Author: Pasi Nummisalo

Document: Call Control

Date: 1/18/99

Version: 1.0

History:

Call Control

Call Control module is the top layer of TOVE ATM signaling protocol stack providing call and service control.

1 Introduction

This implementation is based on ITU-T Q.1214 [1] and Q.1224 [2] standards.

1.1 Standard architecture

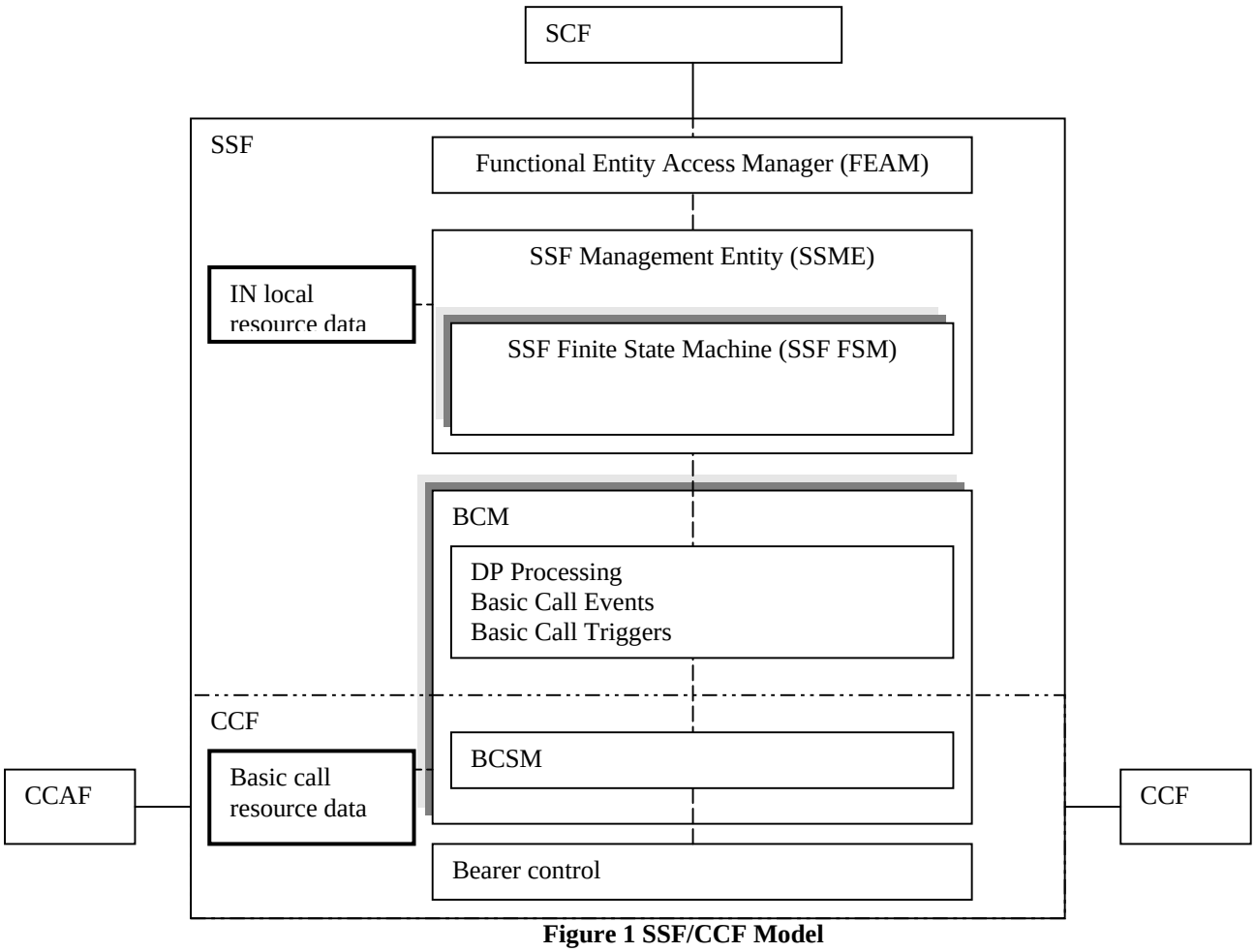


Figure 1 SSF/CCF Model

1.2 CCAF

The CCAF is the call control agent function that provides access for users. It is the interface between user and network call control functions [2].

1.3 CCF

The CCF is the call control function in the network that provides call/service processing and control [2].

1.3.1 BCM/BCSM

The Basic Call State Model (BCSM) provides a high-level model description of CCF activities required to establish and maintain communication paths for users. The BCSM identifies points in basic call and connection processing when IN service logic instances are permitted to interact with basic call and connection control capabilities [2].

