MODULE *E2AP*

The *E2AP* module provides a formal specification of the *E2AP* protocol. The spec defines the client and server interfaces for *E2AP* and provides helpers for managing and operating on connections.

LOCAL INSTANCE *Naturals*

LOCAL INSTANCE *Sequences*

LOCAL INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

CONSTANT *Nil*

VARIABLE *conns*

The *E2AP* protocol is implemented on *SCTP*

LOCAL $SCTP \triangleq$ INSTANCE *SCTP*

$vars \triangleq \langle conns \rangle$

MODULE *Cause*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by *E2AP*.

MODULE *Misc*

CONSTANTS
Unspecified,
ControlProcessingOverload,
HardwareFailure,
OMIntervention

$All \triangleq$
{*Unspecified*,
ControlProcessingOverload,
HardwareFailure,
OMIntervention}

ASSUME $\forall\, c \in All : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsControlProcessingOverload(m) \triangleq m.cause = ControlProcessingOverload$
$IsHardwareFailure(m) \triangleq m.cause = HardwareFailure$
$IsOMIntervention(m) \triangleq m.cause = OMIntervention$

$Misc \triangleq$ INSTANCE *Misc* WITH
Unspecified $\leftarrow$ "Unspecified",

ControlProcessingOverload ← "ControlProcessingOverload",
HardwareFailure ← "HardwareFailure",
OMIntervention ← "OMIntervention"

MODULE *Protocol*

CONSTANTS
Unspecified,
TransferSyntaxError,
AbstractSyntaxErrorReject,
AbstractSyntaxErrorIgnoreAndNotify,
MessageNotCompatibleWithReceiverState,
SemanticError,
AbstractSyntaxErrorFalselyConstructedMessage

$All \triangleq$
{*Unspecified*,
TransferSyntaxError,
AbstractSyntaxErrorReject,
AbstractSyntaxErrorIgnoreAndNotify,
MessageNotCompatibleWithReceiverState,
SemanticError,
AbstractSyntaxErrorFalselyConstructedMessage}

ASSUME $\forall\, c \in All : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsTransferSyntaxError(m) \triangleq m.cause = TransferSyntaxError$
$IsAbstractSyntaxErrorReject(m) \triangleq m.cause = AbstractSyntaxErrorReject$
$IsAbstractSyntaxErrorIgnoreAndNotify(m) \triangleq m.cause = AbstractSyntaxErrorIgnoreAndNotify$
$IsMessageNotCompatibleWithReceiverState(m) \triangleq m.cause = MessageNotCompatibleWithReceiverState$
$IsSemanticError(m) \triangleq m.cause = SemanticError$
$IsAbstractSyntaxErrorFalselyConstructedMessage(m) \triangleq m.cause = AbstractSyntaxErrorFalselyConstru$

Protocol ≜ INSTANCE *Protocol* WITH
Unspecified ← "Unspecified",
TransferSyntaxError ← "TransferSyntaxError",
AbstractSyntaxErrorReject ← "AbstractSyntaxErrorReject",
AbstractSyntaxErrorIgnoreAndNotify ← "AbstractSyntaxErrorIgnoreAndNotify",
MessageNotCompatibleWithReceiverState ← "MessageNotCompatibleWithReceiverState",
SemanticError ← "SemanticError",
AbstractSyntaxErrorFalselyConstructedMessage ← "AbstractSyntaxErrorFalselyConstructedMessage"

MODULE *RIC*

CONSTANTS

Unspecified,
RANFunctionIDInvalid,
ActionNotSupported,
ExcessiveActions,
DuplicateAction,
DuplicateEvent,
FunctionResourceLimit,
RequestIDUnknown,
InconsistentActionSubsequentActionSequence,
ControlMessageInvalid,
CallProcessIDInvalid

$All \triangleq$
{*Unspecified*,
RANFunctionIDInvalid,
ActionNotSupported,
ExcessiveActions,
DuplicateAction,
DuplicateEvent,
FunctionResourceLimit,
RequestIDUnknown,
InconsistentActionSubsequentActionSequence,
ControlMessageInvalid,
CallProcessIDInvalid}

ASSUME $\forall c \in All : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsRANFunctionIDInvalid(m) \triangleq m.cause = RANFunctionIDInvalid$
$IsActionNotSupported(m) \triangleq m.cause = ActionNotSupported$
$IsExcessiveActions(m) \triangleq m.cause = ExcessiveActions$
$IsDuplicateAction(m) \triangleq m.cause = DuplicateAction$
$IsDuplicateEvent(m) \triangleq m.cause = DuplicateEvent$
$IsFunctionResourceLimit(m) \triangleq m.cause = FunctionResourceLimit$
$IsRequestIDUnknown(m) \triangleq m.cause = RequestIDUnknown$
$IsInconsistentActionSubsequentActionSequence(m) \triangleq m.cause = InconsistentActionSubsequentActionSe$
$IsControlMessageInvalid(m) \triangleq m.cause = ControlMessageInvalid$
$IsCallProcessIDInvalid(m) \triangleq m.cause = CallProcessIDInvalid$

