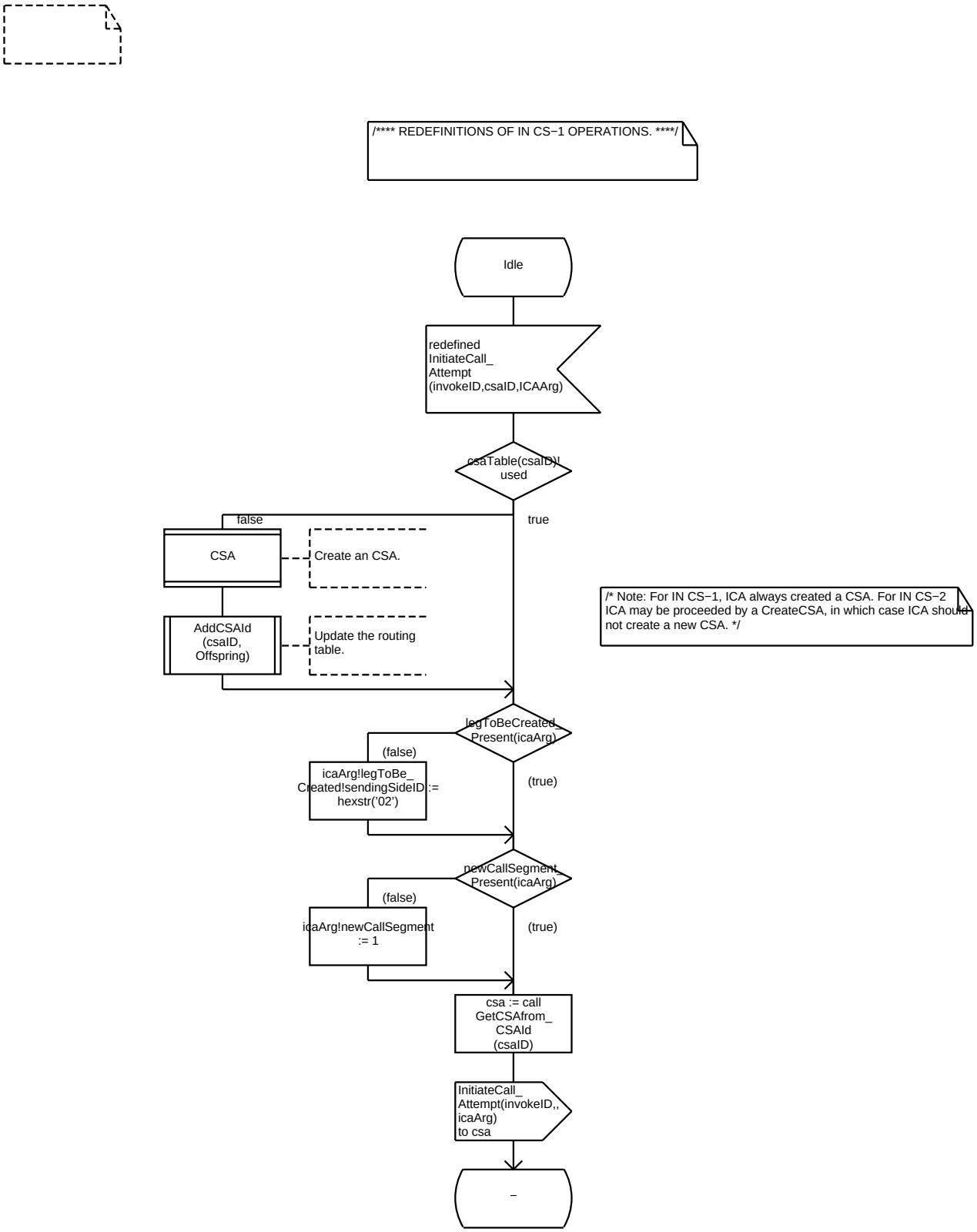
```
SIGNALLIST SSME2_In =
  (CS2_INAP_SSME_IN);

SIGNALLIST SSME2_Out =
  (CS2_INAP_SSME_OUT);

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

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

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

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

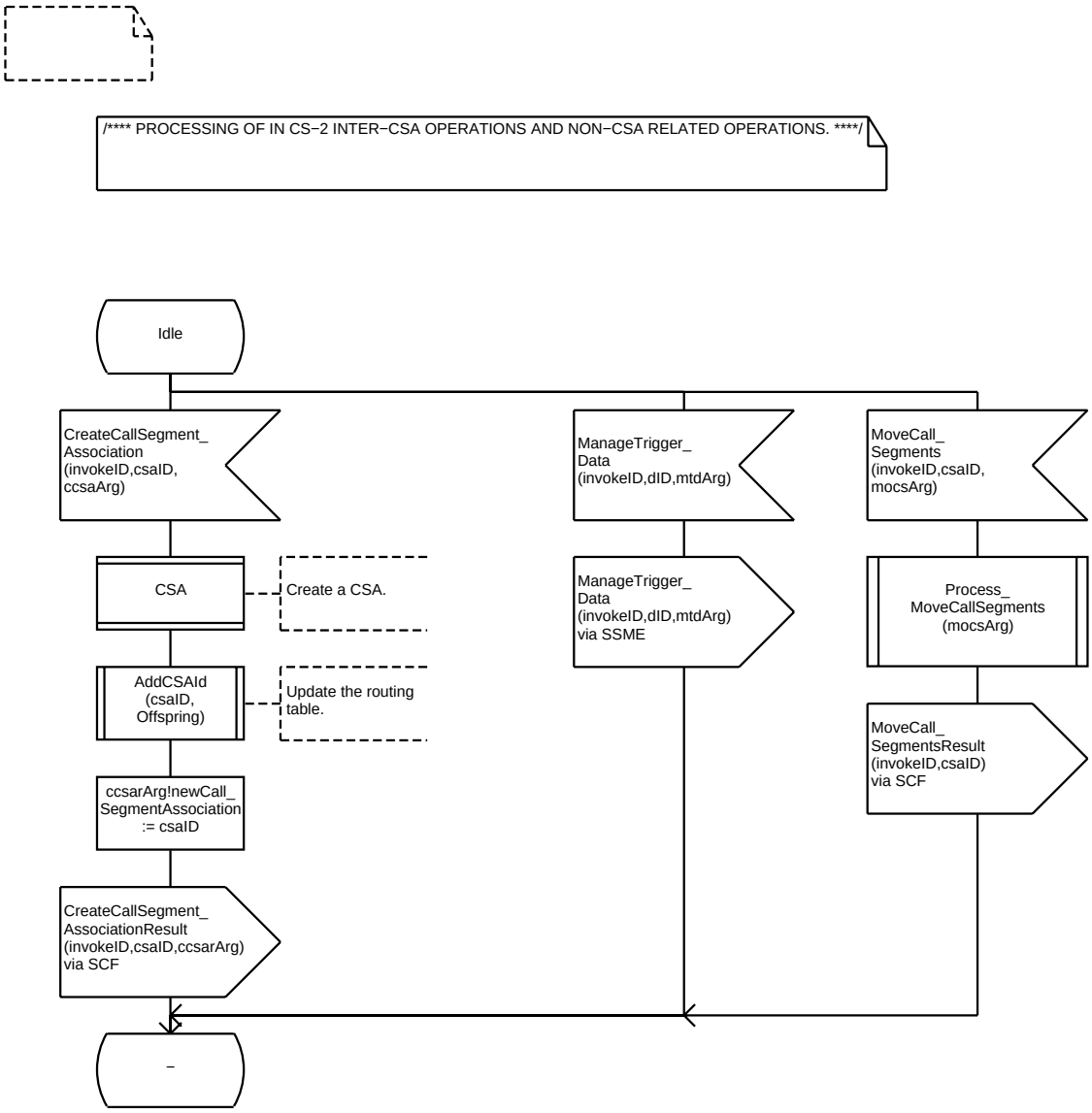
```
/* Definition of signal lists. */

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

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

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

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

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

SIGNALLIST SSF2_Out =
  (CS2_INAP_SSF_OUT),
  ExportLegResp,
  ImportLegResp;
```


SCF
(CS2_INAP_To_SCF)
(CS2_INAP_From_SCF)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> InterfaceHandler
1(4)
/**** VARIABLE DECLARATIONS ****/
DCL
/* IN CS−2 operation arguments. */
ccsaArg CreateCallSegmentAssociationArg,
ccsarArg CreateCallSegmentAssociationResultArg,
ctwaArg ContinueWithArgumentArg,
dfcwaArg DisconnectForwardConnectionWithArgumentArg,
dlArg DisconnectLegArg,
erArg EntityReleasedArg,
mtdArg ManageTriggerDataArg,
mtdrArg ManageTriggerDataResultArg,
mcsArg MergeCallSegmentsArg,
mocsArg MoveCallSegmentsArg,
mlArg MoveLegArg,
rutsiArg ReportUTSIArg,
rrutsiArg RequestReportUTSIArg,
sstuiArg SendSTUIArg,
slArg SplitLegArg;
SigCon
(Sig_ Con2_ _Out)
(Sig_ Con2_ _In)
IBI
(IBI2)
(IBI2)
DCL
/* Signalling control primitive parameters. */
nsrArg NetworkSRType,
udArg UserDataType;
CSA
(CSA2_Out)
(CSA2_In)
SSME
(SSME2_Out)
(SSME2_In)

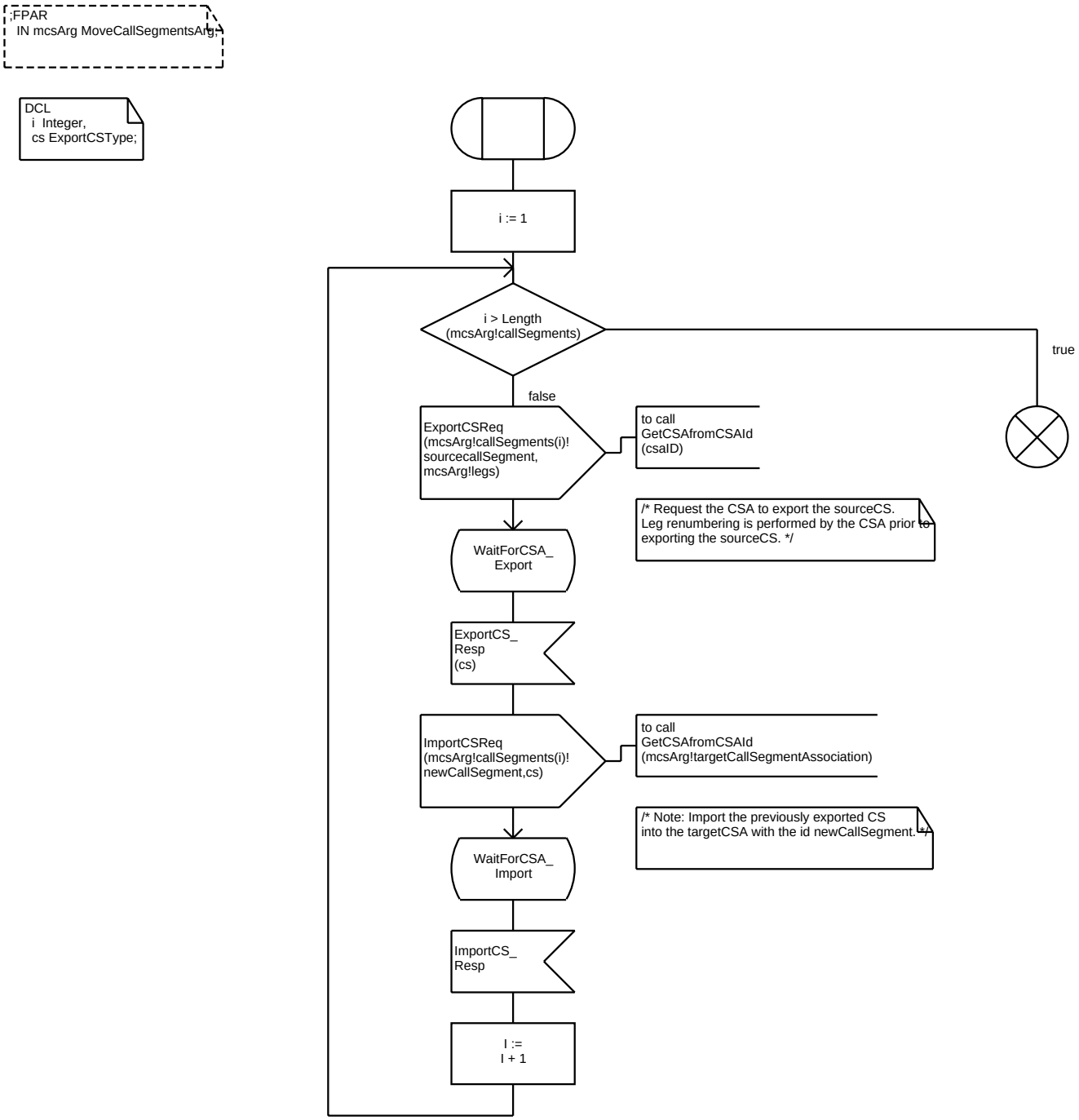

/* Procedure for processing of the IN CS−2 operation
MoveCallSegments. */

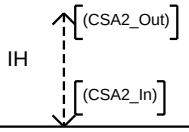

Process_
MoveCall_
Segments


/**** REDEFINITIONS OF IN CS−1 OPERATIONS. ****/
Idle
redefined
InitiateCall_
Attempt
(invokeID,csaID,ICAArg)
csaTable(csaID)!
used
false
true
CSA
Create an CSA.
AddCSAId
(csaID,
Offspring)
Update the routing
table.
/* Note: For IN CS−1, ICA always created a CSA. For IN CS−2
ICA may be proceeded by a CreateCSA, in which case ICA should
not create a new CSA. */
legToBeCreated_
Present(icaArg)
(false)
icaArg!legToBe_
Created!sendingSideID :=
hexstr('02')
(true)
newCallSegment_
Present(icaArg)
(false)
icaArg!newCallSegment
:= 1
(true)
csa := call
GetCSAfrom_
CSAId
(csaID)
InitiateCall_
Attempt(invokeID,,
icaArg)
to csa
–


/**** PROCESSING OF IN CS−2 INTER−CSA OPERATIONS AND NON−CSA RELATED OPERATIONS. ****/
Idle
CreateCallSegment_ Association (invokeID,csaID, ccsaArg)
ManageTrigger_ Data (invokeID,dID,mtdArg)
MoveCall_ Segments (invokeID,csaID, mocsArg)
CSA
Create a CSA.
ManageTrigger_ Data (invokeID,dID,mtdArg) via SSME
Process_ MoveCallSegments (mocsArg)
AddCSAId (csaID, Offspring)
Update the routing table.
MoveCall_ SegmentsResult (invokeID,csaID) via SCF
ccsarArg!newCall_ SegmentAssociation := csaID
CreateCallSegment_ AssociationResult (invokeID,csaID,ccsarArg) via SCF
−

Procedure ProcessMoveCallSegments 1(1)

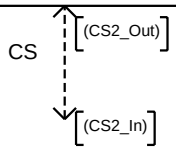




Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegmentAssociation 1(16)

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

```
/* For IN CS−2 the CSA must maintain a queue of RRBEs for arming of
events of legs that does not yet exist. */

NEWTYPE RRBEQueueSeq STRUCT
  Q Boolean;
  RRBE BCSMEvent;
ENDNEWTYPE;

NEWTYPE RRBEQueueType
  ARRAY(LegType,RRBEQueueSeq)
ENDNEWTYPE;
```



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

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

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

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

 leg   ExportLegType,
 legs  ExportLegsType,
 expCS ExportCSType;
```

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

/* Operations on call segments. */

- NoOf_Segments — Returns the number of call segments.
- Export_CS — Exports a call segment.
- Import_CS — Imports a call segment.
- Remove_CS — Removes a call segment.

/* Operations on legs. */

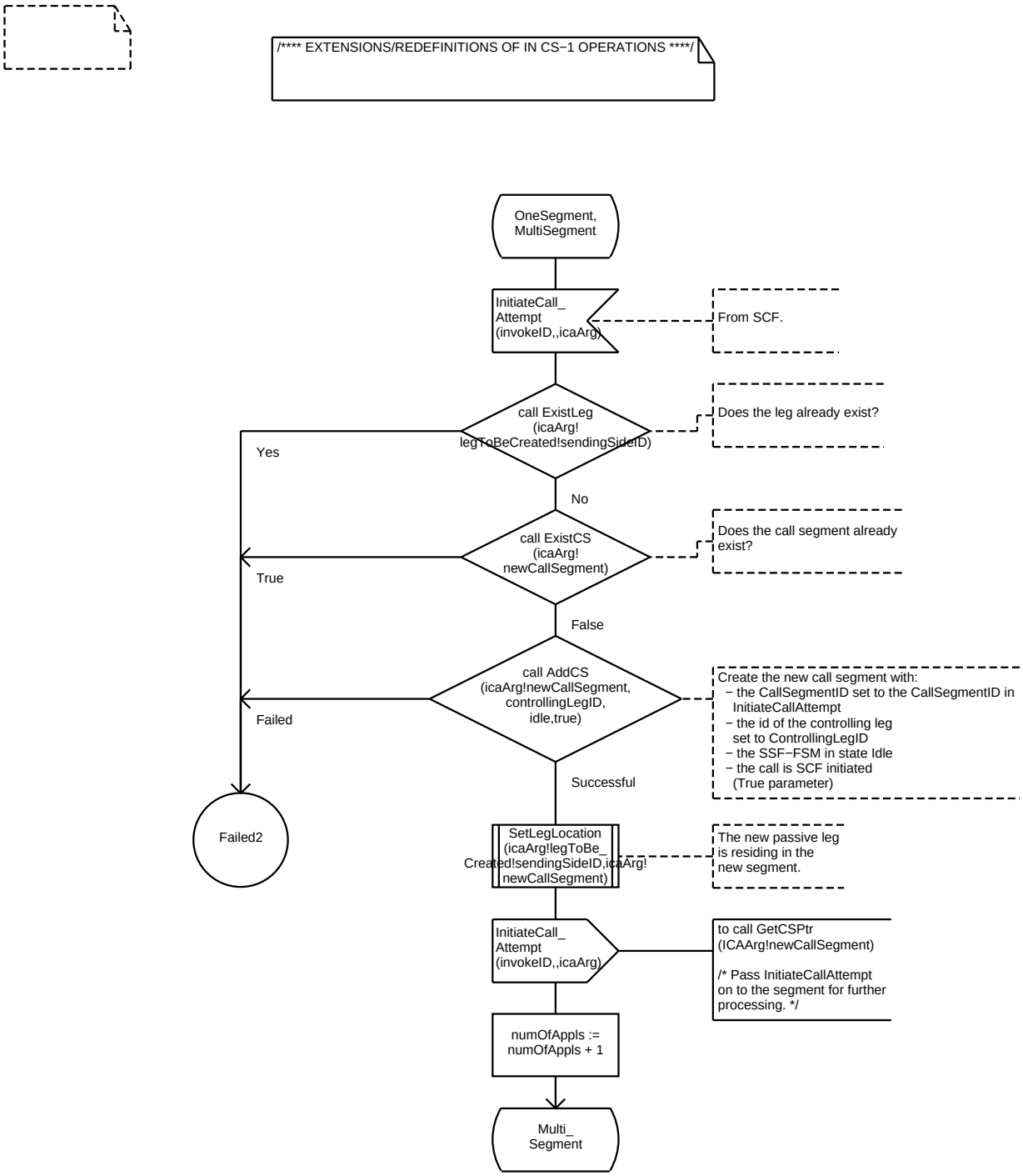
- Export_Leg — Exports a leg from a call segment.
- Import_Leg — Imports a leg into a call segment.
- Remove_Leg — Removes a leg from the leg location table.
- Move_Legs — Moves all passive legs from one call segment into another call segment.
- Export_Controlling_Leg — Sets the status of the controlling leg to surrogate in all CSs.

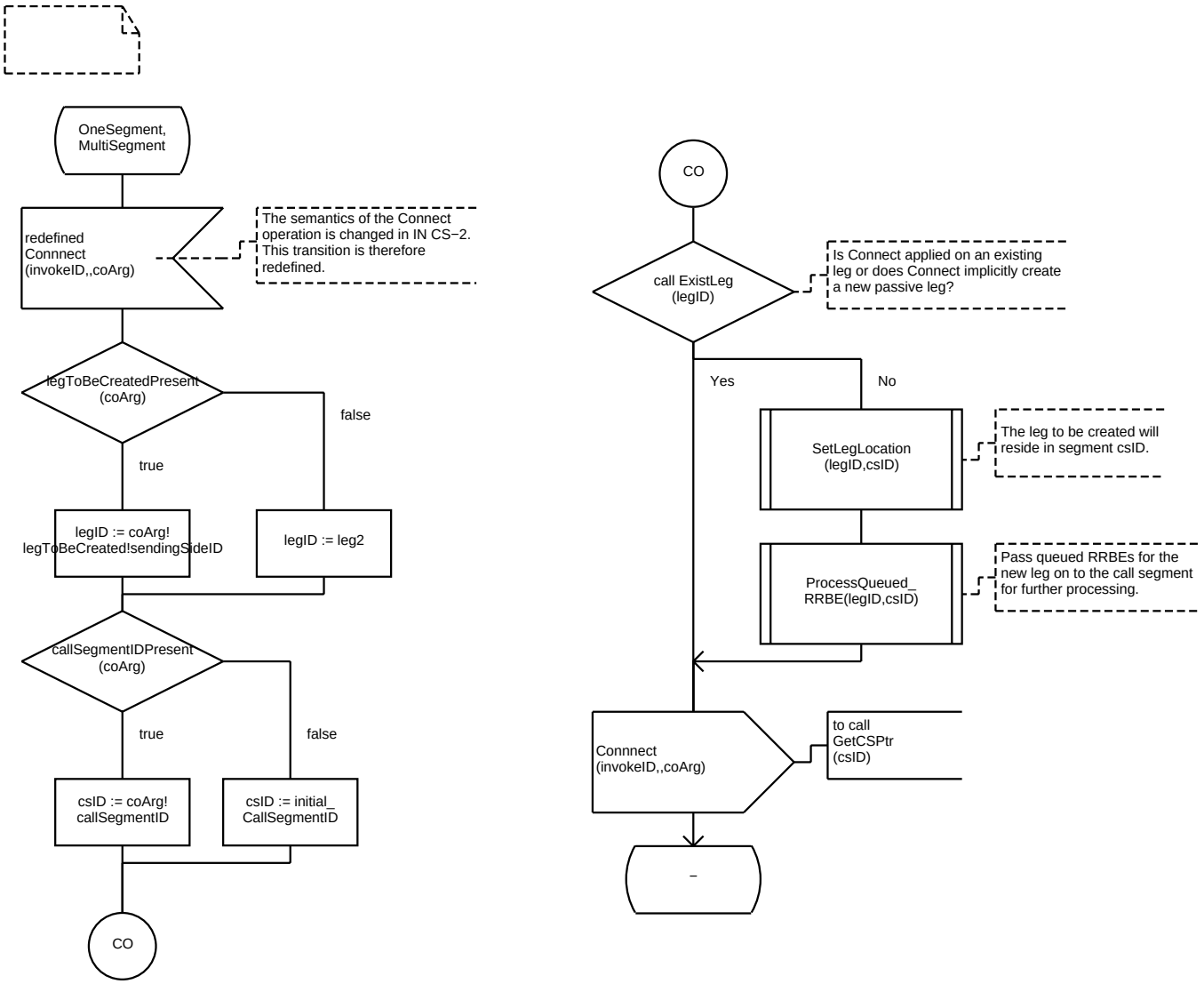
/* Operations on the queue of RRBEs. */

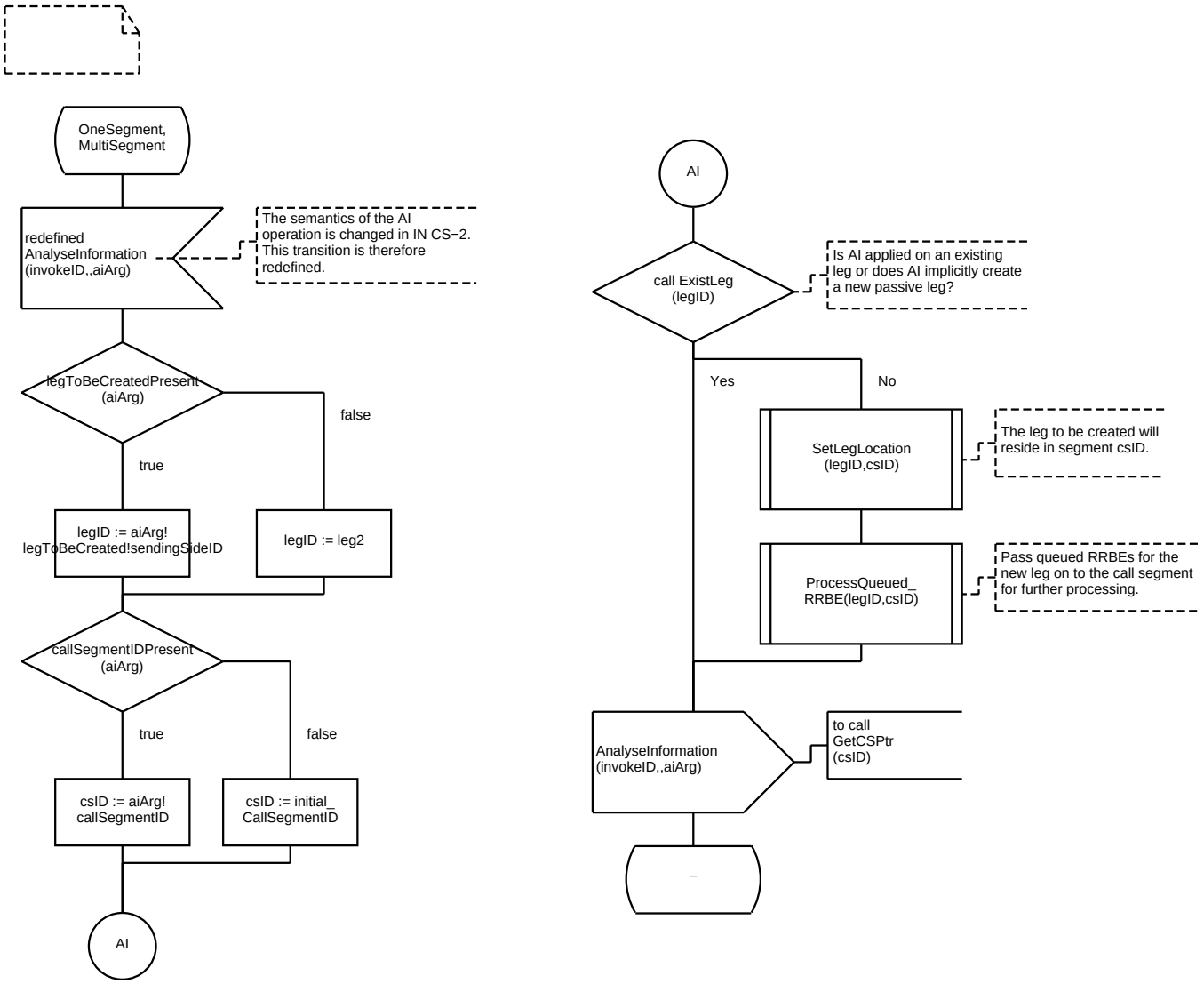
- Process_RRBE
- Process_Queued_RRBE

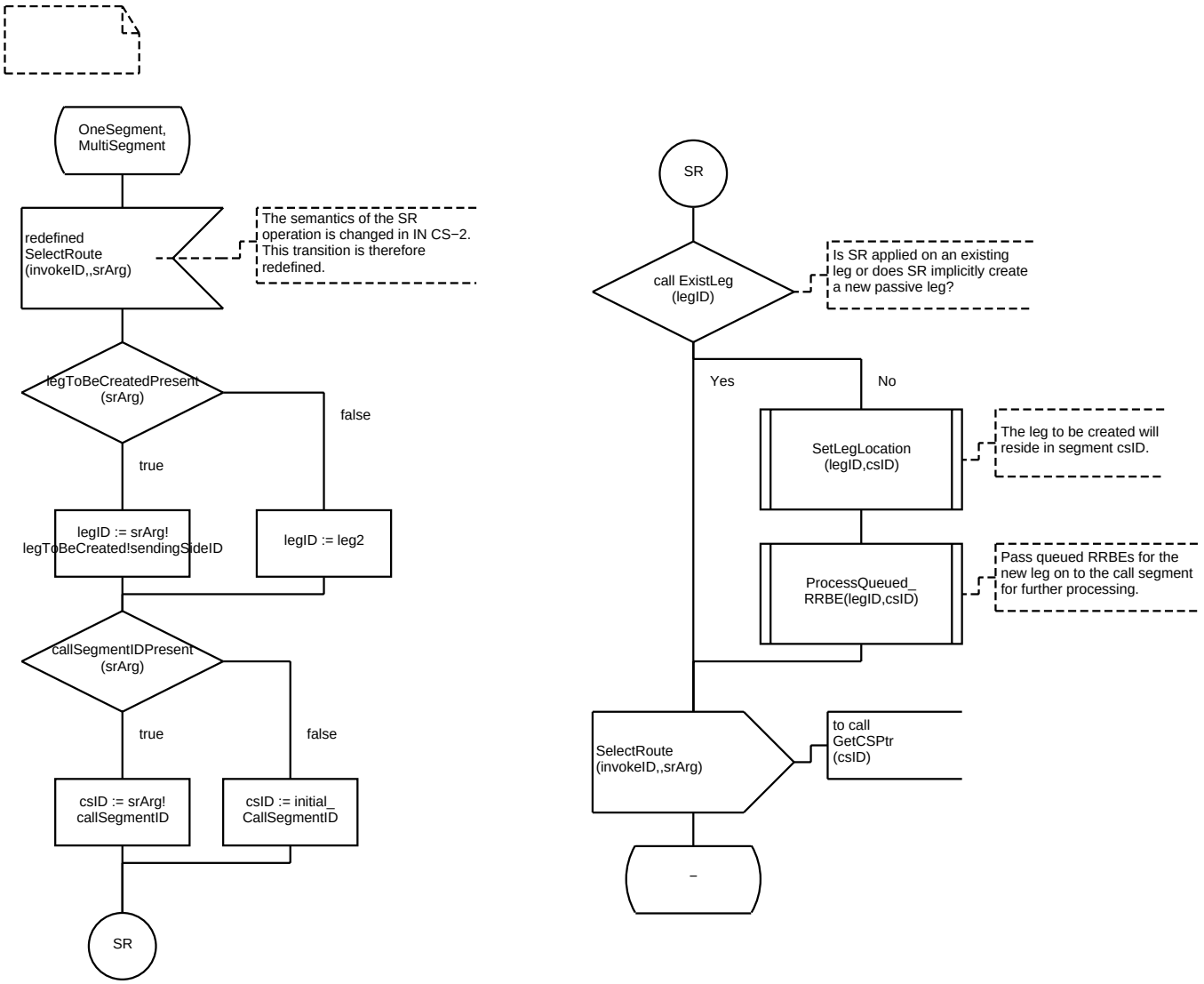
/* Processing of IN CS−2 operations. */

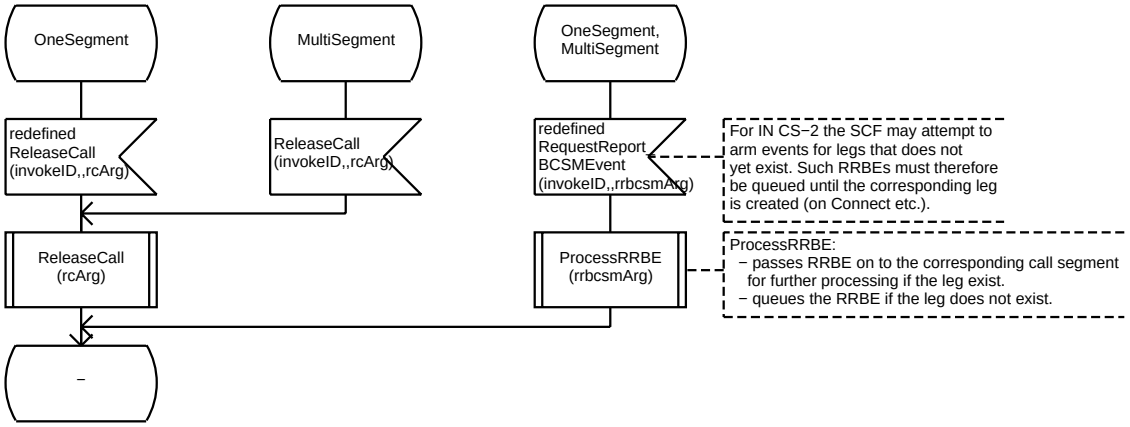
- Disconnect_Controlling_Leg
- Release_Call

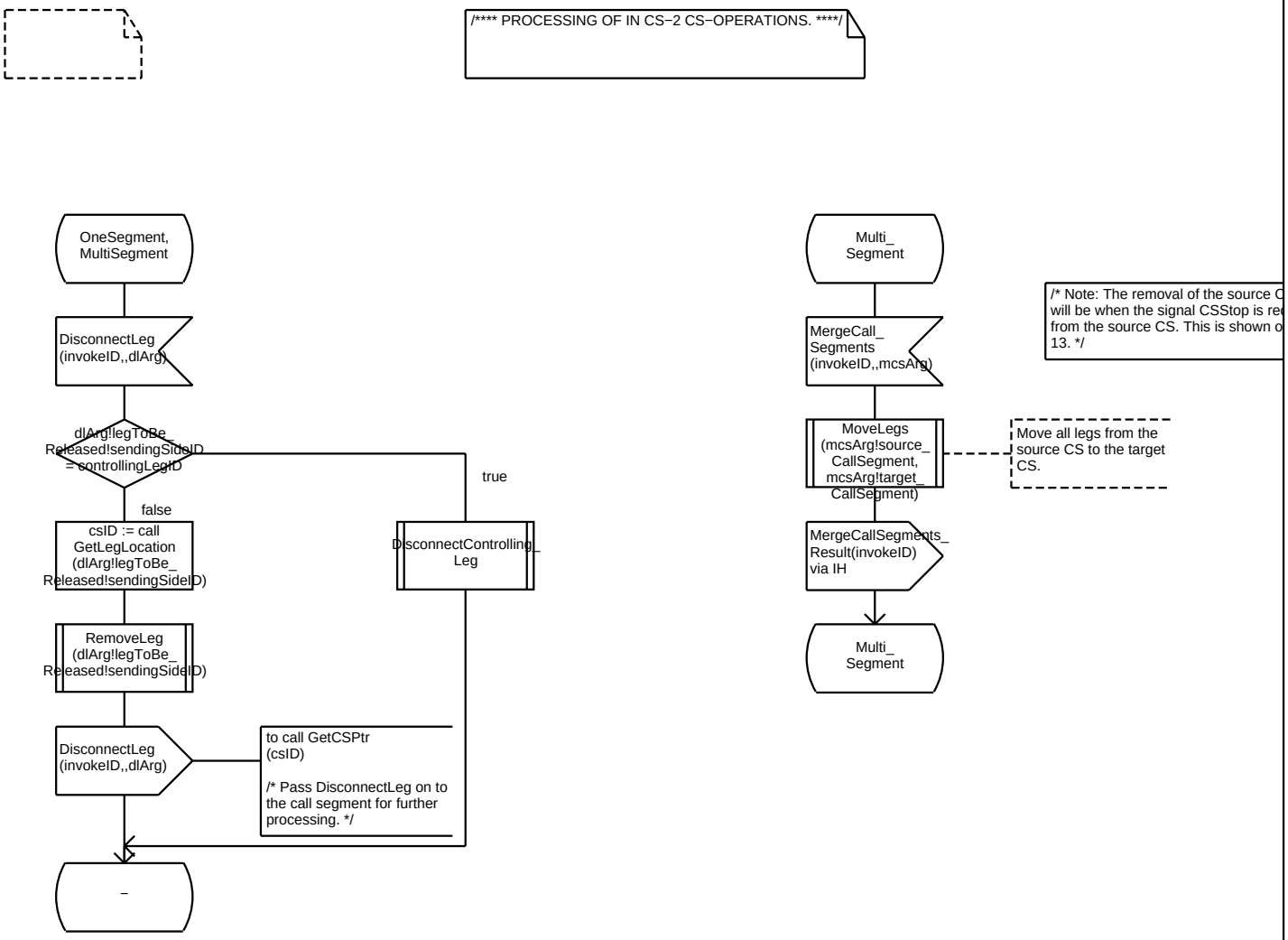

/**** EXTENSIONS/REDEFINITIONS OF IN CS−1 OPERATIONS ****/
OneSegment, MultiSegment
InitiateCall_ Attempt (invokeID,,icaArg)
From SCF.
call ExistLeg (icaArg! legToBeCreated!sendingSideID)
Does the leg already exist?
Yes
No
call ExistCS (icaArg! newCallSegment)
Does the call segment already exist?
True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

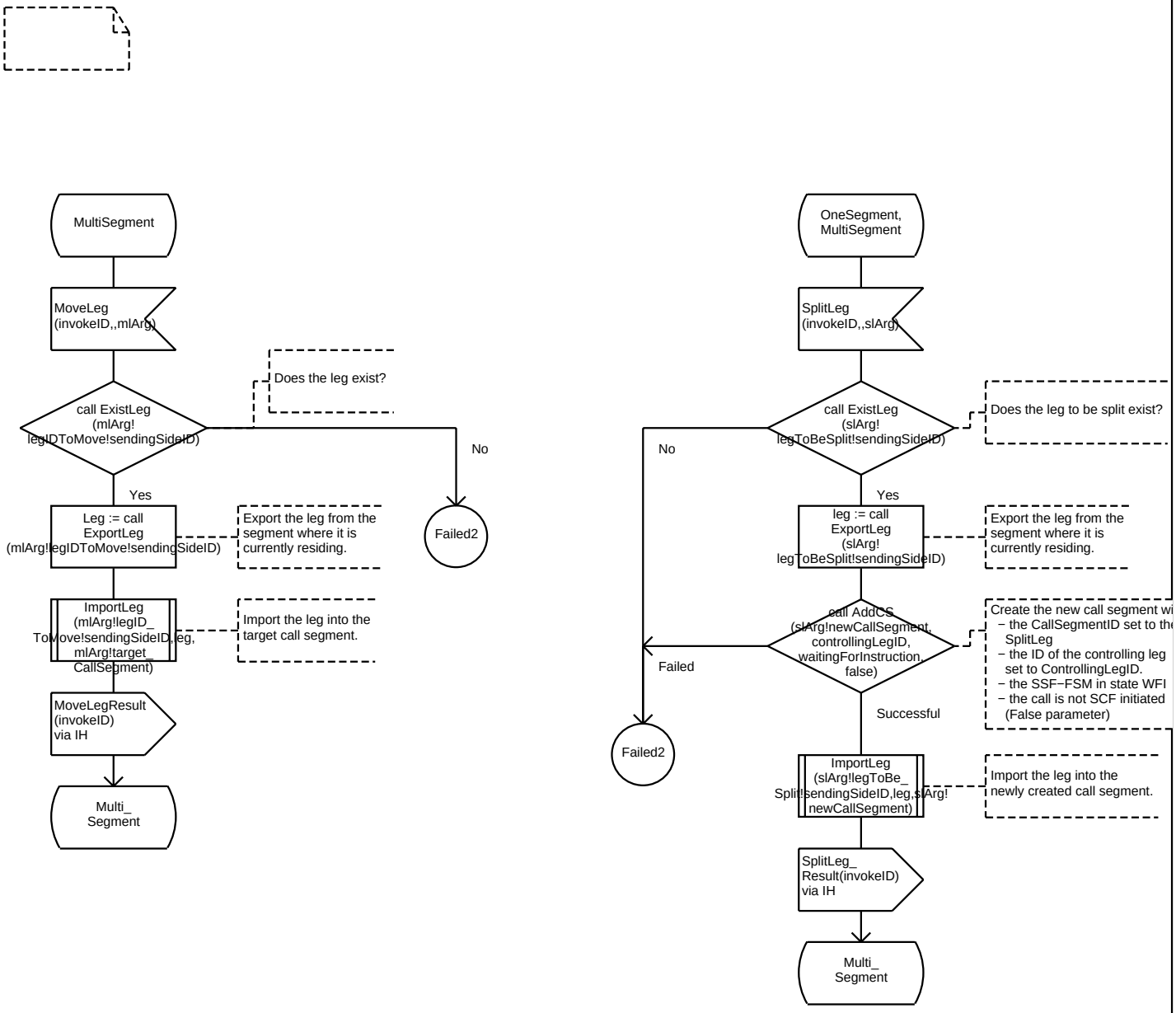

OneSegment,
MultiSegment
redefined
Connnect
(invokeID,,coArg)
The semantics of the Connect operation is changed in IN CS−2. This transition is therefore redefined.
legToBeCreatedPresent
(coArg)
false
true
legID := coArg!
legToBeCreated!sendingSideID
legID := leg2
callSegmentIDPresent
(coArg)
true
false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

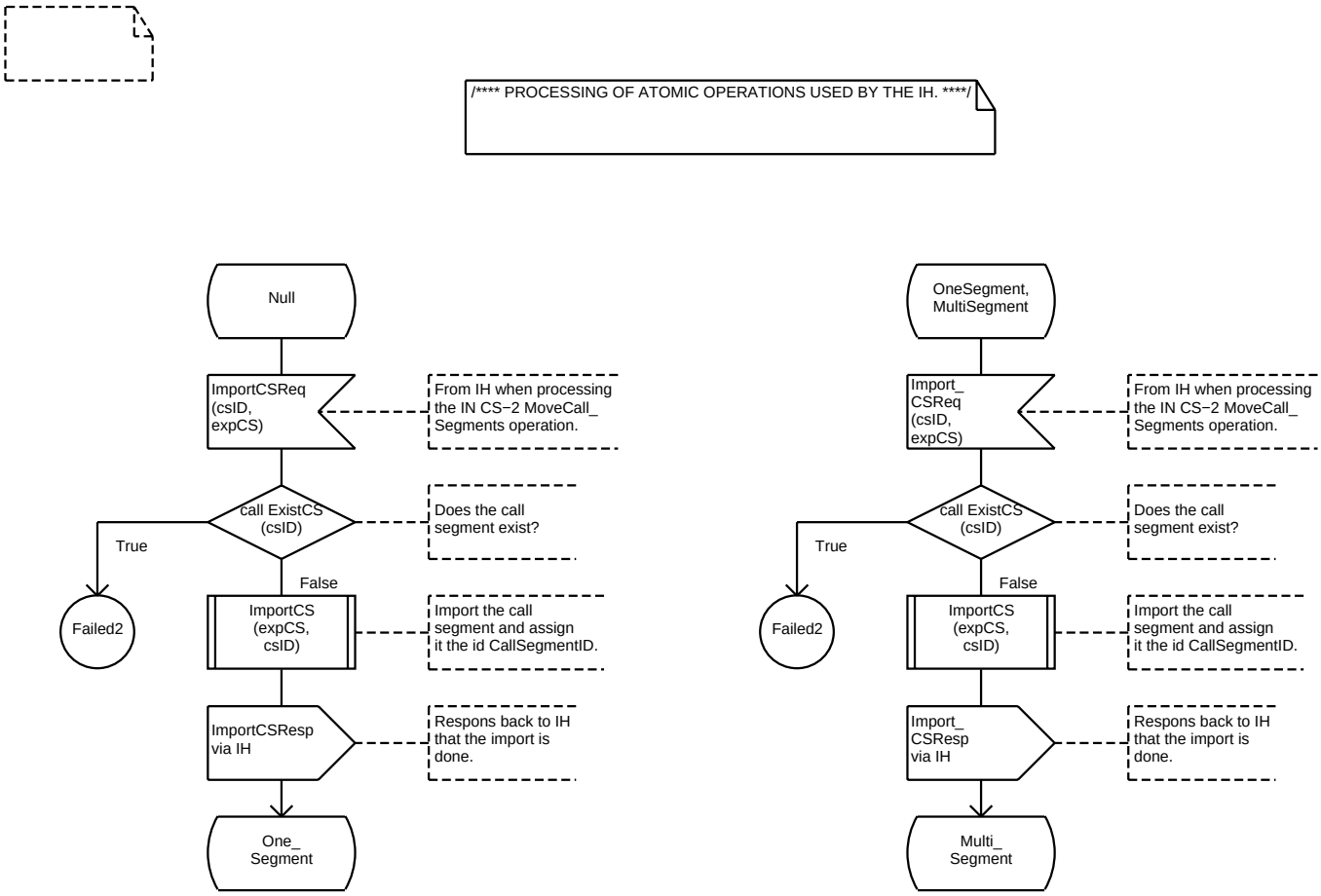

OneSegment,
MultiSegment
redefined
AnalyseInformation
(invokeID,,aiArg)
The semantics of the AI operation is changed in IN CS−2. This transition is therefore redefined.
legToBeCreatedPresent
(aiArg)
false
true
legID := aiArg!
legToBeCreated!sendingSideID
legID := leg2
callSegmentIDPresent
(aiArg)
true
false
csID := aiArg!
callSegmentID
csID := initial_
CallSegmentID
AI
AI
call ExistLeg
(legID)
Is AI applied on an existing leg or does AI implicitly create a new passive leg?
Yes
No
SetLegLocation
(legID,csID)
The leg to be created will reside in segment csID.
ProcessQueued_
RRBE(legID,csID)
Pass queued RRBEs for the new leg on to the call segment for further processing.
AnalyseInformation
(invokeID,,aiArg)
to call
GetCSPtr
(csID)
–

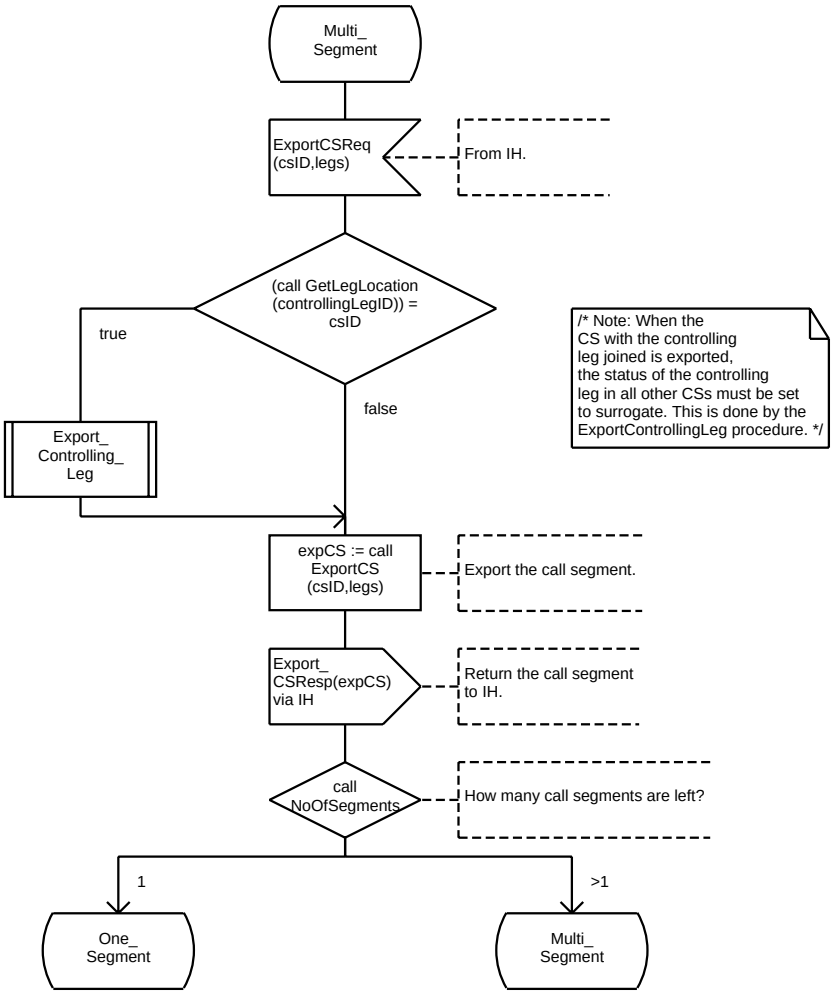

OneSegment,
MultiSegment
redefined
SelectRoute
(invokeID,,srArg)
The semantics of the SR operation is changed in IN CS−2. This transition is therefore redefined.
legToBeCreatedPresent
(srArg)
false
true
legID := srArg!
legToBeCreated!sendingSideID
legID := leg2
callSegmentIDPresent
(srArg)
true
false
csID := srArg!
callSegmentID
csID := initial_
CallSegmentID
SR
SR
call ExistLeg
(legID)
Is SR applied on an existing leg or does SR implicitly create a new passive leg?
Yes
No
SetLegLocation
(legID,csID)
The leg to be created will reside in segment csID.
ProcessQueued_
RRBE(legID,csID)
Pass queued RRBEs for the new leg on to the call segment for further processing.
SelectRoute
(invokeID,,srArg)
to call
GetCSPtr
(csID)
–

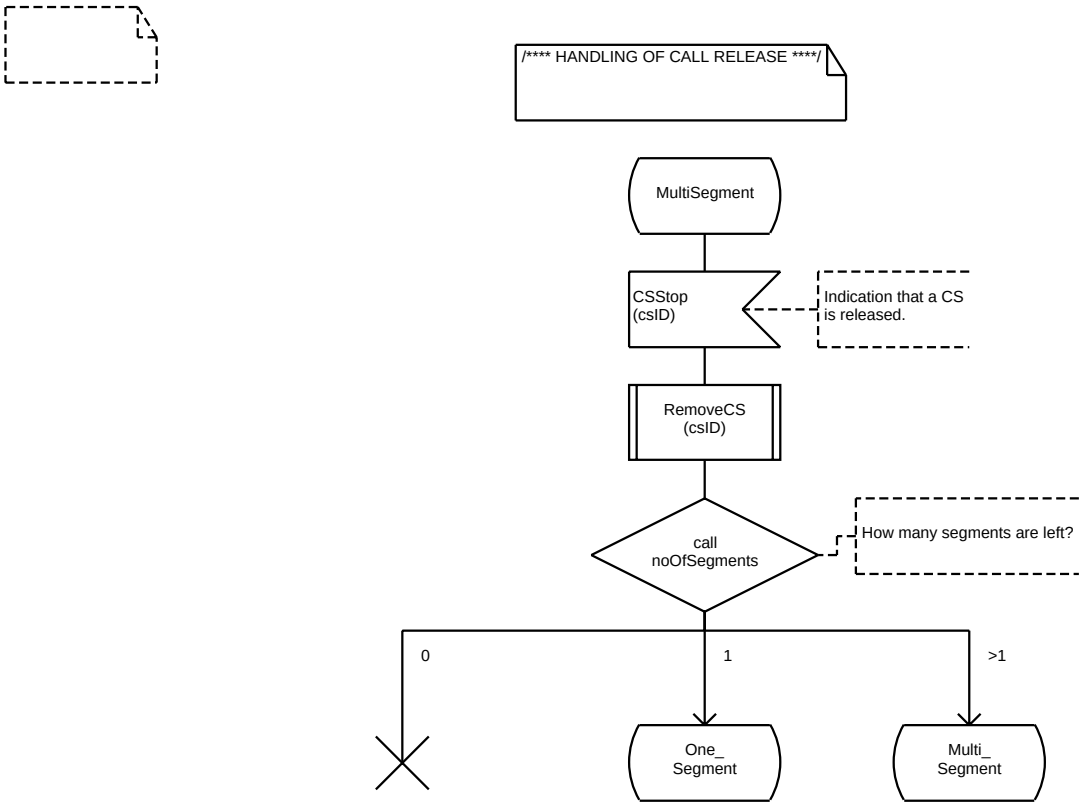

OneSegment
MultiSegment
OneSegment,
MultiSegment
redefined
ReleaseCall
(invokeID,,rcArg)
ReleaseCall
(invokeID,,rcArg)
redefined
RequestReport
BCSMEvent
(invokeID,,rrbcsmArg)
For IN CS−2 the SCF may attempt to
arm events for legs that does not
yet exist. Such RRBEs must therefore
be queued until the corresponding leg
is created (on Connect etc.).
ReleaseCall
(rcArg)
ProcessRRBE
(rrbcsmArg)
ProcessRRBE:
− passes RRBE on to the corresponding call segment
for further processing if the leg exist.
− queues the RRBE if the leg does not exist.
−

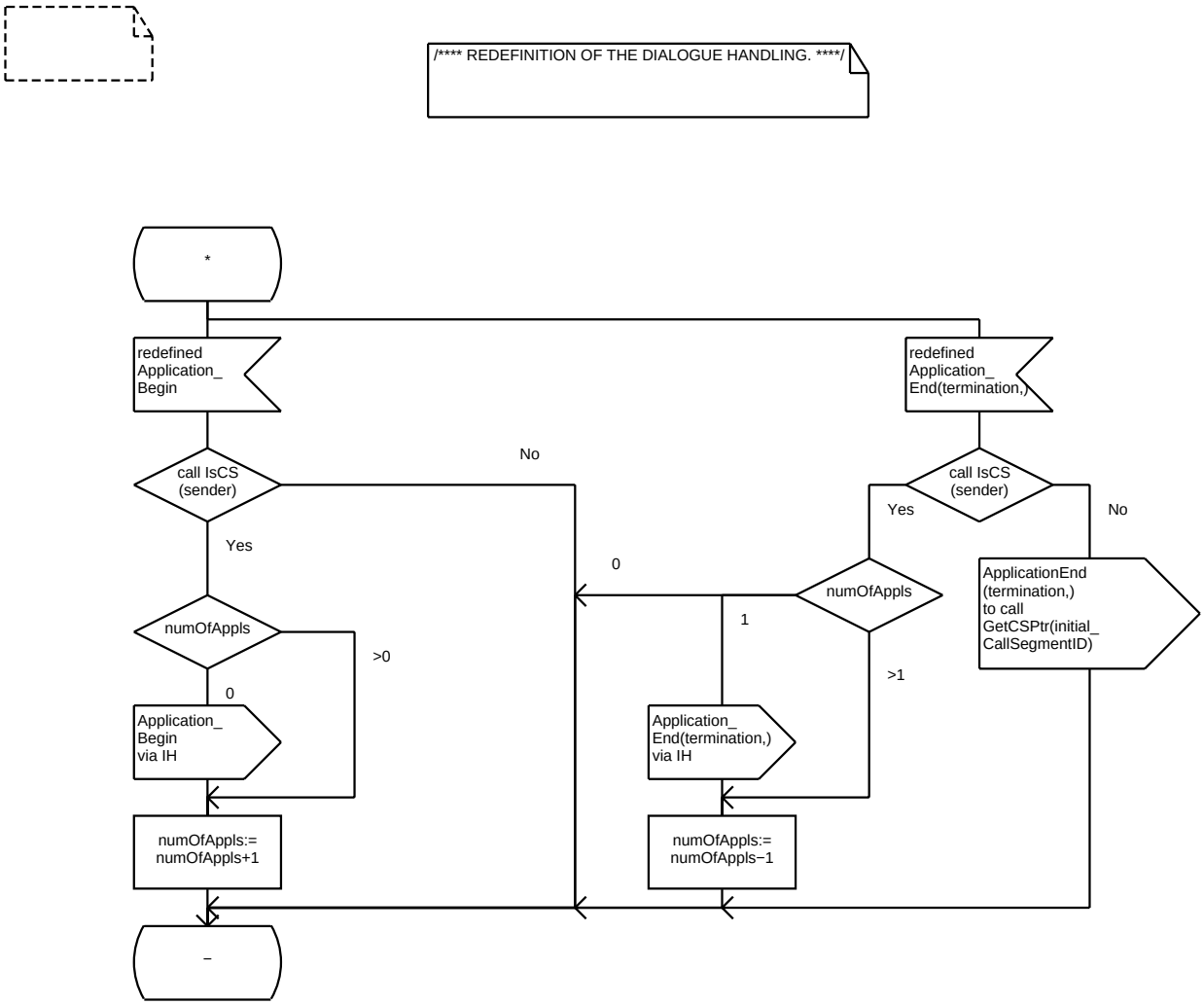

/**** PROCESSING OF IN CS−2 CS−OPERATIONS. ****/
OneSegment, MultiSegment
DisconnectLeg (invokeID,,dlArg)
dlArg!legToBe_ Released!sendingSideID = controllingLegID
true
false
csID := call GetLegLocation (dlArg!legToBe_ Released!sendingSideID)
DisconnectControlling_ Leg
RemoveLeg (dlArg!legToBe_ Released!sendingSideID)
DisconnectLeg (invokeID,,dlArg)
to call GetCSPtr (csID)
/* Pass DisconnectLeg on to the call segment for further processing. */
Multi_ Segment
MergeCall_ Segments (invokeID,,mcsArg)
/* Note: The removal of the source C
will be when the signal CSStop is re
from the source CS. This is shown o
13. */
MoveLegs (mcsArg!source_ CallSegment, mcsArg!target_ CallSegment)
Move all legs from the source CS to the target CS.
MergeCallSegments_ Result(invokeID) via IH
Multi_ Segment

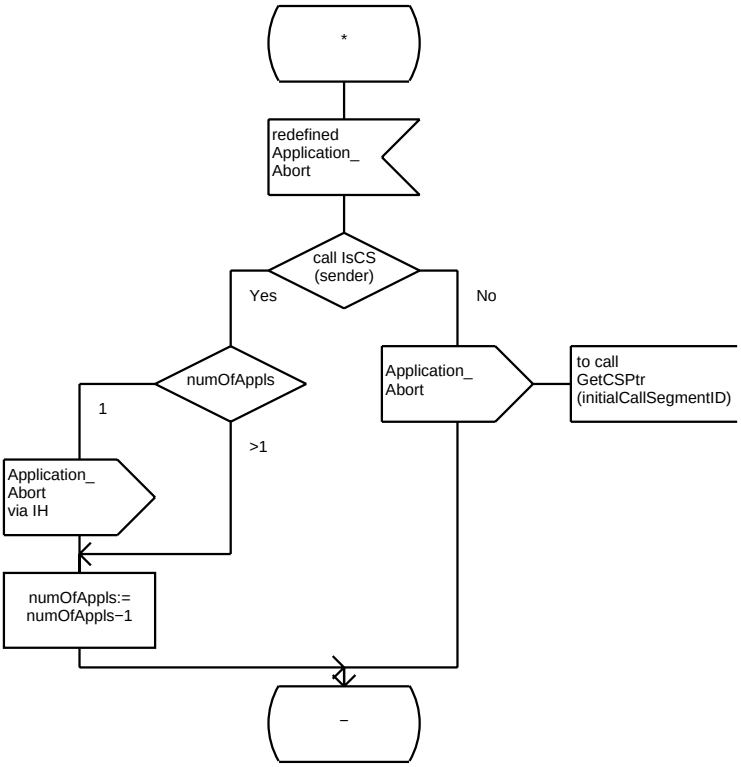

MultiSegment
MoveLeg (invokeID,,mlArg)
call ExistLeg (mlArg! legIDToMove!sendingSideID)
Does the leg exist?
No
Yes
Failed2
Leg := call ExportLeg (mlArg!legIDToMove!sendingSideID)
Export the leg from the segment where it is currently residing.
ImportLeg (mlArg!legID_ ToMove!sendingSideID,leg, mlArg!target_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

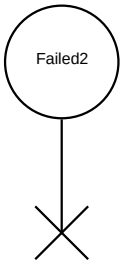

/**** PROCESSING OF ATOMIC OPERATIONS USED BY THE IH. ****/
Null
ImportCSReq (csID, expCS)
From IH when processing the IN CS−2 MoveCall_ Segments operation.
call ExistCS (csID)
Does the call segment exist?
True
False
Failed2
ImportCS (expCS, csID)
Import the call segment and assign it the id CallSegmentID.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

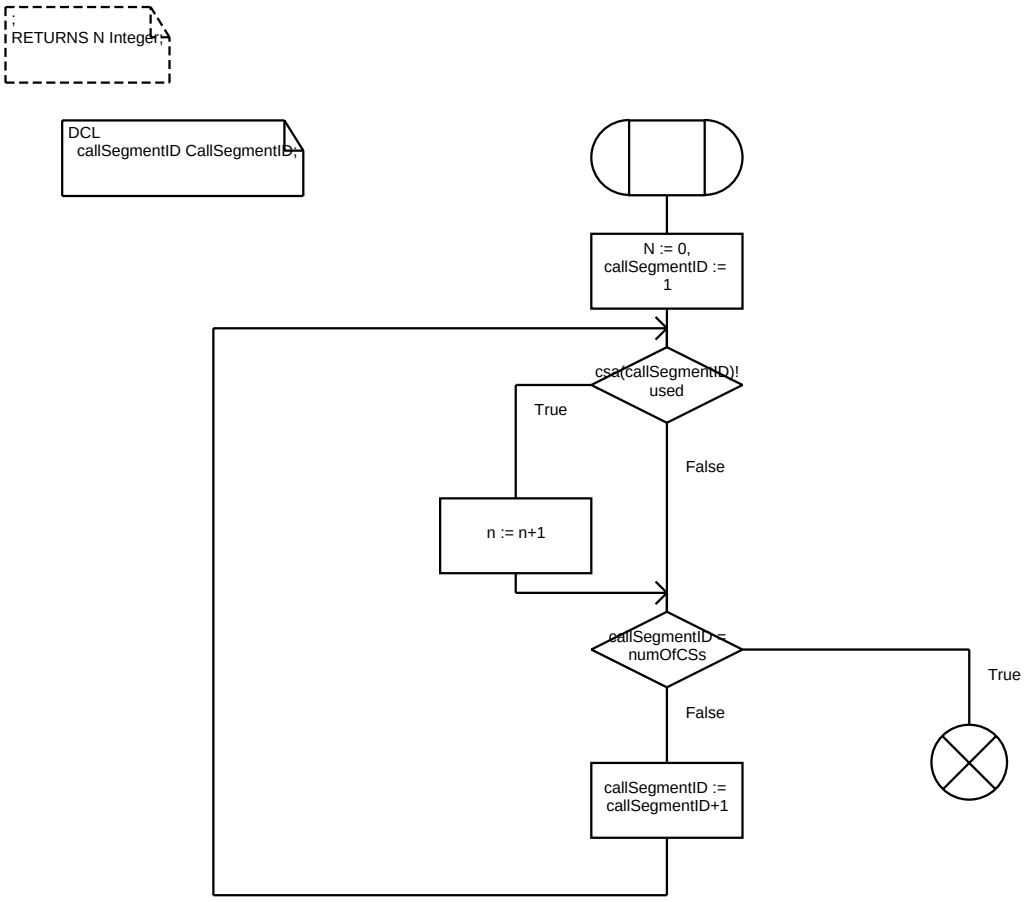

Multi_
Segment
ExportCSReq
(csID,legs)
From IH.
(call GetLegLocation
(controllingLegID)) =
csID
true
false
/* Note: When the
CS with the controlling
leg joined is exported,
the status of the controlling
leg in all other CSs must be set
to surrogate. This is done by the
ExportControllingLeg procedure. */
Export_
Controlling_
Leg
expCS := call
ExportCS
(csID,legs)
Export the call segment.
Export_
CSResp(expCS)
via IH
Return the call segment
to IH.
call
NoOfSegments
How many call segments are left?
1
>1
One_
Segment
Multi_
Segment


/**** HANDLING OF CALL RELEASE ****/
MultiSegment
CSStop
(csID)
Indication that a CS
is released.
RemoveCS
(csID)
call
noOfSegments
How many segments are left?
0
1
>1
One_
Segment
Multi_
Segment


/**** REDEFINITION OF THE DIALOGUE HANDLING. ****/
*
redefined Application_ Begin
redefined Application_ End(termination,)
call IsCS (sender)
No
Yes
numOfAppls
>0
0
Application_ Begin via IH
numOfAppls:= numOfAppls+1
call IsCS (sender)
Yes
No
0
numOfAppls
1
>1
Application_ End(termination,) via IH
numOfAppls:= numOfAppls−1
ApplicationEnd (termination,) to call GetCSPtr(initial_ CallSegmentID)
–

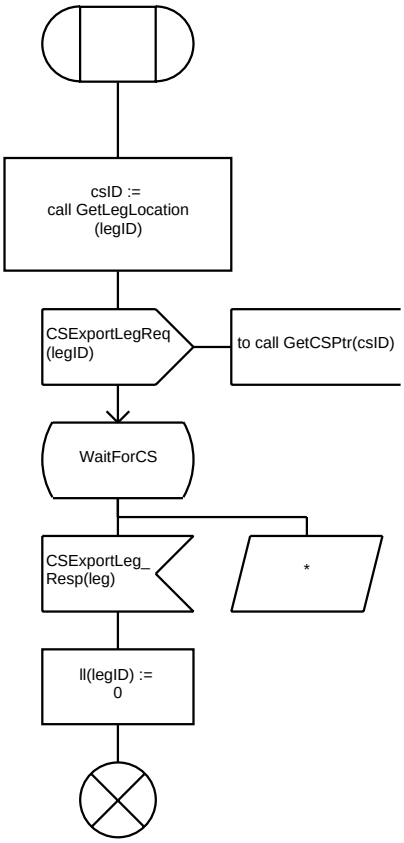

*
redefined
Application_
Abort
call IsCS
(sender)
Yes
No
numOfAppls
1
>1
Application_
Abort
via IH
numOfAppls:=
numOfAppls−1
Application_
Abort
to call
GetCSPtr
(initialCallSegmentID)
–

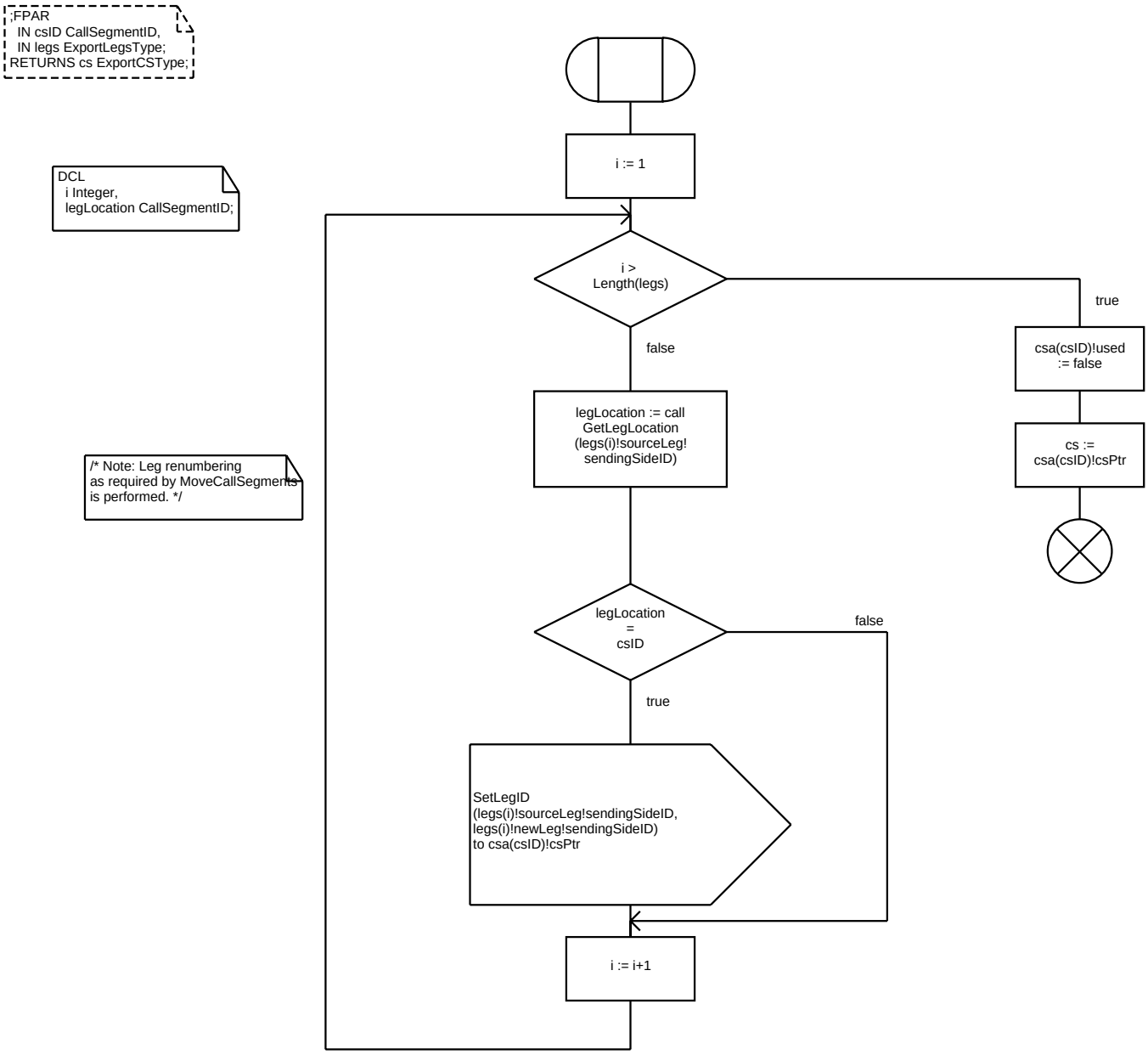

Failed2


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

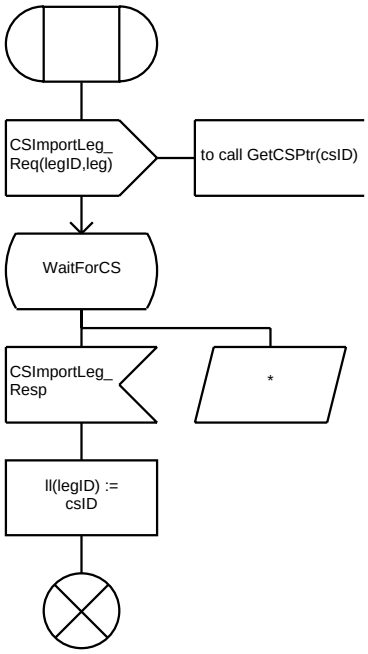
```
;FPAR
  IN legId LegType;
RETURNS leg ExportLegType;
```

```
DCL
  csID CallSegmentID;
```

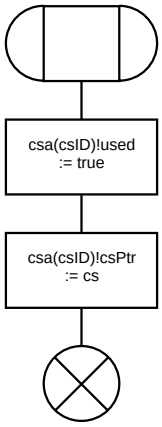



;FPAR
IN csID CallSegmentID,
IN legs ExportLegsType;
RETURNS cs ExportCSType;
DCL
i Integer,
legLocation CallSegmentID;
/* Note: Leg renumbering
as required by MoveCallSegments
is performed. */
i := 1
i >
Length(legs)
true
false
csa(csID)!used
:= false
cs :=
csa(csID)!csPtr
legLocation := call
GetLegLocation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

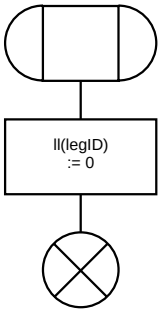
```
;FPAR
  IN legID LegType,
  IN leg ExportLegType,
  IN csID CallSegmentID;
```



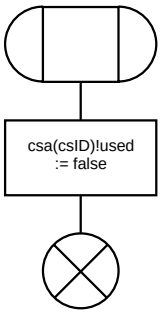
```
;FPAR
  IN cs ExportCSType,
  IN csID CallSegmentID;
```



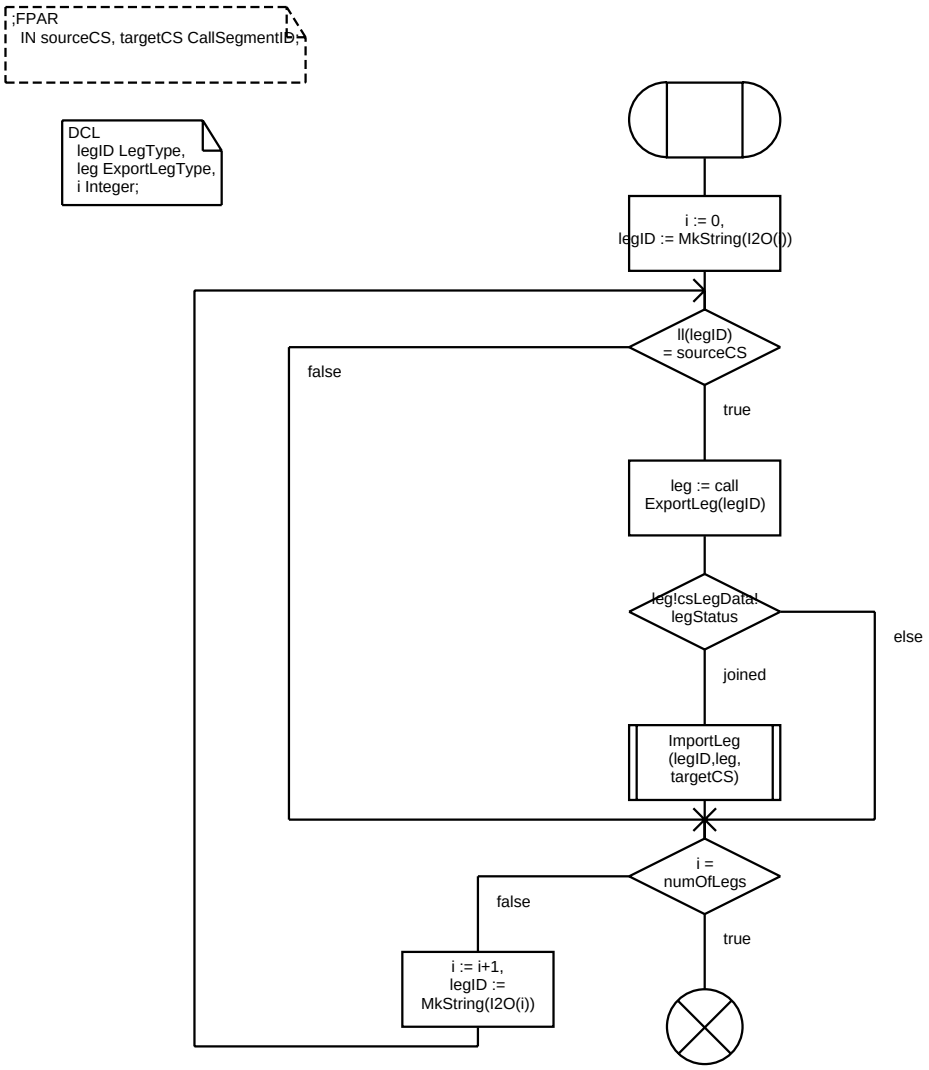
;FPAR
IN legId LegType;

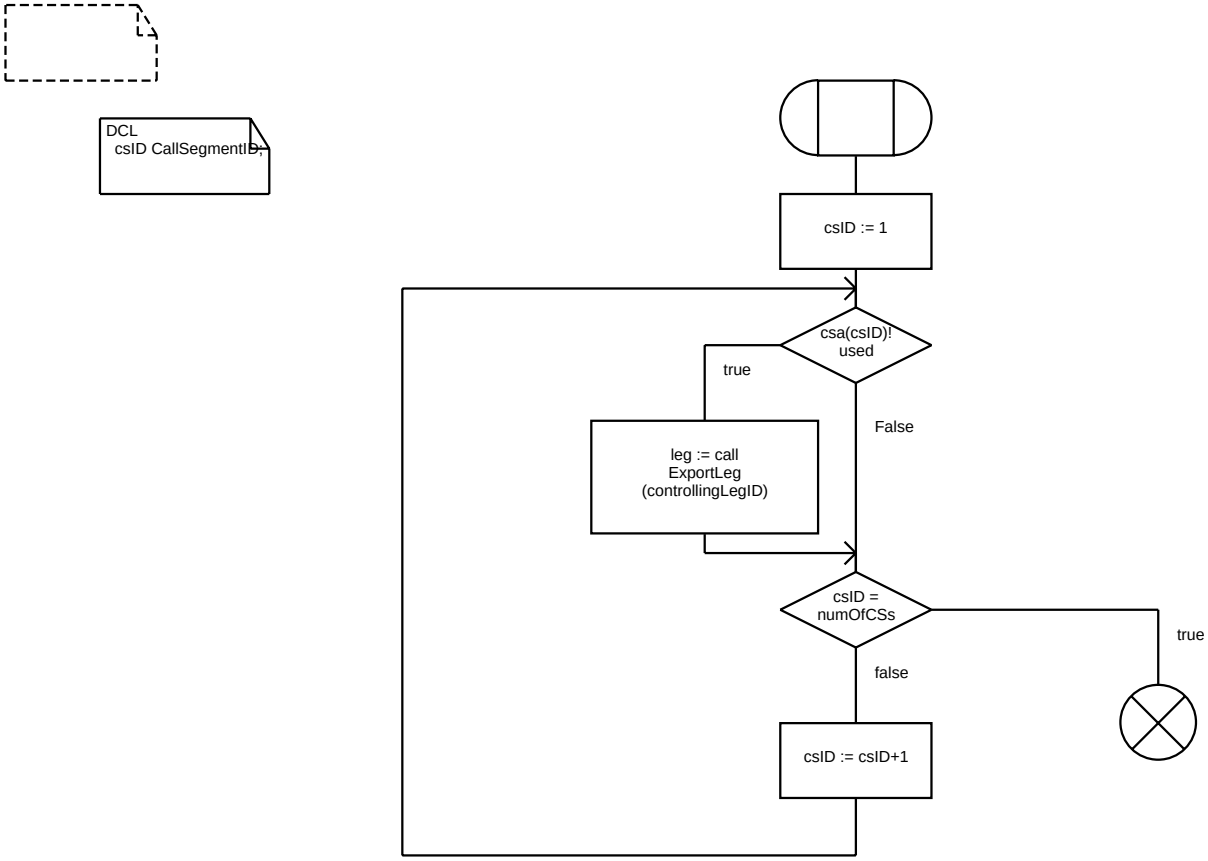


;FPAR
IN csID CallSegmentID;

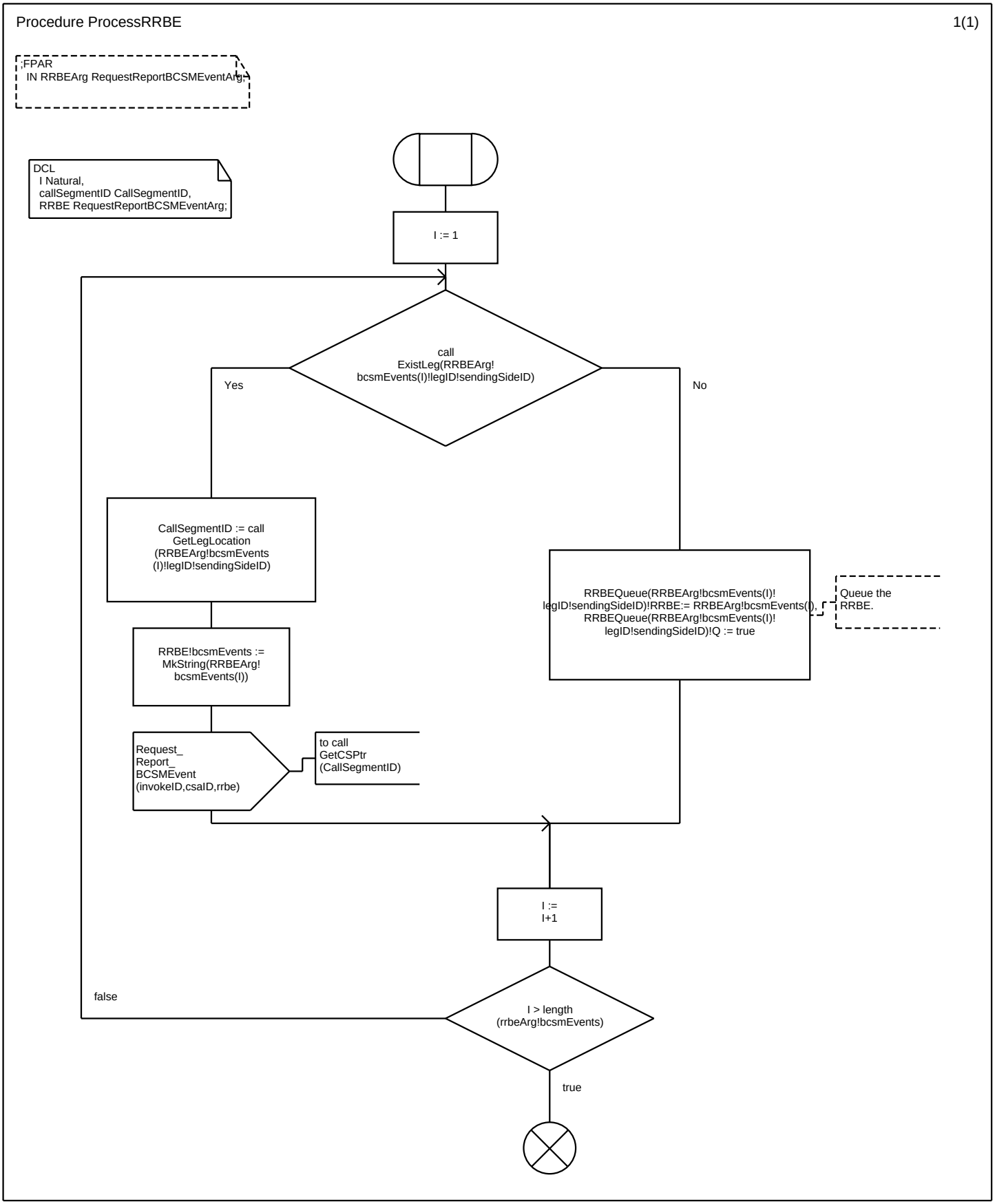


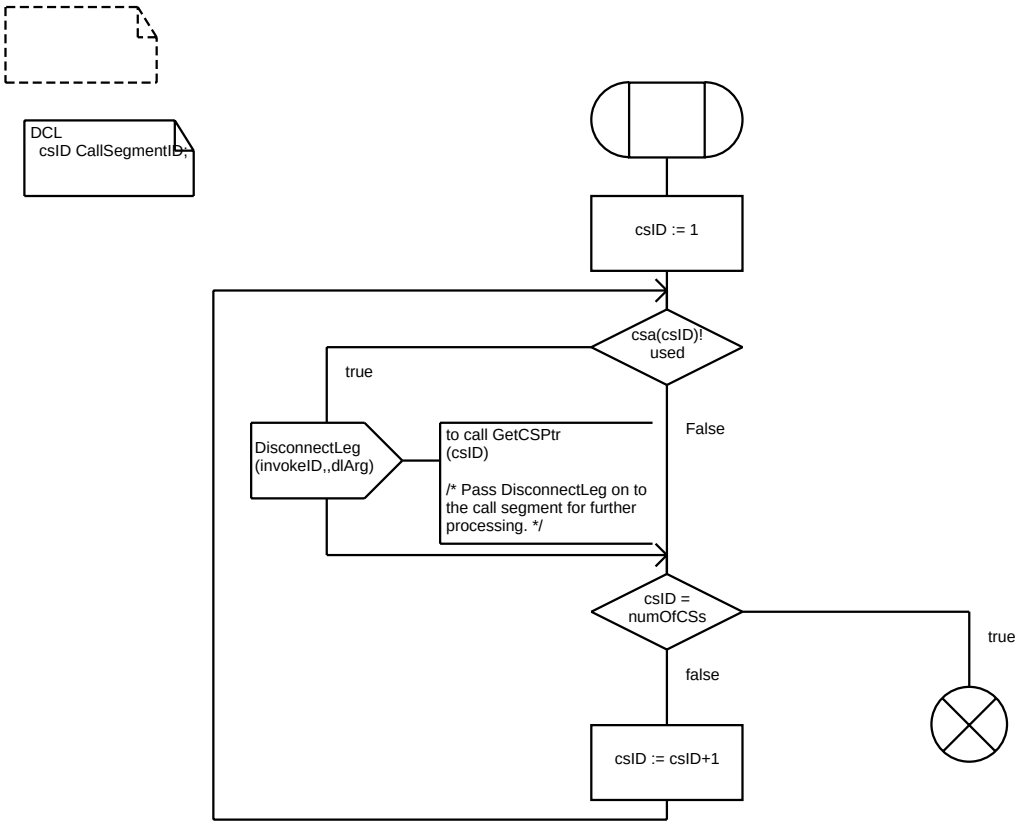
Procedure MoveLegs 1(1)




DCL
csID CallSegmentID;
csID := 1
csa(csID)!
used
true
False
leg := call
ExportLeg
(controllingLegID)
csID =
numOfCSs
true
false
csID := csID+1

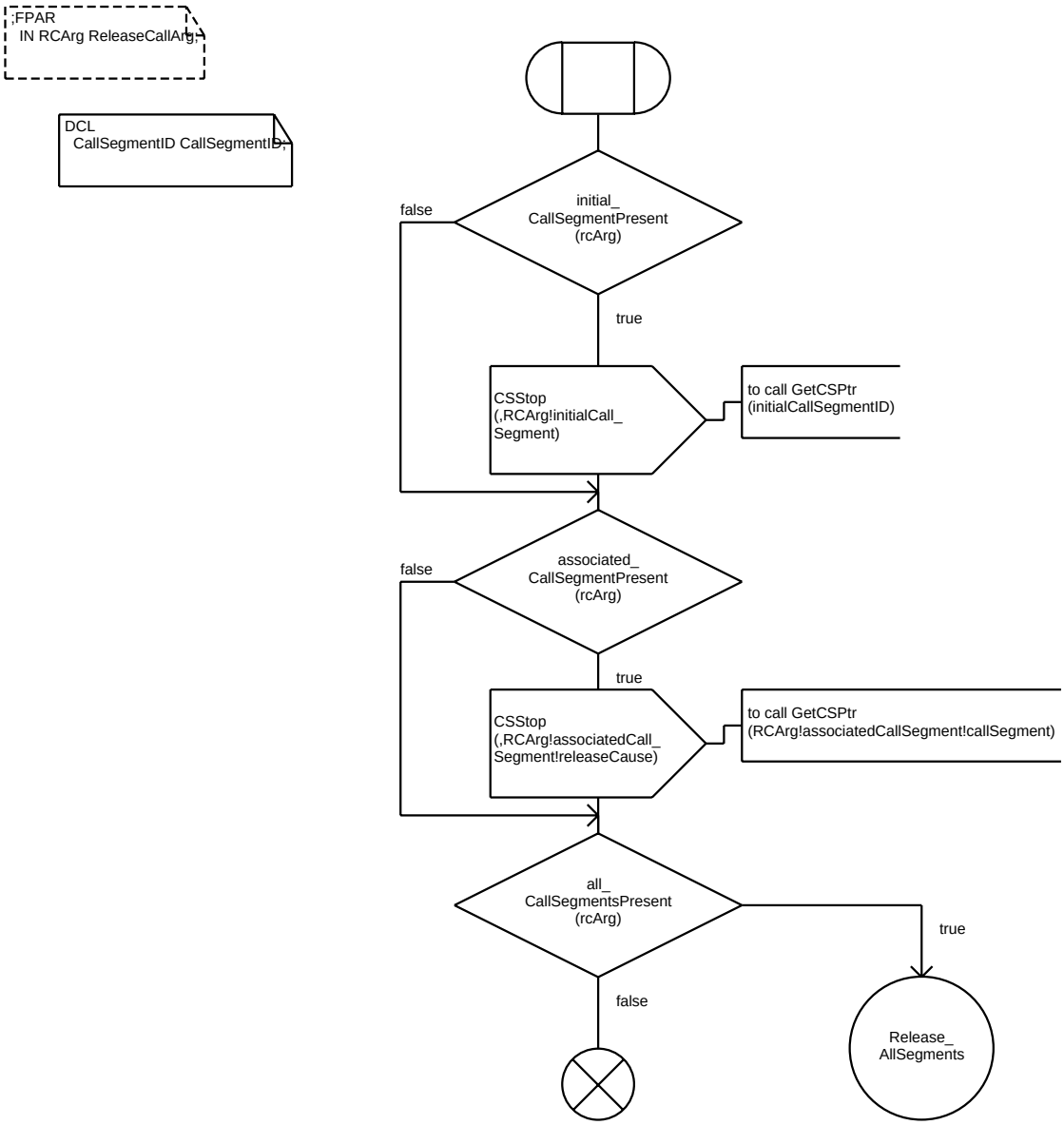
Procedure ProcessRRBE 1(1)



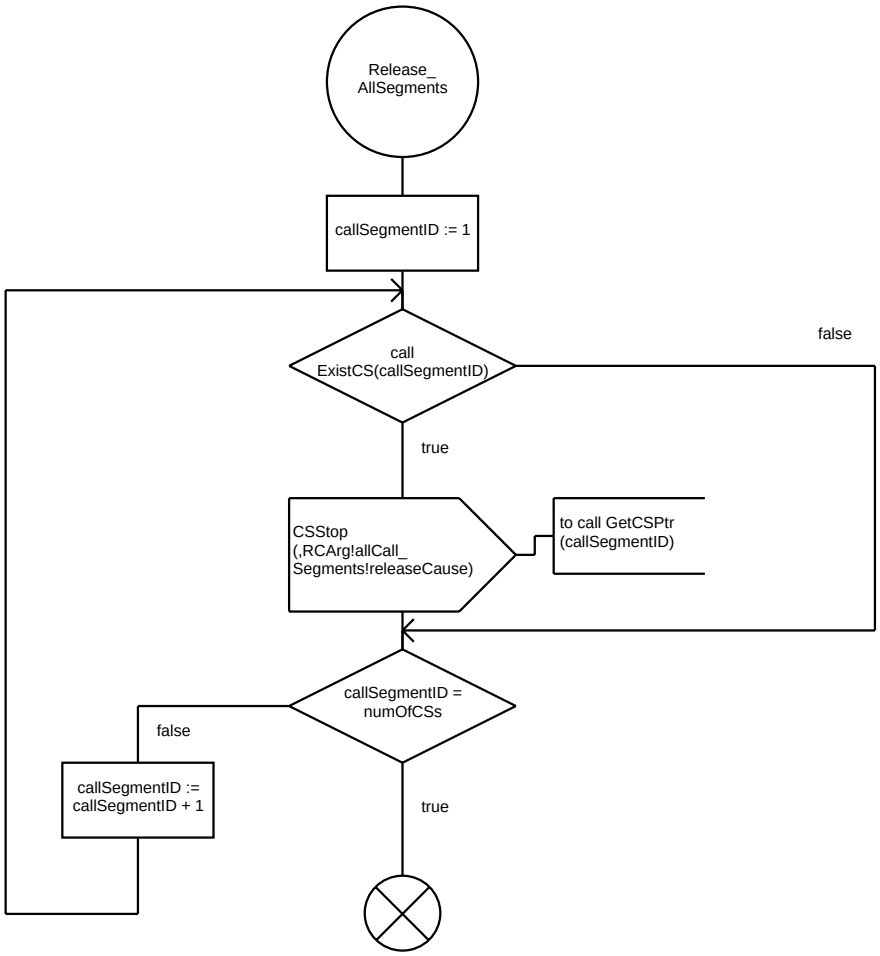

DCL
csID CallSegmentID;
csID := 1
csa(csID)!
used
true
False
DisconnectLeg
(invokeID,,dlArg)
to call GetCSPtr
(csID)
/* Pass DisconnectLeg on to
the call segment for further
processing. */
csID =
numOfCSs
true
false
csID := csID+1