Different components that have been identified to describe a BCSM, to include: points in call (PICs), detection points (DPs), transitions, and events. PICs identify CCF activities associated with one or more basic call/connection states of interest to IN service logic instances. DPs indicate points in basic call and connection processing at which transfer of control can occur. Transitions indicate the normal flow of basic call/connection processing from one PIC to another. Events cause transitions into and out of PICs [2].

BCSM is functionally separated between the origination (O-BCSM) and terminating (T-BCSM) portions of calls. Each of the parts is managed by a functionally separate BCM in the SSF/CCF [2].

BCSM states, transitions and indications are illustrated at appendix 1 and 2.

1.4 SSF

The SSF is the service switching function, which, associated with the CCF, provides the set of functions required for interaction between the CCF and a service control function (SCF) [2].

1.4.1 FEAM

The Functional Entity Access Manager (FEAM) is used to establish and maintain the interfaces to the SCF and SRF [1].

1.4.2 SSME

The SSF Management Entity (SSME) maintains the dialogues with the SCF, and SRF on behalf of all instances of the SSF finite state machine (SSF FSM). These instances of the SSF FSM occur concurrently and asynchronously as calls occur, which explains the need

for a single entity that performs the task of creation, invocation, and maintenance of the SSF FSM [1].

1.4.3 SSF FSM

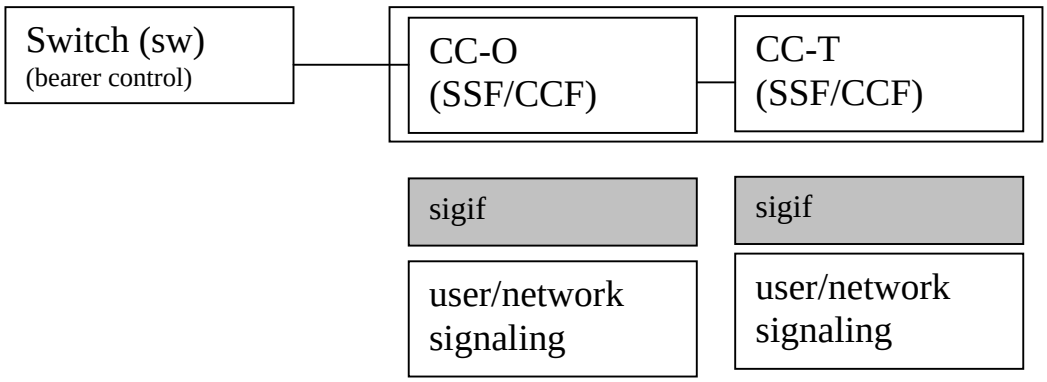
The SSF FSM passes call handling instructions to the related instances of the BCSM as needed. An armed TDP in the BCSM may result a new instance of the SSF FSM for the call. DPs may be dynamically armed by SCF as Event DPs, requiring the SSF FSM to remain active. At some point, further interaction with the SCF is not needed, and the SSF FSM may be terminated while BCSM continues to handle the call [1].

1.5 SCF

The SCF is a function that commands call control functions in the processing of IN provided and/or custom service requests [2].

2. Architecture

This module (CC) implements CCF/SSF functionality. The CCAF function is located at same physical location (same protocol stack) than CC and includes UNI or DSS2 user access signaling protocol. In this implementation common signaling interface (sigif) is used between CCAF and CC. Same sigif interface is also used between network signaling and CC modules. It should be possible to connect CC to different user and network signaling protocols (this feature needs further study).



Module:	CC
Lower interface:	sigif
Lower module:	uni, dss2, bisup
Uses:	Switch (sw)

Table 1 Architecture

3 Implementation details and features implemented

3.1 Protocol

Class name: ccProtocol

Relation to standard: CCF/SSF

The protocol is the main class of CC implementation. A call instance factory is used to create new instance of the protocol for every call. The protocol is initialized with a state structure, which makes it an originating side or a terminating side protocol. Initialization is made based on direction of first incoming message (from lower layer or from originating side). The protocol class serves also as data manager and data storage.

3.2 Acknowledge object

Not in use.

3.3 States

Class naming pattern: ccOstateXx, ccOdpXx, ccTstateXx, ccTdpXx where Xx is the name of state or detection point.