$RIC \triangleq$ INSTANCE RIC WITH
Unspecified ← "Unspecified",
RANFunctionIDInvalid ← "RANFunctionIDInvalid",
ActionNotSupported ← "ActionNotSupported",
ExcessiveActions ← "ExcessiveActions",

$DuplicateAction \leftarrow$ "DuplicateAction",
$DuplicateEvent \leftarrow$ "DuplicateEvent",
$FunctionResourceLimit \leftarrow$ "FunctionResourceLimit",
$RequestIDUnknown \leftarrow$ "RequestIDUnknown",
$InconsistentActionSubsequentActionSequence \leftarrow$ "InconsistentActionSubsequentActionSequence",
$ControlMessageInvalid \leftarrow$ "ControlMessageInvalid",
$CallProcessIDInvalid \leftarrow$ "CallProcessIDInvalid"

MODULE *RICService*

CONSTANTS
Unspecified,
FunctionNotRequired,
ExcessiveFunctions,
RICResourceLimit

$All \triangleq$
{*Unspecified*,
FunctionNotRequired,
ExcessiveFunctions,
RICResourceLimit}

ASSUME $\forall c \in All : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsFunctionNotRequired(m) \triangleq m.cause = FunctionNotRequired$
$IsExcessiveFunctions(m) \triangleq m.cause = ExcessiveFunctions$
$IsRICResourceLimit(m) \triangleq m.cause = RICResourceLimit$

RICService $\triangleq$ INSTANCE *RICService* WITH
Unspecified $\leftarrow$ "Unspecified",
FunctionNotRequired $\leftarrow$ "FunctionNotRequired",
ExcessiveFunctions $\leftarrow$ "ExcessiveFunctions",
RICResourceLimit $\leftarrow$ "RICResourceLimit"

MODULE *Transport*

CONSTANTS
Unspecified,
TransportResourceUnavailable

$All \triangleq$
{*Unspecified*,
TransportResourceUnavailable}

ASSUME $\forall c \in All : c \in$ STRING

$IsUnspecified(m) \triangleq m.cause = Unspecified$
$IsTransportResourceUnavailable(m) \triangleq m.cause = TransportResourceUnavailable$

$Transport \triangleq$ INSTANCE $Transport$ WITH
$Unspecified \leftarrow$ "Unspecified",
$TransportResourceUnavailable \leftarrow$ "TransportResourceUnavailable"

$All \triangleq Misc!All \cup Protocol!All \cup RIC!All \cup RICService!All \cup Transport!All$

$IsCause(c) \triangleq c \in All$

This section defines predicates for identifying $E2AP$ message types on the network.

The *Cause* module provides failure causes
$Cause \triangleq$ INSTANCE $Cause$

MODULE *Messages*

The *Messages* module defines predicates for receiving, sending, and verifying all the messages supported by $E2AP$.

Message type constants
CONSTANTS
$E2SetupRequest$,
$E2SetupResponse$,
$E2SetupFailure$
CONSTANTS
$RICServiceUpdate$,
$RICServiceUpdateAcknowledge$,
$RICServiceUpdateFailure$
CONSTANTS
$ResetRequest$,
$ResetResponse$
CONSTANTS
$RICSubscriptionRequest$,
$RICSubscriptionResponse$,
$RICSubscriptionFailure$
CONSTANTS
$RICSubscriptionDeleteRequest$,
$RICSubscriptionDeleteResponse$,
$RICSubscriptionDeleteFailure$
CONSTANTS
$RICIndication$
CONSTANTS
$RICControlRequest$,

RICControlResponse,
RICControlFailure

CONSTANTS
E2ConnectionUpdate,
E2ConnectionUpdateAcknowledge,
E2ConnectionUpdateFailure

CONSTANTS
E2NodeConfigurationUpdate,
E2NodeConfigurationUpdateAcknowledge,
E2NodeConfigurationUpdateFailure

LOCAL $messageTypes \triangleq$
$\{E2SetupRequest$,
E2SetupResponse,
E2SetupFailure,
RICServiceUpdate,
RICServiceUpdateAcknowledge,
RICServiceUpdateFailure,
ResetRequest,
ResetResponse,
RICSubscriptionRequest,
RICSubscriptionResponse,
RICSubscriptionFailure,
RICSubscriptionDeleteRequest,
RICSubscriptionDeleteResponse,
RICSubscriptionDeleteFailure,
RICControlRequest,
RICControlResponse,
RICControlFailure,
RICServiceUpdate,
E2ConnectionUpdate,
E2ConnectionUpdateAcknowledge,
E2ConnectionUpdateFailure,
E2NodeConfigurationUpdate,
E2NodeConfigurationUpdateAcknowledge,
E2NodeConfigurationUpdateFailure$\}$

Message types should be defined as strings to simplify debugging
ASSUME $\forall\, m \in messageTypes : m \in$ STRING

This section defines predicates for identifying *E2AP* message types on the network.