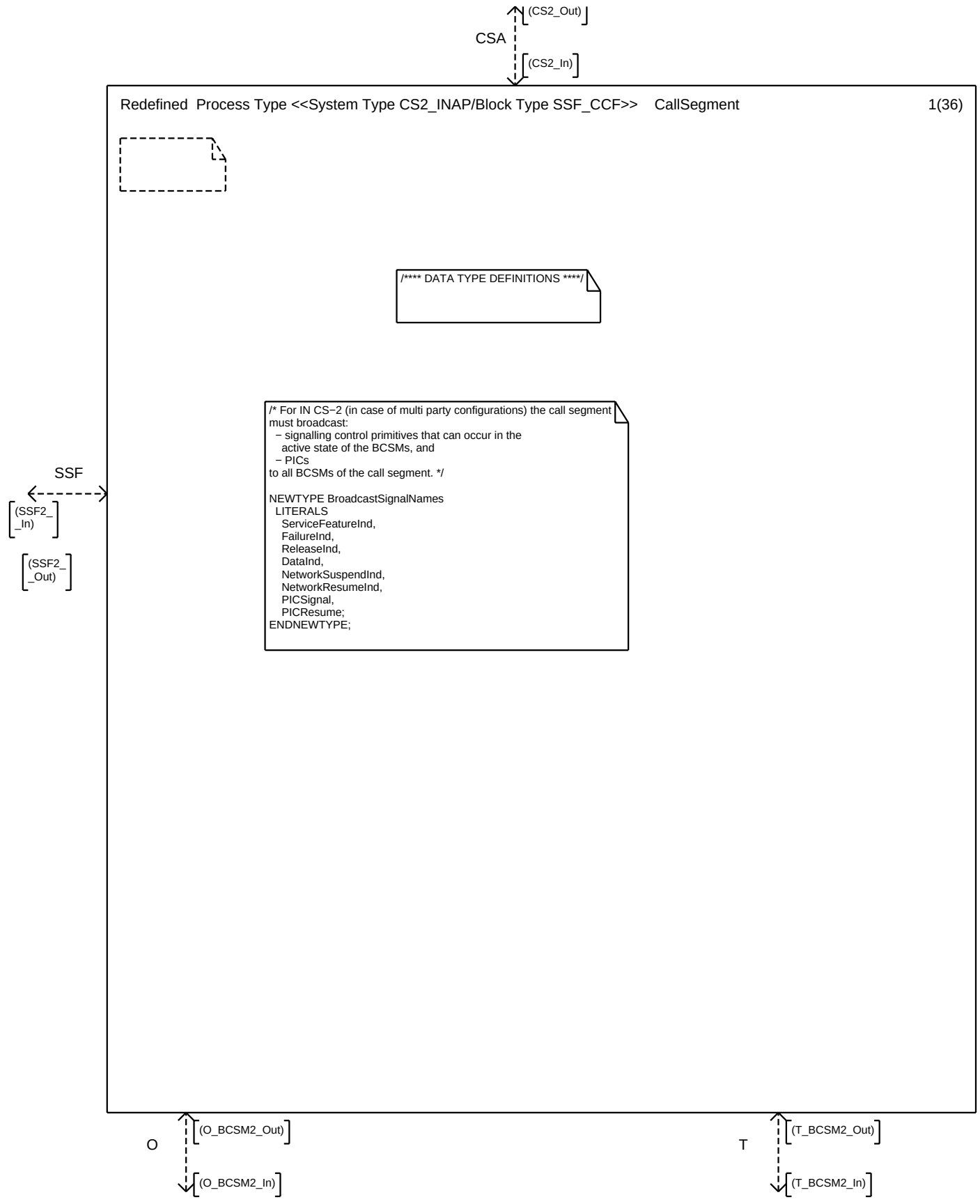
;FPAR
IN legID LegType,
IN csID CallSegmentID;

DCL
rrbeArg RequestReportBCSMEventArg;


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

;FPAR
IN RCArg ReleaseCallArg;



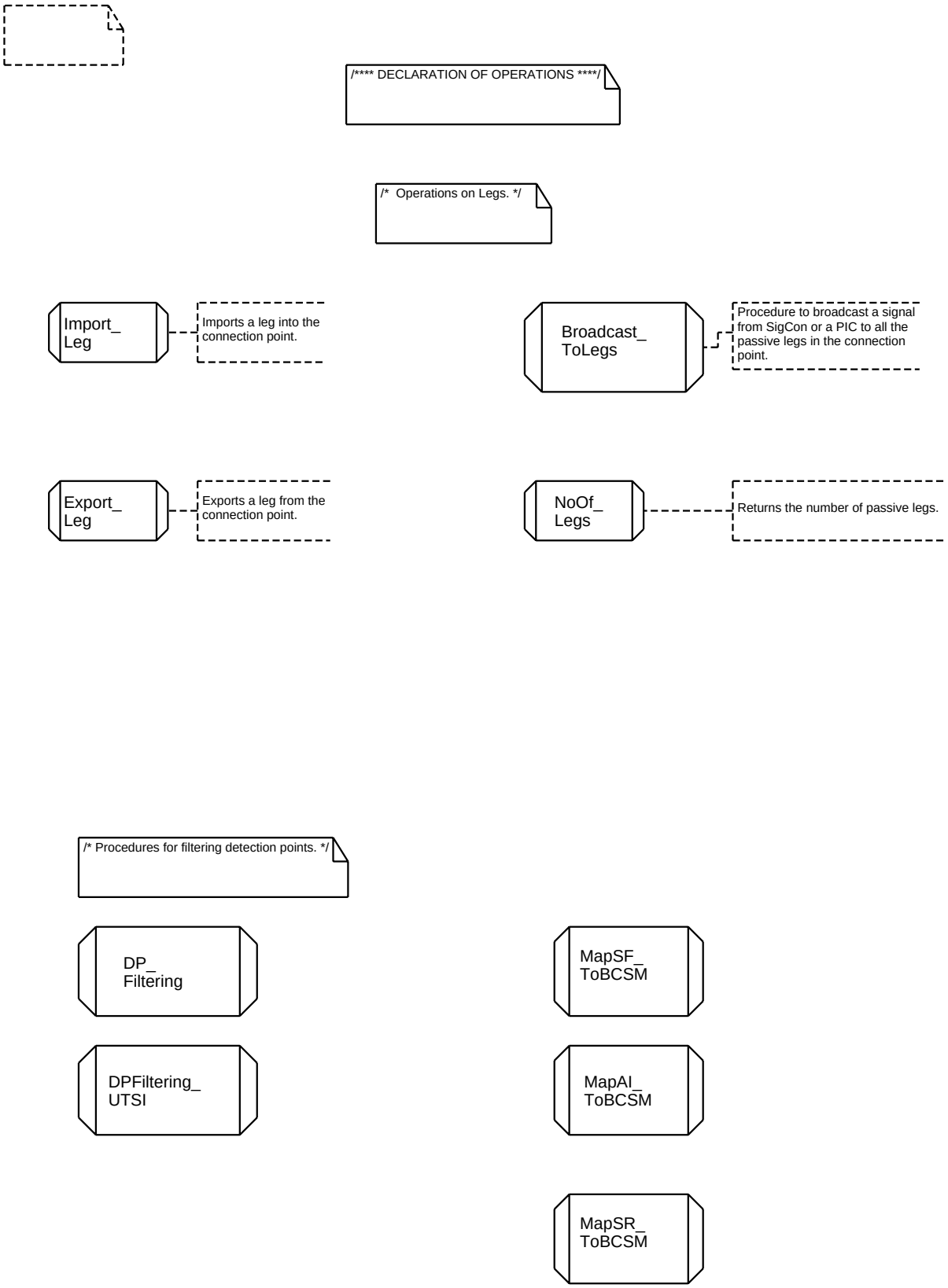

(CS2_Out)
CSA
(CS2_In)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> CallSegment 1(36)
/**** DATA TYPE DEFINITIONS ****/
/* For IN CS−2 (in case of multi party configurations) the call segment must broadcast:
− signalling control primitives that can occur in the active state of the BCSMs, and
− PICs
to all BCSMs of the call segment. */
NEWTYPE BroadcastSignalNames
LITERALS
ServiceFeatureInd,
FailureInd,
ReleaseInd,
DataInd,
NetworkSuspendInd,
NetworkResumeInd,
PICSignal,
PICResume;
ENDNEWTYPE;
SSF
(SSF2_ _In)
(SSF2_ _Out)
O
(O_BCSM2_Out)
(O_BCSM2_In)
T
(T_BCSM2_Out)
(T_BCSM2_In)

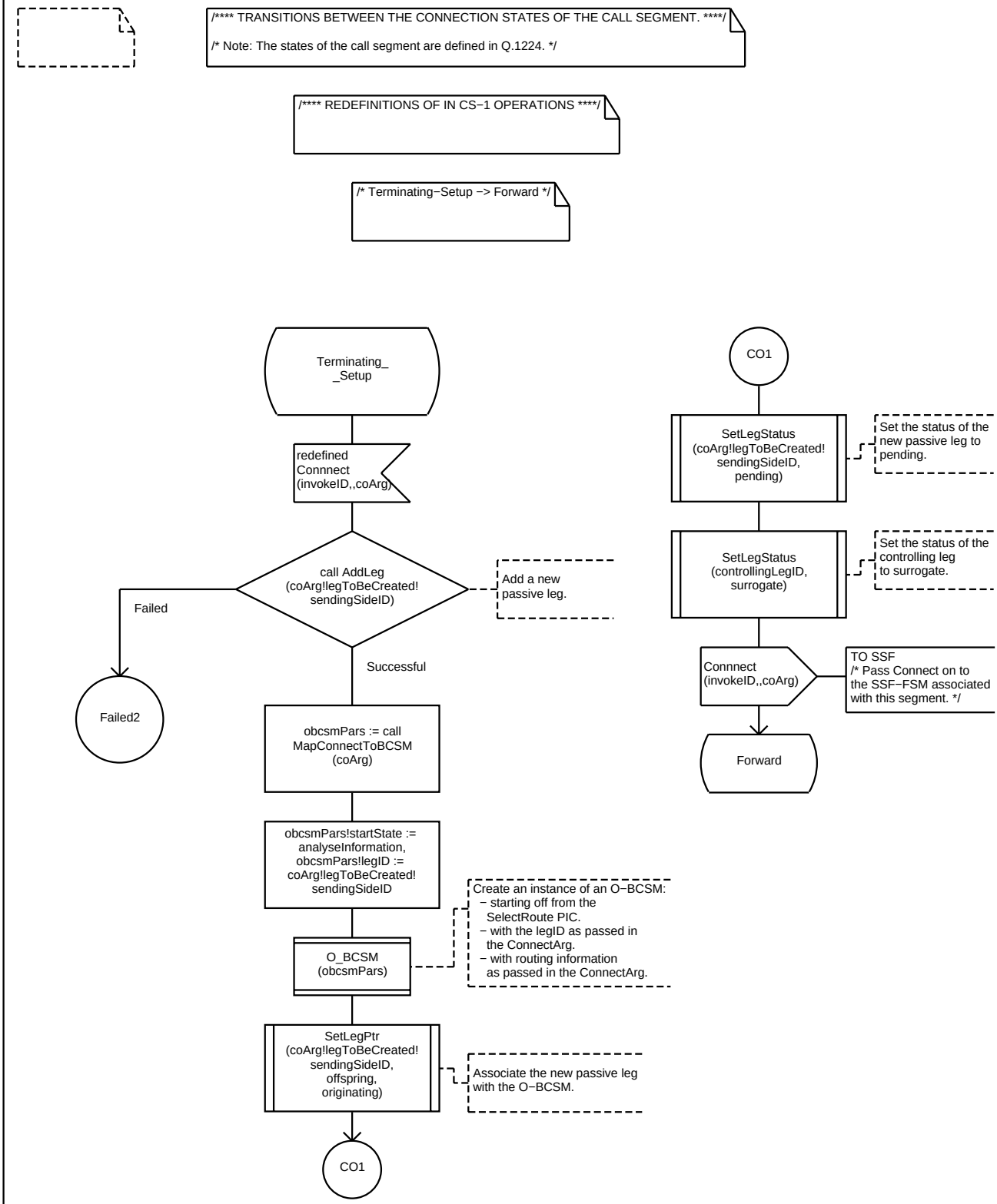
/**** VARIABLE DECLARATIONS ****/

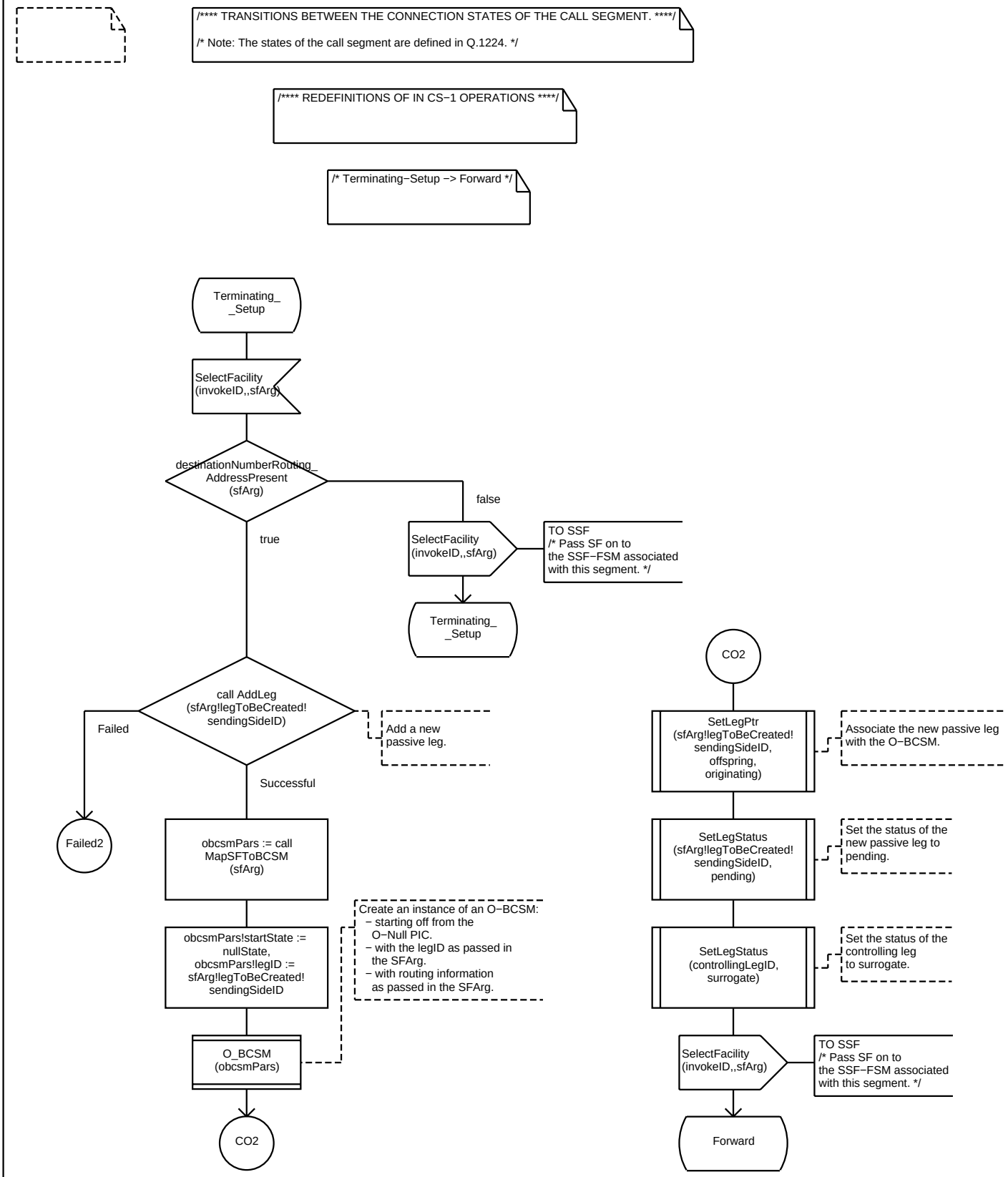
```
DCL
 /* IN CS−2 operation arguments. */
 ctwaArg ContinueWithArgumentArg,
 dfcwaArg DisconnectForwardConnectionWithArgumentArg,
 dlArg DisconnectLegArg,
 erArg EntityReleasedArg,
 rutsiArg ReportUTSIArg,
 rrutsiArg RequestReportUTSIArg,
 sstuiArg SendSTUIArg;
```

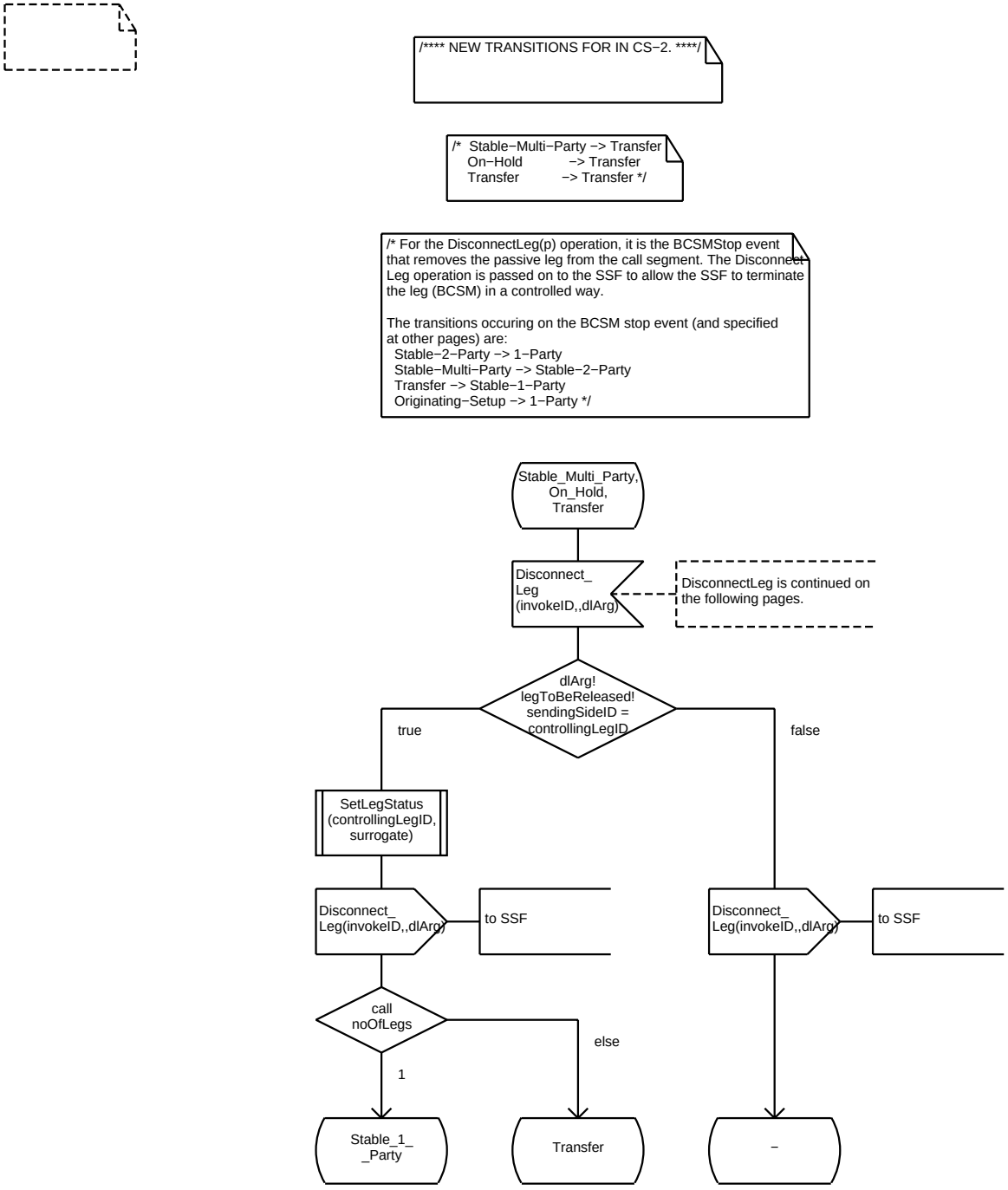
```
DCL
 /* Parameters at the signalling control interface. */
 udArg UserDataType,
 nsrArg NetworkSRType;
```

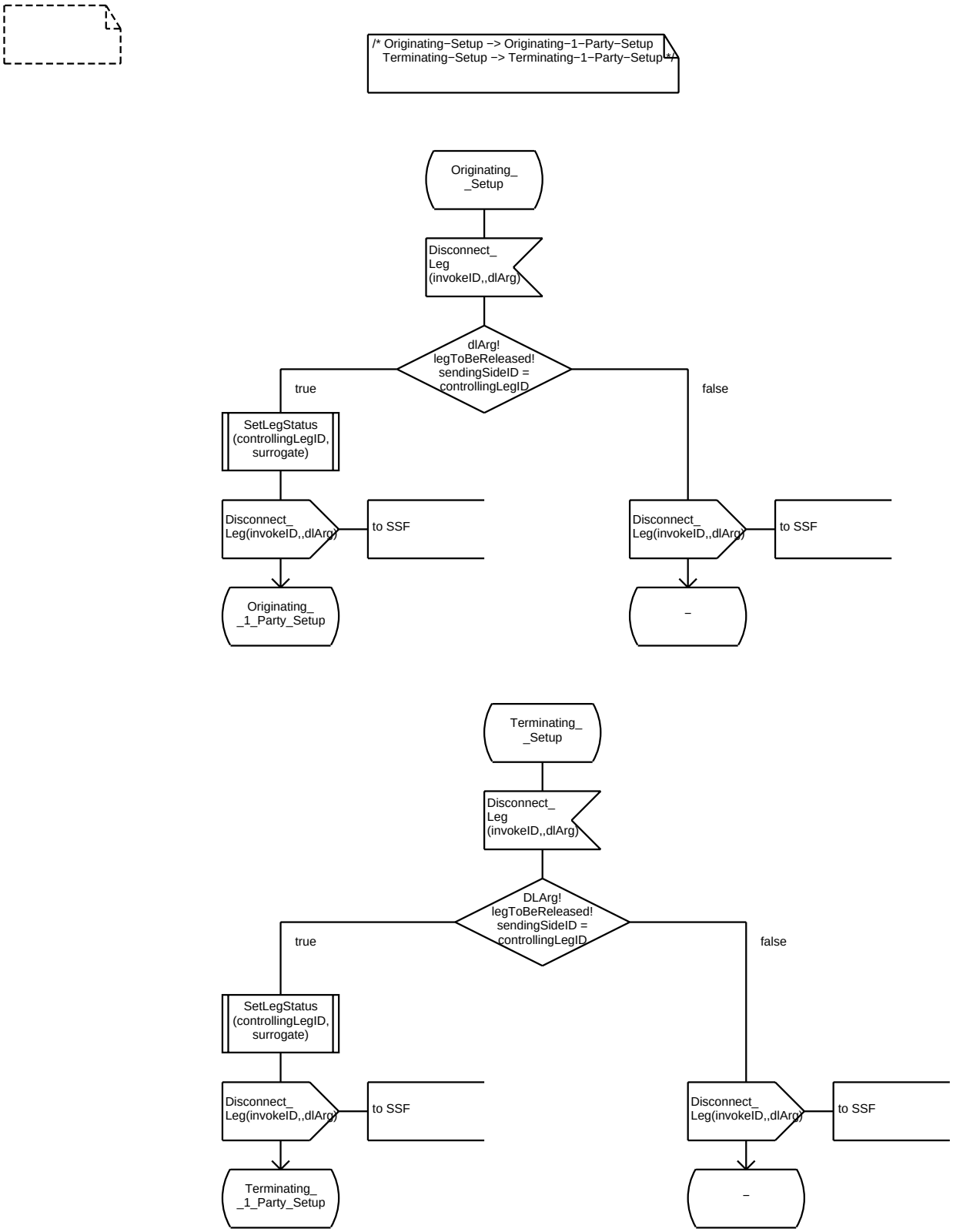
```
DCL
 /* Other variables. */
 currentLegID LegType,
 expLeg ExportLegType,
 dpUTSIArg DPUTSIArg,
 noOfMidCallDPs Natural := 0,
 noOfDisconnectDPs Natural := 0,
 noOfAbandonDPs Natural := 0,
 noOfUTSIDPs Natural := 0,
 party PartyType,
 cwaFlag Boolean := false;
```

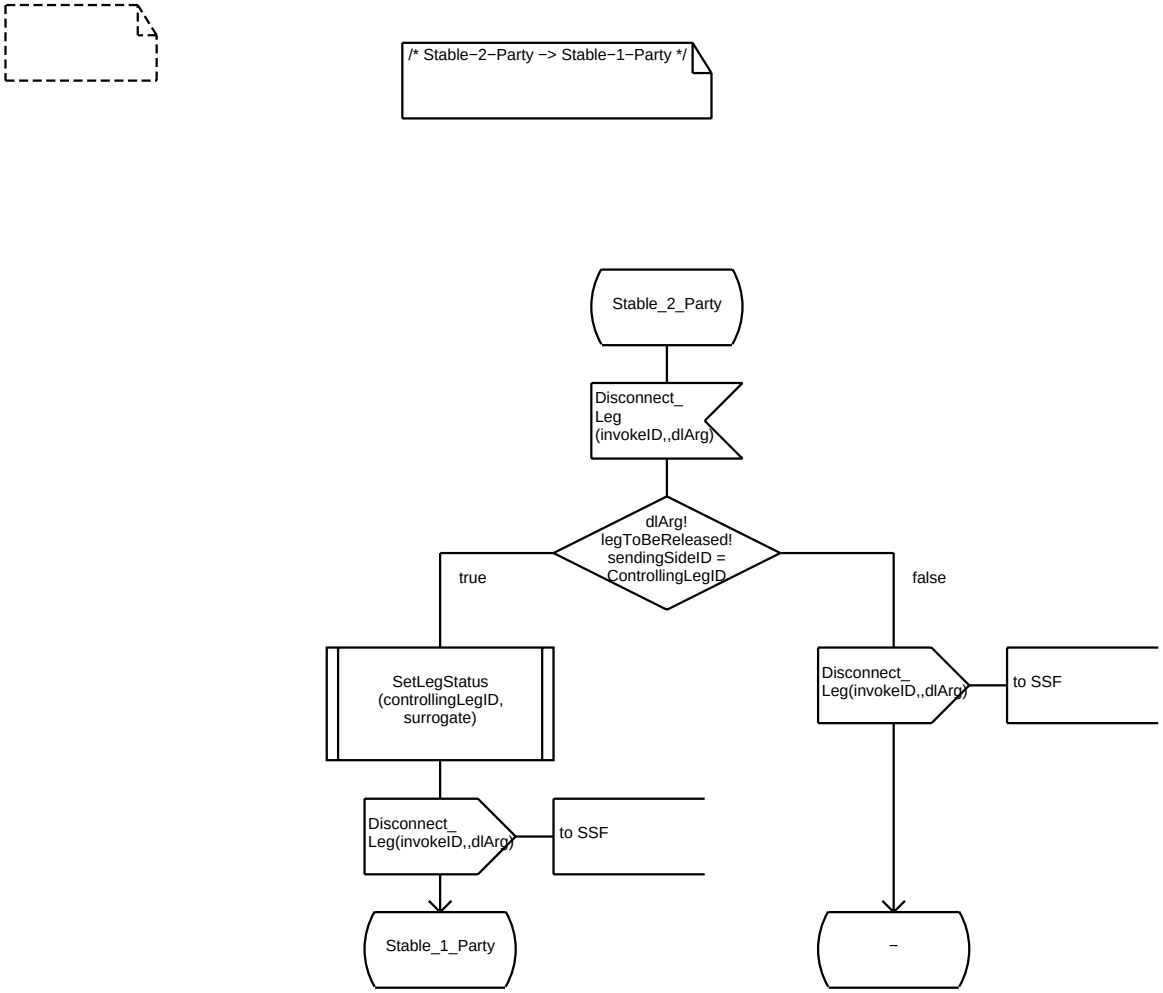

/**** DECLARATION OF OPERATIONS ****/
/* Operations on Legs. */
Import_ Leg
Imports a leg into the connection point.
Broadcast_ ToLegs
Procedure to broadcast a signal from SigCon or a PIC to all the passive legs in the connection point.
Export_ Leg
Exports a leg from the connection point.
NoOf_ Legs
Returns the number of passive legs.
/* Procedures for filtering detection points. */
DP_ Filtering
DPFiltering_ UTSI
MapSF_ ToBCSM
MapAI_ ToBCSM
MapSR_ ToBCSM


/**** TRANSITIONS BETWEEN THE CONNECTION STATES OF THE CALL SEGMENT. ****/
/* Note: The states of the call segment are defined in Q.1224. */
/**** REDEFINITIONS OF IN CS−1 OPERATIONS ****/
/* Terminating−Setup −> Forward */
Terminating_ _Setup
redefined Connnect (invokeID,,coArg)
call AddLeg (coArg!legToBeCreated! sendingSideID)
Add a new passive leg.
Failed
Successful
Failed2
obcsmPars := call MapConnectToBCSM (coArg)
obcsmPars!startState := analyseInformation, obcsmPars!legID := coArg!legToBeCreated! sendingSideID
O_BCSM (obcsmPars)
Create an instance of an O−BCSM:
− starting off from the SelectRoute PIC.
− with the legID as passed in the ConnectArg.
− with routing information as passed in the ConnectArg.
SetLegPtr (coArg!legToBeCreated! sendingSideID, offspring, originating)
Associate the new passive leg with the O−BCSM.
CO1
CO1
SetLegStatus (coArg!legToBeCreated! sendingSideID, pending)
Set the status of the new passive leg to pending.
SetLegStatus (controllingLegID, surrogate)
Set the status of the controlling leg to surrogate.
Connnect (invokeID,,coArg)
TO SSF
/* Pass Connect on to the SSF−FSM associated with this segment. */
Forward

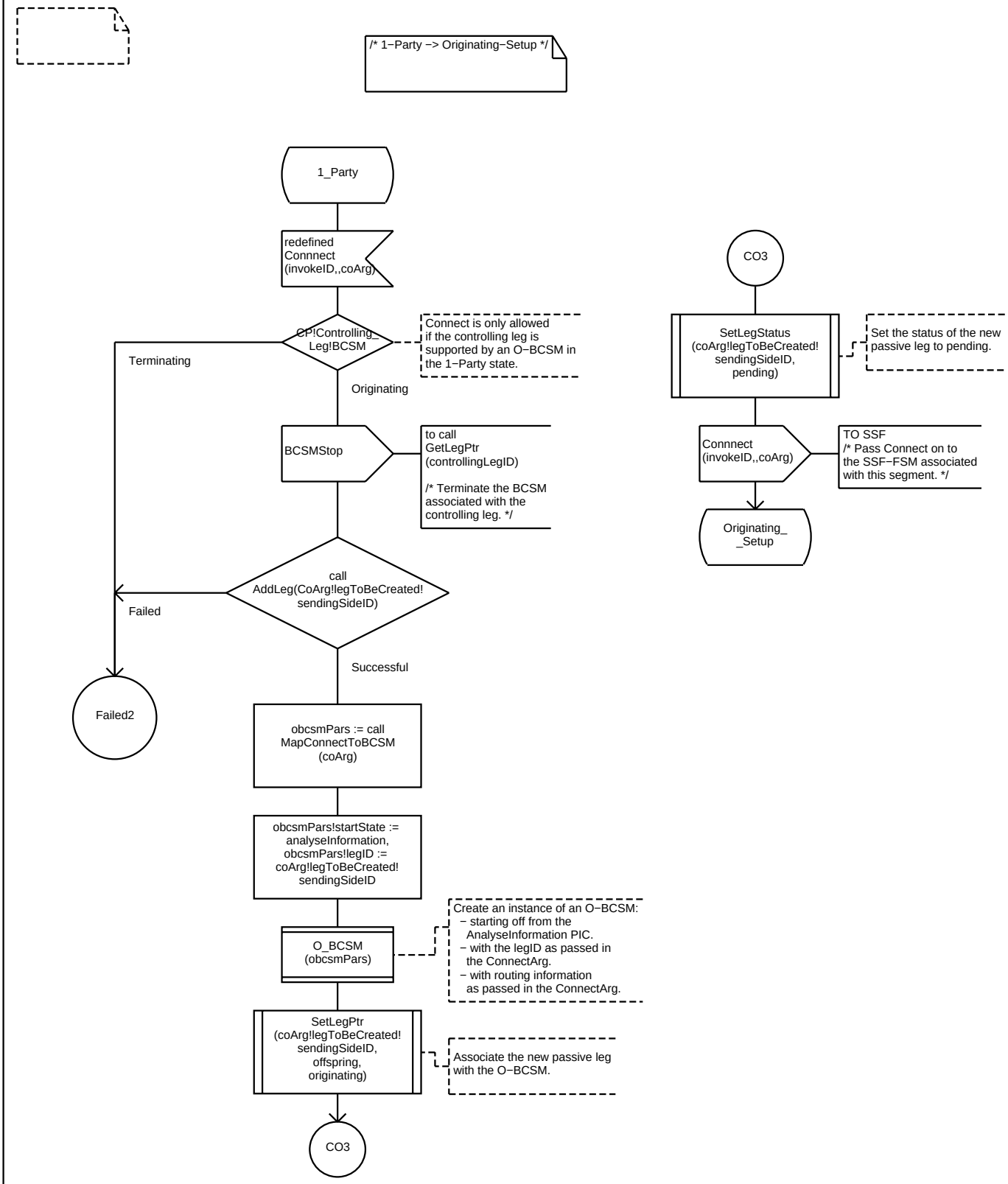

/**** TRANSITIONS BETWEEN THE CONNECTION STATES OF THE CALL SEGMENT. ****/
/* Note: The states of the call segment are defined in Q.1224. */
/**** REDEFINITIONS OF IN CS−1 OPERATIONS ****/
/* Terminating−Setup −> Forward */
Terminating_ _Setup
SelectFacility (invokeID,,sfArg)
destinationNumberRouting_ AddressPresent (sfArg)
false
true
SelectFacility (invokeID,,sfArg)
TO SSF /* Pass SF on to the SSF−FSM associated with this segment. */
Terminating_ _Setup
call AddLeg (sfArg!legToBeCreated! sendingSideID)
Add a new passive leg.
Failed
Successful
Failed2
obcsmPars := call MapSFToBCSM (sfArg)
obcsmPars!startState := nullState, obcsmPars!legID := sfArg!legToBeCreated! sendingSideID
Create an instance of an O−BCSM:
− starting off from the O−Null PIC.
− with the legID as passed in the SFArg.
− with routing information as passed in the SFArg.
O_BCSM (obcsmPars)
CO2
CO2
SetLegPtr (sfArg!legToBeCreated! sendingSideID, offspring, originating)
Associate the new passive leg with the O−BCSM.
SetLegStatus (sfArg!legToBeCreated! sendingSideID, pending)
Set the status of the new passive leg to pending.
SetLegStatus (controllingLegID, surrogate)
Set the status of the controlling leg to surrogate.
SelectFacility (invokeID,,sfArg)
TO SSF /* Pass SF on to the SSF−FSM associated with this segment. */
Forward

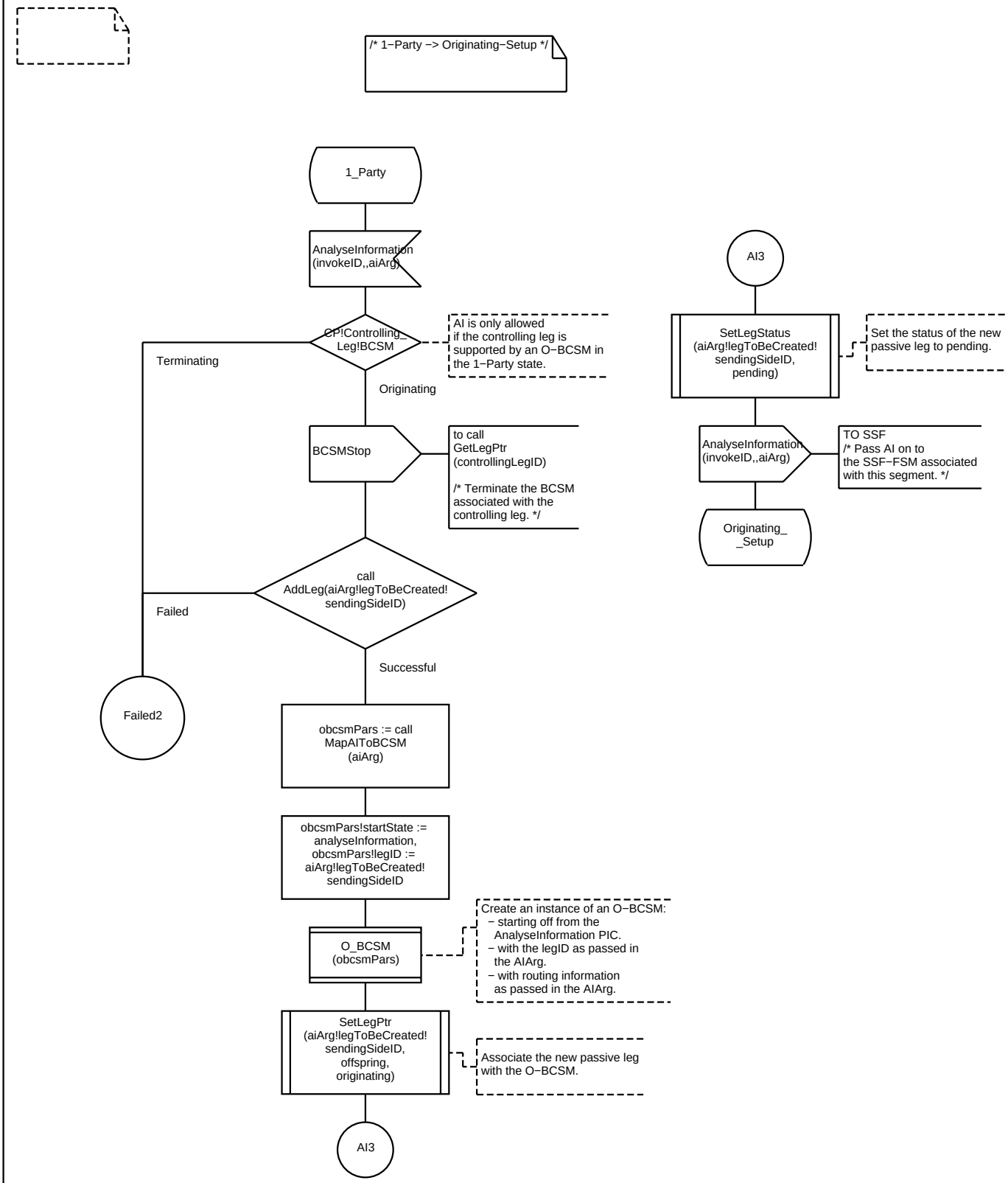

/**** NEW TRANSITIONS FOR IN CS−2. ****/
/* Stable−Multi−Party −> Transfer
On−Hold −> Transfer
Transfer −> Transfer */
/* For the DisconnectLeg(p) operation, it is the BCSMStop event that removes the passive leg from the call segment. The DisconnectLeg operation is passed on to the SSF to allow the SSF to terminate the leg (BCSM) in a controlled way.
The transitions occuring on the BCSM stop event (and specified at other pages) are:
Stable−2−Party −> 1−Party
Stable−Multi−Party −> Stable−2−Party
Transfer −> Stable−1−Party
Originating−Setup −> 1−Party */
Stable_Multi_Party, On_Hold, Transfer
Disconnect_ Leg (invokeID,,dlArg)
DisconnectLeg is continued on the following pages.
dlArg! legToBeReleased! sendingSideID = controllingLegID
true
false
SetLegStatus (controllingLegID, surrogate)
Disconnect_ Leg(invokeID,,dlArg)
to SSF
Disconnect_ Leg(invokeID,,dlArg)
to SSF
call noOfLegs
else
1
Stable_1_ _Party
Transfer
−

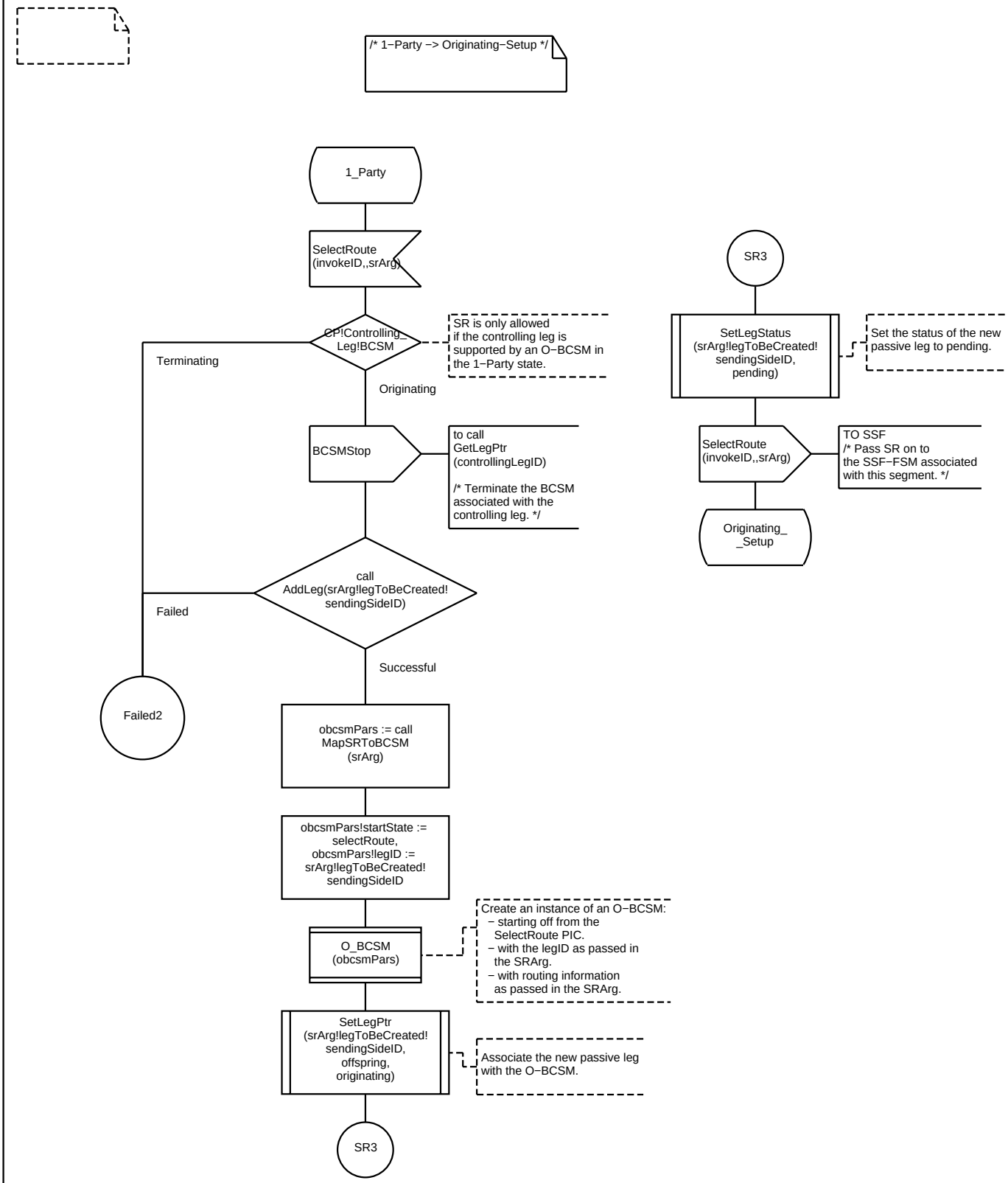

/* Originating−Setup −> Originating−1−Party−Setup
Terminating−Setup −> Terminating−1−Party−Setup */
Originating_
_Setup
Disconnect_
Leg
(invokeID,,dlArg)
dlArg!
legToBeReleased!
sendingSideID =
controllingLegID
true
false
SetLegStatus
(controllingLegID,
surrogate)
Disconnect_
Leg(invokeID,,dlArg)
to SSF
Originating_
_1_Party_Setup
Disconnect_
Leg(invokeID,,dlArg)
to SSF
Terminating_
_Setup
Disconnect_
Leg
(invokeID,,dlArg)
DLArg!
legToBeReleased!
sendingSideID =
controllingLegID
true
false
SetLegStatus
(controllingLegID,
surrogate)
Disconnect_
Leg(invokeID,,dlArg)
to SSF
Terminating_
_1_Party_Setup
Disconnect_
Leg(invokeID,,dlArg)
to SSF

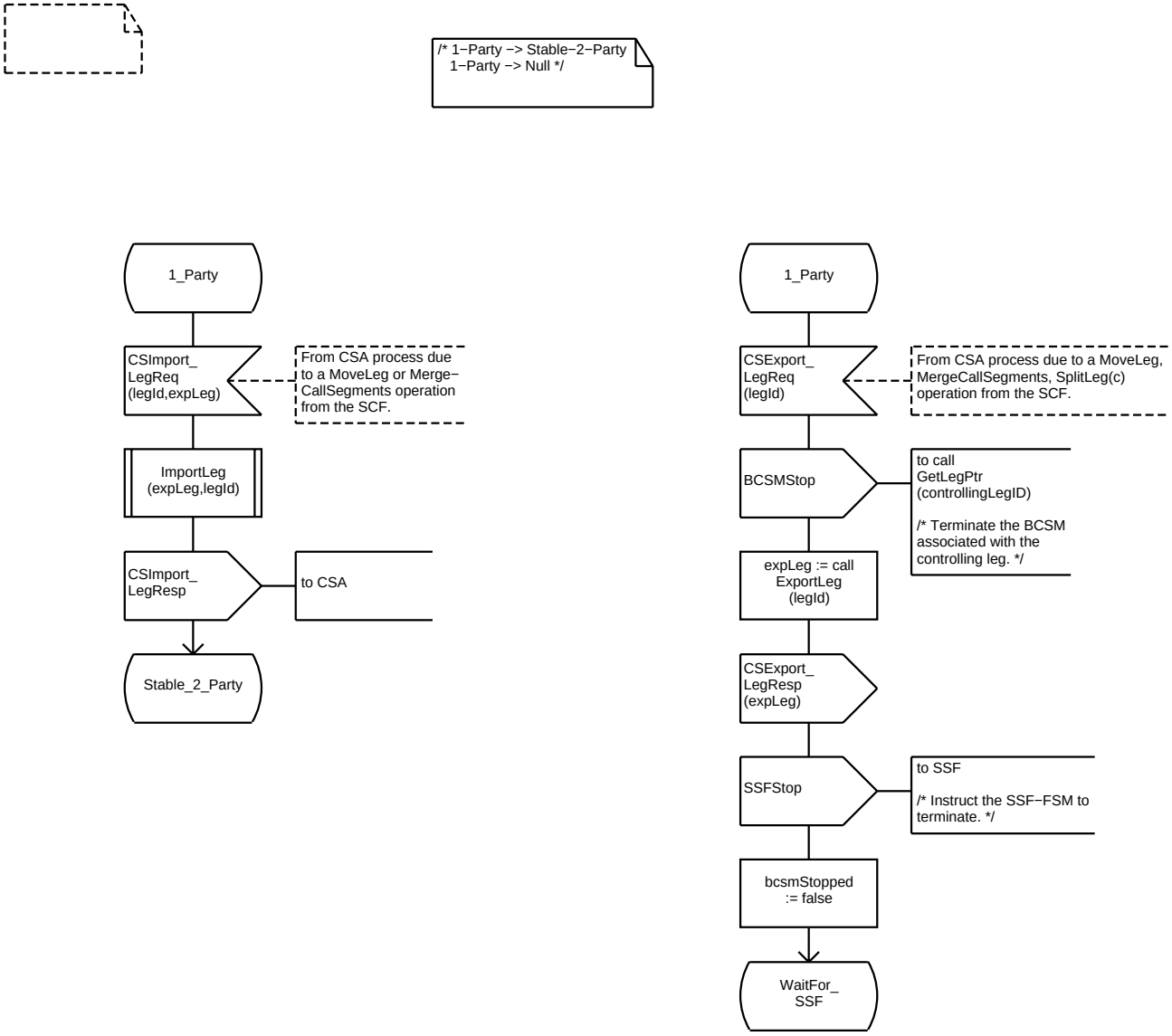


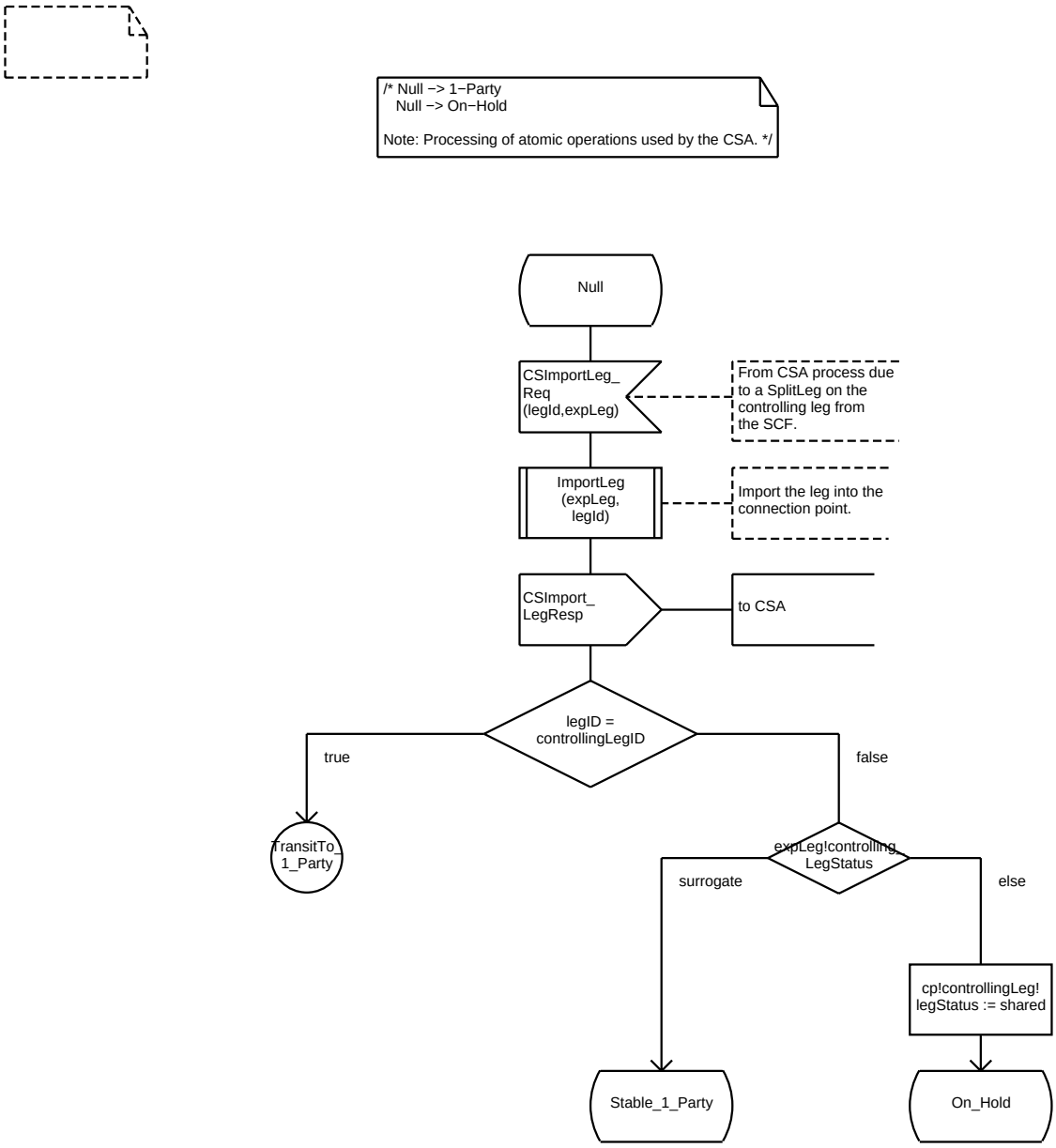
/* Notes:
− DisconnectLeg is not defined for the following states:
Originating−1−Party−Setup, Terminating−1−Party−Setup,
Stable−1−Party.
− DisconnectLeg of the controlling leg in any multi party state
will result in that all BCSMs will generate a ReleaseReq towards
the signalling control. The signalling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

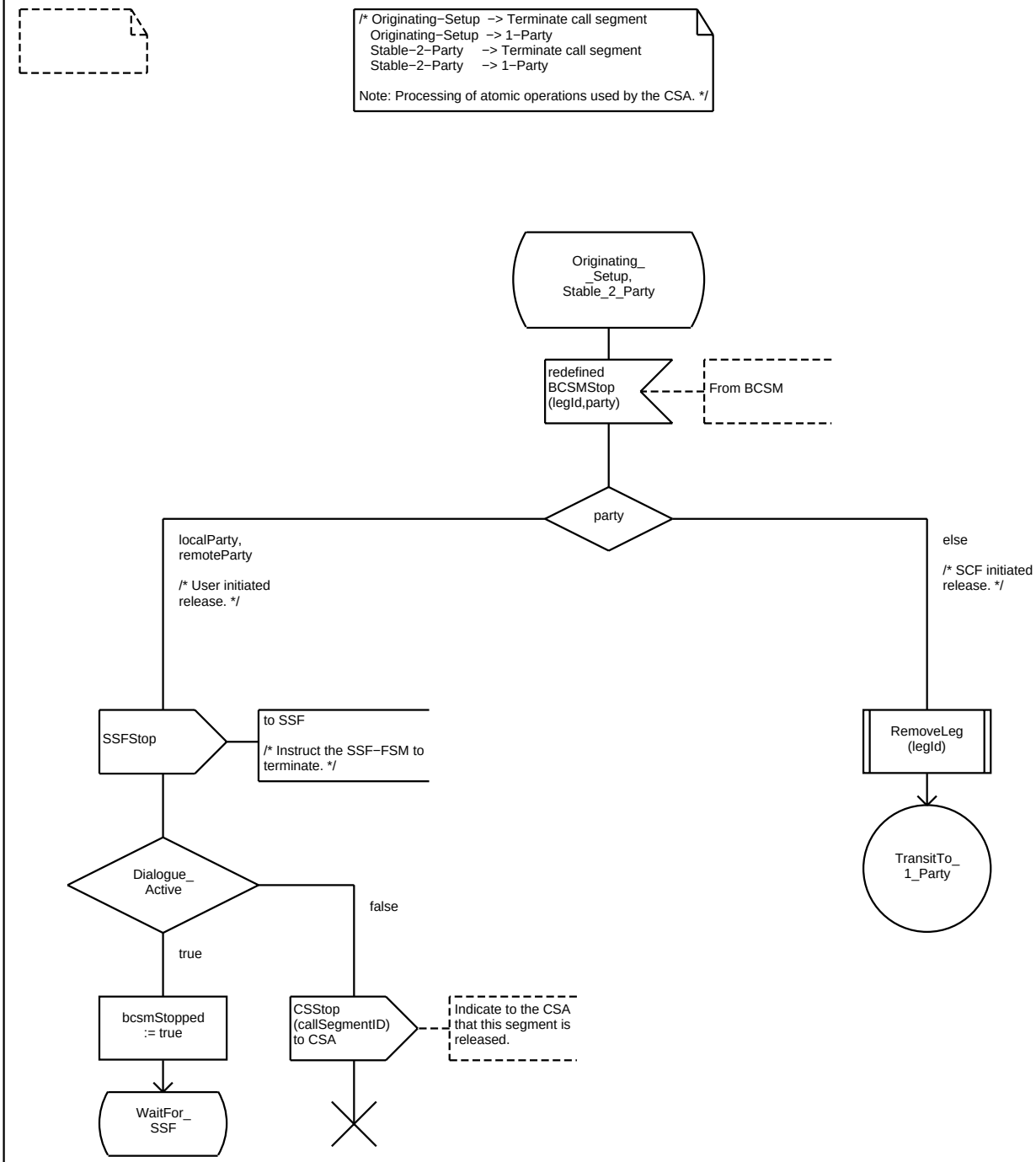

/* 1−Party −> Originating−Setup */
1_Party
redefined Connnect (invokeID,,coArg)
CP!Controlling_ Leg!BCSM
Connect is only allowed if the controlling leg is supported by an O−BCSM in the 1−Party state.
Terminating
Originating
BCSMStop
to call GetLegPtr (controllingLegID)
/* Terminate the BCSM associated with the controlling leg. */
call AddLeg(CoArg!legToBeCreated! sendingSideID)
Failed
Successful
Failed2
obcsmPars := call MapConnectToBCSM (coArg)
obcsmPars!startState := analyseInformation, obcsmPars!legID := coArg!legToBeCreated! sendingSideID
O_BCSM (obcsmPars)
Create an instance of an O−BCSM:
− starting off from the AnalyseInformation PIC.
− with the legID as passed in the ConnectArg.
− with routing information as passed in the ConnectArg.
SetLegPtr (coArg!legToBeCreated! sendingSideID, offspring, originating)
Associate the new passive leg with the O−BCSM.
CO3
CO3
SetLegStatus (coArg!legToBeCreated! sendingSideID, pending)
Set the status of the new passive leg to pending.
Connnect (invokeID,,coArg)
TO SSF
/* Pass Connect on to the SSF−FSM associated with this segment. */
Originating_ _Setup

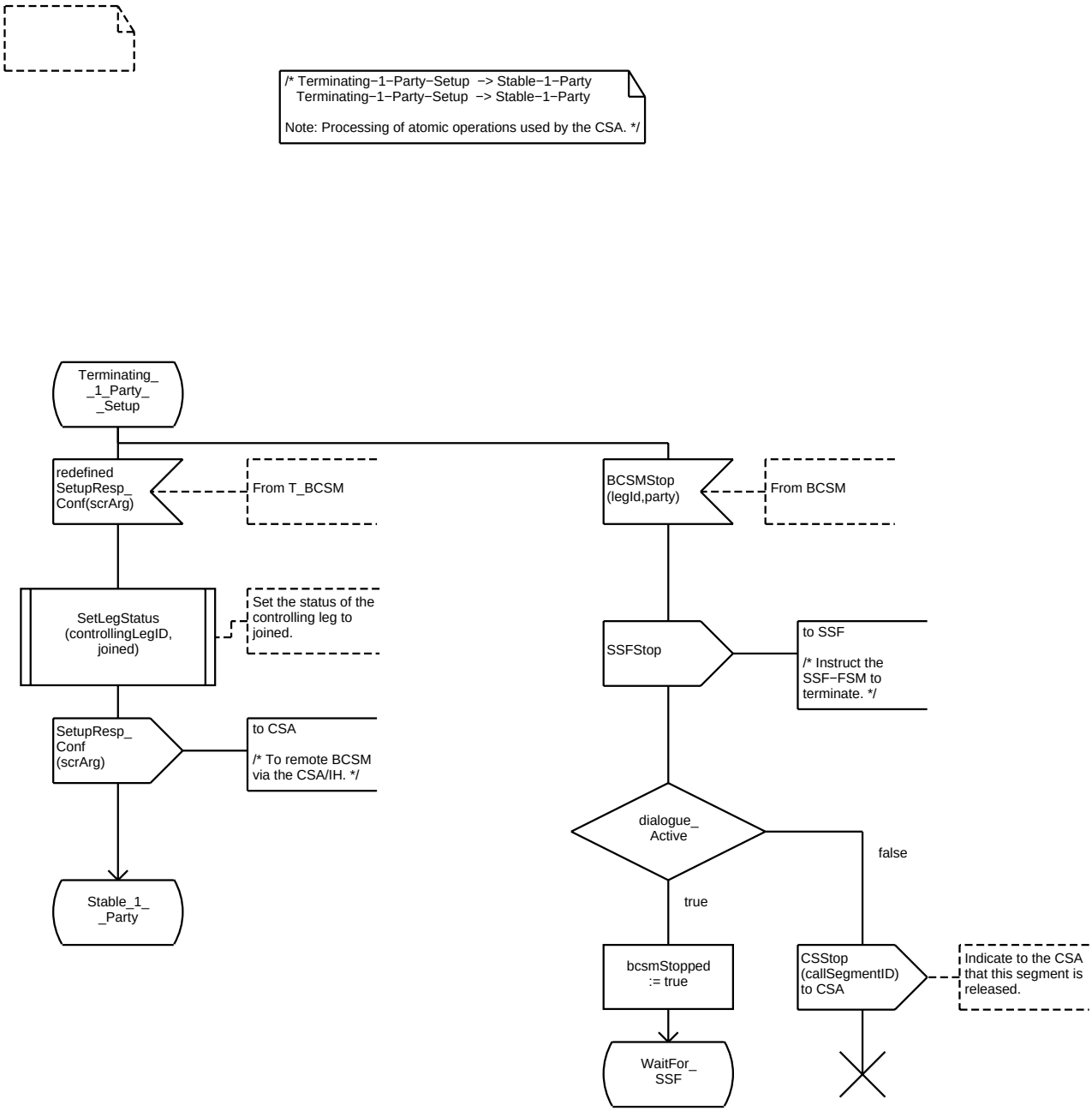

/* 1−Party −> Originating−Setup */
1_Party
AnalyseInformation
(invokeID,,aiArg)
CP!Controlling_
Leg!BCSM
AI is only allowed
if the controlling leg is
supported by an O−BCSM in
the 1−Party state.
Terminating
Originating
BCSMStop
to call
GetLegPtr
(controllingLegID)
/* Terminate the BCSM
associated with the
controlling leg. */
call
AddLeg(aiArg!legToBeCreated!
sendingSideID)
Failed
Successful
Failed2
obcsmPars := call
MapAIToBCSM
(aiArg)
obcsmPars!startState :=
analyseInformation,
obcsmPars!legID :=
aiArg!legToBeCreated!
sendingSideID
O_BCSM
(obcsmPars)
Create an instance of an O−BCSM:
− starting off from the
AnalyseInformation PIC.
− with the legID as passed in
the AIArg.
− with routing information
as passed in the AIArg.
SetLegPtr
(aiArg!legToBeCreated!
sendingSideID,
offspring,
originating)
Associate the new passive leg
with the O−BCSM.
AI3
AI3
SetLegStatus
(aiArg!legToBeCreated!
sendingSideID,
pending)
Set the status of the new
passive leg to pending.
AnalyseInformation
(invokeID,,aiArg)
TO SSF
/* Pass AI on to
the SSF−FSM associated
with this segment. */
Originating_
_Setup

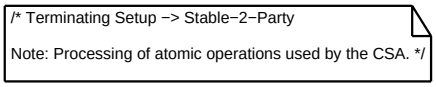

/* 1−Party −> Originating−Setup */
1_Party
SelectRoute
(invokeID,,srArg)
CP!Controlling_
Leg!BCSM
SR is only allowed
if the controlling leg is
supported by an O−BCSM in
the 1−Party state.
Terminating
Originating
BCSMStop
to call
GetLegPtr
(controllingLegID)
/* Terminate the BCSM
associated with the
controlling leg. */
call
AddLeg(srArg!legToBeCreated!
sendingSideID)
Failed
Successful
Failed2
obcsmPars := call
MapSRToBCSM
(srArg)
obcsmPars!startState :=
selectRoute,
obcsmPars!legID :=
srArg!legToBeCreated!
sendingSideID
O_BCSM
(obcsmPars)
Create an instance of an O−BCSM:
− starting off from the
SelectRoute PIC.
− with the legID as passed in
the SRArg.
− with routing information
as passed in the SRArg.
SetLegPtr
(srArg!legToBeCreated!
sendingSideID,
offspring,
originating)
Associate the new passive leg
with the O−BCSM.
SR3
SR3
SetLegStatus
(srArg!legToBeCreated!
sendingSideID,
pending)
Set the status of the new
passive leg to pending.
SelectRoute
(invokeID,,srArg)
TO SSF
/* Pass SR on to
the SSF−FSM associated
with this segment. */
Originating_
_Setup

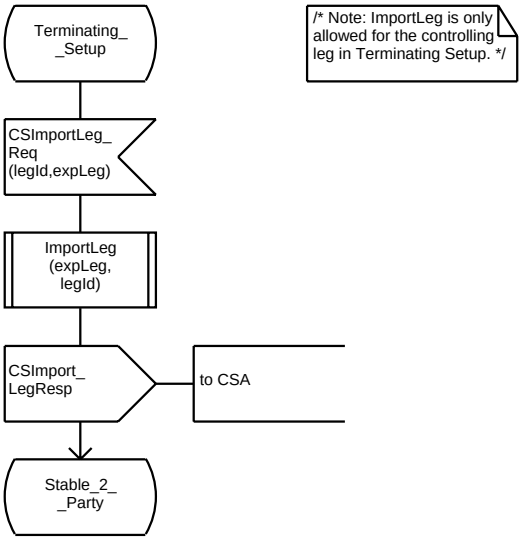

/* 1−Party −> Stable−2−Party
1−Party −> Null */
1_Party
CSImport_
LegReq
(legId,expLeg)
From CSA process due to a MoveLeg or Merge−CallSegments operation from the SCF.
ImportLeg
(expLeg,legId)
CSImport_
LegResp
to CSA
Stable_2_Party
1_Party
CSExport_
LegReq
(legId)
From CSA process due to a MoveLeg,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

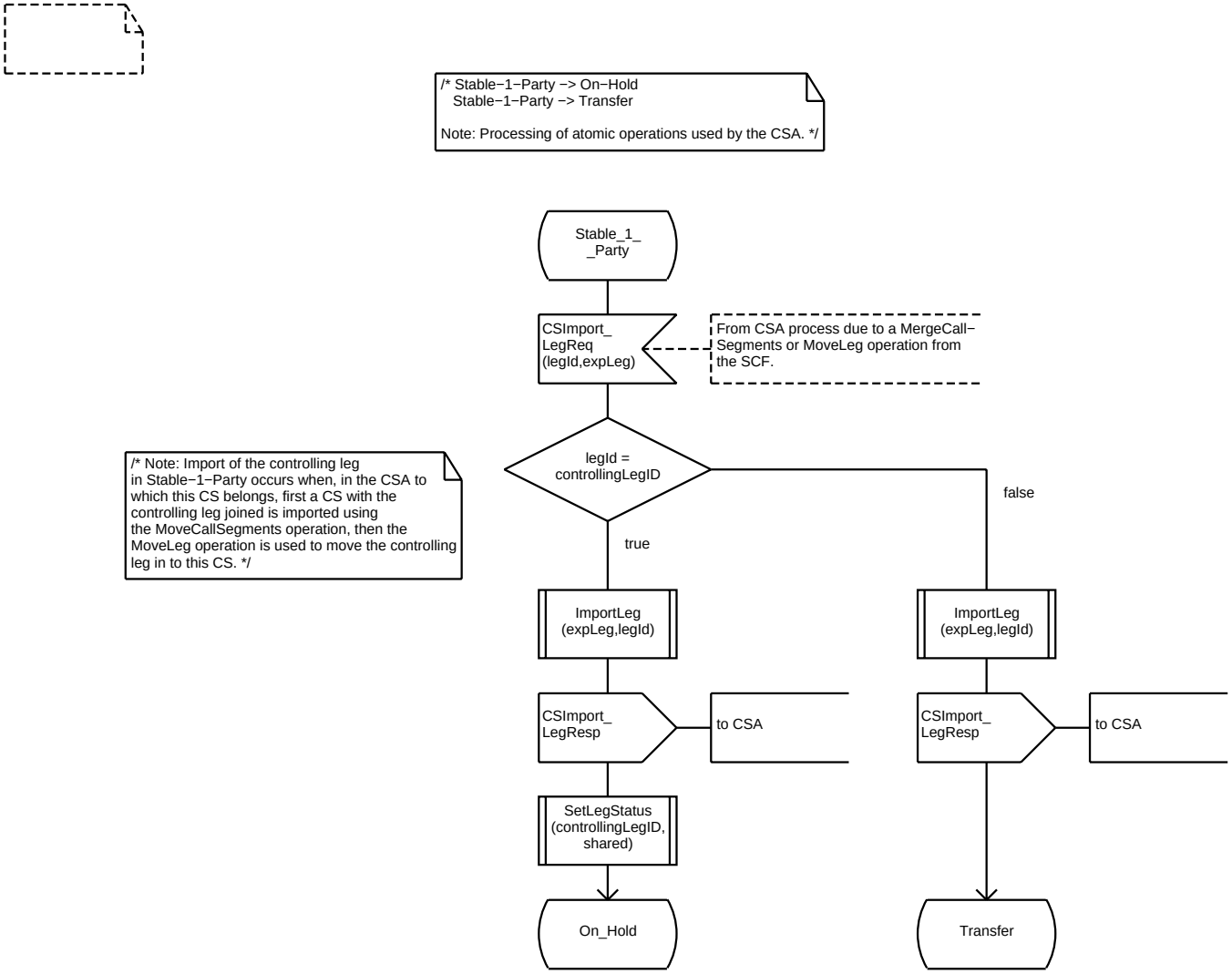

/* Null –> 1–Party
Null –> On–Hold
Note: Processing of atomic operations used by the CSA. */
Null
CSImportLeg_ Req (legId,expLeg)
From CSA process due to a SplitLeg on the controlling leg from the SCF.
ImportLeg (expLeg, legId)
Import the leg into the connection point.
CSImport_ LegResp
to CSA
legID = controllingLegID
true
false
TransitTo_ 1_Party
expLeg!controlling LegStatus
surrogate
else
cp!controllingLeg! legStatus := shared
Stable_1_Party
On_Hold

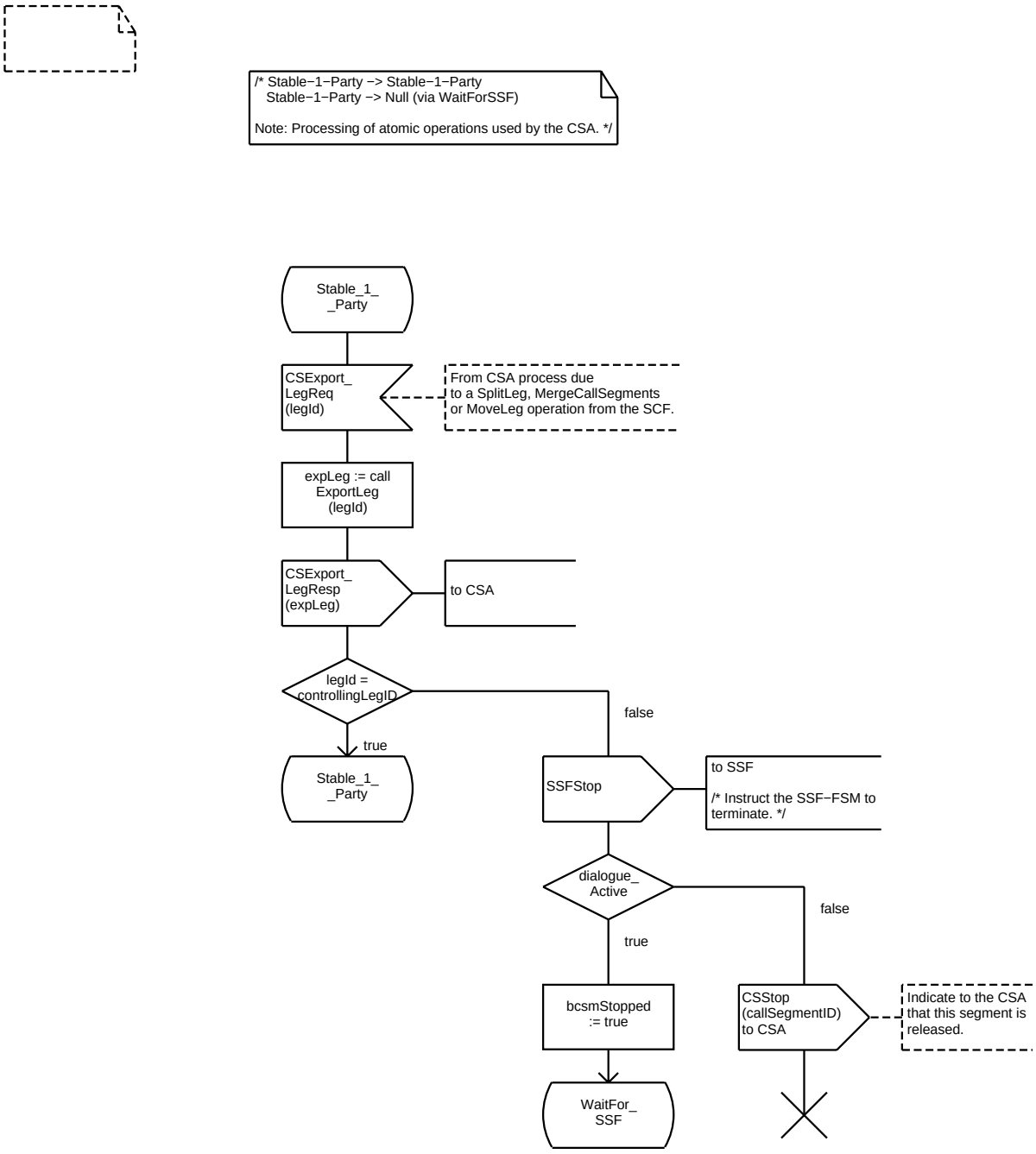

/* Originating−Setup −> Terminate call segment
Originating−Setup −> 1−Party
Stable−2−Party −> Terminate call segment
Stable−2−Party −> 1−Party
Note: Processing of atomic operations used by the CSA. */
Originating_
_Setup,
Stable_2_Party
redefined
BCSMStop
(legId,party)
From BCSM
party
localParty,
remoteParty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

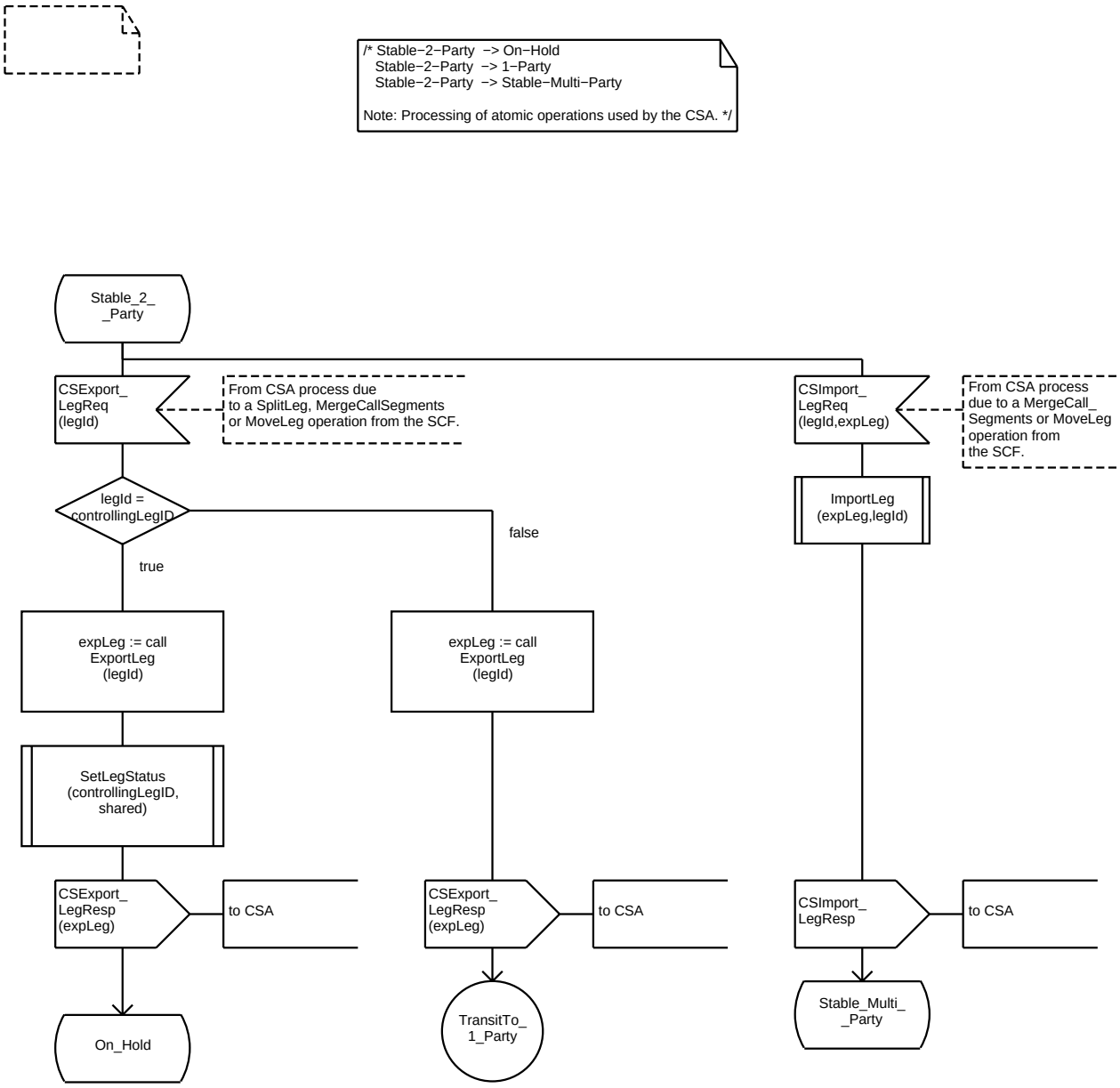

/* Terminating−1−Party−Setup −> Stable−1−Party
Terminating−1−Party−Setup −> Stable−1−Party
Note: Processing of atomic operations used by the CSA. */
Terminating_ _1_Party_ _Setup
redefined SetupResp_ Conf(scrArg)
From T_BCSM
BCSMStop (legId,party)
From BCSM
SetLegStatus (controllingLegID, joined)
Set the status of the controlling leg to joined.
SSFStop
to SSF
/* Instruct the SSF−FSM to terminate. */
SetupResp_ Conf (scrArg)
to CSA
/* To remote BCSM via the CSA/IH. */
dialogue_ Active
false
true
Stable_1_ _Party
bcsmStopped := true
CSStop (callSegmentID) to CSA
Indicate to the CSA that this segment is released.
WaitFor_ SSF

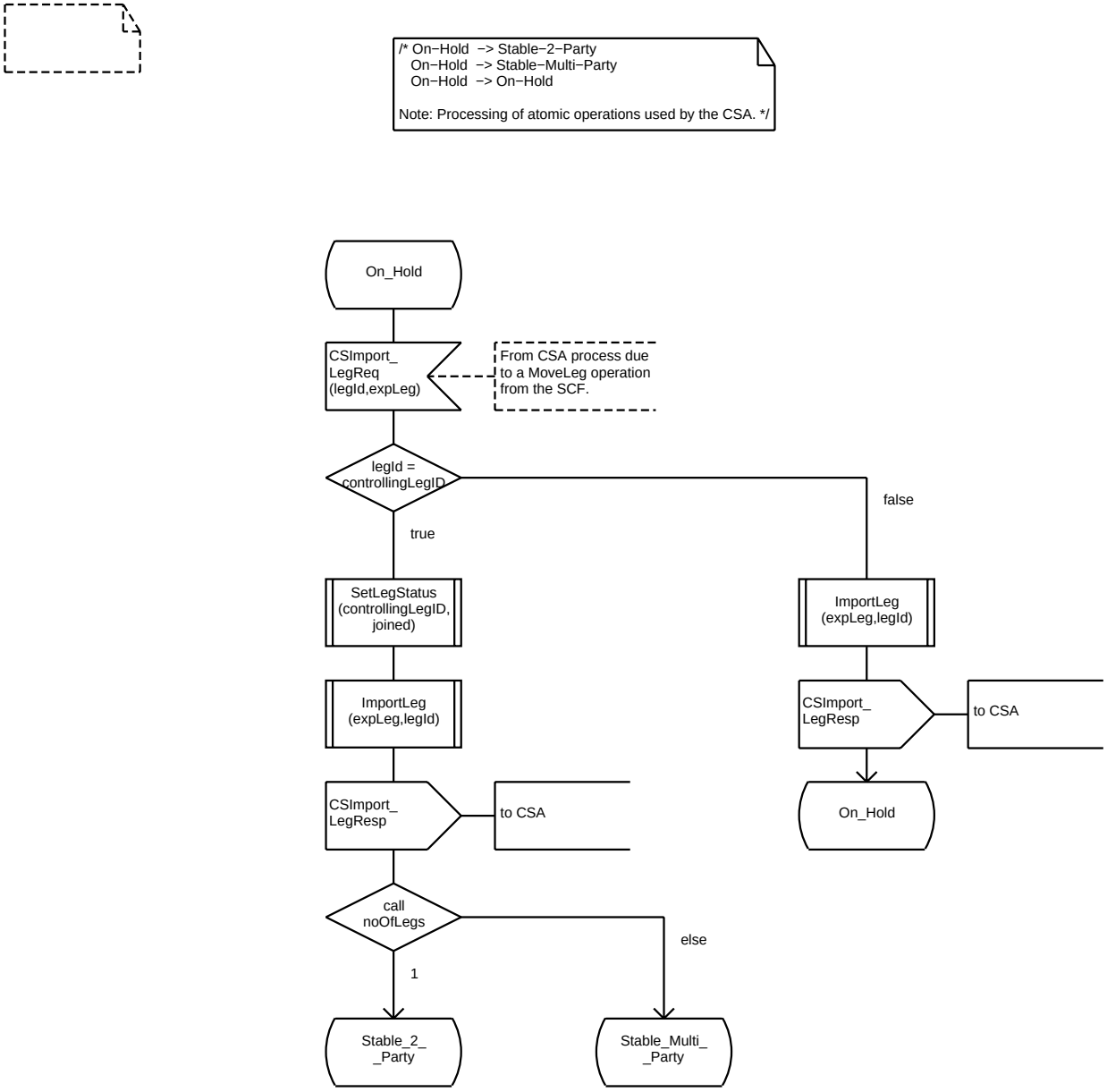

/* Terminating Setup −> Stable−2−Party
Note: Processing of atomic operations used by the CSA. */

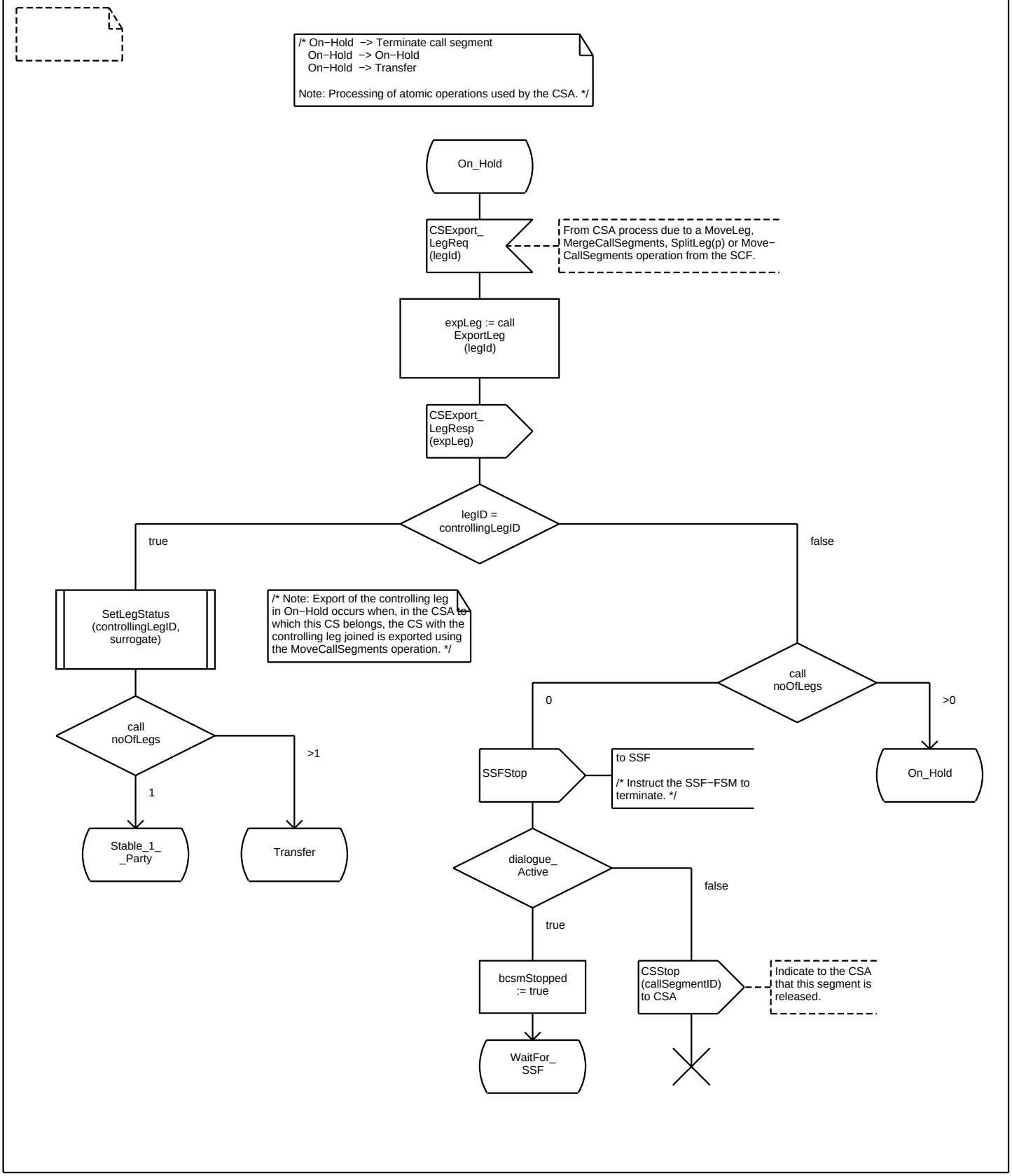

Terminating_
_Setup
/* Note: ImportLeg is only allowed for the controlling leg in Terminating Setup. */
CSImportLeg_
Req
(legId,expLeg)
ImportLeg
(expLeg,
legId)
CSImport_
LegResp
to CSA
Stable_2_
_Party

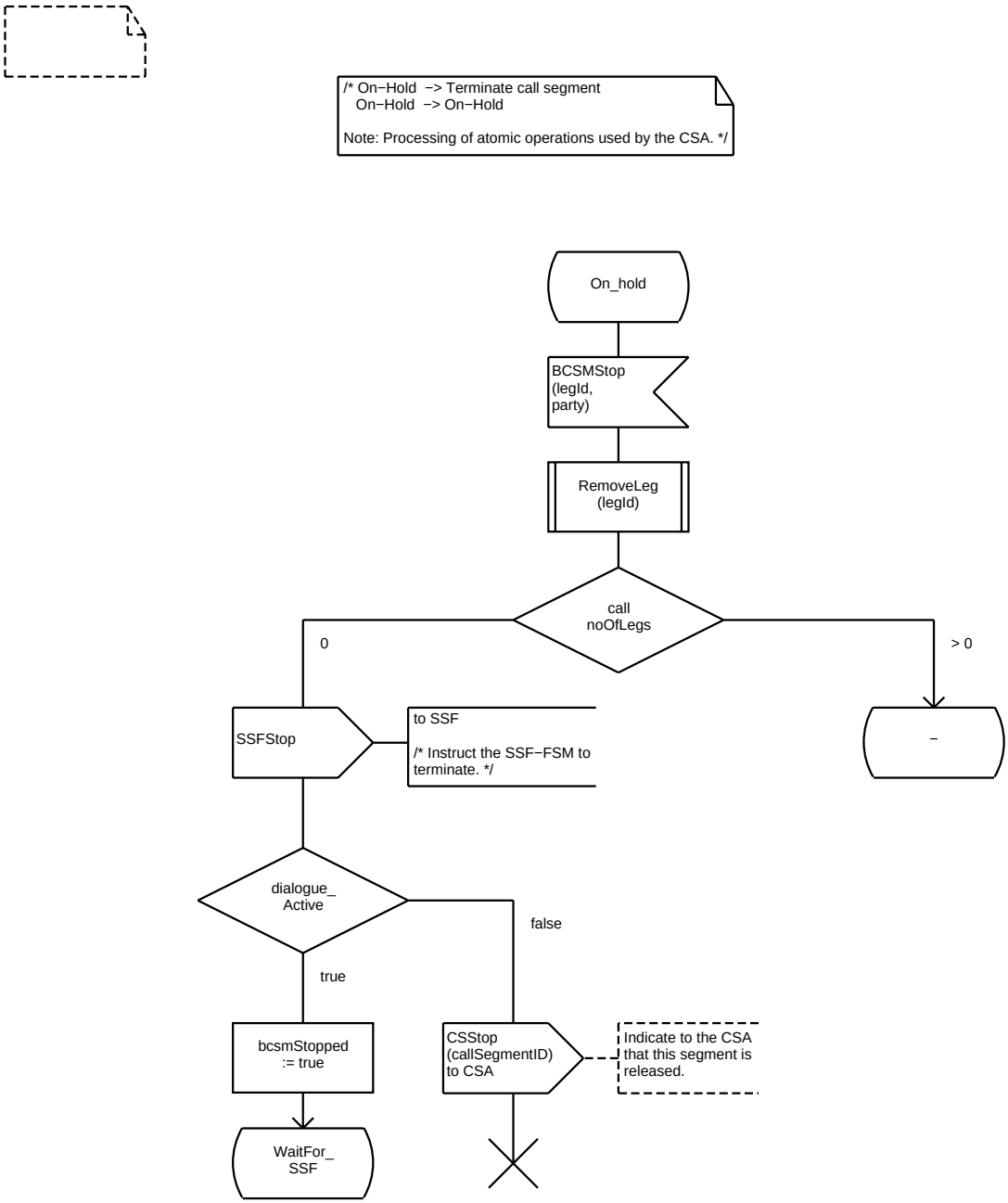

/* Stable−1−Party −> On−Hold
Stable−1−Party −> Transfer
Note: Processing of atomic operations used by the CSA. */
Stable_1_
_Party
CSImport_
LegReq
(legId,expLeg)
From CSA process due to a MergeCall−
Segments or MoveLeg operation from
the SCF.
/* Note: Import of the controlling leg
in Stable−1−Party occurs when, in the CSA to
which this CS belongs, first a CS with the
controlling leg joined is imported using
the MoveCallSegments operation, then the
MoveLeg operation is used to move the controlling
leg in to this CS. */
legId =
controllingLegID
false
true
ImportLeg
(expLeg,legId)
ImportLeg
(expLeg,legId)
CSImport_
LegResp
to CSA
CSImport_
LegResp
to CSA
SetLegStatus
(controllingLegID,
shared)
On_Hold
Transfer

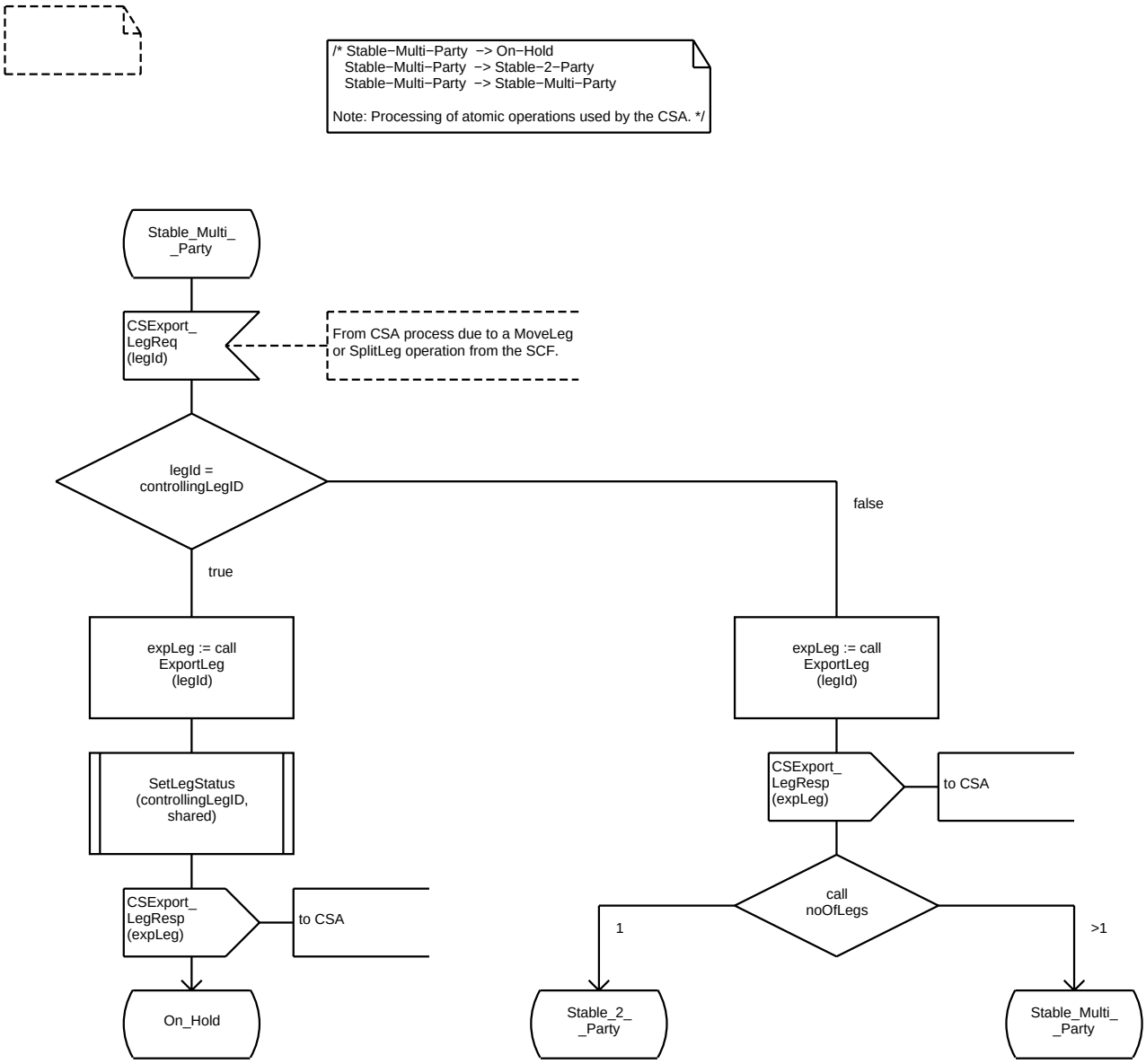

/* Stable−1−Party −> Stable−1−Party
Stable−1−Party −> Null (via WaitForSSF)
Note: Processing of atomic operations used by the CSA. */
Stable_1_ _Party
CSExport_ LegReq (legId)
From CSA process due to a SplitLeg, MergeCallSegments or MoveLeg operation from the SCF.
expLeg := call ExportLeg (legId)
CSExport_ LegResp (expLeg)
to CSA
legId = controllingLegID
false
true
Stable_1_ _Party
SSFStop
to SSF
/* Instruct the SSF−FSM to terminate. */
dialogue_ Active
false
true
bcsmStopped := true
CSStop (callSegmentID) to CSA
Indicate to the CSA that this segment is released.
WaitFor_ SSF