Relation to standard: BCM/SSF FSM

There are two kinds of states: Point In Call states (PIC) and Detection Point states (DP). Both state structures are derived from interface classes sigUpInputs, ccInputs and binapInputs, which present possible messengers and their inputs. The sigUpInputs interface presents messengers coming from a down layer (sigif). The ccInputs interface present internal messages between O and T sides and the binapInputs presents messages from the SCF (INAP protocol). Class ccOstate is also derived from swCallbackInputs interface, which is used to receive bearer control messages from the FCF (Fabric Control Function). CcInput/ccIndications interface includes following messages: setup, alerting, release and release active. See appendix C and D for BCSM states, detection points and messages.

The SSF FSM is embedded in the state structure. When a trigger condition is met call handling is suspended at the current DP state and an application timer (Tssf) is set on (SSF FSM: "waiting for instructions"). Logical "monitoring" state is entered when one or more EDPs are armed and upon receiving a "select route" or a "continue" operation.

3.4 INAP message adapter

Class name: ccInAdapter

Relation to standard: FEAM/SSME

The INAP (Intelligent Network Application Protocol) message adapter translates CORBA requests originating from SCF to INAP-messengers. The INAP adapter implements toveinap_SSF_SCF_initiator interface which is an IDL version of the INAP protocol (direction SCF -> SSF). Implemented INAP messages are: "initialDP", "event report BCSM", "request report BCSM event", "select route", "connect", "release call", and "continue". Appendix B contains the INAP IDL file.

3.5 Cross connector mux

Class name: ccCrossConnectorMux

There is only one Cross Connector Mux (CCM) in the whole system. In the connection establishment state the CCM is used to route messengers to a correct link.

3.7 Detection point data objects

Class name: ccDp

Relation to standard: IN local resource data

Every instance of the protocol has its own DP Data Objects. Initial values for DP data objects (statically armed TDPs) are cloned from ccManagementAdapter class. The Management adapter contains interface for TDP assigment/management. DP data objects hold specific run time information of DPs: type and assigned triggers with criteria. Four DP types are identified and implemented: Trigger Detection Point - Request (TDP-R), Trigger Detection Point - Notification (TDP-R), Event Detection Point - Request (EDP-R), Event Detection Point - Notification (EDP-N). TDP criteria is in this implementation always office based (applies to all calls). Following criteria are applicable for a given TDP: trigger assigned and specific digit string. DP criteria for EDPs are addressed by the RequestReportBCSMEvent information flow.

3.8 Trigger framework

Class naming pattern: ccTriggerXx

Each Detection Point Data Object can hold unlimited number of triggers. Triggers form a TDP criteria, which are conditions that must be met in order to notify the SCF that the TDP was encountered. Specific trigger classes are derived from a ccTriggerBase class, which implements a ccTriggerInterface class.

3.9 Management adapter

Class name: ccManagementAdapter

Relation to standard: SMF-SSF interface

The management adapter implements a ccif IDL interface (see appendix A). This interface can be used to arm DPs with appropriate triggers with criteria (static arming, same for all call instances).

4 Features implemented

Basic call events (messages from signaling layer), basic call state model for originating and terminating sides, fabric control, detection point processing including EDP/TDP activation and deactivation, trigger framework, INAP and management interfaces. Logical three state SSF FSM with states: idle, trigger processing, waiting for instructions, and monitoring.

4.1 Point-to-multipoint

The cc protocol supports point-to-multipoint functionality when a lower level signaling protocol is point-to-multipoint capable and follows certain rules (e.g. TOVE UNI 3.1, UNI 4.0).

4.1.1 Root initiated point-to-multipoint call

Every multipoint party (leaf) has an own O-BCSM (ccProtocol). The first O-BCSM instance (root) is created with static method ccProtocol::create(). Next instance must have reference to the root O-BCSM and can be created with same static method ccProtocol::create(root) with the root O-BCSM as parameter. This instance can be used as a prototype for following leafs. The point-to-multipoint operation is enabled when a setup primitive has a "end point reference" information element assigned. The cc protocol accepts only point-to-point primitives so the lower layer signaling protocol must convert e.g. a "add party" primitive to a setup with the "end point reference" information element.

5 Known limitations

Call segments can't be joined or split. Multiple concurrent instances of IN service logic instance on a single call are not allowed. No end user interaction and SRF connection functionality. States missing from SSF FSM: waiting for end of user interaction, waiting for end of temporary connection. No state machine for SSME and no call gapping or service filtering mechanism. States and detection points marked with [] in appendix C and D don't have functionality.

6 Future development

Multi point functionality and connection to routing module needs further development. More INAP messages should be implemented: authorize origination, authorize termination, collect information etc. Changes derived from OMG CORBA/IN standardization process should be implemented.