$IsE2SetupRequest(msg) \triangleq msg.type = E2SetupRequest$

$IsE2SetupResponse(msg) \triangleq msg.type = E2SetupResponse$

$IsE2SetupFailure(msg) \triangleq msg.type = E2SetupFailure$

$IsRICServiceUpdate(msg) \triangleq msg.type = RICServiceUpdate$

$IsRICServiceUpdateAcknowledge(msg) \triangleq msg.type = RICServiceUpdateAcknowledge$

$IsRICServiceUpdateFailure(msg) \triangleq msg.type = RICServiceUpdateFailure$

$IsResetRequest(msg) \triangleq msg.type = ResetRequest$

$IsResetResponse(msg) \triangleq msg.type = ResetResponse$

$IsRICSubscriptionRequest(msg) \triangleq msg.type = RICSubscriptionRequest$

$IsRICSubscriptionResponse(msg) \triangleq msg.type = RICSubscriptionResponse$

$IsRICSubscriptionFailure(msg) \triangleq msg.type = RICSubscriptionFailure$

$IsRICSubscriptionDeleteRequest(msg) \triangleq msg.type = RICSubscriptionDeleteRequest$

$IsRICSubscriptionDeleteResponse(msg) \triangleq msg.type = RICSubscriptionDeleteResponse$

$IsRICSubscriptionDeleteFailure(msg) \triangleq msg.type = RICSubscriptionDeleteFailure$

$IsRICIndication(msg) \triangleq msg.type = RICIndication$

$IsRICControlRequest(msg) \triangleq msg.type = RICControlRequest$

$IsRICControlResponse(msg) \triangleq msg.type = RICControlResponse$

$IsRICControlFailure(msg) \triangleq msg.type = RICControlFailure$

$IsE2ConnectionUpdate(msg) \triangleq msg.type = E2ConnectionUpdate$

$IsE2ConnectionUpdateAcknowledge(msg) \triangleq msg.type = E2ConnectionUpdateAcknowledge$

$IsE2ConnectionUpdateFailure(msg) \triangleq msg.type = E2ConnectionUpdateFailure$

$IsE2NodeConfigurationUpdate(msg) \triangleq msg.type = E2NodeConfigurationUpdate$

$IsE2NodeConfigurationUpdateAcknowledge(msg) \triangleq msg.type = E2NodeConfigurationUpdateAcknowledge$

$IsE2NodeConfigurationUpdateFailure(msg) \triangleq msg.type = E2NodeConfigurationUpdateFailure$

This section defines predicates for validating $E2AP$ message contents. The predicates provide precise documentation on the $E2AP$ message format and are used within the spec to verify that steps adhere to the $E2AP$ protocol specification.

LOCAL $ValidE2SetupRequest(msg) \triangleq$

$\land$ $\land$ "transactionId" $\in$ DOMAIN msg
  $\land$ $msg[$"transactionId"$] \in Nat$

$\land$ $\land$ "globalE2NodeId" $\in$ DOMAIN msg

$\land\ msg[\text{“globalE2NodeId”}] \in Nat$

LOCAL $ValidE2SetupResponse(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$
$\land\ \land$ “globalRicId” $\in$ DOMAIN msg
$\land\ msg[\text{“globalRicId”}] \in Nat$

LOCAL $ValidE2SetupFailure(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$
$\land\ \land$ “cause” $\in$ DOMAIN msg
$\land\ msg[\text{“cause”}] \in Cause!All$

LOCAL $ValidRICServiceUpdate(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$

LOCAL $ValidRICServiceUpdateAcknowledge(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$

LOCAL $ValidRICServiceUpdateFailure(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$
$\land\ \land$ “cause” $\in$ DOMAIN msg
$\land\ msg[\text{“cause”}] \in Cause!All$