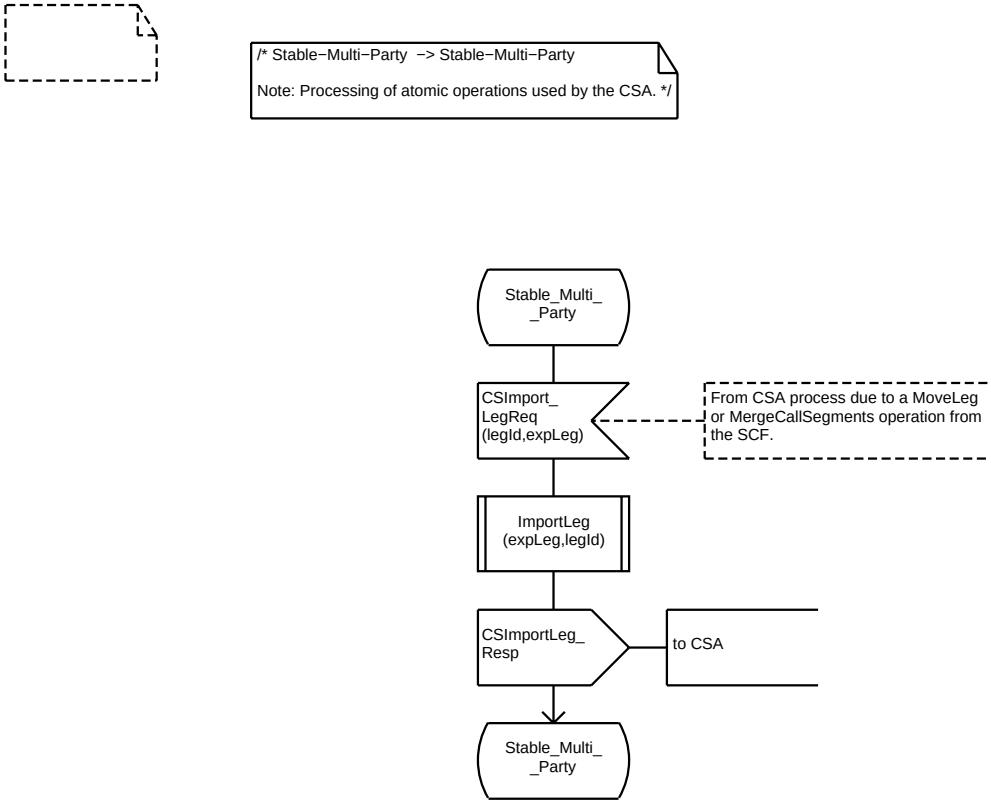

/* Stable−2−Party −> On−Hold
Stable−2−Party −> 1−Party
Stable−2−Party −> Stable−Multi−Party
Note: Processing of atomic operations used by the CSA. */
Stable_2_
_Party
CSExport_
LegReq
(legId)
From CSA process due
to a SplitLeg, MergeCallSegments
or MoveLeg operation from the SCF.
CSImport_
LegReq
(legId,expLeg)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


/* On−Hold −> Stable−2−Party
On−Hold −> Stable−Multi−Party
On−Hold −> On−Hold
Note: Processing of atomic operations used by the CSA. */
On_Hold
CSImport_ LegReq (legId,expLeg)
From CSA process due to a MoveLeg operation from the SCF.
legId = controllingLegID
false
true
SetLegStatus (controllingLegID, joined)
ImportLeg (expLeg,legId)
CSImport_ LegResp
to CSA
ImportLeg (expLeg,legId)
CSImport_ LegResp
to CSA
On_Hold
call noOfLegs
else
1
Stable_2_ _Party
Stable_Multi_ _Party


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

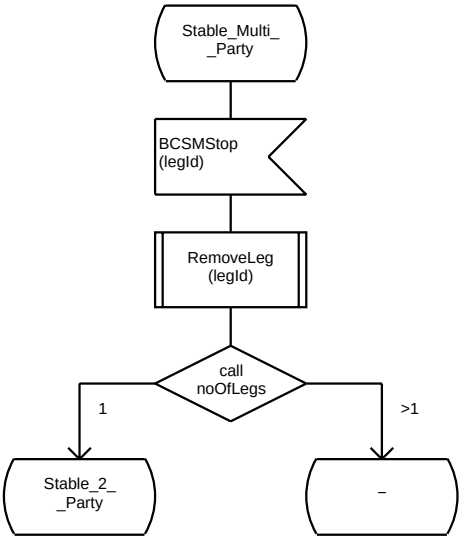

/* On−Hold −> Terminate call segment
On−Hold −> On−Hold
Note: Processing of atomic operations used by the CSA. */
On_hold
BCSMStop
(legId,
party)
RemoveLeg
(legId)
call
noOfLegs
0
> 0
SSFStop
to SSF
/* Instruct the SSF−FSM to
terminate. */
−
dialogue_
Active
false
true
bcsmStopped
:= true
CSStop
(callSegmentID)
to CSA
Indicate to the CSA
that this segment is
released.
WaitFor_
SSF

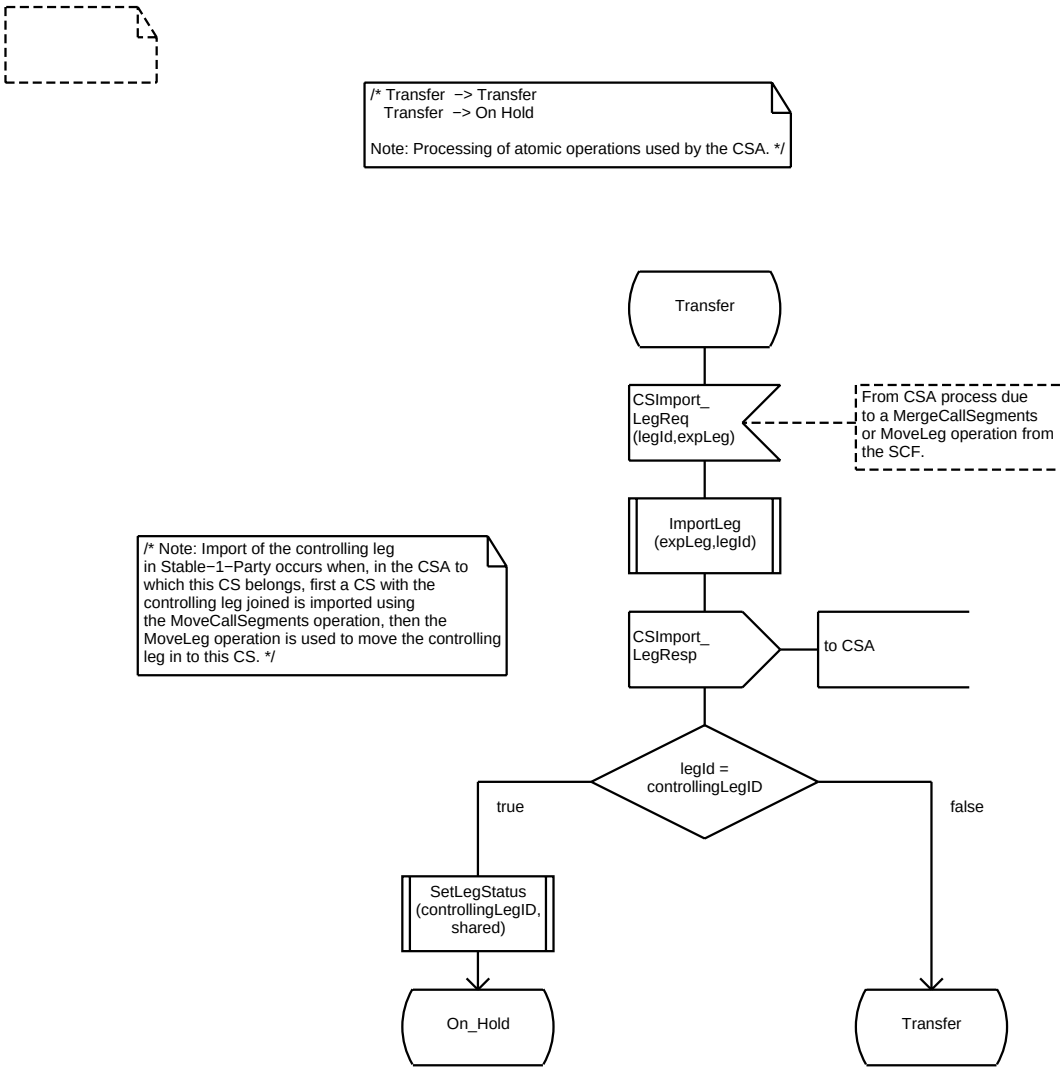

/* Stable−Multi−Party −> On−Hold
Stable−Multi−Party −> Stable−2−Party
Stable−Multi−Party −> Stable−Multi−Party
Note: Processing of atomic operations used by the CSA. */
Stable_Multi_
_Party
CSExport_
LegReq
(legId)
From CSA process due to a MoveLeg
or SplitLeg operation from the SCF.
legId =
controllingLegID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

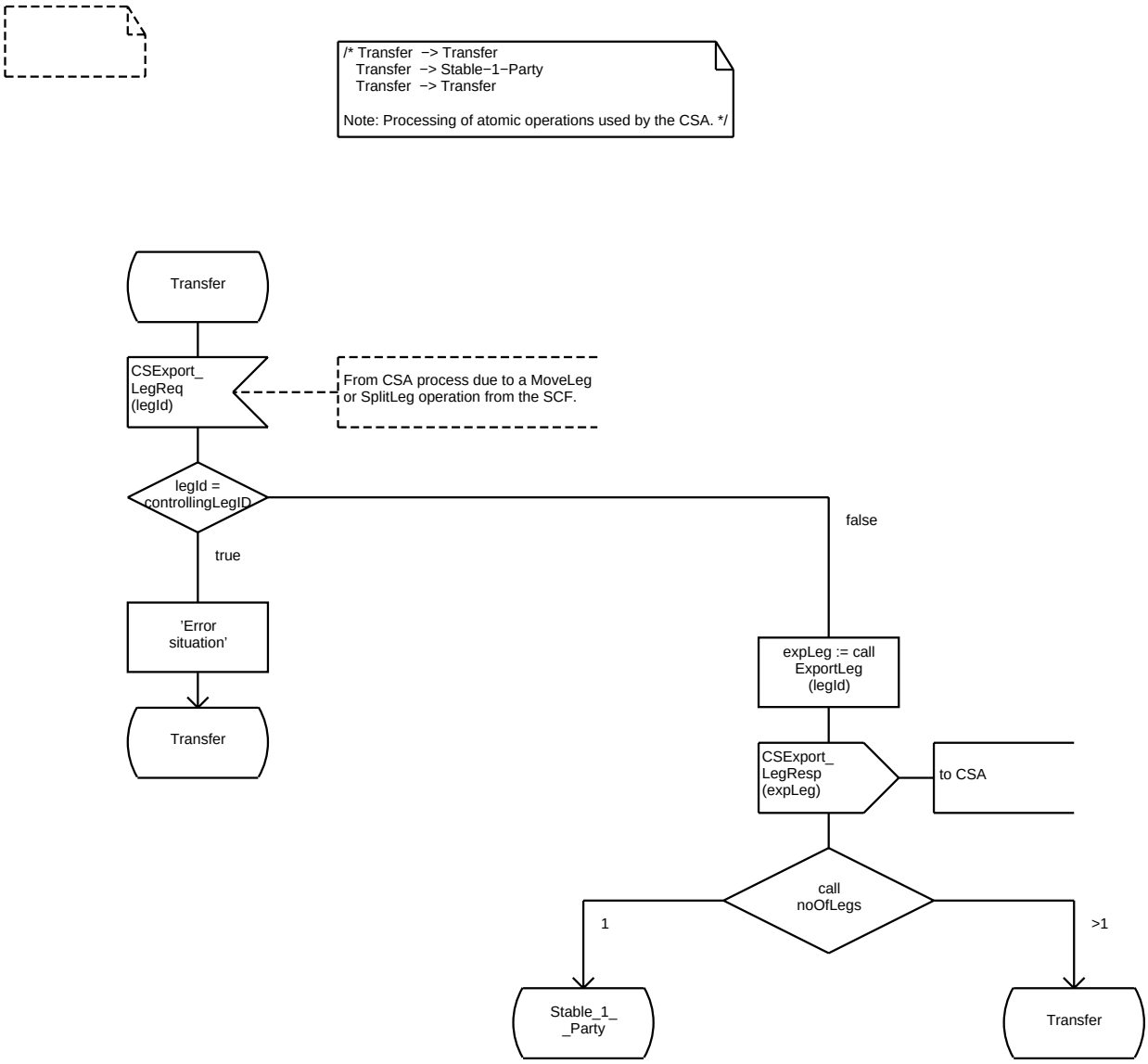

/* Stable−Multi−Party −> Stable−Multi−Party
Note: Processing of atomic operations used by the CSA. */
Stable_Multi_
_Party
CSImport_
LegReq
(legId,expLeg)
From CSA process due to a MoveLeg or MergeCallSegments operation from the SCF.
ImportLeg
(expLeg,legId)
CSImportLeg_
Resp
to CSA
Stable_Multi_
_Party

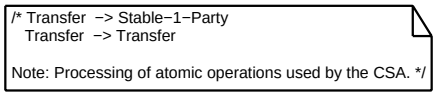
/* Stable−Multi−Party −> Stable−2−Party
Stable−Multi−Party −> Stable−Multi−Party

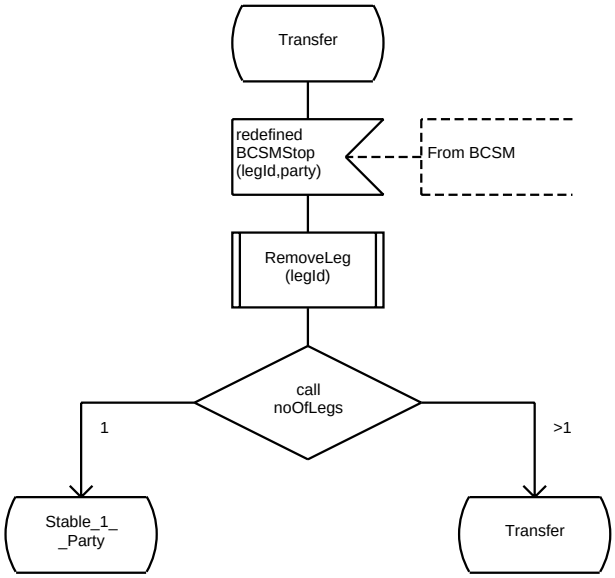
Note: Processing of atomic operations used by the CSA. */

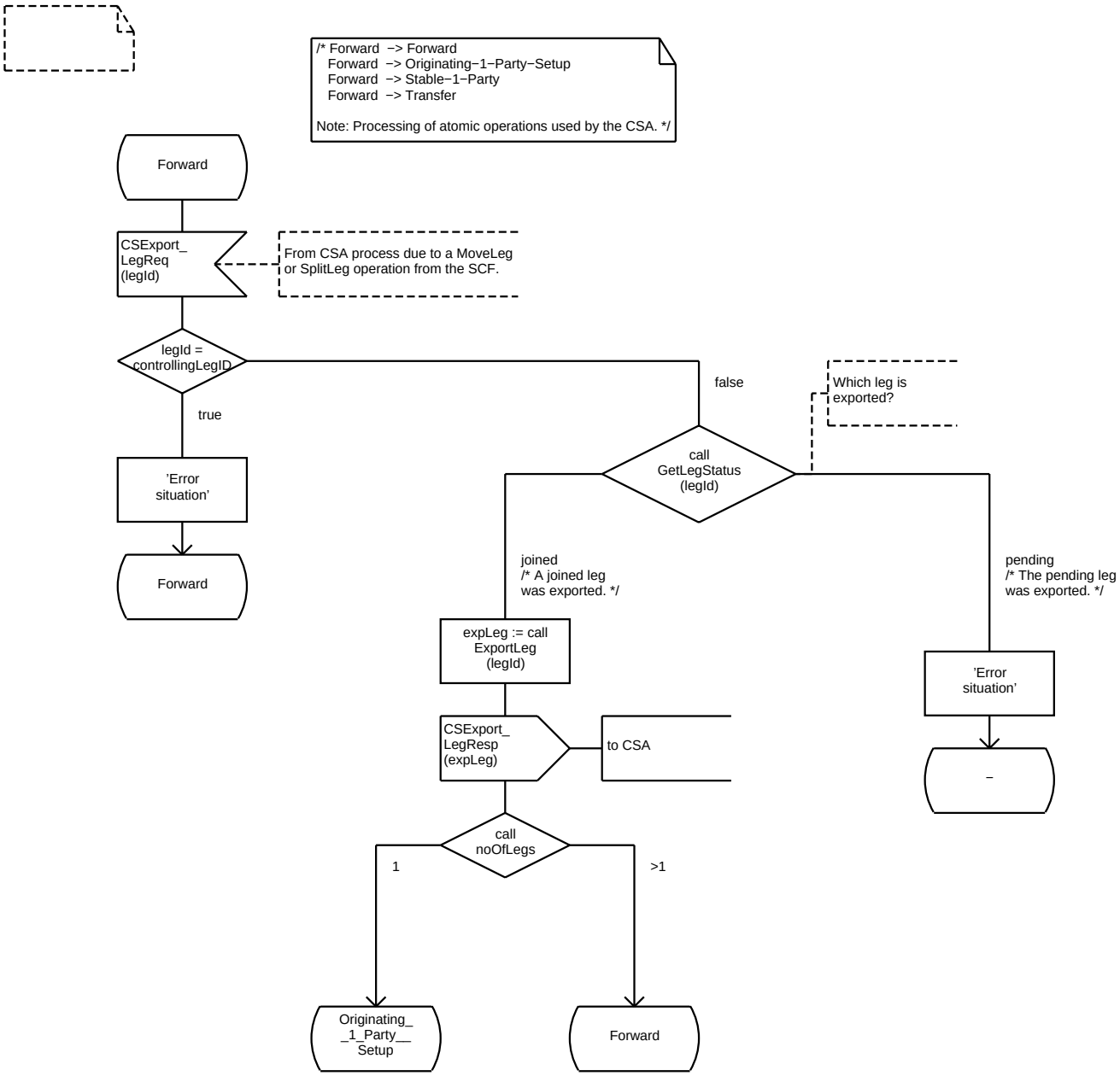


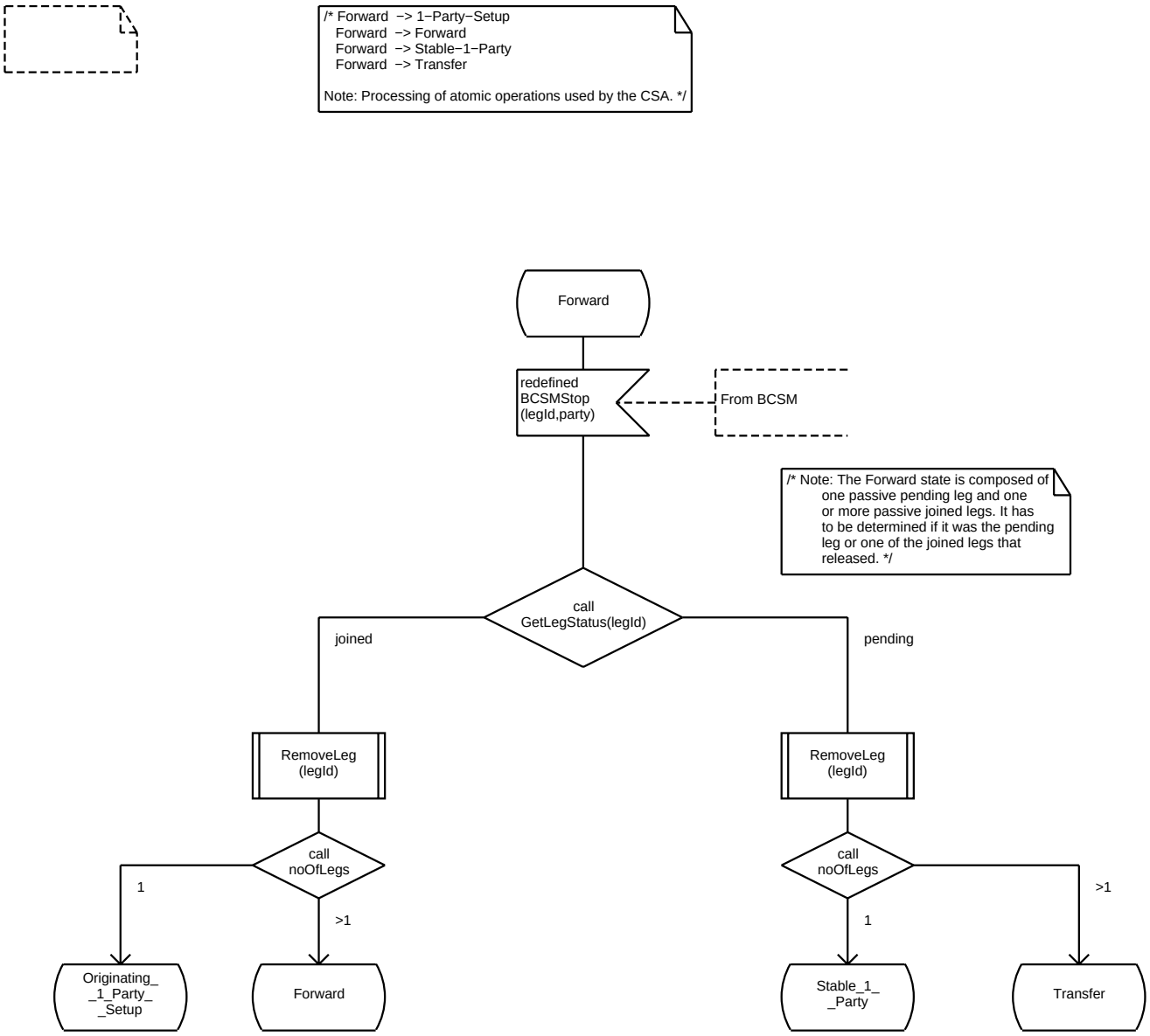

/* Transfer –> Transfer
Transfer –> On Hold
Note: Processing of atomic operations used by the CSA. */
Transfer
CSImport_
LegReq
(legId,expLeg)
From CSA process due
to a MergeCallSegments
or MoveLeg operation from
the SCF.
ImportLeg
(expLeg,legId)
/* Note: Import of the controlling leg
in Stable−1−Party occurs when, in the CSA to
which this CS belongs, first a CS with the
controlling leg joined is imported using
the MoveCallSegments operation, then the
MoveLeg operation is used to move the controlling
leg in to this CS. */
CSImport_
LegResp
to CSA
legId =
controllingLegID
true
false
SetLegStatus
(controllingLegID,
shared)
On_Hold
Transfer

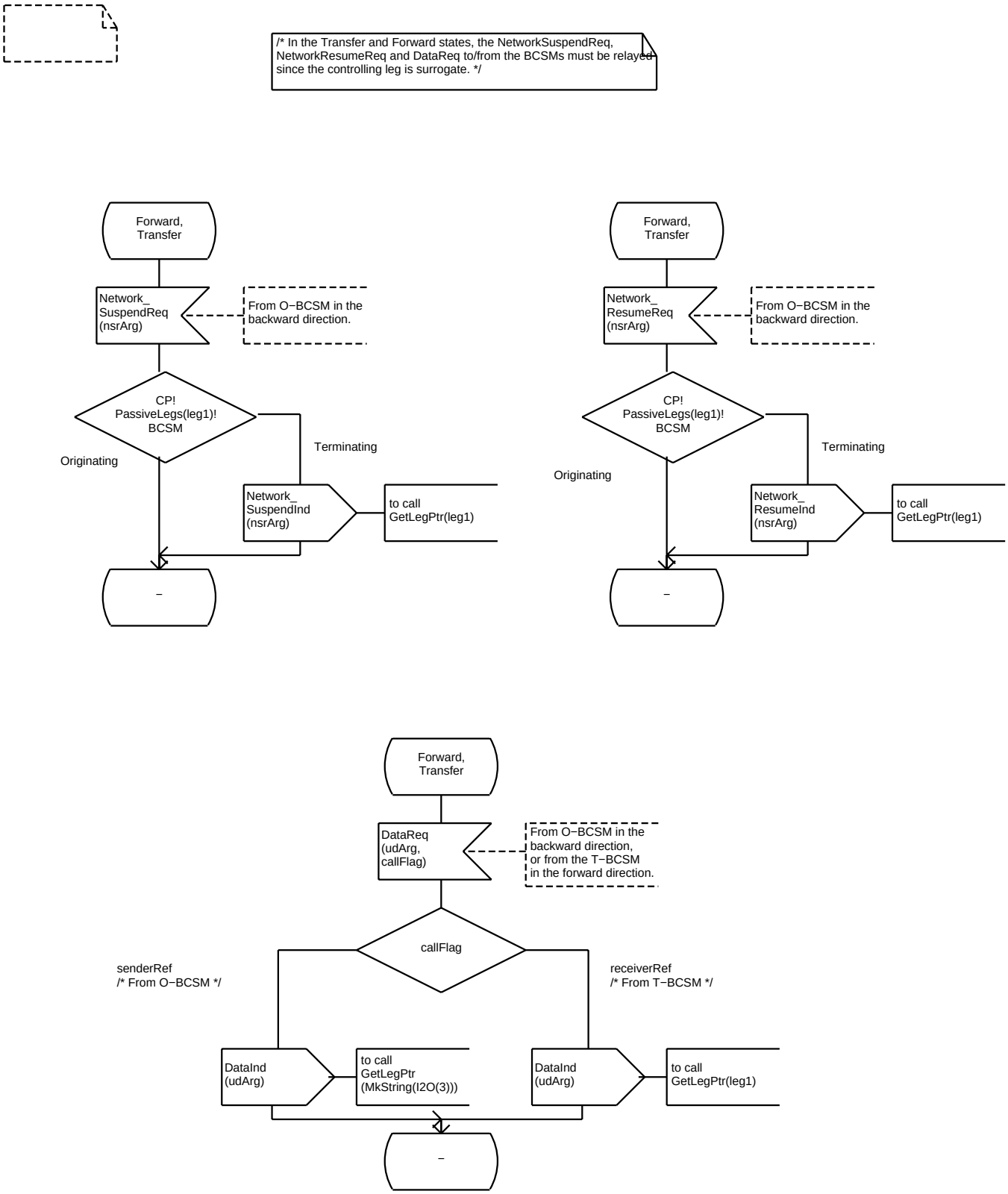

/* Transfer –> Transfer
Transfer –> Stable−1−Party
Transfer –> Transfer
Note: Processing of atomic operations used by the CSA. */
Transfer
CSExport_
LegReq
(legId)
From CSA process due to a MoveLeg
or SplitLeg operation from the SCF.
legId =
controllingLegID
false
true
'Error
situation'
Transfer
expLeg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

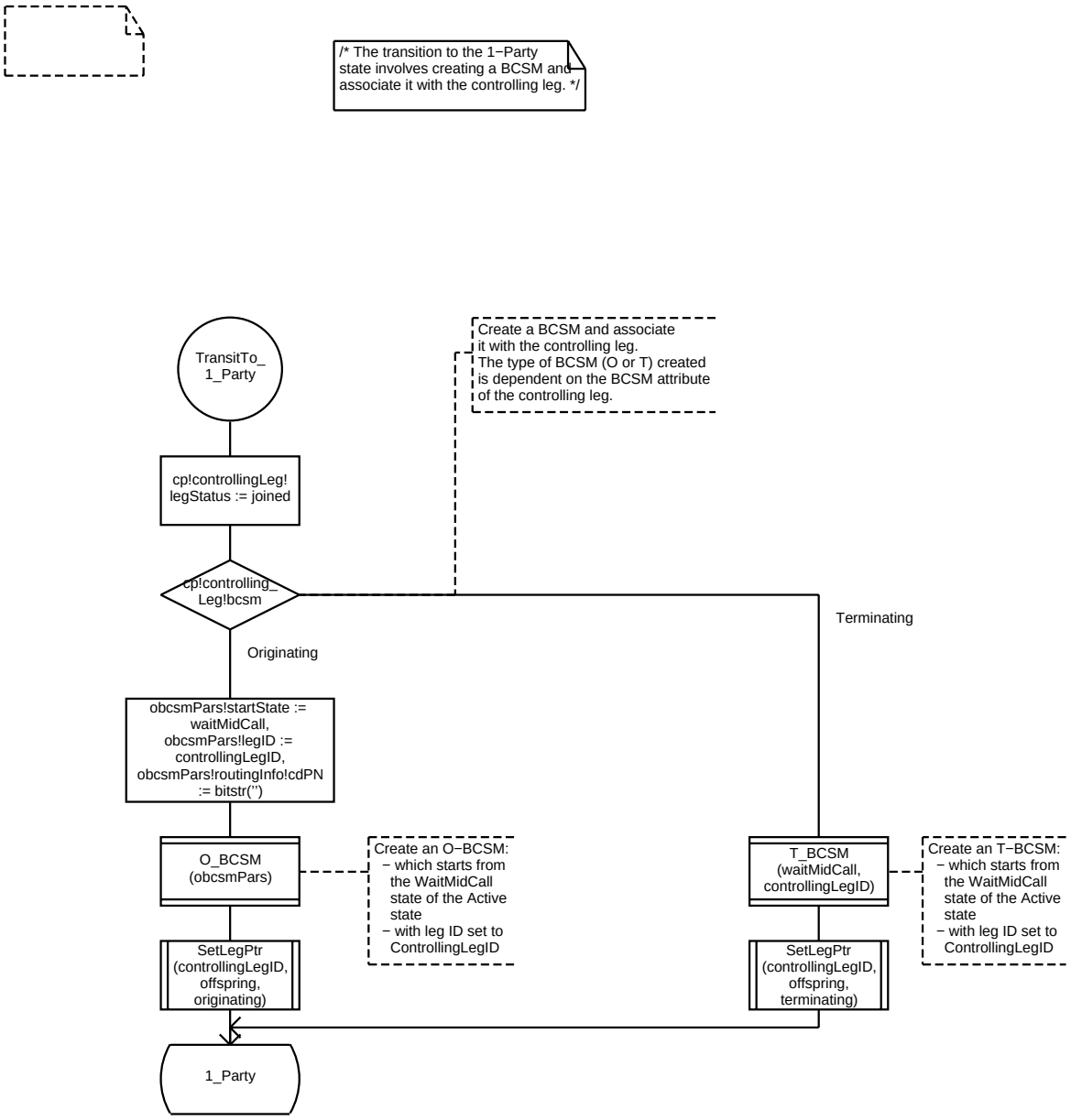

/* Transfer –> Stable−1−Party
Transfer –> Transfer
Note: Processing of atomic operations used by the CSA. */


Transfer
redefined
BCSMStop
(legId,party)
From BCSM
RemoveLeg
(legId)
call
noOfLegs
1
>1
Stable_1_
_Party
Transfer

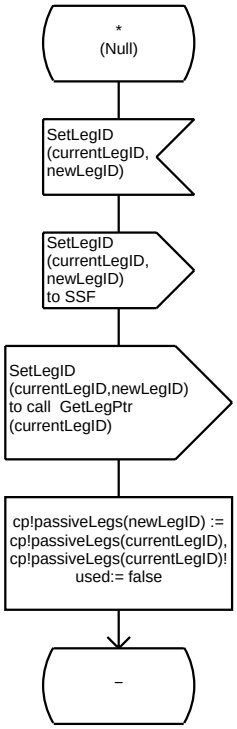

/* Forward –> Forward
Forward –> Originating−1−Party−Setup
Forward –> Stable−1−Party
Forward –> Transfer
Note: Processing of atomic operations used by the CSA. */
Forward
CSExport_
LegReq
(legId)
From CSA process due to a MoveLeg
or SplitLeg operation from the SCF.
legId =
controllingLegID
false
Which leg is
exported?
true
'Error
situation'
Forward
call
GetLegStatus
(legId)
joined
/* A joined leg
was exported. */
pending
/* The pending leg
was exported. */
expLeg := call
ExportLeg
(legId)
'Error
situation'
CSExport_
LegResp
(expLeg)
to CSA
–
call
noOfLegs
1
>1
Originating_
_1_Party__
Setup
Forward

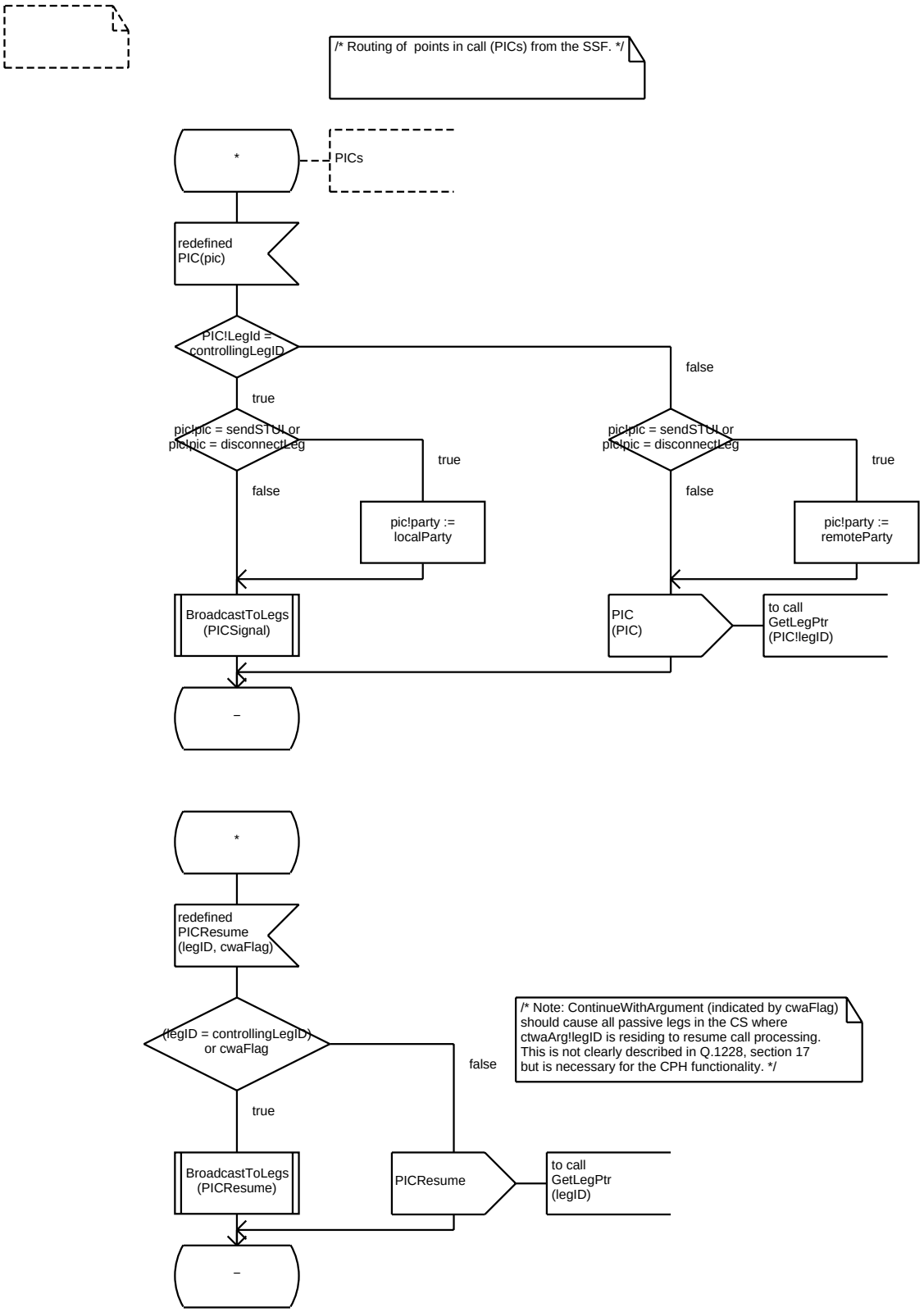

/* Forward −> 1−Party−Setup
Forward −> Forward
Forward −> Stable−1−Party
Forward −> Transfer
Note: Processing of atomic operations used by the CSA. */
Forward
redefined
BCSMStop
(legId,party)
From BCSM
/* Note: The Forward state is composed of one passive pending leg and one or more passive joined legs.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

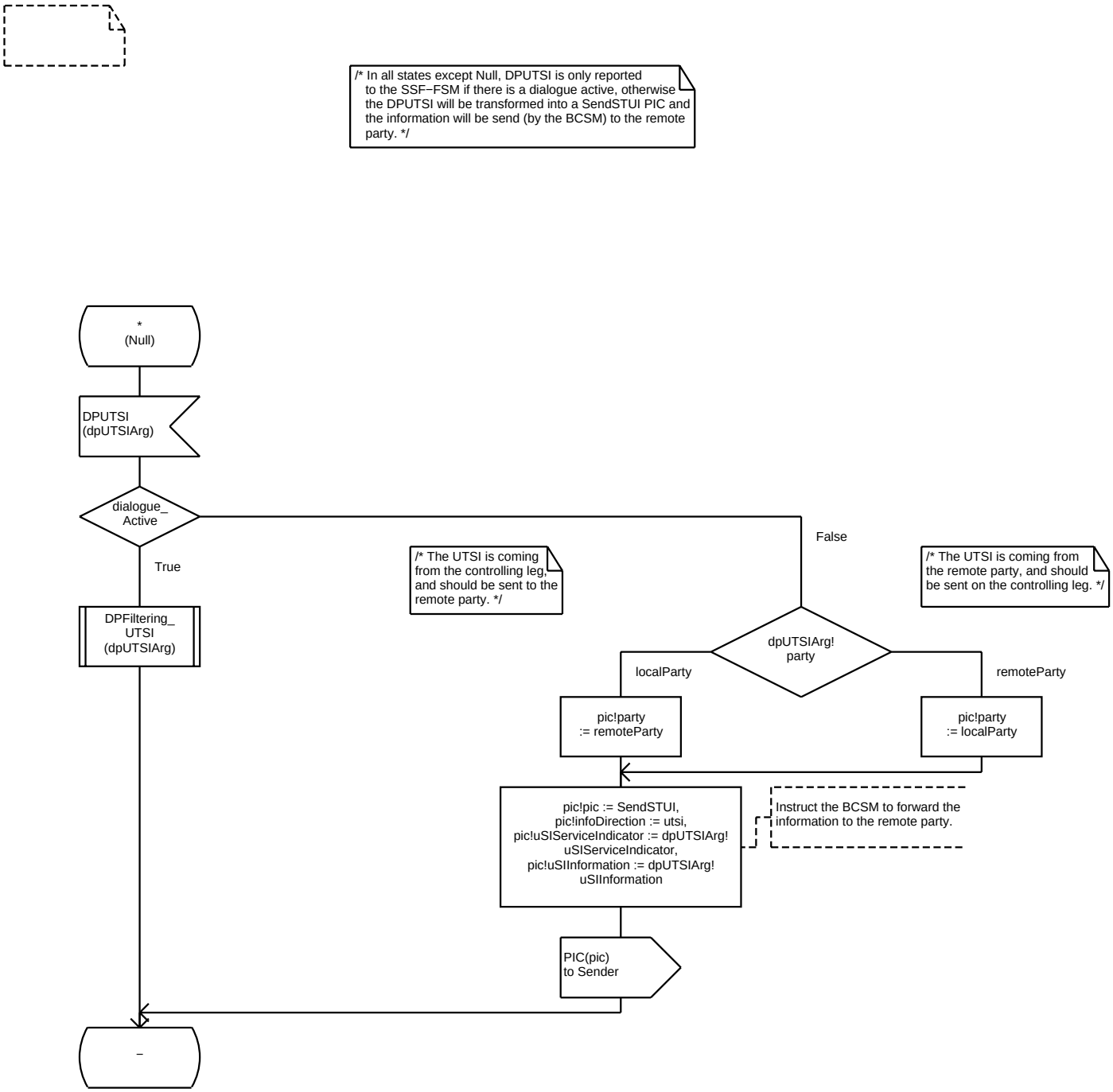

/* In the Transfer and Forward states, the NetworkSuspendReq, NetworkResumeReq and DataReq to/from the BCSMs must be relayed since the controlling leg is surrogate. */
Forward, Transfer
Network_ SuspendReq (nsrArg)
From O−BCSM in the backward direction.
CP! PassiveLegs(leg1)! BCSM
Originating
Terminating
Network_ SuspendInd (nsrArg)
to call GetLegPtr(leg1)
−
Forward, Transfer
Network_ ResumeReq (nsrArg)
From O−BCSM in the backward direction.
CP! PassiveLegs(leg1)! BCSM
Originating
Terminating
Network_ ResumeInd (nsrArg)
to call GetLegPtr(leg1)
−
Forward, Transfer
DataReq (udArg, callFlag)
From O−BCSM in the backward direction, or from the T−BCSM in the forward direction.
callFlag
senderRef /* From O−BCSM */
receiverRef /* From T−BCSM */
DataInd (udArg)
to call GetLegPtr (MkString(I2O(3)))
DataInd (udArg)
to call GetLegPtr(leg1)
−

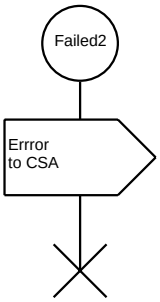

/* The transition to the 1−Party state involves creating a BCSM and associate it with the controlling leg. */
TransitTo_ 1_Party
Create a BCSM and associate it with the controlling leg. The type of BCSM (O or T) created is dependent on the BCSM attribute of the controlling leg.
cp!controllingLeg! legStatus := joined
cp!controlling_ Leg!bcsm
Terminating
Originating
obcsmPars!startState := waitMidCall, obcsmPars!legID := controllingLegID, obcsmPars!routingInfo!cdPN := bitstr('')
O_BCSM (obcsmPars)
Create an O−BCSM: − which starts from the WaitMidCall state of the Active state − with leg ID set to ControllingLegID
T_BCSM (waitMidCall, controllingLegID)
Create an T−BCSM: − which starts from the WaitMidCall state of the Active state − with leg ID set to ControllingLegID
SetLegPtr (controllingLegID, offspring, originating)
SetLegPtr (controllingLegID, offspring, terminating)
1_Party

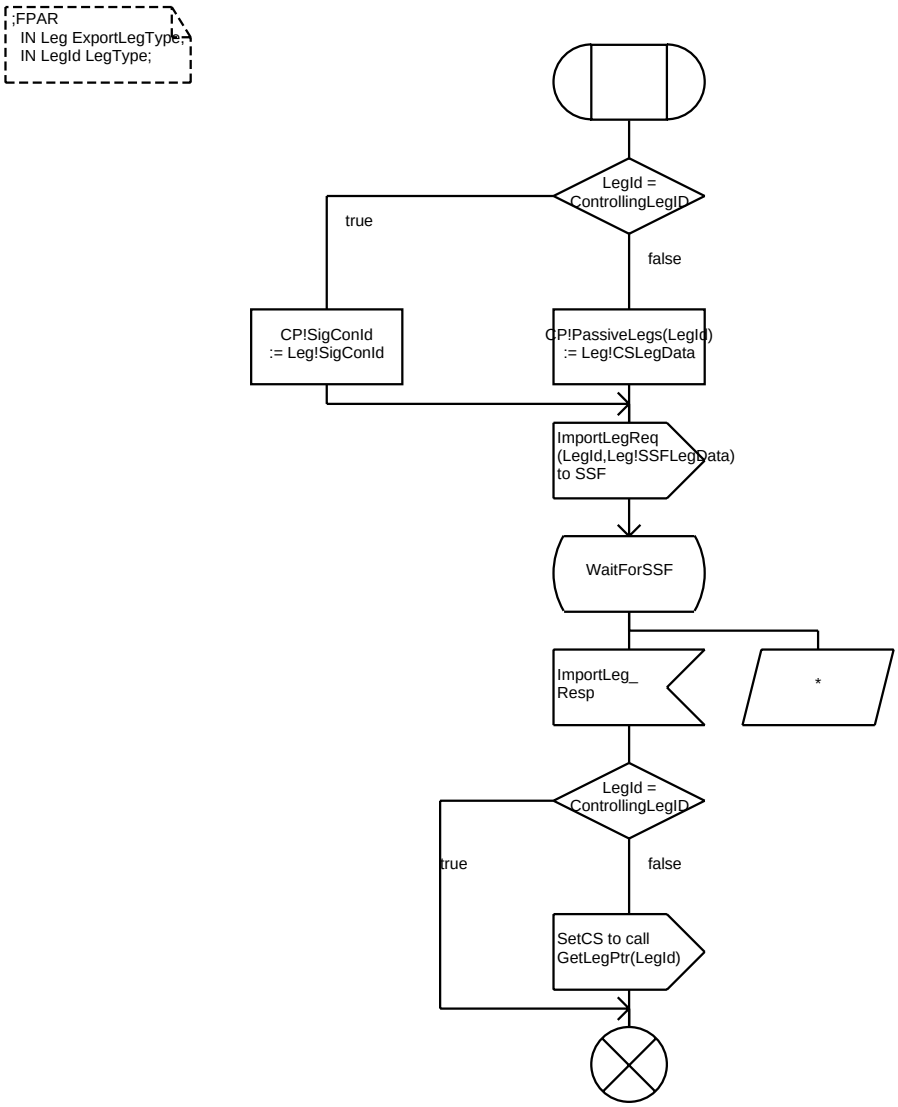
/* When the MoveCallSegments operation is processed by the IH and CSA, the CS may be requested to renumber the legs. In such a case, the CSA sends the signal SetLegID to the CS prior to exporting it into another CSA. */




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


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

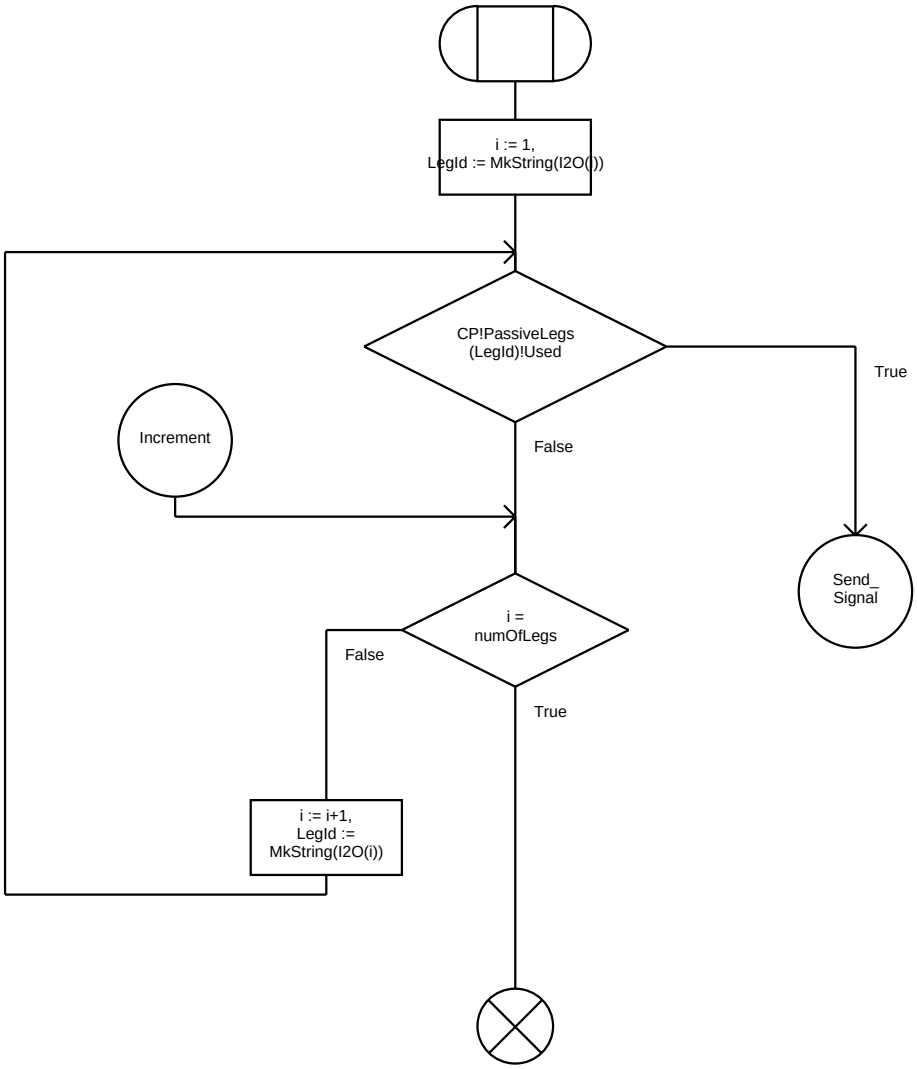

Failed2
Errror
to CSA

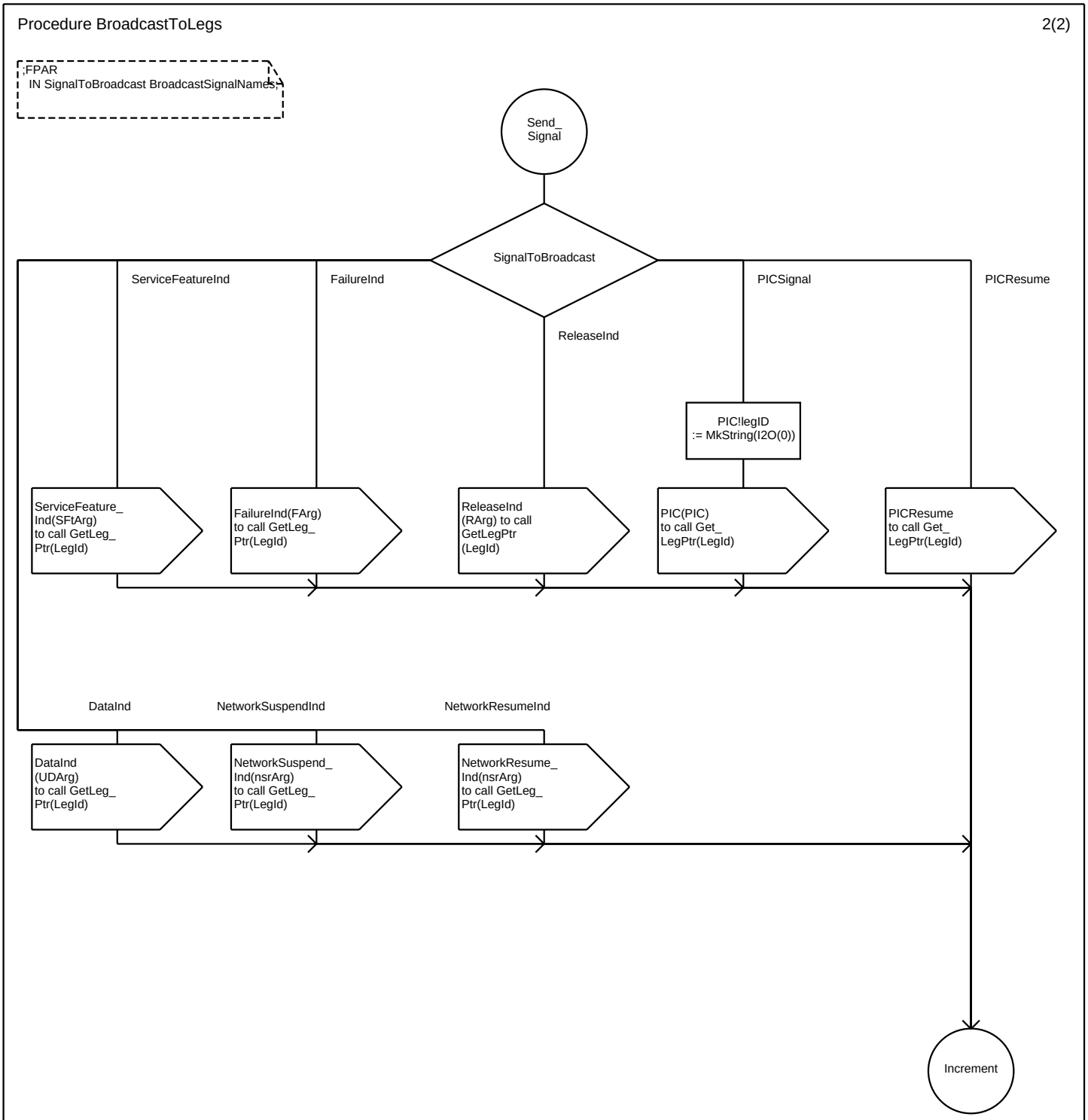

;FPAR
IN Leg ExportLegType;
IN LegId LegType;
LegId = ControllingLegID
true
false
CP!SigConId := Leg!SigConId
CP!PassiveLegs(LegId) := Leg!CSLegData
ImportLegReq (LegId,Leg!SSFLegData) to SSF
WaitForSSF
ImportLeg_ Resp
*
LegId = ControllingLegID
true
false
SetCS to call GetLegPtr(LegId)

Procedure BroadcastToLegs 1(2)

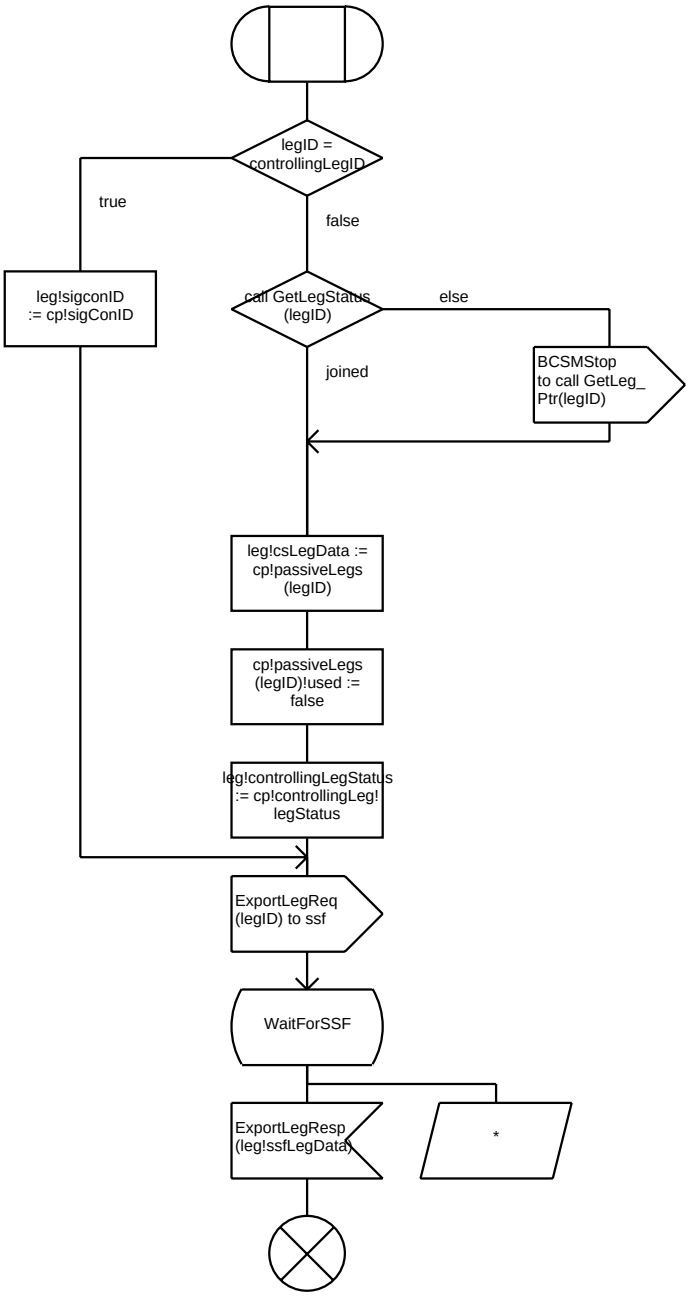
;FPAR
IN SignalToBroadcast BroadcastSignalNames;

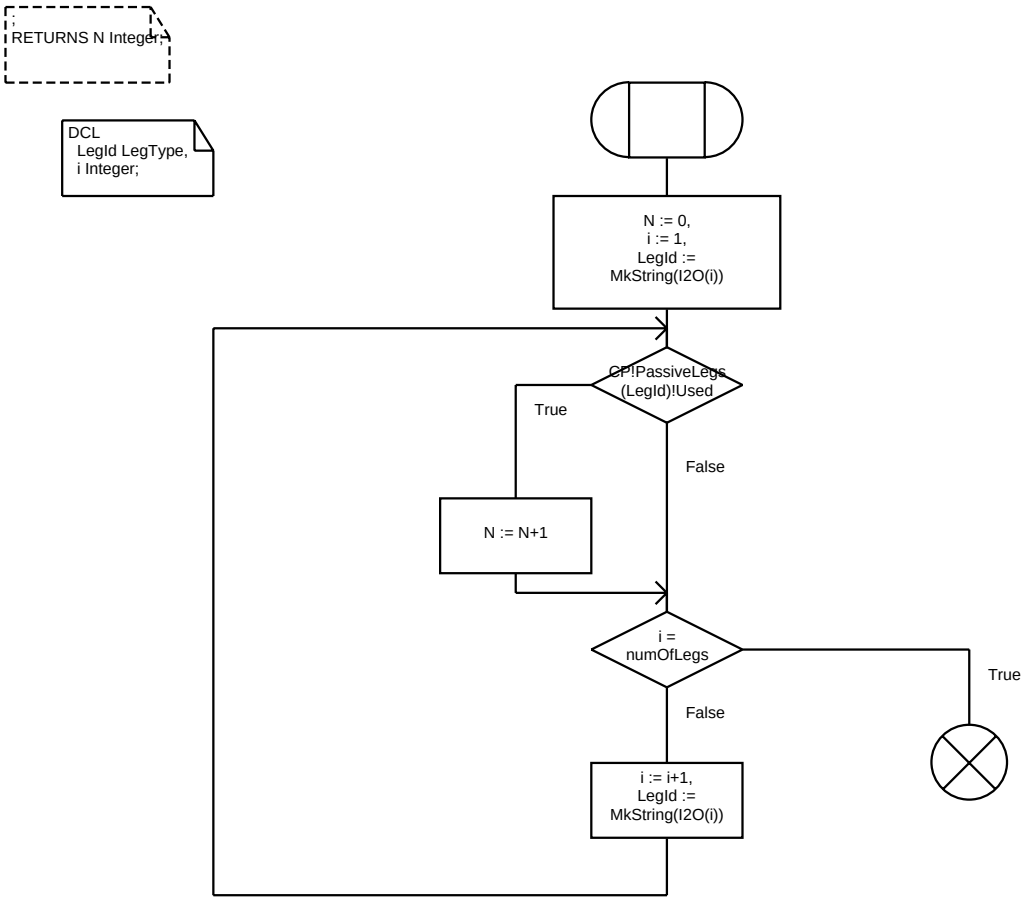
DCL
LegId LegType,
i Integer;

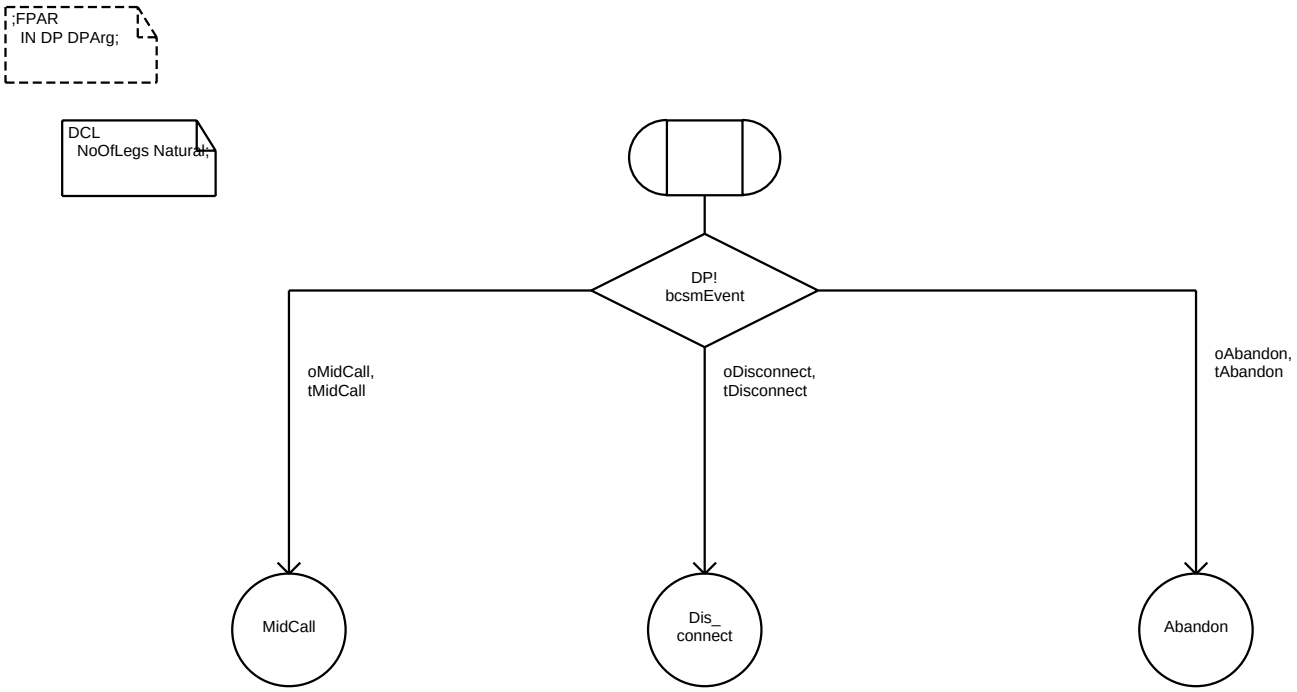


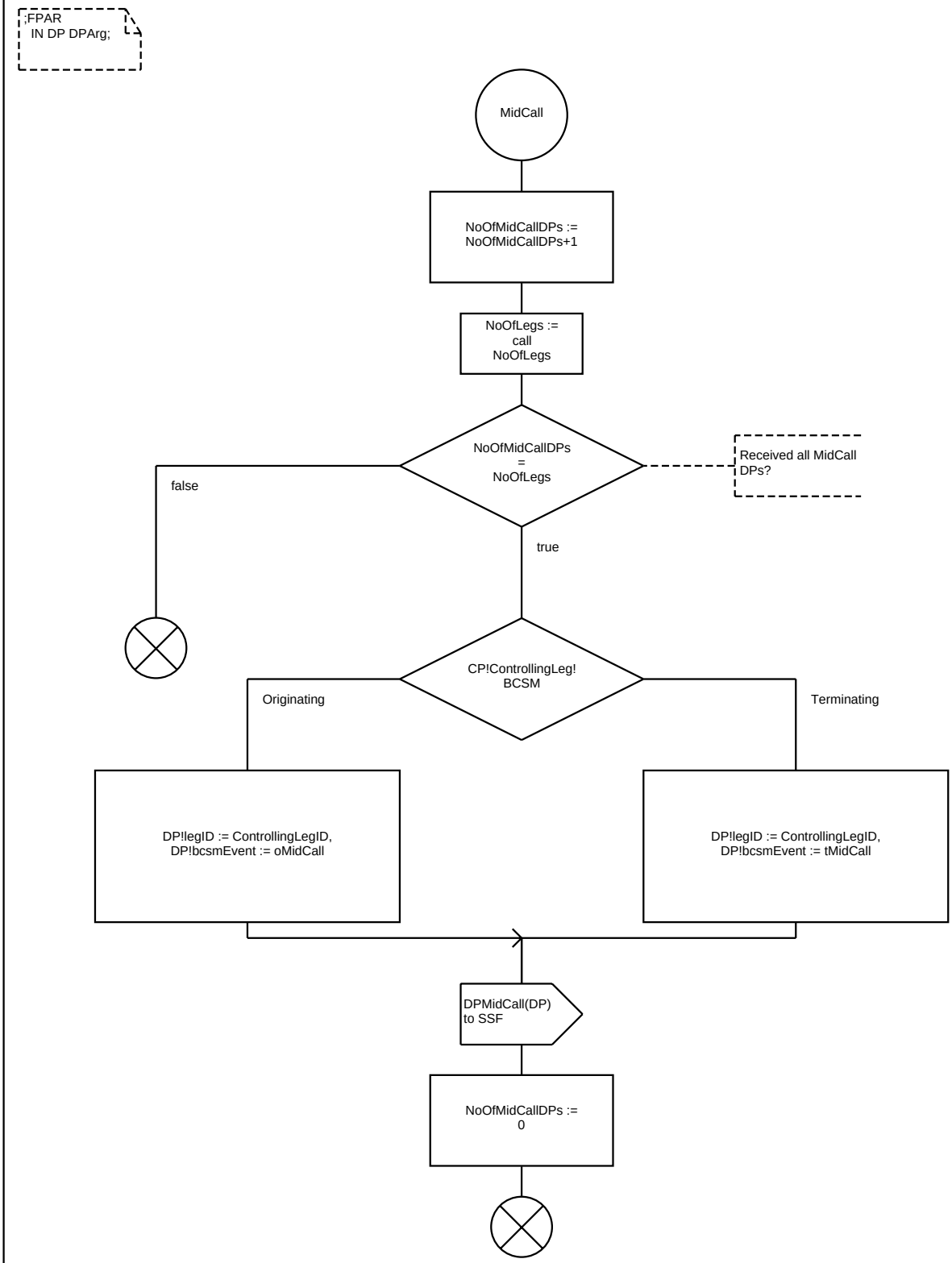

;FPAR
IN SignalToBroadcast BroadcastSignalNames
Send_
Signal
SignalToBroadcast
ServiceFeatureInd
FailureInd
ReleaseInd
PICSignal
PICResume
PIC!legID
:= MkString(I2O(0))
ServiceFeature_
Ind(SFtArg)
to call GetLeg_
Ptr(LegId)
FailureInd(FArg)
to call GetLeg_
Ptr(LegId)
ReleaseInd
(RArg) to call
GetLegPtr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

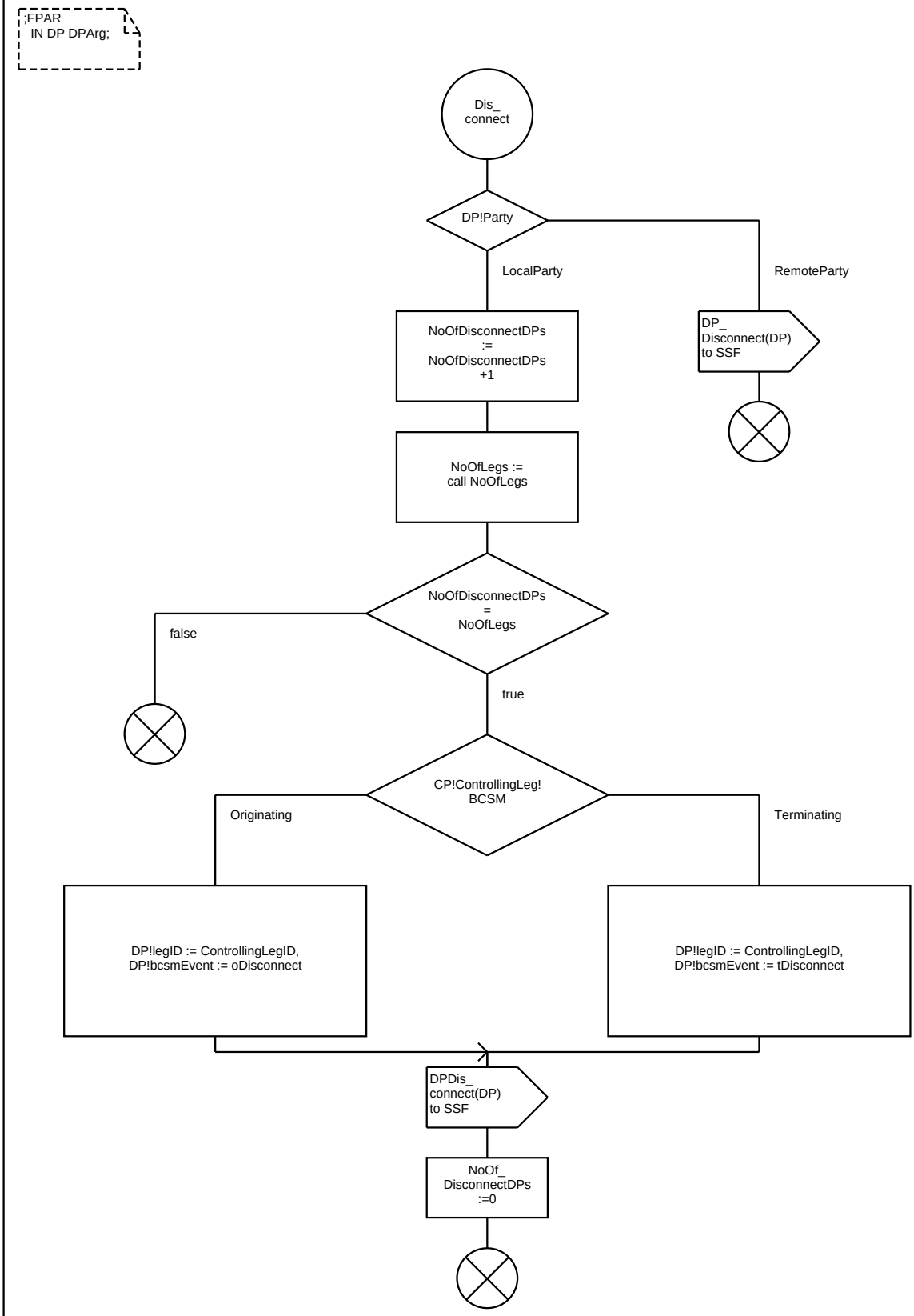
;FPAR
LegId LegType;
RETURNS Leg ExportLegType;

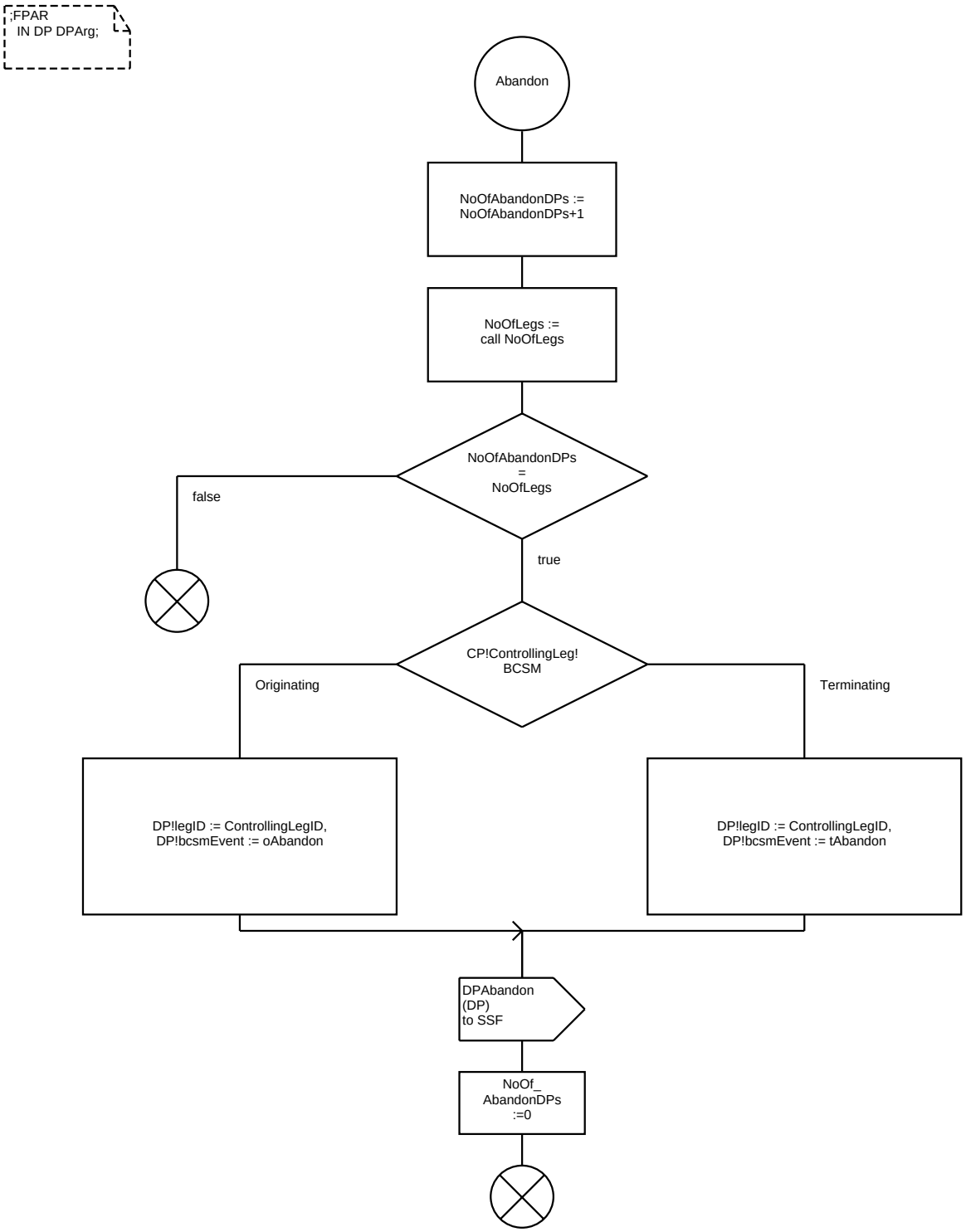



;
RETURNS N Integer;
DCL
LegId LegType,
i Integer;
N := 0,
i := 1,
LegId :=
MkString(I2O(i))
CP!PassiveLegs
(LegId)!Used
True
False
N := N+1
i =
numOfLegs
True
False
i := i+1,
LegId :=
MkString(I2O(i))

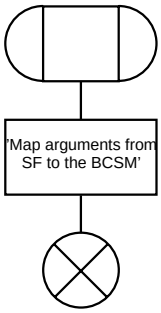

;FPAR
IN DP DPArg;
DCL
NoOfLegs Natural;
DP!
bcsmEvent
oMidCall,
tMidCall
oDisconnect,
tDisconnect
oAbandon,
tAbandon
MidCall
Dis_
connect
Abandon

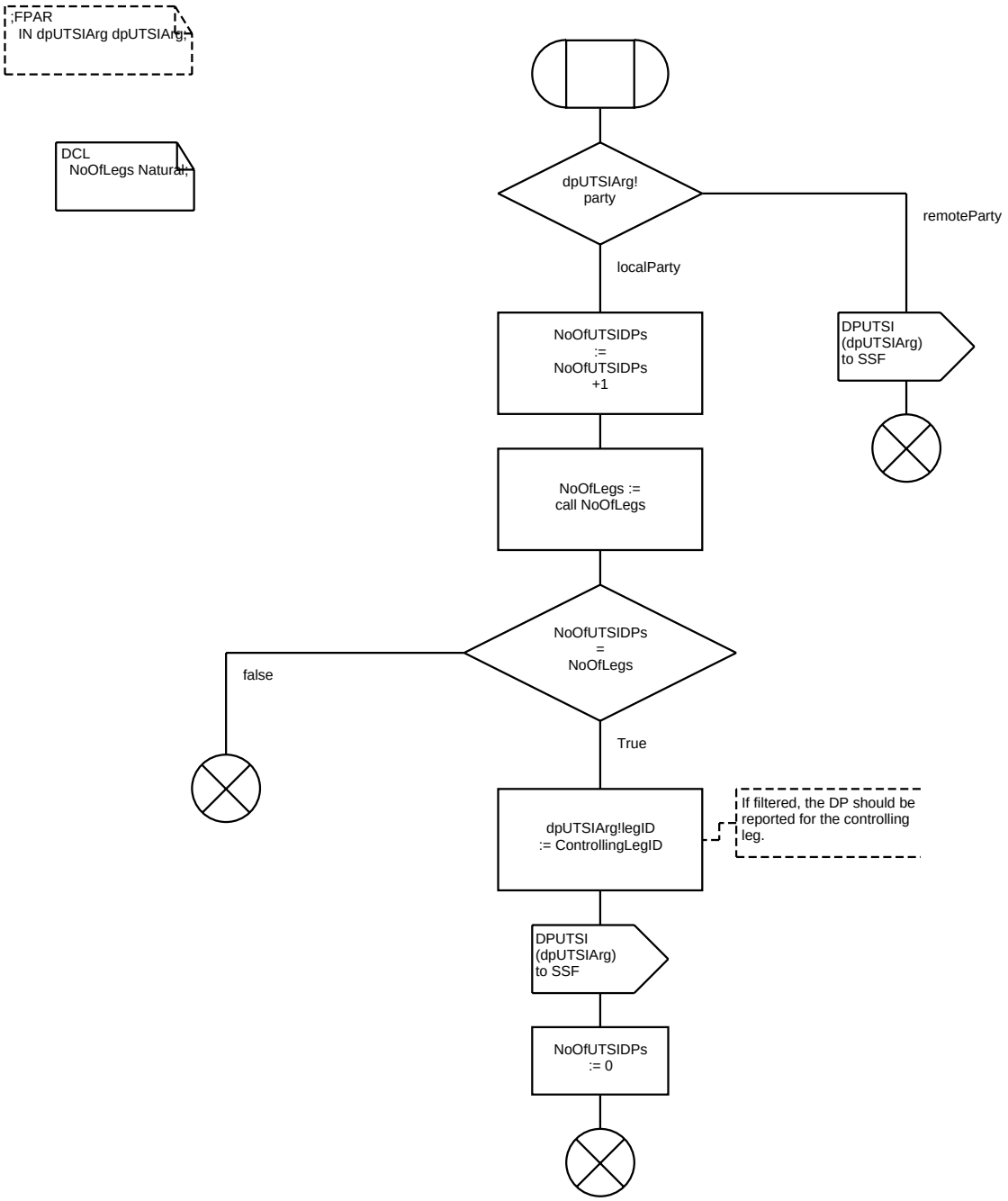

;FPAR
IN DP DPArg;
MidCall
NoOfMidCallDPs :=
NoOfMidCallDPs+1
NoOfLegs :=
call
NoOfLegs
NoOfMidCallDPs
=
NoOfLegs
Received all MidCall
DPs?
false
true
CP!ControllingLeg!
BCSM
Originating
Terminating
DP!legID := ControllingLegID,
DP!bcsmEvent := oMidCall
DP!legID := ControllingLegID,
DP!bcsmEvent := tMidCall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


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

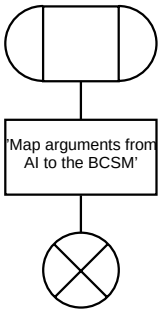

;FPAR
IN DP DPArg;
Abandon
NoOfAbandonDPs :=
NoOfAbandonDPs+1
NoOfLegs :=
call NoOfLegs
NoOfAbandonDPs
=
NoOfLegs
false
true
CP!ControllingLeg!
BCSM
Originating
Terminating
DP!legID := ControllingLegID,
DP!bcsmEvent := oAbandon
DP!legID := ControllingLegID,
DP!bcsmEvent := tAbandon
DPAbandon
(DP)
to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

```
;FPAR
  IN sfArg SelectFacilityArg;
  RETURNS obcsmPars OBCSMPars;
```

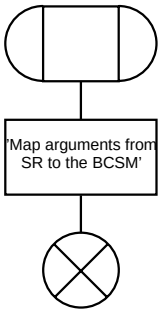



;FPAR
IN dpUTSIArg dpUTSIArg;
DCL
NoOfLegs Natural;
dpUTSIArg!
party
remoteParty
localParty
NoOfUTSIDPs
:=
NoOfUTSIDPs
+1
DPUTSI
(dpUTSIArg)
to SSF
NoOfLegs :=
call NoOfLegs
NoOfUTSIDPs
=
NoOfLegs
false
True
dpUTSIArg!legID
:= ControllingLegID
If filtered, the DP should be reported for the controlling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

```
;FPAR
  IN aiArg AnalyseInformationArg;
  RETURNS obcsmPars OBCSMPars;
```



;FPAR
IN srArg SelectRouteArg;
RETURNS obcsmPars OBCSMPars;




;FPAR
IN srArg SelectRouteArg;
RETURNS obcsmPars OBCSMPars;

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

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

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

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

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

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

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

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

/* Operations for exporting and importing legs. */

Export_ Event_ Record

Import_ Event_ Record

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

IsUTSI_ Armed

IsFacility_ Armed

/* For IN CS−2 the procedures for handling of the DP table and the procedure for checking ACG are redefined. */

redefined Initialise_ DPTable

/* IN CS−2 operations */

ProcessContinue_ WithArgument

Process_ Request_ ReportUTSI

Process_ Disconnect_ Leg

Process_ SendSTUI

Process_ Report_ UTSI

redefined ProcessDP_ Specific

Process_ ReportFacility

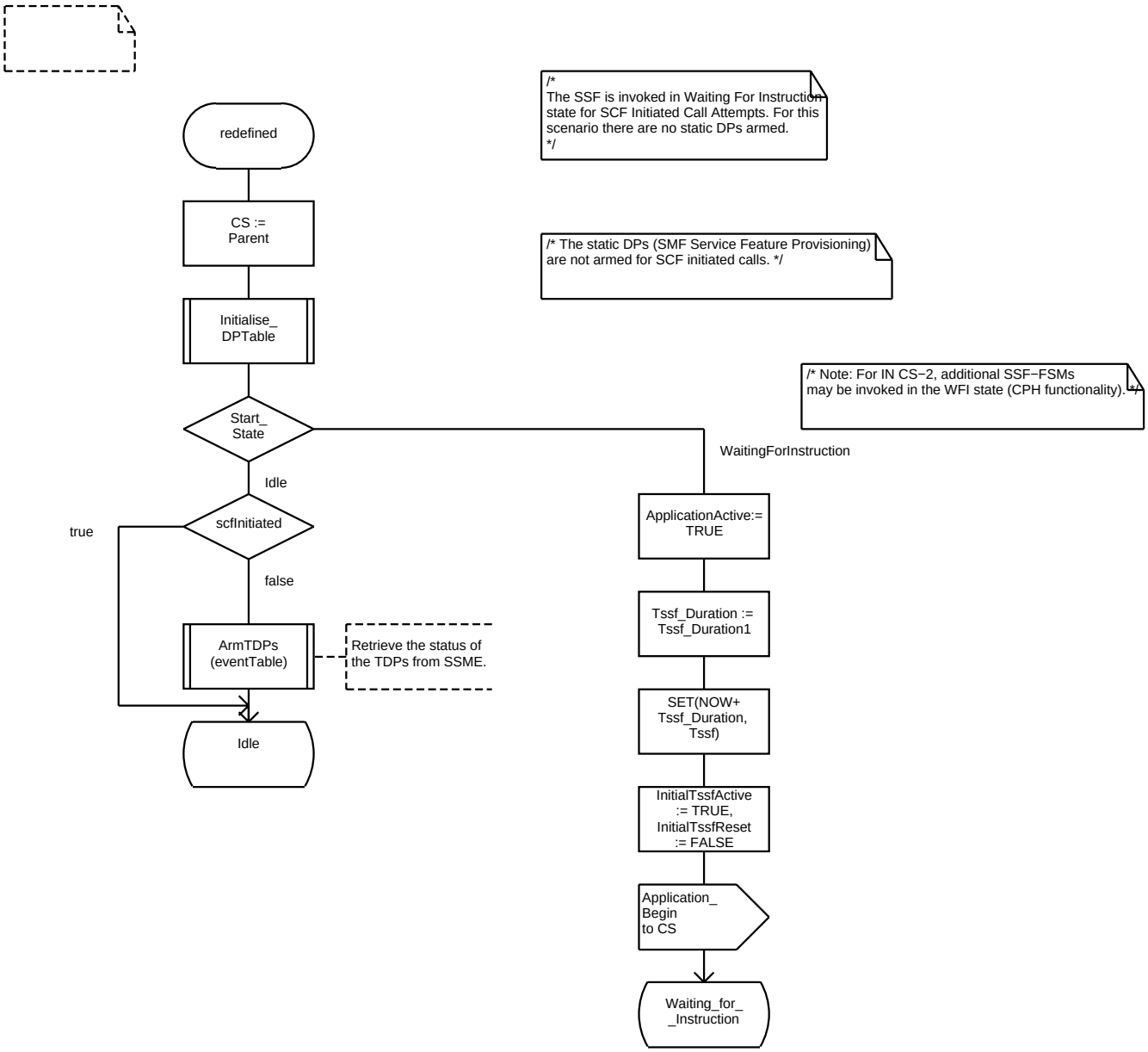
ProcessRequest_ ReportFacility_ Event

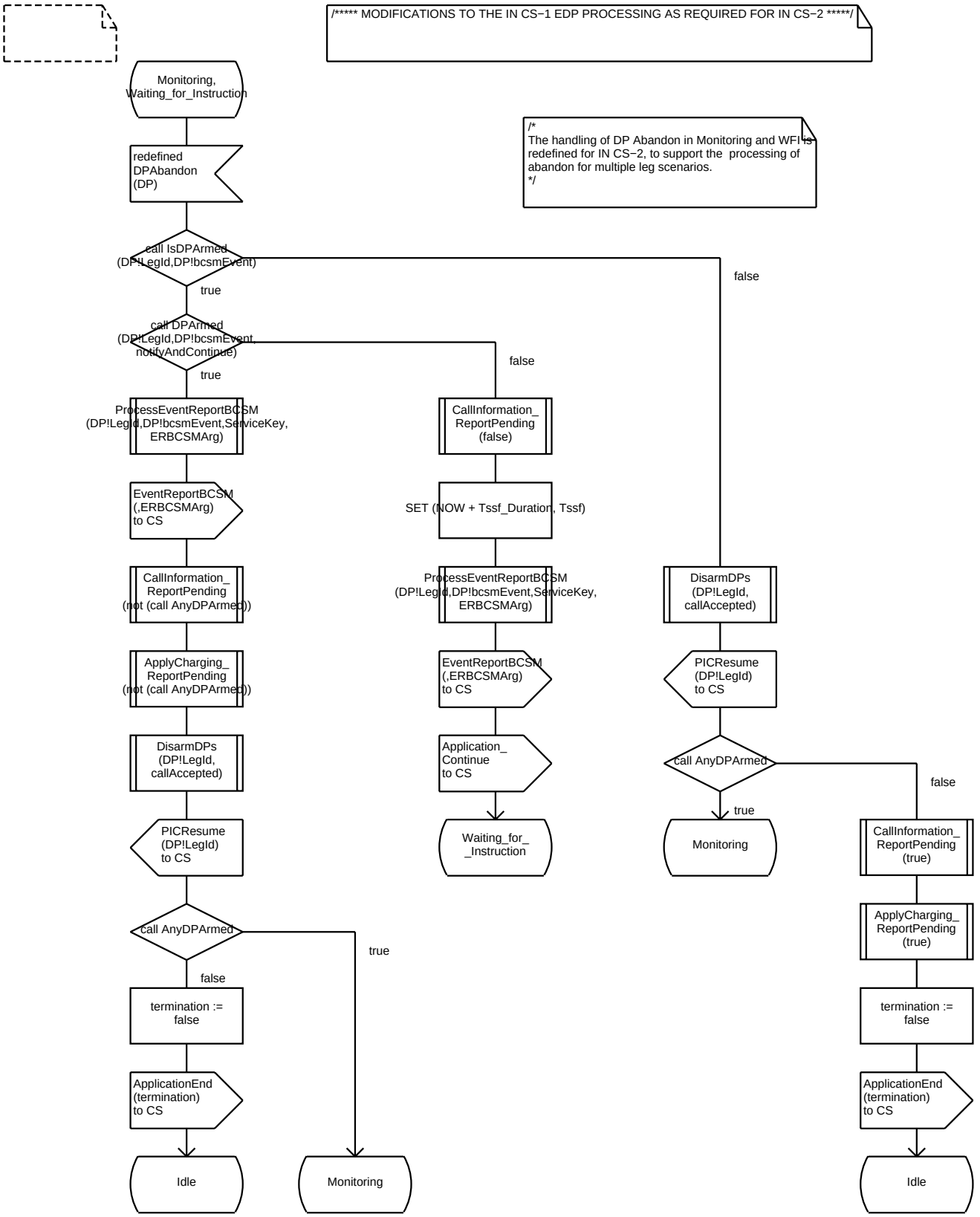
Process_ SendFacility

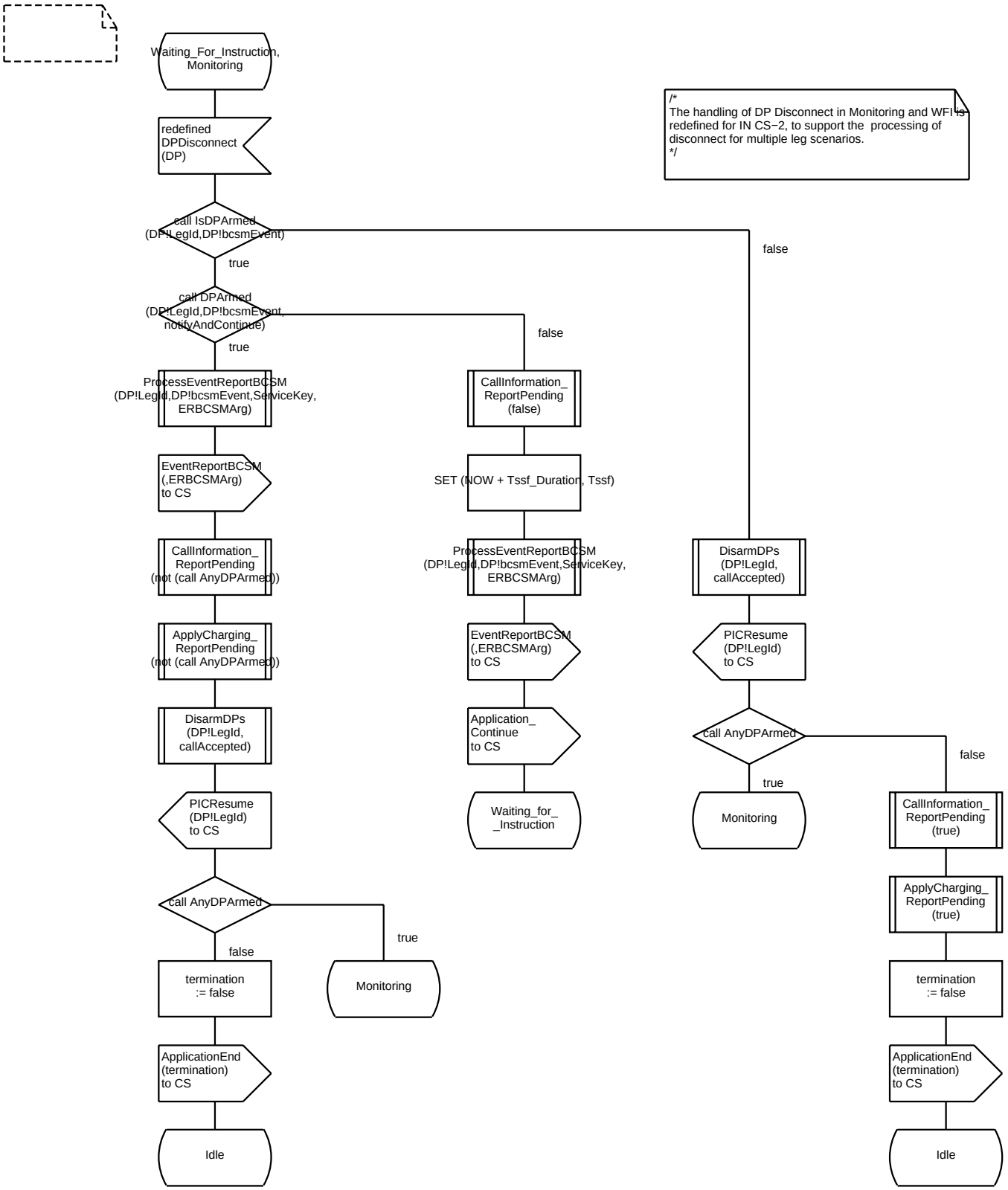
CS

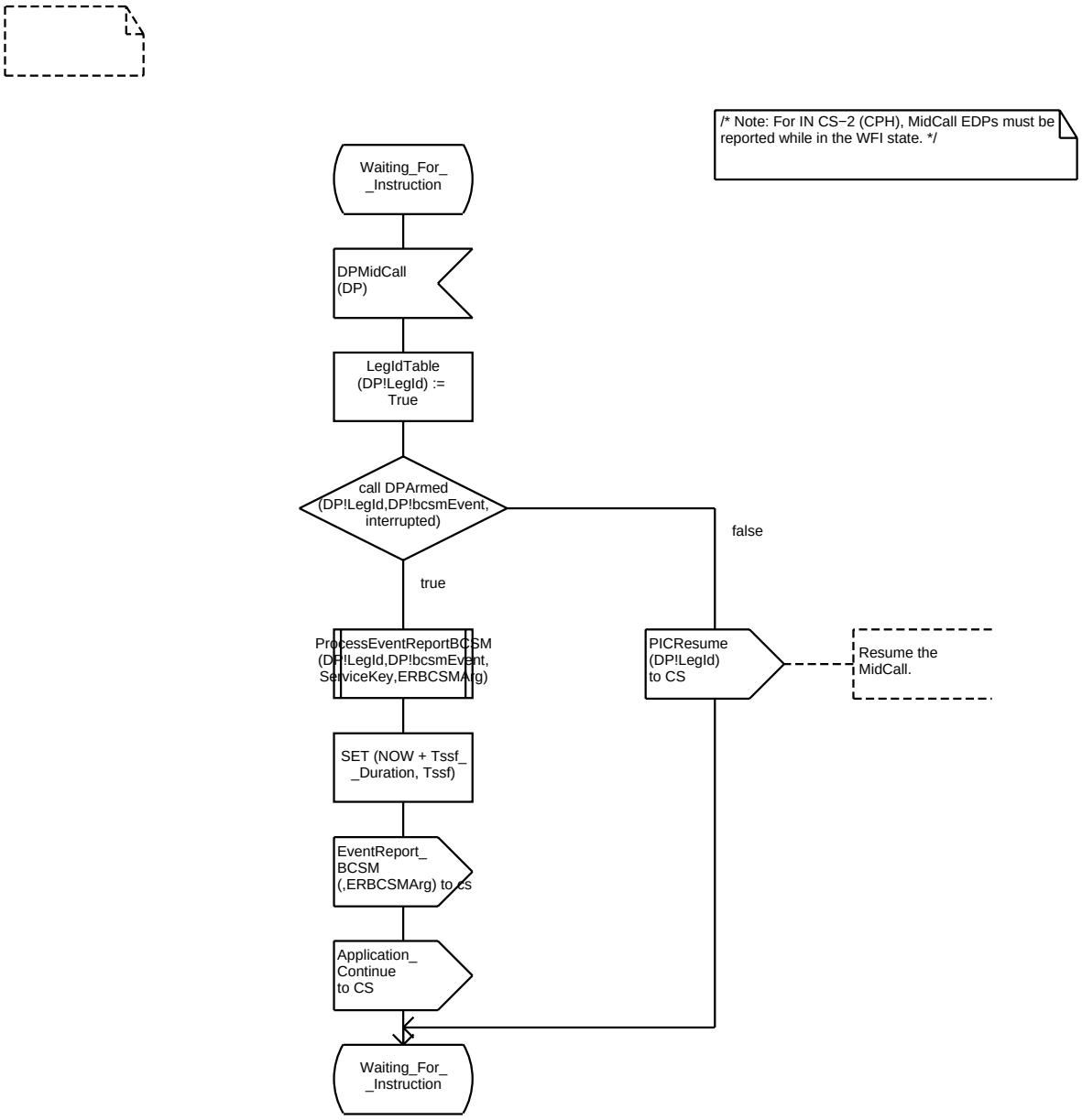
(SSF2_IN)