7 References

[1] ITU-T Recommendation Q.1214 CS-1

[2] ITU-T Recommendation Q.1224 CS-2 Draft, Miyasaki output, January 1996

[3] Brennan, Rob (Editor). Interworking Between CORBA and Intelligent Networks System, Version 2.0 Revised RFP Submission. OMG. 1998. Telecom/98-08-11.

Appendix A, IDL interface for CC management

```
#ifndef CCIF_IDL
#define CCIF_IDL
#include "../inap/toveinap.idl"
#pragma prefix "idl.tove"

// CC management interface module (SMP-SSP)
// The management interface is used for TDP assigment/management.

module ccIf
{
   enum Side
   {
      originating,
          terminating
   };

   struct TriggerType
   {
       string type;
       string criteria;
       string category;
       string address;
       string applicationContext;
       toveinap::ServiceKeyType serviceKey;
   };

   typedef sequence<TriggerType> TriggersType;

   struct TDPtype
   {
          string              type; // DP name
          string              messageType; // Request, notification
          TriggersType     setTriggers;
          TriggersType     allowedTriggers;
   };

   typedef sequence<TDPtype> TDPsType;

   // SSP management interface
   interface managementServer
   {
          // Set DPs message type (request, notification)
      void setMessageType(in Side side, in string dpType,
         in string messageType);

          // Add trigger to DP
          void addTrigger(in Side side, in string dpType,
             in TriggerType newTrigger);

          // Remove trigger from DP
          void deleteTrigger(in Side side, in string dpType,
             in TriggerType oldTrigger);

          // Read TDP type, message type, set triggers and allowed triggers
          void requestTDPreport(in Side side, out TDPsType TDPs);

   };

};
#endif // CCIF_IDL
```

Appendix B, IDL interface for INAP

```
#ifndef INAP_IDL
#define INAP_IDL

#include "../TcSignaling/TcSignaling.idl"

#pragma prefix "idl.tove"

module toveinap
{
    // Data types ----------------------------------------------------------
    // Created from ASN.1 module:
    // IN-CS2-datatypes
    // {ccitt recommendation q 1228 modules(0) in-cs2-datatypes (0)
version1(0)}

    typedef ASN1_Integer minCalledPartyNumberLengthType;
    typedef ASN1_Integer maxCalledPartyNumberLengthType;
    typedef ASN1_Integer minCallingPartyNumberLengthType;
    typedef ASN1_Integer maxCallingPartyNumberLengthType;

    const minCalledPartyNumberLengthType minCalledPartyNumberLength = 5;
    const maxCalledPartyNumberLengthType maxCalledPartyNumberLength = 24;
    const minCallingPartyNumberLengthType minCallingPartyNumberLength = 5;
    const maxCallingPartyNumberLengthType maxCallingPartyNumberLength = 24;

    typedef sequence<octet,1> CallingPartysCategoryType;
    // Indicates the type of calling party (e.g. operator, payphone, ordinary
    // subscriber). Refer to Q.763 for encoding.

    union CallingPartysCategoryTypeOpt switch(boolean)
    {
        case TRUE: CallingPartysCategoryType callingPartysCategory;
    };

    typedef sequence<octet,maxCalledPartyNumberLength> CalledPartyNumberType;
    // Indicates the Called Party Number. Refer to Q.763 for encoding.

    union CalledPartyNumberTypeOpt switch(boolean)
    {
        case TRUE: CalledPartyNumberType calledPartyNumber;
    };

    typedef sequence<octet,maxCallingPartyNumberLength> CallingPartyNumberType;

    union CallingPartyNumberTypeOpt switch(boolean)
    {
        case TRUE: CallingPartyNumberType callingPartyNumber;
    };

    typedef sequence<CalledPartyNumberType,3> DestinationRoutingAddressType;
    // Indicates the list of Called Party Numbers (primary and alternates).

    typedef sequence<octet,2> CauseType;
    // Indicates the cause for interface related information. Refer to the
Q.763
    // Cause parameter for encoding. For the use of cause and location values
    // refer to Q.850

    enum TaskRefusedType
```

```
 {
   generic,
   unobtainable,
   congestion
 };

 enum EventTypeBCSMType
 {
     origAttemptAuthorized,
     collectedInfo,
     analysedInformation,
     routeSelectFailure,
     oCalledPartyBusy,
     oNoAnswer,
     oAnswer,
     oMidCall,
     oDisconnect,
     oAbandon,
     termAttemptAuthorized,
     tBusy,
     tNoAnswer,
     tAnswer,
     tMidCall,
     tDisconnect,
     tAbandon,
     oTermSeized,
     oSuspended,
     tSuspended
};
// Indicates the BCSM detection point event. Refer to 4.2.2.2/Q.1214 for
// additional information on the events.

union EventTypeBCSMTypeOpt switch(boolean)
{
   case TRUE: EventTypeBCSMType eventTypeBCSM;
};

enum MonitorModeType
{
   interrupted,
   notifyAndContinue,
   transparent
};
// Indicates the event is relayed and/or processed by the SSP.

struct BCSMEventType
{
   EventTypeBCSMType    eventTypeBCSM;
   MonitorModeType      monitorMode;
};
// Indicates the BCSM Event information for monitoring.

typedef sequence<BCSMEventType, 24> BCSMEventsType;

enum UnavailableNetworkResourceType
{
   unavailableResources,
   componentFailure,
   basicCallProcessingException,
   resourceStatusFailure,
   endUserFailure
};
// Indicates the network resource that failed.
```

```
typedef ASN1_Integer ServiceKeyType;
// Information that allows the SCF to choose the appropriate service logic.

enum MessageTypeType
{
   request,
   notification
};

enum DPAssignmentType
{
   individualLine,
   groupBased,
   officeBased
};

struct MiscCallInfoType
{
   MessageTypeType      messageType;
   DPAssignmentType     dpAssignment;
};
// Indicates detection point related information.

union MiscCallInfoTypeOpt switch(boolean)
{
   case TRUE: MiscCallInfoType miscCallInfo;
};

enum TriggerTypeType
{
   trigger_featureActivation,
   trigger_verticalServiceCode,
   trigger_customizedAccess,
   trigger_customizedIntercom,
   trigger_emergencyService,
   trigger_aFR,
   trigger_sharedIOTrunk,
   trigger_offHookDelay,
   trigger_channelSetupPRI,
   trigger_tNoAnswer,
   trigger_tBusy,
   trigger_oCalledPartyBusy,
   trigger_oNoAnswer,
   trigger_originationAttemptAuthorized,
   trigger_oAnswer,
   trigger_oDisconnect,
   trigger_termAttemptAuthorized,
   trigger_tAnswer,
   trigger_tDisconnect
};
// The type of trigger which caused call suspension.

union TriggerTypeTypeOpt switch(boolean)
{
   case TRUE: TriggerTypeType triggerType;
};

// Operation arguments ---------------------------------------------------
// Created from ASN.1 module:
// IN-CS2-SSF-SCF-ops-args
// {ccitt recommendation q 1228 modules(0) in-cs2-ssf-scf-ops-args (5)
//  version1(0)}
```