LOCAL $ValidResetRequest(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$

LOCAL $ValidResetResponse(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$

LOCAL $ValidE2ConnectionUpdate(msg) \triangleq$
$\land\ \land$ “transactionId” $\in$ DOMAIN msg
$\land\ msg[\text{“transactionId”}] \in Nat$
$\land\ \land$ “add” $\in$ DOMAIN $msg \Rightarrow$
$\land\ IsFiniteSet(msg[\text{“add”}])$
$\land\ \forall\, a \in msg[\text{“add”}] : a \in$ STRING
$\land$ “update” $\in$ DOMAIN $msg \Rightarrow$
$\land\ IsFiniteSet(msg[\text{“update”}])$
$\land\ \forall\, a \in msg[\text{“update”}] : a \in$ STRING
$\land$ “remove” $\in$ DOMAIN $msg \Rightarrow$
$\land\ IsFiniteSet(msg[\text{“remsgove”}])$

$\land \forall\, a \in msg[\text{“remove”}] : a \in \text{STRING}$

LOCAL $ValidE2ConnectionUpdateAcknowledge(msg) \triangleq$

$\land \quad \land$ “transactionId” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“transactionId”}] \in Nat$
$\land \quad \land$ “succeeded” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“succeeded”}])$
$\quad\quad \land \forall\, a \in msg[\text{“succeeded”}] : a \in \text{STRING}$
$\quad\quad \land$ “failed” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“failed”}])$
$\quad\quad \land \forall\, a \in msg[\text{“failed”}] : a \in \text{STRING}$

LOCAL $ValidE2ConnectionUpdateFailure(msg) \triangleq$

$\land \quad \land$ “transactionId” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“transactionId”}] \in Nat$
$\land \quad \land$ “cause” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“cause”}] \in Cause!All$

LOCAL $ValidE2NodeConfigurationUpdate(msg) \triangleq$

$\land \quad \land$ “transactionId” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“transactionId”}] \in Nat$
$\land \quad \land$ “globalE2NodeId” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“globalE2NodeId”}] \in Nat$
$\land \quad \land$ “add” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“add”}])$
$\quad\quad \land$ “update” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“update”}])$
$\quad\quad \land$ “remove” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“remove”}])$

LOCAL $ValidE2NodeConfigurationUpdateAcknowledge(msg) \triangleq$

$\land \quad \land$ “transactionId” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“transactionId”}] \in Nat$
$\land \quad \land$ “add” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“add”}])$
$\quad\quad \land$ “update” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“update”}])$
$\quad\quad \land$ “remove” $\in$ DOMAIN $msg \Rightarrow$
$\quad\quad \land\, IsFiniteSet(msg[\text{“remove”}])$

LOCAL $ValidE2NodeConfigurationUpdateFailure(msg) \triangleq$

$\land \quad \land$ “transactionId” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“transactionId”}] \in Nat$
$\land \quad \land$ “cause” $\in$ DOMAIN msg
$\quad\quad \land\, msg[\text{“cause”}] \in Cause!All$

LOCAL $ValidRICSubscriptionRequest(msg) \triangleq$

$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$

LOCAL $ValidRICSubscriptionResponse(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$

LOCAL $ValidRICSubscriptionFailure(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$
$\land \quad \land$ "cause" $\in$ DOMAIN msg
$\land\ msg[$"cause"$] \in Cause!All$

LOCAL $ValidRICSubscriptionDeleteRequest(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$

LOCAL $ValidRICSubscriptionDeleteResponse(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$

LOCAL $ValidRICSubscriptionDeleteFailure(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$
$\land \quad \land$ "cause" $\in$ DOMAIN msg
$\land\ msg[$"cause"$] \in Cause!All$

LOCAL $ValidRICIndication(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$

LOCAL $ValidRICControlRequest(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$

LOCAL $ValidRICControlAcknowledge(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$

LOCAL $ValidRICControlFailure(msg) \triangleq$
$\land \quad \land$ "requestId" $\in$ DOMAIN msg
$\land\ msg[$"requestId"$] \in Nat$
$\land \quad \land$ "cause" $\in$ DOMAIN msg
$\land\ msg[$"cause"$] \in Cause!All$

This section defines operators for constructing $E2AP$ messages.

LOCAL $SetType(msg,\ type) \triangleq [msg \text{ EXCEPT } !.type = type]$

LOCAL $SetFailureCause(msg,\ cause) \triangleq [msg \text{ EXCEPT } !.cause = cause]$

$WithE2SetupRequest(msg) \triangleq$
  IF $Assert(ValidE2SetupRequest(msg),$ "Invalid E2SetupRequest")
  THEN $SetType(msg,\ E2SetupRequest)$
  ELSE Nil

$WithE2SetupResponse(msg) \triangleq$
  IF $Assert(ValidE2SetupResponse(msg),$ "Invalid E2SetupResponse")
  THEN $SetType(msg,\ E2SetupResponse)$
  ELSE Nil

$WithE2SetupFailure(msg,\ cause) \triangleq$
  IF $Assert(ValidE2SetupFailure(msg),$ "Invalid E2SetupFailure")
  THEN $SetType(msg,\ SetFailureCause(E2SetupFailure,\ cause))$
  ELSE Nil