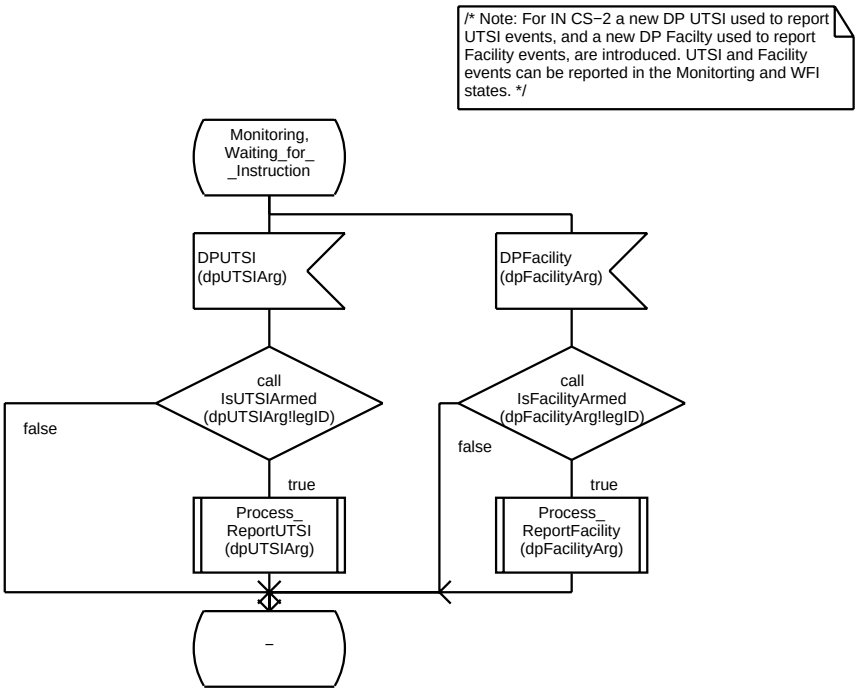
(SSF2_OUT)

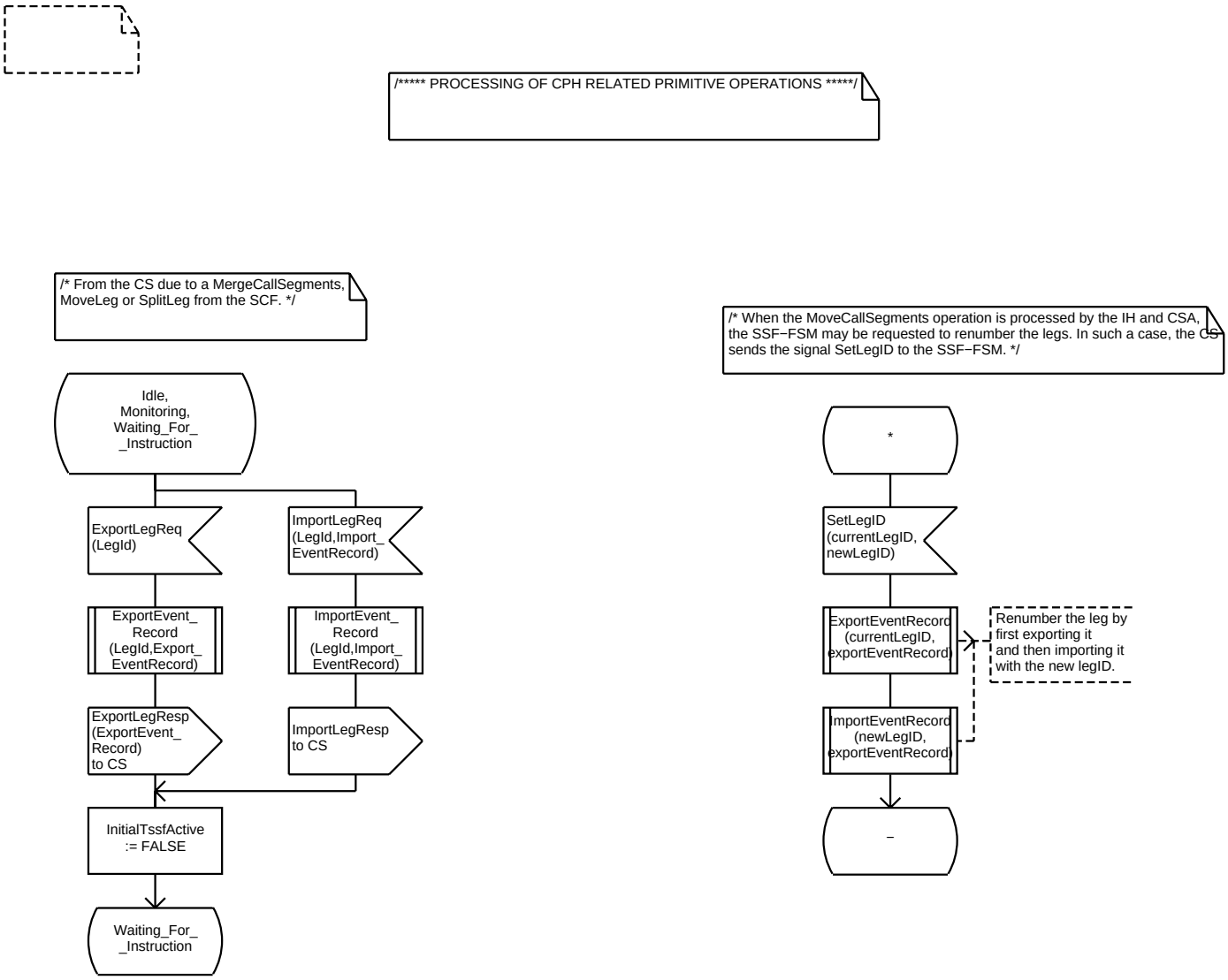

/*
The SSF is invoked in Waiting For Instruction state for SCF Initiated Call Attempts. For this scenario there are no static DPs armed.
*/
redefined
CS := Parent
/* The static DPs (SMF Service Feature Provisioning) are not armed for SCF initiated calls. */
Initialise_ DPTable
/* Note: For IN CS−2, additional SSF−FSMs may be invoked in the WFI state (CPH functionality). */
Start_ State
WaitingForInstruction
Idle
true
scfInitiated
ApplicationActive:= TRUE
false
Tssf_Duration := Tssf_Duration1
ArmTDPs (eventTable)
Retrieve the status of the TDPs from SSME.
SET(NOW+ Tssf_Duration, Tssf)
Idle
InitialTssfActive := TRUE, InitialTssfReset := FALSE
Application_ Begin to CS
Waiting_for_ _Instruction

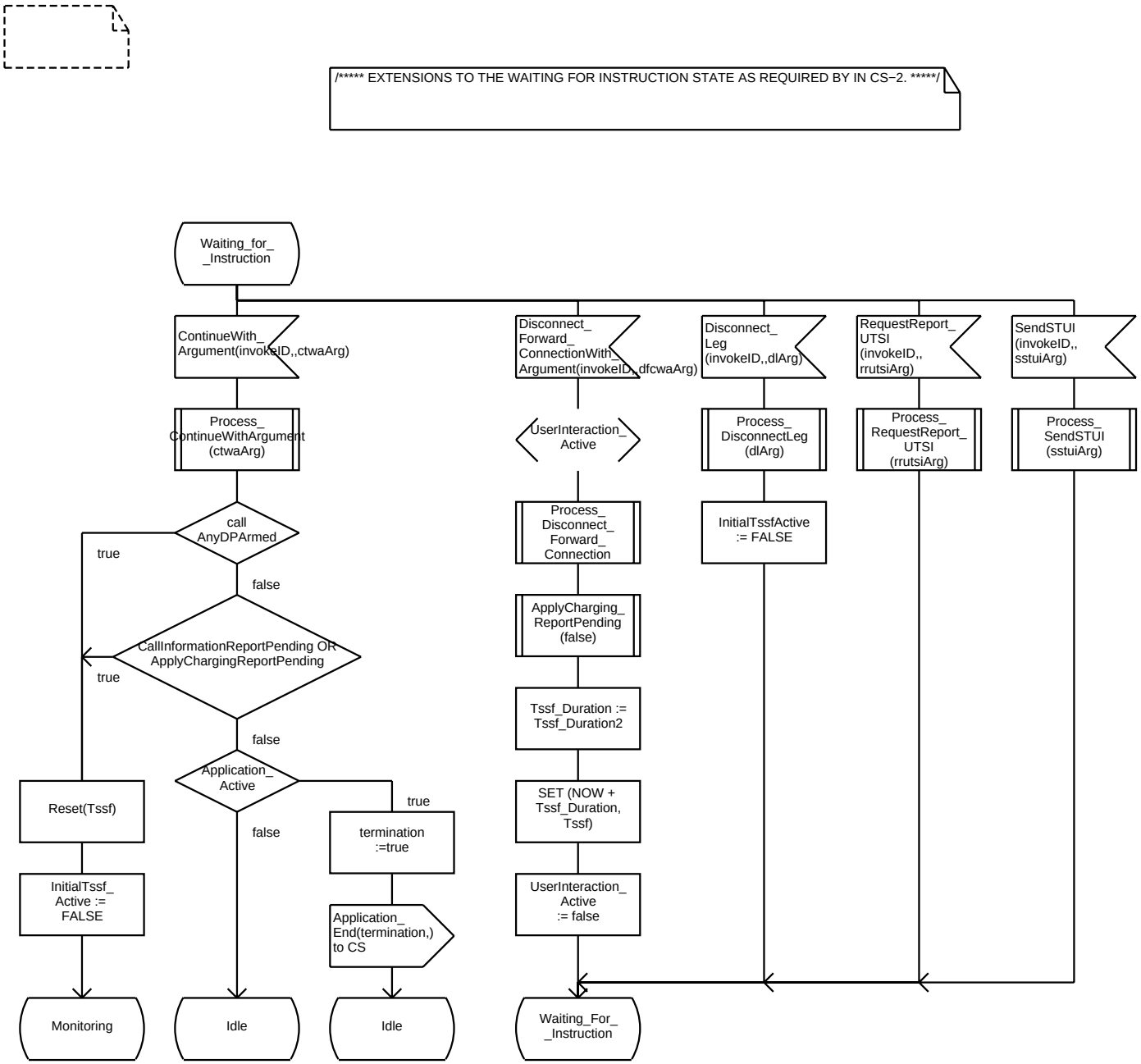

/***** MODIFICATIONS TO THE IN CS−1 EDP PROCESSING AS REQUIRED FOR IN CS−2 *****/
Monitoring, Waiting_for_Instruction
/*
The handling of DP Abandon in Monitoring and WFI is redefined for IN CS−2, to support the processing of abandon for multiple leg scenarios.
*/
redefined DPAbandon (DP)
call IsDPArmed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

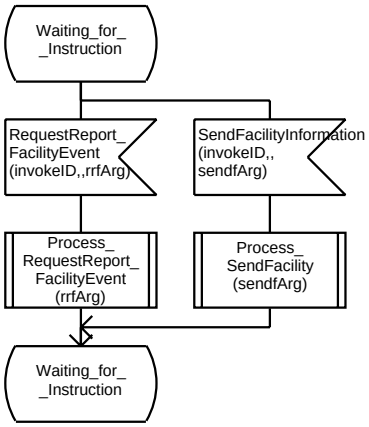

Waiting_For_Instruction, Monitoring
redefined DPDisconnect (DP)
/*
The handling of DP Disconnect in Monitoring and WFI is redefined for IN CS−2, to support the processing of disconnect for multiple leg scenarios.
*/
call IsDPArmed (DP!LegId,DP!bcsmEvent)
false
true
call DPArmed (DP!LegId,DP!bcsmEvent, notifyAndContinue)
false
true
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent,ServiceKey, ERBCSMArg)
CallInformation_ ReportPending (false)
EventReportBCSM (,ERBCSMArg) to CS
SET (NOW + Tssf_Duration, Tssf)
CallInformation_ ReportPending (not (call AnyDPArmed))
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent,ServiceKey, ERBCSMArg)
DisarmDPs (DP!LegId, callAccepted)
ApplyCharging_ ReportPending (not (call AnyDPArmed))
EventReportBCSM (,ERBCSMArg) to CS
PICResume (DP!LegId) to CS
DisarmDPs (DP!LegId, callAccepted)
Application_ Continue to CS
call AnyDPArmed
false
PICResume (DP!LegId) to CS
Waiting_for_ _Instruction
true
Monitoring
CallInformation_ ReportPending (true)
call AnyDPArmed
true
false
ApplyCharging_ ReportPending (true)
termination := false
Monitoring
termination := false
ApplicationEnd (termination) to CS
ApplicationEnd (termination) to CS
Idle
Idle

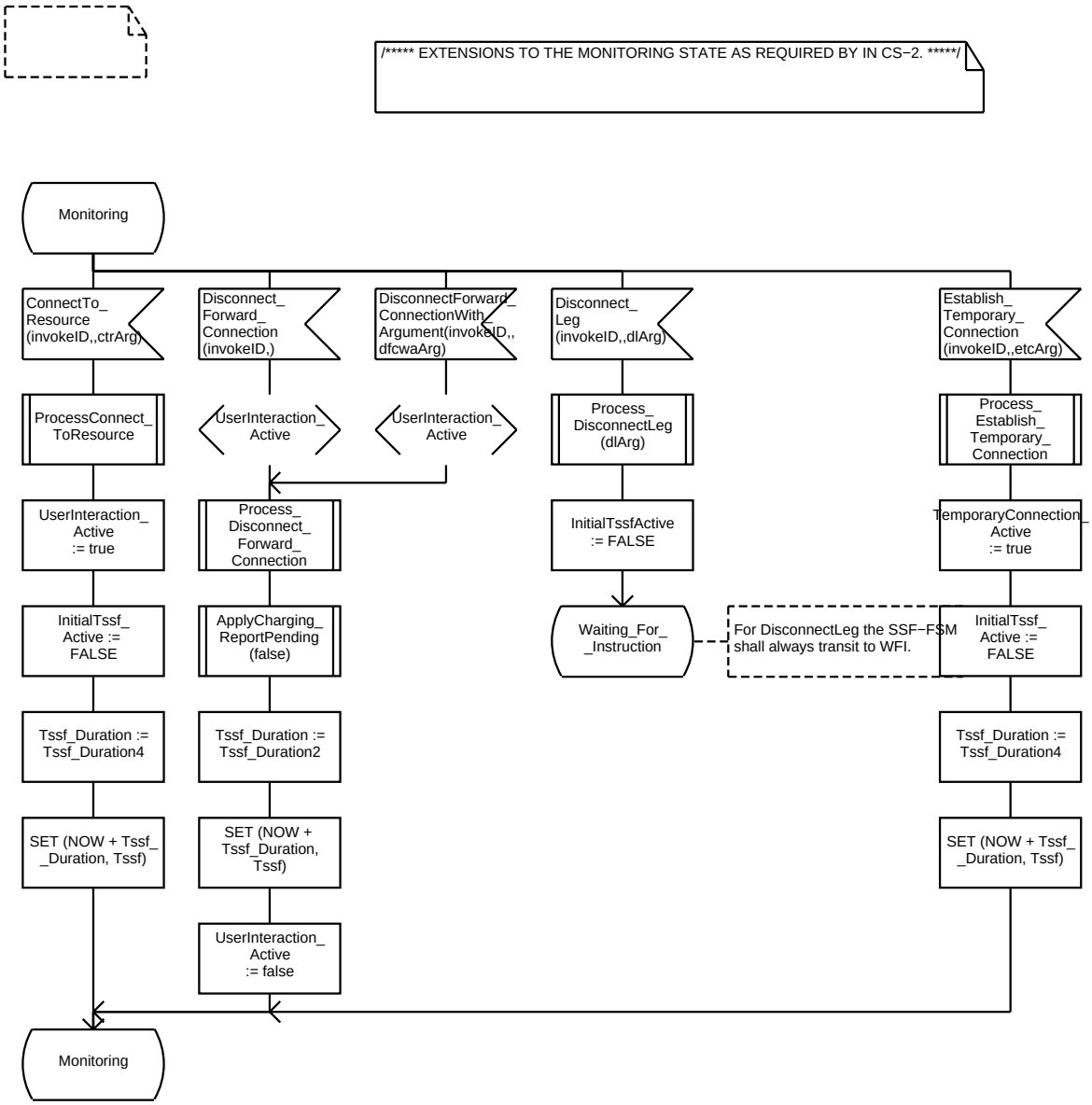

/* Note: For IN CS−2 (CPH), MidCall EDPs must be reported while in the WFI state. */
Waiting_For_ _Instruction
DPMidCall (DP)
LegIdTable (DP!LegId) := True
call DPArmed (DP!LegId,DP!bcsmEvent, interrupted)
false
true
ProcessEventReportBCSM (DP!LegId,DP!bcsmEvent, ServiceKey,ERBCSMArg)
PICResume (DP!LegId) to CS
Resume the MidCall.
SET (NOW + Tssf_ _Duration, Tssf)
EventReport_ BCSM (,ERBCSMArg) to cs
Application_ Continue to CS
Waiting_For_ _Instruction

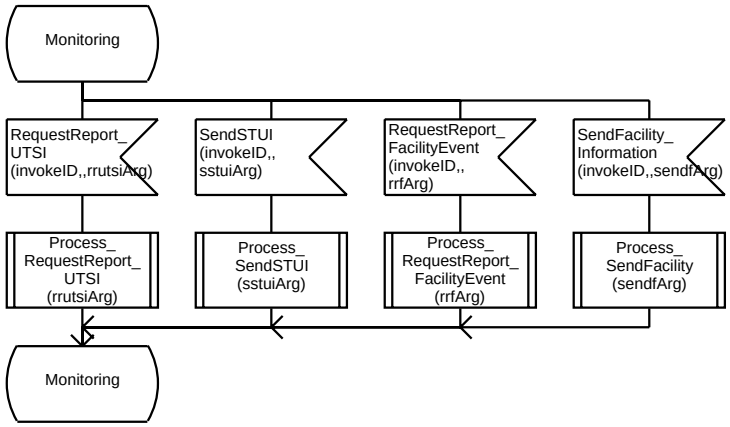

/* Note: For IN CS−2 a new DP UTSI used to report UTSI events, and a new DP Facilty used to report Facility events, are introduced. UTSI and Facility events can be reported in the Monitorting and WFI states. */
Monitoring, Waiting_for_ _Instruction
DPUTSI (dpUTSIArg)
DPFacility (dpFacilityArg)
call IsUTSIArmed (dpUTSIArg!legID)
call IsFacilityArmed (dpFacilityArg!legID)
false
false
true
true
Process_ ReportUTSI (dpUTSIArg)
Process_ ReportFacility (dpFacilityArg)
–


/***** PROCESSING OF CPH RELATED PRIMITIVE OPERATIONS *****/
/* From the CS due to a MergeCallSegments, MoveLeg or SplitLeg from the SCF. */
Idle, Monitoring, Waiting_For_ _Instruction
ExportLegReq (LegId)
ImportLegReq (LegId,Import_ EventRecord)
ExportEvent_ Record (LegId,Export_ EventRecord)
ImportEvent_ Record (LegId,Import_ EventRecord)
ExportLegResp (ExportEvent_ Record) to CS
ImportLegResp to CS
InitialTssfActive := FALSE
Waiting_For_ _Instruction
/* When the MoveCallSegments operation is processed by the IH and CSA, the SSF−FSM may be requested to renumber the legs. In such a case, the CS sends the signal SetLegID to the SSF−FSM. */
*
SetLegID (currentLegID, newLegID)
ExportEventRecord (currentLegID, exportEventRecord)
Renumber the leg by first exporting it and then importing it with the new legID.
ImportEventRecord (newLegID, exportEventRecord)
−


/***** EXTENSIONS TO THE WAITING FOR INSTRUCTION STATE AS REQUIRED BY IN CS−2. *****/
Waiting_for_ _Instruction
ContinueWith_ Argument(invokeID,,ctwaArg)
Disconnect_ Forward_ ConnectionWith_ Argument(invokeID,,dfcwaArg)
Disconnect_ Leg (invokeID,,dlArg)
RequestReport_ UTSI (invokeID,, rrutsiArg)
SendSTUI (invokeID,, sstuiArg)
Process_ ContinueWithArgument (ctwaArg)
UserInteraction_ Active
Process_ DisconnectLeg (dlArg)
Process_ RequestReport_ UTSI (rrutsiArg)
Process_ SendSTUI (sstuiArg)
call AnyDPArmed
true
false
Process_ Disconnect_ Forward_ Connection
InitialTssfActive := FALSE
CallInformationReportPending OR ApplyChargingReportPending
true
ApplyCharging_ ReportPending (false)
false
Tssf_Duration := Tssf_Duration2
Application_ Active
true
false
Reset(Tssf)
termination :=true
SET (NOW + Tssf_Duration, Tssf)
InitialTssf_ Active := FALSE
Application_ End(termination,) to CS
UserInteraction_ Active := false
Monitoring
Idle
Idle
Waiting_For_ _Instruction


Waiting_for_
_Instruction
RequestReport_
FacilityEvent
(invokeID,,rrfArg)
SendFacilityInformation
(invokeID,,
sendfArg)
Process_
RequestReport_
FacilityEvent
(rrfArg)
Process_
SendFacility
(sendfArg)
Waiting_for_
_Instruction

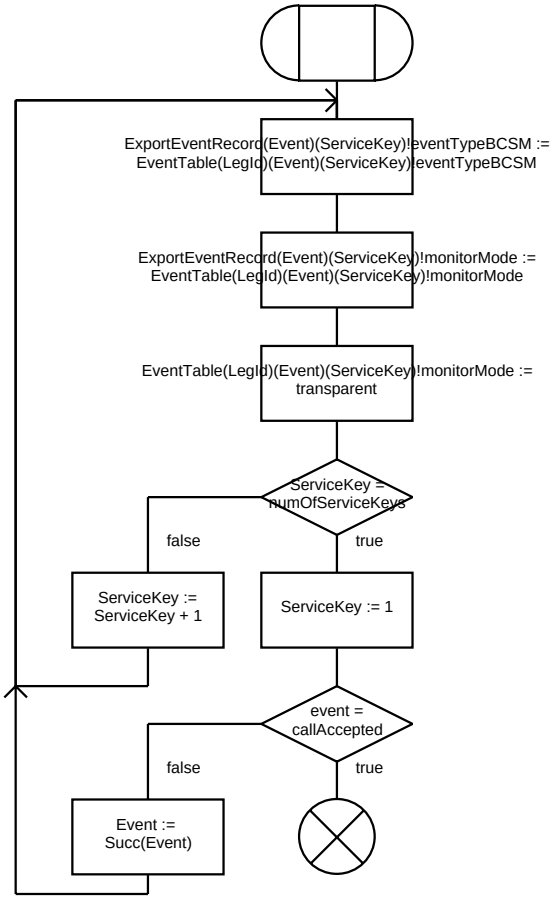

/***** EXTENSIONS TO THE MONITORING STATE AS REQUIRED BY IN CS−2. *****/
Monitoring
ConnectTo_ Resource (invokeID,,ctrArg)
Disconnect_ Forward_ Connection (invokeID,)
DisconnectForward_ ConnectionWith_ Argument(invokeID,, dfcwaArg)
Disconnect_ Leg (invokeID,,dlArg)
Establish_ Temporary_ Connection (invokeID,,etcArg)
ProcessConnect_ ToResource
UserInteraction_ Active
UserInteraction_ Active
Process_ DisconnectLeg (dlArg)
Process_ Establish_ Temporary_ Connection
UserInteraction_ Active := true
Process_ Disconnect_ Forward_ Connection
InitialTssfActive := FALSE
TemporaryConnection_ Active := true
InitialTssf_ Active := FALSE
ApplyCharging_ ReportPending (false)
Waiting_For_ _Instruction
For DisconnectLeg the SSF−FSM shall always transit to WFI.
InitialTssf_ Active := FALSE
Tssf_Duration := Tssf_Duration4
Tssf_Duration := Tssf_Duration2
Tssf_Duration := Tssf_Duration4
SET (NOW + Tssf_ _Duration, Tssf)
SET (NOW + Tssf_Duration, Tssf)
SET (NOW + Tssf_ _Duration, Tssf)
UserInteraction_ Active := false
Monitoring


Monitoring
RequestReport_
UTSI
(invokeID,,rrutsiArg)
SendSTUI
(invokeID,,
sstuiArg)
RequestReport_
FacilityEvent
(invokeID,,
rrfArg)
SendFacility_
Information
(invokeID,,sendfArg)
Process_
RequestReport_
UTSI
(rrutsiArg)
Process_
SendSTUI
(sstuiArg)
Process_
RequestReport_
FacilityEvent
(rrfArg)
Process_
SendFacility
(sendfArg)
Monitoring

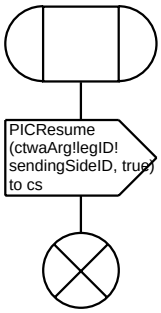
;
FPAR
IN LegId LegType,
IN/OUT ExportEventRecord EventRecordType;

DCL

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

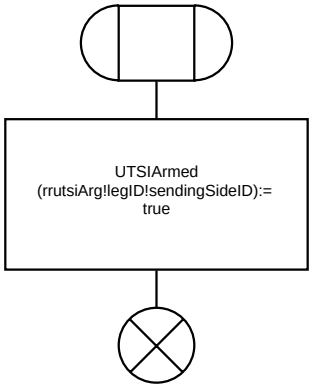


;FPAR
IN ctwaArg ContinueWithArgumentArg;



/* Note: ContinueWithArgument should cause the passive legs in the CS where ctwaArg!legID is residing to resume call processing, and not only the ctwaArg!legID. This is not clearly described in Q.1228, section 17 but is necessary for the CPH functionality. */

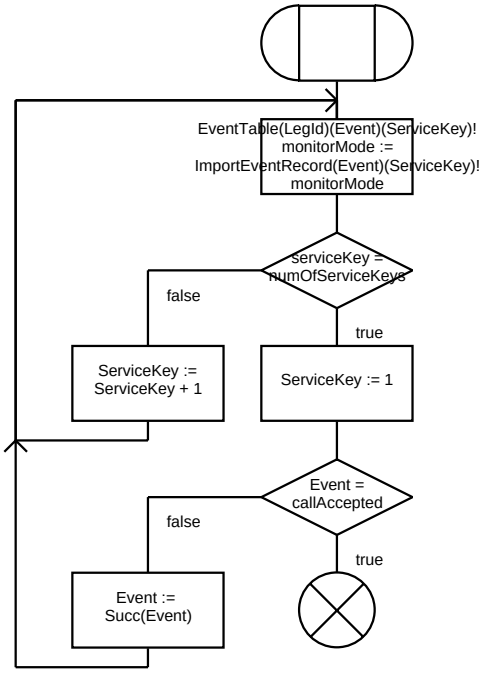
;FPAR
rrutsiArg RequestReportUTSIArg;

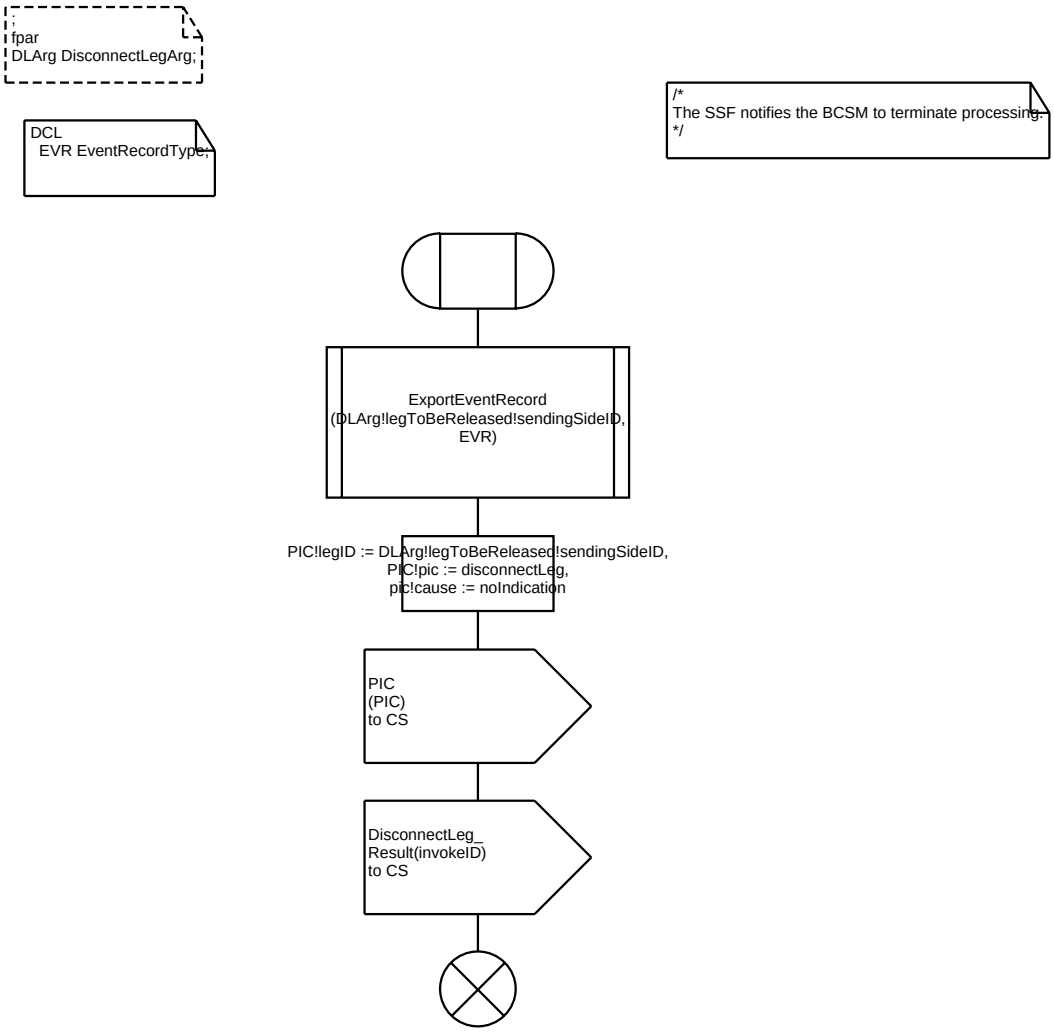


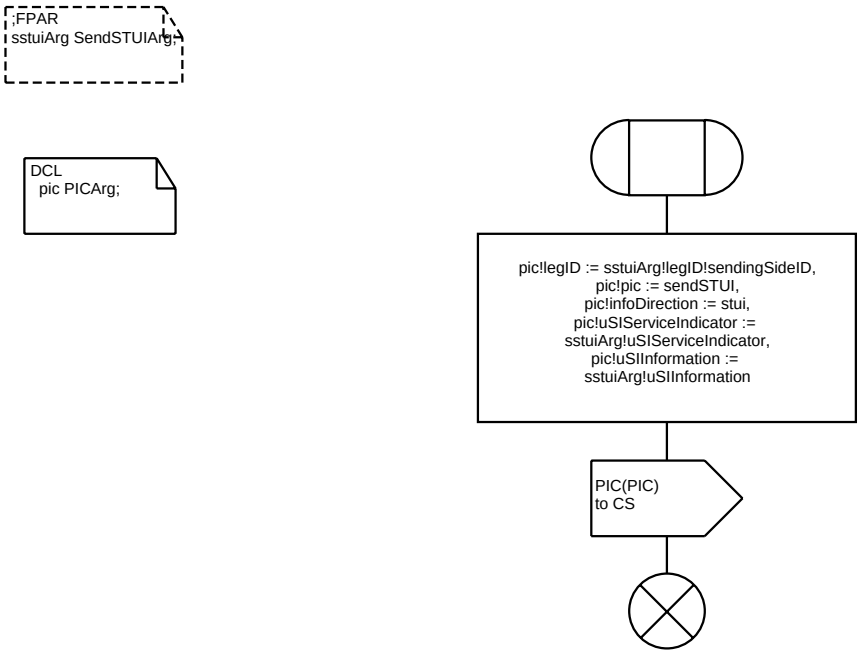
;
FPAR
IN legID LegType,
IN/OUT ImportEventRecord EventRecordType;

DCL

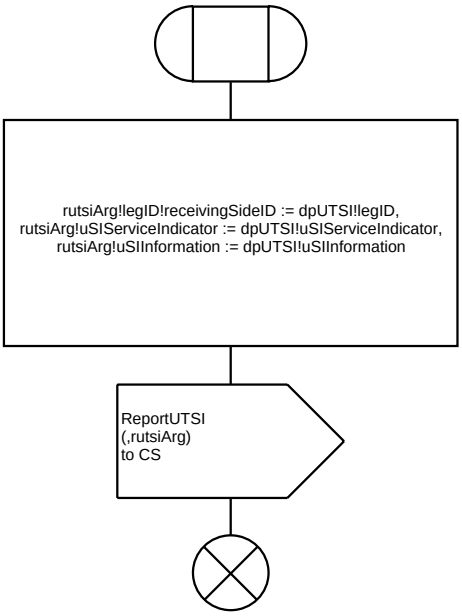
event EventTypeBCSM := origAttemptAuthorized,
serviceKey ServiceKey := 1;

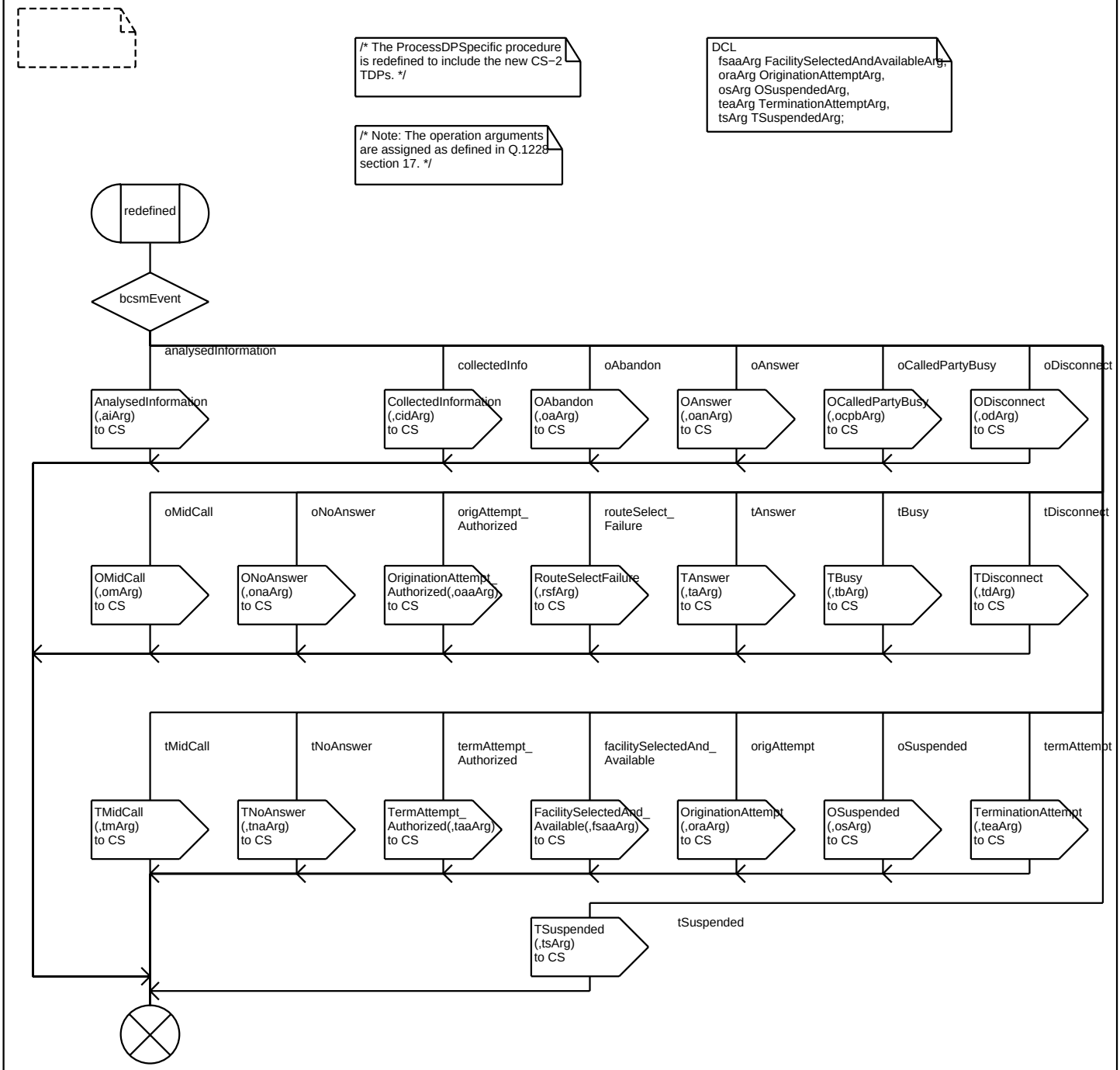



;
fpar
DLArg DisconnectLegArg;
DCL
EVR EventRecordType;
/*
The SSF notifies the BCSM to terminate processing.
*/
ExportEventRecord
(DLArg!legToBeReleased!sendingSideID,
EVR)
PIC!legID := DLArg!legToBeReleased!sendingSideID,
PIC!pic := disconnectLeg,
pic!cause := noIndication
PIC
(PIC)
to CS
DisconnectLeg_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

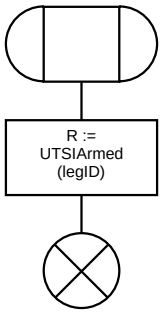

;FPAR
sstuiArg SendSTUIArg;
DCL
pic PICArg;
pic!legID := sstuiArg!legID!sendingSideID,
pic!pic := sendSTUI,
pic!infoDirection := stui,
pic!uSIServiceIndicator :=
sstuiArg!uSIServiceIndicator,
pic!uSIInformation :=
sstuiArg!uSIInformation
PIC(PIC)
to CS

;FPAR
IN dpUTSI DPUTSIArg;

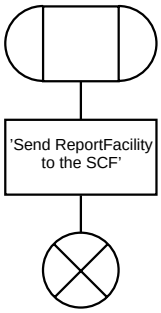



/* The ProcessDPSpecific procedure is redefined to include the new CS−2 TDPs. */
/* Note: The operation arguments are assigned as defined in Q.1228 section 17. */
DCL
fsaaArg FacilitySelectedAndAvailableArg,
oraArg OriginationAttemptArg,
osArg OSuspendedArg,
teaArg TerminationAttemptArg,
tsArg TSuspendedArg;
redefined
bcsmEvent
analysedInformation
collectedInfo
oAbandon
oAnswer
oCalledPartyBusy
oDisconnect
AnalysedInformation (,aiArg) to CS
CollectedInformation (,cidArg) to CS
OAbandon (,oaArg) to CS
OAnswer (,oanArg) to CS
OCalledPartyBusy (,ocpbArg) to CS
ODisconnect (,odArg) to CS
oMidCall
oNoAnswer
origAttempt_ Authorized
routeSelect_ Failure
tAnswer
tBusy
tDisconnect
OMidCall (,omArg) to CS
ONoAnswer (,onaArg) to CS
OriginationAttempt_ Authorized(,oaaArg) to CS
RouteSelectFailure (,rsfArg) to CS
TAnswer (,taArg) to CS
TBusy (,tbArg) to CS
TDisconnect (,tdArg) to CS
tMidCall
tNoAnswer
termAttempt_ Authorized
facilitySelectedAnd_ Available
origAttempt
oSuspended
termAttempt
TMidCall (,tmArg) to CS
TNoAnswer (,tnaArg) to CS
TermAttempt_ Authorized(,taaArg) to CS
FacilitySelectedAnd_ Available(,fsaaArg) to CS
OriginationAttempt (,oraArg) to CS
OSuspended (,osArg) to CS
TerminationAttempt (,teaArg) to CS
tSuspended
TSuspended (,tsArg) to CS

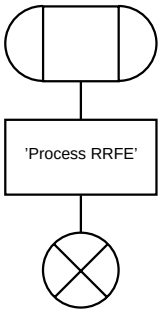
;FPAR
IN legID LegType;
RETURNS R Boolean;



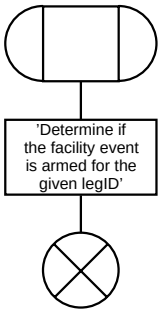
;fpar
IN dpFacility DPFacilityArg;



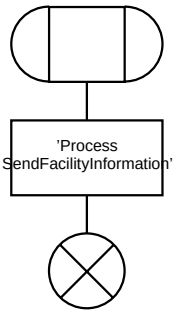
;fpar
IN rrfArg RequestReportFacilityEventArg;

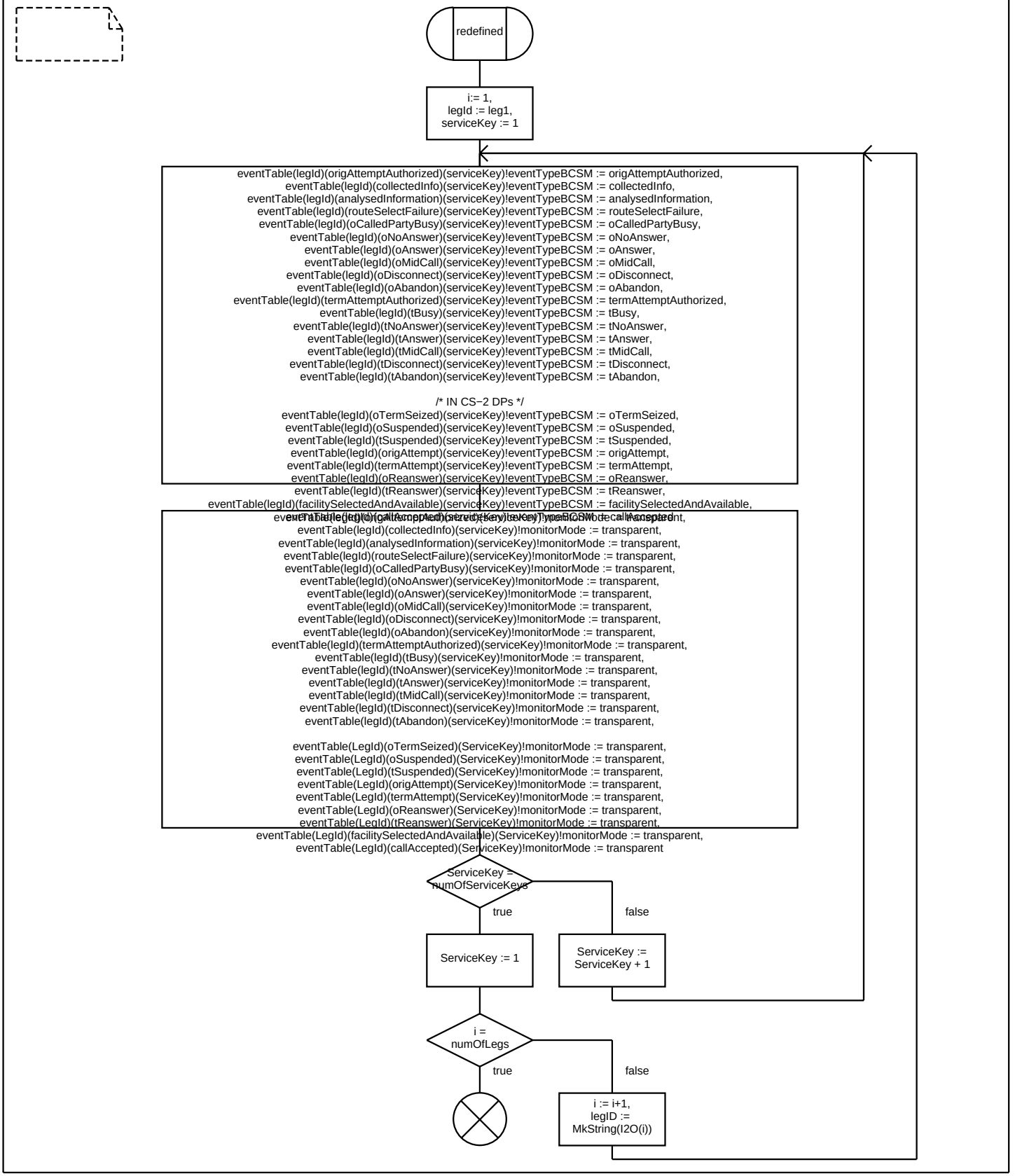


;FPAR
IN legID LegType;
RETURNS R Boolean;



;fpar
IN sefArg SendFacilityInformationArg;




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

Redefined Process Type SSME_FSM 1(1)

DCL
/* Declaration of operation arguments. */
mtdArg ManageTriggerDataArg;

Process_
ManageTriggerData

Processing of the ManageTriggerData operation.

redefined
CheckACG

For IN CS−2, the procedure for determining if call gapping should be applied is redefined.

redefined
InitialiseTDPTable

For IN CS−2, the procedure for initialising the TDP table is redefined.

Idle

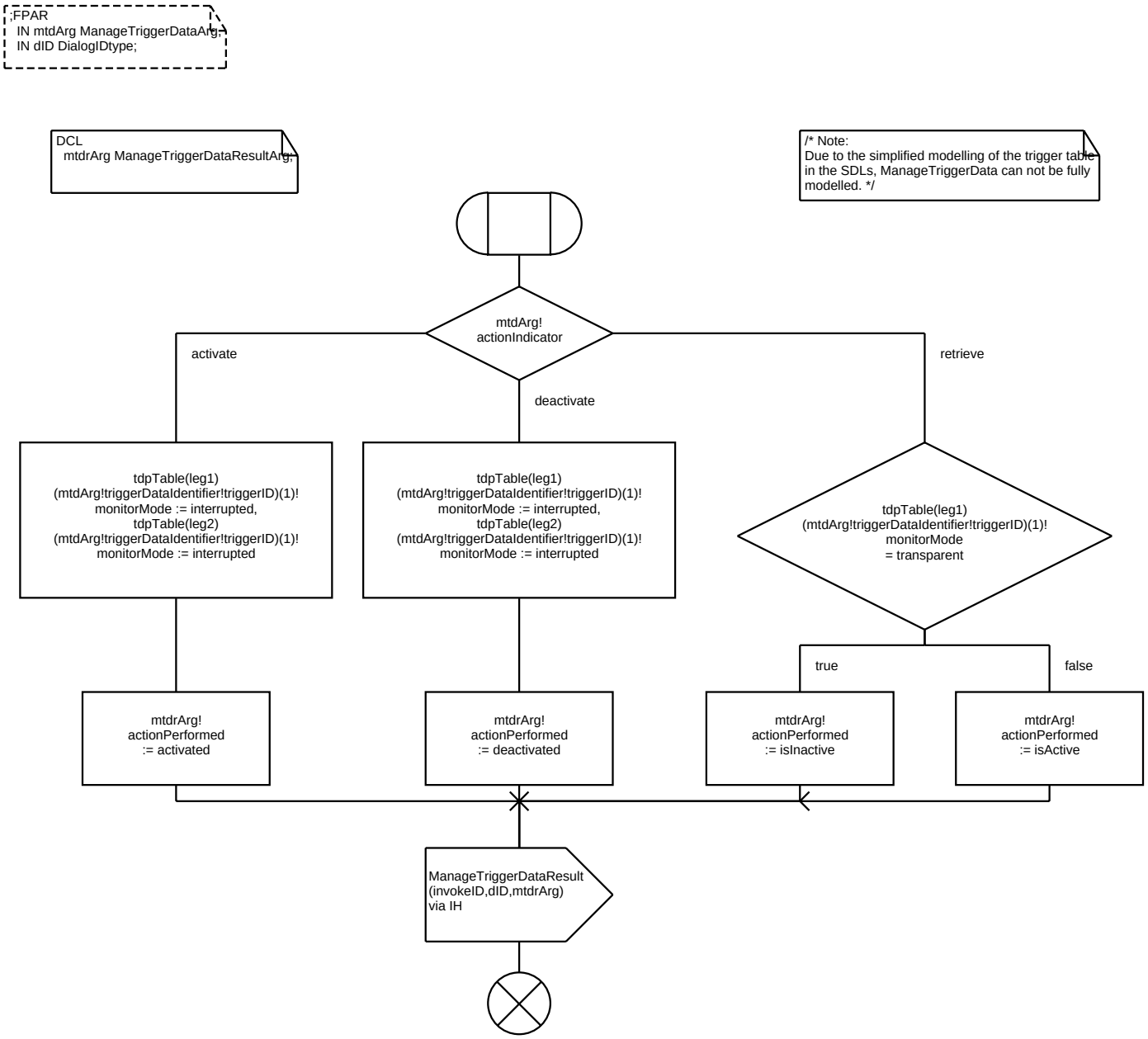
ManageTriggerData
(invokeID,dID,mtdArg)

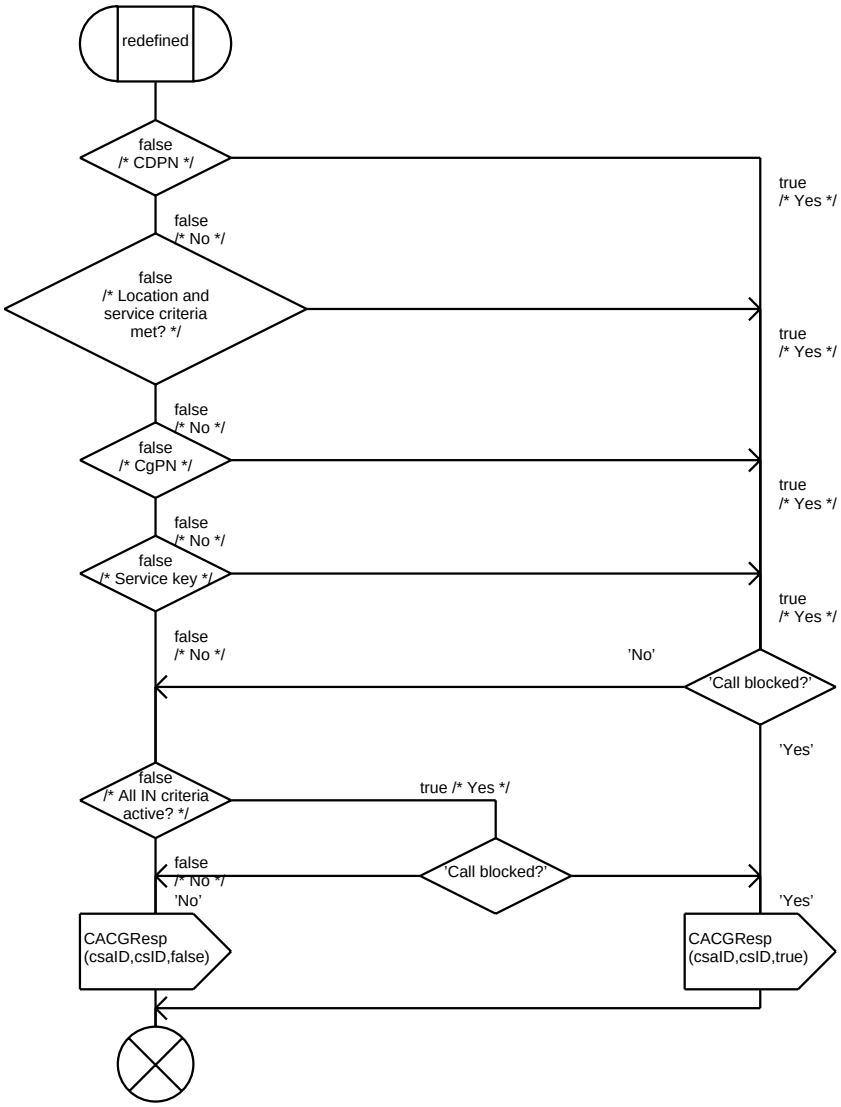
Process_
ManageTriggerData
(mtdArg,dID)

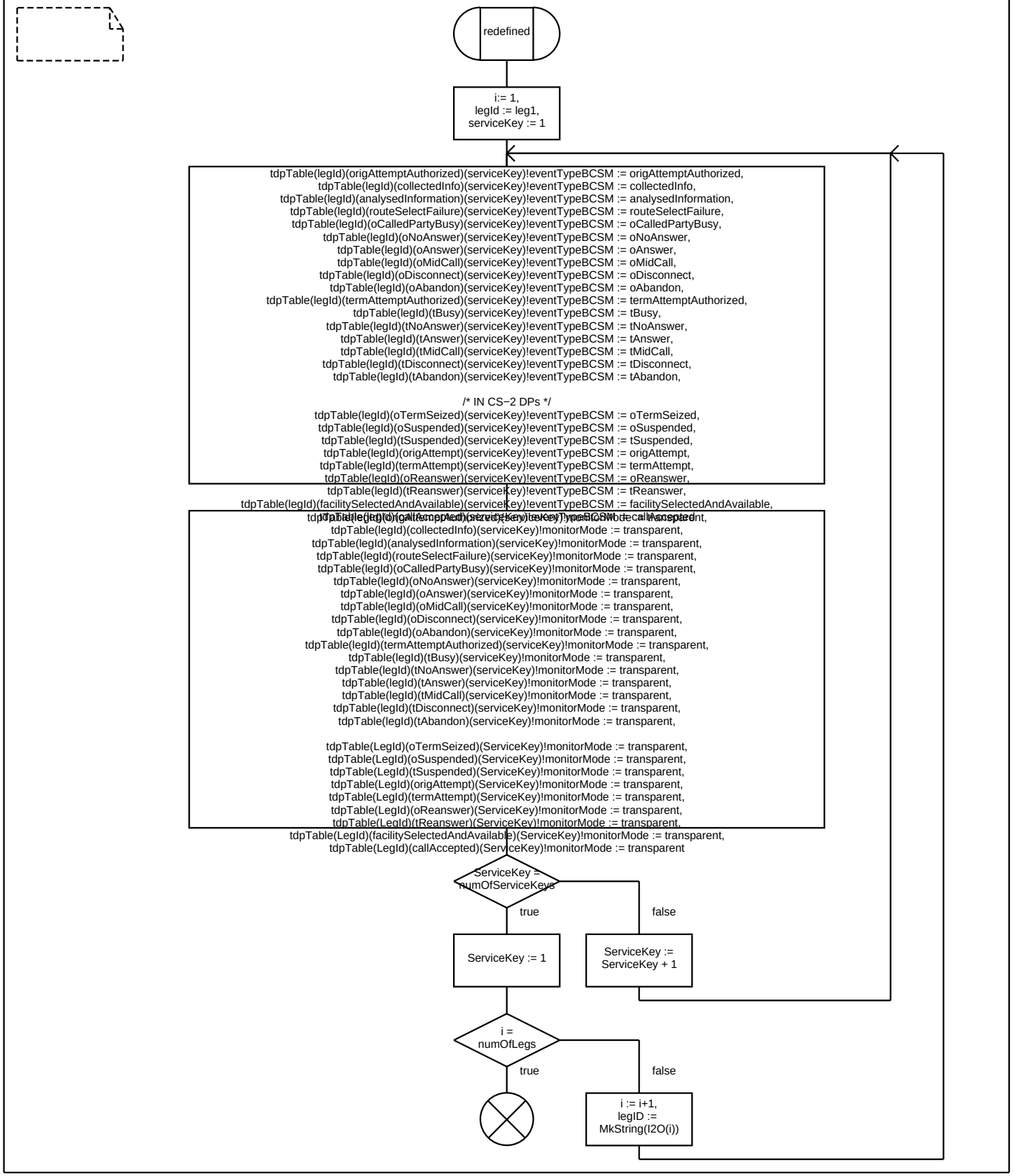
−

*

Procedure ProcessManageTriggerData 1(1)




redefined
false
/* CDPN */
true
/* Yes */
false
/* No */
false
/* Location and
service criteria
met? */
true
/* Yes */
false
/* No */
false
/* CgPN */
true
/* Yes */
false
/* No */
false
/* Service key */
true
/* Yes */
false
/* No */
'No'
'Call blocked?'
'Yes'
false
/* All IN criteria
active? */
true /* Yes */
false
/* No */
'Call blocked?'
'Yes'
'No'
CACGResp
(csaID,csID,false)
CACGResp
(csaID,csID,true)


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

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

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

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


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

/***** Procedure definitions for the new IN CS−2 PICs and the redefined IN CS−1 PICs. *****/

redefined PIC_O_ _Active

redefined PIC_Send_Call

PIC_O_ _Suspended

PIC_O_ _Retention

redefined PIC_O_ _Alerting

PIC_ _Disconnect

/***** Procedure definitions for the new IN CS−2 DPs and the redefined IN CS−1 DPs. *****/

redefined DP_origAttempt_ Authorised

redefined DP_oAnswer

redefined DP_oAbandon

DP_oSuspended

redefined DP_Collected_ _Info

redefined DP_oNoAnswer

redefined DP_ _oDisconnect

redefined DP_Analysed_ _Information

redefined DP_oCalled_ PartyBusy

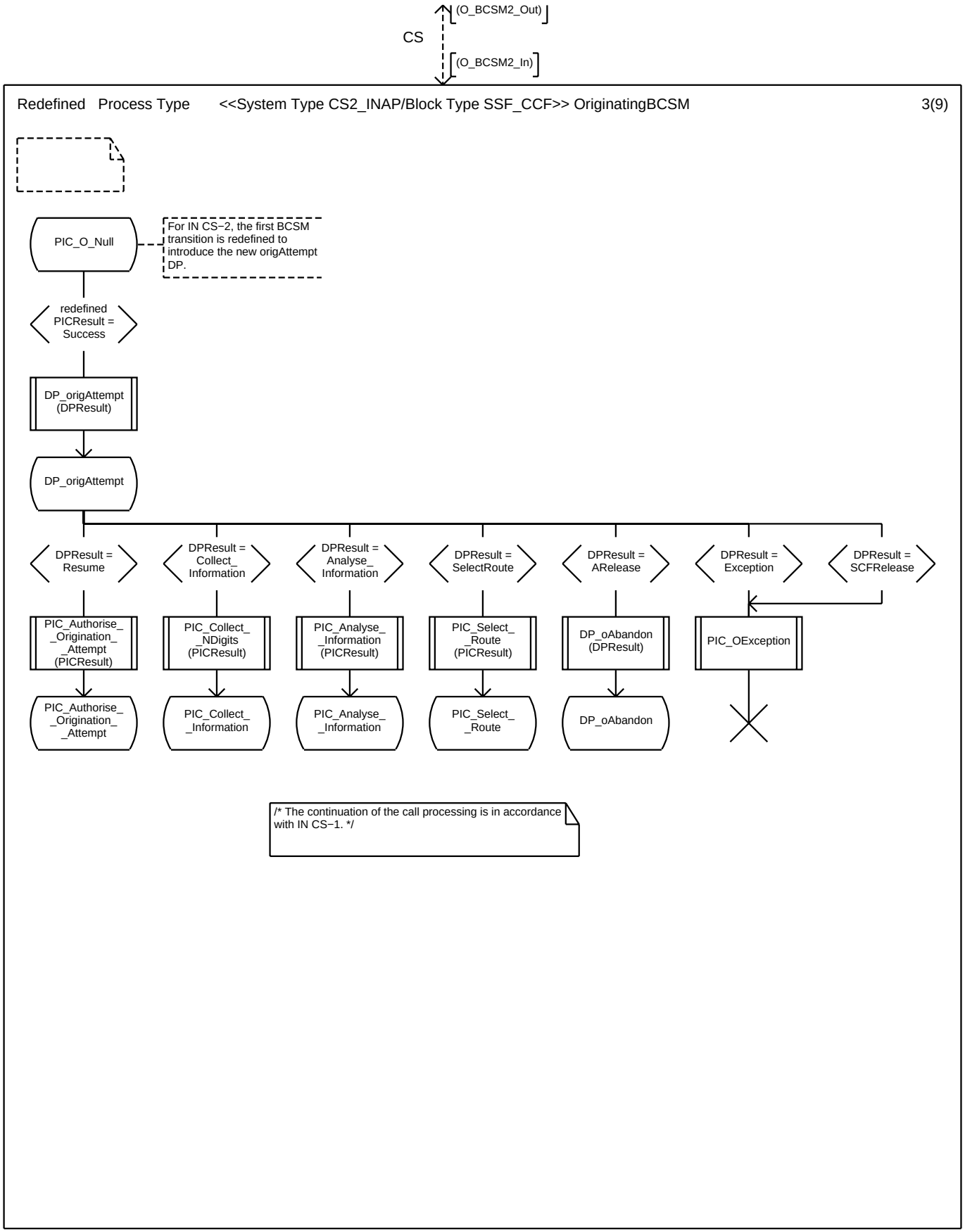
DP_oReanswer

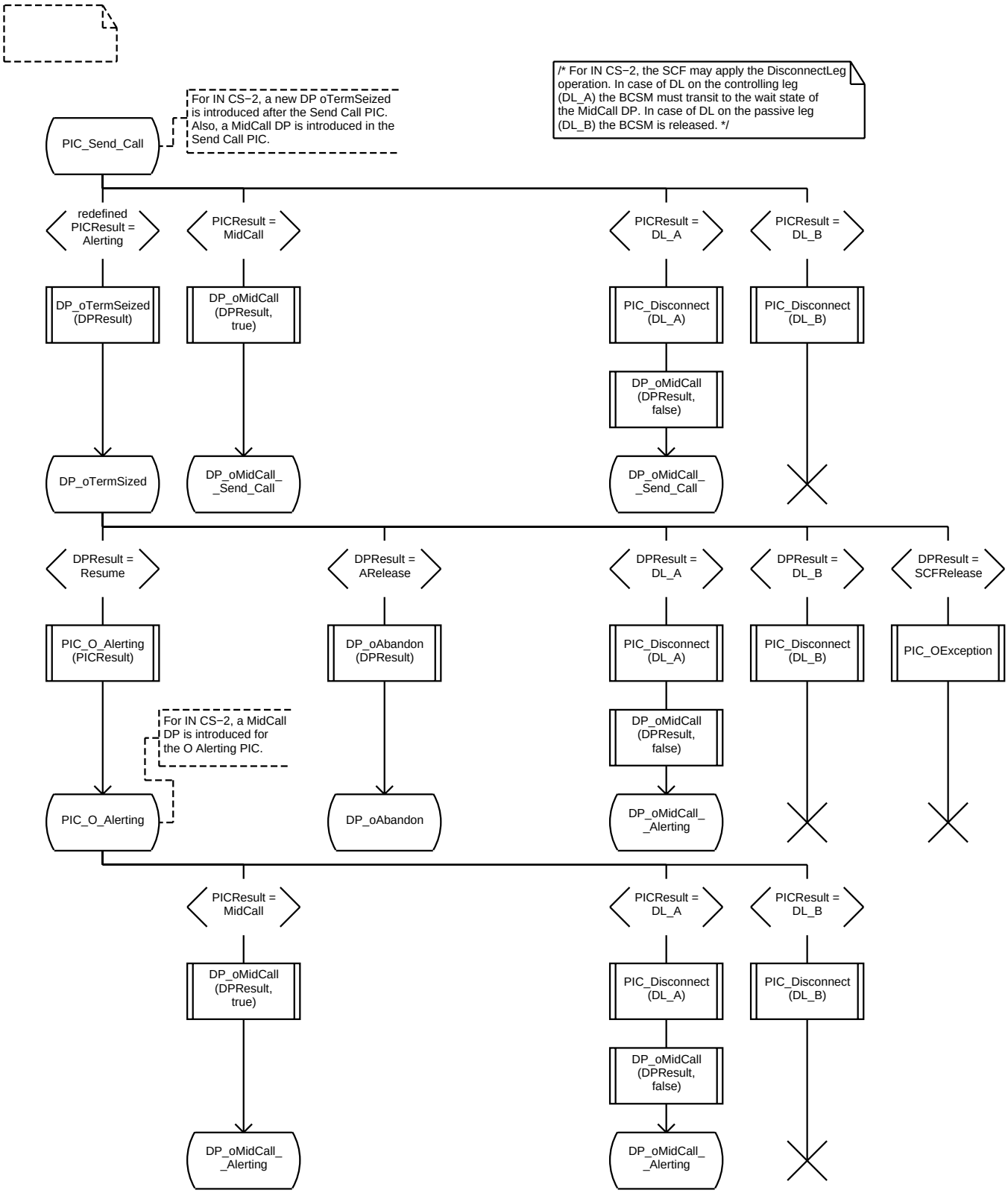
redefined DP_Route_ _Select_ _Failure

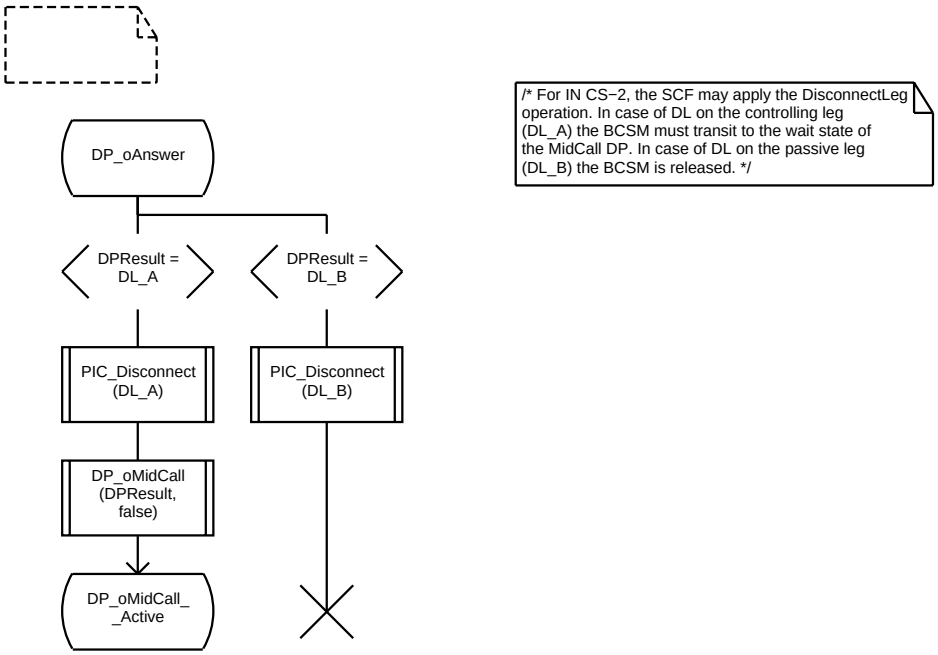
redefined DP_oMidCall

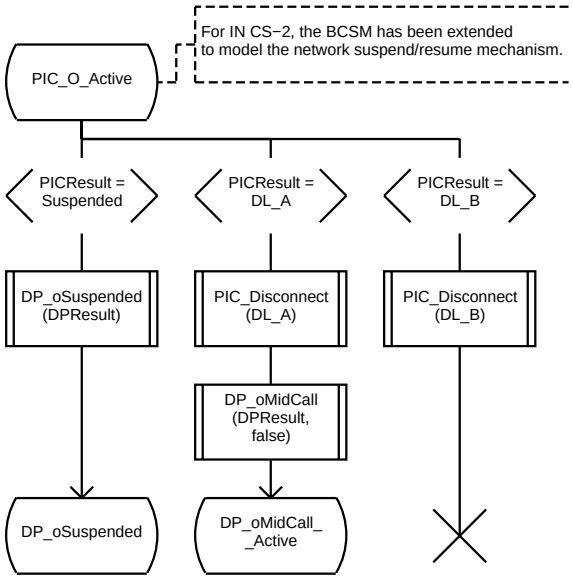
DP_orig_ Attempt

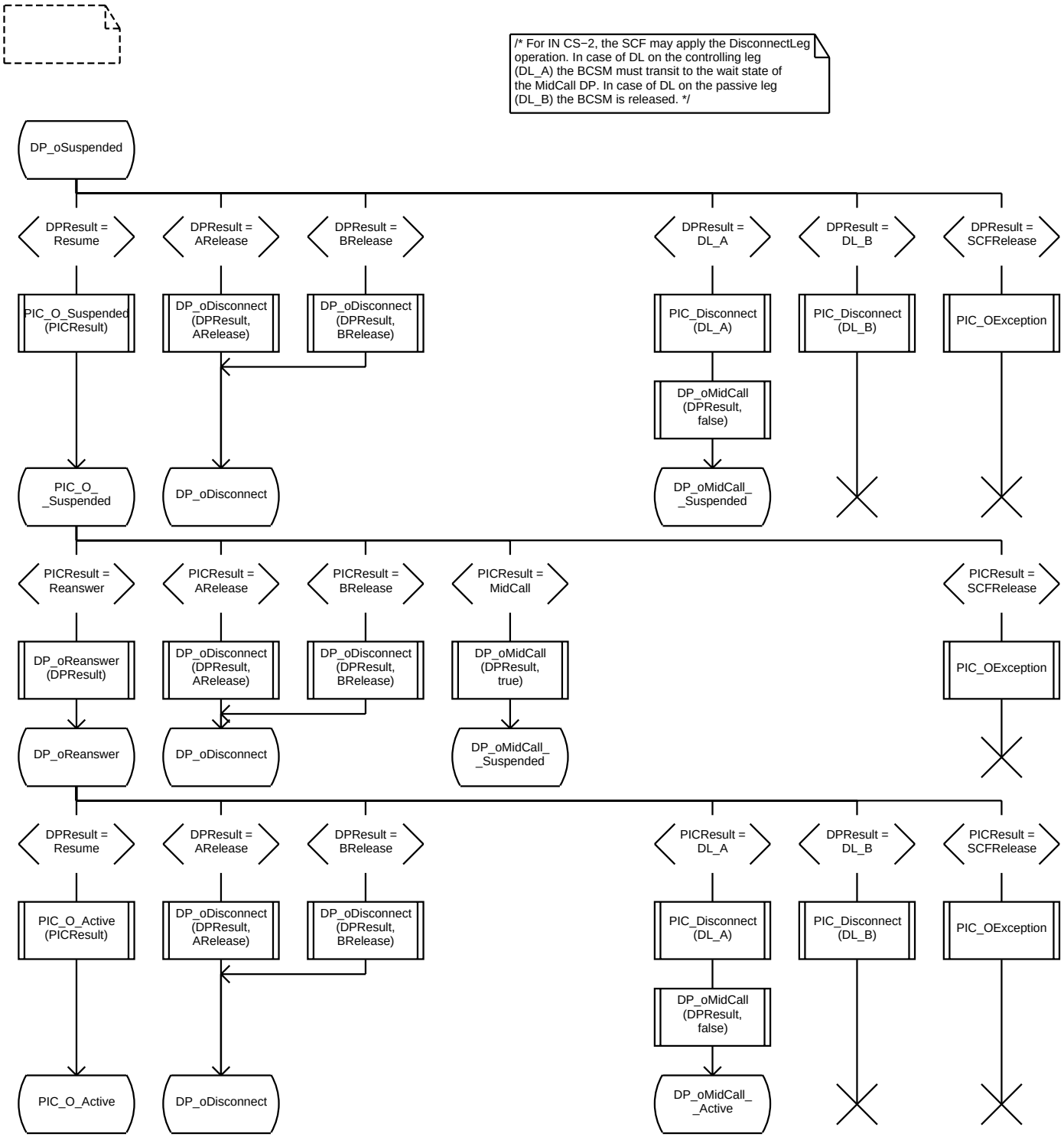
DP_ _oTermSeized

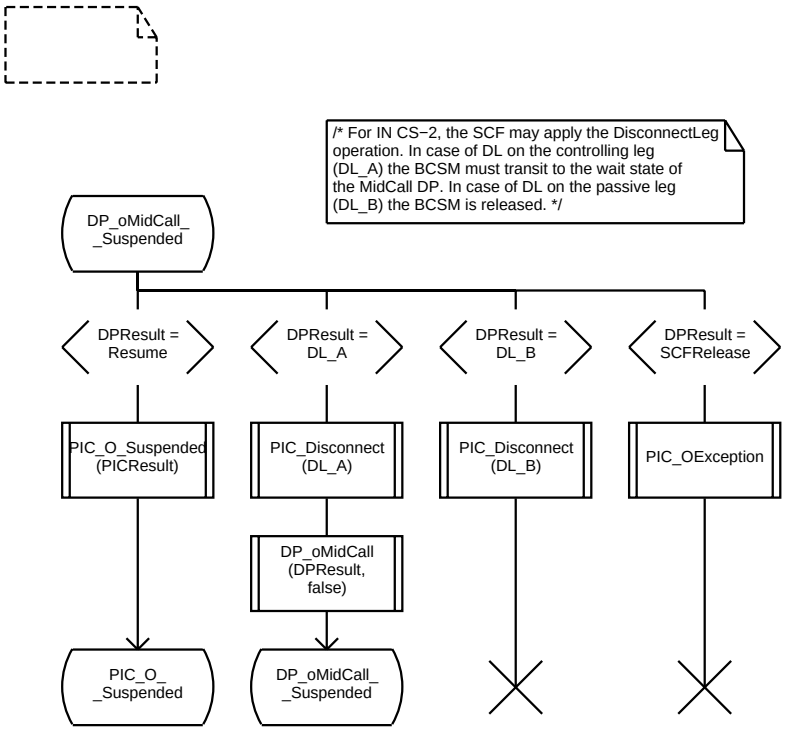

CS
(O_BCSM2_Out)
(O_BCSM2_In)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> OriginatingBCSM 3(9)
PIC_O_Null
For IN CS−2, the first BCSM transition is redefined to introduce the new origAttempt DP.
redefined PICResult = Success
DP_origAttempt (DPResult)
DP_origAttempt
DPResult = Resume
DPResult = Collect_ Information
DPResult = Analyse_ Information
DPResult = SelectRoute
DPResult = ARelease
DPResult = Exception
DPResult = SCFRelease
PIC_Authorise_ _Origination_ _Attempt (PICResult)
PIC_Collect_ _NDigits (PICResult)
PIC_Analyse_ _Information (PICResult)
PIC_Select_ _Route (PICResult)
DP_oAbandon (DPResult)
PIC_OException
PIC_Authorise_ _Origination_ _Attempt
PIC_Collect_ _Information
PIC_Analyse_ _Information
PIC_Select_ _Route
DP_oAbandon
/* The continuation of the call processing is in accordance with IN CS−1. */

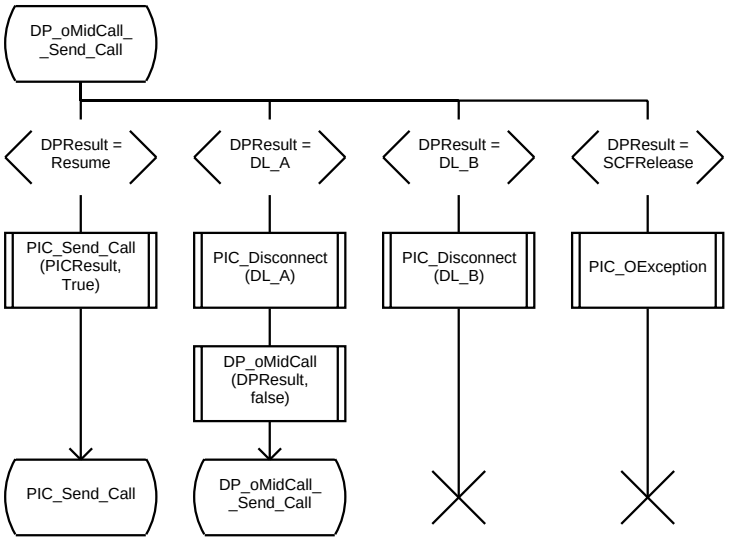

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
PIC_Send_Call
For IN CS−2, a new DP oTermSeized is introduced after the Send Call PIC. Also, a MidCall DP is introduced in the Send Call PIC.
redefined PICResult = Alerting
PICResult = MidCall
PICResult = DL_A
PICResult = DL_B
DP_oTermSeized (DPResult)
DP_oMidCall (DPResult, true)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oTermSized
DP_oMidCall_ _Send_Call
DP_oMidCall_ _Send_Call
DPResult = Resume
DPResult = ARelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_O_Alerting (PICResult)
DP_oAbandon (DPResult)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
For IN CS−2, a MidCall DP is introduced for the O Alerting PIC.
DP_oMidCall (DPResult, false)
PIC_O_Alerting
DP_oAbandon
DP_oMidCall_ _Alerting
PICResult = MidCall
PICResult = DL_A
PICResult = DL_B
DP_oMidCall (DPResult, true)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oMidCall_ _Alerting
DP_oMidCall_ _Alerting

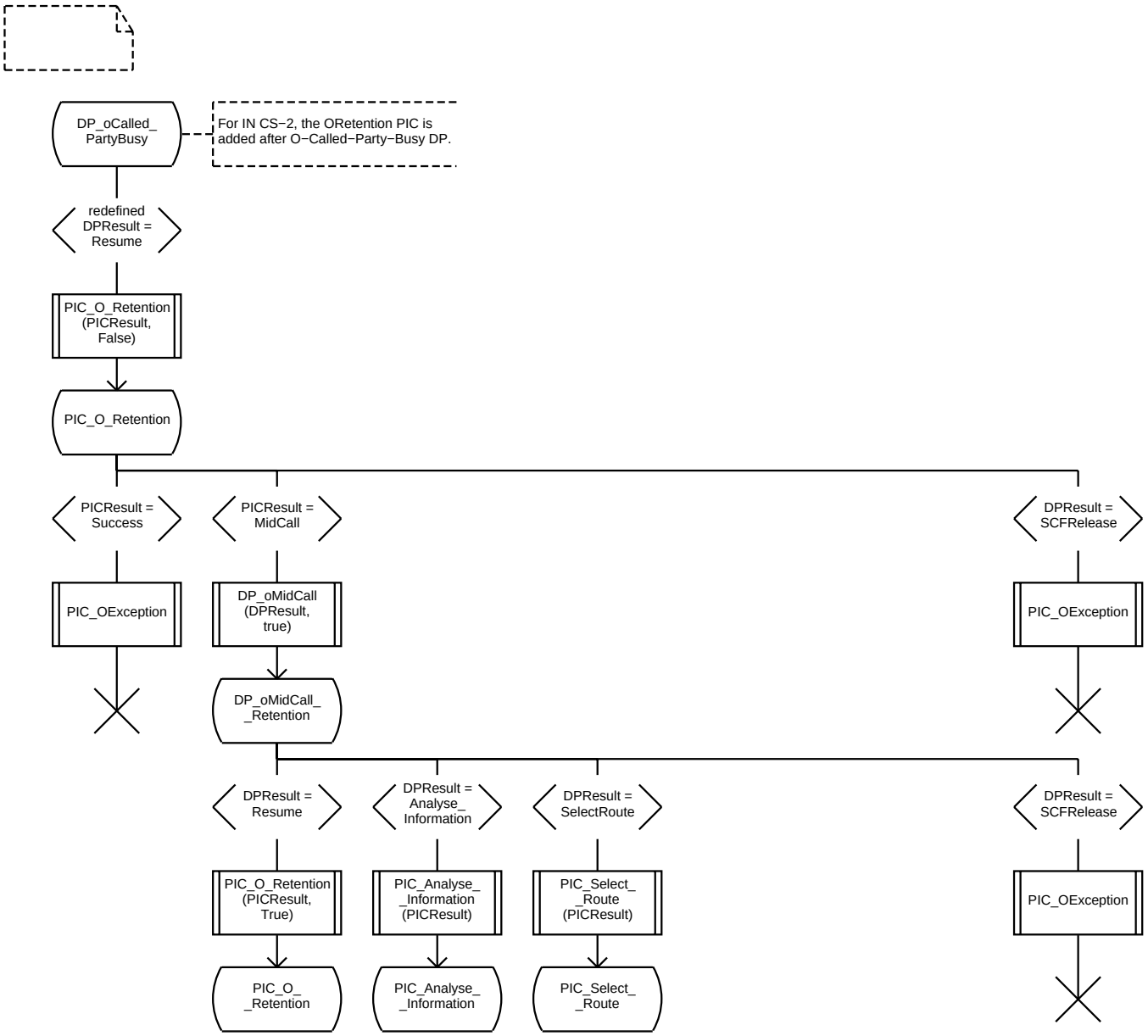

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oAnswer
DPResult = DL_A
DPResult = DL_B
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oMidCall_ _Active

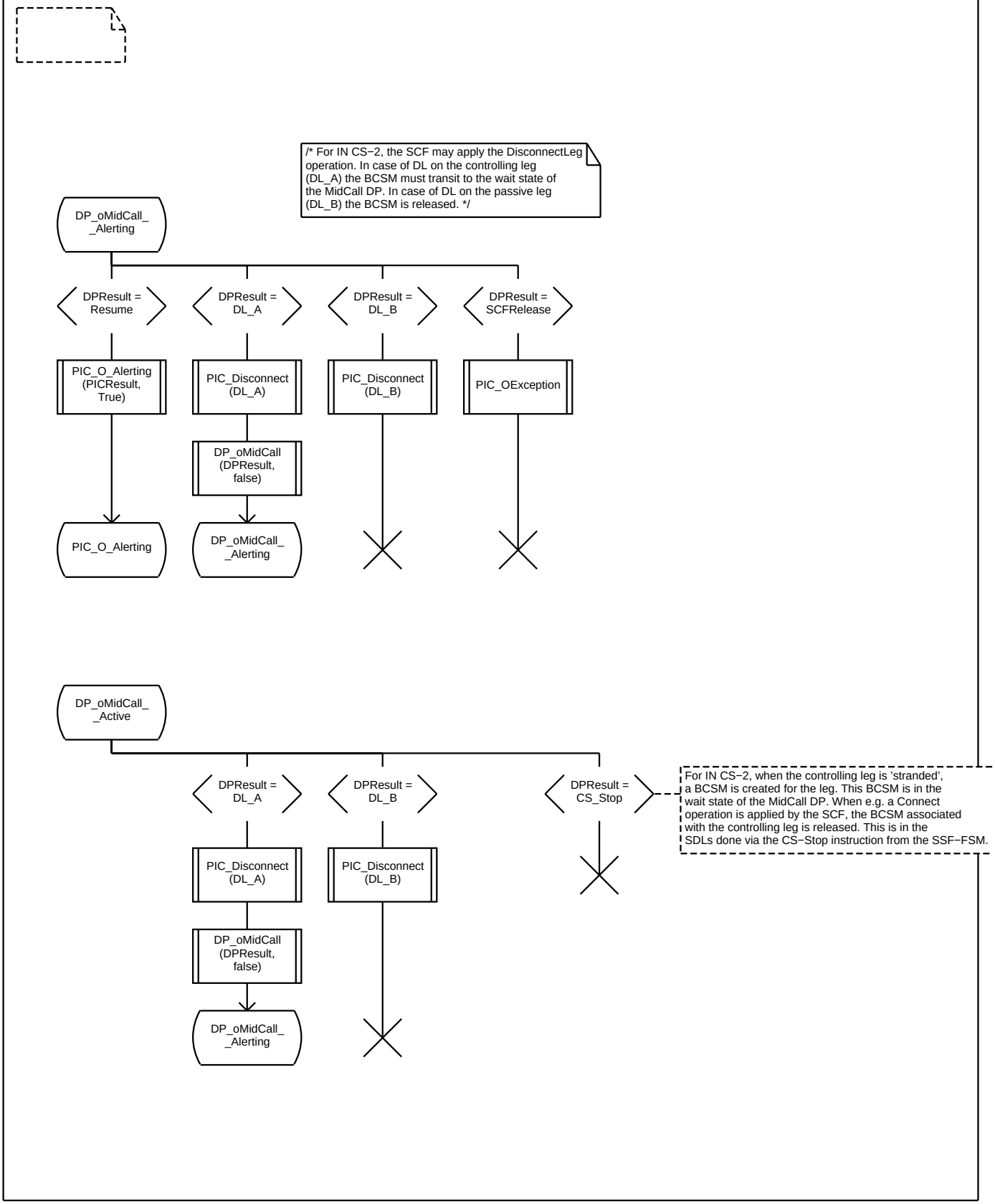

For IN CS−2, the BCSM has been extended to model the network suspend/resume mechanism.
PIC_O_Active
PICResult = Suspended
PICResult = DL_A
PICResult = DL_B
DP_oSuspended (DPResult)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oSuspended
DP_oMidCall_ _Active

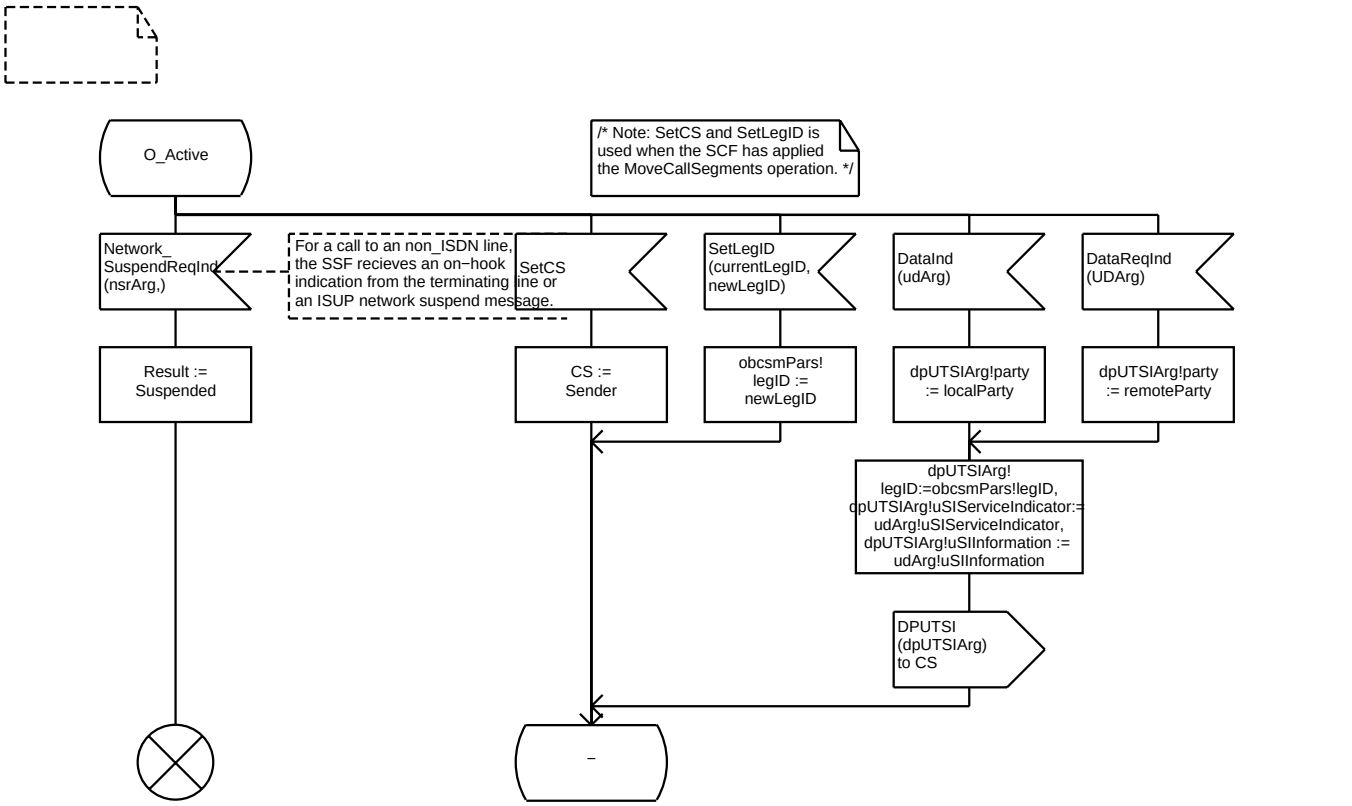

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oSuspended
DPResult = Resume
DPResult = ARelease
DPResult = BRelease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