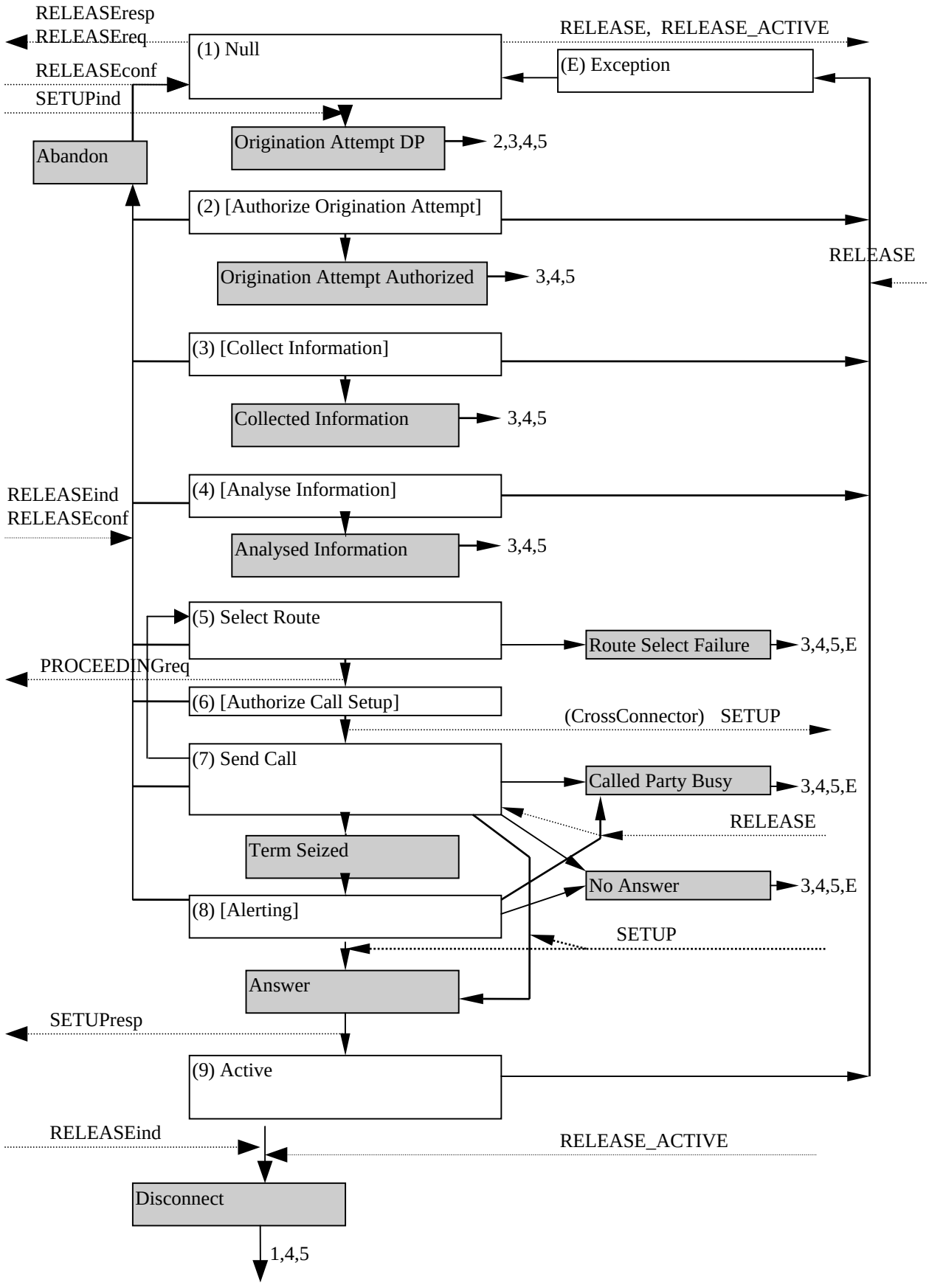
```
   struct ConnectArgType
   {
      DestinationRoutingAddressType destinationRoutingAddress;
   };

   struct RequestReportBCSMEventArgType
   {
      BCSMEventsType bcsmEvents;
   };

   struct SelectRouteArgType
   {
      DestinationRoutingAddressType destinationRoutingAddress;
   };

   struct ReleaseCallArgType
   {
      CauseType cause;
   };

   struct InitialDPArgType
   {
      ServiceKeyType                   serviceKey;
      CalledPartyNumberTypeOpt          calledPartyNumber;
      CallingPartyNumberTypeOpt     callingPartyNumber;
      CallingPartysCategoryTypeOpt      callingPartysCategory;
      MiscCallInfoTypeOpt           miscCallInfo;
      TriggerTypeTypeOpt            triggerType;
      EventTypeBCSMTypeOpt          eventTypeBCSM;
   };

   struct EventReportBCSMArgType
   {
      EventTypeBCSMType     eventTypeBCSM;
      MiscCallInfoType      miscCallInfo;
   };

   // Exceptions ------------------------------------------------------------
   // Created from ASN.1 module:
   // IN-CS2-errortypes
   // {ccitt recommendation q 1228 modules(0) in-cs2-errortypes (1)
version1(0)}

   exception missingCustomerRecord {TcSignaling::DialogFlowCtr dfc;};
   // The Service Logic Program could not be found in the SCF.
   // Error code: localValue : 6

   exception missingParameter {TcSignaling::DialogFlowCtr dfc;};
   // An expected optional parameter was not received.
   // Error code: localValue : 7

   exception parameterOutOfRange {TcSignaling::DialogFlowCtr dfc;};
   // The parameter was not as expected (e.g. missing or out of range).
   // Error code: localValue : 8

   exception systemFailure
   {
      UnavailableNetworkResourceType error_param;
      TcSignaling::DialogFlowCtr dfc;
   };
   // The operation could not be completed due to a system failure at the
   // serving physical entity.
```

```
   // Error code: localValue : 11

   exception taskRefused
   {
      TaskRefusedType error_param;
      TcSignaling::DialogFlowCtr dfc;
   };
   // An entity normally capable of the task requested cannot or chooses not to
   // perform the task at this time.  This includes error situations like
   // congestion and unobtainable address as used in e.g. the connect
operation.
   // Error code: localValue : 12

   exception unexpectedComponentSequence {TcSignaling::DialogFlowCtr dfc;};