$WithRICServiceUpdate(msg) \triangleq$
  IF $Assert(ValidRICServiceUpdate(msg),$ "Invalid RICServiceUpdate")
  THEN $SetType(msg,\ RICServiceUpdate)$
  ELSE Nil

$WithRICServiceUpdateAcknowledge(msg) \triangleq$
  IF $Assert(ValidRICServiceUpdateAcknowledge(msg),$ "Invalid RICServiceUpdateAcknowledge")
  THEN $SetType(msg,\ RICServiceUpdateAcknowledge)$
  ELSE Nil

$WithRICServiceUpdateFailure(msg,\ cause) \triangleq$
  IF $Assert(ValidRICServiceUpdateFailure(msg),$ "Invalid RICServiceUpdateFailure")
  THEN $SetType(msg,\ SetFailureCause(RICServiceUpdateFailure,\ cause))$
  ELSE Nil

$WithResetRequest(msg) \triangleq$
  IF $Assert(ValidResetRequest(msg),$ "Invalid ResetRequest")
  THEN $SetType(msg,\ ResetRequest)$
  ELSE Nil

$WithResetResponse(msg) \triangleq$
  IF $Assert(ValidResetResponse(msg),$ "Invalid ResetResponse")
  THEN $SetType(msg,\ ResetResponse)$
  ELSE Nil

$WithRICSubscriptionRequest(msg) \triangleq$
  IF $Assert(ValidRICSubscriptionRequest(msg),$ "Invalid RICSubscriptionRequest")
  THEN $SetType(msg,\ RICSubscriptionRequest)$
  ELSE Nil

$WithRICSubscriptionResponse(msg) \triangleq$
IF $Assert(ValidRICSubscriptionResponse(msg),$ "Invalid RICSubscriptionResponse")
THEN $SetType(msg, RICSubscriptionResponse)$
ELSE Nil

$WithRICSubscriptionFailure(msg, cause) \triangleq$
IF $Assert(ValidRICSubscriptionFailure(msg),$ "Invalid RICSubscriptionFailure")
THEN $SetType(msg, SetFailureCause(RICSubscriptionFailure, cause))$
ELSE Nil

$WithRICSubscriptionDeleteRequest(msg) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteRequest(msg),$ "Invalid RICSubscriptionDeleteRequest")
THEN $SetType(msg, RICSubscriptionDeleteRequest)$
ELSE Nil

$WithRICSubscriptionDeleteResponse(msg) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteResponse(msg),$ "Invalid RICSubscriptionDeleteResponse")
THEN $SetType(msg, RICSubscriptionDeleteResponse)$
ELSE Nil

$WithRICSubscriptionDeleteFailure(msg, cause) \triangleq$
IF $Assert(ValidRICSubscriptionDeleteFailure(msg),$ "Invalid RICSubscriptionDeleteFailure")
THEN $SetType(msg, SetFailureCause(RICSubscriptionDeleteFailure, cause))$
ELSE Nil

$WithRICIndication(msg) \triangleq$
IF $Assert(ValidRICIndication(msg),$ "Invalid RICIndication")
THEN $SetType(msg, RICIndication)$
ELSE Nil

$WithRICControlRequest(msg) \triangleq$
IF $Assert(ValidRICControlRequest(msg),$ "Invalid RICControlRequest")
THEN $SetType(msg, RICControlRequest)$
ELSE Nil

$WithRICControlAcknowledge(msg) \triangleq$
IF $Assert(ValidRICControlAcknowledge(msg),$ "Invalid RICControlAcknowledge")
THEN $SetType(msg, RICControlResponse)$
ELSE Nil

$WithRICControlFailure(msg, cause) \triangleq$
IF $Assert(ValidRICControlFailure(msg),$ "Invalid RICControlFailure")
THEN $SetType(msg, SetFailureCause(RICControlFailure, cause))$
ELSE Nil