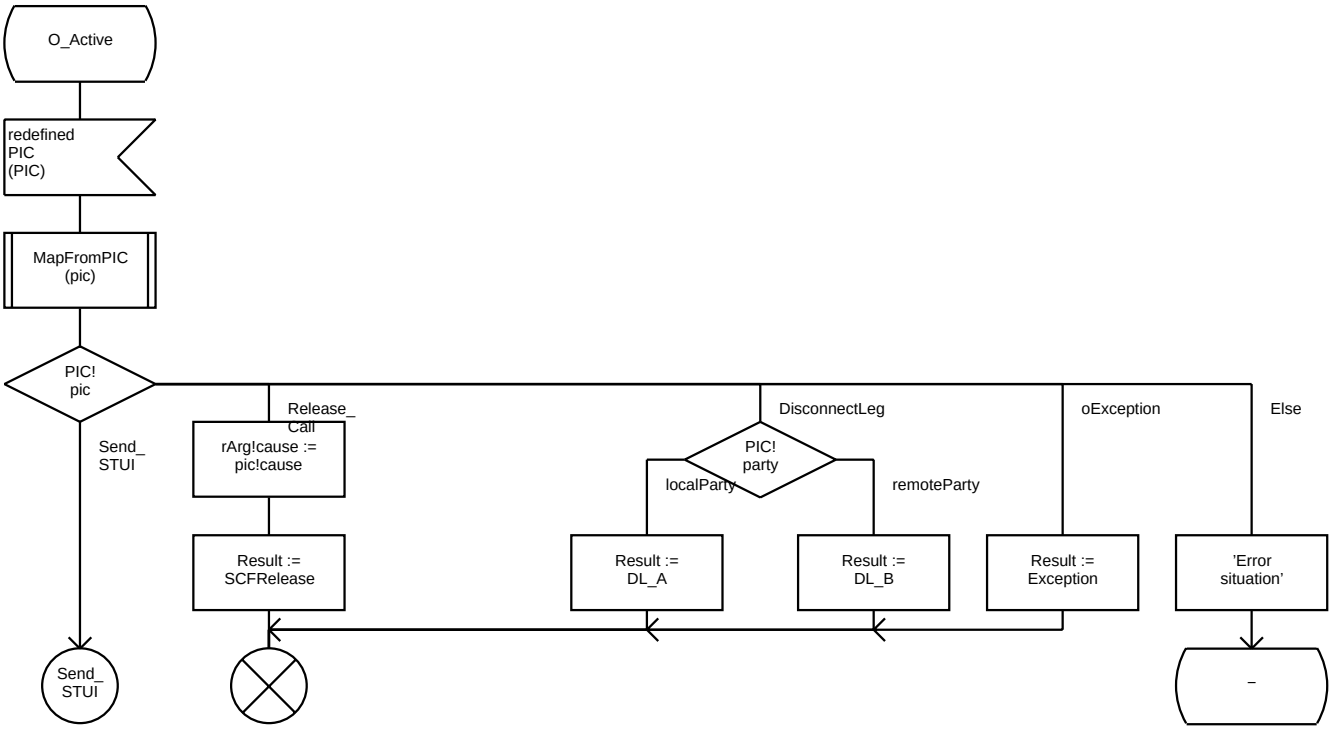

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oMidCall_ _Suspended
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_O_Suspended (PICResult)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oMidCall (DPResult, false)
PIC_O_ _Suspended
DP_oMidCall_ _Suspended

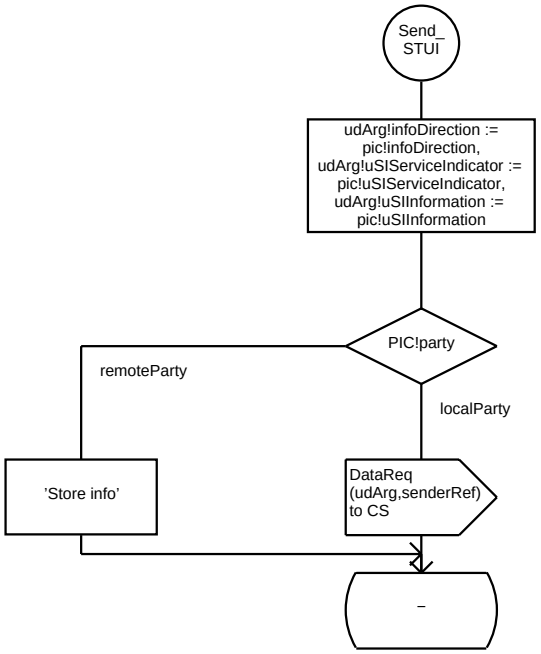

DP_oMidCall_ _Send_Call
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_Send_Call (PICResult, True)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oMidCall (DPResult, false)
PIC_Send_Call
DP_oMidCall_ _Send_Call

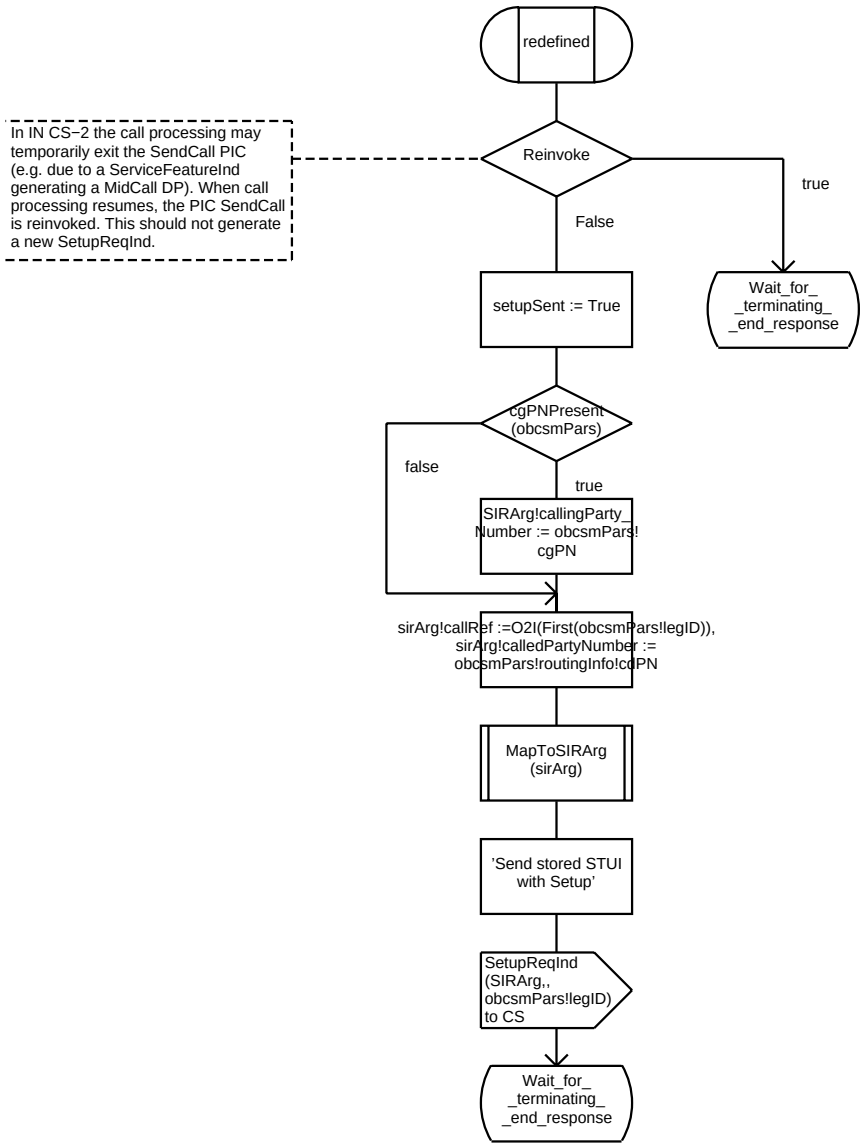

DP_oCalled_
PartyBusy
For IN CS−2, the ORetention PIC is added after O−Called−Party−Busy DP.
redefined
DPResult =
Resume
PIC_O_Retention
(PICResult,
False)
PIC_O_Retention
PICResult =
Success
PICResult =
MidCall
DPResult =
SCFRelease
PIC_OException
DP_oMidCall
(DPResult,
true)
PIC_OException
DP_oMidCall_
_Retention
DPResult =
Resume
DPResult =
Analyse_
Information
DPResult =
SelectRoute
DPResult =
SCFRelease
PIC_O_Retention
(PICResult,
True)
PIC_Analyse_
_Information
(PICResult)
PIC_Select_
_Route
(PICResult)
PIC_OException
PIC_O_
_Retention
PIC_Analyse_
_Information
PIC_Select_
_Route

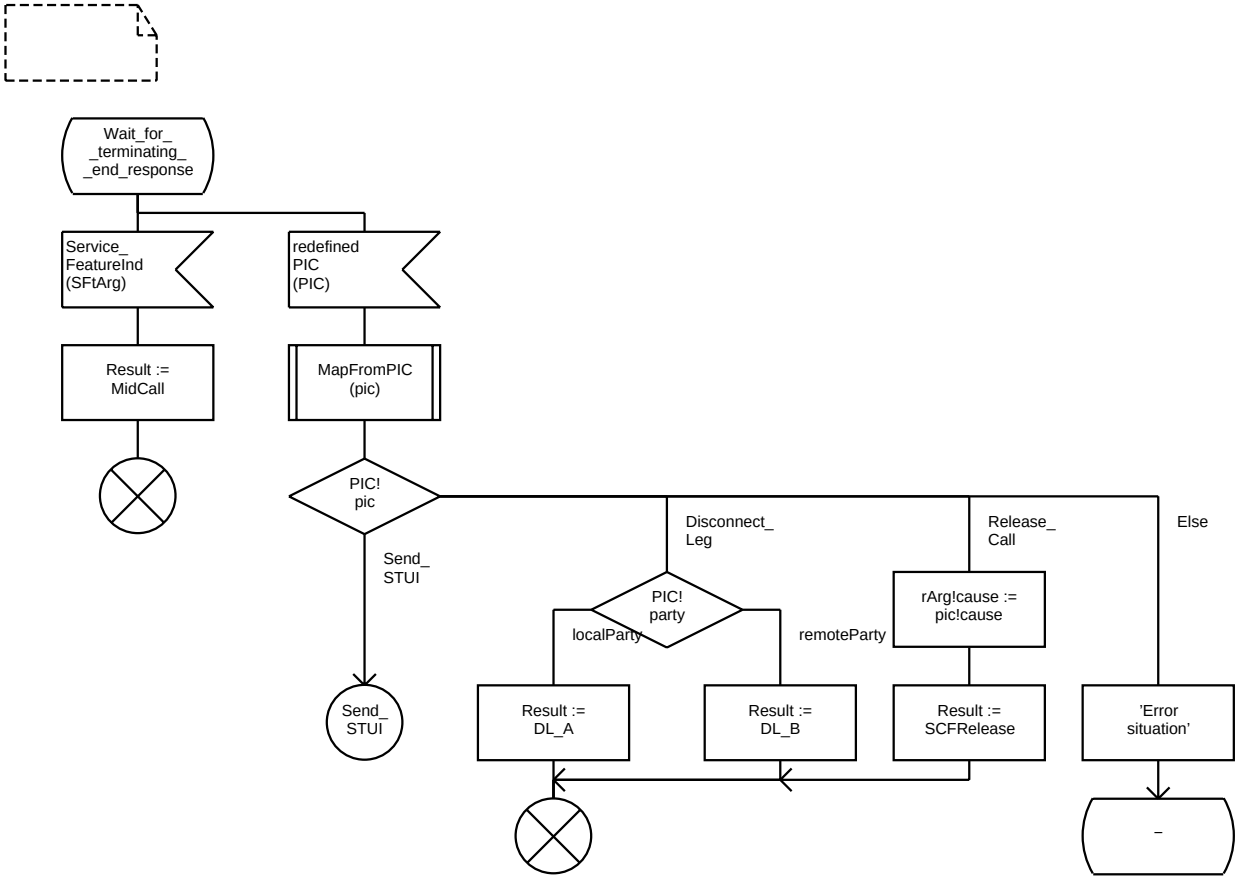

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_oMidCall_ _Alerting
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_O_Alerting (PICResult, True)
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
PIC_OException
DP_oMidCall (DPResult, false)
PIC_O_Alerting
DP_oMidCall_ _Alerting
DP_oMidCall_ _Active
DPResult = DL_A
DPResult = DL_B
DPResult = CS_Stop
For IN CS−2, when the controlling leg is 'stranded', a BCSM is created for the leg. This BCSM is in the wait state of the MidCall DP. When e.g. a Connect operation is applied by the SCF, the BCSM associated with the controlling leg is released. This is in the SDLs done via the CS−Stop instruction from the SSF−FSM.
PIC_Disconnect (DL_A)
PIC_Disconnect (DL_B)
DP_oMidCall (DPResult, false)
DP_oMidCall_ _Alerting

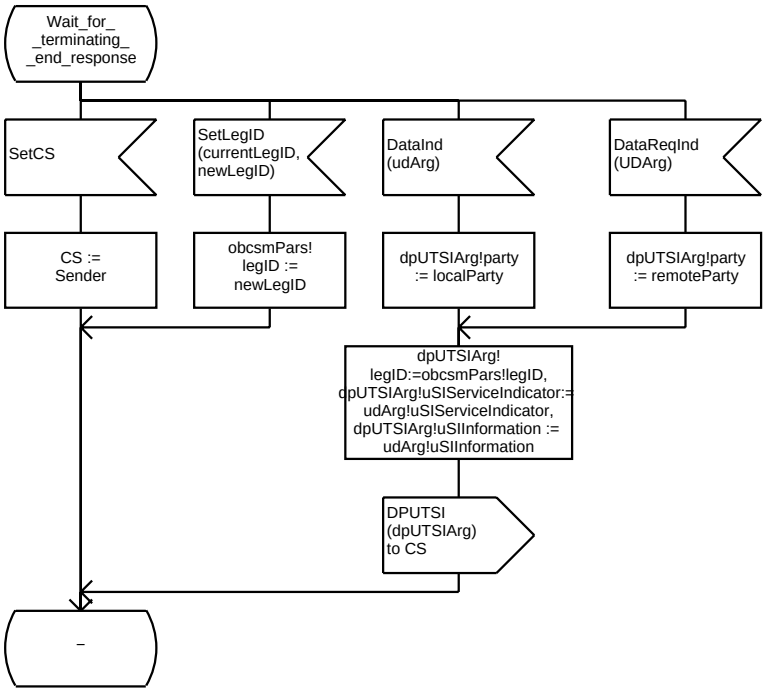

O_Active
/* Note: SetCS and SetLegID is used when the SCF has applied the MoveCallSegments operation. */
Network_ SuspendReqInd (nsrArg,)
For a call to an non_ISDN line, the SSF recieves an on−hook indication from the terminating an ISUP network suspend message.
SetCS line or
SetLegID (currentLegID, newLegID)
DataInd (udArg)
DataReqInd (UDArg)
Result := Suspended
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
dpUTSIArg!party := remoteParty
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
−

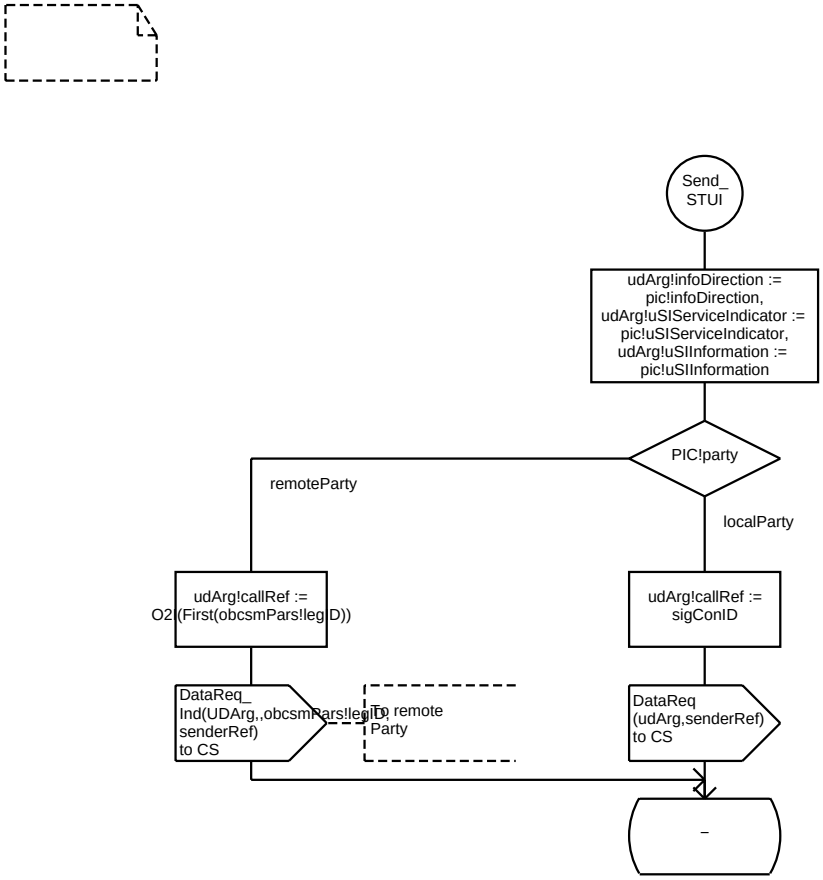

O_Active
redefined PIC (PIC)
MapFromPIC (pic)
PIC! pic
Send_ STUI
Release_ Call
rArg!cause := pic!cause
DisconnectLeg
PIC! party
localParty
remoteParty
oException
Else
Result := SCFRelease
Result := DL_A
Result := DL_B
Result := Exception
'Error situation'
Send_ STUI
−

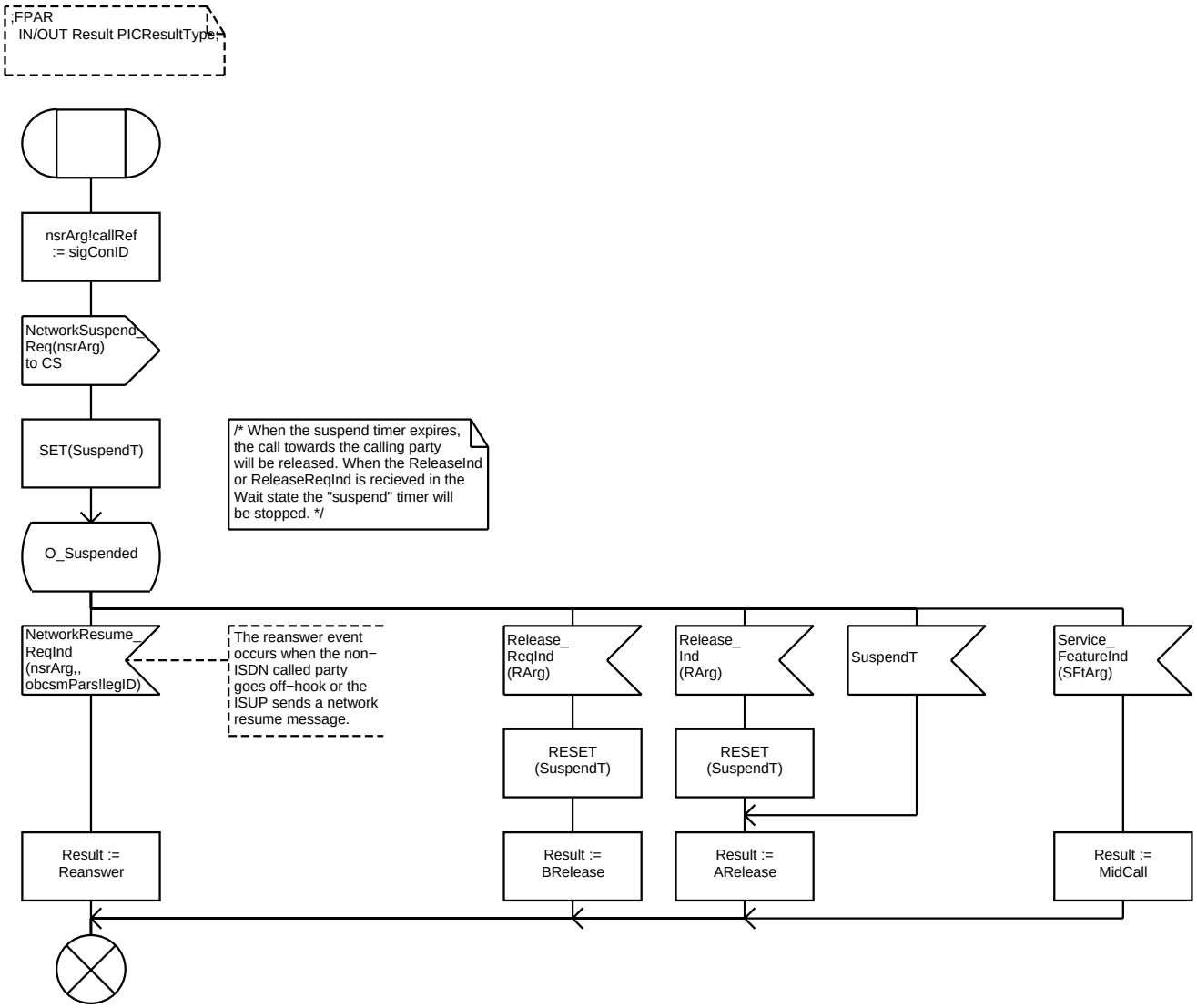

Send_
STUI
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
remoteParty
localParty
'Store info'
DataReq
(udArg,senderRef)
to CS
–

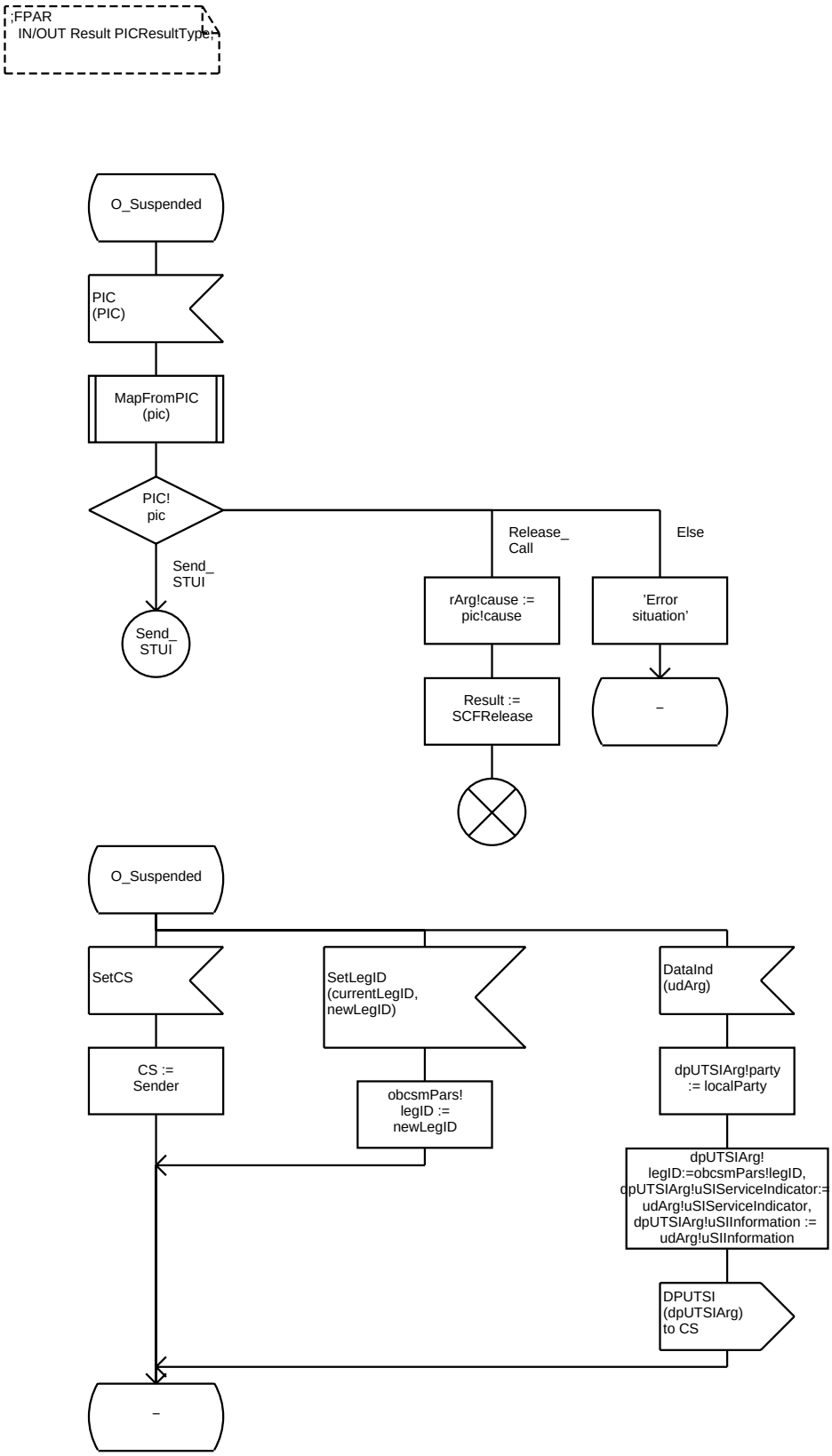

redefined
In IN CS−2 the call processing may temporarily exit the SendCall PIC (e.g. due to a ServiceFeatureInd generating a MidCall DP). When call processing resumes, the PIC SendCall is reinvoked. This should not generate a new SetupReqInd.
Reinvoke
true
False
setupSent := True
Wait_for_ _terminating_ _end_response
cgPNPresent (obcsmPars)
false
true
SIRArg!callingParty_ Number := obcsmPars! cgPN
sirArg!callRef :=O2I(First(obcsmPars!legID)), sirArg!calledPartyNumber := obcsmPars!routingInfo!cdPN
MapToSIRArg (sirArg)
'Send stored STUI with Setup'
SetupReqInd (SIRArg,, obcsmPars!legID) to CS
Wait_for_ _terminating_ _end_response


Wait_for_
terminating
_end_response
Service_
FeatureInd
(SFtArg)
redefined
PIC
(PIC)
Result :=
MidCall
MapFromPIC
(pic)
PIC!
pic
Send_
STUI
Disconnect_
Leg
Release_
Call
Else
PIC!
party
localParty
remoteParty
rArg!cause :=
pic!cause
Send_
STUI
Result :=
DL_A
Result :=
DL_B
Result :=
SCFRelease
'Error
situation'
–

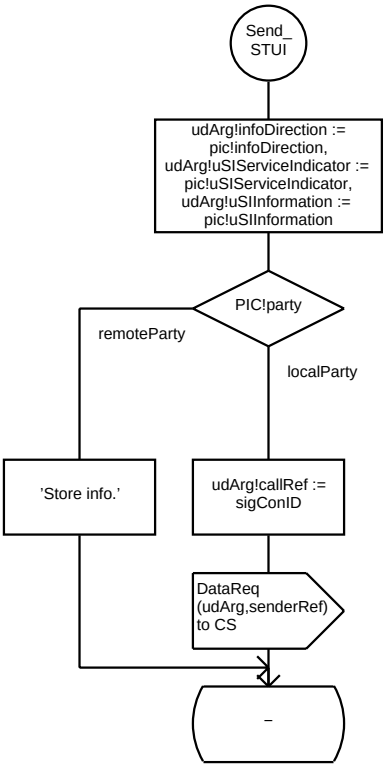

Wait_for_
terminating
_end_response
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
DataReqInd
(UDArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–


Send_
STUI
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
remoteParty
localParty
udArg!callRef :=
O2.(First(obcsmPars!legID))
udArg!callRef :=
sigConID
DataReq_
Ind(UDArg,,obcsmPars!legID,
senderRef)
to CS
To remote
Party
DataReq
(udArg,senderRef)
to CS
–

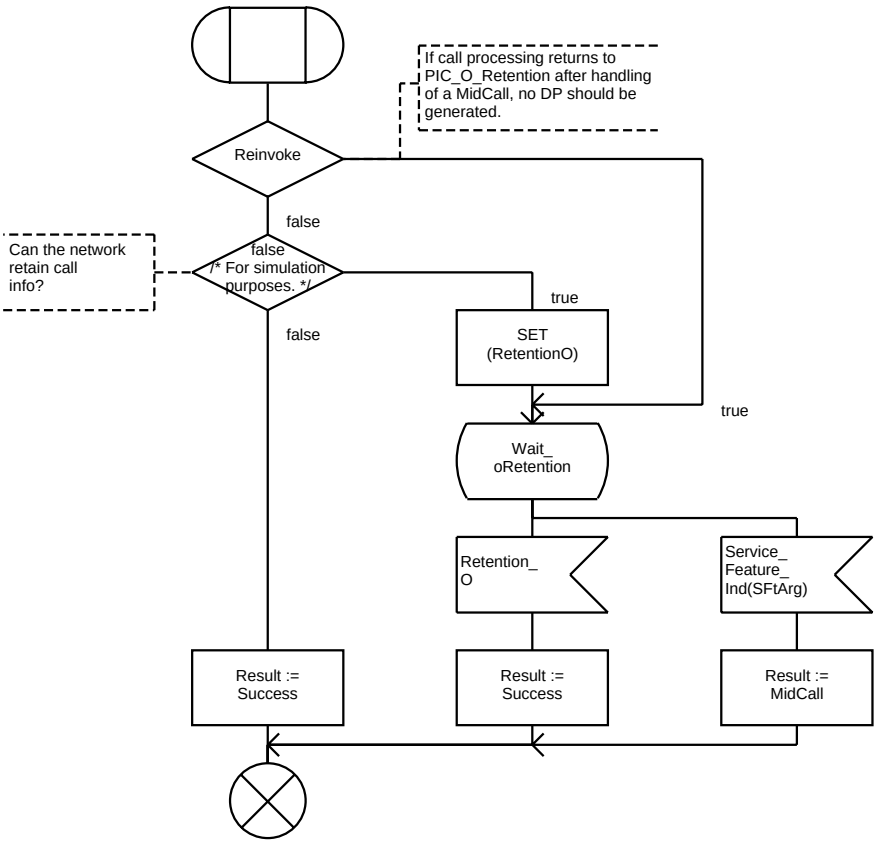

;FPAR
IN/OUT Result PICResultType;
nsrArg!callRef
:= sigConID
NetworkSuspend_
Req(nsrArg)
to CS
SET(SuspendT)
/* When the suspend timer expires, the call towards the calling party will be released. When the ReleaseInd or ReleaseReqInd is recieved in the Wait state the "suspend" timer will be stopped.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

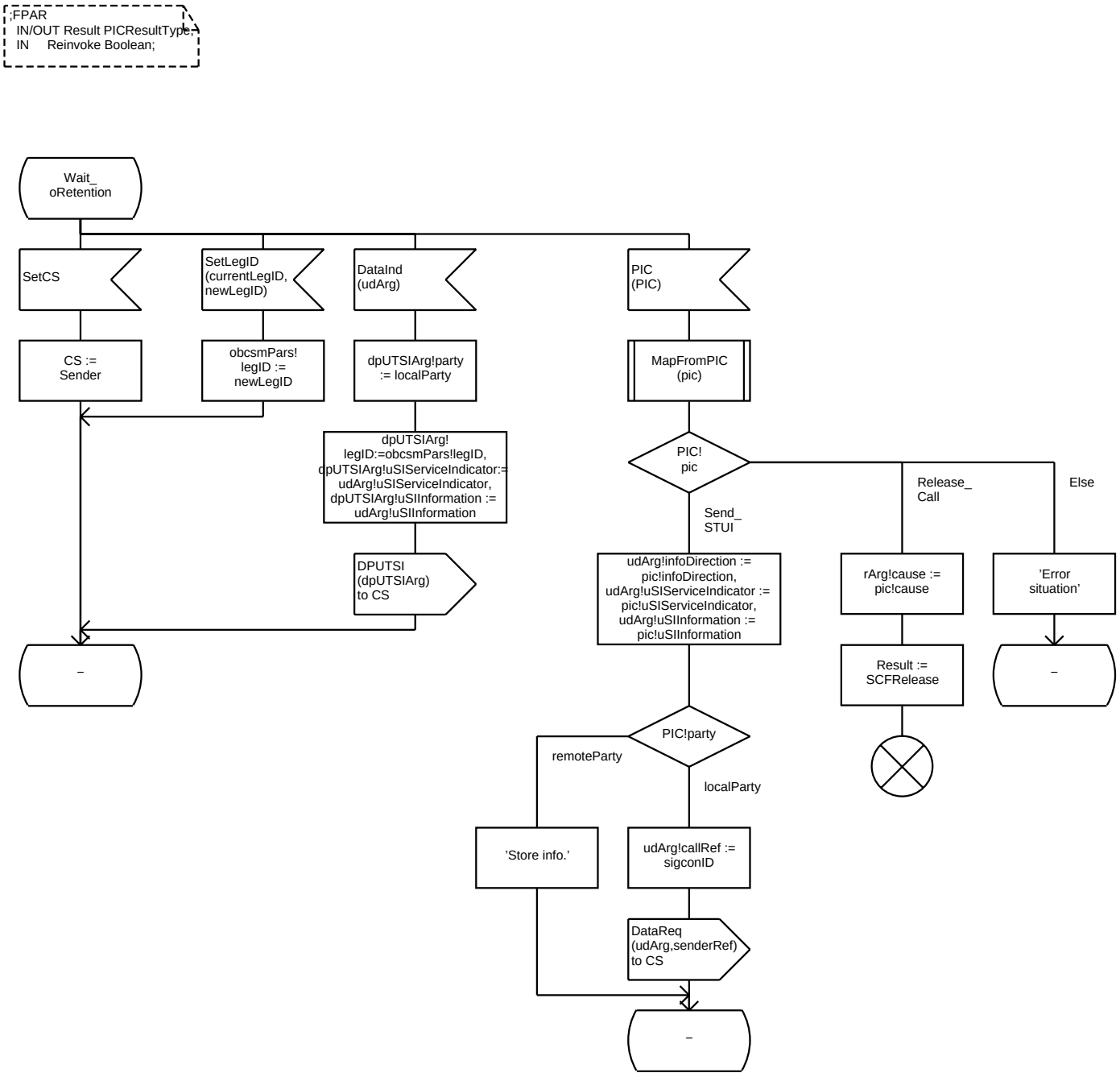

;FPAR
IN/OUT Result PICResultType;
O_Suspended
PIC
(PIC)
MapFromPIC
(pic)
PIC!
pic
Send_
STUI
Release_
Call
Else
Send_
STUI
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
–
O_Suspended
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party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

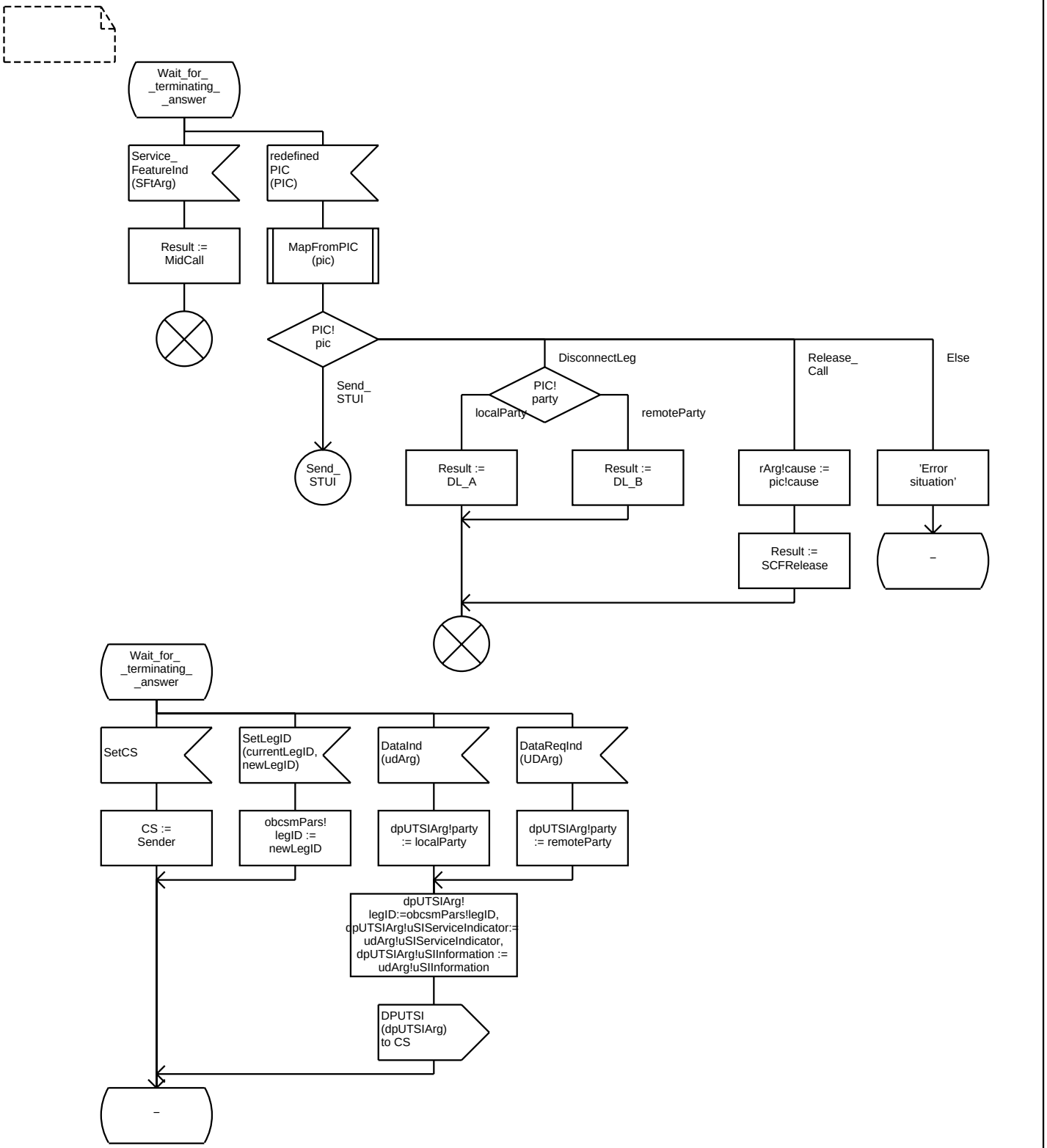
;FPAR
IN/OUT Result PICResultType;

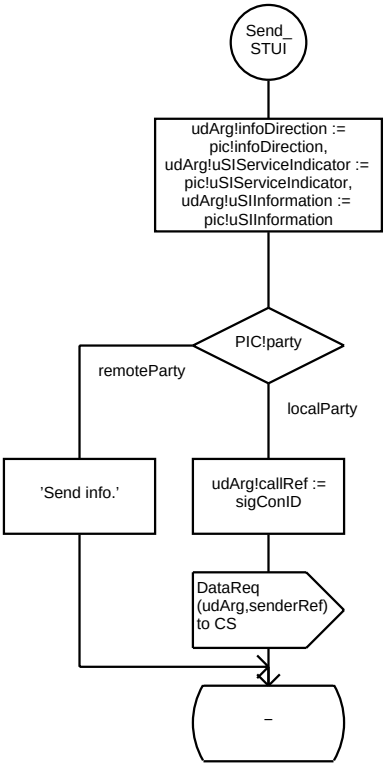


```
;FPAR
  IN/OUT Result PICResultType;
  IN     Reinvoke Boolean;
```

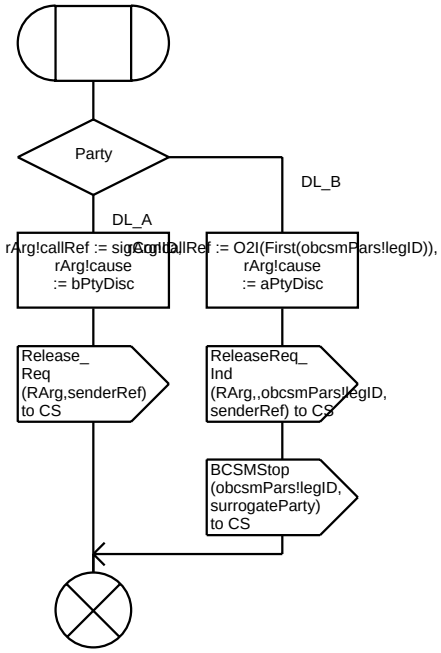


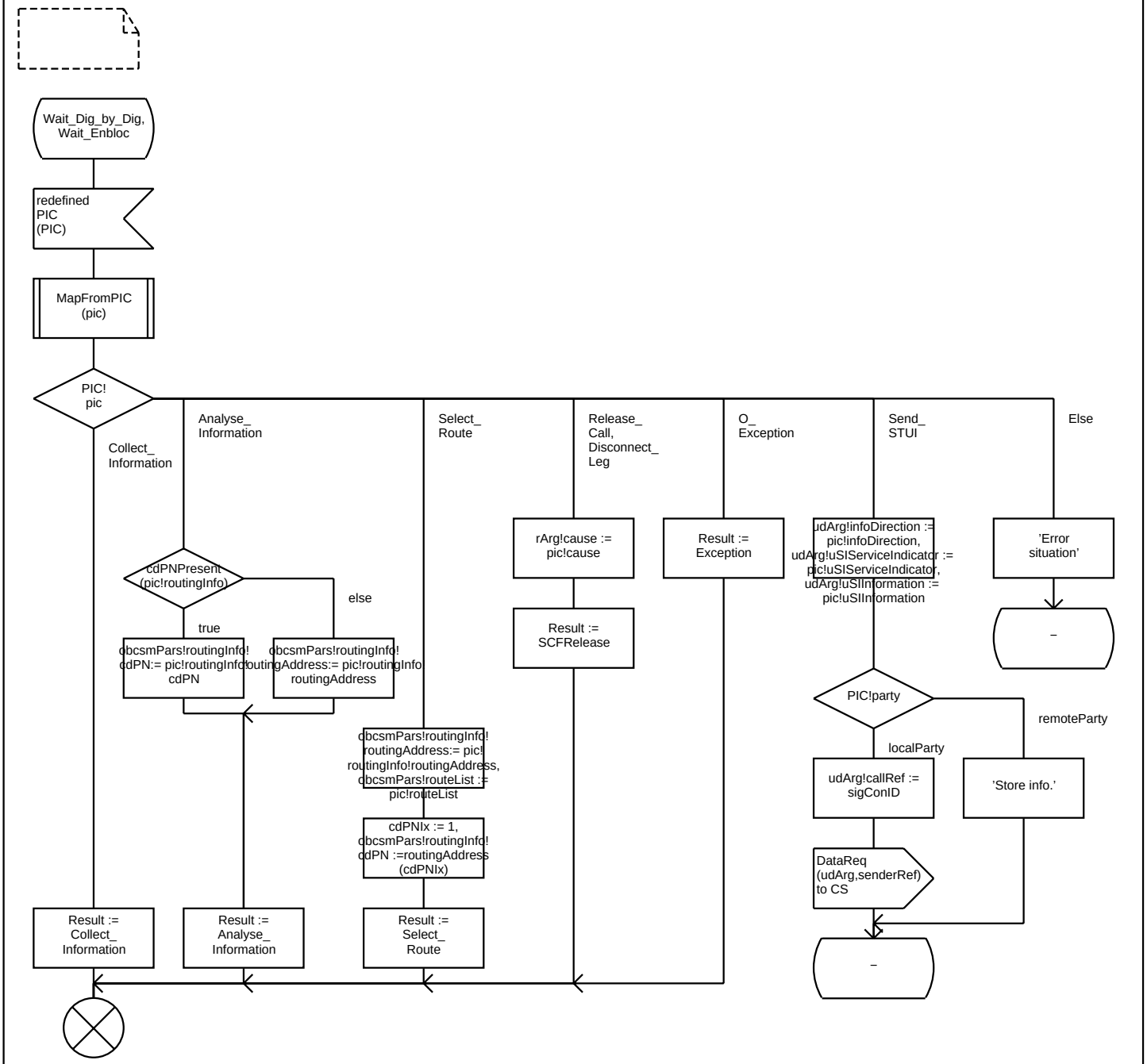

;FPAR
IN/OUT Result PICResultType;
IN Reinvoke Boolean;
Wait_ oRetention
SetCS
SetLegID (currentLegID, newLegID)
DataInd (udArg)
PIC (PIC)
CS := Sender
obcsmPars! legID := newLegID
dpUTSIArg!party := localParty
MapFromPIC (pic)
dpUTSIArg! legID:=obcsmPars!legID, dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator,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

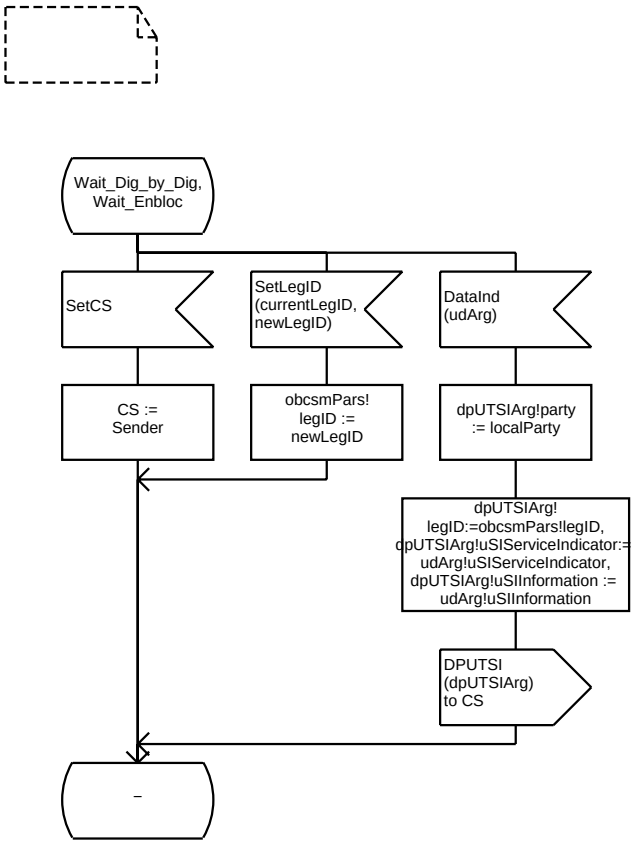

Wait_for_
terminating
_answer
Service_
FeatureInd
(SFtArg)
redefined
PIC
(PIC)
Result :=
MidCall
MapFromPIC
(pic)
PIC!
pic
Send_
STUI
DisconnectLeg
Release_
Call
Else
PIC!
party
localParty
remoteParty
Send_
STUI
Result :=
DL_A
Result :=
DL_B
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
–
Wait_for_
terminating
_answer
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
DataReqInd
(UDArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

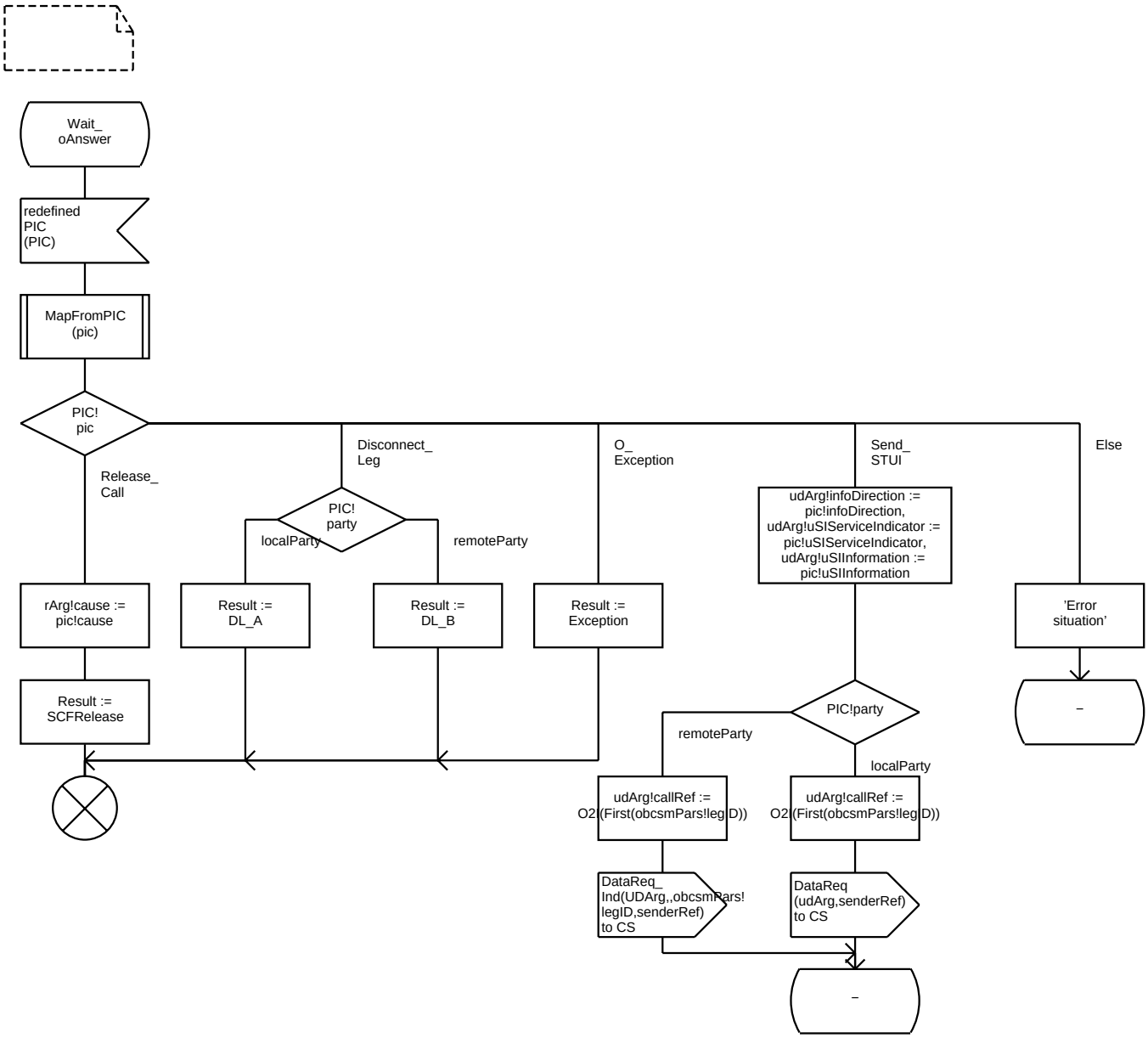

Send_
STUI
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
remoteParty
localParty
'Send info.'
udArg!callRef :=
sigConID
DataReq
(udArg,senderRef)
to CS
–

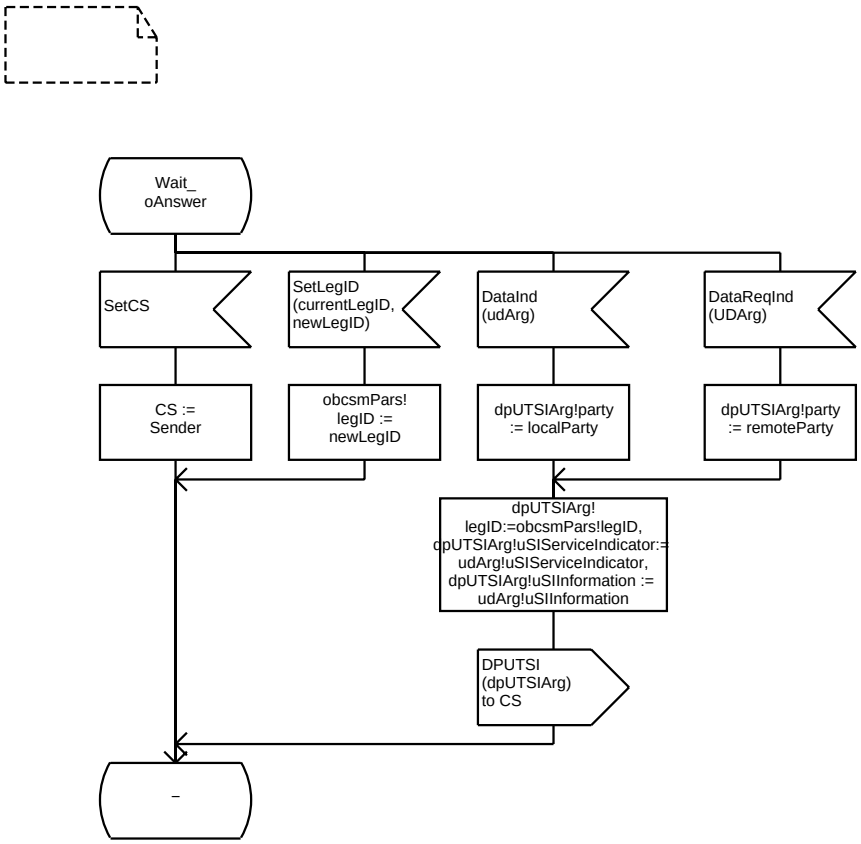
;FPAR
IN Party DPResultType;

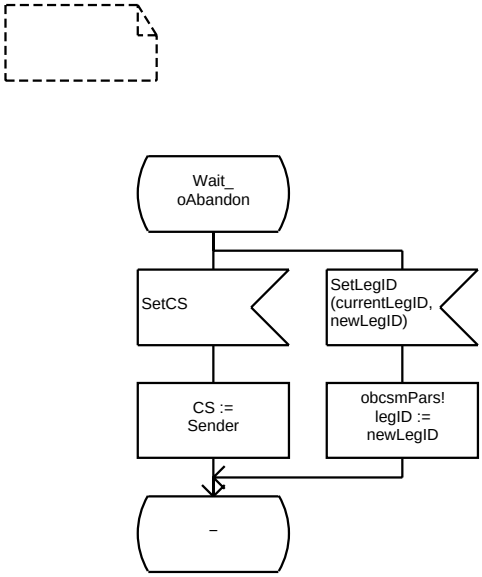


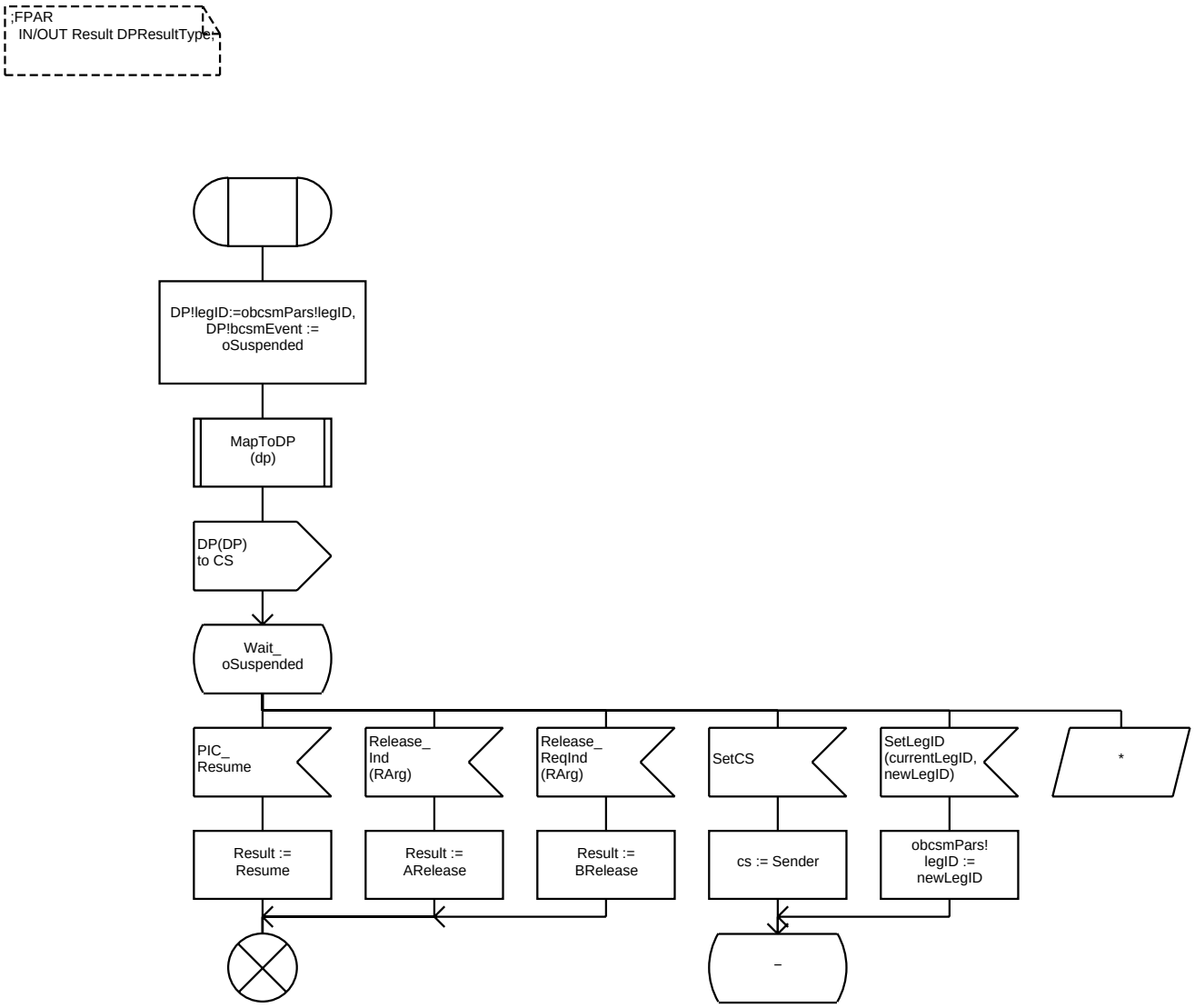

Wait_Dig_by_Dig, Wait_Enbloc
redefined PIC (PIC)
MapFromPIC (pic)
PIC! pic
Collect_ Information
Analyse_ Information
Select_ Route
Release_ Call, Disconnect_ Leg
O_ Exception
Send_ STUI
Else
cdPNPresent (pic!routingInfo)
true
else
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
obcsmPars!routingInfo!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

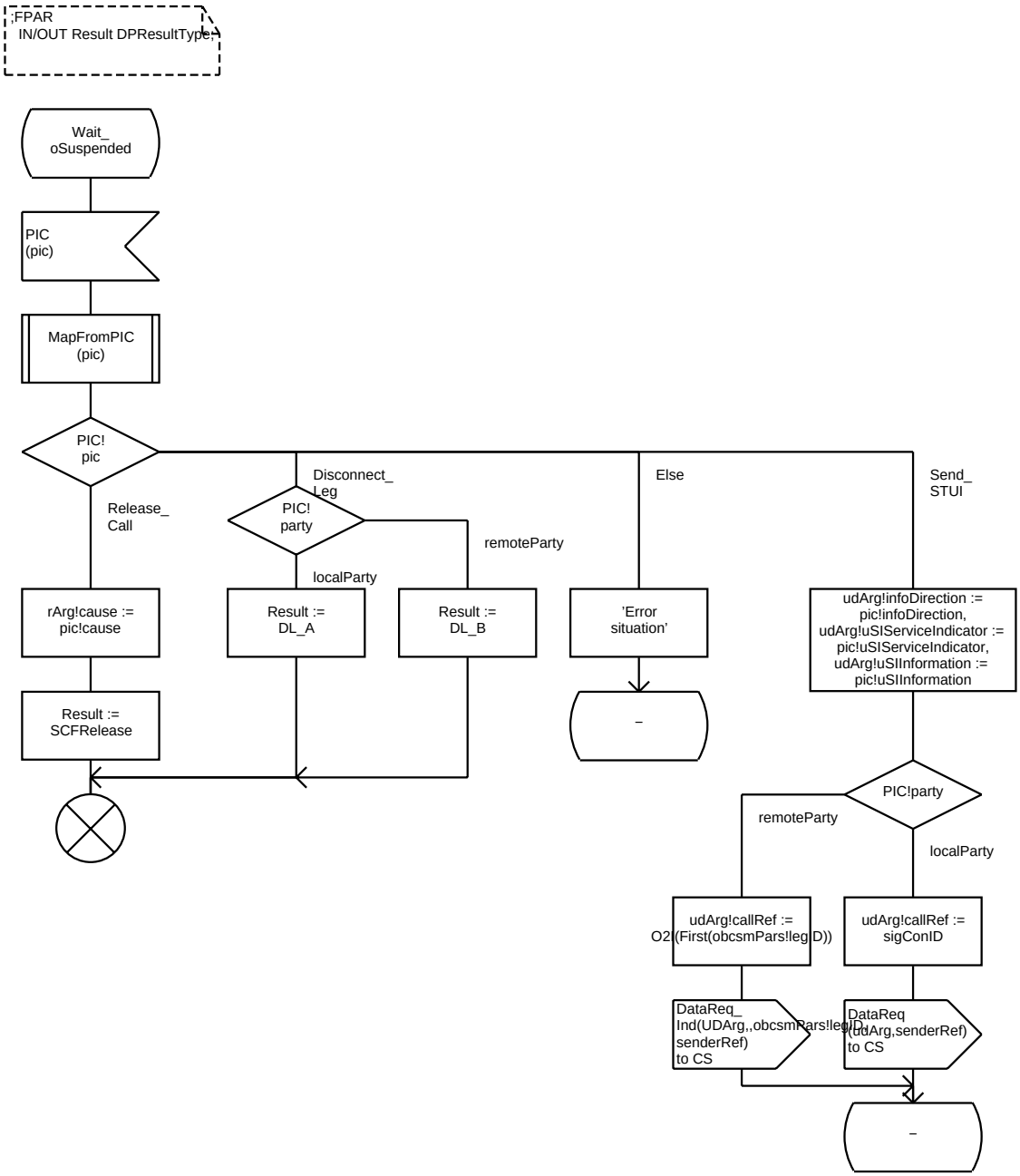

Wait_Dig_by_Dig,
Wait_Enbloc
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

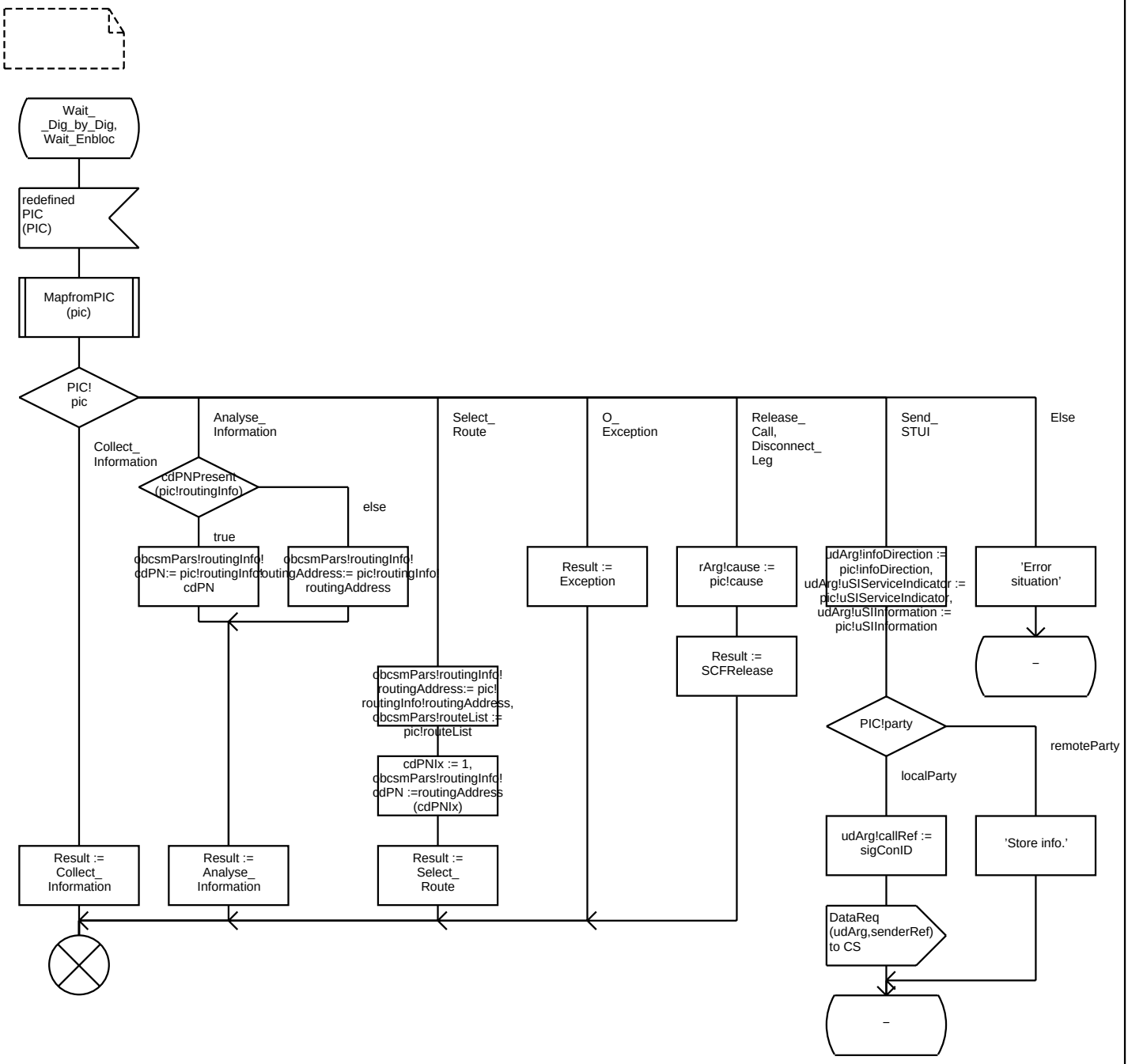

Wait_
oAnswer
redefined
PIC
(PIC)
MapFromPIC
(pic)
PIC!
pic
Release_
Call
Disconnect_
Leg
O_
Exception
Send_
STUI
Else
PIC!
party
localParty
remoteParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
rArg!cause :=
pic!cause
Result :=
DL_A
Result :=
DL_B
Result :=
Exception
'Error
situation'
Result :=
SCFRelease
PIC!party
remoteParty
localParty
udArg!callRef :=
O2(First(obcsmPars!legID))
udArg!callRef :=
O2(First(obcsmPars!legID))
DataReq_
Ind(UDArg,,obcsmPars!
legID,senderRef)
to CS
DataReq
(udArg,senderRef)
to CS

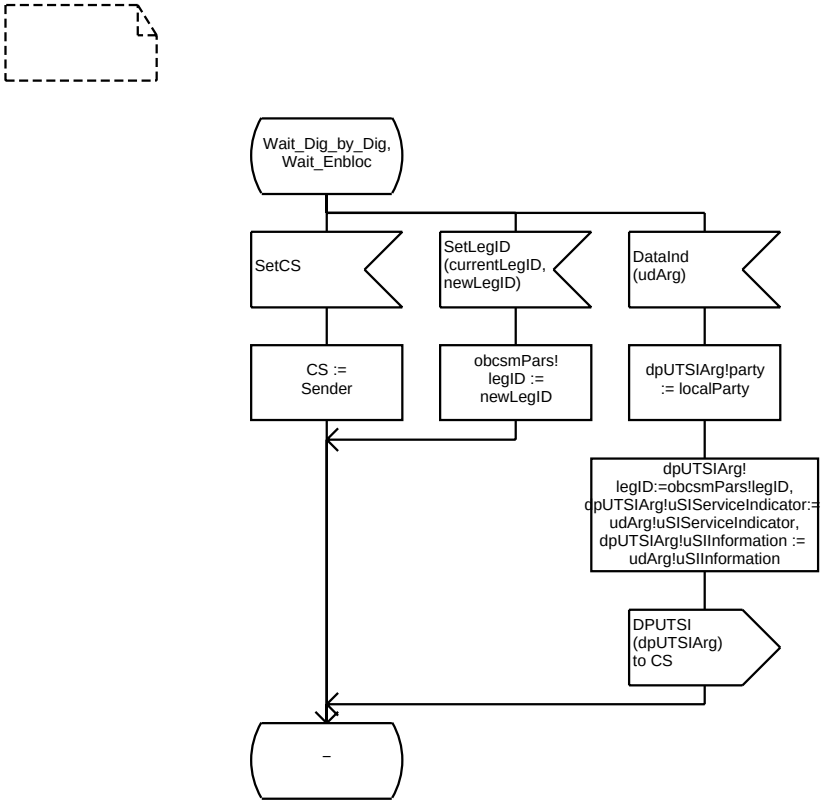

Wait_
oAnswer
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
DataReqInd
(UDArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

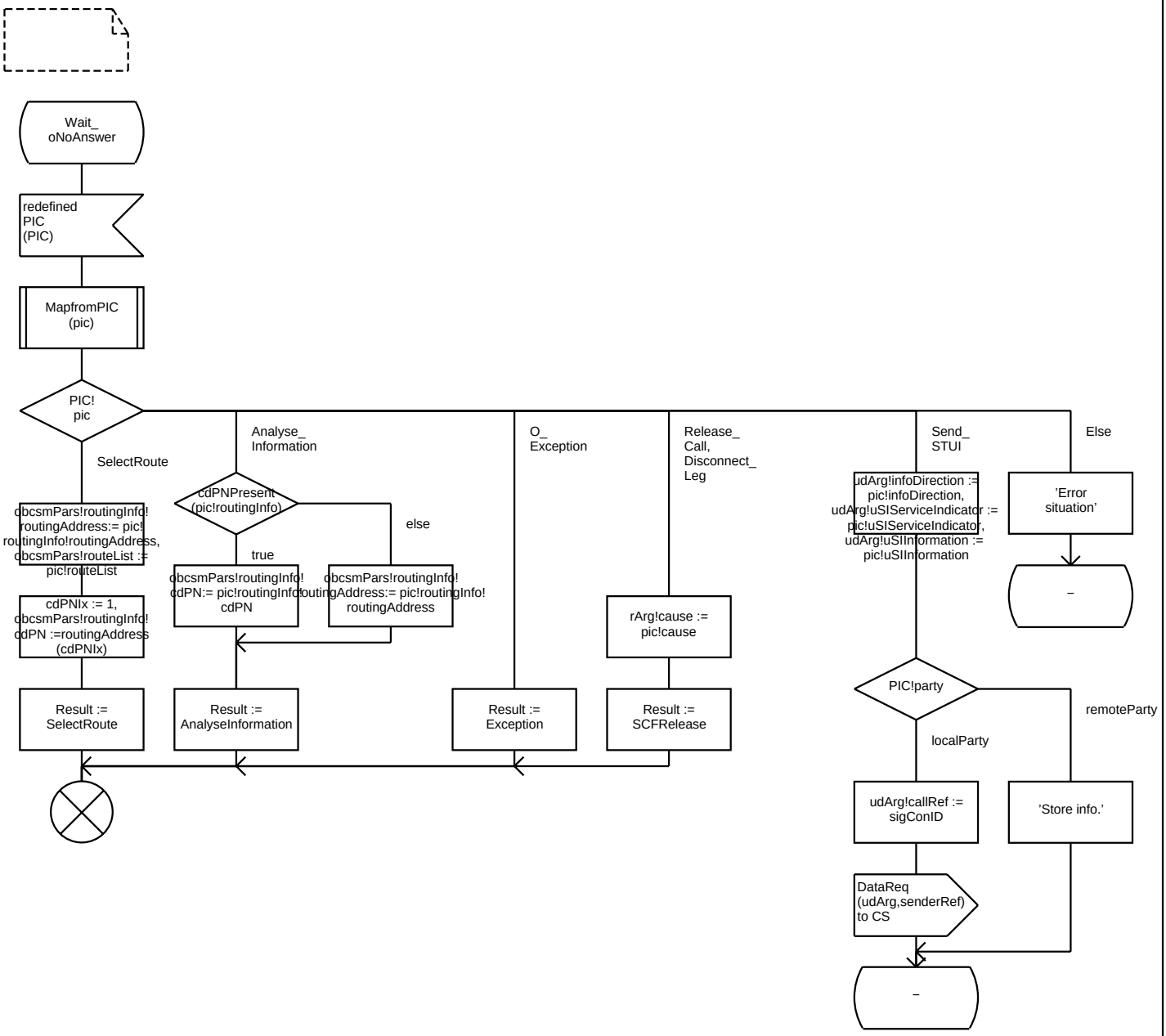

Wait_
oAbandon
SetCS
SetLegID
(currentLegID,
newLegID)
CS :=
Sender
obcsmPars!
legID :=
newLegID
–

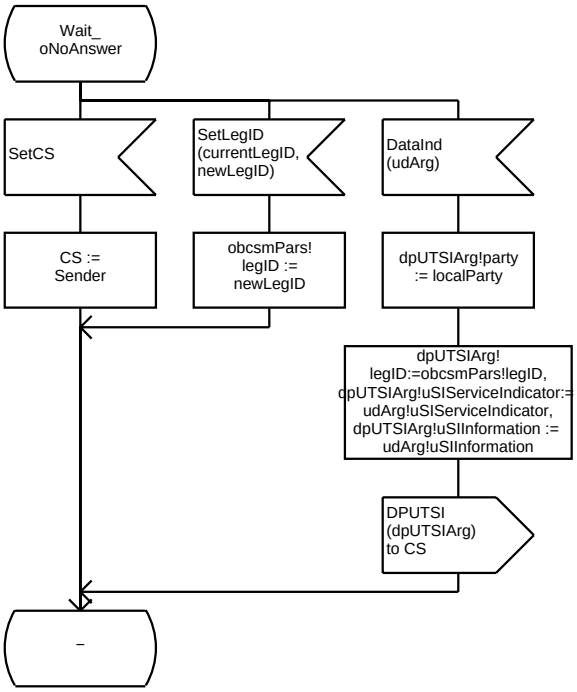

;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
oSuspended
MapToDP
(dp)
DP(DP)
to CS
Wait_
oSuspended
PIC_
Resume
Release_
Ind
(RArg)
Release_
ReqInd
(RArg)
SetCS
SetLegID
(currentLegID,
newLegID)
*
Result :=
Resume
Result :=
ARelease
Result :=
BRelease
cs := Sender
obcsmPars!
legID :=
newLegID
–

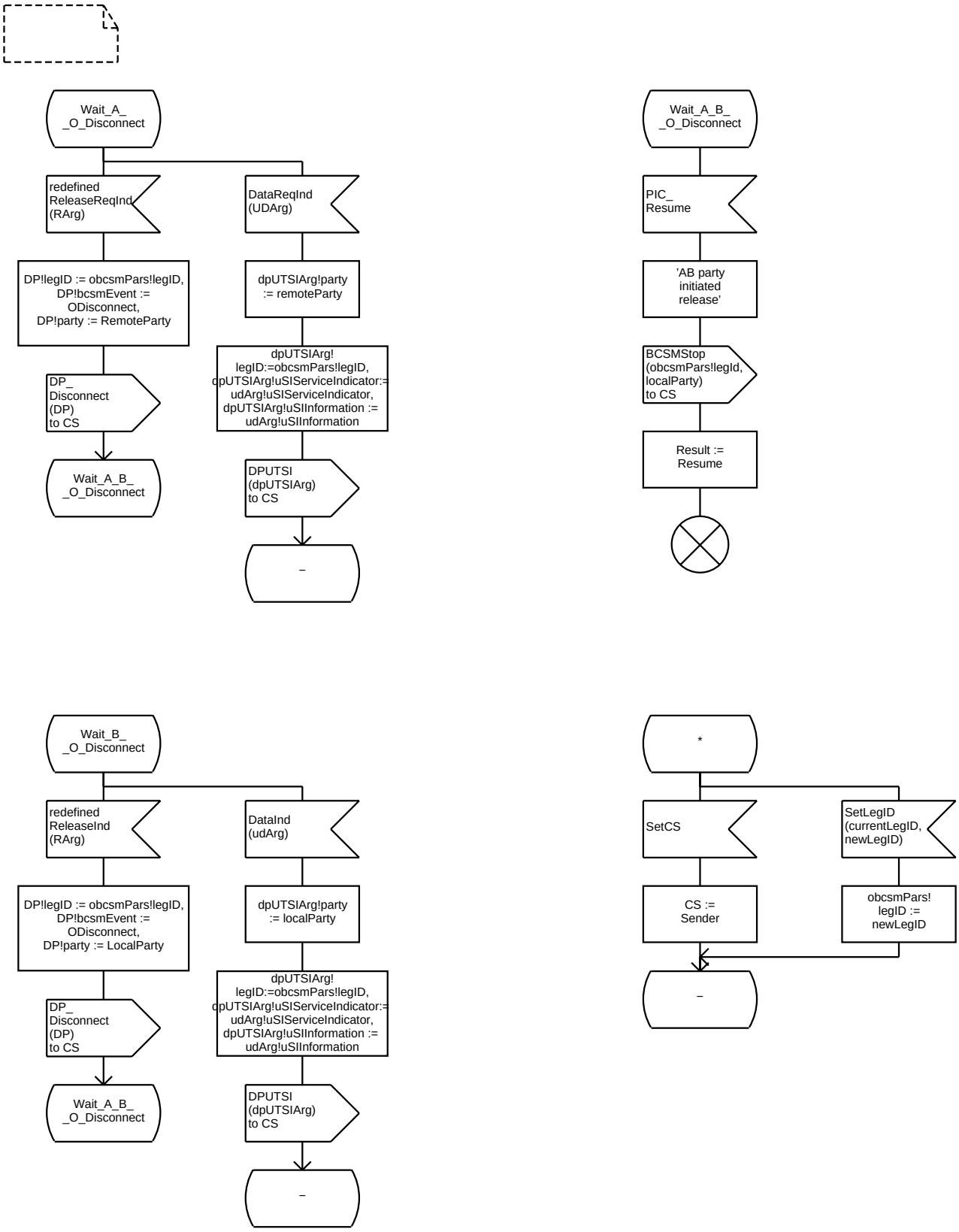

;FPAR
IN/OUT Result DPResultType;
Wait_
oSuspended
PIC
(pic)
MapFromPIC
(pic)
PIC!
pic
Release_
Call
Disconnect_
Leg
PIC!
party
remoteParty
localParty
Else
Send_
STUI
rArg!cause :=
pic!cause
Result :=
DL_A
Result :=
DL_B
'Error
situation'
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
Result :=
SCFRelease
–
PIC!party
remoteParty
localParty
udArg!callRef :=
O2(First(obcsmPars!legID))
udArg!callRef :=
sigConID
DataReq_
Ind(UDArg,,obcsmPars!legID,
senderRef)
to CS
DataReq
(udArg,senderRef)
to CS
–

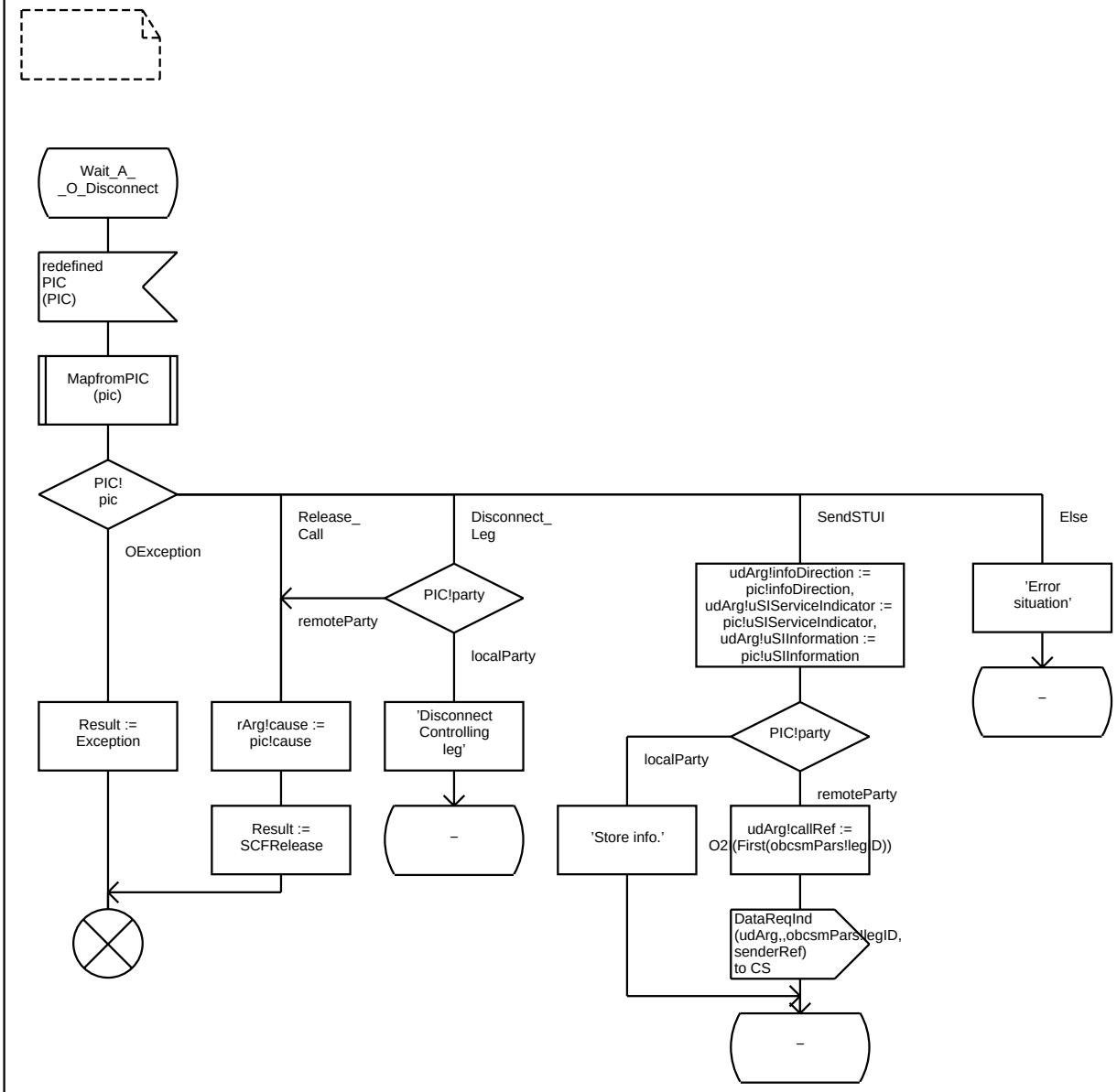

Wait_
_Dig_by_Dig,
Wait_Enbloc
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Collect_
Information
Analyse_
Information
Select_
Route
O_
Exception
Release_
Call,
Disconnect_
Leg
Send_
STUI
Else
cdPNPresent
(pic!routingInfo)
true
else
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Exception
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
SCFRelease
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
PIC!party
remoteParty
localParty
udArg!callRef :=
sigConID
'Store info.'
DataReq
(udArg,senderRef)
to CS
Result :=
Collect_
Information
Result :=
Analyse_
Information
Result :=
Select_
Route

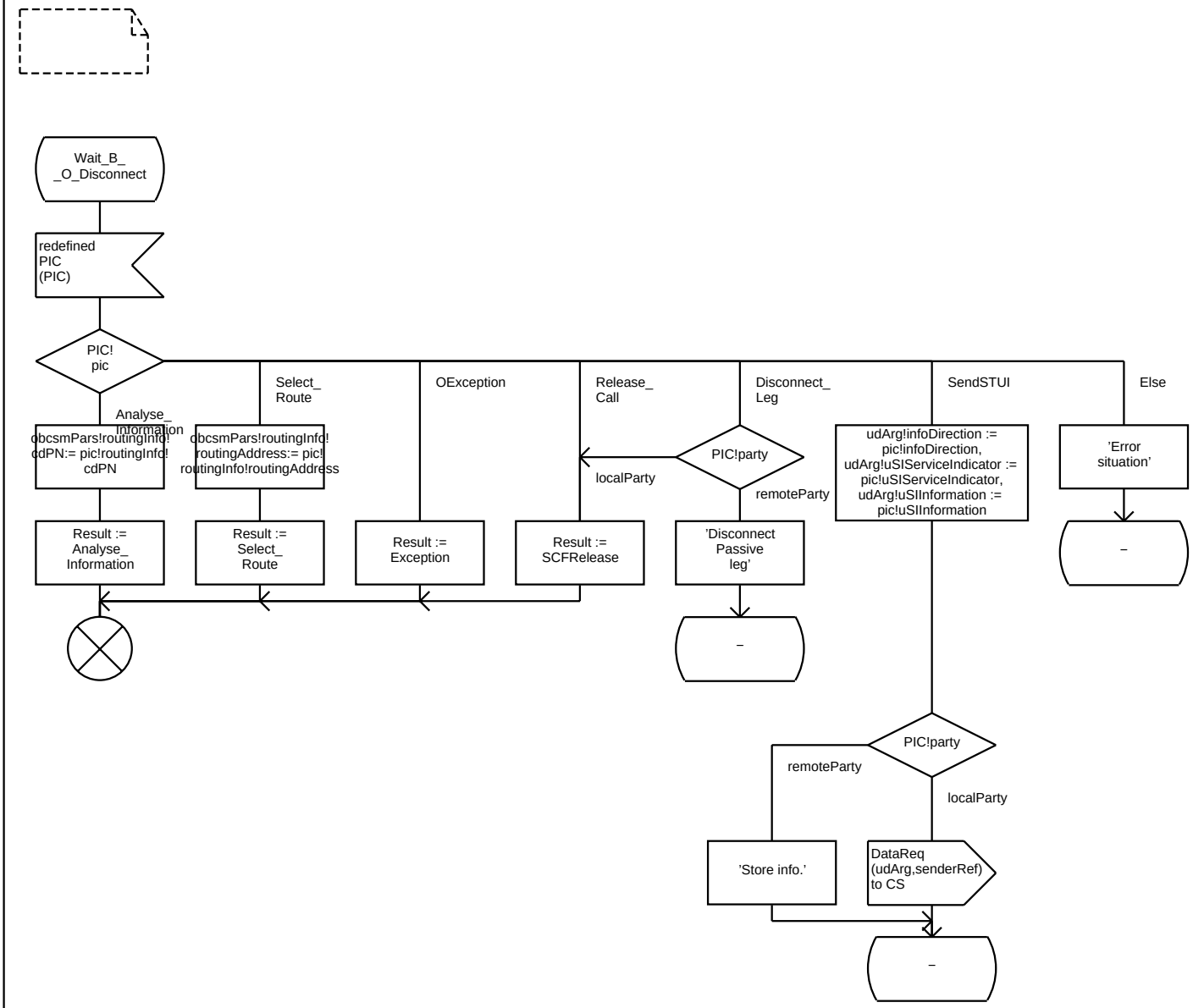

Wait_Dig_by_Dig,
Wait_Enbloc
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

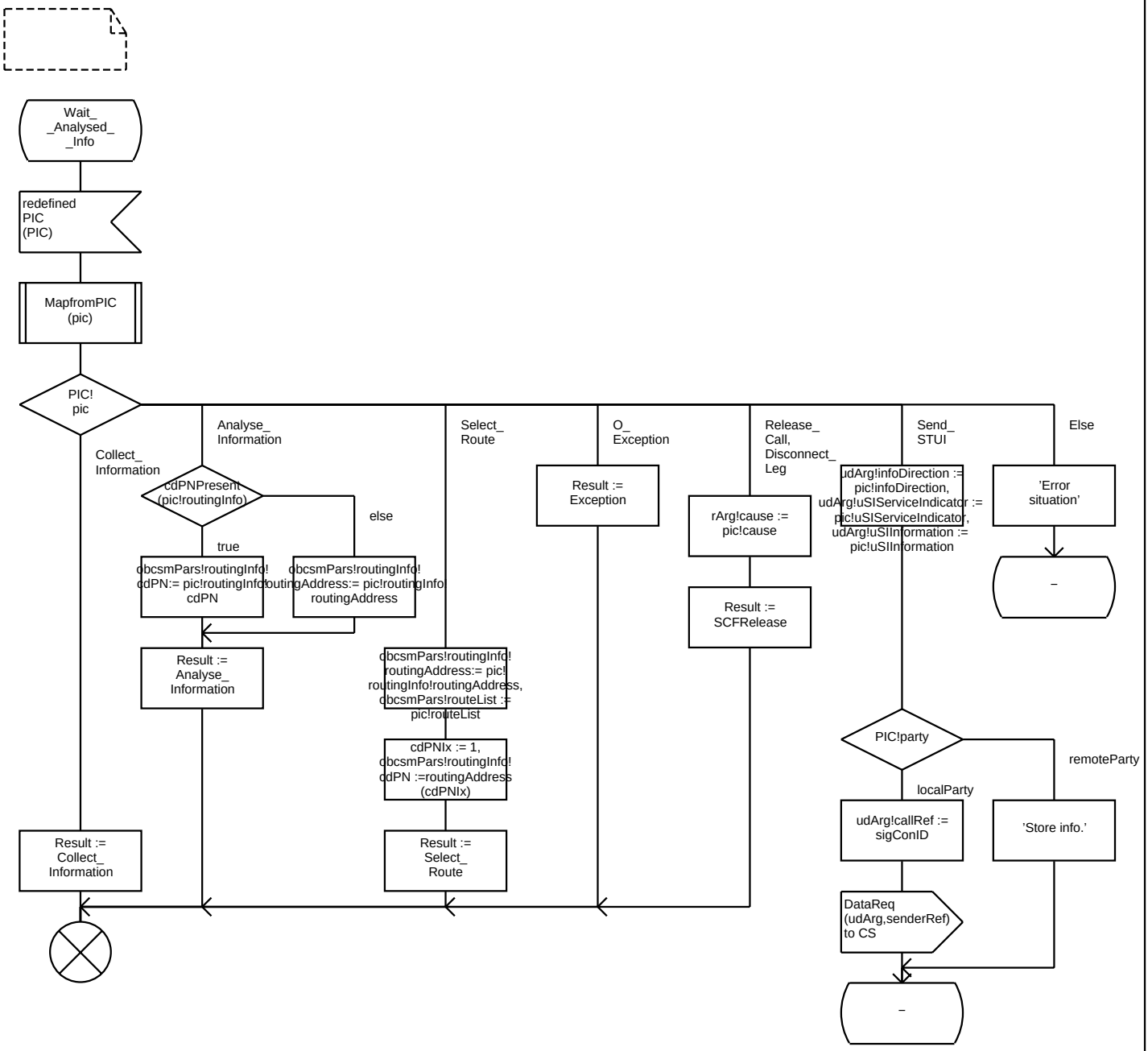

Wait_
oNoAnswer
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
SelectRoute
Analyse_
Information
O_
Exception
Release_
Call,
Disconnect_
Leg
Send_
STUI
Else
cdPNPresent
(pic!routingInfo)
else
true
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
–
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
rArg!cause :=
pic!cause
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
PIC!party
remoteParty
localParty
Result :=
SelectRoute
Result :=
AnalyseInformation
Result :=
Exception
Result :=
SCFRelease
udArg!callRef :=
sigConID
'Store info.'
DataReq
(udArg,senderRef)
to CS
–