   // An incorrect sequence of Components was received
   // (e.g. "DisconnectForwardConnection" followed by "PlayAnnouncement").
   // Error code: localValue : 14

   exception unexpectedDataValue {TcSignaling::DialogFlowCtr dfc;};
   // The data value was not as expected (e.g. routing number expected but
   // billing number received)
   // Error code: localValue : 15

   exception unexpectedParameter {TcSignaling::DialogFlowCtr dfc;};
   // A parameter received was not expected.
   // Error code: localValue : 16

   // Operations ------------------------------------------------------------
   // Created from ASN.1 module:
   // IN-CS2-SSF-SCF-ops-args
   // {ccitt recommendation q 1228 modules(0) in-cs2-ssf-scf-ops-args (5)
   //  version1(0)}

   interface SSF_SCF_initiator : TcSignaling::TcUser
   {
      void connect(in ConnectArgType ConnectArg,
         inout TcSignaling::TcContext ctext)
            raises (missingParameter, parameterOutOfRange, systemFailure,
            taskRefused, unexpectedComponentSequence, unexpectedDataValue,
            unexpectedParameter);
      // This operation is used to request the SSF to perform the call
      // processing actions to route or forward a call to a specified
      // destination. To do so, the SSF may or may not use destination
      // information from the calling party (e.g. dialed digits) and existing
      // call setup information (e.g. route index to a list of -- trunk
groups),
      // depending on the information provided by the SCF.
      // - When address information is only included in the Connect operation,
      //   call processing resumes at PIC3 in  the O-BCSM.
      // - When address information and routing information is included, call
      //   processing resumes at PIC4.
      // Operation code: localValue : 20

      void releaseCall(in ReleaseCallArgType releaseCallArg,
         inout TcSignaling::TcContext ctext);
      // This operation is used to tear down an existing call at any phase of
      // the call for all parties involved in the call.
      // Operation code: localValue : 22

      void requestReportBCSMEvent(in RequestReportBCSMEventArgType
         requestReportBCSMEventArg, inout TcSignaling::TcContext ctext)
            raises (missingParameter, parameterOutOfRange, systemFailure,
            taskRefused, unexpectedComponentSequence, unexpectedDataValue,
```