$WithE2ConnectionUpdate(msg) \triangleq$
IF $Assert(ValidE2ConnectionUpdate(msg),$ "Invalid E2ConnectionUpdate")
THEN $SetType(msg, E2ConnectionUpdate)$

```
    ELSE Nil

  WithE2ConnectionUpdateAcknowledge(msg) ≜
    IF Assert(ValidE2ConnectionUpdateAcknowledge(msg), "Invalid E2ConnectionUpdateAcknowledge")
     THEN SetType(msg, E2ConnectionUpdateAcknowledge)
     ELSE Nil

  WithE2ConnectionUpdateFailure(msg, cause) ≜
    IF Assert(ValidE2ConnectionUpdateFailure(msg), "Invalid E2ConnectionUpdateFailure")
     THEN SetType(msg, SetFailureCause(E2ConnectionUpdateFailure, cause))
     ELSE Nil

  WithE2NodeConfigurationUpdate(msg) ≜
    IF Assert(ValidE2NodeConfigurationUpdate(msg), "Invalid E2NodeConfigurationUpdate")
     THEN SetType(msg, E2NodeConfigurationUpdate)
     ELSE Nil

  WithE2NodeConfigurationUpdateAcknowledge(msg) ≜
    IF Assert(ValidE2NodeConfigurationUpdateAcknowledge(msg), "Invalid E2NodeConfigurationUpdateAckn
     THEN SetType(msg, E2NodeConfigurationUpdateAcknowledge)
     ELSE Nil

  WithE2NodeConfigurationUpdateFailure(msg, cause) ≜
    IF Assert(ValidE2NodeConfigurationUpdateFailure(msg), "Invalid E2NodeConfigurationUpdateFailure")
     THEN SetType(msg, SetFailureCause(E2NodeConfigurationUpdateFailure, cause))
     ELSE Nil
```

The *Messages* module is instantiated locally to avoid access from outside the module.

```
LOCAL Messages ≜ INSTANCE Messages WITH
  E2SetupRequest ← "E2SetupRequest",
  E2SetupResponse ← "E2SetupResponse",
  E2SetupFailure ← "E2SetupFailure",
  ResetRequest ← "ResetRequest",
  ResetResponse ← "ResetResponse",
  RICSubscriptionRequest ← "RICSubscriptionRequest",
  RICSubscriptionResponse ← "RICSubscriptionResponse",
  RICSubscriptionFailure ← "RICSubscriptionFailure",
  RICSubscriptionDeleteRequest ← "RICSubscriptionDeleteRequest",
  RICSubscriptionDeleteResponse ← "RICSubscriptionDeleteResponse",
  RICSubscriptionDeleteFailure ← "RICSubscriptionDeleteFailure",
  RICIndication ← "RICIndication",
  RICControlRequest ← "RICControlRequest",
  RICControlResponse ← "RICControlResponse",
  RICControlFailure ← "RICControlFailure",
```

$RICServiceUpdate \leftarrow$ "RICServiceUpdate",
$RICServiceUpdateAcknowledge \leftarrow$ "RICServiceUpdateAcknowledge",
$RICServiceUpdateFailure \leftarrow$ "RICServiceUpdateFailure",
$E2ConnectionUpdate \leftarrow$ "E2ConnectionUpdate",
$E2ConnectionUpdateAcknowledge \leftarrow$ "E2ConnectionUpdateAcknowledge",
$E2ConnectionUpdateFailure \leftarrow$ "E2ConnectionUpdateFailure",
$E2NodeConfigurationUpdate \leftarrow$ "E2NodeConfigurationUpdate",
$E2NodeConfigurationUpdateAcknowledge \leftarrow$ "E2NodeConfigurationUpdateAcknowledge",
$E2NodeConfigurationUpdateFailure \leftarrow$ "E2NodeConfigurationUpdateFailure"

MODULE $E2Node$

The *Client* module provides operators for managing and operating on $E2AP$ client connections and specifies the message types supported for the client.

CONSTANT ID

MODULE $Send$

This module provides message type operators for the message types that can be send by the $E2AP$ client.

$E2SetupRequest(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithE2SetupResponse(msg))$

$RICServiceUpdate(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICServiceUpdate(msg))$

$ResetRequest(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithResetRequest(msg))$

$ResetResponse(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithResetResponse(msg))$

$RICSubscriptionResponse(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICSubscriptionResponse(msg))$

$RICSubscriptionFailure(conn,\ msg,\ cause) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICSubscriptionFailure(msg,\ cause))$

$RICSubscriptionDeleteResponse(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICSubscriptionDeleteResponse(msg))$

$RICSubscriptionDeleteFailure(conn,\ msg,\ cause) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICSubscriptionDeleteFailure(msg,\ cause))$

$RICIndication(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICIndication(msg))$

$RICControlAcknowledge(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICControlAcknowledge(msg))$

$RICControlFailure(conn,\ msg,\ cause) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithRICControlFailure(msg,\ cause))$

$E2ConnectionUpdate(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Send(conn,\ Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

Instantiate the $E2AP!Client!$Requests module
$Send \triangleq$ INSTANCE $Send$

MODULE *Reply*

This module provides message type operators for the message types that can be send by the $E2AP$ client.