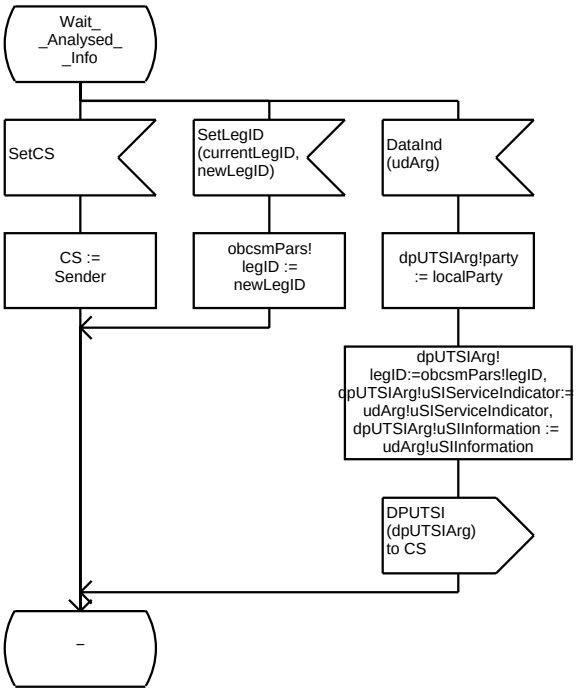

Wait_
oNoAnswer
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

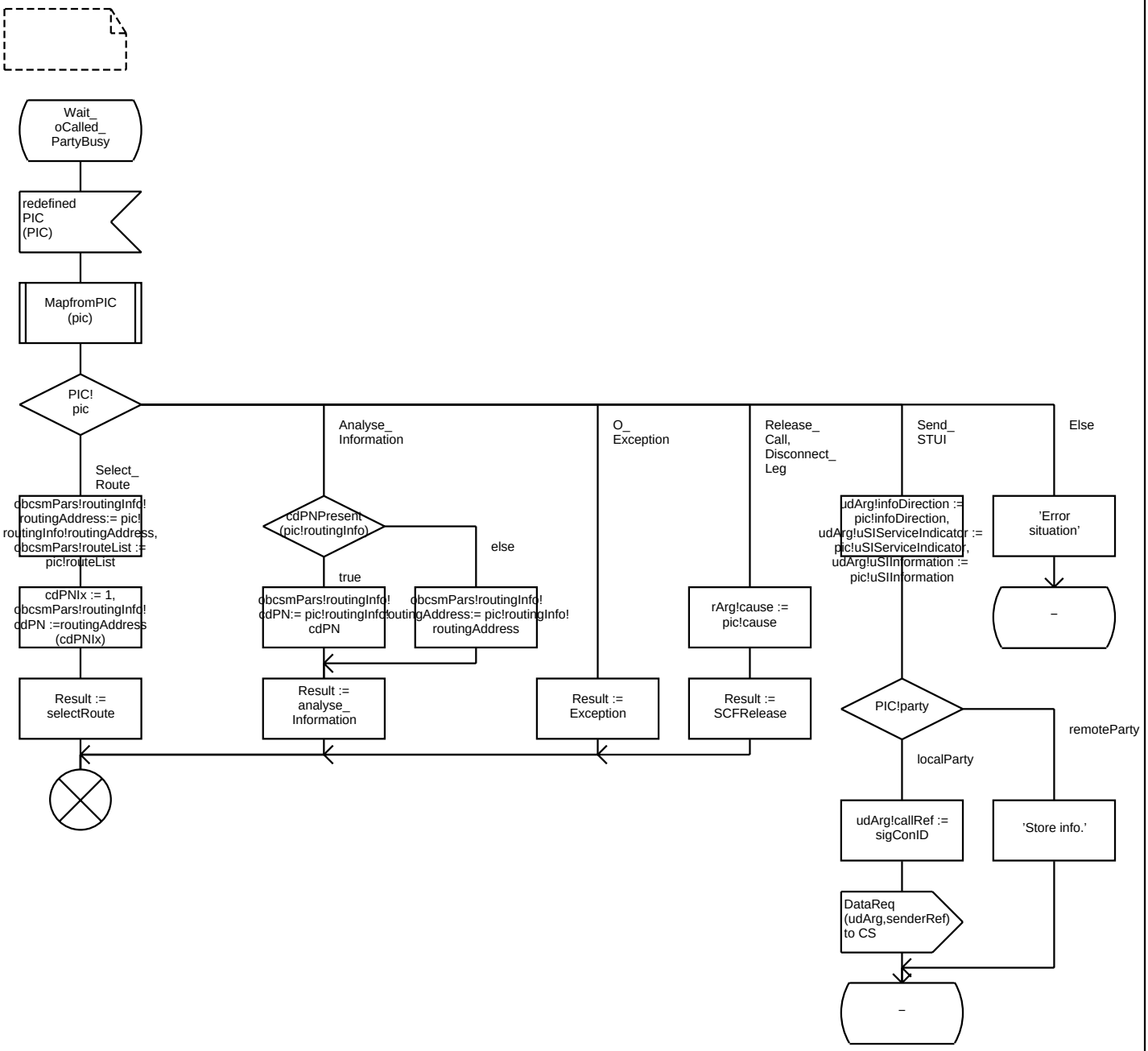

Wait_A_
_O_Disconnect
redefined
ReleaseReqInd
(RArg)
DP!legID := obcsmPars!legID,
DP!bcsmEvent :=
ODisconnect,
DP!party := RemoteParty
DP_
Disconnect
(DP)
to CS
Wait_A_B_
_O_Disconnect
DataReqInd
(UDArg)
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–
Wait_A_B_
_O_Disconnect
PIC_
Resume
'AB party
initiated
release'
BCSMStop
(obcsmPars!legId,
localParty)
to CS
Result :=
Resume
Wait_B_
_O_Disconnect
redefined
ReleaseInd
(RArg)
DP!legID := obcsmPars!legID,
DP!bcsmEvent :=
ODisconnect,
DP!party := LocalParty
DP_
Disconnect
(DP)
to CS
Wait_A_B_
_O_Disconnect
DataInd
(udArg)
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–
*
SetCS
CS :=
Sender
SetLegID
(currentLegID,
newLegID)
obcsmPars!
legID :=
newLegID
–

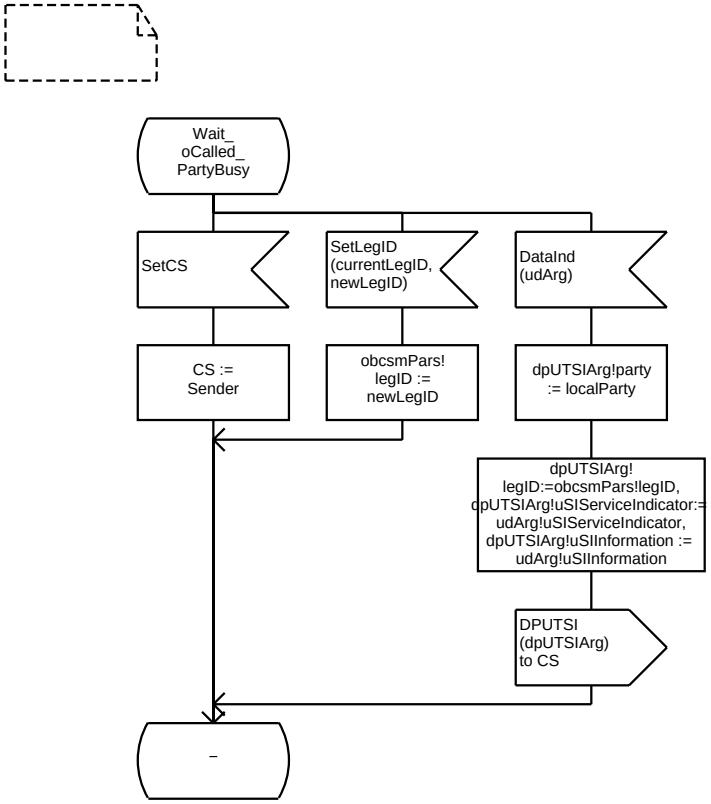

Wait_A_
_O_Disconnect
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
OException
Release_
Call
Disconnect_
Leg
SendSTUI
Else
PIC!party
remoteParty
localParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
Exception
rArg!cause :=
pic!cause
'Disconnect
Controlling
leg'
PIC!party
localParty
remoteParty
Result :=
SCFRelease
'Store info.'
udArg!callRef :=
O2(First(obcsmPars!legID))
DataReqInd
(udArg,,obcsmPars!legID,
senderRef)
to CS

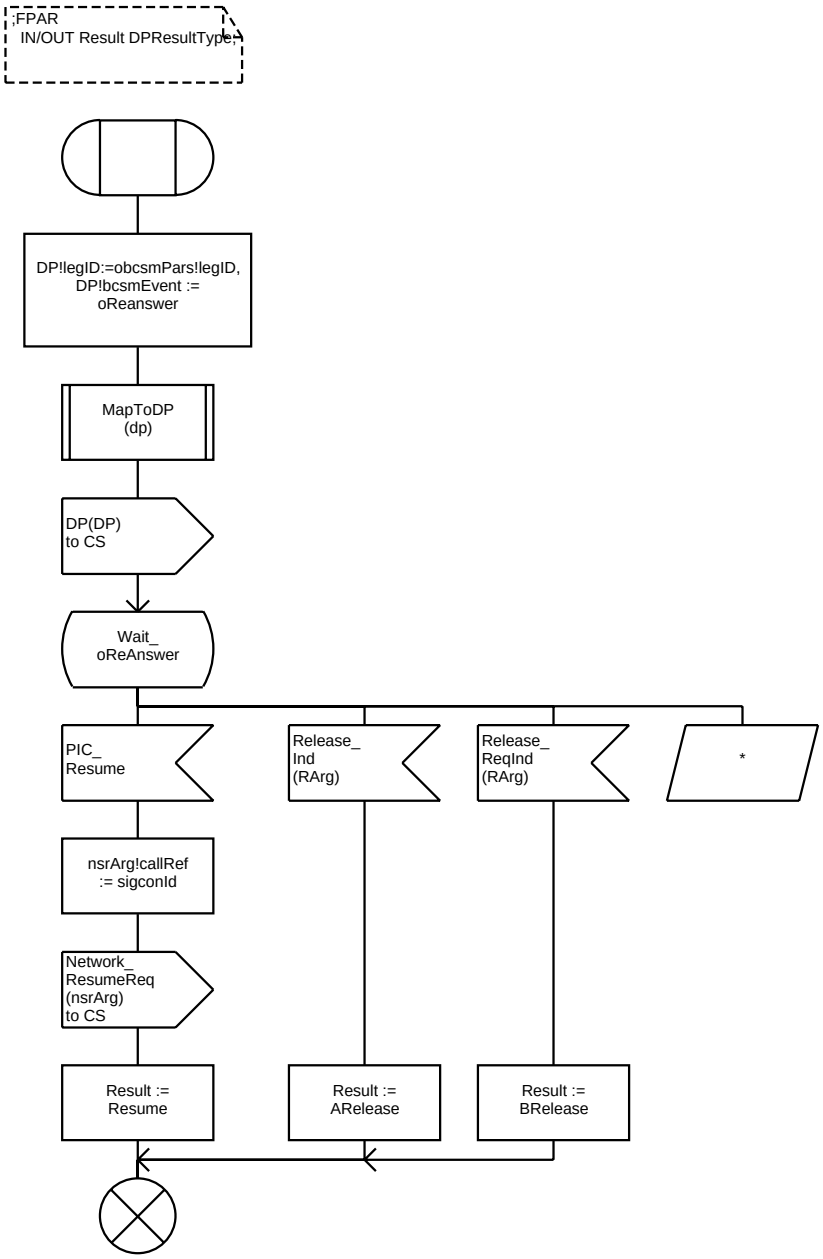

Wait_B_
_O_Disconnect
redefined
PIC
(PIC)
PIC!
pic
Analyse_
Information
Select_
Route
OException
Release_
Call
Disconnect_
Leg
SendSTUI
Else
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress
PIC!party
localParty
remoteParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
Analyse_
Information
Result :=
Select_
Route
Result :=
Exception
Result :=
SCFRelease
'Disconnect
Passive
leg'
–
–
PIC!party
remoteParty
localParty
'Store info.'
DataReq
(udArg,senderRef)
to CS
–

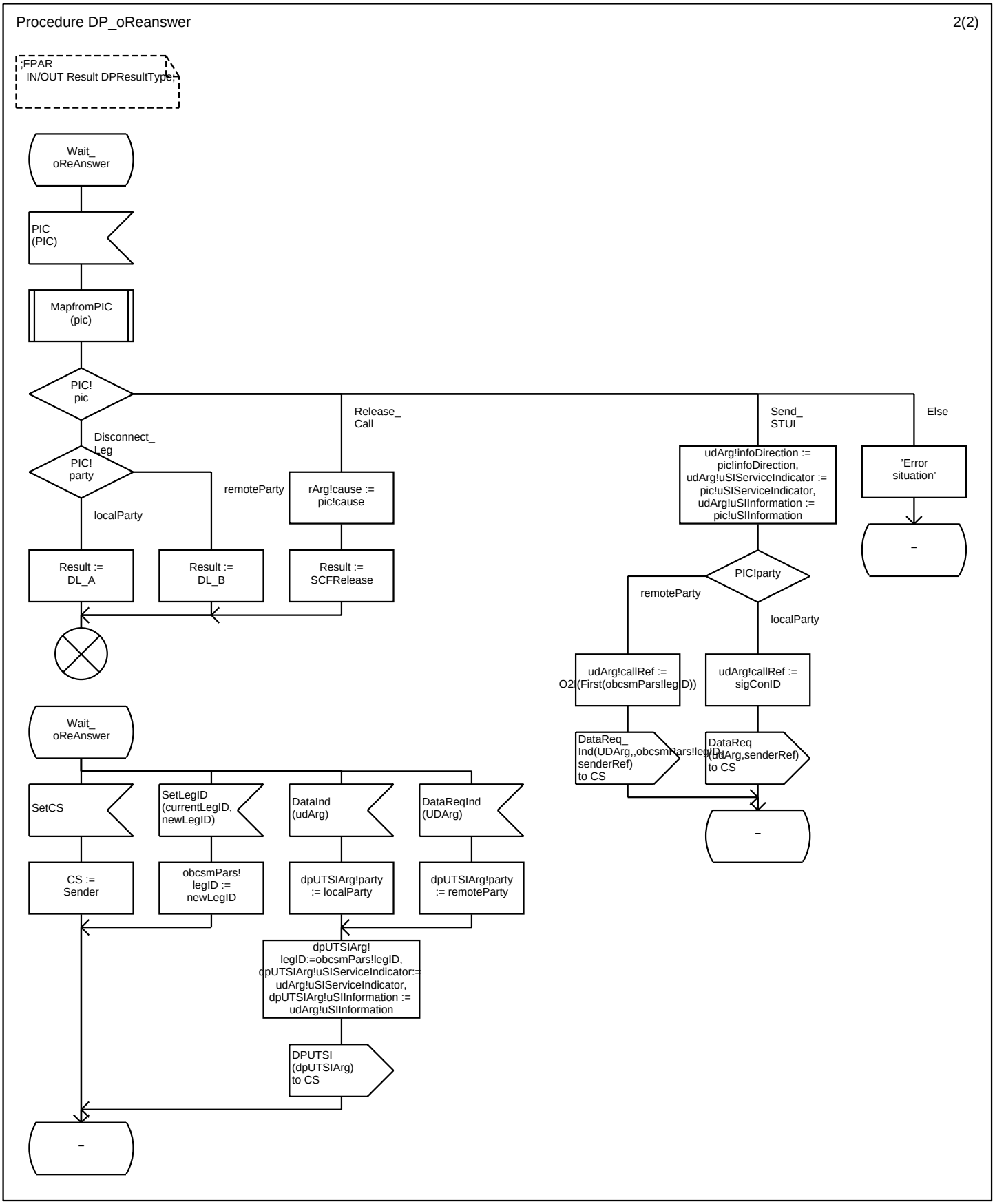

Wait_
Analysed
_Info
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Collect_
Information
Analyse_
Information
Select_
Route
O_
Exception
Release_
Call,
Disconnect_
Leg
Send_
STUI
Else
cdPNPresent
(pic!routingInfo)
true
else
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!
routingAddress:= pic!routingInfo!
routingAddress
Result :=
Analyse_
Information
obcsmPars!routingInfo!
routingAddress:= pic!
routingInfo!routingAddress,
obcsmPars!routeList :=
pic!routeList
cdPNIx := 1,
obcsmPars!routingInfo!
cdPN :=routingAddress
(cdPNIx)
Result :=
Select_
Route
Result :=
Exception
rArg!cause :=
pic!cause
Result :=
SCFRelease
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
–
PIC!party
remoteParty
localParty
udArg!callRef :=
sigConID
'Store info.'
DataReq
(udArg,senderRef)
to CS
–
Result :=
Collect_
Information

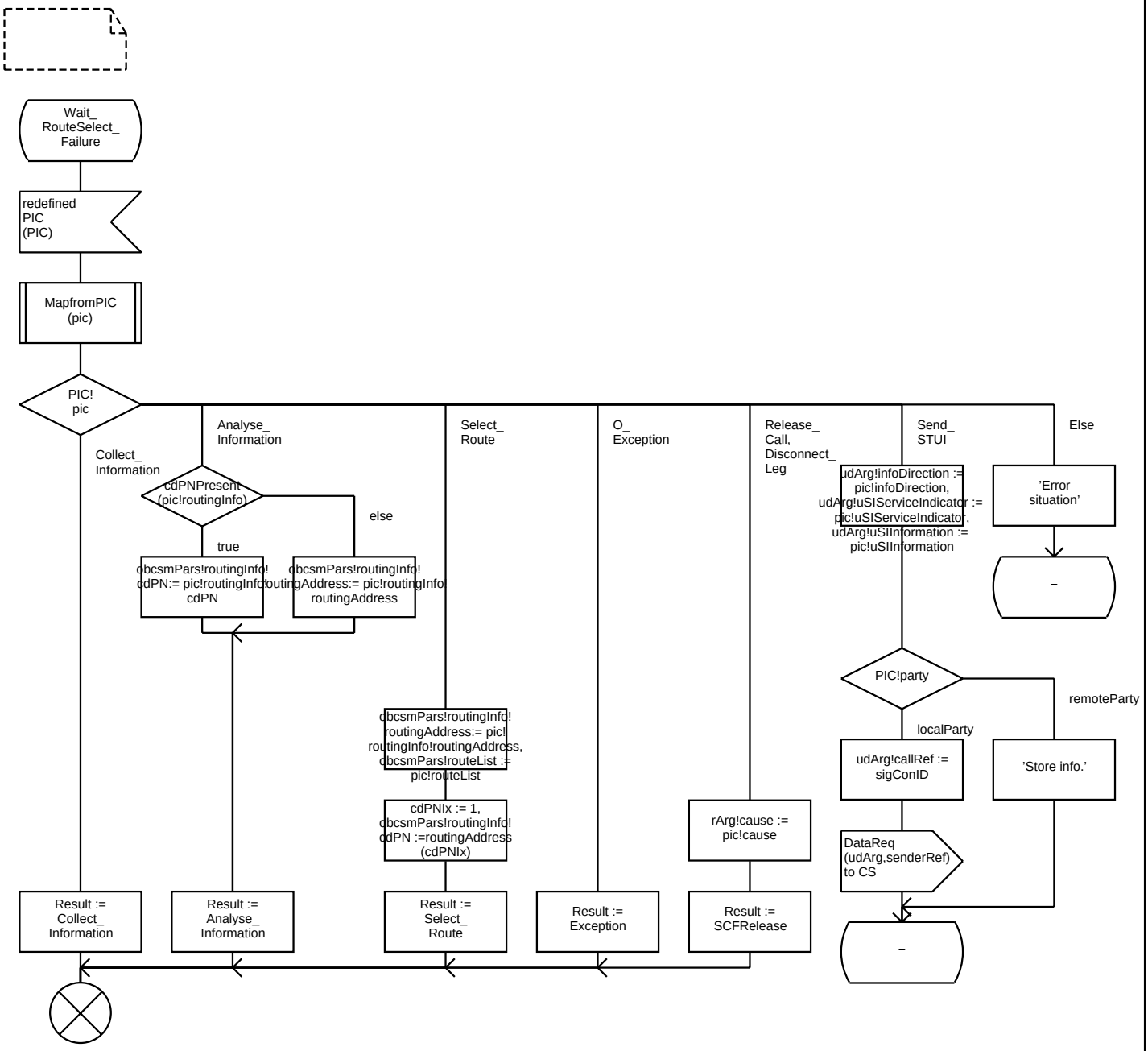

Wait_
Analysed
_Info
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

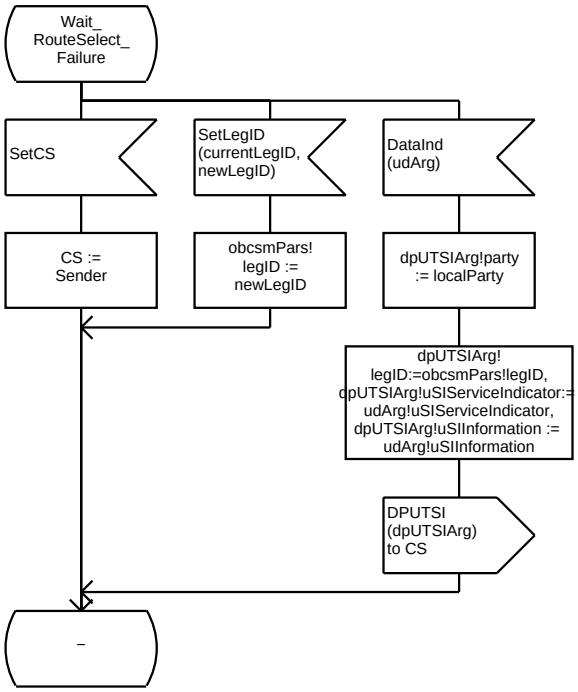

Wait_ oCalled_ PartyBusy
redefined PIC (PIC)
MapfromPIC (pic)
PIC! pic
Select_ Route
obcsmPars!routingInfo! routingAddress:= pic! routingInfo!routingAddress, obcsmPars!routeList := pic!routeList
cdPNIx := 1, obcsmPars!routingInfo! cdPN :=routingAddress (cdPNIx)
Result := selectRoute
Analyse_ Information
cdPNPresent (pic!routingInfo)
true
else
obcsmPars!routingInfo! cdPN:= pic!routingInfo! cdPN
obcsmPars!routingInfo! routingAddress:= pic!routingInfo! routingAddress
Result := analyse_ Information
O_ Exception
Result := Exception
Release_ Call, Disconnect_ Leg
rArg!cause := pic!cause
Result := SCFRelease
Send_ STUI
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
PIC!party
localParty
udArg!callRef := sigConID
DataReq (udArg,senderRef) to CS
–
remoteParty
'Store info.'
Else
'Error situation'
–

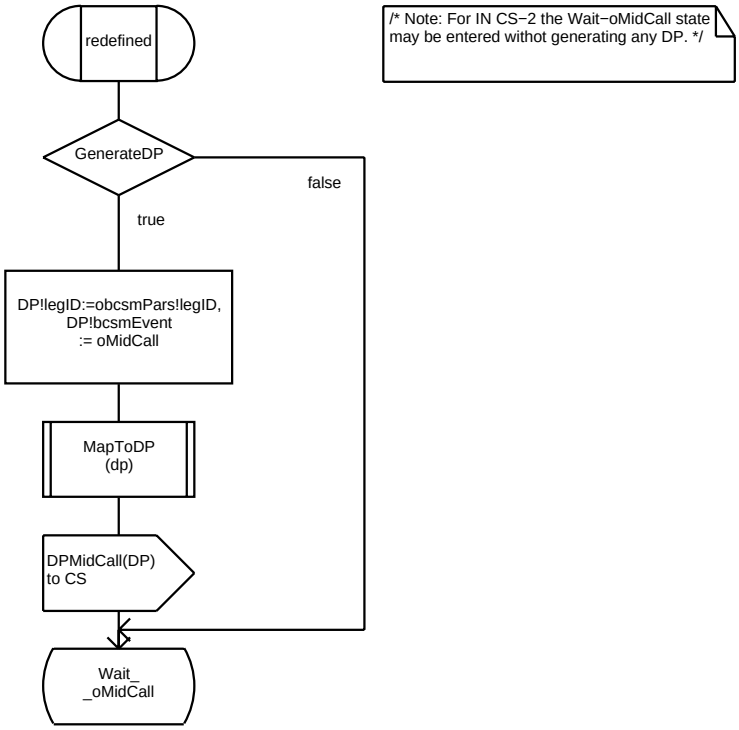

Wait_
oCalled_
PartyBusy
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

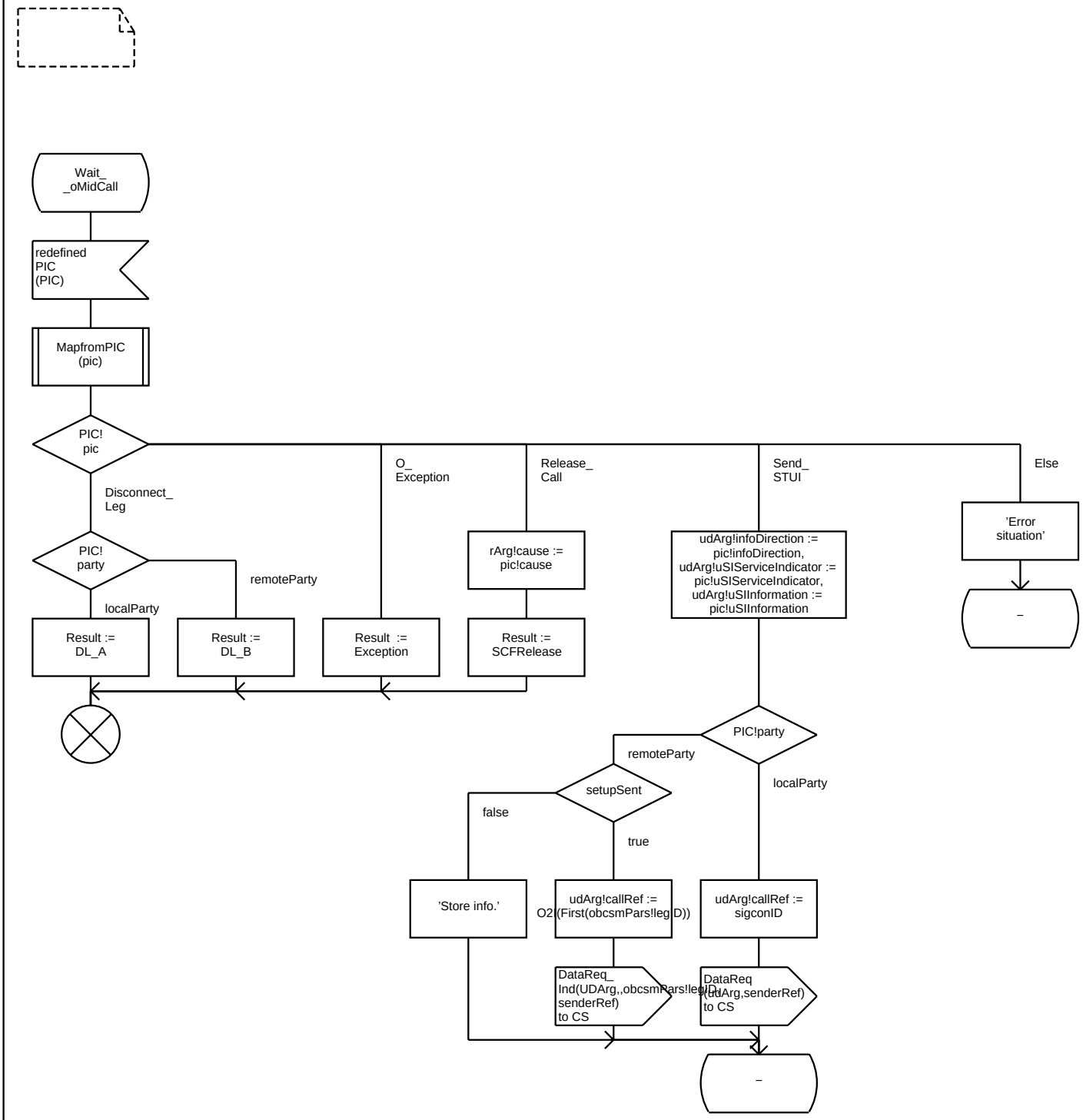

;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
oReanswer
MapToDP
(dp)
DP(DP)
to CS
Wait_
oReAnswer
PIC_
Resume
Release_
Ind
(RArg)
Release_
ReqInd
(RArg)
*
nsrArg!callRef
:= sigconId
Network_
ResumeReq
(nsrArg)
to CS
Result :=
Resume
Result :=
ARelease
Result :=
BRelease

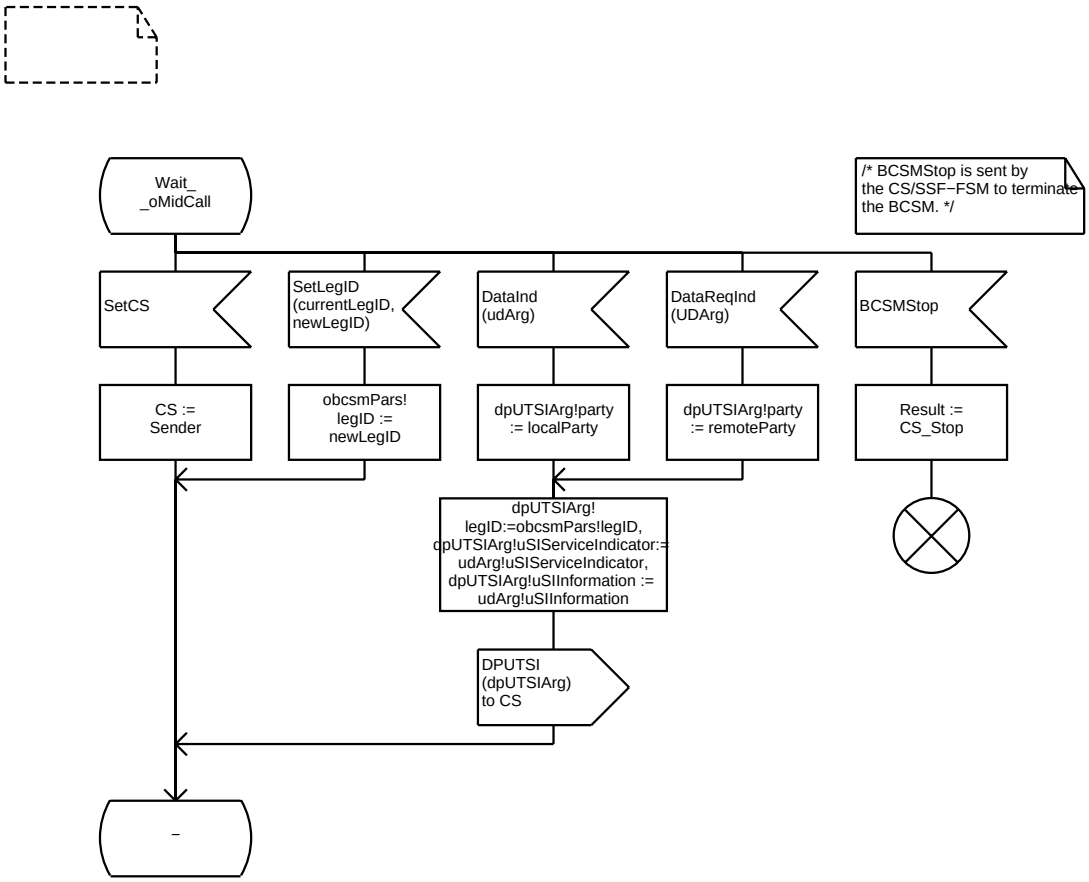

;FPAR
IN/OUT Result DPResultType;
Wait_
oReAnswer
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Release_
Call
Send_
STUI
Else
Disconnect_
Leg
PIC!
party
remoteParty
localParty
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
DL_A
Result :=
DL_B
Result :=
SCFRelease
PIC!party
remoteParty
localParty
udArg!callRef :=
O2(First(obcsmPars!legID))
udArg!callRef :=
sigConID
DataReq_
Ind(UDArg,,obcsmPars!legID,
senderRef)
to CS
DataReq
(udArg,senderRef)
to CS
Wait_
oReAnswer
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
DataReqInd
(UDArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS

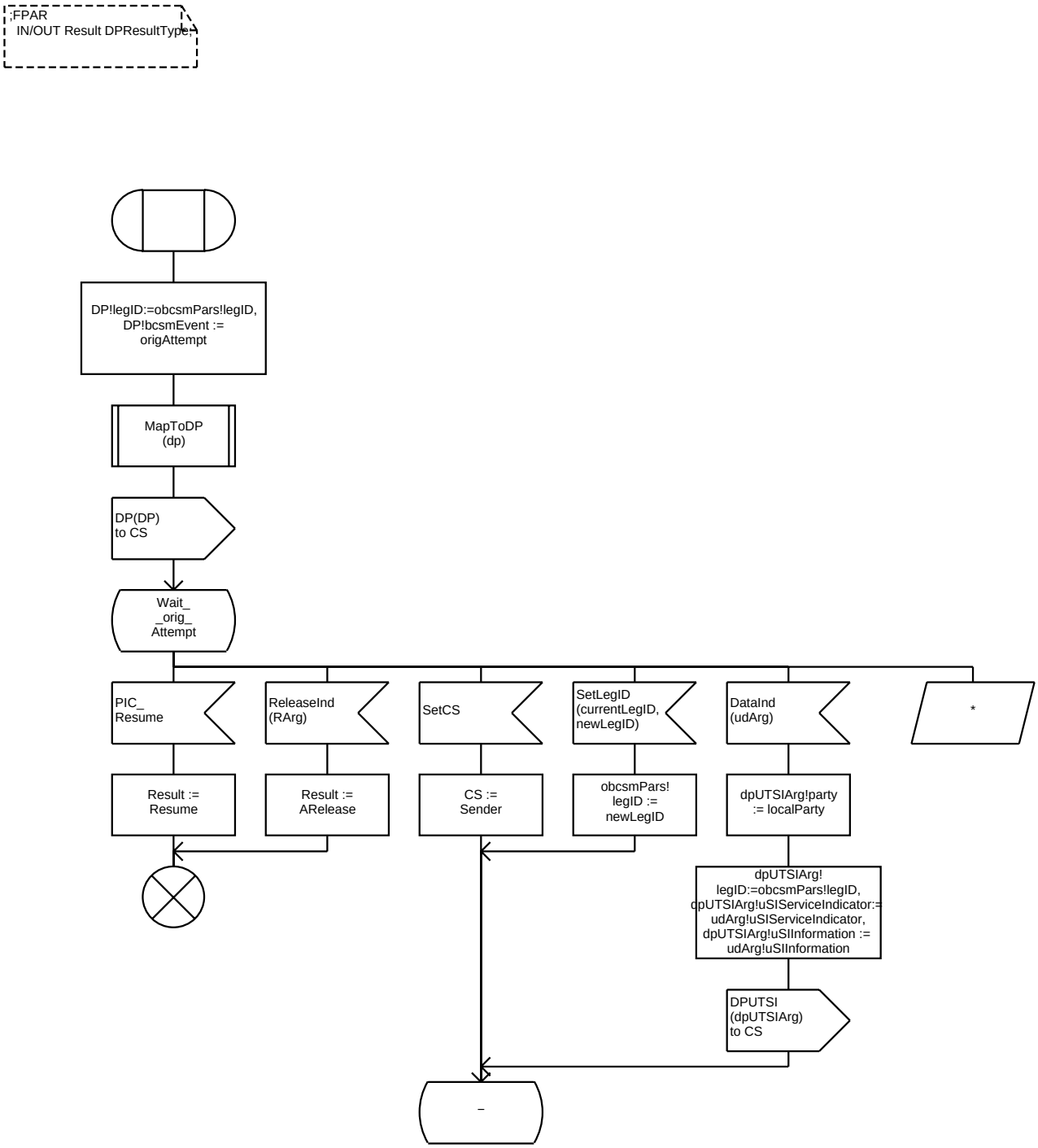

Wait_
RouteSelect_
Failure
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Collect_
Information
Analyse_
Information
Select_
Route
O_
Exception
Release_
Call,
Disconnect_
Leg
Send_
STUI
Else
cdPNPresent
(pic!routingInfo)
else
true
obcsmPars!routingInfo!
cdPN:= pic!routingInfo!
cdPN
obcsmPars!routingInfo!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

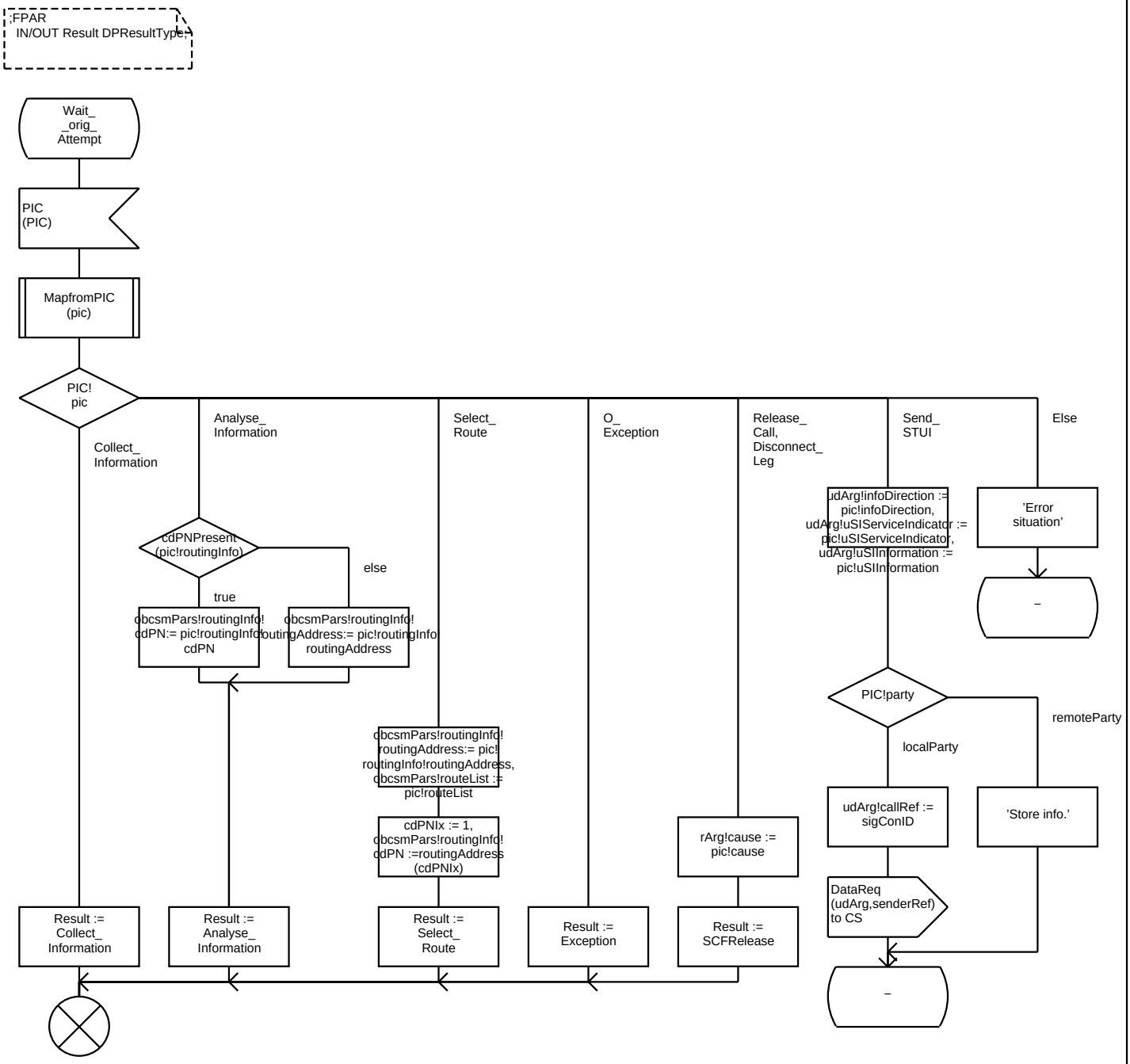

Wait_
RouteSelect_
Failure
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

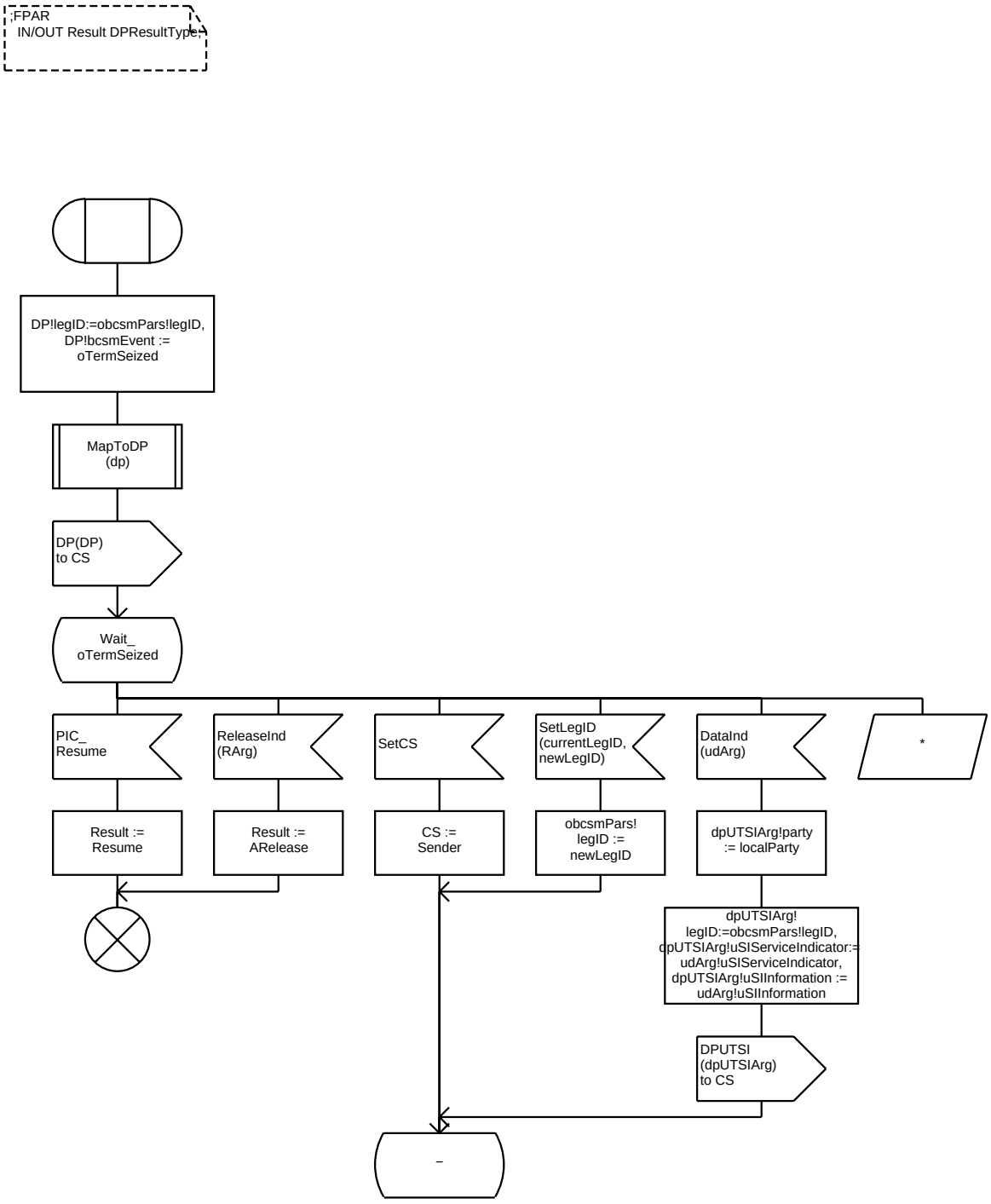

redefined
/* Note: For IN CS−2 the Wait−oMidCall state may be entered withot generating any DP. */
GenerateDP
false
true
DP!legID:=obcsmPars!legID, DP!bcsmEvent := oMidCall
MapToDP (dp)
DPMidCall(DP) to CS
Wait_ _oMidCall

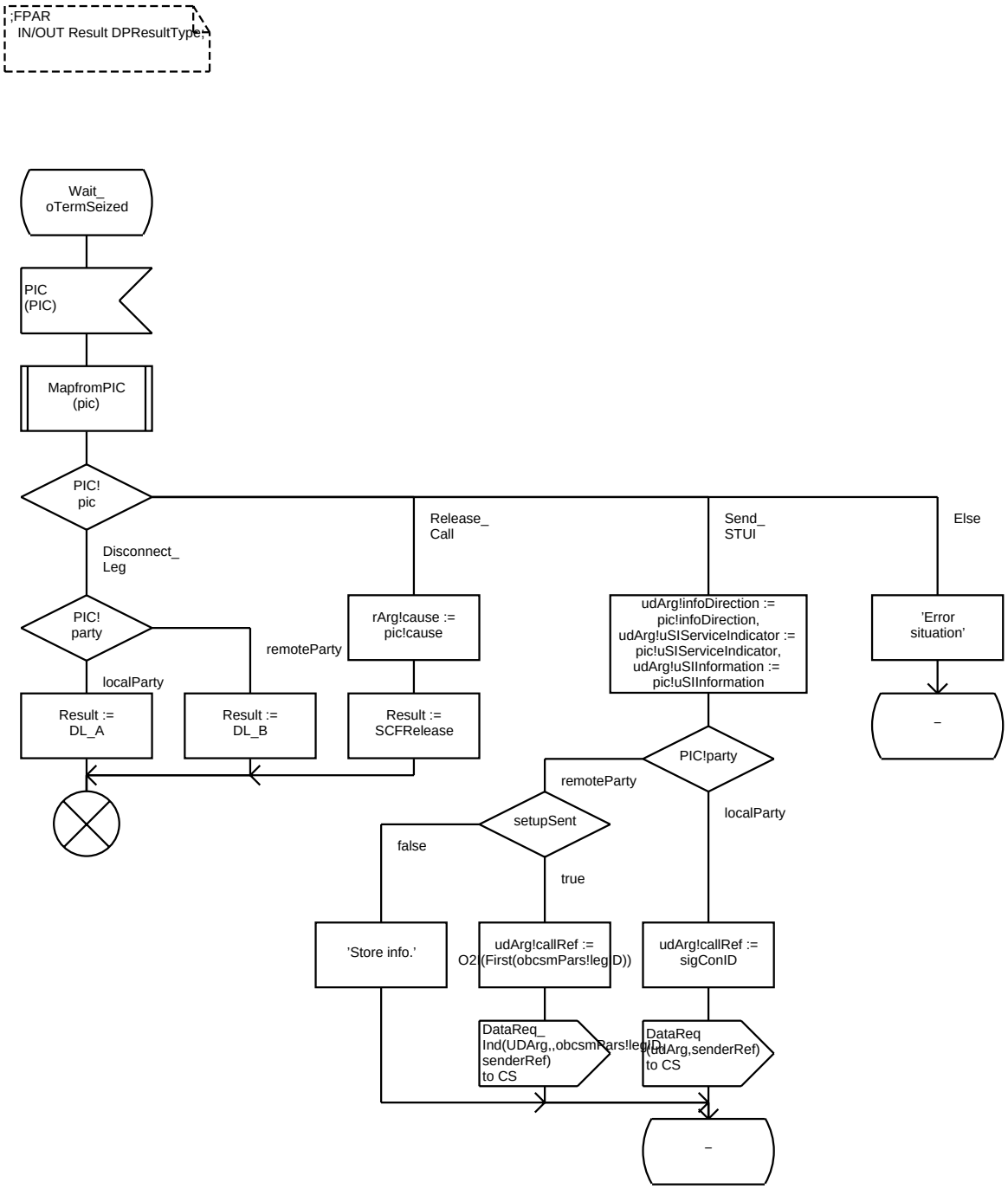

Wait_
_oMidCall
redefined
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Disconnect_
Leg
O_
Exception
Release_
Call
Send_
STUI
Else
PIC!
party
remoteParty
localParty
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
DL_A
Result :=
DL_B
Result :=
Exception
Result :=
SCFRelease
PIC!party
remoteParty
localParty
setupSent
false
true
'Store info.'
udArg!callRef :=
O2(First(obcsmPars!legID))
udArg!callRef :=
sigconID
DataReq_
Ind(UDArg,,obcsmPars!legID,
senderRef)
to CS
DataReq
(udArg,senderRef)
to CS


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


;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
origAttempt
MapToDP
(dp)
DP(DP)
to CS
Wait_
orig
Attempt
PIC_
Resume
ReleaseInd
(RArg)
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
*
Result :=
Resume
Result :=
ARelease
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–


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


;FPAR
IN/OUT Result DPResultType;
DP!legID:=obcsmPars!legID,
DP!bcsmEvent :=
oTermSeized
MapToDP
(dp)
DP(DP)
to CS
Wait_
oTermSeized
PIC_
Resume
ReleaseInd
(RArg)
SetCS
SetLegID
(currentLegID,
newLegID)
DataInd
(udArg)
*
Result :=
Resume
Result :=
ARelease
CS :=
Sender
obcsmPars!
legID :=
newLegID
dpUTSIArg!party
:= localParty
dpUTSIArg!
legID:=obcsmPars!legID,
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DPUTSI
(dpUTSIArg)
to CS
–

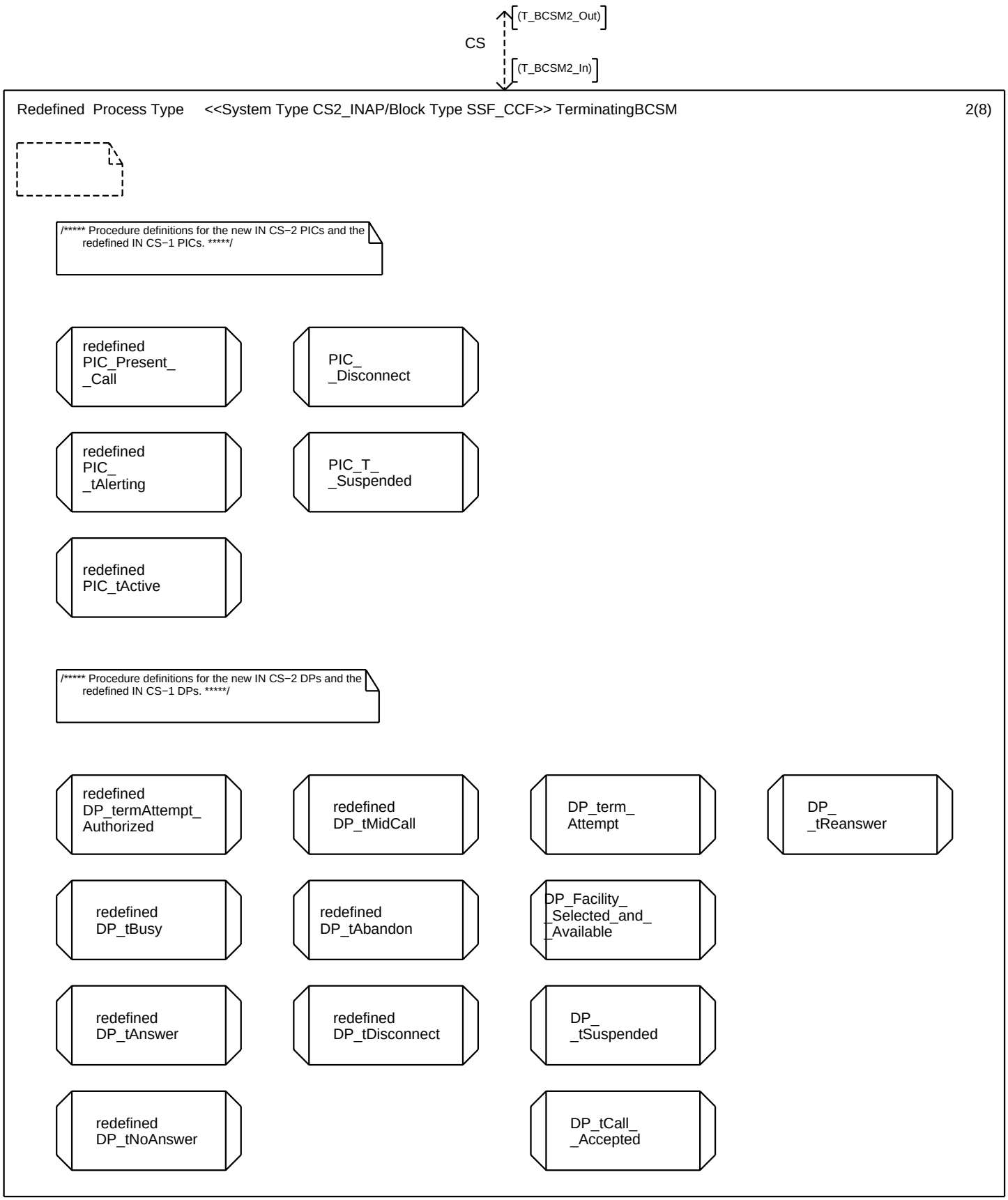

;FPAR
IN/OUT Result DPResultType;
Wait_
oTermSeized
PIC
(PIC)
MapfromPIC
(pic)
PIC!
pic
Disconnect_
Leg
Release_
Call
Send_
STUI
Else
PIC!
party
remoteParty
localParty
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
DL_A
Result :=
DL_B
Result :=
SCFRelease
–
PIC!party
remoteParty
localParty
setupSent
false
true
'Store info.'
udArg!callRef :=
O2(First(obcsmPars!legID))
udArg!callRef :=
sigConID
DataReq_
Ind(UDArg,,obcsmPars!legID,
senderRef)
to CS
DataReq
(udArg,senderRef)
to CS
–

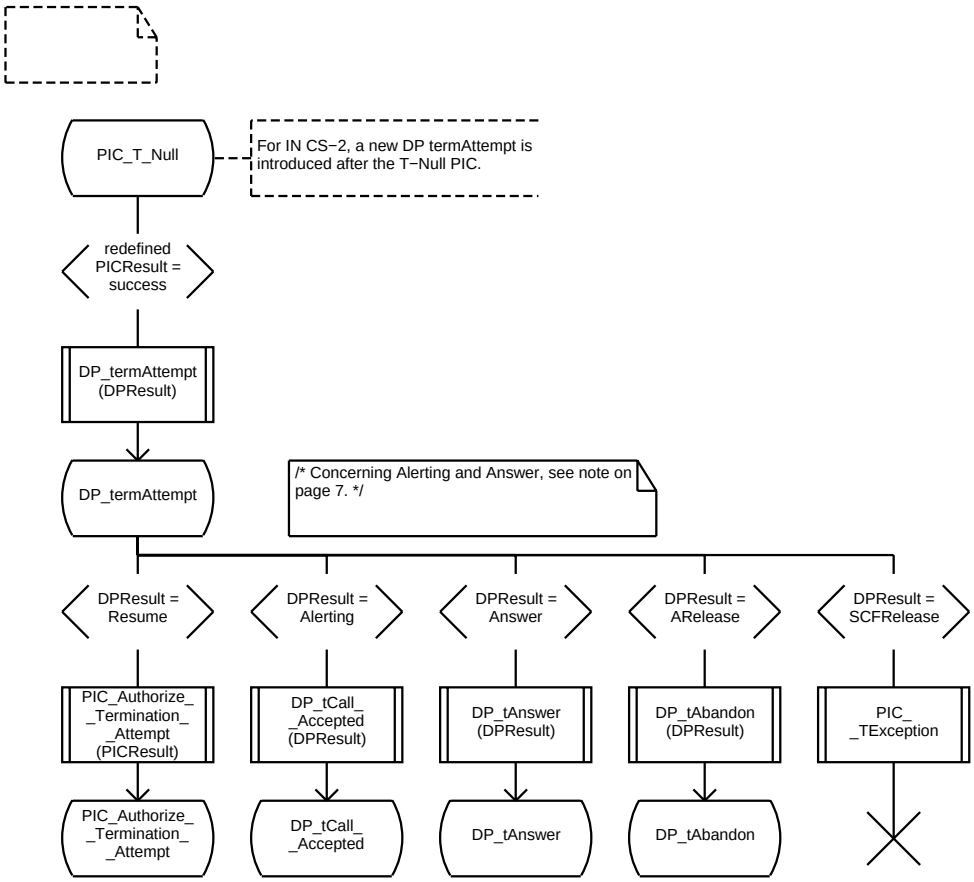
/***** T−BCSM FOR CORE INAP CS−2. *****/

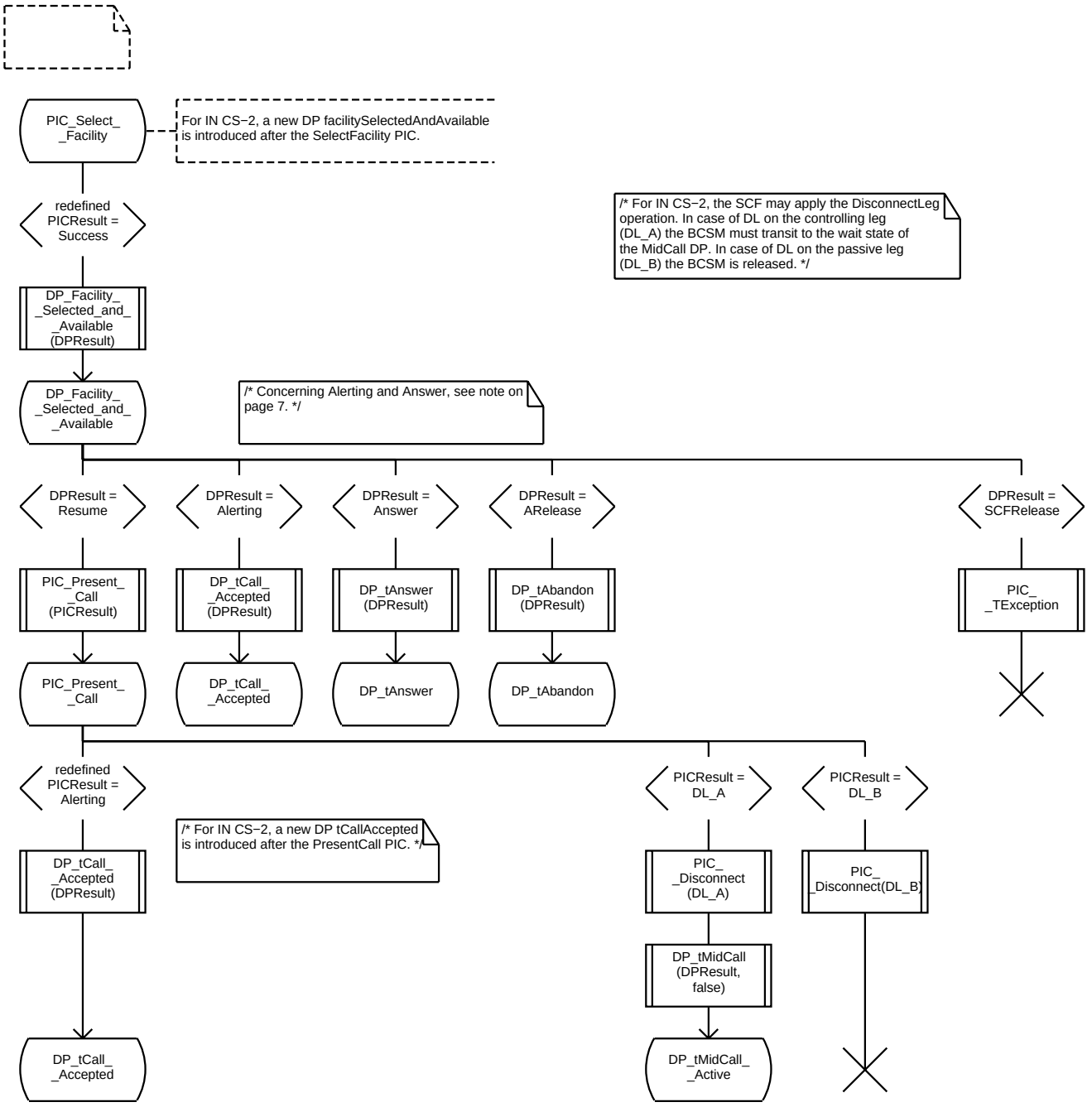
/* Note: The T−BCSM for IN CS−2 only shows the extentions required to the IN CS−1 T−BCSM. Therefore, to understand the complete BCSM behaviour it is necessary to read the two BCSM descriptions together. */

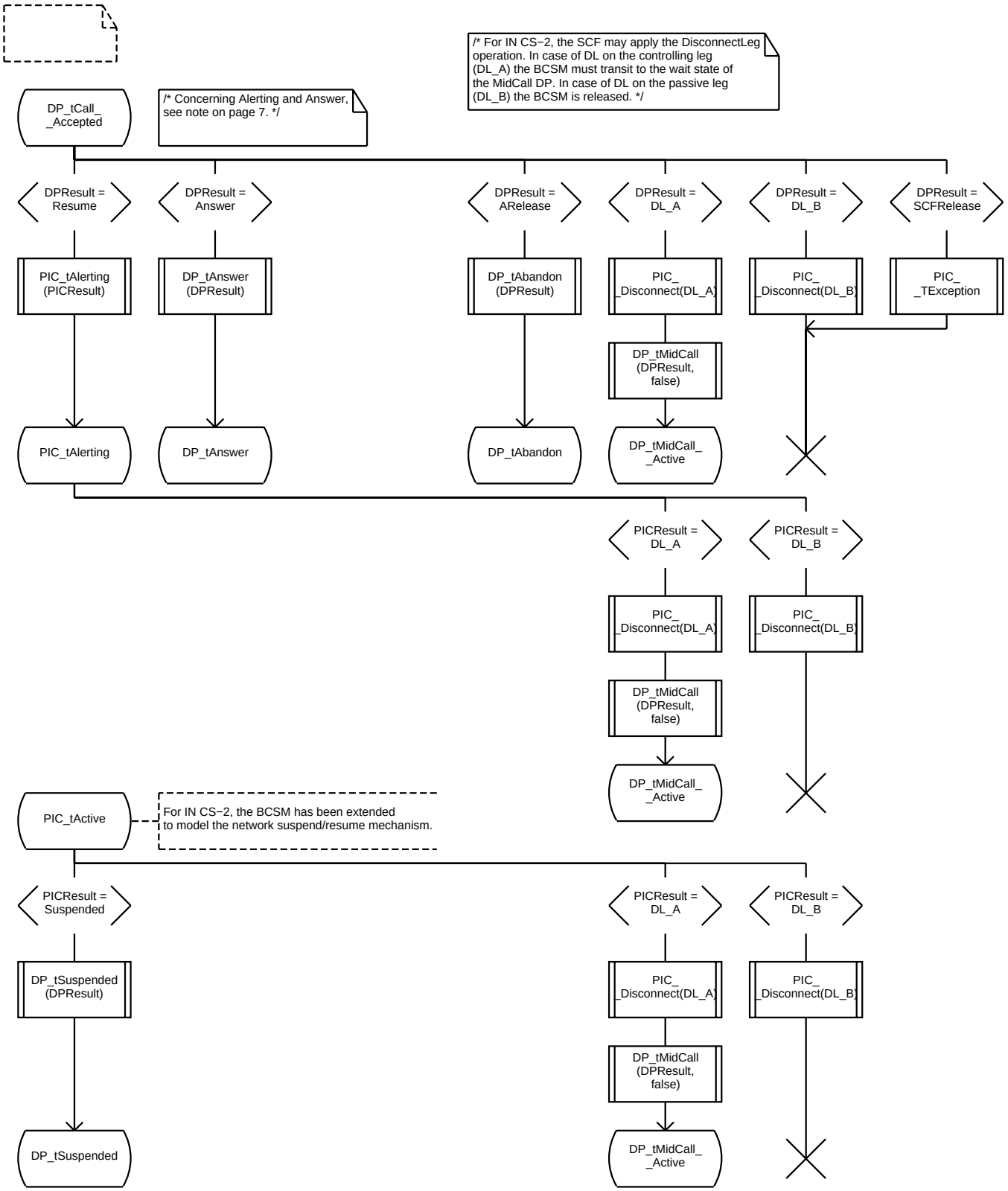
/***** VARIABLE AND TIMER DECLARATIONS. *****/

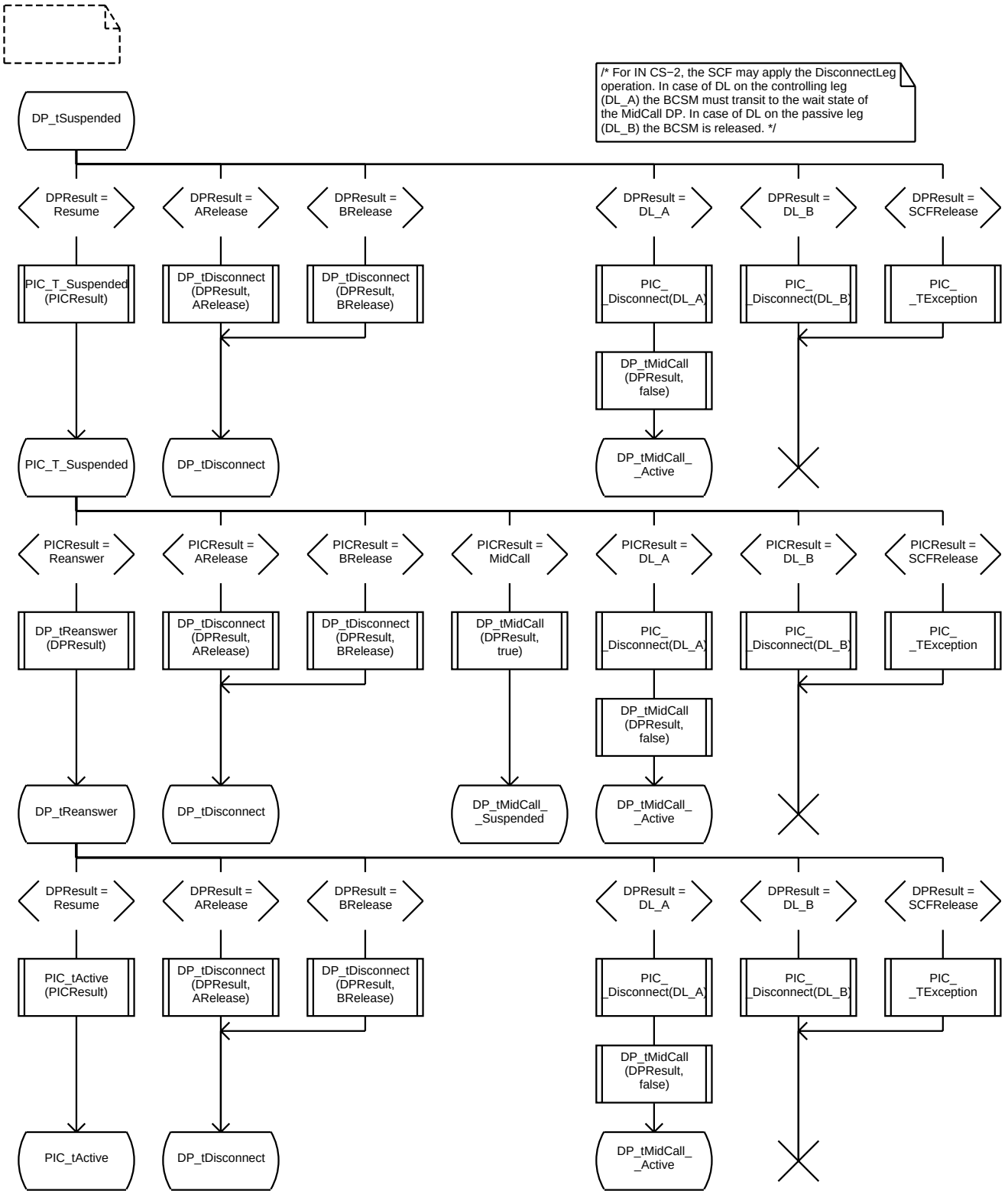
```
DCL
  currentLegID, newLegID LegType,
  udArg UserDataType,
  dpUTSIArg DPUTSIArg,
  nsrArg NetworkSRType;
```

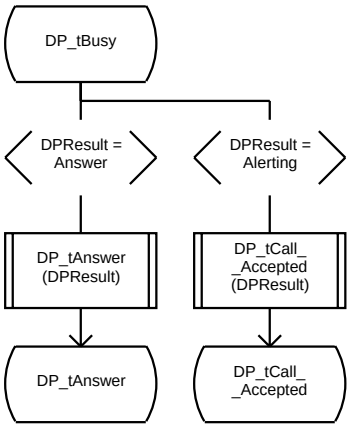

CS
(T_BCSM2_Out)
(T_BCSM2_In)
Redefined Process Type <<System Type CS2_INAP/Block Type SSF_CCF>> TerminatingBCSM
2(8)
/***** Procedure definitions for the new IN CS−2 PICs and the redefined IN CS−1 PICs. *****/
redefined PIC_Present_ _Call
PIC_ _Disconnect
redefined PIC_ _tAlerting
PIC_T_ _Suspended
redefined PIC_tActive
/***** Procedure definitions for the new IN CS−2 DPs and the redefined IN CS−1 DPs. *****/
redefined DP_termAttempt_ Authorized
redefined DP_tMidCall
DP_term_ Attempt
DP_ _tReanswer
redefined DP_tBusy
redefined DP_tAbandon
DP_Facility_ _Selected_and_ _Available
redefined DP_tAnswer
redefined DP_tDisconnect
DP_ _tSuspended
redefined DP_tNoAnswer
DP_tCall_ _Accepted


PIC_T_Null
For IN CS−2, a new DP termAttempt is introduced after the T−Null PIC.
redefined PICResult = success
DP_termAttempt (DPResult)
DP_termAttempt
/* Concerning Alerting and Answer, see note on page 7. */
DPResult = Resume
DPResult = Alerting
DPResult = Answer
DPResult = AResult
DPResult = SCFRelease
PIC_Authorize_ _Termination_ _Attempt (PICResult)
DP_tCall_ _Accepted (DPResult)
DP_tAnswer (DPResult)
DP_tAbandon (DPResult)
PIC_ _TException
PIC_Authorize_ _Termination_ _Attempt
DP_tCall_ _Accepted
DP_tAnswer
DP_tAbandon


PIC_Select_ _Facility
For IN CS−2, a new DP facilitySelectedAndAvailable is introduced after the SelectFacility PIC.
redefined PICResult = Success
/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_Facility_ _Selected_and_ _Available (DPResult)
DP_Facility_ _Selected_and_ _Available
/* Concerning Alerting and Answer, see note on page 7. */
DPResult = Resume
DPResult = Alerting
DPResult = Answer
DPResult = ARelease
DPResult = SCFRelease
PIC_Present_ _Call (PICResult)
DP_tCall_ _Accepted (DPResult)
DP_tAnswer (DPResult)
DP_tAbandon (DPResult)
PIC_ _TException
PIC_Present_ _Call
DP_tCall_ _Accepted
DP_tAnswer
DP_tAbandon
redefined PICResult = Alerting
/* For IN CS−2, a new DP tCallAccepted is introduced after the PresentCall PIC. */
PICResult = DL_A
PICResult = DL_B
DP_tCall_ _Accepted (DPResult)
PIC_ _Disconnect (DL_A)
PIC_ _Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tCall_ _Accepted
DP_tMidCall_ _Active

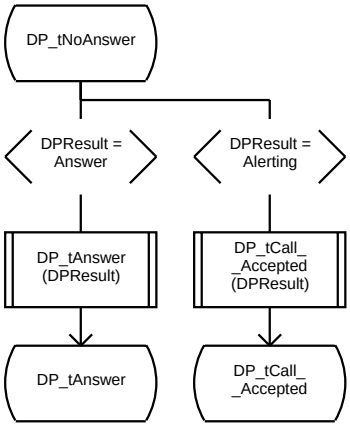

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_tCall_ _Accepted
/* Concerning Alerting and Answer, see note on page 7. */
DPResult = Resume
DPResult = Answer
DPResult = ARelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_tAlerting (PICResult)
DP_tAnswer (DPResult)
DP_tAbandon (DPResult)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
PIC_ _TException
DP_tMidCall (DPResult, false)
PIC_tAlerting
DP_tAnswer
DP_tAbandon
DP_tMidCall_ _Active
PICResult = DL_A
PICResult = DL_B
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tMidCall_ _Active
PIC_tActive
For IN CS−2, the BCSM has been extended to model the network suspend/resume mechanism.
PICResult = Suspended
PICResult = DL_A
PICResult = DL_B
DP_tSuspended (DPResult)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tSuspended
DP_tMidCall_ _Active

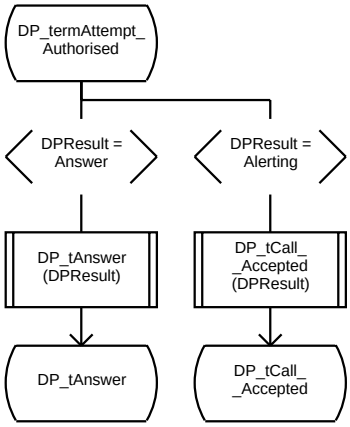

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_tSuspended
DPResult = Resume
DPResult = AResult
DPResult = BRelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_T_Suspended (PICResult)
DP_tDisconnect (DPResult, AResult)
DP_tDisconnect (DPResult, BRelease)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
PIC_ _TException
DP_tMidCall (DPResult, false)
PIC_T_Suspended
DP_tDisconnect
DP_tMidCall_ _Active
PICResult = Reanswer
PICResult = AResult
PICResult = BRelease
PICResult = MidCall
PICResult = DL_A
PICResult = DL_B
PICResult = SCFRelease
DP_tReanswer (DPResult)
DP_tDisconnect (DPResult, AResult)
DP_tDisconnect (DPResult, BRelease)
DP_tMidCall (DPResult, true)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
PIC_ _TException
DP_tMidCall (DPResult, false)
DP_tReanswer
DP_tDisconnect
DP_tMidCall_ _Suspended
DP_tMidCall_ _Active
DPResult = Resume
DPResult = AResult
DPResult = BRelease
DPResult = DL_A
DPResult = DL_B
DPResult = SCFRelease
PIC_tActive (PICResult)
DP_tDisconnect (DPResult, AResult)
DP_tDisconnect (DPResult, BRelease)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
PIC_ _TException
DP_tMidCall (DPResult, false)
PIC_tActive
DP_tDisconnect
DP_tMidCall_ _Active

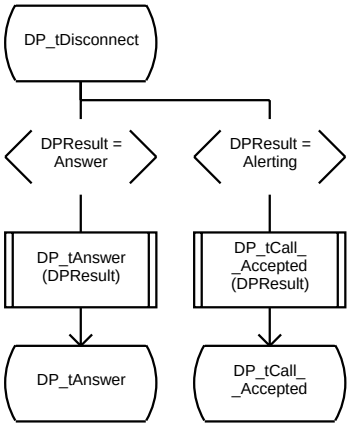


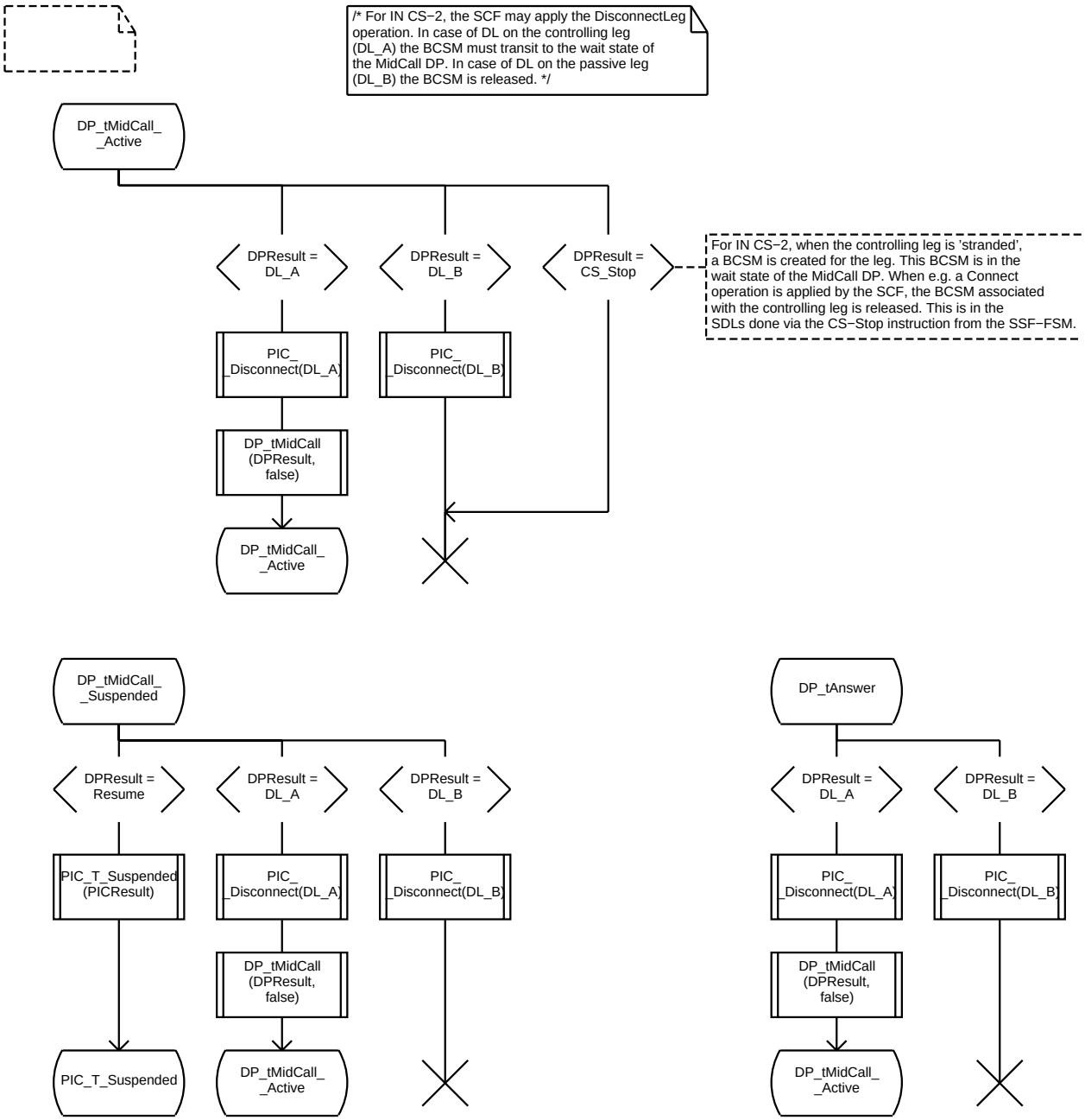
/* For IN CS−2, the SCF may perform a Connect in the CS state Terminating−Setup to forward the call. When in CS state Terminating−Setup, the T−BCSM is suspended at one of the following DPs:
termAttempt
termAttemptAuthorized
facilitySelectedAndAvailable
tNoAnswer
tBusy
tDisconnect

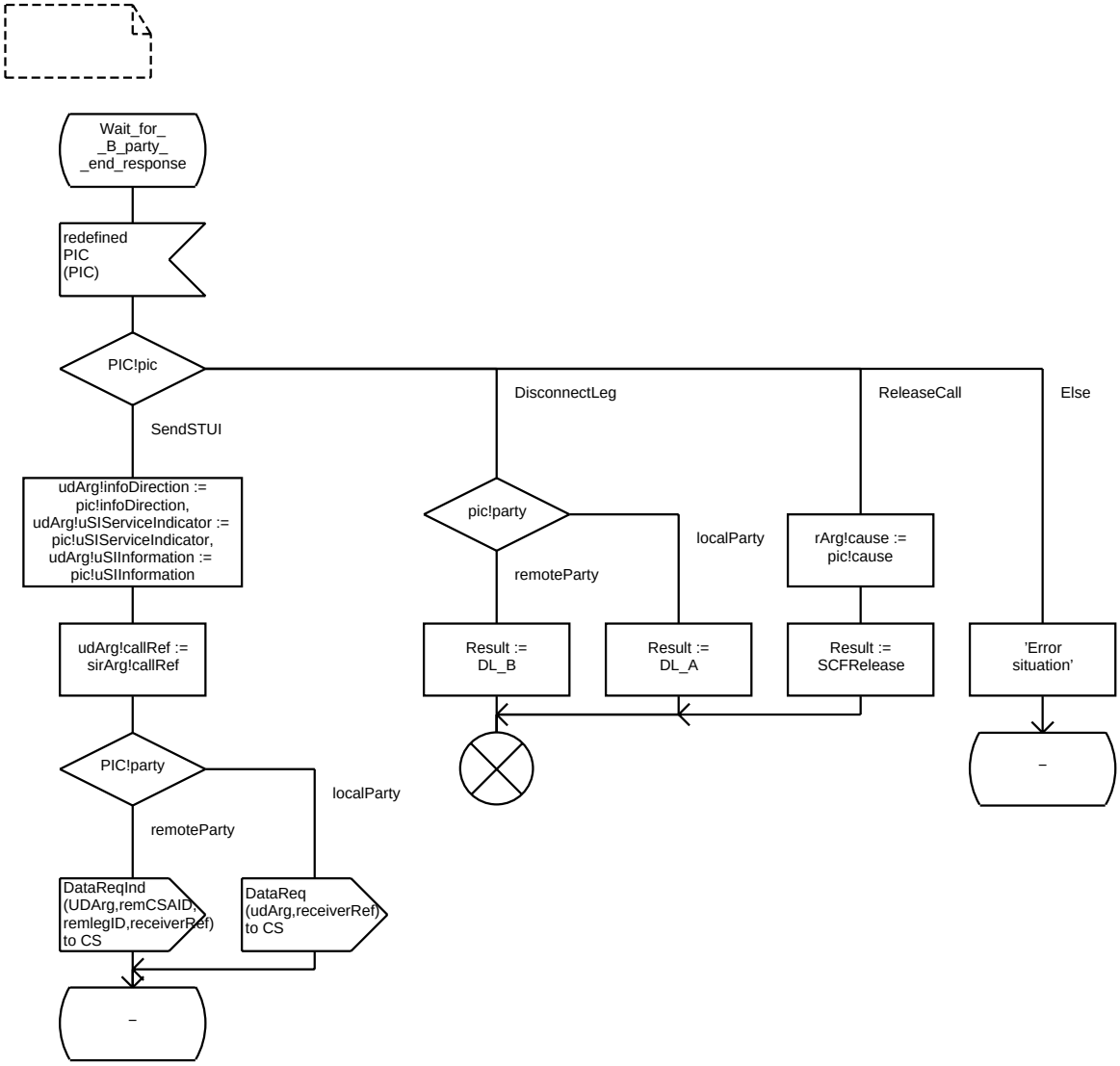
When the Callprogress(bptyAlerted) or the SetupResp (answer) comes from the C−party, the T−BCSM has to transit to the corresponding DP. */

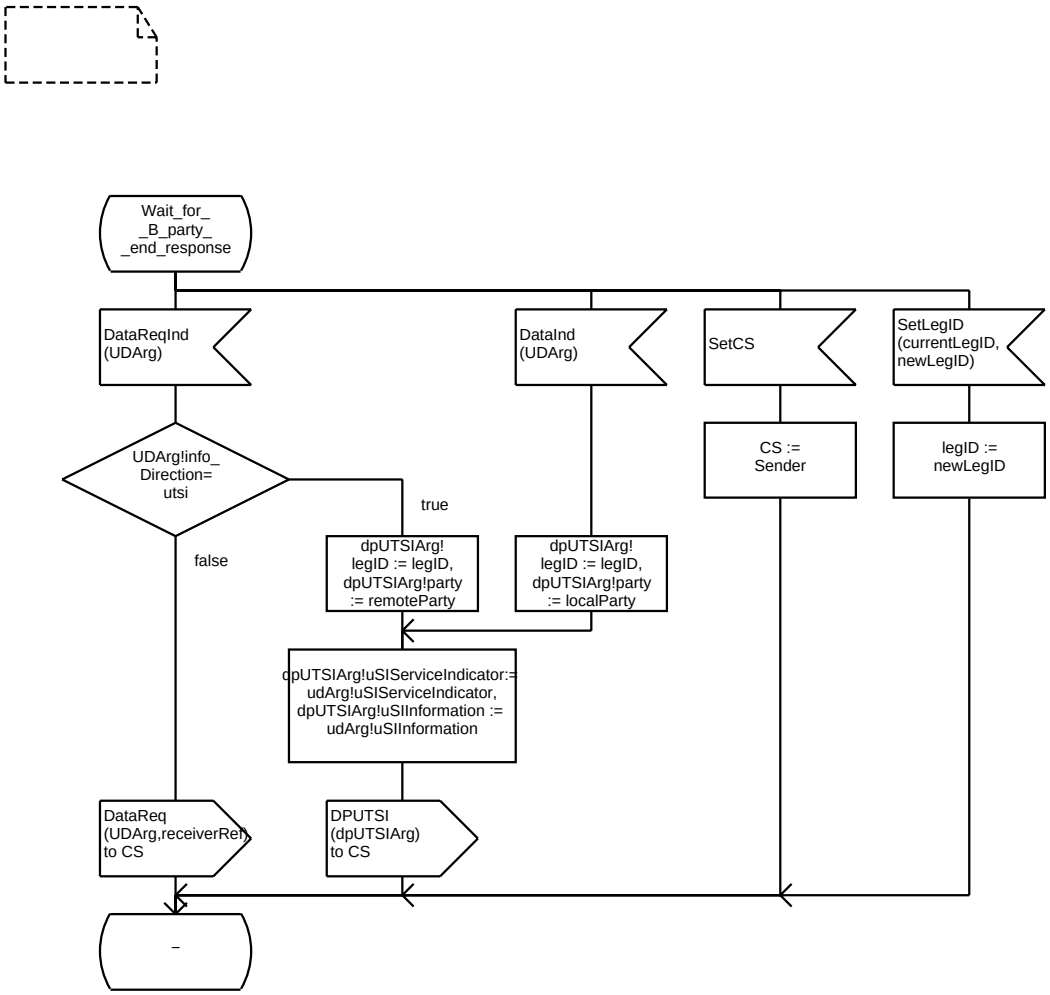




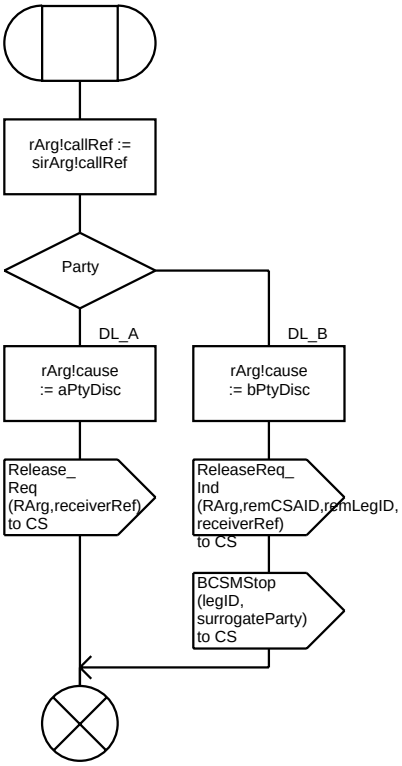


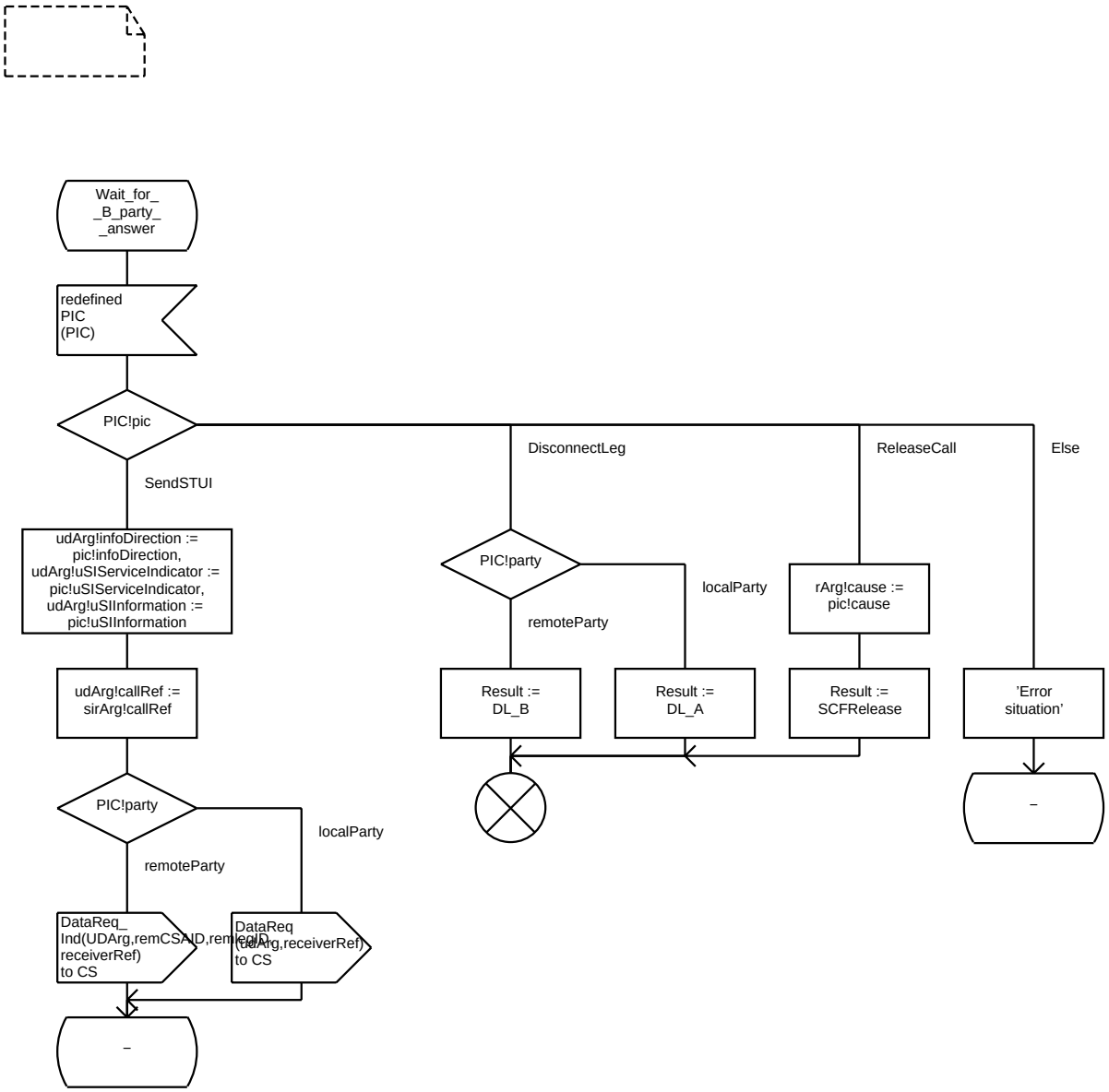

/* For IN CS−2, the SCF may apply the DisconnectLeg operation. In case of DL on the controlling leg (DL_A) the BCSM must transit to the wait state of the MidCall DP. In case of DL on the passive leg (DL_B) the BCSM is released. */
DP_tMidCall_ _Active
DPResult = DL_A
DPResult = DL_B
DPResult = CS_Stop
For IN CS−2, when the controlling leg is 'stranded', a BCSM is created for the leg. This BCSM is in the wait state of the MidCall DP. When e.g. a Connect operation is applied by the SCF, the BCSM associated with the controlling leg is released. This is in the SDLs done via the CS−Stop instruction from the SSF−FSM.
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tMidCall_ _Active
DP_tMidCall_ _Suspended
DPResult = Resume
DPResult = DL_A
DPResult = DL_B
PIC_T_Suspended (PICResult)
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
DP_tMidCall (DPResult, false)
PIC_T_Suspended
DP_tMidCall_ _Active
DP_tAnswer
DPResult = DL_A
DPResult = DL_B
PIC_ _Disconnect(DL_A)
PIC_ _Disconnect(DL_B)
DP_tMidCall (DPResult, false)
DP_tMidCall_ _Active