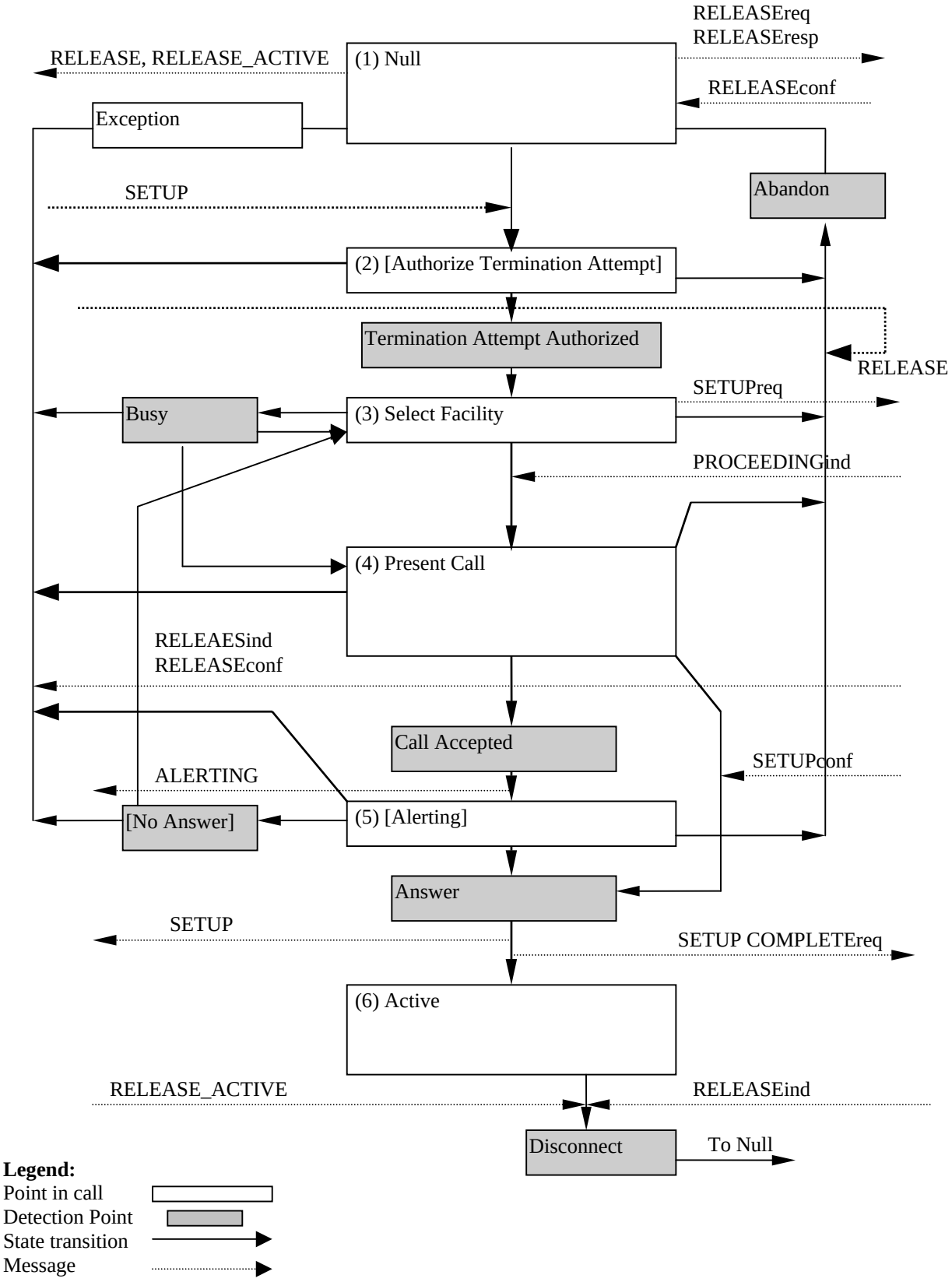
```
            unexpectedParameter);
      // This operation is used to request the SSF to monitor for a call-
related
      // event (e.g. BCSM events such as busy or no answer), then send a
      // notification back to the SCF when the event is detected.
      // Operation code: localValue : 23

      void selectRoute(in SelectRouteArgType selectRouteArg,
         inout TcSignaling::TcContext ctext)
                raises(missingParameter, parameterOutOfRange, systemFailure,
            taskRefused, unexpectedComponentSequence, unexpectedDataValue,
            unexpectedParameter);
      // This operation is used to request the SSF to perform the originating
      // basic call processing actions to determine routing information and
      // select a route for a call, based either on call information available
      // to the SSF, or on call information provided by the SCF
      // (e.g. for alternate routing), to include the called party address,
      // type of call, carrier, route index, and one or more alternate route
      // indices. Based on the routing information, the SSF attempts to select
      // a primary route for the call, and if the route is busy, attempts to
      // select an alternate route. The SSF may fail to select a route for the
      // call if all routes are busy.
      // Operation code: localValue : 29

      void continue(inout TcSignaling::TcContext ctext);
      // This operation is used to request the SSF to proceed with call
      // processing at the DP at which it previously suspended call processing
      // to await SCF instructions (i.e. proceed to the next point in call in
      // the BCSM). The SSF continues call processing without substituting new
      // data from SCF.
      // Operation code: localValue : 31
   };

   interface SSF_SCF_responder : TcSignaling::TcUser
   {
      void initialDP(in InitialDPArgType initialDPArg,
         inout TcSignaling::TcContext ctext)