$ResetResponse(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithResetResponse(msg))$

$RICSubscriptionResponse(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithRICSubscriptionResponse(msg))$

$RICSubscriptionFailure(conn,\ msg,\ cause) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithRICSubscriptionFailure(msg,\ cause))$

$RICSubscriptionDeleteResponse(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithRICSubscriptionDeleteResponse(msg))$

$RICSubscriptionDeleteFailure(conn,\ msg,\ cause) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithRICSubscriptionDeleteFailure(msg,\ cause))$

$RICIndication(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithRICIndication(msg))$

$RICControlAcknowledge(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithRICControlAcknowledge(msg))$

$RICControlFailure(conn,\ msg,\ cause) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithRICControlFailure(msg,\ cause))$

$E2ConnectionUpdate(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn,\ msg) \triangleq$
$\quad \land\ SCTP!Client(ID)!Reply(conn,\ Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

Instantiate the $E2AP!Client!Reply$ module
$Reply \triangleq$ INSTANCE $Reply$

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an $E2AP$ client.

$E2SetupResponse(conn,\ handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsE2SetupResponse(m)$
$\quad\quad \land\ SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land\ handler(m))$

$RICServiceUpdateAcknowledge(conn,\ handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsRICServiceUpdateAcknowledge(m)$
$\quad\quad \land\ SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land\ handler(m))$

$RICServiceUpdateFailure(conn,\ handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsRICServiceUpdateFailure(m)$
$\quad\quad \land\ SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land\ handler(m))$

$ResetRequest(conn,\ handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsResetRequest(m)$
$\quad\quad \land\ SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land\ handler(m))$

$ResetResponse(conn,\ handler(\_)) \triangleq$
$\quad SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\quad\quad \land\ Messages!IsResetResponse(m)$
$\quad\quad \land\ SCTP!Client(ID)!Receive(conn)$
$\quad\quad \land\ handler(m))$

$RICSusbcriptionRequest(conn,\ handler(\_)) \triangleq$
$SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\wedge\ Messages!IsRICSubscriptionRequest(m)$
$\wedge\ SCTP!Client(ID)!Receive(conn)$
$\wedge\ handler(m))$

$RICSubscriptionDeleteRequest(conn,\ handler(\_)) \triangleq$
$SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\wedge\ Messages!IsRICSubscriptionDeleteRequest(m)$
$\wedge\ SCTP!Client(ID)!Receive(conn)$
$\wedge\ handler(m))$

$RICControlRequest(conn,\ handler(\_)) \triangleq$
$SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\wedge\ Messages!IsRICControlRequest(m)$
$\wedge\ SCTP!Client(ID)!Receive(conn)$
$\wedge\ handler(m))$

$E2ConnectionUpdate(conn,\ handler(\_)) \triangleq$
$SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\wedge\ Messages!IsE2ConnectionUpdate(m)$
$\wedge\ SCTP!Client(ID)!Receive(conn)$
$\wedge\ handler(m))$

$E2ConnectionUpdateAcknowledge(conn,\ handler(\_)) \triangleq$
$SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\wedge\ Messages!IsE2ConnectionUpdateAcknowledge(m)$
$\wedge\ SCTP!Client(ID)!Receive(conn)$
$\wedge\ handler(m))$

$E2NodeConfigurationUpdate(conn,\ handler(\_)) \triangleq$
$SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\wedge\ Messages!IsE2NodeConfigurationUpdate(m)$
$\wedge\ SCTP!Client(ID)!Receive(conn)$
$\wedge\ handler(m))$

$E2NodeConfigurationUpdateAcknowledge(conn,\ handler(\_)) \triangleq$
$SCTP!Client(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\wedge\ Messages!IsE2NodeConfigurationUpdateAcknowledge(m)$
$\wedge\ SCTP!Client(ID)!Receive(conn)$
$\wedge\ handler(m))$

Instantiate the $E2AP!Client!$Responses module
$Handle \triangleq$ INSTANCE $Receive$

$Connect(dst) \triangleq SCTP!Client(ID)!Connect(dst)$

$Disconnect(conn) \triangleq SCTP!Client(ID)!Disconnect(conn)$

The set of all open $E2AP$ connections
$Connections \triangleq SCTP!Client(ID)!Connections$

Provides operators for the $E2AP$ client
$E2Node(ID) \triangleq$ INSTANCE $E2Node$

MODULE RIC

The *Server* module provides operators for managing and operating on $E2AP$ servers and specifies the message types supported for the server.

CONSTANT ID

MODULE $Send$

This module provides message type operators for the message types that can be send by the $E2AP$ server.