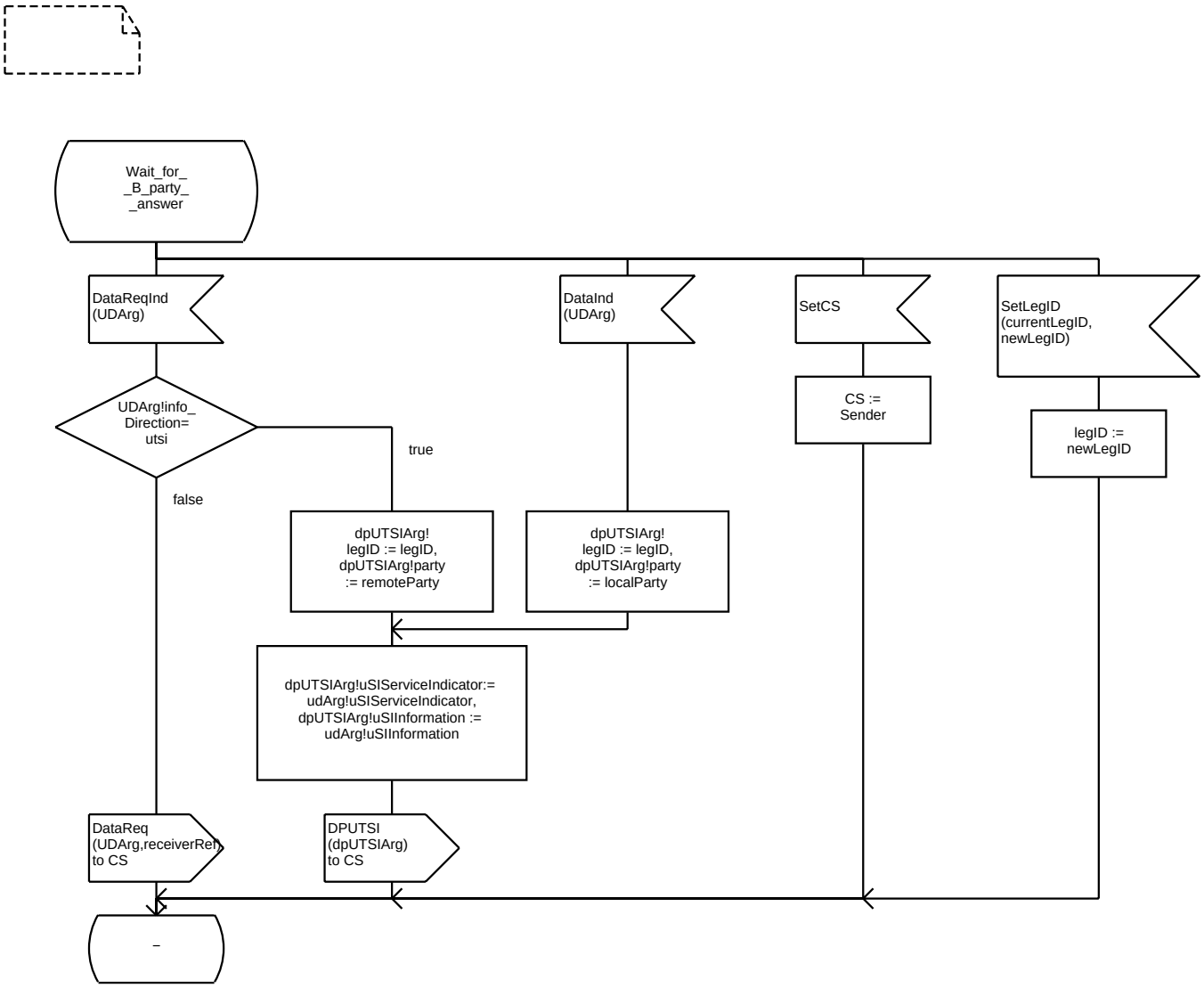

Wait_for_
_B_party_
_end_response
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
pic!party
localParty
remoteParty
rArg!cause :=
pic!cause
udArg!callRef :=
sirArg!callRef
Result :=
DL_B
Result :=
DL_A
Result :=
SCFRelease
'Error
situation'
PIC!party
localParty
remoteParty
DataReqInd
(UDArg,remCSAID,
remlegID,receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS
–
–

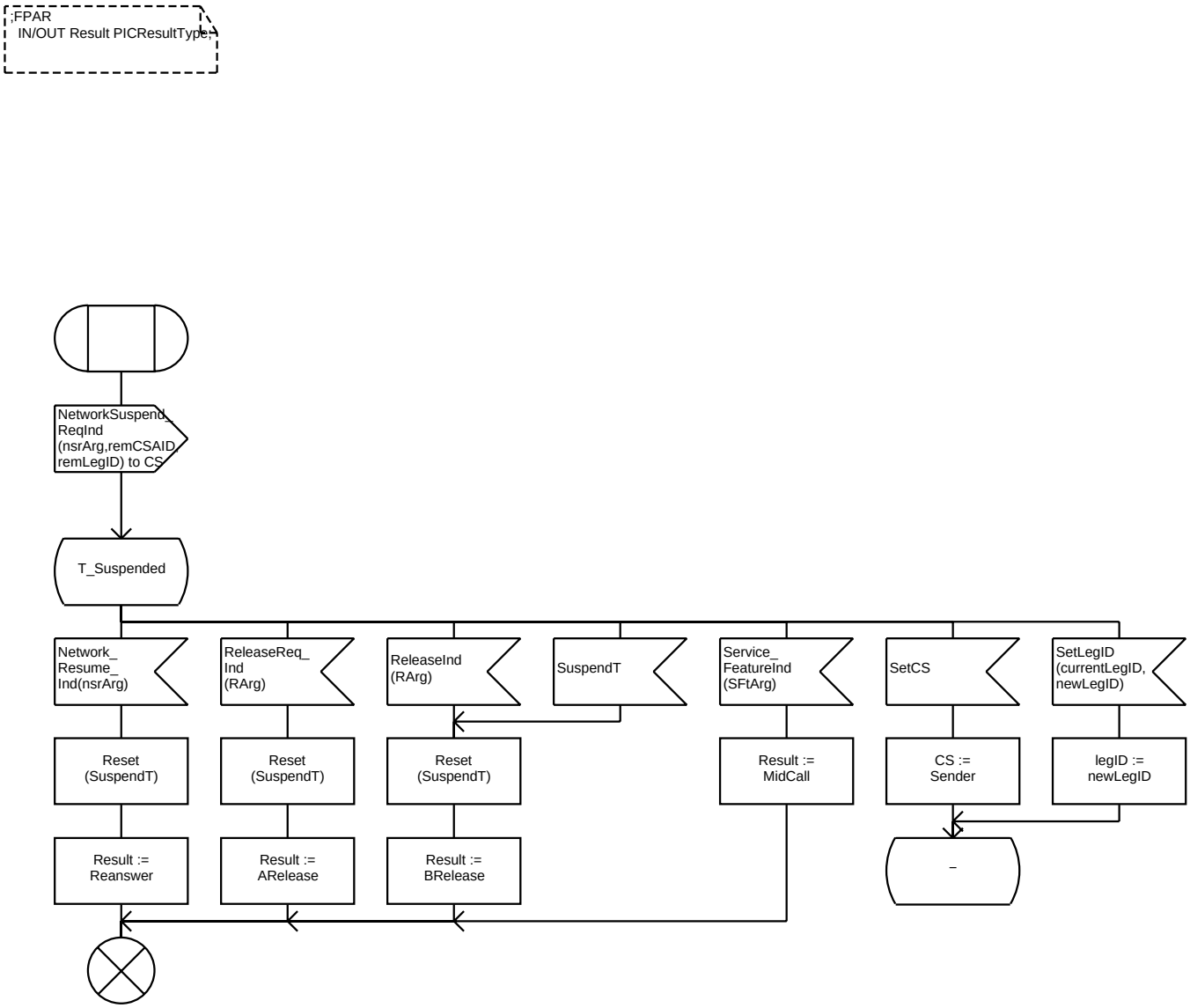

Wait_for_
_B_party_
_end_response
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

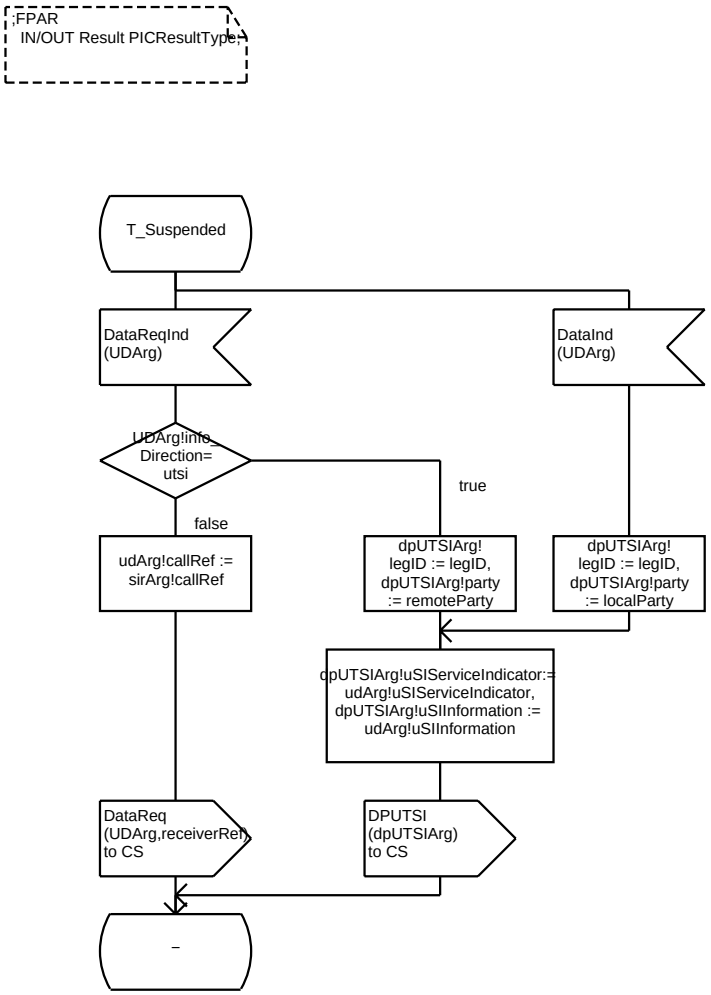
;FPAR
IN Party DPResultType;

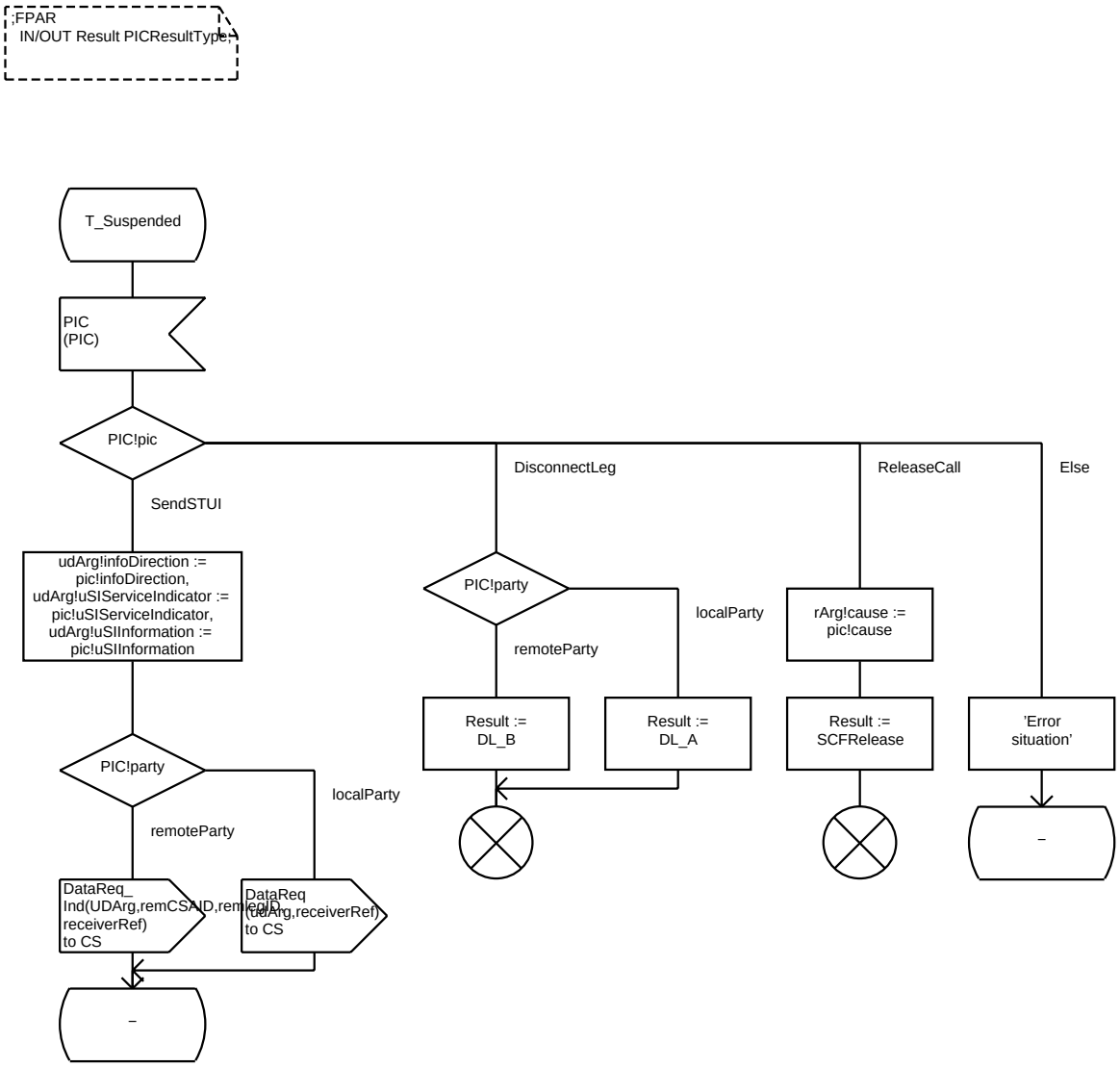


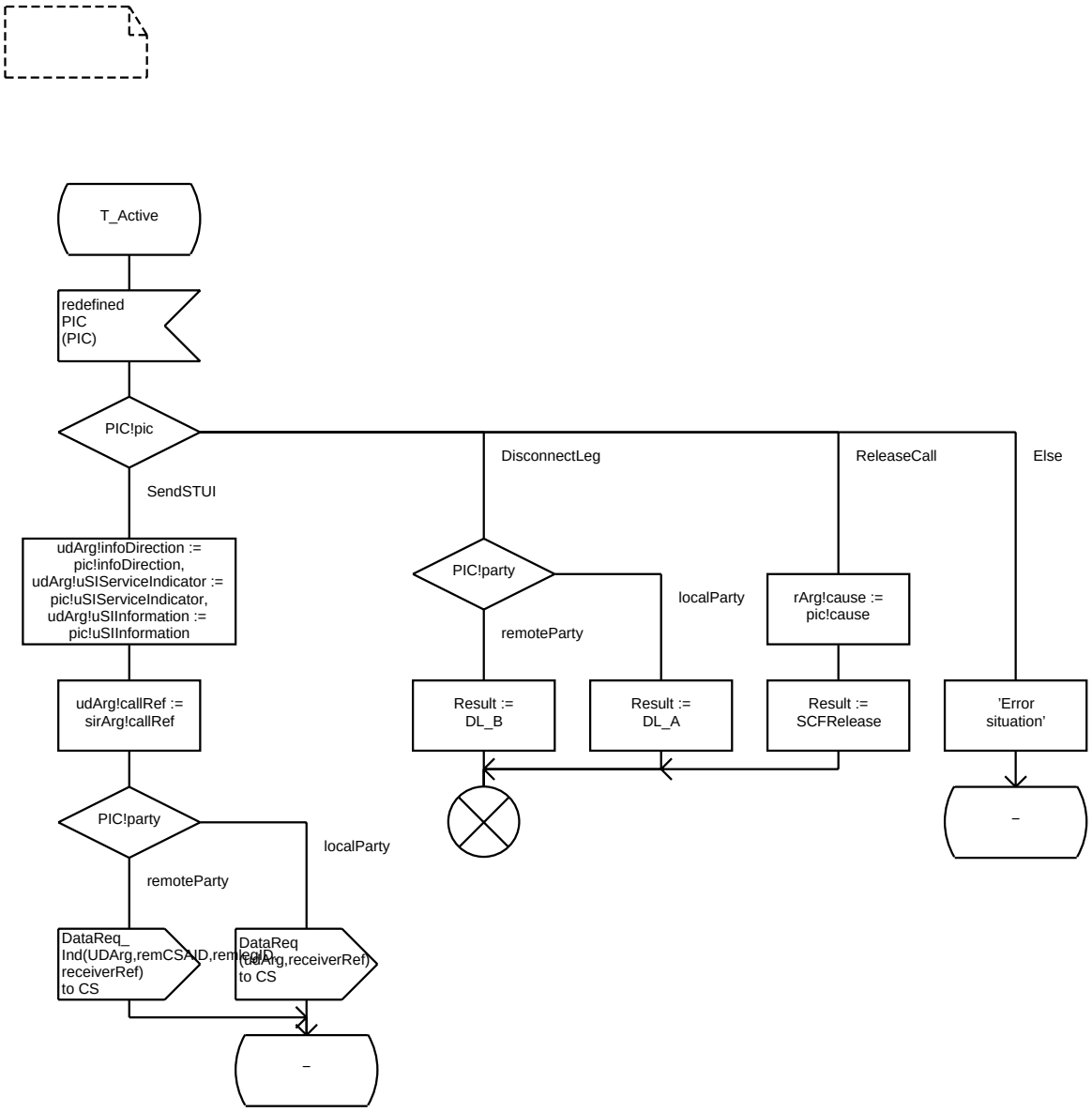

Wait_for_
_B_party_
_answer
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
udArg!callRef :=
sirArg!callRef
Result :=
DL_B
Result :=
DL_A
Result :=
SCFRelease
'Error
situation'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remLegID,
receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS

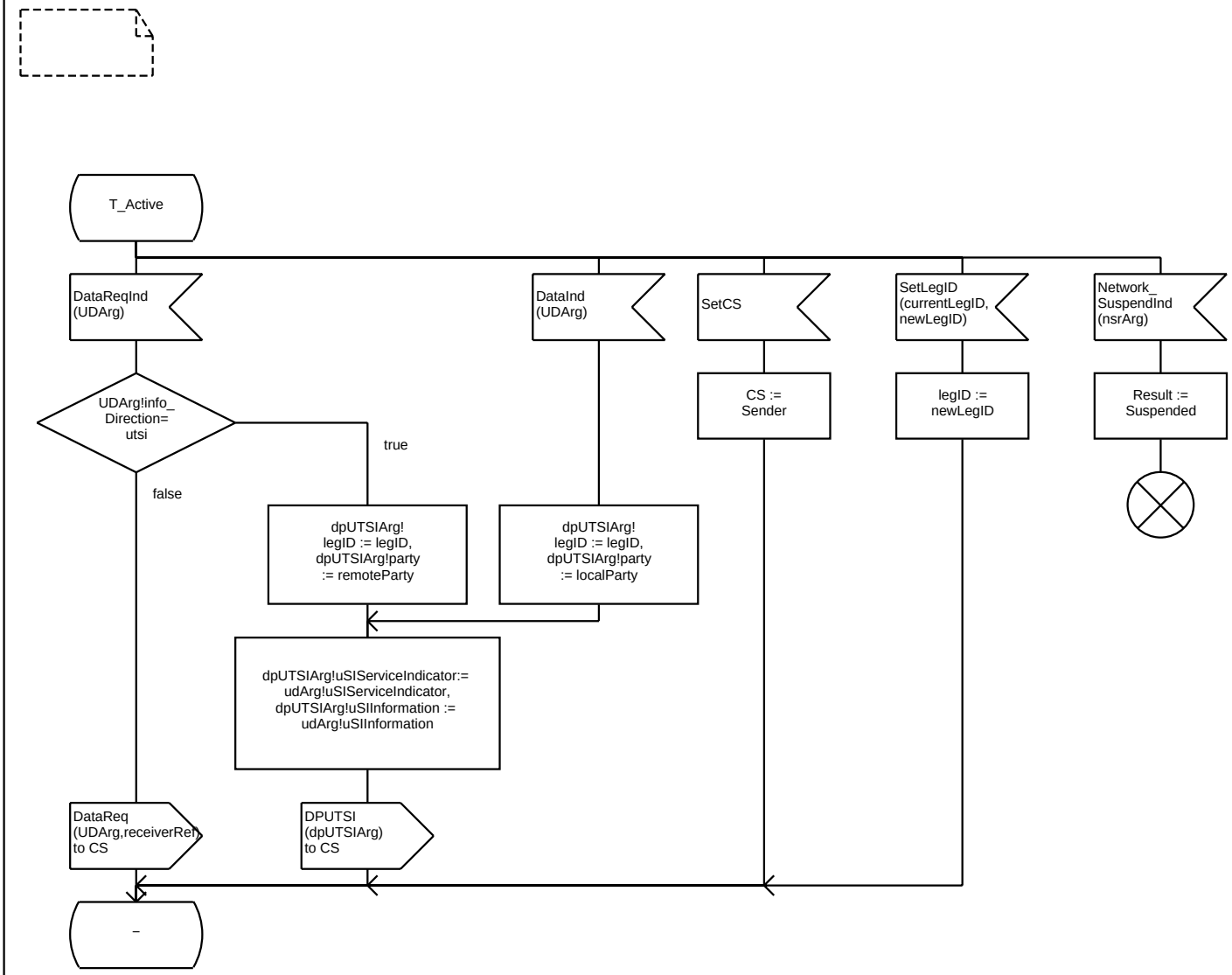

Wait_for_
_B_party_
_answer
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

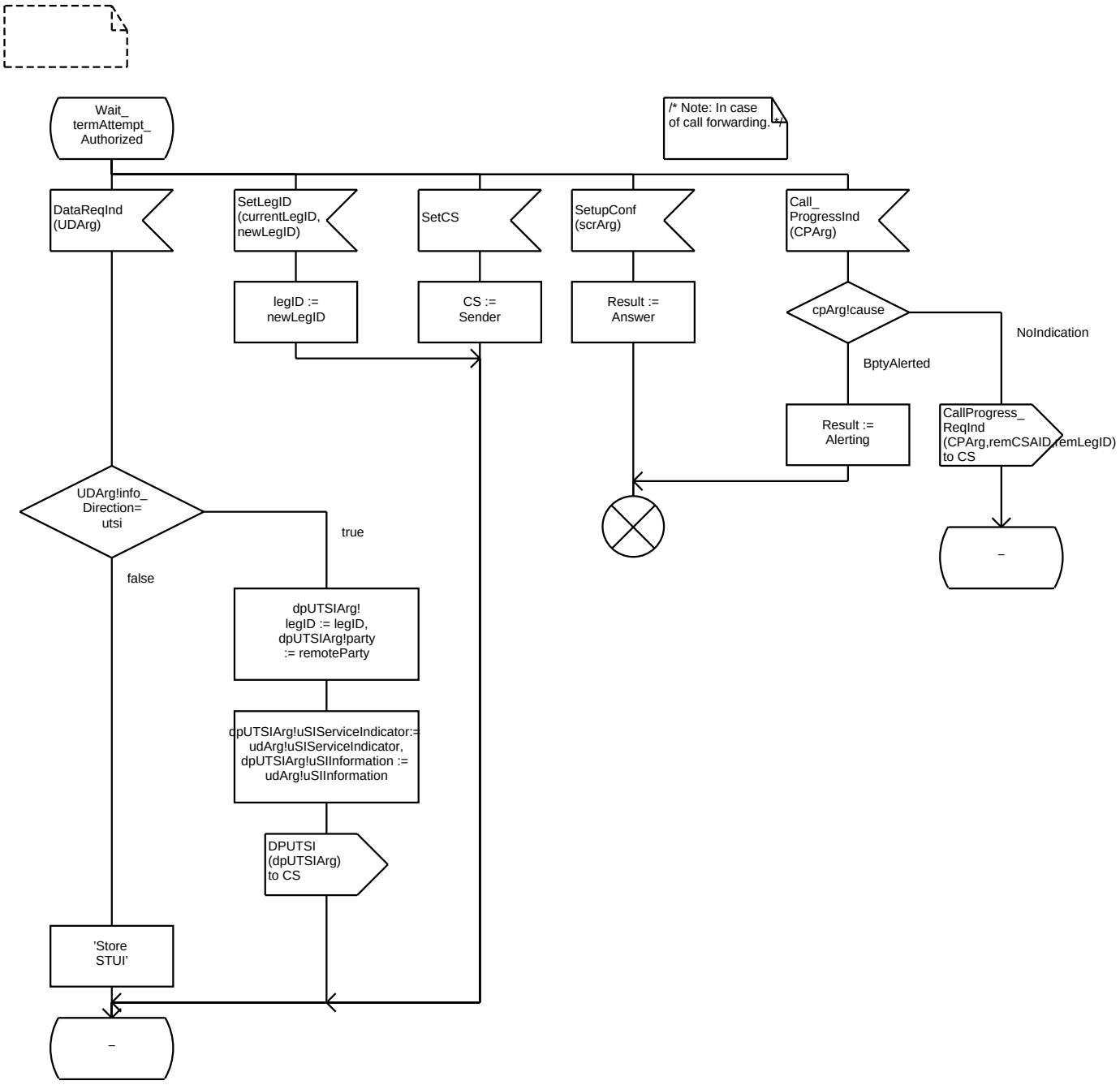

;FPAR
IN/OUT Result PICResultType;
NetworkSuspend_ ReqInd (nsrArg,remCSAID, remLegID) to CS
T_Suspended
Network_ Resume_ Ind(nsrArg)
ReleaseReq_ Ind (RArg)
ReleaseInd (RArg)
SuspendT
Service_ FeatureInd (SFtArg)
SetCS
SetLegID (currentLegID, newLegID)
Reset (SuspendT)
Reset (SuspendT)
Reset (SuspendT)
Result := MidCall
CS := Sender
legID := newLegID
Result := Reanswer
Result := ARelease
Result := BRelease
–

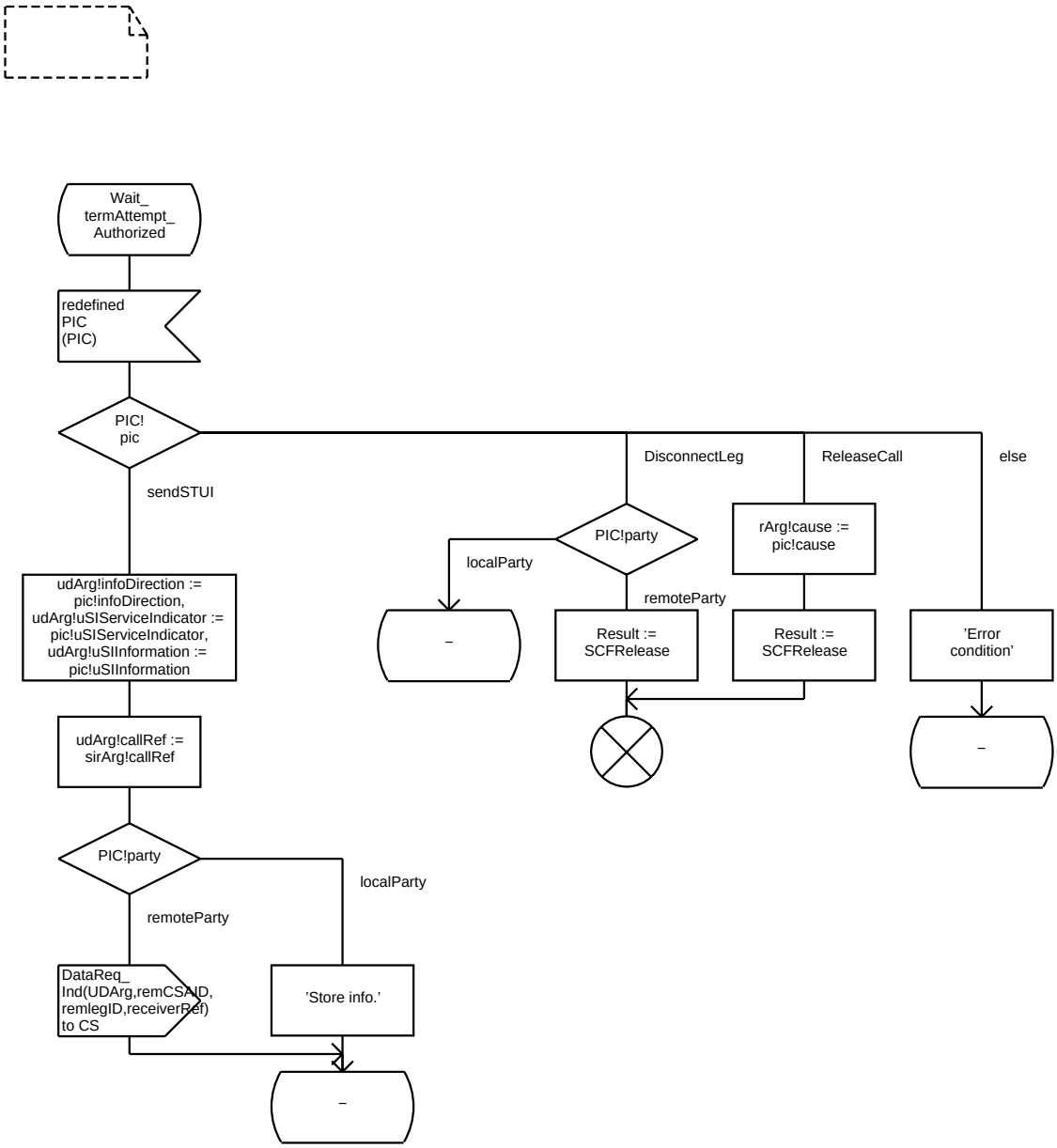

;FPAR
IN/OUT Result PICResultType;
T_Suspended
DataReqInd
(UDArg)
DataInd
(UDArg)
UDArg!info_
Direction=
utsi
true
false
udArg!callRef :=
sirArg!callRef
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

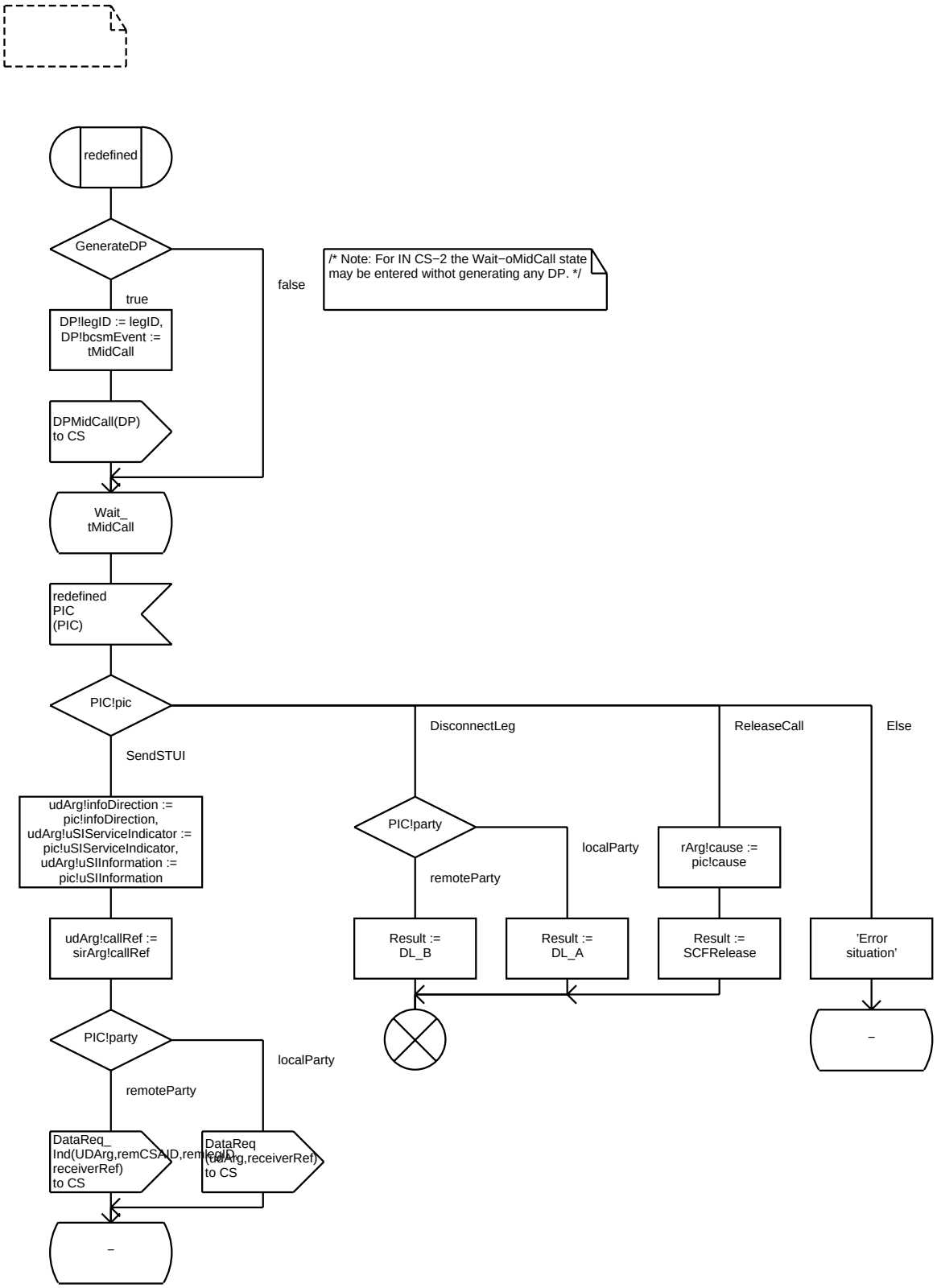

;FPAR
IN/OUT Result PICResultType;
T_Suspended
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
Result :=
DL_B
Result :=
DL_A
Result :=
SCFRelease
'Error
situation'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remLegID,
receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS
–
–

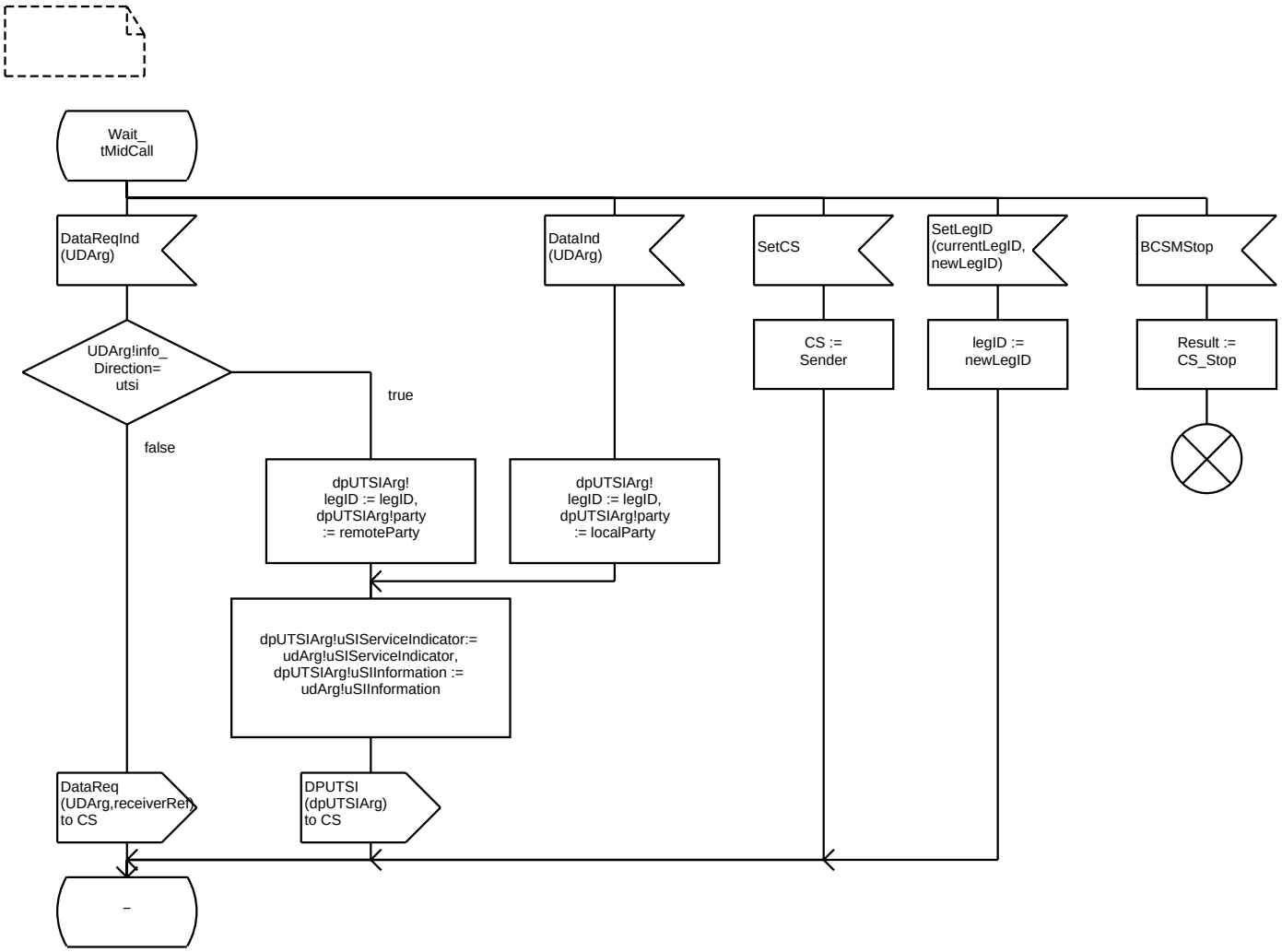

T_Active
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remLegID,
receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS
–
PIC!party
localParty
remoteParty
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
Result :=
SCFRelease
'Error
situation'
–

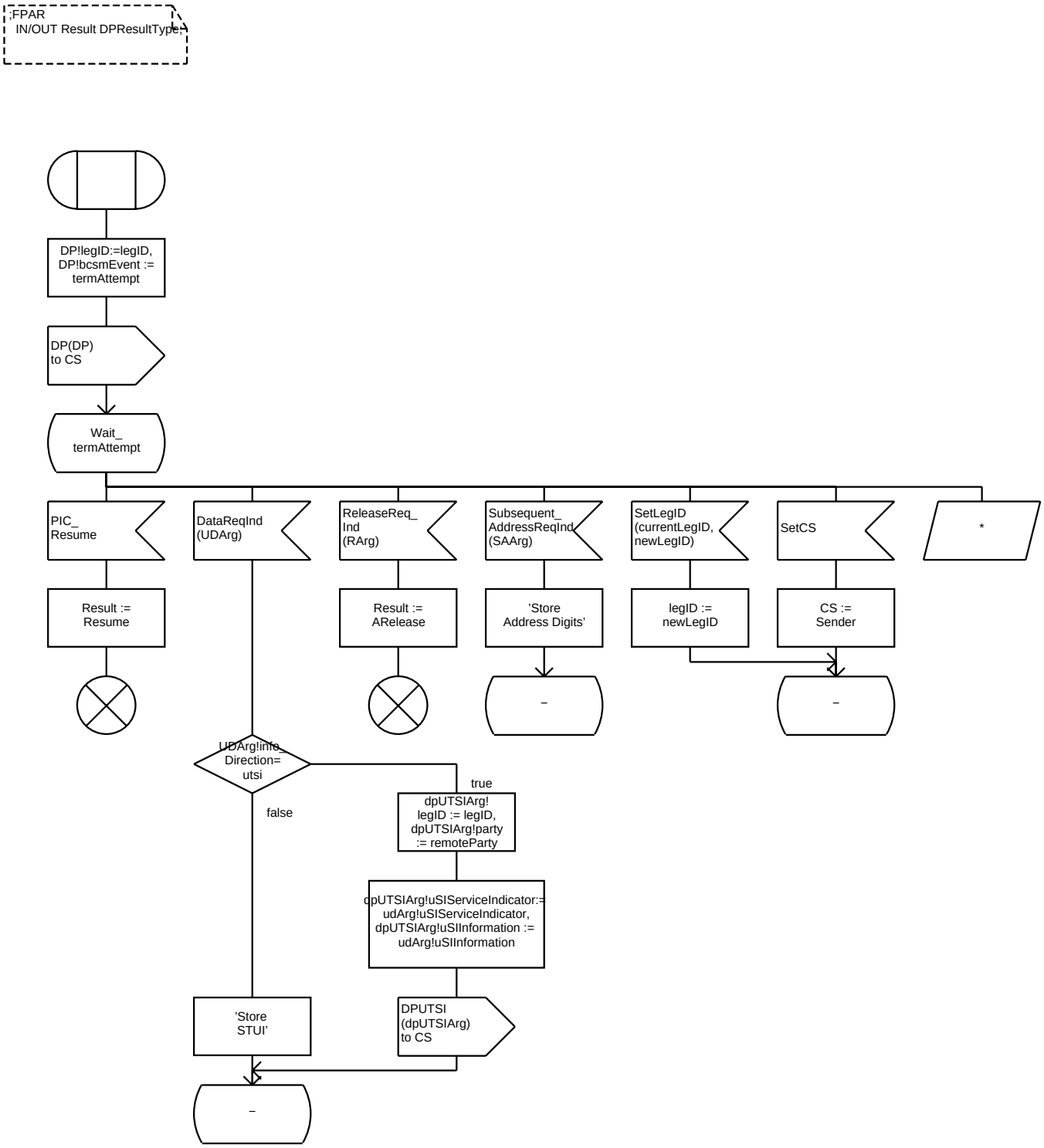

T_Active
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
Network_
SuspendInd
(nsrArg)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
Result :=
Suspended
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–


Wait_ termAttempt_ Authorized
/* Note: In case of call forwarding. */
DataReqInd (UDArg)
SetLegID (currentLegID, newLegID)
SetCS
SetupConf (scrArg)
Call_ ProgressInd (CPArg)
legID := newLegID
CS := Sender
Result := Answer
cpArg!cause
NoIndication
BptyAlerted
Result := Alerting
CallProgress_ ReqInd (CPArg,remCSAID,remLegID) to CS
UDArg!info_ Direction= utsi
true
false
dpUTSIArg! legID := legID, dpUTSIArg!party := remoteParty
dpUTSIArg!uSIServiceIndicator:= udArg!uSIServiceIndicator, dpUTSIArg!uSIInformation := udArg!uSIInformation
DPUTSI (dpUTSIArg) to CS
'Store STUI'
–
–

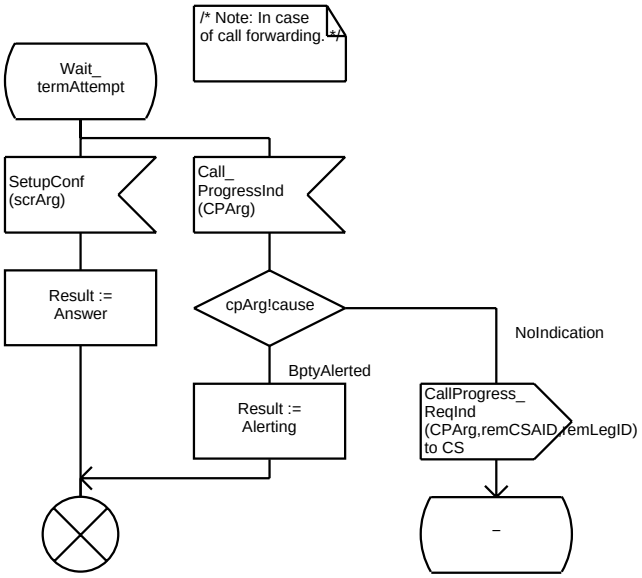

Wait_
termAttempt_
Authorized
redefined
PIC
(PIC)
PIC!
pic
sendSTUI
DisconnectLeg
ReleaseCall
else
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
–
Result :=
SCFRelease
Result :=
SCFRelease
'Error
condition'
–
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,
remlegID,receiverRef)
to CS
'Store info.'
–

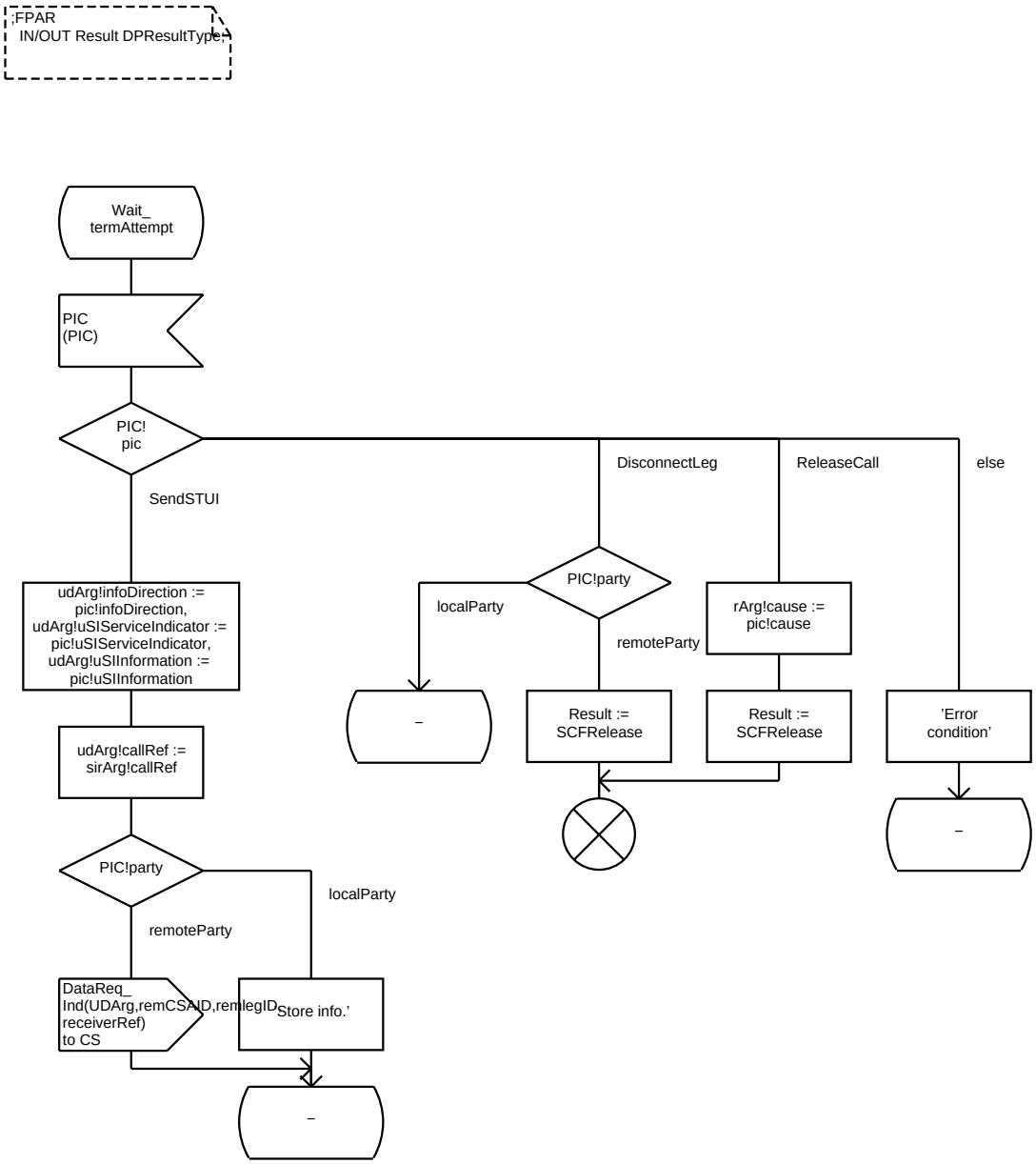

redefined
GenerateDP
false
/* Note: For IN CS−2 the Wait−oMidCall state may be entered withot generating any DP. */
true
DP!legID := legID, DP!bcsmEvent := tMidCall
DPMidCall(DP) to CS
Wait_ tMidCall
redefined PIC (PIC)
PIC!pic
DisconnectLeg
ReleaseCall
Else
SendSTUI
udArg!infoDirection := pic!infoDirection, udArg!uSIServiceIndicator := pic!uSIServiceIndicator, udArg!uSIInformation := pic!uSIInformation
PIC!party
localParty
remoteParty
rArg!cause := pic!cause
udArg!callRef := sirArg!callRef
Result := DL_B
Result := DL_A
Result := SCFRelease
'Error situation'
PIC!party
localParty
remoteParty
DataReq_ Ind(UDArg,remCSAID,remLegID, receiverRef) to CS
DataReq (udArg,receiverRef) to CS
–
–

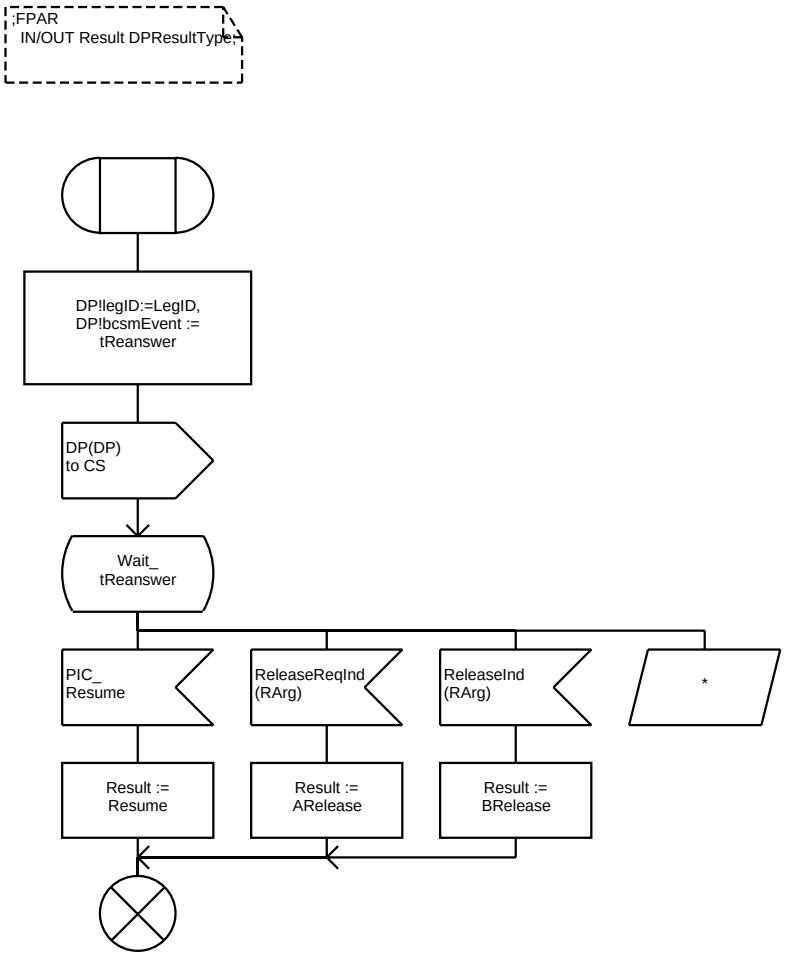

Wait_
tMidCall
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
BCSMStop
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
Result :=
CS_Stop
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

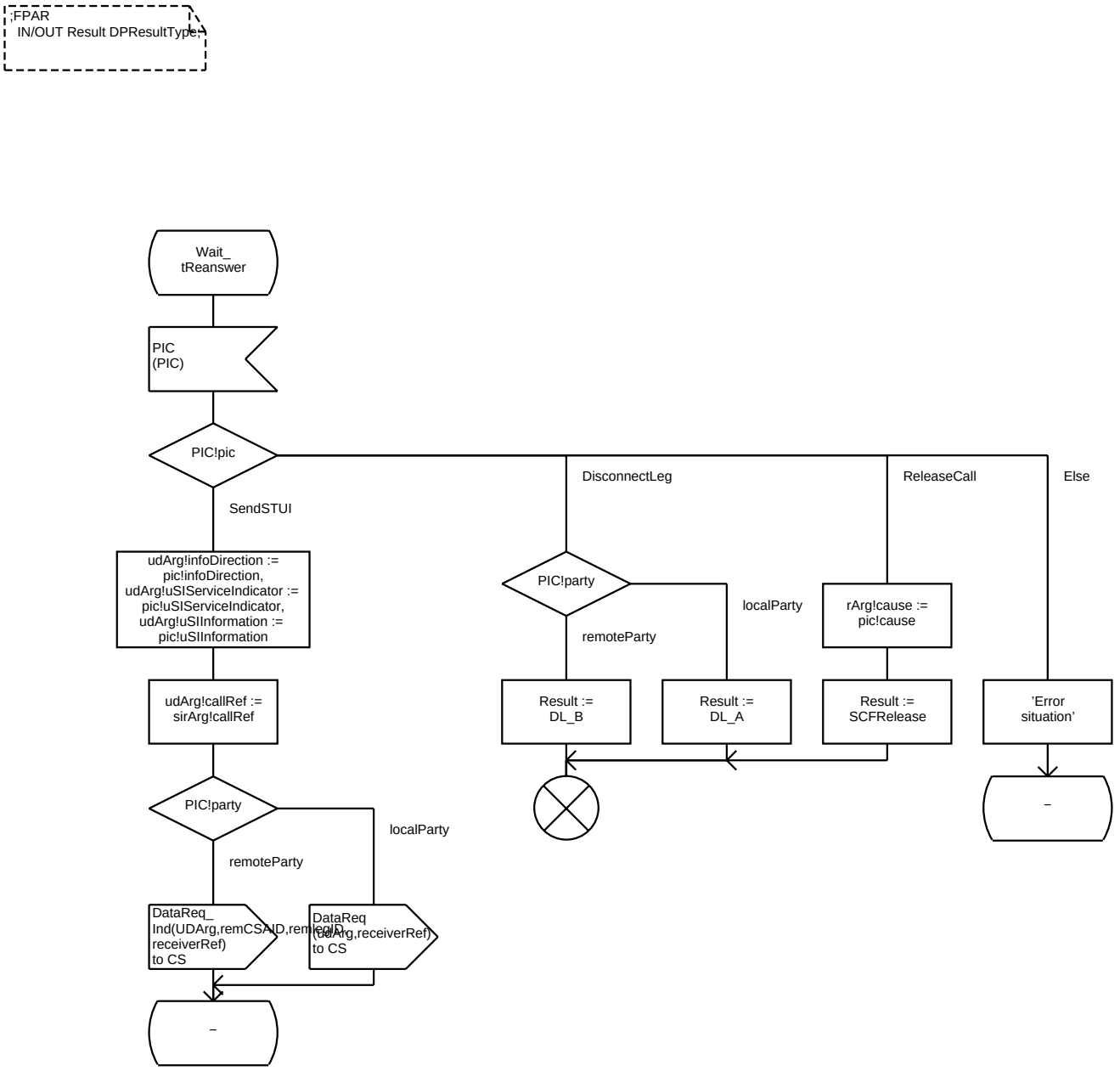

;FPAR
IN/OUT Result DPResultType;
DP!legID:=legID,
DP!bcsmEvent :=
termAttempt
DP(DP)
to CS
Wait_
termAttempt
PIC_
Resume
DataReqInd
(UDArg)
ReleaseReq_
Ind
(RArg)
Subsequent_
AddressReqInd
(SAArg)
SetLegID
(currentLegID,
newLegID)
SetCS
*
Result :=
Resume
Result :=
ARelease
'Store
Address Digits'
legID :=
newLegID
CS :=
Sender
–
–
UDArg!info_
Direction=
utsi
true
false
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
'Store
STUI'
DPUTSI
(dpUTSIArg)
to CS
–

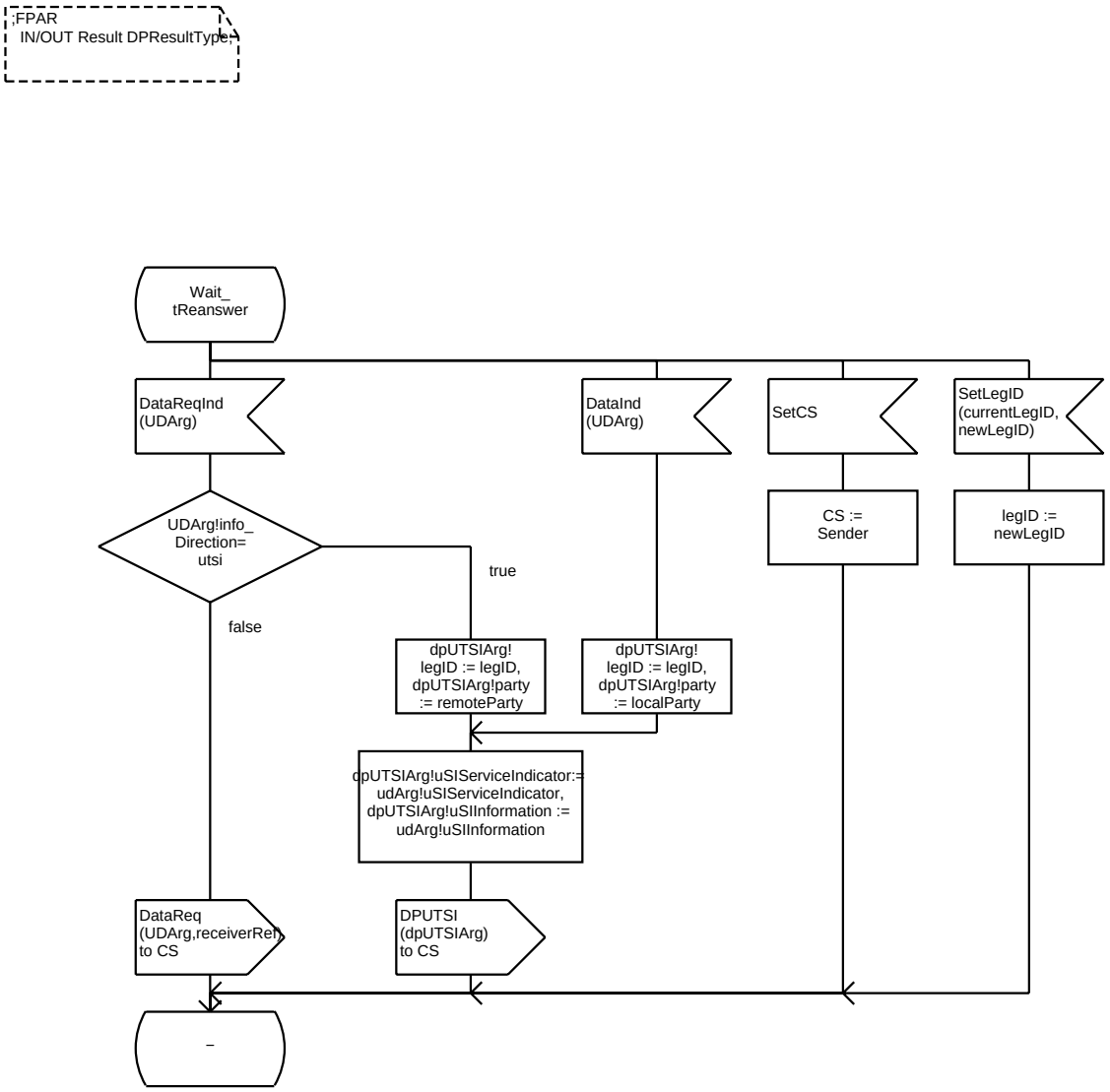
;FPAR
IN/OUT Result DPResultType;

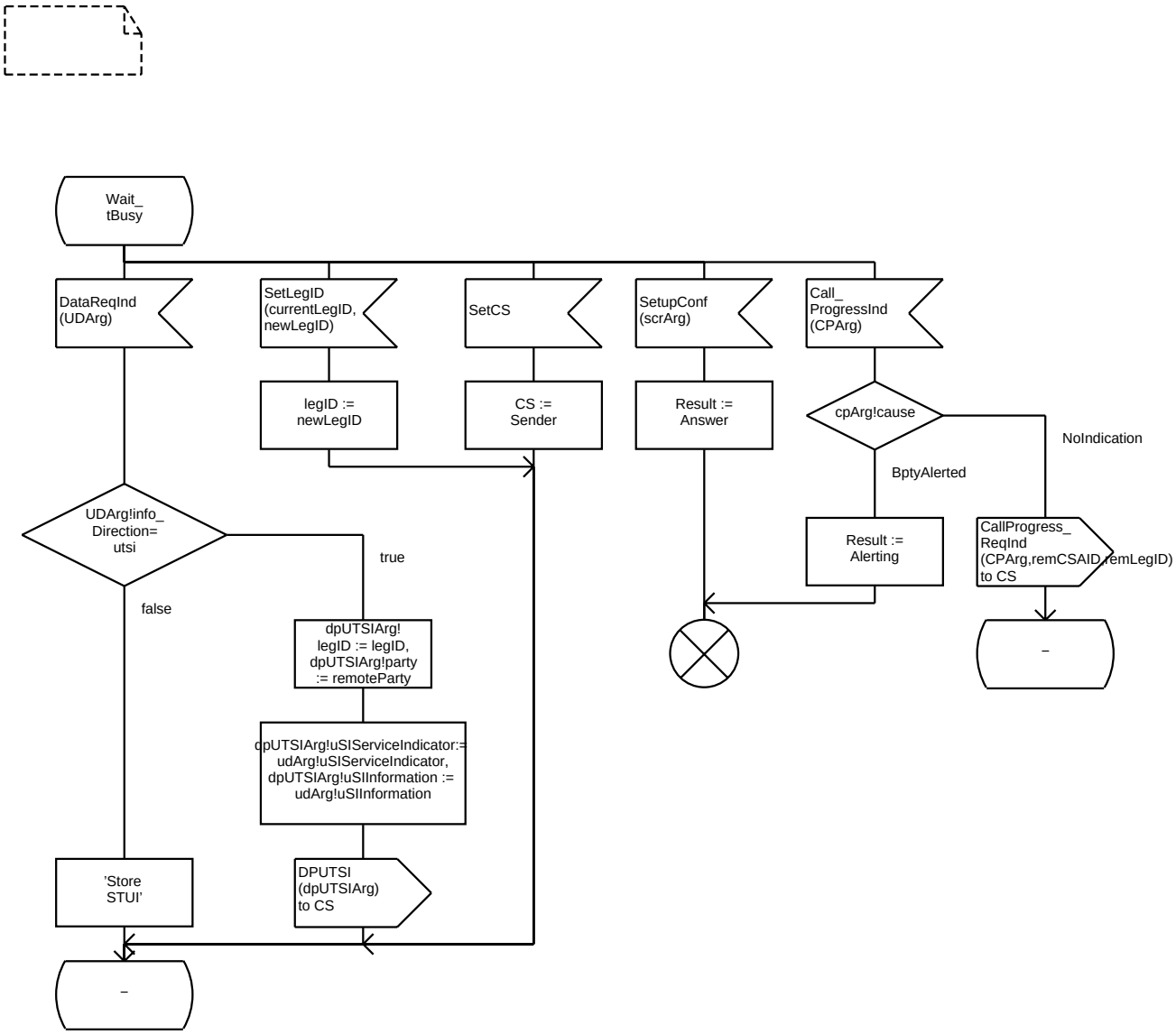


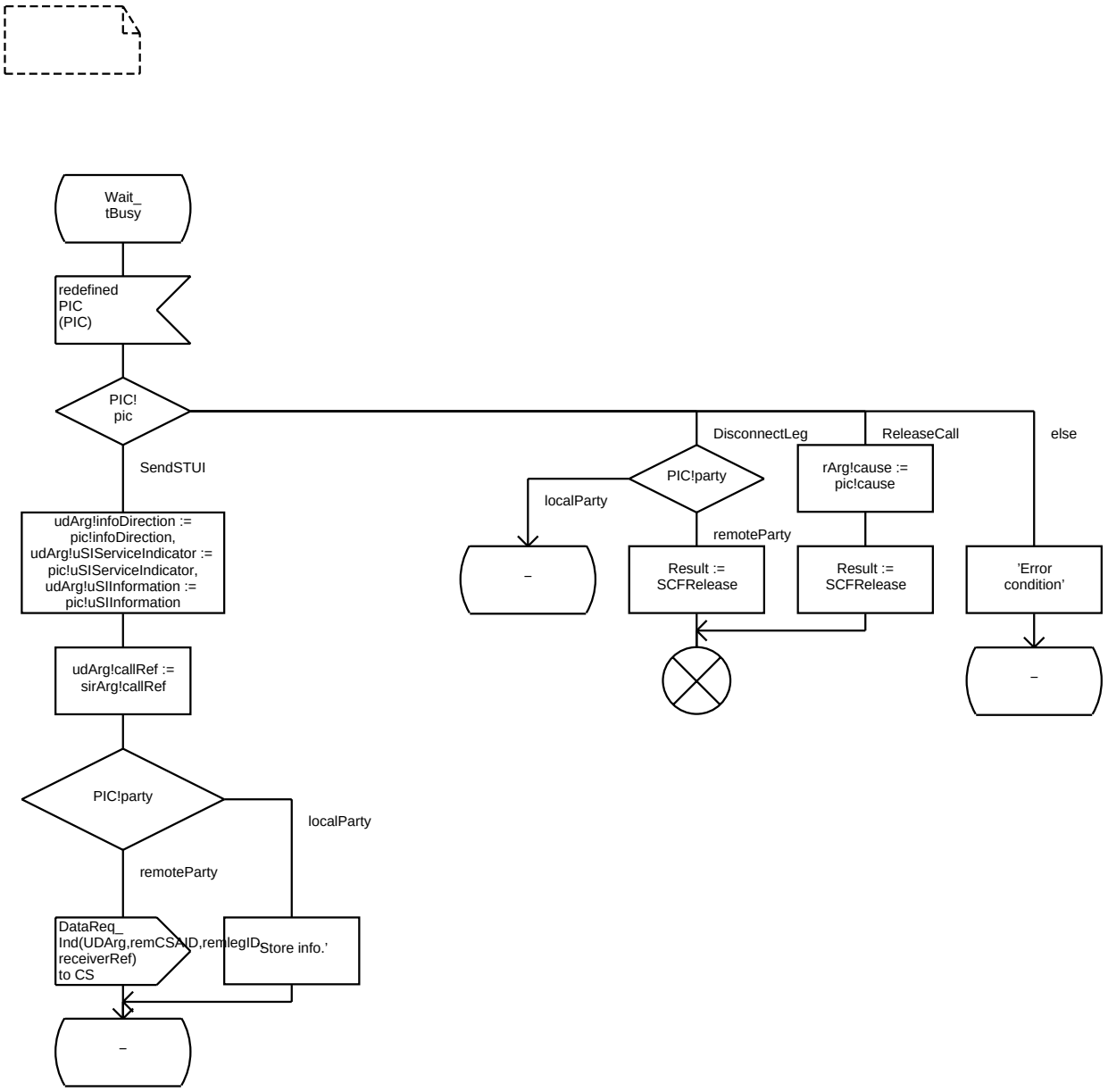

;FPAR
IN/OUT Result DPResultType;
Wait_
termAttempt
PIC
(PIC)
PIC!
pic
SendSTUI
DisconnectLeg
ReleaseCall
else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
–
Result :=
SCFRelease
Result :=
SCFRelease
'Error
condition'
–
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remlegID,
receiverRef)
to CS
Store info.'
–

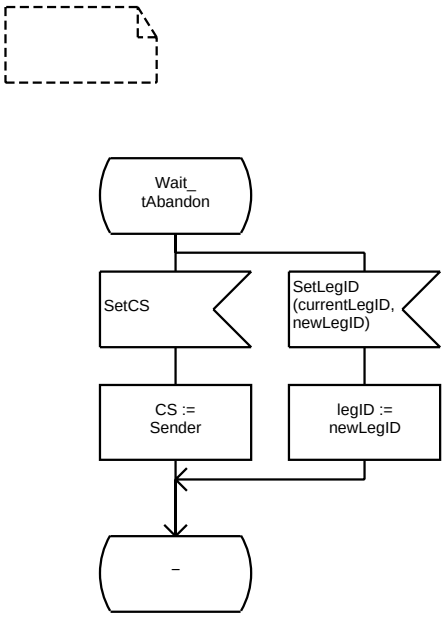

;FPAR
IN/OUT Result DPResultType;
DP!legID:=LegID,
DP!bcsmEvent :=
tReanswer
DP(DP)
to CS
Wait_
tReanswer
PIC_
Resume
ReleaseReqInd
(RArg)
ReleaseInd
(RArg)
*
Result :=
Resume
Result :=
ARelease
Result :=
BRelease

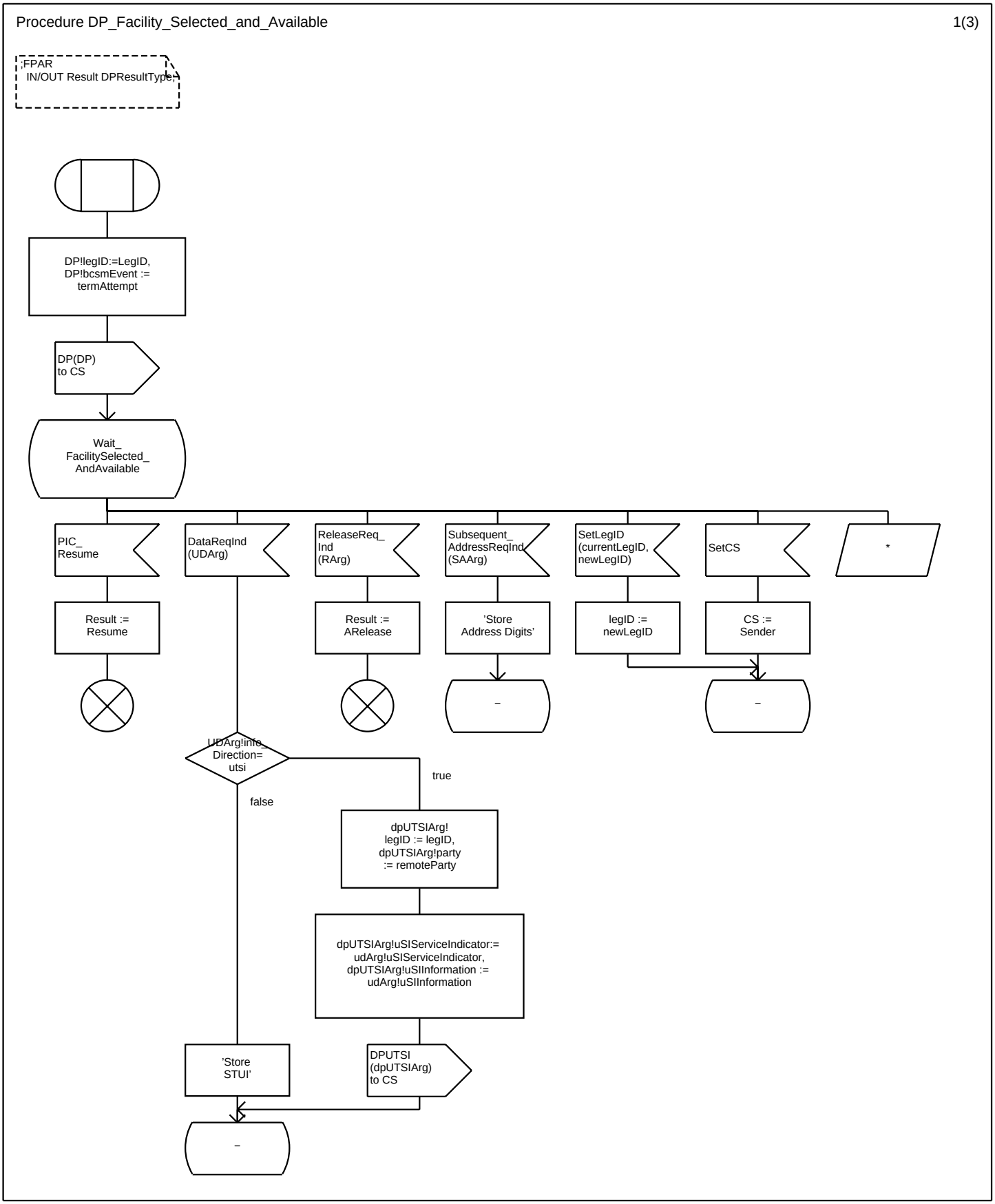

;FPAR
IN/OUT Result DPResultType;
Wait_
tReanswer
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remLegID,
receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS
–
PIC!party
localParty
remoteParty
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
Result :=
SCFRelease
'Error
situation'
–

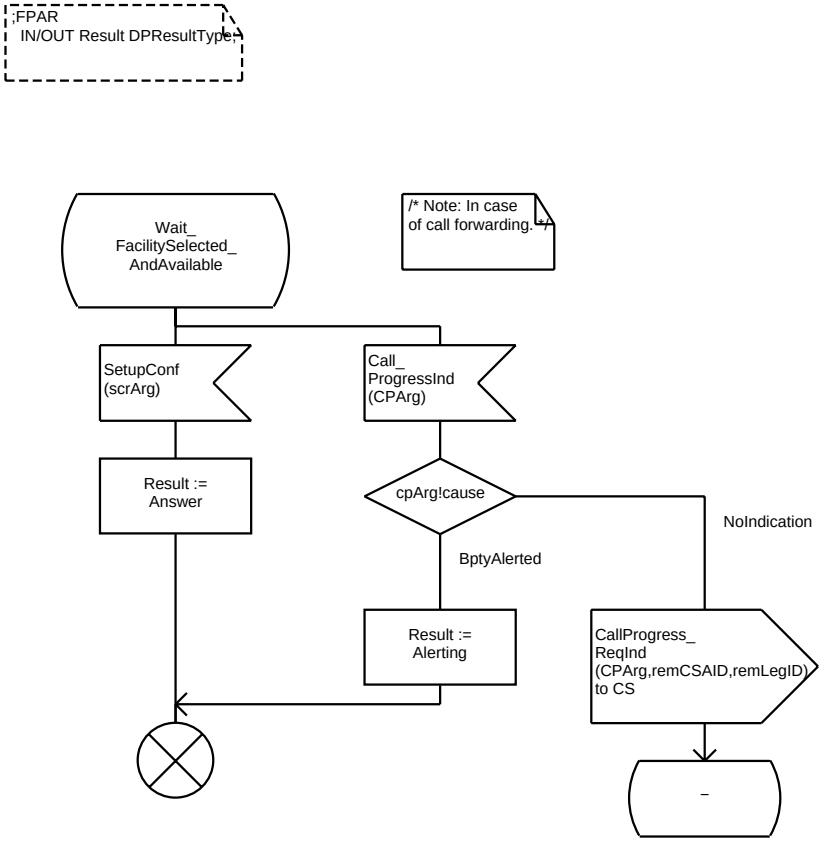

;FPAR
IN/OUT Result DPResultType;
Wait_
tReanswer
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS

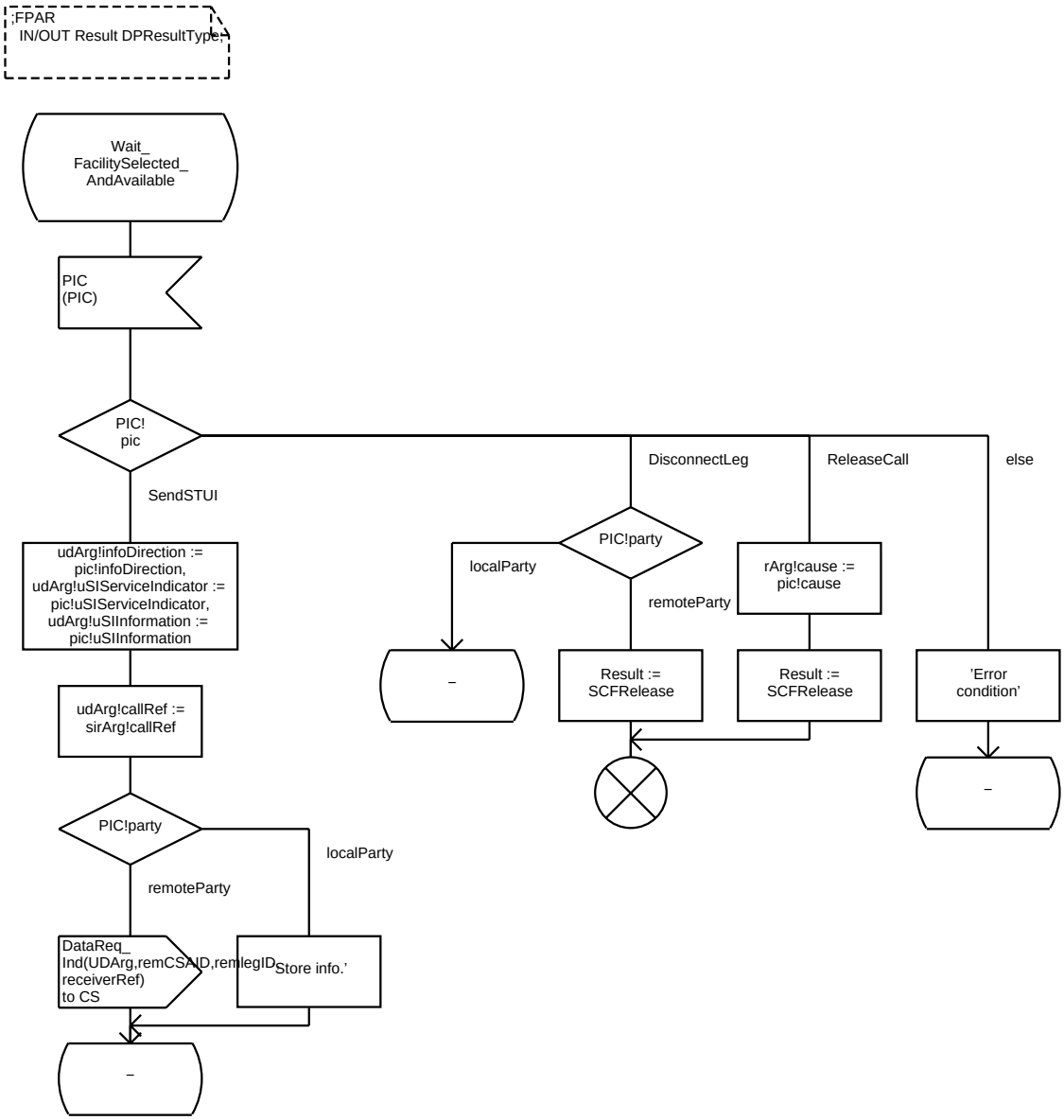

Wait_
tBusy
DataReqInd
(UDArg)
SetLegID
(currentLegID,
newLegID)
SetCS
SetupConf
(scrArg)
Call_
ProgressInd
(CPArg)
legID :=
newLegID
CS :=
Sender
Result :=
Answer
cpArg!cause
NoIndication
BptyAlerted
UDArg!info_
Direction=
utsi
true
false
Result :=
Alerting
CallProgress_
ReqInd
(CPArg,remCSAID,remLegID)
to CS
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
'Store
STUI'
DPUTSI
(dpUTSIArg)
to CS
–
–

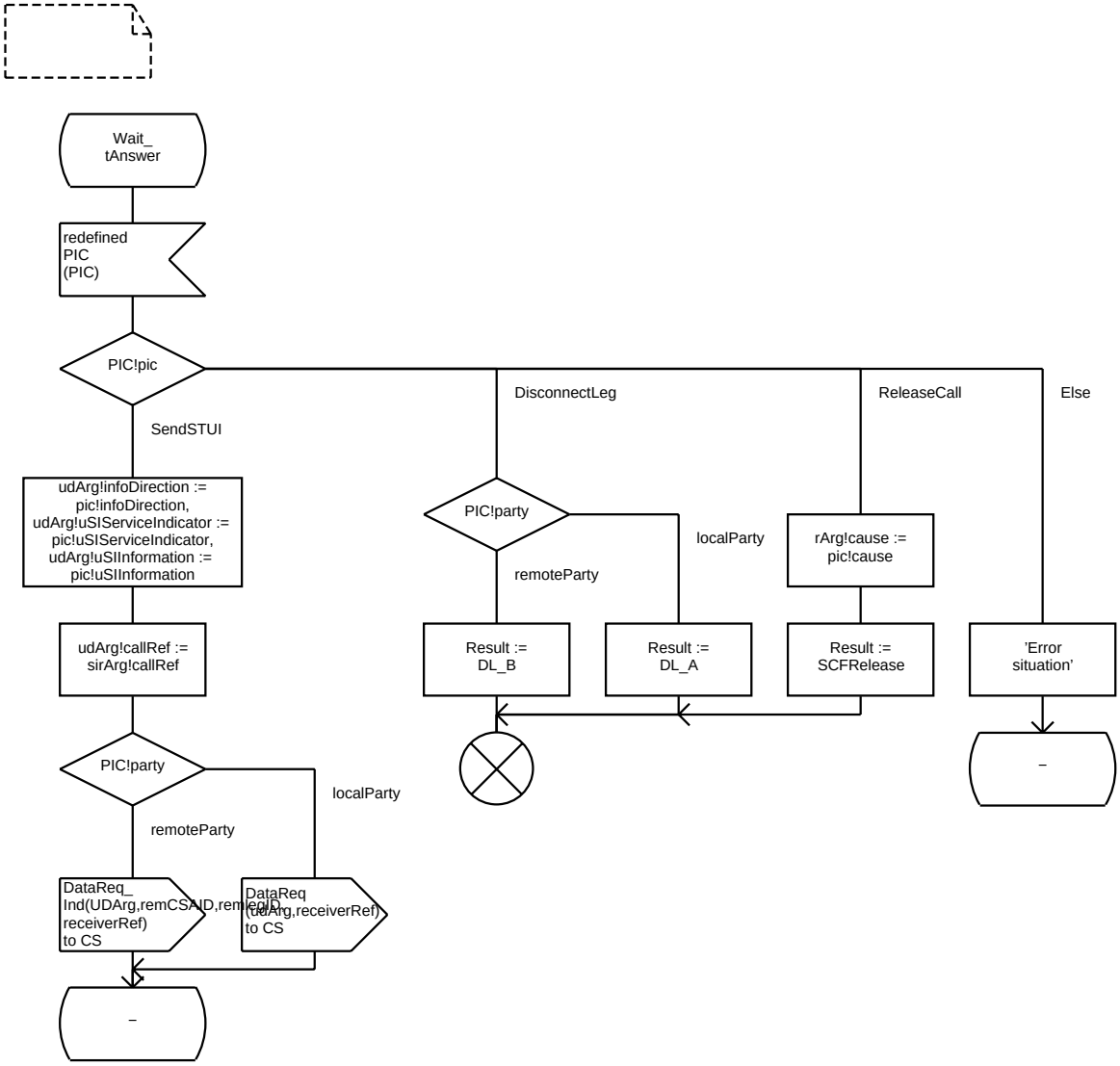

Wait_
tBusy
redefined
PIC
(PIC)
PIC!
pic
SendSTUI
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remlegID,
receiverRef)
to CS
Store info.'
–
DisconnectLeg
ReleaseCall
else
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
–
Result :=
SCFRelease
Result :=
SCFRelease
'Error
condition'
–

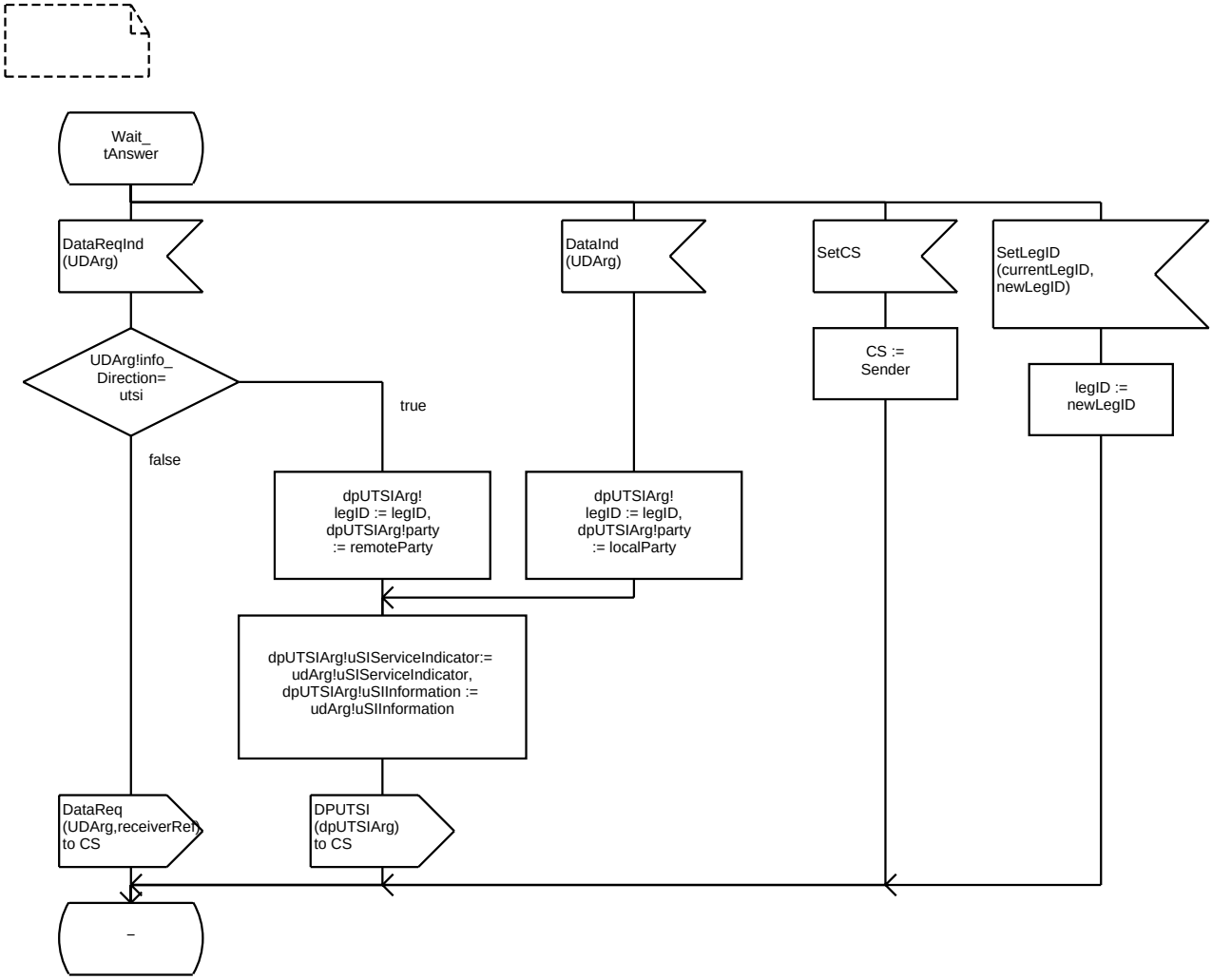

Wait_
tAbandon
SetCS
SetLegID
(currentLegID,
newLegID)
CS :=
Sender
legID :=
newLegID
–

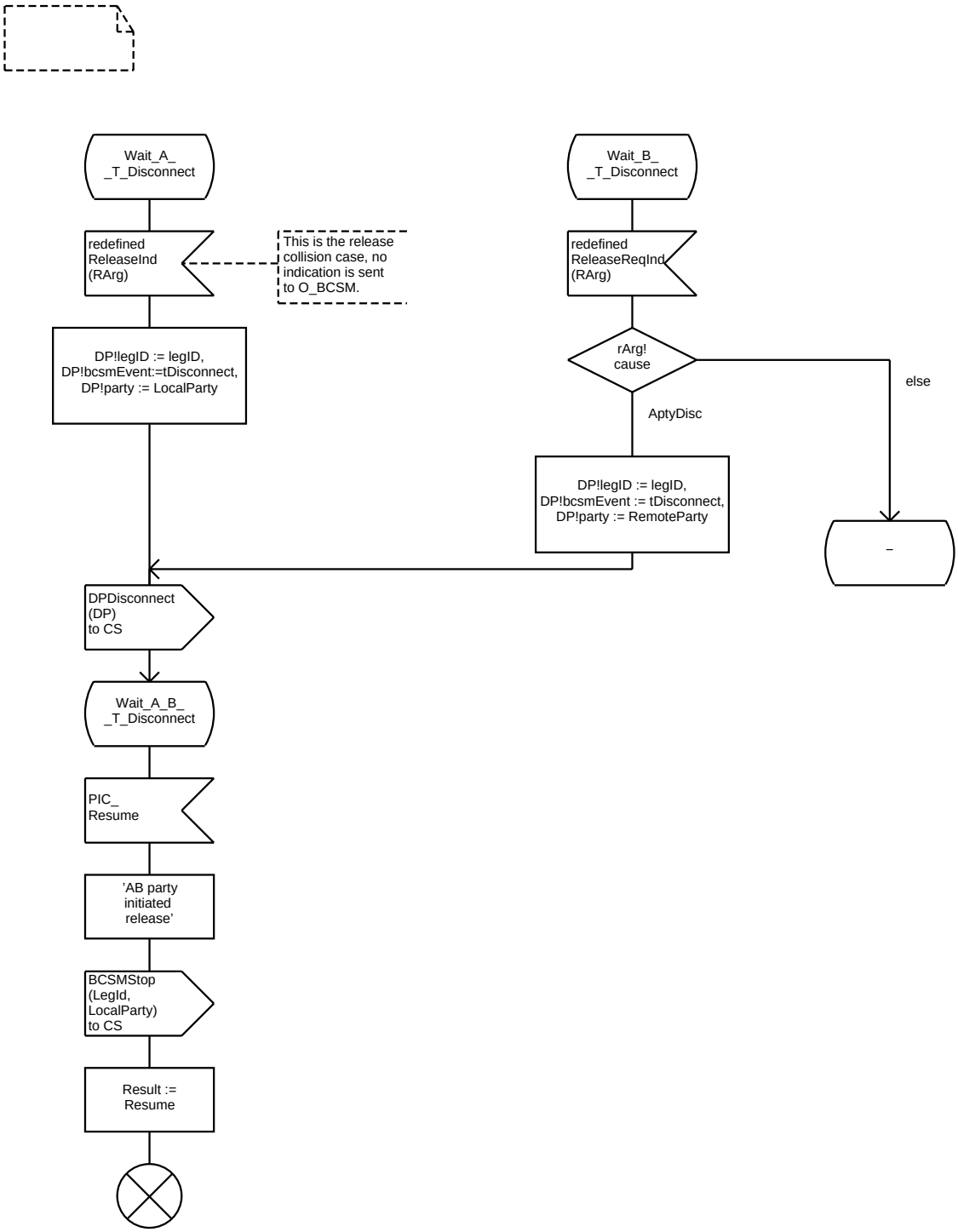

;FPAR
IN/OUT Result DPResultType;
DP!legID:=LegID,
DP!bcsmEvent :=
termAttempt
DP(DP)
to CS
Wait_
FacilitySelected_
AndAvailable
PIC_
Resume
DataReqInd
(UDArg)
ReleaseReq_
Ind
(RArg)
Subsequent_
AddressReqInd
(SAArg)
SetLegID
(currentLegID,
newLegID)
SetCS
*
Result :=
Resume
Result :=
ARelease
'Store
Address Digits'
legID :=
newLegID
CS :=
Sender
–
–
UDArg!info_
Direction=
utsi
true
false
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
'Store
STUI'
DPUTSI
(dpUTSIArg)
to CS
–

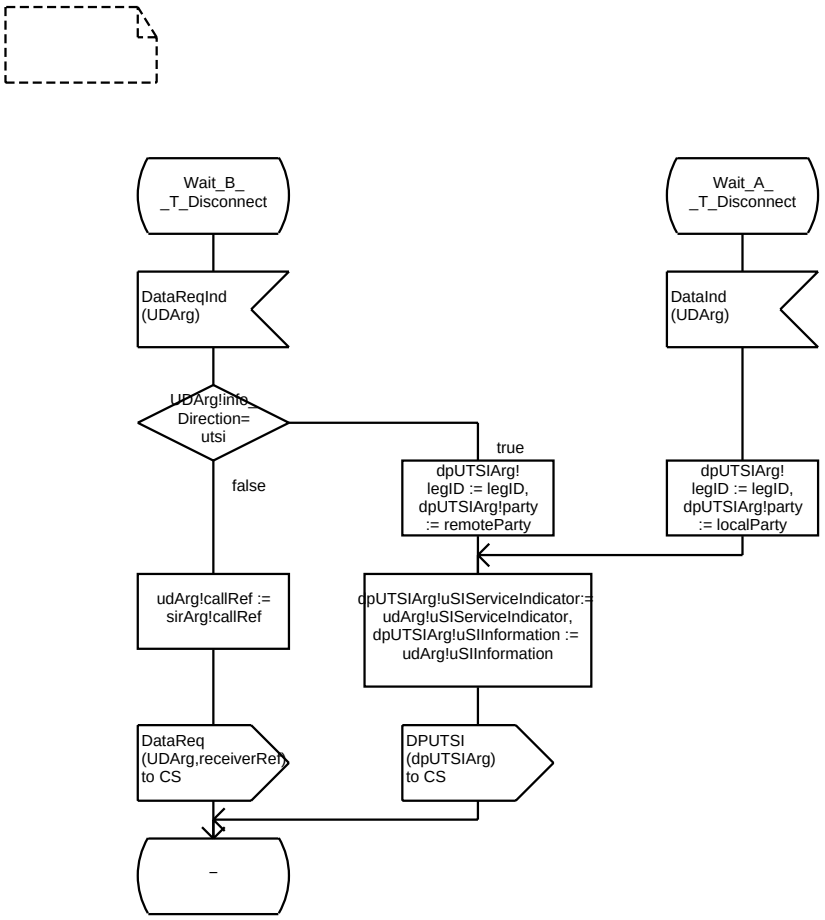

;FPAR
IN/OUT Result DPResultType;
Wait_
FacilitySelected_
AndAvailable
/* Note: In case
of call forwarding. */
SetupConf
(scrArg)
Call_
ProgressInd
(CPArg)
Result :=
Answer
cpArg!cause
NoIndication
BptyAlerted
Result :=
Alerting
CallProgress_
ReqInd
(CPArg,remCSAID,remLegID)
to CS
–