            raises (missingCustomerRecord, missingParameter,
               parameterOutOfRange, systemFailure, taskRefused,
               unexpectedComponentSequence, unexpectedDataValue,
               unexpectedParameter);
      // This operation is used after a TDP to indicate request for service.
      // Operation code: localValue : 0

      void eventReportBCSM(in EventReportBCSMArgType eventReportBCSMArg,
         inout TcSignaling::TcContext ctext);
      // This operation is used to notify the SCF of a call-related event
      // (e.g. BCSM events such as busy or no answer) previously requested by
      // the SCF in a RequestReportBCSMEvent operation.
      // Operation code: localValue : 24
   };

   // Factory interface ------------------------------------------------------
   // Defines creation operations for application contexts.

   interface TcUserFactory : TcSignaling::TcUserGenericFactory
   {
         SSF_SCF_responder create_SSF_SCF_responder(
            in SSF_SCF_initiator initiator,
            in TcSignaling::AssociationId a_id,
            in TcSignaling::TcContextSetting tc_context_setting)
               raises(TcSignaling::NoMoreAssociations,
                  TcSignaling::UnsupportedTcContext);
```

```
         SSF_SCF_responder create_SSF_SCF_responder_with_dialogdata(
            in SSF_SCF_initiator initiator,
            in TcSignaling::AssociationId a_id,
            in TcSignaling::TcContextSetting tc_context_setting,
            in string protocol_version,
            in TcSignaling::DialogUserData d_u_d)
               raises (TcSignaling::NoMoreAssociations,
                  TcSignaling::InvalidParameter,
                  TcSignaling::UnsupportedTcContext);

         SSF_SCF_initiator create_SSF_SCF_initiator();
   };

}; // module inap

#endif // INAP_IDL
```

Appendix C, set of transitions and indications for the O-BCSM



Appendix D, set of transitions and indications for the T-BCSM



Legend:

Point in call

Detection Point

State transition

Message