$E2SetupResponse(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2SetupResponse(msg))$

$RICServiceUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithRICServiceUpdateAcknowledge(msg))$

$RICServiceUpdateFailure(conn, msg, cause) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithRICServiceUpdateFailure(msg, cause))$

$ResetRequest(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithResetRequest(msg))$

$ResetResponse(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithResetResponse(msg))$

$E2ConnectionUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2ConnectionUpdate(msg))$

$E2ConnectionUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2ConnectionUpdateAcknowledge(msg))$

$E2NodeConfigurationUpdate(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdate(msg))$

$E2NodeConfigurationUpdateAcknowledge(conn, msg) \triangleq$
$\quad \land SCTP!Server(ID)!Send(conn, Messages!WithE2NodeConfigurationUpdateAcknowledge(msg))$

Instantiate the $E2AP!Server!Send$ module
$Send \triangleq$ INSTANCE $Send$

MODULE *Reply*

This module provides message type operators for the message types that can be send by the *E2AP* server.

E2SetupResponse(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithE2SetupResponse*(*msg*))

RICServiceUpdateAcknowledge(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithRICServiceUpdateAcknowledge*(*msg*))

RICServiceUpdateFailure(*conn*, *msg*, *cause*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithRICServiceUpdateFailure*(*msg*, *cause*))

ResetRequest(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithResetRequest*(*msg*))

ResetResponse(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithResetResponse*(*msg*))

E2ConnectionUpdate(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithE2ConnectionUpdate*(*msg*))

E2ConnectionUpdateAcknowledge(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithE2ConnectionUpdateAcknowledge*(*msg*))

E2NodeConfigurationUpdate(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithE2NodeConfigurationUpdate*(*msg*))

E2NodeConfigurationUpdateAcknowledge(*conn*, *msg*) $\triangleq$
  $\wedge$ *SCTP*!*Server*(*ID*)!*Reply*(*conn*, *Messages*!*WithE2NodeConfigurationUpdateAcknowledge*(*msg*))

Instantiate the *E2AP*!*Server*!*Reply* module

Reply $\triangleq$ INSTANCE *Reply*

MODULE *Receive*

This module provides predicates for the types of messages that can be received by an *E2AP* server.

E2SetupRequest(*conn*, *handler*(_)) $\triangleq$
  SCTP!*Server*(*ID*)!*Handle*(*conn*, LAMBDA *x*, *m* :
    $\wedge$ *Messages*!*IsE2SetupRequest*(*m*)
    $\wedge$ *SCTP*!*Server*(*ID*)!*Receive*(*conn*)
    $\wedge$ *handler*(*m*))

RICServiceUpdate(*conn*, *handler*(_)) $\triangleq$
  SCTP!*Server*(*ID*)!*Handle*(*conn*, LAMBDA *x*, *m* :
    $\wedge$ *Messages*!*IsRICServiceUpdate*(*m*)

```
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

ResetRequest(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsResetRequest(m)
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

ResetResponse(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsResetResponse(m)
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

RICSubscriptionResponse(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICSubscriptionResponse(m)
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

RICSubscriptionDeleteResponse(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICSubscriptionDeleteResponse(m)
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

RICControlResponse(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICControlResponse(m)
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

RICIndication(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsRICIndication(m)
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

E2ConnectionUpdate(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
        ∧ Messages!IsE2ConnectionUpdate(m)
        ∧ SCTP!Server(ID)!Receive(conn)
        ∧ handler(m))

E2ConnectionUpdateAcknowledge(conn, handler(_)) ≜
    SCTP!Server(ID)!Handle(conn, LAMBDA x, m :
```

$\land\ Messages!IsE2ConnectionUpdateAcknowledge(m)$
$\land\ SCTP!Server(ID)!Receive(conn)$
$\land\ handler(m))$

$E2NodeConfigurationUpdate(conn,\ handler(\_)) \triangleq$
$SCTP!Server(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\land\ Messages!IsE2NodeConfigurationUpdate(m)$
$\land\ SCTP!Server(ID)!Receive(conn)$
$\land\ handler(m))$

$E2NodeConfigurationUpdateAcknowledge(conn,\ handler(\_)) \triangleq$
$SCTP!Server(ID)!Handle(conn,$ LAMBDA $x,\ m :$
$\land\ Messages!IsE2NodeConfigurationUpdateAcknowledge(m)$
$\land\ SCTP!Server(ID)!Receive(conn)$
$\land\ handler(m))$

Instantiate the $E2AP!Server!$Requests module
$Handle \triangleq$ INSTANCE $Receive$

The set of all open $E2AP$ connections
$Connections \triangleq SCTP!Server(ID)!Connections$

Provides operators for the $E2AP$ server
$RIC(ID) \triangleq$ INSTANCE RIC

$Init \triangleq SCTP!Init$

$Next \triangleq SCTP!Next$

---

* Modification History
* Last modified *Tue Sep* 21 08:24:54 *PDT* 2021 by *jordanhalterman*
* Created *Mon Sep* 13 10:53:17 *PDT* 2021 by *jordanhalterman*