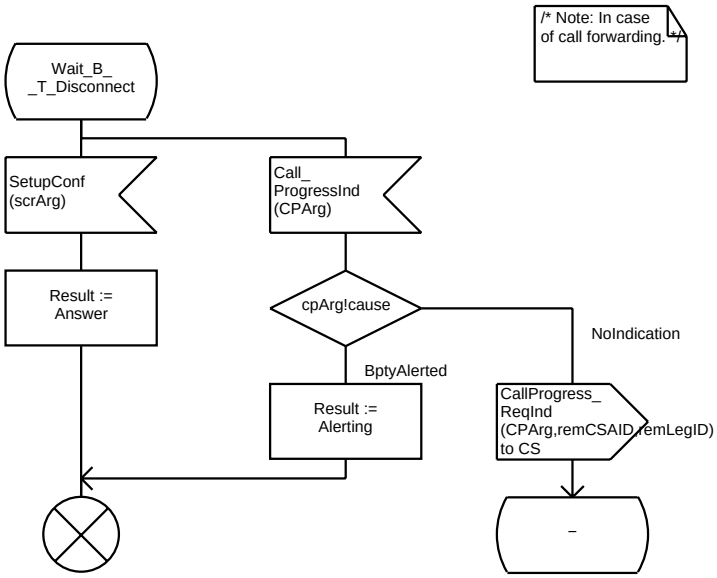

;FPAR
IN/OUT Result DPResultType;
Wait_
FacilitySelected_
AndAvailable
PIC
(PIC)
PIC!
pic
SendSTUI
DisconnectLeg
ReleaseCall
else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
–
Result :=
SCFRelease
Result :=
SCFRelease
'Error
condition'
–
udArg!callRef :=
sirArg!callRef
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remlegID
receiverRef)
to CS
Store info.'
–

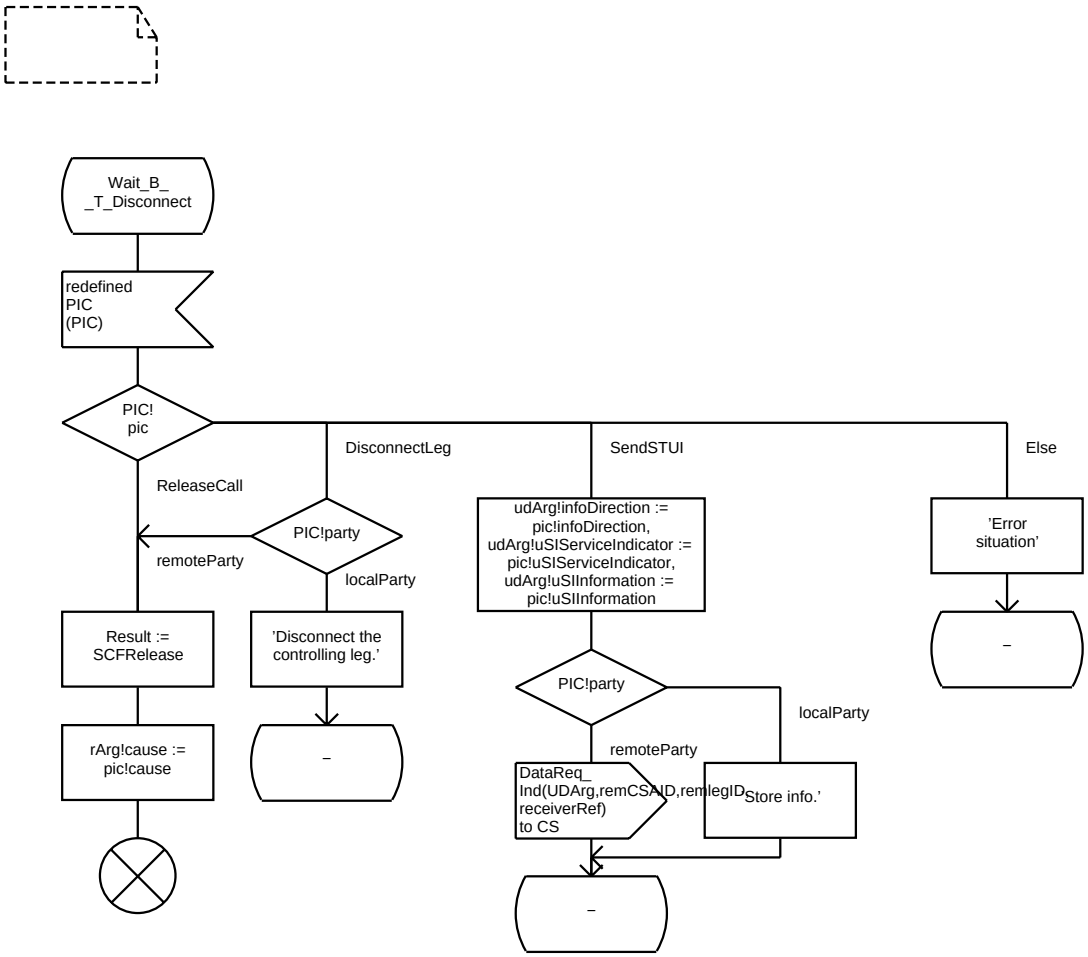

Wait_
tAnswer
redefined
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
rArg!cause :=
pic!cause
udArg!callRef :=
sirArg!callRef
Result :=
DL_B
Result :=
DL_A
Result :=
SCFRelease
'Error
situation'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remLegID,
receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS
–
–

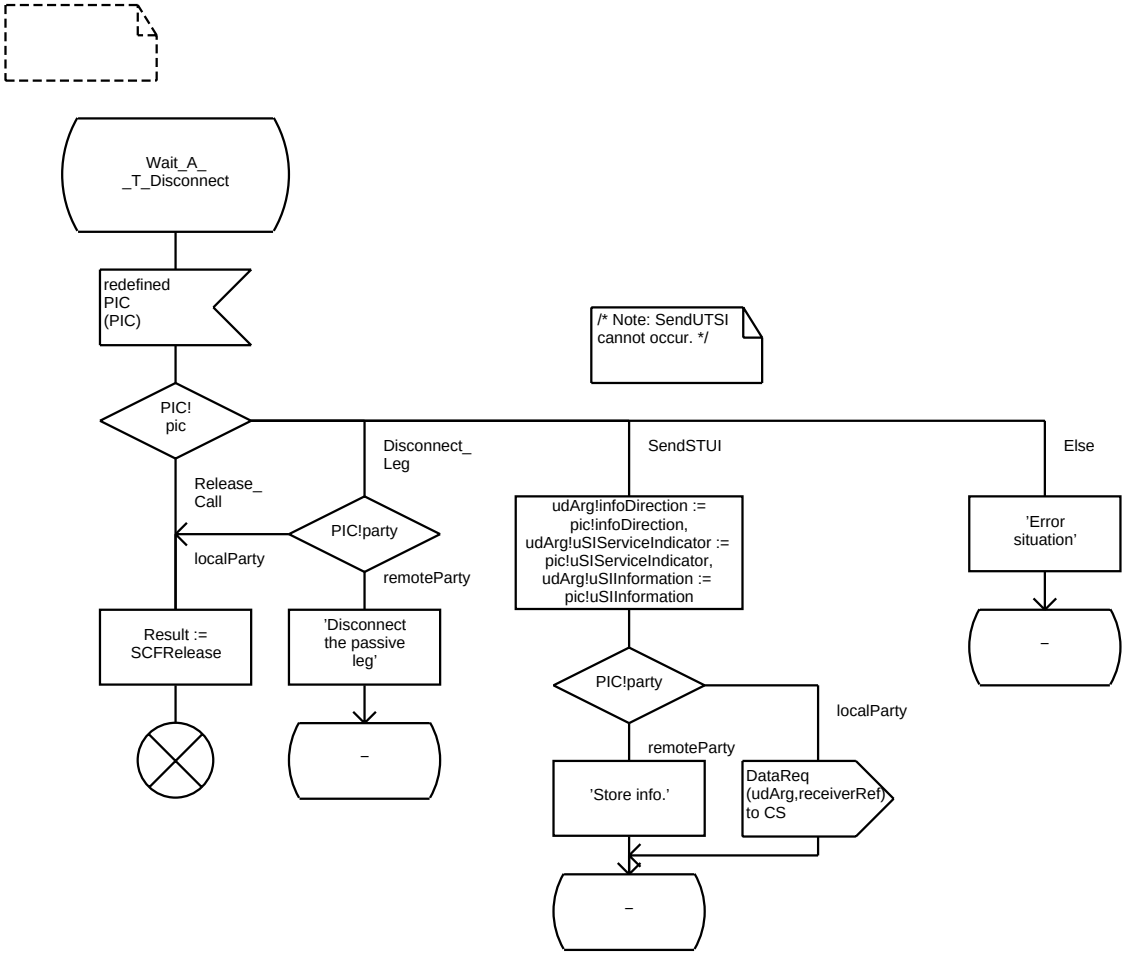

Wait_
tAnswer
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

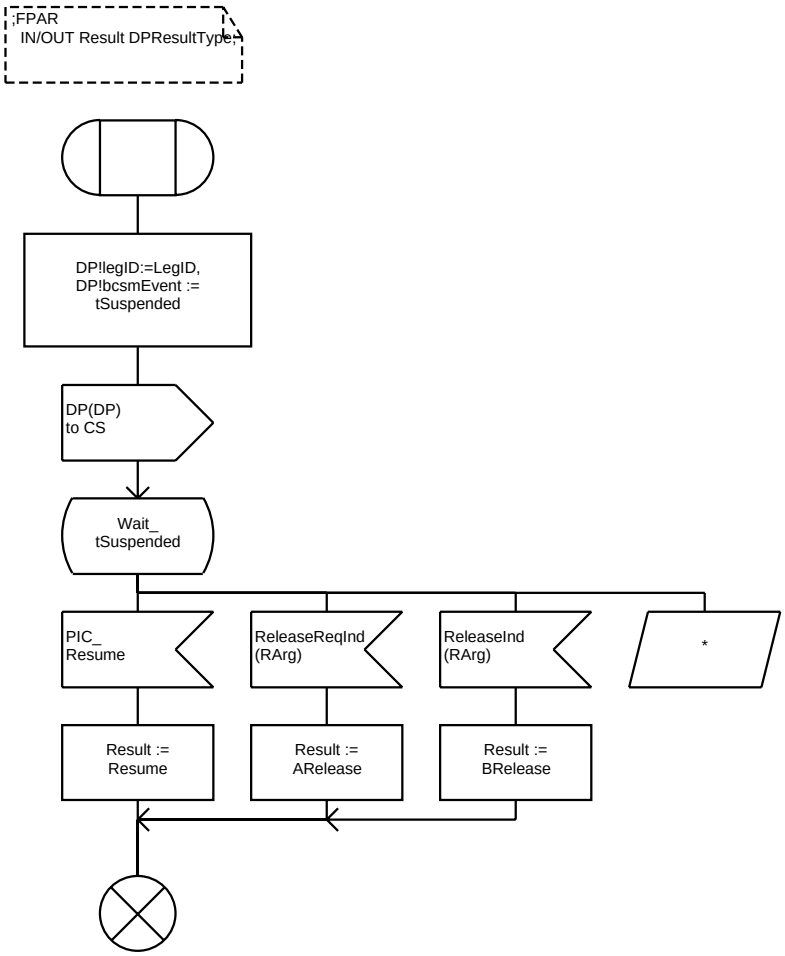

Wait_A_
_T_Disconnect
redefined
ReleaseInd
(RArg)
This is the release collision case, no indication is sent to O_BCSM.
DP!legID := legID,
DP!bcsmEvent:=tDisconnect,
DP!party := LocalParty
Wait_B_
_T_Disconnect
redefined
ReleaseReqInd
(RArg)
rArg!
cause
else
AptyDisc
DP!legID := legID,
DP!bcsmEvent := tDisconnect,
DP!party := RemoteParty
–
DPDisconnect
(DP)
to CS
Wait_A_B_
_T_Disconnect
PIC_
Resume
'AB party
initiated
release'
BCSMStop
(LegId,
LocalParty)
to CS
Result :=
Resume

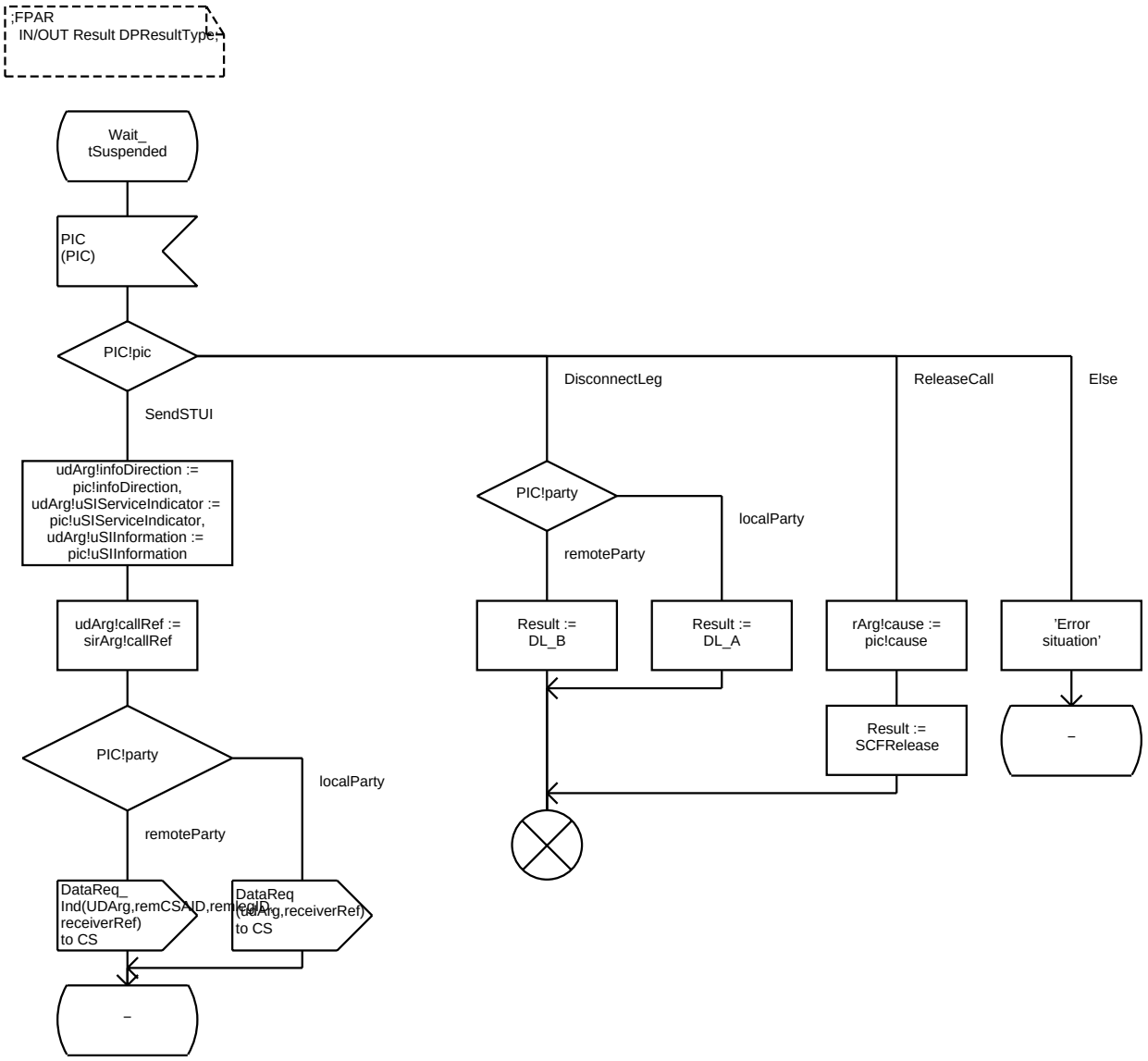

Wait_B_
_T_Disconnect
Wait_A_
_T_Disconnect
DataReqInd
(UDArg)
DataInd
(UDArg)
UDArg!info_
Direction=
utsi
true
false
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
udArg!callRef :=
sirArg!callRef
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

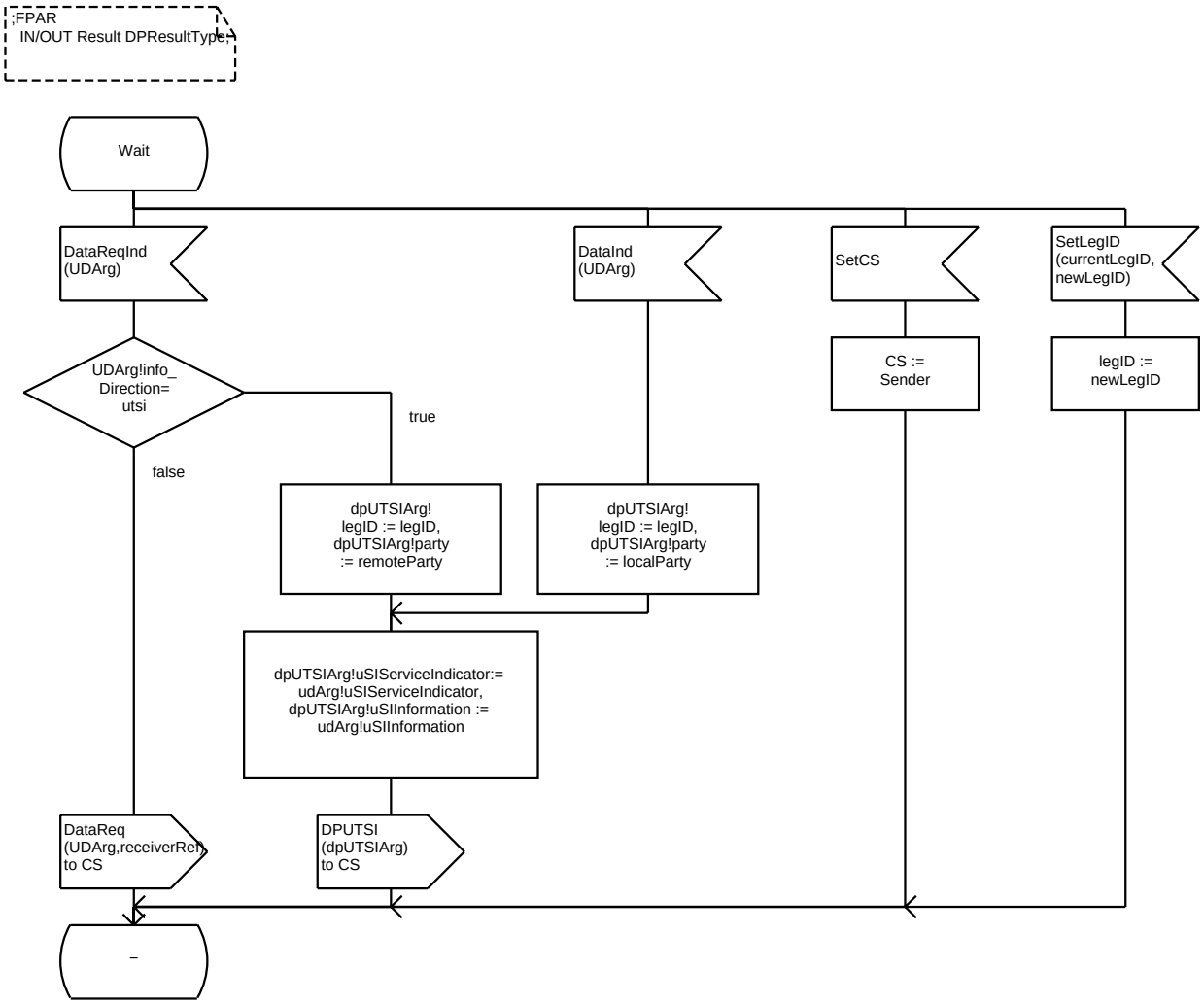

/* Note: In case of call forwarding. */
Wait_B_ _T_Disconnect
SetupConf (scrArg)
Call_ ProgressInd (CPArg)
Result := Answer
cpArg!cause
NoIndication
BptyAlerted
Result := Alerting
CallProgress_ ReqInd (CPArg,remCSAID,remLegID) to CS
–

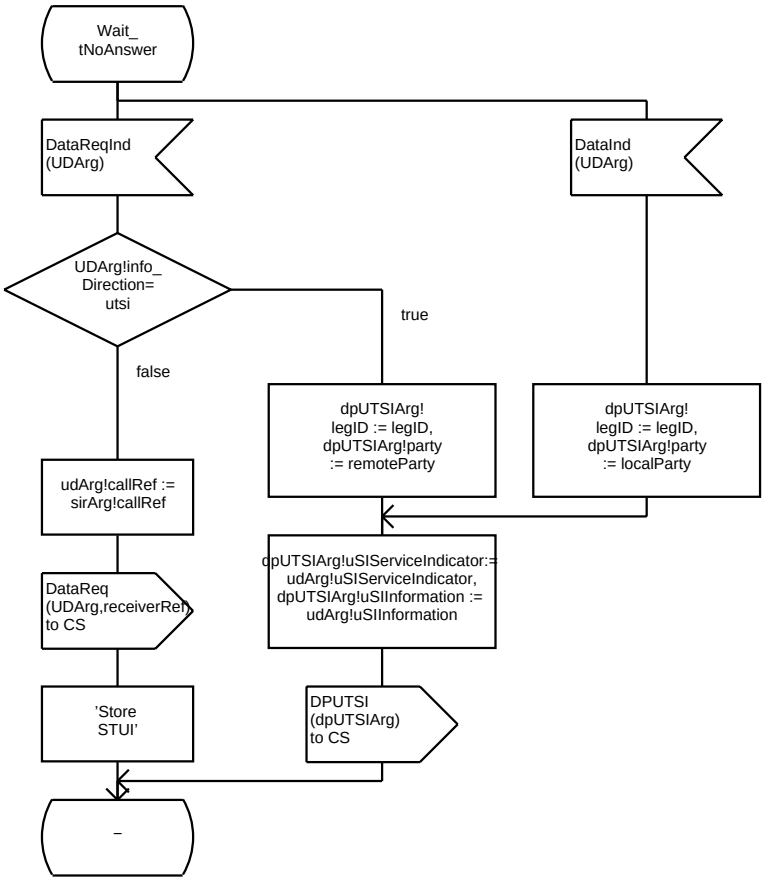

Wait_B_
_T_Disconnect
redefined
PIC
(PIC)
PIC!
pic
DisconnectLeg
SendSTUI
Else
ReleaseCall
PIC!party
remoteParty
localParty
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
Result :=
SCFRelease
'Disconnect the
controlling leg.'
–
rArg!cause :=
pic!cause
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remlegID,
receiverRef)
to CS
'Store info.'
–

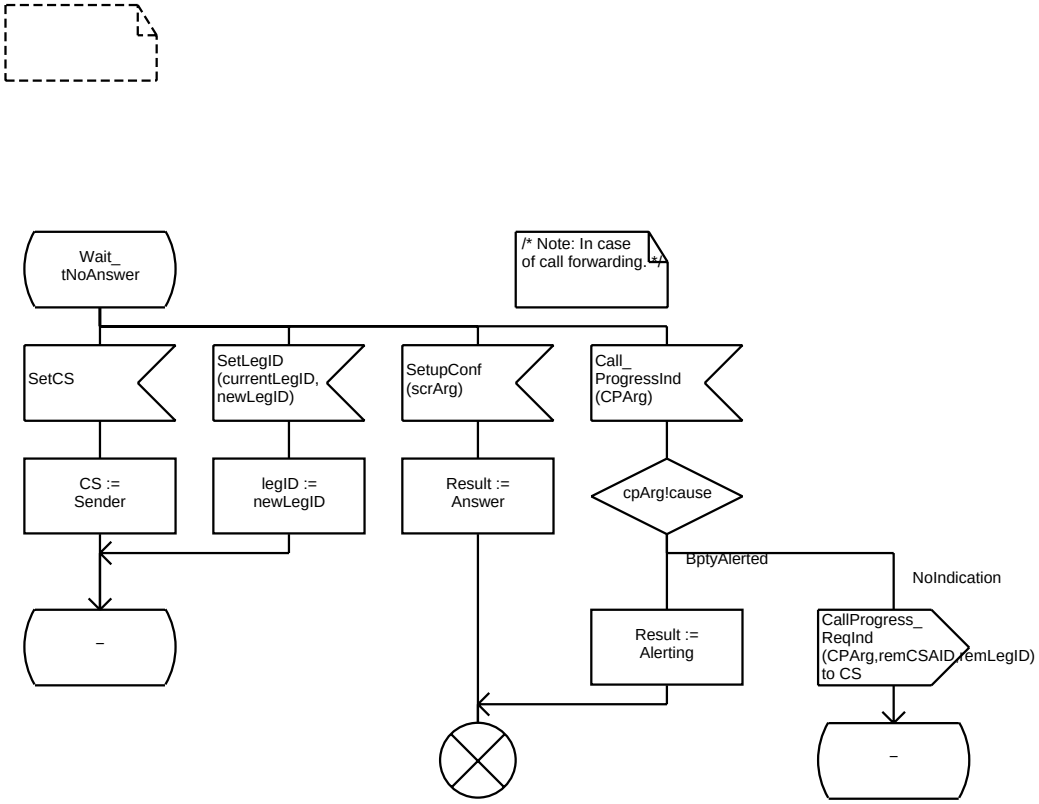

Wait_A_
_T_Disconnect
redefined
PIC
(PIC)
/* Note: SendUTSI
cannot occur. */
PIC!
pic
Disconnect_
Leg
SendSTUI
Else
Release_
Call
PIC!party
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
'Error
situation'
localParty
remoteParty
Result :=
SCFRelease
'Disconnect
the passive
leg'
–
PIC!party
localParty
remoteParty
'Store info.'
DataReq
(udArg,receiverRef)
to CS
–

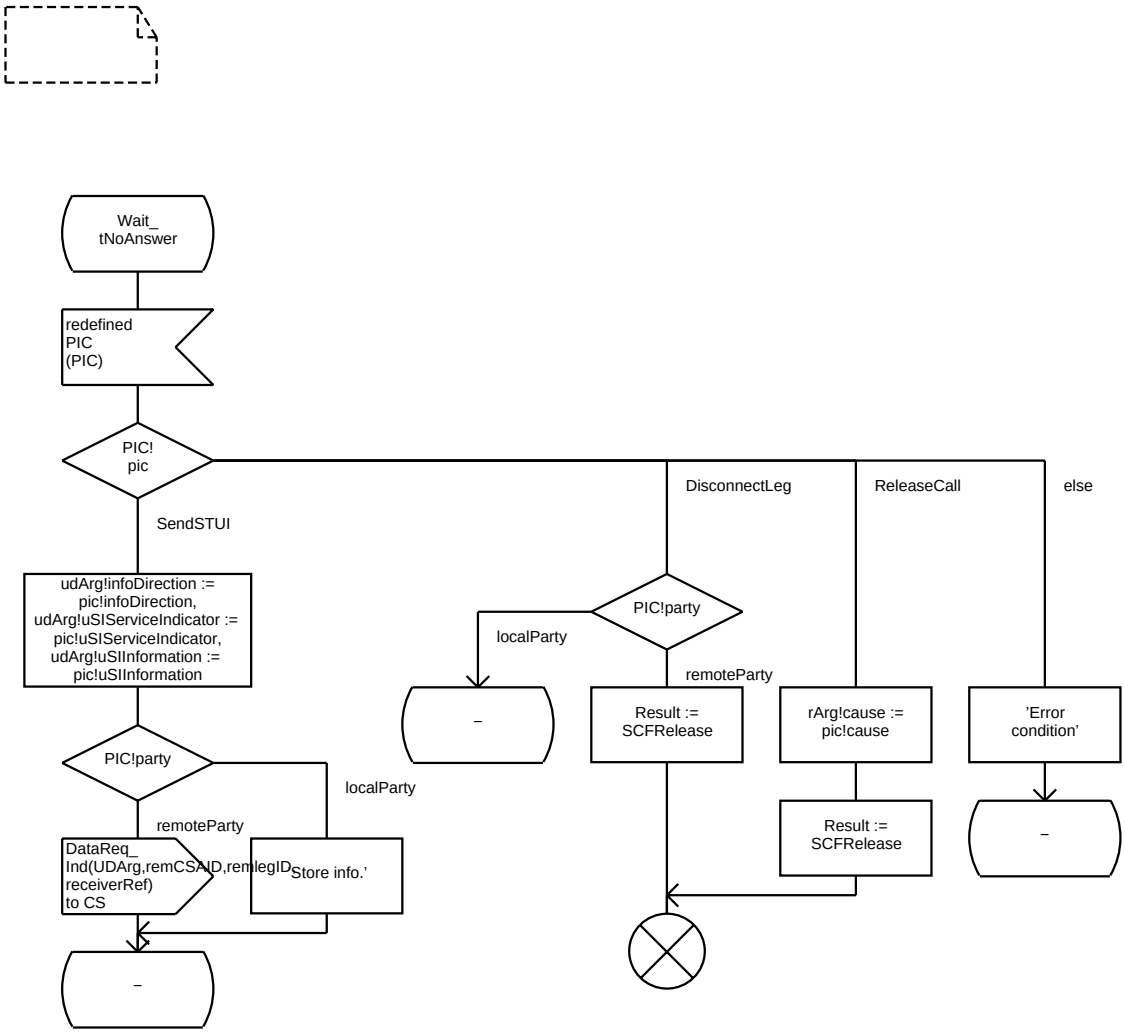

;FPAR
IN/OUT Result DPResultType;
DP!legID:=LegID,
DP!bcsmEvent :=
tSuspended
DP(DP)
to CS
Wait_
tSuspended
PIC_
Resume
ReleaseReqInd
(RArg)
ReleaseInd
(RArg)
*
Result :=
Resume
Result :=
ARelease
Result :=
BRelease

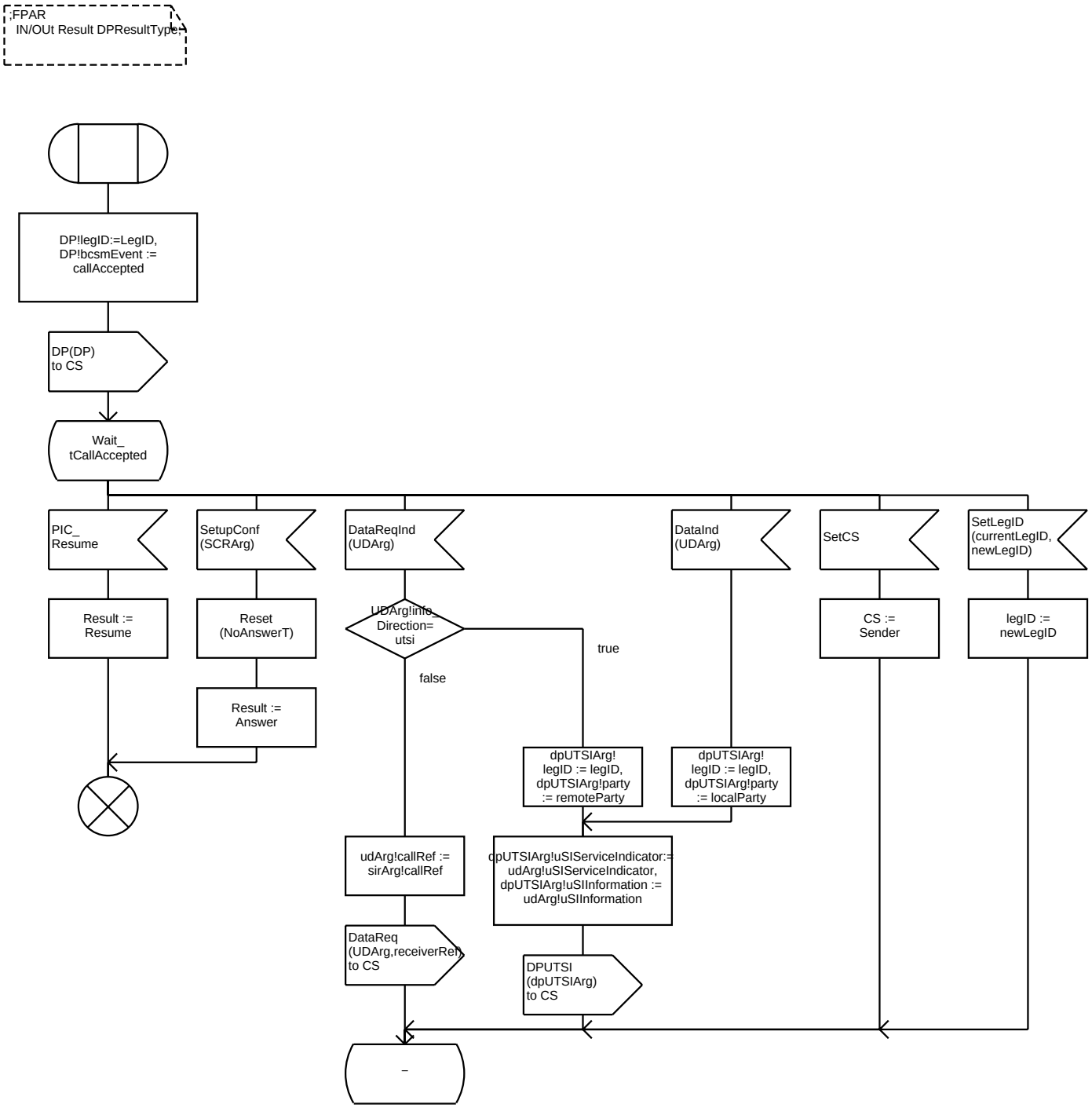

;FPAR
IN/OUT Result DPResultType;
Wait_
tSuspended
PIC
(PIC)
PIC!pic
SendSTUI
DisconnectLeg
ReleaseCall
Else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
udArg!callRef :=
sirArg!callRef
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remLegID,
receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS

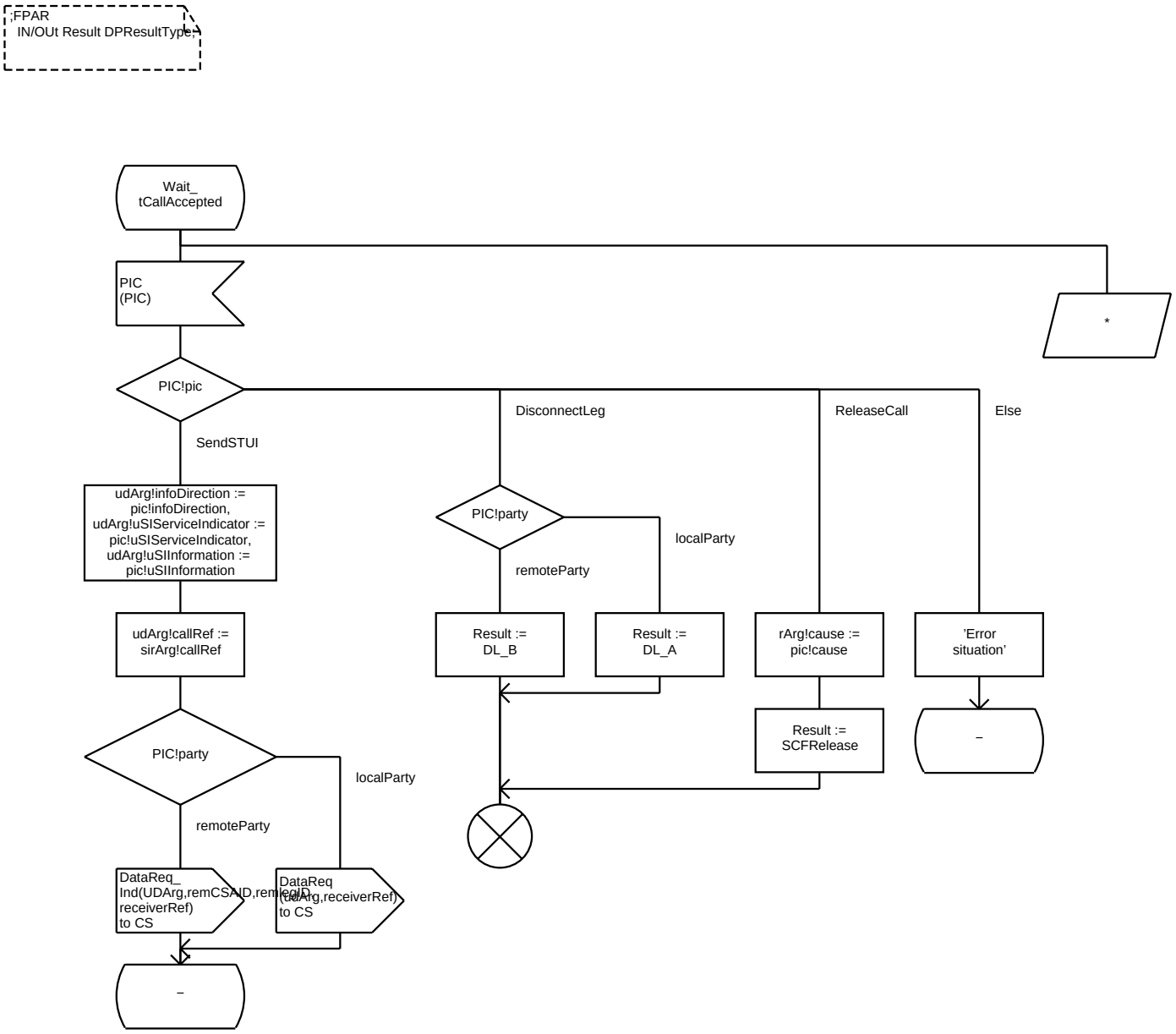

;FPAR
IN/OUT Result DPResultType;
Wait
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

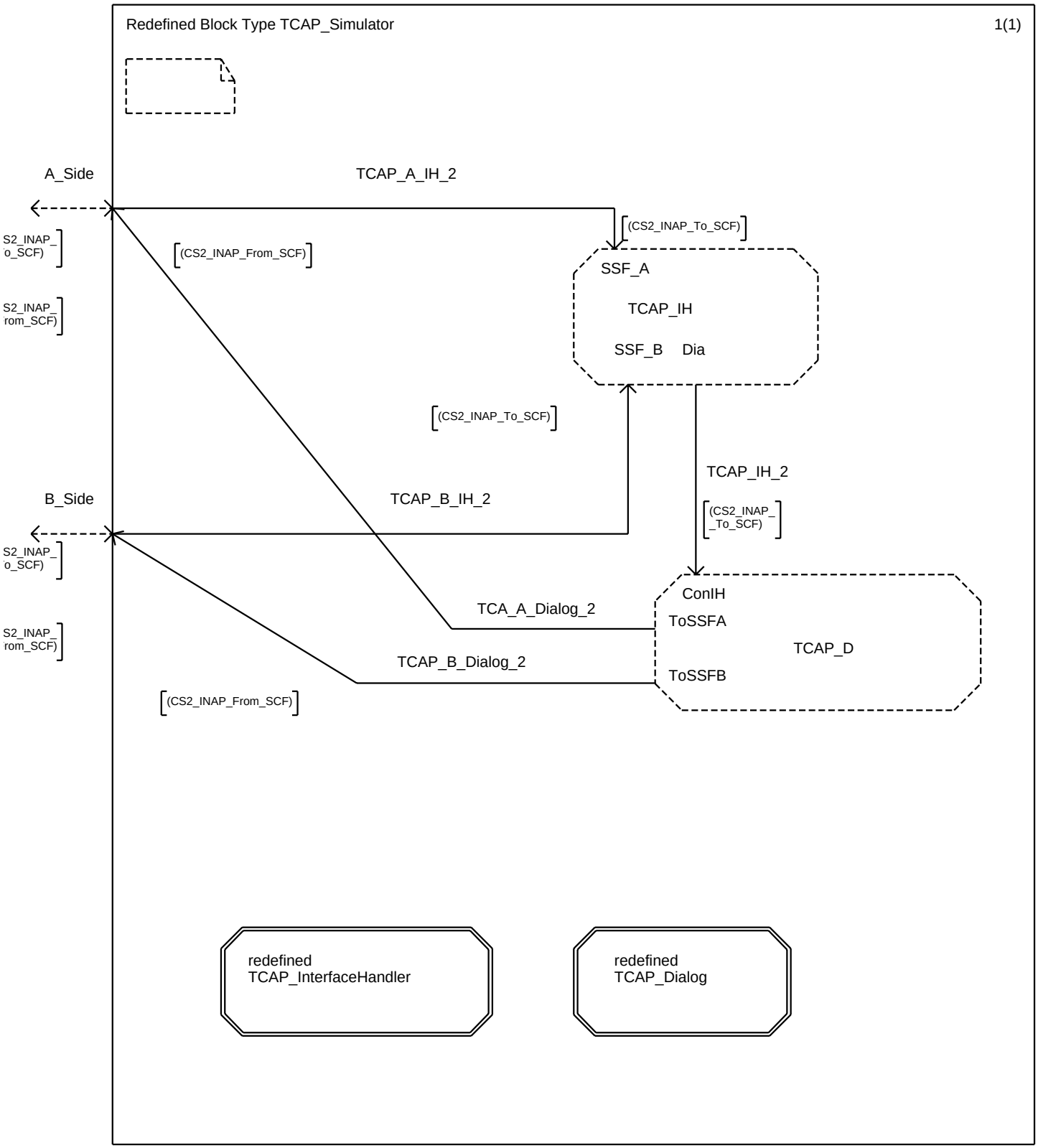

Wait_
tNoAnswer
DataReqInd
(UDArg)
DataInd
(UDArg)
UDArg!info_
Direction=
utsi
true
false
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
udArg!callRef :=
sirArg!callRef
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
'Store
STUI'
DPUTSI
(dpUTSIArg)
to CS
–

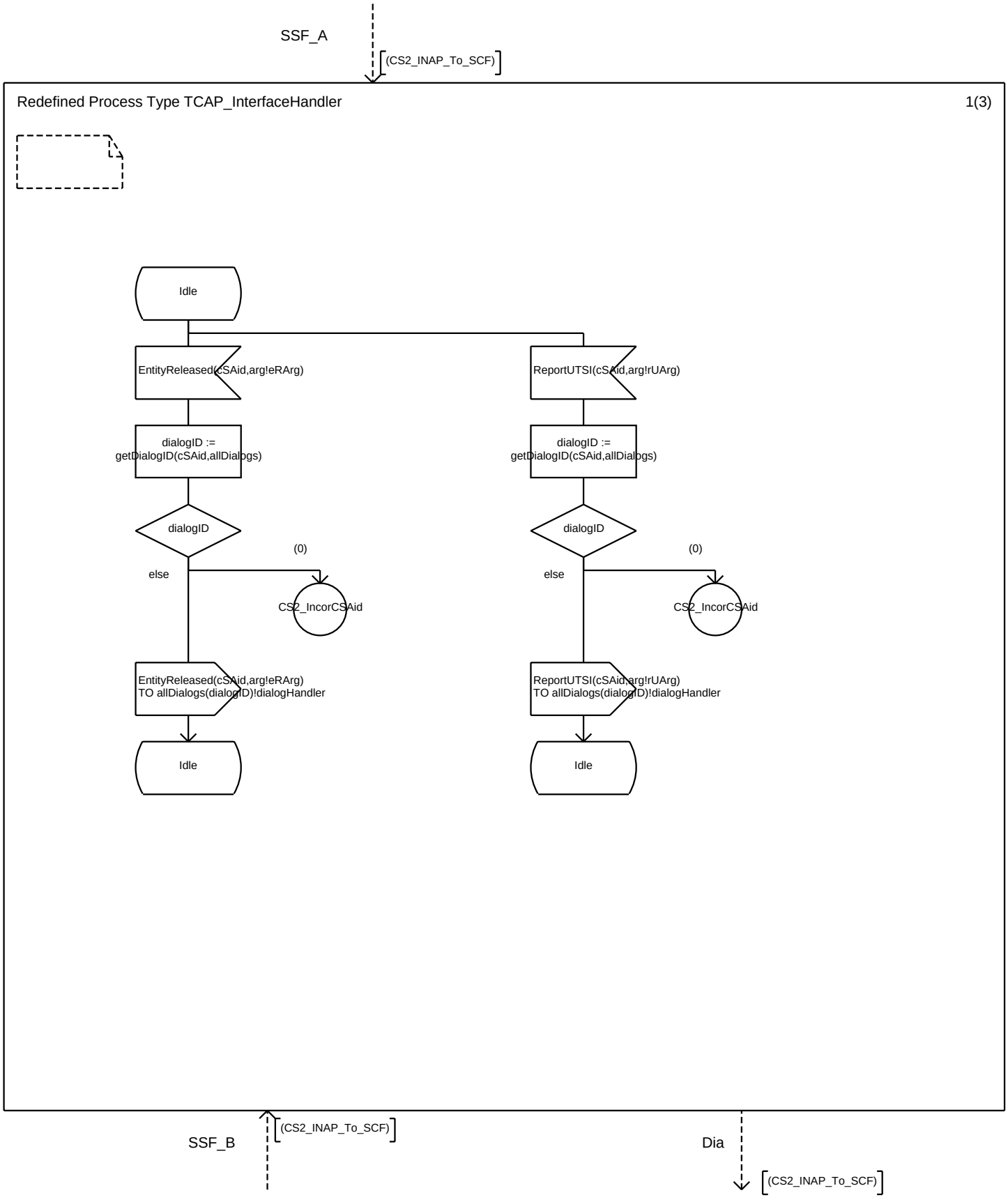

Wait_
tNoAnswer
/* Note: In case
of call forwarding. */
SetCS
SetLegID
(currentLegID,
newLegID)
SetupConf
(scrArg)
Call_
ProgressInd
(CPArg)
CS :=
Sender
legID :=
newLegID
Result :=
Answer
cpArg!cause
BptyAlerted
NoIndication
–
Result :=
Alerting
CallProgress_
ReqInd
(CPArg,remCSAID,remLegID)
to CS
–

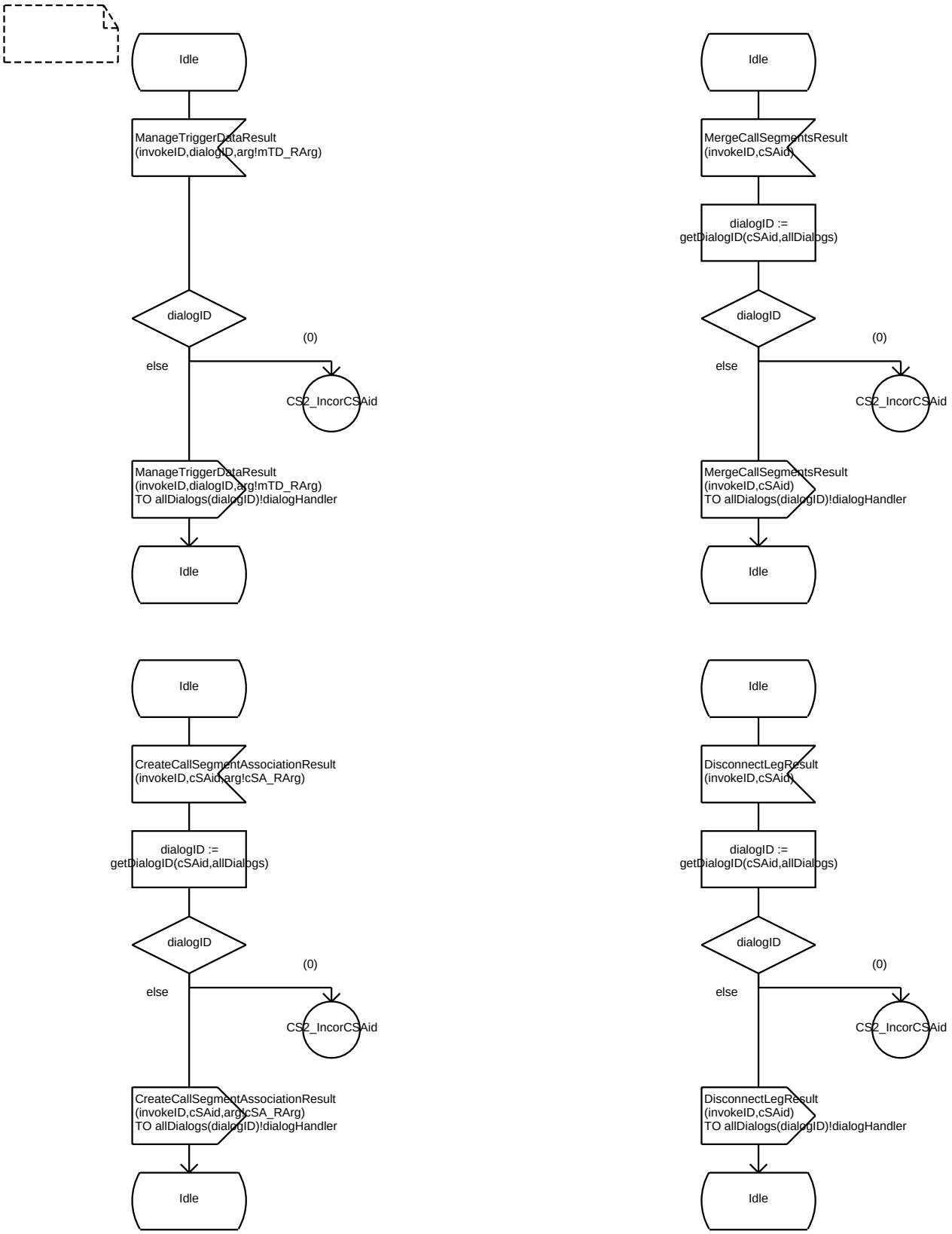

Wait_
tNoAnswer
redefined
PIC
(PIC)
PIC!
pic
SendSTUI
DisconnectLeg
ReleaseCall
else
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
–
Result :=
SCFRelease
rArg!cause :=
pic!cause
'Error
condition'
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remlegID,
receiverRef)
to CS
Store info.'
Result :=
SCFRelease
–
–

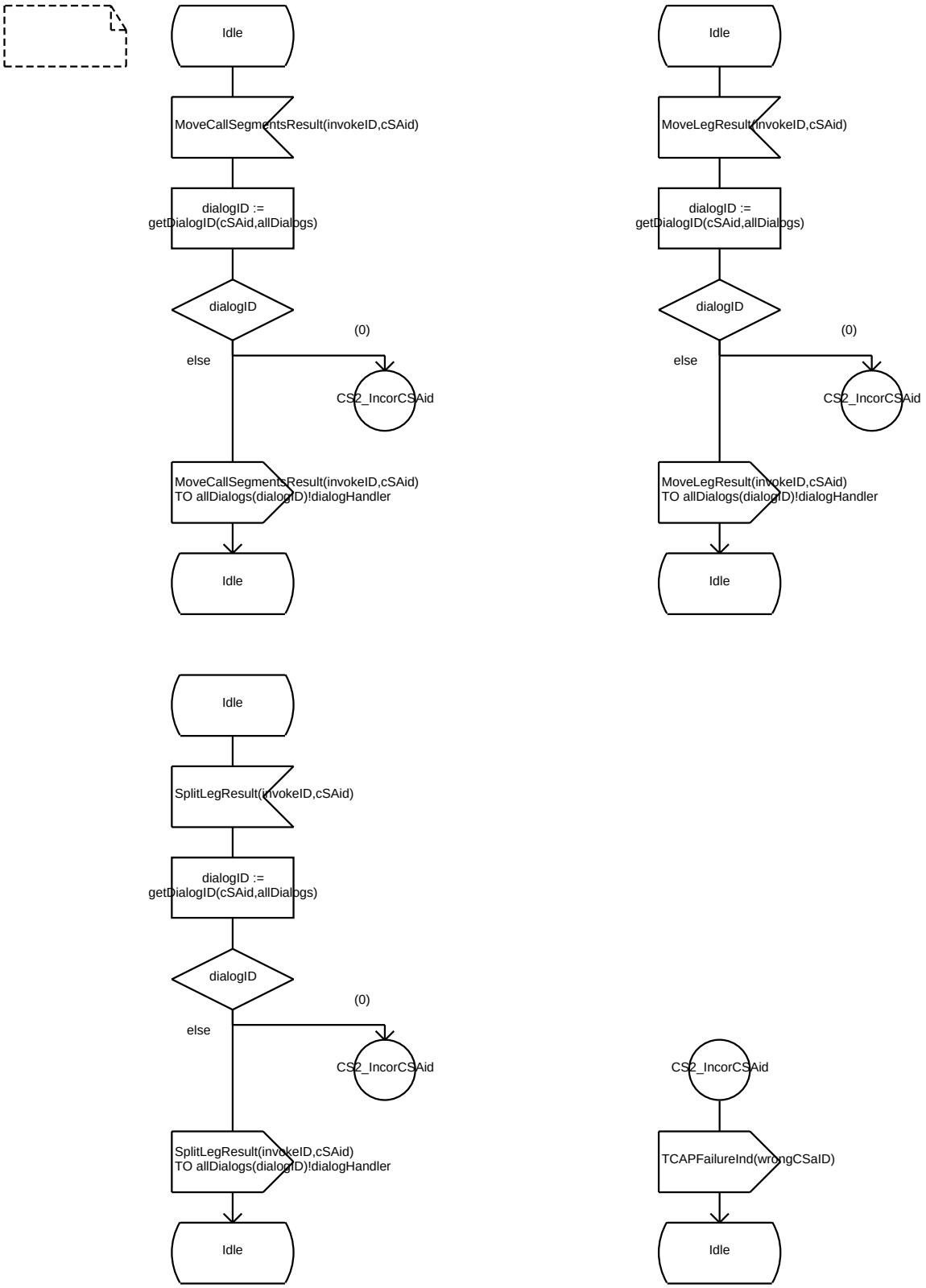

;FPAR
IN/OUt Result DPResultType;
DP!legID:=LegID,
DP!bcsmEvent :=
callAccepted
DP(DP)
to CS
Wait_
tCallAccepted
PIC_
Resume
SetupConf
(SCRArg)
DataReqInd
(UDArg)
DataInd
(UDArg)
SetCS
SetLegID
(currentLegID,
newLegID)
Result :=
Resume
Reset
(NoAnswerT)
UDArg!info_
Direction=
utsi
true
false
CS :=
Sender
legID :=
newLegID
Result :=
Answer
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= remoteParty
dpUTSIArg!
legID := legID,
dpUTSIArg!party
:= localParty
udArg!callRef :=
sirArg!callRef
dpUTSIArg!uSIServiceIndicator:=
udArg!uSIServiceIndicator,
dpUTSIArg!uSIInformation :=
udArg!uSIInformation
DataReq
(UDArg,receiverRef)
to CS
DPUTSI
(dpUTSIArg)
to CS
–

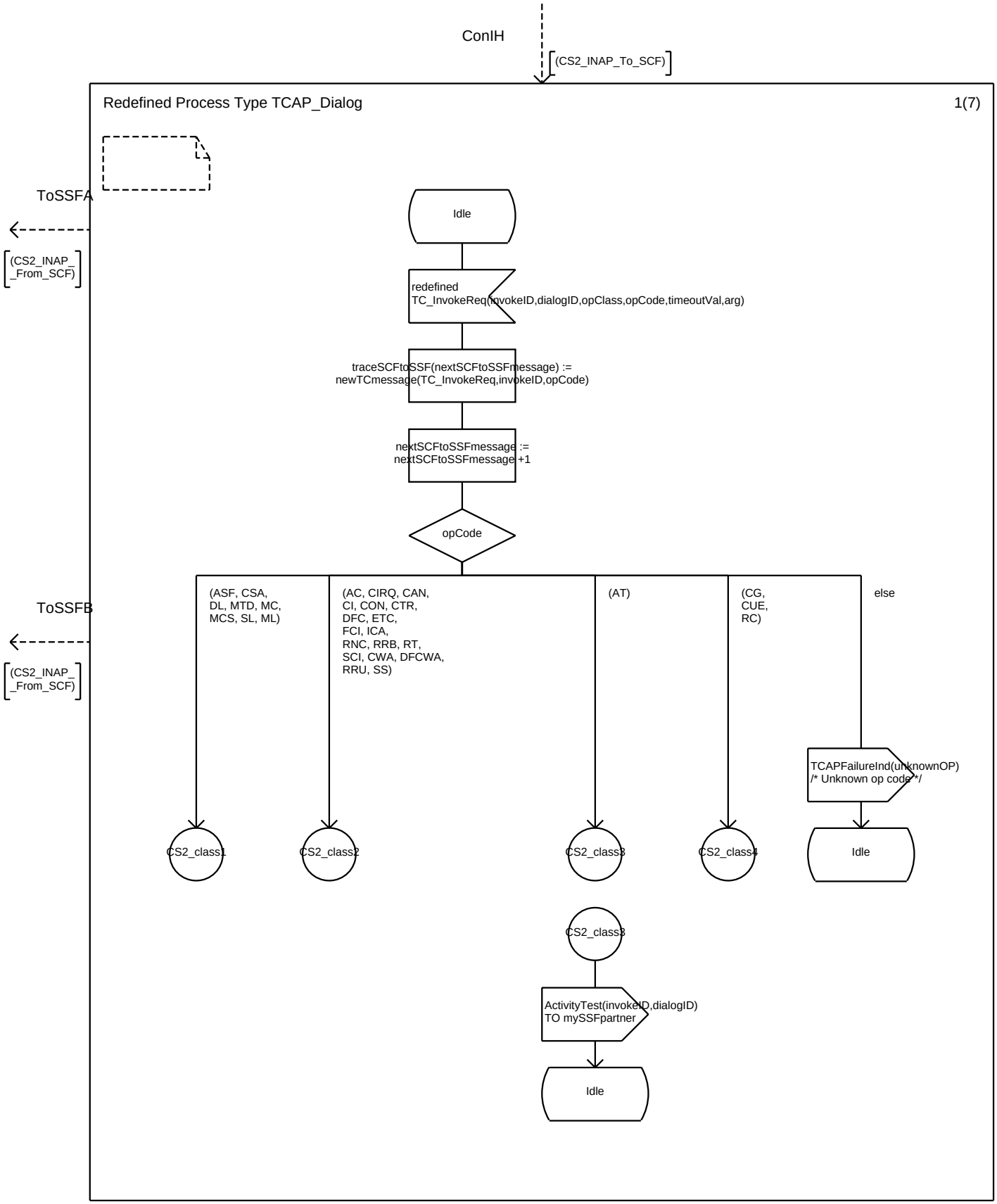

;FPAR
IN/OUt Result DPResultType;
Wait_
tCallAccepted
PIC
(PIC)
*
PIC!pic
DisconnectLeg
ReleaseCall
Else
SendSTUI
udArg!infoDirection :=
pic!infoDirection,
udArg!uSIServiceIndicator :=
pic!uSIServiceIndicator,
udArg!uSIInformation :=
pic!uSIInformation
PIC!party
localParty
remoteParty
udArg!callRef :=
sirArg!callRef
Result :=
DL_B
Result :=
DL_A
rArg!cause :=
pic!cause
'Error
situation'
Result :=
SCFRelease
PIC!party
localParty
remoteParty
DataReq_
Ind(UDArg,remCSAID,remEqID,
receiverRef)
to CS
DataReq
(udArg,receiverRef)
to CS

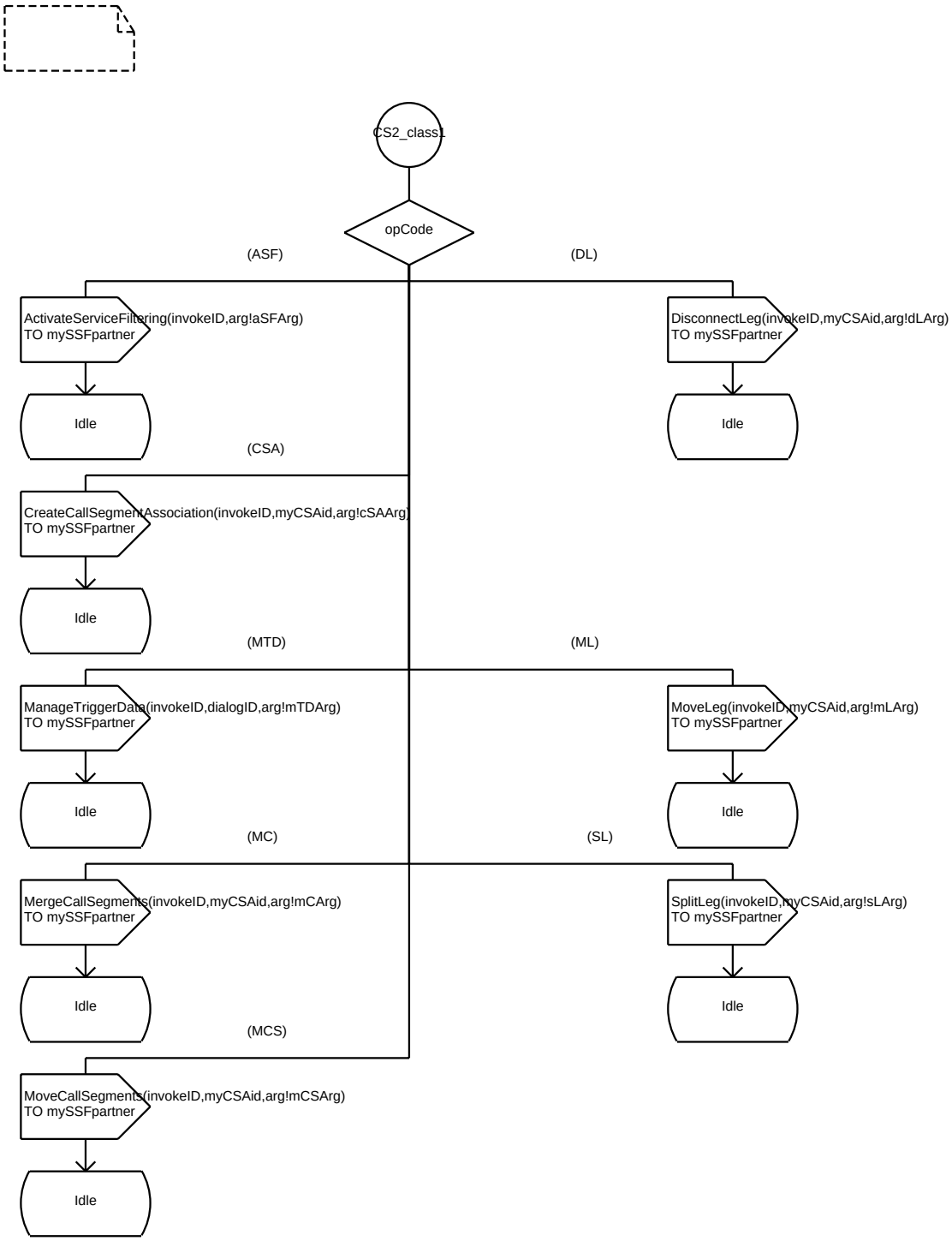

Redefined Block Type TCAP_Simulator
1(1)
A_Side
TCAP_A_IH_2
(CS2_INAP_To_SCF)
(CS2_INAP_From_SCF)
SSF_A
TCAP_IH
SSF_B Dia
(CS2_INAP_To_SCF)
TCAP_IH_2
B_Side
TCAP_B_IH_2
(CS2_INAP_ _To_SCF)
ConIH
TCA_A_Dialog_2
ToSSFA
TCAP_D
TCAP_B_Dialog_2
ToSSFB
(CS2_INAP_From_SCF)
redefined
TCAP_InterfaceHandler
redefined
TCAP_Dialog

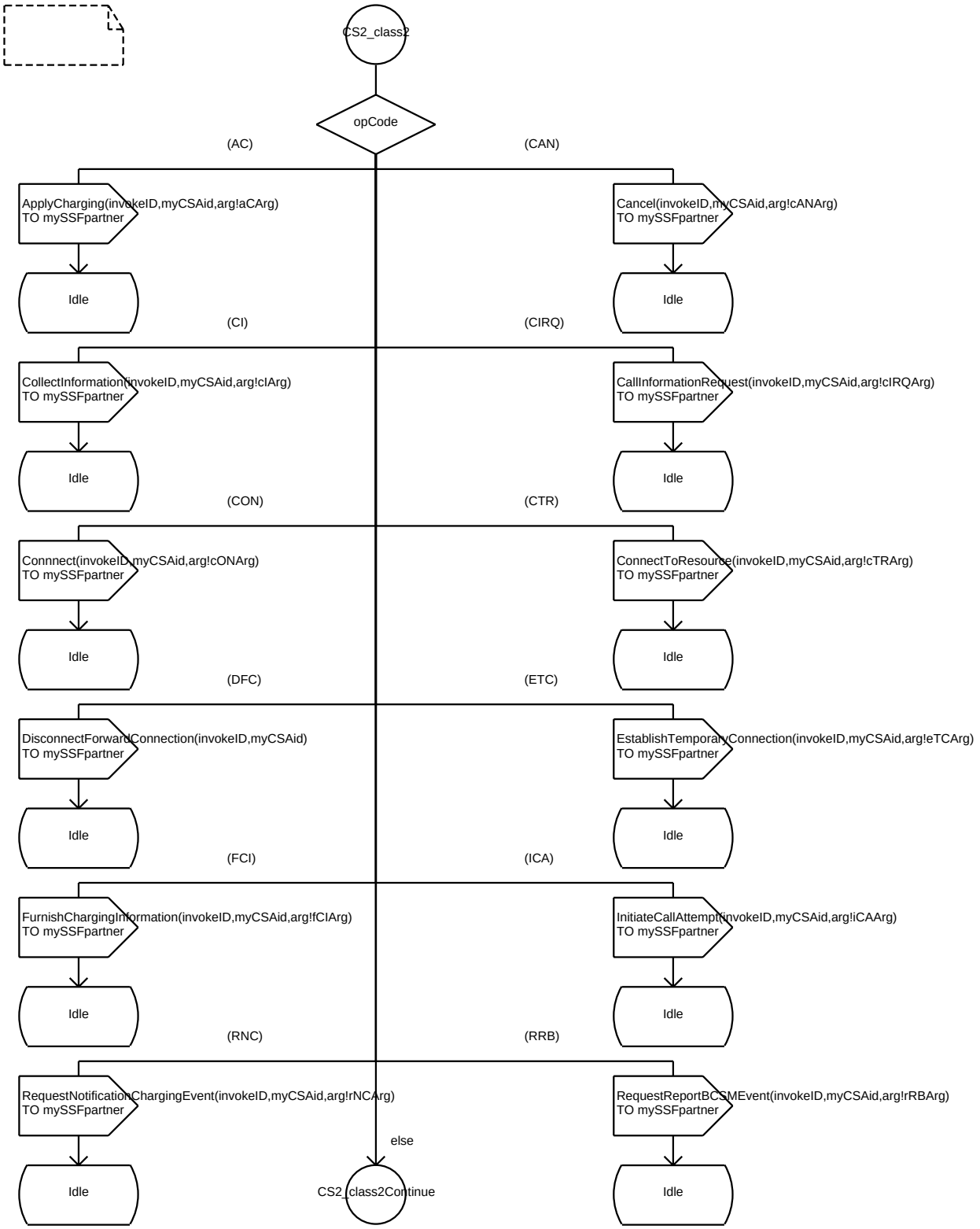

SSF_A
(CS2_INAP_To_SCF)
Redefined Process Type TCAP_InterfaceHandler
1(3)
Idle
EntityReleased(cSAid,arg!eRArg)
dialogID := getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
EntityReleased(cSAid,arg!eRArg) TO allDialogs(dialogID)!dialogHandler
Idle
ReportUTSI(cSAid,arg!rUArg)
dialogID := getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
ReportUTSI(cSAid,arg!rUArg) TO allDialogs(dialogID)!dialogHandler
Idle
SSF_B
(CS2_INAP_To_SCF)
Dia
(CS2_INAP_To_SCF)

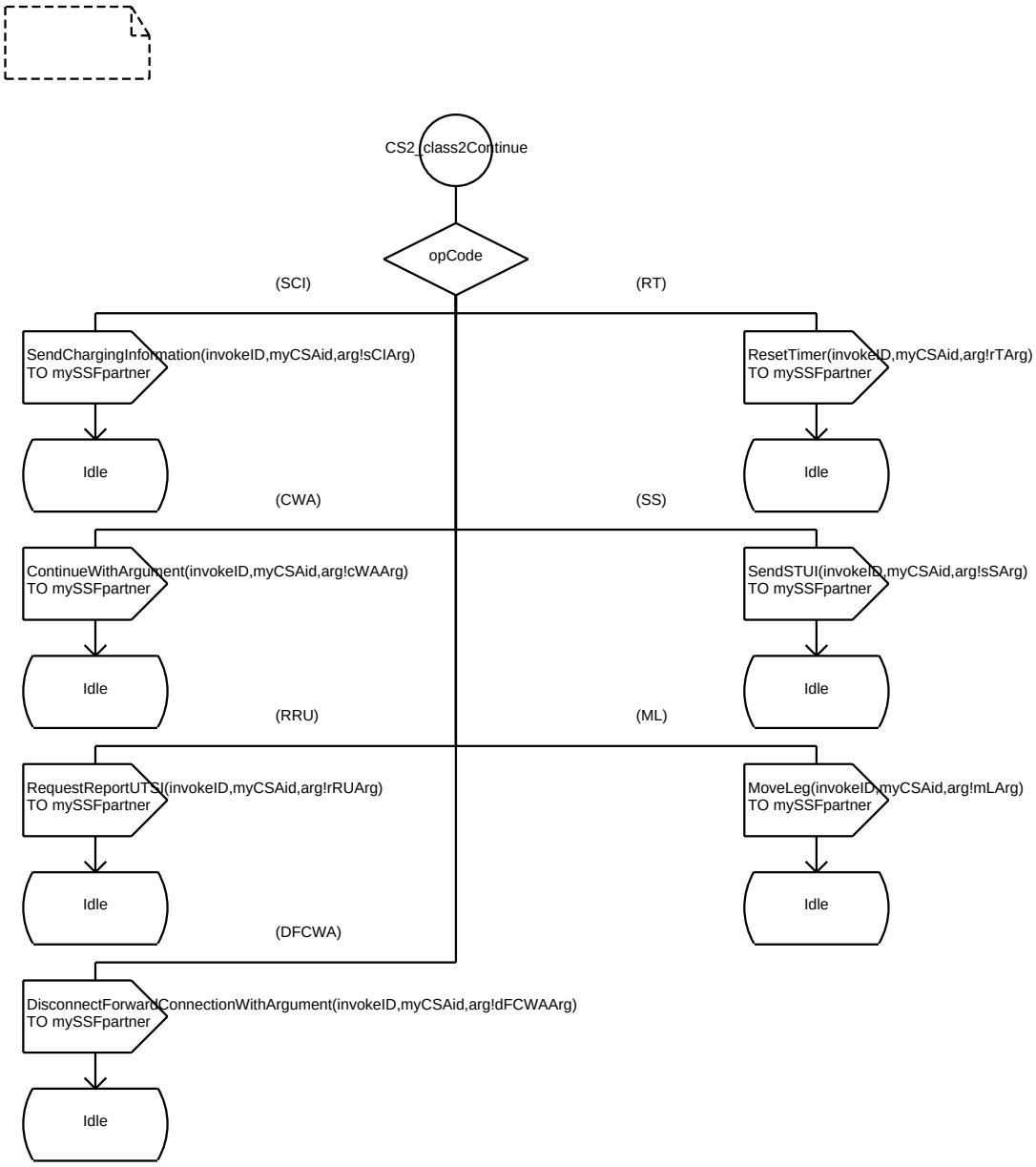

Idle
ManageTriggerDataResult
(invokeID,dialogID,arg!mTD_RArg)
dialogID
(0)
else
CS2_IncorCSAid
ManageTriggerDataResult
(invokeID,dialogID,arg!mTD_RArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
MergeCallSegmentsResult
(invokeID,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
MergeCallSegmentsResult
(invokeID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
CreateCallSegmentAssociationResult
(invokeID,cSAid,arg!cSA_RArg)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
CreateCallSegmentAssociationResult
(invokeID,cSAid,arg!cSA_RArg)
TO allDialogs(dialogID)!dialogHandler
Idle
Idle
DisconnectLegResult
(invokeID,cSAid)
dialogID :=
getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
DisconnectLegResult
(invokeID,cSAid)
TO allDialogs(dialogID)!dialogHandler
Idle

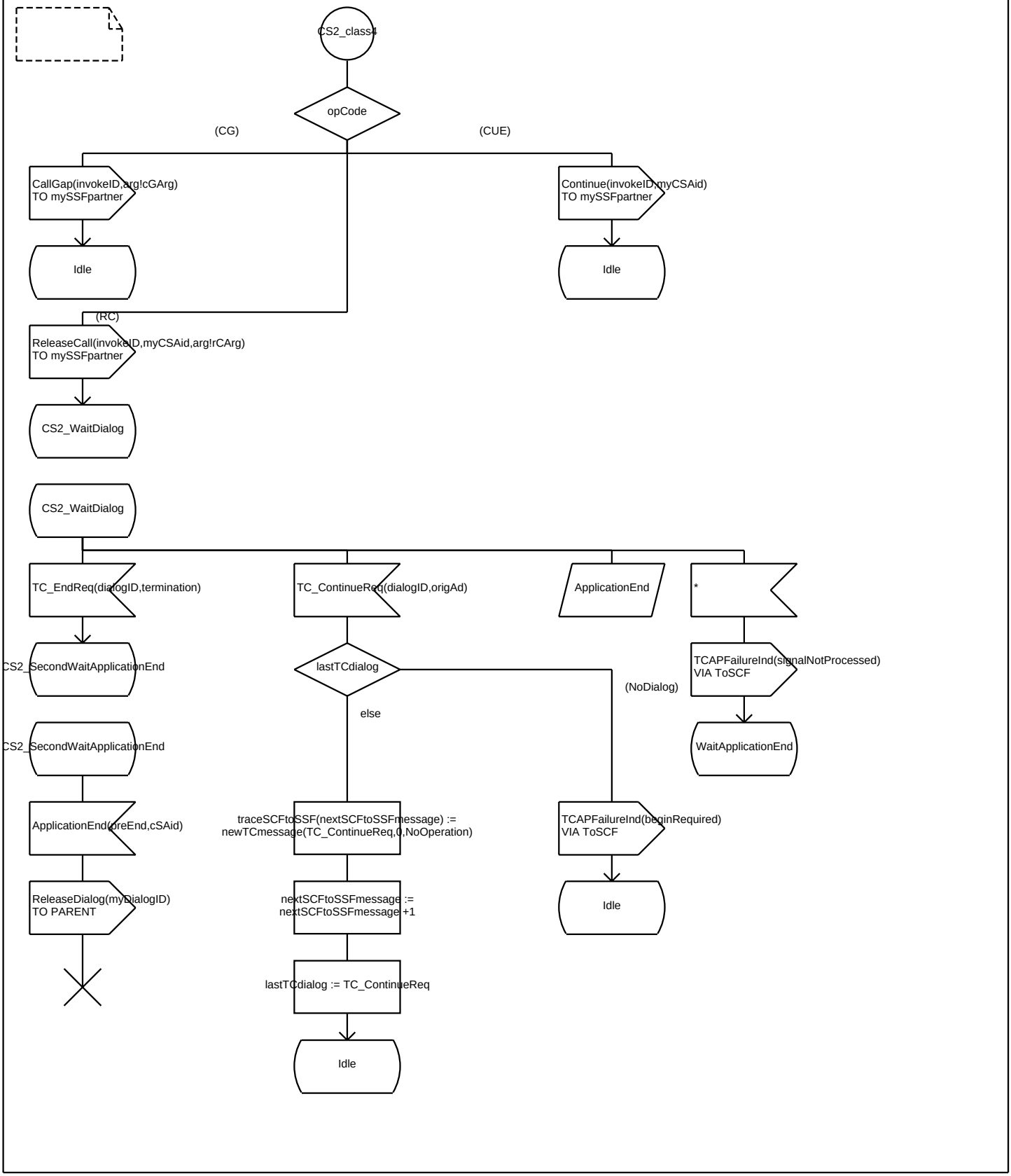

Idle
MoveCallSegmentsResult(invokeID,cSAid)
dialogID := getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
MoveCallSegmentsResult(invokeID,cSAid) TO allDialogs(dialogID)!dialogHandler
Idle
Idle
MoveLegResult(invokeID,cSAid)
dialogID := getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
MoveLegResult(invokeID,cSAid) TO allDialogs(dialogID)!dialogHandler
Idle
Idle
SplitLegResult(invokeID,cSAid)
dialogID := getDialogID(cSAid,allDialogs)
dialogID
(0)
else
CS2_IncorCSAid
SplitLegResult(invokeID,cSAid) TO allDialogs(dialogID)!dialogHandler
Idle
CS2_IncorCSAid
TCAPFailureInd(wrongCSaID)
Idle

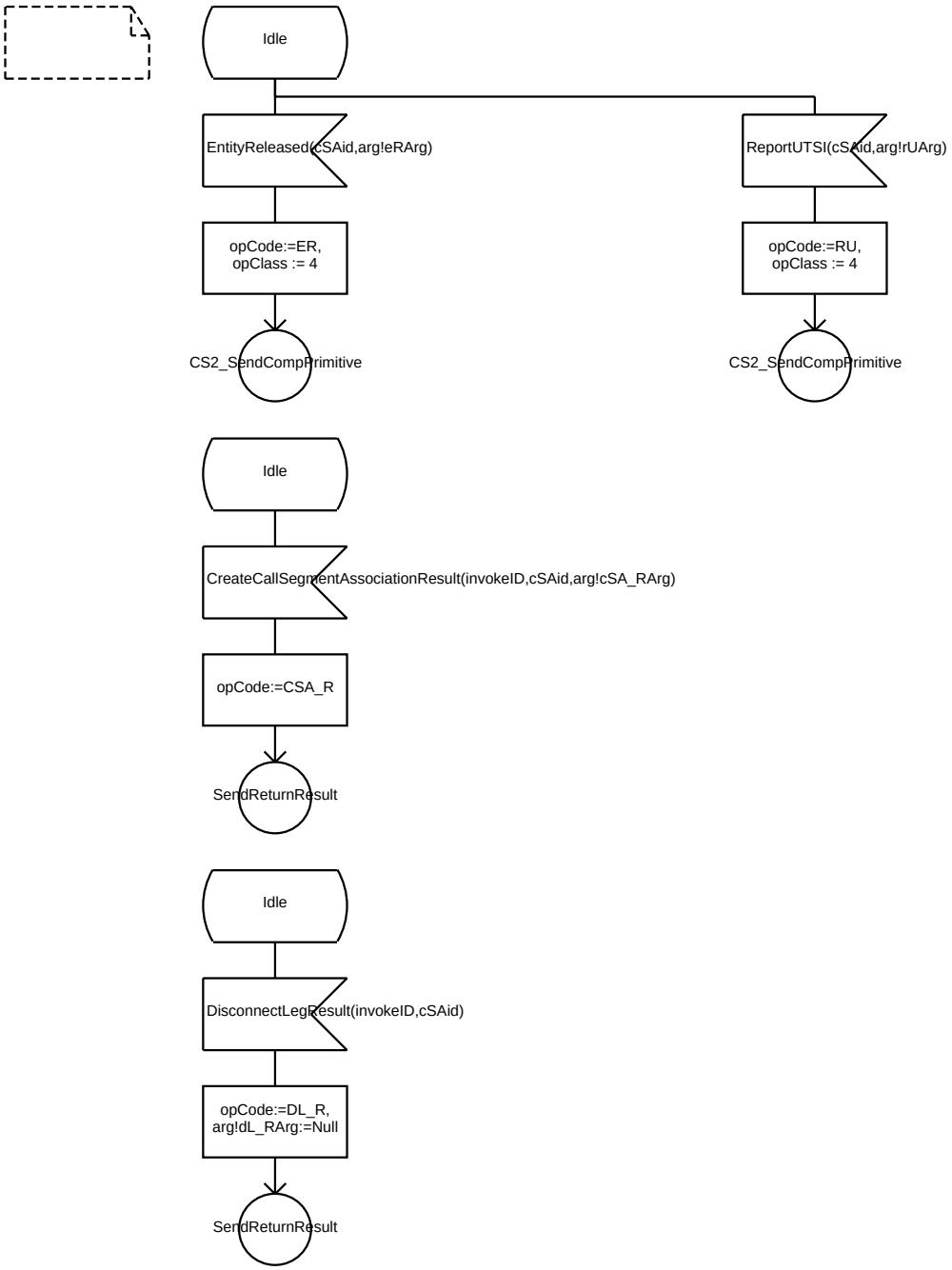

ConIH
(CS2_INAP_To_SCF)
Redefined Process Type TCAP_Dialog
1(7)
ToSSFA
(CS2_INAP_
_From_SCF)
ToSSFB
(CS2_INAP_
_From_SCF)
Idle
redefined
TC_InvokeReq(invokeID,dialogID,opClass,opCode,timeoutVal,arg)
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_InvokeReq,invokeID,opCode)
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
opCode
(ASF, CSA,
DL, MTD, MC,
MCS, SL, ML)
(AC, CIRQ, CAN,
CI, CON, CTR,
DFC, ETC,
FCI, ICA,
RNC, RRB, RT,
SCI, CWA, DFCWA,
RRU, SS)
(AT)
(CG,
CUE,
RC)
else
TCAPFailureInd(unknownOP)
/* Unknown op code */
CS2_class1
CS2_class2
CS2_class3
CS2_class4
Idle
CS2_class3
ActivityTest(invokeID,dialogID)
TO mySSFpartner
Idle

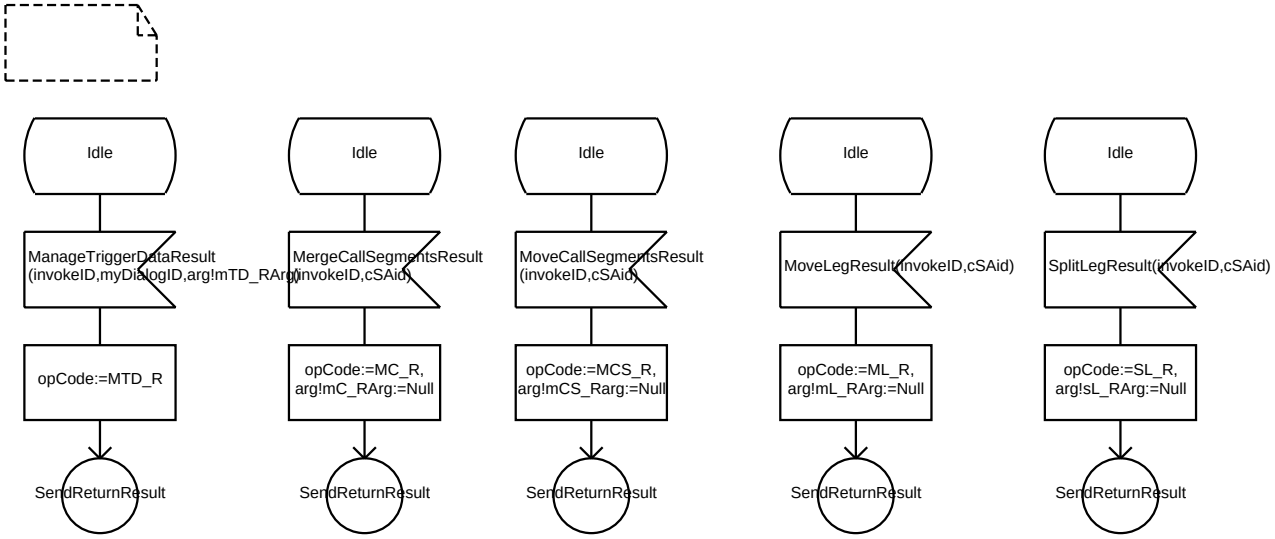

CS2_class1
opCode
(ASF)
(DL)
ActivateServiceFiltering(invokeID,arg!aSFArg)
TO mySSFpartner
DisconnectLeg(invokeID,myCSAid,arg!dLArg)
TO mySSFpartner
Idle
Idle
(CSA)
CreateCallSegmentAssociation(invokeID,myCSAid,arg!cSAArg)
TO mySSFpartner
Idle
(MTD)
(ML)
ManageTriggerData(invokeID,dialogID,arg!mTDArg)
TO mySSFpartner
MoveLeg(invokeID,myCSAid,arg!mLArg)
TO mySSFpartner
Idle
Idle
(MC)
(SL)
MergeCallSegments(invokeID,myCSAid,arg!mCArg)
TO mySSFpartner
SplitLeg(invokeID,myCSAid,arg!sLArg)
TO mySSFpartner
Idle
Idle
(MCS)
MoveCallSegments(invokeID,myCSAid,arg!mCSArg)
TO mySSFpartner
Idle

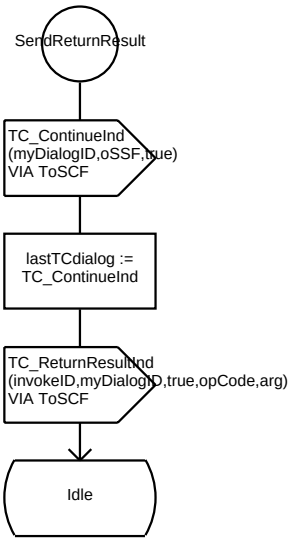

CS2_class2
opCode
(AC)
(CAN)
ApplyCharging(invokeID,myCSAid,arg!aCArg)
TO mySSFpartner
Cancel(invokeID,myCSAid,arg!cANArg)
TO mySSFpartner
Idle
Idle
(CI)
(CIRQ)
CollectInformation(invokeID,myCSAid,arg!cIArg)
TO mySSFpartner
CallInformationRequest(invokeID,myCSAid,arg!cIRQArg)
TO mySSFpartner
Idle
Idle
(CON)
(CTR)
Connnect(invokeID,myCSAid,arg!cONArg)
TO mySSFpartner
ConnectToResource(invokeID,myCSAid,arg!cTRArg)
TO mySSFpartner
Idle
Idle
(DFC)
(ETC)
DisconnectForwardConnection(invokeID,myCSAid)
TO mySSFpartner
EstablishTemporaryConnection(invokeID,myCSAid,arg!eTCArg)
TO mySSFpartner
Idle
Idle
(FCI)
(ICA)
FurnishChargingInformation(invokeID,myCSAid,arg!fCIArg)
TO mySSFpartner
InitiateCallAttempt(invokeID,myCSAid,arg!iCAArg)
TO mySSFpartner
Idle
Idle
(RNC)
(RRB)
RequestNotificationChargingEvent(invokeID,myCSAid,arg!rNCArg)
TO mySSFpartner
RequestReportBCSMEvent(invokeID,myCSAid,arg!rRBArg)
TO mySSFpartner
else
Idle
CS2_class2Continue
Idle

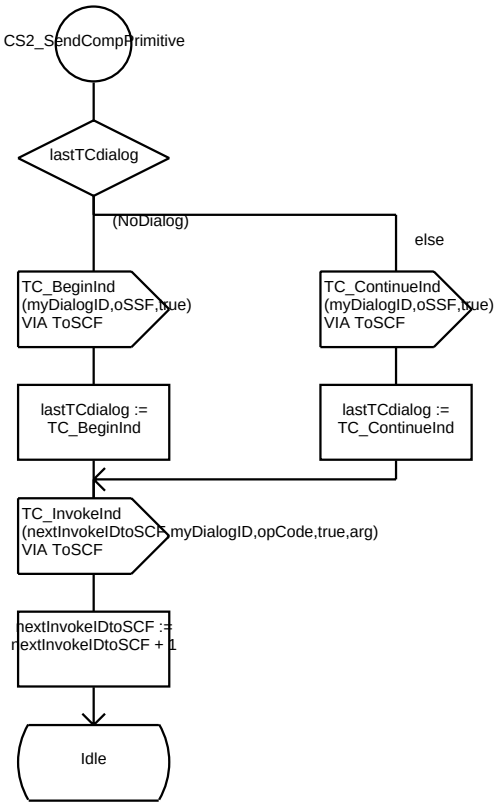

CS2_class2Continue
opCode
(SCI)
(RT)
SendChargingInformation(invokeID,myCSAid,arg!sCIArg)
TO mySSFpartner
ResetTimer(invokeID,myCSAid,arg!rTArg)
TO mySSFpartner
Idle
Idle
(CWA)
(SS)
ContinueWithArgument(invokeID,myCSAid,arg!cWAArg)
TO mySSFpartner
SendSTUI(invokeID,myCSAid,arg!sSArg)
TO mySSFpartner
Idle
Idle
(RRU)
(ML)
RequestReportUTSI(invokeID,myCSAid,arg!rRUArg)
TO mySSFpartner
MoveLeg(invokeID,myCSAid,arg!mLArg)
TO mySSFpartner
Idle
Idle
(DFCWA)
DisconnectForwardConnectionWithArgument(invokeID,myCSAid,arg!dFCWAArg)
TO mySSFpartner
Idle


CS2_class4
opCode
(CG)
(CUE)
CallGap(invokeID,arg!cGArg)
TO mySSFpartner
Continue(invokeID,myCSAid)
TO mySSFpartner
Idle
Idle
(RC)
ReleaseCall(invokeID,myCSAid,arg!rCArg)
TO mySSFpartner
CS2_WaitDialog
CS2_WaitDialog
TC_EndReq(dialogID,termination)
TC_ContinueReq(dialogID,origAd)
ApplicationEnd
*
CS2_SecondWaitApplicationEnd
lastTCdialog
TCAPFailureInd(signalNotProcessed)
VIA ToSCF
(NoDialog)
else
CS2_SecondWaitApplicationEnd
WaitApplicationEnd
ApplicationEnd(preEnd,cSAid)
traceSCFtoSSF(nextSCFtoSSFmessage) :=
newTCmessage(TC_ContinueReq,0,NoOperation)
TCAPFailureInd(beginRequired)
VIA ToSCF
ReleaseDialog(myDialogID)
TO PARENT
nextSCFtoSSFmessage :=
nextSCFtoSSFmessage +1
Idle
lastTCdialog := TC_ContinueReq
Idle


Idle
EntityReleased(cSAid,arg!eRArg)
ReportUTSI(cSAid,arg!rUArg)
opCode:=ER,
opClass := 4
opCode:=RU,
opClass := 4
CS2_SendCompPrimitive
CS2_SendCompPrimitive
Idle
CreateCallSegmentAssociationResult(invokeID,cSAid,arg!cSA_RArg)
opCode:=CSA_R
SendReturnResult
Idle
DisconnectLegResult(invokeID,cSAid)
opCode:=DL_R,
arg!dL_RArg:=Null
SendReturnResult


Idle
ManageTriggerDataResult (invokeID,myDialogID,arg!mTD_RArg)
opCode:=MTD_R
SendReturnResult
Idle
MergeCallSegmentsResult (invokeID,cSAid)
opCode:=MC_R, arg!mC_RArg:=Null
SendReturnResult
Idle
MoveCallSegmentsResult (invokeID,cSAid)
opCode:=MCS_R, arg!mCS_Rarg:=Null
SendReturnResult
Idle
MoveLegResult(invokeID,cSAid)
opCode:=ML_R, arg!mL_RArg:=Null
SendReturnResult
Idle
SplitLegResult(invokeID,cSAid)
opCode:=SL_R, arg!sL_RArg:=Null
SendReturnResult


SendReturnResult
TC_ContinueInd (myDialogID,oSSF,true) VIA ToSCF
lastTCdialog := TC_ContinueInd
TC_ReturnResultInd (invokeID,myDialogID,true,opCode,arg) VIA ToSCF
Idle


CS2_SendCompPrimitive
lastTCdialog
(NoDialog)
else
TC_BeginInd (myDialogID,oSSF,true) VIA ToSCF
TC_ContinueInd (myDialogID,oSSF,true) VIA ToSCF
lastTCdialog := TC_BeginInd
lastTCdialog := TC_ContinueInd
TC_InvokeInd (nextInvokeIDtoSCF,myDialogID,opCode,true,arg) VIA ToSCF
nextInvokeIDtoSCF := nextInvokeIDtoSCF + 1
Idle